UNIVERSITAS INDONESIA

PEMBANGKITAN SINYAL KENDALI ANALOG UNTUK PENGENDALIAN MOTOR DENGAN ANTARMUKA PC BERBASIS FPGA (MESA 5I20) DAN SERVO AMPLIFIER (MESA 7I33)

SKRIPSI

CANDRADITYA PRADHANA

0606073820

FAKULTAS TEKNIK UNIVERSITAS INDONESIA

TEKNIK ELEKTRO

DEPOK

Juni 2010

UNIVERSITAS INDONESIA

PEMBANGKITAN SINYAL KENDALI ANALOG UNTUK PENGENDALIAN MOTOR DENGAN ANTARMUKA PC BERBASIS FPGA (MESA 5I20) DAN SERVO AMPLIFIER (MESA 7I33)

SKRIPSI

Diajukan sebagai salah satu syarat memperoleh gelar sarjana

CANDRADITYA PRADHANA

0606073820

FAKULTAS TEKNIK

PROGRAM STUDI TEKNIK ELEKTRO

DEPOK

JUNI 2010

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk telah saya
nyatakan dengan benar.

Nama	**: Candraditya Pradhana**
NPM	**: 0606073820**
Tanda Tangan	**:**
Tanggal	**: 15 Juni 2010**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh :

Nama	: Candraditya Pradhana
NPM	: 0606073820
Program Studi	: Teknik Elektro
Judul Skripsi	: Pembangkitan Sinyal Kendali Analog untuk Pengendalian Motor dengan Antarmuka PC Berbasis FPGA (mesa 5I20) dan Servo Amplifier (mesa 7I33)

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Teknik pada Program Studi Teknik Elektro, Fakultas Teknik, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing :
Ir. Wahidin Wahab, M.Sc, Ph.D ( )

Penguji 1 :
Dr. Abdul Muis, S.T, M.Eng. ( )

Penguji 2 :
Dr. Abdul Halim, M.Eng ( )

Ditetapkan di : Depok

Tanggal : 2 Juli 2010

UCAPAN TERIMA KASIH

Dengan selesainya penulisan tugas seminar ini penulis bersyukur kepada Allah SWT atas karunia-Nya dan penulis juga mengucapkan terima kasih kepada:

1. Bapak Wahidin Wahab dan Bapak Abdul Muis selaku pembimbing dalam skripsi ini, yang telah membimbing saya dan teman-teman selama penulisan seminar ini
2. Kedua orangtua saya, Bapak Samasta Pradhana dan Ibu Endang Titi Purwani, serta kedua kakak saya, Anindya Pradhana dan Bhaswara Pradhana, serta Dewi Rizki yang telah memberikan doa, dukungan, serta kehangatan selama ini.
3. Bapak Lie yang telah membimbing serta membantu pula dalam sharing pengetahuan serta diskusi mengenai rancangan mesin CNC ini
4. Kepada teman-teman kelompok saya yang selalu membantu dalam penulisan seminar ini, Ferdi Andika dan Deni Umaryadi
5. Teman-teman Elektro, Tim Robot UI, IME FTUI, dan teman-teman semua yang tidak dapat saya tulis semuanya, terimakasih untuk semuanya.

Keluarga besar saya, eyang-eyang dan sepupu-sepupu semua, yang telah memberikan dukungan, nesehat, semangat, dan lainnya.

Depok, Juni 2010

Candraditya Pradhana

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI
TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademika Universitas Indonesia, saya bertanda tangan di bawah ini :

Nama	: Candraditya Pradhana
NPM	: 0606073820
Program studi	: Teknik Elektro
Departemen	: Teknik Elektro
Fakultas	: Teknik
Jenis karya	: Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty Free Right*)** atas karya ilmiah saya yang berjudul :

PEMBANGKITAN SINYAL KENDALI ANALOG UNTUK PENGENDALIAN MOTOR DENGAN ANTARMUKA PC BERBASIS FPGA (MESA 5I20) DAN SERVO AMPLIFIER (MESA 7I33)

Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non Eksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia / formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta sebagai pemegang Hak Cipta.
Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 15 Juni 2010
Yang menyatakan

Candraditya Pradhana

ABSTRAK

Nama : Candraditya Pradhana

Program studi : Teknik Elektro

Judul : Pembangkitan Sinyal Kendali Analog untuk Pengendalian Motor dengan Antarmuka PC Berbasis FPGA (Mesa 5I20) dan *Servo Amplifier* (Mesa 7I33)

Pengendalian motor pada mesin CNC sangatlah penting dikarenakan mesin ini membutuhkan ketelitian yang sangat tinggi dengan kecepatan proses yang cukup cepat. Dengan demikian dibutuhkan divais-divais yang menunjang dalam proses tersebut. Salah satu dari divais tersebut adalah FPGA. FPGA dengan performance dan jumlah *clock* yang tinggi menjadi kunci dalam pengembangan pengendalian motor. Selain itu, FPGA menunjang *parallel processing* yang berguna untuk mengendalikan beberapa axis sekaligus. Pembangkitan sinyal analog untuk mengendalikan motor dapat menggunakan sinyal modulasi PWM (Pulse Width Modulation) dan PDM (Pulse Density Modulation) yang dilewatikan pada low pass filter. Dalam skripsi ini digunakan PWM dan PDM sebagai sinyal kendalinya dan kemudian dibandingkan kinerjanya. Dari hasil percobaan didapatkan PDM adalah sinyal modulasi yang terbaik untuk mengendalikan motor untuk mesin CNC.

Kata kunci : FPGA, Mesin CNC, *servo amplifier*, PWM, PDM, sinyal kendali analog

ABSTRACT

Name : Candraditya Pradhana

Study program : Electrical Engineering

Title : Motor Control Analog Signal Generator with PC Based on FPGA (Mesa 5I20) and Servo Amplifier (Mesa 7I33)

Motor controlling on CNC machine is very important because this machine needs very high accuracy with quick process. So, devices needed to comply with that. One of those devices is FPGA. FPGA with high performance and clock will be the key of motor controlling. Besides, FPGA support parallel processing that used for controlling many axes simultaneously. Generation of analog signal for motor controlling uses PWM and PDM that passes into Low pass filter. In this thesis, PWM and PDM used for the controlling signal and compare the performance. The experiment result shows that PDM is better for motor controlling in CNC machine.

Key word : FPGA, CNC machine, servo amplifier, PWM, PDM, analog control signal

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Dunia industri sudah mengalami perkembangan yang sangat pesat. Dimulai dari revolusi industri hingga saat ini dengan penggunaan alat-alat yang telah memiliki tingkat efisiensi dan produksi yang sangat tinggi. Hal ini disebabkan telah banyak penemuan dan pengembangan di bidang industri yang terintegrasi langsung dalam proses produksi industri tersebut. Dengan perkembangan-perkembangan itu, produksi dari industri dapat meningkat dengan signifikan baik dari segi kualitas maupun kuantitas, dari segi kuantitas dapat diproduksi dalam jumlah yang besar dengan kecepatan yang tinggi sedangkan dari segi kualitas dapat terjamin keseragaman mutu untuk seluruh produk industri tersebut.

Saat ini permintaan barang dengan kuantitas yang besar serta dengan kualitas yang baik menjadi salah satu pertimbangan dalam mengembangkan teknologi yang digunakan oleh industri. Manusia memiliki keterbatasan di dalam hal ketelitian dan ketekunan, sehingga produksi barang secara manual sudah tidak memungkinkan lagi. Dengan ditemukannya teknologi yang digunakan untuk produksi, permintaan pasar atas kualitas dan kuantitas yang tinggi dapat tercapai. Namun demikian pengurangan jumlah tenaga kerja akibat digantikan oleh mesin ini dapat menambah jumlah pengangguran. Jumlah tenaga kerja yang berkurang dalam pergantian proses produksi ini dapat dialihkan menjadi teknisi, operator, serta pengawas dalam proses produksi itu sendiri. Dengan peralihan fungsi ini sumber daya manusia pun akan semakin baik dari segi kualitas. Sumber daya manusia yang semakin baik serta dengan alat yang baik akan menghasilkan produk yang semakin baik pula dalam segi kualitas maupun kuantitas.

Salah satu alat yang saat ini digunakan oleh dunia industri adalah mesin CNC. Harga mesin ini yang cukup mahal menjadi salah satu alasan bahwa industri kecil dan

menengah tidak dapat membeli alat ini. Dengan demikian, industri kecil dan menengah tidak dapat mengembangkan proses produksinya sehingga akan selalu tertinggal dengan industri besar yang sudah mapan, karena industri selalu menggunakan teknologi maju untuk proses produksinya, sehingga kualitas produknya lebih baik dan dengan harga yang dapat bersaing dengan ketat.

Dengan demikian, salah satu solusi yang dapat ditawarkan untuk dapat mengembangkan industri kecil dan menengah adalah memroduksi mesin CNC yang berharga murah sehingga dapat dijangkau oleh industri kecil dan menengah. Salah satu yang digunakan untuk mengurangi harga dari mesin CNC murah ini adalah dengan menggunakan sistem operasi *open source* yang gratis.

Pada skripsi ini akan dibahas mengenai sinyal pengendali PWM (Pulse Width Modulation) dan PDM (Pulse Density Modulation) yang akan merupakan bagian sangat penting untuk mengendalikan motor. Dari sinyal digital tersebut kemudian akan di filter sehingga dapat menjadi tegangan analog yang dapat menjadi tegangan referensi pada mesin CNC. Motor merupakan penggerak utama bagi setiap mesin CNC sehingga pengendalian motor yang baik dan mudah merupakan kunci keberhasilan rancangan perangkat CNC ini. Maka dalam hal ini, pengendali PWM dan PDM akan merupakan faktor utama untuk keberhasilan realisasi rancangan mesin CNC yang baik.

1.2 Tujuan

Adapun tujuan skripsi ini adalah :

1. Memahami mesin CNC
2. Memahami hubungan antarmuka mesin CNC berbasis PC
3. Menggunakan Mesa 5I20 dan Mesa 7I33 sebagai interface pada sistem mesin CNC
4. Memahami proses pembangkitan sinyal PWM dan PDM
5. Menganalisa sinyal digital PWM-PDM agar dapat menghasilkan tegangan analog referensi yang sesuai

1.3 Batasan Masalah

Batasan masalah yang dibahas dalam tulisan ini, yaitu :

1. Antarmuka pada mesin CNC berbasis PC
2. Pembangkitan sinyal kendali analog pada motor
3. Analisa terhadap sinyal pengendali PWM dan PDM

1.4 Metodologi Penulisan

Metode penulisan yang digunakan pada penyusunan skripsi ada beberapa tahap, yaitu :

1. Studi Literatur

studi literatur dari artikel-artikel atau media-media lain yang berkaitan dengan masalah yang dibahas

2. Perancangan dan pembuatan alat

Merancang dan menganalisa pengendali PWM dan PDM kemudian mencoba dengan melakukan implementasi

3. Pengujian sistem

Pengujian sistem dilakukan untuk mendapatkan data-data hasil percobaan.

4. Metode analisis

Menganalisis data-data hasil percobaan yang kemudian akan dibuat kesimpulan tentang karakteristik serta kelebihan dan kekurangannya.

1.5 Sistematika Penulisan

Sistematika penulisan buku skripsi ini adalah sebagai berikut :

1. Bab 1 : Pendahuluan

Bab ini berisi tentang Latar Belakang, Tujuan, Batasan Masalah, Metodologi Penulisan serta Sistematika Penulisan

2. Bab 2 : Landasan Teori

Bab ini berisi tinjauan literatur tentang konsep alat-alat yang diterapkan pada mesin CNC

3. Bab 3 : Implementasi Antarmuka pada Mesin CNC

Bab ini berisi tentang spesifikasi dan konfigurasi mesa 5i20 dan mesa 7i33, pembangkitan PWM-PDM pada FPGA, serta DAC (*digital to analog converter)* dengan menggunakan *low pass filter*

4. Bab 4 : Analisa

Bab ini berisi tentang analisa data dari variasi frekuensi serta kecepatan pada mesin CNC yang dibangun

5. Bab 5 : Penutup

Bab ini berisi kesimpulan dari penulisan buku skripsi ini

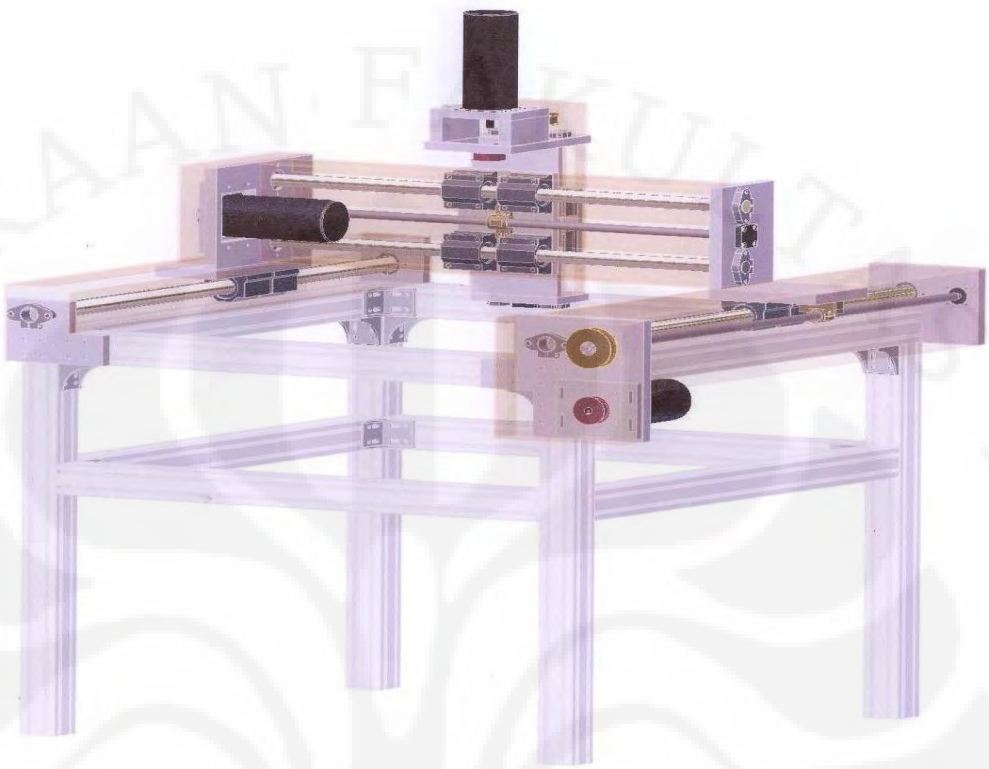
BAB 2

LANDASAN TEORI

2.1 Mesin CNC

Mesin CNC (Computer Numerical Control) adalah mesin yang dilengkapi dengan sistem mekanik dan kontrol berbasis computer yang mampu membaca instruksi kode N, G, F, T, dan lain-lain, dimana kode-kode tersebut akan menginstruksikan ke mesin CNC agar bekerja sesuai dengan program benda kerja yang akan dibuat. Mesin CNC pertama kali dibuat pada tahun 1952 yang dikembangkan oleh John Pearseon dari MIT (Massachusetts Institute of Technology) yang digunakan angkatan udara Amerika Serikat. Mesin CNC ini digunakan untuk membuat benda khusus yang rumit dikarenakan pada saat itu angkatan udara Amerika Serikat membutuhkan benda yang rumit dan membutuhkan ketelitian yang baik.

Otomatisasi CNC dapat mengantikan peran operator pada penggunaan mesin perkakas lainnya, seperti mengatur gerakan pahat pada posisi memotong, dan lain-lain. Mesin CNC dilengkapi dengan berbagai alat potong sehingga benda yang akan dibuat menjadi presisi dan dapat melakukan interpolasi yang diarahkan secara numeric berdasarkan angka. Parameter pada sistem operasi mesin CNC ini dapat dirubah dengan perangkat lunak pada komputer yang digunakan sehingga sesuai dengan hardware yang digunakan. Tingkat ketelitian mesin CNC sangat tinggi hingga mencapai seperseribu millimeter, karena penggunaan *ballscrew* pada transportiernya. *Ballscrew* bekerja seperti *lager* yang tidak memiliki kelonggaran/*spelling* namun dapat bergerak dengan lancar. Contoh gambar mesin CNC dapat dilihat pada gambar 2.1.



Gambar 2.1 mesin CNC

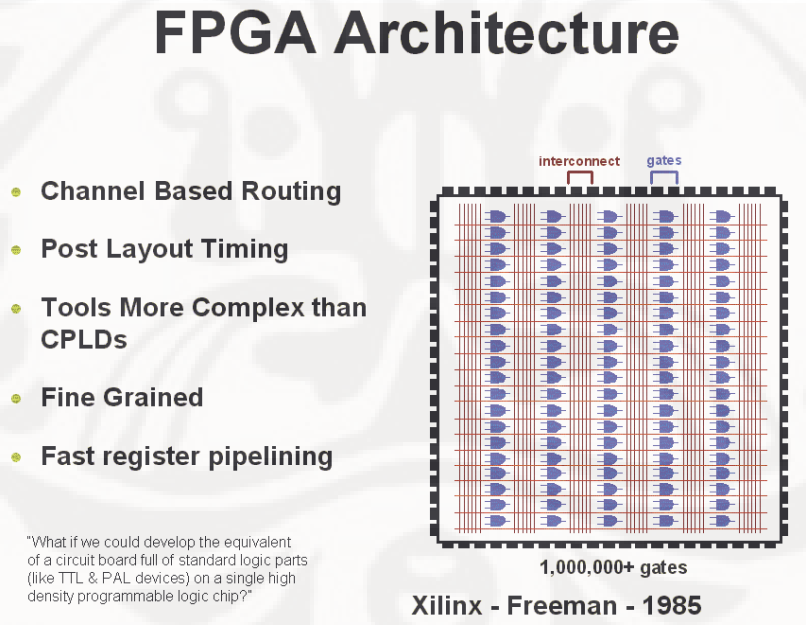
Gambar 2.2 diagram blok rancangan mesin CNC

Dalam mesin CNC tersebut terdapat tiga bagian penting, yaitu *main controller, interface,* dan *driver* dengan *actuator*. Dalam *main controller* digunakan program *open source* EMC2 sedangkan untuk *interface* digunakan FPGA Mesa 5I20 serta *servo amplifier* Mesa 7I33. Untuk *driver* dan *actuator* digunakan motor yaskawa dengan servopack. Diagram blok rancangan mesin CNC tersebut dapat digambarkan seperti pada gambar 2.2.

2.2 FPGA (Field-Programable Gate Array)

FPGA adalah logic device yang tersiri dari larik 2 dimensi dari logic cell dan programable switches. FPGA diprogram dengan menggunakan diagram rangkaian logika atau menggunakan HDL (*Hardware Description Languange*), untuk menetapkan bagaimana chip-nya bekerja.

FPGA memiliki komponen gerbang terprogram (*programmable logic*) dan sambungan terprogram. Komponen gerbang terprogram yang dimiliki meliputi jenis gerbang logika biasa (AND, OR, XOR, NOT) maupun jenis fungsi matematis dan kombinatorik yang lebih kompleks (*decoder*, *adder*, *subtractor*, *multiplier*, dll). Blok-blok komponen di dalam FPGA bisa juga mengandung elemen memori (*register*) mulai dari flip-flop sampai pada RAM (*Random Access Memory*). Gambar arsitektur FPGA dapat dilihat pada gambar 2.3:



Gambar 2.3 Arksitektur FPGA [1]

Pengertian Terprogram (*programmable*) dalam FPGA adalah mirip dengan interkoneksi saklar dalam breadboard yang bisa diubah oleh pembuat desain. Dalam FPGA, interkoneksi ini bisa diprogram kembali oleh pengguna maupun pendesain di

dalam lab atau lapangan (*field*). Oleh karena itu jajaran gerbang logika (*Gate Array*) ini disebut *field-programmable*. Jenis gerbang logika yang bisa diprogram meliputi semua gerbang dasar untuk memenuhi kebutuhan yang manapun.

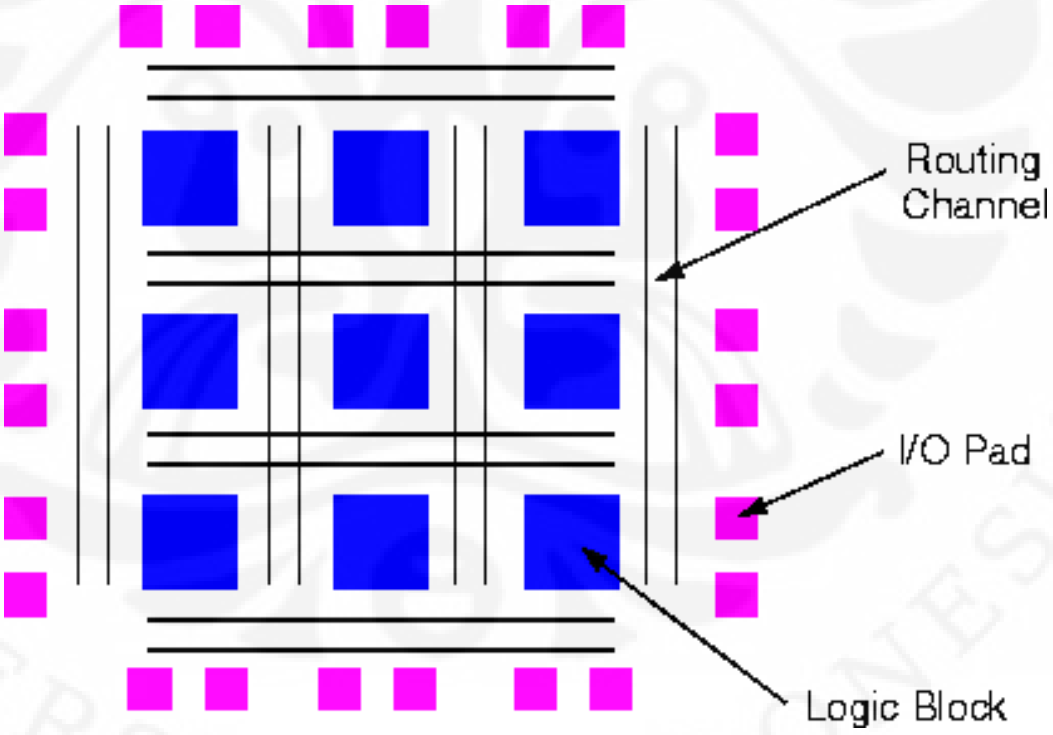
Fisik dari FPGA (*field programmable gate array*) berbentuk chip IC. Setiap chip PGA terdiri dari puluhan hingga puluhan ribu sel logika. Masing-masing sel logika mempunyai keluaran yang berjumlah satu atau dua, tergantung dari fungsinya. Secara umum, Arsitektur FPGA dapat dipandang sebagai kumpulan blok, dimana tiga elemen penyusunnya adalah CLB (*combinational logic block*), IOB (*input/output block*), dan interkoneksi. Untuk memilih FPGA membutuhkan analisis mengenai *memory*, *performance*, dan I/O *interface*. Perkembangan FPGA pada saat ini berlangsung sangan cepat. Xilinx merupakan salah satu perusahaan yang memproduksi FPGA. Beberapa jenis FPGA yang telah diproduksi adalah VIRTEX, SPARTAN, XC4000, dan XC5000.

FPGAs berbeda dari *general-purpose mikroprosesor* (misalnya Intel) dalam hal fleksibilitas *logic*-nya. *Mikroprosesor* mempunyai *hardware* yang tetap. *Assembly programmer* memprogram suatu komputasi dengan keterbatasan pada tetapnya banyak register, siklus *fetch-decodeexecute*, serta fungsi-fungsi ALU(*arithmetic and logic unit*) dan pada banyaknya bit suatu register. FPGAs berbeda dari mikrokontroler (misalnya ATMEL), karena mikrokontroler pada prinsipnya adalah mikroprosesor yang diprogram dengan bahasa assembly dan dirancang sebagai pengendali bukan untuk komputasi. Mikroprosesor dan mikrokontroler mengimplementasikan suatu komputasi pada *hardware* yang tetap. *Hardware* pada FPGAs diserahkan sepenuhnya pada design *engineer* untuk memprogramnya. Sebelum diprogram, FPGAs hanyalah tersusun atas blok-blok yang belum dikonfigurasikan dan interkoneksi yang belum disusun dan difungsikan. Oleh karena itu, istilah yang lebih tepat adalah merekonfigurasikan FPGAs, bukan memprogramnya. Chip FPGAs yang sama dikonfigurasikan dengan data yang berbeda akan mengimplementasikan hardware yang berbeda.

Alur kerja yang umum dalam memprogram FPGA:

1. Menggunakan komputer untuk mendeskripsikan fungsi logika yang diinginkan. Bisa dengan menggambar skematiknya atau menuliskan programnya
2. Menyusun (compile) fungsi logika, menggunakan software yang disediakan oleh vendor FPGA, lalu membuat file biner yang dapat diunduh ke dalam FPGA
3. Menghubungkan kabel dari komputer ke FPGA, dan mengunduh file biner ke FPGA

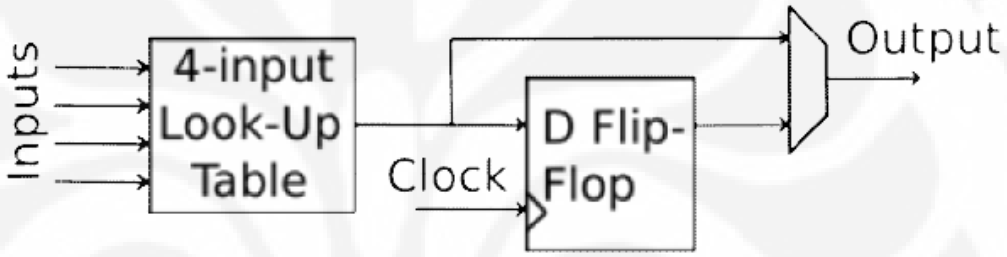
Arsitektur FPGA paling umum adalah terdiri atas susunan dari CLB (*configurable logic blocks*), *pad* I/O, dan *routing channel*. Struktur tersebut dapat dilihat pada gambar 2.4. Blok logika FPGA (model awal / klasik) terdiri dari 4-input lookup table (LUT), dan flip-flop. Dewasa ini, pabrikan FPGA telah mulai mengganti dengan 6- input LUT dalam komponen performa tinggi mereka.



Gambar 2.4 Struktur Umum FPGA [1]

a. Logic Block / Cells

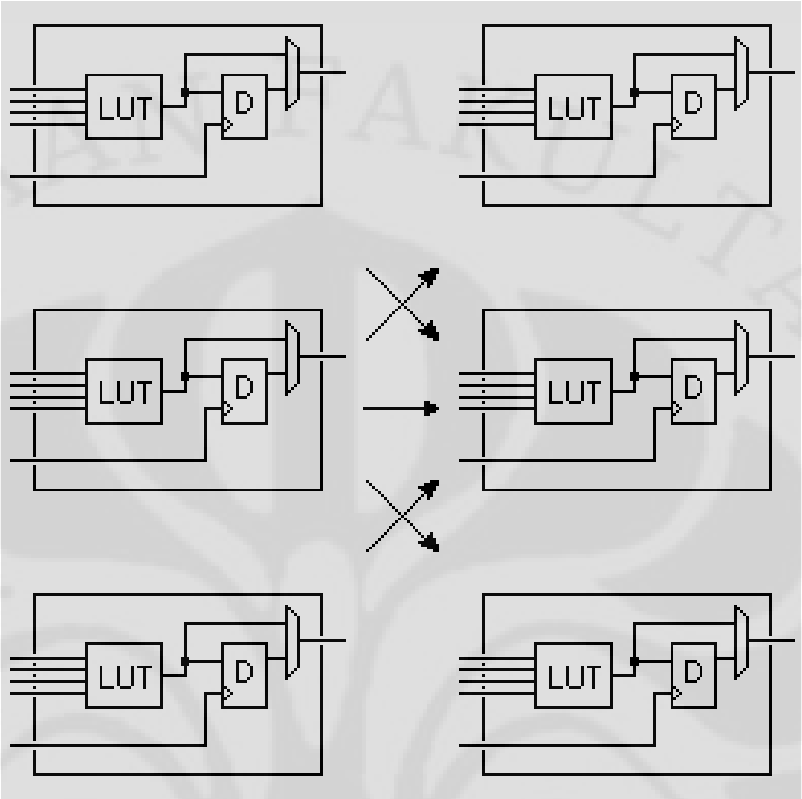
Blok logika memiliki 4 *input* untuk LUT dan 1 *input clock*. Signal *clock* biasanya dihubungkan via *routing* tersendiri pada FPGA umumnya. Berikut adalah gambar blok logika secara umum dimana 4 buah input yang dikonversikan dengan menggunakan look up table yang kemudian akan menjadi masukkan pada D Flip-flop dan menjadi *output*. Logic block tersebut terdapat pada gambar 2.5.



Gambar 2.5 *logic block* umum [1]

b. Routing Channel / Interconnect

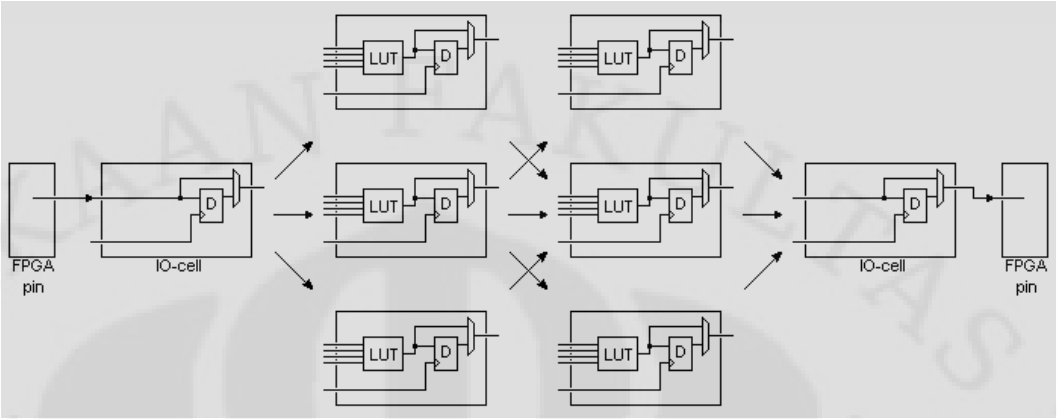
Tiap pin output blok logika bisa dihubungkan ke *wiring segment* manapun pada *channel* yang di dekatnya. Dengan cara yang sama, sebuah *pad* I/O bisa dihubungkan ke *wiring segment* manapun juga. Tiap-tiap logic-cell bisa dihubungkan melalui interconnect (kabel / mux). Interkoneksi tiap logic cell dapat dibuat dengan bebas. Tiap cell hanya berfungsi sedikit, tapi bila digabungkan, fungsi logika yang kompleks dapat dibuat. Koneksi tersebut dapat dilihat pada gambar 2.6.



Gambar 2.6 Interconnect [1]

c. IO-Cells

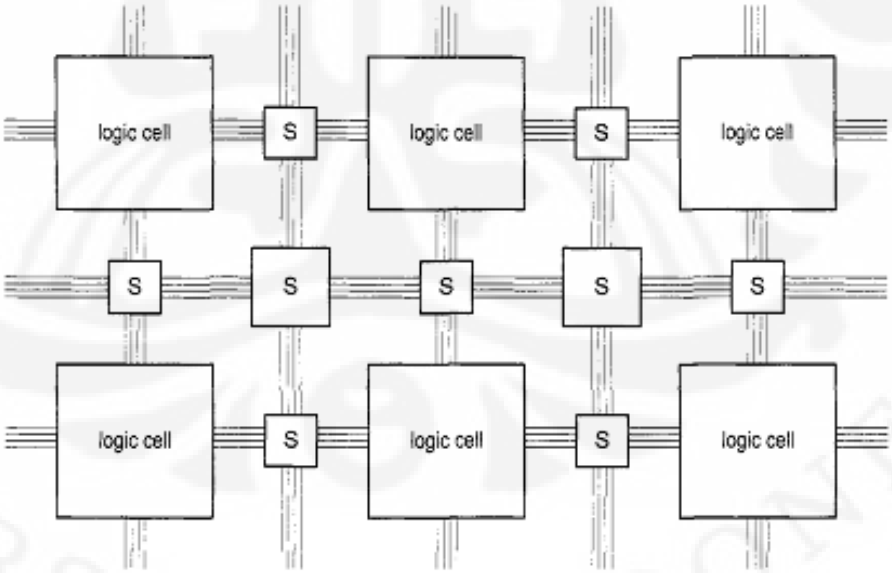
Kabel interconnect juga bisa dihubungkan dengan I/O cell yang terhubung dengan pin-pin tertentu di FPGA. Dari gambar 2.7 didapatkan bahwa pin terkoneksi dengan I/O pad yang kemudian akan terjadi koneksi-koneksi dengan logic cell sehingga memenuhi diagram alir dari yang telah dirancang kemudian nilainya akan dikeluarkan kembali pada I/O pad lainnya. Berikut ini adalah gambar interconnect I/O:



Gambar 2.7 Interconnect I/O [1]

d. Switch Box

Dimana pun channel vertical dan horizontal berpotongan, maka akan terdapat switch box. Terdapat 3 programmable switch yang dapat menghubungkan suatu kabel dengan 3 kabel lain di dekatnya. Pada gambar 2.8 diterangkan tentang FPGA dengan switch box yang menjadi koneksi pada tiap logic-cell. Gambar tersebut menjelaskan bahwa konfigurasi switch menjadi sangat penting dalam memprogram FPGA.



Gambar 2.8 Struktur konseptual dari FPGA (*ket : S = Switch Box*) [1]

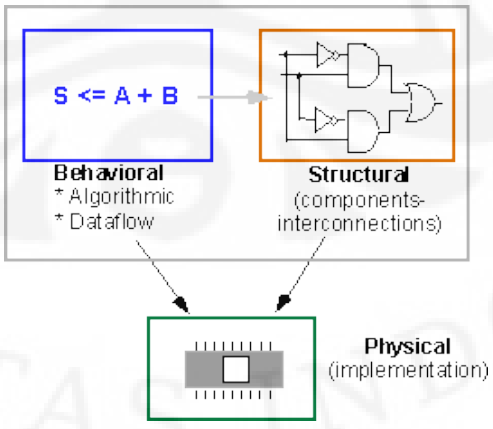
Keluarga FPGA modern saat telah ditingkatkan kapabilitasnya menuju kemampuan yang lebih tinggi. Contoh-contohnya adalah multipliers, generic DSP blocks, embedded processors, high speed IO logic dan embedded memori.

2.3 VHD Languange

VHDL (VHSIC Hardware Description Language) adalah sebuah bahasa pemrograman VHSIC (Very High Speed Integrated Circuit) yang dikembangkan oleh IEEE (Institute of Electrical and Electronic Engineering). Versi awal dari VHDL adalah versi 1987 (IEEE 1076-1987). VHDL termasuk dalam bahasa pemodelan yang digunakan untuk merancang atau memodelkan rangkaian digital. Keuntungan pada penggunaan VHDL, yaitu:

- VHDL mampu melakukan desain hardware hingga sistem yang lebih kompleks
- Mudah dalam mencari dan mendeteksi kesalahan dengan lebih mudah dalam simulasi
- Bahasa pemrograman yang mudah dimengerti dan dipelajari

Sebuah sistem digital dapat diwakili pada tingkatan abstraksi yang berbeda. Hal ini membuat deskripsi dan rancangan sistem yang kompleks dapat diatur. Gambar 2.9 menjelaskan tentang tingkatan abstraksi tersebut.



Gambar 2.9 tingkatan abstraksi: Behavioral, Structural and Physical [2]

Terdapat 3 cara untuk mendeskripsikan tingkatan abstraksi VHDL, yaitu:

- Behavioral

Tingkat tertinggi dari tingkatan abstraksi adalah tingkatan perilaku yang menggambarkan sebuah sistem dalam hal apa yang dilakukan (atau bagaimana berperilaku) dan bukan dari segi komponen dan interkoneksi di antara mereka. Sebuah deskripsi perilaku menentukan hubungan antara input dan output sinyal. Hal ini dapat berbentuk ekspresi Boolean atau deskripsi yang lebih abstrak seperti Daftar Transfer atau tingkat algoritmik. Sebagai contoh, mari kita perhatikan sebuah rangkaian sederhana yang memperingatkan penumpang mobil ketika pintu terbuka atau sabuk pengaman tidak digunakan setiap kali kunci mobil dimasukkan ke dalam kunci kontak mengunci Pada tingkat perilaku ini dapat dinyatakan sebagai:

Warning = Ignition_on AND (Door_open OR Seatbelt_off)

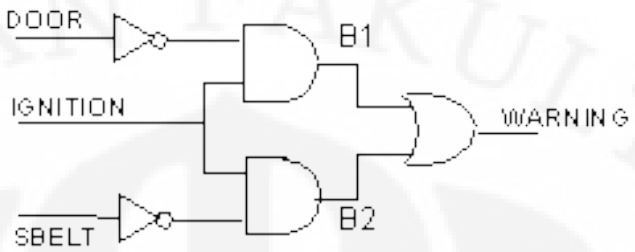
Pendekatan *behavioral* yang memodelkan komponen perangkat keras berbeda dari dua metode yang lainnya, yang mana tidak mencerminkan bagaimana desain diimplementasikan. Seperti pada dasarnya black box. Pendekatan *behavioral* secara akurat memodelkan apa yang terjadi pada masukan dan keluaran dari kotak hitam, tapi apa yang di dalam kotak (cara kerjanya) tidak menjadi soal.

- Structural

Pada desain structural dimana setiap bagian desain dibagi kedalam beberapa blok. Blok-blok ini kemudian disambungkan sehingga akan menjadi satu desain utuh. Setiap blok disebut entity. Entity mendeskripsikan interface pada blok dan bagaimana blok dapat beroperasi. Deskripsi interface merupakan spesifikasi input dan output pada blok. Deskripsi dari operasi adalah seperti hubungan skematik blok-blok.

Tingkat structural menggambarkan suatu sistem sebagai kumpulan dari gerbang dan komponen yang saling berhubungan untuk melakukan fungsi yang dikehendaki. Sebuah deskripsi struktural dapat dibandingkan dengan skema yang saling berhubungan gerbang logika. Ini adalah sebuah representasi yang biasanya lebih

dekat dengan realisasi fisik suatu sistem. Untuk contoh di atas, representasi struktural ditunjukkan pada Gambar 2.10.



Gambar 2.10 Struktur representasi dari rangkaian *buzzer* [2]

- Physical

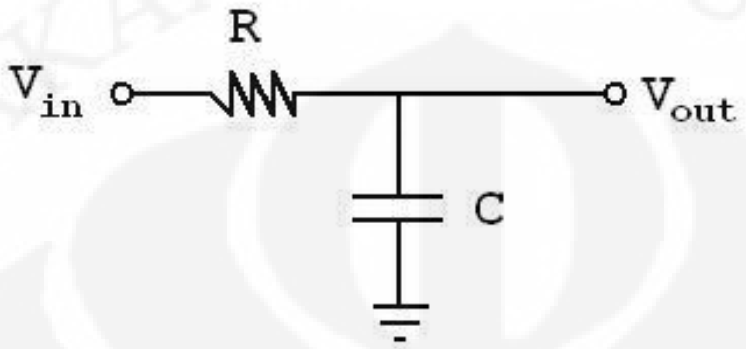
VHDL memungkinkan seseorang untuk menggambarkan sistem digital di struktural atau tingkat perilaku. Tingkat perilaku dapat dibagi lagi menjadi dua jenis gaya: Data flow dan algoritmik. Dalam pendekatan data flow, rangkaian dideskripsikan dengan mengindikasikan bagaimana input dan output dari komponen *built-in primitive* (misal gerbang *and*) yang dihubungkan bekerja. Dengan kata lain, dideskripsikan bagaimana signal (data) ditransmisikan di dalam rangkaian. Selain itu, dataflow representasi yang menggambarkan bagaimana data bergerak melalui sistem. Ini biasanya dilakukan dalam hal aliran data antara register (Register transfer level). Model aliran data penggunaan bersamaan membuat pernyataan yang dijalankan secara paralel data segera setelah tiba di masukan. Di sisi lain, pernyataan sekuensial dijalankan dalam urutan yang mereka ditetapkan. VHDL memungkinkan baik sinyal bersamaan dan berurutan yang akan menentukan cara di mana mereka dieksekusi.

2.4 Filter Pasif

Rangkaian filter adalah rangkaian yang digunakan untuk memfilter sinyal sesuai dengan yang kita inginkan. Selain itu, rangkaian filter dapat digunakan pula untuk rangkaian differensiator maupun integrator. Ada empat jenis filter yang mempunyai tanggapan frekuensi ideal, yaitu:

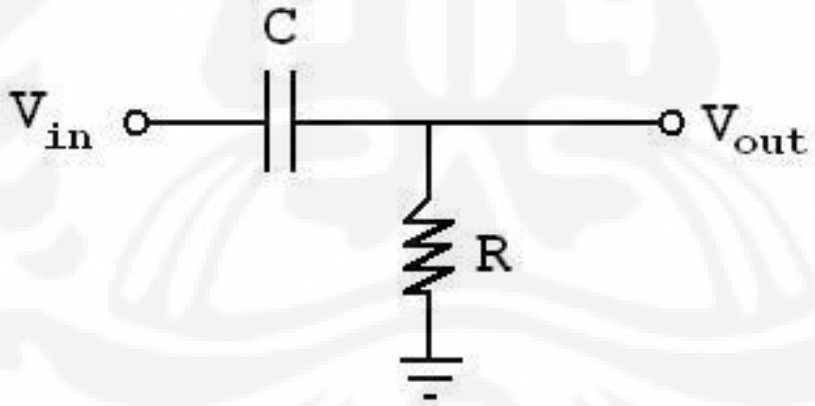
a. Low pass filter, keluaran filter yang dinyatakan oleh H(j2Πf) muncul untuk

frekuensi-frekuensi rendah. Konfigurasi low pass filter dapat digambarkan seperti pada gambar 2.11.



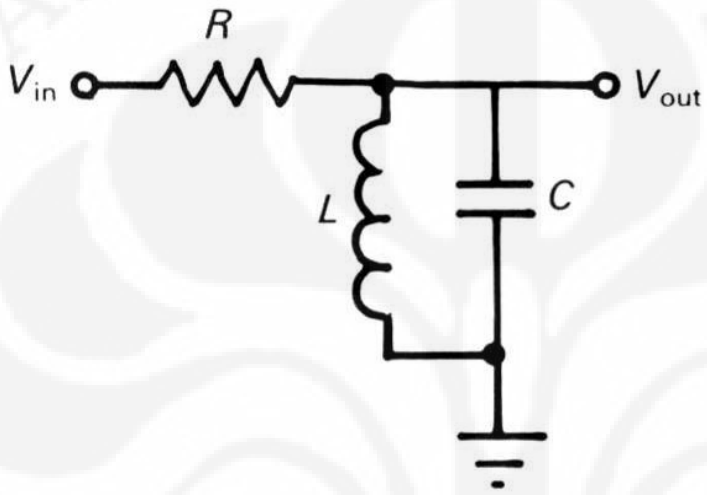
Gambar 2.11 konfigurasi low pass filter [3]

b. High pass filter, keluaran filter yang dinyatakan oleh H(j2Πf) muncul untuk frekuensi-frekuensi atas. Konfigurasi high pass filter dapat digambarkan seperti pada gambar 2.12.



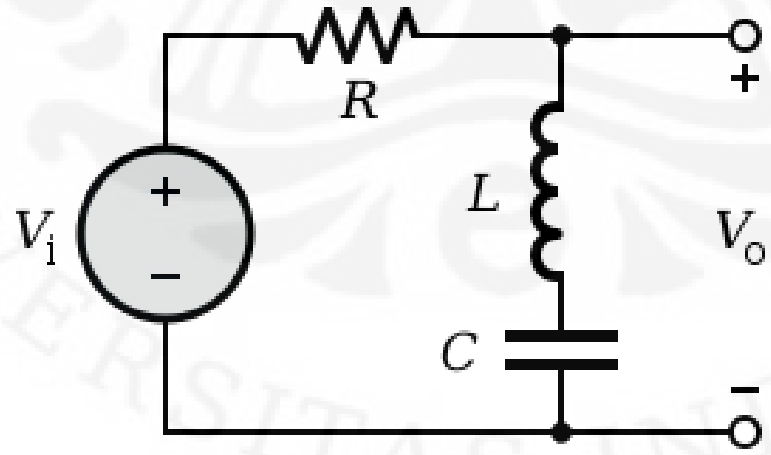
Gambar 2.12. konfigurasi high pass filter[3]

c. Band pass filter, keluaran filter yang dinyatakan oleh H(j2Πf) muncul untuk frekuensi-frekuensi batas bawah f1 dan batas atas f2. Konfigurasi band pass filter dapat digambarkan seperti pada gambar 2.13.



Gambar 2.13. konfigurasi band pass filter [4]

d. Band rejection filter, keluaran filter yang dinyatakan oleh H(j2Πf) tidak muncul pada batas antara f1 dan f2. Konfigurasi band rejection filter dapat digambarkan seperti pada gambar 2.14.



Gambar 2.14. konfigurasi band rejection filter [5]

Frekuensi cut-off adalah frekuensi batas yang ingin kita filter. Seperti kita ketahui, kapasitor memiliki resistansi (Xc) dan frekuensi cut-off adalah keadaan dimana Xc=R. Dengan demikian dapat diturunkan rumus sebagai berikut:

$$R = \frac{1}{\omega.\mathrm{C}} \tag{2.1}$$

$$R = \frac{1}{2.\pi.fc.C} \tag{2.2}$$

$$fc = \frac{1}{2.\pi.R.C} \tag{2.3}$$

2.5 Sinyal Modulasi

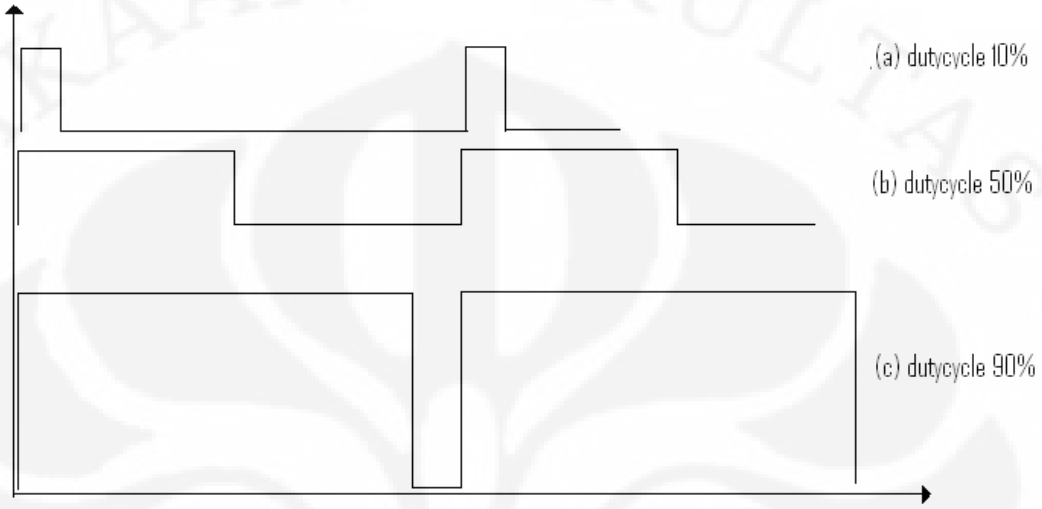
Modulasi adalah proses untuk menggabungkan beberapa sifat informasi dari sinyal ke dalam sinyal carrier. Dalam pengendalian motor, modulasi menjadi salah satu hal penting yang harus diperhatikan. Dengan menggunakan sinyal modulasi yang sesuai dengan karakteristik motor maka output dari motor akan sesuai dengan rancangan. Berikut ini adalah contoh-contoh dari modulasi yang sering digunakan untuk mengendalikan motor:

- Pulse-width modulation (PWM)
- Pulse-density modulation (PDM)

2.5.1.Pulse Width Modulation (PWM)

PWM merupakan salah satu jenis modulasi dengan cara mengatur lebar pulsa high dan low. PWM bekerja dengan membuat gelombang persegi (square-wave) dengan perbandingan pulsa yang ditentukan oleh sinyal informasi. Tujuan penggunaan PWM adalah untuk memberikan tegangan rata-rata yang berbeda. PWM diusahakan berada pada frekuensi yang tetap agar data yang diambil tidak salah apabila receiver mengambil dengan sampling time yang sama. Contoh gambar PWM

dengan perbedaaan tiap sinyal informasi (dengan *duty cycle* yang berbeda) pada PWM digambarkan pada gambar 2.15.



Gambar 2.15. Sinyal PWM dengan duty cycle yang beragam

Dalam penentuan besar PWM digunakan parameter duty cycle. Duty Cycle merupakan besarnya perbandingan lama nilai high dengan periode satu gelombang. Duty cycle biasanya ditentukan dalam bentuk persen. Pada gambar diatas bisa kita lihat bahwa untuk gambar (a) dengan duty cycle 10% memiliki kira-kira perbandingan nilai high adalah 10% dari periodanya, begitu pula dengan gambar (b) yang memiliki duty cycle 50% dan gambar (c) dengan duty cycle 90%.

2.5.2.Pulse-density modulation (PDM)

PDM adalah salah satu motode modulasi dengan cara mengatur jarak antara lebar pulsa satu dengan lebar pulsa yang lainnya. Fungsi dari PDM sendiri digunakan sebagai modulasi tegangan analog menjadi sinyal digital. Konsep dari DAC dengan menggunakan PDM sendiri melalui proses *delta sigma modulation*. berikut ini adalah salah satu algoritma yang digunakan untuk mendapatkan nilai modulasi PDM.

e[-1] = 0
for n **from** 0 **to** sampling

if x[n] > e[n-1] **then** y[n]=1

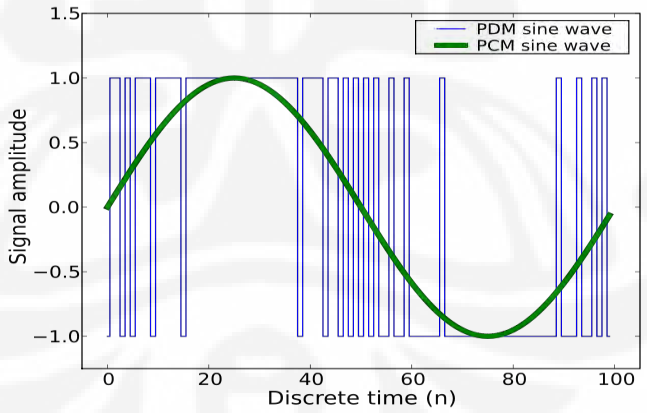
else y[n]= -1

e[n] = y[n]-x[n]+e[n-1]

end for

Pada algoritma diatas, x[n] adalah posisi tegangan analog yang ingin dikonversikan. Selanjutnya y[n] adalah nilai digital yang telah dikonversikan dan e[n] adalah nilai error. Metode yang digunakan adalah dengan memperbaharui nilai pada sampling berikutnya. Ketika nilai x[n] lebih besar dari e[n-1] maka nilai digital yang dikeluarkan adalah 1 dan sebaliknya apabila lebih kecil atau sama dengan akan bernilai -1. Selanjutnya nilai error akan diperbarui berdasarkan nilai saat itu. Selanjutnya algoritma ini akan diulang terus menerus hingga waktu sampling selesai.

Pada gambar 2.16 dijelaskan bentuk pengkodean sinyal analog menjadi digital dengan menggunakan prinsip PDM. Untuk nilai yang mendekati nilai 1 maka sinyal digitalnya akan semakin rapat sedangkan untuk yang semakin negatif maka sinyal digitalnya akan semakin renggang.



Gambar 2.16. pengkodean *analog to digital converter* dengan PDM [6]

BAB 3

IMPLEMENTASI ANTARMUKA PADA MESIN CNC

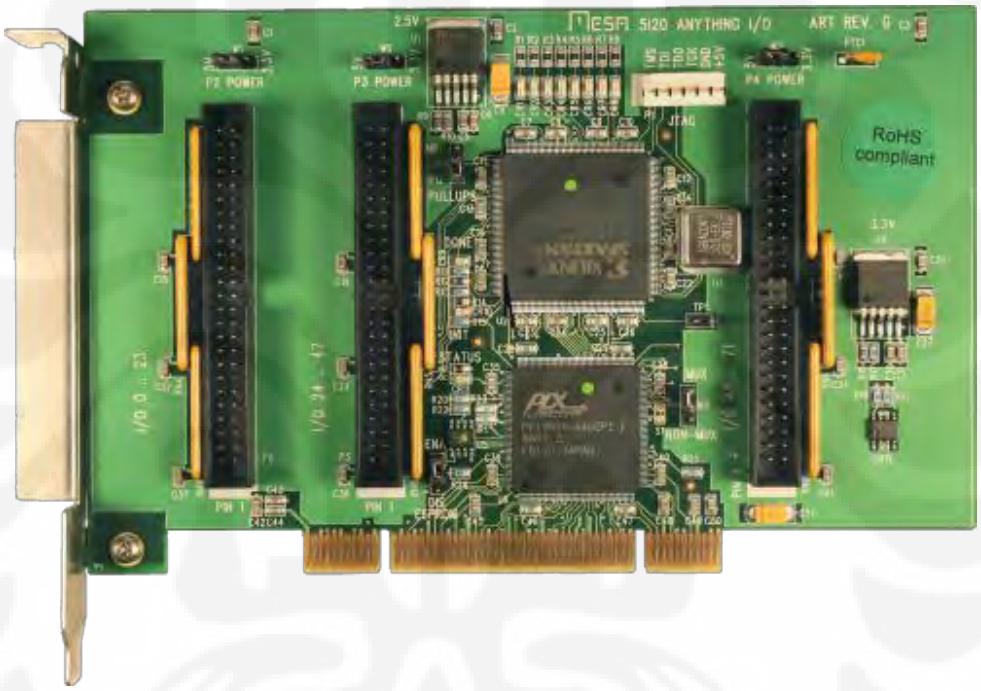
Pada hubungan antarmuka mesin CNC berbasis PC ini terdapat dua buah modul rangkaian yang dipakai, yaitu FPGA mesa 5i20 dan servo amplifier mesa 7i33. Penggunaan FPGA yang disambungkan dengan PC ini biasa disebut hubungan *master-slave*. Hubungan *master-slave* adalah model komunikasi dimana satu prosesor mengontrol langsung prosesor lain dengan demikian tiap prosesor dapat mengerjakan tugasnya secara spesifik sehingga dapat mengurangi waktu proses selain itu dapat meningkatkan peforma. Selain itu, penggunaan FPGA pada perancangan ini diperuntukan agar tidak membebani computer dalam menjalankan program EMC2.

Mesin CNC memiliki keakuratan yang sangat tinggi sehingga dibutuhkan sebuah divais yang memiliki kecepatan dan keakuratan yang sangat tinggi. Pemilihan FPGA sebagai basis untuk menjadi antarmuka dari mesin CNC ini adalah karena kemampuan FPGA yang sangat baik untuk mengontrol dalam jumlah banyak dan dalam waktu yang singkat. Untuk saat ini, divais antarmuka yang memiliki *clock* cukup tinggi adalah FPGA. FPGA dapat dimaksimalkan untuk mengendalikan satu kanal dengan kecepatan akses yang sangat cepat atau banyak kanal dengan kecepatan akses yang tidak terlalu cepat. Mesa 5I20 yang dipakai dalam mesin CNC ini memiliki *clock external* hingga 50MHz. Dengan demikian, FPGA menjadi salah satu pilihan sebagai basis dalam antarmuka mesin CNC ini.

Selain karena *clock* yang sangat tinggi, mesin CNC juga membutuhkan kerja yang parallel antar satu axis dengan yang lainnya. FPGA mengakomodir kebutuhan tersebut. FPGA dapat mengerjakan beberapa pekerjaan secara parallel. Dengan demikian, mesin CNC dapat menjalankan tiap axis secara bersama-sama. Kelebihan lain dari FPGA adalah FPGA dapat mengonfigurasi hardware dengan menggunakan software. Hal ini sangat penting untuk mendapatkan output yang kita inginkan dan tidak terjebak hanya pada output yang telah dikonfigurasikan oleh pabrik.

3.1 Mesa 5i20

Mesa 5i20 adalah *general purpose programmable I/O card* untuk PCI *bus*. Mesa 5i20 menggunakan sebuah 200K gate Xilinx FPGA untuk semua logika. FPGA dapat diunduh dari PCI *bus* sehingga memungkinkan untuk membuat fungsi I/O yang spesifik, bahkan membuat *micro-controller* di dalam FPGA. Pada gambar 3.1 adalah gambar Mesa 5i20.



Gambar 3.1 Mesa 5i20 [7]

Beberapa fungsi telah dibuat oleh Mesa dalam bentuk *firmware-firmware* yang dapat digunakan langsung pada Mesa 5i20. *Firmware-firmware* tersebut termasuk 72 bit I/O paralel dengan tiga 24 bit port, 12 *channel host* berbasis pengendali motor servo, 8 *channel micro-controller* berbasis pengendali motor servo, dan 8 channel 32 bit *timer counter card* yang mampu berfungsi dengan frekuensi 100 MHz.

Semua I/O bit bertegangan 5V dengan arus 24mA. *Pullup resistor* juga disediakan sehingga memungkinkan kita mengoneksikan langsung dengan *opto-*

isolator, kontak, dan lain-lain. 5i20 menggunakan tiga 50 pin konektor sebagai I/O modul dan *ground.*

Penggunaan mesa 5I20 dalam mesin CNC ini adalah sebagai antarmuka PC dengan *actuator.* Namun demikian selain sebagai antarmuka, mesa 5I20 berfungsi sebagai penunjang kerja dari PC agar PC tidak terbebani untuk menjalankan program-programnya. Hubungan ini biasa disebut hubungan *master-slave.* Dengan memberikan beberapa modul perintah yang dapat diprogram dalam FPGA maka kerja dari komputer menjadi lebih ringan. Salah satu tugas dari FPGA mesa 5I20 adalah membangkitkan sinyal pengendali PWM atau PDM yang dengan nilai referensi yang bersumber dari komputer.

3.2.1 Spesifikasi dan Konfigurasi

Spesifikasi menjadi hal yang sangat penting dalam mengonfigurasikan mesa 5I20. Pada tabel 3.1 akan dijelaskan tentang spesifikasi-spesifikasi dari mesa 5I20.

Tabel 3.1. tabel spesifikasi mesa 5I20

Power	Min	Maks	Catatan :
Suplai Power	4,5 V	5,5 V	
Konsumsi Power	---	1000mA	Tergantung dari konfigurasi FPGA
Maksimum arus pada tiap i/o pada mode 5 V	---	500 mA	Total dari 3 port
Maksimum arus pada tiap i/o pada mode 3,3 V	---	500 mA	Total dari 3 port

Berdasarkan spesifikasi diatas, mesa 5I20 dapat dikonfigurasikan seperti pada gambar 3.2

Gambar 3.2. konfigurasi mesa 5I20

i. PCI card

Hubungan antara FPGA yang digunakan dalam Mesa 5I20 belum dapat berkomunikasi dengan komputer. Diperlukan suatu divais yang digunakan untuk menjadi antarmuka antara PCI *card* komputer dengan FPGA sehingga digunakan sebuah PLX PCI9030 yang merupakan modul komunikasi PCI. Dengan penggunaan PLX PCI9030 maka terbentuk jalur komunikasi via PCI card untuk mengakses memori dan *port* secara langsung.

ii. Mode Multiplexed/Non-Multiplexed

Antar muka *local bus* dari *chip* PCI ke FPGA dapat dioperasikan dalam dua mode. *Multiplexed* dan *non-multiplexed*. Pada mode *multiplexed*, *local bus addresses* dihasilkan pada baris data pada awal siklus *local bus*. Pada mode *non-multiplexed*, *addresses* dihasilkan pada pin *local bus address* yang terpisah. keuntungan mode *multiplexed* adalah semua 32 *address* bit tersedia pada *chip* FPGA sedangkan kerugiannya adalah dimana konfigurasi FPGA harus dikancingkan pada *addresses*. *Default* mode-nya adalah mode *multiplexed* dan konfigurasi FPGA yang digunakan dalam proyek ini adalah dengan menggunakan mode *multiplexed*.

iii. Pull Up Enable

Xilinx FPGA pada mesa 5i20 memiliki pilihan untuk *pull-ups* pada semua pin I/O pada powerup dan reset. Kondisi default adalah mengaktifkan pull-ups sehingga pin antarmuka *chip* FPGA/PCI tidak melayang(*floating*) ketika FPGA pada kondisi yang tidak dikonfigurasikan. Untuk mengaktifkan *built-in pull-ups*, (kondisi *default*) *jumper* W4 harus berada di posisi bawah dan untuk menonaktifkan pull-ups internal, W4 harus berada di posisi atas.

iv. EEPROM Enable

Chip PLX9030 PCI-*local bus* dikonfigurasikan melalui serial EEPROM. Jika EEPROM terjadi kesalahan program dan tidak sempurna maka tidak mungkin dapat menulis kembali EEPROM dari PCI *bus*. Untuk menghindari hal ini, EEPROM dapat

dinonaktifkan sementara. *Control* W6 adalah fungsi mengaktifkan EEPROM. Ketika W6 pada posisi diatas (default) EEPROM diaktifkan dan ketika W6 pada posisi kebawah EEPROM dinonaktifkan.

v. Connector Power

Power connection pada *connector* I/O dapat menyuplai 3.3V atau 5V. suplai *power* dibatasi hingga mencapai total 400mA. W1 memilih suplai *power* untuk P2, W2 memilih suplai *power* untuk P3, dan W3 memilih suplai *power* untuk P4. Ketika W1,W2,atau W3 ada pada posisi kiri, *power* 5V akan disuplai pada *connector* dan dihubungkan dengan *resistor pullup* sedangkan apabila W1, W2, atau W3 ada pada posisi kanan, maka suplai yang digunakan 3,3V.

vi. I/O Connectors

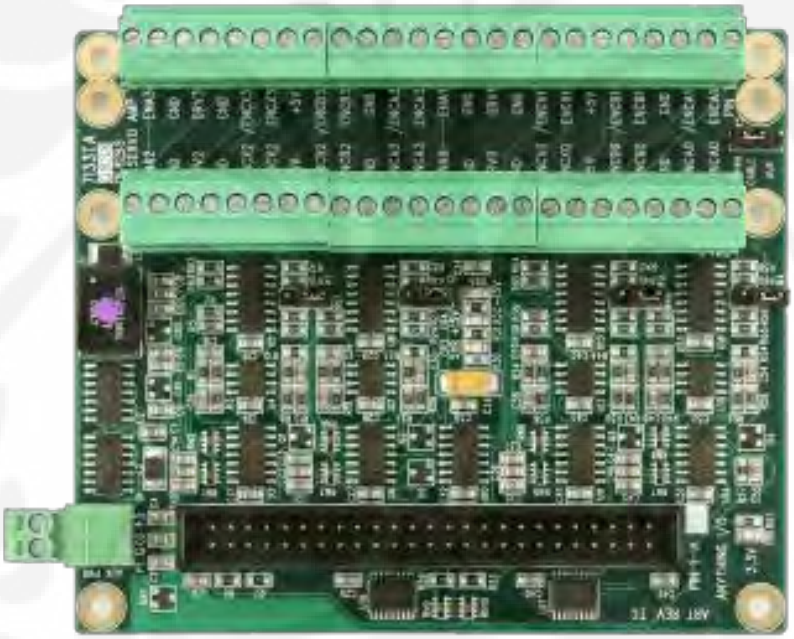
P2, P3, dan P4 adalah I/O *connector* pada Mesa 5i20. Pada tiap kotak tersebut terdiri dari 50 pin yang menggunakan standar *female* IDC *connectors*. Tabel 3.2 merupakan tabel dari konfigurasi tiap pin pada mesin 5I20.

Tabel 3.2.konfigurasi I/O mesa 5I20

PIN	FUNGSI	ARAH SINYAL	PIN	FUNGSI	ARAH SINYAL
1	QB1	Dari 7i33	27	QA3	Dari 7i33
3	QA1	Dari 7i33	29	QB2	Dari 7i33
5	QB0	Dari 7i33	31	QA2	Dari 7i33
7	QA0	Dari 7i33	33	IDX3	Dari 7i33
9	IDX1	Dari 7i33	35	IDX2	Dari 7i33
11	IDX0	Dari 7i33	37	PWM3	Ke 7i33
13	PWM1	ke 7i33	39	PWM2	Ke 7i33
15	PWM0	Ke 7i33	41	DIR3	Ke 7i33
17	DIR1	Ke 7i33	43	DIR2	Ke 7i33
19	DIR0	Ke 7i33	45	/ENA3	Ke 7i33
21	/ENA1	Ke 7i33	47	/ENA2	Ke 7i33
23	/ENA0	Ke 7i33	49	+5V POWER	Ke 7i33
25	QB3	Dari 7i33	genap	GND	

3.2 Mesa 7i33

7i33 adalah antarmuka analog servo 4 axis dengan *Mesa's anything I / O board* yang digunakan untuk aplikasi pengendalian gerakan. Model 7i33 mengubah input PWM dan direction yang berasal dari board I / O dan mengkonversinya menjadi tegangan analog ±10V yang kemudian dikoneksikan langsung pada analog input servo amplifiers. Selain itu, 7i33 juga menghubungkan antara input sinyal dari encoder dengan board I / O. Sinyal input encoder dapat berupa TTL input maupun RS-422 input. Gambar 3.3 merupakan gambar dari mesa 7I33.



Gambar 3.3 Mesa 7I33 [8]

3.2.1 Spesifikasi dan Konfigurasi

Spesifikasi menjadi hal yang sangat penting dalam mengonfigurasikan mesa 7I33. Pada tabel 3.3 akan dijelaskan tentang spesifikasi-spesifikasi dari mesa 7I33.

Tabel 3.3.spesifikasi mesa 7I33

	Min	Maks	Satuan
Power Supply	4,75	5,25	VDC
Power Consumption (tanpa beban luar)	---	100	mA
Analog Output Voltage	±9,7	±10,3	V
Beban minimum Aout	5K		ohm
Output ripple @ 100KHz PWM		2	%FS
Output ripple @ 6MHz PDM		0,2	%FS
Linearity (PWM or PDM)		0,2	%FS
Encoder Frequency (TTL)	DC	1	MHz
Encoder Frequency (RS-422)	DC	10	MHz
Operating Temperatur	0	+70	^{0}C
Operating temperature (-I version)	-40	+85	^{0}C
Operation Humidity	0	95%	NON-COND

Berdasarkan spesifikasi diatas, mesa 5I20 dapat dikonfigurasikan seperti pada gambar 3.4

Gambar 3.4 konfigurasi mesa 7I33

i. *Controller connector*

Controller connector adalah pin konfigurasi yang menghubungkan antara mesa 5I20 dan mesa 7I33. Pada tabel 3.4 merupakan konfigurasi *controller connector*.

Tabel 3.4 konfigurasi controller connector

PIN	FUNGSI	ARAH SINYAL	PIN	FUNGSI	ARAH SINYAL
1	QB1	Dari 7i33	27	QA3	Dari 7i33
3	QA1	Dari 7i33	29	QB2	Dari 7i33
5	QB0	Dari 7i33	31	QA2	Dari 7i33
7	QA0	Dari 7i33	33	IDX3	Dari 7i33
9	IDX1	Dari 7i33	35	IDX2	Dari 7i33
11	IDX0	Dari 7i33	37	PWM3	Ke 7i33
13	PWM1	Ke 7i33	39	PWM2	Ke 7i33
15	PWM0	Ke 7i33	41	DIR3	Ke 7i33
17	DIR1	Ke 7i33	43	DIR2	Ke 7i33
19	DIR0	Ke 7i33	45	/ENA3	Ke 7i33
21	/ENA1	Ke 7i33	47	/ENA2	Ke 7i33
23	/ENA0	Ke 7i33	49	+5V POWER	Ke 7i33
25	QB3	Dari 7i33	genap	GND	

ii. TTL/RS-422 *selection*

TTL/RS-422 *selection* memiliki 4 buah *jumper*, yang tiap *jumper*-nya mengkonfigurasikan mode TTL atau RS-422 pada tiap *channel* 7i33. *Jumper* W1, W2, W3, W4 yang menentukan mode dari *input encoder* tersebut. Apabila *jumper* pada posisi turun maka *input* TTL yang digunakan, begitu pula sebaliknya apabila *jumper* ada pada posisi naik maka mode RS-422 yang digunakan. Pada tabel 3.5 merupakan tabel konfigurasi tiap jumper untuk pemilihan mode encoder tersebut.

Tabel 3.5 konfigurasi TTL/RS-422 *selection*

Jumper	Fungsi	*Default Setting*
W4	CH0 TTL/RS-422 *select*	bawah = TTL

W3	CH1 TTL/RS-422 *select*	bawah = TTL
W2	CH2 TTL/RS-422 *select*	bawah = TTL
W1	CH3 TTL/RS-422 *select*	bawah = TTL

iii. Servo amplifier/ encoder connector

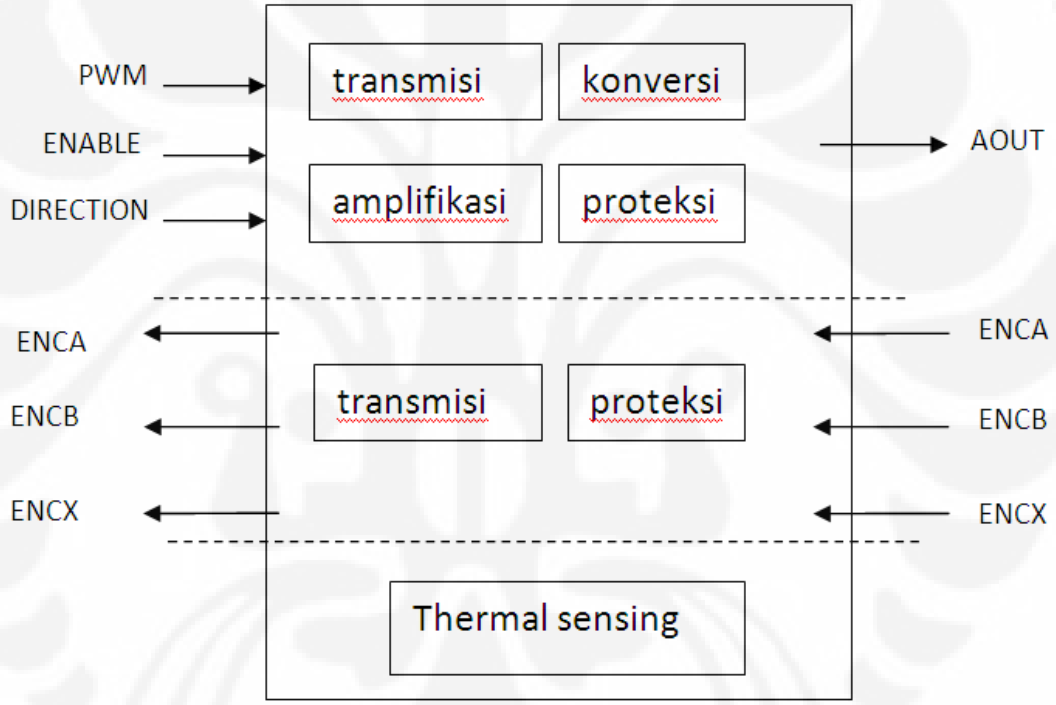
Konfigurasi pin dari mesa 7I33 dan servopack terdapat pada tabel 3.6.

Tabel 3.6 konfigurasi *Servo amplifier/ encoder connector*

TB1 pin	Fungsi	arah sinyal	TB2 pin	Fungsi	arah sinyal
1	ENCA0	Ke 7I33	1	ENCA1	Ke 7I33
2	/ENCAO	Ke 7I33	2	/ENCA1	Ke 7I33
3	GND	Dari 7I33	3	GND	Dari 7I33
4	ENCB0	Ke 7I33	4	ENCB1	Ke 7I33
5	/ENCB0	Ke 7I33	5	/ENCB1	Ke 7I33
6	+5V	Dari 7I33	6	+5V	Dari 7I33
7	IDX0	Ke 7I33	7	IDX1	Ke 7I33
8	/IDX0	Ke 7I33	8	/IDX1	Ke 7I33
9	GND	Dari 7I33	9	GND	Dari 7I33
10	AOUT0	Dari 7I33	10	AOUT1	Dari 7I33
11	GND	Dari 7I33	11	GND	Dari 7I33
12	ENABLE0	Dari 7I33	12	ENABLE1	Dari 7I33
13	ENCA2	Ke 7I33	13	ENCA3	Ke 7I33
14	/ENCA2	Ke 7I33	14	/ENCA3	Ke 7I33
15	GND	Dari 7I33	15	GND	Dari 7I33
16	ENCB2	Ke 7I33	16	ENCB3	Ke 7I33
17	/ENCB2	Ke 7I33	17	/ENCB3	Ke 7I33
18	+5V	Dari 7I33	18	+5V	Dari 7I33
19	IDX2	Ke 7I33	19	IDX3	Ke 7I33

20	/IDX2	Ke 7I33	20	/IDX3	Ke 7I33
21	GND	Dari 7I33	21	GND	Dari 7I33
22	AOUT2	Dari 7I33	22	AOUT3	Dari 7I33
23	GND	Dari 7I33	23	GND	Dari 7I33
24	ENABLE2	Dari 7I33	24	ENABLE3	Dari 7I33

3.2.2 Pembahasan rangkaian 7I33



Gambar 3.5. diagram blok mesa 7I33

7I33 merupakan *servo amplifier* yang menyambungkan antara FPGA dengan *servopack* SGDM. Data yang dikeluarkan oleh Mesa 5I20 adalah sinyal *digital* sedangkan masukkan dari *servopack* adalah *analog*. Dengan demikian dibutuhkan suatu *driver* yang digunakan untuk menghubungkan diantara keduanya. Berdasarkan gambar 3.5ditunjukkan diagram blok dari mesa 7I33 yang memiliki tujuan utama untuk men-transmisikan data dari mesa 5I20 menuju servopack.

Berdasarkan data dari hasil percobaan didapatkan mesa 7I33 berfungsi untuk,

a. Transmisi sinyal

Tujuan utama dari Mesa 7I33 merupakan mentransmisikan sinyal-sinyal dari 5I20 sehingga dapat menjadi referensi sinyal untuk servopack. Selain itu, 7I33 juga berfungsi meneruskan sinyal encoder ke mesa 5I20.

b. Digital ke analog conveter (DAC)

Sinyal referensi yang digunakan oleh SGDM adalah analog sedangkan keluaran dari Mesa 5I20, dengan demikian dibutuhkan suatu driver yang digunakan untuk konversi tegangan DC menjadi tegangan Analog. Mekede yang digunakan untuk konversi ini adalah dengan menggunakan filter integraker. Dengan membuat low-pass filter pada rangkaian 7I33. Namun demikian untuk medapatkan nilai keluaran referensi yang sesuai dengan nilai masukkannya maka diperlukan spesifikasi yang baik. Salah satu cara terbaik adalah dengan menggunakan frekuensi tinggi sehingga fungsi kapasiker dalam charge dan discharge akan sangat terasa. Selain itu dengan menggunakan modulasi pulse density modulation sebagai pengatur tingkat keluaran dari sinyal. Pemilihan PDM sebagai modulasinya karena PDM memvariasikan jarak antar sinyal high. Dengan demikian, pada waktu discharge tegangan tidak akan terlalu signifikan turun karena sinyal high kemudian akan menaikkan kembali nilai tegangan tersebut. Selain itu, beban pada output harus lebih besar dari 5K ohm agar tegangan tidak drop pada saat diaktifkan.

c. Konversi tegangan

Pada Mesa 5I20 terdapat 3 buah input (PWM, Enable, Direction)ke 7I33 yang kemudian akan dijadikan 1 pin output (Aout) yang kemudian disambungkan pada pin di servo pack. Nilai referensi pada servo pack mencakup ±10 V. Nilai dari duty cycle PWM kemudian akan menjadi referensi tegangan yang akan dikonversikan dua kalinya dari nilai PWM. Conkeh: PWM dengan duty cycle 50% akan menghasilkan tegangan analog sebesar 5V. Nilai plus dan minus menandakan arah perputaran dari

moker. Conkeh : Aout bernilai +10 V maka moker akan bergerak searah jarum jam demikian pula sebaliknya jika Aout bernilai -10V maka moker akan bergerak berlawanan dengan arah jarum jam.

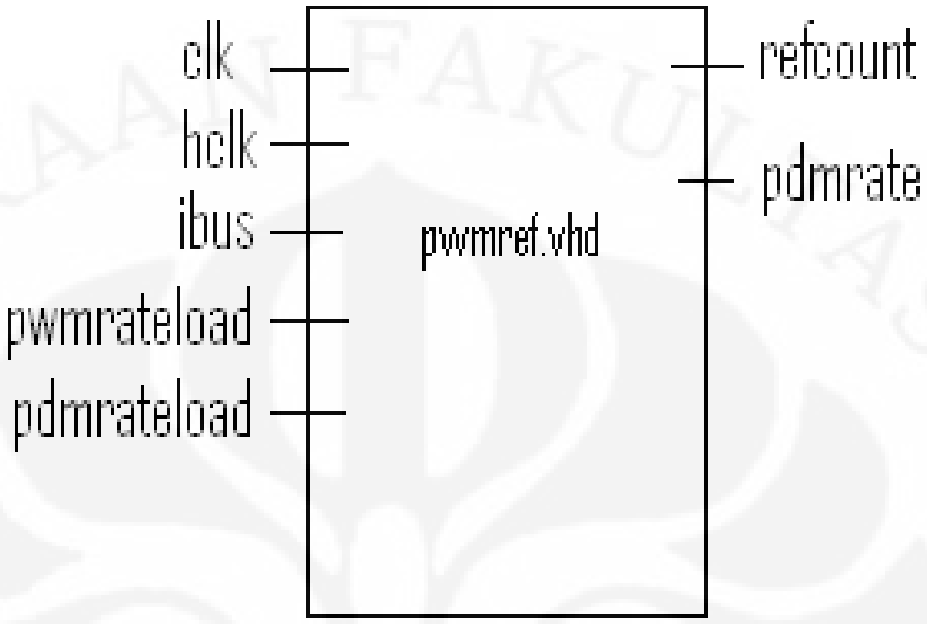
d. Proteksi

Selain hal-hal yang ada diatas, Mesa 7I33 bertujuan pula untuk proteksi. Hal ini dilakukan dengan cara menggunakan opkecouler yang bertujuan untuk memisahkan power yang digunakan untuk sinyal yang berasal dari computer dengan power untuk moker. Dengan demikian, tidak ada koneksi power (hanya data) antara Mesa 5I20 dengan servo pack. Selain itu, pada 7I33 terdapat pula LM 35 yang bertujuan sebagai thermal sensing. Ketika LM 35 membaca keadaan sekarang berada pada level diatas 85^0 maka saluran power dari 5I20 akan berhenti dan tidak mengirimkan sinyal pada servo pack. Hal ini disebabkan untuk menjaga agar system berjalan pada kondisi yang sesuai dengan spesifikasi.

3.3 Pembangkitan PWM-PDM pada FPGA Mesa 5i20

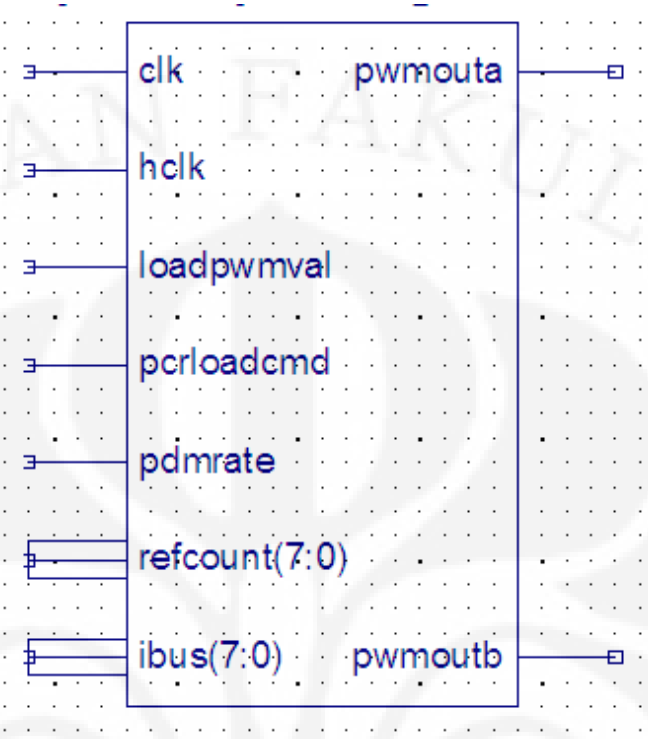
Pembangkitan PWM-PDM pada mesin CNC ini dibangkitkan oleh FPGA mesa 5i20 dikarenakan untuk membangkitkan PWM-PDM dibutuhkan perintah yang cukup rumit serta membutuhkan waktu yang cukup lama. Agar perintah yang masuk PC tidak dibebani dengan pembangkitan sinyal PWM-PDM tersebut, maka pembangkitan sinyal tersebut dilakukan pada FPGA. Namun demikian, komputer tetap mengeluarkan mode-mode yang digunakan oleh FPGA serta nilai referensi yang akan dibangkitkan.

Nilai PWM-PDM ini dikeluarkan komputer berdasarkan nilai balikan encoder yang kemudian dengan pengendali PID yang berada pada komputer akan mengatur sinyal keluaran PWM-PDM tersebut. Program pada FPGA untuk membangkitkan nilai PWM dan PDM terdiri dari dua file dengan extension .vhd, yaitu pwmref.vhd dan pwmpdmgen.vhd.



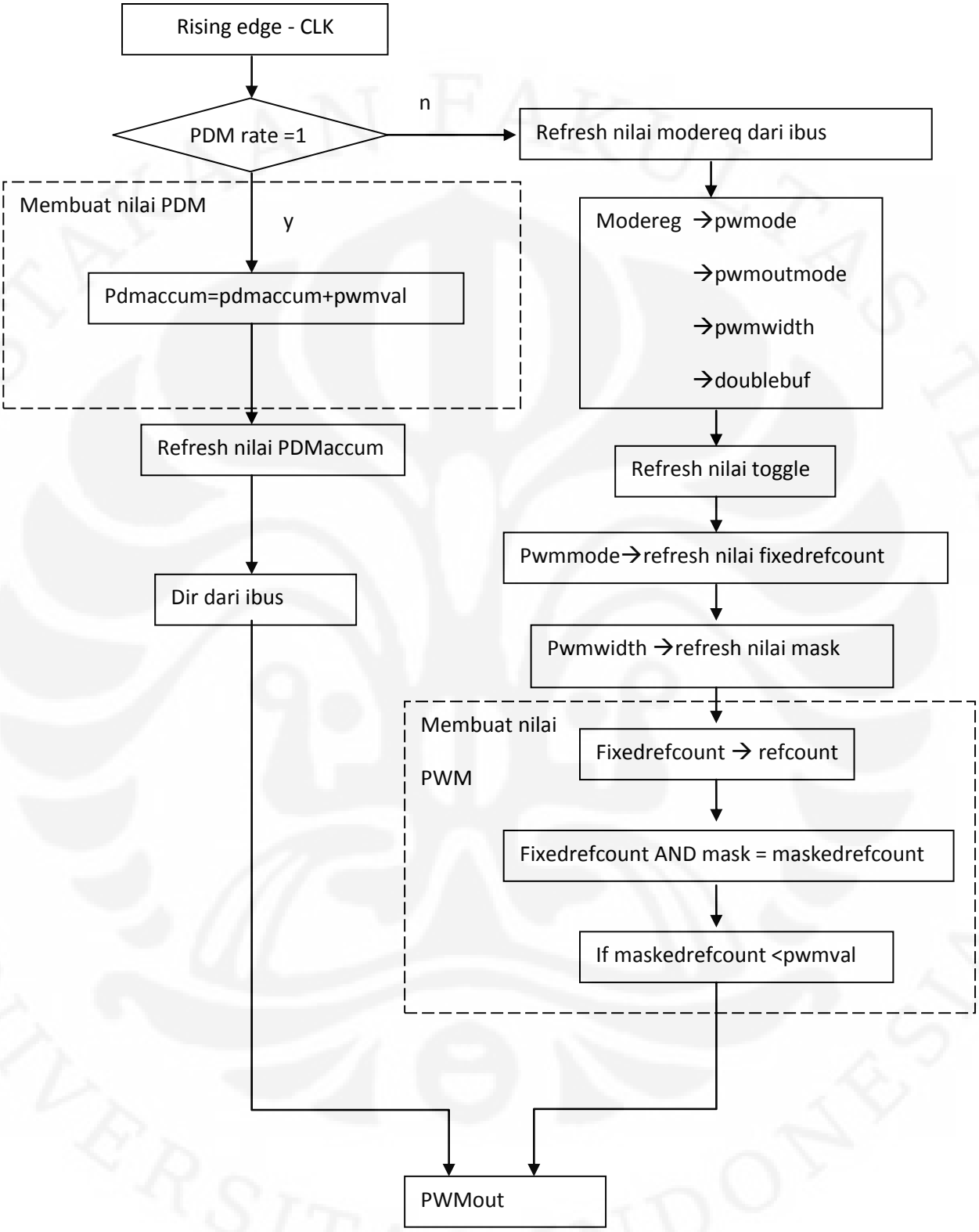
Gambar 3.6. diagram blok pwmref.vhd

Pada gambar 3.6 dapat dilihat bahwa inputan dari blok diagram pwmref.vhd adalah clk, hclk, ibus, pwmrateload, pdmrateload dengan output refcount dan pdmrate. Input clk dan hclk digunakan untuk men-trigger code yang ada pada proses kemudian mengunduh nilai pwmrateload dan pdmrateload dari computer kemudian memrosesnya sehingga dodapatkan refcount dan pdm rate yang kemudian dipakai untuk membangkitkan sinyal pada pwmpdmgen.vhd. Berikut ini adalah gambar diagram blok dari pwmpdmgen.vhd:



Gambar 3.7. diagram blok pwmpdmgen.vhd

Pada gambar 3.7 dapat dilihat bahwa inputan dari blok pwmpdmgen.vhd adalah clk, hclk, loadpwmval, pcrloadcmd, pdmrate, refcount, dan ibus, sedangkan outputnya adalah pwmouta dan pwmoutb. Clk dan hclk digunakan untuk men-trigger code yang terdapat pada program. Pdmrate dan refcount didapatkan dari pwmref.vhd yang digunakan untuk mendapatkan nilai referensi dalam membangkitkan PWM-PDM. Loadpwmval serta pcrloadcmd digunakan untuk mendapatkan variabel-variabel yang penting pada proses di pwmpdmgen.vhd. sedangkan ibus digunakan untuk men-generate mode-mode dalam membangkitkan PWM-PDM tersebut. Output dari pwmpdmgen.vhd kemudian akan dikeluarkan dalam bentuk pwmouta dan pwmoutb yang merupakan keluaran dari FPGA berbentuk sinyal pengendali digital untuk PWM-PDM serta arah dari pergerakan motor



Gambar 3.8 diagram alir pwmpdmgen.vhd

Dari gambar 3.8, dapat dilihat diagram alir pembentukan sinyal PWM atau PDM. Program akan mulai ketika port clk mendapatkan trigger, selanjutnya akan membaca mode modulasi apa yang akan dipakai, apabila PDMrate bernilai 1 maka sinyal modulasi PDM yang akan dibangkitkan oleh FPGA sedangkan apabila PDM rate bernilai 0 maka sinyal kendali PWM yang akan dibangkitkan oleh FPGA. Berdasarkan gambar 3.6 dan 3.7 dapat dilihat bahwa pdmrate didapatkan dari pwmref.vhd dan kemudian disambungkan sehingga menjadi input pada pwmpdmgen.vhd. Pada mode PDM, nilai PDM akan berganti sesuai dengan nilai referensi yang dikeluarkan melalui pwmref.vhd oleh komputer. Nilai tersebut kemudian dimasukkan ke dalam pdmaccum. Setelah itu, nilai itu akan ditambahkan dengan sinyal *direction* dan akan dikeluarkan pada port PWMout.

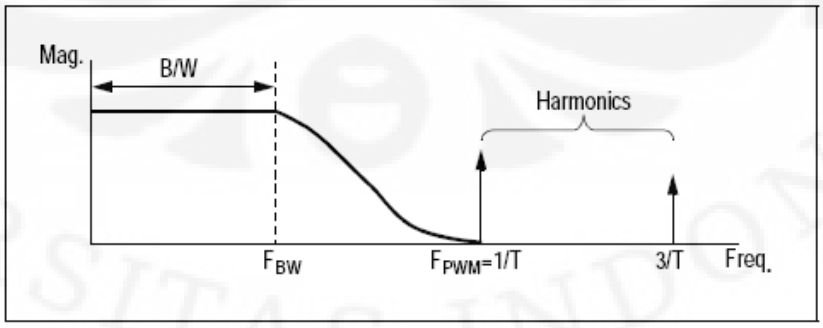
Sedangkan untuk PWM, pembangkitan sinyalnya agak rumit. Hal ini berkaitan dengan frekuensi yang dibangkitkan harus selalu sama. Pertama-tama, modereg diperbaharui nilainya dari ibus yang menjadi parameter-parameter yang digunakan pada proses berikutnya. Variabel tersebut adalah pwmmode, pwmoutmode, pwmwidth, dan doublebuf. Pwmmode digunakan untuk memperbaharui nilai dari referensi untuk membangkitkan PWM. Pwmoutmode digunakan untuk mengkonfigurasikan kombinasi mode keluaran PWM, PDM, serta direction yang tepat pada FPGA. Pwmwidth digunakan untuk mode dalam mengubah nilai dari mask. Doublebuf digunakan sebagai *flag* dapat membangkitan pwmval sedangkan toggle digunakan sebagai *flag* dalam pembangkitan nilai fixedrefcount. Fixedrefcount adalah nilai dari PWM yang menjadi referensi. Selanjutnya nilai mask dan fixedrefcount akan di hubungkan dengan gerbang AND dan disebut maskedrefcoun selanjutnya nilai maskedrefcount ini dibandingkan dengan nilai pwmval dan kemudian akan didapatkan sinyal PWM-nya.

3.4 DAC (Digital ke Analog Converter) dengan Low Pass Filter

Sinyal PWM-PDM yang dibangkitkan oleh FPGA mesa 5i20 harus dikonversikan dulu ke sinyal analog agar dapat digunakan sebagai referensi untuk

servo pack. Sinyal *square wave* (digital) dapat dirubah ke sinyal analog dengan menggunakan *low pass filter*. Sinyal digital pada pembahasan ini digunakan PWM, hal ini dikarenakan PWM memiliki nilai high dan kemudian low yang pasti pada tiap frekuensi sedangkan pada PDM nilai high dan low pada kondisi selanjutnya masih dikompensasi perhitungan dengan keadaan sebelumnya sehingga sulit untuk mendapatkan nilai pasti dari suatu PDM. Dengan demikian,dengan menggunakan PDM tidak bisa dimasukkan dalam perhitungan untuk mendapatkan nilai analognya.

Dengan pemilihan low-pass filter yang baik maka akan dapat dihasilkan sinyal analog dengan *ripple* yang sedikit sehingga akan mendekati nilai referensi analog tertentu. Namun demikian tidak mungkin untuk mendapatkan nilai analog yang diinginkan tanpa ada *ripple-ripple* tegangan dikarenakan digunakannya kapasitor. Prinsip perubahan sinyal digital ke analog adalah penggunaan dari *low-pass filter* sebagai *integrator*. Perubahan tersebut dapat dilakukan dikarenakan prinsip *charge-discharge* pada kapasitor. Prinsip ini yang kemudian digunakan untuk melihat respon dari sinyal digital itu sendiri. Besar frekuensi dari suatu sinyal akan sangat mempengaruhi keluaran dari sistem tersebut. Semakin besar frekuensi dari sinyal maka keluaran analog dengan *ripple* yang kecil dapat dibuat, berbeda dengan pada penggunaan frekuensi rendah. Dikarenakan respon dari kapasitor yang cepat sedangkan respon dari sinyal sangat lambat sehingga tidak akan berpengaruh secara signifikan pada keluaran sistem. Gambar 3.10 adalah gambar respon *low-pass filter* dalam domain frekuensi.



Gambar 3.10. respon *low pass filter* dalam domain frekuensi [9]

Dengan nilai *time constant* dari 7I33 sebesar 150µs, maka

$$f = \frac{1}{2\pi RC} = \frac{1}{2\pi 150.10^{-6}} = 1khz$$

Dengan demikian didapatkan nilai F_{BW} adalah 1 khz dan penggunaan sinyal digital dari mesa 5I20 adalah sinyal PWM dengan frekuensi 100khz.

$$dB = 10\log\left(\frac{V_{out}}{V_{in}}\right)^2 = 10\log|H(f)|^2 \quad (3.1)$$

$$|H(f)| = \frac{1}{\sqrt{1+(2\pi\ x\ f_{PWM}\ x\ RC)^2}} \quad (3.2)$$

$$dB = -10\log[1 + (2\pi\ x\ f_{PWM}\ x\ RC)^2] \quad (3.3)$$

$$= -10\log[1 + (2\pi\ x\ 10^5\ x\ 150\ x\ 10^{-6})^2] = -39\ dB$$

Dengan gain sebesar -39db serta dengan PWM berdutycycle 50 % maka akan didapatkan tegangan *low pass* sebesar,

$$V_{main} = 2\ x\ V_{maks} x\ dutycycle\ x\ sinc(dutycycle) \quad (3.4)$$

$$Vmain = 2\ x\ 10\ x\ 50\%\ x\ 0.63 = 6{,}3$$

$$V_{noise} = 10^{\frac{dB}{20}}\ x\ V_{main} \quad (3.5)$$

$$V_{noise} = 10^{\frac{-39}{20}}\ x\ 6{,}3 = 0{,}0756$$

$$V_{peak\ to\ peak} = V_{maks}\ x\ dutycycle\ \pm V_{noise} \quad (3.6)$$

$$V_{lowpass} = 5\ \pm 0{,}0756\ V$$

Berdasarkan perhitungan diatas dapat disimpulkan bahwa besar frekuensi PWM berbanding terbalik dengan V_{noise}. Dengan demikian, semakin besar frekuensi dari PWM maka V_{noise} akan semakin kecil sehingga *level ripple* dari sinyal analognya pun akan semakin kecil.

BAB 4

ANALISA

Dalam bab ini,penulis akan membahas tentang analisa dari sinyal PWM dan PDM yang berasal dari *interface* mesin CNC yang dikontrol melalui program EMC2. Pembangkitan sinyal kendali digunakan sebagai nilai referensi yang digunakan untuk servopack. Sinyal pengendali pada komputer kemudian akan dibangkitkan oleh FPGA mesa 5I20 dalam bentuk sinyal modulasi PWM atau PDM. Namun demikian, pembangkitan sinyal pengendali pada komputer adalah sinyal digital yang akan dikonversi menjadi tegangan analog oleh *low-pass filter* yang ada pada rangkaian mesa 7I33.

PWM dan PDM adalah salah satu bentuk modulasi yang sangat penting dalam pengendalian suatu motor. Dengan penggunaan metode modulasi tersebut, kita akan dapat mengendalikan kecepatan motor dengan sangat baik. Dalam analisa ini akan dilakukan pengujian pengendalian motor dengan menggunakan sinyal modulasi PWM dan PDM. Dalam pengujian ini, terdapat 2 buah variasi yang dianalisa, yaitu frekuensi serta kecepatan dari motor.

Dalam pengambilan data, data yang diambil adalah data ketika kecepatan motor hampir mencapai kecepatan referensi. Namun demikian dikarenakan pengendalian motor yang digunakan pada EMC2 menggunakan PID, maka terdapat kompensasi kecepatan terhadap perubahan jarak pada nilai *output* tiap *sampling time* sehingga tidak mungkin mendapatkan kecepatan referensi secara konstan. Selain itu, dikarenakan prinsip *filter* yang menggunakan *low pass filter* yang bekerja sebagai integrator, maka akan terjadi *ripple* pada *output* dari *filter*. Dengan demikian, data yang diambil untuk pengolahan data serta analisa adalah tegangan rata-rata dari *output*.

Dalam EMC2, jarak direpresentasikan dalam inchi. Dengan demikian dibuat persamaan untuk mendapatkan jarak putar dari motor. Jarak putar motor untuk

mencapai 1 inchi adalah sebanyak 8,46 putaran dan pada motor ini digunakan encoder dengan 2048 ppr. Dengan demikian didapatkan nilai variabel putaran encoder untuk mendapatkan 1 inchi motor adalah 17340 pulse.

$$v_{motor} = \frac{Pulse_{konversi}}{PPR_{encoder}} x\ v_{konversi} \quad (4.1)$$

Berdasarkan perhitungan diatas,

Dengan $V_{konversi}$=180, maka didapatkan kecepatan motor = 1524 rpm

Dengan $V_{konversi}$=120, maka didapatkan kecepatan motor = 1016 rpm

Dengan $V_{konversi}$=60, maka didapatkan kecepatan motor = 508 rpm

Kecepatan maksimum dari motor yaskawa yang dipakai dalam mesin CNC ini adalah 3000 rpm dengan nilai referensi tegangan analog 10V. Berdasarkan data yang diperoleh dapat digambarkan hubungan antara nilai referensi tegangan dengan kecepatan motor. Berdasarkan gambar yang diplot berdasarkan data percobaan didapatkan bahwa kecepatan motor dengan tegangan referensi berbanding lurus. Dengan menggunakan persamaan linear didapat,

$$V_{motor} = \frac{V_{maks}}{v_{maks}}\ x\ v_{motor} \quad (4.2)$$

Dari persamaan diatas dengan memasukkan nilai kecepatan motor yang ingin dikendalikan, sehingga didapatkan tegangan referensi untuk motor.

Untuk 1524 rpm,

$$V_{motor} = \frac{10\ V}{3000\ rpm}\ x\ 1524\ rpm = 5{,}08\ V$$

Untuk 1016 rpm,

$$V_{motor} = \frac{10\ V}{3000\ rpm}\ x\ 1016\ rpm = 3{,}38\ V$$

Untuk 508 rpm,

$$V_{motor} = \frac{10\ V}{3000\ rpm}\ x\ 508\ rpm = 1{,}69\ V$$

Tabel 4.1 data hasil percobaan dengan menggunakan modulasi PDM

no	mode	f(MHz)	v_{motot}(rpm)	S(inch)	V_{maks}(V)	V_{min}(V)	$V_{rata\text{-}rata}$(V)	V_{pp}(V)
1	PDM	1	1524	8	5.8	5.4	5.6	0.4
2	PDM	1	1016	8	4.2	3.8	4	0.4
3	PDM	1	508	8	2.4	2.2	2.3	0.2
4	PDM	6	1524	8	5.6	4.8	5.2	0.8
5	PDM	6	1016	8	4	3.6	3.8	0.4
6	PDM	6	508	8	2.2	1.8	2	0.4

Berdasarkan tabel 4.1, didapatkan nilai tegangan referensi yang akan digunakan sebagai input servopack. Pada percobaan dengan metode PDM serta dengan frekuensi 6MHz didapatkan nilai Vmotor pada kecepatan 1524 rpm adalah 5,2 V, dengan demikian dapat dihitung *error* linear tegangan,

$$\%error = \frac{5{,}2V - 5{,}08\ V}{5{,}08\ V}\ x\ 100\% = 2{,}3\%$$

Untuk 1016 rpm,

$$\%error = \frac{3{,}8V - 3{,}38\ V}{3{,}38\ V}\ x\ 100\% = 12{,}4\%$$

Untuk 508 rpm,

$$\%error = \frac{2V - 1{,}69\ V}{1{,}69\ V}\ x\ 100\% = 18{,}3\%$$

Sedangkan untuk frekuensi 1MHz didapatkan nilai error,

Untuk 1524 rpm,

$$\%error = \frac{5{,}2V - 5{,}08\,V}{5{,}08\,V}\ x\ 100\% = 10{,}2\%$$

Untuk 1016 rpm,

$$\%error = \frac{4V - 3{,}38\,V}{3{,}38\,V}\ x\ 100\% = 18{,}3\%$$

Untuk 508 rpm,

$$\%error = \frac{2.3V - 1{,}69\,V}{1{,}69\,V}\ x\ 100\% = 36{,}0\%$$

Berdasarkan hasil perhitungan diatas dapat dilihat bahwa nilai error dari frekuensi 1 MHz lebih besar dari pada frekuensi 6 MHz. Hal ini disebabkan frekuensi 6 MHz memiliki sampling time yang lebih kecil sehingga dalam mengompensasi kecepatan terhadap encoder menjadi lebih cepat. Berbeda dengan pada frekuensi 1 MHz yang memiliki sampling time yang lebih besar sehingga dalam mengompensasi kecepatan terhadap encoder akan lebih lama.

Tabel 4.2 data hasil percobaan dengan menggunakan modulasi PWM

no	mode	f(kHz)	v_{motot}(rpm)	S(inch)	V_{maks}(V)	V_{min}(V)	$V_{rata\text{-}rata}$(V)	V_{pp}(V)
1	PWM	100	1524	8	6.8	4.8	5.8	2
2	PWM	100	1016	8	4.8	2.8	3.8	2
3	PWM	100	508	8	2.8	1	1.9	1.8
4	PWM	10000	1524	8	7	4	5.5	3
5	PWM	10000	1016	8	4.6	2.4	3.5	2.2
6	PWM	10000	508	8	3	0.8	1.9	2.2

Sedangkan untuk mode PWM frekuensi 10MHz dari hasil yang didapatkan pada tabel 4.2 didapatkan nilai *error*,

Untuk 1524 rpm,

$$\%error = \frac{5{,}5V - 5{,}08\ V}{5{,}08\ V}\ x\ 100\% = 8{,}2\%$$

Untuk 1016 rpm,

$$\%error = \frac{3{,}5V - 3{,}38\ V}{3{,}38\ V}\ x\ 100\% = 12{,}0\%$$

Untuk 508 rpm,

$$\%error = \frac{1.9V - 1{,}69\ V}{1{,}69\ V}\ x\ 100\% = 12{,}4\%$$

Sedangkan untuk frekuensi 100kHz didapatkan nilai *error*,

Untuk 1524 rpm,

$$\%error = \frac{5{,}8V - 5{,}08\ V}{5{,}08\ V}\ x\ 100\% = 14{,}1\%$$

Untuk 1016 rpm,

$$\%error = \frac{3{,}8V - 3{,}38\ V}{3{,}38\ V}\ x\ 100\% = 12{,}4\%$$

Untuk 508 rpm,

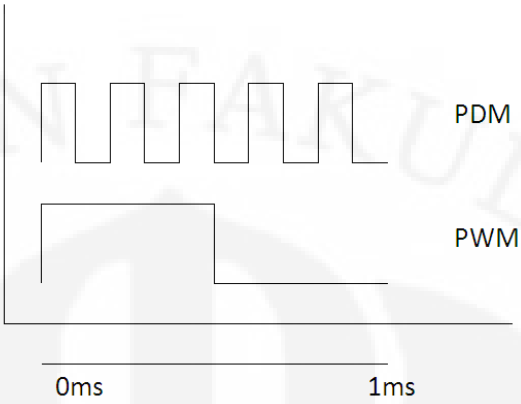
$$\%error = \frac{1.9V - 1{,}69\ V}{1{,}69\ V}\ x\ 100\% = 12{,}4\%$$

Berdasarkan hasil percobaan diatas dapat dilihat bahwa nilai *error* dari frekuensi 100 kHz lebih besar dari pada frekuensi 10 MHz. Hal ini disebabkan

frekuensi 10 MHz memiliki *sampling time* yang lebih kecil sehingga dalam mengompensasi kecepatan terhadap encoder menjadi lebih cepat. Berbeda dengan pada frekuensi 100 kHz yang memiliki *sampling time* yang lebih besar sehingga dalam mengompensasi kecepatan terhadap encoder akan lebih lama.

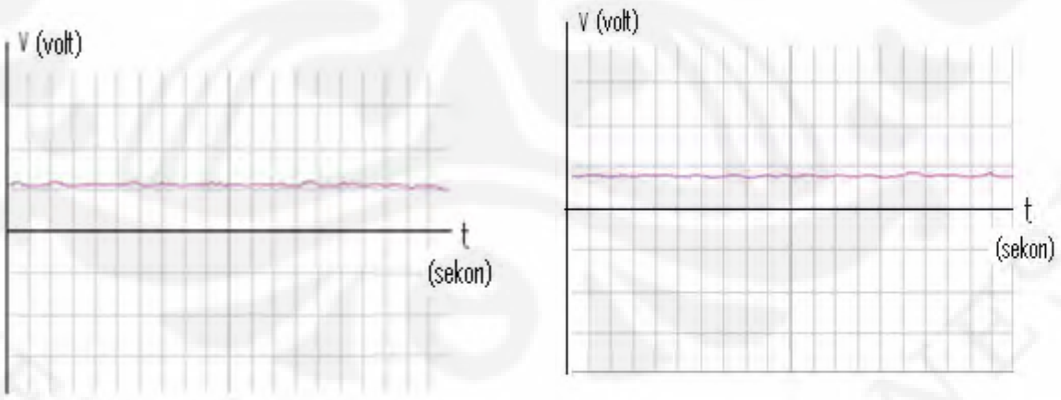
Berdasarkan perhitungan didapatkan bahwa *error* dari tiap modulasi masih cukup tinggi untuk sebuah pengontrolan yang membutuhkan ketelitian yang sangat tinggi sehingga kompensasi dari kesalahan-kesalahan dikontrol dengan pengendali PID agar didapatkan nilai posisi yang sesuai. Apabila dilihat dari hasil perhitungan error antara PWM dengan PDM tidak terlihat signifikan. Hal ini disebabkan cara modulasi yang mirip antara PWM dan PDM. Dengan menggunakan prinsip jumlah panjang pulsa *high* dan *low* sehingga tegangan rata-rata antara PWM dan PDM tidak terlihat berbeda secara signifikan.

Perbedaan antara PDM dan PWM akan terlihat pada nilai V *peak to peak*. Dari data diatas dapat dilihat bahwa V *peak to peak* dari PWM lebih besar dibandingkan V *peak to peak* dari PDM. Hal ini disebabkan oleh bentuk sinyal pemodulasian dari PWM yang menggunakan *dutycycle* dalam membentuk sinyal *high* dan low. Dengan demikian, waktu yang dibutuhkan untuk *charge* dan *discharge* kapasitor akan lebih lama. Lain hal dengan PDM, PDM mengatur nilai *high* dan low berdasarkan *density* dari suatu *sampling time*. Sebagai contoh : pada PWM *dutycycle* 50% dengan periode 1 ms, maka akan didapatkan sinyal digital *high* sepanjang 0.5ms dan low selama 0,5 ms. Berbeda pada PDM *dutycycle* 50% selama 1 ms dengan banyak sampling time 10 kali. Dengan demikian sinyal PDM-nya adalah 1010101010 dengan masing-masing sinyal *high* dan *low* selama 0,2 ms. Penjelasan diatas dapat dilihat pada gambar 4.1. Dengan konfigurasi seperti itu maka waktu *charge* dan *discharge* dari PDM akan lebih singkat dari PWM. Berdasarkan penjelasan diatas dapat disimpulkan bahwa sehingga nilai V *peak to peak* akan menjadi semakin kecil.



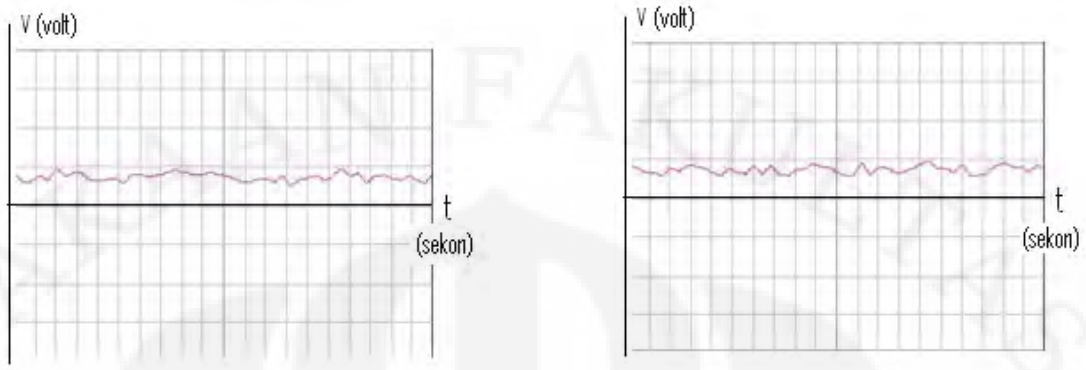
Gambar 4.1 implementasi sinyal PWM dan PDM

Perbedaan metode dalam modulasi antara PWM dan PDM seperti yang ditunjukkan pada paragraph diatas ini membuat *output* yang berbeda pada rangakaian *filter*. Perbedaan sinyal keluaran hasil percobaan antara penggunaan PWM dan PDM akan dijelaskan dengan menganalisa hasil tegangan analog dari 7I33 dengan dua tipe modulasi tersebut. Perbedaan cara pemodulasian tersebut secara langsung membuat hasil output dari 7I33 dengan rangkaian *low pass filter* menjadi berbeda pula. Pada gambar 4.2 dan 4.3 adalah gambar yang menjelaskan perbedaan hasil *output* antara PWM dan PDM dalam time/div sebesar 100ms dan volt/div sebesar 5v:



(a) 1 MHz (b) 6MHz

Gambar 4.2 Sinyal DC output filter dengan frekuensi mode PDM 1MHz dan 6 MHz



(a) 1 MHz (b) 6MHz

Gambar 4.3 Sinyal DC output filter dengan frekuensi mode PWM 1MHz dan 6 MHz

Berdasarkan data diatas dapat dilihat bahwa tegangan *noise* antara PWM dan PDM berbeda cukup jauh. Tegangan *noise* merepresentasikan proses *charge* dan *discharge* dari kapasitor yang digunakan dalam *low pass filter*. Pada PWM, tegangan *noise* ini disebabkan jarak antara pulsa *high* yang terlalu jauh dikarenakan pada percobaan digunakan kecepatan yang tidak terlalu cepat. Pada percobaan digunakan kecepatan 508 rpm, 1016 rpm yang lebih kecil daripada *duty cycle* 50% dan 1524 rpm dengan *duty cycle* 50%. Dengan nilai yang cukup jauh maka proses *charge* dan *discharge* dari kapasitor akan semakin lama, dengan demikian tegangan *noise* ini akan semakin besar. Berbeda dengan PDM, karena jarak dari pulsa *high* yang diatur. Dengan demikian waktu proses *charge* dan *discharge* dari kapasitor dapat dipersingkat sehingga tegangan *noise* yang dihasilkan akan semakin kecil.

Berdasarkan hasil analisa diatas dapat disimpulkan untuk mendapatkan sinyal analog dari sinyal digital dengan menggunakan *low-pass filter* lebih baik dengan menggunakan modulasi PDM daripada PWM.

BAB 5

KESIMPULAN

Berdasarkan hasil pengambilan data serta analisa, maka dapat ditarik kesimpulan, yaitu :

1. PWM dan PDM merupakan dua contoh modulasi yang baik untuk mengontrol motor
2. Pada PDM menghasilkan error tegangan referensi terhadap tegangan output sbb:
    - Pada kecepatan 508 rpm =18,3%
    - Pada kecepatan 1016 rpm =12,4%
    - Pada kecepatan 1524 rpm =2,3%

    Selanjutnya pada PWM dihasilkan,

    - Pada kecepatan 508 rpm =12,4%
    - Pada kecepatan 1016 rpm =12,04%
    - Pada kecepatan 1524 rpm =8,2%
3. Dalam DAC dengan menggunakan *filter* hasil output pada frekuensi tinggi lebih baik daripada frekuensi rendah
4. Dalam DAC dengan menggunakan *low pass filter*, Hasil dengan menggunakan PDM menghasilkan sinyal analog yang lebih baik daripada PWM karena memiliki *ripple* yang lebih kecil

DAFTAR REFERENSI

[1] Saroso, Dony Harris. "Desain Function Generator Berbasis PLD (FPGA)". Universitas Indonesia. Depok. 2009.

[2] Van der Spiegel, Jan. "VHDL tutorial". University of Pennsylvania. 31 desember 2009. http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html

[3]The Ohio State University. Filters. (15 Juni 2010). http://www.mecheng.osu.edu/electronics-laboratory/filters-low-pass-and-high-pass

[4] Nastavni Materijal. Folije. Universitas Studiorum Zagrabiensis MDCLXIX. 2008. http://www.phy.hr/~nnovosel/etuf/nastavni_materijali.php

[5]Wikipedia. File: Band-Reject Filter.svg. (15 Juni 2010). http://commons.wikimedia.org/wiki/File:Band-Reject_Filter.svg

[6]Wikipedia. Pulse Density Modulation. (15 Juni 2010). http://en.wikipedia.org/wiki/Pulse-density_modulation

[7] Mesa *5i20 Anything I/O Manual*. November 21, 2009. http://www.mesanet.com

[8] Mesa 7I33/7I33t Manual, Quad Analog Servo Amp Interface. November 21, 2009. http://www.mesanet.com

[9] Choi, June. D/A Converter Using PWM. MCU Application Team.

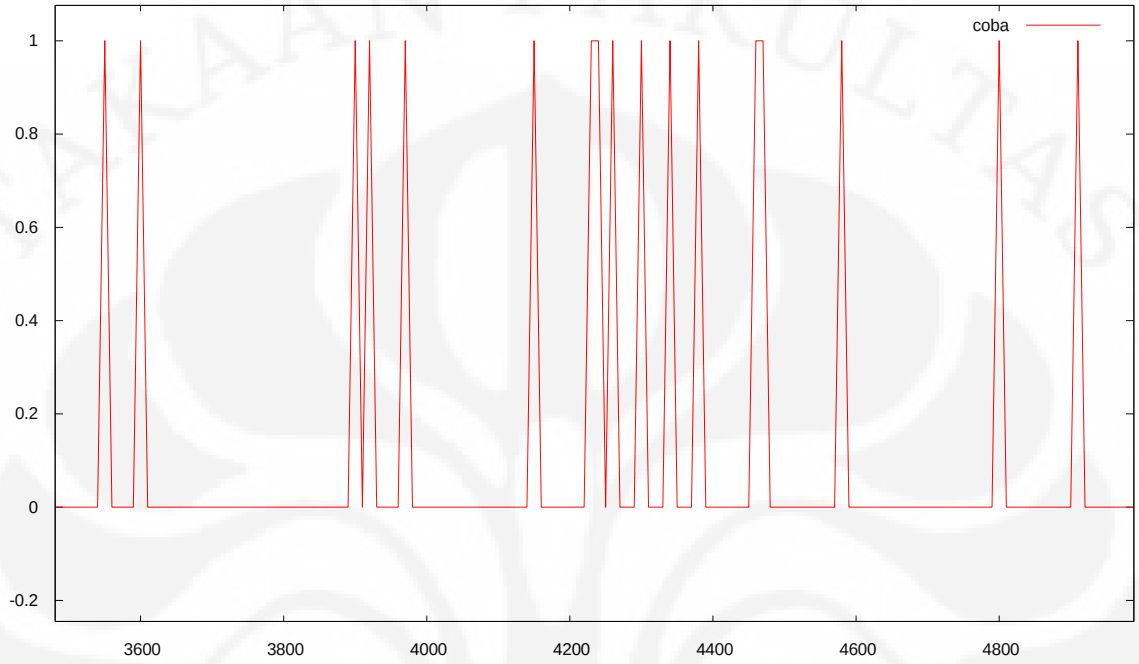
[10] Sumbodo, Wirawan. "Mesin CNC". Universitas Negeri Semarang. Semarang. 2008

[11] Permadi, Aditya Dyan, dkk. “Field Programmable Gate Arrays”. Institut Teknologi Telkom. Bandung. 2008

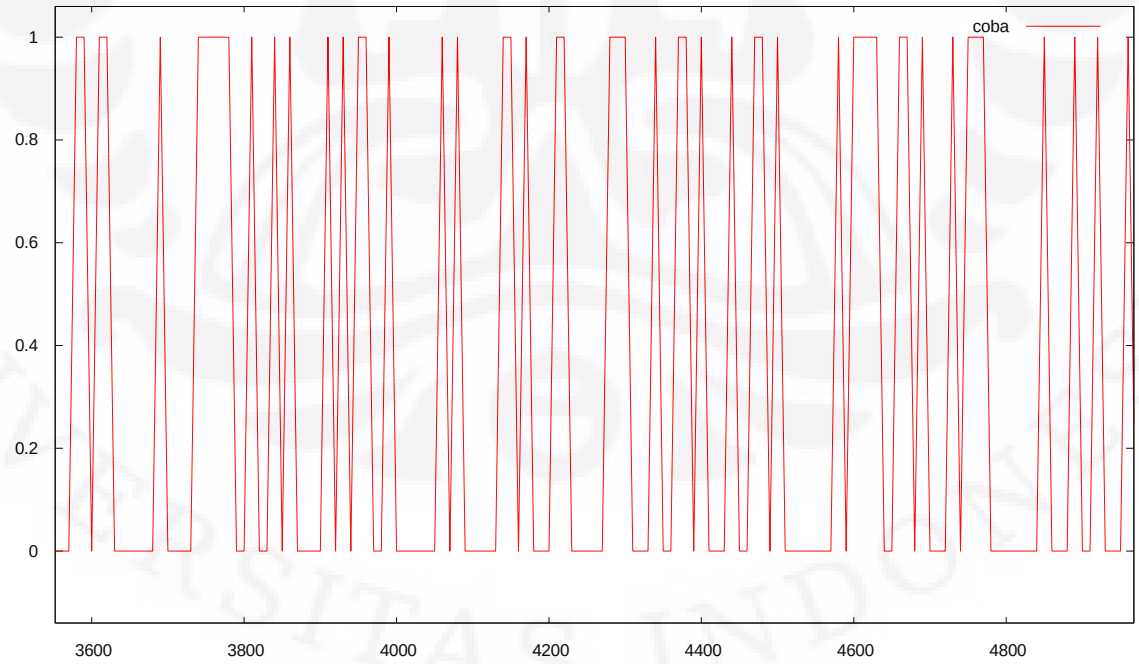
[12] Parnell, Karen and Nick Mehta. “Programmable Logic Design Quick Start Handbook”. Xilinx Inc.. 2003

Lampiran 1

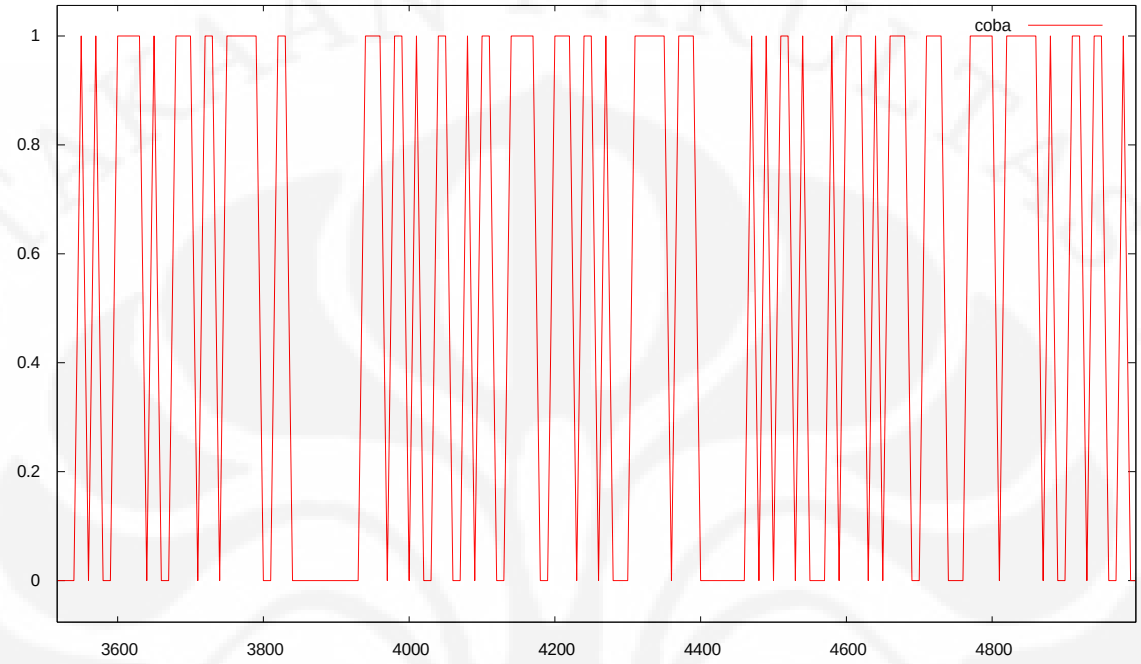
Sinyal input pada mode PDM dengan kecepatan 508rpm:



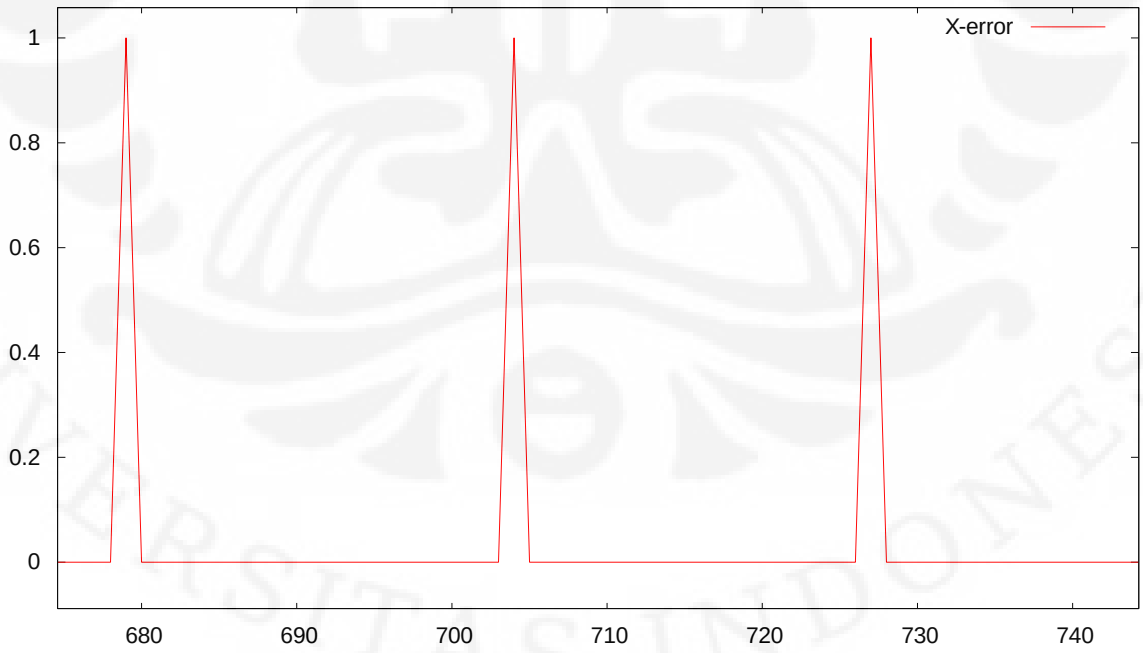
Sinyal input pada mode PDM dengan kecepatan 1016rpm:



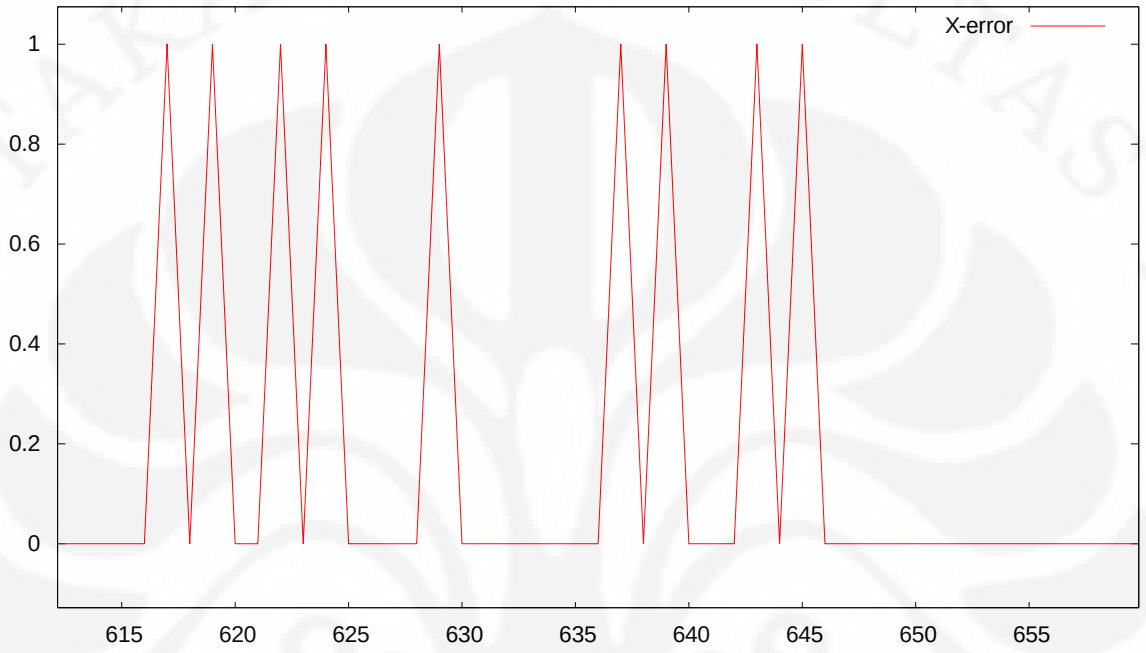
Sinyal input pada mode PDM dengan kecepatan 1524rpm:



Sinyal input pada mode PWM dengan kecepatan 508rpm:



Sinyal input pada mode PDM dengan kecepatan 1016rpm:



Lampiran 2
Source code pwmpdngen.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Copyright (C) 2007, Peter C. Wallace, Mesa Electronics
-- http://www.mesanet.com
-- This program is is licensed under a disjunctive dual license giving you
-- the choice of one of the two following sets of free software/open source
-- licensing terms:
--    * GNU General Public License (GPL), version 2.0 or later
--    * 3-clause BSD License
-- The GNU GPL License:
--     This program is free software; you can redistribute it and/or modify
--     it under the terms of the GNU General Public License as published by
--     the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
--     This program is distributed in the hope that it will be useful,
--     but WITHOUT ANY WARRANTY; without even the implied warranty of
--     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--     GNU General Public License for more details.
--     You should have received a copy of the GNU General Public License
--     along with this program; if not, write to the Free Software
--     Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301 USA
-- The 3-clause BSD License:
--     Redistribution and use in source and binary forms, with or without
--     modification, are permitted provided that the following conditions
--     are met:
--         * Redistributions of source code must retain the above copyright
--           notice, this list of conditions and the following disclaimer.
--         * Redistributions in binary form must reproduce the above
--           copyright notice, this list of conditions and the following
--           disclaimer in the documentation and/or other materials
--           provided with the distribution
--         * Neither the name of Mesa Electronics nor the names of its
--           contributors may be used to endorse or promote products
--           derived from this software without specific prior written
--           permission.
--
-- Disclaimer:
--
--     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
--     "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
--     LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
--     FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
--     COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
--     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
--     BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
--     LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
--     CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
--     LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
--     ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
--     POSSIBILITY OF SUCH DAMAGE.
--
```

```
entity pwmpdmgenh is
	generic (
		buswidth : integer;
		refwidth : integer
		);
	port (
		clk: in std_logic;
		hclk: in std_logic;
		refcount: in std_logic_vector (refwidth-1 downto 0);
		ibus: in std_logic_vector (buswidth -1 downto 0);
		loadpwmval: in std_logic;
		pcrloadcmd: std_logic;
		pdmrate : in std_logic;
		pwmouta: out std_logic;
		pwmoutb: out std_logic
	);
end pwmpdmgenh;

architecture behavioral of pwmpdmgenh is

signal pwmval: std_logic_vector (refwidth-2 downto 0);
signal prepwmval: std_logic_vector (refwidth-2 downto 0);
signal fixedrefcount: std_logic_vector (refwidth-2 downto 0);
signal pdmaccum: std_logic_vector (refwidth-1 downto 0);
alias  pdmbit: std_logic is pdmaccum(refwidth-1);
signal maskedrefcount: std_logic_vector (refwidth-2 downto 0);
signal pwm: std_logic;
signal dir: std_logic;
signal toggle: std_logic;
signal mask: std_logic_vector (refwidth-2 downto 0);
signal predir: std_logic;
signal premodereg: std_logic_vector(5 downto 0);
signal modereg: std_logic_vector(5 downto 0);
alias  pwmwidth: std_logic_vector(1 downto 0) is modereg(1 downto 0);
alias  pwmmode: std_logic is modereg(2);
alias  pwmoutmode: std_logic_vector(1 downto 0) is modereg(4 downto 3);
alias  doublebuf: std_logic is modereg(5);
signal loadpwmreq: std_logic;
signal oldloadpwmreq: std_logic;
signal olderloadpwmreq: std_logic;
signal loadpcrreq: std_logic;
signal oldloadpcrreq: std_logic;
signal olderloadpcrreq: std_logic;
signal oldtoggle: std_logic;
signal cyclestart: std_logic;

begin
	apwmgen: process  (clk,hclk,refcount,ibus,loadpwmval,pwmval,pwm,
		fixedrefcount,mask,pwmmode,toggle,oldtoggle,
			pwmwidth,olderloadpwmreq,olderloadpcrreq,
			pwmoutmode,dir,pdmaccum
		)

	begin
		if rising_edge(hclk) then
			if pdmrate = '1' then
```

```
                    pdmaccum <= ('0'&pdmaccum(refwidth-2 downto 0)) + ('0'&pwmval);
                end if;
                if oldloadpwmreq = '1'  and olderloadpwmreq = '1' then
                    pwmval <= prepwmval;
                    dir <= predir;
                    oldloadpwmreq <= '0';
                end if;
                if oldloadpcrreq = '1' and olderloadpcrreq ='1' then
                modereg <= premodereg;
                end if;

                olderloadpwmreq <= oldloadpwmreq;
                olderloadpcrreq <= oldloadpcrreq;
                if (loadpwmreq and (cyclestart or (not doublebuf))) = '1' then
                    oldloadpwmreq <= '1';
                end if;

--              oldloadpwmreq <= loadpwmreq and (cyclestartflag or (not doublebuf));

                oldloadpcrreq <= loadpcrreq;
                -- was combinatorial but now pipelined to meet 100 MHz timing
                if (UNSIGNED(maskedrefcount) < UNSIGNED(pwmval)) then
                    pwm <= '1';
                else
                    pwm <= '0';
                end if;
                case pwmmode is
                    when '0' => fixedrefcount <= refcount(refwidth-2 downto 0);
                    when '1' =>
                        if toggle = '1' then                    -- symmetrical mode
                            fixedrefcount <=  (not refcount(refwidth-2 downto 0));
                            else
                            fixedrefcount <= refcount(refwidth-2 downto 0);
                        end if;
                    when others => null;
                end case;
                oldtoggle <= toggle;
            end if; -- hclk

            maskedrefcount <= fixedrefcount and mask;

            if pwmmode = '1' then
                cyclestart <= ((not toggle) and oldtoggle); -- falling edge of toggle sym mode
            else
                cyclestart <= toggle xor oldtoggle;     -- both edges of toggle ramp mode
            end if;

            case pwmwidth is
                when "00" =>
                                                mask(refwidth-2 downto refwidth-4) <= "000";
                                                mask(refwidth-5 downto 0) <= ( others => '1');
                                                toggle <= refcount(refwidth-4);
                when "01" =>
                                                mask(refwidth-2 downto refwidth-3) <= "00";
                                                mask(refwidth-4 downto 0) <= ( others => '1');
                                                toggle <= refcount(refwidth-3);
```

```
                        when "10" =>
                                                mask(refwidth-2) <= '0';
                                                mask(refwidth-3 downto 0) <= ( others => '1');
                                                toggle <= refcount(refwidth-2);
                        when "11" =>
                                                mask <= (others => '1');
                                                toggle <= refcount(refwidth-1);
                        when others => null;
                end case;

                if rising_edge(clk) then -- 33/48/50 mhz local bus clock
                        if loadpwmval = '1' then
                                prepwmval <= ibus((refwidth-2)+16 downto 16); -- Fixme! only works for
buswidth 32
                                predir <= ibus(BusWidth -1);
                                loadpwmreq <= '1';
                        end if;
                        if pcrloadcmd = '1' then
                                premodereg <= ibus(5 downto 0);
                                loadpcrreq <= '1';
                        end if;
                end if; -- clk

                if olderloadpwmreq = '1' then -- asynchronous request clear, could use flancter but dont need
async clear
                        loadpwmreq <= '0';
                end if;

                if olderloadpcrreq = '1' then -- asynchronous request clear ""
                        loadpcrreq <= '0';
                end if;

                case pwmoutmode is
                        when "00" =>
                                pwmouta <= pwm;                         -- normal sign magnitude
                                pwmoutb <= dir;
                        when "01" =>
                                pwmouta <= dir;                 -- reversed pwm/dir = locked antiphase
                                pwmoutb <= pwm;
                        when "10" =>
                                if dir = '1' then                       -- up/down mode
                                        pwmouta <= pwm;
                                        pwmoutb <= '0';
                                else
                                        pwmouta <= '0';
                                        pwmoutb <= pwm;
                                end if;
                        when "11" =>
                                pwmouta <= pdmbit;
                                pwmoutb <= dir;
                        when others => null;
                end case;
--              pwmoutc <= ena;
                end process;
end behavioral;
```

Lampiran 3

Source code pwmref.vhd

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_ARITH.ALL;
use IEEE.std_logic_UNSIGNED.ALL;
--
-- Copyright (C) 2007, Peter C. Wallace, Mesa Electronics
-- http://www.mesanet.com
--
-- This program is is licensed under a disjunctive dual license giving you
-- the choice of one of the two following sets of free software/open source
-- licensing terms:
--
--    * GNU General Public License (GPL), version 2.0 or later
--    * 3-clause BSD License
--
--
-- The GNU GPL License:
--
--     This program is free software; you can redistribute it and/or modify
--     it under the terms of the GNU General Public License as published by
--     the Free Software Foundation; either version 2 of the License, or
--     (at your option) any later version.
--
--     This program is distributed in the hope that it will be useful,
--     but WITHOUT ANY WARRANTY; without even the implied warranty of
--     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--     GNU General Public License for more details.
--
--     You should have received a copy of the GNU General Public License
--     along with this program; if not, write to the Free Software
--     Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301 USA
--
--
-- The 3-clause BSD License:
--
--     Redistribution and use in source and binary forms, with or without
--     modification, are permitted provided that the following conditions
--     are met:
--
--         * Redistributions of source code must retain the above copyright
--           notice, this list of conditions and the following disclaimer.
--
--         * Redistributions in binary form must reproduce the above
--           copyright notice, this list of conditions and the following
--           disclaimer in the documentation and/or other materials
--           provided with the distribution.
--
--         * Neither the name of Mesa Electronics nor the names of its
--           contributors may be used to endorse or promote products
--           derived from this software without specific prior written
--           permission.
```

```
-- Disclaimer:
--
--     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
--     "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
--     LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
--     FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
--     COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
--     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
--     BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
--     LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
--     CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
--     LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
--     ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
--     POSSIBILITY OF SUCH DAMAGE.
--

entity pwmrefh is
                        generic (
                                buswidth : integer;
                                refwidth : integer
                                );
                        Port (
                                clk: in std_logic;
                                hclk: in std_logic;
                                refcount: out std_logic_vector (refwidth-1 downto 0);
                                ibus: in std_logic_vector (buswidth -1 downto 0);
                                pdmrate: out std_logic;
                                pwmrateload: in std_logic;
                                pdmrateload: in std_logic
                        );
end pwmrefh;

architecture behavioral of pwmrefh is

signal count: std_logic_vector (refwidth -1 downto 0);
signal pwmrateacc: std_logic_vector (16 downto 0);
alias  pwmratemsb: std_logic is pwmrateacc(16);
signal oldpwmratemsb: std_logic;
signal pwmratelatch: std_logic_vector (15 downto 0);
signal prepwmratelatch: std_logic_vector (15 downto 0);
signal pwmratelatchloadreq: std_logic;
signal oldpwmratelatchloadreq: std_logic;
signal olderpwmratelatchloadreq: std_logic;
signal pdmrateacc: std_logic_vector (16 downto 0);
alias  pdmratemsb: std_logic is pdmrateacc(16);
signal oldpdmratemsb: std_logic;
signal pdmratelatch: std_logic_vector (15 downto 0);
signal prepdmratelatch: std_logic_vector (15 downto 0);
signal pdmratelatchloadreq: std_logic;
signal oldpdmratelatchloadreq: std_logic;
signal olderpdmratelatchloadreq: std_logic;
signal prate: std_logic;
```

```
begin
	apwmref: process  (clk,
	    olderpwmratelatchloadreq, olderpdmratelatchloadreq, prate, hclk, count, ibus)
	begin
		if rising_edge(hclk) then                                   -- 100 Mhz high speed clock

			if oldpwmratelatchloadreq = '1' and olderpwmratelatchloadreq = '1' then
				pwmratelatch <= prepwmratelatch;
			end if;
			oldpwmratelatchloadreq    <= pwmratelatchloadreq;
			olderpwmratelatchloadreq <= oldpwmratelatchloadreq;
			pwmrateacc <= pwmrateacc + pwmratelatch;
			oldpwmratemsb <= pwmratemsb;
			if oldpwmratemsb /= pwmratemsb then
				count <= count + 1;
			end if;			-- old /= new


			if oldpdmratelatchloadreq = '1' and olderpdmratelatchloadreq = '1' then
				pdmratelatch <= prepdmratelatch;
			end if;
			oldpdmratelatchloadreq     <= pdmratelatchloadreq;
			olderpdmratelatchloadreq  <= oldpdmratelatchloadreq;
			pdmrateacc <= pdmrateacc + pdmratelatch;
			oldpdmratemsb <= pdmratemsb;
			if oldpdmratemsb /= pdmratemsb then
				prate <= '1';
			else
				prate <= '0';
			end if;			-- old /= new

		end if;  -- hclk

		if rising_edge(clk) then       -- 33/48/50 Mhz local bus clock
			if pwmrateload = '1' then
			prepwmratelatch <= ibus;
				pwmratelatchloadreq        <= '1';
			end if;
			if pdmrateload = '1' then
			prepdmratelatch <= ibus;
				pdmratelatchloadreq        <= '1';
			end if;
		end if;  -- clk

		if olderpwmratelatchloadreq = '1' then          -- asyncronous request clear
			pwmratelatchloadreq        <= '0';
		end if;
		if olderpdmratelatchloadreq = '1' then          -- asyncronous request clear
			pdmratelatchloadreq        <= '0';
		end if;
		refcount <= count;
		pdmrate <= prate;
	end process;
end behavioral;
```