IMPLEMENTASI KONTROLER *PROPORSIONAL INTEGRAL DERIVATIF* BERBASIS *FIELD PROGRAMMABLE GATE ARRAY* UNTUK PENGONTROL KECEPATAN MOTOR DC

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI TEKNIK ELEKTRONIKA

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh:

APRILIANAWATI SUTARTO

NIM. 115060300111030-63

KEMENTERIAN RISET DAN PENDIDIKAN TINGGI

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

MALANG

2015

LEMBAR PERSETUJUAN

IMPLEMENTASI KONTROLER *PROPORSIONAL INTEGRAL DERIVATIF* BERBASIS *FIELD PROGRAMMABLE GATE ARRAY* UNTUK PENGONTROL KECEPATAN MOTOR DC

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI TEKNIK ELEKTRONIKA

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

APRILIANAWATI SUTARTO
NIM. 115060300111030-63

Skripsi ini telah direvisi dan disetujui oleh dosen pembimbing
pada tanggal 14 Agustus 2015

Dosen Pembimbing I	Dosen Pembimbing II
Ir. Nanang Sulistyanto, MT.	**Mochammad Rif'an, ST., MT**
NIP. 19700113 199403 1 002	**NIP. 1971031 200012 1 001**

LEMBAR PENGESAHAN

IMPLEMENTASI KONTROLER *PROPORSIONAL INTEGRAL DERIVATIF* BERBASIS *FIELD PROGRAMMABLE GATE ARRAY* UNTUK PENGONTROL KECEPATAN MOTOR DC

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI TEKNIK ELEKTRONIKA

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

APRILIANAWATI SUTARTO
NIM. 115060300111030-63

Skripsi ini telah diuji dan dinyatakan lulus sidang skripsi
pada tanggal 30 Juni 2015

Dosen Penguji I	Dosen Penguji II
Akhmad Zainuri, ST., MT	**Eka Maulana, ST., MT., M.Eng**
NIP. 19840120 201212 1 003	**NIK. 841130 06 1 1 0280**

Dosen Penguji III

Dr. Eng, Panca Mudjirahardjo, ST., MT
NIP. 19700329 200012 1 001

Mengetahui,
Ketua Jurusan Teknik Elektro

M. Aziz Muslim ST., MT., Ph.D
NIP. 19741203 200012 1 001

RINGKASAN

Aprilianawati Sutarto, Jurusan Teknik Elektro Fakultas Teknik Universitas Brawijaya, Juni 2015, *Implementasi Kontroler Proporsional Integral Derivatif Berbasis Field Programmable Gate Array untuk Pengontrol Kecepatan Motor DC*, Dosen Pembimbing: Nanang Sulistiyanto dan Mochammad Rif'an.

Kontroler Proporsional-Integral-Derivatif (PID) adalah salah satu jenis algoritma kontroler yang paling sering digunakan. Algoritma ini telah diimplementasikan dalam berbagai sistem baik sebagai kontroler yang dapat berdiri sendiri, maupun sebagai bagian dari sistem bertingkat DDC (Direct Digital Control) dalam suatu proses sistem kontrol. Belakangan ini, *Field-Programable Gate Array* (FPGA), telah menjadi salah satu solusi alternatif untuk merealisasikan rangkaian kontroler digital, yang sebelumnya didominasi oleh sistem *general-purpose* mikroprosesor.

Penelitian ini membahas tentang kontroler PID yang diimplementasikan ke dalam sebuah Modul FPGA Nexys 2 yang dioperasikan dalam frequensi 50 Mhz. Kontroler PID berbasis FPGA ini akan digunakan dalam sebuah sistem untuk mengatur kecepatan putaran motor DC. Metode yang digunakan dalam penelitian ini adalah dengan membuat sebuah rangkain PID, PWM, pembaca sensor, dan *interface keyboard* menggunakan kombinasi gerbang logika dan pengaturan waktu yang pada akhirnya akan diimplementasikan dalam FPGA. Penelitian ini menunjukkan bahwa dengan menggunakan frekuensi 50 MHz, kontroler PID dapat mengeluarkan hasil perhitungan dengan sangat cepat sehingga dibutuhkan sebuah penahan agar rangkaian ini dapat bekerja sesuai dengan waktu sampling yang telah ditentukan.

Untuk mengatur kecepatan motor DC, rangkaian PWM 16 bit dalam penelitian ini memiliki frekuensi sinyal keluaran 762,941 Hz dengan *duty-cycle* yang dapat diatur dari 0% sampai 100%. Selain itu, rangkaian pembaca sensor 16 bit dalam penelitian ini dapat membaca sinyal masukan dengan lebar frekuensi antara 2 Hz-65 kHz dengan sampling PID satu detik. Saat sinyal dari sensor kurang dari itu, sinyal tidak dapat terbaca, dan bila lebih dari itu, sinyal yang terbaca tidak sesuai dengan nilai yang sebenarnya karena sudah melebihi kapasitas.

Kata Kunci: Kontroler PID, FPGA, Pengontrol kecepatan motor DC.

SUMMARY

Aprilianawati Sutarto, Department of Electrical Engineering, Faculty of Engineering, University of Brawijaya, June 2015, *Implementation of FPGA-based PID Controller for Speed Control of DC Motors*, Academic Supervisor: Nanang Sulistiyanto and Mochammad Rif'an.

The PID controller is by far the most common control algorithm. This algorithm has been implemented in many different forms, as a stand-alone controller or as a part of a DDC (Direct Digital Control) package in a hierarchical distributed process control system. Recently, field-programable gate arrays (FPGAs) have become an alternative solution for the realization of digital control system, which were previously dominated by general-purpose microprocessor systems.

This research discusses a PID controller that implemented into a Nexys 2 FPGA Module that operates at frequency of 50 Mhz. This FPGA-based PID controller is used in a system to control the speed of DC motors. The method used in this research is to create a circuit of a PID controller, PWM generator, sensor reader and *keyboard* interface using a combination of logic gates and timing that will eventually be implemented in FPGA. This research shows that by using a frequency of 50 MHz, the PID controller can calculate so quickly that it needs a holder so that this circuit can work in accordance with a predetermined sampling time.

To control the speed of DC motors, the 16-bit PWM generator circuit in this research have an output signal with frequency of 762,941 Hz and a *duty-cycle* that can be set from 0% to 100%. The 16-bit sensor reader's circuit in this research can read the signal input with frequency between 2 Hz-65 kHz with the sampling of PID controller, one second. If the input frequency is less than that, the signal is not readable, and when more than that, the signal's reading does not correspond to the actual values because it has already exceeds the capacity.

Keyword: PID controller, FPGA, speed control of DC motors.

PENGANTAR

Alhamdulillah, puji dan syukur penulis panjatkan kepada Allah SWT, karena atas segala petunjuk dan nikmat-Nya lah skripsi ini dapat diselesaikan. Skripsi dengan judul "Implementasi Kontroler *Proporsional Integral Derivatif* Berbasis *Filed Programmable Gate Array* untuk Pengontrol Kecepatan Motor DC" ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Teknik di Jurusan Teknik Elektro Universitas Brawijaya. Penulis menyadari bahwa dalam penyusunan skripsi ini tidak terlepas dari bantuan berbagai pihak. Oleh karena itu, dengan ketulusan dan kerendahan hati penulis menyampaikan terima kasih kepada:

- Ayah, Ibu, adik, dan saudara penulis atas segala macam dukungan yang telah diberikan kepada penulis hingga terselesaikannya skripsi ini.
- Bapak M. Aziz Muslim, ST., MT., Ph.D. selaku Ketua Jurusan Teknik Elektro Universitas Brawijaya dan Bapak Hadi Suyono, ST., MT., Ph.D. selaku Sekretaris Jurusan Teknik Elektro Universitas Brawijaya.
- Ibu Erni Yudaningtyas, Ir., MT. selaku dosen pembimbing akademik atas segala bimbingan, nasehat dan motivasi yang telah diberikan.
- Bu Nurrusa'adah, Ir., MT. selaku Dosen Pembimbing Paket B dan kedua dosen Pembimbing skripsi ini Bapak Ir. Nanang Sulistiyanto, MT. dan Pak Mochammad Rif'an, ST., MT. atas segala bimbingan, kritik, dan saran yang telah diberikan.
- Bapak Ahmad Dulhadi, ST. selaku pranata di Laboratorium Sistem Digital, yang sudah membantu dalam hal peminjaman alat Laboratorium utuk skripsi ini.
- Seluruh dosen dan karyawan Teknik Elektro Universitas Brawijaya, yang telah memberikan banyak ilmu dan pelajaran berharga selama penulis menempuh ilmu di Teknik Elektro Universitas Brawijaya.
- Nurotul Auliya' dan Rizal Wahyudi yang merupakan teman seperjuangan dalam menyelesaikan proyek yang menjadi dasar penulisan skripsi ini.
- Sahabat penulis, Fadila N. Eritha dan Nurotul Auliya' yang tiada henti memberikan semangat dan dorongan agar skripsi ini dapat cepat selesai.
- Seluruh teman-teman asisten Laboratorium Sistem Digital mulai angkatan 2009 hingga 2013, yang telah berbagi suka duka bersama selama menuntut ilmu di Laboratorium.

- Semua teman-teman Inverter 2011, terutama paket B yang telah berbagi suka dan duka selama empat tahun ini mulai dari probin maba yang tak terlupakan hingga penulisan skripsi sekarang ini.
- Dan semua orang yang telah membantu dan tidak bisa penulis sebutkan satu persatu, terimakasih banyak atas semua bantuannya.

Pada akhirnya, penulis menyadari bahwa skripsi ini masih belum sempurna. Oleh karena itu, penulis sangat mengharapkan kritik dan saran yang membangun. Penulis berharap semoga skripsi ini dapat bermanfaat bagi pengembangan ilmu pengetahuan dan teknologi serta bagi masyarakat.

Malang, 11 Juni 2015

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Kontroler Proporsional-Integral-Derivatif (PID) adalah salah satu jenis algoritma kontroler yang paling sering digunakan. Sebagian besar sistem yang menggunakan umpan balik di dalamnya, telah menerapkan algoritma ini maupun variasinya. Algoritma ini telah diimplementasikan dalam berbagai sistem baik sebagai kontroler yang dapat berdiri sendiri, maupun sebagai bagian dari sistem bertingkat DDC (Direct Digital Control) dalam suatu proses sistem kontrol. Algoritma kontroler PID dapat dilihat dari berbagai sisi, baik itu sebagai sebuah alat yang dapat dioperasikan secara mandiri maupun secara analisis (Astrom & Hagglund, 1995).

Dewasa ini, lebih dari 95% proses kontrol pada industri sudah menggunakan kontroler PID. Kontroler ini juga telah diimplementasikan dalam bentuk IC sebagai kontroler yang didesain dengan satu tujuan dalam penggunaanya. Kontroler PID juga sering dikombinasikan dengan gerbang logika, fungsi sekuensial, rangkaian pemilih, maupun blok sistem sederhana. Penggunaan kontroler PID sangat luas, mulai dari sistem otomatis yang digunakan dalam sistem pembangkit energi, transporasi, maupun dalam industri skala besar (Åström, 2002).

Implementasi kontroler PID telah mengalami beberapa tahap evolusi. Pada awal penggunaanya di tahun 1911 sebagai alat kemudi kapal, kontroler PID diimplementasikan menggunakan rangkaian mekanik dan pneumatik (Bennett, 1996). Setelah dari rangkaian mekanik dan pneumatik, rangkaian PID diimplementasikan dengan mikroprosesor berbahan tabung elektron, transistor, sampai rangkaian terintegrasi. Namun, yang paling berpangaruh dalam perkembangan kontroler PID adalah sistem berbasis mikroprosesor. Kontroler PID modern pada masa kini, hampir semuanya terbuat dari sistem berbasis mikroprosesor. Hal itu dikarenakan, kontroler PID berbasis mikroprosesor dapat dikembangkan sehingga dapat memiliki fitur tambahan seperti tuning otomatis, *gain scheduling*, dan adaptasi terus menerus secara *realtime* (Åström, 2002).

Salah satu bentuk dari perkembangan kontroler PID adalah kontroler berbasis digital. Kontroler digital menggunakan algoritma PID dan menerapkannya ke dalam sistem seperti mikrokontroler. Kontroler digital memiliki beberapa keunggulan bila dibandingkan dengan kontroler analog yang menggunakan komponen elektris. Beberapa kelebihannya antara lain lebih mudah untuk diintegrasikan dengan sistem lainnya sehingga memudahkan untuk

membentuk suatu jaringan kontrol yang lebih rumit. Selain itu, tingkat akurasi kontroler digital tidak bergantung pada komponen elektronik yang digunakan, berbeda dengan kontroler analog yang harus menggunakan nilai komponen yang presisi.

Belakangan ini, *Field-Programable Gate Array* (FPGA), telah menjadi salah satu solusi alternatif untuk merealisasikan rangkaian kontroler digital, yang sebelumnya didominasi oleh sistem *general-purpose* mikroprosesor (Chan, Moallem, & Wang, 2007). Kontroler berbasis FPGA menawarkan beberapa kelebihan dibandingkan dengan sistem berbasis mikroprosesor yang lain. FPGA memiliki kecepatan yang cukup tinggi diantara sistem lainnya yang nilainya dapat mencapai puluhan MHz. FPGA juga memiliki memory internal yang lebih besar, mampu digunakan untuk merealisasikan fungsi yang kompleks, memiliki konsumsi daya yang lebih rendah, serta lebih fleksibel. FPGA sudah digunakan dalam beberapa aplikasi seperti *pulse width modulation* (PWM) inverter, AC/DC konverter, DC-DC konverter, dan *analog to digital converter* (ADC). Di dalam FPGA juga terdapat kombinasi gerbang logika yang sudah tersusun dan siap digunakan yang biasa disebut *Configurable Logic Block* (CLB) yang dapat memudahkan pengguna untuk membentuk suatu fungsi yang rumit. Selain itu, FPGA lebih mudah untuk diintegrasikan dengan berbagai perangkat lain seperti *keyboard* dan monitor.

Mempertimbangkan beberapa hal di atas, FPGA dapat menjadi pilihan yang tepat untuk merealisasikan sebuah kontroler PID. Hal itu dikarenakan FPGA memiliki kecepatan yang lebih tinggi, *memory internal* yang lebih besar, serta lebih fleksibel daripada mikroprosesor pada umumnya. Selain itu, FPGA juga memiliki juga memiliki berbagai fitur yang dapat mempermudah perancangan suatu sistem dan akan lebih mudah untuk diintegrasikan dengan *keyboard* dan monitor. Dari beberapa hal tersebut, FPGA diharapkan dapat digunakan untuk melakukan perhitungan algoritma yang lebih rumit, namun dengan waktu yang masih lebih cepat daripada mikrokontroler pada umumnya.

Agar kontroler yang telah dirancang dapat dinyatakan baik, maka dibutuhkan sebuah sistem yang dapat menunjukkan performasi dari kontroler tersebut. Sistem pengaturan kecepatan motor DC dipilih, karena aplikasinya dalam dunia industri sangat luas. Sistem ini juga cukup sederhana, dan merupakan sebuah sistem linear yang cukup mudah untuk diaplikasikan. Selain itu, pengatur kecepatan motor DC juga dapat dijadikan landasan bagi sistem yang jauh lebih rumit seperti penggerak konveyor, penggerak lengan robot, dan sebagainya. Sistem kendali dengan motor DC dikendalikan oleh sinyal PWM dan dibutuhkan sebuah sensor untuk membaca keluarannya agar bisa dijadikan *feedback*.

Berdasarkan uraian di atas, penulis memutuskan untuk membuat sebuah rancangan kontroler PID berbasis FPGA dengan masukan dari perangkat berupa *keyboard* dan sistem monitoring yang dapat menampilkan sinyal pengontrolan secara *real time*. Perancangan ini juga merupakan sebuah proyek dari divisi FPGA, Laboratotium Sistem Digital Teknik Elektro Universitas Brawijaya (TEUB). Proyek ini dikerjakan oleh beberapa orang. Fokus dari perancangan yang akan dibahas dalam penulisan ini adalah kontroler PID berbasis FPGA dan aplikasinya pada pengontrolan kecepatan motor dengan judul "Implementasi Kontroler *Proporsional Integral Derivatif* Berbasis *Field Programmable Gate Array* untuk Pengontrol Kecepatan Motor DC".

1.2 Rumusan Masalah

Berdasarkan latar belakang yang menjadi dasar dari penulisan skripsi ini, permasalahan yang akan dibahas, dapat dirumuskan ke dalam poin-poin sebagai berikut:

1. Bagaimanakah bentuk implementasi kontroler PID dalam FPGA dan berapa banyak sumber daya yang dibutuhkan?
2. Bagaimanakah bentuk implementasi dari PWM generator 16 bit yang memiliki *duty-cycle* 0% sampai 100%?
3. Bagaimana bentuk implementasi dari pembaca sensor 16 bit yang dapat membaca kecepatan putaran motor dari 0-2500 rpm?
4. Bagaimanakah bentuk implementasi dari sistem *interface* untuk menghubungkan kontroler PID dengan sistem monitoring dalam FPGA?

1.3 Tujuan

Berdasarkan rumusan masalah yang telah dirumuskan sebelumnya, tujuan dari penelitian ini adalah:

1. Membuat kontroler PID yang diimplementasikan dalam FPGA.
2. Membuat PWM generator 16 bit dengan *duty-cycle* 0%-100% yang diimplementasikan ke dalam FPGA.
3. Membuat rangkaian pembaca sensor 16 bit yang dapat membaca kecepatan putaran motor dari 0-2500 rpm lalu diimplementasikan ke dalam FPGA.
4. Membuat interface untuk menghubungkan kontroler PID dengan *keyboard* dan sistem monitoring yang diimplementasikan ke dalam FPGA.

1.4 Batasan Masalah

Akibat banyaknya kemungkinan yang akan terjadi dalam perancangan ini, penulis membatasi masalah yang akan dibahas dalam perancangan ini meliputi:

1. Sistem monitoring berupa tampilan ke monitor VGA dan *driver keyboard*, beserta penentuan nilai parameter Kp, Ki, dan Kd untuk kontroler PID terdapat pada penelitian lain.
2. Pengontrolan yang dilakukan adalah untuk kecepatan motor DC saja.
3. Arah putaran motor dalam penelitian ini hanya dibuat satu arah.
4. FPGA yang digunakan dalam penelitian ini adalah *board_kit Nexys 2 Spartan 3E*.
5. Sensor yang digunakan pada penelitian ini memiliki range pengukuran 0-2500 rpm dengan resolusi 10°.

BAB 2
TINJAUAN PUSTAKA

2.1 Pengenalan FPGA

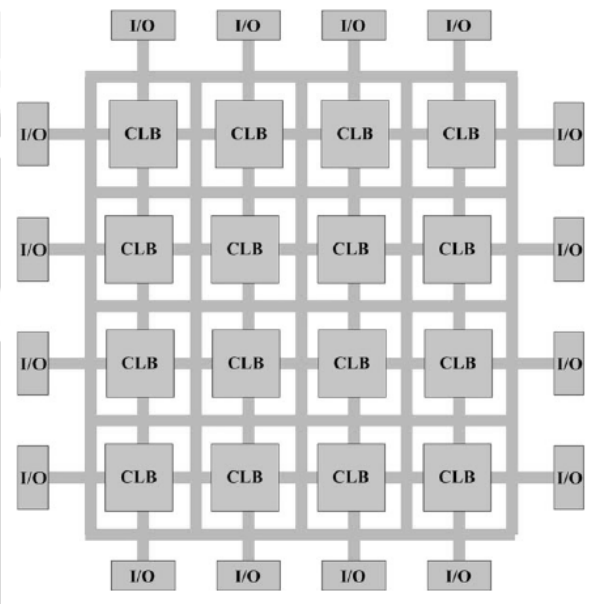
Field-Programmable Gate Array (FPGA) adalah sebuah perangkat pra-fabrikasi yang terbuat dari *silicon*, yang dapat diprogram secara elektrik untuk menjadi hampir semua jenis rangkaian atau sistem digital. Pada umumnya, FPGA terdiri dari tiga bagian utama, *programmable logic block*, yang dapat digunakan sebagai implementasi fungsi logika, *programmable routing*, yang digunakan untuk menghubungkan fungsi-fungsi logika, dan yang ketiga adalah I/O blok yang terhubung pada *logic block* melalui melalui rute interkoneksi dan membuat koneksi *off-chip*.

Salah satu contoh umum dari FPGA ditunjukkan pada Gambar 2.1. Pada gambar itu, *configurable logic block* (CLB) disusun dalam sebuah grid dua dimensi dan saling berhubungan satu sama lain melalui *programmable routing resources.* Blok I/O disusun di bagian pinggir dari grid dan juga terhubung melalui *programmable routing interconnect.* Istilah "programmable" yang terdapat dalam FPGA menunjukkan kemampuan FPGA untuk membuat konfigurasi baru pada *chip* bahkan setelah proses fabrikasi selesai. Rekonfigurabilitas/programabilitas dari FPGA didasarkan pada teknologi programing yang dapat menyebabkan perubahan perilaku *chip* pra-fabrikasi setelah mengalami fabrikasi.

Di dalam FPGA terdapat ribuan gerbang logika yang dapat disusun sesuai dengan keinginan perancangnya. Beberapa dari gerbang logika itu digabungkan untuk menjadi CLB yang berfungsi untuk menyederhanakan rangkaian dengan tingkat kerumitan yang lebih tinggi. Interkoneksi dari masing-masing gerbang logika maupun CLB dapat ditentukan melalui software yang akan melakukan konfigurasi menggunakan SRAM dan ROM. Dalam membuat desain sebuah sistem menggunkan FPGA, terdapat dua macam pemodelan yang paling sering digunakan yaitu model *Moore*, dan *Mealy*. Adanya pemodelan ini bertujuan untuk menjabarkan sebuah proses sekuensial ke dalam implementasi FPGA. (Farooq, Marrakchi, & Mehrez, 2012)

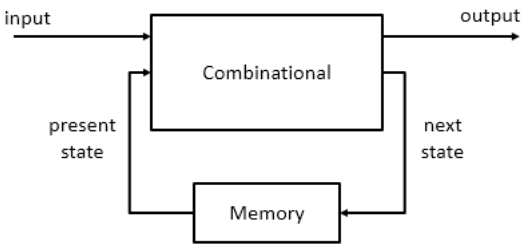
Dari beberapa metode yang dapat digunakan untuk melakukan pengkodean dalam bahasa mesin, bila kita melakukan pengkodean tersebut menggunakan sebuah pemodelan tertentu, maka hal ini dapat memastikan FSM (*Finite State Machine*) yang kita gunakan dapat mengenali secara pasti algoritma yang diberikan. Bahkan hal itu

dapat memberikan kemungkinan untuk memoptimalkan hasil simulasi timing diagram maupun mendebug rangkaian. Pemilihan jenis pemodelan yang digunakan, ditentukan oleh jenis arsitektur dan ukuran dari perangkat yang ingin kita buat. Secara umum, pemodelan Moore lebih baik bila diimplementasikan dalam FPGA sedangkan pemodelan Mealy lebih baik untuk perangkat berbasis CPLD (Complex Programmable Logic Device). (Xilinx, 2013)

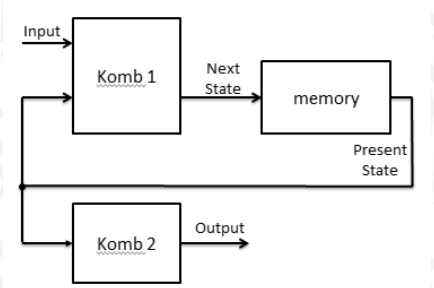


Gambar 2.1 Gambaran umum FPGA.
Sumber: (Farooq, Marrakchi, & Mehrez, 2012)

Perbedaan mendasar dari pemodelan Moore dan pemodelan Melay terdapat pada cara mendefinisikan keluaran. Dalam pemodelan Mealy seperti yang ditunjukkan dalam Gambar 2.2, nilai keluaran ditentukan dari nilai *present state* dan nilai masukan pada saat itu. Sedangkann dalam pemodelan Moore seperti yang ditunjukkan oleh Gambar 2.3, nilai keluaran sistem hanya akan ditentukan oleh nilai *present state* pada saat itu. Secara umum, pemodelan Moore lebih cocok untuk diimplementasikan ke dalam FPGA karena metode penulisan yang sering digunakan adalah dengan meneruskan langsung nilai dari *present state* tersebut kedalam keluaran sehingga tidak membutuhkan gerbang logika tambahan. Apabila terdapat rangkaian tertentu yang digunakan pada keluaran, sebagian besar memang lebih mudah bila digunakan pemodelan Mealy. Namun, pernyataan tersebut tidaklah selalu benar dan harus dilihat dulu kondisi dari keadaan yang kita inginkan. (Xilinx, 2013)



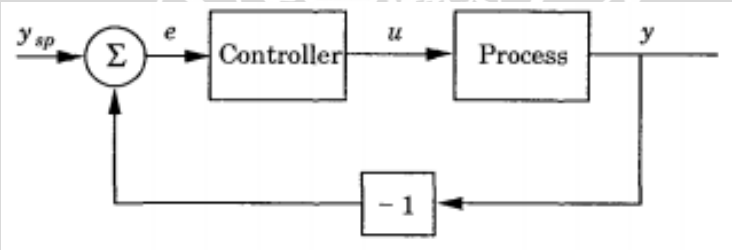
Gambar 2.2 Pemodelan Mealy



Gambar 2.3 Pemodelan Moore

2.2 Kontroler PID

Kontroler PID merupakan jenis kontroler dengan *feedback* di dalamnya. Dengan adanya *feedback*, sebuah sistem dapat menaikkan nilainya saat nilai yang dihasilkan lebih rendah dari *setpoint* dan menurunkan nilainya saat nilai yang dihasilkan lebih besar dari *setpoint*. *Feedback* semacam ini dikenal sebagai *feedback* negatif. Prinsip kerja dari *feedback* dapat dilihat seperti pada Gambar 2.4. Pada blok diagram tersebut, tanda negatif yang ada pada blok menunjukkan bahwa *feedback* yang diberikan bernilai negatif. Sistem dengan *feedback* menjadi sangat disukai karena sistem ini menghasilkan nilai keluaran yang dekat nilainya dengan *setpoint* meskipun diberikan gangguan tertentu. (Astrom & Hagglund, 1995)



Gambar 2.4 Blok diagram dari sistem kontrol dengan *feedback*
Sumber: (Astrom & Hagglund, 1995)

Kontroler PID memiliki struktur yang lebih kompleks bila dibandingkan dengan kontroler proposional. Di dalam kontroler PID terdapat gabungan dari kontroler proporsional, integral, dan derivatif. Hasil akhir dari kontroler PID merupakan penjumlahan total dari komponen proporsional, integral, dan derivatifnya. Dalam penerapannya dalam bentuk digital, kontroler PID mengalami perubahan dari sistem kontinyu menjadi sistem diskrit.

Implementasi diskrit dari komponen proporsional memiliki bentuk identik dengan bentuk kontinyunya. Apabila sistem kontinyu dari kontroler Proporsional dapat ditunjukkan dalam Persamaan (2-1), maka bentuk diskritnya dapat dilihat pada

Persamaan (2-2). Sinyal eror pada kedua persamaan tersebut adalah sinyal seperti yang diperlihatkan pada Gambar 2.4 (Franklin, Powell, & Workman, 1990)

$$u_p(t) = K_p e(t) \quad (2\text{-}1)$$

$$u_p(k) = K_p e(k) \quad (2\text{-}2)$$

Pada komponen derivatif, persamaan sistem dalam bentuk kontinyu dapat dilihat pada Persamaan (2-3), dengan T_D adalah waktu derivatif. Tujuan dari komponen ini adalah untuk mengerem penguatan sehingga *overshoot* yang dihasilkan tidak terlalu besar dan juga meningkatkan stabilitas sistem. Persamaan diferensial dalam persamaan ini dapat diselesaikan dengan melakukan pendekatan fungsi diferensial orde pertama dalam bentuk diskrit. Oleh karena itu, persamaan diskrit dari kontroler derivatif dapat dituliskan seperti pada Persamaan (4-2), dengan *T* adalah waktu sampling dari kontroler PID. (Franklin, Powell, & Workman, 1990)

$$u_d(t) = K_p T_D e(t) \quad (2\text{-}3)$$

$$u_d(k) = K_p T_D \frac{(e(k) - e(k-1))}{T} \quad (2\text{-}4)$$

Persamaan (2-5) menunjukkkan persamaan kontinyu dari komponen integral. Tujuan dari kontroler integral adalah untuk mengurangi nilai *eror steady state* tetapi dengan mengorbankan sedikit stabilitas sistem. Untuk mendapatkan nilai kontrol, nilai eror akan diintegralkan terhadap waktu. T_I dalam persamaan tersebut disebut sebagai waktu integral atau waktu reset. Persamaan ekuivalen dalam bentuk diskrit dari persamaan tersebut dapat dilihat pada Persamaan (2-6), dengan *T* adalah waktu sampling dari kontroler PID, dan $u_i(k-1)$ adalah hasil perhitungan komponen integral sebelumnya. (Franklin, Powell, & Workman, 1990)

$$u_i(t) = \frac{K_p}{T_I} \int_{t_0}^{t} e(t)dt \quad (2\text{-}5)$$

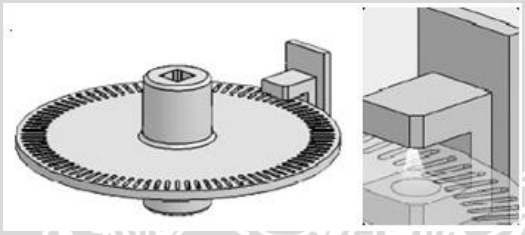
$$u_i(k) = u_i(k-1) + \frac{K_p T}{T_I} e(k) \quad (2\text{-}6)$$

Kontroler PID merupakan gabungan dari komponen proporsional, integral, dan derivatifnya. Dengan menjumlahkan Persamaan (2-2), Persamaan (2-4), dan Persamaan (2-6) maka akan kita dapatkan persamaan kontroler PID seperti yang ditunjukkan pada Persamaan (4-1). Pada persamaan ini, $ui(k-1)$ merupakan hasil kompoen integral pada perhitungan sebelumnya. (Franklin, Powell, & Workman, 1990)

$$u(k) = K_p e(k) + K_p T_D \frac{(e(k) - e(k-1))}{T} + u_i(k-1) + \frac{K_p T}{T_I} e(k) \quad (2\text{-}7)$$

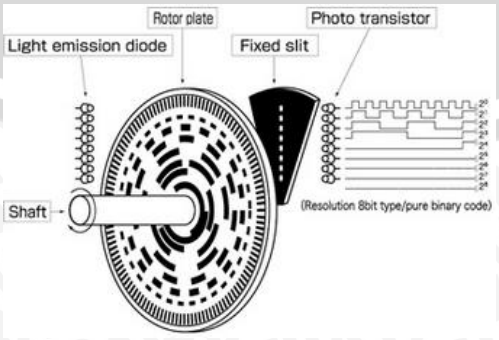
2.3 Rotary Enkoder

Rotary Enkoder adalah alat yang dapat menghasilkan sebuah sinyal digital sebagai akibat dari pergesseran sudut atau linear. Enkoder posisi dapat dikelompokkan dalam dua kategori yaitu enkoder inkremental yang mendeteksi perubahan pergeseran dari beberpa posisi data, dan enkoder absolut yang memberikan posisi aktual. Gambar 2.5 menunjukkan bentuk dasar dari sebuah enkoder inkremental. Enkoder ini terdiri dari sebuah piringan yang berputar dan sebuah pendeteksi. Pirigan yang berputar itu memiliki sejumlah lubang yang dapat dilewati cahaya sehingga cahaya dapat dideteksi oleh sensor yang ada pada sisi lain dari lubang. Saat sensor berputar, sensor akan menghasilkan sebuah pulsa dimana jumlah pulsa berbanding lurus dengan sudut yang dilewati oleh putaran piringan. Pada enkoder jenis ini, banyaknya pulsa yang terhitung akan selalu sama meskipun posisi awal dari piringan yang berputar berbeda.



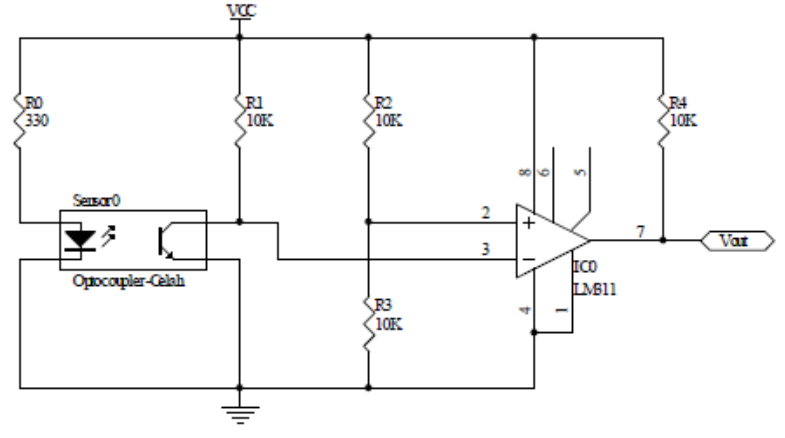
Gambar 2.5 Enkoder inkremental optic
Sumber: (Bolton, 2006)

Absolut Enkoder memiliki keluaran berupa sinyal yang terbaca sebagai kombinasi bilangan biner yang banyak bitnya tergantung dari disk yang digunakan. Satu kombinasi biangan biner akan mewakili sebuah sudut tertentu. Gambar 2.6 menunjukkan bentuk dasar sebuah absolut enkoder. Piringan pada absolut enkoder tersebut disusun berdasarkan kode *Gray* dengan nilai MSB ada di bagian paling dalam. Banyaknya sensor dan lampu LED yang ada pada sensor ini ditentukan oleh banyaknya lingkaran yang ada pada disk. (Bolton, 2006)



Gambar 2.6 Enkoder absolut
Sumber: (Bolton, 2006)

Sensor kecepatan yang digunakan untuk mengukur kecepatan motor DC dalam penelitian ini adalah rotary enkoder produksi Depok Instrumen dengan tipe *DI-REV1 DI-Rotary Enkoder Versi 1*. Rotary enkoder ini terdiri dari sebuah enkoder optis, sebuah sensor posisi optis, dan rangkaian pengkondisi sinyal. Sensor posisi optis yang digunakan dalam perancangan ini adalah *octocoupler* tipe celah sebagai pembaca perubahan posisi lubang. Hasil keluaran sinyal *octocoupler* dimasukkan ke dalam rangkaian pengkondisi sinyal agar sinyal yang dihasilkan menjadi lebih stabil dan tahan terhadap noise. Gambar 2.7 menunjukkan skema rangkaian dari sensor dan rangkaian pengkondisi sinyal dalam rotary enkoder.



Gambar 2.7 Skema rangkaian sensor dan pengkondisi sinyalnya
Sumber: Depok Instruments, 2011.

Rangkaian sensor pada Gambar 2.7 ini memiliki dua pin masukan dan satu pin keluaran, yaitu pin Vcc, Gnd, dan V_{out}. Tabel 2.1 menunjukkan konfigurasi pin dari rangkaian sensor tersebut sesuai dengan datasheetnya. Rangkaian ini pada dasarnya menggunakan sebuah komparator berbasis *op-amp* sebagai pembanding. Prinsip kerja rangkaian dimulai dengan sensor mendeteksi ada atau tidaknya penghalang dan menghasilkan sebuah tegangan keluaran yang dimasukkan pada pin 3 komparator. R_0 pada rangkaian berfungsi sebagai pembatas arus yang mengalir pada sensor. Sementara R_1, berfungsi sebagai rangkaian pembagi tegangan antara tegangan masukan Vcc dan sensor yang diteruskan pada pin 3. R_2 dan R_3 juga merupakan rangkaian pembagi tegangan yang membagi tegangan menjadi separuh tegangan Vcc dan menjadikannya referensi untuk komparator pada pin 2. Selanjutnya, R_4 akan menguatkan hasil komparasi sinyal sehingga sinyal keluaran yang dikeluarkan oleh rangkaian ini seluruhnya merupakan sinyal digital dengan nilai saat logika *high* adalah Vcc dan nilai saat logika *low* adalah Gnd.

Selain rangkaian sensor, sensor ini juga dilengkapi sebuah disk enkoder optis yang memiliki 36 lubang. Sudut antara dua lubang yang berdampingan dengan titik tengahnya adalah 10°. Selain itu, sensor ini beroperasi pada sumber tegangan $3{,}5 - 5{,}5V$. Keluaran dari sensor ini adalah sinyal. Sensor ini akan mengeluarkan logika *low* saat celah sensor terhalang dan logika *high* saat celah sensor tanpa halangan. Pada saat sensor ini mengeluarkan logika *low*, maka ia akan mengeluarkan tegangan dengan range $0 - 0{,}5V$, sedangkan bila ia mengeluarkan logika *high* maka ia akan mengeluarkan tegangan dengan range $3 - 5V$ atau dengan nilai maksimum sebesar $Vcc - 0{,}5V$. Sensor ini dapar membaca perputaran disk dengan frekuensi *toggle* maksimal 1500 Hz, dan kecaparan putaran 2500 rpm. (Depok-Instruments, 2011)

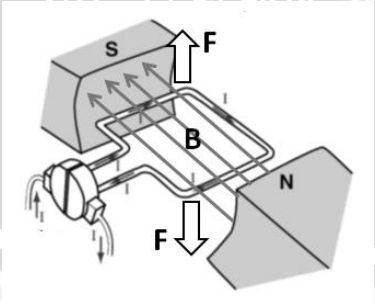
Tabel 2.1 Konfigurasi Pin rangkaian sensor

Nama PIN	Fungsi
GND	Sumber tegangan bawah/ negatif/ *ground*
VCC	Sumber tegangan atas / positif
V_{out}	Data keluaran rangkaian sensor

Sumber: Depok-Instruments, 2011

2.4 Motor DC

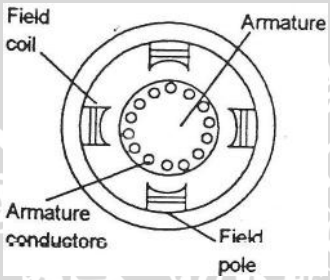
Gambar 2.8 menunjukkan prinsip kerja dasar dari motor DC. Pada dasarnya motor DC adalah sebuah kumparan yang bebas berputar dalam sebuah medan magnet. Ketika arus dialirkan pada kumparan, gaya yang dihasilkan akan membentuk sudut dengan medan magnet di tempat itu sehingga menimbulkan sebuah gerakan rotasi. Namun, agar gerakan rotasi itu dapat terus berlanjut, ketika kumparan berada dalam posisi vertikal, maka polaritas arus yang diberikan harus dibalik.



Gambar 2.8 Prinsip kerja motor DC
Sumber: (Bolton, 2003)

Pada motor DC konvensional, kumparan kawat pada motor DC akan diletakkan pada sebuah tempat berupa silinder magnetik yang disebut dengan *armature*. *Armature* ini dipasang pada sebuah tempat yang bebas berputar. Di sekitar *armature* dipasang

beberapa sumber medan magnet yang disebut *field poles*. Pada motor DC ukuran kecil, field *pole*s ini dapat berupa magnet permanen, atau magnet yang ditimbulkan karena adanya electromagnet dari arus yang mengalir pada *field coils*. Gambar 2.9 menunjukkan gambar sebuah motor DC dengan empat *pole* dan medan magnet yang diproduksi oleh arus *field coils*. Ujung-ujung dari tiap *armature* dihubungkan dengan sebuah cincin bernama komutator yang terhubung dengan konduktor karbon yang dikenal sebagai *brushes*. Ketika *armature* berputar, komutator akan membalikkan arus pada setiap koil sementara koil itu berputar melewati *pole*. Hal ini diperlukan agar gaya yang bekerja pada koil tetap mengarah pada arah yang sama sehingga putaran dapat terus berlanjut. Untuk membalikkan arah putaran motor, cukup dengan membalikkan polaritas dari arus yang mengalir pada *armature* atau membalikkan medan magnet. (Bolton, 2003)

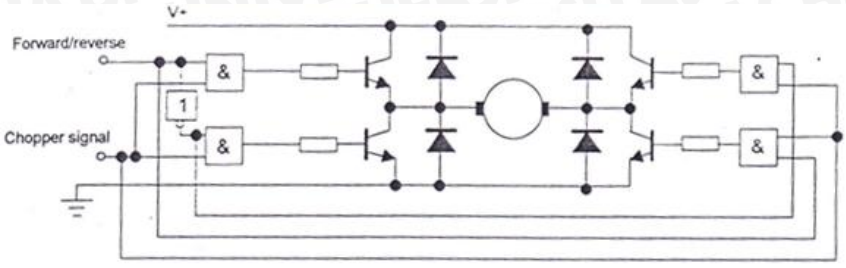


Gambar 2.9 Bagian motor DC
Sumber: (Bolton, 2003)

Kecepatan pada motor DC dengan magnet permanen, tergantung pada besarnya arus yang mengalir pada kumparan *armature*. Kecepatan putaran motor dapat diatur dengan merubah arus pada *armature* itu atau dengan merubah medan magnetik di dalamnya. Secara praktek, cara yang paling umum digyunakan untuk mengatur kecpatan putaran motor adalah dengan mengatur arus *armature*nya. Hal ini dapat dilakukan dengan mengontrol tegangan yang diberikan pada *armature*. Namun, karena biasanya motor DC dicatu oleh sumber tegangan tetap, maka perubahan nilai tegangan didapatkan dari rekayasa rangkaian elektrik.

Sebuah rangkaian seperti pada Gambar 2.10 yang dikenal dengan H-Bridge adalah salah satu rangkaian yang digunakan untuk mengatur besarnya tegangan yang diberikan pada *armature*. Sebuah teknik yang disebut dengan *pulse width modulation* (PWM), digunakan untuk mengatur besarnya tegangan masukan melalui mikroprosessor. Pada dasarnya, PWM berfungsi untuk mengaktifkan dan menonaktifkan transistor secara terus menerus dengan waktu sangat cepat sehingga tegangan rata-rata pada keluaran rangkaian menjadi dapat bervariasi. Dioda pada rangkaian berfungsi sebagai jalan bagi

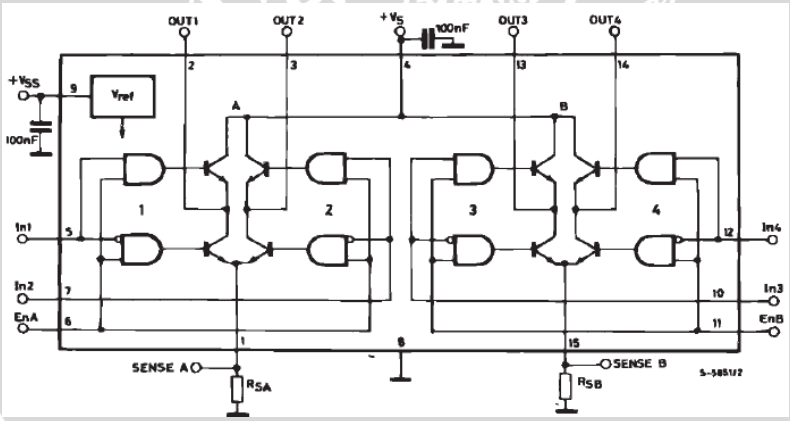
arus yang timbul saat transistor mati akibat motor yang tiba-tiba bersikap seolah generator. Pada rangkaian H-Bridge, arah putaran motor dapat diatur dengan memberikan logika tertentu pada transistor. (Bolton, 2003)



Gambar 2.10 Rangkaian H-Bridge
Sumber: (Bolton, 2003)

2.5 H-Bridge dengan IC L298

Driver motor yang paling sering digunakan untuk kendali motor DC adalah *H-bridge*. *Driver* ini cukup banyak ditemui di pasaran dan salah satu jenisnya adalah L298. L298 adalah sebuah *dual full-bridge* dalam bentuk IC dan memiliki range tegangan hingga 46 volt dengan sumber tegangan berbeda untuk gerbang logika di dalamnya. Blok diagram dari konfigurasi bagian dalam L298 dapat dilihat pada Gambar 2.11 dan spesifikasinya dapat dilihat pada Tabel 2.2.



Gambar 2.11 Blok diagram bagian dalam L298
Sumber: (STMicroelectronics, 2000)

Terdapat lima belas pin pada ic L298. Dua diantaranya adalah sumber tegangan untuk motor dan gerbang logika. Satu pin adalah ground, dan dua pin yang lain berfungsi untuk mengukur jumlah arus yang mengalir pada masing-masing motor. Selain itu, masing-masing motor juga memiliki dua pin keluaran untuk motor, satu pin untuk sinyal enable, dan dua pin untuk mengatur arah putaran motor. Konfigurasi sinyal untuk mengatur arah putaran motor dapat dilihat pada Tabel 2.3, sedangkan keterangan lengkap untuk masing-masing pin dapat dilihat pada Tabel 2.4.

Tabel 2.2 Rating nilai maksimum L298

Symbol	Parameter	Value	Unit
V_s	Power Supply	50	V
V_{ss}	Logic Supply Voltage	7	V
V_I, Ven	Input and Enable Voltage	-0.3 to 7	V
I_o	Peak Output Current (each channel)		
	- Non Repetitive (t=100µs)	3	A
	- Repetitive (80% on-20% off, t_{on} = 10ms)	2.5	A
	- DC Operating	2	A
V_{sens}	Sensing Voltage	-1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	-25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	-40 to 150	°C

Sumber: (STMicroelectronics, 2000)

Tabel 2.3 Konfigurasi sinyal untuk mengatur arah putaran motor

Inputs		Functions
V_{en} = H	C=H ; D=L	Forward
	C=L ; D=H	Reverse
	C=D	Fast Motor Stop
V_{en} = L	C=X ; D=X	Free Running Motor Stop

Sumber: (STMicroelectronics, 2000)

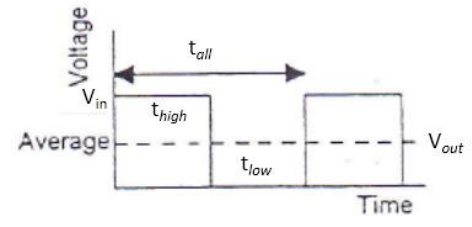
Tabel 2.4 Fungsi PIN pada L298

MW.15	PowerSO	Name	Function
1; 15	2; 19	Sense A; Sense B	Between this pis and groumd is connected the sense resistor to control the current of the load.
2; 3	4; 5	Out 1; Out2	Outputs of the Bridge A. The current that flows trough the load connected between these two pins is monitored at pin 1.
4	6	Vs	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5; 7	7; 9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6; 11	8; 14	Enable A; Enable B	TTL Compatible Enable Input; the L state disable the Bridge A (enable A) and/or the Bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Bloks. A 100nF capacitor must be connected between these pin and ground.
10; 12	13; 15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16; 17	Out 3; Out 4	Outputs of the Bridge B. The current that flows trough the load connected between these two pins is monitored at pin 15.
-	3; 18	N.C.	Not Conected.

Sumber: (STMicroelectronics, 2000)

2.6 Pulse Width Modulation (PWM)

Pulse Width Modulation (PWM) digunakan secara luas dalam sistem control sebagai sebuah cara untuk mengontrol nilai rata-rata dari tegangan DC. Oleh karena itu, saat sebuah nilai tegangan analog konstan dibagi menjadi pulsa dengan lebar pulsa yang beragam, tegangan rat-rata dari nilai itu jadi bisa ditentukan seperti ilustrasi pada Gambar 2.12. Pada gambar tersebut diperlihatkan bahwa nilai *duty-cycle* merupakan perbandingan dari lamanya waktu sinyal berlogika *high* dengan periode sinyal secara keseluruhan, atau dapat dituliskan dalam Persamaan (2-8).



Gambar 2.12 Ilustrasi tegangan rata-rata PWM
Sumber: (Bolton, 2003)

Berdasarkan *duty-cycle,* tegangan rata-rata dari sinyal PWM juga dapat dicari. Tegangan rata-rata keluaran didapatkan melalui perkalian antara *duty-cycle* dengan besarnya tegangan masukan. Dengan nilai *duty-cycle* antara 0% sampai 100%, maka persamaan untuk mencari tegangan keluaran rata-rata sinyal PWM dapat dilihat pada Persamaan (4-2). (Nouman, Klima, & Knobloch, 2013)

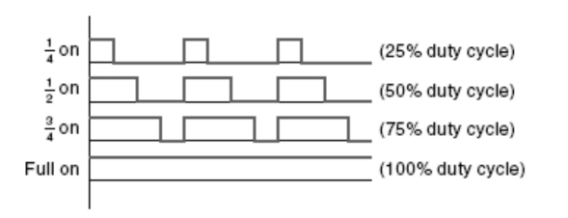
$$dutycycle = \frac{t_{high}}{t_{all}} \times 100\% \quad (2\text{-}8)$$

$$V_{out} = dutycycle \times V_{in} = \frac{t_{high}}{t_{all}} \times V_{in} \quad (2\text{-}9)$$

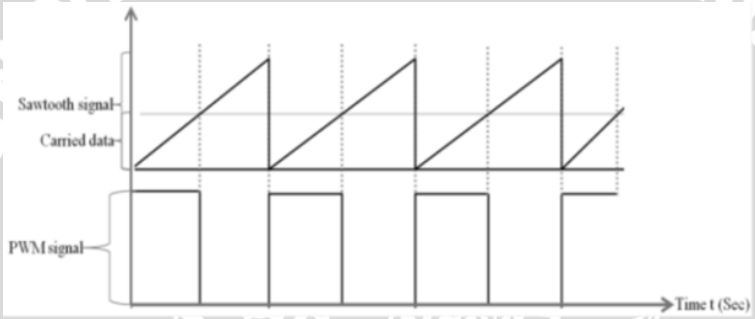
Istilah *duty-cycle* digunakan untuk menamakan bagian dari sinyal PWM yang berlogika *high*. Oleh karena itu, sebuah sinyal PWM dimana separuh dari sinyal itu bernilai tinggi sedangkan separuhnya lagi bernilai rendah, dikatakan memiliki *duty-cycle* $^1/_2$ atau 50%. Gambar 2.13 menunjukkkan sinyal PWM dengan beberapa macam *duty-cycle* yang berbeda. (Bolton, 2003)

Dalam sistem digital, sinyal PWM dapat dibangkitkan oleh sebuah mikroprosessor. Prinsip kerja dari sinyal PWM dapat dilihat seperti pada Gambar 2.14. Untuk membangkitkan sebuha sinyal PWM, dibutuhkan sebuah komparator yang membandingkan nilai dari dua buah masukan. Masukan yang pertama adalah sebuah sinyal gergaji dan yang kedua adalah sebuah masukan yang mewakili besarnya *duty-cycle* yang kita inginkan. Sinyal masukan yang digunakan dapat juga berupa sinyal

sinusoida. Apabila nilai sinyal kurang atau sama dengan masukan yang diinginkan, maka sinyal PWM akan berlogika *high*. Sebaliknya, saat nilai sinyal lebih besar dari masukan yang diinginkan, maka sinyal PWM akan berlogika *low*. (Nouman, Klima, & Knobloch, 2013)



Gambar 2.13 PWM dengan beberapa *duty-cycle*
Sumber: (Bolton, 2003)



Gambar 2.14 Prinsip kerja PWM
Sumber : (Nouman, Klima, & Knobloch, 2013)

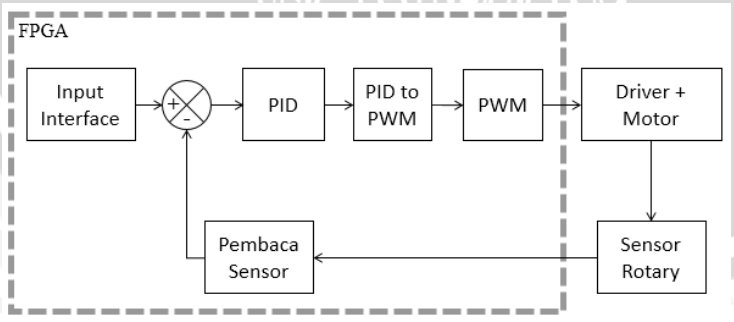
BAB 3
METODE PENELITIAN

Penyusunan skripsi ini didasarkan pada masalah yang bersifat aplikatif, yaitu perencanaan dan perealisasian kontroler PID agar dapat bekerja sesuai dengan rancangan yang mengacu pada rumusan dan tujuan masalah. Langkah-langkah yang dilakukan untuk merealisasikan alat yang dirancang adalah perancangan, simulasi dan pengujian, serta pengambilan kesimpulan.

3.1 Perancangan Alat

Untuk memudahkan perancangan sistem pengontrolan kecepatan motor DC, perancangan dapat dibagi menjadi perancangan perangkat keras dan perancangan rangkaian dalam Modul FPGA. Perancangan perangkat keras terdiri dari perancangan ramgkaian *driver* motor DC dan rangkaian sensor pembaca kecepatan. Sementara itu, perancangan rangkaian dalam Modul FPGA meliputi perancangan kontroler PID, *Input Interface*, Pembaca Sensor, dan PWM genetaror.

Masukan *keyboard* yang berupa kode ASCII akan diterjemahkan oleh *Input Interface*. Data kemudian akan diproses oleh kontroler PID. Dari kontroler PID, data akan dijadikan masukan bagi PWM generator yang pulsa keluarannya akan menentukan kecepatan putaran motor DC. Kecepatan putaran motor DC akan diukur oleh sensor kecepatan putaran berupa rotary enkoder, dan hasilnya akan dinaca oleh Pembaca Sensor. Diagram blok dari sistem pengontrolan kecepatan motor DC dapat dilihat pada Gambar 3.1.



Gambar 3.1 Blok diagram sistem pengontrolan kecepatan motor DC

3.1.1 Perancangan Perangkat Keras (*Hardware*)

Perancangan perangkat keras dalam sistem ini mengacu kepada perancangan sistem pendukung yang tidak diimplementasikan dalam FPGA. Dalam hal ini rangkaian itu adalah rangkaian *driver* motor DC serta perancangan antarmuka sensor *rotary enkoder* sebagai

feedback. Perancangan rangkaian akan didesain menggunakan perangkat lunak eagle. Rangkaian ini nantinya akan dihubungkan dengan pin dalam FPGA.

3.1.2 Perancangan Rangkaian dalam Modul FPGA

Perancangan perangkat lunak dalam sistem ini mengacu pada pemrograman FPGA dalam bahasa VHDL. Untuk mempermudah perancangan, perangkat lunak dalam sistem ini dibagi dalam beberapa blok yaitu *Input Interface*, Pembaca Sensor, kontroler PID, dan PWM generator. Rancangan program dibuat dalam bentuk *diagram Moore* untuk rangkaian sekuensial dan perancangan tabel kebenaran untuk rangkaian kombinasional. Bahasa pemrograman yang digunakan dalam perancangan ini adalah bahasa pemrograman VHDL dengan compiler yang digunakan ISE Design Suite dari Xilinx.

3.2 Pengujian Alat

Untuk menganalisis kinerja alat apakah sesuai dengan yang direncanakan maka dilakukan pengujian sistem. Pengujian dibagi menjadi pengujian *hardware*, rangkaian dalam Modul FPGA, dan pengujian secara keseluruhan. Pengujian dilakukan berdasarkan pada masing-masing blok yang telah ditentukan sebelumnya.

3.2.1 Pengujian Perangkat Keras (*hardware*)

Pengujian *hardware* dilakukan untuk memastikan alat yang digunakan bekerja dengan baik. Pengujian *hardware* terdiri dari pengujian rangkaian *driver* motor DC serta pengujian rangkaian rotary enkoder. Pengujian pada *driver* motor DC dilakukan dengan cara memeriksa besarnya tegangan yang keluar dari *driver*. Sdangkan pengujian rotary enkoder dilakukan untuk menentukan batas atas dan bawah dari kecepatan yang dapat diukur beserta bentuk sinyal yang dikeluarkan rangkaian.

3.2.2 Pengujian Rangkaian dalam Modul FPGA

Pengujian rangkaian dalam Modul FPGA dilakukan dengan dua cara. Cara pertama adalah dengan mensimulasikan program menggunakan ISIM Simulator. Pengujian kedua dilakukan dengan melakukan simulasi modul pada FPGA dan memastikan rangkaian berfungsi sesuai dengan yang diinginkan.

Pengujian rangkaian yang pertama adalah rangkaian *Input Interface*. Pengujian ini dilakukan untuk menentukan apakah nilai yang diinginkan sudah dapat masuk dengan benar. Pengujian implementasi pada modul ini dilakukan dengan cara menekan beberapa macam tombol pada *keyboard* dan melihat keluaran yang dihasilkan. Masukan yang mulanya berupa code ASCII harus sudah berubah menjadi code biner. Keluaran *Input Interface* dapat dilihat

pada ISIM Simulator maupun dengan mengimplementasikan program dan menampilkan keluarannya pada output berupa seven segment.

Pengujian kedua adalah pengujian rangkaian pembaca sensor. Pengujian ini bertujuan untuk memastikan *feedback* yang masuk ke dalam rangkaian kontroler adalah nilai yang tepat. Pengujian ini dilakukan dengan memanfaatkan sebuah *function generator* untuk memberikan pulsa masukan yang mewakili keluaran sensor. Hasil pembacaan sinyal akan ditampilkan dalam seven segment dan hasil pembacaan sepuluh detik pertama akan dicatat.

Pengujian ketiga adalah rangkaian PWM. PWM adalah bagian dari sistem yang merubah hasil perhitungan kontroler PID menjadi nilai yang akan diberikan pada motor DC. Pengujian PWM dilihat dengan menggunakan ISIM Simulator. *Duty-cycle* dari sinyal yang dikeluarkan oleh PWM generator akan dilihat apakah sudah sesuai denganmasukan yang diberikan. Untuk memastikan bahwa PWM sudah bekerja dengan baik, nilai masukan yang diberikan akan diubah-ubah secara berkala dan perubahan transisinya juga akan diamati. Pengujian PWM juga dilakukan dengan mengimplementasikan rangkaian PWM pada FPGA dan menampilkan sinyal keluaran PWM pada osiloskop digital. Parameter yang akan diukur dari sinyal keluaran adalah frekuensi, periode, dan *duty-cycle*.

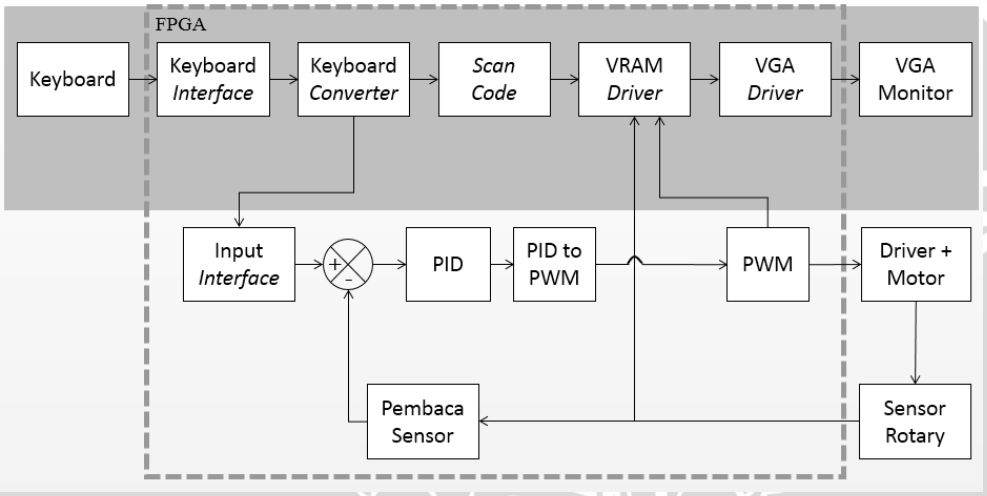
3.2.3 Pengujian Keseluruhan Sistem

Tujuan dari pengujian ini adalah untuk mengetahui kinerja sistem secara keseleruhan. Pengujian sistem dilakukan setelah seluruh program dalam FPGA diimplementasikan dan digabungkan dengan *hardware*. Parameter Kp, Ki, Kd akan dibuat tetap sementara nilai setpoint akan diubah-ubah. Keluaran sistem akan yang merupakan kecepatan putaran mototr yang ditampilkan dalam *seven segment display*, akan dicatat sampai kecepatannya stabil kemudian dibuat grafiknya. Sistem dinyatakan baik apabila hasil pembacaan sensor sudah sesuai dengan apa yang dimasukkan dalam *setpoint*.

BAB 4
PERANCANGAN

4.1 Sistem Keseluruhan

Penilitian ini pada dasarnya merupakan sebuah proyek gabungan yang dikerjakan oleh tiga orang. Proyek ini terdiri atas tiga bagian utama, yaitu kontroler PID, sistem autotuning untuk menemukan nilai parameter kontroler, dan sistem monitoring. Seluruh sistem akan diimplementasikan dalam modul FPGA dan digabung menjadi satu. Blok diagram dari sistem keseluruhan proyek ini dapat dilihat pada Gambar 4.1, sementara bagian yang dibahas dalam skripsi ini, adalah bagian yang diarsir dengan warna terang.



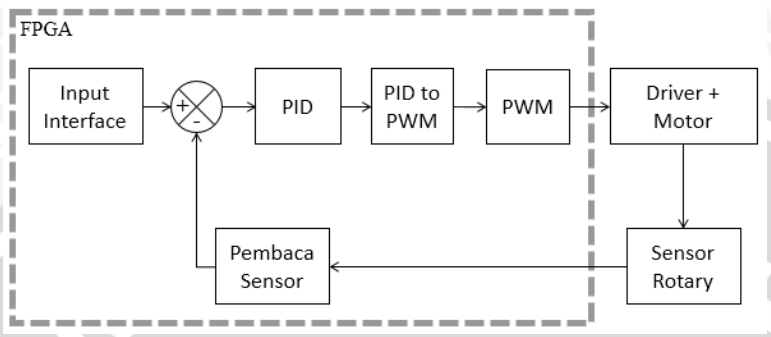
Gambar 4.1 Blok diagram keseluruhan sistem

Kontroler PID berbasis FPGA sebagai pengontrol kecepatan motor DC merupakan fokus dalam pen elitian ini. Kontroler ini mendapatkan masukan Kp, Ki, Kd dari sistem autotuning. Nilai Kp, Ki, Kd juga dapat dimasukkan secara manual bersamaan dengan *setpoint* melalui *keyboard* yang antarmukanya akan dibahas dalam penelitian sistem monitoring. Selanjutnya sistem pengontrolan ini akan mengeluarkan sinyal PWM dan hasil pembacaan sensor yang akan ditampilkan oleh sistem monitoring ke monitor VGA.

Untuk memudahkan perancangan sistem pengontrolan kecepatan motor DC, perancangan dapat dibagi menjadi perancangan perangkat keras dan perancangan rangkaian dalam Modul FPGA. Perancangan perangkat keras terdiri dari perancangan rangkaian *driver* motor DC dan rangkaian sensor pembaca kecepatan. Sementara itu, perancangan rangkaian dalam Modul FPGA meliputi perancangan kontroler PID, *Input Interface*, Pembaca Sensor, dan PWM genetaror.

Masukan *keyboard* yang berupa kode ASCII akan diterjemahkan menjadi perintah dan atau data biner oleh *Input Interface*. Data biner dari keluaran *Input Interface* dan data hasil

pembacaan sensor yang didapat dari Pembaca Sensor kemudian akan diproses oleh kontroler PID. Dari kontroler PID, data akan dijadikan masukan bagi PWM generator yang pulsa keluarannya akan menentukan kecepatan putaran motor DC. Kecepatan putaran motor DC akan diukur oleh sensor kecepatan putaran berupa rotary enkoder, dan hasilnya akan dibaca oleh Pembaca Sensor. Diagram blok dari sistem pengontrolan kecepatan motor DC dapat dilihat pada Gambar 4.2.



Gambar 4.2 blok diagram sistem pengontrolan kecepatan motor DC

4.2 Perancangan Perangkat Keras

4.2.1 Perancangan *Driver* motor DC

Perancangan *driver* motor DC didasarkan pada spesifikasi motor DC yang akan digunakan. Parameter yang harus diperhatikan dalam pemilihan *driver* antara lain adalah range tegangan dan arus maksimal yang dibutuhkan oleh motor. Motor DC yang digunakan dalam perancangan ini adalah motor DC buatan Mabuchi motor dengan tipe RS 445PA 14233. Motor ini memiliki kecapatan maksimal 6500 rpm dalam keadaan tanpa beban, range tegangan antara 12-42 volt, dan arus maksimal 1,47 A pada keadaan beban penuh. Spesifikasi lengkap motor DC yang digunakan dapat dilihat pada Tabel 4.1.

Tabel 4.1 Spesifikasi motor DC

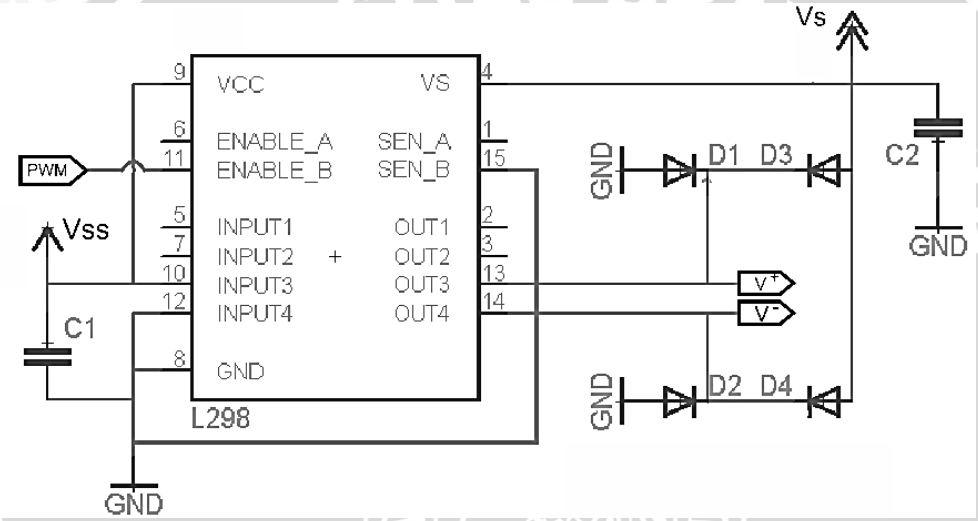
Model	Voltage		No Load		Max Efficiency		STALL		
	Operating Range	Normal	Speed (rpm)	Current (A)	Speed (rpm)	Current (A)	Torque (mN m)	Torque (G cm)	Current (A)
RS-445PA-14233	12~42	42V Constant	6500	0.060	5410	0.30	81.8	834	1.47
RS-445PA-15200	12~42	42V Constant	7600	0.067	6420	0.36	93.7	955	1.98
RS-445PD-18140	12~42	42V Constant	9300	0.070	8130	0.48	134	1366	3.35

Sumber: Mabuchi Motor, 2014

Hal yang perlu diperhatikan untuk perancangan rangkaian *driver* motor DC adalah range tegangan dan arus yang harus dapat ditahan oleh *driver*. Salah satu IC yang memenuhi persyaratan tersebut dan dapat digunakan dalam penelitian ini adalah IC L298. Dari dua buah rangkaian h-Bridge yang ada di dalam IC ini, hanya satu yang akan digunakan sebagai kendali kecepatan motor DC. IC ini akan dihubungkan dengan pin FPGA untuk menghubungkan rangkaian PWM dengan IC tersebut.

Rangkaian *driver* dengan menggunakan IC L298, menggunakan salah satu contoh skema yang ada pada datasheet. Pin 13 dan 14 pada L298 dihubungkan pada polaritas positif dan negatif motor DC. Arah putaran motor DC akan ditentukan oleh pin 10 dan 12. Pin 4 dihubungkan dengan sumber tegangan motor DC 12-42V, dan pin 9 dihubungkan pada sumber tegangan 5V untuk catu daya gerbang logika di dalam IC. Sinyal PWM sebagai sinyal enable yang mengatur kecepatan motor dihubungkan pada pin 11.

Pada penelitian ini, yang menjadi fokus dari penelitian adalah pengontrolan kecepatan motor DC. Oleh karena itu arah putaran motor pada penelitian ini dibuat tetap. Hal itu dilakukan dengan cara memberikan logika tetap pada pin 10 dan pin 12. Skema rangkaian *driver* motor DC dengan L298 dapat dilihat pada Gambar 4.3.



Gambar 4.3 Skema rangkaian Motor DC dengan H-Bridge L298

Dioda D1 sampai D4 pada rangkaian berfungsi sebagai pelindung apabila tiba-tiba terdapat arus berlebih. Dioda yang digunakan harus memiliki tegangan breakdown yang lebih besar dari sumber tegangan yang digunakan untuk motor. Selain itu, pemilihan dioda juga didasarkan pada arus maksimal yang diperlukan oleh motor saat dalam keadaan beban penuh. Berdasarkan hal-hal tersebut, maka dipilihlah dioda dengan tipe 1N5392 yang memiliki tegangan breakdown 70 volt dan tahan sampai arus 1,5 A.

Selain dioda, sebuah kapasitor juga ditambahkan pada masing-masing sumber tegangan dalam rangkaian. Kapasitor C1 berfungsi sebagai filter sumber tegangan motor sedangkan kapasitor C2 adalah filter untuk sumber tegangan logika. Dengan adanya kapasitor ini, maka tegangan yang masuk ke rangkaian dapat menjadi lebih stabil. Kapasitor yang digunakan adalah kapasitor bipolar dari bahan keramik dengan nilai 100 nF sesuai dengan yang dituliskan dalam datasheet.

4.2.2 Perancangan Rangkaian Pembaca Kecepatan Motor DC

Sensor kecepatan yang digunakan untuk mengukur kecepatan motor DC dalam penelitian ini adalah rotary enkoder produksi Depok Instrumen dengan tipe *DI-REV1 DI-Rotary Enkoder Versi 1*. Disk berlubang yang merupakan bagian dari sensor ini, dipasang pada *saft* motor sehingga disk akan memiliki kecepatan putaran sudut yang sama dengan motor. Kemudian, rangkaian sensor posisi optis akan dipasang pada bagian bawah disk untuk membaca kecepatan putaran motor. Peletakan rangkaian pendeteksi harus tepat, karena bila disk tidak dipasang terlalu tinggi atau terlalu rendah, maka hasil pembacaan sinyal akan buruk. Untuk itu, rangkaian pendeteksi harus dipasang di tempat yang stabil dan dipastikan tidak bergerak selama proses pembacaan.

Ada tiga pin yang terdapat dalam sensor ini. Pin Vcc dan Gnd dihubungkan pada tegangan sumber dan ground yang sama dengan yang digunakan untuk sumber gerbang logika pada *driver* motor. Untuk pin V_{out} yang merupakan hasil pembacaan disk, dihubungkan langsung pada pin FPGA yaitu pin JA02 yang merupakan pin masukan rangkaian pembaca sensor.

4.3 Perancangan Rangkaian dalam Modul FPGA

4.3.1 Rangkaian Input Interface

Masukan *keyboard* yang berupa kode ASCII dan sinyal busy akan diterjemahkan menjadi perintah dan data biner oleh *Input Interface*. Tombol pada *keyboard* yang terdiri dari angka, backspace, enter, panah atas dan panah bawah, akan dikelompokkan menjadi tombol angka, backspace, enter, *up*, dan *down*. Pengelompokkan tombol tersebut bertujuan untuk mempermudah perancangan rangkaian. Setiap tombol perintah memiliki fungsinya masing-masing. Tabel kebenaran dari dekoder ASCII menjadi data dan perintah dapat dilihat pada Tabel 4.2.

Pada penelitian ini, banyaknya masukan *keyboard* untuk satu nilai masukan ditentukan hanya sebanyak empat angka. Masing-masing angka akan disimpan pada register 4 bit yang menyatakan satuan (register 1), puluhan (register 2), ratusan (register 4), atau ribuan (register 4). Seperti yang ditunjukkan pada Gambar 4.4, register yang digunakan untuk menyimpan nilai biner adalah register *pararel in-pararel out*. Pada sistem konversi *keyboard* dari proyek sistem monitoring dijelaskan bahwa setiap satu tombol *keyboard* ditekan, maka akan dikirim dua sinyal busy dan dua paket data. Data yang diinginkan untuk dibaca pada rangkaian interface ini, adalah data yang kedua. Oleh karena itu, register pada sistem ini hanya akan aktif pada sinyal busy kedua.

Tabel 4.2 Tabel konversi ASCII menjadi data biner 4 bit.

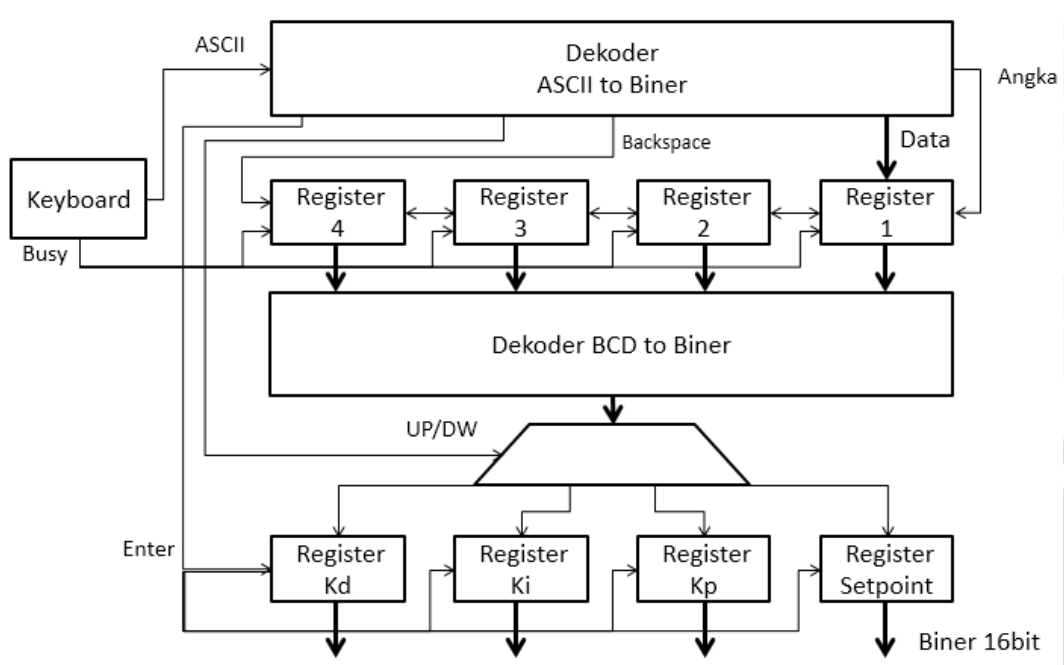
Tombol pada *keyboard*	Kode ASCII	Data
0	30h	0000
1	31h	0001
2	32h	0010
3	33h	0011
4	34h	0100
5	35h	0101
6	36h	0110
7	37h	0111
8	38h	1000
9	39h	1001
Backspace	08	1111
Enter	0D	1111
UP	18	1111
Down	19	1111

Dari data kedua itu, bila data yang terbaca adalah angka, maka data akan disimpan pada register satu pada Gambar 4.4, kemudian data yang sebelumnya ada pada register satu akan digeser pada register dua, register dua pada register tiga, dan seterusnya. Sementara itu, bila data yang terbaca adalah *backspace*, maka data register empat akan diisi nol, dan data yang sebelumnya disimpan disana akan digeser ke register di sebelah kirinya. Tombol *up* dan *down*, menentukan pada register mana nilai yang dimasukkan tadi akan disimpan, dan setiap salah satu tombol ini ditekan, data pada register satu sampai empat akan diset nol kembali. Tombol enter adalah tombol yang akhirnya akan memasukkan data yang telah dikonversi ke dalam register yang bersangkutan.

Nilai biner yang keluar dari keempat register penyimpan data, merupakan nilai biner dalam bentuk *binary code decimal* (BCD). Hal itu dikarenakan angka yang ada pada *keyboard* hanyalah angka nol sampai sembilan sehingga nilai biner keluarannya juga hanya bernilai "0000" sampai "1001". Agar nilai BCD ini dapat diproses dalam sistem yang selanjutnya, maka nilai ini harus dirubah menjadi nilai biner biasa. Metode yang digunakan untuk merubah nilai BCD menjadi nilai biner 16 bit dalam penelitian ini adalah dengan menggunakan kebalikan dari algoritma *shift-add-3*. (Alfke & New, 1997)

Setelah nilai BCD telah dikonversi menjadi kode biner, maka nilai tersebut akan disimpan ke dalam register *setpoint*, Kp, Ki, atau Kd saat tombol enter ditekan. Untuk menentukan dimana nilai biner akan disimpan, sebuah *counter* akan digunakan sebagai sinyal pemilih dalam sebuah demultiplekser. Nilai pada *counter* ini akan berubah saat tombol *up* atau *down* ditekan. Nilai biner akan disimpan pada register yang telah ditentukan itu saat

tombol enter ditekan. Diagram blok dari prinsip kerja keseluruhan Input Enkoder dapat dilihat pada Gambar 4.4.



Gambar 4.4 Blok diagram *Input Interface*

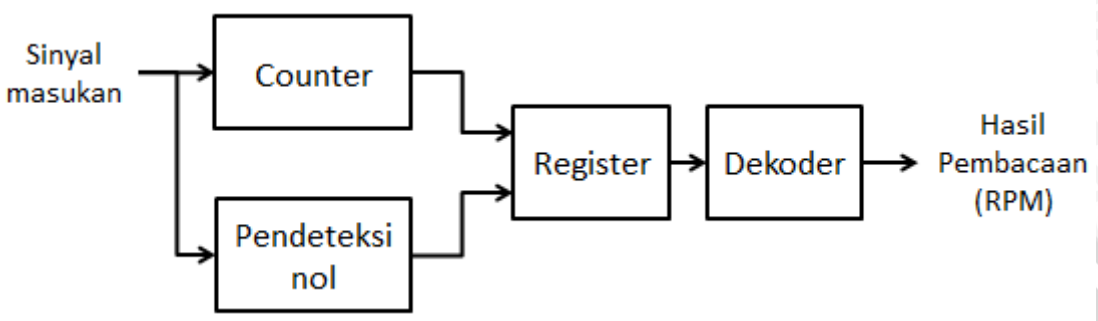
4.3.2 Rangkaian Pembaca Sensor

Hasil keluaran sensor rotary enkoder yang merupakan sederetan pulsa digital perlu diterjemahkan dahulu sebelum nilai kecepatan putaran motor dapat terbaca. Salah satu metode yang digunakan untuk membaca nilai pulsa tersebut adalah dengan menghitung banyaknya pulsa yang dikeluarkan oleh sensor dalam selang waktu tertentu. Kelebihan dari metode ini adalah kemampuannya untuk menentukan kapan motor yang diukur tersebut berhenti berputar. Apabila pada saat selang waktu tertentu yang telah ditentukan tidak ada satu pun pulsa yang dikeluarkan oleh sensor, maka motor telah berhenti berputar.

Banyaknya pulsa yang masuk ke dalam FPGA yang didapat dari sensor, dapat direpresentasikan oleh banyaknya tepi turun pulsa masukan tersebut. Komponen yang dibutuhkan oleh rangkaian pembaca sensor adalah sebuah *counter*, register, dan pendeteksi nilai nol. Pendeteksi nilai nol diperlukan agar saat kecepatan motor sangat rendah sampai mendekai nol, pembacaan tidak harus menunggu terlalu lama. Batas waktu tunggu yang ditentukan dalam penelitian ini adalah selama waktu sampling untuk kontroler PID. Blok diagram dari rangkaian pembaca sensor dapat dilihat pada Gambar 4.5.

Counter pada rangkaian ini akan bertambah setiap sinyal masukan dari sensor mengalami tepi turun. Kapasitas *counter* dalam penelitian ini adalah 16 bit, sehingga nilai maksimal yang dapat terbaca oleh *counter* adalah sebanyak 65.535 sinyal. Setiap selang

waktu sampling PID, dan saat sinyal yang masuk lebih dari nol, nilai terakhir yang terhitung oleh *counter* akan disimpan dalam register untuk dikonversi menjadi kecepatan sensor dalam satuan rotasi per detik (RPS). Frekuensi clock yang digunakan dalam rangkaian ini adalah 50 MHz. Dalam penelitian ini, selang waktu sampling untuk PID dapat diubah-ubah sesuai dengan kombinasi masukan yang diberikan. Nilai sampling itu yaitu 1 ms, 10 ms, 100 ms, dan 1 s.



Gambar 4.5 Blok diagram rangkaian pembaca sensor

4.3.3 Rangkaian Kontroler PID

Rangkaian Kontroler PID didapat dengan menggunakan persamaan pembentuk kontroler PID, kemudian merubahnya dalam bentuk diskrit, dan terakhir merealisasikannya dalam bentuk gerbang logika. Kontroler ini merupakan gabungan dari kontroler proporsional, integral, dan derivatif sehingga ketiga persamaan itu dapat dirangkai secara terpisah sebelum digabungkan menjadi satu seperti pada Persamaan (4-1) pada bab 2. Dalam penelitian ini rangkaian penjumlah dan pengali yang digunakan merupakan bagian dari FPGA yang sudah disediakan dalam bentuk CLB. Untuk mengakses rangkaian tersebut maka, dengan menggunakan compiler ISE Design Suite, digunakanlah IP Core. (Xilinx, 2011)

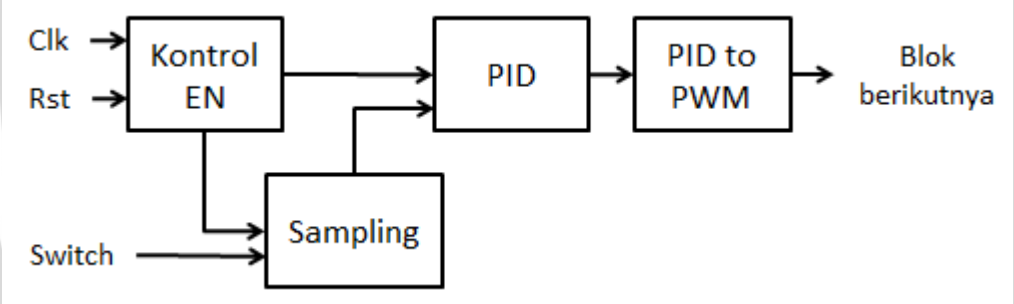
Waktu sampling untuk kontroler PID dalam penelitian ini dapat dirubah sesuai dengan kebutuhan pengguna. Waktu sampling yang disediakan adalah 1 detik, 1ms, 10ms, dan 100ms. Dua buah switch digunakan untuk memilih berapa waktu sampling yang diinginkan. Namun, perlu diperhatikan bahwa pemilihan waktu sampling ini juga akan mempengaruhi penguatan pada komponen integral dan diferensial, dan juga range pengukuran kecepatan. Prinsip kerja rangkaian PID dalam penelitian ini diperlihatkan pada Gambar 4.6, sedangkan Tabel 4.3 menunjukkan kombinasi yang digunakan untuk mengubah waktu sampling.

Dalam Persamaan (4-1), proses perhitungan dilakukan secara bertahap. Untuk itu, dibutuhkan sebuah rangkaian pengontrol agar proses perhitungan dapat berjalan secara runtut dan teratur dengan urutan seperti pada Gambar 4.7. Sebuah *counter* digunakan untuk

menghitung banyaknya sinyal clock yang dibutuhkan untuk setiap tahap. Jumlah sinyal clock yang dibuthkan oleh rangkaian penjumlah, pengurang, dan pengali dalam penelitian ini adalah satu clock, sedangkan untuk rangkaian pengurang, dibutuhkan sinyal clock sesuai banyaknya jumlah bit yang dihitung. Dalam penelitian ini, terdapat dua buah rangkaian pembagi yang keduanya digunakan masing-masing untuk menghitung nilai Ki dan Kd setelah dibagi waktu sampling, dan untuk mengkonversi hasil pembacaan sensor dalam satuan rpm. Banyaknya sinyal clock untuk rangkaian pembagi adalah sebanyak 19 clock. Untuk mengaktifkan sinyal enable ini, sebuat sinyal start diberikan secara berkala sesuai dengan waktu sampling yang digunakan.

Tabel 4.3 Tabel pengaturan waktu sampling.

Kombinasi Biner	Waktu Sampling
00	1 s
01	100 ms
10	10 ms
11	1 ms

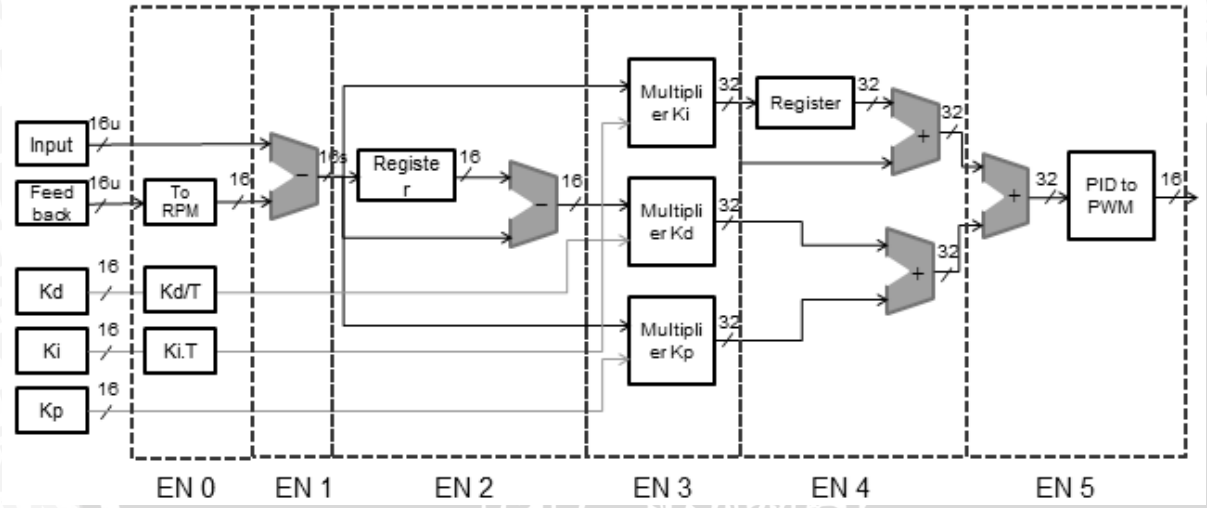


Gambar 4.6 blok diagram kontroler PID

Gambar 4.6 menunjukkan blok diagram dari rangkaian kontroler PID sesuai dengan urutannya. Dalam rangkaian tersebut, Input *setpoint*, Kp, Ki, dan Kd didapat dari rangkaian *Input Interface* sementara *feedback* didapat dari rangkaian pembaca sinyal sensor. Proses perhitungan untuk menyelesaikan Persamaan (4-1) dilakukan secara bertahap. Pada proses pertama pada EN0, nilai Ki dan Kd akan dihitung ulang sesuai dengan waktu samplingnya sesuai dengan Persamaan (2-4) dan Persamaan (2-6). Pada tahap kedua pada EN1, nilai eror dicari dengan mengurangkan nilai *setpoint* dengan *feedback*. Pada proses ketiga pada EN2, beda nilai eror yang digunakan pada komponen derivatif akan dihitung. Selanjutnya, pada EN3, nilai eror (atau beda nilai eror dalam komponen derivatif) akan dikalikan dengan penguatannya masing-masing (Kp, Ki.T, dan Kd/T). Pada EN4, penjumlahan komponen derivatif dan komponen proporsional dilakukan bersamaan dengan penjumlahan nilai saat ini dan nilai sebelumnya pada komponen integral. Pada EN5 dilakukan penjumlahan dari nilai

penjumlahan komponen derivatif dan komponen proporsional dengan komponen integral sehingga didapat hasil sesuai pada Persamaan (4-1).

Hasil keluaran dari kontroler PID adalah data sebanyak 32 bit. Agar data tersebut dapat digunakan untuk mengubah *duty-cycle* rangkaian PWM 16 bit, data itu perlu dikecilkan menjadi 16 bit. Karena pada penilitian ini kecepatan maksimum motor DC yang menjadi tujuan penelitian hanya sebesar 2500 rpm, maka nilai maksimum yang akan dicapai oleh perhitungan kontroler PID tidak akan penuh mencapai 32 bit. Oleh karena itu, hasil PID yang digunakan untuk mengatur *duty-cycle* PWM, hanyalah 16 bit pertama saja. Jika hasil PID seandainya lebih dari 16 bit, maka nilai PWM yang akan dihasilkan menjadi nilai maksimalnya.



Gambar 4.7 Kontroler PID dan tahapannya

Konversi untuk merubah nilai masukan feedback dari rangkaian pembaca sensor menjadi nilai sebenarnya dalam satuan rpm, didapat berdasarkan Persamaan (4-1). Pada saat waktu sampling yang digunakan adalah satu detik, jumlah sinyal pada sensor dikalikan enam puluh untuk mendapatkan jumlah sinyal per menit, dan dibagi tiga puluh enam, sesuai dengan banyaknya lubang pada disk sensor.

$$RPM = sensor \times \frac{60}{36 \times t_{sampling}} = sensor \times \frac{5}{3 \times t_{sampling}} \quad (4\text{-}1)$$

4.3.4 Rangkaian PWM

Sinyal PWM digunakan untuk mengatur kecepatan motor DC. Nilai PWM didapatkan dengan membuat sebuah *counter* yang terus menghitung dari nol sampai nilai maksimalnya kemudian kembali menghitung dari kondisi awal. Selanjutnya sebuah masukan referensi diberikan untuk dibandingkan dengan nilai *counter*. Apabila nilai *counter* kurang atau sama dengan nilai referensi, maka keluaran PWM akan berlogika *high*. Sebaliknya, apabila nilai

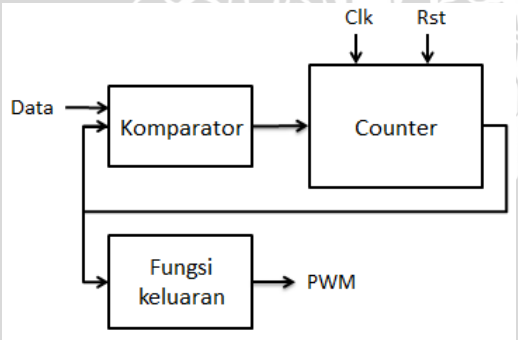
counter lebih tinggi dari nilai referensi maka keluaran PWM akan berlogika *low*. Timing diagram dari perancangan PWM dapat dilihat pada Gambar 4.8.

Gambar 4.8 Timing diagram PWM

Ada beberapa hal yang dibutuhkan dalam perancangan PWM generator 16 bit. Pertama adalah sebuah *counter* 16 bit yang aktif saat sinyal *CLK* tepi turun. Banyaknya nilai *counter* dan frekuensi sinyal masukan yang digunakan untuk *counter* akan menentukan frekuensi sinyal PWM yang dihasilkan. Frekuensi sinyal PWM yang digunakan dalam penelitian ini tidak dispesifikasikan dalam datasheet motor DC. Namun, nilai yang umum digunakan minimal adalah 100 Hz. *Counter* yang digunakan dalam perancangan ini adalah *counter* 16 bit dengan nilai maksimal 65.535. Oleh karena itu, waktu yang dibutuhkan dengan frekuensi *CLK* 50 MHz adalah 1,3107 ms, seperti yang dapat dilihat pada Persamaan (4-2).

$$t = 65.535 \times 20ns \ = 1.310.700ns \quad (4\text{-}2)$$

Komponen yang kedua adalah sebuah komparator. Komparator ini akan membandingkan nilai yang ada pada data masukan dengan nilai *counter*. Saat data masukan lebih dari atau sama dengan nilai *counter* saat ini, maka sinyal keluaran PWM akan berlogika *high*. Sementara itu, saat nilai data masukan lebih dari nilai *counter*, maka sinyal PWM akan berlogika *low*. Diagram Moore dari perancangan PWM dapat dilihat pada Gambar 4.9.



Gambar 4.9 Diagram Moore dari perancangan PWM

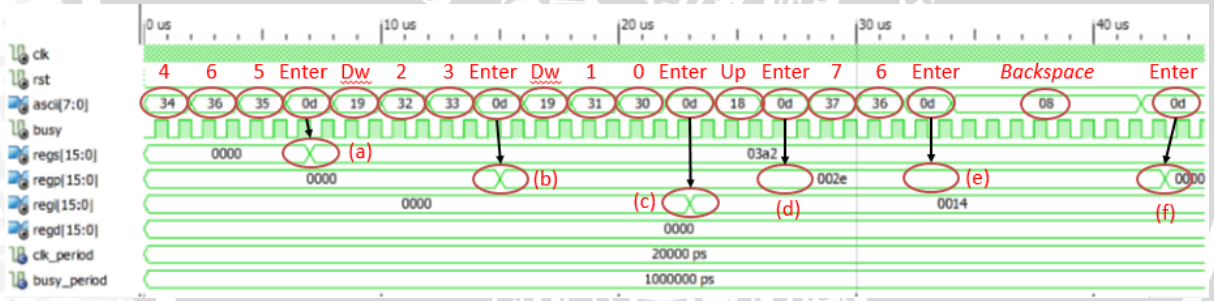
BAB 5
PENGUJIAN DAN ANALISIS

Pengujian dilakukan untuk melihat kinerja sistem sudah sesuai dengan apa yang diinginkan. Data pengujian dan simulasi akan dijadikan acuan dalam mengambil kesimpulan. Pengujian dilakukan dibagi menjadi dua bagian yakni pengujian per bagian dan pengujian sistem secara keseluruhan.

5.1 Simulasi dan Pengujian Rangkaian dalam Modul FPGA

5.1.1 Input Interface

Pengujian pada bagian *Input Interface* adalah pengujian untuk menentukan apakah nilai masukan *keyboard* yang tadinya merupakan kode ASCII sudah dikonversi dengan benar menjadi kode biner. Pengujian ini bertujuan agar sistem dapat menerima masukan dengan tepat dari sistem *monitoring*. Pengujian rangkaian *Input Interface* dilakukan dengan menggunakan simulasi pada program ISIM Simulator dan implementasinya pada FPGA Nexys 2. Gambar 5.1 adalah gambar yang menunjukkan timing diagram simulasi rangkaian *Input Interface*.



Gambar 5.1 Timing diagram simulasi *Input Interface*

Pada simulasi ini, sinyal busy yang merupakan sinyal yang muncul setiap terjadi penekanan tombol *keyboard* diumpamakan memiliki periode 1 µs. Pada kondisi sebenarnya, nilai periode sinyal ini ditentukan oleh kecepatan penekanan *keyboard* yang nilai rata-ratanya berkisar antara 1 ms. Simulasi pada Gambar 5.1 menunjukkan penekanan beberpa tombol angka sebelum tombol enter. Angka yang ditekan adalah 4, 6, dan 5. Saat tombol enter ditekan, nilai 465 yang sudah dimasukkan tadi akan dikonversi menjadi kode biner, kemudian nilainya disimpan pada register set. Bagian dengan tanda (a) pada Gambar 5.1, menunjukkan transisi data pada register set saat tombol enter ditekan.

Saat salah satu tombol panah (*up* atau *down*) ditekan, nilai yang ada pada register yang dituju akan dikeluarkan dari simpanan, tetapi nilai baru yang akan dimasukkan tetap dihitung

dari awal. Menurut perancangan yang sudah dibahas sebelumnya, saat tombol *down* ditekan, nilai yang sebelumnya disimpan pad register set, akan disimpan pada register berikutnya dengan urutan Kp, Ki, dan Kd. Simulasi berikutnya dilakukan dengan menekan tombol 2 dan 3. Nilai yang tersimpan setelah penekanan tombol angka ini berubah menjadi 0023. Nilai tersebut kemudian dikonversi menjadi 0016 dan saat tombol enter ditekan, biner tersebut disimpan pada register Kp seperti pada tanda (b).

Setelah tombol enter ditekan, maka nilai pada register yang saat itu dipilih, akan dimasuki nilai biner yang baru. Namun, apabila setelah tombol enter ditekan kemudian dilakukan penekanan tombol angka, maka nilai tersebut tidak akan masuk pada register karena setelah empat tombol angka atau tombol enter ditekan, register baru dapat diisi kembali setelah isinya dihapus terlebih dahulu. Pada Gambar 5.1 tanda (c) dapat dilihat bahwa saat tombol angka ditekan setelah penekanan enter, nilai pada register adalah tetap dan tidak berubah. Kemudian setelah empat kali penekanan tombol *backspace* kemudian ditekan enter, maka nilai register berubah menjadi 0 kembali.

Setelah rangkaian *Input Interface* diimplementasikan pada FPGA dan dilakukan urutan penekanan *keyboard* yang sama dengan yang ada pada simulasi, maka didapatkan hasil seperti pada Tabel 5.1. Pada pengujian ini, nilai ASCII ditampilkan pada LED sementara nilai pada register ditampikan pada *seven segment*. Dari tabel tersebut, dapat kita simpulkan bahwa hasil implementasi rangkaian *Input Interface* memiliki hasil yang tidak berbeda jauh dengan nilai pada hasil simulasi. Perbedaan yang ada pada keduanya, terjadi akibat konversi dari nilai BCD menjadi biner. Pada nilai masukan yang ganjil, hasil konversi memiliki selisih satu dengan nilai yang dimasukkan. Namun selain dari itu, tidak ada hal lain yang berbeda.

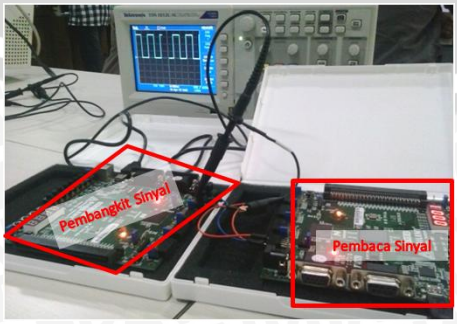
5.1.2 Pembaca Sensor

Pengujian pembaca sensor bertujuan untuk memastikan *feedback* yang masuk ke dalam rangkaian kontroler adalah nilai yang tepat. Selain itu, pengujian ini dilakukan untuk mencaritahu nilai minimal dan maksimal dari sinyal yang dapat dibaca. Pengujian ini dilakukan dengan cara memberikan sebuah sinyal digital pada pin masukan pembaca sensor. Frekuensi dari sinyal masukan itu kemudian akan diubah-ubah dan hasil pembacaannya dicatat selama 10 detik atau sepuluh kali waktu sampling. Pembacaan dilihat selama sepuluh kali bertujuan untuk melihat apakah data yang terbaca itu stabil ataukah nilainya berubah-ubah. Sinyal yang dibaca oleh rangkaian ini berasal dari modul FPGA yang lain. Susunan rangkaian pengujian dapat dilihat pada Gambar 5.2.

Tabel 5.1 Pengujian implementasi *Input Interface*

Tombol *Keyboard* yang ditekan	Kode ASCII pada LED	Register Set	Register Kp	Register Ki	Register Kd
4	34_H	0000_H	0000_H	0000_H	0000_H
6	36_H	0000_H	0000_H	0000_H	0000_H
5	35_H	0000_H	0000_H	0000_H	0000_H
Enter	$0D_H$	$01D0_H$	0000_H	0000_H	0000_H
Down	19_H	$01D0_H$	0000_H	0000_H	0000_H
2	32_H	$01D0_H$	0000_H	0000_H	0000_H
3	33_H	$01D0_H$	0000_H	0000_H	0000_H
Enter	$0D_H$	$01D0_H$	0016_H	0000_H	0000_H
Down	19_H	$01D0_H$	0016_H	0000_H	0000_H
1	31_H	$01D0_H$	0016_H	0000_H	0000_H
0	30_H	$01D0_H$	0016_H	0000_H	0000_H
Enter	$0D_H$	$01D0_H$	0016_H	$000A_H$	0000_H
UP	18_H	$01D0_H$	0016_H	$000A_H$	0000_H
Enter	$0D_H$	$01D0_H$	0016_H	$000A_H$	0000_H
7	37_H	$01D0_H$	0016_H	$000A_H$	0000_H
6	36_H	$01D0_H$	0016_H	$000A_H$	0000_H
Enter	$0D_H$	$01D0_H$	0016_H	$000A_H$	0000_H
Backspace	08_H	$01D0_H$	0016_H	$000A_H$	0000_H
Backspace	08_H	$01D0_H$	0016_H	$000A_H$	0000_H
Backspace	08_H	$01D0_H$	0016_H	$000A_H$	0000_H
Backspace	08_H	$01D0_H$	0016_H	$000A_H$	0000_H
Enter	$0D_H$	$01D0_H$	0000_H	$000A_H$	0000_H

Dalam Tabel 5.2, terdapat data hasil pengujian implementasi rangkaian pembaca sensor. Dengan kapasitas 16 bit, rangkaian ini dapat membaca sampai frekuensi 65 kHz. Dalam pengujian ini, waktu sampling yang digunakan untuk memcara keluaran sensor adalah satu detik. Satu detik dipilih, karena lebih mudah diamati dalam *seven segment display*. Namun, rangkaian ini memiliki kesulitan untuk membaca banyaknya pulsa dengan periode yang sama dengan waktu samplingnya. Misalnya saja dalam pengujian ini adalah sinyal 1 Hz dengan waktu sampling satu detik. Hasil yang terbaca dan ditampilkan dalam seven segment menajdi nol. Hal itu dikarenakan, waktu sampling minimal untuk membaca sebuah sinyal adalah dua kali dari frekuensi sinyal itu.



Gambar 5.2 Pengujian implementasi rangkaian pembaca sensor

Tabel 5.2 Data hasil pembacaan sensor

Freq	Tampilan *seven segment display* detik ke-									
	1	2	3	4	5	6	7	8	9	10
1 Hz	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
10 Hz	000A	000A	000A	000A	000A	000A	000A	000A	000A	000A
100 Hz	0064	0064	0064	0064	0064	0064	0064	0064	0064	0064
1 kHz	03C8	03C8	03C8	03C8	03C8	03C8	03C8	03C8	03C8	03C8
8,333 kHz	1F33	1F33	1F33	1F33	1F34	1F33	1F33	1F33	1F33	1F33
10 kHz	2710	2710	2710	2710	2710	2710	2710	2710	2710	2710
100 kHz	86A0	86A0	86A0	86A0	86A0	86A0	86A0	86A0	86A0	86A0

Saat waktu sampling yang digunakan berbeda dan bukan satu detik, terdapat pergeseran range frekuensi pembacaan dari rangkaian ini. Hal itu dikarenakan saat waktu sampling semakin kecil, maka dengan frekuensi yang sama, banyaknya sinyal yang terbaca dalam selang waktu tersebut menjadi semakin kecil. Hal itu menyebabkan semakin kecil waktu sampling, maka range minimal yang dapat terbaca menjadi semakin besar. Tabel 5.3 menampilkan data yang terbaca dengan frekuensi sinyal masukan yang sama, namun dengan waktu sampling berbeda. Data itu menunjukkan, semakin kecil waktu samplingnya, maka banyaknya sinyal yang dapat terbaca dalam waktu itu menjadi semakin berkurang.

Tabel 5.3 Data pembacaan sensor dengan waktu sampling berbeda

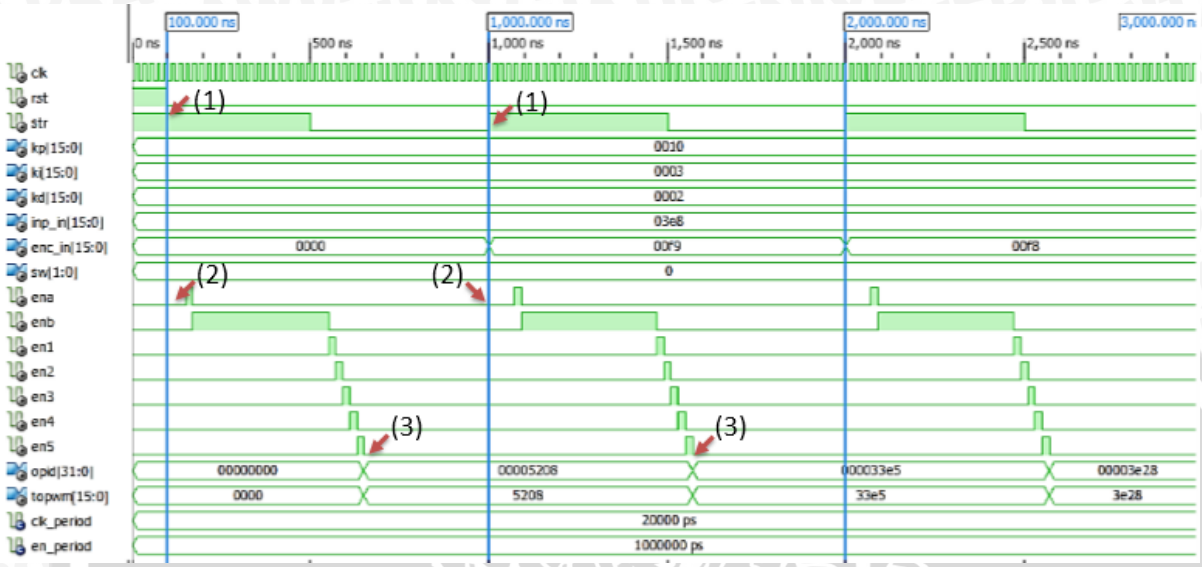
Switch	$T_{sampling}$	$f_{masukan}$	Tampilan *seven segment display*				
			1	2	3	4	5
00	1 s	1 kHz	03E8	03E8	03E8	03E8	03E8
01	100 ms	1 kHz	0064	0064	0064	0064	0064
10	10 ms	1 kHz	000A	000A	000A	000A	000A
11	1 ms	1 kHz	0000	0000	0000	0000	0000

5.1.3 Kontroler PID

Pengujian kontroler PID dilakukan untuk memastikan bahwa kontroler PID hanya akan aktif setiap waktu yang telah ditentukan. Hal ini penting karena bila kontroler terus berjalan tanpa henti, maka hasil perhitungan akan terus dilakukan meskipun belum waktunya sehingga menjadi tidak akurat. Proses perhitungan kontroler PID akan disimulasikan dalam ISIM Simulator. Nilai *setpoint*, Kp, Ki, dan Kp dalam simulasi ini disesuaikan dengan nilai pada pengujian keseluruhan menggunakan seluruh alat, sementara nilai *feedback* didapat dari data pengujian keseluruhan.

Simulasi sinyal *enable* untuk kontroler PID diperlihatkan pada Gambar 5.3. Pada tanda (1), kontroler PID akan mulai menghasilkan sinyal *enable* beberapa saat setelah sinyal start STR berada pada kondisi tepi naik atau pada saat sinyal RST berhenti diaktifkan. Pada saat itu terjadi, sinyal STR akan mengaktifkan sebuah *counter* yang akan mengaktifkan sederetan

sinyal *enable* untuk rangkaian penjumlah, pengurang, atau pengali secara bergantian seperti pada tanda (2). Setelah sinyal *enable* EN5 aktif, maka didapatkan hasil perhitungan seperti yang ditunjukkan pada tanda (3). Setelah perhitungan itu selesai, maka kontroler akan dalam kondisi *hold* sampai ada sinyal *enable* berikutnya. Waktu yang dibutuhkan rangkaian PID untuk satu kali perhitungan adalah sebesar 570 ns.

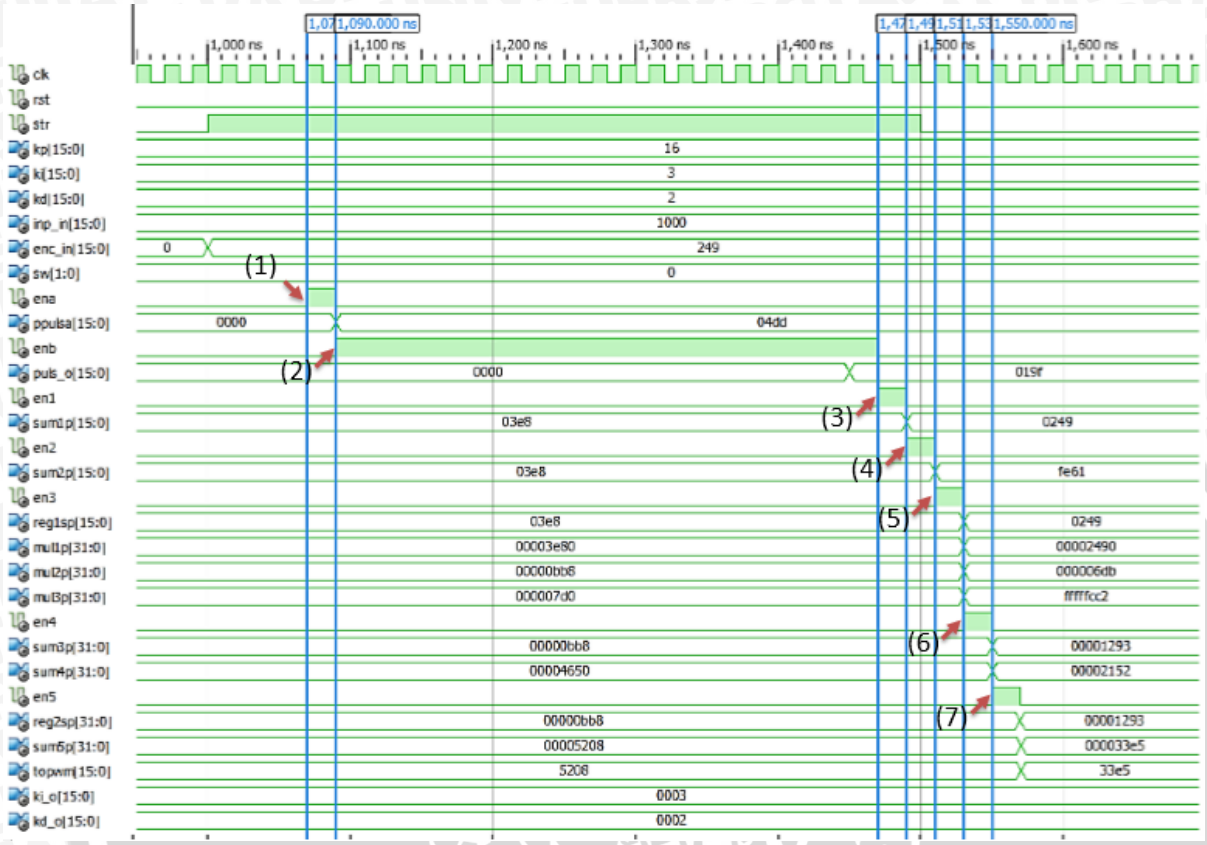


Gambar 5.3 Simulasi sinyal *enable* pada kontroler PID

Terdapat total tujuh sinyal *enable* untuk mengatur *timing* kontroler PID pada penelitian ini. Untuk lebih jelasnya, Gambar 5.4 menunjukkanpada bagian mana masing-masing sinyal enable mempengaruhi kontroler PID. Dalam timing itu terlihat bahwa seluruh rangkaian penjumlah, pengurang, pengali, dan pembagi, aktif pada saat logika CLK tepi naik dan *enable* dalam kondisi *high*. Setelah aktif dan mendapatkan hasil akhir, maka rangkaian-rangkaian itu akan masuk kondisi *hold* sampai ada sinyal *enable* berikutnya.

Pada Gambar 5.4 yang ditunjukkan oleh tanda (1) dan (2), sinyal enable enA dan enB mengaktifkan rangkaian pengali dan pembagi yang mengubah nilai Ki, Kd, dan feedback, sesuai dengan waktu sampling yang digunakan. Pada tanda (3), rangkaian pengurang 1 aktif sehingga didapatkan nilai eror pada sinyal sum1. Tanda (4) menunjukkan proses pengurangan untuk mencari beda nilai eror. Pada tanda (5), nilai sum2 disimpan pada register 1 bersamaan dengan perkalian antara nilai eror dengan Kp, nilai eror dengan Ki, dan beda nilai eror dengan Kd yang hasilnya disimpan pada Mul1, Mul2, dan Mul3. Selanjutnya, tanda (6) menunjukkan penjumlahan antara nilai komponen proporsional Mul1 dengan komponen derivatif Mul3 pada sinyal sum3, dan nilai komponen integral saat ini, Mul2, dengan nilai komponen integral sebelumnya pada register 2 yang ditunjukkan oleh sinyal sum4. Kemudian

pada tanda (7), nilai sum3 dan sum4 dijumlahkan sehingga didapatkan hasil perhitungan PID bersamaan dengan nilai sum4 yang disimpan pada register 2.

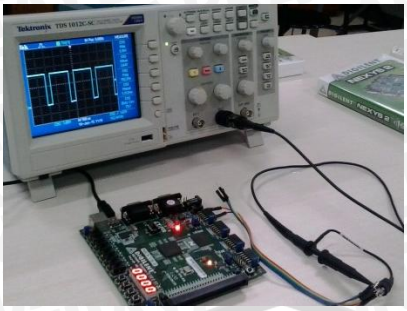


Gambar 5.4 Simulasi timing Kontroler PID

5.1.4 PWM

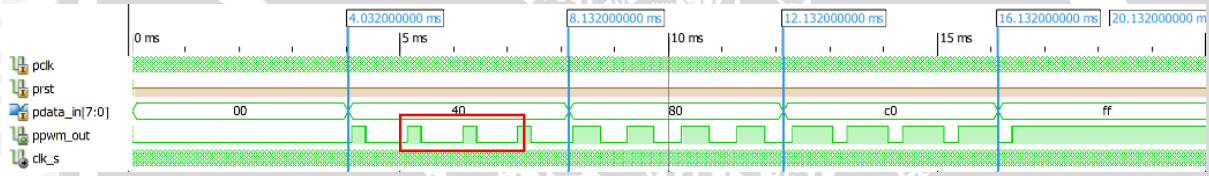
Pengujian pada rangkaian PWM dilakukan dengan dua cara. Cara pertama adalah dengan menggunakan program simulasi ISIM Simulator dan yang kedua adalah dengan mengimplementasikan rangkaian pada FPGA Nexys2 dan mengamati sinyal keluarannya pada osiloskop. Pengujian ini dilakukan untuk mengetahui bentuk sinyal, *duty-cycle*, dan frekuensi dari sinyal yang dihasilkan. Cara kedua adalah dengan mengimplementasikannya dlaam FPGA kemudian menampilkan hasilnya dalam osiloskop seperti pada Gambar 5.5.

Pada Simulasi dengan menggunakan ISIM Simulator, frekuensi sinyal PWM yang dikeluarkan dilihat dengan cara melihat periode dari sinyal PWM tersebut. Periode sinyal dilihat pada saat sinyal PWM tepi naik sampai transisi tepi naik berikutnya. Sementara itu, *duty-cycle* dari keluaran dilihat dari perbandingan antara lamanya logika *high* pada sinyal keluaran dengan periode sinyal tersebut. Gambar simulasi dari sinyal keluaran PWM dengan ISIM Simulator dapat dilihat pada Gambar 5.6 dan Gambar 5.7.

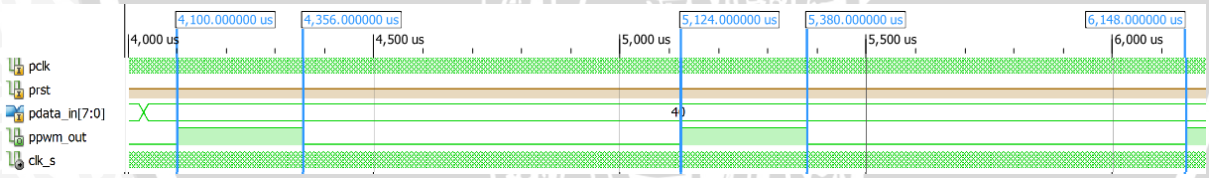


Gambar 5.5 Pengujian implementasi PWM

Dari hasil simulasi pada Gambar 5.6, dapat dilihat bahwa saat dengan memasukkan nilai data yang berbeda, akan didapatkan sinyal keluaran dengan *duty-cycle* yang berbeda juga. PWM 0% didapat dengan memasukkan data x"0000" dan nilai maksimum didapat dengan memasukkan data x"FFFF". Jika salah satu sinyal PWM diperbesar seperti pada Gambar 5.7, *duty-cycle* dapat diukur dengan melakukan perbandingan antara lamanya sinyal berlogika *high* dengan periode keseluruhan sinyal seperti pada Persamaan (2-8).



Gambar 5.6 Simulasi PWM dengan beberapa *duty-cycle*

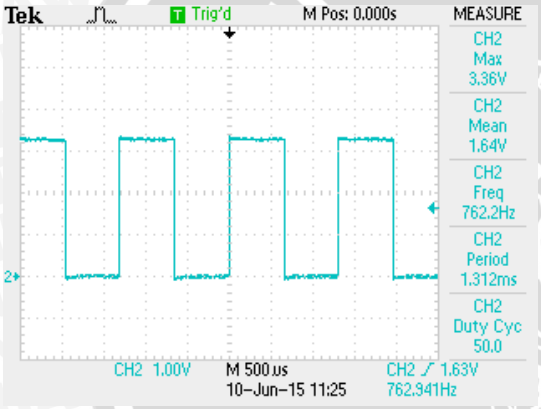


Gambar 5.7 Simulasi PWM saat *duty-cycle* 25%

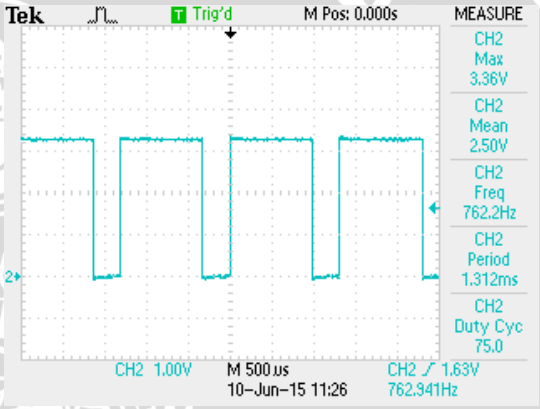
Pengujian berikutnya adalah dengan mengimplmentasikan hasil keluaran PWM dan menampilkan sinyalnya pada osiloskop. Sinyal keluaran akan dikeluarkan pada salah satu pin pada FPGA kemudian diukur menggunakan osiloskop. Parameter yang diukur dengan osiloskop antara lain adalah frekuensi, periode, dan *duty-cycle*. Gambar 5.8 dan Gambar 5.9 adalah tampilan osiloskop dari implementasi PWM dalam FPGA. Tabel 5.4 menunjukkan beberapa kombinasi biner masukan dan hasil pengukuran sinyal PWM yang dihasilkan. Dari tabel itu, dapat kita simpulkan bahwa semakin besar nilai yang diberikan, maka *duty-cycle* sinyal PWM juga akan semakin besar. Sementara itu, frekuensi dan periode sinyal PWM selalu tetap.

Tabel 5.4 Data implementasi PWM

Masukan	PWM		
	Freq	Periode	*Duty-cycle*
0001101000000000	762,941 Hz	1,312 ms	10%
0010011100000000	762,941 Hz	1,312 ms	15%
0011010000000000	762,941 Hz	1,312 ms	20%
0100000000000000	762,941 Hz	1,312 ms	25%
0100110100000000	762,941 Hz	1,312 ms	30%
0101101000000000	762,941 Hz	1,312 ms	35%
0110011100000000	762,941 Hz	1,312 ms	40%
1000000000000000	762,941 Hz	1,312 ms	50%
1001101000000000	762,941 Hz	1,312 ms	60%
1011010000000000	762,941 Hz	1,312 ms	70%
1100000000000000	762,941 Hz	1,312 ms	75%
1100110100000000	762,941 Hz	1,312 ms	80%
1110010000000000	762,941 Hz	1,312 ms	90%



Gambar 5.8 Sinyal PWM 50%



Gambar 5.9 Sinyal PWM 75%

5.2 Hasil Sintesis Rangkaian dalam Modul FPGA

Sintesis FPGA adalah sebuah proses untuk mengaplikasikan rangkaian dari bentuk program menjadi implementasinya dalam FPGA. Dalam proses ini selain program diperiksa apakah penulisannya sudah benar atau belum, diperiksa juga apakah program yang ditulis cocok dengan tipe FPGA yang dipilih. Langkah selanjutnya untuk implementasi dalam FPGA adalah *Implementation*. Pada tahap ini, compiler mencocokkan program dan sumber daya yang ada pada FPGA kemudian membuat peta penggunaan alatnya.

Setelah menggabungkan seluruh program rangkaian dalam FPGA, maka akan didapatkan hasil seperti pada Tabel 5.5. Beberapa sumber daya yang digunakan oleh keseluruhan rangkaian ini antara lain: flip-flop, LUT, *bonded* IOB, RAM, BUFMUX, DCM, dan *Multiplier*. Hal ini berarti bahwa rangkaian yang telah dirancang dapat 100%

diimplementasikan dalam FPGA Nexys 2. Rangkaian ini berhasil disintesis dengan sempurna pada alat xc3s500e-4fg320. *Design goal* yang digunakan untuk melakukan sintesis pada rangkaian ini adalah *Balance*, dan *design strategy* yang digunakan *default*.

Tabel 5.5 Hasil sintesis keseluruhan sistem

Logic Utilization	Used	Avaiable	Utilization
Number of Slice Flip-Flops	1.872	9.312	20%
Number of 4 input LUTs	1.537	9.312	16%
Number of occupied Slices	1.572	4.656	33%
Number of Slices containing only related logic	1.572	741	100%
Number of Slices containing unrelated logic	0	741	0%
Total Number of 4 input LUTs	1.692	9.312	18%
Number used as logic	1.537		
Number used as a route-thru	155		
Number of bonded IOBs	41	232	17%
IOB Flip-Flops	2		
Number of RAMB16s	8	20	40%
Number of BUFMUXs	5	24	20%
Number of DCMs	1	4	25%
Number of MULT 18x18SIOs	5	20	25%
Average Fanout of Non-Clock Nets	2,97		

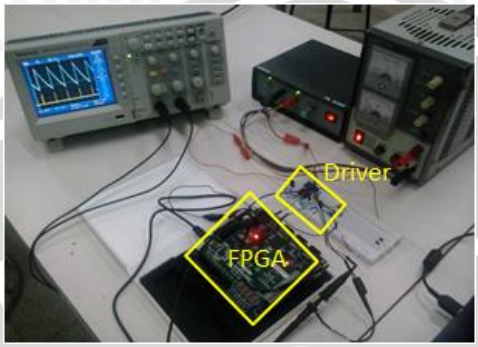
5.3 Pengujian *Hardware*

5.3.1 Pengujian Rangkaian Driver Motor

Pengujian rangkaian *driver* motor dilakukan untuk mencari hubungan antara tegangan keluaran *driver* yang akan dihubungkan pada motor dengan sinyal PWM yang diberikan. Sinyal PWM pada pengujian ini berasal dari modul PWM yang sudah teruji dengan baik. Untuk melakuakan pengujian ini, rangkaian *driver* dibuhungkan dengan sumber tegangan untuk motor sebesar 12 volt dan tegangan untuk gerbang logika sebesar 5 volt. Rangkaian kemudian dihubungkan dengan FPGA untuk menghubungkan sinyal PWM yang berada pada pin JA01 seperti pada Gambar 5.10. Sinyal PWM ini akan dihubungkan pada pin 11 pada *driver* yaitu pada sinyal *enable*. Sinyal PWM dan tegangan keluaran antara pin 13 dan pin 14 *driver* kemudian akan ditampilkan pada osiloskop untuk mengetahui pengaruh sinyal PWM terhadap keluaran *driver*.

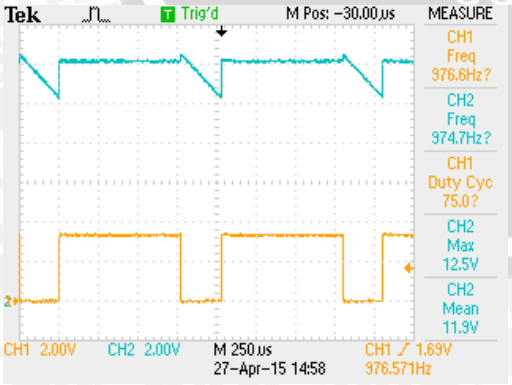
Gambar 5.11 dan Gambar 5.12 menunjukkan hasil sinyal keluaran pada pin 13 dan 14 dengan tegangan sumber 12 volt dalam keadaan tanpa beban. CH1 menunjukkan besarnya *duty-cycle* sinyal PWM yang diberikan, sedangkan CH2 merupakan tegangan keluaran *driver* motor. Pada keadaan tanpa beban, saat sinyal PWM bernilai *high*, maka tegangan keluaran akan bernilai maksimum sesuai dengan sumber tegangan motornya. Saat sinyal PWM bernilai *low*, maka tegangan keluaran akan turun secara perlahan sampai nilai PWM kembali

high. Namun, karena tegangan tidak segera turun, hal ini menyebabkan tegangan rata-rata *driver* pada kondisi tanpa beban menjadi cukup tinggi. Pada keadaan dengan beban, saat sinyal PWM bernilai *high*, maka tegangan keluaran juga akan bernilai maksimum. Namun, saat sinyal PWM bernilai *low*, tegangan keluaran akan turun secara drastis hingga terkadang sampai dibawah nol.

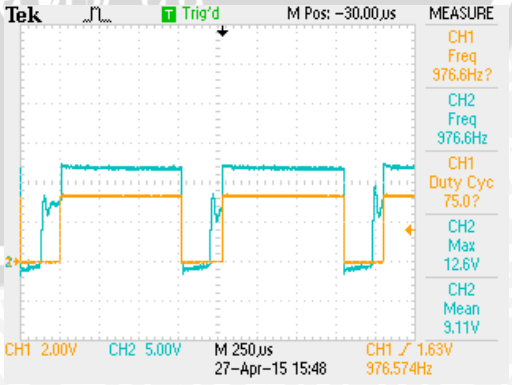


Gambar 5.10 Pengujian rangkaian *driver* motor

Data pada tabel Tabel 5.6 menunjukkan besarnya tegangan keluaran pada *driver* motor dalam keadaan tanpa beban dan dengan beban berupa motor DC. Pada keadaan tanpa beban, tegangan rata-ratanya jauh lebih besar karena tegangan tidak langsung turun saat sinyal PWM berlogika *low*. Bahkan saat diberikan sinyal PWM dengan *duty-cycle* kurang dari satu persen, tegangan rata-rata yang dikeluarkan masih berkisar delapan volt. Lain halnya pada keadaan dengan beban. Pada keadaan ini, bentuk tegangan keluaran menyerupai bentuk sinyal PWM, sehingga tegangan rata-ratanya juga semakin kecil, bahkan mendekati 0 volt saat diberi sinyal PWM dengan *duty-cycle* dibawah 1%.



Gambar 5.11 Sinyal keluaran tanpa beban dengan PWM 75%



Gambar 5.12 Sinyal keluaran dengan beban dengan PWM 75%

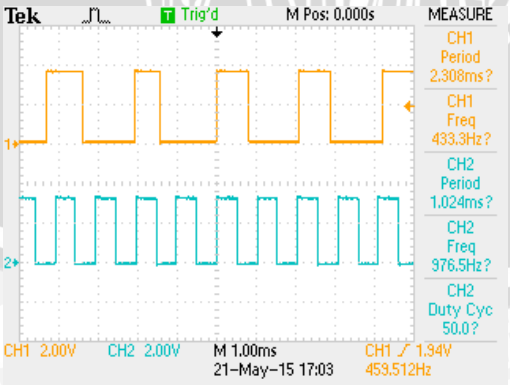
Tabel 5.6 Data pengujian rangkaian *driver* motor

Duty-cycle PWM	V_{out} tanpa beban (V)	V_{out} dengan beban (V)
>1%	8,15	0.01
10%	8,97	0,93
20%	9,63	2,66
30%	10,1	4,50
40%	10,6	5,84
50%	11,1	7,08
60%	11,5	8,18
70%	11,8	8,81
80%	12,0	9,29
90%	12,2	10,30
100%	12,2	11,80

5.3.2 Pengujian sensor rotary enkoder

Pengujian rangkaian rotary enkoder dilakukan untuk melihat bentuk sinyal dan mencari tahu hubungan antara sinyal PWM dengan kecepatan sensor yang dilihat dari frekuensi sinyal keluaran. Pengujian ini dilakukan dengan cara memberikan sinyal PWM dengan *duty-cycle* tertentu pada *driver* motor DC yang sudah dipasangi dengan piringan rotary enkoder. Motor DC akan memutar piringan sehingga sensor akan mengeluarkan sinyal digital. Hasil pembacaan sinyal dan sinyal PWM yang diberikan akan ditampilkan dalam osiloskop.

Gambar 5.13 menunjukkan tampilan osiloskop dari sinyal keluaran sensor dan sinyal PWM yang diberikan pada motor. Sinyal keluaran sensor ditunjukkan pada CH1 sedangkan sinyal PWM ditunjukkan pada CH2. Saat *duty-cycle* PWM dinaikkan, maka Kecepatan putaran motor juga akan meningkat sehingga frekuensi dari sinyal masukan sensor juga akan bertambah. *Duty-cycle* sinyal PWM tidak mempengaruhi bentuk sinyal keluaran sensor, yang berubah hanya frekuensinya saja.



Gambar 5.13 Sinyal keluaran *rotary enkoder* dengan PWM 50%

Tabel 5.7 adalah data yang menunjukkan perubahan frekuensi sinyal keluaran sensor berdasarkan sinyal PWM yang diberikan. Dari data tersebut, pada saat *duty-cycle* yang diberikan kurang dari 20%, maka frekuensi sinyal keluaran sensor menjadi sangat rendah.

Hal itu dikarenakan saat *duty-cycle* sinyal PWM yang diberikan rendah,maka tegangan rata-rata yang dihasilkan *driver* motor juga cukup rendah hingga motor tidak memiliki cukup torsi untuk berputar. Hal ini yang akhirnya menyebabkan piringan sensor tidak berputar hingga sensor tidak memiliki sinyal keluaran.

Tabel 5.7 Data pengujian sensor rotary enkoder

Duty-cycle	*f*
10%	>1 Hz
20%	>1 Hz
30%	90,3 Hz
40%	302,56 Hz
50%	459,51 Hz
60%	601,75 Hz
70%	672,28 Hz
80%	765,83 Hz
90%	835,62 Hz
100%	932,80 Hz

5.4 Pengujian Sistem Keseluruhan

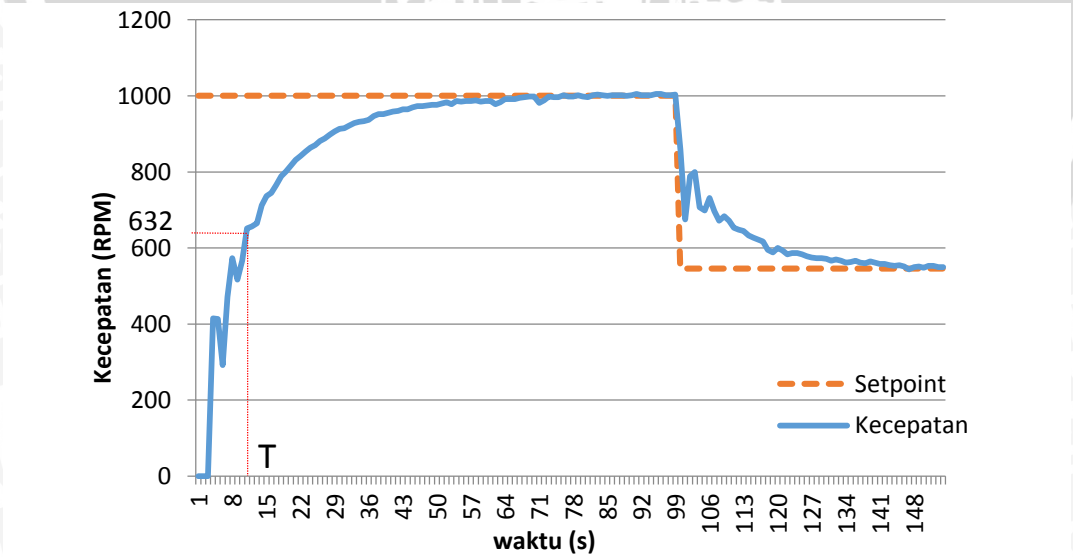
Tujuan dari pengujian ini adalah untuk mengetahui kinerja sistem secara keseleruhan. Pengujian sistem dilakukan setelah sintesis seluruh program dalam FPGA dan menggabungkannya dengan *Hardware*. Parameter Kp, Ki, Kd akan dibuat tetap sementara nilai *setpoint* akan diubah-ubah. Keluaran sistem akan diamati melalui osiloskop, dalam bentuk tegangan keluaran dari FPGA ke motor dan dari hasil pembacaan sensor yang ditampilkan dalam *seven segment display*. Sistem dinyatakan baik apabila hasil pembacaan sensor sudah sesuai dengan apa yang dimasukkan dalam *setpoint*.

Kecepatan putaran motor yang ditampilkan dalam *seven segment display*, adalah banyaknya sinyal yang dikeluarkan sensor dalam wantu sampling yang telah ditentukan dalam bentuk heksa. Nilai kecepatan sebenarnya dari motor itu baru didapatkan dari Persamaan (4-1). Tabel 5.8 menunjukkan data selama 153 detik yang menunjukkan kinerja dari rangkaian pengontrol kecepatan motor DC. Dari data itu dibuatlah grafik seperti pada Gambar 5.14.

Pada pengujian ini, nilai setpoint yang diberikan adalah 1000 rpm kemudian diturunkan menjadi 546 rpm setelah sistem dalam keadaan stabil. Parameter yang diberikan adalah Kp sebesar 16, Ki sebesar 3, dan Kd sebesar 2. Dari hasil pengujian ini ditunjukkan bahwa kontroler PID yang telah dibuat dapat mengatur kecepatan motor DC sesuai dengan setpoint yang diberikan. Kecepatan motor akan naik secara perlahan sampai mendekati kecepatan yang ditentukan. Saat kecepatan motor tiba-tiba kita turunkan, sistem merespon dengan baik.

Tabel 5.8 Data pengujian keseluruhan

Detik ke-	*Setpoint* (rpm)	Kp	Ki	Kd	Nilai yang ditampilkan	Nilai setelah konversi (rpm)
0	1000	16	3	2	0000 H	0
1	1000	16	3	2	0000 H	0
2	1000	16	3	2	0000 H	0
3	1000	16	3	2	00f9 H	415
4	1000	16	3	2	00f8 H	413
5	1000	16	3	2	00af H	292
6	1000	16	3	2	011b H	472
7	1000	16	3	2	0158 H	573
8	1000	16	3	2	0136 H	517
9	1000	16	3	2	0154 H	567
10	1000	16	3	2	0187 H	652
20	1000	16	3	2	01f3 H	832
30	1000	16	3	2	0225 H	915
40	1000	16	3	2	023f H	958
50	1000	16	3	2	024c H	980
60	1000	16	3	2	0250 H	987
70	1000	16	3	2	024d H	982
80	1000	16	3	2	0256 H	997
90	1000	16	3	2	025b H	1005
97	1000	16	3	2	0259 H	1002
98	1000	16	3	2	025A H	1003
99	546	16	3	2	0203 H	858
100	546	16	3	2	0195 H	675
110	546	16	3	2	0188 H	653
120	546	16	3	2	0164 H	593
130	546	16	3	2	0154 H	567
135	546	16	3	2	0154 H	567
140	546	16	3	2	014f H	558
150	546	16	3	2	014c H	553



Gambar 5.14 Grafik performansi Kontroler PID

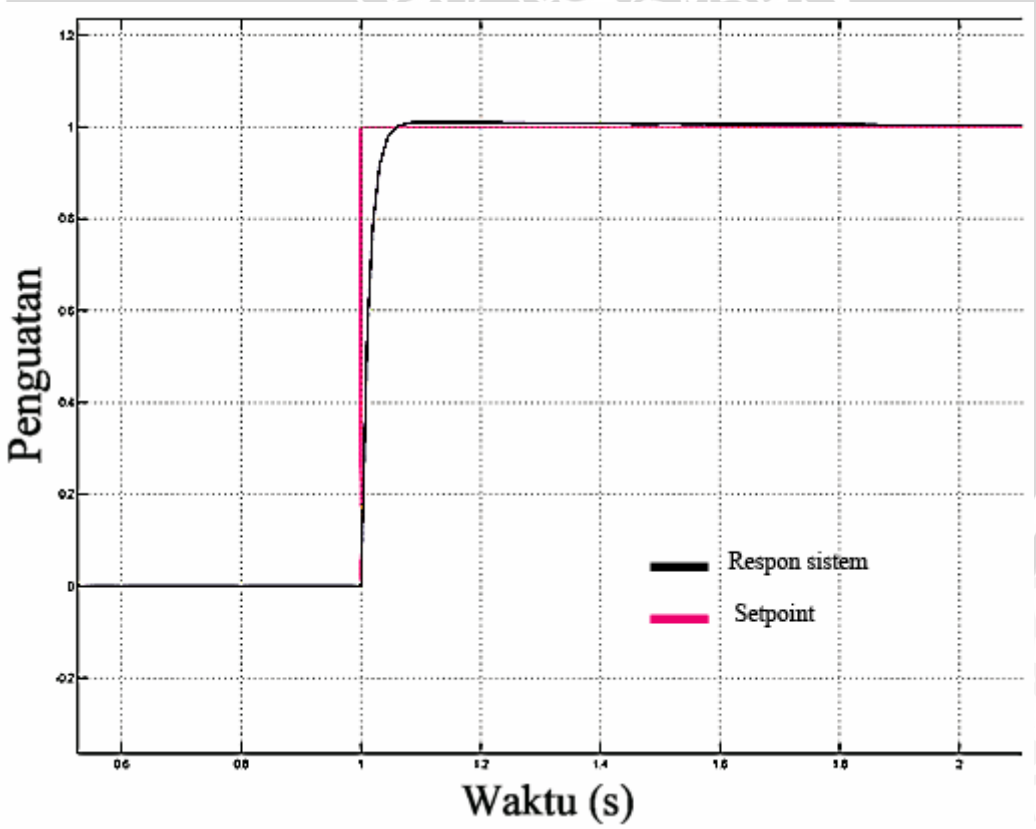
Gafik yang didapat dari hasil pengujian, menunjukkan bahwa sistem yang dikontrol merupakan sistem orde satu. Oleh karena itu, karakteristik dari sistem ini ditunjukkan oleh

nilai penguatan dan konstanta waktunya seperti pada Persamaan (5-1), dengan K adalah penguatannya dan T adalah konstanta waktunya. Nilai penguatan totalnya adalah dari 0 sampai 1000. Sementara konstanta waktu didapatkan dari 63,2% dari nilai maksimal. Itu berarti nilai K adalah 1000 sementara waktu yang berada pada saat kecepatannya 632 rpm adalah 13 detik. Itu berarti nilai T adalah 13 dikalikan dengan waktu sampling satu detik, menjadi 13 dikali satu detik yaitu 13. Sehingga persamaan untuk sistem padaa Gambar 5.14 adalah seperti pada Persamaan (5-2).

$$\frac{C(s)}{R(s)} = \frac{K}{Ts+1} \tag{5-1}$$

$$Gp(s) = \frac{1000}{13s+1} \tag{5-2}$$

Berdasarkan persamaan baru dari Persamaan (5-2), fungsi alih yang baru itu dimasukkan dalam program simulator Scilab. Dari hasil simulasi tersebut akan kita dapatkan grafik karakteristik yang baru, seperti pada Gambar 5.15. Dari grafik Gambar 5.15 tersebut, dapat kita lihat bahwa waktu yang dibutuhkan untuk mencapai waktu sampling tidak sampai 0,2 detik. Oleh sebab itu, sistem ini dapat dinyatakan baik.



Gambar 5.15 Respons fungsi pada *Scilab*.

BAB 6
PENUTUP

6.1 Kesimpulan

Berdasarkan perancangan, hasil simulasi, dan pengujian yang telah dilakukan selama penelitian ini, dapat ditarik kesimpulan:

1. Untuk mengimplementasikan kontroler PID 16 bit, dibutuhkan lima rangkaian penjumlah dan pengurang, lima rangkaian pengali, dua rangkaian pembagi, dua buah register, dan satu rangkaian untuk mengatur waktu perhitungan. Waktu yang dibutuhkan untuk satu kali perhitungan adalah 570 ns. Kontroler PID ini memiliki waktu sampling yang dapat diubah yaitu satu detik, 100 ms, 10 ms, dan 1 ms.
2. Rangkaian PWM 16 bit dalam penelitian ini memiliki keluaran sinyal PWM dengan frekuensi 762,941 Hz dan *duty-cycle* 0-100% yang dapat diubah-ubah.
3. Rangkaian pembaca sensor dalam penelitian ini memiliki kapasitas 16 bit, namun hanya dapat membaca sinyal masukan dengan rentang frekuensi antara 2 Hz-65 kHz.

6.2 Saran

Berikut ini adalah saran untuk penelitian berikutnya:

1. Mengganti sensor *rotary enkoder* yang digunakan dengan *rotary enkoder* yang memiliki resolusi lebih baik dan juga dapat membaca kecepatan di atas 2500 rpm. Hal ini akan memudahkan untuk memvariasikan waktu sampling kontroler PID. Namun, dengan mengganti sensor rotary enkoder yang digunakan, maka variabel pada kontroler PID pada bagian konversi dari banyaknya sinyal yang masuk menjadi kecepatan dalam satuan rpm juga perlu diganti.
2. Mengembangkan bagian perhitungan yang berhubungan dengan berubahnya waktu sampling seperti rentang kecepatan yang dapat dikontrol saat waktu samplingnya 1 ms, 10 ms, atau 100 ms. Hal ini dikarenakan pada penelitian ini, semua pengujiannya menggunakan waktu sampling satu detik sehingga untuk waktu sampling yang lain belum dibuktikan.
3. Mengembangkan bagian konversi dari PID menjadi PWM agar ketiga puluh dua bit data hasil perhitungan PWM dapat digunakan semua untuk mengontrol *duty-cycle* PWM.

DAFTAR PUSTAKA

Alfke, P., & New, B. (1997, October 27). Xilinx Aplication Note : Serial Code Conversion between BCD and Binary.

Åström, K. J. (2002). PID Control. In K. J. Åström, *Control System Design Lecture Notes for ME 155A* (pp. 216-251). Santa Barbara: Department of Mechanical and Environmental Engineering University of California Santa Barbara.

Astrom, K. J., & Hagglund, T. (1995). *PID Controllers: theory, design, and tuning, 2nd Edition.* Instrument Society of America.

Bennett, S. (1996, 3). A brief History of Automatic Control. *IEEE Control Systems Magazine (IEEE)*, pp. 17-25.

Bolton, W. (2003). *Mechatronics: Electronic Control Systems in Mechanical and Electrical Engineering Third Edition.* Harlow, England: Pearson, Prentice Hall.

Bolton, W. (2006). *Sistem Instrumentasi dan Sistem Kontrol.* Jakarta: Erlangga.

Chan, Y. F., Moallem, M., & Wang, W. (2007, AUGUST). Design and Implementation of Modular FPGA-Based PID Controllers. *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 54, NO. 4*, 1898-1906.

Depok-Instruments. (2011). DI-REV1 DI-Rotary Encoder Versi 1. Depok, Indonesia.

Farooq, U., Marrakchi, Z., & Mehrez, H. (2012). *Tree-based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization.* London: Springer Science & Business Media.

Franklin, G. F., Powell, J. D., & Workman, M. L. (1990). *Digital Control of Dynamic Systems 2nd Edition.* Massachusetts: Addison-Wesley Publishing Company.

Mabuchi-Motor. (2014). RS-445PA/PD. Matsudo City, Chiba, Japan.

Nouman, Z., Klima, B., & Knobloch, J. (2013, DECEMBER). Generating PWM Signals With Variable Duty From 0% to 100% Based FPGA SPARTAN3AN. *ElektroRevue VOL.4, NO.4*, 75-79.

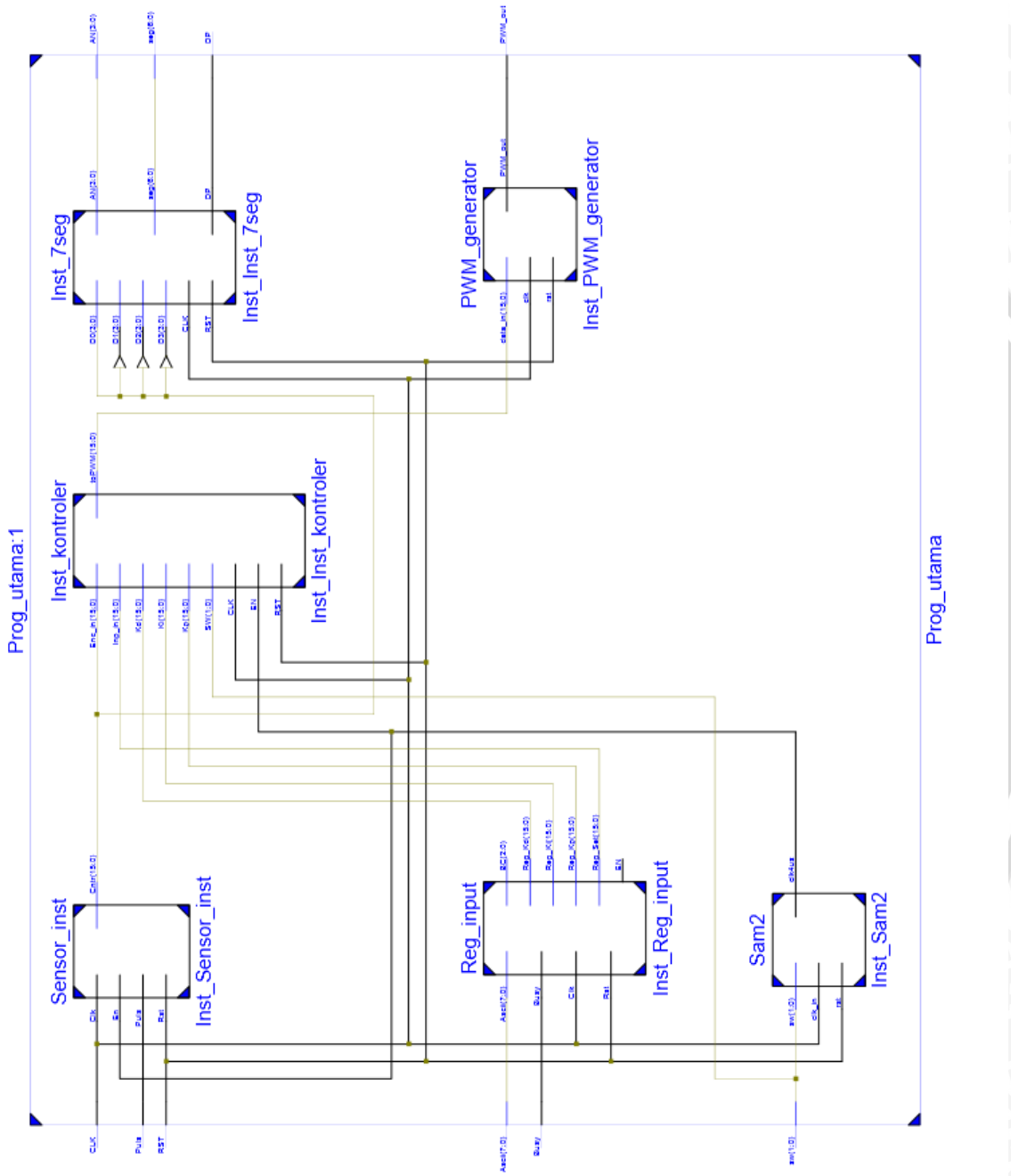
STMicroelectronics. (2000, January). L298 Dual Full-Bidge Driver. Italy.

Xilinx. (2011, Maret). LogiCORE IP Adder/Subtracter v11.0.

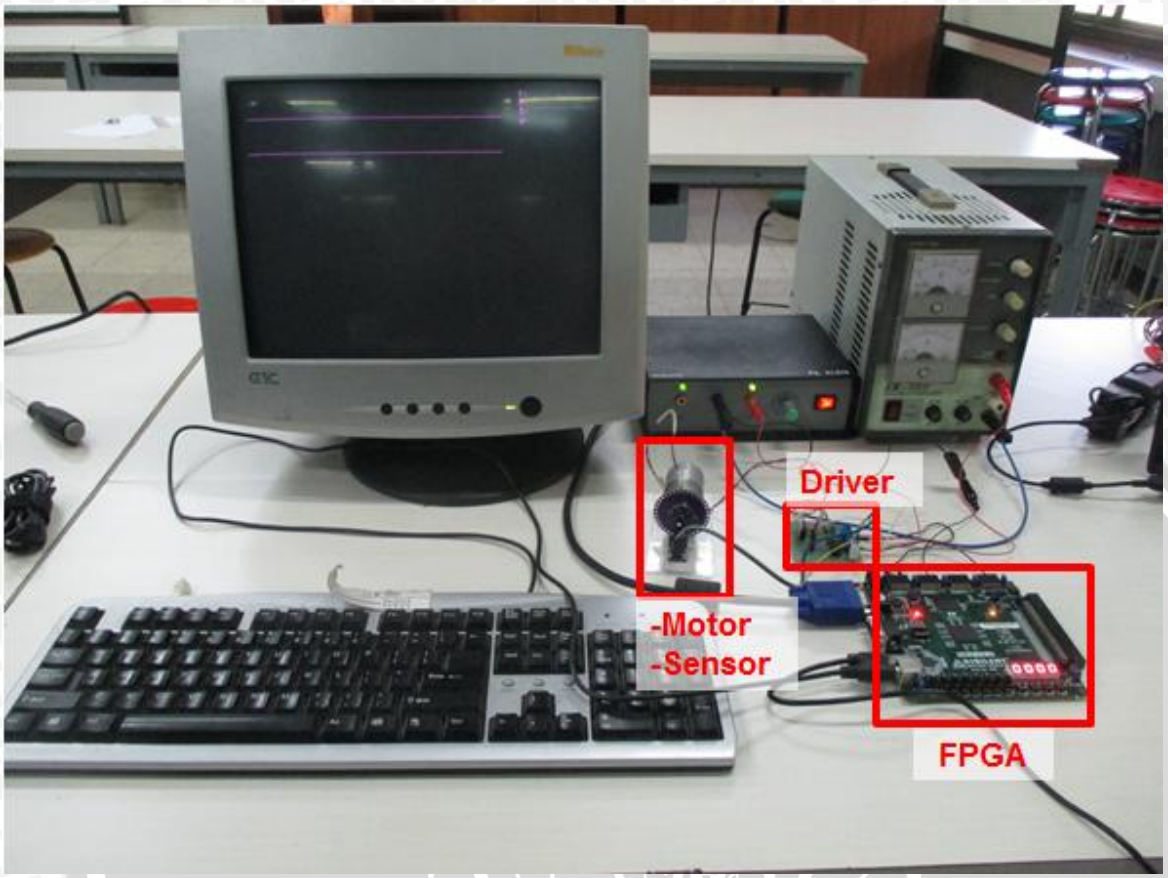
Xilinx. (2013). Language Templates, Info (State-Machine).

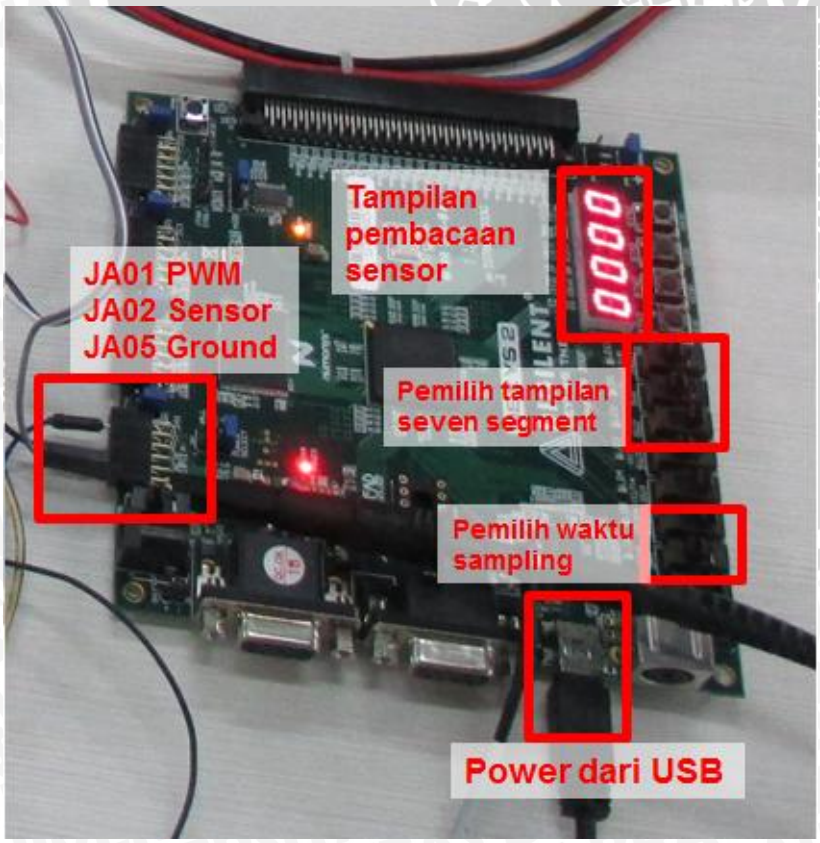
LAMPIRAN

Lampiran 1 Skema rangkaian keseluruhan



Lampiran 2 Dokumentasi hasil penelitian





Lampiran 3 Listing progam FPGA

1. Program Utama

```
----------------------------------------------------------------------
------------
-- Company:
-- Engineer:
--
-- Create Date:    13:36:27 06/29/2015
-- Design Name:
-- Module Name:    Pengontrol_Kecepatan - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Pengontrol_Kecepatan is
    Port ( Clk    : in  STD_LOGIC;
           Rst    : in  STD_LOGIC;
           Busy   : in  STD_LOGIC;
           Puls   : IN std_logic;
           Ascii : IN STD_LOGIC_VECTOR (7 downto 0);
           SW     : IN std_logic_vector (1 downto 0);
           sp     : in STD_LOGIC_VECTOR (2 downto 0);
           AN     : OUT std_logic_vector(3 downto 0);
           seg    : OUT std_logic_vector(6 downto 0);
           DP     : OUT std_logic;
           PWM_out : OUT std_logic);
end Pengontrol_Kecepatan;

architecture Behavioral of Pengontrol_Kecepatan is
------------ SIGNAL ----------------------------------------
signal sRs, sRp, sRi, sRd, sSn, sPWM :
STD_LOGIC_VECTOR (15 downto 0);
signal sd0, sd1, sd2, sd3 : STD_LOGIC_VECTOR (3
downto 0);
signal sEN : std_logic;
------------ COMP_TAG ----------------------------------
COMPONENT Inst_Input
  PORT(
    Clk : IN std_logic;
    Rst : IN std_logic;
    Ascii : IN std_logic_vector(7 downto 0);
    Busy : IN std_logic;
    RegS : OUT std_logic_vector(15 downto 0);
    RegP : OUT std_logic_vector(15 downto 0);
    RegI : OUT std_logic_vector(15 downto 0);
    RegD : OUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;
COMPONENT Inst_kontroler
  PORT(
    CLK : IN std_logic;
    RST : IN std_logic;
    STR : IN std_logic;
    Kp : IN std_logic_vector(15 downto 0);
    Ki : IN std_logic_vector(15 downto 0);
    Kd : IN std_logic_vector(15 downto 0);
    Inp_in : IN std_logic_vector(15 downto 0);
    Enc_in : IN std_logic_vector(15 downto 0);
    SW : IN std_logic_vector(1 downto 0);
    toPWM : OUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;
COMPONENT PWM_generator
  PORT(
    clk : IN std_logic;
    rst : IN std_logic;
    data_in : IN std_logic_vector(15 downto 0);
    PWM_out : OUT std_logic
    );
  END COMPONENT;
COMPONENT Sensor_inst
  PORT(
    Clk : IN std_logic;
    Rst : IN std_logic;
    Puls : IN std_logic;
    En : IN std_logic;
    Cntr : OUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;
COMPONENT Sam2
  PORT(
    clk_in : IN std_logic;
    rst : IN std_logic;
    sw : IN std_logic_vector(1 downto 0);
    clk4us : OUT std_logic
    );
  END COMPONENT;
COMPONENT Inst_7seg
  PORT(
    CLK : IN std_logic;
    RST : IN std_logic;
    D0 : IN std_logic_vector(3 downto 0);
    D1 : IN std_logic_vector(3 downto 0);
    D2 : IN std_logic_vector(3 downto 0);
    D3 : IN std_logic_vector(3 downto 0);
    AN : OUT std_logic_vector(3 downto 0);
    seg : OUT std_logic_vector(6 downto 0);
    DP : OUT std_logic
    );
  END COMPONENT;
COMPONENT mux_tampilan
  PORT(
    sp : IN std_logic_vector(2 downto 0);
    set : IN std_logic_vector(15 downto 0);
    kp : IN std_logic_vector(15 downto 0);
    ki : IN std_logic_vector(15 downto 0);
    kd : IN std_logic_vector(15 downto 0);
    Puls : IN std_logic_vector(15 downto 0);
```

```
    d0 : OUT std_logic_vector(3 downto 0);
    d1 : OUT std_logic_vector(3 downto 0);
    d2 : OUT std_logic_vector(3 downto 0);
    d3 : OUT std_logic_vector(3 downto 0)
    );
  END COMPONENT;

-- COMP_TAG_END ------------------------------------------
--

begin
------------ INST_TAG ----------------------------------
Inst_Inst_Input: Inst_Input PORT MAP(
    Clk => clk,
    Rst => rst,
    Ascii => ascii,
    Busy => busy,
    RegS => sRs,
    RegP => sRp,
    RegI => sRi,
    RegD => sRd
  );
Inst_Inst_kontroler: Inst_kontroler PORT MAP(
    CLK => clk,
    RST => rst,
    STR => sEN,
    Kp => sRp,
    Ki => sRi,
    Kd => sRd,
    Inp_in => sRs,
    Enc_in => sSn,
    SW => sw,
    toPWM => sPWM
  );
Inst_PWM_generator: PWM_generator PORT MAP(
    clk => clk,
    rst => rst,
    data_in => sPWM,
    PWM_out => PWM_out
  );
Inst_Sensor_inst: Sensor_inst PORT MAP(
    Clk => clk,
    Rst => rst,
    Puls => Puls,
    En => sEn,
    Cntr => sSN
  );
Inst_Sam2: Sam2 PORT MAP(
    clk_in => clk,
    rst => rst,
    sw => sw,
    clk4us => sEn
  );
Inst_Inst_7seg: Inst_7seg PORT MAP(
    CLK => clk,
    RST => rst,
    D0 => sd0,
    D1 => sD1,
    D2 => sD2,
    D3 => sD3,
    AN => an,
    seg => seg,
    DP => dp
  );
Inst_mux_tampilan: mux_tampilan PORT MAP(
    sp => sp,
    set => sRs,
    kp => SRp,
    ki => SRi,
    kd => SRd,
    Puls => sSn,
    d0 => sD0,
    d1 => sD1,
    d2 => sD2,
    d3 => sD3
  );
-- INST_TAG_END ------------------------------------------
-
end Behavioral;
```

2. Input Interface

```
----------------------------------------------------------------------------
------------
-- Company:
-- Engineer:
--
-- Create Date:    10:25:08 06/12/2015
-- Design Name:
-- Module Name:    Inst_Input - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Inst_Input is
    Port ( Clk : in  STD_LOGIC;
           Rst : in  STD_LOGIC;
           Ascii : in  STD_LOGIC_VECTOR (7 downto 0);
           Busy  : in  STD_LOGIC;
           RegS : OUT std_logic_vector(15 downto 0);
           RegP : OUT std_logic_vector(15 downto 0);
           RegI : OUT std_logic_vector(15 downto 0);
           RegD : OUT std_logic_vector(15 downto 0));
end Inst_Input;

architecture Behavioral of Inst_Input is
------------ SIGNAL ------------------------------------
signal sR1,sR2,sR3,sR4 : STD_LOGIC_VECTOR (3
downto 0);
signal shp : std_logic_vector(3 downto 0);
signal sbp : std_logic_vector(2 downto 0);
```

```
signal scp : std_logic_vector(1 downto 0);
signal sDat16,sDat : STD_LOGIC_VECTOR (15
downto 0);
------------ COMP_TAG ----------------------------------
COMPONENT Read_ASCII
  PORT(
    Clk : IN std_logic;
    Rst : IN std_logic;
    Busy : IN std_logic;
    Ascii : IN std_logic_vector(7 downto 0);
    hc : OUT std_logic_vector(3 downto 0);
    bc : OUT std_logic_vector(2 downto 0);
    cc : OUT std_logic_vector(1 downto 0);
    R1 : out  STD_LOGIC_VECTOR (3 downto 0);
    R2 : out  STD_LOGIC_VECTOR (3 downto 0);
    R3 : out  STD_LOGIC_VECTOR (3 downto 0);
    R4 : out  STD_LOGIC_VECTOR (3 downto 0)
--    Reg_S : OUT std_logic_vector(15 downto 0);
--    Reg_P : OUT std_logic_vector(15 downto 0);
--    Reg_I : OUT std_logic_vector(15 downto 0);
--    Reg_D : OUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;
--COMPONENT BCD2BIN
--PORT(
--    clk : IN std_logic;
--    rst : IN std_logic;
--    busy : IN std_logic;
--    bc : IN std_logic_vector(2 downto 0);
--    Rs : IN std_logic_vector(15 downto 0);
--    Rp : IN std_logic_vector(15 downto 0);
--    Ri : IN std_logic_vector(15 downto 0);
--    Rd : IN std_logic_vector(15 downto 0);
--    RegO : OUT std_logic_vector(15 downto 0)
--    );
--END COMPONENT;
COMPONENT BCDtoBin
  PORT(
    CLK : IN std_logic;
    RST : IN std_logic;
    Busy : IN std_logic;
    bcd4 : IN std_logic_vector(3 downto 0);
    bcd3 : IN std_logic_vector(3 downto 0);
    bcd2 : IN std_logic_vector(3 downto 0);
    bcd1 : IN std_logic_vector(3 downto 0);
    DAT : OUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;
--COMPONENT mux_in
--PORT(
--    clk : IN std_logic;
--    rst : IN std_logic;
--    bc : IN std_logic_vector(2 downto 0);
--    Rs : IN std_logic_vector(15 downto 0);
--    Rp : IN std_logic_vector(15 downto 0);
--    Ri : IN std_logic_vector(15 downto 0);
--    Rd : IN std_logic_vector(15 downto 0);
--    toBin : OUT std_logic_vector(15 downto 0)
--    );
--END COMPONENT;
COMPONENT Reg_keluar
  PORT(
    Clk : IN std_logic;
    Rst : IN std_logic;
    busy : IN std_logic;
    bc : IN std_logic_vector(2 downto 0);
    cc : IN std_logic_vector(1 downto 0);
    Data : IN std_logic_vector(15 downto 0);
    Reg_S : OUT std_logic_vector(15 downto 0);
    Reg_P : OUT std_logic_vector(15 downto 0);
    Reg_I : OUT std_logic_vector(15 downto 0);
    Reg_D : OUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;
-- COMP_TAG_END ------------------------------------------
--

begin
------------ INST_TAG ---------------------------------
Inst_Read_ASCII: Read_ASCII PORT MAP(
    Clk  => clk,
    Rst  => rst,
    Busy => busy,
    Ascii => ascii,
    hc => shp,
    bc => sbp,
    cc => scp,
    R1 => sR1,
    R2 => sR2,
    R3 => sR3,
    R4 => sR4
  );
--Inst_BCD2BIN: BCD2BIN PORT MAP(
--    clk => clk,
--    rst => rst,
--    busy => busy,
--    bc => sbp,
--    Rs => sR1,
--    Rp => sR2,
--    Ri => sR3,
--    Rd => sR4,
--    RegO => sDat16
--);
Inst_BCDtoBin: BCDtoBin PORT MAP(
    CLK => clk,
    RST => rst,
    Busy => busy,
    bcd4 => sR4,
    bcd3 => sR3,
    bcd2 => sR2,
    bcd1 => sR1,
    DAT => sDat16
  );
--Inst_mux_in: mux_in PORT MAP(
--    clk => clk,
--    rst => rst,
--    bc => sbp,
--    Rs => sR1,
--    Rp => sR2,
--    Ri => sR3,
--    Rd => sR4,
--    toBin => sDat
--);
Inst_Reg_keluar: Reg_keluar PORT MAP(
    Clk => clk,
    Rst => rst,
    Busy => busy,
--    hc => shp,
    bc => sbp,
    cc => scp,
    Data => sDat16,
    Reg_S => RegS,
    Reg_P => RegP,
    Reg_I => RegI,
```

```
    Reg_D => RegD
  );
-- INST_TAG_END ------------------------------------
-
end Behavioral;

----------------------------------------------------------------
------------
-- Company:
-- Engineer:
--
-- Create Date:    15:24:47 06/15/2015
-- Design Name:
-- Module Name:    Read_ASCII - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------
------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Read_ASCII is
   Port ( Clk, Rst, Busy : in  STD_LOGIC;
          Ascii : in  STD_LOGIC_VECTOR (7 downto 0);
          hc : out  STD_LOGIC_VECTOR (3 downto 0);
          bc : out  STD_LOGIC_VECTOR (2 downto 0);
          cc : out  STD_LOGIC_VECTOR (1 downto 0);
          R1 : out  STD_LOGIC_VECTOR (3 downto 0);
          R2 : out  STD_LOGIC_VECTOR (3 downto 0);
          R3 : out  STD_LOGIC_VECTOR (3 downto 0);
          R4 : out  STD_LOGIC_VECTOR (3 downto 0));
--          Reg_S : out STD_LOGIC_VECTOR (15
downto 0);
--          Reg_P : out STD_LOGIC_VECTOR (15
downto 0);
--          Reg_I : out STD_LOGIC_VECTOR (15 downto
0);
--          Reg_D : out STD_LOGIC_VECTOR (15
downto 0));
end Read_ASCII;

architecture Behavioral of Read_ASCII is
signal Rp,Rn, R1p,R1n, R2p,R2n, R3p,R3n, R4p,R4n :
std_logic_vector(3 downto 0) := (others => '0');
signal Rsp,Rpp,Rip,Rdp, Rsn,Rpn,Rin,Rdn :
std_logic_vector(15 downto 0) := (others => '0');

signal hp, hn : std_logic_vector(3 downto 0) := (others
=> '0'); --hurup
signal bp, bn : std_logic_vector(2 downto 0) := (others
=> '0'); --baris
signal cp, cn : std_logic_vector(1 downto 0) := (others
=> '0'); --kode
signal busy_p : std_logic; --, busy_n

begin
----- MEM --------------------------------
process(clk, rst)
  begin
    if rst = '1' then
      hp <= (others => '0');
      bp <= (others => '0');
      cp <= (others => '0');
      Rp <= (others => '0');
      R1p <= (others => '0');
      R2p <= (others => '0');
      R3p <= (others => '0');
      R4p <= (others => '0');
      Rsp <= (others => '0');
      Rpp <= (others => '0');
      Rip <= (others => '0');
      Rdp <= (others => '0');
      busy_p <= '0';
    elsif clk'event and clk = '0' then
      hp  <= hn;
      bp  <= bn;
      cp  <= cn;
      Rp  <= Rn;
      R1p <= R1n;
      R2p <= R2n;
      R3p <= R3n;
      R4p <= R4n;
      Rsp <= Rsn;
      Rpp <= Rpn;
      Rip <= Rin;
      Rdp <= Rdn;
      busy_p <= busy;
    end if;
  end process;
----- konversi ascii to biner ------------------------
process(busy_p, hp, bp, ascii, R1p) --, cntp , hp, bp,
ascii,, busy
begin
  if(busy_p'event and busy_p = '0') then
    case (ascii) is
    when "00110000" => Rn <= x"0"; cn <= "00";
      if hp = 10 then
        hn <= hp;
      else
        hn <= hp+1;
      end if;
    when "00110001" => Rn <= x"1"; cn <= "00";
      if hp = 10 then
        hn <= hp;
      else
        hn <= hp+1;
      end if;
    when "00110010" => Rn <= x"2"; cn <= "00";
      if hp = 10 then
        hn <= hp;
      else
        hn <= hp+1;
      end if;
    when "00110011" => Rn <= x"3"; cn <= "00";
```

```
        if hp = 10 then
          hn <= hp;
        else
          hn <= hp+1;
        end if;
      when "00110100" => Rn <= x"4"; cn <= "00";
        if hp = 10 then
          hn <= hp;
        else
          hn <= hp+1;
        end if;
      when "00110101" => Rn <= x"5"; cn <= "00";
        if hp = 10 then
          hn <= hp;
        else
          hn <= hp+1;
        end if;
      when "00110110" => Rn <= x"6"; cn <= "00";
        if hp = 10 then
          hn <= hp;
        else
          hn <= hp+1;
        end if;
      when "00110111" => Rn <= x"7"; cn <= "00";
        if hp = 10 then
          hn <= hp;
        else
          hn <= hp+1;
        end if;
      when "00111000" => Rn <= x"8"; cn <= "00";
        if hp = 10 then
          hn <= hp;
        else
          hn <= hp+1;
        end if;
      when "00111001" => Rn <= x"9"; cn <= "00";
        if hp = 10 then
          hn <= hp;
        else
          hn <= hp+1;
        end if;
      when "00001000" => Rn <= Rp; cn <= "01"; --
backspace
        if hp = 0 then
          hn <= hp;
        else
          hn <= hp - 1;
        end if;
      when "00011000" => --UP
        hn <= (others => '0'); Rn <= (others => '0'); cn
<= "10";
        if bp = 0 then
          bn <= bp;
        else
          bn <= bp - 1;
        end if;
      when "00011001" => --DOWN
        hn <= (others => '0'); Rn <= (others => '0'); cn
<= "10";
        if bp = 6 then
          bn <= bp;
        else
          bn <= bp + 1;
        end if;
      when "00001101" => Rn <= Rp; cn <= "11";
        if hp < 10 then
          hn <= "1010";
        else
          hn <= hp; --enter
        end if;
      when others =>
        Rn <= Rp; bn <= bp; hn <= hp; cn <= "00";
    end case;
  end if;
end process;
----- masuk 4 register ------------------------------------
process (hp, Rp,R1p,R2p,R3p,R4p, cp, busy_p,
Rsp,Rpp,Rip,Rdp)
begin
if(busy_p'event and busy_p = '0') then
  case (hp) is
    when "0000" =>
      if cp = "10" then
      -------- bila panah maka data sebelumnya diambil --
--
        case (bp) is
          when "000" =>
            R1n <= Rsp (3 downto 0); R2n <= Rsp (7
downto 4); R3n <= Rsp (11 downto 8); R4n <= Rsp (15
downto 12);
          when "010" =>
            R1n <= Rpp (3 downto 0); R2n <= Rpp (7
downto 4); R3n <= Rpp (11 downto 8); R4n <= Rpp (15
downto 12);
          when "100" =>
            R1n <= Rip (3 downto 0); R2n <= Rip (7
downto 4); R3n <= Rip (11 downto 8); R4n <= Rip (15
downto 12);
          when "110" =>
            R1n <= Rdp (3 downto 0); R2n <= Rdp (7
downto 4); R3n <= Rdp (11 downto 8); R4n <= Rdp (15
downto 12);
          when others =>
            R1n <= R1p; R2n <= R2p; R3n <= R3p; R4n
<= R4p;
        end case;
        ---------------------------------------------------
      else
        R1n <= R1p; R2n <= R2p; R3n <= R3p; R4n <=
R4p;
      end if;
    when "0010" =>
      if cp = "00" then
        R1n <= Rp; R2n <= R1p; R3n <= R2p; R4n <=
R3p;
      elsif cp = "01" then
        R1n <= R2p; R2n <= R3p; R3n <= R4p; R4n <=
(others => '0');
      else
        R1n <= R1p; R2n <= R2p; R3n <= R3p; R4n <=
R4p;
      end if;
    when "0100" =>
      if cp = "00" then
        R1n <= Rp; R2n <= R1p; R3n <= R2p; R4n <=
R3p;
      elsif cp = "01" then
        R1n <= R2p; R2n <= R3p; R3n <= R4p; R4n <=
(others => '0');
      else
        R1n <= R1p; R2n <= R2p; R3n <= R3p; R4n <=
R4p;
      end if;
    when "0110" =>
```

```
        if cp = "00" then
          R1n <= Rp; R2n <= R1p; R3n <= R2p; R4n <=
R3p;
        elsif cp = "01" then
          R1n <= R2p; R2n <= R3p; R3n <= R4p; R4n <=
(others => '0');
        else
          R1n <= R1p; R2n <= R2p; R3n <= R3p; R4n <=
R4p;
        end if;
      when "1000" =>
        if cp = "00" then
          R1n <= Rp; R2n <= R1p; R3n <= R2p; R4n <=
R3p;
        elsif cp = "01" then
          R1n <= R2p; R2n <= R3p; R3n <= R4p; R4n <=
(others => '0');
        else
          R1n <= R1p; R2n <= R2p; R3n <= R3p; R4n <=
R4p;
        end if;
      when "1010" =>
        if cp = "00" then
          R1n <= R1p; R2n <= R2p; R3n <= R3p; R4n <=
R4p;
        elsif cp = "01" then
          R1n <= R2p; R2n <= R3p; R3n <= R4p; R4n <=
(others => '0');
        else
          R1n <= R1p; R2n <= R2p; R3n <= R3p; R4n <=
R4p;
        end if;
      when others =>
        R1n <= R1p; R2n <= R2p; R3n <= R3p; R4n <=
R4p;
    end case;
  end if;
  end process;
  ----- pilih reg out ----------------------
  process(busy_p, bp, cp, Rsp,Rpp,Rip,Rdp)
  begin
  if(busy_p'event and busy_p = '0') then
    case (bp) is
      when "000" =>
          Rsn (3 downto 0) <= R1p; Rsn (7 downto 4) <=
R2p; Rsn (11 downto 8) <= R3p; Rsn (15 downto 12) <=
R4p;
          Rpn <= Rpp; Rin <= Rip; Rdn <= Rdp;
      when "010" =>
          Rpn (3 downto 0) <= R1p; Rpn (7 downto 4) <=
R2p; Rpn (11 downto 8) <= R3p; Rpn (15 downto 12) <=
R4p;
          Rsn <= Rsp; Rin <= Rip; Rdn <= Rdp;
      when "100" =>
          Rin (3 downto 0) <= R1p; Rin (7 downto 4) <=
R2p; Rin (11 downto 8) <= R3p; Rin (15 downto 12) <=
R4p;
          Rsn <= Rsp; Rpn <= Rpp; Rdn <= Rdp;
      when "110" =>
          Rdn (3 downto 0) <= R1p; Rdn (7 downto 4) <=
R2p; Rdn (11 downto 8) <= R3p; Rdn (15 downto 12) <=
R4p;
          Rsn <= Rsp; Rpn <= Rpp; Rin <= Rip;
      when others =>
        Rsn <= Rsp; Rpn <= Rpp; Rin <= Rip; Rdn <=
Rdp;
    end case;
  end if;
end process;
----- out ------------------------------
process (hp, bp, cp, R1p,R2p,R3p,R4p)
  begin
    hc <= hp;
    bc <= bp;
    cc <= cp;
--  case (bp) is
--    when "000" =>
--      DAT <= Rsp;
--    when "010" =>
--      DAT <= Rpp;
--    when "100" =>
--      DAT <= Rip;
--    when "110" =>
--      DAT <= Rdp;
--    when others =>
--      DAT <= Rsp;
--  end case;
    R1 <= R1p;
    R2 <= R2p;
    R3 <= R3p;
    R4 <= R4p;

  end process;
end Behavioral;
----------------------------------------------------------------------
------------
-- Company:
-- Engineer:
--
-- Create Date:    23:11:51 06/28/2015
-- Design Name:
-- Module Name:    BCDtoBin - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity BCDtoBin is
   Port ( CLK : in  STD_LOGIC;
          RST : in  STD_LOGIC;
          Busy : in  STD_LOGIC;
          bcd4 : in  STD_LOGIC_VECTOR (3 downto 0);
```

```
        bcd3 : in  STD_LOGIC_VECTOR (3 downto 0);
        bcd2 : in  STD_LOGIC_VECTOR (3 downto 0);
        bcd1 : in  STD_LOGIC_VECTOR (3 downto 0);
        DAT : out  STD_LOGIC_VECTOR (15 downto
0));
end BCDtoBin;

architecture Behavioral of BCDtoBin is
signal bcd4s_p,bcd3s_p,bcd2s_p,bcd1s_p,
bcd4s_n,bcd3s_n,bcd2s_n,bcd1s_n :
STD_LOGIC_VECTOR (3 downto 0);
signal bussy_n, bussy_p : std_logic;
signal cnt_n, cnt_p : unsigned(4 downto 0);
signal Bin_p, bin_n : STD_LOGIC_VECTOR(15
downto 0);

begin
---- Memory ----------------------------------------------------
---
  process (rst, clk)
    begin
     if rst = '1' then
      bcd1s_p <= (others => '0');
      bcd2s_p <= (others => '0');
      bcd3s_p <= (others => '0');
      bcd4s_p <= (others => '0');
      cnt_p <= (others => '0');
      bin_p <= (others => '0');
      bussy_p <= '0';
     elsif (clk'event and clk='0') then
      bcd4s_p(3) <= '0';
      bcd4s_p(2) <= bcd4s_n(3);
      bcd4s_p(1) <= bcd4s_n(2);
      bcd4s_p(0) <= bcd4s_n(1);
      -----
      bcd3s_p(3) <= bcd4s_n(0);
      bcd3s_p(2) <= bcd3s_n(3);
      bcd3s_p(1) <= bcd3s_n(2);
      bcd3s_p(0) <= bcd3s_n(1);
      -----
      bcd2s_p(3) <= bcd3s_n(0);
      bcd2s_p(2) <= bcd2s_n(3);
      bcd2s_p(1) <= bcd2s_n(2);
      bcd2s_p(0) <= bcd2s_n(1);
      -----
      bcd1s_p(3) <= bcd2s_n(0);
      bcd1s_p(2) <= bcd1s_n(3);
      bcd1s_p(1) <= bcd1s_n(2);
      bcd1s_p(0) <= bcd1s_n(1);
      -----
      cnt_p <= cnt_n;
      bussy_p <= bussy_n;
      bin_p <= bin_n;
     end if;
  end process;
---- Kombinasional ---------------------------------------------
---
  process (bcd4,bcd3,bcd2,bcd1,
bcd1s_p,bcd2s_p,bcd3s_p,bcd4s_p, busy,bussy_p, cnt_p,
bin_p)
  begin
    cnt_n <= cnt_p+1;
    bussy_n <= busy;
--  bin_n <= bin_p;
    -----
    bcd4s_n <= bcd4s_p;
    -----
    case (bcd3s_p) is
      when "1000" =>
        bcd3s_n(3) <= '0'; bcd3s_n(2) <= '1'; bcd3s_n(1)
<= '0'; bcd3s_n(0) <= '1';
      when "1001" =>
        bcd3s_n(3) <= '0'; bcd3s_n(2) <= '1'; bcd3s_n(1)
<= '1'; bcd3s_n(0) <= '0';
      when "1010" =>
        bcd3s_n(3) <= '0'; bcd3s_n(2) <= '1'; bcd3s_n(1)
<= '1'; bcd3s_n(0) <= '1';
      when "1011" =>
        bcd3s_n(3) <= '1'; bcd3s_n(2) <= '0'; bcd3s_n(1)
<= '0'; bcd3s_n(0) <= '0';
      when "1100" =>
        bcd3s_n(3) <= '1'; bcd3s_n(2) <= '0'; bcd3s_n(1)
<= '0'; bcd3s_n(0) <= '1';
      when "1101" =>
        bcd3s_n(3) <= '1'; bcd3s_n(2) <= '0'; bcd3s_n(1)
<= '1'; bcd3s_n(0) <= '0';
      when "1110" =>
        bcd3s_n(3) <= '1'; bcd3s_n(2) <= '0'; bcd3s_n(1)
<= '1'; bcd3s_n(0) <= '1';
      when "1111" =>
        bcd3s_n(3) <= '1'; bcd3s_n(2) <= '1'; bcd3s_n(1)
<= '0'; bcd3s_n(0) <= '0';
      when others =>
        bcd3s_n <= bcd3s_p;
    end case;
    -----
    case (bcd2s_p) is
      when "1000" =>
        bcd2s_n(3) <= '0'; bcd2s_n(2) <= '1'; bcd2s_n(1)
<= '0'; bcd2s_n(0) <= '1';
      when "1001" =>
        bcd2s_n(3) <= '0'; bcd2s_n(2) <= '1'; bcd2s_n(1)
<= '1'; bcd2s_n(0) <= '0';
      when "1010" =>
        bcd2s_n(3) <= '0'; bcd2s_n(2) <= '1'; bcd2s_n(1)
<= '1'; bcd2s_n(0) <= '1';
      when "1011" =>
        bcd2s_n(3) <= '1'; bcd2s_n(2) <= '0'; bcd2s_n(1)
<= '0'; bcd2s_n(0) <= '0';
      when "1100" =>
        bcd2s_n(3) <= '1'; bcd2s_n(2) <= '0'; bcd2s_n(1)
<= '0'; bcd2s_n(0) <= '1';
      when "1101" =>
        bcd2s_n(3) <= '1'; bcd2s_n(2) <= '0'; bcd2s_n(1)
<= '1'; bcd2s_n(0) <= '0';
      when "1110" =>
        bcd2s_n(3) <= '1'; bcd2s_n(2) <= '0'; bcd2s_n(1)
<= '1'; bcd2s_n(0) <= '1';
      when "1111" =>
        bcd2s_n(3) <= '1'; bcd2s_n(2) <= '1'; bcd2s_n(1)
<= '0'; bcd2s_n(0) <= '0';
      when others =>
        bcd2s_n <= bcd2s_p;
    end case;
    -----
    case (bcd1s_p) is
      when "1000" =>
        bcd1s_n(3) <= '0'; bcd1s_n(2) <= '1'; bcd1s_n(1)
<= '0'; bcd1s_n(0) <= '1';
      when "1001" =>
        bcd1s_n(3) <= '0'; bcd1s_n(2) <= '1'; bcd1s_n(1)
<= '1'; bcd1s_n(0) <= '0';
      when "1010" =>
```

```
            bcd1s_n(3) <= '0'; bcd1s_n(2) <= '1'; bcd1s_n(1)
<= '1'; bcd1s_n(0) <= '1';
          when "1011" =>
            bcd1s_n(3) <= '1'; bcd1s_n(2) <= '0'; bcd1s_n(1)
<= '0'; bcd1s_n(0) <= '0';
          when "1100" =>
            bcd1s_n(3) <= '1'; bcd1s_n(2) <= '0'; bcd1s_n(1)
<= '0'; bcd1s_n(0) <= '1';
          when "1101" =>
            bcd1s_n(3) <= '1'; bcd1s_n(2) <= '0'; bcd1s_n(1)
<= '1'; bcd1s_n(0) <= '0';
          when "1110" =>
            bcd1s_n(3) <= '1'; bcd1s_n(2) <= '0'; bcd1s_n(1)
<= '1'; bcd1s_n(0) <= '1';
          when "1111" =>
            bcd1s_n(3) <= '1'; bcd1s_n(2) <= '1'; bcd1s_n(1)
<= '0'; bcd1s_n(0) <= '0';
          when others =>
            bcd1s_n <= bcd1s_p;
      end case;
      -----
      bin_n(15) <= bcd1s_n(0);
      bin_n(14) <= bin_p(15);
      bin_n(13) <= bin_p(14);
      bin_n(12) <= bin_p(13);
      bin_n(11) <= bin_p(12);
      bin_n(10) <= bin_p(11);
      bin_n(9) <= bin_p(10);
      bin_n(8) <= bin_p(9);
      bin_n(7) <= bin_p(8);
      bin_n(6) <= bin_p(7);
      bin_n(5) <= bin_p(6);
      bin_n(4) <= bin_p(5);
      bin_n(3) <= bin_p(4);
      bin_n(2) <= bin_p(3);
      bin_n(1) <= bin_p(2);
      bin_n(0) <= bin_p(1);
      ------------------------------------------------------------
      if busy = '1' and bussy_p = '0' then
        cnt_n <= (others => '0');
        bin_n <= (others => '0');
        bcd4s_n <= bcd4;
        bcd3s_n <= bcd3;
        bcd2s_n <= bcd2;
        bcd1s_n <= bcd1;
--    elsif cnt_p = "00000" then
--      bcd1s_n <= bcd1s_p;
--      bcd2s_n <= bcd2s_p;
--      bcd3s_n <= bcd3s_p;
--      bcd4s_n <= bcd4s_p;
      elsif cnt_p = 15 then
        cnt_n <= cnt_p;
        bcd1s_n <= bcd1s_p;
        bcd2s_n <= bcd2s_p;
        bcd3s_n <= bcd3s_p;
        bcd4s_n <= bcd4s_p;
        bin_n <= bin_p;
      end if;
      ------------------------------------------------------------
      -- output ------------------------------------------------
      DAT <= bin_p;
    end process;
end Behavioral;

----------------------------------------------------------------------
------------
-- Company:
-- Engineer:
--
-- Create Date:    22:32:22 06/15/2015
-- Design Name:
-- Module Name:    Reg_keluar - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Reg_keluar is
    Port ( Clk : in  STD_LOGIC;
           Rst, busy : in  STD_LOGIC;
--           hc  : in  STD_LOGIC_VECTOR (3 downto 0);
           bc  : in  STD_LOGIC_VECTOR (2 downto 0);
           cc  : in  STD_LOGIC_VECTOR (1 downto 0);
           Data : in STD_LOGIC_VECTOR (15 downto 0);
           Reg_S : out STD_LOGIC_VECTOR (15 downto
0);
           Reg_P : out STD_LOGIC_VECTOR (15 downto
0);
           Reg_I : out STD_LOGIC_VECTOR (15 downto
0);
           Reg_D : out STD_LOGIC_VECTOR (15 downto
0));
end Reg_keluar;

architecture Behavioral of Reg_keluar is
--signal hp : std_logic_vector(3 downto 0); --hurup
signal bp : std_logic_vector(2 downto 0); --baris
signal cp : std_logic_vector(1 downto 0); --kode
signal Rp, Rsp,Rpp,Rip,Rdp, Rsn,Rpn,Rin,Rdn :
std_logic_vector(15 downto 0);
signal busy_p : std_logic;

begin
----- MEM --------------------------------
process(clk, rst)
  begin
    if rst = '1' then
--      hp <= (others => '0');
        bp <= (others => '0');
        cp <= (others => '0');
        Rp <= (others => '0');
        Rsp <= (others => '0');
```

```
        Rpp <= (others => '0');
        Rip <= (others => '0');
        Rdp <= (others => '0');
        busy_p <= '0';
      elsif clk'event and clk = '0' then
--        hp  <= hc;
        bp  <= bc;
        cp  <= cc;
        Rp  <= Data;
        Rsp <= Rsn;
        Rpp <= Rpn;
        Rip <= Rin;
        Rdp <= Rdn;
        busy_p <= busy;
      end if;
    end process;
----- Kombinasional ----------------------
process(busy_p, bp, cp, Rp,Rsp,Rpp,Rip,Rdp) --hp
begin
  if(busy_p'event and busy_p = '0') then
    if cp = "11" then
      case (bp) is
        when "000" =>
          Rsn <= Rp; Rpn <= Rpp; Rin <= Rip; Rdn <=
Rdp;
        when "010" =>
          Rsn <= Rsp; Rpn <= Rp; Rin <= Rip; Rdn <=
Rdp;
        when "100" =>
          Rsn <= Rsp; Rpn <= Rpp; Rin <= Rp; Rdn <=
Rdp;
        when "110" =>
          Rsn <= Rsp; Rpn <= Rpp; Rin <= Rip; Rdn <=
Rp;
        when others =>
          Rsn <= Rsp; Rpn <= Rpp; Rin <= Rip; Rdn <=
Rdp;
      end case;
    else
      Rsn <= Rsp; Rpn <= Rpp; Rin <= Rip; Rdn <=
Rdp;
    end if;
  end if;
end process;
----- out ---------------------
process (Rsp,Rpp,Rip,Rdp)
  begin
    Reg_S <= Rsp;
    Reg_P <= Rpp;
    Reg_I <= Rip;
    Reg_D <= Rdp;
  end process;
end Behavioral;
```

3. Kontroler PID

```
----------------------------------------------------------------------
---------
-- Company:
-- Engineer:
--
-- Create Date:    23:03:55 06/07/2015
-- Design Name:
-- Module Name:    Inst_kontroler - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Inst_kontroler is
    Port ( CLK,RST,STR : in STD_LOGIC;
           Kp,Ki,Kd  : in  STD_LOGIC_VECTOR (15
downto 0);
           Inp_in : in  STD_LOGIC_VECTOR (15 downto
0);
           Enc_in : in  STD_LOGIC_VECTOR (15 downto
0);
           SW     : IN std_logic_vector(1 downto 0);
           toPWM : out  STD_LOGIC_VECTOR (15
downto 0));
end Inst_kontroler;

architecture Behavioral of Inst_kontroler is
------------ SIGNAL ----------------------------------------
signal senA,sen1,sen2,sen3,sen4,sen5,senB : STD_logic;
signal sKi,sKd,sPuls : STD_LOGIC_VECTOR (15
downto 0);
signal sPID : STD_LOGIC_VECTOR (31 downto 0);
--- simulasi ----
signal sum1,sum2, reg1s : STD_LOGIC_VECTOR (15
downto 0);
signal mul1,mul2,mul3, reg2s, sum3,sum4,sum5 :
STD_LOGIC_VECTOR (31 downto 0);
SIGNAL sPulsa :  STD_LOGIC_VECTOR (15 downto
0);
SIGNAL B1 :  STD_LOGIC_VECTOR (9 downto 0);
SIGNAL B2 :  STD_LOGIC_VECTOR (12 downto 0);
SIGNAL C :  STD_LOGIC_VECTOR (1 downto 0);
------------ COMP_TAG ----------------------------------------
COMPONENT Cntrl_EN
  PORT(
    CLK : IN std_logic;
    RST : IN std_logic;
    STR : IN std_logic;
    Ena : OUT std_logic;
    En1 : OUT std_logic;
    En2 : OUT std_logic;
    En3 : OUT std_logic;
    En4 : OUT std_logic;
    En5 : OUT std_logic;
    Enb : OUT std_logic
    );
```

```
  END COMPONENT;
COMPONENT PID
  PORT(
    CLK : IN std_logic;
    RST : IN std_logic;
    en1 : IN std_logic;
    en2 : IN std_logic;
    en3 : IN std_logic;
    en4 : IN std_logic;
    en5 : IN std_logic;
    Kp : IN std_logic_vector(15 downto 0);
    Ki : IN std_logic_vector(15 downto 0);
    Kd : IN std_logic_vector(15 downto 0);
    Inp_in : IN std_logic_vector(15 downto 0);
    Enc_in : IN std_logic_vector(15 downto 0);
    H_PID : OUT std_logic_vector(31 downto 0)
    );
  END COMPONENT;
COMPONENT S_mux
  PORT(
    CLK : IN std_logic;
    RST : IN std_logic;
    ENa : IN std_logic;
    ENb : IN std_logic;
    SW : IN std_logic_vector(1 downto 0);
    Ki_in : IN std_logic_vector(15 downto 0);
    Kd_in : IN std_logic_vector(15 downto 0);
    Puls_in : IN std_logic_vector(15 downto 0);
    Puls_ot : OUT std_logic_vector(15 downto 0);
    Ki_out : OUT std_logic_vector(15 downto 0);
    Kd_out : OUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;
COMPONENT PtoP
  PORT(
    xPID : IN std_logic_vector(31 downto 0);
    xPWM : OUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;
-- COMP_TAG_END ------------------------------------------
--

begin
------------- INST_TAG ------------------------------------
Inst_Cntrl_EN: Cntrl_EN PORT MAP(
    CLK => clk,
    RST => rst,
    STR => str,
    Ena => sena,
    En1 => sen1,
    En2 => sen2,
    En3 => sen3,
    En4 => sen4,
    En5 => sen5,
    Enb => senb
  );
Inst_PID: PID PORT MAP(
    CLK => clk,
    RST => rst,
    en1 => sen1,
    en2 => sen2,
    en3 => sen3,
    en4 => sen4,
    en5 => sen5,
    Kp => Kp,
    Ki => sKi,
    Kd => sKd,
    Inp_in => Inp_in,
    Enc_in => sPuls,
    H_PID => sPID
  );
Inst_S_mux: S_mux PORT MAP(
    CLK => clk,
    RST => rst,
    ENa => sena,
    ENb => senb,
    SW => sw,
    Ki_in => Ki,
    Kd_in => Kd,
    Puls_in => Enc_in,
    Puls_ot => sPuls,
    Ki_out => sKi,
    Kd_out => sKd
  );
Inst_PtoP: PtoP PORT MAP(
    xPID => sPID,
    xPWM => toPWM
  );
-- INST_TAG_END ---- MEMORY ----------------------
--------

end Behavioral;

----------------------------------------------------------------
---------
-- Company:
-- Engineer:
--
-- Create Date:    22:41:48 06/07/2015
-- Design Name:
-- Module Name:    Cntrl_EN - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Cntrl_EN is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           STR : in  STD_LOGIC;
           Ena : out  STD_LOGIC;
           Enb : out  STD_LOGIC;
           En1 : out  STD_LOGIC;
```

```
        En2 : out  STD_LOGIC;
        En3 : out  STD_LOGIC;
        En4 : out  STD_LOGIC;
        En5 : out  STD_LOGIC);
end Cntrl_EN;

architecture Behavioral of Cntrl_EN is
------------ SIGNAL ---------------------------------------
signal cnt_n,cnt_p : unsigned (4 downto 0);
signal ENbuff_n, ENbuff_p : STD_LOGIC;
------------ COMP_TAG -------------------------------------
-- COMP_TAG_END ---------------------------------------
--

begin
------------ INST_TAG -------------------------------------
-- INST_TAG_END ---- MEMORY ------------------------
--------
  process (clk,rst)
  begin
    if rst = '1' then
      cnt_p <= (others => '0');
      ENbuff_p <= '0';
    elsif (clk'event and clk = '1') then
      cnt_p <= cnt_n;
      ENbuff_p <= ENbuff_n;
    end if;
  end process;
-- C -- U ---------------------------------------------
  process (cnt_p, STR, ENbuff_p)
  begin
    cnt_n <= cnt_p+1;
    ENbuff_n <= not STR;
    if ENbuff_p = '1' and STR = '1' then
      cnt_n <= (others => '0');
    elsif cnt_p = 28 then
      cnt_n <= cnt_p;
    end if;
    ------------------------------------------------------
    case (cnt_p) is
      when "00000" =>
        enb <= '0'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "00001" =>
        enb <= '0'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "00010" =>
        enb <= '0'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "00011" =>
        enb <= '0'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '1';
      when "00100" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "00101" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "00110" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "00111" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "01000" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "01001" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "01010" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "01011" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "01100" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "01101" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "01110" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "01111" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "10000" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "10001" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "10010" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "10011" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "10100" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "10101" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "10110" =>
        enb <= '1'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "10111" =>
        enb <= '0'; en1 <= '1'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "11000" =>
        enb <= '0'; en1 <= '0'; en2 <= '1'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "11001" =>
        enb <= '0'; en1 <= '0'; en2 <= '0'; en3 <= '1'; en4
<= '0'; en5 <= '0'; ena <= '0';
      when "11010" =>
        enb <= '0'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '1'; en5 <= '0'; ena <= '0';
      when "11011" =>
        enb <= '0'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '1'; ena <= '0';
      when others =>
        enb <= '0'; en1 <= '0'; en2 <= '0'; en3 <= '0'; en4
<= '0'; en5 <= '0'; ena <= '0';
    end case;
  end process;
  ------------------------------------------------------
end Behavioral;

------------------------------------------------------------------
---------
```

```
-- Company:
-- Engineer:
--
-- Create Date:    17:14:45 03/21/2015
-- Design Name:
-- Module Name:    PID - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
--------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity PID is
    Port ( CLK,RST : in STD_LOGIC;
           en1,en2,en3,en4,en5 : IN STD_logic;
           Kp,Ki,Kd  : in  STD_LOGIC_VECTOR (15
downto 0);
           Inp_in : in  STD_LOGIC_VECTOR (15 downto
0);
           Enc_in : in  STD_LOGIC_VECTOR (15 downto
0);
           H_PID : out  STD_LOGIC_VECTOR (31 downto
0));
end PID;

architecture Behavioral of PID is
------------ SIGNAL ----------------------------------------
signal rst1_n,rst1_p : STD_LOGIC;
signal sum1,sum2, reg1s : STD_LOGIC_VECTOR (15
downto 0);
signal mul1,mul2,mul3, reg2s, sum3,sum4,sum5 :
STD_LOGIC_VECTOR (31 downto 0);
------------ COMP_TAG ---------------------------------
COMPONENT add_1
    PORT (
      a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      b : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      clk : IN STD_LOGIC;
      ce : IN STD_LOGIC;
      sclr : IN STD_LOGIC;
      s : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
    );
  END COMPONENT;
COMPONENT add_2
    PORT (
      a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      b : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      clk : IN STD_LOGIC;
      ce : IN STD_LOGIC;
      sclr : IN STD_LOGIC;
      s : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
    );
  END COMPONENT;
COMPONENT multp1
    PORT (
      clk : IN STD_LOGIC;
      a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      b : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      ce : IN STD_LOGIC;
      sclr : IN STD_LOGIC;
      p : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
  END COMPONENT;
COMPONENT multp2
    PORT (
      clk : IN STD_LOGIC;
      a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      b : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      ce : IN STD_LOGIC;
      sclr : IN STD_LOGIC;
      p : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
  END COMPONENT;
COMPONENT multp3
    PORT (
      clk : IN STD_LOGIC;
      a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      b : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      ce : IN STD_LOGIC;
      sclr : IN STD_LOGIC;
      p : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
  END COMPONENT;
COMPONENT add_3
    PORT (
      a : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      b : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      clk : IN STD_LOGIC;
      ce : IN STD_LOGIC;
      sclr : IN STD_LOGIC;
      s : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
  END COMPONENT;
COMPONENT add_4
    PORT (
      a : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      b : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      clk : IN STD_LOGIC;
      ce : IN STD_LOGIC;
      sclr : IN STD_LOGIC;
      s : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
  END COMPONENT;
COMPONENT add_5
    PORT (
      a : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      b : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      clk : IN STD_LOGIC;
      ce : IN STD_LOGIC;
      sclr : IN STD_LOGIC;
      s : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
  END COMPONENT;
COMPONENT Reg1
```

```
  PORT(
    clk : IN std_logic;
    rst : IN std_logic;
    Data_in : IN std_logic_vector(15 downto 0);
    Data_out : OUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;
COMPONENT Reg2
  PORT(
    clk : IN std_logic;
    rst : IN std_logic;
    Data_in : IN std_logic_vector(31 downto 0);
    Data_out : OUT std_logic_vector(31 downto 0)
    );
  END COMPONENT;
-- COMP_TAG_END ------------------------------------------
--

begin
------------ INST_TAG ------------------------------------
Instance_sub1 : add_1 PORT MAP (
   a => inp_in,
   b => enc_in,
   clk => clk,
   ce => en1,
   sclr => rst1_p,
   s => sum1
 );
Instance_sub2 : add_2 PORT MAP (
   a => sum1,
   b => reg1s,
   clk => clk,
   ce => en2,
   sclr => rst1_p,
   s => sum2
 );
Instance_Multp_Kp : multp1 PORT MAP (
   clk => clk,
   a => sum1,
   b => Kp,
   ce => en3,
   sclr => rst1_p,
   p => mul1
 );
Instance_Multp_Ki : multp2 PORT MAP (
   clk => clk,
   a => sum1,
   b => Ki,
   ce => en3,
   sclr => rst1_p,
   p => mul2
 );
Instance_Multp_Kd : multp3 PORT MAP (
   clk => clk,
   a => sum2,
   b => Kd,
   ce => en3,
   sclr => rst1_p,
   p => mul3
 );
Instance_add3 : add_3 PORT MAP (
   a => mul2,
   b => reg2s,
   clk => clk,
   ce => en4,
   sclr => rst1_p,
   s => sum3
 );
Instance_add4 : add_4 PORT MAP (
   a => mul1,
   b => mul3,
   clk => clk,
   ce => en4,
   sclr => rst1_p,
   s => sum4
 );
Instance_add5 : add_5 PORT MAP (
   a => sum3,
   b => sum4,
   clk => clk,
   ce => en5,
   sclr => rst1_p,
   s => sum5
 );
Inst_Reg1: Reg1 PORT MAP(
    clk => en3,
    rst => rst,
    Data_in => sum1,
    Data_out => reg1s
  );
Inst_Reg2: Reg2 PORT MAP(
    clk => en5,
    rst => rst,
    Data_in => sum3,
    Data_out => reg2s
  );
-- INST_TAG_END ---- MEMORY -----------------------
-------
  process (clk,rst)
  begin
    if rst = '1' then
      rst1_p <= '1';
    elsif (clk'event and clk = '1') then
      rst1_p <= rst1_n;
    end if;
  end process;
-- reset ------------------------------------------------
  process (rst,rst1_p)
  begin
    rst1_n <= rst;
    if rst1_p = '1' then
      rst1_n <= '0';
    end if;
  end process;
  ------------------------------------------------------
  H_PID <= sum5;
end Behavioral;

----------------------------------------------------------------------
---------
-- Company:
-- Engineer:
--
-- Create Date:    13:05:35 04/30/2015
-- Design Name:
-- Module Name:    Reg1 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
```

```
-- Revision 0.01 - File Created
-- Additional Comments:
----------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Reg1 is
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          Data_in : in  STD_LOGIC_VECTOR (15 downto
0);
          Data_out : out  STD_LOGIC_VECTOR (15
downto 0));
end Reg1;

architecture Behavioral of Reg1 is
signal reg1_p,reg1_n : STD_LOGIC_VECTOR (15
downto 0);

begin
---- MEMORY --------------------------------
  process (clk,rst)
  begin
    if rst = '1' then
      reg1_p <= (others => '0');
    elsif (clk'event and clk = '0') then
      reg1_p <= reg1_n;
    end if;
  end process;
-- register --------------------------------
  process (Data_in, reg1_p)
  begin
    reg1_n <= data_in;
    data_out <= reg1_p;
  end process;
end Behavioral;

----------------------------------------------------------------------
---------
-- Company:
-- Engineer:
--
-- Create Date:    04:28:37 05/05/2015
-- Design Name:
-- Module Name:    Reg2 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Reg2 is
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          Data_in : in  STD_LOGIC_VECTOR (31
downto 0);
          Data_out : out  STD_LOGIC_VECTOR (31
downto 0));
end Reg2;

architecture Behavioral of Reg2 is
signal reg2_p,reg2_n : STD_LOGIC_VECTOR (31
downto 0);

begin
---- MEMORY --------------------------------
  process (clk,rst)
  begin
    if rst = '1' then
      reg2_p <= (others => '0');
    elsif (clk'event and clk = '0') then
      reg2_p <= reg2_n;
    end if;
  end process;
-- register --------------------------------
  process (Data_in, reg2_p)
  begin
    reg2_n <= data_in;
    data_out <= reg2_p;
  end process;
end Behavioral;

----------------------------------------------------------------------
---------
-- Company:
-- Engineer:
--
-- Create Date:    22:18:40 06/07/2015
-- Design Name:
-- Module Name:    S_mux - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
---------
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity S_mux is
    Port ( CLK   : in  STD_LOGIC;
           RST   : in  STD_LOGIC;
           ENa   : in  STD_LOGIC;
           ENb   : in  STD_LOGIC;
           SW    : in  STD_LOGIC_VECTOR (1 downto 0);
           Ki_in : in  STD_LOGIC_VECTOR (15 downto
0);
           Kd_in : in  STD_LOGIC_VECTOR (15 downto
0);
           Puls_in : in  STD_LOGIC_VECTOR (15 downto
0);
           Puls_ot : out  STD_LOGIC_VECTOR (15
downto 0);
           Ki_out : out  STD_LOGIC_VECTOR (15 downto
0);
           Kd_out : out  STD_LOGIC_VECTOR (15
downto 0));
end S_mux;

architecture Behavioral of S_mux is
------------ SIGNAL -------------------------------
SIGNAL H_Kali,sPuls :  STD_LOGIC_VECTOR (15
downto 0);
SIGNAL H_Bagi :  STD_LOGIC_VECTOR (13 downto
0);
SIGNAL B1 :  STD_LOGIC_VECTOR (9 downto 0);
SIGNAL B2 :  STD_LOGIC_VECTOR (12 downto 0);
SIGNAL C :  STD_LOGIC_VECTOR (1 downto 0);
------------ COMP_TAG -----------------------------------
component T_Bagi
      port (
      clk: in std_logic;
      ce: in std_logic;
      sclr: in std_logic;
      rfd: out std_logic;
      dividend: in std_logic_vector(13 downto 0);
      divisor: in std_logic_vector(9 downto 0);
      quotient: out std_logic_vector(13 downto 0);
      fractional: out std_logic_vector(1 downto 0));
    end component;
COMPONENT T_Kali
     PORT (
      clk : IN STD_LOGIC;
      a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      b : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
      ce : IN STD_LOGIC;
      sclr : IN STD_LOGIC;
      p : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
     );
   END COMPONENT;
component Bagi_puls
      port (
      clk: in std_logic;
      ce: in std_logic;
      rfd: out std_logic;
      dividend: in std_logic_vector(15 downto 0);
      divisor: in std_logic_vector(1 downto 0);
      quotient: out std_logic_vector(15 downto 0);
      fractional: out std_logic_vector(1 downto 0));
    end component;
COMPONENT Puls_kali
    PORT (
       clk : IN STD_LOGIC;
       a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
       b : IN STD_LOGIC_VECTOR(12 DOWNTO 0);
       ce : IN STD_LOGIC;
       p : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
    );
   END COMPONENT;
-- COMP_TAG_END ------------------------------------
--

begin
------------ INST_TAG ----------------------------------
Inst_Bagi : T_Bagi
   port map (
      clk => clk,
      ce => enb,
      sclr => rst,
      rfd => open,
      dividend => Ki_in(13 downto 0),
      divisor => B1,
      quotient => H_bagi,
      fractional => open
    );
Inst_t_kali : T_Kali
  PORT MAP (
    clk => clk,
    a => Kd_in,
    b => B1,
    ce => ENa,
    sclr => rst,
    p => H_Kali
  );
 Inst_bagi_puls : Bagi_puls
     port map (
      clk => clk,
      ce => enb,
      rfd => open,
      dividend => sPuls,
      divisor => C,
      quotient => Puls_ot,
      fractional => open
    );
Inst_Kali_puls : Puls_kali
  PORT MAP (
    clk => clk,
    a => Puls_in,
    b => B2,
    ce => ena,
    p => sPuls
  );
-- INST_TAG_END ---- MEMORY ------------------------
--------
   process (SW, Ki_in, Kd_in, H_Bagi,H_Kali,
Puls_in,sPuls)
   begin
     C <= "11";
     case (SW) is
       when "00" =>
```

```
        Ki_out <= Ki_in;
        Kd_out <= Kd_in;
        B1 <= "0000000001";
        B2 <= "0000000000101"; --1s
      when "01" =>
        Ki_out(13 downto 0)  <= H_Bagi;
        Ki_out(15 downto 14) <= "00";
        Kd_out <= H_Kali;
        B1 <= "0000001010";
        B2 <= "0000000110010"; --100ms
      when "10" =>
        Ki_out(13 downto 0)  <= H_Bagi;
        Ki_out(15 downto 14) <= "00";
        Kd_out <= H_Kali;
        B1 <= "0001100100";
        B2 <= "0000111110100"; --10ms
      when "11" =>
        Ki_out(13 downto 0)  <= H_Bagi;
        Ki_out(15 downto 14) <= "00";
        Kd_out <= H_Kali;
        B1 <= "1111101000";
        B2 <= "1001110001000"; --1ms
      when others =>
    end case;
  end process;
  ----------------------------------------------------
end Behavioral;

----------------------------------------------------------------------------
--------
-- Company:
-- Engineer:
--
-- Create Date:    15:46:39 08/04/2015
-- Design Name:
-- Module Name:    PtoP - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity PtoP is
  Port (
       xPID : in  STD_LOGIC_VECTOR (31 downto 0);
       xPWM : out  STD_LOGIC_VECTOR (15 downto
0));
end PtoP;

architecture Behavioral of PtoP is
------------ SIGNAL ----------------------------------
signal sPWM2 : STD_LOGIC_VECTOR (15 downto 0);
------------ COMP_TAG ----------------------------------
-- COMP_TAG_END ------------------------------------------
--

begin
------------ INST_TAG ----------------------------------
-- INST_TAG_END ---- MEMORY ----------------------
--------
process (sPWM2, xPID)
  begin
    if xPID > x"0000FFFF" then
      sPWM2 <= (others => '1');
    else
      sPWM2 <= xPID(15 downto 0);
    end if;
    xPWM <= sPWM2;
  end process;
end Behavioral;
```

4. PWM

```
----------------------------------------------------------------------------
-------------
-- Company:
-- Engineer:
--
-- Create Date:    17:02:43 04/01/2015
-- Design Name:
-- Module Name:    PWM_generator - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity PWM_generator is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
```

```
        data_in : in  STD_LOGIC_VECTOR (15 downto
0);
        PWM_out : out  STD_LOGIC);
end PWM_generator;

architecture Behavioral of PWM_generator is
signal cnt_n, cnt_p: unsigned(15 downto 0);
signal data_n, data_p: STD_LOGIC_VECTOR (15
downto 0);
signal pwm_n, pwm_p: STD_Logic;

begin
---- Memory ------------------------------------------------
-----
  process (clk,rst)
  begin
    if rst = '1' then
      pwm_p <= '0';
      cnt_p <= (others => '0');
      data_p <= (others => '0');
    elsif (clk'event and clk = '0') then
      pwm_p <= pwm_n;
      cnt_p <= cnt_n;
      data_p <= data_n;
    end if;
  end process;
---- Kombinasional nilai PWM -------------------------------
---------
  process (pwm_n,pwm_p, data_in,data_p,cnt_p)
  begin
    pwm_n <= pwm_p;
    cnt_n <= cnt_p+1;
    data_n <= data_p;
    if cnt_p = "00000000" then
      data_n <= data_in;
    end if;
    if STD_LOGIC_VECTOR(cnt_p) <= data_p then
      pwm_n <= '1';
    else
      pwm_n <= '0';
    end if;
  -- KOMBINASIONAL output ----------------------------
---------------
    if cnt_p = "00000000" then
      PWM_out <= pwm_p;
    else PWM_out <= pwm_n;
    end if;
  end process;
end Behavioral;
```

5. Pembaca Sensor

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer:
--
-- Create Date:    13:24:14 06/04/2015
-- Design Name:
-- Module Name:    Sensor_inst - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Sensor_inst is
    Port ( Clk  : in  STD_LOGIC;
           Rst  : in  STD_LOGIC;
           Puls : in  STD_LOGIC;
           En   : in  STD_LOGIC;
           Cntr : OUT std_logic_vector(15 downto 0));
end Sensor_inst;

architecture Behavioral of Sensor_inst is
------------ SIGNAL ------------------------------------
signal sPuls : STD_LOGIC_VECTOR (15 downto 0);
signal sNol : std_logic;
------------ COMP_TAG ----------------------------------
COMPONENT Pulsa_cnt
  PORT(
    Rst : IN std_logic;
    cek : IN std_logic;
    EN : IN std_logic;
    Pulsa : IN std_logic;
    Cnt : OUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;
COMPONENT Pulsa_buff
  PORT(
    EN : IN std_logic;
    Rst : IN std_logic;
    Cek : IN std_logic;
    Cnt_in : IN std_logic_vector(15 downto 0);
    Cnt_out : OUT std_logic_vector(15 downto 0)
    );
  END COMPONENT;
COMPONENT Pulsa_check
  PORT(
    clk : IN std_logic;
    rst : IN std_logic;
    EN : IN std_logic;
    Puls : IN std_logic;
    Pout : OUT std_logic
    );
  END COMPONENT;
-- COMP_TAG_END ------------------------------------------
--

begin
------------ INST_TAG ----------------------------------
Inst_Pulsa_cnt: Pulsa_cnt PORT MAP(
    Rst => rst,
```

```
    cek => sNol,
    EN => en,
    Pulsa => puls,
    Cnt => sPuls
  );
Inst_Pulsa_buff: Pulsa_buff PORT MAP(
    EN => en,
    Rst => rst,
    Cek => sNol,
    Cnt_in => sPuls,
    Cnt_out => Cntr
  );
Inst_Pulsa_check: Pulsa_check PORT MAP(
    clk => clk,
    rst => rst,
    EN => en,
    Puls => Puls,
    Pout => sNol
  );
-- INST_TAG_END ------------------------------------------
-
end Behavioral;

----------------------------------------------------------------------
------------
-- Company:
-- Engineer:
--
-- Create Date:    10:41:41 06/04/2015
-- Design Name:
-- Module Name:    Pulsa_cnt - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Pulsa_cnt is
    Port ( Rst,cek : in  STD_LOGIC;
           EN  : in  STD_LOGIC;
           Pulsa : in  STD_LOGIC;
           Cnt : out  STD_LOGIC_VECTOR (15 downto
0));
end Pulsa_cnt;

architecture Behavioral of Pulsa_cnt is
signal cntbuff_p, cntbuff_n :  unsigned (15 downto 0);
signal enbuff :  STD_LOGIC;

begin
---- Memory ------------------------------------------------
---
--process (clk, rst)
--begin
--    if rst = '1' then
----        cntbuff_p <= (others => '0');
--        enbuff  <= '0';
--        cekp <= '0';
--    elsif (Pulsa'event and Pulsa = '0')  then
----        cntbuff_p <= cntbuff_n;
--        enbuff  <= en;
--        cekp <= cekn;
--    end if;
--end process;
    process (Pulsa, cek, rst)
    begin
        if rst = '1' then
            cntbuff_p <= (others => '0');
            enbuff  <= '0';
--        cekp <= '0';
        elsif (Pulsa'event and Pulsa = '0')  then
            if cek = '1' then
                cntbuff_p <= cntbuff_n;
            end if;
            enbuff <= en;
--        cekp <= cekn;
        end if;
    end process;
---- Kombinasional -----------------------------------------
---
    process (cntbuff_p, enbuff,en)
    begin
--    cntbuff_n <= cntbuff_p+1;
        ------------------------------------------------------
        if (en = '0' and enbuff = '1') then
            cntbuff_n <= (0=>'1', others => '0');
        else
            cntbuff_n <= cntbuff_p+1;
        end if;
        --output ------------------------------------------
        Cnt <= STD_LOGIC_VECTOR(cntbuff_p);
    end process;
end Behavioral;

----------------------------------------------------------------------
------------
-- Company:
-- Engineer:
--
-- Create Date:    13:07:57 06/04/2015
-- Design Name:
-- Module Name:    Pulsa_buff - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
```

```
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Pulsa_buff is
    Port ( EN : in  STD_LOGIC;
           Rst : in  STD_LOGIC;
           Cek : in  STD_LOGIC;
           Cnt_in : in  STD_LOGIC_VECTOR (15 downto
0);
           Cnt_out : out  STD_LOGIC_VECTOR (15
downto 0));
end Pulsa_buff;

architecture Behavioral of Pulsa_buff is
signal buff_n, buff_p : STD_logic_vector (15 downto 0);

begin
---- Memory ----------------------------------------------
---
   process (en, rst)
   begin
     if rst = '1' then
        buff_p <= (others => '0');
     elsif (en'event and en = '0') then
        buff_p <= buff_n;
     end if;
   end process;
---- Kombinasional ---------------------------------------
---
   process (buff_p, cnt_in, cek)
   begin
     buff_n <= cnt_in;
     ------------------------------------------------------
     --output ----------------------------------------------
     if cek = '1' then
        Cnt_out <= buff_p;
     else
        Cnt_out <= (others => '0');
     end if;
   end process;
end Behavioral;

----------------------------------------------------------------------
-------------
-- Company:
-- Engineer:
--
-- Create Date:    12:55:40 06/05/2015
-- Design Name:
-- Module Name:    Pulsa_check - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Pulsa_check is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           EN  : in  STD_LOGIC;
           Puls : in  STD_LOGIC;
           Pout : out  STD_LOGIC);
end Pulsa_check;

architecture Behavioral of Pulsa_check is
signal Puls_n,Puls_p, cek_n,cek_p, cekbuff_n,cekbuff_p,
enbuff_n,enbuff_p : STD_Logic;

begin
---- Memory ----------------------------------------------
---
   process (clk, rst)
   begin
     if rst = '1' then
        Puls_p    <= '0';
        cek_p     <= '0';
        cekbuff_p <= '0';
        enbuff_p  <= '0';
     elsif (clk'event and clk = '0') then
        Puls_p    <= Puls_n;
        cek_p     <= cek_n;
        cekbuff_p <= cekbuff_n;
        enbuff_p  <= enbuff_n;
     end if;
   end process;
---- Kombinasional ---------------------------------------
---
   process (EN, Puls,Puls_p,cek_p, cekbuff_p,enbuff_p)
   begin
     Puls_n    <= Puls;
     cek_n     <= cek_p;
     cekbuff_n <= cekbuff_p;
     enbuff_n  <= en;
     -- ini process ----------
     if en = '0' and enbuff_p = '1' then
        cek_n <= '0';
        cekbuff_n <= cek_p;
     elsif Puls = '0' and Puls_p = '1' then
        cek_n <= '1';
     end if;
     -- output ----------------------------------------------
        Pout <= cekbuff_p;
```

```
  end process;
end Behavioral;


Pemilih Waktu Sampling
----------------------------------------------------------------------------
------------
-- Company:
-- Engineer:
--
-- Create Date:    15:06:44 06/05/2015
-- Design Name:
-- Module Name:    Sam2 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Sam2 is
   Port ( clk_in : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          sw : in std_logic_vector (1 downto 0);
          clk4us : out  STD_LOGIC);
end Sam2;

architecture Behavioral of Sam2 is
signal cnt1_N, cnt1_P, delay : unsigned (27 downto 0);
signal clk1_N, clk1_P : std_logic;
signal data_N, data_P : std_logic_vector (1 downto 0);

begin
---- MEMORY ------------------------------------------------
------
  process(clk_in,rst)
    begin
      if rst = '1' then
        cnt1_P <= (others => '0');
        data_P <= (others => '0');
        clk1_P <= '0';
      elsif falling_edge(clk_in) then
        cnt1_P <= cnt1_N;
        data_P <= data_N;
        clk1_P <= clk1_N;
      end if;
    end process;

---- PULSA KELUARAN ----------------------------------
-----------
  process(clk1_P,cnt1_p,sw,data_p)
    begin
      -- ini disebut default --
      clk1_N <= clk1_P;
      data_N <= sw;
      cnt1_N <= cnt1_P + 1;
      -- beda sw -------------
      if data_p /= sw then
        cnt1_N <= (others => '0');
      end if;
      -- delay ---------------
      case (sw) is
        when "00" =>
          delay <= x"17d783f"; -- 1Hz x"2faf07f"
        when "01" =>
          delay <= x"026259f"; -- 100ms 10Hz
        when "10" =>
          delay <= x"003d08f"; -- 10ms 100Hz
        when "11" =>
          delay <= x"00061a7"; -- 1ms 1kHz
        when others =>
      end case;
      -- ini process ----------
      if cnt1_P = delay then
        cnt1_N <= (others => '0');
        clk1_N <= not clk1_P;
      end if;
      -------------------------
      clk4us <= clk1_P; --output
    end process;

end Behavioral;
```

6. Keluaran Seven Segment

```
----------------------------------------------------------------------------
------------
-- Company:
-- Engineer:
--
-- Create Date:    18:19:42 06/07/2015
-- Design Name:
-- Module Name:    Inst_7seg - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Inst_7seg is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           D0,D1,D2,D3 : in  STD_LOGIC_VECTOR (3
downto 0);
           AN : out  STD_LOGIC_VECTOR (3 downto 0);
           seg : out  STD_LOGIC_VECTOR (6 downto 0);
           DP : out  STD_LOGIC);
end Inst_7seg;

architecture Behavioral of Inst_7seg is
------------ SIGNAL -------------------------------
signal sData : STD_LOGIC_VECTOR (3 downto 0);
signal sAddr : STD_LOGIC_VECTOR (1 downto 0);
------------ COMP_TAG -----------------------------
COMPONENT DekoderHex
  PORT(
    addr : IN std_logic_vector(1 downto 0);
    dat  : IN std_logic_vector(3 downto 0);
    AN  : OUT std_logic_vector(3 downto 0);
    seg : OUT std_logic_vector(6 downto 0);
    dp  : OUT std_logic
    );
  END COMPONENT;
COMPONENT addr_cnt
  PORT(
    clk : IN std_logic;
    rst : IN std_logic;
    addr : OUT std_logic_vector(1 downto 0)
    );
  END COMPONENT;
COMPONENT mux_7sa
  PORT(
    ADDR : IN std_logic_vector(1 downto 0);
    D0 : IN std_logic_vector(3 downto 0);
    D1 : IN std_logic_vector(3 downto 0);
    D2 : IN std_logic_vector(3 downto 0);
    D3 : IN std_logic_vector(3 downto 0);
    Dout : OUT std_logic_vector(3 downto 0)
    );
  END COMPONENT;
-- COMP_TAG_END -------------------------------------
--

begin
------------ INST_TAG -----------------------------
Inst_DekoderHex: DekoderHex PORT MAP(
    addr => saddr,
    dat  => sData,
    AN   => AN,
    seg  => seg,
    dp   => dp
  );
Inst_addr_cnt: addr_cnt PORT MAP(
    clk => clk,
    rst => rst,
    addr => saddr
  );
Inst_mux_7s: mux_7sa PORT MAP(
    ADDR => saddr,
    D0 => d0,
    D1 => d1,
    D2 => d2,
    D3 => d3,
    Dout => sData
  );
-- INST_TAG_END -------------------------------------
-
end Behavioral;

----------------------------------------------------------------
------------
-- Company:
-- Engineer:
--
-- Create Date:    20:01:55 04/01/2015
-- Design Name:
-- Module Name:    DekoderHex - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------
------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity DekoderHex is
    Port ( addr : in  STD_LOGIC_VECTOR (1 downto 0);
           dat : in  STD_LOGIC_VECTOR (3 downto 0);
           AN : out  STD_LOGIC_VECTOR (3 downto 0);
           seg : out  STD_LOGIC_VECTOR (6 downto 0);
           dp : out  STD_LOGIC);
end DekoderHex;

architecture Behavioral of DekoderHex is

begin
  process (DAT,ADDR)
  begin
    case (ADDR) is
      when "00" =>
        AN <= "1110";
      when "01" =>
        AN <= "1101";
      when "10" =>
        AN <= "1011";
      when "11" =>
        AN <= "0111";
      when others =>
    end case;
```

```
    case (DAT) is
      when x"0" =>
        seg <= (6 => '1', others => '0'); --atau bisa juga
seg<="111110"
      when x"1" =>
        seg <= (5=>'0',4=>'0', others => '1');
      when x"2" =>
        seg <= (1=>'1',4=>'1', others => '0');
      when x"3" =>
        seg <= (1=>'1',2=>'1', others => '0');
      when x"4" =>
        seg <= (2=>'1',3=>'1',0=>'1', others => '0');

      when x"5" =>
        seg <= (2=>'1',5=>'1', others => '0');
      when x"6" =>
        seg <= (5=>'1', others => '0');
      when x"7" =>
        seg <= (0=>'0',5=>'0',4=>'0', others => '1');
      when x"8" =>
        seg <= (others => '0');
      when x"9" =>
        seg <= (2=>'1', others => '0');
      when x"a" =>
        seg <= (1=>'1', others => '0');
      when x"b" =>
        seg <= (5=>'1',0=>'1', others => '0');
      when x"c" =>
        seg <= (6=>'0',2=>'0',3=>'0', others => '1');
      when x"d" =>
        seg <= (1=>'1',0=>'1', others => '0');
      when x"e" =>
        seg <= (4=>'1',5=>'1', others => '0');
      when x"f" =>
        seg <= (3=>'1',4=>'1',5=>'1', others => '0');
      when others =>
        seg <= (others => '1');
    end case;
    dp <= '1'; --supaya MATI
  end process;

end Behavioral;

----------------------------------------------------------------------
-------------
-- Company:
-- Engineer:
--
-- Create Date:    14:24:10 12/20/2013
-- Design Name:
-- Module Name:    addr_cnt - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity addr_cnt is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           addr : out  STD_LOGIC_VECTOR (1 downto 0));
end addr_cnt;

architecture Behavioral of addr_cnt is
signal cnt200_N, cnt200_P : unsigned(1 downto 0);
signal cntd_N, cntd_P : unsigned(31 downto 0);

begin
---- MEMORY ----------------------------------------------
------
  process(rst,clk)
    begin
      if rst = '1' then
        cnt200_P <= (others => '0');
        cntd_P <= (others => '0');
      elsif (clk'event and clk = '0') then
        cnt200_P <= cnt200_N;
        cntd_P <= cntd_N;
      end if;
    end process;
---- PULSA KELUARAN ---------------------------------
-----------
  process(cnt200_P, cntd_p)
    begin
      -- ini disebut default ----
      cnt200_N <= cnt200_P;
      cntd_N <= cntd_P+1;
      -- ini process ------------
      if cntd_p = 249999 then
        cntd_n <= (others => '0');
        cnt200_n <= cnt200_p+1;
      end if;
      ---------------------------
      addr <= STD_LOGIC_VECTOR(cnt200_p); --
output
    end process;

end Behavioral;

----------------------------------------------------------------------
-------------
-- Company:
-- Engineer:
--
-- Create Date:    16:41:15 10/30/2013
-- Design Name:
-- Module Name:    mux_7sa - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
```

```
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux_7sa is
   Port ( ADDR : in  STD_LOGIC_VECTOR (1 downto
0);
          D0 : in  STD_LOGIC_VECTOR (3 downto 0);
          D1 : in  STD_LOGIC_VECTOR (3 downto 0);
          D2 : in  STD_LOGIC_VECTOR (3 downto 0);
          D3 : in  STD_LOGIC_VECTOR (3 downto 0);
          Dout : out  STD_LOGIC_VECTOR (3 downto
0));
end mux_7sa;

architecture Behavioral of mux_7sa is

begin
process (ADDR,D0,D1,D2,D3)
begin
 case (ADDR) is
     when "11" =>
       Dout <= D0;
     when "10" =>
       Dout <= D1;
     when "01" =>
       Dout <= D2;
     when "00" =>
       Dout <= D3;
     when others =>
      end case;
end process;

end Behavioral;
```

7. Pemilih Keluaran Seven Segment

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer:
--
-- Create Date:    10:27:50 05/18/2015
-- Design Name:
-- Module Name:    mux_tampilan - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux_tampilan is
    Port ( sp : in  STD_LOGIC_VECTOR (2 downto 0);
           set : in  STD_LOGIC_VECTOR (15 downto 0);
           kp : in  STD_LOGIC_VECTOR (15 downto 0);
           ki : in  STD_LOGIC_VECTOR (15 downto 0);
           kd : in  STD_LOGIC_VECTOR (15 downto 0);
           Puls : in  STD_LOGIC_VECTOR (15 downto 0);
           d0 : out  STD_LOGIC_VECTOR (3 downto 0);
           d1 : out  STD_LOGIC_VECTOR (3 downto 0);
           d2 : out  STD_LOGIC_VECTOR (3 downto 0);
           d3 : out  STD_LOGIC_VECTOR (3 downto 0));
end mux_tampilan;

architecture Behavioral of mux_tampilan is

begin
  process (sp, set, kp,ki,kd, Puls)
  begin
    case (sp) is
       when "100" =>
         d0 <= set(15 downto 12);d1 <= set(11 downto
8);d2 <= set(7 downto 4);d3 <= set(3 downto 0);
       when "101" =>
         d0 <= kp(15 downto 12);d1 <= kp(11 downto
8);d2 <= kp(7 downto 4);d3 <= kp(3 downto 0);
       when "110" =>
         d0 <= ki(15 downto 12);d1 <= ki(11 downto
8);d2 <= ki(7 downto 4);d3 <= ki(3 downto 0);
       when "111" =>
         d0 <= kd(15 downto 12);d1 <= kd(11 downto
8);d2 <= kd(7 downto 4);d3 <= kd(3 downto 0);
       when others =>
         d0 <= Puls(15 downto 12); d1 <= Puls(11
downto 8); d2 <= Puls(7 downto 4); d3 <= Puls(3 downto
0);
     end case;
  end process;
end Behavioral;
```

DI-REV1 [di-ay-ref-wan]
(DI-Rotary Encoder Versi #1)

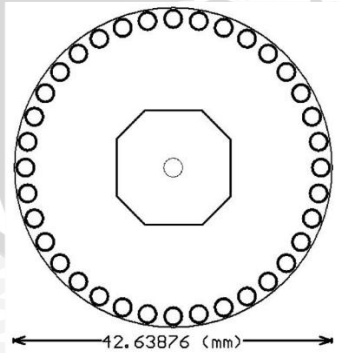
DIM.019

Spesifikasi:

- Terdiri dari dua bagian utama:
  1. Piringan derajat dengan 36 lubang pada kelilingnya dengan sudut antara dua lubang yang berdampingan terhadap titik tengahnya adalah 10°.
  2. Rangkaian sensor pembaca putaran yang menggunakan optocoupler tipe celah sebagai sensor pembaca perubahan posisi lubang piringan derajat.
- Tegangan-tegangan operasi:
  - Sumber (VCC): 3,5 – 5,5V
  - Logika output '0': 0 – 0,5V
  - Logika output '1': 3 – 5V (VCC – 0,5V)
- Logika output:
  - 0: Saat celah sensor terhalang
  - 1: Saat celah sensor tanpa-halangan
- Kecepatan baca sensor:
  - Kondisi logika *toggle* (0/1): 1500Hz
  - Rotasi dengan 36 lubang: 2500RPM

Deskripsi Perangkat Modul:

- Layout:



Gambar 1. Piringan Derajat DI-REV1.

WEIGHT : 127g (APPROX)

勢 仟 /傻

MABUCHI MOTOR

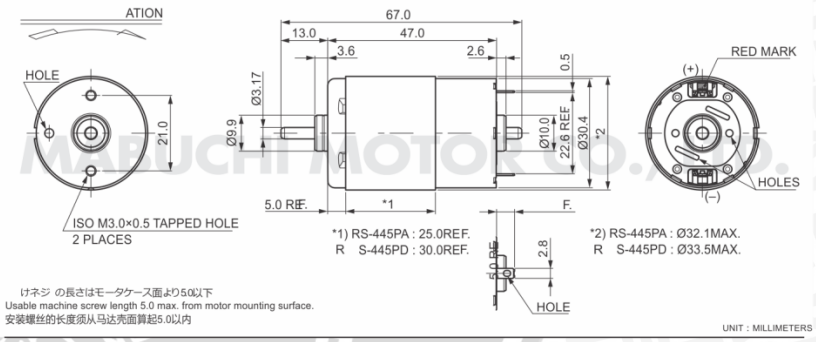
OUTPUT : 3.0W～55W (APPROX)　カーボンブラシ | Carbon-brush motors | 碳精 刷

代表的用途　精密・事務機器：インクジェットプリンター
Typical Applications　Precision and Ofce Equipment : Inkjet Printer
主要用途　PA　精密、办公设备 :喷墨打印机

MODEL	VOLTAGE		NO LOAD		AT MAXIMUM EFFICIENCY				STALL	
	OPERATING RANGE	NOMINAL	SPEED	CURRENT	SPEED	CURRENT	TORQUE	OUTPUT	TORQUE	CURRENT
			r/min	A	r/min	A	mN·m / g·cm	W	mN·m / g·cm	A
RS-445PA-14233	12～42	(!) 42V CONSTANT	6500	0.060	5410	0.30	13.7 / 140	7.78	81.8 / 834	1.47
RS-445PA-15200	12～42	(!) 42V CONSTANT	7600	0.067	6420	0.36	14.6 / 148	9.77	93.7 / 955	1.98
RS-445PD-18140	12～42	(!) 42V CONSTANT	9300	0.070	8130	0.48	16.9 / 173	14.4	134 / 1366	3.35

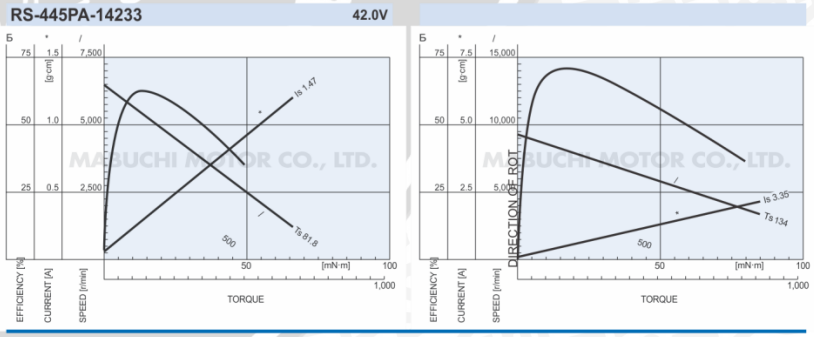
(!) This operating voltage indicates the peak value in case of pulse-width modulation (PWM) power supply.
Besides, as withstand voltage, 1800V for one second can not be guaranteed.
(!) 这个标示使用电压是 PWM执行控制时的峰值电压。另外 ,我们无法提供耐电压1800V 1sec的保证, 敬请见谅。

RS-445PA : エンコーダ搭載仕様もあります。 | Motor with an encoder is also available. | 有配　器的　达。



けネジ の長さはモータケース面より5.0以下
Usable machine screw length 5.0 max. from motor mounting surface.
安装螺丝的长度须从马达壳面算起5.0以内

UNIT : MILLIMETERS

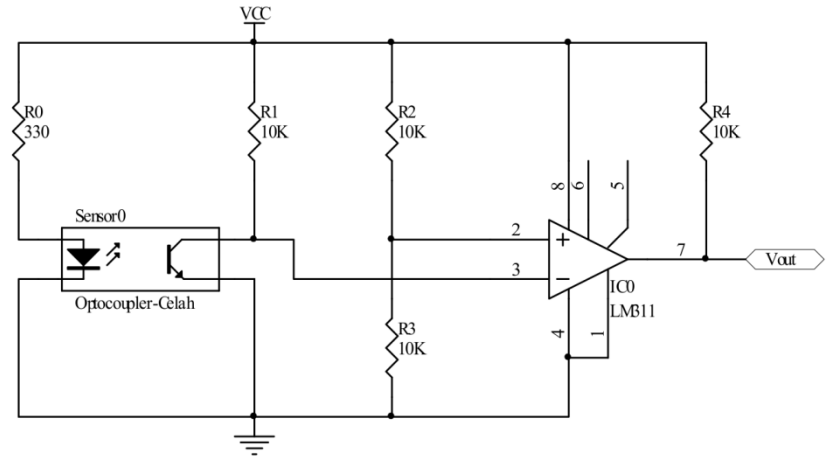


マブチモーター株式会社（本社 営業部）
MABUCHI MOTOR CO., LTD.
（Headquarters Sales Dept. | 公司　部）

千葉県松戸市松飛台430番地 〒270-2280 Tel. 047-710-1106 Fax. 047-710-1132
430 Matsuhidai, Matsudo City, Chiba 270-2280, Japan Tel. 81-47-710-1106 Fax. 81-47-710-1132
E-mail : slsinq@mabuchi-motor.co.jp

Seulas Teori:

- Optocoupler adalah komponen elektronik yang terdiri dari dua komponen penyusun:
  1. LED-IR, yaitu LED yang memancarkan cahaya inframerah yang panjang gelombangnya tak tampak oleh mata manusia.
  2. Phototransistor, yaitu transistor yang penyulut-basisnya adalah cahaya inframerah.
- Rangkaian penggerak (*driver*) optocoupler haruslah terdiri dari dua bagian:
  1. Sumber tegangan untuk menyalakan LED-IR.
  2. Rangkaian pengkondisi sinyal untuk membaca data keluaran dari phototransistor.

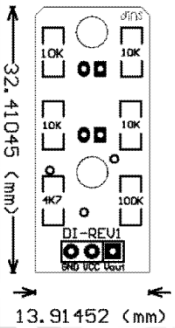


Gambar 3. Skematik Rangkaian Penggerak Optocoupler.

==========

Untuk pertanyaan, kritik, dan atau saran hubungi:

Depok Instruments
Jl. Joglo No.14 (Samping UI)
Kukusan Kelurahan, Beji
Depok 16425
Email: kotakpos@depokinstruments.com
Situs: http://depokinstruments.com



Gambar 2. Rangkaian Sensor DI-REV1.

- Dimensi:
  - Piringan Derajat: 42,64mm(Ø) x 1,9mm(Z)
  - Rangkaian Sensor: 13,91mm(X) x 32,41(Y) x 1,9mm(Z)
- Keterangan Fungsi Pin Rangkaian Sensor:

Tabel 1. Fungsi Pin Rangkaian Sensor DI-REV1.

GND	Sumber tegangan bawah / negatif / *ground*
VCC	Sumber tegangan atas / positif.
V_{out}	Data keluaran rangkaian sensor

Aplikasi:

- Penghitung rotasi:
  - Putaran motor
  - Mekanik (seperti roda, roda gigi, dan kincir)

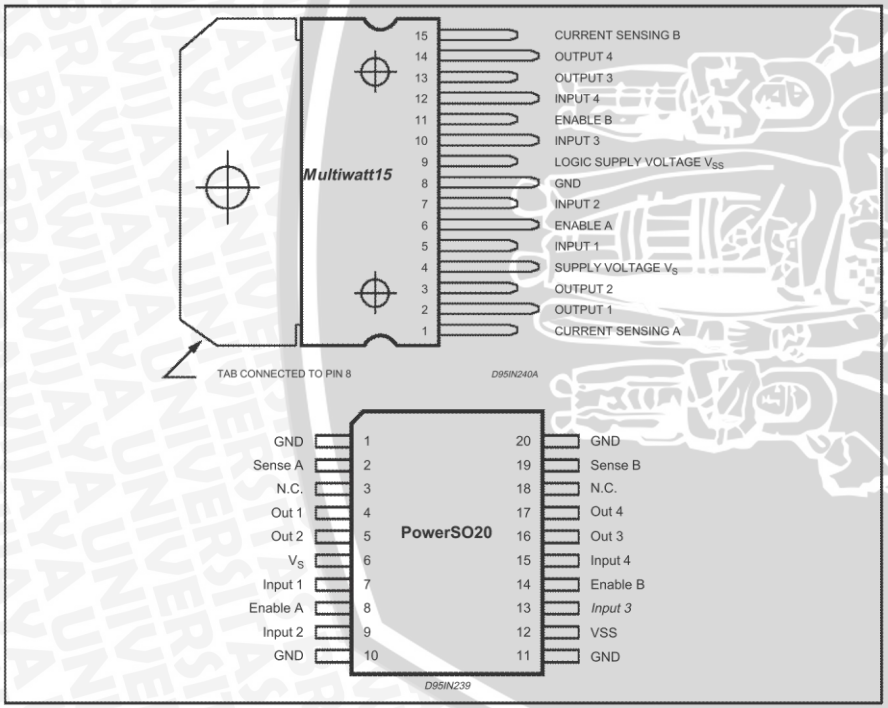
Petunjuk Penggunaan:

1. Pasang piringan derajat pada objek yang akan dihitung rotasinya. Pastikan piringan derajat terpasang dengan baik, kuat dan lurus.
2. Letakkan rangkaian sensor pembaca pada posisi dengan piringan derajat tepat berada di antara celah sensor optocoupler.
3. Beri sumber tegangan (lihat Spesifikasi dan Tabel 1).
4. Hubungkan Vout pada sistem pencacah pulsa seperti mikrokontroler.
5. Sistem Anda telah siap untuk menghitung rotasi.

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I,V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75 (5)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	5C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	5C

PIN CONNECTIONS (top view)



THERMAL DATA

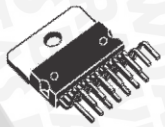
Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	Max.	–	3	5C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	5C/W

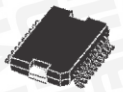
(*) Mounted on aluminum substrate

L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL °0° INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)


Multiwatt15


PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

BLOCK DIAGRAM

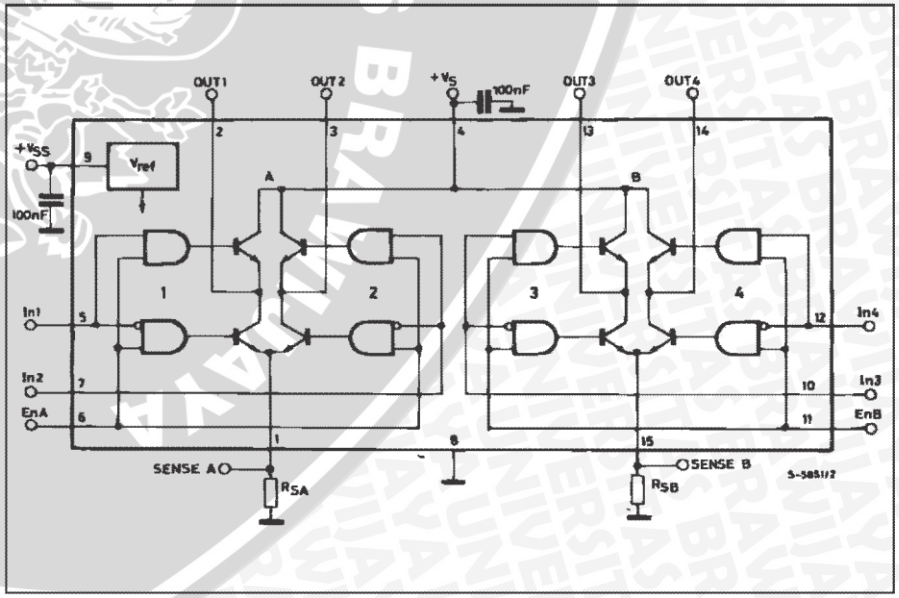


Figure 5 : Sink Current Delay Times vs. Input 0 V Enable Switching.

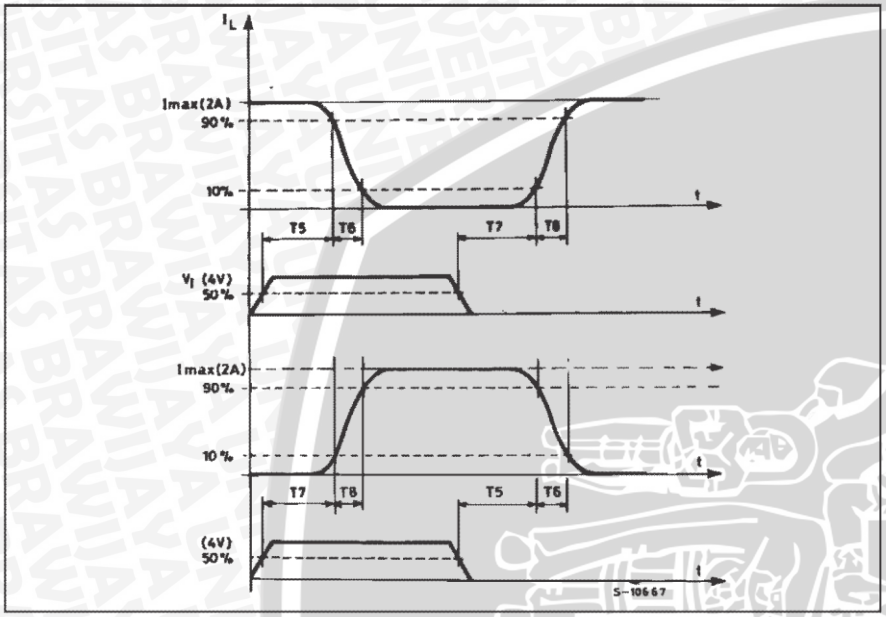
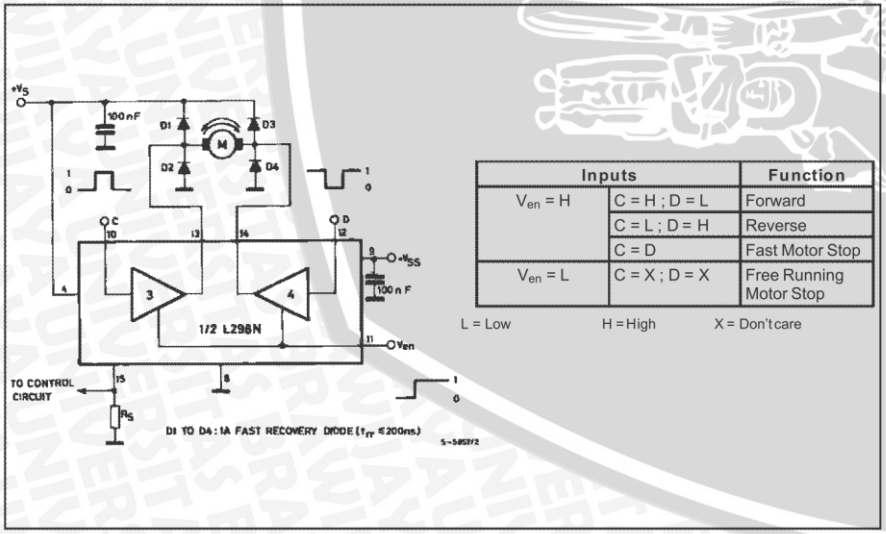


Figure 6 Bidirectional DC Motor Control.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

PIN FUNCTIONS(refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V_S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3;18	N.C.	Not Connected (Vs

ELECTRICAL CHARACTERISTICS = 42V; V_{SS} = 5V, T_j = 25 6; unless otherwise specified)

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 4)	Operative Condition		V_{IH} +2.5		46	V
V_{SS}	Logic Supply Voltage (pin 9)			4.5	5	7	V
I_S	Quiescent Supply Current (pin 4)	V_{en} = H; I_L = 0	V_i = L V_i = H		13 50	22 70	mA mA
		V_{en} = L	V_i = X			4	mA
I_{SS}	Quiescent Current from V_{SS} (pin 9)	V_{en} = H; I_L = 0	V_i = L V_i = H		24 7	36 12	mA mA
		V_{en} = L	V_i = X			6	mA
V_{iL}	Input Low Voltage (pins 5, 7, 10, 12)			–0.3		1.5	V
V_{iH}	Input High Voltage (pins 5, 7, 10, 12)			2.3		VSS	V
I_{iL}	Low Voltage Input Current (pins 5, 7, 10, 12)	V_i = L				–10	mA
I_{iH}	High Voltage Input Current (pins 5, 7, 10, 12)	Vi = H 3 V_{SS} –0.6V			30	100	mA
V_{en} = L	Enable Low Voltage (pins 6, 11)			–0.3		1.5	V
V_{en} = H	Enable High Voltage (pins 6, 11)			2.3		V_{SS}	V
I_{en} = L	Low Voltage Enable Current (pins 6, 11)	V_{en} = L				–10	mA
I_{en} = H	High Voltage Enable Current (pins 6, 11)	V_{en} = H 3 V_{SS} –0.6V			30	100	mA
$V_{CEsat(H)}$	Source Saturation Voltage	I_L = 1A I_L = 2A		0.95	1.35 2	1.7 2.7	V V
$V_{CEsat(L)}$	Sink Saturation Voltage	I_L = 1A (5) I_L = 2A (5)		0.85	1.2 1.7	1.6 2.3	V V
V_{CEsat}	Total Drop	I_L = 1A (5) I_L = 2A (5)		1.80		3.2 4.9	V V
V_{sens}	Sensing Voltage (pins 1, 15)			–1 (1)		2	V

"done" LED will illuminate after the FPGA has been successfully configured. For further information on using Adept, please see the Adept documentation available at the Digilent website.

The Nexys2 board can also be programmed using Xilinx's iMPACT software by connecting a suitable programming cable to the JTAG header. Digilent's JTAG3 cable or any other Xilinx cable may be used.

A demonstration configuration is loaded into the Platform Flash on the Nexys2 board during manufacturing. That configuration, also available on the Digilent webpage, can be used to check all of the devices and circuits on the Nexys2 board.

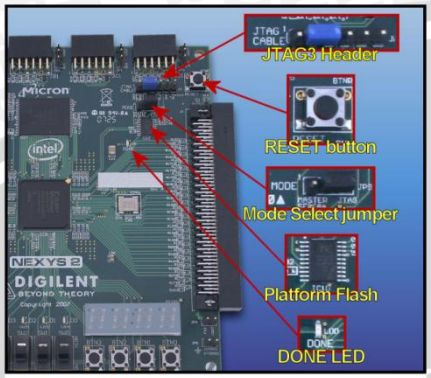


Figure 5: Nexys2 board programming circuits

Clocks

The Nexys2 board includes a 50MHz oscillator and a socket for a second oscillator. Clock signals from the oscillators connect to global clock input pins on the FPGA so they can drive the clock synthesizer blocks available in FPGA. The clock synthesizers (called DLLs, or delay locked loops) provide clock management capabilities that include doubling or quadrupling the input frequency, dividing the input frequency by any integer multiple, and defining precise phase and delay relationships between various clock signals.

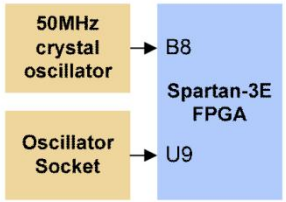


Figure 6: Nexys2 clocks

User I/O

The Nexys2 board includes several input devices, output devices, and data ports, allowing many designs to be implemented without the need for any other components.

Figure 7: Nexys2 board I/O devices

Inputs: Slide Switches and Pushbuttons

Four pushbuttons and eight slide switches are provided for circuit inputs. Pushbutton inputs are normally low, and they are driven high only when the pushbutton is pressed. Slide switches generate constant high or low inputs depending on their position. Pushbutton and slide switch inputs use a

Digilent Nexys2 Board Reference Manual

Revision: July 11, 2011

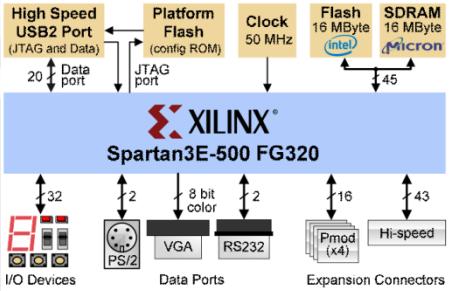
DIGILENT
www.digilentinc.com
215 E Main Suite D | Pullman, WA 99163
(509) 334 6306 Voice and Fax

Overview

The Nexys2 circuit board is a complete, ready-to-use circuit development platform based on a Xilinx Spartan 3E FPGA. Its on-board high-speed USB2 port, 16Mbytes of RAM and ROM, and several I/O devices and ports make it an ideal platform for digital systems of all kinds, including embedded processor systems based on Xilinx's MicroBlaze. The USB2 port provides board power and a programming interface, so the Nexys2 board can be used with a notebook computer to create a truly portable design station.

The Nexys2 brings leading technologies to a platform that anyone can use to gain digital design experience. It can host countless FPGA-based digital systems, and designs can easily grow beyond the board using any or all of the five expansion connectors. Four 12-pin Peripheral Module (Pmod) connectors can accommodate up to eight low-cost Pmods to add features like motor control, A/D and D/A conversion, audio circuits, and a host of sensor and actuator interfaces. All user-accessible signals on the Nexys2 board are ESD and short-circuit protected, ensuring a long operating life in any environment.

The Nexys2 board is fully compatible with all versions of the Xilinx ISE tools, including the free WebPack. Now anyone can build real digital systems for less than the price of a textbook.



- 500K-gate Xilinx Spartan 3E FPGA
- USB2-based FPGA configuration and high-speed data transfers (using the free Adept Suite Software)
- USB-powered (batteries and/or wall-plug can also be used)
- 16MB of Micron PSDRAM &16MB of Intel StrataFlash ROM
- Xilinx Platform Flash for nonvolatile FPGA configurations
- Efficient switch-mode power supplies (good for battery powered applications)
- 50MHz oscillator plus socket for second oscillator
- 60 FPGA I/O's routed to expansion connectors (one high-speed Hirose FX2 connector and four 6-pin headers)
- 8 LEDs, 4-digit 7-seg display, 4 buttons, 8 slide switches
- Ships in a plastic carry case with USB cable

Figure 1: Nexys2 block diagram and features

Power Supplies

The Nexys2 board input power input bus can be driven from a USB cable, from a 5VDC-15VDC, center positive, 2.1mm wall-plug supply, or from a battery pack. A shorting block loaded on the "power select" jumper selects the power source. The USB circuitry is always powered from the USB cable – if no USB cable is attached, the USB circuitry is left unpowered.

The anodes of the seven LEDs forming each digit are tied together into one “common anode” circuit node, but the LED cathodes remain separate. The common anode signals are available as four “digit enable” input signals to the 4-digit display. The cathodes of similar segments on all four displays are connected into seven circuit nodes labeled CA through CG (so, for example, the four “D” cathodes from the four digits are grouped together into a single circuit node called “CD”). These seven cathode signals are available as inputs to the 4-digit display. This signal connection scheme creates a multiplexed display, where the cathode signals are common to all digits but they can only illuminate the segments of the digit whose corresponding anode signal is asserted.

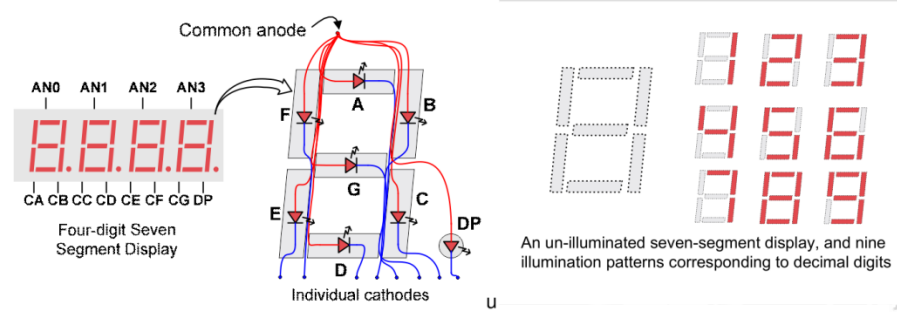
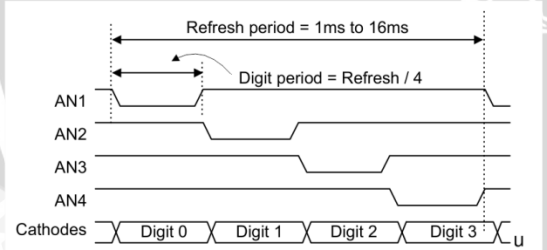


Figure 9: Nexys2 seven-segment displays

A scanning display controller circuit can be used to show a four-digit number on this display. This circuit drives the anode signals and corresponding cathode patterns of each digit in a repeating, continuous succession, at an update rate that is faster than the human eye can detect. Each digit is illuminated just one-quarter of the time, but because the eye cannot perceive the darkening of a digit before it is illuminated again, the digit appears continuously illuminated. If the update or “refresh” rate is slowed to around 45 hertz, most people will begin to see the display flicker.

In order for each of the four digits to appear bright and continuously illuminated, all four digits should be driven once every 1 to 16ms, for a refresh frequency of 1KHz to 60Hz. For example, in a 60Hz refresh scheme, the entire display would be refreshed once every 16ms, and each digit would be illuminated for ¼ of the refresh cycle, or 4ms. The controller must drive the cathodes with the correct pattern when the corresponding anode signal is driven. To illustrate the process, if AN0 is asserted while CB and CC are asserted, then a “1” will be displayed in digit position 1. Then, if AN1 is asserted while CA, CB and CC are asserted, then a “7” will be displayed in digit position 2. If AN0 and CB, CC are driven for 4ms, and then A1 and CA, CB, CC are driven for 4ms in an endless succession, the display will show “17” in the first two digits. An example timing diagram for a four-digit controller is provided.



Figureu41zuSevenvsegmentudisplayutimingudiagramu

series resistor for protection against short circuits (a short circuit would occur if an FPGA pin assigned to a pushbutton or slide switch was inadvertently defined as an output).

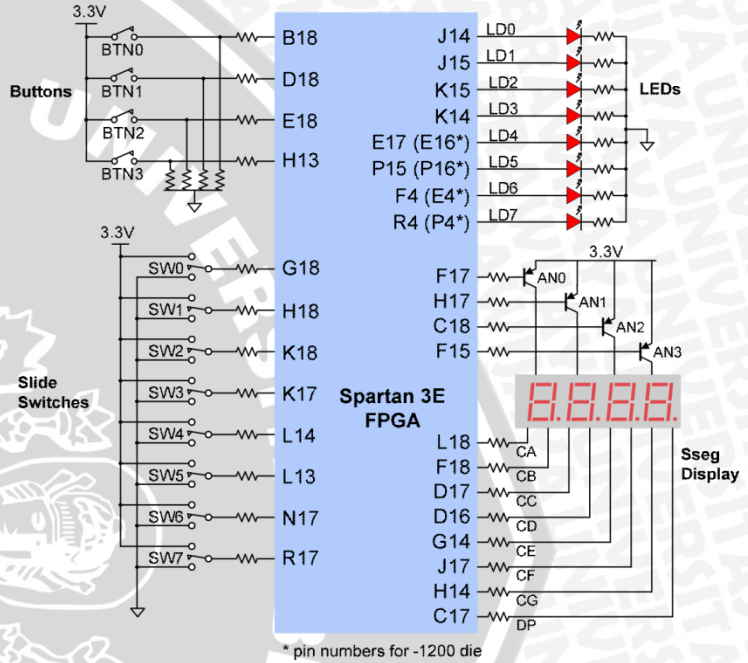


Figure 8: Nexys2 I/O devices and circuits

Outputs: LEDs

Eight LEDs are provided for circuit outputs. LED anodes are driven from the FPGA via 390-ohm resistors, so a logic ‘1’ output will illuminate them with 3-4ma of drive current. A ninth LED is provided as a power-on LED, and a tenth LED indicates FPGA programming status. Note that LEDs 4-7 have different pin assignments due to pinout differences between the -500 and the -1200 die.

Outputs: Seven-Segment Display

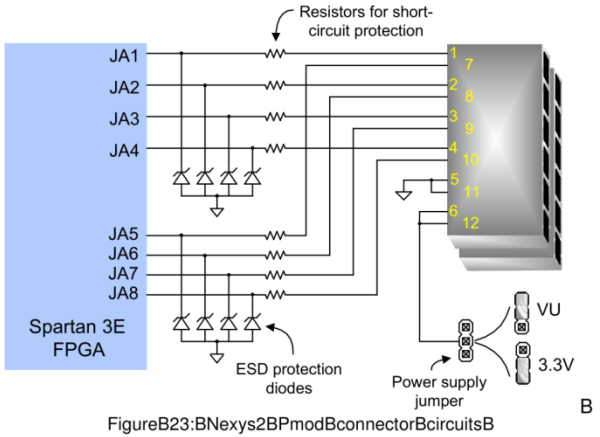
The Nexys2 board contains a four-digit common anode seven-segment LED display. Each of the four digits is composed of seven segments arranged in a “figure 8” pattern, with an LED embedded in each segment. Segment LEDs can be individually illuminated, so any one of 128 patterns can be displayed on a digit by illuminating certain LED segments and leaving the others dark. Of these 128 possible patterns, the ten corresponding to the decimal digits are the most useful.

Peripheral Connectors

The Nexys2 board provides four two-row 6-pin Pmod connectors that together can accommodate up to 8 Pmods. The four 12-pin connectors each have 8 data signals, two GND pins, and two Vdd pins. All data signals include short circuit protection resistors and ESD protection Diodes. A jumper block adjacent to each Pmod connector can connect the Pmod's Vdd signal to the Nexys2 board's 3.3V supply or to the input power bus (VU). If the jumper is set to VU and USB power is driving the main power bus, care should be taken to ensure no more than 200mA is consumed by the Pmod. Further, if the jumper is set to VU, a voltage source connected to the Pmod can drive the main power bus of the Nexys2 board, so care should be taken to avoid connecting conflicting power supplies.

The Pmod connectors are labeled JA (nearest the power jack), JB, JC, and JD (nearest the expansion connector). Pinouts for the Pmod connectors are provided in the table below.

More than 30 low-cost are available for attachment to these connectors. Pmods can either be attached directly, or by using a small cable. Available Pmods include A/D and D/A converters, motor drivers, speaker amplifiers, distance measuring devices, etc. Please see www.digilentinc.com for more information.



FigureB23:BNexys2BPmodBconnectorBcircuitsB

Table 3: Nexys2 Pmod Connector Pin Assignments

Pmod JA		Pmod JB		Pmod JC		Pmod JD	
JA1: L15	JA7: K13	JB1: M13	JB7: P17	JC1: G15	JC7: H15	JD1: J13	JD7: K14[1]
JA2: K12	JA8: L16	JB2: R18	JB8: R16	JC2: J16	JC8: F14	JD2: M18	JD8: K15[2]
JA3: L17	JA9: M14	JB3: R15	JB9: T18	JC3: G13	JC9: G16	JD3: N18	JD9: J15[3]
JA4: M15	JA10: M16	JB4: T17	JB10: U18	JC4: H16	JC10: J12	JD4:; P18	JD10: J14[4]

Notes: [1] shared with LD3 [2] shared with LD3 [3] shared with LD3 [4] shared with LD3