PERANCANGAN DAN PEMBUATAN LENGAN ROBOT *(ROBOT ARMS)* DENGAN ENAM DERAJAT KEBEBASAN

TUGAS AKHIR

Diajukan sebagai Salah Satu Syarat
untuk Memperoleh Gelar Sarjana pada
Jurusan Teknik Mesin



Disusun Oleh :

Nama : GUSTI NOOR HIDAYAT
No. Mahasiswa : 01 525 006

JURUSAN TEKNIK MESIN
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
2008

LEMBAR PENGESAHAN PEMBIMBING

PERANCANGAN DAN PEMBUATAN LENGAN ROBOT *(ROBOT ARMS)* DENGAN ENAM DERAJAT KEBEBASAN

TUGAS AKHIR

Oleh :

GUSTI NOOR HIDAYAT

01 525 006

Yogyakarta, Desember 2008

Dosen Pembimbing Tugas Akhir

AGUNG NUGROHO ADI, ST., MT.

LEMBAR PENGESAHAN PENGUJI

PERANCANGAN DAN PEMBUATAN LENGAN ROBOT *(ROBOT ARMS)*DENGAN ENAM DERAJAT KEBEBASAN

TUGAS AKHIR

Oleh :

GUSTI NOOR HIDAYAT

01 525 006

Telah Dipertahankan di Depan Sidang Penguji
sebagai Salah Satu Syarat untuk Memperoleh Gelar Sarjana
Teknik Mesin Fakultas Teknologi Industri
Universitas Islam Indonesia

Yogyakarta, Januari 2009

Tim Penguji :

Agung Nugroho Adi, ST., MT
Ketua

Muhammad Ridlwan, ST., MT.
Anggota I

Erina Primayanti, ST., M. Eng.
Anggota II

Mengetahui,
Ketua Jurusan Teknik Mesin
Fakultas Teknologi Industri
Universitas Islam Indonesia

UNIVERSITAS ISLAM INDONESIA ★ YOGYAKARTA ★ FAKULTAS TEKNOLOGI INDUSTRI

Muhammad Ridlwan, ST., MT.

HALAMAN PERSEMBAHAN

Saya hanya bisa bersyukur dan terus bersyukur kepada

ALLAH SWT

Atas semua kasih sayang-Mu

Dan

Utusan Allah Untuk alam semesta, Imam dan panutan

MUHAMMAD SAW

Atas ajaran dan contoh yang baik dalam menjalani hidup

Ku persembahkan buat

Abah (Prof. DR. Ir. Gusti Muhammad Hatta. Msc)

Mama (Ir. Violet Burhanuddin. Mp)

Dan

Ading (Gusti Noor Ramadany Saputra)

HALAMAN MOTTO

Sesuatu yang tidak dapat dikerjakan keseluruhannya, maka jangan tinggalkan keseluruhannya (Kaidah Ushul Fiqih).

Hari ini adalah hari terbaik dalam sejarah dunia, meski kemarin tampaknya hal ini tidak mungkin terjadi. (Jack Kent Cooke)

Pusatkan semua pikiran pada tugas yang tengah engkau emban. Sinar matahari tidak akan dapat membakar sesuatu kalau tidak difokuskan melalui kaca pembesar. (Alexander Graham Bell)

"Perumpamaan ilmu yang tidak diamalkan seperti harta yang tidak dinafkahkan di jalan Allah." (Abu Hurairah)

Tujuan kita adalah tetap berada dijalan yang lurus ketika orang lain melakukan kecurangan. (George Bernard Shaw)

KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

Assalamu'alaikum Wr. Wb.

Syukur Alhamdulillah kehadirat Allah SWT yang telah melimpahkan rahmat dan karunia-Nya. Shalawat serta salam semoga tercurah kepada Nabi Muhammad SAW beserta para keluarganya, sahabat dan para pengikutnya hingga akhir zaman, karena dengan segala rahmat, hidayah dan inayah-Nya Tugas Akhir dengan judul **"Perancangan dan Pembuatan Lengan Robot *(Robot Arms)* Dengan Enam Derajat Kebebasan"** dapat diselesaikan

Laporan Tugas Akhir ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Jurusan Teknik Mesin, Fakultas Teknologi Industri, Universitas Islam Indonesia

Kelancaran dan keberhasilan pembuatan Tugas Akhir ini tidak terlepas dari bimbingan serta bantuan berbagai pihak, baik secara langsung maupun tidak langsung. Pada kesempatan ini penulis mengucapkan terima kasih kepada :

1. Allah SWT, Sang Maha Pengasih dan Penyayang. Engkaulah tempat aku memohon bimbingan dan petunjuk.
2. Nabi Muhammad SAW, atas segala perjuangan dan amanahnya.
3. Keluarga tercinta. Ayahanda Prof. DR. Ir. Gusti Muhammad Hatta, Msc, Ibunda Ir. Violet B, Mp dan Adik Gusti Noor Ramadany Saputra yang senantiasa sabar. Memberikan doa dan dukungan baik moril maupun materiil, serta pengertian yang amat berharga.
4. Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
5. Bapak M. Ridlwan, ST.,MT. selaku Ketua Jurusan Teknik Mesin Universitas Islam Indonesia.
6. Bapak Agung Nugroho Adi, ST., MT., atas arahan dan kesabaran beliau selaku dosen pembimbing tugas akhir.

7. Bapak, Ibu dosen dan staf jurusan Teknik Mesin Universitas Islam Indonesia.
8. Teman-teman seperjuangan angkatan 2001 Jurusan Teknik Mesin. Uwa Dahlan, Miming, Budi jeruk, Sugeng, Didin, Inem,
9. Wimy cradel terima kasih atas pelajaran yang berharganya
10. Teman-teman yang selalu setia membantu dan selalu meluangkan waktunya untuk menyelesaikan tugas akhir ini. Awe"eksekutor", Teguh "3gp director", Dwi, Trie, Untung, Sigit, Togel, Boy, Egit, Vigor, Billy, Amet Lombok, Ucup dan semua asisten Lab Konversi Energi.
11. Teman-teman kos "Ayub Mustofa". Fazrin, Susy Similikiti+Adit, Aries, Adrian, Angga, Ape, Anang, Wawan, Yudha, Vidy, Dian, Wega, mas Erwin, Dony, Bengkok, Ridho, Andre, bang Jay, Zikri, Tanto, Ivan, Reza. Terimakasih atas kesabaran, pengertian dan motifasinya.
12. Semua pihak yang tidak dapat kami sebutkan satu-persatu, atas doa dan bantuannya dalam penyusunan tugas akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari sempurna. Oleh karena itu, saran dan kritik yang membangun dari rekan-rekan mahasiswa, dosen dan berbagai pihak sangat diharapkan. Semoga Tugas Akhir ini dapat berguna bagi kita semua, amin.

Wassalamu'alaikum Wr. Wb.

Yogyakarta, 17 Desember 2008

Penyusun

Gusti Noor Hidayat

Abstraksi

Robot memerlukan sistem penggerak untuk dapat memindahkan suatu obyek kesuatu tempat. Lengan robot mempunyai kelebihan dibandingkan dengan sistem lain karena kemampuannya menggantikan beberapa fungsi dari lengan misalnya untuk mengambil, memindahkan dan meletakan suatu objek pada suatu tempat dan gerakan yang dialami oleh robot lebih halus (sedikit getaran). Tujuan perancangan ini adalah untuk membuat lengan robot yang dapat bergerak dengan 6 derajat kebebasan.

Tugas akhir ini merupakan penelitian awal, sebelum pengembangan robot lebih lanjut, untuk mengetahui berbagai kemungkinan yang bisa dimanfaatkan dan diaplikasikan dalam system. Dari hasil perancangan lengan robot (Robot arms) dengan 6 derajat kebebasan dihasilkan mekanik robot yang fungsinya hampir sama pada lengan manusia.

Laporan ini menjelaskan mengenai perancangan lengan robot (robot arms) dengan enam derajat kebebasan. Dengan menggunakan Mikrokontroler ATMega 32 sebagai kontrol dan motor servo sebagai penggerak lengan robot dirancang agar mampu berjalan stabil. Pengembangan yang lebih lanjut guna kesempurnaan robot sangat diharapkan agar gerak (motion) robot dapat lebih halus dan dinamis. Perancangan yang dilakukan sekarang hanya menitikberatkan pada sistem mekanik yang sangat berpengaruh pada kinerja robot.

Kata kunci : ***Lengan robot, Mikrokontroler ATMega 32, Motor servo***

DAFTAR ISI

Halaman Judul TA i
Lembar Pengesahan Dosen Pembimbing ii
Lembar Pengesahan Dosen Penguji iii
Halaman Persembahan iv
Halaman Motto v
Kata Pengantar vi
Abstraksi viii
Daftar Isi ix
Daftar Tabel xii
Daftar Gambar xiii
BAB I PENDAHULUAN
1.1 Latar Belakang 1
1.2 Rumusan Masalah 1
1.3 Batasan Masalah 2
1.4 Tujuan Tugas Akhir 2
1.5 Manfaat Tugas Akhir 2
1.6 Sistematika Penulisan 2
BAB II LANDASAN TEORI
2.1 Robot 4
2.2 Anatomi Robot 5
2.2.1 Sistem Kontrol 6
2.2.2 Mikrokontroler 8
2.2.3 ATMega 32 8
2.2.4 Motor Servo 10
2.2.5 Komunikasi Serial 11
a. Serial Sinkron 12
b. Serial Asinkron 13

2.2.6 Konvertet Logika RS232 14
2.2.7 Borlan Delphi 7 15
2.3 Sistem Tangan 16
2.4 Mekanik Robot 17
2.5 Jenis-Jenis Robot 17
BAB III PERANCANGAN
3.1 Pendahuluan 19
3.2 Diagram Alir Proses Perancangan 19
3.3 Perancangan Robot 20
3.3.1 Mekanik 22
3.3.2 Elektronik 27
3.3.3 Perangkat Lunak 28
BAB IV ANALISA DAN PEMBAHASAN
4.1 Pendahuluan.. 31
4.2 Hasil Perancangan 31
4.3 Riwayat Perancangan 32
4.3.1 Rancangan Mekanis Lengan Robot 32
4.3.2 Rancangan Elektronik Lengan Robot 35
4.4 Analisa Perancangan Lengan Robot 36
4.4.1 Penggerak 36
1. Servo 1 36
2. Servo 2 36
3. Servo 3 40
4. Servo 4 43
5. Servo 5 46
6. Servo 6 46
4.4.2 Beban 46
4.4.3 Mekanisme Robot 46
BAB V KESIMPULAN DAN SARAN
5.1 Kesimpulan 47
5.2 Saran 47

Daftar Pustaka

Lampiran

DAFTAR TABEL

Tabel 2.1 Urutan Pin RS232 14
Tabel 2.2 Perbedaan Logika RS-232 Dengan Logika TTL 15
Tabel 4.1 Perbandingan Sudut Aktuasi Servo 2 39
Tabel 4.2 perbandingan Sudut Aktuasi Servo 3 42
Tabel 4.3 perbandingan Sudut Aktuasi Servo 4 45

DAFTAR GAMBAR

Gambar 2.1. Sistem Robot Dan Orientasi Fungsi (Pitowarno, 2006) 5
Gambar 2.2. Sistem Robot Dengan Kontroler Berbasis Processor Dan Mikrokontroler (Pitowarno, 2006) 6
Gambar 2.3. Kontrol Robot Loop Terbuka 6
Gambar 2.4. Kontrol Robot Loop Tertutup 7
Gambar 2.5. Susunan Pin Mikrokontroler ATMega 32 9
Gambar 2.6. Motor Servo 10
Gambar 2.7. Hubungan Pulsa Dan Posisi Poros Motor Servo 11
Gambar 2.8. Transmisi Simplex Dan Duplex 12
Gambar 2.9. Komunikasi Serial Sinkron 13
Gambar 2.10. Komunikasi Serial Asinkron 13
Gambar 2.11. Port DB9 Male 14
Gambar 2.12. IC Max 232 15
Gambar 2.13. Lingkup Delphi IDE 16
Gambar 2.14. Jenis-Jenis Robot Arms 18
Gambar 3.1. Robot Arms 19
Gambar 3.2. Flowchart Perancangan Robot 20
Gambar 3.3. Perancangan Lengan Robot 21
Gambar 3.4. Robot Arms Secara Utuh 22
Gambar 3.5. Servo 1 Pada Posisi 90° 23
Gambar 3.6. Servo 1 Pada Posisi 0° 23
Gambar 3.7. Servo 1 Pada Posisi 180° 23
Gambar 3.8. Gripper 24
Gambar 3.9. Posisi Gripper Meletakan Benda 24
Gambar 3.10. Sendi Dan Link Pada Lengan Robot 25
Gambar 3.11. Link Pada Pesawat Aeromodeling Dan Link Yang Dibuat Dengan Akrilic 26

Gambar 3.12. Bushing Dan Bantalan Terpasang Disetiap Sambungan Atau Sendi 26
Gambar 3.13. Diagram Elektronik Untuk Lengan Robot..... 27
Gambar 3.14. Diagram alir pemrograman PC menggunakan Borlan Delphi. 28
Gambar 3.15. Diagram alir pemrograman mikrokontroler**.....** 29
Gambar 3.16. Flowchart Gerak Lengan Robot Dengan Komunikasi Serial .. 30
Gambar 4.1. Mekanisme Lengan Robot Pada Awal Perancangan 32
Gambar 4.2. Mekanisme Lengan Robot Dengan Menggunakan PCB 33
Gambar 4.3. Beban Bertambah Akibat Penempatan Motor Servo 3..... 34
Gambar 4.4. Hasil Akhir Pembuatan Mekanisme Lengan Robot 35
Gambar 4.5. Mekanisme Sendi 2 Dan Diagram Polygon..... 37
Gambar 4.6. Mekanisme Sendi 3 Dan Diagram Polygon..... 40
Gambar 4.7. Mekanisme Sendi 4 Dan Diagram Polygon..... 43

BAB I
PENDAHULUAN

1.1 LATAR BELAKANG

Robot memerlukan sistem penggerak untuk dapat memindahkan suatu obyek kesuatu tempat. Lengan robot mempunyai kelebihan dibandingkan dengan sistem lain karena kemampuannya menggantikan beberapa fungsi dari lengan misalnya untuk mengambil, memindahkan dan meletakan suatu objek pada suatu tempat dan gerakan yang dialami oleh robot lebih halus (sedikit getaran).

Lengan robot pada dasarnya memiliki sistem penggerak yang telah ada di alam, seperti manusia atau hewan. Kelebihan lengan robot yaitu dapat menggantikan beberapa fungsi lengan manusia pada suatu kondisi dimana lengan manusia tidak mampu melakukannya. Misalnya lengan tidak mampu mengangkat dan memindahkan suatu objek di luar kemampuan lengan dikarenakan kondisi obyek yang akan dipindahkan mengandung unsur kimia atau bahan berbahaya lainnya bagi manusia misalnya berbahaya untuk kulit. Atau contoh aplikasi lainnya untuk memindahkan benda yang bisa meledak dan benda yang dalam kondisi panas tidak mampu dipindahkan oleh lengan manusia. Berdasarkan hal tersebut diatas, maka lengan robot sangat dibutuhkan untuk menggantikan beberapa fungsi lengan manusia pada situasi dan kondisi yang tidak memungkinkan dan membahayakan bagi keselamatan manusia. Laporan ini disusun untuk menjelaskan perancangan dan pembuatan lengan robot dengan enam derajat kebebasan.

1.2 RUMUSAN MASALAH

Dari latar belakang diatas maka dapat dibuat masalah sebagai berikut :

1. Bagaimana sistem mekanisme lengan robot *(robot arms)* dapat melaksanakan tugas mengangkat benda yang ditentukan
2. Bagaimana cara membuat program untuk mengontrol lengan robot tersebut agar mampu melakukan perintah sesuai dengan tugas yang diberikan.

1.3 BATASAN MASALAH

Agar tidak menyimpang dan lebih terarah pada objek pembahasan maka diperlukan batasan-batasan sebagai berikut :

1. Penelitian difokuskan pada perancangan dan pembuatan perangkat keras dari lengan robot *(Robot arms)* yang memiliki 6 derajat kebebasan.
2. Kontroler yang digunakan adalah ATMEGA32
3. Aktuator yang dipergunakan adalah motor servo standar yang dapat bergerak 180 derajat
4. Bahan rangka pembuatan lengan robot menggunakan papan PCB dan *acrilik.*
5. Beban maksimal yang dapat diangkat +/- 100 gram dan dimensi maksimal berukuran 5x5 cm.

1.4 TUJUAN TUGAS AKHIR

Tujuan perancangan ini adalah untuk membuat lengan robot yang dapat bergerak dengan 6 derajat kebebasan.

1.5 MANFAAT TUGAS AKHIR

Manfaat yang nantinya bisa diambil dari Tugas Akhir ini adalah:

a. Sebagai bahan kajian awal untuk pengembangan lebih lanjut sehingga teknologi lengan robot benar-benar dapat diaplikasikan dalam dunia nyata secara handal.
b. Dapat dijadikan sebagai model sederhana untuk kepentingan proses pembelajaran.
c. Bagi peneliti, merupakan sarana penerapan dan pengembangan ilmu yang telah diperoleh selama di perguruan tinggi.

1.6 SISTEMATIKA PENULISAN

Sistematika penulisan tugas akhir ini terdiri dari bagian pendahuluan,Bab I sampai dengan Bab V dan daftar pustaka. Bagian pendahuluan berisikan halaman judul, lembar pengesahan dosen pembimbing, lembar pengesahan dosen

penguji, halaman motto, kata pengantar, abstraksi, daftar isi, daftar tabel dan juga daftar gambar.

Bab I Pendahuluan berisikan tentang latar belakang masalah yang akan diteliti, rumusan masalah, batasan masalah, tujuan dan manfaat dari tugas akhir dan sistematika penulisan laporan tugas akhir ini.

Bab II Landasan Teori berisikan teori-teori yang melandasi penyusunan tugas akhir ini. Teori-teori ini berisikan penjelasan-penjelasan tentang proses perancangan, penentuan bahan, dan mekanik robot yang digunakan.

Bab III Perancangan berisikan penjelasan tentang proses perancangan robot mulai dari penentuan material yang digunakan, sketsa rancangan dan perhitungan.

Bab IV Hasil dan Pembahasan menjelaskan tentang apa saja menjadi kendala ketika perancangan dan pemberian solusi dari kendala yang ada. Penjabaran secara ringkas atas perancangan robot juga di bahas dalam bab ini.

Bab V Penutup berisikan kesimpulan mengenai semua uraian yang telah dijabarkan pada bab-bab sebelumnya dan juga dilengkapi dengan apa saja yang bisa dilakukan untuk penelitian berikutnya agar robot yang dibuat dapat bermanfaat bagi semuanya. Bagian akhir ini memuat daftar pustaka, yaitu sumber acuan yang digunakan dalam pelaksanaan dan pembuatan laporan tugas akhir dan lampiran yang memuat gambar rangkaian perancangan sistem.

BAB II
LANDASAN TEORI

2.1 Robot

Robot adalah sebuah alat mekanik yang dapat melakukan tugas fisik, baik menggunakan pengawasan dan kontrol manusia, ataupun menggunakan program yang telah didefinisikan terlebih dulu (kecerdasan buatan). Robot berasal dari bahasa Czech, *robota,* yang berarti pekerja, mulai menjadi populer ketika seorang penulis berbangsa Czech (Ceko), Karl Capek, membuat pertunjukan dari lakon komedi yang ditulisnya pada tahun 1921 yang berjudul RUR (Rossum's uneversal robot). Ia bercerita tantang mesin yang menyerupai manusia, tetapi mampu bekerja terus-menerus tanpa lelah. Gaung popularitas istilah robot ini kemudian memperoleh sambutan dengan diperkenalkannya robot jerman dalam film Metropolis tahun 1926 yang sempat dipamerkan dalam New York World' Fair 1939.

Menurut Fu, *et al*. (1987), penelitian dan pengembangan pertama yang berbuah produk robotik dapat dilacak mulai dari tahun 1940-an ketika Argonne National Laboratories di Oak Ridge, Amerika, memperkenalkan sebuah *robotic* yang dinamai *master-slave manipulator*. Robot ini digunakan untuk menangani material radioaktif. Kemudian produk robot komersial pertama diperkenalkan oleh Unimation Incorporated, Amerika, pada tahun 1950 an.

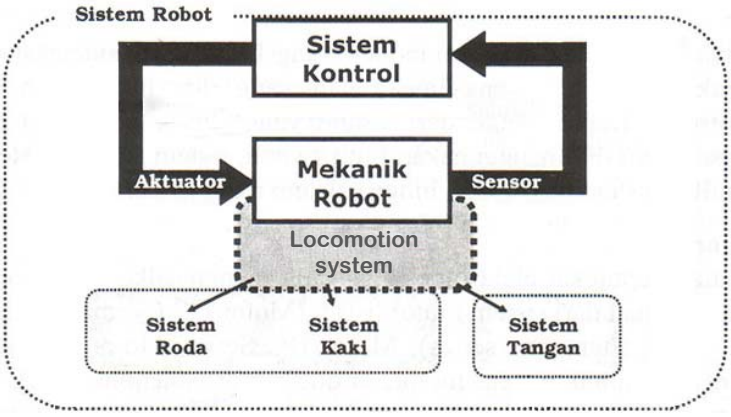
Baru setelah dunia mulai menapak ke zaman industri pada pertengahan tahun 60 an kebutuhan akan otomasi cenderung semakin meningkat. Pada saat itulah robotika diterima sebagai disiplin ilmu baru yang mendampingi ilmu-ilmu dasar dan teknik yang telah mapan sebelumnya. Dikarena keterkaitannya dengan dunia industri maka muncul istilah *industrial robot* dan robot manipulator yang sudah tak asing bagi dunia industri sekarang ini. Definisi robot industri adalah suatu robot tangan yang diciptakan untuk berbagai keperluan untuk meningkatkan produksi, memiliki langan-lengan kaku yang berhubungan secara seri dan sendi yang dapat berputar, memanjang dan memendek (*translasi*). Satu sisi lengan yang disebut sebagai pangkal ditanam pada bidang atau meja yang statis (tidak

bergerak), sedangkan sisi yang lain disebut sebagai ujung (*end of effector*) dapat dimuati dengan *tool* yang disebut dengan tugas robot. Manipulator ini memiliki dua bagian, yaitu tangan atau lengan (*arm*) dan pergelangan (*wris*t). Pada pergelangan ini dapat dipasangkan berbagai *tool.*

Dewasa ini mungkin definisi robot industri sudah tidak sesuai lagi karena teknologi *mobile robot* juga sudah dipakai meluas sejak awal 80-an. Seiring dengan itu pula kemudian muncul istilah robot *humanoid* (konstruksi mirip manusia), *animaloid* (mirip binatang), dan sebagainya. Bahkan kini dalam industri spesifik seperti industri perfilman, industri angkasa luar dan industri pertahanan atau manipulator bisa jadi hanya menjadi bagian saja dari sistem robot secara keseluruhan (Pitowarno, 2006).

2.2 Anatomi Robot

Semakin berkembangnya teknologi robot, teknik dalam desain pun perlu dibahas. Namun sebenarnya desain yang berorientasi fungsi adalah hal yang memiliki daya tarik tersendiri dan memiliki bobot untuk dibahas lebih lanjut. Fungsi disini lebih difokuskan pada teknik yang digunakan untuk menciptakan robot secara cepat, efisien, bermanfaat dan mudah dipahami dalam aplikasinya (Pitowarno, 2006). Dengan berkembangnya kemampuan, desain fungsi komersial dan fungsi lainnya dapat lebih mudah dideskripsikan. Gambar 2.1 berikut merupakan ilustrasi tentang sebuah s*istem robot* yang berhubungan dengan dunia nyata.

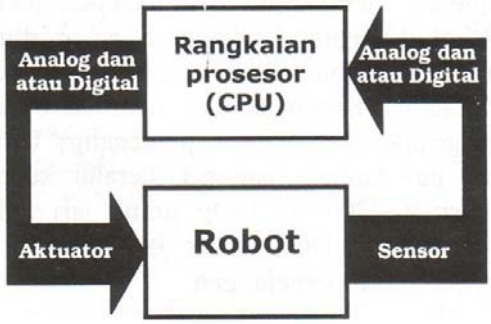


Gambar 2.1: *Sistem robot dan orientasi fungsi* (Pitowarno, 2006)

2.2.1 Sistem Kontrol

Sistem kontrol adalah kelompok komponen yang memelihara proses untuk mendapatkan hasil yang diharapkan dengan cara memanipulasi nilai-nilai variabel dalam sistem. Sistem kontrol digital menggunakan sirkuit digital sebagai kontroler berupa peranti berbasis komputer, seperti mikroprosesor, mikrokontroler, mikrokomputer, dan PLC.

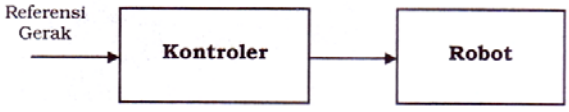
Dalam rangkaian elektronik, sistem kontrol terdiri dari rangkaian *processor* (CPU, memori, komponen *interface Input/output*), *signal conditioning* untuk sensor (analog dan atau digital), dan *driver* untuk aktuator. Bila diperlukan bisa dilengkapi dengan sistem monitor. Sistem robot yang menggunakan kontroler berbasis *processor* ataupun sistem mikrokontroler dapat dilihat pada gambar 2.2 berikut.



Gambar 2.2: *Sistem robot dengan kontroler berbasis processor dam mikrokontroler* (Pitowarno, 2006)

Kontroler berperan sebagai otak atau penerjemah dari hasil pembacaan sensor. Menurut (Pitowarno, 2006), kontrol yang dilakukan oleh sebuah kontroler pada dasarnya terbagi menjadi dua kelompok, yaitu kontrol loop terbuka (*open loop*) dan kontrol loop tertutup (*close loop*).

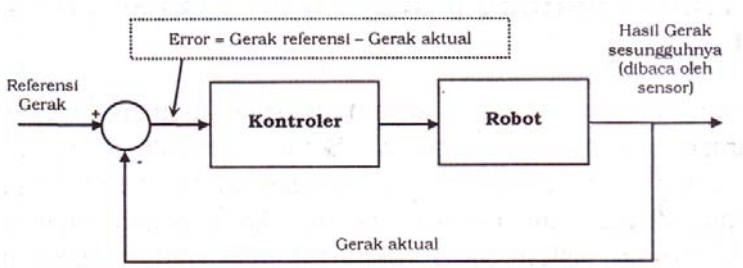
Diagram kontrol loop terbuka pada sistem robot dapat dinyatakan dalam gambar 2.3 berikut ini.



Gambar 2.3: *Kontrol robot loop terbuka*

Kontrol loop terbuka atau umpan maju (*feed forward control*) dapat dinyatakan sebagai kontrol yang outputnya tidak diperhitungkan ulang oleh kontroler. Keadaan apakah robot benar-benar telah mencapai target seperti yang dikehendaki sesuai referensi, adalah tidak dapat mempengaruhi kinerja dari sebuah kontroler. Kontrol ini sesuai untuk sistem operasi robot yang memiliki aktuator yang beroperasi berdasarkan umpan logika berbasis konfigurasi langkah sesuai urutan, misalnya motor step tidak membutuhkan sensor untuk mengetahui posisi akhir pada porosnya. Jika dalam keadaan berfungsi dengan baik dan tidak ada masalah beban lebih maka motor step akan berputar sesuai dengan perintah kontroler dan akan mencapai posisi target dengan tepat.

Kontrol loop tertutup dapat dinyatakan seperti dalam gambar 2.4



Gambar 2.4: *Kontrol robot loop tertutup*

Pada gambar di atas, jika hasil gerak aktual telah sama dengan referensi maka input kontroler akan nol. Artinya kontroler tidak lagi memberikan sinyal aktuasi kepada robot karena target akhir perintah gerak telah diperoleh. Makin kecil error terhitung maka makin kecil pula sinyal pengemudian kontroler terhadap robot, sampai akhirnya mencapai kondisi tenang (*steady state*).

Referensi gerak dan gerak aktual dapat berupa posisi (biasanya didefinisikan melalui kedudukan ujung lengan terakhir atau *end of effector*), kecepatan, akselerasi atau gabungan diantaranya. Kontrol bersifat konvergen jika dalam rentang waktu pengontrolan nilai *error* menuju nol, dan keadaan dikatakan stabil jika setelah konvergen kontroler mampu menjaga agar *error* selalu nol. Dua pengertian dasar, konvergen dan stabil, adalah sangat penting dalam *control loop* tertutup. Stabil dan konvergen diukur dari sifat referensinya. Posisi akhir dianggap

konvergen bila makin lama gerakan makin perlahan dan akhirnya diam sesuai posisi yang dikehendaki dalam referensi, dan dikatakan stabil jika posisi akhir yang diam dapat dipertahankan dalam masa-masa berikutnya.

2.2.2 Mikrokontroler

Mikrokontroler adalah IC tunggal yang berisi sirkuit khusus yang digunakan dalam desain sistem mekatronika untuk suatu fungsi tertentu. Mikrokontroler berbeda dari mikroprosesor serba guna yang digunakan dalam sebuah PC, karena sebuah mikrokontroler umumnya telah berisi komponen pendukung sistem minimal mikroprosesor, yakni memori dan antarmuka I/O (Setiawan, 2006). Mikrokontroler populer yang pertama dibuat oleh Intel pada tahun 1976, yaitu mikrokontroler 8-bit Intel 8748. Mikrokontroler tersebut adalah bagian dari keluarga mikrokontroler MCS-48. Sebelumnya, Texas Instrument telah memasarkan mikrokontroler 4-bit pertama yaitu TMS 1000 pada tahun 1974. TMS 1000 yang mulai dibuat sejak 1971 adalah mikrokomputer dalam sebuah chip, lengkap dengan RAM dan ROM.

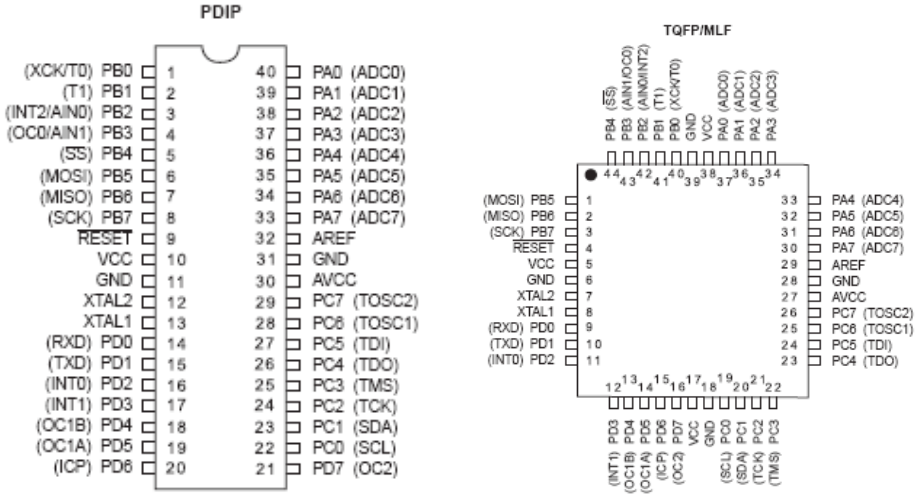
Sebuah sistem dengan mikrokontroler biasa disebut dengan *embedded system* atau *dedicated sytem.* Mikrokontroler dipasarkan dalam berbagai jenis dan ukuran dengan harga yang relatif murah. Selain mudah didapat, pemrogramannya pun lebih mudah tergantung jenis pemahaman yang dikuasai seorang programer. Mikrokontroler yang digunakan pada perancangan ini adalah mikrokontroler ATMEGA 32 yang pemrogramannya menggunakan bahasa tingkat tinggi yaitu Basic compiler.

2.2.3 ATMEGA 32

Mikrokontroler ATMega32 adalah salah satu jenis dari mikrokontroler keluarga AVR 8 bit yang diproduksi oleh Atmel (www.atmel.com). AVR memiliki arsitektur RISC (Reduced Instruction Set Computing), yaitu terdapat pemisahan memori antara kode program dan data. Instruksi pada AVR dikemas dalam kode 16 bit dan sebagian besar instruksi dieksekusi dalam 1 atau 2 clock.

Beberapa fasilitas yang terdapat pada mikrokontroler Atmega32 antara lain :

- Kapasitas memori flash 32 kB, Internal SRAM (*Static Random Access memory*) 2k byte, dan EEPROM (*Electrically Erasable Programmable Read Only Memory*) 1024 byte.
- Saluran I/O sebanyak 32 buah, yang terdiri dari 4 port yaitu Port A, Port B, Port C, dan Port D.
- 8 saluran ADC (*Analog to Digital Converter*) 10 bit.
- 2 buah pewaktu/pencacah (*timer/counter*) 8 bit dan 1 buah pewaktu/pencacah 16 bit.
- Unit interupsi internal dan eksternal.
- Port USART (*Universal Synchronous-Asynchronous Receiver Transmitter*) untuk komunikasi serial.
- Port antarmuka SPI (*Serial Peripheral Interface*).
- Komparator analog.
- 4 saluran output PWM (*Pulse Width Modulation*).
- Komunikasi serial TWI (*Two Wire Interface*).



Gambar 2.5: *susunan pin mikrokontroler ATMega32*

2.2.4 Motor Servo

Motor servo merupakan motor yang banyak dipakai pada peralatan yang dikontrol menggunakan *radio control* (RC). Motor servo ini yang bertanggung-jawab untuk membuat mobil mainan dan sebuah rangkaian mekanik dapat berbelok dan bergerak sesuai yang dikehendaki. Berbeda dengan motor DC dan motor stepper yang didisain menggunakan sistem tanpa umpan balik (*open feedback*), maka motor servo merupakan motor yang didisain bersistem dengan umpan balik (*closed feedback*).

Motor servo datang dengan banyak ukuran. Ada yang berukuran kecil yang biasanya dipakai untuk mainan (pesawat atau mobil) sampai yang besar yang dapat dipakai untuk peralatan industri. Dalam banyak hal, motor servo banyak dipakai untuk rangkaian robot. Hal yang unik dari motor servo adalah bahwa motor servo diatur/dikontrol menggunakan pulsa. Namun dengan menggunakan komputer atau rangkaian mikrokontroler, akan dapat dengan mudah mengontrol motor servo. Rangkaian paling sederhana untuk mengontrol motor servo adalah menggunakan IC 555 (Malik, 2005).



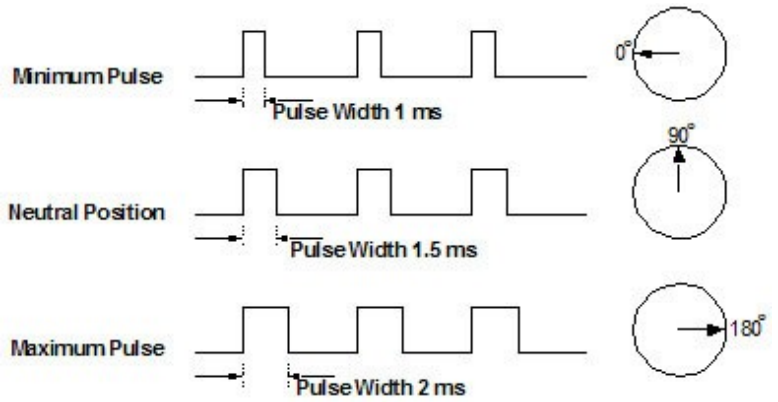
Gambar 2.6: *Motor servo*

Motor servo jika dibuka dari badannya maka akan didapat part-part sebagai berikut. Sebuah motor yang merupakan jantung dari motor servo, sekumpulan *gear-gear* yang berguna untuk mengurangi kecepatan putar motor, sebuah potensiometer dan PCB yang berisi rangkaian kontrol. Sebagaimana dijelaskan di depan bahwa motor servo termasuk jenis *closed loop*. Karena itu dipakailah potensiometer dan rangkaian

kontrol akan membentuk rangkaian umpan balik. Rangkaian kontrol dan motor tersebut didayai dengan tegangan DC sebesar 4.8 volt.

Motor servo memiliki 3 posisi utama yaitu posisi 0°, posisi 90° dan posisi 180°. Poros motor servo biasanya dihubungkan dengan suatu mekanisme sehingga dapat membuat/mengontrol pergerakan roda depan pada suatu mobil mainan. Pada saat poros motor servo pada posisi 0 derajat, maka roda depan mobil mainan akan belok ke kiri, jika posisi poros pada 90 derajat, maka roda depan mobil mainan akan lurus. Sedangkan jika posisinya 180 derajat, maka roda depan mobil mainan akan berbelok ke kanan (Malik, 2005). Karena ada 3 posisi utama seperti dijelaskan di atas, maka dibuatlah sebuah cara khusus untuk mengatur motor servo tersebut.

Cara yang digunakan adalah dengan memberikan pulsa digital dengan lebar yang berbeda-beda. Jika diberikan pulsa dengan lebar 1.5 ms maka motor servo akan berputar menuju 90 derajat (posisi tengah = *middle*). Pulsa dengan lebar 2.0 ms akan membuat poros motor servo menuju 180 derajat (posisi kanan), sedangkan pulsa dengan lebar 1.0 ms akan membuat motor servo menuju 0 derajat (posisi kiri). Pulsa tersebut dikirimkan sebanyak kira-kira 50 kali per detik / 50 Hz.

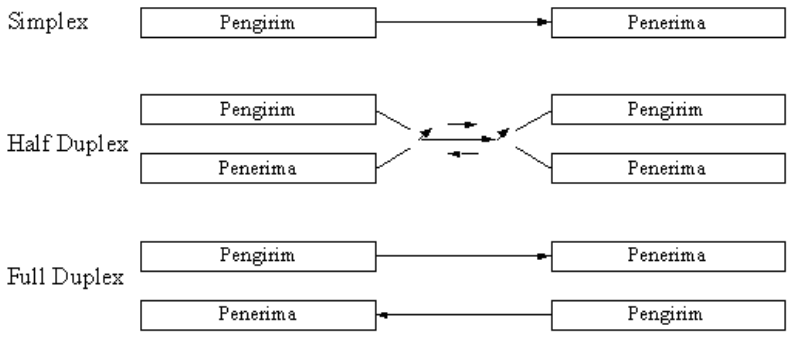


Gambar 2.7: *Hubungan pulsa dan posisi poros motor servo*

2.2.5 Komunikasi Serial

Pada dasarnya komunikasi serial merupakan pertukaran data yang dilakukan per bit sehingga lebih lambat di bandingkan komunikasi paralel yang

mampu mengirim/menerima data secara sekaligus. Namun demikian komunikasi serial mempunyai kelebihan, yaitu mampu bertukar data pada jarak yang lebih jauh dibandingkan pada komunikasi paralel.



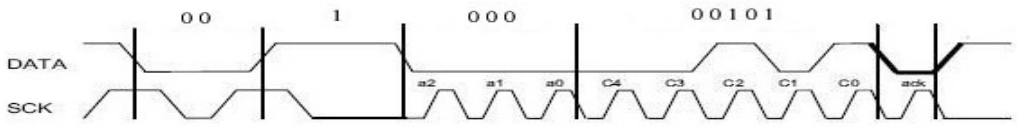
Gambar 2.8 *Transmisi simplex dan duplex*

Berdasarkan pengiriman dan penerimaan data, transmisi data dapat dibagi menjadi dua, yaitu transmisi simplex dan duplex (gambar 2.8). Pada transmisi simplex komunikasi dilakukan satu arah, satu peranti berfungsi sebagai pengirim dan peranti lainnya sebagai penerima. Pada transmisi duplex komunikasi dilakukan dua arah, pada masing-masing peranti terdapat pengirim dan penerima. Transmisi duplex dibagi menjadi dua, yaitu full duplex, dimana data dapat dikirim dan diterima secara bersamaan, serta half duplex, dimana data hanya dapat dikirim atau diterima saja pada waktu bersamaan.

Terdapat dua metode yang dapat digunakan pada komunikasi serial, yaitu sinkron dan asinkron:

a. Serial Sinkron

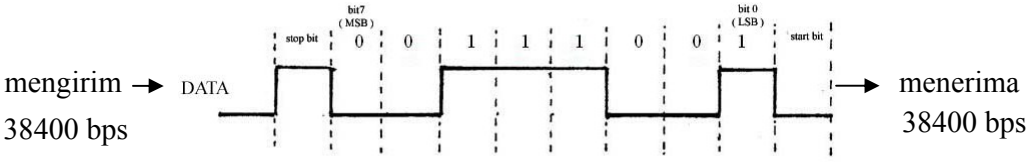
Pada komunikasi serial sinkron, selain jalur data untuk *input / output* terdapat satu jalur tambahan yang dipergunakan untuk *clock*. *Clock* berfungsi untuk mengkoordinasi data yang dikirim atau diterima. Gambar 2.9 menunjukkan contoh jalur komunikasi sinkron.



Gambar 2.9 *Komunikasi serial sinkron*

b. Serial Asinkron

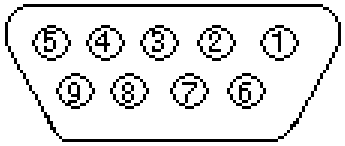
Komunikasi serial asinkron mempunyai satu buah jalur data untuk *input / output. Clock* dibangkitkan secara terpisah pada masing – masing pengirim dan penerima. Pada komunikasi asinkron piranti dapat diprogram untuk data dengan lebar data 7 atau 8 bit, sedangkan *stop bit* dapat berjumlah 1atau 2. Saat ini pengiriman data yang umum dilakukan berupa data ASCII 8 bit dan diakhiri dengan 1 stop bit.



Gambar 2.10 *Komunkasi serial asinkron*

Komunikasi serial membutuhkan port sebagai saluran data. Gambar 2.11 menunjukkan kabel serial DB9 yang umum digunakan sebagai port serial, sedangkan pada tabel 2.12 tertera urutan dan fungsi pinnya.





Gambar 2.11 *Port DB9 male*

Tabel 2.*1* *Urutan pin RS232*

9-pin	25-pin	Definisi pin
1	8	DCD (Data Carrier Detect)<PC-input>
2	3	RX (Receive Data)<PC-input>
3	2	TX (Transmit Data)<PC-output>
4	20	DTR (Data Terminal Ready)<PC-output>
5	7	GND (Signal Ground)<RefZeroVolts>
6	6	DSR (Data Set Ready)<PC-input>
7	4	RTS (Request To Send)<PC-output>
8	5	CTS (Clear To Send))<PC-input>
9	22	RI (Ring Indicator)<PC-input>

2.2.6 Konverter Logika RS232

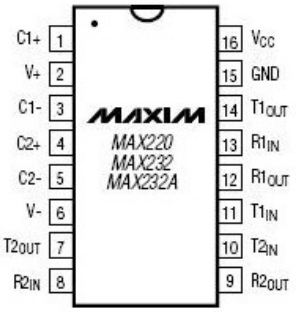
Supaya berbagai peranti dapat berkomunikasi dengan lancar, maka dibuat standar-standar komunikasi. Saat ini terdapat banyak standar komunikasi, beberapa yang populer adalah TTL dan RS-232.

RS-232 adalah standar komunikasi yang dibuat oleh *Electronics Industries Association* pada tahun 1960. Standar ini telah dipergunakan secara luas pada mini komputer, modem dan berbagai macam peralatan lainnya sejak masa sebelum PC berkembang. Karena RS-232 dikembangkan sebelum era TTL, level tegangan input dan outputnya berbeda dan tidak kompatibel dengan level tegangan TTL.

Tabel 2.2 *Perbedaan logika RS-232 dengan logika TTL*

Logika RS-232	Logika TTL
Logika 0 = +3 sampai dengan +25 volt	Logika 0 = 0 volt
Logika 1 = -3 sampai dengan -25 volt	Logika 1 = 5 volt

Untuk dapat berkomunikasi antara logika TTL dengan logika RS-232, maka dibutuhkan suatu konverter. Konverter yang paling mudah digunakan adalah MAX232. Di dalam IC ini terdapat *Charge Pump* yang akan membangkitkan +10 Volt dan -10 Volt dari sumber +5 Volt tunggal.



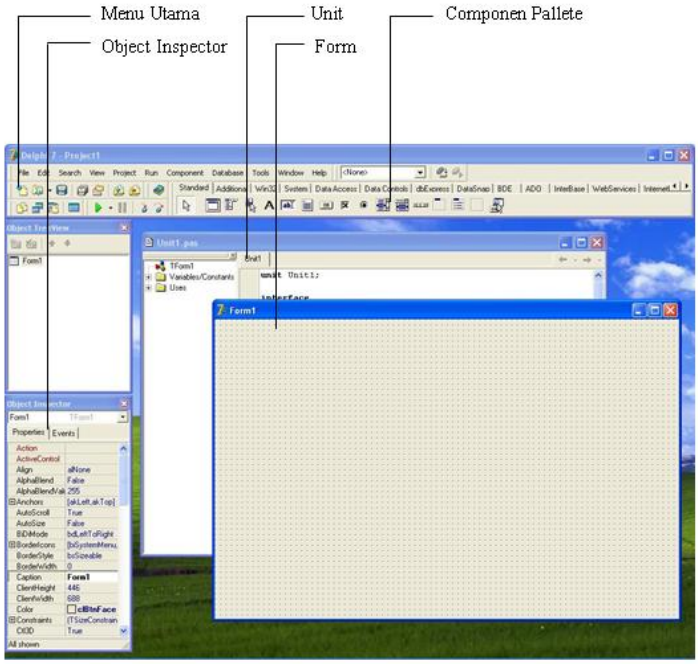
Gambar 2.12 *IC Max 232*

2.2.7 Borland Delphi 7

Borland Delphi adalah sebuah bahasa pemrograman yang dirilis dan dikembangkan oleh perusahaan software *Borland Corporation.* Delphi dirancang untuk beroperasi pada bidang operasi *Microsoft Windows.* Bahasa pengembangan yang dipergunakan adalah Tubo Pascal. Turbo Pascal mempunyai kelebihan dalam kecepatan eksekusi dan kompilasi. *Integrated Development Environtment* (IDE) yang diterapkan oleh Turbo Pascal sangat memudahkan programmer untuk merealisasikan program aplikasi mereka. Dengan IDE seorang programmer dapat dengan cepat dan mudah menulis kode program, melakukan kompilasi, melihat kesalahan *(error)* program, serta langsung menuju letak kesalahan dan

memperbaiki kesalahan tersebut. Ada beberapa keuntungan yang diperoleh dari penggunaan Delphi dalam perancangan aplikasi, antara lain :

1. Pemrograman aplikasi yang cepat, sederhana dan mudah.
2. Banyak dukungan untuk pengembangan kemampuan aplikasi dan kemampuan para programmer Delphi.
3. Delphi mempunyai kompatibilitas yang baik antara versi yang lama dengan versi terbarunya.
4. Aplikasi yang dirancang dengan Delphi sangat bervariasi, misalnya aplikasi matematis, pengolah kata, komputer grafis, multimedia dan sebagainya.
5. Jenis aplikasi yang lebih aplikatif misalnya pemrograman basis data, pemrograman web, dan aplikasi pengendalian.



Gambar 2.13 *Lingkup delphi IDE*

2.3 Sistem Tangan

Tangan adalah bagian atau anggota badan robot selain sistem roda dan kaki. Dalam *konteks mobile robot*, bagian tangan ini dikenal sebagai manipulator

yaitu sistem gerak yang berfungsi untuk memanipulasi (memegang, mengambil, mengangkat, memindah) obyek. Sedangkan pada robot industri fungsi mengolah ini dapat berupa perputaran (memasang mur baut, *drilling*, dan *milling*). Untuk robot tangan, desain sendi lengan diukur berdasarkan DOF (Pitowarno, 2006).

2.4 Mekanik Robot

Adalah sistem mekanik yang dapat terdiri dari setidak-tidaknya adalah sebuah fungsi gerak. Jumlah fungsi gerak disebut sebagai derajat kebebasan atau *Degree of Freedom* (DOF). Sebuah sendi yang diwakili oleh sebuah gerak aktuator disebut sebagai satu derajat kebebasan. Sedangkan derajat kebebasan pada struktur roda dan kaki diukur berdasarkan fungsi *holonomic* atau *non-holonomic*. Derajat kebebasan menunjukan jumlah kemungkinan gerakan pada saat yang bersamaan, mekanisme pada engsel pintu adalah contoh yang paling sederhana. Mekanisme engsel pintu memiliki jumlah derajat kebebasan satu karena gerakan yang terjadi adalah rotasi satu arah. Pada gambar 2.7 terlihat gambar *slider bearing* berikut model kinematiknya yang mempunyai satu derajat kebebasan dimana arah gerakannya adalah translasi satu arah (Hutahean, 2005).

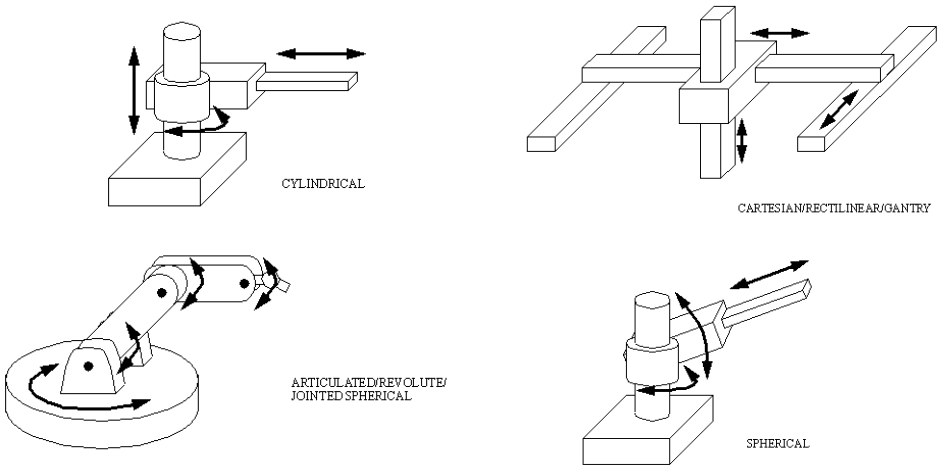
Hal mendasar yang perlu diperhatikan dalam desain mekanik robot adalah perhitungan kebutuhan torsi untuk menggerakan sendi. Motor sebagai penggerak utama (*prime-mover*) yang paling sering dipakai umumnya akan bekerja optimal (torsi dan kecepatan putar paling ideal pada putaran yang relatif tinggi yang hal ini tidak sesuai bila porosnya dihubungkan langsung dengan sendi gerak. Sebab kebanyakan gerakan yang diperlukan pada sisi anggota badan robot adalah relatif pelan namun bertenaga. Untuk itu diperlukan cara-cara transmisi daya motor (atau aktuator secara umum) secara tepat.

2.5 Jenis- jenis robot

Beberapa konsep robot lengan yang dikenal adalah:

- *Rectangular coordinates*. Robot ini hanya dapat bergerak kekanan dan kekiri serta keatas dan kebawah menggunakan sumbu x, y dan z

- *Cylindrical coordinates*. Robot ini dapat memutar sebesar θ derajat dan mampu bergerak lengannya pada sumbu x dan y
- *Spherical coordinates*. Lengan robot ini dapat memutar θ1 dan θ2 derajat, juga mampu bergerak di sumbu z
- *Revolute.* Ini robot paling hebat, mampu bergerak secara detail pada θ_1, θ_2, θ_3 derajat.

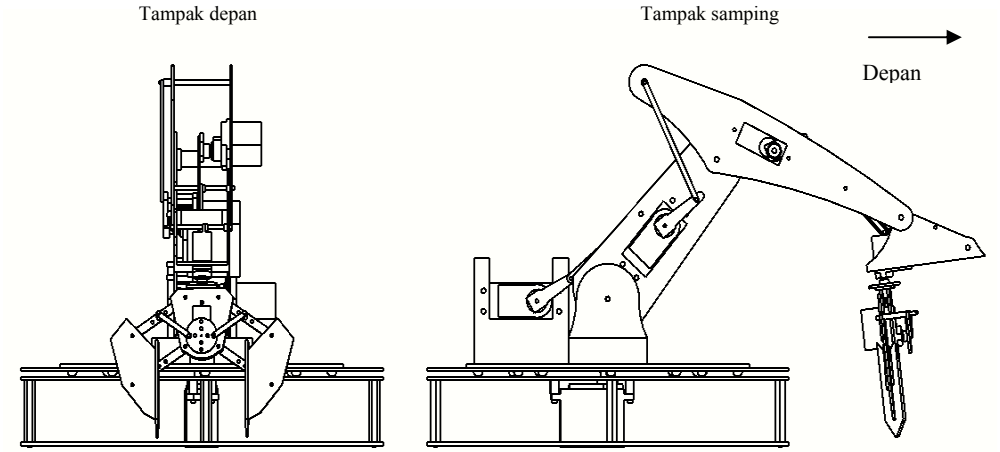


Gambar 2.14 *Jenis-jenis Robot Arms*

BAB III
PERANCANGAN

3.1 Pendahuluan

Bentuk lengan robot yang dibahas pada perancangan ini diilhami oleh bentuk tangan. Sebagian besar rangka dibuat dengan menggunakan lembaran PCB dengan tebal 2 mm yang disambung dengan sistem patri atau solder dan dibentuk sedemikian rupa menyesuaikan dimensi motor servo yang memiliki dimensi 20 x 40 mm.

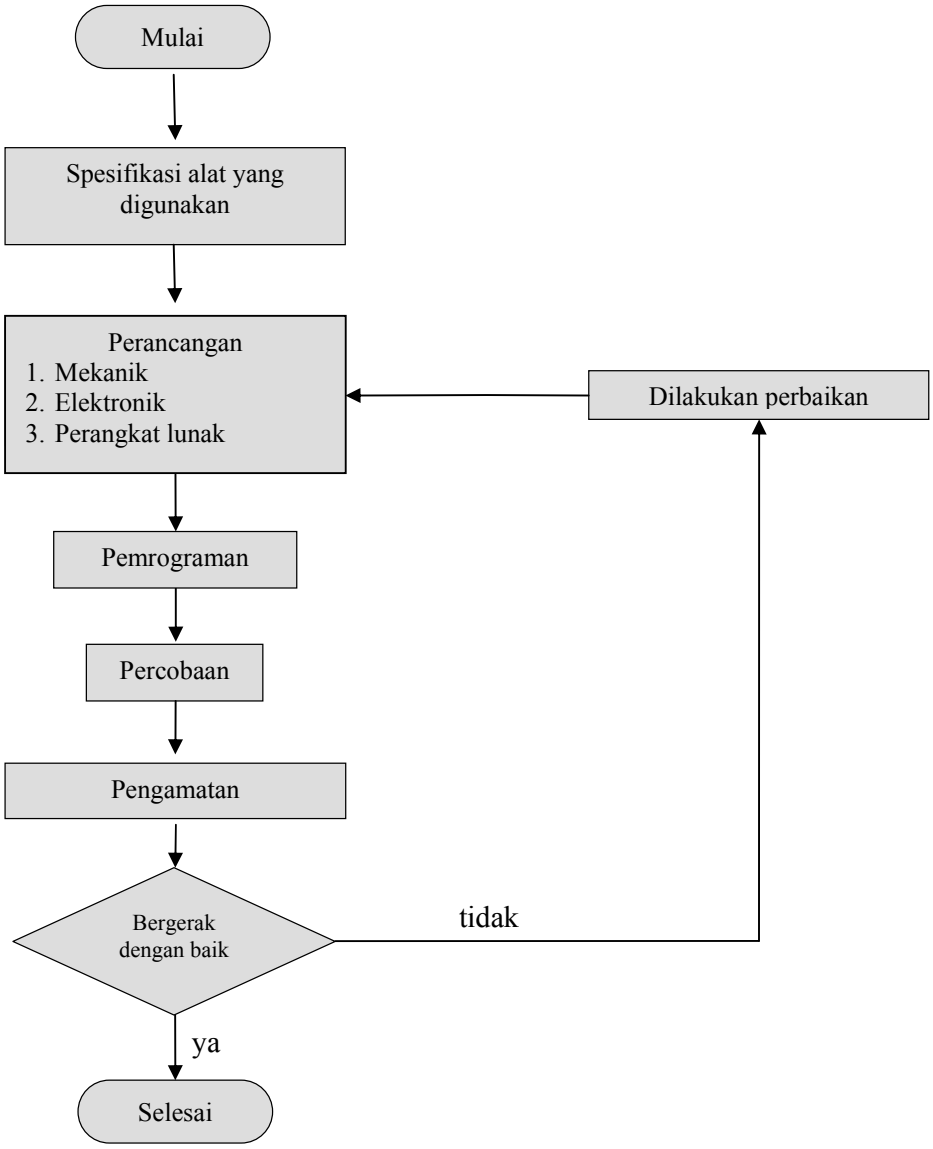


Gambar robot 3.1: *Robot arms*

Selain pertimbangan estetika lengan robot yang dibuat juga mempertimbangkan bentuk dari segi mekanis, lengan robot terdiri dari 6 buah sendi. Setiap sendinya digerakan oleh sebuah motor servo. Motor servo dikontrol oleh sebuah mikrokontroler ATMEGA 32.

3.2 Diagram Alir Proses Perancangan

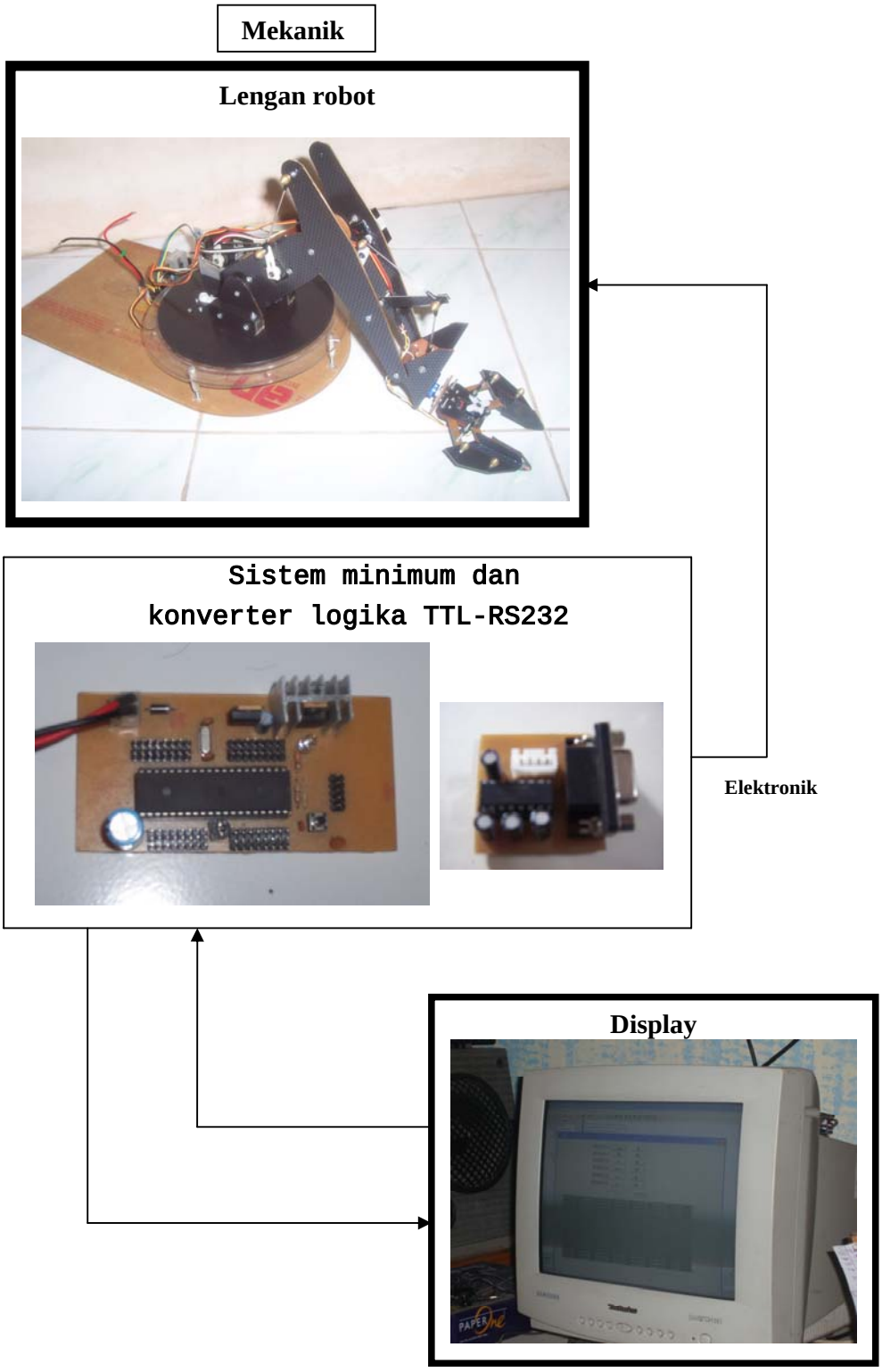
Proses perancangan alat diperlihatkan dengan diagram alir / *flowchart* pada gambar 3.



Gambar 3.2: *Flowchart Perancangan robot*

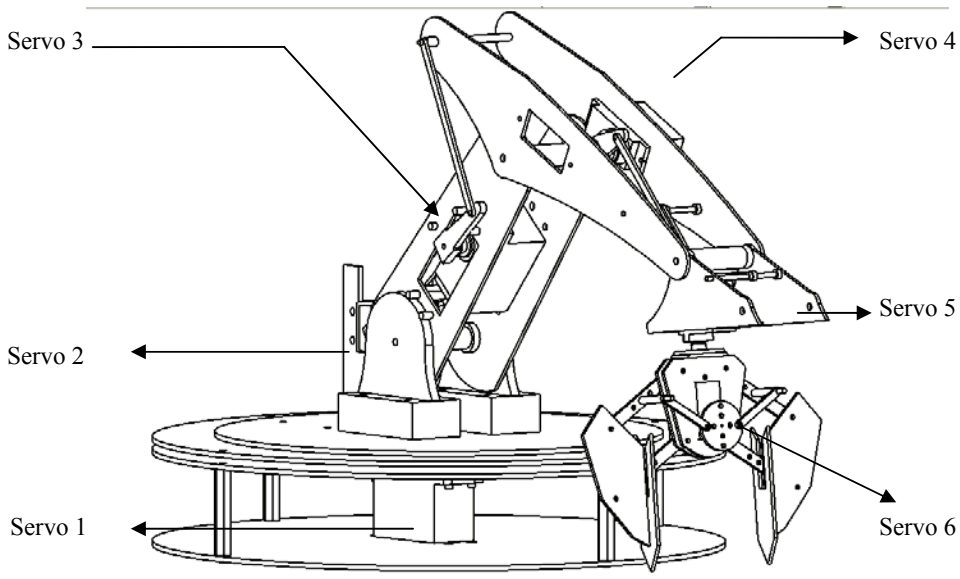
3.3. Perancangan Robot

Perancangan robot dibagi menjadi tiga bagian perancangan yaitu perancangan mekanik, perancanagan elektronik, dan perancangan perangkat lunak. Penyusunan Instrumentasi dapat dilihat pada gambar 3.3 seperti dibawah:



Gambar 3.3: *Perancangan lengan robot*

3.3.1 Mekanik



Gambar 3.4: *Robot arms secara utuh*

Struktur mekanik yang dimiliki lengan robot pada umumnya terdiri dari:

1. Manipulator

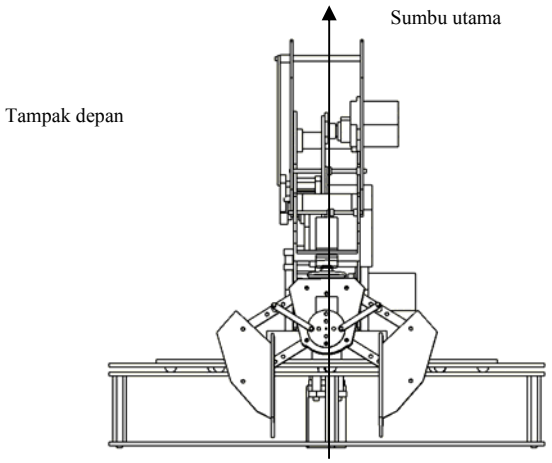
   Adalah tubuh yang membentuk robot. Bentuk-bentuk manipulator antara lain badan robot *(body),* pergelangan *(wrist)* dan lengan *(arm).*

2. *End effector*

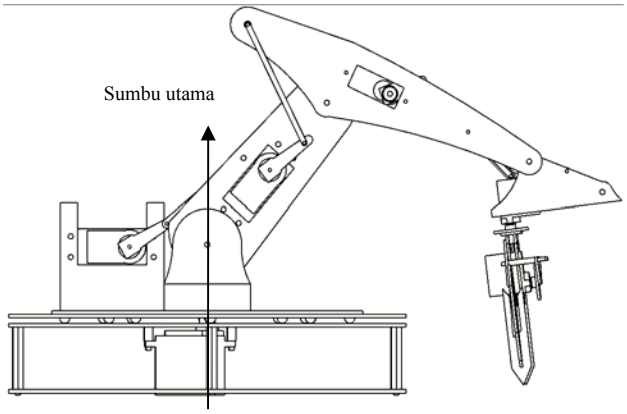
   End effector adalah ujung dari manipulator yang akan dikontrol, seperti penjepit, bor, jarum, pemanas.

Gambar 3.4 di atas merupakan bentuk akhir dari perancangan lengan robot yang mengaplikasikan 6 buah sendi. Dengan menggunakan enam buah aktuator atau penggerak motor servo, masing-masing motor servo menggerakkan setiap sendi pada lengan robot. Motor servo 1 atau servo posisi yang merupakan sumbu utama berfungsi untuk menentukan dimana robot mengambil atau meletakan benda yang akan diambil. Lengan robot pada servo 1 sebagai base atau sumbu utama kemampuan untuk mengambil dan meletakan benda memiliki jangkauan 0°

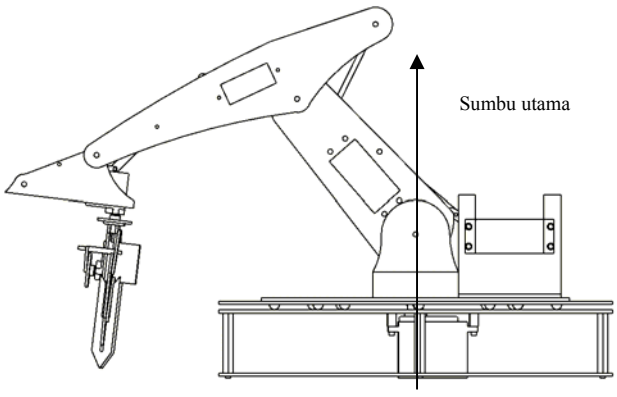
sampai 180°. Apabila robot dilihat tampak depan dan robot berada pada posisi ditengah, servo berada pada posisi 90°.



Gambar 3.5: *servo 1 pada posisi 90°*

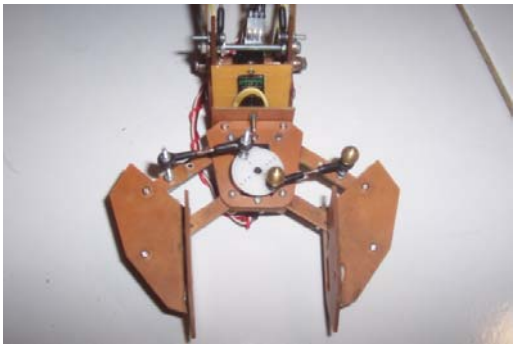


Gambar 3.6: *servo 1 pada posisi 0°*

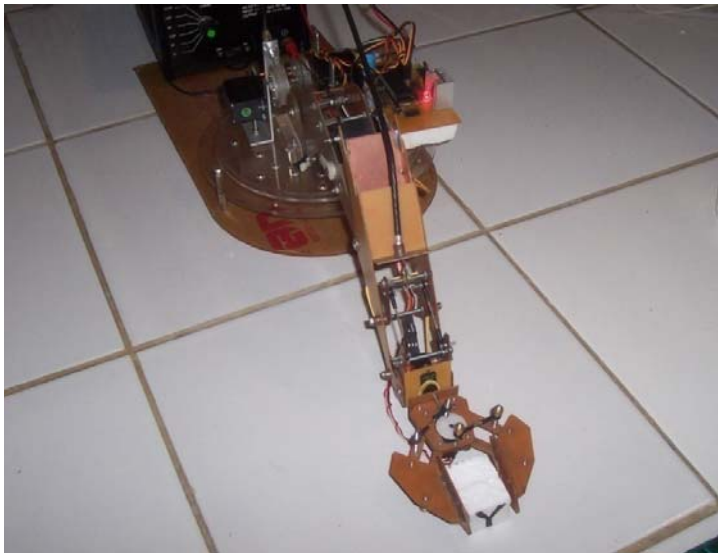


Gambar 3.7: *servo 1 pada posisi 180°*

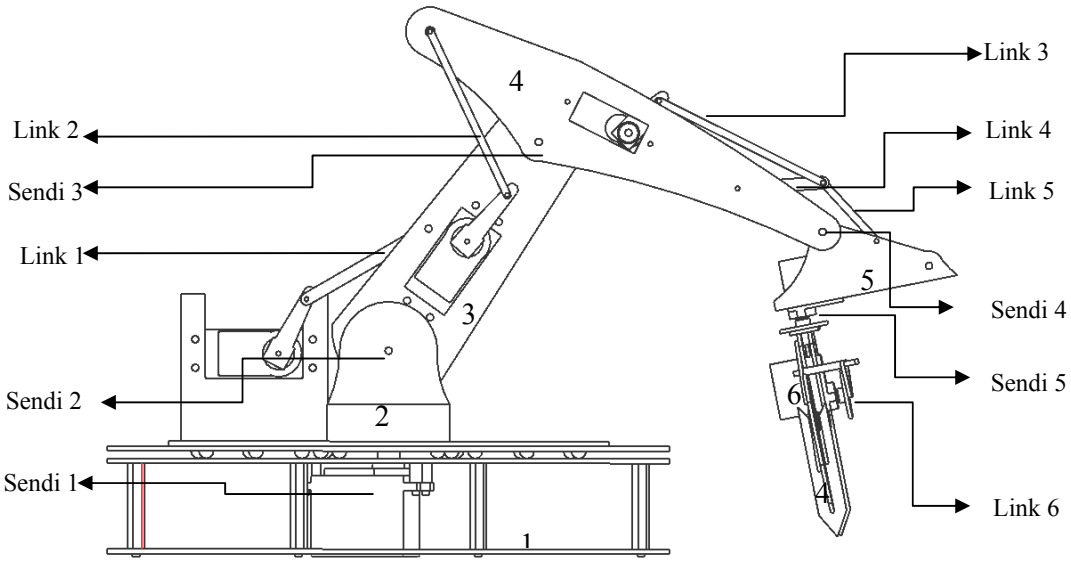
Motor servo 2 dalam mekanik lengan robot selain untuk menggerakan sendi 2, motor servo 2 juga dituntut untuk mampu mengangkat empat buah aktuator atau motor servo, tiga buah link dan gripper. Untuk itu dari keseluruhan motor servo, motor servo 2 memiliki torsi yang paling besar. Kemampuan torsi motor servo 2 memiliki torsi 9.0 kg cm pada tegangan 4.8 volt. Sedangkan motor servo 3, 4 dan 5 berfungsi sebagai penggerak sendi 3, 4 dan 5. Motor servo 6 fungsinya sebagai penggerak gripper untuk dapat mengambil dan meletakan benda dengan cara dicekam.



Gambar 3.8: *gripper*



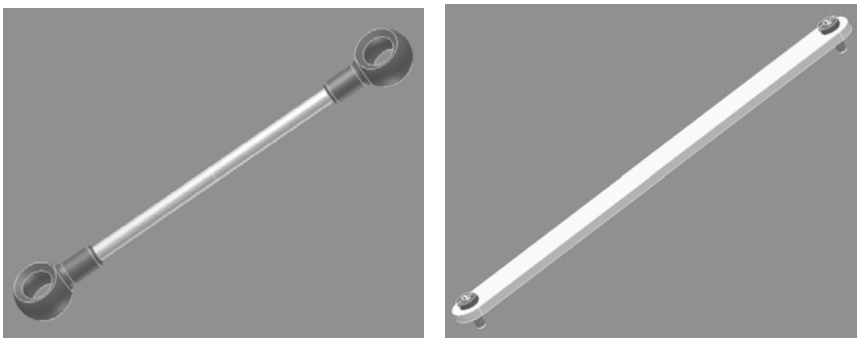
Gambar 3.9: *posisi gripper meletakan benda*



Gambar 3.10: *Sendi dan Link pada lengan robot*

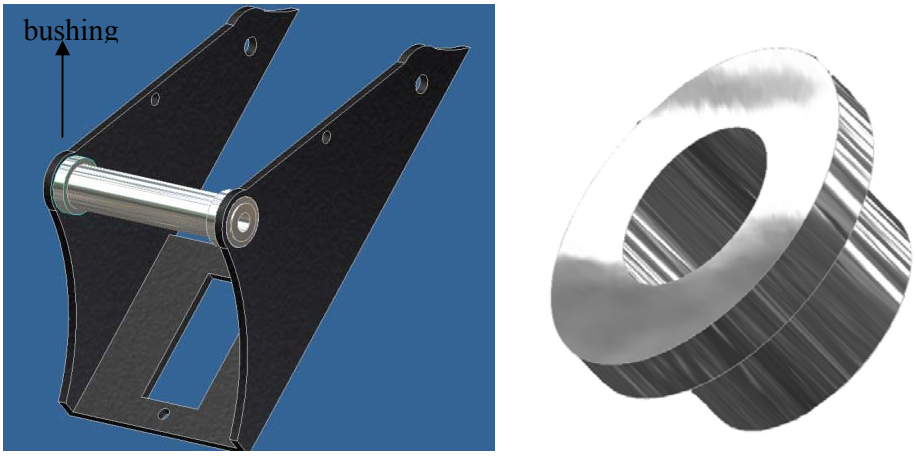
Pada gambar 3.10 terlihat bahwa ada beberapa sendi yang menggunakan *link*. Diantaranya pada sendi 2, sendi 3 dan sendi 4. Penggunaan *link* pada beberapa sendi lengan robot dimaksudkan agar torsi dari penggerak dapat lebih ditingkatkan dan memperkecil beban yang sebelumnya diterima oleh penggerak. Dan pada gripper *link* yang digunakan hanya sebagai bagian dari mekanik pada gripper.

Pada perancangan, *link* yang digunakan disini menggunakan *link* yang biasa dijumpai pada pesawat model *aeromodeling* yang memiliki kefleksibelan yang baik karena dapat bergerak secara universal atau bebas. Namun karena adanya keterbatasan *part* yang tidak memadai dan karena panjang *link* yang ada dipasaran tidak mencukupi dengan kriteria yang dibutuhkan pada robot, pemilihan akan bahan pengganti sangat dibutuhkan. Dan untuk mengantisipasi keterbatasan itu maka penggunakan akrilic yang dipotong sesuai dengan kebutuhan dapat diandalkan. Selain lebih murah dan efisien potongan akrilic juga sudah cukup kuat untuk menopang berat yang mungkin saja diterima oleh *link* itu sendiri.



Gambar 3.11: *Link pada pesawat aeromodeling dan link yang dibuat dengan akrilik*

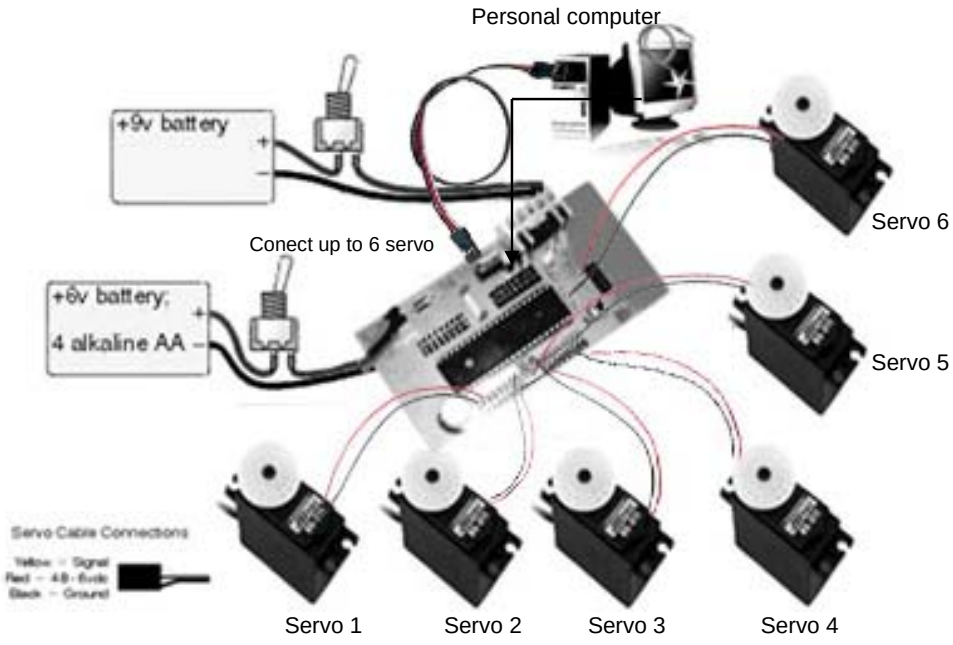
Untuk sambungan pada setiap batang hubung, *bushing* dan bantalan lebih dipercaya untuk mengurangi resiko gesekan yang bisa berakibat pada ausnya sendi dan kaki secara keseluruhan. *Bushing* dibuat menggunakan logam dari baut yang dibubut dengan ketelitian +/- 10 mikron. Namun pada kenyataannya, penggunaaan *bearing* jauh lebih baik. Karena *bearing* dengan ukuran yang relatif kecil sulit dijumpai di pasaran, penggunaan *bushing* masih cukup untuk mengantisipasi ausnya sendi.



Gambar 3.12: *Bushing dan bantalan terpasang disetiap sambungan atau sendi*

3.3.2 Elektronik

Rangkaian elektronik pada perancangan lengan robot secara sederhana dapat dilihat pada gambar 3.13.

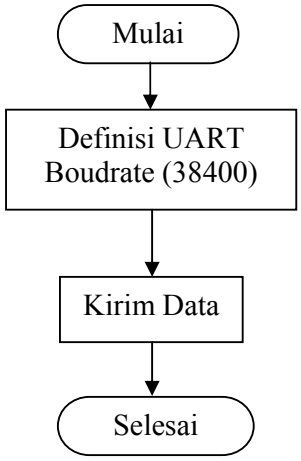


Gambar 3.13: *Diagram elektronik untuk lengan robot*

Sesuai dengan gambar diagram, mikrokontroler memberi sinyal pada 6 buah servo yang digunakan pada robot. Perancangan elektronik ini sendiri membutuhkan 2 sumber daya yang terpisah. Hal ini harus dilakukan karena sumber daya pada mikrokontroler tidak boleh terganggu. Pada saat motor servo bekerja kebutuhan akan daya cenderung berubah tergantung beban yang diterima untuk masing-masing servo. Kondisi yang seperti inilah yang biasa menyebabkan mikrokontroler reset ketika berjalan. Untuk itu pemisahan akan sumber daya harus dilakukan untuk mengantisipasi terjadinya gangguan. Port I/O pada ATMega32 yang digunakan berjumlah 32, konektor yang dipakai untuk servo hanya 6 port untuk penggunaan 6 buah motor servo. Komputer merupakan perangkat keras dalam proses pemrograman dimana penulisan dan *download* dilakukan secara serial.

3.3.3 Perangkat Lunak

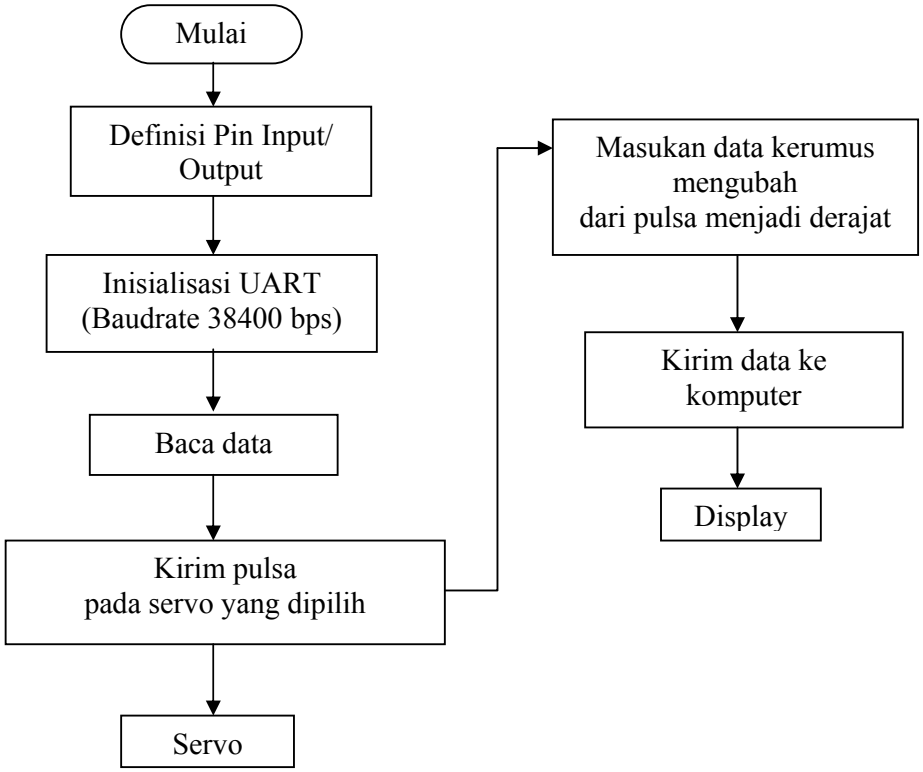
Setelah perancangan mekanik dan elektronik selesai, perancangan perangkat lunak pun harus terencana dan terorganisir. Perancangan disini dimaksudkan untuk mengetahui kestabilan robot ketika berjalan. Antarmuka (*interfacing*) merupakan suatu cara menghubungkan komputer dengan piranti lain di luar komputer (Romy dan Joseph, 2007). Dalam perancangan ini, *PC* digunakan sebagai *display* untuk menampilkan button untuk mengendalikan 6 buah motor servo dan menampilkan derajat motor servo yang bergerak. Untuk dapat menampilkan nilai maupun hasil pengukuran pada *PC*, dibutuhkan program tambahan. Program yang digunakan adalah *Borland Delphi 7.* Untuk mengakses data dari port serial harus menggunakan bahasa *asembler.* Untuk mempermudah pengaksesan data, digunakan komponen tambahan yaitu *comport.* Fungsi *comport* adalah untuk mengakses data dari port serial. Gambar 3.14 menunjukkan diagram alir pemrograman *Delphi.*



Gambar 3.14 *Diagram alir pemrograman PC menggunakan Borland Delphi*

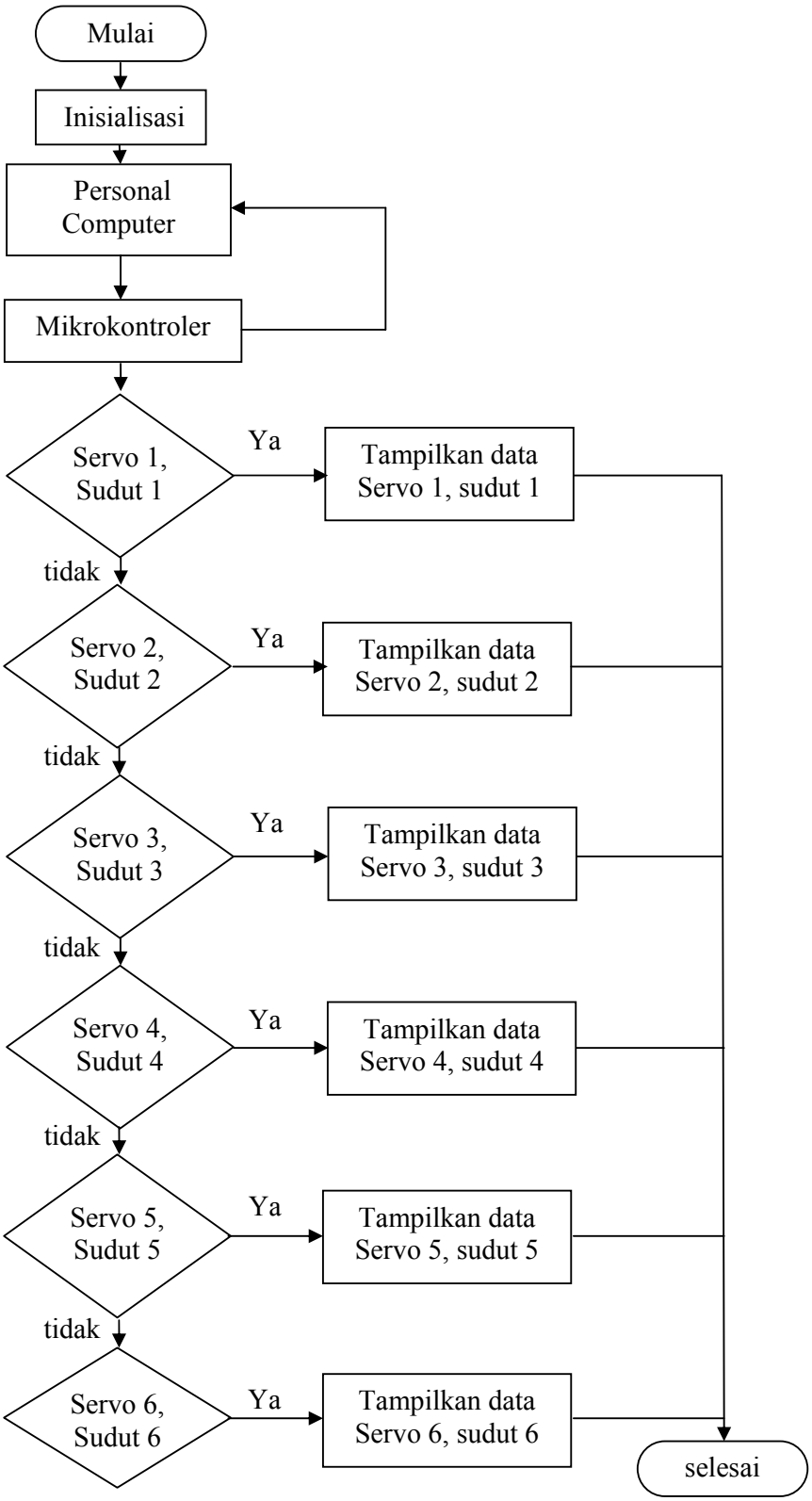
Program diawali dengan inisialisasi, inisialisasi digunakan untuk menentukan kondisi awal register yang akan digunakan pada mikrokontroler pada saat pertama kali aktif. Pada program *interfacing* lengan robot disini inisialisasi digunakan untuk menentukan *microkontroler* dan kristal yang akan digunakan, menentuknan port mana saja yang akan digunakan dan fungsinya sebagai input

atau output, timer yang digunakan sesuai dengan keperluan, penentuan kecepatan *bauidrate* yang sama dengan *baudrate* yang digunakan mikrokontroler untuk keperluan komunikasi serial, inisialisasi servo 1 sampai dengan servo 6 berada pada port keberapa dan inisialisasi keadaan awal motor servo bila akan memulai program dan inisialisasi kalibrasi dari pulsa kederajat.



Gambar 3.15 *Diagram alir pemrograman mikrokontroler*

Kemudian dilanjutkan dengan pengiriman dan penerimaan data, tampilan *display* pada *personal computer* yang menampilkan button servo 1 sampai dengan servo 6 apabila salah satunya ditekan maka akan mengirimkan perintah dalam bentuk data karakter ASCII ke mikrokontroler untuk selanjutnya dieksekusi menjalankan servo yang akan dijalankan dalam bentuk pulsa. Dan mikrokontroler akan memberikan *feedback* yang di tampilkan pada *display* untuk mengetahui berapa derajat servo yang telah digerakan. Gambar 3.14 Berikut adalah *flowchart interfacing* gerak lengan robot dengan komunikasi serial.



Gambar 3.16: *flowchart* gerak lengan robot dengan komunikasi serial

BAB IV
HASIL DAN PEMBAHASAN

4.1 Pendahuluan

Pengembangan pada perancangan lengan robot masih terus dilakukan, karena pada awal perancangan banyak kendala yang terjadi. Dikarenakan penelitian akan gerak dan bentuk yang dilakukan sudah mengalami beberapa tahap perbaikan dan penyesuaian, pemaparan tentang kendala dan tahap yang dilakukan akan dibahas dalam Bab ini. Pembahasan pun dilakukan untuk menerangkan beberapa istilah dan argumen yang pada awal atau Bab sebelumnya belum dibahas.

4.2 Hasil Perancangan

Dari hasil perancangan lengan robot *(Robot arms)* dengan 6 derajat kebebasan dihasilkan mekanik robot yang fungsinya hampir sama pada lengan manusia. Sesuai dengan judul penelitian yang diambil, robot dibuat dengan 6 derajat kebebasan. Derajat kebebasan biasa disebut dengan *degree of freedom* (DOF). DOF adalah jumlah fungsi gerak pada suatu sistem mekanik robot yang mewakili keseluruhan gerak. Pada hasil perancangan yang telah dilakukan, lengan robot dapat memindahkan suatu objek yang telah ditentukan dan lengan robot tersebut dapat dikendalikan, baik secara otomatis atau secara manual. Pengendalian secara otomatis yaitu pengendalian lengan robot dengan cara membuat program yang di awali dengan menentukan lebih dahulu kemana lengan robot akan bergerak, yaitu dengan menentukan pemberian pulsa pada masing-masing motor servo untuk menghasilkan jarak dan gerak yang diinginkan. Sedangkan pengendalian secara manual yaitu pengendalian lengan robot dengan komunikasi antar muka *(interface)* dimana *personal Computer* (PC) digunakan sebagai *display* untuk memudahkan *user* pada saat mengendalikan lengan robot sesuai dengan perintah yang diberikan. Lengan robot yang dibuat, menggunakan enam buah motor servo yang setiap sendi digerakan oleh sebuah aktuator dengan menggunakan motor servo dan mikrokontroler yang digunakan ATMega32.

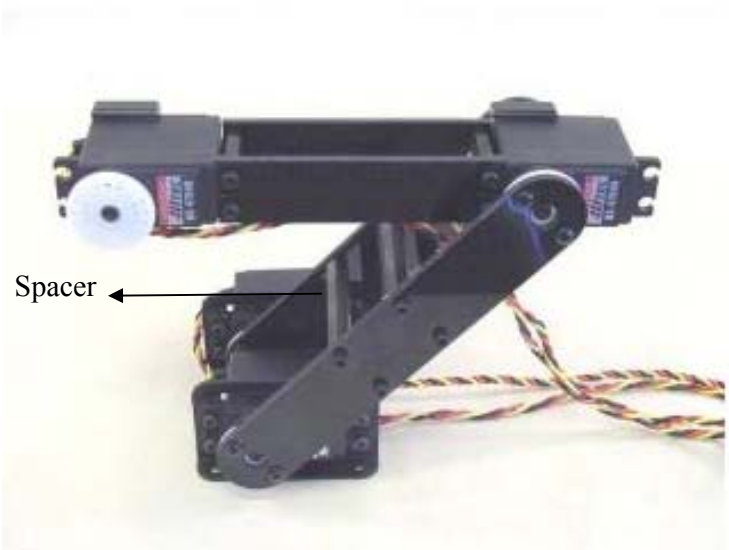
4.3 Riwayat Perancangan

Dalam perancangan, ditemui beberapa kendala sehingga perlu dilakukan beberapa koreksi dan perubahan rancangan. Beberapa koreksi tersebut adalah dalam perancangan mekanis robot, perangkat elektronik dan program pada kontroler.

4.3.1 Rancangan Mekanis Lengan Robot

Pada proses perancangan dan pembuatan mekanik lengan robot (*robot arms)* dengan 6 derajat kebebasan pengembangan yang dilakukan pada perancangan lengan robot masih terus dilakukan, karena pada awal perancangan banyak kendala yang terjadi. Dan penelitian akan gerak dan bentuk yang dilakukan sudah mengalami beberapa tahap perbaikan dan penyesuaian.

Awal perancangan mekanik lengan robot seperti ditunjukan pada gambar 4.1. Material yang digunakan untuk membuat manipulator atau badan robot menggunakan lembar akrilic dengan tebal 2 mm secara keseluruhan dan 3 mm untuk beberapa bagian yang dibentuk sesuai dengan model yang dirancang. Metode penyambungan menggunakan *spacer* yang panjangnya bervariasi tergantung kebutuhan (lihat gambar 4.1).



Gambar 4.1: *Mekanisme lengan robot pada awal perancangan*

Setelah lengan robot dibentuk, muncul beberapa kendala. Kendala utama yang terjadi yaitu masalah beban robot. Pada awal perancangan lengan robot tidak mempertimbangkan beban robot secara keseluruhan. Lengan robot dibuat dengan menggunakan material akrilic dan penggunaan *spacer* sebagai penyambung. Hal tersebut disebabkan karena keterbatasan kemampuan motor servo dan penempatan motor servo yang langsung berada pada setiap join lengan robot. Selain lebih berat, konstruksi dengan menggunakan akrilik dan *spacer* dianggap kurang praktis. Karena jika robot mengalami perbaikan, rangka haruslah dibongkar secara menyeluruh. Dan atas dasar pertimbangan inilah, rangka yang dibuat tidak menggunakan akrilik melainkan dengan menggunakan *Printing Circuit Board* (PCB) pada perancangan dan pembuatan lengan robot selanjutnya (lihat gambar 4.2).

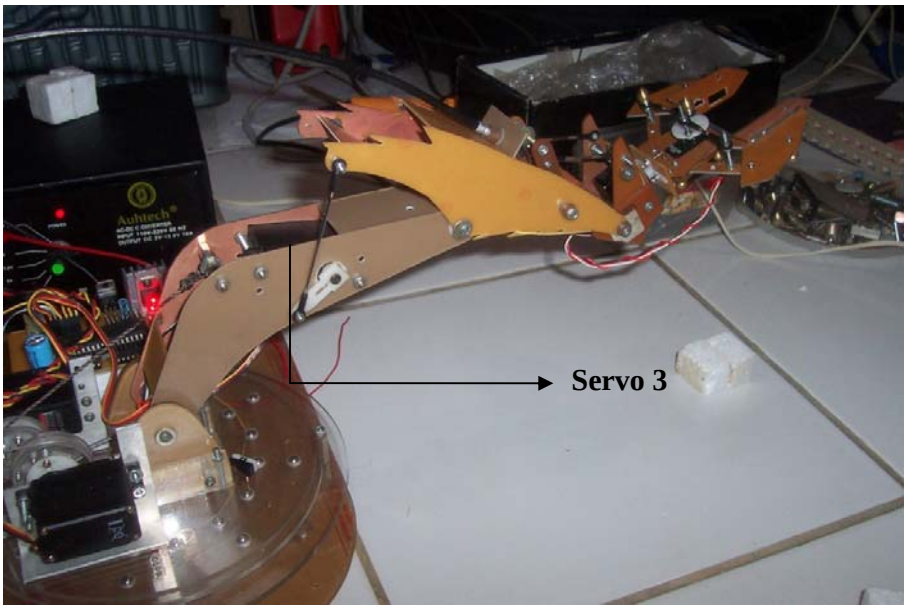


. **Gambar 4.2:** *Mekanisme lengan robot dengan menggunakan PCB*

Pada mekanisme perancangan dan pembuatan lengan robot pada tahap yang kedua ada beberapa konstruksi mekanik yang dirubah, agar dapat meningkatkan kinerja motor servo dan torsi yang dihasilkan motor servo menjadi lebih besar. Dan material yang digunakan untuk membuat badan robot dengan menggunakan PCB. PCB adalah *board* atau papan landasan yang biasa digunakan pada rangkaian elektronik untuk menempatkan komponen elektronik. Di pasaran,

papan PCB datang dengan berbagai ukuran dan jenis. Papan PCB yang digunakan disini adalah papan PCB polos *single layer* dengan tebal 2 mm yang pada salah satu bagiannya memiliki lapisan tembaga yang nantinya dapat dimanfaatkan dalam hal penyambungan. Karena memiliki lapisan tembaga, penyambungan dapat dilakukan dengan menggunakan metode patri (*soldering).* Selain jauh lebih ringan, rangka dengan PCB lebih banyak memberikan keuntungan praktis dan mekanis.

Pada perancangan dan pembuatan mekanik lengan robot yang kedua kendala yang terjadi yaitu penempatan motor servo 3 yang membuat beban pada lengan robot sedikit bertambah (lihat gambar 4.3). Walaupun demikian lengan robot dapat berjalan dengan lancar. Berbeda dengan mekanik pada perancangan awal lengan robot dimana motor servo tidak mampu sama sekali untuk bergerak karena beban dari badan robot yang terlalu berat. Hal tersebut disebabkan penempatan motor servo langsung pada setiap sendi lengan robot sehingga motor servo tidak dapat *mereduce* beban badan robot dan benda yang diangkat.



. **Gambar 4.3:** *Beban bertambah akibat penempatan motor servo 3*

Hasil akhir dari perancangan dan pembuatan lengan robot terdapat beberapa perubahan mekanik untuk memperbaiki mekanisme gerak pada lengan robot. Setelah mengalami banyak perubahan, konstruksi robot yang digunakan pada perancangan ini dianggap paling baik untuk hal gerakan yang cenderung berubah. Walaupun pengembangan yang lebih fokus masih perlu dilakukan. Material yang digunakan untuk badan robot masih menggunakan PCB mengingat selain jauh lebih ringan, rangka dengan PCB lebih banyak memberikan keuntungan praktis dan mekanis (lihat gambar 4.4).





. **Gambar 4.4:** *hasil akhir pembuatan mekanisme lengan robot*

4.3.2 Rancangan Elektronik Lengan Robot

Pada perancangan elektrik lengan robot menggunakan aktuator untuk menggerakan setiap sendi pada mekanisme lengan robot, aktuator yang digunakan yaitu motor servo. Motor servo yang digunakan memiliki torsi yang berbeda-beda, penempatan motor servo berdasarkan seberapa besar torsi yang dibutuhkan pada masing-masing sendi. Dalam perancangan elektrik lengan robot kendala yang sering terjadi yaitu pada motor servo. Motor servo sering rusak akibat pemberian beban yang melebihi kapasitas kemampuan dari motor servo itu sendiri.

4.4 Analisa Perancangan Lengan Robot

Dari hasil penelitian mengenai perancangan dan pembuatan lengan robot dengan 6 derajat kebebasan. Dalam perancangannya itu sendiri, perlu diperhatikan bentuk konstruksi robot yang dibuat dengan mempertimbangkan beberapa aspek yang saling berhubungan, diantaranya adalah penggerak, beban dan mekanisme. Hal ini dilakukan karena konstruksi sebuah robot bisa saja berbeda tergantung jenis dan karakteristik robot yang dibuat.

4.4.1 Penggerak

Penggerak merupakan sentral dari seluruh rangkaian mekanisme robot. Penggerak dituntut agar mampu menggerakkan mekanisme dengan baik. Diawal perancangannya penggerak/motor servo robot ini dihubungkan langsung ke setiap sendi. Tetapi karena torsi yang dimiliki penggerak terlampau kecil, robot tidak dapat bergerak. Dengan mengubah posisi penggerak dan menambah *link*, kendala yang telah dijelaskan sebelumnya dapat diminimalisir. Hal ini sangatlah menguntungkan karena penambahan *link* itu sendiri dimaksudkan agar torsi dari penggerak dapat lebih ditingkatkan dan memperkecil beban yang sebelumnya diterima oleh penggerak.

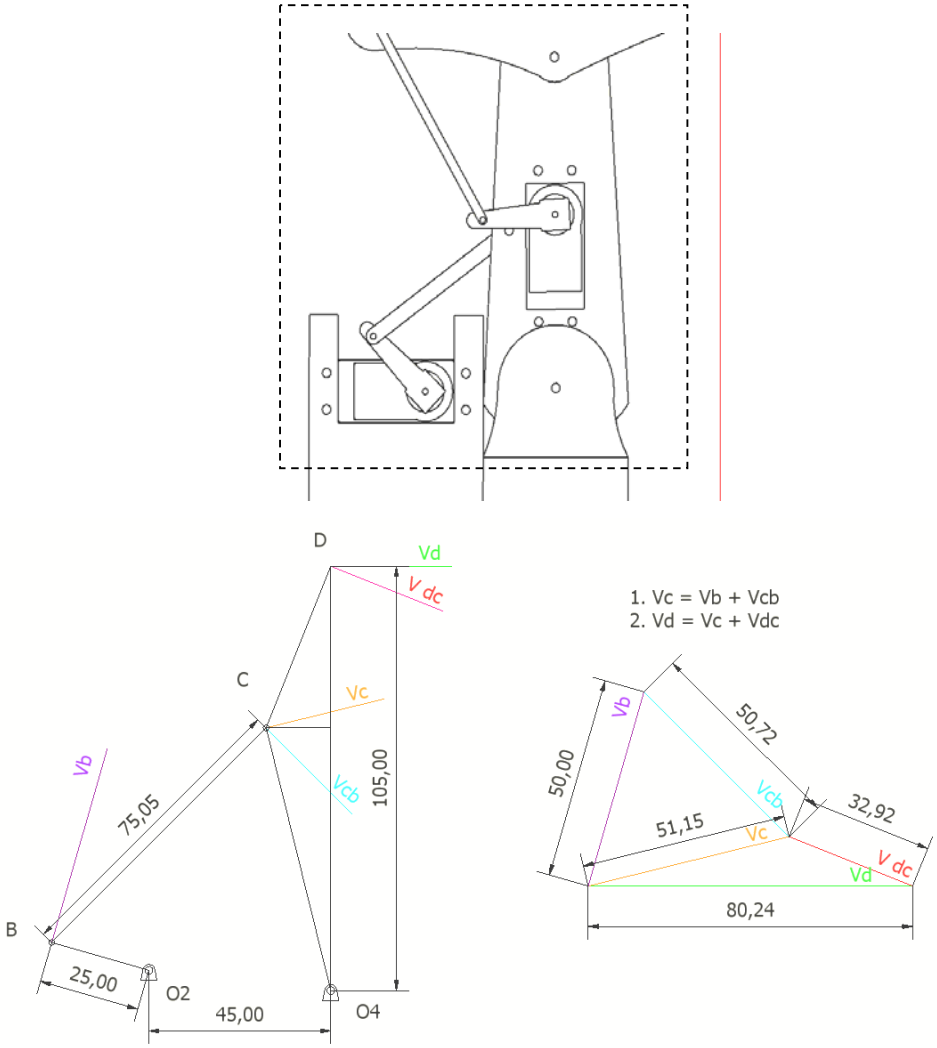
Untuk menjelaskan perbandingan besar derajat pada motor servo dan besar derajat sendi yang terbentuk pada mekanisme lengan robot, untuk memudahkan pengukuran jarak dari base ke benda yang di pindahkan. Berikut adalah perhitungan yang menjelaskan perbandingan aktuasi antara derajat pada motor servo dan derajat yang terbentuk pada setiap sendi.

1. **Servo 1**

   Pada servo 1, servo terletak pada sendi 1. Sehingga putaran sudut yang terbentuk pada servo sama dengan sudut yang terbentuk pada sendi 1. Derajat yang terbentuk pada sendi 1 yaitu 0° sampai 180°.

2. **Servo 2**

Berbeda dengan sudut 1, pada sudut 2 derajat yang terbentuk antara motor servo 2 dengan sudut 2 berbeda. Berikut adalah perhitungan sudut yang terbentuk pada sudut 2.



Gambar 4.5: *Mekanisme sendi 2 dan diagram polygon*

Dimana :

Pada data sheet motor servo 2 yang menggunakan Hitech HS-635HB

Operating speed: 0.18 sec/60°

- Waktu untuk 1° = $\frac{0.18 \text{ second}}{60°} = 0.003 \text{ second}$

Misalkan $\Delta\theta_2 = 10°$

Δt = waktu untuk 1° x $\Delta\theta_2$

= 0.003 second x 10°

= 0.03 second

- Kecepatan sudut (ω)

$$\omega_2 = \frac{\Delta\emptyset_2}{\Delta t}$$

$$= \frac{10°}{0.03 \text{ second}}$$

$$= 333.4 \text{ rad/second}$$

- Kecepatan linear (V_B)

$V_B = \omega_2 \times R_2$

$= (333.4 \text{ rad/ second}) \times (25.10^{-3} \text{ m})$

$= 8.4$ m/s

V_B pada diagram polygon = 5 cm
Jadi, skala 1: 1.7
Dalam satuan (mm) 1 : 0.17

- V_D = panjang yang terbentuk x skala
  = 80.24 mm x (0.17)
  = 13.64 m/s

- $$\omega_4 = \frac{V_D}{\text{panjang yang terbentuk}}$$

$$= \frac{13.64 \text{ m/s}}{80.24 \text{ mm}}$$

$$= \frac{13.64 \text{ m/s}}{80{,}24.10^{-3}\text{m}}$$

$= 170$ rad/second

- $\theta_4 = \omega_4 \times \Delta t$
  $= (170 \text{ rad/ s}) \times (0.03 \text{ s})$
  $= 5.1° \approx 5°$

- Sehingga dapat disimpulkan perbandingan derajat pada motor servo 2 dan sendi 2 memiliki perbandingan 1 : ½

Tabel 4.1 *Perbandingan sudut aktuasi servo 2*

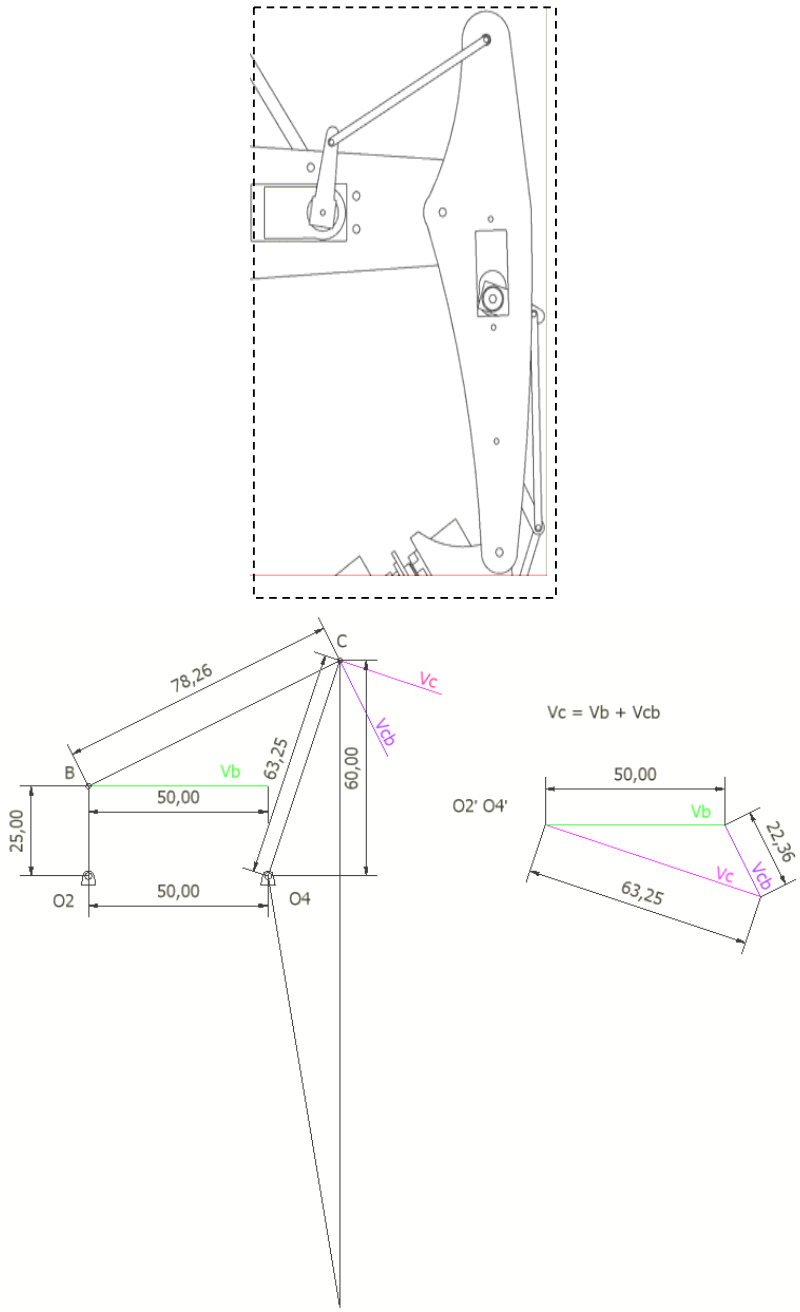
t	θ referensi	θ x	l x	skala	l O2 B	Δt	ω2	vb	vd	ω4 (mm)	ω4 (m)	θ4	ratio
0.18	60	**10**	80.24	0.17	0.025	0.03	333.3	8.333	13.641	0.17	170	**5.1**	0.5
0.18	60	**20**	80.24	0.17	0.025	0.06	333.3	8.333	13.641	0.17	170	**10.2**	0.5
0.18	60	**30**	80.24	0.17	0.025	0.09	333.3	8.333	13.641	0.17	170	**15.3**	0.5
0.18	60	**40**	80.24	0.17	0.025	0.12	333.3	8.333	13.641	0.17	170	**20.4**	0.5
0.18	60	**50**	80.24	0.17	0.025	0.15	333.3	8.333	13.641	0.17	170	**25.5**	0.5
0.18	60	**60**	80.24	0.17	0.025	0.18	333.3	8.333	13.641	0.17	170	**30.6**	0.5
0.18	60	**70**	80.24	0.17	0.025	0.21	333.3	8.333	13.641	0.17	170	**35.7**	0.5
0.18	60	**80**	80.24	0.17	0.025	0.24	333.3	8.333	13.641	0.17	170	**40.8**	0.5
0.18	60	**90**	80.24	0.17	0.025	0.27	333.3	8.333	13.641	0.17	170	**45.9**	0.5
0.18	60	**100**	80.24	0.17	0.025	0.3	333.3	8.333	13.641	0.17	170	**51**	0.5

Keterangan :
Pada data sheet

Operating speed = $\frac{0.18 \text{ second}}{60°}$

t = waktu per 60°
θ = sudut referensi (°)
θ_x = sudut yang diberikan (°)
l_x = panjang garis yang terbentuk kecepatan linear V_D (mm)
l O_2B = panjang O_2 B (m)
Δt = waktu untuk setiap sudut yang diberikan θ_x (second)
ω_2 = kecepatan sudut pada O_2 (rad/second)
V_b = kecepatan linear O_2B (m/s)
V_d = kecepatan linear O_4D (m/s)
ω_4 = kecepatan sudut pada O_4D (rad/second)
θ_4 = sudut aktuasi yang terbentuk (°)
ratio = perbandingan sudut yang diberikan (θ_x) dengan sudut aktuasi yang terbentuk (θ_4)

3. Servo 3



Gambar 4.6: *Mekanisme sendi3 dan diagram polygon*

Dimana :

Pada data sheet motor servo 2 yang menggunakan Hitech HS-311

Operating speed: 0.19 sec/60°

- Waktu untuk 1° $= \frac{0.19 \text{ second}}{60°} = 0.00317$ second

- Misalkan $\Delta \theta_2 = 10°$

  Δt = waktu untuk 1° x $\Delta\theta_2$

  = 0.00317 second x 10°

  = 0.0317 second

- Kecepatan sudut (ω)

  $\omega_2 = \frac{\Delta \emptyset_2}{\Delta t}$

  $= \frac{10°}{0.0317 \text{ second}}$

  = 315.457 rad/second

- Kecepatan linear (V_B)

  $V_B = \omega_2 \times R_2$

  = (315.457 rad/ second) x (25.10^{-3} m)

  = 7.89 m/s

  V_B pada diagram polygon = 5 cm
  Jadi, skala 1: 1.58
  Dalam satuan (mm) 1 : 0.158

- V_C = panjang yang terbentuk x skala

  = 63.25 mm x (0.158)

  = 9.976 m/s

- $\omega_4 = \frac{V_C}{\text{panjang yang terbentuk}}$

  $= \frac{9.976 \text{ m/s}}{63.25 \text{ mm}}$

$$= \frac{9.976 \text{ m/s}}{63.25.10^{-3} \text{ m}}$$

$$= 157.725 \text{ rad/second}$$

- $\theta_4 = \omega_4 \text{ x } \Delta t$
  $= (157.725 \text{ rad/ s}) \text{ x } (0.0317 \text{ s})$
  $= 4.99 ° \approx 5°$

- Sehingga dapat disimpulkan perbandingan derajat pada motor servo 3 dan sendi 3 memiliki perbandingan 1 : ½

Tabel 4.2 *Perbandingan sudut aktuasi servo 3*

t	θ refrensi	θ x	l x	skala	l O2 B	Δt	ω2	Vb	Vc	ω4 (mm)	ω4 (m)	θ4	ratio
0.19	60	**10**	63.25	0.16	0.025	0.032	315.8	7.895	9.9935	0.158	158	**5.003**	0.5
0.19	60	**20**	63.25	0.16	0.025	0.063	315.8	7.895	9.9935	0.158	158	**10.01**	0.5
0.19	60	**30**	63.25	0.16	0.025	0.095	315.8	7.895	9.9935	0.158	158	**15.01**	0.5
0.19	60	**40**	63.25	0.16	0.025	0.127	315.8	7.895	9.9935	0.158	158	**20.01**	0.5
0.19	60	**50**	63.25	0.16	0.025	0.158	315.8	7.895	9.9935	0.158	158	**25.02**	0.5
0.19	60	**60**	63.25	0.16	0.025	0.19	315.8	7.895	9.9935	0.158	158	**30.02**	0.5
0.19	60	**70**	63.25	0.16	0.025	0.222	315.8	7.895	9.9935	0.158	158	**35.02**	0.5
0.19	60	**80**	63.25	0.16	0.025	0.253	315.8	7.895	9.9935	0.158	158	**40.03**	0.5
0.19	60	**90**	63.25	0.16	0.025	0.285	315.8	7.895	9.9935	0.158	158	**45.03**	0.5
0.19	60	**100**	63.25	0.16	0.025	0.317	315.8	7.895	9.9935	0.158	158	**50.03**	0.5

Keterangan :
Pada data sheet

$$\text{Operating speed} = \frac{0.19 \text{ second}}{60°}$$

t = waktu per 60°
θ = sudut referensi (°)
θ_x = sudut yang diberikan (°)
l_x = panjang garis yang terbentuk kecepatan linear V_D (mm)
l O_2B = panjang O_2 B (m)
Δt = waktu untuk setiap sudut yang diberikan θ_x (second)
ω_2 = kecepatan sudut pada O_2 (rad/second)
V_b = kecepatan linear O_2B (m/s)
V_c = kecepatan linear O_4C (m/s)

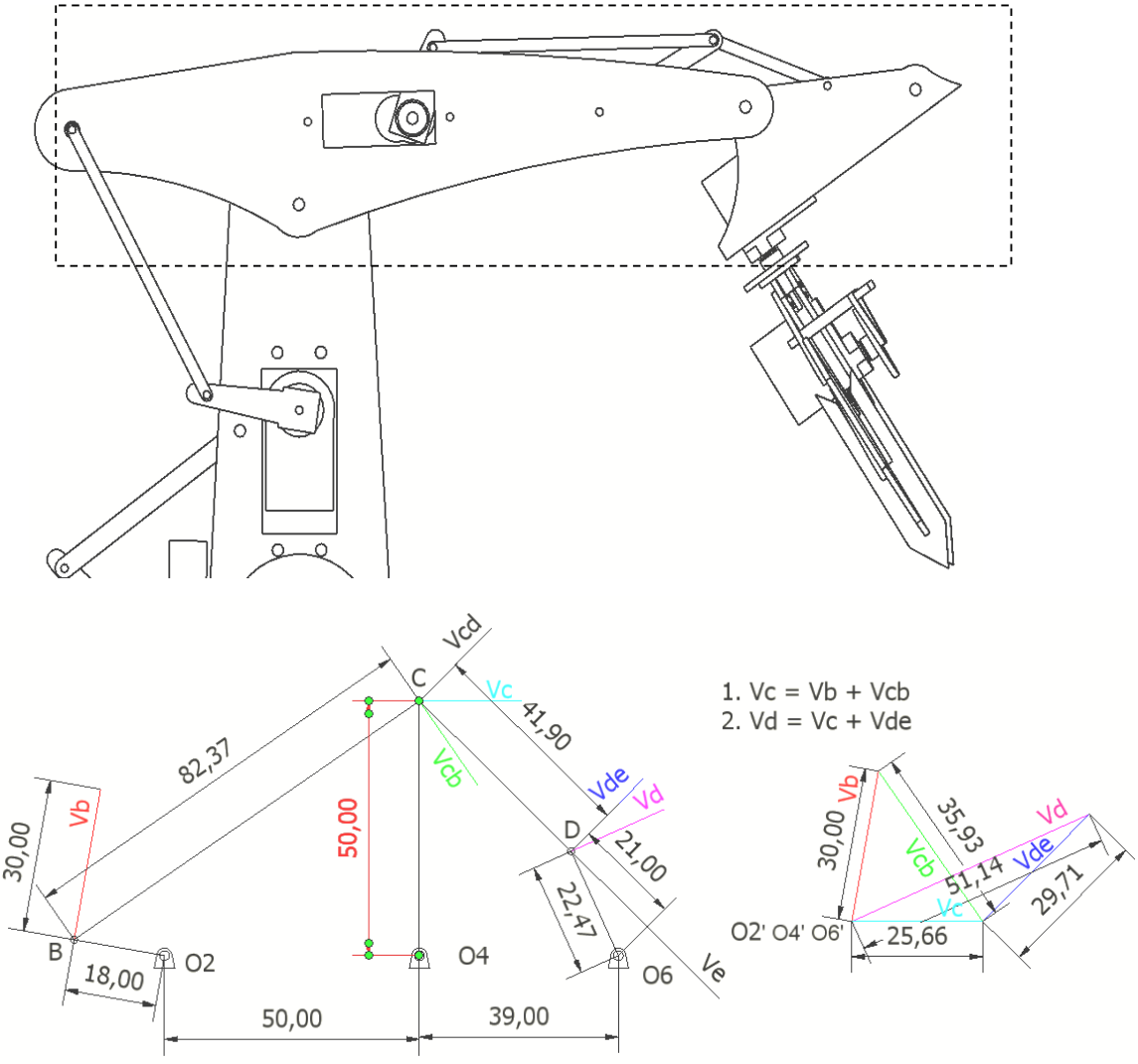
ω_4 = kecepatan sudut pada O_4D (rad/second)
θ_4 = sudut aktuasi yang terbentuk (°)
ratio = perbandingan sudut yang diberikan (θ_x) dengan sudut aktuasi yang terbentuk (θ_4)

4. Servo 4

Dimana :

Pada data sheet motor servo 4 yang menggunakan INO-LAB HG-D260MG.

Operating speed: 0.16 sec/60



Gambar 4.7: *Mekanisme sendi 4 dan diagram polygon*

- Waktu untuk 1° = $\frac{0.16 \text{ second}}{60°}$ = 0.0027 second

  Misalkan $\Delta \theta_2 = 10°$

  Δt = waktu untuk 1° x $\Delta\theta_2$

  = 0.0027 second x 10°

  = 0.027 second

- Kecepatan sudut (ω)

  $\omega_2 = \frac{\Delta \emptyset_2}{\Delta t}$

  $= \frac{10°}{0.027\ s}$

  = 375 rad/second

- Kecepatan linear (V_B)

  $V_B = \omega_2 \times R_2$

  = (375 rad/ second) x (18.10^{-3} m)

  = 6.75 m/s

  V_B pada diagram polygon = 3 cm

  Jadi, skala 1: 2.25

  Dalam satuan (mm) 1 : 0.225

- V_D = panjang yang terbentuk x skala

  = 51.14 mm x (0.225)

  = 11.5065 m/s

- $\omega_6 = \frac{V_D}{\text{Panjang yang terbentuk}}$

  $= \frac{11.5065\ m/s}{51.14\ mm}$

  $= \frac{11.5065\ m/s}{51{,}14.10^{-2} m}$

$= 225 \text{ rad/second}$

- $\theta_6 = \omega_6 \text{ x } \Delta t$
  $= (225 \text{ rad/ s}) \text{ x } (0.027 \text{ s})$
  $= 6.075° \approx 6°$

- Sehingga dapat disimpulkan perbandingan derajat pada motor servo 4 dan sendi 4 memiliki perbandingan 1 : 3/5

Tabel 4.3 *Perbandingan sudut aktuasi servo 4*

t	θ refrensi	θ x	l x	skala	l O2 B	Δt	ω2	Vb	Vd	ω6 (mm)	ω6 (m)	θ6	ratio
0.16	60	**10**	51.14	0.23	0.018	0.027	375	6.75	11.507	0.225	225	**6**	0.6
0.16	60	**20**	51.14	0.23	0.018	0.053	375	6.75	11.507	0.225	225	**12**	0.6
0.16	60	**30**	51.14	0.23	0.018	0.08	375	6.75	11.507	0.225	225	**18**	0.6
0.16	60	**40**	51.14	0.23	0.018	0.107	375	6.75	11.507	0.225	225	**24**	0.6
0.16	60	**50**	51.14	0.23	0.018	0.133	375	6.75	11.507	0.225	225	**30**	0.6
0.16	60	**60**	51.14	0.23	0.018	0.16	375	6.75	11.507	0.225	225	**36**	0.6
0.16	60	**70**	51.14	0.23	0.018	0.187	375	6.75	11.507	0.225	225	**42**	0.6
0.16	60	**80**	51.14	0.23	0.018	0.213	375	6.75	11.507	0.225	225	**48**	0.6
0.16	60	**90**	51.14	0.23	0.018	0.24	375	6.75	11.507	0.225	225	**54**	0.6
0.16	60	**100**	51.14	0.23	0.018	0.267	375	6.75	11.507	0.225	225	**60**	0.6

Keterangan :
Pada data sheet

Operating speed = $\frac{0.16 \text{ second}}{60°}$

t = waktu per 60°
θ = sudut referensi (°)
θ_x = sudut yang diberikan (°)
l_x = panjang garis yang terbentuk kecepatan linear V_D (mm)
l O_2B = panjang O_2 B (m)
Δt = waktu untuk setiap sudut yang diberikan θ_x (second)
ω_2 = kecepatan sudut pada O_2 (rad/second)
V_b = kecepatan linear O_2B (m/s)
V_d = kecepatan linear O_6D (m/s)
ω_6 = kecepatan sudut pada O_6D (rad/second)
θ_6 = sudut aktuasi yang terbentuk (°)
ratio = perbandingan sudut yang diberikan (θ_x) dengan sudut aktuasi yang terbentuk (θ_6)

5. **Servo 5**

   Pada servo 5, servo terletak pada sendi 5. Sehingga putaran sudut yang terbentuk pada servo sama dengan sudut yang terbentuk pada sendi 5. Derajat yang terbentuk pada sendi 5 yaitu 0° sampai 180°.

6. **Servo 6**

   Motor servo 6 digunakan sebagai penggerak gripper/*endeffector*. Dari derajat yang dihasilkan motor servo 6, lengan robot mampu mengankat benda dengan lebar maksimal benda 5 cm dan lebar minimal 1 cm.

4.4.2 Beban

Pada perancangan dan pembuatan lengan robot dengan 6 derajat kebebasan selain mempertimbangkan beban pada lengan robot secara keseluruhan, beban lebih difokuskan pada masing-masing sendi yang memerlukan torsi yang berdeda. Karena setiap sendi pada lengan robot memiliki pembebanan yang berbeda-beda. Beban pada manipulator/badan robot sangat berpengaruh pada kinerja setiap motor servo. Selain itu perlu diperhatikan pemilihan material yang digunakan pada perancangannya lengan robot, dimana motor servo yang tersedia memiliki kemampuan torsi yang terbatas.

4.4.3 Mekanisme Robot

Pada mekanisme lengan robot pada dasarnya memiliki sistem penggerak yang telah ada di alam, seperti manusia atau hewan. Mekanisme lengan robot yang dibuat mengadopsi bentuk lengan manusia. Perancangan dan pembuatan mekanisme lengan robot disini, robot dituntut mampu memindahkan benda yang telah ditentukan. Dalam perancangannya sendiri bentuk mekanisme lengan robot telah mengalami banyak perubahan. Walaupun pengembangan yang lebih fokus masih perlu terus untuk dilakukan.

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan hasil perancangan lengan robot yang dibuat, maka dapat disimpulkan bahwa:

1. Lengan robot dapat berjalan stabil jika kombinasi gerak pada perancangan mekanisme lengan robot dapat berjalan secara harmonis.
2. Jenis penggerak, beban material, dan konstruksi robot sangat berpengaruh terhadap gerak robot.
3. Komponen dengan berat yang berlebih akan mengganggu kestabilan gerak robot.
4. Pemisahan *power supply* untuk penggerak dan kontroler perlu dilakukan untuk mencegah terjadinya gangguan pada kontroler.

5.2 Penelitian Lanjutan

Dengan berbagai macam pertimbangan, saran dan perbaikan pada penelitian berikutnya perlu dilakukan untuk memperoleh gerakan yang lebih baik dan dinamis. Berikut merupakan saran dan perbaikan yang perlu dilakukan:

1. Pemasangan motor servo dengan torsi yang besar disetiap sendinya perlu dilakukan agar gerak robot dapat lebih fleksibel.
2. Untuk mengotomatisasikan robot, pemasangan sensor pada robot perlu dipertimbangkan.
3. Gerakan robot saat melakukan suatu gerakan, setiap sendi dapat bergerak secara bersamaan.

DAFTAR PUSTAKA

Malik, Moh. Ibnu. 2006. Pengantar Membuat Robot. Yogyakarta: Penerbit Gava Media.

Pitowarno, Endra. 2006. Robotika, Disain, Kontrol, dan Kecerdasan Buatan. Yogyakarta: CV Andi Offset.

Sigit, Riyanto, dkk. 2007. Robotika, Sensor, dan Aktuator. Yogyakarta: Graha Ilmu.

Martin, George H. 1985. Kinematika Dan Dinamika Teknik. Michigan State University: Associate Professor Emeritus Of Mechanical Engineering.

Waldron, Kenneth J And Kinzel, Gary L. 1999. Kinematics, Dynamics, And Design Of Machinery. Ohio State University.

Mazidi, Muhammad & Mazidi, Janice, (2000). The 8051 Microcontroller and Embedded Systems. Prentice Hall.

Romy Budhi Widodo & Joseph Dedy Irawan (2007). *Interfacing Paralel & Serial Menggunakan Delphi*. Graha Ilmu, Yogyakarta.

LAMPIRAN

2. Datasheet Motor Servo

HS-635HB



The HS-635BB utilizes the revolutionary â€œKarbonite â„¢â€□ gear train. Karbonite â„¢ is four times stronger than our standard nylon gears and even after hundreds of thousands of cycles it does not show any signs of wear. With a powerful motor, and dual ball-bearing mounted output shaft, the HS-635BB offers plenty of torque and speed. If your application requires increased speed and torque, the HS-635BB is the perfect upgrade from a standard size servo.

Downloads

View our Servo Manual
View our Spec Sheet

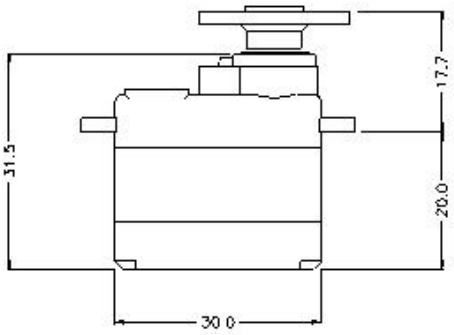
Detailed Specifications

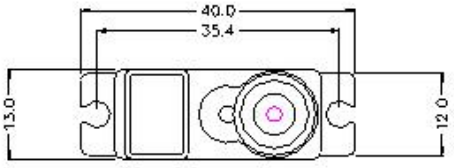
Motor Type:	3 Pole	
Bearing Type:	Dual Ball Bearing	
Speed:	0.18 / 0.15 sec @ 60 deg.	
	English	**Metric**
Torque:	69.43 / 83.32 oz.in (4.8v/6v)	5.0 / 6.0 kg.cm
Size:	1.59" x 0.77" x 1.52"	40.60 x 19.80 x 38.80mm
Weight:	1.76oz	50.00g



TECHNICAL DATA

OPERATING VOLTAGE :	4.8V ~ 6.0V	
TEST VOLTAGE :	AT 4.8V	AT 6.0V
OPERATING SPEED :	0.16s/60	0.12s/60
STALL TORQUE :	3.4Kg.cm(47.21oz.in)	4.0Kg.cm(55.54oz.in)
FEATURES		
MOTOR TYPE :	COLED METAL BRUSH 3POLE MOTOR	
TOP BALL BEARING :	MR106 BALL BEARING	
DIMENTIONS		
SIZE :	30.0x13.0x31.5mm(1.18x0.51x1.24in)	
WEIGHT :	20.61g(0.72oz)	





HS-311



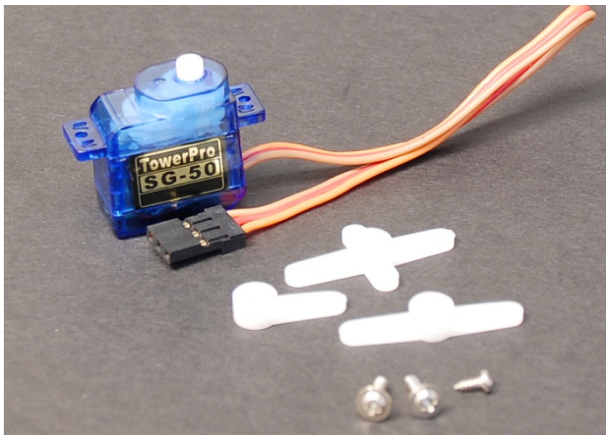
The HS-311 sport servo provides the novice modeler all the performance and reliability you would expect to find in a more expensive servo. Combined with precise resin gears and SMT circuitry, the HS-311 represents a remarkable value in today's R/C market.

Downloads

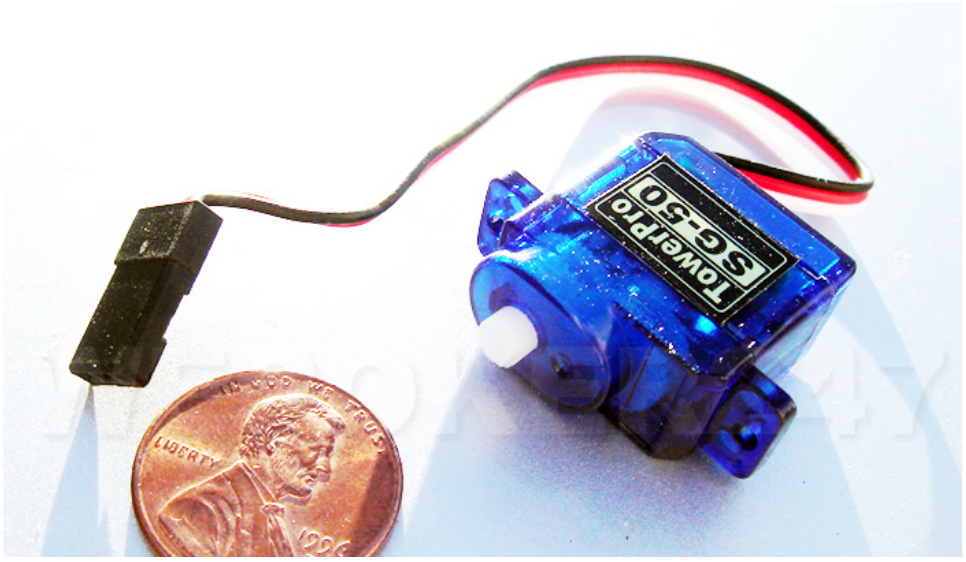
View our Servo Manual
View our Spec Sheet

Detailed Specifications

Motor Type:	3 Pole	
Bearing Type:	Nylon	
Speed:	0.19 / 0.15 sec @ 60 deg.	
	English	**Metric**
Torque:	42 / 48.6 oz.in (4.8v/6v)	3.0 / 3.5 kg.cm
Size:	1.57" x 0.78" x 1.43"	40.00 x 20.00 x 36.50mm
Weight:	1.51oz	43.00g



Operating Voltage: 3.0V - 7.2V
Operating Speed: 0.08sec/60 degrees at no load
Stall Torque: 0.8kg/cm
Motor Type: coreless motor
Gear Type: All Nylon Gear
Connector Wire Length: 150mm
Dimensions: 20.9mm x 11.4mm x 22mm
Weight: 5.0g



HS-65MG



Considered one of the most powerful servos in the micro class, the HS-65 is offered with a choice of "shock resistant" Metal or "zero wear" Karbonite gears. The 65 series features a top-ball bearing for long life and positive centering, 31 ounces/inches of power at 6 volts and is an excellent choice for higher performance micro helicopters, electric park flyers, and 1/18 scale cars.

Downloads

View our Servo Manual
View our Spec Sheet

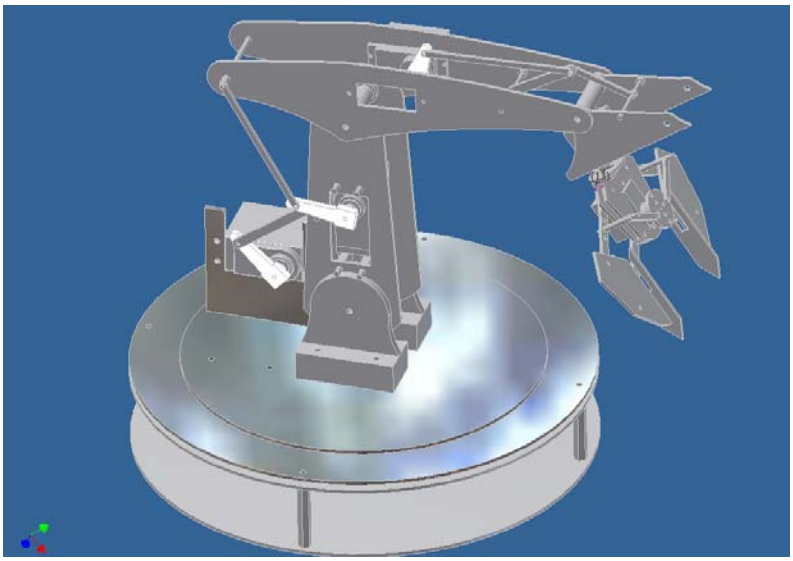
Detailed Specifications

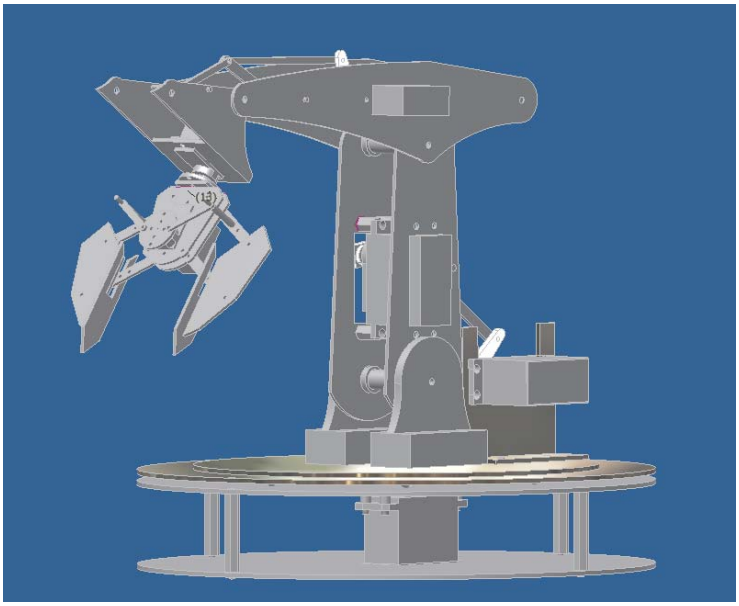
Motor Type:	3 Pole	
Bearing Type:	Top Ball Bearing	
Speed:	0.14 / 0.11 sec @ 60 deg.	
	English	**Metric**
Torque:	24.99 / 30.55 oz.in (4.8v/6v)	1.8 / 2.2 kg.cm
Size:	0.92" x 0.45" x 0.94"	23.60 x 11.60 x 24.00mm
Weight:	0.42oz	11.90g

LAMPIRAN

1. Gambar Teknik

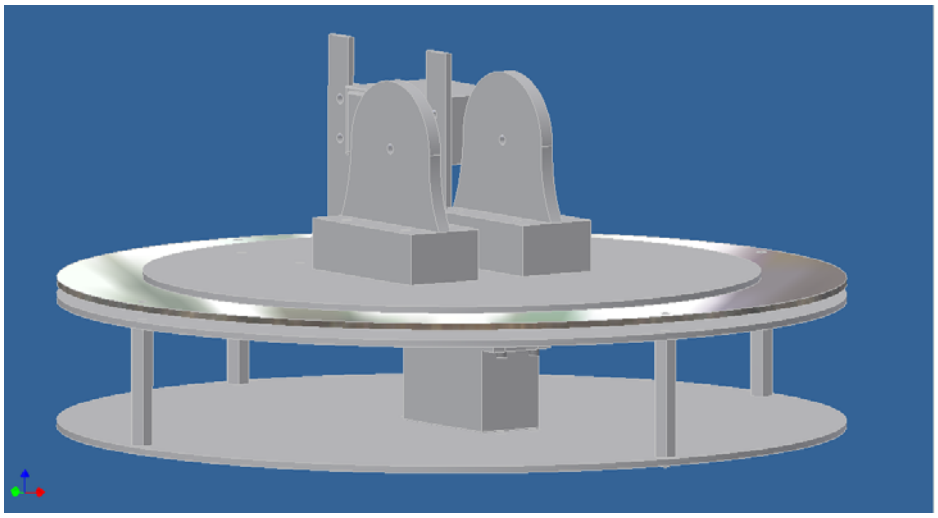
1.1 Hasil Perancangan Lengan Robot Dengan Enam Derajat Kebebasasan

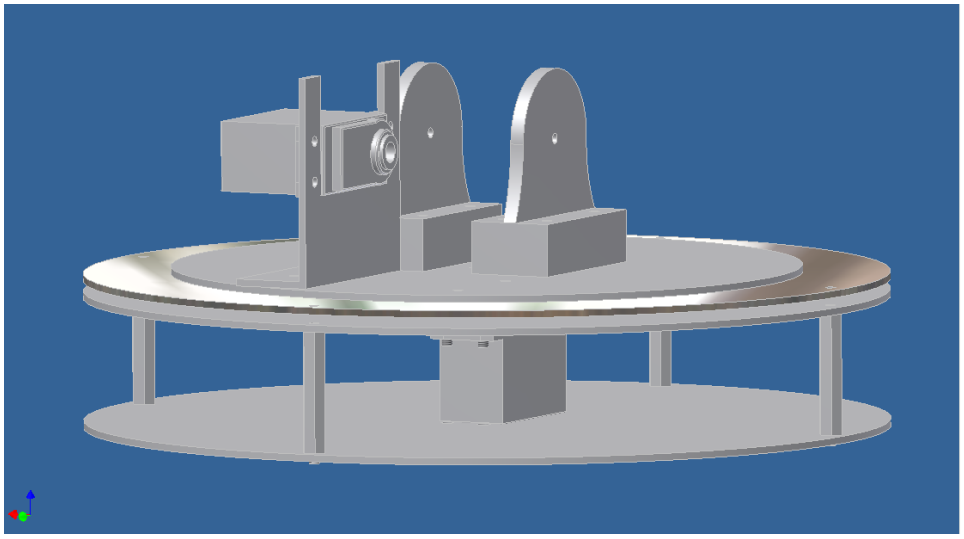




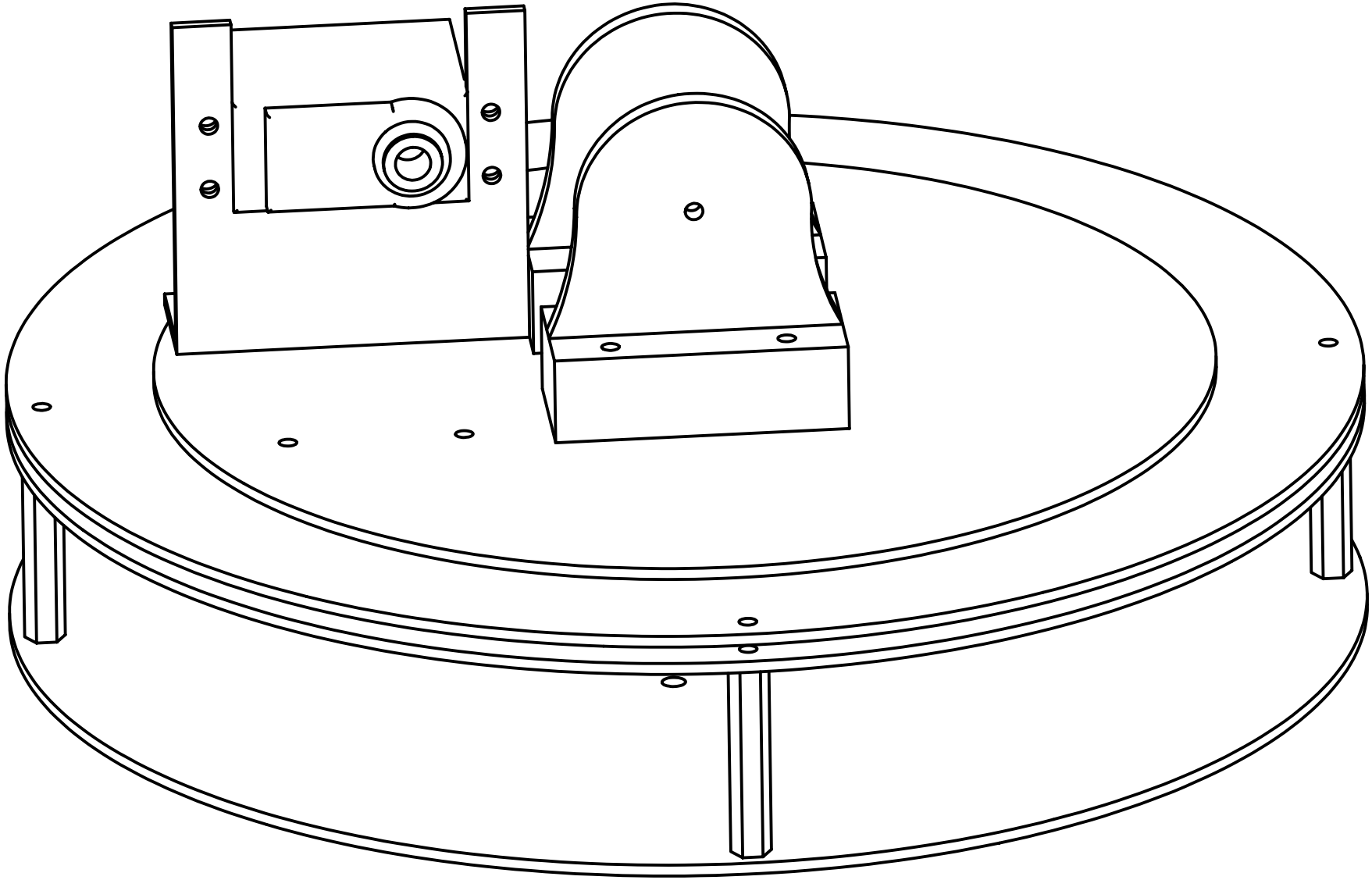
Gambar 1.1 : *Hasil Perancangan Lengan Robot*

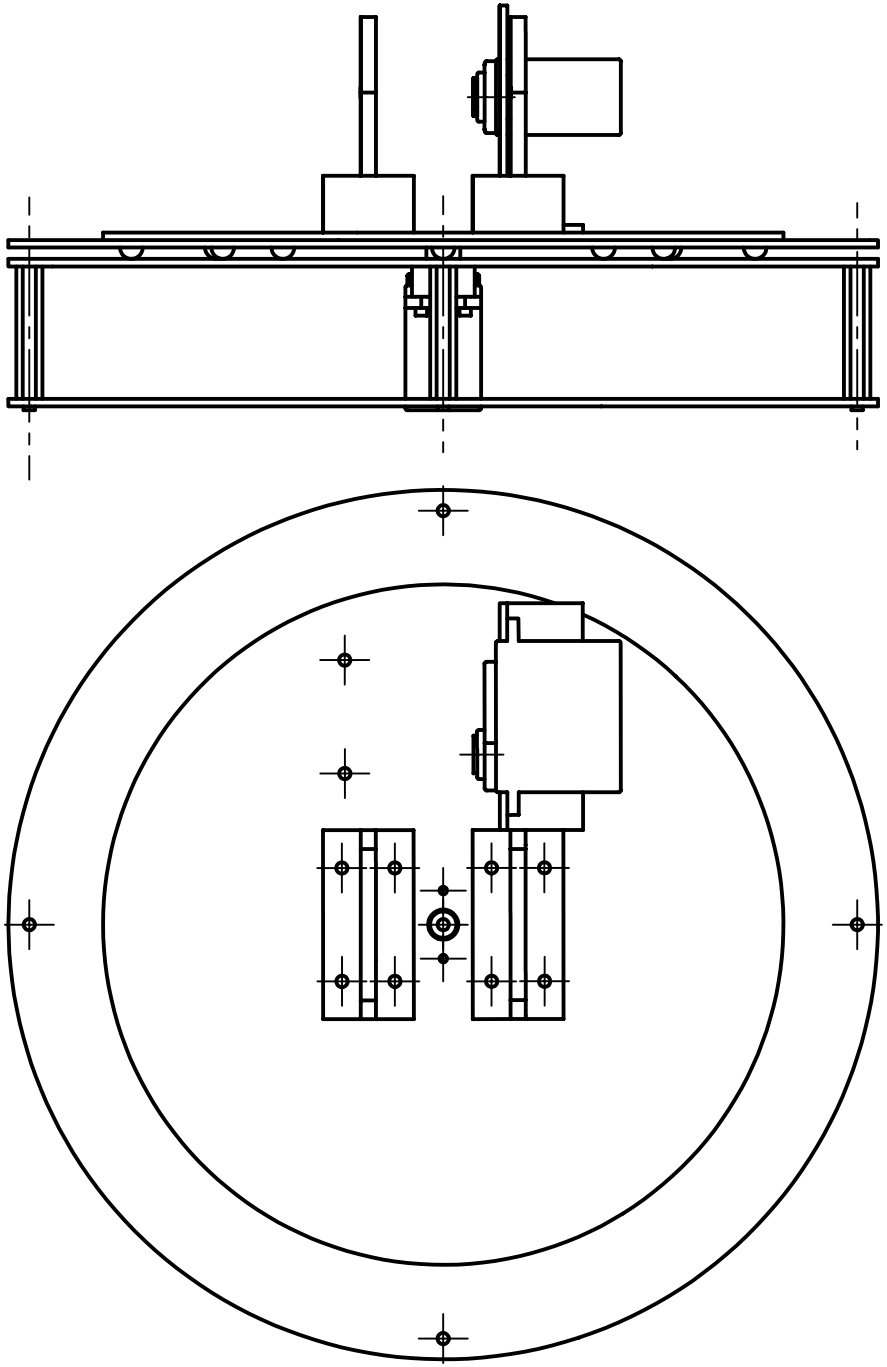
1.1 Bagian Bawah

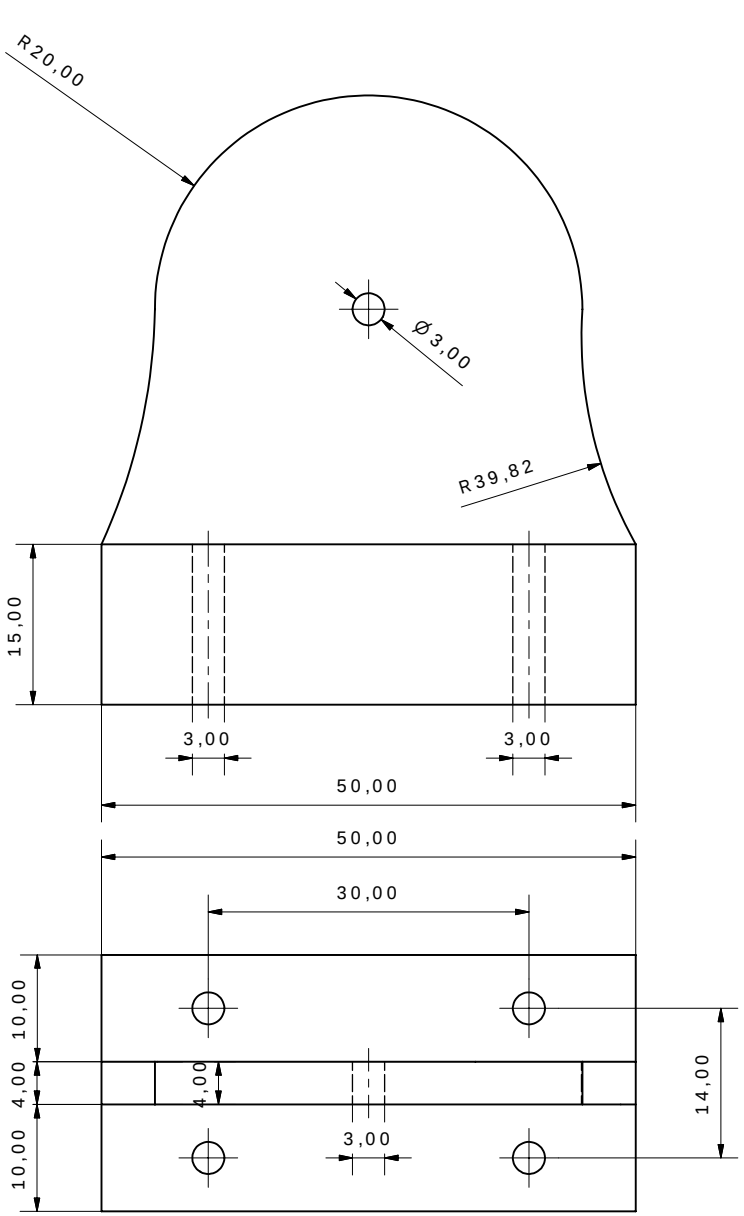


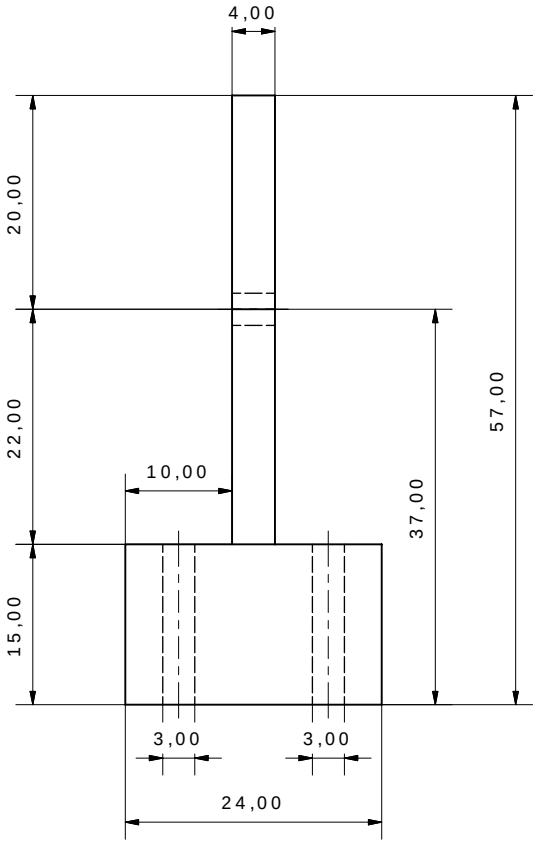


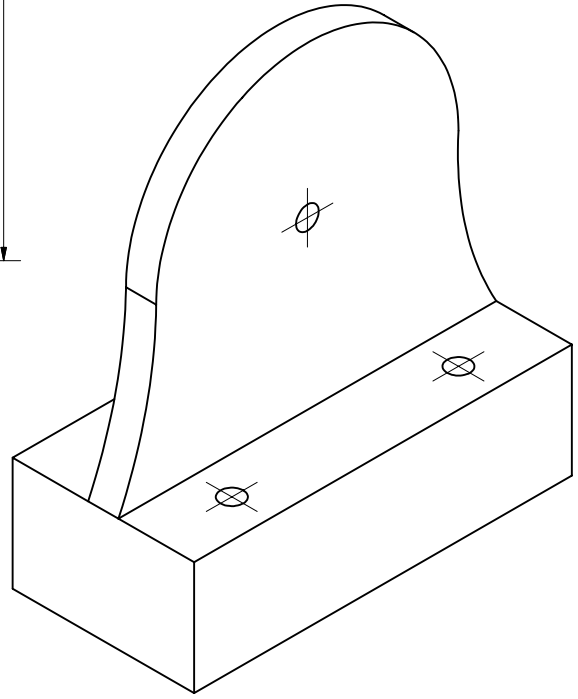
Gambar 1.2 : *Perancangan Bagian Bawah (Manipulator 1) Lengan Robot*





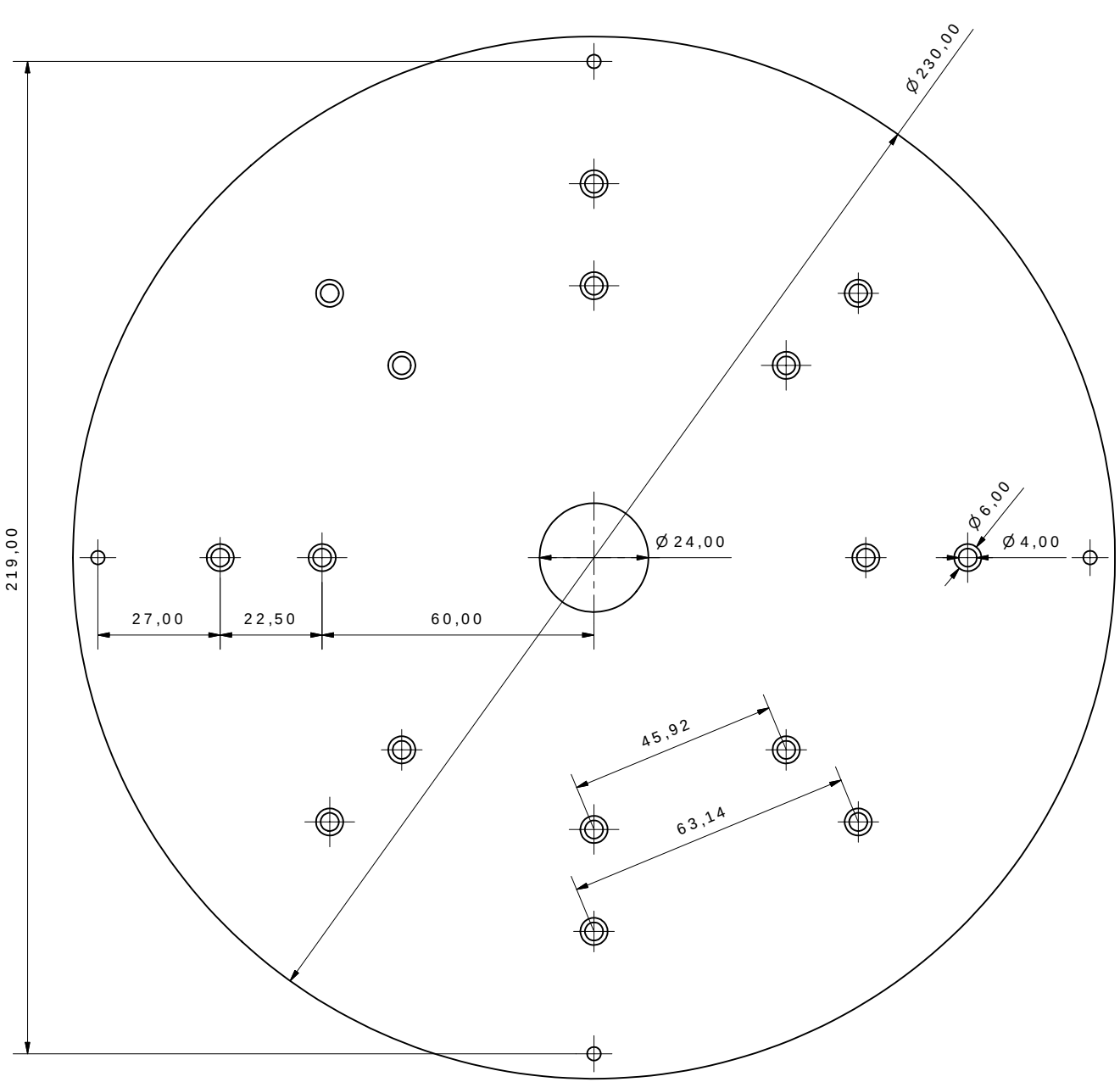


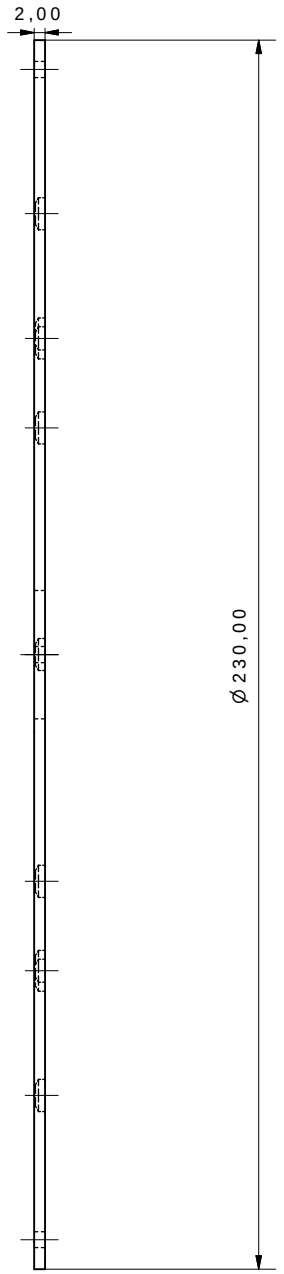




Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/20/2008	

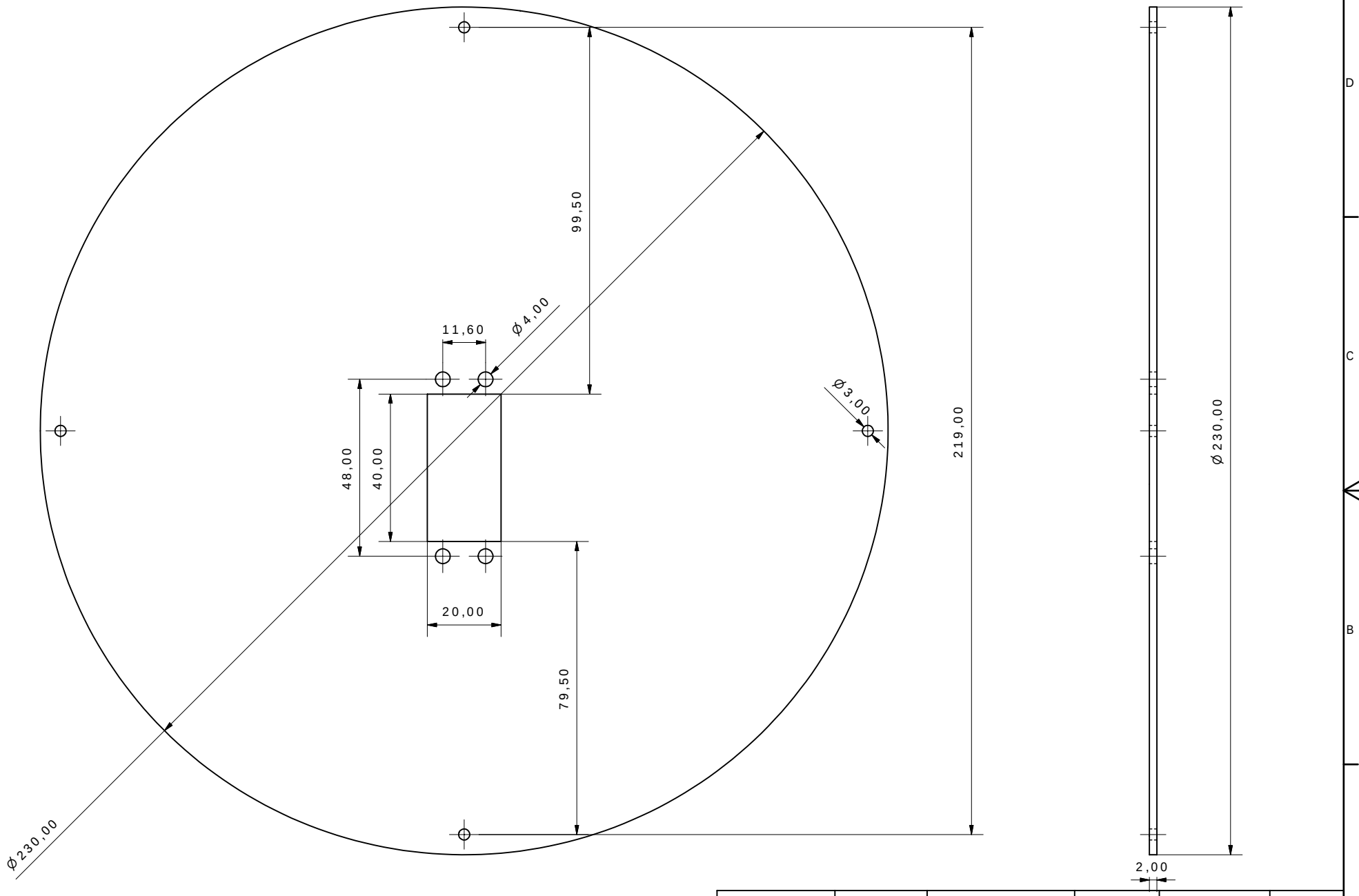
kaki bawah 2		Edition	Sheet 1 / 1





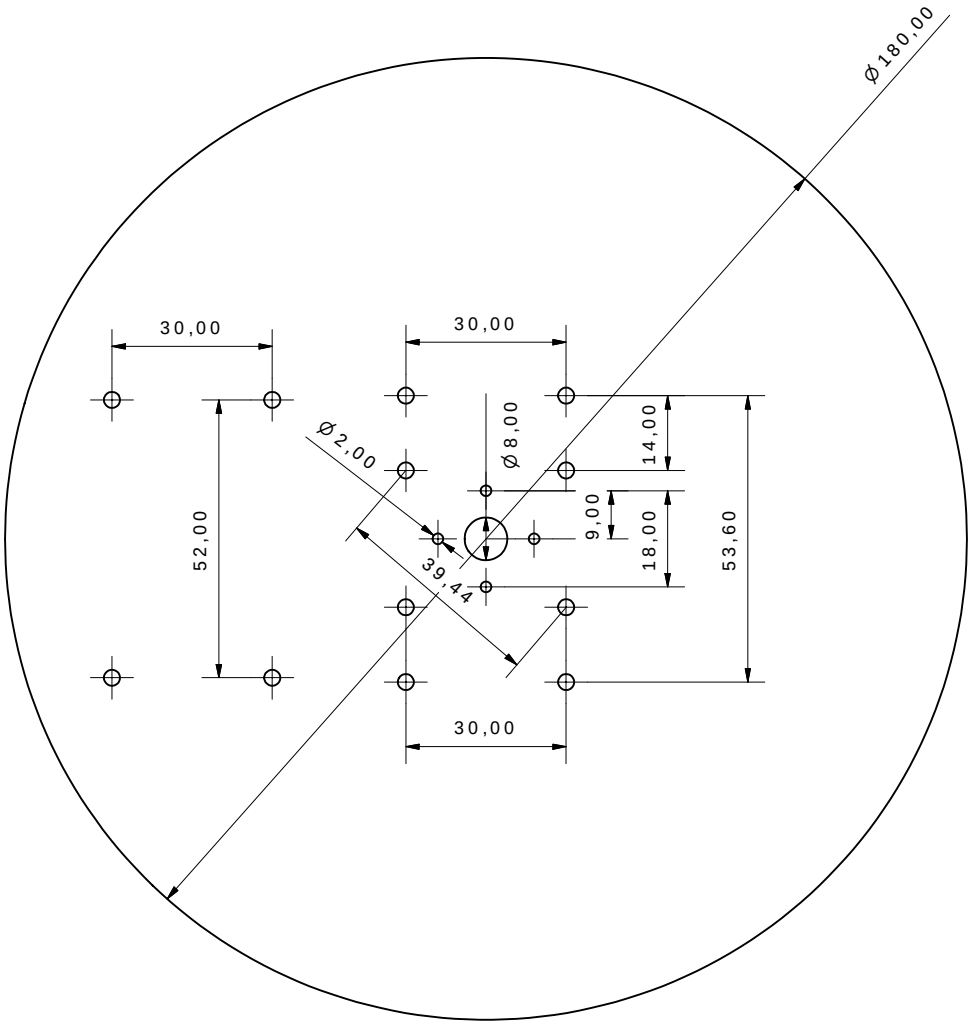
Designed by	Checked by	Approved by	Date	Date	
GUSTI HIDAYAT				12/20/2008	

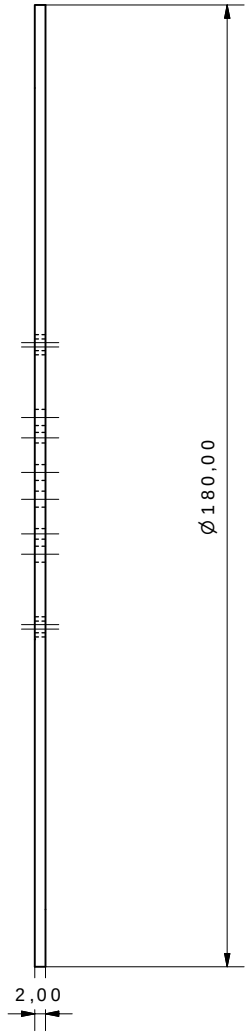
landasan bawah 2	Edition	Sheet	1 / 1



Designed by	Checked by	Approved by	Date	Date	
GUSTI HIDAYAT				12/20/2008	

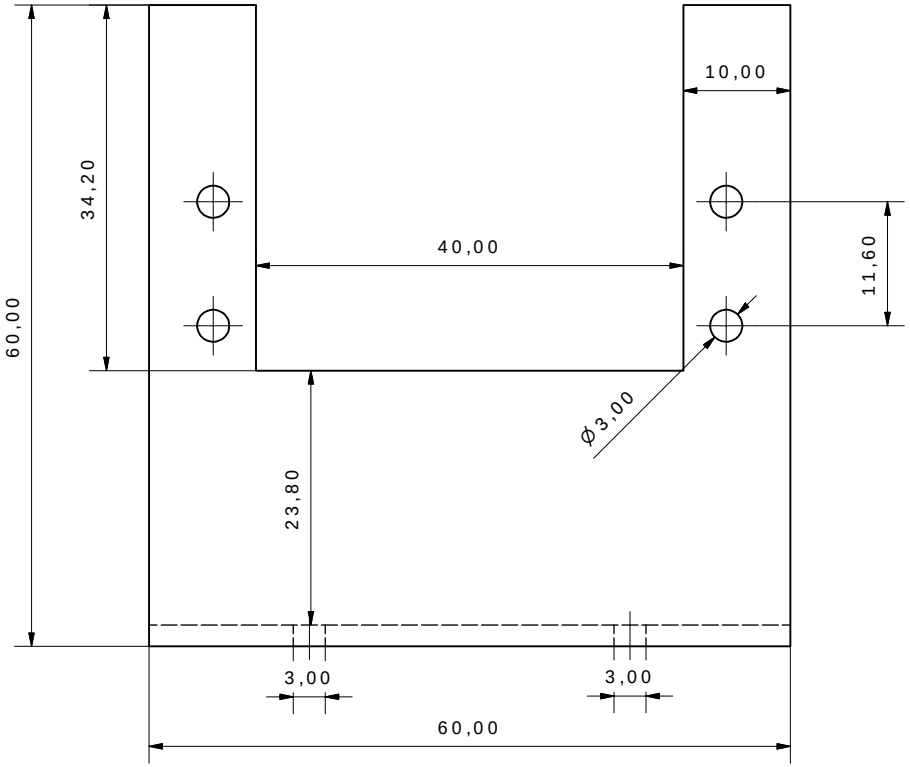
landasan bawah 3	Edition	Sheet 1 / 1	

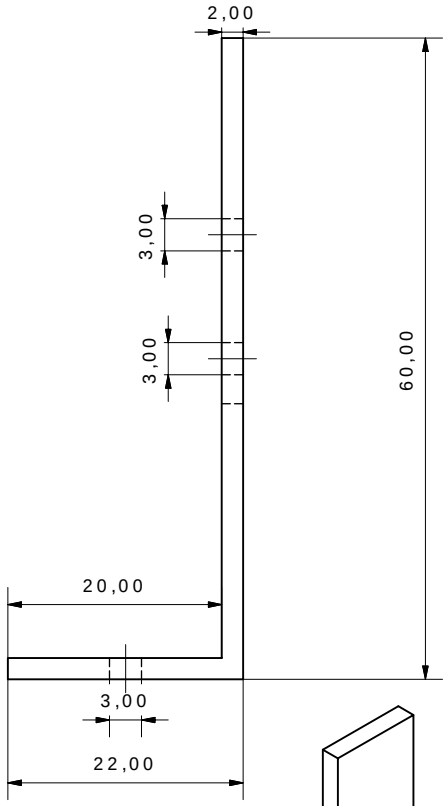


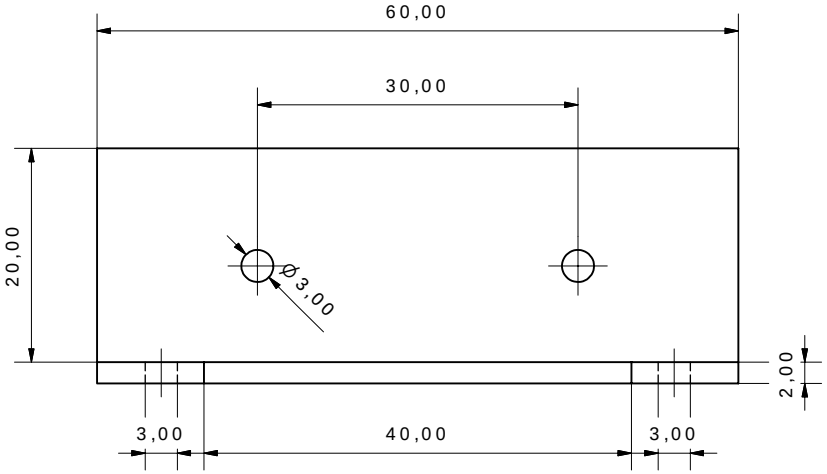


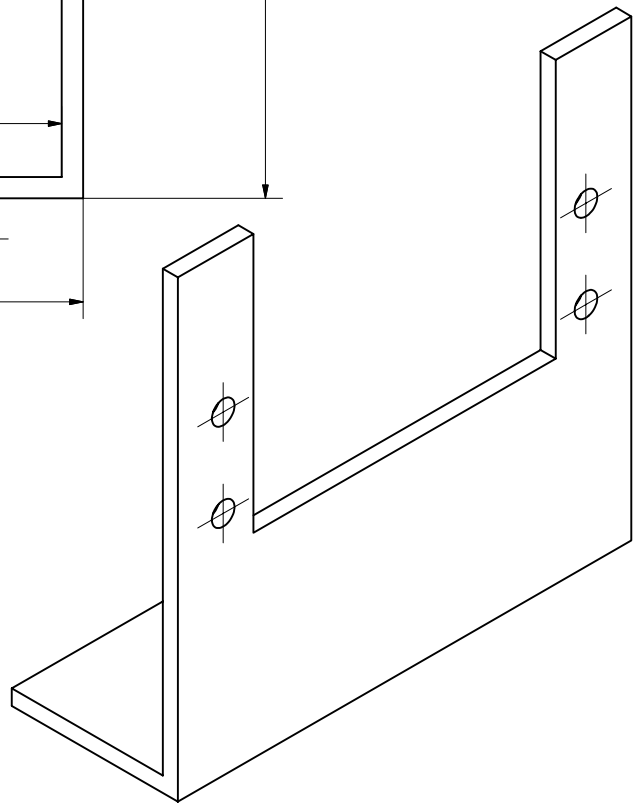
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/20/2008	

landasan putar		Edition	Sheet 1 / 1



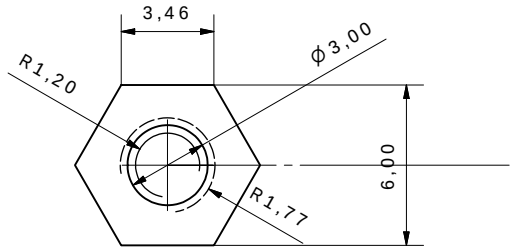


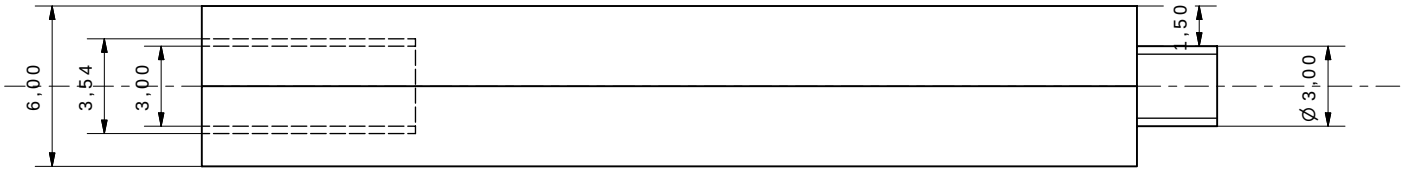


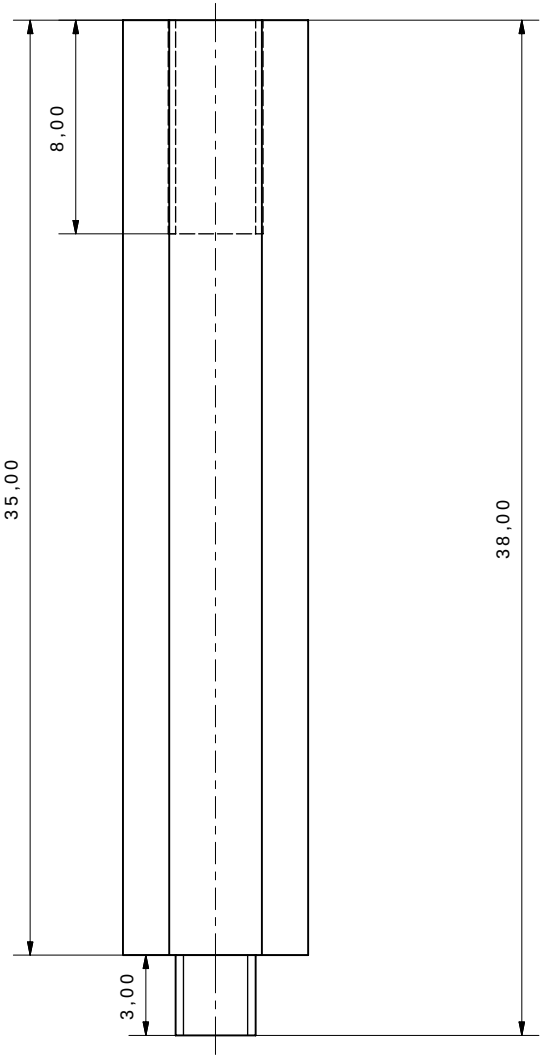


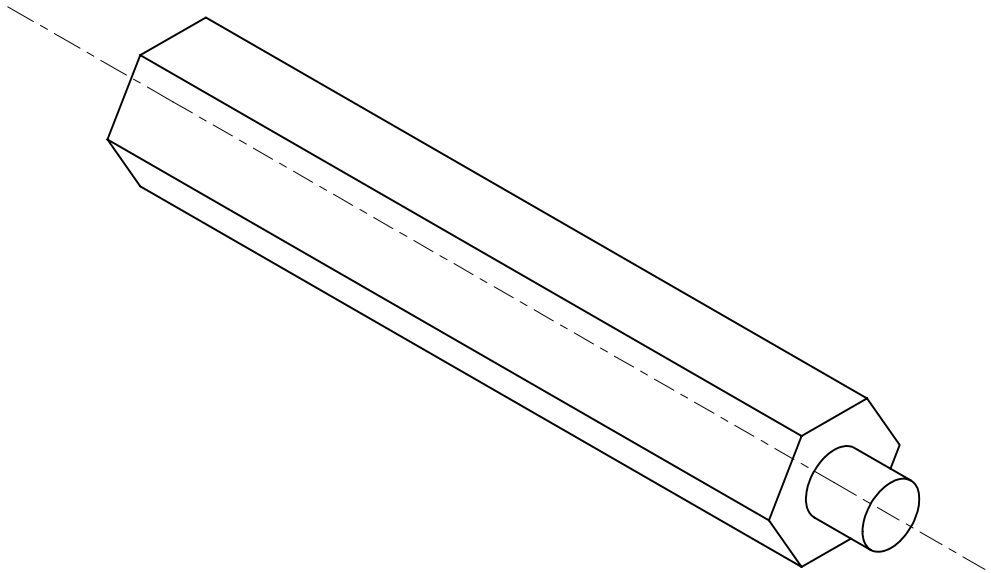
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/20/2008	

servo 2 fixture		Edition	Sheet 1 / 1





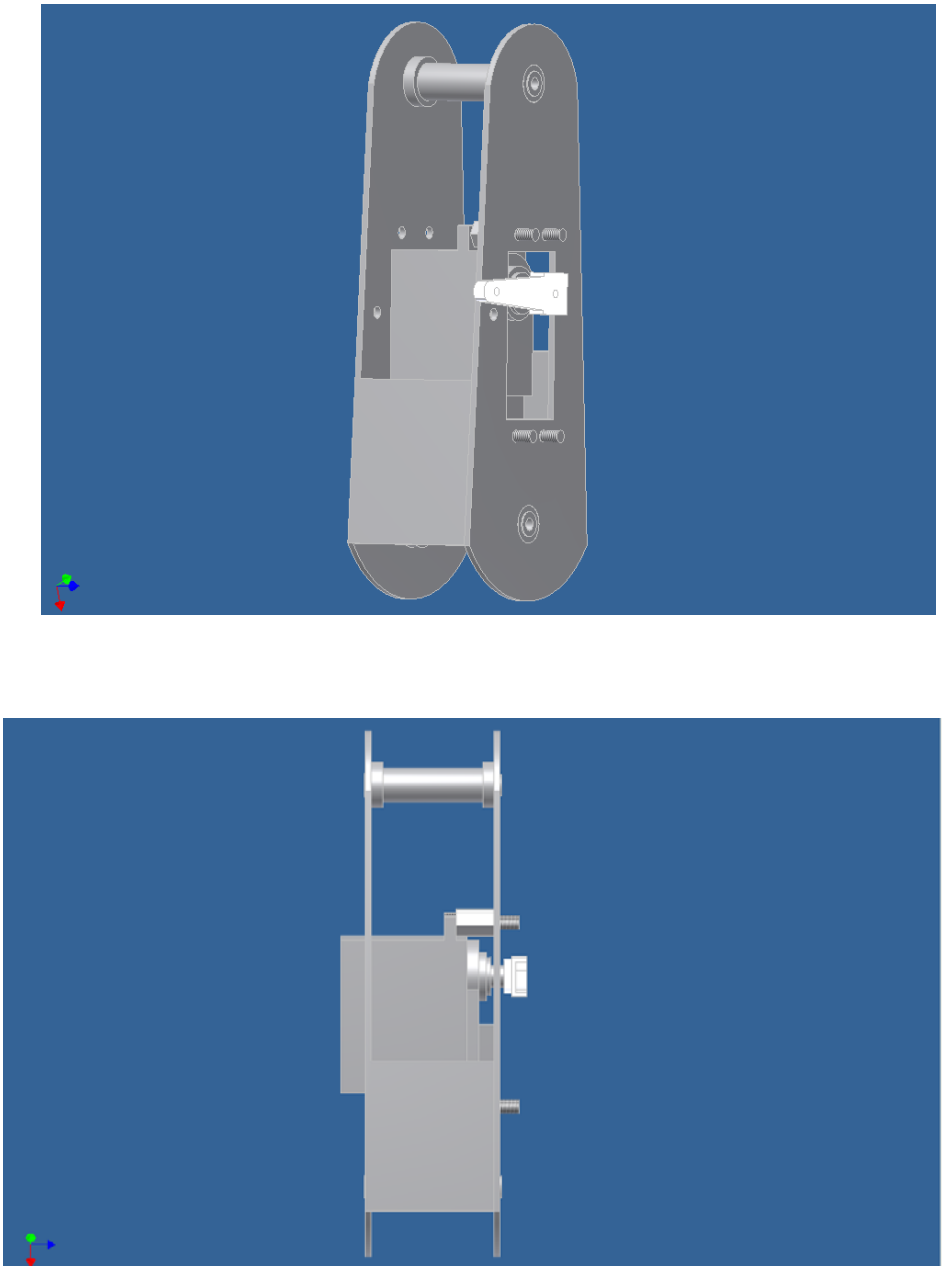




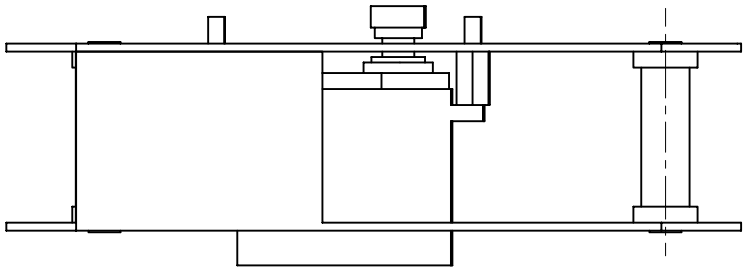
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/20/2008	

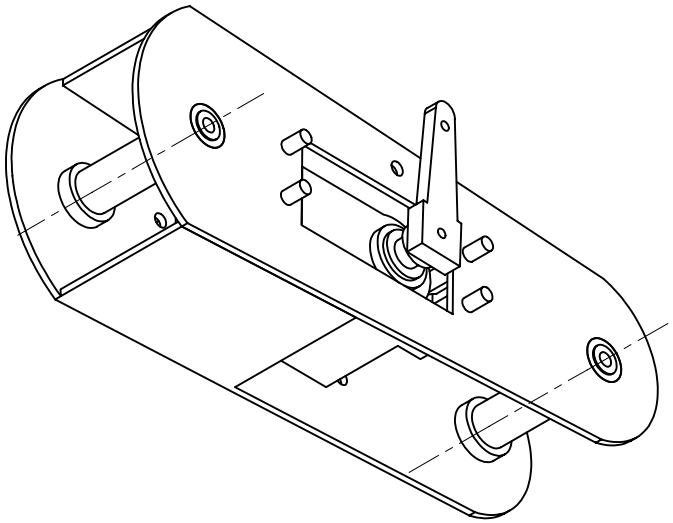
spacer 35 mm		Edition	Sheet 1 / 1

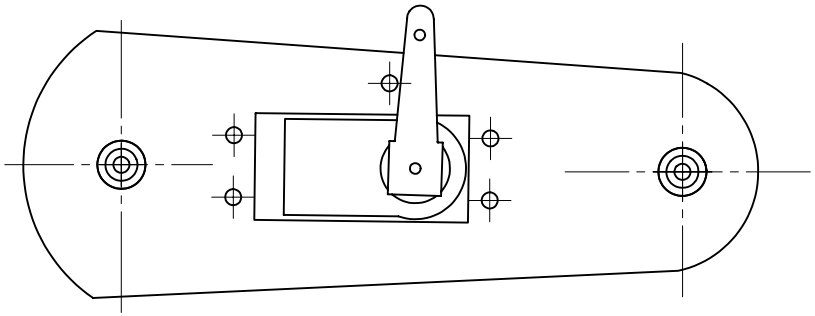
1.1 Manipulator 2

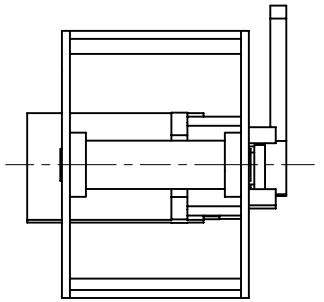


Gambar 1.3 : *Perancangan Manipulator 2 Lengan Robot*



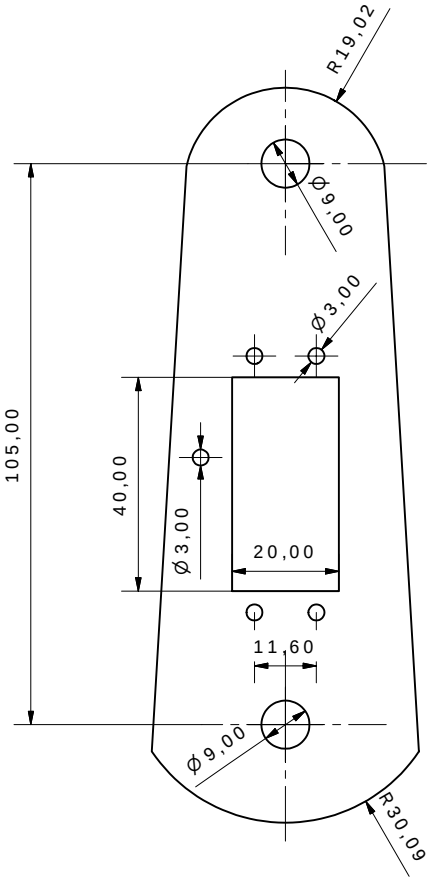


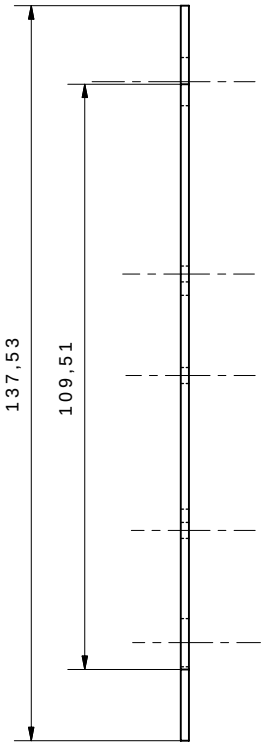


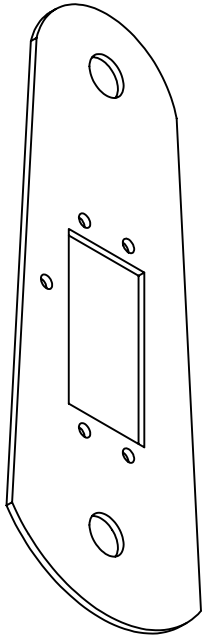


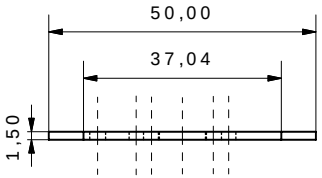
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/21/2008	

Assembly2 manipulator 2	Edition	Sheet 1 / 1	



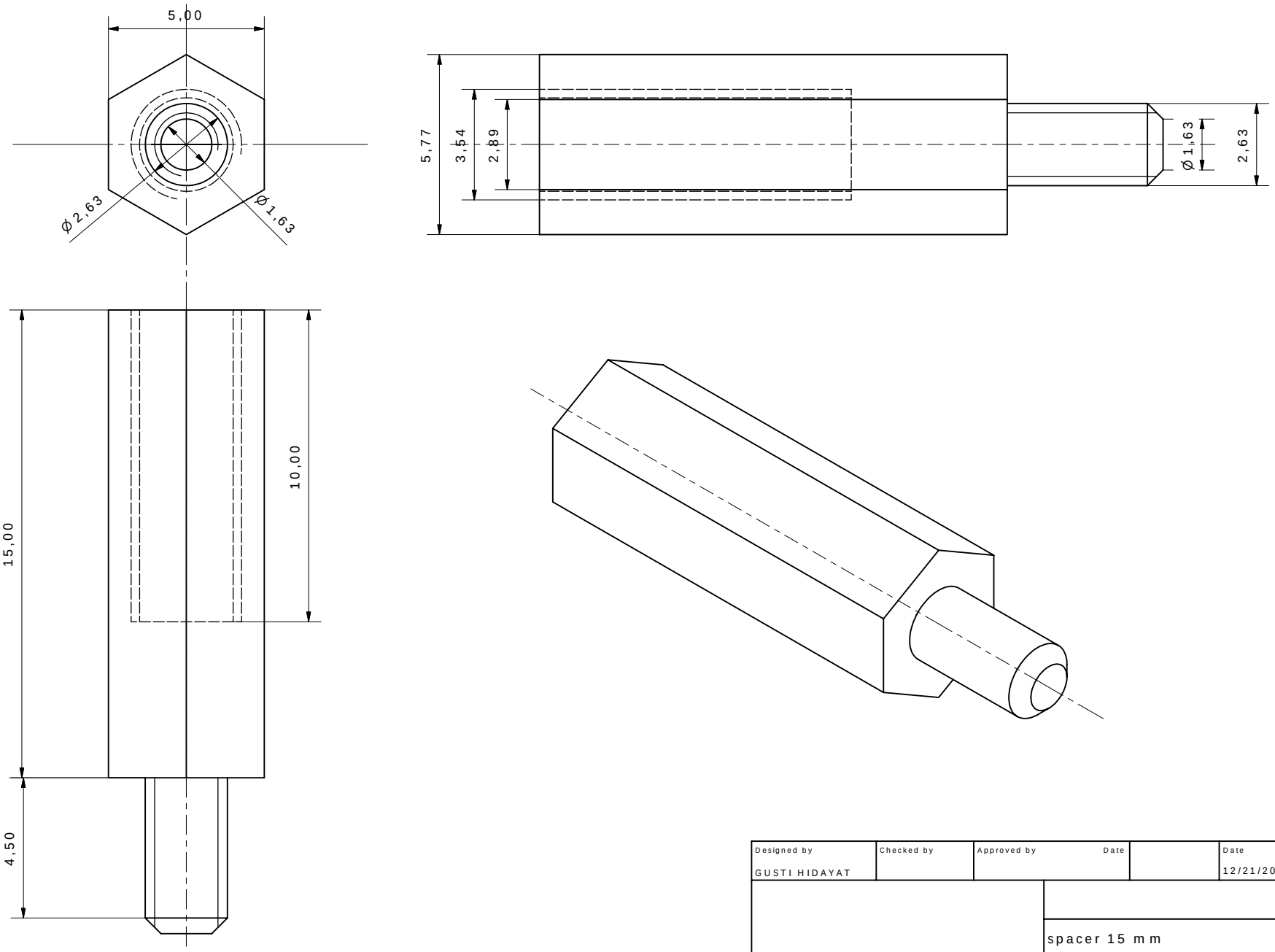






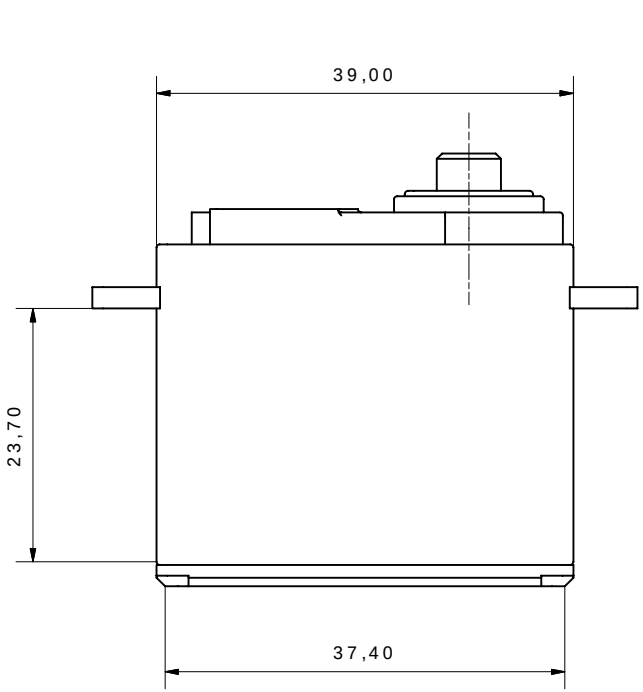
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/21/2008	

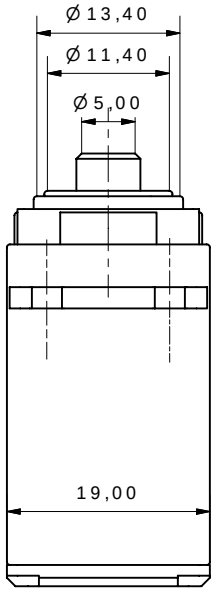
new arm1		Edition	Sheet 1 / 1

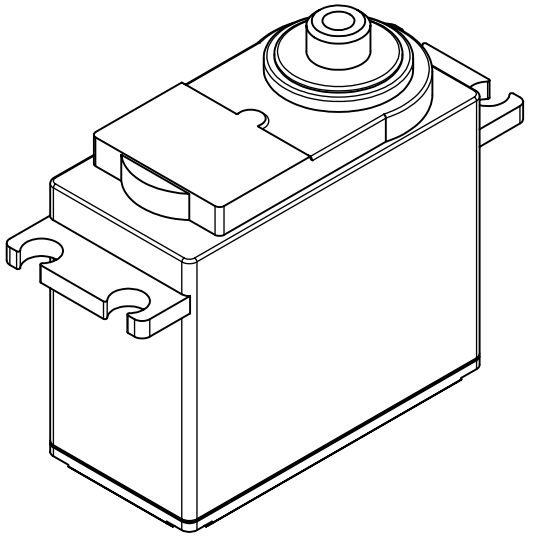


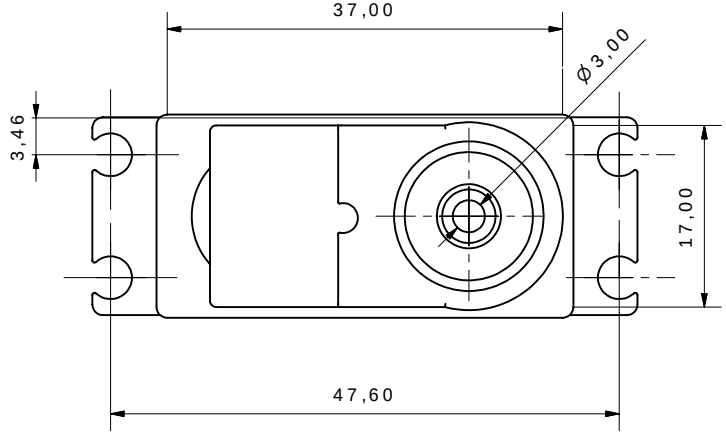
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/21/2008	

spacer 15 mm		Edition	Sheet 1 / 1



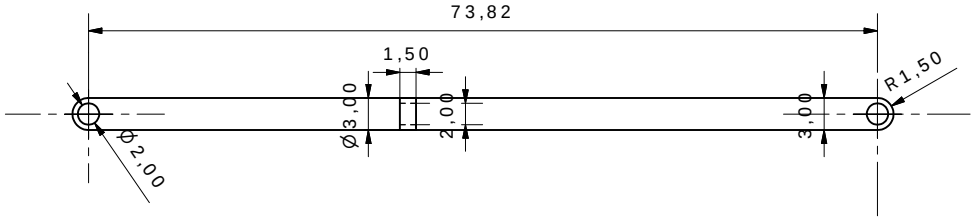


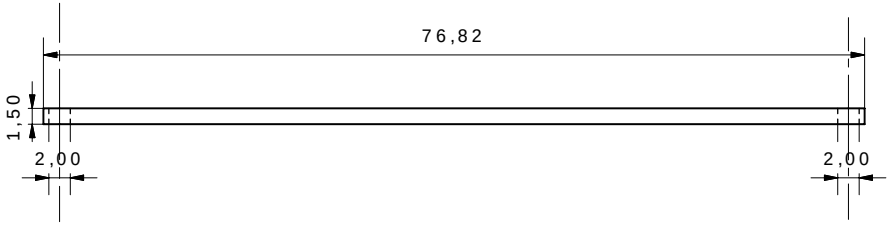


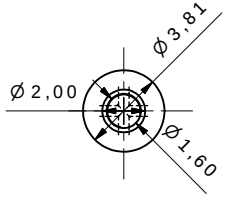


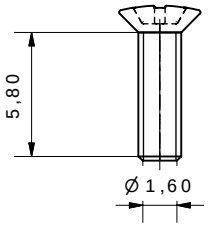
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/21/2008	

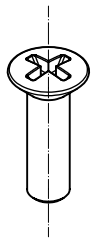
servo		Edition	Sheet 1 / 1

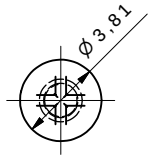


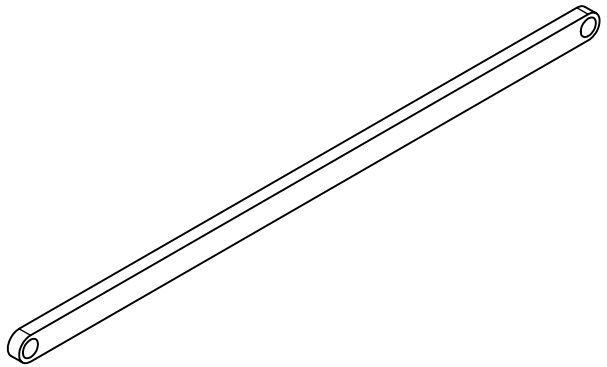






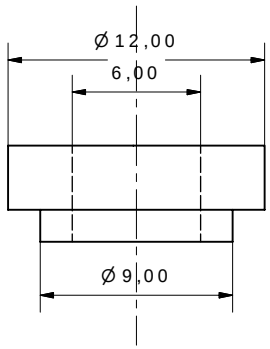


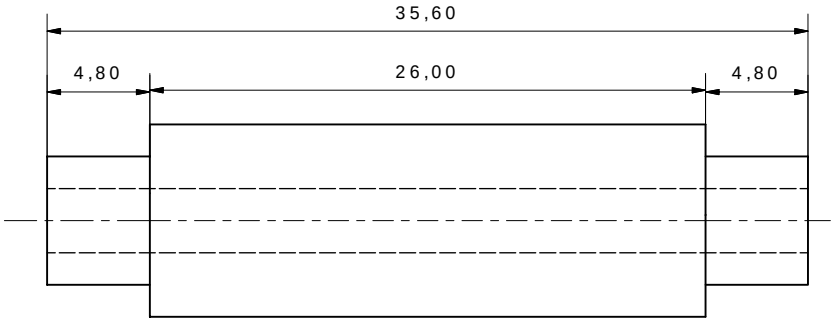


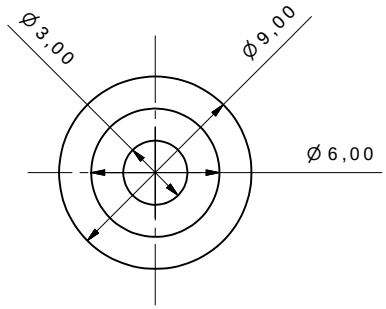


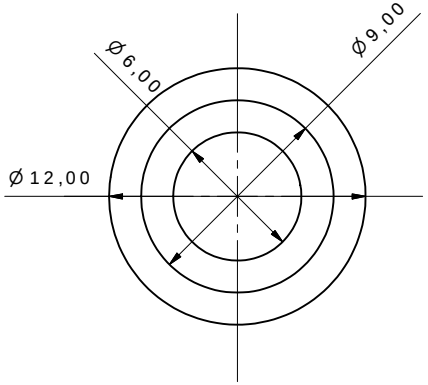
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/21/2008	

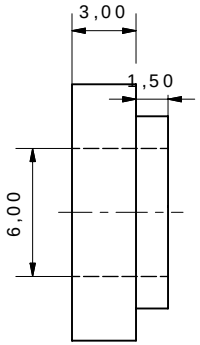
link dan baut		Edition	Sheet 1 / 1

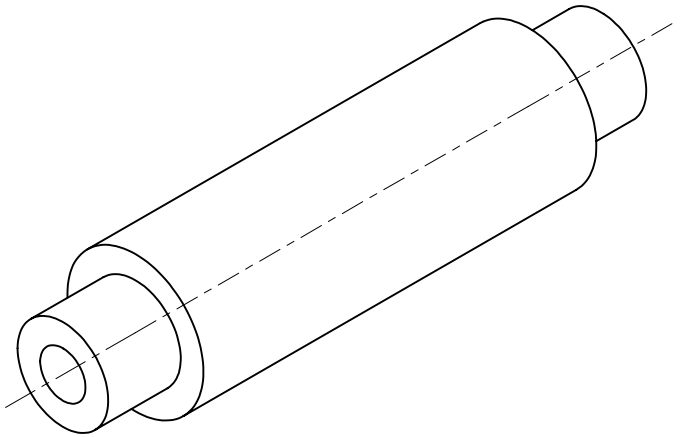


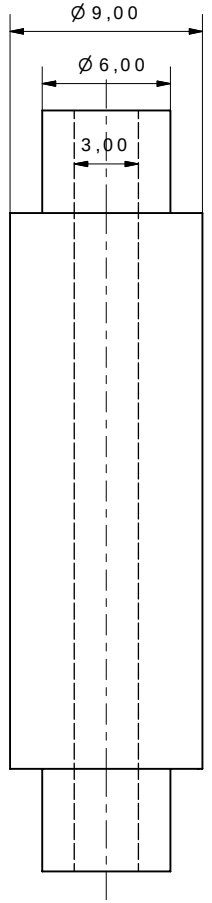


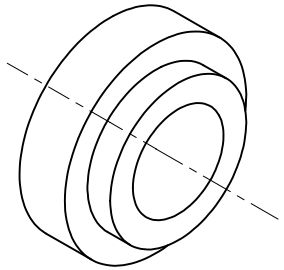






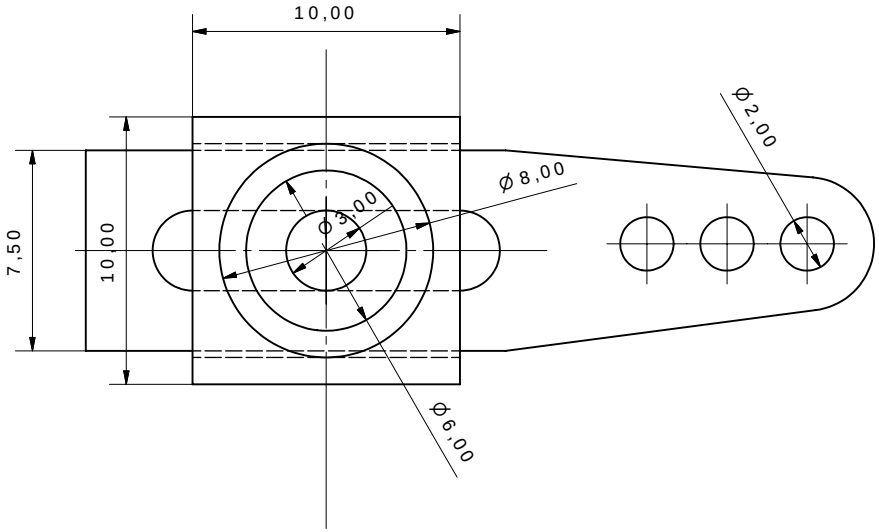


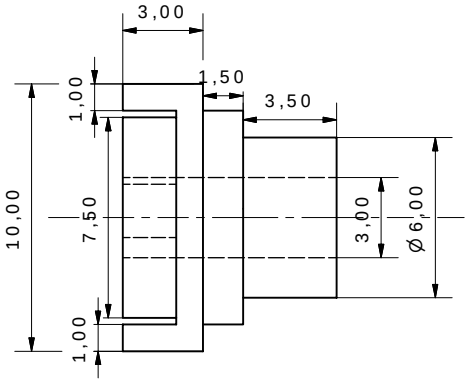


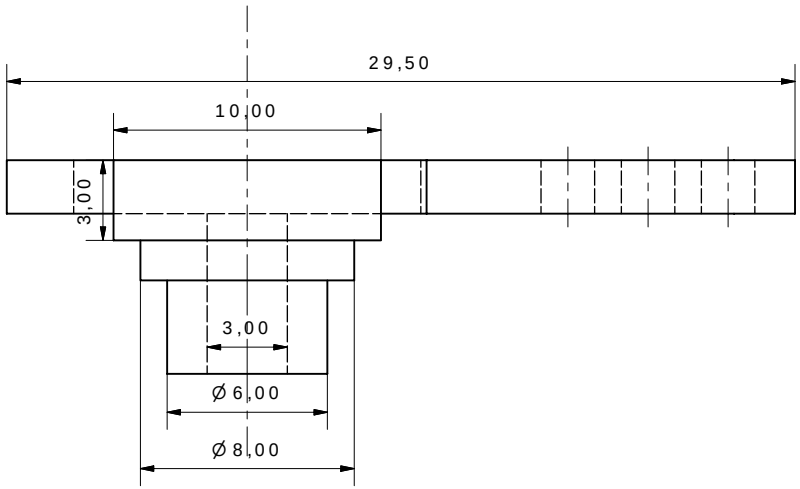


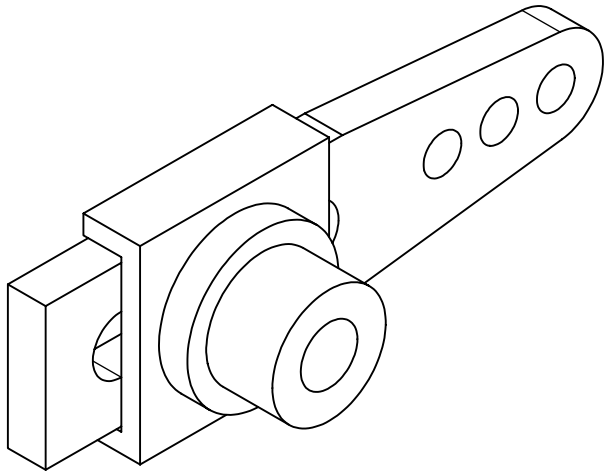
Designed by	Checked by	Approved by	Date	Date	
GUSTI HIDAYAT				12/21/2008	

bushing 12,9,6	Edition	Sheet 1 / 1	





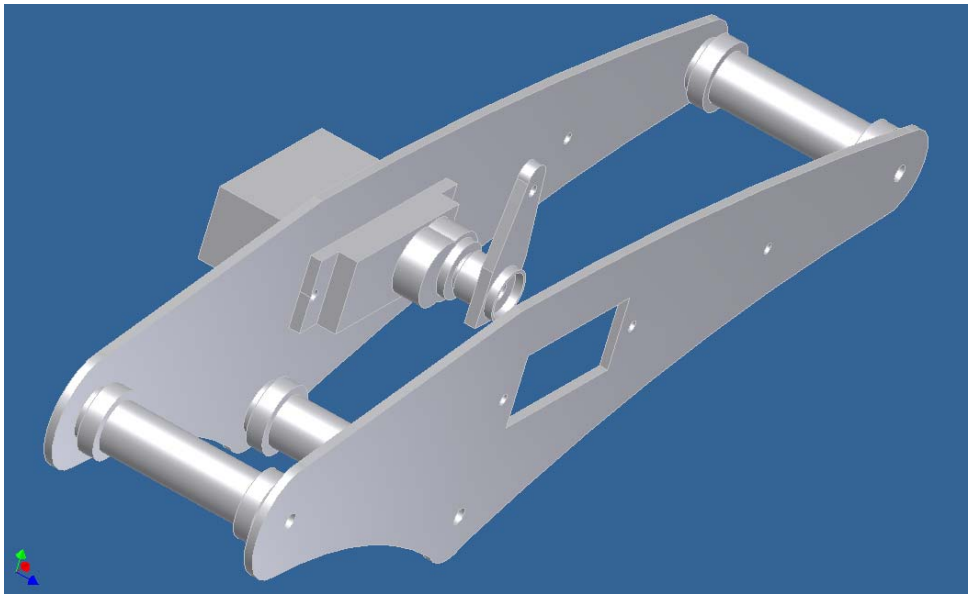


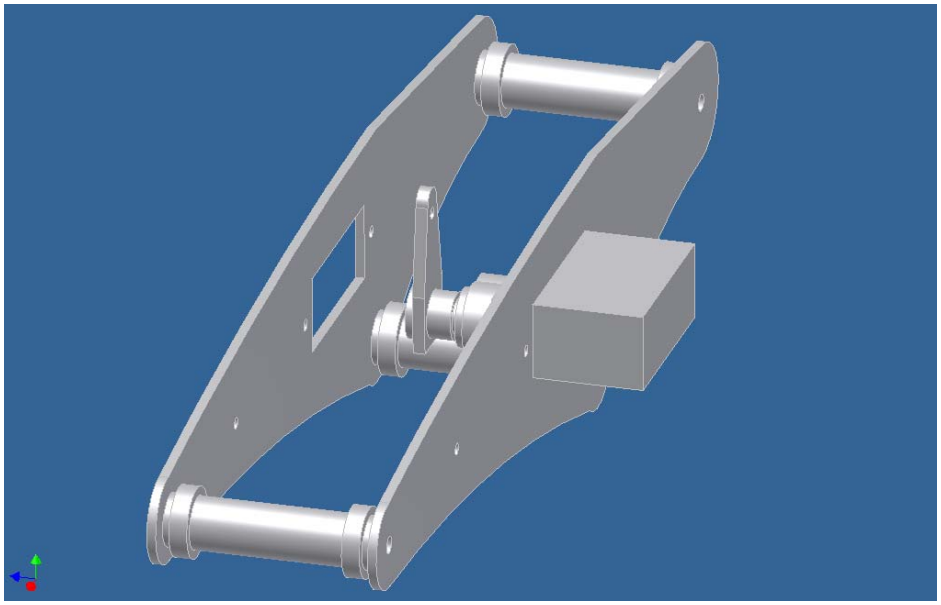


Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/21/2008	

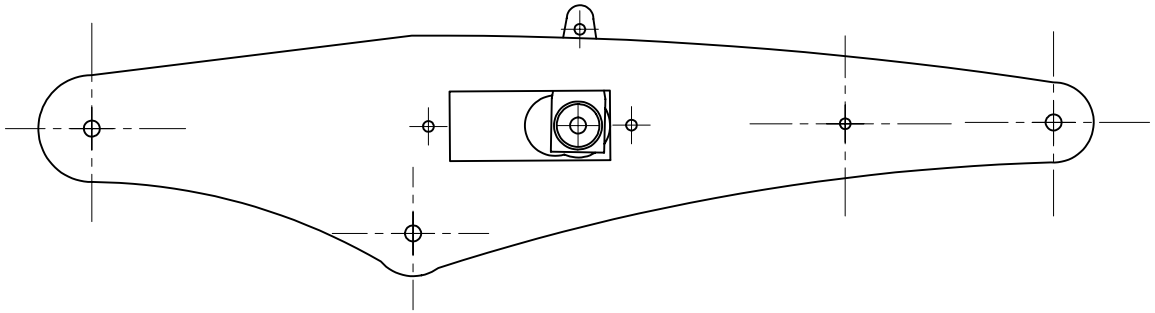
tuas servo1_MIR		Edition	Sheet 1 / 1

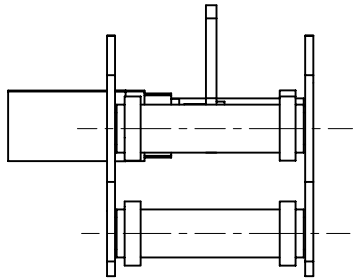
1.1 Manipulator 3

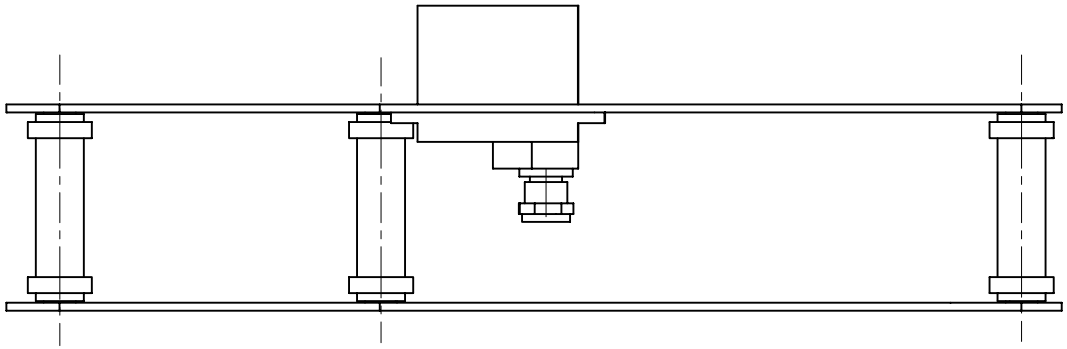


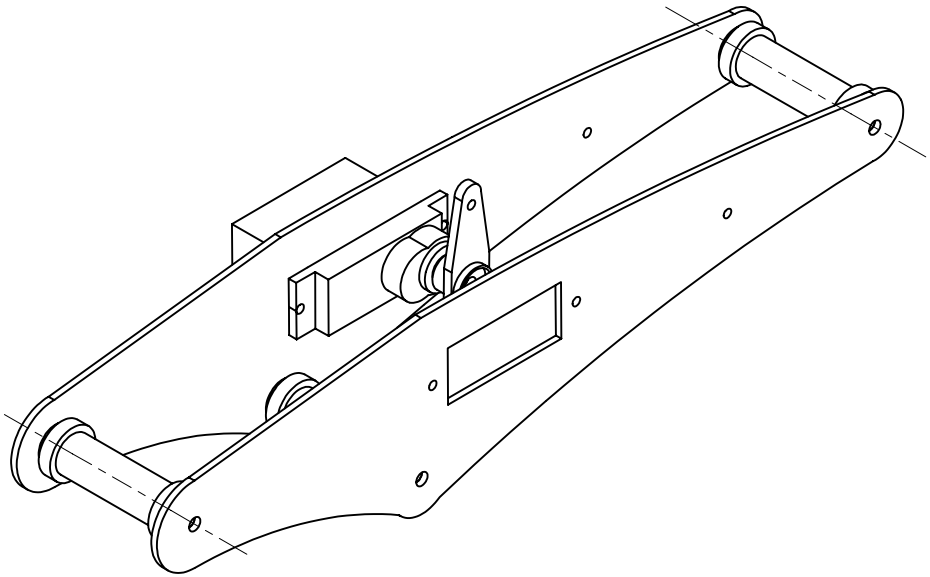


Gambar 1.4 : *Perancangan Manipulator 3 Lengan Robot*



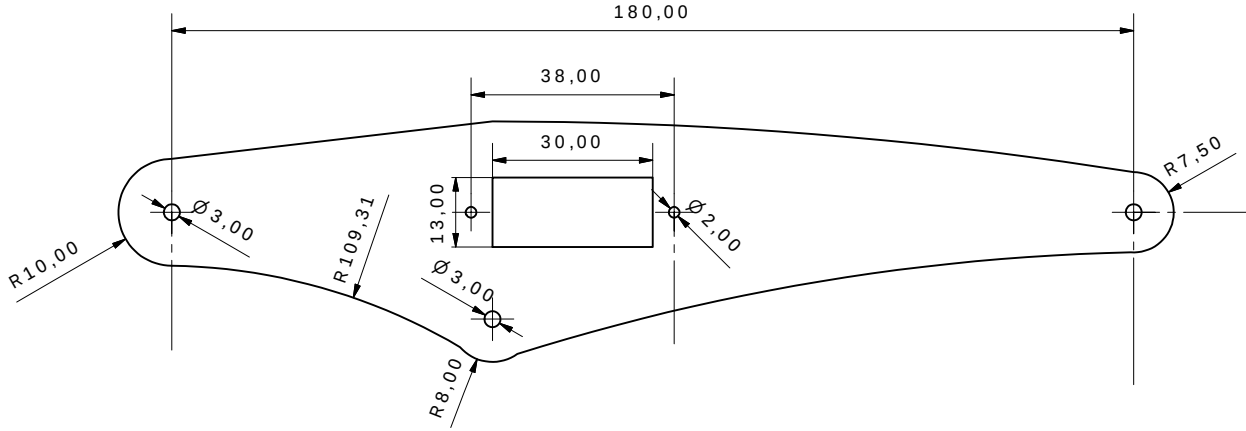


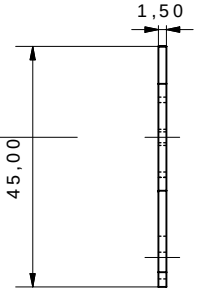


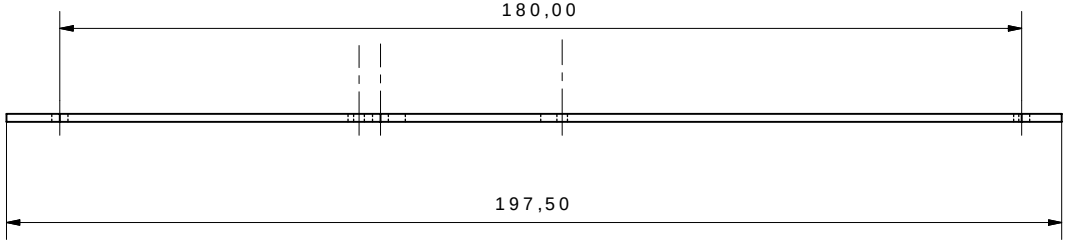


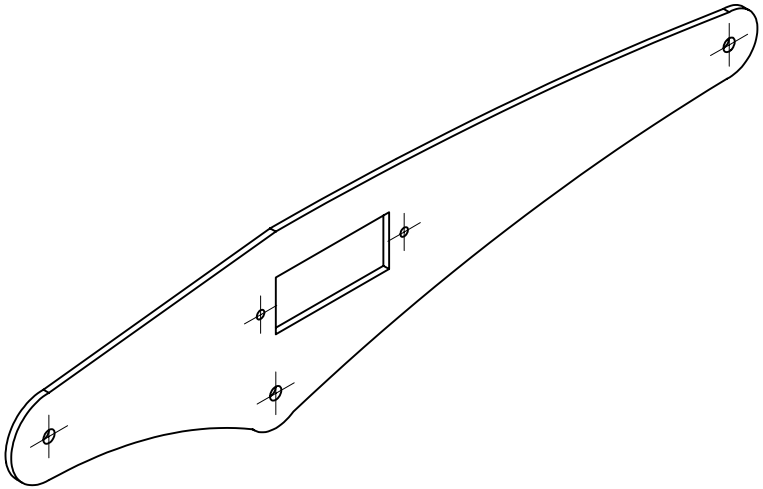
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/22/2008	

Assembly1		Edition	Sheet 1 / 1



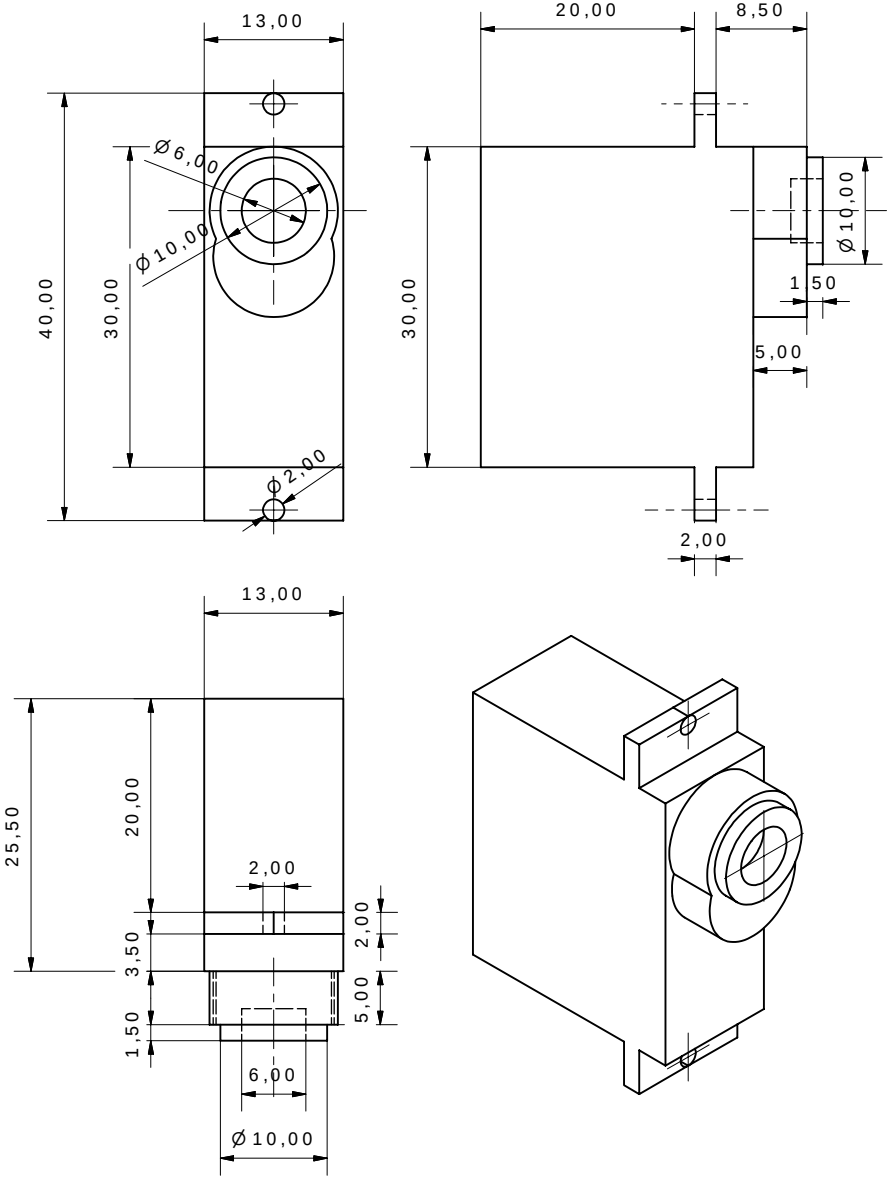


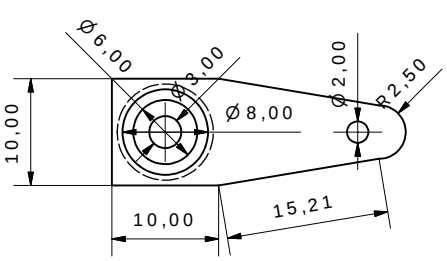


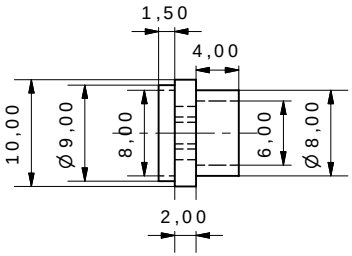


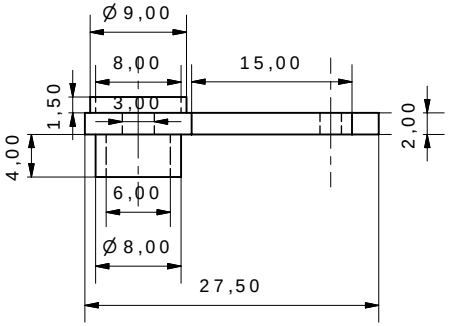
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/21/2008	

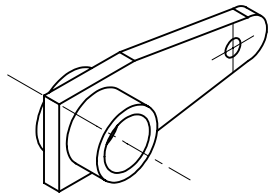
armse2		Edition	Sheet 1 / 1

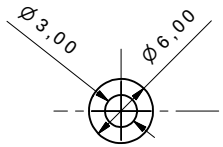


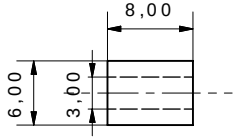


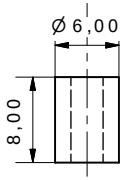


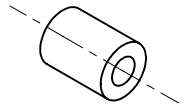








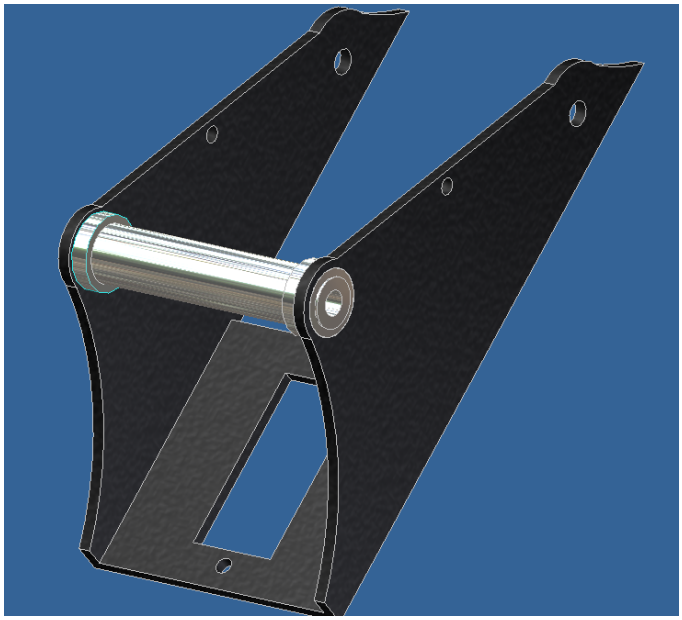


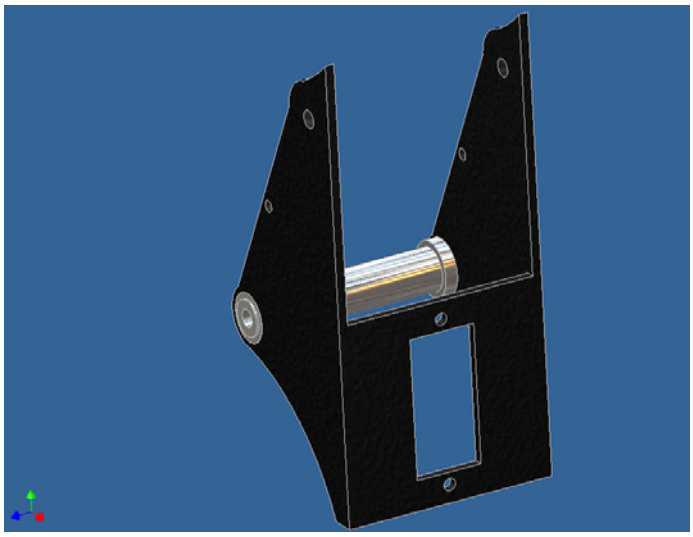


Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/21/2008	

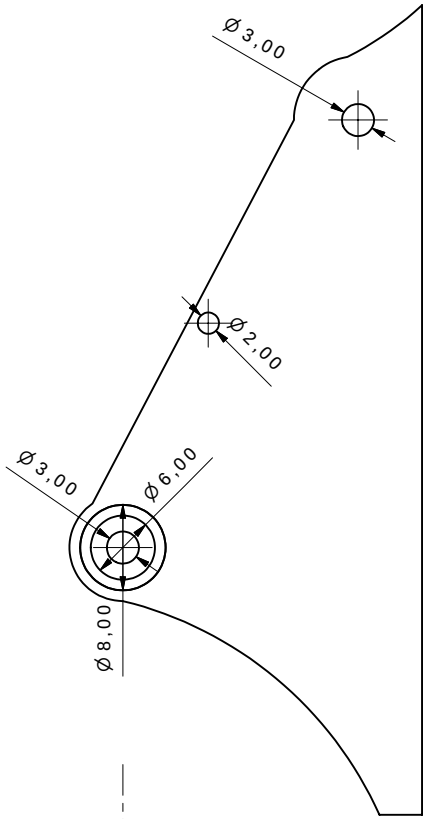
servo kecil		Edition	Sheet 1 / 1

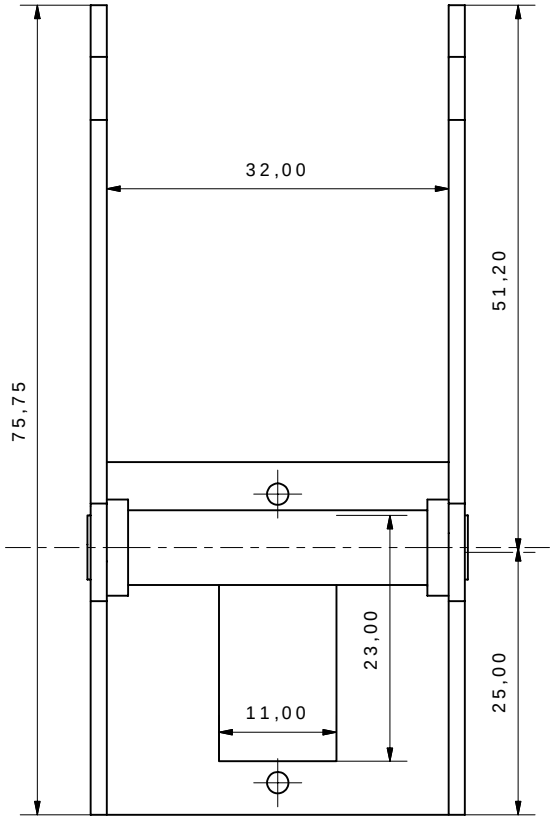
1.1 Manipulator 4

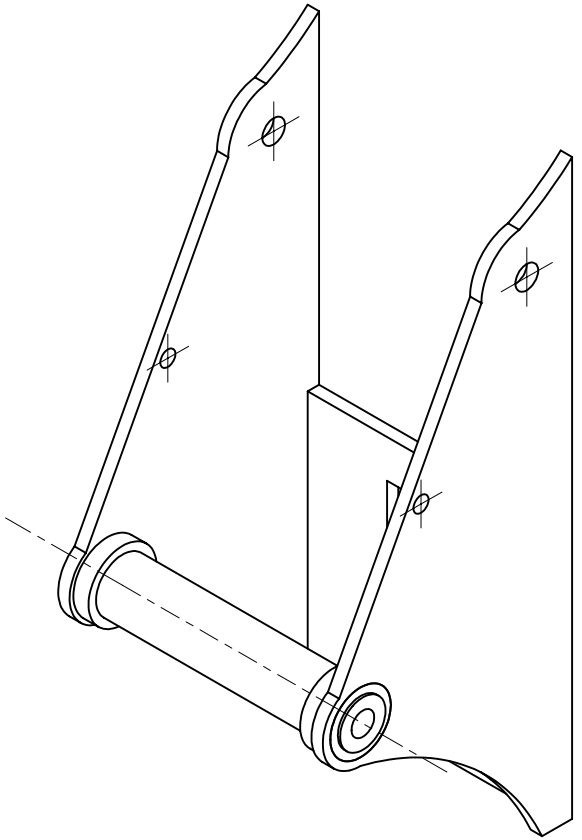


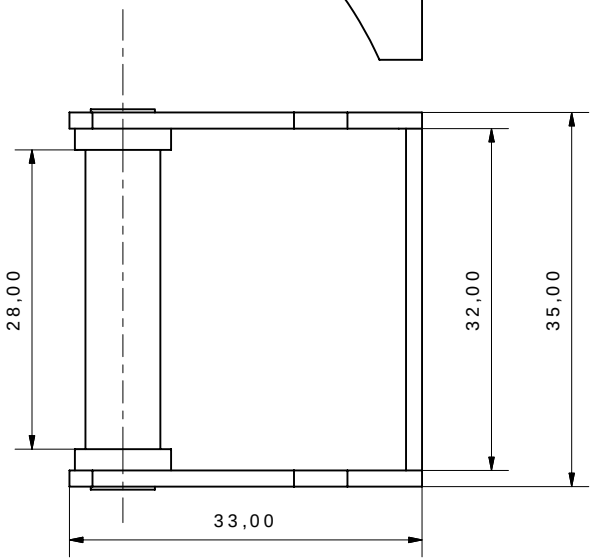


Gambar 1.5 : *Perancangan Manipulator 4 Lengan Robot*





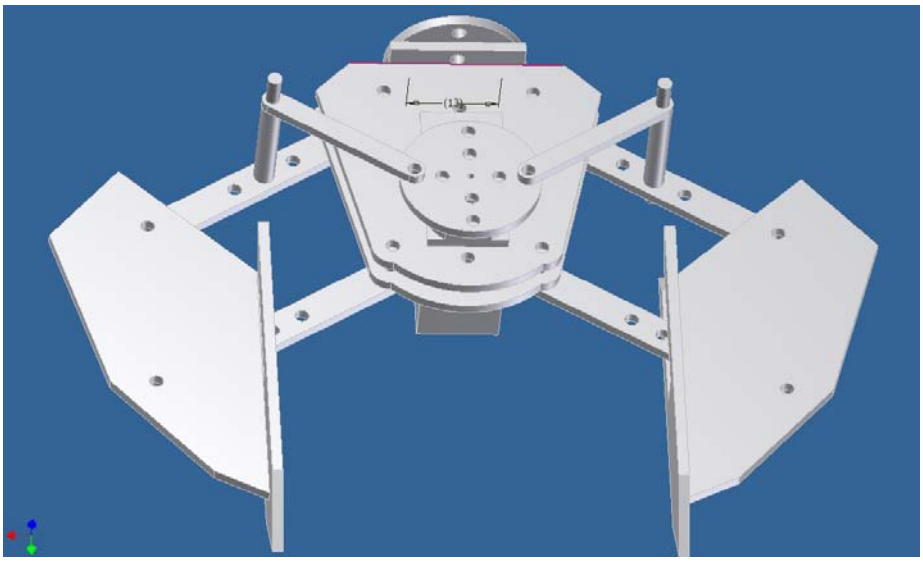


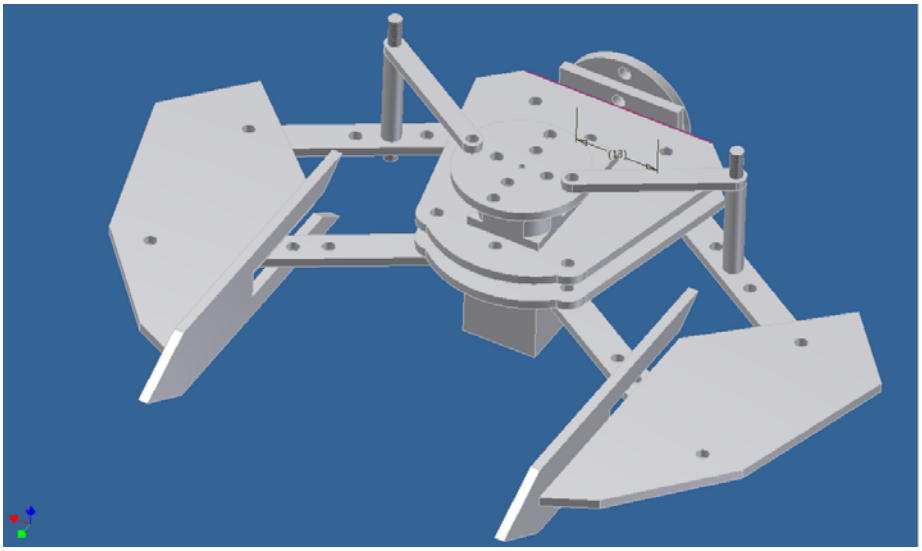


Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/22/2008	

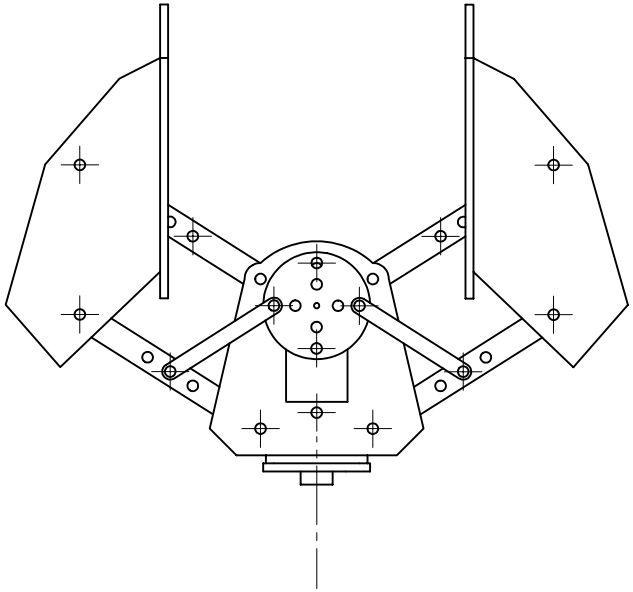
Assembly bushing		Edition	Sheet 1 / 1

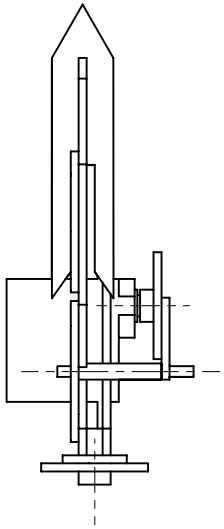
1.1 Gripper

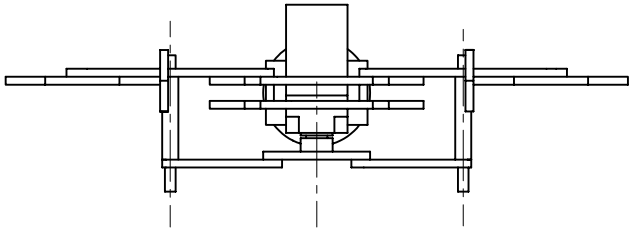


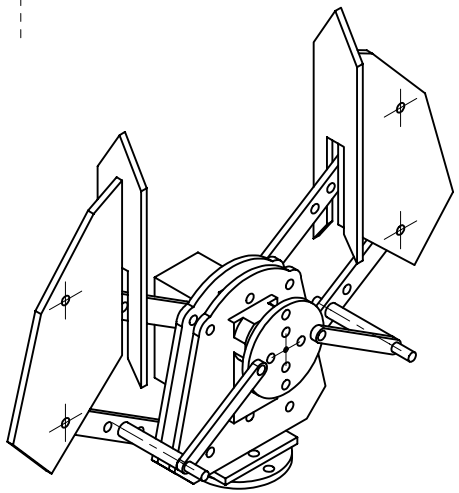


Gambar 1.6 : *Perancangan Gripper Lengan Robot*



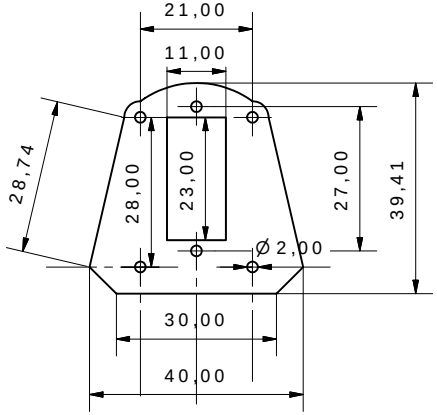


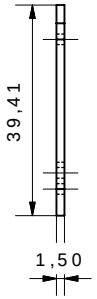


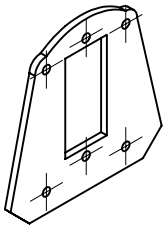


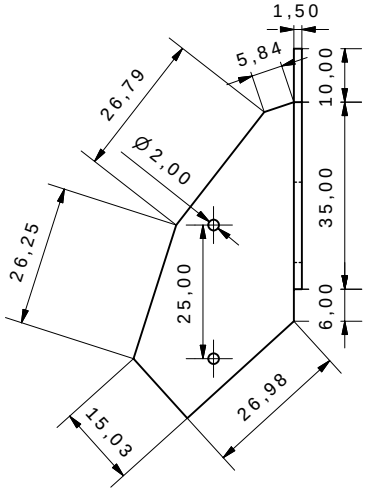
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/21/2008	

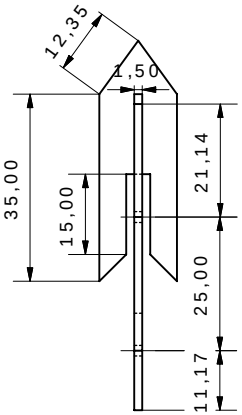
Assembly gripper		Edition	Sheet 1 / 1

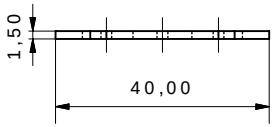


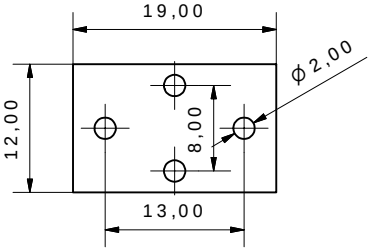


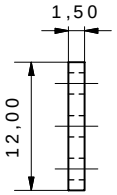


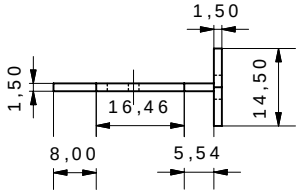


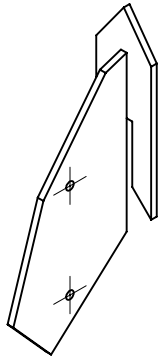


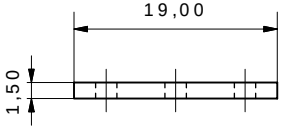


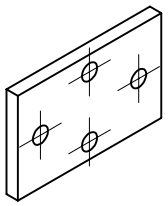






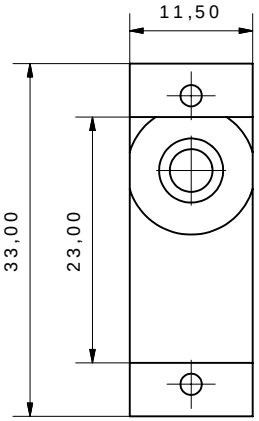


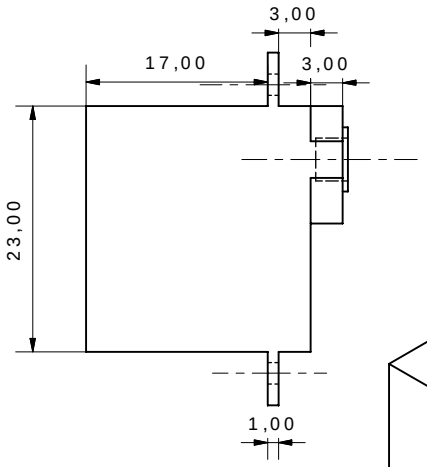


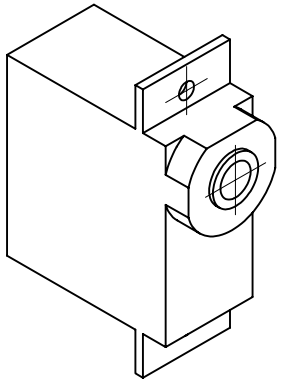


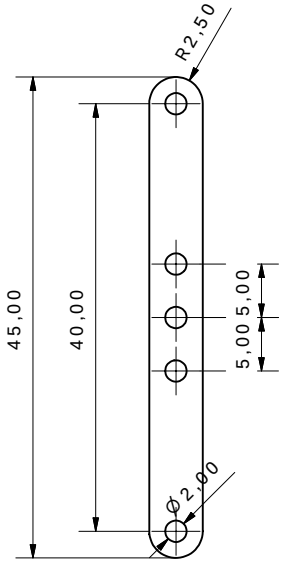
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/21/2008	

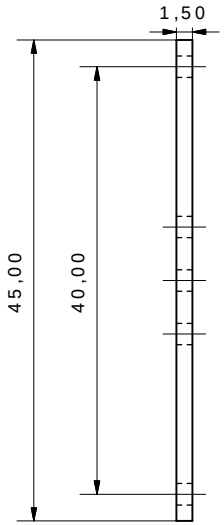
gripper new		Edition	Sheet 1 / 1

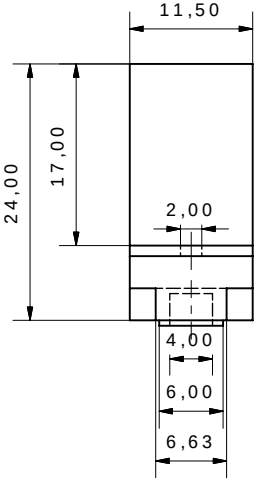


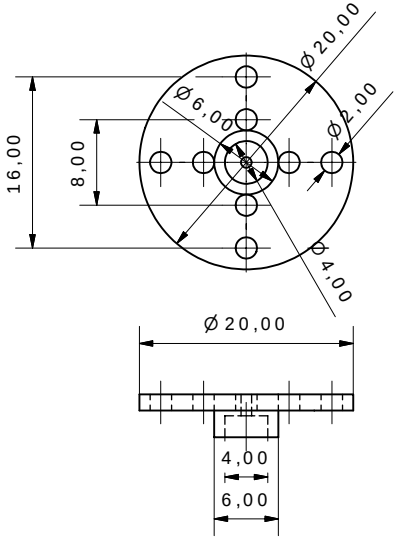


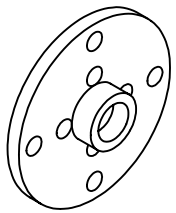


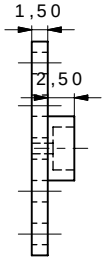


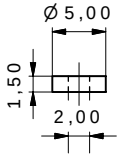


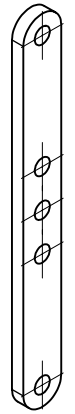








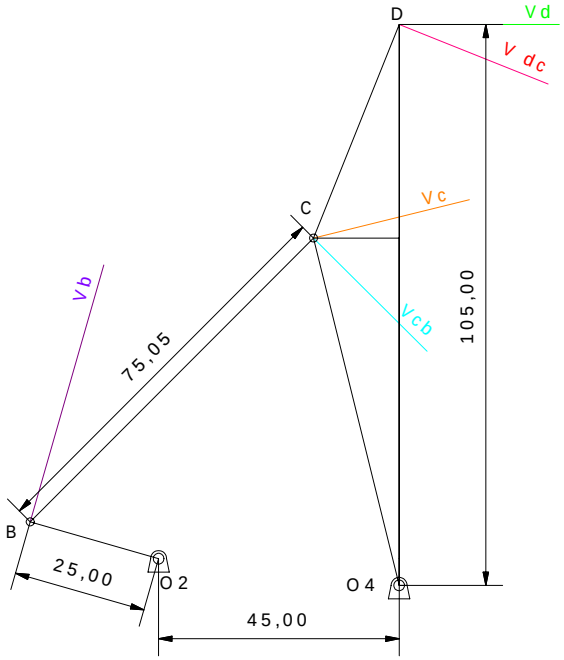




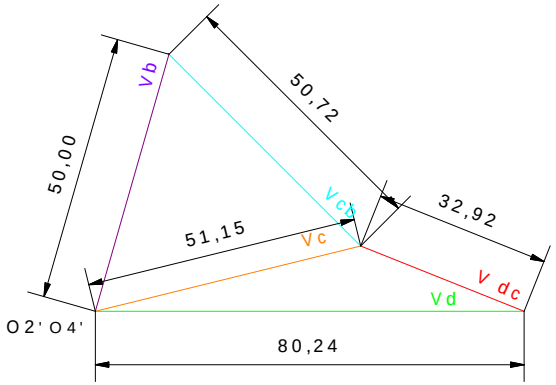
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					12/21/2008	

servo kecil banget2	Edition	Sheet	1 / 1

SERVO 2

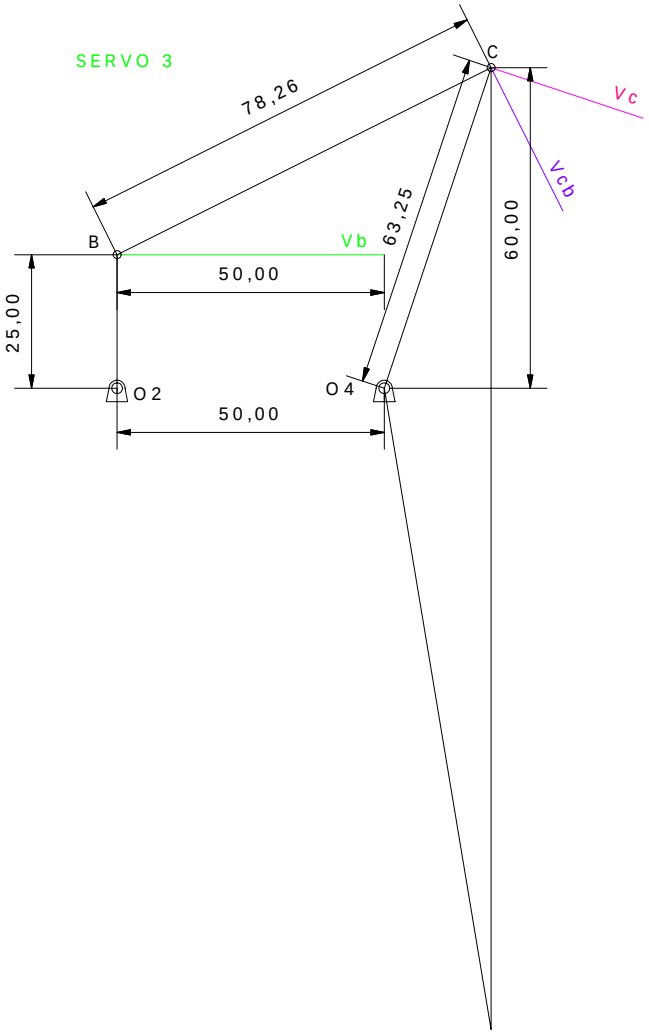


1. Vc = Vb + Vcb
2. Vd = Vc + Vdc

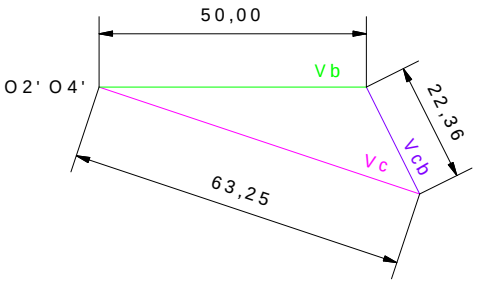


Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					1/5/2009	

Part1 new		Edition	Sheet 1 / 1



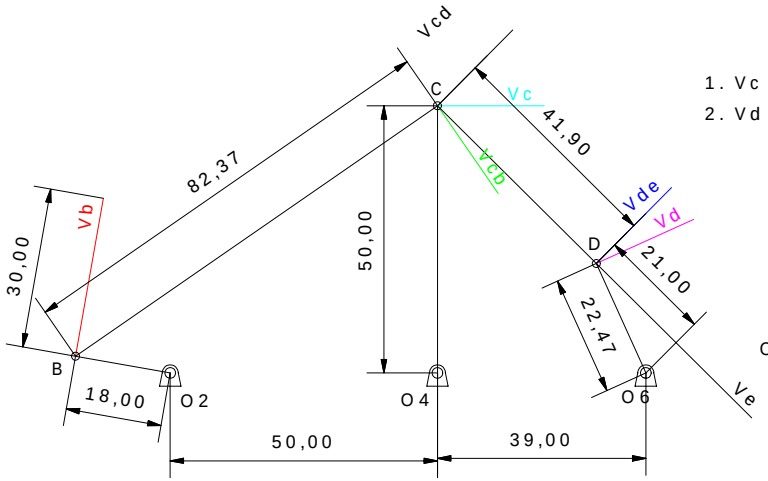
Vc = Vb + Vcb



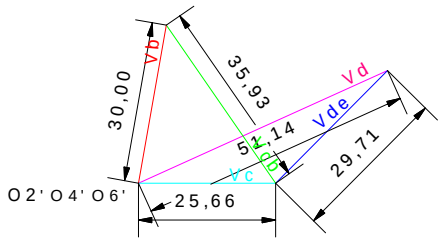
Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					1/5/2009	

Part2 new		Edition	Sheet 1 / 1

SERVO 4



1. Vc = Vb + Vcb
2. Vd = Vc + Vde



Designed by	Checked by	Approved by	Date		Date	
GUSTI HIDAYAT					1/5/2009	

Part3 new 2	Edition	Sheet 1 / 1	

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 32K Bytes of In-System Self-Programmable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 1024 Bytes EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 2K Byte Internal SRAM
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    8 Single-ended Channels
    7 Differential Channels in TQFP Package Only
    2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad MLF
- Operating Voltages
  - 2.7 - 5.5V for ATmega32L
  - 4.5 - 5.5V for ATmega32
- Speed Grades
  - 0 - 8 MHz for ATmega32L
  - 0 - 16 MHz for ATmega32
- Power Consumption at 1 MHz, 3V, 25°C for ATmega32L
  - Active: 1.1 mA
  - Idle Mode: 0.35 mA
  - Power-down Mode: < 1 µA

8-bit AVR® Microcontroller with 32K Bytes In-System Programmable Flash

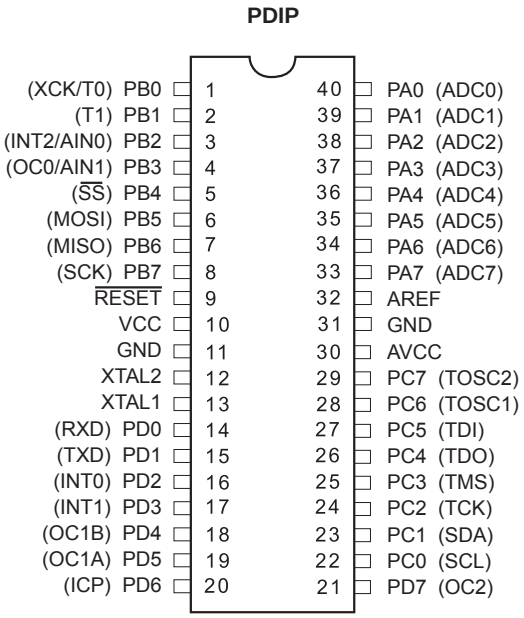
ATmega32
ATmega32L

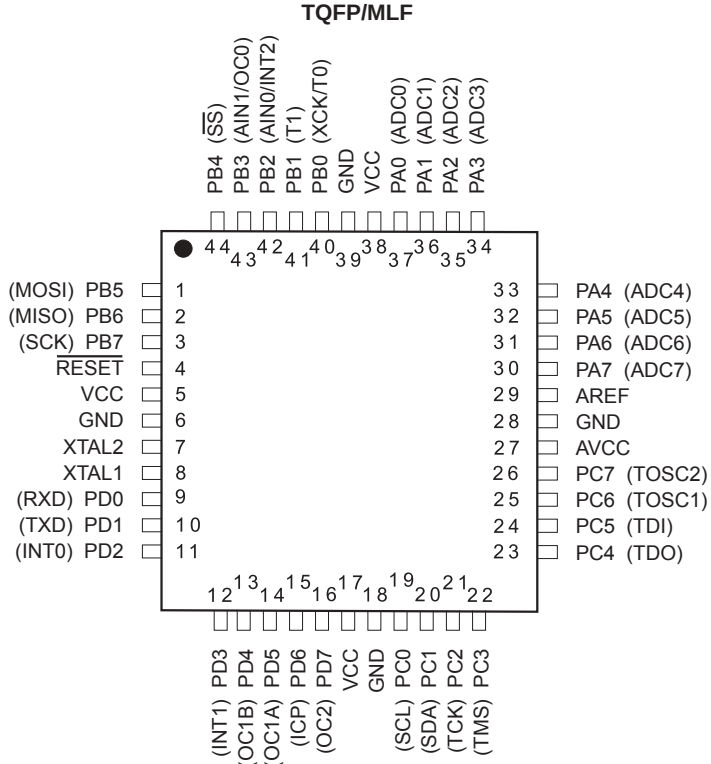
Preliminary

2503F–AVR–12/03

Pin Configurations

Figure 1. Pinouts ATmega32





Disclaimer

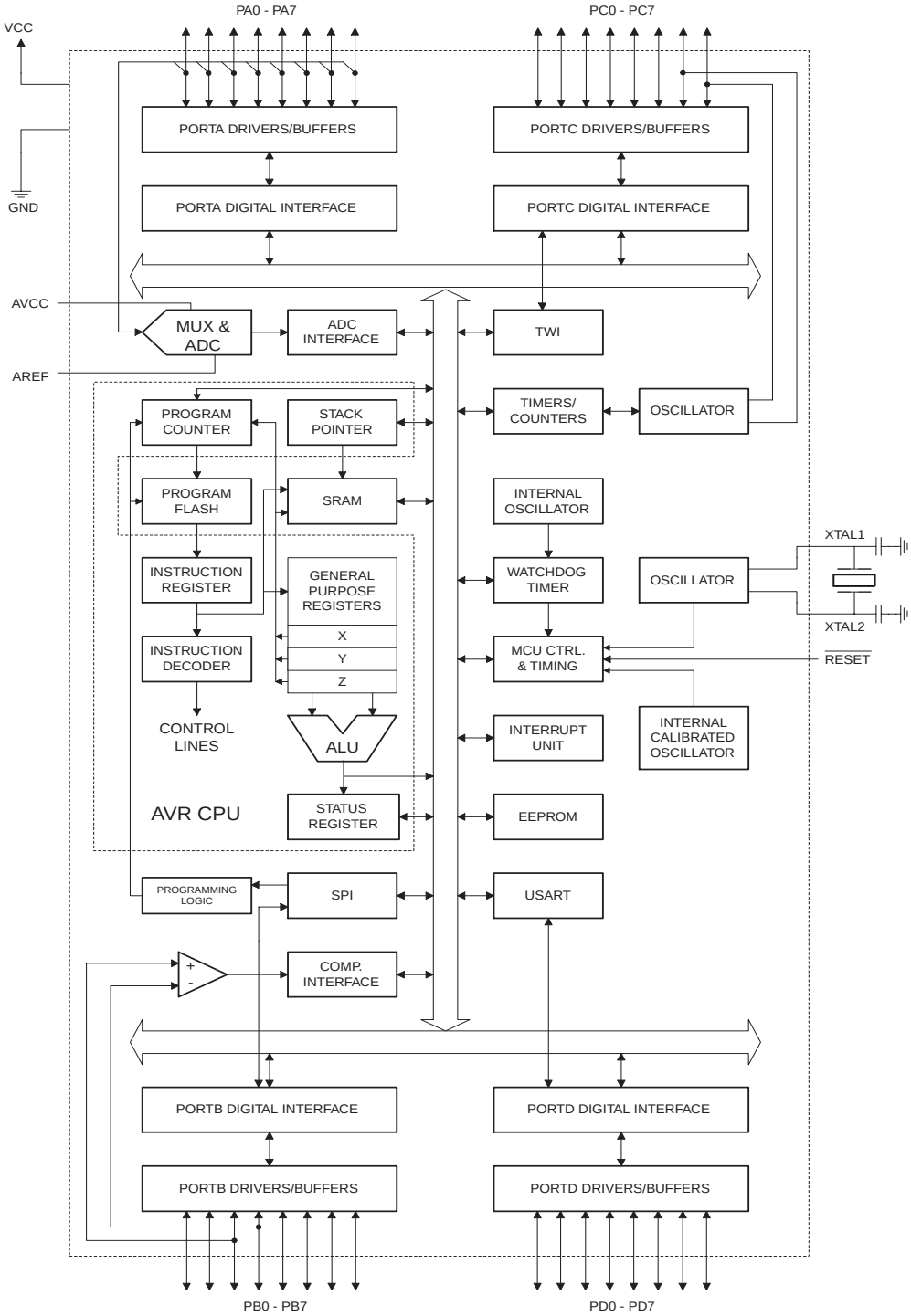
Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

Overview

The ATmega32 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega32 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

Block Diagram

Figure 2. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega32 provides the following features: 32K bytes of In-System Programmable Flash Program memory with Read-While-Write capabilities, 1024 bytes EEPROM, 2K byte SRAM, 32 general purpose I/O lines, 32 general purpose working registers, a JTAG interface for Boundary-scan, On-chip Debugging support and programming, three flexible Timer/Counters with compare modes, Internal and External Interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain (TQFP package only), a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the USART, Two-wire interface, A/D Converter, SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next External Interrupt or Hardware Reset. In Power-save mode, the Asynchronous Timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption. In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run.

The device is manufactured using Atmel's high density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega32 is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many embedded control applications.

The ATmega32 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

Pin Descriptions

VCC

Digital supply voltage.

GND

Ground.

Port A (PA7..PA0)

Port A serves as the analog inputs to the A/D Converter.

Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B (PB7..PB0)

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B also serves the functions of various special features of the ATmega32 as listed on page 55.

Port C (PC7..PC0)

Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pins PC5(TDI), PC3(TMS) and PC2(TCK) will be activated even if a reset occurs.

The TD0 pin is tri-stated unless TAP states that shift out data are entered.

Port C also serves the functions of the JTAG interface and other special features of the ATmega32 as listed on page 58.

Port D (PD7..PD0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port D also serves the functions of various special features of the ATmega32 as listed on page 60.

$\overline{\text{RESET}}$

Reset Input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 15 on page 35. Shorter pulses are not guaranteed to generate a reset.

XTAL1

Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting Oscillator amplifier.

AVCC

AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter.

AREF

AREF is the analog reference pin for the A/D Converter.

About Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C Compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Please confirm with the C Compiler documentation for more details.

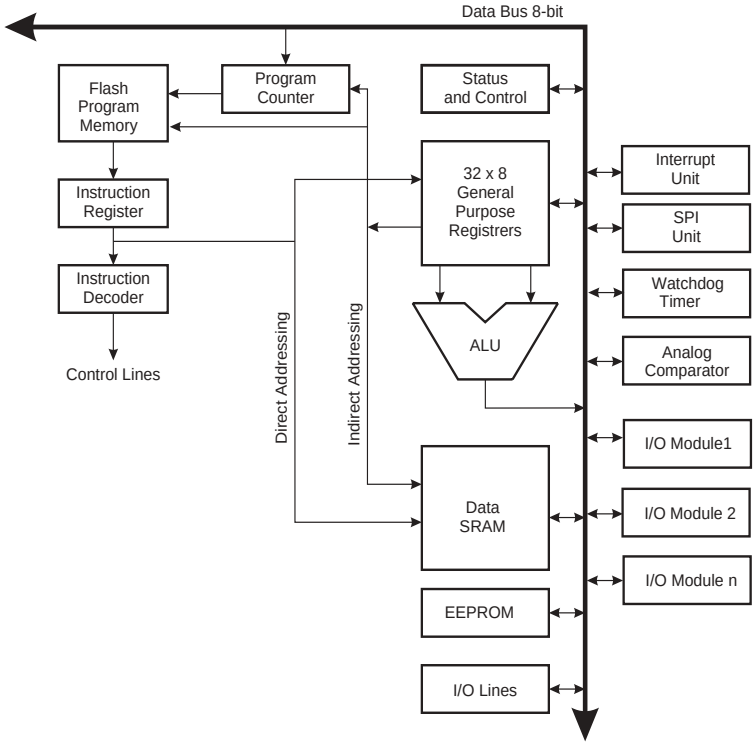
AVR CPU Core

Introduction

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

Architectural Overview

Figure 3. Block Diagram of the AVR MCU Architecture



In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory.

The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing – enabling efficient address calculations. One of the these address pointers can also be used as an address pointer for look up tables in Flash Program memory. These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After

an arithmetic operation, the Status Register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the reset routine (before subroutines or interrupts are executed). The Stack Pointer SP is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional global interrupt enable bit in the Status Register. All interrupts have a separate interrupt vector in the interrupt vector table. The interrupts have priority in accordance with their interrupt vector position. The lower the interrupt vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, $20 - $5F.

ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See the “Instruction Set” section for a detailed description.

Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
	I	**T**	**H**	**S**	**V**	**N**	**Z**	**C**	**SREG**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

- **Bit 6 – T: Bit Copy Storage**

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

- **Bit 5 – H: Half Carry Flag**

The Half Carry Flag H indicates a half carry in some arithmetic operations. Half Carry is useful in BCD arithmetic. See the “Instruction Set Description” for detailed information.

- **Bit 4 – S: Sign Bit, S = N $\oplus$ V**

The S-bit is always an exclusive or between the Negative Flag N and the Two’s Complement Overflow Flag V. See the “Instruction Set Description” for detailed information.

- **Bit 3 – V: Two’s Complement Overflow Flag**

The Two’s Complement Overflow Flag V supports two’s complement arithmetics. See the “Instruction Set Description” for detailed information.

- **Bit 2 – N: Negative Flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 1 – Z: Zero Flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 0 – C: Carry Flag**

The Carry Flag C indicates a carry in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

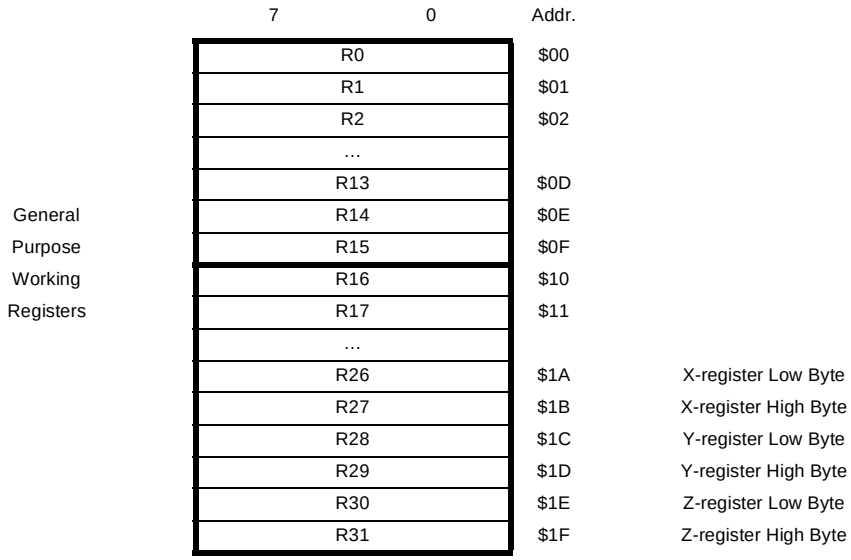
General Purpose Register File

The Register File is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the Register File:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

Figure 4 shows the structure of the 32 general purpose working registers in the CPU.

Figure 4. AVR CPU General Purpose Working Registers



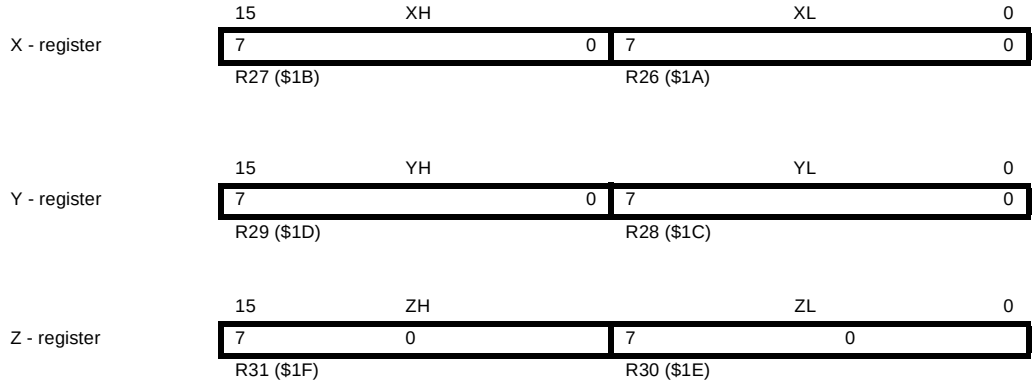
Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

As shown in Figure 4, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y-, and Z-pointer Registers can be set to index any register in the file.

The X-register, Y-register and Z-register

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the Data Space. The three indirect address registers X, Y, and Z are defined as described in Figure 5.

Figure 5. The X-, Y-, and Z-registers



In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the Instruction Set Reference for details).

Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above $60. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

Bit	15	14	13	12	11	10	9	8	
	SP15	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
	SP7	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Instruction Execution Timing

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock clk_{CPU}, directly generated from the selected clock source for the chip. No internal clock division is used.

Figure 6 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

Figure 6. The Parallel Instruction Fetches and Instruction Executions

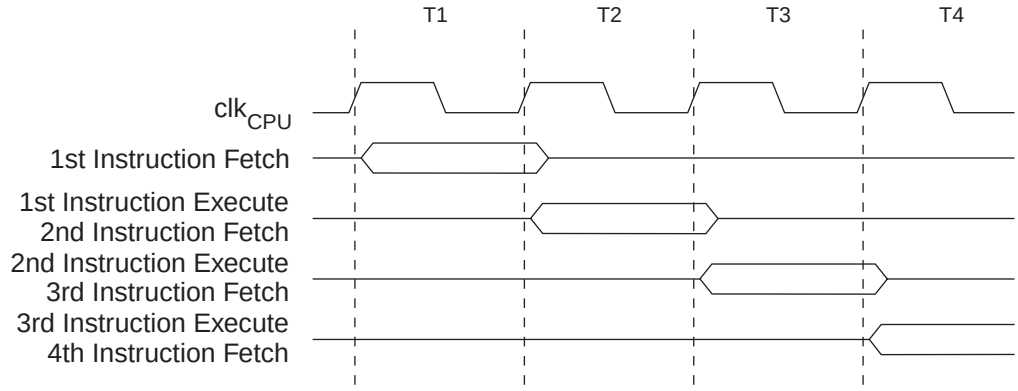
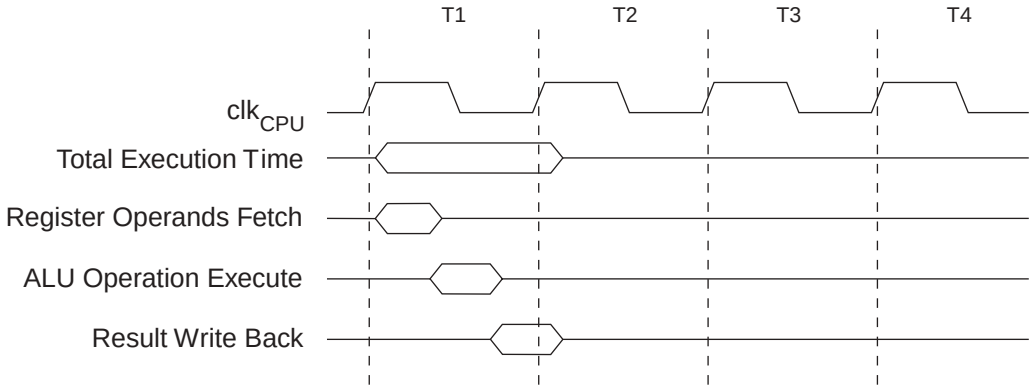


Figure 7 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 7. Single Cycle ALU Operation



Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate reset vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security. See the section “Memory Programming” on page 254 for details.

The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. The complete list of vectors is shown in “Interrupts” on page 42. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0

– the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the General Interrupt Control Register (GICR). Refer to “Interrupts” on page 42 for more information. The Reset Vector can also be moved to the start of the boot Flash section by programming the BOOTRST fuse, see “Boot Loader Support – Read-While-Write Self-Programming” on page 242.

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the global interrupt enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Note that the Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Assembly Code Example

```
in  r16, SREG      ; store SREG value
cli     ; disable interrupts during timed sequence
sbi EECR, EEMWE    ; start EEPROM write
sbi EECR, EEWE
out SREG, r16      ; restore SREG value (I-bit)
```

C Code Example

```
char cSREG;
cSREG = SREG;  /* store SREG value */
/* disable interrupts during timed sequence */
_CLI();
EECR |= (1<<EEMWE); /* start EEPROM write */
EECR |= (1<<EEWE);
SREG = cSREG; /* restore SREG value (I-bit) */
```

When using the SEI instruction to enable interrupts, the instruction following SEI will be executed before any pending interrupts, as shown in this example.

Assembly Code Example

```
sei   ; set global interrupt enable
sleep ; enter sleep, waiting for interrupt
; note: will enter sleep before any pending
; interrupt(s)
```

C Code Example

```
_SEI(); /* set global interrupt enable */
_SLEEP(); /* enter sleep, waiting for interrupt */
/* note: will enter sleep before any pending interrupt(s) */
```

Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum. After four clock cycles the program vector address for the actual interrupt handling routine is executed. During this four clock cycle period, the Program Counter is pushed onto the Stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes four clock cycles. During these four clock cycles, the Program Counter (two bytes) is popped back from the Stack, the Stack Pointer is incremented by two, and the I-bit in SREG is set.

AVR ATmega32 Memories

This section describes the different memories in the ATmega32. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the ATmega32 features an EEPROM Memory for data storage. All three memory spaces are linear and regular.

In-System Reprogrammable Flash Program Memory

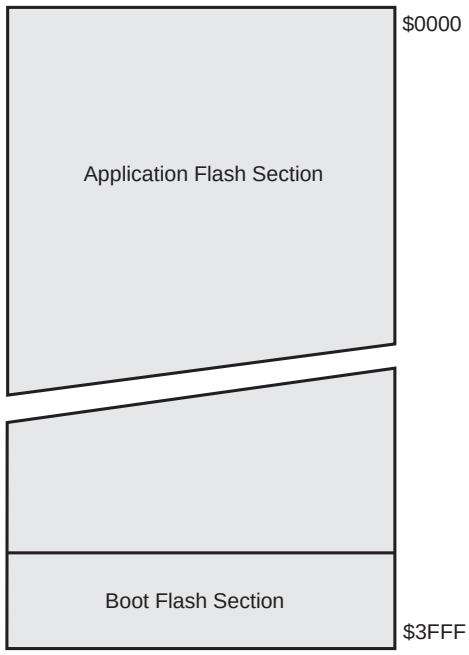
The ATmega32 contains 32K bytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 16K x 16. For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section.

The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega32 Program Counter (PC) is 14 bits wide, thus addressing the 16K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in “Boot Loader Support – Read-While-Write Self-Programming” on page 242. “Memory Programming” on page 254 contains a detailed description on Flash Programming in SPI, JTAG, or Parallell Programming mode.

Constant tables can be allocated within the entire program memory address space (see the LPM – Load Program Memory Instruction Description).

Timing diagrams for instruction fetch and execution are presented in “Instruction Execution Timing” on page 11.

Figure 8. Program Memory Map



SRAM Data Memory

Figure 9 shows how the ATmega32 SRAM Memory is organized.

The lower 2144 Data Memory locations address the Register File, the I/O Memory, and the internal data SRAM. The first 96 locations address the Register File and I/O Memory, and the next 2048 locations address the internal data SRAM.

The five different addressing modes for the data memory cover: Direct, Indirect with Displacement, Indirect, Indirect with Pre-decrement, and Indirect with Post-increment. In the Register File, registers R26 to R31 feature the indirect Addressing Pointer Registers.

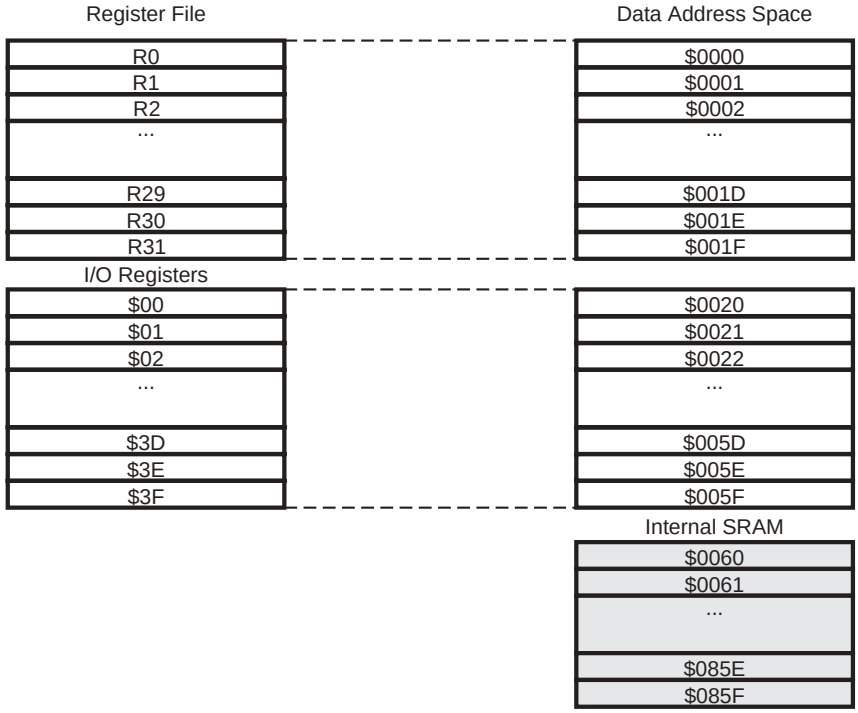
The direct addressing reaches the entire data space.

The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.

When using register indirect addressing modes with automatic pre-decrement and post-increment, the address registers X, Y, and Z are decremented or incremented.

The 32 general purpose working registers, 64 I/O Registers, and the 2048 bytes of internal data SRAM in the ATmega32 are all accessible through all these addressing modes. The Register File is described in “General Purpose Register File” on page 9.

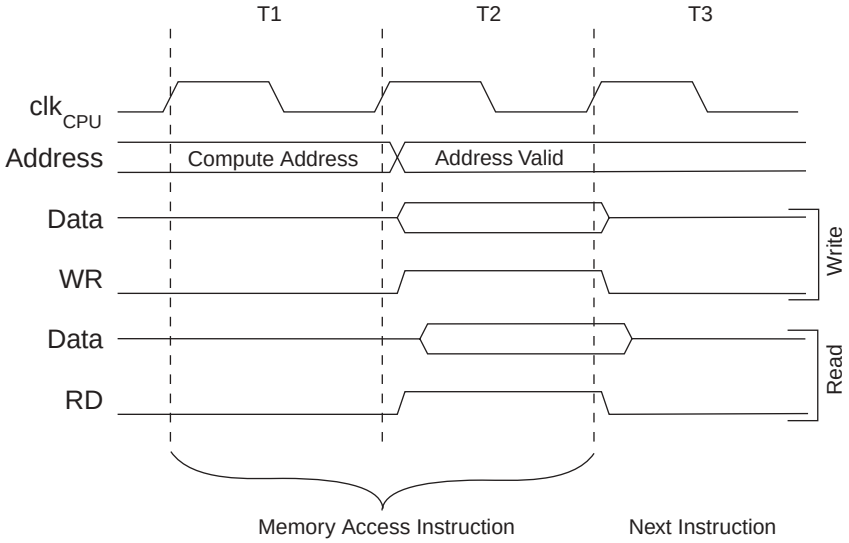
Figure 9. Data Memory Map



Data Memory Access Times

This section describes the general access timing concepts for internal memory access. The internal data SRAM access is performed in two clk_{CPU} cycles as described in Figure 10.

Figure 10. On-chip Data SRAM Access Cycles



EEPROM Data Memory

The ATmega32 contains 1024 bytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

"Memory Programming" on page 254 contains a detailed description on EEPROM Programming in SPI, JTAG, or Parallell Programming mode.

EEPROM Read/Write Access

The EEPROM Access Registers are accessible in the I/O space.

The write access time for the EEPROM is given in Table 1. A self-timing function, however, lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies, V_{CC} is likely to rise or fall slowly on Power-up/down. This causes the device for some period of time to run at a voltage lower than specified as minimum for the clock frequency used. See "Preventing EEPROM Corruption" on page 20 for details on how to avoid problems in these situations.

In order to prevent unintentional EEPROM writes, a specific write procedure must be followed. Refer to the description of the EEPROM Control Register for details on this.

When the EEPROM is read, the CPU is halted for four clock cycles before the next instruction is executed. When the EEPROM is written, the CPU is halted for two clock cycles before the next instruction is executed.

The EEPROM Address Register – EEARH and EEARL

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	EEAR9	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	X	
	X	X	X	X	X	X	X	X	

- **Bits 15..10 – Res: Reserved Bits**

These bits are reserved bits in the ATmega32 and will always read as zero.

- **Bits 9..0 – EEAR9..0: EEPROM Address**

The EEPROM Address Registers – EEARH and EEARL – specify the EEPROM address in the 1024 bytes EEPROM space. The EEPROM data bytes are addressed linearly between 0 and 1023. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

The EEPROM Data Register – EEDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..0 – EEDR7.0: EEPROM Data**

For the EEPROM write operation, the EEDR Register contains the data to be written to the EEPROM in the address given by the EEAR Register. For the EEPROM read operation, the EEDR contains the data read out from the EEPROM at the address given by EEAR.

The EEPROM Control Register – EECR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	X	0	

- **Bits 7..4 – Res: Reserved Bits**

These bits are reserved bits in the ATmega32 and will always read as zero.

- **Bit 3 – EERIE: EEPROM Ready Interrupt Enable**

Writing EERIE to one enables the EEPROM Ready Interrupt if the I bit in SREG is set. Writing EERIE to zero disables the interrupt. The EEPROM Ready interrupt generates a constant interrupt when EEWE is cleared.

- **Bit 2 – EEMWE: EEPROM Master Write Enable**

The EEMWE bit determines whether setting EEWE to one causes the EEPROM to be written. When EEMWE is set, setting EEWE within four clock cycles will write data to the EEPROM at the selected address If EEMWE is zero, setting EEWE will have no effect.

When EEMWE has been written to one by software, hardware clears the bit to zero after four clock cycles. See the description of the EEWE bit for an EEPROM write procedure.

- **Bit 1 – EEWE: EEPROM Write Enable**

The EEPROM Write Enable Signal EEWE is the write strobe to the EEPROM. When address and data are correctly set up, the EEWE bit must be written to one to write the value into the EEPROM. The EEMWE bit must be written to one before a logical one is written to EEWE, otherwise no EEPROM write takes place. The following procedure should be followed when writing the EEPROM (the order of steps 3 and 4 is not essential):

1. Wait until EEWE becomes zero.
2. Wait until SPMEN in SPMCR becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a logical one to the EEMWE bit while writing a zero to EEWE in EECR.
6. Within four clock cycles after setting EEMWE, write a logical one to EEWE.

The EEPROM can not be programmed during a CPU write to the Flash memory. The software must check that the Flash programming is completed before initiating a new EEPROM write. Step 2 is only relevant if the software contains a Boot Loader allowing the CPU to program the Flash. If the Flash is never being updated by the CPU, step 2 can be omitted. See “Boot Loader Support – Read-While-Write Self-Programming” on page 242 for details about boot programming.

Caution: An interrupt between step 5 and step 6 will make the write cycle fail, since the EEPROM Master Write Enable will time-out. If an interrupt routine accessing the EEPROM is interrupting another EEPROM Access, the EEAR or EEDR reGister will be modified, causing the interrupted EEPROM Access to fail. It is recommended to have the Global Interrupt Flag cleared during all the steps to avoid these problems.

When the write access time has elapsed, the EEWE bit is cleared by hardware. The user software can poll this bit and wait for a zero before writing the next byte. When EEWE has been set, the CPU is halted for two cycles before the next instruction is executed.

- **Bit 0 – EERE: EEPROM Read Enable**

The EEPROM Read Enable Signal – EERE – is the read strobe to the EEPROM. When the correct address is set up in the EEAR Register, the EERE bit must be written to a logic one to trigger the EEPROM read. The EEPROM read access takes one instruction, and the requested data is available immediately. When the EEPROM is read, the CPU is halted for four cycles before the next instruction is executed.

The user should poll the EEWE bit before starting the read operation. If a write operation is in progress, it is neither possible to read the EEPROM, nor to change the EEAR Register.

The calibrated Oscillator is used to time the EEPROM accesses. Table 1 lists the typical programming time for EEPROM access from the CPU.

Table 1. EEPROM Programming Time

Symbol	Number of Calibrated RC Oscillator Cycles[1]	Typ Programming Time
EEPROM write (from CPU)	8448	8.5 ms

Note: 1. Uses 1 MHz clock, independent of CKSEL Fuse setting.

The following code examples show one assembly and one C function for writing to the EEPROM. The examples assume that interrupts are controlled (for example by disabling interrupts globally) so that no interrupts will occur during execution of these functions. The examples also assume that no Flash Boot Loader is present in the software. If such code is present, the EEPROM write function must also wait for any ongoing SPM command to finish.

Assembly Code Example

```
EEPROM_write:
  ; Wait for completion of previous write
  sbic EECR,EEWE
  rjmp EEPROM_write
  ; Set up address (r18:r17) in address register
  out  EEARH, r18
  out  EEARL, r17
  ; Write data (r16) to data register
  out  EEDR,r16
  ; Write logical one to EEMWE
  sbi  EECR,EEMWE
  ; Start eeprom write by setting EEWE
  sbi  EECR,EEWE
  ret
```

C Code Example

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
  /* Wait for completion of previous write */
  while(EECR & (1<<EEWE))
    ;
  /* Set up address and data registers */
  EEAR = uiAddress;
  EEDR = ucData;
  /* Write logical one to EEMWE */
  EECR |= (1<<EEMWE);
  /* Start eeprom write by setting EEWE */
  EECR |= (1<<EEWE);
}
```

The next code examples show assembly and C functions for reading the EEPROM. The examples assume that interrupts are controlled so that no interrupts will occur during execution of these functions.

Assembly Code Example

```
EEPROM_read:
  ; Wait for completion of previous write
  sbic EECR,EEWE
  rjmp EEPROM_read
  ; Set up address (r18:r17) in address register
  out  EEARH, r18
  out  EEARL, r17
  ; Start eeprom read by writing EERE
  sbi  EECR,EERE
  ; Read data from data register
  in   r16,EEDR
  ret
```

C Code Example

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
  /* Wait for completion of previous write */
  while(EECR & (1<<EEWE))
    ;
  /* Set up address register */
  EEAR = uiAddress;
  /* Start eeprom read by writing EERE */
  EECR |= (1<<EERE);
  /* Return data from data register */
  return EEDR;
}
```

EEPROM Write During Power-down Sleep Mode

When entering Power-down Sleep mode while an EEPROM write operation is active, the EEPROM write operation will continue, and will complete before the Write Access time has passed. However, when the write operation is completed, the crystal Oscillator continues running, and as a consequence, the device does not enter Power-down entirely. It is therefore recommended to verify that the EEPROM write operation is completed before entering Power-down.

Preventing EEPROM Corruption

During periods of low $V_{CC,}$ the EEPROM data can be corrupted because the supply voltage is too low for the CPU and the EEPROM to operate properly. These issues are the same as for board level systems using EEPROM, and the same design solutions should be applied.

An EEPROM data corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the EEPROM requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage is too low.

EEPROM data corruption can easily be avoided by following this design recommendation:

Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD). If the detection level of the internal BOD does not match the needed detection level, an external low V_{CC} Reset Protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.

I/O Memory

The I/O space definition of the ATmega32 is shown in “Register Summary” on page 299.

All ATmega32 I/Os and peripherals are placed in the I/O space. The I/O locations are accessed by the IN and OUT instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O Registers within the address range $00 - $1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions. Refer to the Instruction Set section for more details. When using the I/O specific commands IN and OUT, the I/O addresses $00 - $3F must be used. When addressing I/O Registers as data space using LD and ST instructions, $20 must be added to these addresses.

For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.

Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers $00 to $1F only.

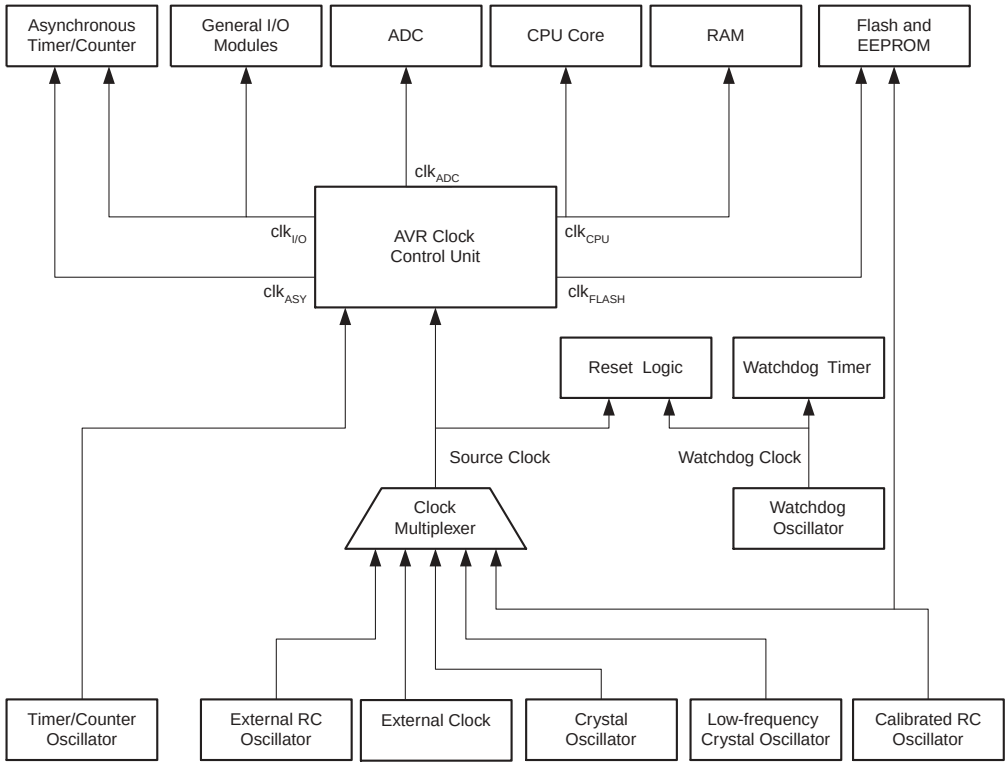
The I/O and Peripherals Control Registers are explained in later sections.

System Clock and Clock Options

Clock Systems and their Distribution

Figure 11 presents the principal clock systems in the AVR and their distribution. All of the clocks need not be active at a given time. In order to reduce power consumption, the clocks to modules not being used can be halted by using different sleep modes, as described in “Power Management and Sleep Modes” on page 30. The clock systems are detailed Figure 11.

Figure 11. Clock Distribution



CPU Clock – clk_{CPU}

The CPU clock is routed to parts of the system concerned with operation of the AVR core. Examples of such modules are the General Purpose Register File, the Status Register and the data memory holding the Stack Pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

I/O Clock – $clk_{I/O}$

The I/O clock is used by the majority of the I/O modules, like Timer/Counters, SPI, and USART. The I/O clock is also used by the External Interrupt module, but note that some external interrupts are detected by asynchronous logic, allowing such interrupts to be detected even if the I/O clock is halted. Also note that address recognition in the TWI module is carried out asynchronously when $clk_{I/O}$ is halted, enabling TWI address reception in all sleep modes.

Flash Clock – clk_{FLASH}

The Flash clock controls operation of the Flash interface. The Flash clock is usually active simultaneously with the CPU clock.

Asynchronous Timer Clock – clk_{ASY}

The Asynchronous Timer clock allows the Asynchronous Timer/Counter to be clocked directly from an external 32 kHz clock crystal. The dedicated clock domain allows using this Timer/Counter as a real-time counter even when the device is in sleep mode.

ADC Clock – clk_{ADC}

The ADC is provided with a dedicated clock domain. This allows halting the CPU and I/O clocks in order to reduce noise generated by digital circuitry. This gives more accurate ADC conversion results.

Clock Sources

The device has the following clock source options, selectable by Flash Fuse bits as shown below. The clock from the selected source is input to the AVR clock generator, and routed to the appropriate modules.

Table 2. Device Clocking Options Select(1)

Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

Note: 1. For all fuses “1” means unprogrammed while “0” means programmed.

The various choices for each clocking option is given in the following sections. When the CPU wakes up from Power-down or Power-save, the selected clock source is used to time the start-up, ensuring stable Oscillator operation before instruction execution starts. When the CPU starts from Reset, there is as an additional delay allowing the power to reach a stable level before commencing normal operation. The Watchdog Oscillator is used for timing this real-time part of the start-up time. The number of WDT Oscillator cycles used for each time-out is shown in Table 3. The frequency of the Watchdog Oscillator is voltage dependent as shown in “Register Summary” on page 299.

Table 3. Number of Watchdog Oscillator Cycles

Typ Time-out (V_{CC} = 5.0V)	Typ Time-out (V_{CC} = 3.0V)	Number of Cycles
4.1 ms	4.3 ms	4K (4,096)
65 ms	69 ms	64K (65,536)

Default Clock Source

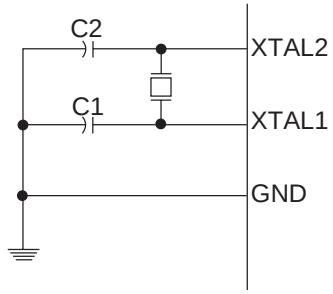
The device is shipped with CKSEL = “0001” and SUT = “10”. The default clock source setting is therefore the 1 MHz Internal RC Oscillator with longest startup time. This default setting ensures that all users can make their desired clock source setting using an In-System or Parallel Programmer.

Crystal Oscillator

XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator, as shown in Figure 12. Either a quartz crystal or a ceramic resonator may be used. The CKOPT Fuse selects between two different Oscillator amplifier modes. When CKOPT is programmed, the Oscillator output will oscillate will a full rail-to-rail swing on the output. This mode is suitable when operating in a very noisy environment or when the output from XTAL2 drives a second clock buffer. This mode has a wide frequency range. When CKOPT is unprogrammed, the Oscillator has a smaller output swing. This reduces power consumption considerably. This mode has a limited frequency range and it can not be used to drive other clock buffers.

For resonators, the maximum frequency is 8 MHz with CKOPT unprogrammed and 16 MHz with CKOPT programmed. C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Some initial guidelines for choosing capacitors for use with crystals are given in Table 4. For ceramic resonators, the capacitor values given by the manufacturer should be used.

Figure 12. Crystal Oscillator Connections



The Oscillator can operate in three different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL3..1 as shown in Table 4.

Table 4. Crystal Oscillator Operating Modes

CKOPT	CKSEL3..1	Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
1	101[1]	0.4 - 0.9	–
1	110	0.9 - 3.0	12 - 22
1	111	3.0 - 8.0	12 - 22
0	101, 110, 111	1.0 ≤	12 - 22

Note: 1. This option should not be used with crystals, only with ceramic resonators.

The CKSEL0 Fuse together with the SUT1..0 fuses select the start-up times as shown in Table 5.

Table 5. Start-up Times for the Crystal Oscillator Clock Selection

CKSEL0	SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
0	00	258 CK[1]	4.1 ms	Ceramic resonator, fast rising power
0	01	258 CK[1]	65 ms	Ceramic resonator, slowly rising power
0	10	1K CK[2]	–	Ceramic resonator, BOD enabled
0	11	1K CK[2]	4.1 ms	Ceramic resonator, fast rising power
1	00	1K CK[2]	65 ms	Ceramic resonator, slowly rising power
1	01	16K CK	–	Crystal Oscillator, BOD enabled
1	10	16K CK	4.1 ms	Crystal Oscillator, fast rising power
1	11	16K CK	65 ms	Crystal Oscillator, slowly rising power

Notes:
1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device, and if frequency stability at start-up is not important for the application.

Low-frequency Crystal Oscillator

To use a 32.768 kHz watch crystal as the clock source for the device, the Low-frequency Crystal Oscillator must be selected by setting the CKSEL fuses to “1001”. The crystal should be connected as shown in Figure 12. By programming the CKOPT Fuse, the user can enable internal capacitors on XTAL1 and XTAL2, thereby removing the need for external capacitors. The internal capacitors have a nominal value of 36 pF.

When this Oscillator is selected, start-up times are determined by the SUT fuses as shown in Table 6.

Table 6. Start-up Times for the Low-frequency Crystal Oscillator Clock Selection

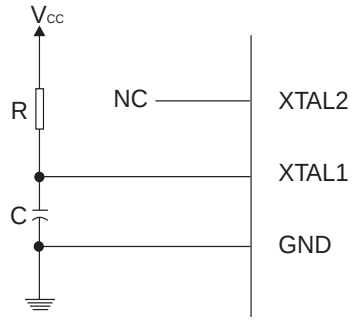
SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	1K CK(1)	4.1 ms	Fast rising power or BOD enabled
01	1K CK(1)	65 ms	Slowly rising power
10	32K CK	65 ms	Stable frequency at start-up
11	Reserved		

Note: 1. These options should only be used if frequency stability at start-up is not important for the application.

External RC Oscillator

For timing insensitive applications, the external RC configuration shown in Figure 13 can be used. The frequency is roughly estimated by the equation f = 1/(3RC). C should be at least 22 pF. By programming the CKOPT Fuse, the user can enable an internal 36 pF capacitor between XTAL1 and GND, thereby removing the need for an external capacitor. For more information on Oscillator operation and details on how to choose R and C, refer to the External RC Oscillator application note.

Figure 13. External RC Configuration



The Oscillator can operate in four different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL3..0 as shown in Table 7.

Table 7. External RC Oscillator Operating Modes

CKSEL3..0	Frequency Range (MHz)
0101	≤ 0.9
0110	0.9 - 3.0
0111	3.0 - 8.0
1000	8.0 - 12.0

When this Oscillator is selected, start-up times are determined by the SUT fuses as shown in Table 8.

Table 8. Start-up Times for the External RC Oscillator Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	18 CK	–	BOD enabled
01	18 CK	4.1 ms	Fast rising power
10	18 CK	65 ms	Slowly rising power
11	6 CK(1)	4.1 ms	Fast rising power or BOD enabled

Note: 1. This option should not be used when operating close to the maximum frequency of the device.

Calibrated Internal RC Oscillator

The Calibrated Internal RC Oscillator provides a fixed 1.0, 2.0, 4.0, or 8.0 MHz clock. All frequencies are nominal values at 5V and 25°C. This clock may be selected as the system clock by programming the CKSEL fuses as shown in Table 9. If selected, it will operate with no external components. The CKOPT Fuse should always be unprogrammed when using this clock option. During Reset, hardware loads the calibration byte into the OSCCAL Register and thereby automatically calibrates the RC Oscillator. At 5V, 25°C and 1.0 MHz Oscillator frequency selected, this calibration gives a frequency within ± 3% of the nominal frequency. Using run-time calibration methods as described in application notes available at www.atmel.com/avr it is possible to achieve ± 1% accuracy at any given V_{CC} and Temperature. When this Oscillator is used as the Chip Clock, the Watchdog Oscillator will still be used for the Watchdog Timer and for the reset time-out. For more information on the pre-programmed calibration value, see the section “Calibration Byte” on page 256.

Table 9. Internal Calibrated RC Oscillator Operating Modes

CKSEL3..0	Nominal Frequency (MHz)
0001(1)	1.0
0010	2.0
0011	4.0
0100	8.0

Note: 1. The device is shipped with this option selected.

When this Oscillator is selected, start-up times are determined by the SUT fuses as shown in Table 10. XTAL1 and XTAL2 should be left unconnected (NC).

Table 10. Start-up Times for the Internal Calibrated RC Oscillator Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10(1)	6 CK	65 ms	Slowly rising power
11	Reserved		

Note: 1. The device is shipped with this option selected.

Oscillator Calibration Register – OSCCAL

Bit	7	6	5	4	3	2	1	0	
	CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	OSCCAL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	Device Specific Calibration Value								

- **Bits 7..0 – CAL7..0: Oscillator Calibration Value**

Writing the calibration byte to this address will trim the Internal Oscillator to remove process variations from the Oscillator frequency. During Reset, the 1 MHz calibration value which is located in the signature row High Byte (address 0x00) is automatically loaded into the OSCCAL Register. If the internal RC is used at other frequencies, the calibration values must be loaded manually. This can be done by first reading the signature row by a programmer, and then store the calibration values in the Flash or EEPROM. Then the value can be read by software and loaded into the OSCCAL Register. When OSCCAL is zero, the lowest available frequency is chosen. Writing non-zero values to this register will increase the frequency of the Internal Oscillator. Writing $FF to the register gives the highest available frequency. The calibrated Oscillator is used to time EEPROM and Flash access. If EEPROM or Flash is written, do not calibrate to more than 10% above the nominal frequency. Otherwise, the EEPROM or Flash write may fail. Note that the Oscillator is intended for calibration to 1.0, 2.0, 4.0, or 8.0 MHz. Tuning to other values is not guaranteed, as indicated in Table 11.

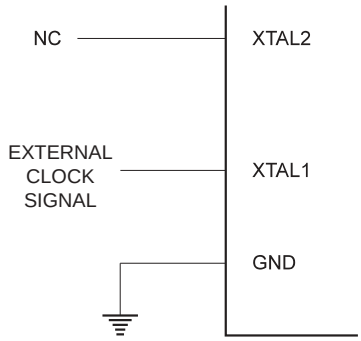
Table 11. Internal RC Oscillator Frequency Range.

OSCCAL Value	Min Frequency in Percentage of Nominal Frequency (%)	Max Frequency in Percentage of Nominal Frequency (%)
$00	50	100
$7F	75	150
$FF	100	200

External Clock

To drive the device from an external clock source, XTAL1 should be driven as shown in Figure 14. To run the device on an external clock, the CKSEL fuses must be programmed to “0000”. By programming the CKOPT Fuse, the user can enable an internal 36 pF capacitor between XTAL1 and GND.

Figure 14. External Clock Drive Configuration



When this clock source is selected, start-up times are determined by the SUT fuses as shown in Table 12.

Table 12. Start-up Times for the External Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10	6 CK	65 ms	Slowly rising power
11	Reserved		

When applying an external clock, it is required to avoid sudden changes in the applied clock frequency to ensure stable operation of the MCU. A variation in frequency of more than 2% from one clock cycle to the next can lead to unpredictable behavior. It is required to ensure that the MCU is kept in reset during such changes in the clock frequency.

Timer/Counter Oscillator

For AVR microcontrollers with Timer/Counter Oscillator pins (TOSC1 and TOSC2), the crystal is connected directly between the pins. No external capacitors are needed. The Oscillator is optimized for use with a 32.768 kHz watch crystal. Applying an external clock source to TOSC1 is not recommended.

Power Management and Sleep Modes

Sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application's requirements.

To enter any of the six sleep modes, the SE bit in MCUCR must be written to logic one and a SLEEP instruction must be executed. The SM2, SM1, and SM0 bits in the MCUCR Register select which sleep mode (Idle, ADC Noise Reduction, Power-down, Power-save, Standby, or Extended Standby) will be activated by the SLEEP instruction. See Table 13 for a summary. If an enabled interrupt occurs while the MCU is in a sleep mode, the MCU wakes up. The MCU is then halted for four cycles in addition to the start-up time, it executes the interrupt routine, and resumes execution from the instruction following SLEEP. The contents of the Register File and SRAM are unaltered when the device wakes up from sleep. If a Reset occurs during sleep mode, the MCU wakes up and executes from the Reset Vector.

Figure 11 on page 22 presents the different clock systems in the ATmega32, and their distribution. The figure is helpful in selecting an appropriate sleep mode.

MCU Control Register – MCUCR

The MCU Control Register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SE: Sleep Enable**

The SE bit must be written to logic one to make the MCU enter the sleep mode when the SLEEP instruction is executed. To avoid the MCU entering the sleep mode unless it is the programmers purpose, it is recommended to write the Sleep Enable (SE) bit to one just before the execution of the SLEEP instruction and to clear it immediately after waking up.

- **Bits 6...4 – SM2..0: Sleep Mode Select Bits 2, 1, and 0**

These bits select between the six available sleep modes as shown in Table 13.

Table 13. Sleep Mode Select

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby(1)
1	1	1	Extended Standby(1)

Note: 1. Standby mode and Extended Standby mode are only available with external crystals or resonators.

Idle Mode

When the SM2..0 bits are written to 000, the SLEEP instruction makes the MCU enter Idle mode, stopping the CPU but allowing SPI, USART, Analog Comparator, ADC, Two-wire Serial Interface, Timer/Counters, Watchdog, and the interrupt system to continue operating. This sleep mode basically halts clk_{CPU} and clk_{FLASH}, while allowing the other clocks to run.

Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the Timer Overflow and USART Transmit Complete interrupts. If wake-up from the Analog Comparator interrupt is not required, the Analog Comparator can be powered down by setting the ACD bit in the Analog Comparator Control and Status Register – ACSR. This will reduce power consumption in Idle mode. If the ADC is enabled, a conversion starts automatically when this mode is entered.

ADC Noise Reduction Mode

When the SM2..0 bits are written to 001, the SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the ADC, the External Interrupts, the Two-wire Serial Interface address watch, Timer/Counter2 and the Watchdog to continue operating (if enabled). This sleep mode basically halts $clk_{I/O}$, clk_{CPU}, and clk_{FLASH}, while allowing the other clocks to run.

This improves the noise environment for the ADC, enabling higher resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered. Apart form the ADC Conversion Complete interrupt, only an External Reset, a Watchdog Reset, a Brown-out Reset, a Two-wire Serial Interface Address Match Interrupt, a Timer/Counter2 interrupt, an SPM/EEPROM ready interrupt, an External level interrupt on INT0 or INT1, or an external interrupt on INT2 can wake up the MCU from ADC Noise Reduction mode.

Power-down Mode

When the SM2..0 bits are written to 010, the SLEEP instruction makes the MCU enter Power-down mode. In this mode, the External Oscillator is stopped, while the External interrupts, the Two-wire Serial Interface address watch, and the Watchdog continue operating (if enabled). Only an External Reset, a Watchdog Reset, a Brown-out Reset, a Two-wire Serial Interface address match interrupt, an External level interrupt on INT0 or INT1, or an External interrupt on INT2 can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

Note that if a level triggered interrupt is used for wake-up from Power-down mode, the changed level must be held for some time to wake up the MCU. Refer to “External Interrupts” on page 64 for details.

When waking up from Power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL fuses that define the reset time-out period, as described in “Clock Sources” on page 23.

Power-save Mode

When the SM2..0 bits are written to 011, the SLEEP instruction makes the MCU enter Power-save mode. This mode is identical to Power-down, with one exception:

If Timer/Counter2 is clocked asynchronously, i.e., the AS2 bit in ASSR is set, Timer/Counter2 will run during sleep. The device can wake up from either Timer Overflow or Output Compare event from Timer/Counter2 if the corresponding Timer/Counter2 interrupt enable bits are set in TIMSK, and the Global Interrupt Enable bit in SREG is set.

If the Asynchronous Timer is NOT clocked asynchronously, Power-down mode is recommended instead of Power-save mode because the contents of the registers in the

Asynchronous Timer should be considered undefined after wake-up in Power-save mode if AS2 is 0.

This sleep mode basically halts all clocks except clk_{ASY}, allowing operation only of asynchronous modules, including Timer/Counter2 if clocked asynchronously.

Standby Mode

When the SM2..0 bits are 110 and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Standby mode. This mode is identical to Power-down with the exception that the Oscillator is kept running. From Standby mode, the device wakes up in six clock cycles.

Extended Standby Mode

When the SM2..0 bits are 111 and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Extended Standby mode. This mode is identical to Power-save mode with the exception that the Oscillator is kept running. From Extended Standby mode, the device wakes up in six clock cycles..

Table 14. Active Clock Domains and Wake Up Sources in the Different Sleep Modes

	Active Clock domains					Oscillators		Wake-up Sources					
Sleep Mode	**clk_{CPU}**	**clk_{FLASH}**	**clk_{IO}**	**clk_{ADC}**	**clk_{ASY}**	**Main Clock Source Enabled**	**Timer Oscillator Enabled**	**INT2 INT1 INT0**	**TWI Address Match**	**Timer 2**	**SPM / EEPROM Ready**	**ADC**	**Other I/O**
Idle			X	X	X	X	X(2)	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X	X	X	
Power-down								X(3)	X				
Power-save					X(2)		X(2)	X(3)	X	X(2)			
Standby(1)						X		X(3)	X				
Extended Standby(1)					X(2)	X	X(2)	X(3)	X	X(2)			

Notes: 1. External Crystal or resonator selected as clock source.
2. If AS2 bit in ASSR is set.
3. Only INT2 or level interrupt INT1 and INT0.

Minimizing Power Consumption

There are several issues to consider when trying to minimize the power consumption in an AVR controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as possible of the device's functions are operating. All functions not needed should be disabled. In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption.

Analog to Digital Converter

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion. Refer to "Analog to Digital Converter" on page 199 for details on ADC operation.

Analog Comparator

When entering Idle mode, the Analog Comparator should be disabled if not used. When entering ADC Noise Reduction mode, the Analog Comparator should be disabled. In the other sleep modes, the Analog Comparator is automatically disabled. However, if the Analog Comparator is set up to use the Internal Voltage Reference as input, the Analog Comparator should be disabled in all sleep modes. Otherwise, the Internal Voltage Reference will be enabled, independent of sleep mode. Refer to “Analog Comparator” on page 196 for details on how to configure the Analog Comparator.

Brown-out Detector

If the Brown-out Detector is not needed in the application, this module should be turned off. If the Brown-out Detector is enabled by the BODEN Fuse, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to “Brown-out Detection” on page 37 for details on how to configure the Brown-out Detector.

Internal Voltage Reference

The Internal Voltage Reference will be enabled when needed by the Brown-out Detector, the Analog Comparator or the ADC. If these modules are disabled as described in the sections above, the internal voltage reference will be disabled and it will not be consuming power. When turned on again, the user must allow the reference to start up before the output is used. If the reference is kept on in sleep mode, the output can be used immediately. Refer to “Internal Voltage Reference” on page 39 for details on the start-up time.

Watchdog Timer

If the Watchdog Timer is not needed in the application, this module should be turned off. If the Watchdog Timer is enabled, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to “Watchdog Timer” on page 39 for details on how to configure the Watchdog Timer.

Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important thing is then to ensure that no pins drive resistive loads. In sleep modes where the both the I/O clock ($clk_{I/O}$) and the ADC clock (clk_{ADC}) are stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. Refer to the section “Digital Input Enable and Sleep Modes” on page 51 for details on which pins are enabled. If the input buffer is enabled and the input signal is left floating or have an analog signal level close to $V_{CC}/2$, the input buffer will use excessive power.

JTAG Interface and On-chip Debug System

If the On-chip debug system is enabled by the OCDEN Fuse and the chip enter Power down or Power save sleep mode, the main clock source remains enabled. In these sleep modes, this will contribute significantly to the total current consumption. There are three alternative ways to avoid this:

- Disable OCDEN Fuse.
- Disable JTAGEN Fuse.
- Write one to the JTD bit in MCUCSR.

The TDO pin is left floating when the JTAG interface is enabled while the JTAG TAP controller is not shifting data. If the hardware connected to the TDO pin does not pull up the logic level, power consumption will increase. Note that the TDI pin for the next device in the scan chain contains a pull-up that avoids this problem. Writing the JTD bit in the MCUCSR register to one or leaving the JTAG fuse unprogrammed disables the JTAG interface.

System Control and Reset

Resetting the AVR

During Reset, all I/O Registers are set to their initial values, and the program starts execution from the Reset Vector. The instruction placed at the Reset Vector must be a JMP – absolute jump – instruction to the reset handling routine. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa. The circuit diagram in Figure 15 shows the reset logic. Table 15 defines the electrical parameters of the reset circuitry.

The I/O ports of the AVR are immediately reset to their initial state when a reset source goes active. This does not require any clock source to be running.

After all reset sources have gone inactive, a delay counter is invoked, stretching the Internal Reset. This allows the power to reach a stable level before normal operation starts. The time-out period of the delay counter is defined by the user through the CKSEL Fuses. The different selections for the delay period are presented in “Clock Sources” on page 23.

Reset Sources

The ATmega32 has five sources of reset:

- Power-on Reset. The MCU is reset when the supply voltage is below the Power-on Reset threshold (V_{POT}).
- External Reset. The MCU is reset when a low level is present on the $\overline{\text{RESET}}$ pin for longer than the minimum pulse length.
- Watchdog Reset. The MCU is reset when the Watchdog Timer period expires and the Watchdog is enabled.
- Brown-out Reset. The MCU is reset when the supply voltage V_{CC} is below the Brown-out Reset threshold (V_{BOT}) and the Brown-out Detector is enabled.
- JTAG AVR Reset. The MCU is reset as long as there is a logic one in the Reset Register, one of the scan chains of the JTAG system. Refer to the section “IEEE 1149.1 (JTAG) Boundary-scan” on page 223 for details.

Figure 15. Reset Logic

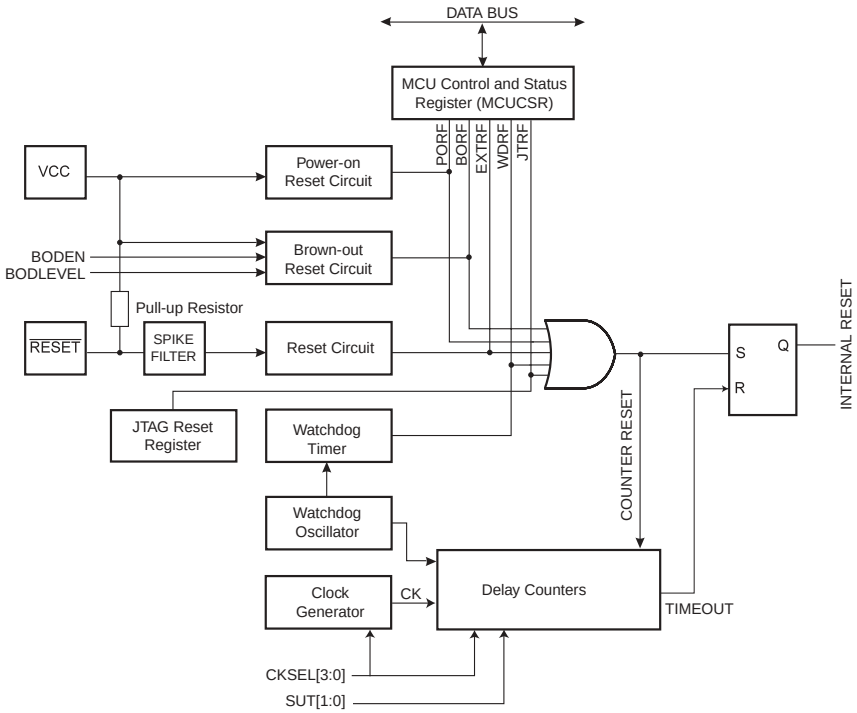


Table 15. Reset Characteristics

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{POT}	Power-on Reset Threshold Voltage (rising)			1.4	2.3	V
	Power-on Reset Threshold Voltage (falling)(1)			1.3	2.3	V
V_{RST}	$\overline{\text{RESET}}$ Pin Threshold Voltage		0.1 V_{CC}		0.9V_{CC}	V
t_{RST}	Minimum pulse width on $\overline{\text{RESET}}$ Pin				1.5	µs
V_{BOT}	Brown-out Reset Threshold Voltage(2)	BODLEVEL = 1	2.5	2.7	3.2	V
		BODLEVEL = 0	3.7	4.0	4.2	
t_{BOD}	Minimum low voltage period for Brown-out Detection	BODLEVEL = 1		2		µs
		BODLEVEL = 0		2		µs
V_{HYST}	Brown-out Detector hysteresis			50		mV

Notes: 1. The Power-on Reset will not work unless the supply voltage has been below V_{POT} (falling).

2. V_{BOT} may be below nominal minimum operating voltage for some devices. For devices where this is the case, the device is tested down to $V_{CC} = V_{BOT}$ during the production test. This guarantees that a Brown-out Reset will occur before V_{CC} drops to a voltage where correct operation of the microcontroller is no longer guaranteed. The test is performed using BODLEVEL = 1 for ATmega32L and BODLEVEL = 0 for ATmega32. BODLEVEL = 1 is not applicable for ATmega32.

Power-on Reset

A Power-on Reset (POR) pulse is generated by an On-chip detection circuit. The detection level is defined in Table 15. The POR is activated whenever V_{CC} is below the detection level. The POR circuit can be used to trigger the Start-up Reset, as well as to detect a failure in supply voltage.

A Power-on Reset (POR) circuit ensures that the device is reset from Power-on. Reaching the Power-on Reset threshold voltage invokes the delay counter, which determines how long the device is kept in RESET after V_{CC} rise. The RESET signal is activated again, without any delay, when V_{CC} decreases below the detection level.

Figure 16. MCU Start-up, $\overline{\text{RESET}}$ Tied to V_{CC}.

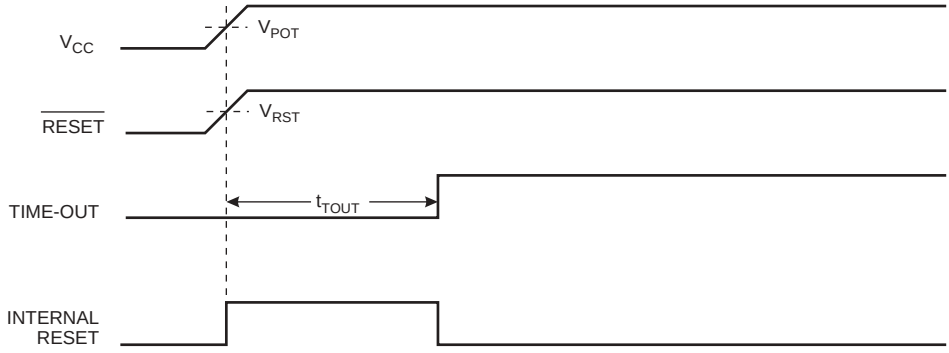
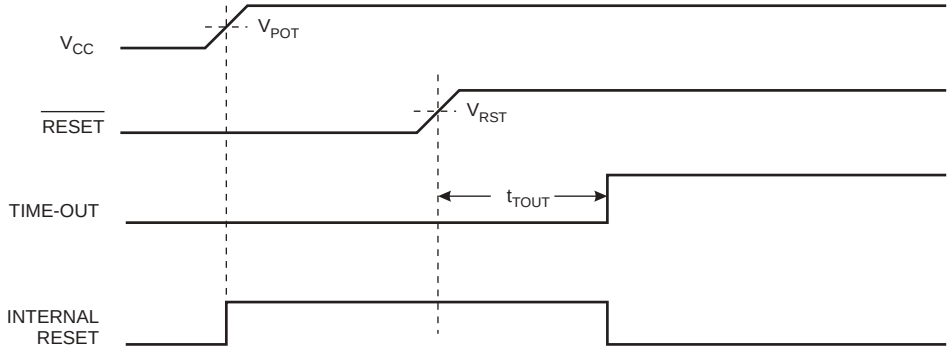


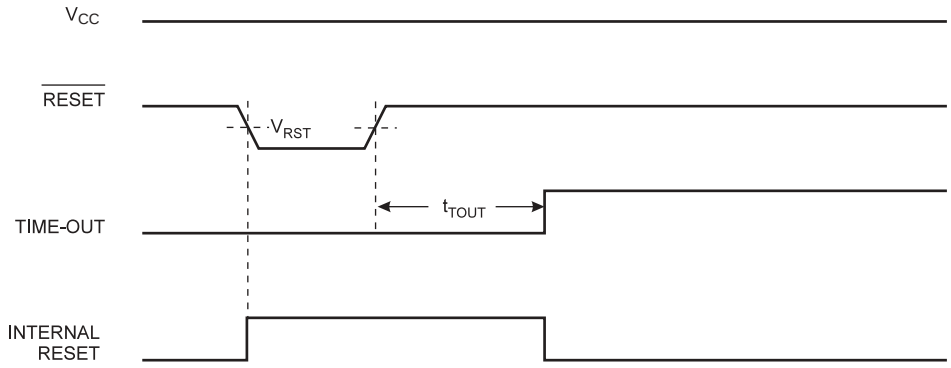
Figure 17. MCU Start-up, $\overline{\text{RESET}}$ Extended Externally



External Reset

An External Reset is generated by a low level on the $\overline{\text{RESET}}$ pin. Reset pulses longer than the minimum pulse width (see Table 15) will generate a reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset. When the applied signal reaches the Reset Threshold Voltage – V_{RST} – on its positive edge, the delay counter starts the MCU after the Time-out period t_{TOUT} has expired.

Figure 18. External Reset During Operation



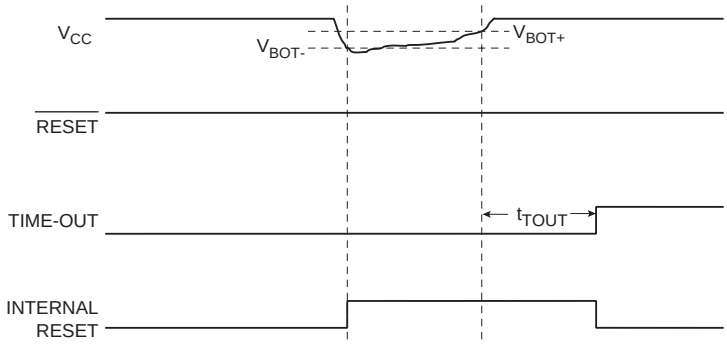
Brown-out Detection

ATmega32 has an On-chip Brown-out Detection (BOD) circuit for monitoring the V_{CC} level during operation by comparing it to a fixed trigger level. The trigger level for the BOD can be selected by the fuse BODLEVEL to be 2.7V (BODLEVEL unprogrammed), or 4.0V (BODLEVEL programmed). The trigger level has a hysteresis to ensure spike free Brown-out Detection. The hysteresis on the detection level should be interpreted as $V_{BOT+} = V_{BOT} + V_{HYST}/2$ and $V_{BOT-} = V_{BOT} - V_{HYST}/2$.

The BOD circuit can be enabled/disabled by the fuse BODEN. When the BOD is enabled (BODEN programmed), and V_{CC} decreases to a value below the trigger level (V_{BOT-} in Figure 19), the Brown-out Reset is immediately activated. When V_{CC} increases above the trigger level (V_{BOT+} in Figure 19), the delay counter starts the MCU after the Time-out period t_{TOUT} has expired.

The BOD circuit will only detect a drop in V_{CC} if the voltage stays below the trigger level for longer than t_{BOD} given in Table 15.

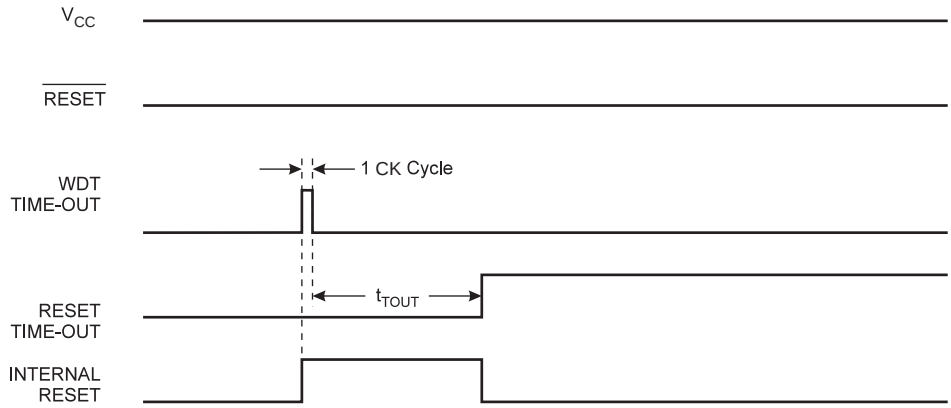
Figure 19. Brown-out Reset During Operation



Watchdog Reset

When the Watchdog times out, it will generate a short reset pulse of one CK cycle duration. On the falling edge of this pulse, the delay timer starts counting the Time-out period t_{TOUT}. Refer to page 39 for details on operation of the Watchdog Timer.

Figure 20. Watchdog Reset During Operation



MCU Control and Status Register – MCUCSR

The MCU Control and Status Register provides information on which reset source caused an MCU Reset.

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

- **Bit 4 – JTRF: JTAG Reset Flag**

This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 3 – WDRF: Watchdog Reset Flag**

This bit is set if a Watchdog Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 2 – BORF: Brown-out Reset Flag**

This bit is set if a Brown-out Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 1 – EXTRF: External Reset Flag**

This bit is set if an External Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 0 – PORF: Power-on Reset Flag**

This bit is set if a Power-on Reset occurs. The bit is reset only by writing a logic zero to the flag.

To make use of the Reset Flags to identify a reset condition, the user should read and then reset the MCUCSR as early as possible in the program. If the register is cleared before another reset occurs, the source of the reset can be found by examining the Reset Flags.

Internal Voltage Reference

ATmega32 features an internal bandgap reference. This reference is used for Brown-out Detection, and it can be used as an input to the Analog Comparator or the ADC. The 2.56V reference to the ADC is generated from the internal bandgap reference.

Voltage Reference Enable Signals and Start-up Time

The voltage reference has a start-up time that may influence the way it should be used. The start-up time is given in Table 16. To save power, the reference is not always turned on. The reference is on during the following situations:

1. When the BOD is enabled (by programming the BODEN Fuse).
2. When the bandgap reference is connected to the Analog Comparator (by setting the ACBG bit in ACSR).
3. When the ADC is enabled.

Thus, when the BOD is not enabled, after setting the ACBG bit or enabling the ADC, the user must always allow the reference to start up before the output from the Analog Comparator or ADC is used. To reduce power consumption in Power-down mode, the user can avoid the three conditions above to ensure that the reference is turned off before entering Power-down mode.

Table 16. Internal Voltage Reference Characteristics

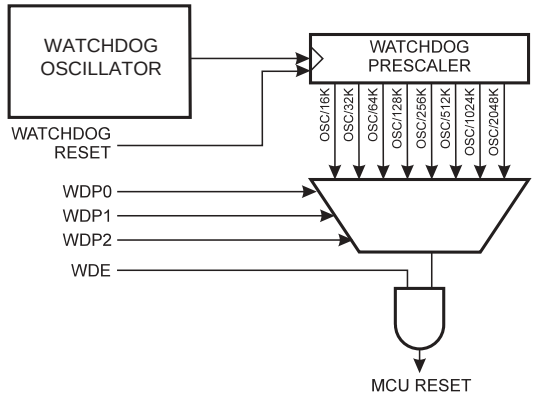
Symbol	Parameter	Min	Typ	Max	Units
V_{BG}	Bandgap reference voltage	1.15	1.23	1.35	V
t_{BG}	Bandgap reference start-up time		40	70	µs
I_{BG}	Bandgap reference current consumption		10		µA

Watchdog Timer

The Watchdog Timer is clocked from a separate On-chip Oscillator which runs at 1 MHz. This is the typical value at V_{CC} = 5V. See characterization data for typical values at other V_{CC} levels. By controlling the Watchdog Timer prescaler, the Watchdog Reset interval can be adjusted as shown in Table 17 on page 40. The WDR – Watchdog Reset – instruction resets the Watchdog Timer. The Watchdog Timer is also reset when it is disabled and when a Chip Reset occurs. Eight different clock cycle periods can be selected to determine the reset period. If the reset period expires without another Watchdog Reset, the ATmega32 resets and executes from the Reset Vector. For timing details on the Watchdog Reset, refer to page 38.

To prevent unintentional disabling of the Watchdog, a special turn-off sequence must be followed when the Watchdog is disabled. Refer to the description of the Watchdog Timer Control Register for details.

Figure 21. Watchdog Timer



Watchdog Timer Control Register – WDTCR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..5 – Res: Reserved Bits**

These bits are reserved bits in the ATmega32 and will always read as zero.

- **Bit 4 – WDTOE: Watchdog Turn-off Enable**

This bit must be set when the WDE bit is written to logic zero. Otherwise, the Watchdog will not be disabled. Once written to one, hardware will clear this bit after four clock cycles. Refer to the description of the WDE bit for a Watchdog disable procedure.

- **Bit 3 – WDE: Watchdog Enable**

When the WDE is written to logic one, the Watchdog Timer is enabled, and if the WDE is written to logic zero, the Watchdog Timer function is disabled. WDE can only be cleared if the WDTOE bit has logic level one. To disable an enabled Watchdog Timer, the following procedure must be followed:

1. In the same operation, write a logic one to WDTOE and WDE. A logic one must be written to WDE even though it is set to one before the disable operation starts.
2. Within the next four clock cycles, write a logic 0 to WDE. This disables the Watchdog.

- **Bits 2..0 – WDP2, WDP1, WDP0: Watchdog Timer Prescaler 2, 1, and 0**

The WDP2, WDP1, and WDP0 bits determine the Watchdog Timer prescaling when the Watchdog Timer is enabled. The different prescaling values and their corresponding Timeout Periods are shown in Table 17.

Table 17. Watchdog Timer Prescale Select

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	17.1 ms	16.3 ms
0	0	1	32K (32,768)	34.3 ms	32.5 ms
0	1	0	64K (65,536)	68.5 ms	65 ms
0	1	1	128K (131,072)	0.14 s	0.13 s
1	0	0	256K (262,144)	0.27 s	0.26 s
1	0	1	512K (524,288)	0.55 s	0.52 s
1	1	0	1,024K (1,048,576)	1.1 s	1.0 s
1	1	1	2,048K (2,097,152)	2.2 s	2.1 s

The following code example shows one assembly and one C function for turning off the WDT. The example assumes that interrupts are controlled (for example by disabling interrupts globally) so that no interrupts will occur during execution of these functions.

Assembly Code Example

```
WDT_off:
  ; Write logical one to WDTOE and WDE
  ldi  r16, (1<<WDTOE)|(1<<WDE)
  out  WDTCR, r16
  ; Turn off WDT
  ldi  r16, (0<<WDE)
  out  WDTCR, r16
  ret
```

C Code Example

```
void WDT_off(void)
{
  /* Write logical one to WDTOE and WDE */
  WDTCR = (1<<WDTOE) | (1<<WDE);
  /* Turn off WDT */
  WDTCR = 0x00;
}
```

Interrupts

This section describes the specifics of the interrupt handling as performed in ATmega32. For a general explanation of the AVR interrupt handling, refer to “Reset and Interrupt Handling” on page 11.

Interrupt Vectors in ATmega32

Table 18. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Notes: 1. When the BOOTRST fuse is programmed, the device will jump to the Boot Loader address at reset, see “Boot Loader Support – Read-While-Write Self-Programming” on page 242.

2. When the IVSEL bit in GICR is set, interrupt vectors will be moved to the start of the Boot Flash section. The address of each Interrupt Vector will then be the address in this table added to the start address of the Boot Flash section.

Table 19 shows Reset and Interrupt Vectors placement for the various combinations of BOOTRST and IVSEL settings. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa.

Table 19. Reset and Interrupt Vectors Placement[1]

BOOTRST	IVSEL	Reset address	Interrupt Vectors Start Address
1	0	$0000	$0002
1	1	$0000	Boot Reset Address + $0002
0	0	Boot Reset Address	$0002
0	1	Boot Reset Address	Boot Reset Address + $0002

Note: 1. The Boot Reset Address is shown in Table 100 on page 253. For the BOOTRST Fuse “1” means unprogrammed while “0” means programmed.

The most typical and general program setup for the Reset and Interrupt Vector Addresses in ATmega32 is:

```
Address   Labels   Code                 Comments
$000              jmp   RESET          ; Reset Handler
$002              jmp   EXT_INT0       ; IRQ0 Handler
$004              jmp   EXT_INT1       ; IRQ1 Handler
$006              jmp   EXT_INT2       ; IRQ2 Handler
$008              jmp   TIM2_COMP      ; Timer2 Compare Handler
$00A              jmp   TIM2_OVF       ; Timer2 Overflow Handler
$00C              jmp   TIM1_CAPT      ; Timer1 Capture Handler
$00E              jmp   TIM1_COMPA     ; Timer1 CompareA Handler
$010              jmp   TIM1_COMPB     ; Timer1 CompareB Handler
$012              jmp   TIM1_OVF       ; Timer1 Overflow Handler
$014              jmp   TIM0_COMP      ; Timer0 Compare Handler
$016              jmp   TIM0_OVF       ; Timer0 Overflow Handler
$018              jmp   SPI_STC        ; SPI Transfer Complete Handler
$01A              jmp   USART_RXC      ; USART RX Complete Handler
$01C              jmp   USART_UDRE     ; UDR Empty Handler
$01E              jmp   USART_TXC      ; USART TX Complete Handler
$020              jmp   ADC            ; ADC Conversion Complete Handler
$022              jmp   EE_RDY         ; EEPROM Ready Handler
$024              jmp   ANA_COMP       ; Analog Comparator Handler
$026              jmp   TWI            ; Two-wire Serial Interface Handler
$028              jmp   SPM_RDY        ; Store Program Memory Ready Handler
;
$02A     RESET:   ldi   r16,high(RAMEND) ; Main program start
$02B              out   SPH,r16        ; Set Stack Pointer to top of RAM
$02C              ldi   r16,low(RAMEND)
$02D              out   SPL,r16
$02E              sei                  ; Enable interrupts
$02F              <instr>  xxx
...       ...     ...
```

When the BOOTRST Fuse is unprogrammed, the Boot section size set to 4K bytes and the IVSEL bit in the GICR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address   Labels   Code                 Comments
$000      RESET:   ldi   r16,high(RAMEND) ; Main program start
$001               out   SPH,r16         ; Set Stack Pointer to top of RAM
$002             ldi   r16,low(RAMEND)
$003               out   SPL,r16
$004               sei                   ; Enable interrupts
$005               <instr>  xxx
;
.org $3802
$3802              jmp   EXT_INT0        ; IRQ0 Handler
$3804              jmp   EXT_INT1        ; IRQ1 Handler
...       ....     ..                    ;
$3828              jmp   SPM_RDY         ; Store Program Memory Ready Handler
```

When the BOOTRST Fuse is programmed and the Boot section size set to 4K bytes, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address   Labels         Code                    Comments
.org $002
$002               jmp   EXT_INT0        ; IRQ0 Handler
$004               jmp   EXT_INT1        ; IRQ1 Handler
...       ....     ..                    ;
$028               jmp   SPM_RDY         ; Store Program Memory Ready Handler
;
.org $3800
$3800     RESET:   ldi   r16,high(RAMEND) ; Main program start
$3801              out   SPH,r16         ; Set Stack Pointer to top of RAM
$3802              ldi   r16,low(RAMEND)
$3803              out   SPL,r16
$3804              sei                   ; Enable interrupts
$3805              <instr>  xxx
```

When the BOOTRST Fuse is programmed, the Boot section size set to 4K bytes and the IVSEL bit in the GICR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address   Labels   Code                   Comments
.org $3800
$3800              jmp   RESET        ; Reset handler
$3802              jmp   EXT_INT0     ; IRQ0 Handler
$3804              jmp   EXT_INT1     ; IRQ1 Handler
...       ....     ..                 ;
$3828              jmp   SPM_RDY      ; Store Program Memory Ready Handler
;
$382A     RESET:   ldi   r16,high(RAMEND) ; Main program start
$382B              out   SPH,r16          ; Set Stack Pointer to top of RAM
$382C              ldi   r16,low(RAMEND)
$382D              out   SPL,r16
$382E              sei                    ; Enable interrupts
$382F              <instr>  xxx
```

Moving Interrupts Between Application and Boot Space

The General Interrupt Control Register controls the placement of the Interrupt Vector table.

General Interrupt Control Register – GICR

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1 – IVSEL: Interrupt Vector Select**

When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the interrupt vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash section is determined by the BOOTSZ fuses. Refer to the section “Boot Loader Support – Read-While-Write Self-Programming” on page 242 for details. To avoid unintentional changes of Interrupt Vector tables, a special write procedure must be followed to change the IVSEL bit:

1. Write the Interrupt Vector Change Enable (IVCE) bit to one.
2. Within four cycles, write the desired value to IVSEL while writing a zero to IVCE.

Interrupts will automatically be disabled while this sequence is executed. Interrupts are disabled in the cycle IVCE is set, and they remain disabled until after the instruction following the write to IVSEL. If IVSEL is not written, interrupts remain disabled for four cycles. The I-bit in the Status Register is unaffected by the automatic disabling.

Note: If Interrupt Vectors are placed in the Boot Loader section and Boot Lock bit BLB02 is programmed, interrupts are disabled while executing from the Application section. If Interrupt Vectors are placed in the Application section and Boot Lock bit BLB12 is programed, interrupts are disabled while executing from the Boot Loader section. Refer to the section “Boot Loader Support – Read-While-Write Self-Programming” on page 242 for details on Boot Lock bits.

- **Bit 0 – IVCE: Interrupt Vector Change Enable**

The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts, as explained in the IVSEL description above. See Code Example below.

Assembly Code Example

```
Move_interrupts:
  ; Enable change of interrupt vectors
  ldi  r16, (1<<IVCE)
  out  GICR, r16
  ; Move interrupts to boot Flash section
  ldi  r16, (1<<IVSEL)
  out  GICR, r16
  ret
```

C Code Example

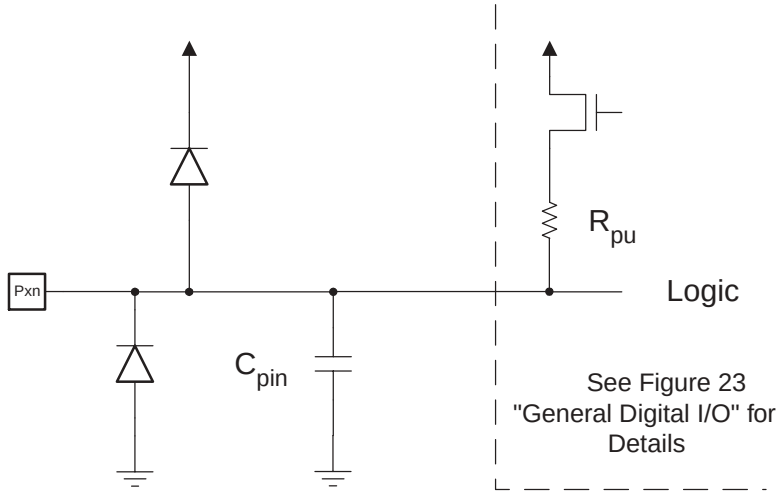
```
void Move_interrupts(void)
{
  /* Enable change of interrupt vectors */
  GICR = (1<<IVCE);
  /* Move interrupts to boot Flash section */
  GICR = (1<<IVSEL);
}
```

I/O Ports

Introduction

All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions. The same applies when changing drive value (if configured as output) or enabling/disabling of pull-up resistors (if configured as input). Each output buffer has symmetrical drive characteristics with both high sink and source capability. The pin driver is strong enough to drive LED displays directly. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both V_{CC} and Ground as indicated in Figure 22. Refer to “Electrical Characteristics” on page 285 for a complete list of parameters.

Figure 22. I/O Pin Equivalent Schematic



All registers and bit references in this section are written in general form. A lower case “x” represents the numbering letter for the port, and a lower case “n” represents the bit number. However, when using the register or bit defines in a program, the precise form must be used. i.e., PORTB3 for bit no. 3 in Port B, here documented generally as PORTxn. The physical I/O Registers and bit locations are listed in “Register Description for I/O Ports” on page 62.

Three I/O memory address locations are allocated for each port, one each for the Data Register – PORTx, Data Direction Register – DDRx, and the Port Input Pins – PINx. The Port Input Pins I/O location is read only, while the Data Register and the Data Direction Register are read/write. In addition, the Pull-up Disable – PUD bit in SFIOR disables the pull-up function for all pins in all ports when set.

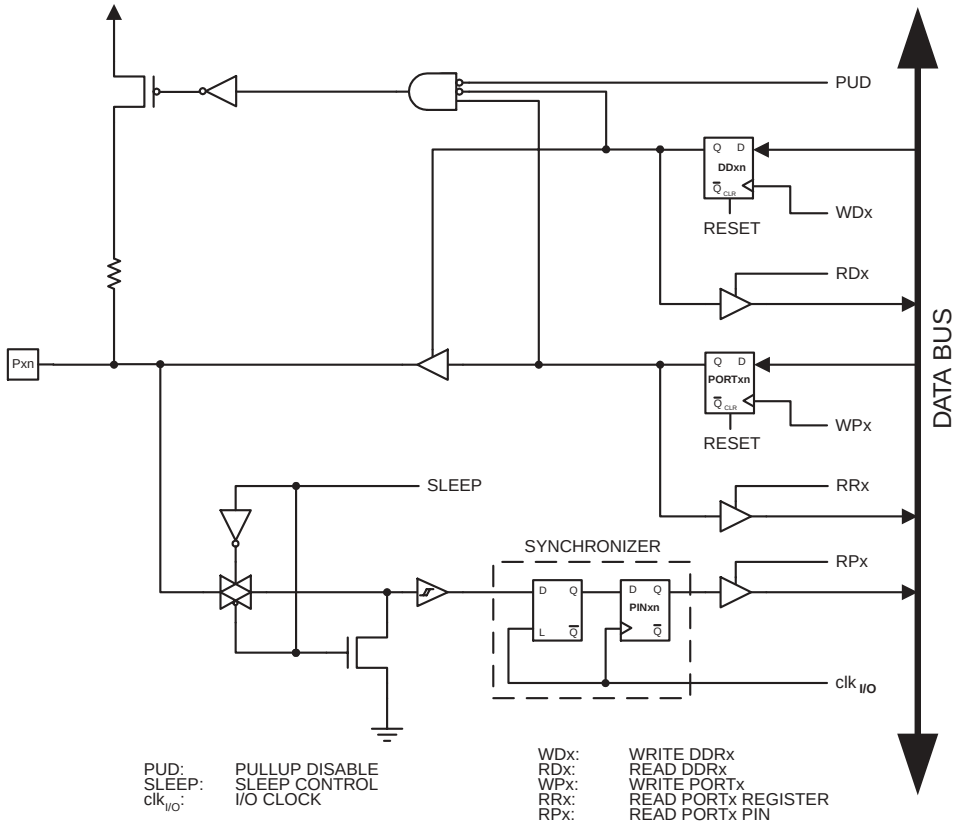
Using the I/O port as General Digital I/O is described in “Ports as General Digital I/O” on page 48. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in “Alternate Port Functions” on page 52. Refer to the individual module sections for a full description of the alternate functions.

Note that enabling the alternate function of some of the port pins does not affect the use of the other pins in the port as general digital I/O.

Ports as General Digital I/O

The ports are bi-directional I/O ports with optional internal pull-ups. Figure 23 shows a functional description of one I/O-port pin, here generically called Pxn.

Figure 23. General Digital I/O(1)



Note: 1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in "Register Description for I/O Ports" on page 62, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when a reset condition becomes active, even if no clocks are running.

If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

When switching between tri-state ({DDxn, PORTxn} = 0b00) and output high ({DDxn, PORTxn} = 0b11), an intermediate state with either pull-up enabled ({DDxn, PORTxn} = 0b01) or output low ({DDxn, PORTxn} = 0b10) must occur. Normally, the pull-up

enabled state is fully acceptable, as a high-impedant environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the SFIOR Register can be set to disable all pull-ups in all ports.

Switching between input with pull-up and output low generates the same problem. The user must use either the tri-state ({DDxn, PORTxn} = 0b00) or the output high state ({DDxn, PORTxn} = 0b11) as an intermediate step.

Table 20 summarizes the control signals for the pin value.

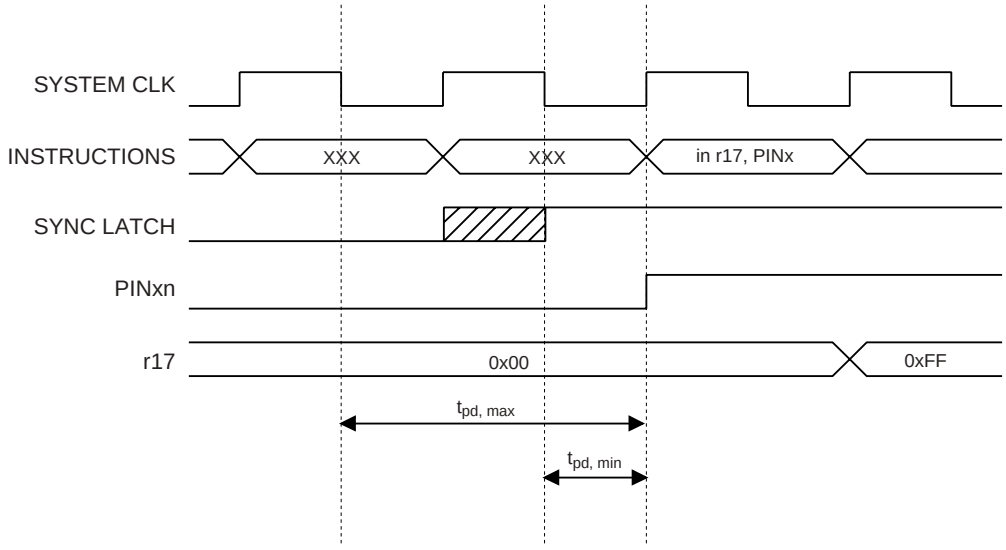
Table 20. Port Pin Configurations

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Reading the Pin Value

Independent of the setting of Data Direction bit DDxn, the port pin can be read through the PINxn Register bit. As shown in Figure 23, the PINxn Register bit and the preceding latch constitute a synchronizer. This is needed to avoid metastability if the physical pin changes value near the edge of the internal clock, but it also introduces a delay. Figure 24 shows a timing diagram of the synchronization when reading an externally applied pin value. The maximum and minimum propagation delays are denoted $t_{pd,max}$ and $t_{pd,min}$ respectively.

Figure 24. Synchronization when Reading an Externally Applied Pin Value

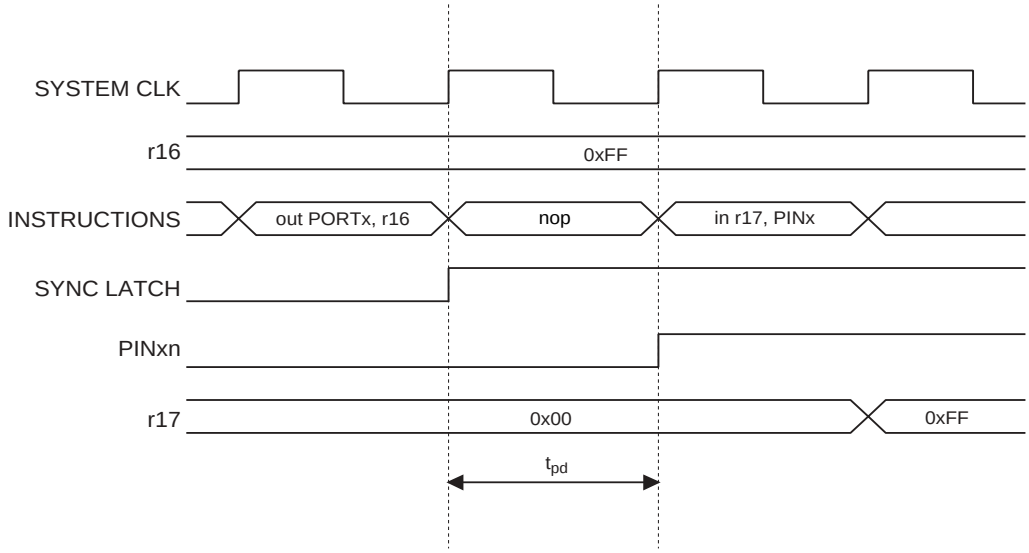


Consider the clock period starting shortly *after* the first falling edge of the system clock. The latch is closed when the clock is low, and goes transparent when the clock is high, as indicated by the shaded region of the “SYNC LATCH” signal. The signal value is latched when the system clock goes low. It is clocked into the PINxn Register at the

succeeding positive clock edge. As indicated by the two arrows $t_{pd,max}$ and $t_{pd,min}$, a single signal transition on the pin will be delayed between ½ and 1½ system clock period depending upon the time of assertion.

When reading back a software assigned pin value, a *nop* instruction must be inserted as indicated in Figure 25. The *out* instruction sets the “SYNC LATCH” signal at the positive edge of the clock. In this case, the delay t_{pd} through the synchronizer is one system clock period.

Figure 25. Synchronization when Reading a Software Assigned Pin Value



The following code example shows how to set port B pins 0 and 1 high, 2 and 3 low, and define the port pins from 4 to 7 as input with pull-ups assigned to port pins 6 and 7. The resulting pin values are read back again, but as previously discussed, a *nop* instruction is included to be able to read back the value recently assigned to some of the pins.

Assembly Code Example[1]

```
...
; Define pull-ups and set outputs high
; Define directions for port pins
ldi   r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
ldi   r17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
out   PORTB,r16
out   DDRB,r17
; Insert nop for synchronization
nop
; Read port pins
in    r16,PINB
...
```

C Code Example[1]

```
unsigned char i;
...
/* Define pull-ups and set outputs high */
/* Define directions for port pins */
PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
/* Insert nop for synchronization*/
_NOP();
/* Read port pins */
i = PINB;
...
```

Note: 1. For the assembly program, two temporary registers are used to minimize the time from pull-ups are set on pins 0, 1, 6, and 7, until the direction bits are correctly set, defining bit 2 and 3 as low and redefining bits 0 and 1 as strong high drivers.

Digital Input Enable and Sleep Modes

As shown in Figure 23, the digital input signal can be clamped to ground at the input of the schmitt-trigger. The signal denoted SLEEP in the figure, is set by the MCU Sleep Controller in Power-down mode, Power-save mode, Standby mode, and Extended Standby mode to avoid high power consumption if some input signals are left floating, or have an analog signal level close to $V_{CC}/2$.

SLEEP is overridden for port pins enabled as External Interrupt pins. If the External Interrupt Request is not enabled, SLEEP is active also for these pins. SLEEP is also overridden by various other alternate functions as described in “Alternate Port Functions” on page 52.

If a logic high level (“one”) is present on an Asynchronous External Interrupt pin configured as “Interrupt on Rising Edge, Falling Edge, or Any Logic Change on Pin” while the External Interrupt is *not* enabled, the corresponding External Interrupt Flag will be set when resuming from the above mentioned sleep modes, as the clamping in these sleep modes produces the requested logic change.

Unconnected pins

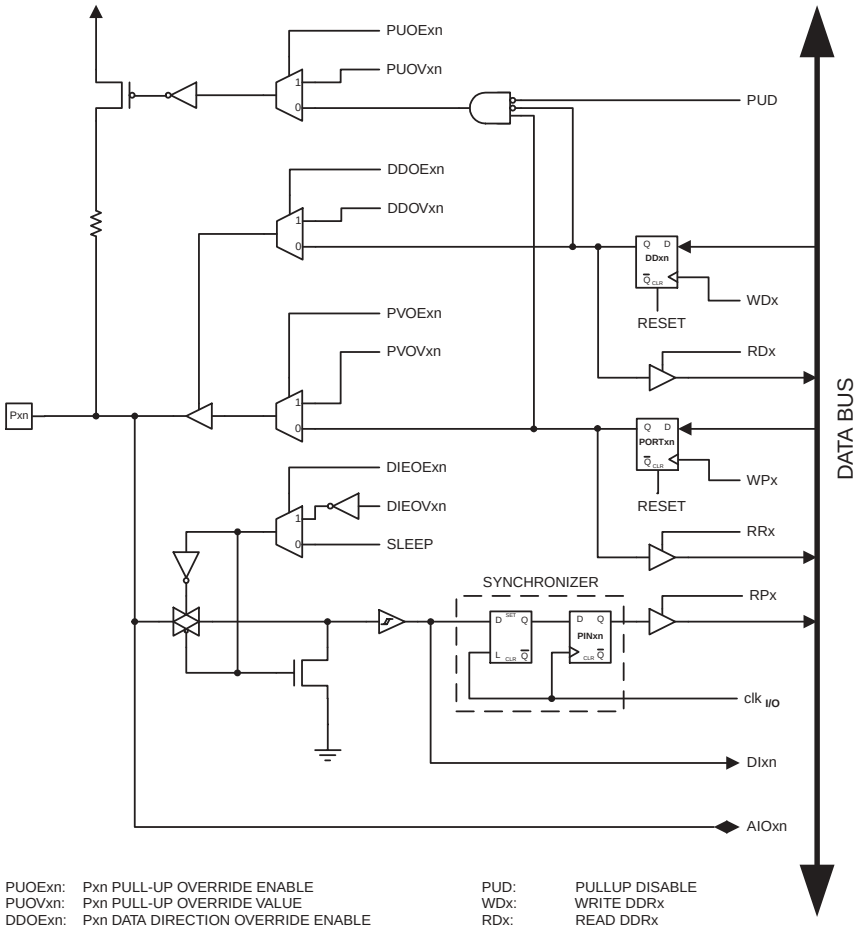
If some pins are unused, it is recommended to ensure that these pins have a defined level. Even though most of the digital inputs are disabled in the deep sleep modes as described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled (Reset, Active mode and Idle mode).

The simplest method to ensure a defined level of an unused pin, is to enable the internal pullup. In this case, the pullup will be disabled during reset. If low power consumption during reset is important, it is recommended to use an external pullup or pulldown. Connecting unused pins directly to V_{CC} or GND is not recommended, since this may cause excessive currents if the pin is accidentally configured as an output.

Alternate Port Functions

Most port pins have alternate functions in addition to being General Digital I/Os. Figure 26 shows how the port pin control signals from the simplified Figure 23 can be overridden by alternate functions. The overriding signals may not be present in all port pins, but the figure serves as a generic description applicable to all port pins in the AVR microcontroller family.

Figure 26. Alternate Port Functions(1)



PUOExn:	Pxn PULL-UP OVERRIDE ENABLE	PUD:	PULLUP DISABLE
PUOVxn:	Pxn PULL-UP OVERRIDE VALUE	WDx:	WRITE DDRx
DDOExn:	Pxn DATA DIRECTION OVERRIDE ENABLE	RDx:	READ DDRx
DDOVxn:	Pxn DATA DIRECTION OVERRIDE VALUE	RRx:	READ PORTx REGISTER
PVOExn:	Pxn PORT VALUE OVERRIDE ENABLE	WPx:	WRITE PORTx
PVOVxn:	Pxn PORT VALUE OVERRIDE VALUE	RPx:	READ PORTx PIN
DIEOExn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE	$clk_{I/O}$:	I/O CLOCK
DIEOVxn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE	DIxn:	DIGITAL INPUT PIN n ON PORTx
SLEEP:	SLEEP CONTROL	AIOxn:	ANALOG INPUT/OUTPUT PIN n ON PORTx

Note: 1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports. All other signals are unique for each pin.

Table 21 summarizes the function of the overriding signals. The pin and port indexes from Figure 26 are not shown in the succeeding tables. The overriding signals are generated internally in the modules having the alternate function.

Table 21. Generic Description of Overriding Signals for Alternate Functions

Signal Name	Full Name	Description
PUOE	Pull-up Override Enable	If this signal is set, the pull-up enable is controlled by the PUOV signal. If this signal is cleared, the pull-up is enabled when {DDxn, PORTxn, PUD} = 0b010.
PUOV	Pull-up Override Value	If PUOE is set, the pull-up is enabled/disabled when PUOV is set/cleared, regardless of the setting of the DDxn, PORTxn, and PUD Register bits.
DDOE	Data Direction Override Enable	If this signal is set, the Output Driver Enable is controlled by the DDOV signal. If this signal is cleared, the Output driver is enabled by the DDxn Register bit.
DDOV	Data Direction Override Value	If DDOE is set, the Output Driver is enabled/disabled when DDOV is set/cleared, regardless of the setting of the DDxn Register bit.
PVOE	Port Value Override Enable	If this signal is set and the Output Driver is enabled, the port value is controlled by the PVOV signal. If PVOE is cleared, and the Output Driver is enabled, the port Value is controlled by the PORTxn Register bit.
PVOV	Port Value Override Value	If PVOE is set, the port value is set to PVOV, regardless of the setting of the PORTxn Register bit.
DIEOE	Digital Input Enable Override Enable	If this bit is set, the Digital Input Enable is controlled by the DIEOV signal. If this signal is cleared, the Digital Input Enable is determined by MCU-state (Normal Mode, sleep modes).
DIEOV	Digital Input Enable Override Value	If DIEOE is set, the Digital Input is enabled/disabled when DIEOV is set/cleared, regardless of the MCU state (Normal Mode, sleep modes).
DI	Digital Input	This is the Digital Input to alternate functions. In the figure, the signal is connected to the output of the schmitt trigger but before the synchronizer. Unless the Digital Input is used as a clock source, the module with the alternate function will use its own synchronizer.
AIO	Analog Input/ output	This is the Analog Input/output to/from alternate functions. The signal is connected directly to the pad, and can be used bi-directionally.

The following subsections shortly describe the alternate functions for each port, and relate the overriding signals to the alternate function. Refer to the alternate function description for further details.

Special Function I/O Register – SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 2 – PUD: Pull-up disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See “Configuring the Pin” on page 48 for more details about this feature.

Alternate Functions of Port A

Port A has an alternate function as analog input for the ADC as shown in Table 22. If some Port A pins are configured as outputs, it is essential that these do not switch when a conversion is in progress. This might corrupt the result of the conversion.

Table 22. Port A Pins Alternate Functions

Port Pin	Alternate Function
PA7	ADC7 (ADC input channel 7)
PA6	ADC6 (ADC input channel 6)
PA5	ADC5 (ADC input channel 5)
PA4	ADC4 (ADC input channel 4)
PA3	ADC3 (ADC input channel 3)
PA2	ADC2 (ADC input channel 2)
PA1	ADC1 (ADC input channel 1)
PA0	ADC0 (ADC input channel 0)

Table 23 and Table 24 relate the alternate functions of Port A to the overriding signals shown in Figure 26 on page 52.

Table 23. Overriding Signals for Alternate Functions in PA7..PA4

Signal Name	PA7/ADC7	PA6/ADC6	PA5/ADC5	PA4/ADC4
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	ADC7 INPUT	ADC6 INPUT	ADC5 INPUT	ADC4 INPUT

Table 24. Overriding Signals for Alternate Functions in PA3..PA0

Signal Name	PA3/ADC3	PA2/ADC2	PA1/ADC1	PA0/ADC0
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	ADC3 INPUT	ADC2 INPUT	ADC1 INPUT	ADC0 INPUT

Alternate Functions of Port B

The Port B pins with alternate functions are shown in Table 25.

Table 25. Port B Pins Alternate Functions

Port Pin	Alternate Functions
PB7	SCK (SPI Bus Serial Clock)
PB6	MISO (SPI Bus Master Input/Slave Output)
PB5	MOSI (SPI Bus Master Output/Slave Input)
PB4	$\overline{SS}$ (SPI Slave Select Input)
PB3	AIN1 (Analog Comparator Negative Input) OC0 (Timer/Counter0 Output Compare Match Output)
PB2	AIN0 (Analog Comparator Positive Input) INT2 (External Interrupt 2 Input)
PB1	T1 (Timer/Counter1 External Counter Input)
PB0	T0 (Timer/Counter0 External Counter Input) XCK (USART External Clock Input/Output)

The alternate pin configuration is as follows:

- **SCK – Port B, Bit 7**

SCK: Master Clock output, Slave Clock input pin for SPI channel. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB7. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB7. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB7 bit.

- **MISO – Port B, Bit 6**

MISO: Master Data input, Slave Data output pin for SPI channel. When the SPI is enabled as a Master, this pin is configured as an input regardless of the setting of DDB6. When the SPI is enabled as a Slave, the data direction of this pin is controlled by

DDB6. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB6 bit.

- **MOSI – Port B, Bit 5**

MOSI: SPI Master Data output, Slave Data input for SPI channel. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB5. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB5. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB5 bit.

- **$\overline{SS}$ – Port B, Bit 4**

$\overline{SS}$: Slave Select input. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB4. As a Slave, the SPI is activated when this pin is driven low. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB4. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB4 bit.

- **AIN1/OC0 – Port B, Bit 3**

AIN1, Analog Comparator Negative Input. Configure the port pin as input with the internal pull-up switched off to avoid the digital port function from interfering with the function of the analog comparator.

OC0, Output Compare Match output: The PB3 pin can serve as an external output for the Timer/Counter0 Compare Match. The PB3 pin has to be configured as an output (DDB3 set (one)) to serve this function. The OC0 pin is also the output pin for the PWM mode timer function.

- **AIN0/INT2 – Port B, Bit 2**

AIN0, Analog Comparator Positive input. Configure the port pin as input with the internal pull-up switched off to avoid the digital port function from interfering with the function of the Analog Comparator.

INT2, External Interrupt Source 2: The PB2 pin can serve as an external interrupt source to the MCU.

- **T1 – Port B, Bit 1**

T1, Timer/Counter1 Counter Source.

- **T0/XCK – Port B, Bit 0**

T0, Timer/Counter0 Counter Source.

XCK, USART External Clock. The Data Direction Register (DDB0) controls whether the clock is output (DDB0 set) or input (DDB0 cleared). The XCK pin is active only when the USART operates in Synchronous mode.

Table 26 and Table 27 relate the alternate functions of Port B to the overriding signals shown in Figure 26 on page 52. SPI MSTR INPUT and SPI SLAVE OUTPUT constitute the MISO signal, while MOSI is divided into SPI MSTR OUTPUT and SPI SLAVE INPUT.

Table 26. Overriding Signals for Alternate Functions in PB7..PB4

Signal Name	PB7/SCK	PB6/MISO	PB5/MOSI	PB4/$\overline{SS}$
PUOE	SPE • $\overline{MSTR}$	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
PUOV	PORTB7 • $\overline{PUD}$	PORTB6 • $\overline{PUD}$	PORTB5 • $\overline{PUD}$	PORTB4 • $\overline{PUD}$
DDOE	SPE • $\overline{MSTR}$	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
DDOV	0	0	0	0
PVOE	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • MSTR	0
PVOV	SCK OUTPUT	SPI SLAVE OUTPUT	SPI MSTR OUTPUT	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	SCK INPUT	SPI MSTR INPUT	SPI SLAVE INPUT	SPI $\overline{SS}$
AIO	–	–	–	–

Table 27. Overriding Signals for Alternate Functions in PB3..PB0

Signal Name	PB3/OC0/AIN1	PB2/INT2/AIN0	PB1/T1	PB0/T0/XCK
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC0 ENABLE	0	0	UMSEL
PVOV	OC0	0	0	XCK OUTPUT
DIEOE	0	INT2 ENABLE	0	0
DIEOV	0	1	0	0
DI	–	INT2 INPUT	T1 INPUT	XCK INPUT/T0 INPUT
AIO	AIN1 INPUT	AIN0 INPUT	–	–

Alternate Functions of Port C

The Port C pins with alternate functions are shown in Table 28. If the JTAG interface is enabled, the pull-up resistors on pins PC5(TDI), PC3(TMS) and PC2(TCK) will be activated even if a reset occurs.

Table 28. Port C Pins Alternate Functions

Port Pin	Alternate Function
PC7	TOSC2 (Timer Oscillator Pin 2)
PC6	TOSC1 (Timer Oscillator Pin 1)
PC5	TDI (JTAG Test Data In)
PC4	TDO (JTAG Test Data Out)
PC3	TMS (JTAG Test Mode Select)
PC2	TCK (JTAG Test Clock)
PC1	SDA (Two-wire Serial Bus Data Input/Output Line)
PC0	SCL (Two-wire Serial Bus Clock Line)

The alternate pin configuration is as follows:

- **TOSC2 – Port C, Bit 7**

TOSC2, Timer Oscillator pin 2: When the AS2 bit in ASSR is set (one) to enable asynchronous clocking of Timer/Counter2, pin PC7 is disconnected from the port, and becomes the inverting output of the Oscillator amplifier. In this mode, a Crystal Oscillator is connected to this pin, and the pin can not be used as an I/O pin.

- **TOSC1 – Port C, Bit 6**

TOSC1, Timer Oscillator pin 1: When the AS2 bit in ASSR is set (one) to enable asynchronous clocking of Timer/Counter2, pin PC6 is disconnected from the port, and becomes the input of the inverting Oscillator amplifier. In this mode, a Crystal Oscillator is connected to this pin, and the pin can not be used as an I/O pin.

- **TDI – Port C, Bit 5**

TDI, JTAG Test Data In: Serial input data to be shifted in to the Instruction Register or Data Register (scan chains). When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **TDO – Port C, Bit 4**

TDO, JTAG Test Data Out: Serial output data from Instruction Register or Data Register. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

The TD0 pin is tri-stated unless TAP states that shifts out data are entered.

- **TMS – Port C, Bit 3**

TMS, JTAG Test Mode Select: This pin is used for navigating through the TAP-controller state machine. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **TCK – Port C, Bit 2**

TCK, JTAG Test Clock: JTAG operation is synchronous to TCK. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **SDA – Port C, Bit 1**

SDA, Two-wire Serial Interface Data: When the TWEN bit in TWCR is set (one) to enable the Two-wire Serial Interface, pin PC1 is disconnected from the port and becomes the Serial Data I/O pin for the Two-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation. When this pin is used by the Two-wire Serial Interface, the pull-up can still be controlled by the PORTC1 bit.

- **SCL – Port C, Bit 0**

SCL, Two-wire Serial Interface Clock: When the TWEN bit in TWCR is set (one) to enable the Two-wire Serial Interface, pin PC0 is disconnected from the port and becomes the Serial Clock I/O pin for the Two-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation. When this pin is used by the Two-wire Serial Interface, the pull-up can still be controlled by the PORTC0 bit.

Table 29 and Table 30 relate the alternate functions of Port C to the overriding signals shown in Figure 26 on page 52.

Table 29. Overriding Signals for Alternate Functions in PC7..PC4

Signal Name	PC7/TOSC2	PC6/TOSC1	PC5/TDI	PC4/TDO
PUOE	AS2	AS2	JTAGEN	JTAGEN
PUOV	0	0	1	0
DDOE	AS2	AS2	JTAGEN	JTAGEN
DDOV	0	0	0	SHIFT_IR + SHIFT_DR
PVOE	0	0	0	JTAGEN
PVOV	0	0	0	TDO
DIEOE	AS2	AS2	JTAGEN	JTAGEN
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	T/C2 OSC OUTPUT	T/C2 OSC INPUT	TDI	–

Table 30. Overriding Signals for Alternate Functions in PC3..PC0[1]

Signal Name	PC3/TMS	PC2/TCK	PC1/SDA	PC0/SCL
PUOE	JTAGEN	JTAGEN	TWEN	TWEN
PUOV	1	1	PORTC1 • $\overline{PUD}$	PORTC0 • $\overline{PUD}$
DDOE	JTAGEN	JTAGEN	TWEN	TWEN
DDOV	0	0	SDA_OUT	SCL_OUT
PVOE	0	0	TWEN	TWEN
PVOV	0	0	0	0
DIEOE	JTAGEN	JTAGEN	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	TMS	TCK	SDA INPUT	SCL INPUT

Note: 1. When enabled, the Two-wire Serial Interface enables slew-rate controls on the output pins PC0 and PC1. This is not shown in the figure. In addition, spike filters are connected between the AIO outputs shown in the port figure and the digital logic of the TWI module.

Alternate Functions of Port D

The Port D pins with alternate functions are shown in Table 31.

Table 31. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	OC2 (Timer/Counter2 Output Compare Match Output)
PD6	ICP (Timer/Counter1 Input Capture Pin)
PD5	OC1A (Timer/Counter1 Output Compare A Match Output)
PD4	OC1B (Timer/Counter1 Output Compare B Match Output)
PD3	INT1 (External Interrupt 1 Input)
PD2	INT0 (External Interrupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)

The alternate pin configuration is as follows:

- **OC2 – Port D, Bit 7**

OC2, Timer/Counter2 Output Compare Match output: The PD7 pin can serve as an external output for the Timer/Counter2 Output Compare. The pin has to be configured as an output (DDD7 set (one)) to serve this function. The OC2 pin is also the output pin for the PWM mode timer function.

- **ICP – Port D, Bit 6**

ICP – Input Capture Pin: The PD6 pin can act as an Input Capture pin for Timer/Counter1.

- **OC1A – Port D, Bit 5**

OC1A, Output Compare Match A output: The PD5 pin can serve as an external output for the Timer/Counter1 Output Compare A. The pin has to be configured as an output (DDD5 set (one)) to serve this function. The OC1A pin is also the output pin for the PWM mode timer function.

- **OC1B – Port D, Bit 4**

OC1B, Output Compare Match B output: The PD4 pin can serve as an external output for the Timer/Counter1 Output Compare B. The pin has to be configured as an output (DDD4 set (one)) to serve this function. The OC1B pin is also the output pin for the PWM mode timer function.

- **INT1 – Port D, Bit 3**

INT1, External Interrupt Source 1: The PD3 pin can serve as an external interrupt source.

- **INT0 – Port D, Bit 2**

INT0, External Interrupt Source 0: The PD2 pin can serve as an external interrupt source.

- **TXD – Port D, Bit 1**

TXD, Transmit Data (Data output pin for the USART). When the USART Transmitter is enabled, this pin is configured as an output regardless of the value of DDD1.

- **RXD – Port D, Bit 0**

RXD, Receive Data (Data input pin for the USART). When the USART Receiver is enabled this pin is configured as an input regardless of the value of DDD0. When the USART forces this pin to be an input, the pull-up can still be controlled by the PORTD0 bit.

Table 32 and Table 33 relate the alternate functions of Port D to the overriding signals shown in Figure 26 on page 52.

Table 32. Overriding Signals for Alternate Functions PD7..PD4

Signal Name	PD7/OC2	PD6/ICP	PD5/OC1A	PD4/OC1B
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC2 ENABLE	0	OC1A ENABLE	OC1B ENABLE
PVOV	OC2	0	OC1A	OC1B
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	ICP INPUT	–	–
AIO	–	–	–	–

Table 33. Overriding Signals for Alternate Functions in PD3..PD0

Signal Name	PD3/INT1	PD2/INT0	PD1/TXD	PD0/RXD
PUOE	0	0	TXEN	RXEN
PUOV	0	0	0	PORTD0 • $\overline{PUD}$
DDOE	0	0	TXEN	RXEN
DDOV	0	0	1	0
PVOE	0	0	TXEN	0
PVOV	0	0	TXD	0
DIEOE	INT1 ENABLE	INT0 ENABLE	0	0
DIEOV	1	1	0	0
DI	INT1 INPUT	INT0 INPUT	–	RXD
AIO	–	–	–	–

Register Description for I/O Ports

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port B Data Register – PORTB

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port B Data Direction Register – DDRB

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port B Input Pins Address – PINB

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port C Data Register – PORTC

Bit	7	6	5	4	3	2	1	0	
	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port C Data Direction Register – DDRC

Bit	7	6	5	4	3	2	1	0	
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port C Input Pins Address – PINC

Bit	7	6	5	4	3	2	1	0	
	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port D Data Register – PORTD

Bit	7	6	5	4	3	2	1	0	
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port D Data Direction Register – DDRD

Bit	7	6	5	4	3	2	1	0	
	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port D Input Pins Address – PIND

Bit	7	6	5	4	3	2	1	0	
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

External Interrupts

The External Interrupts are triggered by the INT0, INT1, and INT2 pins. Observe that, if enabled, the interrupts will trigger even if the INT0..2 pins are configured as outputs. This feature provides a way of generating a software interrupt. The external interrupts can be triggered by a falling or rising edge or a low level (INT2 is only an edge triggered interrupt). This is set up as indicated in the specification for the MCU Control Register – MCUCR – and MCU Control and Status Register – MCUCSR. When the external interrupt is enabled and is configured as level triggered (only INT0/INT1), the interrupt will trigger as long as the pin is held low. Note that recognition of falling or rising edge interrupts on INT0 and INT1 requires the presence of an I/O clock, described in "Clock Systems and their Distribution" on page 22. Low level interrupts on INT0/INT1 and the edge interrupt on INT2 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode. The I/O clock is halted in all sleep modes except Idle mode.

Note that if a level triggered interrupt is used for wake-up from Power-down mode, the changed level must be held for some time to wake up the MCU. This makes the MCU less sensitive to noise. The changed level is sampled twice by the Watchdog Oscillator clock. The period of the Watchdog Oscillator is 1 µs (nominal) at 5.0V and 25°C. The frequency of the Watchdog Oscillator is voltage dependent as shown in "Electrical Characteristics" on page 285. The MCU will wake up if the input has the required level during this sampling or if it is held until the end of the start-up time. The start-up time is defined by the SUT fuses as described in "System Clock and Clock Options" on page 22. If the level is sampled twice by the Watchdog Oscillator clock but disappears before the end of the start-up time, the MCU will still wake up, but no interrupt will be generated. The required level must be held long enough for the MCU to complete the wake up to trigger the level interrupt.

MCU Control Register – MCUCR

The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 3, 2 – ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0**

The External Interrupt 1 is activated by the external pin INT1 if the SREG I-bit and the corresponding interrupt mask in the GICR are set. The level and edges on the external INT1 pin that activate the interrupt are defined in Table 34. The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

Table 34. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

- **Bit 1, 0 – ISC01, ISC00: Interrupt Sense Control 0 Bit 1 and Bit 0**

The External Interrupt 0 is activated by the external pin INT0 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT0 pin that activate the interrupt are defined in Table 35. The value on the INT0 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

Table 35. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

MCU Control and Status Register – MCUCSR

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

- **Bit 6 – ISC2: Interrupt Sense Control 2**

The Asynchronous External Interrupt 2 is activated by the external pin INT2 if the SREG I-bit and the corresponding interrupt mask in GICR are set. If ISC2 is written to zero, a falling edge on INT2 activates the interrupt. If ISC2 is written to one, a rising edge on INT2 activates the interrupt. Edges on INT2 are registered asynchronously. Pulses on INT2 wider than the minimum pulse width given in Table 36 will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. When changing the ISC2 bit, an interrupt can occur. Therefore, it is recommended to first disable INT2 by clearing its Interrupt Enable bit in the GICR Register. Then, the ISC2 bit can be changed. Finally, the INT2 Interrupt Flag should be cleared by writing a logical one to its Interrupt Flag bit (INTF2) in the GIFR Register before the interrupt is re-enabled.

Table 36. Asynchronous External Interrupt Characteristics

Symbol	Parameter	Condition	Min	Typ	Max	Units
t_{INT}	Minimum pulse width for asynchronous external interrupt			50		ns

General Interrupt Control Register – GICR

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – INT1: External Interrupt Request 1 Enable**

When the INT1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control1 bits 1/0 (ISC11 and

ISC10) in the MCU General Control Register (MCUCR) define whether the External Interrupt is activated on rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of External Interrupt Request 1 is executed from the INT1 interrupt Vector.

- **Bit 6 – INT0: External Interrupt Request 0 Enable**

When the INT0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control0 bits 1/0 (ISC01 and ISC00) in the MCU General Control Register (MCUCR) define whether the External Interrupt is activated on rising and/or falling edge of the INT0 pin or level sensed. Activity on the pin will cause an interrupt request even if INT0 is configured as an output. The corresponding interrupt of External Interrupt Request 0 is executed from the INT0 interrupt vector.

- **Bit 5 – INT2: External Interrupt Request 2 Enable**

When the INT2 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control2 bit (ISC2) in the MCU Control and Status Register (MCUCSR) defines whether the External Interrupt is activated on rising or falling edge of the INT2 pin. Activity on the pin will cause an interrupt request even if INT2 is configured as an output. The corresponding interrupt of External Interrupt Request 2 is executed from the INT2 Interrupt Vector.

General Interrupt Flag Register – GIFR

Bit	7	6	5	4	3	2	1	0	
	INTF1	**INTF0**	**INTF2**	**–**	**–**	**–**	**–**	**–**	**GIFR**
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – INTF1: External Interrupt Flag 1**

When an edge or logic change on the INT1 pin triggers an interrupt request, INTF1 becomes set (one). If the I-bit in SREG and the INT1 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. This flag is always cleared when INT1 is configured as a level interrupt.

- **Bit 6 – INTF0: External Interrupt Flag 0**

When an edge or logic change on the INT0 pin triggers an interrupt request, INTF0 becomes set (one). If the I-bit in SREG and the INT0 bit in GICR are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. This flag is always cleared when INT0 is configured as a level interrupt.

- **Bit 5 – INTF2: External Interrupt Flag 2**

When an event on the INT2 pin triggers an interrupt request, INTF2 becomes set (one). If the I-bit in SREG and the INT2 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. Note that when entering some sleep modes with the INT2 interrupt disabled, the input buffer on this pin will be disabled. This may cause a logic change in internal signals which will set the INTF2 Flag. See “Digital Input Enable and Sleep Modes” on page 51 for more information.

8-bit Timer/Counter0 with PWM

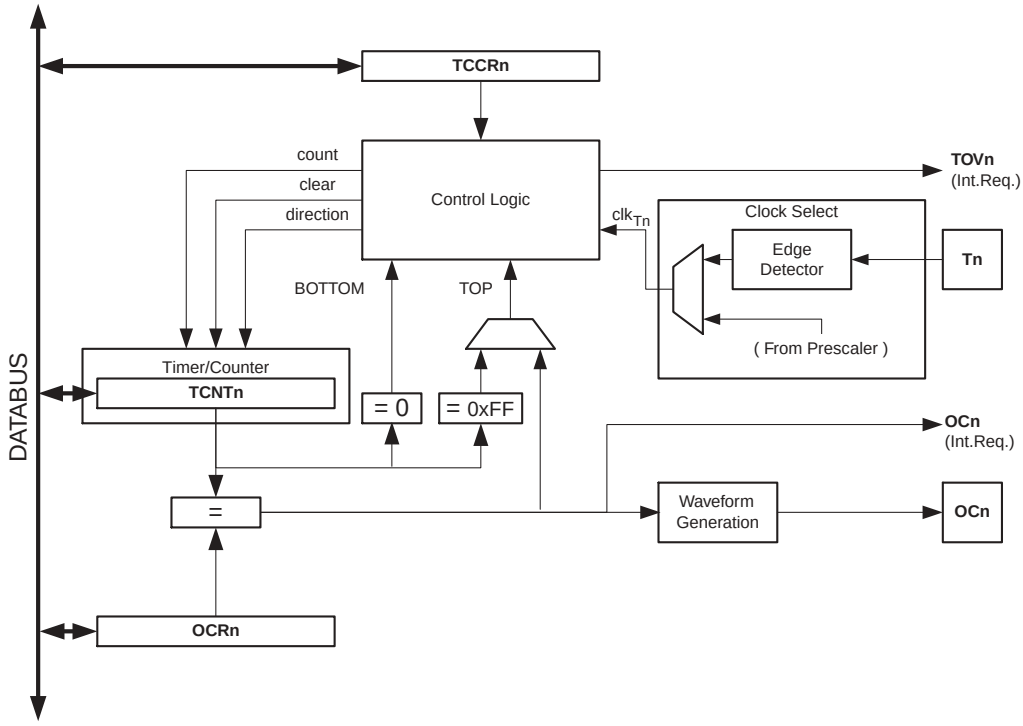
Timer/Counter0 is a general purpose, single channel, 8-bit Timer/Counter module. The main features are:

- **Single Channel Counter**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch-free, Phase Correct Pulse Width Modulator (PWM)**
- **Frequency Generator**
- **External Event Counter**
- **10-bit Clock Prescaler**
- **Overflow and Compare Match Interrupt Sources (TOV0 and OCF0)**

Overview

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 27. For the actual placement of I/O pins, refer to “Pinouts ATmega32” on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “8-bit Timer/Counter Register Description” on page 78.

Figure 27. 8-bit Timer/Counter Block Diagram



Registers

The Timer/Counter (TCNT0) and Output Compare Register (OCR0) are 8-bit registers. Interrupt request (abbreviated to Int.Req. in the figure) signals are all visible in the Timer Interrupt Flag Register (TIFR). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK). TIFR and TIMSK are not shown in the figure since these registers are shared by other timer units.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T0 pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock (clk_{T0}).

The double buffered Output Compare Register (OCR0) is compared with the Timer/Counter value at all times. The result of the compare can be used by the waveform generator to generate a PWM or variable frequency output on the Output Compare Pin (OC0). See “Output Compare Unit” on page 69. for details. The compare match event will also set the Compare Flag (OCF0) which can be used to generate an output compare interrupt request.

Definitions

Many register and bit references in this document are written in general form. A lower case “n” replaces the Timer/Counter number, in this case 0. However, when using the register or bit defines in a program, the precise form must be used i.e., TCNT0 for accessing Timer/Counter0 counter value and so on.

The definitions in Table 37 are also used extensively throughout the document.

Table 37. Definitions

BOTTOM	The counter reaches the BOTTOM when it becomes 0x00.
MAX	The counter reaches its MAXimum when it becomes 0xFF (decimal 255).
TOP	The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX) or the value stored in the OCR0 Register. The assignment is dependent on the mode of operation.

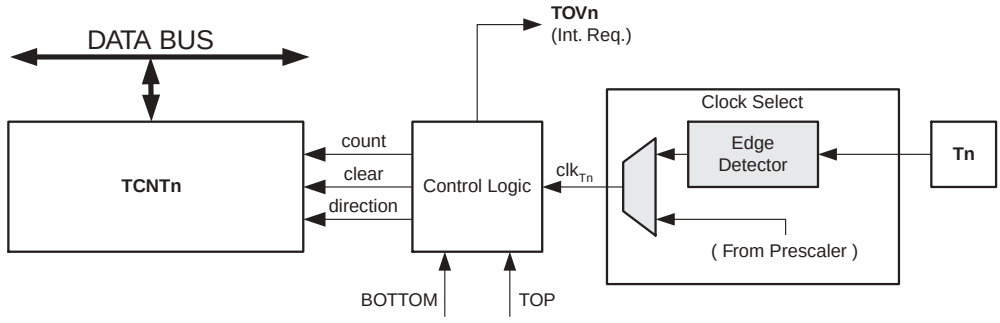
Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the clock select logic which is controlled by the clock select (CS02:0) bits located in the Timer/Counter Control Register (TCCR0). For details on clock sources and prescaler, see “Timer/Counter0 and Timer/Counter1 Prescalers” on page 82.

Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 28 shows a block diagram of the counter and its surroundings.

Figure 28. Counter Unit Block Diagram



Signal description (internal signals):

count Increment or decrement TCNT0 by 1.

direction Select between increment and decrement.

clear Clear TCNT0 (set all bits to zero).

clk_{Tn} Timer/Counter clock, referred to as clk_{T0} in the following.

TOP Signalize that TCNT0 has reached maximum value.

BOTTOM Signalize that TCNT0 has reached minimum value (zero).

Depending of the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock (clk_{T0}). clk_{T0} can be generated from an external or internal clock source, selected by the Clock Select bits (CS02:0). When no clock source is selected (CS02:0 = 0) the timer is stopped. However, the TCNT0 value can be accessed by the CPU, regardless of whether clk_{T0} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the WGM01 and WGM00 bits located in the Timer/Counter Control Register (TCCR0). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare output OC0. For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 71.

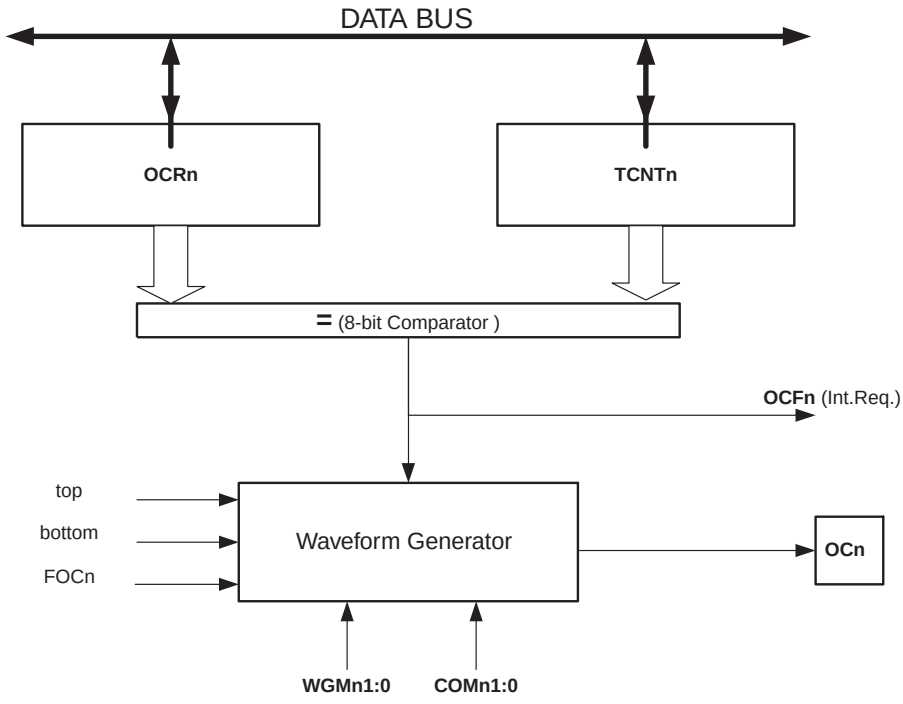
The Timer/Counter Overflow (TOV0) Flag is set according to the mode of operation selected by the WGM01:0 bits. TOV0 can be used for generating a CPU interrupt.

Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the Output Compare Register (OCR0). Whenever TCNT0 equals OCR0, the comparator signals a match. A match will set the Output Compare Flag (OCF0) at the next timer clock cycle. If enabled (OCIE0 = 1 and Global Interrupt Flag in SREG is set), the Output Compare Flag generates an output compare interrupt. The OCF0 Flag is automatically cleared when the interrupt is executed. Alternatively, the OCF0 Flag can be cleared by software by writing a logical one to its I/O bit location. The waveform generator uses the match signal to generate an output according to operating mode set by the WGM01:0 bits and Compare Output mode (COM01:0) bits. The max and bottom signals are used by the waveform generator for handling the special cases of the extreme values in some modes of operation (See “Modes of Operation” on page 71.).

Figure 29 shows a block diagram of the output compare unit.

Figure 29. Output Compare Unit, Block Diagram



The OCR0 Register is double buffered when using any of the Pulse Width Modulation (PWM) modes. For the normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR0 Compare Register to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR0 Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR0 Buffer Register, and if double buffering is disabled the CPU will access the OCR0 directly.

Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (FOC0) bit. Forcing compare match will not set the OCF0 Flag or reload/clear the timer, but the OC0 pin will be updated as if a real compare match had occurred (the COM01:0 bits settings define whether the OC0 pin is set, cleared or toggled).

Compare Match Blocking by TCNT0 Write

All CPU write operations to the TCNT0 Register will block any compare match that occur in the next timer clock cycle, even when the timer is stopped. This feature allows OCR0 to be initialized to the same value as TCNT0 without triggering an interrupt when the Timer/Counter clock is enabled.

Using the Output Compare Unit

Since writing TCNT0 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT0 when using the output compare channel, independently of whether the Timer/Counter is running or not. If the value written to TCNT0 equals the OCR0 value, the compare match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT0 value equal to BOTTOM when the counter is downcounting.

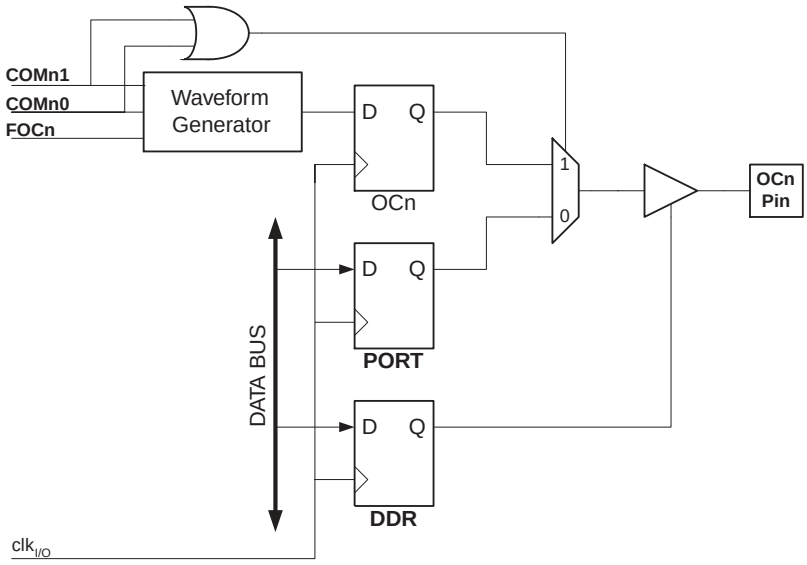
The setup of the OC0 should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC0 value is to use the Force Output Compare (FOC0) strobe bits in Normal mode. The OC0 Register keeps its value even when changing between waveform generation modes.

Be aware that the COM01:0 bits are not double buffered together with the compare value. Changing the COM01:0 bits will take effect immediately.

Compare Match Output Unit

The Compare Output mode (COM01:0) bits have two functions. The Waveform Generator uses the COM01:0 bits for defining the Output Compare (OC0) state at the next compare match. Also, the COM01:0 bits control the OC0 pin output source. Figure 30 shows a simplified schematic of the logic affected by the COM01:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O port Control Registers (DDR and PORT) that are affected by the COM01:0 bits are shown. When referring to the OC0 state, the reference is for the internal OC0 Register, not the OC0 pin. If a System Reset occur, the OC0 Register is reset to “0”.

Figure 30. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OC0) from the Waveform Generator if either of the COM01:0 bits are set. However, the OC0 pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OC0 pin (DDR_OC0) must be set as output before the OC0 value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the output compare pin logic allows initialization of the OC0 state before the output is enabled. Note that some COM01:0 bit settings are reserved for certain modes of operation. See “8-bit Timer/Counter Register Description” on page 78.

Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM01:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM01:0 = 0 tells the waveform generator that no action on the OC0 Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 39 on page 79. For fast PWM mode, refer to Table 40 on page 79, and for phase correct PWM refer to Table 41 on page 79.

A change of the COM01:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC0 strobe bits.

Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGM01:0) and Compare Output mode (COM01:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM01:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM01:0 bits control whether the output should be set, cleared, or toggled at a compare match (See “Compare Match Output Unit” on page 70.).

For detailed timing information refer to Figure 34, Figure 35, Figure 36 and Figure 37 in “Timer/Counter Timing Diagrams” on page 76.

Normal Mode

The simplest mode of operation is the normal mode (WGM01:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero. The TOV0 Flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV0 Flag, the timer resolution can be increased by software. There are no special cases to consider in the normal mode, a new counter value can be written anytime.

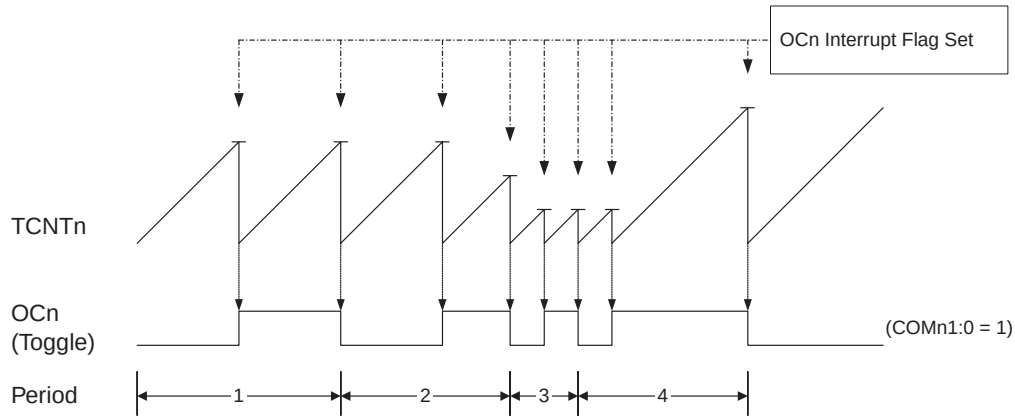
The output compare unit can be used to generate interrupts at some given time. Using the output compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM01:0 = 2), the OCR0 Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT0) matches the OCR0. The OCR0 defines the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 31. The counter value (TCNT0) increases until a compare match occurs between TCNT0 and OCR0, and then counter (TCNT0) is cleared.

Figure 31. CTC Mode, Timing Diagram



An interrupt can be generated each time the counter value reaches the TOP value by using the OCF0 Flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR0 is lower than the current value of TCNT0, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFF) and wrap around starting at 0x00 before the compare match can occur.

For generating a waveform output in CTC mode, the OC0 output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COM01:0 = 1). The OC0 value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum fre-

quency of $f_{OC0} = f_{clk_I/O}/2$ when OCR0 is set to zero (0x00). The waveform frequency is defined by the following equation:

$$f_{OCn} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRn)}$$

The N variable represents the prescale factor (1, 8, 64, 256, or 1024).

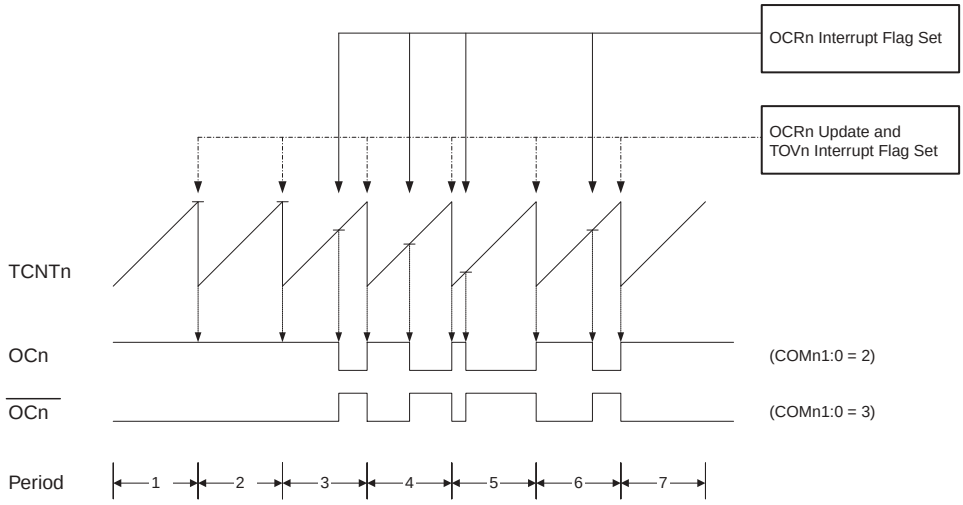
As for the Normal mode of operation, the TOV0 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM01:0 = 3) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to MAX then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC0) is cleared on the compare match between TCNT0 and OCR0, and set at BOTTOM. In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the MAX value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 32. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent compare matches between OCR0 and TCNT0.

Figure 32. Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches MAX. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC0 pin. Setting the COM01:0 bits to 2 will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM01:0 to 3 (See Table 40 on page 79). The actual OC0 value will only be visible on the port pin if the data direction for the port

pin is set as output. The PWM waveform is generated by setting (or clearing) the OC0 Register at the compare match between OCR0 and TCNT0, and clearing (or setting) the OC0 Register at the timer clock cycle the counter is cleared (changes from MAX to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnPWM} = \frac{f_{clk_I/O}}{N \cdot 256}$$

The N variable represents the prescale factor (1, 8, 64, 256, or 1024).

The extreme values for the OCR0 Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCR0 is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR0 equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM01:0 bits.)

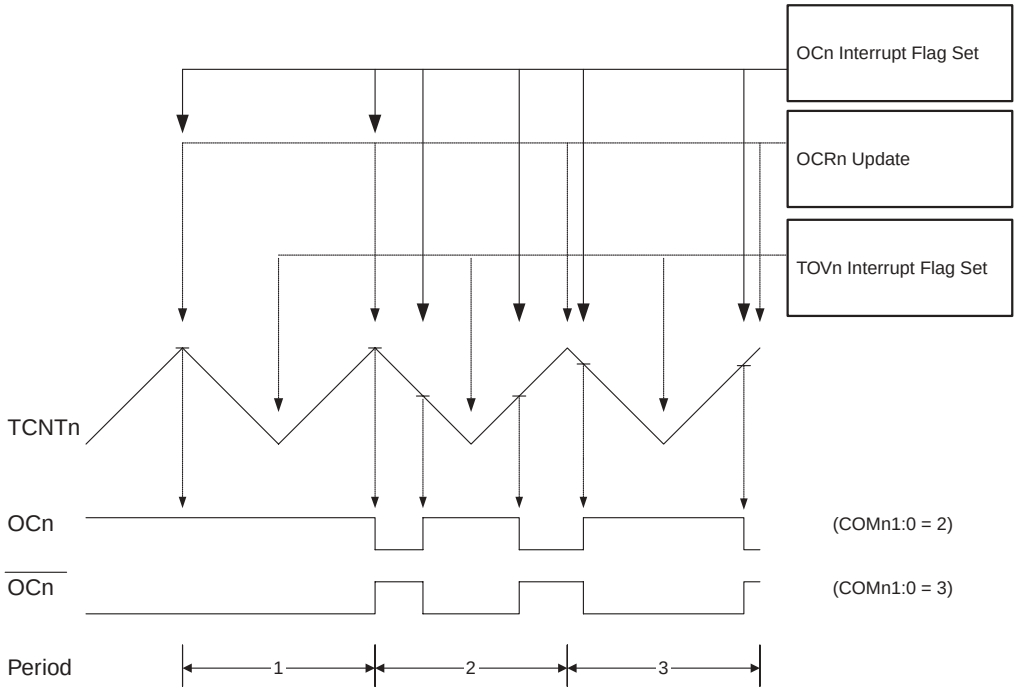
A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC0 to toggle its logical level on each compare match (COM01:0 = 1). The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk_I/O}/2$ when OCR0 is set to zero. This feature is similar to the OC0 toggle in CTC mode, except the double buffer feature of the output compare unit is enabled in the fast PWM mode.

Phase Correct PWM Mode

The phase correct PWM mode (WGM01:0 = 1) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC0) is cleared on the compare match between TCNT0 and OCR0 while upcounting, and set on the compare match while downcounting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode is fixed to eight bits. In phase correct PWM mode the counter is incremented until the counter value matches MAX. When the counter reaches MAX, it changes the count direction. The TCNT0 value will be equal to MAX for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 33. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent compare matches between OCR0 and TCNT0.

Figure 33. Phase Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches BOTTOM. The Interrupt Flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC0 pin. Setting the COM01:0 bits to 2 will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM01:0 to 3 (see Table 41 on page 79). The actual OC0 value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC0 Register at the compare match between OCR0 and TCNT0 when the counter increments, and setting (or clearing) the OC0 Register at compare match between OCR0 and TCNT0 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnPCPWM} = \frac{f_{clk_I/O}}{N \cdot 510}$$

The N variable represents the prescale factor (1, 8, 64, 256, or 1024).

The extreme values for the OCR0 Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR0 is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

At the very start of period 2 in Figure 33 OCn has a transition from high to low even though there is no Compare Match. The point of this transition is to guarantee symmetry around BOTTOM. There are two cases that give a transition without Compare Match:

- OCR0A changes its value from MAX, like in Figure 33. When the OCR0A value is MAX the OCn pin value is the same as the result of a down-counting Compare Match. To ensure symmetry around BOTTOM the OCn value at MAX must correspond to the result of an up-counting Compare Match.

- The timer starts counting from a value higher than the one in OCR0A, and for that reason misses the Compare Match and hence the OCn change that would have happened on the way up.

Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{T0}) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set. Figure 34 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

Figure 34. Timer/Counter Timing Diagram, no Prescaling

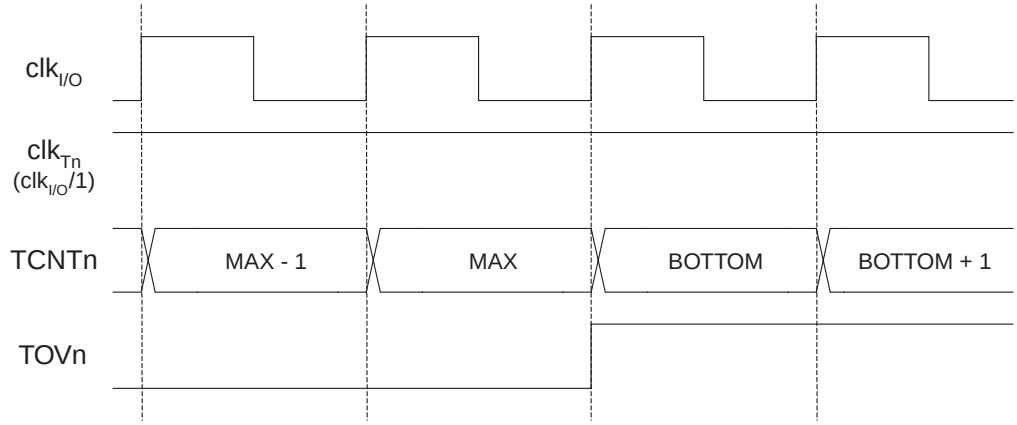


Figure 35 shows the same timing data, but with the prescaler enabled.

Figure 35. Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)

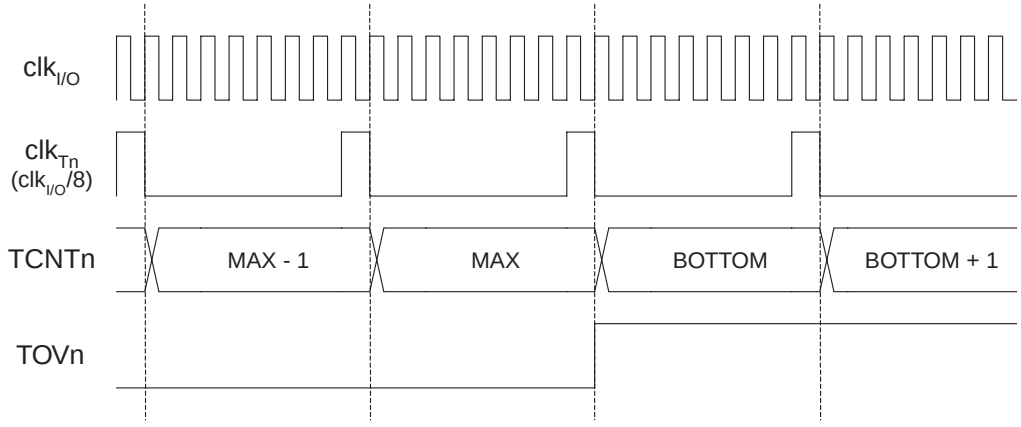


Figure 36 shows the setting of OCF0 in all modes except CTC mode.

Figure 36. Timer/Counter Timing Diagram, Setting of OCF0, with Prescaler ($f_{clk_I/O}/8$)

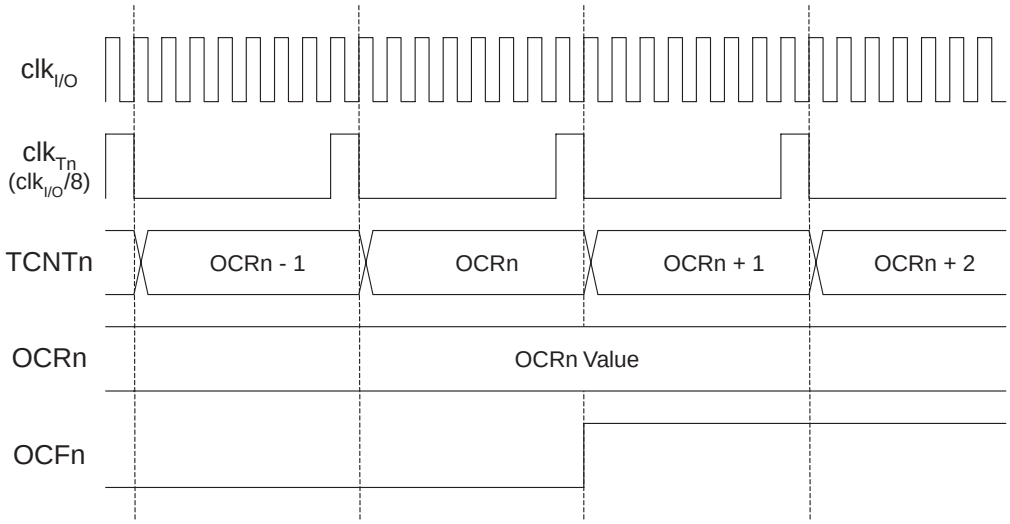
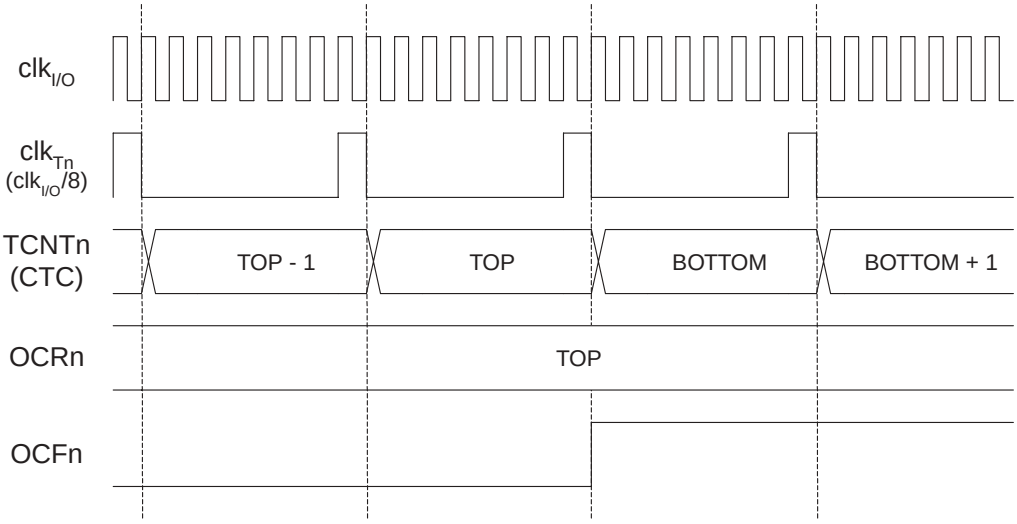


Figure 37 shows the setting of OCF0 and the clearing of TCNT0 in CTC mode.

Figure 37. Timer/Counter Timing Diagram, Clear Timer on Compare Match Mode, with Prescaler ($f_{clk_I/O}/8$)



8-bit Timer/Counter Register Description

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC0: Force Output Compare**

The FOC0 bit is only active when the WGM00 bit specifies a non-PWM mode. However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0 is written when operating in PWM mode. When writing a logical one to the FOC0 bit, an immediate compare match is forced on the Waveform Generation unit. The OC0 output is changed according to its COM01:0 bits setting. Note that the FOC0 bit is implemented as a strobe. Therefore it is the value present in the COM01:0 bits that determines the effect of the forced compare.

A FOC0 strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR0 as TOP.

The FOC0 bit is always read as zero.

- **Bit 6, 3 – WGM01:0: Waveform Generation Mode**

These bits control the counting sequence of the counter, the source for the maximum (TOP) counter value, and what type of Waveform Generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode, Clear Timer on Compare Match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes. See Table 38 and “Modes of Operation” on page 71.

Table 38. Waveform Generation Mode Bit Description[1]

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Note: 1. The CTC0 and PWM0 bit definition names are now obsolete. Use the WGM01:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

- **Bit 5:4 – COM01:0: Compare Match Output Mode**

These bits control the Output Compare pin (OC0) behavior. If one or both of the COM01:0 bits are set, the OC0 output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OC0 pin must be set in order to enable the output driver.

When OC0 is connected to the pin, the function of the COM01:0 bits depends on the WGM01:0 bit setting. Table 39 shows the COM01:0 bit functionality when the WGM01:0 bits are set to a normal or CTC mode (non-PWM).

Table 39. Compare Output Mode, non-PWM Mode

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Table 40 shows the COM01:0 bit functionality when the WGM01:0 bits are set to fast PWM mode.

Table 40. Compare Output Mode, Fast PWM Mode(1)

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at TOP
1	1	Set OC0 on compare match, clear OC0 at TOP

Note: 1. A special case occurs when OCR0 equals TOP and COM01 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 73 for more details.

Table 41 shows the COM01:0 bit functionality when the WGM01:0 bits are set to phase correct PWM mode.

Table 41. Compare Output Mode, Phase Correct PWM Mode(1)

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

Note: 1. A special case occurs when OCR0 equals TOP and COM01 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Phase Correct PWM Mode” on page 74 for more details.

- **Bit 2:0 – CS02:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter.

Table 42. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)

Table 42. Clock Select Bit Description (Continued)

CS02	CS01	CS00	Description
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

Timer/Counter Register – TCNT0

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT0 Register blocks (removes) the compare match on the following timer clock. Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a compare match between TCNT0 and the OCR0 Register.

Output Compare Register – OCR0

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Output Compare Register contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC0 pin.

Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1 – OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable**

When the OCIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Compare Match interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter0 occurs, i.e., when the OCF0 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1 – OCF0: Output Compare Flag 0**

The OCF0 bit is set (one) when a compare match occurs between the Timer/Counter0 and the data in OCR0 – Output Compare Register0. OCF0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0 is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0 (Timer/Counter0 Compare Match Interrupt Enable), and OCF0 are set (one), the Timer/Counter0 Compare Match Interrupt is executed.

- **Bit 0 – TOV0: Timer/Counter0 Overflow Flag**

The bit TOV0 is set (one) when an overflow occurs in Timer/Counter0. TOV0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV0 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE0 (Timer/Counter0 Overflow Interrupt Enable), and TOV0 are set (one), the Timer/Counter0 Overflow interrupt is executed. In phase correct PWM mode, this bit is set when Timer/Counter0 changes counting direction at $00.

Timer/Counter0 and Timer/Counter1 Prescalers

Timer/Counter1 and Timer/Counter0 share the same prescaler module, but the Timer/Counters can have different prescaler settings. The description below applies to both Timer/Counter1 and Timer/Counter0.

Internal Clock Source

The Timer/Counter can be clocked directly by the system clock (by setting the CSn2:0 = 1). This provides the fastest operation, with a maximum Timer/Counter clock frequency equal to system clock frequency ($f_{CLK_I/O}$). Alternatively, one of four taps from the prescaler can be used as a clock source. The prescaled clock has a frequency of either $f_{CLK_I/O}/8$, $f_{CLK_I/O}/64$, $f_{CLK_I/O}/256$, or $f_{CLK_I/O}/1024$.

Prescaler Reset

The prescaler is free running, i.e., operates independently of the clock select logic of the Timer/Counter, and it is shared by Timer/Counter1 and Timer/Counter0. Since the prescaler is not affected by the Timer/Counter's clock select, the state of the prescaler will have implications for situations where a prescaled clock is used. One example of prescaling artifacts occurs when the timer is enabled and clocked by the prescaler (6 > CSn2:0 > 1). The number of system clock cycles from when the timer is enabled to the first count occurs can be from 1 to N+1 system clock cycles, where N equals the prescaler divisor (8, 64, 256, or 1024).

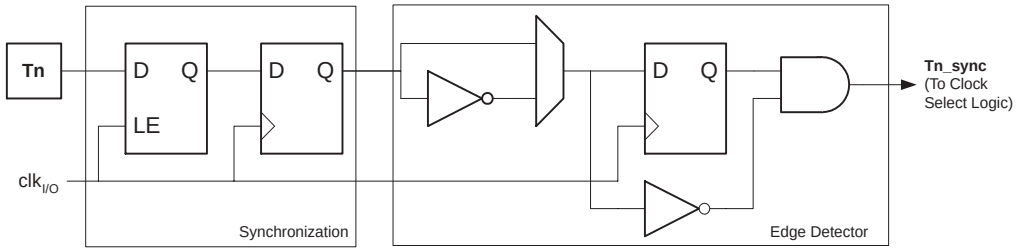
It is possible to use the Prescaler Reset for synchronizing the Timer/Counter to program execution. However, care must be taken if the other Timer/Counter that shares the same prescaler also uses prescaling. A prescaler reset will affect the prescaler period for all Timer/Counters it is connected to.

External Clock Source

An external clock source applied to the T1/T0 pin can be used as Timer/Counter clock (clk_{T1}/clk_{T0}). The T1/T0 pin is sampled once every system clock cycle by the pin synchronization logic. The synchronized (sampled) signal is then passed through the edge detector. Figure 38 shows a functional equivalent block diagram of the T1/T0 synchronization and edge detector logic. The registers are clocked at the positive edge of the internal system clock ($clk_{I/O}$). The latch is transparent in the high period of the internal system clock.

The edge detector generates one clk_{T1}/clk_{T0} pulse for each positive (CSn2:0 = 7) or negative (CSn2:0 = 6) edge it detects.

Figure 38. T1/T0 Pin Sampling



The synchronization and edge detector logic introduces a delay of 2.5 to 3.5 system clock cycles from an edge has been applied to the T1/T0 pin to the counter is updated.

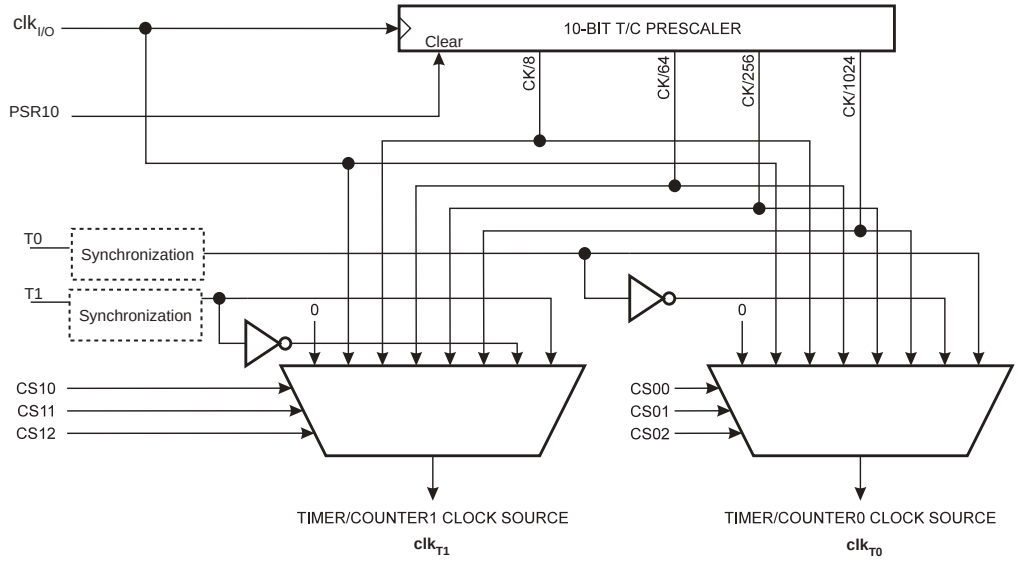
Enabling and disabling of the clock input must be done when T1/T0 has been stable for at least one system clock cycle, otherwise it is a risk that a false Timer/Counter clock pulse is generated.

Each half period of the external clock applied must be longer than one system clock cycle to ensure correct sampling. The external clock must be guaranteed to have less

than half the system clock frequency ($f_{ExtClk} < f_{clk_I/O}/2$) given a 50/50% duty cycle. Since the edge detector uses sampling, the maximum frequency of an external clock it can detect is half the sampling frequency (Nyquist sampling theorem). However, due to variation of the system clock frequency and duty cycle caused by Oscillator source (crystal, resonator, and capacitors) tolerances, it is recommended that maximum frequency of an external clock source is less than $f_{clk_I/O}/2.5$.

An external clock source can not be prescaled.

Figure 39. Prescaler for Timer/Counter0 and Timer/Counter1[1]



Note: 1. The synchronization logic on the input pins (T1/T0) is shown in Figure 38.

Special Function IO Register – SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 0 – PSR10: Prescaler Reset Timer/Counter1 and Timer/Counter0**

When this bit is written to one, the Timer/Counter1 and Timer/Counter0 prescaler will be reset. The bit will be cleared by hardware after the operation is performed. Writing a zero to this bit will have no effect. Note that Timer/Counter1 and Timer/Counter0 share the same prescaler and a reset of this prescaler will affect both timers. This bit will always be read as zero.

16-bit Timer/Counter1

The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement. The main features are:

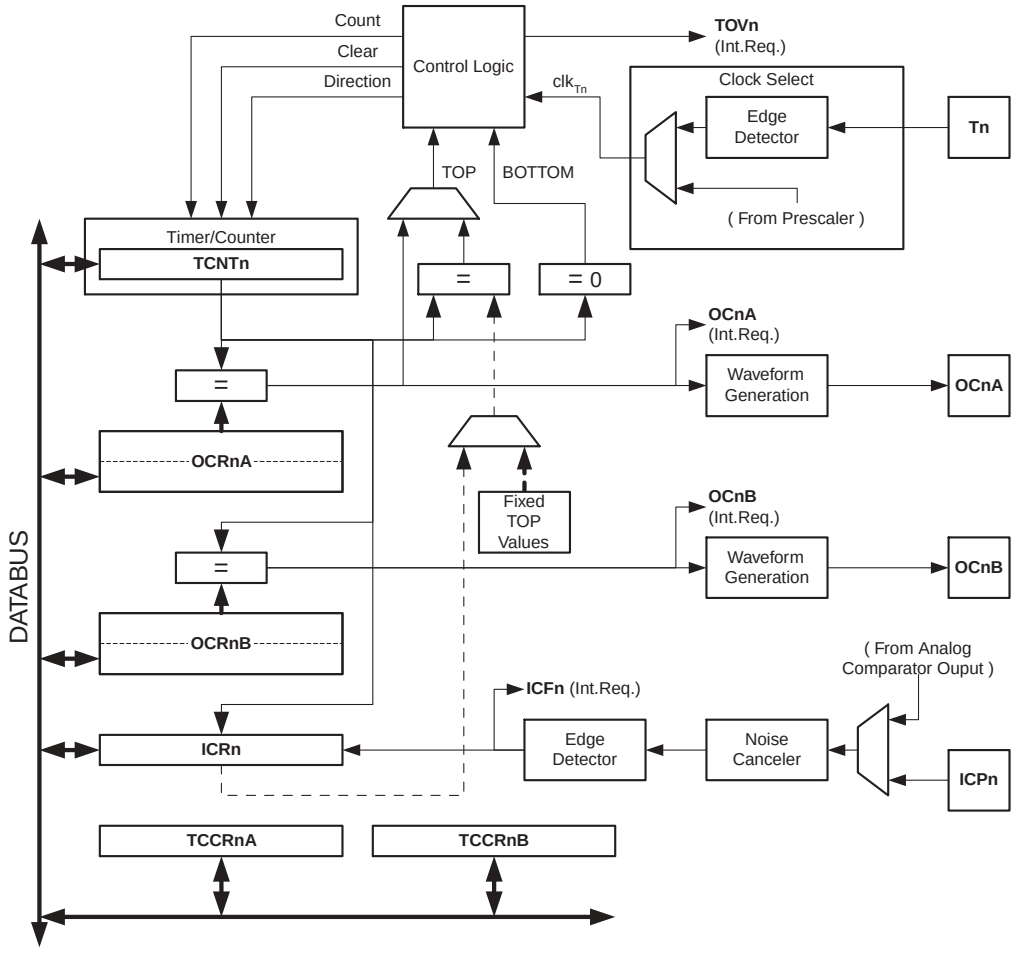
- **True 16-bit Design (i.e., Allows 16-bit PWM)**
- **Two Independent Output Compare Units**
- **Double Buffered Output Compare Registers**
- **One Input Capture Unit**
- **Input Capture Noise Canceler**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch-free, Phase Correct Pulse Width Modulator (PWM)**
- **Variable PWM Period**
- **Frequency Generator**
- **External Event Counter**
- **Four Independent Interrupt Sources (TOV1, OCF1A, OCF1B, and ICF1)**

Overview

Most register and bit references in this section are written in general form. A lower case "n" replaces the Timer/Counter number, and a lower case "x" replaces the output compare unit channel. However, when using the register or bit defines in a program, the precise form must be used i.e., TCNT1 for accessing Timer/Counter1 counter value and so on.

A simplified block diagram of the 16-bit Timer/Counter is shown in Figure 40. For the actual placement of I/O pins, refer to Figure 1 on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “16-bit Timer/Counter Register Description” on page 105.

Figure 40. 16-bit Timer/Counter Block Diagram[1]



Note: 1. Refer to Figure 1 on page 2, Table 25 on page 55, and Table 31 on page 60 for Timer/Counter1 pin placement and description.

Registers

The *Timer/Counter* (TCNT1), *Output Compare Registers* (OCR1A/B), and *Input Capture Register* (ICR1) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in the section “Accessing 16-bit Registers” on page 87. The *Timer/Counter Control Registers* (TCCR1A/B) are 8-bit registers and have no CPU access restrictions. Interrupt requests (abbreviated to Int.Req. in the figure) signals are all visible in the *Timer Interrupt Flag Register* (TIFR). All interrupts are individually masked with the *Timer Interrupt Mask Register* (TIMSK). TIFR and TIMSK are not shown in the figure since these registers are shared by other timer units.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T1 pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the clock select logic is referred to as the timer clock (clk_{T1}).

The double buffered Output Compare Registers (OCR1A/B) are compared with the Timer/Counter value at all time. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin

(OC1A/B). See “Output Compare Units” on page 92. The compare match event will also set the Compare Match Flag (OCF1A/B) which can be used to generate an output compare interrupt request.

The Input Capture Register can capture the Timer/Counter value at a given external (edge triggered) event on either the Input Capture Pin (ICP1) or on the Analog Comparator pins (See “Analog Comparator” on page 196.) The input capture unit includes a digital filtering unit (Noise Canceler) for reducing the chance of capturing noise spikes.

The TOP value, or maximum Timer/Counter value, can in some modes of operation be defined by either the OCR1A Register, the ICR1 Register, or by a set of fixed values. When using OCR1A as TOP value in a PWM mode, the OCR1A Register can not be used for generating a PWM output. However, the TOP value will in this case be double buffered allowing the TOP value to be changed in run time. If a fixed TOP value is required, the ICR1 Register can be used as an alternative, freeing the OCR1A to be used as PWM output.

Definitions

The following definitions are used extensively throughout the document:

Table 43. Definitions

BOTTOM	The counter reaches the *BOTTOM* when it becomes 0x0000.
MAX	The counter reaches its *MAX*imum when it becomes 0xFFFF (decimal 65535).
TOP	The counter reaches the *TOP* when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be one of the fixed values: 0x00FF, 0x01FF, or 0x03FF, or to the value stored in the OCR1A or ICR1 Register. The assignment is dependent of the mode of operation.

Compatibility

The 16-bit Timer/Counter has been updated and improved from previous versions of the 16-bit AVR Timer/Counter. This 16-bit Timer/Counter is fully compatible with the earlier version regarding:

- All 16-bit Timer/Counter related I/O Register address locations, including Timer Interrupt Registers.
- Bit locations inside all 16-bit Timer/Counter Registers, including Timer Interrupt Registers.
- Interrupt Vectors.

The following control bits have changed name, but have same functionality and register location:

- PWM10 is changed to WGM10.
- PWM11 is changed to WGM11.
- CTC1 is changed to WGM12.

The following bits are added to the 16-bit Timer/Counter Control Registers:

- FOC1A and FOC1B are added to TCCR1A.
- WGM13 is added to TCCR1B.

The 16-bit Timer/Counter has improvements that will affect the compatibility in some special cases.

Accessing 16-bit Registers

The TCNT1, OCR1A/B, and ICR1 are 16-bit registers that can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bit access. The same temporary register is shared between all 16-bit registers within each 16-bit timer. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the temporary register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the temporary register in the same clock cycle as the low byte is read.

Not all 16-bit accesses uses the temporary register for the high byte. Reading the OCR1A/B 16-bit registers does not involve using the temporary register.

To do a 16-bit write, *the high byte must be written before the low byte*. For a 16-bit read, *the low byte must be read before the high byte*.

The following code examples show how to access the 16-bit Timer Registers assuming that no interrupts updates the temporary register. The same principle can be used directly for accessing the OCR1A/B and ICR1 Registers. Note that when using “C”, the compiler handles the 16-bit access.

Assembly Code Example[1]

```
...
; Set TCNT1 to 0x01FF
ldi r17,0x01
ldi r16,0xFF
out TCNT1H,r17
out TCNT1L,r16
; Read TCNT1 into r17:r16
in  r16,TCNT1L
in  r17,TCNT1H
...
```

C Code Example[1]

```
unsigned int i;
...
/* Set TCNT1 to 0x01FF */
TCNT1 = 0x1FF;
/* Read TCNT1 into i */
i = TCNT1;
...
```

Note: 1. The example code assumes that the part specific header file is included.

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

It is important to notice that accessing 16-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 16-bit register, and the interrupt code updates the temporary register by accessing the same or any other of the 16-bit Timer Registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

The following code examples show how to do an atomic read of the TCNT1 Register contents. Reading any of the OCR1A/B or ICR1 Registers can be done by using the same principle.

Assembly Code Example[1]

```
TIM16_ReadTCNT1:
  ; Save global interrupt flag
  in  r18,SREG
  ; Disable interrupts
  cli
  ; Read TCNT1 into r17:r16
  in  r16,TCNT1L
  in  r17,TCNT1H
  ; Restore global interrupt flag
  out SREG,r18
  ret
```

C Code Example[1]

```
unsigned int TIM16_ReadTCNT1( void )
{
  unsigned char sreg;
  unsigned int i;
  /* Save global interrupt flag */
  sreg = SREG;
  /* Disable interrupts */
  _CLI();
  /* Read TCNT1 into i */
  i = TCNT1;
  /* Restore global interrupt flag */
  SREG = sreg;
  return i;
}
```

Note: 1. The example code assumes that the part specific header file is included.

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

The following code examples show how to do an atomic write of the TCNT1 Register contents. Writing any of the OCR1A/B or ICR1 Registers can be done by using the same principle.

Assembly Code Example[1]

```
TIM16_WriteTCNT1:
  ; Save global interrupt flag
  in  r18,SREG
  ; Disable interrupts
  cli
  ; Set TCNT1 to r17:r16
  out TCNT1H,r17
  out TCNT1L,r16
  ; Restore global interrupt flag
  out SREG,r18
  ret
```

C Code Example[1]

```
void TIM16_WriteTCNT1 ( unsigned int i )
{
  unsigned char sreg;
  unsigned int i;
  /* Save global interrupt flag */
  sreg = SREG;
  /* Disable interrupts */
  _CLI();
  /* Set TCNT1 to i */
  TCNT1 = i;
  /* Restore global interrupt flag */
  SREG = sreg;
}
```

Note: 1. The example code assumes that the part specific header file is included.

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNT1.

Reusing the Temporary High Byte Register

If writing to more than one 16-bit register where the high byte is the same for all registers written, then the high byte only needs to be written once. However, note that the same rule of atomic operation described previously also applies in this case.

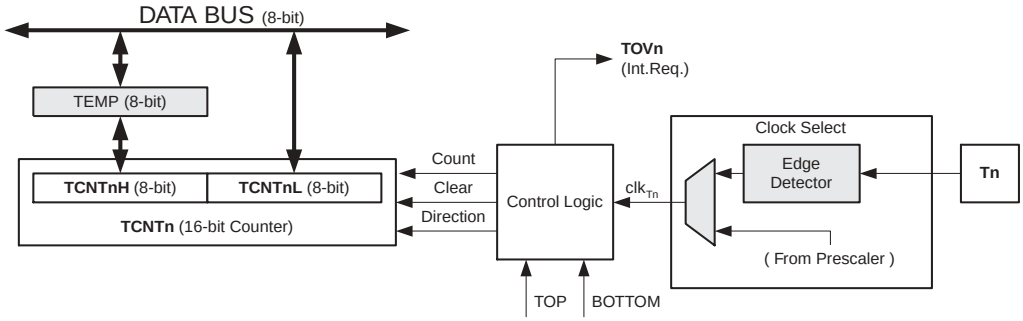
Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the *Clock Select* (CS12:0) bits located in the *Timer/Counter Control Register B* (TCCR1B). For details on clock sources and prescaler, see “Timer/Counter0 and Timer/Counter1 Prescalers” on page 82.

Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit bi-directional counter unit. Figure 41 shows a block diagram of the counter and its surroundings.

Figure 41. Counter Unit Block Diagram



Signal description (internal signals):

Count Increment or decrement TCNT1 by 1.

Direction Select between increment and decrement.

Clear Clear TCNT1 (set all bits to zero).

clk_{T1} Timer/Counter clock.

TOP Signalize that TCNT1 has reached maximum value.

BOTTOM Signalize that TCNT1 has reached minimum value (zero).

The 16-bit counter is mapped into two 8-bit I/O memory locations: *Counter High* (TCNT1H) containing the upper eight bits of the counter, and *Counter Low* (TCNT1L) containing the lower 8 bits. The TCNT1H Register can only be indirectly accessed by the CPU. When the CPU does an access to the TCNT1H I/O location, the CPU accesses the high byte temporary register (TEMP). The temporary register is updated with the TCNT1H value when the TCNT1L is read, and TCNT1H is updated with the temporary register value when TCNT1L is written. This allows the CPU to read or write the entire 16-bit counter value within one clock cycle via the 8-bit data bus. It is important to notice that there are special cases of writing to the TCNT1 Register when the counter is counting that will give unpredictable results. The special cases are described in the sections where they are of importance.

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each *timer clock* (clk_{T1}). The clk_{T1} can be generated from an external or internal clock source, selected by the *Clock Select* bits (CS12:0). When no clock source is selected (CS12:0 = 0) the timer is stopped. However, the TCNT1 value can be accessed by the CPU, independent of whether clk_{T1} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the *Waveform Generation Mode* bits (WGM13:0) located in the *Timer/Counter Control Registers* A and B (TCCR1A and TCCR1B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OC1x. For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 95.

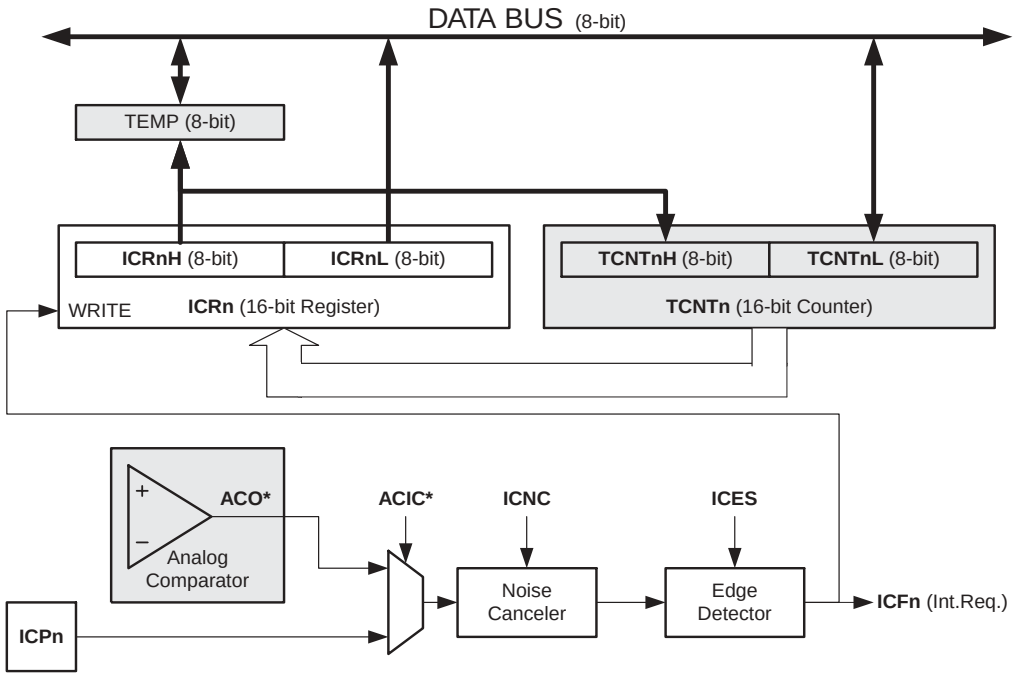
The *Timer/Counter Overflow* (TOV1) Flag is set according to the mode of operation selected by the WGM13:0 bits. TOV1 can be used for generating a CPU interrupt.

Input Capture Unit

The Timer/Counter incorporates an input capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICP1 pin or alternatively, via the Analog Comparator unit. The time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.

The input capture unit is illustrated by the block diagram shown in Figure 42. The elements of the block diagram that are not directly a part of the input capture unit are gray shaded. The small “n” in register and bit names indicates the Timer/Counter number.

Figure 42. Input Capture Unit Block Diagram



When a change of the logic level (an event) occurs on the *Input Capture pin* (ICP1), alternatively on the *Analog Comparator output* (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNT1) is written to the *Input Capture Register* (ICR1). The *Input Capture Flag* (ICF1) is set at the same system clock as the TCNT1 value is copied into ICR1 Register. If enabled (TICIE1 = 1), the Input Capture Flag generates an input capture interrupt. The ICF1 Flag is automatically cleared when the interrupt is executed. Alternatively the ICF1 Flag can be cleared by software by writing a logical one to its I/O bit location.

Reading the 16-bit value in the *Input Capture Register* (ICR1) is done by first reading the low byte (ICR1L) and then the high byte (ICR1H). When the low byte is read the high byte is copied into the high byte temporary register (TEMP). When the CPU reads the ICR1H I/O location it will access the TEMP Register.

The ICR1 Register can only be written when using a Waveform Generation mode that utilizes the ICR1 Register for defining the counter’s TOP value. In these cases the *Waveform Generation mode* (WGM13:0) bits must be set before the TOP value can be written to the ICR1 Register. When writing the ICR1 Register the high byte must be written to the ICR1H I/O location before the low byte is written to ICR1L.

For more information on how to access the 16-bit registers refer to “Accessing 16-bit Registers” on page 87.

Input Capture Trigger Source

The main trigger source for the input capture unit is the *Input Capture pin* (ICP1). Timer/Counter1 can alternatively use the Analog Comparator output as trigger source for the input capture unit. The Analog Comparator is selected as trigger source by setting the *Analog Comparator Input Capture* (ACIC) bit in the *Analog Comparator Control and Status Register* (ACSR). Be aware that changing trigger source can trigger a capture. The Input Capture Flag must therefore be cleared after the change.

Both the *Input Capture pin* (ICP1) and the *Analog Comparator output* (ACO) inputs are sampled using the same technique as for the T1 pin (Figure 38 on page 82). The edge detector is also identical. However, when the noise canceler is enabled, additional logic is inserted before the edge detector, which increases the delay by four system clock cycles. Note that the input of the noise canceler and edge detector is always enabled unless the Timer/Counter is set in a waveform generation mode that uses ICR1 to define TOP.

An input capture can be triggered by software by controlling the port of the ICP1 pin.

Noise Canceler

The noise canceler improves noise immunity by using a simple digital filtering scheme. The noise canceler input is monitored over four samples, and all four must be equal for changing the output that in turn is used by the edge detector.

The noise canceler is enabled by setting the *Input Capture Noise Canceler* (ICNC1) bit in *Timer/Counter Control Register B* (TCCR1B). When enabled the noise canceler introduces additional four system clock cycles of delay from a change applied to the input, to the update of the ICR1 Register. The noise canceler uses the system clock and is therefore not affected by the prescaler.

Using the Input Capture Unit

The main challenge when using the input capture unit is to assign enough processor capacity for handling the incoming events. The time between two events is critical. If the processor has not read the captured value in the ICR1 Register before the next event occurs, the ICR1 will be overwritten with a new value. In this case the result of the capture will be incorrect.

When using the input capture interrupt, the ICR1 Register should be read as early in the interrupt handler routine as possible. Even though the input capture interrupt has relatively high priority, the maximum interrupt response time is dependent on the maximum number of clock cycles it takes to handle any of the other interrupt requests.

Using the input capture unit in any mode of operation when the TOP value (resolution) is actively changed during operation, is not recommended.

Measurement of an external signal’s duty cycle requires that the trigger edge is changed after each capture. Changing the edge sensing must be done as early as possible after the ICR1 Register has been read. After a change of the edge, the Input Capture Flag (ICF1) must be cleared by software (writing a logical one to the I/O bit location). For measuring frequency only, the clearing of the ICF1 Flag is not required (if an interrupt handler is used).

Output Compare Units

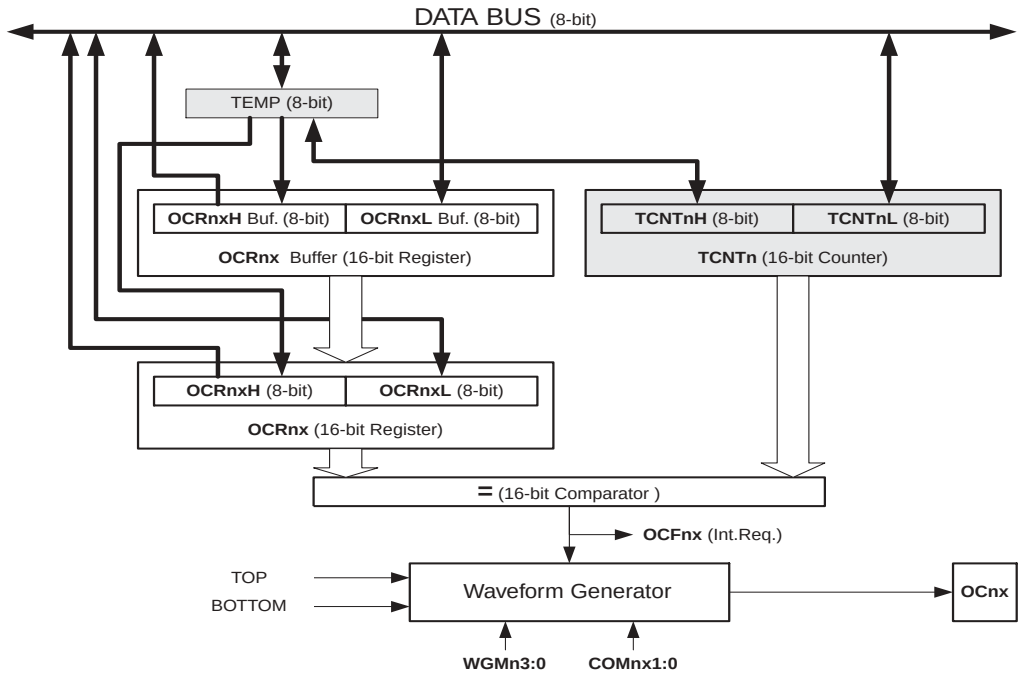
The 16-bit comparator continuously compares TCNT1 with the *Output Compare Register* (OCR1x). If TCNT equals OCR1x the comparator signals a match. A match will set the *Output Compare Flag* (OCF1x) at the next timer clock cycle. If enabled (OCIE1x = 1), the Output Compare Flag generates an output compare interrupt. The OCF1x Flag is automatically cleared when the interrupt is executed. Alternatively the OCF1x Flag can

be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the *Waveform Generation mode* (WGM13:0) bits and *Compare Output mode* (COM1x1:0) bits. The TOP and BOTTOM signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (See “Modes of Operation” on page 95.)

A special feature of output compare unit A allows it to define the Timer/Counter TOP value (i.e., counter resolution). In addition to the counter resolution, the TOP value defines the period time for waveforms generated by the Waveform Generator.

Figure 43 shows a block diagram of the output compare unit. The small “n” in the register and bit names indicates the device number (n = 1 for Timer/Counter1), and the “x” indicates output compare unit (A/B). The elements of the block diagram that are not directly a part of the output compare unit are gray shaded.

Figure 43. Output Compare Unit, Block Diagram



The OCR1x Register is double buffered when using any of the twelve *Pulse Width Modulation* (PWM) modes. For the normal and *Clear Timer on Compare* (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR1x Compare Register to either TOP or BOTTOM of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR1x Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR1x Buffer Register, and if double buffering is disabled the CPU will access the OCR1x directly. The content of the OCR1x (Buffer or Compare) Register is only changed by a write operation (the Timer/Counter does not update this register automatically as the TCNT1 and ICR1 Register). Therefore OCR1x is not read via the high byte temporary register (TEMP). However, it is a good practice to read the low byte first as when accessing other 16-bit registers. Writing the OCR1x Registers must be done via the TEMP Register since the compare of all 16 bits is done continuously. The high byte (OCR1xH) has to be written first. When the high

byte I/O location is written by the CPU, the TEMP Register will be updated by the value written. Then when the low byte (OCR1xL) is written to the lower eight bits, the high byte will be copied into the upper 8-bits of either the OCR1x buffer or OCR1x Compare Register in the same system clock cycle.

For more information of how to access the 16-bit registers refer to “Accessing 16-bit Registers” on page 87.

Force Output Compare

In non-PWM Waveform Generation modes, the match output of the comparator can be forced by writing a one to the *Force Output Compare* (FOC1x) bit. Forcing compare match will not set the OCF1x Flag or reload/clear the timer, but the OC1x pin will be updated as if a real compare match had occurred (the COM11:0 bits settings define whether the OC1x pin is set, cleared or toggled).

Compare Match Blocking by TCNT1 Write

All CPU writes to the TCNT1 Register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR1x to be initialized to the same value as TCNT1 without triggering an interrupt when the Timer/Counter clock is enabled.

Using the Output Compare Unit

Since writing TCNT1 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT1 when using any of the output compare channels, independent of whether the Timer/Counter is running or not. If the value written to TCNT1 equals the OCR1x value, the compare match will be missed, resulting in incorrect waveform generation. Do not write the TCNT1 equal to TOP in PWM modes with variable TOP values. The compare match for the TOP will be ignored and the counter will continue to 0xFFFF. Similarly, do not write the TCNT1 value equal to BOTTOM when the counter is downcounting.

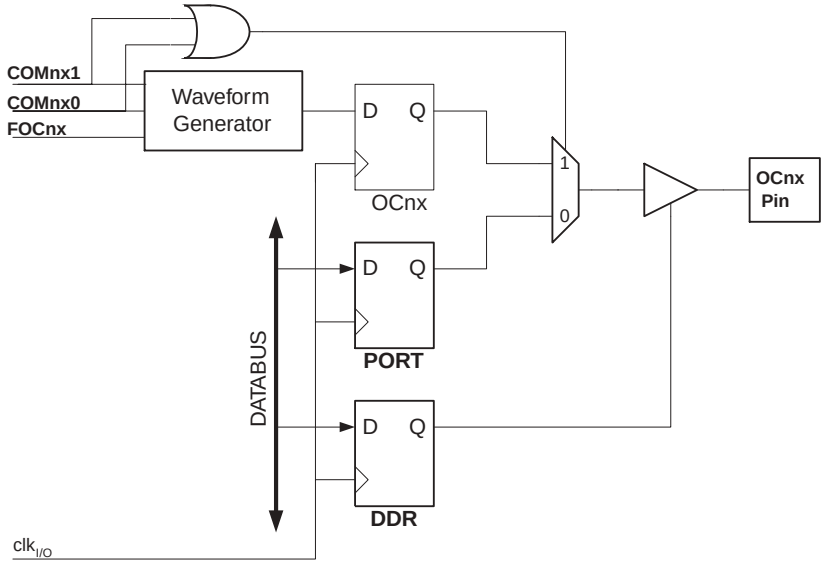
The setup of the OC1x should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC1x value is to use the force output compare (FOC1x) strobe bits in Normal mode. The OC1x Register keeps its value even when changing between waveform generation modes.

Be aware that the COM1x1:0 bits are not double buffered together with the compare value. Changing the COM1x1:0 bits will take effect immediately.

Compare Match Output Unit

The *Compare Output mode* (COM1x1:0) bits have two functions. The Waveform Generator uses the COM1x1:0 bits for defining the Output Compare (OC1x) state at the next compare match. Secondly the COM1x1:0 bits control the OC1x pin output source. Figure 44 shows a simplified schematic of the logic affected by the COM1x1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O Port Control Registers (DDR and PORT) that are affected by the COM1x1:0 bits are shown. When referring to the OC1x state, the reference is for the internal OC1x Register, not the OC1x pin. If a System Reset occur, the OC1x Register is reset to “0”.

Figure 44. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OC1x) from the Waveform Generator if either of the COM1x1:0 bits are set. However, the OC1x pin direction (input or output) is still controlled by the *Data Direction Register* (DDR) for the port pin. The Data Direction Register bit for the OC1x pin (DDR_OC1x) must be set as output before the OC1x value is visible on the pin. The port override function is generally independent of the Waveform Generation mode, but there are some exceptions. Refer to Table 44, Table 45 and Table 46 for details.

The design of the output compare pin logic allows initialization of the OC1x state before the output is enabled. Note that some COM1x1:0 bit settings are reserved for certain modes of operation. See “16-bit Timer/Counter Register Description” on page 105.

The COM1x1:0 bits have no effect on the input capture unit.

Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM1x1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM1x1:0 = 0 tells the Waveform Generator that no action on the OC1x Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 44 on page 105. For fast PWM mode refer to Table 45 on page 106, and for phase correct and phase and frequency correct PWM refer to Table 46 on page 106.

A change of the COM1x1:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC1x strobe bits.

Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the output compare pins, is defined by the combination of the *Waveform Generation mode* (WGM13:0) and *Compare Output mode* (COM1x1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM1x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM1x1:0 bits control whether the output should be set, cleared or toggle at a compare match (See “Compare Match Output Unit” on page 94.)

For detailed timing information refer to “Timer/Counter Timing Diagrams” on page 103.

Normal Mode

The simplest mode of operation is the *Normal* mode (WGM13:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the *Timer/Counter Overflow Flag* (TOV1) will be set in the same timer clock cycle as the TCNT1 becomes zero. The TOV1 Flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV1 Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

The input capture unit is easy to use in Normal mode. However, observe that the maximum interval between the external events must not exceed the resolution of the counter. If the interval between events are too long, the timer overflow interrupt or the prescaler must be used to extend the resolution for the capture unit.

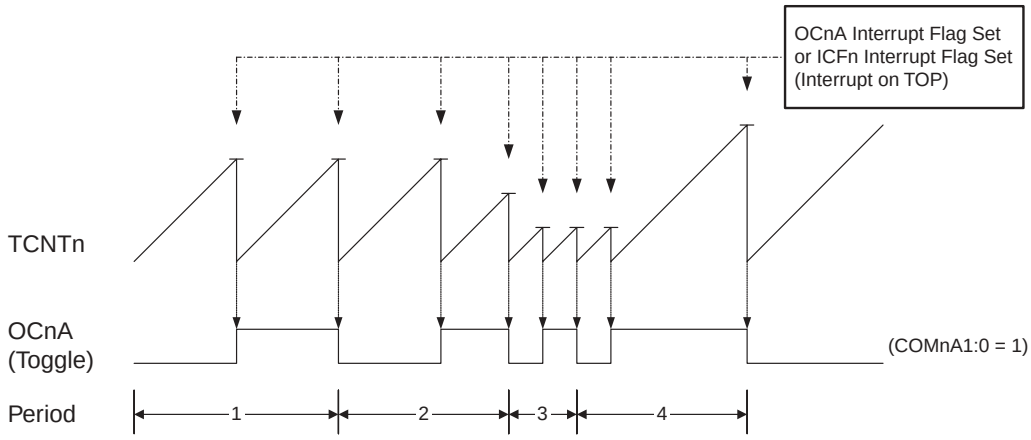
The output compare units can be used to generate interrupts at some given time. Using the output compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

Clear Timer on Compare Match (CTC) Mode

In *Clear Timer on Compare* or CTC mode (WGM13:0 = 4 or 12), the OCR1A or ICR1 Register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT1) matches either the OCR1A (WGM13:0 = 4) or the ICR1 (WGM13:0 = 12). The OCR1A or ICR1 define the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 45. The counter value (TCNT1) increases until a compare match occurs with either OCR1A or ICR1, and then counter (TCNT1) is cleared.

Figure 45. CTC Mode, Timing Diagram



An interrupt can be generated at each time the counter value reaches the TOP value by either using the OCF1A or ICF1 Flag according to the register used to define the TOP value. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR1A or ICR1 is lower than the current value of TCNT1, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. In many cases

this feature is not desirable. An alternative will then be to use the fast PWM mode using OCR1A for defining TOP (WGM13:0 = 15) since the OCR1A then will be double buffered.

For generating a waveform output in CTC mode, the OC1A output can be set to toggle its logical level on each compare match by setting the compare output mode bits to toggle mode (COM1A1:0 = 1). The OC1A value will not be visible on the port pin unless the data direction for the pin is set to output (DDR_OC1A = 1). The waveform generated will have a maximum frequency of $f_{OC1A} = f_{clk_I/O}/2$ when OCR1A is set to zero (0x0000). The waveform frequency is defined by the following equation:

$$f_{OCnA} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

The N variable represents the prescaler factor (1, 8, 64, 256, or 1024).

As for the Normal mode of operation, the TOV1 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x0000.

Fast PWM Mode

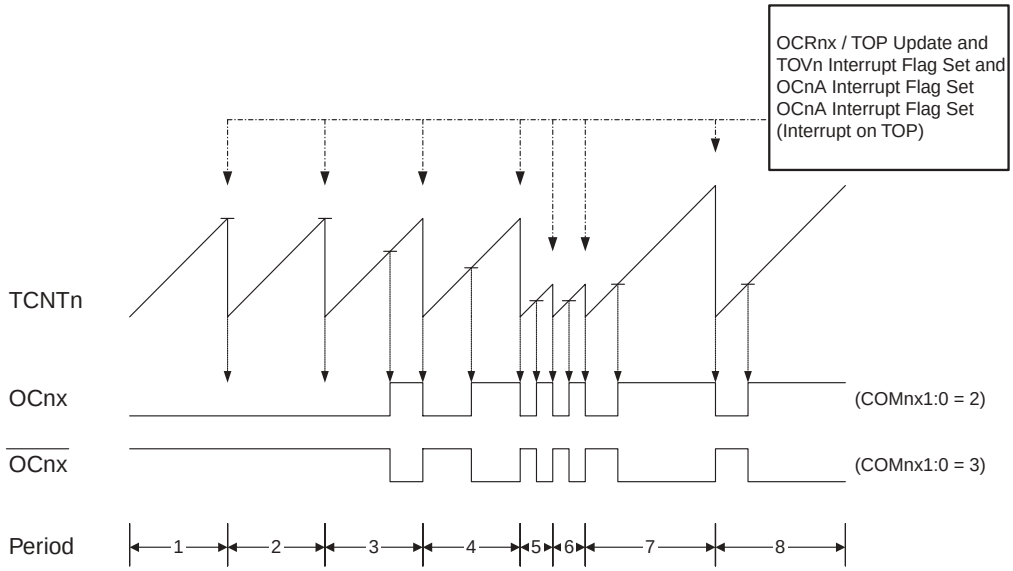
The *fast Pulse Width Modulation* or fast PWM mode (WGM13:0 = 5,6,7,14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is set on the compare match between TCNT1 and OCR1x, and cleared at TOP. In inverting Compare Output mode output is cleared on compare match and set at TOP. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct and phase and frequency correct PWM modes that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), hence reduces total system cost.

The PWM resolution for fast PWM can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{FPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM13:0 = 5, 6, or 7), the value in ICR1 (WGM13:0 = 14), or the value in OCR1A (WGM13:0 = 15). The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 46. The figure shows fast PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x Interrupt Flag will be set when a compare match occurs.

Figure 46. Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set each time the counter reaches TOP. In addition the OC1A or ICF1 Flag is set at the same timer clock cycle as TOV1 is set when either OCR1A or ICR1 is used for defining the TOP value. If one of the interrupts are enabled, the interrupt handler routine can be used for updating the TOP and compare values.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values the unused bits are masked to zero when any of the OCR1x Registers are written.

The procedure for updating ICR1 differs from updating OCR1A when used for defining the TOP value. The ICR1 Register is not double buffered. This means that if ICR1 is changed to a low value when the counter is running with none or a low prescaler value, there is a risk that the new ICR1 value written is lower than the current value of TCNT1. The result will then be that the counter will miss the compare match at the TOP value. The counter will then have to count to the MAX value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. The OCR1A Register however, is double buffered. This feature allows the OCR1A I/O location to be written anytime. When the OCR1A I/O location is written the value written will be put into the OCR1A Buffer Register. The OCR1A Compare Register will then be updated with the value in the Buffer Register at the next timer clock cycle the TCNT1 matches TOP. The update is done at the same timer clock cycle as the TCNT1 is cleared and the TOV1 Flag is set.

Using the ICR1 Register for defining TOP works well when using fixed TOP values. By using ICR1, the OCR1A Register is free to be used for generating a PWM output on OC1A. However, if the base PWM frequency is actively changed (by changing the TOP value), using the OCR1A as TOP is clearly a better choice due to its double buffer feature.

In fast PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to 2 will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to 3 (See Table 44 on page 105). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by

seting (or clearing) the OC1x Register at the compare match between OCR1x and TCNT1, and clearing (or setting) the OC1x Register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot (1 + TOP)}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCR1x is set equal to BOTTOM (0x0000) the output will be a narrow spike for each TOP+1 timer clock cycle. Setting the OCR1x equal to TOP will result in a constant high or low output (depending on the polarity of the output set by the COM1x1:0 bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC1A to toggle its logical level on each compare match (COM1A1:0 = 1). This applies only if OCR1A is used to define the TOP value (WGM13:0 = 15). The waveform generated will have a maximum frequency of $f_{OC1A} = f_{clk_I/O}/2$ when OCR1A is set to zero (0x0000). This feature is similar to the OC1A toggle in CTC mode, except the double buffer feature of the output compare unit is enabled in the fast PWM mode.

Phase Correct PWM Mode

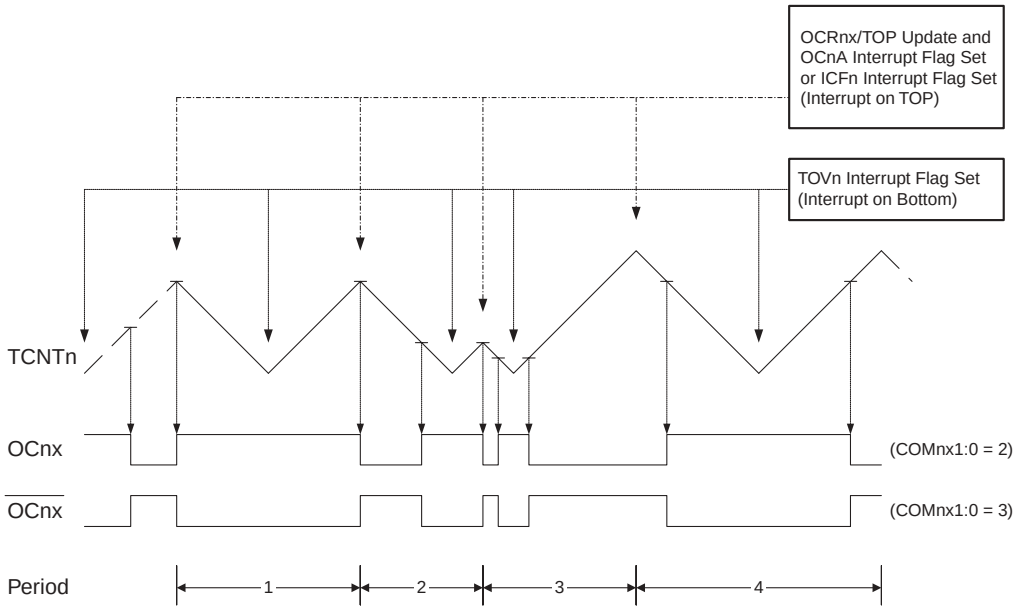
The *phase correct Pulse Width Modulation* or phase correct PWM mode (WGM13:0 = 1,2,3,10, or 11) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is, like the phase and frequency correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is cleared on the compare match between TCNT1 and OCR1x while upcounting, and set on the compare match while downcounting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase correct PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM13:0 = 1, 2, or 3), the value in ICR1 (WGM13:0 = 10), or the value in OCR1A (WGM13:0 = 11). The counter has then reached the TOP and changes the count direction. The TCNT1 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 47. The figure shows phase correct PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x Interrupt Flag will be set when a compare match occurs.

Figure 47. Phase Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set each time the counter reaches BOTTOM. When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 Flag is set accordingly at the same timer clock cycle as the OCR1x Registers are updated with the double buffer value (at TOP). The Interrupt Flags can be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values, the unused bits are masked to zero when any of the OCR1x Registers are written. As the third period shown in Figure 47 illustrates, changing the TOP actively while the Timer/Counter is running in the phase correct mode can result in an unsymmetrical output. The reason for this can be found in the time of update of the OCR1x Register. Since the OCR1x update occurs at TOP, the PWM period starts and ends at TOP. This implies that the length of the falling slope is determined by the previous TOP value, while the length of the rising slope is determined by the new TOP value. When these two values differ the two slopes of the period will differ in length. The difference in length gives the unsymmetrical result on the output.

It is recommended to use the phase and frequency correct mode instead of the phase correct mode when changing the TOP value while the Timer/Counter is running. When using a static TOP value there are practically no differences between the two modes of operation.

In phase correct PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to 2 will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to 3 (See Table 44 on page 105). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x Register at the compare match between OCR1x and TCNT1 when the counter increments, and clearing (or setting) the OC1x Register at compare match between OCR1x and TCNT1 when the counter decrements. The PWM frequency

for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR1x is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 11) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

Phase and Frequency Correct PWM Mode

The *phase and frequency correct Pulse Width Modulation,* or phase and frequency correct PWM mode (WGM13:0 = 8 or 9) provides a high resolution phase and frequency correct PWM waveform generation option. The phase and frequency correct PWM mode is, like the phase correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is cleared on the compare match between TCNT1 and OCR1x while upcounting, and set on the compare match while downcounting. In inverting Compare Output mode, the operation is inverted. The dual-slope operation gives a lower maximum operation frequency compared to the single-slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

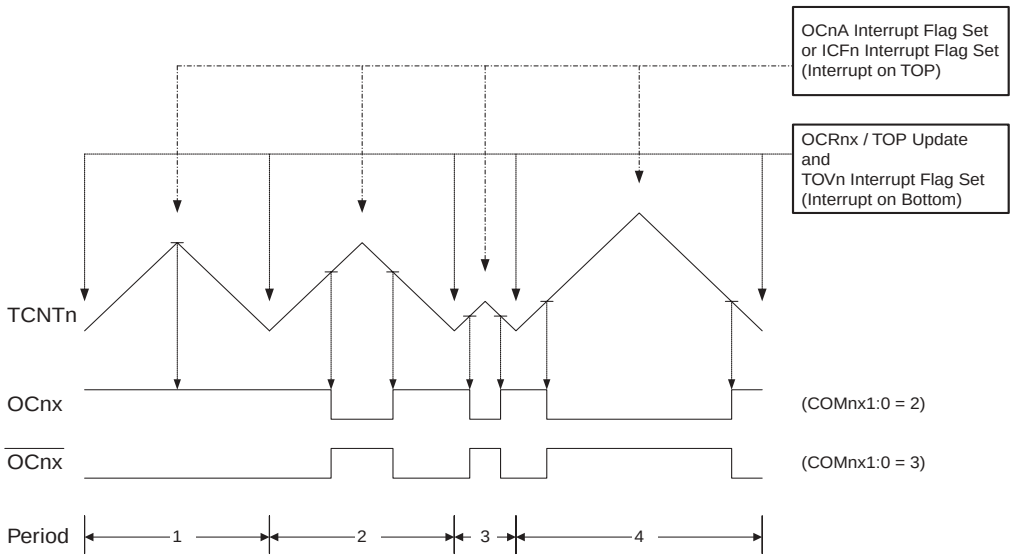
The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCR1x Register is updated by the OCR1x Buffer Register, (see Figure 47 and Figure 48).

The PWM resolution for the phase and frequency correct PWM mode can be defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated using the following equation:

$$R_{PFCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase and frequency correct PWM mode the counter is incremented until the counter value matches either the value in ICR1 (WGM13:0 = 8), or the value in OCR1A (WGM13:0 = 9). The counter has then reached the TOP and changes the count direction. The TCNT1 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct and frequency correct PWM mode is shown on Figure 48. The figure shows phase and frequency correct PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x Interrupt Flag will be set when a compare match occurs.

Figure 48. Phase and Frequency Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set at the same timer clock cycle as the OCR1x Registers are updated with the double buffer value (at BOTTOM). When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 Flag set when TCNT1 has reached TOP. The Interrupt Flags can then be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNT1 and the OCR1x.

As Figure 48 shows the output generated is, in contrast to the phase correct mode, symmetrical in all periods. Since the OCR1x Registers are updated at BOTTOM, the length of the rising and the falling slopes will always be equal. This gives symmetrical output pulses and is therefore frequency correct.

Using the ICR1 Register for defining TOP works well when using fixed TOP values. By using ICR1, the OCR1A Register is free to be used for generating a PWM output on OC1A. However, if the base PWM frequency is actively changed by changing the TOP value, using the OCR1A as TOP is clearly a better choice due to its double buffer feature.

In phase and frequency correct PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to 2 will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to 3 (See Table on page 106). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x Register at the compare match between OCR1x and TCNT1 when the counter increments, and clearing (or setting) the OC1x Register at compare match between OCR1x and TCNT1 when the counter decrements. The PWM frequency for the output when using phase and frequency correct PWM can be calculated by the following equation:

$$f_{OCnxPFCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represents special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR1x is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be set to high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 9) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{T1}) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set, and when the OCR1x Register is updated with the OCR1x buffer value (only for modes utilizing double buffering). Figure 49 shows a timing diagram for the setting of OCF1x.

Figure 49. Timer/Counter Timing Diagram, Setting of OCF1x, No Prescaling

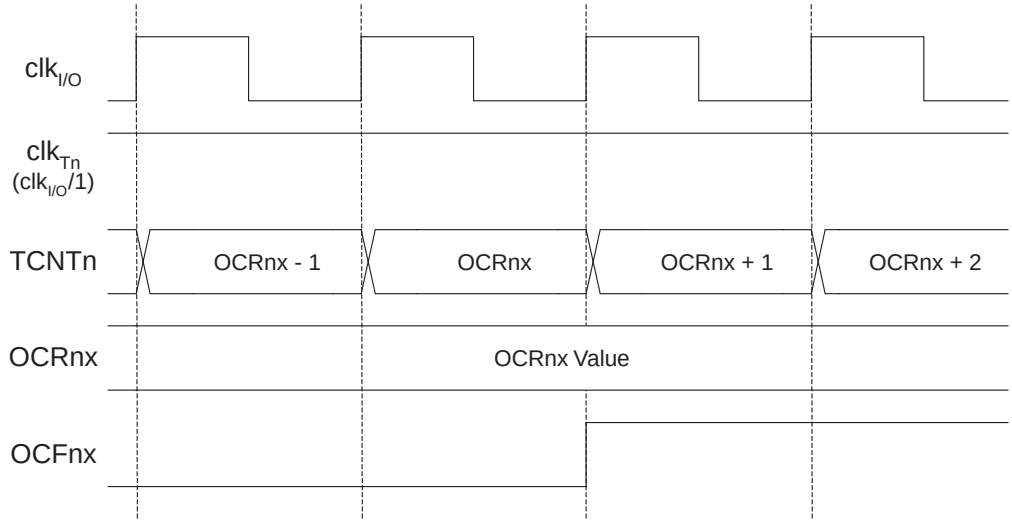


Figure 50 shows the same timing data, but with the prescaler enabled.

Figure 50. Timer/Counter Timing Diagram, Setting of OCF1x, with Prescaler ($f_{clk_I/O}/8$)

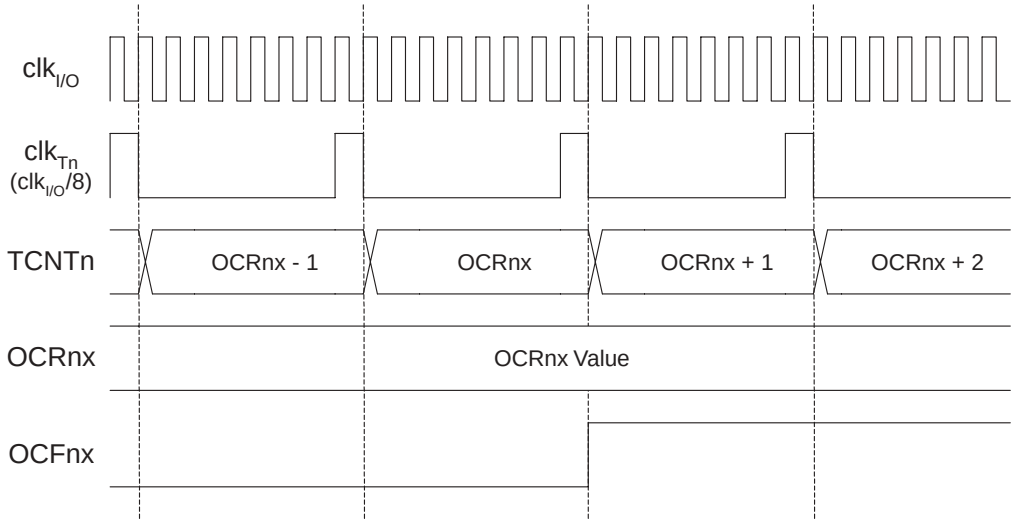


Figure 51 shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode the OCR1x Register is updated at BOTTOM. The timing diagrams will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that set the TOV1 Flag at BOTTOM.

Figure 51. Timer/Counter Timing Diagram, no Prescaling

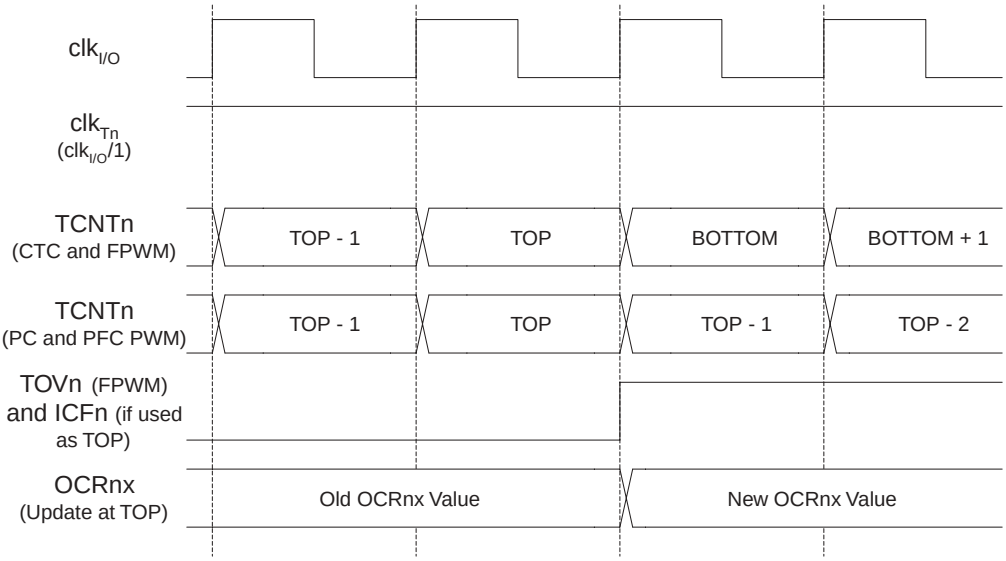
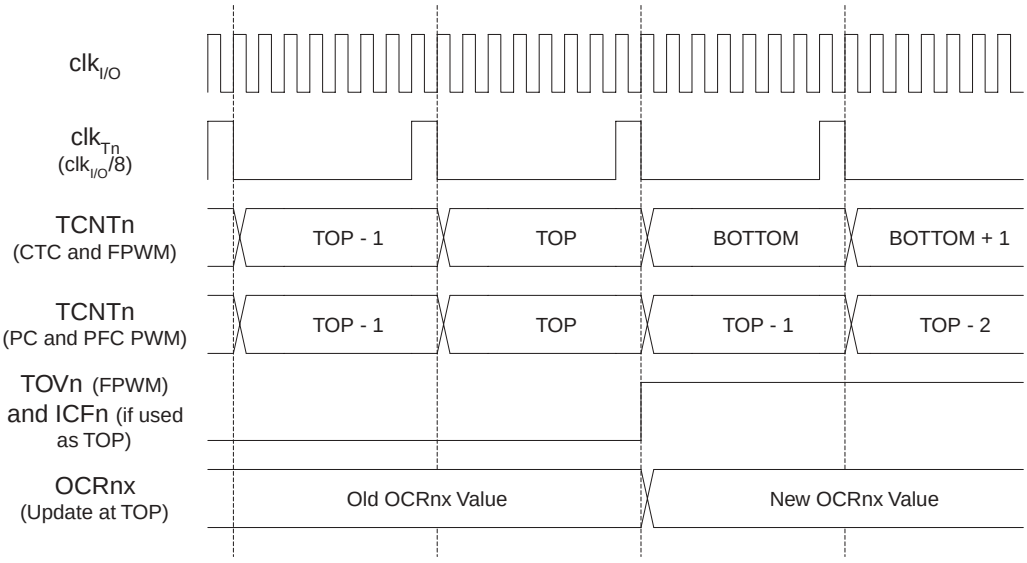


Figure 52 shows the same timing data, but with the prescaler enabled.

Figure 52. Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)



16-bit Timer/Counter Register Description

Timer/Counter1 Control Register A – TCCR1A

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – COM1A1:0: Compare Output Mode for Channel A**

- **Bit 5:4 – COM1B1:0: Compare Output Mode for Channel B**

The COM1A1:0 and COM1B1:0 control the Output Compare pins (OC1A and OC1B respectively) behavior. If one or both of the COM1A1:0 bits are written to one, the OC1A output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COM1B1:0 bit are written to one, the OC1B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the *Data Direction Register* (DDR) bit corresponding to the OC1A or OC1B pin must be set in order to enable the output driver.

When the OC1A or OC1B is connected to the pin, the function of the COM1x1:0 bits is dependent of the WGM13:0 bits setting. Table 44 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to a normal or a CTC mode (non-PWM).

Table 44. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on compare match
1	0	Clear OC1A/OC1B on compare match (Set output to low level)
1	1	Set OC1A/OC1B on compare match (Set output to high level)

Table 45 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the fast PWM mode.

Table 45. Compare Output Mode, Fast PWM[1]

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM13:0 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on compare match, set OC1A/OC1B at TOP
1	1	Set OC1A/OC1B on compare match, clear OC1A/OC1B at TOP

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. In this case the compare match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 97. for more details.

Table 46 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the phase correct or the phase and frequency correct, PWM mode.

Table 46. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM [1]

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 9 or 14: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM13:0 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on compare match when up-counting. Set OC1A/OC1B on compare match when downcounting.
1	1	Set OC1A/OC1B on compare match when up-counting. Clear OC1A/OC1B on compare match when downcounting.

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. See “Phase Correct PWM Mode” on page 99. for more details.

- **Bit 3 – FOC1A: Force Output Compare for Channel A**
- **Bit 2 – FOC1B: Force Output Compare for Channel B**

The FOC1A/FOC1B bits are only active when the WGM13:0 bits specifies a non-PWM mode. However, for ensuring compatibility with future devices, these bits must be set to zero when TCCR1A is written when operating in a PWM mode. When writing a logical one to the FOC1A/FOC1B bit, an immediate compare match is forced on the Waveform Generation unit. The OC1A/OC1B output is changed according to its COM1x1:0 bits setting. Note that the FOC1A/FOC1B bits are implemented as strobes. Therefore it is the value present in the COM1x1:0 bits that determine the effect of the forced compare.

A FOC1A/FOC1B strobe will not generate any interrupt nor will it clear the timer in Clear Timer on Compare match (CTC) mode using OCR1A as TOP.

The FOC1A/FOC1B bits are always read as zero.

- **Bit 1:0 – WGM11:0: Waveform Generation Mode**

Combined with the WGM13:2 bits found in the TCCR1B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 47. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. (See “Modes of Operation” on page 95.)

Table 47. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Timer/Counter1 Control Register B – TCCR1B

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNC1: Input Capture Noise Canceler**

Setting this bit (to one) activates the Input Capture Noise Canceler. When the Noise Canceler is activated, the input from the Input Capture Pin (ICP1) is filtered. The filter function requires four successive equal valued samples of the ICP1 pin for changing its output. The input capture is therefore delayed by four Oscillator cycles when the Noise Canceler is enabled.

- **Bit 6 – ICES1: Input Capture Edge Select**

This bit selects which edge on the Input Capture Pin (ICP1) that is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICES1 setting, the counter value is copied into the Input Capture Register (ICR1). The event will also set the Input Capture Flag (ICF1), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

When the ICR1 is used as TOP value (see description of the WGM13:0 bits located in the TCCR1A and the TCCR1B Register), the ICP1 is disconnected and consequently the input capture function is disabled.

- **Bit 5 – Reserved Bit**

This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCR1B is written.

- **Bit 4:3 – WGM13:2: Waveform Generation Mode**

See TCCR1A Register description.

- **Bit 2:0 – CS12:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter, see Figure 49 and Figure 50.

Table 48. Clock Select Bit Description

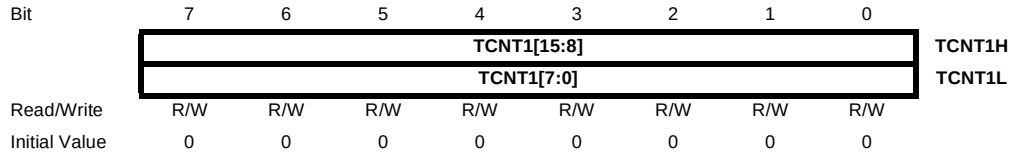
CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)

Table 48. Clock Select Bit Description (Continued)

CS12	CS11	CS10	Description
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter1, transitions on the T1 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

Timer/Counter1 – TCNT1H and TCNT1L

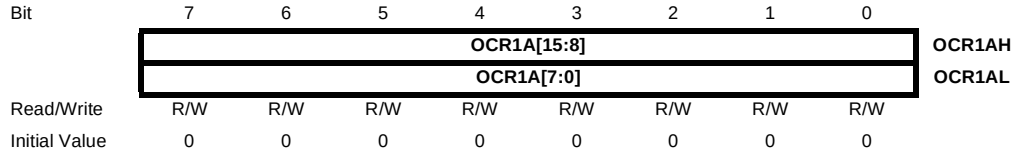


The two *Timer/Counter* I/O locations (TCNT1H and TCNT1L, combined TCNT1) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 87.

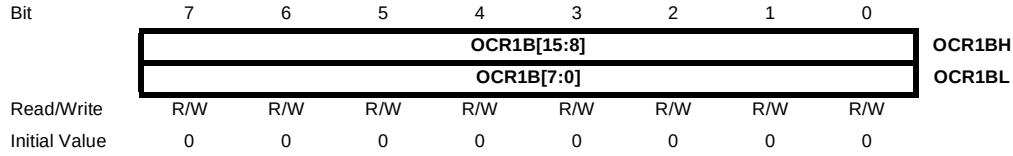
Modifying the counter (TCNT1) while the counter is running introduces a risk of missing a compare match between TCNT1 and one of the OCR1x Registers.

Writing to the TCNT1 Register blocks (removes) the compare match on the following timer clock for all compare units.

Output Compare Register 1 A – OCR1AH and OCR1AL



Output Compare Register 1 B – OCR1BH and OCR1BL



The Output Compare Registers contain a 16-bit value that is continuously compared with the counter value (TCNT1). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC1x pin.

The Output Compare Registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 87.

Input Capture Register 1 – ICR1H and ICR1L

Bit	7	6	5	4	3	2	1	0	
	ICR1[15:8]								ICR1H
	ICR1[7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Input Capture is updated with the counter (TCNT1) value each time an event occurs on the ICP1 pin (or optionally on the analog comparator output for Timer/Counter1). The Input Capture can be used for defining the counter TOP value.

The Input Capture Register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 87.

Timer/Counter Interrupt Mask Register – TIMSK(1)

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: 1. This register contains interrupt control bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.

- **Bit 5 – TICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture Interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 42.) is executed when the ICF1 Flag, located in TIFR, is set.

- **Bit 4 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare A match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 42.) is executed when the OCF1A Flag, located in TIFR, is set.

- **Bit 3 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare B match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 42.) is executed when the OCF1B Flag, located in TIFR, is set.

- **Bit 2 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Overflow Interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 42.) is executed when the TOV1 Flag, located in TIFR, is set.

Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: This register contains flag bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.

- **Bit 5 – ICF1: Timer/Counter1, Input Capture Flag**

This flag is set when a capture event occurs on the ICP1 pin. When the Input Capture Register (ICR1) is set by the WGM13:0 to be used as the TOP value, the ICF1 Flag is set when the counter reaches the TOP value.

ICF1 is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

- **Bit 4 – OCF1A: Timer/Counter1, Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register A (OCR1A).

Note that a Forced Output Compare (FOC1A) strobe will not set the OCF1A Flag.

OCF1A is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCF1A can be cleared by writing a logic one to its bit location.

- **Bit 3 – OCF1B: Timer/Counter1, Output Compare B Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register B (OCR1B).

Note that a forced output compare (FOC1B) strobe will not set the OCF1B Flag.

OCF1B is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCF1B can be cleared by writing a logic one to its bit location.

- **Bit 2 – TOV1: Timer/Counter1, Overflow Flag**

The setting of this flag is dependent of the WGM13:0 bits setting. In normal and CTC modes, the TOV1 Flag is set when the timer overflows. Refer to Table 47 on page 107 for the TOV1 Flag behavior when using another WGM13:0 bit setting.

TOV1 is automatically cleared when the Timer/Counter1 Overflow interrupt vector is executed. Alternatively, TOV1 can be cleared by writing a logic one to its bit location.

8-bit Timer/Counter2 with PWM and Asynchronous Operation

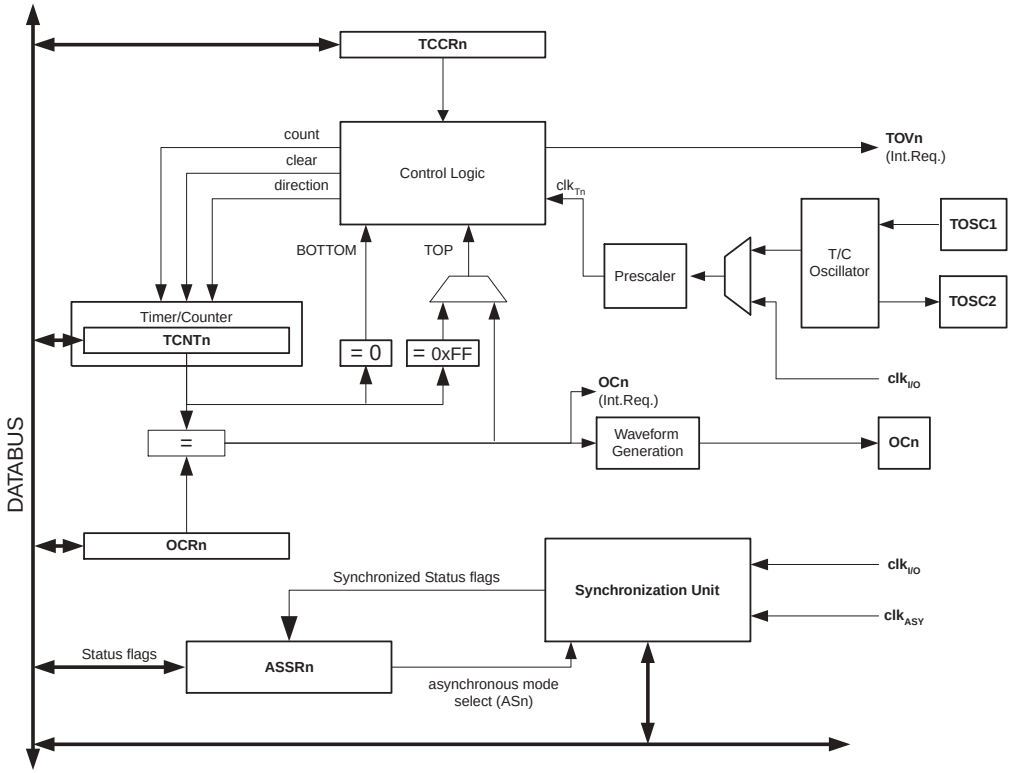
Timer/Counter2 is a general purpose, single channel, 8-bit Timer/Counter module. The main features are:

- **Single Channel Counter**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch-free, Phase Correct Pulse Width Modulator (PWM)**
- **Frequency Generator**
- **10-bit Clock Prescaler**
- **Overflow and Compare Match Interrupt Sources (TOV2 and OCF2)**
- **Allows clocking from External 32 kHz Watch Crystal Independent of the I/O Clock**

Overview

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 53. For the actual placement of I/O pins, refer to “Pinouts ATmega32” on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “8-bit Timer/Counter Register Description” on page 123.

Figure 53. 8-bit Timer/Counter Block Diagram



Registers

The Timer/Counter (TCNT2) and Output Compare Register (OCR2) are 8-bit registers. Interrupt request (shorten as Int.Req.) signals are all visible in the Timer Interrupt Flag Register (TIFR). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK). TIFR and TIMSK are not shown in the figure since these registers are shared by other timer units.

The Timer/Counter can be clocked internally, via the prescaler, or asynchronously clocked from the TOSC1/2 pins, as detailed later in this section. The asynchronous operation is controlled by the Asynchronous Status Register (ASSR). The Clock Select logic block controls which clock source the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock (clk_{T2}).

The double buffered Output Compare Register (OCR2) is compared with the Timer/Counter value at all times. The result of the compare can be used by the waveform generator to generate a PWM or variable frequency output on the Output Compare Pin (OC2). See “Output Compare Unit” on page 114. for details. The compare match event will also set the Compare Flag (OCF2) which can be used to generate an output compare interrupt request.

Definitions

Many register and bit references in this document are written in general form. A lower case “n” replaces the Timer/Counter number, in this case 2. However, when using the register or bit defines in a program, the precise form must be used (i.e., TCNT2 for accessing Timer/Counter2 counter value and so on). The definitions in Table 49 are also used extensively throughout the document.

Table 49. Definitions

BOTTOM	The counter reaches the BOTTOM when it becomes zero (0x00).
MAX	The counter reaches its MAXimum when it becomes 0xFF (decimal 255).
TOP	The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX) or the value stored in the OCR2 Register. The assignment is dependent on the mode of operation.

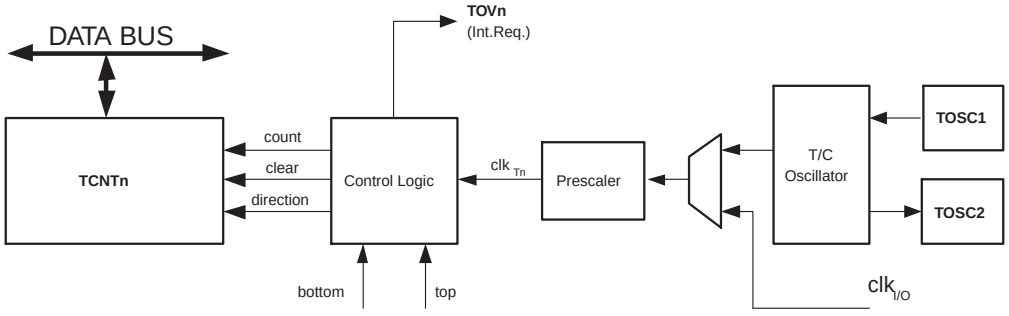
Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal synchronous or an external asynchronous clock source. The clock source clk_{T2} is by default equal to the MCU clock, $clk_{I/O}$. When the AS2 bit in the ASSR Register is written to logic one, the clock source is taken from the Timer/Counter Oscillator connected to TOSC1 and TOSC2. For details on asynchronous operation, see “Asynchronous Status Register – ASSR” on page 126. For details on clock sources and prescaler, see “Timer/Counter Prescaler” on page 129.

Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 54 shows a block diagram of the counter and its surrounding environment.

Figure 54. Counter Unit Block Diagram



Signal description (internal signals):

count Increment or decrement TCNT2 by 1.

direction Selects between increment and decrement.

clear Clear TCNT2 (set all bits to zero).

clk_{T2} Timer/Counter clock.

top Signalizes that TCNT2 has reached maximum value.

bottom Signalizes that TCNT2 has reached minimum value (zero).

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock (clk_{T2}). clk_{T2} can be generated from an external or internal clock source, selected by the Clock Select bits (CS22:0). When no clock source is selected (CS22:0 = 0) the timer is stopped. However, the TCNT2 value can be accessed by the CPU, regardless of whether clk_{T2} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

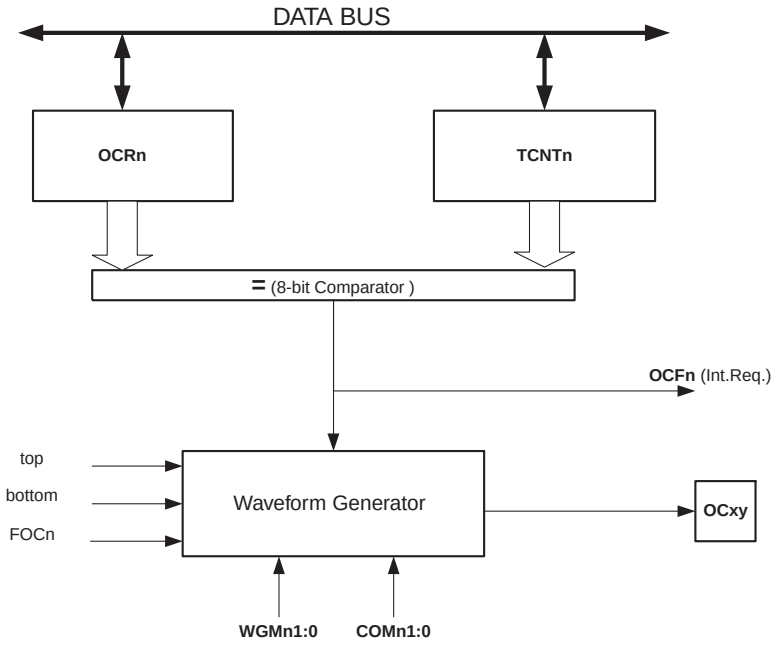
The counting sequence is determined by the setting of the WGM21 and WGM20 bits located in the Timer/Counter Control Register (TCCR2). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare output OC2. For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 116.

The Timer/Counter Overflow (TOV2) Flag is set according to the mode of operation selected by the WGM21:0 bits. TOV2 can be used for generating a CPU interrupt.

Output Compare Unit

The 8-bit comparator continuously compares TCNT2 with the Output Compare Register (OCR2). Whenever TCNT2 equals OCR2, the comparator signals a match. A match will set the Output Compare Flag (OCF2) at the next timer clock cycle. If enabled (OCIE2 = 1), the Output Compare Flag generates an output compare interrupt. The OCF2 Flag is automatically cleared when the interrupt is executed. Alternatively, the OCF2 Flag can be cleared by software by writing a logical one to its I/O bit location. The waveform generator uses the match signal to generate an output according to operating mode set by the WGM21:0 bits and Compare Output mode (COM21:0) bits. The max and bottom signals are used by the waveform generator for handling the special cases of the extreme values in some modes of operation (“Modes of Operation” on page 116). Figure 55 shows a block diagram of the output compare unit.

Figure 55. Output Compare Unit, Block Diagram



The OCR2 Register is double buffered when using any of the Pulse Width Modulation (PWM) modes. For the normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR2 Compare Register to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR2 Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR2 Buffer Register, and if double buffering is disabled the CPU will access the OCR2 directly.

Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (FOC2) bit. Forcing compare match will not set the OCF2 Flag or reload/clear the timer, but the OC2 pin will be updated as if a real compare match had occurred (the COM21:0 bits settings define whether the OC2 pin is set, cleared or toggled).

Compare Match Blocking by TCNT2 Write

All CPU write operations to the TCNT2 Register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR2 to be initialized to the same value as TCNT2 without triggering an interrupt when the Timer/Counter clock is enabled.

Using the Output Compare Unit

Since writing TCNT2 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT2 when using the output compare channel, independently of whether the Timer/Counter is running or not. If the value written to TCNT2 equals the OCR2 value, the compare match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT2 value equal to BOTTOM when the counter is downcounting.

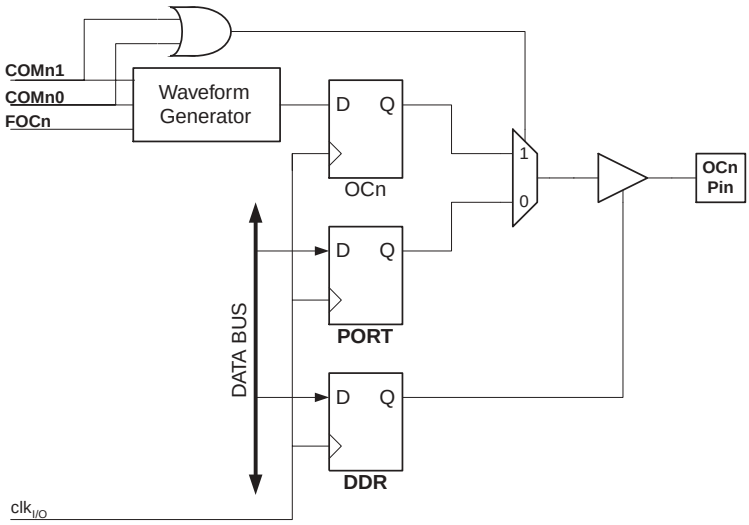
The setup of the OC2 should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC2 value is to use the Force Output Compare (FOC2) strobe bit in Normal mode. The OC2 Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COM21:0 bits are not double buffered together with the compare value. Changing the COM21:0 bits will take effect immediately.

Compare Match Output Unit

The Compare Output mode (COM21:0) bits have two functions. The Waveform Generator uses the COM21:0 bits for defining the Output Compare (OC2) state at the next compare match. Also, the COM21:0 bits control the OC2 pin output source. Figure 56 shows a simplified schematic of the logic affected by the COM21:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O Port Control Registers (DDR and PORT) that are affected by the COM21:0 bits are shown. When referring to the OC2 state, the reference is for the internal OC2 Register, not the OC2 pin.

Figure 56. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OC2) from the waveform generator if either of the COM21:0 bits are set. However, the OC2 pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OC2 pin (DDR_OC2) must be set as output before the OC2 value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the output compare pin logic allows initialization of the OC2 state before the output is enabled. Note that some COM21:0 bit settings are reserved for certain modes of operation. See “8-bit Timer/Counter Register Description” on page 123.

Compare Output Mode and Waveform Generation

The waveform generator uses the COM21:0 bits differently in Normal, CTC, and PWM modes. For all modes, setting the COM21:0 = 0 tells the Waveform Generator that no action on the OC2 Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 51 on page 124. For fast PWM mode, refer to Table 52 on page 124, and for phase correct PWM refer to Table 53 on page 124.

A change of the COM21:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC2 strobe bits.

Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the output compare pins, is defined by the combination of the Waveform Generation mode (WGM21:0) and Compare Output mode (COM21:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM21:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM21:0 bits control whether the output should be set, cleared, or toggled at a compare match (See “Compare Match Output Unit” on page 115.).

For detailed timing information refer to “Timer/Counter Timing Diagrams” on page 121.

Normal Mode

The simplest mode of operation is the Normal mode (WGM21:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV2) will be set in the same timer clock cycle as the TCNT2 becomes zero. The TOV2 Flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV2 Flag, the timer resolution can be increased by software. There are no special cases to consider in the normal mode, a new counter value can be written anytime.

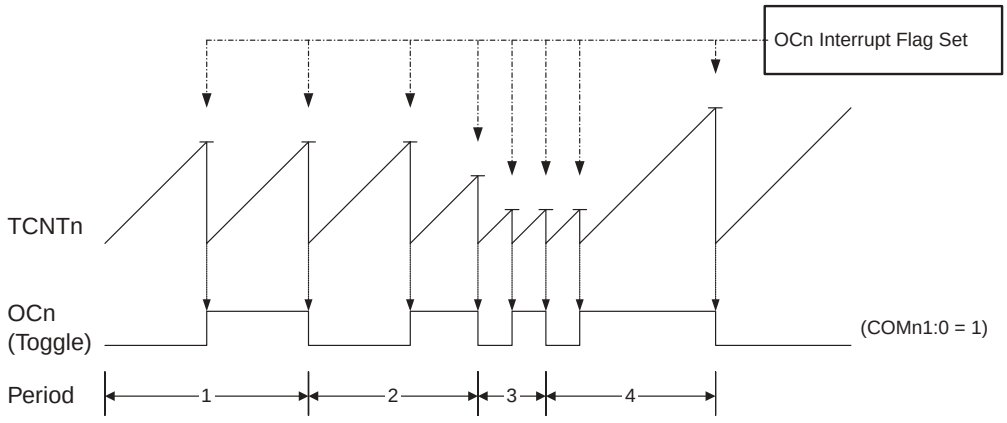
The Output Compare unit can be used to generate interrupts at some given time. Using the output compare to generate waveforms in normal mode is not recommended, since this will occupy too much of the CPU time.

Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM21:0 = 2), the OCR2 Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT2) matches the OCR2. The OCR2 defines the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 57. The counter value (TCNT2) increases until a compare match occurs between TCNT2 and OCR2, and then counter (TCNT2) is cleared.

Figure 57. CTC Mode, Timing Diagram



An interrupt can be generated each time the counter value reaches the TOP value by using the OCF2 Flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR2 is lower than the current value of TCNT2, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFF) and wrap around starting at 0x00 before the compare match can occur.

For generating a waveform output in CTC mode, the OC2 output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COM21:0 = 1). The OC2 value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum fre-

quency of $f_{OC2} = f_{clk_I/O}/2$ when OCR2 is set to zero (0x00). The waveform frequency is defined by the following equation:

$$f_{OCn} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRn)}$$

The N variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

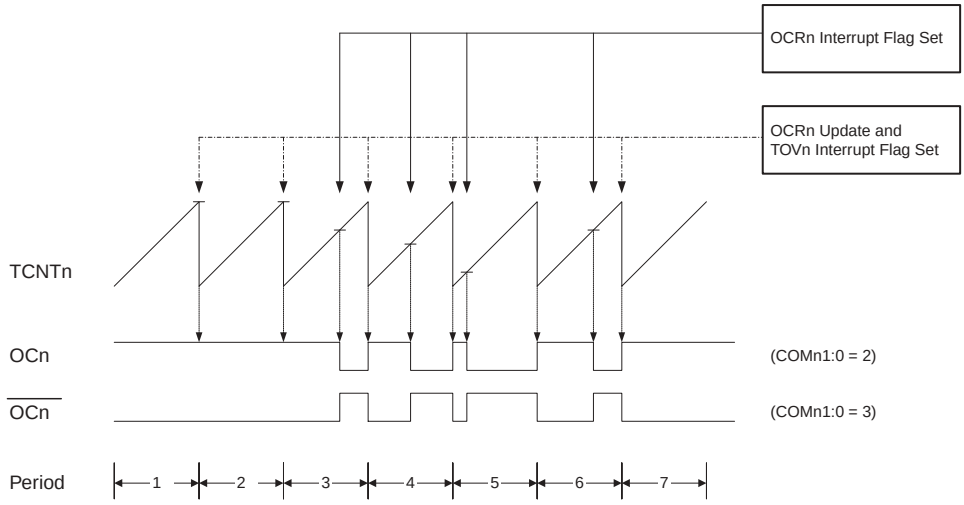
As for the Normal mode of operation, the TOV2 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM21:0 = 3) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to MAX then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC2) is cleared on the compare match between TCNT2 and OCR2, and set at BOTTOM. In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that uses dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the MAX value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 58. The TCNT2 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT2 slopes represent compare matches between OCR2 and TCNT2.

Figure 58. Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV2) is set each time the counter reaches MAX. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC2 pin. Setting the COM21:0 bits to 2 will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM21:0 to 3 (see Table 52 on page 124). The actual OC2 value will only be visible on the port pin if the data direction for the port

pin is set as output. The PWM waveform is generated by setting (or clearing) the OC2 Register at the compare match between OCR2 and TCNT2, and clearing (or setting) the OC2 Register at the timer clock cycle the counter is cleared (changes from MAX to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnPWM} = \frac{f_{clk_I/O}}{N \cdot 256}$$

The N variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

The extreme values for the OCR2 Register represent special cases when generating a PWM waveform output in the fast PWM mode. If the OCR2 is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR2 equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM21:0 bits.)

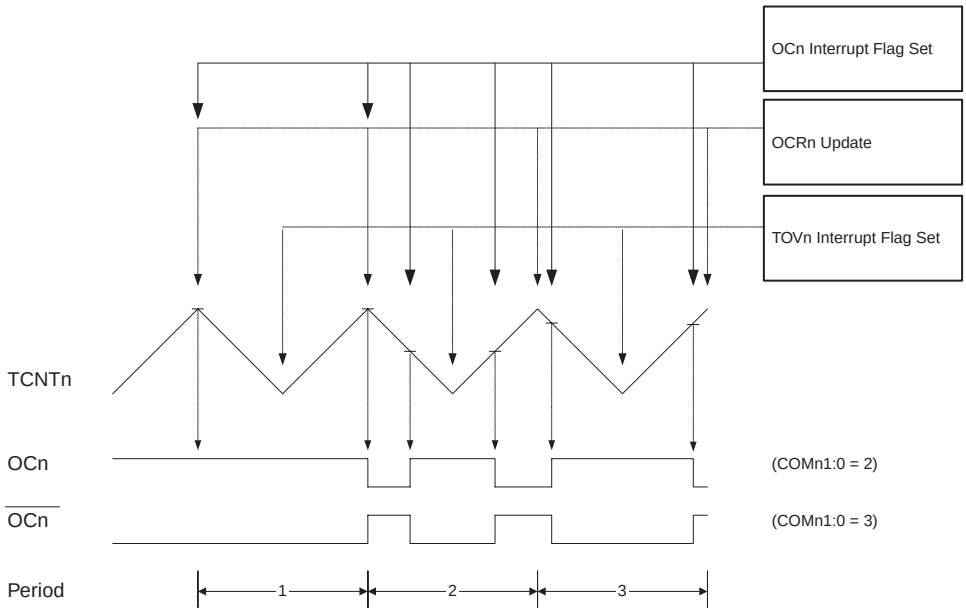
A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC2 to toggle its logical level on each compare match (COM21:0 = 1). The waveform generated will have a maximum frequency of $f_{oc2} = f_{clk_I/O}/2$ when OCR2 is set to zero. This feature is similar to the OC2 toggle in CTC mode, except the double buffer feature of the output compare unit is enabled in the fast PWM mode.

Phase Correct PWM Mode

The phase correct PWM mode (WGM21:0 = 1) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC2) is cleared on the compare match between TCNT2 and OCR2 while upcounting, and set on the compare match while downcounting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode is fixed to 8 bits. In phase correct PWM mode the counter is incremented until the counter value matches MAX. When the counter reaches MAX, it changes the count direction. The TCNT2 value will be equal to MAX for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 59. The TCNT2 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT2 slopes represent compare matches between OCR2 and TCNT2.

Figure 59. Phase Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV2) is set each time the counter reaches BOTTOM. The Interrupt Flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC2 pin. Setting the COM21:0 bits to 2 will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM21:0 to 3 (see Table 53 on page 124). The actual OC2 value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC2 Register at the compare match between OCR2 and TCNT2 when the counter increments, and setting (or clearing) the OC2 Register at compare match between OCR2 and TCNT2 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnPCPWM} = \frac{f_{clk_I/O}}{N \cdot 510}$$

The N variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

The extreme values for the OCR2 Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR2 is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

At the very start of period 2 in Figure 59 OCn has a transition from high to low even though there is no Compare Match. The point of this transition is to guarantee symmetry around BOTTOM. THere are two cases that give a transition without Compare Match.

- OCR2A chages its value from MAX, like in Figure 59. When the OCR2A value is MAX the OCn pin value is the same as the result of a down-counting Compare Match. To ensure symmetry around BOTTOM the OCn value at MAX must correspond to the result of an up-counting Compare Match.

- The timer starts counting from a value higher than the one in OCR2A, and for that reason misses the Compare Match and hence the OCn change that would have happened on the way up.

Timer/Counter Timing Diagrams

The following figures show the Timer/Counter in Synchronous mode, and the timer clock (clk_{T2}) is therefore shown as a clock enable signal. In Asynchronous mode, $clk_{I/O}$ should be replaced by the Timer/Counter Oscillator clock. The figures include information on when Interrupt Flags are set. Figure 60 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

Figure 60. Timer/Counter Timing Diagram, no Prescaling

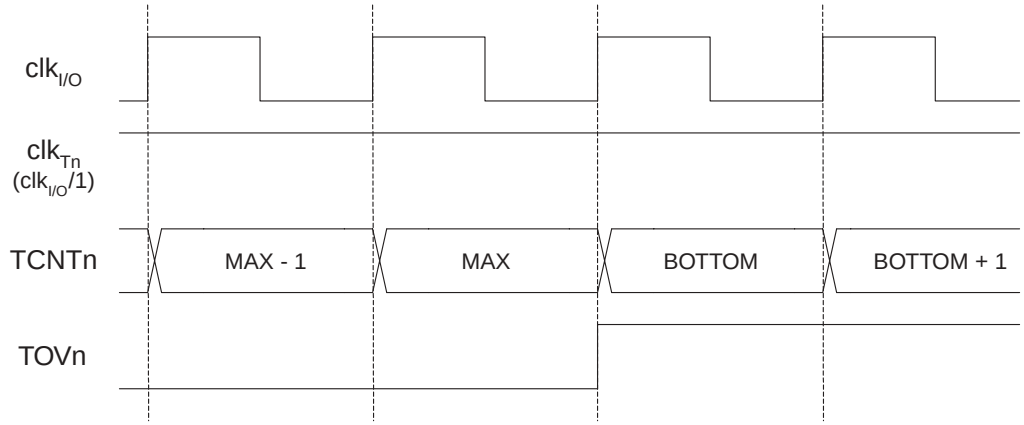


Figure 61 shows the same timing data, but with the prescaler enabled.

Figure 61. Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)

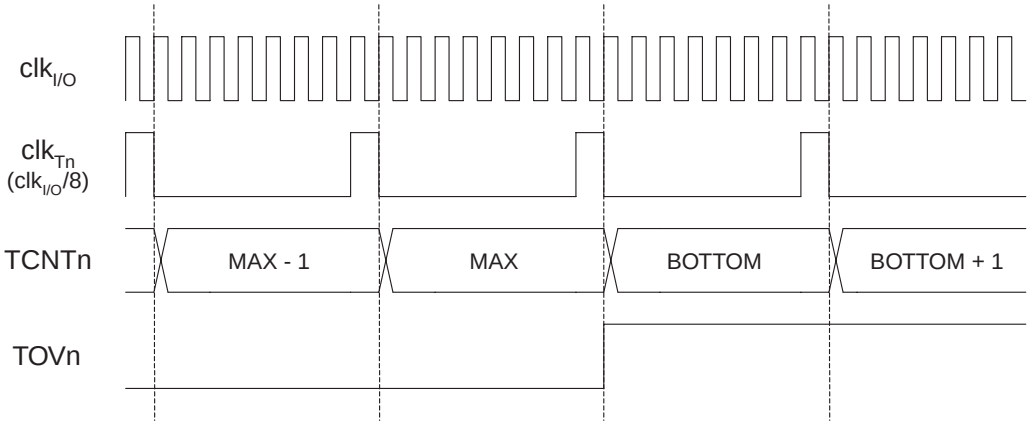


Figure 62 shows the setting of OCF2 in all modes except CTC mode.

Figure 62. Timer/Counter Timing Diagram, Setting of OCF2, with Prescaler ($f_{clk_I/O}/8$)

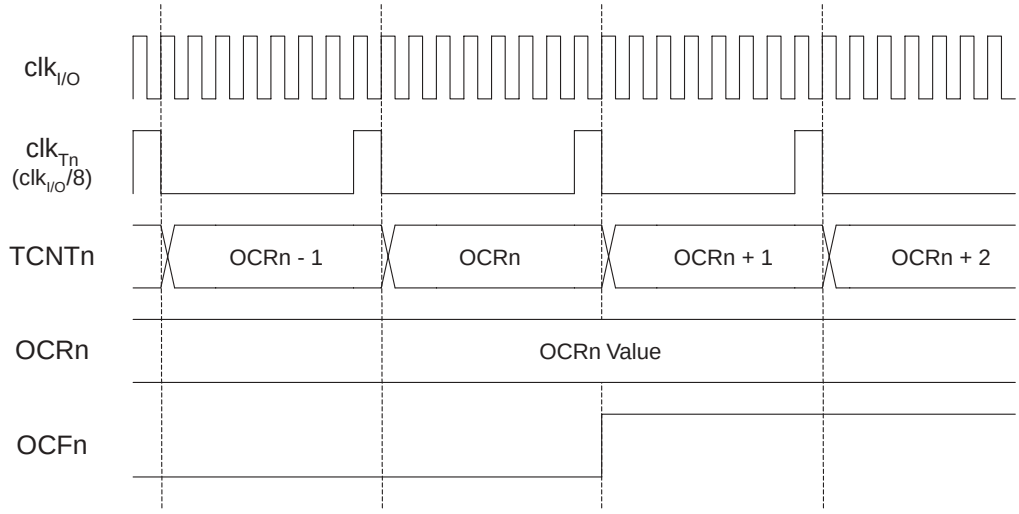
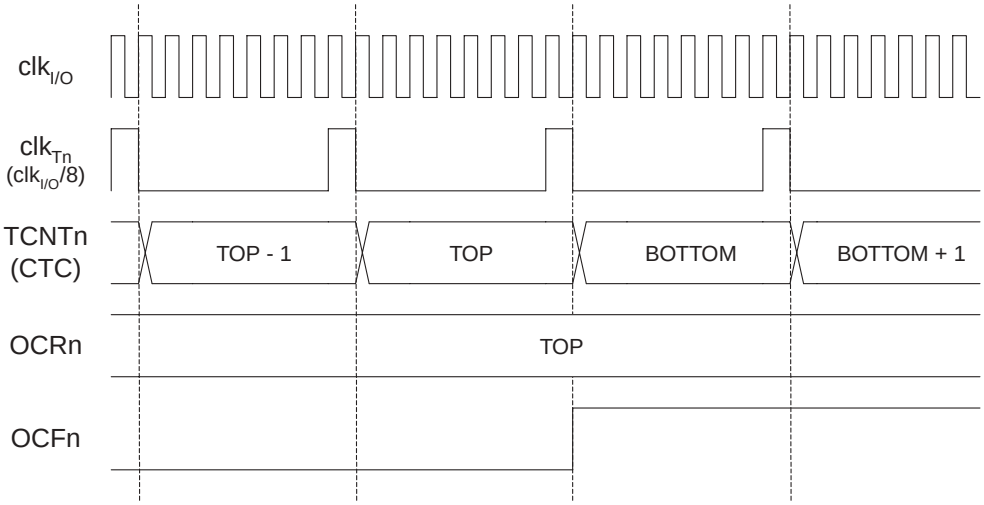


Figure 63 shows the setting of OCF2 and the clearing of TCNT2 in CTC mode.

Figure 63. Timer/Counter Timing Diagram, Clear Timer on Compare Match Mode, with Prescaler ($f_{clk_I/O}/8$)



8-bit Timer/Counter Register Description

Timer/Counter Control Register – TCCR2

Bit	7	6	5	4	3	2	1	0	
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	TCCR2
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC2: Force Output Compare**

The FOC2 bit is only active when the WGM bits specify a non-PWM mode. However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR2 is written when operating in PWM mode. When writing a logical one to the FOC2 bit, an immediate compare match is forced on the waveform generation unit. The OC2 output is changed according to its COM21:0 bits setting. Note that the FOC2 bit is implemented as a strobe. Therefore it is the value present in the COM21:0 bits that determines the effect of the forced compare.

A FOC2 strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR2 as TOP.

The FOC2 bit is always read as zero.

- **Bit 6, 3 – WGM21:0: Waveform Generation Mode**

These bits control the counting sequence of the counter, the source for the maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode, Clear Timer on Compare match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes. See Table 50 and “Modes of Operation” on page 116.

Table 50. Waveform Generation Mode Bit Description[1]

Mode	WGM21 (CTC2)	WGM20 (PWM2)	Timer/Counter Mode of Operation	TOP	Update of OCR2	TOV2 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR2	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Note: 1. The CTC2 and PWM2 bit definition names are now obsolete. Use the WGM21:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

- **Bit 5:4 – COM21:0: Compare Match Output Mode**

These bits control the Output Compare pin (OC2) behavior. If one or both of the COM21:0 bits are set, the OC2 output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to OC2 pin must be set in order to enable the output driver.

When OC2 is connected to the pin, the function of the COM21:0 bits depends on the WGM21:0 bit setting. Table 51 shows the COM21:0 bit functionality when the WGM21:0 bits are set to a normal or CTC mode (non-PWM).

Table 51. Compare Output Mode, non-PWM Mode

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Toggle OC2 on compare match
1	0	Clear OC2 on compare match
1	1	Set OC2 on compare match

Table 52 shows the COM21:0 bit functionality when the WGM21:0 bits are set to fast PWM mode.

Table 52. Compare Output Mode, Fast PWM Mode(1)

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on compare match, set OC2 at TOP
1	1	Set OC2 on compare match, clear OC2 at TOP

Note: 1. A special case occurs when OCR2 equals TOP and COM21 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 118 for more details.

Table 53 shows the COM21:0 bit functionality when the WGM21:0 bits are set to phase correct PWM mode

Table 53. Compare Output Mode, Phase Correct PWM Mode(1)

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on compare match when up-counting. Set OC2 on compare match when downcounting.
1	1	Set OC2 on compare match when up-counting. Clear OC2 on compare match when downcounting.

Note: 1. A special case occurs when OCR2 equals TOP and COM21 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Phase Correct PWM Mode” on page 119 for more details.

- **Bit 2:0 – CS22:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter, see Table 54.

Table 54. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler)
1	1	1	clk_{T2S}/1024 (From prescaler)

Timer/Counter Register – TCNT2

Bit	7	6	5	4	3	2	1	0	
	TCNT2[7:0]								TCNT2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT2 Register blocks (removes) the compare match on the following timer clock. Modifying the counter (TCNT2) while the counter is running, introduces a risk of missing a compare match between TCNT2 and the OCR2 Register.

Output Compare Register – OCR2

Bit	7	6	5	4	3	2	1	0	
	OCR2[7:0]								OCR2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Output Compare Register contains an 8-bit value that is continuously compared with the counter value (TCNT2). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC2 pin.

Asynchronous Operation of the Timer/Counter

Asynchronous Status Register – ASSR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	ASSR
Read/Write	R	R	R	R	R/W	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 3 – AS2: Asynchronous Timer/Counter2**

When AS2 is written to zero, Timer/Counter 2 is clocked from the I/O clock, $clk_{I/O}$. When AS2 is written to one, Timer/Counter2 is clocked from a Crystal Oscillator connected to the Timer Oscillator 1 (TOSC1) pin. When the value of AS2 is changed, the contents of TCNT2, OCR2, and TCCR2 might be corrupted.

- **Bit 2 – TCN2UB: Timer/Counter2 Update Busy**

When Timer/Counter2 operates asynchronously and TCNT2 is written, this bit becomes set. When TCNT2 has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that TCNT2 is ready to be updated with a new value.

- **Bit 1 – OCR2UB: Output Compare Register2 Update Busy**

When Timer/Counter2 operates asynchronously and OCR2 is written, this bit becomes set. When OCR2 has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that OCR2 is ready to be updated with a new value.

- **Bit 0 – TCR2UB: Timer/Counter Control Register2 Update Busy**

When Timer/Counter2 operates asynchronously and TCCR2 is written, this bit becomes set. When TCCR2 has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that TCCR2 is ready to be updated with a new value.

If a write is performed to any of the three Timer/Counter2 Registers while its update busy flag is set, the updated value might get corrupted and cause an unintentional interrupt to occur.

The mechanisms for reading TCNT2, OCR2, and TCCR2 are different. When reading TCNT2, the actual timer value is read. When reading OCR2 or TCCR2, the value in the temporary storage register is read.

Asynchronous Operation of Timer/Counter2

When Timer/Counter2 operates asynchronously, some considerations must be taken.

- Warning: When switching between asynchronous and synchronous clocking of Timer/Counter2, the Timer Registers TCNT2, OCR2, and TCCR2 might be corrupted. A safe procedure for switching clock source is:
  1. Disable the Timer/Counter2 interrupts by clearing OCIE2 and TOIE2.
  2. Select clock source by setting AS2 as appropriate.
  3. Write new values to TCNT2, OCR2, and TCCR2.
  4. To switch to asynchronous operation: Wait for TCN2UB, OCR2UB, and TCR2UB.
  5. Clear the Timer/Counter2 Interrupt Flags.

6. Enable interrupts, if needed.

- The Oscillator is optimized for use with a 32.768 kHz watch crystal. Applying an external clock to the TOSC1 pin may result in incorrect Timer/Counter2 operation. The CPU main clock frequency must be more than four times the Oscillator frequency.
- When writing to one of the registers TCNT2, OCR2, or TCCR2, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the temporary register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means for example that writing to TCNT2 does not disturb an OCR2 write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT2, OCR2, or TCCR2, the user must wait until the written register has been updated if Timer/Counter2 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare2 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR2 or TCNT2. If the write cycle is not finished, and the MCU enters sleep mode before the OCR2UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter2 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or Extended Standby mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR2, TCNT2, or OCR2.
  2. Wait until the corresponding Update Busy Flag in ASSR returns to zero.
  3. Enter Power-save or Extended Standby mode.
- When the asynchronous operation is selected, the 32.768 kHz Oscillator for Timer/Counter2 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter2 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter2 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.
- Description of wake up from Power-save or Extended Standby mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT2 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT2 is clocked on the asynchronous TOSC clock, reading TCNT2 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up

from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT2 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT2 is thus as follows:

1. Write any value to either of the registers OCR2 or TCCR2.
2. Wait for the corresponding Update Busy Flag to be cleared.
3. Read TCNT2.

- During asynchronous operation, the synchronization of the Interrupt Flags for the asynchronous timer takes three processor cycles plus one timer cycle. The timer is therefore advanced by at least one before the processor can read the timer value causing the setting of the Interrupt Flag. The output compare pin is changed on the timer clock and is not synchronized to the processor clock.

Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – OCIE2: Timer/Counter2 Output Compare Match Interrupt Enable**

When the OCIE2 bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Compare Match interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter2 occurs, i.e., when the OCF2 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

- **Bit 6 – TOIE2: Timer/Counter2 Overflow Interrupt Enable**

When the TOIE2 bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter2 occurs, i.e., when the TOV2 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – OCF2: Output Compare Flag 2**

The OCF2 bit is set (one) when a compare match occurs between the Timer/Counter2 and the data in OCR2 – Output Compare Register2. OCF2 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF2 is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE2 (Timer/Counter2 Compare match Interrupt Enable), and OCF2 are set (one), the Timer/Counter2 Compare match Interrupt is executed.

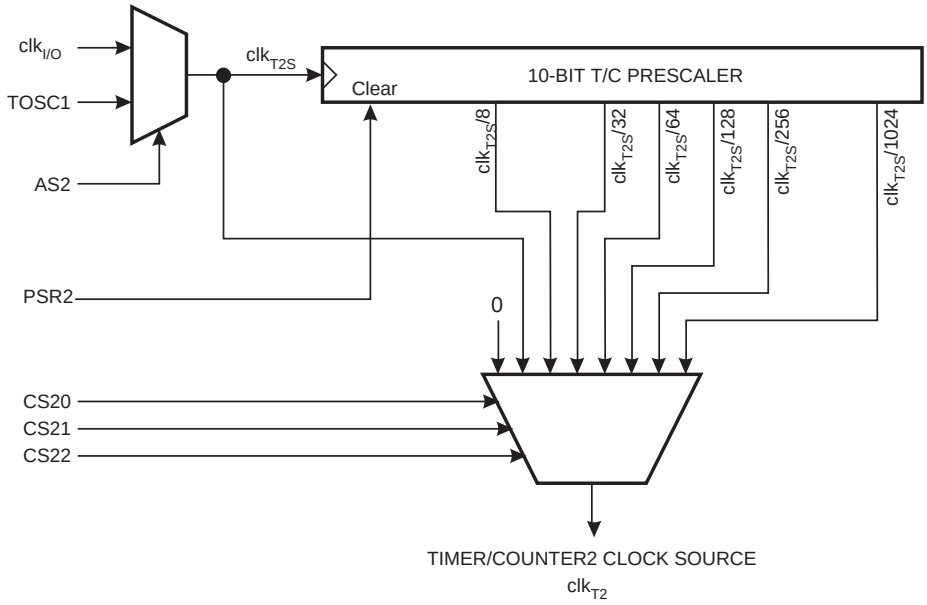
- **Bit 6 – TOV2: Timer/Counter2 Overflow Flag**

The TOV2 bit is set (one) when an overflow occurs in Timer/Counter2. TOV2 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV2 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE2

(Timer/Counter2 Overflow Interrupt Enable), and TOV2 are set (one), the Timer/Counter2 Overflow interrupt is executed. In PWM mode, this bit is set when Timer/Counter2 changes counting direction at $00.

Timer/Counter Prescaler

Figure 64. Prescaler for Timer/Counter2



The clock source for Timer/Counter2 is named clk_{T2S}. clk_{T2S} is by default connected to the main system I/O clock clk_{IO}. By setting the AS2 bit in ASSR, Timer/Counter2 is asynchronously clocked from the TOSC1 pin. This enables use of Timer/Counter2 as a Real Time Counter (RTC). When AS2 is set, pins TOSC1 and TOSC2 are disconnected from Port C. A crystal can then be connected between the TOSC1 and TOSC2 pins to serve as an independent clock source for Timer/Counter2. The Oscillator is optimized for use with a 32.768 kHz crystal. Applying an external clock source to TOSC1 is not recommended.

For Timer/Counter2, the possible prescaled selections are: $clk_{T2S}/8$, $clk_{T2S}/32$, $clk_{T2S}/64$, $clk_{T2S}/128$, $clk_{T2S}/256$, and $clk_{T2S}/1024$. Additionally, clk_{T2S} as well as 0 (stop) may be selected. Setting the PSR2 bit in SFIOR resets the prescaler. This allows the user to operate with a predictable prescaler.

Special Function IO Register – SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1 – PSR2: Prescaler Reset Timer/Counter2**

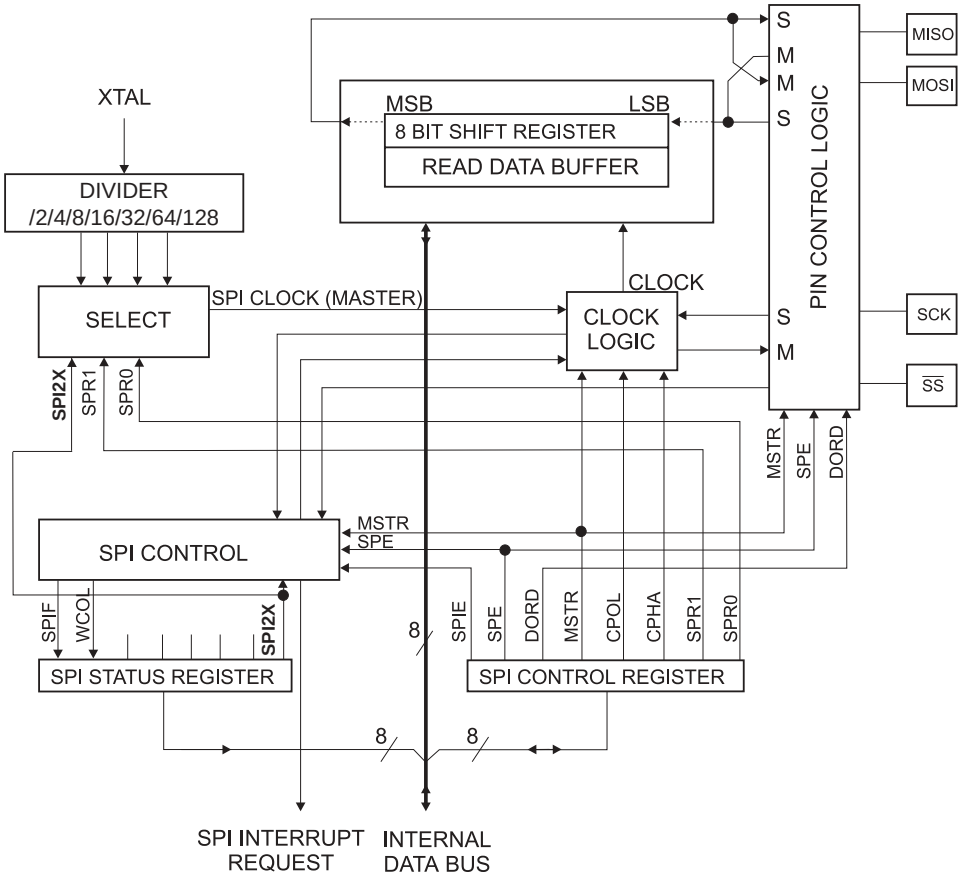
When this bit is written to one, the Timer/Counter2 prescaler will be reset. The bit will be cleared by hardware after the operation is performed. Writing a zero to this bit will have no effect. This bit will always be read as zero if Timer/Counter2 is clocked by the internal CPU clock. If this bit is written when Timer/Counter2 is operating in asynchronous mode, the bit will remain one until the prescaler has been reset.

Serial Peripheral Interface – SPI

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the ATmega32 and peripheral devices or between several AVR devices. The ATmega32 SPI includes the following features:

- **Full-duplex, Three-wire Synchronous Data Transfer**
- **Master or Slave Operation**
- **LSB First or MSB First Data Transfer**
- **Seven Programmable Bit Rates**
- **End of Transmission Interrupt Flag**
- **Write Collision Flag Protection**
- **Wake-up from Idle Mode**
- **Double Speed (CK/2) Master SPI Mode**

Figure 65. SPI Block Diagram[1]



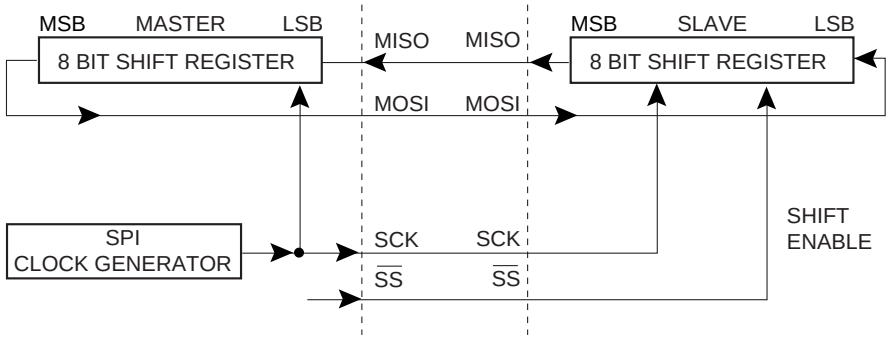
Note: 1. Refer to Figure 1 on page 2, and Table 25 on page 55 for SPI pin placement.

The interconnection between Master and Slave CPUs with SPI is shown in Figure 66. The system consists of two Shift Registers, and a Master clock generator. The SPI Master initiates the communication cycle when pulling low the Slave Select $\overline{SS}$ pin of the desired Slave. Master and Slave prepare the data to be sent in their respective Shift Registers, and the Master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from Master to Slave on the Master Out – Slave In, MOSI, line, and from Slave to Master on the Master In – Slave Out, MISO, line. After each data packet, the Master will synchronize the Slave by pulling high the Slave Select, $\overline{SS}$, line.

When configured as a Master, the SPI interface has no automatic control of the $\overline{SS}$ line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI Data Register starts the SPI clock generator, and the hardware shifts the eight bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of Transmission Flag (SPIF). If the SPI Interrupt Enable bit (SPIE) in the SPCR Register is set, an interrupt is requested. The Master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the Slave Select, $\overline{SS}$ line. The last incoming byte will be kept in the Buffer Register for later use.

When configured as a Slave, the SPI interface will remain sleeping with MISO tri-stated as long as the $\overline{SS}$ pin is driven high. In this state, software may update the contents of the SPI Data Register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the $\overline{SS}$ pin is driven low. As one byte has been completely shifted, the end of Transmission Flag, SPIF is set. If the SPI Interrupt Enable bit, SPIE, in the SPCR Register is set, an interrupt is requested. The Slave may continue to place new data to be sent into SPDR before reading the incoming data. The last incoming byte will be kept in the Buffer Register for later use.

Figure 66. SPI Master-slave Interconnection



The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI Data Register before the entire shift cycle is completed. When receiving data, however, a received character must be read from the SPI Data Register before the next character has been completely shifted in. Otherwise, the first byte is lost.

In SPI Slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the frequency of the SPI clock should never exceed $f_{osc}/4$.

When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and $\overline{SS}$ pins is overridden according to Table 55. For more details on automatic port overrides, refer to “Alternate Port Functions” on page 52.

Table 55. SPI Pin Overrides

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

Note: See “Alternate Functions of Port B” on page 55 for a detailed description of how to define the direction of the user defined SPI pins.

The following code examples show how to initialize the SPI as a master and how to perform a simple transmission. DDR_SPI in the examples must be replaced by the actual Data Direction Register controlling the SPI pins. DD_MOSI, DD_MISO and DD_SCK must be replaced by the actual data direction bits for these pins. For example if MOSI is placed on pin PB5, replace DD_MOSI with DDB5 and DDR_SPI with DDRB.

Assembly Code Example[1]

```
SPI_MasterInit:
  ; Set MOSI and SCK output, all others input
  ldi   r17,(1<<DD_MOSI)|(1<<DD_SCK)
  out   DDR_SPI,r17
  ; Enable SPI, Master, set clock rate fck/16
  ldi   r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
  out   SPCR,r17
  ret

SPI_MasterTransmit:
  ; Start transmission of data (r16)
  out   SPDR,r16
Wait_Transmit:
  ; Wait for transmission complete
  sbis  SPSR,SPIF
  rjmp  Wait_Transmit
  ret
```

C Code Example[1]

```
void SPI_MasterInit(void)
{
  /* Set MOSI and SCK output, all others input */
  DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
  /* Enable SPI, Master, set clock rate fck/16 */
  SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
  /* Start transmission */
  SPDR = cData;
  /* Wait for transmission complete */
  while(!(SPSR & (1<<SPIF)))
    ;
}
```

Note: 1. The example code assumes that the part specific header file is included.

The following code examples show how to initialize the SPI as a Slave and how to perform a simple reception.

Assembly Code Example[1]

```
SPI_SlaveInit:
  ; Set MISO output, all others input
  ldi   r17,(1<<DD_MISO)
  out   DDR_SPI,r17
  ; Enable SPI
  ldi   r17,(1<<SPE)
  out   SPCR,r17
  ret

SPI_SlaveReceive:
  ; Wait for reception complete
  sbis  SPSR,SPIF
  rjmp  SPI_SlaveReceive
  ; Read received data and return
  in    r16,SPDR
  ret
```

C Code Example[1]

```
void SPI_SlaveInit(void)
{
  /* Set MISO output, all others input */
  DDR_SPI = (1<<DD_MISO);
  /* Enable SPI */
  SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
  /* Wait for reception complete */
  while(!(SPSR & (1<<SPIF)))
    ;
  /* Return data register */
  return SPDR;
}
```

Note: 1. The example code assumes that the part specific header file is included.

$\overline{SS}$ Pin Functionality

Slave Mode

When the SPI is configured as a Slave, the Slave Select ($\overline{SS}$) pin is always input. When $\overline{SS}$ is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs. When $\overline{SS}$ is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI logic will be reset once the $\overline{SS}$ pin is driven high.

The $\overline{SS}$ pin is useful for packet/byte synchronization to keep the slave bit counter synchronous with the master clock generator. When the $\overline{SS}$ pin is driven high, the SPI Slave will immediately reset the send and receive logic, and drop any partially received data in the Shift Register.

Master Mode

When the SPI is configured as a Master (MSTR in SPCR is set), the user can determine the direction of the $\overline{SS}$ pin.

If $\overline{SS}$ is configured as an output, the pin is a general output pin which does not affect the SPI system. Typically, the pin will be driving the $\overline{SS}$ pin of the SPI Slave.

If $\overline{SS}$ is configured as an input, it must be held high to ensure Master SPI operation. If the $\overline{SS}$ pin is driven low by peripheral circuitry when the SPI is configured as a Master with the $\overline{SS}$ pin defined as an input, the SPI system interprets this as another master selecting the SPI as a slave and starting to send data to it. To avoid bus contention, the SPI system takes the following actions:

1. The MSTR bit in SPCR is cleared and the SPI system becomes a slave. As a result of the SPI becoming a slave, the MOSI and SCK pins become inputs.
2. The SPIF Flag in SPSR is set, and if the SPI interrupt is enabled, and the I-bit in SREG is set, the interrupt routine will be executed.

Thus, when interrupt-driven SPI transmission is used in master mode, and there exists a possibility that $\overline{SS}$ is driven low, the interrupt should always check that the MSTR bit is still set. If the MSTR bit has been cleared by a slave select, it must be set by the user to re-enable SPI master mode.

SPI Control Register – SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	**SPE**	**DORD**	**MSTR**	**CPOL**	**CPHA**	**SPR1**	**SPR0**	**SPCR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIE: SPI Interrupt Enable**

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the if the global interrupt enable bit in SREG is set.

- **Bit 6 – SPE: SPI Enable**

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: Master/Slave Select**

This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If $\overline{SS}$ is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.

- **Bit 3 – CPOL: Clock Polarity**

When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. Refer to Figure 67 and Figure 68 for an example. The CPOL functionality is summarized below:

Table 56. CPOL Functionality

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

- **Bit 2 – CPHA: Clock Phase**

The settings of the Clock Phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to Figure 67 and Figure 68 for an example. The CPHA functionality is summarized below:

Table 57. CPHA Functionality

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

- **Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0**

These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on the Slave. The relationship between SCK and the Oscillator Clock frequency f_{osc} is shown in the following table:

Table 58. Relationship Between SCK and the Oscillator Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

SPI Status Register – SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIF: SPI Interrupt Flag**

When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If $\overline{SS}$ is an input and is driven low when the SPI is in Master mode, this will also set the SPIF Flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the SPI Data Register (SPDR).

- **Bit 6 – WCOL: Write COLlision Flag**

The WCOL bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

- **Bit 5..1 – Res: Reserved Bits**

These bits are reserved bits in the ATmega32 and will always read as zero.

- **Bit 0 – SPI2X: Double SPI Speed Bit**

When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode (see Table 58). This means that the minimum SCK period will be two CPU clock periods. When the SPI is configured as Slave, the SPI is only guaranteed to work at $f_{osc}/4$ or lower.

The SPI interface on the ATmega32 is also used for program memory and EEPROM downloading or uploading. See page 268 for SPI Serial Programming and Verification.

SPI Data Register – SPDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

The SPI Data Register is a read/write register used for data transfer between the Register File and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 67 and Figure 68. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. This is clearly seen by summarizing Table 56 and Table 57, as done below:

Table 59. CPOL and CPHA Functionality

	Leading Edge	Trailing Edge	SPI Mode
CPOL = 0, CPHA = 0	Sample (Rising)	Setup (Falling)	0
CPOL = 0, CPHA = 1	Setup (Rising)	Sample (Falling)	1
CPOL = 1, CPHA = 0	Sample (Falling)	Setup (Rising)	2
CPOL = 1, CPHA = 1	Setup (Falling)	Sample (Rising)	3

Figure 67. SPI Transfer Format with CPHA = 0

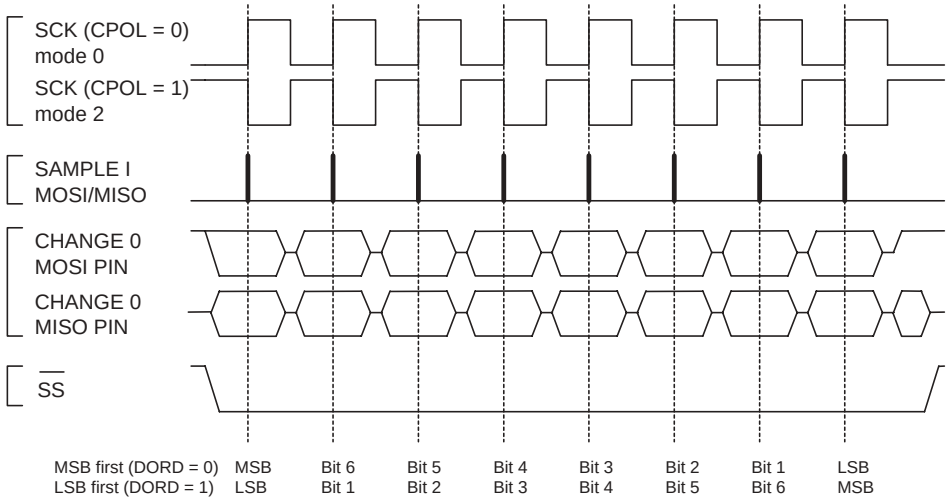
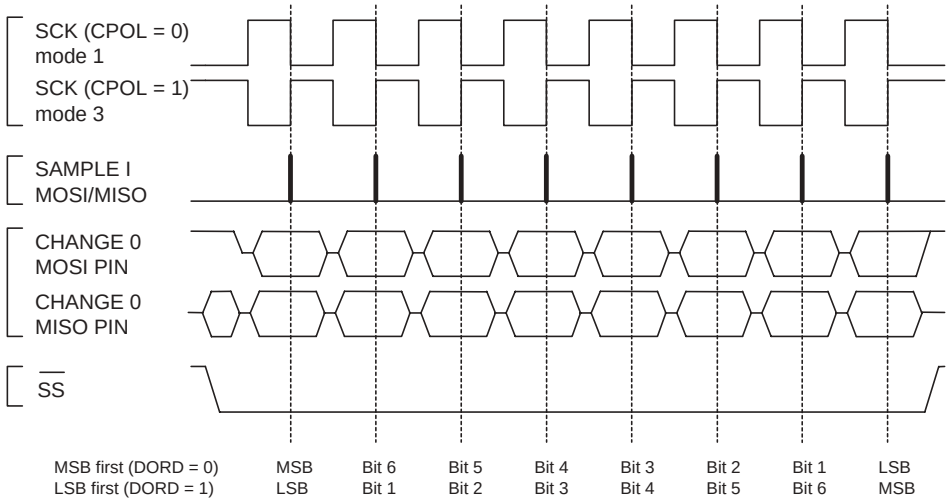


Figure 68. SPI Transfer Format with CPHA = 1



USART

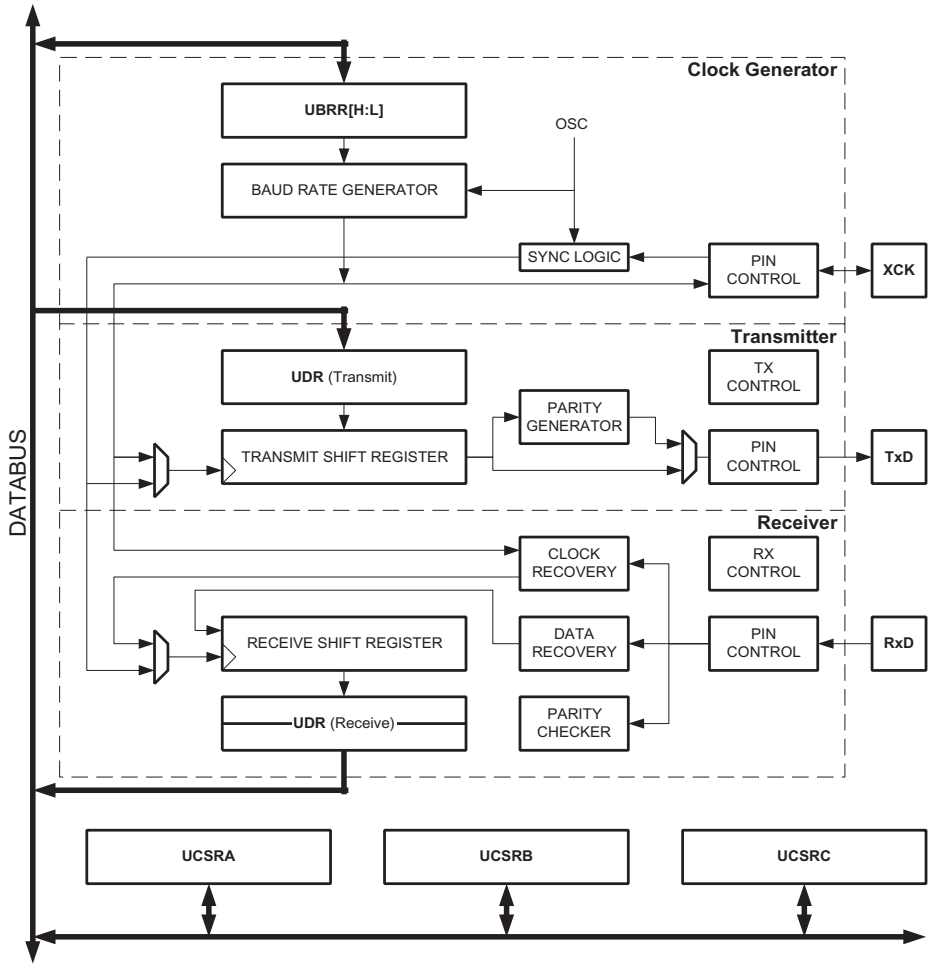
The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. The main features are:

- **Full Duplex Operation (Independent Serial Receive and Transmit Registers)**
- **Asynchronous or Synchronous Operation**
- **Master or Slave Clocked Synchronous Operation**
- **High Resolution Baud Rate Generator**
- **Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits**
- **Odd or Even Parity Generation and Parity Check Supported by Hardware**
- **Data OverRun Detection**
- **Framing Error Detection**
- **Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter**
- **Three Separate Interrupts on TX Complete, TX Data Register Empty, and RX Complete**
- **Multi-processor Communication Mode**
- **Double Speed Asynchronous Communication Mode**

Overview

A simplified block diagram of the USART transmitter is shown in Figure 69. CPU accessible I/O Registers and I/O pins are shown in bold.

Figure 69. USART Block Diagram[1]



Note: 1. Refer to Figure 1 on page 2, Table 33 on page 62, and Table 27 on page 57 for USART pin placement.

The dashed boxes in the block diagram separate the three main parts of the USART (listed from the top): Clock Generator, Transmitter and Receiver. Control Registers are shared by all units. The clock generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCK (Transfer Clock) pin is only used by Synchronous Transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, parity generator and control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the receiver includes a parity checker, control logic, a Shift Register and a two level receive buffer (UDR). The receiver supports the same frame formats as the transmitter, and can detect frame error, data overrun and parity errors.

AVR USART vs. AVR UART – Compatibility

The USART is fully compatible with the AVR UART regarding:

- Bit locations inside all USART Registers
- Baud Rate Generation
- Transmitter Operation
- Transmit Buffer Functionality
- Receiver Operation

However, the receive buffering has two improvements that will affect the compatibility in some special cases:

- A second Buffer Register has been added. The two Buffer Registers operate as a circular FIFO buffer. Therefore the UDR must only be read once for each incoming data! More important is the fact that the Error Flags (FE and DOR) and the 9th data bit (RXB8) are buffered with the data in the receive buffer. Therefore the status bits must always be read before the UDR Register is read. Otherwise the error status will be lost since the buffer state is lost.
- The receiver Shift Register can now act as a third buffer level. This is done by allowing the received data to remain in the serial Shift Register (see Figure 69) if the Buffer Registers are full, until a new start bit is detected. The USART is therefore more resistant to Data OverRun (DOR) error conditions.

The following control bits have changed name, but have same functionality and register location:

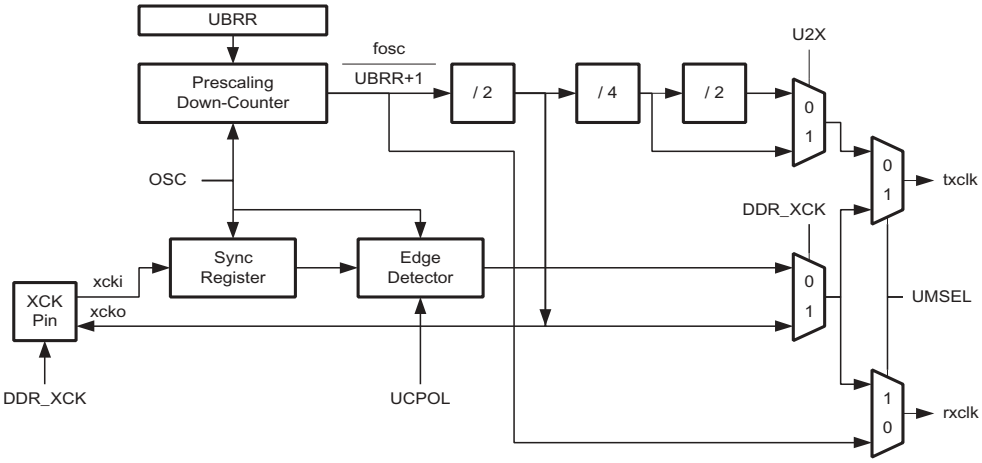
- CHR9 is changed to UCSZ2
- OR is changed to DOR

Clock Generation

The clock generation logic generates the base clock for the Transmitter and Receiver. The USART supports four modes of clock operation: Normal Asynchronous, Double Speed Asynchronous, Master Synchronous and Slave Synchronous mode. The UMSEL bit in USART Control and Status Register C (UCSRC) selects between asynchronous and synchronous operation. Double Speed (Asynchronous mode only) is controlled by the U2X found in the UCSRA Register. When using Synchronous mode (UMSEL = 1), the Data Direction Register for the XCK pin (DDR_XCK) controls whether the clock source is internal (Master mode) or external (Slave mode). The XCK pin is only active when using Synchronous mode.

Figure 70 shows a block diagram of the clock generation logic.

Figure 70. Clock Generation Logic, Block Diagram



Signal description:

txclk Transmitter clock (Internal Signal).

rxclk Receiver base clock (Internal Signal).

xcki Input from XCK pin (Internal Signal). Used for synchronous slave operation.

xcko Clock output to XCK pin (Internal Signal). Used for synchronous master operation.

fosc XTAL pin frequency (System Clock).

Internal Clock Generation – The Baud Rate Generator

Internal clock generation is used for the asynchronous and the synchronous master modes of operation. The description in this section refers to Figure 70.

The USART Baud Rate Register (UBRR) and the down-counter connected to it function as a programmable prescaler or baud rate generator. The down-counter, running at system clock (fosc), is loaded with the UBRR value each time the counter has counted down to zero or when the UBRRL Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output (= fosc/(UBRR+1)). The Transmitter divides the baud rate generator clock output by 2, 8 or 16 depending on mode. The baud rate generator output is used directly by the receiver's clock and data recovery units. However, the recovery units use a state machine that uses 2, 8 or 16 states depending on mode set by the state of the UMSEL, U2X and DDR_XCK bits.

Table 60 contains equations for calculating the baud rate (in bits per second) and for calculating the UBRR value for each mode of operation using an internally generated clock source.

Table 60. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(\boldsymbol{UBRR}+1)}$	$\boldsymbol{UBRR} = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(\boldsymbol{UBRR}+1)}$	$\boldsymbol{UBRR} = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(\boldsymbol{UBRR}+1)}$	$\boldsymbol{UBRR} = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps).

BAUD Baud rate (in bits per second, bps)

f_{OSC} System Oscillator clock frequency

UBRR Contents of the UBRRH and UBRRL Registers, (0 - 4095)

Some examples of UBRR values for some system clock frequencies are found in Table 68 (see page 163).

Double Speed Operation (U2X)

The transfer rate can be doubled by setting the U2X bit in UCSRA. Setting this bit only has effect for the asynchronous operation. Set this bit to zero when using synchronous operation.

Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. Note however that the receiver will in this case only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used. For the Transmitter, there are no downsides.

External Clock

External clocking is used by the synchronous slave modes of operation. The description in this section refers to Figure 70 for details.

External clock input from the XCK pin is sampled by a synchronization register to minimize the chance of meta-stability. The output from the synchronization register must then pass through an edge detector before it can be used by the Transmitter and receiver. This process introduces a two CPU clock period delay and therefore the maximum external XCK clock frequency is limited by the following equation:

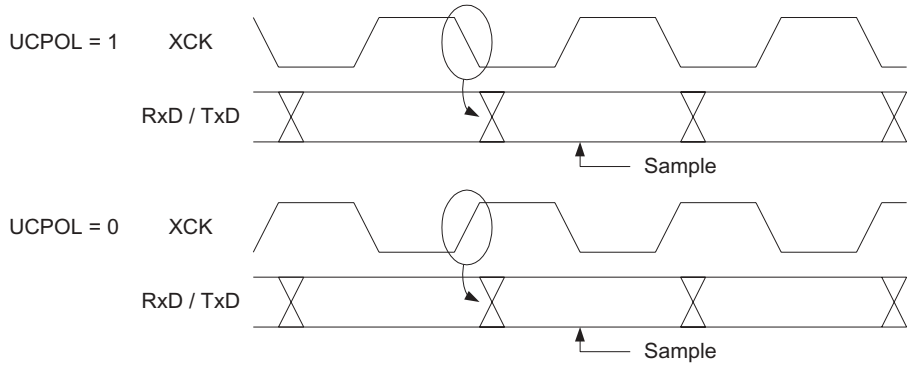
$$f_{XCK} < \frac{f_{OSC}}{4}$$

Note that f_{osc} depends on the stability of the system clock source. It is therefore recommended to add some margin to avoid possible loss of data due to frequency variations.

Synchronous Clock Operation

When Synchronous mode is used (UMSEL = 1), the XCK pin will be used as either clock input (Slave) or clock output (Master). The dependency between the clock edges and data sampling or data change is the same. The basic principle is that data input (on RxD) is sampled at the opposite XCK clock edge of the edge the data output (TxD) is changed.

Figure 71. Synchronous Mode XCK Timing.



The UCPOL bit UCRSC selects which XCK clock edge is used for data sampling and which is used for data change. As Figure 71 shows, when UCPOL is zero the data will be changed at rising XCK edge and sampled at falling XCK edge. If UCPOL is set, the data will be changed at falling XCK edge and sampled at rising XCK edge.

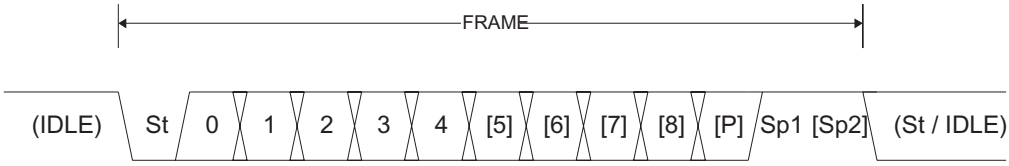
Frame Formats

A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The USART accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. Figure 72 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

Figure 72. Frame Formats



St Start bit, always low.

(n) Data bits (0 to 8).

P Parity bit. Can be odd or even.

Sp Stop bit, always high.

IDLE No transfers on the communication line (RxD or TxD). An IDLE line must be high.

The frame format used by the USART is set by the UCSZ2:0, UPM1:0, and USBS bits in UCSRB and UCSRC. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

The USART Character SiZe (UCSZ2:0) bits select the number of data bits in the frame. The USART Parity mode (UPM1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the USART Stop Bit Select (USBS) bit. The receiver ignores the second stop bit. An FE (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

Parity Bit Calculation

The parity bit is calculated by doing an exclusive-or of all the data bits. If odd parity is used, the result of the exclusive or is inverted. The relation between the parity bit and data bits is as follows::

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

P_{even} Parity bit using even parity

P_{odd} Parity bit using odd parity

d_n Data bit n of the character

If used, the parity bit is located between the last data bit and first stop bit of a serial frame.

USART Initialization

The USART has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting frame format and enabling the Transmitter or the Receiver depending on the usage. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and interrupts globally disabled) when doing the initialization.

Before doing a re-initialization with changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The TXC Flag can be used to check that the Transmitter has completed all transfers, and the RXC Flag can be used to check that there are no unread data in the receive buffer. Note that the TXC Flag must be cleared before each transmission (before UDR is written) if it is used for this purpose.

The following simple USART initialization code examples show one assembly and one C function that are equal in functionality. The examples assume asynchronous operation using polling (no interrupts enabled) and a fixed frame format. The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 registers. When the function writes to the UCSRC Register, the URSEL bit (MSB) must be set due to the sharing of I/O location by UBRRH and UCSRC.

Assembly Code Example[1]

```
USART_Init:
  ; Set baud rate
  out   UBRRH, r17
  out   UBRRL, r16
  ; Enable receiver and transmitter
  ldi   r16, (1<<RXEN)|(1<<TXEN)
  out   UCSRB,r16
  ; Set frame format: 8data, 2stop bit
  ldi   r16, (1<<URSEL)|(1<<USBS)|(3<<UCSZ0)
  out   UCSRC,r16
  ret
```

C Code Example[1]

```
void USART_Init( unsigned int baud )
{
  /* Set baud rate */
  UBRRH = (unsigned char)(baud>>8);
  UBRRL = (unsigned char)baud;
  /* Enable receiver and transmitter */
  UCSRB = (1<<RXEN)|(1<<TXEN);
  /* Set frame format: 8data, 2stop bit */
  UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
}
```

Note: 1. The example code assumes that the part specific header file is included.

More advanced initialization routines can be made that include frame format as parameters, disable interrupts and so on. However, many applications use a fixed setting of the Baud and Control Registers, and for these types of applications the initialization code can be placed directly in the main routine, or be combined with initialization code for other I/O modules.

Data Transmission – The USART Transmitter

The USART Transmitter is enabled by setting the *Transmit Enable* (TXEN) bit in the UCSRB Register. When the Transmitter is enabled, the normal port operation of the TxD pin is overridden by the USART and given the function as the transmitter's serial output. The baud rate, mode of operation and frame format must be set up once before doing any transmissions. If synchronous operation is used, the clock on the XCK pin will be overridden and used as transmission clock.

Sending Frames with 5 to 8 Data Bit

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The CPU can load the transmit buffer by writing to the UDR I/O location. The buffered data in the transmit buffer will be moved to the Shift Register when the Shift Register is ready to send a new frame. The Shift Register is loaded with new data if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the Shift Register is loaded with new data, it will transfer one complete frame at the rate given by the Baud Register, U2X bit or by XCK depending on mode of operation.

The following code examples show a simple USART transmit function based on polling of the *Data Register Empty* (UDRE) Flag. When using frames with less than eight bits, the most significant bits written to the UDR are ignored. The USART has to be initialized before the function can be used. For the assembly code, the data to be sent is assumed to be stored in Register R16

Assembly Code Example[1]

```
USART_Transmit:
  ; Wait for empty transmit buffer
  sbis  UCSRA,UDRE
  rjmp  USART_Transmit
  ; Put data (r16) into buffer, sends the data
  out   UDR,r16
  ret
```

C Code Example[1]

```
void USART_Transmit( unsigned char data )
{
  /* Wait for empty transmit buffer */
  while ( !( UCSRA & (1<<UDRE)) )
        ;
  /* Put data into buffer, sends the data */
  UDR = data;
}
```

Note: 1. The example code assumes that the part specific header file is included.

The function simply waits for the transmit buffer to be empty by checking the UDRE Flag, before loading it with new data to be transmitted. If the Data Register Empty Interrupt is utilized, the interrupt routine writes the data into the buffer.

Sending Frames with 9 Data Bit

If 9-bit characters are used (UCSZ = 7), the ninth bit must be written to the TXB8 bit in UCSRB before the low byte of the character is written to UDR. The following code examples show a transmit function that handles 9-bit characters. For the assembly code, the data to be sent is assumed to be stored in Registers R17:R16.

Assembly Code Example[1]

```
USART_Transmit:
  ; Wait for empty transmit buffer
  sbis  UCSRA,UDRE
  rjmp  USART_Transmit
  ; Copy 9th bit from r17 to TXB8
  cbi   UCSRB,TXB8
  sbrc  r17,0
  sbi   UCSRB,TXB8
  ; Put LSB data (r16) into buffer, sends the data
  out   UDR,r16
  ret
```

C Code Example[1]

```
void USART_Transmit( unsigned int data )
{
  /* Wait for empty transmit buffer */
  while ( !( UCSRA & (1<<UDRE))) )
        ;
  /* Copy 9th bit to TXB8 */
  UCSRB &= ~(1<<TXB8);
  if ( data & 0x0100 )
    UCSRB |= (1<<TXB8);
  /* Put data into buffer, sends the data */
  UDR = data;
}
```

Note: 1. These transmit functions are written to be general functions. They can be optimized if the contents of the UCSRB is static. (i.e., only the TXB8 bit of the UCSRB Register is used after initialization).

The ninth bit can be used for indicating an address frame when using multi processor communication mode or for other protocol handling as for example synchronization.

Transmitter Flags and Interrupts

The USART transmitter has two flags that indicate its state: USART Data Register Empty (UDRE) and Transmit Complete (TXC). Both flags can be used for generating interrupts.

The Data Register Empty (UDRE) Flag indicates whether the transmit buffer is ready to receive new data. This bit is set when the transmit buffer is empty, and cleared when the transmit buffer contains data to be transmitted that has not yet been moved into the Shift Register. For compatibility with future devices, always write this bit to zero when writing the UCSRA Register.

When the Data Register empty Interrupt Enable (UDRIE) bit in UCSRB is written to one, the USART Data Register Empty Interrupt will be executed as long as UDRE is set (provided that global interrupts are enabled). UDRE is cleared by writing UDR. When interrupt-driven data transmission is used, the Data Register Empty Interrupt routine must either write new data to UDR in order to clear UDRE or disable the Data Register

empty Interrupt, otherwise a new interrupt will occur once the interrupt routine terminates.

The Transmit Complete (TXC) Flag bit is set one when the entire frame in the transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer. The TXC Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC Flag is useful in half-duplex communication interfaces (like the RS485 standard), where a transmitting application must enter receive mode and free the communication bus immediately after completing the transmission.

When the Transmit Compete Interrupt Enable (TXCIE) bit in UCSRB is set, the USART Transmit Complete Interrupt will be executed when the TXC Flag becomes set (provided that global interrupts are enabled). When the transmit complete interrupt is used, the interrupt handling routine does not have to clear the TXC Flag, this is done automatically when the interrupt is executed.

Parity Generator

The parity generator calculates the parity bit for the serial frame data. When parity bit is enabled (UPM1 = 1), the transmitter control logic inserts the parity bit between the last data bit and the first stop bit of the frame that is sent.

Disabling the Transmitter

The disabling of the transmitter (setting the TXEN to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the transmit Shift Register and transmit Buffer Register do not contain data to be transmitted. When disabled, the transmitter will no longer override the TxD pin.

Data Reception – The USART Receiver

The USART Receiver is enabled by writing the Receive Enable (RXEN) bit in the UCSRB Register to one. When the receiver is enabled, the normal pin operation of the RxD pin is overridden by the USART and given the function as the receiver's serial input. The baud rate, mode of operation and frame format must be set up once before any serial reception can be done. If synchronous operation is used, the clock on the XCK pin will be used as transfer clock.

Receiving Frames with 5 to 8 Data Bits

The receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCK clock, and shifted into the receive Shift Register until the first stop bit of a frame is received. A second stop bit will be ignored by the receiver. When the first stop bit is received, i.e., a complete serial frame is present in the receive Shift Register, the contents of the Shift Register will be moved into the receive buffer. The receive buffer can then be read by reading the UDR I/O location.

The following code example shows a simple USART receive function based on polling of the Receive Complete (RXC) Flag. When using frames with less than eight bits the most significant bits of the data read from the UDR will be masked to zero. The USART has to be initialized before the function can be used.

Assembly Code Example(1)

```
USART_Receive:
  ; Wait for data to be received
  sbis  UCSRA, RXC
  rjmp  USART_Receive
  ; Get and return received data from buffer
  in    r16, UDR
  ret
```

C Code Example(1)

```
unsigned char USART_Receive( void )
{
  /* Wait for data to be received */
  while ( !(UCSRA & (1<<RXC)) )
        ;
  /* Get and return received data from buffer */
  return UDR;
}
```

Note: 1. The example code assumes that the part specific header file is included.

The function simply waits for data to be present in the receive buffer by checking the RXC Flag, before reading the buffer and returning the value.

Receiving Frames with 9 Databits

If 9 bit characters are used (UCSZ=7) the ninth bit must be read from the RXB8 bit in UCSRB **before** reading the low bits from the UDR. This rule applies to the FE, DOR and PE Status Flags as well. Read status from UCSRA, then data from UDR. Reading the UDR I/O location will change the state of the receive buffer FIFO and consequently the TXB8, FE, DOR and PE bits, which all are stored in the FIFO, will change.

The following code example shows a simple USART receive function that handles both 9-bit characters and the status bits.

Assembly Code Example[1]

```
USART_Receive:
  ; Wait for data to be received
  sbis  UCSRA, RXC
  rjmp  USART_Receive
  ; Get status and 9th bit, then data from buffer
  in    r18, UCSRA
  in    r17, UCSRB
  in    r16, UDR
  ; If error, return -1
  andi  r18,(1<<FE)|(1<<DOR)|(1<<PE)
  breq  USART_ReceiveNoError
  ldi   r17, HIGH(-1)
  ldi   r16, LOW(-1)
USART_ReceiveNoError:
  ; Filter the 9th bit, then return
  lsr   r17
  andi  r17, 0x01
  ret
```

C Code Example[1]

```
unsigned int USART_Receive( void )
{
  unsigned char status, resh, resl;
  /* Wait for data to be received */
  while ( !(UCSRA & (1<<RXC)) )
        ;
  /* Get status and 9th bit, then data */
  /* from buffer */
  status = UCSRA;
  resh = UCSRB;
  resl = UDR;
  /* If error, return -1 */
  if ( status & (1<<FE)|(1<<DOR)|(1<<PE) )
    return -1;
  /* Filter the 9th bit, then return */
  resh = (resh >> 1) & 0x01;
  return ((resh << 8) | resl);
}
```

Note: 1. The example code assumes that the part specific header file is included.

The receive function example reads all the I/O Registers into the Register File before any computation is done. This gives an optimal receive buffer utilization since the buffer location read will be free to accept new data as early as possible.

Receive Compete Flag and Interrupt

The USART Receiver has one flag that indicates the receiver state.

The Receive Complete (RXC) Flag indicates if there are unread data present in the receive buffer. This flag is one when unread data exist in the receive buffer, and zero when the receive buffer is empty (i.e., does not contain any unread data). If the receiver is disabled (RXEN = 0), the receive buffer will be flushed and consequently the RXC bit will become zero.

When the Receive Complete Interrupt Enable (RXCIE) in UCSRB is set, the USART Receive Complete Interrupt will be executed as long as the RXC Flag is set (provided that global interrupts are enabled). When interrupt-driven data reception is used, the receive complete routine must read the received data from UDR in order to clear the RXC Flag, otherwise a new interrupt will occur once the interrupt routine terminates.

Receiver Error Flags

The USART Receiver has three Error Flags: Frame Error (FE), Data OverRun (DOR) and Parity Error (PE). All can be accessed by reading UCSRA. Common for the Error Flags is that they are located in the receive buffer together with the frame for which they indicate the error status. Due to the buffering of the Error Flags, the UCSRA must be read before the receive buffer (UDR), since reading the UDR I/O location changes the buffer read location. Another equality for the Error Flags is that they can not be altered by software doing a write to the flag location. However, all flags must be set to zero when the UCSRA is written for upward compatibility of future USART implementations. None of the Error Flags can generate interrupts.

The Frame Error (FE) Flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The FE Flag is zero when the stop bit was correctly read (as one), and the FE Flag will be one when the stop bit was incorrect (zero). This flag can be used for detecting out-of-sync conditions, detecting break conditions and protocol handling. The FE Flag is not affected by the setting of the USBS bit in UCSRC since the receiver ignores all, except for the first, stop bits. For compatibility with future devices, always set this bit to zero when writing to UCSRA.

The Data OverRun (DOR) Flag indicates data loss due to a receiver buffer full condition. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the receive Shift Register, and a new start bit is detected. If the DOR Flag is set there was one or more serial frame lost between the frame last read from UDR, and the next frame read from UDR. For compatibility with future devices, always write this bit to zero when writing to UCSRA. The DOR Flag is cleared when the frame received was successfully moved from the Shift Register to the receive buffer.

The Parity Error (PE) Flag indicates that the next frame in the receive buffer had a parity error when received. If parity check is not enabled the PE bit will always be read zero. For compatibility with future devices, always set this bit to zero when writing to UCSRA. For more details see “Parity Bit Calculation” on page 143 and “Parity Checker” on page 150.

Parity Checker

The Parity Checker is active when the high USART Parity mode (UPM1) bit is set. Type of parity check to be performed (odd or even) is selected by the UPM0 bit. When enabled, the parity checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit from the serial frame. The result of the check is stored in the receive buffer together with the received data and stop bits. The Parity Error (PE) Flag can then be read by software to check if the frame had a parity error.

The PE bit is set if the next character that can be read from the receive buffer had a parity error when received and the parity checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read.

Disabling the Receiver

In contrast to the Transmitter, disabling of the Receiver will be immediate. Data from ongoing receptions will therefore be lost. When disabled (i.e., the RXEN is set to zero) the Receiver will no longer override the normal function of the RxD port pin. The receiver buffer FIFO will be flushed when the receiver is disabled. Remaining data in the buffer will be lost

Flushing the Receive Buffer

The receiver buffer FIFO will be flushed when the Receiver is disabled, i.e., the buffer will be emptied of its contents. Unread data will be lost. If the buffer has to be flushed during normal operation, due to for instance an error condition, read the UDR I/O location until the RXC Flag is cleared. The following code example shows how to flush the receive buffer.

Assembly Code Example[1]

```
USART_Flush:
  sbis  UCSRA, RXC
  ret
  in    r16, UDR
  rjmp  USART_Flush
```

C Code Example[1]

```
void USART_Flush( void )
{
  unsigned char dummy;
  while ( UCSRA & (1<<RXC) ) dummy = UDR;
}
```

Note: 1. The example code assumes that the part specific header file is included.

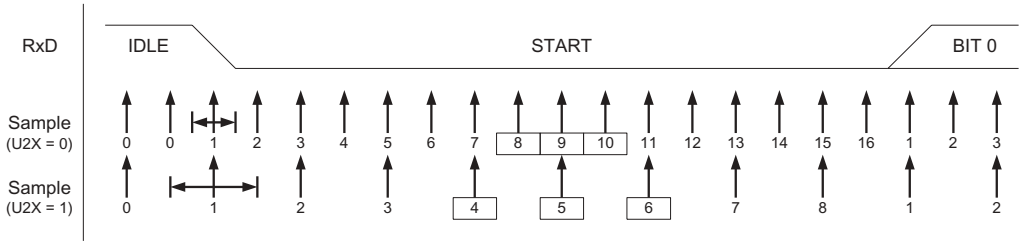
Asynchronous Data Reception

The USART includes a clock recovery and a data recovery unit for handling asynchronous data reception. The clock recovery logic is used for synchronizing the internally generated baud rate clock to the incoming asynchronous serial frames at the RxD pin. The data recovery logic samples and low pass filters each incoming bit, thereby improving the noise immunity of the receiver. The asynchronous reception operational range depends on the accuracy of the internal baud rate clock, the rate of the incoming frames, and the frame size in number of bits.

Asynchronous Clock Recovery

The clock recovery logic synchronizes internal clock to the incoming serial frames. Figure 73 illustrates the sampling process of the start bit of an incoming frame. The sample rate is 16 times the baud rate for Normal mode, and 8 times the baud rate for Double Speed mode. The horizontal arrows illustrate the synchronization variation due to the sampling process. Note the larger time variation when using the double speed mode (U2X = 1) of operation. Samples denoted zero are samples done when the RxD line is idle (i.e., no communication activity).

Figure 73. Start Bit Sampling

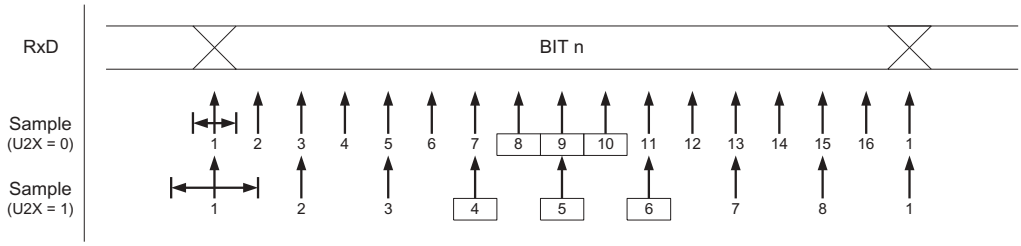


When the clock recovery logic detects a high (idle) to low (start) transition on the RxD line, the start bit detection sequence is initiated. Let sample 1 denote the first zero-sample as shown in the figure. The clock recovery logic then uses samples 8, 9, and 10 for Normal mode, and samples 4, 5, and 6 for Double Speed mode (indicated with sample numbers inside boxes on the figure), to decide if a valid start bit is received. If two or more of these three samples have logical high levels (the majority wins), the start bit is rejected as a noise spike and the receiver starts looking for the next high to low-transition. If however, a valid start bit is detected, the clock recovery logic is synchronized and the data recovery can begin. The synchronization process is repeated for each start bit.

Asynchronous Data Recovery

When the receiver clock is synchronized to the start bit, the data recovery can begin. The data recovery unit uses a state machine that has 16 states for each bit in normal mode and 8 states for each bit in Double Speed mode. Figure 74 shows the sampling of the data bits and the parity bit. Each of the samples is given a number that is equal to the state of the recovery unit.

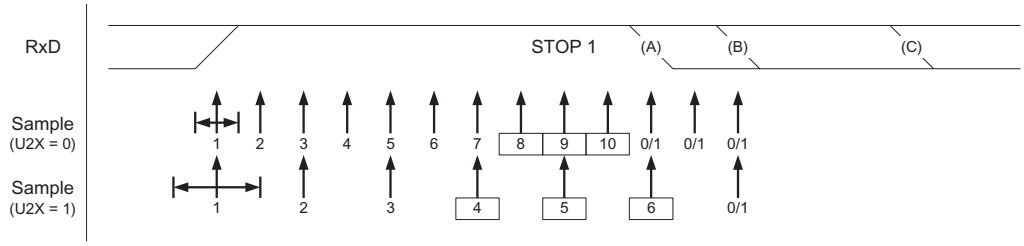
Figure 74. Sampling of Data and Parity Bit



The decision of the logic level of the received bit is taken by doing a majority voting of the logic value to the three samples in the center of the received bit. The center samples are emphasized on the figure by having the sample number inside boxes. The majority voting process is done as follows: If two or all three samples have high levels, the received bit is registered to be a logic 1. If two or all three samples have low levels, the received bit is registered to be a logic 0. This majority voting process acts as a low pass filter for the incoming signal on the RxD pin. The recovery process is then repeated until a complete frame is received. Including the first stop bit. Note that the receiver only uses the first stop bit of a frame.

Figure 75 shows the sampling of the stop bit and the earliest possible beginning of the start bit of the next frame.

Figure 75. Stop Bit Sampling and Next Start Bit Sampling



The same majority voting is done to the stop bit as done for the other bits in the frame. If the stop bit is registered to have a logic 0 value, the Frame Error (FE) Flag will be set.

A new high to low transition indicating the start bit of a new frame can come right after the last of the bits used for majority voting. For Normal Speed mode, the first low level sample can be at point marked (A) in Figure 75. For Double Speed mode the first low level must be delayed to (B). (C) marks a stop bit of full length. The early start bit detection influences the operational range of the receiver.

Asynchronous Operational Range

The operational range of the receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If the Transmitter is sending frames at too fast or too slow bit rates, or the internally generated baud rate of the receiver does not have a similar (see Table 61) base frequency, the receiver will not be able to synchronize the frames to the start bit.

The following equations can be used to calculate the ratio of the incoming data rate and internal receiver baud rate.

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F}$$

$$R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

D Sum of character size and parity size (D = 5 to 10 bit)

S Samples per bit. S = 16 for Normal Speed mode and S = 8 for Double Speed mode.

S_F First sample number used for majority voting. S_F = 8 for Normal Speed and S_F = 4 for Double Speed mode.

S_M Middle sample number used for majority voting. S_M = 9 for Normal Speed and S_M = 5 for Double Speed mode.

R_{slow} is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate. R_{fast} is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

Table 61 and Table 62 list the maximum receiver baud rate error that can be tolerated. Note that Normal Speed mode has higher toleration of baud rate variations.

Table 61. Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (U2X = 0)

D # (Data+Parity Bit)	R_{slow} (%)	R_{fast}(%)	Max Total Error (%)	Recommended Max Receiver Error (%)
5	93.20	106.67	+6.67/-6.8	± 3.0
6	94.12	105.79	+5.79/-5.88	± 2.5
7	94.81	105.11	+5.11/-5.19	± 2.0
8	95.36	104.58	+4.58/-4.54	± 2.0
9	95.81	104.14	+4.14/-4.19	± 1.5
10	96.17	103.78	+3.78/-3.83	± 1.5

Table 62. Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (U2X = 1)

D # (Data+Parity Bit)	R_{slow} (%)	R_{fast} (%)	Max Total Error (%)	Recommended Max Receiver Error (%)
5	94.12	105.66	+5.66/-5.88	± 2.5
6	94.92	104.92	+4.92/-5.08	± 2.0
7	95.52	104.35	+4.35/-4.48	± 1.5
8	96.00	103.90	+3.90/-4.00	± 1.5
9	96.39	103.53	+3.53/-3.61	± 1.5
10	96.70	103.23	+3.23/-3.30	± 1.0

The recommendations of the maximum receiver baud rate error was made under the assumption that the receiver and transmitter equally divides the maximum total error.

There are two possible sources for the receivers baud rate error. The receiver's system clock (XTAL) will always have some minor instability over the supply voltage range and the temperature range. When using a crystal to generate the system clock, this is rarely a problem, but for a resonator the system clock may differ more than 2% depending of the resonators tolerance. The second source for the error is more controllable. The baud rate generator can not always do an exact division of the system frequency to get the baud rate wanted. In this case an UBRR value that gives an acceptable low error can be used if possible.

Multi-processor Communication Mode

Setting the Multi-processor Communication mode (MPCM) bit in UCSRA enables a filtering function of incoming frames received by the USART Receiver. Frames that do not contain address information will be ignored and not put into the receive buffer. This effectively reduces the number of incoming frames that has to be handled by the CPU, in a system with multiple MCUs that communicate via the same serial bus. The Transmitter is unaffected by the MPCM setting, but has to be used differently when it is a part of a system utilizing the Multi-processor Communication mode.

If the receiver is set up to receive frames that contain 5 to 8 data bits, then the first stop bit indicates if the frame contains data or address information. If the receiver is set up for frames with nine data bits, then the ninth bit (RXB8) is used for identifying address and data frames. When the frame type bit (the first stop or the ninth bit) is one, the frame contains an address. When the frame type bit is zero the frame is a data frame.

The Multi-processor Communication mode enables several slave MCUs to receive data from a master MCU. This is done by first decoding an address frame to find out which MCU has been addressed. If a particular Slave MCU has been addressed, it will receive the following data frames as normal, while the other slave MCUs will ignore the received frames until another address frame is received.

Using MPCM

For an MCU to act as a master MCU, it can use a 9-bit character frame format (UCSZ = 7). The ninth bit (TXB8) must be set when an address frame (TXB8 = 1) or cleared when a data frame (TXB = 0) is being transmitted. The slave MCUs must in this case be set to use a 9-bit character frame format.

The following procedure should be used to exchange data in Multi-processor Communication mode:

1. All slave MCUs are in Multi-processor Communication mode (MPCM in UCSRA is set).
2. The Master MCU sends an address frame, and all slaves receive and read this frame. In the Slave MCUs, the RXC Flag in UCSRA will be set as normal.
3. Each Slave MCU reads the UDR Register and determines if it has been selected. If so, it clears the MPCM bit in UCSRA, otherwise it waits for the next address byte and keeps the MPCM setting.
4. The addressed MCU will receive all data frames until a new address frame is received. The other slave MCUs, which still have the MPCM bit set, will ignore the data frames.
5. When the last data frame is received by the addressed MCU, the addressed MCU sets the MPCM bit and waits for a new address frame from Master. The process then repeats from 2.

Using any of the 5- to 8-bit character frame formats is possible, but impractical since the receiver must change between using n and n+1 character frame formats. This makes full-duplex operation difficult since the transmitter and receiver uses the same character size setting. If 5- to 8-bit character frames are used, the transmitter must be set to use two stop bit (USBS = 1) since the first stop bit is used for indicating the frame type.

Do not use Read-Modify-Write instructions (SBI and CBI) to set or clear the MPCM bit. The MPCM bit shares the same I/O location as the TXC Flag and this might accidentally be cleared when using SBI or CBI instructions.

Accessing UBRRH/ UCSRC Registers

The UBRRH Register shares the same I/O location as the UCSRC Register. Therefore some special consideration must be taken when accessing this I/O location.

Write Access

When doing a write access of this I/O location, the high bit of the value written, the USART Register Select (URSEL) bit, controls which one of the two registers that will be written. If URSEL is zero during a write operation, the UBRRH value will be updated. If URSEL is one, the UCSRC setting will be updated.

The following code examples show how to access the two registers.

Assembly Code Example[1]

```
...
; Set UBRRH to 2
ldi r16,0x02
out UBRRH,r16
...
; Set the USBS and the UCSZ1 bit to one, and
; the remaining bits to zero.
ldi r16,(1<<URSEL)|(1<<USBS)|(1<<UCSZ1)
out UCSRC,r16
...
```

C Code Example[1]

```
...
/* Set UBRRH to 2 */
UBRRH = 0x02;
...
/* Set the USBS and the UCSZ1 bit to one, and */
/* the remaining bits to zero. */
UCSRC = (1<<URSEL)|(1<<USBS)|(1<<UCSZ1);
...
```

Note: 1. The example code assumes that the part specific header file is included.

As the code examples illustrate, write accesses of the two registers are relatively unaffected of the sharing of I/O location.

Read Access

Doing a read access to the UBRRH or the UCSRC Register is a more complex operation. However, in most applications, it is rarely necessary to read any of these registers.

The read access is controlled by a timed sequence. Reading the I/O location once returns the UBRRH Register contents. If the register location was read in previous system clock cycle, reading the register in the current clock cycle will return the UCSRC contents. Note that the timed sequence for reading the UCSRC is an atomic operation. Interrupts must therefore be controlled (for example by disabling interrupts globally) during the read operation.

The following code example shows how to read the UCSRC Register contents.

Assembly Code Example(1)

```
USART_ReadUCSRC:
  ; Read UCSRC
  in  r16,UBRRH
  in  r16,UCSRC
  ret
```

C Code Example(1)

```
unsigned char USART_ReadUCSRC( void )
{
  unsigned char ucsrc;
  /* Read UCSRC */
  ucsrc = UBRRH;
  ucsrc = UCSRC;
  return ucsrc;
}
```

Note: 1. The example code assumes that the part specific header file is included.

The assembly code example returns the UCSRC value in r16.

Reading the UBRRH contents is not an atomic operation and therefore it can be read as an ordinary register, as long as the previous instruction did not access the register location.

USART Register Description

USART I/O Data Register – UDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR Register location. Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDRE Flag in the UCSRA Register is set. Data written to UDR when the UDRE Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use read modify write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

USART Control and Status Register A – UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	**TXC**	**UDRE**	**FE**	**DOR**	**PE**	**U2X**	**MPCM**	**UCSRA**
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – RXC: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero. The RXC Flag can be used to generate a Receive Complete interrupt (see description of the RXCIE bit).

- **Bit 6 – TXC: USART Transmit Complete**

This flag bit is set when the entire frame in the transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDR). The TXC Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC Flag can generate a Transmit Complete interrupt (see description of the TXCIE bit).

- **Bit 5 – UDRE: USART Data Register Empty**

The UDRE Flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE Flag can generate a Data Register empty Interrupt (see description of the UDRIE bit).

UDRE is set after a reset to indicate that the transmitter is ready.

- **Bit 4 – FE: Frame Error**

This bit is set if the next character in the receive buffer had a Frame Error when received. i.e., when the first stop bit of the next character in the receive buffer is zero. This bit is valid until the receive buffer (UDR) is read. The FE bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRA.

- **Bit 3 – DOR: Data OverRun**

This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.

- **Bit 2 – PE: Parity Error**

This bit is set if the next character in the receive buffer had a Parity Error when received and the parity checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.

- **Bit 1 – U2X: Double the USART Transmission Speed**

This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.

Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

- **Bit 0 – MPCM: Multi-processor Communication Mode**

This bit enables the Multi-processor Communication mode. When the MPCM bit is written to one, all the incoming frames received by the USART receiver that do not contain address information will be ignored. The transmitter is unaffected by the MPCM setting. For more detailed information see “Multi-processor Communication Mode” on page 155.

USART Control and Status Register B – UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – RXCIE: RX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the RXC Flag. A USART Receive Complete Interrupt will be generated only if the RXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set.

- **Bit 6 – TXCIE: TX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the TXC Flag. A USART Transmit Complete Interrupt will be generated only if the TXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC bit in UCSRA is set.

- **Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable**

Writing this bit to one enables interrupt on the UDRE Flag. A Data Register Empty Interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE bit in UCSRA is set.

- **Bit 4 – RXEN: Receiver Enable**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FE, DOR, and PE Flags.

- **Bit 3 – TXEN: Transmitter Enable**

Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. The disabling of the Transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the transmit Shift Register and transmit Buffer Register

do not contain data to be transmitted. When disabled, the transmitter will no longer override the TxD port.

- **Bit 2 – UCSZ2: Character Size**

The UCSZ2 bits combined with the UCSZ1:0 bit in UCSRC sets the number of data bits (Character Size) in a frame the receiver and transmitter use.

- **Bit 1 – RXB8: Receive Data Bit 8**

RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDR.

- **Bit 0 – TXB8: Transmit Data Bit 8**

TXB8 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDR.

USART Control and Status Register C – UCSRC

Bit	7	6	5	4	3	2	1	0	
	URSEL	**UMSEL**	**UPM1**	**UPM0**	**USBS**	**UCSZ1**	**UCSZ0**	**UCPOL**	**UCSRC**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

The UCSRC Register shares the same I/O location as the UBRRH Register. See the “Accessing UBRRH/ UCSRC Registers” on page 156 section which describes how to access this register.

- **Bit 7 – URSEL: Register Select**

This bit selects between accessing the UCSRC or the UBRRH Register. It is read as one when reading UCSRC. The URSEL must be one when writing the UCSRC.

- **Bit 6 – UMSEL: USART Mode Select**

This bit selects between Asynchronous and Synchronous mode of operation.

Table 63. UMSEL Bit Settings

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

- **Bit 5:4 – UPM1:0: Parity Mode**

These bits enable and set type of parity generation and check. If enabled, the transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPM0 setting. If a mismatch is detected, the PE Flag in UCSRA will be set.

Table 64. UPM Bits Settings

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- **Bit 3 – USBS: Stop Bit Select**

This bit selects the number of Stop Bits to be inserted by the Transmitter. The Receiver ignores this setting.

Table 65. USBS Bit Settings

USBS	Stop Bit(s)
0	1-bit
1	2-bit

- **Bit 2:1 – UCSZ1:0: Character Size**

The UCSZ1:0 bits combined with the UCSZ2 bit in UCSRB sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

Table 66. UCSZ Bits Settings

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- **Bit 0 – UCPOL: Clock Polarity**

This bit is used for Synchronous mode only. Write this bit to zero when Asynchronous mode is used. The UCPOL bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK).

Table 67. UCPOL Bit Settings

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

USART Baud Rate Registers – UBRRL and UBRRH

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

The UBRRH Register shares the same I/O location as the UCSRC Register. See the “Accessing UBRRH/ UCSRC Registers” on page 156 section which describes how to access this register.

- **Bit 15 – URSEL: Register Select**

This bit selects between accessing the UBRRH or the UCSRC Register. It is read as zero when reading UBRRH. The URSEL must be zero when writing the UBRRH.

- **Bit 14:12 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRH is written.

- **Bit 11:0 – UBRR11:0: USART Baud Rate Register**

This is a 12-bit register which contains the USART baud rate. The UBRRH contains the four most significant bits, and the UBRRL contains the 8 least significant bits of the USART baud rate. Ongoing transmissions by the transmitter and receiver will be corrupted if the baud rate is changed. Writing UBRRL will trigger an immediate update of the baud rate prescaler.

Examples of Baud Rate Setting

For standard crystal and resonator frequencies, the most commonly used baud rates for asynchronous operation can be generated by using the UBRR settings in Table 68. UBRR values which yield an actual baud rate differing less than 0.5% from the target baud rate, are bold in the table. Higher error ratings are acceptable, but the receiver will have less noise resistance when the error ratings are high, especially for large serial frames (see "Asynchronous Operational Range" on page 153). The error values are calculated using the following equation:

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \bullet 100\%$$

Table 68. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max (1)	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

1. UBRR = 0, Error = 0.0%

Table 69. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 3.6864 MHz				f_{osc} = 4.0000 MHz				f_{osc} = 7.3728 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	–	–	0	-7.8%	–	–	0	0.0%	0	-7.8%	1	-7.8%
1M	–	–	–	–	–	–	–	–	–	–	0	-7.8%
Max (1)	230.4 kbps		460.8 kbps		250 kbps		0.5 Mbps		460.8 kbps		921.6 kbps	

1. UBRR = 0, Error = 0.0%

Table 70. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
Max (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, Error = 0.0%

Table 71. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 16.0000 MHz				f_{osc} = 18.4320 MHz				f_{osc} = 20.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max (1)	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

1. UBRR = 0, Error = 0.0%

Two-wire Serial Interface

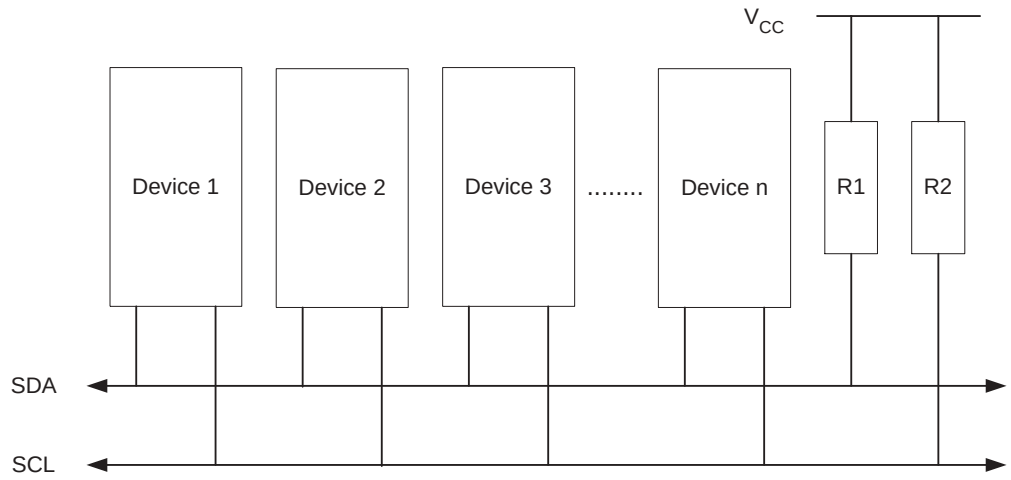
Features

- **Simple Yet Powerful and Flexible Communication Interface, Only Two Bus Lines Needed**
- **Both Master and Slave Operation Supported**
- **Device Can Operate as Transmitter or Receiver**
- **7-bit Address Space allows up to 128 Different Slave Addresses**
- **Multi-master Arbitration Support**
- **Up to 400 kHz Data Transfer Speed**
- **Slew-rate Limited Output Drivers**
- **Noise Suppression Circuitry Rejects Spikes on Bus Lines**
- **Fully Programmable Slave Address with General Call Support**
- **Address Recognition causes Wake-up when AVR is in Sleep Mode**

Two-wire Serial Interface Bus Definition

The Two-wire Serial Interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 different devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA). The only external hardware needed to implement the bus is a single pull-up resistor for each of the TWI bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

Figure 76. TWI Bus Interconnection



TWI Terminology

The following definitions are frequently encountered in this section.

Table 72. TWI Terminology

Term	Description
Master	The device that initiates and terminates a transmission. The master also generates the SCL clock.
Slave	The device addressed by a master.
Transmitter	The device placing data on the bus.
Receiver	The device reading data from the bus.

Electrical Interconnection

As depicted in Figure 76, both bus lines are connected to the positive supply voltage through pull-up resistors. The bus drivers of all TWI-compliant devices are open-drain or open-collector. This implements a wired-AND function which is essential to the operation of the interface. A low level on a TWI bus line is generated when one or more TWI devices output a zero. A high level is output when all TWI devices tri-state their outputs, allowing the pull-up resistors to pull the line high. Note that all AVR devices connected to the TWI bus must be powered in order to allow any bus operation.

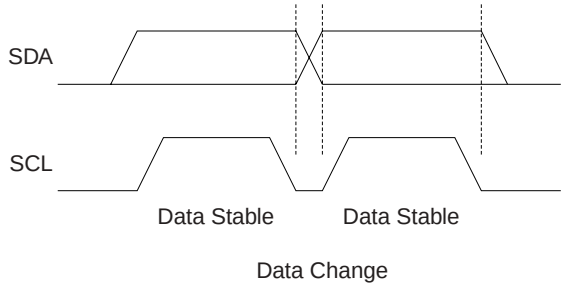
The number of devices that can be connected to the bus is only limited by the bus capacitance limit of 400 pF and the 7-bit slave address space. A detailed specification of the electrical characteristics of the TWI is given in “Two-wire Serial Interface Characteristics” on page 288. Two different sets of specifications are presented there, one relevant for bus speeds below 100 kHz, and one valid for bus speeds up to 400 kHz.

Data Transfer and Frame Format

Transferring Bits

Each data bit transferred on the TWI bus is accompanied by a pulse on the clock line. The level of the data line must be stable when the clock line is high. The only exception to this rule is for generating start and stop conditions.

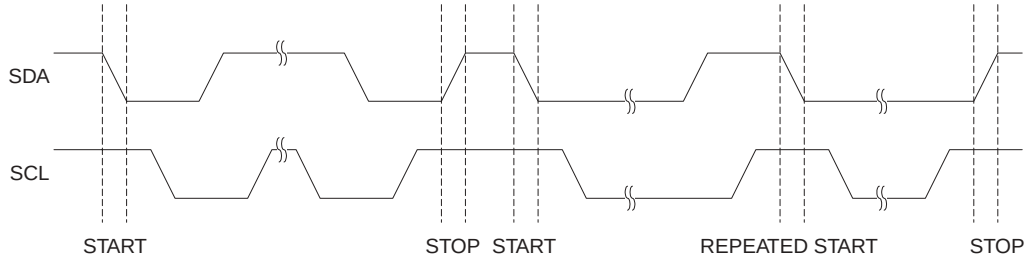
Figure 77. Data Validity



START and STOP Conditions

The master initiates and terminates a data transmission. The transmission is initiated when the master issues a START condition on the bus, and it is terminated when the master issues a STOP condition. Between a START and a STOP condition, the bus is considered busy, and no other master should try to seize control of the bus. A special case occurs when a new START condition is issued between a START and STOP condition. This is referred to as a REPEATED START condition, and is used when the master wishes to initiate a new transfer without releasing control of the bus. After a REPEATED START, the bus is considered busy until the next STOP. This is identical to the START behavior, and therefore START is used to describe both START and REPEATED START for the remainder of this datasheet, unless otherwise noted. As depicted below, START and STOP conditions are signalled by changing the level of the SDA line when the SCL line is high.

Figure 78. START, REPEATED START, and STOP Conditions



Address Packet Format

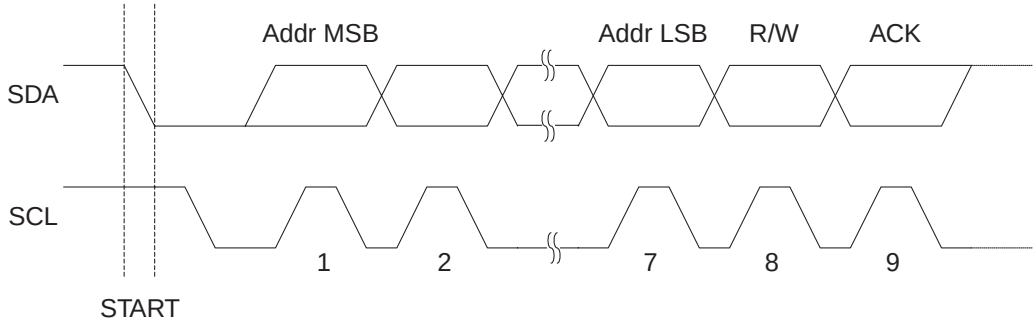
All address packets transmitted on the TWI bus are nine bits long, consisting of seven address bits, one READ/WRITE control bit and an acknowledge bit. If the READ/WRITE bit is set, a read operation is to be performed, otherwise a write operation should be performed. When a slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle. If the addressed slave is busy, or for some other reason can not service the master's request, the SDA line should be left high in the ACK clock cycle. The master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission. An address packet consisting of a slave address and a READ or a WRITE bit is called SLA+R or SLA+W, respectively.

The MSB of the address byte is transmitted first. Slave addresses can freely be allocated by the designer, but the address 0000 000 is reserved for a general call.

When a general call is issued, all slaves should respond by pulling the SDA line low in the ACK cycle. A general call is used when a master wishes to transmit the same message to several slaves in the system. When the general call address followed by a Write bit is transmitted on the bus, all slaves set up to acknowledge the general call will pull the SDA line low in the ack cycle. The following data packets will then be received by all the slaves that acknowledged the general call. Note that transmitting the general call address followed by a Read bit is meaningless, as this would cause contention if several slaves started transmitting different data.

All addresses of the format 1111 xxx should be reserved for future purposes.

Figure 79. Address Packet Format

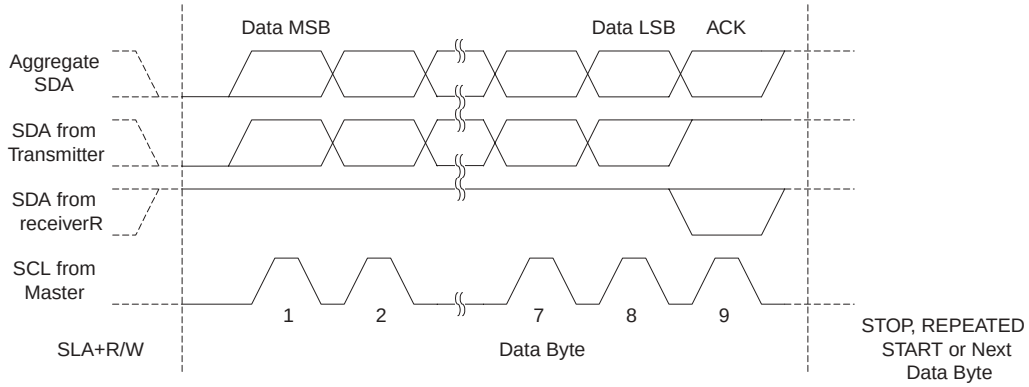


Data Packet Format

All data packets transmitted on the TWI bus are nine bits long, consisting of one data byte and an acknowledge bit. During a data transfer, the master generates the clock and the START and STOP conditions, while the receiver is responsible for acknowledging the reception. An Acknowledge (ACK) is signalled by the receiver pulling the SDA line low during the ninth SCL cycle. If the receiver leaves the SDA line high, a NACK is signalled. When the receiver has received the last byte, or for some reason cannot receive

any more bytes, it should inform the transmitter by sending a NACK after the final byte. The MSB of the data byte is transmitted first.

Figure 80. Data Packet Format

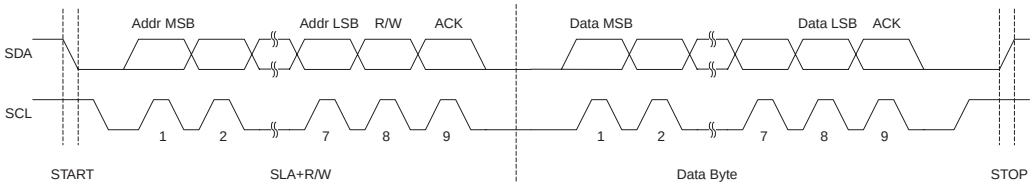


Combining Address and Data Packets into a Transmission

A transmission basically consists of a START condition, a SLA+R/W, one or more data packets and a STOP condition. An empty message, consisting of a START followed by a STOP condition, is illegal. Note that the wired-ANDing of the SCL line can be used to implement handshaking between the master and the slave. The slave can extend the SCL low period by pulling the SCL line low. This is useful if the clock speed set up by the master is too fast for the slave, or the slave needs extra time for processing between the data transmissions. The slave extending the SCL low period will not affect the SCL high period, which is determined by the master. As a consequence, the slave can reduce the TWI data transfer speed by prolonging the SCL duty cycle.

Figure 81 shows a typical data transmission. Note that several data bytes can be transmitted between the SLA+R/W and the STOP condition, depending on the software protocol implemented by the application software.

Figure 81. Typical Data Transmission



Multi-master Bus Systems, Arbitration and Synchronization

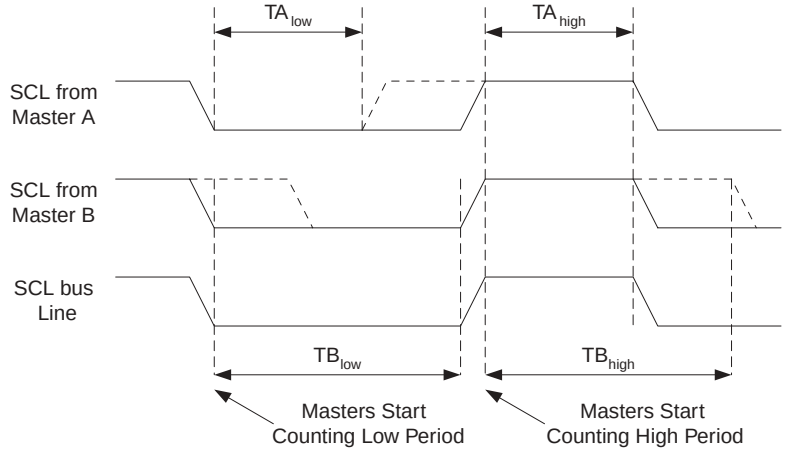
The TWI protocol allows bus systems with several masters. Special concerns have been taken in order to ensure that transmissions will proceed as normal, even if two or more masters initiate a transmission at the same time. Two problems arise in multi-master systems:

- An algorithm must be implemented allowing only one of the masters to complete the transmission. All other masters should cease transmission when they discover that they have lost the selection process. This selection process is called arbitration. When a contending master discovers that it has lost the arbitration process, it should immediately switch to slave mode to check whether it is being addressed by the winning master. The fact that multiple masters have started transmission at the same time should not be detectable to the slaves, i.e., the data being transferred on the bus must not be corrupted.

- Different masters may use different SCL frequencies. A scheme must be devised to synchronize the serial clocks from all masters, in order to let the transmission proceed in a lockstep fashion. This will facilitate the arbitration process.

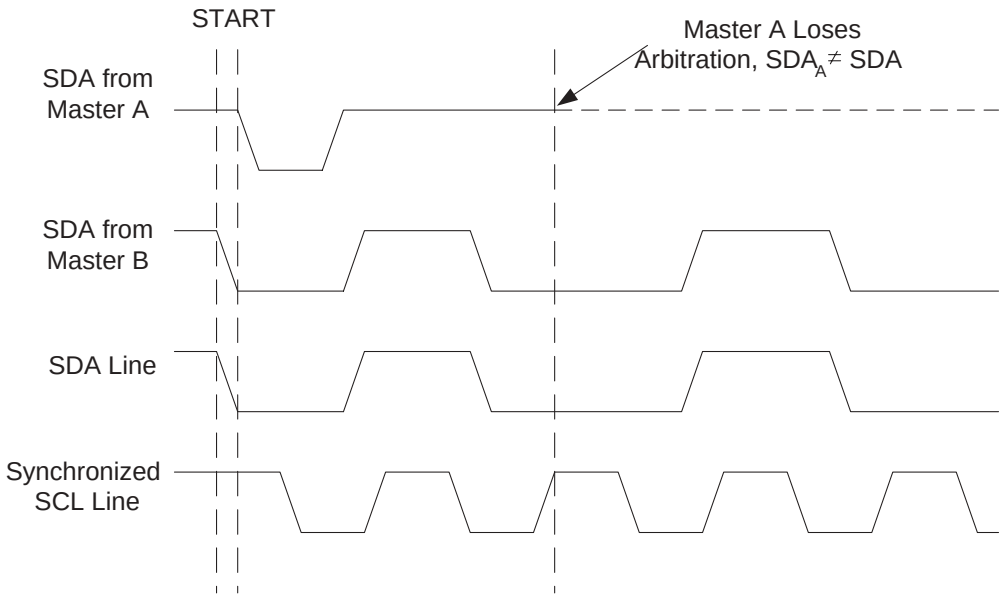
The wired-ANDing of the bus lines is used to solve both these problems. The serial clocks from all masters will be wired-ANDed, yielding a combined clock with a high period equal to the one from the master with the shortest high period. The low period of the combined clock is equal to the low period of the master with the longest low period. Note that all masters listen to the SCL line, effectively starting to count their SCL high and low time-out periods when the combined SCL line goes high or low, respectively.

Figure 82. SCL Synchronization between Multiple Masters



Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data. If the value read from the SDA line does not match the value the master had output, it has lost the arbitration. Note that a master can only lose arbitration when it outputs a high SDA value while another master outputs a low value. The losing master should immediately go to slave mode, checking if it is being addressed by the winning master. The SDA line should be left high, but losing masters are allowed to generate a clock signal until the end of the current data or address packet. Arbitration will continue until only one master remains, and this may take many bits. If several masters are trying to address the same slave, arbitration will continue into the data packet.

Figure 83. Arbitration between Two Masters



Note that arbitration is not allowed between:

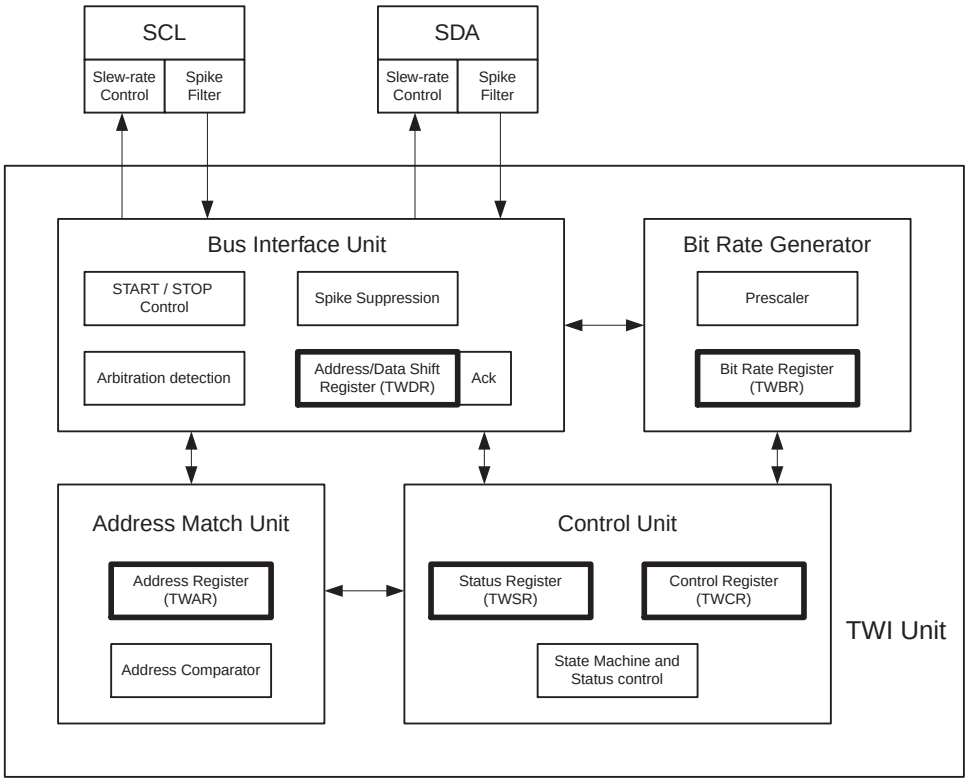
- A REPEATED START condition and a data bit
- A STOP condition and a data bit
- A REPEATED START and a STOP condition

It is the user software's responsibility to ensure that these illegal arbitration conditions never occur. This implies that in multi-master systems, all data transfers must use the same composition of SLA+R/W and data packets. In other words: All transmissions must contain the same number of data packets, otherwise the result of the arbitration is undefined.

Overview of the TWI Module

The TWI module is comprised of several submodules, as shown in Figure 84. All registers drawn in a thick line are accessible through the AVR data bus.

Figure 84. Overview of the TWI Module



SCL and SDA Pins

These pins interface the AVR TWI with the rest of the MCU system. The output drivers contain a slew-rate limiter in order to conform to the TWI specification. The input stages contain a spike suppression unit removing spikes shorter than 50 ns. Note that the internal pullups in the AVR pads can be enabled by setting the PORT bits corresponding to the SCL and SDA pins, as explained in the I/O Port section. The internal pull-ups can in some systems eliminate the need for external ones.

Bit Rate Generator Unit

This unit controls the period of SCL when operating in a Master mode. The SCL period is controlled by settings in the TWI Bit Rate Register (TWBR) and the Prescaler bits in the TWI Status Register (TWSR). Slave operation does not depend on Bit Rate or Prescaler settings, but the CPU clock frequency in the slave must be at least 16 times higher than the SCL frequency. Note that slaves may prolong the SCL low period, thereby reducing the average TWI bus clock period. The SCL frequency is generated according to the following equation:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

- TWBR = Value of the TWI Bit Rate Register
- TWPS = Value of the prescaler bits in the TWI Status Register

Note: TWBR should be 10 or higher if the TWI operates in Master mode. If TWBR is lower than 10, the master may produce an incorrect output on SDA and SCL for the reminder of the byte. The problem occurs when operating the TWI in Master mode, sending Start + SLA + R/W to a slave (a slave does not need to be connected to the bus for the condition to happen).

Bus Interface Unit

This unit contains the Data and Address Shift Register (TWDR), a START/STOP Controller and Arbitration detection hardware. The TWDR contains the address or data bytes to be transmitted, or the address or data bytes received. In addition to the 8-bit TWDR, the Bus Interface Unit also contains a register containing the (N)ACK bit to be transmitted or received. This (N)ACK Register is not directly accessible by the application software. However, when receiving, it can be set or cleared by manipulating the TWI Control Register (TWCR). When in Transmitter mode, the value of the received (N)ACK bit can be determined by the value in the TWSR.

The START/STOP Controller is responsible for generation and detection of START, REPEATED START, and STOP conditions. The START/STOP controller is able to detect START and STOP conditions even when the AVR MCU is in one of the sleep modes, enabling the MCU to wake up if addressed by a master.

If the TWI has initiated a transmission as master, the Arbitration Detection hardware continuously monitors the transmission trying to determine if arbitration is in process. If the TWI has lost an arbitration, the Control Unit is informed. Correct action can then be taken and appropriate status codes generated.

Address Match Unit

The Address Match unit checks if received address bytes match the 7-bit address in the TWI Address Register (TWAR). If the TWI General Call Recognition Enable (TWGCE) bit in the TWAR is written to one, all incoming address bits will also be compared against the General Call address. Upon an address match, the Control Unit is informed, allowing correct action to be taken. The TWI may or may not acknowledge its address, depending on settings in the TWCR. The Address Match unit is able to compare addresses even when the AVR MCU is in sleep mode, enabling the MCU to wake up if addressed by a master.

Control Unit

The Control unit monitors the TWI bus and generates responses corresponding to settings in the TWI Control Register (TWCR). When an event requiring the attention of the application occurs on the TWI bus, the TWI Interrupt Flag (TWINT) is asserted. In the next clock cycle, the TWI Status Register (TWSR) is updated with a status code identifying the event. The TWSR only contains relevant status information when the TWI Interrupt Flag is asserted. At all other times, the TWSR contains a special status code indicating that no relevant status information is available. As long as the TWINT Flag is set, the SCL line is held low. This allows the application software to complete its tasks before allowing the TWI transmission to continue.

The TWINT Flag is set in the following situations:

- After the TWI has transmitted a START/REPEATED START condition
- After the TWI has transmitted SLA+R/W
- After the TWI has transmitted an address byte
- After the TWI has lost arbitration
- After the TWI has been addressed by own slave address or general call
- After the TWI has received a data byte
- After a STOP or REPEATED START has been received while still addressed as a slave
- When a bus error has occurred due to an illegal START or STOP condition

TWI Register Description

TWI Bit Rate Register – TWBR

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..0 – TWI Bit Rate Register**

TWBR selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the Master modes. See “Bit Rate Generator Unit” on page 173 for calculating bit rates.

TWI Control Register – TWCR

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The TWCR is used to control the operation of the TWI. It is used to enable the TWI, to initiate a master access by applying a START condition to the bus, to generate a receiver acknowledge, to generate a stop condition, and to control halting of the bus while the data to be written to the bus are written to the TWDR. It also indicates a write collision if data is attempted written to TWDR while the register is inaccessible.

- **Bit 7 – TWINT: TWI Interrupt Flag**

This bit is set by hardware when the TWI has finished its current job and expects application software response. If the I-bit in SREG and TWIE in TWCR are set, the MCU will jump to the TWI Interrupt Vector. While the TWINT Flag is set, the SCL low period is stretched. The TWINT Flag must be cleared by software by writing a logic one to it. Note that this flag is not automatically cleared by hardware when executing the interrupt routine. Also note that clearing this flag starts the operation of the TWI, so all accesses to the TWI Address Register (TWAR), TWI Status Register (TWSR), and TWI Data Register (TWDR) must be complete before clearing this flag.

- **Bit 6 – TWEA: TWI Enable Acknowledge Bit**

The TWEA bit controls the generation of the acknowledge pulse. If the TWEA bit is written to one, the ACK pulse is generated on the TWI bus if the following conditions are met:

1. The device’s own slave address has been received.
2. A general call has been received, while the TWGCE bit in the TWAR is set.
3. A data byte has been received in Master Receiver or Slave Receiver mode.

By writing the TWEA bit to zero, the device can be virtually disconnected from the Two-wire Serial Bus temporarily. Address recognition can then be resumed by writing the TWEA bit to one again.

- **Bit 5 – TWSTA: TWI START Condition Bit**

The application writes the TWSTA bit to one when it desires to become a master on the Two-wire Serial Bus. The TWI hardware checks if the bus is available, and generates a START condition on the bus if it is free. However, if the bus is not free, the TWI waits

until a STOP condition is detected, and then generates a new START condition to claim the bus Master status. TWSTA must be cleared by software when the START condition has been transmitted.

- **Bit 4 – TWSTO: TWI STOP Condition Bit**

Writing the TWSTO bit to one in Master mode will generate a STOP condition on the Two-wire Serial Bus. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically. In slave mode, setting the TWSTO bit can be used to recover from an error condition. This will not generate a STOP condition, but the TWI returns to a well-defined unaddressed slave mode and releases the SCL and SDA lines to a high impedance state.

- **Bit 3 – TWWC: TWI Write Collision Flag**

The TWWC bit is set when attempting to write to the TWI Data Register – TWDR when TWINT is low. This flag is cleared by writing the TWDR Register when TWINT is high.

- **Bit 2 – TWEN: TWI Enable Bit**

The TWEN bit enables TWI operation and activates the TWI interface. When TWEN is written to one, the TWI takes control over the I/O pins connected to the SCL and SDA pins, enabling the slew-rate limiters and spike filters. If this bit is written to zero, the TWI is switched off and all TWI transmissions are terminated, regardless of any ongoing operation.

- **Bit 1 – Res: Reserved Bit**

This bit is a reserved bit and will always read as zero.

- **Bit 0 – TWIE: TWI Interrupt Enable**

When this bit is written to one, and the I-bit in SREG is set, the TWI interrupt request will be activated for as long as the TWINT Flag is high.

TWI Status Register – TWSR

Bit	7	6	5	4	3	2	1	0	
	TWS7	**TWS6**	**TWS5**	**TWS4**	**TWS3**	**–**	**TWPS1**	**TWPS0**	**TWSR**
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

- **Bits 7..3 – TWS: TWI Status**

These five bits reflect the status of the TWI logic and the Two-wire Serial Bus. The different status codes are described later in this section. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the prescaler bits to zero when checking the Status bits. This makes status checking independent of prescaler setting. This approach is used in this datasheet, unless otherwise noted.

- **Bit 2 – Res: Reserved Bit**

This bit is reserved and will always read as zero.

- **Bits 1..0 – TWPS: TWI Prescaler Bits**

These bits can be read and written, and control the bit rate prescaler.

Table 73. TWI Bit Rate Prescaler

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

To calculate bit rates, see “Bit Rate Generator Unit” on page 173. The value of TWPS1..0 is used in the equation.

TWI Data Register – TWDR

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

In Transmit mode, TWDR contains the next byte to be transmitted. In Receive mode, the TWDR contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when the TWI Interrupt Flag (TWINT) is set by hardware. Note that the Data Register cannot be initialized by the user before the first interrupt occurs. The data in TWDR remains stable as long as TWINT is set. While data is shifted out, data on the bus is simultaneously shifted in. TWDR always contains the last byte present on the bus, except after a wake up from a sleep mode by the TWI interrupt. In this case, the contents of TWDR is undefined. In the case of a lost bus arbitration, no data is lost in the transition from Master to Slave. Handling of the ACK bit is controlled automatically by the TWI logic, the CPU cannot access the ACK bit directly.

- **Bits 7..0 – TWD: TWI Data Register**

These eight bits contin the next data byte to be transmitted, or the latest data byte received on the Two-wire Serial Bus.

TWI (Slave) Address Register – TWAR

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

The TWAR should be loaded with the 7-bit slave address (in the seven most significant bits of TWAR) to which the TWI will respond when programmed as a slave transmitter or receiver. In multimaster systems, TWAR must be set in masters which can be addressed as slaves by other masters.

The LSB of TWAR is used to enable recognition of the general call address ($00). There is an associated address comparator that looks for the slave address (or general call address if enabled) in the received serial address. If a match is found, an interrupt request is generated.

- **Bits 7..1 – TWA: TWI (Slave) Address Register**

These seven bits constitute the slave address of the TWI unit.

- **Bit 0 – TWGCE: TWI General Call Recognition Enable Bit**

If set, this bit enables the recognition of a General Call given over the Two-wire Serial Bus.

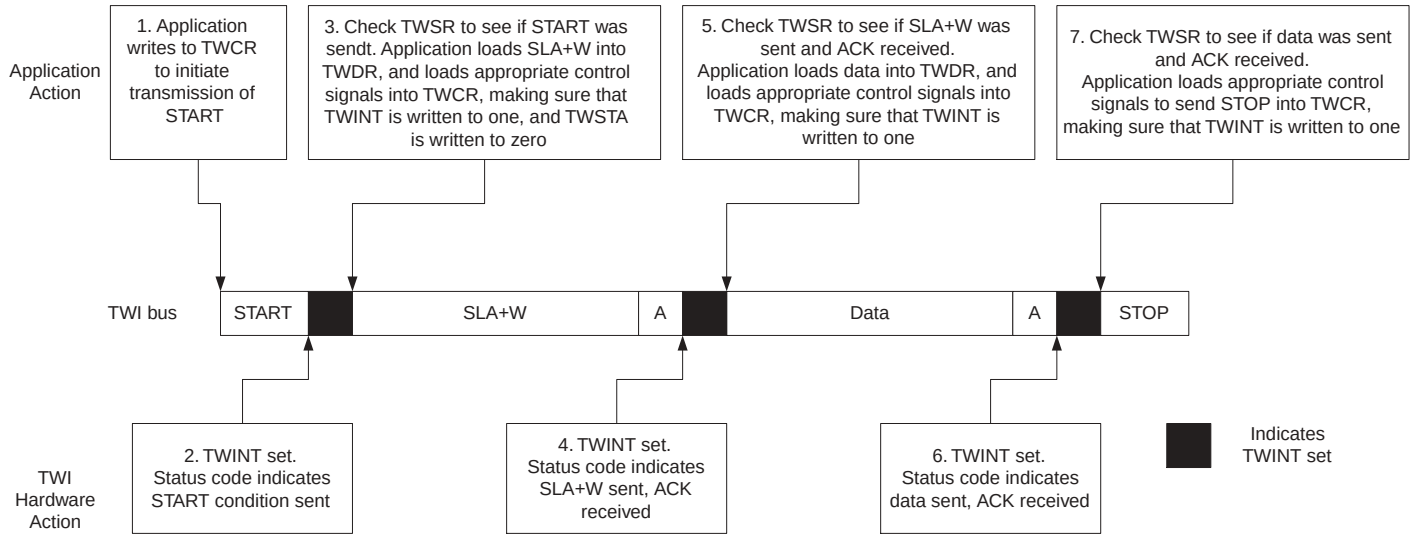
Using the TWI

The AVR TWI is byte-oriented and interrupt based. Interrupts are issued after all bus events, like reception of a byte or transmission of a START condition. Because the TWI is interrupt-based, the application software is free to carry on other operations during a TWI byte transfer. Note that the TWI Interrupt Enable (TWIE) bit in TWCR together with the Global Interrupt Enable bit in SREG allow the application to decide whether or not assertion of the TWINT Flag should generate an interrupt request. If the TWIE bit is cleared, the application must poll the TWINT Flag in order to detect actions on the TWI bus.

When the TWINT Flag is asserted, the TWI has finished an operation and awaits application response. In this case, the TWI Status Register (TWSR) contains a value indicating the current state of the TWI bus. The application software can then decide how the TWI should behave in the next TWI bus cycle by manipulating the TWCR and TWDR Registers.

Figure 85 is a simple example of how the application can interface to the TWI hardware. In this example, a master wishes to transmit a single data byte to a slave. This description is quite abstract, a more detailed explanation follows later in this section. A simple code example implementing the desired behaviour is also presented.

Figure 85. Interfacing the Application to the TWI in a Typical Transmission



1. The first step in a TWI transmission is to transmit a START condition. This is done by writing a specific value into TWCR, instructing the TWI hardware to transmit a START condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the Flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the START condition.
2. When the START condition has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the START condition has successfully been sent.

3. The application software should now examine the value of TWSR, to make sure that the START condition was successfully transmitted. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load SLA+W into TWDR. Remember that TWDR is used both for address and data. After TWDR has been loaded with the desired SLA+W, a specific value must be written to TWCR, instructing the TWI hardware to transmit the SLA+W present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the address packet.
4. When the address packet has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the address packet has successfully been sent. The status code will also reflect whether a slave acknowledged the packet or not.
5. The application software should now examine the value of TWSR, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load a data packet into TWDR. Subsequently, a specific value must be written to TWCR, instructing the TWI hardware to transmit the data packet present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the data packet.
6. When the data packet has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the data packet has successfully been sent. The status code will also reflect whether a slave acknowledged the packet or not.
7. The application software should now examine the value of TWSR, to make sure that the data packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must write a specific value to TWCR, instructing the TWI hardware to transmit a STOP condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the STOP condition. Note that TWINT is NOT set after a STOP condition has been sent.

Even though this example is simple, it shows the principles involved in all TWI transmissions. These can be summarized as follows:

- When the TWI has finished an operation and expects application response, the TWINT Flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT Flag is set, the user must update all TWI Registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI Register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be

set. Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

In the following an assembly and C implementation of the example is given. Note that the code below assumes that several definitions have been made, for example by using include-files.

	Assembly code example	C example	Comments
1	`ldi r16, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)` `out TWCR, r16`	`TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)`	Send START condition

2	`wait1:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait1`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			Wait for TWINT Flag set. This indicates that the START condition has been transmitted

3	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, START`
	`brne ERROR`
		`if ((TWSR & 0xF8) != START)`
		`ERROR();`
			Check value of TWI Status Register. Mask prescaler bits. If status different from START go to ERROR

	`ldi r16, SLA_W`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT) | (1<<TWEN)`
	`out TWCR, r16`
		`TWDR = SLA_W;`
		`TWCR = (1<<TWINT) | (1<<TWEN);`
			Load SLA_W into TWDR Register. Clear TWINT bit in TWCR to start transmission of address

4	`wait2:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait2`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			Wait for TWINT Flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received.

5	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_SLA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_SLA_ACK)`
		`ERROR();`
			Check value of TWI Status Register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR

	`ldi r16, DATA`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT) | (1<<TWEN)`
	`out TWCR, r16`
		`TWDR = DATA;`
		`TWCR = (1<<TWINT) | (1<<TWEN);`
			Load DATA into TWDR Register. Clear TWINT bit in TWCR to start transmission of data

6	`wait3:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait3`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			Wait for TWINT Flag set. This indicates that the DATA has been transmitted, and ACK/NACK has been received.

7	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_DATA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_DATA_ACK)`
		`ERROR();`
			Check value of TWI Status Register. Mask prescaler bits. If status different from MT_DATA_ACK go to ERROR

	`ldi r16, (1<<TWINT)|(1<<TWEN)|(1<<TWSTO)` `out TWCR, r16`	`TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);`	Transmit STOP condition

Transmission Modes

The TWI can operate in one of four major modes. These are named Master Transmitter (MT), Master Receiver (MR), Slave Transmitter (ST) and Slave Receiver (SR). Several of these modes can be used in the same application. As an example, the TWI can use MT mode to write data into a TWI EEPROM, MR mode to read the data back from the EEPROM. If other masters are present in the system, some of these might transmit data to the TWI, and then SR mode would be used. It is the application software that decides which modes are legal.

The following sections describe each of these modes. Possible status codes are described along with figures detailing data transmission in each of the modes. These figures contain the following abbreviations:

S: START condition

Rs: REPEATED START condition

R: Read bit (high level at SDA)

W: Write bit (low level at SDA)

A: Acknowledge bit (low level at SDA)

$\overline{A}$: Not acknowledge bit (high level at SDA)

Data: 8-bit data byte

P: STOP condition

SLA: Slave Address

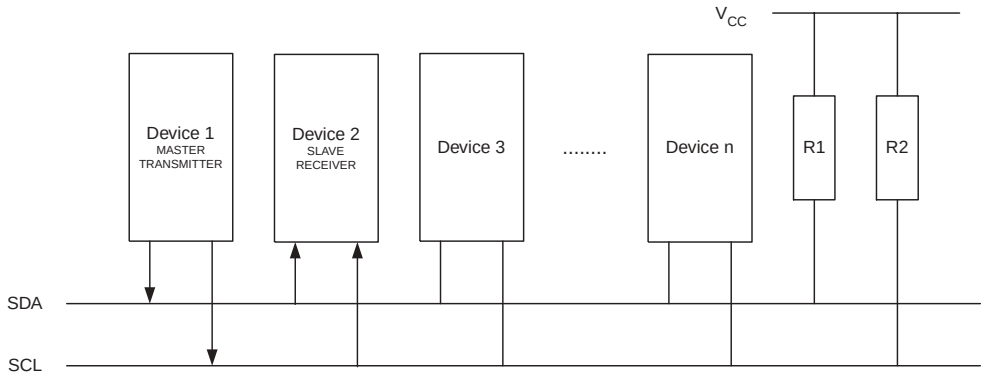
In Figure 87 to Figure 93, circles are used to indicate that the TWINT Flag is set. The numbers in the circles show the status code held in TWSR, with the prescaler bits masked to zero. At these points, actions must be taken by the application to continue or complete the TWI transfer. The TWI transfer is suspended until the TWINT Flag is cleared by software.

When the TWINT Flag is set, the status code in TWSR is used to determine the appropriate software action. For each status code, the required software action and details of the following serial transfer are given in Table 74 to Table 77. Note that the prescaler bits are masked to zero in these tables.

Master Transmitter Mode

In the Master Transmitter mode, a number of data bytes are transmitted to a slave receiver (see Figure 86). In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether Master Transmitter or Master Receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 86. Data Transfer in Master Transmitter Mode



A START condition is sent by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

TWEN must be set to enable the Two-wire Serial Interface, TWSTA must be written to one to transmit a START condition and TWINT must be written to one to clear the TWINT Flag. The TWI will then test the Two-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT Flag is set by hardware, and the status code in TWSR will be $08 (See Table 74). In order to enter MT mode, SLA+W must be transmitted. This is done by writing SLA+W to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	0	X	1	0	X

When SLA+W have been transmitted and an acknowledgement bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in master mode are $18, $20, or $38. The appropriate action to be taken for each of these status codes is detailed in Table 74.

When SLA+W has been successfully transmitted, a data packet should be transmitted. This is done by writing the data byte to TWDR. TWDR must only be written when TWINT is high. If not, the access will be discarded, and the Write Collision bit (TWWC) will be set in the TWCR Register. After updating TWDR, the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	0	X	1	0	X

This scheme is repeated until the last byte has been sent and the transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	1	X	1	0	X

A REPEATED START condition is generated by writing the following value to TWCR:

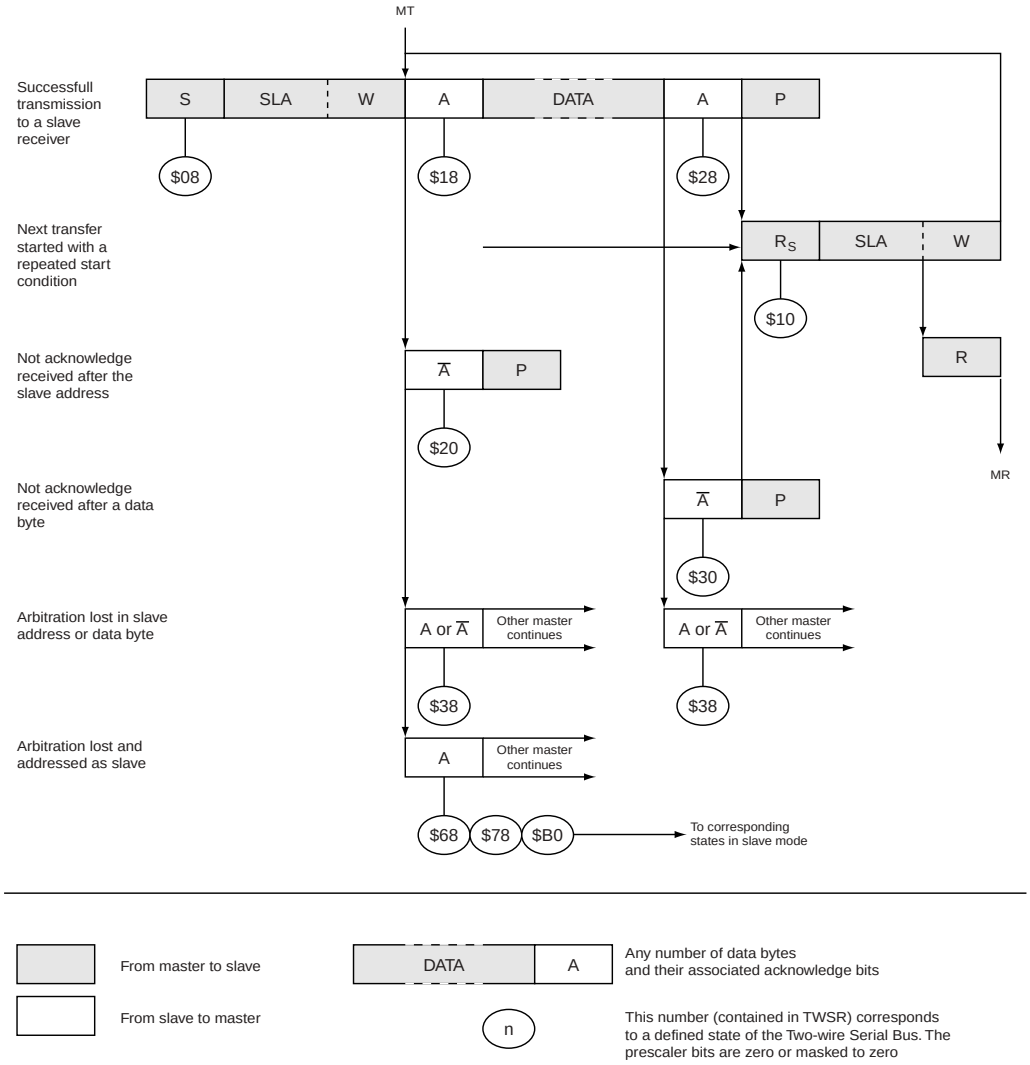
TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

After a repeated START condition (state $10) the Two-wire Serial Interface can access the same slave again, or a new slave without transmitting a STOP condition. Repeated START enables the master to switch between slaves, master transmitter mode and master receiver mode without losing control of the bus.

Table 74. Status Codes for Master Transmitter Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
$08	A START condition has been transmitted	Load SLA+W	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
$10	A repeated START condition has been transmitted	Load SLA+W or	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
		Load SLA+R	0	0	1	X	SLA+R will be transmitted; Logic will switch to Master Receiver mode
$18	SLA+W has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be Reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be Reset
$20	SLA+W has been transmitted; NOT ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
$28	Data byte has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
$30	Data byte has been transmitted; NOT ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
$38	Arbitration lost in SLA+W or data bytes	No TWDR action or	0	0	1	X	Two-wire Serial Bus will be released and not addressed slave mode entered
		No TWDR action	1	0	1	X	A START condition will be transmitted when the bus becomes free

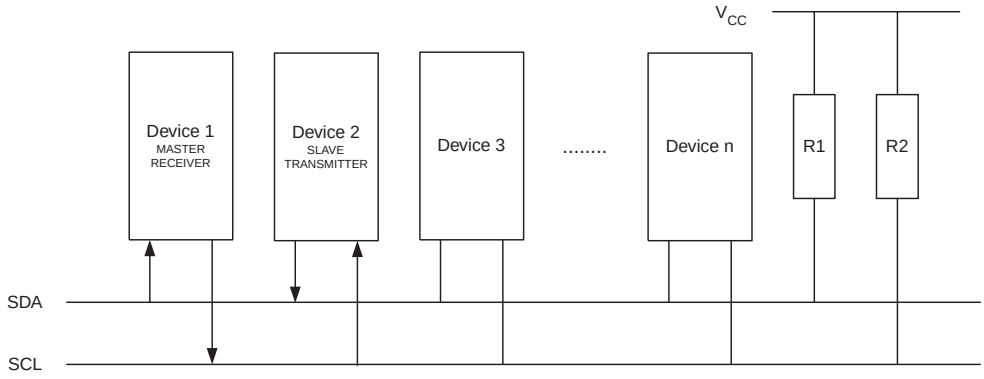
Figure 87. Formats and States in the Master Transmitter Mode



Master Receiver Mode

In the Master Receiver mode, a number of data bytes are received from a slave transmitter (see Figure 88). In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether Master Transmitter or Master Receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 88. Data Transfer in Master Receiver Mode



A START condition is sent by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

TWEN must be written to one to enable the Two-wire Serial Interface, TWSTA must be written to one to transmit a START condition and TWINT must be set to clear the TWINT Flag. The TWI will then test the Two-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT Flag is set by hardware, and the status code in TWSR will be $08 (See Table 74). In order to enter MR mode, SLA+R must be transmitted. This is done by writing SLA+R to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	0	X	1	0	X

When SLA+R have been transmitted and an acknowledgement bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in master mode are $38, $40, or $48. The appropriate action to be taken for each of these status codes is detailed in Table 75. Received data can be read from the TWDR Register when the TWINT Flag is set high by hardware. This scheme is repeated until the last byte has been received. After the last byte has been received, the MR should inform the ST by sending a NACK after the last received data byte. The transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	1	X	1	0	X

A REPEATED START condition is generated by writing the following value to TWCR:

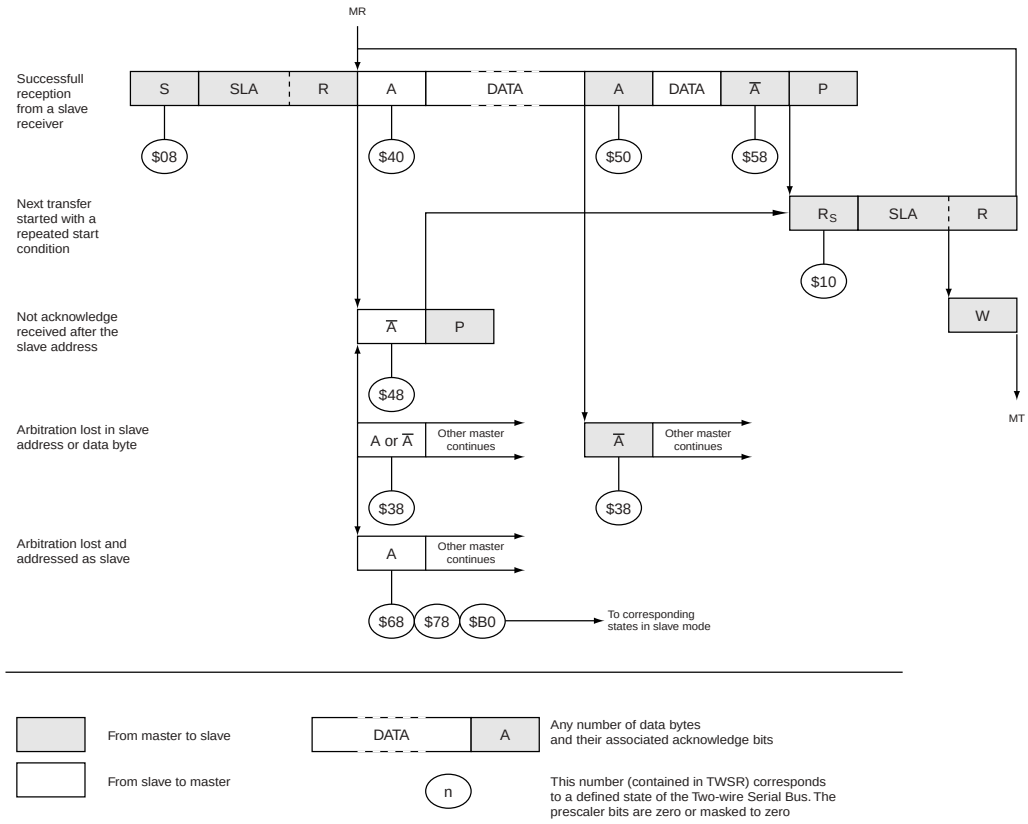
TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

After a repeated START condition (state $10) the Two-wire Serial Interface can access the same slave again, or a new slave without transmitting a STOP condition. Repeated START enables the master to switch between slaves, Master Transmitter mode and Master Receiver mode without losing control over the bus.

Table 75. Status Codes for Master Receiver Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
$08	A START condition has been transmitted	Load SLA+R	0	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
$10	A repeated START condition has been transmitted	Load SLA+R or	0	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
		Load SLA+W	0	0	1	X	SLA+W will be transmitted Logic will switch to masTer Transmitter mode
$38	Arbitration lost in SLA+R or NOT ACK bit	No TWDR action or	0	0	1	X	Two-wire Serial Bus will be released and not addressed slave mode will be entered
		No TWDR action	1	0	1	X	A START condition will be transmitted when the bus becomes free
$40	SLA+R has been transmitted; ACK has been received	No TWDR action or	0	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	0	0	1	1	Data byte will be received and ACK will be returned
$48	SLA+R has been transmitted; NOT ACK has been received	No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
$50	Data byte has been received; ACK has been returned	Read data byte or	0	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	0	0	1	1	Data byte will be received and ACK will be returned
$58	Data byte has been received; NOT ACK has been returned	Read data byte or	1	0	1	X	Repeated START will be transmitted
		Read data byte or	0	1	1	X	STOP condition will be transmitted and TWSTO Flag will be reset
		Read data byte	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset

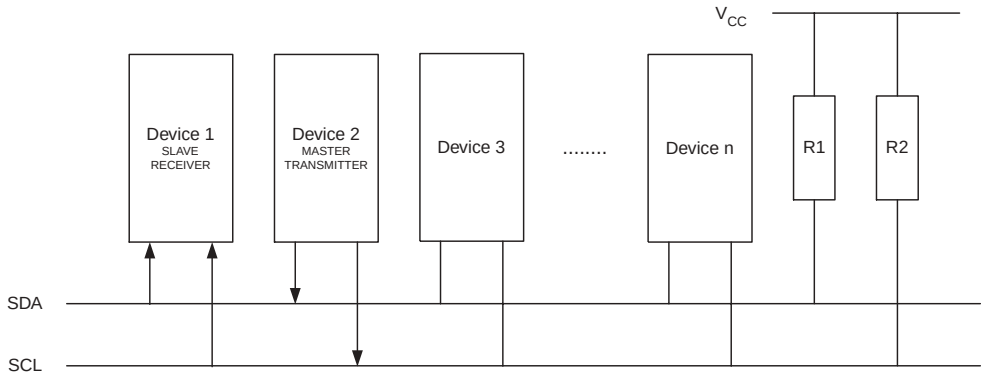
Figure 89. Formats and States in the Master Receiver Mode



Slave Receiver Mode

In the Slave Receiver mode, a number of data bytes are received from a master transmitter (see Figure 90). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 90. Data Transfer in Slave Receiver Mode



To initiate the Slave Receiver mode, TWAR and TWCR must be initialized as follows:

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Value	Device's Own Slave Address							

The upper seven bits are the address to which the Two-wire Serial Interface will respond when addressed by a master. If the LSB is set, the TWI will respond to the general call address ($00), otherwise it will ignore the general call address.

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	0	1	0	0	0	1	0	X

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "0" (write), the TWI will operate in SR mode, otherwise ST mode is entered. After its own slave address and the write bit have been received, the TWINT Flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 76. The Slave Receiver mode may also be entered if arbitration is lost while the TWI is in the Master mode (see states $68 and $78).

If the TWEA bit is reset during a transfer, the TWI will return a "Not Acknowledge" ("1") to SDA after the next received data byte. This can be used to indicate that the slave is not able to receive any more bytes. While TWEA is zero, the TWI does not acknowledge its own slave address. However, the Two-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the Two-wire Serial Bus.

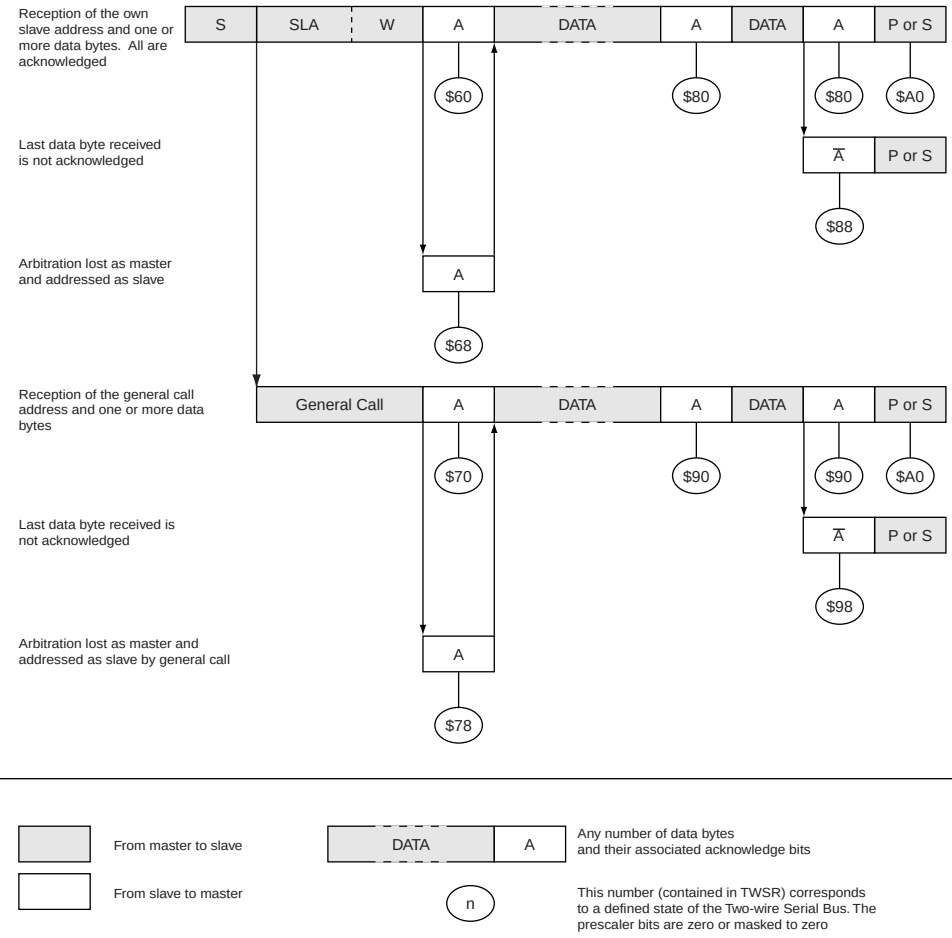
In all sleep modes other than Idle Mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the Two-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock low during the wake up and until the TWINT Flag is cleared (by writing it to one). Further data reception will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the Two-wire Serial Interface Data Register – TWDR does not reflect the last byte present on the bus when waking up from these sleep modes.

Table 76. Status Codes for Slave Receiver Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
$60	Own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
$68	Arbitration lost in SLA+R/W as master; own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
$70	General call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
$78	Arbitration lost in SLA+R/W as master; General call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
$80	Previously addressed with own SLA+W; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned
$88	Previously addressed with own SLA+W; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		Read data byte or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		Read data byte	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
$90	Previously addressed with general call; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned
$98	Previously addressed with general call; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		Read data byte or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		Read data byte	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
$A0	A STOP condition or repeated START condition has been received while still addressed as slave	No action	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
			0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
			1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
			1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free

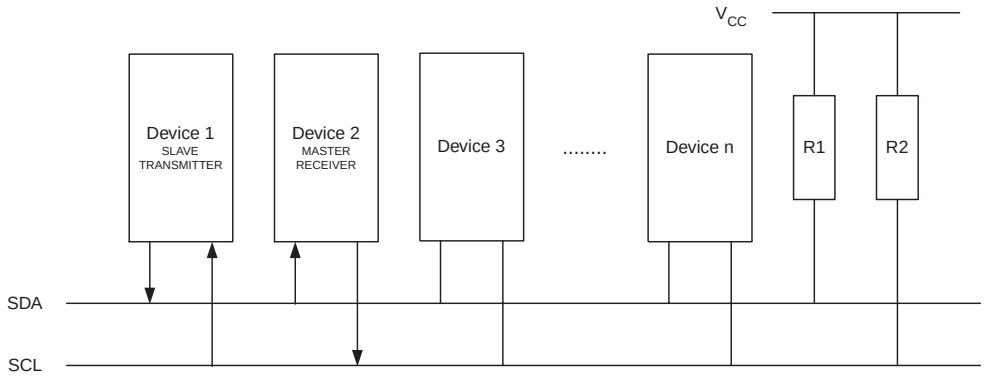
Figure 91. Formats and States in the Slave Receiver Mode



Slave Transmitter Mode

In the Slave Transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 92). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 92. Data Transfer in Slave Transmitter Mode



To initiate the Slave Transmitter mode, TWAR and TWCR must be initialized as follows:

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Value	Device's Own Slave Address							

The upper seven bits are the address to which the Two-wire Serial Interface will respond when addressed by a master. If the LSB is set, the TWI will respond to the general call address ($00), otherwise it will ignore the general call address.

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	0	1	0	0	0	1	0	X

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "1" (read), the TWI will operate in ST mode, otherwise SR mode is entered. After its own slave address and the write bit have been received, the TWINT Flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 77. The slave transmitter mode may also be entered if arbitration is lost while the TWI is in the Master mode (see state $B0).

If the TWEA bit is written to zero during a transfer, the TWI will transmit the last byte of the transfer. State $C0 or state $C8 will be entered, depending on whether the master receiver transmits a NACK or ACK after the final byte. The TWI is switched to the not addressed Slave mode, and will ignore the master if it continues the transfer. Thus the master receiver receives all "1" as serial data. State $C8 is entered if the master demands additional data bytes (by transmitting ACK), even though the slave has transmitted the last byte (TWEA zero and expecting NACK from the master).

While TWEA is zero, the TWI does not respond to its own slave address. However, the Two-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the Two-wire Serial Bus.

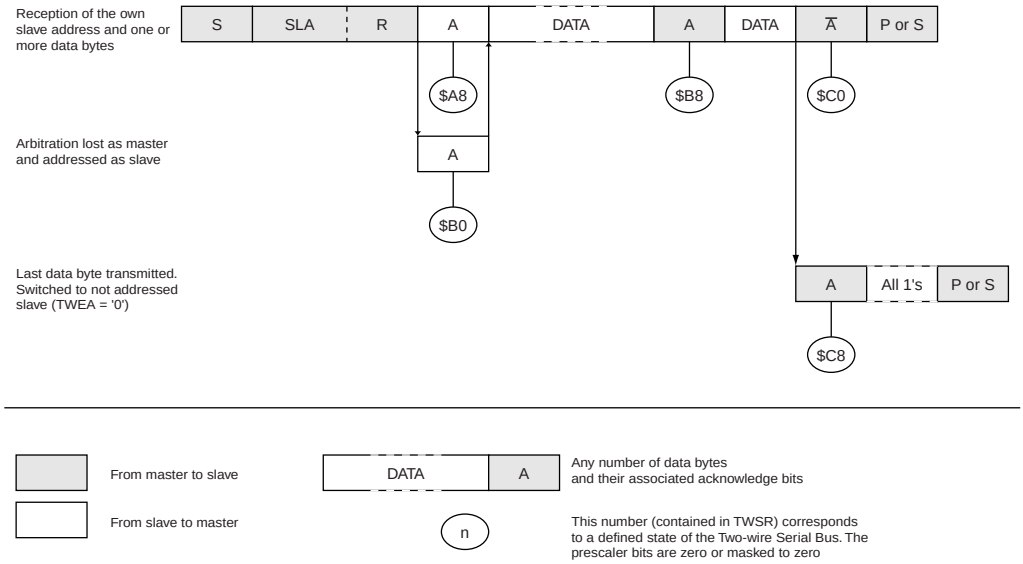
In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the Two-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock will low during the wake up and until the TWINT Flag is cleared (by writing it to one). Further data transmission will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the Two-wire Serial Interface Data Register – TWDR does not reflect the last byte present on the bus when waking up from these sleep modes.

Table 77. Status Codes for Slave Transmitter Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
$A8	Own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
$B0	Arbitration lost in SLA+R/W as master; own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
$B8	Data byte in TWDR has been transmitted; ACK has been received	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
$C0	Data byte in TWDR has been transmitted; NOT ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		No TWDR action or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		No TWDR action or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
$C8	Last data byte in TWDR has been transmitted (TWEA = “0”); ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		No TWDR action or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		No TWDR action or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free

Figure 93. Formats and States in the Slave Transmitter Mode



Miscellaneous States

There are two status codes that do not correspond to a defined TWI state, see Table 78.

Status $F8 indicates that no relevant information is available because the TWINT Flag is not set. This occurs between other states, and when the TWI is not involved in a serial transfer.

Status $00 indicates that a bus error has occurred during a Two-wire Serial Bus transfer. A bus error occurs when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. When a bus error occurs, TWINT is set. To recover from a bus error, the TWSTO Flag must set and TWINT must be cleared by writing a logic one to it. This causes the TWI to enter the not addressed slave mode and to clear the TWSTO Flag (no other bits in TWCR are affected). The SDA and SCL lines are released, and no STOP condition is transmitted.

Table 78. Miscellaneous States

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
$F8	No relevant state information available; TWINT = “0”	No TWDR action	No TWCR action				Wait or proceed current transfer
$00	Bus error due to an illegal START or STOP condition	No TWDR action	0	1	1	X	Only the internal hardware is affected, no STOP condition is sent on the bus. In all cases, the bus is released and TWSTO is cleared.

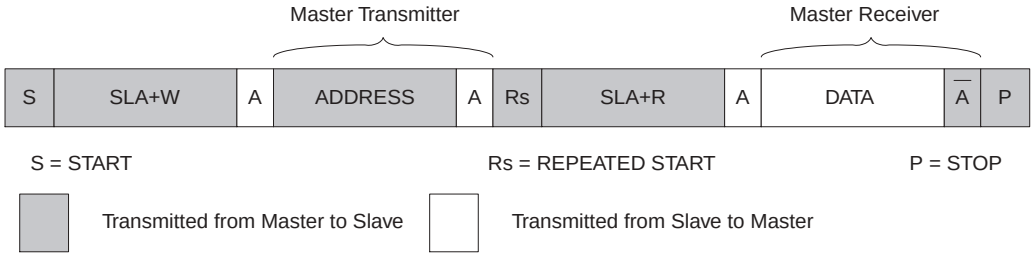
Combining Several TWI Modes

In some cases, several TWI modes must be combined in order to complete the desired action. Consider for example reading data from a serial EEPROM. Typically, such a transfer involves the following steps:

1. The transfer must be initiated
2. The EEPROM must be instructed what location should be read
3. The reading must be performed
4. The transfer must be finished

Note that data is transmitted both from master to slave and vice versa. The master must instruct the slave what location it wants to read, requiring the use of the MT mode. Subsequently, data must be read from the slave, implying the use of the MR mode. Thus, the transfer direction must be changed. The master must keep control of the bus during all these steps, and the steps should be carried out as an atomical operation. If this principle is violated in a multimaster system, another master can alter the data pointer in the EEPROM between steps 2 and 3, and the master will read the wrong data location. Such a change in transfer direction is accomplished by transmitting a REPEATED START between the transmission of the address byte and reception of the data. After a REPEATED START, the master keeps ownership of the bus. The following figure shows the flow in this transfer.

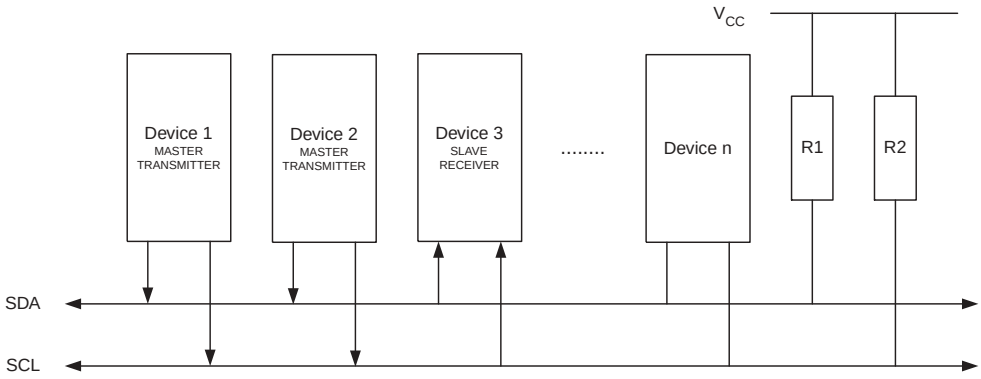
Figure 94. Combining Several TWI Modes to Access a Serial EEPROM



Multi-master Systems and Arbitration

If multiple masters are connected to the same bus, transmissions may be initiated simultaneously by one or more of them. The TWI standard ensures that such situations are handled in such a way that one of the masters will be allowed to proceed with the transfer, and that no data will be lost in the process. An example of an arbitration situation is depicted below, where two masters are trying to transmit data to a slave receiver.

Figure 95. An Arbitration Example

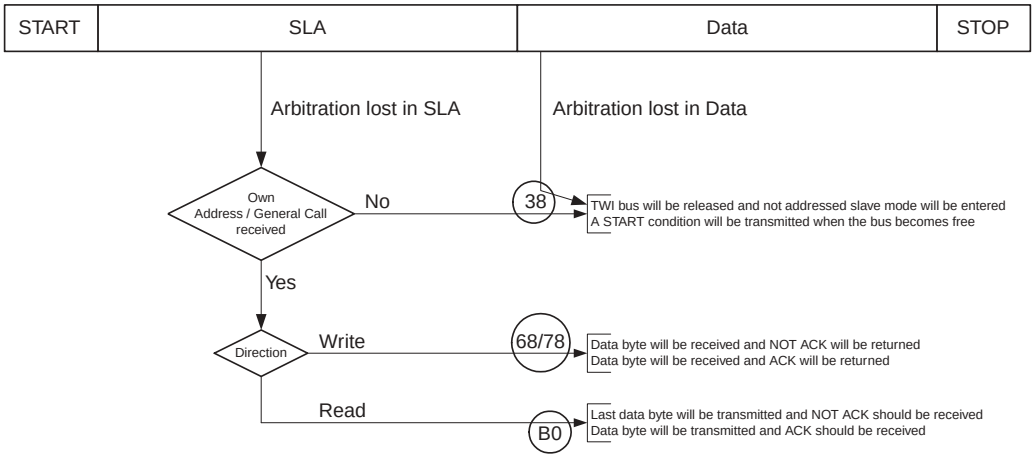


Several different scenarios may arise during arbitration, as described below:

- Two or more masters are performing identical communication with the same slave. In this case, neither the slave nor any of the masters will know about the bus contention.
- Two or more masters are accessing the same slave with different data or direction bit. In this case, arbitration will occur, either in the READ/WRITE bit or in the data bits. The masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Losing masters will switch to not addressed slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.
- Two or more masters are accessing different slaves. In this case, arbitration will occur in the SLA bits. Masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Masters losing arbitration in SLA will switch to slave mode to check if they are being addressed by the winning master. If addressed, they will switch to SR or ST mode, depending on the value of the READ/WRITE bit. If they are not being addressed, they will switch to not addressed slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.

This is summarized in Figure 96. Possible status values are given in circles.

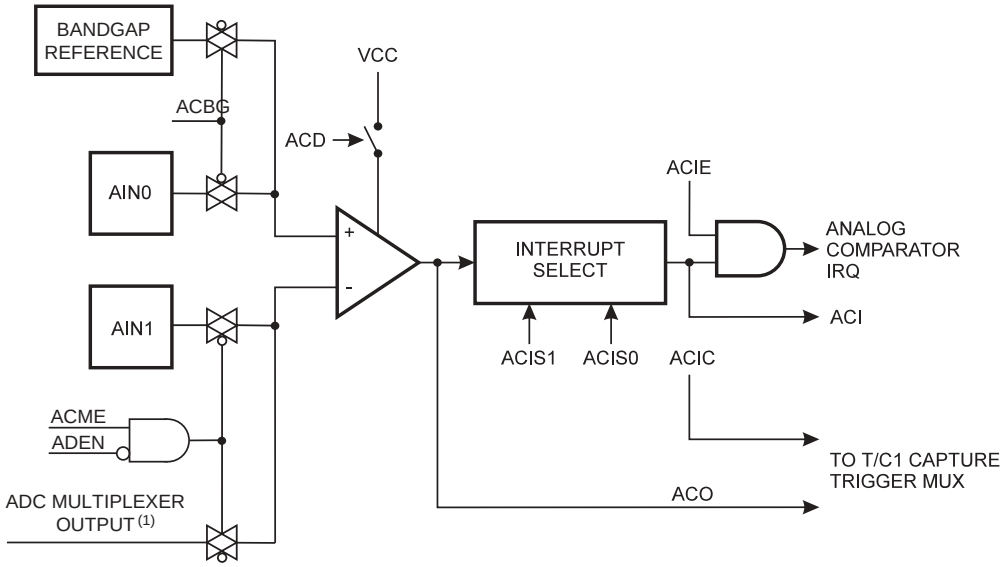
Figure 96. Possible Status Codes Caused by Arbitration



Analog Comparator

The Analog Comparator compares the input values on the positive pin AIN0 and negative pin AIN1. When the voltage on the positive pin AIN0 is higher than the voltage on the negative pin AIN1, the Analog Comparator Output, ACO, is set. The comparator's output can be set to trigger the Timer/Counter1 Input Capture function. In addition, the comparator can trigger a separate interrupt, exclusive to the Analog Comparator. The user can select Interrupt triggering on comparator output rise, fall or toggle. A block diagram of the comparator and its surrounding logic is shown in Figure 97.

Figure 97. Analog Comparator Block Diagram(2)



Notes: 1. See Table 80 on page 198.
2. Refer to Figure 1 on page 2 and Table 25 on page 55 for Analog Comparator pin placement.

Special Function IO Register – SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 3 – ACME: Analog Comparator Multiplexer Enable**

When this bit is written logic one and the ADC is switched off (ADEN in ADCSRA is zero), the ADC multiplexer selects the negative input to the Analog Comparator. When this bit is written logic zero, AIN1 is applied to the negative input of the Analog Comparator. For a detailed description of this bit, see "Analog Comparator Multiplexed Input" on page 198.

Analog Comparator Control and Status Register – ACSR

Bit	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

- **Bit 7 – ACD: Analog Comparator Disable**

When this bit is written logic one, the power to the Analog Comparator is switched off. This bit can be set at any time to turn off the Analog Comparator. This will reduce power consumption in active and Idle mode. When changing the ACD bit, the Analog Comparator Interrupt must be disabled by clearing the ACIE bit in ACSR. Otherwise an interrupt can occur when the bit is changed.

- **Bit 6 – ACBG: Analog Comparator Bandgap Select**

When this bit is set, a fixed bandgap reference voltage replaces the positive input to the Analog Comparator. When this bit is cleared, AIN0 is applied to the positive input of the Analog Comparator. See “Internal Voltage Reference” on page 39.

- **Bit 5 – ACO: Analog Comparator Output**

The output of the Analog Comparator is synchronized and then directly connected to ACO. The synchronization introduces a delay of 1 - 2 clock cycles.

- **Bit 4 – ACI: Analog Comparator Interrupt Flag**

This bit is set by hardware when a comparator output event triggers the interrupt mode defined by ACIS1 and ACIS0. The Analog Comparator Interrupt routine is executed if the ACIE bit is set and the I-bit in SREG is set. ACI is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ACI is cleared by writing a logic one to the flag.

- **Bit 3 – ACIE: Analog Comparator Interrupt Enable**

When the ACIE bit is written logic one and the I-bit in the Status Register is set, the Analog Comparator Interrupt is activated. When written logic zero, the interrupt is disabled.

- **Bit 2 – ACIC: Analog Comparator Input Capture Enable**

When written logic one, this bit enables the Input Capture function in Timer/Counter1 to be triggered by the Analog Comparator. The comparator output is in this case directly connected to the Input Capture front-end logic, making the comparator utilize the noise canceler and edge select features of the Timer/Counter1 Input Capture interrupt. When written logic zero, no connection between the Analog Comparator and the Input Capture function exists. To make the comparator trigger the Timer/Counter1 Input Capture interrupt, the TICIE1 bit in the Timer Interrupt Mask Register (TIMSK) must be set.

- **Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select**

These bits determine which comparator events that trigger the Analog Comparator interrupt. The different settings are shown in Table 79.

Table 79. ACIS1/ACIS0 Settings

ACIS1	ACIS0	Interrupt Mode
0	0	Comparator Interrupt on Output Toggle
0	1	Reserved
1	0	Comparator Interrupt on Falling Output Edge
1	1	Comparator Interrupt on Rising Output Edge

When changing the ACIS1/ACIS0 bits, the Analog Comparator Interrupt must be disabled by clearing its Interrupt Enable bit in the ACSR Register. Otherwise an interrupt can occur when the bits are changed.

Analog Comparator Multiplexed Input

It is possible to select any of the ADC7..0 pins to replace the negative input to the Analog Comparator. The ADC multiplexer is used to select this input, and consequently, the ADC must be switched off to utilize this feature. If the Analog Comparator Multiplexer Enable bit (ACME in SFIOR) is set and the ADC is switched off (ADEN in ADCSRA is zero), MUX2..0 in ADMUX select the input pin to replace the negative input to the Analog Comparator, as shown in Table 80. If ACME is cleared or ADEN is set, AIN1 is applied to the negative input to the Analog Comparator.

Table 80. Analog Comparator Multiplexed Input

ACME	ADEN	MUX2..0	Analog Comparator Negative Input
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

Analog to Digital Converter

Features

- **10-bit Resolution**
- **0.5 LSB Integral Non-linearity**
- **±2 LSB Absolute Accuracy**
- **65 - 260 µs Conversion Time**
- **Up to 15 kSPS at Maximum Resolution**
- **8 Multiplexed Single Ended Input Channels**
- **7 Differential Input Channels**
- **2 Differential Input Channels with Optional Gain of 10x and 200x**(1)
- **Optional Left adjustment for ADC Result Readout**
- **0 - V_{CC} ADC Input Voltage Range**
- **Selectable 2.56V ADC Reference Voltage**
- **Free Running or Single Conversion Mode**
- **ADC Start Conversion by Auto Triggering on Interrupt Sources**
- **Interrupt on ADC Conversion Complete**
- **Sleep Mode Noise Canceler**

Note: 1. The differential input channels are not tested for devices in PDIP Package. This feature is only guaranteed to work for devices in TQFP and MLF Packages

The ATmega32 features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows 8 single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND).

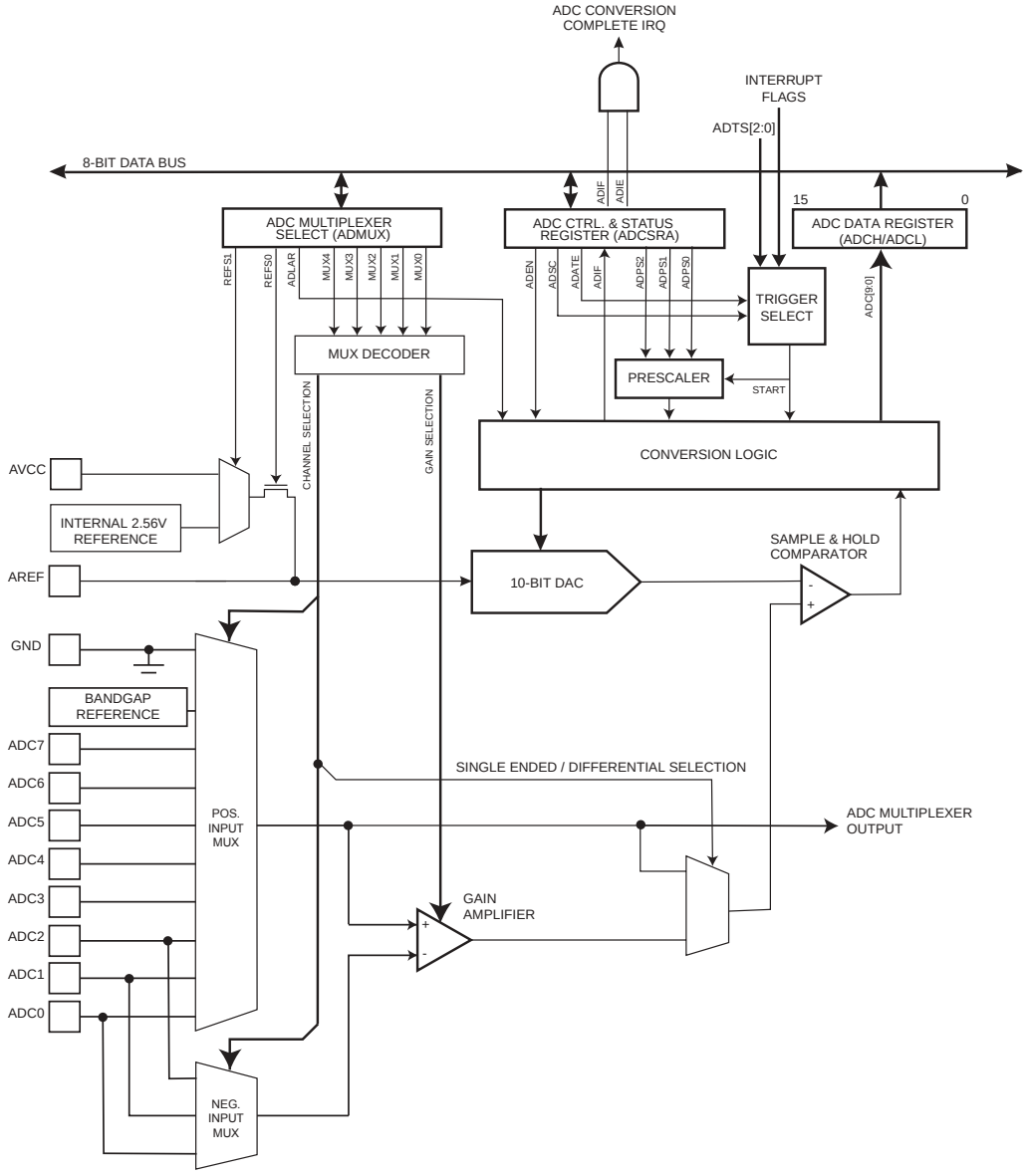
The device also supports 16 differential voltage input combinations. Two of the differential inputs (ADC1, ADC0 and ADC3, ADC2) are equipped with a programmable gain stage, providing amplification steps of 0 dB (1x), 20 dB (10x), or 46 dB (200x) on the differential input voltage before the A/D conversion. Seven differential analog input channels share a common negative terminal (ADC1), while any other ADC input can be selected as the positive input terminal. If 1x or 10x gain is used, 8-bit resolution can be expected. If 200x gain is used, 7-bit resolution can be expected.

The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 98.

The ADC has a separate analog supply voltage pin, AVCC. AVCC must not differ more than ±0.3 V from V_{CC}. See the paragraph “ADC Noise Canceler” on page 207 on how to connect this pin.

Internal reference voltages of nominally 2.56V or AVCC are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.

Figure 98. Analog to Digital Converter Block Schematic



Operation

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AVCC or an internal 2.56V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

The analog input channel and differential gain are selected by writing to the MUX bits in ADMUX. Any of the ADC input pins, as well as GND and a fixed bandgap voltage reference, can be selected as single ended inputs to the ADC. A selection of ADC input pins can be selected as positive and negative inputs to the differential gain amplifier.

If differential channels are selected, the differential gain stage amplifies the voltage difference between the selected input channel pair by the selected gain factor. This

amplified value then becomes the analog input to the ADC. If single ended channels are used, the gain amplifier is bypassed altogether.

The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering power saving sleep modes.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX.

If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion. Once ADCL is read, ADC access to Data Registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.

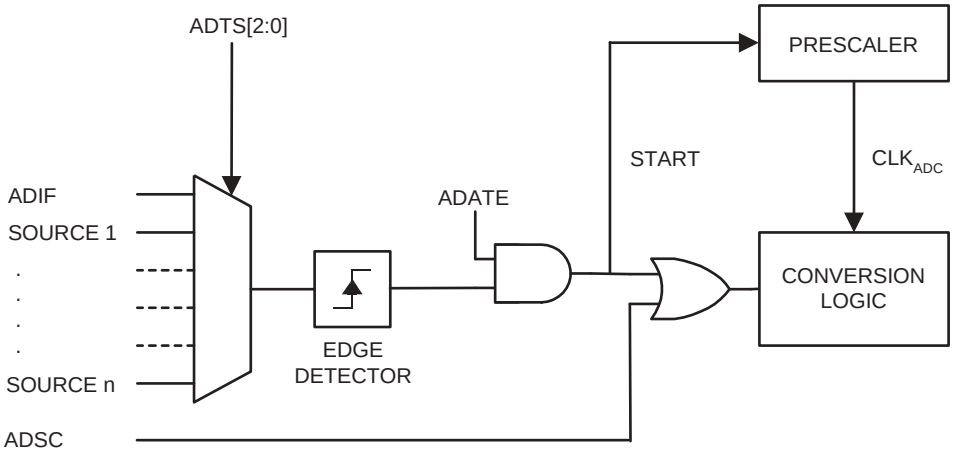
The ADC has its own interrupt which can be triggered when a conversion completes. When ADC access to the Data Registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

Starting a Conversion

A single conversion is started by writing a logical one to the ADC Start Conversion bit, ADSC. This bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

Alternatively, a conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit, ADATE in ADCSRA. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in SFIOR (see description of the ADTS bits for a list of the trigger sources). When a positive edge occurs on the selected trigger signal, the ADC prescaler is reset and a conversion is started. This provides a method of starting conversions at fixed intervals. If the trigger signal still is set when the conversion completes, a new conversion will not be started. If another positive edge occurs on the trigger signal during conversion, the edge will be ignored. Note that an Interrupt Flag will be set even if the specific interrupt is disabled or the global interrupt enable bit in SREG is cleared. A conversion can thus be triggered without causing an interrupt. However, the Interrupt Flag must be cleared in order to trigger a new conversion at the next interrupt event.

Figure 99. ADC Auto Trigger Logic

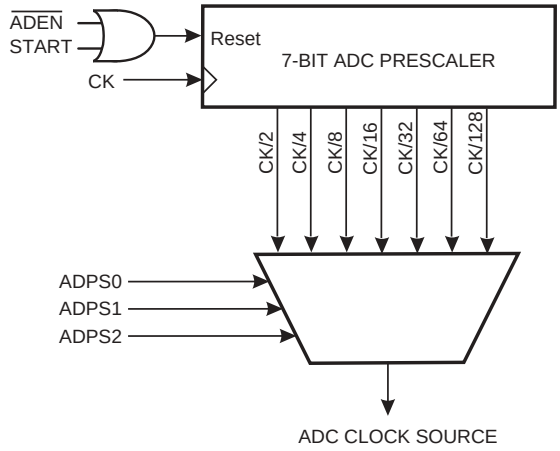


Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in Free Running mode, constantly sampling and updating the ADC Data Register. The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA. In this mode the ADC will perform successive conversions independently of whether the ADC Interrupt Flag, ADIF is cleared or not.

If Auto Triggering is enabled, single conversions can be started by writing ADSC in ADCSRA to one. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as one during a conversion, independently of how the conversion was started.

Prescaling and Conversion Timing

Figure 100. ADC Prescaler



By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate.

The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100 kHz. The prescaling is set by the ADPS bits in ADCSRA. The prescaler starts counting from the moment the ADC is switched on by

setting the ADEN bit in ADCSRA. The prescaler keeps running for as long as the ADEN bit is set, and is continuously reset when ADEN is low.

When initiating a single ended conversion by setting the ADSC bit in ADCSRA, the conversion starts at the following rising edge of the ADC clock cycle. See “Differential Gain Channels” on page 205 for details on differential conversion timing.

A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry.

The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of a first conversion. When a conversion is complete, the result is written to the ADC Data Registers, and ADIF is set. In single conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

When Auto Triggering is used, the prescaler is reset when the trigger event occurs. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place 2 ADC clock cycles after the rising edge on the trigger source signal. Three additional CPU clock cycles are used for synchronization logic.

When using Differential mode, along with Auto Trigging from a source other than the ADC Conversion Complete, each conversion will require 25 ADC clocks. This is because the ADC must be disabled and re-enabled after every conversion.

In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADSC remains high. For a summary of conversion times, see Table 81.

Figure 101. ADC Timing Diagram, First Conversion (Single Conversion Mode)

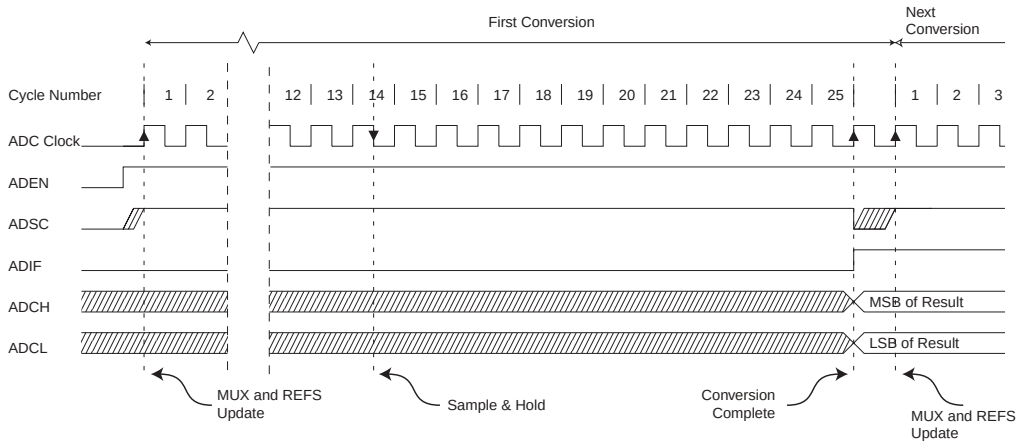


Figure 102. ADC Timing Diagram, Single Conversion

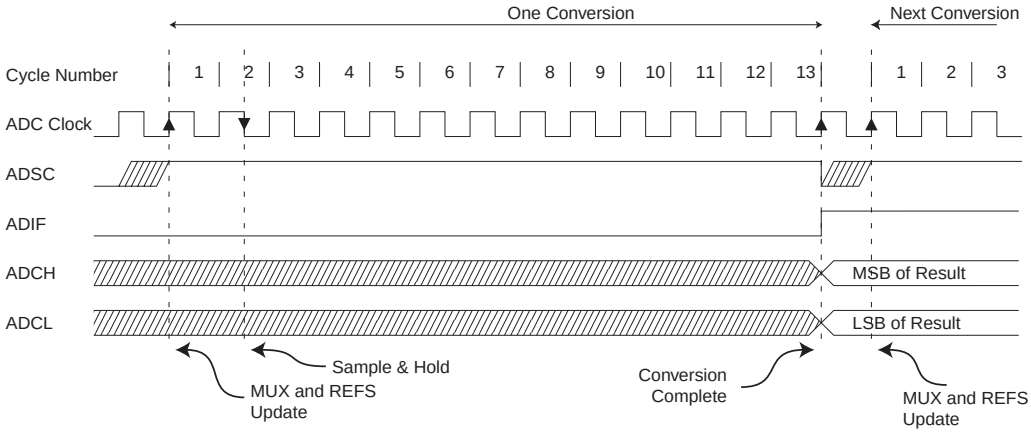


Figure 103. ADC Timing Diagram, Auto Triggered Conversion

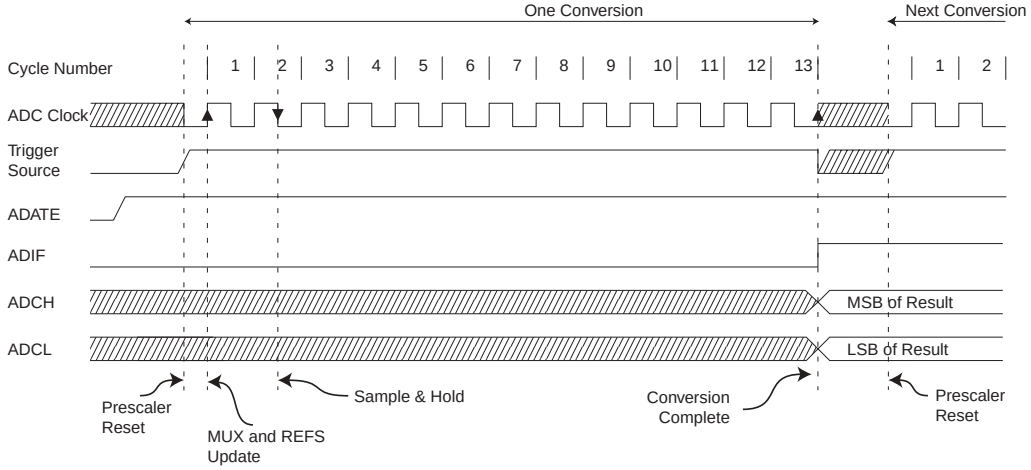


Figure 104. ADC Timing Diagram, Free Running Conversion

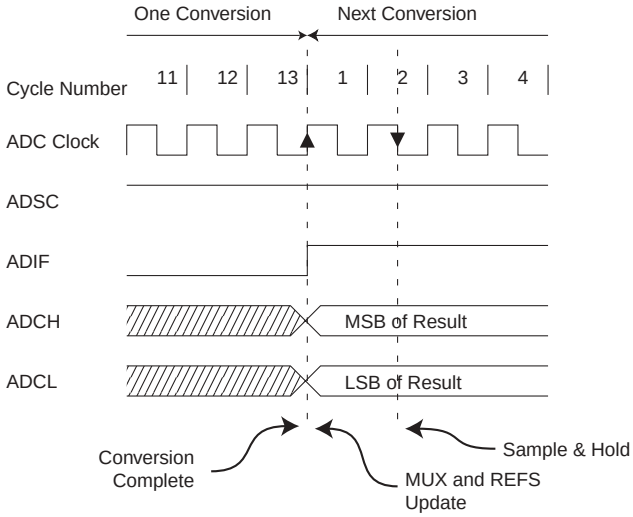


Table 81. ADC Conversion Time

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	14.5	25
Normal conversions, single ended	1.5	13
Auto Triggered conversions	2	13.5
Normal conversions, differential	1.5/2.5	13/14

Differential Gain Channels

When using differential gain channels, certain aspects of the conversion need to be taken into consideration.

Differential conversions are synchronized to the internal clock CK_{ADC2} equal to half the ADC clock. This synchronization is done automatically by the ADC interface in such a way that the sample-and-hold occurs at a specific phase of CK_{ADC2}. A conversion initiated by the user (i.e., all single conversions, and the first free running conversion) when CK_{ADC2} is low will take the same amount of time as a single ended conversion (13 ADC clock cycles from the next prescaled clock cycle). A conversion initiated by the user when CK_{ADC2} is high will take 14 ADC clock cycles due to the synchronization mechanism. In Free Running mode, a new conversion is initiated immediately after the previous conversion completes, and since CK_{ADC2} is high at this time, all automatically started (i.e., all but the first) free running conversions will take 14 ADC clock cycles.

The gain stage is optimized for a bandwidth of 4 kHz at all gain settings. Higher frequencies may be subjected to non-linear amplification. An external low-pass filter should be used if the input signal contains higher frequency components than the gain stage bandwidth. Note that the ADC clock frequency is independent of the gain stage bandwidth limitation. For example, the ADC clock period may be 6 µs, allowing a channel to be sampled at 12 kSPS, regardless of the bandwidth of this channel.

If differential gain channels are used and conversions are started by Auto Triggering, the ADC must be switched off between conversions. When Auto Triggering is used, the ADC prescaler is reset before the conversion is started. Since the gain stage is dependent of a stable ADC clock prior to the conversion, this conversion will not be valid. By disabling and then re-enabling the ADC between each conversion (writing ADEN in ADCSRA to “0” then to “1”), only extended conversions are performed. The result from the extended conversions will be valid. See “Prescaling and Conversion Timing” on page 202 for timing details.

Changing Channel or Reference Selection

The MUXn and REFS1:0 bits in the ADMUX Register are single buffered through a temporary register to which the CPU has random access. This ensures that the channels and reference selection only takes place at a safe point during the conversion. The channel and reference selection is continuously updated until a conversion is started. Once the conversion starts, the channel and reference selection is locked to ensure a sufficient sampling time for the ADC. Continuous updating resumes in the last ADC clock cycle before the conversion completes (ADIF in ADCSRA is set). Note that the conversion starts on the following rising ADC clock edge after ADSC is written. The user is thus advised not to write new channel or reference selection values to ADMUX until one ADC clock cycle after ADSC is written.

If Auto Triggering is used, the exact time of the triggering event can be indeterministic. Special care must be taken when updating the ADMUX Register, in order to control which conversion will be affected by the new settings.

If both ADATE and ADEN is written to one, an interrupt event can occur at any time. If the ADMUX Register is changed in this period, the user cannot tell if the next conversion is based on the old or the new settings. ADMUX can be safely updated in the following ways:

1. When ADATE or ADEN is cleared.
2. During conversion, minimum one ADC clock cycle after the trigger event.
3. After a conversion, before the Interrupt Flag used as trigger source is cleared.

When updating ADMUX in one of these conditions, the new settings will affect the next ADC conversion.

Special care should be taken when changing differential channels. Once a differential channel has been selected, the gain stage may take as much as 125 µs to stabilize to the new value. Thus conversions should not be started within the first 125 µs after selecting a new differential channel. Alternatively, conversion results obtained within this period should be discarded.

The same settling time should be observed for the first differential conversion after changing ADC reference (by changing the REFS1:0 bits in ADMUX).

ADC Input Channels

When changing channel selections, the user should observe the following guidelines to ensure that the correct channel is selected:

In Single Conversion mode, always select the channel before starting the conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the conversion to complete before changing the channel selection.

In Free Running mode, always select the channel before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the first conversion to complete, and then change the channel selection. Since the next conversion has already started automatically, the next result will reflect the previous channel selection. Subsequent conversions will reflect the new channel selection.

When switching to a differential gain channel, the first conversion result may have a poor accuracy due to the required settling time for the automatic offset cancellation circuitry. The user should preferably disregard the first conversion result.

ADC Voltage Reference

The reference voltage for the ADC (V_{REF}) indicates the conversion range for the ADC. Single ended channels that exceed V_{REF} will result in codes close to 0x3FF. V_{REF} can be selected as either AVCC, internal 2.56V reference, or external AREF pin.

AVCC is connected to the ADC through a passive switch. The internal 2.56V reference is generated from the internal bandgap reference (V_{BG}) through an internal amplifier. In either case, the external AREF pin is directly connected to the ADC, and the reference voltage can be made more immune to noise by connecting a capacitor between the AREF pin and ground. V_{REF} can also be measured at the AREF pin with a high impedant voltmeter. Note that V_{REF} is a high impedant source, and only a capacitive load should be connected in a system.

If the user has a fixed voltage source connected to the AREF pin, the user may not use the other reference voltage options in the application, as they will be shorted to the

external voltage. If no external voltage is applied to the AREF pin, the user may switch between AVCC and 2.56V as reference selection. The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result.

If differential channels are used, the selected reference should not be closer to AVCC than indicated in Table 122 on page 291.

ADC Noise Canceler

The ADC features a noise canceler that enables conversion during sleep mode to reduce noise induced from the CPU core and other I/O peripherals. The noise canceler can be used with ADC Noise Reduction and Idle mode. To make use of this feature, the following procedure should be used:

1. Make sure that the ADC is enabled and is not busy converting. Single Conversion Mode must be selected and the ADC conversion complete interrupt must be enabled.
2. Enter ADC Noise Reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.
3. If no other interrupts occur before the ADC conversion completes, the ADC interrupt will wake up the CPU and execute the ADC Conversion Complete interrupt routine. If another interrupt wakes up the CPU before the ADC conversion is complete, that interrupt will be executed, and an ADC Conversion Complete interrupt request will be generated when the ADC conversion completes. The CPU will remain in active mode until a new sleep command is executed.

Note that the ADC will not be automatically turned off when entering other sleep modes than Idle mode and ADC Noise Reduction mode. The user is advised to write zero to ADEN before entering such sleep modes to avoid excessive power consumption. If the ADC is enabled in such sleep modes and the user wants to perform differential conversions, the user is advised to switch the ADC off and on after waking up from sleep to prompt an extended conversion to get a valid result.

Analog Input Circuitry

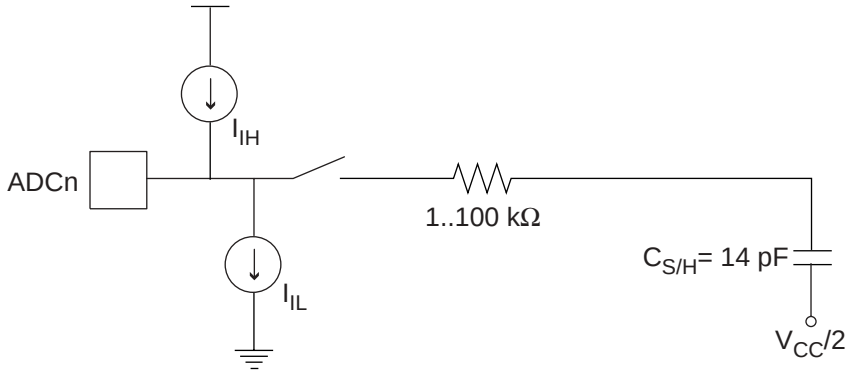
The Analog Input Circuitry for single ended channels is illustrated in Figure 105. An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin, regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10 kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, with can vary widely. The user is recommended to only use low impedant sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.

If differential gain channels are used, the input circuitry looks somewhat different, although source impedances of a few hundred kΩ or less is recommended.

Signal components higher than the Nyquist frequency ($f_{ADC}/2$) should not be present for either kind of channels, to avoid distortion from unpredictable signal convolution. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.

Figure 105. Analog Input Circuitry

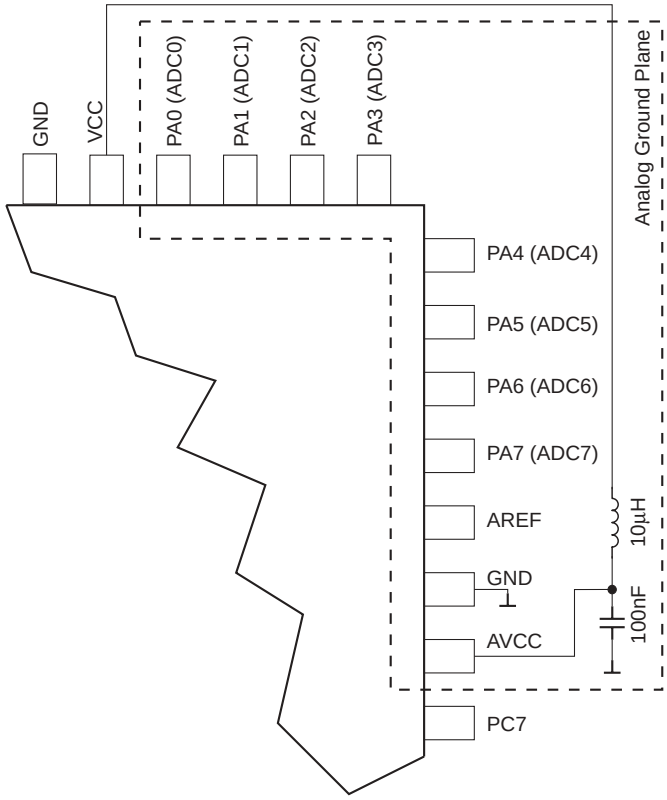


Analog Noise Canceling Techniques

Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. If conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

1. Keep analog signal paths as short as possible. Make sure analog tracks run over the analog ground plane, and keep them well away from high-speed switching digital tracks.
2. The AVCC pin on the device should be connected to the digital V_{CC} supply voltage via an LC network as shown in Figure 106.
3. Use the ADC noise canceler function to reduce induced noise from the CPU.
4. If any ADC port pins are used as digital outputs, it is essential that these do not switch while a conversion is in progress.

Figure 106. ADC Power Connections



Offset Compensation Schemes

The gain stage has a built-in offset cancellation circuitry that nulls the offset of differential measurements as much as possible. The remaining offset in the analog path can be measured directly by selecting the same channel for both differential inputs. This offset residue can be then subtracted in software from the measurement results. Using this kind of software based offset correction, offset on any channel can be reduced below one LSB.

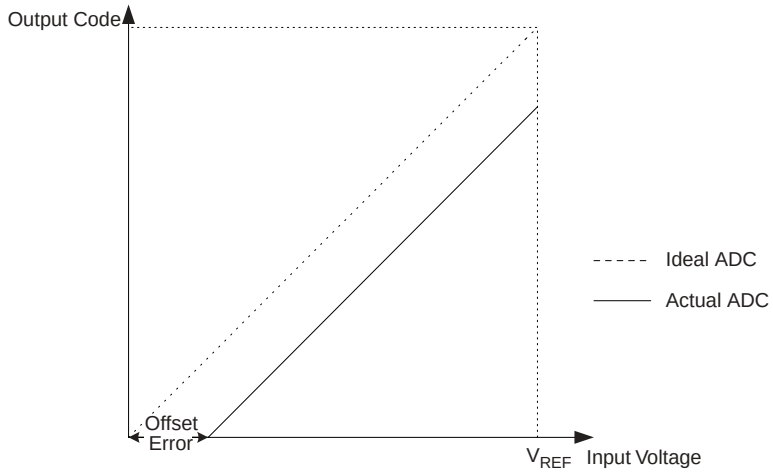
ADC Accuracy Definitions

An n-bit single-ended ADC converts a voltage linearly between GND and V_{REF} in 2^n steps (LSBs). The lowest code is read as 0, and the highest code is read as 2^n-1.

Several parameters describe the deviation from the ideal behavior:

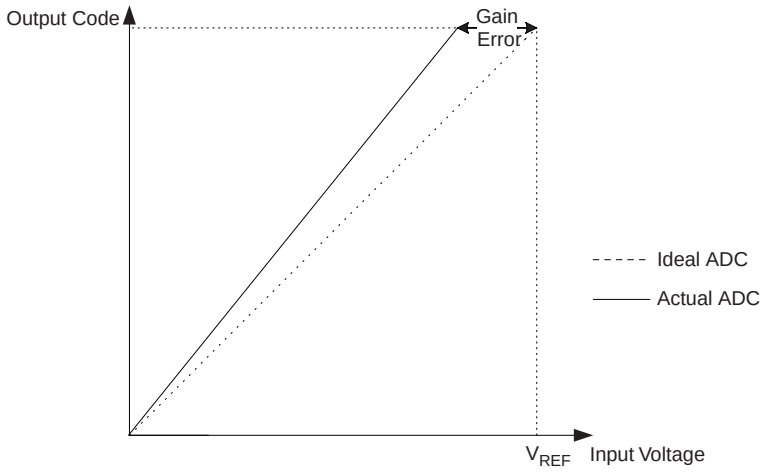
- Offset: The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

Figure 107. Offset Error



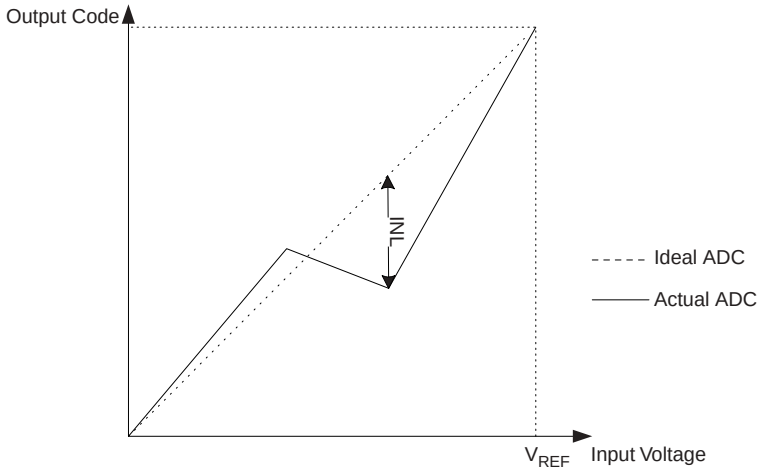
- Gain Error: After adjusting for offset, the Gain Error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB

Figure 108. Gain Error



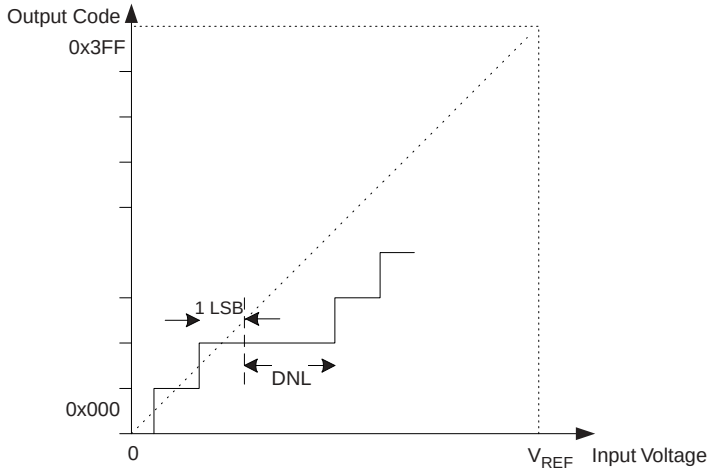
- Integral Non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

Figure 109. Integral Non-linearity (INL)



- Differential Non-linearity (DNL): The maximum deviation of the actual code width (the interval between two adjacent transitions) from the ideal code width (1 LSB). Ideal value: 0 LSB.

Figure 110. Differential Non-linearity (DNL)



- Quantization Error: Due to the quantization of the input voltage into a finite number of codes, a range of input voltages (1 LSB wide) will code to the same value. Always ±0.5 LSB.
- Absolute Accuracy: The maximum deviation of an actual (unadjusted) transition compared to an ideal transition for any code. This is the compound effect of Offset, Gain Error, Differential Error, Non-linearity, and Quantization Error. Ideal value: ±0.5 LSB.

ADC Conversion Result

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

where V_{IN} is the voltage on the selected input pin and V_{REF} the selected voltage reference (see Table 83 on page 212 and Table 84 on page 213). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB.

If differential channels are used, the result is

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

where V_{POS} is the voltage on the positive input pin, V_{NEG} the voltage on the negative input pin, GAIN the selected gain factor, and V_{REF} the selected voltage reference. The result is presented in two's complement form, from 0x200 (-512d) through 0x1FF (+511d). Note that if the user wants to perform a quick polarity check of the results, it is sufficient to read the MSB of the result (ADC9 in ADCH). If this bit is one, the result is negative, and if this bit is zero, the result is positive. Figure 111 shows the decoding of the differential input range.

Table 82 shows the resulting output codes if the differential input channel pair (ADCn - ADCm) is selected with a gain of GAIN and a reference voltage of V_{REF}.

Figure 111. Differential Measurement Range

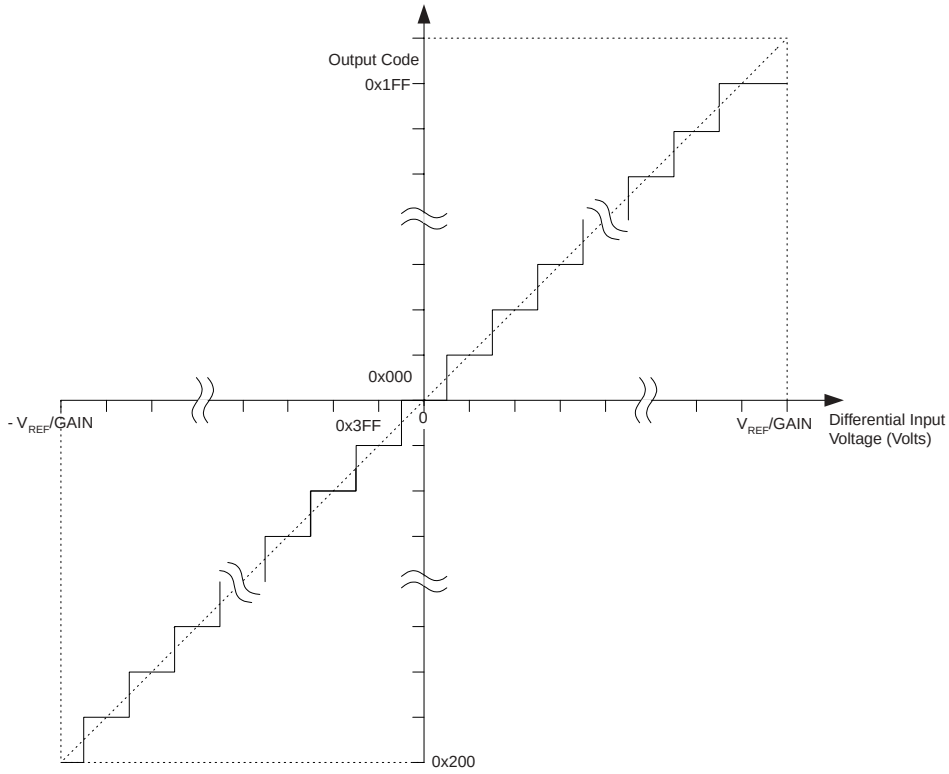


Table 82. Correlation between Input Voltage and Output Codes

V_{ADCn}	Read code	Corresponding Decimal Value
V_{ADCm} + V_{REF}/GAIN	0x1FF	511
V_{ADCm} + 0.999 V_{REF}/GAIN	0x1FF	511
V_{ADCm} + 0.998 V_{REF}/GAIN	0x1FE	510
...	...	...
V_{ADCm} + 0.001 V_{REF}/GAIN	0x001	1
V_{ADCm}	0x000	0
V_{ADCm} - 0.001 V_{REF}/GAIN	0x3FF	-1
...	...	...
V_{ADCm} - 0.999 V_{REF}/GAIN	0x201	-511
V_{ADCm} - V_{REF}/GAIN	0x200	-512

Example:

ADMUX = 0xED (ADC3 - ADC2, 10x gain, 2.56V reference, left adjusted result)

Voltage on ADC3 is 300 mV, voltage on ADC2 is 500 mV.

ADCR = 512 * 10 * (300 - 500) / 2560 = -400 = 0x270

ADCL will thus read 0x00, and ADCH will read 0x9C. Writing zero to ADLAR right adjusts the result: ADCL = 0x70, ADCH = 0x02.

ADC Multiplexer Selection Register – ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 83. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

Table 83. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

- **Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately,

regardless of any ongoing conversions. For a complete description of this bit, see “The ADC Data Register – ADCL and ADCH” on page 215.

- **Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the ADC. These bits also select the gain for the differential channels. See Table 84 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

Table 84. Input Channel and Gain Selections

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010(1)		ADC0	ADC0	200x
01011(1)		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110(1)		ADC2	ADC2	200x
01111(1)		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x

Table 84. Input Channel and Gain Selections (Continued)

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
11101		ADC5	ADC2	1x
11110	1.22 V (V_{BG})	N/A		
11111	0 V (GND)			

Note: 1. The differential input channels are not tested for devices in PDIP Package. This feature is only guaranteed to work for devices in TQFP and MLF Packages

ADC Control and Status Register A – ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running Mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in SFIOR.

- **Bit 4 – ADIF: ADC Interrupt Flag**

This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

- **Bit 3 – ADIE: ADC Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

- **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**

These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

Table 85. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

The ADC Data Register – ADCL and ADCH

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

When an ADC conversion is complete, the result is found in these two registers. If differential channels are used, the result is presented in two's complement form.

When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH.

The ADLAR bit in ADMUX, and the MUXn bits in ADMUX affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

- **ADC9:0: ADC Conversion Result**

These bits represent the result from the conversion, as detailed in “ADC Conversion Result” on page 211.

Special FunctionIO Register – SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:5 – ADTS2:0: ADC Auto Trigger Source**

If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS2:0 settings will have no effect. A conversion will be triggered by the rising edge of the selected Interrupt Flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to Free Running mode (ADTS[2:0]=0) will not cause a trigger event, even if the ADC Interrupt Flag is set.

Table 86. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

- **Bit 4 – Res: Reserved Bit**

This bit is reserved for future use in the ATmega32. For ensuring compability with future devices, this bit must be written zero when SFIOR is written.

JTAG Interface and On-chip Debug System

Features

- **JTAG (IEEE std. 1149.1 Compliant) Interface**
- **Boundary-scan Capabilities According to the IEEE std. 1149.1 (JTAG) Standard**
- **Debugger Access to:**
  - **All Internal Peripheral Units**
  - **Internal and External RAM**
  - **The Internal Register File**
  - **Program Counter**
  - **EEPROM and Flash Memories**
  - **Extensive On-chip Debug Support for Break Conditions, Including**
  - **AVR *Break* Instruction**
  - **Break on Change of Program Memory Flow**
  - **Single Step Break**
  - **Program Memory Breakpoints on Single Address or Address Range**
  - **Data Memory Breakpoints on Single Address or Address Range**
- **Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface**
- **On-chip Debugging Supported by AVR Studio®**

Overview

The AVR IEEE std. 1149.1 compliant JTAG interface can be used for

- Testing PCBs by using the JTAG Boundary-scan capability
- Programming the non-volatile memories, Fuses and Lock bits
- On-chip Debugging

A brief description is given in the following sections. Detailed descriptions for Programming via the JTAG interface, and using the Boundary-scan Chain can be found in the sections “Programming via the JTAG Interface” on page 272 and “IEEE 1149.1 (JTAG) Boundary-scan” on page 223**,** respectively. The On-chip Debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only.

Figure 112 shows a block diagram of the JTAG interface and the On-chip Debug system. The TAP Controller is a state machine controlled by the TCK and TMS signals. The TAP Controller selects either the JTAG Instruction Register or one of several Data Registers as the scan chain (Shift Register) between the TDI input and TDO output. The Instruction Register holds JTAG instructions controlling the behavior of a Data Register.

The ID-Register, Bypass Register, and the Boundary-scan Chain are the Data Registers used for board-level testing. The JTAG Programming Interface (actually consisting of several physical and virtual Data Registers) is used for JTAG Serial Programming via the JTAG interface. The Internal Scan Chain and Break Point Scan Chain are used for On-chip Debugging only.

Test Access Port – TAP

The JTAG interface is accessed through four of the AVR’s pins. In JTAG terminology, these pins constitute the Test Access Port – TAP. These pins are:

- TMS: Test Mode Select. This pin is used for navigating through the TAP-controller state machine.
- TCK: Test Clock. JTAG operation is synchronous to TCK.
- TDI: Test Data In. Serial input data to be shifted in to the Instruction Register or Data Register (Scan Chains).
- TDO: Test Data Out. Serial output data from Instruction Register or Data Register.

The IEEE std. 1149.1 also specifies an optional TAP signal; TRST – Test ReSeT – which is not provided.

When the JTAGEN fuse is unprogrammed, these four TAP pins are normal port pins and the TAP controller is in reset. When programmed and the JTD bit in MCUCSR is cleared, the TAP input signals are internally pulled high and the JTAG is enabled for Boundary-scan and programming. In this case, the TAP output pin (TDO) is left floating in states where the JTAG TAP controller is not shifting data, and must therefore be connected to a pull-up resistor or other hardware having pull-ups (for instance the TDI-input of the next device in the scan chain). The device is shipped with this fuse programmed.

For the On-chip Debug system, in addition to the JTAG interface pins, the $\overline{\text{RESET}}$ pin is monitored by the debugger to be able to detect external reset sources. The debuggerbta can also pull the $\overline{\text{RESET}}$ pin low to reset the whole system, assuming only open collectors on the reset line are used in the application.

Figure 112. Block Diagram

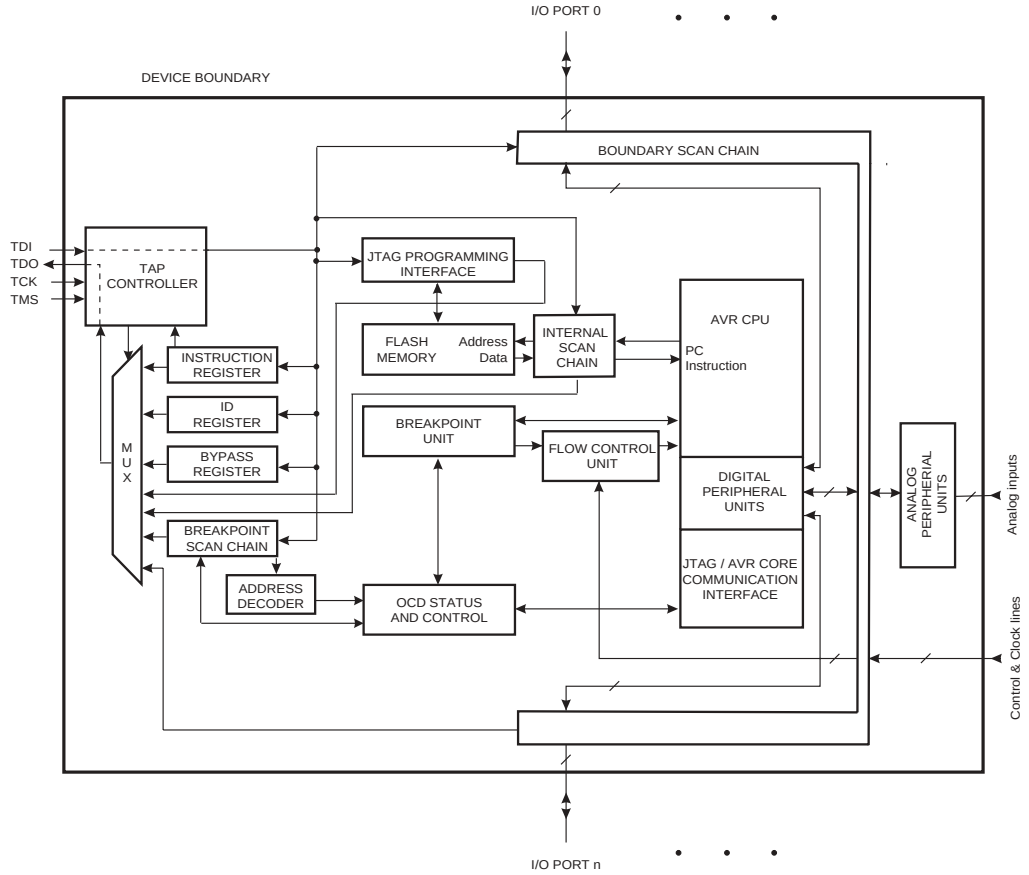
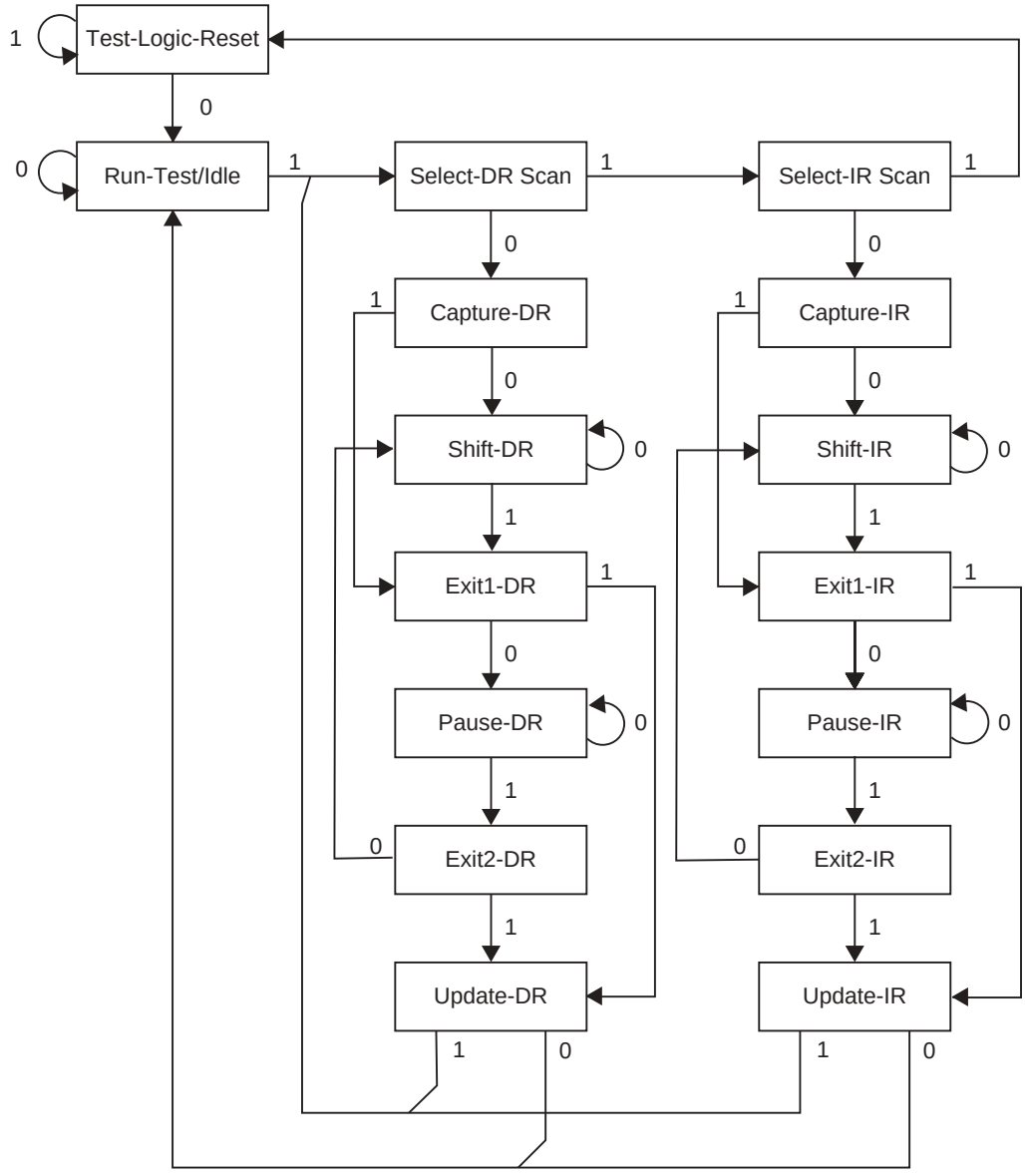


Figure 113. TAP Controller State Diagram



TAP Controller

The TAP controller is a 16-state finite state machine that controls the operation of the Boundary-scan circuitry, JTAG programming circuitry, or On-chip Debug system. The state transitions depicted in Figure 113 depend on the signal present on TMS (shown adjacent to each state transition) at the time of the rising edge at TCK. The initial state after a Power-On Reset is Test-Logic-Reset.

As a definition in this document, the LSB is shifted in and out first for all Shift Registers.

Assuming Run-Test/Idle is the present state, a typical scenario for using the JTAG interface is:

- At the TMS input, apply the sequence 1, 1, 0, 0 at the rising edges of TCK to enter the Shift Instruction Register – Shift-IR state. While in this state, shift the four bits of the JTAG instructions into the JTAG Instruction Register from the TDI input at the rising edge of TCK. The TMS input must be held low during input of the 3 LSBs in order to remain in the Shift-IR state. The MSB of the instruction is shifted in when

this state is left by setting TMS high. While the instruction is shifted in from the TDI pin, the captured IR-state 0x01 is shifted out on the TDO pin. The JTAG Instruction selects a particular Data Register as path between TDI and TDO and controls the circuitry surrounding the selected Data Register.

- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. The instruction is latched onto the parallel output from the Shift Register path in the Update-IR state. The Exit-IR, Pause-IR, and Exit2-IR states are only used for navigating the state machine.
- At the TMS input, apply the sequence 1, 0, 0 at the rising edges of TCK to enter the Shift Data Register – Shift-DR state. While in this state, upload the selected Data Register (selected by the present JTAG instruction in the JTAG Instruction Register) from the TDI input at the rising edge of TCK. In order to remain in the Shift-DR state, the TMS input must be held low during input of all bits except the MSB. The MSB of the data is shifted in when this state is left by setting TMS high. While the Data Register is shifted in from the TDI pin, the parallel inputs to the Data Register captured in the Capture-DR state is shifted out on the TDO pin.
- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. If the selected Data Register has a latched parallel-output, the latching takes place in the Update-DR state. The Exit-DR, Pause-DR, and Exit2-DR states are only used for navigating the state machine.

As shown in the state diagram, the Run-Test/Idle state need not be entered between selecting JTAG instruction and using Data Registers, and some JTAG instructions may select certain functions to be performed in the Run-Test/Idle, making it unsuitable as an Idle state.

Note: Independent of the initial state of the TAP Controller, the Test-Logic-Reset state can always be entered by holding TMS high for five TCK clock periods.

For detailed information on the JTAG specification, refer to the literature listed in “Bibliography” on page 222.

Using the Boundary-scan Chain

A complete description of the Boundary-scan capabilities are given in the section “IEEE 1149.1 (JTAG) Boundary-scan” on page 223.

Using the On-chip Debug System

As shown in Figure 112, the hardware support for On-chip Debugging consists mainly of:

- A scan chain on the interface between the internal AVR CPU and the internal peripheral units
- Break Point unit
- Communication interface between the CPU and JTAG system

All read or modify/write operations needed for implementing the Debugger are done by applying AVR instructions via the internal AVR CPU Scan Chain. The CPU sends the result to an I/O memory mapped location which is part of the communication interface between the CPU and the JTAG system.

The Break Point Unit implements Break on Change of Program Flow, Single Step Break, 2 Program Memory Break Points, and 2 combined Break Points. Together, the 4 Break Points can be configured as either:

- 4 single Program Memory Break Points
- 3 Single Program Memory Break Point + 1 single Data Memory Break Point
- 2 single Program Memory Break Points + 2 single Data Memory Break Points

- 2 single Program Memory Break Points + 1 Program Memory Break Point with mask (“range Break Point”)
- 2 single Program Memory Break Points + 1 Data Memory Break Point with mask (“range Break Point”)

A debugger, like the AVR Studio, may however use one or more of these resources for its internal purpose, leaving less flexibility to the end-user.

A list of the On-chip Debug specific JTAG instructions is given in “On-chip Debug Specific JTAG Instructions” on page 221.

The JTAGEN Fuse must be programmed to enable the JTAG Test Access Port. In addition, the OCDEN Fuse must be programmed and no Lock bits must be set for the On-chip Debug system to work. As a security feature, the On-chip Debug system is disabled when *any* Lock bits are set. Otherwise, the On-chip Debug system would have provided a back-door into a secured device.

The AVR JTAG ICE from Atmel is a powerful development tool for On-chip Debugging of all AVR 8-bit RISC Microcontrollers with IEEE 1149.1 compliant JTAG interface. The JTAG ICE and the AVR Studio user interface give the user complete control of the internal resources of the microcontroller, helping to reduce development time by making debugging easier. The JTAG ICE performs real-time emulation of the micrcontroller while it is running in a target system.

Please refer to the Support Tools section on the AVR pages on www.atmel.com for a full description of the AVR JTEG ICE. AVR Studio can be downloaded free from Software section on the same web site.

All necessary execution commands are available in AVR Studio, both on source level and on disassembly level. The user can execute the program, single step through the code either by tracing into or stepping over functions, step out of functions, place the cursor on a statement and execute until the statement is reached, stop the execution, and reset the execution target. In addition, the user can have an unlimited number of code breakpoints (using the BREAK instruction) and up to two data memory breakpoints, alternatively combined as a mask (range) Break Point.

On-chip Debug Specific JTAG Instructions

The On-chip Debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only. Instruction opcodes are listed for reference.

PRIVATE0; $8 Private JTAG instruction for accessing On-chip Debug system.

PRIVATE1; $9 Private JTAG instruction for accessing On-chip Debug system.

PRIVATE2; $A Private JTAG instruction for accessing On-chip Debug system.

PRIVATE3; $B Private JTAG instruction for accessing On-chip Debug system.

On-chip Debug Related Register in I/O Memory

On-chip Debug Register – OCDR

Bit	7	6	5	4	3	2	1	0	
	MSB/IDRD							LSB	OCDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The OCDR Register provides a communication channel from the running program in the microcontroller to the debugger. The CPU can transfer a byte to the debugger by writing to this location. At the same time, an Internal Flag; I/O Debug Register Dirty – IDRD – is set to indicate to the debugger that the register has been written. When the CPU reads the OCDR Register the 7 LSB will be from the OCDR Register, while the MSB is the IDRD bit. The debugger clears the IDRD bit when it has read the information.

In some AVR devices, this register is shared with a standard I/O location. In this case, the OCDR Register can only be accessed if the OCDEN Fuse is programmed, and the debugger enables access to the OCDR Register. In all other cases, the standard I/O location is accessed.

Refer to the debugger documentation for further information on how to use this register.

Using the JTAG Programming Capabilities

Programming of AVR parts via JTAG is performed via the 4-pin JTAG port, TCK, TMS, TDI and TDO. These are the only pins that need to be controlled/observed to perform JTAG programming (in addition to power pins). It is not required to apply 12V externally. The JTAGEN fuse must be programmed and the JTD bit in the MCUSR Register must be cleared to enable the JTAG Test Access Port.

The JTAG programming capability supports:

- Flash programming and verifying
- EEPROM programming and verifying
- Fuse programming and verifying
- Lock bit programming and verifying

The Lock bit security is exactly as in Parallel Programming mode. If the Lock bits LB1 or LB2 are programmed, the OCDEN Fuse cannot be programmed unless first doing a chip erase. This is a security feature that ensures no back-door exists for reading out the content of a secured device.

The details on programming through the JTAG interface and programming specific JTAG instructions are given in the section “Programming via the JTAG Interface” on page 272.

Bibliography

For more information about general Boundary-scan, the following literature can be consulted:

- IEEE: IEEE Std 1149.1-1990. IEEE Standard Test Access Port and Boundary-scan Architecture, IEEE, 1993
- Colin Maunder: The Board Designers Guide to Testable Logic Circuits, Addison-Wesley, 1992

IEEE 1149.1 (JTAG) Boundary-scan

Features

- **JTAG (IEEE std. 1149.1 Compliant) Interface**
- **Boundary-scan Capabilities According to the JTAG Standard**
- **Full Scan of all Port Functions as well as Analog Circuitry having Off-chip Connections**
- **Supports the Optional IDCODE Instruction**
- **Additional Public AVR_RESET Instruction to Reset the AVR**

System Overview

The Boundary-scan chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having Off-chip connections. At system level, all ICs having JTAG capabilities are connected serially by the TDI/TDO signals to form a long Shift Register. An external controller sets up the devices to drive values at their output pins, and observe the input values received from other devices. The controller compares the received data with the expected result. In this way, Boundary-scan provides a mechanism for testing interconnections and integrity of components on Printed Circuits Boards by using the four TAP signals only.

The four IEEE 1149.1 defined mandatory JTAG instructions IDCODE, BYPASS, SAMPLE/PRELOAD, and EXTEST, as well as the AVR specific public JTAG instruction AVR_RESET can be used for testing the Printed Circuit Board. Initial scanning of the Data Register path will show the ID-code of the device, since IDCODE is the default JTAG instruction. It may be desirable to have the AVR device in Reset during Test mode. If not reset, inputs to the device may be determined by the scan operations, and the internal software may be in an undetermined state when exiting the Test mode. Entering reset, the outputs of any Port Pin will instantly enter the high impedance state, making the HIGHZ instruction redundant. If needed, the BYPASS instruction can be issued to make the shortest possible scan chain through the device. The device can be set in the reset state either by pulling the external $\overline{\text{RESET}}$ pin low, or issuing the AVR_RESET instruction with appropriate setting of the Reset Data Register.

The EXTEST instruction is used for sampling external pins and loading output pins with data. The data from the output latch will be driven out on the pins as soon as the EXTEST instruction is loaded into the JTAG IR-Register. Therefore, the SAMPLE/PRELOAD should also be used for setting initial values to the scan ring, to avoid damaging the board when issuing the EXTEST instruction for the first time. SAMPLE/PRELOAD can also be used for taking a snapshot of the external pins during normal operation of the part.

The JTAGEN Fuse must be programmed and the JTD bit in the I/O Register MCUCSR must be cleared to enable the JTAG Test Access Port.

When using the JTAG interface for Boundary-scan, using a JTAG TCK clock frequency higher than the internal chip frequency is possible. The chip clock is not required to run.

Data Registers

The Data Registers relevant for Boundary-scan operations are:

- Bypass Register
- Device Identification Register
- Reset Register
- Boundary-scan Chain

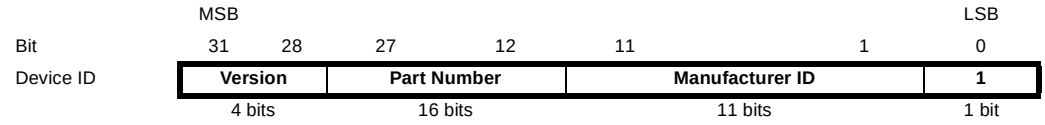
Bypass Register

The Bypass Register consists of a single Shift Register stage. When the Bypass Register is selected as path between TDI and TDO, the register is reset to 0 when leaving the Capture-DR controller state. The Bypass Register can be used to shorten the scan chain on a system when the other devices are to be tested.

Device Identification Register

Figure 114 shows the structure of the Device Identification Register.

Figure 114. The Format of the Device Identification Register



- **Version**

Version is a 4 bit number identifying the revision of the component. The relevant version number is shown in Table 87.

Table 87. JTAG Version Numbers

Version	JTAG Version Number (Hex)
ATmega32 revision A	0x0

- **Part Number**

The part number is a 16-bit code identifying the component. The JTAG Part Number for ATmega32 is listed in Table 88.

Table 88. AVR JTAG Part Number

Part Number	JTAG Part Number (Hex)
ATmega32	0x9502

- **Manufacturer ID**

The Manufacturer ID is a 11 bit code identifying the manufacturer. The JTAG manufacturer ID for ATMEL is listed in Table 89.

Table 89. Manufacturer ID

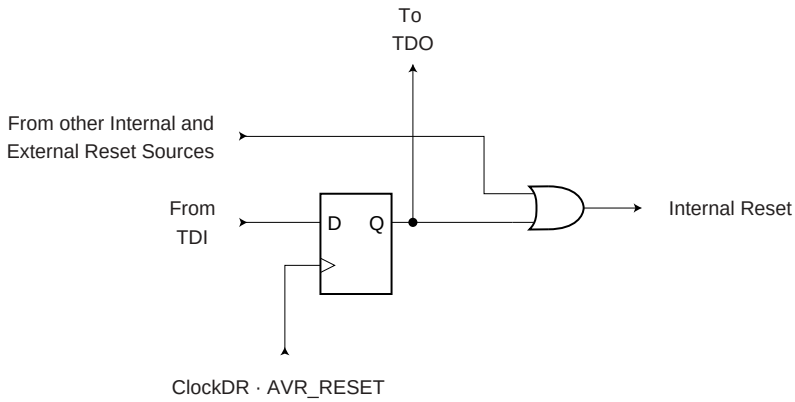
Manufacturer	JTAG Man. ID (Hex)
ATMEL	0x01F

Reset Register

The Reset Register is a Test Data Register used to reset the part. Since the AVR tri-states Port Pins when reset, the Reset Register can also replace the function of the unimplemented optional JTAG instruction HIGHZ.

A high value in the Reset Register corresponds to pulling the External Reset low. The part is reset as long as there is a high value present in the Reset Register. Depending on the Fuse settings for the clock options, the part will remain reset for a Reset Time-Out Period (refer to “Clock Sources” on page 23) after releasing the Reset Register. The output from this Data Register is not latched, so the reset will take place immediately, as shown in Figure 115.

Figure 115. Reset Register



Boundary-scan Chain

The Boundary-scan Chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having Off-chip connections.

See “Boundary-scan Chain” on page 227 for a complete description.

Boundary-scan Specific JTAG Instructions

The instruction register is 4-bit wide, supporting up to 16 instructions. Listed below are the JTAG instructions useful for Boundary-scan operation. Note that the optional HIGHZ instruction is not implemented, but all outputs with tri-state capability can be set in high-impedant state by using the AVR_RESET instruction, since the initial state for all port pins is tri-state.

As a definition in this datasheet, the LSB is shifted in and out first for all Shift Registers.

The OPCODE for each instruction is shown behind the instruction name in hex format. The text describes which Data Register is selected as path between TDI and TDO for each instruction.

EXTEST; $0

Mandatory JTAG instruction for selecting the Boundary-scan Chain as Data Register for testing circuitry external to the AVR package. For port-pins, Pull-up Disable, Output Control, Output Data, and Input Data are all accessible in the scan chain. For Analog circuits having Off-chip connections, the interface between the analog and the digital logic is in the scan chain. The contents of the latched outputs of the Boundary-scan chain is driven out as soon as the JTAG IR-register is loaded with the EXTEST instruction.

The active states are:

- Capture-DR: Data on the external pins are sampled into the Boundary-scan Chain.
- Shift-DR: The Internal Scan Chain is shifted by the TCK input.
- Update-DR: Data from the scan chain is applied to output pins.

IDCODE; $1

Optional JTAG instruction selecting the 32-bit ID-register as Data Register. The ID-register consists of a version number, a device number and the manufacturer code chosen by JEDEC. This is the default instruction after power-up.

The active states are:

- Capture-DR: Data in the IDCODE-register is sampled into the Boundary-scan Chain.
- Shift-DR: The IDCODE scan chain is shifted by the TCK input.

SAMPLE_PRELOAD; $2

Mandatory JTAG instruction for pre-loading the output latches and talking a snap-shot of the input/output pins without affecting the system operation. However, the output latches are not connected to the pins. The Boundary-scan Chain is selected as Data Register.

The active states are:

- Capture-DR: Data on the external pins are sampled into the Boundary-scan Chain.
- Shift-DR: The Boundary-scan Chain is shifted by the TCK input.
- Update-DR: Data from the Boundary-scan Chain is applied to the output latches. However, the output latches are not connected to the pins.

AVR_RESET; $C

The AVR specific public JTAG instruction for forcing the AVR device into the Reset mode or releasing the JTAG Reset source. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as Data Register. Note that the reset will be active as long as there is a logic 'one' in the Reset Chain. The output from this chain is not latched.

The active states are:

- Shift-DR: The Reset Register is shifted by the TCK input.

BYPASS; $F

Mandatory JTAG instruction selecting the Bypass Register for Data Register.

The active states are:

- Capture-DR: Loads a logic “0” into the Bypass Register.
- Shift-DR: The Bypass Register cell between TDI and TDO is shifted.

Boundary-scan Related Register in I/O Memory

MCU Control and Status Register – MCUCSR

The MCU Control and Status Register contains control bits for general MCU functions, and provides information on which reset source caused an MCU Reset.

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	**–**	**JTRF**	**WDRF**	**BORF**	**EXTRF**	**PORF**	**MCUCSR**
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

- **Bit 7 – JTD: JTAG Interface Disable**

When this bit is zero, the JTAG interface is enabled if the JTAGEN Fuse is programmed. If this bit is one, the JTAG interface is disabled. In order to avoid unintentional disabling or enabling of the JTAG interface, a timed sequence must be followed when changing this bit: The application software must write this bit to the desired value twice within four cycles to change its value.

If the JTAG interface is left unconnected to other JTAG circuitry, the JTD bit should be set to one. The reason for this is to avoid static current at the TDO pin in the JTAG interface.

- **Bit 4 – JTRF: JTAG Reset Flag**

This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

Boundary-scan Chain

The Boundary-scan chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having Off-chip connection.

Scanning the Digital Port Pins

Figure 116 shows the Boundary-scan Cell for a bi-directional port pin with pull-up function. The cell consists of a standard Boundary-scan cell for the Pull-up Enable – PUExn – function, and a bi-directional pin cell that combines the three signals Output Control – OCxn, Output Data – ODxn, and Input Data – IDxn, into only a two-stage Shift Register. The port and pin indexes are not used in the following description.

The Boundary-scan logic is not included in the figures in the datasheet. Figure 117 shows a simple digital Port Pin as described in the section “I/O Ports” on page 47. The Boundary-scan details from Figure 116 replaces the dashed box in Figure 117.

When no alternate port function is present, the Input Data – ID – corresponds to the PINxn Register value (but ID has no synchronizer), Output Data corresponds to the PORT Register, Output Control corresponds to the Data Direction – DD Register, and the Pull-up Enable – PUExn – corresponds to logic expression $\overline{PUD} \cdot \overline{DDxn} \cdot PORTxn$.

Digital alternate port functions are connected outside the dotted box in Figure 117 to make the scan chain read the actual pin value. For Analog function, there is a direct connection from the external pin to the analog circuit, and a scan chain is inserted on the interface between the digital logic and the analog circuitry.

Figure 116. Boundary-scan Cell for Bidirectional Port Pin with Pull-up Function.

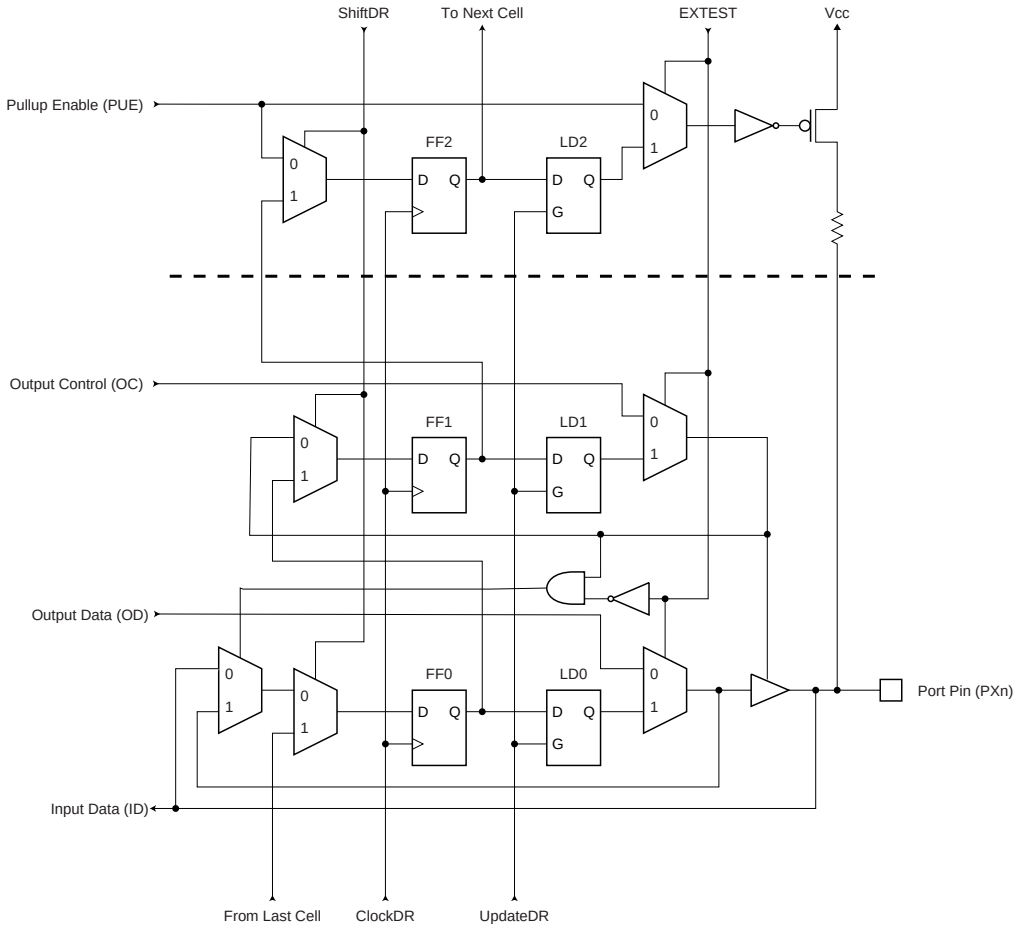
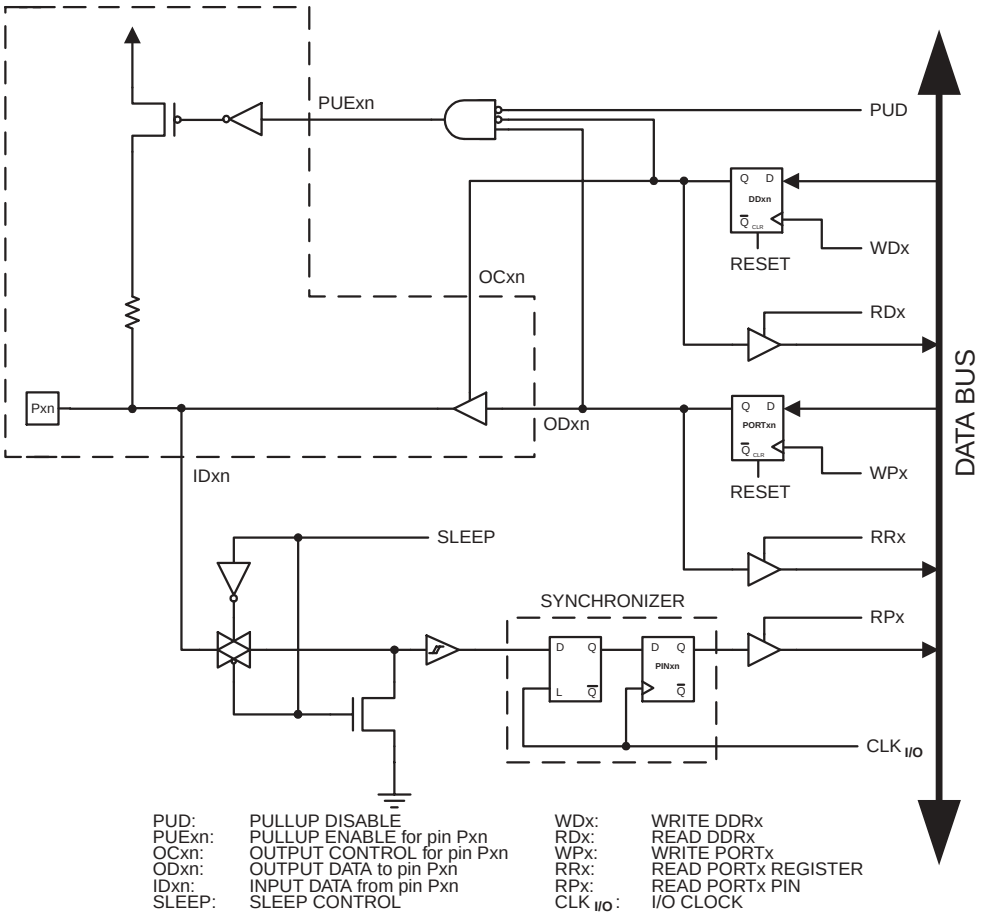


Figure 117. General Port Pin Schematic Diagram(1)



Note: 1. See Boundary-scan descriptin for details.

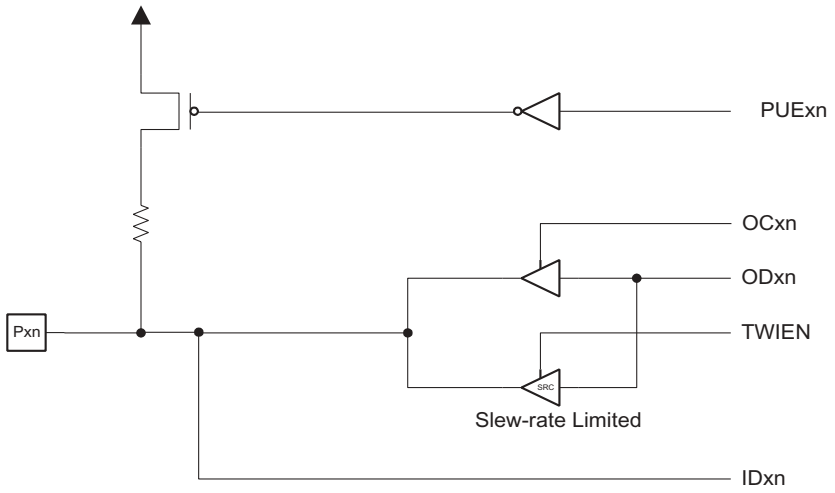
Boundary-scan and the Two-wire Interface

The 2 Two-wire Interface pins SCL and SDA have one additional control signal in the scan-chain; Two-wire Interface Enable – TWIEN. As shown in Figure 118, the TWIEN signal enables a tri-state buffer with slew-rate control in parallel with the ordinary digital port pins. A general scan cell as shown in Figure 122 is attached to the TWIEN signal.

Notes: 1. A separate scan chain for the 50 ns spike filter on the input is not provided. The ordinary scan support for digital port pins suffice for connectivity tests. The only reason for having TWIEN in the scan path, is to be able to disconnect the slew-rate control buffer when doing boundary-scan.

2. Make sure the OC and TWIEN signals are not asserted simultaneously, as this will lead to drive contention.

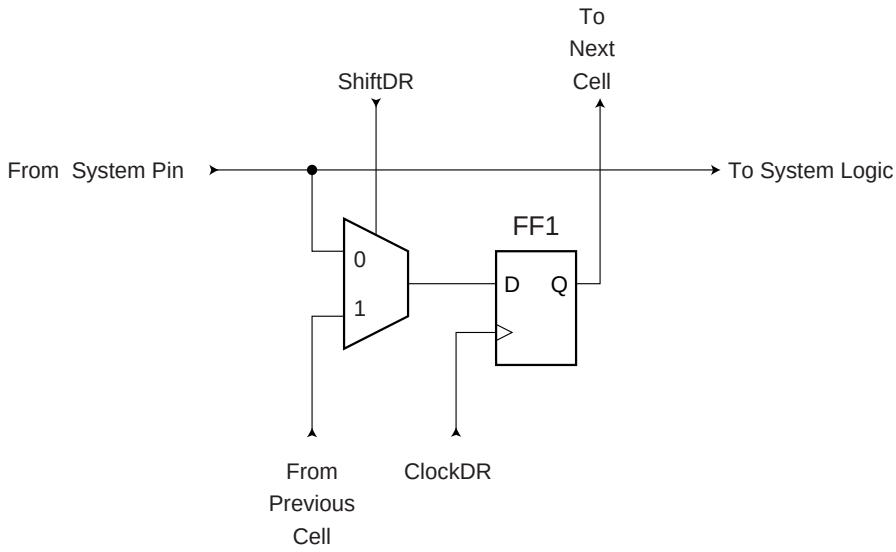
Figure 118. Additional Scan Signal for the Two-wire Interface



Scanning the RESET Pin

The RESET pin accepts 5V active low logic for standard reset operation, and 12V active high logic for High Voltage Parallel Programming. An observe-only cell as shown in Figure 119 is inserted both for the 5V reset signal; RSTT, and the 12V reset signal; RSTHV.

Figure 119. Observe-only Cell



Scanning the Clock Pins

The AVR devices have many clock options selectable by fuses. These are: Internal RC Oscillator, External RC, External Clock, (High Frequency) Crystal Oscillator, Low Frequency Crystal Oscillator, and Ceramic Resonator.

Figure 120 shows how each Oscillator with external connection is supported in the scan chain. The Enable signal is supported with a general boundary-scan cell, while the Oscillator/Clock output is attached to an observe-only cell. In addition to the main clock, the Timer Oscillator is scanned in the same way. The output from the internal RC Oscillator is not scanned, as this Oscillator does not have external connections.

Figure 120. Boundary-scan Cells for Oscillators and Clock Options

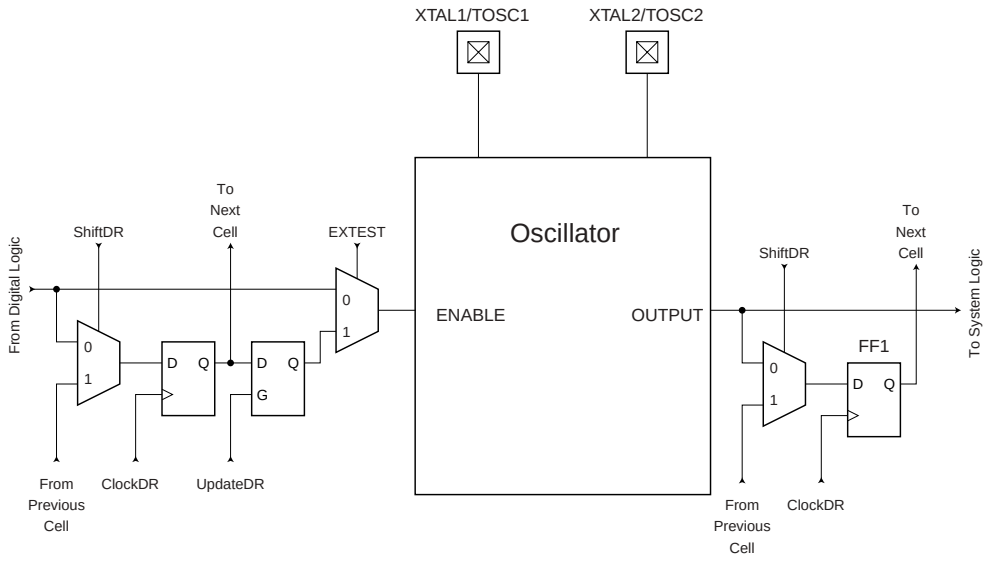


Table 90 summaries the scan registers for the external clock pin XTAL1, Oscillators with XTAL1/XTAL2 connections as well as 32 kHz Timer Oscillator.

Table 90. Scan Signals for the Oscillators(1)(2)(3)

Enable Signal	Scanned Clock Line	Clock Option	Scanned Clock Line when not Used
EXTCLKEN	EXTCLK (XTAL1)	External Clock	0
OSCON	OSCCK	External Crystal External Ceramic Resonator	0
RCOSCEN	RCCK	External RC	1
OSC32EN	OSC32CK	Low Freq. External Crystal	0
TOSKON	TOSCK	32 kHz Timer Oscillator	0

Notes:
1. Do not enable more than one clock source as main clock at a time.
2. Scanning an Oscillator output gives unpredictable results as there is a frequency drift between the Internal Oscillator and the JTAG TCK clock. If possible, scanning an external clock is preferred.
3. The clock configuration is programmed by fuses. As a fuse is not changed run-time, the clock configuration is considered fixed for a given application. The user is advised to scan the same clock option as to be used in the final system. The enable signals are supported in the scan chain because the system logic can disable clock options in sleep modes, thereby disconnecting the Oscillator pins from the scan path if not provided. The INTCAP fuses are not supported in the scan-chain, so the boundary scan chain can not make a XTAL Oscillator requiring internal capacitors to run unless the fuse is correctly programmed.

Scanning the Analog Comparator

The relevant Comparator signals regarding Boundary-scan are shown in Figure 121. The Boundary-scan cell from Figure 122 is attached to each of these signals. The signals are described in Table 91.

The Comparator need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

Figure 121. Analog Comparator

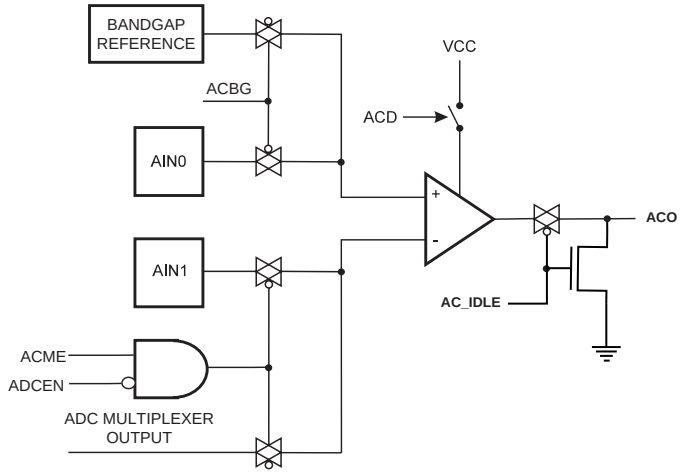


Figure 122. General Boundary-scan Cell used for Signals for Comparator and ADC

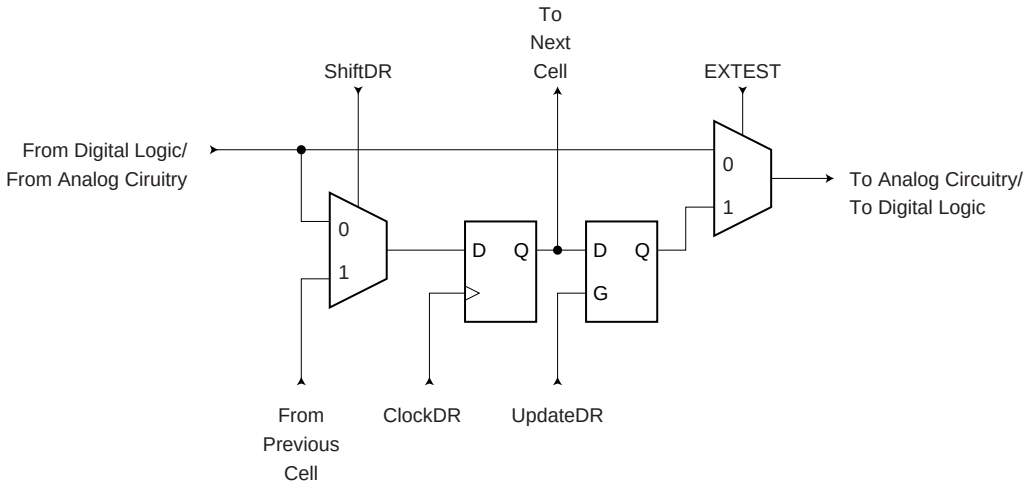


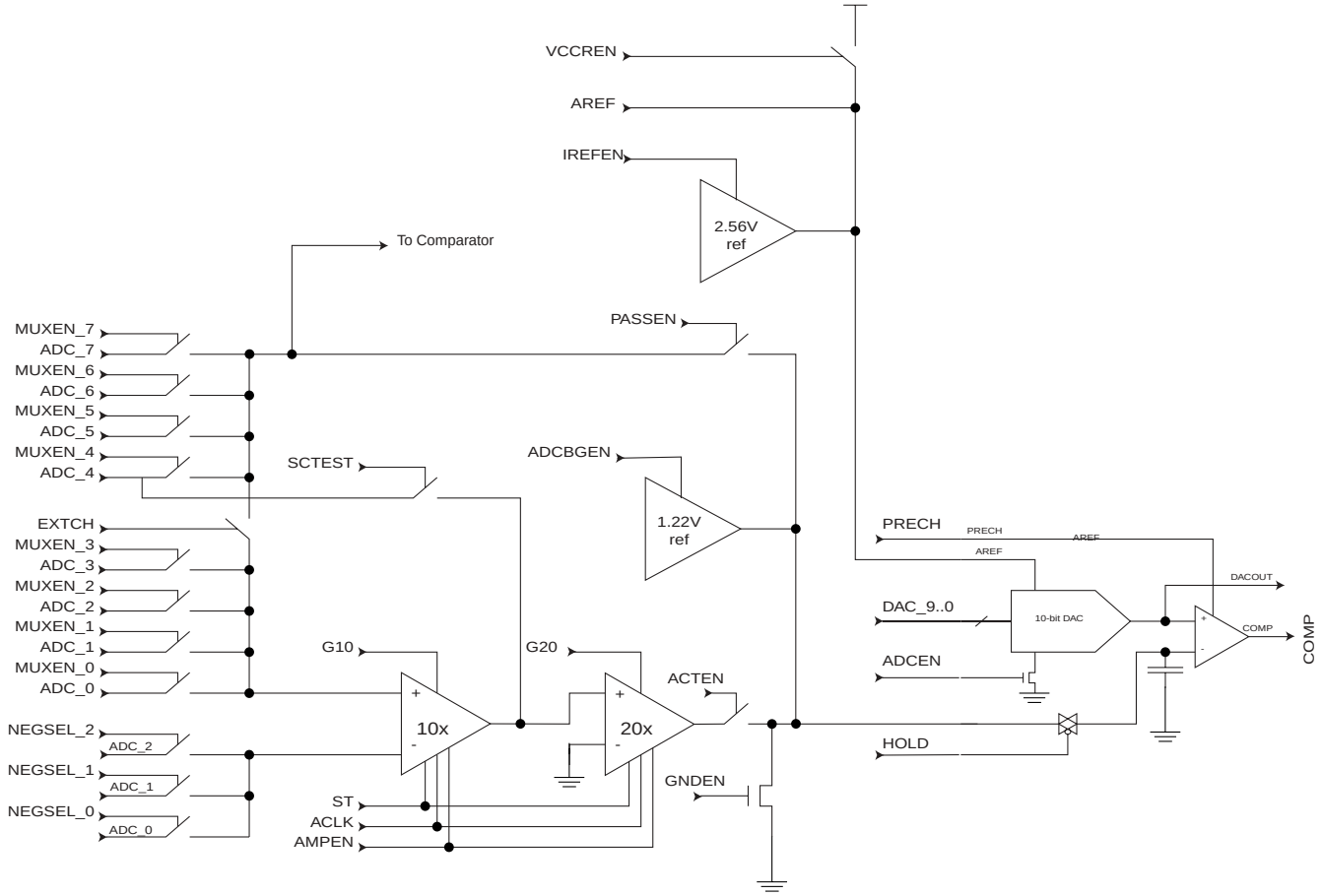
Table 91. Boundary-scan Signals for the Analog Comparator

Signal Name	Direction as Seen from the Comparator	Description	Recommended Input when Not in Use	Output Values when Recommended Inputs are Used
AC_IDLE	Input	Turns off Analog comparator when true	1	Depends upon µC code being executed
ACO	Output	Analog Comparator Output	Will become input to µC code being executed	0
ACME	Input	Uses output signal from ADC mux when true	0	Depends upon µC code being executed
ACBG	Input	Bandgap Reference enable	0	Depends upon µC code being executed

Scanning the ADC

Figure 123 shows a block diagram of the ADC with all relevant control and observe signals. The Boundary-scan cell from Figure 122 is attached to each of these signals. The ADC need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

Figure 123. Analog to Digital Converter



The signals are described briefly in Table 92.

Table 92. Boundary-scan Signals for the ADC

Signal Name	Direction as Seen from the ADC	Description	Recommended Input when Not in Use	Output Values when Recommended Inputs are used, and CPU is not Using the ADC
COMP	Output	Comparator Output	0	0
ACLK	Input	Clock signal to gain stages implemented as Switch-cap filters	0	0
ACTEN	Input	Enable path from gain stages to the comparator	0	0
ADCBGEN	Input	Enable Band-gap reference as negative input to comparator	0	0
ADCEN	Input	Power-on signal to the ADC	0	0
AMPEN	Input	Power-on signal to the gain stages	0	0
DAC_9	Input	Bit 9 of digital value to DAC	1	1
DAC_8	Input	Bit 8 of digital value to DAC	0	0
DAC_7	Input	Bit 7 of digital value to DAC	0	0
DAC_6	Input	Bit 6 of digital value to DAC	0	0
DAC_5	Input	Bit 5 of digital value to DAC	0	0
DAC_4	Input	Bit 4 of digital value to DAC	0	0
DAC_3	Input	Bit 3 of digital value to DAC	0	0
DAC_2	Input	Bit 2 of digital value to DAC	0	0
DAC_1	Input	Bit 1 of digital value to DAC	0	0
DAC_0	Input	Bit 0 of digital value to DAC	0	0
EXTCH	Input	Connect ADC channels 0 - 3 to by-pass path around gain stages	1	1
G10	Input	Enable 10x gain	0	0
G20	Input	Enable 20x gain	0	0
GNDEN	Input	Ground the negative input to comparator when true	0	0
HOLD	Input	Sample&Hold signal. Sample analog signal when low. Hold signal when high. If gain stages are used, this signal must go active when ACLK is high.	1	1
IREFEN	Input	Enables Band-gap reference as AREF signal to DAC	0	0
MUXEN_7	Input	Input Mux bit 7	0	0
MUXEN_6	Input	Input Mux bit 6	0	0
MUXEN_5	Input	Input Mux bit 5	0	0
MUXEN_4	Input	Input Mux bit 4	0	0
MUXEN_3	Input	Input Mux bit 3	0	0

Table 92. Boundary-scan Signals for the ADC (Continued)

Signal Name	Direction as Seen from the ADC	Description	Recommended Input when Not in Use	Output Values when Recommended Inputs are used, and CPU is not Using the ADC
MUXEN_2	Input	Input Mux bit 2	0	0
MUXEN_1	Input	Input Mux bit 1	0	0
MUXEN_0	Input	Input Mux bit 0	1	1
NEGSEL_2	Input	Input Mux for negative input for differential signal, bit 2	0	0
NEGSEL_1	Input	Input Mux for negative input for differential signal, bit 1	0	0
NEGSEL_0	Input	Input Mux for negative input for differential signal, bit 0	0	0
PASSEN	Input	Enable pass-gate of gain stages.	1	1
PRECH	Input	Precharge output latch of comparator. (Active low)	1	1
SCTEST	Input	Switch-cap TEST enable. Output from x10 gain stage send out to Port Pin having ADC_4	0	0
ST	Input	Output of gain stages will settle faster if this signal is high first two ACLK periods after AMPEN goes high.	0	0
VCCREN	Input	Selects Vcc as the ACC reference voltage.	0	0

Note: Incorrect setting of the switches in Figure 123 will make signal contention and may damage the part. There are several input choices to the S&H circuitry on the negative input of the output comparator in Figure 123. Make sure only one path is selected from either one ADC pin, Bandgap reference source, or Ground.

If the ADC is not to be used during scan, the recommended input values from Table 92 should be used. The user is recommended **not** to use the Differential Gain stages during scan. Switch-cap based gain stages require fast operation and accurate timing which is difficult to obtain when used in a scan chain. Details concerning operations of the differential gain stage is therefore not provided.

The AVR ADC is based on the analog circuitry shown in Figure 123 with a successive approximation algorithm implemented in the digital logic. When used in Boundary-scan, the problem is usually to ensure that an applied analog voltage is measured within some limits. This can easily be done without running a successive approximation algorithm: apply the lower limit on the digital DAC[9:0] lines, make sure the output from the comparator is low, then apply the upper limit on the digital DAC[9:0] lines, and verify the output from the comparator to be high.

The ADC need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

When using the ADC, remember the following:

- The Port Pin for the ADC channel in use must be configured to be an input with pull-up disabled to avoid signal contention.
- In Normal mode, a dummy conversion (consisting of 10 comparisons) is performed when enabling the ADC. The user is advised to wait at least 200 ns after enabling the ADC before controlling/observing any ADC signal, or perform a dummy conversion before using the first result.
- The DAC values must be stable at the midpoint value 0x200 when having the HOLD signal low (Sample mode).

As an example, consider the task of verifying a 1.5V ± 5% input signal at ADC channel 3 when the power supply is 5.0V and AREF is externally connected to V_{CC}.

The lower limit is: $\lceil 1024 \cdot 1.5V \cdot 0{,}95/5V \rceil = 291 = 0x123$
The upper limit is: $\lceil 1024 \cdot 1.5V \cdot 1.05/5V \rceil = 323 = 0x143$

The recommended values from Table 92 are used unless other values are given in the algorithm in Table 93. Only the DAC and Port Pin values of the Scan-chain are shown. The column “Actions” describes what JTAG instruction to be used before filling the Boundary-scan Register with the succeeding columns. The verification should be done on the data scanned out when scanning in the data on the same row in the table.

Table 93. Algorithm for Using the ADC

Step	Actions	ADCEN	DAC	MUXEN	HOLD	PRECH	PA3. Data	PA3. Control	PA3. Pullup_ Enable
1	SAMPLE _PRELO AD	1	0x200	0x08	1	1	0	0	0
2	EXTEST	1	0x200	0x08	0	1	0	0	0
3		1	0x200	0x08	1	1	0	0	0
4		1	0x123	0x08	1	1	0	0	0
5		1	0x123	0x08	1	0	0	0	0
6	Verify the COMP bit scanned out to be 0	1	0x200	0x08	1	1	0	0	0
7		1	0x200	0x08	0	1	0	0	0
8		1	0x200	0x08	1	1	0	0	0
9		1	0x143	0x08	1	1	0	0	0
10		1	0x143	0x08	1	0	0	0	0
11	Verify the COMP bit scanned out to be 1	1	0x200	0x08	1	1	0	0	0

Using this algorithm, the timing constraint on the HOLD signal constrains the TCK clock frequency. As the algorithm keeps HOLD high for five steps, the TCK clock frequency has to be at least five times the number of scan bits divided by the maximum hold time, $t_{hold,max}$.

ATmega32 Boundary-scan Order

Table 94 shows the scan order between TDI and TDO when the Boundary-scan chain is selected as data path. Bit 0 is the LSB; the first bit scanned in, and the first bit scanned out. The scan order follows the pin-out order as far as possible. Therefore, the bits of Port A is scanned in the opposite bit order of the other ports. Exceptions from the rules are the Scan chains for the analog circuits, which constitute the most significant bits of the scan chain regardless of which physical pin they are connected to. In Figure 116, PXn. Data corresponds to FF0, PXn. Control corresponds to FF1, and PXn. Pullup_enable corresponds to FF2. Bit 2, 3, 4, and 5 of Port C is not in the scan chain, since these pins constitute the TAP pins when the JTAG is enabled.

Table 94. ATmega32 Boundary-scan Order

Bit Number	Signal Name	Module
140	AC_IDLE	Comparator
139	ACO	
138	ACME	
137	ACBG	
136	COMP	ADC
135	PRIVATE_SIGNAL1[1]	
134	ACLK	
133	ACTEN	
132	PRIVATE_SIGNAL2[2]	
131	ADCBGEN	
130	ADCEN	
129	AMPEN	
128	DAC_9	
127	DAC_8	
126	DAC_7	
125	DAC_6	
124	DAC_5	
123	DAC_4	
122	DAC_3	
121	DAC_2	
120	DAC_1	
119	DAC_0	
118	EXTCH	
117	G10	
116	G20	
115	GNDEN	
114	HOLD	
113	IREFEN	

Table 94. ATmega32 Boundary-scan Order (Continued)

Bit Number	Signal Name	Module
112	MUXEN_7	ADC
111	MUXEN_6	
110	MUXEN_5	
109	MUXEN_4	
108	MUXEN_3	
107	MUXEN_2	
106	MUXEN_1	
105	MUXEN_0	
104	NEGSEL_2	
103	NEGSEL_1	
102	NEGSEL_0	
101	PASSEN	
100	PRECH	
99	SCTEST	
98	ST	
97	VCCREN	
96	PB0.Data	Port B
95	PB0.Control	
94	PB0.Pullup_Enable	
93	PB1.Data	
92	PB1.Control	
91	PB1.Pullup_Enable	
90	PB2.Data	
89	PB2.Control	
88	PB2.Pullup_Enable	
87	PB3.Data	
86	PB3.Control	
85	PB3.Pullup_Enable	
84	PB4.Data	
83	PB4.Control	
82	PB4.Pullup_Enable	

Table 94. ATmega32 Boundary-scan Order (Continued)

Bit Number	Signal Name	Module
81	PB5.Data	Port B
80	PB5.Control	
79	PB5.Pullup_Enable	
78	PB6.Data	
77	PB6.Control	
76	PB6.Pullup_Enable	
75	PB7.Data	
74	PB7.Control	
73	PB7.Pullup_Enable	
72	RSTT	Reset Logic (Observe-Only)
71	RSTHV	
70	EXTCLKEN	Enable signals for main clock/Oscillators
69	OSCON	
68	RCOSCEN	
67	OSC32EN	
66	EXTCLK (XTAL1)	Clock input and Oscillators for the main clock (Observe-Only)
65	OSCCK	
64	RCCK	
63	OSC32CK	
62	TWIEN	TWI
61	PD0.Data	Port D
60	PD0.Control	
59	PD0.Pullup_Enable	
58	PD1.Data	
57	PD1.Control	
56	PD1.Pullup_Enable	
55	PD2.Data	
54	PD2.Control	
53	PD2.Pullup_Enable	
52	PD3.Data	
51	PD3.Control	
50	PD3.Pullup_Enable	
49	PD4.Data	
48	PD4.Control	
47	PD4.Pullup_Enable	

Table 94. ATmega32 Boundary-scan Order (Continued)

Bit Number	Signal Name	Module
46	PD5.Data	Port D
45	PD5.Control	
44	PD5.Pullup_Enable	
43	PD6.Data	
42	PD6.Control	
41	PD6.Pullup_Enable	
40	PD7.Data	
39	PD7.Control	
38	PD7.Pullup_Enable	
37	PC0.Data	Port C
36	PC0.Control	
35	PC0.Pullup_Enable	
34	PC1.Data	
33	PC1.Control	
32	PC1.Pullup_Enable	
31	PC6.Data	
30	PC6.Control	
29	PC6.Pullup_Enable	
28	PC7.Data	
27	PC7.Control	
26	PC7.Pullup_Enable	
25	TOSC	32 kHz Timer Oscillator
24	TOSCON	
23	PA7.Data	Port A
22	PA7.Control	
21	PA7.Pullup_Enable	
20	PA6.Data	
19	PA6.Control	
18	PA6.Pullup_Enable	
17	PA5.Data	
16	PA5.Control	
15	PA5.Pullup_Enable	
14	PA4.Data	
13	PA4.Control	
12	PA4.Pullup_Enable	

Table 94. ATmega32 Boundary-scan Order (Continued)

Bit Number	Signal Name	Module
11	PA3.Data	Port A
10	PA3.Control	
9	PA3.Pullup_Enable	
8	PA2.Data	
7	PA2.Control	
6	PA2.Pullup_Enable	
5	PA1.Data	
4	PA1.Control	
3	PA1.Pullup_Enable	
2	PA0.Data	
1	PA0.Control	
0	PA0.Pullup_Enable	

Notes: 1. PRIVATE_SIGNAL1 should always be scanned in as zero.
2. PRIVATE_SIGNAL2 should always be scanned in as zero.

Boundary-scan Description Language Files

Boundary-scan Description Language (BSDL) files describe Boundary-scan capable devices in a standard format used by automated test-generation software. The order and function of bits in the Boundary-scan Data Register are included in this description. A BSDL file for ATmega32 is available.

Boot Loader Support – Read-While-Write Self-Programming

The Boot Loader Support provides a real Read-While-Write Self-Programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the Program memory. The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the Boot Loader memory is configurable with Fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

Features

- **Read-While-Write Self-Programming**
- **Flexible Boot Memory size**
- **High Security (Separate Boot Lock Bits for a Flexible Protection)**
- **Separate Fuse to Select Reset Vector**
- **Optimized Page(1) Size**
- **Code Efficient Algorithm**
- **Efficient Read-Modify-Write Support**

Note: 1. A page is a section in the flash consisting of several bytes (see Table 111 on page 258) used during programming. The page organization does not affect normal operation.

Application and Boot Loader Flash Sections

The Flash memory is organized in two main sections, the Application section and the Boot Loader section (see Figure 125). The size of the different sections is configured by the BOOTSZ Fuses as shown in Table 100 on page 253 and Figure 125. These two sections can have different level of protection since they have different sets of Lock bits.

Application Section

The Application section is the section of the Flash that is used for storing the application code. The protection level for the application section can be selected by the Application Boot Lock bits (Boot Lock bits 0), see Table 96 on page 245. The Application section can never store any Boot Loader code since the SPM instruction is disabled when executed from the Application section.

BLS – Boot Loader Section

While the Application section is used for storing the application code, the The Boot Loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire Flash, including the BLS itself. The protection level for the Boot Loader section can be selected by the Boot Loader Lock bits (Boot Lock bits 1), see Table 97 on page 245.

Read-While-Write and no Read-While-Write Flash Sections

Whether the CPU supports Read-While-Write or if the CPU is halted during a Boot Loader software update is dependent on which address that is being programmed. In addition to the two sections that are configurable by the BOOTSZ Fuses as described above, the Flash is also divided into two fixed sections, the Read-While-Write (RWW) section and the No Read-While-Write (NRWW) section. The limit between the RWW- and NRWW sections is given in Table 101 on page 253 and Figure 125 on page 244. The main difference between the two sections is:

- When erasing or writing a page located inside the RWW section, the NRWW section can be read during the operation.
- When erasing or writing a page located inside the NRWW section, the CPU is halted during the entire operation.

Note that the user software can never read any code that is located inside the RWW section during a Boot Loader software operation. The syntax “Read-While-Write section” refers to which section that is being programmed (erased or written), not which section that actually is being read during a Boot Loader software update.

RWW – Read-While-Write Section

If a Boot Loader software update is programming a page inside the RWW section, it is possible to read code from the Flash, but only code that is located in the NRWW section. During an on-going programming, the software must ensure that the RWW section never is being read. If the user software is trying to read code that is located inside the RWW section (i.e., by a call/jmp/lpm or an interrupt) during programming, the software might end up in an unknown state. To avoid this, the interrupts should either be disabled or moved to the Boot Loader section. The Boot Loader section is always located in the NRWW section. The RWW Section Busy bit (RWWSB) in the Store Program Memory Control Register (SPMCR) will be read as logical one as long as the RWW section is blocked for reading. After a programming is completed, the RWWSB must be cleared by software before reading code located in the RWW section. See “Store Program Memory Control Register – SPMCR” on page 246. for details on how to clear RWWSB.

NRWW – No Read-While-Write Section

The code located in the NRWW section can be read when the Boot Loader software is updating a page in the RWW section. When the Boot Loader code updates the NRWW section, the CPU is halted during the entire page erase or page write operation.

Table 95. Read-While-Write Features

Which Section does the Z-pointer Address during the Programming?	Which Section can be Read during Programming?	Is the CPU Halted?	Read-While-Write Supported?
RWW section	NRWW section	No	Yes
NRWW section	None	Yes	No

Figure 124. Read-While-Write vs. No Read-While-Write

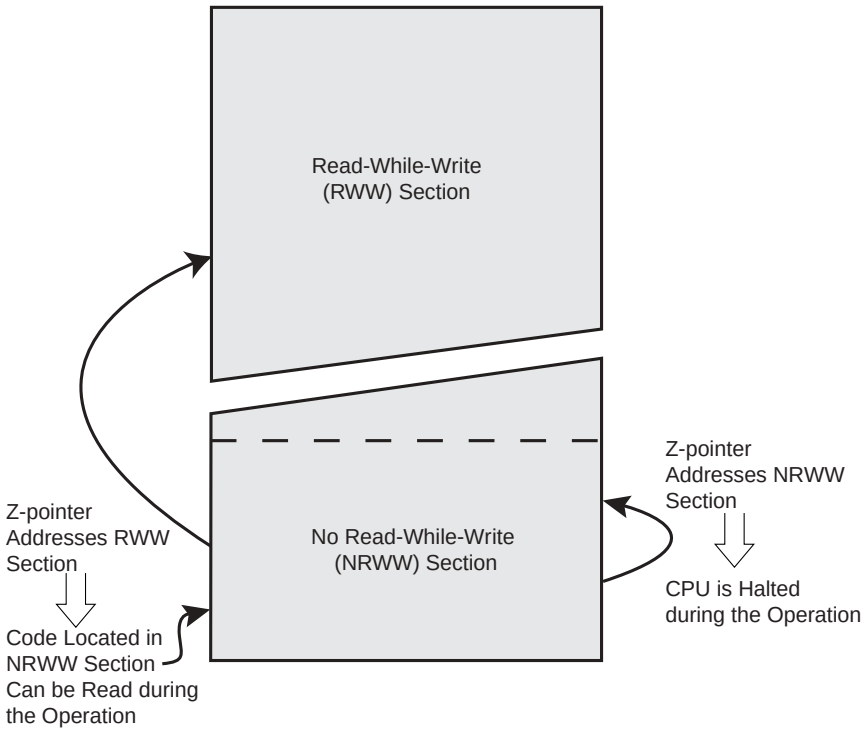
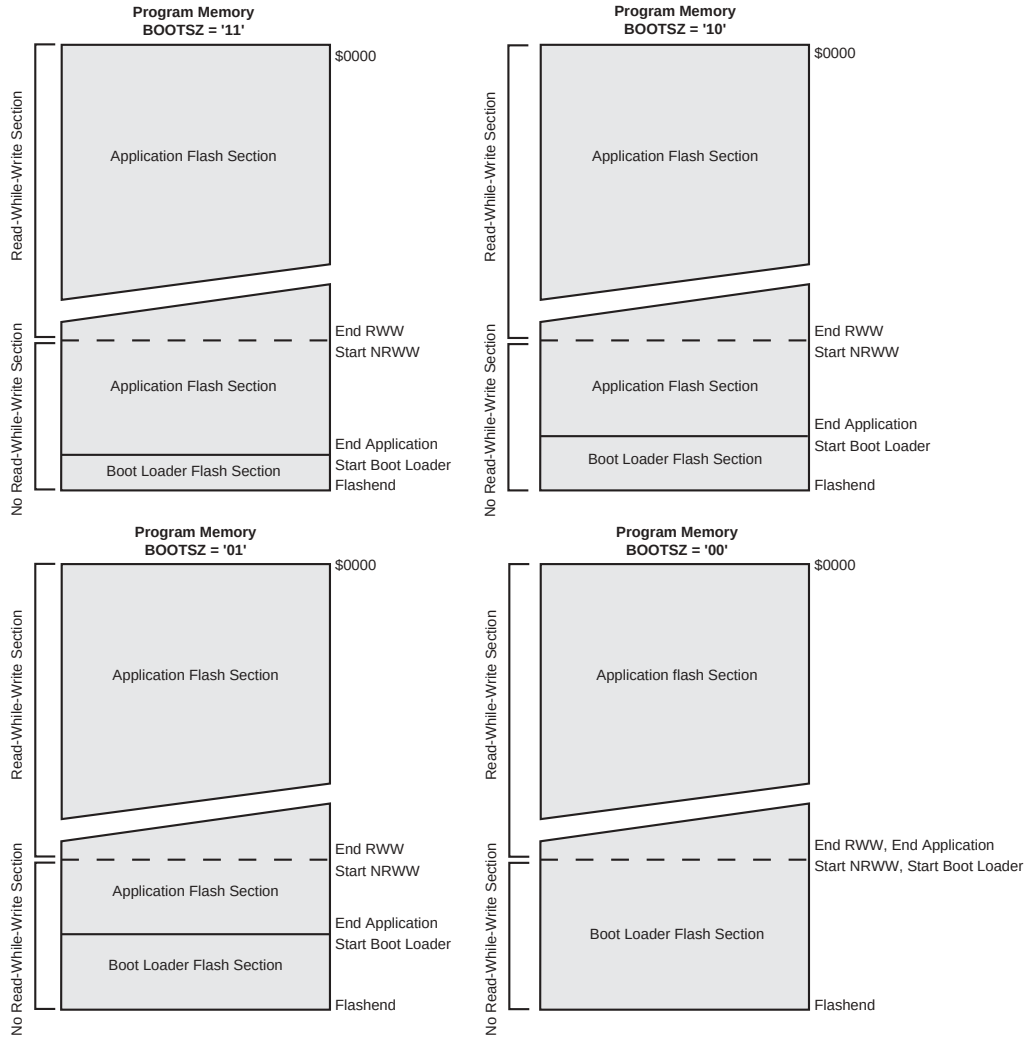


Figure 125. Memory Sections[1]



Note: 1. The parameters in the figure above are given in Table 100 on page 253.

Boot Loader Lock Bits

If no Boot Loader capability is needed, the entire Flash is available for application code. The Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

The user can select:

- To protect the entire Flash from a software update by the MCU
- To protect only the Boot Loader Flash section from a software update by the MCU
- To protect only the Application Flash section from a software update by the MCU
- Allow software update in the entire Flash

See Table 96 and Table 97 for further details. The Boot Lock bits can be set in software and in Serial or Parallel Programming mode, but they can be cleared by a Chip Erase command only. The general Write Lock (Lock Bit mode 2) does not control the programming of the Flash memory by SPM instruction. Similarly, the general Read/Write Lock (Lock Bit mode 3) does not control reading nor writing by LPM/SPM, if it is attempted.

Table 96. Boot Lock Bit0 Protection Modes (Application Section)[1]

BLB0 Mode	BLB02	BLB01	Protection
1	1	1	No restrictions for SPM or LPM accessing the Application section.
2	1	0	SPM is not allowed to write to the Application section.
3	0	0	SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If interrupt vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
4	0	1	LPM executing from the Boot Loader section is not allowed to read from the Application section. If interrupt vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.

Note: 1. “1” means unprogrammed, “0” means programmed

Table 97. Boot Lock Bit1 Protection Modes (Boot Loader Section)[1]

BLB1 mode	BLB12	BLB11	Protection
1	1	1	No restrictions for SPM or LPM accessing the Boot Loader section.
2	1	0	SPM is not allowed to write to the Boot Loader section.
3	0	0	SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If interrupt vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.
4	0	1	LPM executing from the Application section is not allowed to read from the Boot Loader section. If interrupt vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.

Note: 1. “1” means unprogrammed, “0” means programmed

Entering the Boot Loader Program

Entering the Boot Loader takes place by a jump or call from the application program. This may be initiated by a trigger such as a command received via USART, or SPI interface. Alternatively, the Boot Reset Fuse can be programmed so that the Reset Vector is pointing to the Boot Flash start address after a reset. In this case, the Boot Loader is started after a reset. After the application code is loaded, the program can start executing the application code. Note that the fuses cannot be changed by the MCU itself. This means that once the Boot Reset Fuse is programmed, the Reset Vector will always point to the Boot Loader Reset and the fuse can only be changed through the serial or parallel programming interface.

Table 98. Boot Reset Fuse[1]

BOOTRST	Reset Address
1	Reset Vector = Application reset (address $0000)
0	Reset Vector = Boot Loader reset (see Table 100 on page 253)

Note: 1. “1” means unprogrammed, “0” means programmed

Store Program Memory Control Register – SPMCR

The Store Program Memory Control Register contains the control bits needed to control the Boot Loader operations.

Bit	7	6	5	4	3	2	1	0	
	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCR
Read/Write	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPMIE: SPM Interrupt Enable**

When the SPMIE bit is written to one, and the I-bit in the Status Register is set (one), the SPM ready interrupt will be enabled. The SPM ready Interrupt will be executed as long as the SPMEN bit in the SPMCR Register is cleared.

- **Bit 6 – RWWSB: Read-While-Write Section Busy**

When a self-programming (Page Erase or Page Write) operation to the RWW section is initiated, the RWWSB will be set (one) by hardware. When the RWWSB bit is set, the RWW section cannot be accessed. The RWWSB bit will be cleared if the RWWSRE bit is written to one after a Self-Programming operation is completed. Alternatively the RWWSB bit will automatically be cleared if a page load operation is initiated.

- **Bit 5 – Res: Reserved Bit**

This bit is a reserved bit in the ATmega32 and always read as zero.

- **Bit 4 – RWWSRE: Read-While-Write Section Read Enable**

When programming (Page Erase or Page Write) to the RWW section, the RWW section is blocked for reading (the RWWSB will be set by hardware). To re-enable the RWW section, the user software must wait until the programming is completed (SPMEN will be cleared). Then, if the RWWSRE bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles re-enables the RWW section. The RWW section cannot be re-enabled while the Flash is busy with a page erase or a page write (SPMEN is set). If the RWWSRE bit is written while the Flash is being loaded, the Flash load operation will abort and the data loaded will be lost.

- **Bit 3 – BLBSET: Boot Lock Bit Set**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles sets Boot Lock bits, according to the data in R0. The data in R1 and the address in the Z-pointer are ignored. The BLBSET bit will automatically be cleared upon completion of the Lock bit set, or if no SPM instruction is executed within four clock cycles.

An LPM instruction within three cycles after BLBSET and SPMEN are set in the SPMCR Register, will read either the Lock bits or the Fuse bits (depending on Z0 in the Z-pointer) into the destination register. See “Reading the Fuse and Lock Bits from Software” on page 250 for details.

- **Bit 2 – PGWRT: Page Write**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles executes Page Write, with the data stored in the temporary buffer. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGWRT bit will auto-clear upon completion of a page write, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire page write operation if the NRWW section is addressed.

- **Bit 1 – PGERS: Page Erase**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles executes Page Erase. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGERS bit will auto-clear upon completion of a page erase, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire page write operation if the NRWW section is addressed.

- **Bit 0 – SPMEN: Store Program Memory Enable**

This bit enables the SPM instruction for the next four clock cycles. If written to one together with either RWWSRE, BLBSET, PGWRT' or PGERS, the following SPM instruction will have a special meaning, see description above. If only SPMEN is written, the following SPM instruction will store the value in R1:R0 in the temporary page buffer addressed by the Z-pointer. The LSB of the Z-pointer is ignored. The SPMEN bit will auto-clear upon completion of an SPM instruction, or if no SPM instruction is executed within four clock cycles. During page erase and page write, the SPMEN bit remains high until the operation is completed.

Writing any other combination than "10001", "01001", "00101", "00011" or "00001" in the lower five bits will have no effect.

Addressing the Flash during Self-Programming

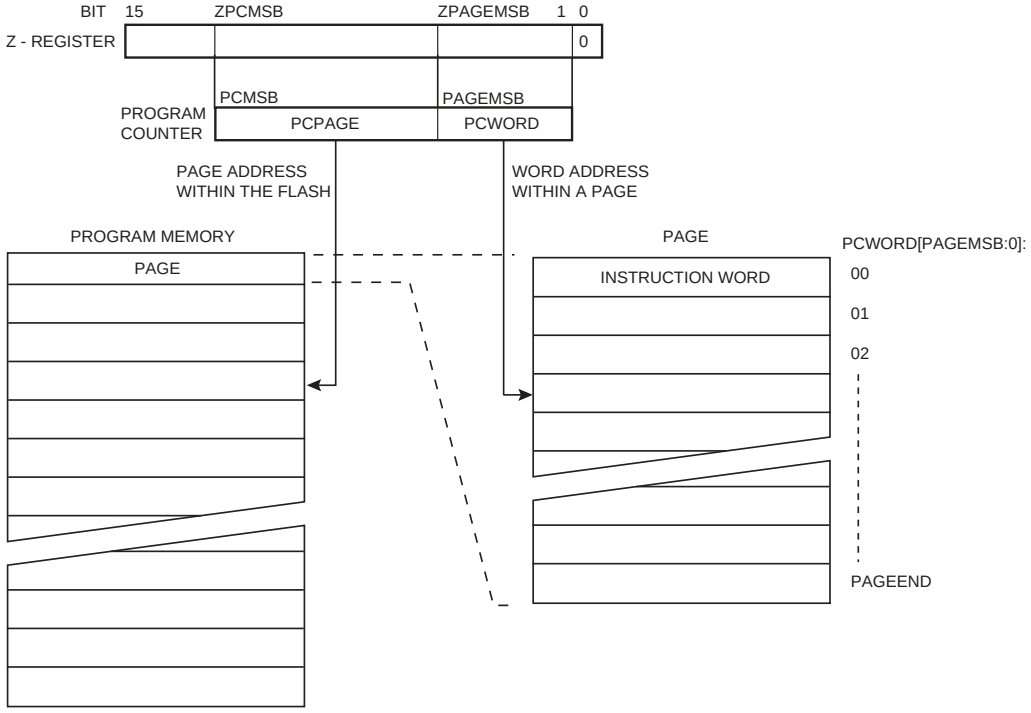
The Z-pointer is used to address the SPM commands.

Bit	15	14	13	12	11	10	9	8
ZH (R31)	**Z15**	**Z14**	**Z13**	**Z12**	**Z11**	**Z10**	**Z9**	**Z8**
ZL (R30)	**Z7**	**Z6**	**Z5**	**Z4**	**Z3**	**Z2**	**Z1**	**Z0**
	7	6	5	4	3	2	1	0

Since the Flash is organized in pages (see Table 111 on page 258), the Program Counter can be treated as having two different sections. One section, consisting of the least significant bits, is addressing the words within a page, while the most significant bits are addressing the pages. This is shown in Figure 126. Note that the Page Erase and Page Write operations are addressed independently. Therefore it is of major importance that the Boot Loader software addresses the same page in both the Page Erase and Page Write operation. Once a programming operation is initiated, the address is latched and the Z-pointer can be used for other operations.

The only SPM operation that does not use the Z-pointer is Setting the Boot Loader Lock bits. The content of the Z-pointer is ignored and will have no effect on the operation. The LPM instruction does also use the Z pointer to store the address. Since this instruction addresses the Flash byte by byte, also the LSB (bit Z0) of the Z-pointer is used.

Figure 126. Addressing the Flash during SPM[1]



Note: 1. The different variables used in Figure 126 are listed in Table 102 on page 253.

Self-Programming the Flash

The program memory is updated in a page by page fashion. Before programming a page with the data stored in the temporary page buffer, the page must be erased. The temporary page buffer is filled one word at a time using SPM and the buffer can be filled either before the page erase command or between a page erase and a page write operation:

Alternative 1, fill the buffer before a Page Erase

- Fill temporary page buffer
- Perform a Page Erase
- Perform a Page Write

Alternative 2, fill the buffer after Page Erase

- Perform a Page Erase
- Fill temporary page buffer
- Perform a Page Write

If only a part of the page needs to be changed, the rest of the page must be stored (for example in the temporary page buffer) before the erase, and then be rewritten. When using alternative 1, the Boot Loader provides an effective Read-Modify-Write feature which allows the user software to first read the page, do the necessary changes, and then write back the modified data. If alternative 2 is used, it is not possible to read the old data while loading since the page is already erased. The temporary page buffer can be accessed in a random sequence. It is essential that the page address used in both the page erase and page write operation is addressing the same page. See “Simple Assembly Code Example for a Boot Loader” on page 251 for an assembly code example.

Performing Page Erase by SPM

To execute Page Erase, set up the address in the Z-pointer, write “X0000011” to SPMCR and execute SPM within four clock cycles after writing SPMCR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE in the Z-register. Other bits in the Z-pointer must be written zero during this operation.

- Page Erase to the RWW section: The NRWW section can be read during the page erase.
- Page Erase to the NRWW section: The CPU is halted during the operation.

Filling the Temporary Buffer (Page Loading)

To write an instruction word, set up the address in the Z-pointer and data in R1:R0, write “00000001” to SPMCR and execute SPM within four clock cycles after writing SPMCR. The content of PCWORD in the Z-register is used to address the data in the temporary buffer. The temporary buffer will auto-erase after a page write operation or by writing the RWWSRE bit in SPMCR. It is also erased after a system reset. Note that it is not possible to write more than one time to each address without erasing the temporary buffer.

Note: If the EEPROM is written in the middle of an SPM Page Load operation, all data loaded will be lost.

Performing a Page Write

To execute Page Write, set up the address in the Z-pointer, write “X0000101” to SPMCR and execute SPM within four clock cycles after writing SPMCR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE. Other bits in the Z-pointer must be written to zero during this operation.

- Page Write to the RWW section: The NRWW section can be read during the Page Write.
- Page Write to the NRWW section: The CPU is halted during the operation.

Using the SPM Interrupt

If the SPM interrupt is enabled, the SPM interrupt will generate a constant interrupt when the SPMEN bit in SPMCR is cleared. This means that the interrupt can be used instead of polling the SPMCR Register in software. When using the SPM interrupt, the Interrupt Vectors should be moved to the BLS section to avoid that an interrupt is accessing the RWW section when it is blocked for reading. How to move the interrupts is described in “Interrupts” on page 42.

Consideration while Updating BLS

Special care must be taken if the user allows the Boot Loader section to be updated by leaving Boot Lock bit11 unprogrammed. An accidental write to the Boot Loader itself can corrupt the entire Boot Loader, and further software updates might be impossible. If it is not necessary to change the Boot Loader software itself, it is recommended to program the Boot Lock bit11 to protect the Boot Loader software from any internal software changes.

Prevent Reading the RWW Section during Self-Programming

During Self-Programming (either Page Erase or Page Write), the RWW section is always blocked for reading. The user software itself must prevent that this section is addressed during the Self-Programming operation. The RWWSB in the SPMCR will be set as long as the RWW section is busy. During self-programming the Interrupt Vector table should be moved to the BLS as described in “Interrupts” on page 42, or the interrupts must be disabled. Before addressing the RWW section after the programming is completed, the user software must clear the RWWSB by writing the RWWSRE. See “Simple Assembly Code Example for a Boot Loader” on page 251 for an example.

Setting the Boot Loader Lock Bits by SPM

To set the Boot Loader Lock bits, write the desired data to R0, write “X0001001” to SPMCR and execute SPM within four clock cycles after writing SPMCR. The only accessible Lock bits are the Boot Lock bits that may prevent the Application and Boot Loader section from any software update by the MCU.

Bit	7	6	5	4	3	2	1	0
R0	1	1	BLB12	BLB11	BLB02	BLB01	1	1

See Table 96 and Table 97 for how the different settings of the Boot Loader bits affect the Flash access.

If bits 5..2 in R0 are cleared (zero), the corresponding Boot Lock bit will be programmed if an SPM instruction is executed within four cycles after BLBSET and SPMEN are set in SPMCR. The Z-pointer is don't care during this operation, but for future compatibility it is recommended to load the Z-pointer with $0001 (same as used for reading the Lock bits). For future compatibility It is also recommended to set bits 7, 6, 1, and 0 in R0 to "1" when writing the Lock bits. When programming the Lock bits the entire Flash can be read during the operation.

EEPROM Write Prevents Writing to SPMCR

Note that an EEPROM write operation will block all software programming to Flash. Reading the Fuses and Lock bits from software will also be prevented during the EEPROM write operation. It is recommended that the user checks the status bit (EEWE) in the EECR Register and verifies that the bit is cleared before writing to the SPMCR Register.

Reading the Fuse and Lock Bits from Software

It is possible to read both the Fuse and Lock bits from software. To read the Lock bits, load the Z-pointer with $0001 and set the BLBSET and SPMEN bits in SPMCR. When an LPM instruction is executed within three CPU cycles after the BLBSET and SPMEN bits are set in SPMCR, the value of the Lock bits will be loaded in the destination register. The BLBSET and SPMEN bits will auto-clear upon completion of reading the Lock bits or if no LPM instruction is executed within three CPU cycles or no SPM instruction is executed within four CPU cycles. When BLBSET and SPMEN are cleared, LPM will work as described in the Instruction set Manual.

Bit	7	6	5	4	3	2	1	0
Rd	–	–	BLB12	BLB11	BLB02	BLB01	LB2	LB1

The algorithm for reading the Fuse Low bits is similar to the one described above for reading the Lock bits. To read the Fuse Low bits, load the Z-pointer with $0000 and set the BLBSET and SPMEN bits in SPMCR. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCR, the value of the Fuse Low bits (FLB) will be loaded in the destination register as shown below. Refer to Table 106 on page 256 for a detailed description and mapping of the Fuse Low bits.

Bit	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

Similarly, when reading the Fuse High bits, load $0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCR, the value of the Fuse High bits (FHB) will be loaded in the destination register as shown below. Refer to Table 105 on page 255 for detailed description and mapping of the Fuse High bits.

Bit	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

Fuse and Lock bits that are programmed, will be read as zero. Fuse and Lock bits that are unprogrammed, will be read as one.

Preventing Flash Corruption

During periods of low $V_{CC,}$ the Flash program can be corrupted because the supply voltage is too low for the CPU and the Flash to operate properly. These issues are the same

as for board level systems using the Flash, and the same design solutions should be applied.

A Flash program corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the Flash requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage for executing instructions is too low.

Flash corruption can easily be avoided by following these design recommendations (one is sufficient):

1. If there is no need for a Boot Loader update in the system, program the Boot Loader Lock bits to prevent any Boot Loader software updates.
2. Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD) if the operating voltage matches the detection level. If not, an external low V_{CC} Reset Protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.
3. Keep the AVR core in Power-down Sleep mode during periods of low V_{CC}. This will prevent the CPU from attempting to decode and execute instructions, effectively protecting the SPMCR Register and thus the Flash from unintentional writes.

Programming Time for Flash when using SPM

The Calibrated RC Oscillator is used to time Flash accesses. Table 99 shows the typical programming time for Flash accesses from the CPU.

Table 99. SPM Programming Time.

Symbol	Min Programming Time	Max Programming Time
Flash write (Page Erase, Page Write, and write Lock bits by SPM)	3.7 ms	4.5 ms

Simple Assembly Code Example for a Boot Loader

```
  ;-the routine writes one page of data from RAM to Flash
  ; the first data location in RAM is pointed to by the Y pointer
  ; the first data location in Flash is pointed to by the Z pointer
  ;-error handling is not included
  ;-the routine must be placed inside the boot space
  ; (at least the Do_spm sub routine). Only code inside NRWW section can
  ; be read during self-programming (page erase and page write).
  ;-registers used: r0, r1, temp1 (r16), temp2 (r17), looplo (r24),
  ; loophi (r25), spmcrval (r20)
  ; storing and restoring of registers is not included in the routine
  ; register usage can be optimized at the expense of code size
  ;-It is assumed that either the interrupt table is moved to the Boot
  ; loader section or that the interrupts are disabled.
.equ  PAGESIZEB = PAGESIZE*2        ; PAGESIZEB is page size in BYTES, not
                                    ; words
.org SMALLBOOTSTART
Write_page:
  ; page erase
  ldi   spmcrval, (1<<PGERS) | (1<<SPMEN)
  call  Do_spm

  ; re-enable the RWW section
  ldi   spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm

  ; transfer data from RAM to Flash page buffer
  ldi   looplo, low(PAGESIZEB)      ;init loop variable
```

```
  ldi   loophi, high(PAGESIZEB)     ;not required for PAGESIZEB<=256
Wrloop:
  ld    r0, Y+
  ld    r1, Y+
  ldi   spmcrval, (1<<SPMEN)
  call  Do_spm
  adiw  ZH:ZL, 2
  sbiw  loophi:looplo, 2            ;use subi for PAGESIZEB<=256
  brne  Wrloop

  ; execute page write
  subi  ZL, low(PAGESIZEB)          ;restore pointer
  sbci  ZH, high(PAGESIZEB)         ;not required for PAGESIZEB<=256
  ldi   spmcrval, (1<<PGWRT) | (1<<SPMEN)
  call  Do_spm

  ; re-enable the RWW section
  ldi   spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm

  ; read back and check, optional
  ldi   looplo, low(PAGESIZEB)      ;init loop variable
  ldi   loophi, high(PAGESIZEB)     ;not required for PAGESIZEB<=256
  subi  YL, low(PAGESIZEB)          ;restore pointer
  sbci  YH, high(PAGESIZEB)
Rdloop:
  lpm   r0, Z+
  ld    r1, Y+
  cpse  r0, r1
  jmp   Error
  sbiw  loophi:looplo, 1            ;use subi for PAGESIZEB<=256
  brne  Rdloop

  ; return to RWW section
  ; verify that RWW section is safe to read
Return:
  in    temp1, SPMCR
  sbrs  temp1, RWWSB              ; If RWWSB is set, the RWW section is not
                                  ; ready yet
  ret
  ; re-enable the RWW section
  ldi   spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm
  rjmp  Return

Do_spm:
  ; check for previous SPM complete
Wait_spm:
  in    temp1, SPMCR
  sbrc  temp1, SPMEN
  rjmp  Wait_spm
  ; input: spmcrval determines SPM action
  ; disable interrupts if enabled, store status
  in    temp2, SREG
  cli
  ; check that no EEPROM write access is present
Wait_ee:
  sbic  EECR, EEWE
  rjmp  Wait_ee
  ; SPM timed sequence
  out   SPMCR, spmcrval
  spm
  ; restore SREG (to enable interrupts if originally enabled)
  out   SREG, temp2
  ret
```

ATmega32 Boot Loader Parameters

In Table 100 through Table 102, the parameters used in the description of the self programming are given.

Table 100. Boot Size Configuration[1]

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application section	Boot Reset Address (start Boot Loader Section)
1	1	256 words	4	$0000 - $3EFF	$3F00 - $3FFF	$3EFF	$3F00
1	0	512 words	8	$0000 - $3DFF	$3E00 - $3FFF	$3DFF	$3E00
0	1	1024 words	16	$0000 - $3BFF	$3C00 - $3FFF	$3BFF	$3C00
0	0	2048 words	32	$0000 - $37FF	$3800 - $3FFF	$37FF	$3800

Note: 1. The different BOOTSZ Fuse configurations are shown in Figure 125

Table 101. Read-While-Write Limit[1]

Section	Pages	Address
Read-While-Write section (RWW)	224	$0000 - $37FF
No Read-While-Write section (NRWW)	32	$3800 - $3FFF

Note: 1. For details about these two section, see “NRWW – No Read-While-Write Section” on page 243 and “RWW – Read-While-Write Section” on page 243

Table 102. Explanation of Different Variables used in Figure 126 and the Mapping to the Z-pointer

Variable		Corresponding Z-value[1]	Description
PCMSB	13		Most significant bit in the Program Counter. (The Program Counter is 14 bits PC[13:0])
PAGEMSB	5		Most significant bit which is used to address the words within one page (64 words in a page requires 6 bits PC [5:0]).
ZPCMSB		Z14	Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1.
ZPAGEMSB		Z6	Bit in Z-register that is mapped to PAGEMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1.
PCPAGE	PC[13:6]	Z14:Z7	Program Counter page address: Page select, for page erase and page write
PCWORD	PC[5:0]	Z6:Z1	Program Counter word address: Word select, for filling temporary buffer (must be zero during page write operation)

Note: 1. Z15: always ignored
Z0: should be zero for all SPM commands, byte select for the LPM instruction.
See “Addressing the Flash during Self-Programming” on page 247 for details about the use of Z-pointer during Self-Programming.

Program And Data Memory Lock Bits

The ATmega32 provides six Lock bits which can be left unprogrammed (“1”) or can be programmed (“0”) to obtain the additional features listed in Table 104. The Lock bits can only be erased to “1” with the Chip Erase command.

Table 103. Lock Bit Byte[1]

Lock Bit Byte	Bit No.	Description	Default Value
	7	–	1 (unprogrammed)
	6	–	1 (unprogrammed)
BLB12	5	Boot Lock bit	1 (unprogrammed)
BLB11	4	Boot Lock bit	1 (unprogrammed)
BLB02	3	Boot Lock bit	1 (unprogrammed)
BLB01	2	Boot Lock bit	1 (unprogrammed)
LB2	1	Lock bit	1 (unprogrammed)
LB1	0	Lock bit	1 (unprogrammed)

Note: 1. “1” means unprogrammed, “0” means programmed

Table 104. Lock Bit Protection Modes

Memory Lock Bits[2]			Protection Type
LB Mode	**LB2**	**LB1**	
1	1	1	No memory lock features enabled.
2	1	0	Further programming of the Flash and EEPROM is disabled in Parallel and SPI/JTAG Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode.[1]
3	0	0	Further programming and verification of the Flash and EEPROM is disabled in Parallel and SPI/JTAG Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode.[1]
BLB0 Mode	**BLB02**	**BLB01**	
1	1	1	No restrictions for SPM or LPM accessing the Application section.
2	1	0	SPM is not allowed to write to the Application section.
3	0	0	SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If interrupt vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
4	0	1	LPM executing from the Boot Loader section is not allowed to read from the Application section. If interrupt vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
BLB1 Mode	**BLB12**	**BLB11**	

Table 104. Lock Bit Protection Modes (Continued)

Memory Lock Bits[2]			Protection Type
1	1	1	No restrictions for SPM or LPM accessing the Boot Loader section.
2	1	0	SPM is not allowed to write to the Boot Loader section.
3	0	0	SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If interrupt vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.
4	0	1	LPM executing from the Application section is not allowed to read from the Boot Loader section. If interrupt vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.

Notes: 1. Program the fuse bits before programming the Lock bits.
2. "1" means unprogrammed, "0" means programmed

Fuse Bits

The ATmega32 has two fuse bytes. Table 105 and Table 106 describe briefly the functionality of all the fuses and how they are mapped into the fuse bytes. Note that the fuses are read as logical zero, "0", if they are programmed.

Table 105. Fuse High Byte

Fuse High Byte	Bit No.	Description	Default Value
OCDEN[4]	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN[5]	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN[1]	5	Enable SPI Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
CKOPT[2]	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 100 for details)	0 (programmed)[3]
BOOTSZ0	1	Select Boot Size (see Table 100 for details)	0 (programmed)[3]
BOOTRST	0	Select reset vector	1 (unprogrammed)

Notes: 1. The SPIEN Fuse is not accessible in SPI Serial Programming mode.
2. The CKOPT Fuse functionality depends on the setting of the CKSEL bits. See See "Clock Sources" on page 23. for details.
3. The default value of BOOTSZ1..0 results in maximum Boot Size. See Table 100 on page 253.
4. Never ship a product with the OCDEN Fuse programmed regardless of the setting of Lock bits and the JTAGEN Fuse. A programmed OCDEN Fuse enables some parts of the clock system to be running in all sleep modes. This may increase the power consumption.
5. If the JTAG interface is left unconnected, the JTAGEN fuse should if possible be disabled. This to avoid static current at the TDO pin in the JTAG interface.

Table 106. Fuse Low Byte

Fuse Low Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown-out Detector trigger level	1 (unprogrammed)
BODEN	6	Brown-out Detector enable	1 (unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed)(1)
SUT0	4	Select start-up time	0 (programmed)(1)
CKSEL3	3	Select Clock source	0 (programmed)(2)
CKSEL2	2	Select Clock source	0 (programmed)(2)
CKSEL1	1	Select Clock source	0 (programmed)(2)
CKSEL0	0	Select Clock source	1 (unprogrammed)(2)

Notes: 1. The default value of SUT1..0 results in maximum start-up time. SeeTable 10 on page 28 for details.
2. The default setting of CKSEL3..0 results in internal RC Oscillator @ 1MHz. See Table 2 on page 23 for details.

The status of the Fuse bits is not affected by Chip Erase. Note that the Fuse bits are locked if Lock bit1 (LB1) is programmed. Program the Fuse bits before programming the Lock bits.

Latching of Fuses

The Fuse values are latched when the device enters programming mode and changes of the Fuse values will have no effect until the part leaves Programming mode. This does not apply to the EESAVE Fuse which will take effect once it is programmed. The fuses are also latched on Power-up in Normal mode.

Signature Bytes

All Atmel microcontrollers have a three-byte signature code which identifies the device. This code can be read in both serial and parallel mode, also when the device is locked. The three bytes reside in a separate address space.

For the ATmega32 the signature bytes are:

1. $000: $1E (indicates manufactured by Atmel)
2. $001: $95 (indicates 32KB Flash memory)
3. $002: $02 (indicates ATmega32 device when $001 is $95)

Calibration Byte

The ATmega32 stores four different calibration values for the internal RC Oscillator. These bytes resides in the signature row High Byte of the addresses 0x000, 0x0001, 0x0002, and 0x0003 for 1, 2, 4, and 8 Mhz respectively. During Reset, the 1 MHz value is automatically loaded into the OSCCAL Register. If other frequencies are used, the calibration value has to be loaded manually, see “Oscillator Calibration Register – OSCCAL” on page 28 for details.

Parallel Programming Parameters, Pin Mapping, and Commands

This section describes how to parallel program and verify Flash Program memory, EEPROM Data memory, Memory Lock bits, and Fuse bits in the ATmega32. Pulses are assumed to be at least 250 ns unless otherwise noted.

Signal Names

In this section, some pins of the ATmega32 are referenced by signal names describing their functionality during parallel programming, see Figure 127 and Table 107. Pins not described in the following table are referenced by pin names.

The XA1/XA0 pins determine the action executed when the XTAL1 pin is given a positive pulse. The bit coding is shown in Table 109.

When pulsing $\overline{WR}$ or $\overline{OE}$, the command loaded determines the action executed. The different Commands are shown in Table 110.

Figure 127. Parallel Programming

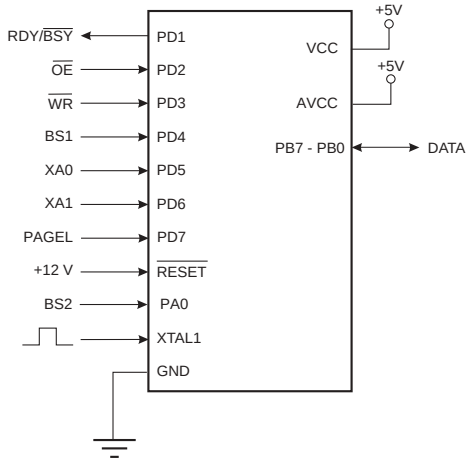


Table 107. Pin Name Mapping

Signal Name in Programming Mode	Pin Name	I/O	Function
RDY/$\overline{BSY}$	PD1	O	0: Device is busy programming, 1: Device is ready for new command
$\overline{OE}$	PD2	I	Output Enable (Active low)
$\overline{WR}$	PD3	I	Write Pulse (Active low)
BS1	PD4	I	Byte Select 1 (“0” selects low byte, “1” selects high byte)
XA0	PD5	I	XTAL Action Bit 0
XA1	PD6	I	XTAL Action Bit 1
PAGEL	PD7	I	Program Memory and EEPROM data Page Load
BS2	PA0	I	Byte Select 2 (“0” selects low byte, “1” selects 2’nd high byte)
DATA	PB7-0	I/O	Bidirectional Data bus (Output when $\overline{OE}$ is low)

Table 108. Pin Values used to Enter Programming Mode

Pin	Symbol	Value
PAGEL	Prog_enable[3]	0
XA1	Prog_enable[2]	0
XA0	Prog_enable[1]	0
BS1	Prog_enable[0]	0

Table 109. XA1 and XA0 Coding

XA1	XA0	Action when XTAL1 is Pulsed
0	0	Load Flash or EEPROM Address (High or low address byte determined by BS1)
0	1	Load Data (High or Low data byte for Flash determined by BS1)
1	0	Load Command
1	1	No Action, Idle

Table 110. Command Byte Bit Coding

Command Byte	Command Executed
1000 0000	Chip Erase
0100 0000	Write Fuse Bits
0010 0000	Write Lock Bits
0001 0000	Write Flash
0001 0001	Write EEPROM
0000 1000	Read Signature Bytes and Calibration byte
0000 0100	Read Fuse and Lock bits
0000 0010	Read Flash
0000 0011	Read EEPROM

Table 111. No. of Words in a Page and no. of Pages in the Flash

Flash Size	Page Size	PCWORD	No. of Pages	PCPAGE	PCMSB
16K words (32K bytes)	64 words	PC[5:0]	256	PC[13:6]	13

Table 112. No. of Words in a Page and no. of Pages in the EEPROM

EEPROM Size	Page Size	PCWORD	No. of Pages	PCPAGE	EEAMSB
1024 bytes	4 bytes	EEA[1:0]	256	EEA[9:2]	9

Parallel Programming

Enter Programming Mode

The following algorithm puts the device in Parallel Programming mode:

1. Apply 4.5 - 5.5V between V_{CC} and GND, and wait at least 100 µs.
2. Set $\overline{\text{RESET}}$ to “0” and toggle XTAL1 at least 6 times
3. Set the Prog_enable pins listed in Table 108 on page 258 to “0000” and wait at least 100 ns.
4. Apply 11.5 - 12.5V to $\overline{\text{RESET}}$. Any activity on Prog_enable pins within 100 ns after +12V has been applied to $\overline{\text{RESET}}$, will cause the device to fail entering Programming mode.

Note, if External Crystal or External RC configuration is selected, it may not be possible to apply qualified XTAL1 pulses. In such cases, the following algorithm should be followed:

1. Set Prog_enable pins listed in Table 108 on page 258 to “0000”.
2. Apply 4.5 - 5.5V between V_{CC} and GND simultanously as 11.5 - 12.5V is applied to $\overline{\text{RESET}}$.
3. Wait 100 µs.
4. Re-program the fuses to ensure that External Clock is selected as clock source (CKSEL3:0 = 0b0000) If Lock bits are programmed, a Chip Erase command must be executed before changing the fuses.
5. Exit Programming mode by power the device down or by bringing $\overline{\text{RESET}}$ pin to 0b0.
6. Entering Programming mode with the original algorithm, as described above.

Considerations for Efficient Programming

The loaded command and address are retained in the device during programming. For efficient programming, the following should be considered.

- The command needs only be loaded once when writing or reading multiple memory locations.
- Skip writing the data value $FF, that is the contents of the entire EEPROM (unless the EESAVE fuse is programmed) and Flash after a Chip Erase.
- Address high byte needs only be loaded before programming or reading a new 256 word window in Flash or 256 byte EEPROM. This consideration also applies to Signature bytes reading.

Chip Erase

The Chip Erase will erase the Flash and EEPROM[1] memories plus Lock bits. The Lock bits are not reset until the program memory has been completely erased. The Fuse bits are not changed. A Chip Erase must be performed before the Flash and/or the EEPROM are reprogrammed.

Note: 1. The EEPRPOM memory is preserved during chip erase if the EESAVE Fuse is programmed.

Load Command “Chip Erase”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “1000 0000”. This is the command for Chip Erase.
4. Give XTAL1 a positive pulse. This loads the command.
5. Give $\overline{\text{WR}}$ a negative pulse. This starts the Chip Erase. RDY/$\overline{\text{BSY}}$ goes low.
6. Wait until RDY/$\overline{\text{BSY}}$ goes high before loading a new command.

Programming the Flash

The Flash is organized in pages, see Table 111 on page 258. When programming the Flash, the program data is latched into a page buffer. This allows one page of program data to be programmed simultaneously. The following procedure describes how to program the entire Flash memory:

A. Load Command “Write Flash”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “0001 0000”. This is the command for Write Flash.
4. Give XTAL1 a positive pulse. This loads the command.

B. Load Address Low byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “0”. This selects low address.
3. Set DATA = Address low byte ($00 - $FF).
4. Give XTAL1 a positive pulse. This loads the address low byte.

C. Load Data Low Byte

1. Set XA1, XA0 to “01”. This enables data loading.
2. Set DATA = Data low byte ($00 - $FF).
3. Give XTAL1 a positive pulse. This loads the data byte.

D. Load Data High Byte

1. Set BS1 to “1”. This selects high data byte.
2. Set XA1, XA0 to “01”. This enables data loading.
3. Set DATA = Data high byte ($00 - $FF).
4. Give XTAL1 a positive pulse. This loads the data byte.

E. Latch Data

1. Set BS1 to “1”. This selects high data byte.
2. Give PAGEL a positive pulse. This latches the data bytes. (See Figure 129 for signal waveforms)

F. Repeat B through E until the entire buffer is filled or until all data within the page is loaded.

While the lower bits in the address are mapped to words within the page, the higher bits address the pages within the FLASH. This is illustrated in Figure 128 on page 261. Note that if less than 8 bits are required to address words in the page (pagesize < 256), the most significant bit(s) in the address low byte are used to address the page when performing a page write.

G. Load Address High byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “1”. This selects high address.
3. Set DATA = Address high byte ($00 - $FF).
4. Give XTAL1 a positive pulse. This loads the address high byte.

H. Program Page

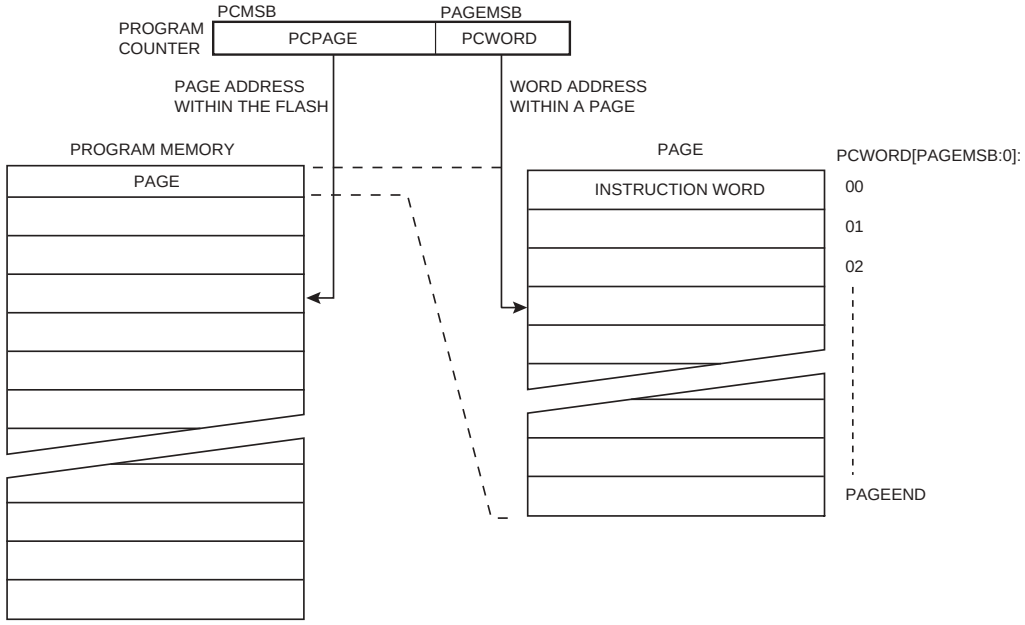
1. Set BS1 = “0”
2. Give $\overline{WR}$ a negative pulse. This starts programming of the entire page of data. RDY/$\overline{BSY}$goes low.
3. Wait until RDY/$\overline{BSY}$ goes high. (See Figure 129 for signal waveforms)

I. Repeat B through H until the entire Flash is programmed or until all data has been programmed.

J. End Page Programming

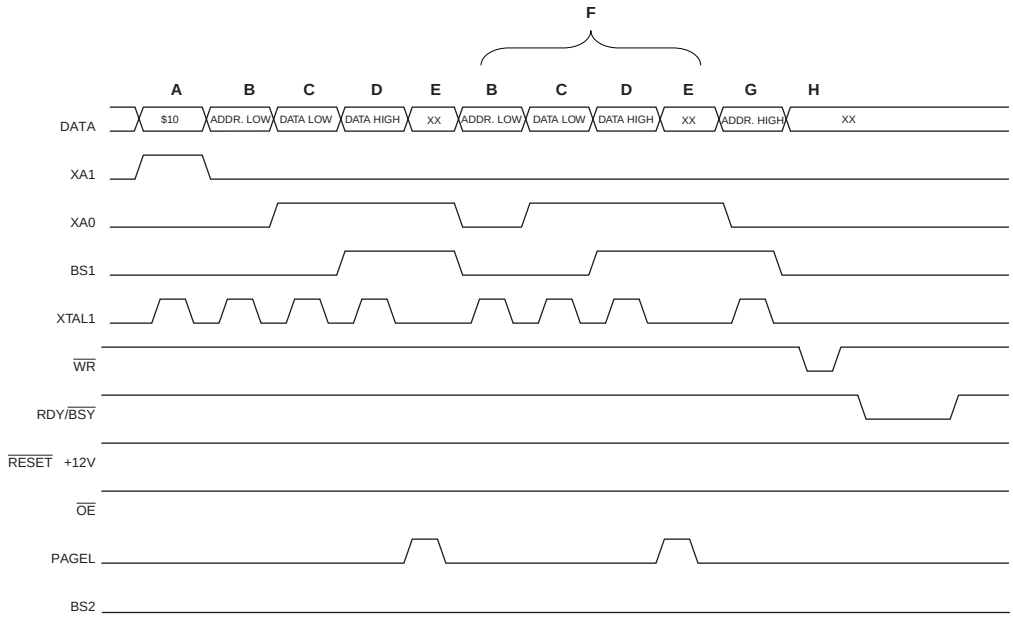
1. 1. Set XA1, XA0 to “10”. This enables command loading.
2. Set DATA to “0000 0000”. This is the command for No Operation.
3. Give XTAL1 a positive pulse. This loads the command, and the internal write signals are reset.

Figure 128. Addressing the Flash which is Organized in Pages



Note: 1. PCPAGE and PCWORD are listed in Table 111 on page 258.

Figure 129. Programming the Flash Waveforms[1]



Note: 1. “XX” is don’t care. The letters refer to the programming description above.

Programming the EEPROM

The EEPROM is organized in pages, see Table 112 on page 258. When programming the EEPROM, the program data is latched into a page buffer. This allows one page of data to be programmed simultaneously. The programming algorithm for the EEPROM data memory is as follows (refer to “Programming the Flash” on page 260 for details on Command, Address and Data loading):

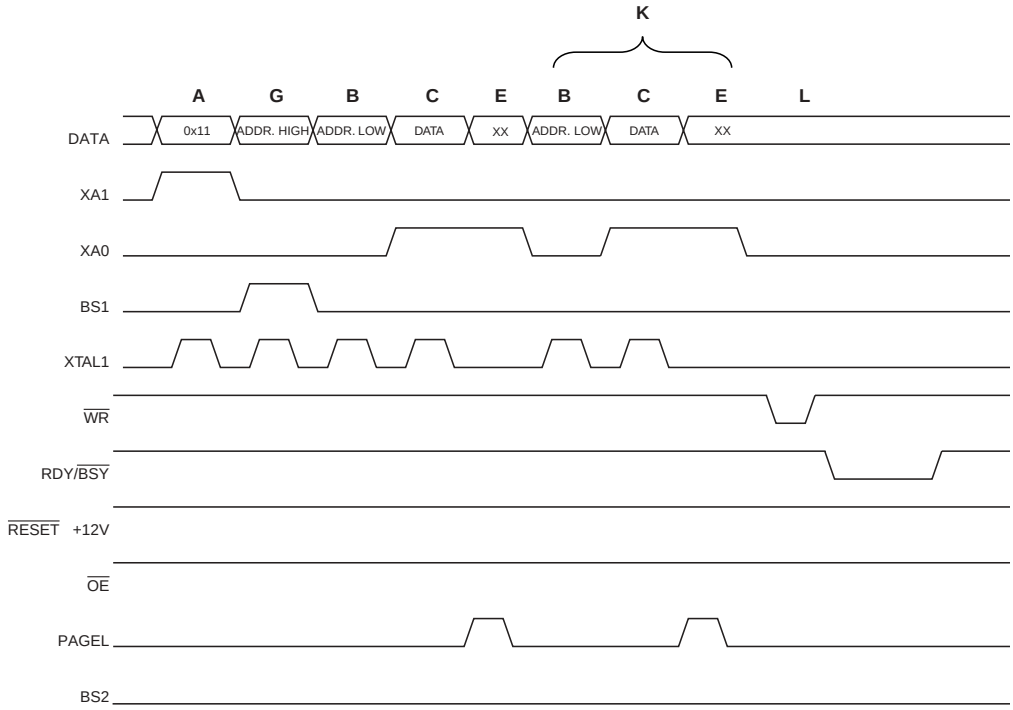
1. A: Load Command “0001 0001”.
2. G: Load Address High Byte ($00 - $FF)
3. B: Load Address Low Byte ($00 - $FF)
4. C: Load Data ($00 - $FF)
5. E: Latch data (give PAGEL a positive pulse)

K: Repeat 3 through 5 until the entire buffer is filled

L: Program EEPROM page

1. Set BS1 to “0”.
2. Give $\overline{WR}$ a negative pulse. This starts programming of the EEPROM page. RDY/$\overline{BSY}$ goes low.
3. Wait until to RDY/$\overline{BSY}$ goes high before programming the next page. (See Figure 130 for signal waveforms)

Figure 130. Programming the EEPROM Waveforms



Reading the Flash

The algorithm for reading the Flash memory is as follows (refer to “Programming the Flash” on page 260 for details on Command and Address loading):

1. A: Load Command “0000 0010”.
2. G: Load Address High Byte ($00 - $FF)
3. B: Load Address Low Byte ($00 - $FF)
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The Flash word low byte can now be read at DATA.
5. Set BS1 to “1”. The Flash word high byte can now be read at DATA.

6. Set $\overline{OE}$ to “1”.

Reading the EEPROM

The algorithm for reading the EEPROM memory is as follows (refer to “Programming the Flash” on page 260 for details on Command and Address loading):

1. A: Load Command “0000 0011”.
2. G: Load Address High Byte ($00 - $FF)
3. B: Load Address Low Byte ($00 - $FF)
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The EEPROM Data byte can now be read at DATA.
5. Set $\overline{OE}$ to “1”.

Programming the Fuse Low Bits

The algorithm for programming the Fuse Low bits is as follows (refer to “Programming the Flash” on page 260 for details on Command and Data loading):

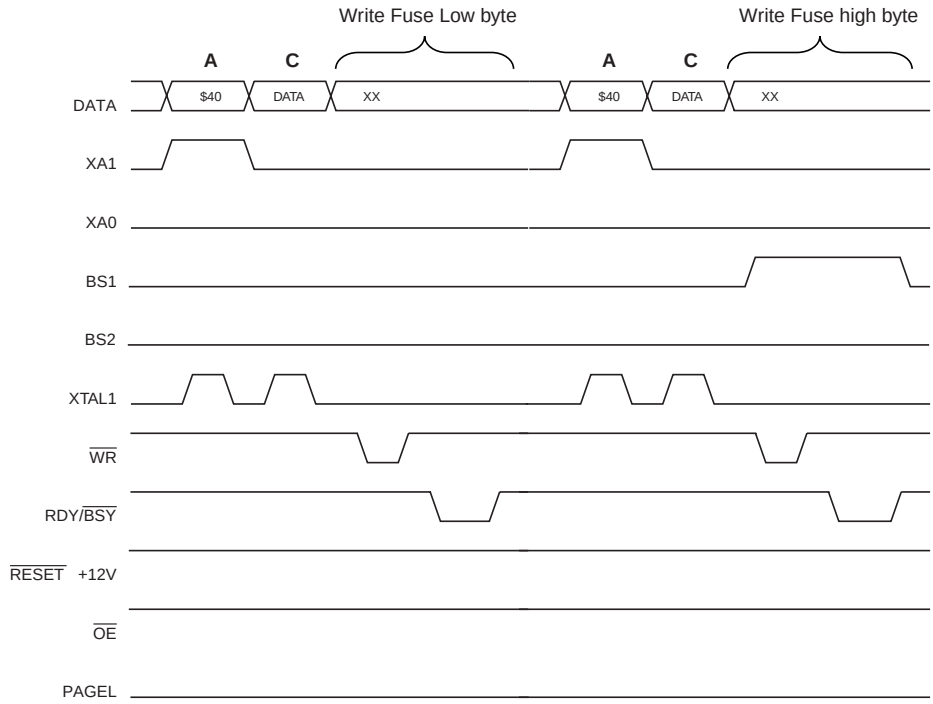
1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Set BS1 to “0” and BS2 to “0”.
4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

Programming the Fuse High Bits

The algorithm for programming the Fuse high bits is as follows (refer to “Programming the Flash” on page 260 for details on Command and Data loading):

1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Set BS1 to “1” and BS2 to “0”. This selects high data byte.
4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. Set BS1 to “0”. This selects low data byte.

Figure 131. Programming the Fuses



Programming the Lock Bits

The algorithm for programming the Lock bits is as follows (refer to “Programming the Flash” on page 260 for details on Command and Data loading):

1. A: Load Command “0010 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs the Lock bit.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

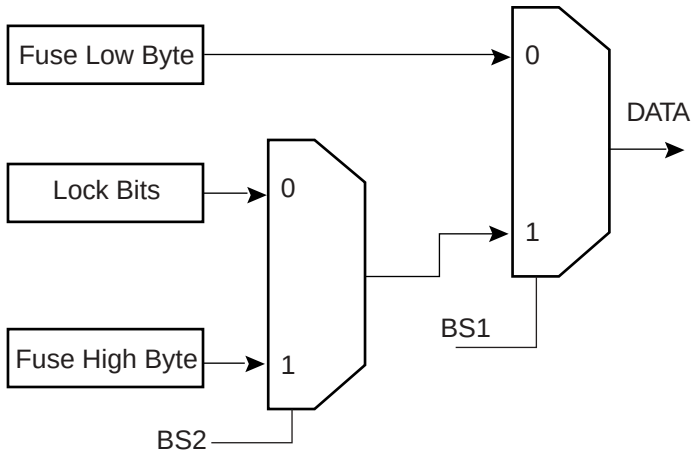
The Lock bits can only be cleared by executing Chip Erase.

Reading the Fuse and Lock Bits

The algorithm for reading the Fuse and Lock bits is as follows (refer to “Programming the Flash” on page 260 for details on Command loading):

1. A: Load Command “0000 0100”.
2. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “0”. The status of the Fuse Low bits can now be read at DATA (“0” means programmed).
3. Set $\overline{OE}$ to “0”, BS2 to “1” and BS1 to “1”. The status of the Fuse High bits can now be read at DATA (“0” means programmed).
4. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “1”. The status of the Lock bits can now be read at DATA (“0” means programmed).
5. Set $\overline{OE}$ to “1”.

Figure 132. Mapping between BS1, BS2 and the Fuse- and Lock Bits during Read



Reading the Signature Bytes

The algorithm for reading the Signature bytes is as follows (refer to “Programming the Flash” on page 260 for details on Command and Address loading):

1. A: Load Command “0000 1000”.
2. B: Load Address Low Byte ($00 - $02).
3. Set $\overline{OE}$ to “0”, and BS1 to “0”. The selected Signature byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

Reading the Calibration Byte

The algorithm for reading the Calibration byte is as follows (refer to “Programming the Flash” on page 260 for details on Command and Address loading):

1. A: Load Command “0000 1000”.
2. B: Load Address Low Byte, $00.
3. Set $\overline{OE}$ to “0”, and BS1 to “1”. The Calibration byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

Parallel Programming Characteristics

Figure 133. Parallel Programming Timing, Including some General Timing Requirements

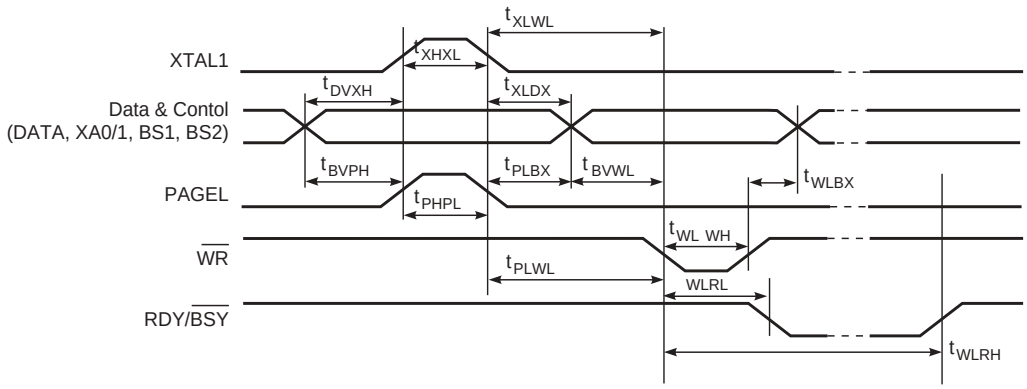
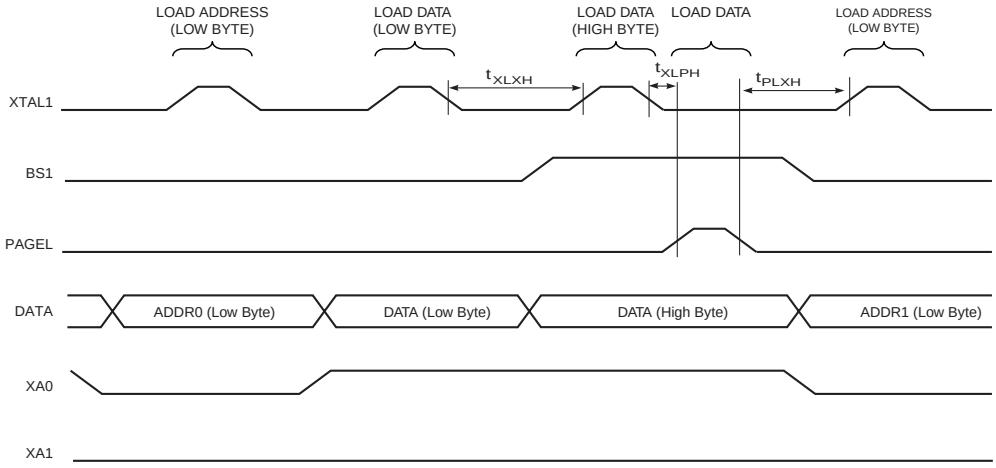
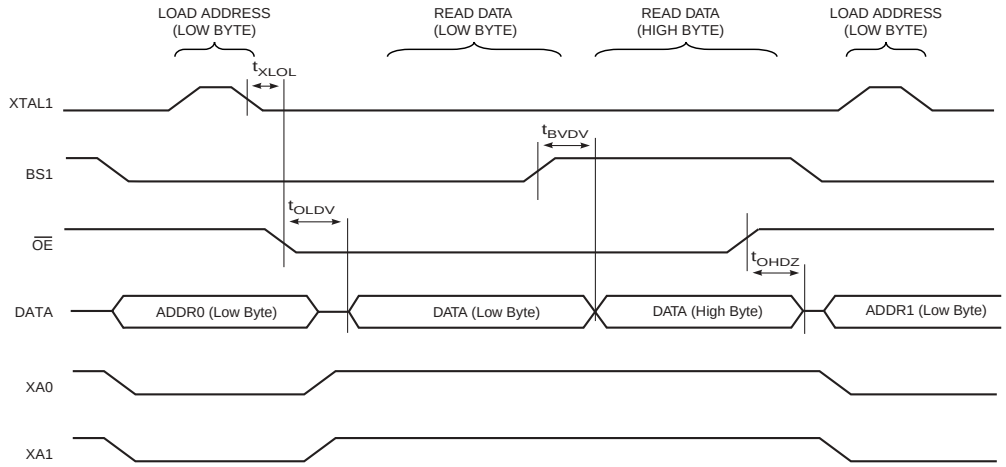


Figure 134. Parallel Programming Timing, Loading Sequence with Timing Requirements(1)



Note: 1. The timing requirements shown in Figure 133 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to loading operation.

Figure 135. Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements(1)



Note: 1. The timing requirements shown in Figure 133 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to reading operation.

Table 113. Parallel Programming Characteristics, V_{CC} = 5 V ± 10%

Symbol	Parameter	Min	Typ	Max	Units
V_{PP}	Programming Enable Voltage	11.5		12.5	V
I_{PP}	Programming Enable Current			250	µA
t_{DVXH}	Data and Control Valid before XTAL1 High	67			ns
t_{XLXH}	XTAL1 Low to XTAL1 High	200			ns
t_{XHXL}	XTAL1 Pulse Width High	150			ns
t_{XLDX}	Data and Control Hold after XTAL1 Low	67			ns
t_{XLWL}	XTAL1 Low to $\overline{WR}$ Low	0			ns
t_{XLPH}	XTAL1 Low to PAGEL high	0			ns
t_{PLXH}	PAGEL low to XTAL1 high	150			ns
t_{BVPH}	BS1 Valid before PAGEL High	67			ns
t_{PHPL}	PAGEL Pulse Width High	150			ns
t_{PLBX}	BS1 Hold after PAGEL Low	67			ns
t_{WLBX}	BS2/1 Hold after $\overline{WR}$ Low	67			ns
t_{PLWL}	PAGEL Low to $\overline{WR}$ Low	67			ns
t_{BVWL}	BS1 Valid to $\overline{WR}$ Low	67			ns
t_{WLWH}	$\overline{WR}$ Pulse Width Low	150			ns
t_{WLRL}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ Low	0		1	µs
t_{WLRH}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ High(1)	3.7		4.5	ms
t_{WLRH_CE}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ High for Chip Erase(2)	7.5		9	ms
t_{XLOL}	XTAL1 Low to $\overline{OE}$ Low	0			ns

Table 113. Parallel Programming Characteristics, V_{CC} = 5 V ± 10% (Continued)

Symbol	Parameter	Min	Typ	Max	Units
t_{BVDV}	BS1 Valid to DATA valid	0		250	ns
t_{OLDV}	$\overline{OE}$ Low to DATA Valid			250	ns
t_{OHDZ}	$\overline{OE}$ High to DATA Tri-stated			250	ns

Notes: 1. t_{WLRH} is valid for the Write Flash, Write EEPROM, Write Fuse bits and Write Lock bits commands.
2. t_{WLRH_CE} is valid for the Chip Erase command.

SPI Serial Downloading

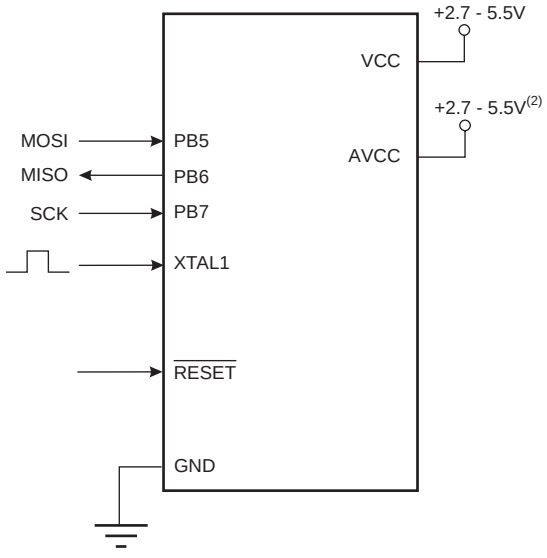
Both the Flash and EEPROM memory arrays can be programmed using the serial SPI bus while $\overline{RESET}$ is pulled to GND. The serial interface consists of pins SCK, MOSI (input), and MISO (output). After $\overline{RESET}$ is set low, the Programming Enable instruction needs to be executed first before program/erase operations can be executed. NOTE, in Table 114 on page 268, the pin mapping for SPI programming is listed. Not all parts use the SPI pins dedicated for the internal SPI interface.

SPI Serial Programming Pin Mapping

Table 114. Pin Mapping SPI Serial Programming

Symbol	Pins	I/O	Description
MOSI	PB5	I	Serial Data in
MISO	PB6	O	Serial Data out
SCK	PB7	I	Serial Clock

Figure 136. SPI Serial Programming and Verify(1)



Notes: 1. If the device is clocked by the Internal Oscillator, it is no need to connect a clock source to the XTAL1 pin.
2. V_{CC} -0.3V < AVCC < V_{CC} +0.3V, however, AVCC should always be within 2.7 - 5.5V

When programming the EEPROM, an auto-erase cycle is built into the self-timed programming operation (in the serial mode ONLY) and there is no need to first execute the

Chip Erase instruction. The Chip Erase operation turns the content of every memory location in both the Program and EEPROM arrays into $FF.

Depending on CKSEL Fuses, a valid clock must be present. The minimum low and high periods for the serial clock (SCK) input are defined as follows:

Low:> 2 CPU clock cycles for f_{ck} < 12 MHz, 3 CPU clock cycles for f_{ck} ≥ 12 MHz

High:> 2 CPU clock cycles for f_{ck} < 12 MHz, 3 CPU clock cycles for f_{ck} ≥ 12 MHz

SPI Serial Programming Algorithm

When writing serial data to the ATmega32, data is clocked on the rising edge of SCK.

When reading data from the ATmega32, data is clocked on the falling edge of SCK. See Figure 137 for timing details.

To program and verify the ATmega32 in the SPI Serial Programming mode, the following sequence is recommended (See four byte instruction formats in Table 116):

1. Power-up sequence:
   Apply power between V_{CC} and GND while $\overline{\text{RESET}}$ and SCK are set to “0”. In some systems, the programmer can not guarantee that SCK is held low during power-up. In this case, $\overline{\text{RESET}}$ must be given a positive pulse of at least two CPU clock cycles duration after SCK has been set to “0”.
2. Wait for at least 20 ms and enable SPI Serial Programming by sending the Programming Enable serial instruction to pin MOSI.
3. The SPI Serial Programming instructions will not work if the communication is out of synchronization. When in sync. the second byte ($53), will echo back when issuing the third byte of the Programming Enable instruction. Whether the echo is correct or not, all four bytes of the instruction must be transmitted. If the $53 did not echo back, give $\overline{\text{RESET}}$ a positive pulse and issue a new Programming Enable command.
4. The Flash is programmed one page at a time. The memory page is loaded one byte at a time by supplying the 6 LSB of the address and data together with the Load Program Memory Page instruction. To ensure correct loading of the page, the data low byte must be loaded before data high byte is applied for a given address. The Program Memory Page is stored by loading the Write Program Memory Page instruction with the 8 MSB of the address. If polling is not used, the user must wait at least t_{WD_FLASH} before issuing the next page. (See Table 115). Accessing the SPI Serial Programming interface before the Flash write operation completes can result in incorrect programming.
5. The EEPROM array is programmed one byte at a time by supplying the address and data together with the appropriate Write instruction. An EEPROM memory location is first automatically erased before new data is written. If polling is not used, the user must wait at least t_{WD_EEPROM} before issuing the next byte. (See Table 115). In a chip erased device, no $FFs in the data file(s) need to be programmed.
6. Any memory location can be verified by using the Read instruction which returns the content at the selected address at serial output MISO.
7. At the end of the programming session, $\overline{\text{RESET}}$ can be set high to commence normal operation.
8. Power-off sequence (if needed):
   Set $\overline{\text{RESET}}$ to “1”.
   Turn V_{CC} power off.

Data Polling Flash

When a page is being programmed into the Flash, reading an address location within the page being programmed will give the value $FF. At the time the device is ready for a new page, the programmed value will read correctly. This is used to determine when the next page can be written. Note that the entire page is written simultaneously and any address within the page can be used for polling. Data polling of the Flash will not work for the value $FF, so when programming this value, the user will have to wait for at least t_{WD_FLASH} before programming the next page. As a chip erased device contains $FF in all locations, programming of addresses that are meant to contain $FF, can be skipped. See Table 115 for t_{WD_FLASH} value

Data Polling EEPROM

When a new byte has been written and is being programmed into EEPROM, reading the address location being programmed will give the value $FF. At the time the device is ready for a new byte, the programmed value will read correctly. This is used to determine when the next byte can be written. This will not work for the value $FF, but the user should have the following in mind: As a chip erased device contains $FF in all locations, programming of addresses that are meant to contain $FF, can be skipped. This does not apply if the EEPROM is re-programmed without chip erasing the device. In this case, data polling cannot be used for the value $FF, and the user will have to wait at least t_{WD_EEPROM} before programming the next byte. See Table 115 for t_{WD_EEPROM} value.

Table 115. Minimum Wait Delay before Writing the Next Flash or EEPROM Location

Symbol	Minimum Wait Delay
t_{WD_FLASH}	4.5 ms
t_{WD_EEPROM}	9.0 ms
t_{WD_ERASE}	9.0 ms

Figure 137. SPI Serial Programming Waveforms

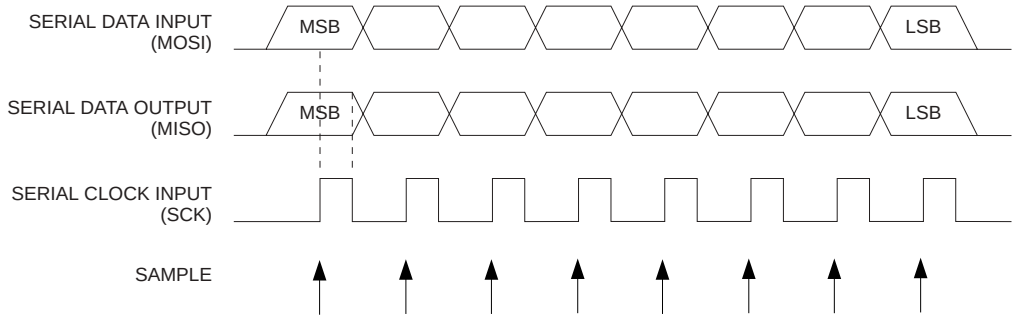


Table 116. SPI Serial Programming Instruction Set

Instruction	Instruction Format				Operation
	Byte 1	Byte 2	Byte 3	Byte4	
Programming Enable	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Enable SPI Serial Programming after $\overline{\text{RESET}}$ goes low.
Chip Erase	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Chip Erase EEPROM and Flash.
Read Program Memory	0010 **H**000	00**aa aaaa**	**bbbb bbbb**	**oooo oooo**	Read **H** (high or low) data **o** from Program memory at word address **a**:**b**.
Load Program Memory Page	0100 **H**000	00xx xxxx	xx**bb bbbb**	**iiii iiii**	Write **H** (high or low) data **i** to Program Memory page at word address **b**. Data low byte must be loaded before Data high byte is applied within the same address.
Write Program Memory Page	0100 1100	00**aa aaaa**	**bb**xx xxxx	xxxx xxxx	Write Program Memory Page at address **a**:**b**.
Read EEPROM Memory	1010 0000	00xx xx**aa**	**bbbb bbbb**	**oooo oooo**	Read data **o** from EEPROM memory at address **a**:**b**.
Write EEPROM Memory	1100 0000	00xx xx**aa**	**bbbb bbbb**	**iiii iiii**	Write data **i** to EEPROM memory at address **a**:**b**.
Read Lock Bits	0101 1000	0000 0000	xxxx xxxx	xx**oo oooo**	Read Lock bits. “0” = programmed, “1” = unprogrammed. See Table 103 on page 254 for details.
Write Lock Bits	1010 1100	111x xxxx	xxxx xxxx	11**ii iiii**	Write Lock bits. Set bits = “0” to program Lock bits. See Table 103 on page 254 for details.
Read Signature Byte	0011 0000	00xx xxxx	xxxx xx**bb**	**oooo oooo**	Read Signature Byte **o** at address **b**.
Write Fuse Bits	1010 1100	1010 0000	xxxx xxxx	**iiii iiii**	Set bits = “0” to program, “1” to unprogram. See Table 106 on page 256 for details.
Write Fuse High Bits	1010 1100	1010 1000	xxxx xxxx	**iiii iiii**	Set bits = “0” to program, “1” to unprogram. See Table 105 on page 255 for details.
Read Fuse Bits	0101 0000	0000 0000	xxxx xxxx	**oooo oooo**	Read Fuse bits. “0” = programmed, “1” = unprogrammed. See Table 106 on page 256 for details.
Read Fuse High Bits	0101 1000	0000 1000	xxxx xxxx	**oooo oooo**	Read Fuse high bits. “0” = programmed, “1” = unprogrammed. See Table 105 on page 255 for details.
Read Calibration Byte	0011 1000	00xx xxxx	0000 0000	**oooo oooo**	Read Calibration Byte

Note: **a** = address high bits
b = address low bits
H = 0 – Low byte, 1 – High Byte
o = data out
i = data in
x = don’t care

SPI Serial Programming Characteristics

For Characteristics of SPI module, see “SPI Timing Characteristics” on page 289.

Programming via the JTAG Interface

Programming through the JTAG interface requires control of the four JTAG specific pins: TCK, TMS, TDI and TDO. Control of the reset and clock pins is not required.

To be able to use the JTAG interface, the JTAGEN Fuse must be programmed. The device is default shipped with the fuse programmed. In addition, the JTD bit in MCUCSR must be cleared. Alternatively, if the JTD bit is set, the External Reset can be forced low. Then, the JTD bit will be cleared after two chip clocks, and the JTAG pins are available for programming. This provides a means of using the JTAG pins as normal port pins in running mode while still allowing In-System Programming via the JTAG interface. Note that this technique can not be used when using the JTAG pins for Boundary-scan or On-chip Debug. In these cases the JTAG pins must be dedicated for this purpose.

As a definition in this datasheet, the LSB is shifted in and out first of all Shift Registers.

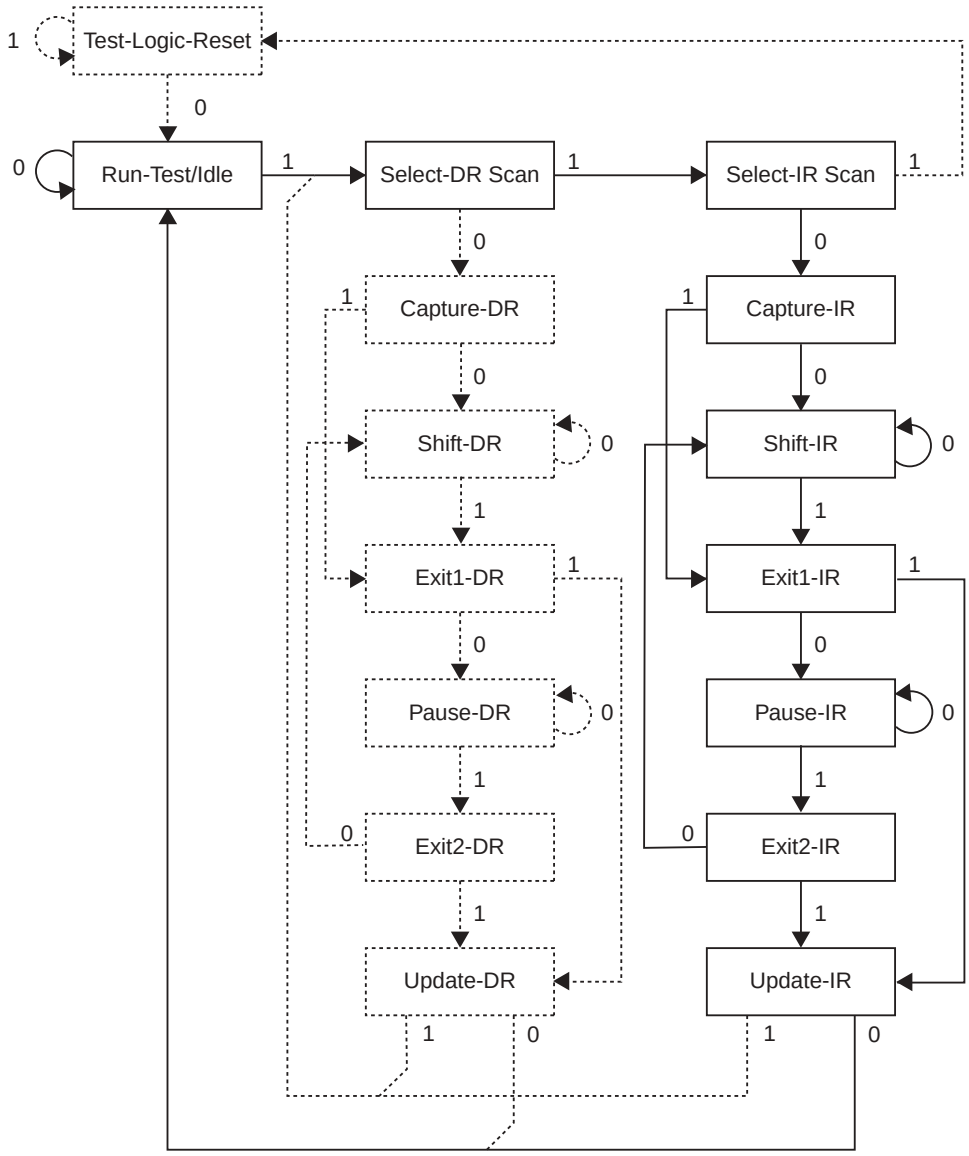
Programming Specific JTAG Instructions

The instruction register is 4-bit wide, supporting up to 16 instructions. The JTAG instructions useful for Programming are listed below.

The OPCODE for each instruction is shown behind the instruction name in hex format. The text describes which Data Register is selected as path between TDI and TDO for each instruction.

The Run-Test/Idle state of the TAP controller is used to generate internal clocks. It can also be used as an idle state between JTAG sequences. The state machine sequence for changing the instruction word is shown in Figure 138.

Figure 138. State Machine Sequence for Changing the Instruction Word



AVR_RESET ($C)

The AVR specific public JTAG instruction for setting the AVR device in the Reset mode or taking the device out from the Reset Mode. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as Data Register. Note that the Reset will be active as long as there is a logic “one” in the Reset Chain. The output from this chain is not latched.

The active states are:

- Shift-DR: The Reset Register is shifted by the TCK input.

PROG_ENABLE ($4)

The AVR specific public JTAG instruction for enabling programming via the JTAG port. The 16-bit Programming Enable Register is selected as Data Register. The active states are the following:

- Shift-DR: The programming enable signature is shifted into the Data Register.
- Update-DR: The programming enable signature is compared to the correct value, and Programming mode is entered if the signature is valid.

PROG_COMMANDS ($5)

The AVR specific public JTAG instruction for entering programming commands via the JTAG port. The 15-bit Programming Command Register is selected as Data Register. The active states are the following:

- Capture-DR: The result of the previous command is loaded into the Data Register.
- Shift-DR: The Data Register is shifted by the TCK input, shifting out the result of the previous command and shifting in the new command.
- Update-DR: The programming command is applied to the Flash inputs
- Run-Test/Idle: One clock cycle is generated, executing the applied command (not always required, see Table 117 below).

PROG_PAGELOAD ($6)

The AVR specific public JTAG instruction to directly load the Flash data page via the JTAG port. The 1024 bit Virtual Flash Page Load Register is selected as Data Register. This is a virtual scan chain with length equal to the number of bits in one Flash page. Internally the Shift Register is 8-bit. Unlike most JTAG instructions, the Update-DR state is not used to transfer data from the Shift Register. The data are automatically transferred to the Flash page buffer byte by byte in the Shift-DR state by an internal state machine. This is the only active state:

- Shift-DR: Flash page data are shifted in from TDI by the TCK input, and automatically loaded into the Flash page one byte at a time.

Note: The JTAG instruction PROG_PAGELOAD can only be used if the AVR device is the first device in JTAG scan chain. If the AVR cannot be the first device in the scan chain, the byte-wise programming algorithm must be used.

PROG_PAGEREAD ($7)

The AVR specific public JTAG instruction to read one full Flash data page via the JTAG port. The 1032 bit Virtual Flash Page Read Register is selected as Data Register. This is a virtual scan chain with length equal to the number of bits in one Flash page plus 8. Internally the Shift Register is 8-bit. Unlike most JTAG instructions, the Capture-DR state is not used to transfer data to the Shift Register. The data are automatically transferred from the Flash page buffer byte by byte in the Shift-DR state by an internal state machine. This is the only active state:

- Shift-DR: Flash data are automatically read one byte at a time and shifted out on TDO by the TCK input. The TDI input is ignored.

Note: The JTAG instruction PROG_PAGEREAD can only be used if the AVR device is the first device in JTAG scan chain. If the AVR cannot be the first device in the scan chain, the byte-wise programming algorithm must be used.

Data Registers

The Data Registers are selected by the JTAG Instruction Registers described in section “Programming Specific JTAG Instructions” on page 272. The Data Registers relevant for programming operations are:

- Reset Register
- Programming Enable Register
- Programming Command Register
- Virtual Flash Page Load Register
- Virtual Flash Page Read Register

Reset Register

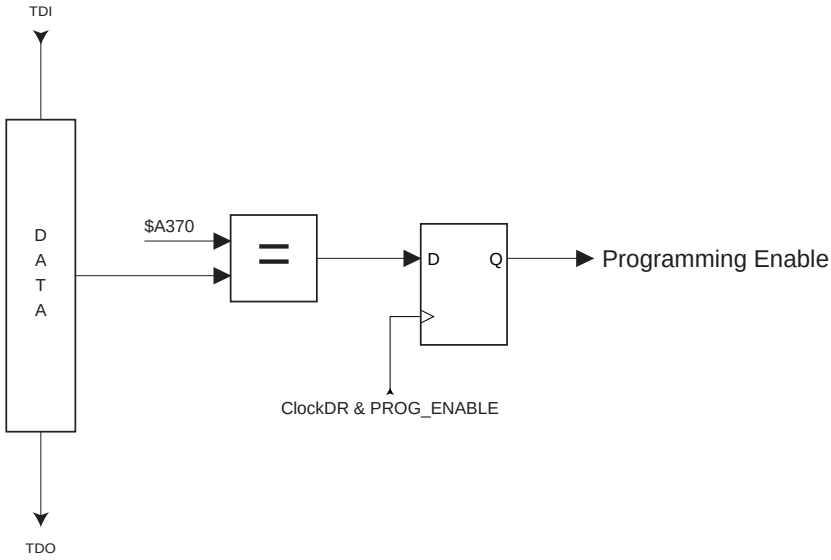
The Reset Register is a Test Data Register used to reset the part during programming. It is required to reset the part before entering programming mode.

A high value in the Reset Register corresponds to pulling the external Reset low. The part is reset as long as there is a high value present in the Reset Register. Depending on the Fuse settings for the clock options, the part will remain reset for a Reset Time-out Period (refer to “Clock Sources” on page 23) after releasing the Reset Register. The output from this Data Register is not latched, so the reset will take place immediately, as shown in Figure 115 on page 225.

Programming Enable Register

The Programming Enable Register is a 16-bit register. The contents of this register is compared to the programming enable signature, binary code 1010_0011_0111_0000. When the contents of the register is equal to the programming enable signature, programming via the JTAG port is enabled. The register is reset to 0 on Power-on Reset, and should always be reset when leaving Programming mode.

Figure 139. Programming Enable Register



Programming Command Register

The Programming Command Register is a 15-bit register. This register is used to serially shift in programming commands, and to serially shift out the result of the previous command, if any. The JTAG Programming Instruction Set is shown in Table 117. The state sequence when shifting in the programming commands is illustrated in Figure 141.

Figure 140. Programming Command Register

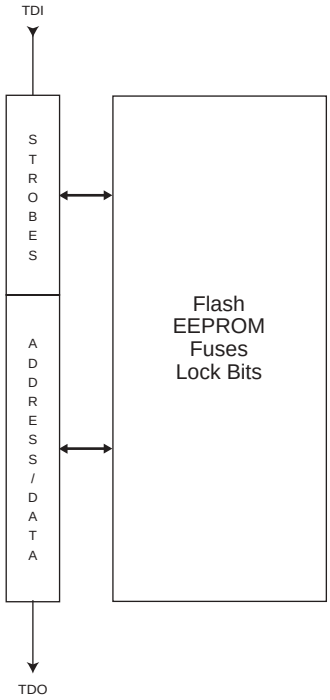


Table 117. JTAG Programming Instruction Set

a = address high bits, **b** = address low bits, **H** = 0 – Low byte, 1 – High Byte, **o** = data out, **i** = data in, x = don't care

Instruction	TDI sequence	TDO sequence	Notes
1a. Chip erase	0100011_10000000 0110001_10000000 0110011_10000000 0110011_10000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	
1b. Poll for chip erase complete	0110011_10000000	xxxxx**o**x_xxxxxxxx	(2)
2a. Enter Flash Write	0100011_00010000	xxxxxxx_xxxxxxxx	
2b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
2c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
2d. Load Data Low Byte	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	
2e. Load Data High Byte	0010111_**iiiiiiii**	xxxxxxx_xxxxxxxx	
2f. Latch Data	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2g. Write Flash Page	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2h. Poll for Page Write complete	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
3a. Enter Flash Read	0100011_00000010	xxxxxxx_xxxxxxxx	
3b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
3c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
3d. Read Data Low and High Byte	0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo** xxxxxxx_**oooooooo**	 low byte high byte
4a. Enter EEPROM Write	0100011_00010001	xxxxxxx_xxxxxxxx	
4b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
4c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
4d. Load Data Byte	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	
4e. Latch Data	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4f. Write EEPROM Page	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4g. Poll for Page Write complete	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
5a. Enter EEPROM Read	0100011_00000011	xxxxxxx_xxxxxxxx	
5b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)

Table 117. JTAG Programming Instruction Set (Continued)

a = address high bits, **b** = address low bits, **H** = 0 – Low byte, 1 – High Byte, **o** = data out, **i** = data in, x = don't care

Instruction	TDI sequence	TDO sequence	Notes
5c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
5d. Read Data Byte	0110011_**bbbbbbbb** 0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
6a. Enter Fuse Write	0100011_01000000	xxxxxxx_xxxxxxxx	
6b. Load Data Low Byte(6)	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6c. Write Fuse High byte	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6d. Poll for Fuse Write complete	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
6e. Load Data Low Byte(7)	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6f. Write Fuse Low byte	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6g. Poll for Fuse Write complete	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
7a. Enter Lock Bit Write	0100011_00100000	xxxxxxx_xxxxxxxx	
7b. Load Data Byte(8)	0010011_11**iiiiii**	xxxxxxx_xxxxxxxx	(4)
7c. Write Lock Bits	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
7d. Poll for Lock Bit Write complete	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
8a. Enter Fuse/Lock Bit Read	0100011_00000100	xxxxxxx_xxxxxxxx	
8b. Read Fuse High Byte(6)	0111110_00000000 0111111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8c. Read Fuse Low Byte(7)	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8d. Read Lock Bits(8)	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xx**oooooo**	(5)
8e. Read Fuses and Lock Bits	0111110_00000000 0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo** xxxxxxx_**oooooooo** xxxxxxx_**oooooooo**	(5) fuse high byte fuse low byte lock bits
9a. Enter Signature Byte Read	0100011_00001000	xxxxxxx_xxxxxxxx	
9b. Load Address Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
9c. Read Signature Byte	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	

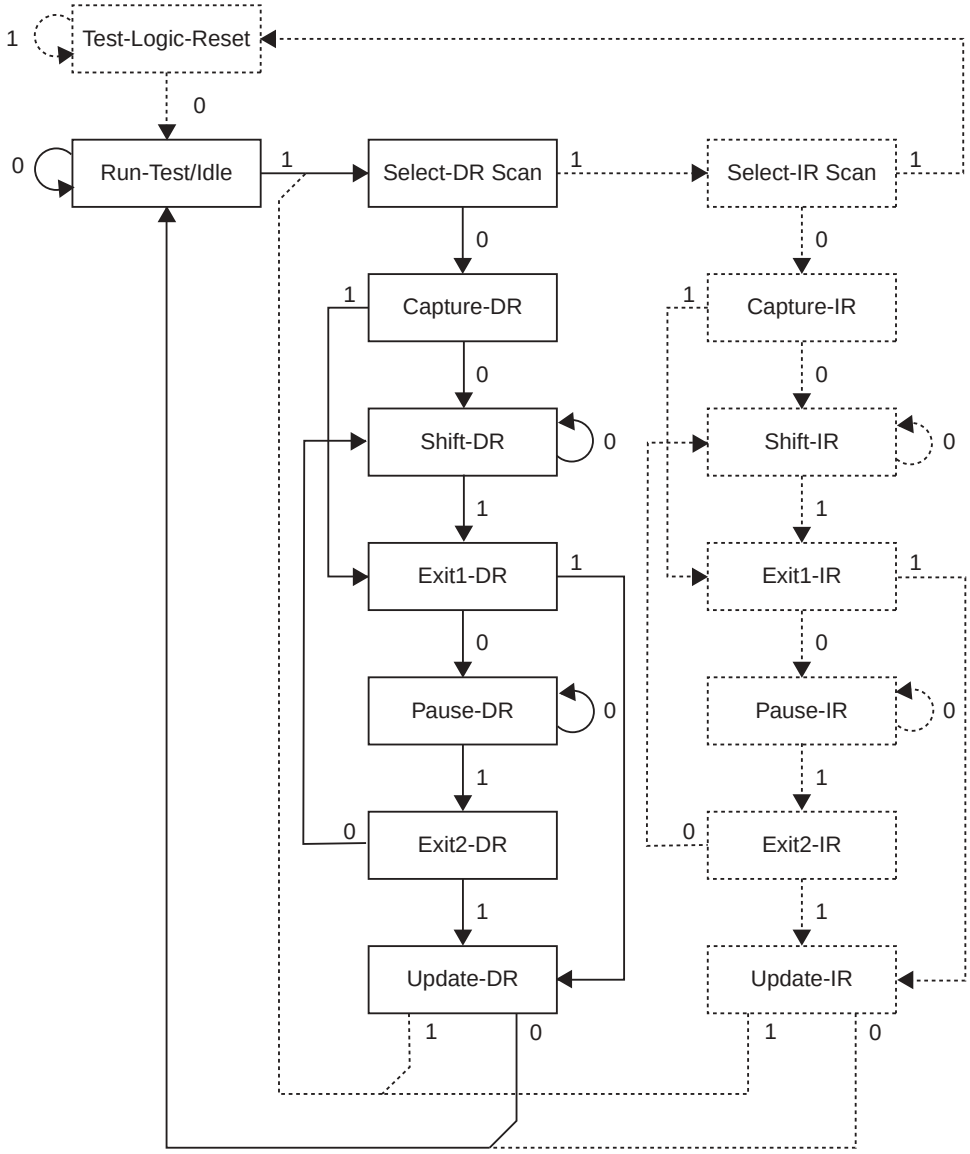
Table 117. JTAG Programming Instruction Set (Continued)

a = address high bits, **b** = address low bits, **H** = 0 – Low byte, 1 – High Byte, **o** = data out, **i** = data in, x = don't care

Instruction	TDI sequence	TDO sequence	Notes
10a. Enter Calibration Byte Read	0100011_00001000	xxxxxxx_xxxxxxxx	
10b. Load Address Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
10c. Read Calibration Byte	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
11a. Load No Operation Command	0100011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	

Notes:
1. This command sequence is not required if the seven MSB are correctly set by the previous command sequence (which is normally the case).
2. Repeat until **o** = "1".
3. Set bits to "0" to program the corresponding fuse, "1" to unprogram the fuse.
4. Set bits to "0" to program the corresponding lock bit, "1" to leave the lock bit unchanged.
5. "0" = programmed, "1" = unprogrammed.
6. The bit mapping for fuses high byte is listed in Table 105 on page 255
7. The bit mapping for fuses low byte is listed in Table 106 on page 256
8. The bit mapping for Lock bits byte is listed in Table 103 on page 254
9. Address bits exceeding PCMSB and EEAMSB (Table 111 and Table 112) are don't care

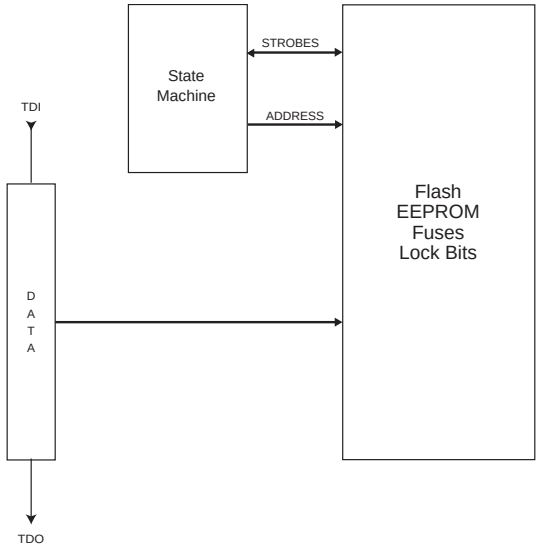
Figure 141. State Machine Sequence for Changing/Reading the Data Word



Virtual Flash Page Load Register

The Virtual Flash Page Load Register is a virtual scan chain with length equal to the number of bits in one Flash page. Internally the Shift Register is 8-bit, and the data are automatically transferred to the Flash page buffer byte by byte. Shift in all instruction words in the page, starting with the LSB of the first instruction in the page and ending with the MSB of the last instruction in the page. This provides an efficient way to load the entire Flash page buffer before executing Page Write.

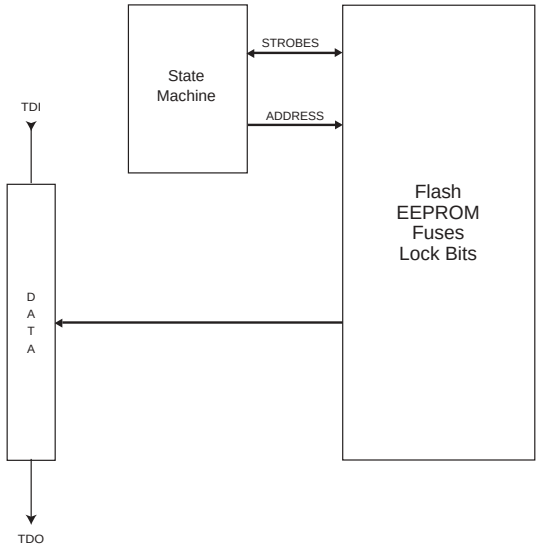
Figure 142. Virtual Flash Page Load Register



Virtual Flash Page Read Register

The Virtual Flash Page Read Register is a virtual scan chain with length equal to the number of bits in one Flash page plus 8. Internally the Shift Register is 8-bit, and the data are automatically transferred from the Flash data page byte by byte. The first 8 cycles are used to transfer the first byte to the internal Shift Register, and the bits that are shifted out during these 8 cycles should be ignored. Following this initialization, data are shifted out starting with the LSB of the first instruction in the page and ending with the MSB of the last instruction in the page. This provides an efficient way to read one full Flash page to verify programming.

Figure 143. Virtual Flash Page Read Register



Programming Algorithm

All references below of type “1a”, “1b”, and so on, refer to Table 117.

Entering Programming Mode

1. Enter JTAG instruction AVR_RESET and shift 1 in the Reset Register.
2. Enter instruction PROG_ENABLE and shift 1010_0011_0111_0000 in the Programming Enable Register.

Leaving Programming Mode

1. Enter JTAG instruction PROG_COMMANDS.
2. Disable all programming instructions by usning no operation instruction 11a.
3. Enter instruction PROG_ENABLE and shift 0000_0000_0000_0000 in the programming Enable Register.
4. Enter JTAG instruction AVR_RESET and shift 0 in the Reset Register.

Performing Chip Erase

1. Enter JTAG instruction PROG_COMMANDS.
2. Start chip erase using programming instruction 1a.
3. Poll for Chip Erase complete using programming instruction 1b, or wait for t_{WLRH_CE} (refer to Table 113 on page 267).

Programming the Flash

Before programming the Flash a Chip Erase must be performed. See “Performing Chip Erase” on page 282.

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load address high byte using programming instruction 2b.
4. Load address low byte using programming instruction 2c.
5. Load data using programming instructions 2d, 2e and 2f.
6. Repeat steps 4 and 5 for all instruction words in the page.
7. Write the page using programming instruction 2g.
8. Poll for Flash write complete using programming instruction 2h, or wait for t_{WLRH} (refer to Table 113 on page 267).
9. Repeat steps 3 to 7 until all data have been programmed.

A more efficient data transfer can be achieved using the PROG_PAGELOAD instruction:

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load the page address using programming instructions 2b and 2c. PCWORD (refer to Table 111 on page 258) is used to address within one page and must be written as 0.
4. Enter JTAG instruction PROG_PAGELOAD.
5. Load the entire page by shifting in all instruction words in the page, starting with the LSB of the first instruction in the page and ending with the MSB of the last instruction in the page.
6. Enter JTAG instruction PROG_COMMANDS.
7. Write the page using programming instruction 2g.
8. Poll for Flash write complete using programming instruction 2h, or wait for t_{WLRH} (refer to Table 113 on page 267).
9. Repeat steps 3 to 8 until all data have been programmed.

Reading the Flash

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash read using programming instruction 3a.
3. Load address using programming instructions 3b and 3c.
4. Read data using programming instruction 3d.
5. Repeat steps 3 and 4 until all data have been read.

A more efficient data transfer can be achieved using the PROG_PAGEREAD instruction:

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash read using programming instruction 3a.
3. Load the page address using programming instructions 3b and 3c. PCWORD (refer to Table 111 on page 258) is used to address within one page and must be written as 0.
4. Enter JTAG instruction PROG_PAGEREAD.
5. Read the entire page by shifting out all instruction words in the page, starting with the LSB of the first instruction in the page and ending with the MSB of the last instruction in the page. Remember that the first 8 bits shifted out should be ignored.
6. Enter JTAG instruction PROG_COMMANDS.
7. Repeat steps 3 to 6 until all data have been read.

Programming the EEPROM

Before programming the EEPROM a Chip Erase must be performed. See “Performing Chip Erase” on page 282.

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM write using programming instruction 4a.
3. Load address high byte using programming instruction 4b.
4. Load address low byte using programming instruction 4c.
5. Load data using programming instructions 4d and 4e.
6. Repeat steps 4 and 5 for all data bytes in the page.
7. Write the data using programming instruction 4f.
8. Poll for EEPROM write complete using programming instruction 4g, or wait for t_{WLRH} (refer to Table 113 on page 267).
9. Repeat steps 3 to 8 until all data have been programmed.

Note that the PROG_PAGELOAD instruction can not be used when programming the EEPROM

Reading the EEPROM

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM read using programming instruction 5a.
3. Load address using programming instructions 5b and 5c.
4. Read data using programming instruction 5d.
5. Repeat steps 3 and 4 until all data have been read.

Note that the PROG_PAGEREAD instruction can not be used when reading the EEPROM

Programming the Fuses

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse write using programming instruction 6a.
3. Load data high byte using programming instructions 6b. A bit value of “0” will program the corresponding fuse, a “1” will unprogram the fuse.
4. Write Fuse High byte using programming instruction 6c.
5. Poll for Fuse write complete using programming instruction 6d, or wait for t_{WLRH} (refer to Table 113 on page 267).
6. Load data low byte using programming instructions 6e. A “0” will program the fuse, a “1” will unprogram the fuse.
7. Write Fuse low byte using programming instruction 6f.
8. Poll for Fuse write complete using programming instruction 6g, or wait for t_{WLRH} (refer to Table 113 on page 267).

Programming the Lock Bits

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Lock bit write using programming instruction 7a.
3. Load data using programming instructions 7b. A bit value of “0” will program the corresponding Lock bit, a “1” will leave the Lock bit unchanged.
4. Write Lock bits using programming instruction 7c.
5. Poll for Lock bit write complete using programming instruction 7d, or wait for t_{WLRH} (refer to Table 113 on page 267).

Reading the Fuses and Lock Bits

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse/Lock bit read using programming instruction 8a.
3. To read all Fuses and Lock bits, use programming instruction 8e.
   To only read Fuse high byte, use programming instruction 8b.
   To only read Fuse low byte, use programming instruction 8c.
   To only read Lock bits, use programming instruction 8d.

Reading the Signature Bytes

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Signature byte read using programming instruction 9a.
3. Load address $00 using programming instruction 9b.
4. Read first signature byte using programming instruction 9c.
5. Repeat steps 3 and 4 with address $01 and address $02 to read the second and third signature bytes, respectively.

Reading the Calibration Byte

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Calibration byte read using programming instruction 10a.
3. Load address $00 using programming instruction 10b.
4. Read the calibration byte using programming instruction 10c.

Electrical Characteristics

Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA

*NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 2.7V to 5.5V (Unless Otherwise Noted)

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{IL}	Input Low Voltage	Except XTAL1 pin	-0.5		0.2 V_{CC}(1)	V
V_{IL1}	Input Low Voltage	XTAL1 pin, External Clock Selected	-0.5		0.1 V_{CC}(1)	V
V_{IH}	Input High Voltage	Except XTAL1 and $\overline{\text{RESET}}$ pins	0.6 V_{CC}(2)		V_{CC} + 0.5	V
V_{IH1}	Input High Voltage	XTAL1 pin, External Clock Selected	0.7 V_{CC}(2)		V_{CC} + 0.5	V
V_{IH2}	Input High Voltage	$\overline{\text{RESET}}$ pin	0.9 V_{CC}(2)		V_{CC} + 0.5	V
V_{OL}	Output Low Voltage(3) (Ports A,B,C,D)	I_{OL} = 20 mA, V_{CC} = 5V I_{OL} = 10 mA, V_{CC} = 3V			0.7 0.5	V V
V_{OH}	Output High Voltage(4) (Ports A,B,C,D)	I_{OH} = -20 mA, V_{CC} = 5V I_{OH} = -10 mA, V_{CC} = 3V	4.0 2.2			V V
I_{IL}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin low (absolute value)			1	µA
I_{IH}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin high (absolute value)			1	µA
R_{RST}	Reset Pull-up Resistor		30		60	kΩ
R_{pu}	I/O Pin Pull-up Resistor		20		50	kΩ

T_A = -40°C to 85°C, V_{CC} = 2.7V to 5.5V (Unless Otherwise Noted)

Symbol	Parameter	Condition	Min	Typ	Max	Units
I_{CC}	Power Supply Current	Active 1 MHz, V_{CC} = 3V (ATmega32L)		1.1		mA
		Active 4 MHz, V_{CC} = 3V (ATmega32L)		3.8	5	mA
		Active 8 MHz, V_{CC} = 5V (ATmega32)		12	15	mA
		Idle 1 MHz, V_{CC} = 3V (ATmega32L)		0.35		mA
		Idle 4 MHz, V_{CC} = 3V (ATmega32L)		1.2	2.5	mA
		Idle 8 MHz, V_{CC} = 5V (ATmega32)		5.5	8	mA
	Power-down Mode(5)	WDT enabled, V_{CC} = 3V		< 25	20	µA
		WDT disabled, V_{CC} = 3V		< 1	10	µA
V_{ACIO}	Analog Comparator Input Offset Voltage	V_{CC} = 5V V_{in} = V_{CC}/2			40	mV
I_{ACLK}	Analog Comparator Input Leakage Current	V_{CC} = 5V V_{in} = V_{CC}/2	-50		50	nA
t_{ACID}	Analog Comparator Propagation Delay	V_{CC} = 2.7V V_{CC} = 4.0V		750 500		ns

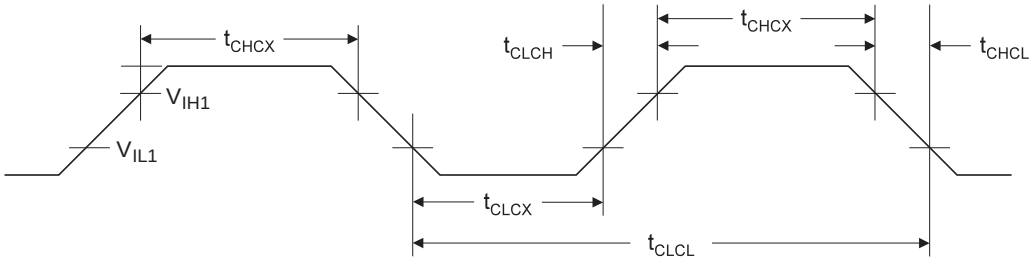
Notes:
1. “Max” means the highest value where the pin is guaranteed to be read as low
2. “Min” means the lowest value where the pin is guaranteed to be read as high
3. Although each I/O port can sink more than the test conditions (20 mA at Vcc = 5V, 10 mA at Vcc = 3V) under steady state conditions (non-transient), the following must be observed:
   PDIP Package:
   1] The sum of all IOL, for all ports, should not exceed 400 mA.
   2] The sum of all IOL, for port A0 - A7, should not exceed 200 mA.
   3] The sum of all IOL, for ports B0 - B7,C0 - C7, D0 - D7 and XTAL2, should not exceed 300 mA.
   TQFP and MLF Package:
   1] The sum of all IOL, for all ports, should not exceed 400 mA.
   2] The sum of all IOL, for ports A0 - A7, should not exceed 200 mA.
   3] The sum of all IOL, for ports B0 - B4, should not exceed 200 mA.
   4] The sum of all IOL, for ports B3 - B7, XTAL2, D0 - D2, should not exceed 200 mA.
   5] The sum of all IOL, for ports D3 - D7, should not exceed 200 mA.
   6] The sum of all IOL, for ports C0 - C7, should not exceed 200 mA.
   If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
4. Although each I/O port can source more than the test conditions (20 mA at Vcc = 5V, 10 mA at Vcc = 3V) under steady state conditions (non-transient), the following must be observed:
   PDIP Package:
   1] The sum of all IOH, for all ports, should not exceed 400 mA.
   2] The sum of all IOH, for port A0 - A7, should not exceed 200 mA.
   3] The sum of all IOH, for ports B0 - B7,C0 - C7, D0 - D7 and XTAL2, should not exceed 300 mA.
   TQFP and MLF Package:
   1] The sum of all IOH, for all ports, should not exceed 400 mA.
   2] The sum of all IOH, for ports A0 - A7, should not exceed 200 mA.
   3] The sum of all IOH, for ports B0 - B4, should not exceed 200 mA.
   4] The sum of all IOH, for ports B3 - B7, XTAL2, D0 - D2, should not exceed 200 mA.
   5] The sum of all IOH, for ports D3 - D7, should not exceed 200 mA.

6] The sum of all IOH, for ports C0 - C7, should not exceed 200 mA.If IOH exceeds the test condition, VOH may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.

5. Minimum V_{CC} for Power-down is 2.5V.

External Clock Drive Waveforms

Figure 144. External Clock Drive Waveforms



External Clock Drive

Table 118. External Clock Drive

Symbol	Parameter	V_{CC} = 2.7V to 5.5V		V_{CC} = 4.5V to 5.5V		Units
		Min	Max	Min	Max	
$1/t_{CLCL}$	Oscillator Frequency	0	8	0	16	MHz
t_{CLCL}	Clock Period	125		62.5		ns
t_{CHCX}	High Time	50		25		ns
t_{CLCX}	Low Time	50		25		ns
t_{CLCH}	Rise Time		1.6		0.5	µs
t_{CHCL}	Fall Time		1.6		0.5	µs
Δt_{CLCL}	Change in period from one clock cycle to the next		2		2	%

Table 119. External RC Oscillator, Typical Frequencies (V_{CC} = 5V)

R [kΩ](1)	C [pF]	f(2)
100	47	87 kHz
33	22	650 kHz
10	22	2.0 MHz

Notes: 1. R should be in the range 3 kΩ - 100 kΩ, and C should be at least 20 pF. The C values given in the table includes pin capacitance. This will vary with package type.
2. The frequency will vary with package type and board layout.

Two-wire Serial Interface Characteristics

Table 120 describes the requirements for devices connected to the Two-wire Serial Bus. The ATmega32 Two-wire Serial Interface meets or exceeds these requirements under the noted conditions.

Timing symbols refer to Figure 145.

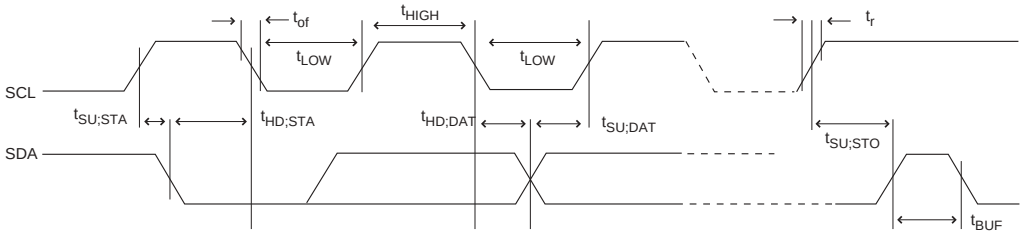
Table 120. Two-wire Serial Bus Requirements

Symbol	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low-voltage		-0.5	0.3 V_{CC}	V
V_{IH}	Input High-voltage		0.7 V_{CC}	V_{CC} + 0.5	V
V_{hys}(1)	Hysteresis of Schmitt Trigger Inputs		0.05 V_{CC}(2)	–	V
V_{OL}(1)	Output Low-voltage	3 mA sink current	0	0.4	V
t_r(1)	Rise Time for both SDA and SCL		20 + 0.1C_b(3)(2)	300	ns
t_{of}(1)	Output Fall Time from V_{IHmin} to V_{ILmax}	10 pF < C_b < 400 pF(3)	20 + 0.1C_b(3)(2)	250	ns
t_{SP}(1)	Spikes Suppressed by Input Filter		0	50(2)	ns
I_i	Input Current each I/O Pin	0.1V_{CC} < V_i < 0.9V_{CC}	-10	10	µA
C_i(1)	Capacitance for each I/O Pin		–	10	pF
f_{SCL}	SCL Clock Frequency	f_{CK}(4) > max(16f_{SCL}, 250kHz)(5)	0	400	kHz
Rp	Value of Pull-up resistor	f_{SCL} ≤ 100 kHz	$\frac{V_{CC} - 0,4V}{3mA}$	$\frac{1000ns}{C_b}$	Ω
		f_{SCL} > 100 kHz	$\frac{V_{CC} - 0,4V}{3mA}$	$\frac{300ns}{C_b}$	Ω
$t_{HD;STA}$	Hold Time (repeated) START Condition	f_{SCL} ≤ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
t_{LOW}	Low Period of the SCL Clock	f_{SCL} ≤ 100 kHz(6)	4.7	–	µs
		f_{SCL} > 100 kHz(7)	1.3	–	µs
t_{HIGH}	High period of the SCL clock	f_{SCL} ≤ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
$t_{SU;STA}$	Set-up time for a repeated START condition	f_{SCL} ≤ 100 kHz	4.7	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
$t_{HD;DAT}$	Data hold time	f_{SCL} ≤ 100 kHz	0	3.45	µs
		f_{SCL} > 100 kHz	0	0.9	µs
$t_{SU;DAT}$	Data setup time	f_{SCL} ≤ 100 kHz	250	–	ns
		f_{SCL} > 100 kHz	100	–	ns
$t_{SU;STO}$	Setup time for STOP condition	f_{SCL} ≤ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
t_{BUF}	Bus free time between a STOP and START condition	f_{SCL} ≤ 100 kHz	4.7	–	µs
		f_{SCL} > 100 kHz	1.3	–	µs

Notes:
1. In ATmega32, this parameter is characterized and not 100% tested.
2. Required only for f_{SCL} > 100 kHz.
3. C_b = capacitance of one bus line in pF.
4. f_{CK} = CPU clock frequency

5. This requirement applies to all ATmega32 Two-wire Serial Interface operation. Other devices connected to the Two-wire Serial Bus need only obey the general f_{SCL} requirement.
6. The actual low period generated by the ATmega32 Two-wire Serial Interface is ($1/f_{SCL}$ - $2/f_{CK}$), thus f_{CK} must be greater than 6 MHz for the low time requirement to be strictly met at f_{SCL} = 100 kHz.
7. The actual low period generated by the ATmega32 Two-wire Serial Interface is ($1/f_{SCL}$ - $2/f_{CK}$), thus the low time requirement will not be strictly met for f_{SCL} > 308 kHz when f_{CK} = 8 MHz. Still, ATmega32 devices connected to the bus may communicate at full speed (400 kHz) with other ATmega32 devices, as well as any other device with a proper t_{LOW} acceptance margin.

Figure 145. Two-wire Serial Bus Timing



SPI Timing Characteristics

See Figure 146 and Figure 147 for details.

Table 121. SPI Timing Parameters

	Description	Mode	Min	Typ	Max	
1	SCK period	Master		See Table 58		
2	SCK high/low	Master		50% duty cycle		
3	Rise/Fall time	Master		3.6		
4	Setup	Master		10		
5	Hold	Master		10		
6	Out to SCK	Master		0.5 • t_{SCK}		ns
7	SCK to out	Master		10		
8	SCK to out high	Master		10		
9	$\overline{SS}$ low to out	Slave		15		
10	SCK period	Slave	4 • t_{ck}			
11	SCK high/low	Slave	2 • t_{ck}			
12	Rise/Fall time	Slave			1.6	µs
13	Setup	Slave	10			
14	Hold	Slave	t_{ck}			
15	SCK to out	Slave		15		ns
16	SCK to $\overline{SS}$ high	Slave	20			
17	$\overline{SS}$ high to tri-state	Slave		10		
18	$\overline{SS}$ low to SCK	Salve	2 • t_{ck}			

Figure 146. SPI Interface Timing Requirements (Master Mode)

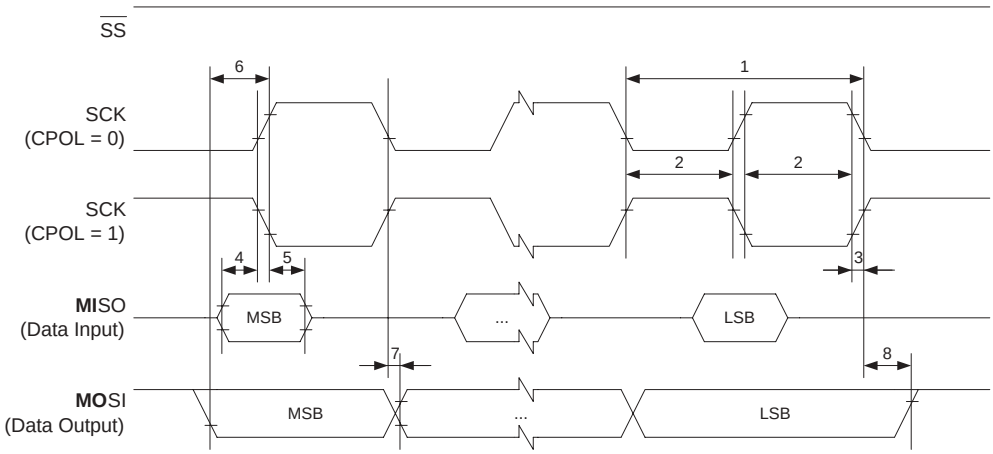
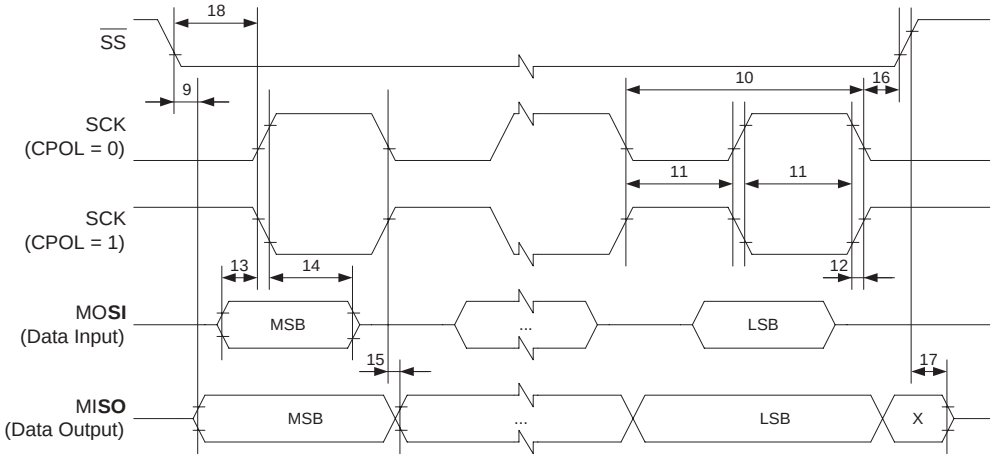


Figure 147. SPI Interface Timing Requirements (Slave Mode)



ADC Characteristics

Table 122. ADC Characteristics, Single Ended channels, T_A = -40°C to 85°C

Symbol	Parameter	Condition	Min	Typ	Max	Units
	Resolution	Single Ended Conversion		10		Bits
	Absolute Accuracy (Including INL, DNL, Quantization Error, Gain, and Offset Error)	Single Ended Conversion V_{REF} = 4V, V_{CC} = 4V ADC clock = 200 kHz		1.5		LSB
		Single Ended Conversion V_{REF} = 4V, V_{CC} = 4V ADC clock = 1 MHz		3		LSB
		Single Ended Conversion V_{REF} = 4V, V_{CC} = 4V ADC clock = 200 kHz Noise Reduction mode		1.5		LSB
		Single Ended Conversion V_{REF} = 4V, V_{CC} = 4V ADC clock = 1 MHz Noise Reduction mode		3		LSB
	Integral Non-Linearity (INL)	Single Ended Conversion V_{REF} = 4V, V_{CC} = 4V ADC clock = 200 kHz		0.75		LSB
	Differential Non-linearity (DNL)	Single Ended Conversion V_{REF} = 4V, V_{CC} = 4V ADC clock = 200 kHz		0.25		LSB
	Gain Error	Single Ended Conversion V_{REF} = 4V, V_{CC} = 4V ADC clock = 200 kHz		0.75		LSB
	Offset Error	Single Ended Conversion V_{REF} = 4V, V_{CC} = 4V ADC clock = 200 kHz		0.75		LSB
	Clock Frequency		50		1000	kHz
	Conversion Time		13		260	µs
AVCC	Analog Supply Voltage		V_{CC} - 0.3(1)		V_{CC} + 0.3(2)	V
V_{REF}	Reference Voltage		2.0		AVCC	V
V_{IN}	Input voltage		GND		V_{REF}	V
	ADC conversion output		0		1023	LSB
	Input bandwith			38.5		kHz
V_{INT}	Internal Voltage Reference		2.3	2.56	2.7	V
R_{REF}	Reference Input Resistance			32		kΩ
R_{AIN}	Analog Input Resistance			100		MΩ

Notes: 1. Minimum for AVCC is 2.7V.
2. Maximum for AVCC is 5.5V.

Table 123. ADC Characteristics, Differential channels, T_A = -40°C to 85°C

Symbol	Parameter	Condition	Min	Typ	Max	Units
	Resolution	Gain = 1x			10	Bits
		Gain = 10x			10	Bits
		Gain = 200x			10	Bits
	Absolute Accuracy	Gain = 1x V_{REF} = 4V, V_{CC} = 5V ADC clock = 50 - 200 kHz		17		LSB
		Gain = 10x V_{REF} = 4V, V_{CC} = 5V ADC clock = 50 - 200 kHz		16		LSB
		Gain = 200x V_{REF} = 4V, V_{CC} = 5V ADC clock = 50 - 200 kHz		7		LSB
	Integral Non-Linearity (INL) (Accuracy after calibration for Offset and Gain Error)	Gain = 1x V_{REF} = 4V, V_{CC} = 5V ADC clock = 50 - 200 kHz		0.75		LSB
		Gain = 10x V_{REF} = 4V, V_{CC} = 5V ADC clock = 50 - 200 kHz		0.75		LSB
		Gain = 200x V_{REF} = 4V, V_{CC} = 5V ADC clock = 50 - 200 kHz		2		LSB
	Gain Error	Gain = 1x		1.6		%
		Gain = 10x		1.5		%
		Gain = 200x		0.2		%
	Offset Error	Gain = 1x V_{REF} = 4V, V_{CC} = 5V ADC clock = 50 - 200 kHz		1		LSB
		Gain = 10x V_{REF} = 4V, V_{CC} = 5V ADC clock = 50 - 200 kHz		1.5		LSB
		Gain = 200x V_{REF} = 4V, V_{CC} = 5V ADC clock = 50 - 200 kHz		4.5		LSB
	Clock Frequency		50		200	kHz
	Conversion Time		65		260	µs
AVCC	Analog Supply Voltage		V_{CC} - 0.3(1)		V_{CC} + 0.3(2)	V
V_{REF}	Reference Voltage		2.0		AVCC - 0.5	V
V_{IN}	Input voltage		GND		V_{CC}	V
V_{DIFF}	Input differential voltage		-V_{REF}/Gain		V_{REF}/Gain/	V
	ADC conversion output		-511		511	LSB
	Input bandwith			4		kHz

Table 123. ADC Characteristics, Differential channels, T_A = -40°C to 85°C (Continued)

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{INT}	Internal Voltage Reference		2.3	2.56	2.7	V
R_{REF}	Reference Input Resistance			32		kΩ
R_{AIN}	Analog Input Resistance			100		MΩ

Notes: 1. Minimum for AVCC is 2.7V.
2. Maximum for AVCC is 5.5V.

5.

ATmega32 Typical Characteristics – Preliminary Data

The following charts show typical behavior. These figures are not tested during manufacturing. All current consumption measurements are performed with all I/O pins configured as inputs and with internal pull-ups enabled. A sine wave generator with rail-to-rail output is used as clock source.

The power consumption in Power-down mode is independent of clock selection.

The current consumption is a function of several factors such as: operating voltage, operating frequency, loading of I/O pins, switching rate of I/O pins, code executed and ambient temperature. The dominating factors are operating voltage and frequency.

The current drawn from capacitive loaded pins may be estimated (for one pin) as C_L*V_{CC}*f where C_L = load capacitance, V_{CC} = operating voltage and f = average switching frequency of I/O pin.

The parts are characterized at frequencies higher than test limits. Parts are not guaranteed to function properly at frequencies higher than the ordering code indicates.

The difference between current consumption in Power-down mode with Watchdog Timer enabled and Power-down mode with Watchdog Timer disabled represents the differential current drawn by the Watchdog Timer.

Figure 148. RC Oscillator Frequency vs. Temperature (the devices are calibrated to 1 MHz at Vcc = 5V, T=25c)

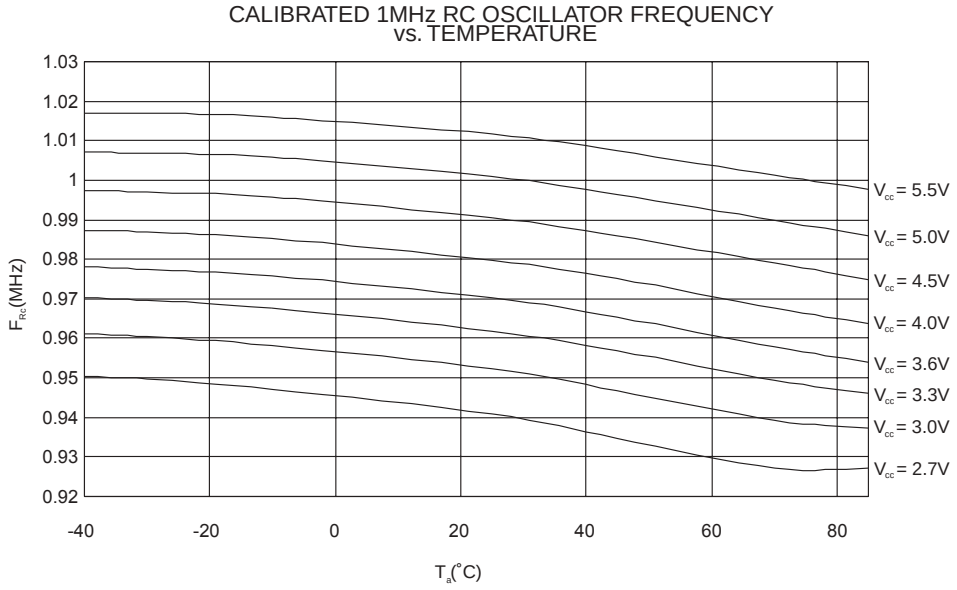


Figure 149. RC Oscillator Frequency vs. Operating Voltage (the devices are calibrated to 1 MHz at Vcc = 5V, T=25c)

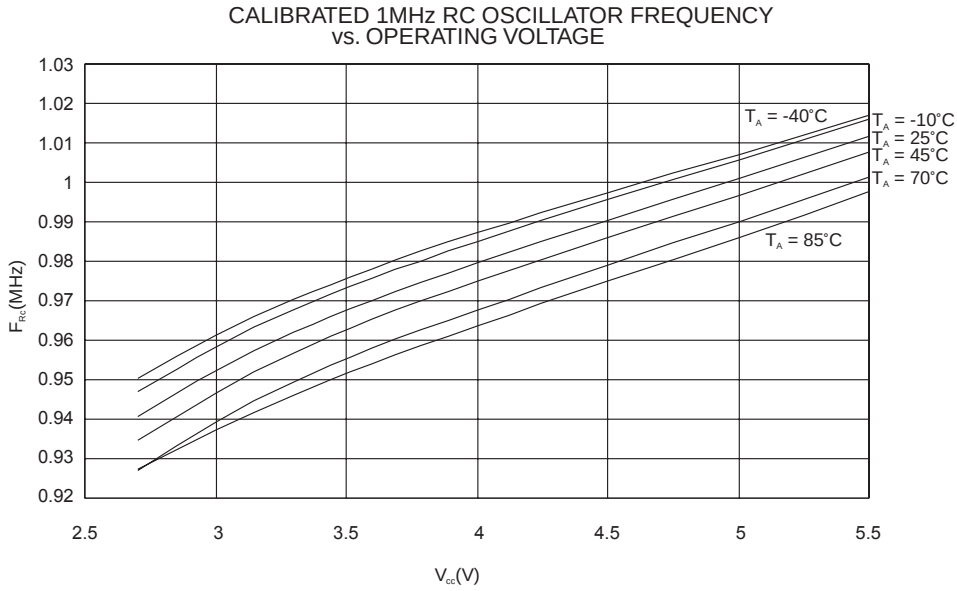


Figure 150. RC Oscillator Frequency vs. Temperature (the devices are calibrated to 2 MHz at Vcc = 5V, T=25c)

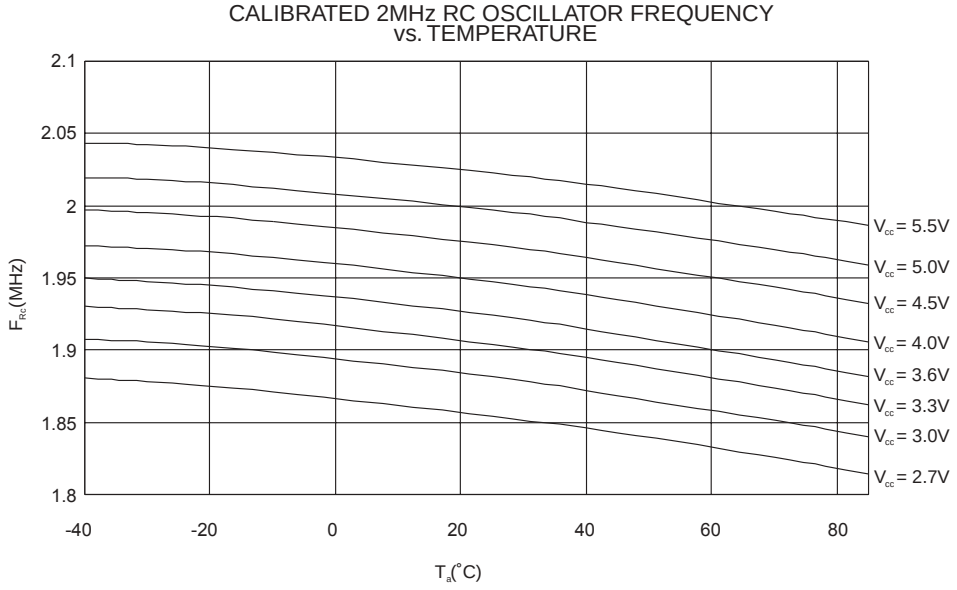


Figure 151. RC Oscillator Frequency vs. Operating Voltage (the devices are calibrated to 2 MHz at Vcc = 5V, T=25c)

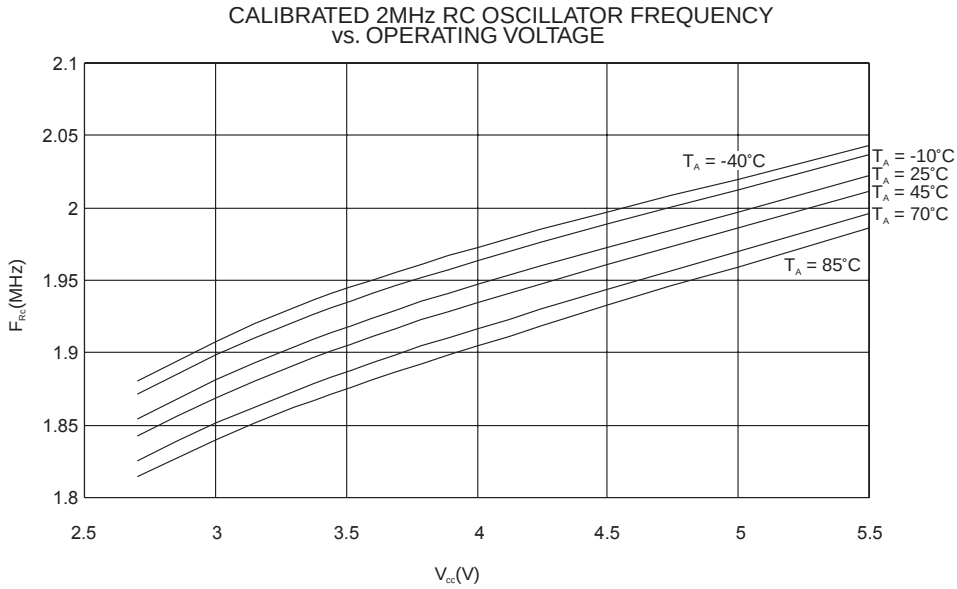


Figure 152. RC Oscillator Frequency vs. Temperature (the devices are calibrated to 4 MHz at Vcc = 5V, T=25c)

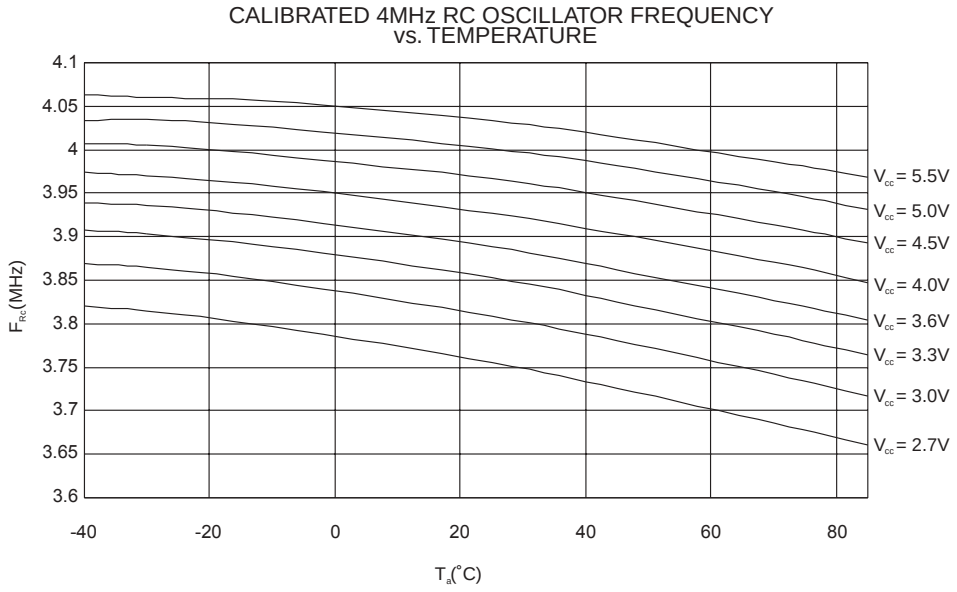


Figure 153. RC Oscillator Frequency vs. Operating Voltage (the devices are calibrated to 4 MHz at Vcc = 5V, T=25c)

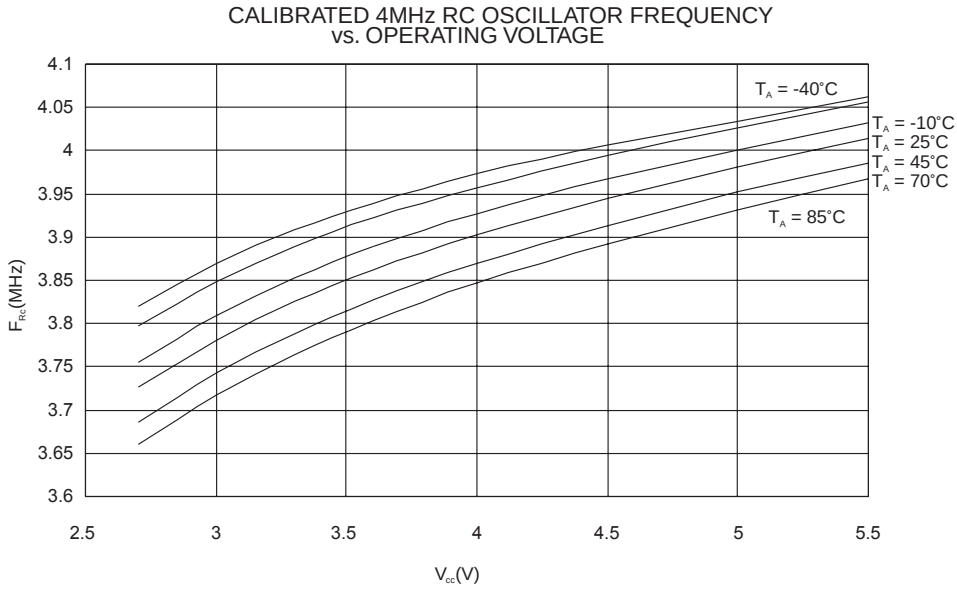


Figure 154. RC Oscillator Frequency vs. Temperature (the devices are calibrated to 8 MHz at Vcc = 5V, T=25c)

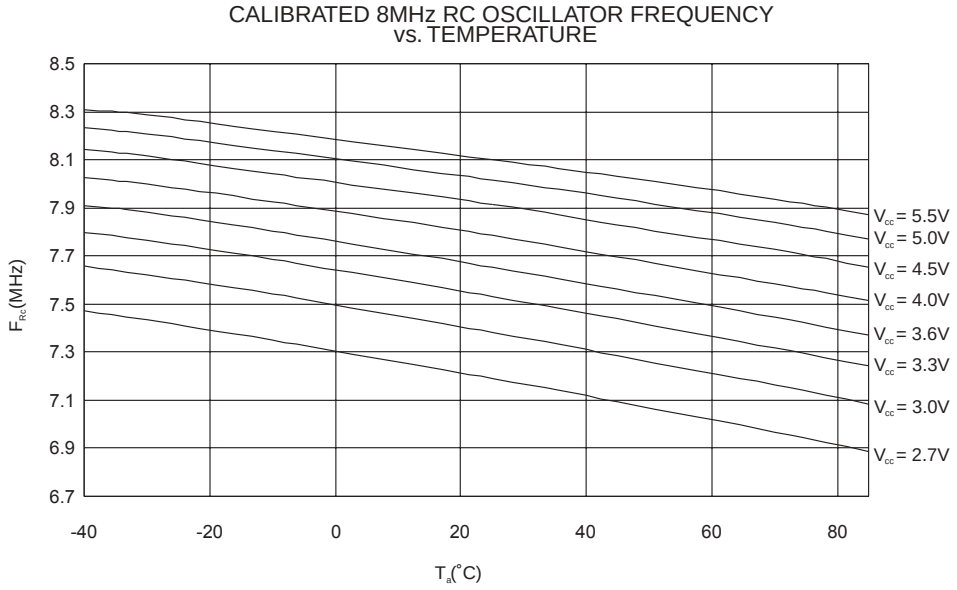
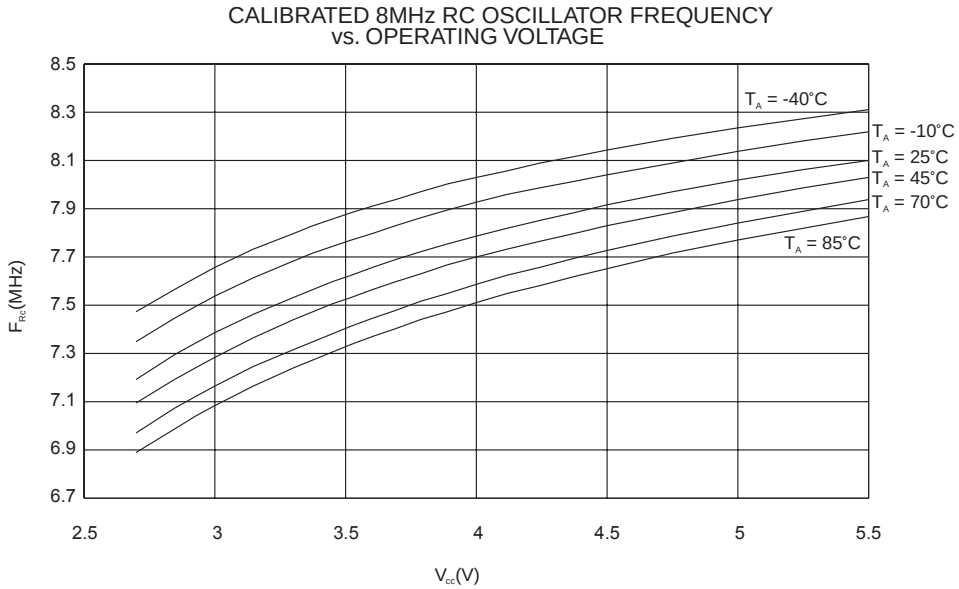


Figure 155. RC Oscillator Frequency vs. Operating Voltage (the devices are calibrated to 8 MHz at Vcc = 5V, T=25c)



Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	8
$3E ($5E)	SPH	–	–	–	–	SP11	SP10	SP9	SP8	10
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	10
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register								80
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	45, 65
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–	66
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	80, 110, 128
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	81, 111, 128
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	246
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	175
$35 ($55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	30, 64
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	38, 65, 226
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	78
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)								80
$31[1] ($51)[1]	OSCCAL	Oscillator Calibration Register								28
	OCDR	On-Chip Debug Register								222
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	54,83,129,196,216
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	105
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	108
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								109
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								109
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								109
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								109
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								109
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								109
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								110
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								110
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	123
$24 ($44)	TCNT2	Timer/Counter2 (8 Bits)								125
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register								125
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	126
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	40
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]				162
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	160
$1F ($3F)	EEARH	–	–	–	–	–	–	EEAR9	EEAR8	17
$1E ($3E)	EEARL	EEPROM Address Register Low Byte								17
$1D ($3D)	EEDR	EEPROM Data Register								17
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	17
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	62
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	62
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	62
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	62
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	62
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	63
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	63
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	63
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	63
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	63
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	63
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	63
$0F ($2F)	SPDR	SPI Data Register								136
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	136
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	134
$0C ($2C)	UDR	USART I/O Data Register								157
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	158
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	159
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte								162
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	197
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	212
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	214
$05 ($25)	ADCH	ADC Data Register High Byte								215
$04 ($24)	ADCL	ADC Data Register Low Byte								215
$03 ($23)	TWDR	Two-wire Serial Interface Data Register								177
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	177

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	176
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register								175

Notes:
1. When the OCDEN Fuse is unprogrammed, the OSCCAL Register is always accessed on this address. Refer to the debugger specific documentation for details on how to use the OCDR Register.
2. Refer to the USART description for details on how to access UBRRH and UCSRC.
3. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
4. Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers $00 to $1F only.

Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← Stack	None	4
RETI		Interrupt Return	PC ← Stack	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	Stack ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← Stack	None	2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-Chip Debug Only	None	N/A

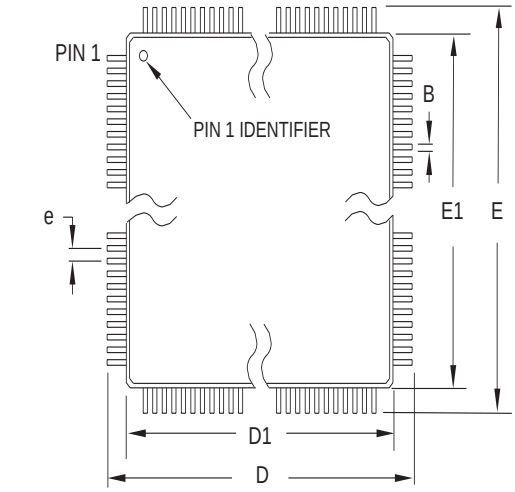
Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
8	2.7 - 5.5V	ATmega32L-8AC ATmega32L-8PC ATmega32L-8MC	44A 40P6 44M1	Commercial (0°C to 70°C)
		ATmega32L-8AI ATmega32L-8PI ATmega32L-8MI	44A 40P6 44M1	Industrial (-40°C to 85°C)
16	4.5 - 5.5V	ATmega32-16AC ATmega32-16PC ATmega32-16MI	44A 40P6 44M1	Commercial (0°C to 70°C)
		ATmega32-16AI ATmega32-16PI ATmega32-16MC	44A 40P6 44M1	Industrial (-40°C to 85°C)

Package Type	
44A	44-lead, Thin (1.0 mm) Plastic Gull Wing Quad Flat Package (TQFP)
40P6	40-pin, 0.600” Wide, Plastic Dual Inline Package (PDIP)
44M1	44-pad, 7 x 7 x 1.0 mm body, lead pitch 0.50 mm, Micro Lead Frame Package (MLF)

Packaging Information

44A



COMMON DIMENSIONS
(Unit of Measure = mm)

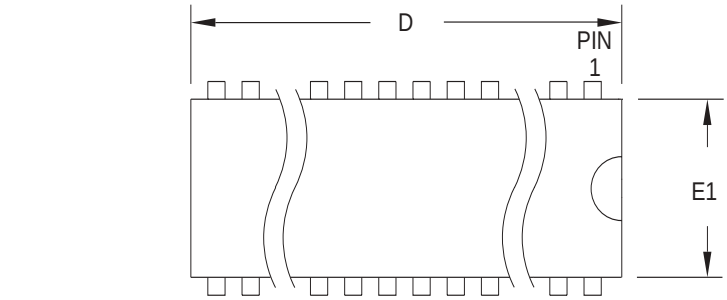
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.20	
A1	0.05	–	0.15	
A2	0.95	1.00	1.05	
D	11.75	12.00	12.25	
D1	9.90	10.00	10.10	Note 2
E	11.75	12.00	12.25	
E1	9.90	10.00	10.10	Note 2
B	0.30	–	0.45	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.80 TYP			

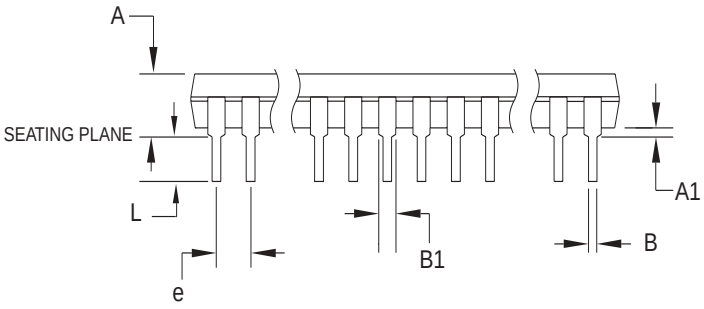
Notes:
1. This package conforms to JEDEC reference MS-026, Variation ACB.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10 mm maximum.

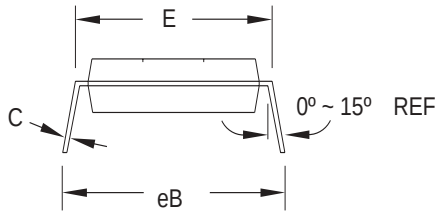
10/5/2001

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**44A,** 44-lead, 10 x 10 mm Body Size, 1.0 mm Body Thickness, 0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP)	44A	B

40P6







COMMON DIMENSIONS
(Unit of Measure = mm)

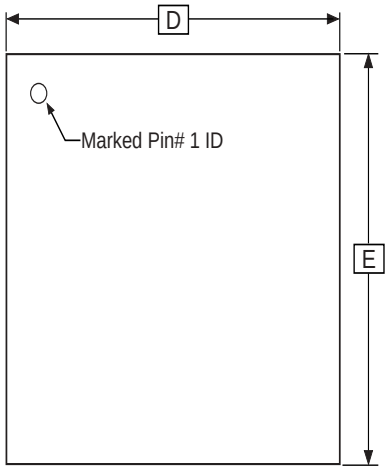
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	4.826	
A1	0.381	–	–	
D	52.070	–	52.578	Note 2
E	15.240	–	15.875	
E1	13.462	–	13.970	Note 2
B	0.356	–	0.559	
B1	1.041	–	1.651	
L	3.048	–	3.556	
C	0.203	–	0.381	
eB	15.494	–	17.526	
e	2.540 TYP			

Notes:
1. This package conforms to JEDEC reference MS-011, Variation AC.
2. Dimensions D and E1 do not include mold Flash or Protrusion. Mold Flash or Protrusion shall not exceed 0.25 mm (0.010").

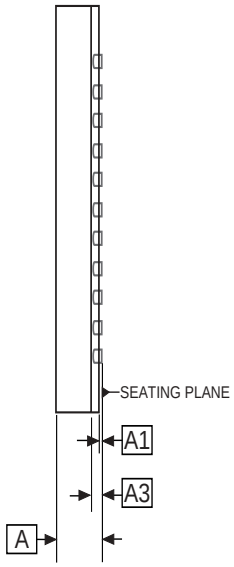
09/28/01

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**40P6**, 40-lead (0.600"/15.24 mm Wide) Plastic Dual Inline Package (PDIP)	40P6	B

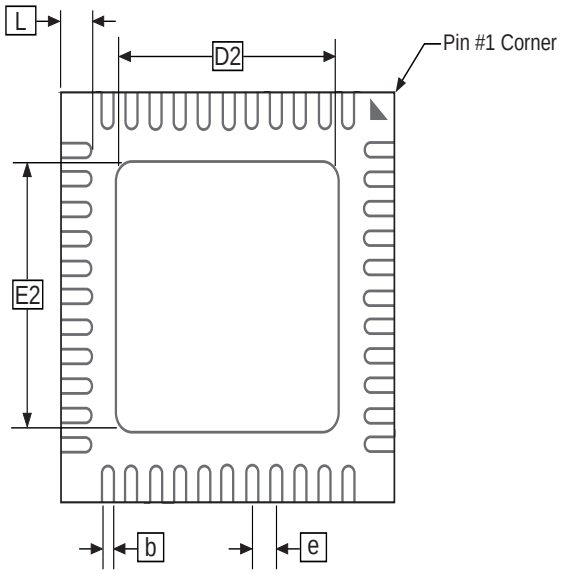
44M1



TOP VIEW



SIDE VIEW



BOTTOM VIEW

COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	–	0.02	0.05	
A3		0.25 REF		
b	0.18	0.23	0.30	
D		7.00 BSC		
D2	5.00	5.20	5.40	
E		7.00 BSC		
E2	5.00	5.20	5.40	
e		0.50 BSC		
L	0.35	0.55	0.75	

Notes: 1. JEDEC Standard MO-220, Fig. 1 (SAW Singulation) VKKD-1.

01/15/03

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**44M1**, 44-pad, 7 x 7 x 1.0 mm Body, Lead Pitch 0.50 mm Micro Lead Frame Package (MLF)	44M1	C

Errata

ATmega32 Rev. A

There are no errata for this revision of ATmega32.However, a proposal for solving problems regarding the JTAG instruction IDCODE is presented below.

IDCODE masks data from TDI input

The public but optional JTAG instruction IDCODE is not implemented correctly according to IEEE1149.1; a logic one is scanned into the shift register instead of the TDI input while shifting the Device ID Register. Hence, captured data from the preceding devices in the boundary scan chain are lost and replaced by all-ones, and data to succeeding devices are replaced by all-ones during Update-DR.

If ATmega32 is the only device in the scan chain, the problem is not visible.

Problem Fix / Workaround

Select the Device ID Register of the ATmega32 (Either by issuing the IDCODE instruction or by entering the Test-Logic-Reset state of the TAP controller) to read out the contents of its Device ID Register and possibly data from succeeding devices of the scan chain. Note that data to succeeding devices cannot be entered during this scan, but data to preceding devices can. Issue the BYPASS instruction to the ATmega32 to select its Bypass Register while reading the Device ID Registers of preceding devices of the boundary scan chain. Never read data from succeeding devices in the boundary scan chain or upload data to the succeeding devices while the Device ID Register is selected for the ATmega32. Note that the IDCODE instruction is the default instruction selected by the Test-Logic-Reset state of the TAP-controller.

Alternative Problem Fix / Workaround

If the Device IDs of all devices in the boundary scan chain must be captured simultaneously (for instance if blind interrogation is used), the boundary scan chain can be connected in such way that the ATmega32 is the fist device in the chain. Update-DR will still not work for the succeeding devices in the boundary scan chain as long as IDCODE is present in the JTAG Instruction Register, but the Device ID registered cannot be uploaded in any case.

Datasheet Change Log for ATmega32

Please note that the referring page numbers in this section are referred to this document. The referring revision in this section are referring to the document revision.

Changes from Rev. 2503E-09/03 to Rev. 2503F-12/03

1. **Updated “Calibrated Internal RC Oscillator” on page 27.**

Changes from Rev. 2503D-02/03 to Rev. 2503E-09/03

1. **Updated and changed “On-chip Debug System” to “JTAG Interface and On-chip Debug System” on page 33.**
2. **Updated Table 15 on page 35.**
3. **Updated “Test Access Port – TAP” on page 217 regarding the JTAGEN fuse.**
4. **Updated description for Bit 7 – JTD: JTAG Interface Disable on page 226.**
5. **Added a note regarding JTAGEN fuse to Table 105 on page 255.**
6. **Updated Absolute Maximum Ratings* , DC Characteristics and ADC Characteristics in “Electrical Characteristics” on page 285.**
7. **Added a proposal for solving problems regarding the JTAG instruction IDCODE in “Errata” on page 308.**

Changes from Rev. 2503C-10/02 to Rev. 2503D-02/03

1. **Added EEAR9 in EEARH in “Register Summary” on page 299.**
2. **Added Chip Erase as a first step in“Programming the Flash” on page 282 and “Programming the EEPROM” on page 283.**
3. **Removed reference to “Multi-purpose Oscillator” application note and “32 kHz Crystal Oscillator” application note, which do not exist.**
4. **Added information about PWM symmetry for Timer0 and Timer2.**
5. **Added note in “Filling the Temporary Buffer (Page Loading)” on page 249 about writing to the EEPROM during an SPM Page Load.**
6. **Added “Power Consumption” data in “Features” on page 1.**
7. **Added section “EEPROM Write During Power-down Sleep Mode” on page 20.**
8. **Added note about Differential Mode with Auto Triggering in “Prescaling and Conversion Timing” on page 202.**
9. **Updated Table 90 on page 230.**
10. **Added updated “Packaging Information” on page 305.**

Changes from Rev. 2503B-10/02 to Rev. 2503C-10/02

1. **Updated the “DC Characteristics” on page 285.**

Changes from Rev. 2503A-03/02 to Rev. 2503B-10/02

1. **Canged the endurance on the Flash to 10,000 Write/Erase Cycles.**
2. **Bit nr.4 – ADHSM – in SFIOR Register removed.**
3. **Added the section “Default Clock Source” on page 23.**
4. **When using External Clock there are some limitations regards to change of frequency. This is described in “External Clock” on page 29 and Table 118 on page 287.**
5. **Added a sub section regarding OCD-system and power consumption in the section “Minimizing Power Consumption” on page 32.**
6. **Corrected typo (WGM-bit setting) for:**
   - **“Fast PWM Mode” on page 73 (Timer/Counter0)**
   - **“Phase Correct PWM Mode” on page 74 (Timer/Counter0)**
   - **“Fast PWM Mode” on page 118 (Timer/Counter2)**
   - **“Phase Correct PWM Mode” on page 119 (Timer/Counter2)**
7. **Corrected Table 67 on page 162 (USART).**
8. **Updated V_{IL}, I_{IL}, and I_{IH} parameter in “DC Characteristics” on page 285.**
9. **Updated Description of OSCCAL Calibration Byte.**

   In the datasheet, it was not explained how to take advantage of the calibration bytes for 2, 4, and 8 MHz Oscillator selections. This is now added in the following sections:

   Improved description of “Oscillator Calibration Register – OSCCAL” on page 28 and “Calibration Byte” on page 256.
10. **Corrected typo in Table 42.**
11. **Corrected description in Table 45 and Table 46.**
12. **Updated Table 119, Table 121, and Table 122.**
13. **Added “Errata” on page 308.**

Table of Contents

Features ***1***

Pin Configurations ***2***

Disclaimer ***2***

Overview ***3***

Block Diagram 3

Pin Descriptions 4

About Code Examples ***5***

AVR CPU Core *6*

Introduction 6

Architectural Overview 6

ALU – Arithmetic Logic Unit 7

Status Register 8

General Purpose Register File 9

Stack Pointer 10

Instruction Execution Timing 11

Reset and Interrupt Handling 11

AVR ATmega32 Memories ***14***

In-System Reprogrammable Flash Program Memory 14

SRAM Data Memory 15

EEPROM Data Memory 16

I/O Memory 21

System Clock and Clock Options ***22***

Clock Systems and their Distribution 22

Clock Sources 23

Default Clock Source 23

Crystal Oscillator 24

Low-frequency Crystal Oscillator 26

External RC Oscillator 26

Calibrated Internal RC Oscillator 27

External Clock 29

Timer/Counter Oscillator 29

Power Management and Sleep Modes ***30***

Idle Mode 31

ADC Noise Reduction Mode 31

Power-down Mode 31

Power-save Mode 31

Standby Mode 32

Extended Standby Mode 32

Minimizing Power Consumption 32

System Control and Reset 34

Internal Voltage Reference 39

Watchdog Timer 39

Interrupts 42

Interrupt Vectors in ATmega32 42

I/O Ports 47

Introduction 47

Ports as General Digital I/O 48

Alternate Port Functions 52

Register Description for I/O Ports 62

External Interrupts 64

8-bit Timer/Counter0 with PWM 67

Overview 67

Timer/Counter Clock Sources 68

Counter Unit 68

Output Compare Unit 69

Compare Match Output Unit 70

Modes of Operation 71

Timer/Counter Timing Diagrams 76

8-bit Timer/Counter Register Description 78

Timer/Counter0 and Timer/Counter1 Prescalers 82

16-bit Timer/Counter1 84

Overview 84

Accessing 16-bit Registers 87

Timer/Counter Clock Sources 89

Counter Unit 89

Input Capture Unit 91

Output Compare Units 92

Compare Match Output Unit 94

Modes of Operation 95

Timer/Counter Timing Diagrams 103

16-bit Timer/Counter Register Description 105

8-bit Timer/Counter2 with PWM and Asynchronous Operation .. 112

Overview 112

Timer/Counter Clock Sources 113

Counter Unit 113

Output Compare Unit 114

Compare Match Output Unit 115
Modes of Operation 116
Timer/Counter Timing Diagrams 121
8-bit Timer/Counter Register Description 123
Asynchronous Operation of the Timer/Counter 126
Timer/Counter Prescaler 129

Serial Peripheral Interface – SPI 130
$\overline{SS}$ Pin Functionality 134
Data Modes 137

USART 138
Overview 138
Clock Generation 139
Frame Formats 142
USART Initialization 144
Data Transmission – The USART Transmitter 145
Data Reception – The USART Receiver 148
Asynchronous Data Reception 151
Multi-processor Communication Mode 155
Accessing UBRRH/ UCSRC Registers 156
USART Register Description 157
Examples of Baud Rate Setting 163

Two-wire Serial Interface 167
Features 167
Two-wire Serial Interface Bus Definition 167
Data Transfer and Frame Format 168
Multi-master Bus Systems, Arbitration and Synchronization 170
Overview of the TWI Module 173
TWI Register Description 175
Using the TWI 178
Transmission Modes 181
Multi-master Systems and Arbitration 194

Analog Comparator 196
Analog Comparator Multiplexed Input 198

Analog to Digital Converter 199
Features 199
Operation 200
Starting a Conversion 201
Prescaling and Conversion Timing 202
Changing Channel or Reference Selection 205
ADC Noise Canceler 207
ADC Conversion Result 211

JTAG Interface and On-chip Debug System 217
Features 217
Overview 217
Test Access Port – TAP 217
TAP Controller 219
Using the Boundary-scan Chain 220
Using the On-chip Debug System 220
On-chip Debug Specific JTAG Instructions 221
On-chip Debug Related Register in I/O Memory 222
Using the JTAG Programming Capabilities 222
Bibliography 222

IEEE 1149.1 (JTAG) Boundary-scan 223
Features 223
System Overview 223
Data Registers 223
Boundary-scan Specific JTAG Instructions 225
Boundary-scan Chain 227
ATmega32 Boundary-scan Order 237
Boundary-scan Description Language Files 241

Boot Loader Support – Read-While-Write Self-Programming 242
Features 242
Application and Boot Loader Flash Sections 242
Read-While-Write and no Read-While-Write Flash Sections 242
Boot Loader Lock Bits 244
Entering the Boot Loader Program 245
Addressing the Flash during Self-Programming 247
Self-Programming the Flash 248

Memory Programming 254
Program And Data Memory Lock Bits 254
Fuse Bits 255
Signature Bytes 256
Calibration Byte 256
Parallel Programming Parameters, Pin Mapping, and Commands 257
Parallel Programming 259
SPI Serial Downloading 268
SPI Serial Programming Pin Mapping 268
Programming via the JTAG Interface 272

Electrical Characteristics 285
Absolute Maximum Ratings* 285
DC Characteristics 285
External Clock Drive Waveforms 287
External Clock Drive 287

Two-wire Serial Interface Characteristics 288
SPI Timing Characteristics 289
ADC Characteristics 291

ATmega32 Typical Characteristics – Preliminary Data 294

Register Summary 299

Instruction Set Summary 301

Ordering Information 304

Packaging Information 305
44A 305
40P6 306
44M1 307

Errata 308
ATmega32 Rev. A 308

Datasheet Change Log for ATmega32 309
Changes from Rev. 2503E-09/03 to Rev. 2503F-12/03 309
Changes from Rev. 2503D-02/03 to Rev. 2503E-09/03 309
Changes from Rev. 2503C-10/02 to Rev. 2503D-02/03 309
Changes from Rev. 2503B-10/02 to Rev. 2503C-10/02 309
Changes from Rev. 2503A-03/02 to Rev. 2503B-10/02 310

Table of Contents i

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

Disclaimer: Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.

© Atmel Corporation 2003. **All rights reserved.** Atmel® and combinations thereof, AVR®, and AVR Studio® are the registered trademarks of Atmel Corporation or its subsidiaries. Microsoft®, Windows®, Windows NT®, and Windows XP® are the registered trademarks of Microsoft Corporation. Other terms and product names may be the trademarks of others

Printed on recycled paper.

2503F–AVR–12/03