IMPLEMENTASI TAPIS *FINITE IMPULSE RESPONSE* DENGAN *DISCRETE COSINE TRANSFORM* PADA FPGA XILINX ARTIX-7 XC7A200T

SKRIPSI

untuk memenuhi salah satu persyaratan

mencapai derajat Sarjana S1



Disusun oleh:

Fahmi Lukistriya

13524081

Jurusan Teknik Elektro

Fakultas Teknologi Industri

Universitas Islam Indonesia

Yogyakarta

2017

LEMBAR PENGESAHAN

IMPLEMENTASI TAPIS FINITE IMPULSE RESPONSE DENGAN DISCRETE COSINE TRANSFORM PADA FPGA XILINX ARTIX-7 XC7A200T

TUGAS AKHIR

Diajukan sebagai Salah Satu Syarat untuk Memperoleh
Gelar Sarjana Teknik
pada Program Studi Teknik Elektro
Fakultas Teknologi Industri
Universitas Islam Indonesia

Disusun oleh:

Fahmi Lukistriya
13524081

Yogyakarta, 13 Agustus 2018

Menyetujui,

Pembimbing

Dr.Eng., Hendra Setiawan, S.T., M.T.
025200526

LEMBAR PENGESAHAN

SKRIPSI

IMPLEMENTASI TAPIS FINITE IMPULSE RESPONSE DENGAN DISCRETE COSINE TRANSFORM PADA FPGA XILINX ARTIX-7 XC7A200T

Dipersiapkan dan disusun oleh:

Fahmi Lukistriya

13524081

Telah dipertahankan di depan dewan penguji

Pada tanggal: 13 Agustus 2018

Susunan dewan penguji

Ketua Penguji : Dr.Eng., Hendra Setiawan, S.T., M.T., _____

Anggota Penguji 1: Dwi Ana Ratna Wati, S.T., M.Eng., _____

Anggota Penguji 2: Tito Yuwono, S.T., M.Sc., _____

Skripsi ini telah diterima sebagai salah satu persyaratan untuk memperoleh gelar Sarjana

Yogyakarta, 13 Agustus 2018

Ketua Program Studi Teknik Elektro

UNIVERSITAS ISLAM INDONESIA ★ YOGYAKARTA ★ FAKULTAS TEKNOLOGI INDUSTRI

Yusuf Aziz Amrulloh S.T., M.Sc., Ph.D.

045240101

PERNYATAAN

Dengan ini Saya menyatakan bahwa:

Skripsi ini tidak mengandung karya yang diajukan untuk memperoleh gelar kesarjanaan di suatu Perguruan Tinggi, dan sepanjang pengetahuan Saya juga tidak mengandung karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Informasi dan materi Skripsi yang terkait hak milik, hak intelektual, dan paten merupakan milik bersama antara tiga pihak yaitu penulis, dosen pembimbing, dan Universitas Islam Indonesia. Dalam hal penggunaan informasi dan materi Skripsi terkait paten maka akan diskusikan lebih lanjut untuk mendapatkan persetujuan dari ketiga pihak tersebut diatas.

Yogyakarta, 16 Juli 2018



Fahmi Lukistriya

KATA PENGANTAR

Segala puji bagi Allah Subhanahu wa Ta'ala, Rabb semesta alam, tiada daya upaya melainkan ata kehendak-Nya. Sholawat serta salam semoga dilimpahkan kepada Nabi Muhammad Sholalloohu 'alaihi wasallam, keluarga, para sahabat, dan semua pengikut hingga hari akhir.

Atas izin Allah Subhanahu wa Ta'ala tugas akhir yang berjudul "IMPLEMENTASI TAPIS FINITE IMPULSE RESPONSE DENGAN DISCRETE COSINE TRANSFORM PADA FPGA XILINX ARTIX-7 XC7A200T" dapat kami selesaikan melalui proses yang cukup panjang, tugas akhir ini disusun untuk memenuhi salah satu persyaratan untuk menyelesaikan program studi teknik elektro di Universitas Islam Indonesia.

Selama mengikuti pendidikan program studi teknik elektro sampai dengan proses penyelesaian tugas akhir, berbagai pihak telah memberikan fasilitas, membantu, membina dan membimbing penulis, untuk itu penulis mengucapkan terimakasih khususnya kepada:

1) Kedua orang tua yang telah memberikan dukungan dan do'a selama penulis menuntut ilmu di UII.
2) Bapak Dr.Eng., Hendra Setiawan, S.T., M.T. selaku ketua jurusan teknik elektro sekaligus pembimbing tugas akhir.
3) Bapak dan Ibu dosen pengajar yang telah memberikan ilmunya kepada penulis.
4) Bapak Mike Field dari Chirstchurch New Zealand pemilik akun github Hamsterworks sebagai pemilik *source code* dari *codec* audio.
5) Keluarga besar teknik elektro UII.

Penulis memohon maaf jika terdapat kesalahan dalam penulisan tugas akhir ini, semoga yang sedikit ini dapat bermanfaat dan dapat memberikan kontribusi dalam bidang keilmuan yang terkait.

Yogyakarta, 16 Juli 2018



Fahmi Lukistriya

ARTI LAMBANG DAN SINGKATAN

Singkatan	Kepanjangan
FPGA	: *field-programmable gate array*
DCT	: *discrete cosine transform*
FIR	: *finite impulse response*
VHDL	: VHSIC *hardware description language*
RAM	: *random acces memory*
ASIC	: *application specific integrated circuit*
IC	: *integrated circuit*
IP	: *inteligence property*
ADC	: *analog to digital converter*
DAC	: *digital to analog converter*
IO	: *input output*
I2C	: *inter-integrated circuit*
IDCT	: *invers discrete cosine transform*

ABSTRAK

Proses pengambilan sinyal yang rentan dipengaruhi hal luar membuat sinyal turut membawa derau yang dapat mempengaruhi keluaran jika tidak ditapis dari sinyal. Penapisan derau dapat dilakukan pada domain waktu salah satunya dengan menggunakan tapis FIR dan dalam domain frekuensi dengan membawa terlebih dahulu sinyal menuju domain frekuensi menggunakan transformasi DCT. FPGA merupakan salah satu perangkat yang dapat digunakan untuk menerapkan tapis digital, FPGA memiliki kelebihan dibandingkan dengan *microcontroller* dalam hal kecepatan karena memiliki fleksibilitas pada tingkat *hardware*, penerapan tapis pada FPGA perlu memperhatikan penggunaan *resource* dan *timing diagram* agar rancangan dapat berjalan baik pada FPGA.

FPGA yang digunakan pada penelitian ini adalah Xilinx artix-7 XCA200T, FPGA ini terpasang pada *development board* Nexys Video 4, *board* ini menggunakan sumber clock 100 MHz, *board* ini telah dilengkapi dengan *codec* audio ADAU1761 yang dapat digunakan menggunakan antarmuka I^2C, bahasa yang digunakan pada penelitian ini adalah VHDLdan ditulis pada perangkat lunak Xilinx ISE 14.7, simulator yang digunakan pada penelitian ini adalah Isim. Proses analisis dilakukan dengan membandingkan audio hasil perekaman pada FPGA dan perancangan pada MATLAB.

Penggunaan *resources* dan *timing diagram* rancangan tapis pada domain waktu maupun domain frekuensi tidak melebihi spesifikasi yang dimiliki oleh FPGA sehingga rancangan yang dibuat dapat diterapkan pada FPGA. Unjuk kerja yang dihasilkan pada penapisan derau di domain waktu dapat berjalan dengan melihat tanggapan frekuensi yang mirip, sedangkan untuk penapisan derau pada domain frekuensi menunjukan hasil yang tidak terlalu baik dilihat dari grafik tanggapan frekuensi yang tidak mirip dari FPGA dengan pembanding pada MATLAB.

Kata kunci : FPGA, DCT, FIR, VHDL, Xilinx Artix-7

DAFTAR ISI

LEMBAR PENGESAHAN.....i
LEMBAR PENGESAHAN.....i
PERNYATAAN.....iii
KATA PENGANTAR.....iv
ARTI LAMBANG DAN SINGKATAN.....v
ABSTRAK.....vi
DAFTAR ISI.....vii
DAFTAR GAMBAR.....ix
DAFTAR TABEL.....xi
BAB 1 PENDAHULUAN.....xi1
1.1 Latar Belakang Masalah.....1
1.2 Persamaanan Masalah.....1
1.3 Batasan Masalah.....2
1.4 Tujuan Penelitian.....2
1.5 Manfaat Penelitian.....2
BAB 2 TINJAUAN PUSTAKA.....3
2.1 Tinjuan Pustaka.....3
2.2 Tinjauan Teori.....5
2.2.1 *Field Programmable Logic Controller*.....5
2.2.2 Tapis FIR *(Finite Impulse Response)*.....6
2.2.3 *Discrete Cosine Transform*(DCT).....7
BAB 3 METODOLOGI.....8
3.1 Alur Penelitian.....8
3.2 Spesifikasi Tapis.....9

3.3 Perancangan Pada MATLAB 9
3.4 Perancangan Pada FPGA 10
3.5 Penulisan Program VHDL 13
3.6 Simulasi dan Penerapan Pada FPGA 13
3.7 Proses Pengujian 15
BAB 4 HASIL DAN PEMBAHASAN 16
4.1 Hasil Perancangan Pada MATLAB 16
4.2 Hasil Perancangan Pada FPGA 16
1.3 Simulasi Hasil Perancangan 20
1.4 Penggunaan *Resource* Dan Laporan *Timing Diagram* 21
1.5 Unjuk Kerja Tapis Digital 23
BAB 5 KESIMPULAN DAN SARAN 28
5.1 Kesimpulan 28
5.2 Saran 28
DAFTAR PUSTAKA 29
LAMPIRAN 30

DAFTAR GAMBAR

Gambar 2.1 FPGA Digilent Nexys Video Xilinx Artix-7 5
Gambar 2.2 Struktur FIR dalam domain waktu 6
Gambar 3.1 Alur Penelitian 8
Gambar 3.2 Proses pengubahan koefisien menjadi biner 10
Gambar 3.3 Proses perancangan penapisan domain frekuensi pada MATLAB 10
Gambar 3.4 Teknik antarmuka dengan *codec* audio ADAU1761 11
Gambar 3.5 Pemrosesan sinyal dan antarmuka dengan *codec* audio 12
Gambar 3.6 Membuat *project* diXilinxISE14.*7* versi *webpack* 13
Gambar 3.7 Proses membuat file *testbench* untuk simulasi pada Isim 14
Gambar 3.8 Proses pembentukan file *.bit 14
Gambar 3.9 Mendapatkan tanggapan frekuensi 145
Gambar 4.1 Koefisien tapis FIR dengan frekuensi *cut off* 700 Hz 16
Gambar 4.2 Diagram alir tapis FIRpada domain waktu 16
Gambar 4.3 Diagram alir penapisan pada domain frekuensi 17
Gambar 4.4 Proses masukan dan keluaran 17
Gambar 4.5 Proses DCT penapisan domain frekuensi 18
Gambar 4.6 Proses penapisan pada domain frekuensi 19
Gambar 4.7 Proses IDCT penapisan domain frekuensi 19
Gambar 4.8 Informasi *log* proses membentuk simulasi pada Isim 20
Gambar 4.9 Simulasi penapisan domain waktu pada perangkat lunak Isim 20
Gambar 4.10 Tanggapan frekuensi tapis FIR 64 Tap fc700 Hz 23
Gambar 4.11 Tanggapan frekuensi tapis FIR 128 Tap *fc* 700 Hz 24
Gambar 4.12 Tanggapan frekuensi tapis FIR 256 Tap fc700 Hz 24
Gambar 4.13 Tanggapan frekuensi penapisan domain frekuensi DCT 64 Tap fc700 Hz 24
Gambar 4.14 Tanggapan frekuensi penapisan domain frekuensi DCT 128 Tap fc700 Hz 25

Gambar 4.15 Tanggapan frekuensi penapisan domain frekuensi DCT 256 Tap fc700 Hz 25

Gambar 4.16 Tanggapan frekuensi penapisan pada domain waktu 26

Gambar 4.17 Tanggapan frekuensi penapisan pada domain frekuensi 26

DAFTAR TABEL

Tabel 3.1 Keterangan sinyal pada antarmuka 11

Tabel 3.2 Keterangan pemrosesan sinyal dan data interface gambar 3.5 12

Tabel 4.1 Laporan *resource* tapis FIR pada domain waktu 21

Tabel 4.2 Laporan *resource* penapisan pada domain frekuensi 22

Tabel 4.3 Laporan *timing diagram* tapis FIR domain waktu 22

Tabel 4.4 Laporan *timing diagram* penapisan pada domain frekuensi 23

Tabel 4.5 Laporan *roll-off* tanggapan frekuensi 27

BAB 1 PENDAHULUAN

1.1 Latar Belakang Masalah

Sinyal yang menjadi masukan suatu sistem mengandung informasi yang akan diproses untuk mendapatkan suatu keluaran, teknik pengambilan data dan sumber gangguan lain menyebabkan data yang diperoleh membawa serta derau dalam sinyal, derau ini dapat menganggu hasil keluaran jika tidak dilakukan penapisan pada sinyal.

Penerapan tapis digital dapat dilakukan pada perangkat microcontroller, mini komputer dan FPGA. Microcontroller merupakan IC yang dirancang untuk menjalankan fungsi spesifik dalam suatu sistem, dalam microcontroller telah terdapat sebuah inti prosesor, ruang memori, dan IO peripherals seperti ADC dan DAC, keberadaan IO peripherals ADC dan DAC membantu pengguna tanpa harus menambah pheripheral dari luar, keberadaan IO peripherals seperti ADC dan DAC saat ini tidak hanya terdapat pada microcontroller saja, mini komputer seperti Raspberry Pi telah menyertakan IO peripherals seperti pada microcontroller. Microcontroller dibangun menggunakan kumpulan logic element yang terkoneksi dan membentuk suatu fungsi tertentu, koneksi dan susunan dari logic element pada microcontroller adalah sesuatu yang tetap sehingga pengguna hanya dapat memanggil fungsi untuk digunakan bukan merubah struktur dari fungsi tersebut [1]. Perangkat selain microcontroller dan mini PC yang dapat digunakan untuk melakukan pemrosesan sinyal digital adalah FPGA, secara umum IC FPGA tidak menyertakan IO peripheral seperti dua perangkat sebelumnya sehingga diperlukan pheripheral tambahan untuk mendapatkan hal yang sama seperti kedua perangkat sebelumnya, FPGA merupakan IC yang berisi kumpulan logic element yang dapat dikonfigurasi ulang setiap koneksinya, pengguna dapat memaksimalkan unjuk kerja dari suatu proses yang sensitive terhadap perubahan clock dimana hal ini sulit dilakukan pada microcontroller dan mini komputer [2].

Banyak development board FPGA yang telah menyertakan codec audio sebagai antarmuka dengan sinyal audio, salah satu development board itu adalah board Digilent Nexys 4 Video dengan IC ADAU1761, IC ini merupakan codec audio dengan kemampuan pencuplikan mencapai 96 kHz dan lebar data 24 bit, codec audio ini dapat digunakan untuk melakukan penelitian pada bidang pemrosesan sinyal digital dengan data audio yang memiliki detail tinggi.

1.2 Rumusan Masalah

Rumusan masalah pada penelitian ini adalah:

1) Bagaimana kebutuhan *resources* dari tapis *finite impulse response* dengan *discrete cosine transform* dan tapis *finite impulse response* dalam domain waktu?
2) Bagaimana unjuk kerja dari tapis *finite impulse response* dengan *discrete cosine transform* dan tapis *finite impulse response* dalam domain waktu?

1.3 Batasan Masalah

Batasan masalah dalam penelitian ini adalah:

1) Papan FPGA yang digunakan adalah Digilent Nexys Video yang menggunakan FPGA Xilinx Artix-7 XC7A200T-1SBG484C.
2) Perangkat lunak yang digunakan adalah Xilinx ISE 14.7 versi webpack dengan XST untuk *synthesis* dan bahasa yang digunakan adalah VHDL.
3) Teknik antarmuka dari *codec* dan FPGA telah tersedia pada vendor pembuat *development board* dan tidak termasuk dalam pembahasan pada penelitian.

1.4 Tujuan Penelitian

1) Mengetahui unjuk kerja dari tapis *finite impulse response* dengan *discrete cosine transform* dan tapis *finite impulse response* dalam domain waktu pada FPGA Xilinx Artix-7 XCA200T.
2) Mengetahui kebutuhan *resources* dari tapis *finite impulse response* dengan *discrete cosine transform* dan tapis *finite impulse response* dalam domain waktupada FPGA Xilinx Artix-7 XCA200T.

1.5 Manfaat Penelitian

1) Menambah khazanah/sumber/ rujukan pada bidang penelitian tentang implementasi tapis digital pada sebuah FPGA.
2) Menambah alternatif untuk implementasi tapis *finite impulse response* dan tranformasi *discrete cosine transform* pada sebuah FPGA.
3) Memberikan informasi tenteng penggunaan *resource* dan *timing diagram* implementasi tapis digital pada FPGA.

BAB 2
TINJAUAN PUSTAKA

2.1 Tinjuan Pustaka

Tinjauan pustaka ini berisi literatur dari berbagai sumber seperti jurnal dan *conference paper* yang berkaitan dengan tapis digital dan penerapanya pada FPGA serta optimasi yang dapat dilakukan, berikut ini adalah beberapa penelitian mengenai penerapan tapis digital pada FPGA dan optimasi yang dilakukan.

Penapisan derau pada sinyal dapat dilakukan secara langsung dengan menyeleksi frekuensi yang dianggap derau, penapisan tersebut dapat dilakukan setelah sinyal yang masih dalam domain waktu dibawa menuju representasi dalam domain frekuensi, proses tersebut membutuhkan transformasi diskrit dan *inverse* nya, diantara transformasi diskrit yang ada adalah jenis transformasi diskrit FFT, DCT dan DFT. Penelitian yang dilakukan oleh Vinod [3] menggunakan transformasi DCT dan DWT sebagai alternatif selain FFT dan DFT, penelitian ini menggunakan transformasi DCT dan DWT tersebut untuk mendapatkan kompresi energi dari sinyal keluaran yang dihasilkan, peneliti menggunakan transformasi DCT dan DWT untuk membawa sinyal menuju domain frekuensi dan *inverse* nya untuk membawa sinyal kembali ke domain waktu, proses transformasi diskrit dilakukan pada MATLAB, hasil dari penelitian ini adalah kedua transformasi disktrit mampu melakukan kompresi sinyal sebanyak 30% dari sinyal aslinya.

Pemrosesan yang dilakukan baik itu transformasi diskrit maupun tapis derau banyak dilakukan dengan menggunakan komputer sebagai unit pemrosesan, penelitian banyak menggunakan bahasa pemrograman tingkat tinggi seperti MATLAB sebagai pengeksekusi algoritma di komputer. Noor Azhar [4] dalam penelitianya melakukan penerapan tapis digital tidak pada sebuah komputer, akan tetapi dilakukan pada sebuah FPGA, peneliti menerapkan algoritma *least mean square* pada FPGA Altera Flex10KLC84, data sinyal masukan dan sinyal hasil dari penapisan didapat dan dikirimkan kembali dengan menggunakan komputer melalui komunikasi serial USART, data yang diterima dianalisis dengan menggunakan MATLAB, penelitian ini membandingkan kinerja pemrosesan pada MATLAB dengan hasil yang didapat pemrosesan di FPGA, analisis dilakukan dengan melakukan perbandingan antara perhitungan yang dilakukan oleh MATLAB dan perhitungan oleh FPGA, hasil yang didapatkan dari penelitian ini adalah akurasi nilai keluaran dari FPGA dapat mencapai 100 % dengan perhitungan yang dilakukan di MATLAB.

Penerapan tapis digital atau algoritma lain pada FPGA menggunakan *resources* yang tersedia untuk membentuk rangkaian pada FPGA, penggunaan *resources* yang melebihi spesifikasi dapat menjadi penyebab gagalnya perancangan atau pembentukan rangkaian pada proses synthesis FPGA, teknik optimasi banyak dilakukan oleh peneliti untuk menekan penggunaan *resources* agar rancangan dapat diterapkan pada FPGA. Penelitian tentang optimasi dilakukan oleh Alfred dan kawan-kawan [5], peneliti menggunakan *distributed arithmetic* sebagai pengganti metode komputasi konvensional pada proses penapisan menggunakan tapis FIR, *distributed arithmetic* adalah metode komputasi yang memanfaatkan penggunaan memori untuk memangkas operasi aritmatika yang terjadi pada FPGA, penelitian ini menggunakan *hardware* side dari FPGA sebagai pemroses utamanya, hasil yang didapat adalah penerapan *distributed arithmetic* dengan penggunaan *resources* yang sama dengan penerapan menggunakan komputasi konvensional mampu meningkatkan *througput* keluaran FPGA sebesar 40%. Selain penelitian yang dilakukan oleh Alfred dan kawan-kawan, Yajun Zhou [6] juga melakukan optimasi pada komputasi tapis FIR di FPGA menggunakan *distributed arithmetic*, jumlah tap yang digunakan adalah sepanjang 32 tap tapis FIR. Hasil dari penelitian ini adalah rancangan yang dibuat mampu menghemat 50 % *dsp block* dari penggunaan *resources* dan proses penapisan tetap berjalan seperti saat menggunakan komputasi konvensional.

Penerapan penapisan menggunakan FIR adalah proses yang sederhana jika diterapkan pada bahasa tingkat tinggi seperti MATLAB maupun saat diterapkan pada FPGA, berbeda dengan transformasi diskrit yang mengandung perkalian serta pembagian yang jumlahnya banyak, penerapan pada FPGA menjadi rumit dan membutuhkan banyak *resources*. Trasnformasi diskrit DCT untuk citra digital pada FPGA pernah dilakukan oleh Imam dan kawan kawan [7], peneliti menggunakan simulink dan stateflow dari MATLAB sebagai media untuk melakukan perancangan FPGA, penelitian ini menggunakan HDL coder yang merupakan *library* tambahan pada MATLAB untuk membuat file HDL-nya, desain dari DCT dioptimasi untuk mengoptimasi penggunaan *resources* dari FPGA. Hasil yang didapat dari penelirian ini adalah DCT yang dirancang menggunakan MATLAB dapat diterapkan pada FPGA XC3S700AN-4FGG484, hasil ini didapatkan dari informasi penggunaan *resources* yang tidak melebihi spesifikasi *resources* dari FPGA yang digunakan dan unjuk kerja yang baik dari DCT.

2.2 Tinjauan Teori

2.2.1 *Field Programmable Logic Controller*

FPGA merupakan sirkuit terpadu (IC) dalam teknologi digital, di mana rangkaian logika dapat diprogram.Konfigurasi FPGA umumnya ditentukan dengan menggunakan bahasa deskripsi perangkat keras (HDL), mirip dengan yang digunakan untuk ASIC. FPGA kontemporer memiliki sumber daya logika gate yang luas dan RAM blok untuk melaksanakan komputasi digital yang kompleks. FPGA dapat digunakan untuk mengimplementasikan fungsi logis seperti pada ASIC. Kemampuan untuk dimodifikasi setelah manufaktur, konfigurasi ulang sebagian dari desain dan biaya rekayasa yang relatif lebih rendah dibandingkan ASIC, FPGA dapat menawarkan keuntungan pada banyak aplikasi [7].

Bahasa yang digunakan untuk membentuk koneksi dalam FPGA salah satunya adalah VHDL *(VHSIC Hardware Description Language)*, VHSIC *(Very High Speed Integrated Circuit)* merupakan salah satu jenis bahasa HDL yang digunakan untuk mendeskripsikan berbagai fungsi rangkaian digital seperti FPGA *(Field-programmable Gate Arrays)*, Gerbang logika, Flip-flop, dan sebagainya. *Development board* FPGA yang digunakan pada penelitian ini adalah Digilent Nexys Video yang ditunjukan pada gambar 2.1.



Gambar 2.1 FPGA Digilent Nexys Video Xilinx Artix-7

Development board Digilent Nexys 4 Video menggunakan IC FPGA Xilinx Artix 7 XCA200C dengan spesifikasi sebagai berikut.

- 33,650 *logic slices*, masing masing dengan empat *6-input* LUTs dan *8 flip-flops*
- 13 Mbits *fast block* RAM
- 10 *clock management*, masing masing dengan *phase-locked loop* (PLL)
- 740 DSP *slices*
- 100 MHz *clock.*

Development board ini telah terpasang sebuah *codec* audio, *codec* audio yang terdapat pada *development board* Digilent Nexys 4 Video adalah IC ADAU1761 dengan spesifikasi sebagai berikut,

- lebar data 24 bit
- frekuensi cuplik 8 sampai 96kHz

2.2.2 Tapis FIR *(Finite Impulse Response)*

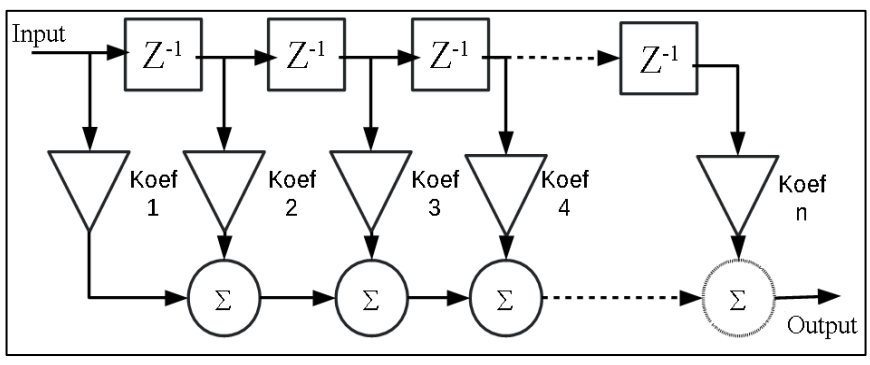
Tapis atau tapis adalah suatu sistem yang berfungsi untuk menyaring sinyal, sebagian sinyal akan dibiarkan lewat, sebagian yang lain akan akan ditahan. Tapis yang sering digunakan adalah tapis untuk menyaring sinyal berdasarkan frekuensi sinyal, artinya sinyal dengan frekuensi tertentu akan dibiarkan lewat, sinyal frekuensi yang lain akan ditahan. ada dua macam tapis digital, yaitu tapis IIR (*Infinite Impulse Response*) dan tapis FIR *(Finite Impulse Response*). Berikut ini persamaan dari *finite impulse response.*

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \quad \text{.....} \quad 1)$$

Keterangan:

y : keluaran hasil penapisan
x : masukan untuk penapisan
h : koefisien tapis FIR
n : urutan dari masukan tapis
k : urutan dari koefisien tapis

Tapis FIR adalah sistem yang murni umpan maju (*feed-forward),* stabil, strukturnya sederhana dan fasenya linier[8], dari persamaan diatas dapat diilustrasikan tapis *finite impulse response* menggunakan gambar 2.2 sebagai berikut.



Gambar 2.2 Struktur FIR dalam domain waktu

Keterangan:

Output : keluaran tapis

Input : masukan tapis

Koef : koefisien tapis

$\sum$: penjumlahan

Z^{-1} : *hold input* atau menahan masukan sebanyak 1 satuan waktu

2.2.3 *Discrete Cosine Transform*(DCT)

Pada domain frekuensi, sinyal yang acak dapat dibagi-bagi menjadi bentuk dasar sinyal sinusoidal yang kesemua sinyal itu adalah komponen dasar dari sinyal acak yang mengalami operasi penjumlahan. Selain pada transformasi *discrete fourier transform* dan algoritma penyempurnaya yaitu *fast fourier transform* terdapat transformasi lain yang dapat mengurai sebuah sinyal menjadi komponon dasar berdasarkan frekuensinya dan salah satunya adalah *discrete cosine transform*, pada dasarnya transformasi dengan DCT merupakan bagian real dari transformasi menggunakan DFT [8].

$$y(k) = \sum_{n=0}^{N-1} x(n)\, cos\left[\frac{\pi}{2}\left(n+\frac{1}{2}\right)k\right] \; 0 \le n \le N-1 \quad \text{.....} \quad 2)$$

Dan *inverse* dari *discrete cosine transform* adalah:

$$x(n) = \frac{2}{N}\left\{\frac{1}{2}y(0) + \sum_{n=0}^{N-1} y(k)\, cos\left[\frac{\pi}{2}\left(n+\frac{1}{2}\right)k\right]\right\}, 0 \le n \le N-1 \quad \text{.....} \quad 3)$$

Keterangan:

x : blok data dalam domain frekuensi

y : blok data dalam domain waktu

N : panjang blok data

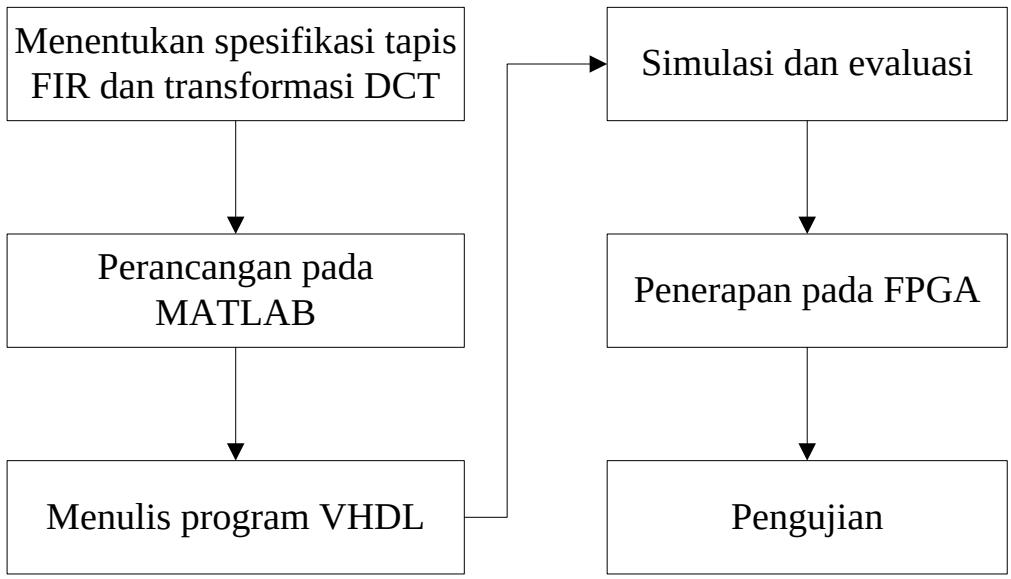
n : urutan dari nilai dalam domain waktu

k : urutan dari nilai dalam domain frekuensi

BAB 3
METODOLOGI

3.1 Alur Penelitian

Alur dari penelitian ditunjukan menggunakan gambar 3.1 sebagai berikut.



Gambar 3.1 Alur Penelitian

Penjelasan masing masing blok adalah sebagai berikut :

a. Menentukan spesifikasi *finite impulse response* dan *discrete cosine transform*

   Menentukan spesifikasi dari tapis *finite impulse response* dan transformasi *discrete cosine transform*, tahap ini menghasilkan spesifikasi tapis *finite impulse response* dan transformasi *discrete cosine transform*.

b. Perancangan pada MATLAB

   Perancangan dilakukan pada MATLAB, tahap ini menghasilkan tanggapan frekuensi dari tapis *finite impulse response*, koefisien tapis *finite impulse response* dan pengali tranformasi *discrete cosine transform* dalam bentuk bilangan biner.

c. Menulis program VHDL

   Penulisan program VHDL dilakukan pada *software* ISE Xilinx 14.7 berdasarkan perancangan yang telah dilakukan pada MATLAB, tahap ini menghasilkan sintaks program VHDL.

d. Simulasi dan evaluasi

   Simulasi dilakukan pada perangkat lunak, tahap ini menghasilkan informasi tentang *timing* setiap *port* dan kesalahan yang terjadi pada program secara sintaks maupun logika. Evaluasi program dilakukan jika dalam simulasi ditemukan kesalahan sintaks kesalahan *runtime* maupun kesalahan logika, tahap ini menghasilkan program VHDL yang telah terkoreksi dari kesalahan sintaks maupun logika berdasarkan informasi dari simulasi.

e. Penerapan pada FPGA

   Tahap penerapan pada FPGA ini adalah tahap *synthesis*, tahap *place and route*, tahap *mapping*, dan tahap pembentukan file *.hex, tahap ini menghasilkan informasi mengenai penggunaan *resources* dan *timing diagram*.

f. Pengujian

   Pengujian menggunakan file audio dengan frekuensi tertentu untuk mendapatkan keluaran audio kembali dari FPGA tersebut, file audio keluaran dari FPGA ini akan dilakukan analisis menggunakan MATLAB, tahap ini menghasilkan informasi mengenai unjuk kerja dari tapis digital.

3.2 Spesifikasi Tapis

Tapis *finite impulse response* yang akan diterapkan kedalam FPGA menggunakan spesifikasi sebagai berikut:

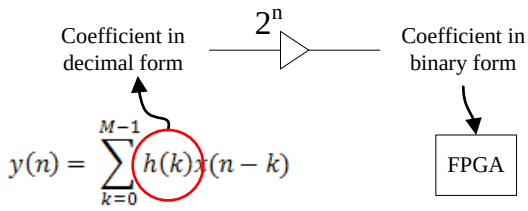
- Jumlah tap tapis : 64, 128, dan 256 tap
- Frekuensi cuplik : 48.000 Hz
- Frekuensi *cut-off* : 600, 700, dan 800 Hz
- Jenis tapis : lolos rendah
- *Windowing* : Hanning

Transformasi *discrete cosine transform* diproses setiap satu blok dengan jumlah data setiap bloknya adalah 256 data masukan, pemrosesan dilakukan dengan metode *overlap save* setengah.

3.3 Perancangan Pada MATLAB

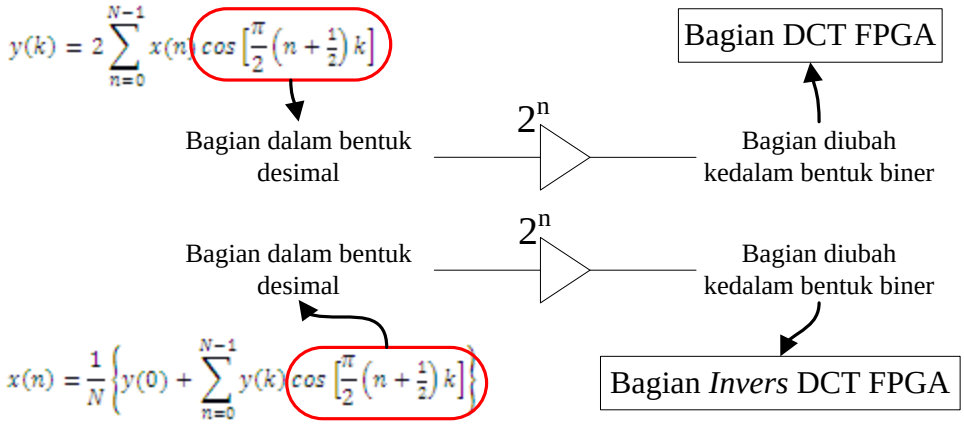
Perancangan pada MATLAB dilakukan dengan mengubah koefisien tapis dari FIR kedalam bentuk yang memudahkan pemrosesan pada FPGA, gambar 3.2 berikut ini adalah

proses pengubahan koefisien tapis dari bilangan decimal menjadi biner pada perancangan tapis domain waktu.



Gambar 3.2 Proses pengubahan koefisien menjadi biner

Sedangkan pada domain frekuensi, penapisan membutuhkan sebuah tranformasi diskrit dan *invers* dari transformasi diskrit tersebut, gambar 3.3 berikut ini adalah pengubahan bagian dari persamaan transformasi diskrit dan *invers* nya dengan menggunakan MATLAB.



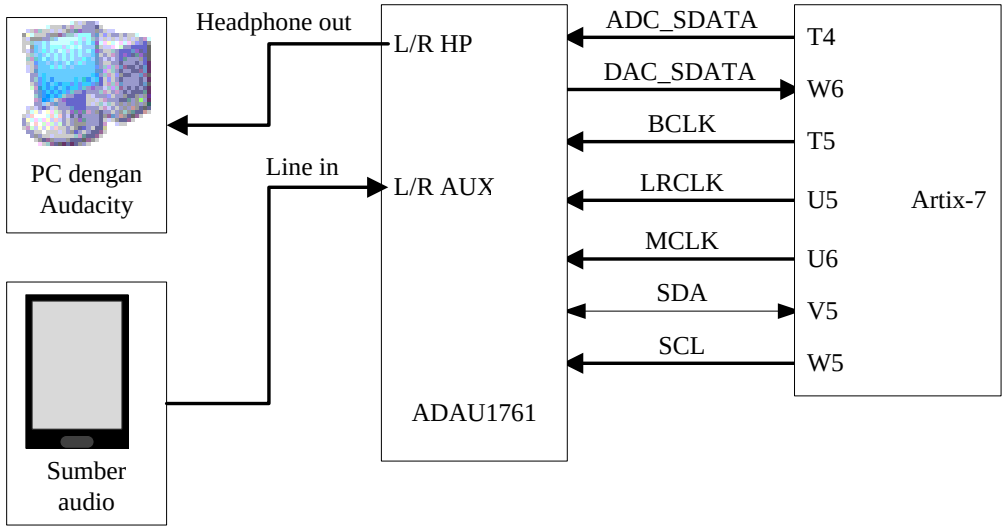
Gambar 3.3 Proses perancangan penapisan domain frekuensi pada MATLAB

Proses pada gambar 3.2 dan 3.3 adalah proses untuk membuat koefisien pengali yang nantinya akan digunakan pada FPGA, tapis pada FPGA dirancang dengan menggunakan perkalian antar bilangan biner sehingga dibutuhkan koefisien pengali pada tapis yang berbentuk bilangan biner, koefisien asli pada yang dilingkari merah pada gambar masih berbentuk bilangan desimal pecahan, bilangan ini dikalikan terlebih dahulu dengan konstanta 2 pangkat n untuk mendapat bilangan bulat agar bisa dikonversi kedalam bentuk biner.

3.4 Perancangan Pada FPGA

Perancangan dilakukan pada komputer dengan menggunakan perangkat lunak Xilinx ISE 14.7 untuk diterapkan kedalam sebuah FPGA dengan mengunduhkan file *.bit yang berisi

deskripsi gerbang logika dari runtun tapis, gambar 3.4 menjelaskan teknik antarmuka antara FPGAdengan *codec* audio ADAU1761.



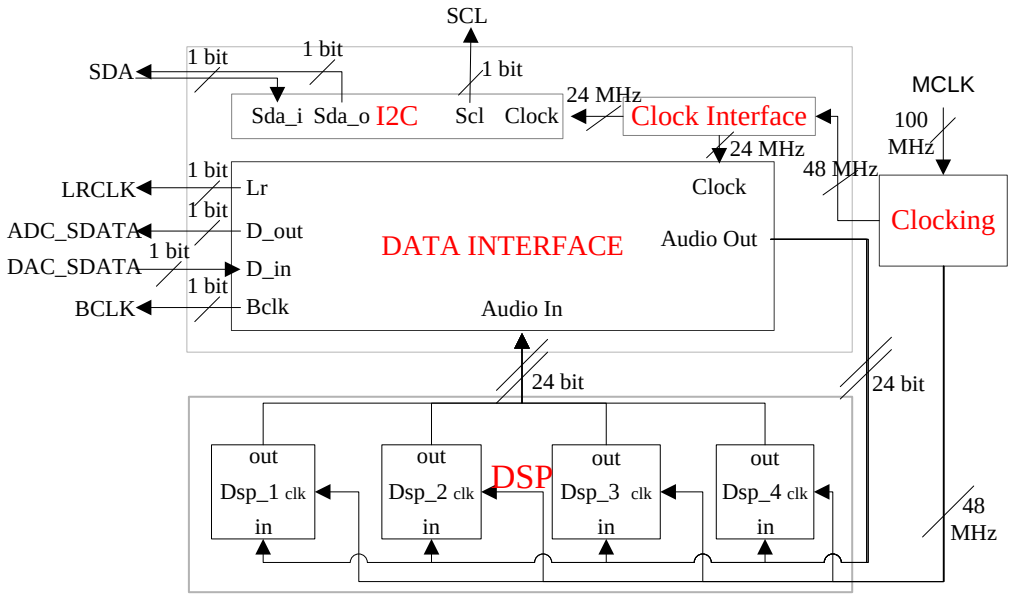
Gambar 3.4 Teknik antarmuka dengan *codec* audio ADAU1761

Keterangan dari masing masing jalur koneksi teknik antarmuka dengan *codec* audio pada gambar 3.4 dijelaskan pada tabel 3.1 sebagai berikut.

Tabel 3.1Keterangan sinyal pada antarmuka

Nama Sinyal	Pin FPGA	Fungsi Pin
ADC_SDATA	T4	*Serialized* audio dari ADC *conversion (record)*
DAC_SDATA	W6	*Serialized* audio dikonversi DAColeh *codec (record)*
BCLK	T5	*Serial data port clock*
LRCLK	U5	*Serial data port frame clock*
MCLK	U6	*Master clock*
SDA	V5	I^2C *configuration interface* Serial data
SCL	W5	I^2C *configuration interface* Serial *clock*
Headphone Out	-	*Playback* dan direkam menggunakan *software* Audacity
Line In	-	*Record* dan diputar menggunakan *smartphone*

Data yang telah didapatkan dari teknik antarmuka dengan *codec* audio akan dilakukan pemrosesan lebih lanjut pada program VHDL yang lain, gambar 3.5 berikut menjelaskan tentang koneksi antar *codec* audio dan pemrosesan sinyal digital pada FPGA.



Gambar 3.5 Pemrosesan sinyal dan antarmuka dengan *codec* audio

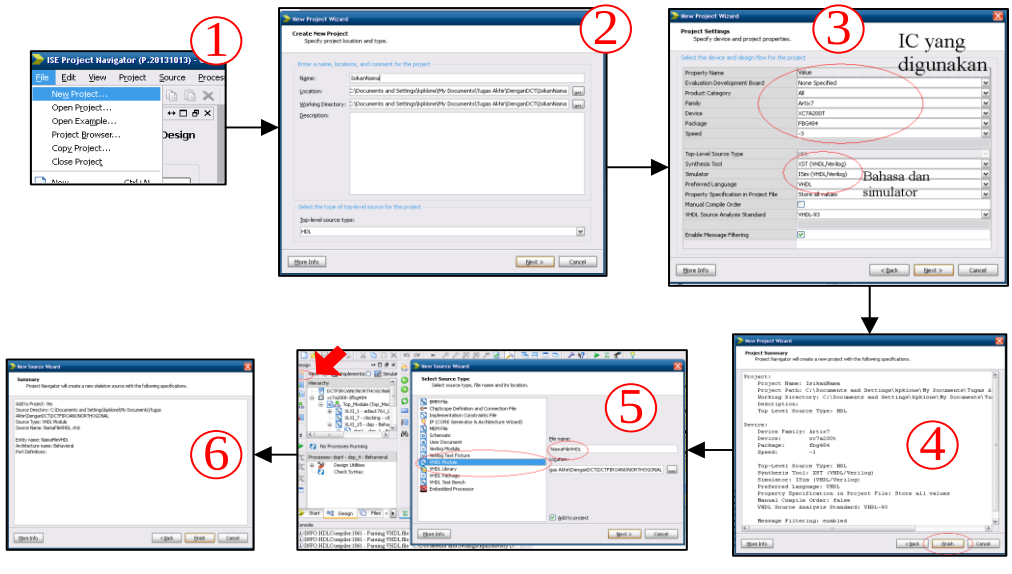
Pemrosesan sinyal tapis *finite impulse response* dan *discrete cosine transform* dilakukan pada blok DSP, keterangan gambar 3.5 ditunjukan pada tabel 3.2 berikut.

Tabel 3.2 Keterangan pemrosesan sinyal dan data interface gambar 3.5

Nama *Port*	Blok	Keterangan
In	Dsp_n	Masukan data audio 24 bit dari data interface
Out	Dsp_n	Keluaran data audio 24 bit dari data interface
ADC_DATA	Data interface	*Serialized* audio dari ADC *conversion (record)*
DAC_DATA	Data interface	*Serialized* audio dikonversi DAColeh *codec (record)*
LRCLK	Data interface	*Serial data port clock*
BCLK	Data interface	*Serial data port frame clock*
SCL	I^2C	I^2C *configuration interface* Serial data
SDA	I^2C	I^2C *configuration interface* Serial *clock*
MCLK	Clocking	*Master clock*
Line in	Data interface	Data 24 bit biner dari proses *record codec*
Hp Out	Data interface	Data 24 bit biner menuji proses *playback codec*
Audio In	Data interface	Data dari pengolahan DSP
Audio Out	Data interface	Data menuju pengolahan DSP

3.5 Penulisan Program VHDL

Program ditulis pada perangkat lunak Xilinx ISE 14.7 versi *webpack* yang mendukung *chip* FPGA Xilinx Artix-7 XCA200T, bahasa yang digunakan pada perancangan ini adalah bahasa VHDL, gambar 3.6 berikut ini menunjukan proses dalam membuat suatu *project* pada perangkat lunak Xilinx ISE 14.7 versi *webpack*.



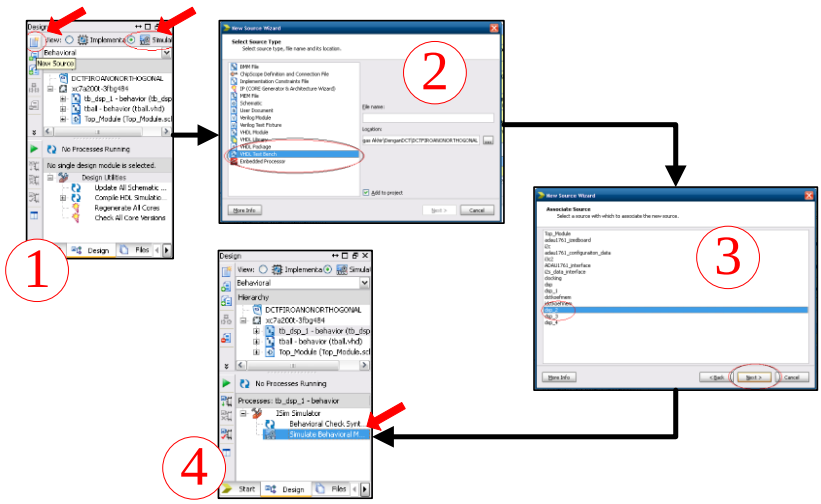
Gambar 3.6 Membuat *project* di Xilinx ISE 14.7 versi *webpack*

Keterangan:

1) Setelah perangkat lunak Xilinx ISE 14.7 terbuka, untuk membuat *project* baru untuk VHDL tekan tombol New Project pada menu File.
2) Setelah muncul jendela seperti pada gambar 3.6 isikan nama file dengan syarat nama file tidak mengandung spasi dan selalu dimulai dengan huruf bukan angka atau tanda baca.
3) Isikan *chip* FPGA yang digunakan dan bahasa yang digunakan serta simulator yang akan digunakan untuk simulasi.
4) Tekan tombol finish untuk membuat file *project* baru.
5) Setelah *project* terbentuk, file VHDL dapat dibuat dengan menekan tombol *new source*, pada jendela yang terbuka pilih VHDL module untuk membuat file terbaru.
6) Tekan tombol finish untuk membuat file VHDL baru.

3.6 Simulasi dan Penerapan Pada FPGA

Simulasi dilakukan menggunakan perangkat lunak Isim yang telah tersedia dalam paket Xilinx ISE 14.7 versi *webpack*, gambar 3.7 berikut adalah proses melakukan simulasi.

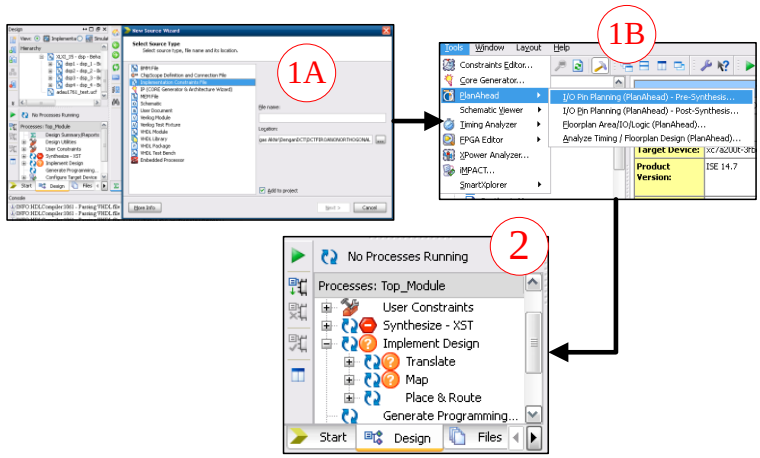


Gambar 3.7 Proses membuat file *testbench* untuk simulasi pada Isim

Keterangan:

1) Pilih tab *simulation* pada beranda Xilinx ISE, setelah berada pada tab *simulation* tekan tombol *new source* untuk membuat *testbench* simulasi dari VHDL.
2) Setelah jendela terbuka pilih VHDL *testbench* untuk membuat file *testbench*.
3) Jendela selanjutnya adalah file VHDL yang akan dilakukan simulasi, perintah ini akan membuat file *testbench* dari file VHDL yang akan disimulasikan secara otomatis.
4) Setelah file terbentuk, tekan tombol *simulate behavior* di kolom bawah untuk melakukan simulasi.

Selanjutnya adalah proses untuk membentuk file *.bit yang berisi deskripsi dari rangkaian sirkuit yang akan diterapkan pada FPGA, gambar 3.8 berikut ini adalah proses dari pembentukan file *.bit.



Gambar 3.8 Proses pembentukan file *.bit

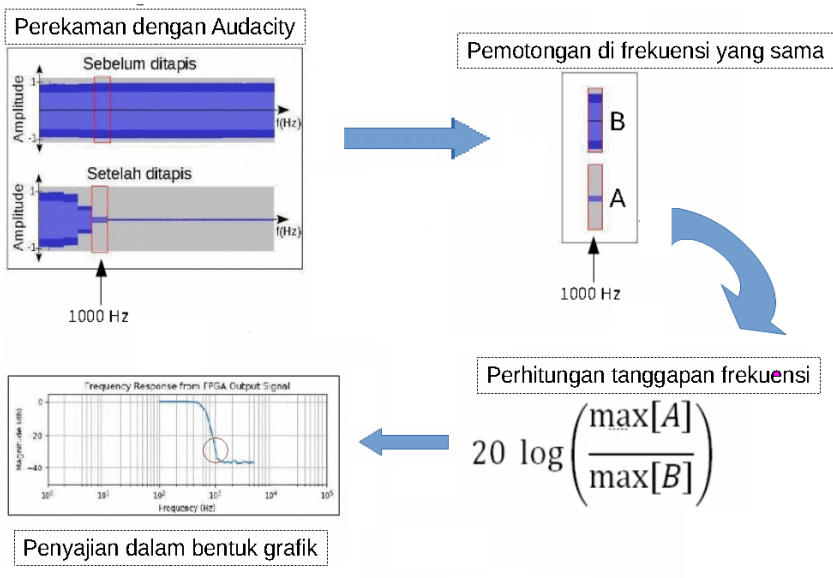
Keterangan:

1) Membuat file *constraint* berisi *netlist* atau sambungan dari file VHDL dengan *port* yang ada pada FPGA.
2) Proses ke-2 adalah proses *synthesize* dengan menggunakan XST, proses *translate*, proses *mapping*, proses *place and route*, dan yang terakhir adalah proses pembentukan file *.bit.

3.7 Proses Pengujian

Proses pengujian ditujukan untuk mendapatkan informasi unjuk kerja dari tapis pada FPGA, pengujian dilakukan dengan merekam sinyal hasil penapisan yang keluar dari FPGA menggunakan perangkat lunak audacity, sambungan antara FPGA dan perangkat perekaman dihubungkan dengan kabel jack audio 3.5 mm, ilustrasi sambungan pada proses pengujian dapat dilihat pada gambar 3.4.

Unjuk kerja dari tapis didapatkan dari *magnitude* dan *roll-off* tanggapan frekuensi, *magnitude* tanggapan frekuensi untuk sinyal keluaran hasil penapisan pada FPGA didapatkan dari perbandingan nilai mutlak *peak* sinyal sebelum dan sesudah penapisan. Penyajian grafik dari magnitude tanggapan frekuensi dari keluaran tapis pada FPGA disandingkan dengan tanggapan frekuensi dari koefisien tapis FIR, gambar 3.9 berikut ini adalah ilustrasi dari proses pengujian



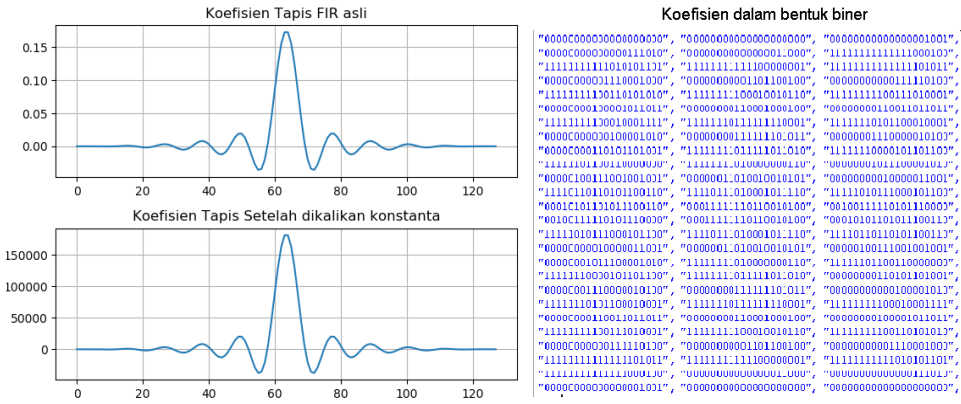
Gambar 3.9 Mendapatkan tanggapan frekuensi

Informasi *roll-off* didapatkan dari perubahan magnitude antar titik frekuensi yang terpaut satu decade, frekuensi tersebut berada pada sumbu frekuensi dalam nilai logaritmik.

BAB 4
HASIL DAN PEMBAHASAN

4.1 Hasil Perancangan Pada MATLAB

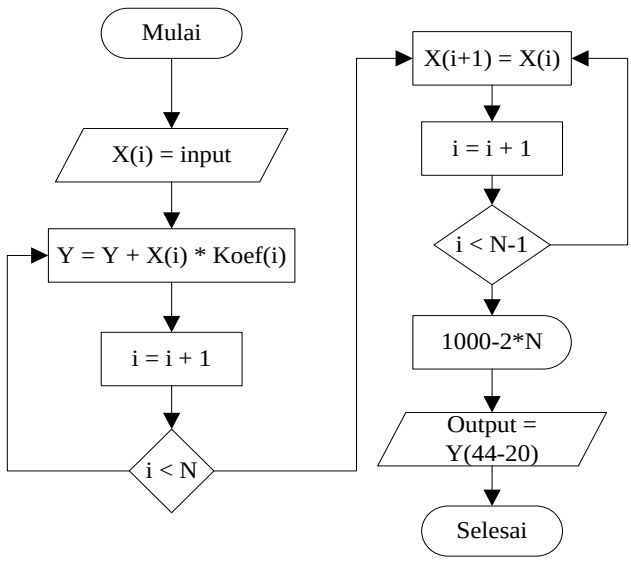
Hasil dari proses pada MATLAB adalah koefisien dari tapis *finite impulse response* dan transformasi *discrete cosine transform* dalam bentuk bilangan biner, gambar 4.1 berikut ini adalah koefisien tapis dalam bentuk.



Gambar 4.1 Proses pembentukan koefisien tapis dalam bentuk biner

4.2 Hasil Perancangan Pada FPGA

Perancangan pada FPGA akan dilakukan berdasarkan diagram alir, gambar 4.2 berikut ini adalah diagram alir dari tapis *finite impulse response* pada domain waktu.



Gambar 4.2 Diagram alir tapis FIR pada domain waktu

Keterangan:

input : masukan dari codec (24 bit)

X : penyimpanan untuk masukan dari codec (24 bit)

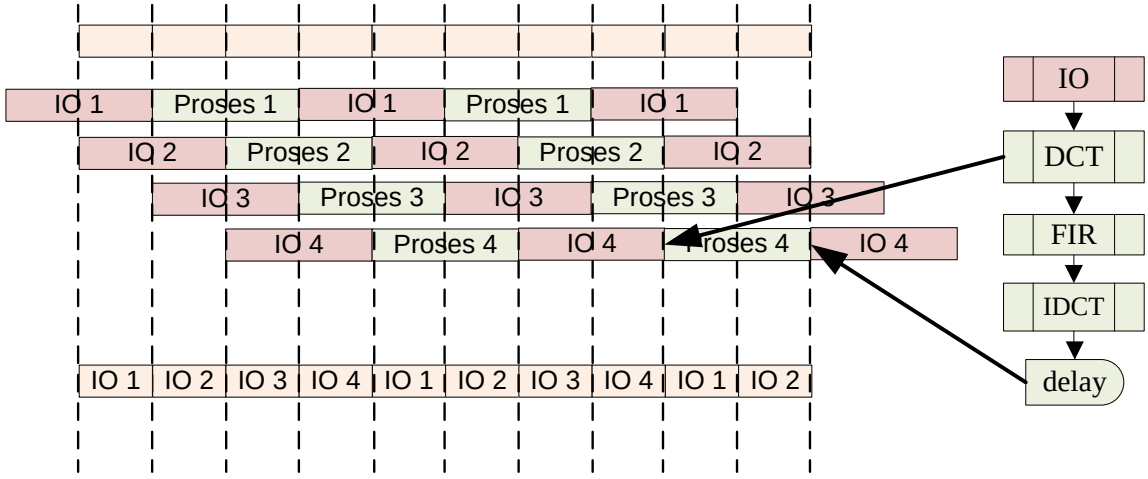
Koef : koefisien tapis dalam bentuk biner (20 bit)

Y : penyimpanan untuk masukan dari codec (44 bit)

output : keluaran menuju codec (24 bit)

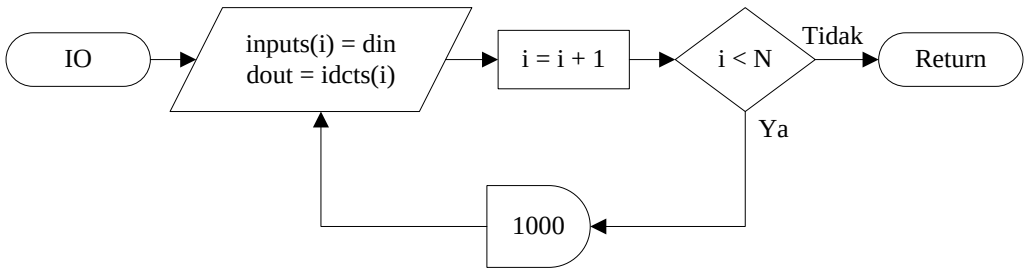
N : Jumlah tap tapis

Penerapan tapis pada domain frekuensi membutuhkan sebuah transformasi diskrit, pada penelitian ini transformasi yang digunakan adalah *discrete cosine transform*, gambar 4.3 berikut ini merupakan diagram alir dari proses penapisan pada domain frekuensi.



Gambar 4.3 Diagram alir penapisan pada domain frekuensi

Proses penapisan dalam domain frekuensi dimulai dengan proses masukan sekaligus proses keluaran, data masukan dan data keluaran terhubung secara langsung dengan bagian *codec* audio, gambar 4.4 berikut ini merupakan proses dari masukan dan keluaran.



Gambar 4.4 Proses masukan dan keluaran

Keterangan:

inputs : tempat masukan dari codec disimpan

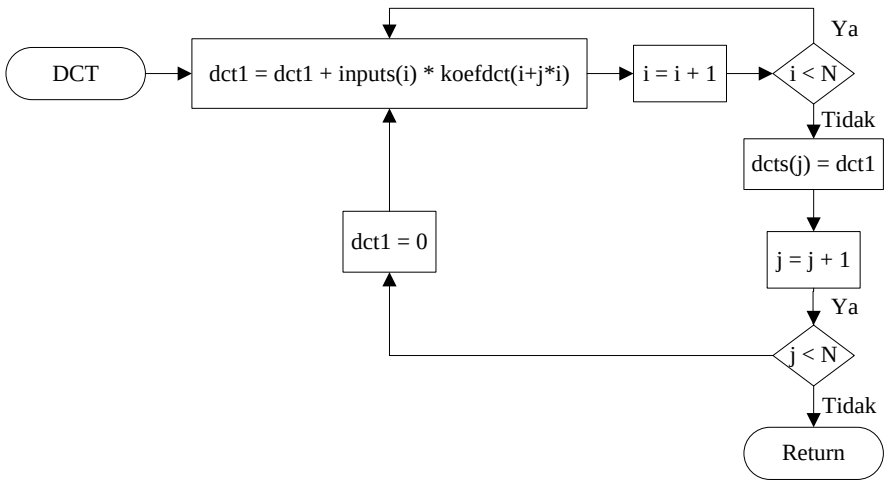
idcts : merupakan hasil dari inverse transformasi dan keseluruhan proses penapisan

din : jalur masukan dari codec

dout : jalur keluaran menuju codec

N : jumlah blok data yang diproses

Proses selanjutnya adalah proses DCT, proses ini adalah proses transformasi diskrit untuk data pada blok masukan yang telah disimpan pada array inputs, proses yang terjadi pada bagian ini adalah proses konvolusi antara data masukan audio dan keofisien transformasi yang telah dibuat pada MATLAB, gambar 4.5 berikut adalah diagram alir proses tersebut.



Gambar 4.5 Proses DCT penapisan domain frekuensi

Keterangan:

dcts : penyimpanan untuk hasil transformasi yang dilakukan

dct1 : penyimpanan sementara untuk memudahkan perhitungan

inputs : penyimpanan dari blok data masukan

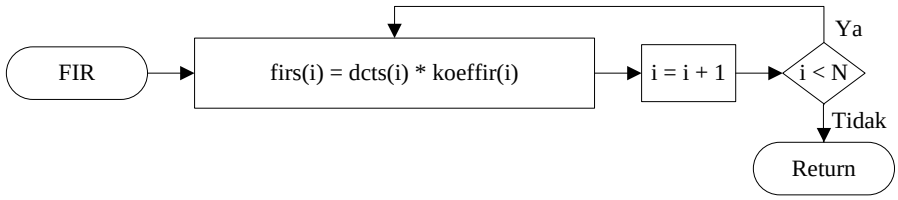
koefdct : koefisien pengali untuk transformasi agar memudahkan proses

N : Jumlah data dalam satu blok

j : urutan data dalam domain frekuensi

I : urutan data dalam domain waktu

Proses selanjutnya adalah proses penapisan, proses ini adalah perkalian antara koefisien tapis *finite impulse response* yang telah dibawa kedalam domain frekuensi dengan data hasil dari transformasi pada proses sebelumnya, gambar 4.6 berikut ini adalah proses penapisan tersebut.



Gambar 4.6 Proses penapisan pada domain frekuensi

Keterangan:

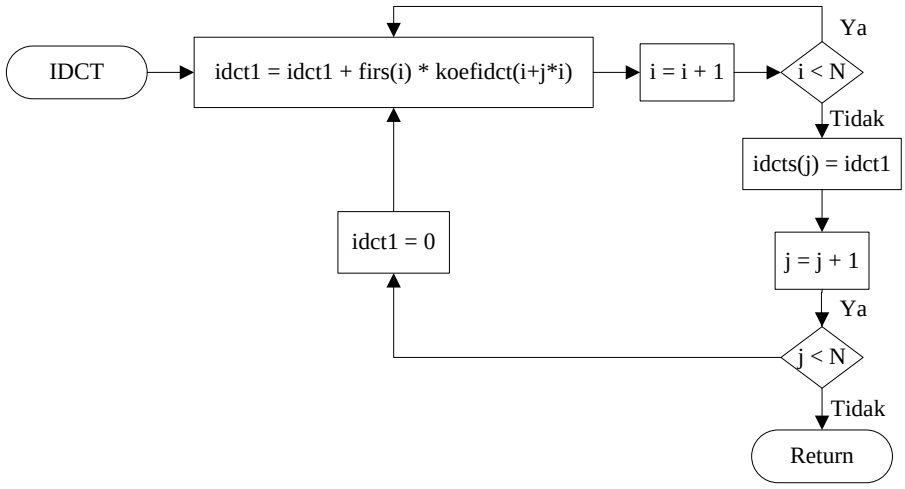
firs : penyimpanan untuk hasil penapisan domain frekuensi

koeffir : koefisien pengali untuk penapisan agar memudahkan proses

N : Jumlah data dalam satu blok

i : urutan data

Hasil dari penapisan pada proses gambar 4.6 masih berada pada domain frekuensi, data yang akan dikirim kepada *codec* audio harus berada pada domain waktu dan untuk mendapatkan sinyal dalam domain waktu diperlukan proses *inverse* dari transformasi DCT, gambar 4.7 berikut ini adalah proses *inverse* yang dilakukan.



Gambar 4.7 Proses IDCT penapisan domain frekuensi

Keterangan:

idcts : penyimpanan untuk hasil inverse transformasi yang dilakukan

dct1 : penyimpanan sementara untuk memudahkan perhitungan

koefidct : koefisien pengali untuk inverse transformasi agar memudahkan proses

N : Jumlah data dalam satu blok

J : urutan data dalam domain waktu

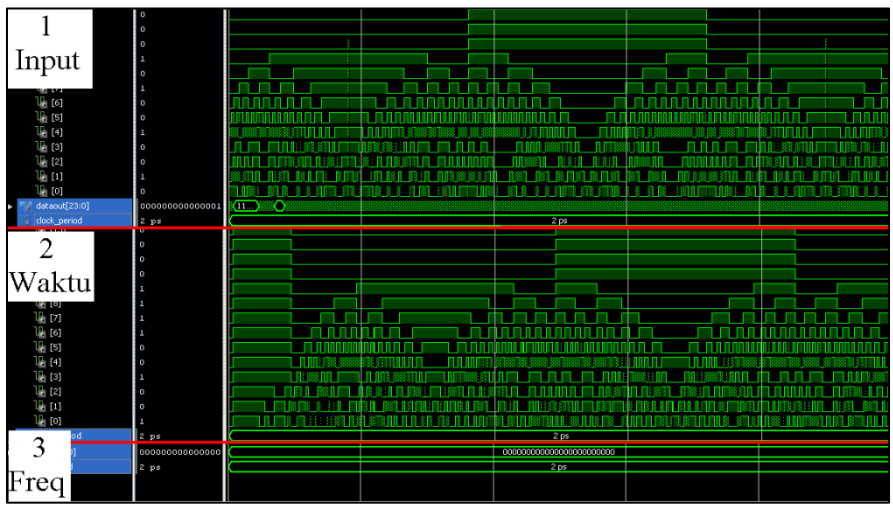
i : urutan data dalam domain frekuensi

1.3 Simulasi Hasil Perancangan

Simulasi dilakukan menggunakan perangkat lunak untuk mengetahui adanya kesalahan *syntaks* maupun kesalahan logika pada program VHDL yang ditulis, Untuk mengetahui adanya kesalahan pada *syntaks* pada program VHDL dapat dilihat pada *log* pada saat proses membuat simulasi pada perangkat lunak Isim, gambar 4.8 berikut ini adalah *log* dari proses pembuatan simulasi pada Isim.



Gambar 4.8 Informasi *Log* proses membentuk simulasi pada Isim



Gambar 4.9 Simulasi penapisan domain waktu pada perangkat lunak Isim

Terdapat 3 tahap pengecekan *error* yang akan dilakukan, pertama adalah pengecekan *syntaks error*, pengecekan ini bertujuan untuk mengetahui kesalahan *syntaks* pada program VHDL, gambar 4.8 menunjukan *log* proses membentuk simulasi pada perangkat lunak Isim, tidak adanya *warning* dan *error* pada *log* proses menunjukan bahwa semua *syntaks* telah sesuai dengan standard dari bahasa VHDL Sinyal pada simulasi yang berwarna hijau menunjukan bahwa tidak ada *runtime error* yang mungkin terjadi pada program VHDL. Kemungkinan

adanya *logical error* dapat dilihat menggunakan perbandingan antara bentuk sinyal masukan dan sinyal keluaran, sinyal masukan merupakan sinyal sinusoidal dengan frekuensi 400 Hz, normalnya sinyal ini tidak akan mengalami pelemahan karena masih dibawah frekuensi *cut off*, gambar 4.9 menunjukan bahwa tidak ada *logical error*.

1.4 Penggunaan *Resource* Dan Laporan *Timing Diagram*

Laporan penggunaan *resource* memberikan informasi tentang kemampuan dari FPGA untuk dapat menjalankan tapis yang telah dibuat, berikut ini adalah jumlah *resource* yang dimiliki oleh FPGA Xilinx Artix-7 XCA200T.

Register : 269.200
LUTs : 134.600
IOBs : 285
DSP48E1s : 740

Jumlah *register* diatas menunjukan batasan maksimal yang dapat digunakan oleh rancangan tapis digital pada FPGA, tabel 4.1 dan tabel 4.2berikut ini adalah laporan penggunaan *resource* perancangan tapis digital pada FPGA.

Tabel 4.1 Laporan *resource* tapis FIR

Jumlah tap	Frekuensi Cut off	Register	LUTs	IOBs	DSP48E1s
64	600	1782	1490	8	2
	700	1782	1489	8	2
	800	1782	1538	8	2
128	600	3350	2657	8	2
	700	3350	2658	8	2
	800	3350	2663	8	2
256	600	6541	4075	8	2
	700	6495	4194	8	2
	800	6517	4053	8	2

Penapisan derau dalam domain frekuensi yang diterapkan pada FPGA menggunakan spsifikasi yang sama dalam hal panjang blok yang diproses, panjang blok yang sama mengakibatkan *resources* dari rancangan ini tidak berbeda antara jumlah tap tapis yang satu dengan jumlah tap tapis yang lain.

Tabel 4.2 Laporan *resource* penapisan pada domain frekuensi dengan DCT

Jumlah tap	Frekuensi *Cut off*	Register	LUTs	IOBs	DSP48E1s
64	600	25201	11152	8	16
	700	25201	11152	8	16
	800	25201	11301	8	16
128	600	25201	11107	8	16
	700	25201	11192	8	16
	800	25201	11192	8	16
256	600	25205	11112	8	16
	700	25205	11112	8	16
	800	25205	11112	8	16

Penggunaan *resource* pada rancangan tapis FIR di domain waktu mengalami kenaikan seiring dengan kenaikan jumlah tap dan tidak mengalami perubahan pada perbedaan frekuensi *cut-off*, sedangkan pada rancangan tapis domain frekuensi yang menggunakan transformasi DCT tidak terdapat perbedaan yang berarti pada penggunaan *resource* karena jumlah data yang diproses setiap satu kali pemrosesan memiliki jumlah data yang sama sehingga jumlah *resource* yang digunakan juga sama.

Selain penggunaan *resource*, informasi lain yang perlu diperhatikan adalah *timing diagram*, tabel 4.3 dan 4.4 berikut adalah laporan *timing diagram* dari rancangan pada FPGA.

Tabel 4.3 Laporan *timing diagram* tapis FIR

Jumlah tap	Frekuensi Cut off	Periode (ns)	*Clock* (MHz)
64	600	8,75	114,23
	700	8,75	114,23
	800	8,78	113,90
128	600	9,06	110,39
	700	9,06	110,39
	800	9,06	110,39
256	600	9,12	109,69
	700	9,28	107,76
	800	9,25	108,06

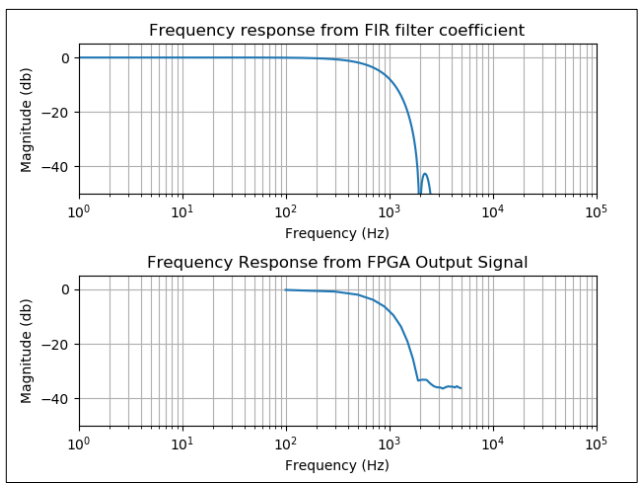
Tabel 4.4 Laporan *timing diagram* penapisan pada domain frekuensi dengan DCT

Jumlah tap	Frekuensi Cut off	Periode (ns)	*Clock* (MHz)
64	600	7,43	134,53
	700	7,43	134,53
	800	7,43	134,53
128	600	7,16	139,60
	700	7,16	139,60
	800	7,16	139,60
256	600	7,42	134,73
	700	7,42	134,73
	800	7,42	134,73

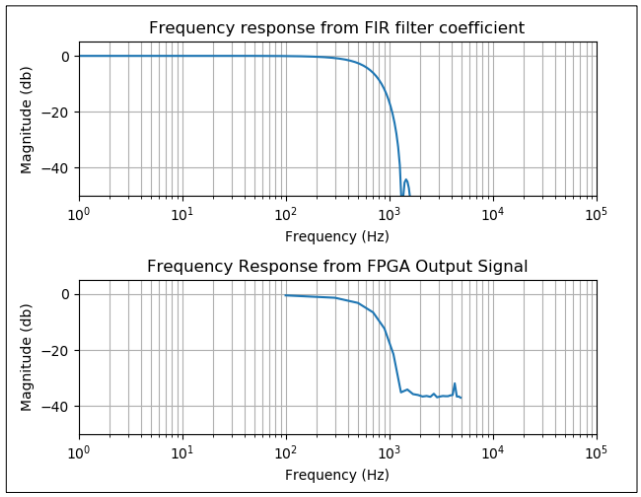
Sumber *clock* yang tersedia pada FPGA adalah 100 MHz, kecepatan *clock* tersebut merupakan kecepatan *clock* minimal yang harus dipenuhi oleh rancangan FPGA, laporan *timing diagram* tabel 4.3 dan 4.4 menunjukan bahwa rancangan telah memenuhi spesifikasi dari FPGA karena dapat dijalankan pada *clock* yang melebihi kecepatan *clock* minimal dari FPGA. Penggunaan *resources* dan laporan *timing diagram* yang dihasilakan tidak ada yang melebihi spesifikasi sehingga rancangan pada penelitian ini dapat diterapkan pada FPGA.

1.5 Unjuk Kerja Tapis Digital

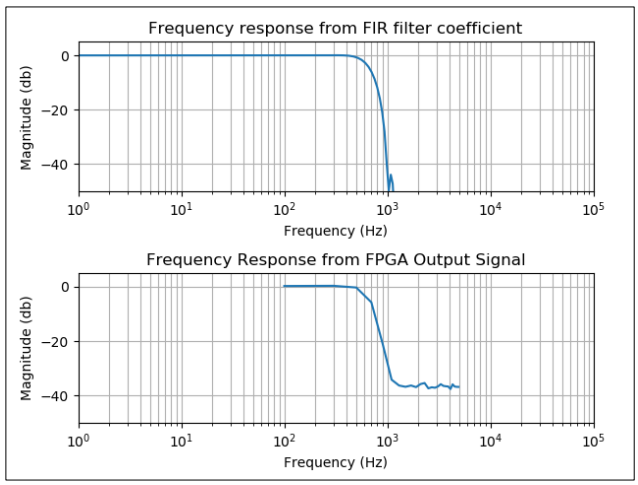
Analisi hasil perekaman audio keluaran dari FPGA dilakukan pada MATLAB, berikut ini adalah tanggapan frekuensi dari FPGA menggunakan MATLAB.



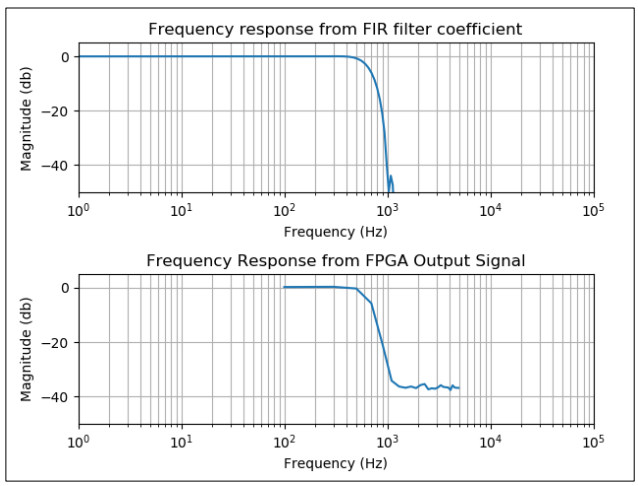
Gambar 4.10 Tanggapan frekuensi tapis FIR 64 Tap



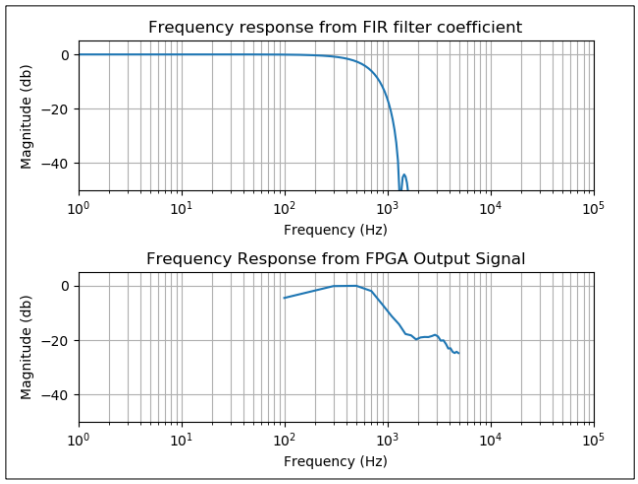
Gambar 4.11 Tanggapan frekuensi tapis FIR 128 Tap



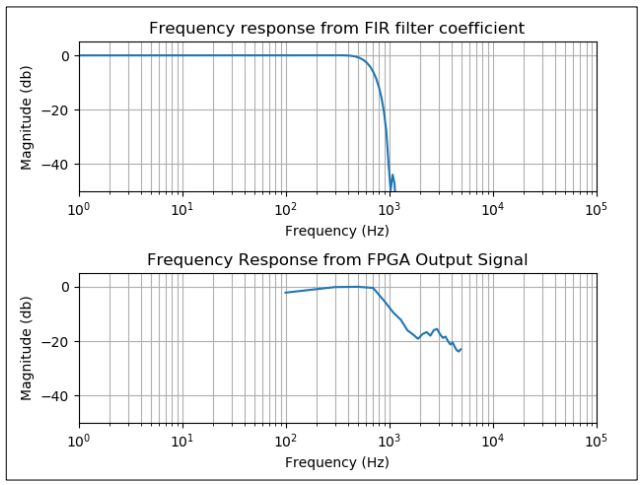
Gambar 4.12 Tanggapan frekuensi tapis FIR 256 Tap



Gambar 4.13 Tanggapan frekuensi penapisan domain frekuensi dengan DCT 64 Tap



Gambar 4.14 Tanggapan frekuensi penapisan domain frekuensi dengan DCT128 Tap



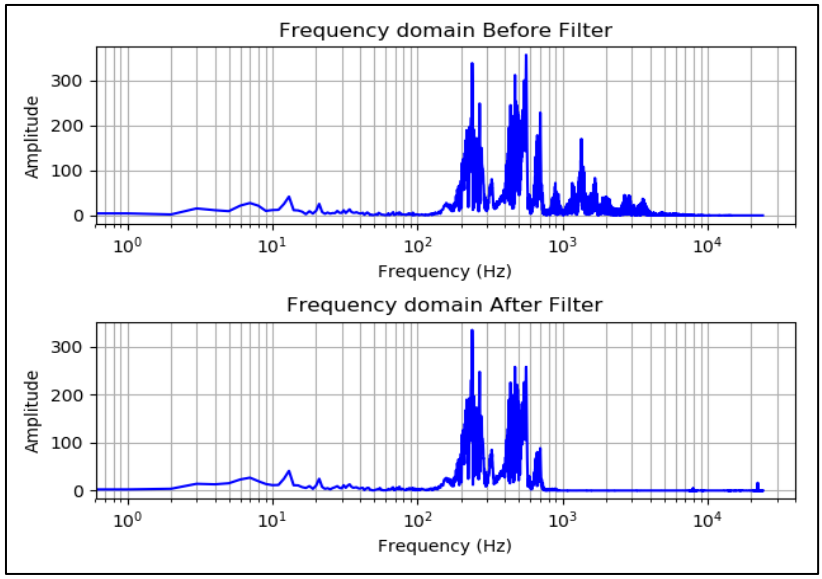
Gambar 4.15 Tanggapan frekuensi penapisan domain frekuensi dengan DCT 256 Tap

Gambar 4.10 sampai 4.15 merupakan grafik perbandingan antara tanggapan frekuensi koefisien tapis FIR dengan tanggapan frekuensi hasil perekaman dari FPGA, proses penapisan menggunakan FIR pada domain waktu merupakan proses yang sederhana dengan hanya melakukan konvolusi linier antara data masukan dan koefisien tap tapis FIR, penapisan pada domain waktu menghasilkan unjuk kerja yang baik dilihat dari kemiripan tanggapan frekuensi dari keluaran FPGA dan koefisien tap tapis FIR.

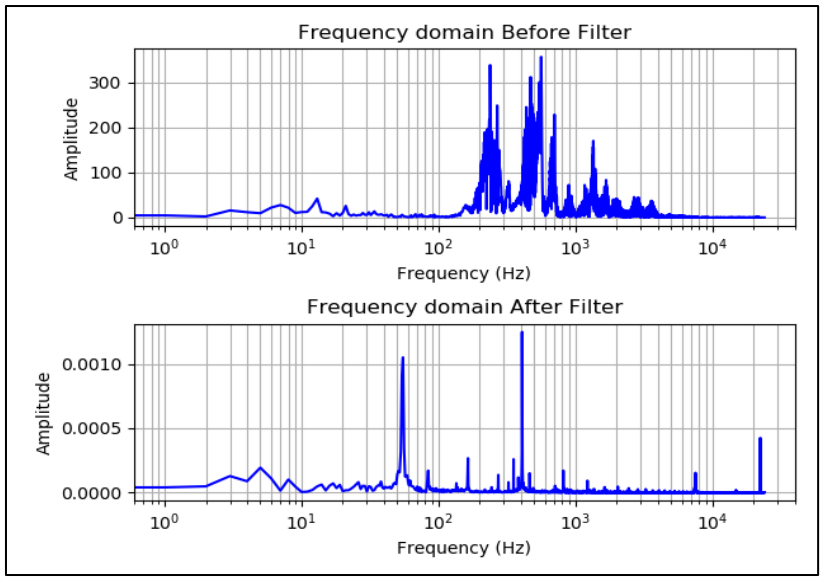
Proses penapisan pada domain frekuensi, jumlah data dalam satu blok masukan adalah 256 blok yang akan ditransformasikan kedalam 256 titik frekuensi antara 0 sampai 48000 Hz, sehingga jarak antara satu titik frekeunsi dengan titik yang lainya adalah 187 Hz, jarak yang jauh antar titik ini sangat berpengaruh pada proses penapisan.

Penapisan pada sinyal dengan bentuk gelombang sinus telah dibuktikan oleh gambar 4.10 sampai 4.15, gambar 4.16 dan 4.17 berikut ini merupakan tanggapan frekuensi dari

rekaman suara tertentu yang bukan gelombang sinus sebelum dan sesudah dilakukan penapisan pada FPGA.



Gambar 4.16 Tanggapan frekuensi penapisan pada domain waktu



Gambar 4.17 Tanggapan frekuensi penapisan pada domain frekuensi

Unjuk kerja dari tapis tidak hanya dapat dilihat dari kemiripan dari bentuk grafik tanggaan frekuensi, grafik tanggapan frekuensi dapat dilakukan analisi lebih lanjut dengan mencari *roll-off* dari masing masing grafik tanggapan frekuensi, tabel berikut ini adalah hasil dari mencari *roll-off* pada masing masing grafik

Tabel 4.5 *Roll-off* tanggapan frekuensi

Penapisan pada domain waktu			Penapisan pada domain frekuensi		
Jumlah tap	*Fc*	*Rolloff*	Jumlah tap	*Fc*	*Rolloff*
64	600	37	64	600	20
	700	36		700	18
	800	37		800	20
128	600	38	128	600	20
	700	38		700	20
	800	37		800	20
256	600	39	256	600	20
	700	39		700	20
	800	39		800	20

Berdasarkan *roll-off* yang terdapat pada tabel 4.5, penerapan pada domain waktu memiliki hasil yang lebih baik dengan peredaman yang lebih besar dari pada penerapan pada domain frekuensi.

Hasil yang buruk pada penapisan di domain frekuensi dapat diakibatkan oleh kesalahan *timing* dari masing masing proses, kesalahan ini membuat semua proses aritmatika tidak berjalan urut dan nilai yang dihasilkan pada keluaran menjadi melenceng.

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang dapat diambil pada penelitian ini adalah

1. Penapisan derau menggunakan tapis *finite impulse response* pada domain waktu dengan jumlah tap 64, 128, dan 256 dapat diterapkan pada FPGA Artix-7 XCA7000T berdasarkan *timing diagram* dan jumlah *resources* yang digunakan.
2. Penapisan derau pada domain frekuensi dengan menggunakan transformasi *discrete cosine transform* dapat diterapkan pada FPGA Artix-7 XCA7000T berdasarkan *timing diagram* dan jumlah *resources* yang digunakan.
3. Unjuk kerja yang ditunjukan oleh penapisan derau pada domain waktu dengan tapis *finite impulse response* melalui grafik tanggapan frekuensi menunjukan grafik yang mirip dengan koefisien yang ada pada MATLAB.
4. Unjuk kerja yang ditunjukan oleh penapisan derau pada domain frekuensi dengan menggunakan transformasi *discrete cosine transform* menujukan hasil yang berbeda dengan simulasi pada MATLAB, terjadi pelemahan pada frekuensi yang tidak sesuai dengan rancangan dan *roll-off* dari tapis yang kecil.s

5.2 Saran

1. Perancangan tapis *finite impulse response* pada domain waktu dapat dilakukan lebih tinggi pada jumlah tap diatas 256 dengan memperhatikan penggunaan *delay*, *resources*, dan *timing* dari *clock*.
2. Perancangan pada FPGA dapat dimaksimalkan dengan menggunakan simulator untuk VHDL yang mampu menampilkan bentuk analog dari sinyal dan mampu melakukan simulasi dengan waktu yang lebih lama.

DAFTAR PUSTAKA

[1] M.-P. Nitin Waghmare, "Comparison of Embedded System and FPGA Application for Universal Multi-Resource Bus," *Int. J. Adv. Res. Comput. Eng. Technol. IJARCET*, vol. 1, no. 7, p. 7, Sep. 2012.

[2] A. T. Falk Salewski, "*Fault Handling in FPGAs and Microcontrollers in Safety-Critical Embedded Applications: A Comparative Survey*," *10th Euromicro Conf. Digit. Syst. Des. Archit. Methods Tools*, p. 9, Aug. 2007.

[3] V. J. T. Jithin James, "*Audio Compression Using DCT and DWT Techniques*," *J. Inf. Eng. Appl.*, vol. 4, no. 4, p. 7, 2014.

[4] A. E. Putra and S. N. Azhar, "Implementasi Sistem Penghilang Derau Adaptif menggunakan Algoritma LMS pada FPGA Altera," pp. 1–11, 2011.

[5] D. J. Allred, W. Huang, V. Krishnan, and D. V. Anderson, "*An FPGA Implementation for a High Throughput Adaptive Filter Using Distributed Arithmetic*," *12th Annu. IEEE Symp. Field-Program. Cust. Comput. Mach.*, no. 4, pp. 324–325, 2004.

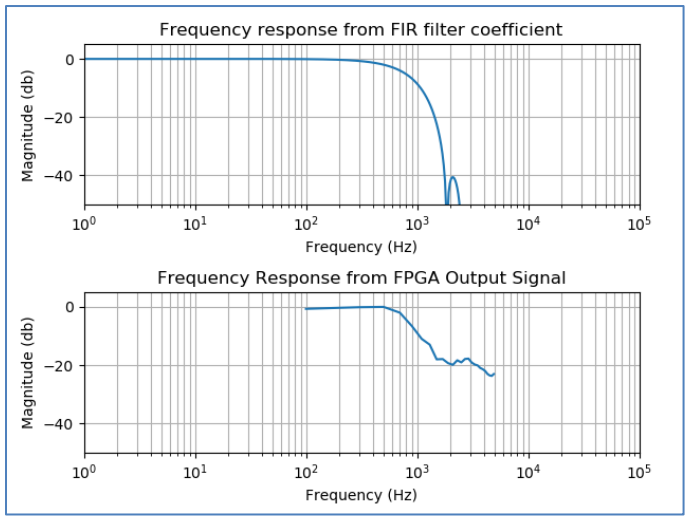
[6] Y. Zhou and P. Shi, "*Distributed arithmetic for FIR filter implementation on FPGA*," *2011 Int. Conf. Multimed. Technol. ICMT 2011*, vol. 1, no. 4, pp. 294–297, 2011.

[7] E. Imam, M. E. M. Ahmed, and G. Abdalla, "*Design and implementation of discrete cosine transform algorithm on FPGA device*," 2016, pp. 13–18.

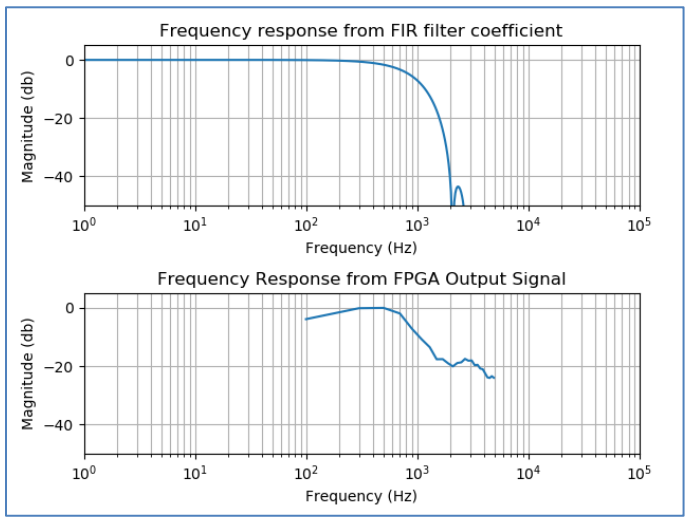
[8] John G. Proakis, Dimitris K Manolakis "*Digital Signal Processing_ Principles, Algorithms and Applications -Prentice Hall",* (1995).

LAMPIRAN

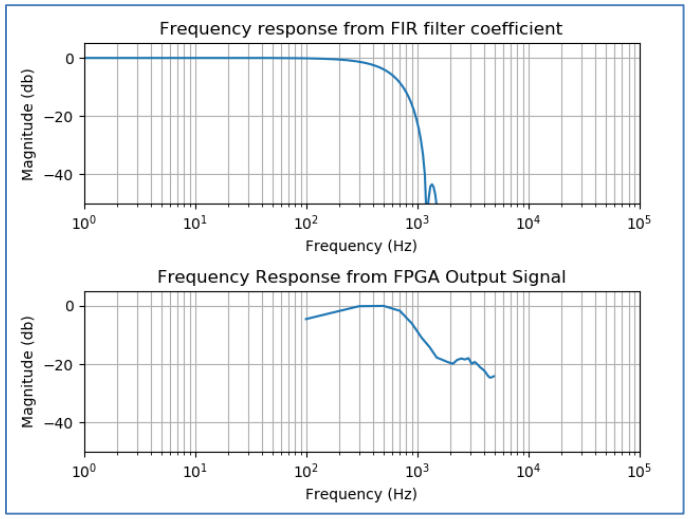
Berikut ini adalah lampiran gambar untuk tanggapan frekeunsi dari penapisan pada domain frekeunsi.



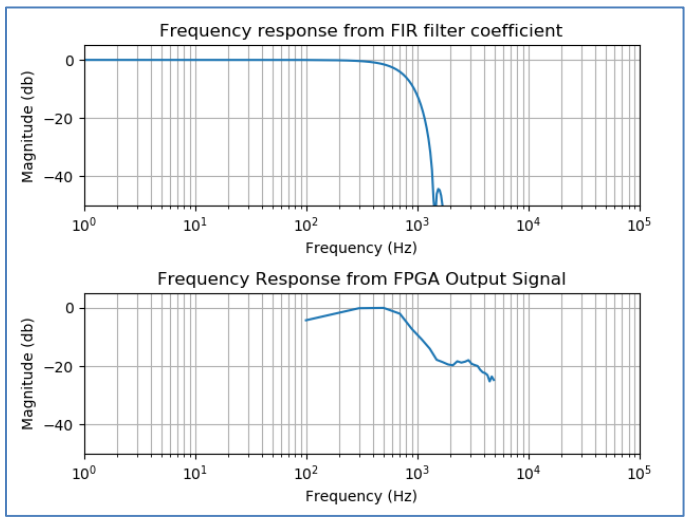
Gambar 1 Tanggapan frekeunsi pada tapis 64 tap dan frekuensi *cut-off* 600 Hz



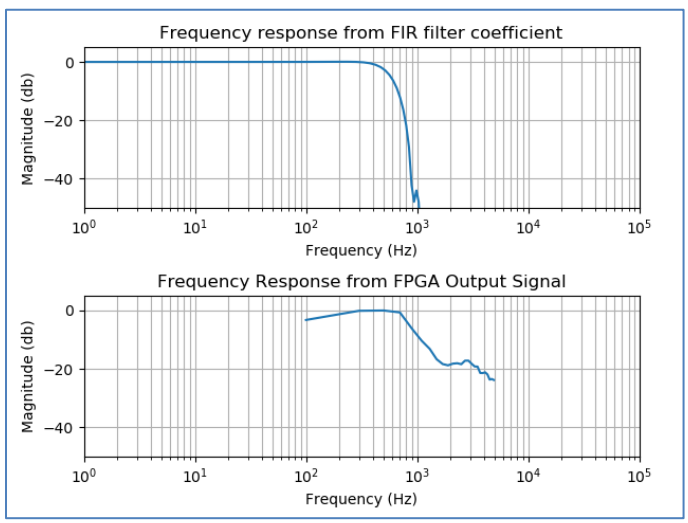
Gambar 2 Tanggapan frekeunsi pada tapis 64 tap dan frekuensi *cut-off* 800 Hz



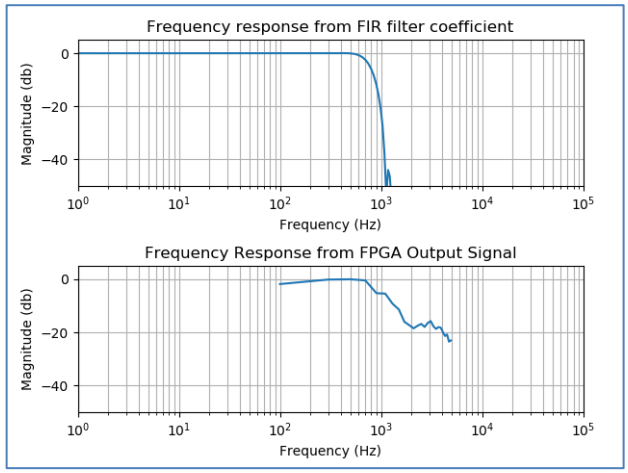
Gambar 3 Tanggapan frekeunsi pada tapis 128 tap dan frekuensi *cut-off* 600 Hz



Gambar 4Tanggapan frekeunsi pada tapis 128 tap dan frekuensi *cut-off* 800 Hz

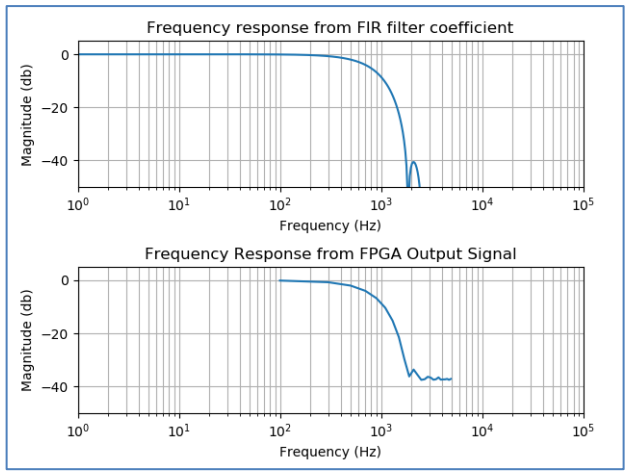


Gambar 5 Tanggapan frekeunsi pada tapis 256 tap dan frekuensi *cut-off* 600 Hz

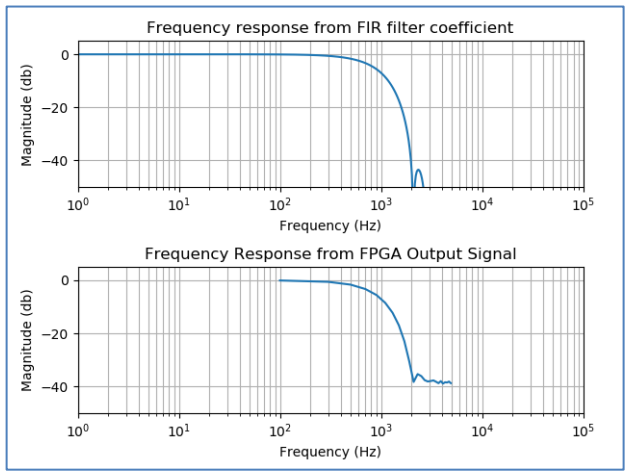


Gambar 6 Tanggapan frekeunsi pada tapis 256 tap dan frekuensi *cut-off* 800 Hz

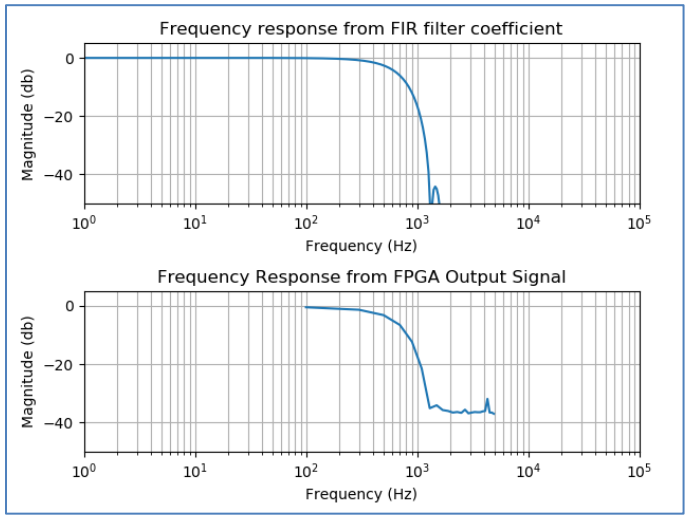
Berikut ini adalah lampiran gambar untuk tanggapan frekuensi dari penapisan pada domain waktu.



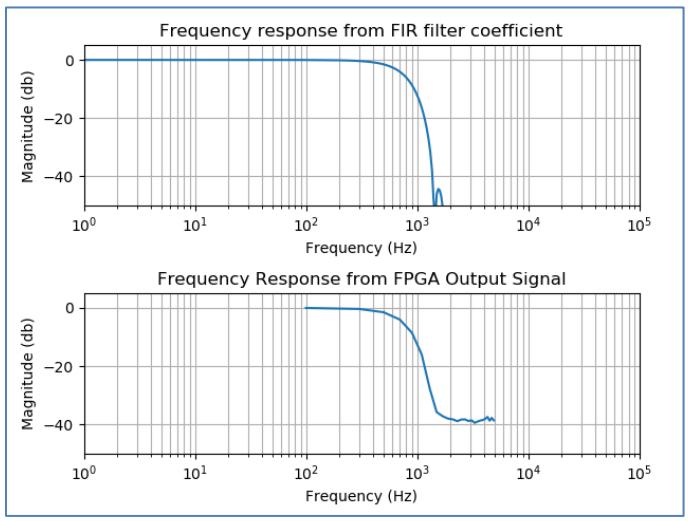
Gambar 7 Tanggapan frekeunsi pada tapis 64 tap dan frekuensi *cut-off* 600 Hz



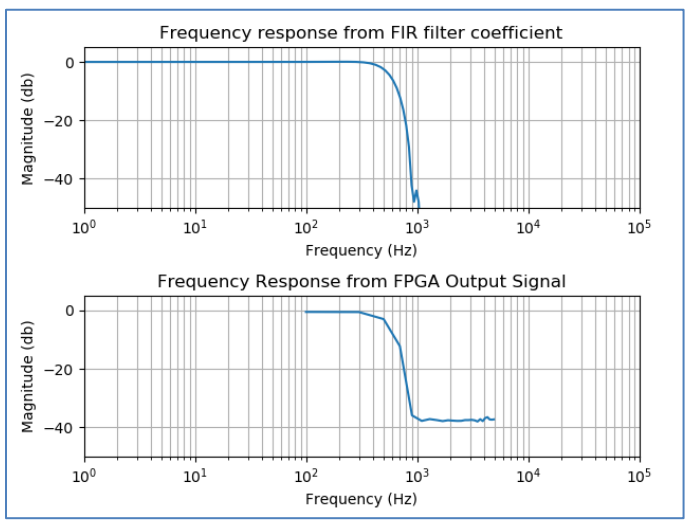
Gambar 8 Tanggapan frekeunsi pada tapis 64 tap dan frekuensi *cut-off* 800 Hz



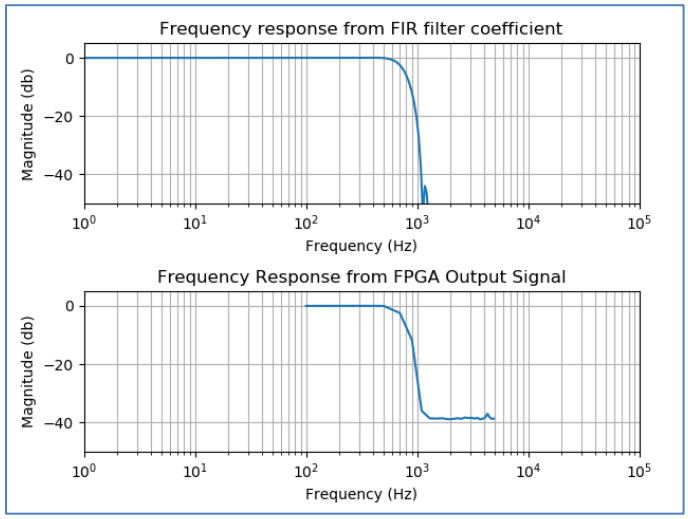
Gambar 9 Tanggapan frekeunsi pada tapis 128 tap dan frekuensi *cut-off* 600 Hz



Gambar 10 Tanggapan frekeunsi pada tapis 128 tap dan frekuensi *cut-off* 800 Hz



Gambar 11 Tanggapan frekeunsi pada tapis 256 tap dan frekuensi *cut-off* 600 Hz



Gambar 12 Tanggapan frekeunsi pada tapis 256 tap dan frekuensi *cut-off* 800 Hz

Berikut ini adalah lampiran dari sintaks program VHDL untuk bagian penapisan pada domain waktu menggunakan *finite impulse response*.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity filter256 is port(
  clock : in std_logic;
  datain : in std_logic_vector(23 downto 0);
  dataout : out std_logic_vector(23 downto 0));
end filter256;
architecture Behavioral of filter256 is
  signal i : integer := 0;
  type statelist is(filters, shift, waits);
  signal state : statelist := filters;
  type taplist is array(255 downto 0) of signed(23 downto 0);
  signal tap : taplist := (others => (others => '0'));
  signal filter : signed(43 downto 0) := (others => '0');
  type koeflist is array(255 downto 0) of signed(19 downto 0);
  signalkoef : koeflist:=(
```

"00000000000000000000”,”00000000000000000000”,”00000000000000000000”,”11111111111111111111”,”11111111111111111111”,”111
11111111111111110”,”11111111111111111111”,”00000000000000000000”,”00000000000000000001”,”00000000000000000100”,”000000
00000000001000”,”00000000000000001110”,”00000000000000010101”,”00000000000000011111”,”00000000000000101010”,”000000000
00000110111”,”00000000000001000111”,”00000000000001011000”,”00000000000001101100”,”00000000000010000010”,”000000000000
10011010”,”00000000000010110011”,”00000000000011001110”,”00000000000011101010”,”00000000000100001000”,”000000000001001
00110”,”00000000000101000100”,”00000000000101100010”,”00000000000110000000”,”00000000000110011100”,”000000000011011011
11”,”00000000000111001111”,”00000000000111100101”,”00000000000111110111”,”00000000001000000100”,”00000000001000001101”,
”00000000001000010001”,”00000000001000001110”,”00000000001000000101”,”00000000000111110100”,”00000000000111011011”,”000
00000000110111010”,”00000000000110010001”,”00000000000101011110”,”00000000000100100001”,”00000000000011011011”,”000000
00000010001011”,”00000000000000110000”,”11111111111111001101”,”11111111111101011111”,”11111111111011100111”,”111111111

11001100110”,”1111111110111011100”,”1111111110101001010”,”1111111110010110000”,”1111111110000010000”,”111111111011
01101010”,”1111111101011000000”,”1111111101000010010”,”1111111100101100001”,”1111111100010110001”,”111111111000000
00001”,”1111111011101010100”,”1111111011010101011”,”1111111011000001000”,”1111111010101101101”,”111111101001101110
00”,”1111111010001010110”,”1111111001111011111”,”1111111001101111000”,”1111111001100100011”,”1111111001011100010”,
”1111111001010110111”,”1111111001010100101”,”1111111001010101100”,”1111111001011010000”,”1111111001100010001”,”111
1111001101110010”,”1111111001111110011”,”1111111010010010111”,”1111111010101011111”,”1111111011001001100”,”111111
11011101011111”,”1111111100010011000”,”1111111100111111000”,”1111111101110000000”,”1111111110100101111”,”111111111
11100000110”,”0000000000100000011”,”0000000000100101000”,”0000000010101110011”,”0000000011111100011”,”000000001010
01110110”,”0000000010100101100”,”0000000100000000001”,”0000000100101111010”,”0000000101100000101”,”000000011001001
01111”,”0000000111000110000”,”0000000111111000110”,”0000001000110101100”,”0000001001101010001”,”0000001010100010000
01”,”0000001011011010111”,”0000001100010011010”,”0000001101001011101”,”0000001110000010000”,”0000001110111100100”,
”0000001111110011001”,”0000010000101011110”,”0000010001011110001”,”0000010010010101101”,”0000010011000100111”,”000
0010011101010011”,”0000010100100011010”,”0000010101001111010”,”0000010101011110000111”,”0000010110011110001”,”000001
01110000101100”,”0000101111000110111”,”0000011000000001110”,”0000011000011010111”,”0000011000110011000”,”000001100
10001001000”,”0000011001010011101”,”0000011001011110110”,”0000011001100111000”,”0000011001101010111”,”000001100110
10101111”,”0000011001100111000”,”0000011001011110110”,”0000011001010011101”,”0000011001000100100”,”000001100011000
11000”,”0000011000011010111”,”0000011000000001110”,”0000010111100011011”,”0000010111000010110”,”000001011001111100
01”,”0000010101111000011”,”0000010101001111010”,”0000010100100011010”,”0000010011101010011”,”0000010011000100111”,
”0000010010010111010”,”0000010001011110001”,”0000010000101001111010”,”0000011111100111001”,”0000001110111100100”,”000
0001110000100100”,”0000011010010111101”,”0000011000100110010”,”0000010110110100111”,”0000010101000100001”,”000000
10011010100001”,”0000010001100101100”,”0000001111111000110”,”0000001110001110000”,”0000001100100101111”,”000000010
11000000101”,”0000001001011110101”,”0000000100000000001”,”0000000110100101100”,”0000000010100111010”,”000000000111
11100011”,”0000000010101110011”,”0000000000100101000”,”0000000000100000011”,”1111111111100000110”,”111111111010010
01111”,”1111111101110000000”,”1111111100111111000”,”1111111100010011000”,”1111111011101011111”,”1111111010110011
00”,”1111111010101011111”,”1111111010010010111”,”1111111001111110011”,”1111111001101110010”,”1111111001100010001”,
”1111111001011010000”,”1111111001010101100”,”1111111001010100101”,”1111111001010110111”,”1111111001011100010”,”111
1111001100100011”,”1111111100110111000”,”1111111001111011111”,”1111111010001010110”,”1111111010011011100”,”111111
11010101101101”,”1111111011000001000”,”1111111011010101011”,”1111111011101010100”,”1111111100000000001”,”111111111
00010110001”,”1111111100101100001”,”1111111101000010010”,”1111111101011000000”,”1111111101101101010”,”111111111100
00010000”,”1111111110010110000”,”1111111110101001010”,”1111111110111011100”,”1111111111001100110”,”111111111110111
00111”,”1111111111101011111”,”1111111111111001101”,”0000000000000110000”,”0000000000010001011”,”000000000000110110
11”,”0000000000100100001”,”0000000000101011110”,”0000000000110010001”,”0000000000110111010”,”0000000000111011011”,
”0000000000111110100”,”0000000001000000101”,”0000000001000001110”,”0000000001000010001”,”0000000001000001101”,”000
0000001000000100”,”0000000000111110111”,”0000000000111100101”,”0000000000111001111”,”0000000000110110111”,”000000
00000110011100”,”0000000000110000000”,”0000000000101100010”,”0000000000101000100”,”0000000000100100110”,”000000000
00100001000”,”0000000000011101010”,”0000000000011001110”,”0000000000010110011”,”0000000000010011010”,”000000000000
10000010”,”0000000000001101100”,”0000000000001011000”,”0000000000001000111”,”0000000000000110111”,”000000000000001
01010”,”0000000000000011111”,”0000000000000010101”,”0000000000000001110”,”0000000000000001000”,”000000000000000001
00”,”0000000000000000001”,”0000000000000000000”,”1111111111111111111”,”1111111111111111110”,”1111111111111111111”,
”1111111111111111111”,”0000000000000000000”,”0000000000000000000”,”0000000000000000000};

```
begin
  proseskoefisien:process(i, clock, state, koef)
  begin
        if rising_edge(clock) then
              case state is
              when filters => if i < 256 then
                                    if i = 0 then
                                          i <= i + 1;
                                          filter <= koef(i) * tap(i);
                                    elsif i = 255 then
                                          state <= shift;
                                          filter <= filter + koef(i) * tap(i);
                                    else
                                          i <= i + 1;
                                          filter <= filter + koef(i) * tap(i);
                                    end if;
```

```
                end if;
            when shift => if i > 0 or i = 0 then
                        if i = 0 then
                            state <= waits;
                            tap(i) <= signed(datain);
                            dataout <= std_logic_vector(filter(43 downto 20));
                        else
                            tap(i) <= tap(i - 1);
                            i <= i - 1;
                        end if;
                end if;
            when waits =>if i < 488 then
                        if i = 487 then
                            i <= 0;
                            state <= filters;
                        else
                            i <= i + 1;
                        end if;
                end if;
            when others => state <= filters;
            end case;
        end if;
    end process;
    end Behavioral;
```

Berikut ini adalah sintaks dari program VHDL untuk penapisan pada domain frekuensi dengan menggunakan transformasi diskrit *discrete cosine transform.*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity dsp_1 is
   Port ( clk : in  STD_LOGIC;
        sin : in  STD_LOGIC;
        din : in  STD_LOGIC_VECTOR (23 downto 0);
        dout : out  STD_LOGIC_VECTOR (23 downto 0));
end dsp_1;
```

```
architecture Behavioral of dsp_1 is
  component koefmem is port(
            clka : IN  std_logic;
            addra : IN  std_logic_vector(15 downto 0);
            douta : OUT  std_logic_vector(11 downto 0));
  end component;
  type firkoefmem is array(255 downto 0) of signed(17 downto 0);
  signal dfir : firkoefmem := (
"100000000000000000", "011101010110001100", "010101111101111100", "001011011101001001", "000000000000000000", "110101110101111100",
"101110101111100101", "101011100111001010", "101100011001000000", "110000001100101000", "110101101010000010", "111011010100001010",
"000000000000000000", "000011000011000101", "000100010110010010", "000100001110101101", "000011010000101000", "000010000001011100",
"000000111101001101", "000000010001110100", "000000000000000000", "111111111111111011", "000000000111001010", "000000001101011100",
"000000001110110101", "000000001011100100", "000000000110010100", "000000000001111010", "000000000000000000", "000000000000011111",
"000000000010001100", "000000000011101001", "000000000011111110", "000000000011000111", "000000000001101100", "000000000000011110",
"000000000000000000", "000000000000010000", "000000000000111001", "000000000001011011", "000000000001100010", "000000000001001101",
"000000000000101001", "000000000000001010", "000000000000000000", "000000000000001001", "000000000000011100", "000000000000101100",
"000000000000101111", "000000000000100100", "000000000000010011", "000000000000000100", "000000000000000000", "000000000000000101",
"000000000000001111", "000000000000011000", "000000000000011001", "000000000000010011", "000000000000001010", "000000000000000010",
"000000000000000000", "000000000000000011", "000000000000001001", "000000000000001110", "000000000000001111", "000000000000001011",
"000000000000000110", "000000000000000001", "000000000000000000", "000000000000000010", "000000000000000110", "000000000000001001",
"000000000000001001", "000000000000000111", "000000000000000011", "000000000000000000", "000000000000000000", "000000000000000001",
"000000000000000100", "000000000000000110", "000000000000000110", "000000000000000101", "000000000000000010", "000000000000000000",
"000000000000000000", "000000000000000001", "000000000000000011", "000000000000000100", "000000000000000100", "000000000000000011",
"000000000000000001", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000010", "000000000000000011",
"000000000000000011", "000000000000000010", "000000000000000001", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000001", "000000000000000010", "000000000000000010", "000000000000000010", "000000000000000001", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000001", "000000000000000001", "000000000000000010", "000000000000000001",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000001",
"000000000000000001", "000000000000000001", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000001", "000000000000000001", "000000000000000001", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000001", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000",
"000000000000000000", "000000000000000000", "000000000000000000", "000000000000000000");
  type inputbank is array(255 downto 0) of signed(23 downto 0);
  signal inputs : inputbank;
  signal dct1 : signed(35 downto 0) := (others => '0');
  signal dct2 : signed(36 downto 0) := (others => '0');
  type dctbank is array(255 downto 0) of signed(26 downto 0);
  signal dcts : dctbank;
  signal fir1 : signed(44 downto 0) := (others => '0');
  type firbank is array(255 downto 0) of signed(28 downto 0);
  signal firs : firbank;
```

```
  signal idct0 : signed(40 downto 0) := (others => '0');
  signal idct1 : signed(40 downto 0) := (others => '0');
  signal idct2 : signed(40 downto 0) := (others => '0');
  signal idct3 : signed(23 downto 0) := (others => '0');
  type idctbank is array(255 downto 0) of signed(23 downto 0);
  signal idcts : idctbank:= (others => (others => '0'));
  signal koef : std_logic_vector(11 downto 0) := (others => '0');
  signal add : std_logic_vector(15 downto 0) := (others => '0');
  signal addr : signed(16 downto 0):= (others => '0');
  signal i, j, k, l : integer := 0;
  signal en : std_logic := '0';
  type statelist is (idle, io, dct, fir, idct, jeda);
  signal state : statelist := idle;
begin
  Koefisien:koefmem port map(clka => clk, addra => add, douta => koef);
  Proses:process(clk, en, j, k, din, en, sin, dfir, koef)
  begin
        if rising_edge(clk) then
              case state iswhen idle => if sin = '1' then
                          state <= io;
                          dout <= "000000000000000000000000";
                    end if;
              when io =>if en = '1' then
                          if j < 257 then
                                if j = 256 then
                                      state <= dct;
                                      j <= 0;
                                else
                                      inputs(j) <= signed(din);
                                      dout <= std_logic_vector(idcts(j));
                                      j <= j + 1;
                                end if;
                          end if;
                    end if;
              when dct =>if j < 257 then
                          if j = 256 then
                                j <= 0;
```

```
                addr <= (others => '0');
                state <= fir;
            else
                if k < 257 then
                    if k = 256 then
                        k <= 0;
                        j <= j + 1;
                        dct2(36 downto 1) <= dct1;
                        dcts(j) <= dct2(36 downto 10);
                    elsif k = 0 then
                        addr <= addr + "00000000000000001";
                        dct1 <= signed(koef) * inputs(k);
                        k <= k + 1;
                    else
                        addr <= addr + "00000000000000001";
                        dct1 <= dct1 + signed(koef) * inputs(k);
                        k <= k + 1;
                    end if;
                end if;
            end if;
        end if;
    when fir =>
        if k < 257 then
            if k = 256 then
                k <= 0;
                j <= j + 1;
                state <= idct;
            elsif k = 0 then
                fir1 <= dfir(k) * dcts(k);
                firs(k) <= fir1(44 downto 16);
                idct0 <= fir1(44 downto 4);
                k <= k + 1;
            else
                fir1 <= dfir(k) * dcts(k);
                firs(k) <= fir1(44 downto 16);
                k <= k + 1;
            end if;
```

```
        end if;
when idct =>if j < 257 then
                if j = 256 then
                        j <= 0;
                        addr <= (others => '0');
                        state <= jeda;
                else
                        if k < 257 then
                                if k = 256 then
                                k <= 0;
                                j <= j + 1;
                                idct2 <= idct0 + idct1;
                                idct3(22 downto 0) <= idct2(40 downto 18);
                                idct3(23) <= idct2(40);
                                idcts(j) <= idct3;
                                elsif k = 0 then
                                        addr <= addr + "00000000000000001";
                                        idct1 <= signed(koef) * firs(k);
                                        k <= k + 1;
                                else
                                        addr <= addr + "00000000000000001";
                                        idct1 <= idct1 + signed(koef) * firs(k);
                                        k <= k + 1;
                                end if;
                        end if;
                end if;
        end if;
when jeda =>
        if k < 124673 then
                if k = 124673 then
                        k <= 0;
                        state <= idle;
                else
                        k <= k + 1;
                end if;
        end if;
when others => state <= idle;
```

```
                end case;
        end if;
  end process;
  add <= std_logic_vector(addr(15 downto 0));
  Enable:process(clk, i)
  begin
        if rising_edge(clk) then
                if i < 1001 then
                        if i = 1000 then
                                i <= 0;
                        else
                                i <= i + 1;
                                en <= '0';
                                if i = 0 then
                                        en <= '1';
                                end if;
                        end if;
                end if;
        end if;
  end process;
end Behavioral;
```