

**IMPLEMENTASI DETEKSI KOMPLEKS QRS ECG DENGAN ALGORITMA PAN-TOMPKINS PADA PERANGKAT FPGA**

**(FPGA IMPLEMENTATION OF ECG QRS COMPLEX DETECTION  
USING PAN-TOMPKINS ALGORITHM)**

**TUGAS AKHIR**

Diajukan sebagai syarat untuk memperoleh gelar  
Sarjana Teknik pada program studi Teknik Telekomunikasi  
Institut Teknologi Telkom

Oleh :

**AHMAD ZAKY RAMDANI**  
**NIM : 111070131**



**FAKULTAS ELEKTRO DAN KOMUNIKASI**  
**INSTITUT TEKNOLOGI TELKOM**  
**BANDUNG**  
**2011**

 INSTITUT TEKNOLOGI TELKOM	<b>INSTITUT TEKNOLOGI TELKOM</b> <b>Jl. Telekomunikasi No. 1 Ters. Buah Batu Bandung</b> <b>40257</b>	No. Dokumen	ITT-AK-FEK-PTT-FM-004/001
	<b>FORMULIR LEMBAR PENGESAHAN TUGAS AKHIR</b>	No. Revisi	00
		Berlaku efektif	02 Mei 2011

## **LEMBAR PENGESAHAN**

Tugas Akhir dengan judul:

### **IMPLEMENTASI DETEksi KOMPLEKS QRS ECG DENGAN ALGORITMA PAN-TOMPKINS PADA PERANGKAT FPGA**

**(FPGA IMPLEMENTATION OF ECG QRS COMPLEX DETECTION USING PAN-TOMPKINS ALGORITHM)**

Telah diperiksa dan disetujui sebagai salah satu syarat untuk memperoleh gelar  
 Sarjana Teknik pada Program Studi Teknik Telekomunikasi  
 Institut Teknologi Telkom

Oleh :

**AHMAD ZAKY RAMDANI**  
**NIM : 111070131**

Bandung, Juli 2011  
 Disetujui dan disahkan oleh :

**Pembimbing I**

**Achmad Rizal, ST., MT.**  
**NIK: 01750224-1**

**Pembimbing II**

**Iswahyudi Hidayat, ST., MT.**  
**NIK: 02770269-1**

	<b>INSTITUT TEKNOLOGI TELKOM</b>	No. Dokumen	ITT-AK-FEK-PTT-FM-001/004
	<b>Jl. Telekomunikasi No. 1 Ters. Buah Batu Bandung 40257</b>	No. Revisi	00
	<b>FORMULIR PERNYATAAN ORISINALITAS</b>	Berlaku efektif	02 Mei 2011

## **HALAMAN PERNYATAAN ORISINALITAS**

NAMA : AHMAD ZAKY RAMDANI

NIM : 111070131

ALAMAT : Jl. Raflesia 162, Cibaduyut, Kab Bandung

NO TELPON : 085659191905

EMAIL : ahmadzakyramdani@gmail.com

Menyatakan bahwa Tugas Akhir ini merupakan karya orisinal saya sendiri, dengan judul :

### **IMPLEMENTASI DETEksi KOMPLEKS QRS ECG DENGAN ALGORITMA PAN-TOMPKINS PADA PERANGKAT FPGA**

**(FPGA IMPLEMENTATION OF ECG QRS COMPLEX DETECTION USING  
PAN-TOMPKINS ALGORITHM)**

Atas pernyataan ini, saya siap menanggung resiko / sanksi yang dijatuhan kepada saya apabila kemudian ditemukan adanya pelanggaran terhadap kejujuran akademik atau etika keilmuan dalam karya ini, atau ditemukan bukti yang menunjukan ketidak aslian karya ini.



Bandung, 27 Juli 2011

AHMAD ZAKY RAMDANI

111070131

## **HALAMAN PERSEMBAHAN**

*Dan mintalah pertolongan (kepada Allah) dengan sabar dan sholat.....*

(QS Al Baqoroh : 45)

*Boleh jadi kamu membenci sesuatu padahal ia amat baik bagimu, dan boleh jadi pula kamu menyukai sesuatu padahal ia amat buruk bagimu, Allah mengetahui sedang kamu tidak mengetahui.*

(QS Al Baqarah : 216)

*Sesungguhnya Aku ini adalah Allah, tidak ada Tuhan (yang hak) selain Aku, maka sembahlah Aku dan dirikanlah shalat untuk mengingat Aku.*

(Thaa Haa : 14)

*Segungguhnya hari kiamat itu akan datang.Aku merahasian (waktunya) agar supaya tiap-tiap diri itu dibalas dengan apa yang ia usahakan.*

(Thaa Haa : 15)

*Kami akan memperlihatkan kepada mereka tanda-tanda (kebesaran) Kami di segenap penjuru dan pada diri mereka sendiri, sehingga jelaslah bagi mereka bahwa Al-Qur'an itu adalah benar. Tidak cukupkah (bagi kamu) bahwa Tuhanmu menjadi saksi atas segala sesuatu ?*

(Fussilat : 53)

*Kupersembahkan karya kecil ini untuk ALLAH SWT.*

*Untuk kedua orang tuaku, kakak-kakakku, saudara – saudaraku  
dan anak istriku kelak.*

*Semoga karya kecil ini dapat menjadi pemberat amalku  
dan juga berguna untuk kebaikan.*

*Sekaligus menjadi jalan manfaat... Amin.*

## ABSTRAK

Serangan dan gangguan jantung merupakan penyebab kematian nomor satu di dunia. Dalam menanggulangi terjadinya serangan serta gangguan pada jantung, maka *monitoring* terhadap kondisi jantung sangatlah penting. ECG merupakan sinyal hasil aktivitas kelistrikan jantung yang dapat memberikan informasi kondisi fisik dari pasien dan dapat mengindikasikan sebuah penyakit. Suatu sinyal ECG memiliki komponen utama berupa kompleks QRS. Sehingga pendekripsi nilai kompleks QRS memegang peranan yang sangat penting pada sistem pengolahan sinyal ECG. Salah satu metode yang menjadi referensi untuk perhitungan sebuah kompleks QRS secara *real time* adalah dengan metode Pan-Tompkins.

Tujuan penelitian ini adalah untuk membuat sebuah *hardware* yang diimplementasikan untuk menghitung nilai dari QRS kompleks. *Hardware* yang akan direalisasikan mengambil masukan berupa sinyal ECG *realtime* yang telah mengalami konversi menjadi sinyal digital melalui perangkat *Analog to Digital Converter* (ADC). Operasi pada sinyal digital keluaran ADC adalah penghitungan nilai kompleks QRS dengan bantuan FPGA yang akan menjalankan sistem sesuai dengan metode Pan-Tompkins. Penggunaan FPGA digunakan karena perangkatnya yang relatif lebih murah dan mudah untuk dimodifikasi dibandingkan dengan perangkat yang biasa digunakan saat ini.

Pada metode yang akan digunakan ini sinyal kompleks ECG akan mengalami beberapa blok pengolahan, yaitu blok *Bandpass filter*, blok Differensiasi, blok *Squaring Operation* (pengkuadratan), blok integrasi, blok terakhir yaitu *Thresholding*. Blok-blok tersebut diimplementasikan pada perangkat FPGA sebagai sistem operasi perhitungan yang terdiri dari blok logika *adder*, *resister*, *address loader*, *register*, *memory* serta *control unit*. Hasil implementasi sistem ini didapatkan perangkat pendekripsi kompleks QRS dengan tingkat ketepatan sebesar 92% dengan total waktu pemprosesan sebesar 245 ns untuk clock 2 ms.

Kata kunci: *Electrocardiograf*, Pan-Tompkins, QRS kompleks, ADC, FPGA

## ***ABSTRACT***

*Heart attack is number one cause of human dead in the world. In purpose of controlling the number of death caused by heart attack monitoring the condition of heart is very important. ECG is an interpretation of the electrical activity of the heart over time captured and externally recorded that can detect the condition of heart and trouble on it. An ECG signal has QRS complex as its main component. So, detecting the value of QRS complex is very important in the way to get the information of heart conditions. Pan-Tompkins method is the most widely common use in hardware implementation of QRS complex detector.*

*This research's goal is to implement the hardware that can be used in detecting the value of QRS complex. The future implemented hardware will be supplied by ECG signal that has been converted to digital signal before by an Analog to Digital Converter (ADC). Operation to ADC output signal will be handled by programmed FPGA that runs the operating as the Pan-Tompkins method. This research is implemented in FPGA due to its flexibility in programmed and also the price is relatively cheaper than other hardwares that commonly use nowadays.*

*In Pan-Tompkins method the implemented system will consist of several blocks, they are band pass filter block, derivation block, squaring operation block, integration block and decision block. The blocks will be implemented on FPGA as calculating operation blocks system that consists of adder, register, address loader, memory and also control unit.*

*As the result, this project successfully produces QRS detection system with 92% accuracy and 245 ms processing time delay within clock of 5 ms.*

***Keywords:*** *Electrocardiograph, Pan-Tompkins, complex QRS, ADC, FPGA*

## **KATA PENGANTAR**



*Assalaamualaikum Warahmatulloh Wabarakatuh*

Alhamdulillah, puji sukur penulis panjatkan kehadirat Allah SWT Tuhan semesta alam yang Maha Mengetahui lagi Maha Menguasai Ilmu Pengetahuan, yang telah memberikan hidayah, rahmat, inayah serta limpahan berkah-Nya pada penulis sehingga dapat menyelesaikan Tugas Akhir yang berjudul : **“IMPLEMENTASI DETEKSI KOMPLEKS QRS ECG DENGAN ALGORITMA PAN-TOMPKIN PADA PERANGKAT FPGA”**, yang disusun sebagai persyaratan menempuh sidang Tugas akhir pada Program Studi Sarjana Teknik Telekomunikasi Institut Teknologi Telkom Bandung.

Perancangan yang dilakukan dalam tugas akhir ini adalah merancang *prototype* sistem DETEKSI KOMPLEKS QRS menggunakan pengkodean VHDL dan diimplementasikan pada FPGA (*Field Programmable Gate Array*).

Tugas akhir ini tentu masih jauh dari sempurna. Maka dari itu penulis mengharapkan kritik dan saran dari seluruh pihak agar tercapainya sasaran, tujuan serta manfaat terselesaikannya Tugas Akhir ini. Saran dan kritik dapat didiskusikan melalui email penulis [ahmadzakyramdani@gmail.com](mailto:ahmadzakyramdani@gmail.com)

Akhirnya semoga Tugas Akhir ini bisa memberikan manfaat kepada semua pihak.

*Wassalamuallaikum Warahmatulloh Wabarakatuh*

Bandung, Juli 2011

**Penulis**

## **UCAPAN TERIMA KASIH**

Penulis menyadari bahwa selama menempuh pendidikan di Institut Teknologi Telkom, khususnya selama penyusunan Tugas Akhir ini, penulis mendapat banyak bantuan, perhatian, do'a, masukan dan dorongan dari banyak pihak. Oleh karena itu, pada kesempatan ini penulis mengucapkan terima kasih, kepada :

1. Allah SWT, Dzat Maha Pengasih lagi Maha Penyayang, Tuhan semesta alam yang senantiasa memberikan yang terbaik kepada hamba-Nya.
2. Bapak dan Ibuku yang selalu memberi nasehat, do'a dan banyak perhatian. Terima kasih, biarlah Allah yang membalas semua kebaikan bapak dan ibu.
3. Keluargaku, kakak-kakak serta keponakan kecilku yang selalu memangat semangat dan insirasi tersendiri.
4. Bapak Achmad Rizal, ST. MT. dan Bapak Iswahyudi Hidayat, ST. MT yang telah sabar menjadi pembimbing, mengarahkan serta memberikan motivasi untuk mengerjakan tugas akhir ini, sehingga dapat selesai sesuai target atau tujuan yang diinginkan. Terima kasih banyak dan saya minta maaf jika selama mengerjakan TA sering berbuat salah.
5. Mas Sugondo Hadyoso yang telah banyak membantu dalam penggerjaan Tugas Akhir ini, semoga sukses selalu mas terima kasih yang sangat luar biasa.
6. Bpk Deny Darlis yang telah memberikan masukan-masukan serta arahan dalam tugas akhir ini.
7. Guru-guru terbaik yang banyak memberikan inspirasi hingga saya dapat melangkah sejauh ini, Ibu Anna, Ibu Neneng serta seluruh guru SDN Moh Toha Bandung, Bpk Kundrat, Bpk Hamid, Ibu Siti Ghatijah serta seluruh guru SMPN 5 Bandung, Ibu Dini, serta seluruh guru SMAN 8 Bandung, Bpk Bambang Krisnarno, Bpk Heroe Widjanto, Bpk Rendi Munadi serta seluruh jajaran dosen IT Telkom.
8. Ratih Aflita Rahmawati yang selalu menjadi sahabat terbaik, teman berbagi suka maupun duka dan selalu memberikan semangat tersendiri.

9. Keluarga besar Swara Waditra Sunda, Panata Calagara Pagelaran 2009, 2010, 2011. Acha, Dennis, Esti, Tasha, Depi, Iman, Iyan, Luthfy, Yana, Wildan, Gagah, Arfi, Tria, Mochi, Arif dan angkatan 2008 lainnya. Rayi, Rina, Bebi, Imam, Fathia, Hani, Hanna, Dea, Dina, Dine, Audy, Ghaliza, Cakra, Faris, dan angkatan 2009 lainnya kecuali Kepin. Yeye, Johan, Farina, Ijah, Syifa, Anggi, Dio dan angkatan 2010 lainnya serta para sesepuh 2006 dan angkatan sebelumnya yang telah mendahului kita semua. Hatur nuhun untuk kebersamaan dan kekeluargannya selama ini.
10. Keluarga besar Laboratorium Teknik digital yang telah memberikan sebuah arti kebersamaan dan kekeluargaan. Kepada saudara-saudaraku tekdig 07 terima kasih banyak. Untuk kakak-kakak angkatan 2004, 2005, 2006 serta adik-adikku tekdig 08 dan 09 terima kasih banyak buat semua.
11. Temen-temen lab C.201 yang sudah sekian lama berbagi ruangan. Untuk semuanya terima kasih banyak.
12. Sahabat-sahabat luar biasa yang sangat membantu dalam logistik penggerjaan Tugas Akhir ini, Acis, Tata, Deti, Okky, Ai, Meitha, Wahyu, Firmansyah Lukman, Isal, Tirta, Rendi dan masih banyak lagi. Semoga kalian sukses selalu.
13. Keluarga besar teman-teman sepermainan Bandung 2007, Senna, Teguh, Isan, Widodo, Wavid, Ryan, Ivan, Kukuh, Reza, Yudit, Dani, Rizky, Rizky, Faisal, Revrian, Mbak Ira, Shendy, Mbak Irna, Syahna, Annisa, Fera dan teman-teman lainnya.
14. Temen-temen seperjuangan anak-anak TT-31-02 terima kasih banyak, semoga kalian sukses semua. Amien.
15. Dan semua yang telah membantu dalam penyusunan Tugas Akhir ini.

Semoga Allah SWT membalas budi baik semua pihak yang telah membantu dalam penyelesaian Tugas Akhir ini. Amin ya Robbal ‘Alamin.

## DAFTAR ISI

Halaman

### **LEMBAR PENGESAHAN**

### **LEMBAR PERNYATAAN ORISINALITAS**

### **LEMBAR PERSEMBAHAN**

<b>ABSTRAK .....</b>	<b>i</b>
<b>ABSTRACT .....</b>	<b>ii</b>
<b>KATA PENGANTAR.....</b>	<b>iii</b>
<b>UCAPAN TERIMAKASIH .....</b>	<b>iv</b>
<b>DAFTAR ISI.....</b>	<b>vi</b>
<b>DAFTAR GAMBAR.....</b>	<b>ix</b>
<b>DAFTAR TABEL .....</b>	<b>xii</b>
<b>DAFTAR SINGKATAN.....</b>	<b>xiii</b>
<b>DAFTAR ISTILAH .....</b>	<b>xiv</b>
<b>DAFTAR LAMPIRAN .....</b>	<b>xv</b>

### **BAB I PENDAHULUAN**

1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	2
1.3 Tujuan.....	2
1.4 Batasan Masalah.....	3
1.5 Metode Penelitian.....	3
1.6 Sistematika Penulisan.....	4

### **BAB II DASAR TEORI**

2.1 Elektrocardiogram (ECG).....	5
2.1.1 Kompleks QRS.....	5
2.2 Algoritma Pan-Tompkins.....	8
2.2.1 Filtering Sinyal.....	9
2.2.2 Differensiasi.....	12
2.2.1 Pengkuadratan.....	12
2.2.2 Integrasi.....	13
2.2.1 Threshold.....	13

2.3 FPGA (Field Programmable Gate Array) .....	13
2.4 VHSIC Hardware Description Language (VHDL).....	14
2.5 Analog to Digital Converter (ADC).....	15

### **BAB III PERANCANGAN SISTEM DETEKSI KOMPLEKS QRS**

3.1 Diagram Alir Perancangan dan Implementasi Deteksi Kompleks QRS .....	16
3.2 Model dan Parameter Sistem Deteksi QRS Algoritma Pan-Tompkins.....	17
3.3 Arsitektur Sistem pada VHDL.....	17
3.3.1 Blok Sistem Low Pass Filter .....	19
3.3.2 Blok Sistem High Pass Filter .....	23
3.3.3 Blok Sistem Derivatif.....	25
3.3.4 Blok Sistem Fungsi Kuadrat.....	26
3.3.5 Blok Sistem Integrasi .....	27
3.3.6 Blok Sistem Decision .....	28
3.3.7 Integrasi Keseluruhan Blok Sistem .....	28
3.4 Implementasi Hardware Sistem Deteksi QRS.....	29
3.4.1 Konversi ECG ke Sinyal Digital.....	29
3.4.2 Monitoring Keluaran Sinyal.....	29
3.5 Sistem Pengambilan Data.....	29
3.6 Performansi Sistem.....	29
3.7 Perangkat Lunak yang Digunakan.....	30

### **BAB IV PENGUJIAN DAN ANALISA SISTEM DETEKSI QRS**

4.1 Perancangan Sinyal Test Bench .....	31
4.2 Pengujian Blok-Blok Sistem pada ModelSim.....	32
4.2.1 Pengujian Blok Sistem LPF.....	32
4.2.2 Pengujian Blok Sistem HPF.....	34
4.2.3 Pengujian Blok Sistem Derivatif.....	36
4.2.4 Pengujian Blok Sistem Pengkuadrat.....	37
4.2.5 Pengujian Blok Sistem Integrator.....	38
4.2.6 Pengujian Blok Sistem Decision.....	40
4.3 Analisa Sistem Keseluruhan.....	40

## **BAB V IMPLEMENTASI PERANGKAT KERAS SISTEM DETEKSI QRS PADA FPGA**

5.1 Perancangan Keras Virtex-4 FPGA XC4VLX25-FF668 .....	47
5.2 Implementasi Sistem Deteksi QRS.....	48
5.2.1 Design Entry.....	49
5.2.2 Assigned Package PIN.....	49
5.2.3 Sintesis Rangkaian.....	50
5.3 Pengujian Perangkat Keras .....	57

## **BAB VI KESIMPULAN DAN SARAN**

6.1 Kesimpulan.....	62
6.2 Saran.....	62

## **DAFTAR PUSTAKA .....** xvi

## **LAMPIRAN**

## DAFTAR GAMBAR

Gambar 2.1 Spektrum Frekuensi sinyal ECG .....	6
Gambar 2.2 Sinyal Kompleks QRS .....	7
Gambar 2.3 Skema Pengolahan Sinyal ECG pada Algoritma Pan-Tompkins.....	8
Gambar 2.4 Respon magnitude low pass filter dengan frekuensi sampling 200 Hz	9
Gambar 2.5 Respon fasa dari low pass filter .....	10
Gambar 2.6 Plot pole dan zero low pass filter .....	10
Gambar 2.7 Respon magnitude high pass filter .....	11
Gambar 2.8 Respon Amplitud dan Respon Fasa High Pass Filter .....	12
Gambar 2.9 Arsitektur bagian dalam FPGA.....	14
Gambar 2.10 Proses Pensemplingan Sinyal Analog Menjadi Sinyal Diskrit .....	16
Gambar 2.11 Proses Kuantisasi Sinyal Diskrit .....	16
Gambar 3.1 Diagram alir perancangan dan implementasi sistem deteksi QRS .....	18
Gambar 3.2 Blok sistem deteksi QRS pada algoritma Pan-Tompkins .....	19
Gambar 3.3 Realisasi matematika <i>low pass filter</i> yang digunakan .....	20
Gambar 3.4 Blok program implementasi <i>low pass filter</i> pada VHDL .....	20
Gambar 3.5 Blok program <i>register</i> pada <i>low pass filter</i> .....	21
Gambar 3.6 Blok program <i>multiplier</i> pada <i>low pass filter</i> .....	22
Gambar 3.7 Skema penjumlahan pada blok <i>adder low pass filter</i> .....	22
Gambar 3.8 Realisasi matematika <i>high pass filter</i> yang digunakan .....	23
Gambar 3.9 Blok program implementasi <i>high pass filter</i> pada VHDL .. ....	24
Gambar 3.10 Blok program <i>multiplier</i> pada <i>high pass filter</i> .....	25
Gambar 3.11 Blok program implementasi sistem <i>derivative</i> pada VHDL .....	25
Gambar 3.12 Blok program divider pembangian dengan 8 .....	26
Gambar 3.13 Blok program pengkuadrat.....	27
Gambar 3.14 Blok program integrator.....	28
Gambar 3.15 Integrasi blok-blok sistem pada VHDL .....	29
Gambar 4.1 Perbandingan sinyal asli dengan sinyal test bench .....	31
Gambar 4.2 Hasil pengujian banual blok LPF.....	32
Gambar 4.3 Test bech keluaran blok sistem LPF .....	33
Gambar 4.4 Hasil pengujian manual blok HPF .....	34
Gambar 4.5 Test bench keluaran sistem HPF.....	35

Gambar 4.6 Hasil pengujian manual blok derivative.....	36
Gambar 4.7 Test bench keluaran blok sistem derivative ..	37
Gambar 4.8 Hasil pengujian manual blok pengkuadrat.....	37
Gambar 4.9 Test bench keluaran blok sistem fungsi kuadrat .....	38
Gambar 4.10 Hasil pengujian manual blok integrator .....	38
Gambar 4.11 Test bench keluaran blok sistem integrator.....	40
Gambar 4.12 Test bench keluaran blok sistem decision.....	40
Gambar 4.13 Perbandingan keluaran untuk setiap blok sistem .....	41
Gambar 4.14 Hasil pengujian sinyal ECG normal 1.....	42
Gambar 4.15 Hasil pengujian sinyal ECG normal 2.....	42
Gambar 4.16 Hasil pengujian sinyal ECG arrhythmia 1 .....	42
Gambar 4.17 Hasil pengujian sinyal ECG arrhythmia 2 .....	42
Gambar 4.18 Hasil pengujian sinyal ECG arrhythmia 3 .....	42
Gambar 4.19 Hasil pengujian sinyal ECG arrhythmia 4 .....	42
Gambar 4.20 Hasil pengujian sinyal ECG arrhythmia 5 .....	42
Gambar 4.21 Hasil pengujian sinyal ECG arrhythmia 6 .....	43
Gambar 4.22 Hasil pengujian sinyal ECG arrhythmia 7 .....	43
Gambar 4.23 Hasil pengujian sinyal ECG arrhythmia 8 .....	43
Gambar 4.24 Hasil pengujian sinyal ECG arrhythmia 9 .....	43
Gambar 4.25 Hasil pengujian sinyal ECG arrhythmia 10 .....	43
Gambar 4.26 Hasil pengujian sinyal ECG arrhythmia 11 .....	43
Gambar 4.27 Hasil pengujian sinyal ECG arrhythmia 12 .....	43
Gambar 4.28 Hasil pengujian sinyal ECG arrhythmia 13 .....	43
Gambar 4.29 Hasil pengujian sinyal ECG arrhythmia 14 .....	44
Gambar 4.30 Hasil pengujian sinyal ECG arrhythmia 15 .....	44
Gambar 4.31 Hasil pengujian sinyal ECG arrhythmia 16 .....	44
Gambar 4.32 Hasil pengujian sinyal ECG arrhythmia 17 .....	44
Gambar 4.33 Hasil pengujian sinyal ECG arrhythmia 18 .....	44
Gambar 4.34 Hasil pengujian sinyal ECG arrhythmia 19 .....	44
Gambar 4.35 Hasil pengujian sinyal ECG arrhythmia 20 .....	44
Gambar 4.36 Hasil pengujian sinyal ECG arrhythmia 21 .....	44
Gambar 4.37 Hasil pengujian sinyal ECG arrhythmia 22 .....	44
Gambar 4.38 Hasil pengujian sinyal ECG arrhythmia 23 .....	44

Gambar 4.39 Hasil pengujian sinyal ECG arrhythmia 24 .....	45
Gambar 4.40 Hasil pengujian sinyal ECG arrhythmia 25 .....	45
Gambar 4.41 Hasil pengujian sinyal ECG arrhythmia 26 .....	45
Gambar 4.42 Hasil pengujian sinyal ECG arrhythmia 27 .....	45
Gambar 4.43 Hasil pengujian sinyal ECG arrhythmia 28 .....	45
Gambar 5.1 Perangkat Keras Virtex-4 FPGA XC4VLX25 .....	47
Gambar 5.2 Diagram Alir implementasi Rangkaian pada FPGA .....	48
Gambar 5.3 Design Entry Rangkaian .....	49
Gambar 5.4 Tampilan <i>package</i> dari pin-pin masukan dan keluaran pada FPGA...	49
Gambar 5.5 Tampilan blok sistem deteksi QRS hasil sintesa .....	50
Gambar 5.6 Tampilan blok inti sistem deteksi QRS.....	50
Gambar 5.7 Tampilan blok LPF sistem deteksi QRS .....	51
Gambar 5.8 Perbandingan masukan dan keluaran implementasi blok LPF .....	51
Gambar 5.9 Tampilan blok HPF sistem deteksi QRS.....	52
Gambar 5.10 Perbandingan masukan dan keluaran implementasi blok HPF .....	52
Gambar 5.11 Tampilan blok derivatif sistem deteksi QRS .....	53
Gambar 5.12 Perbandingan masukan dan keluaran implementasi blok derivative	53
Gambar 5.13 Tampilan blok pengkuadrat sistem deteksi QRS .....	54
Gambar 5.14 Perbandingan masukan dan keluaran implementasi blok pengkuadrat	54
Gambar 5.15 Tampilan blok integrasi sistem deteksi QRS .....	55
Gambar 5.16 Perbandingan masukan dan keluaran implementasi blok integrasi ..	55
Gambar 5.17 Tampilan blok decision sistem deteksi QRS.....	56
Gambar 5.18 Skenario pengujian implementasi sistem deteksi kompleks QRS ....	58
Gambar 5.19 Pengujian perangakat sistem deteksi kompleks QRS .....	58
Gambar 5.20 Hasil pengujian implemntasi deteksi QRS sinyal ECG normal.....	59
Gambar 5.21 Hasil pengujian implemntasi deteksi QRS sinyal ECG A. Fibrillation	59
Gambar 5.22 Hasil pengujian implemntasi deteksi QRS sinyal ECG 2° BLK 1....	60
Gambar 5.23 Hasil pengujian implemntasi deteksi QRS sinyal ECG RBBB .....	60
Gambar 5.24 Hasil pengujian implemntasi deteksi QRS sinyal ECG PAC .....	60

## **DAFTAR TABEL**

Tabel 4.1	Hasil Penghitungan secara Manual untuk keluaran LPF.....	33
Tabel 4.2	Hasil Penghitungan secara Manual untuk keluaran HPF.....	35
Tabel 4.3	Hasil Penghitungan secara Manual untuk keluaran Derivative.	36
Tabel 4.4	Hasil Penghitungan secara Manual untuk keluaran Pengkuadrat	38
Tabel 4.5	Hasil Penghitungan secara Manual untuk keluaran Integrator...	39
Tabel 4.6	<i>Processing Time Delay</i> yang Dihasilkan Setiap Blok Sistem.	41
Tabel 4.7	Hasil Pengujian Sistem Deteksi QRS pada ModelSim.....	46
Tabel 5.1	Penggunaan <i>Resource</i> Komponen pada FPGA .....	56
Tabel 5.2	Hasil Ringkasan Implementasi pada FPGA menggunakan Xilinx.....	57
Tabel 5.3	Hasil Sintesa Implementasi pada FPGA menggunakan Xilinx.....	57
Tabel 5.4	Hasil Pengujian dari Perangkat Sistem Deteksi QRS.....	61

## **DAFTAR SINGKATAN**

LPF	:	Low Pass Filter
HPF	:	High Pass Filter
ECG	:	Electrocardiogram
ADC	:	Analog to Digital Converter
FPGA	:	Field Programmable Gate Array
HDL	:	Hardware Description Language
IEEE	:	Institute of Electrical and Electronics Engineers
IC	:	Integrated Circuit
RBBB	:	Right Bundle Branch Block
PVC	:	Premature Ventricular Contraction
PAC	:	Premature Atrial Contraction

## **DAFTAR ISTILAH**

### ***Arrhythmia***

Ketidakstabilan detak jantung yang disebabkan oleh gangguan pada organ jantung.

### **Arsitektur**

Merupakan struktur mekanisme kerja suatu program yang dibangun dari beberapa komponen ataupun program lainnya.

### ***Bit***

*Binary Digit* yang merupakan bagian terkecil dari data digital, direpresentasikan dengan nilai '1' dan '0'.

### **CLB**

*Configurable Logic Block* merupakan kumpulan dari slice yang identik.

### ***Clock***

Sederetan pulsa detak yang terdiri dari bit '1' dan '0' yang menjadi pemicu rangkaian.

### ***Electrocardiograf***

Merupakan suatu perangkat yang digunakan untuk menangkap sinyal aktivitas kelistrikan dari jantung.

### **Filter**

Suatu perangkat yang digunakan untuk meloloskan dan membuang sinyal dengan frekuensi tertentu.

### ***Flowchart***

Skema aliran suatu program maupun sistem yang akan dijalankan.

### **FPGA**

*Programable Logic Device* (PLD) yang dibangun dari sekumpulan sel fungsi logika dasar yang dapat diprogram.

### ***Slices***

Bagian dasar dari FPGA yang tersusun dari *Look Up Table*, *Function Generator*, dan *flip-flop* dan dapat diatur sehingga dapat melakukan fungsi tertentu.

### ***Test bench***

Script tambahan yang digunakan untuk membantu proses simulasi program.

### **VHDL**

Bahasa pemrograman yang digunakan untuk mendeskripsikan *hardware* mulai dari tingkat yang abstrak hingga tingkat yang kompleks.

### ***Waveform***

Tampilan keluaran dari proses simulasi yang dilakukan dalam bentuk sinyal.

## **DAFTAR LAMPIRAN**

LAMPIRAN A	: Listing Program.....	A-1
LAMPIRAN B	: Tabel Penghitungan manual LPF dan HPF.....	B-1
LAMPIRAN C	: Data Sheet Xilinx Virtex4 XC4VLX25.....	C-1
LAMPIRAN D	: User Manual PS400 Patient Simulator.....	D-1

## **BAB I**

### **PENDAHULUAN**

#### **1.1 Latar belakang**

Biomedis merupakan salah satu cabang ilmu elektronika yang mempelajari kaitan antara ilmu elektro dengan ilmu medis, sehingga ikut membantu perkembangan teknologi di bidang medis saat ini. Salah satu teknologi yang berhasil dikembangkan oleh dunia biomedis adalah sinyal *electrocardiograf* (ECG). ECG merupakan suatu sinyal yang dihasilkan dari aktivitas kelistrikan dari jantung manusia yang memiliki informasi medik mengenai kondisi kesehatan manusia yang bersangkutan.

Suatu sinyal ECG memiliki komponen utama berupa kompleks QRS. Kompleks QRS adalah bentuk umum dari sinyal ECG yang normal dan berhubungan dengan deplarisasi ventrikel. Karena ventrikel mengandung lebih banyak massa otot daripada atrium, kompleks QRS lebih besar daripada gelombang P. Dengan demikian deteksi QRS dapat mendeteksi kelainan frekuensi,keteraturan, tempat asal atau konduksi impuls listrik pada jantung.

Sehingga pendektsian nilai kompleks QRS memegang peranan yang sangat penting pada sistem pengolahan sinyal ECG. Salah satu metode yang dapat digunakan untuk perhitungan sebuah kompleks QRS secara *real time* adalah dengan metode Pan-Tompkins. Metode yang diperkenalkan oleh Jianpu Pan dan Willi Tompkins pada 1987 ini memiliki tingkat ketepatan deteksi hingga 98% dan merupakan metode yang paling umum digunakan pada implementasi perangkat keras sistem pendeksi QRS realtime. Selain metode ini terdapat pula metode lainnya seperti wavelet dan metode yang diperkenalkan Timo Bragge, Mika P. Tarvainen, and Pasi A. Karjalainen.

Tujuan dari penelitian ini adalah untuk mendapatkan sebuah *hardware* yang diimplementasikan untuk menghitung nilai dari QRS kompleks. Penggunaan FPGA (*Field Programmable Gate Array*) digunakan karena perangkatnya yang relatif lebih murah dan mudah untuk dimodiifikasi dibandingkan dengan perangkat yang biasa digunakan saat ini.

Perangkat yang akan dirancang merupakan penggabungan dari tiga buah blok perangkat yang terdiri dari ECG analog yang berfungsi untuk mengambil sinyal ECG dari penyadapan pada tubuh manusia. Blok selanjutnya adalah ADC yang berfungsi untuk mengubah sinyal ECG hasil penyadapan menjadi urutan bit digital. Tahap paling akhir dari hardware ini adalah pengolahan sinyal ECG untuk deteksi kompleks QRS dengan perangkat FPGA yang mengambil input bit biner hasil keluaran ADC.

Pada tugas akhir ini, sistem pendekripsi QRS diimplementasikan pada perangkat FPGA dengan membangun arsitektur sistem yang dapat mewakili kinerja dari sistem secara optimal. Pembangunan arsitektur sistem ini didefinisikan dengan bahasa pemrograman VHDL (*VHSIC Hardware Description Language*) dengan perangkat implementasi FPGA (Field Programmable Gate Array) Virtex 4 XC4VLX. Masukan sistem ini adalah keluaran konversi 8 bit dari sinyal ECG analog. Sinyal ECG yang diujikan berasal dari perangkat PS400 yang merupakan pembangkit sinyal sebagai pengganti tubuh manusia.

## **1.2 Rumusan masalah**

Sistem deteksi QRS dapat digunakan sebagai pendekripsi gangguan jantung terutama yang menjadikan keteraturan dan frekuensi dari detak jantung sebagai parameternya. Dengan perancangan sistem deteksi pada perangkat FPGA akan memudahkan pengembangan sistem selanjutnya karena kompatibilitas FPGA yang tinggi. Selain itu implementasi sistem pada FPGA akan memberikan keuntungan secara ekonomi apabila akan dilakukan industrialisasi perangkat. Namun dengan pendefinisian sistem pada bahasa VHDL maka membutuhkan arsitektur sistem yang berbasiskan pada komponen logika. Secara garis besar, rumusan masalah pada tugas akhir ini adalah:

- a. Bagaimana cara untuk melakukan perhitungan terhadap nilai kompleks QRS ECG pada masing jenis kondisi jantung.
- b. Bagaimana proses pengimplementasian sistem perhitungan nilai kompleks QRS dengan pembangunan arsitektur pada bahasa VHDL yang pendefinisiannya berdasar pada komponen logika.
- c. Bagaimana performansi sistem ditinjau dari ketepatan sistem dalam mendekripsi nilai kompleks QRS serta *processing time* dari sistem dimulai dari mulai kemunculan sinyal hingga dapat didekripsi sistem.

## **1.3 Tujuan**

Tugas akhir ini bertujuan untuk mengimplementasikan sistem pendekripsi kompleks QRS secara real time yang memiliki kinerja yang baik sehingga layak untuk dipergunakan. Kelayakan sistem pada hal ini ditinjau dari parameter kemampuan sistem untuk mendekripsi secara tepat serta waktu yang dibutuhkan untuk sistem melakukan pendekripsi karena sistem yang dirancang untuk keperluan penganalisaan real time. Secara kualitas, sistem ini diharapkan dapat mendekripsi pada berbagai macam kondisi jantung. Secara garis besar Tujuan dari tugas akhir ini adalah :

- a. Merancang suatu perangkat keras pendekripsi kompleks QRS pada sinyal ECG.

- b. Merealisasikan sistem blok operasi Pan-Tompkins pada perangkat FPGA yang memiliki kecepatan dan akurasi yang tinggi dengan delay yang rendah.
- c. Menganalisa performansi sistem dengan parameter tingkat akurasi pada berbagai kondisi jantung.

#### **1.4 Batasan Masalah**

Batasan dari masalah yang akan dibahas dalam tugas akhir ini adalah:

- a. Sinyal ECG yang menjadi inputan sistem deteksi QRS merupakan hasil konversi ADC beresolusi 8 bit.
- b. Pemrosesan sinyal pada perangkat FPGA digunakan data dengan lebar 10 bit.
- c. Implementasi hardware menggunakan Xilinx Virtex4 Xc4vlx25
- d. Sistem diimplementasikan bersifat real time dengan menggunakan algoritma Pan-Tompkins dengan frekuensi sampling 200 Hz.
- e. Masukan dari ECG analog yang diujikan berasal dari *Patient Simulator PS400*.
- f. Untuk tingkat akurasi, pengujian dilakukan pada program VHDL yang dirancang dengan masukan hasil sampling ECG dari [www.physionet.org](http://www.physionet.org).
- g. Parameter yang diujikan hanya tingkat akurasi dan waktu proses yang dibutuhkan sedangkan untuk tingkat presisi tidak dilakukan.

#### **1.5 Metodologi Penelitian**

- a. Identifikasi masalah

Pada tahap identifikasi ini ditentukan latar belakang masalah yang mendasari penelitian ini, tujuan dilakukan penelitian, serta rumusan dan batasan masalah.

- b. Studi Literatur

Melakukan studi literatur serta pengumpulan data tentang beberapa materi yang berkaitan dengan implementasi *hardware* ini, seperti karakteristik kompleks QRS, *signal processing*, FPGA dan materi yang berkaitan lainnya. Studi literatur dilakukan melalui internet, makalah-makalah, buku-buku, serta melalui diskusi dan konsultasi dengan dosen pembimbing.

- c. Analisis Sistem

Menganalisa deskripsi dan kebutuhan sistem berdasarkan batasan masalah dan ketersediaan data.

d. Desain

Pada tahap ini, penulis melakukan pemodelan sistem pendekripsi nilai kompleks QRS.

e. Implementasi

Mengimplementasi sebuah aplikasi perangkat keras yang mampu mendekripsi nilai kompleks QRS hasil keluaran ADC melalui metode Pan-Tompkins.

f. Pengujian

Menguji sistem untuk melihat performansi kerja perangkat tersebut, evaluasi keberhasilan metode dan menganalisa faktor-faktor yang mempengaruhi performansinya.

g. Penyusunan laporan

Dilakukan analisa hasil implementasi dan pengujian sistem yang telah dilakukan dan kemudian disusun ke dalam sebuah laporan tugas akhir yang dapat dipergunakan untuk media penyebarluasan pengetahuan dan untuk kepentingan pengembangan lebih lanjut.

## **1.6 Sistematika Penulisan**

Secara umum keseluruhan Tugas Akhir ini dibagi menjadi enam bab bahasan, ditambah dengan lampiran dan daftar istilah yang diperlukan. Penjelasan masing - masing bab adalah sebagai berikut:

### **BAB 1 : PENDAHULUAN**

Bab ini berisi gambaran umum dari percobaan yang dilakukan. Tercakup di dalamnya yaitu latar belakang, perumusan masalah, tujuan, batasan masalah, metode penelitian serta sistematika penulisan.

### **BAB 2 : DASAR TEORI**

Pada bab ini berisi paparan umum mengenai karakteristik sinyal ECG dan kompleks QRSSnya, algoritma pendekripsi yang digunakan serta permasalahannya termasuk di dalamnya dasar-dasar teori yang digunakan pada proses pengimplementasian sistem ini.

### **BAB 3 : PERANCANGAN SISTEM DETEKSI KOMPLEKS QRS**

Bab ini membahas mengenai spesifikasi sistem deteksi kompleks QRS dan model sistem yang dirancang. Tahap perancangan untuk masing-masing blok sistem dan integrasi keseluruhan blok.

### **BAB 4 : PENGUJIAN DAN ANALISA SISTEM DETEKSI KOMPLEKS QRS**

---

**BAB I : Pendahuluan**

---

Pada bab ini akan dilakukan pengujian pada masing-masing blok sistem yang telah dirancang pada bahasa VHDL. Sistem diujikan untuk tiap-tiap bloknya untuk kemudian dianalisa mengenai kesesuaian dari keluaran sistem dengan keluaran yang diharapkan. Selanjutnya dilakuakn pula pengujian ketepatan sistem mendeteksi kompleks QRS pada beberapa masukan ECG.

**BAB 5 : IMPLEMENTASI PERANGKAT KERAS SISTEM DETEKSI QRS PADA FPGA**

Bab ini dilakuakn implementasi sistem pada perangkat FPGA serta dilakukan pengujian dari sistem secara realtime.

**BAB 6 : KESIMPULAN DAN SARAN**

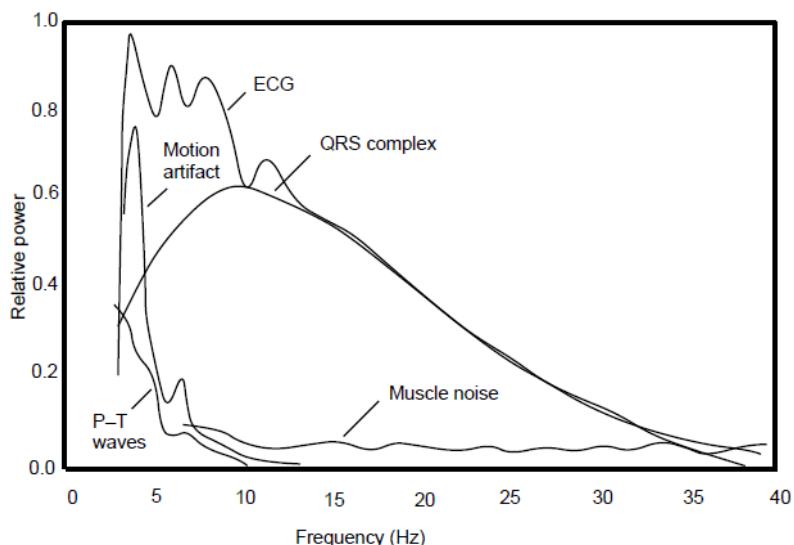
Pada bab ini merupakan bab terakhir dari laporan tugas akhir yaitu berupa kesimpulan untuk sistem yang penulis kerjakan, serta saran untuk pengembangan sistem ini selajutnya.

## BAB II

### DASAR TEORI

#### 2.1 Electrocardiogram (ECG)

*Electrocardiogram (ECG)* merupakan sinyal yang dihasilkan dari aktivitas kelistrikan jantung dengan mengukur perbedaan biopotensial dari bagian luar tubuh. ECG merupakan sinyal AC dengan bandwidth antara 0.05 Hz hingga 100 Hz. Analisis sejumlah gelombang dan vektor normal depolarisasi dan repolarisasi menghasilkan informasi diagnostik yang penting. Elektrokardiogram tidak menilai kontraktilitas jantung secara langsung. Namun, ECG dapat memberikan indikasi menyeluruh atas naik-turunnya suatu kontraktilitas.



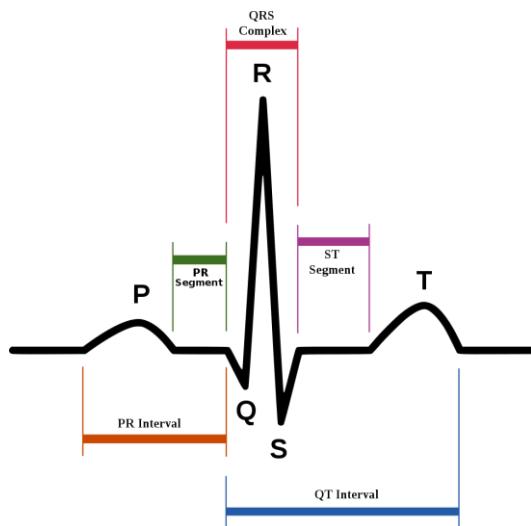
Gambar 2.1 Spektrum frekuensi sinyal ECG <sup>[2]</sup>

Sinyal ECG normal seperti pada gambar 2.2, terdiri dari sebuah gelombang P, gelombang QRS dan gelombang T. Besar amplitudo dari Sinyal ECG bervariasi tergantung pada pemasangan elektroda dan pada kondisi fisik dari pasien. Variabel-varibel klinis yang penting dari sinyal ECG antara lain magnitudo, polaritas dan durasi waktu. Variasi dari tanda-tanda tersebut dapat mengindikasikan sebuah penyakit.

#### 2.1.1 Kompleks QRS

Diagnosis dari sinyal ECG berdasarkan kepada pengamatan karakteristik sinyal ECG itu sendiri. Kompleks QRS merupakan bagian sinyal yang paling memberikan karakteristik tersendiri pada sinyal ECG. Kompleks QRS sendiri adalah struktur ECG yang berhubungan dengan deplarisasi ventrikel. Karena ventrikel mengandung lebih banyak massa otot daripada

atrium, kompleks QRS lebih besar daripada gelombang P. Kompleks QRS yang normal berdurasi 0,06-0,10 s (60-100 ms) yang ditunjukkan dengan 3 kotak kecil atau kurang, namun setiap ketidaknormalan konduksi bisa lebih panjang, dan menyebabkan perluasan kompleks QRS.



Gambar 2.2 Sinyal Kompleks QRS<sup>[6]</sup>

Gelombang QRS yang normal memiliki periode 60 ms hingga 12 ms yang terdiri dari gelombang Q, gelombang R dan gelombang S. Gelombang R merupakan defleksi positif pertama pada gelombang QRS. Umumnya gelombang R positif di *lead I, II, V5* dan *V6*, sedangkan di *lead aVR, V1* dan *V2* biasanya hanya kecil atau tidak ada sama sekali. Gelombang Q merupakan refleksi negatif pertama pada gelombang QRS dan memiliki amplituda lebih besar daripada 1/3 tinggi gelombang R, berdurasi lebih besar daripada 0,04 s (40 ms), atau di sadapan prekordial kanan dianggap tidak normal, dan mungkin menggambarkan infark miokardium. Gelombang Q abnormal disebut gelombang Q pathologis. Gelombang S merupakan defleksi negatif sesudah gelombang R dan terdeteksi dalam pada *lead aVR* dan *V1* sedangkan dari *V2* sampai *V6* akan terlihat makin lama makin menghilang atau berkurang dalamnya.

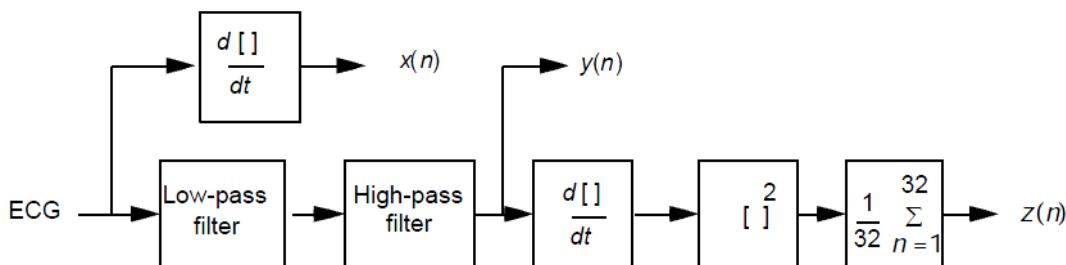
Kelainan jantung yang berkaitan langsung terhadap kestabilan kemunculan kompleks QRS adalah aritmia. Aritmia merupakan keadaan ketidak stabilan detak jantung yang disebabkan oleh penyakit otot jantung, klep jantung atau arteri koroner. Ada beberapa klasifikasi kondisi aritmia, diantaranya saja *atrial fibrillation, premature* dan *atrial contraction* yang disebabkan oleh gangguan pada bagian atrium jantung, *ventricular tachycardia, ventricular fibrillation*, dan *premature ventricular contraction* yang disebabkan

oleh gangguan ventricular jantung , serta *right bundle branch block* dan *second-degree atrioventricular block* yang terjadi karena adanya penyumbatan jantung.

## 2.2 Algoritma Pan-Tompkins

Algoritma Pan-Tompkins merupakan suatu metode untuk mendeteksi kompleks QRS secara real time yang dikembangkan oleh Jiapu Pan dan Willis J. Tompkins yang dipublikasikan pada 1985. Gelombang QRS adalah bentuk gelombang yang muncul pada kebanyakan sinyal ECG. Ada banyak teknik untuk mendeteksi gelombang QRS pada elektrokardiograf. Keakuratan dari pendektsian puncak R adalah persyaratan untuk fungsi analisa ECG yang tepat. Pada hampir semua pengenalan parameter ECG berdasarkan kepada titik tetap yang dapat diidentifikasi pada setiap siklus gelombang.

Puncak sinyal R cocok untuk digunakan sebagai titik referensi, karena mempunyai amplituda yang terbesar dan bentuk gelombang yang paling tajam. Waktu dan ukuran amplituda dapat diketahui ketika puncak dari setiap gelombang R terdeteksi pada setiap siklus gelombang. Teknik *Real-Time QRS Detection* meliputi *bandpass filtering*, differensiasi, pengukuran daya rata-rata dan *thresholding*.



Gambar 2.3 Skema Pengolahan Sinyal ECG pada Algoritma Pan-Tompkins<sup>[2]</sup>

### 2.2.1 Filtering Sinyal

Tahap pertama dalam mendeteksi kompleks QRS pada perangkat keras ini adalah memproses sinyal ECG dengan cara melewatkannya sinyal pada sebuah filter. Sinyal dilewatkan pada sebuah bandpass filter agar didapatkan daya sinyal pada frekuensi rata-rata dimana kompleks QRS ini memiliki daya yang paling optimal yaitu pada frekuensi 5 Hz sampai dengan 15 Hz.

Pada algoritma Pan-Tompkins ini, bandpass filter yang akan diimplementasikan adalah integer filter rekursif yang dimana pole-polenya diletakan untuk menghilangkan zero pada satuan lingkaran z. Band pass filter ini akan dibentuk dari masing-masing low pass filter dan high pass filter.

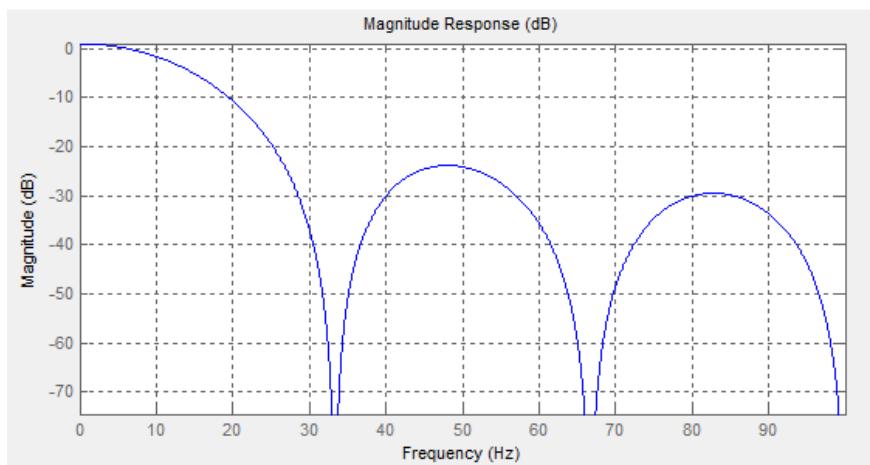
#### 2.2.1.1 Low Pass Filter

Dalam memperoleh daya maksimal dari kompleks QRS yang berada pada frekuensi 5 Hz sampai dengan 15 Hz, langkah pertama sinyal akan dilewatkan melalui low pass filter yang dibentuk dari persamaan fungsi transfer:

$$H(z) = \frac{(1-z^{-6})^2}{(1-z^{-1})^2} \quad (2.1)$$

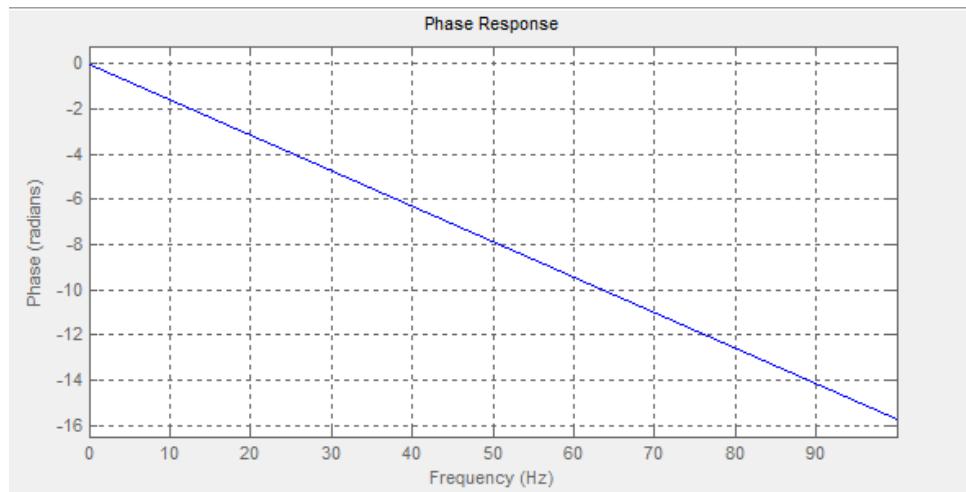
Dengan fungsi transfer tersebut, maka akan didapatkan persamaan respon magnitud dari filter sebagai berikut:

$$|H(\omega T)| = \frac{(\sin(3\omega T))^2}{(\sin(\frac{\omega T}{2}))^2} \quad (2.2)$$

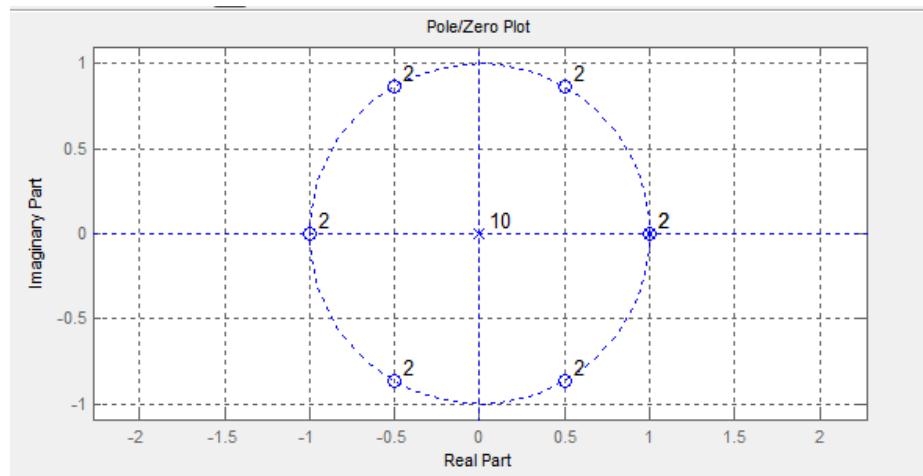


**Gambar 2.4 Respon magnitude dari low pass filter dengan frekuensi sampling 200 Hz**

Untuk penggunaan frekuensi sampling 200 Hz, maka didapatkan frekuensi cut off -3 dB sekitar 11 Hz dan untuk frekuensi 60 Hz dimana powerline noise bekerja akan mengalami redaman lebih dari 35 dB yang menyebabkan noise tereduksi secara signifikan. Low pass filter ini menyebabkan gain sebesar 36 kali sehingga diperlukan penurunan daya sekitar 36 kali. Sedangkan respon fasa yang dihasilkan bersifat linier.



**Gambar 2.5 Respon fasa dari low pass filter**



**Gambar 2.6 Plot pole dan zero low pass filter**

### 2.2.1.2 High Pass Filter

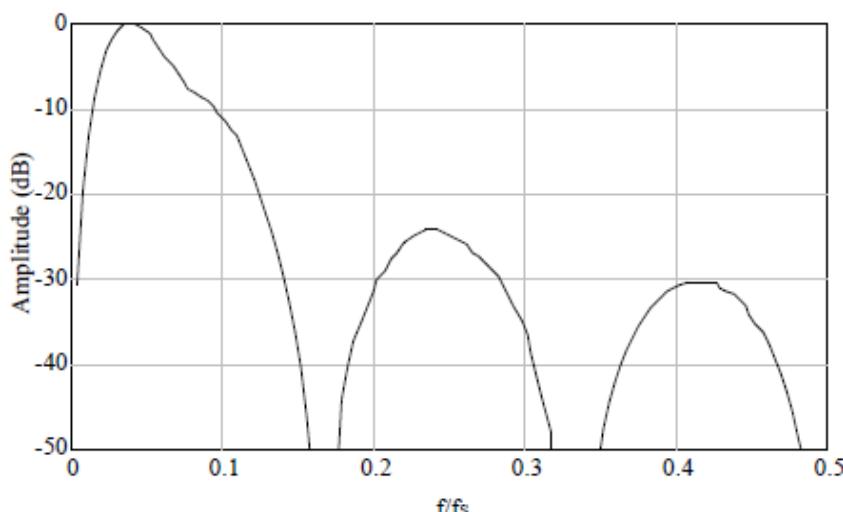
Setelah melewati bagian low pass filter, sebagai bagian proses filter sinyal maka selanjutnya akan dilakukan proses filtering melalui high pass filter. Setelah dilakukan pembatasan frekuensi atas melalui low pass filter tadi, pada proses melalui high pass filter ini akan dilakukan pembatasan frekuensi bawah. High pass filter ini merupakan hasil subtraksi low pass filter orde satu dari all pass filter dengan delay. Adapun low pass filter untuk frekuensi sampling 200 Hz memiliki fungsi transfer :

$$H_{LP}(z) = \frac{Y(z)}{X(z)} = \frac{1-z^{-32}}{1-z^{-1}} \quad (2.3)$$

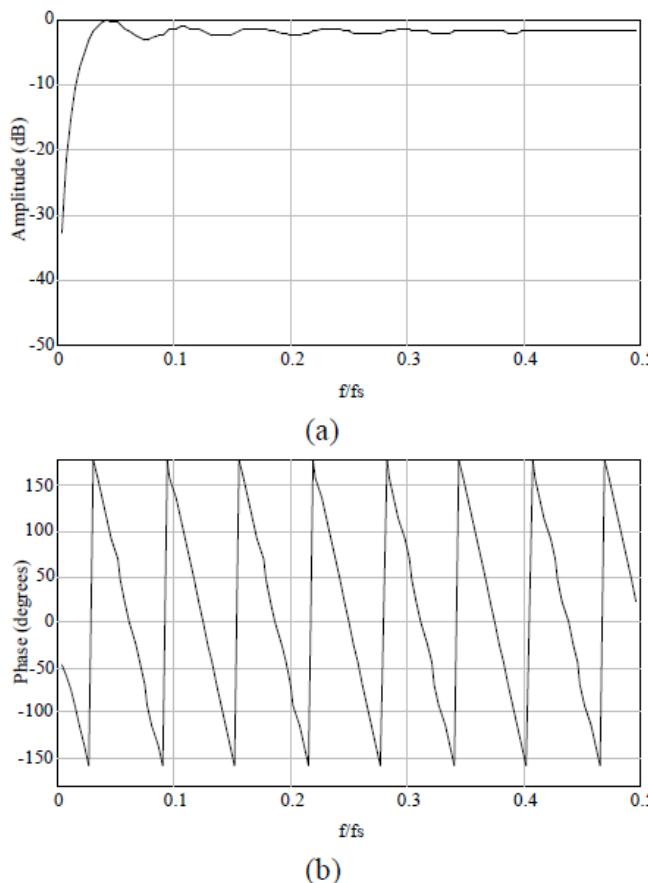
Filter tersebut menghasilkan penguatan dc sebesar 32 kali dan delay sebanyak 15.5 periode sampling. High pass filter dibangun dari hasil pembagian low pass filter dengan besar penguatan dc lalu dikurangi dengan sinyal asli, sehingga fungsi transfer high pass filter adalah :

$$H_{hp}(z) = \frac{P(z)}{X(z)} = z^{-16} - \frac{H_{lp}(z)}{32} \quad (2.4)$$

High pass filter yang digunakan memiliki frekuensi cut off sebesar 5 Hz dengan penguatan sebesar 1 kali dan menghasilkan delay sebesar 16 sampling. Filter inipun akan menghasilkan respon fasa yang linear. Sehingga band pass filter hasil penggabungan low pass filter dan high pass filter akan menghasilkan frekuensi tengah 10 Hz yang akan menyebabkan filter memiliki respon frekuensi yang sesuai dengan spektrum frekuensi rata-rata dari kompleks QRS dan memberikan redaman kepada sinyal dengan frekuensi lebih rendah maupun lebih tinggi.



**Gambar 2.7 Respon amplitud dari band pass filter**



Gambar 2.8 (a) Respon Amplitud dan (b) Respon Fasa High Pass Filter<sup>[2]</sup>

### 2.2.2 Differensiasi

Setelah dilakukan filter, sinyal diolah dengan dilakukan differensiasi dengan tujuan untuk memperoleh slope dari kompleks QRS kemudian dikuadratkan agar memiliki nilai data yang positif . Differensiasi dilakukan untuk membatasi frekuensi yang akan diproses, dimana hanya frekuensi yang tinggilah yang akan diloloskan. Tujuan pembatasan ini adalah untuk memperoleh struktur sinyal QRS dan membuang sinyal P dan T. Proses differensiasi sinyal akan diakukan sesuai dengan persamaan matematis berikut ini:

$$y(n) = \frac{1}{8}[2x(n) + x(n-1) - x(n-3) - 2x(n-4)] \quad (2.5)$$

### 2.2.3 Pengkuadratan

Setelah mengalami differensiasi sinyal akan mengalami tahap pengkuadratan dengan tujuan memperoleh nilai yang positif dan memiliki ketimpangan nilai yang tinggi antar amplituda satu sama lain. Berikut ini adalah persamaan kuadrat dari sinyal:

$$y(n) = x(n)^2 \quad (2.6)$$

### 2.2.4 Integrasi

Setelah mengalami pengkuadratan, sinyal kompleks QRS kemudian akan mengalami integrasi. Integrasi merupakan proses yang dilakukan untuk memperoleh informasi dari sinyal. Proses integrasi yang dilakukan pada sistem ini adalah pengolahan sisnyal berdasarkan persamaan berikut ini:

$$y(n) = \frac{1}{N} [x(n - (N - 1)) + x(n - (N - 2)) + \dots + x(n)] \quad (2.7)$$

Besar nilai N menyatakan jumlah dari *sliding window* yang digunakan dimana besarnya bergantung kepada nilai frekuensi sampling. Pada implementasi sistem ini, jumlah N yang digunakan adalah 32. Nilai ini dipilih karena menghasilkan hasil yang terbaik berdasarkan hasil simulasi.

#### 2.2.5 Threshold

Blok selanjutnya dari sistem yang akan direalisasikan adalah blok *threshold*. Pada blok ini akan diberikan batasan nilai dari sinyal hasil keluaran blok integrasi dimana sinyal yang memiliki nilai diatas *threshold* akan diidentifikasi sebagai puncak sinyal dan diberi nilai '1' sedangkan untuk nilai dibawah *threshold* akan tidak akan teridentifikasi dan diberi nilai '0'.

### 2.3 Field-Programmable Gate Array (FPGA)

FPGA (*Field Programmable Logic Array*) merupakan suatu IC (*Integrated Circuit*) tipe HDL (*High speed IC Description Language*) yang dapat diprogram untuk melakukan fungsi-fungsi logika tertentu sesuai dengan kebutuhan. FPGA merupakan *Programmable Logic Device* (PLD) yang dibangun dari sekumpulan sel fungsi logika dasar yang dapat diprogram. Sel-sel logika ini terhubung satu sama lain melalui suatu jaringan interkoneksi yang juga dapat diprogram. Jadi dalam kata lain, FPGA ini dapat diprogram sesuai dengan keinginan *user*.

Kelebihan dari FPGA sendiri adalah sebagai berikut :

- Relatif murah, karena jumlahnya bisa diperbanyak.
- Dapat dikonfigurasi oleh *End User*.
- Tidak memerlukan proses fabrikasi.
- Tersedia solusi yang mendukung *chip customized VLSI*.

Sekarang ini sudah terdapat bermacam-macam keluarga FPGA dengan spesifikasi perangkat yang berbeda. Salah satu perusahaan yang memproduksi FPGA adalah Xilinx. Jenis-jenis FPGA yang diproduksi Xilinx antara lain VIRTEX, SPARTAN, XC3S1000, XC3000, XC4000 dan XC5000.

FPGA memiliki tiga elemen utama untuk membentuk arsitektur umumnya yaitu *Input/Output Blok ( IOB )*, *Configurable Logic Block ( CLB )* dan *Interkoneksi*.

1. *Configurable Logic Blocks*, yang memiliki fungsi :

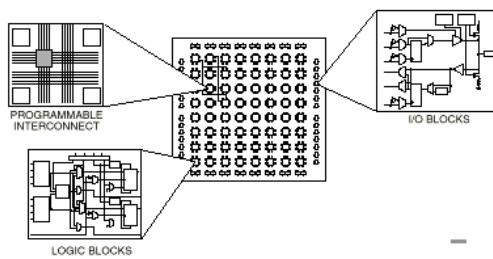
- Memiliki Look up table berdasarkan struktur kompleks komponen dalam FPGA.
- Mengimplementasikan rangkaian sekuensial

2. *Programmable Interconnect*, yang berfungsi:

- Berisi *wire segments* (kabel penyambungan antar segmen) dan *programmable switches* (port *switch* yang biasa diprogram sebagai input rangkaian).
- Menghubungkan antara *Configurable Logic Blocks* yang berbeda

3. *Input/output block*, yang memiliki fungsi :

- Sebagai interface antara *external package pin* dari perangkat dan *internal user logic* pada FPGA.



Gambar 2.9 Arsitektur bagian dalam FPGA<sup>[10]</sup>

## 2.4 VHSIC Hardware Description Language (VHDL)

VHDL (VHSIC Hardware Description Language) adalah salah satu bahasa pengkodean untuk mendeskripsikan rangkaian digital dalam kode yang dimengerti oleh perangkat (Hardware Description Language). VHDL merupakan salah satu dari jenis HDL. VHDL adalah penyempurnaan dari VHSIC (Very High Speed Integrated Circuit) yang pertama kali digunakan pada tahun 1983.

Keuntungan perancangan menggunakan VHDL adalah:

- Mampu mendesain hardware sampai level tertinggi ( sistem yang kompleks )
- Dapat mencari dan mendeteksi kesalahan dengan lebih mudah dalam simulasi
- Proses implementasi program independen sehingga memungkinkan untuk melakukan beberapa perubahan sampai menit – menit terakhir.
- Hardware untuk implementasi sangat flexibel, sehingga dapat digunakan untuk berbagai perancangan dan dapat dipilih sesuai kebutuhan.
- Bahasa pemrograman mudah dimengerti dan dipelajari dengan cepat.

Struktur dasar VHDL :

1. *Library* merupakan kumpulan modul-modul yang digunakan dalam program. Sebelum digunakan dalam program, modul yang ada dalam *library* perlu diinisialisai terlebih dahulu.
2. Bentuk rangkaian yang dibuat berfungsi untuk menggambarkan port pada input ataupun outputnya disebut *Entity*. *Entity* juga dapat berisi parameter yang akan digunakan di dalam desain.
3. *Architecture* merupakan diskripsi kerja dari sistem atau *entity* yang akan kita rancang. Pada *architecture* terdapat satu set komponen yang saling berhubungan. Dalam pendeskripsinya, *architecture* dapat didesain secara *behavioral* (secara prinsip kerja alat), *structural* (struktur dalam port map), *dataflow* (secara rangkaian logika), ataupun dengan campuran ketiga cara tersebut.
4. *Process* merupakan bagian dari suatu *architecture*. Di dalam suatu *architecture* bisa terdapat satu atau lebih *process* yang saling *independen* ataupun berhubungan.
5. Untuk mendeklarasikan kumpulan desain *entity* tertentu sebagai kumpulan *instan* pada sebuah *architecture* tertentu di dalam suatu *entity* disebut Configurasi.
6. *Signal* dapat dianalogikan seperti kabel yang menghubungkan antar bagian dalam sistem yang didesain. *Signal* dideklarasikan di dalam *architecture*. Nilai logika dari suatu *signal* baru berubah setelah seluruh proses dieksekusi. Apabila dalam suatu *signal* nilainya diubah beberapa kali dalam suatu proses maka, nilai terakhir yang akan di pakai pada eksekusi selanjutnya. Sedangkan jika nilai dari suatu *variable* langsung berubah,tanpa harus menunggu selesainya proses eksekusi. *Variable* hanya dapat dideklarasikan di dalam *process*.

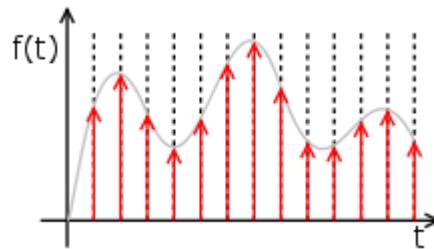
### 2.3 Analog to Digital Converter (ADC)

Pengubah analog-ke-digital adalah perangkat elektronika yang dapat mengubah sinyal digital (biasanya dalam notasi biner) bentuk sinyal analog (baik sebagai arus, tegangan, maupun muatan listrik) ke sinyal digital (biasanya dalam notasi biner). Alat pengubah analog-ke-digital ini sering dikenal sebagai ADC (*Analog to Digital Converter*) yang banyak dijumpai pada rangkaian elektronika dan instrumentasi.

Secara umum, suatu sistem ADC memiliki tiga proses dalam mengkonversi sinyal analog menjadi rangkaian bit sinyal digital yaitu *sampling*, kuantisasi dan *encoding*. *Sampling* merupakan proses untuk mendapatkan sinyal diskrit dari sinyal analog.  $X(t)$

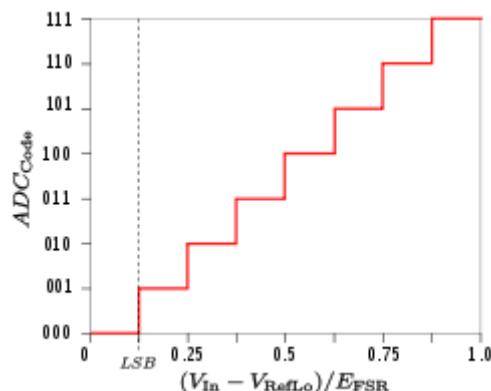
merepresentasikan sinyal analog dan  $x(n)$  merepresentasikan sinyal diskrit hasil sampling dengan menggunakan nilai  $x(t)$  dalam interval  $nTs$ . Sinyal diskrit mengambil sinyal sample dari sinyal analog (kontinyu) pada interval tertentu. Secara sistematis:

$$X[n] = X(t) \Big|_{t=nTs}$$



**Gambar 2.10 Proses Pensemplingan Sinyal Analog Menjadi Sinyal Diskrit**

Setelah mengalami pensamplingan, sinyal diskrit hasil sampling akan mengalami proses selanjutnya, yaitu kuantisasi. Kuantisasi adalah proses pemberian level untuk tiap-tiap representasi sample dalam bentuk sinyal pulsa dengan amplitude proposional terhadap sinyal yang disample. Pada proses ini terjad pembulatan amplitude tiap pulsa ke level terdekat. Semakin besar nilai bit representasi yang digunakan, maka semakin banyak pula level kuantisasi, sehingga error kuantisasi semakin kecil.



**Gambar 2.11 Proses Kuantisasi Sinyal Diskrit**

Tahap terakhir dari proses konversi sinyal analog ke digital adalah *encoding*. *Encoding* adalah proses representasi sinyal hasil kuantisasi ke dalam deretan bit biner (digital).

## **BAB III**

### **PERANCANGAN SISTEM DETEksi KOMPLEKS QRS**

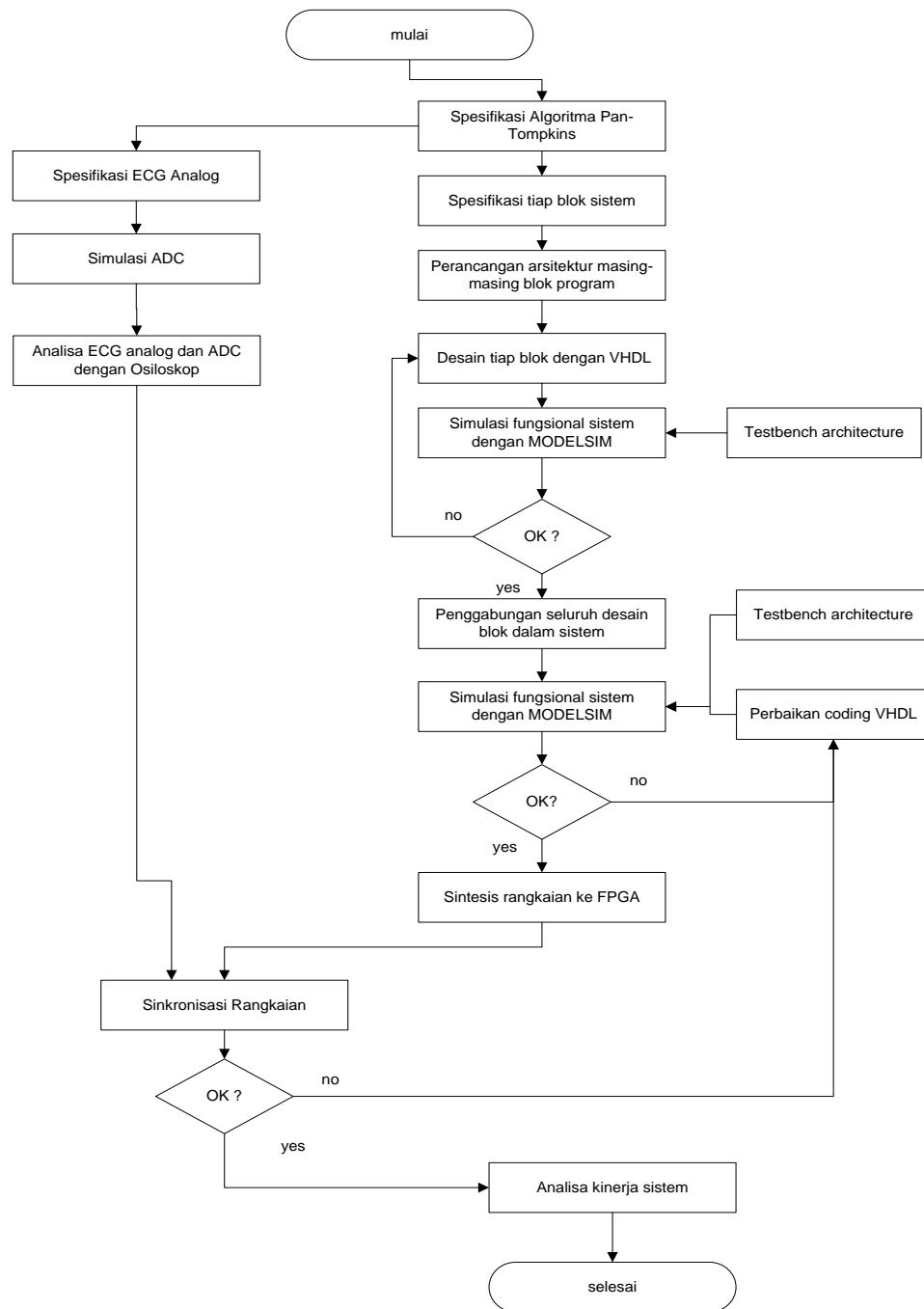
Dalam bab ini akan diberikan mengenai gambaran dari sistem pendeksi QRS yang dirancang berdasarkan pada metode Pan-Tompkins. Selanjutnya akan dijelaskan mengenai implementasi sistem pendeksi QRS tersebut pada arsitektur program VHDL. Arsitektur program VHDL yang dirancang dibagi kedalam 5 blok sesuai dengan blok pada algoritma pendeksi QRS Pan-Tompkins dimana untuk masing-masing bloknya akan terbagi lagi menjadi beberapa sub-blok yang menyesuaikan kepada sistem kerja blok tersebut. Kemudian baru akan diimplementasikan pada *board* FPGA.

Selain perancangan dan pemodelan secara *software* untuk arsitektur program pendeksi kompleks QRS dengan VHDL, akan dilakukan juga perancangan sistem ADC yang disesuaikan dengan kebutuhan sistem yaitu dapat dioperasikan dengan perangkat FPGA dan dapat mengkonversi sinyal keluaran ECG analog menjadi urutan bit ECG digital.

#### **3.1 Diagram Alir Perancangan dan Implementasi Deteksi Kompleks QRS**

Perancangan sistem deteksi kompleks QRS ini dimulai dengan melakukan spesifikasi dari algoritma yang akan digunakan yaitu Pan-Tompkin hingga dapat ditentukan kebutuhan sistem yang nantinya akan dibangun, baik kebutuhan dari blok program yang akan dirancang maupun kebutuhan karakteristik masukan sistem yang nantinya dapat diproses. Dari kebutuhan yang telah ditentukan tadi dirancanglah arsitektur tiap blok sistem yang sesuai dengan karakter pemrograman pada VHDL yang kemudian didefinisikan melalui kode VHDL melalui *software* ModelSim. Secara paralel dilakukan pula spesifikasi dari jenis keluaran sinyal ECG analog untuk dikonversi menjadi sinyal digital. Hasil spesifikasi sinyal ECG tersebut, beserta dengan kebutuhan sistem algoritma Pan-Tompkins menjadi dasar dari perancangan ADC yang dilakukan pada tahap selanjutnya.

Selanjutnya dilakukan pengujian baik untuk program VHDL menggunakan testbench pada ModelSim, maupun pada ADC dan ECG analog dengan osiloskop. Selanjutnya program VHDL yang telah memenuhi spesifikasi akan diimplementasikan pada *board* FPGA yang kemudian akan diintegrasikan menjadi satu sistem dengan ADC yang telah dirancang tadi. Untuk lebih lengkapnya, proses perancangan dan implementasi sistem deteksi QRS ini dapat dilihat pada diagram alir yang ada pada gambar 3.1.



Gambar 3.1 Diagram alir perancangan dan implementasi sistem deteksi QRS

### 3.2 Model dan Parameter Sistem Deteksi QRS Algoritma Pan-Tompkins

Pada algoritma ini prinsip yang digunakan dalam mendeteksi komplek QRS adalah dengan melakukan pengolahan sinyal berupa *filtering* sinyal di frekuensi optimal tempat bekerjanya kompleks QRS. Selanjutnya dilakukan pengambilan informasi dari kelandaian aktifitas sinyal pada domain waktu melalui diferensiasi dan integrasi *moving window*. Tahap selanjutnya adalah penentuan *threshold* dari sinyal yang terdeteksi, dan dari batas *threshold*

ini dilakukan penentuan apakah komponen sinyal merupakan kompleks QRS atau bukan. Secara lengkap proses pada algoritma Pan-Tompkins dapat kita lihat pada gambar 3.2 berikut ini.



**Gambar 3.2 Blok sistem deteksi QRS pada algoritma Pan-Tompkins**

Dalam implementasinya, setiap blok sistem algoritma Pan-Tompkins akan diterjemahkan dalam arsitektur pada program VHDL. Pada sistem ini akan digunakan sampling sebanyak 200 sampling perdetiknya, dengan masing-masing sampling berupa *logic vector* ber-*array* 10 bit, sehingga setiap sampling mengalami 1024 level kuantisasi. Masukan dari sistem ini adalah 8 bit *unsigned* ECG yang kemudian ditambah 2 bit untuk menjadikannya bilangan 10 bit *signed*. Dengan sistem seperti ini diperkirakan tidak akan terjadi *overload* bit akibat operasi matematika yang terjadi pada masing-masing blok sistem.

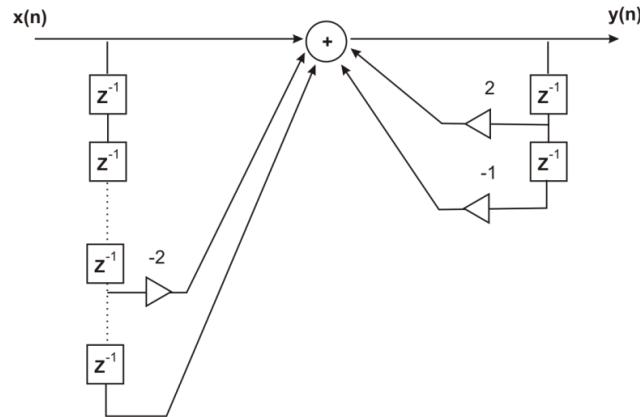
### 3.3 Arsitektur Sistem pada VHDL

Setelah memahami karakteristik dan kebutuhan dari sistem deteksi QRS yang akan dirancang, tahap selanjutnya adalah pengadaptasi model sistem tersebut ke model sistem yang dibangun oleh komponen-komponen logika sehingga dapat didefinisikan pada bahasa VHDL. Pada implementasi sistem ini, dibangun beberapa blok sistem yang masing-masing blok sistemnya terdiri dari komponen-komponen logika seperti *shift register*, adder, multiplier, divider serta address loader.

#### 3.3.1 Blok Sistem Low Pass Filter

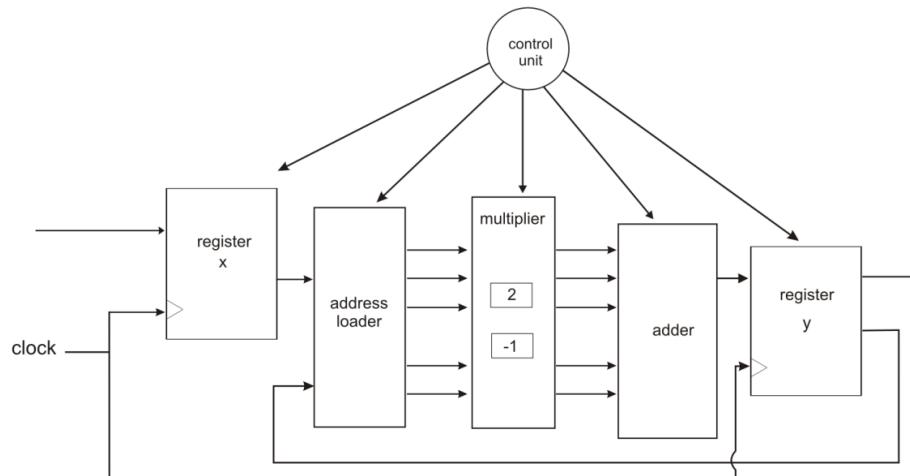
Pada blok ini akan dirancang sebuah sistem yang dapat membatasi frekuensi dari sinyal masukan, frekuensi yang digunakan disini adalah sekitar 11 Hz dengan sampling sebanyak 200 sample per detik sehingga sesuai dengan algoritma Pan-Tompkins persamaan perbedaan filter yang digunakan adalah :

$$y(n) = 2y(n-1) - y(n-2) + x(n) - 2x(n-6) + x(n-12) \quad [1] \quad (3.1)$$



**Gambar 3.3 Realisasi matematika *low pass filter* yang digunakan**

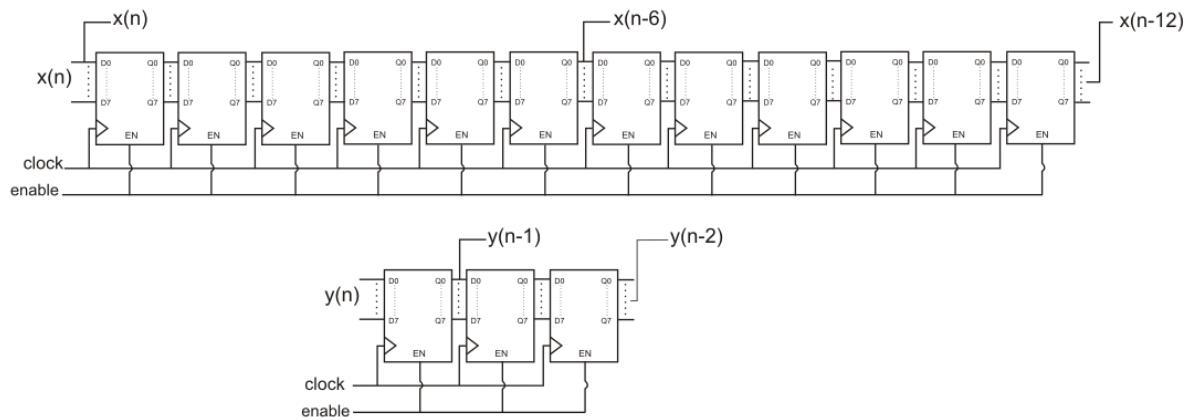
Pada perancangan *hardware* ini, *low pass filter* akan diimplementasikan pada bahasa VHDL dengan mengadaptasi sistem realisasi secara matematika seperti yang terlihat pada gambar 3.3. Dari realisasi filter tadi, kemudian dirancang sistem pada bahasa VHDL berupa sebuah arsitektur sistem yang terdiri dari blok-blok sistem logika, yaitu *control unit*, *register*, *address loader*, *multiplier* serta *adder*. Adapun design arsitektur dari *low pass filter* adalah seperti yang terlihat pada gambar 3.4 berikut ini.



**Gambar 3.4 Blok program implementasi *low pass filter* pada VHDL**

Pada sistem ini dirancang sebuah register yang digunakan untuk menyimpan masukan dan keluaran dari periode sebelumnya. Sehingga dengan kata lain register ini berfungsi sebagai pengganti *delay* yang ada pada *data flow filter* secara matematika. Terdapat 2 buah register yang digunakan, masing-masing untuk menyimpan nilai x (masukan) dan untuk y (keluaran), keduanya berupa *shift register SIPO* (*Serial In Parallel Out*). Register untuk nilai x dirancang memiliki kapasitas memori 120 bit. Besar kapasitas ini didapatkan dari kebutuhan sistem yang membutuhkan *delay* hingga  $z^{-12}$  dengan masing-masing masukan

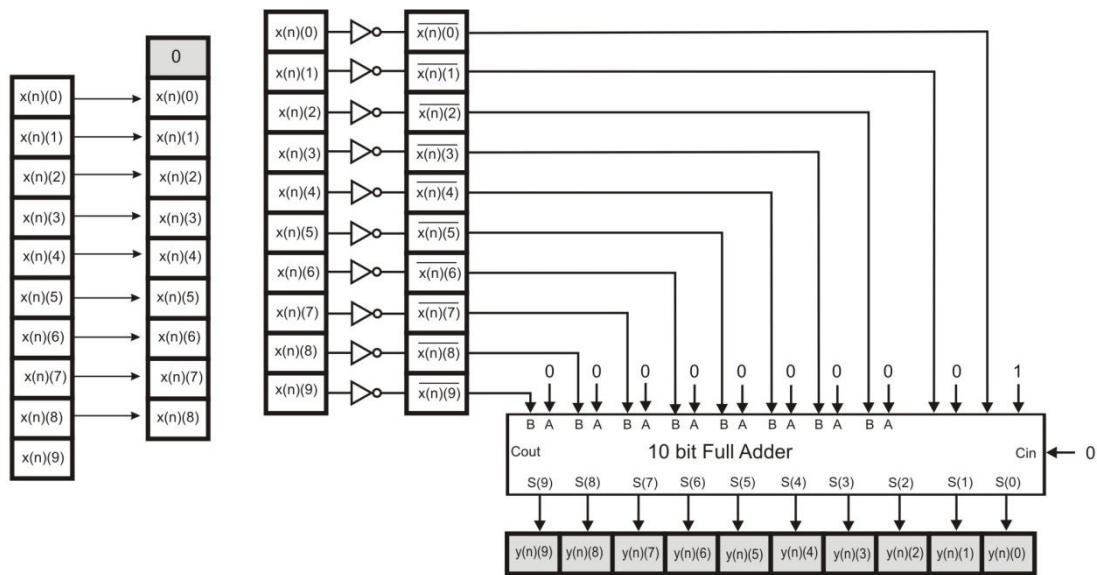
terdiri dari 10 bit. Sedangkan register yang digunakan untuk nilai  $y$ , kapasitas yang disediakan adalah 2 byte, karena delay yang dibutuhkan hanya sampai  $z^{-2}$  dengan besar data 1 byte.



**Gambar 3.5 Blok program register pada low pass filter**

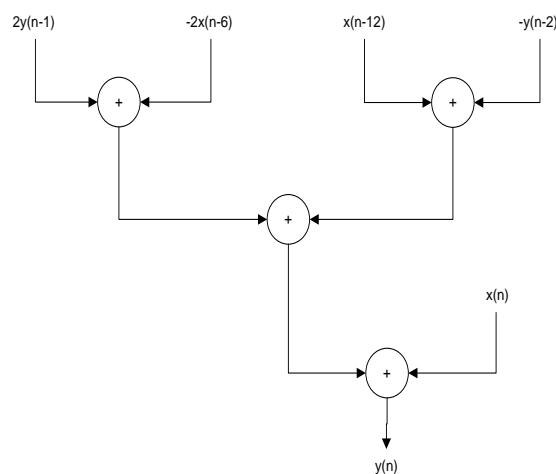
Pada sistem ini digunakan 2 macam *address loader*, yang merupakan sistem pemetaan port (*port mapping*) dari blok program ke blok program lainnya. *Address loader* yang pertama, digunakan untuk pengaturan data masuk ke sistem *low pass filter*, sedangkan *address loader* yang kedua merupakan sistem pemetaan data yang berasal dari register yang dihubungkan dengan blok sistem *multiplier* sehingga *address loader* kedua ini ada dua buah masing masing untuk register  $x$  dan register  $y$ . Sesuai dengan persamaan 3.3 *address loader* untuk register  $x$  ini akan bekerja dengan cara mengambil data dari register  $x$  ke 6 yang merupakan  $x(n-6)$  yang kemudian disalurkan ke *multiplier* bernilai -2, sehingga didapatkan nilai  $-2x(n-6)$  yang kemudian menjadi komponen masukan dari adder. Sistem kerja ini juga bekerja pada nilai komponen masukan *adder* lainnya, yaitu  $x(n-12)$ ,  $y(n-2)$  dan  $y(n-1)$ .

Blok selanjutnya dari implementasi sistem *low pass filter* ini adalah *multiplier* yang berfungsi untuk memperoleh nilai komponen penjumlahan dari nilai yang telah disimpan pada register. Blok ini merupakan sebuah sistem fungsi perkalian, yang pada blok sistem *low pass filter* memiliki 3 nilai pengali yaitu -1, 2 dan -2. Untuk nilai pengali -1 yang dikerjakan pada blok ini adalah perubahan nilai yang masuk dari nilai positif menjadi nilai negatif, sistem bilangan negatif yang digunakan di sini adalah sistem bilangan 2's *complement*. Untuk bilangan pengali 2, sistem kerja dari sistem adalah menggeser bit satu tingkat ke arah MSB, hal ini akan menyebabkan nilai dari bilangan akan meningkat 2 kali lipat. Pada bilangan pengali -2 yang dilakukan pada sistem adalah pergeseran satu bit ke arah MSB lalu kemudian dikonversi sesuai sistem bilangan 2's *complement*.



**Gambar 3.6 Blok program *multiplier* pada *low pass filter***

Blok selanjutnya dari sistem ini adalah *adder*, fungsi dari *adder* ini adalah untuk menjumlahkan semua keluaran dari multiplier sehingga diperolehlah nilai  $y(n)$  yang sesuai dengan persamaan dari filter. Operasi penjumlahan yang digunakan disini tidaklah dilakukan secara langsung, namun bertahap terbagi menjadi beberapa operasi penjumlahan dengan dua operator. Hal ini dilakukan untuk mengurangi kebutuhan pembulatan bilangan karena pada adder ini terdapat jumlah bit yang sama antara masukan dengan keluaran sehingga akan membutuhkan banyak pembulatan jika dilakukan operasi penjumlahan secara langsung.



**Gambar 3.7 Skema penjumlahan pada blok *adder* *low pass filter***

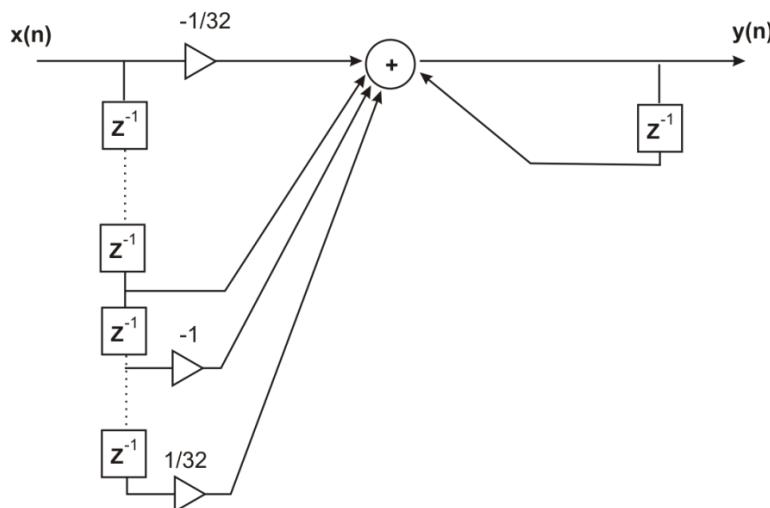
Blok sistem selanjutnya adalah control unit. *Control unit* ini berfungsi untuk mengatur sinkronisasi kerja keseluruhan blok sistem low pass filter ini. Dalam melaksanakan fungsi

untuk mensinkronisasi keseluruhan blok, *control unit* mengeluarkan tiga buah sinyal, yaitu *clock*, *enable* serta *reset*. Sinyal *enable* digunakan untuk mengatur blok sistem untuk mulai dan berhenti bekerja. Sinyal *reset* digunakan untuk mengatur keseluruhan sistem kembali ke mode awal, artinya seluruh register akan kembali bernilai 0. Sinyal terakhir yaitu *clock* berfungsi untuk mengatur aliran keluar masuk data ke dalam setiap blok, untuk memperoleh sistem yang bekerja secara serempak, maka sumber *clock* yang tunggal sangatlah penting.

### 3.3.2 Blok Sistem High Pass Filter

Setelah melewati *low pass filter*, sinyal ECG kemudian akan mengalami *filtering* kedua yang kali ini dipergunakan untuk membatasi frekuensi rendah yang akan dilewatkan. Frekuensi *cut off* yang digunakan adalah 5 Hz dengan persamaan perbedaan dari filter:

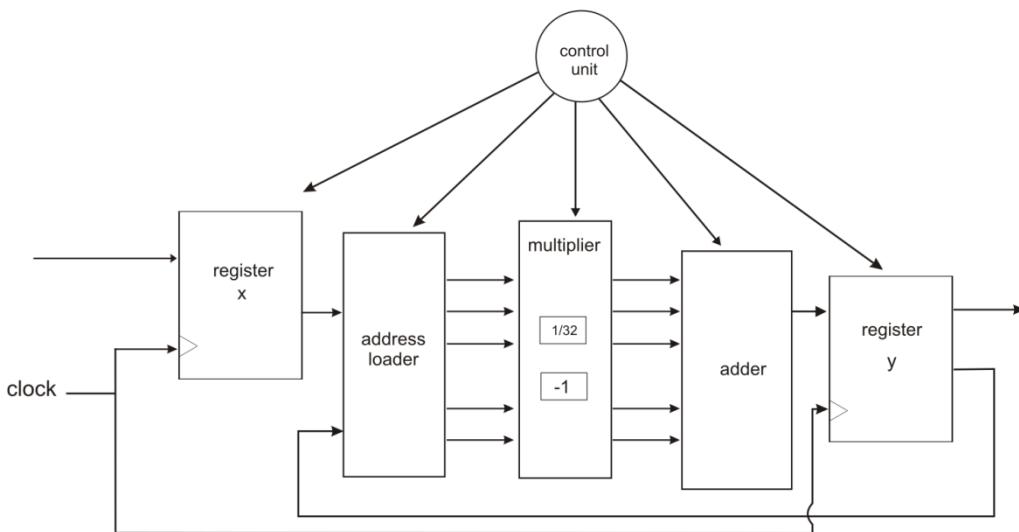
$$y(n) = y(n - 1) + x(n - 16) - x(n - 17) - 1/32 [x(n) - x(n - 32)]^{[1]} \quad (3.2)$$



Gambar 3.8 Realisasi matematika *high pass filter* yang digunakan

Pada implementasinya dengan bahasa pemrograman VHDL, hampir sama dengan apa yang diimplementasikan pada blok *low pass filter*, *high pass filter* akan diimplementasikan menjadi beberapa blok yaitu, *control unit*, *address loader*, *register*, *multiplier* serta *adder* seperti pada gambar 3.9.

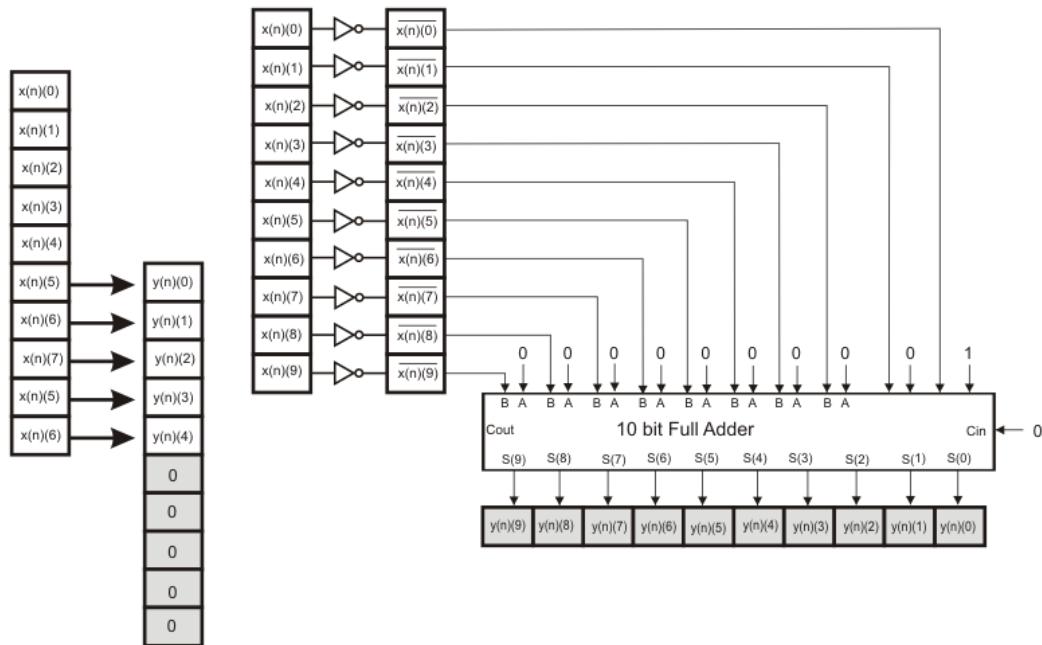
Hampir serupa dengan blok *low pass filter*, pada *high pass filter* ini pun terdapat sebuah register yang digunakan, masing-masing untuk menyimpan nilai *x* (masukan) dan untuk *y* (keluaran), keduanya berupa *shift register SIPO* (*Serial In Parallel Out*). Register untuk nilai *x* dirancang memiliki kapasitas memori 320 bit karena pada *high pass filter* ini dibutuhkan *delay* hingga  $z^{-32}$ . Sedangkan register yang digunakan untuk nilai *y*, kapasitas yang dibutuhkan hanya 1 byte, karena *delay* yang dibutuhkan hanya sampai  $z^{-1}$ .



**Gambar 3.9 Blok program implementasi *high pass filter* pada VHDL**

Kemudian blok sistem selanjutnya adalah *address loader* yang prinsipnya sama dengan *address loader* pada *low pass filter*, yaitu mengambil data dari register, lalu dioperasikan melalui *multiplier* sesuai dengan operator pengali yang diinginkan. Pada sistem *high pass filter* ini faktor pengali yang digunakan adalah  $1/32$ ,  $-1/32$  dan  $-1$ . Untuk operator pengali  $-1$ , sama dengan yang digunakan pada *low pass filter* sehingga pada *high pass filter* ini tidak perlu dibuat ulang namun cukup dipanggil sebagai komponen. Sedangkan untuk operator  $1/32$ , prinsip kerja yang digunakan adalah menggeser 5 bit ke arah LSB sehingga nilai akan tereduksi besar 32 kali. Namun hal ini hanya dapat dilakukan untuk bilangan positif, sehingga untuk bilangan negatif akan dikonversi terlebih dahulu ke bilangan positif, baru kemudian kembali dikonversi menjadi bilangan negatif setelah dilakukan perkalian  $1/32$ . Untuk operator  $-1/32$  dilakukan gabungan dari dua operator sebelumnya.

Hasil keluaran dari *multiplier* ini kemudian dijumlahkan pada blok *adder*. Sistem kerja *adder* yang digunakan disini sama dengan *adder* yang terdapat pada blok *half pass filter*. Untuk sinkronisasi sistem, pada blok *high pass filter* inipun terdapat *control unit* yang secara prinsip kerja serupa dengan *control unit* pada *low pass filter* yaitu dengan menggunakan sinyal *clock*, *reset* serta *enable*. Hasil keluaran dari blok *high pass filter* ini kemudian masuk pada blok sistem selanjutnya yaitu blok sistem *derivative* untuk didiferensiasi.

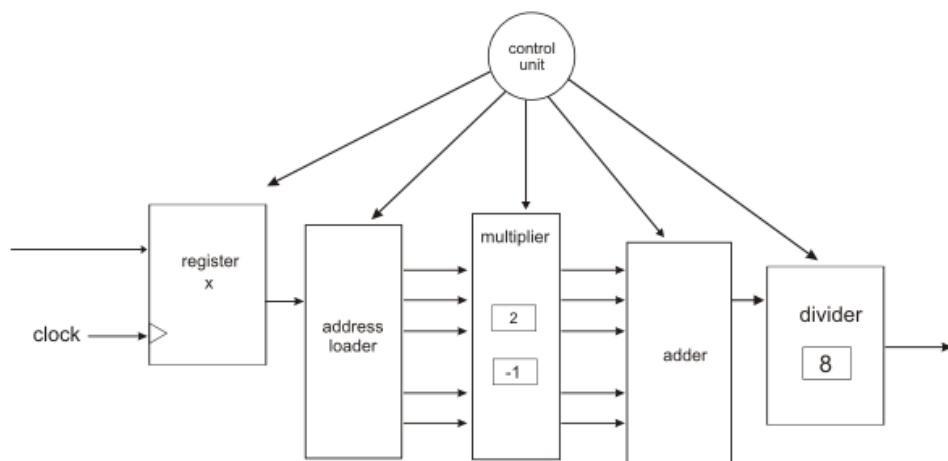
Gambar 3.10 Blok program *multiplier* pada *high pass filter*

### 3.3.3 Blok Sistem Derivatif

Setelah melewati band pass filter, sinyal ECG yang kini telah menyisakan frekuensi optimalnya akan didifferensiasi dengan tujuan memperoleh informasi kelandaian. Sistem differensiasi ini diimplementasikan berdasarkan pada persamaan:

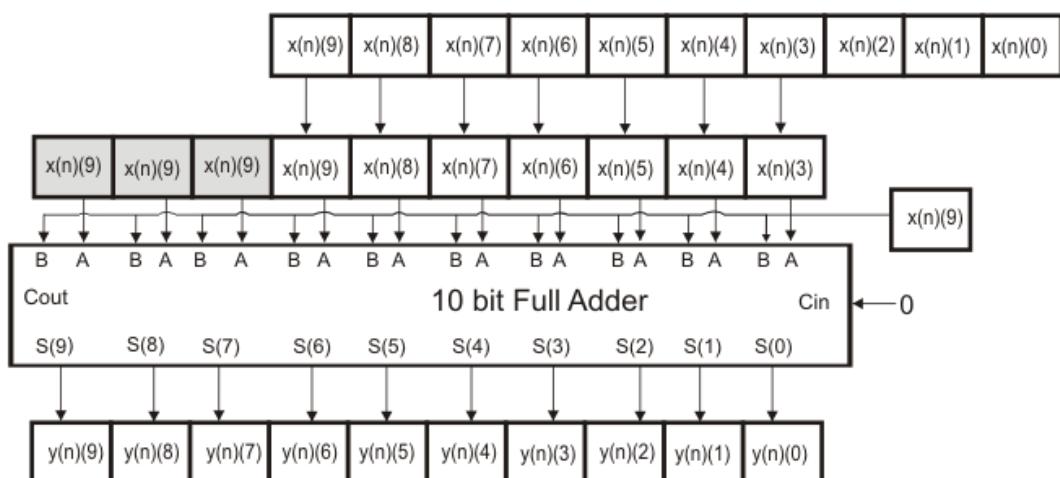
$$y(n) = 1/8 [2x(n) + x(n - 1) - x(n - 3) - 2x(n - 4)]^{[1]} \quad (3.3)$$

Pada sistem differensiator ini, register yang digunakan hanya satu buah yaitu untuk menyimpan nilai dari masukan (register x). Sedangkan untuk nilai y tidak dibutuhkan register karena tidak adanya sistem pengumpan balik. Register yang dirancang disini memiliki kapasitas 40 bit karena sistem membutuhkan *delay* hingga  $z^{-4}$ .

Gambar 3.11 Blok program implementasi sistem *derivative* pada VHDL

Blok selanjutnya dari sistem *derivative* ini adalah *address loader*, *multiplier* dan adder yang memiliki prinsip yang sama dengan blok serupa yang dirancang pada *low pass filter* dan *high pass filter*. Selain blok-blok tersebut, pada sistem *derivative* ini terdapat blok *divider* yang fungsinya untuk membagi nilai keluaran adder dengan nilai 8.

Prinsip yang digunakan dalam divider ini adalah menggeser bit ke arah LSB sebanyak 3 kali sebagai representasi 8 sebagai 2 pangkat 3. Selanjutnya bit yang telah digeser tadi ditambahkan dengan bit ke tujuh  $x(n)(7)$  dengan demikian bila bit berupa bilangan negatif akan tetap bernilai sesuai dengan sistem 2's complement. Hasil pembagian ini kemudian dikeluarkan dari *divider* dan menjadi hasil keluaran sistem *derivative* yang kemudian akan mengalami proses selanjutnya yaitu pengkuadratan.

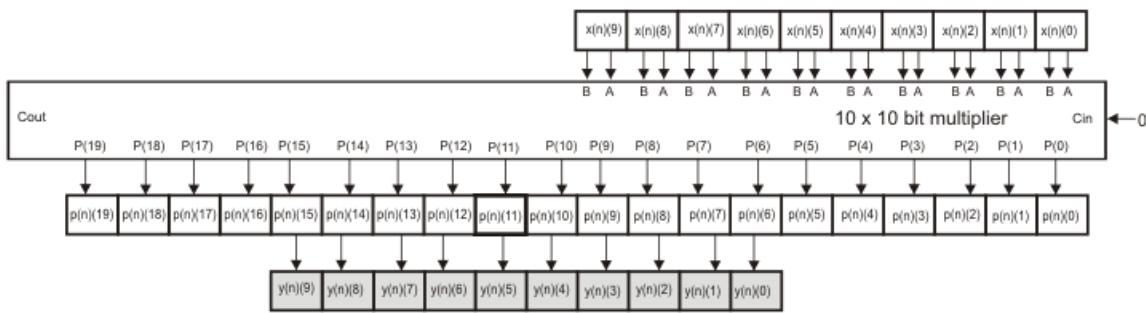


Gambar 3.12 Blok program divider pembangian dengan 8

### 3.3.4 Blok Sistem Fungsi Kuadrat

Proses yang dilakukan terhadap sinyal selanjutnya adalah pengkuadratan dengan tujuan memperoleh nilai mutlak dari sinyal. Proses ini terjadi pada blok sistem fungsi kuadrat, yang pada implementasi ini didesign sebagai suatu blok yang tediri dari 10x10 bit multiplier dengan kedua operannya adalah  $x(n)$ . Sehingga munculah hasil perkalian antara  $x(n)$  dengan  $x(n)$  yang menghasilkan nilai  $(x(n))^2$  yang kita sebut  $p(n)$  dengan jumlah array 20. Karena operasi pada sistem ini menggunakan logic vector ber-array 10, maka terhadap nilai dari  $p(n)$  harus dilakukan pemotongan yang dalam hal ini dilakukan pembagian dengan nilai 32 yang artinya bit bergeser 5 kali ke arah MSB dengan tujuan menghilangkan hasil operasi yang bernilai dibawah 32. Hal ini menyebabkan susunan *bit vector* yang ada tinggal  $p(n)(5)$  hingga  $p(n)(17)$ .

Selanjutnya karena hasil simulasi menunjukkan tidak terdapatnya daya dari sinyal yang mencapai level 32768 atau  $2^{16}$  maka untuk  $p(n)(16)$  hingga  $p(n)(20)$  dapat diabaikan. Dengan demikian dapat dilakukan pemetaan bit-bit  $y(n)$  yaitu dimulai  $p(n)(6)$  hingga  $p(n)(15)$ . Proses secara lengkap dari sistem kerja blok pengkuadratan ini dapat dilihat pada gambar 3.13 berikut ini.



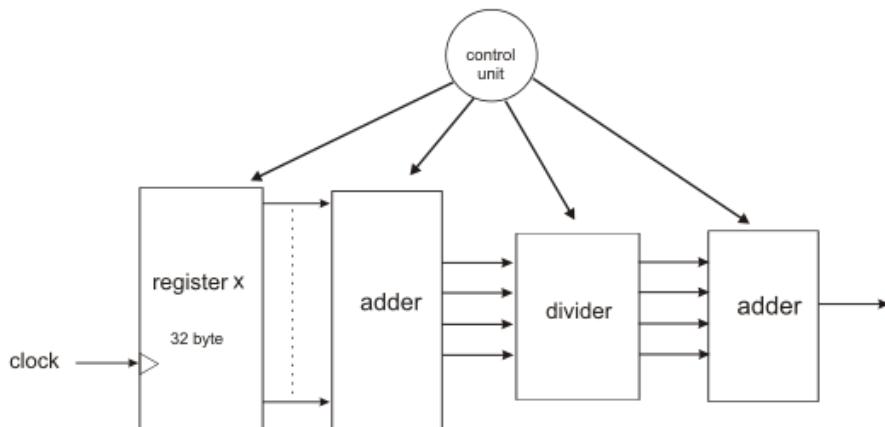
Gambar 3.13 Blok program pengkuadrat

### 3.3.5 Blok Sistem Integrasi

Pada blok ini dilakukan pengintegralan pada sistem dengan tujuan mendapatkan grafik kecenderungan nilai sinyal pada periode tertentu. Teknik yang dilakukan adalah mencari nilai rata-rata dari masukan selama periode tersebut. Pada sistem yang diimplementasikan kali ini periode yang digunakan adalah 32 sampling, artinya nilai yang diambil adalah hasil penjumlahan dari 32 masukan sebelumnya kemudian dibagi 32. Nilai 32 ini merupakan nilai kelipatan dua terbaik dari hasil yang di simulasikan. Dengan demikian diperoleh persamaan untuk moving window integral berikut ini:

$$y(n) = 1/32 [x(n - 32) + x(n - 31) + \dots + x(n)]^{[1]} \quad (3.4)$$

Blok sistem ini diimplementasikan dari sebuah register bekapasitas 320 bit, untuk 32 slot dengan masing-masing 10 bit. Register yang digunakan berupa register SIPO dengan *output* paralel yang langsung terhubung pada *adder*. Penjumlahan pada *adder* dilakukan bertingkat dengan dengan masing masing dijumlahkan untuk 8 buah nilai yaitu untuk  $x(n)(0)$  hingga  $x(n)(7)$ ,  $x(n)(8)$  hingga  $x(n)(15)$ ,  $x(n)(16)$  hingga  $x(n)(23)$ , serta  $x(n)(24)$  hingga  $x(n)(31)$ . Kemudian masing-masing hasil penjumlahan dilewatkan kepada pembagi 32 dan baru dijumlahkan keempatnya. Sistem seperti ini digunakan agar dapat memperkecil kemungkinan terjadinya overload nilai hingga diatas 10 bit untuk hasil penjumlahan secara langsung.



Gambar 3.14 Blok program integrator

### 3.3.6 Blok Sistem Decision

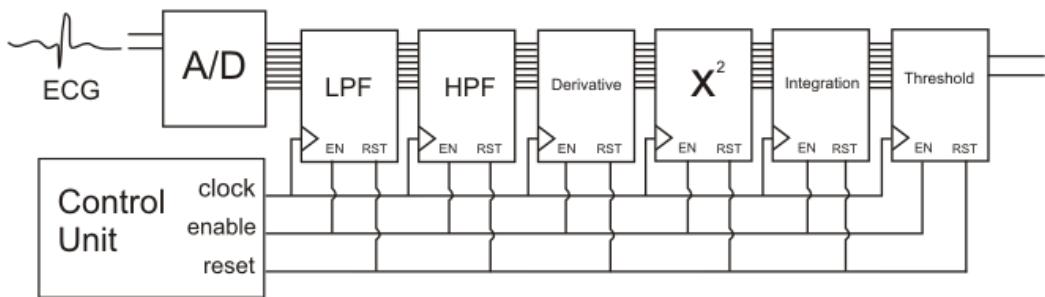
Blok ini berfungsi untuk memberikan batas nilai pada sinyal masukan yang menandakan bahwa sinyal tersebut merupakan kompleks QRS atau bukan. Bit 0 akan merepresentasikan kondisi bahwa sinya bukanlah sebuah kompleks QRS sedangkan untuk setiap komplek QRS akan direpresentasikan bit 1. Dengan prinsip demikian maka blok ini akan menerima masukan sinyal 10 bit yang kemudian dititjau niali dari kurvanya yang kemudian menghasilkan keluaran sinyal 1 bit yang memberikan informasi setiap kompleks QRS yang terdeteksi.

### 3.3.7 Integrasi Keseluruhan Blok Sistem

Secara keseluruhan, sistem deteksi kompleks QRS pada VHDL terbangun dari 6 blok sistem operasi serta satu buah control unit. *Controlling unit* berfungsi dalam mengatur lalu lintas data yang masuk ke dalam sistem yang kemudian diproses pada blok operasi dengan cara mengeluarkan sinyal clock, reset dan enable yang terhubung dengan control unit yang ada pada masing-masing blok. Dengan demikian control unit ini berfungsi untuk sinkronisasi kerja sistem secara keseluruhan, yang mengatur lalu lintas data aktifasi setiap bloknya.

Proses kerja dari integrasi seluruh blok sistem ini hampir mirip dengan sistem kerja sebuah shift register yang dimana sinyal ECG masukan akan berpindah state register untuk setiap clock. Namun untuk sistem ini setiap state register bekerja sebagai blok yang memiliki operasi tersendiri sehingga nilai yang masuk dan keluar setiap bloknya pasti akan berbeda. Sistem kerja seperti ini tidak memerlukan tambahan memori untuk melaksanakan fungsinya karena masing-masing blok operasi telah memiliki memori tersendiri berupa

register. Sistem kerja dari pengintegrasian keseluruhan blok ini dapat dilihat pada gambar 3.15.



Gambar 3.15 Integrasi blok-blok sistem pada VHDL

### 3.4 Implementasi Hardware Sistem Deteksi QRS

#### 3.4.1 Konversi ECG ke Sinyal Digital

Untuk pengujian perangkat keras, pada sistem ini sinyal inputan yang berasal dari ECG analog akan dilewatkan pada perangkat ADC terlebih dahulu. Keluaran dari ADC ini adalah sinyal digital dengan kuantisasi 8 bit. Adapun proses ADC akan dilakukan oleh perangkat ADC 0804. Sistem ECG analog dan ADC ini terintegrasi menjadi satu rangkaian hasil modifikasi rangkaian dari tugas akhir sebelumnya.

#### 3.4.2 Monitoring Keluaran Sinyal

Pada sistem deteksi kompleks QRS ini sistem memiliki dua monitoring terhadap nilai keluaran sinyal. Monitoring pertama dilakukan terhadap keluaran ECG analog dengan pengamatan menggunakan osiloskop. Sedangkan monitoring kedua dilakukan terhadap sistem keluaran dari FPGA dengan menggunakan logic analyzer yang disambungkan ke komputer. Hasil pengamatan terhadap dua monitor ini digunakan untuk melihat apakah sistem telah melakukan pendekripsi dengan akurat atau tidak.

### 3.5 Sistem pengambilan data

Pada tugas akhir ini, data masukan yang akan diproses merupakan data keluaran dari ECG dengan pengukuran pada 3 titik penyadapan yaitu RL, RA dan LA. Keluaran hasil pengukuran real time dari ECG ini akan menjadi masukan dari sistem yang akan diimplementasikan. ECG analog yang digunakan merupakan rangkaian ECG analog yang dikembangkan pada tugas akhir sebelumnya.

### 3.6 Performansi Sistem

Penilaian performansi sistem dilakukan dengan melihat tingkat akurasi sistem serta waktu yang dibutuhkan sistem dalam mendekripsi nilai dari kompleks QRS. Tingkat akurasi

yang akan ditunjukkan merupakan perbandingan nilai kompleks QRS terukur dibandingkan dengan sistem pengukuran lain yang telah ada.

$$\frac{\text{Akurasi} = \text{Jumlah data yang terdeteksi tepat}}{\text{Jumlah data yang diujikan}} \times 100\% \quad (3.5)$$

Sedangkan untuk waktu yang dibutuhkan sistem, dilihat dari total delay yang dibutuhkan dari mulai terjadinya puncak gelombang R hingga terdeteksinya kompleks QRS oleh sistem.

### 3.7 Perangkat Lunak Yang Digunakan

Untuk membantu jalannya simulasi sampai ke tahap sintesa perancangan sistem digunakan beberapa *software* berikut.

1. Microsoft EXCELL 2007, digunakan untuk mempermudah penghitungan nilai dari sinyal ECG sebelum dirubah menjadi test bench.
2. MODELSIM 6.3f, digunakan sebagai *software* pengkodean VHDL dan juga sebagai simulator terhadap sistem MIMO yang dirancang. Selain itu, MODELSIM juga berfungsi untuk menampilkan sinyal yang nantinya akan dikeluarkan oleh FPGA.
3. XILINX ISE 12.0, digunakan untuk mensintesa hasil pengkodean VHDL ke dalam board FPGA menjadi rangkaian ataupun gerbang-gerbang logika.
4. Altium Design, digunakan untuk melakuakn perancangan sirkuit untuk perancangan ADC.

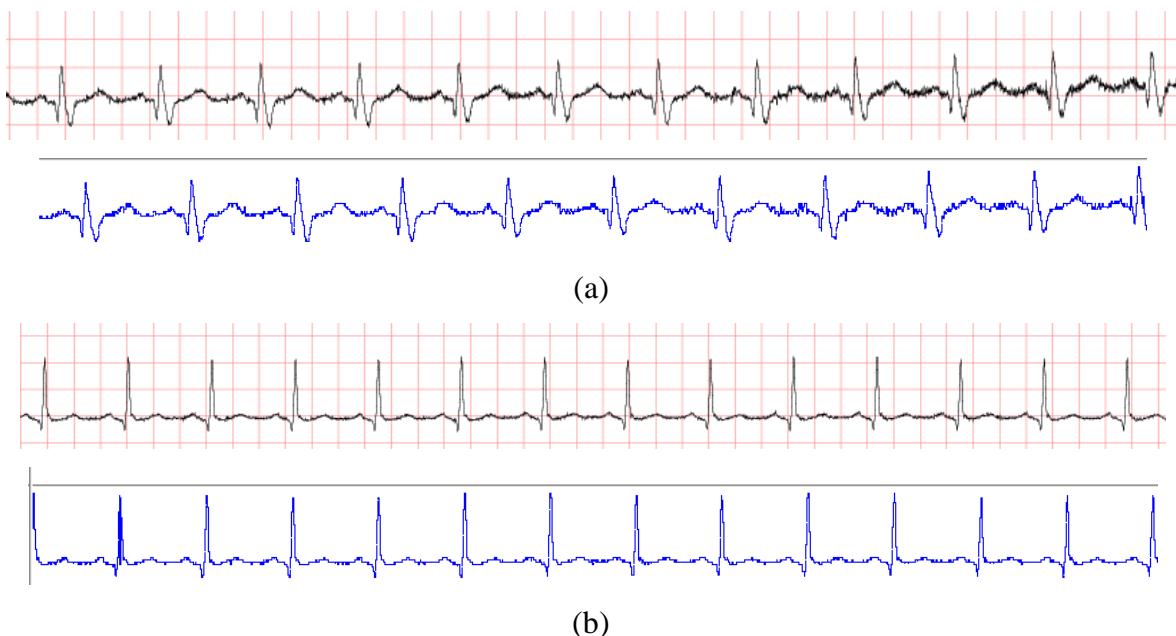
## **BAB IV**

### **PENGUJIAN DAN ANALISA SISTEM DETEKSI QRS**

Pada bab ini akan diberikan hasil pengujian dari sistem yang telah dirancang dan diimplementasikan pada bab sebelumnya. Pengujian terhadap sistem akan dilakukan menggunakan simulasi dengan software ModelSim 6.3f. Langkah pengujinya sendiri yaitu dengan memberikan masukan kepada masing-masing blok sistem berupa test bench yang merupakan hasil kuantisasi dan encoding dari sinyal ECG waktu diskrit dengan periode sampling 2 ms. Sinyal ECG yang digunakan sebagai masukan diambil dari PTB Diagnostic ECG database, MIT Arrhythmia Database, ST Petersburg Arrhythmia Database pada website [www.physionet.org](http://www.physionet.org) yang sudah menjadi standar penelitian.

#### **4.1 Perancangan Sinyal Test Bench**

Tahap pertama dalam pengujian sistem ini adalah merancang sinyal test bech yang sesuai dengan karakteristik sinyal ECG. Pada perancangan test bench ini, sinyal ECG yang diambil dari database dikalikan dengan 1000 untuk membulatkan hingga tidak ada angka pecahan. Kemudian dilakukan kuantisasi dengan membagi daya dari sinyal dengan 5000 sebagai asumsi nilai tertinggi yang mungkin muncul dan kemudian dikalikan dengan 128 untuk melakukan levelisasi terhadap sinyal dimana ada 128 level kuantisasi. Pada pengujian sistem ini akan dirancang 30 buah test bench sinyal ECG dengan masing-masing sinyal berdurasi 10 sekon.



**Gambar 4.1 Perbandingan sinyal asli dengan sinyal test bench**

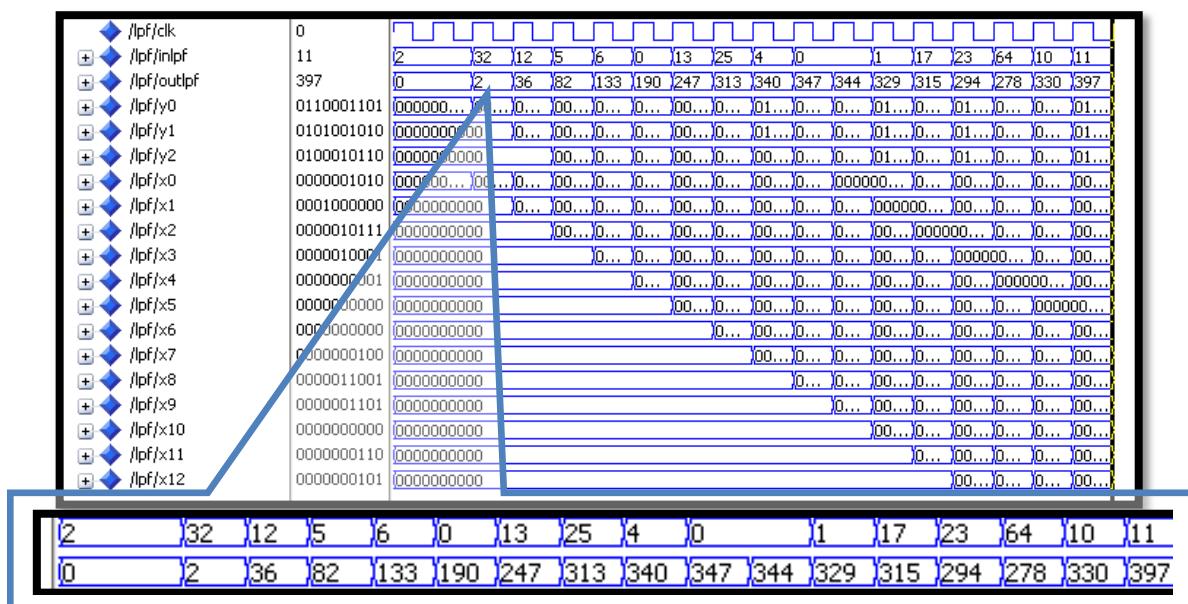
Sinyal test bench tersebut kemudian akan menjadi masukan dari sistem sebagai pengganti sinyal ECG.

## 4.2 Pengujian Blok-blok Sistem pada ModelSim

Setelah perancangan sinyal test bench tadi, dilakukan pengujian untuk sistem deteksi QRS dengan cara menganalisa keluaran masing-masing blok sistem. Pengujian dilakukan dengan menggunakan kedua macam sinyal testbench yang telah dibentuk tadi.

### 4.2.1 Pengujian Blok Sistem LPF

Pada bagian ini akan dilakukan pengujian terhadap blok program LPF yang telah dirancang dimana LPF akan diuji dengan diberi nilai masukan secara manual dengan bantuan software Microsoft Excel untuk kemudian dilihat hasil keluarannya dibandingkan dengan hasil perhitungan secara manual sesuai persamaan perbedaan LPF yang telah dijabarkan pada persamaan 3.1. Hasil penghitungan secara manual dapat dilihat pada table 4.1 dimana hasil lebih lengkapnya dapat dilihat pada lampiran B. Data masukan yang dihitung manual tadi dijadikan masukan pada blok program LPF dan disimulasikan pada software ModelSim untuk kemudian dibandingkan nilai keluaran yang dihasilkan lalu dibandingkan sebagai verifikasi bahwa blok program telah dapat berfungsi sebagai LPF sesuai persamaan yang diinginkan.



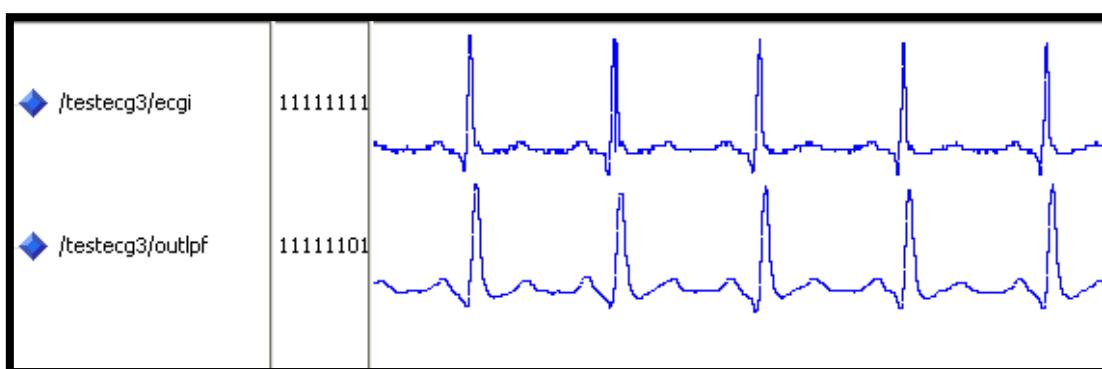
Gambar 4.2 Hasil Pengujian Manual pada Blok LPF

**Tabel 4.1 Hasil Penghitungan Secara Manual Untuk Keluaran LPF**

<b>x(n)</b>	<b>x(n-6)</b>	<b>x(n-12)</b>	<b>y(n-2)</b>	<b>y(n-1)</b>	<b>y(n)</b>	<b>y(n) keluaran blok LPF</b>
<b>2</b>	0	0	0	0	<b>2</b>	<b>2</b>
<b>32</b>	0	0	0	2	<b>36</b>	<b>36</b>
<b>12</b>	0	0	2	36	<b>82</b>	<b>82</b>
<b>5</b>	0	0	36	82	<b>133</b>	<b>133</b>
<b>6</b>	0	0	82	133	<b>190</b>	<b>190</b>
<b>0</b>	0	0	133	190	<b>247</b>	<b>247</b>
<b>13</b>	2	0	190	247	<b>313</b>	<b>313</b>
<b>25</b>	32	0	247	313	<b>340</b>	<b>340</b>
<b>4</b>	12	0	313	340	<b>347</b>	<b>347</b>
<b>0</b>	5	0	340	347	<b>344</b>	<b>344</b>
<b>0</b>	6	0	347	344	<b>329</b>	<b>329</b>
<b>1</b>	0	0	344	329	<b>315</b>	<b>315</b>
<b>17</b>	13	2	329	315	<b>294</b>	<b>294</b>
<b>23</b>	25	32	315	294	<b>278</b>	<b>278</b>
<b>64</b>	4	12	294	278	<b>330</b>	<b>330</b>
<b>10</b>	0	5	278	330	<b>397</b>	<b>397</b>

Setelah dilakukan pengujian secara manual, ternyata diperoleh hasil yang sama antara masukan yang dihitung secara manual sesuai persamaan perbedaan LPF dengan hasil keluaran dari blok program LPF untuk masing-masing kombinasi masukan yang sama sehingga dapat disimpulkan bahwa blok program LPF telah berfungsi sesuai persamaan yang diinginkan.

Selanjutnya pengujian dilakukan dengan melewatkannya sinyal testbench yang telah dirancang tadi untuk kemudian dilihat hasil keluarannya apakah sesuai dengan hasil yang diinginkan. Sistem LPF ini dirancang untuk menghilangkan noise-noise berfrekuensi tinggi, dengan frekuensi cut off filter 11 Hz. Pada simulasi menggunakan sinyal testbench maka diperoleh hasil keluaran dari filter seperti pada gambar 4.3.

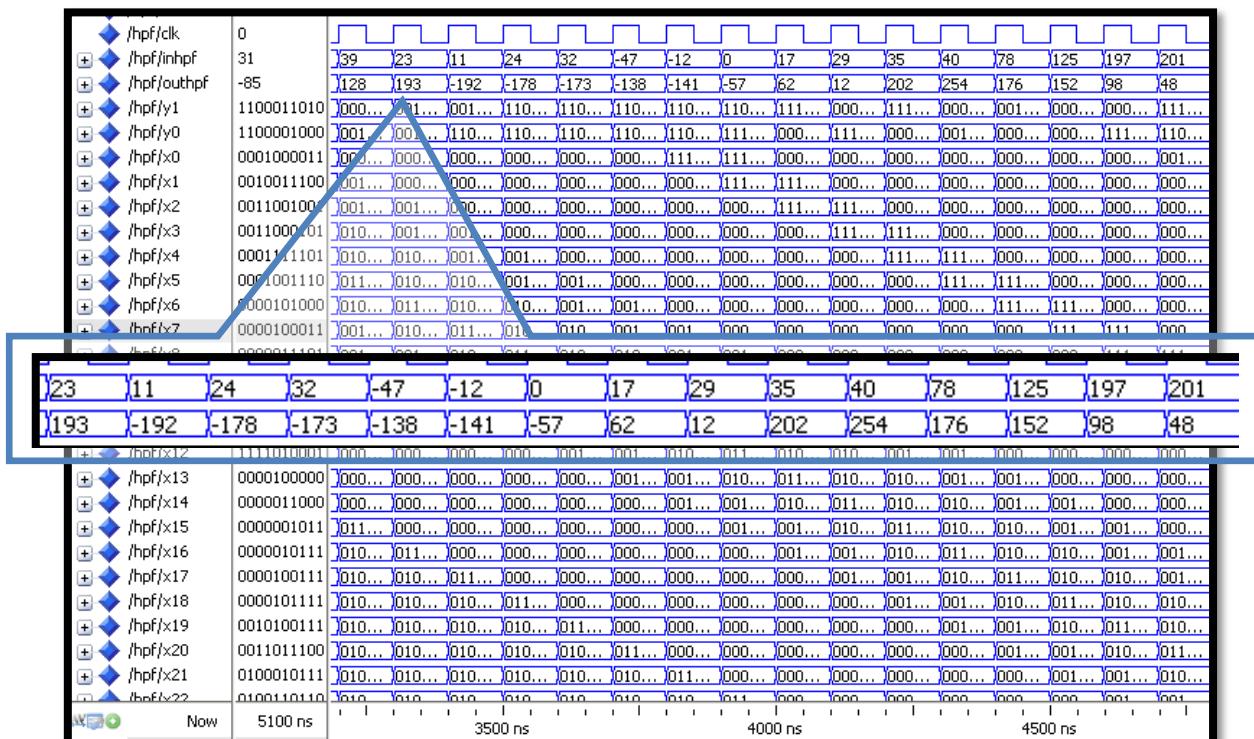
**Gambar 4.3 Test bech keluaran blok sistem LPF**

Dari hasil keluaran LPF dengan masukan test bench ECG, dapat terlihat bahwa noise berfrekuensi tinggi telah tereduksi sehingga pada sinyal ECG sudah jelas masing-masing gelombang P, Q, R, S dan T. Hal ini membuktikan bahwa LPF telah menjalankan fungsi dengan semestinya. Dari gambar 4.3 terlihat ada delay yang ditandai adanya pergeseran sinyal di domain waktu untuk sinyal masukan dengan sinyal keluaran dari LPF. Delay yang terjadi sebesar 6 clock atau 30 ms.

#### 4.2.2 Pengujian Blok Sistem HPF

Sinyal hasil keluaran blok LPF tadi selanjutnya akan menjadi masukan dari blok HPF dengan tujuan menghilangkan sinyal-sinyal berfrekuensi rendah seperti gelombang P dan T. Sebelum diimplementasikan, blok sistem HPF yang telah dirancang diujikan terlebih dahulu dengan masukan nilai acak dan dihitung secara manual sesuai persamaan 3.2. Kemudian nilai-nilai tersebut dijadikan nilai masukan sistem HPF untuk kemudian dibandingkan kesesuaian hasil perhitungan manual dan hasil keluaran blok program HPF untuk validasi bahwa blok program yang dirancang telah sesuai dengan persamaan perbedaan filter yang diinginkan.

Hasil perhitungan secara manual dapat dilihat pada table 4.2 yang lebih lengkapnya dapat dilihat pada lampiran B. Sedangkan keluaran dari program HPF yang dirancang dapat dilihat pada gambar 4.4.



Gambar 4.4 Hasil Pengujian Manual pada Blok HPF

**Tabel 4.2 Hasil Penghitungan Secara Manual Untuk Keluaran HPF**

$x(n)$	$x(n-16)$	$x(n-17)$	$x(n-32)$	$y(n-1)$	$y(n)$	$y(n)$ keluaran blok HPF
<b>39</b>	397	330	0	128.1875	<b>193.9688</b>	<b>193</b>
<b>23</b>	11	397	2	193.9688	<b>-192.688</b>	<b>-192</b>
<b>11</b>	24	11	36	-192.688	<b>-178.906</b>	<b>-178</b>
<b>24</b>	28	24	82	-178.906	<b>-173.094</b>	<b>-173</b>
<b>32</b>	59	28	133	-173.094	<b>-138.938</b>	<b>-138</b>
<b>-47</b>	49	59	190	-138.938	<b>-141.531</b>	<b>-141</b>
<b>-12</b>	125	49	247	-141.531	<b>-57.4375</b>	<b>-57</b>
<b>0</b>	235	125	313	-57.4375	<b>62.34375</b>	<b>62</b>
<b>17</b>	175	235	340	62.34375	<b>12.4375</b>	<b>12</b>
<b>29</b>	355	175	347	12.4375	<b>202.375</b>	<b>202</b>
<b>35</b>	397	355	344	202.375	<b>254.0313</b>	<b>254</b>
<b>40</b>	310	397	329	254.0313	<b>176.0625</b>	<b>176</b>
<b>78</b>	279	310	315	176.0625	<b>152.4688</b>	<b>152</b>
<b>125</b>	220	279	294	152.4688	<b>98.75</b>	<b>98</b>
<b>197</b>	167	220	278	98.75	<b>48.28125</b>	<b>48</b>
<b>201</b>	47	167	330	48.28125	<b>-67.6875</b>	<b>-67</b>
<b>156</b>	39	47	397	-67.6875	<b>-68.1563</b>	<b>-68</b>
<b>67</b>	23	39	11	-68.1563	<b>-85.9063</b>	<b>-85</b>
<b>31</b>	11	23	24	-85.9063	<b>-98.125</b>	<b>-98</b>

Dari perbandingan hasil perhitungan didapatkan nilai yang sama antara hasil perhitungan manual dengan nilai yang dihasilkan oleh program, namun hasil keluaran blok program mengalami pembulatan nilai dengan membuang semua nilai dibelakang koma. Hal ini terjadi karena pada program pembagian yang dilakukan yaitu pembagian dengan nilai 32 dilakukan secara pemotongan bit secara.

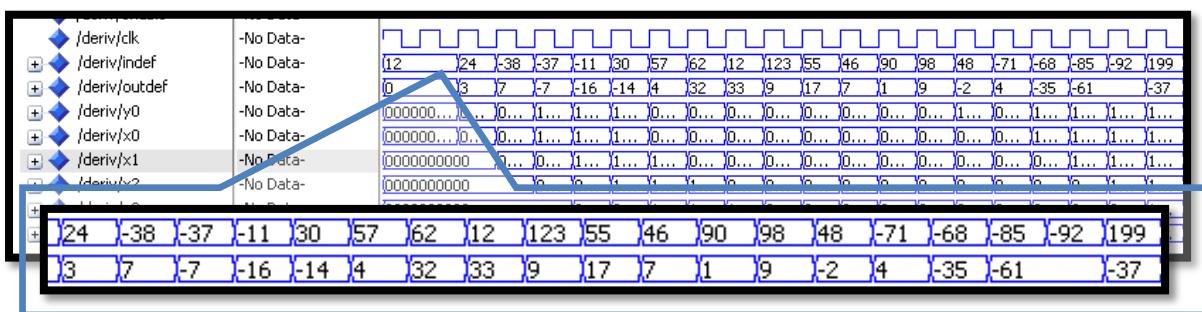
Kemudian blok HPF tersebut diuji dengan diberi masukan sinyal test bench. Hasil simulasi keluaran blok sistem HPF dapat dilihat pada gambar 4.5. Pada hasil simulasi terlihat bahwa terjadi pengurangan daya dari gelombang berfrekuensi rendah seperti gelombang P dan gelombang T. Selain itu terdapat pergeseran sinyal sebesar 14 clock atau 70 ms.

**Gambar 4.5 Test bench keluaran sistem HPF**

#### 4.2.3 Pengujian Blok Sistem Derivative

Setelah melewati blok HPF, sinyal kemudian menjadi masukan blok Derivative dimana sinyal akan mengalami differensiasi. Tujuan dari difensiasi ini adalah untuk memperjelas perbedaan dimana sinyal berdaya rendah akan direndam sebaliknya sinyal berdaya tinggi akan diperkuat sehingga kemiringan dari kompleks QRS menjadi lebih curam.

Pada bagian ini blok derivative akan dibandingkan antara hasil perhitungan secara manual sesuai persamaan 3.3 dengan hasil keluaran blok program untuk nilai-nilai masukan yang sama sebagai verifikasi bahwa sistem yang dirancang telah sesuai dengan persamaan yang diinginkan. Perbandingan hasil pengujian ini dapat dilihat pada tabel 4.3 sedangkan untuk hasil keluaran dari blok program derivative dapat dilihat pada gambar 4.6 dibawah ini.



Gambar 4.6 Hasil Pengujian Manual pada Blok HPF

Tabel 4.3 Hasil Penghitungan Secara Manual Untuk Keluaran Derivative

$x(n)$	$x(n-1)$	$x(n-2)$	$x(n-3)$	$x(n-4)$	$y(n)$	$y(n)$ keluaran blok derivative
12	0	0	0	0	3	3
24	12	0	0	0	7.5	7
-38	24	12	0	0	-6.5	-7
-37	-38	24	12	0	-15.5	-16
-11	-37	-38	24	12	-13.375	-14
30	-11	-37	-38	24	4.875	4
57	30	-11	-37	-38	32.125	32
62	57	30	-11	-37	33.25	33
12	62	57	30	-11	9.75	9
123	12	62	57	30	17.625	17
55	123	12	62	57	7.125	7
46	55	123	12	62	1.375	1
90	46	55	123	12	9.875	9
98	90	46	55	123	-1.875	-2
48	98	90	46	55	4.75	4
-71	48	98	90	46	-34.5	-35
-68	-71	48	98	90	-60.625	-61
-85	-68	-71	48	98	-60.25	37

Dari table 4.3 diatas didapatkan hasil bahwa hasil perhitungan secara manual untuk persamaan 3.3 menghasilkan nilai yang sama dengan keluaran blok sistem derivative yang dirancang. Hanya ada sedikit perbedaan yaitu masalah pembulatan hasil pada program yang menyebabkan tidak adanya bilangan dibelakang koma.

Selanjutnya blok derivative ini diuji dengan diberi masukan sinyal test bench untuk kemudian dilihat keluaran yang dihasilkannya. Hasil keluaran dari pengujian dengan sinyal test bench ini dapat dilihat pada gambar 4.7 berikut.



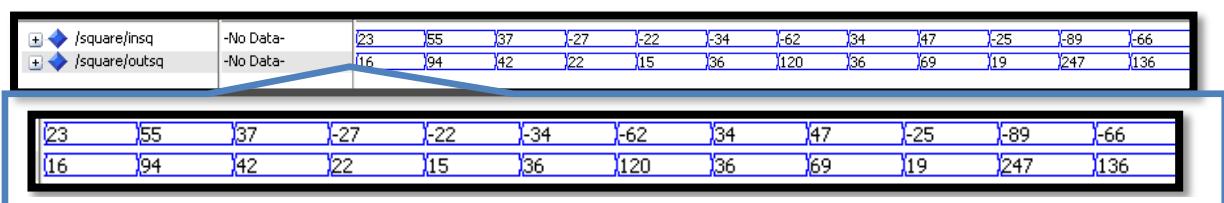
**Gambar 4.7 Test bench keluaran blok sistem derivative**

Test bench keluaran yang telah didiferensiasi mengalai penurunan daya pada sinyal berdaya rendah dan pada kompleks QRS terlihat adanya penguatan. Sistem ini menghasilkan delay sebanyak 5 clock atau 25 ms.

#### 4.2.4 Pengujian Blok Sistem Pengkuadrat

Sinyal hasil keluaran proses diferensiasi kemudian masukan blok sistem pengkuadratan dengan tujuan memperoleh nilai positif. Selain dikuadratkan pada blok ini sinyal pun mengalami pemotongan nilai, sebagai konsekuensi keterbatasan jumlah bit dimana operasi pengkuadratan akan menghasilkan output logic vektor berarray 20 sedangkan slot output yang tersedia hanya berarray 10.

Blok sistem pengkuadratan ini diverifikasi dengan membandingkan keluaran sistem dengan hasil penghitungan manual dimana operasi penghitungan yang dilakukan adalah pengkuadratan nilai masukan kemudian dibagi dengan 32 sebagai representasi digesernya nilai hasil pengkuadratan 5 bit ke arah LSB. Untuk hasil verifikasi lebih lengkapnya dapat dilihat pada gambar 4.8 dan table 4.4 dibawah ini.

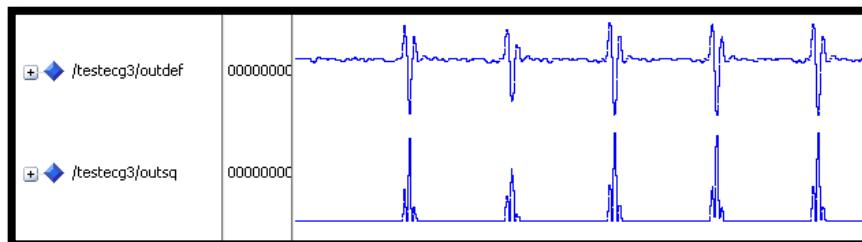


**Gambar 4.8 Hasil Pengujian Manual pada Blok Pengkuadrat**

**Tabel 4.4 Hasil Penghitungan Secara Manual Untuk Keluaran Blok Pengkuadrat**

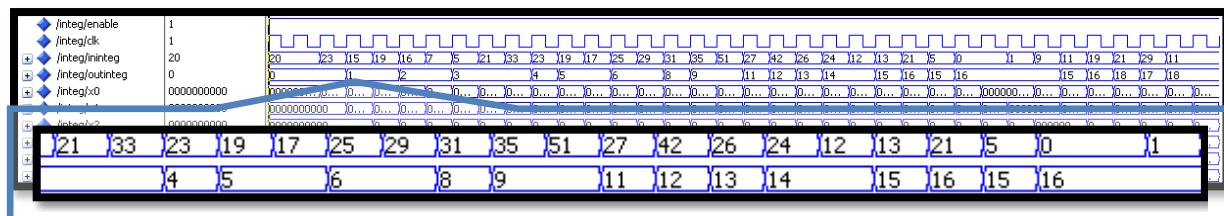
<b>x(n)</b>	<b>x2(n)</b>	<b>y(n)</b>	<b>y(n) keluaran blok pengkuadrat</b>
23	529	16.53125	16
55	3025	94.53125	94
37	1369	42.78125	42
-27	729	22.78125	22
-22	484	15.125	15
-34	1156	36.125	36
-62	3844	120.125	120
34	1156	36.125	36
47	2209	69.03125	69
-25	625	19.53125	19
-89	7921	247.5313	247
-66	4356	136.125	136

Dari hasil simulasi, terlihat bahwa setelah melewati blok ini tidak terdapat lagi sinyal dengan daya bernilai negative selain itu karena dilakukan pemotongan nilai sinyal, kini mayoritas dari sinyal bernilai 0 dan hanya pada puncak dari kawasan kompleks QRS yang memiliki nilai daya tinggi. Pada blok sistem ini relative tidak ada delay yang dihasilkan karena blok hanya berupa operator perkalian tanpa membutuhkan sistem memori.

**Gambar 4.9 Test bench keluaran blok sistem fungsi kuadrat**

#### 4.2.5 Pengujian Blok Sistem Integrator

Setelah melewati proses pengkuadratan nilai, sinyal ECG kemudian diintegrasikan dengan menjadi masukan blok integrator. Pada blok ini dibentuk kurva dari nilai keluaran-keluaran nilai hasil pengkuadratan.

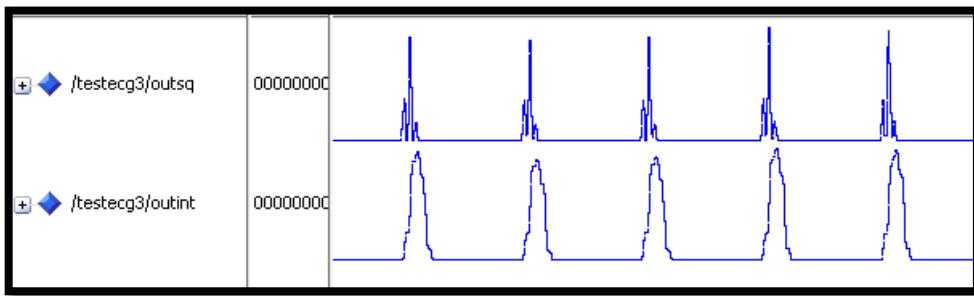
**Gambar 4.10 Hasil Pengujian Manual pada Blok Integrator**

**Tabel 4.5 Hasil Penghitungan Secara Manual Untuk Keluaran Blok Integrator**

x(n)	sum	y(n)	Y(n) keluaran blok integrator
21	126	3.9375	3
33	159	4.96875	3
23	182	5.6875	4
19	201	6.28125	5
17	218	6.8125	5
25	243	7.59375	6
29	272	8.5	6
31	303	9.46875	8
35	338	10.5625	9
51	389	11.15625	9
27	416	13	11
42	458	14.3125	12
26	484	15.125	13
24	508	15.875	14
12	520	16.25	14
13	533	16.65625	15
21	554	17.3125	16
5	559	17.46875	15
0	559	17.46875	16
0	559	17.46875	16
1	560	17.5	16

Pada table 4.5 dapat dilihat hasil verifikasi blok integrator dimana didalamnya dibandingkan hasil perhitungan manual dengan hasil keluaran blok dengan masukan yang sama. Terdapat perbedaan nilai pada keduanya, perbedaan tersebut dihasilkan dari pembulatan bilangan karena keterbatasan bit yang digunakan pada operasi ini, namun perbedaan tersebut hanya berkisar 1 hingga 2 sehingga dapat diabaikan karena tidak memberikan efek yang besar pada perhitungan keseluruhan. Maksimal perbedaan nilai akibat pembulatan yang dapat terjadi adalah hingga 4, hal ini karena pembagian terhadap nilai 32 yang dibagi menjadi 4 kali.

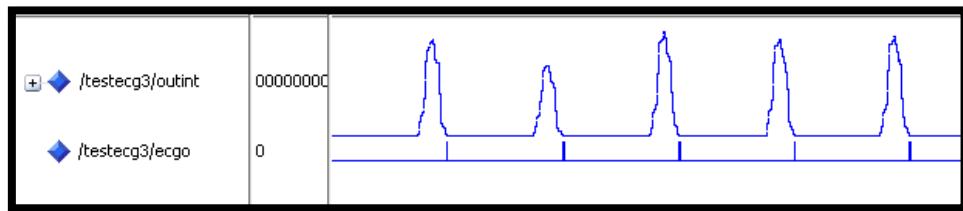
Setelah melalui proses verifikasi, kemudain blok integrator diujikan dengan diberi masukan testbench yang telah dibuat, Blok integrator sendiri memiliki fungsi untuk membentuk kurva kecenderungan nilai dari keluaran blok derivative. Kurva ini memiliki informasi kecenderungan nilai daya sinyal dari masukan-masukan sebelumnya. Blok ini memerlukan waktu operasi sebesar 7 clock sehingga menghasilkan delay sebesar 35 ms.



Gambar 4.11 Test bench keluaran blok sistem integrator

#### 4.2.6 Pengujian Blok Sistem Decision

Pada blok decision ini dilakukan pembatasan nilai minimum yang dideteksi sebagai kompleks QRS atau bukan. Sistem yang diterapkan oleh penulis, yaitu dengan menjumlahkan 32 nilai masukan terakhir dan nilai tersebut baru dikeluarkan setelah masukan bernilai 0 yang artinya kurva dari sinyal berakhir. Sistem semacam ini akan menyebabkan fleksibilitas yang tinggi untuk memberikan nilai pembatas (threshold) dan juga peluang yang sangat kecil untuk kompleks QRS terdeteksi dua kali namun kelemahan dari sistem ini adalah delay menjadi besar karena kompleks QRS baru nyatakan terdeteksi apabila kurva kawasan kompleks QRS telah melewati sistem seluruhnya. Delay yang dihasilkan oleh sistem sekitar 16 clock atau 80 ms yang dihitung dari mulai puncak R (puncak daya kurva) hingga kompleks QRS dinyatakan terdeteksi.

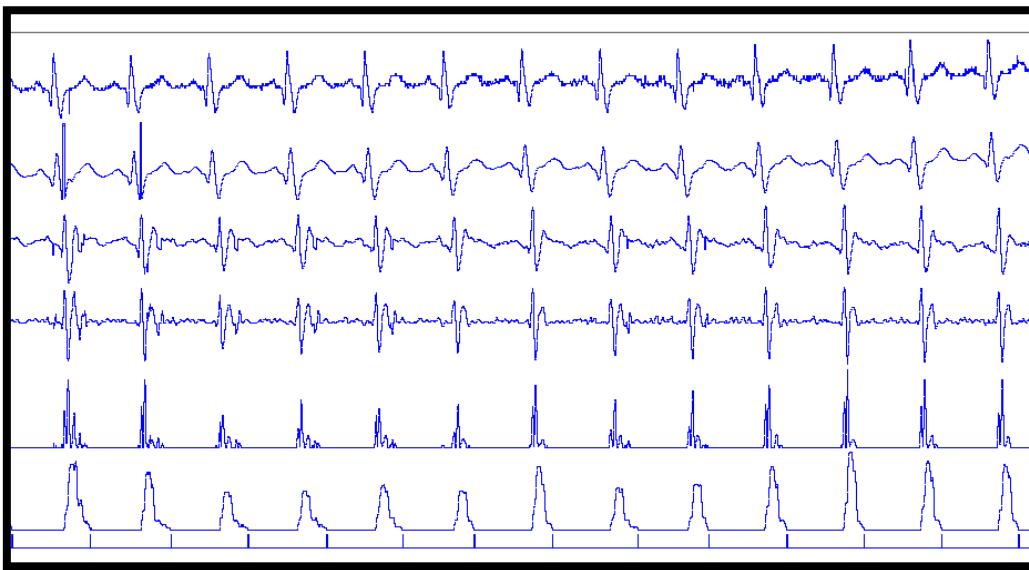


Gambar 4.12 Test bench keluaran blok sistem decision

### 4.3 Analisa Kinerja Sistem Keseluruhan

Secara keseluruhan dari dua sinyal test bench yang diujikan sistem mampu mendeteksi seluruh kompleks QRS yang ada. Namun yang diujikan adalah sinyal ECG hasil sampling yang memiliki noise relative kecil bila dibandingkan dengan sinyal ECG yang langsung diambil dari hasil pencuplikan pada tubuh bergantung pada kualitas electrocardiograf yang digunakan. Sehingga diperlukan pengujian kembali secara real time melalui implementasi perangkat keras yang akan dibahas pada bab selanjutnya. Dari hasil pengujian ini disimpulkan bahwa sistem dapat diimplementasikan pada perangkat keras untuk kemudian

dilakukan pengujian lebih lanjut lagi secara real time. Secara keseluruhan sistem menghasilkan processing time delay sebesar rata-rata 48 clock dimulai dari terjadinya puncak gelombang R hingga terdeteksinya QRS.



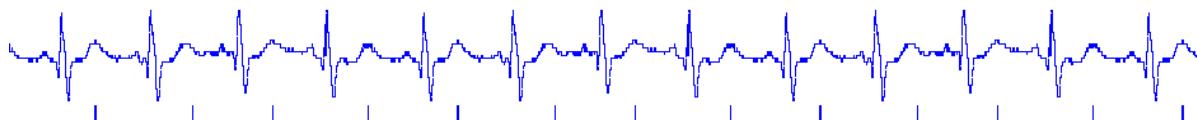
**Gambar 4.13 Perbandingan keluaran untuk setiap blok sistem**

Hasil simulasi secara test bench ini menunjukan bahwa sistem memiliki akurasi 100 % namun hal ini baru sebatas pengujian terbatas dan hanya sebagai parameter apakah sistem layak diimplementasikan pada perangkat keras apa tidak. Dari hasil pengujian ini disimpulkan bahwa sistem dapat diimplementasikan pada perangkat keras untuk kemudian dilakukan pengujian lebih lanjut lagi secara real time. Secara keseluruhan sistem menghasilkan processing time delay sebesar rata-rata 49 clock dimulai dari terjadinya puncak gelombang R hingga terdeteksinya QRS.

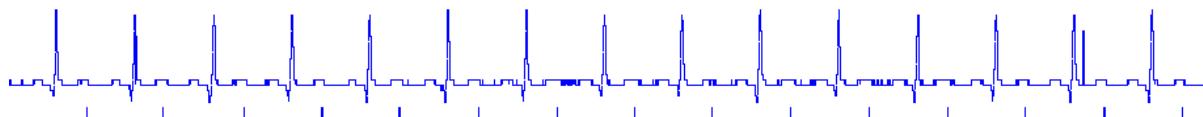
**Tabel 4.6 Processing Time Delay yang dihasilkan masing-masing blok sistem**

Blok Sistem	Total Delay (clock)	Total Delay (ms)
LPF	6	30
HPF	14	70
Derivative	5	25
Squaring	1	5
Integration	7	35
Decision	16	80
<b>Total</b>	<b>49</b>	<b>245</b>

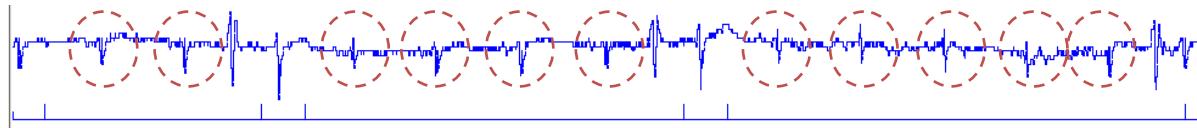
Selanjutnya dilakukan pengujian terhadap sistem dengan memberikan masukan beberapa jenis sinyal ECG untuk masing-masing kondisi jantung yang berbeda. Hal ini dilakukan untuk pengujian lebih lanjut akan kelayakan sistem serta menganalisa setiap keluaran yang dihasilkan dibandingkan dengan karakteristik masing-masing masukan.



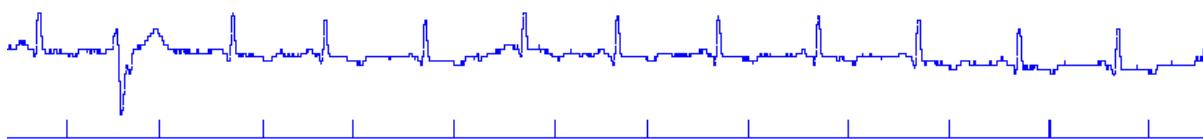
Gambar 4.14 Hasil pengujian sinyal ECG normal 1<sup>[10]</sup>



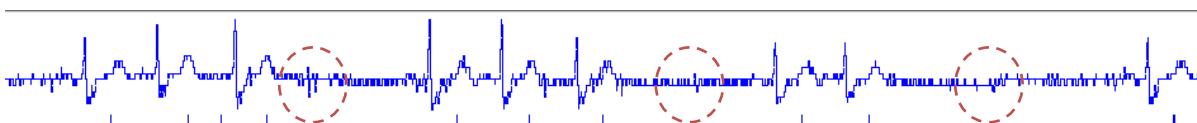
Gambar 4.15 Hasil pengujian sinyal ECG normal 2<sup>[11]</sup>



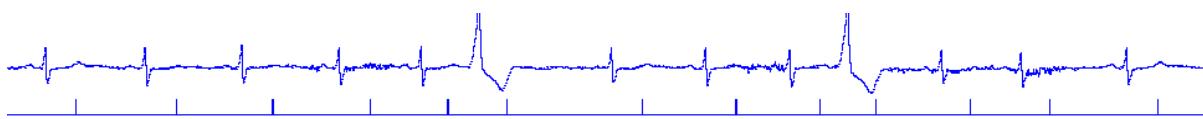
Gambar 4.16 Hasil pengujian sinyal ECG arrhythmia 1<sup>[12]</sup>



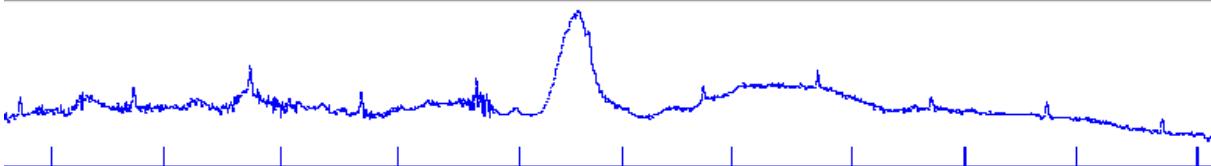
Gambar 4.17 Hasil pengujian sinyal ECG arrhythmia 2<sup>[13]</sup>



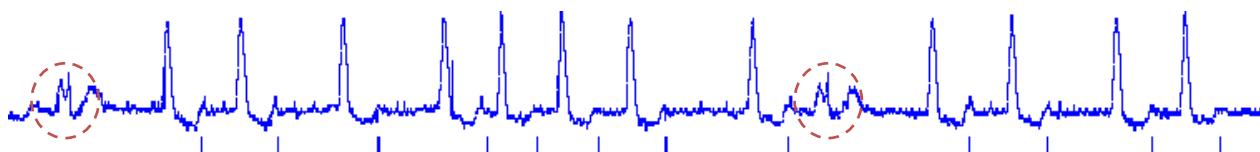
Gambar 4.18 Hasil pengujian sinyal ECG arrhythmia 3<sup>[14]</sup>



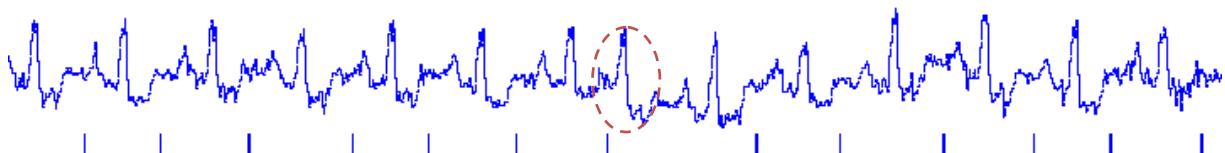
Gambar 4.19 Hasil pengujian sinyal ECG arrhythmia 4<sup>[15]</sup>



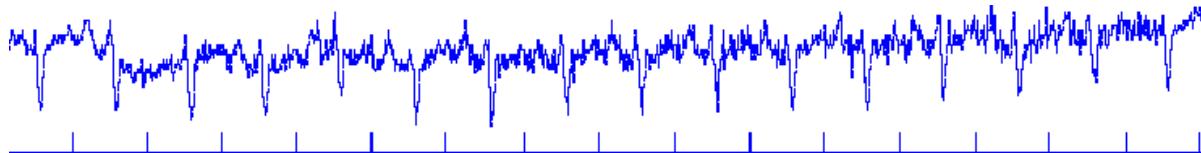
Gambar 4.20 Hasil pengujian sinyal ECG arrhythmia 5<sup>[16]</sup>



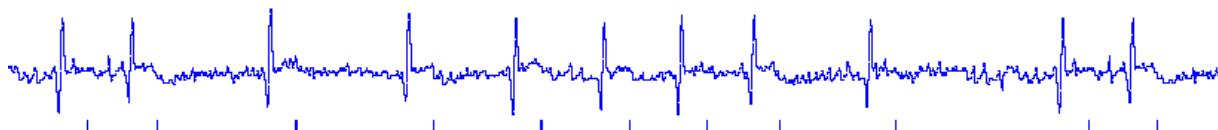
Gambar 4.21 Hasil pengujian sinyal ECG arrhythmia 6 <sup>[17]</sup>



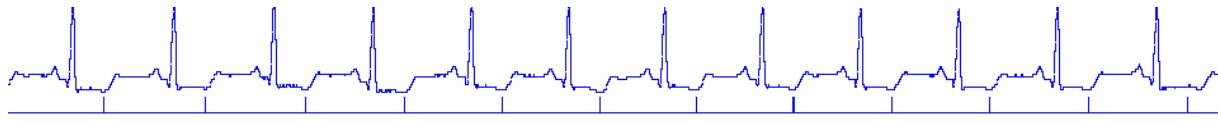
Gambar 4.22 Hasil pengujian sinyal ECG arrhythmia 7 <sup>[18]</sup>



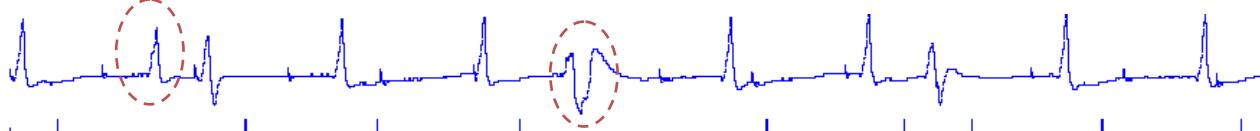
Gambar 4.23 Hasil pengujian sinyal ECG arrhythmia 8 <sup>[19]</sup>



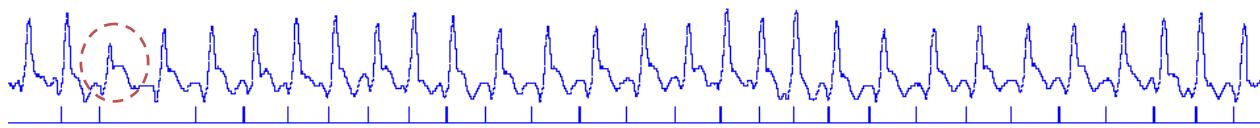
Gambar 4.24 Hasil pengujian sinyal ECG arrhythmia 9 <sup>[20]</sup>



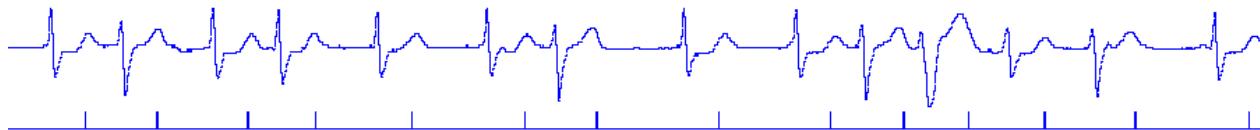
Gambar 4.25 Hasil pengujian sinyal ECG arrhythmia 10 <sup>[21]</sup>



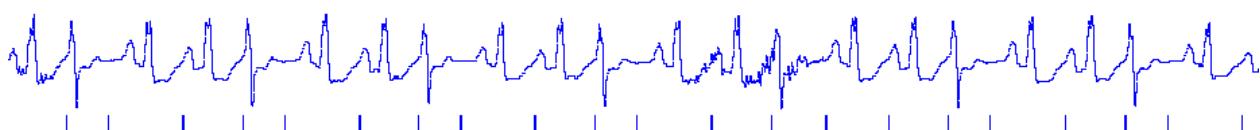
Gambar 4.26 Hasil pengujian sinyal ECG arrhythmia 11 <sup>[22]</sup>

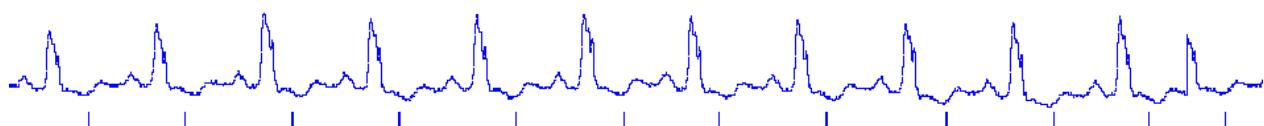
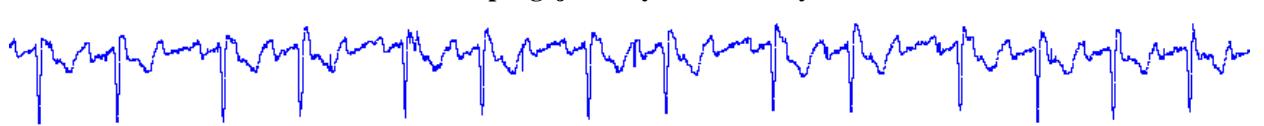
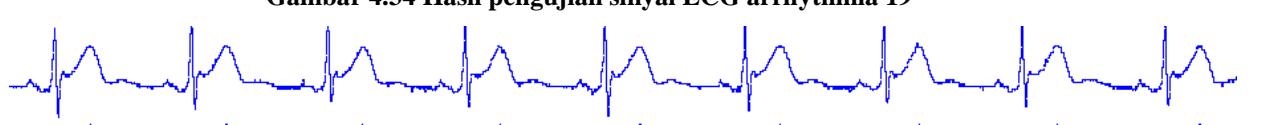
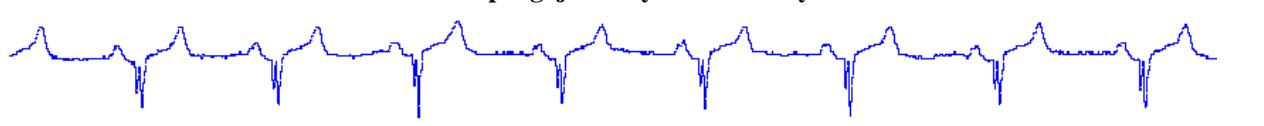
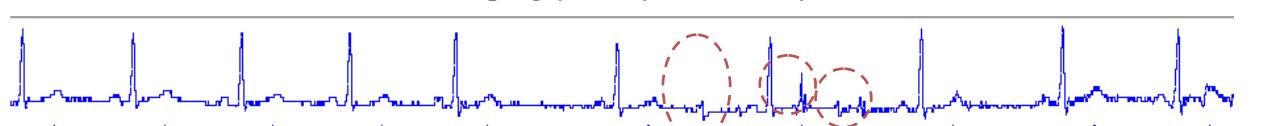


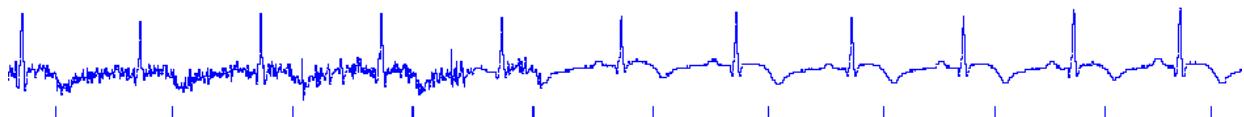
Gambar 4.27 Hasil pengujian sinyal ECG arrhythmia 12 <sup>[23]</sup>



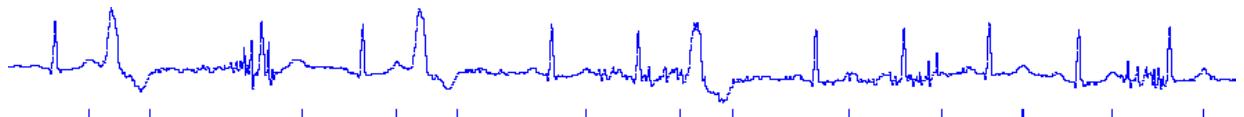
Gambar 4.28 Hasil pengujian sinyal ECG arrhythmia 13 <sup>[24]</sup>



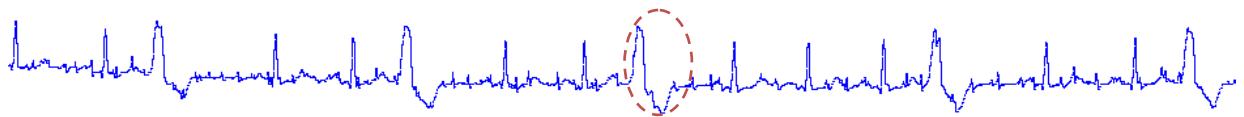
**Gambar 4.29 Hasil pengujian sinyal ECG arrhythmia 14** [25]**Gambar 4.30 Hasil pengujian sinyal ECG arrhythmia 15** [26]**Gambar 4.31 Hasil pengujian sinyal ECG arrhythmia 16** [27]**Gambar 4.32 Hasil pengujian sinyal ECG arrhythmia 17** [28]**Gambar 4.33 Hasil pengujian sinyal ECG arrhythmia 18** [29]**Gambar 4.34 Hasil pengujian sinyal ECG arrhythmia 19** [30]**Gambar 4.35 Hasil pengujian sinyal ECG arrhythmia 20** [31]**Gambar 4.36 Hasil pengujian sinyal ECG arrhythmia 21** [32]**Gambar 4.37 Hasil pengujian sinyal ECG arrhythmia 22** [33]**Gambar 4.38 Hasil pengujian sinyal ECG arrhythmia 23** [34]



Gambar 4.39 Hasil pengujian sinyal ECG arrhythmia 24<sup>[35]</sup>



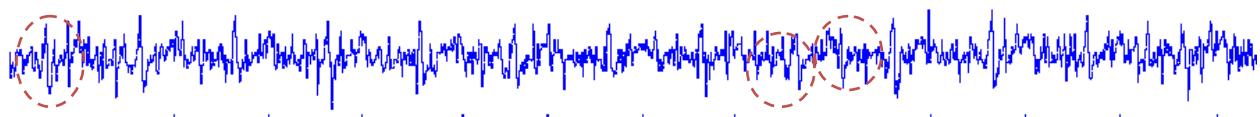
Gambar 4.40 Hasil pengujian sinyal ECG arrhythmia 25<sup>[36]</sup>



Gambar 4.41 Hasil pengujian sinyal ECG arrhythmia 26<sup>[37]</sup>



Gambar 4.42 Hasil pengujian sinyal ECG arrhythmia 27<sup>[38]</sup>



Gambar 4.43 Hasil pengujian sinyal ECG arrhythmia 28<sup>[39]</sup>

Hasil simulasi secara test bench ini menunjukkan bahwa sistem memiliki akurasi 100 % untuk sinyal ECG pada kondisi normal, sedangkan namun untuk sinyal ECG pada kondisi jantung arrhythmia beberapa kali terjadi kondisi dimana kompleks QRS tidak terdeteksi terutama untuk kompleks QRS yang berdaya lemah.dari total kompleks QRS yang ada sebanyak 406 yang berhasil tedeteksi adalah sebanyak 377 atau sekitar 92%. Hasil pengukuran selengkapnya dapat dilihat pada table 4.7.

**Tabel 4.7 Hasil pengujian sistem deteksi QRSpada ModelSim**

No	Jenis Sinyal ECG	Jumlah QRS	Jumlah Terdeteksi
1	Normal 1	13	13
2	Normal 2	15	15
3	Arrhythmia 1	17	6
4	Arrhythmia 2	12	12
5	Arrhythmia 3	12	9
6	Arrhythmia 4	13	13
7	Arrhythmia 5	11	11
8	Arrhythmia 6	14	12
9	Arrhythmia 7	14	13
10	Arrhythmia 8	16	16
11	Arrhythmia 9	11	11
12	Arrhythmia 10	12	12
13	Arrhythmia 11	11	9
14	Arrhythmia 12	28	27
15	Arrhythmia 13	14	14
16	Arrhythmia 14	21	21
17	Arrhythmia 15	12	12
18	Arrhythmia 16	15	15
19	Arrhythmia 17	12	12
20	Arrhythmia 18	14	14
21	Arrhythmia 19	11	11
22	Arrhythmia 20	9	9
23	Arrhythmia 21	8	8
24	Arrhythmia 22	12	10
25	Arrhythmia 23	13	10
26	Arrhythmia 24	11	11
27	Arrhythmia 25	13	13
28	Arrhythmia 26	16	15
29	Arrhythmia 27	12	12
30	Arrhythmia 28	14	11
<b>Total</b>		<b>406</b>	<b>377</b>

## **BAB V**

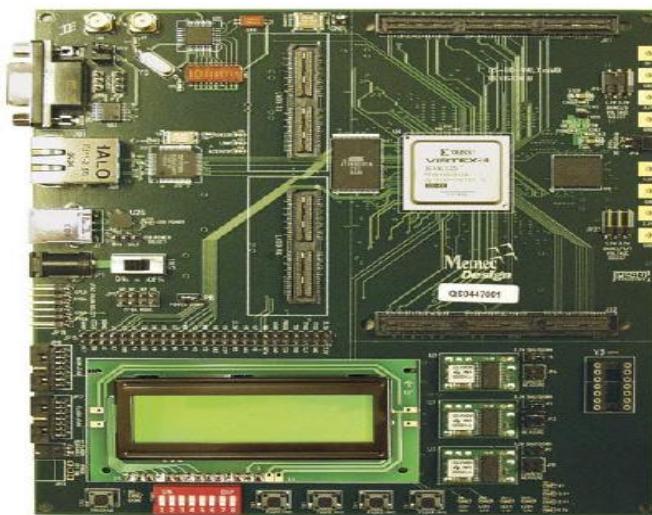
### **IMPLEMENTASI PERANGKAT KERAS**

### **SISTEM DETEKSI QRS PADA FPGA**

#### **5.1 Perangkat Keras Virtex-4 FPGA XC4VLX25-FF668**

FPGA atau *Field Programmable Gate Array* adalah sebuah komponen elektronika semikonduktor yang mempunyai komponen gerbang terprogram dan sambungan terprogram yang sering digunakan untuk mengimplementasikan rangkaian digital. FPGA merupakan *Programmable Logic Device* (PLD) yang dibangun dari sekumpulan sel fungsi logika dasar yang dapat diprogram. Sel-sel logika ini terhubung satu sama lain melalui suatu jaringan interkoneksi yang juga dapat diprogram. Penyusun dasar FPGA adalah CLB (*Configurable Logic Block*) dan LUT (*Look-Up Table*).

Pada perancangan ini digunakan FPGA seri Virtex-4 Xilinx XC4VLX25-FF668-10 dengan spesifikasi 64-MB DDR SDRAM, 32 bit *interface running*, kecepatan data 266 MHz, *clock oscillator* 100 MHz, dan fitur-fitur pendukung lainnya. Pada *board* ini juga tersedia port *input/output* dan juga dilengkapi dengan LCD panel. Berikut adalah gambar perangkat keras Virtex-4 FPGA XC4VLX25.

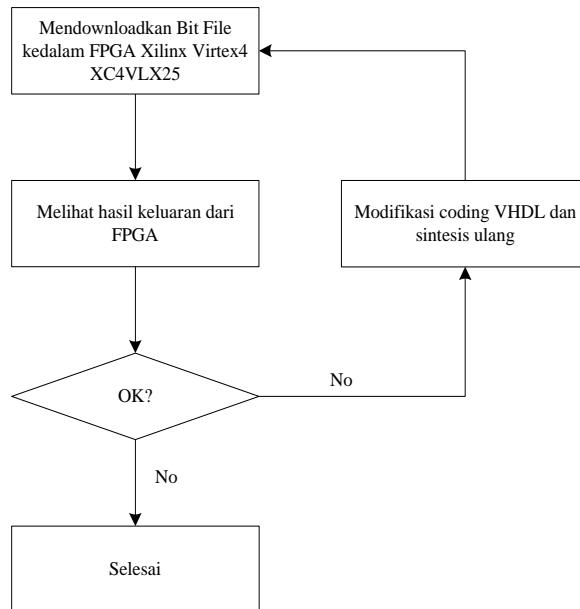


Gambar 5.1 Perangkat Keras Virtex-4 FPGA XC4VLX25

## 5.2 Implementasi Sistem Deteksi QRS

Implementasi dilakukan dengan menanamkan modul yang dirancang pada MODELSIM ke dalam FPGA menggunakan *software* Xilinx 12i.

Langkah-langkah implementasi dapat dilihat pada diagram alir berikut.



**Gambar 5.2 Diagram Alir implementasi Rangkaian pada FPGA**

Tahapan-tahapan yang dilakukan yaitu :

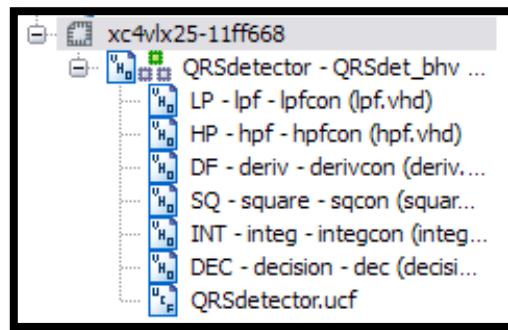
1. *Design Entry*, yaitu menambahkan sumber kode VHDL yang telah dirancang ke dalam Xilinx 12i.
2. *Assigned Package Pin*, yaitu memetakan port input/output pada FPGA yang akan digunakan.
3. *Synthesize*, yaitu menerjemahkan kode VHDL ke dalam daftar jaringan (netlist) dan gerbang dasar pada FPGA.
4. *Implement Design*, dengan cara :
  - a. *Translate*, yaitu menerjemahkan netlist dan blok dasar berdasarkan constraint-nya.
  - b. *Map*, yaitu pemetaan rangkaian ke blok Xilinx FPGA, gerbang logika dan distribusinya.

- c. *Place and Route*, yaitu penempatan hasil pemetaan pada blok dan gerbang yang dimaksud pada *floorplan*.

*Generate Programming File*, yaitu membuat file **.bit** yang akan dikirimkan pada FPGA melalui JTAG.

### 5.2.1 Design Entry

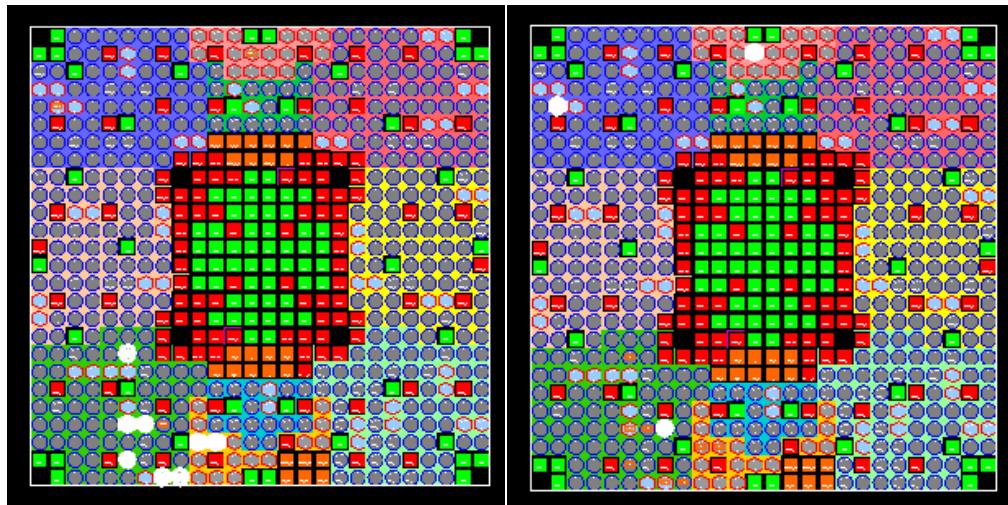
Pada tahap ini, seluruh program VHDL penyusun sistem deteksi QRS dimasukkan dalam IC XC4VLX25 dengan menggunakan *software* Xilinx. Program VHDL tersebut adalah program yang akan *be generate* dan ditanamkan dalam IC XC4VLX25 seperti terlihat pada gambar dibawah ini.



Gambar 5.3 Design Entry Rangkaian

### 5.2.2 Assigned Package PIN

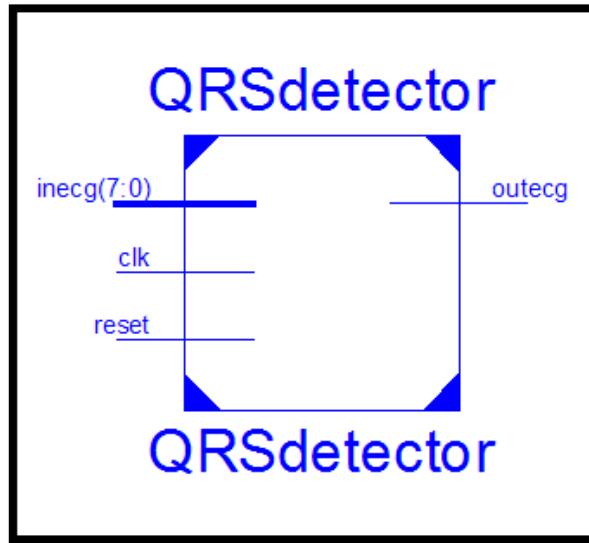
Tahap ini adalah memetakan port input/output modul yang dirancang pada port input/output FPGA.



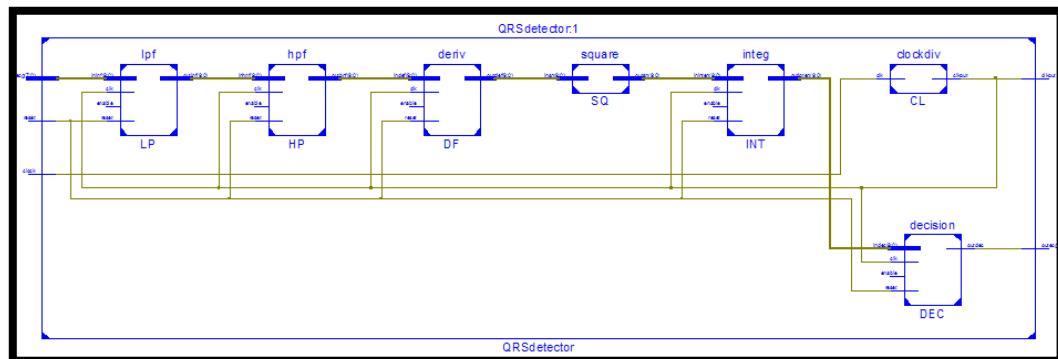
Gambar 5.4 Tampilan package dari pin-pin masukan (kiri) dan keluaran (kanan) pada FPGA

### 5.2.3 Sintesis Rangkaian

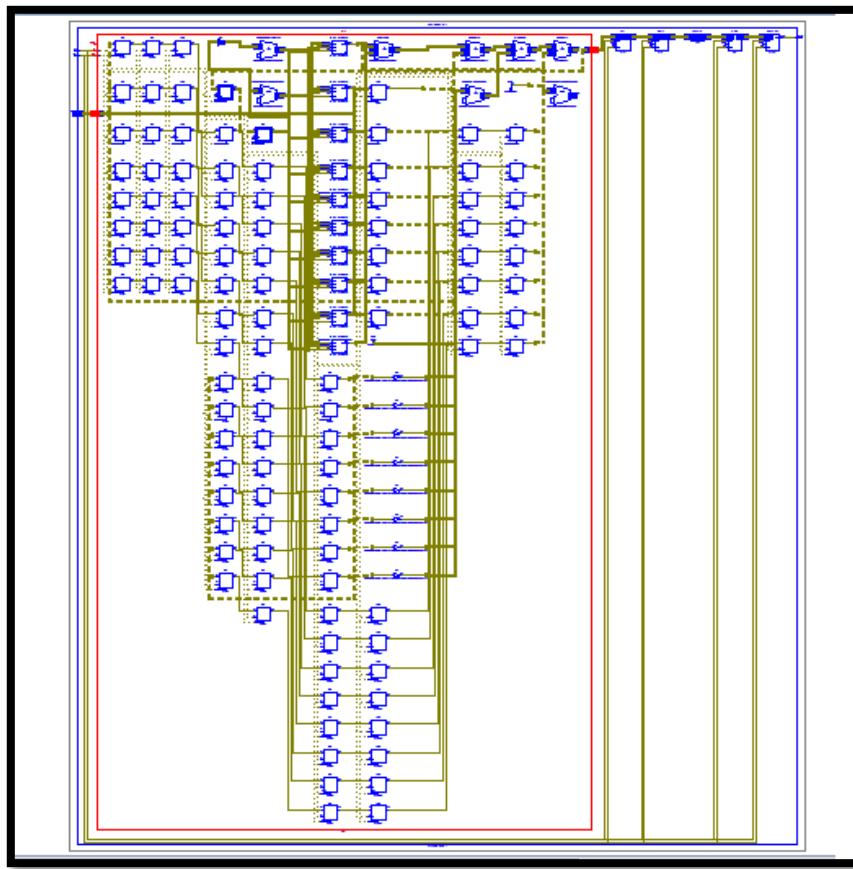
Dalam tahap ini, modul yang dirancang dapat diterjemahkan dalam netlist dan gerbang logika pada FPGA. Untuk lebih jelasnya terlihat pada gambar berikut.



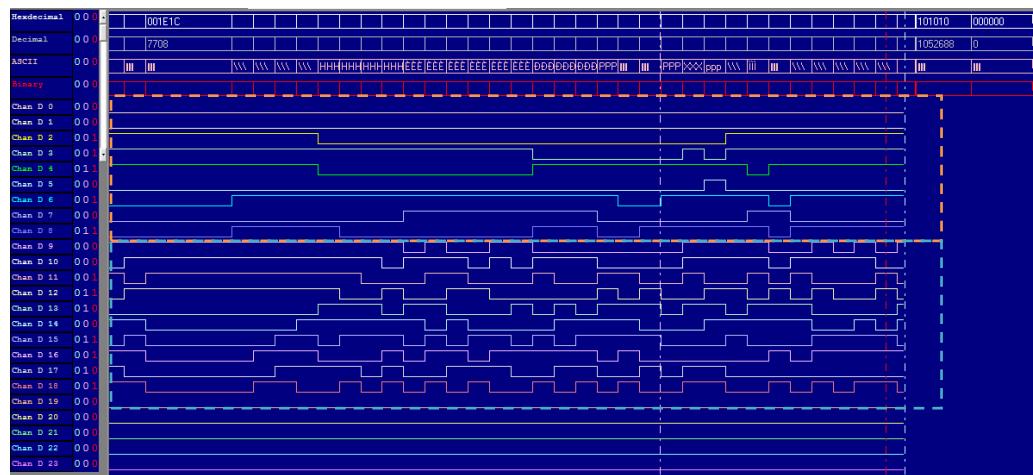
Gambar 5.5 Tampilan blok sistem deteksi QRS hasil sintesa



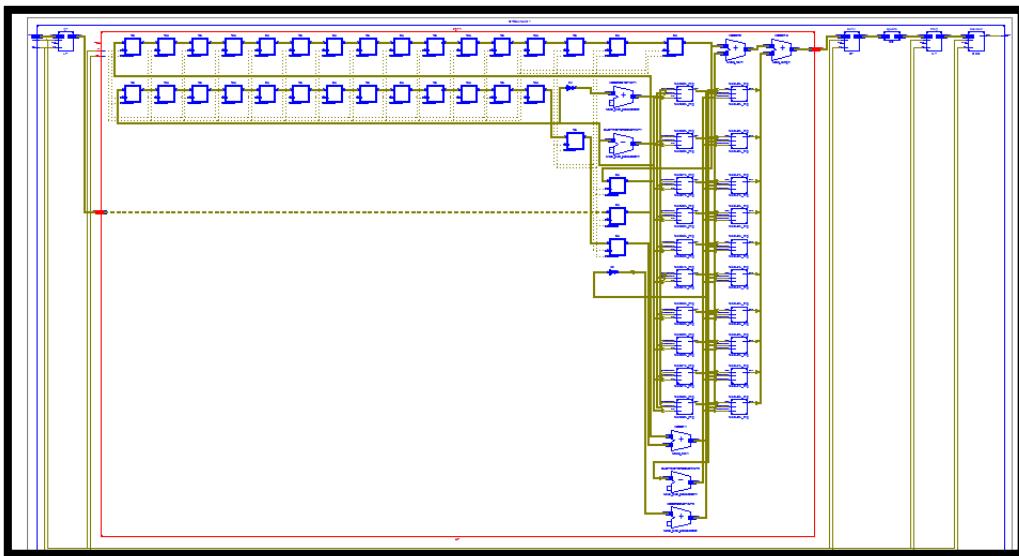
Gambar 5.6 Tampilan blok inti sistem deteksi QRS



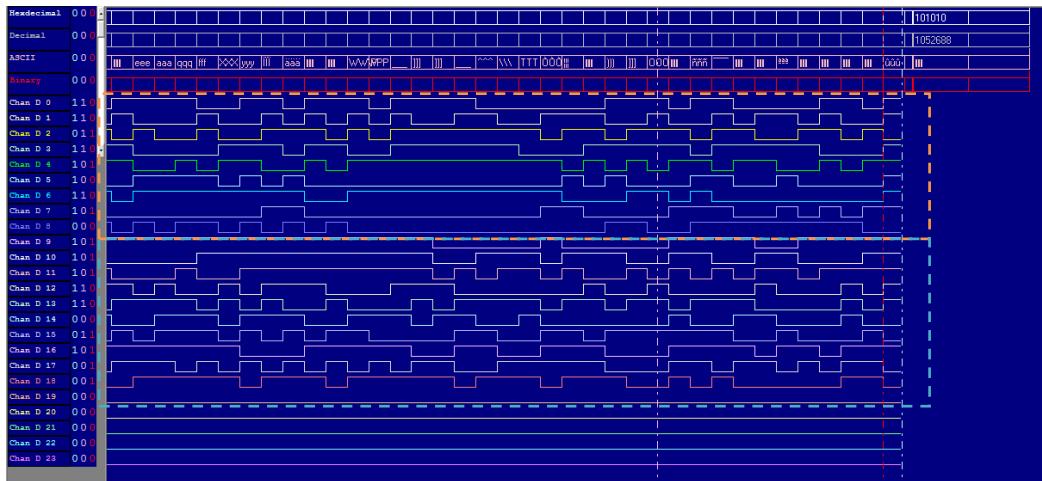
Gambar 5.7 Tampilan blok LPF sistem deteksi QRS



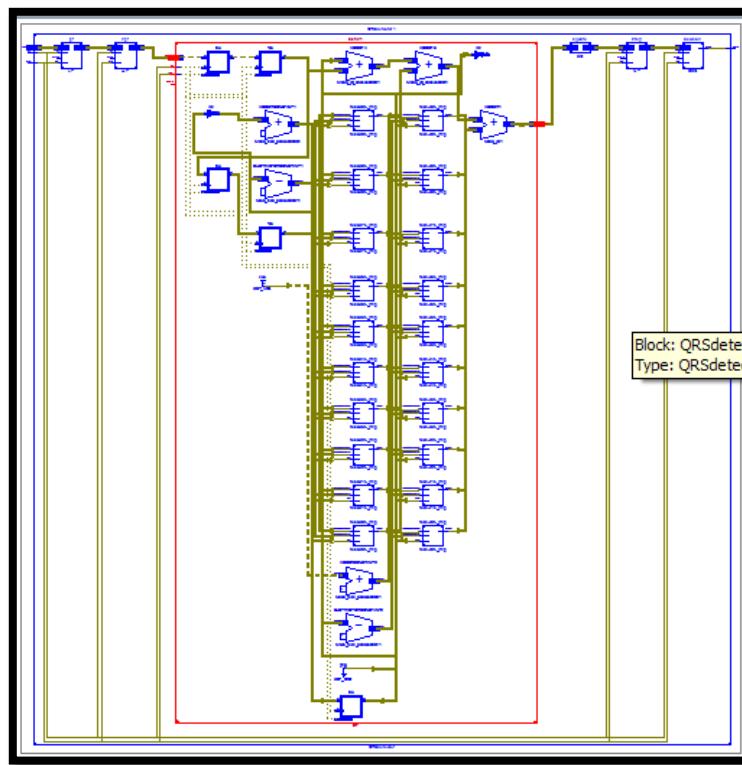
Gambar 5.8 Perbandingan masukan dan keluaran implementasi blok LPF



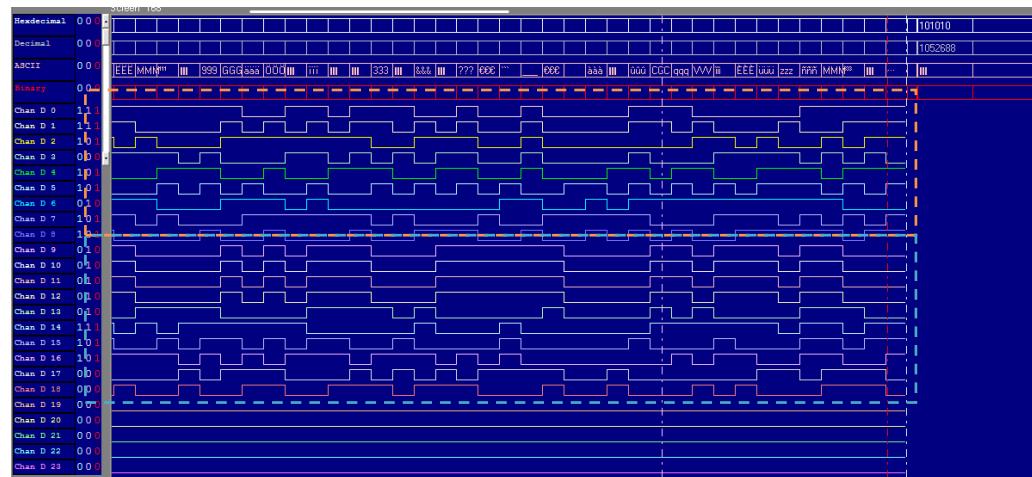
Gambar 5.9 Tampilan blok HPF sistem deteksi QRS



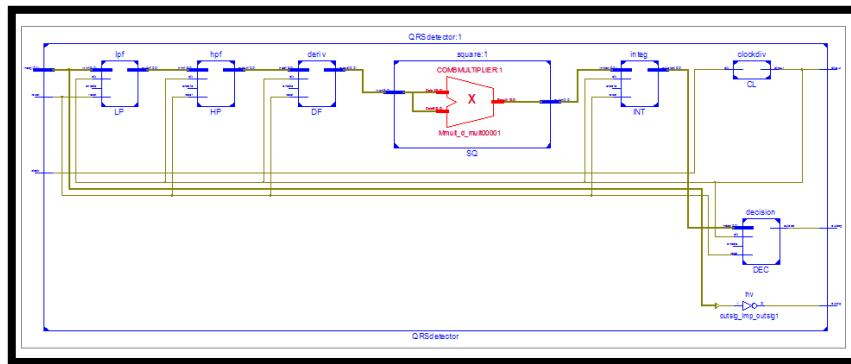
Gambar 5.10 Perbandingan masukan dan keluaran implementasi blok HPF



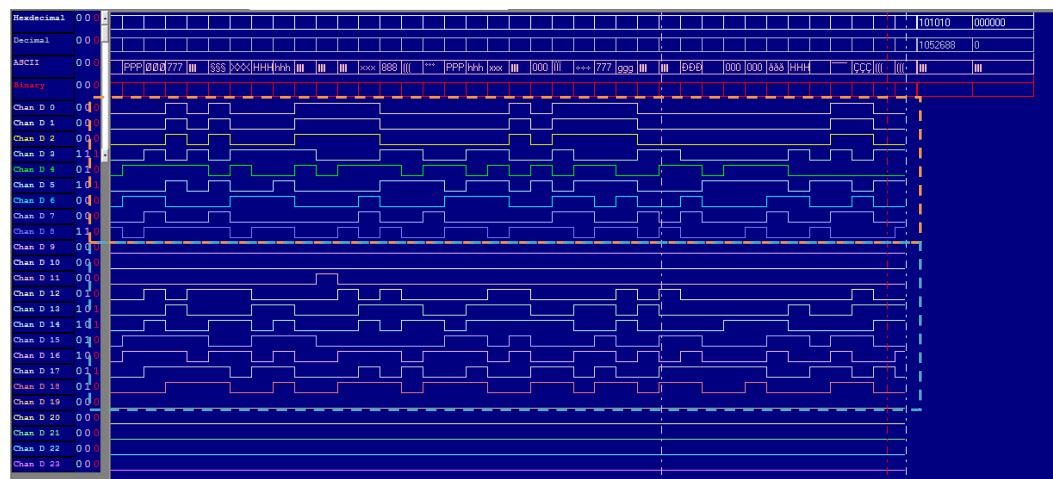
Gambar 5.11 Tampilan blok derivative sistem deteksi QRS



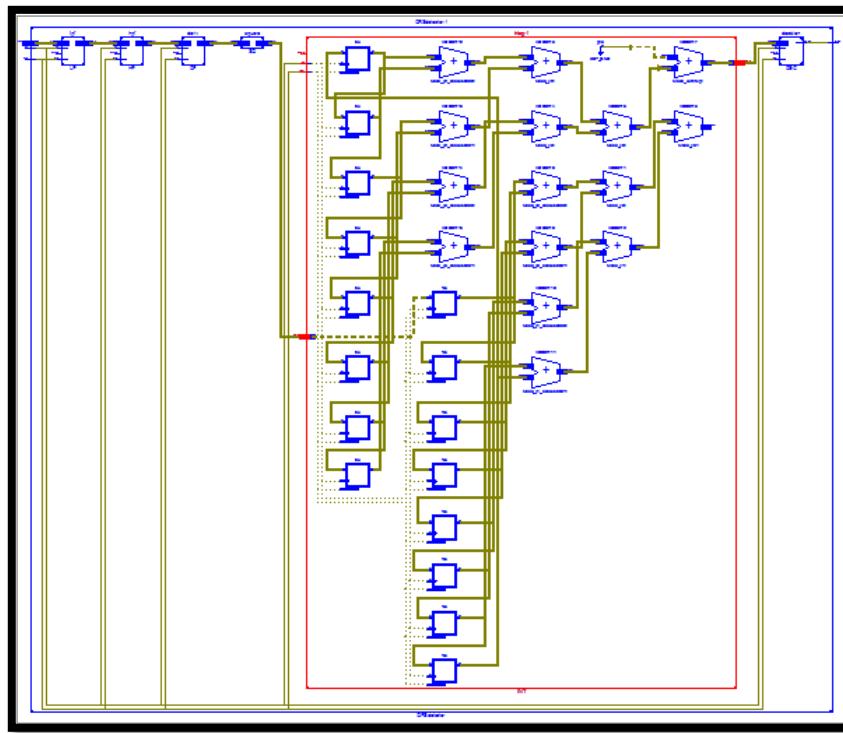
Gambar 5.12 Perbandingan masukan dan keluaran implemtasi blok derivative



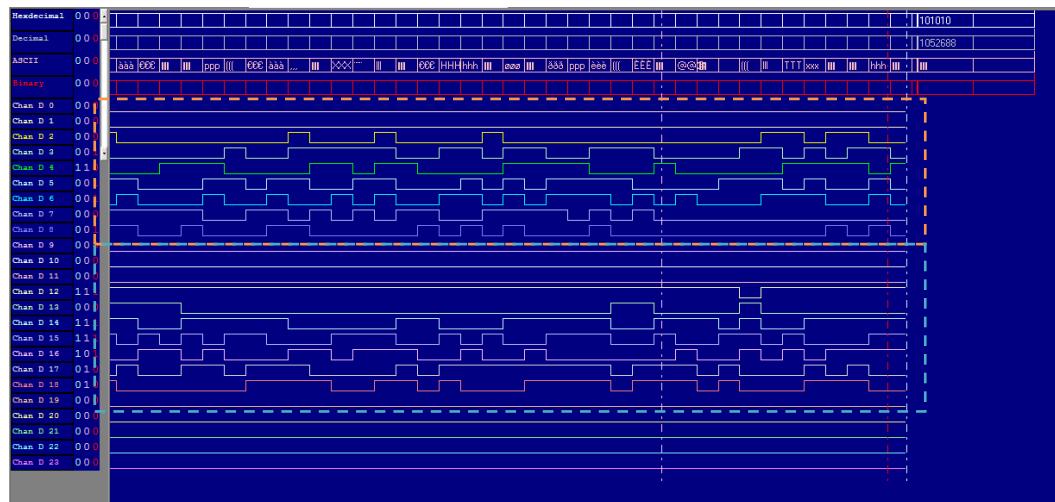
Gambar 5.13 Tampilan blok pengkuadrat sistem deteksi QRS



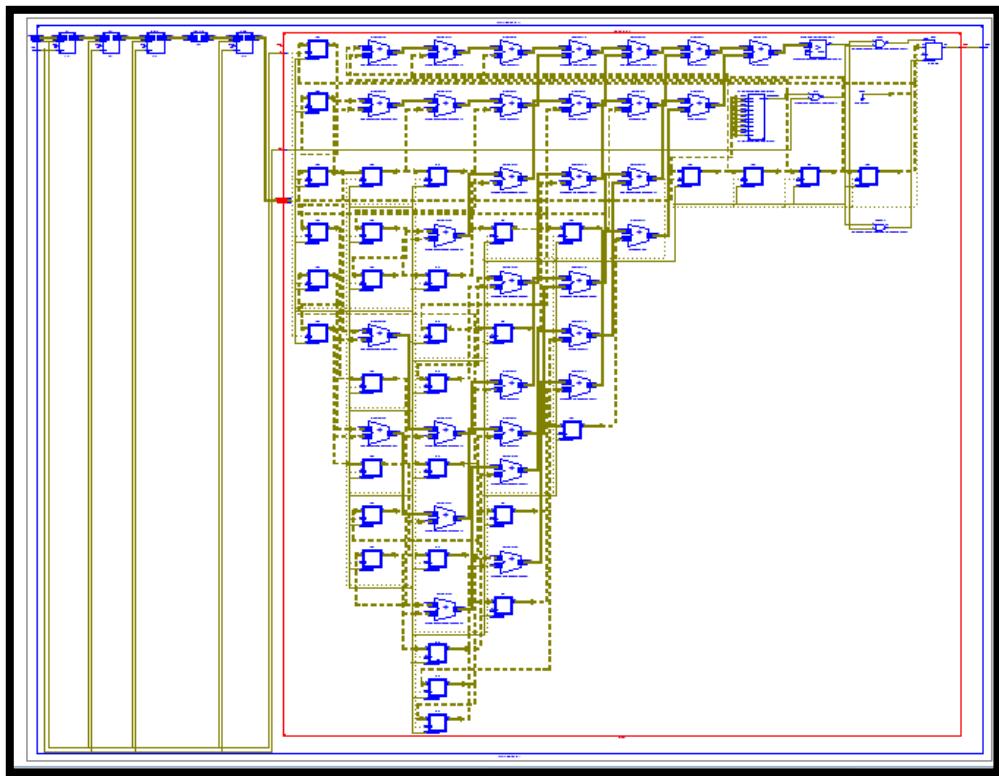
Gambar 5.14 Perbandingan masukan dan keluaran implementasi blok pengkuadrat



Gambar 5.15 Tampilan blok integrasi sistem deteksi QRS



Gambar 5.16 Perbandingan masukan dan keluaran implementasi blok integrator



**Gambar 5.17 Tampilan blok decision sistem deteksi QRS**

Laporan penggunaan *resource* pada FPGA secara lengkap sebagai hasil sintesa menggunakan Xilinx 12.i terdapat pada tabel di bawah ini.

**Tabel 5.1 Penggunaan *Resource* Komponen pada FPGA.**

Komponen	Jumlah
• Adders/Subtractors	68
10-bit adder	31
10-bit subtractor	6
13-bit adder	31
• Multiplier	1
10 x 10 bit Multiplier	1
• Comparators	1
13-bit comparator greater	1
• Register	1097
Flip-flop	1097
• Shift Register	36
15-bit shiftbregister	20
6-bit multiplier	16

Dari sintesis juga didapatkan keterangan hasil implementasi rangkaian sebagai berikut.

**Tabel 5.2 Hasil Ringkasan Implementasi pada FPGA menggunakan Xilinx**

Device Utilization Summary					[1]
Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Flip Flops	609	21,504	2%		
Number of 4 input LUTs	647	21,504	3%		
Number of occupied Slices	657	10,752	6%		
Number of Slices containing only related logic	657	657	100%		
Number of Slices containing unrelated logic	0	657	0%		
Total Number of 4 input LUTs	680	21,504	3%		
Number used as logic	611				
Number used as a route-thru	33				
Number used as Shift registers	36				
Number of bonded IOBs	12	448	2%		
IOB Flip Flops	1				
Number of BUFG/BUFGCTRLs	2	32	6%		
Number used as BUFGs	2				
Number of DSP48s	1	48	2%		
Average Fanout of Non-Clock Nets	2.35				

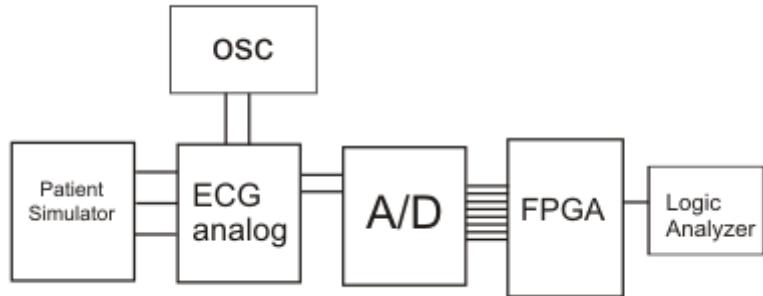
**Tabel 5.3 Hasil Sintesa Implementasi pada FPGA menggunakan Xilinx**

Parameter	Nilai
Periode minimum	9.339 ns
Frekuensi maksimum	107.078 MHz
Delay kedatangan minimum input sebelum <i>clock</i>	4.221 ns
Delay maksimum setelah <i>clock</i>	9.339 ns
Total memory yang digunakan	97204 kilobytes

### 5.3. Pengujian Perangkat Keras

Setelah program berhasil ditanam pada FPGA dengan menggunakan *software* Xilinx, maka tahap selanjutnya adalah menampilkan sinyal keluaran dengan menggunakan *Logic Analyzer*. *Logic Analyzer* akan menampilkan sinyal digital dari

perangkat FPGA dalam bentuk sinyal diskrit yaitu ‘0’ dan ‘1’. Skenario pengimplementasian program pada FPGA terlihat pada gambar berikut.



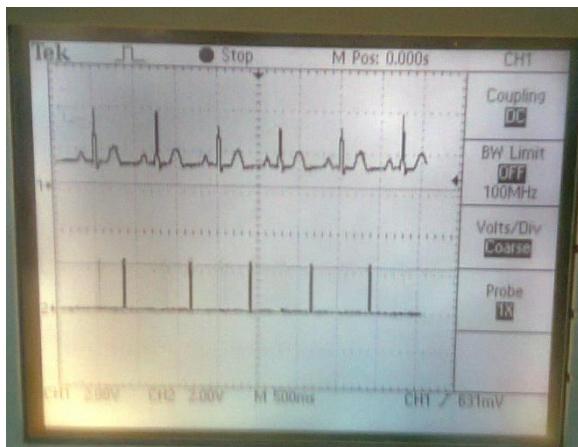
**Gambar 5.18 Skenario implementasi sistem deteksi kompleks QRS**



**Gambar 5.19 Proses pengujian implementasi sistem deteksi QRS**

Dalam pengimplementasian pada FPGA ini, *data rate* yang dibutuhkan sebesar 200 Hz karena frekuensi sampling yang digunakan adalah 200 sample per detik. FPGA ini memiliki master clock dengan frekuensi 100 MHz sehingga untuk mendapatkan *data rate* 200 Hz harus diperkecil frekuensinya atau dengan kata lain *clock cycle* harus diperlambat. Oleh sebab itu diperlukan program yang digunakan untuk memperlambat *clock cycle* tersebut. Program tersebut adalah *clock devider*.

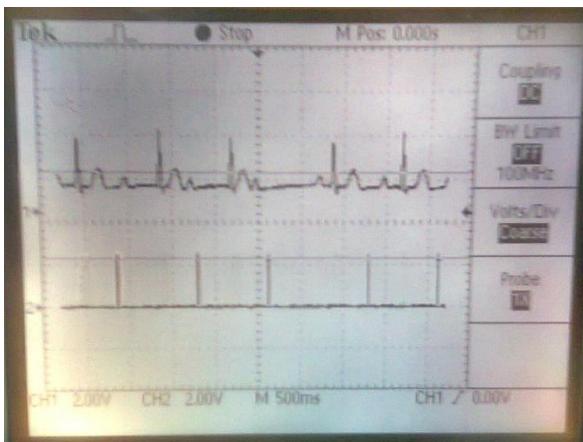
Selanjutnya sistem deteksi QRS yang telah diimplementasikan diuji menggunakan perangkat PS 400 sebagai sumber sinyal ECG yang dimana diujikan sinyal ECG normal, Atrial Fibrillation, Wenkerbach, Right Bundle Branch Block, dan Premature Atrial Contraction. Pengujian dilakukan dengan menggunakan osiloskop dua channel, dimana channel pertama merupakan keluaran perangkat ECG analog dan channel kedua adalah keluaran perangkat FPGA hasil implementasi sistem deteksi QRS. Berikut ini data hasil pengukuran pada perangkat sistem deteksi QRS yang telah dimplementasikan pada FPGA.



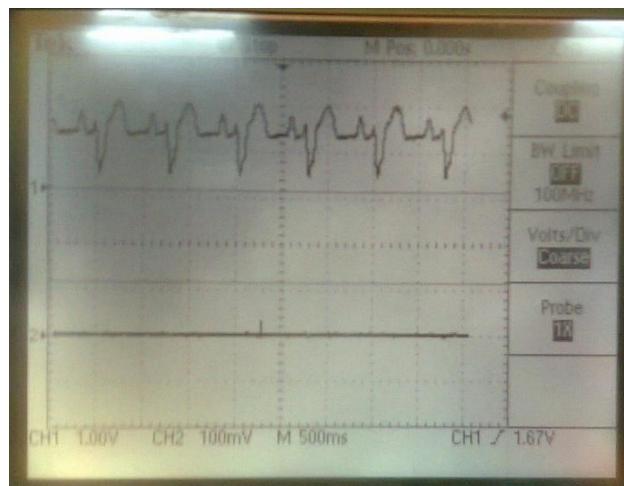
Gambar 5.20 Hasil pengujian implementasi deteksi QRS sinyal ECG normal



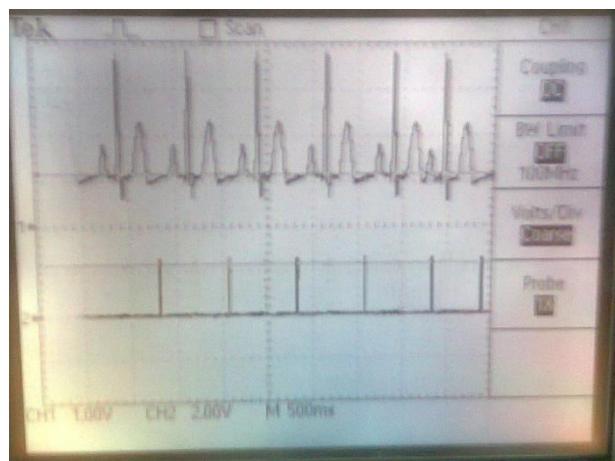
Gambar 5.21 Hasil pengujian implementasi deteksi QRS sinyal ECG Atrial Fibrillation



Gambar 5.22 Hasil pengujian implementasi deteksi QRS sinyal ECG 2° BLK 1



Gambar 5.23 Hasil pengujian implementasi deteksi QRS sinyal RBBB



Gambar 5.24 Hasil pengujian implementasi deteksi QRS sinyal ECG PAC

**Tabel 5.4 Hasil Pengujian dari Perangkat Sistem Deteksi QRS**

Jenis Sinyal ECG	Jumlah QRS	Jumlah terdeteksi
Normal	30	30
Atrial fibrillation.	17	17
2° BLK 1	20	20
Right bundle branch block.	16	0
Premature atrial contraction	27	27

Dari hasil pengujian terhadap perangkat, ternyata yang paling sulit untuk terdeteksi adalah sinyal ECG dengan kondisi RBBB yang memiliki puncak R yang sangat rendah sehingga kemungkinan terpotong saat melalui ADC karena batas ADC yang hanya dapat memproses sinyal dengan nilai daya positif. Sehingga dapat disimpulkan bahwa perangkat sistem deteksi QRS yang dirancang ini memiliki ketergantungan pada kualitas ADC yang digunakan.

## **BAB VI**

### **KESIMPULAN DAN SARAN**

#### **6.1 Kesimpulan**

Kesimpulan yang dapat diambil berdasarkan hasil implementasi yang didapat adalah :

1. Implementasi sistem pada bahasa VHDL berbeda dengan implementasi pada bahasa C karena sifat pemrosesannya yang sesuai flow data sehingga diperlukan arsitektur khusus yang berorientasikan rangkaian logika.
2. Perancangan sistem pada perangkat FPGA memudahkan pengembangan lebih lanjut karena kompatibilitas sistem FPGA yang mudah untuk diintegrasikan dengan perangkat lain.
3. Sistem deteksi QRS yang diimplementasikan pada tugas akhir ini memiliki *delay* sebesar 49 clock (245 ms) dihitung dari mulai kemunculan QRS. Sehingga alat dapat dikatakan cukup baik karena QRS terdeteksi sebelum kedatangan QRS berikutnya yang rata-rata jarak R-R adalah 660 ms (80 detak per menit)
4. Untuk memproses sinyal analog pada FPGA dibutuhkan perangkat ADC tambahan karena keterbatasan FPGA yang hanya dapat mengolah sinyal digital.
5. Hasil simulasi menunjukkan bahwa sistem memiliki tingkat ketepatan mendeteksi sebesar 92%.

#### **6.2 Saran**

Beberapa saran yang dapat dilakukan untuk pengembangan sistem deteksi QRS ini adalah :

1. Untuk pengembangan selanjutnya, hasil keluaran dari sistem ini dapat diolah lebih lanjut terutama untuk sistem control digital yang berdasar pada detak jantung.
2. Untuk pengembangan selanjutnya dapat dilakukan dengan melakukan pengukuran tingkat presisi dari sistem deteksi QRS.

3. Perancangan sistem yang serupa diharapkan dapat diimplementasikan pada programmable logic device yang lebih sederhana seperti CPLD, PAL, dan GAL.
4. Untuk pengembangan selanjutnya dapat dicoba implementasi dengan algoritma yang berbeda pada device yang sama atau algoritma yang sama dengan device yang sama.

## DAFTAR PUSTAKA

- [1] Pan J., and Tompkins W. J. 1985. *A Real-Time QRS Detection Algorithm*, IEEE Trans. Biomed. Eng.
- [2] Pavlatos C., Dimopoulos., dan Manis G., 2003. *Hardware Implementation of Pan & Tompkins QRS Detection Algorithm*. National Technical University of Athens.
- [3] Tompkins W. J., 1993. *Biomedical Digital Signal Processing*. Prentice Hall.
- [4] Ludeman L.C., 1987. *Fundamental of Digital Signal Processing*. John Wiley and Sons
- [5] Eugene N. B., 2001. *Biomedical Signal Processing and Signal Modeling*. John Wiley and Sons
- [6] Brown S., and Vranesic Z., 2008. *Fundamentals of Digital Logic with VHDL Design*. McGraw-Hill.
- [7] Meyer-Bease U., 2001 *Digital Signal Processing with Field Programmable Gate Array*. Springle-Verlag.
- [8] Hadiyoso S., 2008. *Sistem Monitoring Photoplethysmograph Digital dengan Wireless LAN (802.11b) sebagai Pengirim Data*. IT Telkom: Tidak Diterbitkan
- [9] Bragge T., Tarvainen M.P., Karjalainen P. A. 2004. High-Resolution QRS Detection Algorith for Sparsely Sampled ECG Recordings. University of Kuopio.
- [10] PTB diagnostic ECG Databse: patient001/s0010,  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [11] MIT BIH Arrhythmia Database: 100, <http://www.physionet.org/cgi-bin/atm/ATM>
- [12] St. Petersburg INCART 12-lead Arrhythmia Database: I74  
<http://www.physionet.org/cgi-bin/atm/ATM>

- [13] St. Petersburg INCART 12-lead Arrhythmia Database: I02  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [14] St. Petersburg INCART 12-lead Arrhythmia Database: I01  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [15] MIT BIH Arrhythmia Database: 232, <http://www.physionet.org/cgi-bin/atm/ATM>
- [16] St. Petersburg INCART 12-lead Arrhythmia Database: I25  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [17] Intracardiac Atrial Fibrillation Database : iaf2tva,  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [18] CU Ventricular Tachyarrhythmia Database: cu19,  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [19] BIDMC Congestive Heart Failure Database: chf04  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [20] CU Ventricular Tachyarrhythmia Database: cu20,  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [21] CU Ventricular Tachyarrhythmia Database: cu23,  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [22] CU Ventricular Tachyarrhythmia Database: cu25,  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [23] CU Ventricular Tachyarrhythmia Database: cu26,  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [24] CU Ventricular Tachyarrhythmia Database: cu29,  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [25] CU Ventricular Tachyarrhythmia Database: cu16,  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [26] BIDMC Congestive Heart Failure Database: chf01  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [27] BIDMC Congestive Heart Failure Database: chf02  
<http://www.physionet.org/cgi-bin/atm/ATM>

- [28] BIDMC Congestive Heart Failure Database: chf07  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [29] Intracardiac Atrial Fibrillation Database : iaf8tva,  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [30] European ST-T Database: e0104, <http://www.physionet.org/cgi-bin/atm/ATM>
- [31] European ST-T Database: e0105, <http://www.physionet.org/cgi-bin/atm/ATM>
- [32] European ST-T Database: e0106, <http://www.physionet.org/cgi-bin/atm/ATM>
- [33] MIT-BIH Atrial Fibrillation Database : 08405 <http://www.physionet.org/cgi-bin/atm/ATM>
- [34] St. Petersburg INCART 12-lead Arrhythmia Database: I60  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [35] St. Petersburg INCART 12-lead Arrhythmia Database: I64  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [36] St. Petersburg INCART 12-lead Arrhythmia Database: I65  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [37] St. Petersburg INCART 12-lead Arrhythmia Database: I67  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [38] St. Petersburg INCART 12-lead Arrhythmia Database: I74  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [39] St. Petersburg INCART 12-lead Arrhythmia Database: I12  
<http://www.physionet.org/cgi-bin/atm/ATM>
- [40] non-personal article , 12 December 2010, *Electrocardiography*, International Wikipedia, <http://en.wikipedia.org/wiki/Electrocardiography>, access on 14 December 2010
- [41] Sujatmiko, W. 2011. *Perancangan dan Implementasi MIMO Encoder Decoder STBC Alamouti 2x2 Berbasis FPGA*. IT Telkom: Tidak diterbitkan.

## LAMPIRAN A

### Listing Program

---

```

Author      : Ahmad Zaky Ramdani
NIM        : 111070131
Email       : ahmadzakyramdani@gmail.com
Program Name: Pan-Tompkins algoritm QRS detector
Created     : June 2011

```

---

#### Low Pass Filter

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity lpf is
port (
    reset      : in std_logic;
    enable      : in std_logic;
    clk         : in std_logic;
    inlpf      : in std_logic_vector ( 9 downto 0 ) ;
    outlpf     : out std_logic_vector (9 downto 0));
end entity;

architecture lpfcon of lpf is

function mul2 (a : std_logic_vector(9 downto 0)) return std_logic_vector is
variable result : std_logic_vector (9 downto 0);
begin
    result := a(8 downto 0) & '0';
    return result;
end function mul2;

function conv (a : std_logic_vector (9 downto 0)) return std_logic_vector is
variable result, b : std_logic_vector (9 downto 0);
begin
    if a(9) = '0' then
        b := not a;
        result := b + "0000000001";
    elsif a(9) = '1' then
        b := a - "0000000001";
        result := not b;
    end if;
    return result;
end function conv;

signal res1 : std_logic_vector ( 9 downto 0 ) ;
signal res2 : std_logic_vector ( 9 downto 0 ) ;
signal res3 : std_logic_vector ( 9 downto 0 ) ;
signal res4 : std_logic_vector ( 9 downto 0 ) ;

```

```

signal res5 : std_logic_vector ( 9 downto 0 ) ;
signal y0, y1, y2 : std_logic_vector ( 9 downto 0 ) ;
signal x0, x1, x2 : std_logic_vector ( 9 downto 0 ) ;
signal x3, x4, x5 : std_logic_vector ( 9 downto 0 ) ;
signal x6, x7, x8 : std_logic_vector ( 9 downto 0 ) ;
signal x9, x10, x11 : std_logic_vector ( 9 downto 0 ) ;
signal x12 : std_logic_vector ( 9 downto 0 ) ;

begin

res1 <= mul2(y1);
res2 <= conv(y2);
res3 <= x0;
res4 <= conv(mul2(x6));
res5 <= x12;

y0 <= res1 + res2 + res3 + res4 + res5;

outlpf <= y0;

regy : process (enable, reset, clk)
begin
    if (reset = '1') then
        y1 <= (others => '0');
        y2 <= (others => '0');
    elsif (enable = '0') then
        y1 <= (others => '0');
        y2 <= (others => '0');
    elsif (clk'event and clk = '1') then
        y2 <= y1 ;
        y1 <= y0 ;
    end if ;
end process;

regx : process (clk)
begin
    if (reset = '1') then
        x0 <= (others => '0');
        x1 <= (others => '0');
        x2 <= (others => '0');
        x3 <= (others => '0');
        x4 <= (others => '0');
        x5 <= (others => '0');
        x6 <= (others => '0');
        x7 <= (others => '0');
        x8 <= (others => '0');
        x9 <= (others => '0');
        x10 <= (others => '0');
        x11 <= (others => '0');
        x12 <= (others => '0');
    elsif (enable = '0') then
        x0 <= (others => '0');
        x1 <= (others => '0');
        x2 <= (others => '0');
        x3 <= (others => '0');
        x4 <= (others => '0');
        x5 <= (others => '0');

```

```

        x6  <= (others => '0');
        x7  <= (others => '0');
        x8  <= (others => '0');
        x9  <= (others => '0');
        x10 <= (others => '0');
        x11 <= (others => '0');
        x12 <= (others => '0');
    elsif clk'event and (clk = '1') then
        x12 <= x11;
        x11 <= x10;
        x10 <= x9;
        x9  <= x8;
        x8  <= x7;
        x7  <= x6;
        x6  <= x5;
        x5  <= x4;
        x4  <= x3;
        x3  <= x2;
        x2  <= x1;
        x1  <= x0;
        x0  <= inlpf;
    end if;
end process;

end architecture;
-----High Pass Filter-----
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity hpf is
port (
    reset      : in std_logic;
    enable     : in std_logic;
    clk        : in std_logic;
    inhpf     : in std_logic_vector ( 9 downto 0 );
    outhpf    : out std_logic_vector ( 9 downto 0 ));
end entity;

architecture hpfcon of hpf is

function div32 (a : std_logic_vector(9 downto 0)) return std_logic_vector is
variable result : std_logic_vector ( 9 downto 0 );
begin
    result := a(9) & a(9) & a(9) & a(9) & a(9) & a(9 downto 5);
    return result;
end function div32;

function conv (a : std_logic_vector ( 9 downto 0 )) return std_logic_vector is
variable result, b : std_logic_vector ( 9 downto 0 );
begin
    if a(9) = '0' then
        b := not a;
        result := b + "0000000001";
    elsif a(9) = '1' then

```

```

        b := a - "0000000001";
        result := not b;
    end if;
    return result;
end function conv;

signal res1 : std_logic_vector ( 9 downto 0 ) ;
signal res2 : std_logic_vector ( 9 downto 0 ) ;
signal res3 : std_logic_vector ( 9 downto 0 ) ;
signal res4 : std_logic_vector ( 9 downto 0 ) ;
signal res5 : std_logic_vector ( 9 downto 0 ) ;
signal res6 : std_logic_vector ( 9 downto 0 ) ;
signal res7 : std_logic_vector ( 9 downto 0 ) ;
signal res8 : std_logic_vector ( 9 downto 0 ) ;
signal y1, y0 : std_logic_vector ( 9 downto 0 ) ;
signal x0, x1, x2, x3 : std_logic_vector ( 9 downto 0 );
signal x4, x5, x6, x7 : std_logic_vector ( 9 downto 0 );
signal x8, x9, x10, x11 : std_logic_vector ( 9 downto 0 );
signal x12, x13, x14, x15 : std_logic_vector ( 9 downto 0 );
signal x16, x17, x18, x19 : std_logic_vector ( 9 downto 0 );
signal x20, x21, x22, x23 : std_logic_vector ( 9 downto 0 );
signal x24, x25, x26, x27 : std_logic_vector ( 9 downto 0 );
signal x28, x29, x30, x31 : std_logic_vector ( 9 downto 0 );
signal x32 : std_logic_vector ( 9 downto 0 );

begin

res1 <= x16;
res2 <= conv(x17);
res3 <= y1;
res4 <= x0 + x32;
res5 <= div32(res4);
res6 <= conv(res5);
res7 <= res1 + res2;
res8 <= res7 + res3;
outhpf <= res8 + res6;

regy : process (reset, enable, clk)
begin
    if (reset = '1') then
        y0 <= (others => '0');
        y1 <= (others => '0');
    elsif (enable = '0') then
        y0 <= (others => '0');
        y1 <= (others => '0');
    elsif (clk'event and clk = '1') then
        y1 <= y0 ;
    end if ;
end process;

regx : process (clk)
begin
    if (reset = '1') then
        x0 <= (others => '0');
        x1 <= (others => '0');
        x2 <= (others => '0');
        x3 <= (others => '0');

```

```

x4  <= (others => '0');
x5  <= (others => '0');
x6  <= (others => '0');
x7  <= (others => '0');
x8  <= (others => '0');
x9  <= (others => '0');
x10 <= (others => '0');
x11 <= (others => '0');
x12 <= (others => '0');
x13 <= (others => '0');
x14 <= (others => '0');
x15 <= (others => '0');
x16 <= (others => '0');
x17 <= (others => '0');
x18 <= (others => '0');
x19 <= (others => '0');
x20 <= (others => '0');
x21 <= (others => '0');
x22 <= (others => '0');
x23 <= (others => '0');
x24 <= (others => '0');
x25 <= (others => '0');
x26 <= (others => '0');
x27 <= (others => '0');
x28 <= (others => '0');
x29 <= (others => '0');
x30 <= (others => '0');
x31 <= (others => '0');
x32 <= (others => '0');

elsif (enable = '0') then
  x0  <= (others => '0');
  x1  <= (others => '0');
  x2  <= (others => '0');
  x3  <= (others => '0');
  x4  <= (others => '0');
  x5  <= (others => '0');
  x6  <= (others => '0');
  x7  <= (others => '0');
  x8  <= (others => '0');
  x9  <= (others => '0');
  x10 <= (others => '0');
  x11 <= (others => '0');
  x12 <= (others => '0');
  x13 <= (others => '0');
  x14 <= (others => '0');
  x15 <= (others => '0');
  x16 <= (others => '0');
  x17 <= (others => '0');
  x18 <= (others => '0');
  x19 <= (others => '0');
  x20 <= (others => '0');
  x21 <= (others => '0');
  x22 <= (others => '0');
  x23 <= (others => '0');
  x24 <= (others => '0');
  x25 <= (others => '0');
  x26 <= (others => '0');

```

```

        x27 <= (others => '0');
        x28 <= (others => '0');
        x29 <= (others => '0');
        x30 <= (others => '0');
        x31 <= (others => '0');
        x32 <= (others => '0');
    elsif clk'event and (clk = '1') then
        x32 <= x31;
        x31 <= x30;
        x30 <= x29;
        x29 <= x28;
        x28 <= x27;
        x27 <= x26;
        x26 <= x25;
        x25 <= x24;
        x24 <= x23;
        x23 <= x22;
        x22 <= x21;
        x21 <= x20;
        x20 <= x19;
        x19 <= x18;
        x18 <= x17;
        x17 <= x16;
        x16 <= x15;
        x15 <= x14;
        x14 <= x13;
        x13 <= x12;
        x12 <= x11;
        x11 <= x10;
        x10 <= x9;
        x9 <= x8;
        x8 <= x7;
        x7 <= x6;
        x6 <= x5;
        x5 <= x4;
        x4 <= x3;
        x3 <= x2;
        x2 <= x1;
        x1 <= x0;
        x0 <= inhpf;
    end if;
end process;

end architecture;
-----Derivative-----
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity deriv is
    port (
        reset      : in std_logic;
        enable     : in std_logic;
        clk        : in std_logic;
        indef     : in std_logic_vector (9 downto 0) ;
        outdef    : out std_logic_vector (9 downto 0));

```

```

end entity;

architecture derivcon of deriv is

function mul2 (a : std_logic_vector(9 downto 0)) return std_logic_vector is
    variable result : std_logic_vector (9 downto 0);
begin
    result := a(9) & a(7 downto 0) & '0';
    return result;
end function mul2;

function conv (a : std_logic_vector (9 downto 0)) return std_logic_vector is
variable result, b : std_logic_vector (9 downto 0);
begin
    if a(9) = '0' then
        b := not a;
        result := b + "0000000001";
    elsif a(9) = '1' then
        b := a - "0000000001";
        result := not b;
    end if;
    return result;
end function conv;

function div8 (a : std_logic_vector(9 downto 0)) return std_logic_vector is
    variable result : std_logic_vector (9 downto 0);
begin
    result := a(9) & a(9) & a(9) & a(9 downto 3);
    return result;
end function div8;

signal res1  : std_logic_vector (9 downto 0) ;
signal res2  : std_logic_vector (9 downto 0) ;
signal res3  : std_logic_vector (9 downto 0) ;
signal res4  : std_logic_vector (9 downto 0) ;
signal y0    : std_logic_vector (9 downto 0) ;
signal x0    : std_logic_vector (9 downto 0) ;
signal x1    : std_logic_vector (9 downto 0) ;
signal x2    : std_logic_vector (9 downto 0) ;
signal x3    : std_logic_vector (9 downto 0) ;
signal x4    : std_logic_vector (9 downto 0) ;

begin

res1  <= mul2(x0);
res2  <= x1;
res3  <= conv(x3);
res4  <= conv(mul2(x4));
y0    <= res1 + res2 + res3 + res4 ;
outdef <= div8(y0);

regx : process (reset, enable, clk)
begin
    if (reset = '1') then
        x0 <= (others => '0');
        x1 <= (others => '0');
        x2 <= (others => '0');

```

```

        x3 <= (others => '0');
        x4 <= (others => '0');
    elsif (enable = '0') then
        x0 <= (others => '0');
        x1 <= (others => '0');
        x2 <= (others => '0');
        x3 <= (others => '0');
        x4 <= (others => '0');
    elsif clk'event and (clk = '1') then
        x4 <= x3;
        x3 <= x2;
        x2 <= x1;
        x1 <= x0;
        x0 <= indef;
    end if;
end process;

end architecture;

```

-----Squaring-----

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity square is
    port (
        insq      : in std_logic_vector (9 downto 0) ;
        outsq     : out std_logic_vector (9 downto 0)) ;
end entity;

architecture sqcon of square is

function square (a : std_logic_vector (9 downto 0)) return std_logic_vector
is
    variable d : std_logic_vector (19 downto 0);
    variable result : std_logic_vector (9 downto 0);
begin
    d := a * a;
    result := d (14 downto 5);
    return result;
end function square;

begin
    outsq <= square (insq) ;
end architecture;

```

-----Moving Window Integration-----

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity integ is

```

```

port (
    reset      : in std_logic;
    enable     : in std_logic;
    clk        : in std_logic;
    ininteg   : in std_logic_vector ( 9 downto 0) ;
    outinteg  : out std_logic_vector (9 downto 0));
end entity integ;

architecture integcon of integ is
function div32 (a : std_logic_vector(9 downto 0)) return std_logic_vector is
    variable result : std_logic_vector (9 downto 0);
    begin
        result := "000" & a(9 downto 3);
        return result;
    end function div32;

signal y0, y1, y2, y3      : std_logic_vector ( 9 downto 0);
signal y4, y5, y6, y7      : std_logic_vector ( 9 downto 0);
signal y8, y9, y10, y11    : std_logic_vector ( 9 downto 0);
signal y12, y13, y14, y15  : std_logic_vector ( 9 downto 0);
signal x0, x1, x2, x3      : std_logic_vector ( 9 downto 0);
signal x4, x5, x6, x7      : std_logic_vector ( 9 downto 0);
signal x8, x9, x10, x11   : std_logic_vector ( 9 downto 0);
signal x12, x13, x14, x15  : std_logic_vector ( 9 downto 0);
signal x16, x17, x18, x19  : std_logic_vector ( 9 downto 0);
signal x20, x21, x22, x23  : std_logic_vector ( 9 downto 0);
signal x24, x25, x26, x27  : std_logic_vector ( 9 downto 0);
signal x28, x29, x30, x31  : std_logic_vector ( 9 downto 0);
signal x32 : std_logic_vector ( 9 downto 0);

begin

regx : process (reset, enable, clk)
begin
    if (reset = '1') then
        x0  <= (others => '0');
        x1  <= (others => '0');
        x2  <= (others => '0');
        x3  <= (others => '0');
        x4  <= (others => '0');
        x5  <= (others => '0');
        x6  <= (others => '0');
        x7  <= (others => '0');
        x8  <= (others => '0');
        x9  <= (others => '0');
        x10 <= (others => '0');
        x11 <= (others => '0');
        x12 <= (others => '0');
        x13 <= (others => '0');
        x14 <= (others => '0');
        x15 <= (others => '0');
        x16 <= (others => '0');
        x17 <= (others => '0');
        x18 <= (others => '0');
        x19 <= (others => '0');
        x20 <= (others => '0');
        x21 <= (others => '0');
    end if;
end process;

```

```

x22 <= (others => '0');
x23 <= (others => '0');
x24 <= (others => '0');
x25 <= (others => '0');
x26 <= (others => '0');
x27 <= (others => '0');
x28 <= (others => '0');
x29 <= (others => '0');
x30 <= (others => '0');
x31 <= (others => '0');
x32 <= (others => '0');

elsif (enable = '0') then
    x0 <= (others => '0');
    x1 <= (others => '0');
    x2 <= (others => '0');
    x3 <= (others => '0');
    x4 <= (others => '0');
    x5 <= (others => '0');
    x6 <= (others => '0');
    x7 <= (others => '0');
    x8 <= (others => '0');
    x9 <= (others => '0');
    x10 <= (others => '0');
    x11 <= (others => '0');
    x12 <= (others => '0');
    x13 <= (others => '0');
    x14 <= (others => '0');
    x15 <= (others => '0');
    x16 <= (others => '0');
    x17 <= (others => '0');
    x18 <= (others => '0');
    x19 <= (others => '0');
    x20 <= (others => '0');
    x21 <= (others => '0');
    x22 <= (others => '0');
    x23 <= (others => '0');
    x24 <= (others => '0');
    x25 <= (others => '0');
    x26 <= (others => '0');
    x27 <= (others => '0');
    x28 <= (others => '0');
    x29 <= (others => '0');
    x30 <= (others => '0');
    x31 <= (others => '0');
    x32 <= (others => '0');

elsif clk'event and (clk = '1') then
    x32 <= x31;
    x31 <= x30;
    x30 <= x29;
    x29 <= x28;
    x28 <= x27;
    x27 <= x26;
    x26 <= x25;
    x25 <= x24;
    x24 <= x23;
    x23 <= x22;
    x22 <= x21;

```

```

        x21 <= x20;
        x20 <= x19;
        x19 <= x18;
        x18 <= x17;
        x17 <= x16;
        x16 <= x15;
        x15 <= x14;
        x14 <= x13;
        x13 <= x12;
        x12 <= x11;
        x11 <= x10;
        x10 <= x9;
        x9  <= x8;
        x8  <= x7;
        x7  <= x6;
        x6  <= x5;
        x5  <= x4;
        x4  <= x3;
        x3  <= x2;
        x2  <= x1;
        x1  <= x0;
        x0  <= ininteg;
    end if;
end process;

y0 <= (x0 + x1) + (x2 + x3);
y1 <= (x4 + x5) + (x6 + x7);
y2 <= (x8 + x9) + (x10 + x11);
y3 <= (x12 + x13) + (x14 + x15);
y4 <= (x16 + x17) + (x18 + x19);
y5 <= (x20 + x21) + (x22 + x23);
y6 <= (x24 + x25) + (x26 + x27);
y7 <= (x28 + x29) + (x30 + x31);

y8 <= y0 + y1;
y9 <= y2 + y3;
y10 <= y4 + y5;
y11 <= y6 + y7;
y12 <= div32(y8);
y13 <= div32(y9);
y14 <= div32 (y10);
y15 <= div32 (y11);

outinteg <= (y12 + y13) ;

end architecture;

-----decision-----
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity decision is
    generic ( threshold : std_logic_vector(12 downto 0) :=
"0000000100000" ) ;

```

```

port (
    reset      : in std_logic;
    enable     : in std_logic;
    clk        : in std_logic;
    indec      : in std_logic_vector (9 downto 0) ;
    outdec     : out std_logic );
end entity;

architecture dec of decision is

signal sum1      : std_logic_vector ( 12 downto 0);
signal sum2      : std_logic_vector ( 12 downto 0);
signal sum3      : std_logic_vector ( 12 downto 0);
signal sum4      : std_logic_vector ( 12 downto 0);
signal sum5      : std_logic_vector ( 12 downto 0);
signal sum6      : std_logic_vector ( 12 downto 0);
signal sum7      : std_logic_vector ( 12 downto 0);
signal sum8      : std_logic_vector ( 12 downto 0);
signal sumx      : std_logic_vector ( 12 downto 0);
signal sumy      : std_logic_vector ( 12 downto 0);
signal sum       : std_logic_vector ( 12 downto 0);
signal x0, x1, x2, x3      : std_logic_vector ( 12 downto 0);
signal x4, x5, x6, x7      : std_logic_vector ( 12 downto 0);
signal x8, x9, x10, x11     : std_logic_vector ( 12 downto 0);
signal x12, x13, x14, x15    : std_logic_vector ( 12 downto 0);
signal x16, x17, x18, x19    : std_logic_vector ( 12 downto 0);
signal x20, x21, x22, x23    : std_logic_vector ( 12 downto 0);
signal x24, x25, x26, x27    : std_logic_vector ( 12 downto 0);
signal x28, x29, x30, x31    : std_logic_vector ( 12 downto 0);

begin
    sum1 <= (x0 + x1 + x2 + x3);
    sum2 <= (x4 + x5 + x6 + x7);
    sum3 <= (x8 + x9 + x10 + x11);
    sum4 <= (x12 + x13 + x14 + x15);
    sum5 <= (x16 + x17 + x18 + x19);
    sum6 <= (x20 + x21 + x22 + x23);
    sum7 <= (x24 + x25 + x26 + x27);
    sum8 <= (x28 + x29 + x30 + x31);
    sumx <= sum1 + sum2 + sum3 + sum4;
    sumy <= sum5 + sum6 + sum7 + sum8;
    sum <= sumx + sumy;

    regx : process (clk)
    begin
        if (reset = '1') or (indec = "0000000000") then
            x0 <= (others => '0');
            x1 <= (others => '0');
            x2 <= (others => '0');
            x3 <= (others => '0');
            x4 <= (others => '0');
            x5 <= (others => '0');
            x6 <= (others => '0');
            x7 <= (others => '0');
            x8 <= (others => '0');
            x9 <= (others => '0');
            x10 <= (others => '0');

```

```

x11 <= (others => '0');
x12 <= (others => '0');
x13 <= (others => '0');
x14 <= (others => '0');
x15 <= (others => '0');
x16 <= (others => '0');
x17 <= (others => '0');
x18 <= (others => '0');
x19 <= (others => '0');
x20 <= (others => '0');
x21 <= (others => '0');
x22 <= (others => '0');
x23 <= (others => '0');
x24 <= (others => '0');
x25 <= (others => '0');
x26 <= (others => '0');
x27 <= (others => '0');
x28 <= (others => '0');
x29 <= (others => '0');
x30 <= (others => '0');
x31 <= (others => '0');
      if      sum > threshold then
        outdec <= '1';
      else
        outdec <= '0';
    end if;
  elsif clk'event and (clk = '1') then
    x31 <= x30;
    x30 <= x29;
    x29 <= x28;
    x28 <= x27;
    x27 <= x26;
    x26 <= x25;
    x25 <= x24;
    x24 <= x23;
    x23 <= x22;
    x22 <= x21;
    x21 <= x20;
    x20 <= x19;
    x19 <= x18;
    x18 <= x17;
    x17 <= x16;
    x16 <= x15;
    x15 <= x14;
    x14 <= x13;
    x13 <= x12;
    x12 <= x11;
    x11 <= x10;
    x10 <= x9;
    x9  <= x8;
    x8  <= x7;
    x7  <= x6;
    x6  <= x5;
    x5  <= x4;
    x4  <= x3;
    x3  <= x2;
    x2  <= x1;

```

```

        x1  <= x0;
        x0  <= "000" & indec;
        outdec <= '0';
    end if;
end process;

end architecture;

-----Program Inti-----
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity QRSdetector is
    port (
        reset      : in std_logic;
        clk        : in std_logic;
        inecg     : in std_logic_vector (7 downto 0);
        outecg    : out std_logic
    );
end entity;

architecture QRSdet_bhv of QRSdetector is

component lpf is
    port (
        reset      : in std_logic;
        enable     : in std_logic;
        clk        : in std_logic;
        inlpf     : in std_logic_vector ( 9 downto 0);
        outlpf    : out std_logic_vector (9 downto 0));
end component lpf;

component hpf is
    port (
        reset      : in std_logic;
        enable     : in std_logic;
        clk        : in std_logic;
        inhpf    : in std_logic_vector ( 9 downto 0);
        outhpf   : out std_logic_vector (9 downto 0));
end component hpf;

component deriv is
    port (
        reset      : in std_logic;
        enable     : in std_logic;
        clk        : in std_logic;
        indef     : in std_logic_vector (9 downto 0);
        outdef    : out std_logic_vector (9 downto 0));
end component deriv;

component square is
    port (
        insq      : in std_logic_vector (9 downto 0);

```

```

        outsq      : out std_logic_vector (9 downto 0));
end component;

component integ is
    port (
        reset      : in std_logic;
        enable     : in std_logic;
        clk        : in std_logic;
        ininteg   : in std_logic_vector ( 9 downto 0);
        outinteg  : out std_logic_vector (9 downto 0));
end component integ;

component decision is
    generic ( threshold : std_logic_vector(12 downto 0) := "00000100000000");
    port (
        reset      : in std_logic;
        enable     : in std_logic;
        clk        : in std_logic;
        indec     : in std_logic_vector (9 downto 0);
        outdec    : out std_logic );
end component decision;

signal enable      : std_logic;
signal insystem   : std_logic_vector ( 9 downto 0);
signal outlpf1    : std_logic_vector ( 9 downto 0);
signal outhpf1    : std_logic_vector ( 9 downto 0);
signal outdef1    : std_logic_vector ( 9 downto 0);
signal outsq1     : std_logic_vector ( 9 downto 0);
signal outint1    : std_logic_vector ( 9 downto 0);

begin

initial : process
begin
    enable <= '0'; wait for 10 ns;
    enable <= '1'; wait;
end process;

input : process (clk)
begin
    insystem <= "00" & inecg;
end process;

LP : lpf
port map (reset  => reset,
           enable  => enable,
           clk     => clk,
           inlpf   => insystem,
           outlpf  => outlpf1
           );

HP : hpf
port map (reset  => reset,
           enable  => enable,
           clk     => clk,
           inhpf   => outlpf1,

```

```
outhpf => outhpf1
);

DF : deriv
port map (reset => reset,
           enable => enable,
           clk     => clk,
           indef   => outhpf1,
           outdef  => outdef1
);

SQ : square
port map (insq  => outdef1,
           outsq => outsq1
);

INT : integ
port map (reset    => reset,
           enable   => enable,
           clk      => clk,
           ininteg  => outsq1,
           outinteg => outint1
);

DEC : decision
port map (reset  => reset,
           enable => enable,
           clk    => clk,
           indec  => outint1,
           outdec => outecg
);

end architecture;
```

**Tabel B.1 Hasil Penghitungan Manual Blok LPF**

x(n)	x(n-1)	x(n-2)	x(n-3)	x(n-4)	x(n-5)	x(n-6)	x(n-7)	x(n-8)	x(n-9)	x(n-10)	x(n-11)	x(n-12)	y(n-2)	y(n-1)	y(n)
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
32	2	0	0	0	0	0	0	0	0	0	0	0	0	2	36
12	32	2	0	0	0	0	0	0	0	0	0	0	2	36	82
5	12	32	2	0	0	0	0	0	0	0	0	0	36	82	133
6	5	12	32	2	0	0	0	0	0	0	0	0	82	133	190
0	6	5	12	32	2	0	0	0	0	0	0	0	133	190	247
13	0	6	5	12	32	2	0	0	0	0	0	0	190	247	313
25	13	0	6	5	12	32	2	0	0	0	0	0	247	313	340
4	25	13	0	6	5	12	32	2	0	0	0	0	313	340	347
0	4	25	13	0	6	5	12	32	2	0	0	0	340	347	344
0	0	4	25	13	0	6	5	12	32	2	0	0	347	344	329
1	0	0	4	25	13	0	6	5	12	32	2	0	344	329	315
17	1	0	0	4	25	13	0	6	5	12	32	2	329	315	294
23	17	1	0	0	4	25	13	0	6	5	12	32	315	294	278
64	23	17	1	0	0	4	25	13	0	6	5	12	294	278	330
10	64	23	17	1	0	0	4	25	13	0	6	5	278	330	397
11	10	64	23	17	1	0	0	4	25	13	0	6	330	397	481

**Tabel B-2 Hasil Penghitungan Manual Blok HPF**

x(n)	x(n-1)	x(n-2)	x(n-3)	x(n-4)	x(n-5)	x(n-6)	x(n-7)	x(n-8)	x(n-9)	x(n-10)	x(n-11)	x(n-12)	x(n-13)	x(n-14)	x(n-15)	x(n-16)	x(n-17)	x(n-18)	x(n-19)	x(n-20)	x(n-21)	x(n-22)	x(n-23)	x(n-24)	x(n-25)	x(n-26)	x(n-27)	x(n-28)	x(n-29)	x(n-30)	x(n-31)	x(n-32)	y(n-1)	y(n)
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.0625			
36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.063	-1.1875			
82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1.188	-3.75			
133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-3.75	-7.90625			
190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-7.906	-13.8438			
247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-13.84	-21.5625			
313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-21.56	-31.3438			
340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-31.34	-41.9688			
347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-41.97	-52.8125			
344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-52.81	-63.5625				
329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-63.56	-73.8438				
315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-73.84	-83.6875				
294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-83.69	-92.875					
278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-92.88	-101.563					
330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	0	-101.6	-111.875					
397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	0	-111.9	-124.281					
11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	0	-124.3	-122.625					
24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	0	0	-122.6	-89.375					
28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	0	-89.38	-44.25						
59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	-44.25	4.90625						
49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	4.9063	60.375					
125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	60.375	113.4688				
235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	113.47	172.125			
175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	0	0	172.13	193.6563		
355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	0	189.56	175.4688		

310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	175.47	150.7813																				
279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	150.78	128.0625																			
220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	128.06	100.1875																		
167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	0	100.19	78.96875																	
99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	78.969	127.875																	
99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	127.88	191.7813																
23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	191.78	-194.875															
11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	36	2	0	0	0	-194.9	-181.094														
-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	82	0	0	0	-181.1	-173.594															
-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	-173.6	-136.969														
-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	-137	-140.656													
0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	-140.7	-56.9375												
17	0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	-56.94	62.3125											
29	17	0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	0	62.313	12.03125									
35	29	17	0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	0	12.031	201.7813								
40	35	29	17	0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	0	201.78	211.2813							
78	40	35	29	17	0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	0	211.28	174.125						
125	78	40	35	29	17	0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	0	149.0625						
197	125	78	40	35	29	17	0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	0	93.09375					
201	197	125	78	40	35	29	17	0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	0	42.5				
156	201	197	125	78	40	35	29	17	0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	0	-20.0625			
67	156	201	197	125	78	40	35	29	17	0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	0	-20.06	-9.75	
31	67	156	201	197	125	78	40	35	29	17	0	-12	-47	-30	11	23	99	99	167	220	279	310	355	355	175	235	125	49	59	28	24	11	397	330	278	294	315	329	344	347	340	313	247	190	133	0	0	0	0	0	-9.75	-86.375

# **Virtex-4™ MB Development Board User's Guide**



**Version 3.0  
December 2005**



---

## Table of Contents

<b>1</b>	<b>OVERVIEW .....</b>	<b>1</b>
<b>2</b>	<b>THE VIRTEX-4 MB SYSTEM BOARD .....</b>	<b>1</b>
<b>3</b>	<b>FUNCTIONAL DESCRIPTION .....</b>	<b>2</b>
3.1	LVDS INTERFACE .....	3
3.1.1	<i>SPI-4.2 Interface.....</i>	4
3.1.2	<i>SPI-4.2 Pin Assignments.....</i>	4
3.1.3	<i>LVDS Connector.....</i>	6
3.2	DDR SDRAM .....	7
3.3	FLASH .....	8
3.4	CLOCK SOURCES .....	9
3.4.1	<i>Programmable LVDS Clock Source .....</i>	11
3.4.2	<i>ICS8442 Programmable LVDS Clock Synthesizer.....</i>	11
3.4.3	<i>ICS8442 Clock Generation.....</i>	13
3.4.4	<i>ICS8442 Programming Modes .....</i>	14
3.4.5	<i>ICS8442 M and N Settings .....</i>	14
3.5	10/100 ETHERNET PHY .....	18
3.6	LCD PANEL.....	20
3.7	USB 2.0 TO RS232 PORT.....	20
3.8	RS232 .....	21
3.9	USER DIP AND PB SWITCHES .....	22
3.10	USER LEDS .....	23
3.11	VBAT JUMPER.....	23
3.12	CONFIGURATION AND DEBUG PORTS.....	23
3.12.1	<i>JTAG Chain .....</i>	23
3.12.2	<i>System ACE Module Connector.....</i>	24
3.12.3	<i>Serial Data Flash.....</i>	26
3.12.4	<i>JTAG Port (PC4) .....</i>	31
3.12.5	<i>Configuration Modes .....</i>	31
3.13	VOLTAGE REGULATORS .....	32
3.14	BANK I/O VOLTAGE .....	33
3.15	P240 EXPANSION MODULE SIGNAL ASSIGNMENTS .....	33
<b>4</b>	<b>REVISIONS .....</b>	<b>36</b>
	<b>APPENDIX A .....</b>	<b>37</b>

---

# Figures

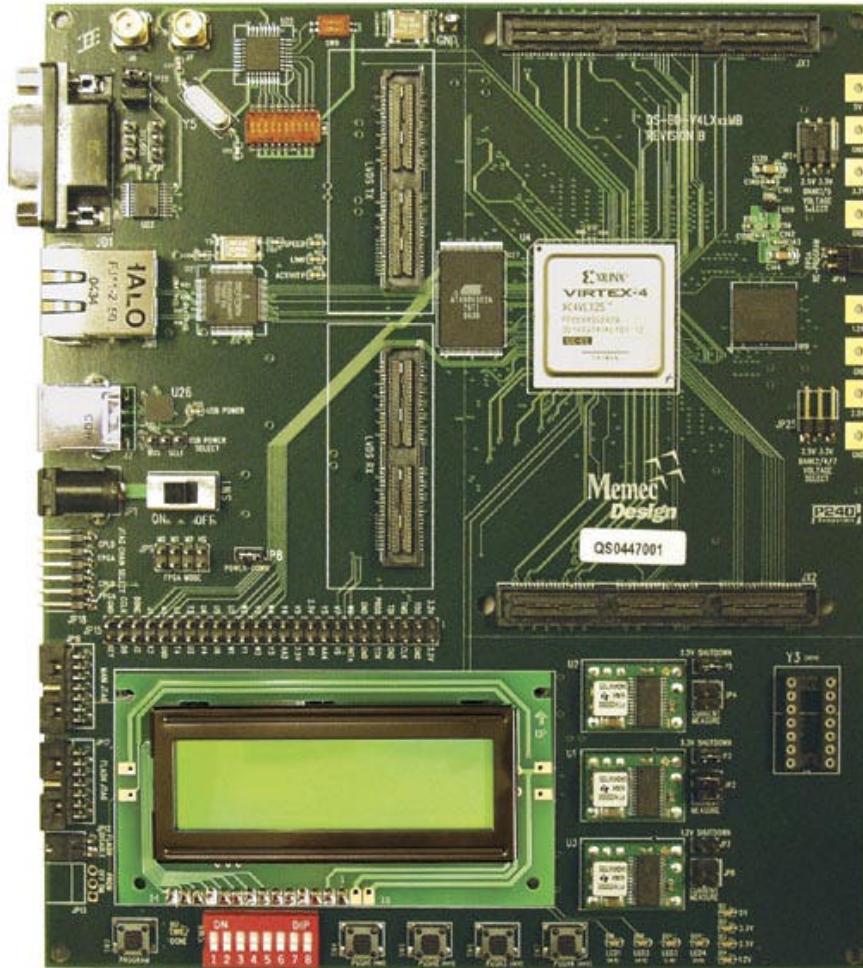
FIGURE 1 - VIRTEX-4 MB DEVELOPMENT PLATFORM BLOCK DIAGRAM .....	3
FIGURE 2- SPI-4.2 INTERFACE .....	4
FIGURE 3 – SAMTEC QSE TYPE CONNECTOR FOR THE SPI-4.2 INTERFACE.....	7
FIGURE 4 – DDR SDRAM INTERFACE.....	7
FIGURE 5 – FLASH INTERFACE .....	8
FIGURE 6 - CLOCK SOURCES ON THE VIRTEX-4 MB BOARD .....	10
FIGURE 7 – ICS8442 CLOCK SYNTHESIZER .....	12
FIGURE 8 – ICS8442 CLOCK SYNTHESIZER INTERFACE TO THE FPGA .....	15
FIGURE 9 – ICS8442 CLOCK SYNTHESIZER M AND N DIP SWITCHES .....	16
FIGURE 10 – M AND N DIP SWITCHES FOR THE SYNTHESIZERS .....	16
FIGURE 11 – 10/100 ETHERNET INTERFACE.....	19
FIGURE 12 – USB 2.0 TO RS232 SERIAL INTERFACE .....	21
FIGURE 13 - RS232 INTERFACE .....	22
FIGURE 14 – VITEX-4 MB DEVELOPMENT BOARD JTAG CHAIN.....	24
FIGURE 15 – SYSTEMACE MODULE.....	25
FIGURE 16 – VIRTEX-4 MB DEVELOPMENT BOARD CONFIGURATION INTERFACE .....	26
FIGURE 17 – VIRTEX-4 MB DEVELOPMENT BOARD JTAG CHAIN.....	28
FIGURE 18 – SERIAL FLASH CONFIGURATION INTERFACE.....	29
FIGURE 19 – PC4 JTAG PORT CONNECTOR.....	31
FIGURE 20 - VOLTAGE REGULATORS .....	32

## 1 Overview

The Memec Virtex-4™ MB Development Kit provides a complete development platform for designing and verifying applications based on the Xilinx Virtex-4 FPGA family. This kit enables designers to implement DSP and embedded processor based applications with extreme flexibility using IP cores and customized modules. The Virtex-4 FPGA along with Xilinx MicroBlaze soft processor core makes it possible to prototype processor based applications, enabling software design teams early access to a hardware platform prior to working with the final product/target board.

The Virtex-4 MB system board utilizes the Xilinx XC4VLX25/LX60/SX35-10FF668C FPGA. The board includes 64MB of DDR SDRAM, 4MB of Flash, 16-bit LVDS Transmit and Receive ports, programmable LVDS clock source, USB-RS232 Bridge, a 10/100 Ethernet PHY, 100 MHz clock source, RS-232 port, and additional user support circuitry to develop a complete system. The board also supports the Memec P240 expansion module standard, allowing application specific expansion modules to be easily added.

## 2 The Virtex-4 MB System Board

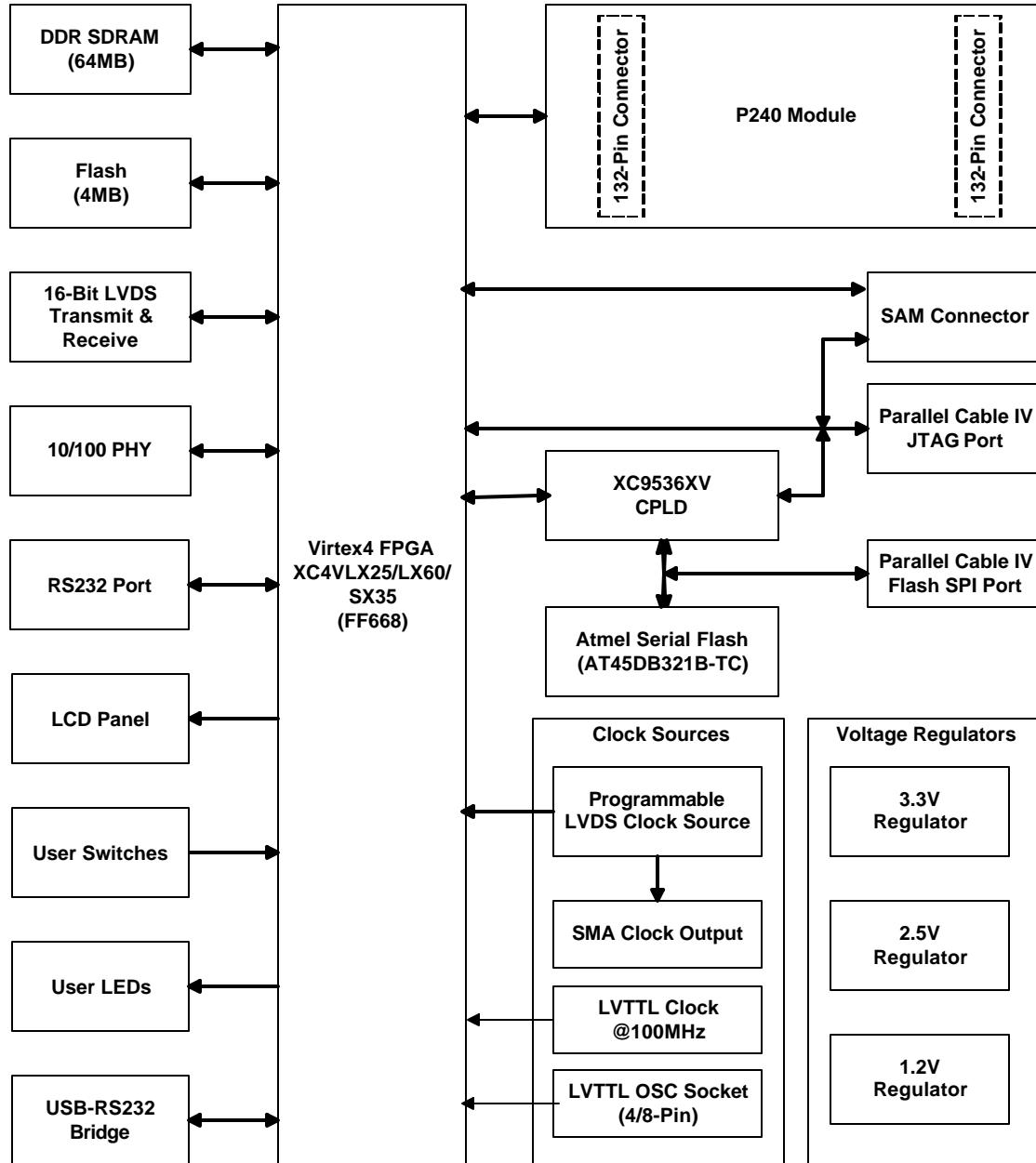


---

### 3 Functional Description

A high-level block diagram of the Virtex-4™ MB development platform is shown below followed by a brief description of each sub-section. A list of features for this board is shown below:

- Xilinx XC4VLX25/LX60/SX35-10FF668 FPGA
- 64MB of DDR SDRAM
- 4MB of Flash
- 16-Bit LVDS Transmit and Receive Interfaces
- 10/100 Ethernet PHY
- Programmable LVDS Clock Source (25-700 MHz)
- User LVDS Clock Outputs via Differential SMA Connectors
- On-board 100MHz LVTTL Oscillator
- On-board LVTTL Oscillator Socket (4/8-Pin Oscillators)
- P240 Connectors
- LCD Panel
- 32Mb Serial Flash for FPGA configuration
- PC4 JTAG Programming/Configuration Port
- SystemACE™ Module Connector
- RS232 Port
- Four User LEDs
- Four User Push Button Switches
- An 8-position DIP Switch
- USB-RS232 Bridge



**Figure 1 - Virtex-4 MB Development Platform Block Diagram**

### 3.1 LVDS Interface

The Virtex-4 MB development board provides high-speed LVDS connectors supporting a SPI-4.2 interface. This interface consists of 36 LVDS signal pairs (72 FPGA signals) and 6 single-ended signals. In addition to the SPI-4.2 interface, the LVDS interface is designed to support XSBI 16-bit LVDS @644Mbps to support a 10GbE interface on the Virtex-4 MB development platform. The following sections provide a brief description of the LVDS interface on this development board.

### 3.1.1 SPI-4.2 Interface

The Virtex-4 MB development board provides a SPI-4.2 via a 16-bit parallel LVDS electrical interface. The following figure shows the SPI-4.2 interface on the board. The transmit and receive interface of the SPI-4.2 are implemented using LVDS signals while the status flow control signals are implemented using single-ended LVTTL signals.

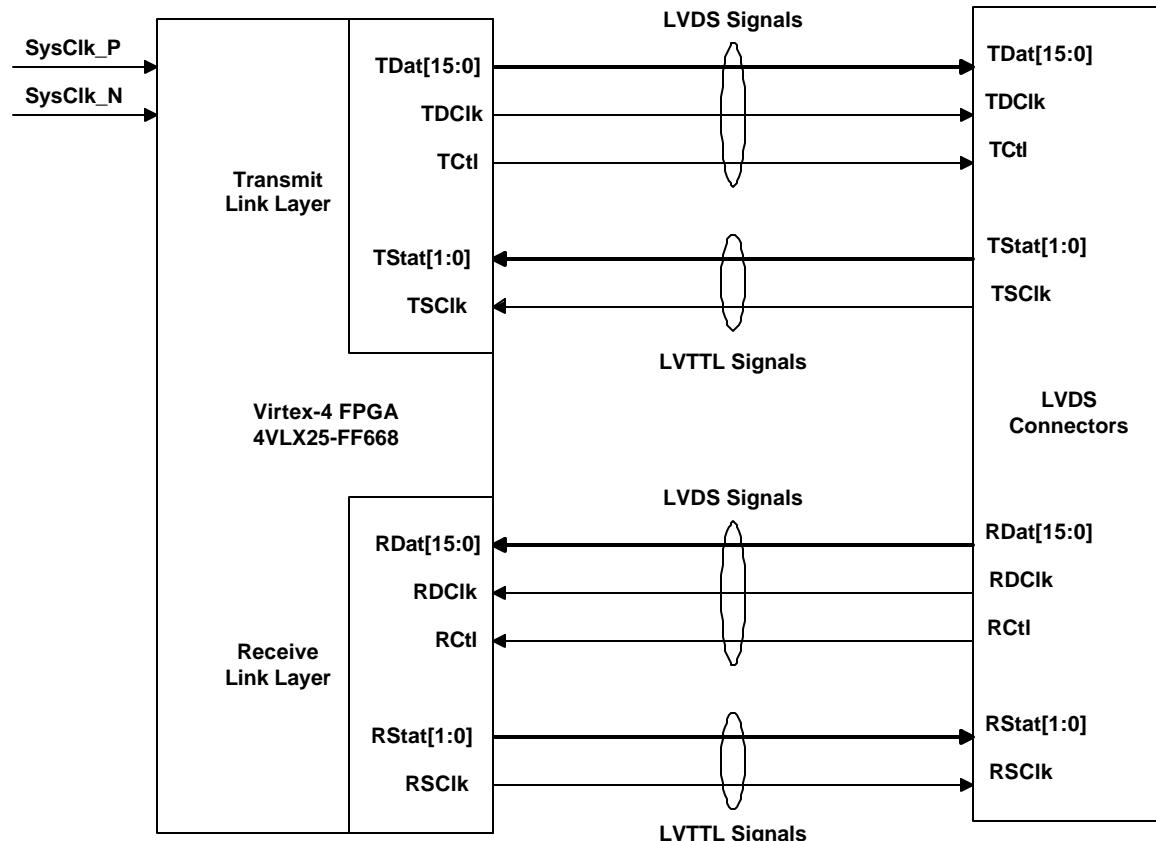


Figure 2- SPI-4.2 Interface

### 3.1.2 SPI-4.2 Pin Assignments

The following table shows the SPI-4.2 pin assignments for the 4VLX25/LX60/SX35 FPGA in the FF668-pin package. These pin assignments must be used in the board design in order to meet the SPI-4.2 interface core requirements.

Table 1- SPI-4.2 Transmit Pin Assignments

Virtex-4 Pin #	LVDS Signal Name	J4 Connector Pin # LVDS TX		LVDS Signal Name	Virtex-4 Pin #
	5.0V	1	2	5.0V	
	5.0V	3	4	5.0V	
	GND	5	6	GND	
	3.3V	7	8	3.3V	
	3.3V	9	10	3.3V	
	GND	11	12	GND	
	2.5V	13	14	2.5V	

	<b>2.5V</b>	15	16	<b>2.5V</b>	
	<b>GND</b>	17	18	<b>GND</b>	
R8	TSCLK	19	20	NC	
	NC	21	22	NC	
	<b>GND</b>	23	24	<b>GND</b>	
T8	TSTAT0	25	26	NC	
T7	TSTAT1	27	28	NC	
	<b>GND</b>	29	30	<b>GND</b>	
A7	TDat_N(15)	31	32	TDat_N(14)	D7
A8	TDat_P(15)	33	34	TDat_P(14)	D8
	<b>GND</b>	35	36	<b>GND</b>	
E10	TDat_N(13)	37	38	TDat_N(12)	A5
F10	TDat_P(13)	39	40	TDat_P(12)	A6
G8	TDat_N(11)	41	42	TDat_N(10)	C7
F8	TDat_P(11)	43	44	TDat_P(10)	B7
	<b>GND</b>	45	46	<b>GND</b>	
D5	TDat_N(9)	47	48	TDat_N(8)	B9
C5	TDat_P(9)	49	50	TDat_P(8)	A9
	<b>GND</b>	51	52	<b>GND</b>	
B3	TDat_N(7)	53	54	TDat_N(6)	D4
A3	TDat_P(7)	55	56	TDat_P(6)	C4
	<b>GND</b>	57	58	<b>GND</b>	
D6	TDat_N(5)	59	60	TDat_N(4)	E5
E7	TDat_P(5)	61	62	TDat_P(4)	E6
	<b>GND</b>	63	64	<b>GND</b>	
G7	TDat_N(3)	65	66	TDat_N(2)	C1
F7	TDat_P(3)	67	68	TDat_P(2)	C2
	<b>GND</b>	69	70	<b>GND</b>	
H7	TDat_N(1)	71	72	TDat_N(0)	E4
H8	TDat_P(1)	73	74	TDat_P(0)	D3
	<b>GND</b>	75	76	<b>GND</b>	
C8	TCtl_N	77	78	TDCLK_N	F9
D9	TCtl_P	79	80	TDCLK_P	E9
	<b>GND</b>	81	82	<b>GND</b>	
	<b>GND</b>	83	84	<b>GND</b>	
	<b>GND</b>	85	86	<b>GND</b>	
	<b>GND</b>	87	88	<b>GND</b>	

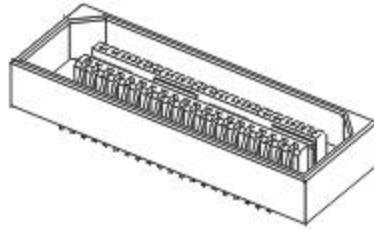
Table 2 – SPI-4.2 Receive Pin Assignments

Virtex-4 Pin #	LVDS Signal Name	J5 Connector Pin # LVDS RX		LVDS Signal Name	Virtex-4 Pin #
	<b>5.0V</b>	1	2	<b>5.0V</b>	
	<b>5.0V</b>	3	4	<b>5.0V</b>	
	<b>GND</b>	5	6	<b>GND</b>	
	<b>3.3V</b>	7	8	<b>3.3V</b>	
	<b>3.3V</b>	9	10	<b>3.3V</b>	
	<b>GND</b>	11	12	<b>GND</b>	
	<b>2.5V</b>	13	14	<b>2.5V</b>	
	<b>2.5V</b>	15	16	<b>2.5V</b>	
	<b>GND</b>	17	18	<b>GND</b>	
U5	RSCLK	19	20	NC	
	NC	21	22	NC	
	<b>GND</b>	23	24	<b>GND</b>	
J1	RSTAT0	25	26	NC	

J2	RSTAT1	27	28	NC	
	<b>GND</b>	29	30	<b>GND</b>	
AB4	RDat_N(15)	31	32	RDat_N(14)	AB2
AC4	RDat_P(15)	33	34	RDat_P(14)	AB3
	<b>GND</b>	35	36	<b>GND</b>	
AB5	RDat_N(13)	37	38	RDat_N(12)	AC1
AC5	RDat_P(13)	39	40	RDat_P(12)	AC2
AE3	RDat_N(11)	41	42	RDat_N(10)	AD1
AF3	TDat_P(11)	43	44	RDat_P(10)	AD2
	<b>GND</b>	45	46	<b>GND</b>	
AE4	RDat_N(9)	47	48	RDat_N(8)	AC3
AF4	RDat_P(9)	49	50	RDat_P(8)	AD3
	<b>GND</b>	51	52	<b>GND</b>	
AF5	RDat_N(7)	53	54	RDat_N(6)	Y7
AF6	RDat_P(7)	55	56	RDat_P(6)	AA7
	<b>GND</b>	57	58	<b>GND</b>	
Y9	RDat_N(5)	59	60	RDat_N(4)	AD4
AA9	RDat_P(5)	61	62	RDat_P(4)	AD5
	<b>GND</b>	63	64	<b>GND</b>	
Y1	RDat_N(3)	65	66	RDat_N(2)	V7
Y2	RDat_P(3)	67	68	RDat_P(2)	W7
	<b>GND</b>	69	70	<b>GND</b>	
Y8	RDat_N(1)	71	72	RDat_N(0)	AA10
AA8	RDat_P(1)	73	74	RDat_P(0)	Y10
	<b>GND</b>	75	76	<b>GND</b>	
AA1	RCtl_N	77	78	RDCLK_N	AF10
AB1	RCtl_P	79	80	RDCLK_P	AF11
	<b>GND</b>	81	82	<b>GND</b>	
	<b>GND</b>	83	84	<b>GND</b>	
	<b>GND</b>	85	86	<b>GND</b>	
	<b>GND</b>	87	88	<b>GND</b>	

### 3.1.3 LVDS Connector

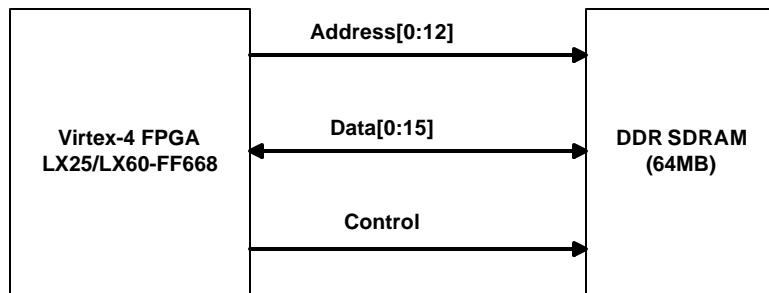
The design of the SPI-4.2 interface requires use of a high-speed and high quality connector. The V4MB development board uses the SAMTEC QSE type connector for this interface. The **QSE-040-01-L-Dx-A** connector from SAMTEC provides up to 28 LVDS signal connections in addition to an adequate number of ground connections for improving the signal quality. Two of these connectors are used on the V4MB development board to implement the SPI-4.2 interface. In addition, a mating LVDS extension cable is available from Samtec (part number #EQCD-040-06.00-TTR-TBL-1). The following figure shows the QSE type connector from SAMTEC (the picture is obtained from the SAMTEC web site (<http://www.samtec.com/>)).



**Figure 3 – SAMTEC QSE Type Connector for the SPI-4.2 Interface**

### 3.2 DDR SDRAM

The Virtex-4™ MB development board provides 64MB of DDR SDRAM memory (x16). A high-level block diagram of the DDR SDRAM interface is shown below followed by a table describing the SDRAM memory interface signals.



**Figure 4 – DDR SDRAM Interface**

**Table 3 – DDR SDRAM Interface Pin Assignments**

Signal Name	Description	FPGA Pin #
ddr_addr[0]	Address 0	N23
ddr_addr[1]	Address 1	K23
ddr_addr[2]	Address 2	N24
ddr_addr[3]	Address 3	J23
ddr_addr[4]	Address 4	V23
ddr_addr[5]	Address 5	P23
ddr_addr[6]	Address 6	U23
ddr_addr[7]	Address 7	P24
ddr_addr[8]	Address 8	T24
ddr_addr[9]	Address 9	R23
ddr_addr[10]	Address 10	K24
ddr_addr[11]	Address 11	T23
ddr_addr[12]	Address 12	R24
ddr_dq[0]	Data 0	K20
ddr_dq[1]	Data 1	J20
ddr_dq[2]	Data 2	L20

ddr_dq[3]	Data 3	J21
ddr_dq[4]	Data 4	M20
ddr_dq[5]	Data 5	K22
ddr_dq[6]	Data 6	N20
ddr_dq[7]	Data 7	J22
ddr_dq[8]	Data 8	T21
ddr_dq[9]	Data 9	P20
ddr_dq[10]	Data 10	T20
ddr_dq[11]	Data 11	T19
ddr_dq[12]	Data 12	U22
ddr_dq[13]	Data 13	P22
ddr_dq[14]	Data 14	U21
ddr_dq[15]	Data 15	U20
ddr_ba[0]	Bank Select 0	L23
ddr_ba[1]	Bank Select 1	M24
ddr_dm[0]	Write Mask0	M22
ddr_dm[1]	Write Mask1	N22
ddr_dqs[0]	Data Strobe0	M19
ddr_dqs[1]	Data Strobe1	N19
ddr_csn	Chip Select	L24
ddr_rasn	Row Address Strobe	M21
ddr_casn	Column Address Strobe	M23
ddr_wen	Write Enable	L21
ddr_clk	Clock	R20
ddr_clkn	Clock	R19
ddr_clke	Clock Enable	K21

### 3.3 Flash

The Virtex-4™ MB development board provides 4MB of flash memory (x16). A high-level block diagram of the flash interface is shown below followed by a table describing the flash memory interface signals.

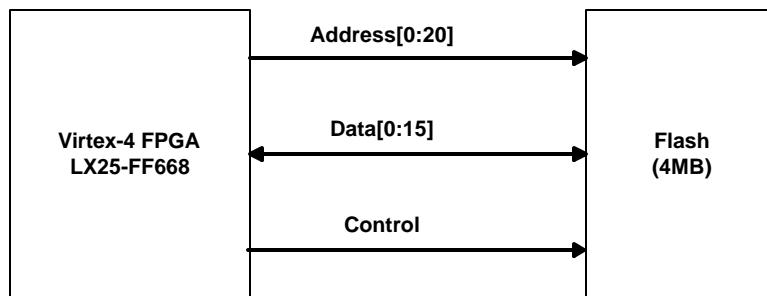


Figure 5 – Flash Interface

Table 4 – Flash Interface Pin Assignments

Signal Name	Description	FPGA Pin #
flash_addr[0]	Address 0	M2
flash_addr[1]	Address 1	T6
flash_addr[2]	Address 2	R5

flash_addr[3]	Address 3	P5
flash_addr[4]	Address 4	P6
flash_addr[5]	Address 5	P7
flash_addr[6]	Address 6	P8
flash_addr[7]	Address 7	N8
flash_addr[8]	Address 8	K6
flash_addr[9]	Address 9	J7
flash_addr[10]	Address 10	M7
flash_addr[11]	Address 11	M8
flash_addr[12]	Address 12	L8
flash_addr[13]	Address 13	K7
flash_addr[14]	Address 14	J4
flash_addr[15]	Address 15	J6
flash_addr[16]	Address 16	R3
flash_addr[17]	Address 17	N7
flash_addr[18]	Address 18	N5
flash_addr[19]	Address 19	L7
flash_addr[20]	Address 20	M6
flash_d[0]	Data 0	P2
flash_d[1]	Data 1	R2
flash_d[2]	Data 2	U1
flash_d[3]	Data 3	K3
flash_d[4]	Data 4	L3
flash_d[5]	Data 5	M4
flash_d[6]	Data 6	N4
flash_d[7]	Data 7	P3
flash_d[8]	Data 8	R1
flash_d[9]	Data 9	T1
flash_d[10]	Data 10	K4
flash_d[11]	Data 11	L4
flash_d[12]	Data 12	M3
flash_d[13]	Data 13	N3
flash_d[14]	Data 14	P4
flash_d[15]	Data 15	R4
flash_cen	Chip Select	M1
flash_oen	Output Enable	N2
flash_wen	Write Enable	J5
flash_rdy	Ready	M5
flash_reset	Reset	K5

### 3.4 Clock Sources

The Clock Generation section of the Virtex-4 MB board provides all the necessary clocks for a MicroBlaze processor, the I/O devices located on the board, as well as the DDR SDRAM memory. In general, the clock sources on the board are grouped into two categories; differential and single-ended clock sources. The differential clock sources are primarily used by the LVDS interface, while the single-ended clock sources are used by the processor section.

An on-board 100MHz oscillator provides the system clock input to the processor section. This 100Mhz clock will be used by the Virtex-4 Digital Clock Managers (DCMs) to generate various processor clocks. In addition to the above clock inputs, a socket is provided on the board that can

be used to provide single ended LVTTL clock input to the FPGA via an 8 or 4-pin oscillator. The following figure shows the clock resources on the Virtex-4 MB development board.

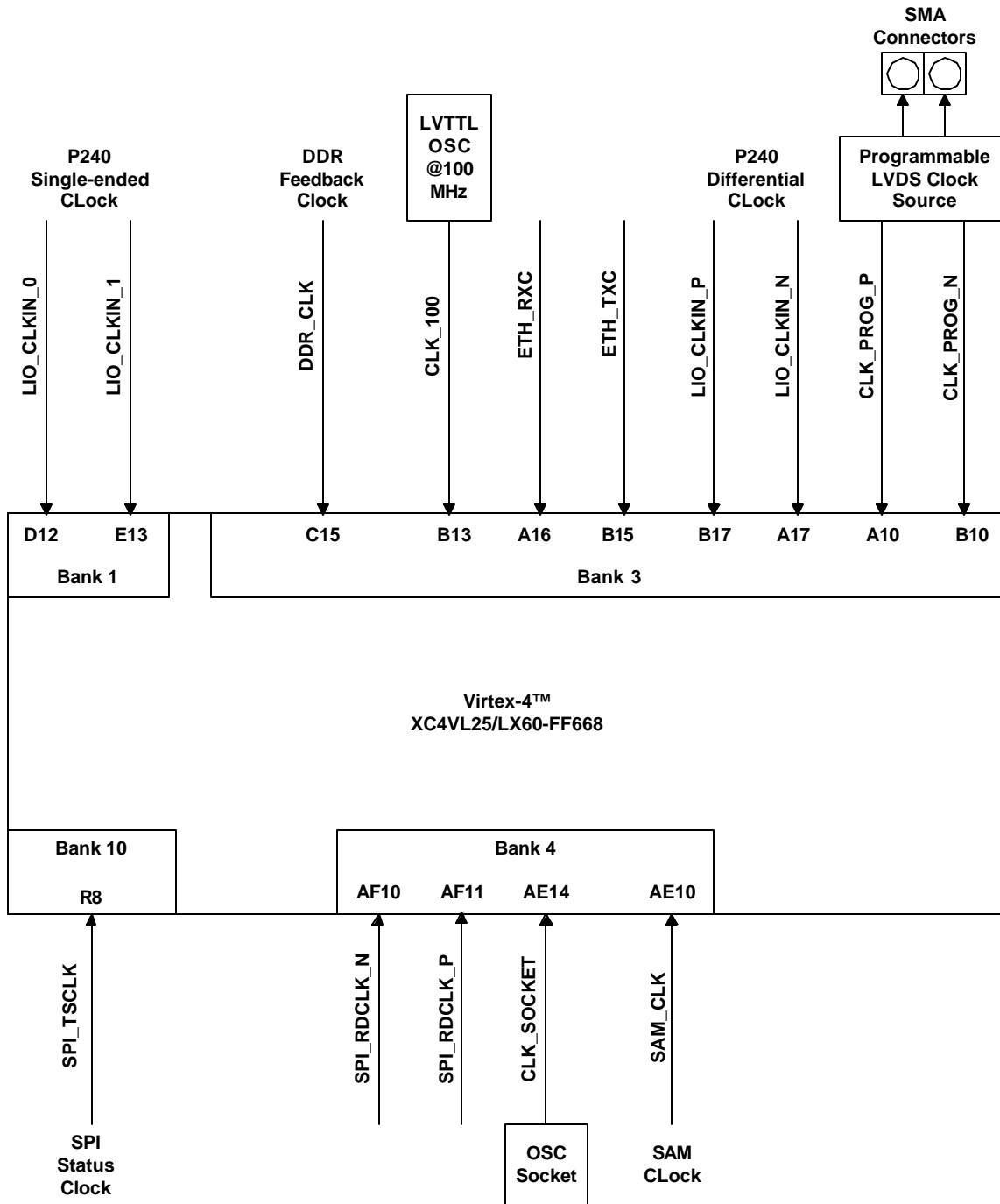


Figure 6 - Clock Sources on the Virtex-4 MB Board

The following table provides a brief description of each clock input to the Virtex-4 FPGA.

**Table 5 - Clock Inputs**

Signal Name	FPGA Pin #	Description
CLK_PROG_P, CLK_PROG_N	A10, B10	<b>Positive and Negative Differential System Clock Inputs</b> – These clock inputs are connected to the output of an LVDS clock synthesizer. This programmable clock source can generate a clock frequency of 25 to 700MHz. Refer to the Programmable LVDS Clock Source section for more information.
LIO_CLKIN_P, LIO_CLKIN_N	B17, A17	<b>P240 Module Differential Clock Input</b> – This clock input is connected to the P240 connector located on the Virtex-4 board.
LIO_CLKIN_0, LIO_CLKIN_1	D12, E13	<b>P240 Module Single-ended Clock Inputs</b> – These clock inputs are connected to the P240 connector located on the Virtex-4 board.
SPI_RDCLK_P, SPI_RDCLK_N	AF11, AF10	<b>Positive and Negative Differential SPI-4.2 Receive Clock Inputs</b> – These clock inputs are connected to the LVDS receive connector on the Virtex-4 MB board. For the SPI-4.2 applications, these clock inputs are the SPI-4.2 receive clock outputs.
DDR_CLK	C15	<b>DDR Feedback Clock Input</b> – This clock input is connected to the DDR clock.
CLK_100	B13	<b>System Clock</b> – This clock input is connected to a 100MHz LVTTL oscillator.
CLK_SOCKET	AE14	<b>LVTTL Clock Input</b> – LVTTL socket on the Virtex-4 board.
SPI_TSCLK	R8	<b>SPI-4.2 Transmit Status Clock Input</b> – This clock input is connected to the SPI-4.2 transmit status clock output.
ETH_RXC	A16	<b>Ethernet Receive Clock Input</b> – This clock input is connected to the Ethernet receive clock.
ETH_TXC	B15	<b>Ethernet Transmit Clock Input</b> – This clock input is connected to the Ethernet transmit clock.
SAM_CLK	AE10	<b>SystemACE Module Clock Input</b> – This clock input is connected to the SystemACE Module connector.

### 3.4.1 Programmable LVDS Clock Source

A programmable LVDS clock synthesizer is used on the Virtex-4 MB development board to generate a reference clock input to the LVDS interface. The use of this variable clock source, allows designers to prototype various interconnect technologies with different clock source requirements. The differential output port is also well suited for DSP applications when driving external DACs or ADCs.

### 3.4.2 ICS8442 Programmable LVDS Clock Synthesizer

The Virtex-4 MB development board design uses the ICS8442 LVDS clock synthesizer for generating various clock frequencies. A list of features included in the ICS8442 device is shown below.

- Output frequency range: 25MHz to 700MHz
- RMS period jitter: 2.7ps (typical)
- Cycle-to-cycle jitter: 27ps (typical)
- Output rise and fall time: 650ps (maximum)
- Output duty cycle: 48/52

The following figure shows a high-level block diagram of the ICS8442 programmable LVDS clock synthesizer.

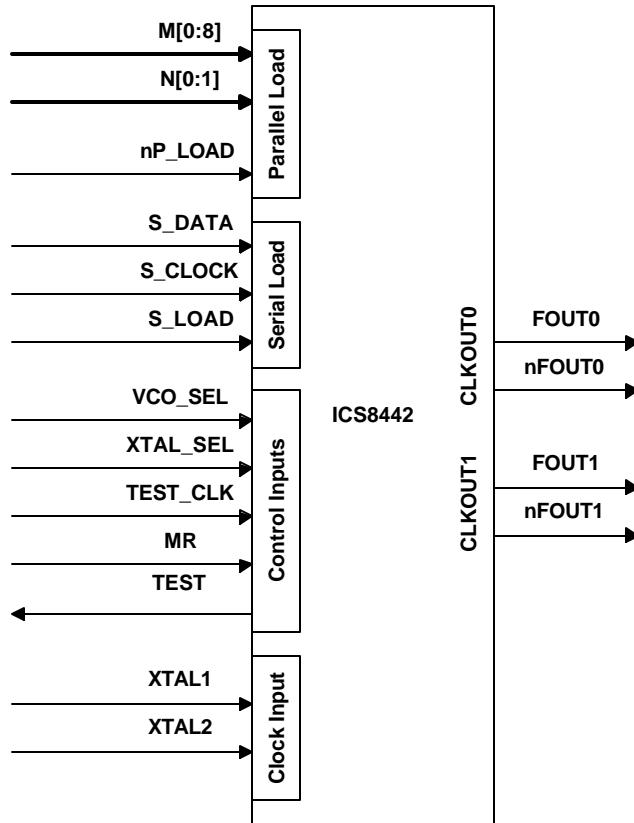


Figure 7 – ICS8442 Clock Synthesizer

Table 6 – ICS8442 Clock Synthesizer Pin Description

Signal Name	Direction	Pull up/Pull down	Description
M[0:4], M[6:8]	Input	Pull down	The M divider inputs, latched on the rising edge of the nP_LOAD signal.
M[5]	Input	Pull up	
N[0:1]	Input	Pull down	The N divider inputs, latched on the rising edge of the nP_LOAD signal.
TEST	Output		The TEST output is active during the serial mode of operations. Please refer to the datasheet for more information.
MR	Input	Pull down	Active high reset signal.
S_CLOCK	Input	Pull down	Serial interface clock input. Data is shifted into the device on the rising edge of this clock.
S_DATA	Input	Pull down	Serial interface data input.
S_LOAD	Input	Pull down	Serial interface load signal. The contents of the serial data shift register is loaded into the internal dividers on the rising edge of this signal.
TEST_CLK	Input	Pull down	Test clock input.
nP_LOAD	Input	Pull down	The rising edge of this signal is used to load the M and N divider inputs into the device.
XTAL1, XTAL2	Input		Crystal clock input/output

XTAL_SEL	Input	Pull up	This signal is used to select between the crystal and the TEST_CLK input to the device. When this high, crystal is selected.
VCO_SEL	Input	Pull up	This signal is used to place the internal PLL in the bypass mode. When this signal is set to low, the PLL is placed in the bypass mode. For normal operations, this signal must be set to high.
FOUT0, FOUT1	Output		Positive LVDS clock outputs
nFOUT0, nFOUT1	Output		Negative LVDS clock outputs

The Input Clock Select signals of the ICS8442 can be used to provide a reference clock input to the device other than the 25MHz crystal oscillator (for test purposes). The following table shows how these Input Clock Select signals are used to generate the output clock or to test the ICS8442 device. Please refer to the ICS8442 datasheet for more information on using the TEST\_CLK clock input.

**Table 7 – Input Clock Select Signal Description**

VCO_SEL	XTAL_SEL	Reference Clock Input	FOUT[0:1]
0	0	TEST_CLK	TEST_CLK/N (the TEST_CLK must be between 10 and 25MHz). This mode can be used to test the ICS8442 device by routing the input clock to the outputs.
0	1	25MHz crystal	25MHz crystal/N (This mode can be used to test the ICS8442 device by routing the 25MHz crystal clock to the outputs).
1	0	TEST_CLK	ICS8442 PLL Output/N (Normal Operation)
1	1	25MHz crystal	ICS8442 PLL Output/N (Normal Operation)

### 3.4.3 ICS8442 Clock Generation

The ICS8442 output clocks are generated based on the following formula (assuming the crystal clock input is set to 25MHz):

$$FOUT[0:1] = 25 \times M/N$$

Where  $8 < M < 28$  and N can take a value of 1, 2, 4, or 8. The variable M is determined by setting the binary number M[0:8] while N is set according to the following table:

**Table 8 – ICS8442 N Settings**

N[1:0]	N	Output Clock Frequency Range (MHz)	
		Minimum	Maximum
00	1	200	700
01	2	100	350
10	4	50	175
11	8	25	87.5

For example, to generate a 62.5MHz clock, N[1:0] will be set to "10" (it can also be set to "11" since either one will be the correct frequency range for the 62.5MHz clock) and M will be set to "000001010" (decimal 10). So, from the above formula:

$$FOUT[0:1] = 25 \times 10/4 = 62.5\text{Mhz}$$

The following table shows how the M and N values can be set to generate a clock source for a few common applications. All the values for M and N are based on the 25MHz crystal clock input to the ICS8442 device. A complete list of frequencies generated by the ICS8442 (based on a 25MHz input clock) is provided in the following sections.

**Table 9 – Examples of the ICS8442 M and N Settings**

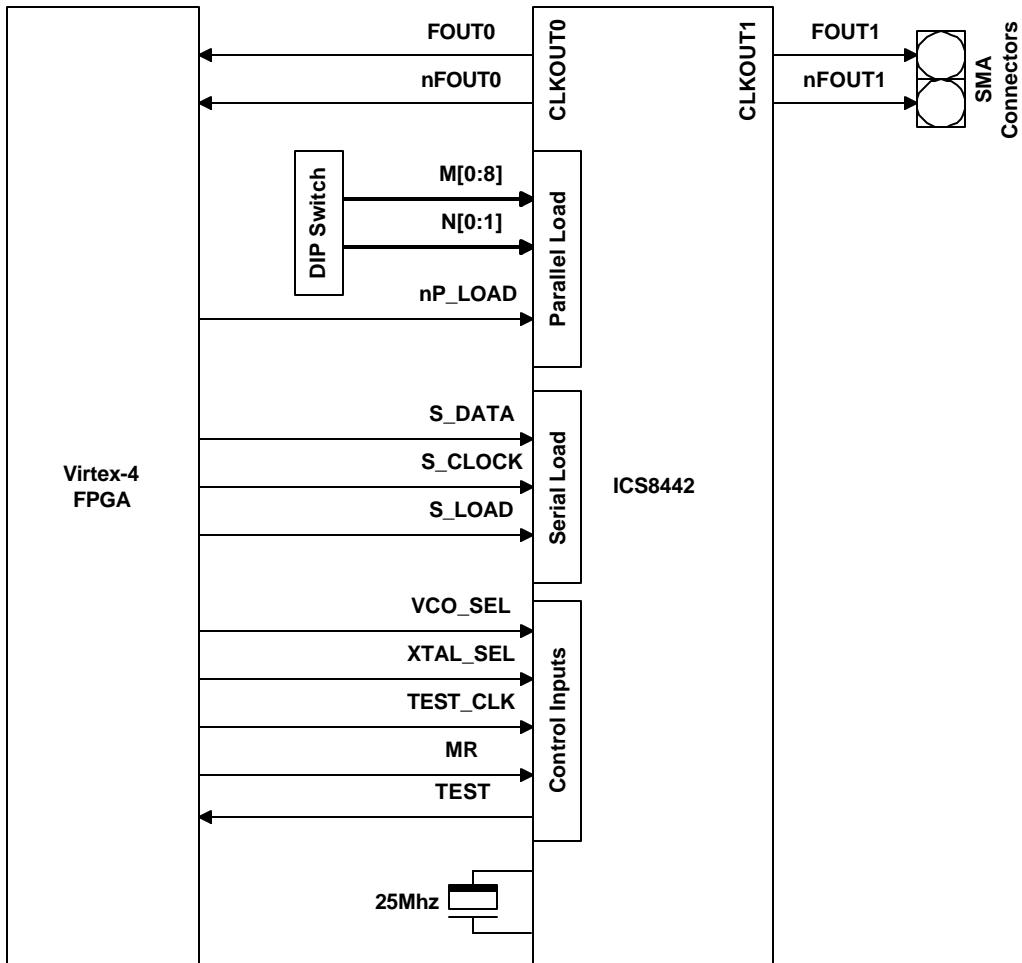
Interconnect Technology	FOUT0 and FOUT1 (MHz)	ICS8442 M and N Settings										
		M8	M7	M6	M5	M4	M3	M2	M1	M0	N1	N0
Gigabit Ethernet	62.5	0	0	0	0	0	1	0	1	0	1	0
Fiber Channel	53.125	0	0	0	0	1	0	0	0	1	1	1
	106.25	0	0	0	0	1	0	0	0	1	1	0
Infiniband	125	0	0	0	0	1	0	1	0	0	1	0
XAUI	156.25	0	0	0	0	1	1	0	0	1	1	0

### 3.4.4 ICS8442 Programming Modes

The ICS8442 provides two different methods of programming the M and N values into the device; a Parallel Mode and a Serial Mode. In parallel mode, M and N values are programmed into the device when the nP\_LOAD signal pulses low. In the serial mode, the I2C pins (S\_DATA and S\_CLOCK) along with the S\_LOAD signal are used to shift the M and N values into the device. Please refer to the ICS8442 datasheet for more information on programming modes of loading the M and N values into the device.

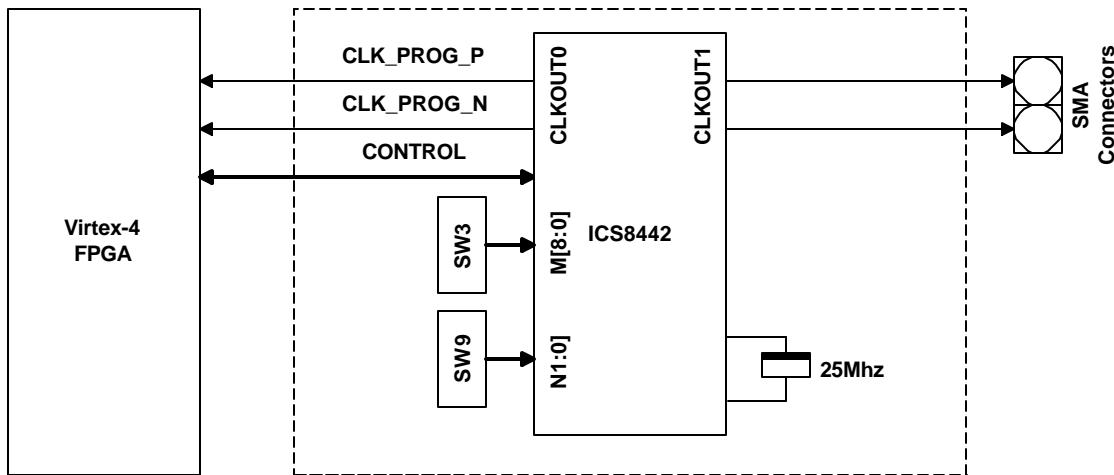
### 3.4.5 ICS8442 M and N Settings

The following figure shows how the ICS8442 programmable LVDS clock synthesizer is used on the Virtex-4 MB board. DIP Switches are provided on the board for manual setting of the M and N values.



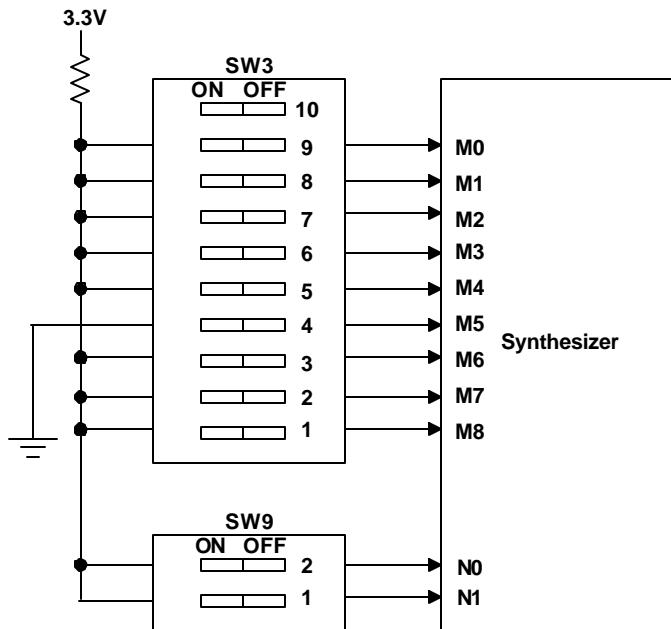
**Figure 8 – ICS8442 Clock Synthesizer Interface to the FPGA**

As shown in the above figure, the ICS8442 device outputs two identical LVDS clock sources. One of these clock sources can be used to provide the reference clock input to the LVDS interface on the Virtex-4 MB development board, while the other clock output can be used to trigger a scope during testing. The second output could also be used to provide a low jitter, LVDS clock source to a user board, such as the P240 module.



**Figure 9 – ICS8442 Clock Synthesizer M and N DIP Switches**

The following tables show the DIP Switch settings for M and N selections. Please refer to Table 6 for the information on pull-up and pull-down resistors provided internal to the ICS8442 device for the M and N input signals.



**Figure 10 – M and N DIP Switches for the Synthesizers**

**Table 10 – DIP Switch Setting for M[8:0]**

SW1, SW10, and SW2	M[8:0]	Switch Position	
		OFF	ON
DIP1	M8	0	1
DIP2	M7	0	1
DIP3	M6	0	1
DIP4	M5	1	0 Note (1)
DIP5	M4	0	1
DIP6	M3	0	1
DIP7	M2	0	1
DIP8	M1	0	1
DIP9	M0	0	1
DIP10	Unused	NA	NA

Note(1) – The polarity of M5 (DIP4) is the opposite of all other DIP switch positions.

**Table 11 – DIP Switch Setting for N[1:0]**

SW9, SW11, and SW13	N[1:0]	Switch Position	
		OFF	ON
DIP1	N1	0	1
DIP2	N0	0	1

The following table shows a complete list of frequencies generated by the ICS8442 device based on a 25MHz crystal reference clock input.

**Table 12 – Synthesizer Clock Outputs for M and N Values**

M[8:0]	N[1:0]	FOUT[1:0] (MHz)	M[8:0]	N[1:0]	FOUT[1:0] (MHz)
000001000	11	25 (Min)	000011000	10	150
000001001	11	28.125	000011001	10	156.25
000001010	11	31.25	000001101	01	162.5
000001011	11	34.375	000011010	10	162.5
000001100	11	37.5	000011011	10	168.75
000001101	11	40.625	000001110	01	175
000001110	11	43.75	000011100	10	175
000001111	11	46.875	000001111	01	187.5
000001000	10	50	000001000	00	200
000010000	11	50	000010000	01	200
000010001	11	53.125	000010001	01	212.5
000001001	10	56.25	000001001	00	225
000010010	11	56.25	000010010	01	225
000010011	11	59.375	000010011	01	237.5
000001010	10	62.5	000001010	00	250
000010100	11	62.5	000010100	01	250
000010101	11	65.625	000010101	01	262.5
000001011	10	68.75	000001011	00	275
000010110	11	68.75	000010110	01	275
000010111	11	71.875	000010111	01	287.5
000001100	10	75	000001100	00	300
000011000	11	75	000011000	01	300

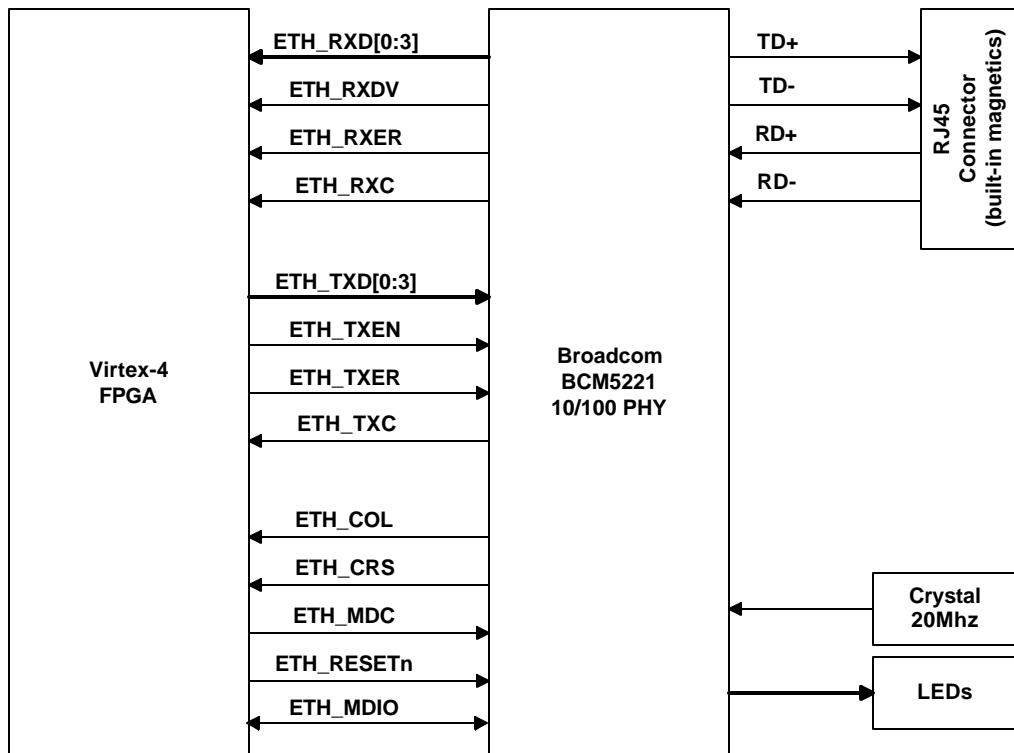
000011001	11	78.125	000011001	01	312.5
000001101	10	81.25	000001101	00	325
000011010	11	81.25	000011010	01	325
000011011	11	84.375	000011011	01	337.5
000001110	10	87.5	000001110	00	350
000011100	11	87.5	000011100	01	350
000001111	10	93.75	000001111	00	375
000001000	01	100	000010000	00	400
000010000	10	100	000010001	00	425
000010001	10	106.25	000010010	00	450
000001001	01	112.5	000010011	00	475
000010010	10	112.5	000010100	00	500
000010011	10	118.75	000010101	00	525
000001010	01	125	000010110	00	550
000010100	10	125	000010111	00	575
000010101	10	131.25	000011000	00	600
000001011	01	137.5	000011001	00	625
000010110	10	137.5	000011010	00	650
000010111	10	143.75	000011011	00	675
000001100	01	150	000011100	00	700 (Max)

**Table 13 – FPGA Pin Assignments for the Synthesizer Interface**

Signal Name	Virtex-4 Pin #	Comments
SYNTH_PLOAD	V2	This input is used to load the M and N values into the synthesizer using the parallel mode configuration along with the DIP switch settings for M and N.
SYNTH_RESET	K1	This input signal resets the synthesizer.
SYNTH_SCLK	L1	This clock input is used to load the M and N values into the synthesizer using serial mode configuration.
SYNTH_SDATA	T4	Serial data input to the synthesizer for loading the M and N values.
SYNTH_SLOAD	T3	This input signal is used to load the M and N values into the synthesizer using the serial mode configuration.
SYNTH_TEST	U2	A test clock input can be provided to the synthesizer using this clock input.
SYNTH_VCOSEL	V1	This input signal can be used to bypass the PLL for test purposes.
SYNTH_XTALSEL	U3	This input signal is used to select between the test clock input and the on-board crystal as clock source to the synthesizer.
SYNTH_DOUT	K2	This output signal is used as the test clock output.

### 3.5 10/100 Ethernet PHY

The Virtex-4 MB development board provides a 10/100 Ethernet port for network connection. A high-level block diagram of the 10/100 Ethernet interface is shown in the following figure followed by FPGA pin assignments for this interface.



**Figure 11 – 10/100 Ethernet Interface**

The following table shows the FPGA pin assignments for the Ethernet interface.

**Table 14 – Ethernet Pin Assignments**

Signal Name	Virtex-4 Pin #
ETH_TXC	B15
ETH_RXC	A16
ETH_CRS	B14
ETH_RXDV	H1
ETH_RXD[0]	G4
ETH_RXD[1]	G3
ETH_RXD[2]	F3
ETH_RXD[3]	H4
ETH_COL	C14
ETH_RXER	H2
ETH_TXEN	G2
ETH_TXER	G1
ETH_TXD[0]	H5
ETH_TXD[1]	H6
ETH_TXD[2]	F1
ETH_TXD[3]	C12
ETH_MDC	F4
PHY_RESETn	E3
ETH_MDIO	H3

### 3.6 LCD Panel

The Virtex-4 MB development board provides an 8-bit interface to a 2x16 LCD panel (MYTECH MOC-16216B-B). The following table shows the LCD interface signals.

**Table 15 – LCD Interface Signals**

Signal Name	Description	Virtex-4 Pin #
D0	LCD Data Bit 0	K25
D1	LCD Data Bit 1	P19
D2	LCD Data Bit 2	AC10
D3	LCD Data Bit 4	AB10
D4	LCD Data Bit 4	AF12
D5	LCD Data Bit 5	AE12
D6	LCD Data Bit 6	AC17
D7	LCD Data Bit 7	AB17
EN	LCD Enable Signal	L19
RW	LCD Write Signal (this signal is connected to logic "0" on the Virtex-4 MB board, enabling write only cycles).	
RS	LCD Register Select Signal	N21

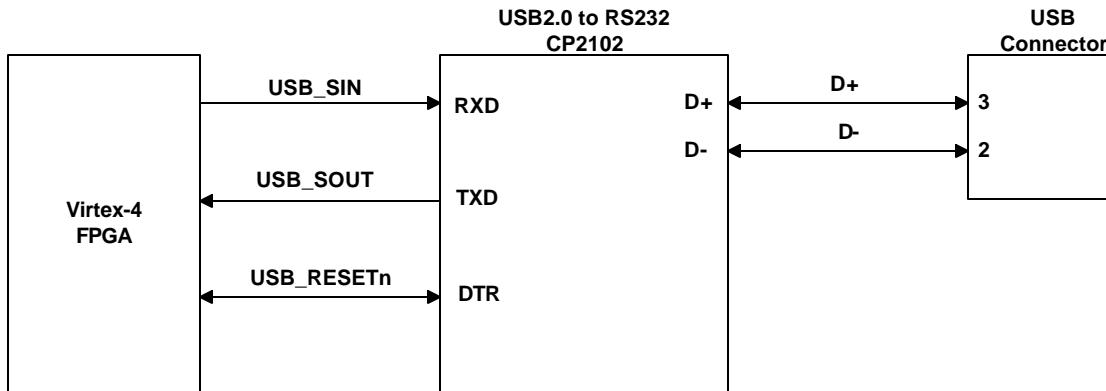
### 3.7 USB 2.0 to RS232 Port

The Virtex-4 MB development board implements a USB 2.0 port. This is accomplished using the Cygnal CP2101 USB-to-UART Bridge Controller. The FPGA interfaces to the CP2102 as a simple UART. The UART interface to the CP2102 can run at speeds ranging from 300 to 921,600 baud.

The CP2102 is a highly integrated USB-to-UART Bridge Controller, providing a simple solution for USB serial communications using a minimum of components and PCB space. The CP2102 includes a USB 2.0 full-speed function controller, USB transceiver, oscillator, EEPROM, and asynchronous serial data bus (UART) with full modem control signals in a compact 5mm X 5mm MLP-28 package. No other external USB components are required.

The on-chip EEPROM may be used to customize the USB Vendor ID, Product ID, Product Description String, Power Descriptor, Device Release Number, and Device Serial Number as desired. The EEPROM is programmed on-board via the USB allowing the programming step to be easily integrated into the product manufacturing and testing process.

Royalty-free Virtual COM Port (VCP) device drivers provided by Cygnal allow the Virtex-4 MB development board to appear as a COM port to PC applications. The CP2102 UART interface implements all RS232 signals, including control and handshaking signals. These signals are interfaced to the Virtex-4 FPGA as follows:



**Figure 12 – USB 2.0 to RS232 Serial Interface**

The following table shows the RS232 interface signal names and their Virtex -4 FPGA pin assignments.

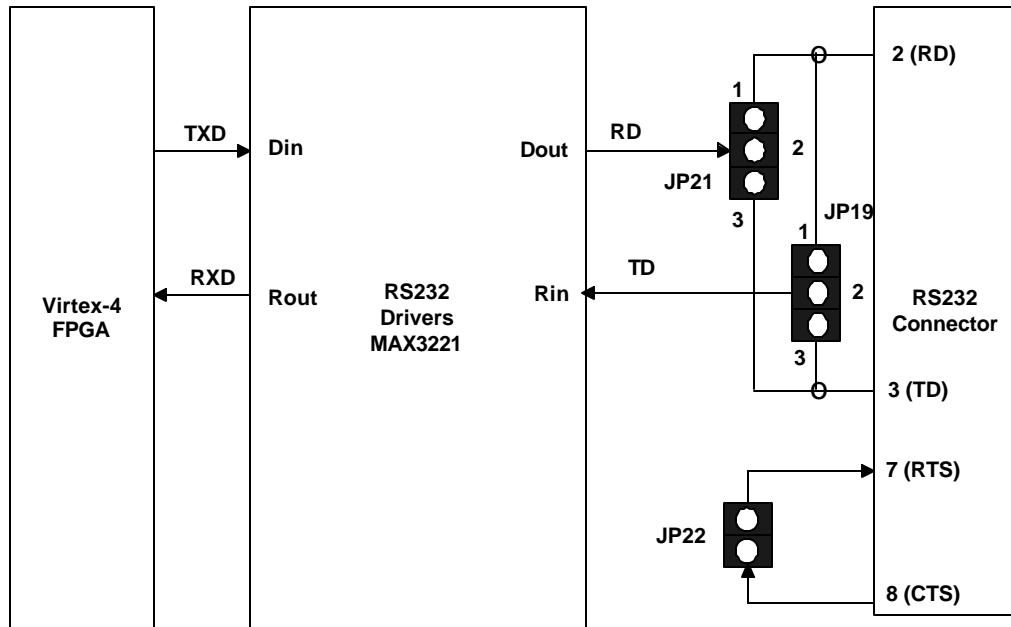
**Table 16- USB 2.0 to RS232 Port Signal Description**

FPGA Signal Name	Virtex-4 Pin #	Description
<b>RS232 Signals</b>		
USB_SIN	R7	RS232 receive signal
USB_SOUT	L6	RS232 transmit signal
<b>USB 2.0 Signals</b>		
D+	NA	USB D+ signal
D-	NA	USB D- signal
<b>Common Signal</b>		
USB_RESETn	R6	CP2102 reset signal

To use the USB port, the CP2102 device drivers must be installed. These drivers are included on the Virtex-4 MB Development Kit CD and contained in the self-extracting file **CP2101.exe**. To install the CP2101/2 virtual COM port device drivers, refer to Appendix A.

### 3.8 RS232

The Virtex-4 MB development board provides an RS232 interface with RX and TX signals and jumpers for connecting the RTS and CTS signals. The following figure shows the RS232 interface to the Virtex-4 LX25/LX60/SX35 FPGA.



**Figure 13 - RS232 Interface**

**Table 17 – RS232 Signals**

Signal Name	Description	Virtex-4 Pin #
RS232_RXD	Received Data, RD	U4
RS232_TXD	Transmit Data, TD	V4

**Table 18 - RS232 Jumper Settings**

Mode of Operation	JP19	JP21
DCE	Install a jumper on pins 2-3	Install a jumper on pins 1-2
DTE	Install a jumper on pins 1-2	Install a jumper on pins 2-3

A Jumper must be installed on JP22, if RTS and CTS signal connections are needed.

### 3.9 User DIP and PB Switches

The Virtex-4 MB development board provides four user push button switches as described in the following table. An active low signal is generated when a given switch is pressed.

**Table 19 – Push Button Switch Pin Assignments**

Signal Name	Description	Virtex-4 Pin #
PUSH1	SW5	E2
PUSH2	SW6	E1
PUSH3	SW7	G10
PUSH4	SW8	G9

The Virtex-4 MB development board provides an 8-position DIP switch as described in the following table. An active low signal is generated when a given switch is ON.

**Table 20 – DIP Switch Pin Assignments**

Signal Name	Description	Virtex-4 Pin #
DIP1	User Switch Input 1	C10
DIP2	User Switch Input 2	D10
DIP3	User Switch Input 3	B6
DIP4	User Switch Input 4	C6
DIP5	User Switch Input 5	B4
DIP6	User Switch Input 6	A4
DIP7	User Switch Input 7	D2
DIP8	User Switch Input 8	D1

### 3.10 User LEDs

The Virtex-4 MB development board provides four user LEDs that can be turned “ON” by driving the LED<sub>x</sub> signal to logic “0”. The following table shows the user LEDs and their associated Virtex-4 FPGA pin assignments.

**Table 21 – LED Pin Assignments**

LED Designation	LED #	Virtex-4 Pin #
DS9	LED1	N25
DS10	LED2	V25
DS11	LED3	L26
DS12	LED4	K26

### 3.11 VBAT Jumper

A 3-pin jumper is used to provide user access to the VBAT input of the FPGA. If user is not sourcing the VBAT voltage, a jumper must be installed on pins 1-2 of the JP27 jumper. User can source voltage to the VBAT input via pins 2 and 3 of this jumper. The following table shows the pin assignments for the VBAT jumper.

**Table 22 – VBAT Jumper (JP27)**

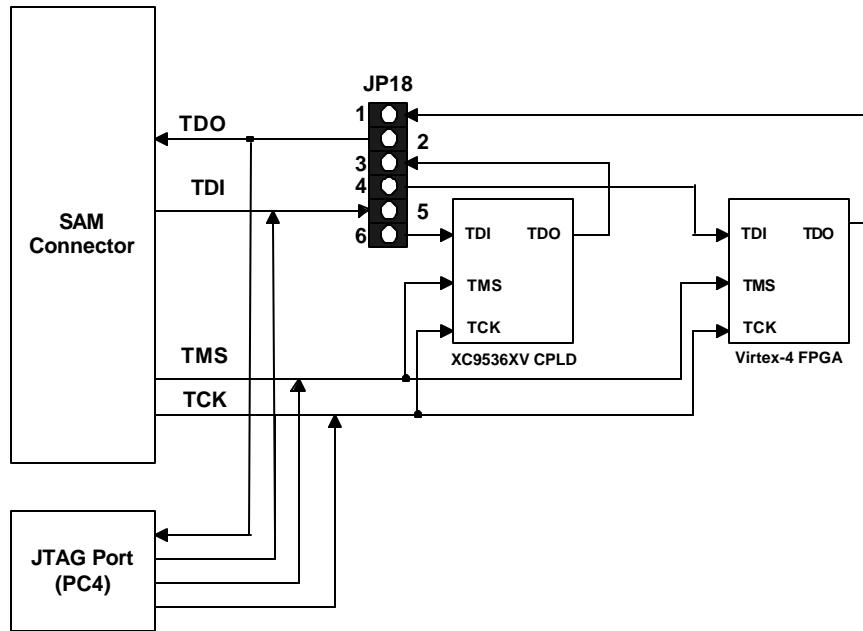
Pin Number	Description
1	2.5V
2	VBAT
3	Ground

### 3.12 Configuration and Debug Ports

Various methods of configuration and debug support are provided on the Virtex-4 MB development board to assist designers during the testing and debugging of their applications. The following sections provide brief descriptions of each of these interfaces.

#### 3.12.1 JTAG Chain

The following figure shows the JTAG chain on the Virtex-4 MB development board. The XC9536XV along with a serial flash is used to configure the FPGA. The serial flash programming procedure is explained in section 3.11.3.

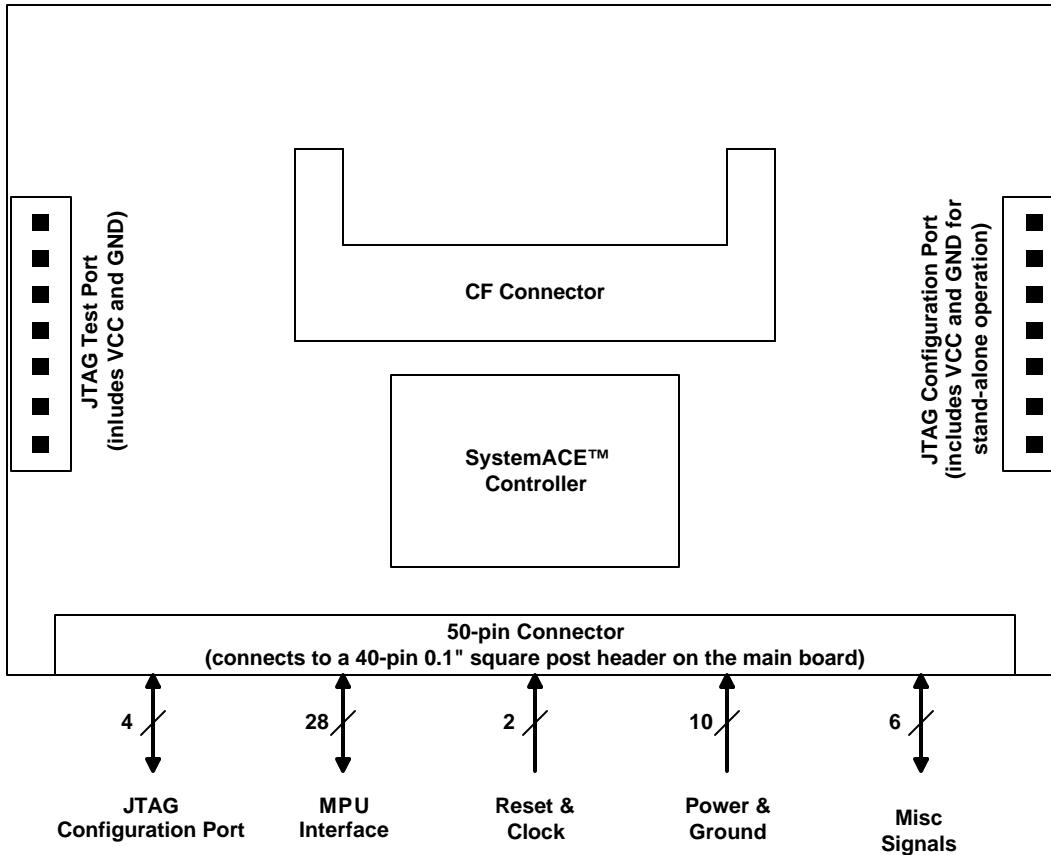


**Figure 14 – Vitex-4 MB Development Board JTAG Chain**

### 3.12.2 System ACE Module Connector

The Virtex-4 MB development board provides the SAM 50-pin connector on the board for using the Memec System ACE Module (SAM). The SAM can be used to configure the FPGA or to provide bulk flash memory to the MicroBlaze processor.

The Virtex-4 MB development board provides a System ACE interface that can be used to configure the Virtex-4 FPGA. The interface also gives software designers the ability to run real-time operating systems (RTOS) from removable CompactFlash cards. The Memec System ACE module (DS-KIT-SYSTEMACE) can be used to perform both of these functions. The figure below shows the System ACE module connected to the header on the Virtex-4 board.



**Figure 15 – SystemACE Module**

### 3.12.2.1 System ACE Controller Signal Description

The following table shows the System ACE Module signal assignments to the FPGA I/O pins.

**Table 23 - SAM Interface Signals**

Virtex-4 Pin #	System ACE Signal Name	SAM Connector Pin # (JP16)		System ACE Signal Name	Virtex-4 Pin #
	<b>3.3V</b>	1	2	<b>3.3V</b>	
	TDO	3	4	<b>GND</b>	
	TMS	5	6	CLOCK	AE10
	TDI	7	8	<b>GND</b>	
	PROGRAMn	9	10	TCK	
	<b>GND</b>	11	12	<b>GND</b>	
AB7	OE <sub>n</sub>	13	14	INIT <sub>n</sub>	
AB6	MPA0	15	16	WE <sub>n</sub>	V6
Y5	MPA2	17	18	MPA1	AC9
	<b>2.5V</b>	19	20	MPA3	AD8
V5	MPD00	21	22	<b>2.5V</b>	
AC7	MPD02	23	24	MPD01	AC8
AC6	MPD04	25	26	MPD03	AD6
AF9	MPD06	27	28	MPD05	AB9
AF8	MPD08	29	30	MPD07	AE9

AD10	MPD10	31	32	MPD09	AF7
AD11	MPD12	33	34	MPD11	AD12
AE6	MPD14	35	36	MPD13	AE13
W6	MPA4	37	38	MPD15	AA4
AA3	MPA6	39	40	MPA5	Y3
Y4	IRQ	41	42	GND	
W3	RESETn	43	44	CEn	W5
	DONE	45	46	BRDY	W4
	CCLK	47	48	BITSTREAM	
	GND	49	50	NC	

### 3.12.3 Serial Data Flash

This section describes the procedure for programming the Atmel serial data flash on the Memec Virtex-4 MB development board. This serial flash along with a CPLD is used to configure the Virtex-4 FPGA located on the development board on power up. The following figure shows a high-level block diagram of the serial flash interface to the Virtex-4 FPGA.

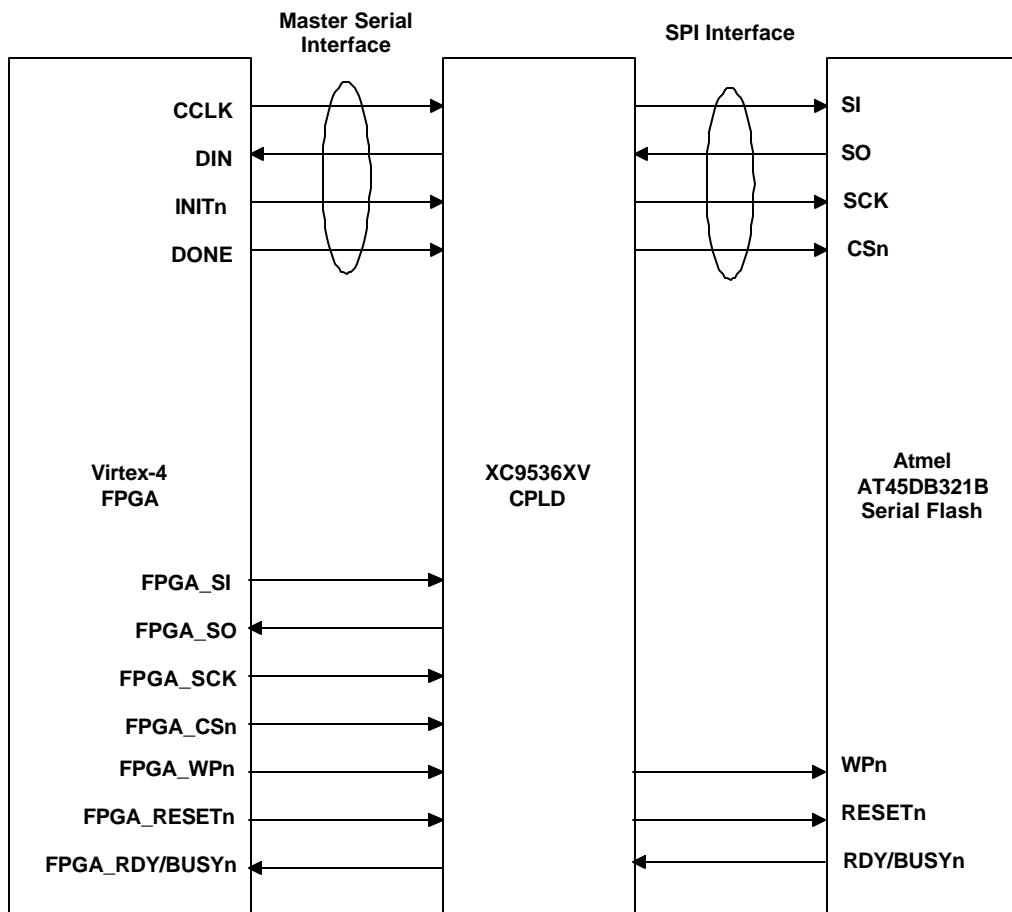


Figure 16 – Virtex-4 MB Development Board Configuration Interface

An interface is provided between the FPGA and the CPLD to allow access to the serial flash after the FPGA has been configured. This interface uses FPGA I/O pins to interface to the serial flash via the SPI port. The Virtex-4 FPGA uses 8Mb/18.3Mb/14.5Mb (LX25/LX60/SX35) of the serial flash memory for configuration and this interface allows the rest of the flash to be used for general-purpose application after the FPGA has been configured. The following table shows the signals used to implement the interface between the FPGA and the CPLD after the FPGA has been configured.

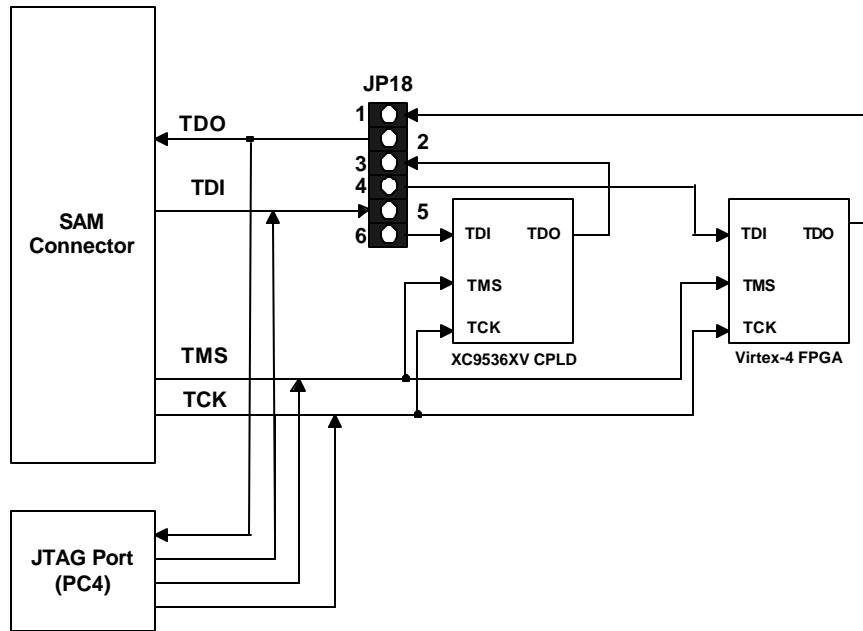
**Table 24 – FPGA SPI Interface Pin Assignments**

Signal Name	Description	Virtex-4 Pin #
FPGA_SI	Serial Flash SPI port data input signal	R26
FPGA_SO	Serial Flash SPI port data output signal	T26
FPGA_SCK	Serial Flash SPI port clock input signal	P25
FPGA_CSn	Serial Flash SPI port chip select input signal	U26
FPGA_WPn	Serial Flash SPI port write protect input signal	U25
FPGA_RESETn	Serial Flash SPI port reset input signal	M25
FPGA_RDY/BUSYn	Serial Flash SPI port ready output signal	V26

The primary function of the CPLD is to translate the Master Serial interface to the SPI interface of the serial flash. The XC9536XV CPLD uses the FPGA CCLK clock along with the INITn and DONE signals to drive the SPI SI, SCK and CSn signals. The SO output of the serial flash is used by the CPLD to drive the DIN signal of the FPGA. For more information on detail of the CPLD design, please refer to the Xilinx XAPP800.

### 3.12.3.1 JTAG Chain on the Virtex-4 MB Development Board

The following figure shows the JTAG chain on the Virtex-4 MB development board. As mentioned in the above section, the CPLD is used for interfacing to the configuration flash and does not provide any user logic. Hence, this CPLD is programmed by Memec prior to shipping the board. The programming file for the CPLD is provided in case re-programming of the CPLD becomes necessary. **The CPLD must be programmed prior to performing any operations on the serial flash such as erasing, programming, reading or verifying.**



**Figure 17 – Virtex-4 MB Development Board JTAG Chain**

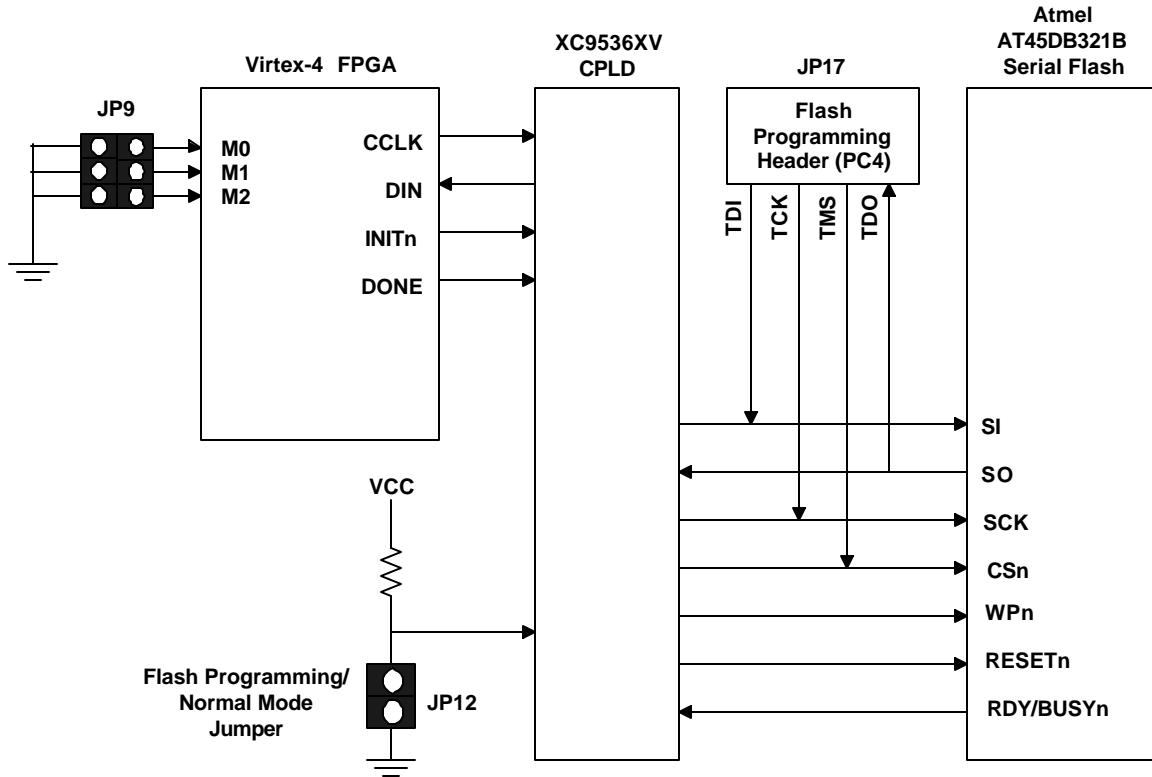
The following table shows jumper settings for the JTAG chain on the Virtex-4 MB development board. Since CPLD is already programmed by Memec prior to shipment, the board is shipped with jumpers installed on pins 1-2 and 4-5 (FPGA only, in the JTAG chain).

**Table 25 – JTAG Chain Jumper Settings**

Devices in the JTAG Chain	JP18 Jumpers Installed
CPLD and FPGA	Pins 1-2, 3-4 and 5-6
CPLD	Pins 2-3 and 5-6
FPGA	Pins 1-2 and 4-5

### 3.12.3.2 Configuration Flash on the Virtex-4 MB Development Board

The following figure shows the detail interface between the FPGA and the serial flash. A PC4 cable is used to program the serial flash with the FPGA bitstream. Once the flash is programmed, the CPLD will read the data from the flash and configure the FPGA over the Master Serial interface.



**Figure 18 – Serial Flash Configuration Interface**

### 3.12.3.3 Procedure for Programming the Serial Flash

1. The Memec Virtex-4 MB development board is shipped with a self-extracting zip file called **Serial\_Flash\_Programming**. Double-click on this self-extracting zip file to unzip it. After unzipping this file, a folder called **C:\Flash\_Utility**s is created.
2. In order to program the flash, flash programming utilities included in the **xapp800** must be downloaded from the following web site:  
<http://www.xilinx.com/products/xaw/coolvhdlq.htm>
3. Click on the above link to download the **xapp800** zip file and unzip it to a temporary folder on your hard drive. You need to register prior to downloading the xapp800 zip file.
4. Copy **xmcsutil.exe** and **xspi\_at.exe** files from this temporary folder to the **C:\Flash\_Utility**s folder. The following table shows the contents of the **C:\Flash\_Utility**s folder after copying these two executable files.

**Table 26 – Files in the Flash\_Utility Folder**

File Name	Description
-----------	-------------

xmcsutil.exe	A utility that is used to reverse the bytes in an MCS file. This is needed by the xspl_at utility.
xspl_at.exe	This utility is used to erase, program and verify the Atmel serial flash on the Virtex-4 MB development board.
prog_flash.bat	This batch file calls the xmcsutil and xspl_at utilities to erase, program and verify the Atmel serial flash on the Virtex-4 MB development board.
spi_cpld.jed	Programming file for the XC9536XV CPLD

5. Generate a bit file for the FPGA
6. Use iMPACT to generate an MCS file for the bit file generated in the previous step. When generating the MCS file, select a single platform flash device that will hold the entire design configuration bits. The following table shows the platform flash devices that must be used when generating the MCS file in iMPACT for the Virtex-4 MB board:

**Table 27 – Platform Flash Selection**

FPGA	Platform Flash Used
LX25	XCF08P, XCF16P or XCF32P
LX60	XCF32P
Sx35	XCF16P or XCF32P

7. Un-install JP9 jumpers.
8. Make sure JP12 jumper is un-installed. When JP12 jumper is un-installed, the CPLD outputs are placed in the tri-state mode allowing the Flash Programming Header to drive the serial flash SPI bus.
9. Connect a PC4 cable to the Flash Programming Header (JP17) and power up the Virtex-4 MB development board.
10. Copy the MCS file to the **C:\Flash\_Utilities** folder
11. Open a DOS window in the **C:\Flash\_Utilities** folder and enter the following command to program the serial flash:

**C:\ Flash\_Utilities > prog\_flash design\_name.mcs flash\_part\_number**

Where:      **design\_name.mcs** -> The mcs file generated using the bit file

**flash\_part\_number** -> Either AT45DB321B or AT45DB321C (V4MB board is populated with one or the other device).

The prog\_flash batch file will erase the flash, program and verify it.

12. Once the flash programming is completed, open the **verify\_result.txt** file in the C:\Flash\_Utils folder. If the flash programming was successful, you should see the following line in the verify\_result.txt file:

-> **Total byte mismatches [0]**

If there is anything other than this line in the verify\_result.txt file, the flash programming was NOT successful. Check the following jumper settings:

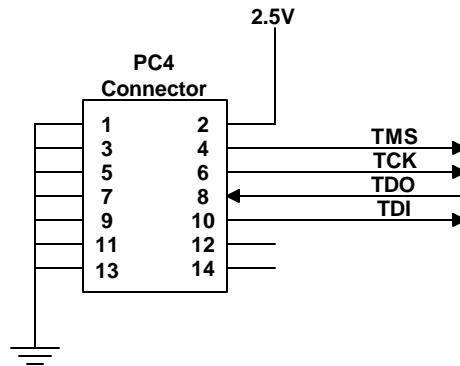
- Make sure JP9 jumpers are un-installed.
- Make sure JP12 jumper is un-installed.

After checking these jumper settings go back to the step 11 and re-program the flash.

- Upon completion of the serial flash programming, power down the board and **remove the PC4 cable from the Flash Programming header**.
- Set the mode jumpers to Master Serial (install all mode jumpers on JP9)
- Install a jumper on JP12.
- Power up the board and FPGA will configure.

### 3.12.4 JTAG Port (PC4)

The Virtex-4 MB development board provides a JTAG port (PC4 type) connector for configuration of the FPGA. The following figure shows the pin assignments for the PC4 header on this development board.



**Figure 19 – PC4 JTAG Port Connector**

### 3.12.5 Configuration Modes

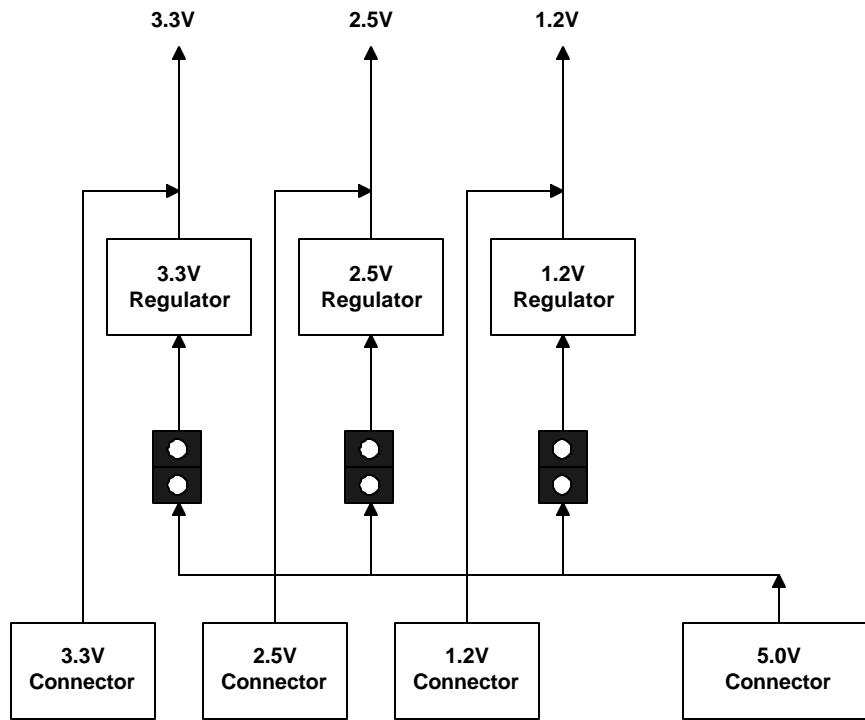
The following table shows the Virtex-4 configuration modes.

**Table 28 - FPGA Configuration Mode Jumper Settings**

Mode	PC Pull-up	Configuration Mode Jumpers			
		1-2 (M2)	3-4 (M1)	5-6 (M0)	7-8 (HSWAP_EN)
Master Serial	Yes	Closed	Closed	Closed	Closed
Master Serial	No	Closed	Closed	Closed	Open
Slave Serial	Yes	Open	Open	Open	Closed
Slave Serial	No	Open	Open	Open	Open
Master SelectMap	Yes	Closed	Open	Open	Closed
Master SelectMap	No	Closed	Open	Open	Open
Slave SelectMap	Yes	Open	Open	Closed	Closed
Slave SelectMap	No	Open	Open	Closed	Open
JTAG	Yes	Open	Closed	Open	Closed
JTAG	No	Open	Closed	Open	Open

### 3.13 Voltage Regulators

The following figure shows the voltage regulators that are used on Virtex-4 MB development board to provide various on-board voltage sources. As shown in the following figure, a connector is used to provide the main 5.0V voltage to the board. This voltage source is provided to all on-board regulators to generate the 1.2V, 2.5V, and 3.3V voltages.



**Figure 20 - Voltage Regulators**

The following table shows the power provided on the development board for the on-board voltage sources. A 32.5W power adapter (5V @ 6.5A) is used to provide power to the on-board regulators. The following table shows typical power usage on the Virtex-4 MB development board.

**Table 29 - Power**

Voltage	Current (A)	Power (W)	Comments
1.2V	1.5	1.8	FPGA Core voltage
2.5V	2.0	5.0	FPGA I/O voltage, P240 supply voltage
3.3V	3	9.9	FPGA I/O voltage, P240 supply voltage.
<b>Total Power</b>		<b>16.7</b>	

For the on-board digital voltages (1.2V, 2.5V, and 3.3V), if the current provided by the on-board regulator is not sufficient for some applications, the user can directly drive the voltage source and bypass the on-board regulators.

### 3.14 Bank I/O Voltage

The following table shows the Virtex-4 bank I/O voltages on the Virtex-4 MB development board.

**Table 30 – I/O Bank Voltages**

Bank #	I/O Voltage
0	2.5V
1	2.5V/3.3V
2	2.5V/3.3V
3	2.5V
4	2.5V
5	2.5V/3.3V
6	2.5V
7	2.5V/3.3V
8	2.5V
9	2.5V
10	3.3V

### 3.15 P240 Expansion Module Signal Assignments

The following tables show the Virtex-4 pin assignments to the P240 Expansion Module connectors (JX1 & JX2) located on the Virtex-4 MB development board.

**Table 31 – P240 Connector Pin Assignments**

Virtex-4 FPGA Pin #	I/O Connector Signal Name	JX1 Pin #		I/O Connector Signal Name	Virtex-4 FPGA Pin #
	5.0V	1	2	5.0V	
	5.0V	3	4	5.0V	
	5.0V	5	6	5.0V	
	5.0V	7	8	5.0V	
G17	LIO_SE_42	9	10	LIO_SE_43	H26
G18	LIO_SE_40	11	12	LIO_SE_41	H25
C23	LIO_SE_38	13	14	LIO_SE_39	G26
E22	LIO_SE_36	15	16	LIO_SE_37	G25
H23	LIO_SE_34	17	18	LIO_SE_35	F26
H24	LIO_SE_32	19	20	LIO_SE_33	E26
	3.3V	21	22	3.3V	
	3.3V	23	24	3.3V	
	3.3V	25	26	3.3V	

	<b>3.3V</b>	27	28	<b>3.3V</b>	
G23	LIO_SE_30	29	30	LIO_SE_31	E25
G24	LIO_SE_28	31	32	LIO_SE_29	D26
F23	LIO_SE_26	33	34	LIO_SE_27	D25
F24	LIO_SE_24	35	36	LIO_SE_25	C26
E23	LIO_SE_22	37	38	LIO_SE_23	C25
E24	LIO_SE_20	39	40	LIO_SE_21	B24
D23	LIO_SE_18	41	42	LIO_SE_19	A24
D24	LIO_SE_16	43	44	LIO_SE_17	B23
	<b>2.5V</b>	45	46	<b>2.5V</b>	
	<b>2.5V</b>	47	48	<b>2.5V</b>	
	<b>2.5V</b>	49	50	<b>2.5V</b>	
	<b>2.5V</b>	51	52	<b>2.5V</b>	
C24	LIO_SE_14	53	54	LIO_SE_15	A23
F16	LIO_SE_12	55	56	LIO_SE_13	F11
F15	LIO_SE_10	57	58	LIO_SE_11	C16
F12	LIO_SE_8	59	60	LIO_SE_9	D16
F13	LIO_SE_6	61	62	LIO_SE_7	D15
D13	LIO_SE_4	63	64	LIO_SE_5	D14
C11	LIO_SE_2	65	66	LIO_SE_3	E14
D11	LIO_SE_0	67	68	LIO_SE_1	F14
	<b>GND</b>	69	70	<b>GND</b>	
H22	LIO_LVDS_P14	71	72	LIO_LVDS_P15	C21
H21	LIO_LVDS_N14	73	74	LIO_LVDS_N15	B21
	<b>GND</b>	75	76	<b>GND</b>	
F20	LIO_LVDS_P12	77	78	LIO_LVDS_P13	F18
E20	LIO_LVDS_N12	79	80	LIO_LVDS_N13	E18
G19	LIO_LVDS_P10	81	82	LIO_LVDS_P11	E21
F19	LIO_LVDS_N10	83	84	LIO_LVDS_N11	D21
	<b>GND</b>	85	86	<b>GND</b>	
H20	LIO_LVDS_P8	87	88	LIO_LVDS_P9	D20
G20	LIO_LVDS_N8	89	90	LIO_LVDS_N9	D19
	<b>GND</b>	91	92	<b>GND</b>	
D22	LIO_LVDS_P6	93	94	LIO_LVDS_P7	E17
C22	LIO_LVDS_N6	95	96	LIO_LVDS_N7	F17
	<b>GND</b>	97	98	<b>GND</b>	
C20	LIO_LVDS_P4	99	100	LIO_LVDS_P5	A22
B20	LIO_LVDS_N4	101	102	LIO_LVDS_N5	A21
	<b>GND</b>	103	104	<b>GND</b>	
C19	LIO_LVDS_P2	105	106	LIO_LVDS_P3	A20
D18	LIO_LVDS_N2	107	108	LIO_LVDS_N3	A19
	<b>GND</b>	109	110	<b>GND</b>	
C17	LIO_LVDS_P0	111	112	LIO_LVDS_P1	B18
D17	LIO_LVDS_N0	113	114	LIO_LVDS_N1	A18
	<b>GND</b>	115	116	<b>GND</b>	
B17	LIO_CLKIN_P	117	118	LIO_CLKIN_1	E13
A17	LIO_CLKIN_N	119	120	LIO_CLKIN_0	D12
	<b>GND</b>	121	122	<b>GND</b>	
	<b>GND</b>	123	124	<b>GND</b>	
	<b>GND</b>	125	126	<b>GND</b>	

	<b>GND</b>	127	128	<b>GND</b>	
	<b>GND</b>	129	130	<b>GND</b>	
	<b>GND</b>	131	132	<b>GND</b>	

**Table 32– P240 Connector Pin Assignments**

Virtex-4 FPGA Pin #	I/O Connector Signal Name	JX2 Pin #		I/O Connector Signal Name	Virtex-4 FPGA Pin #
	<b>5.0V</b>	1	2	<b>5.0V</b>	
	<b>5.0V</b>	3	4	<b>5.0V</b>	
	<b>5.0V</b>	5	6	<b>5.0V</b>	
	<b>5.0V</b>	7	8	<b>5.0V</b>	
W26	RIO_SE_42	9	10	RIO_SE_43	W23
W25	RIO_SE_40	11	12	RIO_SE_41	W24
Y26	RIO_SE_38	13	14	RIO_SE_39	Y23
Y25	RIO_SE_36	15	16	RIO_SE_37	Y24
AA26	RIO_SE_34	17	18	RIO_SE_35	AA23
AB26	RIO_SE_32	19	20	RIO_SE_33	AA24
	<b>3.3V</b>	21	22	<b>3.3V</b>	
	<b>3.3V</b>	23	24	<b>3.3V</b>	
	<b>3.3V</b>	25	26	<b>3.3V</b>	
	<b>3.3V</b>	27	28	<b>3.3V</b>	
AB25	RIO_SE_30	29	30	RIO_SE_31	AB23
AC26	RIO_SE_28	31	32	RIO_SE_29	AB24
AC25	RIO_SE_26	33	34	RIO_SE_27	AB21
V20	RIO_SE_24	35	36	RIO_SE_25	W22
AD26	RIO_SE_22	37	38	RIO_SE_23	Y22
W20	RIO_SE_20	39	40	RIO_SE_21	W21
AD25	RIO_SE_18	41	42	RIO_SE_19	AC21
AD19	RIO_SE_16	43	44	RIO_SE_17	AC19
	<b>2.5V</b>	45	46	<b>2.5V</b>	
	<b>2.5V</b>	47	48	<b>2.5V</b>	
	<b>2.5V</b>	49	50	<b>2.5V</b>	
	<b>2.5V</b>	51	52	<b>2.5V</b>	
AA16	RIO_SE_14	53	54	RIO_SE_15	AC16
AA15	RIO_SE_12	55	56	RIO_SE_13	AC15
AB14	RIO_SE_10	57	58	RIO_SE_11	AC14
AA14	RIO_SE_8	59	60	RIO_SE_9	AD14
AB13	RIO_SE_6	61	62	RIO_SE_7	AC13
AA13	RIO_SE_4	63	64	RIO_SE_5	AD13
AA12	RIO_SE_2	65	66	RIO_SE_3	AC12
AA11	RIO_SE_0	67	68	RIO_SE_1	AC11
	<b>GND</b>	69	70	<b>GND</b>	
Y20	RIO_LVDS_P14	71	72	RIO_LVDS_P15	V21
Y21	RIO_LVDS_N14	73	74	RIO_LVDS_N15	V22
	<b>GND</b>	75	76	<b>GND</b>	
AA19	RIO_LVDS_P12	77	78	RIO_LVDS_P13	AC23
AA20	RIO_LVDS_N12	79	80	RIO_LVDS_N13	AC24
AA18	RIO_LVDS_P10	81	82	RIO_LVDS_P11	AC22
Y18	RIO_LVDS_N10	83	84	RIO_LVDS_N11	AB22
	<b>GND</b>	85	86	<b>GND</b>	

---

AF24	RIO_LVDS_P8	87	88	RIO_LVDS_P9	AB20
AE24	RIO_LVDS_N8	89	90	RIO_LVDS_N9	AC20
	<b>GND</b>	91	92	<b>GND</b>	
AF23	RIO_LVDS_P6	93	94	RIO_LVDS_P7	AD22
AE23	RIO_LVDS_N6	95	96	RIO_LVDS_N7	AD23
	<b>GND</b>	97	98	<b>GND</b>	
AF21	RIO_LVDS_P4	99	100	RIO_LVDS_P5	AE21
AF22	RIO_LVDS_N4	101	102	RIO_LVDS_N5	AD21
	<b>GND</b>	103	104	<b>GND</b>	
AF19	RIO_LVDS_P2	105	106	RIO_LVDS_P3	AC18
AF20	RIO_LVDS_N2	107	108	RIO_LVDS_N3	AB18
	<b>GND</b>	109	110	<b>GND</b>	
AF18	RIO_LVDS_P0	111	112	RIO_LVDS_P1	Y19
AE18	RIO_LVDS_N0	113	114	RIO_LVDS_N1	W19
	<b>GND</b>	115	116	<b>GND</b>	
Y17	RIO_CLKOUT_P	117	118	RIO_CLKOUT_1	U7
AA17	RIO_CLKOUT_N	119	120	RIO_CLKOUT_0	U6
	<b>GND</b>	121	122	<b>GND</b>	
	<b>GND</b>	123	124	<b>GND</b>	
	<b>GND</b>	125	126	<b>GND</b>	
	<b>GND</b>	127	128	<b>GND</b>	
	<b>GND</b>	129	130	<b>GND</b>	
	<b>GND</b>	131	132	<b>GND</b>	

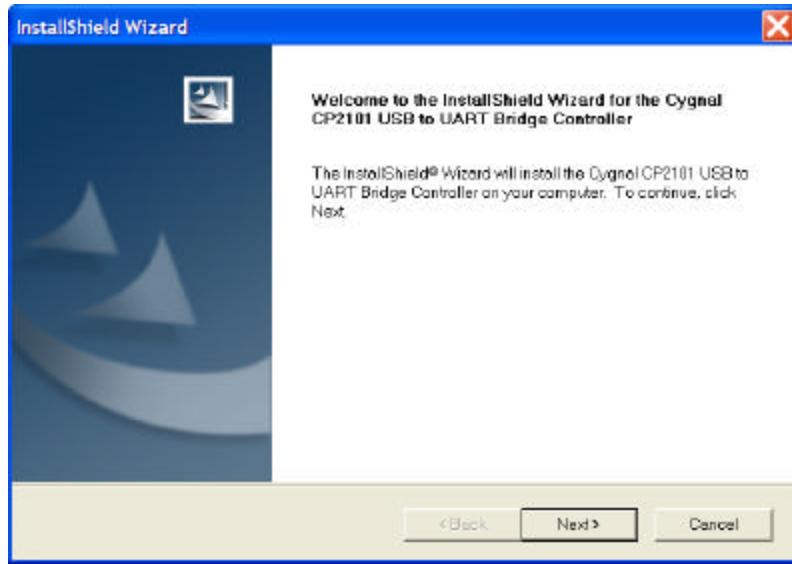
## 4 Revisions

V3.0 Initial release for Rev 3 board

Dec 20, 2005

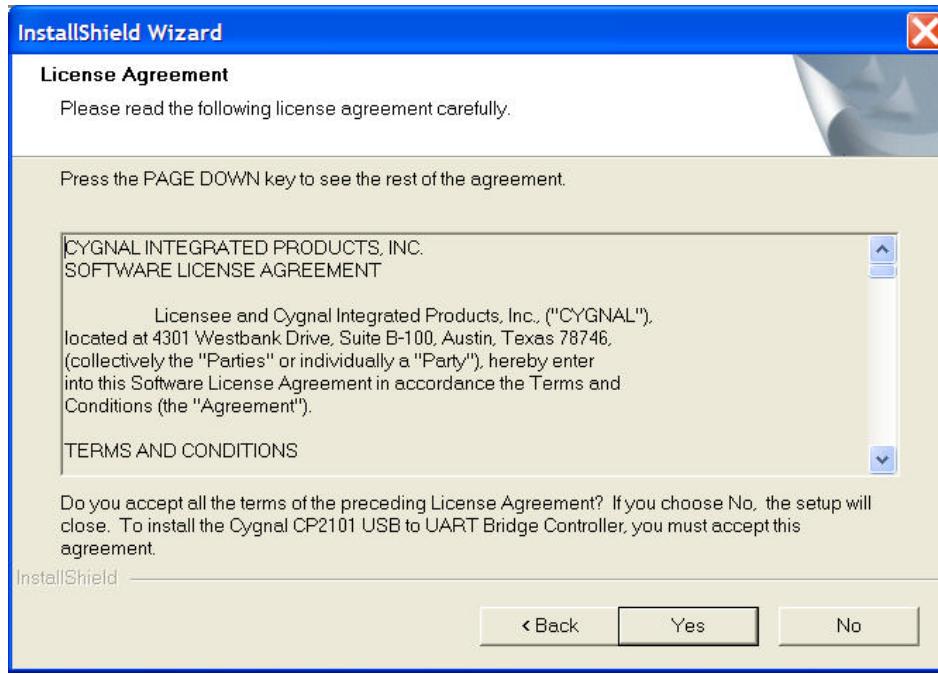
## Appendix A

1. Double-click CP2101\_Drivers.exe.



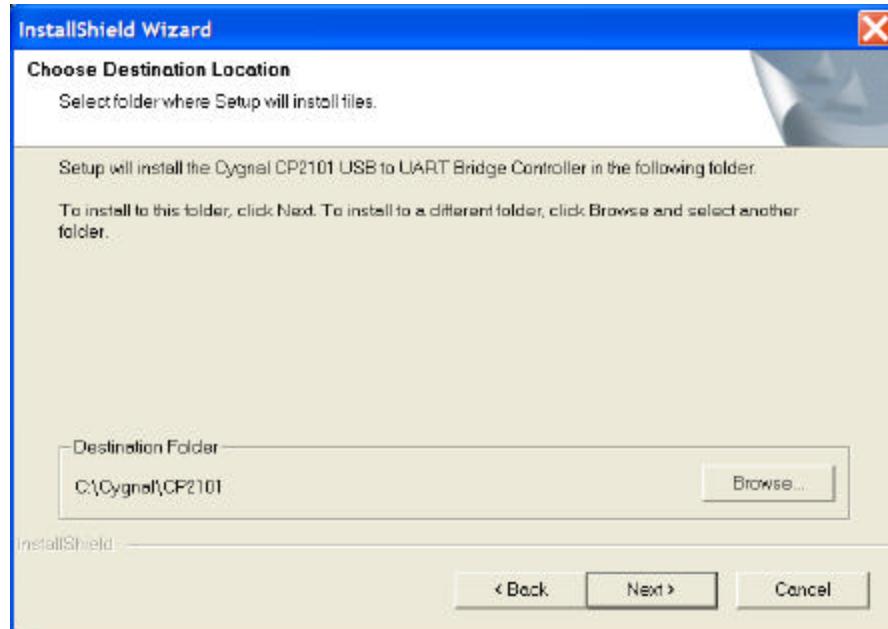
### Launching CP2101 Driver Installation

2. Click Next.
3. Read the license agreement and then click Yes.



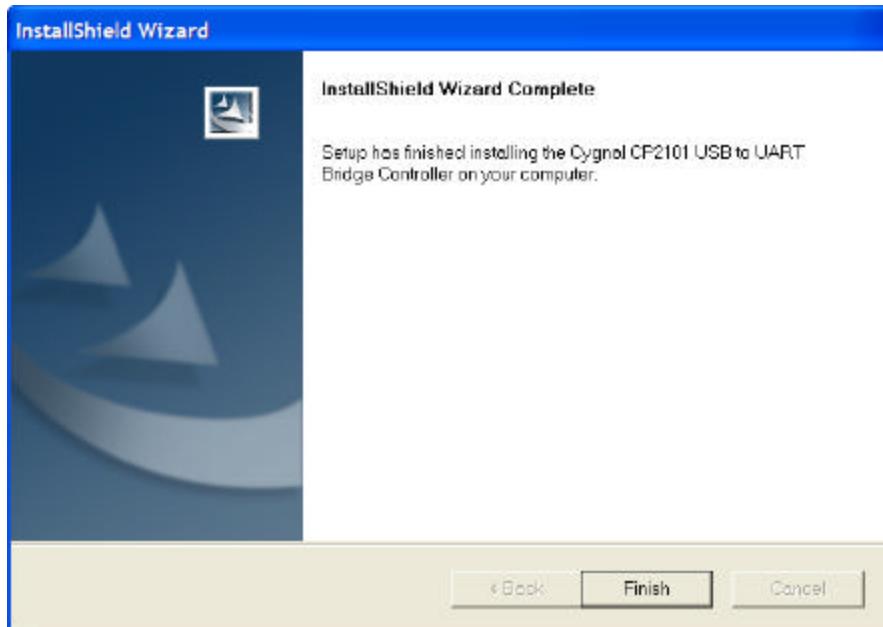
### Cygnal License Agreement

- 
4. Browse to an acceptable installation directory, and then click Next.



#### CP2101 Destination Location

5. The drivers are extracted to the selected directory. Click Finish once the extraction completes.



#### CP2101 Installation Successful

6. To finish the installation, plug the USB cable into the board and a USB port on the PC.

7. Turn the board power switch to the ON position.
8. The **Found New Hardware Wizard** launches. Click the radio button to **Install the software automatically (Recommended)** and then click Next.



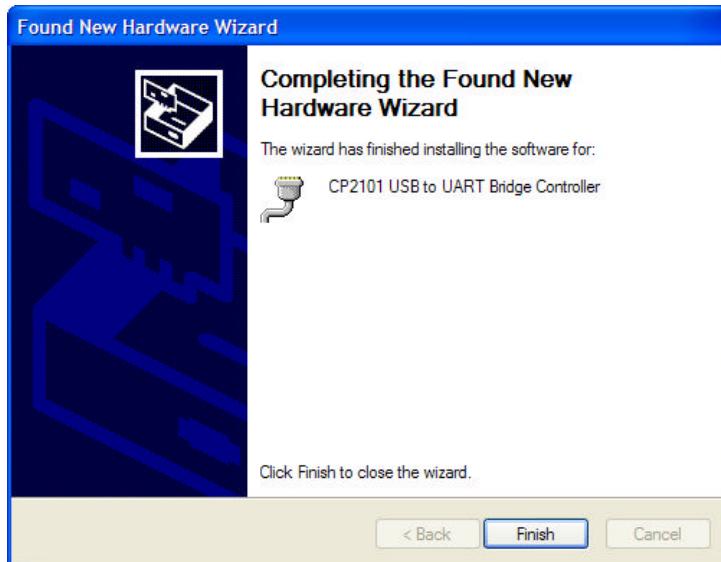
### Found New Hardware Wizard

9. The driver installation begins. If installing on WindowsXP, a warning is received stating that Windows Logo testing has not passed, as shown below. Click **Continue Anyway**.



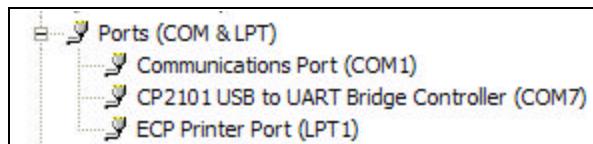
### Windows Logo Testing Not Passed

- 
10. The driver installation completes at this point. Click Finish in the **Found New Hardware Wizard**.



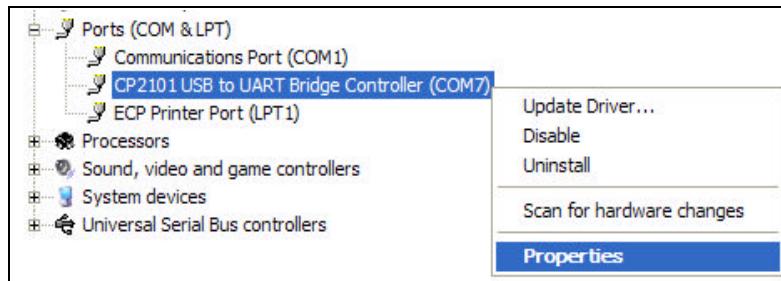
#### **CP2101 Driver Installation Complete**

11. Open the Device Manager (Control Panel → System → Hardware tab → Device Manager).
12. Under the **Ports** heading, a new device shows up, called **CP2101 USB to UART Bridge Controller**.



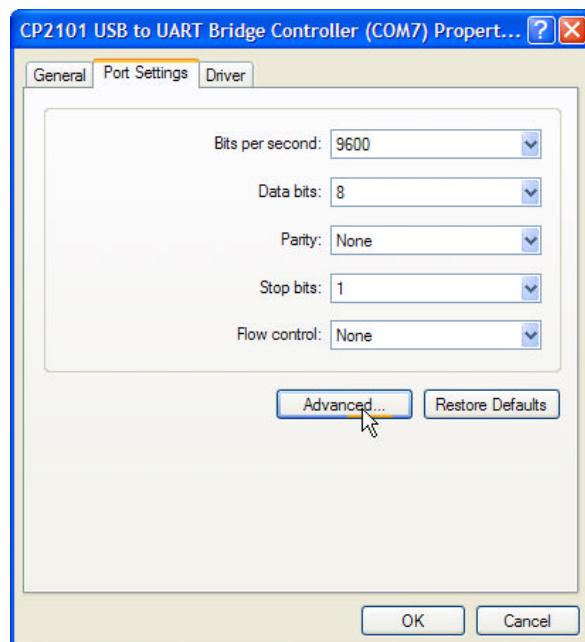
#### **CP2101 Recognized as COM Port**

13. If the CP2101 does not show up under ports, it may show up under "Other Devices" with a yellow exclamation mark. In this case, unplug the USB cable, run the setup manually (C:\Cygnal\CP2101\WIN\Setup.exe), and then plug the USB cable back in.
14. The O/S automatically assigns a COM Port number, typically between COM3 and COM7. For consistency, the COM number will be manually changed. Right click on **CP2101 USB to UART Bridge Controller** and select **Properties**.



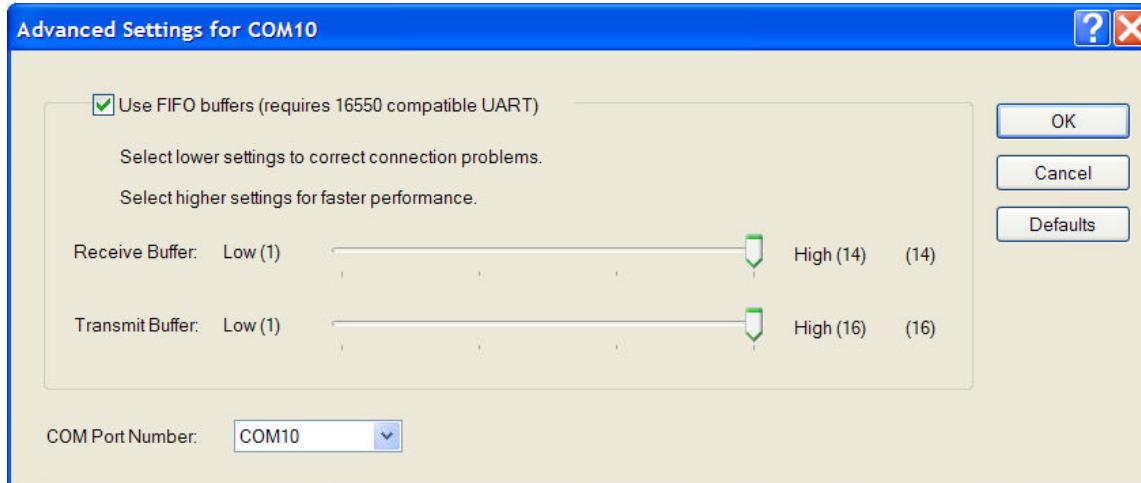
### COM Port Properties

15. Change to the **Port Settings** tab and select **Advanced**.



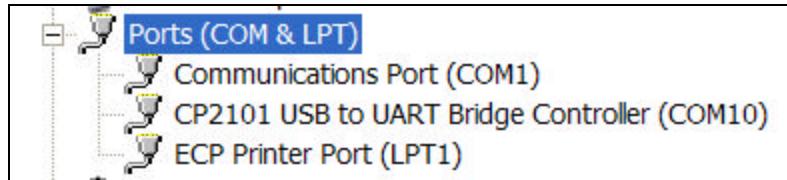
### Port Settings – Advanced

16. Select COM10 in the **COM Port Number** field, and then click OK twice.



### Changing the COM Port Number

17. Close the Device Manager, and then re-open it. Under **Ports**, the CP2101 USB to UART Bridge Controller is now assigned to COM10, as shown below.



CP2101 Assigned to COM10



# PS400

Patient Simulator

## Operators Manual

PN 2572345

March 2006

© 2006 Fluke Corporation, All rights reserved. Printed in USA  
All product names are trademarks of their respective companies.

## ***Warranty and Product Support***

Fluke Biomedical warrants this instrument against defects in materials and workmanship for one full year from the date of original purchase. During the warranty period, we will repair or, at our option, replace at no charge a product that proves to be defective, provided you return the product, shipping prepaid, to Fluke Biomedical. This warranty does not apply if the product has been damaged by accident or misuse or as the result of service or modification by other than Fluke Biomedical. IN NO EVENT SHALL FLUKE BIOMEDICAL BE LIABLE FOR CONSEQUENTIAL DAMAGES.

Only serialized products and their accessory items (those products and items bearing a distinct serial number tag) are covered under this one-year warranty. PHYSICAL DAMAGE CAUSED BY MISUSE OR PHYSICAL ABUSE IS NOT COVERED UNDER THE WARRANTY. Items such as cables and non-serialized modules are not covered under this warranty.

Recalibration of instruments is not covered under the warranty.

This warranty gives you specific legal rights, and you may also have other rights which vary from state to state, province to province, or country to country. This warranty is limited to repairing the instrument to Fluke Biomedical's specifications.

### ***Warranty Disclaimer***

Should you elect to have your instrument serviced and/or calibrated by someone other than Fluke Biomedical, please be advised that the original warranty covering your product becomes void when the tamper-resistant Quality Seal is removed or broken without proper factory authorization. We strongly recommend, therefore, that you send your instrument to Fluke Biomedical for factory service and calibration, especially during the original warranty period.

### ***Notices***

---

#### **All Rights Reserved**

© Copyright 2006, Fluke Biomedical. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language without the written permission of Fluke Biomedical.

---

#### **Copyright Release**

Fluke Biomedical agrees to a limited copyright release that allows you to reproduce manuals and other printed materials for use in service training programs and other technical publications. If you would like other reproductions or distributions, submit a written request to Fluke Biomedical.

## Unpacking and Inspection

---

Follow standard receiving practices upon receipt of the instrument. Check the shipping carton for damage. If damage is found, stop unpacking the instrument. Notify the carrier and ask for an agent to be present while the instrument is unpacked. There are no special unpacking instructions, but be careful not to damage the instrument when unpacking it. Inspect the instrument for physical damage such as bent or broken parts, dents, or scratches.

## Technical Support

---

For application support or answers to technical questions, either email [techservices@flukebiomedical.com](mailto:techservices@flukebiomedical.com) or call 1-800- 648-7952 or 1-425-446-6945.

## Claims

---

Our routine method of shipment is via common carrier, FOB origin. Upon delivery, if physical damage is found, retain all packing materials in their original condition and contact the carrier immediately to file a claim. If the instrument is delivered in good physical condition but does not operate within specifications, or if there are any other problems not caused by shipping damage, please contact Fluke Biomedical or your local sales representative.

## Standard Terms and Conditions

---

### Refunds and Credits

Please note that only serialized products and their accessory items (i.e., products and items bearing a distinct serial number tag) are eligible for partial refund and/or credit. Nonserialized parts and accessory items (e.g., cables, carrying cases, auxiliary modules, etc.) are not eligible for return or refund. Only products returned within 90 days from the date of original purchase are eligible for refund/credit. In order to receive a partial refund/credit of a product purchase price on a serialized product, the product must not have been damaged by the customer or by the carrier chosen by the customer to return the goods, and the product must be returned complete (meaning with all manuals, cables, accessories, etc.) and in "as new" and resalable condition. Products not returned within 90 days of purchase, or products which are not in "as new" and resalable condition, are not eligible for credit return and will be returned to the customer. The Return Procedure (see below) must be followed to assure prompt refund/credit.

### Restocking Charges

Products returned within 30 days of original purchase are subject to a minimum restocking fee of 15 %. Products returned in excess of 30 days after purchase, but prior to 90 days, are subject to a minimum restocking fee of 20 %. Additional charges for damage and/or missing parts and accessories will be applied to all returns.

### Return Procedure

All items being returned (including all warranty-claim shipments) must be sent freight-prepaid to our factory location. When you return an instrument to Fluke Biomedical, we recommend using United Parcel Service, Federal Express, or Air Parcel Post. We also recommend that you insure your shipment for its actual replacement cost. Fluke Biomedical will not be responsible for lost shipments or instruments that are received in damaged condition due to improper packaging or handling.

Use the original carton and packaging material for shipment. If they are not available, we recommend the following guide for repackaging:

- Use a double-walled carton of sufficient strength for the weight being shipped.
- Use heavy paper or cardboard to protect all instrument surfaces. Use nonabrasive material around all projecting parts.
- Use at least four inches of tightly packed, industry-approved, shock-absorbent material around the instrument.

### Returns for partial refund/credit:

Every product returned for refund/credit must be accompanied by a Return Material Authorization (RMA) number, obtained from our Order Entry Group at 1-800-648-7952 or 1-425-446-6945.

**Repair and calibration:**

To find the nearest service center, goto [www.flukebiomedical.com/service](http://www.flukebiomedical.com/service) or

In the U.S.A.:

Cleveland Calibration Lab  
Tel: 1-800-850-4606  
Email: [globalcal@flukebiomedical.com](mailto:globalcal@flukebiomedical.com)

Everett Calibration Lab  
Tel: 1-888-99 FLUKE (1-888-993-5853)  
Email: [service.status@fluke.com](mailto:service.status@fluke.com)

In Europe, Middle East, and Africa:

Eindhoven Calibration Lab  
Tel: +31-402-675300  
Email: [ServiceDesk@fluke.com](mailto:ServiceDesk@fluke.com)

In Asia:

Everett Calibration Lab  
Tel: +425-446-6945  
Email: [service.international@fluke.com](mailto:service.international@fluke.com)

---

## Certification

This instrument was thoroughly tested and inspected. It was found to meet Fluke Biomedical's manufacturing specifications when it was shipped from the factory. Calibration measurements are traceable to the National Institute of Standards and Technology (NIST). Devices for which there are no NIST calibration standards are measured against in-house performance standards using accepted test procedures.

---

## WARNING

Unauthorized user modifications or application beyond the published specifications may result in electrical shock hazards or improper operation. Fluke Biomedical will not be responsible for any injuries sustained due to unauthorized equipment modifications.

---

## Restrictions and Liabilities

Information in this document is subject to change and does not represent a commitment by Fluke Biomedical. Changes made to the information in this document will be incorporated in new editions of the publication. No responsibility is assumed by Fluke Biomedical for the use or reliability of software or equipment that is not supplied by Fluke Biomedical, or by its affiliated dealers.

---

## Manufacturing Location

The PS400 Patient Simulator is manufactured in Everett, WA, U.S.A.

## ***Table of Contents***

<b>Title</b>	<b>Page</b>
Introduction .....	1
Safety Information .....	1
General Specifications .....	2
Instrument Specifications.....	2
Accessories .....	3
Instrument Familiarity .....	4
Preparation for Use .....	5
Operating the Simulator.....	6
Power-Up .....	6
Selection Waveforms.....	7
Menu 1 Selections.....	7
Menu 2 Selections.....	8
Amplitude Selection.....	9
Waveform Examples.....	9
Menu 1 Waveforms.....	9
Menu 2 Waveforms.....	14

## ***List of Tables***

<b>Table</b>	<b>Title</b>	<b>Page</b>
1.	Menu 1 Selections.....	7
2.	Menu 2 Selections.....	8

## ***List of Figures***

<b>Figure</b>	<b>Title</b>	<b>Page</b>
1.	PS400 Controls and Indicators.....	4
2.	Installing the Battery.....	6

# **PS400 Patient Simulator**

## ***Introduction***

The PS400 Patient Simulator (hereafter “the Simulator”) is a high-performance ECG/arrhythmia simulator that generates all the waveforms necessary to check the frequency response, linearity, amplitude accuracy, and beats-per-minute measurement accuracy of an electrocardiograph. In addition, the Simulator generates arrhythmias.

The Simulator has a complete 12-lead output and a single high-level output.

The Simulator is for biomedical equipment technicians or service representatives to test the basic operation of physiological monitoring and diagnostic equipment. These tests are typically conducted during routine or scheduled preventive maintenance inspections.

This instrument can also verify calibration to the precision stated in Simulator Instrument Specifications. For the recommended inspection protocol and the required level of precision, refer to the calibration manual for the actual equipment under test. The Simulator is not intended to be used as the primary signal source for the initial validation of patient-related equipment.

## ***Safety Information***

In this manual, a **⚠⚠Warning** identifies conditions and actions that pose hazards to the user. A **⚠Caution** identifies conditions and actions that may damage the Simulator or the equipment under test. Do not proceed beyond a warning or caution until the indicated conditions are fully understood and met.

To ensure safe operation of the Simulator, observe all instructions and warnings contained in this manual.

## General Specifications

### Power Requirements

Battery .....	9 V Alkaline
Battery Life .....	200 hours
Line Power .....	Battery Eliminator (115 or 230 V to 7.7 Vdc @ 100 mA unregulated Connector: 2.5 mm center (+))

### Temperature Performance

Operating.....	15 °C to 35 °C (59 °F to 95 °F)
Storage .....	0 °C to 55 °C (32 °F to 131 °F)
Weight.....	0.41 kg (0.9 lb)
Size .....	3.56 cm H x 13.21 cm L x 9.91 cm W (1.4" H x 5.2" L x 3.9" W)

## Instrument Specifications

### Waveforms:

ECG waveforms .....	30, 60, 120, 180, and 240 BPM
Square wave.....	2 Hz
Sinewave .....	10, 40, 50, 60, and 100 Hz

### Arrhythmias:

Atrial fibrillation
Second degree A-V block, type 1
Right bundle branch block
Premature atrial contraction
Premature ventricular contraction, early
Premature ventricular contraction, RonT
Multifocal PVCs
Bigeminy
Run of 5 PVCs
Ventricular tachycardia
Ventricular fibrillation
Paced

### Automated sequence:

Pulse .....	4 seconds
Sine wave .....	10, 40, 60, and 100 Hz
Triangle wave.....	2 Hz

### Rate accuracy:

#### Amplitudes:

Lead I .....	0.5, 1.0, 1.5, 2.0 mV
High level output .....	0.25, 0.50, 0.75, & 1.00 V
Accuracy .....	±2 %

2Hz square wave at 2 mV (all leads)

## **Accessories**

<b>Standard</b>	<b>Fluke Part#</b>
Operators Manual	2572345
Battery, 9 V Alkaline	614487
Battery Eliminator (90-264 Vac, 9V, IEC – 320/C6 inlet)	2183983
Power Cord (IEC – 320/C5 connector)	See Note
Carrying Case (Vinyl)	2248424

## **Optional**

Cable, High Level ECG (3.5 mm phone to BNC)	2200116
Calibration Manual	2577801

Note: Refer to the current Fluke Biomedical Price List for availability, part number and price.

## Instrument Familiarity

Figure 1 identifies the controls and connectors of the Simulator.

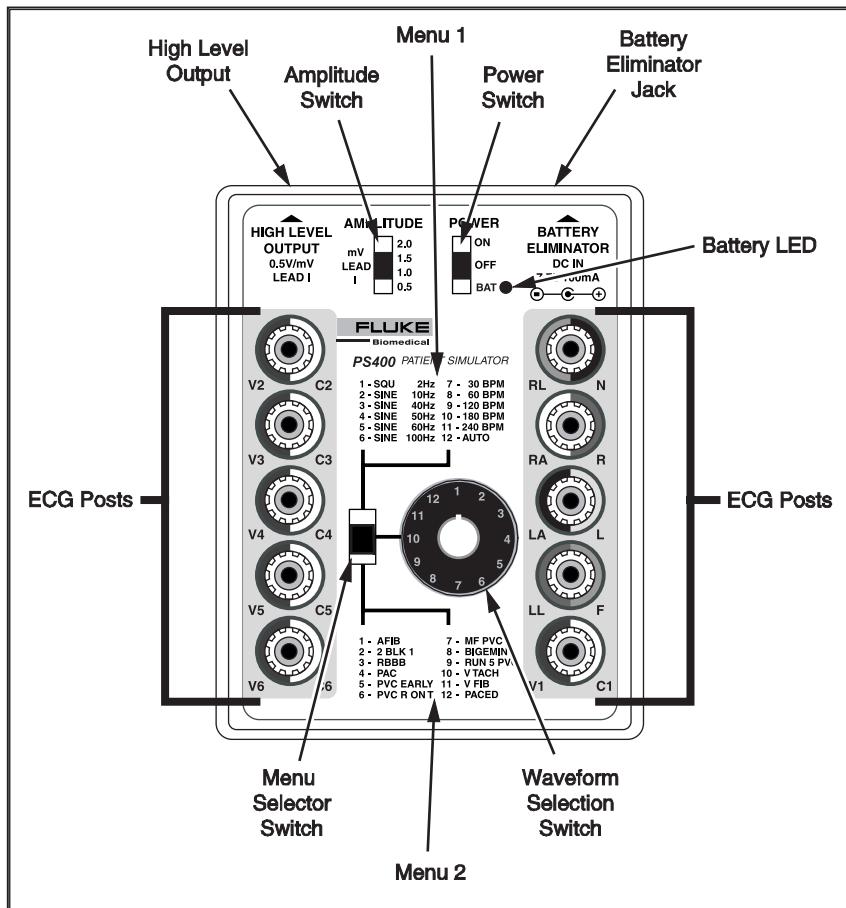


Figure 1. PS400 Controls and Indicators

ehd1.eps

## **Preparation for Use**

Use the 9 volt battery supplied by Fluke Biomedical or use the optional Battery Eliminator.

To install or replace the battery, locate the Battery Compartment at the bottom of the instrument case. (See the illustration on the next page.) Press down on the arrow and slide the Battery Compartment Cover off. Connect the 9-volt alkaline\* battery (Duracell MN1604 or equivalent) as illustrated in Figure 2. Make sure not to pinch the Battery Connector wires. Replace the cover.

*Note*

*Use only alkaline batteries.*

To check the battery, set the Power Switch to “BAT”. If the Battery LED lights, the battery is good. If not, the battery is dead.

*Note*

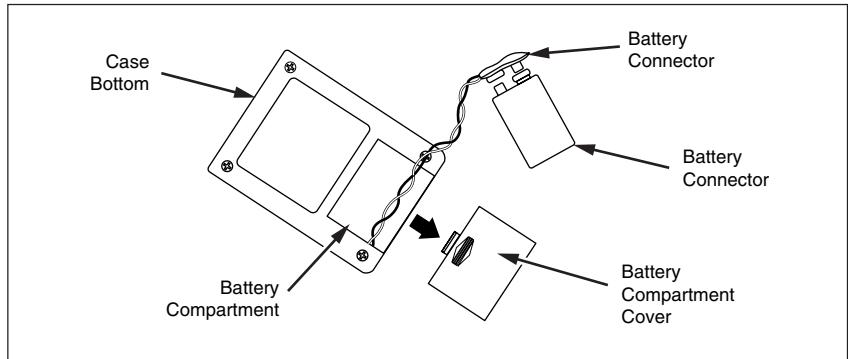
*Refer to the Instrument Familiarity section for the location of controls and indicators*

*Note*

*No special cables are required for using the ECG Posts.*

*Note*

*The High Level Output uses a 3.5-mm 2-conductor phone plug. Refer to Accessories, Optional in the Accessories section.*



ehd2.eps

**Figure 2. Installing the Battery**

## ***Operating the Simulator***

The following sections cover powering-Up the Simulator and how to select the various waveforms used in testing.

### ***Power-Up***

Confirm that the battery is installed (see the Preparation for Use section) or that the Battery Eliminator is firmly connected to the Battery Eliminator Jack, which is located on the top of the instrument.

#### *Note*

*For locations of PS400 controls and indicators, see the Instrument Familiarity section.*

The Power Switch has three positions: “ON”, “OFF”, and “BAT”.

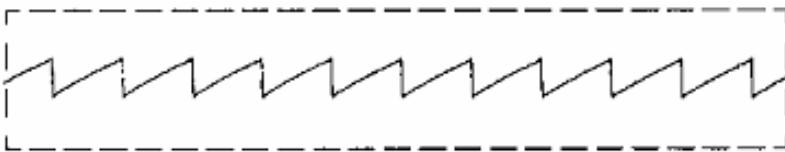
To power up the Simulator, push the Power Switch to the “ON” position.

#### *Note*

*The Simulator does not have a continuous power-on indicator (extending battery life). Turn the instrument off when not using it.*

To check the battery’s status, hold the Power Switch in the “BAT” position. The Battery LED illuminates if the battery is good. If the Battery LED does not light, replace the battery.

If the battery is low but not dead, the Simulator output waveform is a sawtooth, such as illustrated below, regardless of the waveform selected. Amplitude of the sawtooth waveform varies depending on how low the battery is.



ehd3.bmp

### **Selection Waveforms**

The Simulator is able to produce 24 different waveforms.

To select a waveform, select one of the two menus using the Menu Selector Switch, and turn the 12-position rotary Waveform Selector Switch to the number that corresponds to the desired waveform.

The waveforms are available on both the ECG Posts and the High Level Output.

## Menu 1 Selections

Setting the Menu Selector Switch to the *upper* position enables the following selections from Menu 1 as described in Table 1.

**Table 1. Menu 1 Selections**

Switch Position	Simulation
1 – SQU 2 Hz	Square waveform at 2 Hz
2 – Sine10 Hz	Square waveform at 10 Hz
3 – Sine40 Hz	Square waveform at 40 Hz
4 – Sine50 Hz	Square waveform at 50 Hz
5 – Sine60 Hz	Square waveform at 60 Hz
6 – Sine100 Hz	Square waveform at 100 Hz
7 – 30 BPM	ECG waveform at 30 BPM
8 – 60 BPM	ECG waveform at 60 BPM
9 – 120 BPM	ECG waveform at 120 BPM
10 – 180 BPM	ECG waveform at 180 BPM
11 – 240 BPM	ECG waveform at 240 BPM
12 – Auto	Automated sequence: each segment runs for 4 seconds in this order, a 4-second pulse, sine waves 10, 40, 60, and 100 Hz, and a 2-Hz triangle waveform.

## **Menu 2 Selections**

Setting the Menu Selection switch to the lower position enables the following selections from Menu 2 as described in Table 2.

**Table 2. Menu 2 Selections**

<b>Switch Postion</b>	<b>Simulation</b>
AFIB	Atrial Fibrillation.
2° BLK 1	Second degree A–V block, type 1, Wenckebach.
RBBB	Right bundle branch block.
PAC	Premature atrial contraction.
PVC	Premature ventricular contraction, early.
PVC R ON T	Premature ventricular contraction, R on T.
MF PVC	Multifocal PVCs.
BIGEMINY	Bigeminy
RUN 5 PVC	Run of 5 PVCs.
V TACH	Ventricular tachycardia.
V FIB	Ventricular fibrillation.
PACED	Paced

## **Amplitude Selection**

Use the 4-position Amplitude Switch to make the following selections (on Lead I):

- 0.5 mV
- 1.0 mV
- 1.5 mV
- 2.0 mV

## **Waveform Examples**

The Simulator waveform selections and their purposes are listed in this section.

The waveform examples illustrated were plotted using an ECG strip chart recorder set at 1 mV/cm sensitivity and 25 mm/s chart speed in the Lead I

configuration. The frequency response was set in the monitoring quality mode (-3 dB @ 30 Hz).

Because of inherent differences in frequency response, signal acquisition, and filtering techniques, the responses of the ECG equipment under test may vary from the waveforms illustrated.

The major differences are noticed between instruments intended for either monitoring or diagnostic applications. For example, the high-end frequency response (-3 dB) can range from 30 Hz for a bedside monitor to over 100 Hz for a 12-lead diagnostic recorder. Additionally, to attenuate noise on the ECG signal during recording, narrow “band reject” filters, common to many instruments, eliminate any response at the power system line frequencies of either 50 or 60 Hz.

Not all waveform selections are illustrated because of the wide range of responses possible.

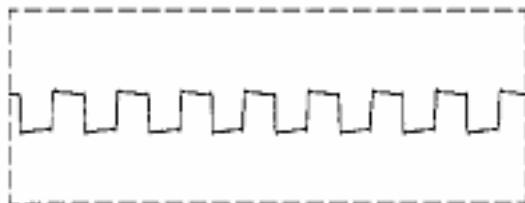
### ***Menu 1 Waveforms***

The following waveform selections are selectable through the Waveform Selection Switch when the Menu Selection switch is in the Menu 1 position.

## SQU 2 Hz

Waveform Selection Switch Position: 1

Purpose: Test amplitude accuracy (gain/damping).

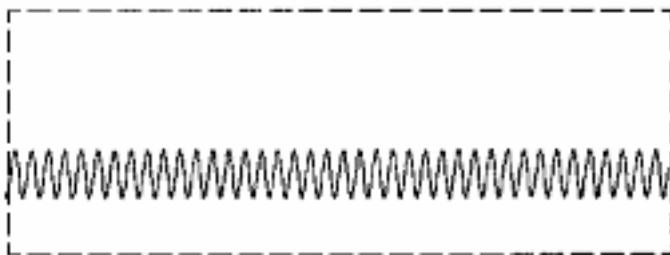


ehd4.bmp

## Sine 10 Hz

Waveform Selection Switch Position: 2

Purpose: Test bandpass response.



ehd5.bmp

## Sine 40 Hz

Waveform Selection Switch Position: 3

Purpose: Test low frequency response. (monitor -3 dB point).

*Note*

*Waveform example not shown because of wide range of responses possible.*

## Sine 50 Hz

Waveform Selection Switch Position: 4

Purpose: Test 50–Hz rejection (notch filter).

*Note*

*Waveform example not shown because of wide range of responses possible.*

## Sine 60 Hz

Waveform Selection Switch Position: 5

Purpose: Test 60–Hz rejection (notch filter).

*Note*

*Waveform example not shown because of wide range of responses possible.*

## Sine 100 Hz

Waveform Selection Switch Position: 6

Purpose: Test high-end frequency roll-off (diagnostic –3 dB point).

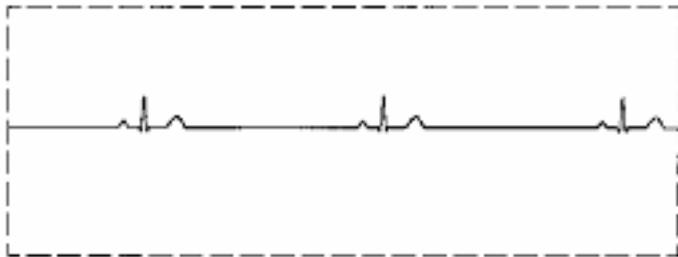
*Note*

*Waveform example not shown because of wide range of responses possible.*

## 30 BPM

Waveform Selection Switch Position: 7

Purpose: Test ECG rate accuracy.

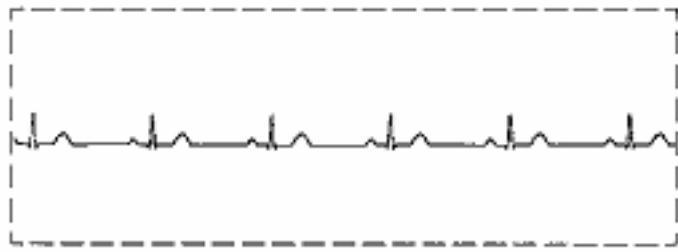


ehd6.bmp

## 60 BPM

Waveform Selection Switch Position: 8

Purpose: Test ECG rate accuracy.

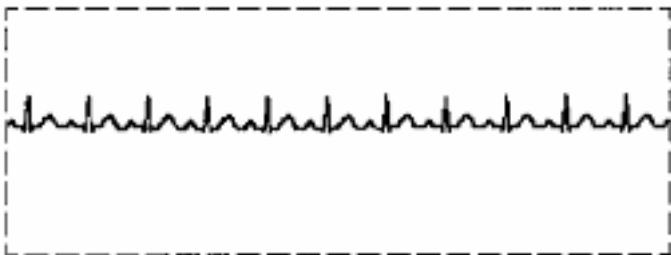


ehd7.bmp

## 120 BPM

Waveform Selection Switch Position: 9

Purpose: Test ECG rate accuracy.

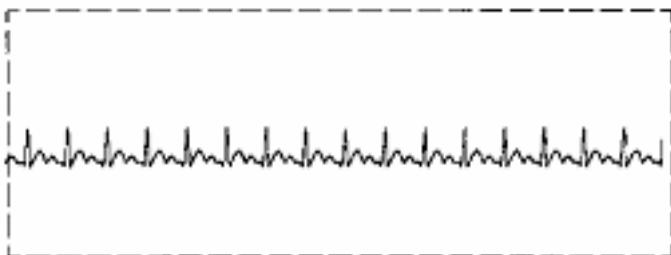


ehd8.bmp

## 180 BPM

Waveform Selection Switch Position: 10

Purpose: Test ECG rate accuracy.

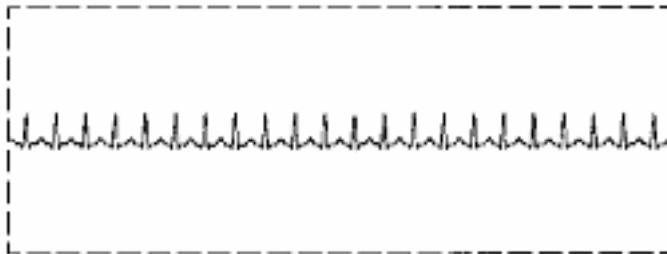


ehd9.bmp

## 240 BPM

Waveform Selection Switch Position: 11

Purpose: Test ECG rate accuracy.



ehd10.bmp

## AUTO

Waveform Selection Switch Position: 12

Purpose: Test monitor performance (gain/damping) linearity and frequency response.

This selection is an automated sequence of six segments that run for 4 seconds each, in this order: a 4-second pulse; sine waveforms 10, 40, 60, and 100 Hz; and a 2-Hz triangle waveform; and then the sequence repeats.

### *Note*

*Waveform example not shown because of wide range of responses possible.*

## *Menu 2 Waveforms*

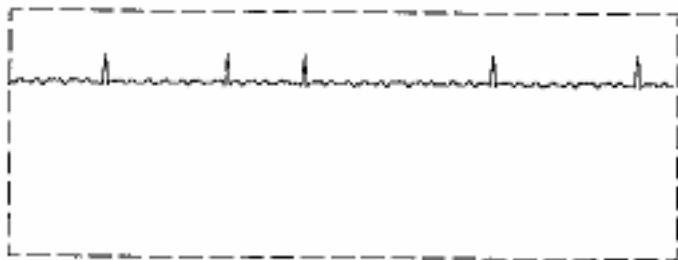
The following waveform selections are selectable through the Waveform Selection Switch when the Menu Selection switch is in the Menu 2 position.

## AFIB

Waveform Selection Switch Position: 1

Description: Atrial fibrillation.

Purpose: Test the ECG monitor's ability to display a particular arrhythmia, and as a training aid for health-care personnel.

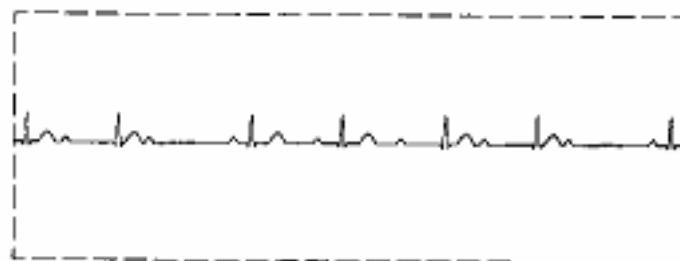


ehd11.bmp

## 2° BLK 1

Waveform Selection Switch Position: 2

Description: Second degree A-V block, type 1, Wenckebach.



ehd12.bmp

## R B B B

Waveform Selection Switch Position: 3

Description: Right bundle branch block.

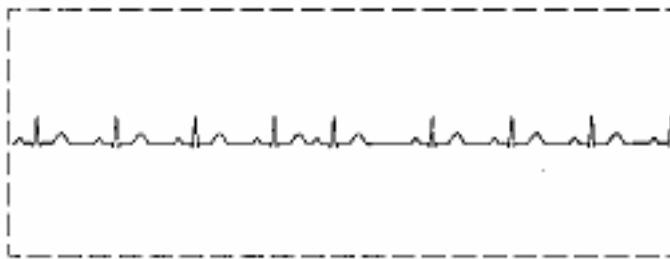


ehd13.bmp<sup>P</sup>

## PAC

Waveform Selection Switch Position: 4

Description: Premature atrial contraction.



ehd14.bmp

## PVC

Waveform Selection Switch Position: 5

Description: Premature ventricular contraction, early.

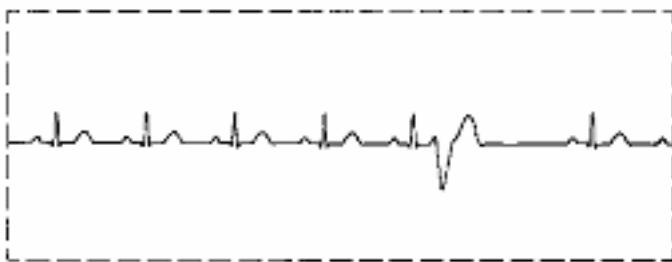


ehd15.bmp

## PVC R ON T

Waveform Selection Switch Position: 6

Description: Premature ventricular contraction, R on T – 65% premature;  
260 ms after previous normal R-wave peak.

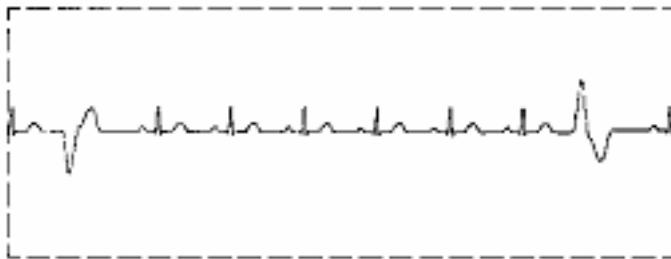


ehd16.bmp

## MF PVC

Waveform Selection Switch Position: 7

Description: Multi-focal premature ventricular contractions.



ehd17.bmp

## BIGEMINY

Waveform Selection Switch Position: 8

Description: Normal beat followed by a PVC, repeated.

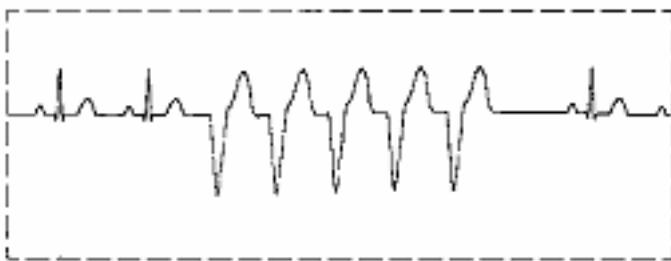


ehd18.bmp

## RUN 5 PVC

Waveform Selection Switch Position: 9

Description: Run of 5 PVCs and 36 normal beats, repeated.

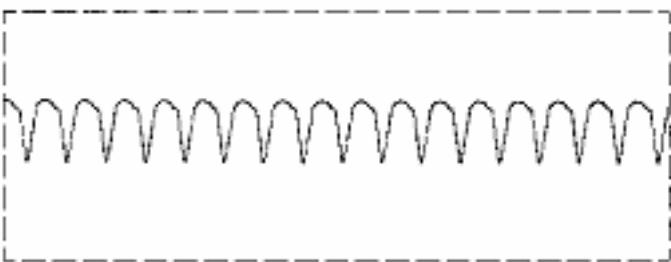


ehd19.bmp

## V TACH

Waveform Selection Switch Position: 10

Description: Ventricular tachycardia.

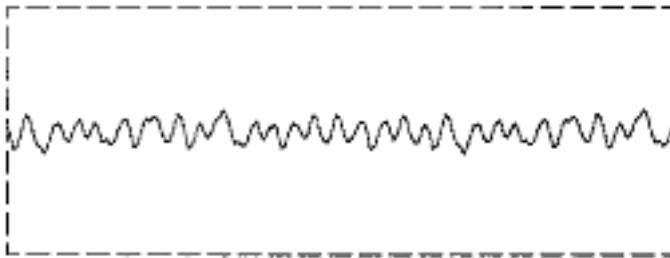


ehd20.bmp

## V FIB

Waveform Selection Switch Position: 11

Description: Ventricular fibrillation.

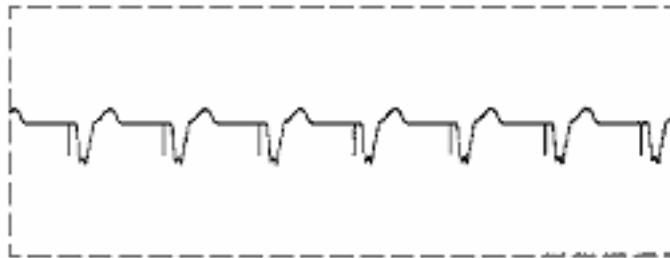


ehd21.bmp

## PACED

Waveform Selection Switch Position: 12

Description: Paced.



ehd22.bmp

*Note*

*The simulated pacer spike, preceding the ventricular response, may be filtered out by the instrument under test and replaced by an internally generated marker. Different instruments generate a wide range of markers to indicate that a spike has been sensed. Refer to the operation/service manual of the ECG equipment under test for specific information regarding the type of marker and pacer spike sensing level.*