UNIVERSITAS INDONESIA

PERANCANGAN DAN IMPLEMENTASI ALGORITMA DYNAMIC TIME WARP PADA FIELD PROGRAMMABLE GATE ARRAY SEBAGAI MODUL FEATURE MATCHING

TESIS

ANDI YUSUF
0906577646

PROGRAM STUDI TEKNIK ELEKTRO
PASCASARJANA FAKULTAS TEKNIK
UNIVERSITAS INDONESIA
DEPOK
2011

UNIVERSITAS INDONESIA

PERANCANGAN DAN IMPLEMENTASI ALGORITMA DYNAMIC TIME WARP PADA FIELD PROGRAMMABLE GATE ARRAY SEBAGAI MODUL FEATURE MATCHING

TESIS

Diajukan sebagai salah satu syarat kelulusan Sarjana Magister

ANDI YUSUF
0906577646

PROGRAM STUDI TEKNIK ELEKTRO
PASCASARJANA FAKULTAS TEKNIK
UNIVERSITAS INDONESIA
DEPOK
2011

HALAMAN PERNYATAAN ORISINILITAS

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

Nama	:	Andi Yusuf
NPM	:	0906577646
Tanda Tangan	:	
Tanggal	:	Juli 2011

HALAMAN PENGESAHAN

Thesis ini diajukan oleh

Nama	: Andi Yusuf
NPM	: 0906577646
Program Studi	: Magister
Judul Seminar	: Perancangan dan Implementasi Algoritma *Dynamic Time Warp* pada *Field Programmable Gate Array* sebagai Modul *Feature Matching*

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

DEWAN PENGUJI

Pembimbing	: Prof. Dr. Ir. Harry Sudibyo, DEA	(.....)
Penguji	: Dr. Ir. Feri Yusivar, M.Eng	(.....)
Penguji	: Ir. Purnomo Sidi Priambodo, M.Sc., Ph.D.	(.....)
Penguji	: Dr. Ir. Armein D. Diponegoro	(.....)

Ditetapkan di :

Tanggal :

UCAPAN TERIMA KASIH

Puji syukur kepada Allah SWT atas berkat dan rahmat-Nya, saya dapat menyelesaikan tesis ini. Penulisan tesis ini dilakukan dalam rangka memenuhi salah satu syarat kelulusan Sarjana Magister di bidang Teknik Elektro. Saya sadar bahwa tanpa bantuan dari berbagai pihak, sejak awal masa perkuliahan hingga selesainya tesis ini banyak sekali halangan dan rintangan dalam menyelesaikan tesis ini. Seiring dengan terselesaikannya tesis ini, saya mengucapkan terima kasih yang sedalam-dalamnya kepada :

(1) Prof. Dr. Ir. Harry Sudibyo, DEA selaku pembimbing yang telah menyediakan waktu, tenaga dan pikirannya untuk mengarahkan penulis dalam terlaksananya penulisan tesis ini;
(2) Lembaga Sandi Negara selaku Institusi yang memberikan kesempatan kepada penulis untuk dapat menuntut ilmu di Universitas Indonesia dan mengijinkan dalam penggunaan sarana dan prasarana lab sebagai tempat riset dalam penyelesaiaan tesis ini ;
(3) Orang tua tersayang , istri tercinta dan seluruh keluarga yang telah memberi dukungan dan doanya;
(4) Rekan – rekan mahasiswa Program Pascasarjana Fakultas Teknik, Program Studi Teknik Elektro Angkatan 2009/2010 atas diskusi penelitiannya.

Akhir kata, semoga Allah SWT berkenan membalas segala kebaikan semua pihak yang membantu dalam pelaksaan riset ini. Semoga Tesis ini membawa manfaat bagi pengembangan ilmu dalam bidang Teknik Desain VLSI dan *Embedded System*.

Depok, Juli 2011

Penulis

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI

TESIS UNTUK KEPENTINGAN AKADEMIS

Sebagai civitas akademik Universitas Indonesia , saya yang bertanda tangan di bawah ini :

Nama : Andi Yusuf

NPM : 0906577646

Program Studi : Magister

Departemen : Teknik

Fakultas : Elektro

Jenis Karya : Tesis

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

Perancangan dan Implementasi Algoritma *Dynamic Time Warp* pada *Field Programmable Gate Array* sebagai Modul *Feature Matching* .

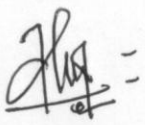
Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihkan media /formatkan , mengelola dalam bentuk pangkalan data (database), merawat, dan mempublikasikan seminar saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Jakarta

Pada tanggal : Juli 2011

Yang menyatakan



(Andi Yusuf)

ABSTRAK

Nama : Andi Yusuf

Program Studi : Teknik Elektro

Judul : Perancangan dan Implementasi Algoritma Dynamic Time Warp pada Field Programmable Gate Array Sebagai Modul Feature Matching

Pengenalan ucapan atau disebut juga *speech recognition* adalah suatu pengembangan teknik dan sistem yang memungkinkan perangkat system untuk menerima masukan berupa kata yang diucapkan. Teknologi ini memungkinkan suatu perangkat untuk mengenali kata yang diucapkan dengan cara merubah kata tersebut menjadi sinyal digital dan mencocokkan dengan suatu pola tertentu yang tersimpan dalam suatu perangkat. Pola tertentu yang tersimpan pada suatu perangkat sebenarnya sampel kata yang diucapkan pengguna. Salah satu algoritma yang digunakan sebagai pemodelan dasar untuk pengenalan ucapan adalah *Dynamic Time Warping* (DTW). DTW digunakan sebagai algoritma untuk mencocokkan pola yang dimaksud dengan mengukur dua buah sekuensial pola dalam waktu yang berbeda[7].

Dalam penelitian ini akan dibahas mengenai perancangan IC *pattern matching* menggunakan algoritma DTW dan diimplementasikan pada sebuah *Field Programmable Gate Array* (FPGA). Algoritma DTW yang digunakan merupakan pengembangan dari algoritma standar yaitu FastDTW[13]. Perancangan difokuskan pada pembuatan layout *Complementary Metal Oxide Silicon* (CMOS) pada skala 0,18µm dengan metode *semi custom*. Layout yang terbentuk baik layout untuk IC DTW maupun layout – layout gerbang logika dasar penyusun IC tersebut, dapat dilihat *behavior*-nya. Dengan menggunakan *Computer Aided Design* (CAD) Electric *behavior* dapat diterjemahkan dalam bahasa *hardware* yang dikenal dengan *Very High Speed Integrated Circuit Hardware Description Language* (VHSIC HDL atau VHDL). Proses verifikasi dilakukan dengan membuat *prototype* perangkat keras menggunakan rangkaian ADC dan FPGA Spartan-IIELC yang telah diimplementasikan VHDL dari IC DTW.

Kata kunci : *FastDTW, CMOS, semi custom, VHDL dan FPGA*

ABSTRACT

Name : Andi Yusuf

Study Program : Electrical Engineering

Title : Design and Implementation of Dynamic Time Warp Algorithm on Field Programmable Gate Array as Feature Matching Module

Speech recognition is also called a development of techniques and systems that enable the device system to receive input of the spoken word. This technology allows a device to recognize words spoken in a way to change the word into a digital signal and the match with a particular pattern stored in a device. Certain patterns that are stored on a device is a spoken word sample of users. One algorithm used as a basis for modeling of speech recognition is the Dynamic Time Warping (DTW). DTW is used as an algorithm to match the pattern in question by measuring two sequential patterns in different time [7].

In this research will be discussed regarding the design of the IC pattern matching using DTW algorithm and implemented on a Field Programmable Gate Array (FPGA). DTW algorithm used is the development of a standard algorithm that is FastDTW [13]. The design focused on making the layout of Complementary Metal Oxide Silicon (CMOS) on a scale of 0.18 μm with a method of semi-custom. Formed a good layout for IC DTW and layout of the basic logic gate, we can see his behavior. By using Computer Aided Design (CAD) Electric, behavior can be translated in hardware language, known as Very High Speed Integrated Circuit Hardware Description Language (VHSIC HDL or VHDL). The verification process is done by making a prototype hardware uses a circuit of ADC and the FPGA Spartan-IIELC that have been implemented VHDL from IC DTW

Key words : FastDTW, CMOS, semi custom, VHDL dan FPGA

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

DAFTAR ISTILAH

ISTILAH BIDANG ILMU ELEKTRO

1. Biometric

   Pengukuran yang dilakukan berdasarkan proses biologis atau karakteristik secara fisik.

2. *Embedded*

   Sistem komputer tertanam yang didesain untuk mengerjakan satu atau beberapa pekerjaan tertentu saja yang bekerja secara *real-time.*

3. CMOS

   Transistor semikonduktor oksida logam komplementer yang terbentuk dari tansistor p-MOS dan n-MOS.

4. Capacitance

   Suatu media yang dapat menyimpan energi listrik dengan fenomena medan listrik

5. Dielectric

   Suatu bahan yang mengandung substrat tertentu dengan bahan ini merupakan bahan isolator

6. Permitivity

   Suatu nilai konstanta substrat bahan baik isolator maupun conductor

7. *Slice*

   Kumpulan beberapa *logic cells* dalam suatu arsitektur *Field Programmable Gate Array*.

8. *Logic cells*

   Kumpulan beberapa gerbang logika seperti flip flop dan mutiplexer.

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Biometric merupakan suatu metode dari proses pengenalan identitas seseorang yang didasarkan pada karakter fisik dan karakter tingkah laku yang unik yang dimiliki manusia [2]. *Biometric* menawarkan keuntungan yang pasti, seperti *negative recognition* dan *non-repudiation* yang tidak dapat disediakan oleh *token* dan *password* [1]. *Negative recognition* adalah proses ketika suatu sistem menentukan bahwa seseorang telah terdaftar di dalam sistem, walaupun pihak tersebut menyangkalnya. Sedangkan *non-repudiation* merupakan cara untuk menjamin bahwa seseorang yang telah mengakses suatu sistem tidak akan dapat menyangkal bahwa ia telah menggunakan/mengaksesnya. Salah satu teknik *biometric* berdasarkan karakteristik tingkah laku yaitu *speech recognition*. *Speech recognition* adalah suatu pengembangan teknik dan sistem yang memungkinkan perangkat system untuk menerima masukan berupa kata yang diucapkan. Teknologi ini memungkinkan suatu perangkat untuk mengenali dan memahami kata-kata yang diucapkan dengan cara merubah kata-kata menjadi sinyal digital dan mencocokkan sinyal digital tersebut dengan suatu pola tertentu yang tersimpan dalam suatu perangkat. Salah satu algoritma yang digunakan sebagai pemodelan dasar untuk pengenalan ucapan adalah *Dynamic Time Warping* (DTW). DTW digunakan sebagai algoritma untuk mencocokkan pola – pola yang dimaksud, dengan mengukur dua buah sekuensial pola dalam waktu yang berbeda dan menghitung nilai warping pathnya[7].

Berkembangnya teknologi IC membawa dampak positif terhadap penelitian mengenai pengenalan suara yang menggunakan metode DTW. Pada tahun 1983 Neil Weste, David J. Burr dan Bryan D. Ackland melakukan penelitian terkait pembuatan IC DTW dengan teknik *multiprocessing array*[8]. Dilanjutkan penelitian – penelitian yang serupa dengan pembahasan dikaitkan pada memory dan jenis input suara [9]-[11] yang dilakukan pada tahun 1987 – 1996 dan konfigurasi terhadap data path [12] yang dilakukan pada tahun 2002. Setelah itu perkembangan penelitian terkait

pembuatan IC DTW seakan jalan ditempat, terlebih munculnya metode baru sekitar tahun 1990 yaitu *Hiden Markov Model* (HMM) membuat penelitian terkait pembuatan IC untuk pengenalan suara lebih memilih menggunakan metode tersebut. Hal yang menarik terjadi pada tahun 2007 ketika Stan Salvador dan Philip Chan melakukan penelitian dengan memperkenalkan metode baru untuk memperbaiki metode DTW terkait *quadratic time* dan *space complexciy* yaitu FastDTW[13].

Dalam penelitian ini akan dibahas mengenai perancangan IC *pattern matching* mengunakan algoritma DTW dan diimplementasikan pada sebuah *Field Programmable Gate Array* (FPGA). Algoritma DTW yang digunakan merupakan pengembangan dari algoritma standar yaitu FastDTW[13]. Perancangan difokuskan pada pembuatan layout *Complementary Metal Oxide Silicon* (CMOS) pada skala 0,18µm dengan metode *semi custom*. Layout yang terbentuk baik layout untuk IC DTW maupun layout – layout gerbang logika dasar penyusun IC tersebut, dapat dilihat *behavior*-nya. Dengan menggunakan *Computer Aided Design* (CAD) Electric, *behavior* dapat diterjemahkan dalam bahasa *hardware* yang dikenal dengan *Very High Speed Integrated Circuit Hardware Description Language* (VHSIC HDL atau VHDL). Proses verifikasi dilakukan dengan membuat *prototype* perangkat keras menggunakan rangkaian ADC dan FPGA Spartan IIELC yang telah diimplementasikan VHDL dari IC DTW.

1.2 Perumusan Masalah

Penelitian yang dilakukan Stan Salvador dan Philip Chan dengan memperkenalkan metode FastDTW mendapatkan hasil yang cukup signifikan terkait waktu eksekusi bila dibandingkan dengan metode DTW yang dapat dilihat pada tabel 1.1. Bagaimana merancang IC *pattern matching* dengan menggunakan algoritma FastDTW dihadapkan pada waktu delay *switching* transistor CMOS sehingga menghasilkan waktu eksekusi yang lebih cepat akan dibahas pada penelitian ini. Masalah lain adalah bagaimana mengimplementasikan rancangan layout CMOS IC FastDTW pada sebuah *device* FPGA sehingga hasil perancangan tidak sekedar simulasi proses melainkan berupa *prototype* perangkat keras.

Tabel 1.1 Waktu eksekusi (dalam detik) dari DTW dan FastDTW dengan panjang yang berbeda dari empat *time series* [13]

Method	Length of Time Series			
	100	1000	10000	100000
DTW	0,02	0,92	57,45	7969,59
FastDTW (radius = 0)	0,01	0,02	0,38	67,94
FastDTW (radius = 100)	0,02	0,06	8,42	207,19

1.3 Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk melakukan perancangan layout CMOS IC *pattern matching* menggunakan algoritma DTW dan pembuatan *prototype* perangkat keras *speech recognition* menggunakan FPGA sehingga dalam hasil simulasi layout CMOS dan implementasi didapatkan waktu proses lebih cepat dibandingkan penelitian yang dilakukan Stan Salvador dan Philip Chan[13].

1.4 Manfaat Penelitian

Manfaat dari penelitian ini adalah mendapatkan karakterisasi dan performa dari IC *pattern matching* menggunakan algoritma DTW dan pembuatan *prototype* menggunakan *device* FPGA dapat dijadikan solusi lain dalam perancangan IC digital untuk *pattern matching* ketika pabrikasi IC sulit untuk dilakukan. Sehingga dalam bidang industri *protoype* yang dibuat dapat digunakan sebagai modul pengenalan suara dalam aplikasi autentikasi suara dan *voice command* pada sistem embedded atau dapat dijadikan *prototype* awal untuk dikembangankan dalam pembuatan modul notulensi elektronik.

1.5 Batasan Penelitian

Dalam thesis ini, perancangan layout CMOS pada skala 0,18 µm dengan menggunakan Java CAD 'Electric' dan LTSpice serta digunakan *device* FPGA Spartan IIELC pada tahap implementasi. Adapun batasan dari penelitian ini adalah

1. Algoritma yang digunakan untuk metode *feature matching* adalah algoritma DTW yaitu FastDTW pada radius = 0 dan tidak membahas algoritma yang lain.
2. Pada penellitian ini tidak dibahas mengenai algoritma untuk metode *feature extraction*. Adapun pembuatan *hardware prototype speech recognition* terkait modul untuk *feature extraction* menggunakan penelitian yang sudah ada [27].
3. Input suara berupa pengucapan huruf ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’.

1.6 Model Operasional Penelitian

Dalam Thesis ini, metode operasional penelitian yang dilakukan adalah sebagai berikut:

- Studi literatur dari berbagai buku teks, paper, dan internet untuk mendapatkan data primer.
- Pembuatan layout CMOS IC FastDTW menggunakan Java CAD *tools* Electric dan LTSpice .
- Pembuatan *prototype* perangkat keras menggunakan rangkaian ADC dan FPGA Sparta IIELC
- Pengujian hasil perancangan dilakukan dengan cara membandingkan simulasi pengukuran *timming diagram* dan analisis transient dengan hasil aplikasi FastDTW menggunakan Java yang dibuat Stan Salvador dan Philip Chan [13].
- Pengujian hasil implementasi dilakukan dengan cara membandingkan hasil implementasi pada FPGA Spartan-IIELC dengan hasil aplikasi FastDTW menggunakan Java yang dibuat Stan Salvador dan Philip Chan [13].
- Percobaan dilakukan dengan menginputkan suara ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’ pada modul *feature extracting* dan dibandingkan dengan *feature* 1, *feature* 2, *feature* 3, *feature* 4 dan *feature* 5 yang ada pada modul *feature matching*.

BAB 2

DASAR TEORI

2.1 Biometric Suara

Dalam kehidupan sehari-hari, dibutuhkan suatu cara untuk melakukan verifikasi identitas seseorang. Proses otentikasi yang terpercaya dapat membuat komunikasi dan transaksi, atau perjanjian antara dua atau beberapa pihak menjadi lebih aman dan efisien. Otomatisasi proses otentikasi dapat memberikan keamanan, efisiensi, dan pemanfaatan waktu yang lebih baik dalam kehidupan bila dibandingkan dengan proses otentikasi yang dilakukan secara manual.

Otentikasi yang dilakukan secara otomatis oleh alat akan memberikan respon yang berbeda terhadap entitas yang berbeda dan akan memastikan apakah respon tersebut tepat untuk seseorang sesuai dengan kebiasaannya. Terdapat beberapa alasan utama mengapa *biometric* menjadi lebih popular dibandingkan teknik otentikasi lainnya [2], yaitu:

1. Otentikasi yang tepat

Ketepatan dari otentikasi yang cepat dan mudah dapat dibuat oleh sistem yang menggunakan jaminan identitas berupa kunci, kartu, *token*, atau PIN. Dengan menggunakan teknologi *biometric*, pengguna tidak perlu khawatir menghilangkan atau melupakannnya selama fisik dan karakter atau sifat seseorang masih dapat diidentifikasi. Selain itu teknologi *biometric* dapat memberikan nilai efisiensi waktu bagi Teknologi Informasi dan organisasi pendukung yang mengatur sistem otentikasi. Salah satu contohnya yaitu *biometric* membantu mengurangi kebutuhan untuk me-*reset* PIN.

2. Peningkatan kebutuhan untuk sistem otentikasi yang kuat

Password dan PIN dapat dengan mudah dicuri. *Biometric* dapat mengurangi resiko tersebut karena akses dapat diperoleh dengan menggunakan sesuatu yang melekat pada pihak yang bersangkutan. *Biometric* menawarkan metode yang menarik

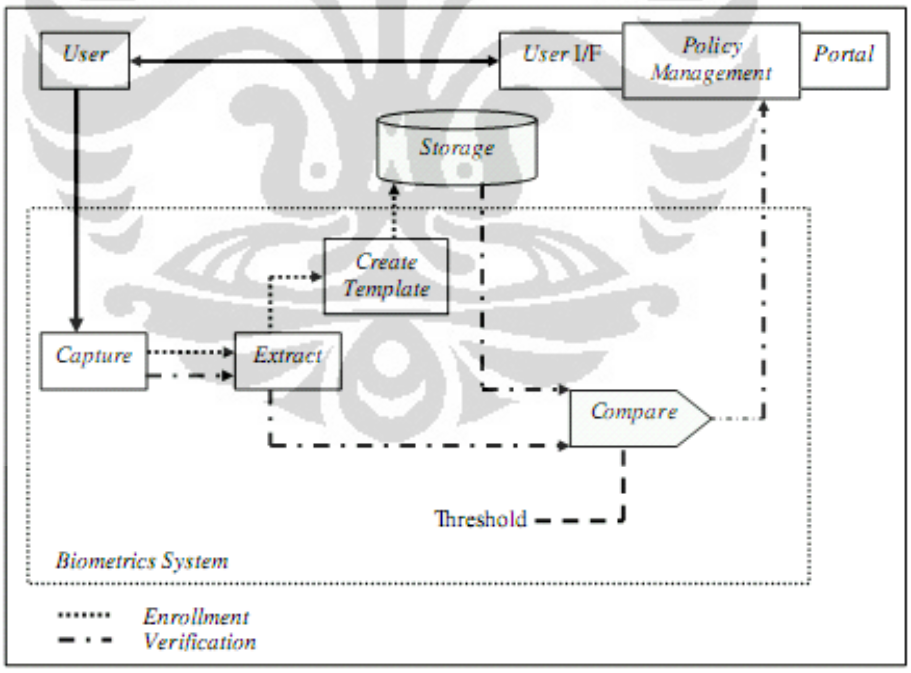
dalam menjaga dari pencurian atau kehilangan alat identitas, seperti kartu atau *password*.

3. Mengurangi biaya

Teknologi *hardware* dan *software* yang semakin banyak diproduksi membuat biaya yang harus dikeluarkan untuk sistem otentikasi *biometric* menjadi menurun, sehingga dapat meningkatkan penjualan bagi pihak-pihak yang terkait. Selain itu, kemajuan pada *computing power*, *networking*, dan *database system* telah mengijinkan sistem *biometric* untuk dapat digunakan dengan mudah.

4. Meningkatkan penggunaan pada bidang industri dan pemerintahan

Saat ini tidak sedikit organisasi yang telah menggunakan teknik *biometric* dan provider yang telah menyediakan alat-alat yang mendukung penerapan teknik *biometric* yang *embedded* pada perlengkapan komputer ataupun produk lain, dan hal tersebut terus berkembang.



Gambar 2.1 Proses dasar *biometric*[4]

Pada gambar 2.1 terdapat beberapa proses dalam sistem *biometric* [4], yaitu:

a. Capture

Capture dilakukan pada dua proses utama, yaitu *enrollment* dan *verification*. Pada proses *capture*, masukan data akan ditangkap dan diubah menjadi sampel-sampel yang akan diolah oleh sistem selanjutnya.

b. Extract

Dalam proses *extract*, masukan *biometric* akan dibagi menjadi beberapa *frame* yang telah ditentukan, dan tetap mempertahankan karakteristik yang dimiliki oleh data masukan. *Extract* merupakan tahapan yang sangat penting karena perubahan dari tingkat keunikan yang dimiliki data akan mempengaruhi tingkat kesalahan dari suatu sistem otentikasi. Idealnya, kualitas yang dimiliki oleh suatu data masukan selama proses *enrollment* adalah tinggi, karena hal tersebut merupakan dasar dari sistem otentikasi yang digunakan. Proses pembandingan akan menggunakan data hasil pengolahan pada proses *enrollment*.

c. *Create Template*

Dalam proses ini akan dibuat suatu *template* dari data masukan setelah mengalami proses *extract*. Pada pembuatan *template* suatu data masukan dapat pula diberikan proses tambahan seperti penyandian atau penambahan identitas yang diberikan sistem terhadap data masukan.

d. Storage (Penyimpanan *template*)

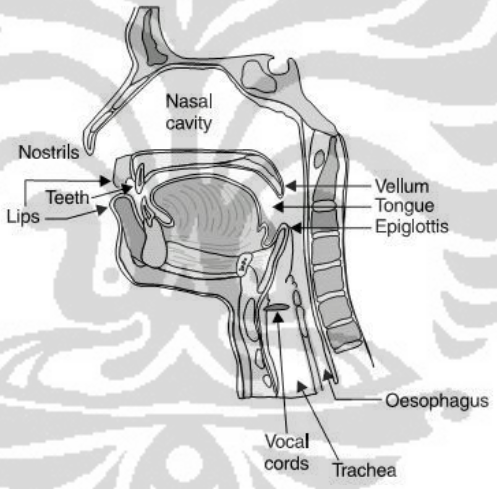
Suatu objek *biometric* yang telah melewati beberapa tahapan proses akan diubah menjadi suatu *template* yang akan disimpan di dalam basis data. *Template* tersebut akan digunakan sebagai acuan dalam melakukan proses otentikasi pengguna.

e. *Compare* (Pembandingan)

Sistem melakukan proses pembandingan dengan cara membandingkan *template* yang sudah tersimpan di dalam basis data dengan karakteristik yang dimiliki oleh pengguna sebagai informasi identitas pengguna.

Suara merupakan kombinasi dari *biometric* yang berdasarkan pada karakteristik fisik dan tingkah laku [1]. *Voice Verification* adalah teknik *biometric* dengan memanfaatkan komponen fisiologi dan tingkah laku seseorang. Bentuk fisik dari

sistem penghasil suara merupakan komponen utama fisiologi. Ciri-ciri fisik dari suara seseorang berdasarkan pada bentuk dan ukuran anggota badan yang terkait, seperti pita suara, mulut, rongga hidung, dan bibir yang digunakan dalam proses sintesis suara. Pada rongga mulut terdapat gerakan dari mulut, rahang, lidah, tekak, dan pangkal tenggorokan untuk mengucapkan kata-kata dengan jelas dan mengendalikan suara yang dihasilkan. Karakter fisik dari hubungan ini memberikan pola akustik yang dapat mengukur suara yang dihasilkan. Bentuk, panjang, dan kerasnya suara bertindak seperti filter bunyi, mempengaruhi nada, titinada, dan resonansi. Karakteristik fisik dari suara manusia ini tidak berbeda untuk seorang individu, namun aspek tingkah laku pada suara berubah seiring berubahnya usia, kondisi kesehatan, kestabilan emosi, dan sebagainya. Gerakan, gaya berbicara, dan pelafalan dari suatu kata merupakan dasar dari aspek tingkah laku pada *biometric* suara.



Gambar 2.2 Anatomi alat penghasil suara

Untuk mengetahui proses pengenalan suara manusia, diperlukan pemahaman tentang urutan proses terjadinya suara pada manusia. Proses tersebut antara lain adalah sebagai berikut : udara mengalir dari *lungs* (paru-paru) bergerak menuju *trachea*, yaitu sebuah tabung tersusun dari cincin *cartilage*, dan melalui *larynx* menuju *vocal tract*. Dalam hal ini *larynx* beraksi sebagai *gate* (pintu gerbang) antara *lungs* dan *mouth* (mulut). *Larynx* tersusun atas *epiglottis*, *vocal cords* dan *false vocal cords* [5]. Ketiganya menutup saat menelan makanan, sehingga makanan tidak masuk ke dalam paru-paru dan membuka kembali pada saat mengambil napas normal; dalam

bahasa Inggris diklasifikasikan dalam terminologi *manner of articulation* dan *place of articulation*. Konsentrasi *Manner of articulation* adalah pada aliran udara, yaitu masalah lintasan dan tingkatan yang terjadi pada vokal yang dilewatkan.

Sedangkan *manner of articulation* dan *voicing* membagi fonem menjadi tiga kelas besar. Fonem yang memproduksi *employ voicing* dan semata-mata merangsang *vocal tract* pada *glottis* disebut *sonorants* (*vowels, diphthongs, glides, liquids*, dan *nasals*). Mereka memiliki sifat *continuous*, *intense*, dan *periodic phonemes*. *Voiced sounds* dihasilkan oleh tekanan pada udara yang mengalir melalui *vocal cords*, sementara *vocal cords* ditekan untuk membuka dan menutup secara cepat untuk menghasilkan sederetan *puffs periodic* yang memiliki *fundamental frequency* (harmonisasi ke-1) sama seperti frekuensi vibrasi *vocal cord*. Frekuensi *vocal cord* tergantung pada tingkat kepejalan, *tension*, dan panjang *vocal cords* dan efek aliran udara yang dihasilkan dalam *glottis*, yaitu sebuah ruang diantara *vocal cord – vocal cord*.

2.2 *Dynamic Time Warp* (DTW)

2.2.1 Standar DTW

Dynamic Time Warping algorithm (DTW) [7] adalah algortima yang menghitung *optimal warping path* antara dua waktu. Algoritma ini menghitung baik antara nilai *warping path* dari dua waktu dan jaraknya. Misalnya, kita memiliki dua sekuens *numeric* (a1, a2, ..., am) dan (b1, b2, ..., bm). Dengan pemisalan ini, maka dapat dikatakan bahwa panjang dua *sekuens* ini bisa saja berbeda. Algoritma ini memulai dengan penghitungan jarak lokal antara elemen dari sekuens menggunakan tipe jarak yang berbeda. Frekuensi yang paling banyak menggunakan metode untuk penghitungan jarak adalah jarak absolut antar nilai dua elemen. Jika dalam matriks maka dapat ditulis dengan memiliki *n* garis dan *m* kolom, secara umum:

$$d_{ij} = \left| a_i - b_j \right|, \; i = \overline{1,n}, \; j = \overline{1,m}. \qquad \text{..... (2.1)}$$

Mulai dengan matrik jarak lokal, kemudian minimum jarak matriks antar sekuens ditentukan menggunakan algoritma program dinamis dan mengikuti kriteria optimasi berikut:

$$a_{ij} = d_{ij} + \min(a_{i-1,j-1}, a_{i-1,j}, a_{i,j-1}) \quad \ldots\ldots\ldots (2.2)$$

Dimana a_{ij} merupakan jarak minimal antara subsekuens. *Warping path* adalah sebuah *path* yang melewati jarak matrik minimum dari elemen a_{11} ke *anm* . Ongkos *warping path* secara global dari dua sekuens:

$$GC = \frac{1}{p}\sum_{i=1}^{p} w_i \quad \ldots\ldots\ldots (2.3)$$

Dimana W_i adalah elemen yang dimiliki *warping path* dan p adalah jumlahnya. Contoh penghitungan dibuat untuk dua sekuens diperlihatkan pada gambar 2.4 dimana *warping path* diberi *highlight*.

	-2	10	-10	15	-13	20	-5	14	2
3	5	12	25	37	53	70	78	89	90
-13	16	28	15	43	37	70	78	105	104
14	32	20	39	16	43	43	62	62	74
-7	37	37	23	38	22	49	45	66	71
9	48	38	42	29	44	33	47	50	57
-2	48	50	46	46	40	55	36	52	54

Gambar 2.3 *Warping path* pada DTW

2.2.2 *Fast DTW*

AlgoritmaFastDTW menggunakan pendekatan bertingkat dengan tiga operasi kunci [13] :

1) *Coarsening* – memperkecil suatu *time series* tertentu kedalam suatu kurun waktu yang lebih kecil yang mewakili kurva yang sama seakurat mungkin dengan titik data yang lebih sedikit.

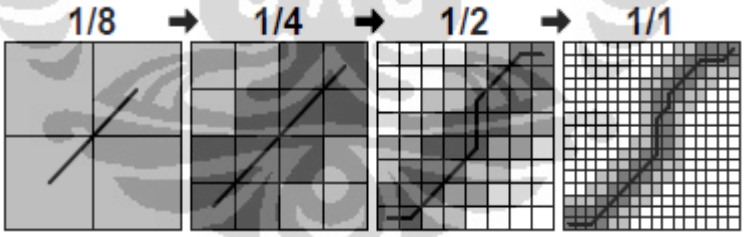
2) *Projection* – mencari jarak minimum *warp path* pada resolusi yang lebih rendah, dan menggunakan *warp path* sebagai dugaan awal untuk jarak minimal *warp path* resolusi yang lebih tinggi.

3) *Refinement* – Memperbaiki *warp path* diproyeksikan dari resolusi yang lebih rendah melalui penyesuaian lokal *warp path*.

Coarsening mengurangi ukuran (atau resolusi) dari serangkaian *time series* dengan rata-rata pasangan yang berdekatan titiknya. *Time series* yang dihasilkan

adalah faktor dua lebih kecil dari *time series* asli. *Coarsening* dijalankan beberapa kali untuk menghasilkan resolusi yang berbeda dari *time series*. *Projection* mengambil *warp path* dihitung pada resolusi yang lebih rendah dan menentukan apa sel-sel dalam tahap berikutnya pada *time series* dengan resolusi melewati *warp*. Sejak resolusi meningkat dengan faktor dua, satu titik dalam *warp path* resolusi rendah akan dipetakan untuk setidaknya empat poin pada resolusi yang lebih tinggi (kemungkinan > 4 jika $|X| \neq |Y|$). Jalan ini diproyeksikan kemudian digunakan sebagai heuristik selama perbaikan solusi untuk menemukan jalan warp pada resolusi yang lebih tinggi. *Refinement* menemukan *warp path* optimal disekitar jalur diproyeksikan, dimana ukuran lingkungan dikontrol oleh parameter radius.

DTW adalah $O(N^2)$ algoritma karena setiap sel dalam *cost matriks* harus diisi untuk memastikan jawaban yang optimal ditemukan, dan ukuran dari matriks tumbuh kuadratik dengan panjang *time series*. Dalam pendekatan bertingkat, *cost matriks* hanya diisi disekitar jalur yang diproyeksikan dari resolusi sebelumnya. Karena panjang *warp path* tumbuh linier dengan panjang input *time series*, maka pendekatan bertingkat adalah O(N) algoritma.



Gambar 2.4 Empat resolusi yang berbeda selama menjalankan *Fast DTW*

Algoritma FastDTW pertama menggunakan pengkasaran untuk membuat semua keputusan yang akan dievaluasi. Gambar 2.5 menunjukkan empat resolusi yang dibuat saat menjalankan algoritma FastDTW pada *time series* yang sebelumnya digunakan. DTW dijalankan untuk menemukan *warp path* yang optimal untuk seri resolusi terendah waktu. Resolusi terendah dari *warp path* ditunjukkan disebelah kiri Gambar 2.5. Setelah *warp path* ditemukan untuk resolusi terendah, diproyeksikan

untuk resolusi berikutnya yang lebih tinggi. Dalam Gambar 2.5, proyeksi *warp path* dari resolusi 1 / 8 ditampilkan sebagai sel sangat berbayang pada resolusi 1 / 4.

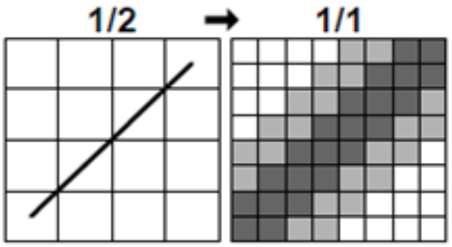
Untuk memperhalus jalur yang diproyeksikan, suatu algoritma DTW terkendala dijalankan dengan batasan yang sangat spesifik yang sel-sel hanya di *warp path* yang diproyeksikan dievaluasi. Ini akan menemukan *warp path* optimal melalui daerah *warp path* yang diproyeksikan dari resolusi yang lebih rendah. Namun, seluruh *warp path* optimal mungkin tidak terkandung dalam jalur yang diproyeksikan. Untuk meningkatkan peluang menemukan solusi optimal, ada parameter radius yang mengontrol penambahan jumlah sel di setiap sisi jalur yang diproyeksikan juga akan dievaluasi ketika memperbaiki *warp path*. Dalam Gambar 2.5, parameter radius diatur 1. Setelah *warp path* dihaluskan pada resolusi 1 / 4, bahwa *warp path* diproyeksikan untuk resolusi 1 / 2, diperluas denganr radius 1, dan lebih halus lagi. Akhirnya, *warp path* diproyeksikan pada resolusi penuh (1 / 1) matriks yang terlihat pada Gambar 2.5. *Warp path* yang halus ini adalah output dari algoritma tersebut.

Perhatikan bahwa *warp path* ditemukan oleh algoritma *FastDTW* padaGambar 2.5 adalah *warp path* optimal yang ditemukan oleh algoritma DTW. Namun, *FastDTW* hanya mengevaluasi sel yang berbayang, sedangkan DTW mengevaluasi semua sel dalam matriks. FastDTW dievaluasi 4+16+44+100=164 sel disemua resolusi, sementaraDTW mengevaluasi semua sel yaitu 235sel (16^2). Peningkatan efisiensi ini tidak terlalu signifikan untuk masalah kecil, terutama mengingat *overhead* menciptakan keempat resolusi. Namun jumlah sel FastDTW dievaluasi berdasarkan skala linier dengan panjang berdasarkan *time series*, sementaraDTW selalu mengevaluasi sel N^2 (jika kedua *time series* memiliki panjang N). FastDTW memiliki skala linear karena lebar jalur melalui matriks yang sedang dievaluasi adalah konstan pada semua resolusi. Algoritma *FastDTW* dapat dilihat pada Gambar 2.6.

```
Fungsi FastDTW()
Input  : X -> time series dengan panjang |X|
         Y -> time series dengan panjang |Y|
Output : 1). minimum jarak warp path antara X dan Y
         2). Warp path antara X dan Y

Integer minTSsize = radius + 2;

If(|X|≤minTSsize OR |Y|≤minTSsize)
{
     Return DTW(X,Y)
}
Else
{
     TimeSeries shrunkX = X.reduceByHalf();
     TimeSeries shrunkY = Y.reduceByHalf();

     WarpPath lowResPath = FastDTW(shrunkX,shrunkY,radius)
     SearchWindow window = ExpandedResWindow(lowResPath,X,Y,radius)
     Return DTW(X,Y,window)
}
```

Gambar 2.5 Algoritma FastDTW

Untuk menyederhanakan perhitungan terkait kompleksitas algoritma *FastDTW* akan diasumsikan bahwa dua resolusi penuh *time series* X dan Y keduanya memiliki panjang N. Semua analisis akan dilakukan pada kasus terburuk perilaku. Jumlah sel dalam *cost matriks* yang diisi oleh FastDTW dalam resolusi tunggal adalah sama dengan jumlah sel pada *warp path* yang diproyeksikan dan setiap sel-sel lain dalam radius (dinotasikan sebagai r dalam analisis untuk menghemat ruang) sel-sel jauh dari jalur yang diproyeksikan. Kasus terburuk, jaur lurus diagonal dari *warp path* yang diproyeksikan, digambarkan pada Gambar 2.7.



Gambar 2.6 Maksimum jumlah sel (kasus terburuk) yang dievaluasi pada radius 1

Sel berbayang pada Gambar 2.7 adalah sel 2Nr pada setiap sisi dari jalur yang diproyeksikan (sel dengan bayangan yang lebih lembut), memiliki 3N sel. Jalur yang diproyeksikan memiliki jumlah maksimum sel pada resolusi dengan dua *time series* yang mengandung titikN:

$$3N + 2(2Nr) = N(4r + 3) \qquad (2.4)$$

Panjang deret waktu pada setiap resolusi (res) mengikuti urutan (N poin yang terkandung dalam *time series* asli) :

$$\left\{ \frac{N}{2^{res}} \right\}_{res=0}^{res=\infty} = N, \frac{N}{2}, \frac{N}{2^2}, \frac{N}{2^3}, \frac{N}{2^4}, \ldots \qquad (2.5)$$

Oleh karena itu, jumlah sel yang dievaluasi disemua resolusi adalah (menggabungkan Persamaan 2.4 dan 2.5) :

$$\sum_{res=0}^{\infty} \frac{N}{2^{res}}(4r+3) = N(4r+3) + \frac{N}{2}(4r+3) + \frac{N}{2^2}(4r+3) + \ldots \qquad (2.6)$$

Deret pada persamaan 2.6 memiliki kesamaan dengan deret

$$\sum_{res=0}^{\infty} \frac{1}{2^{res}} = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots = 2 \qquad (2.7)$$

Lakukan proses perkalian persamaan 2.7 dengan persamaan 2.4 sehingga menghasilkan

$$N(4r+3) + \frac{N}{2}(4r+3) + \frac{N}{2^2}(4r+3) + \ldots = 2N(4r+3) \qquad (2.8)$$

Karena urutan dalam Persamaan 2.8 identik dengan urutan dalam Persamaan 2.6, jumlah sel dievaluasi di semua resolusi adalah

$$\text{Total number of cells filled} = 2N(4r+3) \qquad (2.9)$$

Selain jumlah sel dihitung ada juga kompleksitas waktu untuk menciptakan resolusi kasar dan menentukan *warp path* dengan menelusuri melalui matriks.

Kompleksitas waktu yang dibutuhkan untuk membuat resolusi sebanding dengan jumlah titik disemua resolusi, yang merupakan deret dalam Persamaan 2.5. Solusi Persamaan 2.5 diperoleh dengan mengalikan Persamaan 2.7 dengan N, yang menghasilkan 2N. Sejak beberapa resolusi kedua *time series* harus diciptakan, 2N dikalikan dua untuk mendapatkan kompleksitas waktu akhir.

$$\text{Time to create all resolutions} = 4N \quad (2.10)$$

Kompleksitas waktu yang dibutuhkan untuk menelusuri *warp path* kembali melalui matriks diukur dengan panjang *warp path*. Sebuah resolusi yang mengandung titik N memiliki panjang 2N dalam kasus terburuk (N adalah kasus terbaik untuk garis diagonal).Mengalikan Persamaan 2.7 oleh 2N memberikan panjang terburuk dari semua *warp path* ditambahkan bersama dari setiap resolusi:

$$\text{Time to trace warp path} = 4N \quad (2.11)$$

Menambahkan Persamaan 2.9, 2.10, dan 2.11 memberikan total kompleksitas waktu kasus terburuk FastDTW

$$\text{FastDTW time complexcity} = N(8r + 14) \quad (2.12)$$

Dimana O(N) jika r (radius) adalah nilai konstan kecil.

Kompleksitas ruang FastDTW terdiri dari ruang yang dibutuhkan untuk menyimpan resolusi (selain resolusi penuh input *time series*), jumlah maksimum sel yang digunakan pada satu waktu didalam *cost matriks*, dan ukuran dari *warp path* tersimpan dalam memori. Kompleksitas ruang menyimpan semua resolusi ekstra selain resolusi penuh untuk satu input *time series* adalah Persamaan 2.5 tanpa yang pertama, dimana 2N - N = N. Untuk kedua deret *time series* kompleksitas ruang adalah :

$$\text{Space of resolutions (other that full resolution)} = 2N \quad (2.13)$$

Kompleksitas ruang *cost matriks* adalah ukuran maksimum *cost matriks* yang diciptakan untuk matrik resolusi penuh . Jumlah sel dalam matriks adalah Persamaan 2.4.

$$\text{Space of cost matriks} = N(4r+3) \qquad (2.14)$$

Kompleksitas ruang penyimpanan *warp path* sama dengan *warp path* terpanjang yang terdapat diresolusi penuh. Jika jejak *warp path* perimeter dari *cost matriks*, maka panjang jalur menjadi

$$\text{Space complexity of storing the warp path} = 2N \qquad (2.15)$$

Dan menambahkan Persamaan 2.13, 2.14, dan 2 .15 memberikan total kompleksitas ruang kasus terburuk

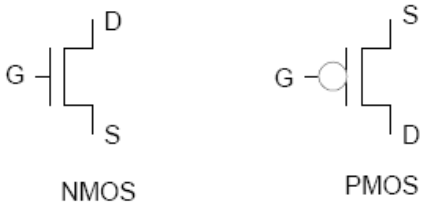
$$\text{FastDTW space complexity} = N(4r+7) \qquad (2.16)$$

Dimana ini juga O(N) jika r (radius) adalah nilai kecil (<N) konstan.

2.3 Rangkaian CMOS

Transistor bisa diilustrasikan sebagai sebuah saklar yang dikontrol dengan menggunakan listrik (arus atau tegangan). Transistor memiliki tiga buah terminal. Keadaan tersambung atau terputus pada dua terminal ditentukan oleh keadaan dari kontrol terminalnya.

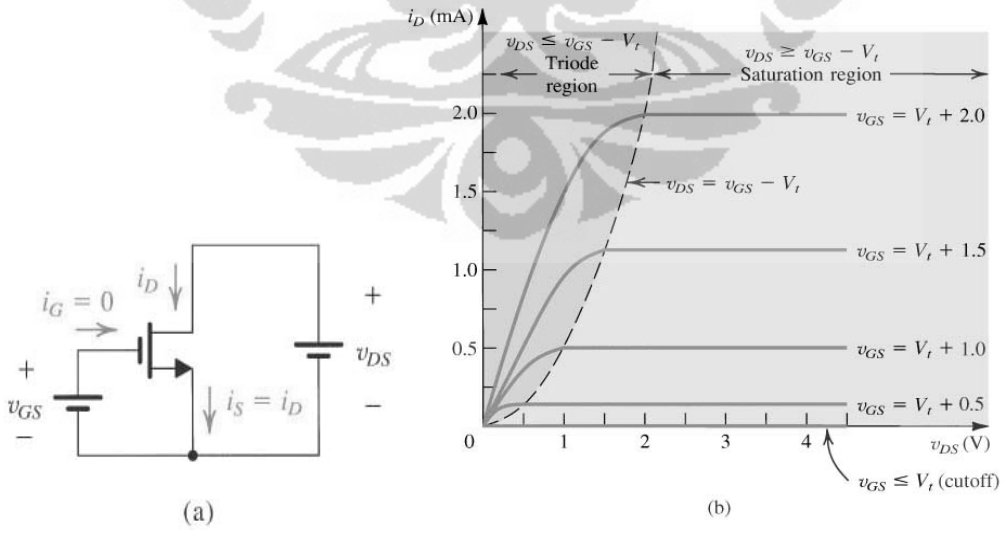
Terdapat dua tipe transistor CMOS seperti terlihat pada Gambar 2.8 yaitu: n-type Metal-Oxide-Semiconductor (NMOS) dan p-type MOS (PMOS). Rangkaian logika transistor yang menggunakan kedua type transistor tersebut dikenal dengan Complementary-MOS (CMOS). Transistor CMOS memiliki tiga terminal: *gate*, *source*, dan *drain*. Terminal *source* dan *drain* dapat ditukar satu sama lain.



Gambar 2.7 Transistor CMOS

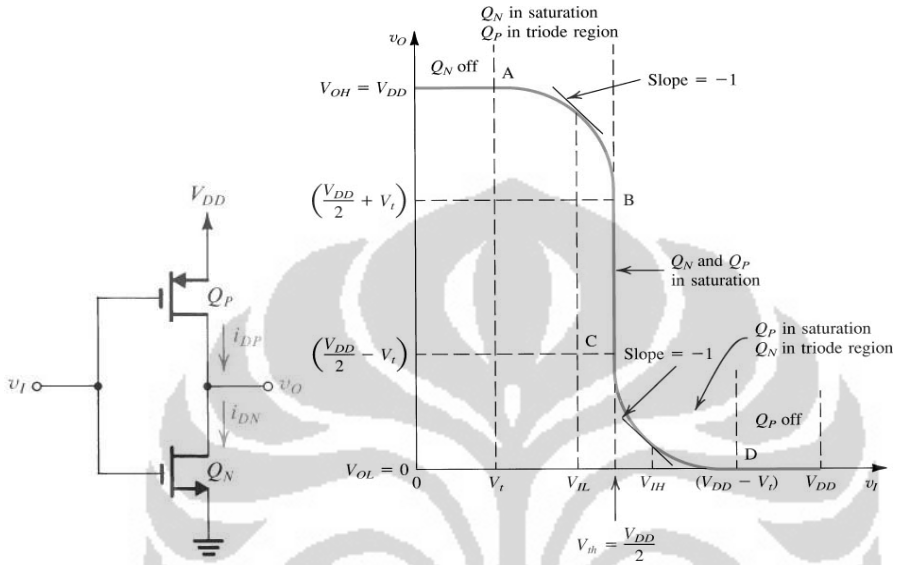
Ketika terminal *gate* dari transistor NMOS dalam kondisi *high*, maka bisa dikatakan bahwa transistor ON sehingga terdapat jalur penghantar dari *source* ke *drain*. Ketika terminal *gate* dalam kondisi *low*, maka bisa dikatakan bahwa transistor OFF maka tidak terdapat jalur penghantar dari *source* ke *drain*. Operasi PMOS berlawanan terhadap NMOS seperti diisyaratkan dengan tanda *bubble* pada terminal *gate*-nya. Ketika terminal *gate* dalam keadaan *low*, maka transistor ON dan ketika terminal *gate* dalam keadaan *high*, maka transistor OFF.

Dimana NMOS bekerja dengan memberikan tegangan positif pada gate, dan sebaliknya, PMOS bekerja dengan memberikan tegangan negatif di gate. NMOS berlaku sebagai *switch* dengan membuatnya bekerja di sekitar daerah saturasinya. Daerah kerja dari NMOS dapat dilihat pada gambar 2.9 (b).



Gambar 2.8 (a) Rangkaian karakteristik n-MOS
(b) Daerah kerja transistor n-MOS

Dan jika NMOS dan PMOS digabungkan, akan dihasilkan devais CMOS (Complementary MOS) yang rangkaian gabungan dan daerah kerjanya dapat dilihat pada gambar 2.10. Dan untuk devais CMOS ini, untuk membuatnya bekerja sebagai *switch*, kita harus mengubah-ubah daerah kerjanya antara cut-off dan saturasi.



Gambar 2.9 Rangkaian dan daerah kerja CMOS inverter

Konsumsi daya utama dari CMOS inverter berasal dari kebocoran atau switching, yang memiliki frekuensi yang sangat tinggi. Keuntungan lain dari CMOS dibandingkan NMOS adalah bahwa ia memiliki kekebalan kebisingan yang lebih tinggi, suhu chip yang lebih rendah, kisaran temperatur operasi yang lebih luas, dan kompleksitas *clocking* kurang. Terdapat beberapa parameter dalam pembahasan karakteristik listrik dari transistor CMOS dalam hal ini spesifik untuk transistor PMOS yaitu

a) Tegangan *threshold* (V_t)

Salah satu parameter fisik yang paling penting dari MOSFET adalah ambang tegangan V_t, didefinisikan sebagai tegangan gerbang dimana transistor mulai menyala. Saat ini ketetapan proses MOS menggunakan implantasi ion ke wilayah

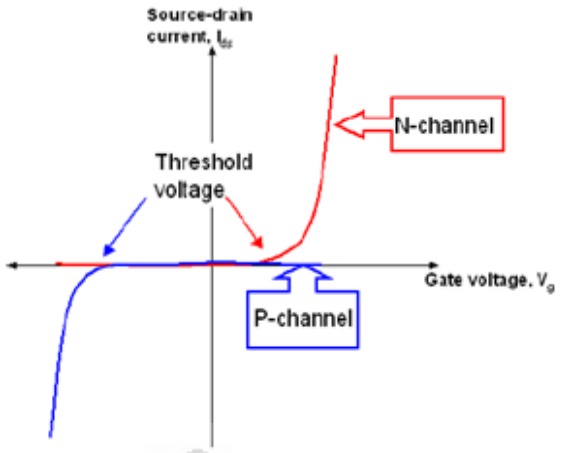
saluran, langkah yang sering disebut ambang menyesuaikan implan, yang mengubah profil doping dekat permukaan silikon substrat.

Ambang menyesuaikan implantasi adalah energi-rendah, proses implantasi dosis rendah. Ambang implantasi menentukan apakah tegangan transistor dapat diaktifkan atau dinonaktifkan, yang disebut tegangan ambang, atau V_t .Sebagai contoh, beberapa elektronik transistor lama/tua membutuhkan 12V DC power supply, sebagian besar sirkuit elektronik perlu 5V atau 3,3V, dan paling maju IC *chip* beroperasi pada 1,8V atau 1.2V. Tegangan operasi ini harus lebih tinggi dari dua kali tegangan ambang untuk memastikan transistor ini dapat diaktifkan atau dinonaktifkan, bagaimanapun juga ini tidak dapat begitu tinggi bahwa ini akan memecah gerbang oksida dan menghancurkan transistor. Dengan mengubah dosis dan energi dari ambang implan, sebuah tegangan ambang yang diinginkan tercapai[19].

Untuk mempermudah, tegangan ambang V_t adalah tegangan gerbang minimum yang diperlukan untuk membuat saluran antara *source* dan *drain*. Hal ini dapat didefinisikan sebagai tegangan minimum untuk inversi kuat terjadi. Selama operasi, kita memberikan tegangan antara *source* dan *gate* untuk membiarkan inversi terjadi sehingga arus I_{DS} dapat mengalir dari *source* ke *drain*. Nilai V_t dikontrol selama proses fabrikasi seperti tersebut di atas dan biasanya VDD/4. Gambar 2.11 menunjukkan titik tegangan ambang di arus *drain*, I_D versus tegangan *gate*, V_G grafik. Untuk PMOS, nilai negatif untuk menarik muatan positif (*hole)* dalam channel [20]. Dalam bentuk yang ideal, V_t berkaitan dengan parameter fisik sebagai berikut:

$$V_t \approx \sqrt{\frac{2\varepsilon_s q N_A (2\psi_B)}{c_{ox}}} + 2\psi_B \quad (2.17)$$

Dimana ε_s adalah permitivitas dielektrik silikon, q adalah muatan listrik, N_A adalah *impurity* (akseptor) konsentrasi dalam silikon tipe-p dan ψ_B adalah permukaan potensial pada inversi. Namun, keadaan yang sebenarnya adalah berbeda karena *interface charge* dan logam atau fungsi kerja polysilicon harus dipertimbangkan.



Gambar 2.10 Grafik arus *drain* (I_d) vs tegangan *gate* (V_g)

Pada kenyataannya tegangan ambang harus dihitung dari tegangan flat band karena perbedaan fungsi kerja dan *interface charges*,

$$V_t effective \approx V_{FB} + V_t \quad (2.18)$$

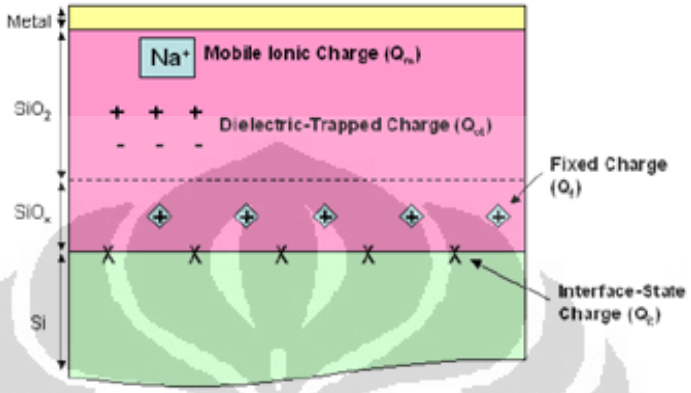
dimanaV_{FB} adalah tegangan *flat band*. V_{FB} dapat didefinisikan sebagai fungsi dari ϕ_{ms} , dimana ϕ_{ms} adalah fungsi kerja polysilicon, Q, untuk perbedaan *defect charges*, dan oksida capacitansi, C_{ox}.

$$V_{FB} = \phi_{ms} \left(\frac{Q_{it}}{c_{ox}} + \frac{Q_f}{c_{ox}} + \frac{Q_{ot}}{c_{ox}} + \frac{Q_{in}}{c_{ox}} \right) \quad (2.19)$$

Seperti yang ditunjukkan pada Gambar 2.12, *defect charges* diklasifikasikan sehubungan dengan lokasi mereka dan tindakan sebagai berikut:

1) *Interface – state charges* Q_{it}, yang terletak sangat dekat dengan interface Si-SiO2 dan memiliki keadaan energi begitu dekat dengan berbagai variasi dari E_F semikonduktor, yaitu terutama dalam celah pita silikon dilarang untuk menjadi mampu bertukar *charges* dengan semikonduktor dalam waktu singkat.
2) *Fixed charges* Q_f, yang terletak di atau sangat dekat *interface* tetapi tidak dapat menukar *charges* seperti Q_{it}.

3) *Dielectric – traped charges* Q_{ot}, situs perangkap yang didistribusikan di dalam *bulk* dielektrik dan menangkap massal (atau memancarkan) *charges* yang dibawa ke dielektrik film melalui suntikan *hot-carrier*, dan sebagainya.
4) *Mobile ionic charges* Q_m, seperti ionnatrium, yang bergerak dalam oksida dalam kondisi penuaan bias-suhu. [21]

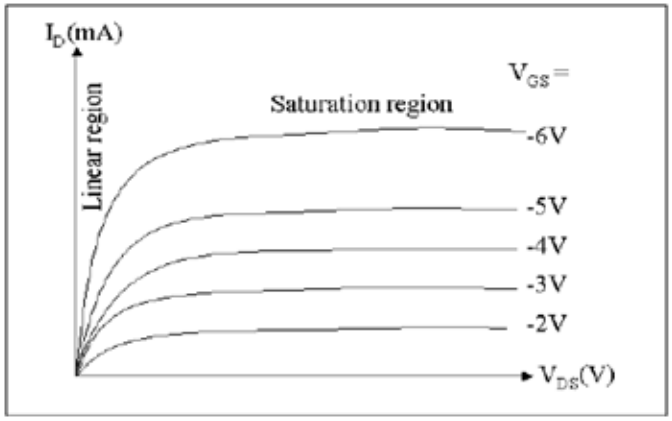


Gambar 2.11 Terminologi untuk *charges* asosiasi dengan pertumbuhan panas SiO2

Dari persamaan (2.17) dan (2.18), dalam rangka untuk mengendalikan V_t, A_N dan ϕ_{ms} nilai harus dikendalikan. Kami menggunakan implantasi ion fosfor untuk mengontrol A_N dan kami digantikan aluminium dengan polysilicon sebagai gerbang. Hal ini karena fungsi kerja untuk polisilikon untuk oksida lebih kecil daripada aluminium untuk oksida

b) Karakteristik arus dan tegangan (I-V)

Arus - Tegangan (IV) karakteristik untuk PMOS transistor dapat dibagi menjadi dua daerah, daerah linier dan daerah saturasi. Gambar 2.5 menunjukkan hubungan antara arus dan tegangan untuk PMOS dengan membaginya menjadi dua wilayah yang berbeda.



Gambar 2.12 Grafik arus *drain* (I_d) vs tegangan *drain* (V_d)

Daerah linear adalah daerah dimana arus *drain* meningkat secara linier dengan tegangan *drain* untuk setiap tegangan gerbang, nilai V_g . Daerah kejenuhan disisi lain mengacu pada daerah dimana arus *drain* hampir konstan, terlepas dari perubahan tegangan *drain*.

(a) Wilayah Linear:

Hubungan antara arus yang mengalir dari *drain* ke sumber, i_{ds} dan tegangan antara *drain* dan *source*, V_{DS} untuk wilayah linier (VDS <<(V_{gs}-V_t)) diberikan sebagai:

$$I_{ds} = \frac{W}{L}\mu_N c_{ox}\left(V_{gs} - V_t\right)V_{ds} \qquad (2.20)$$

Dimana μ_N adalah aktivasi elektron, W adalah lebar saluran, L adalah panjang saluran, V_{gs} adalah tegangan antara *gate* dan *source*, C_{ox} adalah kepadatan kapasitansi oksida per unit area yang didefinisikan sebagai:

$$c_{ox} = \frac{\varepsilon_0 \varepsilon_{ox}}{t_{ox}} \qquad (2.21)$$

Dimana ε_{ox} adalah permitivitas oksida dan t_{ox} adalah ketebalan gerbang oksida.

(b) Wilayah saturasi

Saturasi terjadi ketika$V_{DS}>V_{gs}-V_t$, situasi dimana saluran sumber tegangan, V_{DS} adalah lebih besar dari tegangan *gate-source*, V_{gs} . Arus akan mengalir terus-menerus dan tidak akan tergantung pada peningkatan V_{DS}

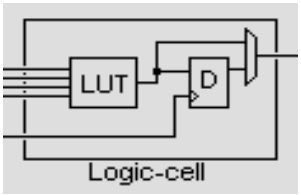
$$I_{dsat} = \frac{W \mu_N \varepsilon_{ox}}{2 t_{ox} L} \left(V_{gs} - V_t \right)^2 \qquad (2.22)$$

2.4 *Field Programmable Gate Array* (FPGA)

2.4.1 Pengertian FPGA

Field Programmable Gate Array (FPGA) merupakan komponen silicon seperti halnya *Integrated Circuit* (IC) yang fungsi dan rancangannya dapat dikonfigurasi oleh pengguna atau perancang dengan menggunakan bahasa *Hardware Description Language* (HDL). FPGA dapat dikonfigurasikan untuk mengimplementasikan fungsi logika apapun yang dapat dilakukan oleh suatu *Application-specific IC* (ASIC). [18]

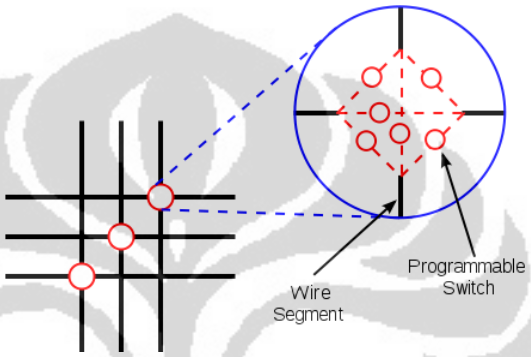
Karakteristik dari FPGA antara lain adalah dapat dirancang sesuai dengan keinginan dan kebutuhan pemakai tanpa melalui tahap "*burn*" di laboratorium atau di "*hardwire*" oleh perusahaan piranti, hal tersebut mungkin dilakukan karena FPGA terdiri atas sekumpulan *Configurable Logic Blocks* (CLB) yang terhubung melalui *Programmable Interconnects* (PI). Pada umumnya setiap CLB terdiri dari terdiri dari beberapa *Logic Cells* yang kadang disebut juga dengan *Slice*, yang masing-masing terdiri dari 4-input *Lookup Table* (LUT), D Flip-Flop dan 2-to-1 Mux, seperti pada gambar 2.14 dibawah ini.



Gambar 2.13 Sebuah Logic Cell pada FPGA [18]

Konfigurasi CLB dalam FPGA dapat berbeda-beda, tergantung dari manufaktur dan varian FPGA yang digunakan, perbedaannya termasuk jumlah *inputs* dan *outputs*, kompleksitas rangkaian CLB dan jumlah transistor yang digunakan. Jadi kemampuan untuk mengimplementasikan fungsi logika disediakan oleh CLB ini.

Setiap *Logic Cell* dapat dihubungkan dengan *Logic Cell* lainnya melalui *Programmable Interconnect* (PI) yang akan membentuk suatu fungsi logika yang compleks, ilustrasi dari PI ditunjukkan pada gambar 2.15 dibawah ini. [18]



Gambar 2.14 Programmable Interconnects [18]

2.4.2 Arsitektur FPGA

Sebuah IC FPGA terdiri 3 (tiga) komponen pendukung utamanya yakni [18]:

- *Configurable Logic Block* (CLB)

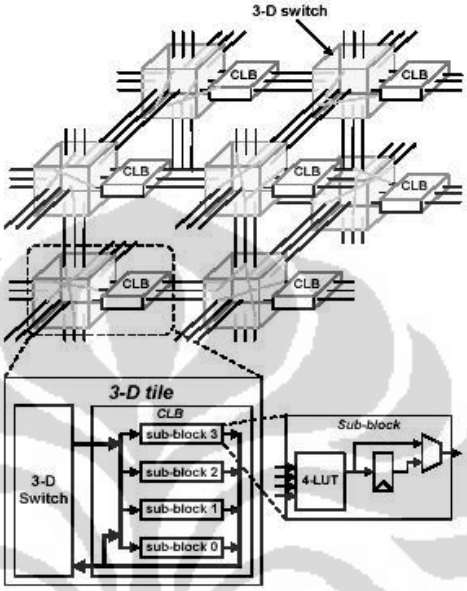
  CLB merupakan komponen dasar yang membentuk IC FPGA berupa matrik-matrik yang saling terhubung oleh PI, yang dapat dikonfigurasikan untuk melakukan rangkaian logika kombinasional, *shift register*, atau *Random Access Memory* (RAM).

- *Input Output Block* (IOB)

  IOB merupakan penghubung atau sebagai antarmuka antara pin-pin terminal IC FPGA dengan kawat penghubung atau jalur-jalur koneksi di luar IC, IOB dikelompokkan ke dalam beberapa *I/O Bank* yang sesuai dengan standar I/O.

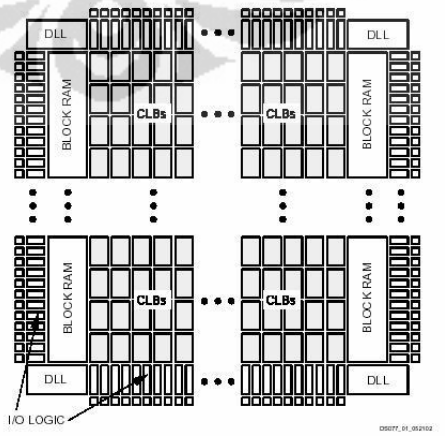
- *Programmable Interconnect* (PI).

PI merupakan komponen yang berperan sebagai kawat atau sakelar penghubung yang dapat dikonfigurasikan dan mengelilingi blok-blok CLB, yang akan menghubungkan antar blok-blok CLB maupun dengan IOB.



Gambar 2.15 Arsitektur Dasar FPGA [16]

Selain CLB, IOB dan PI, beberapa IC FPGA juga dilengkapi dengan elemen-elemen lain seperti RAM dan Delay-Locked Loop (DLL) pada FPGA Spartan II series.

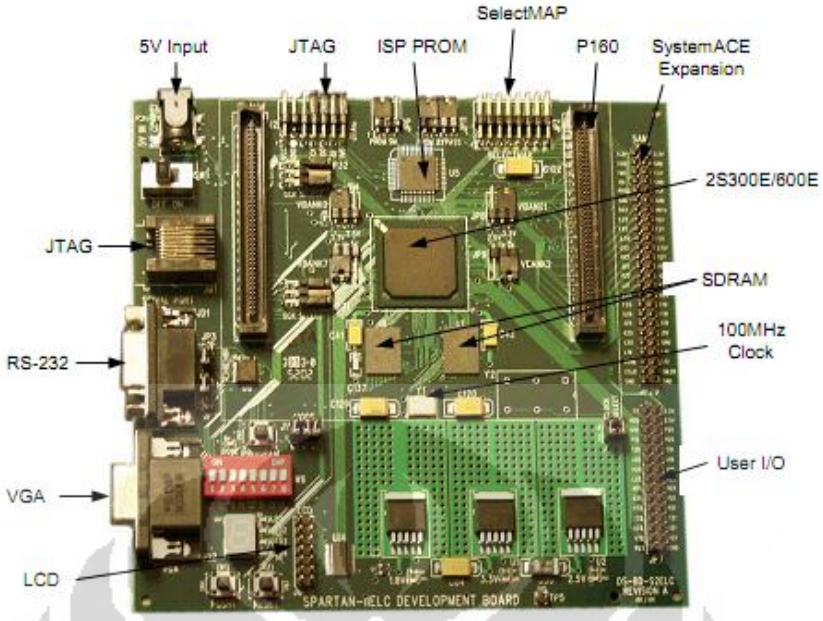


Gambar 2.16 Arsitektur FPGA Spartan II Series [15]

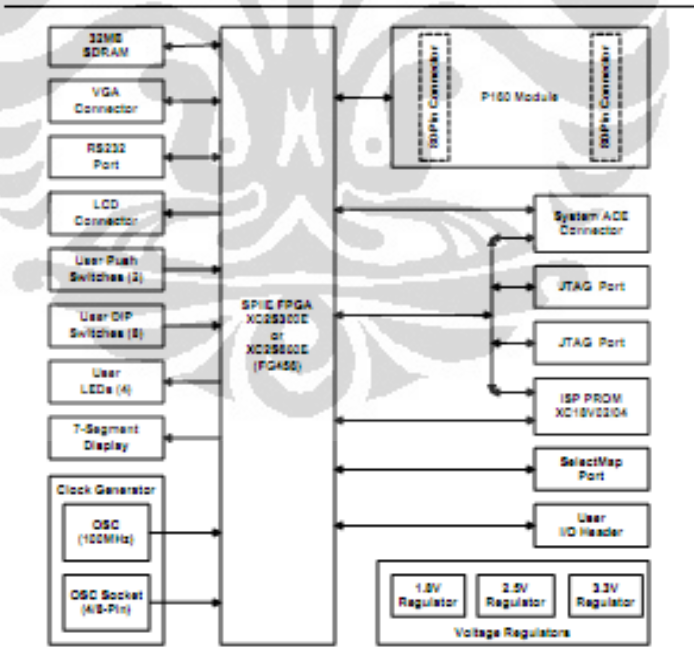
2.4.3 Spesifikasi FPGA Xilinx Spartan-IIE LC Development Board.

Spesifikasi FPGA Xilinx Spartan-IIE LC Development Board yang digunakan pada tugas akhir ini adalah sebagai berikut [17]:

a. Memiliki FPGA Xilinx Spartan-IIE XC2S300E-6FG456C dengan 300.000 gerbang logika, yang terdiri dari 3072 buah CLB, 6144 buah Flip-Flop, 6144 buah 4-input LUT dan 329 *I/O Block* .
b. Internal Clock source : 100 MHz
c. SDRAM 32 Mb.
d. Antarmuka simple "R-2R" *resistor-ladder* VGA
e. Konektor DB9 RS232 sederhana hanya untuk pin Td dan Rd dengan IC MAX3221 dari Texas Instruments.
f. konektor LCD 8-bit 2x16 karakter
g. 4 buah LED dan 1 (satu) buah seven segment display
h. 8 bit user DIP switch.
i. 82 pin *header* yang dapat digunakan oleh pengguna melalui 2 (dua) buah *on-board* header.
j. P160 *Expansion Module Standard*
k. System ACE *Expansion* untuk aplikasi *Real Time Operating System* (RTOS) dari eksternal *CompactFlash*
l. Antarmuka RJ45 untuk menggugah *file bitstream* konfigurasi.
m. XC18V02 ISP PROM untuk menyimpan konfigurasi.



Gambar 2.17 Xilinx FPGA Spartan-IIE LC Development Board [17]



Gambar 2.18 Blok Diagram Spartan-IIE LC Development Board [9]

BAB 3

PERANCANGAN DAN HASIL SIMULASI LAYOUT CMOS IC DTW

3.1 Metode Penelitian

Metode penelitian yang digunakan adalah metode kajian kepustakaan dan metode eksperimen. Metode kajian kepustakaan dilakukan dengan studi literatur berdasarkan sumber kepustakaan dan penelitian – penelitian terkait yang dilakukan sebelumnya. Adapun metode eksperimen dilakukan melalui proses perancangan, proses implementasi dan proses percobaan terhadap input suara.

Proses perancangan dilakukan melalui pembuatan layout CMOS IC *pattern matching* yang menggunakan algoritma DTW. Tahapan yang dilakukan adalah :

1. Pembuatan arsitektur untuk IC DTW berupa blok diagram yang menggambarkan proses pencocokan pola menggunakan algoritma DTW.
2. Pembuatan layout CMOS *standar cell* penyusun layout CMOS IC DTW.
3. Melakukan simulasi pengukuran terhadap karakterisasi tegangan dan estimasi performa dari layout CMOS *standar cell*.
4. Pembuatan layout CMOS *full datapath* IC DTW. Layout CMOS *full datapath* IC DTW merupakan integrasi dari seluruh layout CMOS *standar cell* yang berfungsi sebagai algoritma DTW untuk pencocokan pola.
5. Melakukan pengujian terhadap layout CMOS *full datapath* IC DTW. Proses ini dilakukan melalui perbandingan hasil perhitungan algoritma DTW yang didapatkan pada layout CMOS *full datapath* IC DTW dengan hasil perhitungan algoritma DTW yang didapatkan pada aplikasi JAVA yang dibuat Philip Chan dan Stan Salvador [13].
6. Pembuatan *padframe* dari layout CMOS *full datapath* IC DTW.

Proses implementasi dilakukan dengan mengimplementasikan hasil perancangan yang dibuat, pada sebuah *board* FPGA Spartan – IIELC. Tahapan yang dilakukan adalah :

1. Pembuatan kode program VHDL IC DTW berdasarkan *behavior* dari layout CMOS *standar cell* penyusun layout CMOS IC DTW yang dibuat pada proses perancangan.
2. Pembuatan diagram skematik IC DTW pada FPGA Spartan – IIELC berdasarkan kode program VHDL.
3. Melakukan simulasi fungsional IC DTW pada FPGA Spartan – IIELC berdasarkan kode program VHDL.
4. Melakukan implementasi kode program VHDL pada FPGA Spartan – IIELC. Kode program VHDL yang diimplementasikan adalah kode program VHDL IC DTW dan kode program VHDL untuk komunikasi serial RS232. Hasil implementasi berupa modul *feature matching* berbasis FPGA Spartan – IIELC.
5. Melakukan pengujian terhadap proses implementasi. Proses ini dilakukan melalui perbandingan hasil perhitungan algoritma DTW yang didapatkan pada modul *feature matching* berbasis FPGA Spartan – IIELC dengan hasil perhitungan algoritma DTW yang didapatkan pada aplikasi JAVA yang dibuat Philip Chan dan Stan Salvador [13].

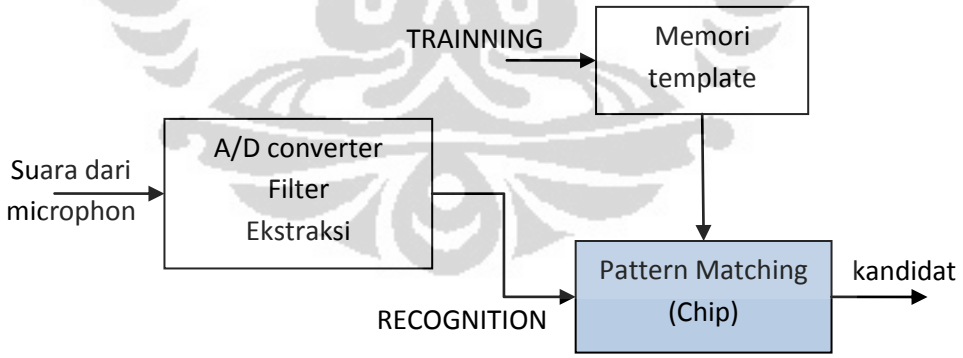
Proses percobaan pengenalan suara dilakukan melalui input suara ‘a’, suara ‘b’, suara ‘c’, suara ‘d’ dan suara ‘e’ yang dibandingkan dengan *feature* suara menggunakan modul *feature matching* berbasis FPGA Spartan – IIELC. Tahapan yang dilakukan adalah :

1. Pembuatan modul *feature extracting* berdasarkan penelitian yang sudah ada [27]. Modul tersebut berfungsi melakukan proses ekstraksi suara input menjadi vektor data digital.
2. Melakukan modifikasi terhadap modul *feature matching* hasil proses implementasi. Hal ini dilakukan terkait input vektor data digital hasil ekstraksi suara ‘a’, suara ‘b’, suara ‘c’, suara ‘d’ dan suara ‘e’ serta penyimpanan *feature* suara modul *feature matching*.
3. Melakukan proses integrasi modul *feature extracting* dengan modul *feature matching*.
4. Melakukan setting konfigurasi alat untuk percobaan pengenalan suara.

5. Melakukan percobaan pengenalan suara dan menghitung estimasi keberhasilan dari percobaan pengenalan suara.

3.2 Arsitektur IC DTW

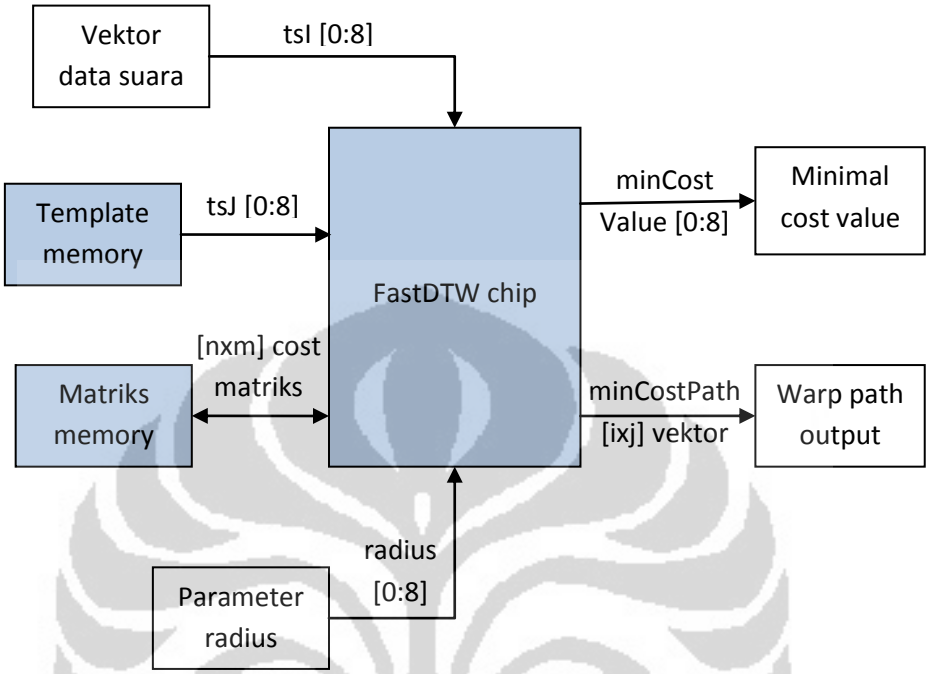
Pada proses sistem pengenalan suara terdapat dua tahap yaitu proses pendaftaran suara dan proses pengenalan suara [4]. Gambar 3.1 memperlihatkan blok diagram sistem pengenalan suara yang terdiri dari blok analog – digital konverter dan filter, blok memori dan blok pencocok pola. Proses pendaftaran suara akan menerima suara untuk dilakukan proses ekstraksi sehingga menghasilkan data *trainning* suara yang disimpan pada memori template. Proses pencocokan suara menerima input suara dari *microphone* dan diolah pada blok analog – digital konverter, filter dan ekstraksi. Hasil pengolahan suara tersebut dibandingkan dengan data *trainning* suara yang tersimpan pada memori template untuk melihat tingkat kecocokannya. Dalam penelitian ini akan dibahas mengenai perancangan layout CMOS DTW yang digunakan untuk pencocokan pola pada sistem pengenalan suara.



Gambar 3.1 Blok diagram sistem pengenalan suara

Seperti telah dibahas pada bab sebelumnya, bahwa algoritma yang digunakan untuk pencocokan pola adalah algoritma DTW dengan tingkat kompleksitas ruang sel

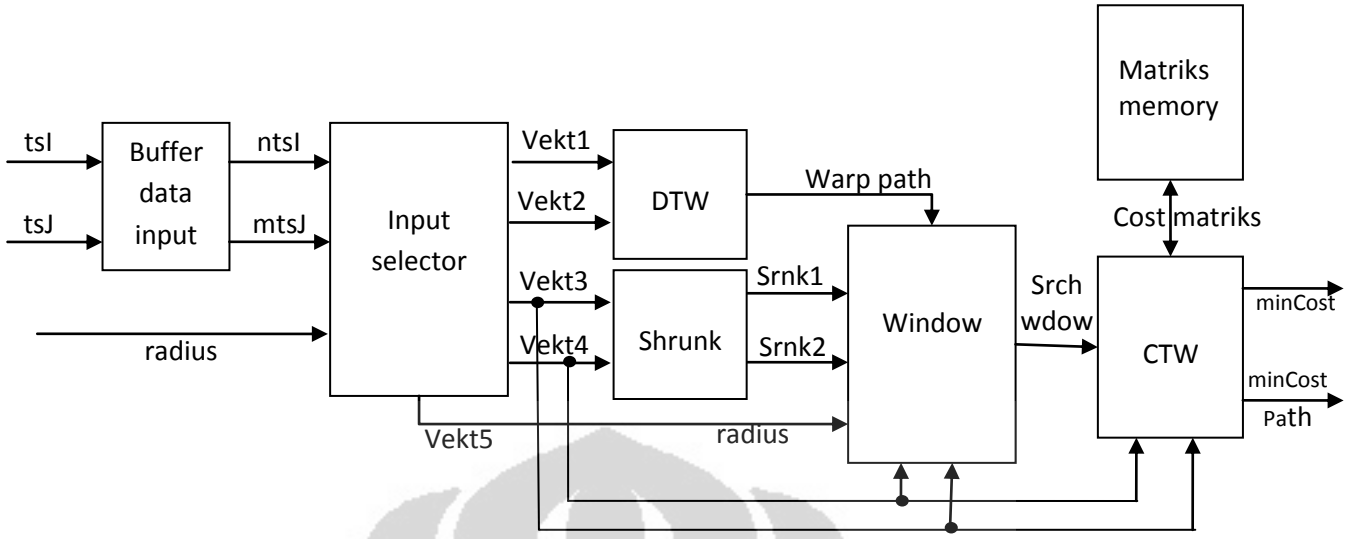
yang lebih kecil dan kompleksitas waktu yang lebih cepat yaitu FastDTW [13]. Adapun sub sistem perancangan IC FastDTW dapat dilihat pada Gambar 3.2.



Gambar 3.2 Sub sistem IC FastDTW

Pada Gambar 3.2 terdapat tiga sub sistem dari IC FastDTW yaitu memori template yang digunakan sebagai penyimpan data *trainning* suara, memori matriks yang digunakan sebagai penyimpan *cost* matriks dan IC FastDTW itu sendiri. Adapun vektor data suara dan parameter radius merupakan input dari IC FastDTW, sedangkan nilai minimal *cost* dan *warp path* minimal *cost* merupakan output dari IC FastDTW.

Detail mengenai sub sistem dari IC FastDTW dapat dilihat pada Gambar 3.3 mengenai datapath IC FastDTW level 1. Vektor data suara tsI 8 bit berasal dari input suara melalui *microphone* dan vektor data suara tsJ 8 bit berasal dari memori template ditampung pada blok buffer data input. Pada blok buffer data input vektor data suara tsI dan tsJ ditampung sementara dalam suatu register sesuai banyaknya paket data yang masuk yaitu sebanyak [n x 8] bit untuk tsI dan [m x 8] bit untuk tsJ.

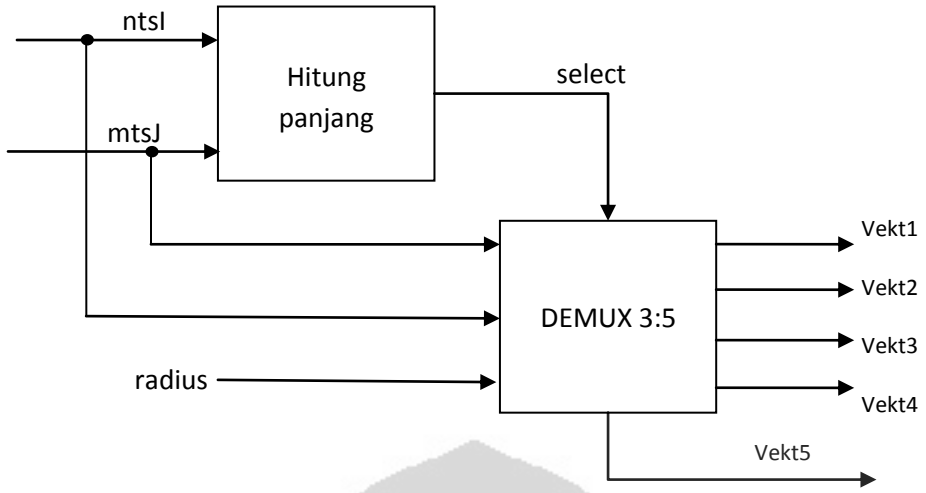


Gambar 3.3 Datapath IC FastDTW (level 1)

Output dari blok Buffer Data Input menjadi ntsI untuk vektor data suara tsI dan mtsJ untuk vektor data suara tsJ. Kemudian ntsI, mtsJ dan radius masuk pada blok Input Selector untuk menghasilkan vektor 1, 2, 3, 4 dan 5. Vektor 1 dan vektor 2 masuk pada blok DTW untuk menghasilkan *warp path* . Vektor 3 dan vektor 4 masuk pada blok Shrunk untuk menghasilkan nilai *shrunk* 1 dan *shrunk* 2.

Proses selanjutnya *warp path*, *shrunk* 1, *shrunk* 2, nilai radius, vektor 3 dan vektor 4 masuk pada blok Window untuk menghasilkan *search window*. Tahapan terakhir adalah blok CTW yang menerima masukan berupa vektor 3, vektor 4 dan *search window* dan terhubung pada blok Matriks Memori untuk melakukan *update* nilai dari *cost matriks* dan menghasilkan nilai *minimal cost* dan *minimal cost path*.

Penjelasan secara detail untuk blok Input Selector, blok DTW, blok Shrunk dan blok CTW akan dijelaskan selanjutnya disertai datapath dari masing – masing blok tersebut.

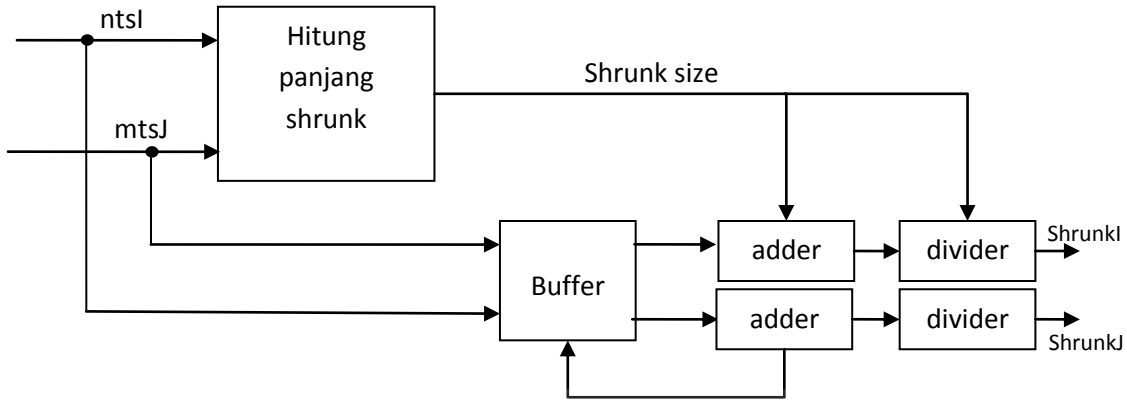


Gambar 3.4 Datapath Input Selector (level 2)

Pada blok Input Selector terdapat datapath untuk menghitung panjang vektor data suara dan *demultiplexing* dari 3 input menjadi 5 output sesuai dengan logika *select*. Nilai minimal untuk panjang vektor data suara adalah 1 baik ntsI maupun mtsJ. Hasil pehitungan panjang untuk kedua vektor tersebut disimpan pada register *select* yang akan mengontrol input dan output pada datapath *demultiplexing*. Logika perubahan output dari input berdasarkan register *select* dapat dilihat pada Tabel 3.1.

Tabel 3.1 Logika perubahan output

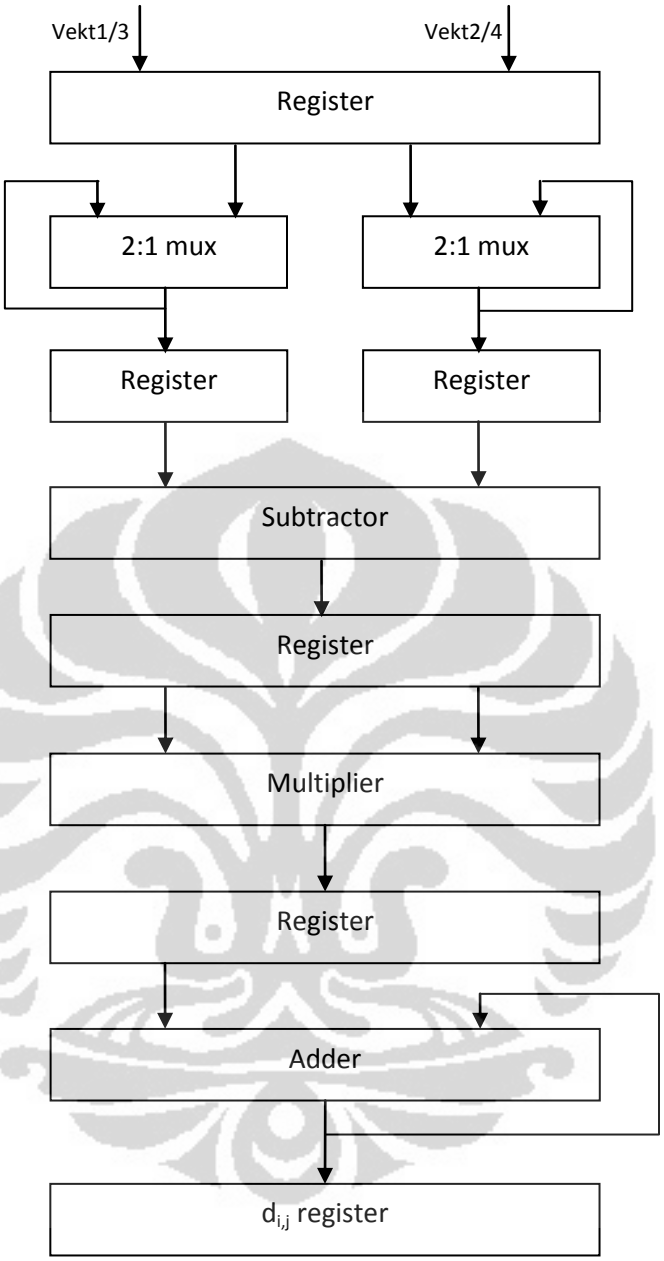
Input	*select*	Output				
		Vektor 1	Vektor 2	Vektor 3	Vektor 4	Vektor 5
(ntsI, mtsJ dan radius)	≤ 2	ntsI	mtsJ	0	0	radius
(ntsI, mtsJ dan radius)	> 2	0	0	ntsI	mtsJ	radius



Gambar 3.5 Datapath Shrunk (level 2)

Proses yang dilakukan pada blok Shrunk sederhananya untuk mencari nilai shrunk dengan cara memperkecil ukuran data vektor yang masuk dan mencari nilai rata – rata dari vektor tersebut untuk mengisi register *shrunk* yang ukurannya sudah diperkecil sehingga mewakili semua nilai pada vektor data yang masuk. Tahapan yang dilakukan dengan melakukan operasi penjumlahan dan pembagian untuk setiap vektor data suara yang masuk berdasarkan ukuran dari *shrunk*, jika ukuran *shrunk* $\neq 2$ maka proses penjumlahan dan pembagian kembali dikerjakan hingga ukuran *shrunk* menjadi 2.

Berikut diberikan contoh perhitungan, misalkan ntsI = [1,2,3,4] dan mtsJ = [4,5,6,7] maka ukuran shrunk yang diharapkan untuk shrunkI dan shrunkJ adalah $\frac{panjangshrunk}{2} = \frac{4}{2} = 2$. Sehingga nilai shrunkI = $\left[\frac{1+2}{2}, \frac{3+4}{2}\right] = [1.5, 3.5]$ dengan ukuran shrunkI = 2 dan nilai shrunkJ = $\left[\frac{4+5}{2}, \frac{6+7}{2}\right] = [4.5, 6.5]$ dengan ukuran shrunkJ = 2.

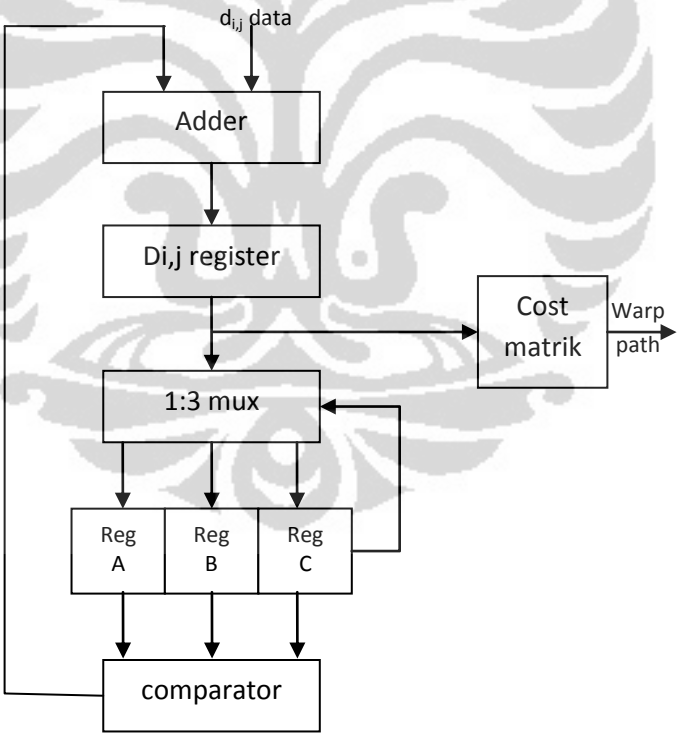


Gambar 3.6 Datapath DTW/CTW Tahap 1 (level 2)

Gambar 3.6 merupakan datapath DTW atau CTW tahap 1, artinya bahwa datapath ini mendeskripsikan proses DTW atau CTW untuk tahap menghasilkan register $d_{i,j}$. karena pada tahap ini baik proses DTW maupun proses CTW meliki tahapan yang

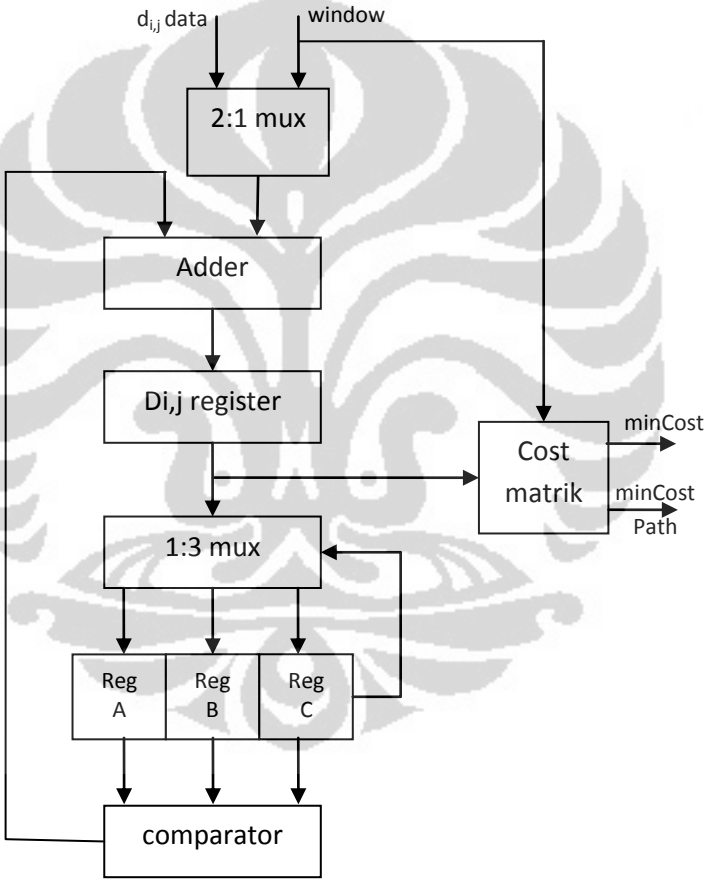
sama maka perancangan datapathnya diintegrasikan pada tahap 1. Perbedaan proses DTW dan CTW akan dijelaskan untuk masing – masing proses pada tahap 2.

Pada tahap 1 masukan untuk vektor 1 atau 3 dan vektor 2 atau 4 ditampung dalam suatu register. Kemudian dilakukan proses *multiplexing* 2 input menjadi 1 output, hal ini dilakukan untuk menjaga agar semua vektor data yang masuk tertampung diregister untuk dioperasikan dengan vektor data lain yang masuk. Vektor – vektor data tersebut kemudian dilakukan operasi pengurangan dan hasilnya disimpan pada suatu register. Data yang tersimpan dalam register dilakukan operasi perkalian terhadap dirinya sendiri (operasi perpangkatan 2) dan hasilnya akan dilakukan operasi penjumlahan terhadap hasil lainnya pada operasi perkalian tersebut. Hasil dari operasi inilah yang disimpan pada register $d_{i,j}$ yang disebut hasil proses perhitungan *euclidien*.



Gambar 3.7 Datapath DTW Tahap 2 (level 2)

Gambar 3.7 memperlihatkan proses yang terjadi pada datapath DTW tahap 2. Tahap ini mendeskripsikan perhitungan nilai $D_{i,j}$ yang didapatkan dari penjumlahan $d_{i,j}$ dengan nilai minimum($D_{i-1,j-1}$, $D_{i-1,j}$, $D_{i,j-1}$) sehingga dibutuhkan operasi penjumlahan dan operasi comparator untuk "nilai lebih kecil dari". Nilai – nilai $D_{i,j}$ yang dihasilkan disimpan pada *cost matriks* dan dilakukan pencarian terhadap *warp path* nya. Output dari tahap 2 proses DTW hanyalah berupa *warp path* DTW dari dua buah vektor data yang masuk.



Gambar 3.8 Datapath CTW Tahap 2 (level 2)

Terhadap perbedaan tahapan untuk proses CTW tahap 2 sesuai gambar 3.8 dibandingkan dengan DTW tahap 2. Perbedaan yang mendasar adalah adanya nilai window pada proses CTW yang mempengaruhi prosesnya. Dengan adanya nilai

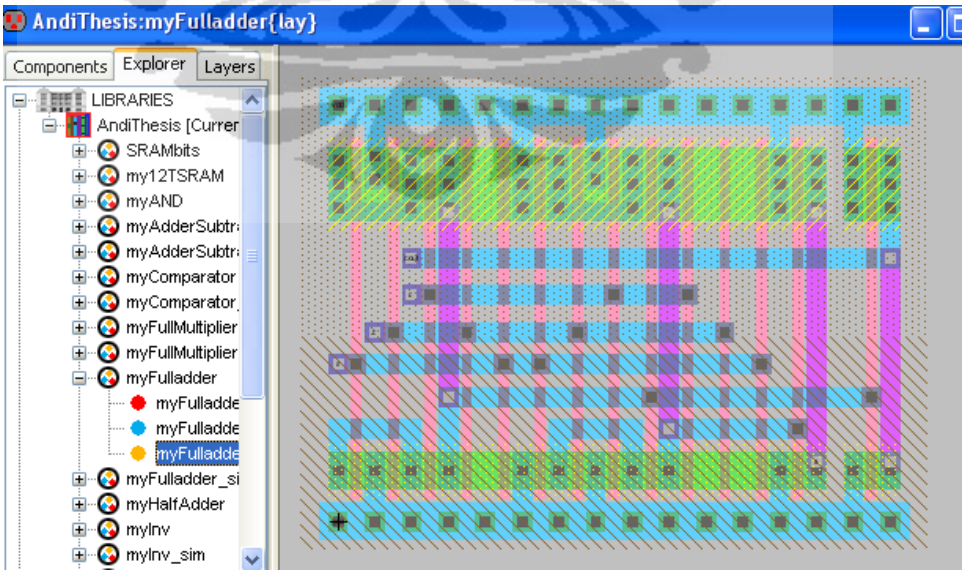
window, proses perhitungan nilai $D_{i,j}$ yang didapatkan dari penjumlahan $d_{i,j}$ dengan nilai minimum($D_{i-1,j-1}$, $D_{i-1,j}$, $D_{i,j-1}$) tidak dilakukan untuk semua nilai i dan j melainkan sesuai dengan nilai windownya. Begitupun nilai – nilai $D_{i,j}$ yang disimpan pada *cost matriks*, indeks matriks disesuaikan dengan nilai windownya. Output dari tahap 2 proses CTW adalah nilai minimum dari *cost matriks* dan *warp path* dari *cost matriks* tersebut.

3.3 Standar Cell Layout CMOS Penyusun IC DTW

Pada sub bab ini dibahas mengenai *standar cell* layout CMOS penyusun IC DTW diantaranya yaitu standar cell layout CMOS Full Adder, Full Subtractor, Full Multiplier dan Comparator.

3.3.1 Standar Cell Layout CMOS Full Adder

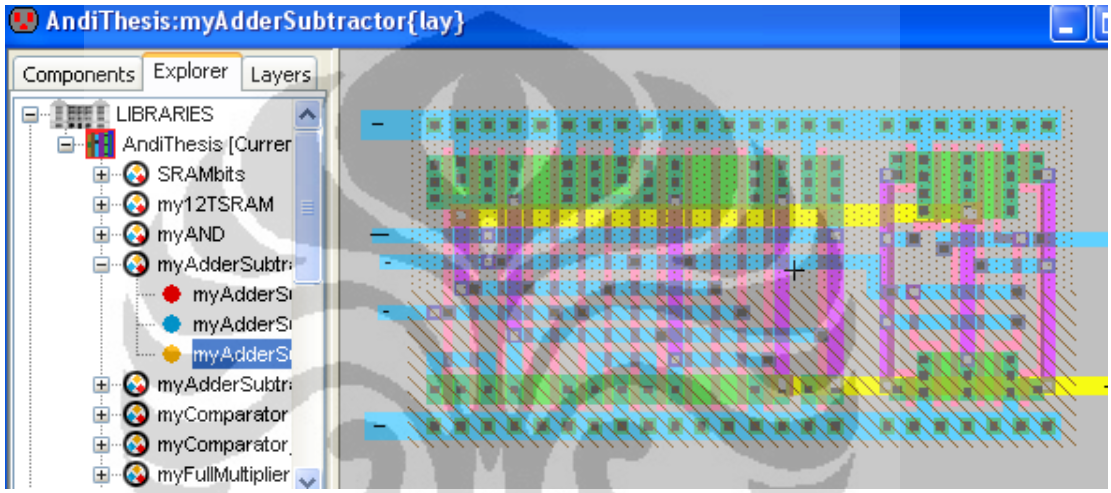
Full adder adalah rangkaian yang digunakan untuk menjumlahkan bilangan-bilangan biner yang lebih dari satu bit. Rangkaian ini terdiri dari 3 terminal input (A,B dan *carry in* (C_{in})) dan 2 terminal output yaitu summary out (SUM) dan *carry out* (C_{out}). *Standar cell* layout CMOS untuk rangkaian full adder 1 bit yang dapat dilihat pada Gambar 3.9. Sedangkan layout CMOS untuk rangkaian full adder 8 bit terbentuk dari 8 buah full adder 1 bit dan dapat dilihat pada Gambar 1 dan 2 pada lampiran 1.



Gambar 3.9 Layout CMOS Rangkaian Full Adder

3.3.2 Standar Cell Layout CMOS Full Subtractor

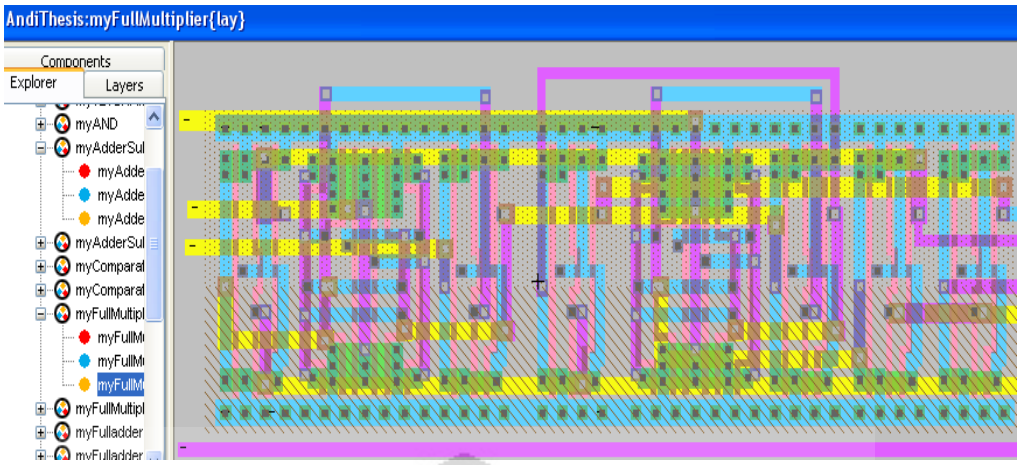
Full subtractor adalah rangkaian yang digunakan untuk mengurangkan bilangan – bilangan biner yang lebih dari satu bit. Rangkaian ini terdiri dari 3 terminal input (X, Y danB_{in}) dan 2 terminal output (d dan B_{out}). *Standar cell* yang dirancang untuk layout CMOS rangkaian full subtractor 1 bit dapat dilihat pada Gambar 3.10. Sedangkan layout CMOS untuk rangkaian full subtractor 8 bit terbentuk dari 8 buah full subtractor 1 bit dan dapat dilihat pada Gambar 1 dan 2 pada lampiran 2.



Gambar 3.10 Layout CMOS Rangkaian Full Subtractor

3.3.3 Standar Cell Layout CMOS Full Multiplier

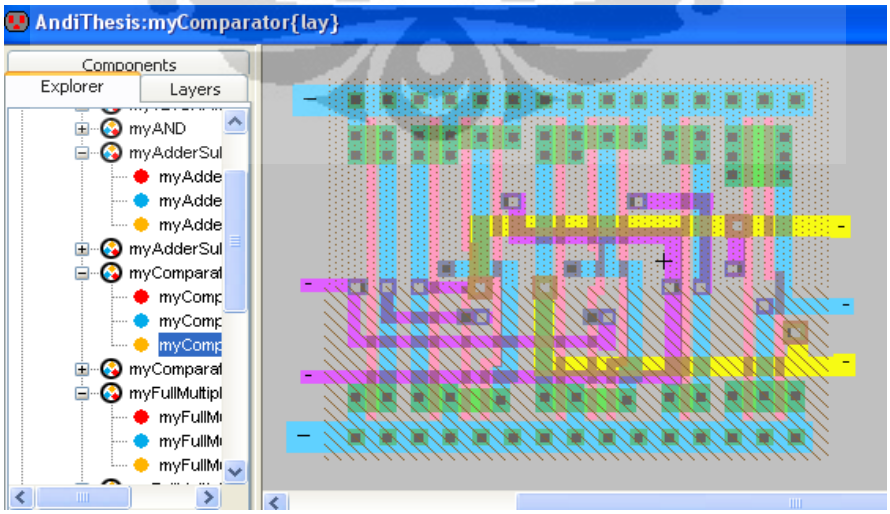
Full multiplication adalah rangkaian yang digunakan untuk mengalikan bilangan-bilangan biner yang lebih dari satu bit. Rangkaian ini pada dasarnya merupakan rangkain full adder yang berulang tergantung inputan. *Standar cell* yang dirancang untuk layout CMOS rangkaian full multiplicator 2 bit dapat dilihat pada Gambar 3.11. Sedangkan layout CMOS untuk rangkaian full multiplicator 8 bit dapat dilihat pada Gambar 1 dan 2 pada lampiran 3.



Gambar 3.11 Layout CMOS Rangkaian Full Multiplicator

3.3.4 Standar Cell Layout CMOS Comparator

Digital comparator atau pembanding biner terdiri dari gerbang standar AND, NOR dan NOT yang membandingkan sinyal digital pada terminal input dan menghasilkan output tergantung pada kondisi input tersebut. Tujuan dari komparator digital adalah untuk membandingkan satu set variabel terhadap suatu variabel lainnya dan menghasilkan nilai perbandingan. Sebagai contoh variabel A (A_1, A_2, A_3 ... A_n) dan B (B_1, B_2, B_3 ... B_n) akan menghasilkan output A > B , A = B atau A < B. *Standar cell* yang dirancang untuk layout CMOS rangkaian comparator 1 bit dapat dilihat pada Gambar 3.12. Sedangkan layout CMOS untuk rangkaian comparator 8 bit dapat dilihat pada Gambar 1 dan 2 pada lampiran 4.



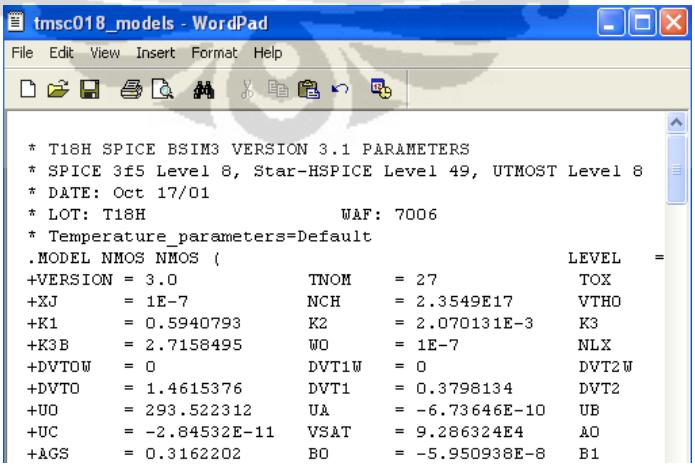
Gambar 3.12 Layout CMOS Rangkaian Comparator

3.4 Karakterisasi dan Estimasi Performa Standar Cell Penyusun IC DTW

Pada sub bab ini akan dijelaskan karakterisasi dan estimasi performa dari layout CMOS yang telah dirancang. Pembahasan karakterisasi difokuskan pada karakterisasi tegangan dan arus dari layout CMOS sedangkan pembahasan estimasi performa lebih difokuskan pada *delay* waktu keluarnya output terhadap input dari layout CMOS. Model yang digunakan adalah BSIM3 versi 3.

BSIM3v3 adalah pemodelan *deep-submicron* MOSFET berbasis *physics* terbaru untuk desain rangkaian digital dan analog dari Grup *Device* di University of California di Barkeley. Pembuatan BSIM3v3 adalah berdasarkan pencarian solusi bagi persamaan Poisson dengan menggunakan pendekatan *Gradual Channel* (GCA) dan pendekatan dimensi *Quasi-Two* (QTDA).[22]

Teknologi yang digunakan dalam pembuatan layout CMOS yaitu MOSIS CMOS 0,18 micron. MOSIS adalah prototipe murah dan layanan untuk produksi dengan volume kecil untuk pembangunan sirkuit VLSI. Sejak 1981, MOSIS telah membuat desain sirkuit lebih dari 50.000 untuk perusahaan komersial,instansi pemerintah, dan lembaga penelitian dan pendidikan di seluruh dunia [23]. Dalam melakukan simulasi terhadap karakterisasi arus dan tegangan dari layout CMOS yang dirancang digunakan parameter *Taiwan Semiconductor* 0,18 micron (TMSC018) yang termasuk salah satu desain pabrikasi yang ada pada *MOSIS Educational Program* (MEP) [24].



Gambar 3.13 TMSC018 Parameter Model

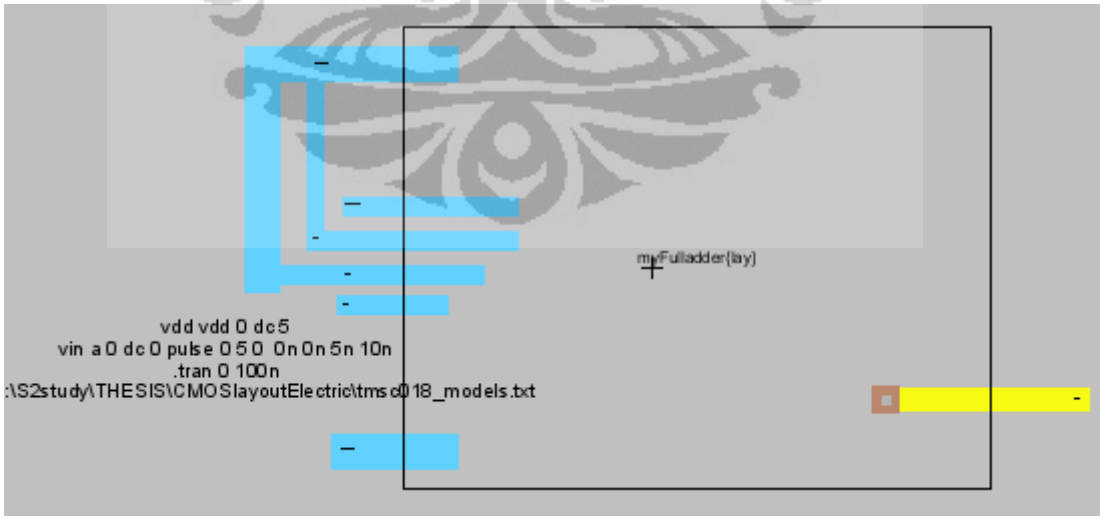
Gambar 3.13 memperlihatkan parameter TMSC018 yang digunakan dalam pembuatan simulasi karakterisasi arus dan tegangan layout CMOS pada penelitian ini. Dalam pembuatan simulasi karakterisasi arus dan tegangan layout CMOS juga digunakan software simulasi LTSpice [25] sedangkan pembuatan simulasi estimasi performa *delay* digunakan software CAD Electric dengan metode IRSIM [26].

3.4.1 Karakterisasi dan Estimasi Performa Layout CMOS Full Adder

Pada simulasi karakterisasi arus dan tegangan dari Full Adder 1 bit, terdapat beberapa setting parameter awal yang ditentukan yaitu

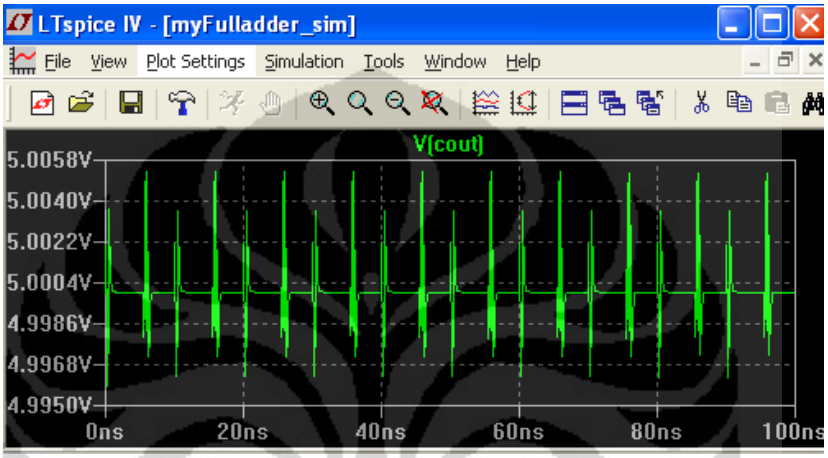
a. *Power supply* dalam hal ini vdd diberikan tegangan DC 5V
b. Input b dan c_{in} dihubungkan pada vdd, hal ini menyebabkan tegangan pada input b dan c_{in} stabil dengan nilai 5V.
c. Input a merupakan tegangan pulsa 0 dan 5V dimana lebar tegangan masing – masing 5ns.
d. Parameter model yang digunakan yaitu tmsc018_models.txt yang merupakan parameter TMSC018.
e. Simulasi analisis *transient* berjalan pada 0 – 100ns.

Gambar 3.14 memperlihatkan setting parameter awal layout CMOS Full Adder 1 bit untuk melakukan simulasi karakterisasi arus dan tegangan.



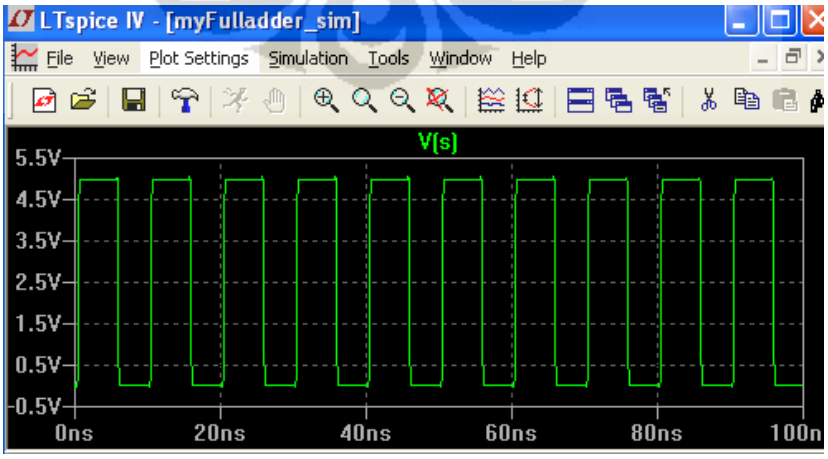
Gambar 3.14 Setting parameter simulasi karakterisasi I – V dari Full Adder

Dari hasil simulasi karakterisasi arus dan tegangan Full Adder 1 bit menggunakan LTSpice didapatkan tegangan untuk c_{out} sebesar 5V, adapun perubahan tegangan terjadi pada *range* tegangan 4,9968V – 5,0058V sesuai Gambar 3.15. Adapun munculnya *range* tegangan tersebut dipengaruhi perubahan tegangan input a dari 5V menjadi 0V atau sebaliknya sehingga terlihat pada periode waktu tertentu yang sangat singkat besar tegangan 5V berubah menjadi tegangan dengan *range* 4,9968V – 5,0058V.



Gambar 3.15 Tegangan pada output c_{out} dari Full Adder

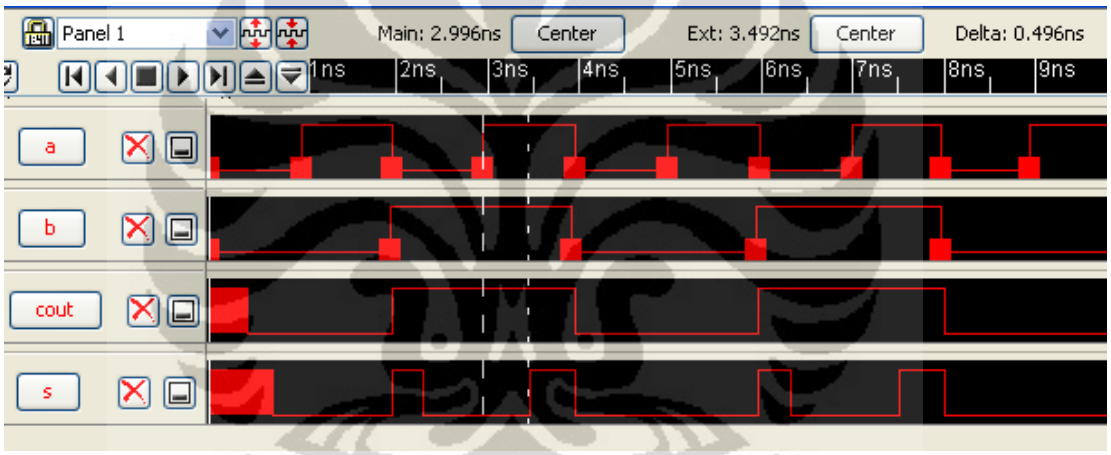
Sedangkan tegangan yang didapat untuk s pada full adder 1 bit yaitu sebesar 5V dan 0V. Terlihat pada Gambar 3.16 perubahan tegangan 5V menjadi 0V atau sebaliknya pada s, disebakan perubahan tegangan pada input a dari 5V menjadi 0V atau sebaliknya.



Gambar 3.16 Tegangan pada output s dari Full Adder

Baik Gambar 3.15 dan 3.16 memperlihatkan bahwa karakterisasi arus dan tegangan yang dihasilkan pada layout CMOS Full Adder sesuai dengan tabel kebenaran rangkaian Full Adder. Bahwa dengan tegangan pada input a sebesar 0V, input b sebesar 5V dan input c_{in} sebesar 5V maka dihasilkan tegangan pada output c_{out} sebesar 5V dan output s sebesar 0V. Dan dengan tegangan pada input a sebesar 5V, input b sebesar 5V dan input c_{in} sebesar 5V maka dihasilkan tegangan pada output c_{out} sebesar 5V dan output s sebesar 5V.

Sedangkan simulasi yang dihasilkan untuk melihat estimasi performa *delay* dari Full Adder 1 bit terlihat pada Gambar 3.17. Pada Gambar tersebut terlihat *delay* yang dihasilkan pada saat input a bernilai logika 1 dan input b bernilai logika 1 terhadap output s bernilai logika 1 sebesar 0,496ns.



Gambar 3.17 *Delay* pada layout CMOS Full Adder

3.4.2 Karakterisasi dan Estimasi Performa Layout CMOS Full Subtractor

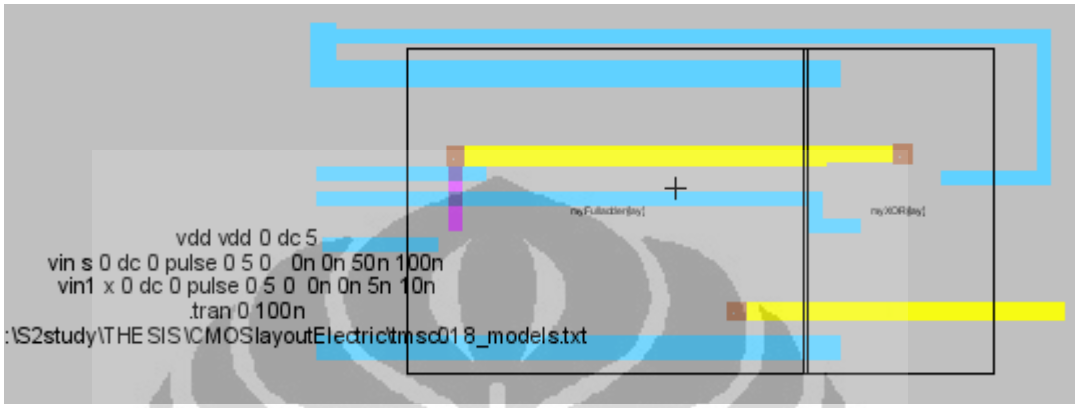
Pada simulasi karakterisasi arus dan tegangan dari Full Subtractor 1 bit, terdapat beberapa setting parameter awal yang ditentukan yaitu

a. *Power supply* dalam hal ini vdd diberikan tegangan DC 5V
b. Input y dihubungkan pada vdd, hal ini menyebabkan tegangan pada input y stabil dengan nilai 5V.
c. Input x merupakan tegangan pulsa 0 dan 5V dimana lebar tegangan masing – masing 5ns.
d. Input s merupakan tegangan pulsa 0 dan 5V dimana lebar tegangan masing – masing 50ns.

e. Parameter model yang digunakan yaitu tmsc018_models.txt yang merupakan parameter TMSC018.

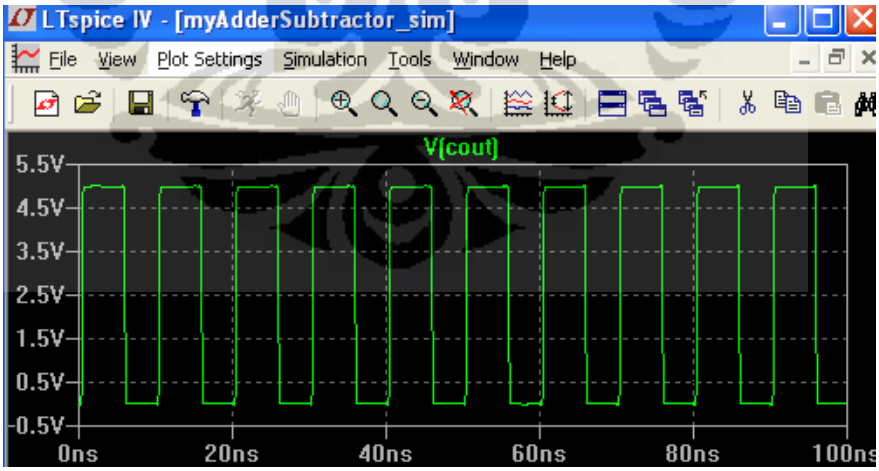
f. Simulasi analisis *transient* berjalan pada 0 – 100ns.

Gambar 3.18 memperlihatkan setting parameter awal layout CMOS Full Subtractor 1 bit untuk melakukan simulasi karakterisasi arus dan tegangan



Gambar 3.18 Setting parameter simulasi karakterisasi I – V dari Full Subtractor

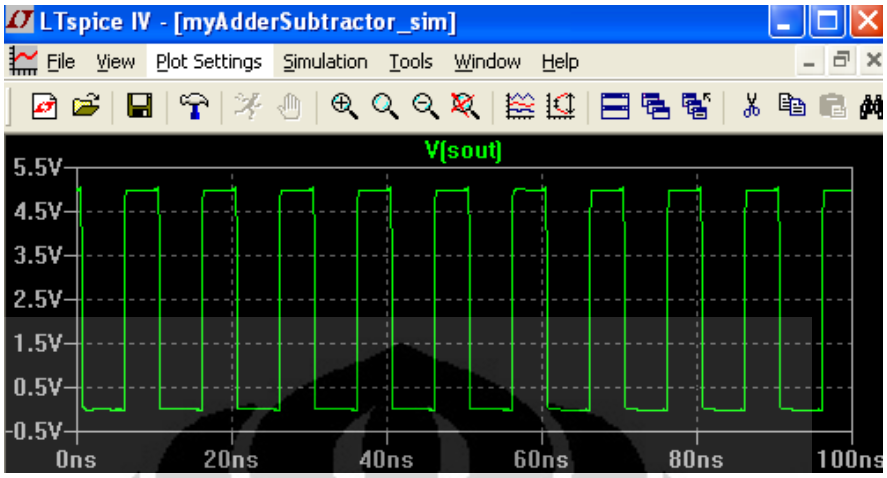
Dari hasil simulasi karakterisasi arus dan tegangan Full Subtractor 1 bit menggunakan LTSpice didapatkan tegangan untuk c_{out} sebesar sebesar 5V dan 0V. Terlihat pada Gambar 3.19 perubahan tegangan 5V menjadi 0V atau sebaliknya pada c_{out}, disebakan perubahan tegangan pada input x dari 5V menjadi 0V atau sebaliknya.



Gambar 3.19 Tegangan pada output c_{out} dari Full Subtractor

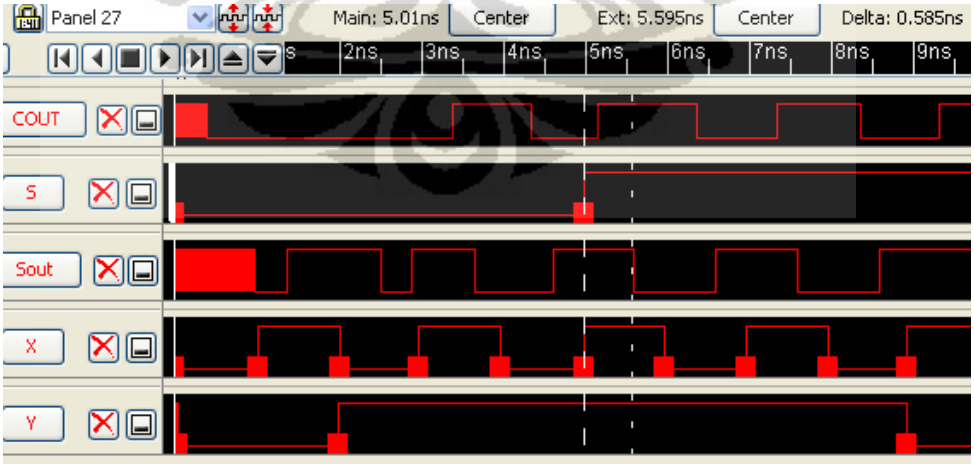
Sedangkan tegangan yang didapat untuk s_{out} pada full adder 1 bit yaitu sebesar 5V dan 0V. Terlihat pada Gambar 3.20 perubahan tegangan 0V menjadi 5V atau

sebaliknya pada s_{out}, disebakan perubahan tegangan pada input x dari 5V menjadi 0V atau sebaliknya.



Gambar 3.20 Tegangan pada output s_{out} dari Full Subtractor

Baik Gambar 3.19 dan 3.20 memperlihatkan bahwa karakterisasi arus dan tegangan yang dihasilkan pada layout CMOS Full Subtractor sesuai dengan tabel kebenaran rangkaian Full Subtractor pada Gambar 3.12. Bahwa dengan tegangan pada input x sebesar 0V, input y sebesar 5V dan input s sebesar 0V maka dihasilkan tegangan pada output c_{out} sebesar 0V dan output s_{out} sebesar 5V. Dan dengan tegangan pada input x sebesar 5V, input y sebesar 5V dan input s sebesar 0V maka dihasilkan tegangan pada output c_{out} sebesar 5V dan output s_{out} sebesar 0V.



Gambar 3.21 *Delay* pada layout CMOS Full Subtractor

Sedangkan simulasi yang dihasilkan untuk melihat estimasi performa *delay* dari Full Subtractor 1 bit terlihat pada Gambar 3.21. Pada Gambar tersebut terlihat

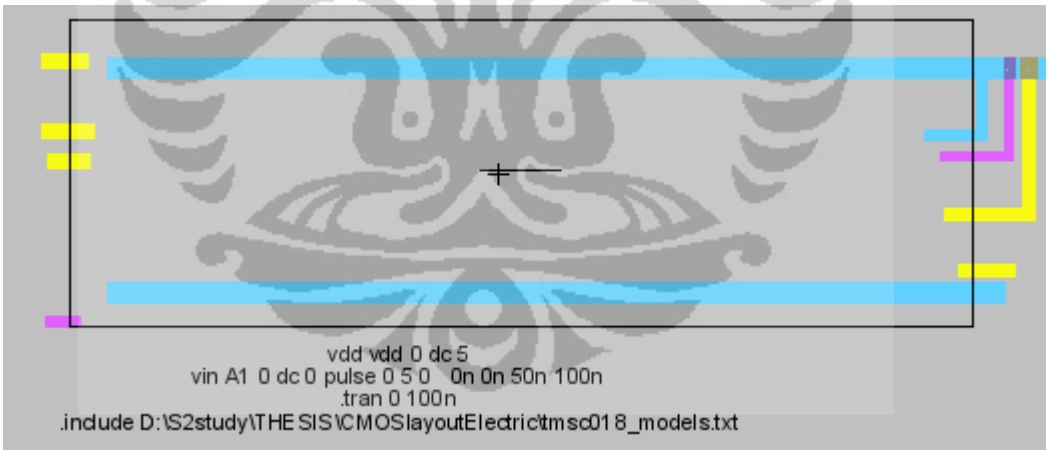
delay yang dihasilkan pada saat input x bernilai logika 1 dan input y bernilai logika 1 terhadap output s_{out} bernilai logika 0 sebesar 0,585ns.

3.4.3 Karakterisasi dan Estimasi Performa Layout CMOS Full Multiplicator

Pada simulasi karakterisasi arus dan tegangan dari Full Multiplicator 2 bit, terdapat beberapa setting parameter awal yang ditentukan yaitu

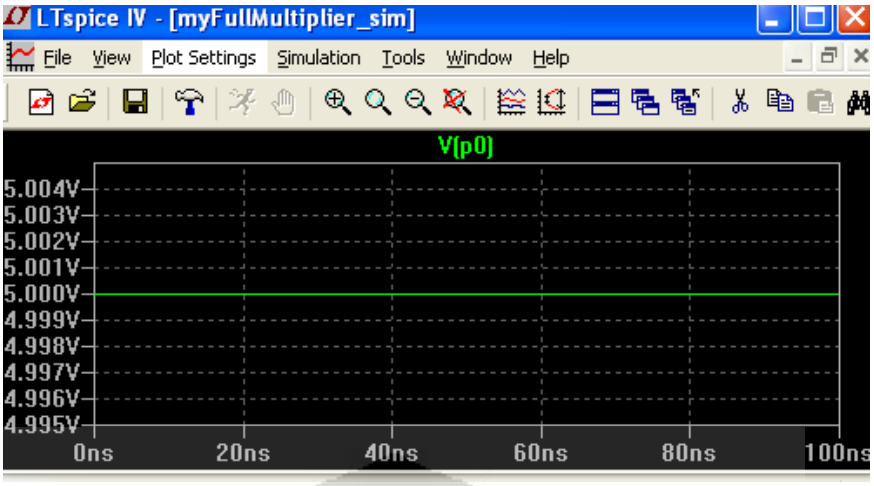
a. *Power supply* dalam hal ini vdd diberikan tegangan DC 5V
b. Input A_0, B_0 dan B_1 dihubungkan pada vdd, hal ini menyebabkan tegangan pada input A_0, B_0 dan B_1 stabil dengan nilai 5V.
c. Input A_1 merupakan tegangan pulsa 0 dan 5V dimana lebar tegangan masing – masing 50ns.
d. Parameter model yang digunakan yaitu tmsc018_models.txt yang merupakan parameter TMSC018.
e. Simulasi analisis *transient* berjalan pada 0 – 100ns.

Gambar 3.22 memperlihatkan setting parameter awal layout CMOS Full Multiplicator 2 bit untuk melakukan simulasi karakterisasi arus dan tegangan

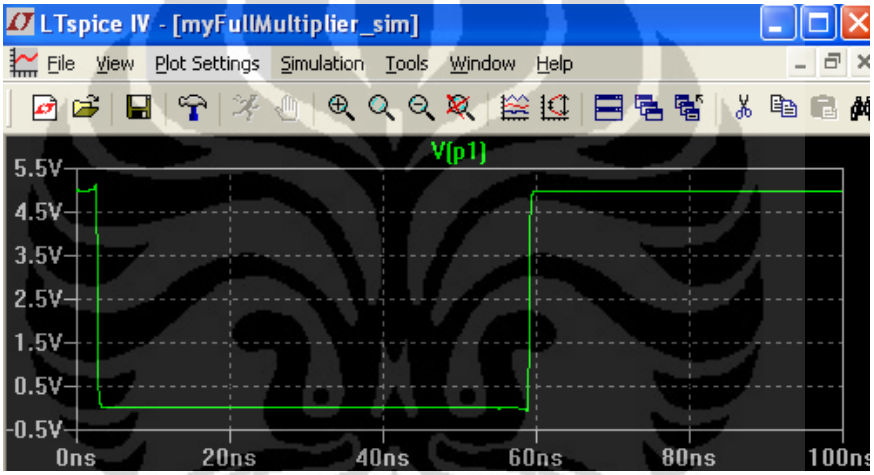


Gambar 3.22 Setting parameter simulasi karakterisasi I – V dari Full Multiplicator

Dari hasil simulasi karakterisasi arus dan tegangan Full Multiplicator 2 bit menggunakan LTSpice didapatkan tegangan untuk output p_0 sebesar 5V sesuai dengan Gambar 3.23. Dan untuk output p1 tegangan yang didapatkan sebesar 0V dan 5V sesuai gambar 3.24.



Gambar 3.23 Tegangan pada output p_0 dari Full Multiplicator

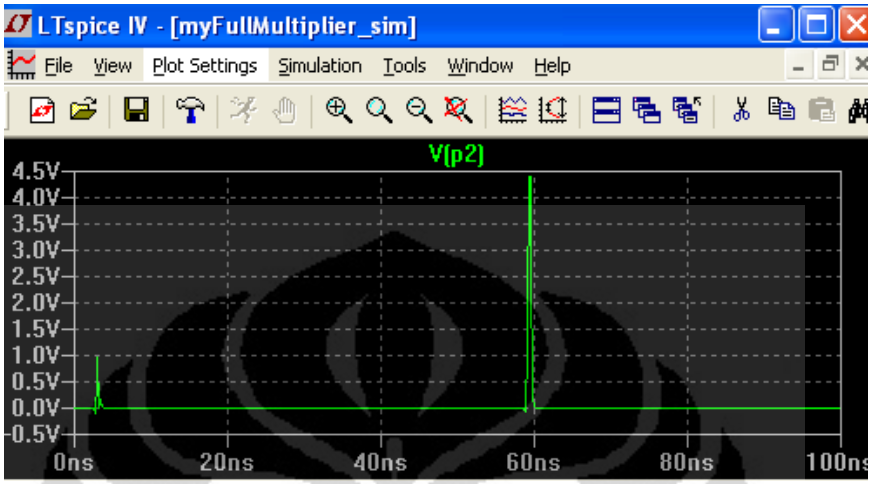


Gambar 3.24 Tegangan pada output p_1 dari Full Multiplicator

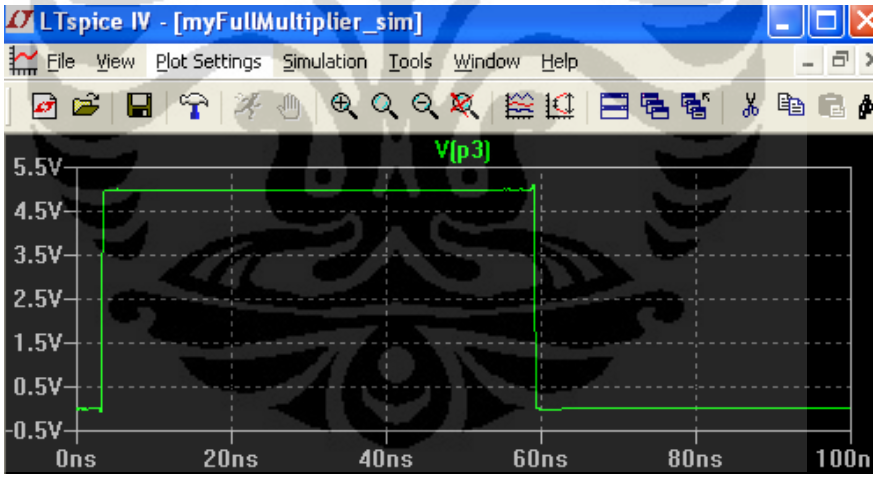
Sedangkan tegangan yang didapatkan untuk output p_2 sebesar 0V sesuai dengan Gambar 3.25. Adapun tegangan sesaat yang didapatkan pada output p_2 sebesar 4,4V dikarenakan perubahan tegangan input A_1 dari 5V menjadi 0V. Dan untuk output p3 tegangan yang didapatkan sebesar 5V dan 0V sesuai gambar 3.26.

Baik Gambar 3.23, 3.24, 3.25 dan 3.26 memperlihatkan bahwa karakterisasi arus dan tegangan yang dihasilkan pada layout CMOS Full Multiplicator sesuai dengan tabel kebenaran rangkaian Full Multiplicator. Bahwa dengan tegangan pada input A_1 sebesar 5V, input A_0 sebesar 5V, input B_0 sebesar 5V dan input B_1 sebesar 5V maka dihasilkan tegangan pada output p_0 sebesar 5V, output p_1 sebesar 0V, output

p_2 sebesar 0V dan output p_3 sebesar 5V. Kondisi tersebut memperlihatkan operasi perkalian untuk input A dengan nilai integer 3 dan input B dengan nilai integer 3 yang menghasilkan output p dengan nilai integer 9.



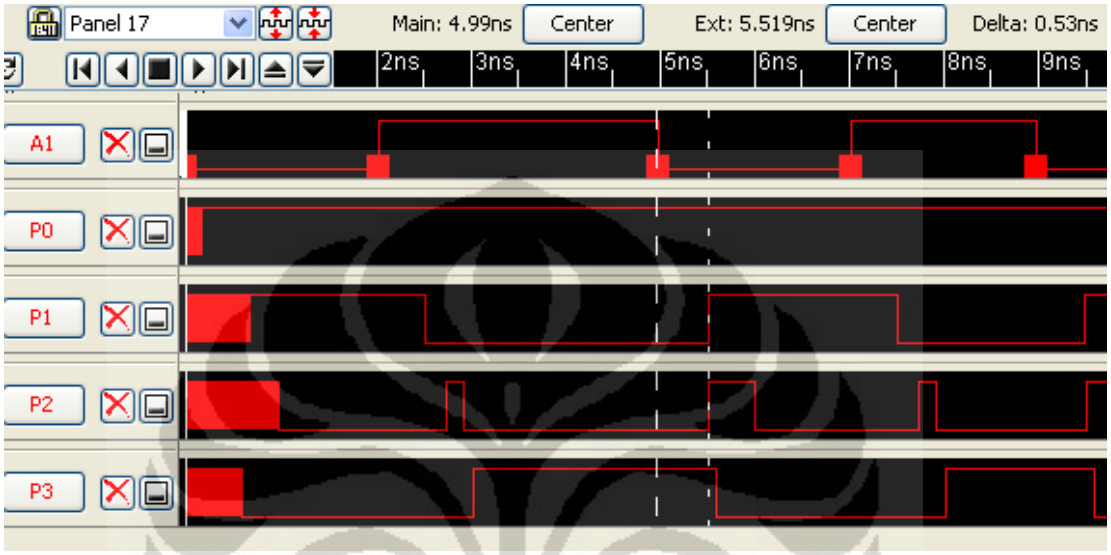
Gambar 3.25 Tegangan pada output p_2 dari Full Multiplicator



Gambar 3.26 Tegangan pada output p_3 dari Full Multiplicator

Dan dengan dengan tegangan pada input A_1 sebesar 0V, input A_0 sebesar 5V, input B_0 sebesar 5V dan input B_1 sebesar 5V maka dihasilkan tegangan pada output p_0 sebesar 5V, output p_1 sebesar 5V, output p_2 sebesar 0V dan output p_3 sebesar 0V. Kondisi tersebut memperlihatkan operasi perkalian untuk input A dengan nilai integer 1 dan input B dengan nilai integer 3 yang menghasilkan output p dengan nilai integer 3.

Sedangkan simulasi yang dihasilkan untuk melihat estimasi performa *delay* dari Full Multiplicator 2 bit terlihat pada Gambar 3.27. Pada Gambar tersebut terlihat *delay* yang dihasilkan pada saat input A_1 bernilai logika 0, A_0 bernilai logika 1, B_1 bernilai logika 1 dan input B_0 bernilai logika 1 terhadap output p_0 bernilai logika 1, p_1 bernilai logika 1, p_2 bernilai logika 0 dan p_3 bernilai logika 0 sebesar 0,53ns.



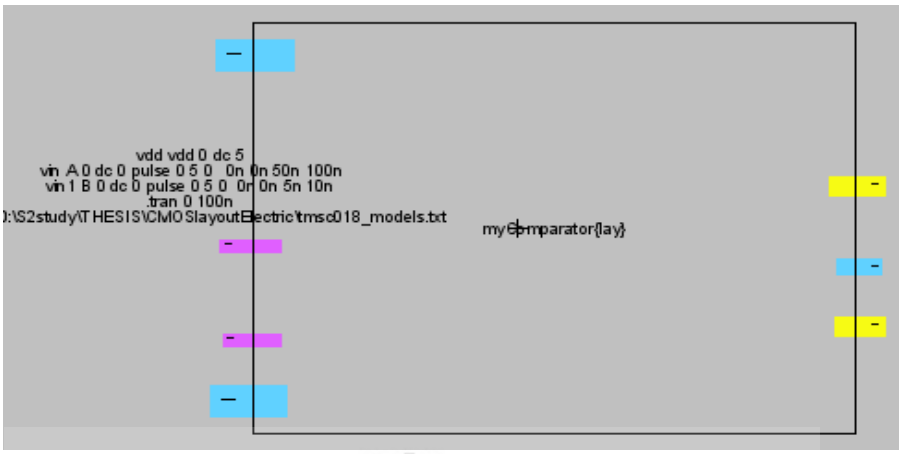
Gambar 3.27 *Delay* pada layout CMOS Full Multiplicator

3.4.4 Karakterisasi dan Estimasi Performa Layout CMOS Comparator

Pada simulasi karakterisasi arus dan tegangan dari Comparator 1 bit, terdapat beberapa setting parameter awal yang ditentukan yaitu

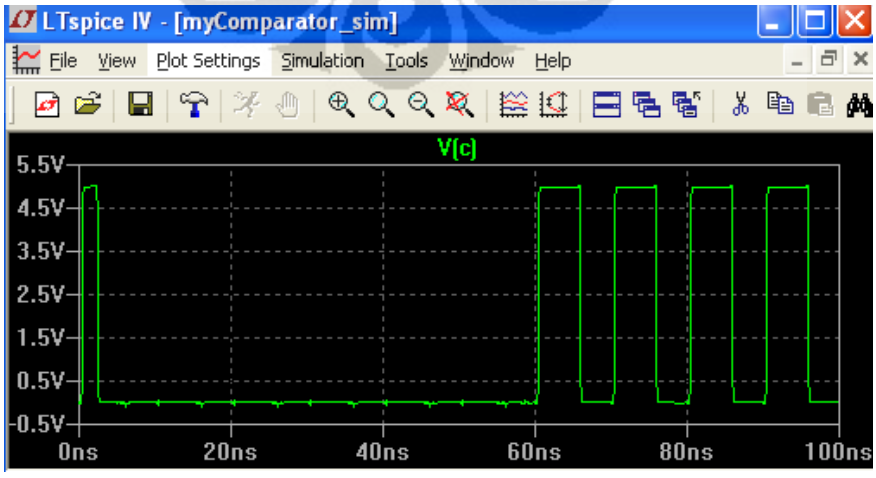
a. *Power supply* dalam hal ini vdd diberikan tegangan DC 5V
b. Input A merupakan tegangan pulsa 0 dan 5V dimana lebar tegangan masing – masing 50ns.
c. Input B merupakan tegangan pulsa 0 dan 5V dimana lebar tegangan masing – masing 5ns.
d. Parameter model yang digunakan yaitu tmsc018_models.txt yang merupakan parameter TMSC018.
e. Simulasi analisis *transient* berjalan pada 0 – 100ns.

Gambar 3.28 memperlihatkan setting parameter awal layout CMOS Comparator 1 bit untuk melakukan simulasi karakterisasi arus dan tegangan.

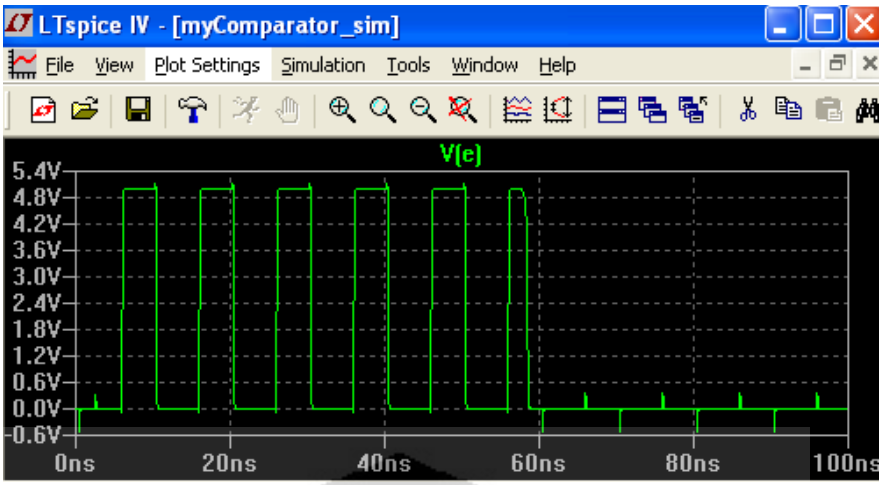


Gambar 3.28 Setting parameter simulasi karakterisasi I – V dari Comparator

Dari hasil simulasi karakterisasi arus dan tegangan Comparator 1 bit menggunakan LTSpice didapatkan tegangan untuk output C sebesar 5V dan 0Vsesuai dengan Gambar 3.29. Dan untuk output D tegangan yang didapatkan sebesar 0V dan 5V sesuai gambar 3.30. Baik Gambar 3.29 dan 3.30 memperlihatkan bahwa karakterisasi arus dan tegangan yang dihasilkan pada layout CMOS Comparator sesuai dengan tabel kebenaran rangkaian Comparator pada Gambar 3.19. Bahwa dengan tegangan pada input A sebesar 5V dan input B sebesar 5V maka dihasilkan tegangan pada output C sebesar 0V dan output D sebesar 0V. Kondisi tersebut memperlihatkan perbandingan bit pada input A dan input B dengan kondisi A = B sehingga nilai output C bernilai logika 0 dan nilai output D bernilai logika 0.



Gambar 3.29 Tegangan pada output C dari Comparator



Gambar 3.30 Tegangan pada output D dari Comparator

Untuk tegangan pada input A sebesar 0V dan input B sebesar 5V maka dihasilkan tegangan pada output C sebesar 5V dan output D sebesar 0V. Kondisi tersebut memperlihatkan perbandingan bit pada input A dan input B dengan kondisi A < B sehingga nilai output C bernilai logika 1 dan nilai output D bernilai logika 0. Sedangkan tegangan pada input A sebesar 5V dan input B sebesar 0V maka dihasilkan tegangan pada output C sebesar 0V dan output D sebesar 5V. Kondisi tersebut memperlihatkan perbandingan bit pada input A dan input B dengan kondisi A > B sehingga nilai output C bernilai logika 0 dan nilai output D bernilai logika 1.

Gambar 3.31 *Delay* pada layout CMOS Comparator

Simulasi yang dihasilkan untuk melihat estimasi performa *delay* dari Comparator 1 bit terlihat pada Gambar 3.31. Pada Gambar tersebut terlihat *delay* yang dihasilkan pada saat input A bernilai logika 0 dan input B bernilai logika 1 terhadap output C bernilai logika 1 sebesar 0,385ns.

3.5 Pad Frame IC DTW

Pada sub bab ini akan dijelaskan layout CMOS datapath dan *pad frame* dari IC DTW. Datapath yang dimaksud merupakan komponen blok fungsi yang dijelaskan pada sub bab 3.2 dan tersusun dari standar cell yang dijelaskan pada sub bab 3.3 dan standar cell dari gerbang logika dasar.

Datapath yang dibahas yaitu datapath DTW dan CTW level 1 sesuai gambar 3.6 dan datapath DTW dan CTW level 2 sesuai gambar 3.7 dan 3.8. Datapath – datapath tersebut diintegrasikan menjadi full datapath dan dihubungkan pada pin input dan output dalam bentuk *pad frame.*

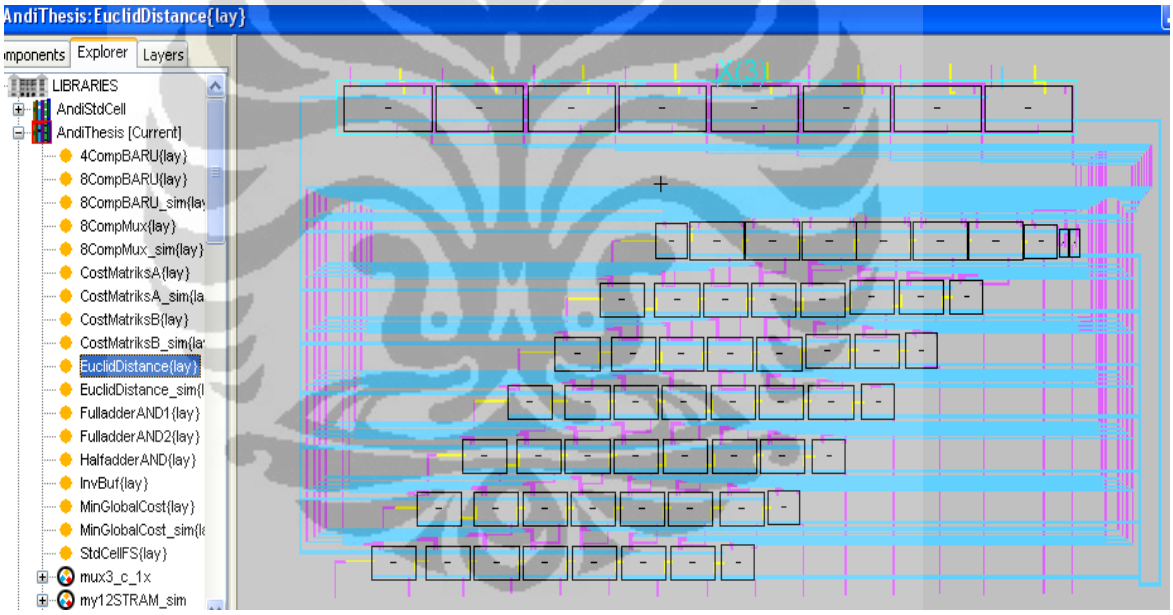
3.5.1 CMOS Layout Datapath DTW dan CTW tahap 1

Datapath DTW dan CTW tahap 1 merupakan datapath yang berfungsi untuk menghitung nilai *euclid distance* dari dua buah nilai. Datapath ini terbentuk dari beberapa standar cell yang dapat dilihat pada tabel 3.2 yaitu *standar cell* gerbang dasar XOR 2 input, gerbang dasar inverter, gerbang dasar AND 2 input, rangkaian half adder, rangkaian full adder, rangkaian full subtractor dan rangkaian full multiplier. Total jumlah transistor PMOS dan NMOS yang dibutuhkan untuk membuat datapath DTW dan CTW tahap 1 adalah 4376 transistor.

Adapun layout CMOS dari datapath DTW dan CTW tahap 1 dapat dilihat pada gambar 3.32. Pada gambar tersebut layout CMOS dari standar cell dihubungkan menggunakan *wire* metal 1, *wire* metal 2 dan wire metal 3 untuk membentuk layout CMOS dari datapath tersebut. Untuk detail layout CMOS 2D dan 3D dari datapath tersebut dapat dilihat pada Gambar 1 dan 2 pada lampiran 6.

Tabel 3.2 Standar cell pembentuk datapath DTW dan CTW tahap 1

No.	Standar cell	Jumlah	Trasistor/Standar cell	Jumlah transistor
1	myFulladder	96	28	2688
2	myHalfAdder	16	18	288
3	myFullMultiplier	8	60	480
4	myFullSubtractor	8	52	416
5	myXOR	24	12	288
6	myAND	100	2	200
7	myInv	8	2	16
8	Total Transistor yang digunakan :			**4376**



Gambar 3.32 Layout CMOS Datapath DTW dan CTW tahap 1

Datapath DTW dan CTW tahap 1 pada IC fast DTW digunakan untuk menghitung nilai *euclid distance* yang digunakan sebagai nilai cost matriks pada i = 0 dan j = 0. Nilai *euclid distance* dari dua buah data didapatkan dengan mengurangi data yang satu dengan yang lainnya dan hasilnya dilakukan operasi perpangkatan dua. Untuk membuktikan bahwa layout CMOS datapath shrunk yang dirancang memenuhi

kondisi diatas, dilakukan simulasi IRSIM untuk melihat output dan delay waktu prosesnya. Diasumsikan nilai vektor X adalah 6 dan nilai vektor Y adalah 5.

Maka hasil simulasi IRSIM untuk datapath DTW dan CTW tahap 1 adalah $(6-5)^2 = 1$yang disimpan pada vektor output P. Dari simulasi ini dapat dikatakan bahwa layout CMOS datapath DTW dan CTW tahap 1 berhasil dirancang dan sesuai kondisi atau fungsi yang diinginkan. Output tersebut dapat dilihat pada gambar 3.33.

Gambar 3.33 Nilai output vektor P pada simulasi IRSIM

Untuk *delay* waktu proses dari datapath DTW dan CTW tahap 1 selama 4,094 ns yang dihitung dari selisih waktu perubahan input dari bit *low* ke bit *high* yaitu 3,014 ns terhadap perubahan output sesuai input pada waktu 7,108 ns.

3.5.2 CMOS Layout Datapath DTW dan CTW tahap 2

Datapath DTW dan CTW tahap 2 merupakan datapath yang berfungsi untuk menghitung nilai *euclid distance* dari dua buah nilai dan hasilnya dijumlahkan dengan nilai *minimal global cost*. Datapath ini terbentuk dari beberapa standar cell yang dapat dilihat pada tabel 3.3 yaitu *standar cell* gerbang dasar XOR 2 input, gerbang dasar inverter, gerbang dasar AND 2 input, gerbang dasar AND 4 input, gerbang dasar NOR 2 input, gerbang dasar XNOR 2 input, gerbang dasar OR 4 input, rangkaian multiplexer, rangkaian half adder, rangkaian full adder, rangkaian full subtractor dan rangkaian full multiplier. Total jumlah transistor PMOS dan NMOS yang dibutuhkan untuk membuat datapath DTW dan CTW tahap 2 adalah 10812 transistor.

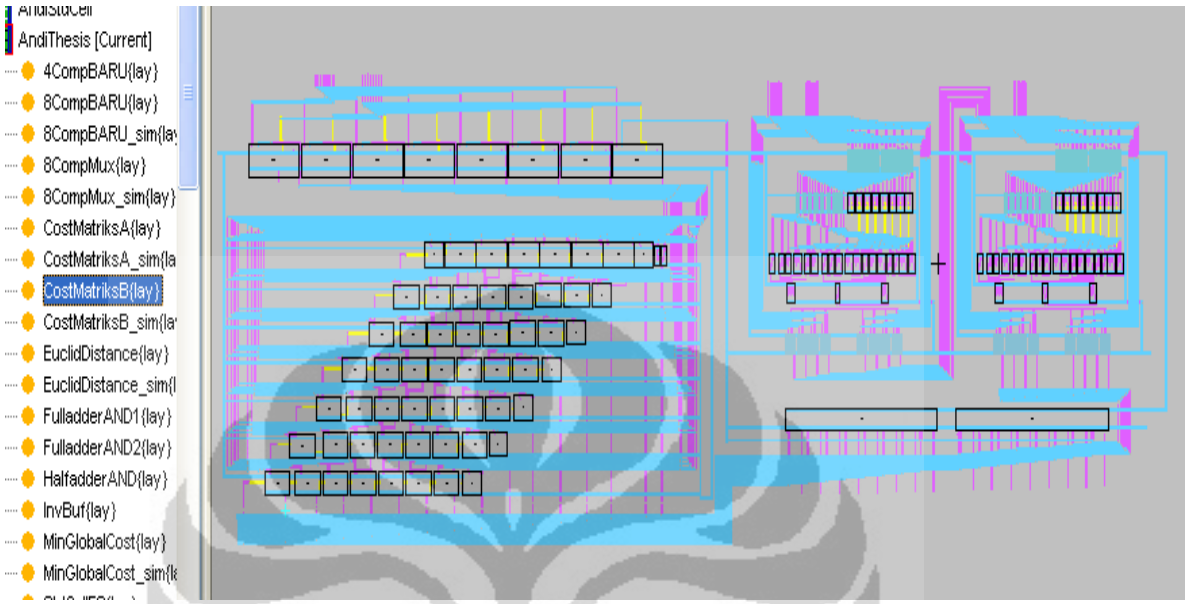
Tabel 3.3 Standar cell pembentuk datapath DTW dan CTW tahap 2

No.	Standar cell	Jumlah	Trasistor/Standar cell	Jumlah transistor
1	myFulladder	128	28	3584
2	myHalfAdder	16	18	288
3	myFullMultiplier	8	60	480
4	myFullSubtractor	16	52	832
5	myXOR	8	12	96
6	myAND	278	2	556
7	myInv	384	2	768
8	myAND4_1x	144	10	1440
9	myNOR	12	4	48
10	myOR4_1x	32	10	320
11	myXNOR	136	12	1632
12	myMUX2_1x	64	12	768
13	Total Transistor yang digunakan :			**10812**

Adapun layout CMOS dari datapath DTW dan CTW tahap 2 dapat dilihat pada gambar 3.34. Pada gambar tersebut layout CMOS dari standar cell dihubungkan menggunakan *wire* metal 1, *wire* metal 2 dan wire metal 3 untuk membentuk layout CMOS dari datapath tersebut. Untuk detail layout CMOS 2D dan 3D dari datapath DTW dan CTW tahap 2 dapat dilihat pada Gambar 1 dan 2 pada lampiran 9. Sedangkan layout CMOS *minimal global cost* dapat dilihat pada Gambar 1 dan 2 lampiran 7.

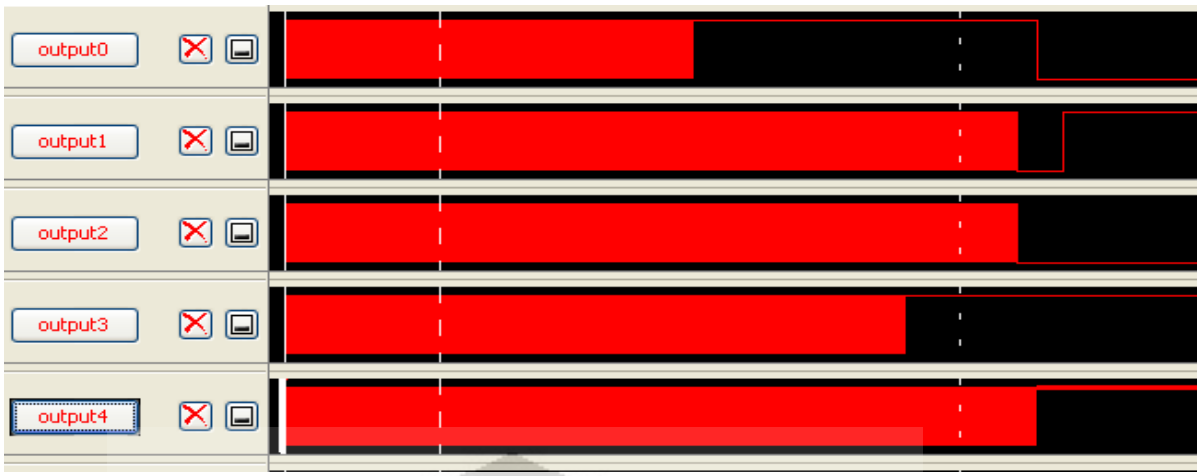
Datapath DTW dan CTW tahap 2 pada IC fast DTW digunakan untuk menghitung nilai *euclid distance* dan dijumlahkan dengan nilai *minimal global* cost yang digunakan sebagai nilai cost matriks pada i > 0 dan j > 0. Nilai *euclid distance* dari dua buah data didapatkan dengan mengurangi data yang satu dengan yang lainnya dan hasilnya dilakukan operasi perpangkatan dua. Nilai *minimal global cost*

didapatkan dari nilai terkecil perbandingan tiga buah nilai cost matriks atau $\min[\cos tmatriks(i-1, j-1), \cos tmatriks(i-1, j), \cos tmatriks(i, j-1)]$.



Gambar 3.34 Layout CMOS Datapath DTW dan CTW tahap 2

Untuk membuktikan bahwa layout CMOS datapath s DTW dan CTW tahap 2 yang dirancang memenuhi kondisi diatas, dilakukan simulasi IRSIM untuk melihat output dan delay waktu prosesnya. Diasumsikan nilai vektor X adalah 7, nilai vektor Y adalah 2, cost matriks(0,0) adalah 1, cost matriks(0,1) adalah 17 dan cost matriks(1,0) adalah 5. Maka hasil simulasi IRSIM untuk datapath DTW dan CTW tahap 2 adalah $(7-2)^2 + \min(1,17,5) = 25+1 = 26$ yang disimpan pada vektor output atau cost matriks (i, j). Dari simulasi ini dapat dikatakan bahwa layout CMOS datapath DTW dan CTW tahap 2 berhasil dirancang dan sesuai kondisi atau fungsi yang diinginkan. Output tersebut dapat dilihat pada gambar 3.35.



Gambar 3.35 Nilai vektor output pada simulasi IRSIM

Untuk *delay* waktu proses dari datapath DTW dan CTW tahap 2 selama 6,712 ns yang dihitung dari selisih waktu perubahan input dari bit *low* ke bit *high* yaitu 1,172 ns terhadap perubahan output sesuai input pada waktu 7,884 ns.

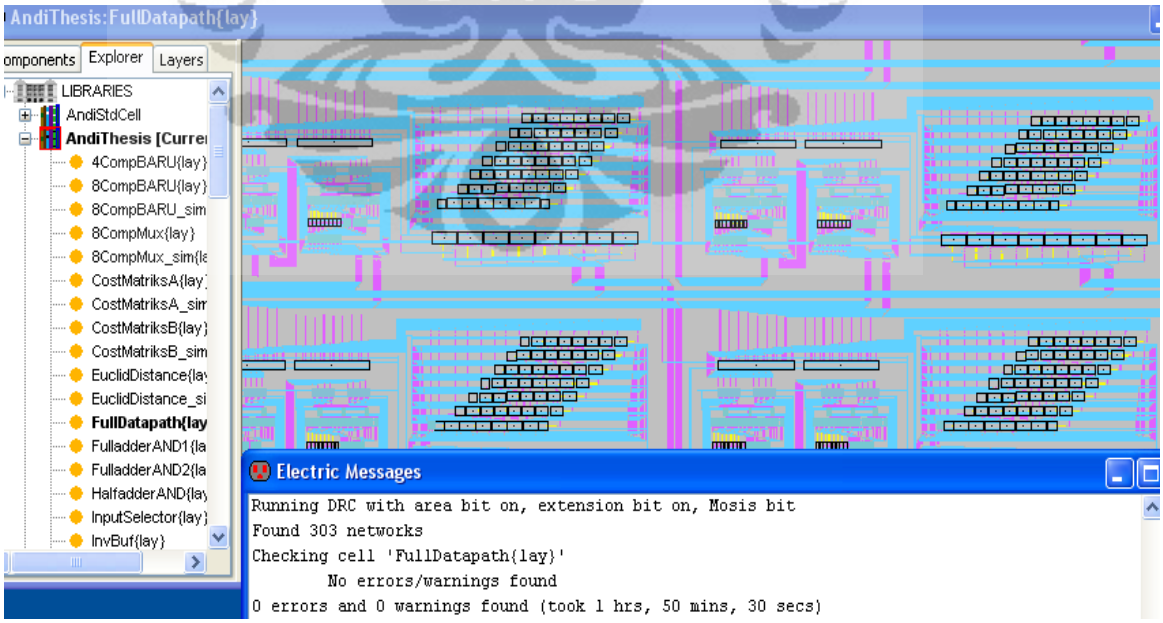
3.5.3 CMOS Layout Full Datapath

Datapath full datapath merupakan datapath hasil integrasi dari seluruh datapath atau datapath yang menghitung nilai minimal cost dari algoritma fast DTW. Datapath ini terbentuk dari beberapa standar cell yang dapat dilihat pada tabel 3.4 yaitu *standar cell* gerbang dasar XOR 2 input, gerbang dasar inverter, gerbang dasar AND 2 input, gerbang dasar AND 4 input, gerbang dasar NOR 2 input, gerbang dasar XNOR 2 input, gerbang dasar OR 4 input, gerbang dasar OR 3 input, rangkaian multiplexer, rangkaian half adder, rangkaian full adder, rangkaian full subtractor dan rangkaian full multiplier. Total jumlah transistor PMOS dan NMOS yang dibutuhkan untuk membuat datapath DTW dan CTW tahap 2 adalah 100286 transistor.

Adapun layout CMOS dari full datapath dapat dilihat pada gambar 3.36. Pada gambar tersebut layout CMOS dari standar cell dihubungkan menggunakan *wire* metal 1, *wire* metal 2 dan wire metal 3 untuk membentuk layout CMOS dari full datapath tersebut. Untuk detail layout CMOS 2D dan 3D dari datapath tersebut dapat dilihat pada Gambar 1 dan 2 pada lampiran 10.

Tabel 3.4 Standar cell pembentuk full datapath

No.	Standar cell	Jumlah	Trasistor/Standar cell	Jumlah transistor
1	myFulladder	1440	28	40320
2	myHalfAdder	192	18	3456
3	myFullMultiplier	96	60	5760
4	myFullSubtractor	168	52	8736
5	myXOR	176	12	2112
6	myAND	2446	2	4892
7	myInv	2728	2	5456
8	myAND4_1x	1008	10	10080
9	myNOR	84	4	336
10	myOR4_1x	224	10	2240
11	myXNOR	952	12	11424
12	myMUX2_1x	448	12	5376
13	myOR3_1x	16	8	98
14	Total Transistor yang digunakan :			**100286**



Gambar 3.36 Layout CMOS Full Datapath

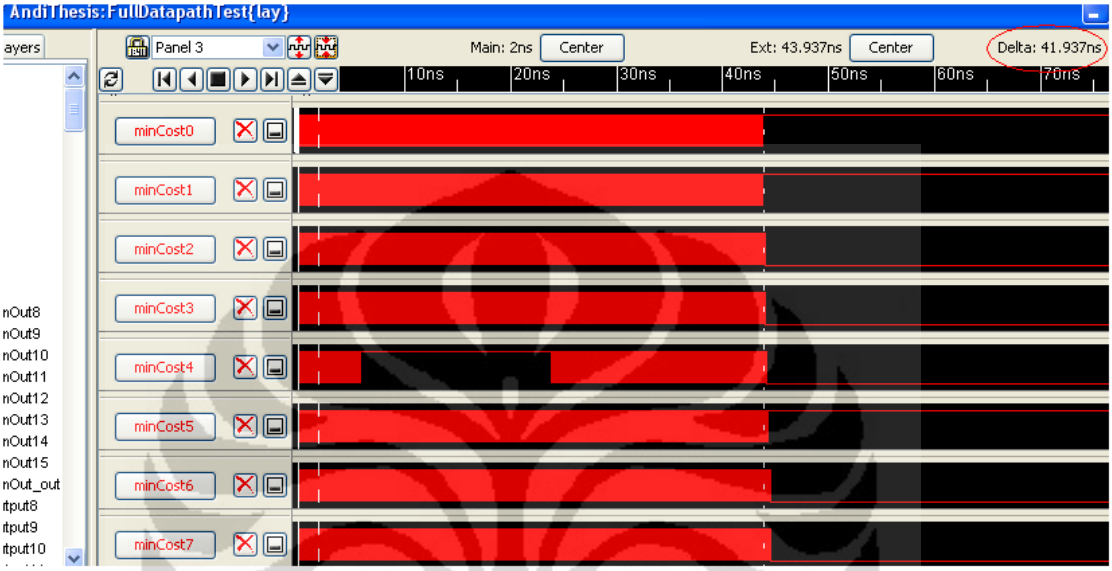
Full datapath pada IC fast DTW merupakan integrasi dari datapath input selector, shrunk, DTW dan CTW tahap 1 dan tahap 2 sesuai dengan gambar 2.6 untuk algoritma fast DTW [13]. Untuk membuktikan bahwa layout CMOS full datapath yang dirancang memenuhi kondisi diatas, dilakukan simulasi IRSIM untuk melihat output dan delay waktu prosesnya. Diasumsikan bahwa banyaknya vektor input adalah 4 dimana input vektor tsI adalah [6, 7, 7, 8] dimana tsiA = 6, tsiB = 7, tsiC = 7 dan tsiD = 8. Dan untuk input vektor tsJ adalah [5, 2, 6, 4] dimana tsiA = 6, tsiB = 7, tsiC = 7 dan tsiD = 8. Maka hasil simulasi IRSIM untuk full datapath dapat dilihat pada tabel 3.5. yang disesuaikan dengan nilai *window* [(0,0), (0,1), (1,0), (1,1), (1,2), (2,1), (2,2), (2,3), (3,2), (3,3)].

Tabel 3.5 Hasil perhitungan full datapath

No	Index i	Index j	Cost matriks (i,j)
1	0	0	$(6-5)^2 = 1$
2	0	1	$(6-2)^2 + \cos tmatriks(0,0) = 16+1 = 17$
3	1	0	$(7-5)^2 + \cos tmatriks(0,0) = 4+1 = 5$
4	1	1	$(7-2)^2 + \min(17,1,5) = 25+1 = 26$
5	1	2	$(7-6)^2 + \min(17,26,\inf) = 1+17 = 18$
6	2	1	$(7-2)^2 + \min(5,26,\inf) = 25+5 = 30$
7	2	2	$(7-6)^2 + \min(18,26,30) = 1+18 = 19$
8	2	3	$(7-4)^2 + \min(18,19,\inf) = 9+18 = 27$
9	3	2	$(8-6)^2 + \min(30,19,\inf) = 4+19 = 23$
10	3	3	$(8-4)^2 + \min(27,19,23) = 16+19 = 35$

. Pada tabel 3.5 nilai cost matriks (3,3) adalah nilai minimum cost atau output dari full datapath. Dari simulasi ini dapat dikatakan bahwa layout CMOS full

datapath berhasil dirancang dan sesuai kondisi atau fungsi dari IC fast DTW untuk banyaknya vektor input yaitu 4. Output tersebut dapat dilihat pada gambar 3.37. dan 3.38. Detail mengenai pengecekan hasil dari proses full datapath dapat dilihat pada lampiran 15 mengenai test vektor tersebut.



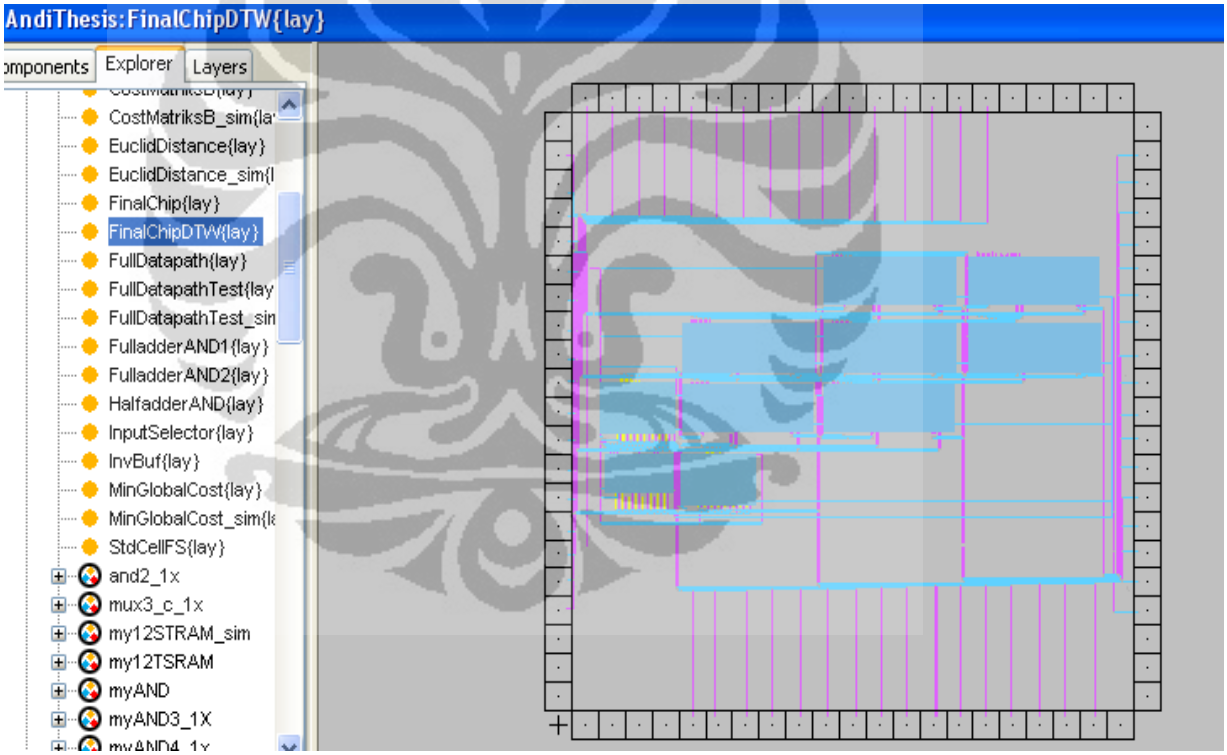
Gambar 3.37 Nilai output minimum cost [0-7] pada simulasi IRSIM

Gambar 3.38 Nilai output minimum cost [8-15] pada simulasi IRSIM

Untuk *delay* waktu proses dari full datapath atau IC fast DTW selama 41,937 ns yang dihitung dari selisih waktu perubahan input dari bit *low* ke bit *high* yaitu 2,007 ns terhadap perubahan output sesuai input pada waktu 43,937 ns.

3.5.4 CMOS Layout Pad Frame IC DTW

Pada sub bab ini full datapath yang telah dirancang diintegrasikan kedalam sebuah pad frame berukuran 84 pin. Hal ini dilakukan sebagai prototype IC DTW dengan banyaknya vektor data input sebanyak 4 dan setiap bit dari vektor data input tersebut mewakili 1 pin. Untuk lebih jelas layout CMOS full datapath dengan pad frame 84 pin dapat dilihat pada gambar 3.39. Untuk detail layout CMOS 2D dan 3D dari pad frame tersebut dapat dilihat pada lampiran 11.



Gambar 3.39 *Prototype* IC fast DTW pada pad frame 84 pin

BAB 4

IMPLEMENTASI IC DTW PADA FPGA SPARTAN IIELC

4.1 Kode Program VHDL IC DTW

Pada sub bab ini akan dilakukan pembahasan mengenai kode program VHDL dari komponen penyusun IC DTW yang menggunakan kode program VHDL dari standar cell yang dibahas pada bab III. Beberapa komponen penyusun tersebut yang akan dibahas kode program VHDL nya adalah full adder 8 bit, full subtractor 8 bit, full multiplicator 8 bit, comparator 8 bit, multiplexer 16 menjadi 8 bit, RAM, euclid distance, cost matriks A, cost matriks B dan integrasi dari seluruh komponen penyusun yaitu DTW itu sendiri.

4.1.1 VHDL Full Adder 8 bit

Kode program VHDL full adder 8 bit menggunakan VHDL full adder 1 bit, yang merupakan *behavior* dari standar cell full adder 1 bit. *Behavior* yang didapat dari layout CMOS standar cell ful adder 1 bit terdeskripsikan sampai pada level transistor CMOS, sedangkan pada VHDL tidak dikenal level transistor melainkan level gerbang logika dasar. Sehingga dibutuhkan penyesuaian agar *behavior* dari layout CMOS standar cell full adder 1 bit dapat dikenali sebagai kode program VHDL full adder 1 bit yang dapat diimplementasikan pada FPGA. Untuk lebih jelas terkait penyesuaian tersebut dapat dilihat pada Tabel 4.1.

Tabel 4.1 *Behavior* dan VHDL full adder 1 bit

Behavior standar cell full adder 1 bit	Kode program VHDL full adder 1 bit

`nmos_0: nMOStran port map(coutb, gnd, cout, open);`
`nmos_1: nMOStran port map(sumb, gnd, s, open);`
`……..`
`……..`
`……..`
`nmos_11: nMOStran port map(c, net_135, coutb, open);`
`nmos_12: nMOStran port map(b, gnd, net_135, open);`
`nmos_13: nMOStran port map(a,
	`begin`
	`s <= x xor y xor cin;`
	`cout <= (x and y)or(cin and x)or(cin and y);`
	`end Behavioral;`

```
 net_135, gnd, open);
   pmos_0: PMOStran port map(coutb,
 vdd, cout, open);
   pmos_1: PMOStran port map(sumb,
 vdd, s, open);
   pmos_2: PMOStran port map(a,
 net_10, vdd, open);
 ……..
 ……..
 ……..
        pmos_12:   PMOStran   port
      map(b, vdd, net_30, open);
  pmos_13:  PMOStran  port  map(a,
net_30, vdd, open);
```

Dari tabel 4.1 *behavior* dari standar cell full adder 1 bit sebenarnya merupakan VHDL yang digenerate dari layout CMOS full adder 1 bit, tetapi hal ini tidak dapat untuk diimplementasikan secara langsung pada FPGA dikarenakan deskripsi komponen primitifnya pada level transistor CMOS. Sehingga konektifitas antar transistor CMOS pada *behavior* tersebut dirubah dalam operasi gerbang logika dasar yang dapat dikenali FPGA. Kode program yang mendeskripsikan operasi gerbang logika dasar tersebut yang merupakan VHDL dari full adder 1 bit.

Untuk membuat full adder 8 bit dibutuhkan 8 buah full adder 1 bit, sehingga VHDL full adder 8 bit terbentuk dari VHDL full adder 1 bit yang dihubungkan setiap port input dan outputnya menggunakan *port map*. Berikut kode program VHDL dari full adder 8 bit :

```
begin

        tahap0 : FullAdder1 port map(cin, x(0),y(0),s(0),c(1));
        tahap1 : FullAdder1 port map(c(1), x(1),y(1),s(1),c(2));
        tahap2 : FullAdder1 port map(c(2), x(2),y(2),s(2),c(3));
        tahap3 : FullAdder1 port map(c(3), x(3),y(3),s(3),c(4));
        tahap4 : FullAdder1 port map(c(4), x(4),y(4),s(4),c(5));
        tahap5 : FullAdder1 port map(c(5), x(5),y(5),s(5),c(6));
        tahap6 : FullAdder1 port map(c(6), x(6),y(6),s(6),c(7));
        tahap7 : FullAdder1 port map(c(7), x(7),y(7),s(7),cout);

end Behavioral;
```

4.1.2 VHDL Full Subtractor 8 bit

Kode program VHDL full subtractor 8 bit menggunakan VHDL full subtractor 1 bit, yang merupakan *behavior* dari standar cell full subtractor 1 bit. *Behavior* yang didapat dari layout CMOS standar cell ful subtractor 1 bit terdeskripsikan sampai pada level transistor CMOS, sedangkan pada VHDL tidak dikenal level transistor melainkan level gerbang logika dasar. Sehingga dibutuhkan penyesuaian agar *behavior* dari layout CMOS standar cell full subtractor 1 bit dapat dikenali sebagai kode program VHDL full adder 1 bit yang dapat diimplementasikan pada FPGA. Untuk lebih jelas terkait penyesuaian tersebut dapat dilihat pada Tabel 4.2.

Tabel 4.2 *Behavior* dan VHDL full subtractor 1 bit

Behavior standar cell full subtractor 1 bit	Kode program VHDL full subtractor 1 bit

`begin`
`and2_1x_0: and2_1x port map(Binp, net_25, net_100, vdd, gnd);`
`and2_1x_1: and2_1x port map(Yinp, net_25, net_95, vdd, gnd);`
`and2_1x_2: and2_1x port map(Binp, Yinp, net_122, vdd, gnd);`
`inv_1x_0: inv_1x port map(Xinp, net_25, vdd, gnd);`
`or3_1x_0: or3_1x port map(net_122, net_95, net_100, Bout, vdd, gnd);`
`pin_1: power port map(vdd);`
`pin_4: ground port map(gnd);`
`xor2_1x_0: xor2_1x port map(Yinp, Xinp, net_72, vdd, gnd);`
`xor2_1x_1: xor2_1x port map(net_72, Binp, Dout, vdd, gnd);`
`end StdCellFS_BODY;`
	`begin`
	`d <= (x xor y) xor bin;`
	`bout <= ((not x) and bin) or ((not x) and y) or (y and bin);`
	`end Behavioral;`

Dari tabel 4.2 *behavior* dari standar cell full subtractor 1 bit sebenarnya merupakan VHDL yang digenerate dari layout CMOS full subtractor 1 bit, tetapi hal ini tidak dapat untuk diimplementasikan secara langsung pada FPGA dikarenakan deskripsi komponen primitifnya pada level transistor CMOS. Tetapi *behavior* tersebut telah memperlihatkan beberapa gerbang logika dasar yang digunakan karena

pembuatan standar cell full subtractor 1 bit menggunakan standar cell dari gerbang logika dasar sehingga penyesuaian yang dilakukuan lebih sederhana.

Untuk membuat full subtractor 8 bit dibutuhkan 8 buah full subtractor 1 bit, sehingga VHDL full subtractor 8 bit terbentuk dari VHDL full subtractor 1 bit yang dihubungkan setiap port input dan outputnya menggunakan *port map*. Berikut kode program VHDL dari full subtractor 8 bit :

```
begin

proses1 : FS1 port map (x(0),y(0),bin,d(0),temp(0));
proses2 : FS1 port map (x(1),y(1),temp(0),d(1),temp(1));
proses3 : FS1 port map (x(2),y(2),temp(1),d(2),temp(2));
proses4 : FS1 port map (x(3),y(3),temp(2),d(3),temp(3));
proses5 : FS1 port map (x(4),y(4),temp(3),d(4),temp(4));
proses6 : FS1 port map (x(5),y(5),temp(4),d(5),temp(5));
proses7 : FS1 port map (x(6),y(6),temp(5),d(6),temp(6));
proses8 : FS1 port map (x(7),y(7),temp(6),d(7),bout);

end Behavioral;
```

4.1.3 VHDL Full Multiplicator 8 bit

Kode program VHDL full multiplicator 8 bit menggunakan VHDL full adder 1 bit dan half adder 1 bit, yang merupakan *behavior* dari standar cell full adder 1 bit dan half adder 1 bit. *Behavior* yang didapat dari layout CMOS standar cell ful adder 1 bit terdeskripsikan sampai pada level transistor CMOS, sedangkan pada VHDL tidak dikenal level transistor melainkan level gerbang logika dasar. Sehingga dibutuhkan penyesuaian agar *behavior* dari layout CMOS standar cell full adder 1 bit dapat dikenali sebagai kode program VHDL full adder 1 bit yang dapat diimplementasikan pada FPGA. Tetapi *behavior* dari layout CMOS standar cell half adder 1 terdeskripsikan pada level gerbang logika dasar sehingga penyesuaiaan yang dilakukan lebih sederhana. Untuk lebih jelas terkait penyesuaian tersebut dapat dilihat pada Tabel 4.3.

Tabel 4.3 *Behavior* dan VHDL half adder dan full adder 1 bit

Behavior standar cell full adder dan half adder 1 bit	Kode program VHDL full adder dan half adder 1 bit
`begin` `  myAND_0: myAND port map(A, B, C, vdd, gnd);`	`begin` `  cout <= x and y;`

```
    myXOR_0: myXOR port map(A, B,
  S, vdd, gnd);
    pin_15: power port map(vdd);
    pin_16: ground port map(gnd);
  end myHalfAdder_BODY;

begin
nmos_0: nMOStran port map(coutb,
gnd, cout, open);
  nmos_1: nMOStran port map(sumb,
gnd, s, open);
……..
……..
……..
  nmos_11: nMOStran port map(c,
net_135, coutb, open);
  nmos_12: nMOStran port map(b,
gnd, net_135, open);
  nmos_13: nMOStran port map(a,
net_135, gnd, open);
  pmos_0: PMOStran port map(coutb,
vdd, cout, open);
  pmos_1: PMOStran port map(sumb,
vdd, s, open);
  pmos_2: PMOStran port map(a,
net_10, vdd, open);
……..
……..
……..
       pmos_12:   PMOStran   port
     map(b, vdd, net_30, open);
 pmos_13:  PMOStran  port  map(a,
net_30, vdd, open);
  end myFullAdder_BODY;
```

```
   s <= x xor y;

end Behavioral;

begin

   s <= x xor y xor cin;
   cout <= (x and y)or(cin and
   x)or(cin and y);

end Behavioral;
```

Dari tabel 4.3 *behavior* dari standar cell full adder 1 bit sebenarnya merupakan VHDL yang digenerate dari layout CMOS full adder 1 bit, tetapi hal ini tidak dapat untuk diimplementasikan secara langsung pada FPGA dikarenakan deskripsi komponen primitifnya pada level transistor CMOS. Sehingga konektifitas antar transistor CMOS pada *behavior* tersebut dirubah dalam operasi gerbang logika dasar yang dapat dikenali FPGA. Kode program yang mendeskripsikan operasi gerbang logika dasar tersebut yang merupakan VHDL dari full adder 1 bit. Sedangkan pada *behavior* dari standar cell half adder 1 bit telah terdeskripsikan pada level gerbang logika dasar sehingga penyesuaiaan yang dilakukan lebih mudah.

Untuk membuat full multiplicator 8 bit dibutuhkan beberapa full adder 1 bit dan half adder 1 bit, sehingga VHDL full Multiplicator 8 bit terbentuk dari VHDL

full adder 1 bit dan half adder 1 bit yang dihubungkan setiap port input dan outputnya menggunakan *port map*. Berikut kode program VHDL dari full multipicator 8 bit :

```
        p(0) <= tempB1(0);
    baris11 : HalfAdder1 port map(tempA1(0),tempB1(1),p(1),reg1(0));
    baris12 : FullAdder1 port map(reg1(0),tempA1(1),tempB1(2),regout1(0),reg1(1));
    baris13 : FullAdder1 port map(reg1(1),tempA1(2),tempB1(3),regout1(1),reg1(2));
    baris14 : FullAdder1 port map(reg1(2),tempA1(3),tempB1(4),regout1(2),reg1(3));
    baris15 : FullAdder1 port map(reg1(3),tempA1(4),tempB1(5),regout1(3),reg1(4));
    baris16 : FullAdder1 port map(reg1(4),tempA1(5),tempB1(6),regout1(4),reg1(5));
    baris17 : FullAdder1 port map(reg1(5),tempA1(6),tempB1(7),regout1(5),reg1(6));
    baris18 : HalfAdder1 port map(tempA1(7),reg1(6),regout1(6),reg1(7));

    baris21 : HalfAdder1 port map(tempA2(0),regout1(0),p(2),reg2(0));
    baris22 : FullAdder1 port map(reg2(0),tempA2(1),regout1(1),regout2(0),reg2(1));
    baris23 : FullAdder1 port map(reg2(1),tempA2(2),regout1(2),regout2(1),reg2(2));
    baris24 : FullAdder1 port map(reg2(2),tempA2(3),regout1(3),regout2(2),reg2(3));
    baris25 : FullAdder1 port map(reg2(3),tempA2(4),regout1(4),regout2(3),reg2(4));
    baris26 : FullAdder1 port map(reg2(4),tempA2(5),regout1(5),regout2(4),reg2(5));
    baris27 : FullAdder1 port map(reg2(5),tempA2(6),regout1(6),regout2(5),reg2(6));
    baris28 : FullAdder1 port map(reg2(6),tempA2(7),reg1(7),regout2(6),reg2(7));
    …….
    …….
    …….
         baris71 : HalfAdder1 port map(tempA7(0),regout6(0),p(7),reg7(0));
    baris72 : FullAdder1 port map(reg7(0),tempA7(1),regout6(1),p(8),reg7(1));
    baris73 : FullAdder1 port map(reg7(1),tempA7(2),regout6(2),p(9),reg7(2));
    baris74 : FullAdder1 port map(reg7(2),tempA7(3),regout6(3),p(10),reg7(3));
    baris75 : FullAdder1 port map(reg7(3),tempA7(4),regout6(4),p(11),reg7(4));
    baris76 : FullAdder1 port map(reg7(4),tempA7(5),regout6(5),p(12),reg7(5));
    baris77 : FullAdder1 port map(reg7(5),tempA7(6),regout6(6),p(13),reg7(6));
         baris78 : FullAdder1 port map(reg7(6),tempA7(7),reg6(7),p(14),p(15));
end Behavioral;
```

4.1.4 VHDL Comparator 8 bit

Kode program VHDL comparator 8 bit menggunakan VHDL comparator 4 bit, yang merupakan *behavior* dari comparator 4 bit. Comparator 4 bit merupakan pengembangan dari standar cell comparator 1 bit. *Behavior* yang didapat dari layout CMOS comparator 4 bit terdeskripsikan sampai pada level gerbang logika dasar sehingga penyesuaiaan kode program VHDL lebih mudah. Untuk lebih jelas terkait penyesuaian tersebut dapat dilihat pada Tabel 4.4.

Tabel 4.4 *Behavior* dan VHDL comparator 4 bit

Behavior standar cell comparator 4 bit	Kode program VHDL comparator 4 bit
begin *myAND4_1_0: myAND4_1x port map(net_15, net_58, net_60, net_62, net_204, vdd, A1);* *myAND4_1_1: myAND4_1x port map(net_299, net_26, net_58,*	*begin* *regA(0) <= (not(not x(0) nand y(0))) or (not(x(0) nand (not y(0))));* *regA(1) <= (not(not x(1) nand y(1))) or (not(x(1) nand (not y(1))));*

```
    net_60, net_349, vdd,
     A1);
   myAND_0:       myAND       port
    map(net_151, net_204, net_344,
    vdd, A1);
   myAND_1: myAND port map(net_29,
    net_58, net_212, vdd, A1);
   myAND_2:       myAND       port
    map(net_149, net_212, net_217,
    vdd, A1);
   myAND_3:       myAND       port
    map(net_148, net_33, net_216,
    vdd, A1);
   myInv_0:  myInv  port  map(A1,
    net_15, vdd, A1);
   myInv_1:  myInv  port  map(A0,
    net_17, vdd, A1);
      ……..
      ……..
      ……..
   myXNOR_1:      myXNOR      port
    map(net_26,  net_49,  net_62,
    vdd, A1);
   myXNOR_2:      myXNOR      port
    map(net_29,  net_32,  net_60,
    vdd, A1);
   myXNOR_3:      myXNOR      port
    map(net_33,  net_57,  net_58,
    vdd, A1);
   pin_184: ground port map(A1);
   pin_192: power port map(vdd);
end E_4CompBARU_BODY;
```

```
regA(2) <= (not(not x(2) nand
y(2))) or (not(x(2) nand (not
y(2))));
regA(3) <= (not(not x(3) nand
y(3))) or (not(x(3) nand (not
y(3))));
……..
……..
……..

(not(y(2) nand (not x(2))));
regB(3) <= (not(not y(3) nand
x(3))) or (not(y(3) nand (not
x(3))));
regB(4) <= not inAkecilB or
regB(0) or regB(1) or regB(2) or
regB(3);
regB(5) <= (y(0) nand (not x(0)))
or regB(1) or regB(2) or regB(3);
regB(6) <= (y(1) nand (not x(1)))
or regB(2) or regB(3);
regB(7) <= (y(2) nand (not x(2)))
or regB(3);
regB(8) <= ((y(3) nand (not
x(3))) and regB(7) and regB(6)
and regB(5)) nand regB(4) ;
outAkecilB <= regB(8);

outAsamaB <= not((inAsamaB and
(not regA(8))) nand (not
regB(8)));

end Behavioral;
```

Dari tabel 4.4 *behavior* dari comparator 4 bit telah terdeskripsikan pada level gerbang logika dasar sehingga dengan penyesuaian yang sederhana didapatkan kode program VHDL dari comparator 4 bit. Untuk membuat comparator 8 bit dibutuhkan 2 buah comparator 4 bit, sehingga VHDL comparator 8 bit terbentuk dari VHDL comparator 4 bit yang dihubungkan setiap port input dan outputnya menggunakan *port map*. Berikut kode program VHDL dari compararor 8 bit :

```
begin
reg1 <= x(3 downto 0);
reg2 <= y(3 downto 0);
reg3 <= x(7 downto 4);
reg4 <= y(7 downto 4);
tahap1 : comp4Fungsi port
map(reg1,reg2,ReginAkecilB,ReginAbesarB,ReginAsamaB,temp1,temp2,temp3);
tahap2 : comp4Fungsi port map(reg3,reg4,temp1,temp3,temp2,outAkecilB, outAsamaB,
outAbesarB);
end Behavioral;
```

4.1.5 VHDL Euclid Distance

Kode program VHDL euclid distance merupakan *behavior* layout CMOS euclid distance. Untuk lebih jelas terkait penyesuaian tersebut dapat dilihat pada Tabel 4.7.

Tabel 4.7 *Behavior* dan VHDL Euclid Distance

Behavior layout CMOS euclid distance	Kode program VHDL euclid distance
begin *myFM8_0: myFM8 port map(net_64, net_71, net_80, net_86, net_94, net_103, net_118, net_121, net_64, net_71, net_80, net_86, net_94, net_103, net_118, net_121, P_0_, P_1_, P_2_, P_3_, P_4_, P_5_, P_6_, P_7_, P_8_, P_9_, P_10_, P_11_, P_12_, P_13_, P_14_, P_15_, gnd, vdd);* *myFS8new_1: myFS8new port map(BIN, BOUT_FS, net_64, net_71, net_80, net_86, net_94, net_103, net_118, net_121, X_0_, X_1_, X_2_, X_3_, X_4_, X_5_, X_6_, X_7_, Y_0_, Y_1_, Y_2_, Y_3_, Y_4_, Y_5_, Y_6_, Y_7_, gnd, vdd);* *pin_1: ground port map(gnd);* *pin_2: power port map(vdd);* *end EuclidDistance_BODY;*	*tahap1 : comp8Bit port map(a,b,regOut1,regOut2,regOut3);* *tahap2 : mySwitching port map(a,b,regOut1,rst,clk,temp1,temp2);* *tahap3 : FS81 port map(binku,temp1,temp2,temp3,bout);* *tahap4 : FM82 port map(temp3,temp3,c);*

Dari tabel 4.7 *behavior* dari euclid distance sebenarnya merupakan VHDL yang digenerate dari layout CMOS euclid distance, yang terdiri dari komponen layout CMOS full subtractor 8 bit dan full multiplicator 8 bit. Karena *behavior* tersebut terbentuk dari dua buah standar cell, maka kode program VHDL untuk euclid distance juga terdiri dari komponen VHDL full subtractor 8 bit dan full multiplicator 8 bit.

4.1.6 VHDL CostMatriksA

Kode program VHDL cost matriks A merupakan *behavior* layout CMOS cost matriks A. Untuk lebih jelas terkait penyesuaian tersebut dapat dilihat pada Tabel 4.8.

Tabel 4.8 *Behavior* dan VHDL CostMatriks A

Behavior layout CMOS CostMatriksA	Kode program VHDL CostMatriksA
begin *EuclidDi_0: EuclidDistance port map(BIN, open, net_199, net_208, net_190,* *net_187, net_184, net_181,* *net_193, net_170, net_285,* *net_284, net_261,* *net_264, net_267, net_70,* *net_273, net_276, X0, X1, X2, X3,* *X4, X5, X6, X7,* *Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7,* *gnd, vdd);* *myFA8_0: myFA8 port map(net_199,* *net_208, net_190, net_187,* *net_184, net_181,* *net_193, net_170, CmIn0, CmIn1,* *CmIn2, CmIn3, CmIn4, CmIn5, CmIn6,* *CmIn7,* *gnd, net_249, CmOut0, CmOut1,* *CmOut2, CmOut3, CmOut4, CmOut5,* *CmOut6,* *CmOut7, gnd, vdd);* *myFA8_1: myFA8 port map(net_285,* *net_284, net_261, net_264,* *net_267, net_70,* *net_273, net_276, CmIn8, CmIn9,* *CmIn10, CmIn11, CmIn12, CmIn13,* *CmIn14,* *CmIn15, net_249, CmOut_out,* *CmOut8, CmOut9, CmOut10, CmOut11,* *CmOut12,* *CmOut13, CmOut14, CmOut15, gnd,* *vdd);* *pin_245: power port map(vdd);* *pin_271: ground port map(gnd);* *end CostMatriksA_BODY;*	*begin* *tahap1 : myEuclidDistance port map(inEuclidA,inEuclidB,clk,rst,boutEuclid,tem* *tahap2 : FA16 port map(cinku,temp,inCostMatriks,c,coutFA16);* *end Behavioral;*

Dari tabel 4.8 *behavior* dari cost matriks A sebenarnya merupakan VHDL yang digenerate dari layout CMOS cost matriks A, yang terdiri dari komponen layout CMOS euclid distance dan full adder 8 bit. Karena *behavior* tersebut terbentuk dari dua buah komponent, maka kode program VHDL untuk cost matriks A juga terdiri dari komponen VHDL euclid distance dan full adder 8 bit.

4.1.7 VHDL CostMatriksB

Kode program VHDL cost matriks B merupakan *behavior* layout CMOS cost matriks B. Untuk lebih jelas terkait penyesuaian tersebut dapat dilihat pada Tabel 4.9.

Tabel 4.9 *Behavior* dan VHDL CostMatriks B

Behavior standar cell full adder 1 bit	Kode program VHDL full adder 1 bit
begin *EuclidDi_0: EuclidDistance port map(gnd, open, net_71, net_70, net_69,* *net_68, net_67, net_66, net_65, net_64, net_72, net_73, net_74, net_75,* *net_76, net_77, net_78, net_79, edA0, edA1, edA2, edA3, edA4, edA5, edA6,* *edA7, edB0, edB1, edB2, edB3, edB4, edB5, edB6, edB7, gnd, vdd);* *MinGloba_0: MinGlobalCost port map(net_83, net_84, net_85, net_86, net_87,* *net_88, net_89, net_90, gnd, mgcA0, mgcA1, mgcA2, mgcA3, mgcA4, mgcA5,* *mgcA6, mgcA7, mgcB0, mgcB1, mgcB2, mgcB3, mgcB4, mgcB5, mgcB6, mgcB7,* *mgcC0, mgcC1, mgcC2, mgcC3, mgcC4, mgcC5, mgcC6, mgcC7, vdd);* *myFA8_0: myFA8 port map(gnd, gnd, gnd, gnd, gnd, gnd, gnd, gnd, net_72,* *net_73, net_74, net_75, net_76, net_77, net_78, net_79, gnd, net_142,* *output8, output9, output10, output11, output12, output13, output14,* *output15, gnd, vdd);* *myFA8_1: myFA8 port map(net_83, net_84, net_85, net_86, net_87, net_88,* *net_89, net_90, net_71, net_70, net_69, net_68, net_67, net_66, net_65,* *net_64, gnd, net_93, output0, output1, output2, output3, output4, output5,* *output6, output7, gnd, vdd);* *pin_337: power port map(vdd);* *pin_338: ground port map(gnd);* *end CostMatriksB_BODY;*	*begin* *tahap1 : myEuclidDistance port map (inEuclidA,inEuclidB,clk,rst,boutEuclid,temp);* *tahap2 : myMGC port map(inMGCp,inMGCq,inMGCr,clk,rst,temp2);* *temp4(7 downto 0) <= temp2;* *temp4(15 downto 8) <= temp3;* *tahap3 : FA16 port map(cinku,temp,temp4,s,coutFA16);* *end Behavioral;*

Dari tabel 4.9 *behavior* dari cost matriks B sebenarnya merupakan VHDL yang digenerate dari layout CMOS cost matriks B, yang terdiri dari komponen layout CMOS euclid distance, comparator 8 bit dan full adder 8 bit. Karena *behavior*

tersebut terbentuk dari tiga buah komponen, maka kode program VHDL untuk cost matriks B juga terdiri dari komponen VHDL euclid distance, comparator 8 bit dan full adder 8 bit.

4.1.8 VHDL FastDTW

Kode program VHDL fast DTW merupakan *behavior* layout CMOS fast DTW. Untuk lebih jelas terkait penyesuaian tersebut dapat dilihat pada Tabel 4.10.

Tabel 4.10 *Behavior* dan VHDL FastDTW

Behavior layout CMOS fast DTW	Kode program VHDL fast DTW
begin *CostMatr_0: CostMatriksA port map(gnd, net_0, net_41, net_43, net_42, net_44, net_47, net_45, net_46, net_36, net_37, net_48, net_49, net_38, net_39, net_50, net_35, net_68, net_69, net_70, net_71, net_174, net_73, net_74, net_75, open, open, open, open, open, open, open, open, tsiA0, tsiA1, tsiA2, tsiA3, tsiA4, tsiA5, tsiA6, tsiA7, tsjB0, tsjB1, tsjB2, tsjB3, tsjB4, tsjB5, tsjB6, tsjB7, gnd, vdd);* *CostMatr_2: CostMatriksB port map(tsiB0, tsiB1, tsiB2, tsiB3, tsiB4, tsiB5, tsiB6, tsiB7, tsjB0, tsjB1, tsjB2, tsjB3, tsjB4, tsjB5, tsjB6, tsjB7, gnd, net_68, net_69, net_70, net_71, net_174, net_73, net_74, net_75, net_0, net_41, net_43, net_42, net_44, net_47, net_45, net_46, net_120, net_121, net_122, net_123, net_124,* *……..* *……..* *……..* *CostMatr_19: CostMatriksB port map(tsiC0, tsiC1, tsiC2, tsiC3, tsiC4, tsiC5, tsiC6, tsiC7, tsjB0, tsjB1, tsjB2, tsjB3, tsjB4, tsjB5, tsjB6, tsjB7, gnd, net_178, net_180, net_179, net_181, net_182, net_183,*	*begin* *tahap0 : myEuclidDistance port map(memoryExtract(0),memoryFeature(0),clk,rst,bout0(0,0),temp0(0,0));* *tahap1 : for j in 1 to 3 generate* *IsamaNol: costMatriksA port map(memoryExtract(0),memoryFeature(j),temp0(0,j-1),clk,rst,bout0(0,j),cout0(0,j),temp0(0,j));* *end generate tahap1;* *tahap2 : for i in 1 to 3 generate* *JsamaNol : costMatriksA port map(memoryExtract(i),memoryFeature(0),temp0(i-1,0),clk,rst,bout0(i,0),cout0(i,0),temp0(i,0));* *end generate tahap2;*

```
    net_184, net_185,
      net_120,  net_121,  net_122,
    net_123,   net_124,   net_125,
    net_126, net_127,
      vdd,  vdd,  vdd,  vdd,  vdd,
    vdd,   vdd,   gnd,   net_381,
    net_375, net_380, net_379,
      net_378,  net_377,  net_376,
    net_374,  open,  open,  open,
    open, open, open,
      open, open, vdd);
   EuclidDi_0:     EuclidDistance
    port  map(gnd,  open,  net_0,
    net_41, net_43, net_42,
      net_44,    net_47,    net_45,
    net_46,    net_36,    net_37,
    net_48, net_49, net_38,
      net_39,    net_50,    net_35,
    tsiA0,  tsiA1,  tsiA2,  tsiA3,
    tsiA4, tsiA5, tsiA6,
      tsiA7, tsjA0, tsjA1, tsjA2,
    tsjA3,  tsjA4,  tsjA5,  tsjA6,
    tsjA7, gnd, vdd);
   contact_1079:   ground   port
    map(gnd);
   pin_1090:      power      port
    map(vdd);
end FullDatapathTest_BODY;
```

```
tahap3 : for i in 1 to 3 generate

tahap4 : for j in 1 to 3 generate

 IJtaksamaNol : costMatriksB port
   map(memoryExtract(i),memoryFeat
    ure(j),temp0(i-1,j)(7 downto
     0),temp0(i-1,j-1)(7 downto
      0),temp0(i,j-1)(7 downto
  0),clk,rst,bout0(i,j),cout0(i,j
          ),temp0(i,j));

     end generate tahap4;

              end
```

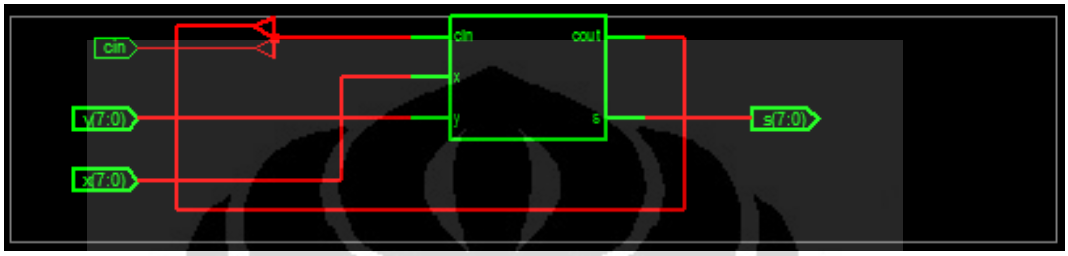
Dari tabel 4.10 *behavior* dari fast DTW sebenarnya merupakan VHDL yang digenerate dari layout CMOS fast DTW, yang terdiri dari komponen layout CMOS euclid distance, cost matriks A dan cost matriks B. Karena *behavior* tersebut terbentuk dari tiga buah komponen, maka kode program VHDL untuk fast DTW juga terdiri dari komponen VHDL euclid distance, cost matriks A dan cost matriks B.

4.2 Rangkaian Skematik dan Simulasi Fungsional IC DTW pada SPARTAN IIELC

Setelah kode program VHDL dari IC DTW dibuat, selanjutnya akan dilihat rangkaian skematik dan simulasi fungsional yang dihasilkan dari kode program VHDL tersebut. Rangkaian skematik dapat digunakan untuk melihat gerbang logika dasar yang digunakan dan sebagai perbandingan dalam proses konversi komponen pada teknologi FPGA. Sedangkan simulasi fungsional dalam hal ini diagram waktu, digunakan untuk melihat apakah kode program VHDL yang dibuat mengeluarkan output yang sesuai dengan output setiap proses pada IC DTW.

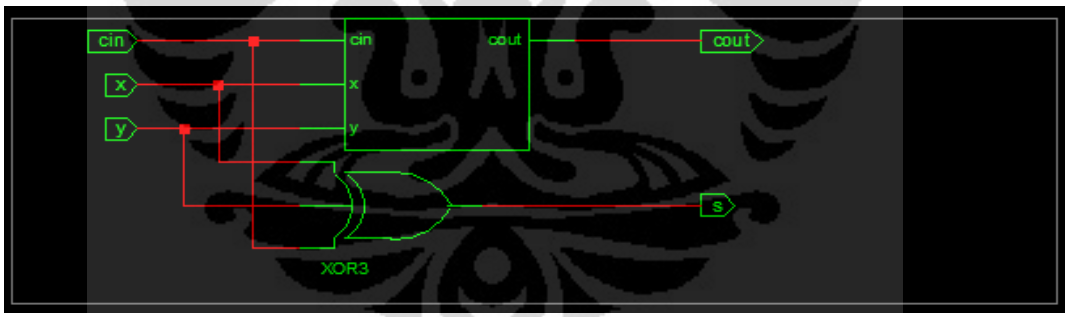
4.2.1 Rangkaian Skematik dan Simulasi Fungsional Full Adder 8 bit

Kode program VHDL full adder 8 bit menggunakan gerbang logika dasar AND, OR dan XOR sehingga rangkaian skematik full adder 8 bit yang dihasilkan akan memperlihatkan gerbang logika dasar tersebut. Gambar 4.1 merupakan rangkaian skematik full adder 8 bit yang terdiri dari port input cin, x[7:0] dan y[7:0], port output s[7:0], multiplexer dan blok proses full adder 1 bit.



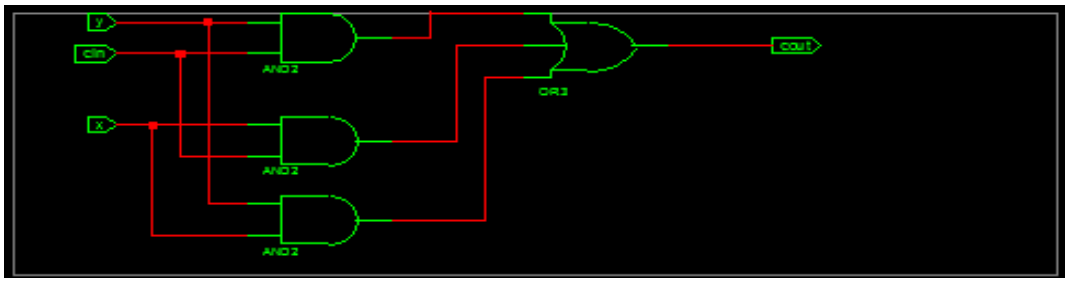
Gambar 4.1 Rangkaian skematik Full Adder 8 bit

Gambar 4.2 merupakan rangkaian skematik blok proses full adder 1 bit yang terdiri dari port input cin, x dan y, port output cout dan s, gerbang XOR 3 input dan blok proses cout.



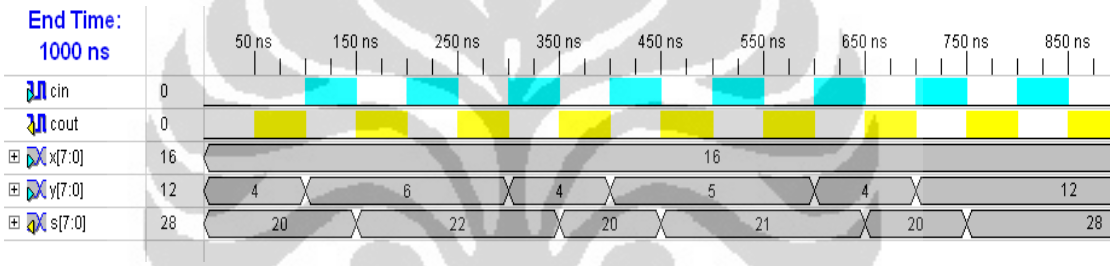
Gambar 4.2 Rangkaian skematik Full Adder 1 bit

Gambar 4.3 merupakan rangkaian skematik blok proses cout yang terdiri dari port input cin, x dan y, port output cout, gerbang AND 2 input dan gerbang OR 3 input.



Gambar 4.3 Rangkaian skematik blok proses cout

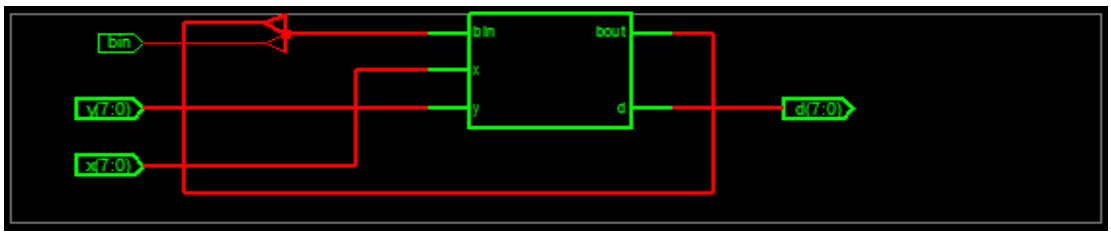
Gambar 4.4 memperlihatkan simulasi fungsional yang dihasilkan dari kode program VHDL full adder 8 bit. Terlihat bahwa simulasi fungsional yang dihasilkan sesuai dengan proses full adder 8 bit yang diinginkan. Sebagai contoh ketika input x[7:0] = 16 dan y[7:0] = 12 akan dihasilkan output s[7:0] = 28.



Gambar 4.4 Simulasi fungsional Full Adder 8 bit

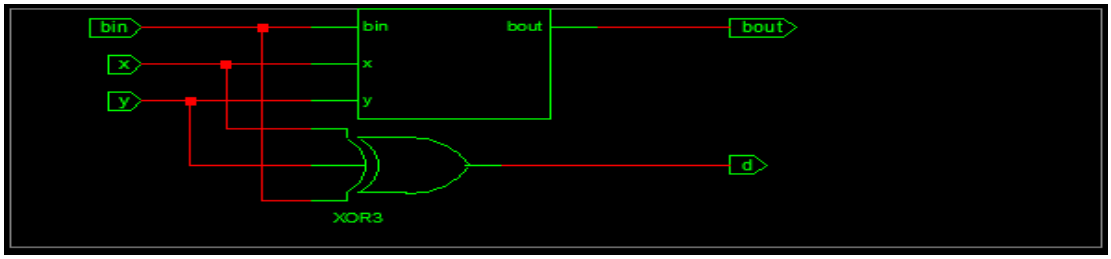
4.2.2 Rangkaian Skematik dan Simulasi Fungsional Full Subtractor 8 bit

Kode program VHDL full subtractor 8 bit menggunakan gerbang logika dasar AND, OR, inverter dan XOR sehingga rangkaian skematik full subtractor 8 bit yang dihasilkan akan memperlihatkan gerbang logika dasar tersebut. Gambar 4.5 merupakan rangkaian skematik full subtractor 8 bit yang terdiri dari port input bin, x[7:0] dan y[7:0], port output d[7:0], multiplexer dan blok proses full subtractor 1 bit.



Gambar 4.5 Rangkaian skematik Full Subtractor 8 bit

Gambar 4.6 merupakan rangkaian skematik blok proses full subtractor 1 bit yang terdiri dari port input bin, x dan y, port output bout dan d, gerbang XOR 3 input dan blok proses bout.

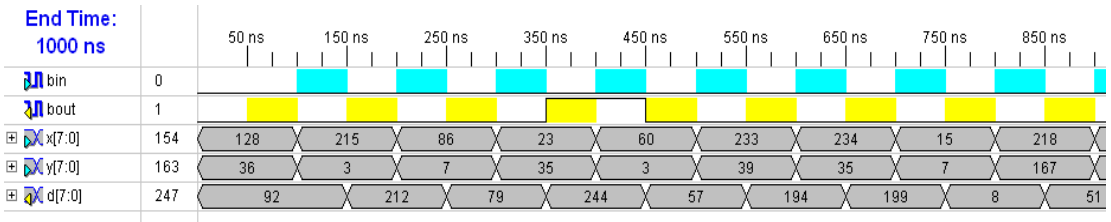


Gambar 4.6 Rangkaian skematik Full Subtractor 1 bit

Gambar 4.7 merupakan rangkaian skematik blok proses bout yang terdiri dari port input bin, x dan y, port output bout, inverter, gerbang AND 2 input dan gerbang OR 3 input.

Gambar 4.7 Rangkaian skematik blok proses bout

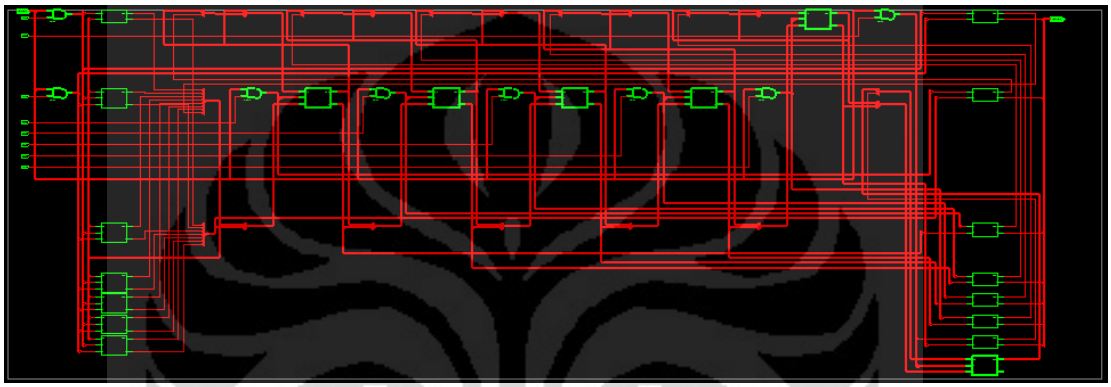
Gambar 4.8 memperlihatkan simulasi fungsional yang dihasilkan dari kode program VHDL full subtractor 8 bit. Terlihat bahwa simulasi fungsional yang dihasilkan sesuai dengan proses full subtractor 8 bit yang diinginkan. Sebagai contoh ketika input x[7:0] = 128 dan y[7:0] = 36 akan dihasilkan output d[7:0] = 96.



Gambar 4.8 Simulasi fungsional Full Subtractor 8 bit

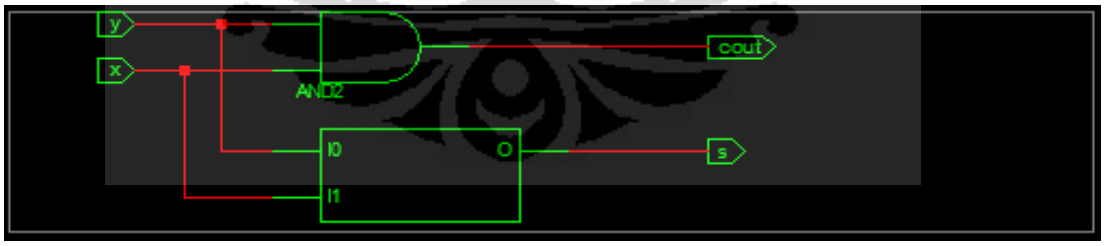
4.2.3 Rangkaian Skematik dan Simulasi Fungsional Full Multiplicator 8 bit

Kode program VHDL full multiplicator 8 bit menggunakan gerbang logika dasar AND, full adder 1 bit dan half adder 1 bit sehingga rangkaian skematik full multiplicator 8 bit yang dihasilkan akan memperlihatkan gerbang logika tersebut. Gambar 4.9 merupakan rangkaian skematik full multiplicator 8 bit yang terdiri dari port input cin, x[7:0] dan y[7:0], port output p[15:0], gerbang logika dasar AND, blok proses half adder 1 bit dan blok proses full adder 1 bit. Rangkaian skematik blok proses full adder 1 bit terlihat pada Gambar 4.2.



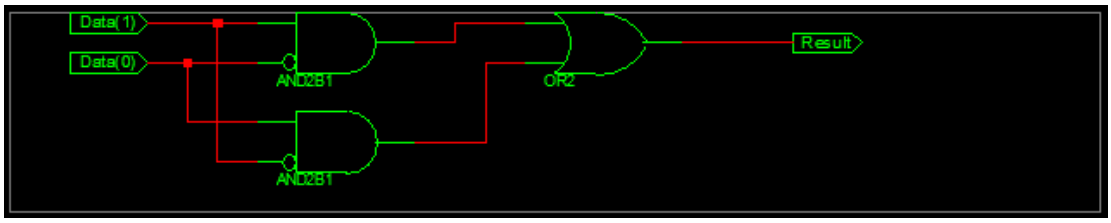
Gambar 4.9 Rangkaian skematik Full Multiplicator 8 bit

Gambar 4.10 merupakan rangkaian skematik blok proses half adder 1 bit yang terdiri dari port input x dan y, port output cout dan s, gerbang AND 2 input dan blok proses s.



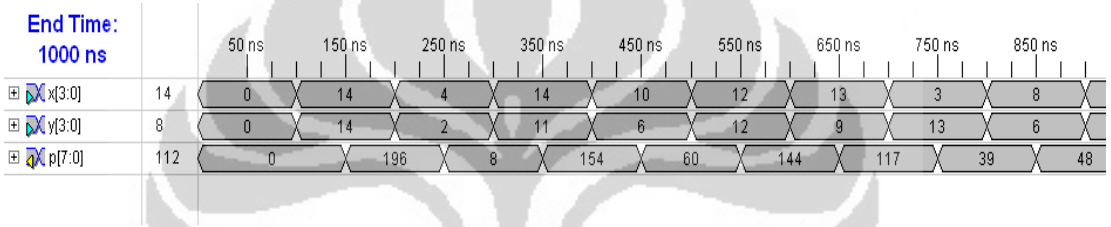
Gambar 4.10 Rangkaian skematik Half Adder 1 bit

Gambar 4.11 merupakan rangkaian skematik blok proses s yang terdiri dari port input data[0] dan data[1], port output result, gerbang AND 2 input, gerbang OR 2 input dan inverter.



Gambar 4.11 Rangkaian skematik blok proses s

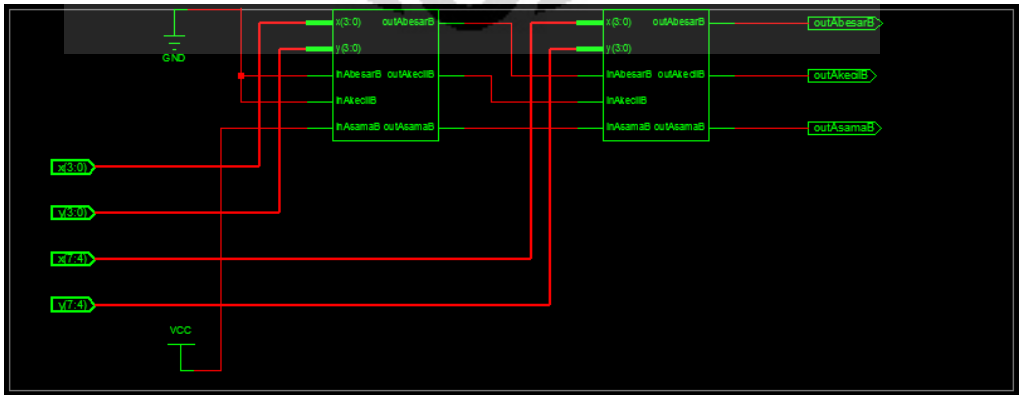
Gambar 4.12 memperlihatkan simulasi fungsional yang dihasilkan dari kode program VHDL full multiplicator 8 bit. Terlihat bahwa simulasi fungsional yang dihasilkan sesuai dengan proses full multiplicator 8 bit yang diinginkan. Sebagai contoh ketika input x[7:0] = 14 dan y[7:0] = 11 akan dihasilkan output p[15:0] = 154.



Gambar 4.12 Simulasi fungsional Full Multiplicator 8 bit

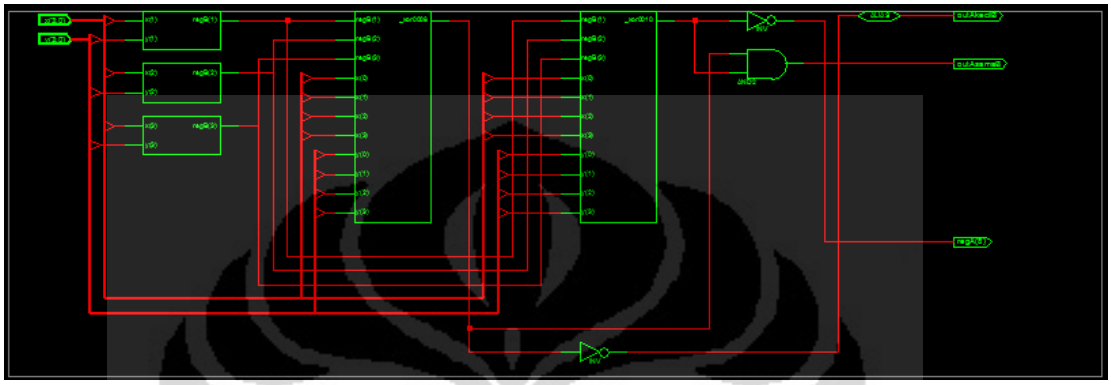
4.2.4 Rangkaian Skematik dan Simulasi Fungsional Comparator 8 bit

Kode program VHDL comparator 8 bit menggunakan comparator 4 bit sehingga rangkaian skematik comparator 8 bit yang dihasilkan akan memperlihatkan dua buah comparator 4 bit. Gambar 4.13 merupakan rangkaian skematik comparator 8 bit yang terdiri dari port input x[7:0] dan y[7:0], blok proses comparator 4 bit dan port output outAbesarB, outAkecilB dan outAsamaB.



Gambar 4.13 Rangkaian skematik Comparator 8 bit

Gambar 4.14 merupakan rangkaian skematik blok proses comparator 4 bit yang terdiri dari port input x[3:0] dan y[3:0], port output outAbesarB, outAkecilB dan outAsamaB, gerbang AND 2 input, inverter, blok proses register 1 dan blok proses register 2. Baik blok proses register 1 atau blok proses register 2 merupakan kombinasional dari operasi gerbang logika dasar, dimana rangkaian skematiknya dapat dilihat pada Gambar 4.15 dan 4.16.

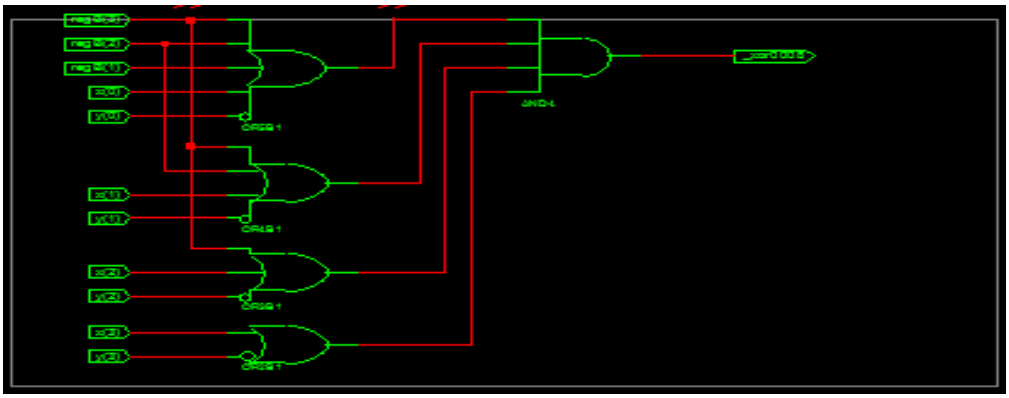


Gambar 4.14 Rangkaian skematik Comparator 4 bit

Gambar 4.15 merupakan rangkaian skematik blok proses register 1 yang terdiri dari port input x[1] dan y[1], port output regB[1], gerbang AND 2 input, inverter dan gerbang OR 2 input.

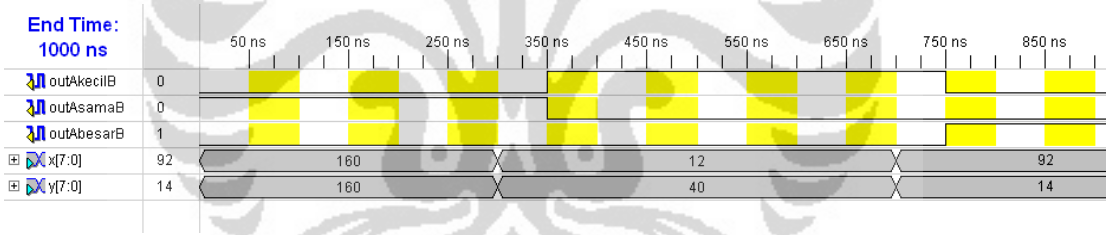
Gambar 4.15 Rangkaian skematik blok proses register 1

Gambar 4.16 merupakan rangkaian skematik blok proses register 2 yang terdiri dari beberapa port input, port output register 2, gerbang AND 4 input dan gerbang OR 2 input, 3 input, 4 input dan 5 input.



Gambar 4.16 Rangkaian skematik blok proses register 2

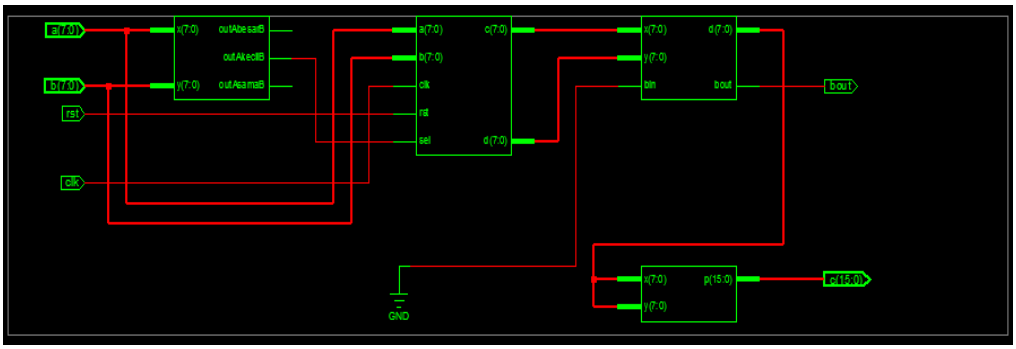
Gambar 4.17 memperlihatkan simulasi fungsional yang dihasilkan dari kode program VHDL comparator 8 bit. Terlihat bahwa simulasi fungsional yang dihasilkan sesuai dengan proses comparator 8 bit yang diinginkan. Sebagai contoh ketika input x[7:0] = 160 dan y[7:0] = 160 akan dihasilkan output outAsamaB = 1, x[7:0] = 12 dan y[7:0] = 40 akan dihasilkan output outAkecilB = 1 dan x[7:0] = 92 dan y[7:0] = 14 akan dihasilkan output outAbesarB = 1.



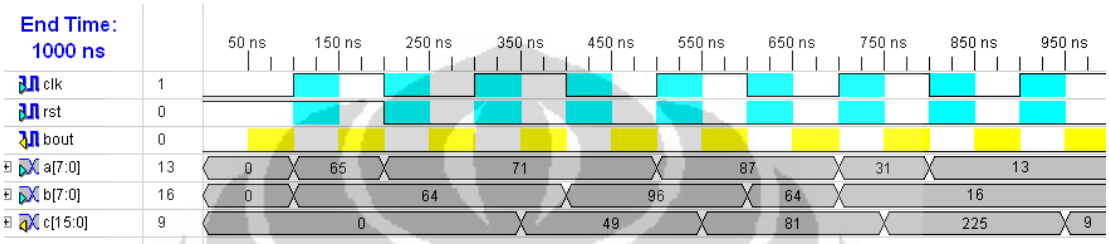
Gambar 4.17 Simulasi fungsional Comparator 8 bit

4.2.5 Rangkaian Skematik dan Simulasi Fungsional Euclid Distance

Rangkaian skematik yang dihasilkan dari VHDL euclid distance terdiri dari komponen comparator 8 bit, full subtractor 8 bit, full multiplicator 8 bit dan multiplexer 16 menjadi 8 bit seperti terlihat pada Gambar 4.23. Sedangkan Gambar 4.24 memperlihatkan simulasi fungsional yang dihasilkan dari kode program VHDL euclid distance. Terlihat bahwa simulasi fungsional yang dihasilkan sesuai dengan proses euclid distance yang diinginkan. Sebagai contoh saat input a[7:0] = 71 dan b[7:0] = 64 maka didapatkan output c[15:0] = 49, hal ini sesuai dengan operasi pada euclid distance yaitu $(71 - 64)^2 = 49$.



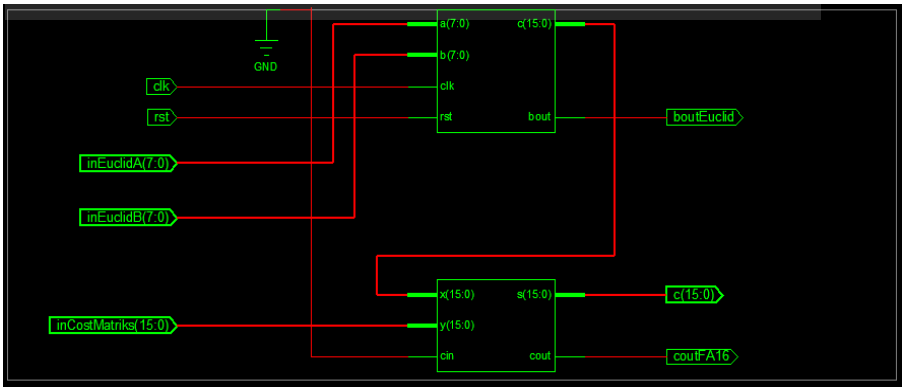
Gambar 4.23 Rangkaian skematik Euclid Distance



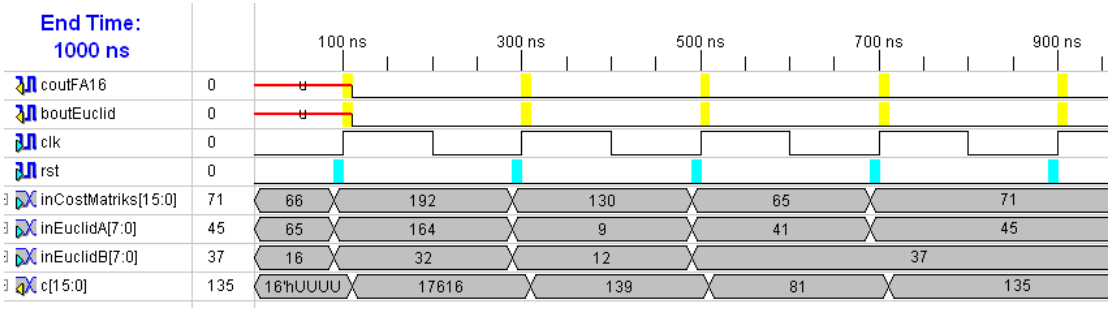
Gambar 4.24 Simulasi fungsional Euclid Distance

4.2.6 Rangkaian Skematik dan Simulasi Fungsional CostMatriksA

Rangkaian skematik yang dihasilkan dari VHDL cost matriks A terdiri dari komponen euclid distance dan full adder 16 bit seperti terlihat pada Gambar 4.25. Sedangkan Gambar 4.26 memperlihatkan simulasi fungsional yang dihasilkan dari kode program VHDL cost matriks A. Terlihat bahwa simulasi fungsional yang dihasilkan sesuai dengan proses cost matriks A yang diinginkan. Sebagai contoh saat input inEuclidA[7:0] = 41, inEuclidB[7:0] = 37 dan inCostMatriks[15:0] = 65 maka didapatkan output c[15:0] = 81, hal ini sesuai dengan operasi pada cost matriks A yaitu $(41 - 37)^2 + 65 = 16 + 65 = 81$.



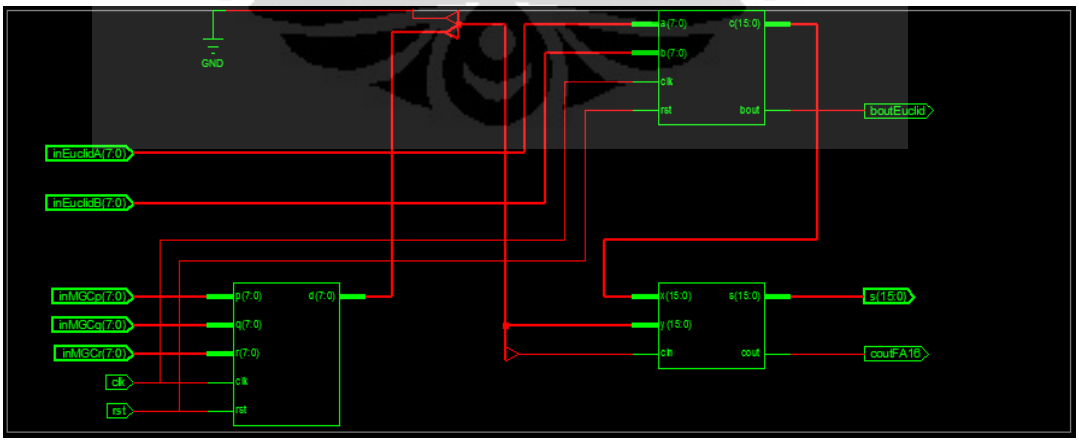
Gambar 4.25 Rangkaian skematik costMatriksA



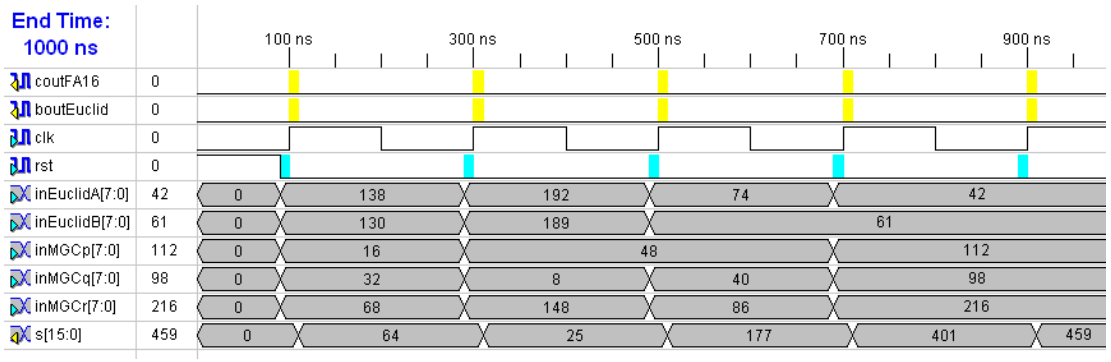
Gambar 4.26 Simulasi fungsional costMatriksA

4.2.7 Rangkaian Skematik dan Simulasi Fungsional CostMatriksB

Rangkaian skematik yang dihasilkan dari VHDL cost matriks B terdiri dari komponen euclid distance, comparator 8 bit dan full adder 16 bit seperti terlihat pada Gambar 4.27. Sedangkan Gambar 4.28 memperlihatkan simulasi fungsional yang dihasilkan dari kode program VHDL cost matriks B. Terlihat bahwa simulasi fungsional yang dihasilkan sesuai dengan proses cost matriks B yang diinginkan. Sebagai contoh saat input inEuclidA[7:0] = 192, inEuclidB[7:0] = 189, inMGCp[7:0] = 16, inMGCq[7:0] = 32 dan inMGCr[7:0] = 68 maka didapatkan output sc[15:0] = 25, hal ini sesuai dengan operasi pada cost matriks B yaitu $(192 - 189)^2$ + min(16, 32, 68) = 9 + 16 = 25.



Gambar 4.27 Rangkaian skematik costMatriksB

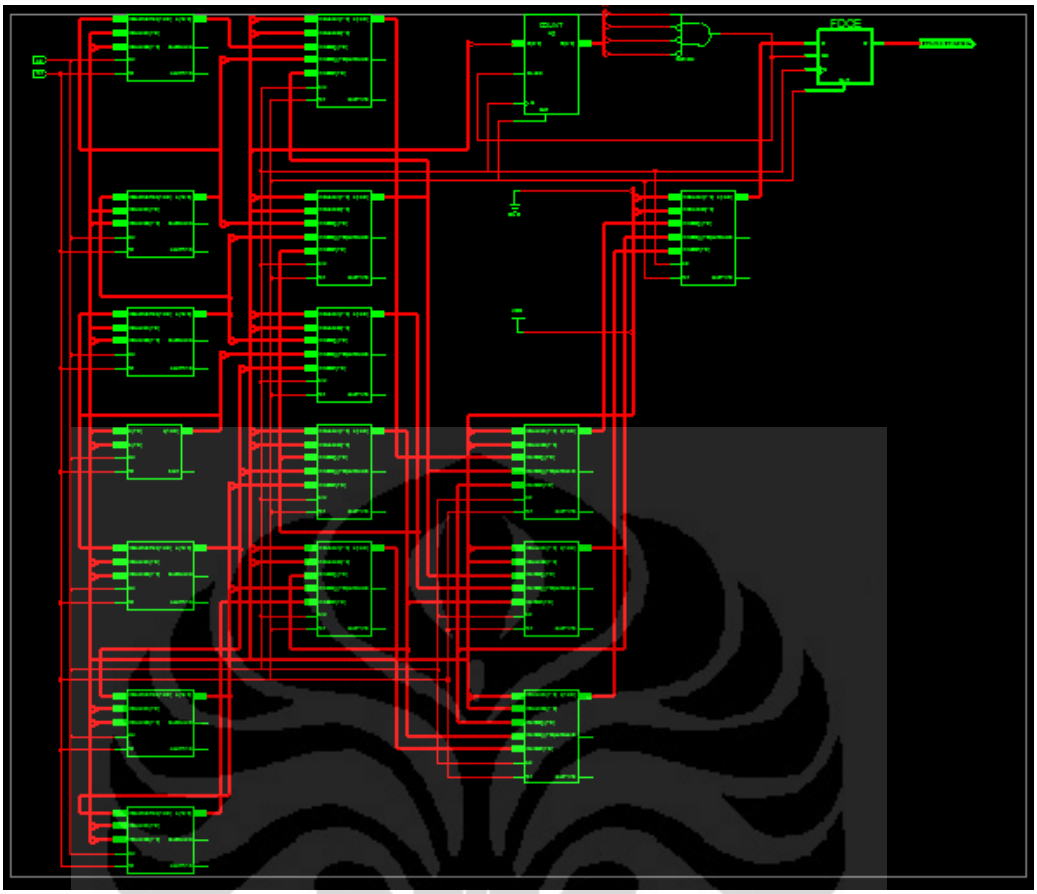


Gambar 4.28 Simulasi fungsional costMatriksB

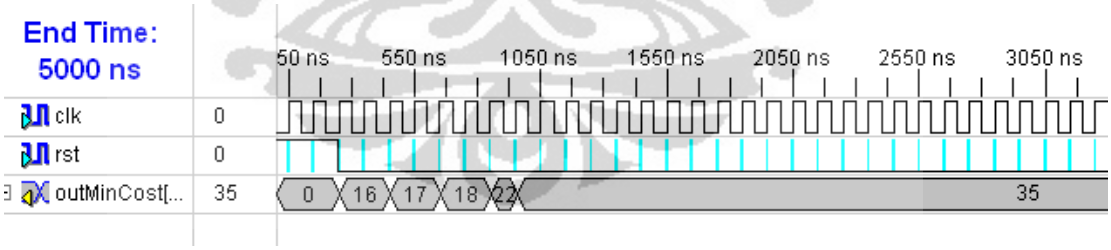
4.2.8 Rangkaian Skematik dan Simulasi Fungsional FastDTW

Rangkaian skematik yang dihasilkan dari VHDL fast DTW terdiri dari komponen euclid distance, cost matriks A dan cost matriks B seperti terlihat pada Gambar 4.29. Sedangkan Gambar 4.30 memperlihatkan simulasi fungsional yang dihasilkan dari kode program VHDL fast DTW tanpa counter waktu. Terlihat bahwa simulasi fungsional yang dihasilkan belum sesuai dengan proses fast DTW yang diinginkan pada waktu pertama setelah reset = 0. Hasil akan menjadi benar ketika waktu kedelapan setelah reset = 0, sebagai contoh input ditentukan secara statis yaitu tsI = [6, 7, 7, 8] dan tsJ = [5, 2, 6, 4] maka output yang dihasilkan seharusnya 35.

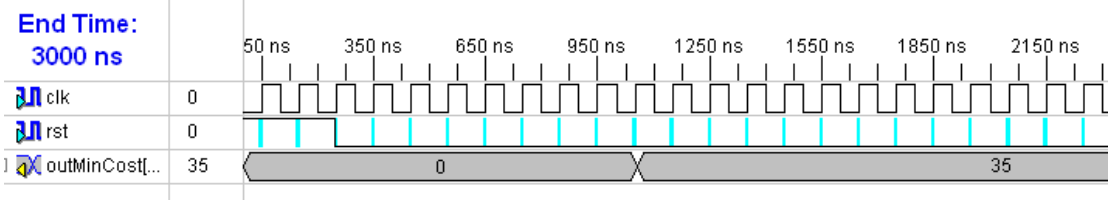
Dalam Gambar 4.30 terlihat output pada waktu pertama dan kedua yaitu 16, output waktu ketiga dan keempat yaitu 17, output waktu kelima dan keenam yaitu 18 dan output waktu ketujuh yaitu 22. Ouput yang diharapkan baru muncul setelah waktu kedelapan yaitu 35. Sehingga perlu diberikan counter waktu delapan *clock* untuk menghasilkan hasil yang diharapkan dan Gambar 4.31 merupakan simulasi fungsional yang dihasilkan dari kode program VHDL fast DTW dengan counter waktu. Terlihat bahwa simulasi fungsional yang dihasilkan sesuai dengan proses fast DTW yang diinginkan dengan waktu kedelapan setelah reset = 0.



Gambar 4.29 Rangkaian skematik Fast DTW



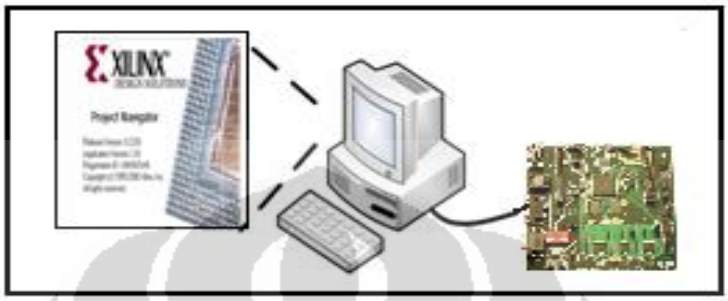
Gambar 4.30 Simulasi fungsional Fast DTW tanpa counter waktu



Gambar 4.31 Simulasi fungsional Fast DTW dengan counter waktu

4.3 Implementasi IC DTW pada SPARTAN IIELC

Pada sub bab 4.3 akan dibahas mengenai implementasi IC DTW pada *board* FPGA Spartan – IIELC. Implementasi yang dilakukan yaitu memasukkan kode program VHDL IC DTW pada *board* FPGA menggunakan kabel *downloader* JTAG paralel dan software Xilinx ISE 8.2i, seperti yang terlihat pada Gambar 4.32.

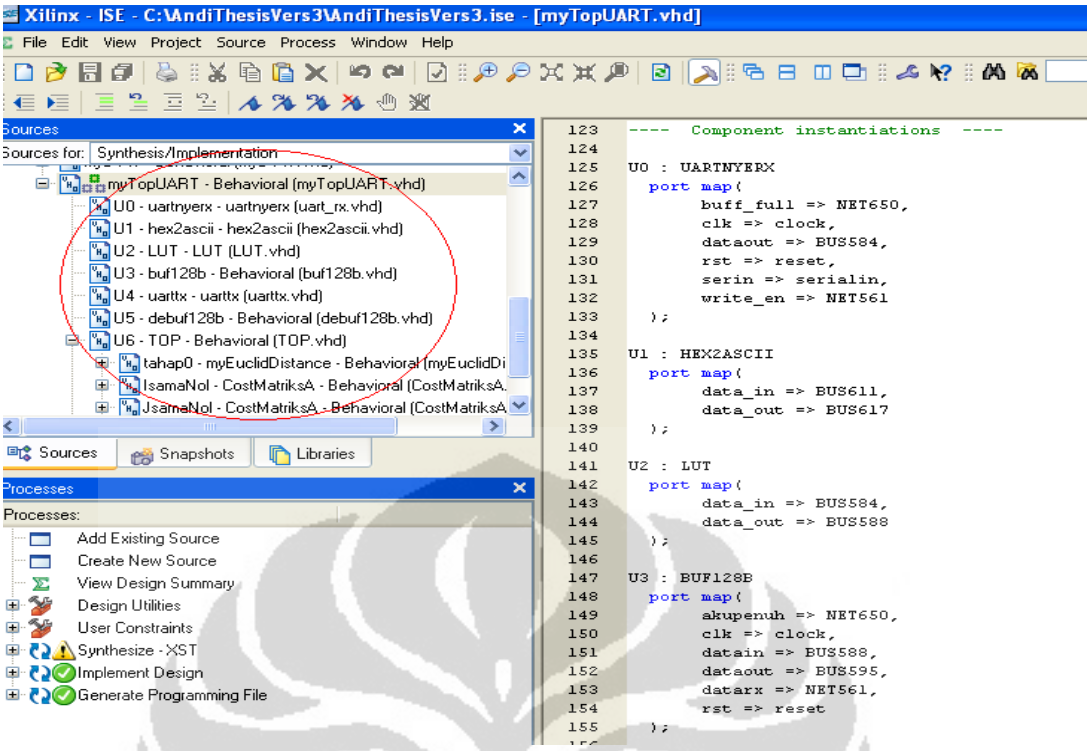


Gambar 4.32 Konfigurasi implementasi IC DTW pada *board* FPGA

Dalam implementasi tersebut perlu dilakukan pembatasan untuk melihat sejauh mana keberhasilan implementasi terhadap pembatasan tersebut. Batasan – batasan implementasi IC DTW pada FPGA Spartan IIELC adalah sebagai berikut :

1. Input berupa dua buah vektor integer (tsI dan tsJ) dengan panjang setiap vektor adalah 4 dan bersifat dinamis, artinya input tersebut nilainya dapat dirubah.
2. Input berasal dari nilai yang ada pada hyperterminal dengan format ASCII.
3. Output proses ditampilkan pada hyperterminal dengan format ASCII.
4. Proses *pattern matching* antara dua buah vektor input dilakukan FPGA Spartan IIELC menggunakan algoritma Fast DTW.
5. Hubungan antara hyperterminal dengan FPGA Spartan IIELC melalui komunikasi serial.

Untuk dapat melakukan implementasi dengan batasan yang telah didefinisikan perlu dilakukan pembuatan kode program VHDL yang mengintegrasikan kode program VHDL fast DTW, kode program VHDL *Look Up Table* (LUT), kode program VHDL buffer dan kode program VHDL untuk komunikasi serial. Hasil integrasi kode program VHDL tersebut dapat dilihat pada Gambar 4.33.

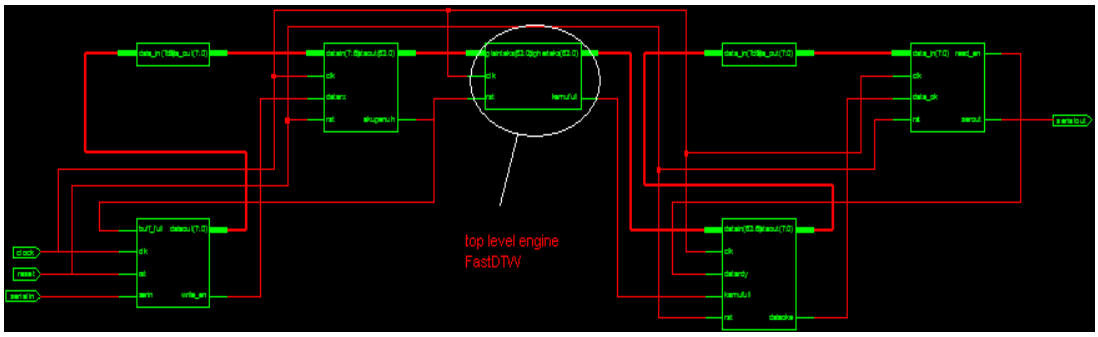


Gambar 4.33 Integrasi kode program VHDL

Lebih jelas mengenai kode program VHDL apa saja yang diintegrasikan dalam implementasi pada FPGA Spartan IIELC, dapat dilihat pada Tabel 4.11. Sedangkan Gambar 4.34 memperlihatkan rangkaian skematik yang dihasilkan dari proses integrasi kode program VHDL tersebut.

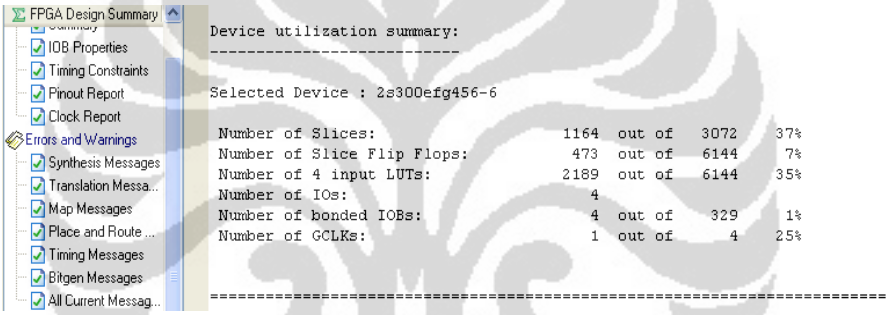
Tabel 4.11 Program VHDL yang diintegrasikan

No	Program VHDL	Keterangan
1.	TOP.vhd	Merupakan top level program VHDL dari *engine* fast DTW sesuai sub bab 4.1.10 dan 4.2.10.
2.	Uart_tx.vhd	Merupakan program VHDL untuk pengiriman komunikasi serial dan menerima 8 bit input untuk dikeluarkan menjadi 1 bit.
3.	Uart_rx.vhd	Merupakan program VHDL untuk penerimaan komunikasi serial dan memproses bit tersebut untuk dikeluarkan menjadi 8 bit.
4.	LUT.vhd	Merupakan program VHDL untuk mengkonversi ASCII menjadi binary integer.
5.	Buffer.vhd	Merupakan program VHDL untuk menampung input 8 bit dan menyimpannya dalam buffer 64 bit.
6.	Debuffer.vhd	Merupakan program VHDL untuk mengeluarkan output 8 bit dari buffer 64 bit.

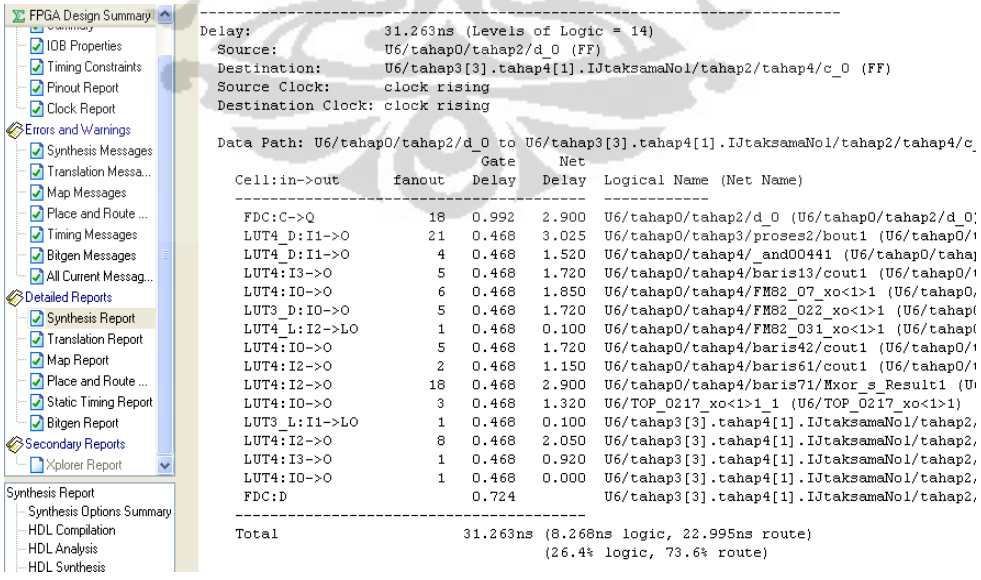


Gambar 4.34 Rangkaian skematik hasil integrasi kode program VHDL

Berdasarkan teknologi FPGA Spartan IIELC dalam hal ini 2s300efg456-6 maka jumlah komponent yang digunakan yaitu *slices* sebanyak 1164, *slices* flip flop sebanyak 473, 4 input LUT sebanyak 2189 dan blok IO sebanyak 4. Gambar 4.35 memperlihatkan jumlah penggunaan komponent pada FPGA Spartan IIELC.

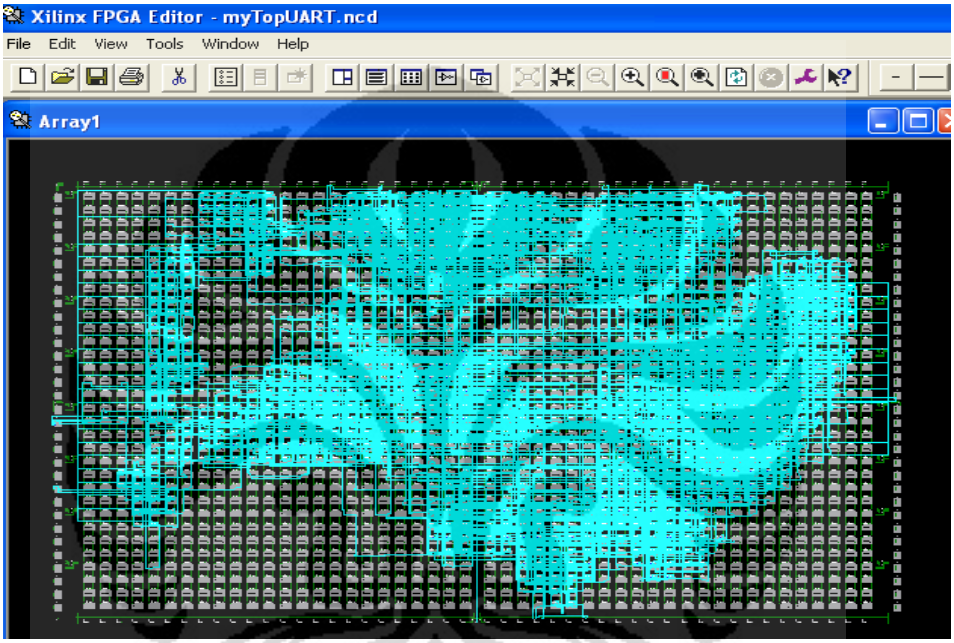


Gambar 4.35 Penggunaan komponen pada FPGA Spartan IIELC

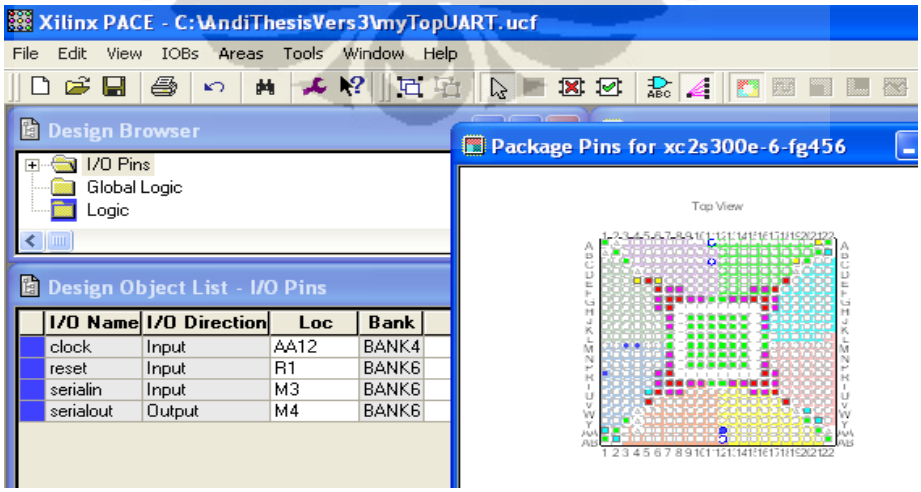


Gambar 4.36 *Delay* waktu proses IC DTW pada FPGA Spartan IIELC

Sedangkan *delay* waktu proses termasuk didalamnya *gate delay* dan *net delay* yang terjadi antar komponent pada *engine* fast DTW sebesar 31,263 ns dimana 8,268 ns untuk *logic* dan 22,995 ns untuk *routing*. Gambar 4.36 memperlihatkan delay yang terjadi untuk implementasi IC DTW pada FPGA Spartan IIELC. Adapun *placement* komponent dan *routing* antar *slice* diperlihatkan pada Gambar 4.37. Baik *placement* dan *routing* dilakukan menggunakan software Xilinx ISE 8.2i secara automatis ketika mengimplementasikan suatu desain pada *board* FPGA.



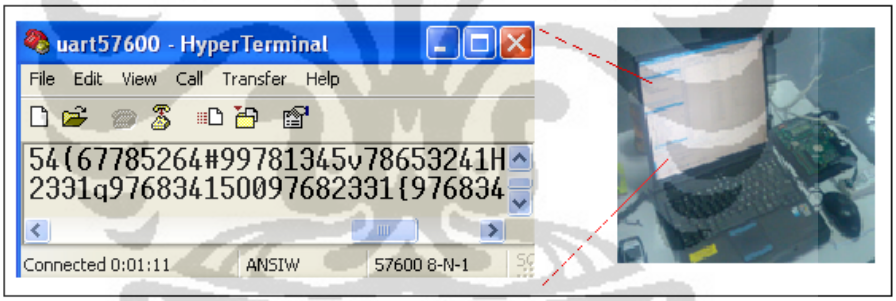
Gambar 4.37 *Routing* komponent IC DTW pada FPGA Spartan IIELC



Gambar 4.38 Konfigurasi pin IC DTW pada FPGA Spartan IIELC

Sesuai dengan kode program VHDL hasil integrasi yang akan diimplementasikan pada FPGA Spartan IIELC dibutuhkan pin input berupa *source clock* yang diambil dari internal oscillator pada FPGA tersebut, reset untuk mengembalikan kondisi awal dan masukan bit secara serial dari hyperterminal. Untuk output hanya membutuhkan 1 pin yaitu keluaran 1 bit yang akan mengirimkan bit secara serial ke hyperterminal untuk ditampilkan hasil dari proses IC DTW. Gambar 4.38 memperlihatkan konfigurasi pin yang dilakukan pada FPGA Spartan IIELC.

Hasil dari implementasi dapat dilihat pada Gambar 4.39 dimana terdapat perangkat komputer yang terhubung melalui komunikasi serial dengan FPGA Spartan IIELC yang telah diimplementasikan IC DTW. Perangkat komputer melalui hyperterminal berfungsi untuk menerima input yang dimasukkan pengguna untuk dikirim pada FPGA dan menampilkan output yang dikirimkan dari FPGA. Sedangkan FPGA Spartan IIELC berfungsi sebagai *pattern matching* yang menggunakan algoritma fast DTW.



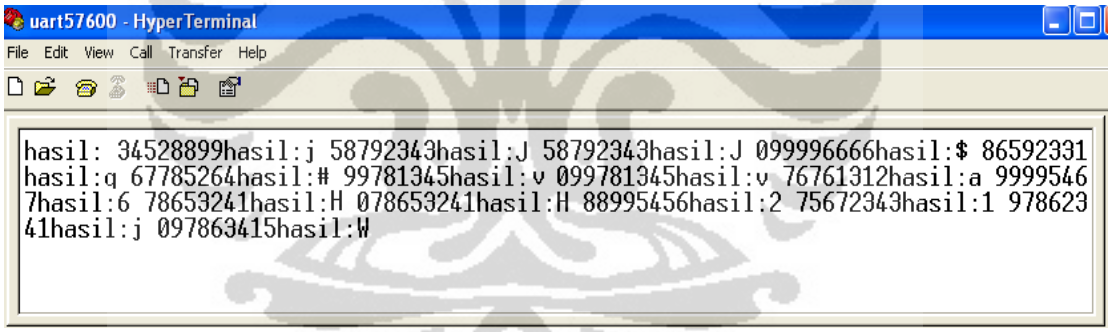
Gambar 4.39 Hasil implementasi IC DTW pada FPGA Spartan IIELC

Beberapa input yang dicoba untuk melihat apakah hasil implementasi berjalan dan mengeluarkan output yang sesuai dapat dilihat pada Tabel 4.12. Sebagai contoh untuk input vektor tsI [6, 7, 7, 8] dan vektor tsJ [5, 2, 6, 4] maka hasil dari proses IC DTW adalah 35 atau sama dengan ASCII "#". Jika dilihat dari hyperterminal pada Gambar 4.38 hasil dari implementasi, input "67785264" merupakan ASCII "6", "7", "7" dan "8". Dan ini secara otomatis akan dikonversikan menjadi integer 6, 7, 7 dan 8 oleh FPGA Spartan IIELC dan menghasilkan output integer 35 untuk dikirimkan pada hyperterminal. Karena hyperterminal menampilkan sesuatu dalam bentuk ASCII, sehingga output integer 35 ditampilkan dalam ASCII "#".

Dapat dilihat bahwa hasil implementasi IC DTW pada FPGA Spartan IIELC berhasil dilakukan dan menghasilkan output sesuai dengan fungsi *pattern matching* menggunakan algoritma DTW, sesuai dengan Tabel 4.12 dan Gambar 4.40.

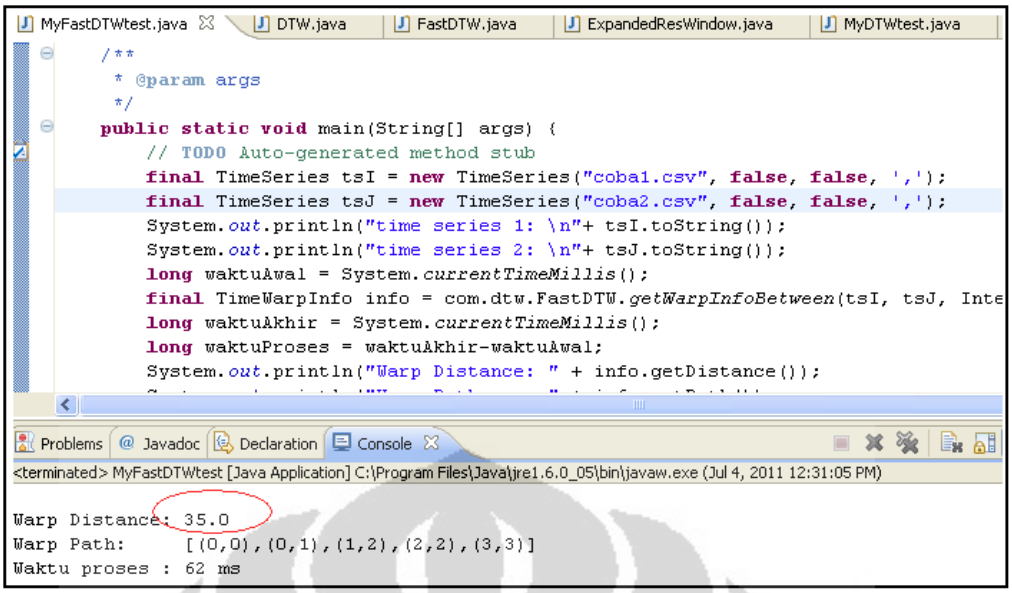
Tabel 4.12 Input dan Output untuk test vector

Vektor tsI	Vektor tsJ	Nilai MinimumCost (dalam integer)	Nilai MinimumCost (dalam ASCII)
[3, 4, 5, 2]	[8, 8, 9, 9]	106	j
[5, 8, 7, 9]	[2, 3, 4, 3]	74	J
[9, 9, 9, 9]	[6, 6, 6, 6]	36	$
[8, 6, 5, 9]	[2, 3, 3, 1]	113	q
[6, 7, 7, 8]	[5, 2, 6, 4]	35	#
[9, 9, 7, 8]	[1, 3, 4, 5]	118	v
[7, 6, 7, 6]	[1, 3, 1, 2]	97	a
[9, 9, 9, 9]	[5, 4, 6, 7]	54	6
[7, 8, 6, 5]	[3, 2, 4, 1]	72	H
[8, 8, 9, 9]	[5, 4, 5, 6]	50	2
[7, 5, 6, 7]	[2, 3, 4, 3]	49	1



Gambar 4.40 Input dan Output IC DTW pada FPGA Spartan IIELC

Untuk memastikan bahwa hasil implementasi sesuai dengan algoritma DTW yang diusulkan oleh Philip Chan dan Stan Salvador [13], maka dilakukan pengujian terhadap hasil implementasi. Proses ini dilakukan melalui perbandingan hasil perhitungan algoritma DTW yang didapatkan pada proses implementasi dengan hasil perhitungan algoritma DTW yang didapatkan pada aplikasi JAVA yang dibuat Philip Chan dan Stan Salvador [13]. Salah satu hasil perhitungan algoritma DTW yang didapatkan dengan menjalankan aplikasi JAVA tersebut, dapat dilihat pada Gambar 4.41.



Gambar 4.41 Input dan Output IC DTW pada FastDTW.java

Berdasarkan hasil pengujian yang dapat dilihat pada Gambar 4.40 dan 4.41 maka dapat disimpulkan bahwa hasil implementasi IC DTW pada FPGA Spartan – IIELC sesuai dengan hasil penelitian yang dilakukan oleh Philip Chan dan Stan Salvador. Berikut beberapa contoh hasil pengujian yang dilakukan terhadap hasil implementasi tersebut yang dapat dilihat pada Tabel 4.13.

Tabel 4.13 Hasil pengujian implementasi

Pengujian Implementasi IC DTW pada FPGA Spartan IIELC (dalam panjang vektor)		Jumlah Pengujian	Jumlah Pengujian yang sesuai dengan Penelitian Stan Salvador dan Philip Chan
Vektor tsI	Vektor tsJ		
Panjang = 4	Panjang = 4	5	5
Panjang = 8	Panjang = 8	5	5
Panjang = 10	Panjang = 10	5	5
Panjang = 15	Panjang = 15	5	5
Panjang = 20	Panjang = 20	5	5

Pada tabel tersebut dapat dikatakan bahwa dari 25 jumlah pengujian dengan 5 buah panjang vektor yang berbeda didapatkan jumlah pengujian yang sesuai dengan penelitian Stan Salvador dan Philip Chan sebanyak 25 atau dengan kata lain estimasi kesesuaian pengujian sebesar 100 %.

BAB 5

HARDWARE PROTOTYPE DAN HASIL PERCOBAAN

5.1 Hardware Prototype

Pada sub bab ini dilakukan pembahasan mengenai modul *feature matching* yang digunakan untuk pengenalan suara. Hal ini dikarenakan pada sistem pengenalan suara terdapat proses *feature extracting* sebelum dilakukan proses *feature matching* [4], maka perlu juga dilakukan pembahasan mengenai modul *feature extracting* walaupun pada penelitian ini tidak dibahas secara detail. Sehingga pembahasan *hardware prototype* dari sistem pengenalan suara menyangkut tiga hal yaitu :

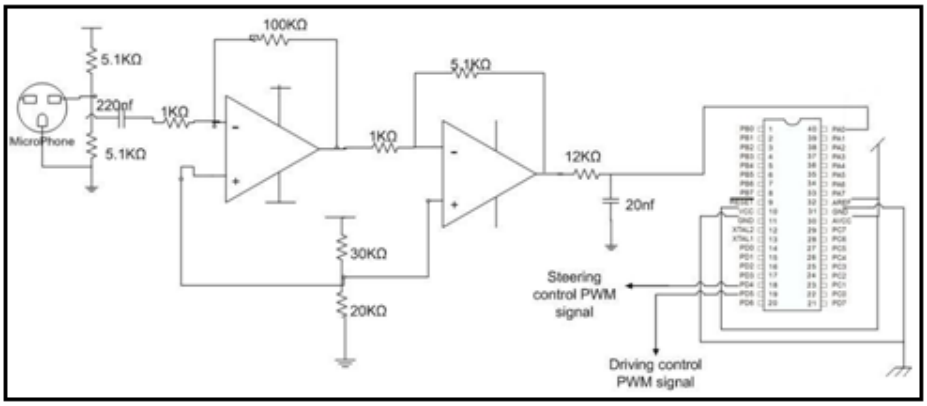
a) Modul *feature extracting*
b) Modul *feature matching*
c) Integrasi kedua modul

Pembahasan dari ketiga hal tersebut diuraikan dibawah ini sebagai berikut

5.1.1 Modul untuk *feature extraction*

Pada sub bab ini dilakukan pembahasan mengenai pembuatan *hardware prototype speech recognition* yang terdiri dari modul untuk *feature extraction* dan modul untuk *feature matching*. Modul untuk *feature extraction* merupakan modul yang digunakan untuk melakukan proses ekstraksi dari suara yang diinputkan sehingga didapatkan vektor data digital. Modul ini terdiri dari rangkaian input mikrofon, amplifier dan mikrokontroller ATMEGA8535 yang digunakan sebagai ADC. Rangkaian skematik dari modul untuk *feature extraction* dapat dilihat pada Gambar 5.1, dan secara fisik dari modul tersebut dapat dilihat pada Gambar 5.2.

Pada Gambar 5.1, sinyal yang keluar dari mikrofon perlu diperkuat. Komponen yang digunakan untuk memperkuat sinyal tersebut adalah IC Op Amp jenis LM358. Komponen tersebut memiliki laju perubahan tegangan yang lebih baik dan itu memberikan respon yang lebih baik untuk sinyal input, sehingga tepat untuk merancang rangkaian amplifikasi.



Gambar 5.1 Rangkaian skematik modul *feature extraction* [27]

Mikrofon yang digunakan memiliki rangkaian amplifier didalamnya tetapi setelah memeriksa melalui osiloskop, kekuatan sinyal cukup rendah. Sehingga dibuatlah penguat sinyal untuk meningkatkan *peak to peak* sinyal audio. Op Amp LM358 tersebut merupakan *negative feed back loops* dan pembalik yang memiliki nilai penguatan sebesar

$$\frac{-R_f}{R_{in}} \quad \text{..... (5.1)}$$

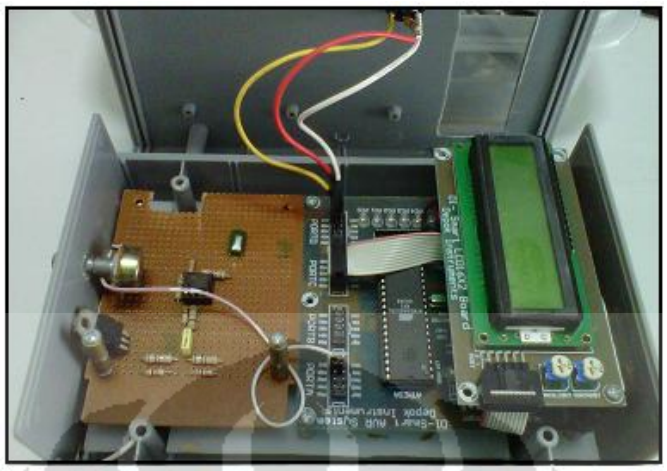
Sehingga total penguatan dari rangkaian tersebut sesuai rumus 5.1 adalah

$$\frac{-100K}{1K} * \frac{-5,1K}{1K} = 500 \quad \text{..... (5.2)}$$

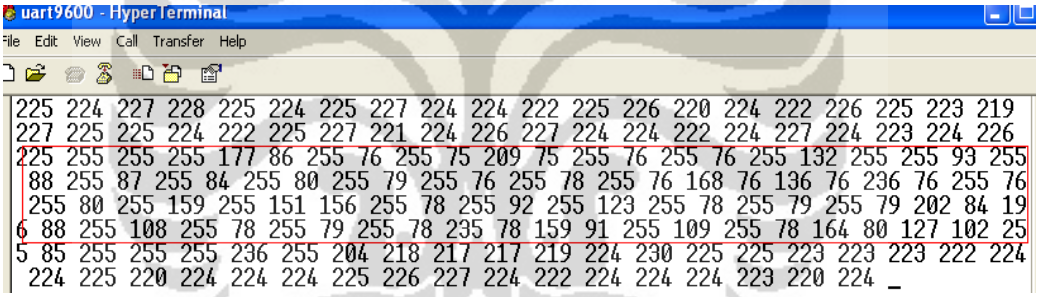
Dengan bias dc 2V yang diberikan oleh rangkaian pembagi resistor di terminal positif dari op amp [27]. Untuk fisik dari modul *feature extraction* dapat dilihat pada Gambar 5.2 sesuai dengan rangkaian skematik pada Gambar 5.1. Adapun penambahan hanya terdapat pada tombol proses yang berfungsi untuk perekaman input suara dan LCD yang berfungsi untuk menampilkan hasil proses *feature extraction* dan proses *feature matching*.

Beberapa hasil percobaan dari modul *feature extraction* dapat dilihat pada Gambar 5.3, 5.4 dan 5.5 yang merupakan hasil dari ekstraksi input pengucapan suara abjad ‘a’, abjad ‘b’ dan abjad ‘c’. Hasil dari modul *feature extraction* ditampilkan pada *hyperterminal* dikarenakan LCD pada modul tersebut hanya dapat menampung 32 karakter sedangkan percobaan dilakukan untuk proses perekaman dengan waktu

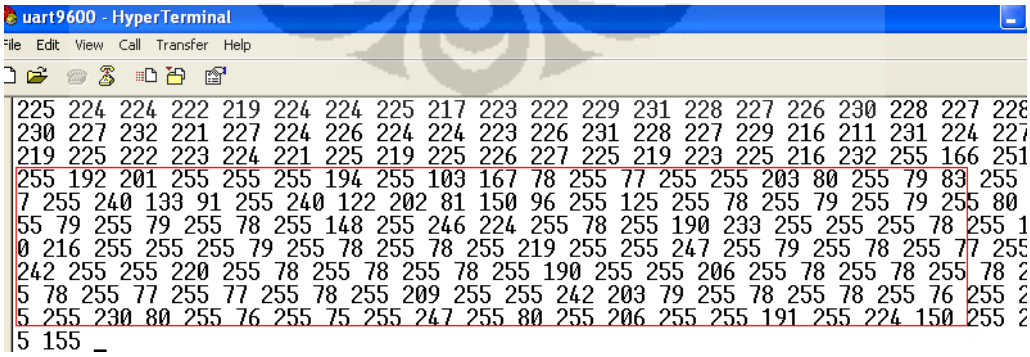
singkat dan panjang (1 detik s/d 30 detik). Hasil percobaan lebih lengkap dapat dilihat pada lampiran 12, lampiran 13 dan lampiran 14.



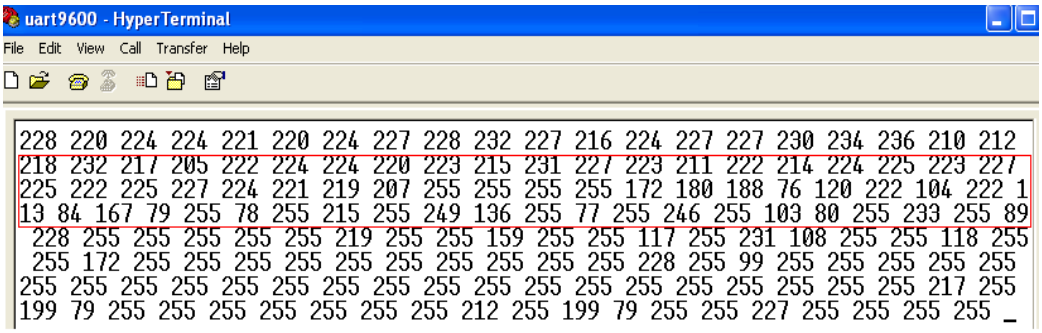
Gambar 5.2 Fisik dari modul *feature extraction*



Gambar 5.3 Hasil *feature extraction* suara 'a'



Gambar 5.4 Hasil *feature extraction* suara 'b'



Gambar 5.5 Hasil *feature extraction* suara 'c'

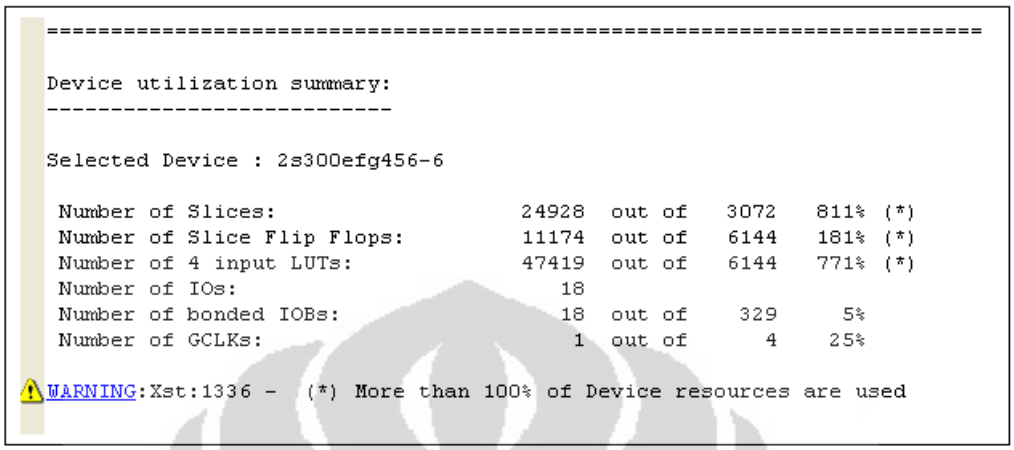
5.1.2 Modul untuk *feature matching*

Modul untuk *feature matching* merupakan modul yang digunakan untuk melakukan proses pencocokan pola menggunakan algoritma DTW dari dua buah vektor data digital. Modul ini merupakan hasil implementasi sesuai pada sub bab 4.1. Secara fisik Modul *feature matching* dapat dilihat pada Gambar 5.6.

Gambar 5.6 Modul *feature matching* berbasis FPGA Spartan IIELC

Input dari modul tersebut berasal dari modul *feature extraction* dimana hasilnya terlihat pada Gambar 5.3, Gambar 5.4 dan Gambar 5.5. Pada Gambar tersebut proses perekaman terjadi selama 10 s/d 30 detik dan menghasilkan vektor data sepanjang 40 s/d 100 data. Pada FPGA Spartan IIELC terdapat keterbatasan jumlah slices, LUT dan IOB maka untuk implementasi dengan panjang vektor 50 s/d 100 data tidak bisa dilakukan karena melebihi 3072 untuk slices dan 6144 untuk LUT . Salah satu contoh

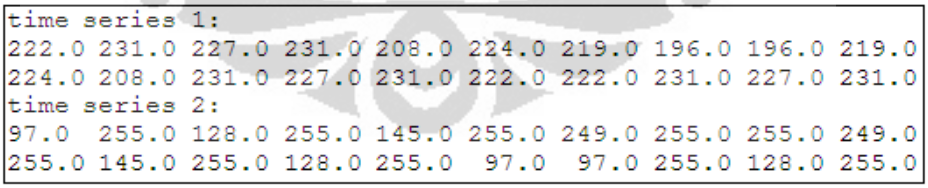
hasil percobaan implementasi untuk jumlah vektor data 50 buah terlihat pada Gambar 5.7. Dimana penggunaan slices sebanyak 24928 dari 3072, slices flip flop sebanyak 11174 dari 6144 dan LUT sebanyak 47419 dari 6144.



```
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2s300efg456-6

 Number of Slices:                    24928  out of   3072   811% (*)
 Number of Slice Flip Flops:          11174  out of   6144   181% (*)
 Number of 4 input LUTs:              47419  out of   6144   771% (*)
 Number of IOs:                          18
 Number of bonded IOBs:                  18  out of    329     5%
 Number of GCLKs:                         1  out of      4    25%

WARNING:Xst:1336 -  (*) More than 100% of Device resources are used
```

Gambar 5.7 Keterbatasan FPGA Spartan IIELC untuk 50 Vektor Data

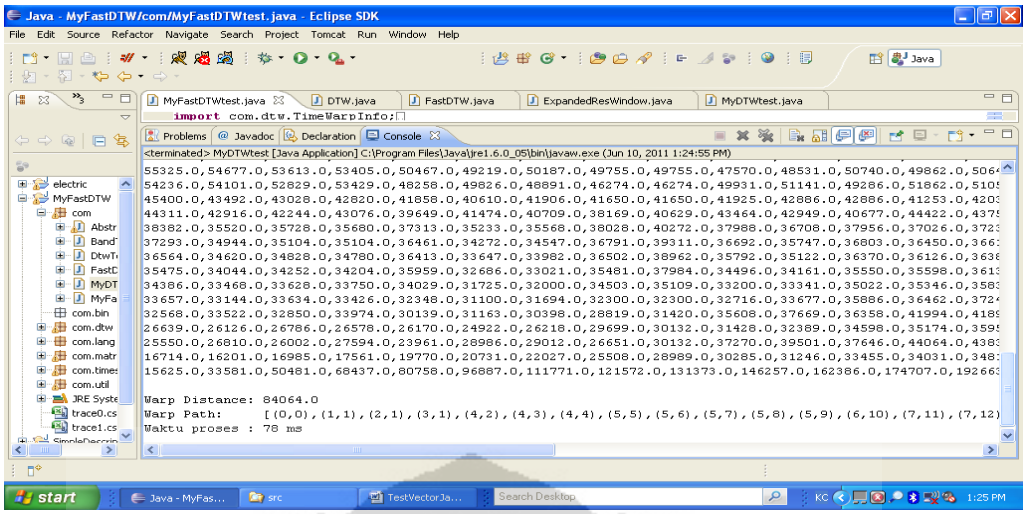
Berikut salah satu contoh percobaan yang dilakukan untuk input vector data sebanyak 20 buah yang akan diproses pada Xilinx ISE dimana hasil yang didapat akan dibandingkan dengan hasil simulasi aplikasi Java yang dibuat oleh Stan Salvador dan Philip Chan[13]. Sedangkan input merupakan vektor yang dihasilkan dari modul *feature extraction* sesuai pembahasan sub bab 5.1.1. Gambar 5.8 memperlihatkan input vektor data tersebut dan Gambar 5.9 merupakan hasil simulasi aplikasi Java.



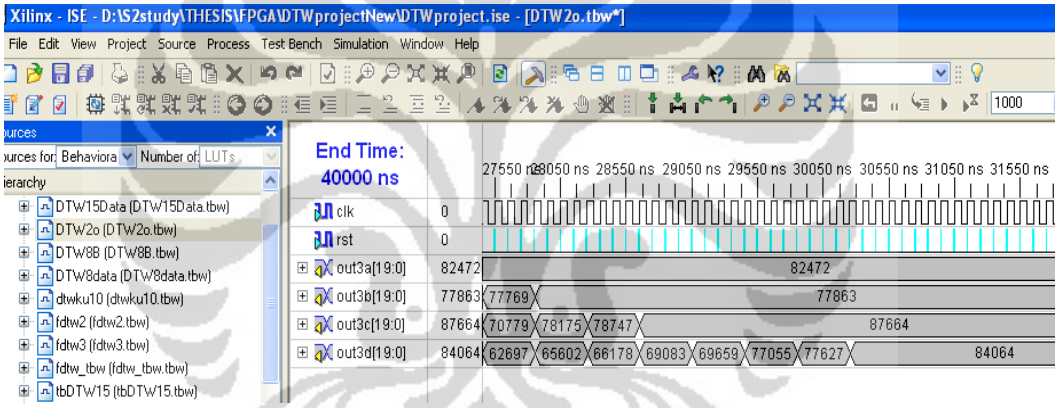
```
time series 1:
222.0 231.0 227.0 231.0 208.0 224.0 219.0 196.0 196.0 219.0
224.0 208.0 231.0 227.0 231.0 222.0 222.0 231.0 227.0 231.0
time series 2:
97.0  255.0 128.0 255.0 145.0 255.0 249.0 255.0 255.0 249.0
255.0 145.0 255.0 128.0 255.0  97.0  97.0 255.0 128.0 255.0
```

Gambar 5.8 Contoh Input Vektor Data

Hasil yang didapat menggunakan aplikasi Java sesuai dengan input pada Gambar 5.8 adalah 84064 sesuai dengan hasil yang didapatkan menggunakan simulasi Xilinx ISE. Hasil tersebut dapat dilihat pada Gambar 5.10 yang merupakan hasil percobaan pada simulasi software, sedangkan untuk hasil nyata dapat dilihat pada hasil percobaan sesuai sub bab 5.2.1.



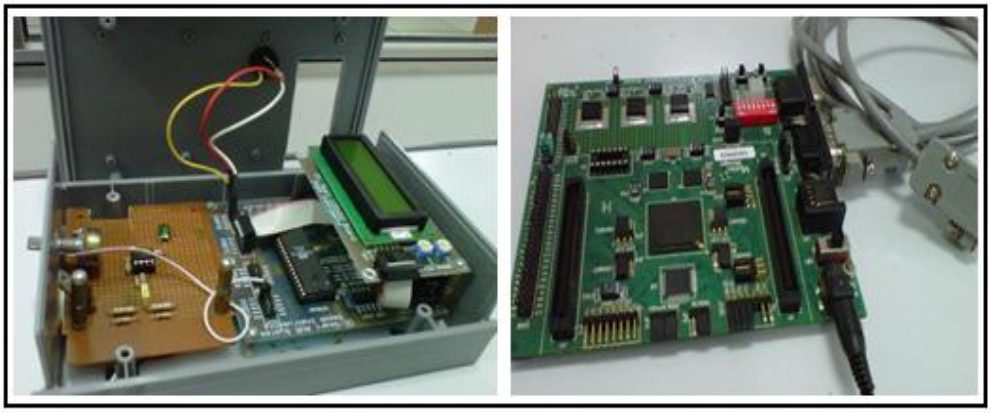
Gambar 5.9 Contoh Hasil Simulasi Aplikasi Java



Gambar 5.10 Contoh Hasil Simulasi Xilinx ISE

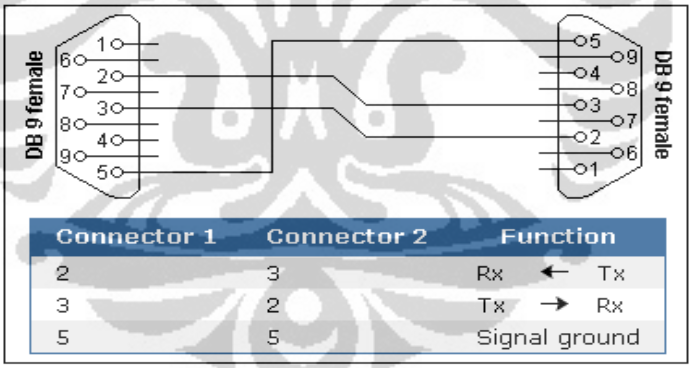
5.1.3 Integrasi modul untuk *speech recognition*

Proses integrasi modul untuk *speech recognition* dilakukan dengan membuat komunikasi serial RS232 dengan mode null modem antara modul *feature extraction* dan modul *feature matching*. Mode null modem dibuat karena port serial yang terdapat pada kedua modul tersebut merupakan port *female*. Gambar 5.11 memperlihatkan modul yang akan diintegrasikan untuk membentuk *hardware prototype speech recognition*.



Gambar 5.11 Modul yang akan diintegrasikan

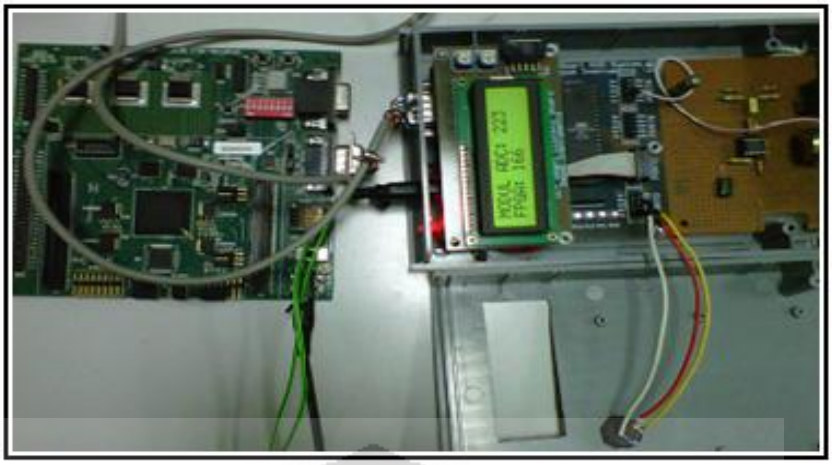
Sedangkan Gambar 5.12 memperlihatkan konfigurasi pin untuk mode null modem pada komunikasi serial RS232. Pada konfigurasi tersebut pin 3 untuk pengiriman data pada konektor 1 dihubungkan dengan pin 2 untuk penerimaan data pada konektor 2 dan sebaliknya. Untuk pin 5 pada konektor 1 yang meupakan sinyal *ground* dihubungkan dengan pin 5 pada konektor 2 yang meupakan sinyal *ground* juga.



Connector 1	Connector 2	Function
2	3	Rx ← Tx
3	2	Tx → Rx
5	5	Signal ground

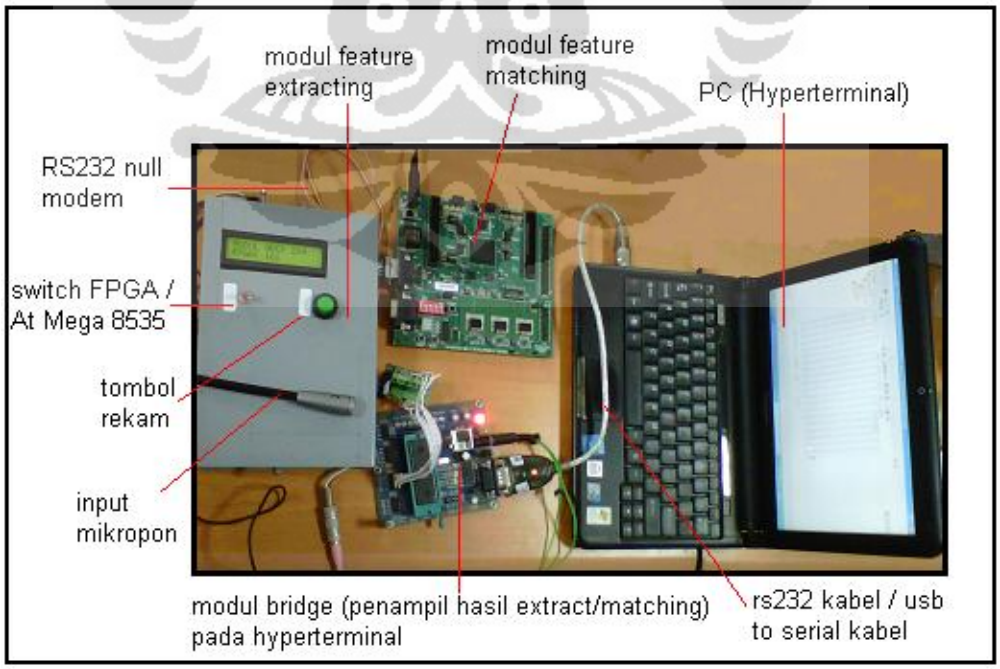
Gambar 5.12 Konfigurasi Pin untuk Mode Null Modem

Bentuk fisik dari *hardware prototype speech recognition* dapat dilihat pada Gambar 5.13 dimana untuk sementara setiap modul menggunakan *power supply* secara terpisah dan kedua modul tersebut hanya diintegrasikan melalui koneksi kabel RS232 dengan mode null modem. Hasil percobaan dari *hardware prototype speech recognition* dapat dilihat pada sub bab 5.2.



Gambar 5.13 *hardware prototype speech recognition*

Sebelum melakukan percobaan perlu dilakukan pembuatan modul untuk menampilkan hasil dari *feature extraction* dan *feature matching* pada layar hypertrminal. Hal ini dilakukan mengingat layar LCD yang terdapat pada modul *feature extraction* tidak cukup untuk menampilkan hasil secara detail. Modul yang dibuat dihubungkan secara paralel pada port B dari modul tersebut dengan port B pada modul *feature extraction*. Gambar 5.14 merupakan konfigurasi alat untuk percobaan pengenalan suara dan secara detail dapat dilihat pada lampiran 17.



Gambar 5.14 Konfigurasi alat untuk percobaan pengenalan suara

5.2 HASIL PERCOBAAN

Pada sub bab ini akan dibahas mengenai hasil percobaan yang didapatkan dari *hardware prototype speech recognition* dengan input suara abjad dan estimasi kesalahan yang dihasilkan dari percobaan yang dilakukan tersebut.

5.2.1 Percobaan untuk input suara abjad ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’

Percobaan dilakukan dengan menyesuaikan konfigurasi alat dalam hal ini modul *feature matching* yang hanya dapat diinput satu vektor data yang dijadikan *feature* suara dikarenakan adanya kerusakan dipswitch untuk pemilihan *feature* suara tersebut. Adapun metode untuk membandingkan kelima *feature* suara dalam satu proses memakan *source* gate yang banyak sehingga tidak mungkin dilakukan, dimana permasalahan ini dapat terlihat pada Gambar 5.7.

Sebelum dilakukan percobaan, terlebih dahulu dilakukan pengambilan sampel untuk menentukan batas penerimaan berdasarkan nilai minimum *distance* yang dihasilkan. Untuk setiap satu *feature* suara diambil data input suara sebanyak 150 data dimana suara ‘a’ sebanyak 30 data, suara ‘b’ sebanyak 30 data, suara ‘c’ sebanyak 30 data, suara ‘d’ sebanyak 30 data dan suara ‘e’ sebanyak 30 data. Dari hasil pengambilan data tersebut batas penerimaan suatu input suara dianggap memiliki kemiripan terhadap *feature* suara yaitu nilai minimum *distance* 0 s/d 100.

Adapun percobaan dikelompokkan berdasarkan *feature* suara yang terdapat pada modul *feature matching* bukan berdasarkan input suara. Untuk setiap percobaan yang menggunakan satu *feature* suara akan dilakukan input suara sebanyak 100 percobaan dimana suara ‘a’ sebanyak 20 percobaan, suara ‘b’ sebanyak 20 percobaan, suara ‘c’ sebanyak 20 percobaan, suara ‘d’ sebanyak 20 percobaan dan suara ‘e’ sebanyak 20 percobaan. Hasil percobaan selengkapnya dapat dilihat pada lampiran 18 dan berikut beberapa hasil dari percobaan tersebut

a. Percobaan input suara ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’ dibandingkan *feature* suara 1

Tabel 5.1 hasil perbandingan *feature* suara 1

Perbandingan	Nilai minimum distance				
Suara ‘a’ vs *feature* 1	133	75	129	173	1
Suara ‘b’ vs *feature* 1	229	197	225	165	229
Suara ‘c’ vs *feature* 1	253	253	255	255	37

Suara ‘d’ vs *feature* 1	193	165	231	163	254
Suara ‘e’ vs *feature* 1	248	165	202	253	253

Dari Tabel 5.1 dan lampiran 18 didapatkan :

- Perbandingan suara ‘a’ terhadap *feature* 1 terdapat 7 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘b’ terhadap *feature* 1 terdapat 1 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘c’ terhadap *feature* 1 terdapat 2 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘d’ terhadap *feature* 1 terdapat 1 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘e’ terhadap *feature* 1 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.

b. Percobaan input suara ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’ dibandingkan *feature* suara 2

Tabel 5.2 hasil perbandingan *feature* suara 2

Perbandingan	Nilai minimum distance				
Suara ‘a’ vs *feature* 2	225	41	194	225	8
Suara ‘b’ vs *feature* 2	161	165	253	255	241
Suara ‘c’ vs *feature* 2	253	255	161	225	117
Suara ‘d’ vs *feature* 2	161	240	249	242	239
Suara ‘e’ vs *feature* 2	249	253	254	229	255

Dari Tabel 5.2 dan lampiran 18 didapatkan :

- Perbandingan suara ‘a’ terhadap *feature* 2 terdapat 3 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘b’ terhadap *feature* 2 terdapat 3 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘c’ terhadap *feature* 2 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘d’ terhadap *feature* 2 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.

- Perbandingan suara ‘e’ terhadap *feature* 2 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.

c. Percobaan input suara ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’ dibandingkan *feature* suara 3

Tabel 5.3 hasil perbandingan *feature* suara 3

Perbandingan	Nilai minimum distance				
Suara ‘a’ vs *feature* 3	231	255	244	229	149
Suara ‘b’ vs *feature* 3	235	169	255	229	149
Suara ‘c’ vs *feature* 3	227	241	197	253	167
Suara ‘d’ vs *feature* 3	253	255	249	215	81
Suara ‘e’ vs *feature* 3	245	253	255	101	253

Dari Tabel 5.3 dan lampiran 18 didapatkan :

- Perbandingan suara ‘a’ terhadap *feature* 3 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘b’ terhadap *feature* 3 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘c’ terhadap *feature* 3 terdapat 4 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘d’ terhadap *feature* 3 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘c’ terhadap *feature* 3 terdapat 1 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.

d. Percobaan input suara ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’ dibandingkan *feature* suara 4

Tabel 5.4 hasil perbandingan *feature* suara 4

Perbandingan	Nilai minimum distance				
Suara ‘a’ vs *feature* 4	255	243	251	253	241
Suara ‘b’ vs *feature* 4	226	255	254	253	255
Suara ‘c’ vs *feature* 4	210	252	255	242	198
Suara ‘d’ vs *feature* 4	249	98	227	271	12
Suara ‘e’ vs *feature* 4	251	227	255	143	227

Dari Tabel 5.4 dan lampiran 18 didapatkan :

- Perbandingan suara ‘a’ terhadap *feature* 4 terdapat 1 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘b’ terhadap *feature* 4 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.
- .Perbandingan suara ‘c’ terhadap *feature* 4 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘d’ terhadap *feature* 4 terdapat 6 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘e’ terhadap *feature* 4 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.

e. Percobaan input suara ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’ dibandingkan *feature* suara 5

Tabel 5.5 hasil perbandingan *feature* suara 5

Perbandingan	Nilai minimum distance				
Suara ‘a’ vs *feature* 5	203	237	237	241	255
Suara ‘b’ vs *feature* 5	255	225	173	255	241
Suara ‘c’ vs *feature* 5	203	196	173	255	243
Suara ‘d’ vs *feature* 5	233	248	255	243	255
Suara ‘e’ vs *feature* 5	3	195	107	237	237

Dari Tabel 5.5 dan lampiran 18 didapatkan :

- Perbandingan suara ‘a’ terhadap *feature* 5 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘b’ terhadap *feature* 5 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘c’ terhadap *feature* 5 terdapat 1 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘d’ terhadap *feature* 5 tidak terdapat nilai minimum *distance* yang memenuhi batas nilai 0 s/d 100.
- Perbandingan suara ‘e’ terhadap *feature* 5 terdapat 7 nilai minimum *distance* dari 20 percobaan yang memenuhi batas nilai 0 s/d 100.

5.2.2 Estimasi keberhasilan pada percobaan

Pembahasan untuk estimasi keberhasilan pada percobaan terhadap *hardware prototype speech recognition* dilakukan dengan menghitung persentase keberhasilan percobaan menggunakan perhitungan sesuai rumus 5.3.

$$\% \text{ keberhasilan} = \frac{\text{jumlah percobaan berhasil}}{\text{jumlah seluruh percobaan}} \times 100\% \quad \text{..... (5.3)}$$

Jika diasumsikan bahwa, nilai minimum *distance* yang dihasilkan dari perbandingan input suara dengan *feature* suara terletak pada batas nilai 0 s/d 100 merupakan input suara yang dikenali sebagai *feature* suara. Dan *feature* 1 merupakan pola suara 'a', *feature* 2 merupakan pola suara 'b', *feature* 3 merupakan pola suara 'c', *feature* 4 merupakan pola suara 'd' dan *feature* 5 merupakan pola suara 'e' maka estimasi keberhasilan dapat dilihat pada tabel 5.6.

Tabel 5.6 Estimasi keberhasilan percobaan

Percobaan	Berhasil	Total Percobaan	% Keberhasilan
Input suara 'a' dikenali *feature* 1	7	20	35%
Input suara 'b' dikenali *feature* 2	3	20	15%
Input suara 'c' dikenali *feature* 3	4	20	20%
Input suara 'd' dikenali *feature* 4	6	20	30%
Input suara 'e' dikenali *feature* 5	7	20	35%

Analisis sementara bahwa estimasi keberhasilan yang dihasilkan sangat kecil (< 50%) disebabkan pembuatan filter pada modul *feature extracting* tidak berhasil dilakukan dan belum diterapkannya algoritma untuk ekstraksi suara seperti LPC, MFCC atau STFT. Sedangkan untuk modul *feature matching* sudah sesuai dengan hasil yang diharapkan, hal ini diperkuat dengan kesesuaian hasil pengujiaan dengan penelitian yang dilakukan oleh Philip Chan dan Stan Salvador mencapai 100%.

BAB 6

KESIMPULAN DAN SARAN

6.1 Kesimpulan

Dari pembahasan yang dilakukan, didapatkan beberapa kesimpulan yaitu :

1. Dalam hal perancangan layout CMOS IC *pattern matching* menggunakan algoritma DTW didapatkan *delay* waktu proses sebesar 41,397 ns untuk input vektor data sebanyak 8 buah. Hal ini jauh lebih cepat dibandingkan dengan percobaan software yang dilakukan Stan Salvador dan Philip Chan sebesar 219 ms dengan jumlah input vektor data yang sama.
2. Dalam hal implementasi layout CMOS IC FastDTW pada sebuah *device* FPGA Spartan IIELC berhasil dilakukan sesuai dengan tes vektor dan *delay* waktu proses yang dibutuhkan sebesar 31,263 ns untuk input vektor data sebanyak 8 buah. Sehingga dapat dikatakan pembuatan modul *feature matching* berbasis FPGA Spartan IIELC berhasil dilakukan.
3. Hasil percobaan untuk pengenalan suara menggunakan modul *feature matching* didapatkan estimasi keberhasilan sebesar 35% untuk input suara ‘a’ yang dikenali sebagai *feature* 1, 15% untuk input suara ‘b’ yang dikenali sebagai *feature* 2, 20% untuk input suara ‘c’ yang dikenali sebagai *feature* 3, 30% untuk input suara ‘d’ yang dikenali sebagai *feature* 4 dan 35% untuk input suara ‘e’ yang dikenali sebagai *feature* 5.

6.2 Saran

Saran dari pembahasan yang dilakukan yaitu :

1. Perlu dilakukan penelitian lebih mendalam mengenai metode dalam proses *feature extracting* dan pembuatan modul *feature extracting* mengingat hal ini sangat mempengaruhi estimasi keberhasilan dalam proses pengenalan suara. .

DAFTAR PUSTAKA

1. Arun A. Ross, Karthik Nandakumar, Anil K. Jain, 2006. *Handbook of Multibiometrics*. Springer.
2. John D. Woodward, Jr., Nicholas M. Orlans, Peter T. Higgins, *Biometrics*, McGraw-Hill, 2003.
3. B. Planerrer, 2005. An Introduction of Speech Recognition, Munich, Germany.
4. www.cesg.gov.uk/site/ast/biometrics/media/BEM_10.pdf
5. Lawrence Rabiner and Biing-Hwang Juang, 1993. Fundamental of Speech Recognition", Prentice-Hall, Englewood Cliffs, N.J.
6. Corry S. Meyers, 1980. *A Comparative Study Of Several Dynamic Time Warping Algorithm For Speech Recognition* . Submitted for Bachellor of Science and Master of Science, Masachusset Institute of Technology.
7. Sakoe, H. & S. Chiba. 1978. *Dynamic programming algorithm optimization for spoken word recognition.* IEEE, Trans. Acoustics, Speech, and Signal Proc., Vol. ASSP-26.
8. Neil Weste, David J. Burr and Bryan D. Ackland, 1983. *Dynamic Time Warp Pattern Matching Using an Integrated Multiprocessing Array*. IEEE, Trans. On Computer, Vol. C.32, No.8.
9. Robert Kavaler, M. Lowey, HY Murveit and Robert W. Broddersen, 1987. *A Dynamic Time Warp Integrated Circuit for 1000 Word Speech Recognition System*. IEEE, Journal of Solid State, Vol. SC.22, No.1.
10. W. Drews, R. Laroia, J.Pandel, A. Schumaccer and A. Stozzle, 1989. *CMOS Processor for Template Based Speech Recognition System*. IEEE proceedings, Vol. 136,Pt. 1, No.2.
11. Sung-Nam King, In-Chul Hwang, Young Woo Kim and Soo-Won Kim, 1996. *A VLSI Chip for Isolated Speech Recognition System*. IEEE, Trans. On Consumer Electronic, Vol. 42, No.3.

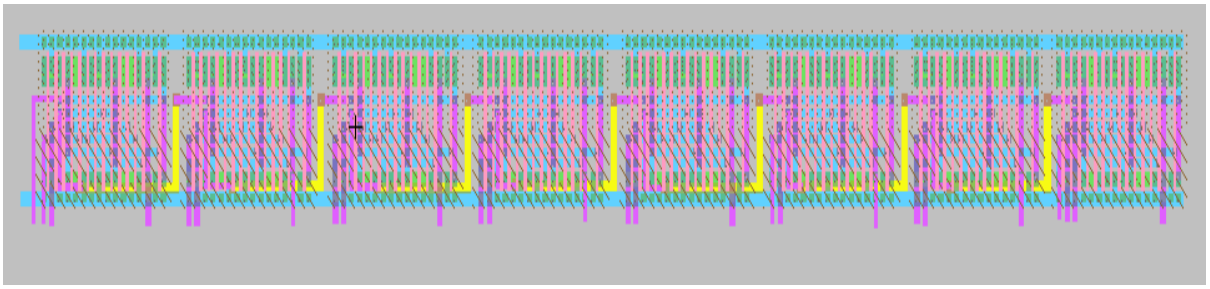
12. Lorenzo Cali, Francesco Letorta and Michele Borgatti, 2002. *A Single Chip Speech Recognition System with Embedded Flash Mmeory and Configurrable Data Path*, ESSCIRC 2002.
13. Stan Salvador and Philip Chan, 2007. *FastDTW : Toward Accurate Dynamic Time Warping in Linear Time and Space*, Project in Dept. of Computer Sciences Florida Institute of Technology, Melbourne
14. Jhing-Fa Wang, Jia-Ching Wang, Han-Chian Chen, Tai-Lung Chen, Chin-Chan Chang and Ming Chi Shih, 2002. C*hip Design of Portable Speech Memopad Suitable for Persons With Visual Disabilities*, IEEE TRANSACTION ON SPEECH AND AUDIO PROCESSING, VOL. 10, NO. 8, November 2002.
15. "Xilinx Spartan-II Family Architecture", www.xilinx.com, 2004
16. Anantha P. Chandrakasan, and Donald E. Troxel, "3D FPGA Design and CAD Flow, chapter 2", MTL/RLE Groups, 2006
17. Spartan-IIE LC Development Board User's Guide version 1.2. (MEMEC Design, 2004)
18. "*What are* FPGAs?". http://www.fpga4fun.com/fpgainfo1, diakses 10 Pebruari 2011.
19. Xiao, H. (2001). *Introduction to Semiconductor Manufacturing Technology*. New Jersey: Prentice Hall.
20. N. Kamal, (2005), *Fabrikasi & Pencirian Elektrik Peranti PMOS 0.24 μm*, BSc. Thesis, Malaysia: Universiti Kebangsaan Malaysia.
21. Chiah, S.B, Zhou, X. et al, (2001). Semi-empirical Approach to ModelingReverse Short-Channel Effect in Submicron MOSFET's, Modeling and Simulation Microsystem, pp 486-489.
22. Yuhua Chen, Mansun Chan, Kelvin Hui, Min-chie Jeng, X. et all, (1996). BSIM3v3 Manual (Final Version), Department of Electrical Engineering and Computer Sciences, University of California, Berkeley.
23. http://www.mosis.com/about/whatis.html, diakses pada 15 Maret 2011.

24. http://www.mosis.com/Technical/Testdata/tsmc-018-prm.html, diakses pada 15 Maret 2011.
25. LTSpice Tutorial, Linear Technology Corporation, 2001 – 2009.
26. Steven M. Rubin, Using the ElectricTM VLSI Design System Version 8.09, Static Free Software and Sun Microsystem, Published by R.L. Ranch Press, 2009.
27. Andre Harison and Chirag Shah, http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2006/avh8_css34/avh8_css34/index.html#intro, diakses pada 27 Januari 2011.

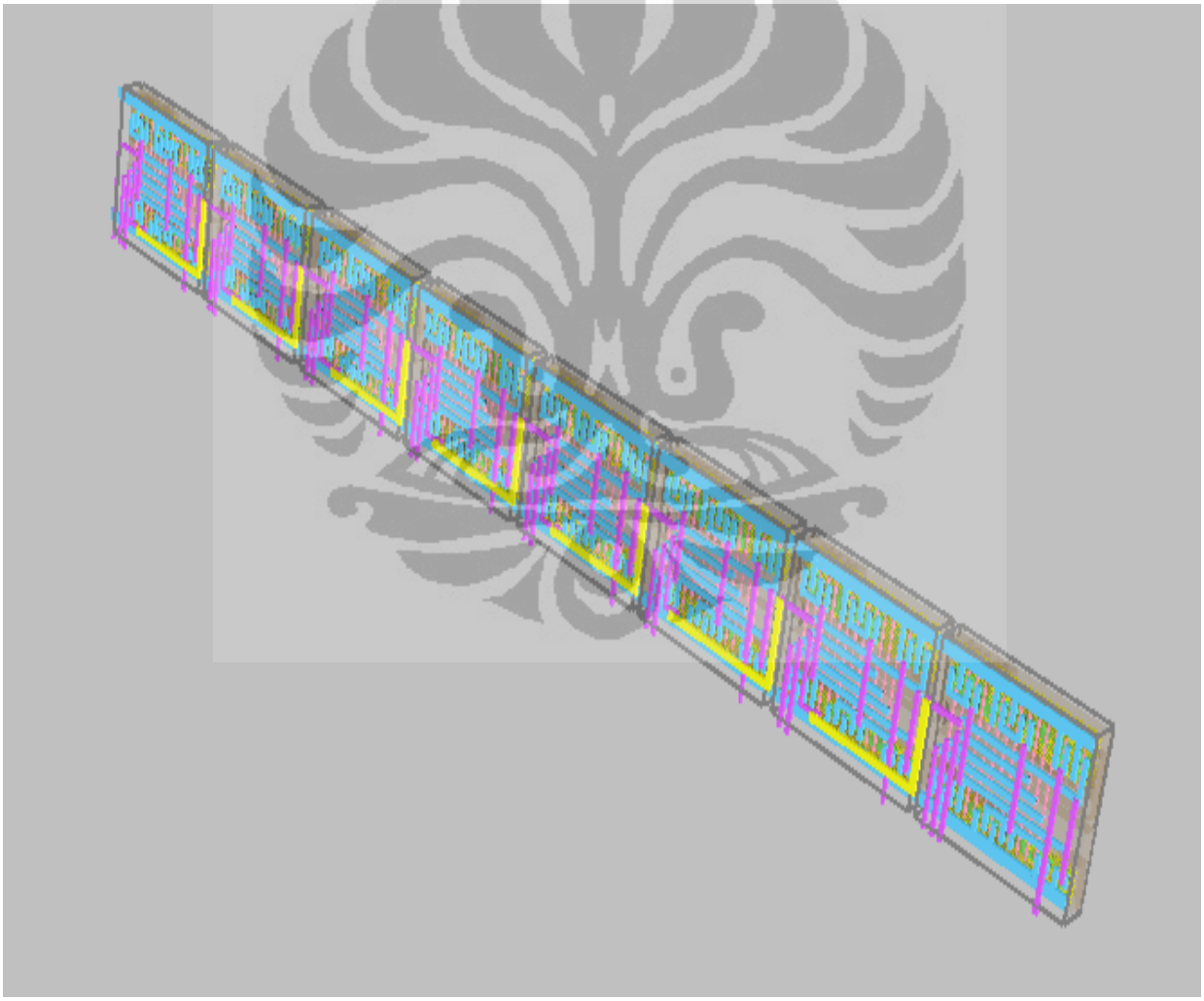
DAFTAR ACUAN

1. Neil Weste and K. Esraghian, June, 1988. *Principle of CMOS VLSI Design : A System Perspective*. Adisson Wesley Co Publ.
2. Stan Salvador and Philip Chan, 2007. *FastDTW : Toward Accurate Dynamic Time Warping in Linear Time and Space*, Project in Dept. of Computer Sciences Florid a Institute of Technology, Melbourne
3. Jhing-Fa Wang, Jia-Ching Wang, Han-Chian Chen, Tai-Lung Chen, Chin-Chan Chang and Ming Chi Shih, 2002. C*hip Design of Portable Speech Memopad Suitable for Persons With Visual Disabilities*, IEEE TRANSACTION ON SPEECH AND AUDIO PROCESSING, VOL. 10, NO. 8, November 2002.

LAMPIRAN 1
(Layout CMOS Full Adder 8 bit 2D dan 3D)

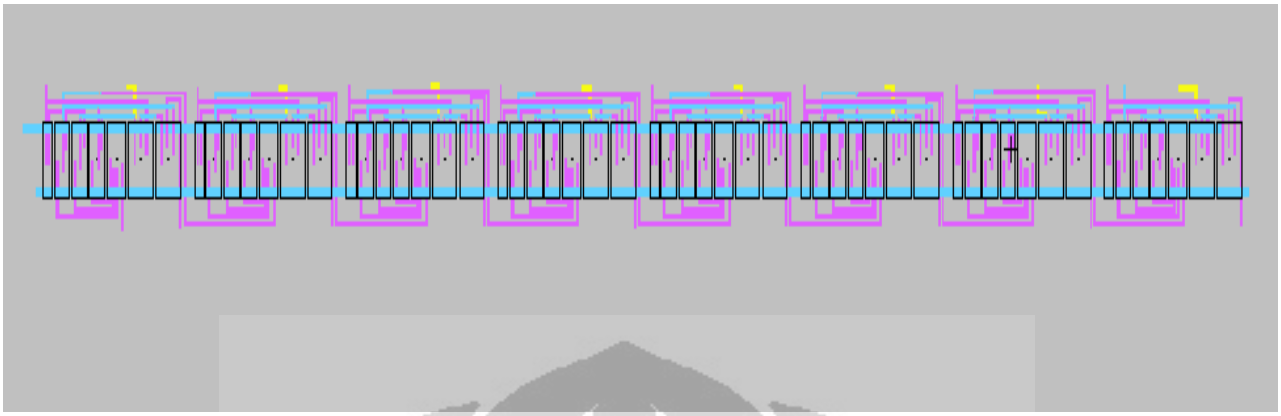


Gambar 1. Layout CMOS Full Adder 8 bit 2D

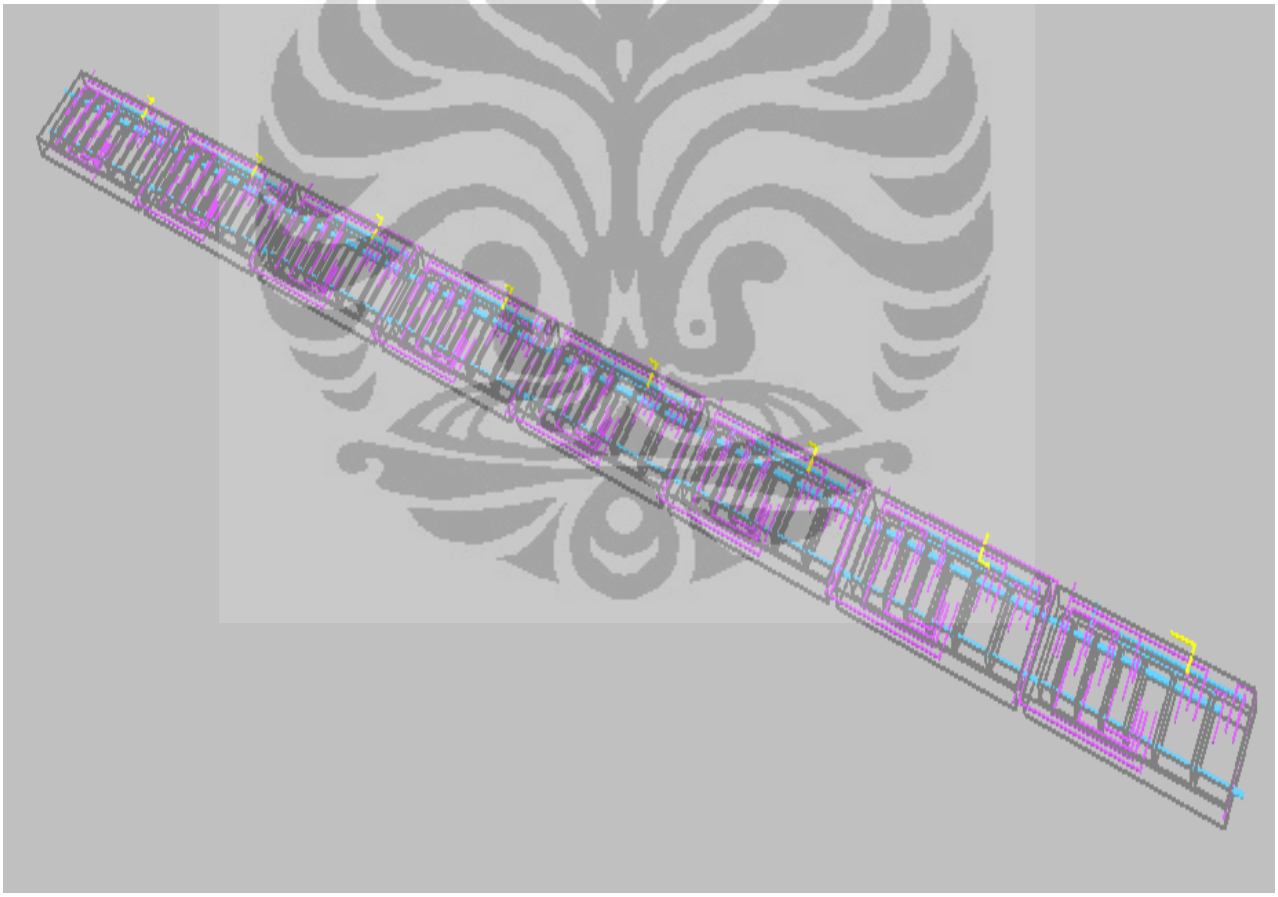


Gambar 2. Layout CMOS Full Adder 8 bit 3D

LAMPIRAN 2
(Layout CMOS Full Subtractor 8 bit 2D dan 3D)

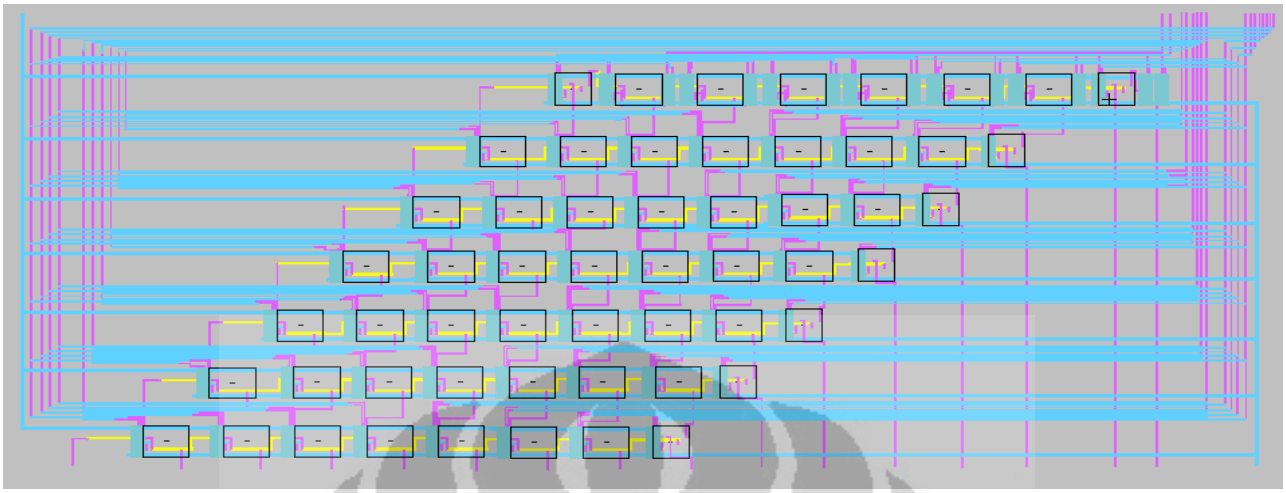


Gambar 1. Layout CMOS Full Subtractor 8 bit 2D

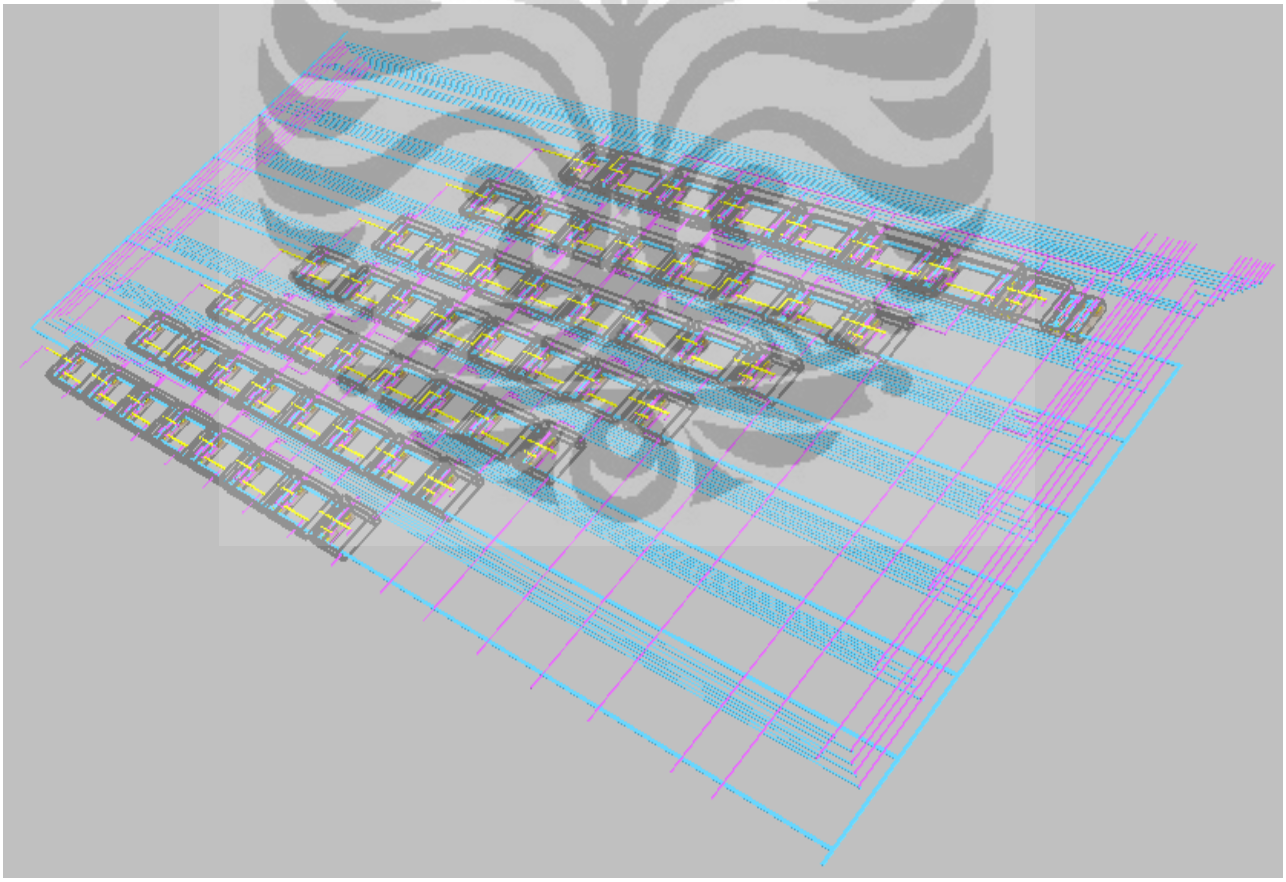


Gambar 2. Layout CMOS Full Subtractor 8 bit 3D

LAMPIRAN 3
(Layout CMOS Full Multiplicator 8 bit 2D dan 3D)

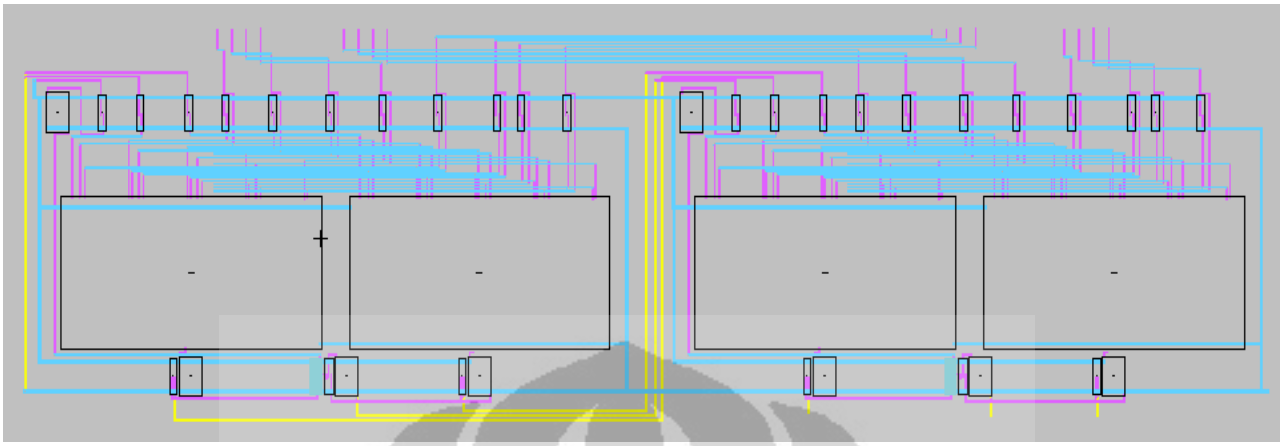


Gambar 1. Layout CMOS Full Multiplicator 8 bit 2D

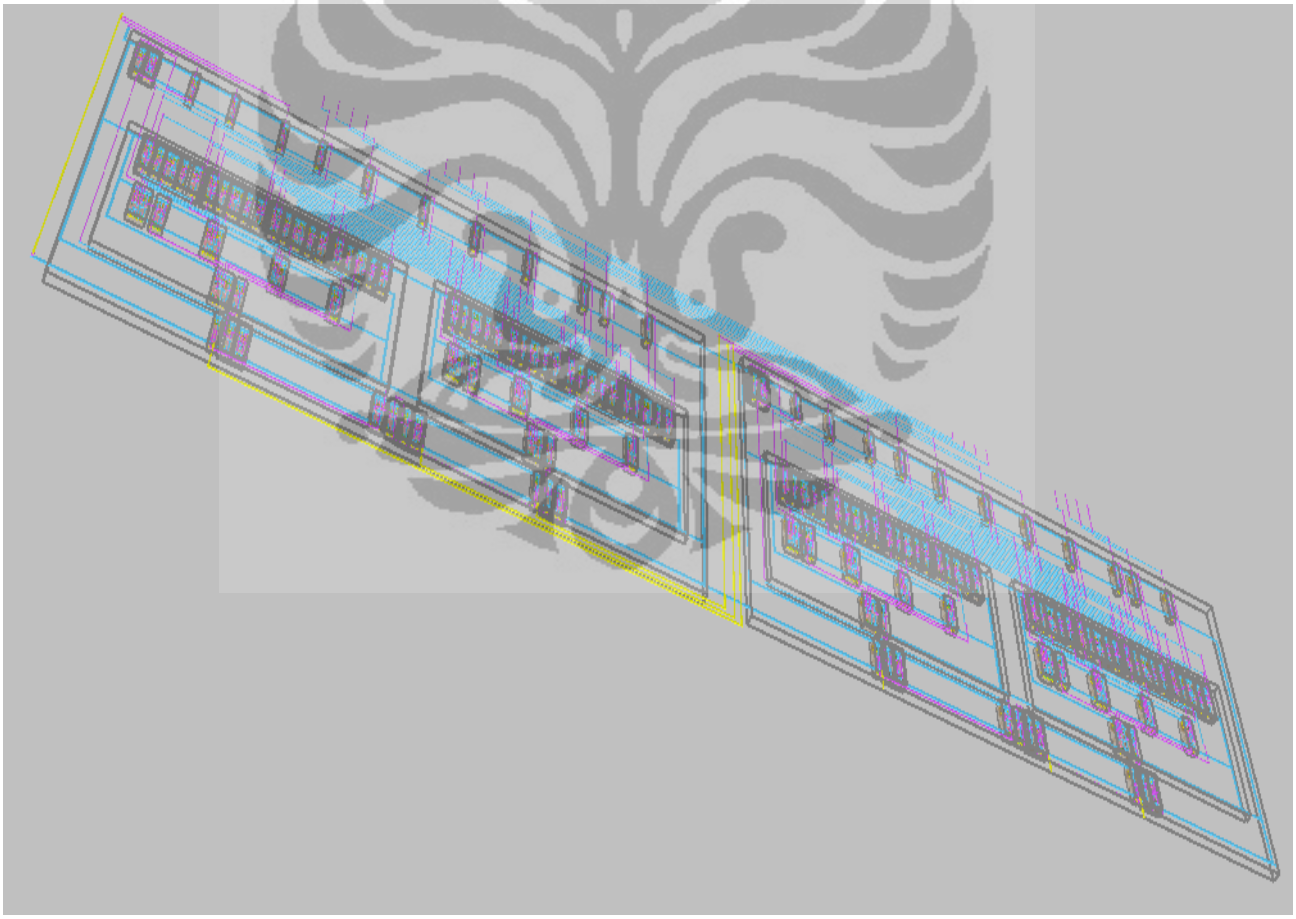


Gambar 2. Layout CMOS Full Multiplicator 8 bit 3D

LAMPIRAN 4
(Layout CMOS Comparator 8 bit 2D dan 3D)

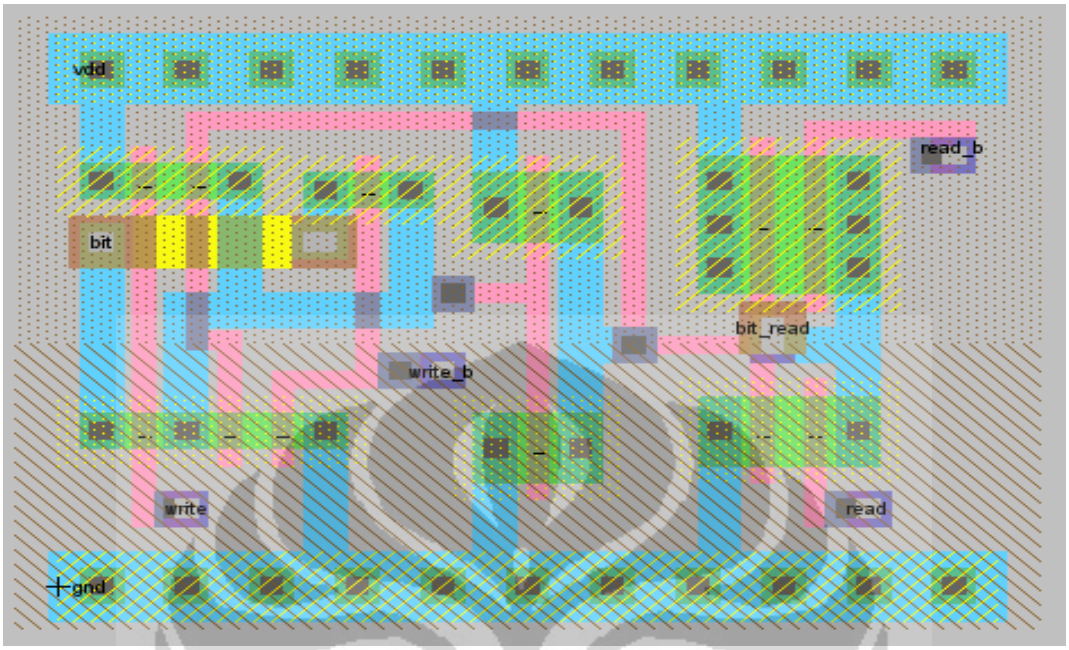


Gambar 1. Layout CMOS Comparator 8 bit 2D

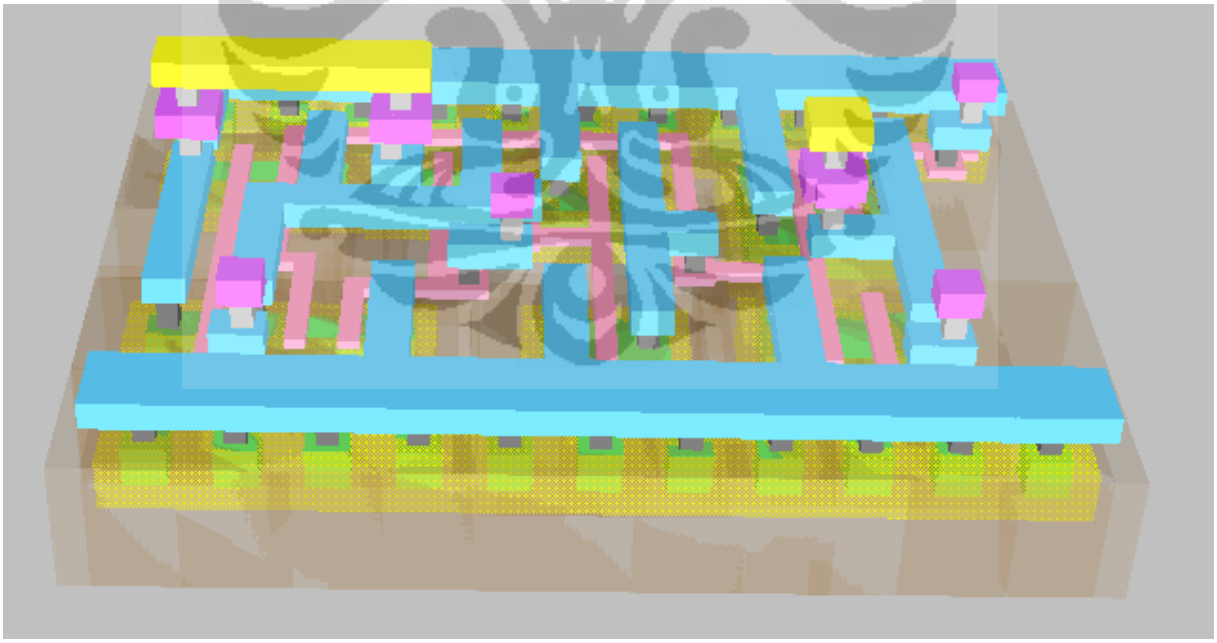


Gambar 2. Layout CMOS Comparator 8 bit 3D

LAMPIRAN 5
(Layout CMOS SRAM 2D dan 3D)

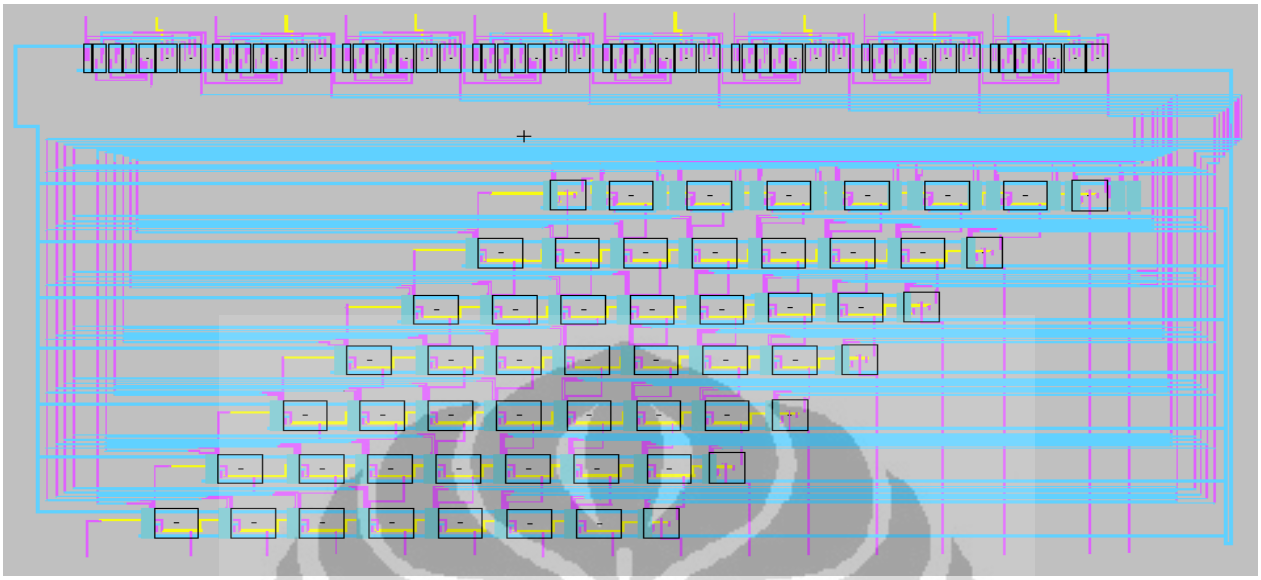


Gambar 1. Layout CMOS SRAM 2D

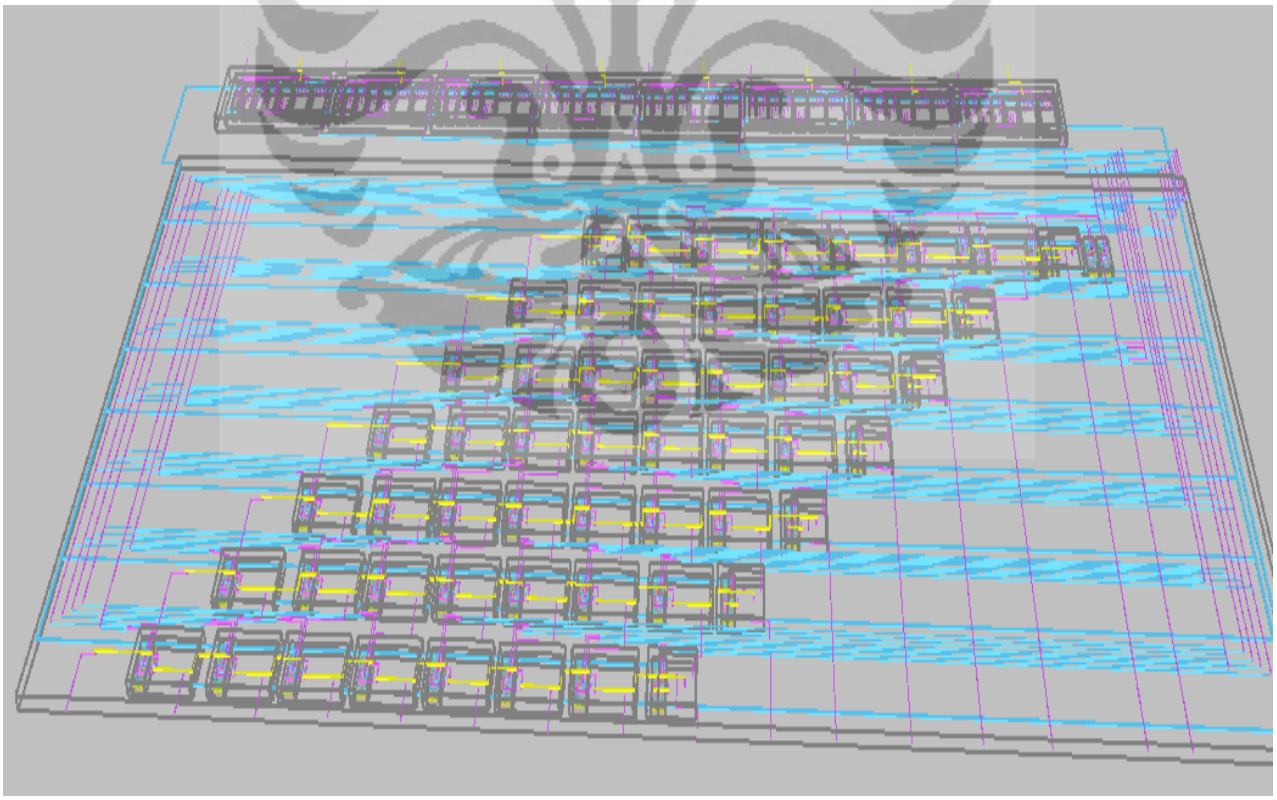


Gambar 2. Layout CMOS SRAM 3D

LAMPIRAN 6
(Layout CMOS Euclid Distance 2D dan 3D)

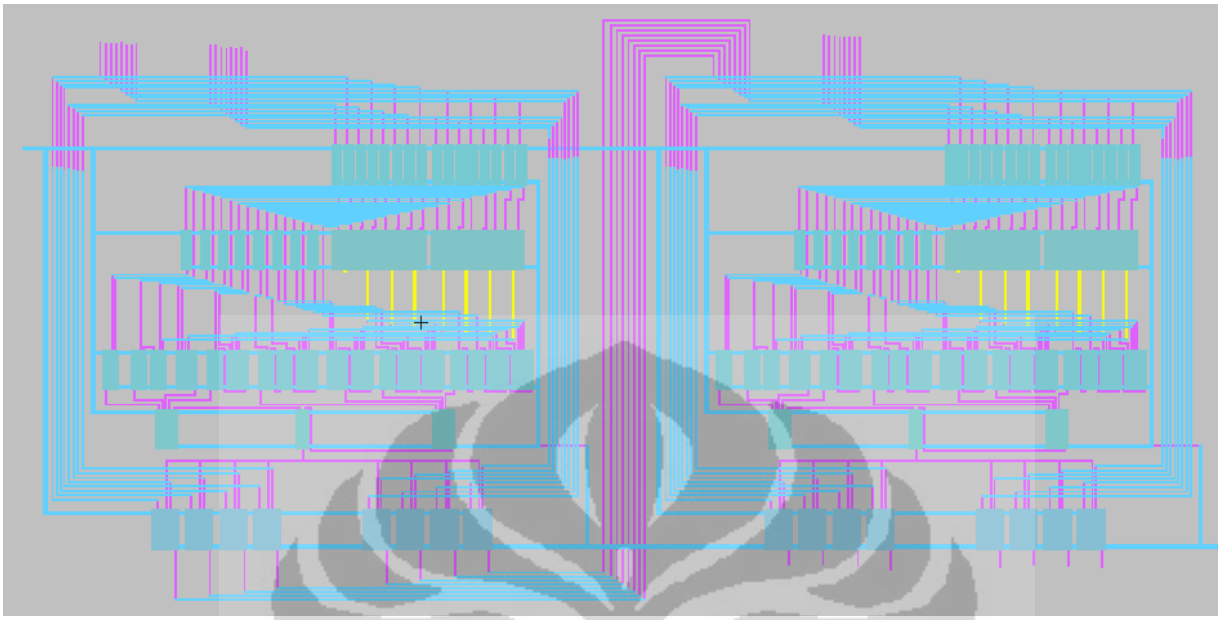


Gambar 1. Layout CMOS Euclid Distance 2D

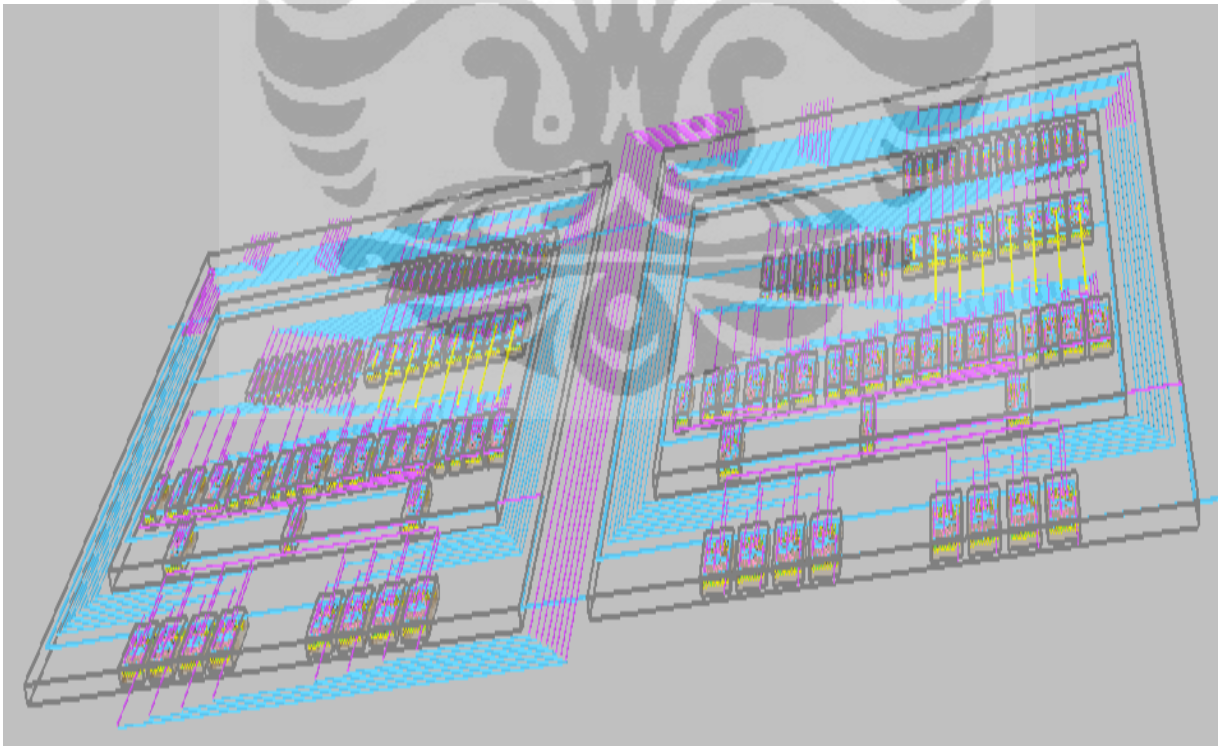


Gambar 2. Layout CMOS Euclid Distance 3D

LAMPIRAN 7
(Layout CMOS Minimal Global Cost 2D dan 3D)



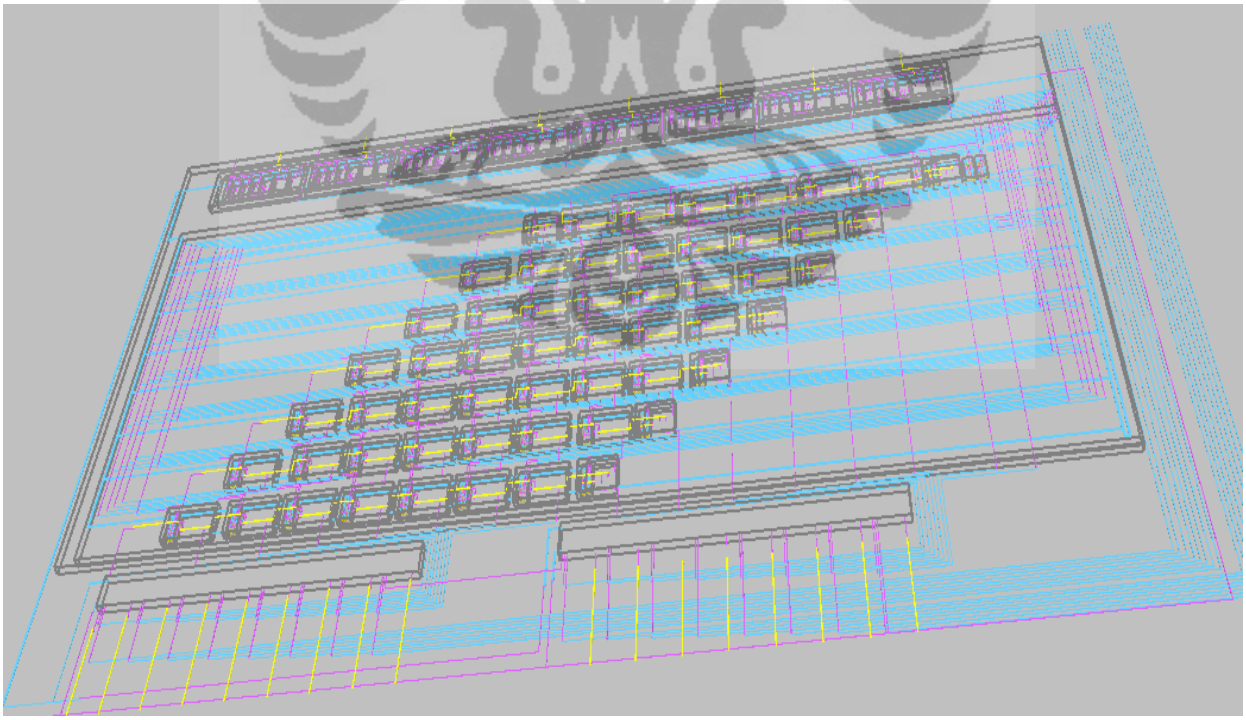
Gambar 1. Layout CMOS Minimal Global Cost 2D



Gambar 2. Layout CMOS Minimal Global Cost 3D

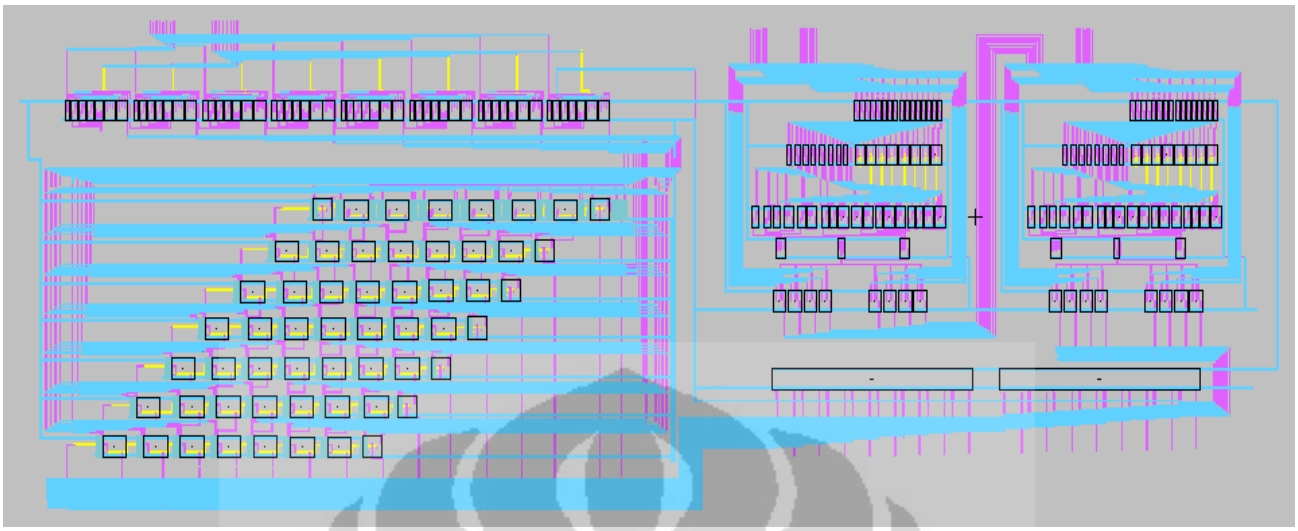
LAMPIRAN 8
(Layout CMOS Cost Matriks A 2D dan 3D)

Gambar 1. Layout CMOS Cost Matriks A 2D



Gambar 2. Layout CMOS Cost Matriks A 3D

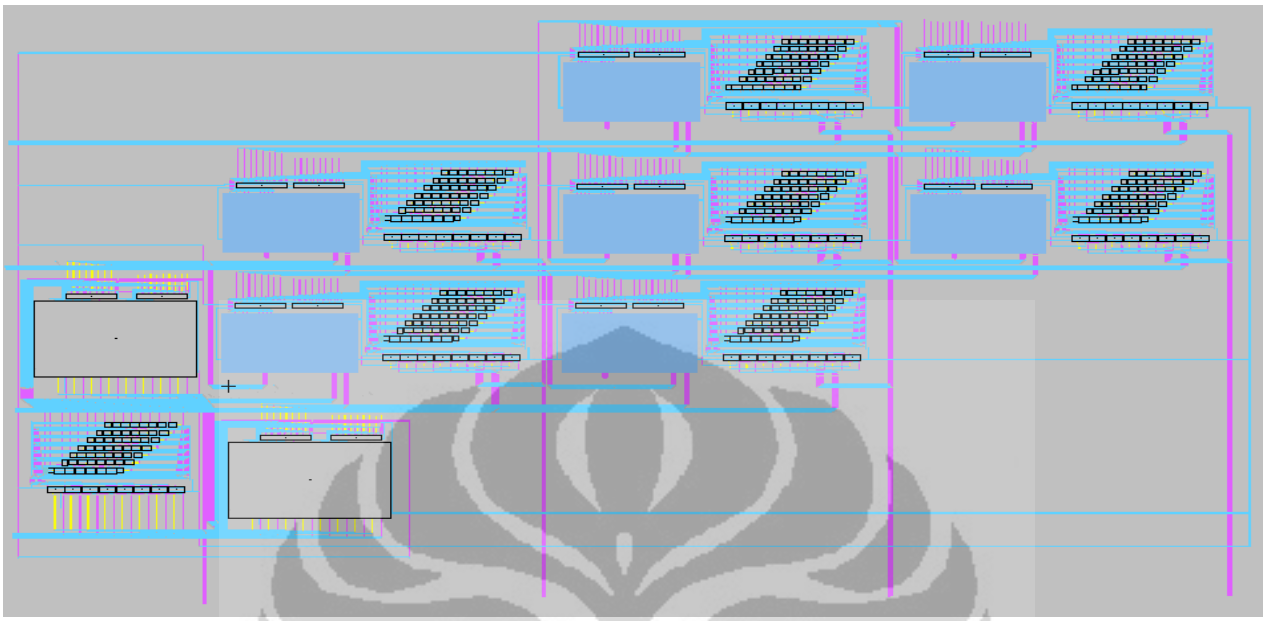
LAMPIRAN 9
(Layout CMOS Cost Matriks B 2D dan 3D)



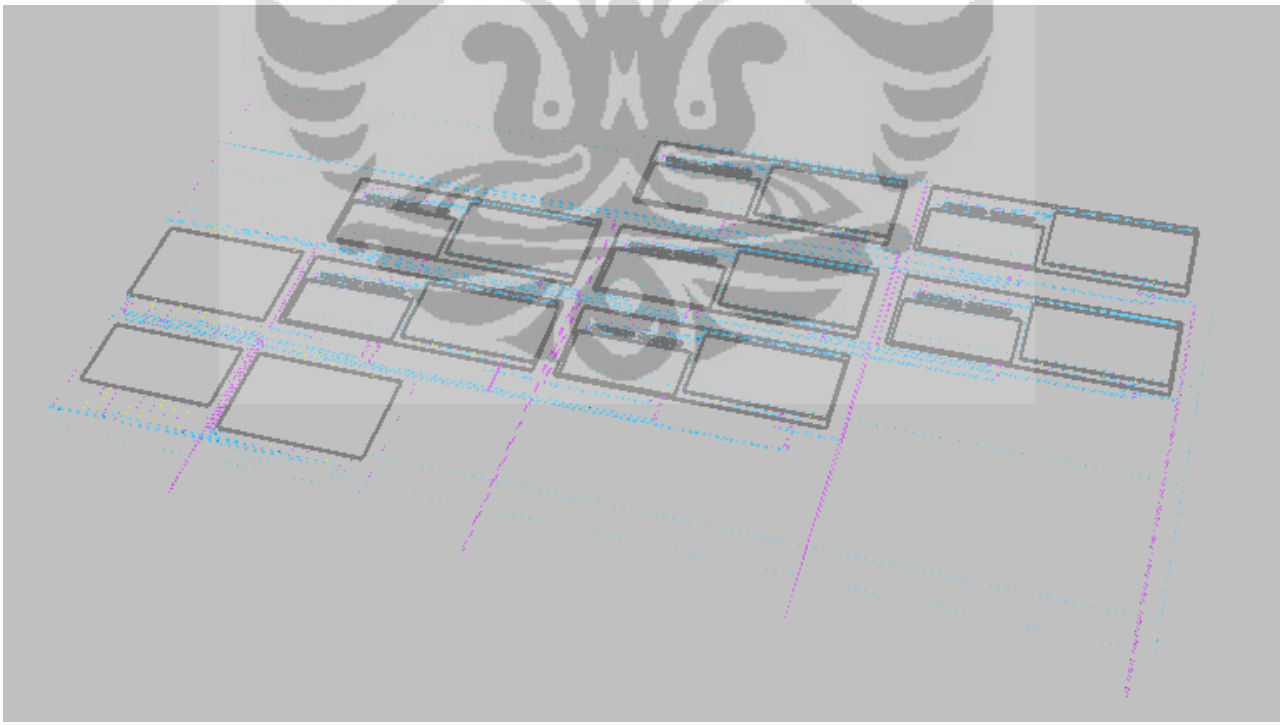
Gambar 1. Layout CMOS Cost Matriks B 2D

Gambar 2. Layout CMOS Cost Matriks B 3D

LAMPIRAN 10
(Layout CMOS Full Datapath 2D dan 3D)

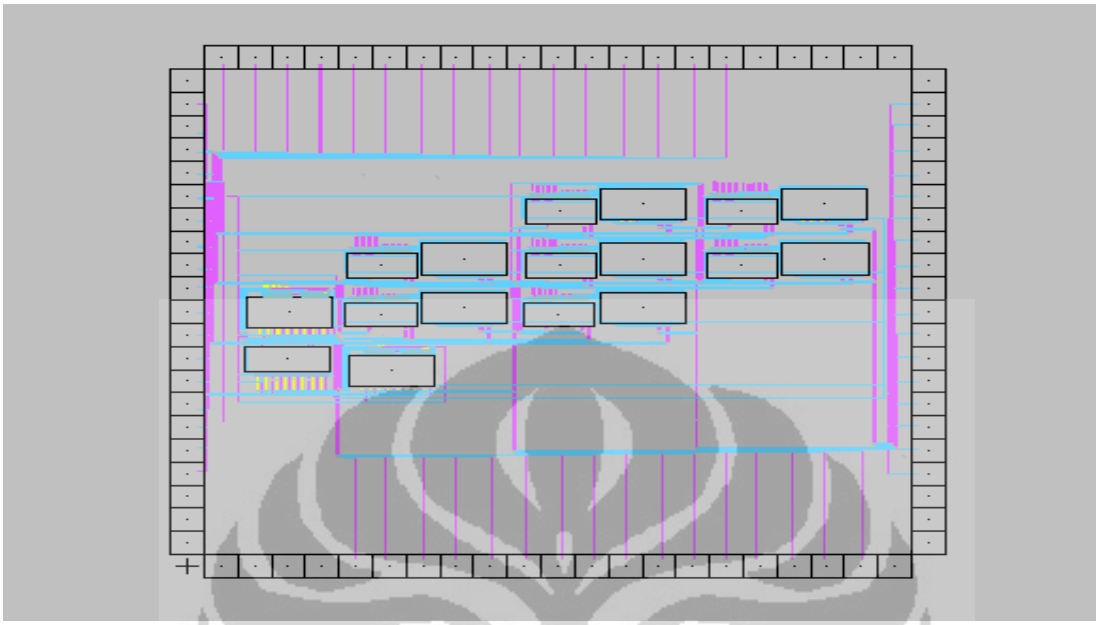


Gambar 1. Layout CMOS Full Datapath 2D

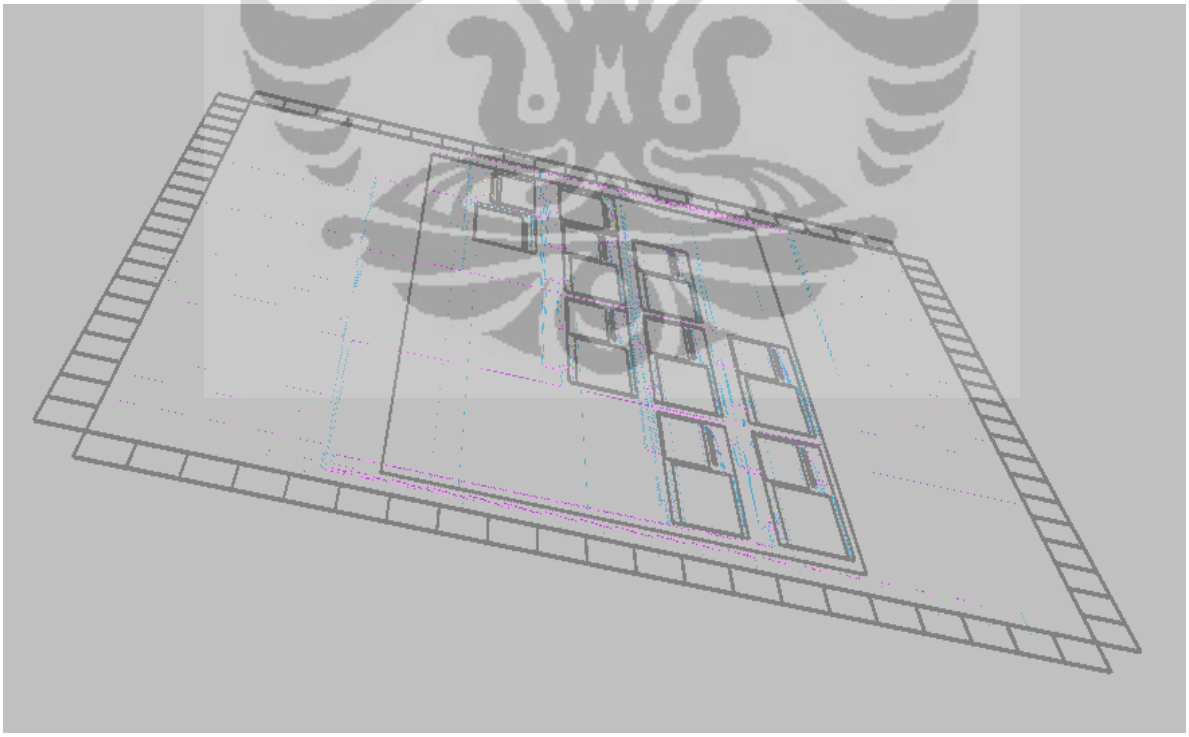


Gambar 2. Layout CMOS Full Datapath 3D

LAMPIRAN 11
(Layout CMOS Final Chip DTW 2D dan 3D)



Gambar 1. Layout CMOS Final Chip DTW 2D



Gambar 2. Layout CMOS Final Chip DTW 3D

LAMPIRAN 12

Test Vector Layout CMOS IC Pattern Matching dengan Algoritma FastDTW

Test vector dilakukan untuk melakukan pengecekan hasil dari layout CMOS IC fast DTW yang dirancang menggunakan software CAD Electric oleh penulis dengan hasil dari pemrograman JAVA fast DTW yang dibuat oleh Stan Salvador.

1. Berikut dilampirkan data percobaan pattern matching dengan algoritma fast DTW dimana ukuran data array yang digunakan adalah 4.
   Algoritm fast DTW dibuat menggunakan bahasa pemrograman JAVA
   Dibuat oleh : Stan Salvador (yang mengusulkan metode fast DTW)

```
DATA INPUT SERIES :
----------------------
time series 1:
6.0, 7.0, 7.0, 8.0
time series 2:
5.0, 2.0, 6.0, 4.0
----------------------

SHRUNK DATA :
------------------------------------------------------------
shrunk I :(4 point time series represented as 2 points)
6.5
7.5
shrunk J :(4 point time series represented as 2 points)
3.5
5.0
------------------------------------------------------------

SHRUNK WARP PATH DAN EX WINDOW:
---------------------------------------
shrunk warp path:[(0,0),(1,1)]
shrunk warp path size:2
warped I:0
warped J:0
block I:2
block J:2
current I:2147483647
current J:2147483647
mark visited0:0...0
mark visited0:0...1
mark visited1:0...0
mark visited1:0...1
last warped I:0
last warped J:0
-------------loop1-
warped I:1
warped J:1
block I:2
block J:2
```

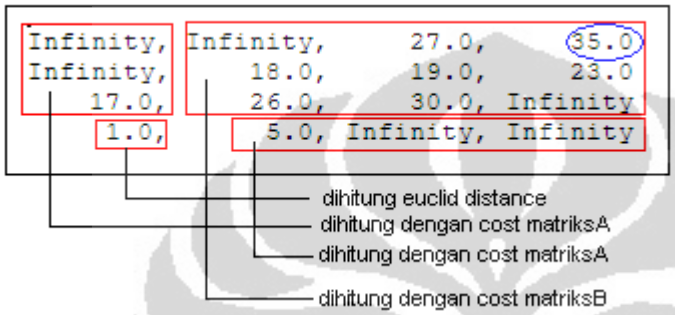
```
current I:0
current J:0
mark visited1:0...2
mark visited2:1...1
mark visited2:1...2
mark visited2:1...3
mark visited3:2...2
mark visited3:2...3
last warped I:1
last warped J:1
-------------loop1-
masuk expand windows
-------------loop2-
window :i=0, j=0...1
i=1, j=0...2
i=2, j=1...3
i=3, j=2...3
-----------------------------------------

COST MATRIKS INIT :
---------------------------------
costMatrix init :
Infinity,Infinity,0.0,0.0
Infinity,0.0,0.0,0.0
0.0,0.0,0.0,Infinity
0.0,0.0,Infinity,Infinity
-----------------------------------

MINIMAL COST DAN WARP PATH :
-----------------------------------
iterator1 :true
iterator2 :true
iterator2 :true
iterator2 :true
iterator2 :true
iterator2 :true
iterator2 :true
iterator2 :true
iterator2 :true
iterator2 :true
iterator2 :false
cost matrix size :10

min cost :35.0

Infinity, Infinity,     27.0,     35.0
Infinity,     18.0,     19.0,     23.0
    17.0,     26.0,     30.0, Infinity
     1.0,      5.0, Infinity, Infinity

Warp Distance: 35.0
Warp Path:     [(0,0),(0,1),(1,2),(2,2),(3,3)]
Waktu proses : 359 ms
-----------------------------------------
```

2. Berikut dilampirkan data percobaan pattern matching dengan algoritma fast DTW dimana ukuran data array yang digunakan adalah 4.
   Layout CMOS IC fast DTW dibuat menggunakan CAD Electric
   Dibuat oleh : Andi Yusuf (S2 UI Elektro – Desain VLSI & Embedded System)

MENGHITUNG COST MATRIKS DAN MIN COST :

Metode perhitungan cost matriks dan minimal cost

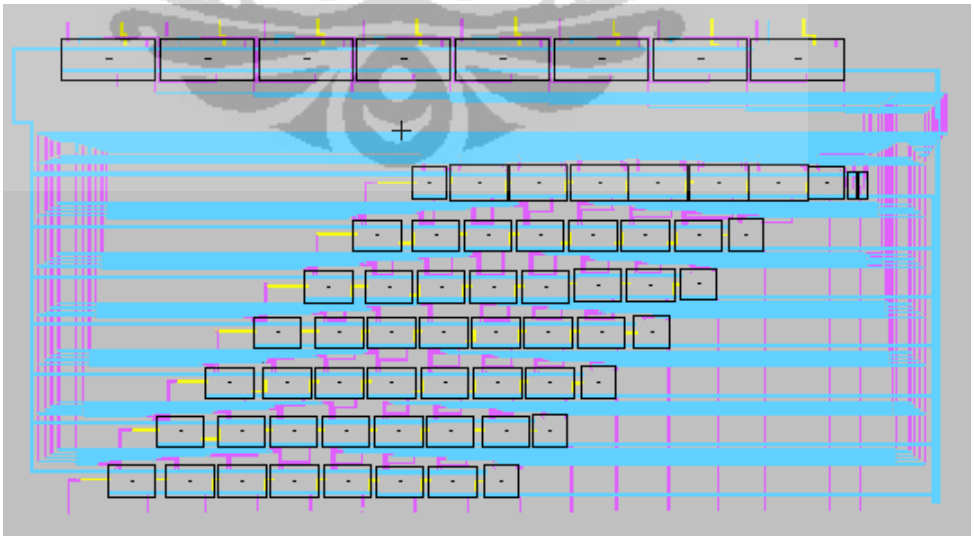
COST MATRIKS



Pengecekan hasil sub datapath
Pengecekan hasil sub datapath adalah pengecekan yang dilakukan dengan mengamati hasil simulasi dari setiap sub datapath penyusun full datapath dari IC fast DTW dengan menggunakan tools simulasi IRSIM dari Electric TM.

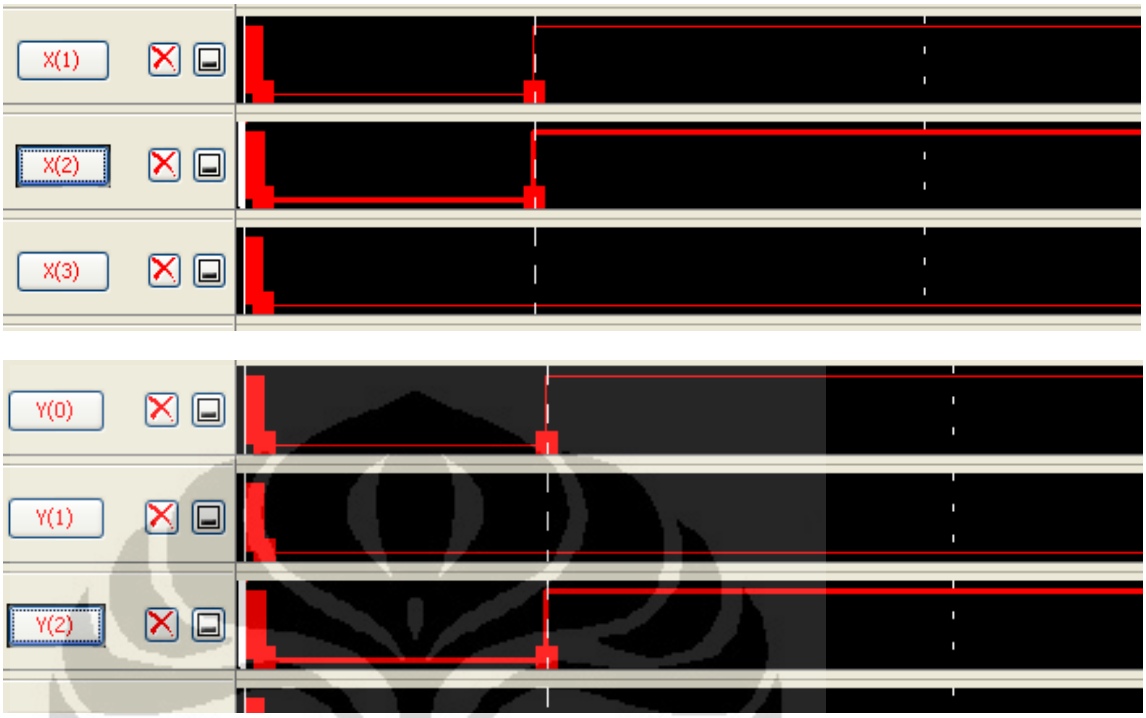
1. Pengecekan Euclid distance

   Layout CMOS Euclid distance :



   Input simulasi IRSIM Euclid Distance :
   1. Input vector x(0) = 6

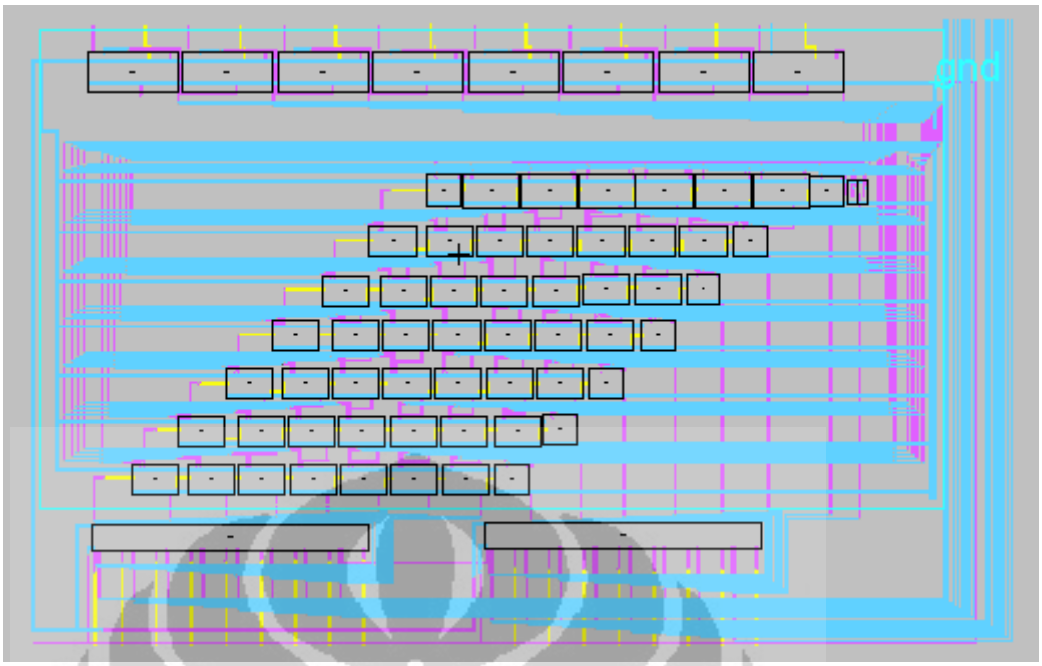
2. Input vector y(0) = 5



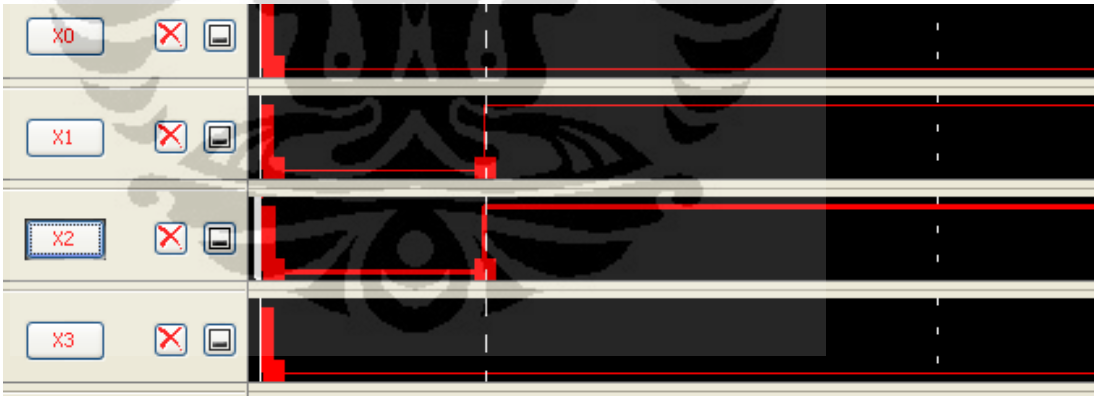
Hasil dari Euclid Distance dengan x = 6 dan y = 5 adalah output(0,0)=1

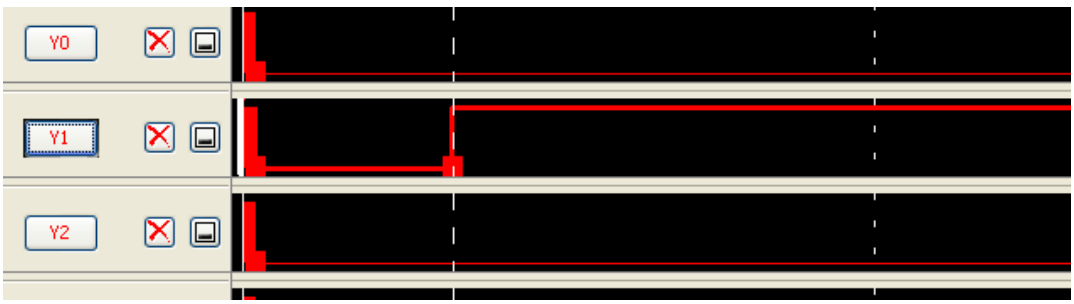
2. Pengecekan cost matriks A

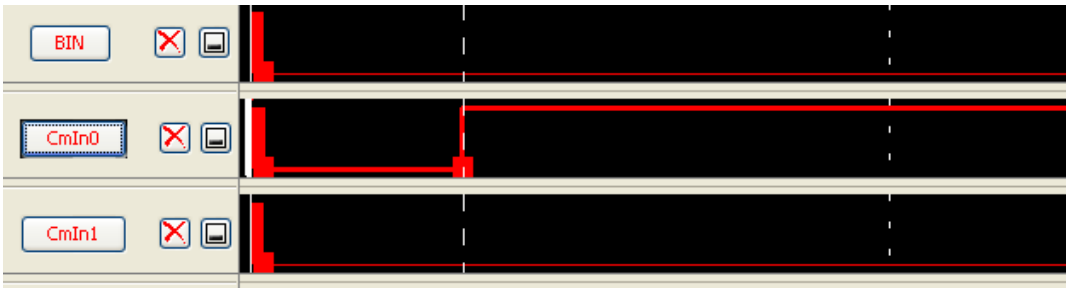
Layout CMOS cost matriks A :



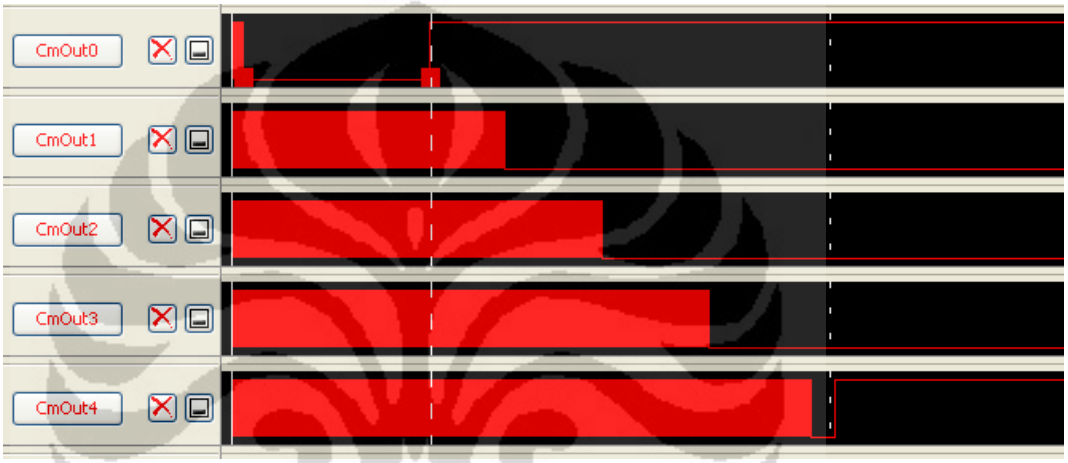
Input simulasi IRSIM cost matriks A :
1. Input vector x(0) = 6
2. Input vector y(1) = 2
3. Output(0,0) = 1 => CmIn = 1





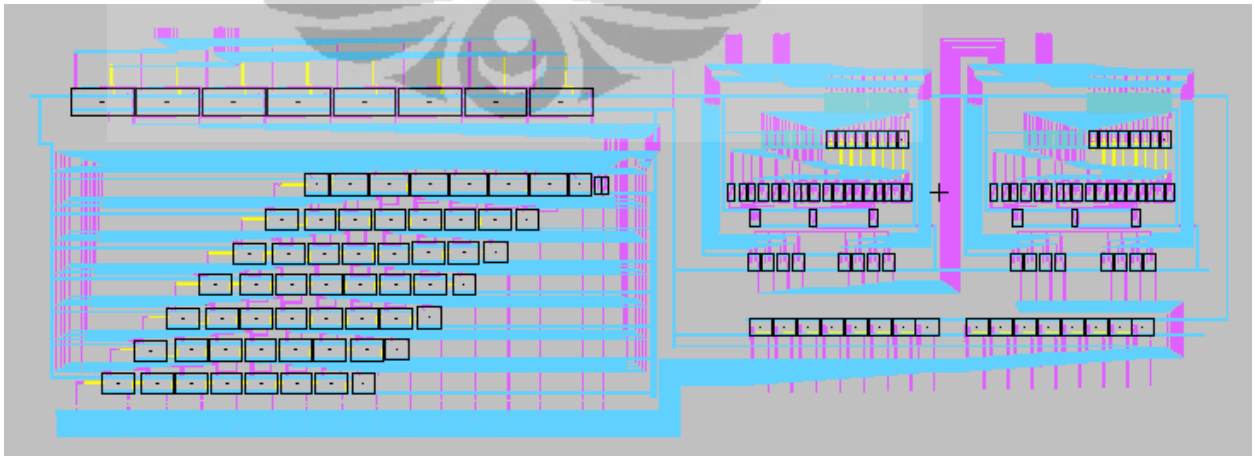


Hasil dari cost matriks A dengan input vector x(0) = 6 , y(0) = 2 dan output(0,0) = 1 adalah output(0,1) = 17

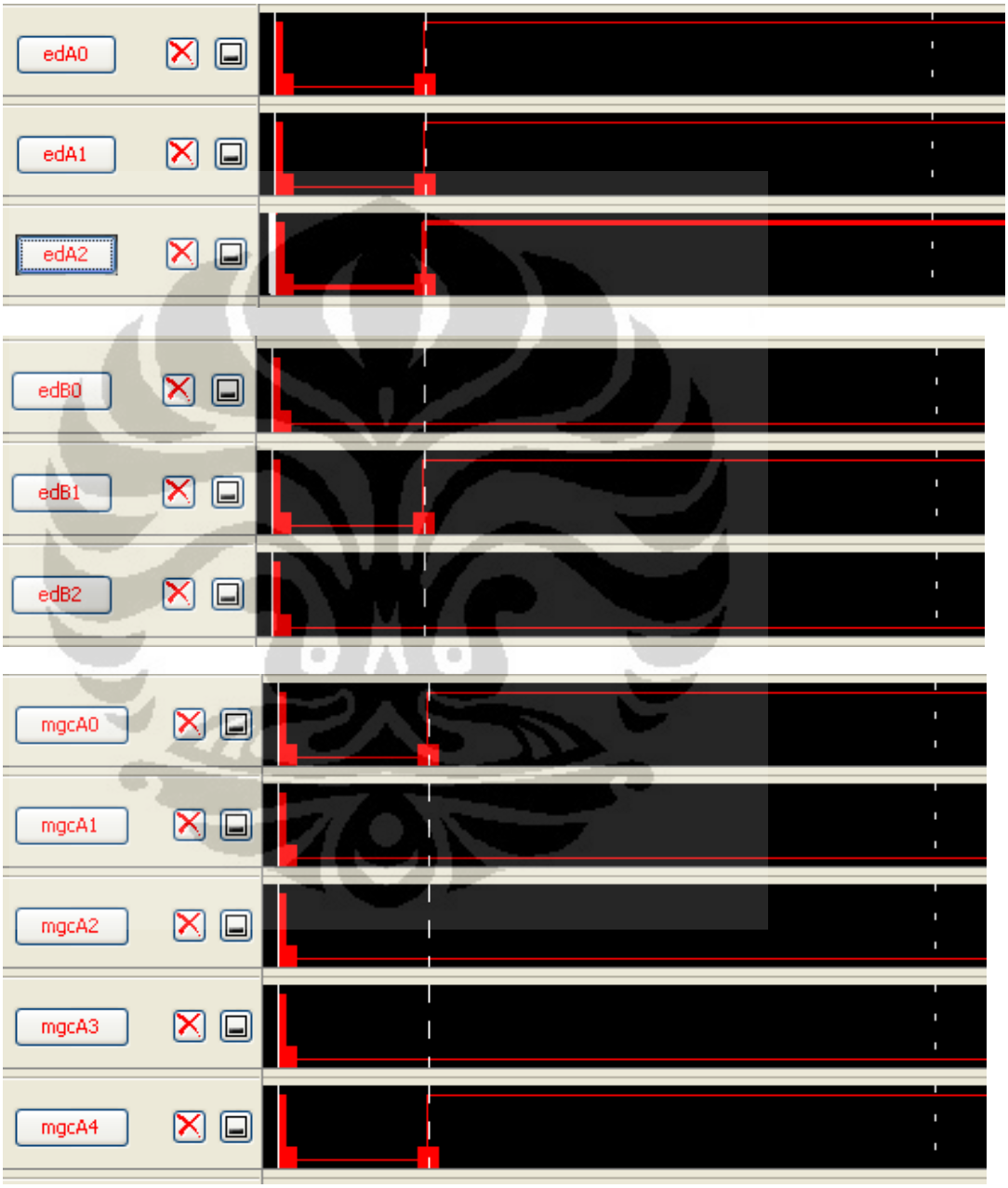


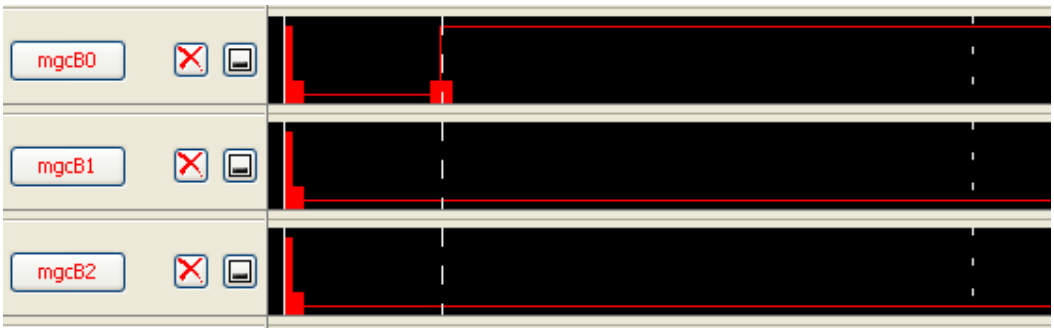
3. Pengecekan cost matriks B

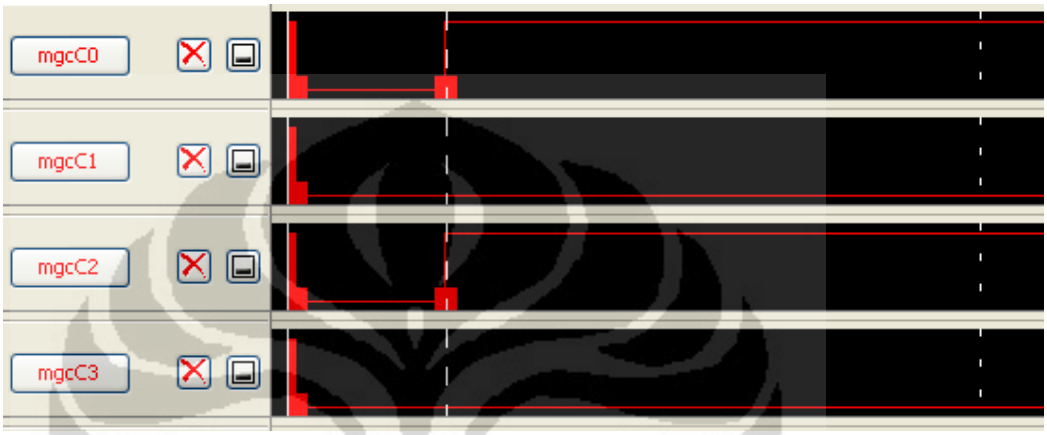
Layout CMOS cost matriks B :



```
Input simulasi IRSIM cost matriks B :
1. Input vector x(1) = 7 => edA = 7
2. Input vector y(1) = 2 => edB = 2
3. Output(0,0) = 1 => mgcA = 1
4. Output(0,1) = 17 => mgcB = 17
5. Output(1,0) = 5 => mgcC = 5
```





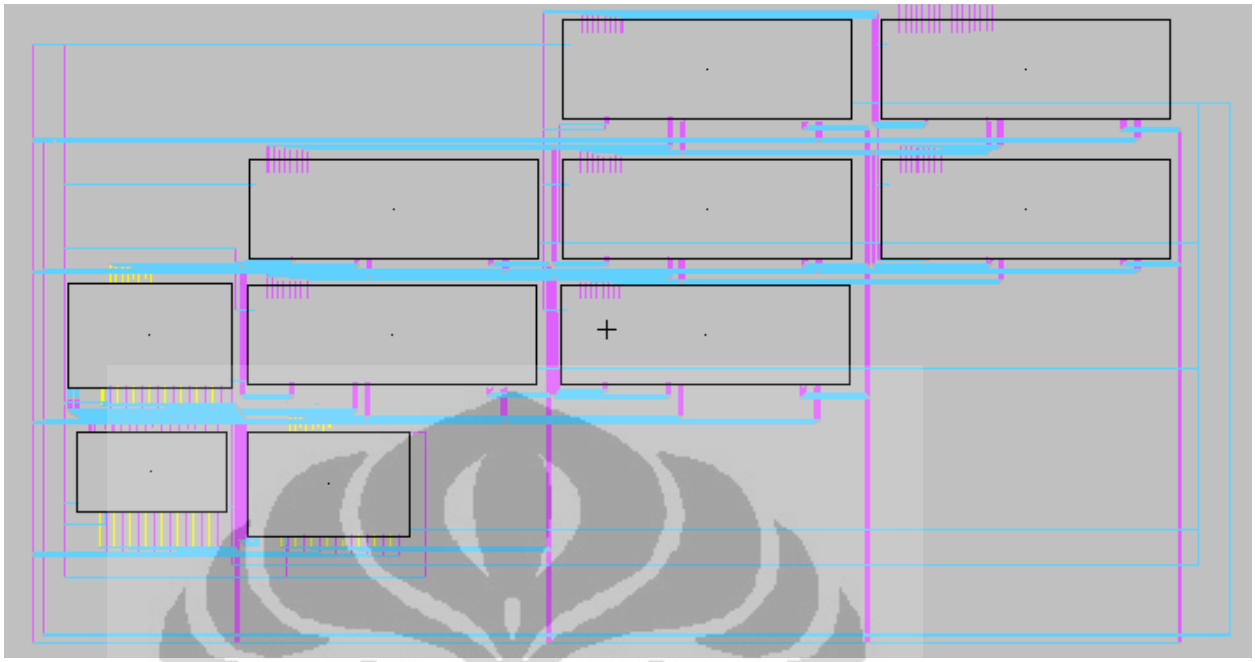


Hasil dari cost matriks B dengan input vector x(1) = 7 , y(1) = 2, output(0,0) = 1, output(0,1) = 17 dan output(1,0) = 5 adalah output(1,1) = 26

Pengecekan hasil full datapath

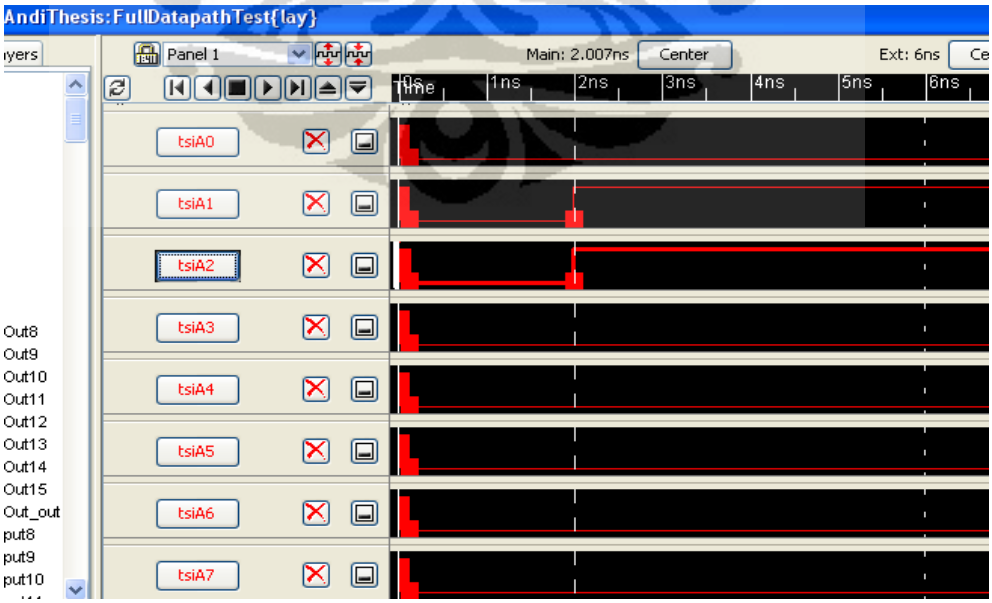
Pengecekan hasil full datapath adalah pengecekan yang dilakukan dengan mengamati hasil simulasi full datapath dari IC fast DTW. Hal ini dilakukan menggunakan LTSpice dan IRSIM untuk full datapath.

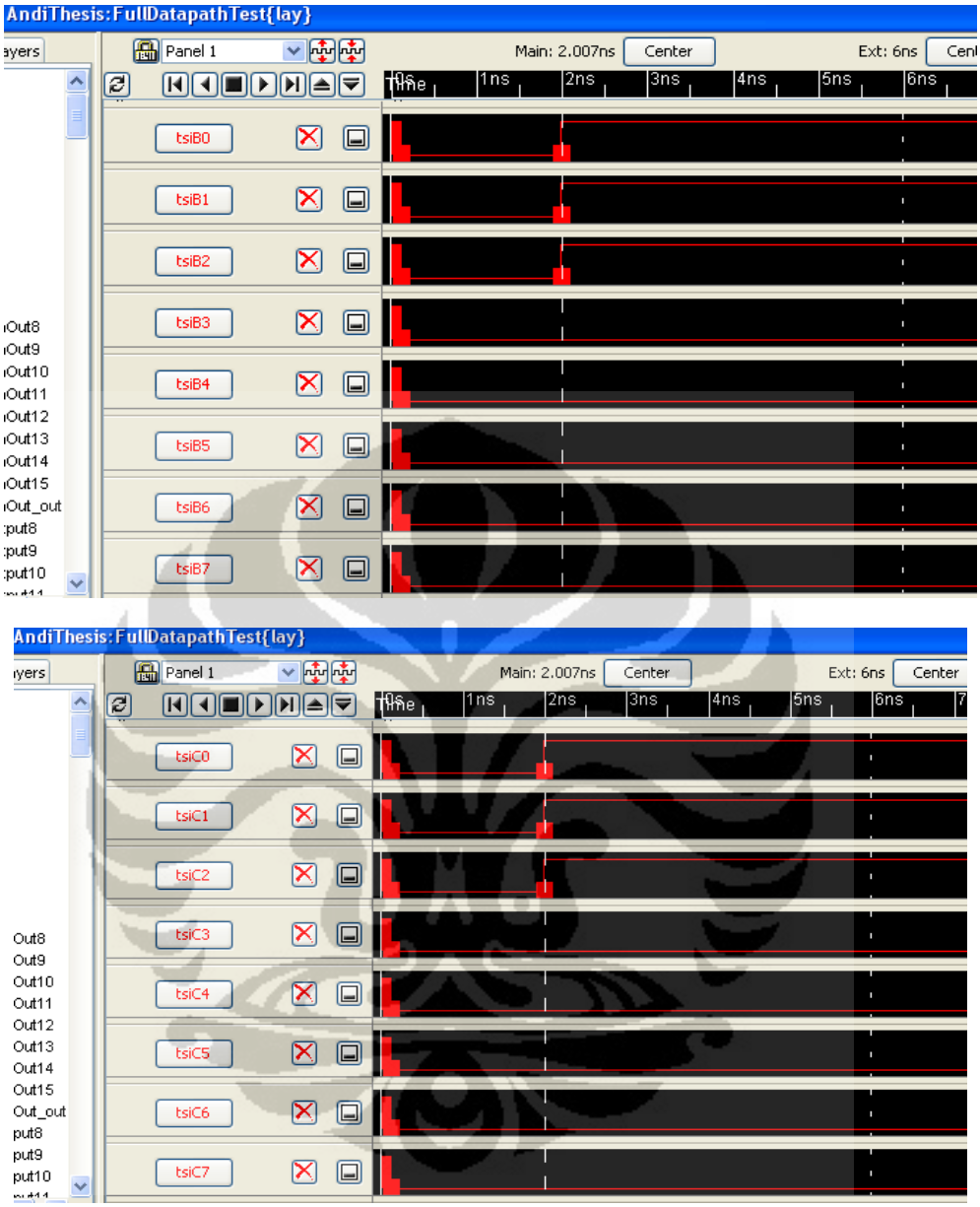
Layout CMOS Full Datapath :

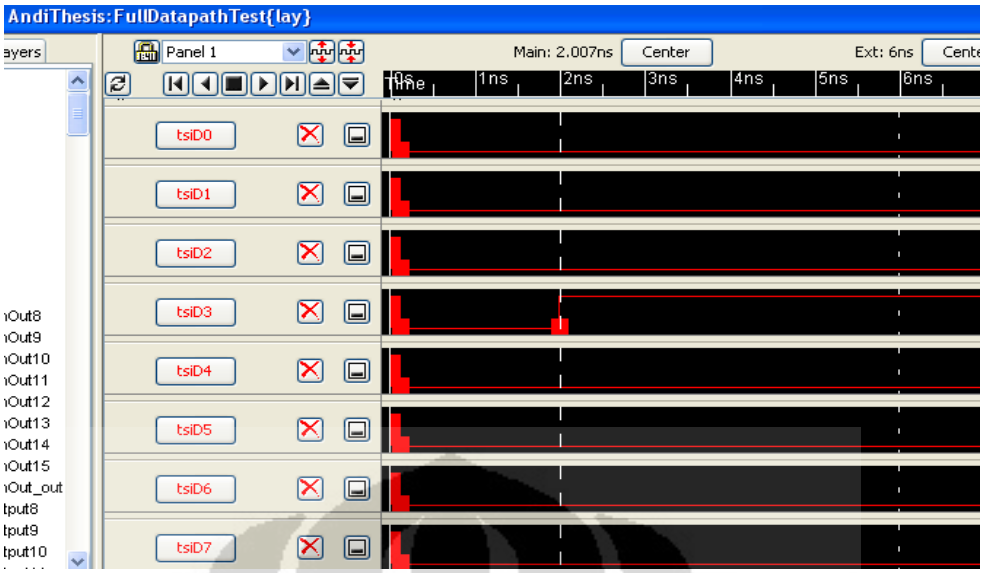


Input simulasi IRSIM Full Datapath :

1. Input vector tsI = (6,7,7,8)
   - tsiA = 6
   - tsiB = 7
   - tsiC = 7
   - tsiD = 8

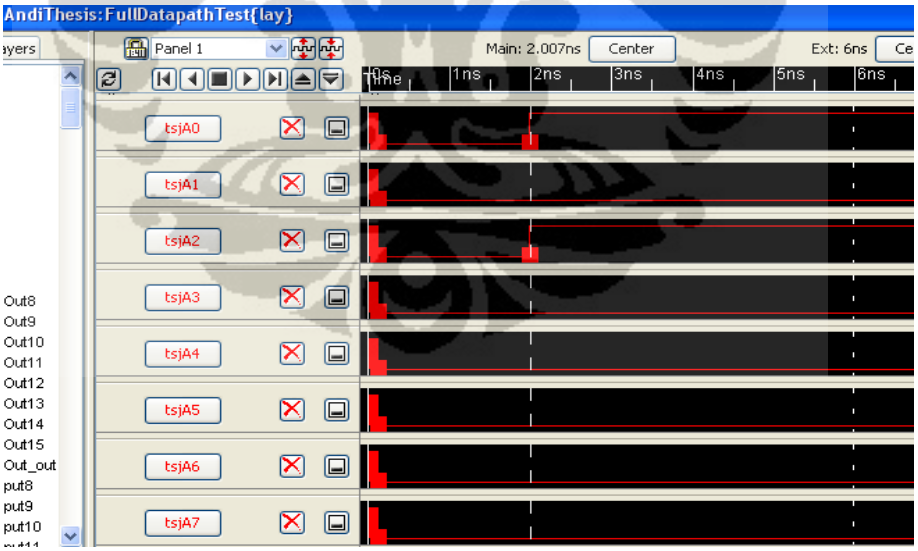


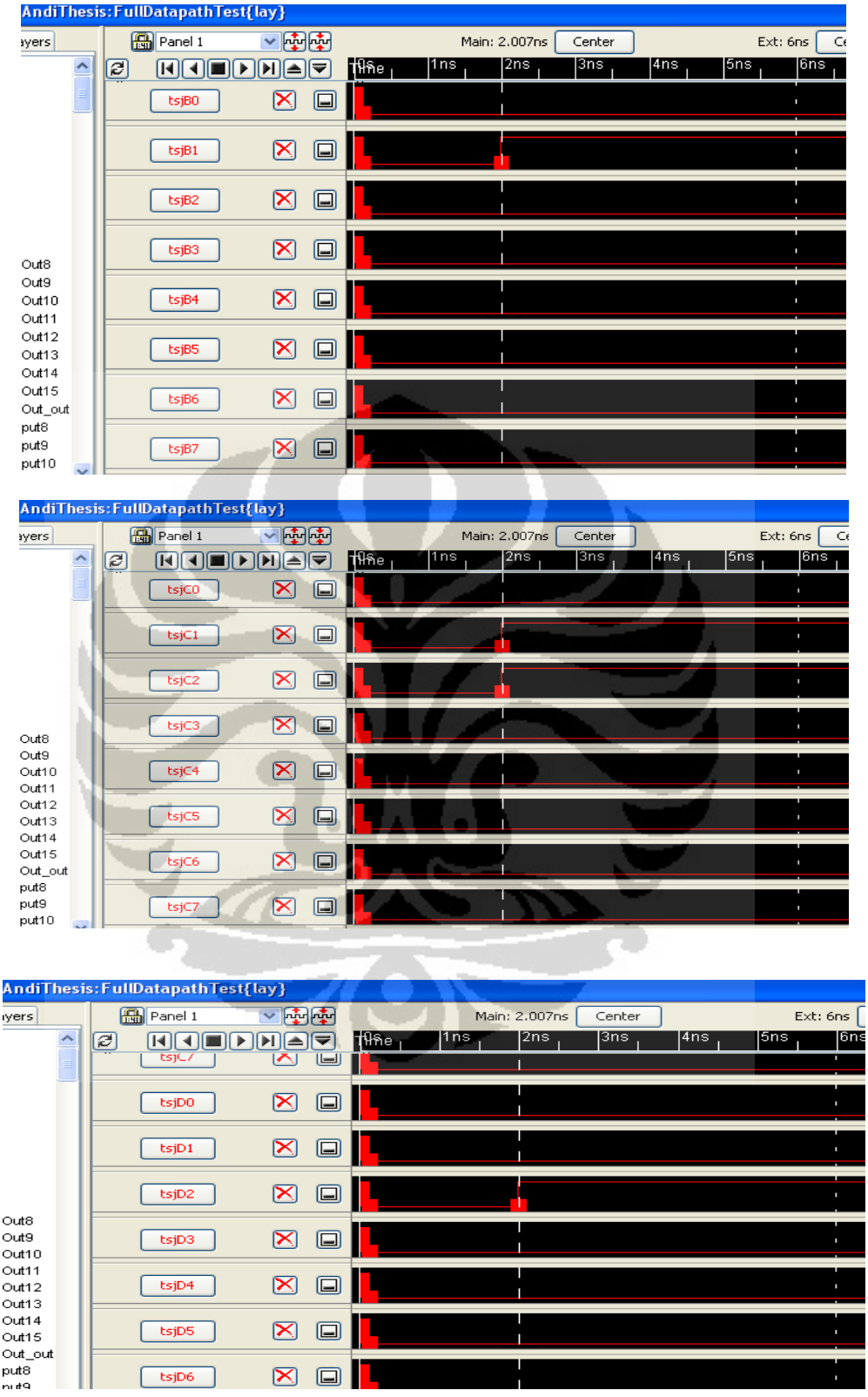

AndiThesis:FullDatapathTest{lay}
Panel 1
Main: 2.007ns
Center
Ext: 6ns
Time
1ns
2ns
3ns
4ns
5ns
6ns
tsiB0
tsiB1
tsiB2
tsiB3
tsiB4
tsiB5
tsiB6
tsiB7
tsiC0
tsiC1
tsiC2
tsiC3
tsiC4
tsiC5
tsiC6
tsiC7
Out8
Out9
Out10
Out11
Out12
Out13
Out14
Out15
Out_out



Input simulasi IRSIM Full Datapath :

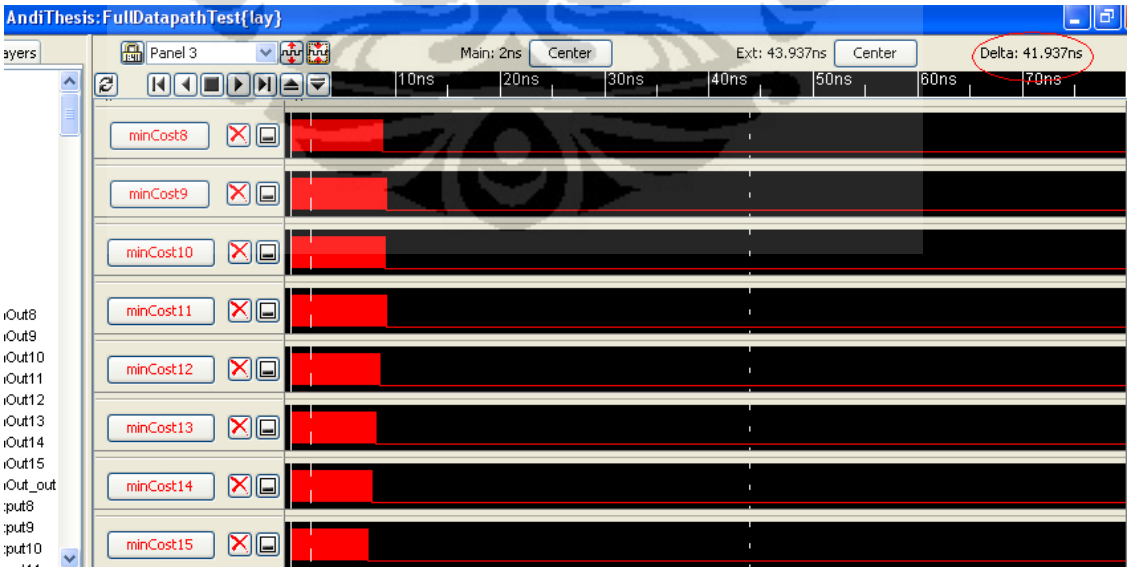
2. Input vector tsJ = (5,2,6,4)
   - tsjA = 5
   - tsjB = 2
   - tsjC = 6
   - tsjD = 4




AndiThesis:FullDatapathTest{lay}
Panel 1
Main: 2.007ns
Center
Ext: 6ns
Time
1ns
2ns
3ns
4ns
5ns
6ns
tsjB0
tsjB1
tsjB2
tsjB3
tsjB4
tsjB5
tsjB6
tsjB7
Out8
Out9
Out10
Out11
Out12
Out13
Out14
Out15
Out_out
tsjC0
tsjC1
tsjC2
tsjC3
tsjC4
tsjC5
tsjC6
tsjC7
tsjD0
tsjD1
tsjD2
tsjD3
tsjD4
tsjD5
tsjD6

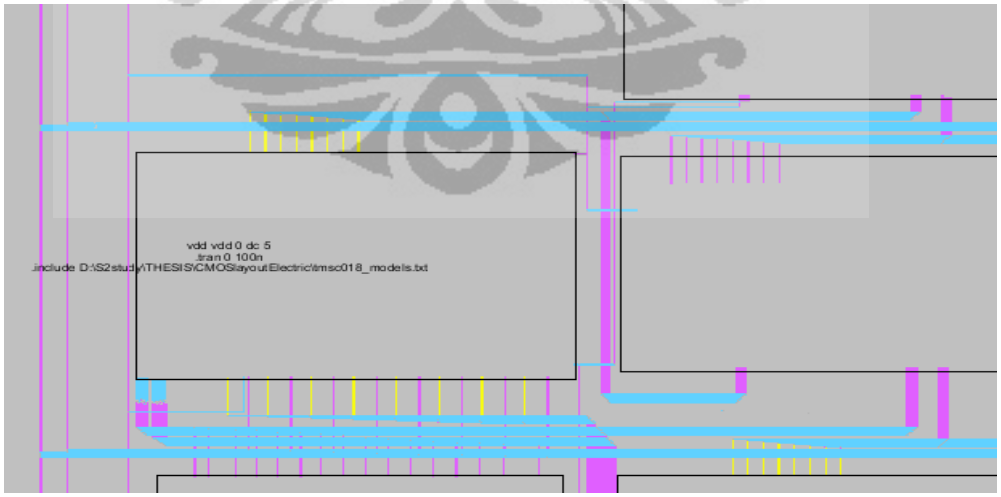
Hasil dari Full Datapath dengan input tsI = (6,7,7,8) dan tsJ = (5,2,6,4) adalah mininimal global cost = 35.

- minCost0 = '1'
- minCost1 = '1'
- minCost2 = '0'
- minCost3 = '0'
- minCost4 = '0'
- minCost5 = '1'
- minCost6 = '0'
- minCost7 = '0'
- minCost8 = '0'
- minCost9 = '0'
- minCost10 = '0'
- minCost11 = '0'
- minCost12 = '0'
- minCost13 = '0'
- minCost14 = '0'
- minCost15 = '0'



Pada simulasi LTSpice terdapat beberapa setting parameter awal yang ditentukan yaitu

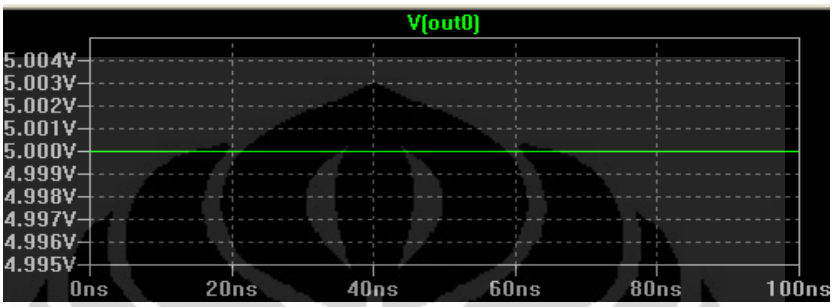
f. *Power supply* dalam hal ini vdd diberikan tegangan DC 5V
g. Input tsiA bernilai 6 sehingga tsiA(1) dan tsiA(2) dihubungkan pada vdd sedangkan tsiA(0) dan tsiA(3-7) dihubungkan pada ground.
h. Input tsiB bernilai 7 sehingga tsiB(0), tsiB(1) dan tsiB(2) dihubungkan pada vdd sedangkan tsiB(3-7) dihubungkan pada ground.
i. Input tsiC bernilai 7 sehingga tsiC(0), tsiC(1) dan tsiC(2) dihubungkan pada vdd sedangkan tsiC(3-7) dihubungkan pada ground.
j. Input tsiD bernilai 8 sehingga tsiD(3) pada vdd sedangkan tsiD(0-2) dan tsiD(4-7) dihubungkan pada ground.
k. Input tsjA bernilai 5 sehingga tsjA(0) dan tsjA(2) dihubungkan pada vdd sedangkan tsjA(1) dan tsjA(3-7) dihubungkan pada ground.
l. Input tsjB bernilai 2 sehingga tsjB(1) dihubungkan pada vdd sedangkan tsjB(0) dan tsjB(2-7) dihubungkan pada ground.
m. Input tsjC bernilai 6 sehingga tsjC(1) dan tsjC(2) dihubungkan pada vdd sedangkan tsjC(0) dan tsjC(3-7) dihubungkan pada ground.
n. Input tsjD bernilai 4 sehingga tsjD(2) pada vdd sedangkan tsjD(0-1) dan tsjD(3-7) dihubungkan pada ground.
o. Parameter model yang digunakan yaitu tmsc018_models.txt yang merupakan parameter TMSC018.
p. Simulasi analisis *transient* berjalan pada 0 – 100ns.

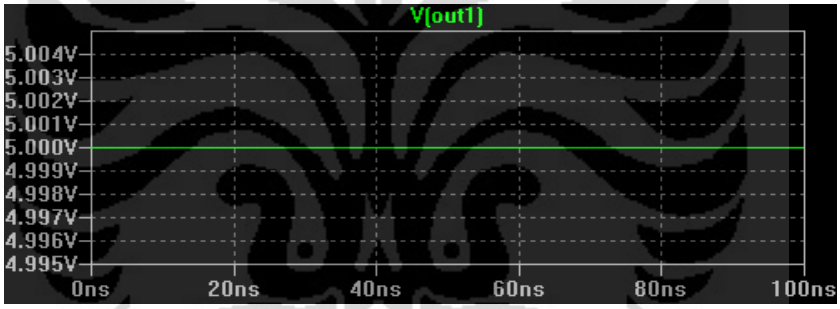


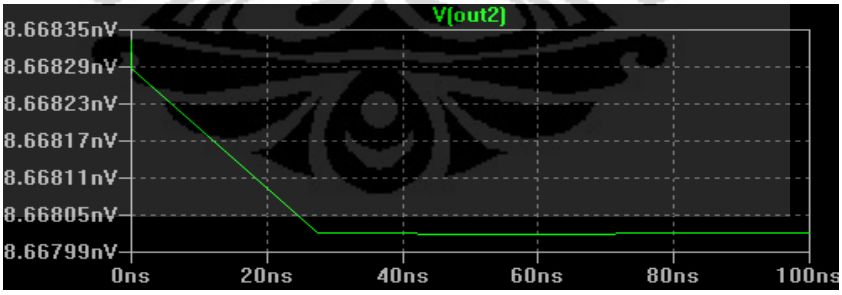
Dari setting parameter tersebut didapatkan hasil output(4,4) = 35 yang diperlihatkan pada grafik tegangan dibawah dimana 5 V menandakan bit 1 dan < 0.5 V menandakan bit 0.

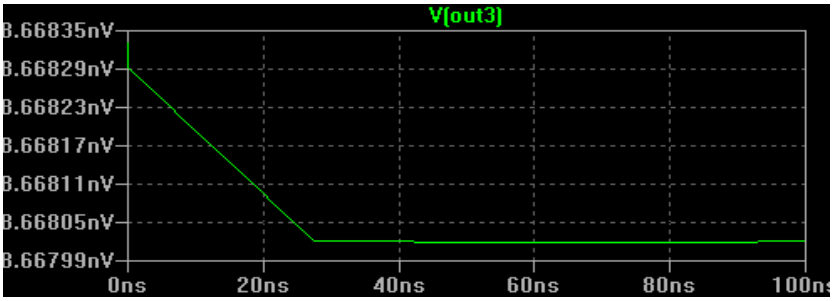
Dimana :

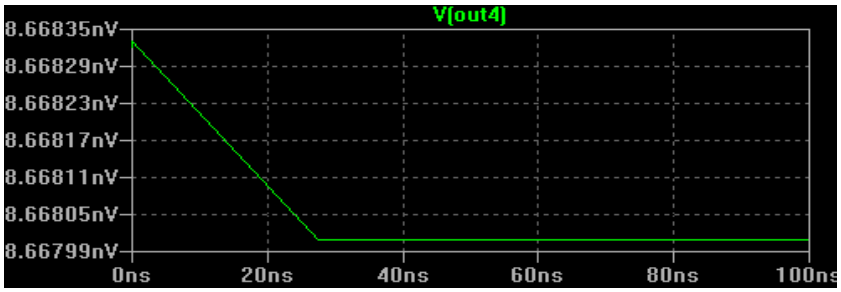
- V(out0) = 5V => output(0)='1'
- V(out1) = 5V => output(1)='1'
- V(out2) = 8,6nV => output(2)='0'
- V(out3) = 8,6nV => output(3)='0'
- V(out4) = 8,6nV => output(4)='0'
- V(out5) = 5V => output(5)='1'
- V(out6) = 8,6nV => output(6)='0'
- V(out7) = 8,6nV => output(7)='0'

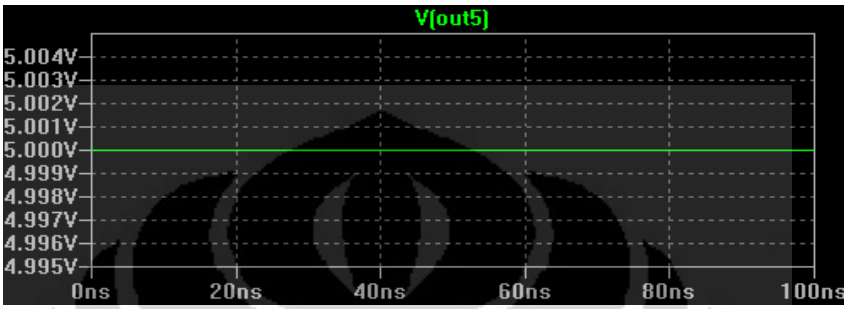


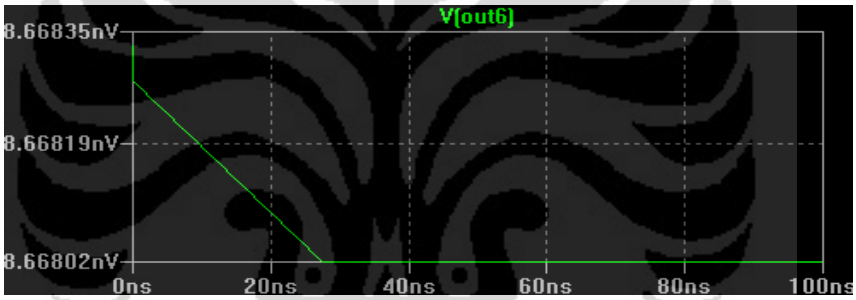


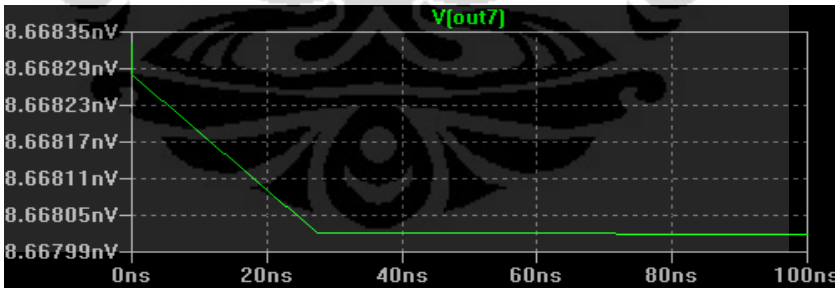






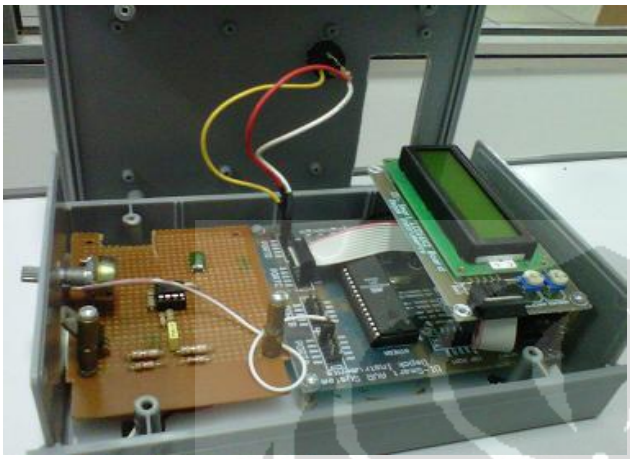


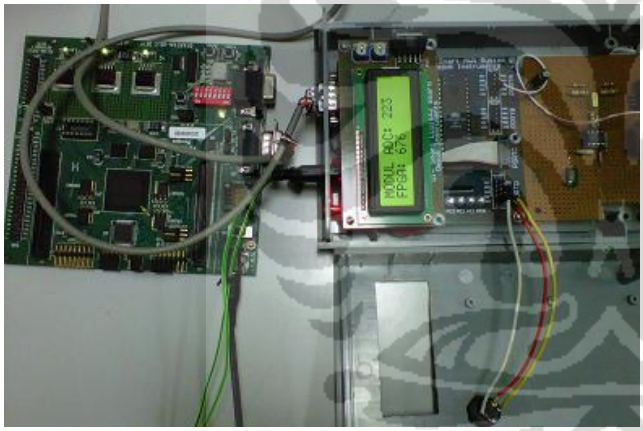




LAMPIRAN 16

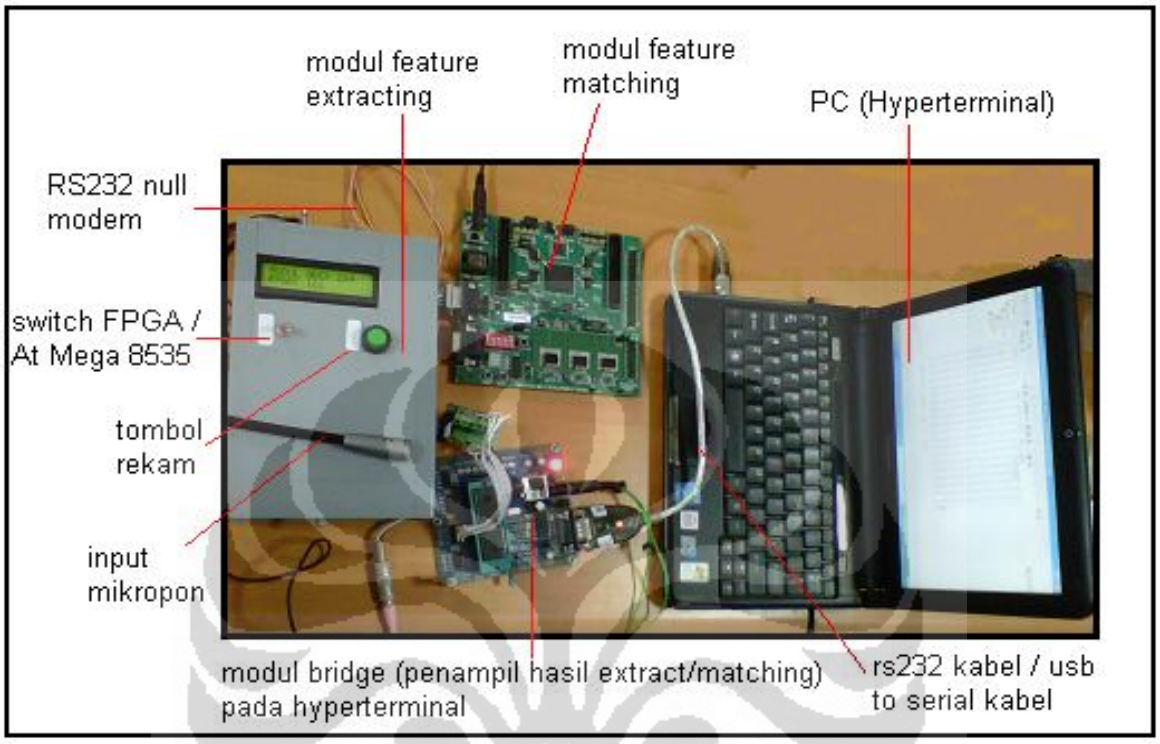
HARDWARE PROTOTYPE SPEECH RECOGNITION (DEVICE)

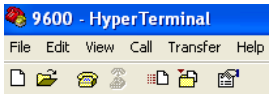




LAMPIRAN 17

SETTING ALAT UNTUK PERCOBAAN PENGENALAN SUARA



No.	Kegiatan	Keterangan
1	Siapkan *power supply* untuk setiap modul dan PC	Setiap modul terhubung power
2	Pilih switch untuk menampilkan hasil FPGA/ATmega8535 pada Hyperterminal	 arah atas ATmega8535 arah bawah FPGA
3	Tekan tombol *record* untuk menginput suara	 tekan tombol
4	Input suara melalui mikropon	 input suara
5	Lihat hasil pada LCD	
6	Untuk hasil detail dapat dilihat pada hyperterminal	

LAMPIRAN 18

HASIL PERCOBAAN PENGENALAN SUARA

a. Percobaan input suara ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’ dibandingkan *feature* suara 1

Tabel 1 hasil perbandingan *feature* suara 1

Perbandingan	Nilai minimum distance									
Suara ‘a’ vs *feature* 1	133	75	129	173	1	42	227	165	253	212
Suara ‘b’ vs *feature* 1	229	197	225	165	229	247	255	165	199	249
Suara ‘c’ vs *feature* 1	253	253	255	255	37	128	229	245	229	37
Suara ‘d’ vs *feature* 1	193	165	231	163	254	255	245	251	253	255
Suara ‘e’ vs *feature* 1	248	165	202	253	253	249	200	229	255	254

Perbandingan	Nilai minimum distance									
Suara ‘a’ vs *feature* 1	37	251	167	229	241	165	173	37	1	37
Suara ‘b’ vs *feature* 1	255	241	249	255	255	69	245	29	229	253
Suara ‘c’ vs *feature* 1	255	253	237	241	255	193	255	165	253	255
Suara ‘d’ vs *feature* 1	253	75	255	255	165	253	169	229	255	247
Suara ‘e’ vs *feature* 1	253	255	255	253	167	233	252	231	245	235

b. Percobaan input suara ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’ dibandingkan *feature* suara 2

Tabel 2 hasil perbandingan *feature* suara 2

Perbandingan	Nilai minimum distance									
Suara ‘a’ vs *feature* 2	225	41	194	225	8	229	240	195	67	233
Suara ‘b’ vs *feature* 2	161	165	253	255	241	226	241	255	136	242
Suara ‘c’ vs *feature* 2	253	255	161	225	117	132	255	132	253	179
Suara ‘d’ vs *feature* 2	161	240	249	242	239	241	225	161	201	229
Suara ‘e’ vs *feature* 2	249	253	254	229	255	239	226	245	255	225

Perbandingan	Nilai minimum distance									
Suara ‘a’ vs *feature* 2	228	113	255	132	255	185	253	243	243	255
Suara ‘b’ vs *feature* 2	35	200	33	249	225	33	245	245	243	241
Suara ‘c’ vs *feature* 2	233	225	252	245	255	255	255	243	169	249
Suara ‘d’ vs *feature* 2	255	161	245	251	225	245	249	195	255	255
Suara ‘e’ vs *feature* 2	253	255	227	245	241	225	255	245	198	220

c. Percobaan input suara ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’ dibandingkan *feature* suara 3

Tabel 3 hasil perbandingan *feature* suara 3

Perbandingan	Nilai minimum distance									
Suara ‘a’ vs *feature* 3	231	255	244	229	149	194	165	249	255	255
Suara ‘b’ vs *feature* 3	235	169	255	229	149	213	165	249	229	225
Suara ‘c’ vs *feature* 3	227	241	197	253	167	241	163	37	193	254
Suara ‘d’ vs *feature* 3	253	255	249	215	181	225	253	165	253	255
Suara ‘e’ vs *feature* 3	245	253	255	101	253	161	255	245	255	225

Perbandingan	Nilai minimum distance									
Suara ‘a’ vs *feature* 3	253	245	245	225	249	253	265	255	229	233
Suara ‘b’ vs *feature* 3	229	167	243	251	232	245	255	255	213	255
Suara ‘c’ vs *feature* 3	225	33	253	239	249	171	165	61	37	254
Suara ‘d’ vs *feature* 3	167	251	229	241	253	253	242	253	185	229
Suara ‘e’ vs *feature* 3	229	255	181	255	248	239	251	245	198	252

d. Percobaan input suara ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’ dibandingkan *feature* suara 4

Tabel 4 hasil perbandingan *feature* suara 4

Perbandingan	Nilai minimum distance									
Suara ‘a’ vs *feature* 4	255	243	251	253	241	247	226	242	227	254
Suara ‘b’ vs *feature* 4	226	255	254	253	255	249	254	195	243	119
Suara ‘c’ vs *feature* 4	210	252	255	242	198	203	255	255	193	252
Suara ‘d’ vs *feature* 4	249	98	227	271	12	241	38	49	122	231
Suara ‘e’ vs *feature* 4	251	227	255	243	227	255	131	255	195	251

Perbandingan	Nilai minimum distance									
Suara ‘a’ vs *feature* 4	255	98	195	255	143	255	251	255	245	241
Suara ‘b’ vs *feature* 4	254	251	251	189	255	255	235	249	122	255
Suara ‘c’ vs *feature* 4	249	254	136	243	255	255	255	230	255	252
Suara ‘d’ vs *feature* 4	227	195	250	17	254	99	196	235	243	249
Suara ‘e’ vs *feature* 4	255	254	201	195	234	254	251	253	255	245

e. Percobaan input suara ‘a’, ‘b’, ‘c’, ‘d’ dan ‘e’ dibandingkan *feature* suara 5

Tabel 5 hasil perbandingan *feature* suara 5

Perbandingan	Nilai minimum distance									
Suara ‘a’ vs *feature* 5	203	237	237	241	255	244	235	183	253	255
Suara ‘b’ vs *feature* 5	255	225	173	255	241	255	235	253	235	206
Suara ‘c’ vs *feature* 5	203	196	73	255	243	251	255	234	240	248
Suara ‘d’ vs *feature* 5	233	248	255	243	255	255	249	255	247	235
Suara ‘e’ vs *feature* 5	3	195	107	237	237	141	95	1	181	250

Perbandingan	Nilai minimum distance									
Suara ‘a’ vs *feature* 5	243	251	235	253	115	153	115	255	173	255
Suara ‘b’ vs *feature* 5	252	251	203	254	255	173	255	235	239	255
Suara ‘c’ vs *feature* 5	251	255	237	177	237	251	247	181	254	173
Suara ‘d’ vs *feature* 5	240	253	252	123	254	253	246	255	243	249
Suara ‘e’ vs *feature* 5	254	173	43	183	253	75	106	173	141	98