APLIKASI FPGA SEBAGAI PENDETEKSI PARAMETER TANAH DENGAN METODE FUZZY LOGIC

TUGAS AKHIR

Disusun Oleh :

ANDY HARYONO KARTIKA

NRP. 2295 100 104

JURUSAN TEKNIK ELEKTRO
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

2000

APLIKASI FPGA SEBAGAI PENDETEKSI PARAMETER TANAH DENGAN METODE FUZZY LOGIC

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik Elektro
Pada
Bidang Studi Elektronika
Jurusan Teknik Elektro
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya

Mengetahui / Menyetujui
Dosen Pembimbing

Ir. SOETIKNO
NIP. 130 445 231

SURABAYA
Pebruari, 2000

ABSTRAK

ABSTRAK

Supaya tanaman dapat tumbuh dengan subur, maka kita harus memperhatikan dan menjaga parameter-parameter yang berhubungan dengan tanah agar tetap optimal. Beberapa parameter tersebut seperti kelembaban tanah dan suhu udara. Untuk menjaga parameter-parameter tersebut diperlukan suatu alat untuk mendeteksi dan menjaga agar parameter-parameter tersebut tetap optimal bagi tanaman.

Alat pengatur tersebut kami rancang dengan menggunakan metode logika samar *(fuzzy logic)*, sebab metode ini amat berguna untuk mengatasi masalah-masalah yang non linier dan dinamis sehingga diperoleh hasil yang akurat, lebih mudah dan cepat. Pembuatan alat pendeteksi parameter tanah ini menggunakan teknologi FPGA, sehingga diperoleh rangkaian yang sederhana dan memiliki densitas yang tinggi.

Implementasi Tugas Akhir ini menggunakan *software Xilinx Foundation Series 1.5* dengan menggunakan *design entry HDL editor* (menggunakan bahasa pemrograman VHDL) dan Synopsys FPGA Express 3.2 untuk menganalisa dan mensintesa rancangan, serta *Xilinx Foundation Series 1.5* untuk proses implementasi. Sedangkan perangkat keras (*hardware*) yang digunakan adalah *XS40-010XL Board* yang menggunakan IC FPGA XC4010XL dengan kapasitas 400 CLB (*Configurable Logic Block*) atau setara dengan 10,000 gerbang logika.

KATA PENGANTAR

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa atas berkat rahmat-Nya, sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

APLIKASI FPGA SEBAGAI PENDETEKSI PARAMETER TANAH DENGAN METODE *FUZZY LOGIC*

Tugas Akhir ini merupakan salah satu syarat dalam menyelesaikan pendidikan di Jurusan Teknik Elektro Fakultas Teknologi Industri Institut Teknologi Sepuluh Nopember Surabaya.

Tugas Akhir ini dibuat berdasarkan teori-teori yang diperoleh pada bangku kuliah dan berbagai *literature* penunjang lainnya.

Dengan selesainya Tugas Akhir ini, penyusun ingin menyampaikan terima kasih kepada :

1. Bapak Ir. Soetikno, selaku Koordinator Bidang Studi Elektronika Jurusan Teknik Elektro FTI-ITS dan selaku Dosen Pembimbing.
2. Bapak Ir. Achmad Jazidie, selaku Ketua Jurusan Teknik Elektro FTI-ITS.
3. Bapak Tri Arief, ST, selaku Dosen Wali.
4. Seluruh Staf Dosen Bidang Studi Elektronika Jurusan Teknik Elektro FTI-ITS.
5. Seluruh Staf dan Karyawan di Jurusan Teknik Elektro FTI-ITS.
6. Teman-teman di Bidang Studi Elektronika Jurusan Teknik Elektro FTI-ITS.

Atas segala bantuan, dorongan, dan bimbingan yang telah diberika kepada penulis.

Penyusun berharap kiranya Tugas Akhir ini dapat bermanfaat bagi pembaca, khususnya teman-teman di Bidang Studi Elektronika Jurusan Teknik Elektro Fakultas Teknologi Industri Institut Teknologi Sepuluh Nopember Surabaya.

Surabaya, Januari 1999

Penulis

DAFTAR ISI

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR TABEL

BAB I
PENDAHULUAN

BAB I

PENDAHULUAN

1.1 LATAR BELAKANG

Perkembangan teknologi dewasa ini sangat pesat sekali dan meliputi segala bidang ilmu yang menyangkut segala sisi kehidupan manusia. Teknologi elektronika merupakan salah satu bidang teknologi yang berkembang sangat pesat. Dalam era globalisasi ini, perkembangan teknologi elektronika telah menyentuh semua bidang kehidupan dan berkaitan dengan teknologi-teknologi lainnya. Kehadiran teknologi telah membuat kegiatan-kegiatan atau proses-proses yang dulunya dilakukan secara manual menjadi suatu proses yang otomatis, cepat, akurat serta lebih efisien.

Kemajuan-kemajuan dalam bidang teknologi elektronika yang menuju ke arah sistem digital serta rangkaian terpadu telah mendorong terciptanya IC-IC yang semakin cepat dengan kemampuan proses yang tinggi dan akurat serta bentuk yang semakin kecil, tetapi dengan kemampuan yang lebih besar. Dengan kemajuan teknologi elektronika ini, maka pekerjaan-pekerjaan yang menggunakan sistem konvensional secara bertahap dapat digantikan dengan sistem yang canggih.

Penerapan teknologi elektronika dalam bidang teknologi yang berkaitan dengan bidang pertanian (*agricultural engineering)* sangat banyak, antara lain

adalah sistem pengairan. Proses irigasi yang dulunya dilakukan secara konvensional, saat ini sudah dapat digantikan dengan cara yang lebih canggih dan otomatis serta memiliki keakurasian yang tinggi sehingga prosesnya lebih efisien. Hal ini dapat dilakukan dengan cara membuat suatu alat yang mampu mengatur sistem irigasi berdasarkan masukan yang berasal dari kelembaban tanah dan suhu udara pada tanah yang akan diairi.

1.2 PERMASALAHAN

Agar tanaman dapat tumbuh dengan baik, maka direncanakan suatu alat pendeteksi kelembaban tanah dan suhu udara dengan menggunakan IC FPGA. Input dari IC FPGA ini berasal dari sensor kelembaban tanah dan sensor suhu udara yang ada pada sebidang tanah. Kedua sensor berfungsi sebagai input dari sebuah alat pengatur yang akan menjalankan proses dengan menggunakan metode logika samar (*fuzzy logic)* yang diterapkan dalam IC FPGA XC4010XL.

1.3 BATASAN MASALAH

Alat yang direncanakan dibatasi untuk mendeteksi tingkat kelembaban tanah dan suhu udara. Alat pendeteksi ini diatur oleh IC FPGA XC4010XL yang berfungsi sebagai panel kontrol, yang nantinya akan berfungsi untuk menentukan kerja sistem irigasi secara keseluruhan. Jadi selain sebagai alat pendeteksi kelembaban tanah dan suhu udara, IC FPGA ini juga berfungsi sebagai alat pengatur yang bekerja berdasarkan metode logika samar (*fuzzy logic).*

1.4 TUJUAN

Perencanaan dan pembuatan **APLIKASI FPGA SEBAGAI PENDETEKSI PARAMETER TANAH DENGAN METODE *FUZZY LOGIC***, bertujuan agar agar sistem irigasi pada suatu bidang tanah dapat diatur sehingga tanaman dapat tumbuh dengan baik.

Selain tujuan diatas, perencanaan dan pembuatan Tugas Akhir ini bertujuan pula untuk mempelajari proses akuisisi data, IC FPGA XC4010XL, dan pemrograman VHDL, yang diharapkan dapat digunakan baik dalam peralatan ini maupun dalam peralatan yang membutuhkannya.

1.5 METODOLOGI

Pengerjaan Tugas Akhir ini dilakukan dengan langkah-langkah sebagai berikut :

- Studi literatur, yang meliputi pengukuran kelembaban tanah dan suhu udara, IC FPGA XC4010XL, *Analog To Digital Converter* MAX114, PWM (Pulse Width Modulation) Driver, LCD (*Liquid Crystal Display*) serta Motor DC.
- Perencanaan perangkat keras (*hardware*) luar dan perencanaan FPGA (*Field Programmable Gate Array*) dengan menggunakan pemrograman VHDL.
- Realisasi perangkat keras (*hardware*) dan perencanaan FPGA sesuai dengan yang direncanakan.
- Pengujian dan pengkalibrasian peralatan yang dibuat serta pengukuran dari dari hasil pemantauan peralatan.
- Penulisan buku Tugas Akhir

1.6 SISTEMATIKA

Dalam buku Tugas Akhir ini, pembahasan mengenai peralatan yang dibuat terbagi lima bab dengan sistematika seperti dijelaskan berikut ini.

Bab I merupakan Pendahuluan yang membahas latar belakang, tujuan serta permasalahan dan pembatasan masalah dari pembuatan peralatan. Dalam bab ini juga dibahas metodologi dan sistematika penulisan Tugas Akhir ini.

Bab II berisi penjelasan dasar teori mengenai sensor kelembaban tanah dan suhu udara, IC FPGA XC4010XL, akuisisi data menggunakan ADC MAX114, LCD (*Liquid Crystal Display*), Motor DC.

Bab III berisi perencanaan perangkat keras dan perencanaan FPGA yang menyangkut diagram blok, algoritma, gambar rangkaian dan penjelasannya.

Bab IV berisi karakteristik dan batasan kemampuan dari peralatan yang diperoleh dari pengujian peralatan serta hasil pengukuran dari peralatan.

Bab V merupakan Bab Penutup yang berisi kesimpulan yang diperoleh dari pembuatan Tugas Akhir ini, serta saran-saran untuk pengembangan dari Tugas Akhir ini.

1.7 RELEVANSI

Hasil Tugas Akhir ini diharapkan dapat membantu semua pihak yang membutuhkan, terutama yang bergerak di bidang pertanian agar dapat memperoleh hasil tanaman yang baik.

BAB II
TEORI PENUNJANG

BAB II

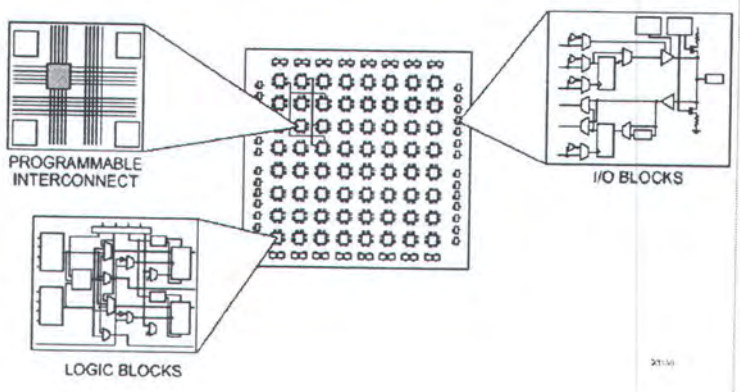
TEORI PENUNJANG

Sebagai pendukung dalam Tugas Akhir ini, maka penulis akan membahas mengenai dasar teori yang meliputi IC FPGA (*Field Programmable Gate Array)*, *Fuzzy Logic*, Sensor Suhu Udara, Sensor Kelembaban Tanah, dan *Analog To Digital Converter.*

2.1 IC FPGA (*Field Programmable Gate Array*)

2.1.1 PENDAHULUAN

FPGA (*Field Programmable Gate Arrays*) adalah sebuah IC *Programmable Logic* yang memiliki arsitektur seperti susunan matriks sel-sel logika yang dapat berhubungan satu sama lain melalui jalur-jalur I/O dengan menggunakan *channel routing*.



Gambar 2.1 Arsitektur umum FPGA

Sel-sel logika yang dimaksud disebut dengan CLB (*Configurable Logic Block*), sedangkan jalur-jalur I/O dinamakan *Programmable Interconnect*, dan untuk menghubungkan IC FPGA dengan dunia luar digunakan kumpulan I/O yang dinamakan IOB (*Input Output Block*).

Pada dasarnya IC-IC FPGA keluaran Xilinx memiliki dua buah elemen konfigurasi utama, yaitu :

- CLB (*Configurable Logic Block*), berfungsi untuk mengimplementasikan fungsi-fungsi logika yang akan dibuat.
- IOB (*Input Output Block*), berfungsi sebagai blok interface dengan rangkaian di luar IC FPGA.

Selain itu, terdapat pula tiga tipe rangkaian yang mendukung arsitektur IC FPGA Xilinx, yakni :

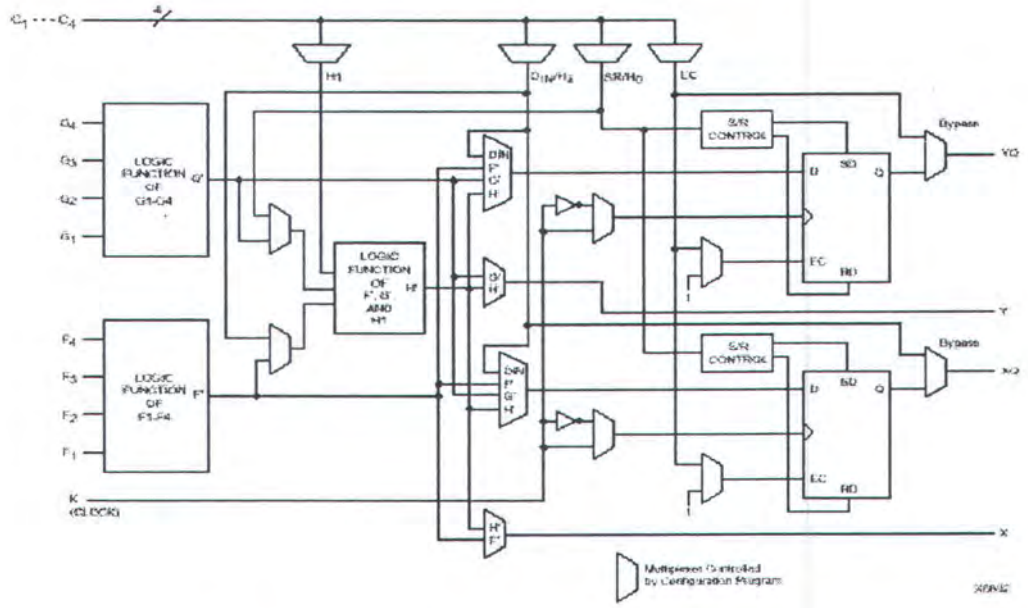
- *Three State Buffer* (TBUF), yang berfungsi sebagai pengatur jalur-jalur horizontal yang berhubungan dengan CLB.
- *Wide Edge Decoder*, yang ada pada setiap IOB dan berfungsi sebagai dekoder alamat yang memiliki banyak bit.
- *On-chip Oscillator*, yang berfungsi menghasilkan sinyal *clock* dengan frekuensi-frekuensi tertentu.

2.1.2 CONFIGURABLE LOGIC BLOCK (CLB)

CLB berfungsi untuk mengimplementasikan sebagian besar dari fungsi-fungsi logika dalam sebuah IC FPGA. Prinsip dasar dari elemen CLB ini dapat dilihat pada gambar 2.2, dimana terdapat 2 buah *Function Generator* 4 masukan

(F dan G), yang berfungsi mengimplementasikan fungsi-fungsi logika. Selain itu terdapat pula *Function Generator* ketiga (H) yang memiliki 3 masukan. Ketiga masukan tersebut dapat berasal dari keluaran *Function Generator* F dan G, sedangkan keluaran lainnya dapat pula berasal dari luar CLB.

Setiap CLB memiliki 2 elemen penyimpan. Elemen penyimpan dan *Function Generator* ini dapat digunakan sendiri-sendiri. Elemen penyimpan ini dapat dikonfigurasikan sebagai *flip-flop* untuk tipe XC4000E dan XC4000X, serta sebagai *latch* khusus untuk tipe XC4000X. DIN (masukan langsung) dapat digunakan sebagai masukan kedua elemen penyimpan ini. H1 dapat men*drive* masukan yang lain melalui *Function Generator* H. Keluaran *Function Generator* dapat men*drive* kedua elemen penyimpan tersebut sendiri-sendiri.



Gambar 2.2 Diagram Blok Sederhana CLB dari XC4000

Function Generator

Masing-masing *Function Generator* memiliki 4 buah masukan (F1-F4 dan G1–G4) dan memiliki 1 buah keluaran, dengan label F' dan G', dan memiliki

kemampuan untuk mengimplementasikan fungsi logika yang mempunyai 4 buah masukan. *Function Generator* ini diimplementasikan sebagai *memory look-up table*. *Function Generator* ketiga (H) dapat mengimplementasikan fungsi logika dengan tiga masukan. Dua dari tiga masukan tersebut dapat berasal dari F' dan G' atau salah satu dari masukan tersebut dari luar CLB (H0 dan H2). Sedangkan masukan ketiga harus berasal dari luar CLB (H1).

Sinyal dari *Function Generator* dapat keluar dari CLB melalui dua buah keluaran. F' atau H' dapat keluar CLB melalui pin keluaran X. G' atau H' dapat keluar CLB melalui pin keluaran Y.

Dari penjelasan diatas, maka sebuah CLB dapat dibuat menjadi

- Sembarang fungsi dengan 4 masukan dan fungsi kedua dengan 4 masukan berbeda ditambah fungsi ketiga dengan masukan yang lain. (bila 3 fungsi berbeda dibuat, maka sebuah keluaran fungsi harus *dicapture* ke *flip-flop internal* CLB.
- *Single function* hingga 5 variabel.
- Sembarang fungsi dengan 4 variabel ditambah beberapa fungsi 6 variabel.
- Beberapa fungsi hingga 9 variabel.

Flip-Flop

Selain dapat dihubungkan ke *interconect network*, keluaran CLB juga dapat dihubungkan ke salah satu atau kedua *flip-flop*. *Flip-flop* ini sendiri dapat dihubungkan juga ke *interconect network*. Dua buah *edge-triggered* D *flip-flop* memiliki masukan *clock* (K), *clock enable* (EC) bersama. *Clock Enable* dari salah satu atau keduanya dapat di-*enable* secara permanen.

Tabel 2.1 Fungsi Elemen Penyimpan CLB

Mode	K	EC	SR	D	Q
Power-Up or GSR	X	X	X	X	SR
Flip-Flop	X	X	1	X	SR
	⎍⎍	1*	0*	D	D
	0	X	0*	X	Q
Latch	1	1*	0*	X	Q
	0	1*	0*	D	D
Both	X	0	0*	X	Q

Legend:

X	Don't care
⎍⎍	Rising edge
SR	Set or Reset value. Reset is default
0*	Input is Low or unconnected (default value)
1*	Input is High or unconnected (default value)

Latch (Untuk tipe XC4000X)

Elemen penyimpan dari CLB juga dapat dikonfigurasikan sebagai *latch*. Kedua buah *latch* tersebut memiliki *clock* (K) dan *clock enable* (EX).

Clock Input

Setiap *flip-flop* dapat di*trigger* pada saat *rising edge* atau *falling edge clock*. *Clock* ini digunakan bersama dengan elemen penyimpan. Bagaimanapun, *clock* ini dapat di*invert* untuk masing-masing elemen penyimpan, dengan cara menempatkan sebuah *inverter* pada *clock* masukan.

Clock Enable

Clock enable adalah aktif "*high*". CE ini digunakan bersama oleh elemen penyimpan. Jika CE tidak dihubungkan apa-apa, maka *Clock Enable* untuk elemen penyimpan akan berada pada *state* aktif. EC tidak dapat di-*invert* di CLB.

Set/Reset

Sebuah masukan asinkronus elemen penyimpan (SR) dapat dikonfigurasikan sebagai set ataupun reset. Konfigurasi ini tergantung saat proses konfigurasi dan terpengaruh juga dari operasi *Global Set/Reset* selama proses

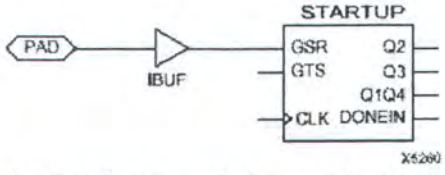
operasi normal serta juga dipengaruhi dari pin SR CLB. Ketiga fungsi *set/reset single flip-flop* dikontrol oleh *configuration data bit.*

Baik masukan dan keadaan set/reset dapat didefinisikan secara terpisah. Set/reset ini didefinisikan dengan atribut INIT, atau menempatkan set/reset *flip-flop library symbol*. SR ini aktif "*high*" dan tidak bisa di-*invert* dari CLB.

Global Set/Reset (GSR)

Jalur *Global Set/reset* yang terpisah dapat men-set atau *clear* elemen penyimpan pada saat dinyalakan, konfigurasi IC ulang, atau ketika net reset diaktifkan. *Net Global Set Reset* (GSR) ini tidak berhubungan dengan *routing resource*, namun menggunakan *dedicated distribution network.*

Setiap *flip-flop* dapat dikonfigurasikan set atau reset secara global dengan cara yang sama seperti Set/reset lokal (SR).



Gambar 2.3 Gambar Rangkaian untuk *Global Set/Reset*

Data Input dan Output

Masukan elemen penyimpan adalah F', G' dan H' atau dengan *Direct In* (DIN) blok masukan. *Flip-flop* menggerakkan blok XQ dan YQ.

Sinyal Kontrol

Multiplekser dalam CLB berfungsi untuk menempatkan 4 buah sinyal kontrol masukan (C1-C4) pada 4 buah sinyal kontrol *internal* (H1, DIN.H2, SR/H0 dan EC).

Saat *logic function enable*, keempat masukan tersebut adalah :

- EC → *Enable Clock*
- SR/H0 → *Ansynchronous set/reset* atau H0
- DIN/H2 → *Direct In* atau H2
- H1 → H1

Saat *memory function enable* :

- EC → *Enable Clock*
- WE → *Write enable*
- D0 → Data masukan ke *Function Generator* F dan/atau G.
- D1 → Data masukan ke *Function Generator* G (mode 16x1 dan 16x2) atau alamat bit kelima(mode 32 x 1)

Menggunakan FPGA Flip-Flop dan Latches

Untuk menggunakan *flip-flop* dan *latch* pada IC FPGA XC4000, dapat dilakukan dengan meletakkan simbol yang bersangkutan pada editor skematik atau pada editor HDL. Sebagai contoh, FDCE adalah D *flip-flop* dengan *enable clock* dan *asynchronous clear*. Untuk menggunakannya, maka kita harus meletakkan simbol LDCE pada editor HDL atau editor skematik.

Menggunakan Function Generator Sebagai RAM

CLB dapat digunakan sebagai sebuah susunan sel memori *Read/Write*. Mode-mode yang ada yaitu *level-sensitive, edge-triggered,* dan *dual-port triggered*. Bergantung pada mode yang dipilih, sebuah CLB dapat dikonfigurasikan sebagai susunan 16x2, 32x1, atau 16x1. Konfigurasi memori CLB dan mode *timing* untuk *single* dan *dual port* dapat dilihat pada tabel 2.2.

Tabel 2.2 Mode RAM dalam CLB

	16 x 1	16 x 2	32 x 1	Edge-Triggered Timing	Level-Sensitive Timing
Single-Port	√	√	√	√	√
Dual-Port	√			√	

Keuntungan dari penggunaan on-chip RAM ini adalah waktu aksesnya yang sangat cepat. Waktu akses baca sama dengan *delay* gerbang, sedangkan waktu akses tulisnya lebih lambat sedikit. Kedua waktu akses tersebut lebih cepat dibandingkan dengan RAM eksternal, karena tidak ada *delay I/O*.

Konfigurasi RAM

Function Generator CLB dapat dijadikan RAM dengan ukuran :

- Dua 16 x 1 RAM : 2 bit data masukan dan keluaran, dengan alamat berbeda
- Satu 32x1 RAM : 1 bit data masukan/keluaran.

Function Generator F dan G hanya dapat dikonfigurasi sebagai RAM 16x1 saja, sedangkan *Function Generator* yang lain dapat digunakan untuk mengimplementasikan persamaan logika dengan 5 masukan.

Untuk tipe XC4000, ada 2 mode timing yaitu :

- *Edge-triggered* (sinkronus) : Data ditulis pada *edge* sinyal *clock* CLB. WE berfungsi sebagai *enable clock*.
- *Level-sensitive* (ansinkronus): Sinyal WE eksternal berfungsi sebagai strobe.

Jumlah *port* baca juga dapat diprogram:

- *Single port* : setiap *Function Generator* mempunyai *common read* dan *write*.
- *Dual port* : Kedua *Function Generator* dikonfigurasikan sebagai satu 16x1 *dual port* RAM dengan sebuah *port write* dan dua buah *port read*.

Untuk memilih mode RAM dalam rancangan kita harus didasari oleh kebutuhan waktu dan banyaknya CLB yang dibutuhkan, dan juga kesederhanaan proses.

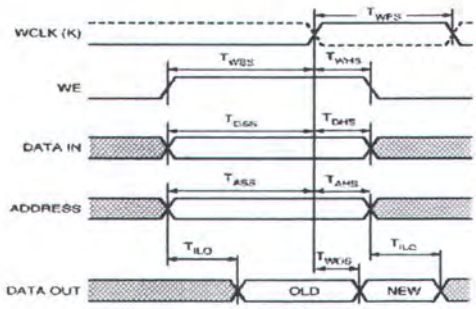
Input dan Output RAM

F1-F4 dan G1-G4 adalah alamat masukan untuk memilih sel memori di setiap *look-up table*. Jika CLB dikonfigurasikan sebagai RAM, DIN/H2, H1, dan SR/H0 menjadi dua masukan data (D0, D1) dan *Write Enable* (WE) untuk memori 16x2. Untuk mode 32x1, maka D1 sebagai alamat ke lima dan D0 sebagai masukan. Isi dari sel memori yang diakses berada pada keluaran *Function Generator* F' dan G'. Mereka dapat keluar dari CLB melalui keluaran X dan Y, atau dapat melalui *flip-flop* CLB.

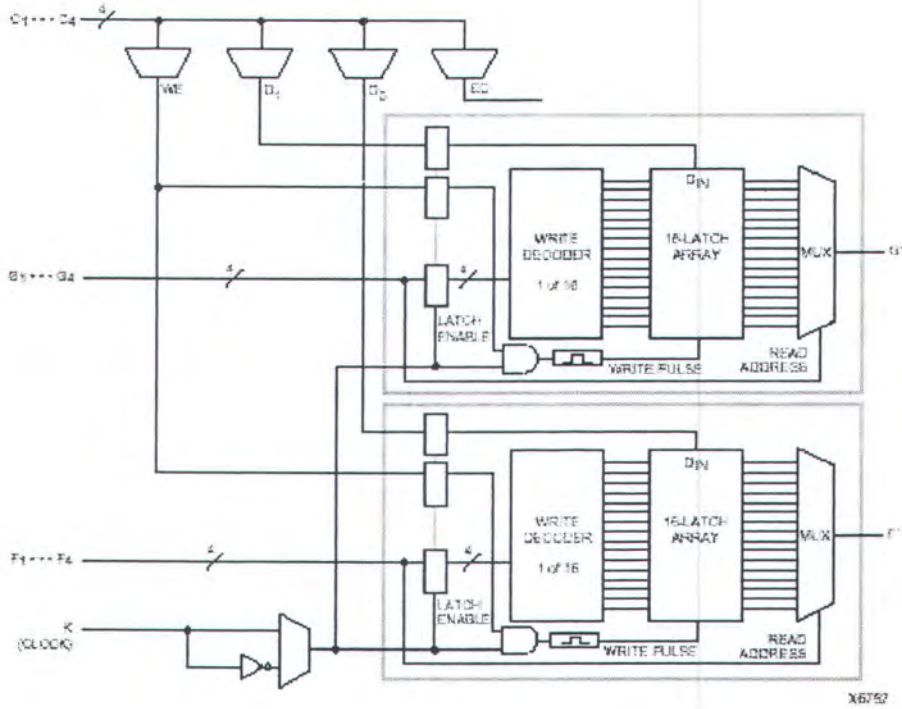
Pada saat CLB dikonfigurasikan sebagai RAM, *Function Generator* lain pada CLB yang tidak dapat digunakan untuk keperluan lain. Pada mode 16x1 dan 16x2, *Function Generator* H' dapat digunakan untuk mengimplementasikan persamaan logika dengan masukan F', G' dan D1, serta D *Flip-flop* dapat digunakan untuk meng-*latch* sinyal F', G', H', atau D0.

Single Port Edge Triggered Mode

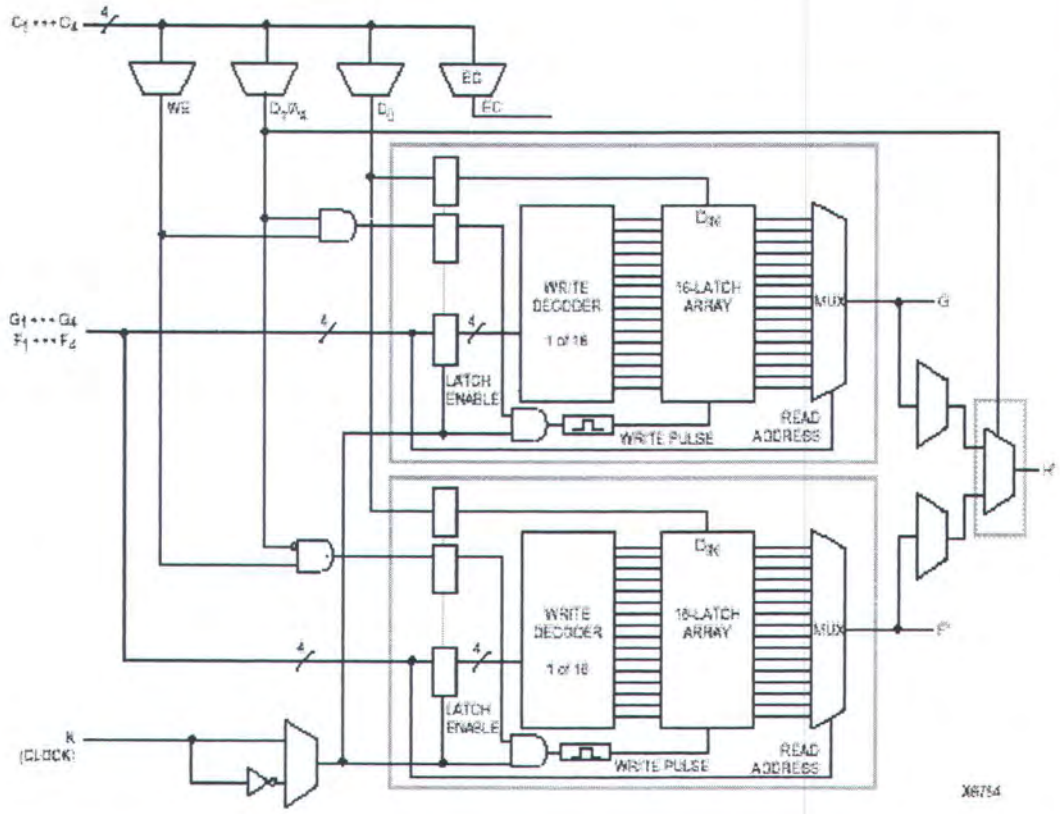
Edge triggered (sinkronus) RAM menyederhanakan kebutuhan waktu. Diagram waktu dari *Edge Triggered* RAM dapat dilihat pada gambar 2.4.



Gambar 2.4 Diagram Waktu Edge Triggered RAM



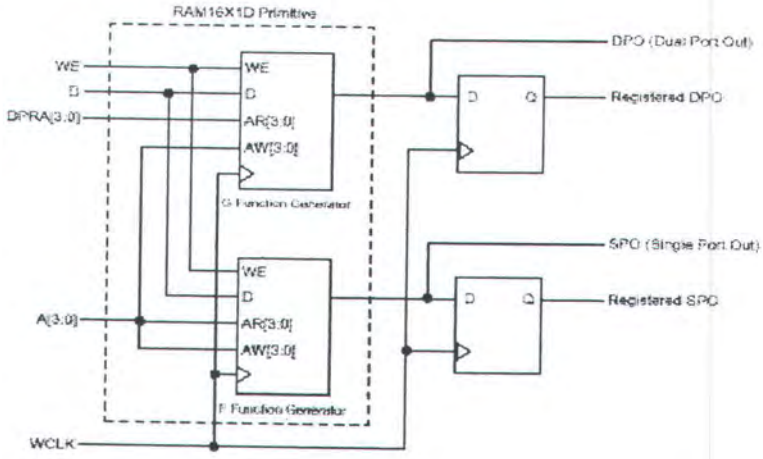
Gambar 2.5 16x2 atau 16x1 Edge-triggered Single Port RAM



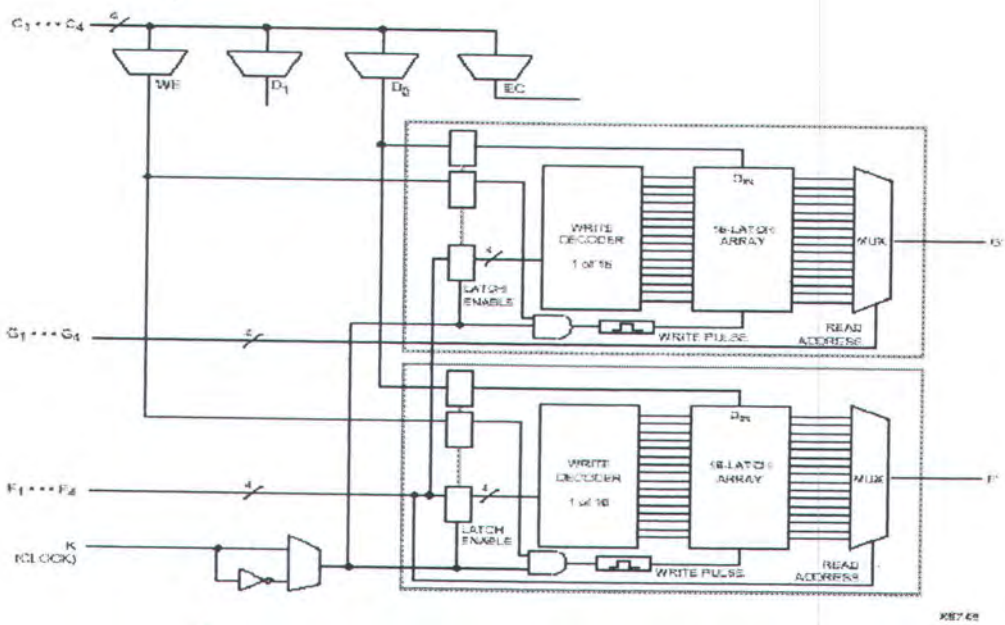
Gambar 2.6 32x1 Edge-triggered Single Port RAM.

Dual Port Edge Triggered Mode

Pada mode ini, *Function Generator* F dan G digunakan untuk membuat sebuah RAM 16x1 dengan sebuah *port* tulis dan dua buah *port* baca. RAM ini dapat dituilis dan dibaca sendiri-sendiri. Diagram waktu dari mode *dual port* ini ada pada gambar 2.4. Sedangkan Diagram Blok sederhana dari CLB XC4000 yang dikonfigurasikan sebagai *dual-port* RAM dapat dilihat pada gambar 2.7.



Gambar 2.7 Diagram Blok sederhana Dual Port RAM XC4000.



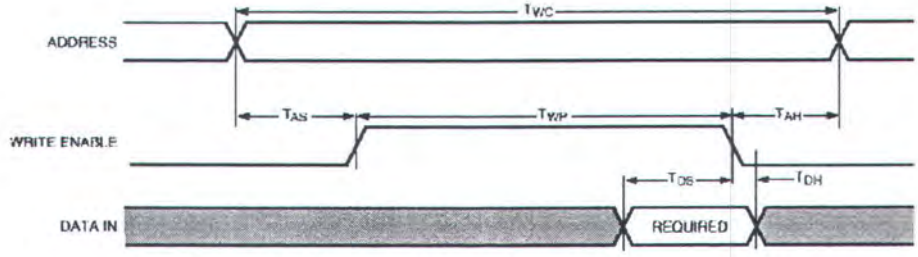
Gambar 2.8 16x1 Edge triggered Dual Port RAM.

Mode Single Port Level Sensitive Timing

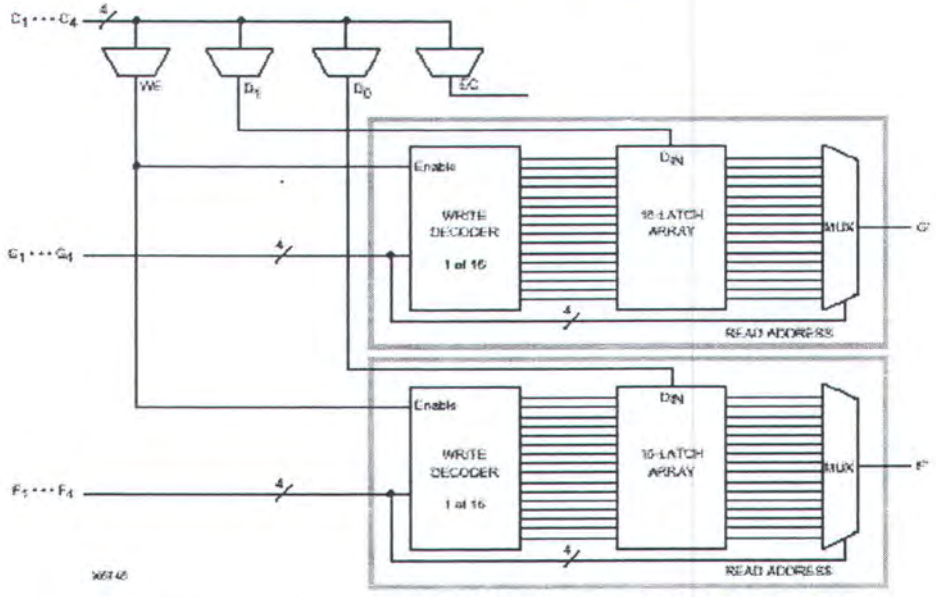
Diagram waktu dari penggunaan CLB sebagai RAM dalam mode *single port Level Sensitive* timing dapat dilihat pada gambar 2.9. Sedangkan konfigurasi dari CLB sebagai RAM tersebut dapat dilihat pada gambar 2.10 dan 2.11.

Inisialisasi RAM Saat Konfigurasi

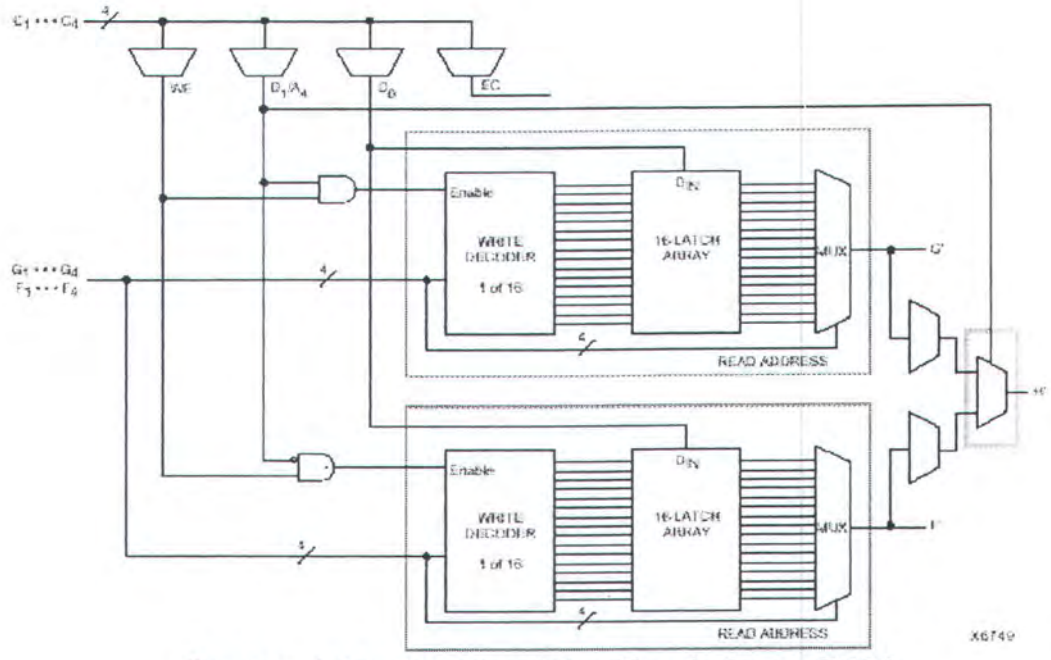
Implementasi RAM dan ROM pada XC4000 diinisialisasi pada saat konfigurasi. Pengisian awal dapat dilakukan dengan menggunakan atribut INIT yang ada pada simbol RAM dan ROM. Jika isi dari RAM tidak didefinisikan, maka semua isinya didefinisikan dengan nol. Isi dari RAM tidak dipengaruhi oleh *Global Set/Reset*.



Gambar 2.9 Diagram Waktu Level Sensitive RAM



Gambar 2.10 16x1 atau 16x2 Level sensitie single port RAM.



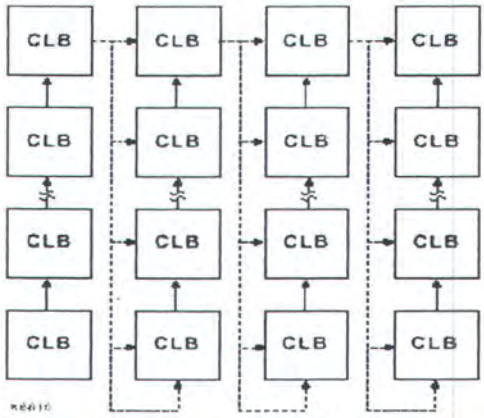
Gambar 2.11 32 x 1 Level Sensitive single port RAM

Fast Carry Logic

Sesungguhnya, *Function Generator* F dan G pada tiap CLB mempunyai logika artimatika untuk menghasilkan sinyal *carry* dan borrow dengan cepat. *Dedicated fast carry logic* ini meningkatkan efisiensi dan performansi dari *adder, subtractor, accumulator,* pembanding dan *counter.*

Seperti telah dijelaskan sebelumnya, 2 buah *Function Generator* dengan masing-masing 4 masukan dapat dikonfigurasi sebagai 2 bit *adder* dengan *build-in hidden carry* yang dapat dikembangkan menjadi *carry* yang lebih panjang. *Fast Carry logic* ini bisa mencapai kecepatan aritmetika dan perhitungan hingga 70 MHz.

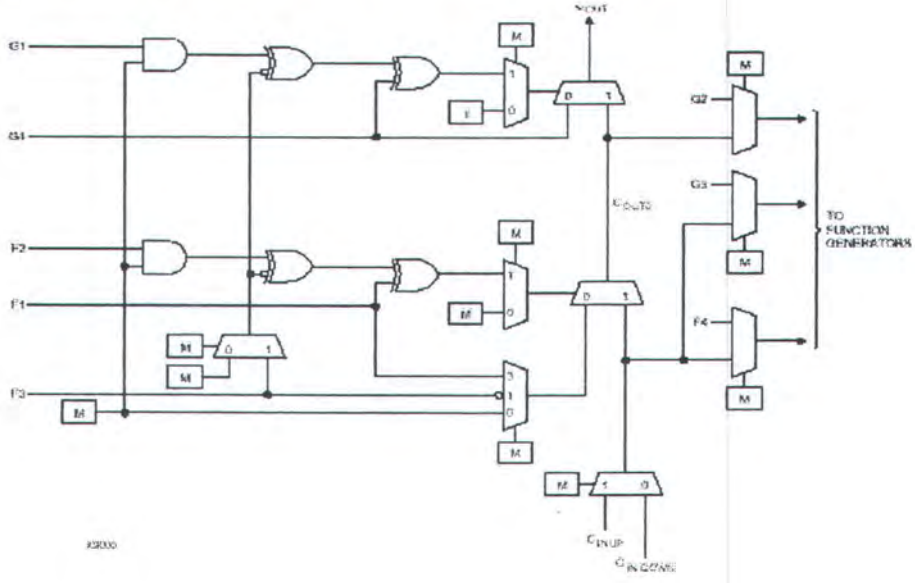
Carry untuk tipe XC4000E bisa *carry-up* dan *carry-down.* Pada bagian atas dan bawah kolom, bila tidak ada CLB yang lain di atas atau di bawahnya, *carry* akan menyebar ke kanan (lihat gambar 10). Untuk meningkatkan kecepatan pada *high capacity* XC4000X, maka *carry chain* diarahkan menaik saja.



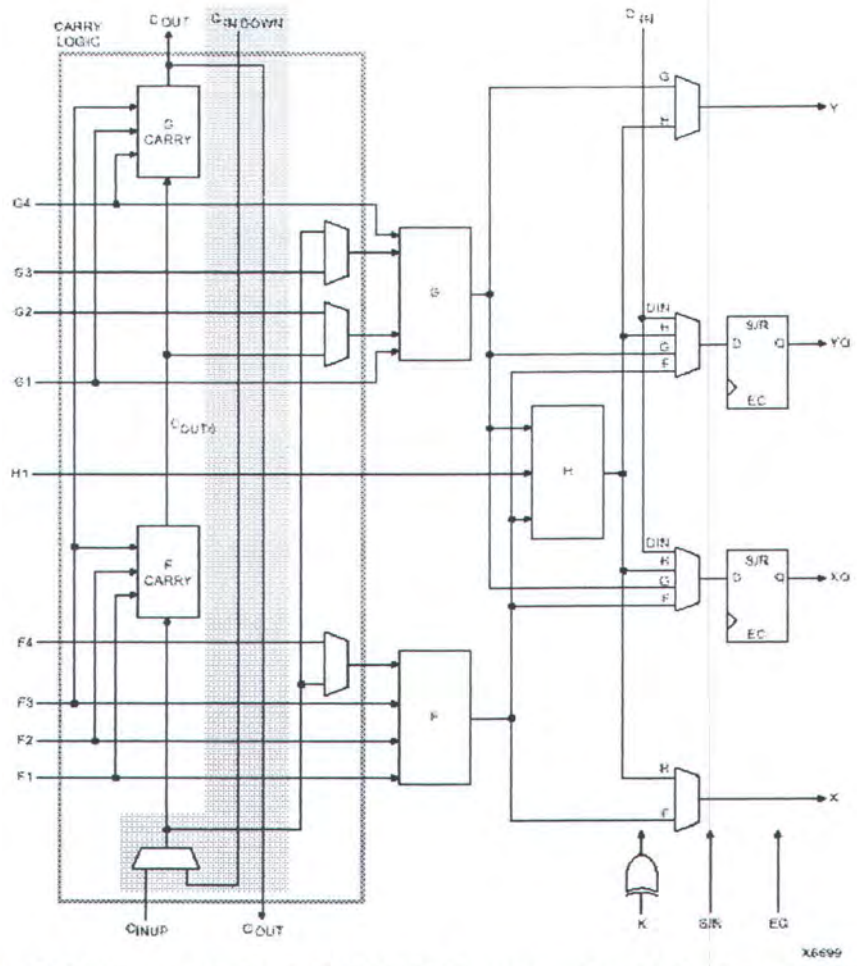
Gambar 2.12 Arah gerak jalur carry XC4000X yang mungkin.

(garis putus-putus menggunakan general interconect)

Gambar diatas menunjukkan CLB XC4000E *fast carry logic*. Bentuk ini mirip dengan tipe XC4000X, kecuali COUT hanya berada di atas dan sinyal CINDOWN tidak ada. Gambar 2.13 menunjukkan gambar *carry logic* untuk XC4000E. Mirip seperti XC4000E, pada XC4000X multiplekser yang ada pada *carry chain* telah dieliminasi untuk mereduksi *delay*. Sebagai tambahan pada XC4000X, multiplekser pada G4 memiliki sebuah *memory-programmable 0 input,* yang mengijinkan G4 untuk berhubungan ke COUT.



Gambar 2.13 Gambaran detil mengenai Fast carry logic XC4000E.



Gambar 2.14 Fast carry logic XC4000E (warna gelap tidak ada pada XC4000X)

2.1.3 INPUT OUTPUT BLOCK (IOB)

User-configurable I/O (Blok IO) menyediakan hubungan/*interface* antara *external package* dan *internal logic.* Setiap IOB dapat dikonfigurasi menjadi masukan, keluaran atau bidirectional. Gambar 2.15 menunjukkan Diagram Blok sederhana dari XC4000E IOB.

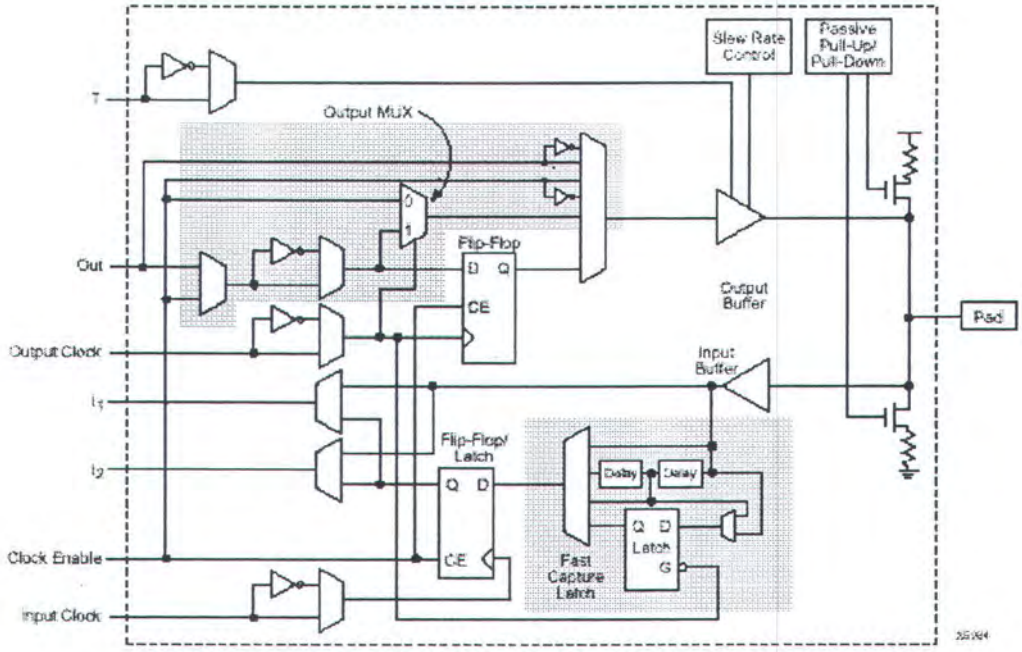
Sinyal Masukan IOB

Masukan pada XC4000XL adalah kompatibel dengan TTL dan 3,3 V CMOS. Keluaran XC4000XL telah di-*pulled* hingga 3,3 V sumber tegangan

positif. I/O pada IC XC4000XL benar-benar toleran dengan masukan 5V meskipun Vcc 3,3 V.

Tabel 2.3 Konfigurasi Masukan IOB XC4000.

Source	XC4000E/EX Series Inputs		XC4000XL Series Inputs
	5 V, TTL	5 V, CMOS	3.3 V CMOS
Any device, Vcc = 3.3 V, CMOS outputs	√	Unreli -able Data	√
XC4000 Series, Vcc = 5 V, TTL outputs	√		√
Any device, Vcc = 5 V, TTL outputs (Voh ≤ 3.7 V)	√		√
Any device, Vcc = 5 V, CMOS outputs	√	√	√



Gambar 2.15 Diagram Blok IOB XC4000X

Tabel 2.4 Fungsi Register Masukan

Mode	Clock	Clock Enable	D	Q
Power-Up or GSR	X	X	X	SR
Flip-Flop	_/‾	1*	D	D
	0	X	X	Q
Latch	1	1*	X	Q
	0	1*	D	D
Both	X	0	X	Q

Legend:

X	Don't care
_/‾	Rising edge
SR	Set or Reset value. Reset is default.
0*	Input is Low or unconnected (default value)
1*	Input is High or unconnected (default value)

Pada IC XC4000X, IOB nya mengandung *two-tap delay elemen,* dengan pilihan full *delay*, *delay* partial dan tanpa *delay*. IC XC4000X juga memiliki tambahan *latch* pada pin masukannya. *Fast Capture Latch* (FCL) didesain bersama *Global Early buffer.*

Sinyal Output IOB

Keluaran sinyal dapat di-*invert* dengan IOB itu sendiri, atau dapat langsung dihubungkan ke *pad* atau dihubungkan ke *Edge triggered flip-flop*. Fungsi ini dapat dilihat pada tabel 2.5. Aktif *high*, *tri-state* dapat diimplementasikan pada keluaran atau bidirectional I/O. Dengan mengatur *configuration control*, sinyal keluaran (OUT) dan keluaran 3-state (T) dapat di-*invert*.

Tabel 2.5 Fungsi Keluaran Flip-flop pada IOB.

Mode	Clock	Clock Enable	T	D	Q
Power-Up or GSR	X	X	0*	X	SR
Flip-Flop	X	0	0*	X	Q
	⎍	1*	0*	D	D
	X	X	1	X	Z
	0	X	0*	X	Q

Legend:

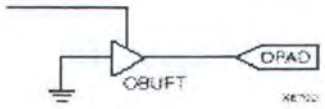
X	Don't care
⎍	Rising edge
SR	Set or Reset value. Reset is default.
0*	Input is Low or unconnected (default value)
1*	Input is High or unconnected (default value)
Z	3-state

Tabel 2.6 Konfigursi Keluaran IOB XC4000X

Destination	XC4000 Series Outputs		
	3.3 V, CMOS	5 V, TTL	5 V, CMOS
Any typical device, Vcc = 3.3 V, CMOS-threshold inputs	√	√	some[1]
Any device, Vcc = 5 V, TTL-threshold inputs	√	√	√
Any device, Vcc = 5 V, CMOS-threshold inputs	Unreliable Data		√

1. Only if destination device has 5 V tolerant inputs

Sebuah keluaran dapat dikonfigurasikan sebagai *open-drain (open-collector)* dengan menempatkan simbol OBUFT pada editor skematik atau editor HDL, dan menghubungkan pin 3 state (T) ke sinyal keluaran dan pin I ke *ground*.



Gambar 2.16 Keluaran Open Drain

Output Slew Rate

Slew rate pada tiap buffer keluaran dikurangi untuk meminimisasi daya bus transien ketika terjadi sinyal non-kritis. Untuk sinyal-sinyal kritis, pasang atribut *FAST* atau gunakan buffer keluaran atau *flip-flop*.

Untuk XC4000, beban kapasitas maksimum secara simultan untuk mode *switching* cepat pada arah yang sama adalah 200 pF pada tiap pasang pin *Power/Ground*. Untuk *slew rate* total keluaran dibatasi dua kali lebih besar untuk masing-masing tipe : 400 pF untuk XC4000E dan 600 pF untuk XC4000X.

Global Three-State

Sebuah jalur *Global 3-state* yang terpisah menyebabkan semua keluaran FPGA berada pada kondisi *high*-impedance, kecuali *boundary scan* di*enable* dan mengerjakan instruksi *EXTEST*. Net global ini (GTS) tidak menggunakan *routing resources*, namun menggunakan *dedicated distribution network.*

GTS dapat dikontrol dari user I/O pin mana saja sebagai masukan global 3-state. Untuk menggunakan net global ini, tempatkan sebuah *pad* masukan dan buffer masukan pada editor skematik atau editor HDL, untuk men-*drive* pin GTS pada simbol STARTUP. Pin ini dapat dilokasikan pada sebuah masukan dengan

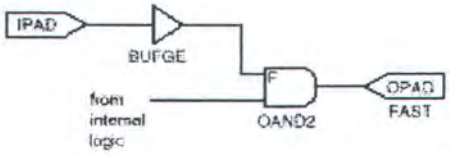
menggunakan atribut LOC yang terdapat pada user-programmable *pad*. Untuk mendapatkan sinyal *invert* dapat dilakukan dengan meletakkan simbol INV pada masukan buffer.

Multiplekser Output/Function Generator 2 Input (hanya XC4000X)

Seperti dapat dilihat pada gambar 2.15, jalur keluaran pada IOB XC4000X memiliki multiplekser tambahan yang tidak terdapat pada IOB XC4000E. Jika dikonfigurasikan sebagai multiplekser, menyebabkan sebuah pin keluaran dapat digunakan oleh dua buah sinyal keluaran, yang berarti menggandakan kemampuan keluaran IC.

Ketika *MUX* dikonfigurasikan sebagai *Function Generator* dengan 2 masukan, persamaan logika dapat diimplementasikan di dalam IOB itu sendiri. Dikombinasikan dengan *Global Early Buffer*, penyusunan ini mengurangi *delay* rangkaian. Sebagai contoh, sebuah dekoder luas dapat diimplementasikan dalam CLB, dan keluarannya diteruskan dengan memanfaatkan sinyal *Read* atau *Write* yang di-*drive* oleh buffer BUFGE seperti pada gambar 2.17. Jalur kritis untuk *delay* pin ke pin dari rangkaian kurang dari 6 nanodetik.

Untuk menggunakan multiplekser ini dapat dilakukan dengan menempatkan simbol yang dimulai dengan huruf "O". Sebagai contoh, sebuah gerbang AND 2 masukan dalam *Function Generator* IOB disebut OAND2.



Gambar 2.17 Jalur Cepat Pin ke Pin Pada XC400X

Gambar 2.18 Simbol AND dan MUX pada IOB XC4000X

Fasilitas IO lainnya

Pull-up dan Pull-down resistors.

Dengan adanya *programmable Pull-up* dan *pull-down resistor* sangat membantu untuk meminimasi konsumsi daya yang dibutuhkan dan mengurangi *noise* yang mungkin timbul.

Independent Clocks.

Clock dapat secara terpisah di-*invert* pada setiap *flip-flop* tanpa menggunakan CLB. Kecuali untuk XC4000X, semua *clock* masukan IOB adalah *independent.*

Early Clock Untuk IOB (hanya XC4000X)

Sumber dari sinyal *clock* ini sama dengan sumber *Global Low-Skew buffer*, tapi dipisahkan oleh buffer. Mereka memiliki beban dan *delay* yang lebih kecil. Sinyal *early clock* dapat men-*drive* ouput *clock* IOB atau masukan *clock* IOB, atau keduanya.

Global Set/Reset

Sama seperti register dalam CLB, sinyal *Global Set/Reset* (GSR) juga dapat digunakan untuk set atau reset register masukan dan keluaran, tergantung pada atribut INIT. Dua buah *flip-flop* dapat dikonfigurasikan sendiri-sendiri untuk men-*set* atau menghapus pada saat reset setelah konfigurasi.

2.1.4 THREE STATE BUFFER

Sepasang *three state buffer* ada pada tiap susunan CLB. *Three state buffer* ini dapat digunakan untuk men-*drive* sinyal menuju jalur horizontal terdekat diatas dan dibawah CLB. *Programmable pull-up resistor* yang dihubungkan pada jalur ini dapat digunakan untuk mengimplementasikan sebuah fungsi *wide wired-AND.*

Mode Three-State Buffer

Three state buffer dapat dikonfigurasi menjadi 3 mode, yaitu :

- *Standart 3-state buffer*
- *Wired-AND* dengan masukan pada I pin
- *Wired OR-AND*

Three State Buffer Standard

Tiga pin dari buffer tersebut digunakan. Tempatkan elemen *library* BUFT pada editor HDL atau skematik. Hubungkan masukan pada pin I dan keluaran pada pin O. Pin T adalah *active-high 3-state* (*active low enable*). Hubungkan pin T pada *ground* untuk mengimplementasikan *buffer standard.*

Wired AND dengan Input pada Pin I

Buffer dapat digunakan sebagai *Wired-AND*. Gunakan simbol WAND1, yang setara dengan *buffer open-drain*. WAND4, WAND8, dan WAND16 juga ada. Pin T dihubungkan secara *internal* pada pin I. Hubungkan masukan pada pin I dan keluaran pada pin O. Hubungkan keluaran dari semua WAND1 bersama-sama dan tambahkan simbol PULLUP.

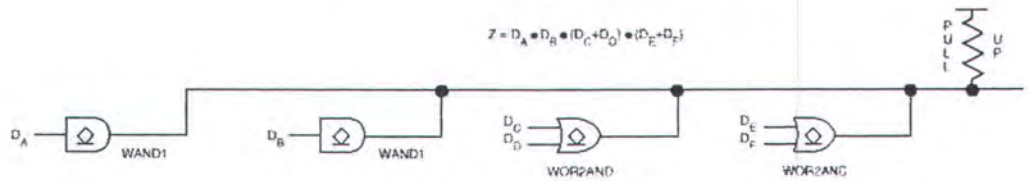
Wired OR-AND

Buffer dapat dikonfigurasikan sebagai sebuah *Wired OR-AND*. Level *High* pada salah satu masukan akan membuat keluaran menjadi nol. Gunakan simbol WORAND2, yang sama dengan gerbang OR 2 masukan *open-drain*. Hubungkan kedua masukan tersebut ke pin I0 dan I1 serta keluaran pada pin O. Hubungkan keluaran dari semua WORAND2 bersama-sama dan tambahkan simbol PULLUP.

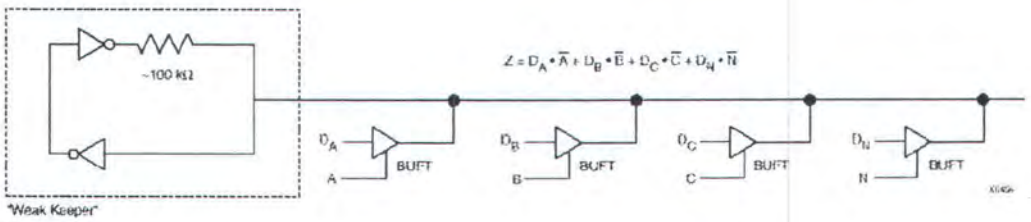
Gambar 2.19 menunjukkan cara mengimplementasikan fungsi *wired-AND* menggunakan *three-state buffer*. Ketika semua masukan buffer *high*, *resistor pull-up* menyebabkan keluaran *high*. Sedangkan gambar 2.20 menunjukkan cara penggunaan buffer 3-state untuk mengimplementasikan sebuah multiplekser. Seleksi dilakukan dengan memberi masukan pada sinyal 3-state pada buffer. Pin T pada *three-state buffer* merupakan *active-low*, seperti ditunjukkan pada tabel 2.7.

Tabel 2.7 Fungsi *Three State Buffer*

IN	T	OUT
X	1	Z
IN	0	IN



Gambar 2.19 Implementasi Fungsi *Wired-AND* dengan *Open Drain Buffer*

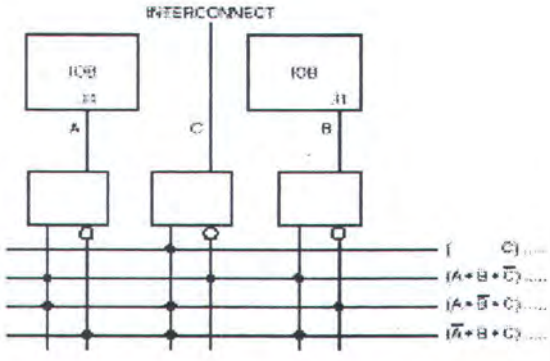


Gambar 2.20 Implementasi Multiplekser dengan *3-State Buffer*

2.1.5 WIDE EDGE DECODER

Rangkaian *Dedicated Decoder* meningkatkan kemampuan dari fungsi *decoding* luas. Ketika alamat atau *field* data lebih besar dibandingkan dengan masukan *Function Generator*, FPGA membutuhkan *multi-level decoding* dan lebih lambat dibandingkan PAL. CLB pada XC4000 memiliki 9 masukan. Dekoder yang memiliki masukan sampai dengan 9 masukan, dengan menggunakan CLB akan kompak dan cepat. Namun dibutuhkan juga dekoder untuk bit yang lebih banyak, sebagai contoh *decoding* alamat untuk sistem mikroprosesor.

XC4000 memiliki *programmable decoder* yang berada pada tiap ujung IC. Masukan dari tiap dekoder ini adalah sinyal I1 dari IOB pada pojok tersebut ditambah sebuah interkoneksi lokal untuk tiap CLB baris atau kolom. Tiap baris atau kolom CLB memiliki keluaran sampai dengan 3 variabel seperti pada gambar 2.21.



Gambar 2.21 Contoh *Edge Decoding* XC4000

Untuk menggunakan *wide edge decoder*, dapat dilakukan dengan meletakkan satu atau lebih simbol WAND (WAND1, WAND4, WAND8, WAND16). Pilih atribut DECODE pada masing-masing simbol WAND.

Hubungkan keluarannya bersama-sama dan sambung ke simbol PULLUP. Atribut lokasi seperti L (pojok kiri) atau TR (tengah kanan posok atas) juga harus diisi untuk memastikan penemaptan yang tepat dari dekoder masukan.

2.1.6 ON-CHIP OSCILLATOR

XC4000 memiliki sebuah *internal oscillator*. Osilator ini digunakan untuk proses *power-on time out*, konfigurasi *memory clearing* dan sumber CCLK pada mode master. Osilator berjalan nominal pada frekuensi 8Mhz yang bervariasi dengan proses, Vcc, dan suhu. Frekuensi keluaran jatuh antara 4 dan 10 MHz. Keluaran oscillator adalah pilihan setelah konfigurasi. Adapun frekuensi-frekuensi yang dapat digunakan pada *On Chip Oscillator* ini adalah 8MHz, 500KHz, 16 KHz, 490 Hz, dan 15 Hz.

Pengunaan komponen ini dapat dilakukan dengan menempatkan simbol OSC4 pada editor skematik atau editor HDL.

Gambar 2.22 Simbol *Oscillator* pada XC400

2.2 FUZZY LOGIC (LOGIKA SAMAR)

2.2.1 PENDAHULUAN

Pada sistem *boolean logic*, perubahan antara *membership* dengan *non-membership* mendadak dari logika '1' ke logika '0' atau sebaliknya. Pada sistem

menggunakan *fuzzy logic*, perubahan dilakukan secara bertahap dari logika '1' ke logika '0' atau sebaliknya hingga didapat perubahan output yang halus.

Teori *fuzzy set* dapat dijelaskan sebagai berikut :

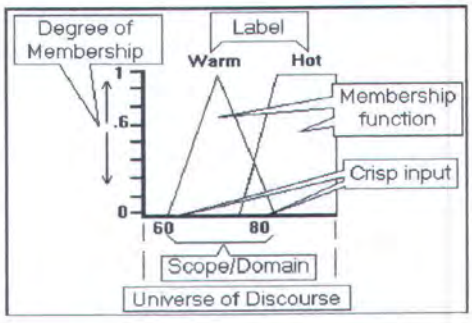
μ_s adalah *membership function* dari S.

$\mu_s(X) = 1$, artinya X adalah benar-benar anggota dari S

$\mu_s(X) = 0$, artinya X adalah bukan anggota dari S

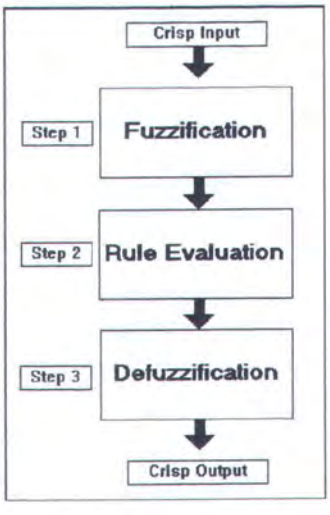
$0 < \mu_s(X) < 1$, artinya X adalah bagian dari S

Hal-hal yang berhubungan dengan *Fuzzy System* :



Gambar 2.23 Sistem Logika Fuzzy

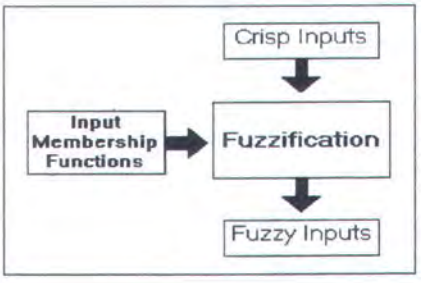
Solusi pada *Fuzzy System* adalah seperti pada diagram berikut ini :



Gambar 2.24 Solusi pada Sistem Fuzzy

2.1.1 FUZZIFIKASI DAN MEMBERSHIP FUNCTION

Fuzzification (Fuzzifikasi) adalah proses yang mengubah *crisp input* menjadi *fuzzy input*. Misalnya *crisp input* 78° untuk suhu udara diubah menjadi "*warm*". Syarat utama untuk mengubah *crisp input* menjadi *fuzzy input* adalah menentukan *membership function* untuk tiap input.

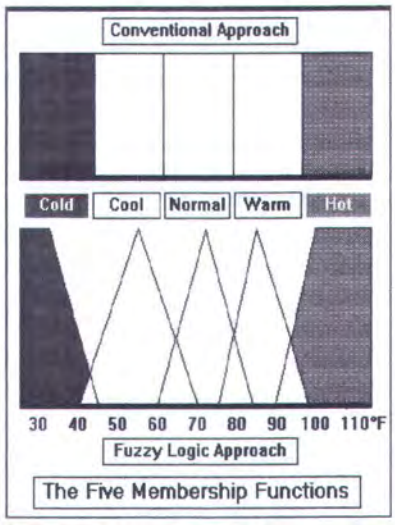


Gambar 2.25 Proses Fuzzifikasi

Langkah-langkah dalam fuzzifikasi adalah sebagai berikut :

LANGKAH PERTAMA, yaitu menentukan *fuzzy label* pada *universe of discourse* dari tiap *crisp input*, misalnya untuk suhu udara dibagi menjadi lima bagian (label) : "*cold*", "*cool*", "*normal*", "*warm*", "*hot*". Setiap *crisp input* dapat mempunyai lebih dari satu *fuzzy label*, misalnya suhu 80° dapat mempunyai *fuzzy label* "*warm*" dan "*hot*". Secara umum, semakin banyak *fuzzy label* yang ditentukan maka resolusi sistem tersebut akan semakin tinggi sehingga menghasilkan sistem kontrol yang lebih halus. Kerugian dari penentuan *fuzzy label* yang banyak adalah pemakaian waktu perhitungan yang lebih lama dan juga akan menyebabkan sistem tersebut tidak stabil. Pada umumnya, banyaknya *fuzzy label* adalah bilangan ganjil seperti 3, 5, 7, 9 tetapi bilangan-bilangan ini tidak mutlak. Selain itu, *fuzzy set* yang ditentukan sebaiknya *balance* (seimbang) dan simetris pada nol atau normalnya.

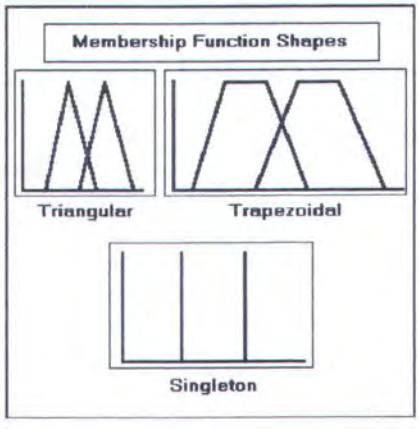
LANGKAH KEDUA, yaitu memberikan angka-angka pada *fuzzy label*, contohnya adalah sebagai berikut :



Gambar 2.26 Contoh Pemberian Angka pada Fuzzy Label

Gambar 2.26 di atas menunjukkan contoh pemberian angka pada lima *fuzzy label* yang menyatakan suhu udara dalam satuan Fahrenheit.

Bentuk dari *membership function* mempengaruhi *fuzzy process*, contohnya mempengaruhi waktu dan besar ruang memori yang dipakai dari mikrokontroler dalam memproses fuzzifikasi dan defuzzifikasi. Adapun bentuk-bentuk membership function yang umum digunakan adalah sebagai berikut :



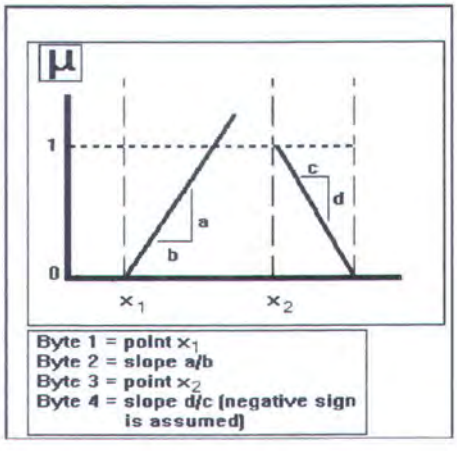
Gambar 2.27 Bentuk-Bentuk Umum Membership Function

Dari ketiga bentuk tersebut, biasanya bentuk *singleton* digunakan sebagai bentuk *output membership function*, karena bentuknya yang sederhana sehingga mudah diproses oleh mikrokontroler atau mikroprosesor dan membuat algoritma defuzzifikasi menjadi sederhana.

Ada beberapa metode untuk mengkonversi *memberhip function* pada mikro-kontroler atau mikroprosesor, yaitu :

Point Slope Representation

Metode ini memberikan waktu proses yang cepat dan ruang memori yang sedikit pada mikrokontroler atau mikroprosesor. Bentuk *membership function* dapat diwakili atau dikonversi dalam empat *bytes*, sebagai berikut :



Gambar 2.28 Point Slope Representation

Byte ke-1 adalah nilai dari X_1 pada *universe of discourse*.

Byte ke-2 merupakan nilai kemiringan atau gradien sisi pertama ($\frac{a}{b}$).

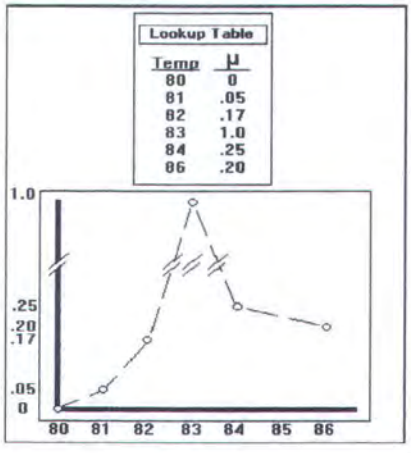
Byte ke-3 adalah nilai dari X_2 pada *universe of discourse*.

Byte ke-4 merupakan nilai kemiringan atau gradien sisi kedua ($\frac{d}{c}$).

Lookup Table Representation

Metode ini tepat digunakan untuk mengkonversi bentuk *membership function* yang berubah-ubah. Metode ini merupakan metode konversi yang cepat tetapi banyak menghabiskan memori. Banyaknya pemakaian memori tergantung dari bentuk *membership function* itu sendiri.

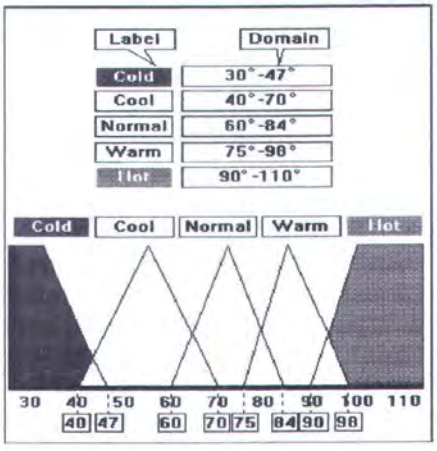
Sebagai tambahan, bentuk yang berubah-ubah ini akan memakan banyak waktu pada defuzzifikasi jika mikrokontroler atau mikroprosesor menggunakan metode *Centre of Gravity* (COG). Metode COG ini akan dibahas pada bagian defuzzifikasi.



Gambar 2.29 Lookup Table Representation

LANGKAH KETIGA, yaitu membuat *input membership function,* dibuat dengan menentukan *degree of membership* setiap input yang mungkin pada label. Nilai pada sumbu Y (μ) menunjuk kepada *degree* yang mana nilai *crisp input* ditetapkan/berlaku. Dari contoh tersebut, nilai 80° mempunyai 2 *fuzzy set*, yaitu "*normal*" dan "*warm*".

Mendefinisikan crisp input dalam bentuk fuzzy akan membuat sistem mempunyai respon yang berubah secara gradual (bertahap) dan halus jika input berubah secara bertahap.



Gambar 2.30 Contoh Input Membership Function

Contoh : pada gambar 2.30 di atas digunakan untuk penyiram tanaman otomatis. Diberlakukan *term* demikian :

Jika suhu "*warm*" maka lama penyiraman "*short*"

Jika suhu "*normal*" maka lama penyiraman "*medium*"

Jadi, jika suhu berubah dari 80° ke 78°, perubahan waktu penyiraman akan sedikit (tergantung dari degree of membership function).

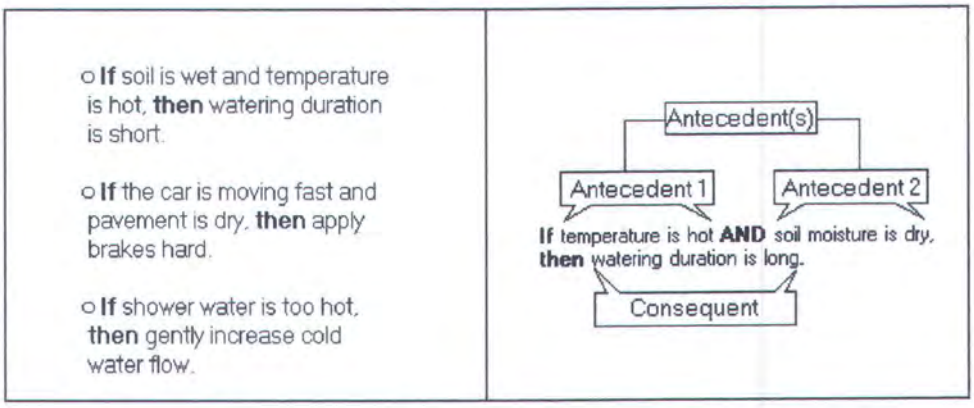
2.1.2 RULE EVALUATION

Prosesor Fuzzy menggunakan *rules* untuk menentukan aksi kontrol apa yang akan muncul sebagai respon dari input yang diberikan pada sistem tersebut. *Rule Evaluation* sering juga disebut *Fuzzy Inference*. Pada tahap ini, *rules* diberlakukan pada *fuzzy input* dan mengevaluasi tiap *rule* dengan input.

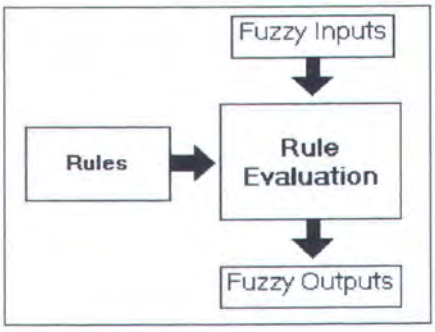
Fuzzy rules biasanya berupa "*if-then*" *statement* yang menyatakan aksi yang akan dilakukan sebagai respon dari *fuzzy input* yang bervariasi.

Jika *fuzzy rules* mempunyai lebih dari satu *antecedent* atau mempunyai lebih dari satu *consequent* maka antara *antecedent* atau *consequent* yang satu

dengan yang lain diberi penghubung yang disebut *fuzzy operator*. Pada contoh gambar 2.31, *rules* mempunyai *fuzzy operator* "***AND***".



Gambar 2.31 Contoh Fuzzy Rules dan Bagian-bagiannya

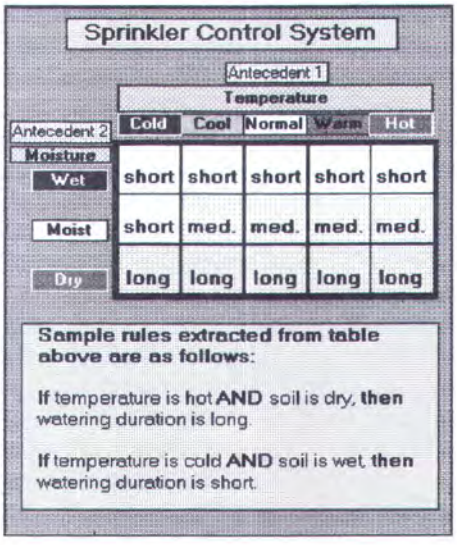


Gambar 2.32 Proses Rule Evaluation

Langkah-langkah *Rule Evaluation* :

1. Menentukan *rule*.
2. Mengevaluasi relevansi atau *degree of membership function* dari tiap *rule's antecendent*.
3. Menentukan *degree of truth* (*rule's strength*) untuk tiap *rule*.
4. Menentukan *fuzzy output* dengan membandingkan *rule strength* dari seluruh *rule* yang menyatakan *consequent label* yang sama.

LANGKAH PERTAMA, yaitu menentukan rule. Cara menentukan rule yang umum yaitu dengan menggunakan matriks *rule*. Contoh matriks *rule* dapat dilihat pada gambar 2.33.



Gambar 2.33 Contoh Matriks Rule

LANGKAH KEDUA, yaitu mengevaluasi relevansi atau *degree of membership function* dari tiap *rule's antecendent*. Untuk memperoleh relevansi dari tiap antecendent, dibuat garis vertikal melalui crisp input (x-value) yang diinginkan lalu dibuat garis horisontal untuk menentukan degree of membership function (y-value) mulai dari perpotongan garis vertikal tersebut dengan membership function ke sumbu Y.

LANGKAH KETIGA, yaitu menentukan *degree of truth* atau *rule's strength*. Dalam menentukan *rule's strength*, perlu diperhatikan pemakaian *fuzzy operator* pada *rule* sebab *fuzzy operator* sangat berpengaruh pada *rule's strength*.

Ada dua operator utama, yaitu "***AND***" (*intersection*) dan "***OR***" (*union*). Untuk operator "***AND***", nilai minimum dari *truth value of antecedent* dipilih untuk menentukan *rule's strength* secara keseluruhan. Untuk operator "***OR***", nilai

maksimum dari *truth value of antecedent* dipilih untuk menentukan *rule's strength* secara keseluruhan.

Rule : ***if*** suhu *is hot* (0.46) ***AND*** tanah *is dry* (0.25) ***then*** waktu siram *long*.

Rule tersebut mempunyai *rule's strength* = 0.25

Direkomendasikan, operator yang digunakan adalah operator "***AND***". Jika digunakan operator "***OR***", diusahakan dikonversi ke operator "***AND***".

Rule : ***if*** (X ***AND*** Y) ***OR*** (C ***AND*** B) ***then*** W

Diubah menjadi : ***if*** X ***AND*** Y ***then*** W

if C ***AND*** B ***then*** W

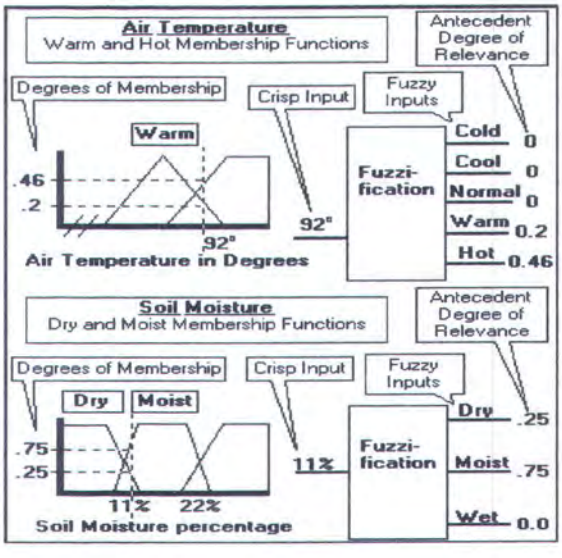
Rule : ***if*** (X ***OR*** Y) ***AND*** (C ***OR*** B) ***then*** W

Diubah menjadi : ***if*** X ***AND*** C ***then*** W

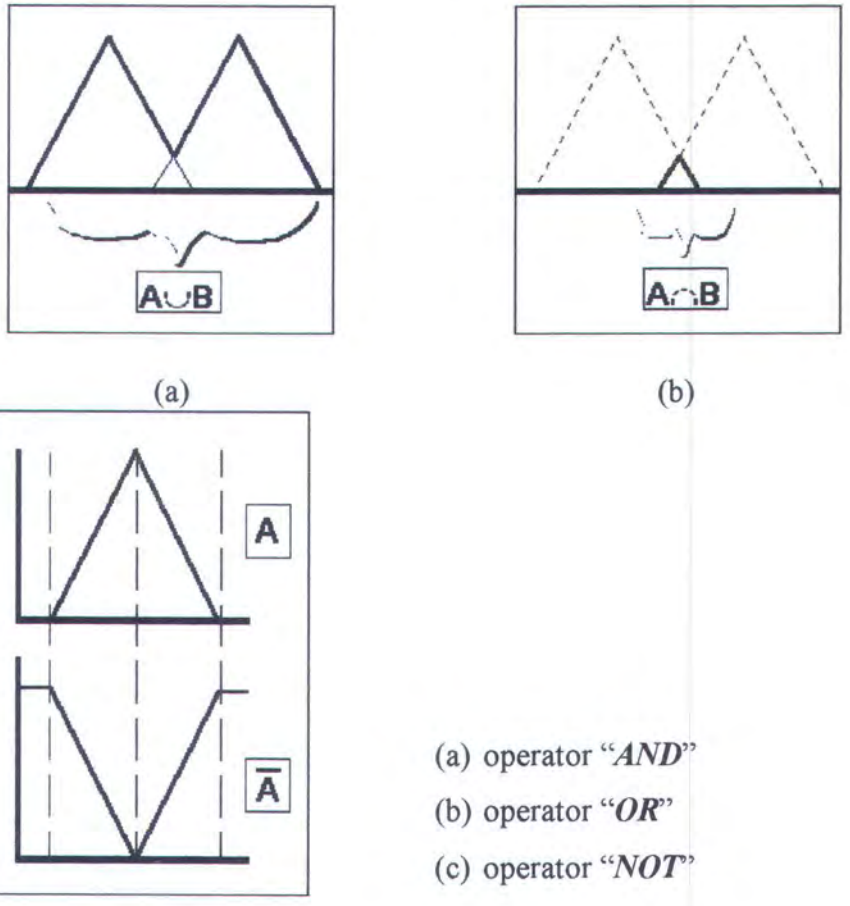
if X ***AND*** B ***then*** W

if Y ***AND*** C ***then*** W

if Y ***AND*** B ***then*** W



Gambar 2.34 Contoh Evaluasi Degree of Membership Function Tiap Rule



(a) (b)

(c)

(a) operator "*AND*"
(b) operator "*OR*"
(c) operator "*NOT*"

Gambar 2.35 Fuzzy Operator

LANGKAH KEEMPAT, yaitu menentukan *fuzzy output* dengan membandingkan *rule's strength* dari seluruh *rule* yang menyatakan *consequent label* yang sama.

Contoh : (i) waktu : "*long*" (0.25) dan "*medium*" (0.4)

(ii) waktu : "*long*" (0.6) dan "*medium*" (0.3)

Jika ada sekumpulan *rule* yang mempunyai dua aksi output misalnya "*long*" dan "*medium*", pada kasus ini, *fuzzy output* ditentukan oleh nilai maksimum *rule's*

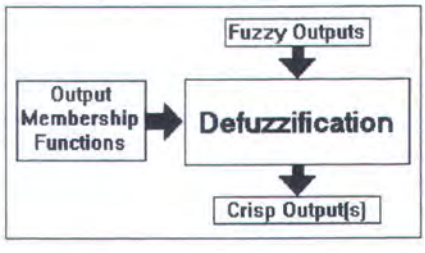
strength dari seluruh *rule* yang menyebabkan aksi yang sama. Jadi, pada contoh tersebut, akan dihasilkan *fuzzy output* : "*long*" (0.6) dan "*medium*" (0.4).

Catatan : *fuzzy output* hanya mempunyai satu aksi output yang sama, jadi, jika ada banyak aksi output yang sama seperti contoh, *rule* yang paling benar (*true*) atau yang mempunyai *rule's strength* paling besar akan dominan.

Metode *Rule Evaluation* yang digunakan ini disebut ***min-max inference***, karena metode ini mengambil nilai minimum dari *antecedent* untuk menentukan *rule's strength* dan mengambil nilai maksimum dari *rule's strength* tiap *consequent* untuk menentukan *fuzzy output*.

2.1.3 DEFUZZIFIKASI

Pada proses *defuzzification* (defuzzifikasi), seluruh output fuzzy yang signifikan (seperti waktu : "*long*", "*medium*", "*short*") akan dikombinasi menjadi sesuatu yang spesifik meliputi hasil untuk variabel output.



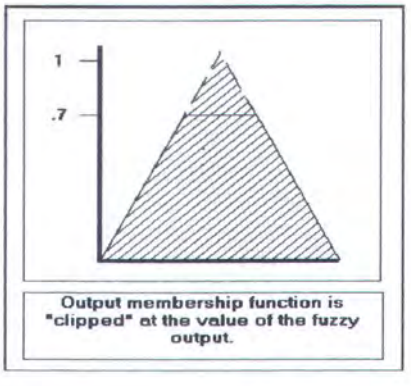
Gambar 2.36 Proses Defuzzifikasi

Langkah-langkah defuzzifikasi :

1. *Lambda-Cut*.
2. Menghitung COG atau *balanced point* atau *crisp output*.

LANGKAH PERTAMA, yaitu *Lambda-Cut*. Seluruh nilai *fuzzy output* secara efektif membatasi atau mengubah nilai *output membership function* masing-masing. Pada proses sebelumnya (*Rule Evaluation*), nilai *fuzzy output* diambil dari nilai terbesar dari *rule's strength* dari tiap *consequent*. Teknik defuzzifikasi yang paling sering digunakan adalah COG (*Center of Gravity*) atau metode *centroid*. Dalam metode ini, tiap *output membership function* yang nilainya di atass dari nilai *fuzzy output* yang bersesuaian akan dipotong.

Dengan *output membership function* seperti gambar 2.37 dan nilai *fuzzy output* sama dengan 0.7 maka *output membership function* akan dipotong dan hasil pemotongan itu ditunjukkan pada gambar dengan garis yang tidak terputus. Hasil *membership function* yang telah dipotong kemudian dikombinasi dan keseluruhan COG dihitung. Pemotongan *membership function* ini disebut ***Lambda-Cut***. *Lambda-Cut* membatasi kebenaran maksimum dari *membership function* atau daerah fuzzy (*fuzzy region*).

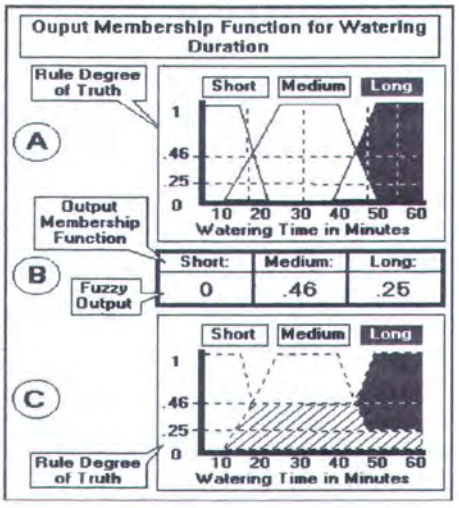


Gambar 2.37 Pemotongan pada Output Membership Function

Rumus : $\mu_A(X) = \min(\mu_A(X), \lambda - \text{cut})$

Gambar 2.38 (a) merupakan *output membership function* dari sistem penyiram otomatis. Dari proses *Rule Evaluation*, dihasilkan *fuzzy output* seperti pada tabel gambar 2.38 (b). Berdasarkan *fuzzy output* tersebut, didapat *output membership function* seperti pada gambar 2.38 (c).

Misal :



Gambar 2.38 Contoh Lambda-Cut pada Defuzzifikasi

LANGKAH KEDUA, yaitu mencari nilai seimbangnya (*balanced point*). Nilai seimbang inilah yang merupakan *crisp output* dari sistem penyiram tanaman otomatis tersebut. Cara menghitung nilai seimbang ini adalah menggunakan rumus COG berikut :

diskrit : $$COG = \frac{\sum_{i=1}^{n} y_i \mu_S(y_i)}{\sum_{i=1}^{n} \mu_S(y_i)}$$ analog : $$COG = \frac{\int_a^b \mu_x \cdot x dx}{\int_a^b \mu_x dx}$$

keterangan : n = jumlah *consequent* pada *fuzzy region* (S)

y_i = nilai crisp ke-i

$\mu_s(y_i)$ = *membership grade* dari *consequent* ke-i

Contoh perhitungan nilai seimbang (*crisp output*) :

Berdasarkan gambar 2.38 (c) diambil lima sampel nilai pada misalnya : 15, 25, 35, 45, dan 55, yang mempunyai nilai *rule degree of truth* masing-masing : 0.3, 0.46, 0.46, 0.46, dan 0.25

Maka nilai seimbangnya (*crisp output*) :

$$COG = \frac{(15 \times 0.3) + (25 \times 0.46) + (35 \times 0.46) + (45 \times 0.46) + (55 \times 0.25)}{0.3 + 0.46 + 0.46 + 0.46 + 0.25}$$

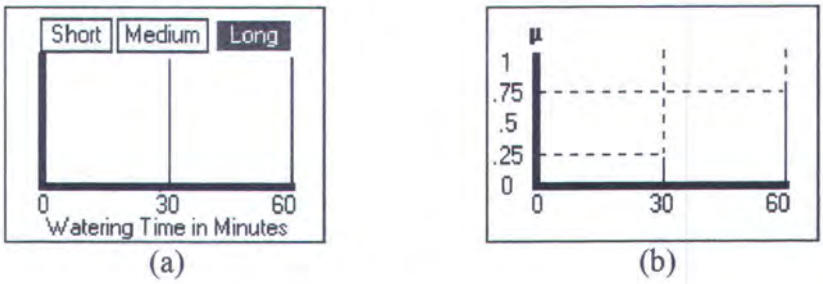
$$COG = 34.5$$

Jadi, *crisp output* = 34.5

Rumus COG di atas berlaku untuk *membership function* yang berbentuk *triangular* dan *trapezoidal*. Untuk *membership function* yang berbentuk *singleton*, rumusnya adalah :

$$COG = \frac{\sum_i (\text{fuzzy\_output}_i) \times (\text{position\_on\_x\_axis}_i)}{\sum_i (\text{fuzzy\_output}_i)}$$

Contoh perhitungan *crisp output* untuk *output membership function* berbentuk *singleton* :



(a) (b)

Gambar 2.39 Proses Defuzzifikasi untuk Bentuk Singleton

Gambar 2.39 (a) menyatakan *output membership function* bentuk *singleton* dari sistem penyiram tanaman otomatis. Gambar 2.39 (b) merupakan *output*

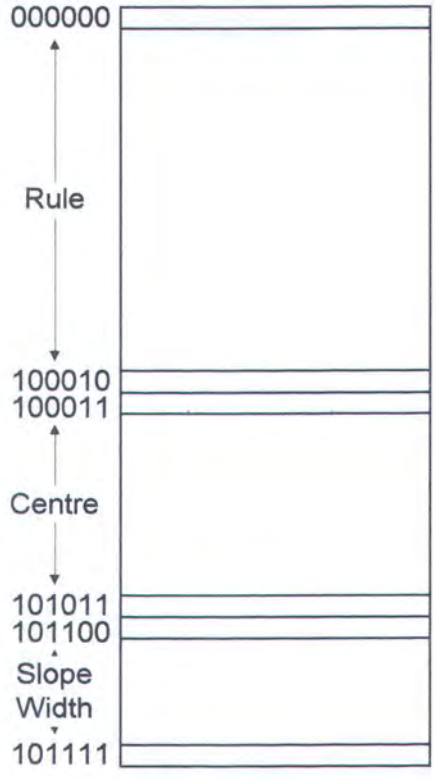
membership function yang telah dipotong berdasarkan *fuzzy output* dari proses *Rule Evaluation*. Maka,

$$COG = \frac{(0 \times 0) + (0.25 \times 30) + (0.75 \times 60)}{0 + 0.25 + 0.75}$$

$$COG = 52.5$$

2.1.4 ORGANISASI MEMORI

Prosesor fuzzy memiliki memori yang digunakan sebagai penyimpan parameter-parametr dan rule-rule yang dibutuhkan oleh suatu proses. Jumlah memori ini adalah 48 x 10 bit data, dimana pada 13 alamat terakhir berfungsi untuk menyimpan nilai-nilai Centre, Width, dan Slope dari proses fuzzy yang dilakukan. Diagram pengorganisasian memori dapat dilihat pada gambar 2.44.



Gambar 2.40 Organisasi Memori

2.3 SENSOR SUHU UDARA LM34

2.3.1 PENDAHULUAN

Pada umumnya sensor suhu elektris susah digunakan. Sebagai contoh, termokopel memiliki amplitudo tegangan keluaran kecil dan memerlukan kompensasi batas dingin. Contoh lain adalah termistor yang tidak linier. Sebagai tambahan, keluaran dari sensor-sensor ini tidak berbanding secara linier terhadap skala suhu apa saja. IC-IC sensor suhu yang dahulu seperti LM3911, LM134 dan LM135, menimbulkan beberapa masalah dimana tegangan keluarannya sebanding dengan skala Kelvin, dan bukannya Celcius atau Fahrenheit yang lebih populer. Untungnya, pada tahun 1983 ditemukan dua buah IC baru, yaitu LM34 (*Precision Fahrenheit Temperature Sensor IC*) dan LM35 (*Precision Celcius Temeprature Sensor*).

LM34 memiliki perubahan tegangan keluaran sebesar 10 mv setiap perubahan 1°F, dan memiliki nilai akurasi sebesar 0,4°F pada suhu kamar (77°F). Karena LM34 memiliki nilai impedansi keluaran yang rendah dan karakteristik keluaran yang linier, maka LM34 dapat dihubungkan ke rangkaian pengatur atau pembacaan data dengan mudah.

Kebanyakan sensor-sensor suhu udara memiliki tegangan keluaran hanya 1μA/°K. Hal ini menyebabkan terjadinya kesalahan 1°K jika terjadi arus bocor sebesar 1 μA. Sebaliknya, LM34 dapat dioperasikan untuk menghasilkan perubahan arus keluaran sebesar 20μA/°F. Jadi dengan adanya arus bocor yang sama sebesar 1μA akan menyebabkan kesalahan sebesar 0,05°F untuk IC LM34.

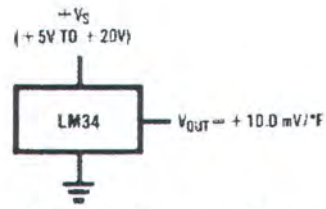
Harga yang murah dan presisi yang tinggi dapat dilakukan dengan melakukan *trimming* dan cara kalibrasi yang tepat. IC ini dapat dioperasikan dengan menggunakan satu atau dua sumber tegangan. Dengan arus *drain* kurang dari 70μA, LM34 tidak cepat panas. Bentuk atau *package* dari IC LM34 ini dapat berupa TO-46, SO-8 maupun TO-92 seperti gambar 2.41.

Gambar 2.41 Diagram Koneksi LM34

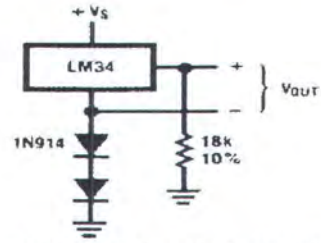
2.3.2 PRINSIP KERJA

LM34 dapat dioperasikan dengan mudah seperti IC–IC sensor suhu udara lainnya. LM34 dapat dilekatkan pada sebuah permukaan dan suhu yang dihasilkan kira-kira 0,02°F dibandingkan suhu permukaan tersebut. Hal ini dengan asumsi suhu udara rata-rata hampir sama dengan suhu permukaan tersebut. Apabila suhu udara lebih besar sekali dibandingkan dengan suhu permukaan, maka suhu yang dihasilkan oleh LM34 merupakan nilai tengah antara suhu permukaan dengan suhu udara.

Gambar 2.42 menggambarkan cara penggunaan IC LM34 ini secara sederhana yang beroperasi pada suhu 5°F sampai dengan 300°F. Sensor ini juga dapat dioperasikan untuk menghasilkan keluaran suhu antara -50°F sampai dengan 300°F dengan menggunakan satu sumber tegangan seperti pada gambar 2.43, yaitu dengan menambahkan sebuah resistor dari pin keluaran ke *ground*, dan menghubungkan dua buah dioda secara seri antara pin *ground*.

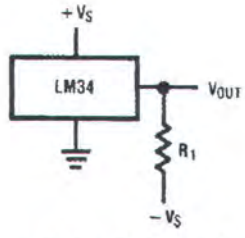


Gambar 2.42 Sensor Suhu Udara Fahrenheit Dasar (+5°F sampai +300°F)



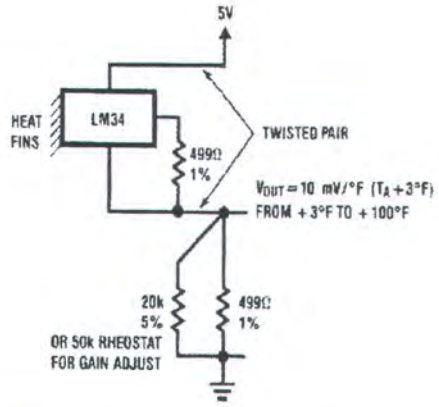
Gambar 2.43 Sensor Suhu Udara Fahrenheit (-50°F sampai dengan +300°F)

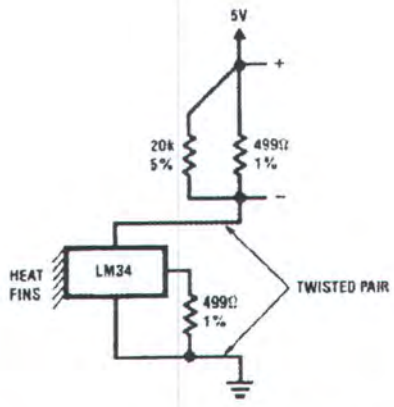
Sedangkan dengan menggunakan dua buah sumber tegangan, sensor LM34 ini dapat digunakan untuk menghasilkan range suhu udara penuh dengan menambahkan resistor pull-down dari pin keluaran ke sumber tegangan negatif seperti digambarkan pada gambar 2.44. Nilai resistor ini harus $|-Vs|/50\mu A$.



Gambar 2.44 Sensor Suhu Udara dengan Range Penuh

Untuk aplikasi dimana sensor berada pada jarak yang jauh dengan rangkaian pembaca, adalah amat mahal apabila kita menggunakan tiga buah kabel. Untuk mengatasi masalah ini dapat digunakan dua buah rangkaian sebagai berikut



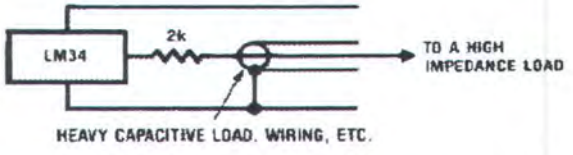


Gambar 2.45 Hubungan 2 kabel LM34 a) Keluaran Referred to *Ground*

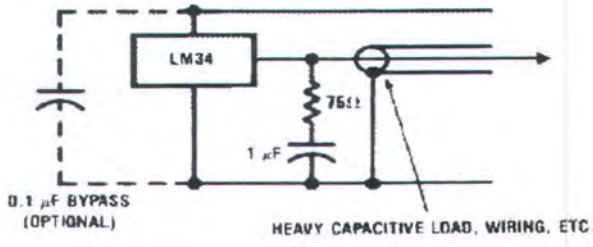
b) *Ground*ed Sensor)

2.3.3 BEBAN KAPASITANSI

Seperti layaknya rangkaian dengan daya kecil lainnya, LM34 memiliki kemampuan terbatas untuk dihubungkan dengan beban kapasitansi. Kemampuan LM34 ini hanya mampu menangani beban sebesar 50pF. Untuk mengantisipasi beban yang lebih berat dapat digunakan resistor (gambar 2.46). Atau untuk meningkatkan toleransi kapasitansi dapat digunakan rangkaian *R-C damper* seri dari pin keluaran ke *ground* (gambar 2.47).



Gambar 2.46 Rangkaian LM34 untuk mengatasi beban kapasitansi besar



Gambar 2.47 RangkaianLM34 dengan *RC Damper*

2.4 SENSOR KELEMBABAN TANAH

2.4.1 PENDAHULUAN

Tanah berfungsi sebagai tempat untuk menyimpan air, baik dari irigasi maupun dari hujan, sehingga dapat menjadi sumber air yang baik bagi tanaman. Tujuan dari penggunaan sensor kelembaban tanah adalah untuk mengukur kandungan air di dalam tanah, sehingga memungkinkan kita untuk melakukan irigasi pada sebidang tanah pada waktu yang tepat, sehingga tanaman tidak kekurangan air dan tumbuh dengan baik.

2.4.2 METODE PENGUKURAN KELEMBABAN TANAH

Kelembaban tanah dapat diukur dengan dua buah metode, yaitu dengan menentukan jumlah kandungan di dalam tanah dan dengan mengukur tegangan pada air tanah (*soil water potential).* Jumlah kandungan air adalah jumlah air per volume tanah dalam kedaan kering atau berat dari tanah dalam keadaan kering, sedangkan *soil water potential* adalah tegangan yang ditimbulkan oleh adanya air di dalam tanah.

Masing-masing metode ini tentunya memiliki kelebihan dan kekurangan masing-masing. Adapun metode untuk menentukan kandungan tanah ada 7 macam, yaitu :

1. Metode Perasaan
2. *Gravimetric*
3. *Neutron attenuation probe*
4. *Time domain reflectometry* (TDR)

5. Sensor kapasitansi
6. Sensor disipasi panas
7. *Velocity Differentiation Domain* (VDD)

Sedangkan metode untuk mengukur *soil water potential* adalah sebagai berikut :

1. *Tensiometer*
2. *Gypsum Blocks*
3. *Granular Matrix Sensor*
4. *Psychrometer*
5. *Pressure plate.*

Tidak ada sensor yang sempuran yang mampu menghasilkan pengukuran akurat untuk semua jenis tanah namun memiliki harga murah. Sensor kelembaban tanah yang efektif adalah sensor yang mampu memberikan data akurat pada range kelembaban tanah yang kritis pada tanah yang diteliti.

Psychrometer tidak dapat menghasilkan pembacaan yang baik pada tanah yang basah, sedangkan *Time domain reflectometry* adalah sensor kelembaban tanah yang akurat namun berharga mahal.

Granular Matrix Sensor (*Watermark Soil Moisture Sensor, Irrometer Co*) mengurangi masalah-masalah yang ada pada *gypsum block* (pembacaan yang kurang akurat akibat kurang kontak antara tanah dengan sensor) dengan menggunakan sebuah *granular matrix* yang terbungkus sebuah logam atau layar plastik.

Granular Matriks Sensor yang kami gunakan adalah model SS200 seperti telihat pada gambar 2.48. Sensor ini terdiri dari dua buah elektroda konsentris yang dikubur dalam matriks material referensi, Matriks material ini dilindungi oleh membran sintetis, Membran ini berfungsi untuk melindungi material matriks dari hal-hal buruk. Sensor model SS200 ini dapat ditinggalkan di dalam tanah sepanjang tahun tanpa harus menggantinya pada musim dingin. Sebuah tablet *buffer gypsum* melawan kadar garam yang ada di dalam air irigasi sehingga menghindari terjadinya perkaratan.

Gambar 2.48 Gambar Sensor Kelembaban Tanah SS200

2.4.2 PRINSIP KERJA SENSOR SS200

Sensor SS200 ini pada dasarnya merupakan suatu peralatan resistif. Sensor ini dapat dioperasikan dengan menggunakan sumber tegangan DC (*Direct Current*) maupun AC (*Alternating Current*). Sumber tegangan DC hanya dapat dipakai selama 50 ms, dan setelah pembacaan, keluaran sensor harus digabungkan bersama untuk menghilangkan tegangan sisa yang masih ada. Data yang diperoleh dengan sumber tegangan DC hanya akan optimal pada 40ms pertama. Selain itu, pengoperasian dengan sumber tegangan DC juga menyebabkan terjadinya tegangan *offset* sebesar 0.550 volt pada keluaran sensor. Tegangan *offset* ini tentunya harus dikurangi sebelum masuk ke ADC.

Untuk mendapatkan hasil yang akurat dari sensor SS200 ini, kita harus mengoptimalkan nilai dari Ra, yaitu tahanan bias sensor SS200. Pemilihan nilai Ra bergantung pada 2 buah hal, yaitu:

a. Sumber tegangan
b. Batas Range tegangan analog.
c. Pembacaan maksimum dari sensor kelembaban tanah.

Berikut ini merupakan contoh dari kriteria yang akan dipakai sebagai pembacaan sensor kelembaban tanah :

a. Sumber tegangan yang digunakan adalah 12 Volt DC.
b. Batas Range tegangan analog antara 0 sampai dengan 5 Volt.
c. Pembacaan masimum skala CB adalah 125 cb (135Ω/cb).

Untuk mendapatkan nilai yang cocok bagi tahanan bias Ra, maka dilakukan perhitungan sebagai berikut :

a. Menghitung pembacaan resistansi maksimum untuk sensor

Max CB x ohms/cb = 125 x 135

= 16.875 ohms

b. Menghitung arus total rangkaian

$$It = \frac{5}{16.875} = 225\mu A$$

c. Menghitung nilai Ra

$$Ra = \frac{Vs - Vspan}{It} \qquad Ra = \frac{12-5}{296\mu A} = 23{,}6K\Omega$$

Setelah kita mendapatkan harga Ra, kita harus mengkonversikannya ke harga tahanan dari sensor kelembaban tanah tersebut sehingga kita dapat memperoleh harga CB (centibar) berdasarkan lembaran kalibrasi.

Untuk mengetahui nilai tahanan dari sensor tersebut, maka dilakukan perhitungan-perhitungan sebagai berikut:

$$Va = Vs - Vdrop$$

$$It = \frac{Va}{Ra}$$

$$Rx = \frac{Vdrop}{It}$$

$$Rx = \frac{(Ra \times Vdrop)}{(Vs - Vdrop)}$$

dimana :

Vs = Sumber tegangan

It = Arus total

Rx = Nilai tahanan sensor

Ra = Tahanan bias

Dengan persamaan diatas maka kita akan mendapatkan harga tahanan sensor kelembaban tanah. Kemudian dari nilai tersebut harus kita konversikan ke nilai CB.

2.4.3 INSTANLASI SENSOR SS200

Penempatan sensor kelembaban tanah merupakan hal yang penting untuk mendapatkan hasil pengukuran yang akurat. Sensor tidak boleh ditempatkan pada bagian-bagian rendah dimana terdapat genangan air, tempat-tempat tinggi, atau

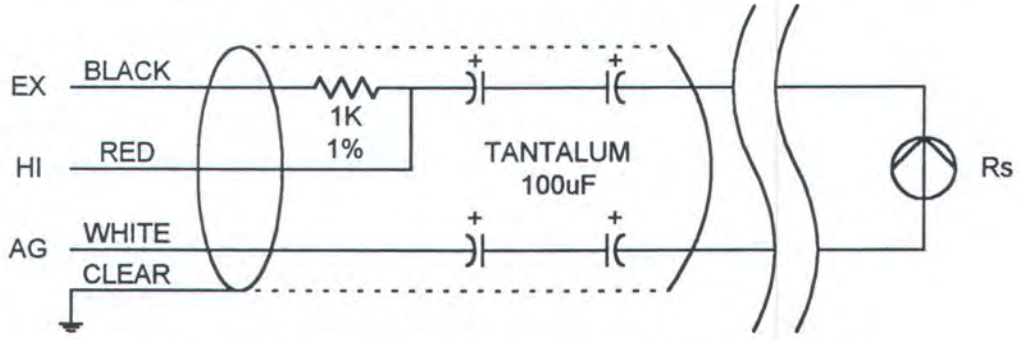
kubangan-kubangan, kecuali tempat tersebut merupakan tempat yang memang akan diukur kelembaban tanahnya. Prosedur untuk mengoperasikan sensor:

1. Rendam sensor sepanjang malam pada air irigasi. Basahi selama 30 menit pada pagi hari dan kemudian keringkan sampai dengan sore hari. Basahkan kembali selama 30 menit, keringkan sampai sore hari lagi, basahi lagi selama 30 menit pada pagi hari berikutnya dan keringkan pada sore harinya. Rendam pada besok malamnya dan sensor siap ditanamkan dalam tanah. Prosedur ini harus dilakukan untuk pemasangan sensor pertama kali.
2. Buat lubang sensor dengan kedalaman yang dibutuhkan selebar 7/8 inchi. Untuk tanah liat atau padat buat lubang yang lebih besar (kira-kira 1 – 1,25 inchi) untuk menhindari terjadinya kerusakan membran sensor pada saat memasukkan sensor ke dalam tanah.
3. Campur sebongkah tanah dengan air sehingga berada dalam konsentrasi krim dan tempatkan satu atau dua sendok meja makan krim ke dalam lubang.
4. Masukkan sensor ke dalam lubang yang telah diisi campuran air dan tanah tadi. Untuk memastikan kontak antara tanah dengan sensor, tekan sensor sehingga campuran tadi menutupi sensor. Kemudian tutup kembali lubang tersebut secara hati-hati.

2.4.4 HUBUNGAN KABEL SENSOR

Gambar hubungan kabel dari sensor kelembaban tanah SS200 ini dapat dilihat pada gambar 2. Kabel merah (Sinyal positif) dihubungkan pada kanal masukan analog dari ADC, sedangkan kabel hitam dihubungkan pada bagian

eksitasi (berupa sinyal kotak), dan kabel putih dihubungkan pada *Ground* rangkaian. Di dalam kabel tampak terdapat rangkaian kapasitor yang berfungsi untuk menahan aksi galvanik yang timbul akibat perbedaan tegangan antara *ground* rangkaian ADC dengan *ground* sensor. Arus bocor yang mengalir ke sensor akan mengakibatkan kerusakan.



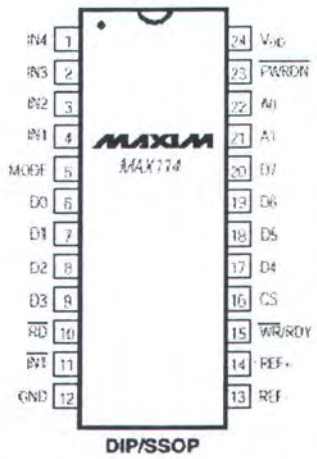
Gambar 2.49 Diagram Hubungan Kabel Sensor SS200

2.5 ANALOG TO DIGITAL CONVERTER

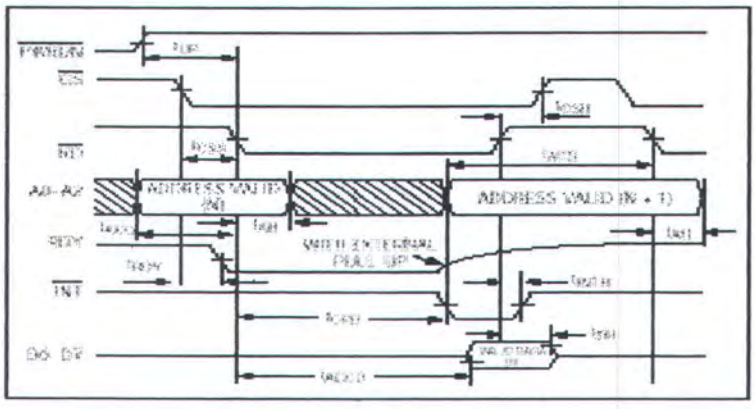
IC MAX114 keluaran Maxim Semiconductor merupakan IC ADC (*Analog to Digital Converter*) 8 bit yang kompatibel (dapat diinterfacekan langsung) dengan mikroprosesor, dan memiliki kanal masukan sebanyak empat buah. IC ini bekerja dengan menggunakan sebuah sumber tegangan 5 Volt dan memiliki waktu konversi sebesar 660 ns (1 Msps).

Pada prinsipnya, proses ADC menggunakan IC MAX114 ini terbagi menjadi dua bagian, yaitu proses pembacaan dan pengkonversian tegangan masukan analog, serta proses pembacaan data. Proses pengkonversian data dimulai dengan memberikan sinyal 'low' pada pin RD. Pada saat sinyal RD 'low', mikroprosesor akan menunggu aktifnya sinyal INT yang ditandai denga pulsa

aktif low pada pin INT, yang melambangkan bahwa data pada keluaran ADC valid. Sinyal INT ini akan kembali '*high*' pada saat naiknya (*rising edge*) sinyal RD. Diagram waktu dari proses konversi dan pembacaan ADC ini dapat dilihat pada gambar 2.51.



Gambar 2.50 Konfigurasi pin MAX114



Gambar 2.51 Diagram Waktu MAX114

BAB III
PERENCANAAN DAN PEMBUATAN ALAT

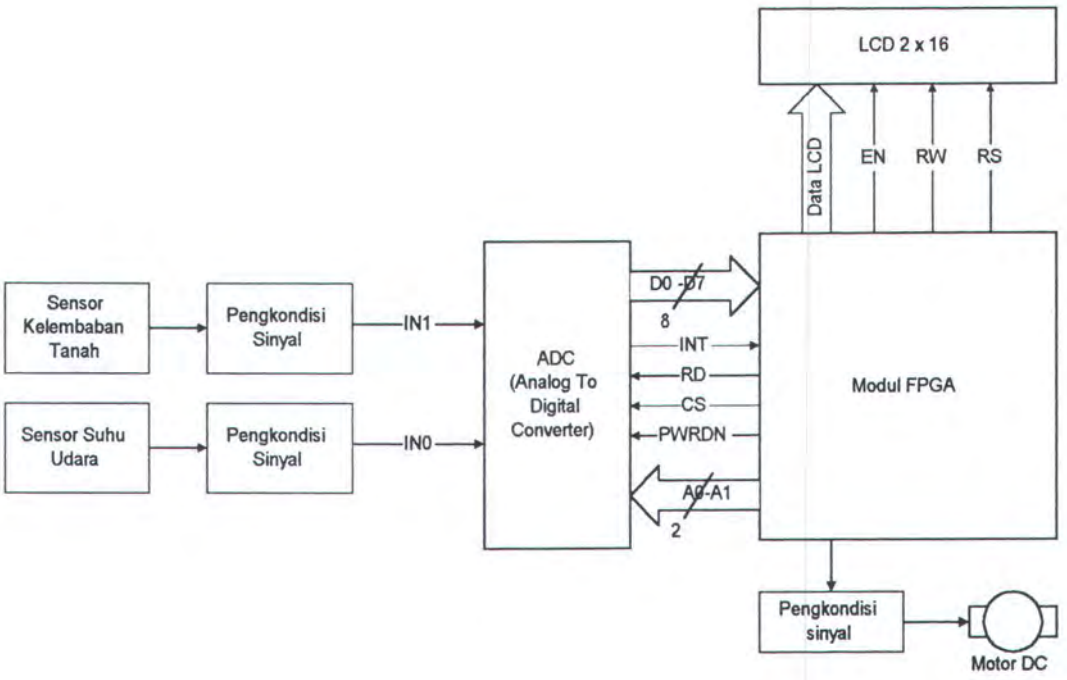
BAB III

PERENCANAAN DAN PEMBUATAN ALAT

Dalam bab ini akan dibahas mengenai perencanaan perangkat keras luar (*hardware eksternal*) dan perencanaan modul-modul FPGA dari Tugas Akhir APLIKASI FPGA SEBAGAI PENDETEKSI PARAMETER TANAH DENGAN METODE *FUZZY LOGIC*. Perencanaan perangkat keras meliput semua perangkat keras yang digunakan di luar IC FPGA, sedangkan perencanaan FPGA meliputi semua modul yang dibuat dengan menggunakan pemrograman VHDL pada IC FPGA XC4010XL.

3.1 PERENCANAAN PERANGKAT KERAS

Perangkat keras terdiri dari modul *display* berupa LCD (*Liquid Crystal Display*) 2 baris x 16 kolom keluaran *Seiko Instrument*, modul ADC (*Analog to Digital Converter*) yang berupa IC MAX114 keluaran *Maxim Semiconductor*, modul suhu udara yang terdiri dari sensor suhu IC LM34 keluaran *National Semiconductor* dan rangkaian pengkondisian sinyal, modul kelembaban tanah yang terdiri dari sensor kelembaban tanah milik *Irrometer*, rangkaian pengkondisian sinyal, rangkaian *pulse generator* dan *sample and hold*, serta modul catu daya dan modul tegangan referensi. Adapun diagram blok dari perencanaan perangkat keras ini dapat dilihat pada gambar 3.1.



Gambar 3.1 Diagram Blok Perencanaan Perangkat Keras

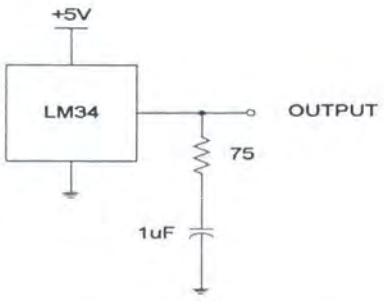
3.2.1 MODUL SUHU UDARA

Modul suhu udara terdiri dari sensor suhu udara LM34 milik *National Semiconductor* dan rangkaian pengkondisi sinyal yang berupa IC LM324.

3.2.1.1 Sensor Suhu Udara

Untuk mendapatkan data mengenai suhu udara, penulis menggunakan sensor suhu udara Fahrenheit LM34. Sensor ini memiliki kemampuan untuk menghasilkan tegangan keluaran sebesar 10mv setiap perubahan 1°F. Sebagai contoh, pada saat suhu udara berada pada 70°F maka IC LM34 ini memiliki tegangan keluaran sebesar 700 mv.

Gambar rangkaian sensor suhu yang penulis gunakan untuk menghasilkan range suhu udara antara 30°F sampai dengan 110°F pada tugas akhir ini dapat dilihat pada gambar 3.2.



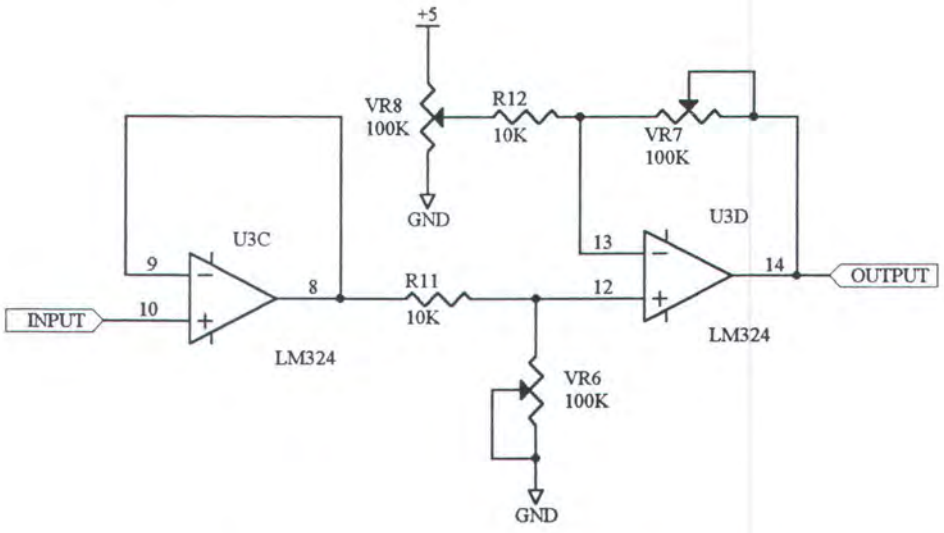
Gambar 3.2 Gambar Rangkaian Suhu Udara

Fungsi dari rangkaian *RC damper* seri adalah untuk mengatasi beban kapasitansi yang besar karena sensor suhu LM34 ini akan diletakkan pada letak yang berjauhan dengan panel kontrol (dalam hal ini adalah *XESS Board XS40-010XL*).

3.2.1.2 Rangkaian Pengkondisi Sinyal

Karena suhu udara yang ingin kita amati adalah range antara 30°F sampai dengan 110°F, maka kita membutuhkan rangkaian pengkondisi sinyal untuk mengubah range tegangan yang dihasilkan pada range suhu tersebut menjadi range 0–5 Volt. Rangkaian pengkondisi sinyal ini terdiri dari rangkaian *buffer* atau *voltage follower* yang berfungsi untuk menyangga atau sebagai *matching impedance* antara sensor LM34 dengan rangkaian sesudah *buffer* , serta rangkaian *differential amplifier* yang berfungsi mengeliminasi tegangan 0,3 volt (suhu pada ssat 30°F) dan menguatkan range tegangan keluaran maksimal (pada saat suhu udara sama dengan 110°F) menjadi 5 Volt.

Rangkaian pengkondisi sinyal ini secara keseluruhan menggunakan IC LM324, yaitu *Operational Amplifier* berempat yang hanya membutuhkan satu buah catu daya saja.



Gambar 3.3 Gambar Rangkaian Pengkondisi Sinyal Suhu Udara

Pada gambar tersebut tampak bahwa *op-amp* paling depan berfungsi sebagai *buffer*, sedangkan *op-amp* kedua berfungsi sebagai *differential amplifier* yang memiliki rumus penguatan (*gain*) sebagai berikut :

$$A = \frac{Vo}{Vin(+) - Vin(-)}$$

$$VR1 = VR3 = A \times R1 = A \times R2$$

$$A = \frac{5}{1100 - 300} = 6{,}25$$

$$VR1 = VR3 = 6{,}25 \times 10K\Omega$$

$$VR1 = VR3 = 62{,}5K\Omega$$

Pada rangkaian tersebut dipilih R1 dan R2 sama dengan 10 KΩ, sedangkan VR2 diatur supaya terdapat tegangan jepit pada masukan negatif *op-amp* sebesar 300 milivolt, yang bertujuan untuk mengeliminasi tegangan 300

milivolt dari keluaran sensor suhu pada keadaan 30°F sehingga diperoleh tegangan keluaran 0 Volt. VR1 dan VR3 merupakan resistor variabel *multiturn* yang diatur pada harga tahanan yang sama untuk menentukan *gain* dari rangkaian pengkondisi sinyal.

Dengan rangkaian pengkondisi sinyal tersebut, maka diperoleh tegangan keluaran sebagai berikut :

a. Pada keadaan minimum (suhu 30°F)

Vout = 6,25 x (300mv – 300mv) = 0 Volt

b. Pada keadaan maksimum (suhu 110°F)

Vout = 6,25 x (1100mv–300mv) = 5 Volt

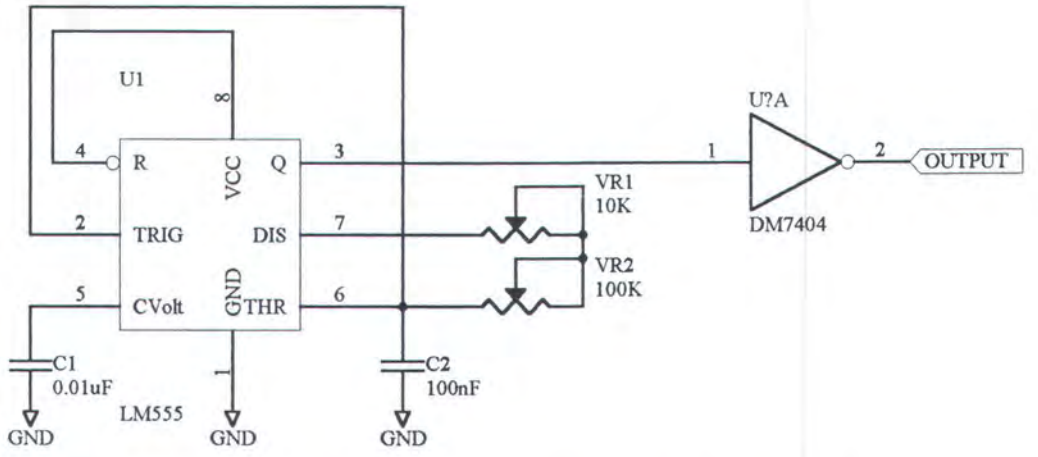
Untuk kalibrasi, dapat dilakukan dengan mengatur potentiometer *multiturn* VR2 sehingga diperoleh nilai suhu udara yang sesuai dengan yang ditunjukkan pada termometer referensi.

3.2.2 MODUL KELEMBABAN TANAH

Untuk mendapatkan data mengenai kelembaban tanah, penulis menggunakan sensor SS200 milik *Irrometer, Corp.* Sensor kelembaban tanah ini dioperasikan dengan menggunakan tegangan pulsa periodik yang memiliki waktu '*high*' sebesar 1 ms, dan waktu '*low*' sebesar 50 ms.

3.2.2.1 Rangkaian Pulse Generator

Gambar rangkaian *pulse generator* untuk menghasilkan sinyal pulsa tersebut adalah sebagai berikut:



Gambar 3.4 Rangkaian Pulse Generator

Pada rangkaian *Astable multivibrator* dipilih C sama dengan 1 μF, sehingga untuk mendapatkan sinyal pulsa tersebut dapat dilakukan perhitungan sebagai berikut :

$$T_L = 0{,}693 \times VR1 \times C \qquad T_H = 0{,}693 \times (VR1 + VR2) \times C$$

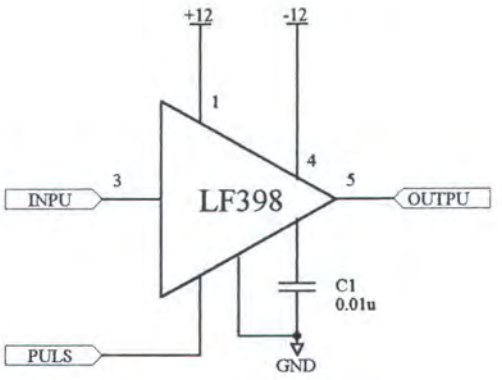
$$VR1 = \frac{T_L}{0{,}693 \times C} \qquad VR2 = \frac{T_H}{0{,}693 \times (VR1 + VR2) \times C}$$

$$VR1 = \frac{1ms}{0{,}693 \times 1\mu F} \qquad VR2 = \frac{50ms}{0{,}693 \times 1\mu F} - VR1$$

$$VR1 = 1443{,}001443\Omega \qquad VR2 = 70{,}7071K\Omega$$

3.2.2.2 Rangkaian Sample and Hold

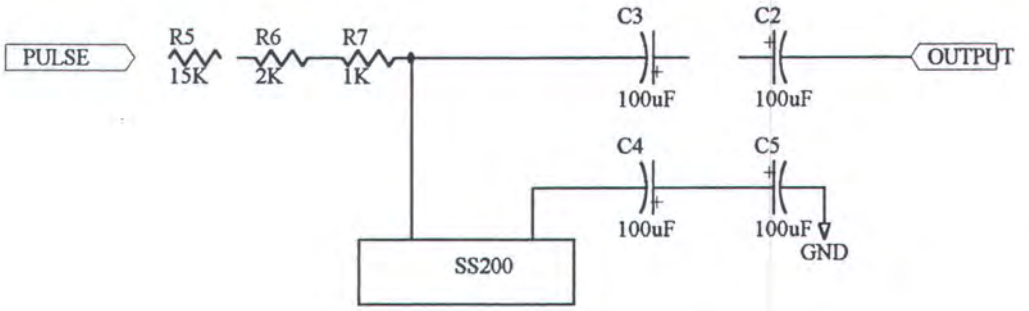
Karena sensor kelembaban tanah SS200 hanya akan menunjukkan hasil pembacaan yang benar (*valid*) pada saat pulsa *high* saja, maka kita perlu menyimpan hasil pembacaan itu saja pada saat pulsa menjadi '*high*'. Hal ini dapat dilakukan dengan menambahkan IC *Sample and Hold* pada modul kelembaban tanah. Gambar rangkaian *Sample and Hold* yang digunakan dapat dilihat pada gambar 3.5.



Gambar 3.5 Gambar Rangkaian *Sample and Hold*

3.2.2.3 Sensor Kelembaban Tanah

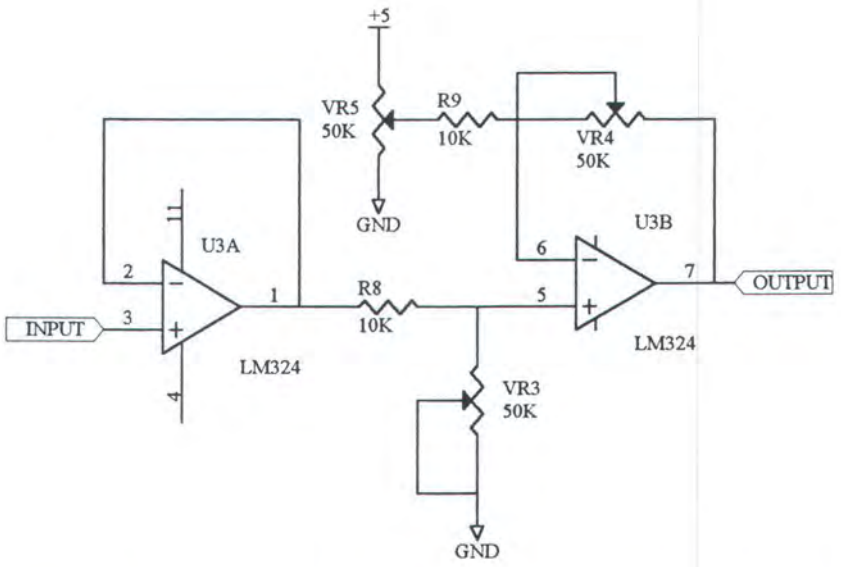
Seperti telah dikemukakan sebelumnya bahwa sensor kelembaban tanah yang digunakan adalah SS200 milik *Irrometer*. Sensor ini pada prinsipnya dioperasikan dalam rangkaian pembagi tegangan. Gambar rangkaian dari sensor kelembaban tanah ini dapat dilihat pada gambar 3.6.



Gambar 3.6 Gambar Rangkaian Sensor SS200

3.2.2.4 Pengkondisian Sinyal

Keluaran dari sensor kelembaban tanah ini kemudian masuk ke rangkaian pengkondisi sinyal, yang terdiri dari rangkaian *buffer* dan rangkaian *differential amplifier*. Rangkaian pengkondisi sinyal ini menggunakan dua buah *Operational Amplifier* yang ada pada IC LM324 (*Operational Amplifier* berempat).



Gambar 3.7 Gambar Rangkaian Pengkondisi Sinyal Sensor Kelembaban Tanah

Rangkaian *buffer* berguna untuk menyangga dan sekaligus sebagai *matching impedance* antara sensor dengan rangkaian *differential amplifier*. Sedangkan rangkaian *differential amplifier* berguna untuk mengeliminasi tegangan pada saat sensor menunjukkan harga 0 *CB* (*centibar*) dan berguna pula untuk menguatkan tegangan keluaran sensor sehingga memiliki range tegangan antara 0 sampai dengan 5 volt (merupakan tegangan masukan yang cocok bagi IC MAX114).

Berikut ini adalah perhitungan-perhitungan pada rangkaian *differential amplifier* :

a. Pada keadaan minimum (CB=0)

$Rx = 550\Omega$

$$Vdrop = \frac{550}{550+17000} \times 5 = 0.1567$$

b. Pada keadaan maksimum (CB=200)

$Rx = 27950\Omega$

$$Vdrop = \frac{27950}{27950+17000} \times 5 = 3{,}1090Volt$$

c. Besar penguatan adalah

$$A = \frac{MaksimumADC}{Maksimumsensor}$$

$$A = \frac{5Volt}{3{,}1090Volt} = 1{,}6082$$

Berdasarkan perhitungan diatas, maka multiturn untuk VR5 diset sehingga menghasilkan tegangan referensi sebesar 0,1567 Volt. Sedangkan multiturn VR6 dan VR7 diset menjadi :

$$VR6 = VR7 = A \times 10K\Omega$$
$$VR6 = VR7 = 1{,}6082 \times 10K\Omega$$
$$VR6 = VR7 = 16{,}082K\Omega$$

Dengan harga-harga diatas maka akan diperoleh tegangan masukan analog bagi kanal kedua ADC MAX114 sebesar 0 Volt sampai dengan 5 volt.

3.2.3 MODUL ANALOG TO DIGITAL CONVERTER

Modul *Analog to Digital Converter* ini terdiri dari IC MAX114 yang dioperasikan dengan menggunakan sumber tegangan 5 Volt yang berasal dari IC *Voltage Reference* MAX6350. Selain itu, penulis juga menggunakan beberapa komponen eksternal seperti kapasitor *bypass* 4,7µF dan 0,1µF pada tegangan masukan analog, serta kapasitor *bypass* 0,1µF pada keluaran IC MAX6350 yang menuju pin REF+ pada IC MAX114. Gambar rangkaian ADC dapat dilihat pada Lampiran A-3.

3.2.4 MODUL LCD (LIQUID CRYSTAL DISPLAY)

Modul *Display* yang digunakan pada Tugas Akhir ini adalah LCD (*Liquid Crystal Display*) M1632 yang memiliki *display* 2 baris x 16 kolom. Tegangan catu yang digunakan adalah 5 Volt, sedangkan untuk mengatur terang gelap tulisan digunakan potensiometer 10KΩ. Port-port data dan sinyal-sinyal kontrol untuk LCD ini dihubungkan dengan pin-pin IC FPGA XC4010XL. Gambar rangkaian untuk LCD dapat dilihat pada Lampiran A-4.

3.2.5 MODUL DRIVER KELUARAN

Modul *driver* keluaran ini berguna untuk men-*drive* sinyal keluaran dari IC FPGA XC4010XL yang berupa sinyal *Pulse Width Modulation* (PWM) sehingga dapat menggerakkan motor DC. Modul ini terdiri dari IC *Operational Amplifier* LF411 yang berfungsi sebagai *buffer* dan transistor BD139 serta MJ2955 yang difungsikan sebagai transistor *switching* untuk menggerakkan motor DC tersebut. Gambar Rangkaian d*river* Keluaran dapat dilihat pada Lampiran A-5

3.2.6 MODUL CATU DAYA

Bagian terpenting dari suatu rangkaian elektronika adalah rangkaian catu daya. Rangkaian catu daya ini berfungsi sebagai sumber tegangan bagi XESS Board XS40-010XL dan rangkaian-rangkaian analog pendukung pada Tugas Akhir ini sebesar +12 Volt dan –12 Volt. Pada dasarnya rangkaian catu daya ini terdiri dari beberapa bagian, yaitu transformator, penyearah, filter, regulator tegangan. Bagian utama dari rangkaian catu daya ini adalah IC regulator tegangan,

yang dirancang untuk dapat mengeluarkan tegangan stabil 12 V (untuk IC LM7812) dan tegangan stabil –12 V (untuk IC LM7912). Untuk dapat mengeluarkan tegangan stabil ±12 Volt, IC ini memerlukan tegangan masukan minimal 14,6 Volt. Tegangan ini diperoleh dari sekunder trafo 12 Volt yang telah disearahkan oleh rangkaian jembatan dioda. Tegangan ini ada sebesar:

$$Vdc = 12\sqrt{2} - 1{,}2$$
$$Vdc = 16{,}9706 - 1{,}2$$
$$Vdc = 15{,}77056 Volt$$

Pemilihan tegangan 15,7706 Volt ini cocok untuk tegangan masukan IC LM7812 dan LM7912, karena dengan tegangan masukan yang lebih kecil (mendekati 12 Volt) akan kita peroleh tegangan keluaran yang kurang stabil. Sebaliknya, dengan tegangan masukan yang lebih besar, disipasi daya pada IC tersebut akan menjadi terlalu besar, dan kemungkinan IC dapat cepat rusak.

Fungsi C1 dan C6 adalah untuk meratakan tegangan yang telah disearahkan oleh rangkaian jembatan dioda, sedangkan kapasitor-kapasitor C2, C4, C7 dan C9 digunakan untuk mengurangi dengung yang diakibatkan oleh peralatan yang kurang sempurna. Dan, kondensator C3 serta C7 berfungsi untuk memperbaiki stabilisasi. Semua kondensator C2, C3, C4, C7, C8, C9 merupakan kondensator tantalum, karena kondensator jenis ini mempunyai penekanan terhadap dengung yang lebih baik dibandingkan dengan kondensator elektrolit biasa. Kondensator C5 dan C10 (jenis mika) juga perlu ditambahkan untuk mengurangi gangguan-gangguan yang ditimbulkan oleh frekuensi tinggi. Gambar rangkaian catu daya ini secara lengkap dapat dilihat pada Lampiran A-6

3.2.7 MODUL XESS BOARD XS40-010XL

Modul *XESS Board XS40-010XL* dari XESS ini digunakan untuk mengimplementasikan modul-modul VHDL yang dibutuhkan dalam Tugas Akhir ini. Modul ini memiliki *IC FPGA XC4010XL* yang berfungsi sebagai pusat perencanaan FPGA, dan IC-IC lain seperti IC mikrokontroler 8051, IC *Static RAM*, 1 buah seven segment, dan IC-IC regulator tegangan. Gambar rangkaian dari *XESS Board XS40-010XL* dapat dilihat pada Lampiran A-7

3.3 PERENCANAAN MODUL FPGA

Modul-modul FPGA (*Field Programmable Gate Array*) yang direncanakan dibuat dengan menggunakan bahasa pemrograman VHDL dengan menggunakan bantuan *Software Synopsys FPGA Express* untuk melakukan sintesa dan analisa. Sedangkan untuk simulasi, baik *timing* maupun fungsional, dan proses implementasi pada IC FPGA XC4010XL menggunakan *Software Xilnx Foundation 1.5*. Sebagai alat implementasi adalah *XESS Board XS40-010XL*. Diagram blok perencanaan modul FPGA dapat dilihat pada lampiran B-1.

3.3.1 MODUL AKUISISI DATA

Modul Akuisisi Data ini berfungsi untuk mengambil data dari IC ADC MAX114 yang berasal dari keluaran sensor kelembaban tanah dan sensor suhu udara. Proses pengambilan data dilakukan terus-menerus dengan frekuensi sampling sebesar 16 KHz, namun data-data tersebut baru akan diolah apabila

terjadi perubahan data. Diagram blok dari Modul Akuisisi Data ini dapat dilihat pada Lampiran B-2.

3.3.1.1 ADC Driver

XESS Board XS4010XL yang menggunakan IC FPGA XC4010XL menggantikan peran mikroprosesor sebagai pengatur ADC (ADC *Driver*), yang diimplementasikan pada modul ADC *Driver*. Modul ADC *Driver* ini mengeluarkan sinyal-sinyal keluaran untuk mengakses IC MAX114:

a. Channel

Digunakan untuk mengakses *channel* masukan dari MAX114 (A1 selalu di*ground*, karena hanya menggunakan 2 buah *channel* masukan yang merupakan kombinasi masukan 00, 01,10,11 bagi pin-pin A0 dan A1).

b. INT

Sinyal ini digunakan untuk menandai berakhirnya proses konversi pada IC MAX114, dan data D0-D7 yang ada pada keluaran MAX114 menjadi *valid* dan siap diambil oleh IC FPGA.

c. RD

Sinyal ini berfungsi sebagai tanda awal konversi dan menaikkan kembali sinyal INT menjadi '*high*'. Pada saat sinyal RD menjadi *low*, ADC akan memulai konversi, dan menunggu pembacaan data setelah sinyal INT menjadi '*low*'. Sinyal INT akan kembali '*low*' setelah sinyal RD dibuat kembali '*high*'.

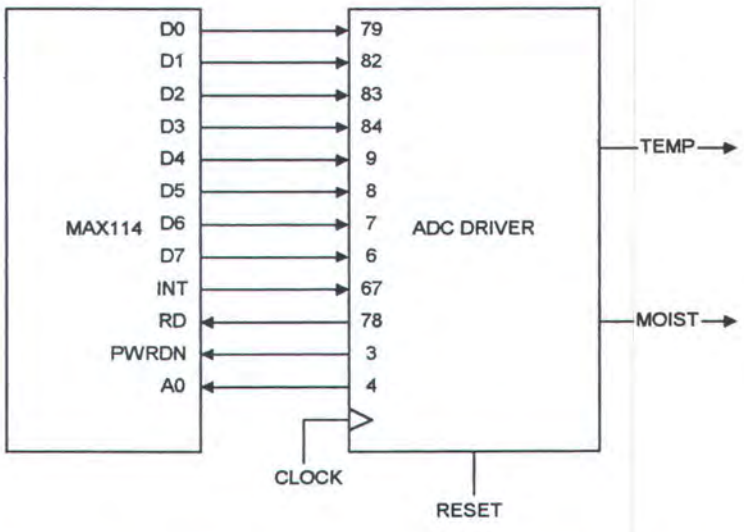
d. PWRDN

Sinyal ini berfungsi untuk menyebabkan ADC berada pada kondisi *power down*. Apabila masukan pada pin PWRDN ini diberi tegangan logika '0' maka

ADC akan mati, arus sumber tegangan 5 Volt akan berkurang menjadi 1μA. Untuk mengaktifkan kembali ADC, dapat dilakukan dengan memberi tegangan masukan logika '1' pada pin PWRDN pada MAX114.

e. *D0-D7*

Sinyal-sinyal ini merupakan Port untuk masuknya data dari ADC menuju IC FPGA. Sifat dari port ini adalah *buffer latched*, yang berarti selama tidak terjadi pengaksesan kembali ADC, maka keluaran dari port ini tetap pada hasil sebelumnya. Selain itu keluaran dari port ini dapat langsung dihubungkan dengan IC FPGA XC4010XL meskipun memiliki tegangan keluaran *high* sekitar 4 Volt, karena IC FPGA ini memiliki sifat '*TTL tolerant I/O*' yang berarti dapat dihubungkan dengan komponen TTL secara langsung tanpa menimbulkan kerusakan meskipun IC FPGA XC4010XL ini beroperasi dalam mode level tegangan 3,3 Volt.



Gambar 3.8 Interkoneksi MAX114 dengan Modul ADC *Driver*

Karena ADC MAX114 tidak membutuhkan *Clock* eksternal, maka kita tidak perlu membuat rangkaian *clock* tambahan bagi ADC MAX114. namun

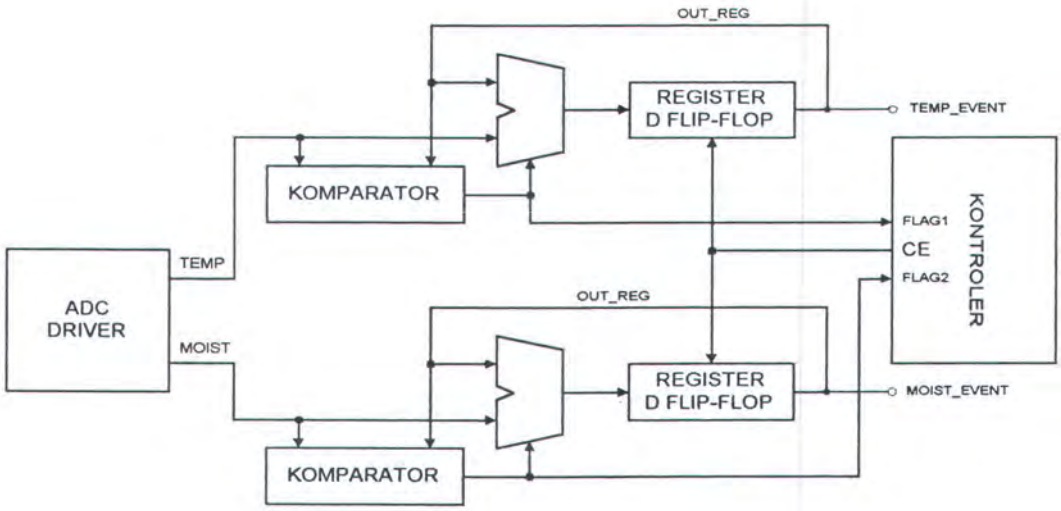
untuk pengoperasian modul ADC *Driver*, kita memerlukan suatu komponen *clock generator*. Komponen ini diperoleh dengan memanfaatkan fasilitas *clock generator* yang ada pada IC FPGA XC4010XL, yang pembahasannya akan dilakukan pada bagian modul *clock generator*.

Pengaksesan alamat *channel* MAX114 dilakukan secara bergantian oleh modul ADC *Driver* dengan memberikan sinyal keluaran pada pin *Channel*, Pengaksesan dilakukan dengan memberikan keluaran '0' terlebih dahulu (yang berarti pengaktifan *channel* 0), baru kemudian pengaktifan *channel* 2 dengan memberikan keluaran '1' pada pin *Channel*.

Untuk mengimplementasikan modul ADC *Driver* ini digunakan rancangan rangkaian sekuensial dengan menggunakan VHDL. Listing program VHDL dari ADC *Driver* ini dapat dilihat pada Lampiran C-1. Proses pembuatan ADC *Driver* ini memerlukan CLB sebanyak 19 CLB dan menggunakan *Clock* dengan frekuensi 16 KHz.

3.3.1.2 Modul Event

Proses pengambilan data dilakukan terus-menerus oleh modul ADC *Driver*, namun data-data tersebut tidak langsung diproses oleh *Fuzzy Controller*, melainkan masuk terlebih dahulu ke modul *Event*. Modul *Event* ini berfungsi untuk membandingkan data yang masuk sekarang dengan data pada siklus sebelumnya. Apabila terjadi perubahan data, maka kontroler baru memproses data secara *fuzzy logic* (logika samar). Diagram blok dari modul *Event* ini dapat dilihat pada gambar 3.9.



Gambar 3.9 Diagram blok Modul *Event*

Untuk mengimplementasikan modul *Event* ini, penulis membuat beberapa komponen yang dapat dijelaskan sebagai berikut :

1. Komparator

Komponen ini berfungsi untuk membandingkan masukan yang berasal dari keluaran sensor pada saat ini dengan keluaran sensor pada saat siklus pengambilan data sebelumnya. Komponen komparator ini dibuat dengan menggunakan fasilitas *Logiblox* yang ada pada *Software Xilnx Foundation 1.5*. Keluaran dari komponen Komparator berupa sinyal Flag, yang berfungsi sebagai penanda bagi modul *Fuzzy Controller* bahwa terjadi perubahan keluaran pada salah satu sensor yang digunakan. Selain itu sinyal ini juga berfungsi untuk memberikan masukan baru bagi modul *Fuzzy Controller*. Jumlah CLB untuk mengimplementasikan komponen Komparator 8 bit ini adalah 4 buah.

2. Multiplexer 2 To 1

Komponen ini berfungsi sebagai pemilih bagi masukan *Fuzzy Controller*. Sinyal pemilihnya sendiri berasal dari sinyal Flag dari komponen Komparator 8

bit. Implementasi komponen *Multiplexer 2 To 1* ini menggunakan fasilitas *Logiblox* yang ada pada *Software Xilnx Foundation 1.5*, dan membutuhkan 4 buah CLB (*Configurable Logic Block*).

3. Register

Komponen Register 8 bit ini berfungsi untuk menyimpan data keluaran sensor, yang akan dibandingkan dengan keluaran sensor untuk siklus konversi ADC berikutnya. Keluaran *Register Data* ini juga berfungsi sebagai masukan bagi modul *Fuzzy Controller*, baik untuk masukan suhu udara (*Temp*) maupun kelembaban tanah (*Moist*). Komponen *Register* ini memiliki masukan sinyal *Enable* yang berfungsi untuk mengaktifkan *Register*. Sinyal *Enable* ini berasal dari modul *Fuzzy Controller*, yang akan aktif apabila terjadi perubahan pada keluaran sensor suhu udara maupun kelembaban tanah. Untuk mengimplementasikan komponen Register 8 bit ini penulis menggunakan pula fasilitas *Logiblox* yang ada pada *Software Xilnx Foundation 1.5*, dan membutuhkan 4 buah CLB (*Configurable Logic Block*).

Modul *Event* ini penulis gunakan pada tiap masukan bagi modul *Fuzzy Controller*, baik untuk sinyal Temp (Suhu Udara) dan Moist (Kelembaban tanah). Oleh karena itu, modul yang digunakan adalah dua buah, dan jumlah CLB (*Configurable Logic Block*) yang digunakan secara keseluruhan untuk mewujudkan proses akuisisi data ini adalah 24 buah. Listing program VHDL untuk membentuk modul *Event* ini dapat dilihat pada lampiran C-2.

Modul ini akan dimasukkan ke dalam rancangan (*design*) utama pada HDL editor dengan menggunakan cara *Instantiate Component*. Informasi-

informasi yang harus dimasukkan dalam program utama mengenai komponen ini dapat dilihat pada file berekstensi *.vhi yang ada pada *directory* atau *folder* proyek *Xilnx Foundation 1.5* yang kita buat.

3.3.2 MODUL FUZZY KONTROLER

Fungsi dari alat ini adalah untuk mengatur lama penyiraman air dengan metode *fuzzy logic* yang dibuat dari IC FPGA XC4010XL berdasarkan masukan yang berasal dari dua buah sensor, yaitu sensor kelembaban tanah dan sensor suhu udara. Seperti diketahui bahwa proses *fuzzy logic* dapat dibagi menjadi 5 buah bagian, yaitu *Fuzzifier, Rule Evaluation, Defuzzifier, Program Storage* dan Kontroler. Diagram bloknya dapat dilihat pada Lampiran B-3.

3.2.2.1 Fuzzifier

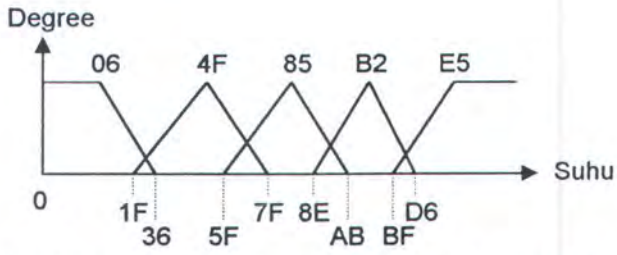
Fungsi dari *fuzzifier* adalah untuk mendapatkan nilai membership (*membership degree*) dari data digital keluaran ADC MAX114 yang bervariasi dari 00H – FFH. Karena ada dua buah masukan untuk proses *fuzzy logic*, maka terdapat dua buah *membership function* yaitu untuk suhu udara dan kelembaban tanah. Berikut ini rangle untuk *membership function* dari suhu udara :

Cold = 30°F - 37°F Cool = 40°F - 70°F

Normal = 60°F - 84°F Warm = 75°F - 98°F

Hot = 90°F - 110°F

Range label diatas dikonversikan dalam bentuk heksa yang bervariasi dari 00H – FFH. *Membership function* dari suhu udara tampak pada gambar 3.10.



Gambar 3.10 Membership function Suhu Udara

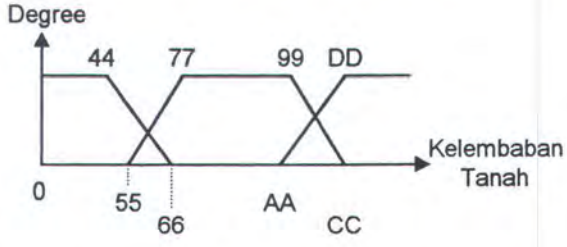
Sedangkan untuk masukan dari sensor kelembaban tanah memiliki range label sebagai berikut :

Dry = 0% – 12%

Moist = 10% - 24%

Wet = 20% - 30%

Membership function dari kelembaban tanah tampak pada gambar 3.11.



Gambar 3.11 Membership function Kelembaban Tanah

Untuk proses implementasi *fuzzifier* dalam VHDL penulis menggunakan proses aritmetika, kombinasi antara perkalian dan penjumlahan untuk mendapatkan *membership degree* dari tiap masukan berdasarkan rumus sebagai berikut :

$$Degree = FF - [(Input - Centre) \times Slope]$$

Dimana :

Masukan = data digital dari MAX114

Centre = Posisi tengah dari sebuah label

Slope = Kemiringan dari sebual label

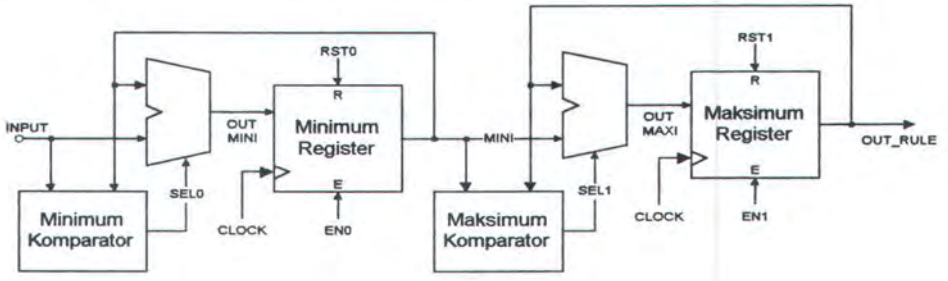
$$Slope = \frac{FFH}{lebarlabel}$$

Karena ada dua buah masukan yang masuk ke *fuzzifier*, yaitu suhu udara dan kelembaban tanah, maka pengoperasian *fuzzifier* ini dilakukan dengan cara *multiplexed in time*. Hal ini menghemat pemakaian IC FPGA dan membutuhkan suatu rangkaian *Controller* tersendiri untuk pengambilan data dan pengolahan data dari semua masukan digital yang masuk ke dalam *fuzzifier*.

Listing program VHDL untuk modul *fuzzifier* dapat dilihat pada Lampiran C-3. Sedangkan jumlah CLB yang dibutuhkan adalah 35 buah.

3.2.2.2 Rule Evaluator

Untuk proses *Rule Evaluation*, penulis menggunakan metode *Min-Max Rule*, yang berarti mencari nilai minimum dari tiap *antecendence* untuk kemudian dicari nilai maksimum dari semua *consequent* yang ada pada tiap *rule*. Diagram blok dari *Rule evaluator* ini dapat dilihat pada gambar 3.12.



Gambar 3.12 Diagram blok Rule evaluator

Operator yang digunakan adalah operator AND yang memiliki bentuk sebagai berikut :

IF *antesendence1* AND *antesendence2* THEN *consequent* _1

⋮

IF *antesendence1* AND *antesendence2* THEN *consequent* _n

Untuk mengimplementasikan *Rule evaluator* dalam VHDL ini, penulis menggunakan beberapa buah modul atau komponen yang sudah disediakan oleh *Software Xilnx Foundation 1.5* (*Logiblox*), yaitu 2 buah komparator (minimum komparator dan maksimum komparator), dua buah register untuk menampung nilai minimum dan nilai maksimum keluaran komparator, dan dua buah multiplekser sebagai pemilih data yang akan diteruskan ke tahap berikutnya. Karena menggunakan *tool Logiblox*, maka perlu digunakan cara *instantiate component* dalam rancangan *rule evaluator* ini.

Register-register yang ada dibuat dari D Flip-Flop yang memiliki reset asinkronus dan *Clock Enable*. Reset RST0 digunakan untuk mereset register minimum menjadi FF, dan reset RST1 digunakan untuk mereset register maksimum menjadi 00. Sedangkan *Clock Enable* digunakan untuk menandakan aktifnya proses baik pada saat mengaktifkan bagian minimum (EN0) maupun pada saat mengaktifkan bagian maksimum (EN1). Pengaktifan sinyal EN0, EN1, RST0, RST1 ini dilakukan oleh kontroler.

Rancangan *Rule evaluator* ini akan disintesa menjadi sebuah makro atau bagian dari rancangan program utama yang nantinya akan diimplementasikan pada IC FPGA XC4010XL.

Untuk mempermudah mendapatkan keluaran pemenang dari tiap *rule* penulis menggunakan aturan FAM (*Fuzzy Associative Matrix*) pada tabel 3.1.

	C1	C2	N	W	H
D	S	S	S	S	S
M	S	M	M	M	M
W	L	L	L	L	L

Tabel 3.1 Diagram Matriks *Rule* Evaluation

Keterangan :

Suhu Udara :

C1=Cold C2=Cool N=Normal W=Warm H=Hot

Kelembaban Tanah:

D=Dry M=Moist W=Wet

Keluaran :

S = Short; M = Medium L = Long

Untuk listing program VHDL dari *rule evaluator* ini dapat dilihat pada lampiran C-4. Sedangkan jumlah CLB yang dibutuhkan adalah 12 buah.

3.2.2.3 Defuzzifier

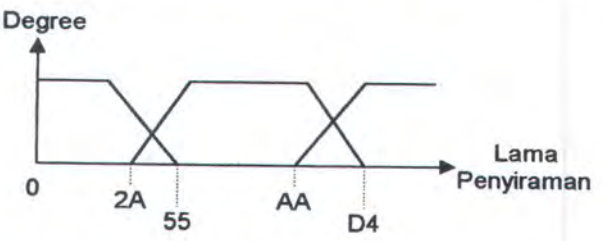
Untuk D*efuzzifier* penulis menggunakan metode COG (*Centre Of Gravity*) *Singleton*. Range label keluaran waktu penyiraman adalah sebagai berikut :

Short = 0 – 20 menit

Medium = 10 – 50 menit

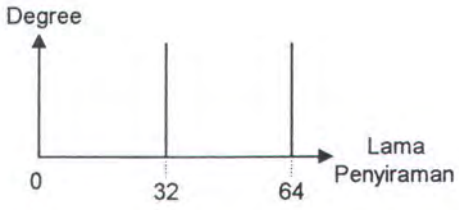
Long = 40 – 60 menit

Sedangkan bentuk *membership function*nya dapat dilihat pada gambar 3.13.



Gambar 3.13 Membership function Aktifnya Sistem Irigasi

Karena menggunakan metode *COG singleton* maka diperoleh membership function seperti pada gambar 3.14.

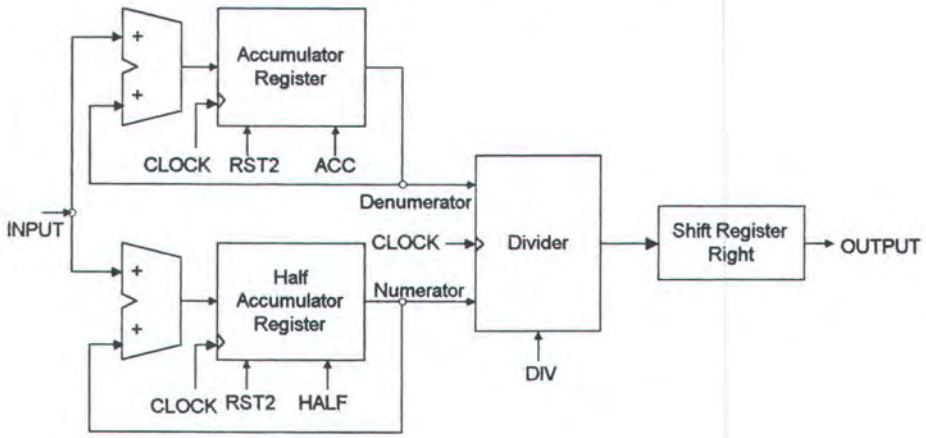


Gambar 3.14 Membership function Tipe Singleton Aktifnya Sistem Irigasi

Untuk memperoleh keluaran defuzzifikasi dilakukan berdasarkan rumus:

$$Output = \frac{\sum_i (fuzzyoutput_i) \times (\sin gleton)}{\sum_i (fuzzyoutput_i)}$$

Untuk mengimplementasikan pada VHDL penulis menggunakan rangkaian penjumlahan, perkalian dan pembagian. Pengoperasian dari rangkaian-rangkaian ini penulis lakukan secara *multiplexed in time* untuk mendapatkan rangkaian d*efuzzifier* yang optimal. Implementasi rancangan d*efuzzifier* ini menggunakan komponen-komponen yang ada pada *Logiblox* dan *Core Generator*. Tiap modul tersebut memiliki sinyal Reset dan *Clock Enable* tersendiri untuk mengaktifkan masing-masing bagian, baik *Accumulator* (sebagai *Clock Enable* adalah ACC) dan *Half Accumulator* (sebagai *Clock Enable* adalah *HALF*), dan sinyal RST2 untuk mereset kedua buah modul tersebut sehingga bernilai '00'.



Gambar 3.15 Diagram blok Modul D*efuzzifier*

Berikut ini adalah perhitungan dari proses de*fuzzifier* :

$$Output = \frac{(A \times 0) + (B \times 32) + (C \times 64)}{A + B + C}$$

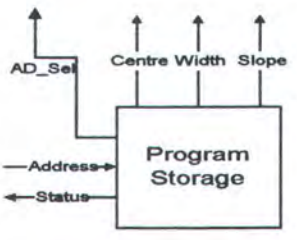
$$Output = \frac{64}{16} \times \frac{(A \times 0) + (B) + (C \times 0.5)}{\left(\frac{A + B + C}{16}\right)}$$

$$Output = 4 \times \frac{Numerator}{Denumerator}$$

Dimana *Numerator* merupakan hasil operasi *Half Accumulator* dan *Denumerator* adalah hasil operasi *Accumulator* dari keluaran *rule evaluator* yang digeser ke kanan sebanyak 4 kali (dibagi 16). Untuk melakukan perkalian dengan bilangan 4 dapat dilakukan dengan menggeser hasil pembagian ke kiri dua kali. Listing program VHDL dari implementasi rangkaian de*fuzzifier* terdiri dari makro *divider* dan program utama de*fuzzifier*, dan dapat dilihat pada lampiran C-5 dan C-6. Total CLB untuk mengimplementasikan modul *divider* adalah 24 buah, sedangkan secara keseluruhan modul *defuzzifier* membutuhkan 39 buah.

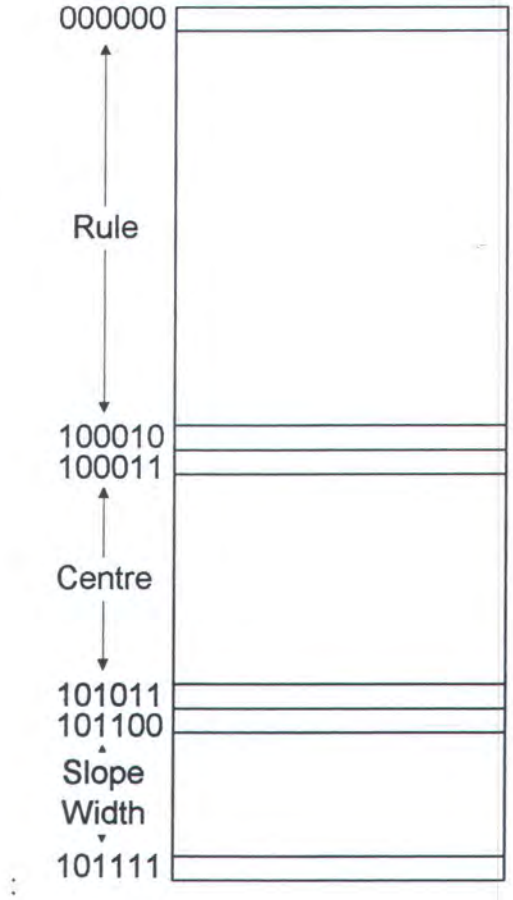
3.2.2.4 Program Storage

Rancangan *Program Storage* berupa ROM (*Read Only Memory*) di dalam IC FPGA XC4010XL sebesar 48x10 bit, yang berfungsi untuk menyimpan *rule-rule* yang digunakan oleh *rule evaluator* dan menyimpan data-data centre, tipe, dan slope yang digunakan oleh *membership function* pada proses *fuzzifier*.



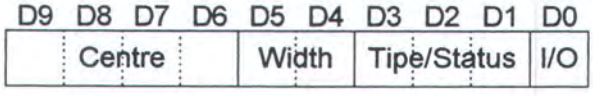
Gambar 3.16 Diagram blok *Program Storage*

Berikut ini adalah pengorganisasian memori pada ROM yang penulis buat



Gambar 3.17 Pengorganisasian Memori pada Kontroler *Fuzzy*

Sedangkan penjelasan dari tiap bit yang ada pada bagian *rule* adalah sebagai berikut :

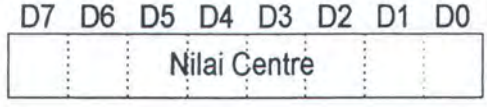


Gambar 3.18 Keterangan Bit-Bit pada Alamat *Rule*

- *Centre* → Alamat nilai centre pada ROM
- *Width* → Alamat nilai width pada ROM
- Tipe/Status → Untuk menentukan tipe dari *membership function* dan status dari kontroler.

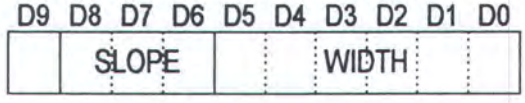
- I/O → untuk menentukan *channel* yang dipilih sebagai masukan MAX114.

Sedangkan isi ROM pada bagian Centre adalah nilai dari Centre sebanyak 8 bit yang digunakan pada rancangan kontroler ini, yaitu ada 9 buah posisi (5 buah untuk *membership function* suhu udara dan 4 buah untuk *membership function* kelembaban tanah).



Gambar 3.19 Keterangan Bit Pada Alamat Centre

Untuk isi ROM pada bagian *Slope Width* dibagi menjadi dua bagian untuk menentukan nilai dari *Width* (6 bit) dan nilai dari *Slope* (3 bit).



Gambar 3.20 Keterangan Bit Pada Alamat Slope dan Width

Pengorganisasian memori secara lengkap dapat dilihat pada lampiran C-7. Implementasi *Program Storage* ini menggunakan fasilitas *Logiblox* yang ada pada perangkat lunak (*Software*) Xilinx Foundation Series 1.5 , yaitu dengan memilih *option* ROM dan memasukkan nilai-nilai dari tiap alamat ROM pada file Program.mem. Karena hanya terdiri dari ROM, maka total CLB yang dibutuhkan untuk mengimplementasikan ROM 48x10 bit adalah 18 CLB.

3.2.2.5 Kontroler

Kontroler merupakan bagian dari proses *fuzzy logic* yang berfungsi untuk mengatur jalannya proses. Kontroler ini berupa *Finite State Machine* yang

memiliki 4 buah *state*, yaitu *state* ADC untuk mengaktifkan ADC *Driver*, *state* Control_Centre untuk menentukan nilai centre *membership function* berdasarkan *rule* yang ada pada ROM, *state* Control_Width untuk menentukan nilai width *membership function* berdasarkan *rule* yang ada pada ROM, *state* Control_Slope untuk menentukan nilai slope *membership function* berdasarkan *rule* yang ada pada ROM, dan *state* Control_Status untuk menentukan apa yang harus dilakukan oleh kontroler. Untuk *state* Control_Status ini dilakukan pengaktifan sinyal-sinyal yang digunakan untuk mengatur pengaktifan register-register yang ada pada ADC, *rule evaluator* maupun de*fuzzifier*. Selain itu, pada *state* ini juga dilakukan penentuan tipe dari *membership function* yang digunakan pada tiap *fuzzy* variabel. Berikut ini adalah sinyal-sinyal kontrol yang ada pada bagian kontroler :

a. RST0 → Mereset Register Minimum menjadi "00".
b. RST1 → Mereset Register Maksimum menjadi "00"
c. RST2 → Mereset Register *Accumulator* dan *Half Accumulator* .
d. AD_Sel → Sebagai sinyal untuk memilih *channel* yang aktif pada MAX114.
e. EN0 → Sebagai *Clock Enable* Register Minimum.
f. EN1 → Sebagai *Clock Enable* Register Maksimum
g. ACC → Sebagai *Clock Enable* Register *Accumulator*
h. HALF → Sebagai *Clock Enable* Register *Half Accumulator* .
i. DIV → Sebagai sinyal *Enable* proses *Divider* .

Pada kontroler ini juga terdapat dekoder alamat ROM yang digunakan untuk mengaktifkan salah satu alamat pada ROM untuk mendapatkan nilai-nilai Centre, Width dan slope dari *membership function* variabel *fuzzy*.

Rancangan kontroler ini terletak dalam program utama, dan listing programnya dapat dilihat pada lampiran C-8. Secara keselruhan, total CLB yang dibutuhkan untuk mengimlementasikan modul kontroler ini adalah 14 buah.

3.3.3 MODUL DISPLAY

Modul *display* (tampilan) berfungsi untuk menampilkan data suhu udara yang dibaca oleh sensor suhu LM34 (dalam derajat Fahrenheit) dan sensor kelembaban tanah 200SS (dalam % yang menunjukkan banyak kandungan air dalam tanah), serta lamanya sistem irigasi aktif (dalam menit). Modul *display* ini terdiri dari beberapa modul VHDL yang akan penulis bahas satu-persatu. Diagram blok dari modul *display* ini dapat dilihat pada Lampiran B-4.

3.3.3.1 Modul Temperature Dekoder

Modul Suhu Dekoder (*Temp_Dekoder*) ini merupakan modul yang berfungsi untuk mengubah keluaran ADC *channel* pertama (dalam bentuk heksa desimal) menjadi keluaran sensor suhu udara (dalam bentuk heksa desimal).

Karena keluaran sensor suhu memiliki range antara 30°F sampai dengan 110°F, maka untuk mencapai nilai range ADC MAX 114 (0 Volt sampai dengan 5 Volt) diperlukan rangkaian *differential amplifier* (menggunakan IC *Operational Amplifier* LF411) untuk mengeliminasi tegangan yang ditimbulkan oleh suhu dibawah 30°F dan menguatkan tegangan keluaran sensor LM34 sehingga keluaran maksimumnya dapat mencapai range maksimum dari masukan ADC MAX114 yaitu 5 Volt. Perhitungan tegangan masukan ADC adalah sebagai berikut :

$$InputADC = [(suhu - 300mv) \times A]$$

$$A = \frac{5Volt}{RS \times 10mv}$$

$$RS = 110 - 30 = 80$$

Maka :

$$A = \frac{5Volt}{80mv} = 62,5$$

dimana :

A = Penguatan modul Pengkondisi sinyal yang terdiri dari rangkaian *Differential amplifier* menggunakan IC *Operational Amplifier* LF411.

RS = Range keluaran dari suhu sensor yang akan diukur atau yang diinginkan. Untuk *Fuzzy Controller* ini, range suhu yang diperlukan adalah 30°F sampai dengan 110°F.

Sedangkan untuk mendapatkan keluaran ADC, kita dapat mencarinya dengan menggunakan perhitungan berikut ini :

$$OutputADC = \frac{InputADC}{LSB}$$

$$LSB = \frac{Max}{256} = \frac{5Volt}{256}$$

$$LSB = 19,53125mvolt$$

Dimana :

LSB = Resolusi dari IC ADC MAX114,yang menunjukkan bahwa 1 bit LSB sama dengan 19,53125 milivolt.

Hasil pembagian tersebut (sinyal Keluaran ADC) ini berbentuk bilangan heksadesimal. Karena yang akan ditampilkan pada LCD (*Liquid Crystal Display*) adalah keluaran dari sensor suhu (dalam bentuk derajat Fahrenheit), maka diperlukan suatu perhitungan balik sebagai berikut :

$$InADC = OutADC \times LSB$$
$$InADC = [(OS \times 10mv) - 300mv] \times A$$
$$A = \frac{5Volt}{800mvolt}$$

$$OS = \left[\left(InADC \times \frac{800mv}{5V}\right) + 300mv\right] \times \frac{1}{10m}$$
$$OS = \left[\left(OutADC \times \frac{5V}{256}\right) \times \frac{80}{5}\right] + 30$$
$$OS = \left(\frac{5}{16} \times OutADC\right) + 30 = \left[\frac{(4+1)}{16} \times OutADC\right] + 30$$
$$OS = \frac{OutADC}{4} + \frac{OutADC}{16} + 30$$

Berdasarkan perhitungan-perhitungan diatas, kita dapat mengimplementasikan Suhu Dekoder ini menggunakan VHDL dengan mudah. Suku pertama dari keluaran sensor dapat diimplementasikan dengan menggunakan *Shift Register Right* (SHR) sebanyak 2 kali, atau dapat langsung diimplementasikan dengan mengambil keluaran ADC dari bit ketujuh sampai dengan bit kelima. Sedangkan untuk suku kedua, dapat diimplementasikan dengan menggunakan *Shift Register Right* sebanyak 4 kali atau dapat diakukan dengan mengambil keluaran ADC dari bit ketiga sampai dengan bit kenol. Kedua buah suku tersebut kemudian dijumlahkan dengan menggunakan komponen Penjumlahan. Keluaran dari Penjumlahan kemudian dimasukkan kembali ke komponen Penjumlahan kedua bersama dengan konstanta bilangan 30 untuk memperoleh hasil keluaran (berupa bilangan heksa desimal), yang nantinya akan berguna sebagai masukan bagi modul *Heksa to Desimal Converter*. Penggunaan operator Penjumlahan ('+') ini memiliki performansi area tinggi (minimum area), karena *Software* FPGA Express 3.2 mampu memanfaatkan penggunaan *dedicated*

carry logic secara langsung sesuai dengan arsitektur IC FPGA XC4010XL sehingga dapat menghemat CLB (*Configurable Logic Block*).

Listing program percangan modul Suhu dekoder menggunakan VHDL ini dapat dilihat pada Lampiran C-9. Perancangan ini secara keseluruhan membutuhkan 9 buah CLB (*Configurable Logic Block*).

3.3.3.2 Modul Moisture Dekoder

Modul Moisture Dekoder (Moist_Dekoder) ini merupakan modul yang berfungsi untuk mengubah keluaran ADC *channel* kedua (dalam bentuk heksa desimal) menjadi keluaran sensor kelembaban tanah (dalam bentuk heksa desimal) yang dinyatakan dalam persen kandungan air dalam tanah.

Karena data yang diperoleh dari keluaran sensor kelembaban tanah berupa tegangan, dimana makin kering tanah maka makin besar tegangannya (resistansi sensor besar) dan makin basah tanah tegangan keluaran sensor semakin kecil (resistansi sensor kecil). Adapun nilai range keluaran sensor tersebut adalah 0% sampai dengan 30%, yang sebanding dengan (00)H sampai dengan (FF)H. Jadi untuk mendapatkan data tampilan LCD untuk kelembaban tanah dapat dilakukan dengan perhitungan sebagai berikut :

$$display = \frac{outputADC}{FF} \times 30\%$$

Listing program perancangan modul Moisture dekoder dapat dilihat pada Lampiran C-10. Perancangan ini secara keseluruhan membutuhkan 9 CLB.

3.3.3.3 Modul Hexa To Desimal Converter

Modul *Heksa to Desimal Converter* (Hex_To_Dec) ini berfungsi untuk mengubah bilangan heksadesimal menjadi bilangan desimal sehingga dapat

langsung ditampilkan di LCD. Masukan modul ini berasal dari modul *Suhu Decoder* (Temp_Dec) yang sudah berupa keluaran sensor suhu udara dalam bentuk heksadesimal serta modul *Moisture Dekoder* (Moist_Dec) yang sudah berupa keluaran sensor kelembaban tanah dalam bentuk heksadesimal.

Pada prinsipnya modul ini berfungsi untuk membagi suatu bilangan dengan bilangan 10 (atau 'A' dalam heksadesimal) untuk mendapatkan sisa pembagian. Sisa pembagian inilah yang nantinya akan ditampilkan pada layar LCD (*Liquid Crystal Display*). Sedangkan hasil pembagian dijadikan sebagai masukan atau bilangan yang akan dibagi pada proses selanjutnya. Proses pembagian ini dilakukan sebanyak tiga kali karena kita menginginkan hasil tersebut dipecah dalam bentuk ratusan, puluhan, dan satuan. Implementasi pembagian dalam VHDL pada dasarnya merupakan proses penggeseran 1 bit ke kanan dan proses pengurangan yang dilakukan sebanyak jumlah bit bilangan yang akan dibagi. Dalam modul ini bilangan yang akan dibagi berjumlah 8 bit, sedangkan bilangan pembagi adalah suatu konstanta, yaitu bilangan 10 (atau 'A' dalam heksadesimal), sedangkan hasil konversi yang merupakan sisa pembagian berjumlah 4 bit pula.

Pengiriman keluaran modul *Heksa to Desimal Converter* ini ke modul LCD *Driver* ditandai dengan pengaktifan sinyal Byte_Valid (aktif *high*), yang menandakan bahwa data hasil sisa pembagian akan ditampilkan di LCD. Sedangkan proses pembagian sendiri berlangsung dengan aktifnya sinyal Byte_Request yang berasal dari modul LCD *Driver* sekaligus untuk menentukan

masukan bagi modul Heksa to Decimal Converter yang bisa berasal dari sensor suhu udara, sensor kelembaban tanah maupun lama sistem irigasi aktif.

Listing program VHDLnya dapat dilihat pada Lampiran C-11. Implementasi rancangan modul ini memerlukan 25 buah CLB pada IC FPGA XC4010XL.

3.3.3.4 Data LCD

Modul data LCD berfungsi umtuk menginisialisai LCD (*Liquid Crystal Display*) sehingga dapat digunakan sebagai modul tampilan, dan memberikan data awal bagi LCD sehingga akan tampak karakter-karakter awal bagi LCD. Rancangan Data LCD ini berupa ROM (*Read Only Memory*) di dalam IC FPGA XC4010XL (*on chip ROM*) sebesar 32x9 bit.

Untuk mengimplementasikan ROM ini penulis menggunakan fasilitas *Logiblox* yang ada pada *Software Xilnx Foundation 1.5*, dimana data-data yang akan dikirimkan ke LCD tersebut dimuat dalam file ROM_LCD.mem. Isi dari file ini dapat dilihat pada Lampiran C-12.

Data-data yang tersimpan dalam ROM tersebut terdiri dari dua bagian, yaitu 8 bit pertama (D0-D7) sebagai data LCD yang akan dimasukkan ke pin-pin D0-D7 dari LCD, serta D8 yang berfungsi sebagai *Register Select* (RS) bagi pin keempat dari LCD. Bit *Register Select* ini bila bernilai '0' berarti LCD akan mengirimkan data berupa instruksi register yang akan menyuruh LCD melakukan suatu instruksi tertentu, sedangkan apabila register ini bernilai '1', maka data yang dikirimkan ke LCD adalah huruf atau karakter yang akan ditampilkan pada LCD.

Berikut ini adalah penjelasan data-data 8 bit yang akan dikirimkan ke LCD yang terdapat dalam ROM dengan alamat mulai dari (00000)H sampai dengan (11111)H :

030 : Data LCD dioperasikan dengan kapasitas 8 bit (D0-D7)

038 : Data LCD dioperasikan dengan kapasitas 8 bit (D0-D7)

008 : *Display Off*

001 : *Display Clear*

006 : Alamat DD RAM ditambah satu bila sebuah karakter sudah ditulis pada DD RAM, dan tampilan LCD tidak bergeser.

00C: LCD *On*, kursor tidak ditampilkan, dan tidak berkedip.

153 : Tampilkan karakter 'S' pada LCD

175 : Tampilkan karakter 'U' pada LCD

168 : Tampilkan karakter 'H' pada LCD

120 : Berikan satu jarak kosong pada LCD.

14C : Tampilkan karakter 'L' pada LCD

165 : Tampilkan karakter 'E' pada LCD

16D : Tampilkan karakter 'M' pada LCD

162 : Tampilkan karakter 'B' pada LCD

161 : Tampilkan karakter 'A' pada LCD

0C3 : Tempatkan kursor pada baris kedua kolom ketiga.

1DF: Tampilkan karakter '°' pada LCD

146 : Tampilkan karakter 'F' pada LCD

0C9 : Tempatkan kursor pada kolom kesembilan baris kedua.

125 : Tampilkan karakter '%' pada LCD

0CF: Tempatkan kursor pada kolom kelimabelas baris kedua pada LCD.

127 : Tampilkan karakter ''' pada LCD

004 : Alamat DD RAM dikurangi satu tiap kali sebuah karakter ditulis pada LCD. Tampilan LCD tidak bergeser.

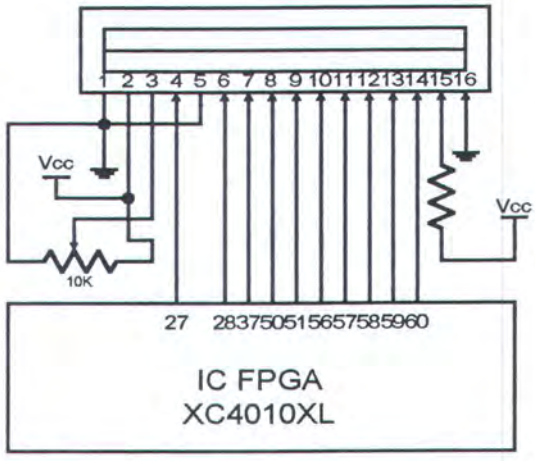
Implementasi ROM untuk LCD ini membutuhkan 9 buah CLB (*Configurable Logic Block*) yang difungsikan sebagai ROM (*Read Only Memory*, dimana 1 buah CLB setara dengan 16 x 2 bit ROM. Keuntungan pemakain *on chip* ROM ini adalah waktu akses yang lebih cepat dibandingkan ROM eksternal dan tidak membutuhkan komponen-komponen penunjang lainnya. Pengaksesan ROM ini melalui alamatnya, yang dilakukan oleh modul *Counter* dan LCD *Driver*.

3.3.3.5 LCD Driver

Modul LCD *Driver* ini terdiri dari proses inisialisasi LCD, penampilan data karakter-karakter awal dan penampilan data suhu udara, kelembaban tanah dan lama aktifnya sistem irigrasi.

Untuk Proses inisialisasi LCD dilakukan dengan bantuan ROM (*Read Only Memory*) yang berisi data-data yang akan dikirimkan ke LCD. Sedangkan untuk proses penampilan data awal juga menggunakan data-data yang tersimpan dalam ROM, namun data-data tersebut tidak ditulis pada Register Instruksi LCD melainkan ditulis pada Register Data LCD. Data-data tersebut akan ditampilkan pada layar LCD.

Interkoneksi atau hubungan antara LCD dengan *XESS Board XS4010XL* maupun rangkaian eksternal lainnya dapat dilihat pada gambar 3.23.



Gambar 3.23 Intekoneksi LCD dengan LCD Driver

Pada gambar tersebut, tampak bahwa pin R/W pada LCD dihubungkan dengan *ground*, karena pada proyek tugas akhir ini LCD hanya berfungsi untuk menampilkan data saja. Untuk pin-pin data D0-D7 LCD dihubungkan dengan pin-pin general purpose I/O FPGA.

Implementasi modul LCD *Driver* dengan menggunakan VHDL ini terdiri dari 4 (empat) buah *state*, yaitu *State* Inisialisasi, *State* Dekoder_Posisi, *State* Out_Data, *State Enable*_Data.

a. *State* Inisialisasi, berfungsi untuk menginisialisasi LCD sehingga dapat berfungsi untuk menampilkan data sepanjang 8 bit, selain itu *state* ini juga berfungsi untuk menampilkan karakter-karakter awal bagi LCD. Diagram alir dari inisialisasi LCD ini dapat dilihat pada Lampiran B-5.
b. *State* Dekoder_Posisi, berfungsi untuk menempatkan kursor pada posisi tertentu pada LCD, yaitu untuk menampilkan suhu udara kursor lebih dulu ditempatkan pada baris kedua kolom kedua. Sedangkan untuk menampilkan kelembaban tanah, kursor terlebih dahulu ditempatkan pada baris kedua kolom kedelapan, dan untuk menampilkan lama aktifnya sistem irigasi kursor lebih

dulu ditempatkan pada baris kedua kolom kelima belas. Pada *state* ini, sinyal Byte_Request dibuat aktif yang merupakan sinyal masukan bagi modul *Hex to Decimal Converter*, sebagai tanda awal dimulainya proses konversi atau pembagian dengan bilangan 10.

c. *State* Out_Data, berfungsi untuk mengeluarkan data yang berasal dari modul Heksa to Decimal Converter. *State* ini berfungsi untuk mengeluarkan data keluaran sensor (baik suhu udara, kelembaban tanah) maupun lamanya sistem irigrasi aktif dengan menunggu aktifnya sinyal Byte_Valid (aktif *high*).

d. *State Enable*_Data, berfungsi untuk meneruskan data-data D0-D7 ke LCD. Proses penerusan data-data LCD dilakukan dengan memberikan sinyal EN '*low*' setelah aktifnya sinyal EN ('*high*') pada *state* Out_Data. Pada *state* ini apabila digit dari bilangan yang ditampilkan sudah mencapai digit ketiga, maka *state* akan kembali pada *state* dekoder_posisi untuk menampilkan data berikutnya.

Listing program dari LCD *Driver* ini dapat dilihat pada Lampiran C-13. Jumlah CLB untuk mengimplementasikan LCD *Driver* adalah 19 buah.

3.3.3.6 Multiplekser 3 Ke 1

Modul multiplekser 3 ke 1 berfungsi untuk memilih data yang masuk ke modul *Hexa to Desimal Converter*. Masukan multiplekser ini berasal dari modul *Suhu Dekoder, Moisture Dekoder* dan Keluaran *Fuzzy Controller*. Sedangkan sinyal pemilih atau selektornya berasal dari modul LCD *Driver* yaitu melalui sinyal MUX_Sel.

Untuk mengimplementasikan modul Multiplekser 3 ke 1 dengan 8 masukan ini penulis menggunakan fasilitas *Logiblox* yang dimiliki oleh *Software Xilnx Foundation 1.5*. Total CLB yang diperlukan untuk implementasi modul ini sebanyak 8 (delapan) buah.

3.3.4 MODUL DRIVER OUTPUT

Untuk mengatur hidup matinya motor untuk menggerakkan katup irigasi, penulis membuat modul untuk mengubah keluaran dari modul *Fuzzy Controller* menjadi keluaran yang mampu mengatur hidup matinya motor. Modul tersebut adalah PWM (*Pulse Width Modulation*) *Driver*. PWM *Driver* ini berfungsi untuk mengubah keluaran linier dari *Fuzzy Controller* yang berupa lama aktifnya sistem irigasi (dalam menit) menjadi bentuk pulsa yang memiliki lebar pulsa sesuai dengan keluaran linier tersebut.

Keluaran lebar pulsa ini diimplementasikan dengan membandingkan keluaran liner modul *Fuzzy Controller* dengan sebuah sinyal *ramp* (*ramp signal*), yang dibentuk dari *timer* 8 bit di dalam IC FPGA XC4010XL. Untuk mengetahui proses pembentukan pulsa tersebut secara lebih jelas dapat dilihat pada Lampiran B-6. Sedangkan Diagram blok dari pembentuk modul PWM *Driver* ini dapat dilihat pada Lampiran B-7. Listing program VHDLnya dapat dilihat pada lampiran C-14.

3.3.4.1 Komparator

Komponen ini berfungsi untuk membandingkan keluaran yang berasal dari *Fuzzy Controller* pada saat ini dengan keluaran *Fuzzy Controller* pada saat siklus

sebelumnya. Keluaran dari komponen Komparator berupa sinyal New_Data, yang berfungsi sebagai penanda bagi komponen *Down Counter* 8 bit bahwa terjadi perubahan masukan, dan data tersebut dimasukkan sebagai data hitungan baru bagi *Counter* tersebut. Sinyal New_Data ini juga berfungsi untuk memberikan sinyal Load bagi *Down Counter* 8 bit, sehingga komponen *Down Counter* segera mendapatkan data baru untuk memulai perhitungan. Selain itu, sinyal New_Data juga berfungsi sebagai masukan D Flip-Flop yang nantinya berfungsi untuk mengaktifkan *Clock* dari komponen *Down Counter* 8 bit. Komponen komparator ini dibuat dengan menggunakan fasilitas *Logiblox*, dan jumlah CLB yang dingunakan untuk mengimplementasikan Komparator 8 bit adalah 4 buah.

3.3.4.2 Multiplekser 2 Ke 1

Komponen ini berfungsi sebagai pemilih bagi masukan *Down Counter*. Sinyal pemilihnya sendiri berasal dari sinyal New_Data yang berasal dari keluaran komparator. Implementasi komponen Multiplekser 2 ke 1 ini menggunakan fasilitas *Logiblox*, dan membutuhkan 4 buah CLB (*Configurable Logic Block*).

3.3.4.3 Register

Komponen Register 8 bit ini berfungsi untuk menyimpan data keluaran *Defuzzifier* yang ada pada *Fuzzy Controller*. Data pada register ini akan dibandingkan dengan keluaran *defuzzifier* pada siklus berikutnya. Keluaran *Register Data* ini juga berfungsi sebagai masukan bagi komponen *Down Counter* 8 bit sebagai data hitungan baru jika terjadi perubahan keluaran *Defuzzifier* pada *Fuzzy Controller*.

Untuk mengimplementasikan komponen Register 8 bit ini penulis menggunakan fasilitas *Logiblox* yang ada pada *Software Xilnx Foundation 1.5*, dan membutuhkan 4 buah CLB (*Configurable Logic Block*).

3.3.4.4 Down Counter

Komponen *Down Counter* 8 Bit ini berfungsi untuk melakukan hitungan mundur terhadap data yang keluar dari *Defuzzifier* pada *Fuzzy Controller*. Data tersebut berupa lama sistem irigasi tersebut aktif dan merupakan data keluaran linier dari proses *Fuzzy Controller*. Apabila terjadi perubahan data, maka data akan di-*update* dengan cara memasukkan keluaran baru dari *Defuzzifier* pada sinyal data D0-D7 komponen *Down Counter* 8 bit dan mengaktifkan sinyal Load. Proses perubahan keluaran ini sendiri tidak dapat terjadi begitu saja, namun setelah data baru di-*load* ke *Down Counter* dan sinyal Load aktif (*high*) oleh aktifnya sinyal pulsa New_Data, maka *Down Counter* baru akan di-*update* pada saat *Clock* berikut (*transistion edge*). Karena sinyal *Clock* yang masuk ke *Down Counter* memiliki frekuensi 1 menit, maka *Counter* 8 bit tidak dapat langsung di-*update*. Untuk mengatasi hal ini dapat dilakukan dengan meng-*OR*-kan sinyal New_Data dengan sinyal *Clock* dengan frekuensi 1 Hz atau waktu periode 1 detik.

Untuk mengimplementasikan komponen *Counter* ini, penulis menggunakan fasilitas *Logiblox* yang dimiliki oleh *Software Xilnx Foundation 1.5*. Jumlah CLB (*Configurable Logic Block*) yang digunakan untuk mengimplementasikan *Down Counter* 8 bit ini adalah 4 buah.

3.3.4.5 Ramp Generator

Komponen *Ramp Generator* ini digunakan untuk membentuk sinyal ramp (*Ramp Generator*) bagi pembentuk sinyal keluaran PWM (*Pulse Width Modulation*). Pada dasarnya komponen ini merupakan modul *Up Counter* 6 bit. Komponen ini mulai menghitung mulai dari 0 sampai dengan batas yang telah ditentukan. Untuk menetukan batas-batas penghitungan dan nilai awal penghitungan kita dapat memasukkan bilangan-bilangan tersebut melalui fasilitas *Logiblox* pada *Software Xilnx Foundation 1.5*. Untuk awal penghitungan adalah 0, sedangkan hitungan maksimum dibuat pada nilai 60.

Sinyal *Clock* untuk modul *Ramp Generator* ini memiliki frekuensi sebesar 1 Hz atau periodenya 1 detik. Hal ini dimaksudkan agar pada saat katup irigasi aktif, maka akan terjadi pembukaan katup selama 1 detik dan kemudian katup tertutup kembali selama satu detik berikutnya. Hal ini dimaksudkan agar tidak terjadi saturasi pada tanah, karena proses pengambilan data untuk kelembaban tanah tidak dapat berlangsung secara terus-menerus. Modul *Ramp Generator* ini dibuat dengan menggunakan faslitas *Logiblox Software Xilnx Foundation 1.5*, dan total CLB (*Configurable Logic Block*) yang dibutuhkan setelah diimplementasi sebagai sebuah makro adalah 4 buah.

3.3.4.6 Komparator Less Than

Komponen komparator *Less Than* ini berfungsi untuk membandingkan sinyal keluaran *Ramp Generator* dengan sinyal keluaran *Down Counter* 8 bit. Apabila sinyal keluaran *Up Counter* 8 Bit lebih besar dibandingkan dengan keluaran komponen *Ramp Generator*, maka keluaran dari komparator ini adalah

'*high*'. Sedangkan jika keluran *Down Counter* lebih kecil dibandingkan dengan keluaran komponen *Ramp Generator*, maka komponen Komprator *Less Than* akan menghasilkan sinyal '*low*'. Sinyal keluaran dari Komparator *Less Than* ini akan berupa sinya-sinyal pulsa yang memiliki lebar pulsa yang bervariasi, bergantung sinyal keluaran linier dari proses *Fuzzy Controller*. Sinyal keluaran inilah yang nantinya akan menggerakkan motor DC.

Untuk mengimplementasikan komponen Komparator *Less Than* ini, penulis menggunakan fasilitas *Logiblox* yang ada pada *Software Xilnx Foundation 1.5*, dan membutuhkan 4 buah CLB (*Configurable Logic Block*).

3.3.4.7 D Flip-Flop

Komponen D Flip-Flop berfungsi untuk memberikan data baru pada *Down Counter* 8 bit. Komponen ini berfungsi untuk menunda sinyal New_Data sebesar 1 *Clock*. Hal ini dimaksudkan agar setelah sinyal Load aktif oleh adanya sinyal New_Data, maka sinyal keluaran D Flip-Flop ini akan mengaktifkan *Clock* tanpa harus menunggu aktifnya *Clock Down Counter* yang memiliki periode sebesar 1 menit. Hal ini dimaksudkan agar data bagi *Down Counter* 8 Bit dapat langsung di-*update* pada saat terjadi perubahan data.

3.3.4.8 Clock Divider

Komponen *Clock Divider* ini berfungsi untuk membuat frekuensi yang dibutuhkan oleh masing-masing komponen yang ada pada modul PWM *Driver*. Untuk komponen *Ramp Generator* dibutuhkan *Clock* sebesar 1 Hz atau periode 1 detik, sedangkan untuk komponen *Down Counter* 8 Bit dibutuhkan sinyal *Clock* dengan periode 1 menit, karena satuan dari waktu irigasi adalah per menit.

Sedangkan untuk komponen-komponen lainnya, sinyal *Clock* berasal dari sinyal *Clock* sistem secara keseluruhan.

Untuk mengimplementasikan modul *Clock Divider* ini dilakukan dengan menggunakan fasilitas *Clock Divider* yang ada pada *Logiblox Software Xilnx Foundation 1.5*.

3.3.5 MODUL CLOCK GENERATOR

Modul *Clock generator* ini berfungsi untuk memberikan *Clock* bagi setiap modul yang ada di dalam perancangan FPGA. Berikut ini adalah tabel frekuensi *Clock* yang dibutuhkan oleh masing-masing modul.

Tabel 3.2 Tabel Frekuensi *Clock* Tiap Modul

Nomor	Modul	Fekuensi Clock
1	Akuisisi Data	16 KHz
2	*Fuzzy Controller*	16 KHz
3	*Display*	16 KHz
4	*Driver* Keluaran	16 KHz
5	Inisialisasi LCD	490 Hz
6	PWM *Driver*	15 Hz

Untuk membuat frekuensi-frekuensi *Clock* tersebut, penulis menggunakan komponen milik Xilinx, yaitu *On Chip Oscillator*. Cara penggunaan komponen ini adalah dengan cara menempatkan simbol OSC4 pada HDL editor, sama seperti penempatan komponen yang menggunakan cara instantiate component. Selain itu untuk mendapatkan beberapa frekuensi yang lain digunakan modul *Clock Divider*, yang ada pada fasilitas *Logiblox* milik *Xilnx Foundation 1.5*. Yang perlu diperhatikan disini adalah keluaran yang dapat dipakai dari simbol OSC4 ini harus maksimal berjumlah tiga macam frekuensi, dimana salah satu frekuensinya harus

8 MHz. Listing program VHDL untuk modul *clock generator* ini dapat dilihat pada Lampiran C-15.

BAB IV
PENGUJIAN DAN PENGUKURAN

BAB IV

PENGUJIAN DAN PENGUKURAN

Pengujian alat dilakukan untuk mengetahui bekerja tidaknya fungsi-fungsi yang telah direncanakan dan mengetahui kemampuan kerjanya, dimana pengujian dilakukan pada rangkaian perangkat keras luar dan modul-modul FPGA dalam IC FPGA XC4010XL.

Untuk pengujian modul FPGA kami lakukan dengan dua cara, yaitu melalui Simulasi *Timing* untuk tiap modul dan melalui pengujian lewat *XESS Board XS40-010XL* untuk pengujian secara keseluruhan. Pengujian dilakukan terhadap modul Akuisisi Data, modul *Display*, modul *Driver* Keluaran dan Modul *Fuzzy Controller*.

Sedangkan pengujian untuk rancangan perangkat keras luar dilakukan tiap modul, yaitu modul Suhu Udara, modul Kelembaban Tanah, modul ADC (Analog to Digital Converter) dan modul *Driver* Keluaran. Selain itu, untuk perangkat keras luar ini juga perlu dilakukan kalibrasi dan pengukuran untuk mendapatkan hasil yang akurat.

4.1 PENGUJIAN ALAT

Pengujian alat secara keseluruhan dilakukan dengan memberikan masukan yang berbeda-beda. Untuk sensor suhu udara dilakukan dengan memanasi

maupun dengan mendinginkan sensor suhu udara LM34, sedangkan untuk sensor kelembaban tanah dilakukan dengan memberi air pada bidang tanah yang diukur atau memindahkannya pada bidang tanah yang lebih kering. Kemudian hasil outputnya dapat diamati pada LCD dan motor. Hasil pengujian secara keseluruhan ini dapat dilihat pada Lampiran D-1.

4.1.1 RANCANGAN PERANGKAT KERAS LUAR

Pengujian perangkat keras luar dilakukan dengan memanasi atau mendinginkan sensor suhu udara LM34 serta memberi air pada bidang tanah tempat sensor kelembaban tanah ditempatkan. Keluaran yang diamati adalah Keluaran pada sensor suhu udara dan sensor kelembaban tanah, serta Keluaran pada kedua buah rangkaian pengkondisi sinyal. Hasil pengujian untuk kedua modul tersebut dapat dilihat pada tabel 4-1 dan 4-2.

4.1.2 RANCANGAN FPGA XC4010XL

Pengujian secara keseluruhan modul FPGA yang ada pada IC FPGA XC4010XL dilakukan dengan mengamati keluaran *XESS Board XS40-010XL*, yaitu dengan mengamati tampilan pada LCD dan *seven segment* pada *XESS Board*. Sedangkan masukan dari *XESS Board* ini berasal dari ADC MAX114 yang kedua kanal masukannya dihubungkan dengan potentiometer *multiturn* untuk mengubah-ubah harga tegangan masukannya. Hasil pengujian ini dapat dilihat pada Lampiran D-2. Selain itu, kami juga mengadakan pengujian menggunakan

Simulasi *Timing* dari Xilinx Foundation 1.5. Hasil simulasi *Timing* tersebut dapat dilihat pada Lampiran D-3.

4.1.2.1 Modul Akuisisi Data

Pengujian modul akuisisi data ini dapat dilihat hasil simulasi *Timing* pada Lampiran D-4. Pada pengujian tersebut dilakukan perubahan data sebanyak 3 kali untuk membuktikan proses akuisisi data yang terjadi.

4.1.2.2 Modul Display

Pengujian modul *display* ini dapat dilihat hasil simulasi *Timing* pada Lampiran D-5. Pada pengujian tersebut dilakukan perubahan data sebanyak 3 kali untuk membuktikan kebenaran modul *display tersebut*.

4.1.2.3 Modul Fuzzy Kontroler

Pengujian modul *Fuzzy Controller* data ini dapat dilihat hasil simulasi *Timing* pada Lampiran D-6. Pada pengujian tersebut dilakukan perubahan data sebanyak 3 kali untuk membuktikan proses *fuzzy* yang terjadi.

4.1.2.4 Modul Driver Keluaran

Pengujian modul *Driver* Keluaran ini dapat dilihat hasil simulasi *Timing* pada Lampiran D-7. Pada pengujian tersebut dilakukan perubahan data sebanyak 3 kali untuk membuktikan kebenaran keluaran dari proses *fuzzy logic*.

4.2 KALIBRASI DAN PENGUKURAN

4.2.1 MODUL SUHU UDARA

Pengukuran dan kalibrasi pada modul suhu yang meliputi sensor suhu LM34 dan rangkaian pengkondisi sinyal ini dilakukan dengan membandingkan

hasil keluaran modul ini dengan termometer referensi seperti pada tabel 4.1. Kalibrasi dilakukan pada suhu 84°F dan selanjutnya keluaran rangkaian pengkondisi sinyal diukur dengan menggunakan multimeter digital. Karena pada suhu 84°F keluaran sensor suhu LM34 tidak sesuai dengan dengan suhu yang ditunjukkan oleh termometer pada saat itu, maka perlu dilakukan kalibrasi dengan mengatur besar tegangan referensi dengan mengubah harga tahanan *multiturn* VR8 100K sehingga tampilan pada LCD menunjukkan suhu yang sama dengan nilai yang ditunjukkan pada termometer.

Tabel 4.1 Hasil Pengukuran Suhu

Nomor	Termometer (°F)	Hasil Pembacaan (°F)	Error (%)
1	80	80	0
2	82	81	1,2195
3	84	84	0
4	86	84	2,3256
5	88	87	1,1364
6	90	90	0
7	92	92	0
8	94	94	0
9	96	95	1,0417
10	98	98	0

4.2.2 MODUL KELEMBABAN TANAH

Pengukuran dan kalibrasi pada modul kelembaban tanah yang meliputi sensor kelembaban tanah SS200 dan rangkaian pengkondisi sinyal ini dilakukan dengan membandingkan hasil Keluaran modul ini dengan lembar kalibrasi yang ada pada tabel . Kemudian IC FPGA akan mengolah data terebut menjadi dalam bentuk persen dengan range antara 0 sampai dengan 200 cb yang sebanding dengan 0% sampai dengan 30%.

Tabel 4.2 Kalibrasi Sensor Kelembaban Tanah SS200

CB	OHMS	CB	OHMS	CB	OHMS	CB	OHMS	CB	OHMS	CB	OHMS
0	550	35	6000	70	11450	105	16200	140	20575	175	24950
1	600	36	6160	71	11600	106	16325	141	20700	176	25075
2	650	37	6320	72	11750	107	16450	142	20825	177	25200
3	700	38	6480	73	11900	108	16575	143	20950	178	25325
4	750	39	6640	74	12050	109	16700	144	21075	179	25450
5	800	40	6800	75	12200	110	16825	145	21200	180	25575
6	850	41	6960	76	12335	111	16950	146	21325	181	25700
7	900	42	7120	77	12470	112	17075	147	21450	182	25825
8	950	43	7280	78	12605	113	17200	148	21575	183	25950
9	1000	44	7440	79	12740	114	17325	149	21700	184	26075
10	1100	45	7600	80	12875	115	17450	150	21825	185	26200
11	1280	46	7760	81	13010	116	17575	151	21950	186	26325
12	1460	47	7920	82	13145	117	17700	152	22075	187	26450
13	1640	48	8080	83	13280	118	17825	153	22200	188	26575
14	1820	49	8240	84	13415	119	17950	154	22325	189	26700
15	2000	50	8400	85	13550	120	18075	155	22450	190	26825
16	2200	51	8560	86	13685	121	18200	156	22575	191	26950
17	2400	52	8720	87	13820	122	18325	157	22700	192	27075
18	2600	53	8880	88	13955	123	18450	158	22825	193	27200
19	2800	54	9040	89	14090	124	18575	159	22950	194	27325
20	3000	55	9200	90	14225	125	18700	160	23075	195	27450
21	3200	56	9350	91	14360	126	18825	161	23200	196	27575
22	3400	57	9500	92	14495	127	18950	162	23325	197	27700
23	3600	58	9650	93	14630	128	19075	163	23450	198	27825
24	3800	59	9800	94	14765	129	19200	164	23575	199	27950
25	4000	60	9950	95	14900	130	19325	165	23700		
26	4200	61	10100	96	15035	131	19450	166	23825		
27	4400	62	10250	97	15170	132	19575	167	23950		
28	4600	63	10400	98	15305	133	19700	168	24075		
29	4800	64	10550	99	15440	134	19825	169	24200		
30	5000	65	10700	100	15575	135	19950	170	24325		
31	5200	66	10850	101	15700	136	20075	171	24450		
32	5400	67	11000	102	15825	137	20200	172	24575		
33	5600	68	11150	103	15950	138	20325	173	24700		
34	5800	69	11300	104	16075	139	20450	174	24825		

BAB V
PENUTUP

BAB V

PENUTUP

Dari hasil perancangan dan pengujian alat Tugas Akhir APLIKASI FPGA SEBAGAI PENDETEKSI PARAMETER TANAH, maka penulis mendapatkan kesimpulan dan saran yang berguna bagi pengembangan Tugas Akhir ini.

5.1 KESIMPULAN

1. Dengan adanya alat ini maka sistem pendeteksian parameter tanah yang meliputi suhu udara dan kelembaban tanah dapat dipantau dari jarak jauh sehingga memudahkan pengawasan dan pengendalian terhadap sistem irigasi suatu bidang tanah, yang pada akhirnya akan menyebabkan tanaman tumbuh dengan baik.
2. Pemakaian IC LM34 sebagai sensor suhu udara dan sensor kelembaban tanah SS200 menyebabkan kita dapat memperoleh data-data parameter tanah dengan akurat dan cepat.
3. Pemanfaatan IC FPGA XC4010XL sebagai kontroler yang berbasis logika samar *(fuzzy logic)* dan pendeteksi parameter tanah merupakan hal yang efisien, karena semua rancangan kontroler dan pendeteksi parameter tersebut berada dalam satu IC FPGA XC4010XL.

4. Metode pemrograman VHDL pada *Xilinx Foundation Series 1.5* memungkinkan kita untuk mendapatkan perancangan yang lebih sederhana dan menghemat CLB (*Configurable Logic Black*), karena proses sintesa dan implementasinya sesuai dengan arsitektur IC FPGA XC4010XL.
5. Tugas Akhir APLIKASI FPGA SEBAGAI PENDETEKSI PARAMETER TANAH DENGAN METODE FUZZY LOGIC terealisasi dengan menggunakan *XESS Board XS40-010XL* dengan memanfaatkan 279 CLB dari total 400 CLB, 24 pin dari total 63 pin I/O yang dimiliki oleh IC FPGA XC4010XL.
6. Penggunaan IC ADC MAX114 sebagai komponen akuisisi data merupakan hal yang menguntungkan dan sesuai dengan kebutuhan perancangan Tugas Akhir ini, karena selain memiliki kecepatan sampling yang cepat, juga mampu dioperasikan dengan menggunakan sumber tegangan tunggal 5 Volt.
7. Kesalahan yang terjadi pada pengukuran kelembaban tanah maupun suhu udara sangat dipengaruhi oleh faktor lingkungan yaitu faktor kontak antara tanah dengan sensor kelembaban tanah, dan kepekaan sensor LM34 itu sendiri.

5.2 SARAN

1. Penambahan parameter-parameter tanah yang dideteksi, sehingga proses irigasi dapat berjalan lebih akurat untuk mendukung proses pertumbuhan tanah dengan baik.

2. Perlunya dibuat suatu modul FPGA yang mampu men-*download*-kan parameter-parameter maupun rule dari *Fuzzy Controller* tersebut secara otomatis.
3. Perlu pula mempelajari dan mengembangkan fasilitas *JTAG* yang terdapat pada IC FPGA XC4010XL yang memungkinkan pemrograman dan pengetesan IC hanya dengan menggunakan 4 pin saja.
4. Perlu adanya peningkatan fasilitas pada keluaran *Fuzzy Controller* untuk mengaktifkan sistem irigasi, misalnya menambahkan keluaran sudut membukanya katup irigasi.
5. Perlunya peningkatan pemanfaatan IC FPGA XC4010XL untuk mendukung perkembangan teknologi di bidang pertanian.

DAFTAR PUSTAKA

DAFTAR PUSTAKA

1. -----, **APPLINX CD,** Tenth Revision Second Quarter, Xilinx, 1999.
2. -----, **CORE SOLUTIONS DATA BOOK 1999,** Xilinx,1999.
3. Boylestad, Robert dan Nashelsky Louis, **ELECTRONIC DEVICE AND CIRCUIT THORY**, Fifth Edition, Prentice Hall International, 1992.
4. -----, **FPGA EXPRESS VHDL REFERENCE MANUAL,** Synopsys, 1997.
5. -----, **FULL LINE DATA CATALOG CD**, Maxim Semiconductor, 1999.
6. Jamshidi Mohammad, Vadiee, Naderm, J.Ross Timothym **FUZZY LOGIC AND CONTROL SOFTWARE AND HARDWARE APPLICATIONS,** Prentice Hall International, USA, 1993.
7. -----. **FUZZY LOGIC EDUCATIONAL PROGRAM,** Motorolla, 1992-1994.
8. -----, **LCD MODULE USER MANUAL**, Seiko Instruments Inc,1987.
9. -----, **NATIONAL DATA ACQUISITION DATABOOK**, National Semiconductor Corporation, 1995.
10. -----, **NATIONAL POWER ICS DATABOOK**, National Semiconductor Corporation, 1995.
11. -----, **NATIONAL OPERATIONAL AMPLIFIER DATABOOK**, National Semiconductor Corporation, 1995.

12. -----, **NLX221, NLX221P, NLX222, NLX222P STAND ALONE FUZZY LOGIC CONTROLLERS PRELIMINARY DATA,** Adaptive Logic, 1994.

13. Coughlin, Robert F dan Driscoll, Frederick F, **OPERATIONAL AMPLIFIER AND LINEAR IC**, Second Edition, Prentice Hall, 1982.

14. -----, **SYNOPSYS FPGA EXPRESS ONLINE HELP SYSTEM,** Synopsys, 1999.

15. -----, **THE PROGRAMMABLE LOGIC DATA BOOK 1999,** Xilinx, 1999.

16. Van den Bout, Dave, **THE PRACTICAL XILINX DESIGNER LAB BOOK 1.5,** Prentice-Hall International, New Jersey, USA,1999.

17. Skahlil Kevin, **VHDL FOR PROGRAMMABLE LOGIC,** Addison-Wesley Publishing Company, Inc, California, 1996.

18. -----, **WATERMARK TECHNICAL SPECS**, Irrometer CO, 1999.

19. -----, **XILINX FOUNDATION SERIES ONLINE HELP SYSTEM,** Xilinx, 1999.

20. -----, **XS40, XSP, AND XS95 BOARD MANUAL,** XESS corp, 1998.

DAFTAR KATA

DAFTAR KATA

Analog to Digital Converter (ADC)

Modul yang berfungsi untuk mengubah sinyal analog menjadi sinyal digital.

Centi Bar (CB)

Satuan besar tegangan yang ditimbulkan oleh sensor kelembaban tanah yang sebanding dengan kandungan air dalam tanah. Makin besar air di dalam tanah, makin kecil tegangan yang ditimbulkan, sebaliknya makin kecil jumlah air dalam tanah, makin besar tegangan yang ditimbulkan.

Configurable Logic Block (CLB)

Bagian dari IC FPGA yang berfungsi untuk mengimlplementasikan fungsi-fungsi logika.

Core Generator

Fasilitas pada *software Xilinx Foundation Series* untuk membentuk suatu komponen.

Defuzzifier

Bagian dari proses *fuzzy logic* yang berfungsi untuk mendapatkan keluaran digital dari keluaran *rule evaluator.*

Falling Edge

Keadaan transisi dari level logika '1' menjadi level logika '0'.

Fast Carry Logic

Fasilitas pada IC FPGA yang berfungsi untuk mengimplementasikan fungsi aritmetika dan fungsi logika.

Field Programmable Gate Array (FPGA)

IC *Programmable Logic* yang memiliki arsitektur seperti susunan matriks sel-sel logika yang dapat berhubungan satu sama lain melalui jalur-jalur I/O.

Flip-Flop

Elemen penyimpan yang perubahan outputnya terjadi pada saat perubahan sinyal clock.

Function Generator

Bagian dari CLB yang dapat digunakan sebagai *memory look up table*, ROM dan RAM.

Fuzzifier

Bagian dari proses *fuzzy logic* yang berfungsi untuk mengubah tegangan input digital menjadi nilai derajat *fuzzy*.

Fuzzy Logic (Logika samar)

Sistem logika yang perubahannya dilakukan secara bertahap dari logika '1' ke logika '0' atau sebaliknya sehingga didapat perubahan keluaran yang halus.

Global Set/Reset

Fasilitas pada IC FPGA yang berfungsi untuk men-*set* atau me-*reset* semua elemen penyimpan pada IC FPGA

Input Output Block (IOB)

Bagian dari IC FPGA yang berfungsi sebagai penghubung antara CLB dengan rangkaian di luar IC FPGA.

Latch

Elemen penyimpan yang perubahan outputnya tidak bergantung pada perubahan sinyal clock.

Logiblox

Fasilitas pada *software Xilinx Foundation Series* untuk membentuk suatu komponen.

Membership Function

Pembagian range suatu input proses fuzzifikasi menjadi beberapa *label*.

On-chip Oscillator

Fasilitas dari IC FPGA yang berfungsi menghasilkan sinyal clock dengan frekuensi-frekuensi tertentu.

Program Storage

Bagian dari proses *fuzzy logic* yang berfungsi untuk menyimpan *rule-rule* yang dibutuhkan dalam proses dan menyimpan parameter-parameter *membership function* pada proses fuzzifikasi.

Pulse Generator

Pembangkit sinyal pulsa dengan periode tertentu.

Pulse Width Modulation

Sinyal pulsa yang memiliki lebar pulsa berlainan, sesuai dengan amplitudo dari suatu sinyal.

Ramp Generator

Sinyal ramp (menanjak) sebagai pembanding sinyal keluaran linier pada proses pembentukan sinyal PWM.

Rising Edge

Keadaan transisi dari level logika '0' menjadi level logika '1'.

Rule Evaluator

Bagian dari proses *fuzzy logic* yang berfungsi untuk mendapatkan pemenang dari suatu kelompok *rule.*

Sample and Hold

IC yang berfungsi untuk menyimpan tegangan input analog selama tidak ada masukan sinyal pulsa. Perubahan keluaran terjadi apabila terdapat masukan sinyal pulsa.

Slew Rate

Waktu yang dibutuhkan untuk melakukan perubahan output.

Three State Buffer (TBUF)

Fasilitas pada IC FPGA yang berfungsi sebagai pengatur jalur-jalur horizontal yang berhubungan dengan CLB.

Wide Edge Decoder

Fasilitas pada IC FPGA yang berfungsi sebagai dekoder alamat yang memiliki banyak bit.

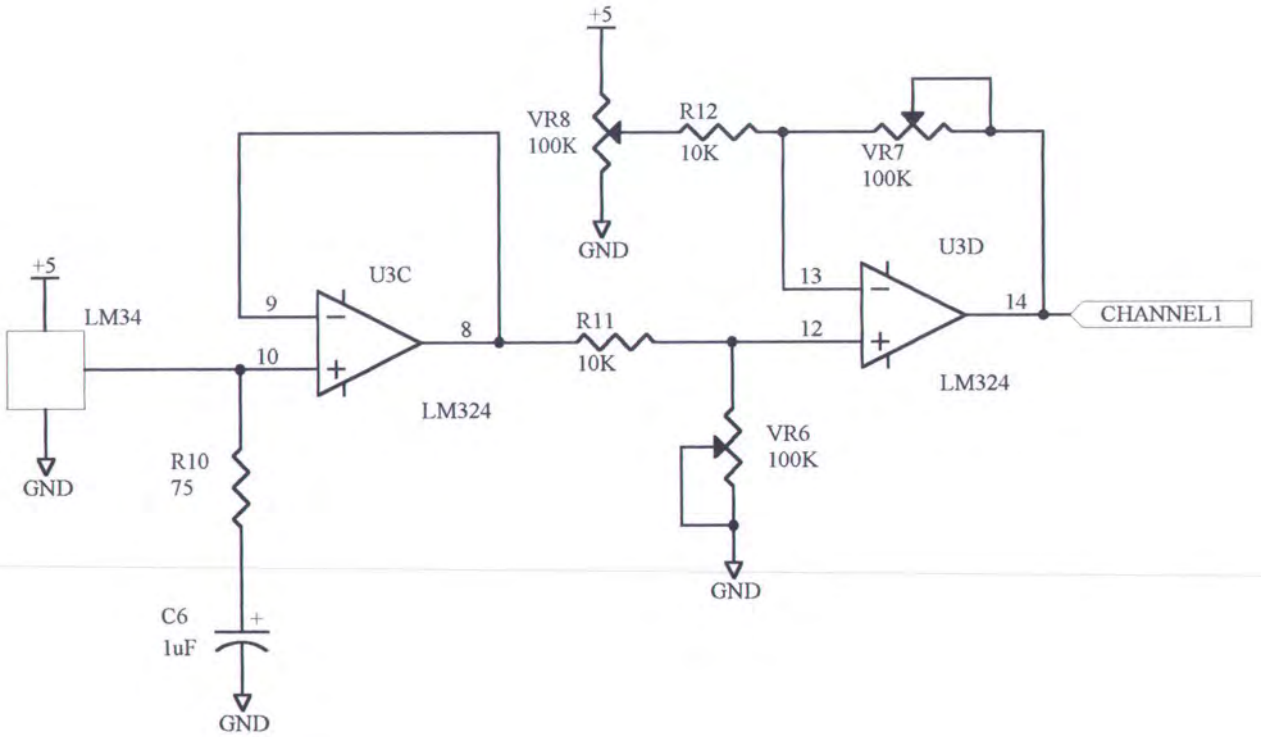
XESS Board XS40-010XL

Modul yang berfungsi untuk mengimplementasikan rancangan FPGA, yang terdiri dari komponen utama IC FPGA XC4010XL dan mikrokontroler 8051.

LAMPIRAN

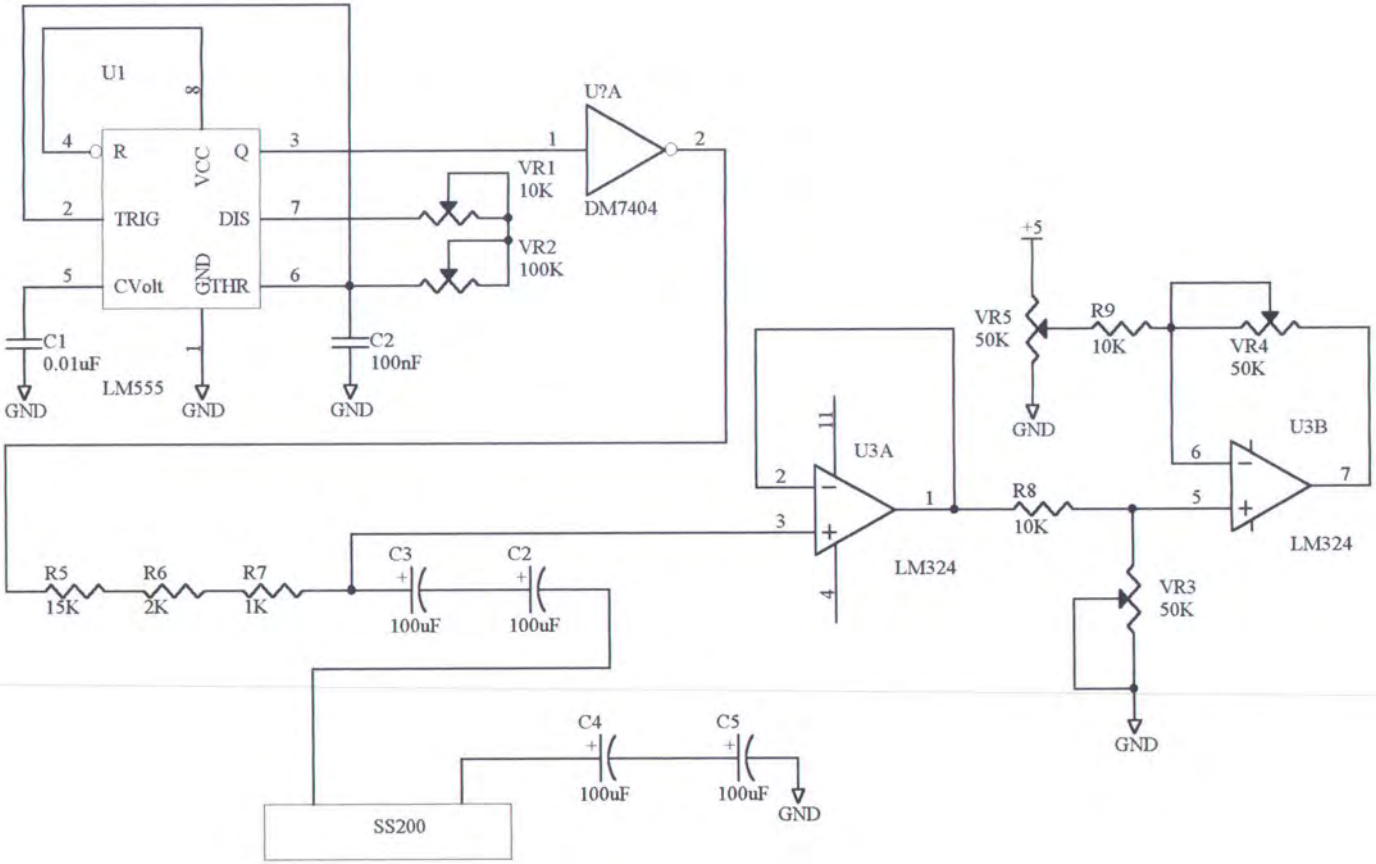
LAMPIRAN A-1

RANGKAIAN MODUL SUHU UDARA



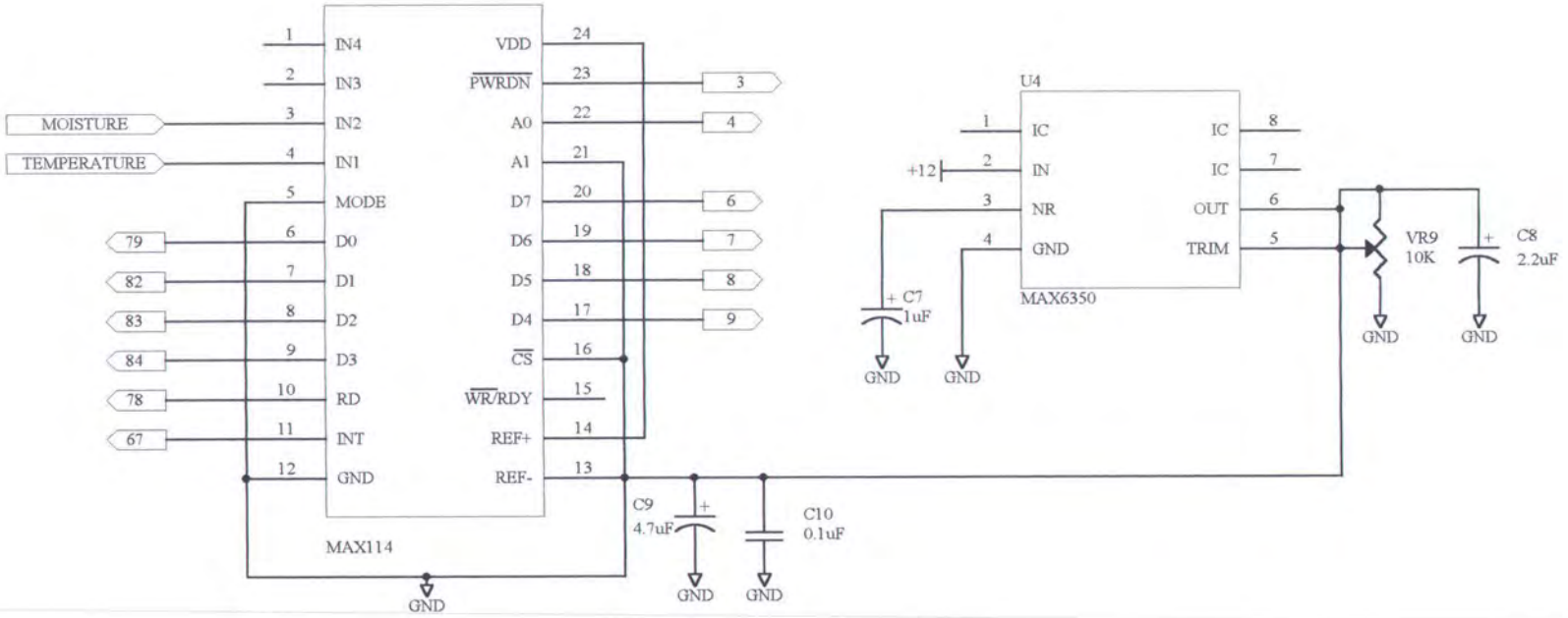
LAMPIRAN A-2

RANGKAIAN MODUL KELEMBABAN TANAH



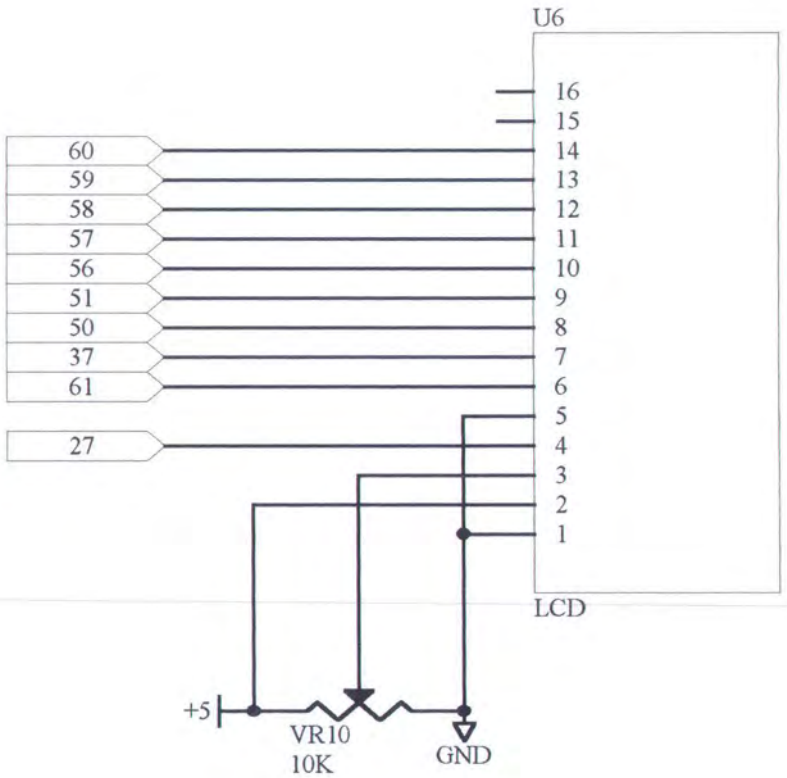
LAMPIRAN A-3

RANGKAIAN MODUL ANALOG TO DIGITAL CONVERTER



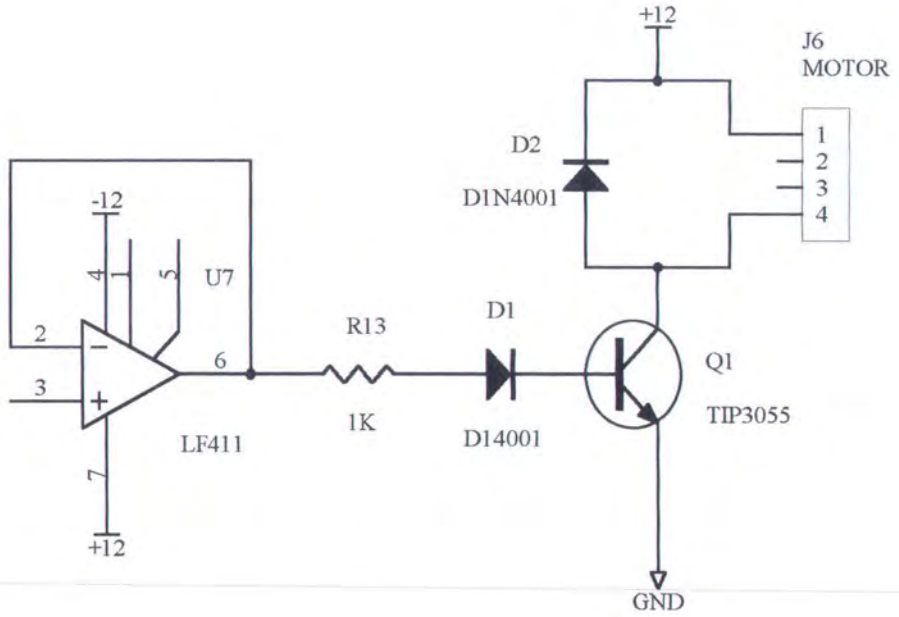
LAMPIRAN A-4

RANGKAIAN MODUL LIQUID CRYSTAL DISPLAY



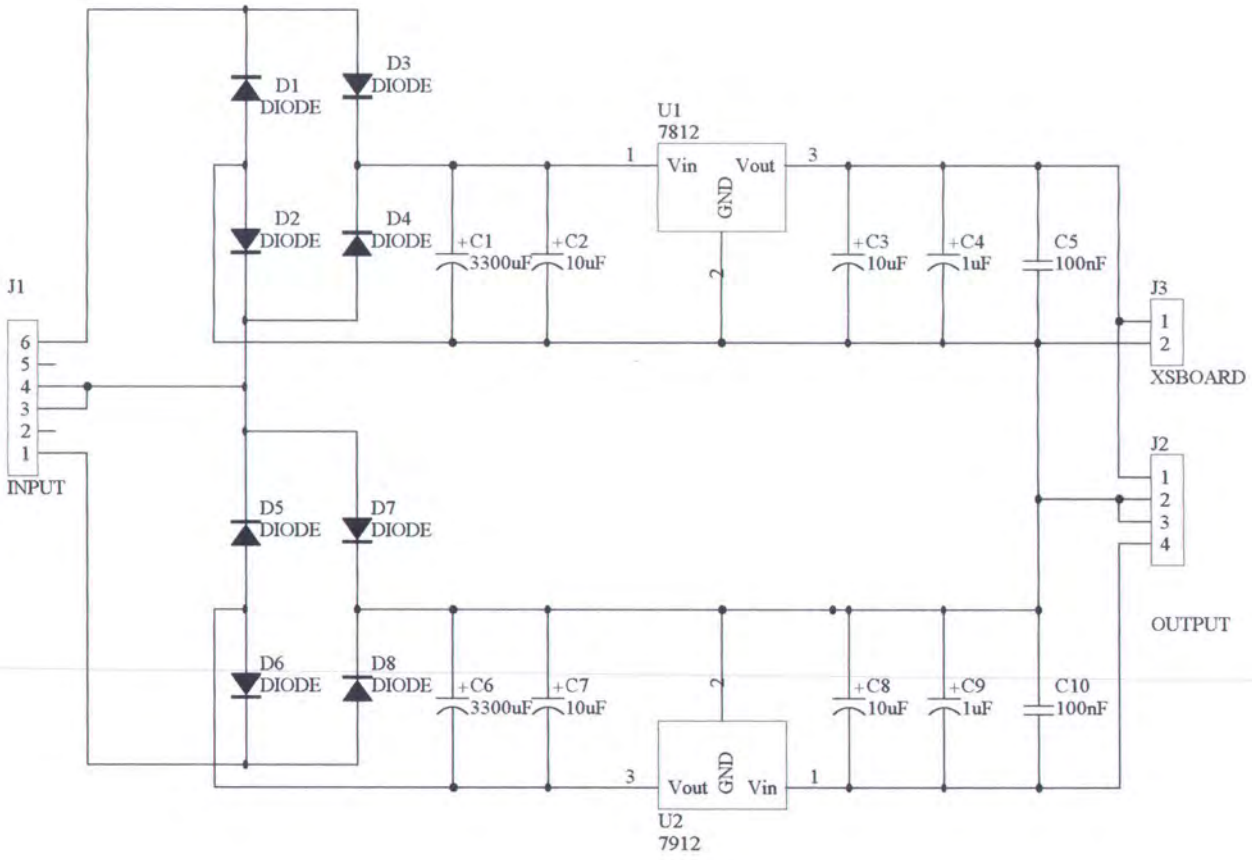
LAMPIRAN A-5

RANGKAIAN MODUL DRIVER OUTPUT



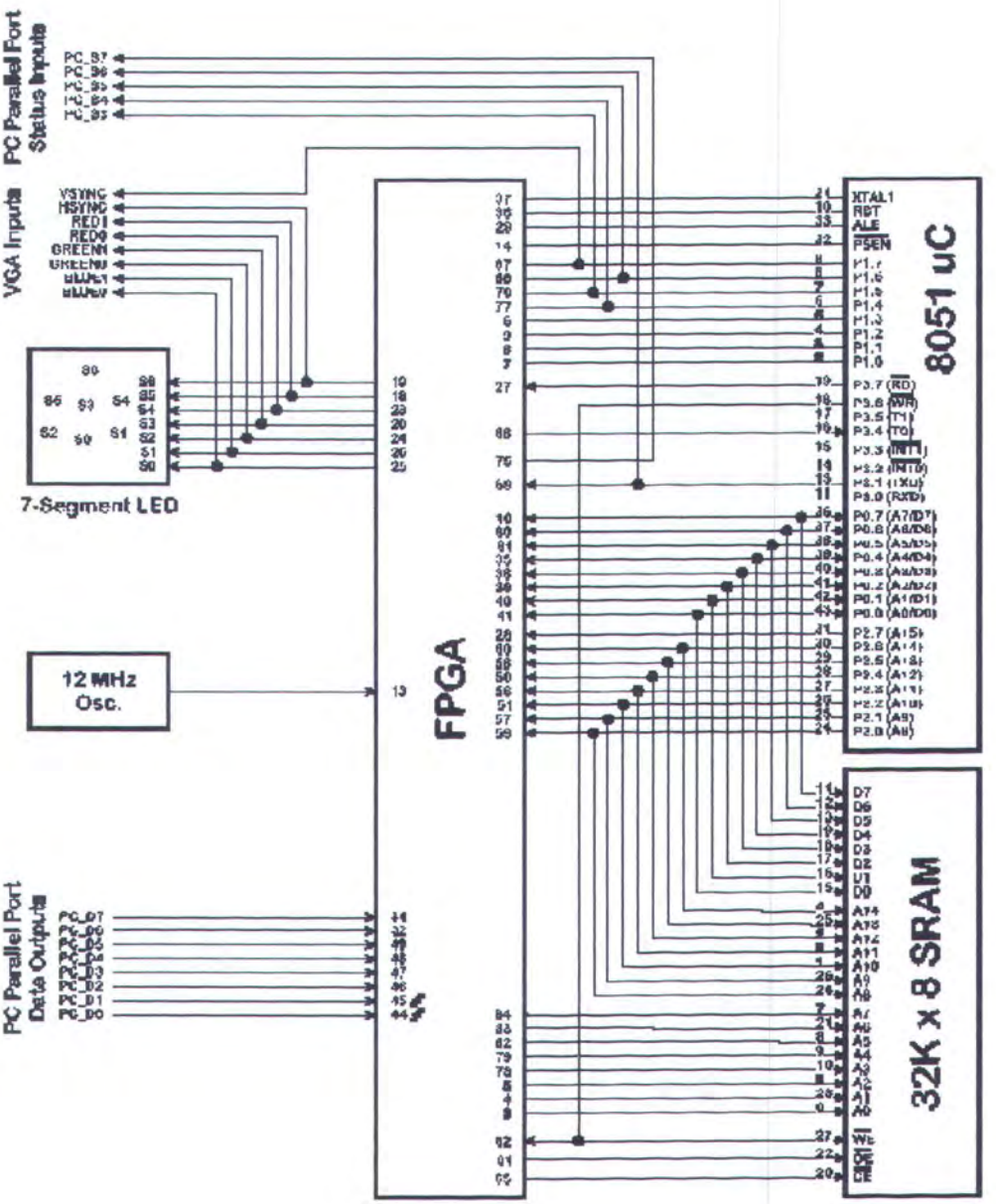
LAMPIRAN A-6

RANGKAIAN MODUL CATU DAYA



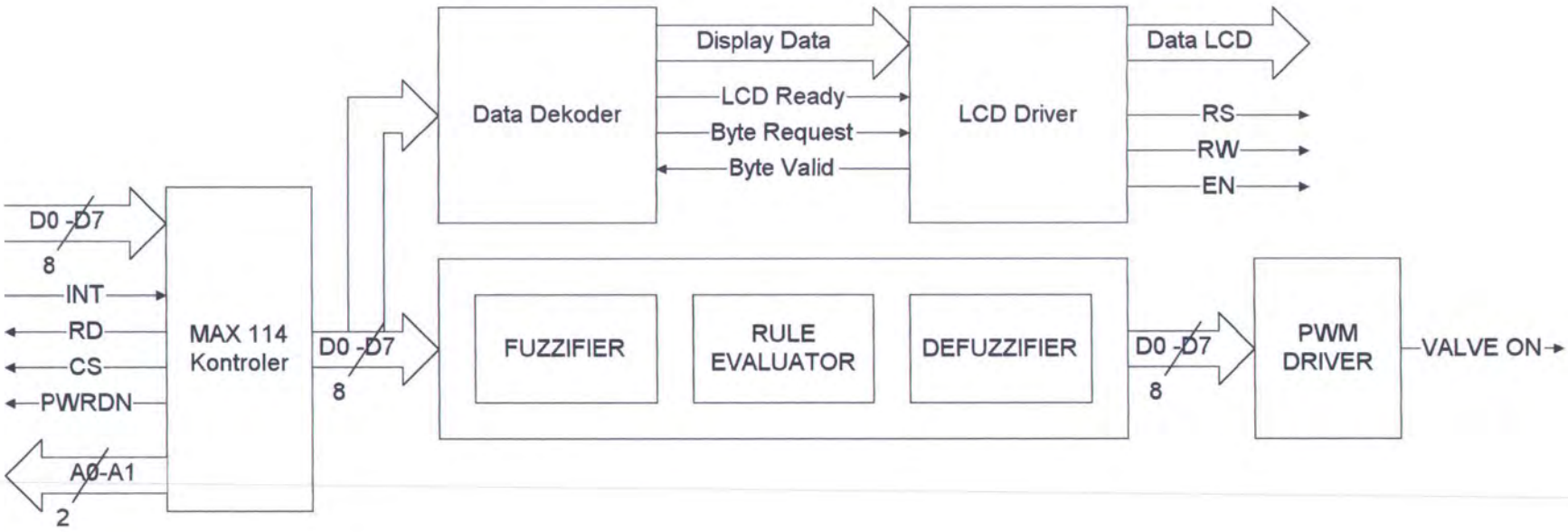
LAMPIRAN A-7

RANGKAIAN MODUL XESS BOARD XS40-010XL



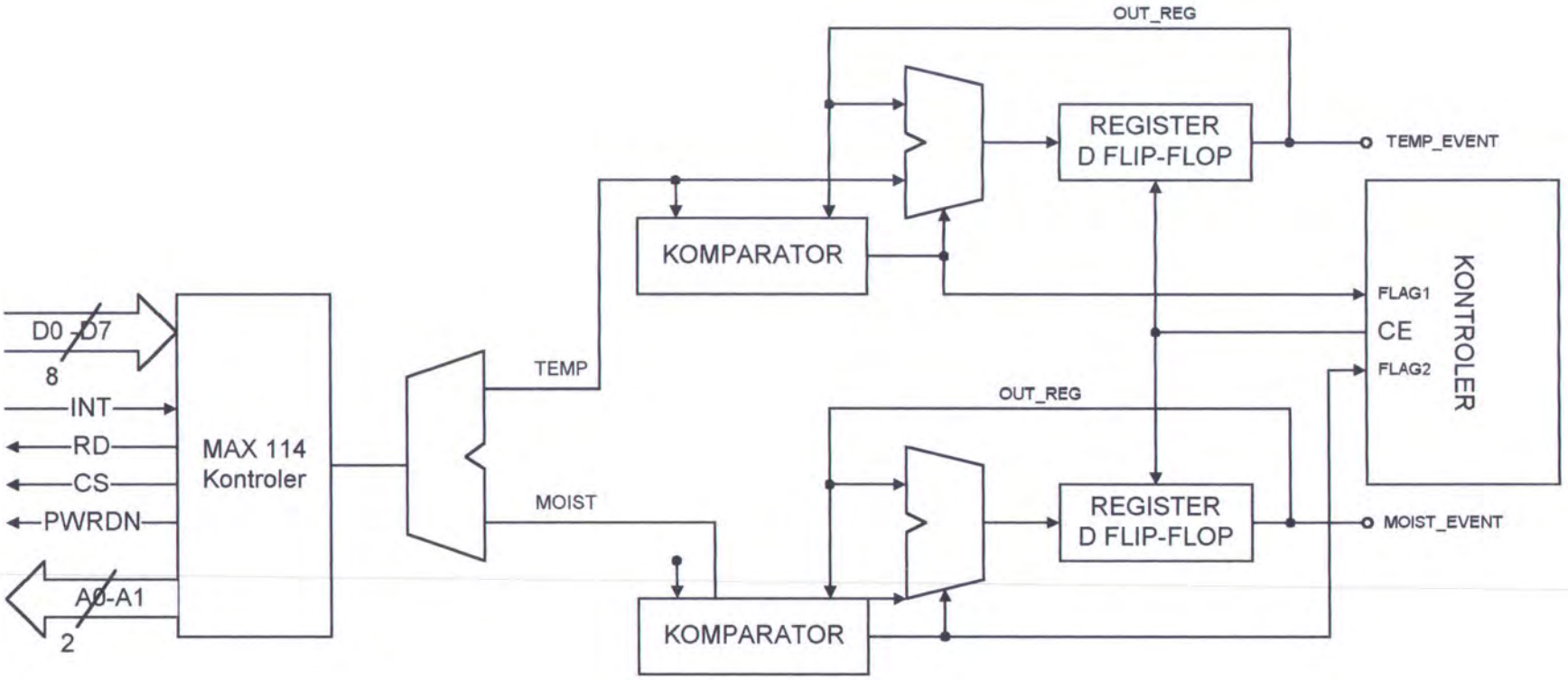
LAMPIRAN B-1

BLOK DIAGRAM PERENCANAAN MODUL FPGA



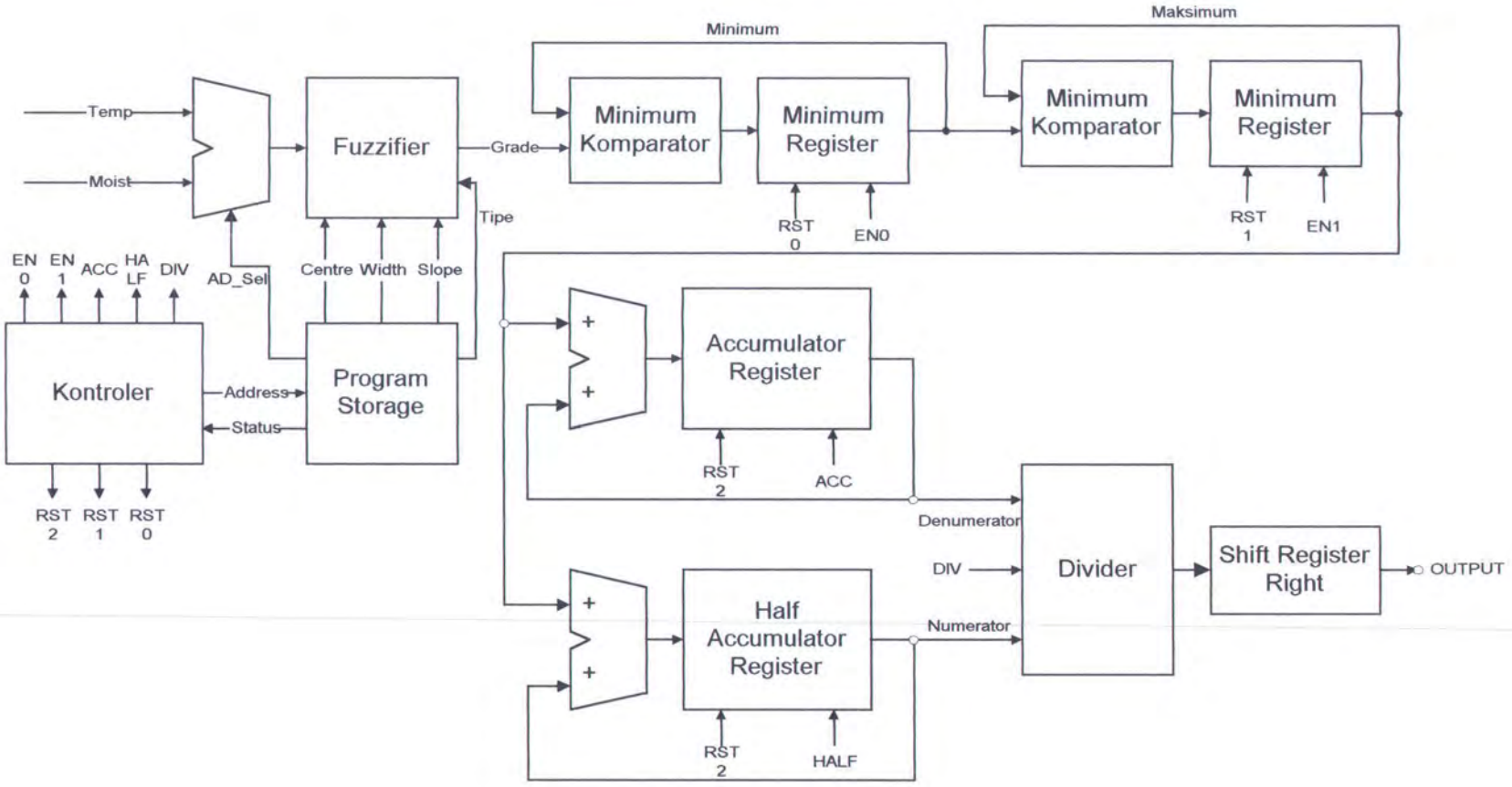
LAMPIRAN B-2

BLOK DIAGRAM PERENCANAAN MODUL AKUISISI DATA



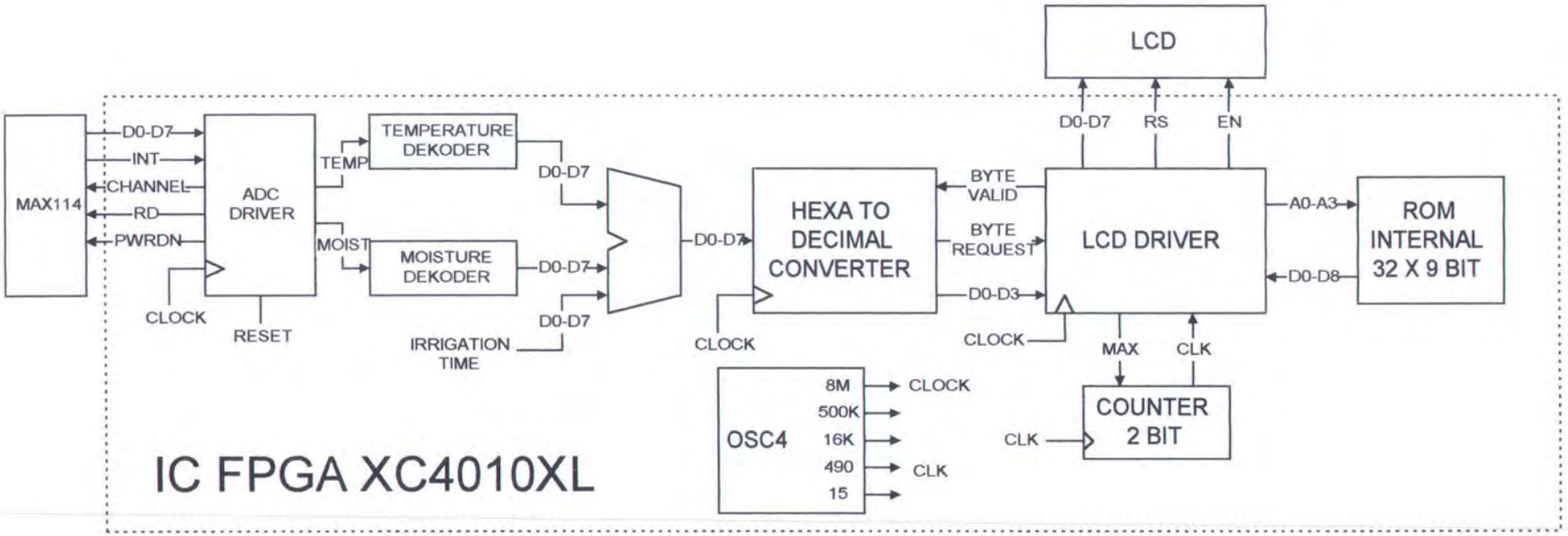
LAMPIRAN B-3

BLOK DIAGRAM PERENCANAAN MODUL FUZZY KONTROLER



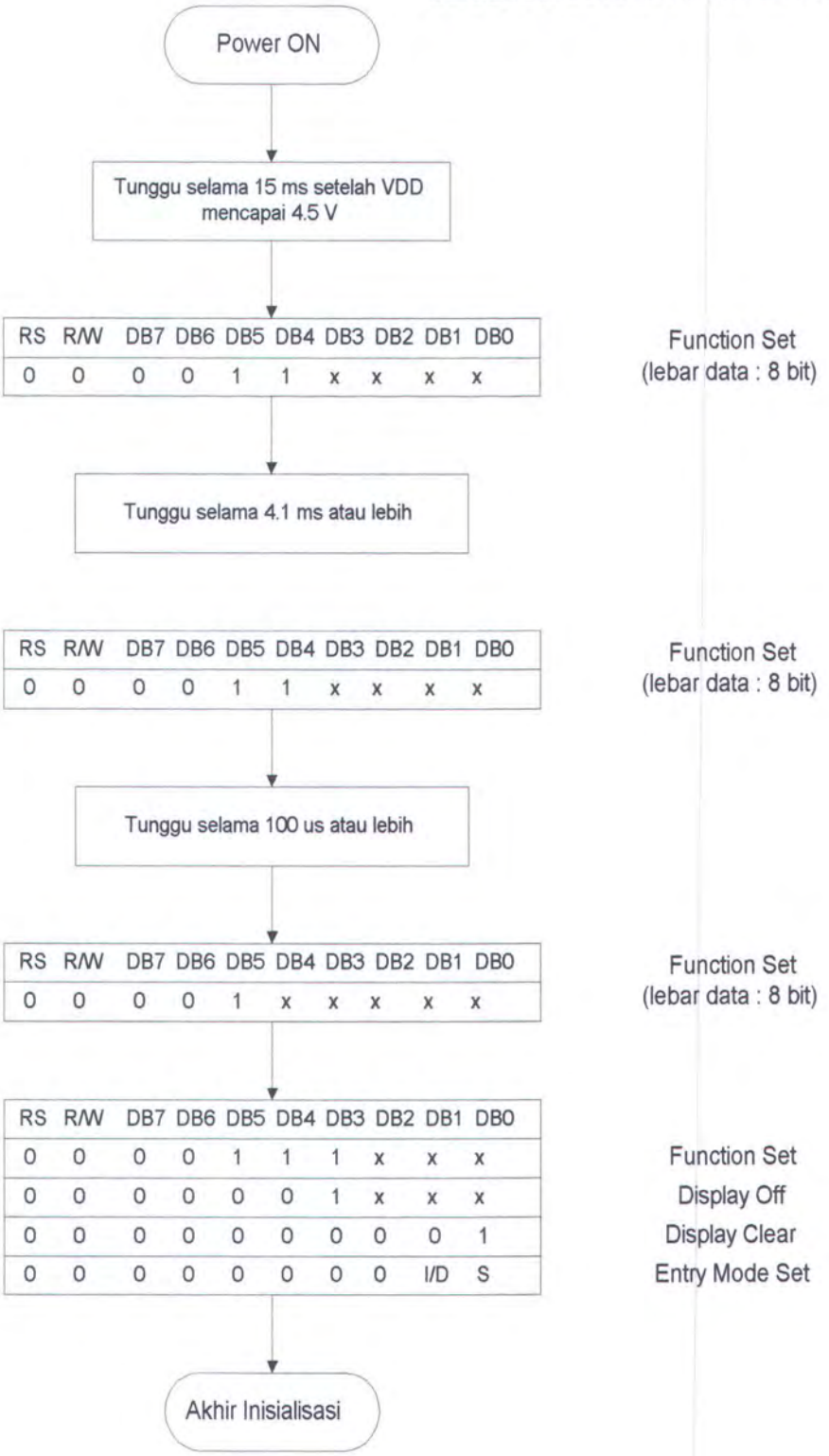
LAMPIRAN B-4

BLOK DIAGRAM PERENCANAAN MODUL DISPLAY



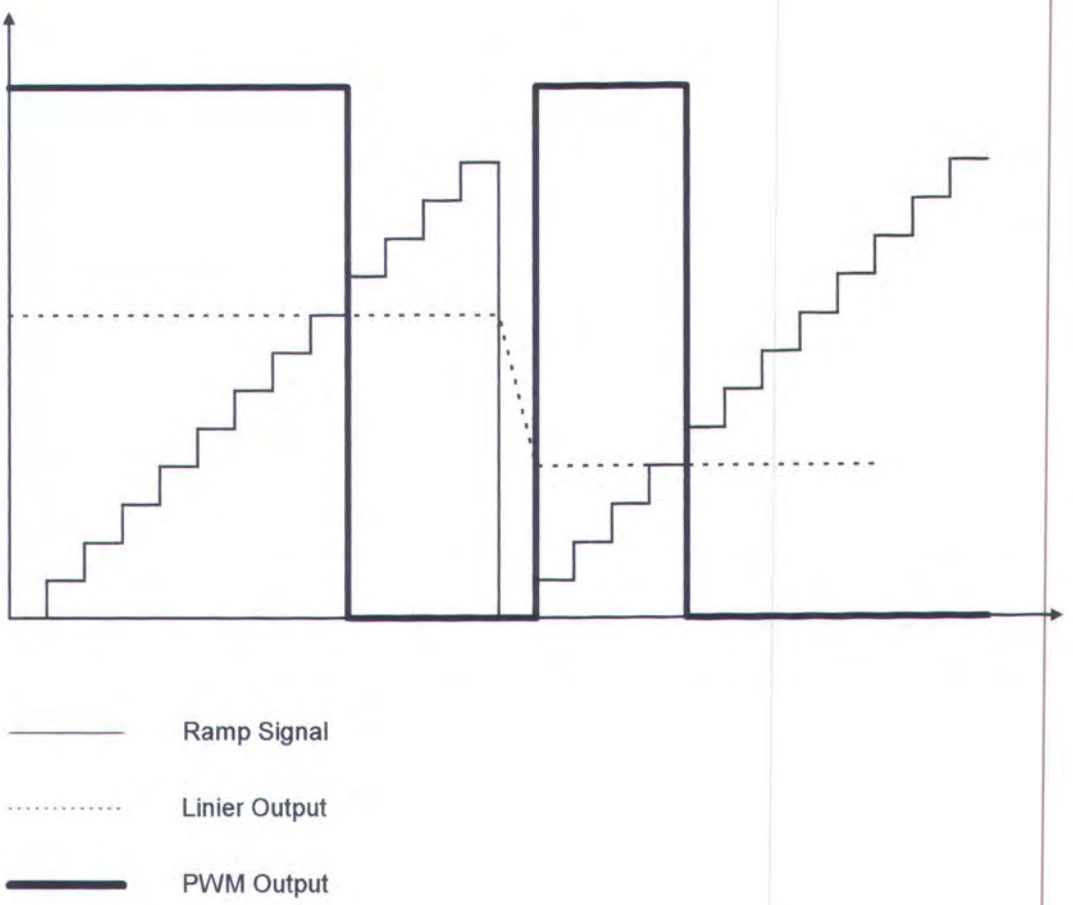
LAMPIRAN B-5

DIAGRAM ALIR INISIALISASI LCD



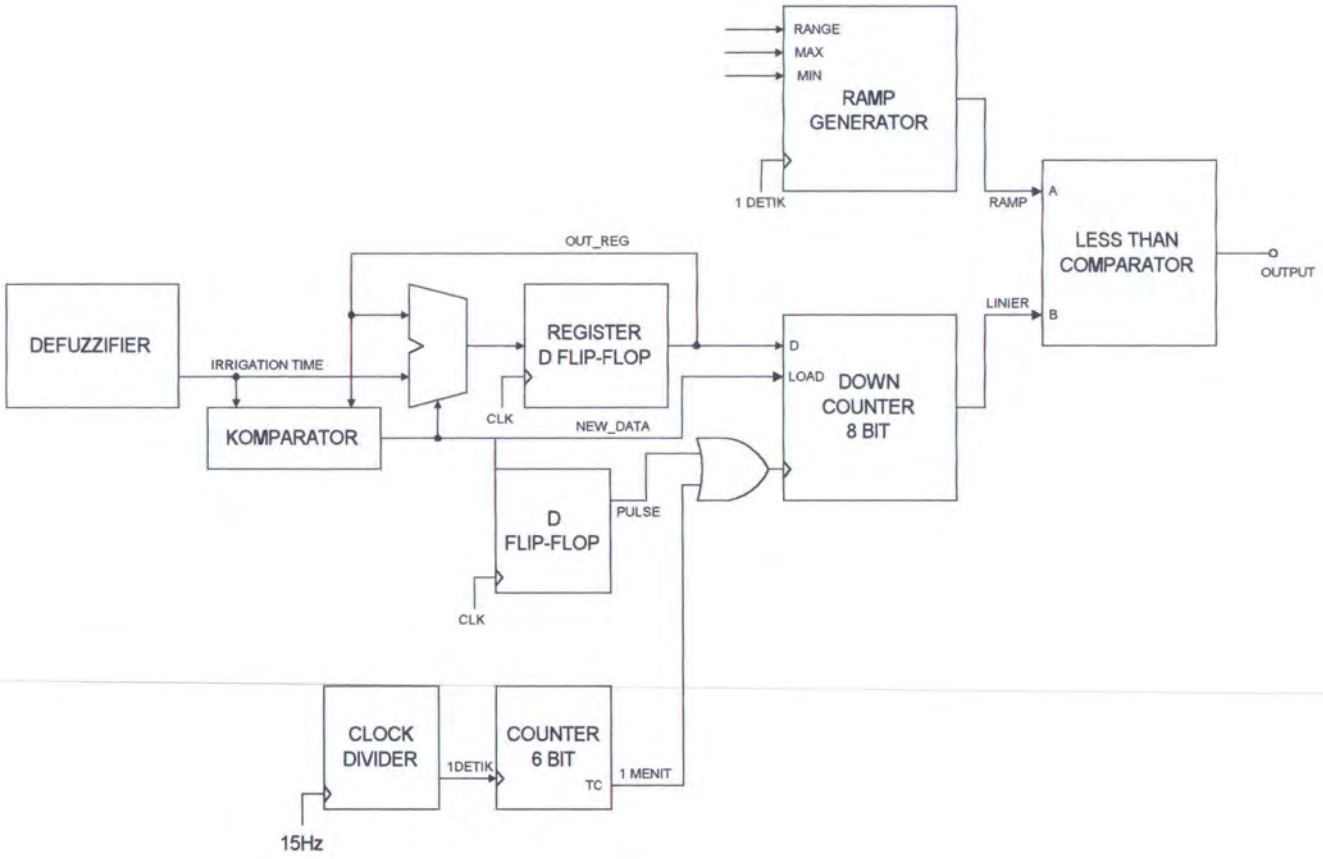
LAMPIRAN B-6

PROSES PEMBENTUKAN SINYAL PWM



LAMPIRAN B-7

BLOK DIAGRAM PERENCANAAN MODUL DRIVER OUTPUT



LAMPIRAN C-1

LISTING PROGRAM VHDL MODUL ADC DRIVER

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;

Entity MAX114 Is
Port(Reset,Clock : In  Std_Logic;
     Input       : In  Std_Logic_Vector(7 downto 0);
     INT         : In  Std_Logic;
     Channel     : Buffer Std_Logic;
     RD,PWRDN    : Out Std_Logic;
     Temp,Moist  : Out Std_Logic_Vector(7 downto 0));
End Entity MAX114;

Architecture ARCH_MAX114 Of MAX114 Is
Signal AD_Mux  : Std_Logic;

Begin
   Process(Reset,Clock)
   Begin
   IF (Reset='1') Then
      RD      <= '1';
      PWRDN   <= '0';
      Channel <= '0';
   ELSIF (Clock'Event And Clock='1') Then
      IF (INT='0') Then
      IF (AD_Mux='0') Then
         Temp    <= Input;
         Channel <= '1';
      ELSE
         Moist   <= Input;
         Channel <= '0';
      END IF;
      RD    <= '1';
       ELSE
               PWRDN <= '1';
               RD    <= '0';
               IF (Channel='1') Then
                  AD_Mux <= '1';
               ELSE
                  AD_Mux <= '0';
               END IF;
       END IF;
   END IF;
   END Process;
End ARCH_MAX114;
```

LAMPIRAN C-2

LISTING PROGRAM VHDL MODUL EVENT

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;

Entity EVENT Is
Port(Rst,Clk,EN : In  Std_Logic;
     Input      : In  Std_Logic_Vector(7 downto 0);
     Flag       : Out Std_Logic;
     Output     : Out Std_Logic_Vector(7 downto 0));
End Entity EVENT;

Architecture Arch_Event Of EVENT Is

Signal Out_Mux,Out_Reg : Std_Logic_Vector(7 downto 0);
Signal Sel             : Std_Logic;

Component COMPARE
Port(A: IN std_logic_vector(7 DOWNTO 0);
     B: IN std_logic_vector(7 DOWNTO 0);
     A_NE_B: OUT std_logic);
End Component COMPARE;

Component MUX2TO1
Port(S : IN  std_logic;
     O : OUT std_logic_vector(7 DOWNTO 0);
     MA: IN  std_logic_vector(7 DOWNTO 0);
     MB: IN  std_logic_vector(7 DOWNTO 0));
End Component MUX2TO1;

Component DATA_REG
Port(D_IN    : IN  std_logic_vector(7 DOWNTO 0);
     ASYNC_CTRL : IN  std_logic;
     CLK_EN : IN  std_logic;
     CLOCK  : IN  std_logic;
     Q_OUT  : OUT std_logic_vector(7 DOWNTO 0));
End Component DATA_REG;

Begin
   U1: COMPARE Port Map(A => Input,
                        B => Out_Reg,
                        A_NE_B => Sel);

   U2: MUX2TO1 Port Map(S  => Sel,
                        O  => Out_Mux,
                        MA => Out_Reg,
                        MB => Input);

   U3: DATA_REG Port Map(D_IN          => Out_Mux,
                         ASYNC_CTRL => Rst,
```

```
                        CLK_EN      => EN,
                        CLOCK       => Clk,
                        Q_OUT       => Out_Reg);
        Flag   <= Sel;
        Output <= Out_Reg;
End Arch_Event;
```

LAMPIRAN C-3

LISTING PROGRAM VHDL MODUL FUZZIFIER

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;
Use IEEE.Std_Logic_Arith.All;

Entity Fuzzifier Is
Port(Input  : In  Std_Logic_Vector(7 downto 0);
     Centre : In  Std_Logic_Vector(7 downto 0);
     Tipe   : In  Std_Logic_Vector(1 downto 0);
     Slope  : In  Std_Logic_Vector(2 downto 0);
     Width  : In  Std_Logic_Vector(5 downto 0);
     Output : Out Std_Logic_Vector(7 downto 0));
End Entity Fuzzifier;

Architecture Arch_Fuzzifier Of Fuzzifier Is
Begin
   Process(Input,Centre,Tipe,Slope,Width)
      Variable Delta : Std_Logic_Vector(7 downto 0);
      Variable Tanda : Std_Logic;
      Constant FF    : Unsigned(7 downto 0) := (Others => '1');
      Begin
         IF (Input<Centre) Then Tanda := '1';
         ELSE Tanda := '0';
         END IF;
         IF    (Tipe="00" And Tanda='1') Then Output <= (Others => '1');
         ELSIF (Tipe="10" And Tanda='0') Then Output <= (Others => '1');
         ELSE
            Delta:=Conv_Std_Logic_Vector(Abs(Signed(Centre)-Signed(Input)),8);
            IF (Unsigned(Delta)<Unsigned(Width)) Then
```

```
                Output <= Conv_Std_Logic_Vector(FF-Unsigned(Delta)*Unsigned(Slope)),8);
             ELSE
                Output <= (Others => '0');
             END IF;
          END IF;
       End Process;
End Arch_Fuzzifier;
```

LAMPIRAN C-4

LISTING PROGRAM VHDL MODUL RULE EVALUATOR

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;

Entity Rule Is
Port(Clock     : In  Std_Logic;
     RST0,RST1 : In  Std_Logic;
     EN0,EN1   : In  Std_Logic;
     Input     : In  Std_Logic_Vector(7 downto 0);
     Output    : Out Std_Logic_Vector(7 downto 0));
End Entity Rule;

Architecture Arch_Rule Of Rule Is

Signal Mini,Out_Mini : Std_Logic_Vector(7 downto 0);
Signal Maxi,Out_Maxi : Std_Logic_Vector(7 downto 0);
Signal Sel0,Sel1     : Std_Logic;

Component Minimum
Port(A     : IN  std_logic_vector(7 DOWNTO 0);
     B     : IN  std_logic_vector(7 DOWNTO 0);
     A_LE_B: OUT std_logic);
End Component Minimum;

Component MUX2To1
Port(S : IN  std_logic;
     O : OUT std_logic_vector(7 DOWNTO 0);
     MA: IN  std_logic_vector(7 DOWNTO 0);
     MB: IN  std_logic_vector(7 DOWNTO 0));
End Component MUX2To1;

Component Maksimum
Port(A     : IN  std_logic_vector(7 DOWNTO 0);
     B     : IN  std_logic_vector(7 DOWNTO 0);
     A_GE_B: OUT std_logic);
End Component Maksimum;

Component Latch_Reg
Port(D_IN      : IN  std_logic_vector(7 DOWNTO 0);
     ASYNC_CTRL: IN  std_logic;
     CLK_EN    : IN  std_logic;
     CLOCK     : IN  std_logic;
     Q_OUT     : OUT std_logic_vector(7 DOWNTO 0));
End Component Latch_Reg;

Component Data_reg
Port(D_IN      : IN  std_logic_vector(7 DOWNTO 0);
     ASYNC_CTRL: IN  std_logic;
     CLK_EN    : IN  std_logic;
     CLOCK     : IN  std_logic;
     Q_OUT     : OUT std_logic_vector(7 DOWNTO 0));
End Component Data_Reg;

Begin
   U1: Minimum Port Map(A      => Input,
                        B      => Mini,
                        A_LE_B => Sel0);
```

```
   U2: MUX2To1 Port Map(S  => Sel0,
                        O  => Out_Mini,
                        MA => Mini,
                        MB => Input);

   U3: Latch_Reg Port Map(D_IN    => Out_Mini,
                          ASYNC_CTRL => RST0,
                          CLK_EN     => EN0,
                          CLOCK      => Clock,
                          Q_OUT      => Mini);

   U4: Maksimum Port Map(A      => Mini,
                         B      => Maxi,
                         A_GE_B => Sel1);

   U5: MUX2To1  Port Map(S  => Sel1,
                         O  => Out_Maxi,
                         MA => Maxi,
                         MB => Mini);

   U6: Data_Reg Port Map(D_IN       => Out_Maxi,
                         ASYNC_CTRL => RST1,
                         CLK_EN     => EN1,
                         CLOCK      => Clock,
                         Q_OUT      => Maxi);

       Output <= Maxi;
End Arch_Rule;
```

LAMPIRAN C-5

LISTING PROGRAM VHDL KOMPONEN DIVIDER

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;
USE IEEE.Std_Logic_Arith.All;

Entity Divider is
   Generic(Dividend_Width:Positive:=8;
            Divisor_Width :Positive:=4);
   Port   (Clock,DIV : In  Std_Logic;
            Dividend  : In  Std_Logic_Vector(Dividend_Width-1 downto 0);
            Divisor   : In  Std_Logic_Vector(Divisor_Width-1 downto 0);
            Quotient  : Out Std_Logic_Vector(Dividend_Width-1 downto 0));
End Divider;

Architecture Arch_Divider of Divider is
Type   State_Type Is (Initial,Shift_And_Sub);
Signal State  : State_Type;
Signal Dummy  : Std_Logic_Vector(Divisor_Width-1 downto 0);

Begin
   Process(DIV,Clock)
   Variable Bilangan : Unsigned(2*Dividend_Width downto 0);
   Variable Sub      : Unsigned(Dividend_Width downto 0);
   Variable Counter  : Integer Range 0 To Dividend_Width;

   Begin
      IF (Clock'Event And Clock='1') Then
            Case State Is
            When Initial =>
            IF (DIV='1') Then
```

```
                    Bilangan:=(Others=>'0');
                    Bilangan(Dividend_Width-1 downto 0):=Unsigned(Dividend);
            Dummy   <= Divisor;
            Counter := 1;
                    State   <= Shift_And_Sub;
                 ELSE
                    State   <= Initial;
                 END IF;
         When Shift_And_Sub =>
         IF (Counter<=Dividend_Width) Then
                    Bilangan:=SHL(Bilangan,"01");
                    Sub:=Bilangan(2*Dividend_Width downto Dividend_Width) - Unsigned(Dummy);
                    IF (Sub(Dividend_Width)='1') Then
                       Bilangan(0):='0';
                    ELSE
            Bilangan(2*Dividend_Width downto Dividend_Width):=Sub;
                       Bilangan(0):='1';
                    END IF;
            Counter := Counter+1;
                    State   <= Shift_And_Sub;
                 ELSE
            Quotient <= Conv_Std_Logic_Vector(Bilangan(Dividend_Width-1 downto 0),Dividend_Width);
            State    <= Initial;
                 END IF;
         End Case;
   End IF;
 End Process;
End Arch_Divider;
```

LAMPIRAN C-6

LISTING PROGRAM VHDL MODUL DEFUZZIFIER

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;
Use IEEE.Std_Logic_Arith.All;

Entity Defuzzifier Is
Port(Reset,Clock     : In  Std_Logic;
       ACC,HALF,DIV : In  Std_Logic;
       Input         : In  Std_Logic_Vector(8 downto 0);
       Output        : Out Std_Logic_Vector(7 downto 0));
End Entity Defuzzifier;

Architecture Arch_Defuzzifier Of Defuzzifier Is
Signal Denumerator : Std_Logic_Vector(7 downto 0);
Signal Numerator   : Std_Logic_Vector(9 downto 0);
Signal GND         : Std_Logic;
Component ACC8
Port(B         : IN  std_logic_vector(7 DOWNTO 0);
     CLK_EN    : IN  std_logic;
     CLOCK     : IN  std_logic;
     ASYNC_CTRL: IN  std_logic;
     Q_OUT     : OUT std_logic_vector(7 DOWNTO 0));
End Component ACC8;

Component Half_ACC
Port   (l : IN  std_logic;
   b : IN  std_logic_VECTOR(8 downto 0);
   ce: IN  std_logic;
   c : IN  std_logic;
   ci: IN  std_logic;
   s : OUT std_logic_VECTOR(9 downto 0));
End Component Half_ACC;

Component Divider
Generic(Dividend_Width:Positive:=8;
       Divisor_Width :Positive:=4);
Port(Clock,DIV : In  Std_Logic;
     Dividend  : In  Std_Logic_Vector(Dividend_Width-1 downto 0);
     Divisor   : In  Std_Logic_Vector(Divisor_Width-1 downto 0);
     Quotient  : Out Std_Logic_Vector(Dividend_Width-1 downto 0));
End Component Divider;

Begin
   U1: ACC8 Port Map(B         => Input(7 downto 0),
                     CLK_EN    => ACC,
                     CLOCK     => Clock,
                     ASYNC_CTRL => Reset,
                     Q_OUT     => Denumerator);
       GND <= '0';
   U2: Half_ACC  Port Map(l  => Reset,
                          b  => Input,
                          ce => HALF,
                          c  => Clock,
```

```
                        ci => GND,
                        s  => Numerator);
   U3: Divider Port Map(Clock     => Clock,
                        DIV       => DIV,
                        Dividend => Numerator(7 downto 0),
                        Divisor  => Denumerator(7 downto 4),
                        Quotient => Output);
End Arch_Defuzzifier;
```

LAMPIRAN C-7

ORGANISASI MEMORI PADA PROGRAM STORAGE

Address	**Centre**	**Width**	**Tipe Status**	**I/O**	***Hexa Desimal***
00	**0000**	**01**	**000**	**0**	***010***
01	0101	00	001	1	*143*
02	0001	10	010	0	*064*
03	0101	00	001	1	*143*
04	0010	01	010	0	*094*
05	0101	00	001	1	*143*
06	0011	01	010	0	*0D4*
07	0101	00	001	1	*143*
08	0100	01	011	0	*116*
09	0101	00	100	1	*148*
0A	**0000**	**01**	**000**	**0**	***010***
0B	0110	00	011	1	*187*
0C	0111	11	001	1	*1F3*
0D	0001	10	010	0	*064*
0E	0110	00	011	1	*187*
0F	0111	11	001	1	*1F3*
10	0010	01	010	0	*094*
11	0110	00	011	1	*187*
12	0111	11	001	1	*1F3*
13	0011	01	010	0	*0D4*
14	0110	00	011	1	*187*
15	0111	11	001	1	*1F3*
16	0100	01	011	0	*116*
17	0110	00	011	1	*187*
18	0111	11	111	1	*1F9*
19	**0000**	**01**	**000**	**0**	***010***
1A	1001	11	101	1	*27B*
1B	0001	10	010	0	*064*
1C	1001	11	101	1	*27B*
1D	0010	01	010	0	*094*
1E	1001	11	101	1	*27B*
1F	0011	01	010	0	*0D4*
20	1001	11	101	1	**27B**
21	0100	01	011	0	*116*

22	1001	11	110	1	*27D*
23	00010000				010
24	01010000				050
25	10000110				086
26	10110100				0B4
27	11100101				0E5
28	01000100				044
29	01110111				077
2A	10011001				099
2B	11011101				0DD
2C	111	100010			1E2
2D	110	100110			1A6
2E	101	110000			170
2F	101	110011			173

LAMPIRAN C-8

LISTING PROGRAM VHDL MODUL KONTROLER

```
Process(Reset,Clock)
   Begin
      IF (Reset='1') Then
       Address <= (Others => '0');
       CS      <= True;
       State   <= ADC;
      ELSIF (Clock'Event And Clock='1') Then
       Case State Is
       When ADC =>
                  IF (INT='0') Then
                IF (Channel='0') Then
                        Temp   <= Input;
                        State  <= ADC;
                        AD_Mux <= '1';
                ELSE
                        Moist  <= Input;
                        State  <= Control_Centre;
                        AD_Mux <= '0';
                END IF;
                RD  <= '1';
                   ELSE
                      IF (AD_Mux='1') Then
                              Channel <= '1';
                      ELSE
                              Channel <= '0';
```

```
                END IF;
                PWRDN <= '1';
                RD    <= '0';
                  State <= ADC;
            END IF;
When Control_Centre =>
     Dummy <= Address;
     IF (CS=True) Then
        Address <= Conv_Std_Logic_Vector(Unsigned(Out_Program(9 downto 6))+Byte_23,6);
        EN1     <= EN0 AND S0;
        State   <= Control_Width;
     ELSE
        DIV     <= ACC AND S3;
        Address <= Conv_Std_Logic_Vector(Unsigned(Out_Program(5 downto 4))+Byte_2C,6);
        ACC     <= '0';
        HALF    <= '0';
        State   <= Control_Slope;
     END IF;
     EN0  <= '0';
When Control_Width =>
     ACC     <= EN1 AND S1;
     HALF    <= EN1 AND S2;
     Centre  <= Out_Program(7 downto 0);
     Address <= Dummy;
     CS      <= False;
     EN1     <= '0';
     State   <= Control_Centre;
When Control_Slope =>
     DIV     <= '0';
     Width   <= Out_Program(5 downto 0);
     Slope   <= Out_program(8 downto 6);
     Address <= Dummy;
     RST0 <= S0; RST1 <= S1; RST2 <= S3;
     ACC  <= S3; HALF <= S3;
       State   <= Control_Status;
```

```
When Control_Status =>
     AD_Sel <= Out_Program(0);
         Address<= Conv_Std_Logic_Vector(Unsigned(Dummy)+1,6);
     State  <= Control_Centre;
           Case Out_Program(3 downto 1) Is
           When "000" =>
                 Tipe <= "00";
                 S0 <= '0'; S1 <= '0'; S2 <= '0'; S3 <= '0';
     When "001" =>
                 Tipe <= "00";
                 S0 <= '1'; S1 <= '0'; S2 <= '0'; S3 <= '0';
           When "010" =>
                 Tipe <= "01";
                 S0 <= '0'; S1 <= '0'; S2 <= '0'; S3 <= '0';
           When "011" =>
                 Tipe <= "10";
                 S0 <= '0'; S1 <= '0'; S2 <= '0'; S3 <= '0';
           When "100" =>
                 Tipe <= "00";
                 S0 <= '1'; S1 <= '1'; S2 <= '0'; S3 <= '0';
           When "101" =>
                 Tipe <= "10";
                 S0 <= '1'; S1 <= '0'; S2 <= '0'; S3 <= '0';
           When "110" =>
                 Tipe <= "10";
                 S0 <= '1'; S1 <= '1'; S2 <= '1'; S3 <= '1';
           State   <= ADC;
           Address <= (Others => '0');
           When "111" =>
                 Tipe <= "00";
                 S0 <= '1'; S1 <= '1'; S2 <= '1'; S3 <= '0';

           When Others =>
           End Case;
     RST0 <= '0'; RST1 <= '0'; RST2 <= '0';
```

```
            ACC  <= '0'; HALF <= '0';
            EN0 <= '1';
            CS     <= True;
        End Case;
      END IF;
END Process;
```

LAMPIRAN C-9

LISTING PROGRAM VHDL MODUL TEMPERATURE DEKODER

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;
Use IEEE.Std_Logic_Arith.All;

Entity TEMP_DEC Is
Port(Input          : In  Std_Logic_Vector(5 downto 0);
     Output         : Out Std_Logic_Vector(7 downto 0));
End Entity TEMP_DEC;

Architecture ARCH_TEMP_DEKODER Of TEMP_DEC Is
Constant Byte_30 : Std_Logic_Vector(6 downto 0) := "0011110";
Signal   A       : Std_Logic_Vector(7 downto 0);

Begin
   Process(Input,A)
   Begin
      A(3 downto 0) <= Input(5 downto 2);
      Output <=
Conv_Std_Logic_Vector(Unsigned(A)+Unsigned(Input)+Unsigned(Byte_30
),8);
   End Process;
End ARCH_TEMP_DEKODER;
```

LAMPIRAN C-10

LISTING PROGRAM VHDL MODUL MOISTURE DEKODER

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;
Use IEEE.Std_Logic_Arith.All;

Entity Moist_Dec Is
Port(Input  : In  Std_Logic_Vector(7 downto 0);
     Output : Out Std_Logic_Vector(7 downto 0));
End Entity Moist_Dec;

Architecture Arch_Moist_Dec Of Moist_Dec Is
Signal   Dummy    : Unsigned(15 downto 0);
Constant Byte_100 : Unsigned(7 downto 0) := "01100100";
Begin
   Process(Input,Dummy)
   Begin
      Dummy  <= Conv_Unsigned((Unsigned(Input)+1)*Byte_100,16);
      Output <= Conv_Std_Logic_Vector(Dummy(15 downto
8)+Dummy(7),8);
   End Process;
End  Arch_Moist_Dec;
```

LAMPIRAN C-11

LISTING PROGRAM VHDL MODUL HEXA TO DESIMAL CONVERTER

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;
USE IEEE.Std_Logic_Arith.All;

Entity HEX_DEC is
Generic(Dividend_Width:Positive:=8;
      Divisor_Width :Positive:=4);
Port    (Clock,DIV  : In  Std_Logic;
      Input       : In  Std_Logic_Vector(Dividend_Width-1 downto 0);
      Byte_Valid : Out Std_Logic;
      Remainder  : Out Std_Logic_Vector(Divisor_Width-1 downto 0));
End Entity HEX_DEC;

Architecture ARCH_HEX_TO_DEC of HEX_DEC is
Signal   State     : Std_Logic_Vector(1 downto 0);
Signal       Dividend : Std_Logic_Vector(Dividend_Width-1 downto 0);
Constant Divisor  : Std_Logic_Vector(Divisor_Width-1 downto 0) := "1010";

Begin
   Process(DIV,Clock,Input)
   Variable Bilangan : Unsigned(2*Dividend_Width downto 0);
   Variable Sub      : Unsigned(Dividend_Width downto 0);
   Variable Counter  : Natural Range 0 To Dividend_Width;
   Variable Digit    : Natural Range 1 To 3;

   Begin
```

```
IF (Clock'Event And Clock='1') Then
      Case State Is
When "00"   =>
      Byte_Valid  <= '0';
        IF (DIV='1') Then
         Digit    := 1;
         Dividend <= Input;
           State    <= "01";
        ELSE
           State    <= "00";
        END IF;
      When "01" =>
               Bilangan:=(Others=>'0');
               Bilangan(Dividend_Width-1 downto 0):=Unsigned(Dividend);
         Counter := 1;
         Byte_Valid <= '0';
         State      <= "10";
      When "10" =>
      IF (Counter<=Dividend_Width) Then
               Bilangan:=SHL(Bilangan,"01");
         Sub:=Bilangan(2*Dividend_Width downto Dividend_Width) - Unsigned(Divisor);
               IF (Sub(Dividend_Width)='1') Then
                        Bilangan(0):='0';
               ELSE
                  Bilangan(2*Dividend_Width downto Dividend_Width):=Sub;
                        Bilangan(0):='1';
               END IF;
         Counter := Counter+1;
               State   <= "10";
            ELSE
         Byte_Valid <= '1';
         Remainder             <=      Conv_Std_Logic_Vector(Bilangan(2*Dividend_Width      downto
Dividend_Width),Divisor_Width);
         IF (Digit=3) Then
            State      <= "00";
```

```
                ELSE
                   Digit    := Digit+1;
                   Dividend <= Conv_Std_Logic_Vector(Bilangan(Dividend_Width-1 downto 0),Dividend_Width);
                   State    <= "01";
                END IF;
             END IF;
       When Others =>
             End Case;
       End IF;
   End Process;
End ARCH_HEX_TO_DEC;
```

LAMPIRAN C-12

ISI FILE ROM_LCD.MEM UNTUK INISIALISASI LCD

```
;
; memfile ROM.mem for LogiBLOX symbol ROM16x8
; Created on Tuesday, November 09, 1999 00:39:04
;
; Header Section
RADIX 10
DEPTH 32
WIDTH 9
DEFAULT 20
;
; Data Section
; Specifies data to be stored in different addresses
; e.g., DATA 0:A, 1:0
RADIX 16
DATA 00:030, 01:030, 02:030, 03:038, 04:008, 05:001, 06:006,
07:00C, 08:153, 09:175,
     0A:168, 0B:175, 0C:120, 0D:14C, 0E:165, 0F:16D, 10:162,
11:161, 12:162, 13:120,
     14:14C, 15:161, 16:16D, 17:161, 18:0C3, 19:1DF, 1A:146,
1B:0C9, 1C:125, 1D:0CF,
     1E:127, 1F:004
; end of LogiBLOX memfile
```

LAMPIRAN C-13

LISTING PROGRAM VHDL LCD DRIVER

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;
Use IEEE.Std_Logic_Arith.All;

Entity DISPLAY Is
Port(Rst,Clock,Clk: In  Std_Logic;
     Temp     : In  Std_Logic_Vector(5 downto 0);
     Moist,Time   : In  Std_Logic_Vector(7 downto 0);
     RS,EN,Ready  : Out Std_Logic;
     Output       : Out Std_Logic_Vector(7 downto 0));
End Entity DISPLAY;

Architecture Arch_Display Of DISPLAY Is
Type State_Type Is (Init,Dekoder_Posisi,Out_Data,Enable_Data);
Signal State     : State_Type;
Signal Mux_Sel   : Std_Logic_Vector(1 downto 0);
Signal Address   : Std_Logic_Vector(4 downto 0);
Signal Count     : Std_Logic_Vector(1 downto 0);
Signal Out_Mux   : Std_Logic_Vector(7 downto 0);
Signal Temp_Hex  : Std_Logic_Vector(7 downto 0);
Signal Moist_Hex : Std_Logic_Vector(7 downto 0);
Signal Data_LCD  : Std_Logic_Vector(3 downto 0);
Signal Out_ROM   : Std_Logic_Vector(8 downto 0);
Signal MAX   : Std_Logic;
Signal Byte_Request,Byte_Valid : Std_Logic;
Constant Byte_30 : Unsigned(5 downto 0) := "110000";

Component TEMP_DEC
Port(Input          : In  Std_Logic_Vector(5 downto 0);
     Output         : Out Std_Logic_Vector(7 downto 0));
End Component TEMP_DEC;

Component MOIST_DEC
Port(Input  : In  Std_Logic_Vector(7 downto 0);
     Output : Out Std_Logic_Vector(7 downto 0));
End Component MOIST_DEC;

Component HEX_DEC
Generic(Dividend_Width:Positive:=8;
      Divisor_Width :Positive:=4);
Port   (Clock,DIV  : In  Std_Logic;
      Input      : In  Std_Logic_Vector(Dividend_Width-1 downto
0);
      Byte_Valid : Out Std_Logic;
      Remainder  : Out Std_Logic_Vector(Divisor_Width-1 downto
0));
End Component HEX_DEC;
```

```
Component ROM_LCD
Port(A  : IN  Std_Logic_Vector(4 DOWNTO 0);
     DO : OUT Std_Logic_Vector(8 DOWNTO 0));
End Component ROM_LCD;

Component COUNTER
Port(CLOCK: IN std_logic;
     ASYNC_CTRL: IN std_logic;
     Q_OUT: OUT std_logic_vector(1 DOWNTO 0));
End Component COUNTER;

Component MUX3TO1
Port(S : IN std_logic_vector(1 DOWNTO 0);
     O : OUT std_logic_vector(7 DOWNTO 0);
     MA: IN std_logic_vector(7 DOWNTO 0);
     MB: IN std_logic_vector(7 DOWNTO 0);
     MC: IN std_logic_vector(7 DOWNTO 0));
End Component MUX3TO1;

Begin
   U1: TEMP_DEC Port Map(Input  => Temp,
                         Output => Temp_Hex);

   U2: MOIST_DEC Port Map(Input  => Moist,
                          Output => Moist_Hex);

   U3: ROM_LCD Port Map(A  => Address,
                        DO => Out_ROM);

   U4: COUNTER Port Map(CLOCK      => Clk,
                        ASYNC_CTRL => MAX,
                        Q_OUT      => Count);

   U5: MUX3TO1 Port Map(S  => Mux_Sel,
                        O  => Out_Mux,
                        MA => Temp_Hex,
                        MB => Moist_Hex,
                        MC => Time);

   U6: HEX_DEC Generic Map(Dividend_Width => 8,
                           Divisor_Width  => 4)
               Port Map(Clock      => Clock,
                        DIV        => Byte_Request,
                        Input      => Out_Mux,
                        Byte_Valid => Byte_Valid,
                        Remainder  => Data_LCD);

   LCD_SM:Process(Clock,Rst)
   Variable Data : Natural Range 0 To 3;
   Begin
      IF (Rst='1') Then
       Address <= (Others=>'0');
       State   <= Init;
      ELSIF (Clock'Event And Clock='1') Then
       Case State Is
```

```
      When Init =>
         IF (Count="00") Then
            MAX    <= '0';
            EN     <= '1';
            RS     <= Out_ROM(8);
              Output <= Out_ROM(7 downto 0);
         ELSIF (Count="11") Then
            MAX <= '1';
            Address <=
Conv_Std_Logic_Vector(Unsigned(Address)+1,5);
            IF (Address="11111") Then
             Ready   <= '1';
               Mux_Sel <= "10";
               State   <= Dekoder_Posisi;
            END IF;
         ELSE
            EN    <= '0';
         END IF;
      When Dekoder_Posisi   =>
           Byte_Request <= '1';
         EN   <= '1';
         RS   <= '0';
         Data := 0;
         IF (Mux_Sel="00") Then
            Output  <= "11001000";
            Mux_Sel <= "01";
         ELSIF (Mux_Sel="01") Then
            Output  <= "11001110";
            Mux_Sel <= "10";
         ELSIF (Mux_Sel="10") Then
            Output  <= "11000010";
            Mux_Sel <= "00";
         END IF;
         State <= Enable_Data;
      When Out_Data =>
         IF (Byte_Valid='1') Then
             RS     <= '1';
             EN     <= '1';
             Output <=
Conv_Std_Logic_Vector(Unsigned(Data_LCD)+Byte_30,8);
             Data   := Data+1;
           State  <= Enable_Data;
           END IF;
      When Enable_Data =>
         EN             <= '0';
         Byte_Request <= '0';
         IF (Data=3) Then
            State <= Dekoder_Posisi;
         ELSE
            State <= Out_Data;
         END IF;
       End Case;
      End IF;
   End Process LCD_SM;
End Arch_Display;
```

LAMPIRAN C-14

LISTING PROGRAM VHDL PWM DRIVER

```
Library IEEE;
Use IEEE.Std_Logic_1164.All;

Entity PWM Is
Port(Rst       : In  Std_Logic;
     Clk,F15   : In  Std_Logic;
     Input     : In  Std_Logic_Vector(7 downto 0);
     Linier    : Out Std_Logic_Vector(7 downto 0);
     Output    : Out Std_Logic);
End Entity PWM;

Architecture Arch_PWM Of PWM Is
Signal Out_Mux,Out_Reg  : Std_Logic_Vector(7 downto 0);
Signal Lin_Out,Ramp     : Std_Logic_Vector(7 downto 0);
Signal Clock,Sec,Min,EN : Std_Logic;
Signal TC,New_Data,Load : Std_Logic;

Component DETIK
Port(CLOCK  : IN std_logic;
     CLK_OUT: OUT std_logic);
End Component DETIK;

Component COMPARE
Port(A: IN std_logic_vector(7 DOWNTO 0);
     B: IN std_logic_vector(7 DOWNTO 0);
     A_NE_B: OUT std_logic);
End Component COMPARE;

Component COUNTER6
Port(CLOCK     : IN std_logic;
     ASYNC_CTRL: IN std_logic;
     TERM_CNT  : OUT std_logic);
End Component COUNTER6;

Component COUNTER8
Port(D_IN      : IN std_logic_vector(7 DOWNTO 0);
     LOAD      : IN std_logic;
     CLK_EN    : IN std_logic;
     CLOCK     : IN std_logic;
     ASYNC_CTRL: IN std_logic;
     Q_OUT     : OUT std_logic_vector(7 DOWNTO 0));
End Component COUNTER8;

Component UP_COUNT
Port(CLK_EN    : IN std_logic;
     CLOCK     : IN std_logic;
     ASYNC_CTRL: IN std_logic;
     Q_OUT     : OUT std_logic_vector(7 DOWNTO 0);
```

```
     TERM_CNT  : OUT std_logic);
End Component UP_COUNT;

Component DATA
Port(D_IN : IN std_logic_vector(7 DOWNTO 0);
     CLOCK: IN std_logic;
     Q_OUT: OUT std_logic_vector(7 DOWNTO 0));
End Component DATA;

Component MUX2TO1
Port(S : IN std_logic;
     O : OUT std_logic_vector(7 DOWNTO 0);
     MA: IN std_logic_vector(7 DOWNTO 0);
     MB: IN std_logic_vector(7 DOWNTO 0));
End Component MUX2TO1;

Component SMALLER
Port(A: IN std_logic_vector(7 DOWNTO 0);
     B: IN std_logic_vector(7 DOWNTO 0);
     A_LT_B: OUT std_logic);
End Component SMALLER;

Component DFF
Port(C : In  Std_Logic;
     D : In  Std_Logic;
     Q : Out Std_Logic);
End Component DFF;

Begin
   U1: DETIK Port Map(CLOCK   => F15,
                      CLK_OUT => Sec);

   U2: COUNTER6 Port Map(CLOCK      => Sec,
                         ASYNC_CTRL => Rst,
                         TERM_CNT   => Min);

   U3: COMPARE Port Map(A => Out_Reg,
                        B => Input,
                        A_NE_B => New_Data);

   U4: MUX2TO1 port map(S  => New_Data,
                        O  => Out_Mux,
                        MA => Out_Reg,
                        MB => Input);

   U5: DATA Port Map(D_IN  => Out_Mux,
                     CLOCK => Clk,
                     Q_OUT => Out_Reg);

       Linier <= Lin_Out;
       Clock  <= Load OR Min;
   U6: COUNTER8 Port Map(D_IN       => Out_Mux,
                         LOAD       => New_Data,
                         CLK_EN     => EN,
                         CLOCK      => Clock,
                           ASYNC_CTRL => Rst,
```

```
                Q_OUT      => Lin_Out);

  U7: UP_COUNT Port Map(CLK_EN     => EN,
                CLOCK      => Sec,
                ASYNC_CTRL => Rst,
                Q_OUT      => Ramp,
                TERM_CNT   => TC);

  U8: SMALLER Port Map(A => Ramp,
                B => Lin_Out,
                A_LT_B => Output);

  U9: DFF Port Map(C => Clk,
              D => New_Data,
              Q => Load);

  U10:Process(Rst,New_Data)
      Begin
       IF (Rst='1') Then
          EN <= '0';
        ELSIF (New_Data='1') Then
           EN <= '1';
        END IF;
      End Process;
End Arch_PWM;
```

LAMPIRAN C-15

LISTING PROGRAM VHDL KOMPONEN CLOCK GENERATOR

```
Component OSC4
Port(F8M  : Out Std_Logic;
     F16K : Out Std_Logic;
     F15  : Out Std_Logic);
End Component OSC4;

Component CLK_LCD
Port(CLOCK  : IN std_logic;
     CLK_OUT: OUT std_logic);
End Component CLK_LCD;

Component BUFG
Port(I : In  Std_Logic;
     O : Out Std_Logic);
End Component BUFG;

U0: OSC4 Port Map(F8M   => F8M,
                  F16K  => F16K,
                  F15   => F15);

U1: BUFG Port Map(I => F16K,
                  O => Clock);

U2: CLK_LCD Port Map(CLOCK   => F16K,
                     CLK_OUT => F490);
```

LAMPIRAN D-1

HASIL PENGUJIAN PROSES FUZZY LOGIC

1. Percobaan 1

Suhu Udara = 82 °F → Heksa = A9
Kelembaban Tanah = 10 % → Heksa = 54
Lama = 68 menit

Cold = 00 Cool = 00 Normal = 2D Warm = BD Hot = 00
Dry = 8F Moist = 00 Wet = 00

	Cold	Cool	Normal	Warm	Hot
Wet	00	00	00	00	00
Moist	00	00	00	00	00
Dry	00	00	2D	8F	00

Short = 00
Medium = 00
Long = 8F

$$Output = 4 \times \frac{(0+0+8F)}{\left[\frac{0+0+8F}{10}\right]}$$

$$Output = 4 \times \frac{8F}{8} = (44)_H = (68)_D$$

2. Percobaan 2

Suhu Udara = 80 °F → Heksa = A3
Kelembaban Tanah = 13 % → Heksa = 70
Lama = 31 menit

Cold = 00 Cool = 00 Normal = 51 Warm = 99 Hot = 00
Dry = 00 Moist = CE Wet = 00

	Cold	Cool	Normal	Warm	Hot
Wet	00	00	00	00	00
Moist	00	00	51	99	00
Dry	00	00	00	00	00

Short = 00
Medium = 99
Long = 00

$$Output = 4 \times \frac{\left(0+\frac{99}{2}+8F\right)}{\left[\frac{0+99+8F}{10}\right]}$$

$$Output = 4 \times \frac{4C}{9} = (20)_H = (32)_D$$

3. Percobaan 3

Suhu Udara = 82 °F → Heksa = A9
Kelembaban Tanah = 11 % → Heksa = 5C
Lama = 52 menit

Cold = 00 Cool = 00 Normal = 2D Warm = BD Hot = 00
Dry = 57 Moist = 42 Wet = 00

	Cold	Cool	Normal	Warm	Hot
Wet	00	00	00	00	00
Moist	00	00	2D	42	00
Dry	00	00	2D	57	00

Short = 00
Medium = 42
Long = 57

$$Output = 4 \times \frac{\left(0 + \frac{42}{2} + 57\right)}{\left[\frac{0 + 42 + 57}{10}\right]}$$

$$Output = 4 \times \frac{78}{9} = (34)_H = (52)_D$$

4. Percobaan 4

Suhu Udara = 81 °F → Heksa = A6
Kelembaban Tanah = 22 % → Heksa = B8
Lama = 20 menit

Cold = 00 Cool = 00 Normal = 3F Warm = AB Hot = 00
Dry = 00 Moist = 64 Wet = 2D

	Cold	Cool	Normal	Warm	Hot
Wet	00	00	2D	2D	00
Moist	00	00	3F	64	00
Dry	00	00	00	00	00

Short = 2D
Medium = 64
Long = 00

$$Output = 4 \times \frac{\left(0 + \frac{64}{2} + 0\right)}{\left[\frac{2D + 64 + 0}{10}\right]}$$

$$Output = 4 \times \frac{32}{9} = (14)_H = (20)_D$$

5. Percobaan 5

Suhu Udara = 82 °F → Heksa = A9
Kelembaban Tanah = 9 % → Heksa = 4F
Lama = 64 menit

Cold = 00 Cool = 00 Normal = 2D Warm = BD Hot = 00
Dry = B2 Moist = 00 Wet = 00

	Cold	Cool	Normal	Warm	Hot
Wet	00	00	00	00	00
Moist	00	00	00	00	00
Dry	00	00	2D	B2	00

Short = 00
Medium = 00
Long = B2

$$Output = 4 \times \frac{(0+0+B2)}{\left[\frac{0+0+B2}{10}\right]}$$

$$Output = 4 \times \frac{B2}{B} = (40)_H = (64)_D$$

6. Percobaan 6

Suhu Udara = 77 °F → Heksa = 99
Kelembaban Tanah = 13 % → Heksa = 70
Lama = 32 menit

Cold = 00 Cool = 00 Normal = 8D Warm = 5D Hot = 00
Dry = 00 Moist = FF Wet = 00

	Cold	Cool	Normal	Warm	Hot
Wet	00	00	00	00	00
Moist	00	00	8D	5D	00
Dry	00	00	00	00	00

Short = 00
Medium = 8D
Long = 00

$$Output = 4 \times \frac{\left(0+\frac{8D}{2}+00\right)}{\left[\frac{0+8D+0}{10}\right]}$$

$$Output = 4 \times \frac{46}{8} = (20)_H = (32)_D$$

LAMPIRAN D-2

HASIL PENGUJIAN ALAT DENGAN XS40-010XL BOARD

Prosedur Pengujian dengan XSBoard :

1. Masukkan Sumber tegangan DC 9 Volt ke jack supply dari XESS Board, perhatikan apakah LED pada seven segment menyala atau tidak. Jika ya, berarti sumber tegangan dapat masuk ke Board X40-010XL. Jika tidak, lepas sumber tegangan dan masukkan kembali secara benar, periksa pula level tegangan DC yang masuk ke Board.
2. Uji dahulu XESS Board XS40010XL dengan menggunakan program XSTEST sebagai berikut :

   XSTEST XS40-10XL

   Apabila LED pada seven segment menunjukkan angka '0' maka XESS Board XS40-010XL dapat berfungsi dengan baik. Jika tidak, lepas sumber tegangan DC 9Volt dan perhatikan kembali prosedur pemakaian XS40 Board dan perhatikan apakah chip anda rusak atau tidak (hal ini dapat dilakukan dengan mengganti IC FPGA XS40010XL dengan yang baru).
3. Download File bit stream hasil implementasi dengan menggunakan Program XSLOAD dengan perintah

   XSLOAD TA.bit

   Perintah (command) diatas akan menyebabkan LED pada seven segment XS40 Board berkedip sebentar, kemudian semua LEDnya akan padam.

4. Hidupkan sumber tegangan untuk LCD dan ADC MAX114. Hal ini akan menyebabkan pada layar LCD akan tampak hitam pada baris pertama (karena LCD belum diinisialisasi). Periksa level tegangan referensi dari ADC serta LCD.
5. Setelah program didownload, lakukan pengetestan dengan mereset rangkaian anda menggunakan program XSPORT sebagai berikut:

   XSPORT 1

   Tekan Enter

   XSPORT 0

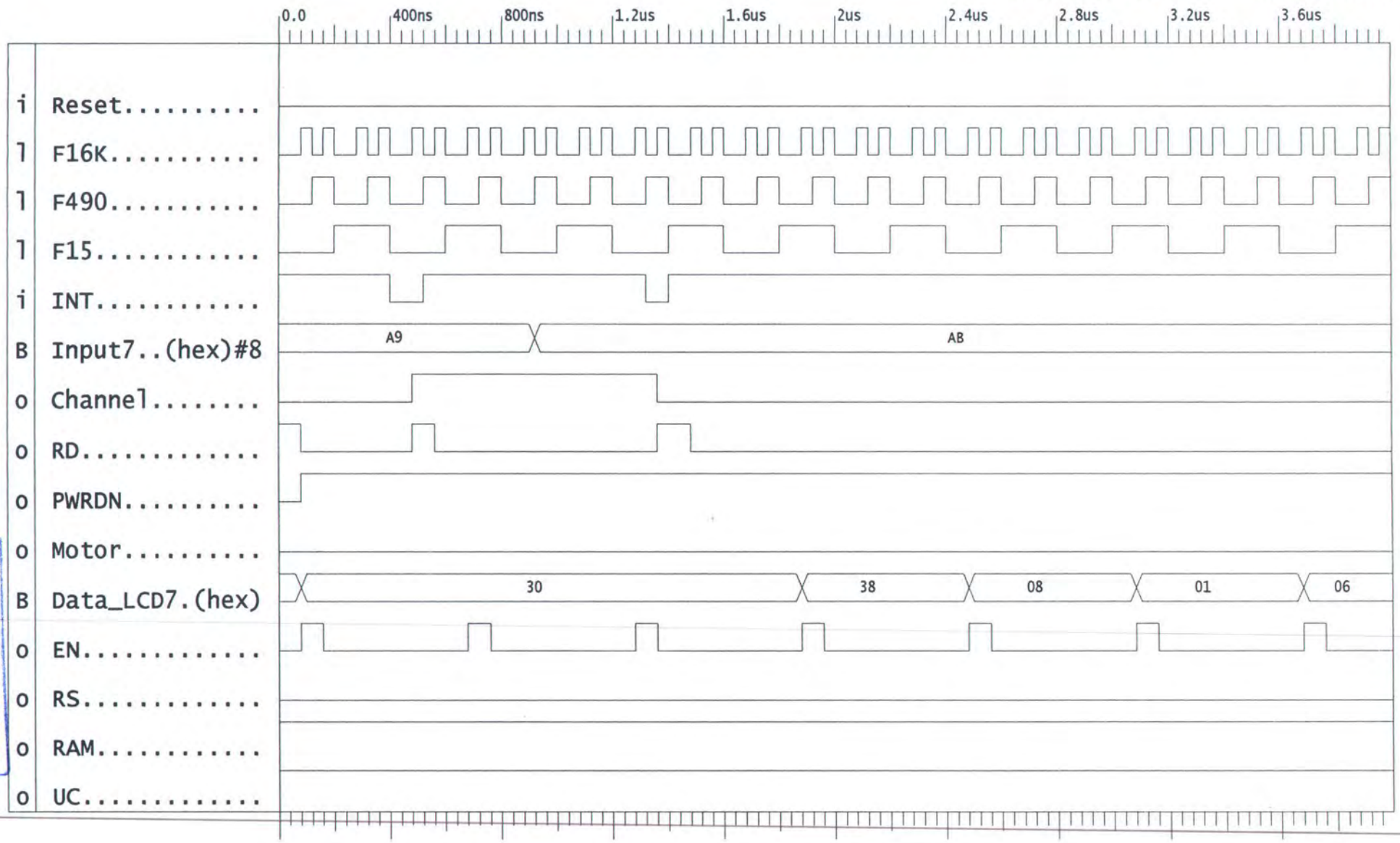
   Tekan Enter

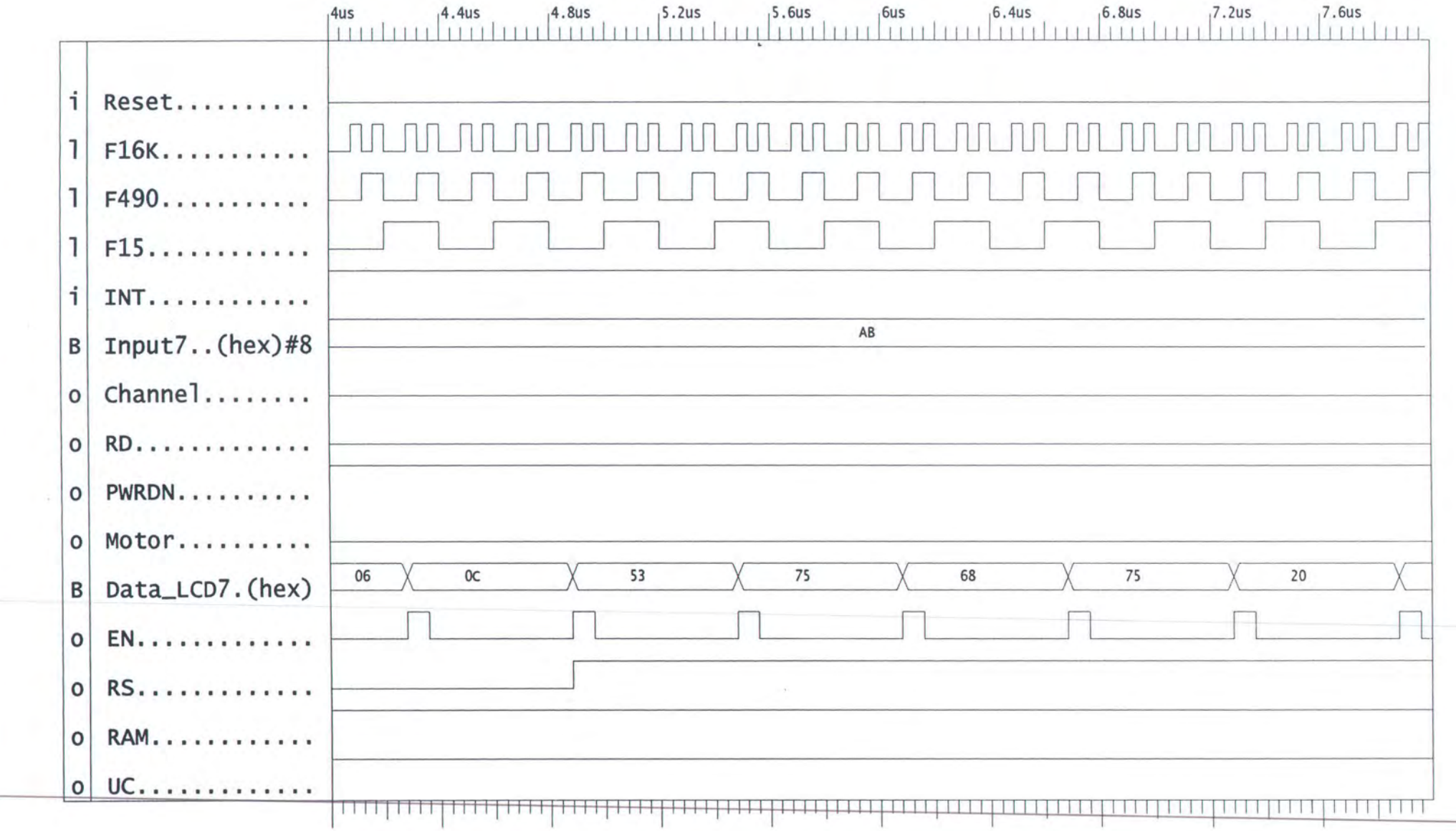
   Pada pengujian diatas dilakukan dengan memberikan input '1' pada pin nomor 44 (Reset) untuk mereset semua flip-flop. Kemudian memberi input '0' untuk membuat rangkaian berjalan dengan normal.
6. Ukur level tegangan DC dari input channel ADC, kemudian catat hasil tampilan pada LCD.
7. Lakukan percobaan 5 berulang-ulang untuk mendapatkan tegangan input analog ADC yang berbeda-beda dengan mengubah posisi multi turn (mengubah harga tahanannya).

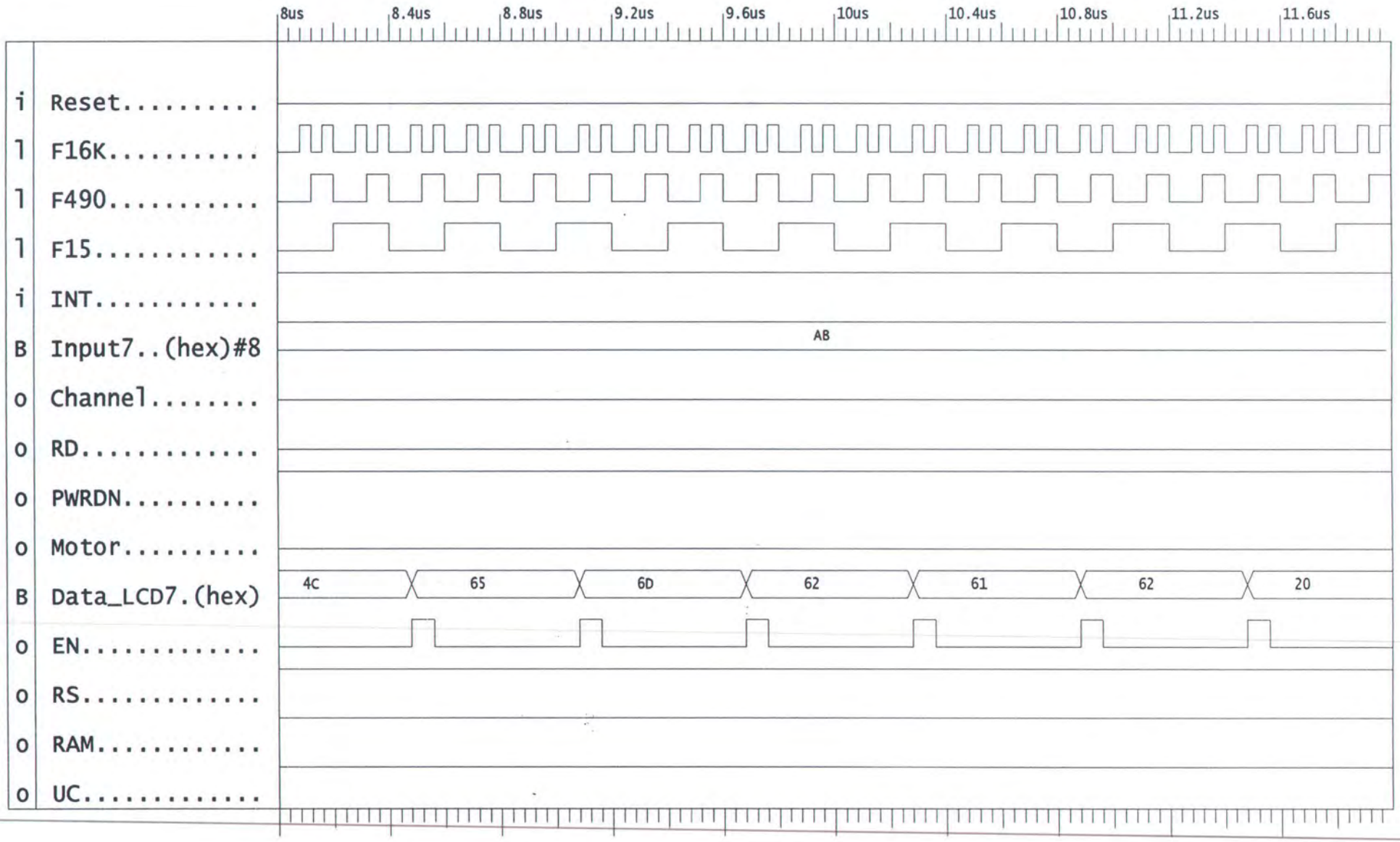
Percobaan	Tegangan Input		Output ADC		Tampilan LCD		
	Channel 1	Channel 2	Channel 1	Channel 2	Suhu	Lembab	Waktu
1	1,76	2,86	5A	92	57	75	32
2	3,85	1,82	C5	5D	91	59	16
3	1,50	2,52	4C	81	53	70	32
4	4,06	3,75	D0	C0	93	88	48
5	3,00	4,00	99	CC	77	93	68

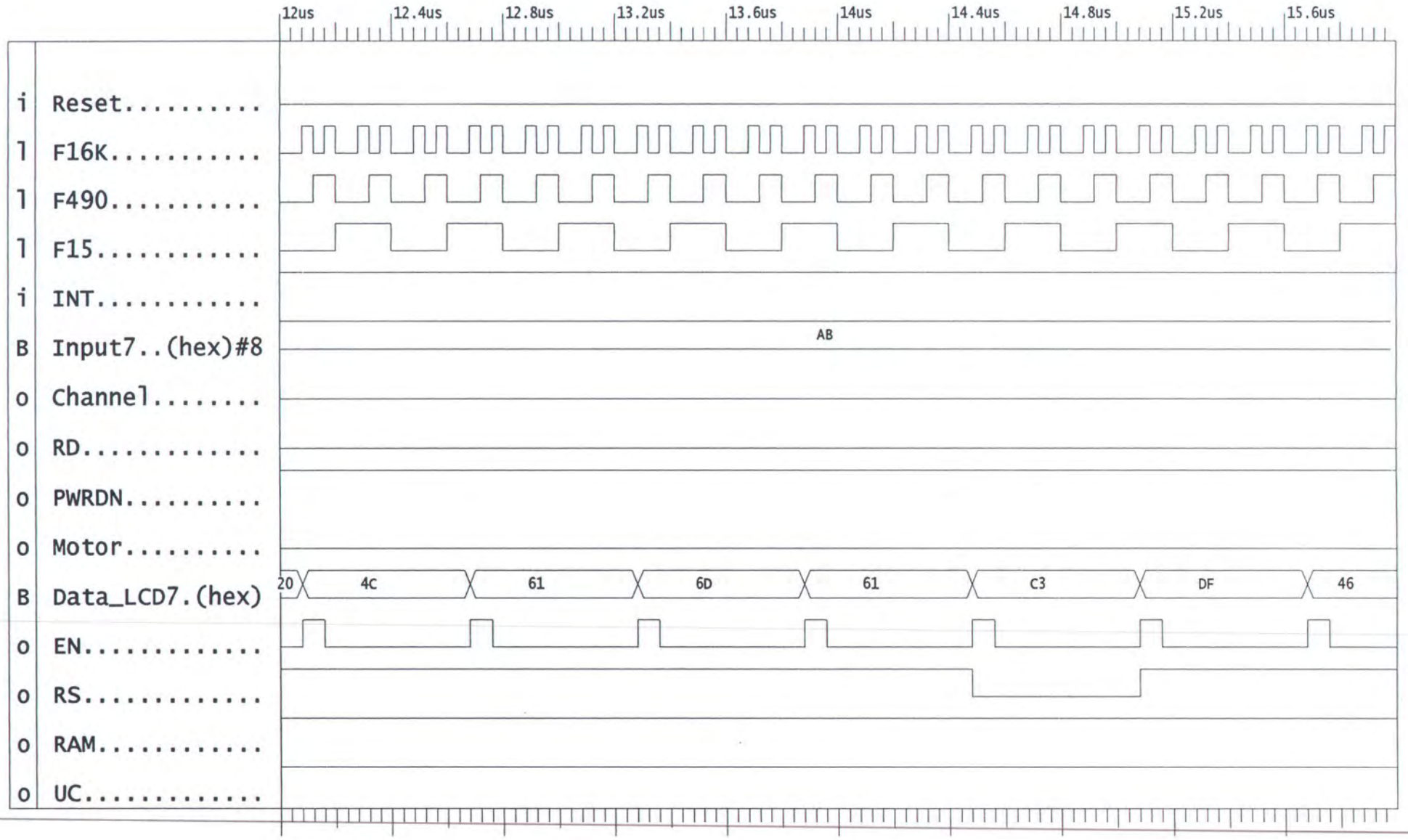
LAMPIRAN D-3

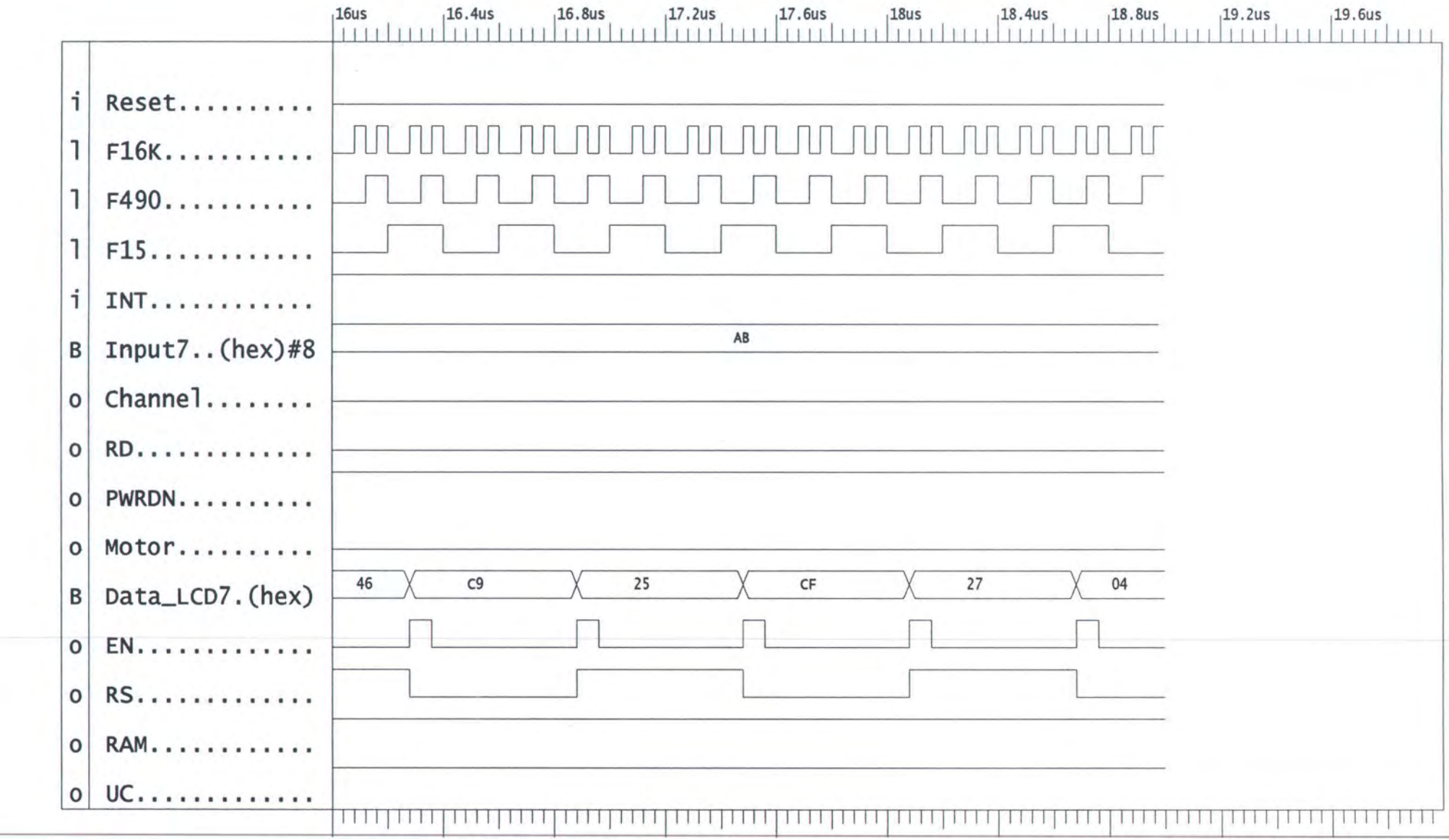
SIMULASI TIMING TUGAS AKHIR

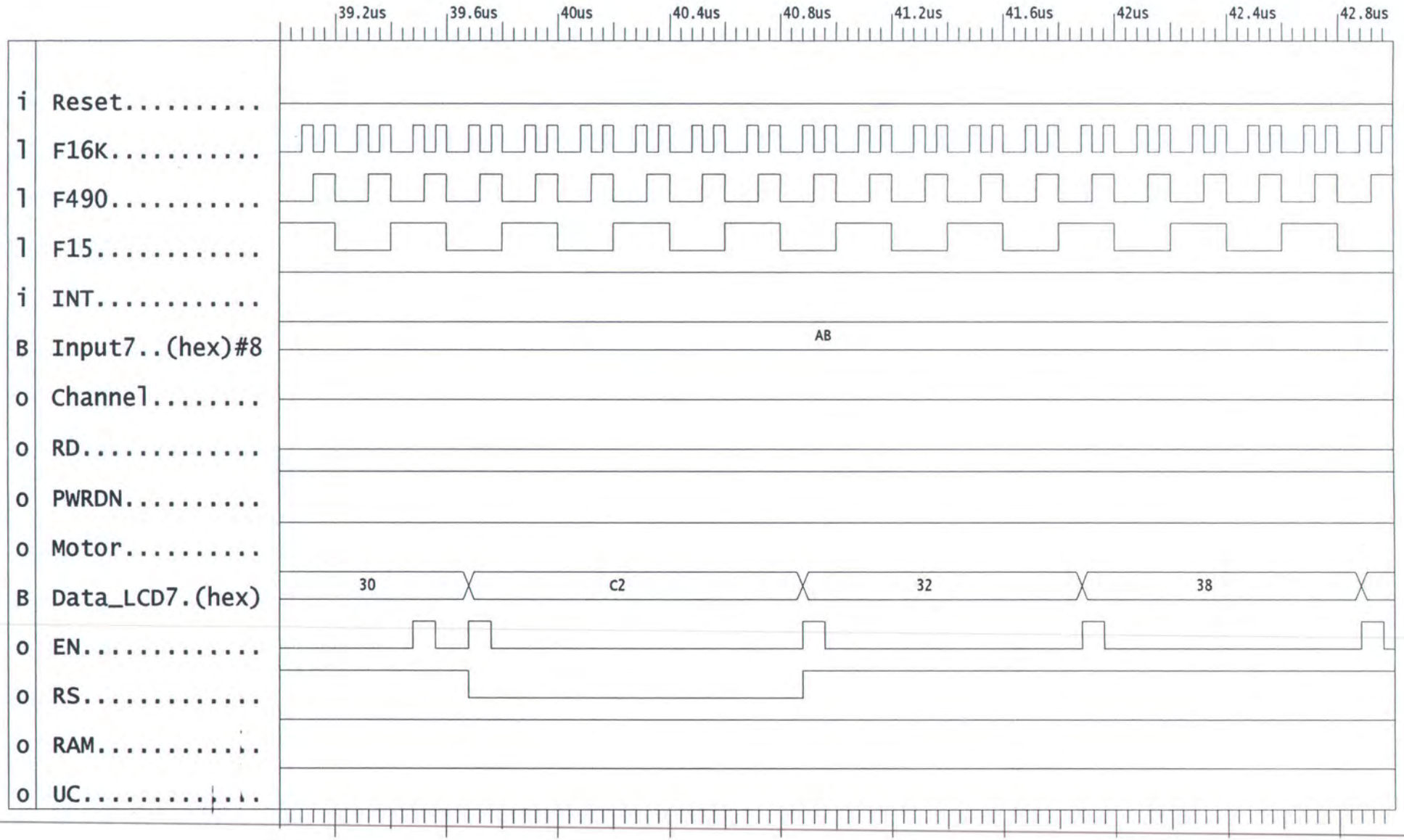


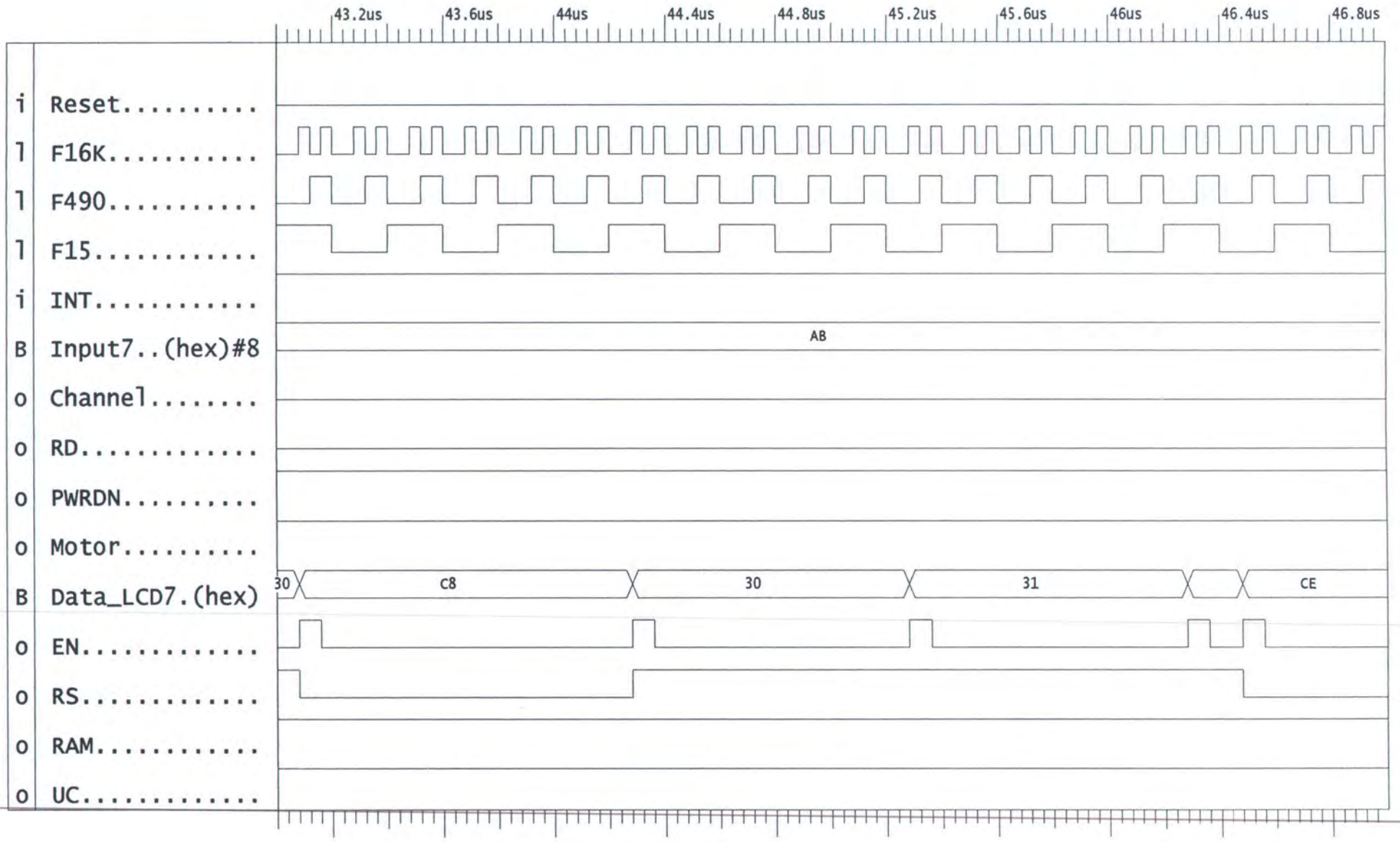

4us
4.4us
4.8us
5.2us
5.6us
6us
6.4us
6.8us
7.2us
7.6us
i Reset.....
l F16K.....
l F490.....
l F15.....
i INT.....
B Input7..(hex)#8
AB
o Channel.....
o RD.....
o PWRDN.....
o Motor.....
B Data_LCD7.(hex)
06
0C
53
75
68
75
20
o EN.....
o RS.....
o RAM.....
o UC.....

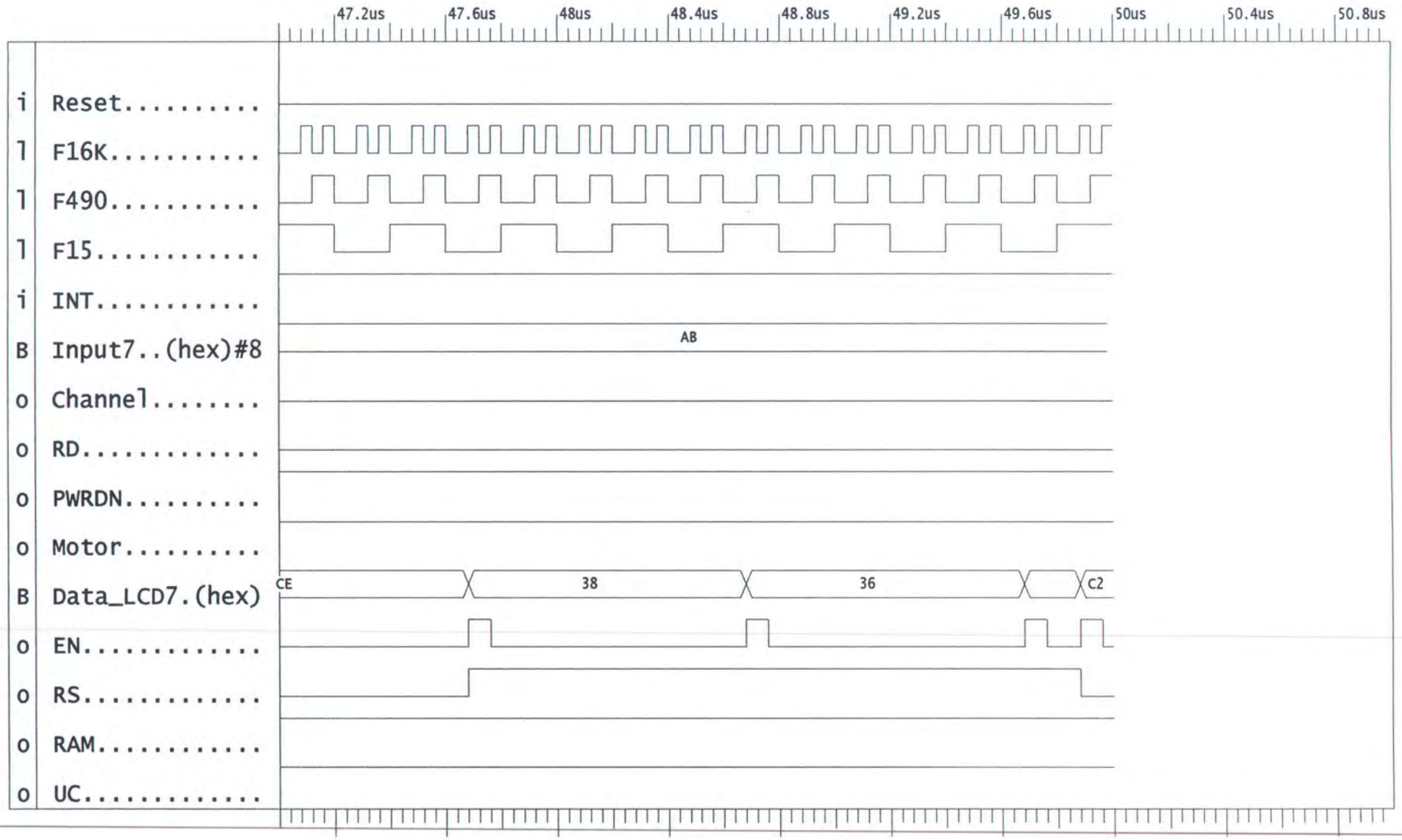

8us
8.4us
8.8us
9.2us
9.6us
10us
10.4us
10.8us
11.2us
11.6us
i Reset.....
l F16K.....
l F490.....
l F15.....
i INT.....
B Input7..(hex)#8
AB
o Channel.....
o RD.....
o PWRDN.....
o Motor.....
B Data_LCD7.(hex)
4C
65
6D
62
61
62
20
o EN.....
o RS.....
o RAM.....
o UC.....

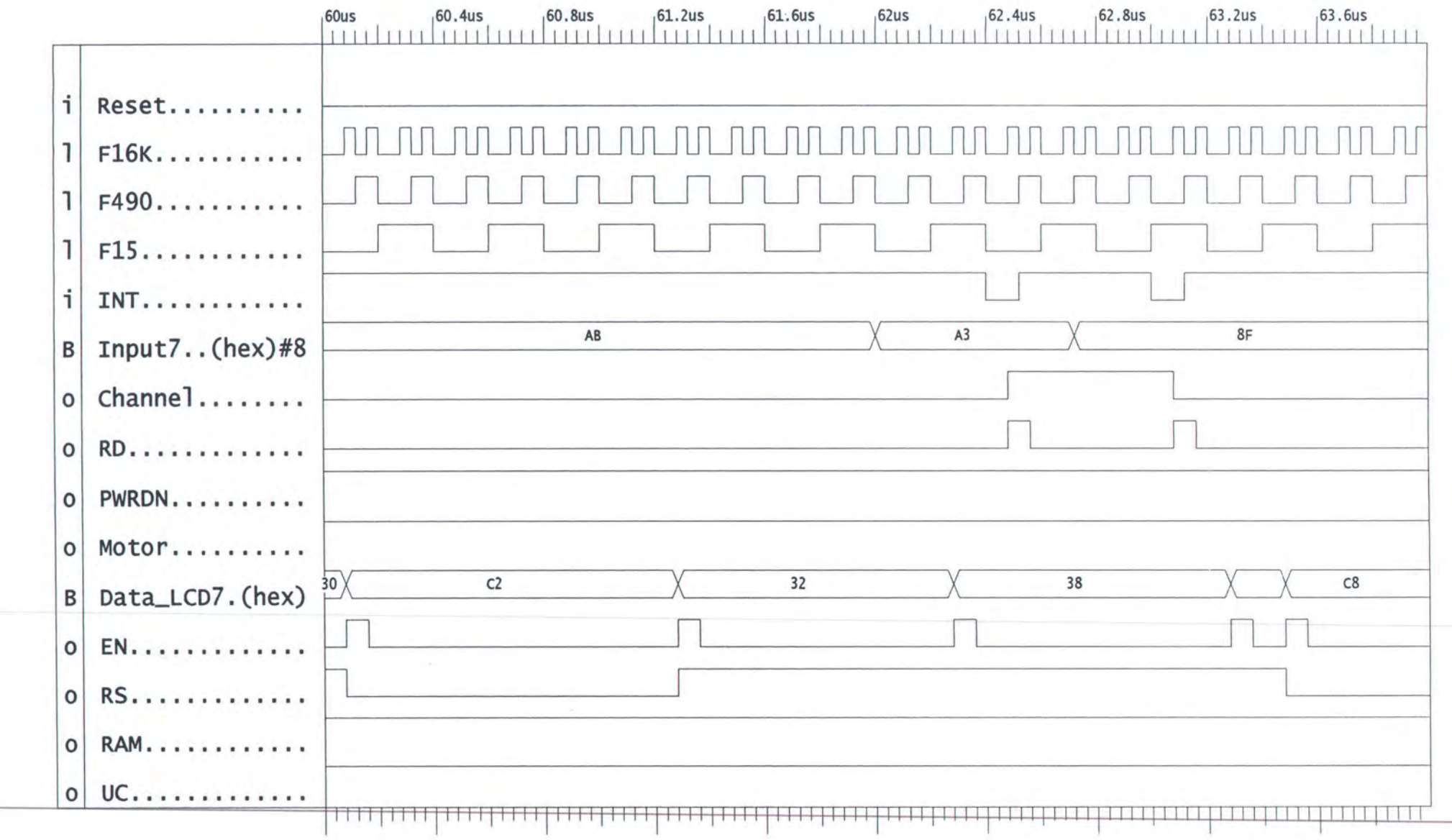

12us
12.4us
12.8us
13.2us
13.6us
14us
14.4us
14.8us
15.2us
15.6us
i Reset.....
l F16K.....
l F490.....
l F15.....
i INT.....
B Input7..(hex)#8
AB
o Channel.....
o RD.....
o PWRDN.....
o Motor.....
B Data_LCD7.(hex)
20
4C
61
6D
61
C3
DF
46
o EN.....
o RS.....
o RAM.....
o UC.....

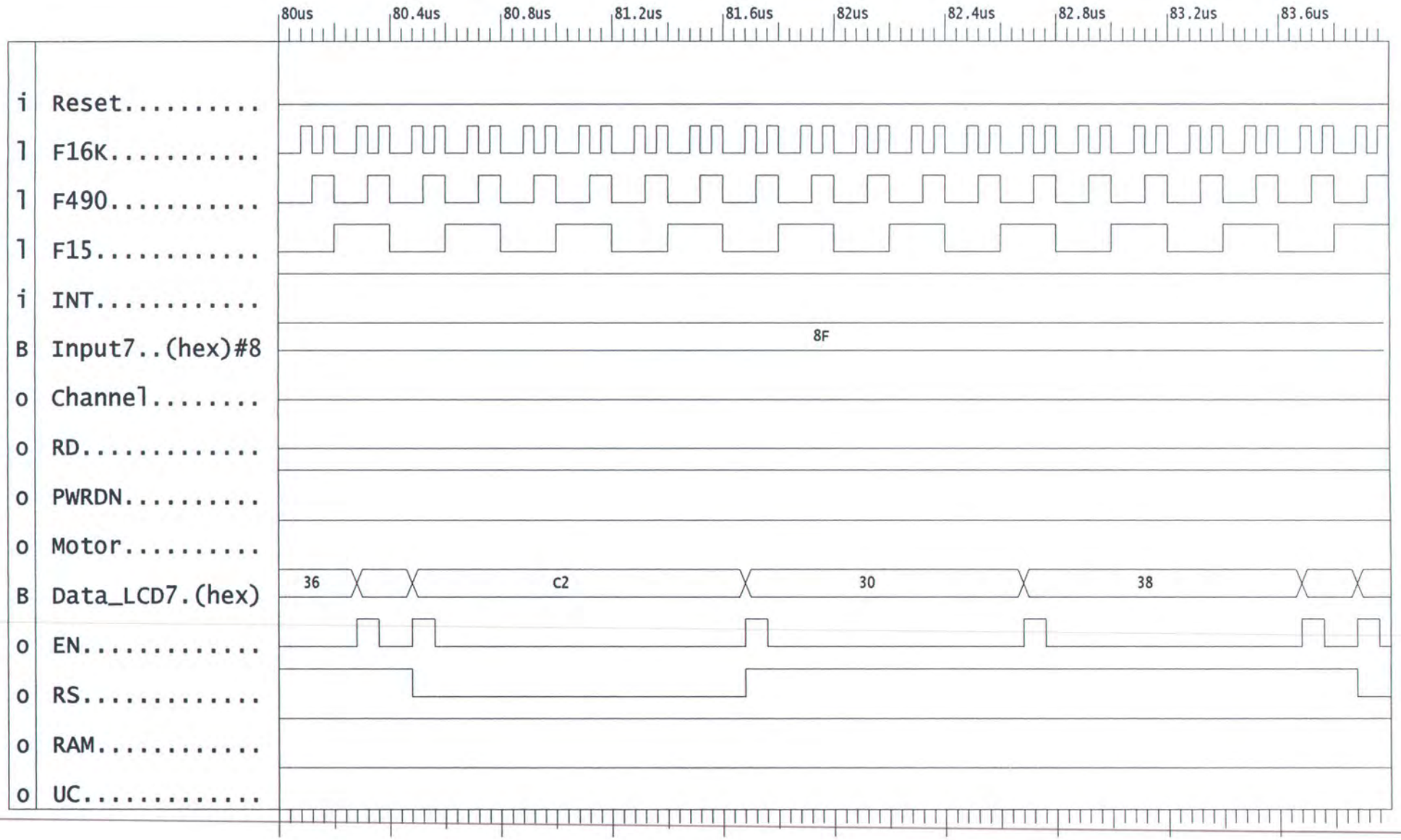

16us
16.4us
16.8us
17.2us
17.6us
18us
18.4us
18.8us
19.2us
19.6us
i Reset.....
l F16K.....
l F490.....
l F15.....
i INT.....
B Input7..(hex)#8
AB
o Channel.....
o RD.....
o PWRDN.....
o Motor.....
B Data_LCD7.(hex)
46
C9
25
CF
27
04
o EN.....
o RS.....
o RAM.....
o UC.....

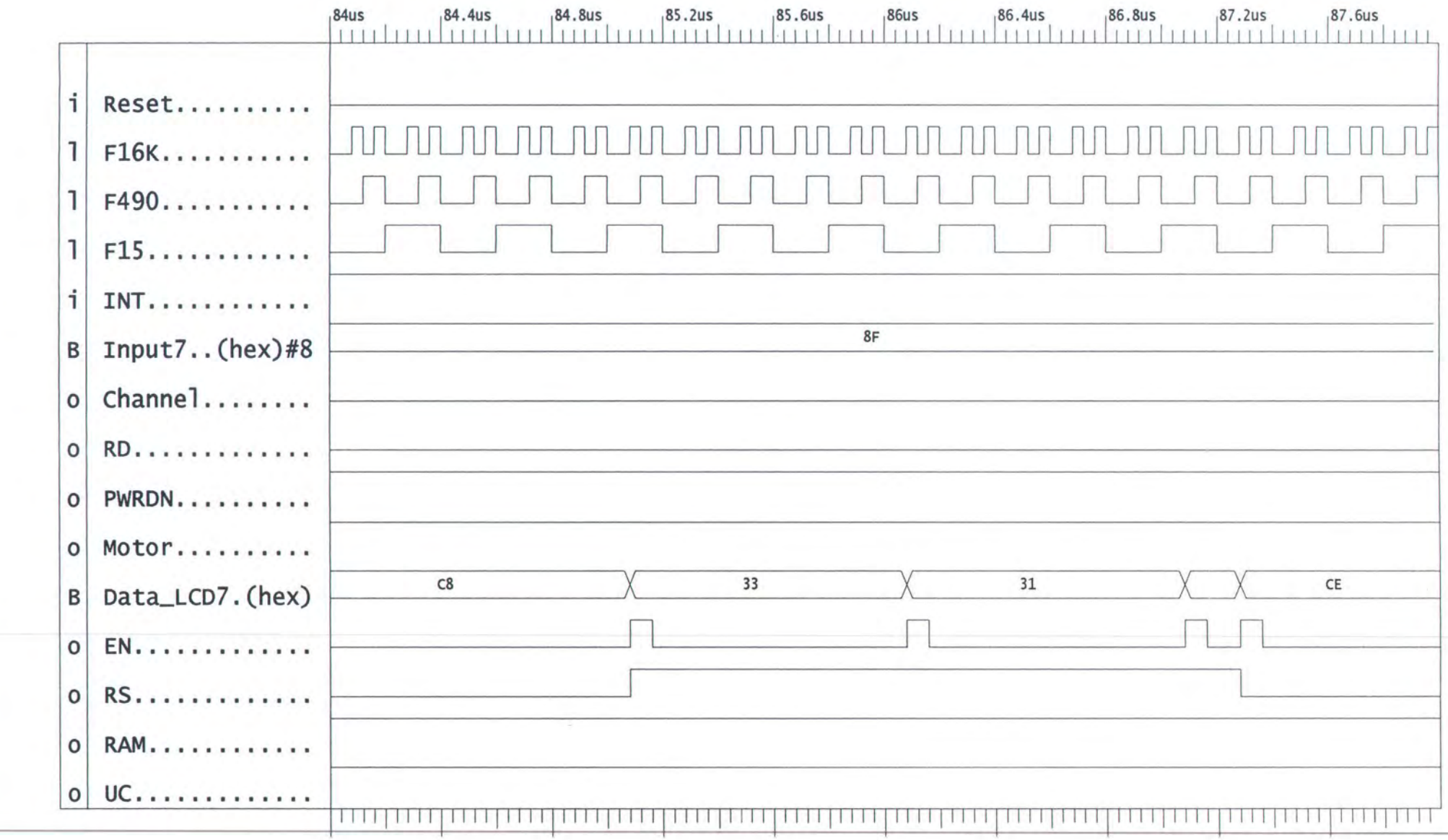

39.2us
39.6us
40us
40.4us
40.8us
41.2us
41.6us
42us
42.4us
42.8us
i Reset.....
l F16K.....
l F490.....
l F15.....
i INT.....
B Input7..(hex)#8
AB
o Channel.....
o RD.....
o PWRDN.....
o Motor.....
B Data_LCD7.(hex)
30
C2
32
38
o EN.....
o RS.....
o RAM.....
o UC.....

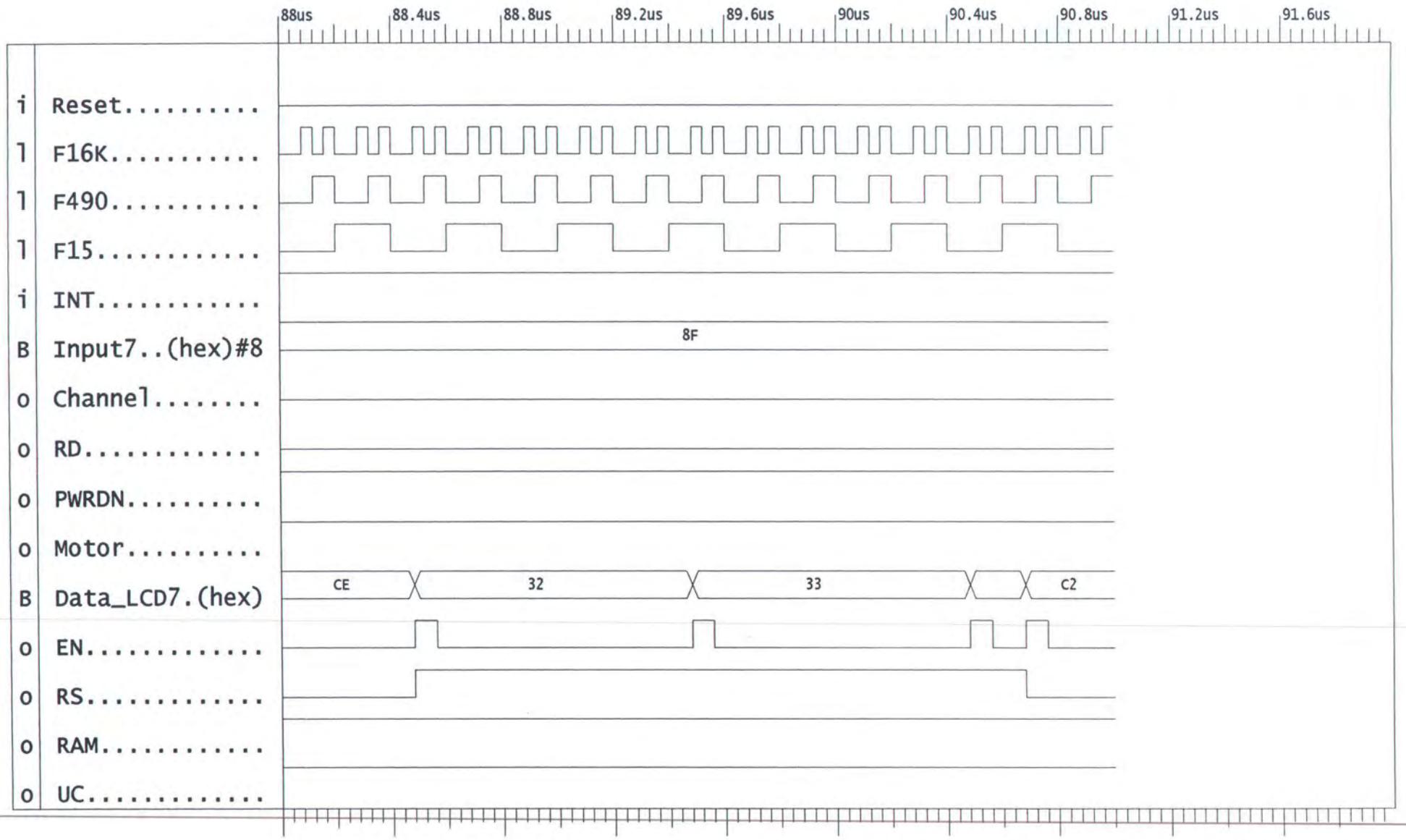

43.2us
43.6us
44us
44.4us
44.8us
45.2us
45.6us
46us
46.4us
46.8us
i Reset.....
l F16K.....
l F490.....
l F15.....
i INT.....
B Input7..(hex)#8
AB
o Channel.....
o RD.....
o PWRDN.....
o Motor.....
B Data_LCD7.(hex)
30
C8
30
31
CE
o EN.....
o RS.....
o RAM.....
o UC.....


47.2us
47.6us
48us
48.4us
48.8us
49.2us
49.6us
50us
50.4us
50.8us
i Reset.....
l F16K.....
l F490.....
l F15.....
i INT.....
B Input7..(hex)#8
AB
o Channel.....
o RD.....
o PWRDN.....
o Motor.....
B Data_LCD7.(hex)
CE
38
36
C2
o EN.....
o RS.....
o RAM.....
o UC.....


60us
60.4us
60.8us
61.2us
61.6us
62us
62.4us
62.8us
63.2us
63.6us
i Reset.....
l F16K.....
l F490.....
l F15.....
i INT.....
B Input7..(hex)#8
AB
A3
8F
o Channel.....
o RD.....
o PWRDN.....
o Motor.....
B Data_LCD7.(hex)
30
C2
32
38
C8
o EN.....
o RS.....
o RAM.....
o UC.....

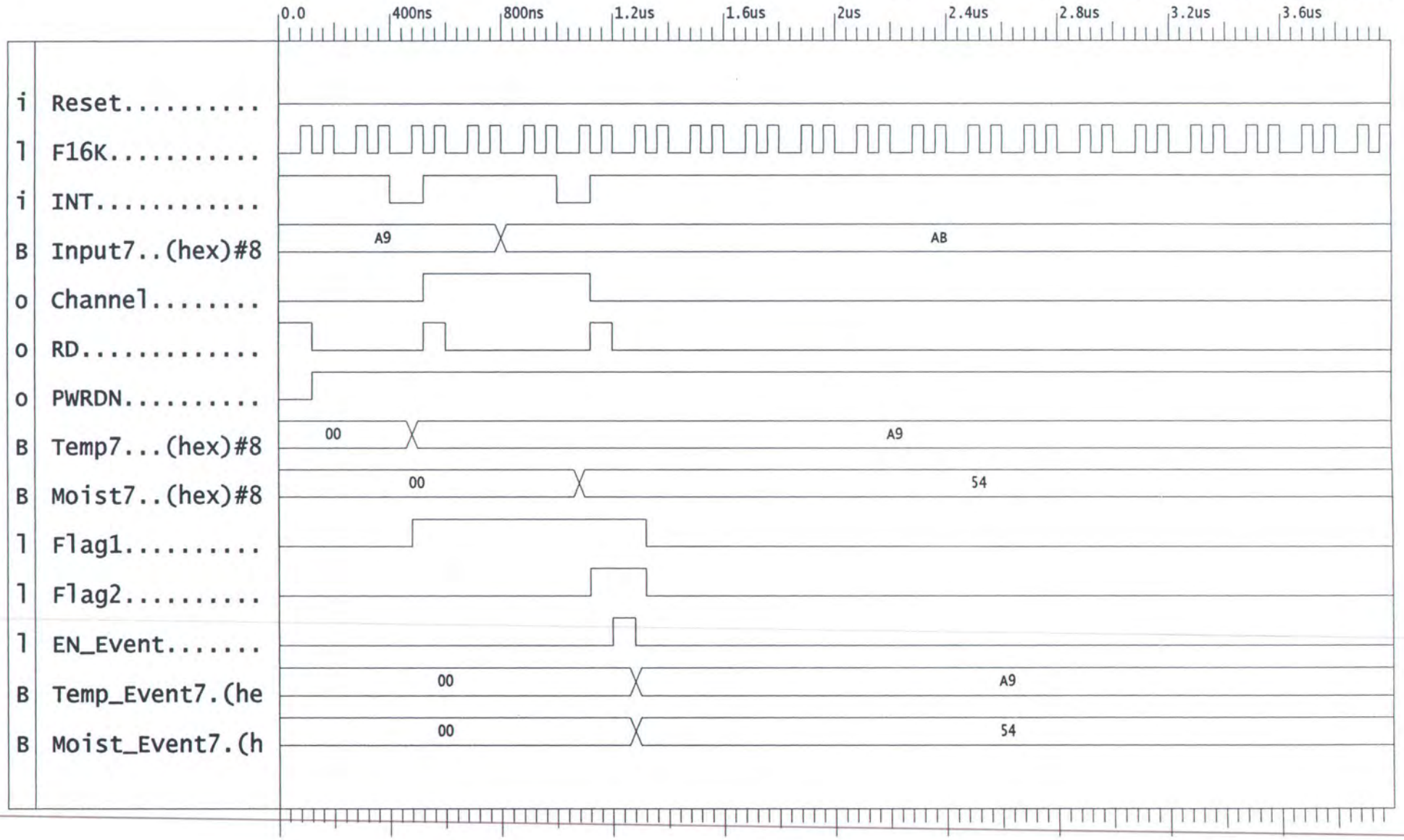

80us
80.4us
80.8us
81.2us
81.6us
82us
82.4us
82.8us
83.2us
83.6us
i Reset.....
l F16K.....
l F490.....
l F15.....
i INT.....
B Input7..(hex)#8
8F
o Channel.....
o RD.....
o PWRDN.....
o Motor.....
B Data_LCD7.(hex)
36
C2
30
38
o EN.....
o RS.....
o RAM.....
o UC.....

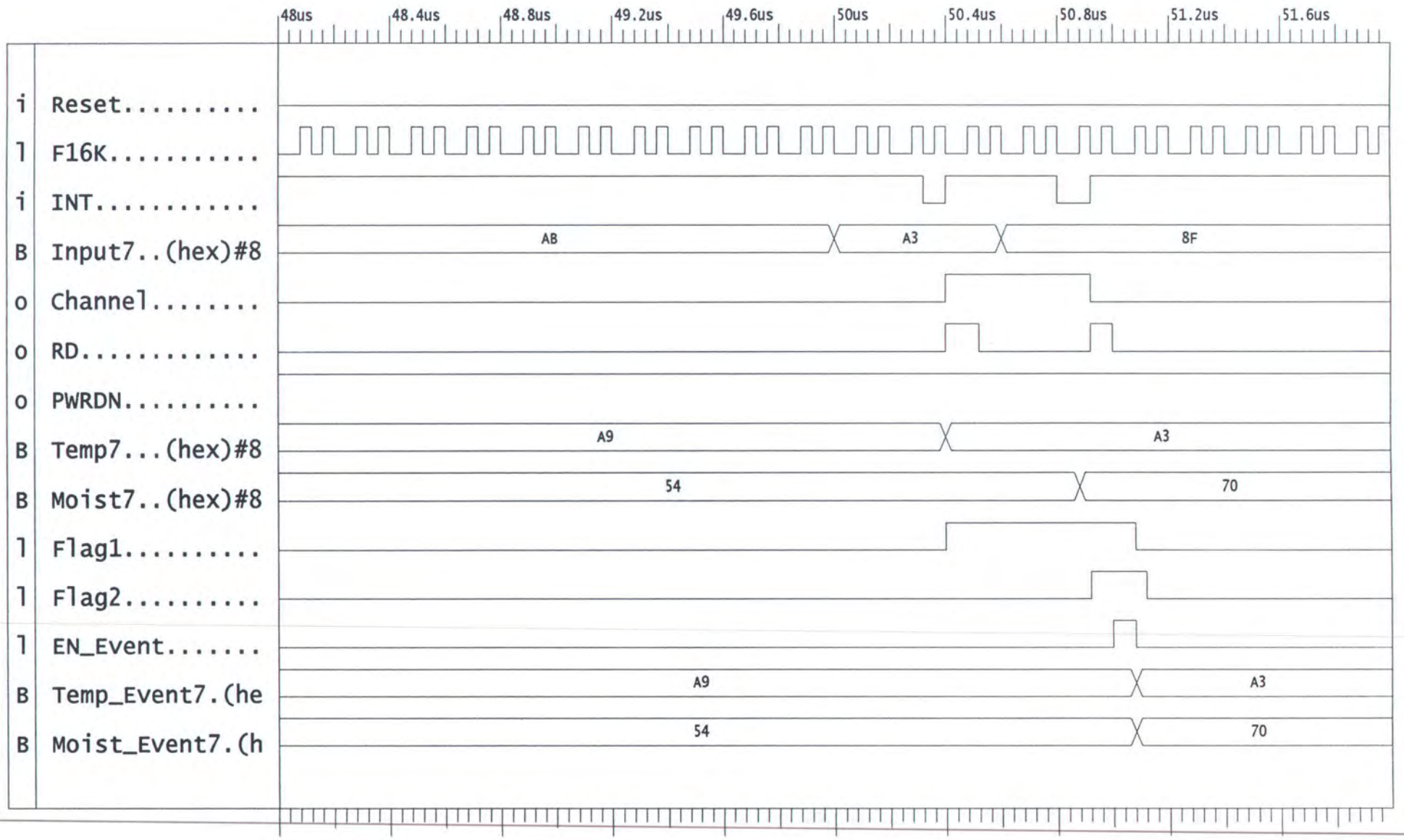

84us
84.4us
84.8us
85.2us
85.6us
86us
86.4us
86.8us
87.2us
87.6us
i Reset.....
l F16K.....
l F490.....
l F15.....
i INT.....
B Input7..(hex)#8
8F
o Channel.....
o RD.....
o PWRDN.....
o Motor.....
B Data_LCD7.(hex)
C8
33
31
CE
o EN.....
o RS.....
o RAM.....
o UC.....

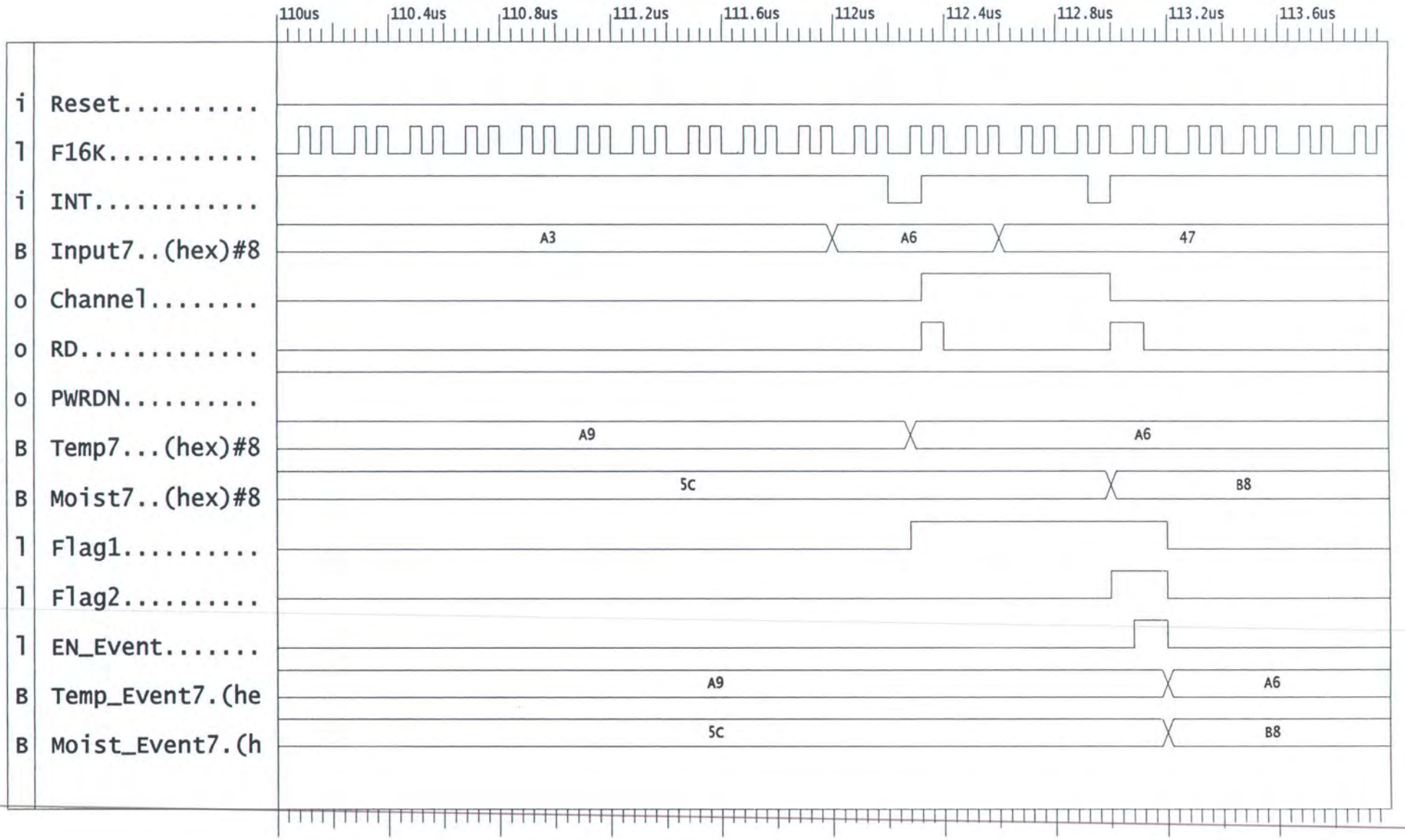

88us
88.4us
88.8us
89.2us
89.6us
90us
90.4us
90.8us
91.2us
91.6us
i Reset.....
l F16K.....
l F490.....
l F15.....
i INT.....
B Input7..(hex)#8
8F
o Channel.....
o RD.....
o PWRDN.....
o Motor.....
B Data_LCD7.(hex)
CE
32
33
C2
o EN.....
o RS.....
o RAM.....
o UC.....

LAMPIRAN D-4

SIMULASI TIMING MODUL AKUISISI DATA

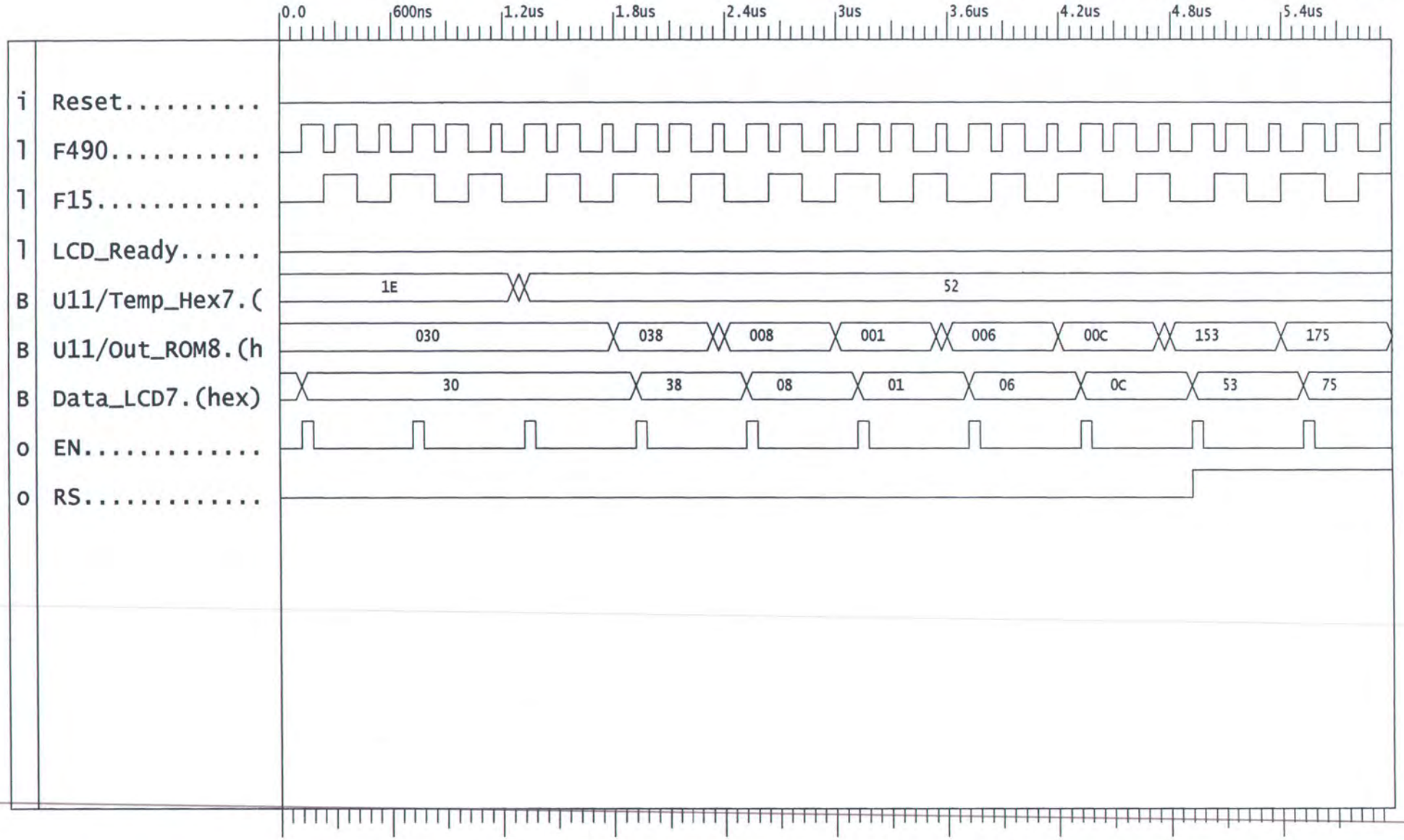


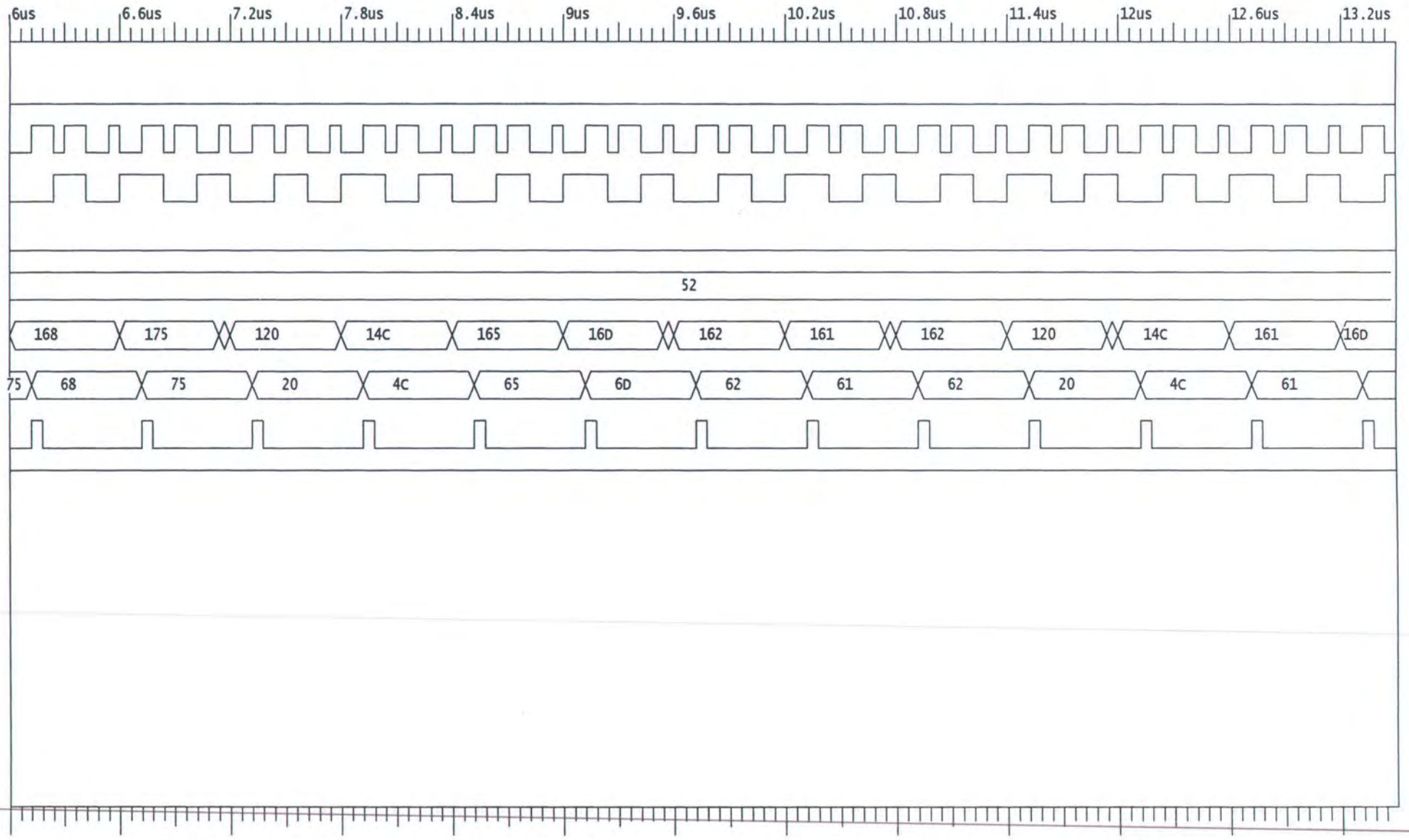

48us
48.4us
48.8us
49.2us
49.6us
50us
50.4us
50.8us
51.2us
51.6us
i Reset.....
l F16K.....
i INT.....
B Input7..(hex)#8
AB
A3
8F
o Channel.....
o RD.....
o PWRDN.....
B Temp7...(hex)#8
A9
A3
B Moist7..(hex)#8
54
70
l Flag1.....
l Flag2.....
l EN_Event.....
B Temp_Event7.(he
A9
A3
B Moist_Event7.(h
54
70

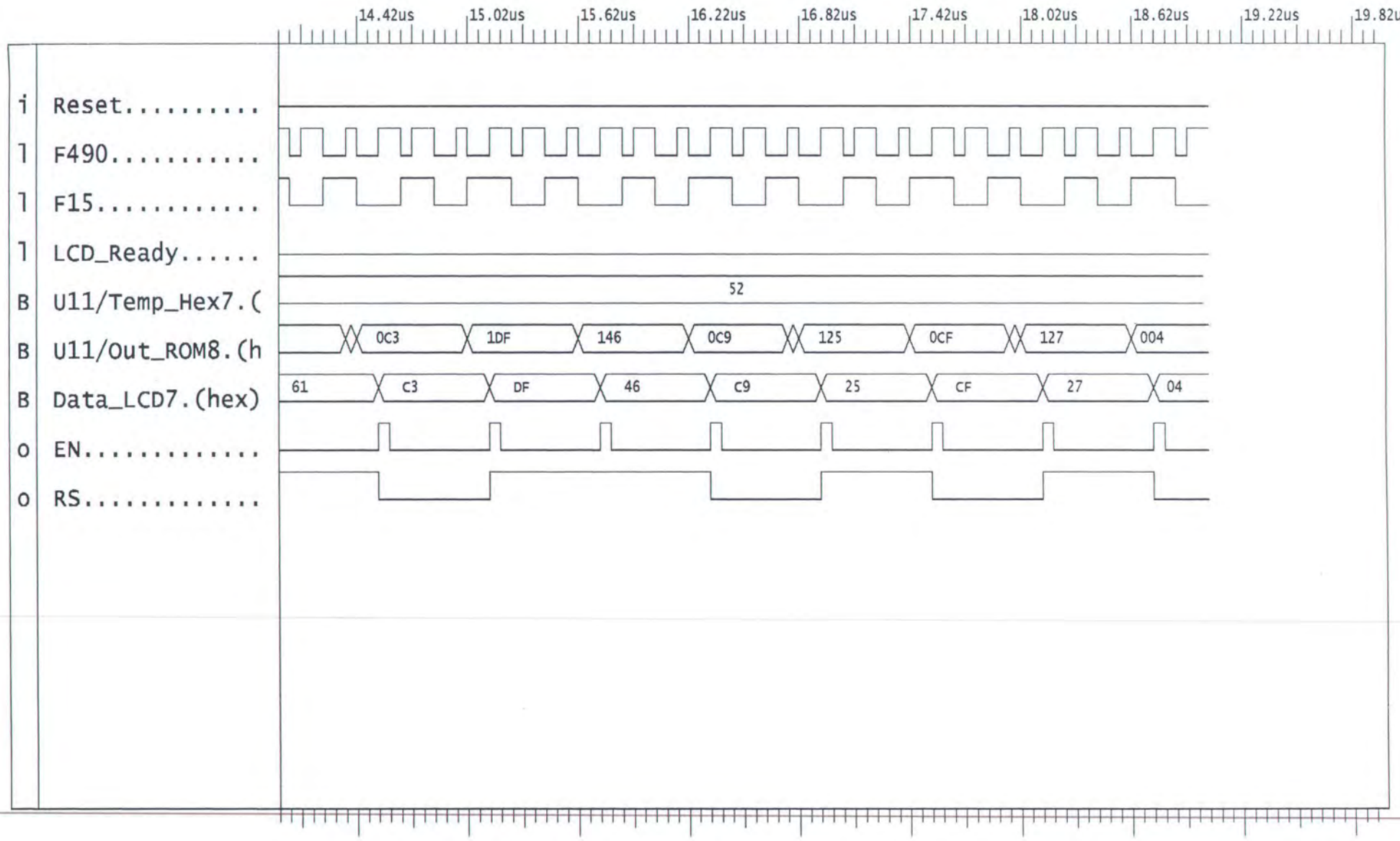

110us
110.4us
110.8us
111.2us
111.6us
112us
112.4us
112.8us
113.2us
113.6us
i Reset.....
l F16K.....
i INT.....
B Input7..(hex)#8
A3
A6
47
o Channel.....
o RD.....
o PWRDN.....
B Temp7...(hex)#8
A9
A6
B Moist7..(hex)#8
5C
B8
l Flag1.....
l Flag2.....
l EN_Event.....
B Temp_Event7.(he
A9
A6
B Moist_Event7.(h
5C
B8

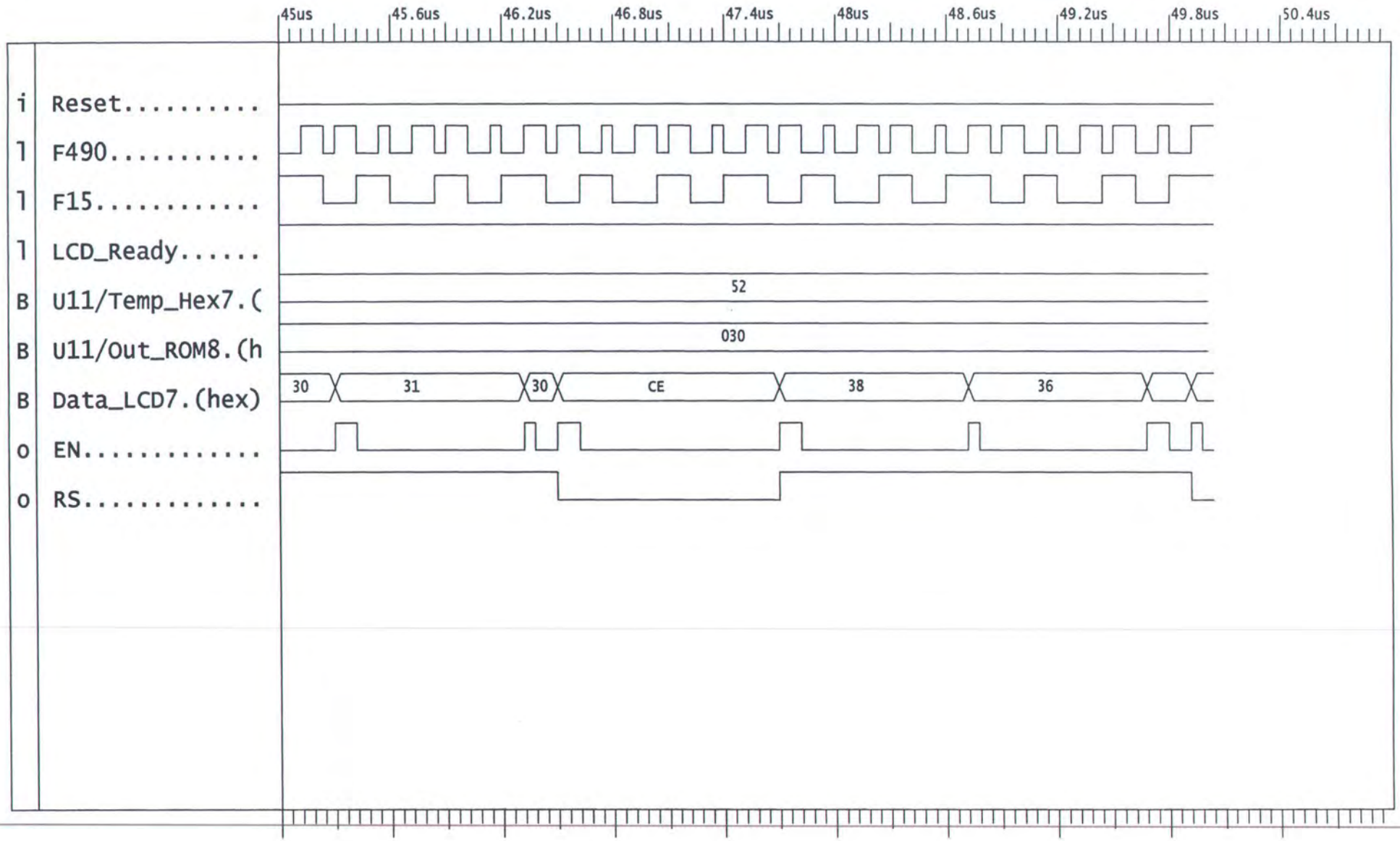
LAMPIRAN D-5

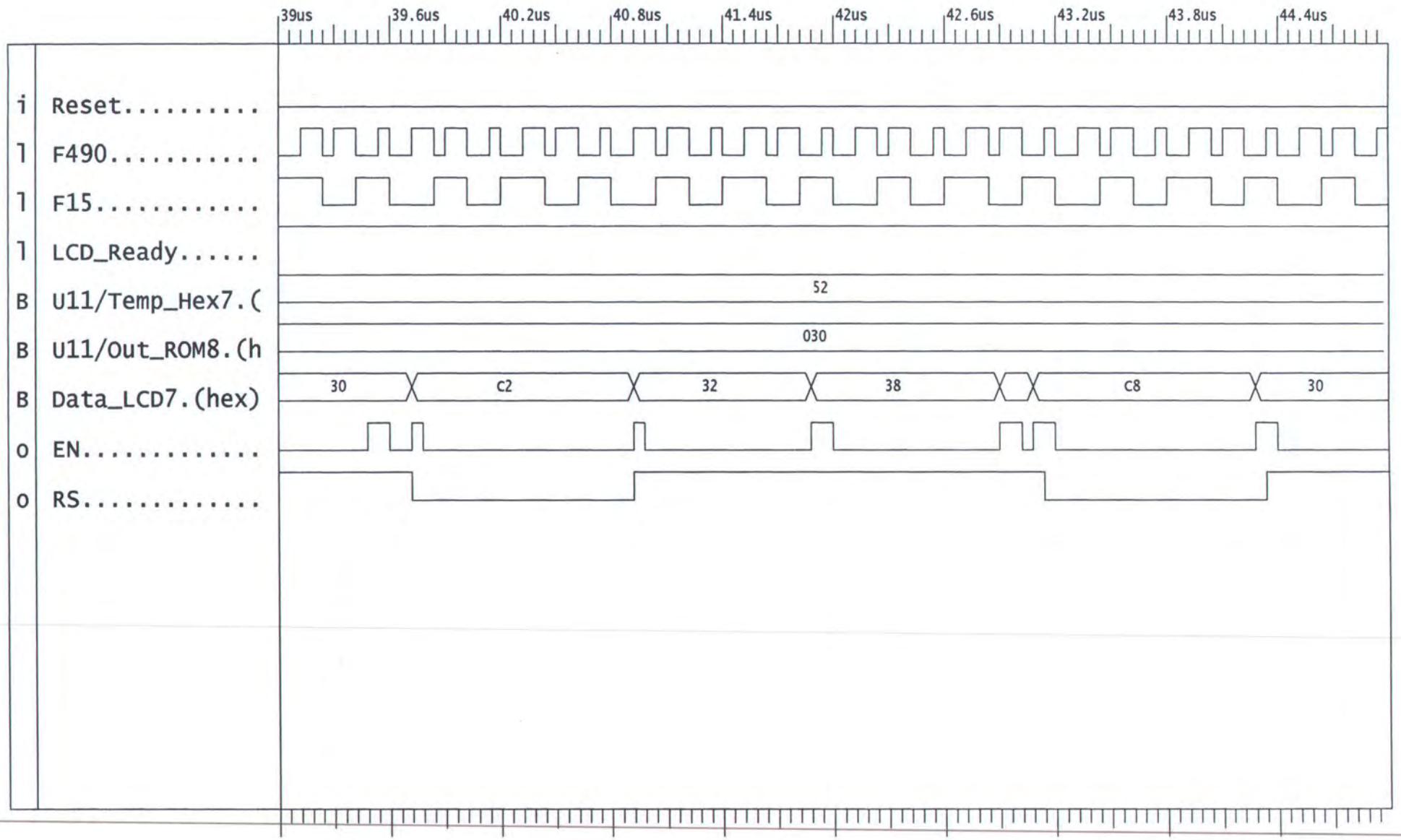
SIMULASI TIMING MODUL DISPLAY

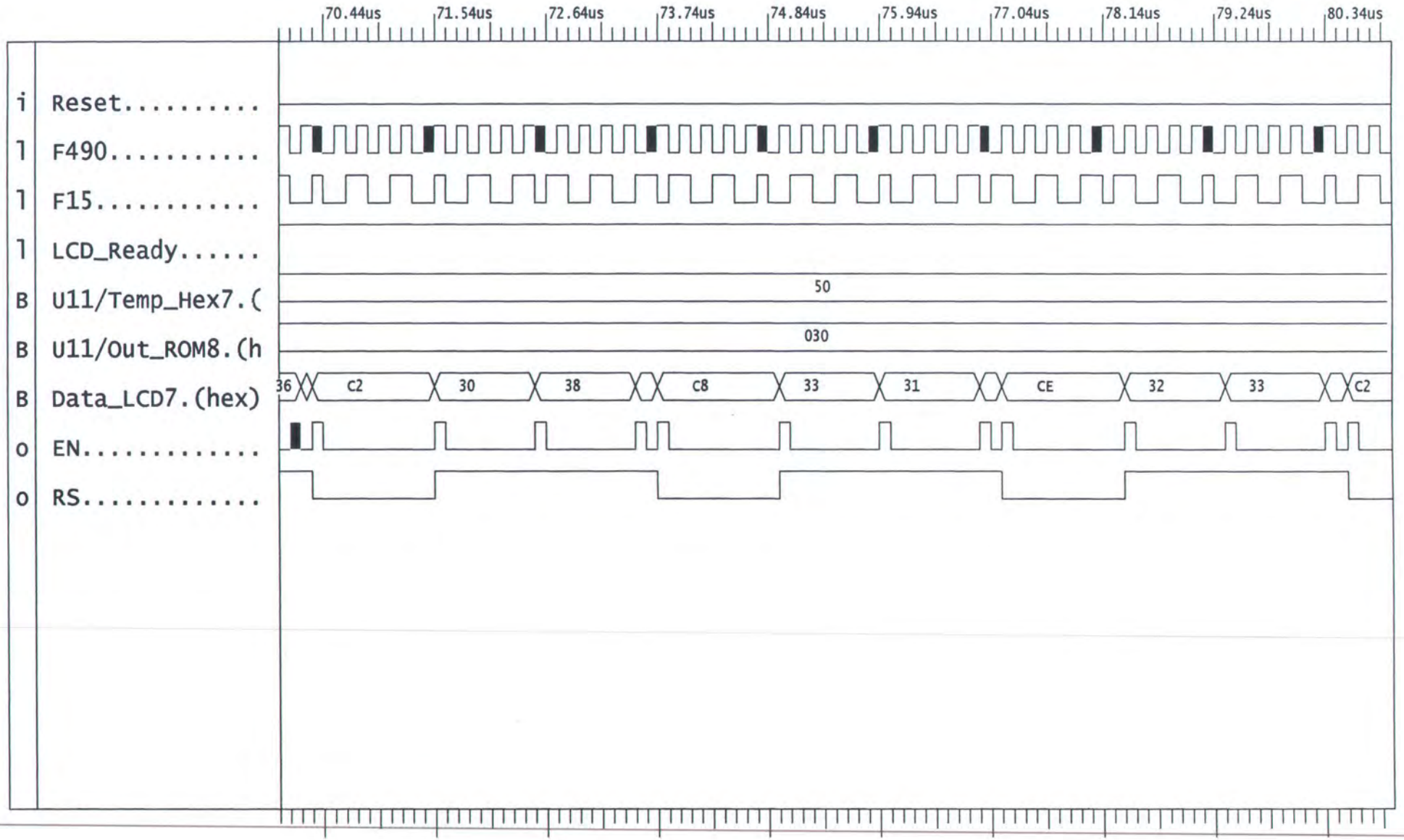



6us
6.6us
7.2us
7.8us
8.4us
9us
9.6us
10.2us
10.8us
11.4us
12us
12.6us
13.2us
52
168
175
120
14C
165
16D
162
161
162
120
14C
161
16D
75
68
75
20
4C
65
6D
62
61
62
20
4C
61


14.42us
15.02us
15.62us
16.22us
16.82us
17.42us
18.02us
18.62us
19.22us
19.82u
i Reset.....
l F490.....
l F15.....
l LCD_Ready.....
B U11/Temp_Hex7.(
52
B U11/Out_ROM8.(h
0C3
1DF
146
0C9
125
0CF
127
004
B Data_LCD7.(hex)
61
C3
DF
46
C9
25
CF
27
04
o EN.....
o RS.....

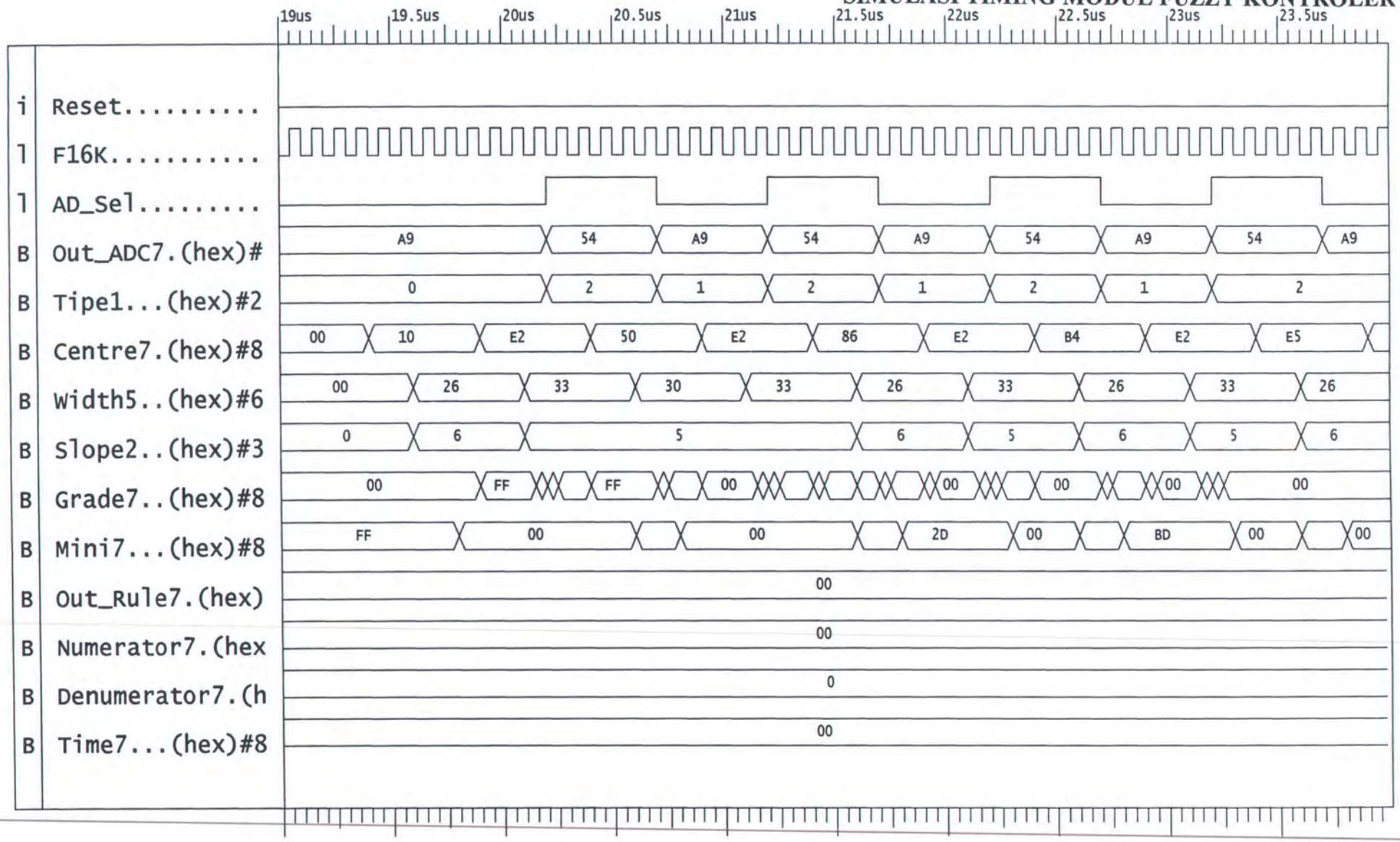

45us
45.6us
46.2us
46.8us
47.4us
48us
48.6us
49.2us
49.8us
50.4us
i Reset.....
l F490.....
l F15.....
l LCD_Ready.....
B U11/Temp_Hex7.(
52
B U11/Out_ROM8.(h
030
B Data_LCD7.(hex)
30
31
30
CE
38
36
o EN.....
o RS.....

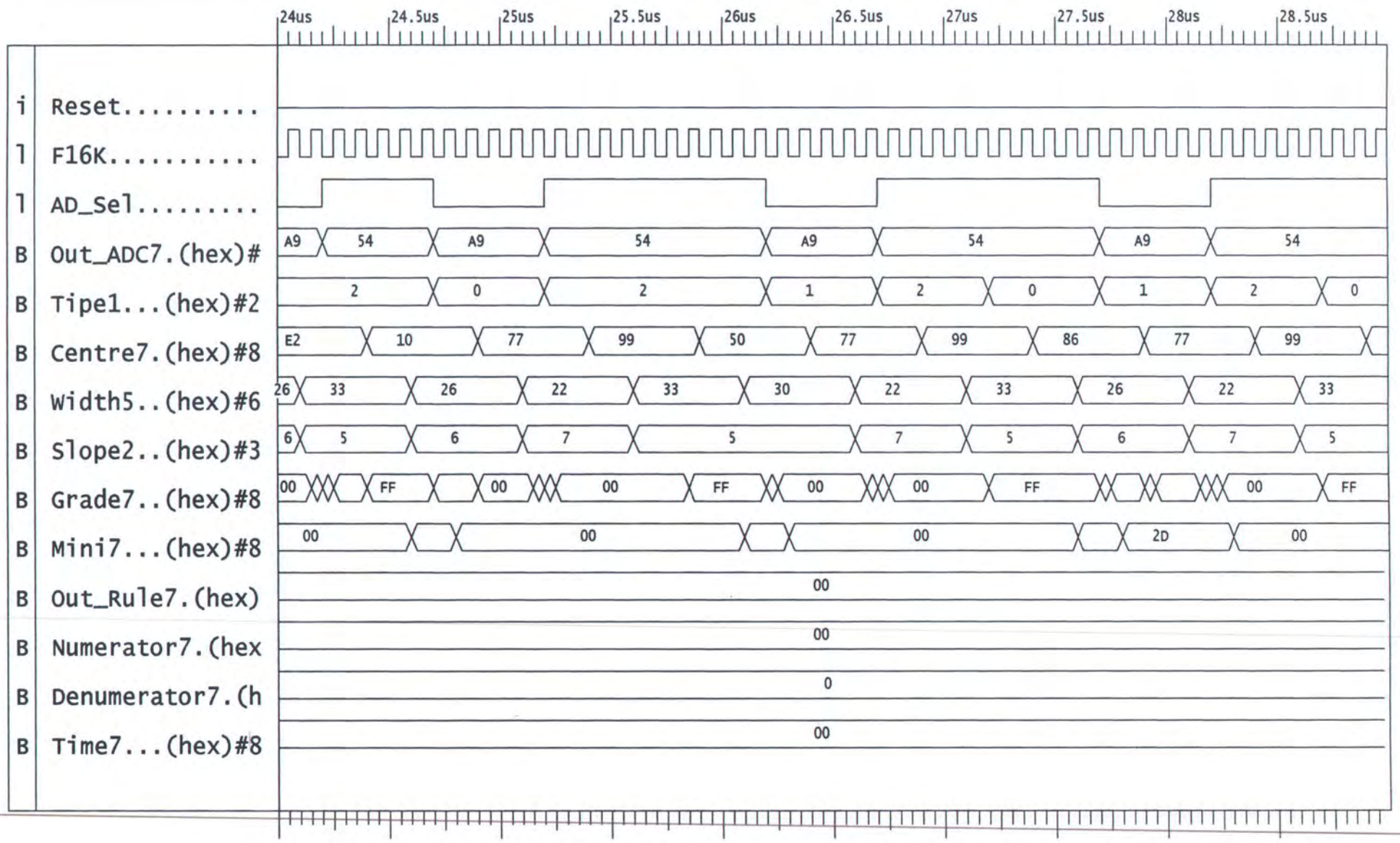

39us
39.6us
40.2us
40.8us
41.4us
42us
42.6us
43.2us
43.8us
44.4us
i Reset.....
l F490.....
l F15.....
l LCD_Ready.....
B U11/Temp_Hex7.(
52
B U11/Out_ROM8.(h
030
B Data_LCD7.(hex)
30
C2
32
38
C8
30
o EN.....
o RS.....

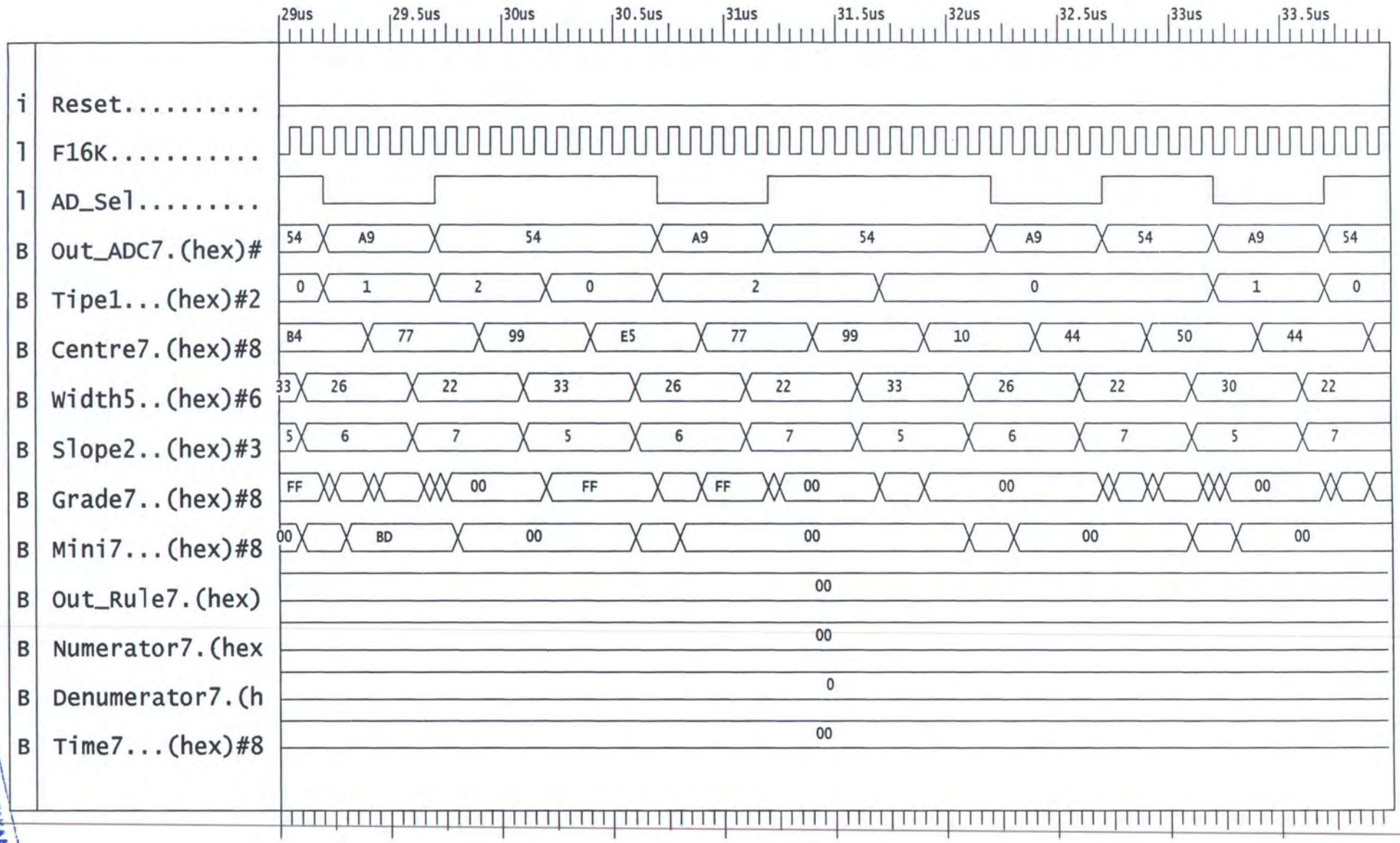

70.44us
71.54us
72.64us
73.74us
74.84us
75.94us
77.04us
78.14us
79.24us
80.34us
i Reset.....
l F490.....
l F15.....
l LCD_Ready.....
B U11/Temp_Hex7.(
B U11/Out_ROM8.(h
B Data_LCD7.(hex)
o EN.....
o RS.....
50
030
36
C2
30
38
C8
33
31
CE
32
33
C2

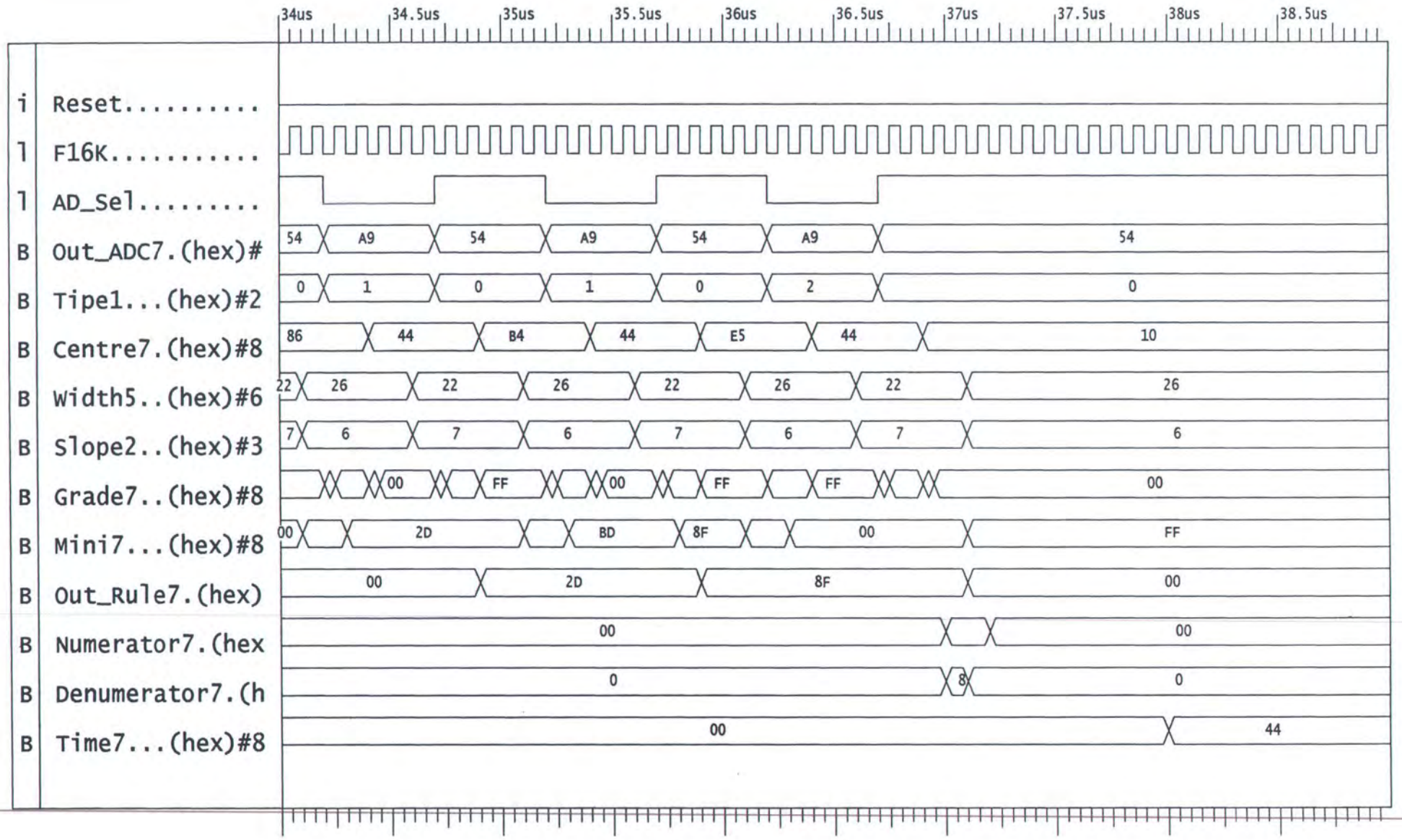
LAMPIRAN D-6

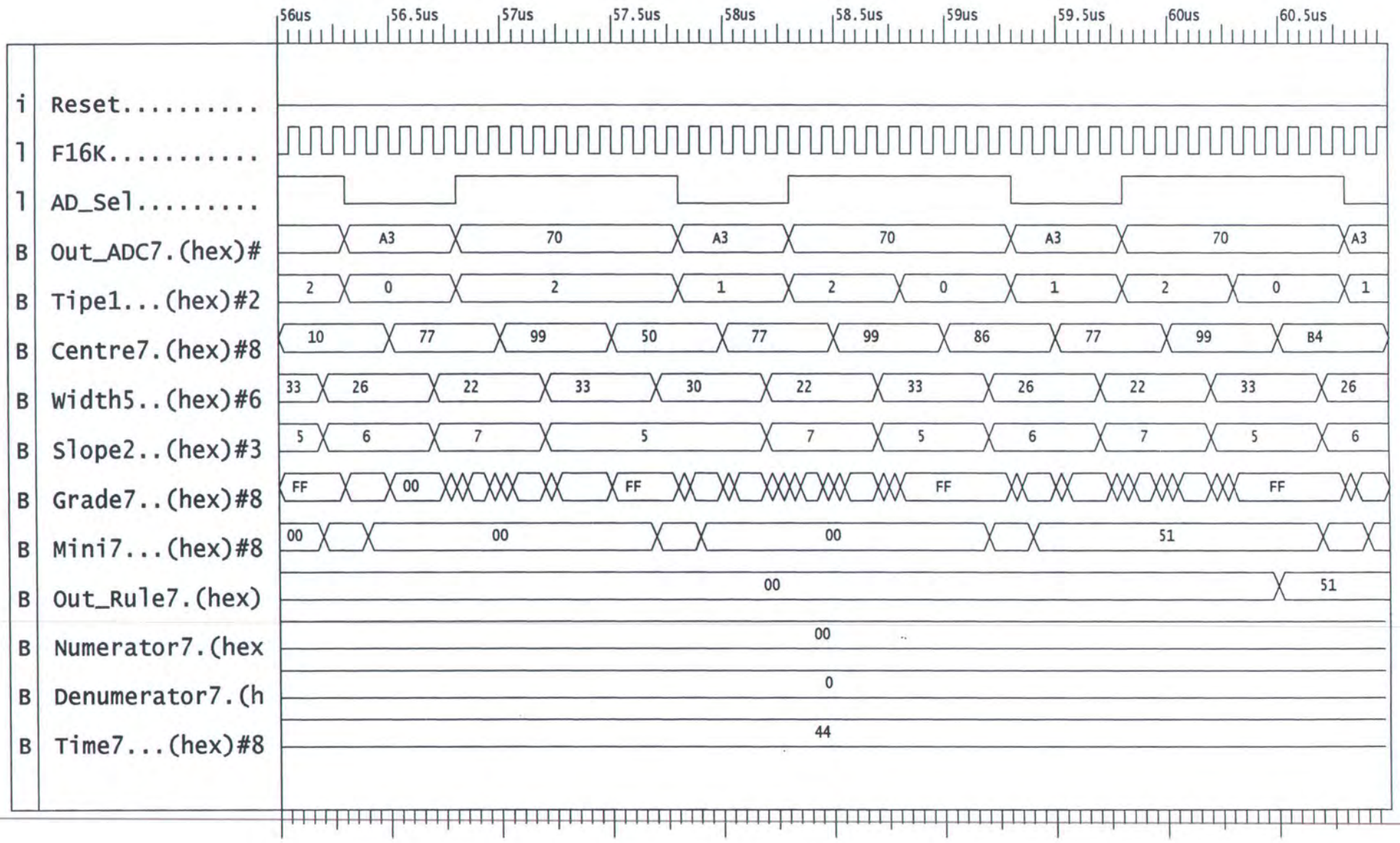
SIMULASI TIMING MODUL FUZZY KONTROLER

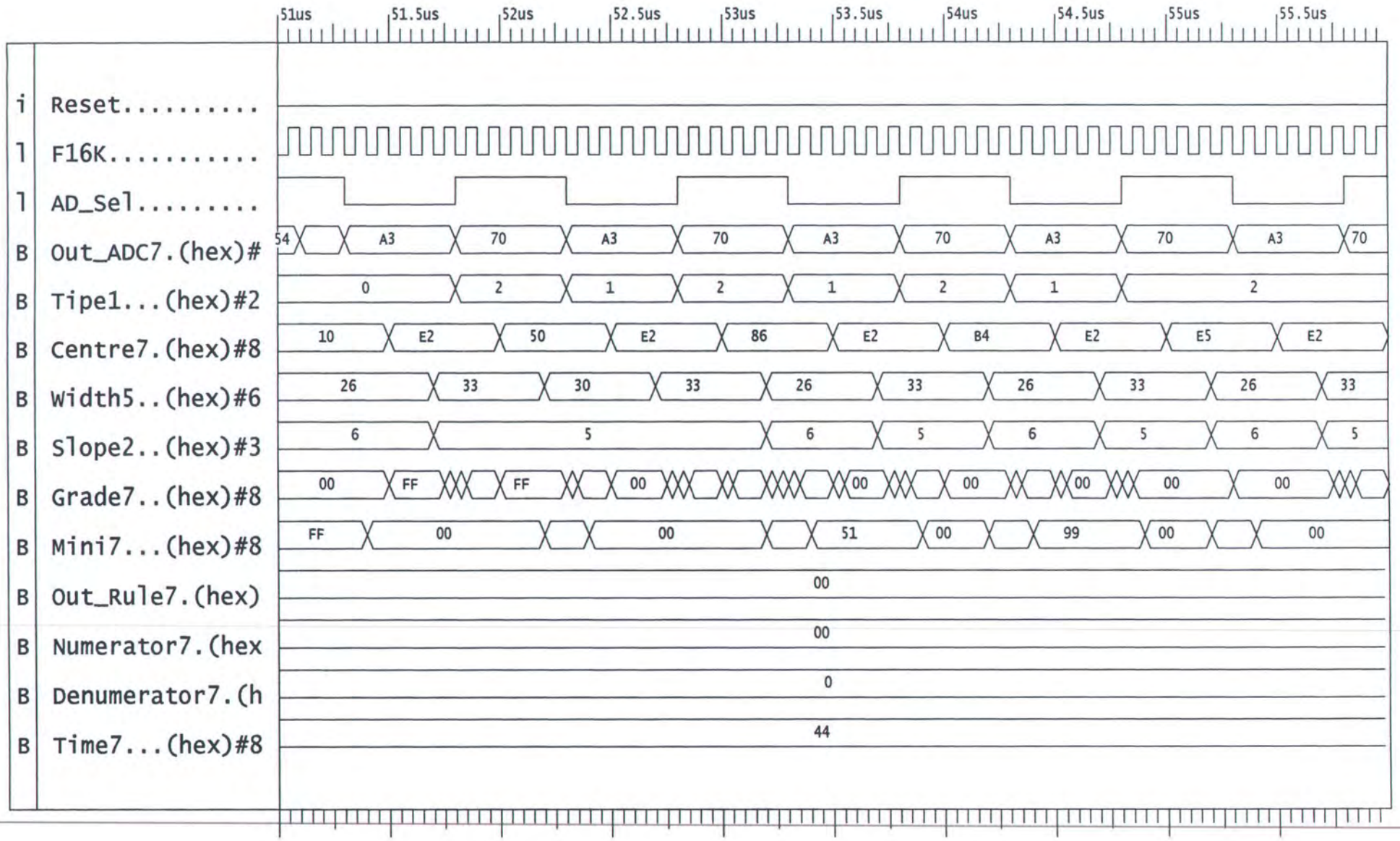


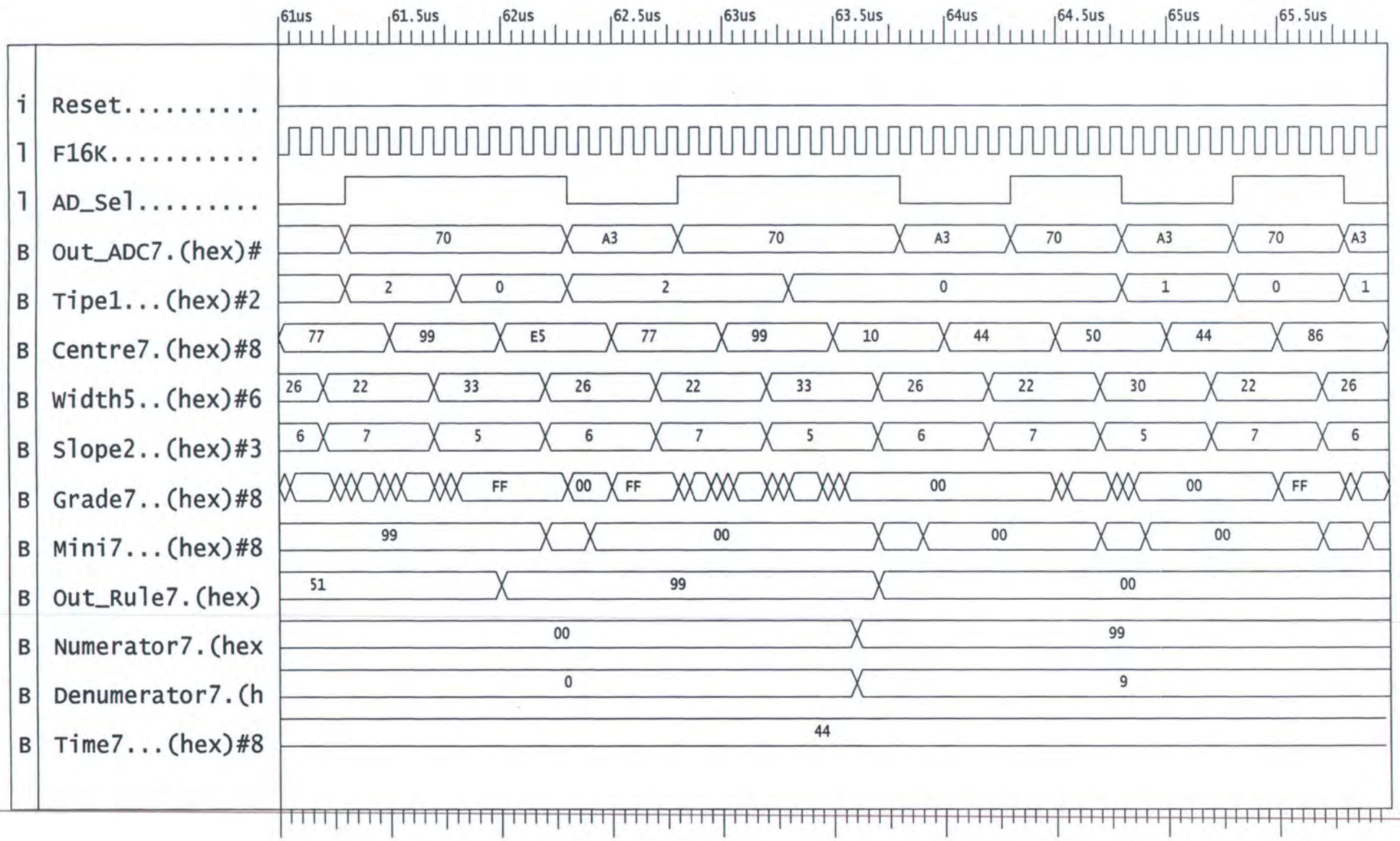

24us
24.5us
25us
25.5us
26us
26.5us
27us
27.5us
28us
28.5us
i Reset.....
l F16K.....
l AD_Sel.....
B Out_ADC7.(hex)#
A9 54 A9 54 A9 54 A9 54
B Tipe1...(hex)#2
2 0 2 1 2 0 1 2 0
B Centre7.(hex)#8
E2 10 77 99 50 77 99 86 77 99
B Width5..(hex)#6
26 33 26 22 33 30 22 33 26 22 33
B Slope2..(hex)#3
6 5 6 7 5 7 5 6 7 5
B Grade7..(hex)#8
00 FF 00 00 FF 00 00 FF 00 FF
B Mini7...(hex)#8
00 00 00 2D 00
B Out_Rule7.(hex)
00
B Numerator7.(hex
00
B Denumerator7.(h
0
B Time7...(hex)#8
00

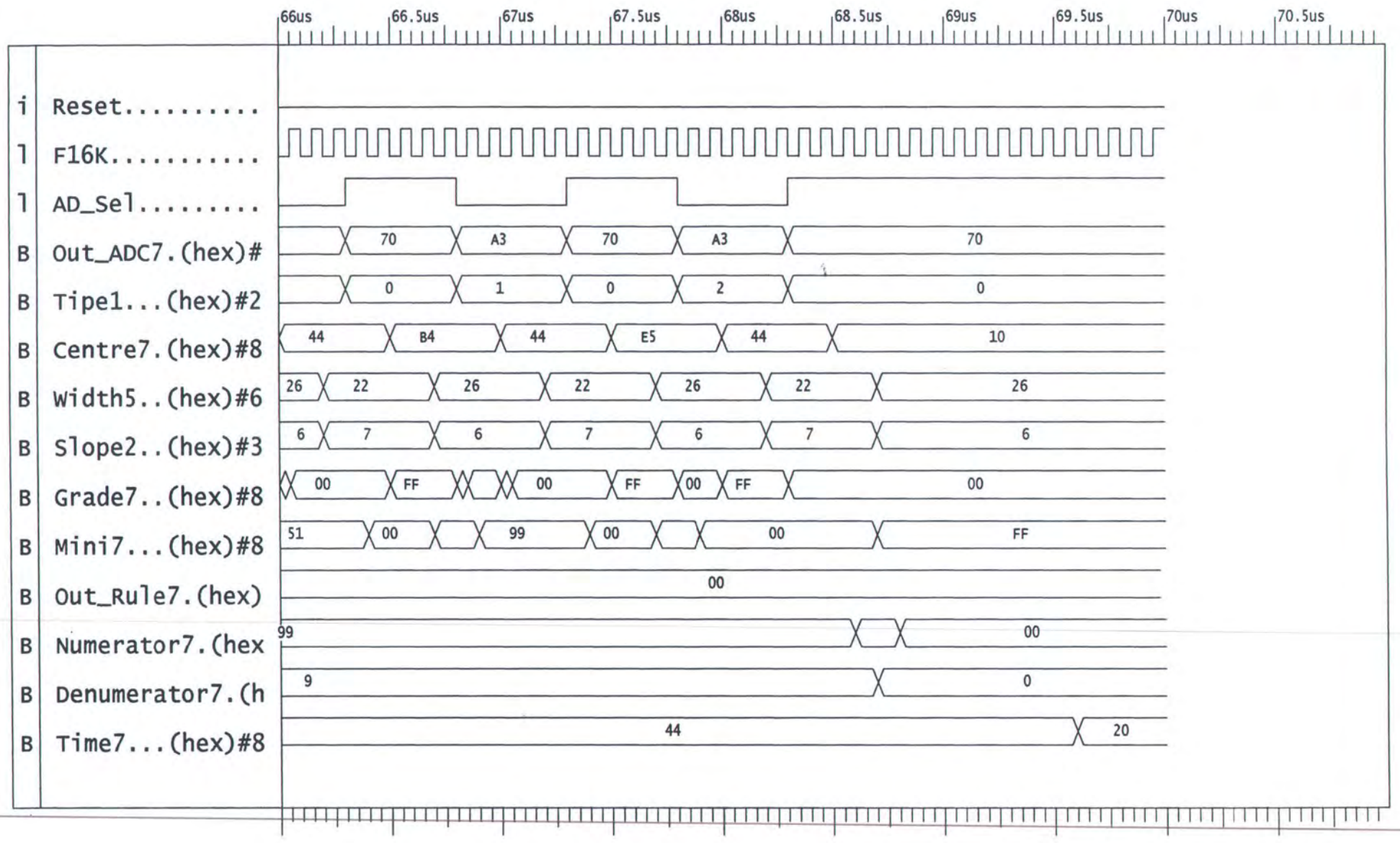


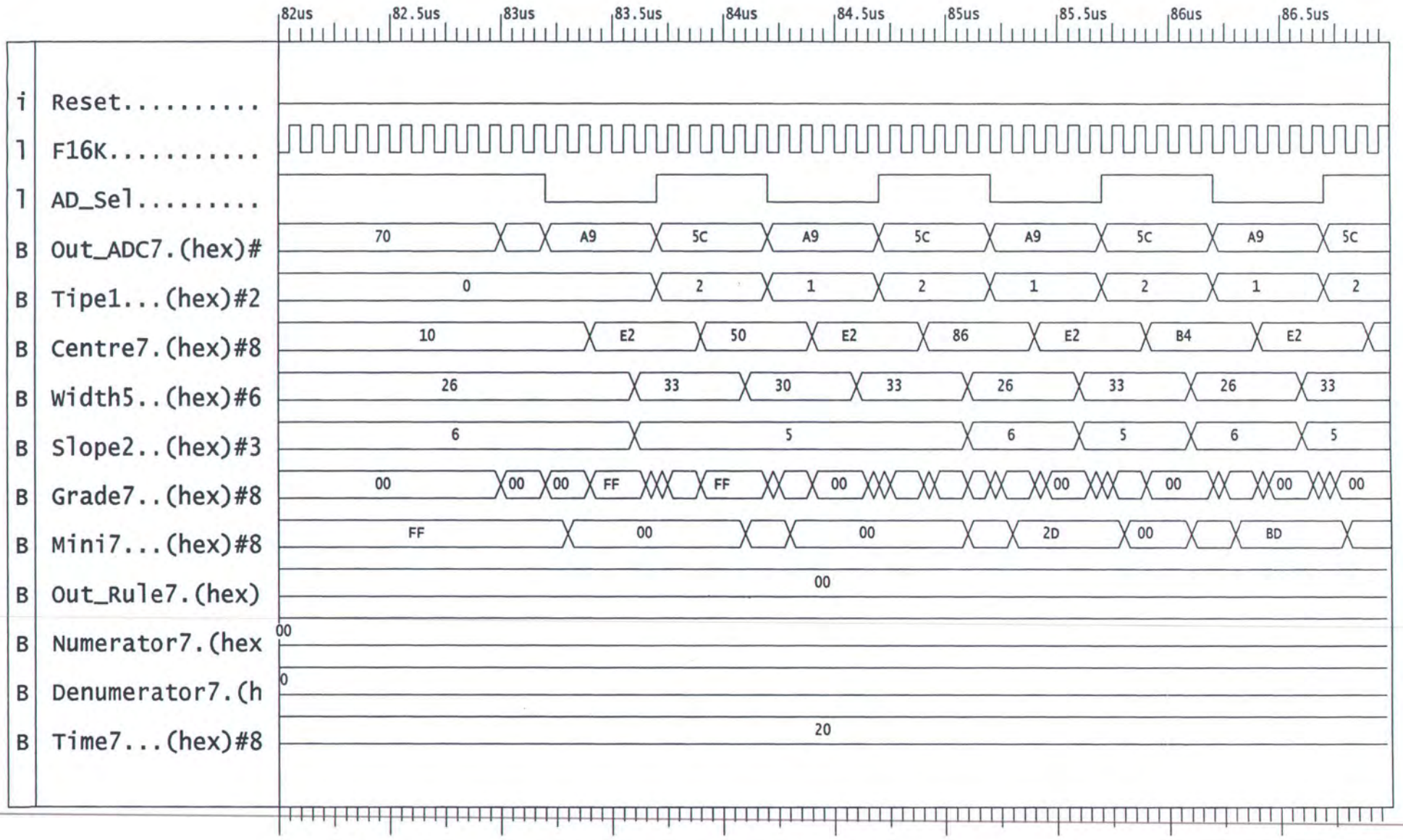

34us
34.5us
35us
35.5us
36us
36.5us
37us
37.5us
38us
38.5us
i Reset.....
l F16K.....
l AD_Sel.....
B Out_ADC7.(hex)#
B Tipe1...(hex)#2
B Centre7.(hex)#8
B Width5..(hex)#6
B Slope2..(hex)#3
B Grade7..(hex)#8
B Mini7...(hex)#8
B Out_Rule7.(hex)
B Numerator7.(hex
B Denumerator7.(h
B Time7...(hex)#8
54 A9 54 A9 54 A9 54
0 1 0 1 0 2 0
86 44 B4 44 E5 44 10
22 26 22 26 22 26 22 26
7 6 7 6 7 6 7 6
00 FF 00 FF FF 00
00 2D BD 8F 00 FF
00 2D 8F 00
00 00
0 8 0
00 44

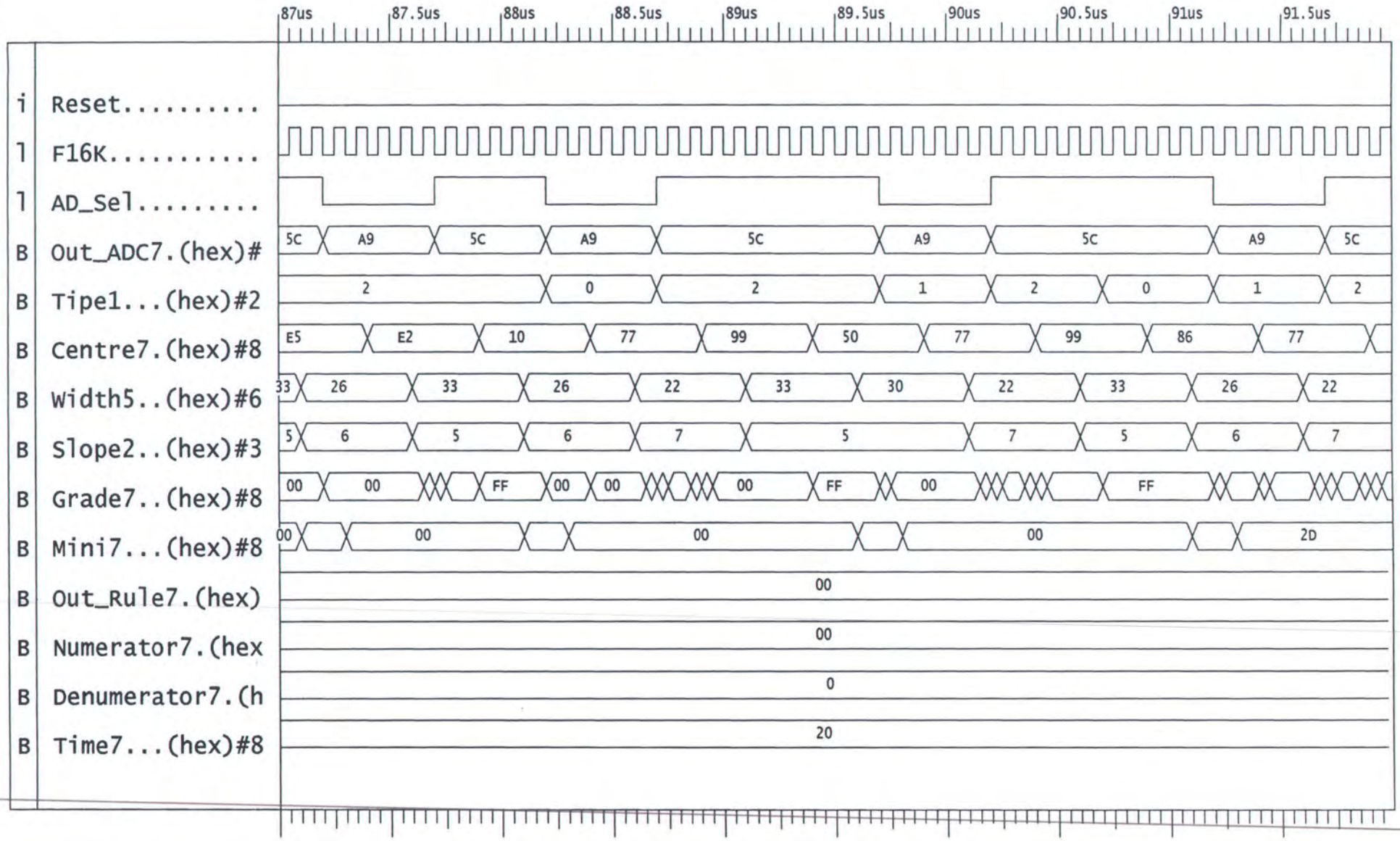

56us
56.5us
57us
57.5us
58us
58.5us
59us
59.5us
60us
60.5us
i Reset.....
l F16K.....
l AD_Sel.....
B Out_ADC7.(hex)#
B Tipe1...(hex)#2
B Centre7.(hex)#8
B Width5..(hex)#6
B Slope2..(hex)#3
B Grade7..(hex)#8
B Mini7...(hex)#8
B Out_Rule7.(hex)
B Numerator7.(hex
B Denumerator7.(h
B Time7...(hex)#8
A3 70 A3 70 A3 70 A3
2 0 2 1 2 0 1 2 0 1
10 77 99 50 77 99 86 77 99 B4
33 26 22 33 30 22 33 26 22 33 26
5 6 7 5 7 5 6 7 5 6
FF 00 FF FF FF
00 00 00 51
00 51
00
0
44

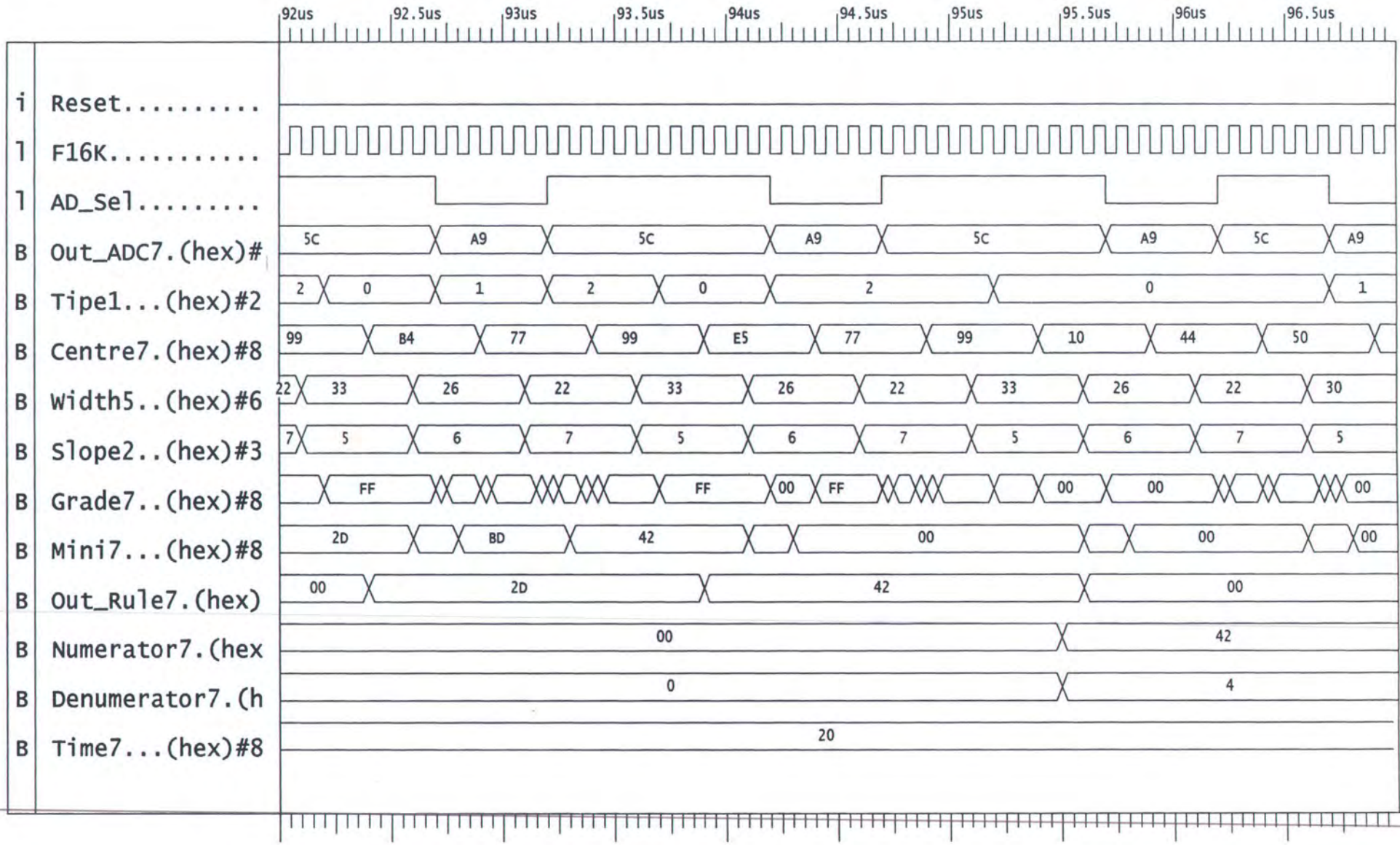

51us
51.5us
52us
52.5us
53us
53.5us
54us
54.5us
55us
55.5us
i Reset.....
l F16K.....
l AD_Sel.....
B Out_ADC7.(hex)#
B Tipe1...(hex)#2
B Centre7.(hex)#8
B Width5..(hex)#6
B Slope2..(hex)#3
B Grade7..(hex)#8
B Mini7...(hex)#8
B Out_Rule7.(hex)
B Numerator7.(hex
B Denumerator7.(h
B Time7...(hex)#8
54 A3 70 A3 70 A3 70 A3 70 A3 70
0 2 1 2 1 2 1 2
10 E2 50 E2 86 E2 B4 E2 E5 E2
26 33 30 33 26 33 26 33 26 33
6 5 6 5 6 5 6 5
00 FF FF 00 00 00 00 00 00
FF 00 00 51 00 99 00 00
00
00
0
44

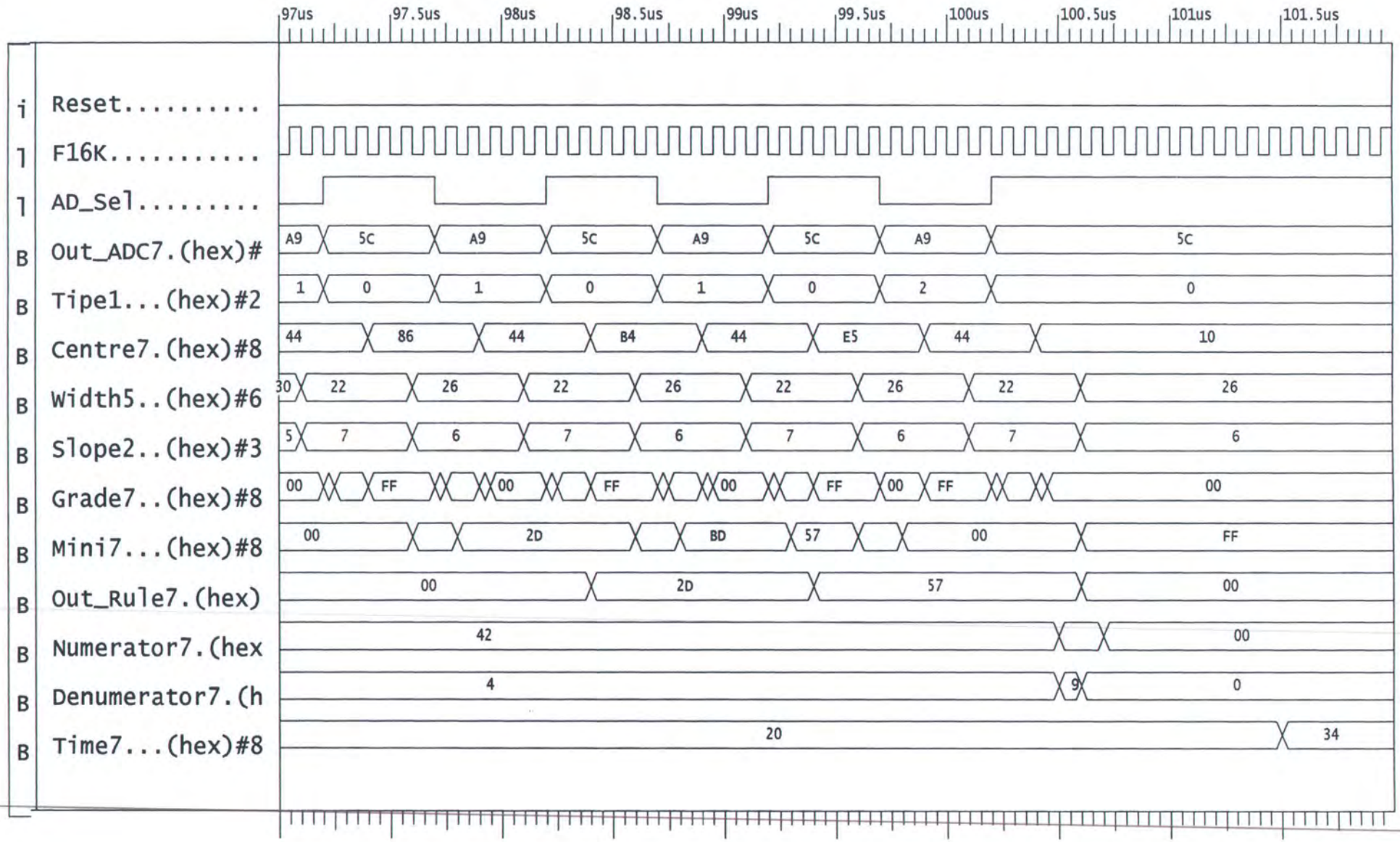

61us
61.5us
62us
62.5us
63us
63.5us
64us
64.5us
65us
65.5us
i Reset.....
l F16K.....
l AD_Sel.....
B Out_ADC7.(hex)#
B Tipe1...(hex)#2
B Centre7.(hex)#8
B Width5..(hex)#6
B Slope2..(hex)#3
B Grade7..(hex)#8
B Mini7...(hex)#8
B Out_Rule7.(hex)
B Numerator7.(hex
B Denumerator7.(h
B Time7...(hex)#8
70 A3 70 A3 70 A3 70 A3
2 0 2 0 1 0 1
77 99 E5 77 99 10 44 50 44 86
26 22 33 26 22 33 26 22 30 22 26
6 7 5 6 7 5 6 7 5 7 6
FF 00 FF 00 00 FF
99 00 00 00
51 99 00
00 99
0 9
44


66us
66.5us
67us
67.5us
68us
68.5us
69us
69.5us
70us
70.5us
i Reset.....
l F16K.....
l AD_Sel.....
B Out_ADC7.(hex)#
B Tipe1...(hex)#2
B Centre7.(hex)#8
B Width5..(hex)#6
B Slope2..(hex)#3
B Grade7..(hex)#8
B Mini7...(hex)#8
B Out_Rule7.(hex)
B Numerator7.(hex
B Denumerator7.(h
B Time7...(hex)#8
70 A3 70 A3 70
0 1 0 2 0
44 B4 44 E5 44 10
26 22 26 22 26 22 26
6 7 6 7 6 7 6
00 FF 00 FF 00 FF 00
51 00 99 00 00 FF
00
99 00
9 0
44 20


82us
82.5us
83us
83.5us
84us
84.5us
85us
85.5us
86us
86.5us
i Reset.....
l F16K.....
l AD_Sel.....
B Out_ADC7.(hex)#
70 A9 5C A9 5C A9 5C A9 5C
B Tipe1...(hex)#2
0 2 1 2 1 2 1 2
B Centre7.(hex)#8
10 E2 50 E2 86 E2 B4 E2
B Width5..(hex)#6
26 33 30 33 26 33 26 33
B Slope2..(hex)#3
6 5 6 5 6 5
B Grade7..(hex)#8
00 00 00 FF FF 00 00 00 00 00
B Mini7...(hex)#8
FF 00 00 2D 00 BD
B Out_Rule7.(hex)
00
B Numerator7.(hex
00
B Denumerator7.(h
0
B Time7...(hex)#8
20

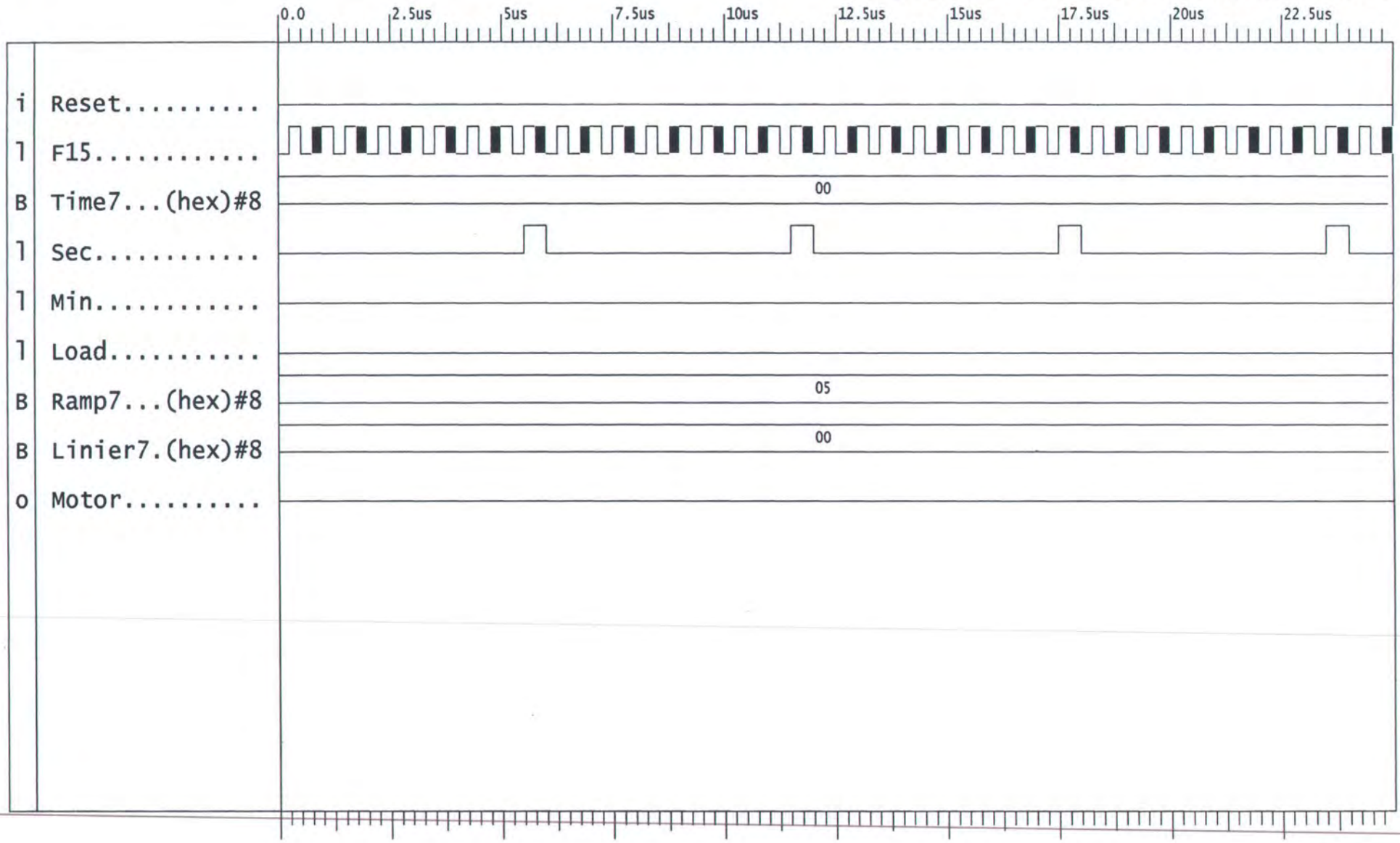

87us
87.5us
88us
88.5us
89us
89.5us
90us
90.5us
91us
91.5us
i Reset.....
l F16K.....
l AD_Sel.....
B Out_ADC7.(hex)#
5C A9 5C A9 5C A9 5C A9 5C
B Tipe1...(hex)#2
2 0 2 1 2 0 1 2
B Centre7.(hex)#8
E5 E2 10 77 99 50 77 99 86 77
B Width5..(hex)#6
33 26 33 26 22 33 30 22 33 26 22
B Slope2..(hex)#3
5 6 5 6 7 5 7 5 6 7
B Grade7..(hex)#8
00 00 FF 00 00 00 FF 00 FF
B Mini7...(hex)#8
00 00 00 00 2D
B Out_Rule7.(hex)
00
B Numerator7.(hex
00
B Denumerator7.(h
0
B Time7...(hex)#8
20

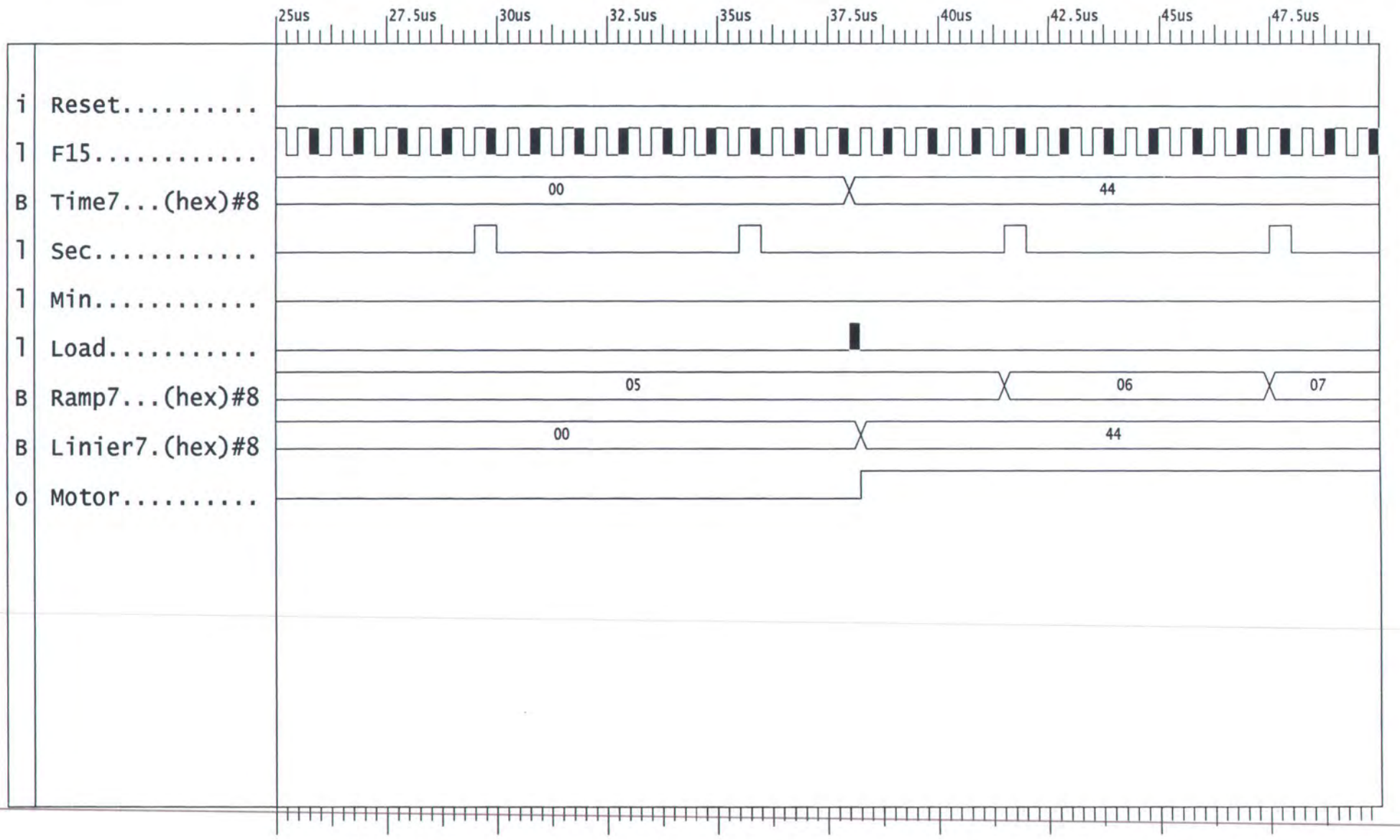

92us
92.5us
93us
93.5us
94us
94.5us
95us
95.5us
96us
96.5us
i Reset.....
l F16K.....
l AD_Sel.....
B Out_ADC7.(hex)#
B Tipe1...(hex)#2
B Centre7.(hex)#8
B Width5..(hex)#6
B Slope2..(hex)#3
B Grade7..(hex)#8
B Mini7...(hex)#8
B Out_Rule7.(hex)
B Numerator7.(hex
B Denumerator7.(h
B Time7...(hex)#8
5C A9 5C A9 5C A9 5C A9
2 0 1 2 0 2 0 1
99 B4 77 99 E5 77 99 10 44 50
22 33 26 22 33 26 22 33 26 22 30
7 5 6 7 5 6 7 5 6 7 5
FF FF 00 FF 00 00 00
2D BD 42 00 00 00
00 2D 42 00
00 42
0 4
20

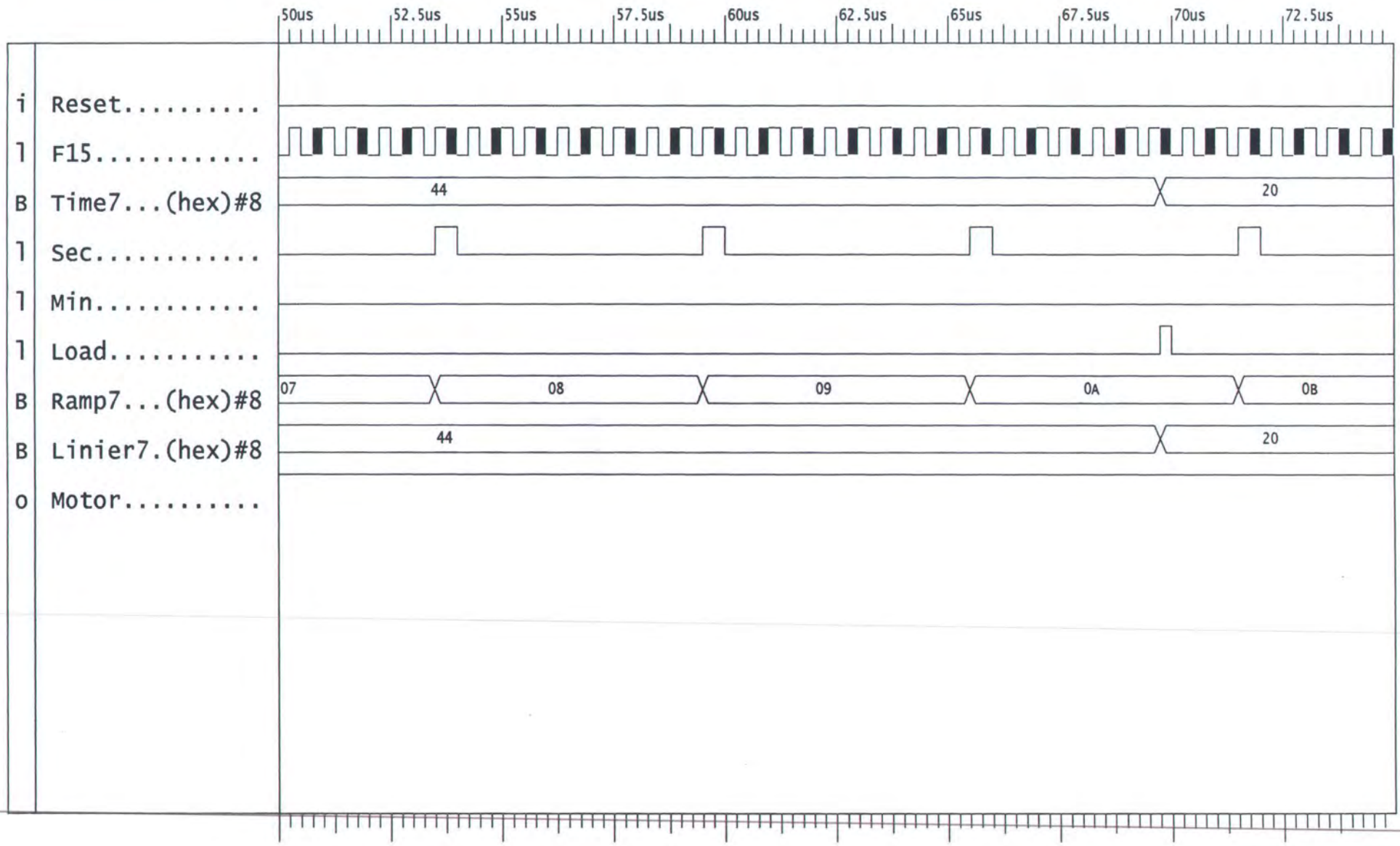

97us
97.5us
98us
98.5us
99us
99.5us
100us
100.5us
101us
101.5us
i Reset.....
] F16K.....
] AD_Sel.....
B Out_ADC7.(hex)#
B Tipe1...(hex)#2
B Centre7.(hex)#8
B Width5..(hex)#6
B Slope2..(hex)#3
B Grade7..(hex)#8
B Mini7...(hex)#8
B Out_Rule7.(hex)
B Numerator7.(hex
B Denumerator7.(h
B Time7...(hex)#8
A9 5C A9 5C A9 5C A9 5C
1 0 1 0 1 0 2 0
44 86 44 B4 44 E5 44 10
30 22 26 22 26 22 26 22 26
5 7 6 7 6 7 6 7 6
00 FF 00 FF 00 FF 00 FF 00
00 2D BD 57 00 FF
00 2D 57 00
42 00
4 9 0
20 34

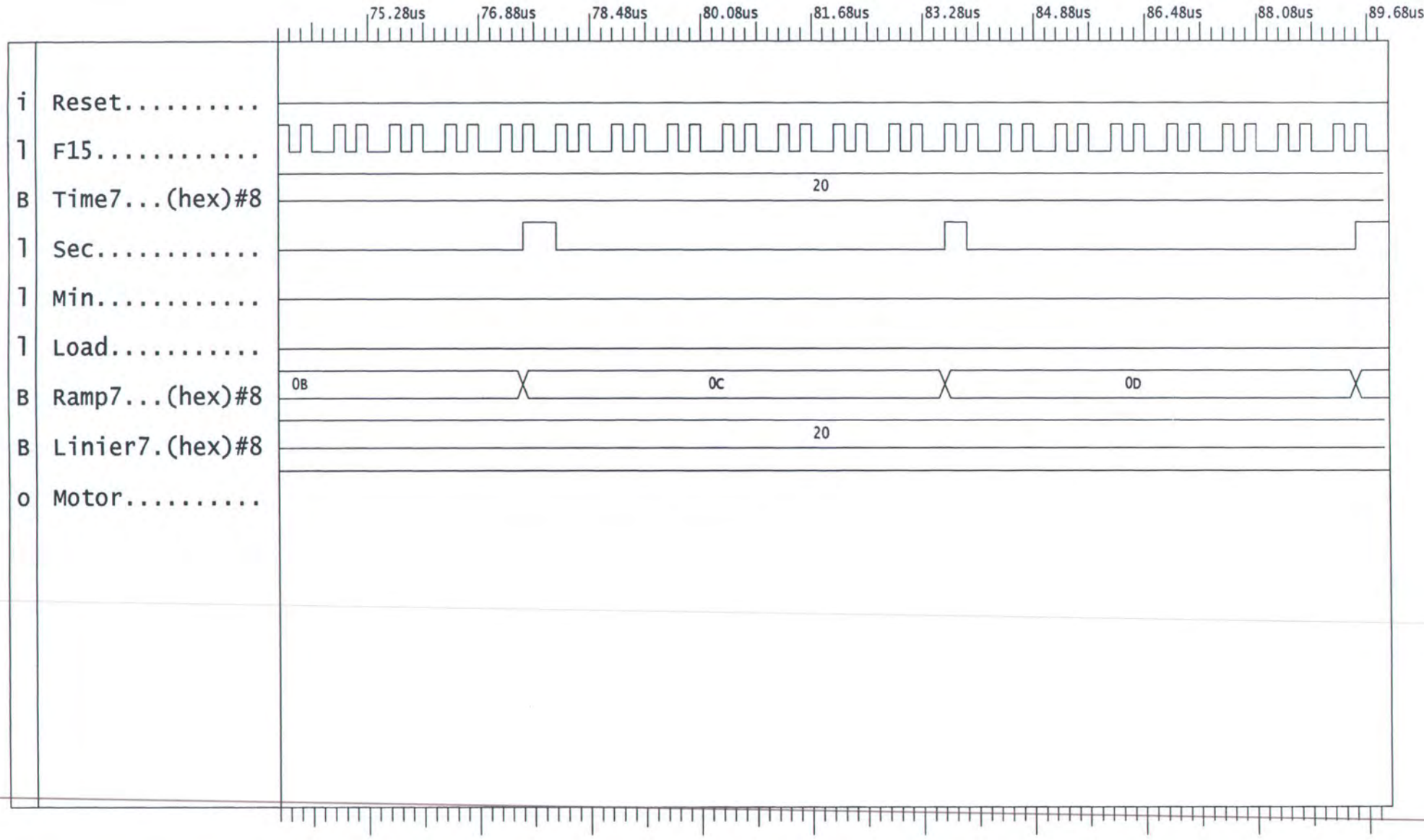
LAMPIRAN D-7

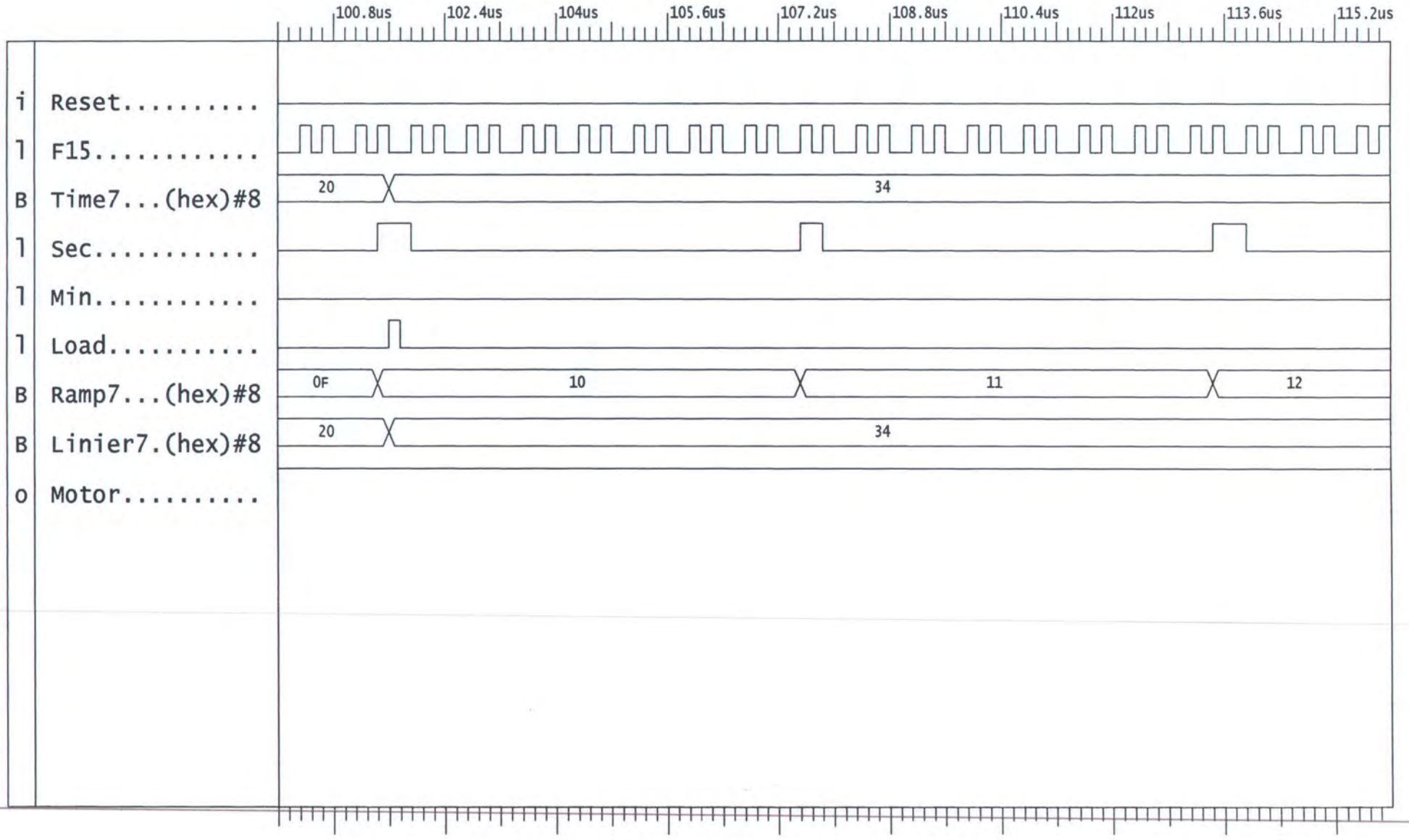
SIMULASI TIMING MODUL DRIVER OUTPUT

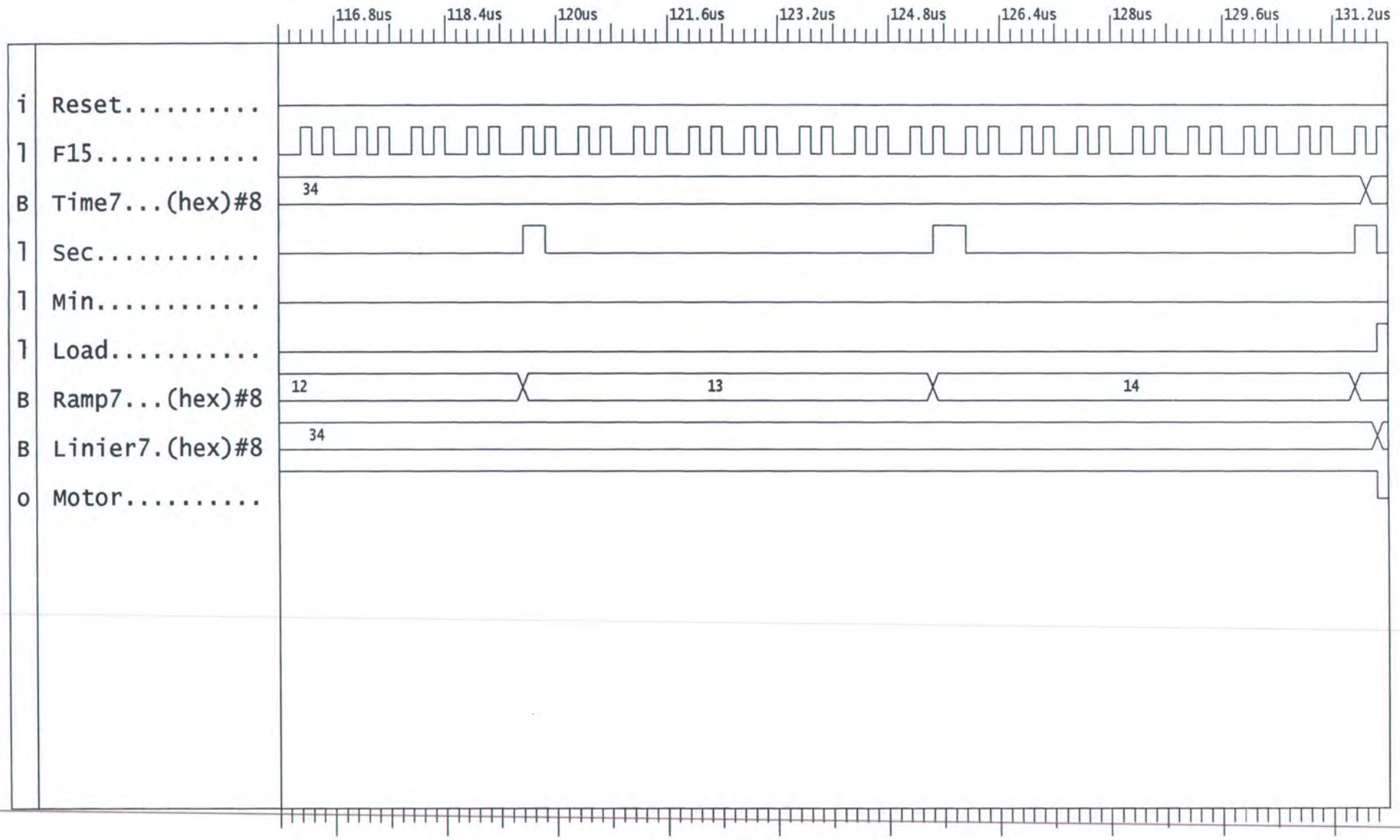



25us
27.5us
30us
32.5us
35us
37.5us
40us
42.5us
45us
47.5us
i Reset.....
l F15.....
B Time7...(hex)#8
00
44
l Sec.....
l Min.....
l Load.....
B Ramp7...(hex)#8
05
06
07
B Linier7.(hex)#8
00
44
o Motor.....


50us
52.5us
55us
57.5us
60us
62.5us
65us
67.5us
70us
72.5us
i Reset.....
l F15.....
B Time7...(hex)#8
44
20
l Sec.....
l Min.....
l Load.....
B Ramp7...(hex)#8
07
08
09
0A
0B
B Linier7.(hex)#8
44
20
o Motor.....


75.28us
76.88us
78.48us
80.08us
81.68us
83.28us
84.88us
86.48us
88.08us
89.68us
i Reset.....
l F15.....
B Time7...(hex)#8
l Sec.....
l Min.....
l Load.....
B Ramp7...(hex)#8
B Linier7.(hex)#8
o Motor.....
20
0B
0C
0D
20


100.8us
102.4us
104us
105.6us
107.2us
108.8us
110.4us
112us
113.6us
115.2us
i Reset.....
l F15.....
B Time7...(hex)#8
20
34
l Sec.....
l Min.....
l Load.....
B Ramp7...(hex)#8
0F
10
11
12
B Linier7.(hex)#8
20
34
o Motor.....


116.8us
118.4us
120us
121.6us
123.2us
124.8us
126.4us
128us
129.6us
131.2us
i Reset.....
l F15.....
B Time7...(hex)#8
34
l Sec.....
l Min.....
l Load.....
B Ramp7...(hex)#8
12
13
14
B Linier7.(hex)#8
34
o Motor.....

RIWAYAT HIDUP

Andy Haryono Kartika, dilahirkan di Semarang pada tanggal 23 April 1977, adalah putra ketiga dari Bapak Eko Purwanto dan Ibu Susi Pudjiowati Ridwan yang bertempat tinggal di Jalan Citarum Selatan 35A Semarang.

Pendidikan yang telah ditempuh selama ini :

1. SD Christus Rex, tahun 1984 – 1989.
2. SMP Domenico Savio, tahun 1990 - 1992
3. SMA Kolese Loyola, tahun 1993 - 1995
4. Terdaftar sebagai mahasiswa Jurusan Teknik Elektro FTI-ITS sejak tahun 1995, dengan nomor pokok 2295100104.

Selama menjadi mahasiswa aktif sebagai:

1. Koordinator Laboratorium Mikroelektronika periode 1999-2000.
2. Koordinator Praktikum Elektronika Lanjut II periode semester gasal 1999/2000.
3. Koordinator Asisten Mata Kuliah Perancangan Sistem Elektronika II periode semester gasal 1999/2000.
4. Asisten Praktikum Rangkaian Listrik.
5. Asisten Praktikum Elektronika.
6. Asisten Praktikum Elektronika Lanjut II.
7. Asisten Mata Kuliah Perancangan Sistem Elektronika II.