

FPGA BASED RECONFIGURABLE BODY AREA NETWORK USING Nios II AND uClinux

A Thesis Submitted
to the College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in the Department of Electrical and Computer Engineering
University of Saskatchewan

by
Anthony Voykin

Saskatoon, Saskatchewan, Canada

© Copyright Anthony Voykin, April 2013. All rights reserved.

Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, it is agreed that the Libraries of this University may make it freely available for inspection. Permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professors who supervised this thesis work or, in their absence, by the Head of the Department of Electrical and Computer Engineering or the Dean of the College of Graduate Studies and Research at the University of Saskatchewan. Any copying, publication, or use of this thesis, or parts thereof, for financial gain without the written permission of the author is strictly prohibited. Proper recognition shall be given to the author and to the University of Saskatchewan in any scholarly use which may be made of any material in this thesis.

Request for permission to copy or to make any other use of material in this thesis in whole or in part should be addressed to:

Head of the Department of Electrical and Computer Engineering
57 Campus Drive
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5A9

Acknowledgments

I would like to thank my advisors Professor Ron Bolton and Professor Francis Bui for providing me with guidance and support necessary to complete my thesis work.

I would also like to thank my colleagues at SIAST for many interesting discussions about my work.

Finally, I would like express my sincere gratitude to my wife Chrissie and children, Abby and Josh, for their understanding and support while I pursued this degree.

Abstract

This research is focused on identifying an appropriate design for a reconfigurable Body Area Network (BAN).

In order to investigate the benefits and drawbacks of the proposed design, a BAN system prototype was built. This system consists of two distinct node types: a slave node and a master node. These nodes communicate using ZigBee radio transceivers. The microcontroller-based slave node acquires sensor data and transmits digitized samples to the master node. The master node is FPGA-based and runs uClinux on a soft-core microcontroller. The purpose of the master node is to receive, process and store digitized sensor data. In order to verify the operation of the BAN system prototype and demonstrate reconfigurability, a specific application was required.

Pattern recognition in electrocardiogram (ECG) data was the application used in this work and the MIT-BIH Arrhythmia Database was used as the known data source for verification. A custom test platform was designed and built for the purpose of injecting data from the MIT-BIH Arrhythmia Database into the BAN system.

The BAN system designed and built in this work demonstrates the ability to record raw ECG data, detect R-peaks, calculate and record R-R intervals, detect premature ventricular and atrial contractions. As this thesis will identify, many aspects of this BAN system were designed to be highly reconfigurable allowing it to be used for a wide range of BAN applications, in addition to pattern recognition of ECG data.

Table of Contents

Permission to Use	i
Acknowledgments	ii
Abstract	iii
Table of Contents	iv
List of Tables	ix
List of Figures	x
List of Abbreviations	xiv
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Thesis Organization	3
2 Electrocardiogram Background	4
2.1 Electromechanical Operation of the Human Heart	4
2.1.1 ECG Electrical Characteristics	6
2.1.2 Description of Fiducial Points	7
3 Body Area Networks	12
3.1 Introduction	12
3.2 Challenges	14
3.2.1 Network Architecture	14

3.2.2	Hardware Selection	16
3.2.3	Software Architecture	21
3.2.4	Wireless Protocol	24
4	System Architecture	29
4.1	Electrocardiogram Analog Front End	31
4.1.1	Block Diagram	31
4.1.2	Schematic	32
4.1.3	ECG AFE Testing	38
4.1.4	Design Considerations	39
4.2	Digital Transmitter	41
4.2.1	Block Diagram	42
4.2.2	Schematic	42
4.2.3	Software	46
4.2.4	Design Considerations	49
4.3	Field Programmable Gate Array Server	54
4.3.1	Block Diagram	55
4.3.2	DE0-Nano Daughter Board Schematic	56
4.3.3	System Configuration	60
4.3.4	User Software	66
4.3.5	User Software Drawbacks and Benefit	75
4.4	System Reconfigurability	76

4.4.1	FPGA Server User Software	76
4.4.2	Network Topology	78
4.4.3	Digital Transmitter Firmware	78
4.4.4	ECG Mote Sensor	79
4.4.5	FPGA Server Hardware	79
5	Test Results	81
5.1	MIT-BIH Arrhythmia Database	81
5.2	Test Platform	82
5.2.1	Block Diagram	83
5.2.2	Hardware	83
5.2.3	Software	84
5.2.4	Adjustment of the MIT-BIH Database Samples	88
5.2.5	Preparation of the Sample Buffer	89
5.3	Test Results	90
5.3.1	Acquired ECG Data	90
5.3.2	R-R Intervals	91
5.3.3	Interesting Event Detection	93
6	Summary and Contributions	95
6.1	Summary	95
6.2	Contributions	99
7	Future Work and Conclusions	100

7.1	Future Work	100
7.2	Conclusions	100
	References	103
A	FPGA Server System Development	112
A.1	Ubuntu Installation	112
A.2	uClinux Distribution	113
A.3	Quartus II Installation on Ubuntu	114
A.4	System Development with Quartus II and SOPC Builder	115
A.5	Building the uClinux Boot Image	123
A.6	Configuring the FPGA and Writing the Configuration to Flash Memory	126
A.7	Customizing the uClinux Boot Sequence	128
A.8	Setting up the System to Boot from EPCS Flash Memory	129
A.9	Developing User Software	129
B	FPGA Pin Assignments	131
C	XBee Configuration	133
D	Schematics	135
E	Bill of Materials	140
F	PCB Artwork	144
G	Software	148
G.1	MIT-BIH Parsing Engine	148
G.2	Test Platform C Code	149

G.3	FPGA Server Top Level Design Entity	157
G.4	FPGA Server User Software	159
G.5	Digital Transmitter Firmware	166
H	System Specifications	177
I	Truncation Noise	180

List of Tables

2.1	ECG Signal Characteristics	7
3.1	Comparison of Low Power Wireless Technologies	27
4.1	Required Bandwidth for ECG Systems	36
4.2	DE0-Nano Development and Education Board Hardware	56
4.3	List of Software Used to Develop BAN System	61
4.4	File Extensions Used in System Development	65
4.5	Data Supporting R-peak Identification Example	73
5.1	R-R Interval Test Data	93
E.1	DE0-Nano Daughter Board BOM	141
E.2	ECG AFE BOM	142
E.3	Digital Transmitter BOM	143

List of Figures

2.1	Human Heart, from [1]	4
2.2	Conduction System of the Heart, from [2]	5
2.3	Normal ECG Signal, from [3]	8
2.4	Illustration of an R-R Interval	8
2.5	Examples of a Sinus Rhythm and a PVC	9
2.6	PVC and Normal Sinus Rhythms	10
2.7	Frequency Spectrum of Sinus Rhythm and PVC	10
2.8	Illustration of R-R Interval in the Presence of a PAC	11
3.1	Mote block diagram	13
3.2	Typical BAN System Architecture	16
4.1	Photograph of System Nodes; a) ZigBee Mote; b) FPGA Server	29
4.2	ECG AFE Photo	31
4.3	ECG AFE Block Diagram	32
4.4	ECG AFE Schematic	33
4.5	HPF Frequency Response	34
4.6	LPF Frequency Response	35
4.7	ECG AFE Oscilloscope Graphic	38
4.8	Illustration of 60Hz Noise	40
4.9	Digital Transmitter Photo	41

4.10	Digital Transmitter Block Diagram	42
4.11	Digital Transmitter Schematic	43
4.12	Digital Transmitter Flow Chart - main program	47
4.13	Digital Transmitter Flow Chart - interrupt service routine	49
4.14	PIC18LF45K22 sleep period	51
4.15	ECG Mote Packet Design	52
4.16	Modular Checksum Example	53
4.17	FPGA Server photo	54
4.18	FPGA Server Block Diagram	55
4.19	DE0-Nano Daughter Board Schematic	58
4.20	FPGA Server Configuration Flow Diagram	63
4.21	FPGA Server User Software Flow Chart - main()	67
4.22	open_port() Flow Chart	68
4.23	start_of_packet() Flow Chart	69
4.24	get_packet() Flow Chart	70
4.25	combine() Flow Chart	71
4.26	Illustration of R-peak Detection	72
4.27	PVC and Normal Sinus Rhythms	74
4.28	Removal of Baseline Drift in $y[n]$	77
5.1	Typical MIT-BIH Arrhythmia Database Data file	82
5.2	Test System Block Diagram	84

5.3	PICDEM 2 PLUS DEMO BOARD	85
5.4	Test Platform Schematic	86
5.5	Test Platform Software Flow Chart	87
5.6	Baseline Generated Using Human Subject and ECG AFE	89
5.7	MIT-BIH Arrhythmia Database 213 Output From Test Platform . . .	90
5.8	MIT-BIH Arrhythmia Database vs. Data in Test System Software . .	91
5.9	MIT212 Acquired vs Raw Samples	92
5.10	MIT223 Interesting Events 1:00 to 1:20	94
A.1	SDRAM Memory Profile	118
A.2	SDRAM Timing	119
A.3	Nios II Configuration	120
A.4	PLL Configuration page 1	121
A.5	PLL Configuration page 6	122
A.6	PLL Configuration page 7	123
A.7	SPI (3 Wire Serial) Configuration	124
A.8	SOPC Builder System Configuration	125
A.9	nios2-configure-sof Output	127
A.10	sof2flash Output	127
A.11	nios2-flash-programmer Output	128
A.12	Booting uClinux	130
B.1	FPGA Server Pin Assignments	132

D.1	DE0-Nano Daughter Board Schematic	136
D.2	ECG AFE Schematic	137
D.3	Digital Transmitter Schematic	138
D.4	Test Platform Schematic	139
F.1	DE0-Nano Daughter Board	145
F.2	ECG AFE	146
F.3	Digital Transmitter	147

List of Abbreviations

ADC	Analog to Digital Converter
AES	Advanced Encryption Standard
AFE	Analog Front End
AP	Action Potential
APB	Arterial Premature Beat
APS	Application Support Sublayer
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
AT	Attention
ATM	Automated Teller Machine
ATV	Atrioventricular
BAN	Body Area Network
BLE	Bluetooth Low Energy
BOM	Bill of Materials
BPM	Beats Per Minute
BSN	Body Sensor Network
CMRR	Common Mode Rejection Ratio
COTS	Common Off The Shelf
CPLD	Complex Programmable Logic Device
DAC	Digital to Analog Converter
dB	Decibels
ECG	Electrocardiogram, Electrocardiography or Electrocardiograph
EDS	Embedded Design Suite
EEG	Electroencephalography
FHSS	Frequency Hopping Spread Spectrum
FPGA	Field Programmable Gate Array
FP	Floating Point
GCC	GNU Compiler Collection

GNU	GNU's Not Unix
GPIO	General Purpose Input Output
GPRS	General Packet Radio Service
GUI	Graphical User Interface
HDL	Hardware Description Language
HPF	High Pass Filter
HR	Heart Rate
HRV	Heart Rate Variability
IC	Integrated Circuit
ICSP	In Circuit Serial Programmer
IDE	Integrated Development Environment
INA	Instrumentation Amplifier
ISM	Industrial, Scientific and Medical
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LA	Left Atrium
LPF	Low Pass Filter
LR-PAN	Low Rate Personal Area Network
LV	Left Ventricle
MAC	Media Access Control
DMIPS	Dhrystone Million Instructions Per Second
MISO	Master In Slave Out
MLII	Modified Limb Lead II
MMU	Memory Management Unit
MOSI	Master Out Slave In
MSSP	Master Synchronous Serial Port
MSE	Mean Squared Error
OS	Operating System
PAC	Premature Atrial Contraction
PCB	Printed Circuit Board

PC	Personal Computer
PHY	Physical layer
PLL	Phase Locked Loop
PVC	Premature Ventricular Contraction
RA	Right Atrium
RISC	Reduced Instruction Set Computer
RF	Radio Frequency
RTL	Register Transfer Level
RTOS	Real Time Operating System
RV	Right Ventricle
SA	Sinoatrial
SD	Secure Digital
SIG	Special Interest Group
SNR	Signal to Noise Ratio
SRAM	Static Random Access Memory
SOPC	System on a Programmable Chip
SPI	Serial Peripheral Interface
TC	Trust Center
TQFP	Thin Quad Flat Pack
TR	Truncated Result
UART	Universal Asynchronous Receiver Transmitter
UWB	Ultra Wide Band
WBAN	Wireless Body Area Network
WBSN	Wireless Body Sensor Network
WiFi	Wireless Fidelity
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network

1. Introduction

Envision a world where adverse health conditions could be identified in realtime and relevant information could be communicated to physicians before health problems become deadly. The potential life saving possibilities are very exciting. This work strives to contribute to the foundation that could make this feasible.

1.1 Motivation

Nearly everyone experiences heart palpitations at some point in their life. A heart palpitation is an irregular heart beat, often described as a skipped beat. A patient experiencing frequent heart palpitations may plan a visit to his/her family doctor. This typically results in a routine test where the heart's electrical impulses are graphed and viewed by a doctor and sent to a specialist for further analysis. This is followed by several weeks, if not months, waiting for results. A likely scenario after receiving the test results could be additional specialized tests such as, wearing a Holter monitor for a day or more, an echocardiogram (i.e., a specialized ultrasound of the heart), or stress tests while monitoring the heart's electrical response. Specialized testing can take months to be scheduled due to lengthy wait lists and may be uncomfortable and/or inconvenient for the patient. It is plausible to suggest that over an extended period of time a heart condition that seemed relatively benign to the patient could escalate to become a serious health condition. While heart palpitations may not be considered life threatening, a patient with heart disease may be unaware of looming health problems until it's too late. The question is, with the appropriate application of technology could these scenarios and possibly many other serious health conditions

be avoided or proactively addressed?

A typical Body Area Network (BAN) locates sensors in or on the human body to collect vital sign and motion data and provide feedback to the user based on the measured data. A BAN designed to measure vital signs and report abnormalities to the user potentially has lifesaving possibilities. The possibility to save lives is certainly not the only benefit provided through the use of BANs. Athletes outfitted with sensors that gather motion, orientation and vital sign data, coupled with a tool that can process the data and provide real-time feedback would be a very valuable training asset. In another example, a patient undergoing physiotherapy could use the same sensors to provide therapists with the data required to assess recovery and proactively adjust therapeutic plans.

There are many applications in many different fields that will benefit from the use of BANs. Quality of life can be improved and lives can possibly be saved. This research is motivated by the potential impact it will have on human lives.

1.2 Objectives

BAN systems are currently being researched in many different applications, such as health care, athletics and entertainment. With each different application comes a distinct set of requirements. In order to accommodate a diverse field of applications, a highly reconfigurable BAN system is proposed. The primary objective of this research is to identify the most appropriate design for such a system. In light of the above challenges, the research objectives are:

1. Review the current state of BAN research through a literature review and identify opportunities to contribute to this research.
2. Determine the most appropriate wireless protocol and network, hardware and software architectures to use in a reconfigurable BAN.
3. Identify a suitable application to demonstrate a reconfigurable BAN system.

4. Verify the BAN system operation using a standardized data source.
5. Demonstrate reconfigurability of the final BAN system.

The chapters within this thesis will fully describe how each of these objectives is accomplished.

1.3 Thesis Organization

Following this introduction, this thesis continues in Chapter 2 with a description of the mechanics of the heart, the heart electrical system, the method used to record the electrical impulses from the heart and some of the abnormalities within these recordings. Chapter 3 contains a review of current research work in the field of BANs. Chapter 4 provides a detailed description of the BAN system realized through this research, including the details of system reconfigurability. Chapter 5 contains a detailed review of the test results. Chapter 6 summarizes this thesis and provides a listing of the contributions of this work. Chapter 7 concludes this thesis and suggests opportunities for future work.

2. Electrocardiogram Background

This chapter contains a discussion of the mechanics of the heart including the heart electrical system, a description of how the electrical impulses from the heart are captured and viewed as well as an explanation of interesting cardiac events that are pertinent to this research.

2.1 Electromechanical Operation of the Human Heart

The heart is a pump that is responsible for maintaining blood flow in the body. The heart contains four chambers: right atrium (RA), right ventricle (RV), left atrium (LA), left ventricle (LV) and a series of one way valves, as shown in Figure 2.1. These chambers and valves aid in the collection and distribution of blood throughout the body. In order to move blood throughout the body the heart muscle contracts, or

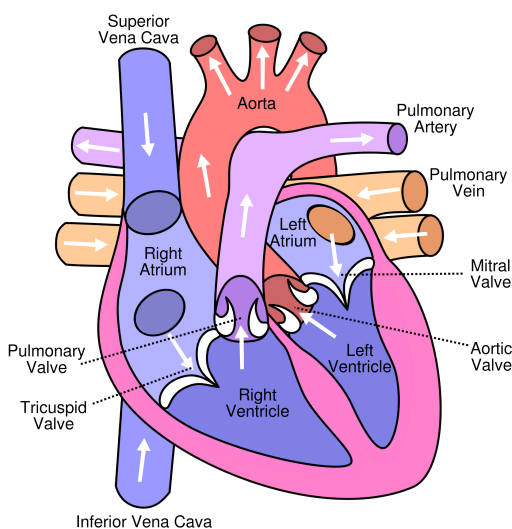


Figure 2.1 Human Heart, from [1]

beats. The atria contract first, followed by the ventricles. Prior to a heartbeat, deoxygenated blood enters the RA via the superior vena cava and the inferior vena cava. At the same time, oxygenated blood returns from the lungs via the pulmonary veins and enters the LA. The atria then contract expelling blood into the ventricles. After a brief delay, the ventricles also contract. When the ventricles contract the one-way valves between the atria and the ventricles close. The contraction of the RV forces the deoxygenated blood out of the heart to the lungs and the contraction of the LV forces oxygenated blood to the rest of the body via the aorta. The atria and ventricles are muscles and in order for these muscle to contract an electrical stimulus is required.

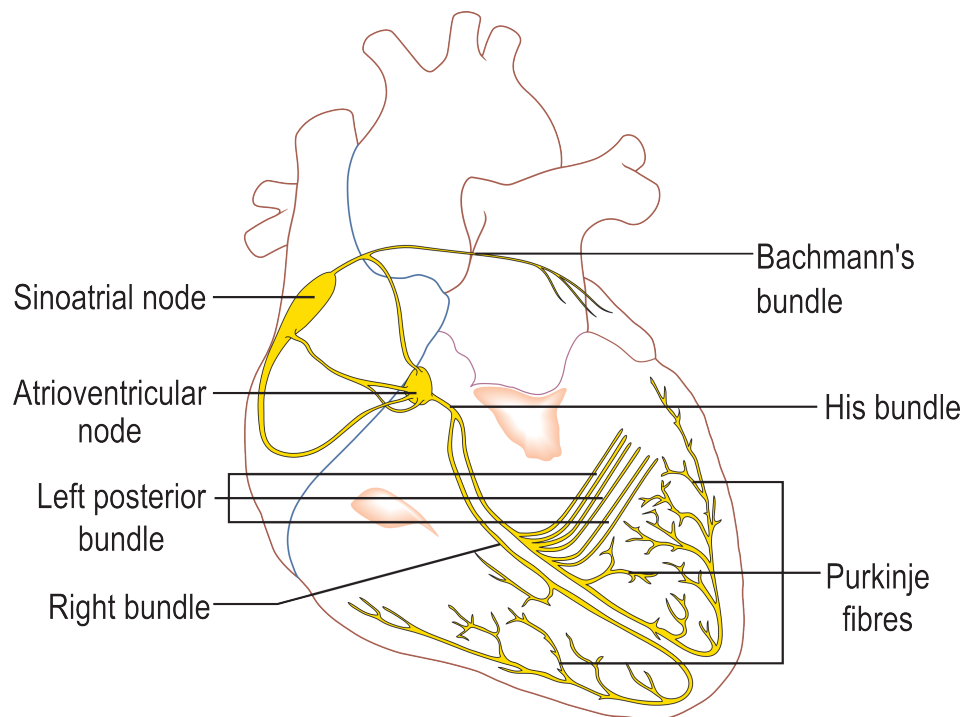


Figure 2.2 Conduction System of the Heart, from [2]

The sinoatrial (SA) node, shown in Figure 2.2 is located in the RA and is responsible for generating the electrical impulses that control the normal rhythm of the heart, as described by Boston Scientific [4]. An electrical impulse generated by the SA node cause the atria to contract. This electrical impulse then moves to the atrioventricular (AV) node through the wires in the heart, also known as the conduction

pathways. The AV node delays the electrical impulse slightly, allowing the ventricles to fill with blood. Following this delay the electrical impulse is distributed to the ventricles via the right and left bundle branches, which causes ventricular contraction. The movement of charge related to this electrical impulse can be captured with an electrocardiograph.

2.1.1 ECG Electrical Characteristics

An electrocardiograph (ECG) is a device that can measure the electrical characteristics of the heart. The device uses at least two electrodes attached to the human body. It measures the voltage difference between these electrodes and records the voltage over time to produce an electrocardiogram (ECG). Electrodes are used to connect the human body to an electrical circuit because the surface of the body is not highly conductive. The electrodes are typically attached to the torso near the heart and the differential voltage measured between them is referred to as a lead.

At this point, a recurring abbreviation needs to be clarified: ECG refers to electrocardiogram, electrocardiography and electrocardiograph. An electrocardiogram is both a graph of the change in differential voltage on the body surface vs. time, as a result of the electrical impulses created when the heart beats and it is also used to refer to the test that generates the graph. Electrocardiography refers to the study of electrocardiograms as well as the science of capturing the electrocardiogram. Finally, an electrocardiograph is the tool used to generate an electrocardiogram. The abbreviation ECG is used throughout this paper and the context that it is used will determine its meaning.

In order to generate an ECG, the signal parameters need to be clearly understood. According to Company-Bosch et. al. and Soundarapandian et. al. [5, 6], the ECG signal characteristics are summarized in Table 2.1. On the surface of the body, the AC signal from the heart is typically in the range of 0.5 to 5mV peak to peak. The frequency components lie between 0.05 and 150Hz. Added to the low level AC signal is a DC offset of up to $\pm 300\text{mV}$. The offset is due to the difference in impedance between

each electrode-to-body contact. Finally, a common mode voltage of up to 1.5V may be present. The common mode signal is primarily due to power line interference when coupled to an AC power source but may also be caused by Radio Frequency (RF) or microwave interference sources.

Table 2.1 ECG Signal Characteristics

Quantity	Value
AC Signal	0.5 to 5mV
DC Offset	$\pm 300\text{mV}$
Common Mode Voltage	up to 1.5V
Frequency Range	0.05 to 150Hz

In addition to these design constraints, the DC voltage between the electrodes can change due to changes in the electrode-skin impedance. This phenomenon is known as “baseline drift” and is generally caused by movement, changes in skin surface moisture and aging of electrodes [7]. If not addressed, baseline drift can saturate the output of an ECG Analog Front End, causing loss of valuable ECG information. When all of the design constraints have been accounted for the electrical impulses from the heart can be captured and analyzed.

2.1.2 Description of Fiducial Points

The ECG of a normal heart beat, also known as a normal sinus rhythm or sinus rhythm is shown in Figure 2.3. The QRS complex occurs due to the depolarization of the ventricles and is normally 60 to 100ms in duration according to Klabunde [8]. The highest peak in the QRS complex is referred to as the R-peak. An R-R interval, shown in Figure 2.4, is defined as the time between R-peaks. For example, a resting heart rate of 60 beats per minute equates to an R-R interval of 1 second. The inverse of the R-R interval is a measure of the instantaneous Heart Rate (HR), as shown in Equation (2.1)

$$\text{HR(Beats Per Minute)} = \frac{60 \text{ sec/min}}{\text{R-R interval(sec/beat)}}. \quad (2.1)$$

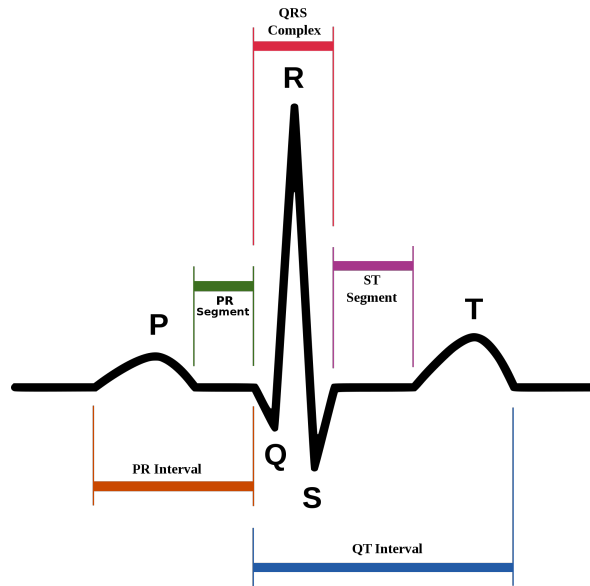


Figure 2.3 Normal ECG Signal, from [3]

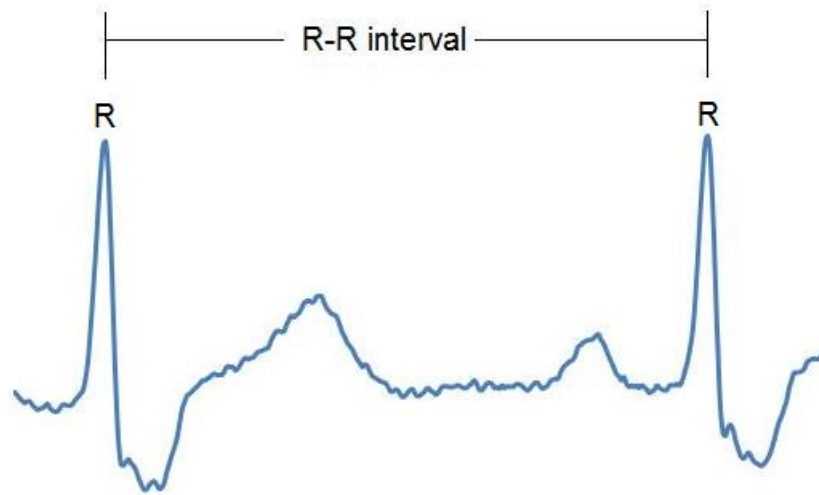


Figure 2.4 Illustration of an R-R Interval

A graph of many heart beats plotted over time is used to analyze heart health. Not all beats are normal sinus rhythm beats. An abnormal event that occurs in some individuals is called a premature ventricular contraction (PVC). A PVC occurs when the ventricles contract before the atria. An example of a PVC alongside a normal sinus rhythm is shown in Figure 2.5. The PVC is characterized by having a longer than usual QRS complex (i.e. greater than 120ms) an unusual shape, according to

Keany et al. [9]. In most cases PVCs are no indication of extenuating health problems. However in patients with existing heart disease PVCs should be carefully monitored. A trait of the PVC that is particularly useful in identifying it is that it doesn't contain the same high frequency components as a normal QRS complex.

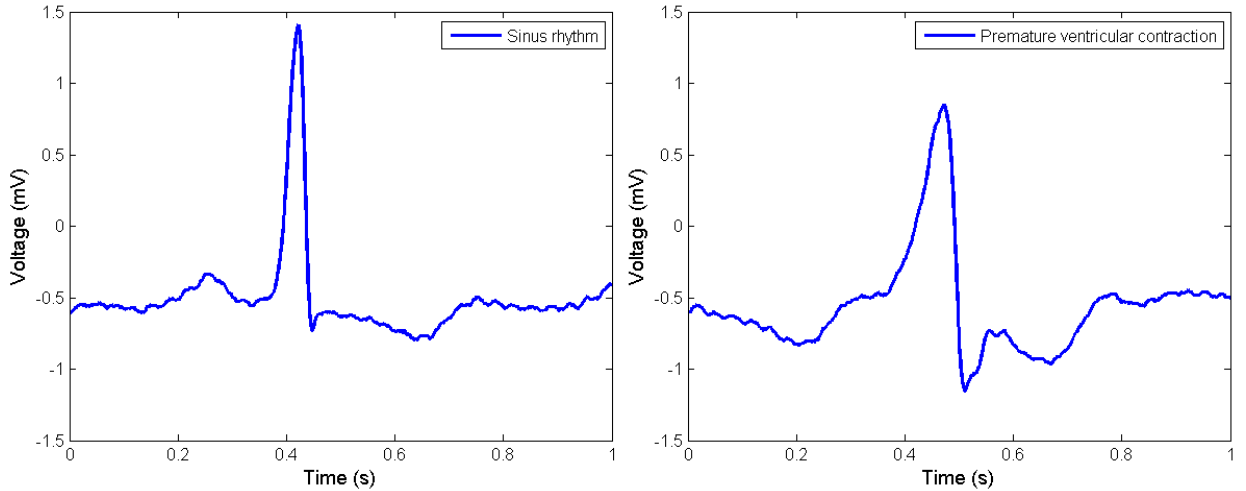


Figure 2.5 Examples of a Sinus Rhythm and a PVC

Signal power for a PVC event is concentrated below 10Hz while a normal QRS complex has more signal power in the higher frequencies, as shown in Figure 2.7. This trait can be used to distinguish normal sinus rhythms and PVCs. For example, the ratio of adjacent R-R intervals in an ECG containing normal sinus rhythms is approximately 1. If the peak of the PVC waveform is not classified as an R-peak the result is an R-R interval twice as long as the underlying sinus rhythm. The ratio of adjacent R-R intervals can then be used identify the occurrence of a PVC. This is done by filtering out the low frequency component of the ECG waveform. Figure 2.6 provides an illustration of two normal sinus rhythms surrounding a PVC, $x[n]$, as well as the filtered waveform, $y[n]$. Once the waveform is filtered a threshold can be applied to the level of the filtered waveform to detect normal sinus rhythms and ignore PVCs.

Another abnormal event that has symptoms similar to a PVC is called a premature atrial contraction (PAC). In a PAC event the heart beats prematurely with what

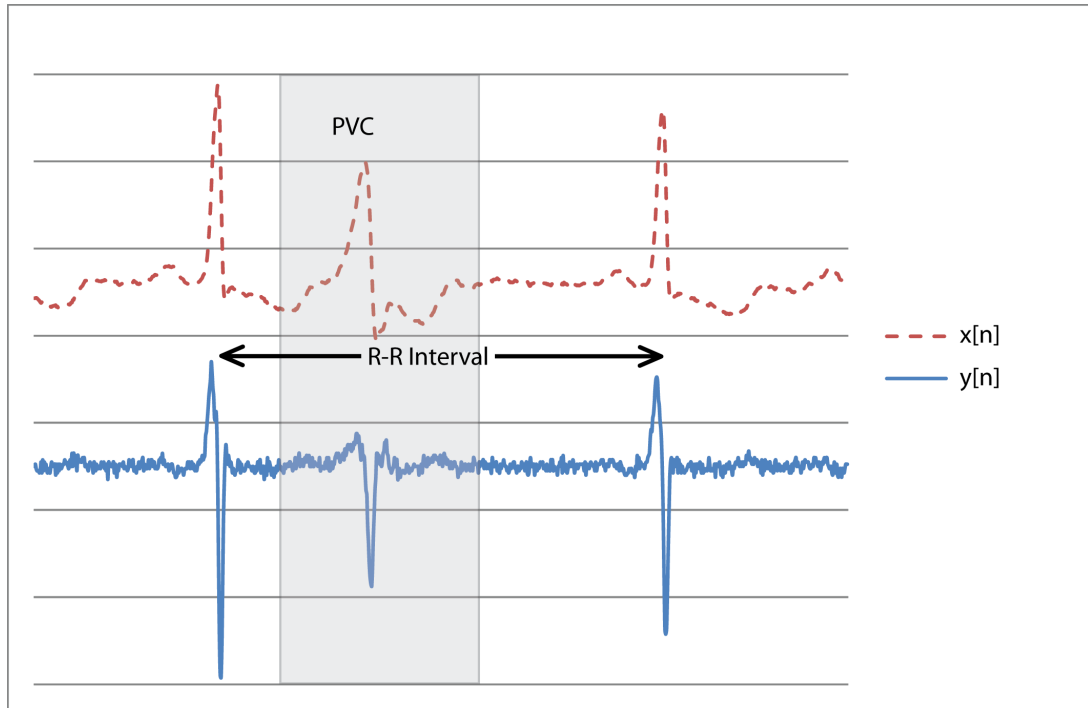


Figure 2.6 PVC and Normal Sinus Rhythms

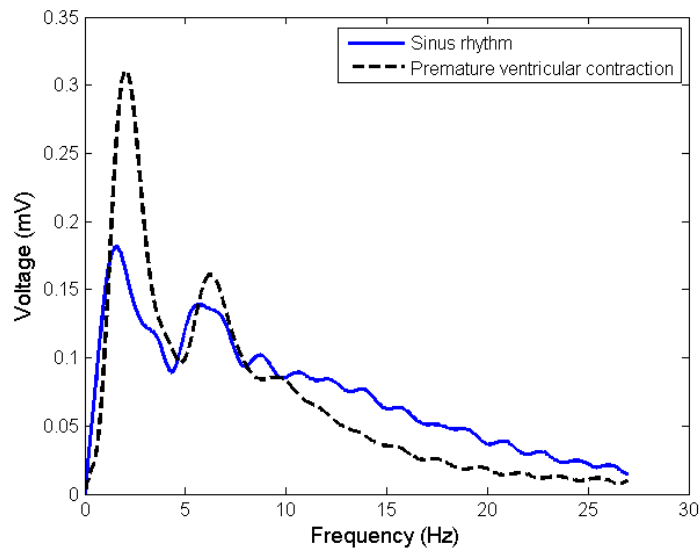


Figure 2.7 Frequency Spectrum of Sinus Rhythm and PVC

appears to be a normal sinus rhythm, as shown in Figure 2.8. Similar to identifying a PVC, PACs can be detected comparing adjacent R-R intervals and flagging significant differences. The R-R interval ratio in the presence of a PAC is greater than 1 but less than 2, as shown in Figure 2.8.

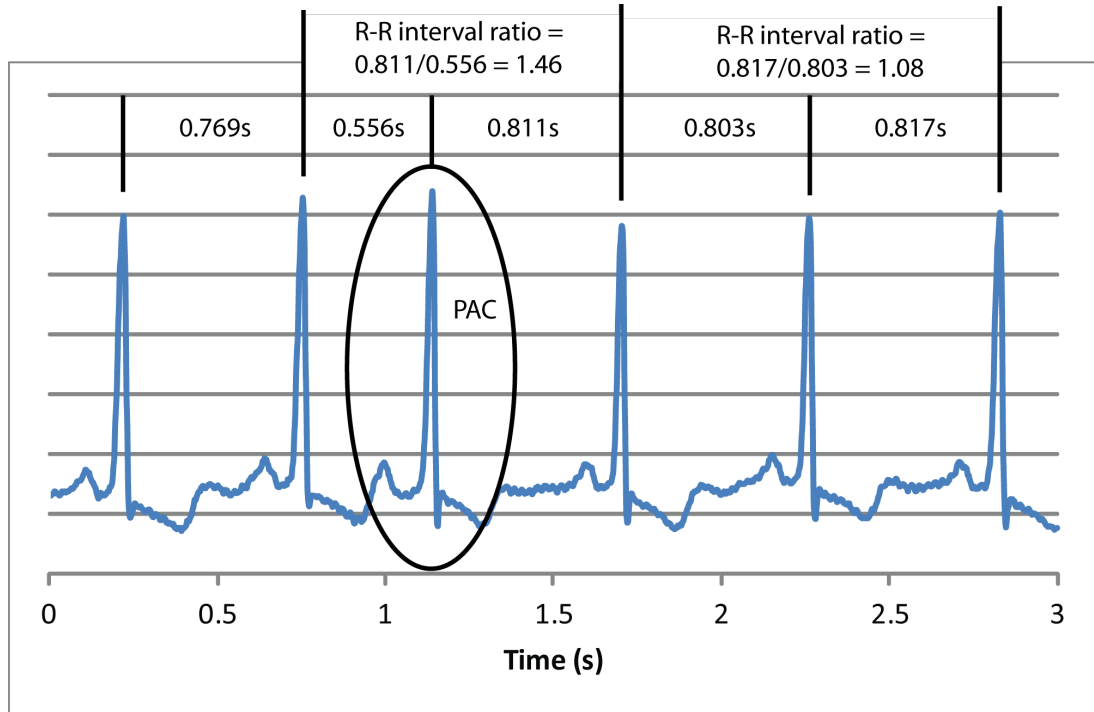


Figure 2.8 Illustration of R-R Interval in the Presence of a PAC

Identifying R-peaks, PVCs, PACs or other fiducial points and abnormal events can be done using a Body Area Network (BAN). A description of BANs, including the challenges associated with designing BANs is discussed in the following chapter.

3. Body Area Networks

This chapter contains a detailed description of a Body Area Network (BAN) and discusses the challenges that come with designing such a network. Based on the information presented, design choices regarding the BAN system designed in this work are also identified.

3.1 Introduction

BAN, Wireless BAN (WBAN), Body Sensor Network (BSN) and Wireless BSN (WBSN) are all terms used to describe sensor networks that monitor signals in and around the human body and communicate the acquired data to a master unit or server, which may be a Personal Computer (PC), smart phone or other small electronic device. This thesis will use the term BAN to encompass all types of body networks.

A typical BAN consists of two node types: slave nodes and master nodes. The slave nodes are also known as motes. Motes are responsible for acquiring sensor data and communicating the digitized data to the master node. The mote typically contains a sensor, an Analog Front End (AFE) for conditioning the signal, an Analog to Digital Converter (ADC), a processor and an RF radio, as shown in the block diagram in Figure 3.1. Examples of physiological signals acquired by motes includes: electrical signals produced by the brain, electroencephalogram (EEG); electrical signals produced by the heart, electrocardiogram (ECG); blood pressure; respiration rate; blood glucose; blood oxygen; electrical signals produced by the skeletal muscles, electromyogram (EMG); electrical signals produced by the eyes, electrooculogram

(EOG); and 3-axis acceleration. Depending on the BAN architecture, motes may also be responsible for processing the acquired data for the purpose of data compression, identifying interesting events, or other tasks. The master node wirelessly collects the sensor data from the motes, processes the data and may forward the data beyond the BAN. The level of processing, as well as data storage and data forwarding functions, depends on the BAN system architecture.

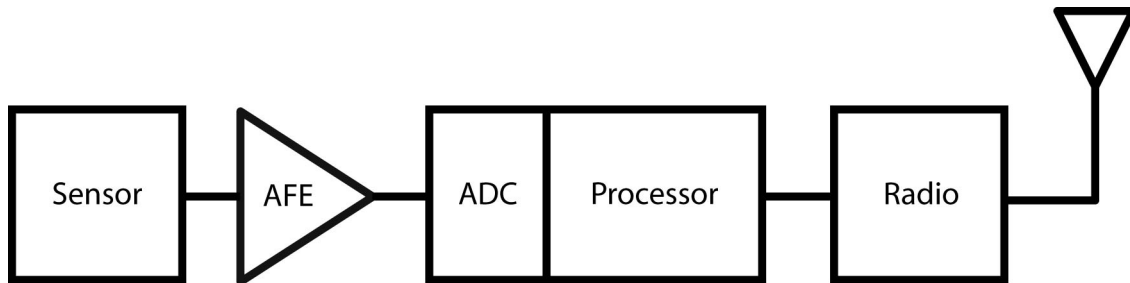


Figure 3.1 Mote block diagram

In recent years research interest focused on biomedical applications of BANs has gathered significant momentum. This is due in part to advancements in wireless communication technology, miniaturization and reduction in power consumption of integrated circuits, and also due to an aging and increasing world population [10]. A more mature and larger population places increasing demands on current health care systems. Appropriate application of BANs will provide many opportunities to alleviate this pressure by reducing hospital stays, technician analysis of test results, wait times for specialized tests or meeting with specialized health care professionals. The benefits gained through use of health care related BANs include reducing health care costs but more importantly these network have the potential to save lives. The ability to detect the onset of adverse health conditions relatively earlier than without BANs will revolutionize the health care industry. Some examples of related and current research include monitoring of ambulatory ECG [11–13], blood glucose [14,15] and pulse oximetry [16]. Regardless of the application, BANs are faced with many challenges.

3.2 Challenges

The benefits of real time physiological monitoring are poised to be significant but prior to BAN systems truly becoming mainstream many challenges must be addressed by researchers. The work accomplished by the BAN nodes must be carried out efficiently due to the small amount of energy available. Each of the nodes must be kept small in order to be comfortable and unobtrusive as possible, facilitating long term use. The BAN system must be affordable and simple to use. Privacy and security are also a significant concern when dealing with personal information, especially health records. Ideally the BAN system will be flexible enough to allow a myriad of sensor modules, applications and communication protocols. Many existing surveys have adequately covered the current state of BANs [17–19]. It is not our goal to exhaustively summarize these surveys. Instead, the remainder of this chapter will focus on four major design challenges, which are especially relevant to reconfigurable BANs: (1) selection of network architecture; (2) selection of hardware; (3) selection of software architecture; and (4) selection of wireless protocol.

3.2.1 Network Architecture

It is evident that there are significant constraints placed on BANs, making the design of such systems non-trivial. As noted by Chen et al. [18], many different systems with different network architectures, communication protocols, processor selections and applications are being researched. The typical BAN architecture uses a star topology on the body, where multiple nodes communicate via ZigBee or Bluetooth with a master node. The master node forwards data via Wireless Fidelity (WiFi) or General Packet Radio Service (GPRS) to a health care professional, according to Dishongh and McGrath [20]. Many implementation variations exist in this architecture, generally driven by the application. Figure 3.2 provides a high level view of the typical architecture. As shown, intra-BAN communications occurs in tier 1 of the BAN system architecture. Any communications that exceed the boundaries of tier 1 are considered tier 2 communications (e.g., event escalation to a health care

professional). What this diagram does not illustrate is the location in the network where information processing of the sensor data occurs or where the sensor data is stored. There are many options in BAN network architectures for processing and storage of sensor data, as the following paragraphs will illustrate.

Patel et al. [21] carries out advanced processing of the data on the mote and only transmits the important features beyond the mote. This architecture provides power savings by reducing the amount of data that requires wireless transmission. However, in this application the processor on the mote must be relatively powerful and consume a relatively large number of processing cycles. In addition, historical data is not available on the master node.

Burns et al. [22] employs a slightly different approach in that the acquired data on the mote is saved to an on-board micro Secure Digital (SD) card. In this application data integrity is maintained during power or communication interruptions. However, the storage of data is distributed among the motes, making data fusion challenging.

Auteri et al. [11] keeps the motes as simple as possible while processing the data at the master node. However, the master node is a PC, which doesn't allow for mobile event escalation for the user.

In this work a network architecture was selected that is a hybrid of those previously discussed. The motes are simple devices responsible for data acquisition and forwarding, with very little processing. The master node processes the data to identify "interesting events" and store the results. This architecture was selected for the following reasons: (1) to limit the relatively high power consumption of advanced processing to a single node (i.e., the master); (2) to centralize the sensor data in order to make data fusion easier and enable real time feedback for the end user; and (3) to keep power consumption of the motes as low as possible by limiting the data processing on these nodes. However, point 3 implies that all raw data will be transmitted beyond the mote requiring relatively high power for RF devices, which is not case. Processing power of the motes is relatively low but these devices are still capable of implementing

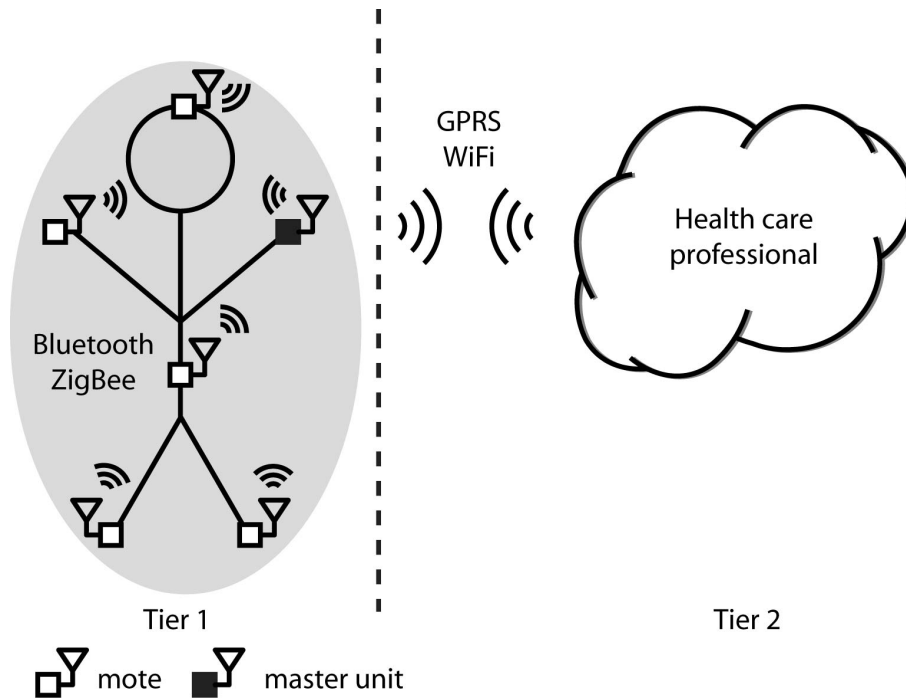


Figure 3.2 Typical BAN System Architecture

algorithms such as compressed sensing or error correcting Reed-Solomon code. In either of these cases the encoding scheme is relatively straight forward requiring limited computing resources but decoding requires significantly more computing resources as noted by Casson and Rodriguez-Villega [23] as well as Sweeney et. al. [24].

3.2.2 Hardware Selection

Many authors have researched the application of FPGAs in BAN systems, specifically placed on the mote [25–29]. There are also a number of studies that place a microcontroller on the mote [11, 13, 22, 30–32]. These documents suggest that microcontrollers and FPGAs both have a place in BANs. It is the opinion of the author that these hardware options should be applied in a complementary fashion. For example, referring back to the typical architecture in Figure 3.2, the motes are kept simple in that they are responsible for data acquisition, data forwarding and data compression and the master node is responsible for data processing, data fusion and storage. In this case the extremely low power sleep mode of modern microcontrollers can be exploited at the mote and the efficient, advanced processing capabilities and

flexibility of the FPGA can be exploited at the master node.

In this work 2 hardware options were considered for processing the data at the master node: (1) microprocessor; (2) FPGA configured as a soft-core processor. In order to accommodate the diverse demands of many different BAN applications, peripherals and communication protocols the hardware requires a high level of flexibility. In addition data processing must be completed efficiently to keep power consumption low.

As BAN applications mature the number of sensors used in each BAN will increase. In addition, different sensors have different data rate demands. For example, the data rate required to record the electrical activity of the brain is significantly more than that required to collect blood pressure data. Based on the previous points several relatively high data rate sensors are likely to be used in a single BAN. For example, a BAN used to detect sleep disorders requires EEG, ECG, EOG and EMG sensors all applied to the same patient at the same time, as shown by Nassir and Barnea [33]. Each of these sensors demands a relatively high data rate. In order to keep the power consumption of the nodes in this BAN as low as possible, both data processing on the node and data transmission from the node to the master node should be as low as possible. To accommodate this, efficient compression algorithms should be applied on the node. As mentioned in the previous section, decoding the data from compression algorithms will require significant computing resources. In this example an FPGA configured as a soft-core processor provides significant advantages over a microprocessor.

Depending on the selection of soft-core processor and size of FPGA selected, implementing a soft-core processor requires only a portion of the FPGA resources. As well, mechanisms are available to communicate data processed in hardware to the software (i.e., operating system) running on the soft-core microcontroller and vice-versa. Therefore processing intensive tasks, such as running several decompression algorithms in parallel, can be completed in hardware and the results can be passed to the operating system. With a powerful enough microprocessor, the same decompress-

sion algorithms could likely be completed in sufficient time. However, in processing intensive applications the FPGA is also more efficient, as the following paragraph illustrates.

It is well known that a hardware solution will consume less energy than a software solution when completing a processing intensive task. Frigo et al. [34] compare the power demands between a microprocessor and a Field Programmable Gate Array (FPGA) in significant data processing applications and note that the FPGA is clearly the lower energy option. In another example, Healy et al. [35] present application layer security that is reactive to attacks in a Wireless Sensor Network (WSN). This is a complex implementation of this level of security that requires significant resources making it well suited for a hardware solution. In another work, Lysecky and Vahid [36] identify reduction in processing time and power by implementing combined hardware/software solutions. They cite possible software speedups of 200 to 1000% and energy reduction by as much as 99%.

Based on the requirements for advanced and efficient data processing the master node in this work contains an FPGA configured with a soft-core processor. The FPGA was selected over a microcontroller for its ability to complete processing intensive tasks in parallel with software running on a soft-core processor as well as to limit the power used by the processing intensive master node. In addition the FPGA provides a migration path to an ASIC.

FPGA Selection

The two largest FPGA manufacturers are Altera and Xilinx, other FPGA manufacturers include Lattice Semiconductor, Microsemi (formerly Actel), Quicklogic and Archonix. Altera and Xilinx will be reviewed in this work. The FPGA selection criteria includes FPGA cost, power consumption, features, including the soft-core processor options, as well as the availability and functionality of the system development software offered by each manufacturer.

The product line from Altera includes programmable logic devices (PLD), application-specific integrated circuits (ASIC), system on a chip (SoC) and FPGAs. Altera has three classes of FPGA: (1) High-End FPGAs called the Stratix Series; (2) Mid-Range FPGAs called the Arria Series; and (3) Lowest Cost and Power FPGAs called the Cyclone Series. Since BAN networks are constrained by power and cost, the Cyclone series was considered as the most appropriate series option for this work. The Cyclone Series of FPGAs is further subdivided into Cyclone and Cyclone II to V. Of these the Cyclone IV is the lowest cost and lowest power, according to Altera. The Cyclone IV devices have up to 150,000 logic elements, 6.3Mb of embedded memory, 4 general purpose phase locked loops and 360 18 x 18 bit embedded multipliers making the hardware highly reconfigurable. System development, functional and timing simulation, power analysis, timing analysis, configuration and verification of Altera devices is done using an integrated development environment (IDE) called Quartus II. Quartus II downloads are available for Windows and Linux operating systems. Included in the Quartus II IDE is a software add-on called SOPC Builder. SOPC Builder is a system development tool that allows specifying the components of an FPGA based hardware system via a graphical user interface (GUI). Examples of available components include memories, processors, serial interfaces, general purpose input outputs (GPIO), phase locked loops, timers and more. Altera offers intellectual property (IP) that can be added to FPGA designs to reduce the design cycle time. Example categories of this IP include digital signal processing (e.g., Fast Fourier Transform, Advanced Encryption Standard), embedded processors (e.g., Nios II, ARM), peripherals (e.g., UART, SPI) and many more.

Xilinx manufactures FPGAs, SoCs and PLDs. The product offering from Xilinx includes 5 FPGA families: (1) Artix-7; (2) Kintex-7; (3) Virtex-7; (4) Spartan-6; and (5) Virtex-6. Of these the Spartan-6 family is described as a family of low-cost and low-power FPGAs, making it a suitable selection for BANs. The Spartan-6 devices contain up to 150,000 logic cells, 4.8Mb of embedded memory, 6 phase locked loops and 180 18x18 bit multipliers, which is comparable to the Cyclone IV devices from

Altera. However, the Xilinx FPGAs in the Spartan-6 family are approximately half the cost of the comparable Cyclone FPGAs from Altera. System development is done using an IDE called Vivado. Within the Vivado IDE Xilinx also offers IP comparable to the IP offerings from Altera. In addition, Xilinx also has soft-core processor IP for use with the Spartan-6 devices called MicroBlaze. The MicroBlaze processor is configured using the MicroBlaze Configuration Wizard.

In summary, both Altera and Xilinx offer low power, low cost FPGAs making them suitable for the constraints of a BAN. The additional features available from each manufacturer are also comparable and they both offer advanced development systems. Xilinx is the superior choice when cost alone is considered. However, none of the hardware or software differences identified thus far were enough to make either manufacturer stand out as the clear choice. To aid the decision making process, the soft-core processors supported by each manufacturer also require investigation.

Altera supports a number of soft-core processor options, such as the ARM Cortex M1 and A9, MIPS32 and Freescale V1 ColdFire but the Nios II processor is the only processor that is both supported by the Cyclone IV family of FPGAs and included in SOPC Builder and was therefore selected for evaluation. The Nios II processor is supported by all of the Altera FPGAs, SoCs and ASICs, allowing for a direct migration path to more powerful programmable devices, should the application demand it. The Nios II is a 32 bit Reduced Instruction Set Computer (RISC) processor with a Harvard architecture. The Nios II processor has 32 bit instructions, 32 general purpose registers and 32 x 32 bit single instruction multiply and divide functions with 32 bit results as well as instructions for 64 and 128 bit results. Three versions of processor are offered: (1) fast (Nios II/f); (2) standard (Nios II/s); and (3) economy (Nios II/e). The fast version is optimized for performance and contains the most configuration and debugging options as well as a 6 stage pipeline. Altera published specifications identify the Nios II/f processor capable of 218 Dhrystone million instructions per second (DMIPS) operating at a max frequency of 185MHz. The Nios II/f requires 1810 logic elements plus the logic elements required by peripherals. Both hardware

acceleration (i.e., moving software functions to hardware) and the ability to add 256 custom instructions are supported by the Nios II processor.

The comparable soft-core processor from Xilinx that is supported by the Spartan-6 family of FPGAs is called MicroBlaze. This processor is also a Harvard architecture processor that uses 32 instructions. Three main configuration options exist for this processor: (1) Performance Optimized MicroBlaze with branch optimizations; (2) Performance Optimized MicroBlaze; and (3) Area Optimized MicroBlaze. The first two options contain a 5 stage pipeline and the last a 3 stage pipeline. The maximum performance specification from the Performance Optimized MicroBlaze is 209 DMIPS at a clock rate of 161MHz. Many aspects of this processor are highly configurable such as the cache size, optional memory management unit (MMU), embedded peripherals and pipeline length. In addition, unused processor instructions (i.e. divide for example) can be removed from the final configuration, saving FPGA area. MicroBlaze also includes a hardware acceleration feature called Fast Simplex Link (FSL) that is a high-speed link between the MicroBlaze processor and hardware logic functions.

Overall, the features and performance of the Nios II and MicroBlaze soft-core processors are very comparable. One item that makes the Nios II processor more attractable is that it has a migration path to an ASIC. However, considering that this work is focused on creating a reconfigurable system the power and cost savings provided by an ASIC design are overshadowed by the fact that the system would no longer be reconfigurable.

In this work the final implementation includes an Altera FPGA configured with a Nios II processor. Since hardware, features and development systems from both Altera and Xilinx are very suitable for a reconfigurable BAN system the final decision was based on access to resources and support.

3.2.3 Software Architecture

In order for a system to be highly reconfigurable, consideration also needs to be given to the software architecture. Regardless of the processor selection, microcon-

troller or soft-core processor on an FPGA, there are two main ways to implement the system firmware: (1) through custom driver development; or (2) through the use of an operating system.

Custom driver development requires programming for every peripheral and initialization of the microcontroller, which can be a challenging undertaking. For example, writing drivers for the management of removable storage is a particularly complex task. On one hand, if the system is not intended to be highly reconfigurable and does not contain overly complex peripherals, writing custom drivers is a preferable solution. In addition, it is easier to exploit the low power sleep modes of modern microcontrollers without having to concern yourself with the complexities of an operating system. In the system designed in this work the motes are designed using custom firmware for the following reasons: (1) to allow extremely low power sleep modes to be easily applied; (2) because the additional services provided by an OS are not required on the mote; (3) processing of the sensor data on the mote is not intended to be easily reconfigurable, as it is on the master node (4) the mote hardware is not relatively complex.

On the other hand, an operating system provides an abstraction layer from the complexities of the processor and peripherals, which enables relatively straight forward reconfiguration of user software. In addition an OS contains services that may be useful to BAN systems (e.g., web server). Researchers have considered the application of operating systems in WSNs. The work done by Wei et al. includes an OpenRISC processor running uC/OS-II [37], Burns et al. [22] and Ton-That et al. [38] are both running TinyOS on their platforms and Peng et al. [39] have applied uClinux to their Safety Monitoring and Warning System.

TinyOS is an embedded OS specifically designed for low power wireless sensor networks that have very limited memory resources. The OS is developed and supported by the TinyOS Alliance. The kernel requires 8kB of program memory and 512kB of data memory. This is an open source OS licensed with a Berkeley Software Distribution license. The current release is version 2.1.2. TinyOS is written in and

user applications are developed using an extension of the C programming language called nesC. TinyOS has ports for Texas Instruments MSP430, Atmel ATmega 128 and Intel XScale PXA271. Good documentation for this OS is available but the learning curve required to be efficient with nesC is significant. This OS is popular in BAN systems however, it does not contain ports for any soft-core processor and therefore was not considered further in this work.

Microcontroller Operating System Version 2 (uC/OS-II) is supported by Micrium Inc. This OS is written in C and also requires a small memory footprint. Depending on the distribution options used the kernel requires 6 to 26kB for program memory and 1+kB for data. This OS contains support for many different processors including 8-64 bit and multi-core variants. Specifically, this OS is ported for Nios II, MicroBlaze, OpenRISC, MSP430, Blackfin and many more. This OS is covered by a commercial royalty-free license (i.e., each end product requires a paid license, \$9,995 per end product when using the latest OS release uC/OS-III). However, the OS is available for free for educational use. uC/OS-II is a real-time operating system (RTOS). A RTOS is designed with a focus on the ensuring the time used to complete a task is deterministic. These operating systems are designed for applications where the time required to complete a task is just as important as the result of the task, for example in closed loop control systems. BAN applications typically do not require the advanced scheduling mechanisms of a RTOS in order to maintain data integrity.

uClinux, pronounced “you see Linux, is an open source operating system originally design for processors that did not contain a memory management unit (MMU). uClinux started as a fork off of Linux 2.0 but later versions include ports for Linux 2.4/6 and also for processors with MMUs. The latest release is version 3.4.0-uc0. Like uC/OS-II this OS is also ported for many processors including soft-core processors, Nios II and MicroBlaze. uClinux is covered under the GNU Public License (GPL), which is a free-software license allowing users to modify the software as necessary for their needs. In this work the master node runs uClinux. In order to reconfigure the user programs a programmer requires knowledge of C and Linux system calls. This

particular OS was selected because it is ported for the selected soft core processor, has open source code, extensive documentation and resources, is covered by GPL, and is popular in the research community.

3.2.4 Wireless Protocol

In order for a wireless technology to be suitable for BANs it must satisfy certain criteria. It must be efficient such that it is capable of long term use (i.e., days, weeks or months) operating on battery power. It must be secure through authentication and encryption. The data rates must be able to satisfy the applications and the wireless technology must be adaptable to accommodate different network architectures and quantity of network nodes. The two most commonly used wireless technologies in BANs are Bluetooth [40–43] and ZigBee [11,32,44,45]. Others include ultra-wideband (UWB), GPRS, WiFi, IEEE 802.15.6 and in-body radio frequency communications, as described by Yang [46].

UWB over IEEE 802.12.3a was a proposed technology for low energy, high data rate (100s of Mbps) Wireless Personal Area Network (WPAN) applications but the Task Group was dissolved in 2006. Following this WiMedia, Bluetooth Special Interest Group (SIG), Wireless Universal Serial Bus (USB) Promoter Group and the USB Implementers Forum all attempted to move the UWB standard forward. Blumrosen et al. [47] are researching UWB in BANs but similar works are limited. Common Off The Shelf (COTS) UWB modules are difficult to find or unavailable. Due to the lack of current research, difficulties and uncertainty with standards and limited availability of parts, UWB has not been given further consideration in this research.

Both GPRS and WiFi are suited for Tier 2 communications but not specifically for intra-BAN communications due to the relatively high power requirements. GPRS also requires a formalized network connection with a telecommunications provider. For these reasons, GPRS and WiFi are not considered suitable wireless technologies for Tier 1 of the BAN.

IEEE 802.15.6 is the Task Group working on the standard for BAN technolo-

gies [48]. According to the Task Group this standard is still in draft and has recently been shared with sponsors to test for support. Since COTS components will not be available for some time this technology will not be further addressed in this research.

Bluetooth technology was originally adopted by the IEEE 802.15.1 Task Group in 2004. Bluetooth operates in the unlicensed 2.400 to 2.4835 GHz Industrial, Scientific and Medical (ISM) band. The physical layer can be set up to limit power consumption to 1mW, which provides a usable range to about 10m, or as much as 100mW, allowing a range of up to 100m. To avoid interference with other devices operating in the same frequency band and for an added security feature, Bluetooth applies a Frequency Hopping Spread Spectrum (FHSS) method for data transmission. While transmitting data, a Bluetooth device will hop between 79 channels, each of 1MHz bandwidth, 1600 times per second using a pseudo random hopping sequence that is known by the receiver and transmitter. The typical network configuration is an ad hoc network, called a piconet. In a piconet, a master can connect with up to seven slaves. In Bluetooth networks there are three distinct security modes according to Healy et al. [35]. Mode 1 is unsecured, Mode 2 initiates security features after a link is established and Mode 3 initiates security features prior to establishing a link. To add to the security features, network nodes can choose to be discoverable or non-discoverable. Non-discoverable nodes are not visible to other nodes in the network. The E0 stream cipher is used to encrypt data. The most recent technology release is Bluetooth v4.0. Bluetooth v4.0 includes a low energy subset called Bluetooth Low Energy (BLE). Prior to BLE, Bluetooth technology had some drawbacks that made it less suitable for a BAN than competing technologies such as ZigBee. However, BLE has reduced power consumption by reducing the duty cycle of transmission to approximately 20% while maintaining the wireless data rate at 1Mbps. This of course impacts the throughput, but at 200kbps this technology still meets the needs of most BAN applications. Network setup time was also addressed in the v4.0 release, going from 6s to 3ms. As a result, BLE is gathering momentum with respect to BANs according to Yu et al. [49].

ZigBee is designed to be a very low power, low data rate technology and is used in many applications including industrial, building and home automation, home entertainment, WSNs, toys, smoke and intruder alarms, and consumer electronics. ZigBee is built on top of the IEEE 802.15.4 standard, which only defines the Physical (PHY) and Media Access Control (MAC) layers for Low Rate Personal Area Networks (LR-PANs). ZigBee defines the network layer and Application Support Sublayer (APS). Two PHYs are specified by the IEEE 802.15.4 standard; the 868/915MHz band used in Europe, U.S. and Australia and the 2.450GHz ISM band used ubiquitously. The wireless data rate is 250kbps per channel and the useable range is 10-100 meters. ZigBee has a useable range of 10-100m. Support for peer to peer and star topologies is available. Three types of devices are defined by ZigBee, Coordinator, Router and End Device. The Coordinators and Routers are defined as Full Function Devices, meaning they include all aspects of IEEE 802.15.4. End Device are defined as Reduced Function Devices, meaning they do not use all aspects of IEEE 802.15.4. Security is based on an open trust model, which means that all layers (i.e., MAC, Network and APS) on a device trust each other. If a layer is responsible for generating a packet then that layer is also responsible for securing it. Security between devices in a network is based on the exchange of 128 bit Advanced Encryption Standard (AES) encrypted keys. ZigBee has an entity called a Trust Center (TC), which is a single device that is trusted by all other devices in the network. The TC is responsible for distributing keys, authenticating devices and managing end to end security in the network [35, 50]. ZigBee supports 64 bit addressing, allowing for over 65000 devices per network. Bandwidth is subdivided into 16 channels, with 5MHz spacing and each ZigBee network uses a distinct channel. In applications that have many ZigBee networks deployed in the same locale there is a possibility that two or more networks will attempt to use the same channel. Sahandi and Lui [51] have researched ZigBee networks in remote patient monitoring in a general hospital ward. The study investigated potential interference problems when ZigBee networks in the same locale used the same channels. The study concluded that multiple ZigBee networks could be applied in the same area provided that appropriate transmission time intervals are

used.

Table 3.1 summarizes several of the important aspects of both Bluetooth and ZigBee, as related to BANs. Two noteworthy points are the number of devices per network and the design complexity. In the near term, eight devices per network may be suitable for BANs but likely this number will be unacceptable as more BAN systems become commercialized. Similarly, the Bluetooth SIG will likely increase this number to respond to market demands. Design complexity is a significant challenge for Bluetooth. If these two protocols are comparable in all other ways, the extra

Table 3.1 Comparison of Low Power Wireless Technologies

Detail	Bluetooth	ZigBee
Data rate	200kbps-24Mbps	250kbps
Devices/ntwk	8	65536
Security	E0 stream cipher	128b AES
Engineering complexity	high	low
Power consumption	low	very low
Frequency	ISM band	ISM band
Related Standard	802.15.1	802.15.4

engineering effort required will have a significant impact on the technology selection for both researchers and commercial designers. However, Bluetooth has something very important that ZigBee doesn't: it is nearly ubiquitous on cell phones. According to Yu et. al. [49] Bluetooth v4.0 will be on all smart phones by the end of 2012. In order for BANs to become mainstream in today's society, integration with smart phones will have to be accommodated. In addition, Bluetooth offers higher data rate services, which may be useful for streaming of sensor data as applications and sensors evolve. However, these higher data rate services are not available with the low power consumption BLE technology. The selection of wireless technology is very difficult and readers are encouraged to allow the applications, and not current momentum, drive the selection.

The system designed in this work uses ZigBee because it satisfies the requirements of BANs in that it uses low power, its secure and it has the required throughput. ZigBee was selected over Bluetooth due to the reduced engineering effort required to configure a network, ZigBee allows more devices per network and because ZigBee is the lower power solution (with the exception of BLE).

Based on the selections made to address each of the four major design challenges identified in Section 3.2, the following chapter includes a detailed description of the BAN system designed in this work.

4. System Architecture

Based on the design choices identified in Chapter 3 a reconfigurable Body Area Network (BAN) system was designed and built. In this system the master node is referred to as the Field Programmable Gate Array (FPGA) Server and the slave node is referred to as the ZigBee Mote. A photograph of the system nodes is provided in Figure 4.1.

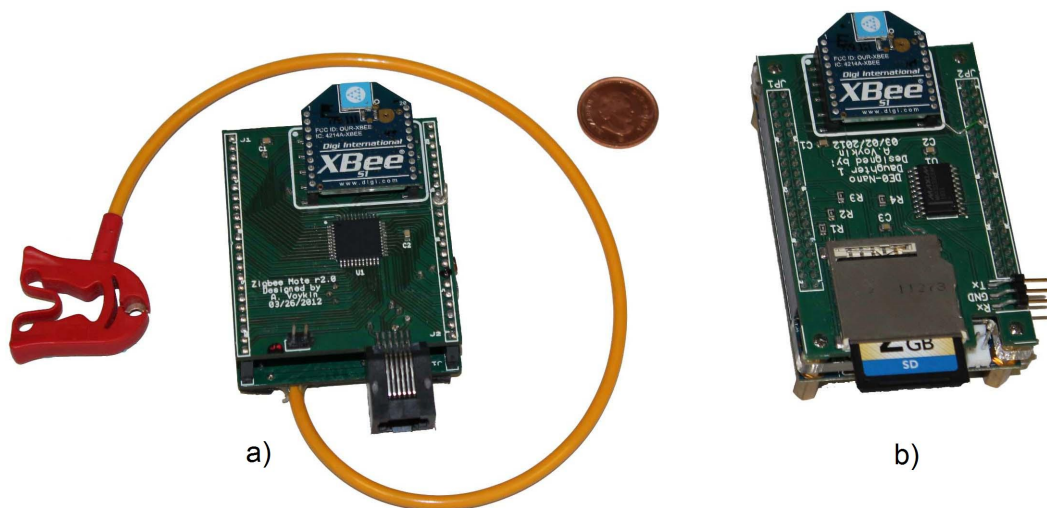


Figure 4.1 Photograph of System Nodes; a) ZigBee Mote; b) FPGA Server

The ZigBee Mote is responsible for data acquisition and wireless transmission of sensor signals. In this work, the mote was used to collect and transmit electrocardiogram (ECG) data. This modular node consists of two pieces of hardware: (1) an ECG Analog Front End (AFE); and (2) Digital Transmitter. Both of these devices were designed and built as part of this work. The ECG AFE contains circuitry for

signal conditioning a modified limb lead II ECG signal (i.e., electrodes on the chest). The major hardware components of the Digital Transmitter include: a Microchip PIC18LF45K22 microcontroller, an XBee radio from Digi International, a -3V DC-DC converter for powering dual-supply operational amplifiers and a pin header for coupling with sensor modules, such as the ECG AFE. As the following sections will show the ZigBee Mote was designed to demand very low power. In the final configuration the ZigBee Mote requires an average of 11mA. Using 2 AAA lithium batteries rated at 3000mAh the mote will work continuously for 11.4 days.

The FPGA Server is responsible for storage and processing of the sensor data received from the ZigBee Mote(s). This node is also modular and consists of: (1) a DE0-Nano Development and Education Board from Terasic Technologies; and (2) a Daughter Board. The DE0-Nano Development and Education Board was purchased from Terasic Technologies. This board contains an Altera Cyclone IV EP4CE22F17C6N FPGA. The FPGA was configured to emulate the Nios II processor and run uClinux. The complete Daughter Board was designed and built as part of this work. It includes an RS232 driver integrated circuit used for system debugging, an XBee radio and a Secure Digital (SD) card for removable storage.

In the configuration demonstrated in this work the system has the ability to record raw ECG data and detect and record R-R intervals, premature ventricular contractions (PVC) and premature atrial contractions (PAC). Moreover, the overall system was designed to be highly reconfigurable, allowing it to be used for other BAN applications besides pattern recognition in ECG data signals.

The remainder of this chapter will describe all of the hardware and software design details for the ZigBee Mote and FPGA Server and close with a description of the reconfiguration abilities of this BAN system. The ZigBee Mote description is first and separated into 2 sections, according to the 2 distinct hardware components: (1) ECG AFE; and (2) Digital Transmitter.

4.1 Electrocardiogram Analog Front End

The overall purpose of the ECG AFE is to prepare the ECG signal measured on the surface of the body for analog to digital conversion. A photograph of the ECG AFE is shown in Figure 4.2.

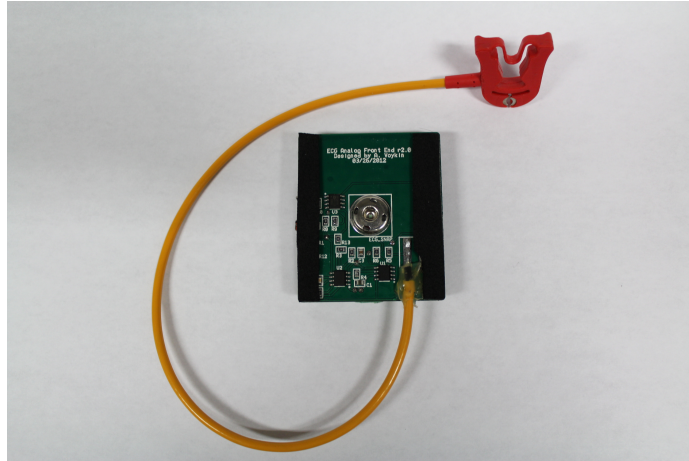


Figure 4.2 ECG AFE Photo

4.1.1 Block Diagram

An ECG signal is difficult to measure due to the low signal voltage, low frequency and sources of interference, as discussed in Section 2.1.1. As this section will prove, a well designed circuit will address these design constraints.

As the high-level block diagram in Figure 4.3 shows, the electrodes attached to the human body connect to an instrumentation amplifier (INA). The INA is used to accommodate the differential nature of the ECG leads and reject common mode voltages in the ECG signal.

In the next stage a passive single pole High Pass Filter (HPF) is used to remove any DC offset in the signal so that subsequent amplification stages do not cause saturation. This block is rectangular to signify that it does not provide gain. This filter is susceptible to loading so it's followed by a buffer.

The output of the buffer feeds a Low Pass Filter (LPF) that has three purposes:

(1) remove high frequency noise; (2) remove alias frequencies; and (3) to provide amplification to the ECG signal.

Finally, the ECG signal is added to a positive DC value through a summing amplifier to offset the ECG signal so that it only has positive voltages. This enables the use of the single supply Analog to Digital Converter (ADC) in the microcontroller.

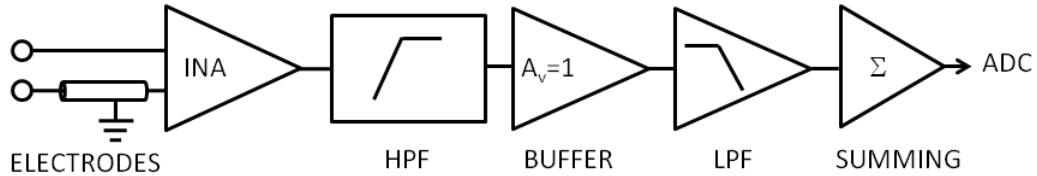


Figure 4.3 ECG AFE Block Diagram

Further details regarding each of these blocks is included in the following schematic description.

4.1.2 Schematic

The circuit-level schematic of the ECG AFE is shown in Figure 4.4. The ECG AFE hardware is connected to the Digital Transmitter using the 1x22 pin headers J1 and J2. Power for the ECG AFE (i.e., $\pm 3V$) comes from the Digital Transmitter via these connectors.

The first amplifier stage is the INA. The INA is a wide supply range, rail to rail output amplifier from Analog Devices, part number AD8227. This INA has a high Common Mode Rejection Ratio (CMRR) of 100dB and a default gain of 5. The high CMRR reduces common mode voltages that exist on the body or are induced from external interference sources. The relatively low gain makes this INA an ideal choice for the first stage of signal conditioning. Too much gain at this stage would saturate the subsequent stages of the ECG AFE due to the unknown amplitude and polarity of the DC offset at the input of the INA, as discussed in Section 2.1.1.

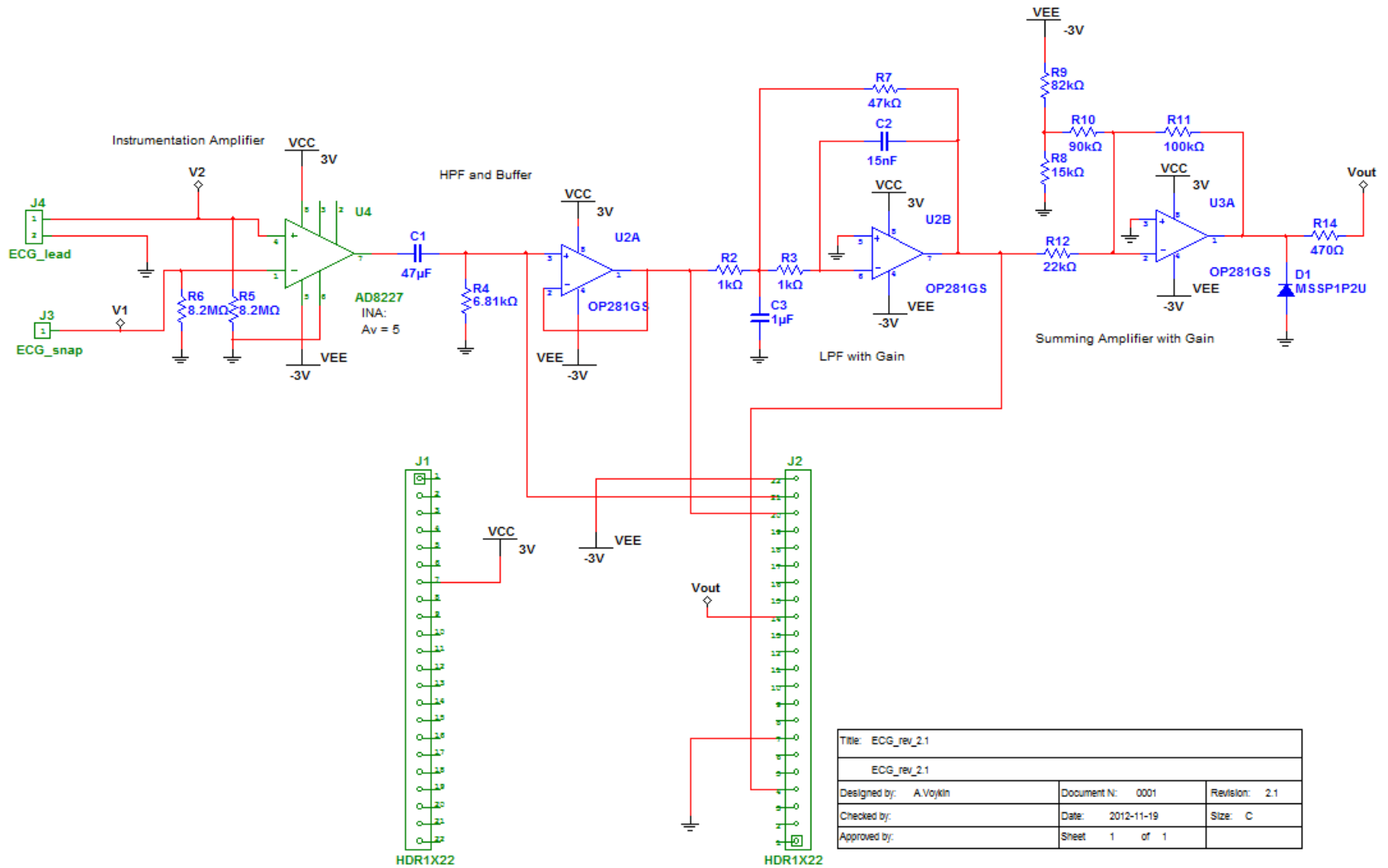


Figure 4.4 ECG AFE Schematic

An $8.2\text{M}\Omega$ resistor is connected to each input of the INA, designated R5 and R6. These resistors are used for input bias current. These resistor values should be high precision to reduce the percentage mismatch in resistance. Any difference in these resistance values will appear as a differential DC input to the INA.

The INA is followed by a passive single pole HPF with a cutoff frequency of 0.497Hz , calculated using

$$f_c = \frac{1}{2\pi RC}. \quad (4.1)$$

The filter response is shown in Figure 4.5.

This filter is required to remove DC voltages caused by differences in impedance between the electrodes and difference in resistance between the biasing resistors, R5 and R6. Since this passive filter is subject to loading, a voltage follower is placed next so that the filter characteristics are buffered from the remaining circuitry.

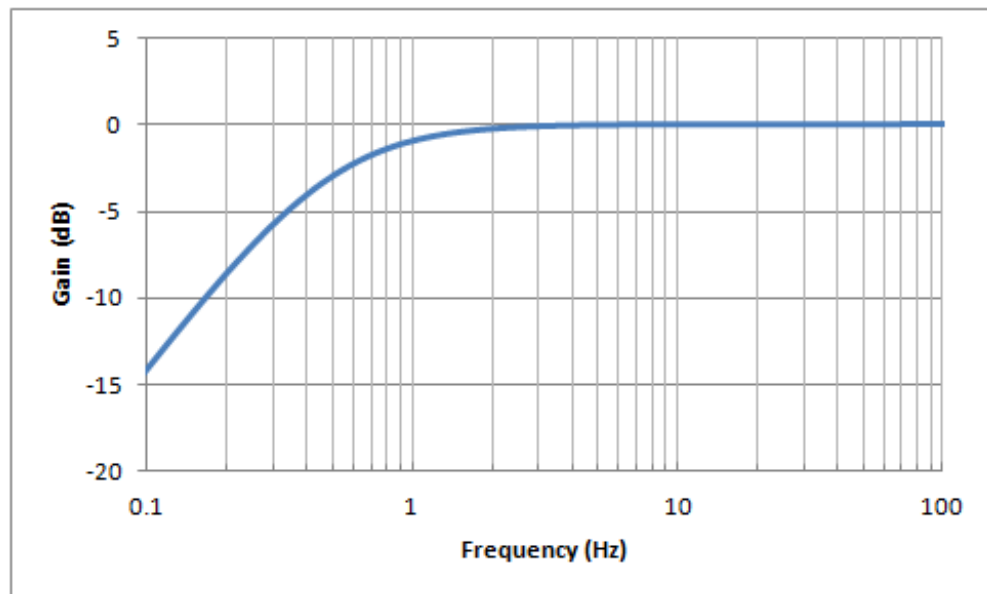


Figure 4.5 HPF Frequency Response

The next stage is a multiple feedback low pass Bessel filter with a cutoff frequency of 150Hz and gain of 50. The filter response is shown in Figure 4.6. The Bessel filter was selected because it does not have ripple in the pass band and it approximates a linear phase response. These characteristics will ensure that the ECG signal suffers

very little degradation as it passes through the filter. The circuit was designed using FilterProTM software version 3.1.0.23446 from Texas Instruments. The selection of the system bandwidth is discussed at the end of this section.

The final amplifier stage is a summing amplifier that adds 0.4V DC to ensure the signal entering the ADC remains positive. This stage also amplifies the signal by 4.54.

Considering the three stages of amplification: (1) INA; (2) LPF; and (3) Summing Amplifier; the end-to-end gain of the system is given by

$$A_v = A_v(\text{INA}) * A_v(\text{LPF}) * A_v(\text{SUM}) = 5 * 50 * 4.54 = 1135. \quad (4.2)$$

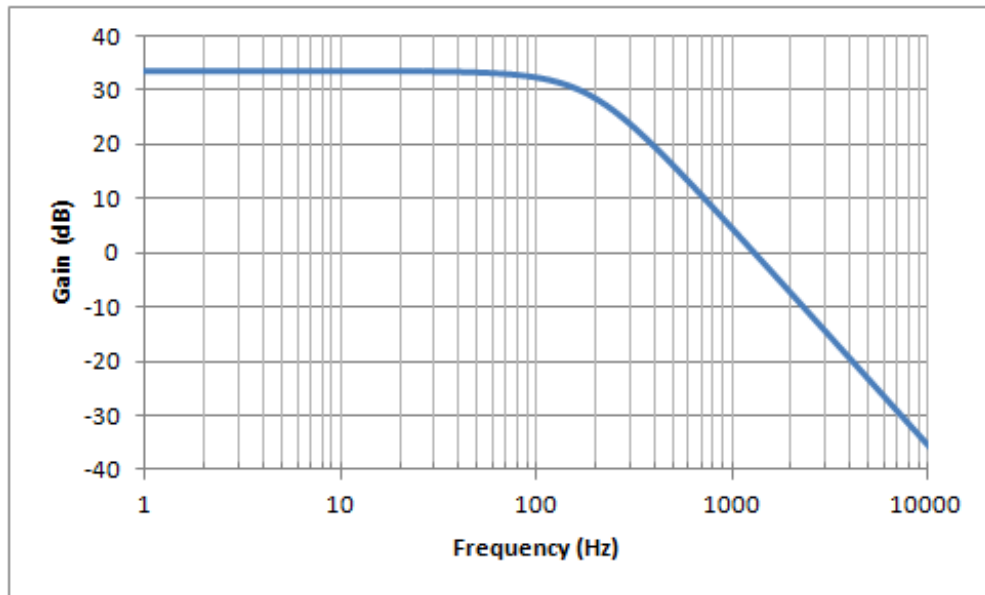


Figure 4.6 LPF Frequency Response

Selection of System Gain

The output of the ECG AFE connects to an ADC input on the Digital Transmitter microcontroller. The ADC is configured to convert a voltage range of 0 to 2.048V. The initial design goal was to set ECG AFE gain from so that the output would span as much of the 2.048V range as possible. The problem is that the ECG AFE

output will exceed the range of the ADC input in the presence of any DC offset. The solution was to reduce the gain in the final stage of the ECG AFE to allow room for the DC offset to vary slightly. For example, when the ECG AFE was connected to the primary researcher the ECG signal at the input to the INA was close to 1.5mV peak to peak. Multiplying by the gain from Equation (4.2) results in an output from the ECG AFE of 1.7025V peak to peak. The difference between the ADC voltage range and the ECG AFE output provides 345.5mV of “wandering room” before the signal exceeds the ADC limits. In order to couple the ECG AFE with the microcontroller ADC 2 additional components are added to the circuit.

The final components in the circuit are a 0.25V Schottky diode (D1) and a 470 Ω resistor (R14). The microcontroller ADC input is limited to -0.3V, D1 is used ensure the input does not drop below this value. R14 is used to protect the microcontroller input in the event of an electrostatic discharge. R14 should be kept relatively low. Higher values (i.e., greater than 10k Ω) will increase the ADC sample period due to the RC time constant between R14 and the ADC input pin capacitance. Microchip uses a 470 Ω resistor in its development systems to protect analog inputs.

Selection of System Bandwidth

According to Chan [52], the required bandwidth for ECG measurement is different for diagnostic and monitor quality systems is as shown in Table 4.1.

Table 4.1 Required Bandwidth for ECG Systems

Quality	Frequency
Diagnostic	0.05 to 150Hz
Monitor	1 to 40Hz

The system designed in this research is capable of monitoring ECG signals. However, in order to build a reconfigurable system a high level of versatility is required. For that reason the initial design goal was to meet the bandwidth requirements of a diagnostic quality system. However, the following problems were encountered with

the HPF when the cutoff frequency was set to 0.05Hz; (1) the bias current from the operational amplifier buffer added a significant DC offset to the output of the ECG AFE; and (2) the ECG AFE output was very slow to react to abrupt changes in the DC offset.

The description of the first problem refers back to the schematic shown in Figure 4.4. The single pole passive HPF is followed by an operational amplifier buffer. In the initial design of the HPF, $R4$ was $681\text{k}\Omega$ and $C1$ was $4.7\mu\text{F}$ giving a cutoff frequency of 0.0497Hz , according to Equation (4.1). The problem is that bias current required by the buffer flows through $R4$ and adds DC voltage to the signal after the HPF. The bias current required by the OP281 amplifier is 3nA . Using Ohm's Law, the DC voltage caused by the bias current at the input of the buffer is 2mV . The gain in the next two stages of the ECG AFE totals 227, according to Equation (4.2). The 2mV DC voltage caused from the bias current results in a DC offset at the output of the ECG AFE of nearly 0.5V . In order to maintain the cutoff frequency selected and mitigate the effects of the amplifier bias current, $R4$ was changed to $6.8\text{k}\Omega$ and $C1$ was changed to $470\mu\text{F}$. A tantalum capacitor was considered to keep the physical size of the capacitor as small as possible. This circuit blocked the DC voltage very well but during testing, the second problem was identified.

The circuit was very slow to respond to baseline changes, especially the relatively rapid changes caused by movement of the patient, as discussed in Section 2.1.1. The problem is that if the baseline changes enough to saturate the output of the ECG AFE then useless data will be recorded for relatively long periods of time while the DC blocking capacitor charges. This is caused by the time constant of the HPF circuit. It takes 5 time constants to fully charge or discharge a capacitor. The time constant in seconds is given by

$$\tau = RC. \tag{4.3}$$

Using $R = 6.8\text{k}\Omega$ and $C = 470\mu\text{F}$ produces a time constant of 3.2s , which means that full charge will take more than 15s . Of course this is an undesirable effect when considering a system that may be used for long periods of time on a mobile patient.

This problem was alleviated by increasing the cutoff frequency to approximately 0.5Hz by increasing C1 to 47 μ F. Increasing C1 by a factor of 10 reduced τ by a factor of 10 to 0.32s.

In addition to the gain and bandwidth several other topics are included in order to fully understand the design decisions made with respect to the ECG AFE circuit.

4.1.3 ECG AFE Testing

Once the ECG AFE was designed and built the hardware was connected to the primary researcher in order to evaluate its operation. Figure 4.7 is an example of the output generated. It should be noted that this example contains two premature ventricular contractions (PVC). In order to test the validity of this hardware, this graphic was shared with a physician and a cardiologist and both were able to diagnose the PVCs.

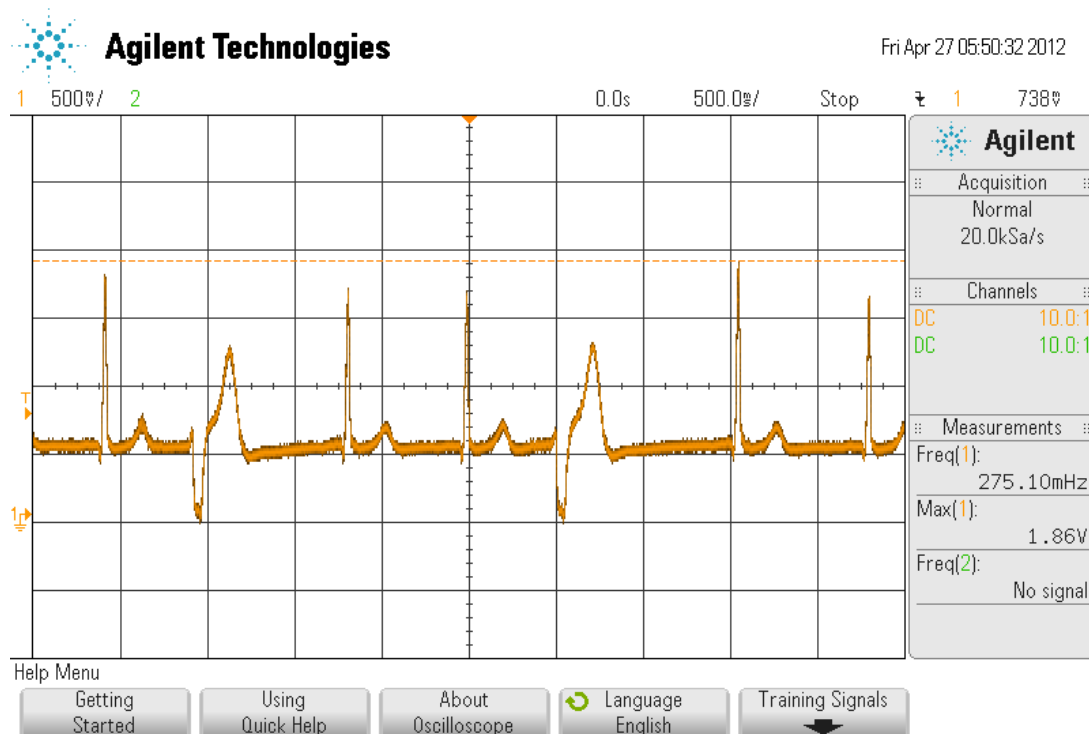


Figure 4.7 ECG AFE Oscilloscope Graphic

4.1.4 Design Considerations

The following considerations will provide some additional insight for designing and testing an ECG AFE.

Analog versus Digital Signal Conditioning

Both analog and digital filtering were investigated as options for signal conditioning of the ECG signal. To aid the explanation the following two assumptions are made, without loss of generality: (1) an INA will be used to connect the ECG electrodes to the remainder of the circuit, regardless of the filtering option selected; and (2) the output of the differential amplifier is 25mV peak to peak (i.e., 5mV input and a gain of 5).

In the digital filtering option the output of the differential amplifier feeds directly to an ADC. ADCs with peak to peak input ranges in the 10s of mV are Common Off The Shelf (COTS) components. However, due to the potential mismatch in electrode-skin impedance between electrodes an undetermined DC offset in the range of $\pm 300\text{mV}$ may also accompany the AC signal. In order to accommodate the ECG signal, both AC and DC components, the ADC input range must be increased to at least $\pm 325\text{mV}$. The result is a reduction in the ADC resolution, which adds significant quantization noise to the digitized ECG signal. For example, the resolution of a 10 bit ADC with an analog input range of $\pm 325\text{mV}$ is approximately $635\mu\text{V}$ per step, according to

$$\text{Resolution} = \frac{V_i}{2^N}, \quad (4.4)$$

where

V_i is the peak to peak analog input voltage level in mV and

N is the number of bits.

The number of levels required to represent the complete ECG AC signal is 40, ac-

cording to the rounded result of Equation (4.5)

$$\text{Levels} = \frac{\text{INA Output (mV)}}{\text{Resolution (mV/level)}} = \frac{25\text{mV}}{0.635\text{mV/level}} = 39.4. \quad (4.5)$$

This represents less than 4% of the available levels from a 10 bit ADC. For this reason, analog filtering was selected over digital filtering in the final implementation

Power Line Noise

Power line noise is often cited as a potential design challenge for ECG AFE designs. In this research the final system is not coupled to a 60Hz power source. However, in order to test the ECG AFE the system is connected to an oscilloscope that is powered with 60Hz AC. This introduced some 60Hz noise into the system. For reader reference, an example of the noise introduced into the system when coupled to an oscilloscope is shown in Figure 4.8. Poor grounding, both circuit and power system, is a major contributor to power line noise observed on an oscilloscope. If overwhelming power line noise is experienced, investigate the grounding circuit used in your measurement.

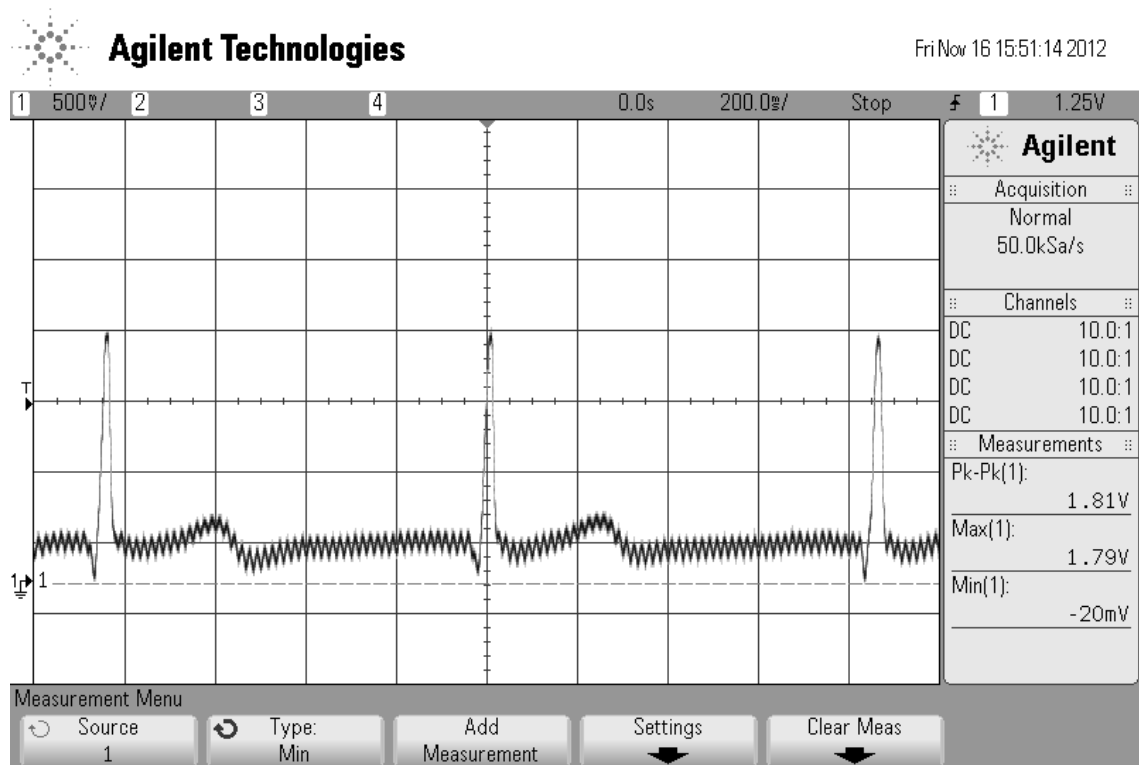


Figure 4.8 Illustration of 60Hz Noise

Recommendations

One of the drawbacks with this ECG AFE design is that the gain and DC offset in the amplifier-filter chain cannot be adjusted. Variances in signal amplitude and DC offset that may be present in different test subjects may drive the ECG AFE into saturation or cause the circuit output to have a very low amplitude. To improve subsequent designs two recommendations are suggested: (1) change resistor R9 to a 100k Ω potentiometer to allow for variable DC offset; and (2) change R12 to a 50k Ω potentiometer to allow for variable gain. Alternatively, feedback could be added to automatically maintain the appropriate gain and DC offset levels.

4.2 Digital Transmitter

The ECG AFE mounts to the Digital Transmitter to make the ZigBee Mote. The Digital Transmitter is responsible for powering the ECG AFE, converting the analog signal from the ECG AFE to digital and communicating the digital samples to the FPGA Server. A photograph of the hardware is shown in Figure 4.9.

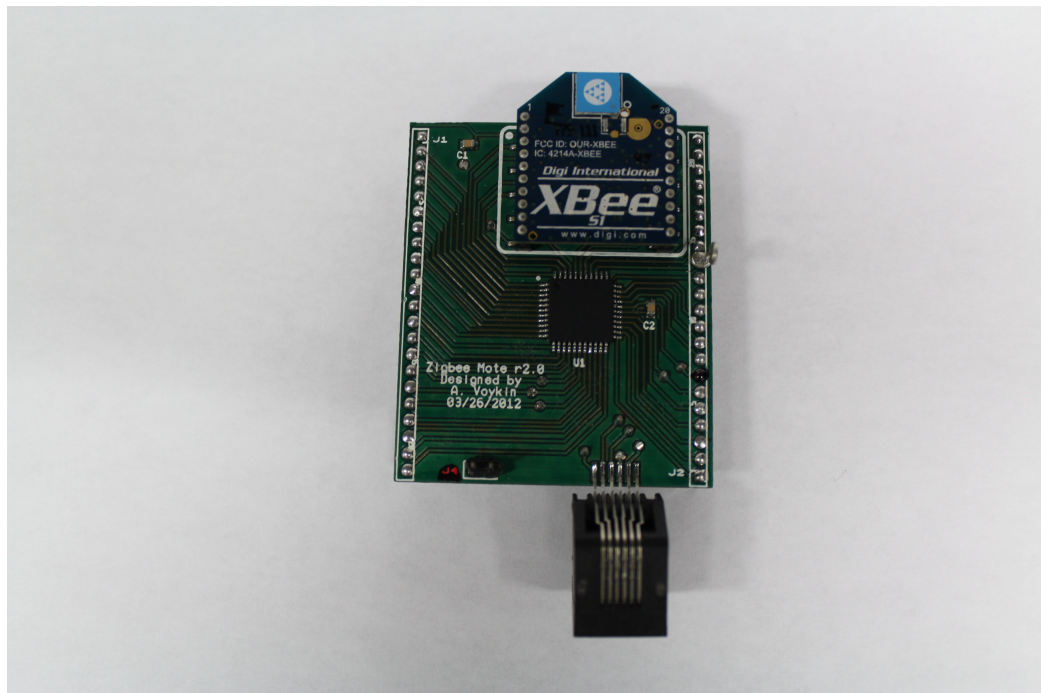


Figure 4.9 Digital Transmitter Photo

4.2.1 Block Diagram

The significant blocks of the Digital Transmitter are shown in Figure 4.10. The

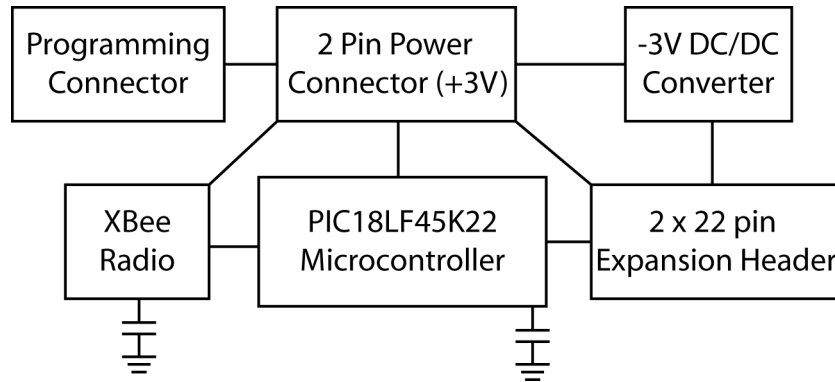
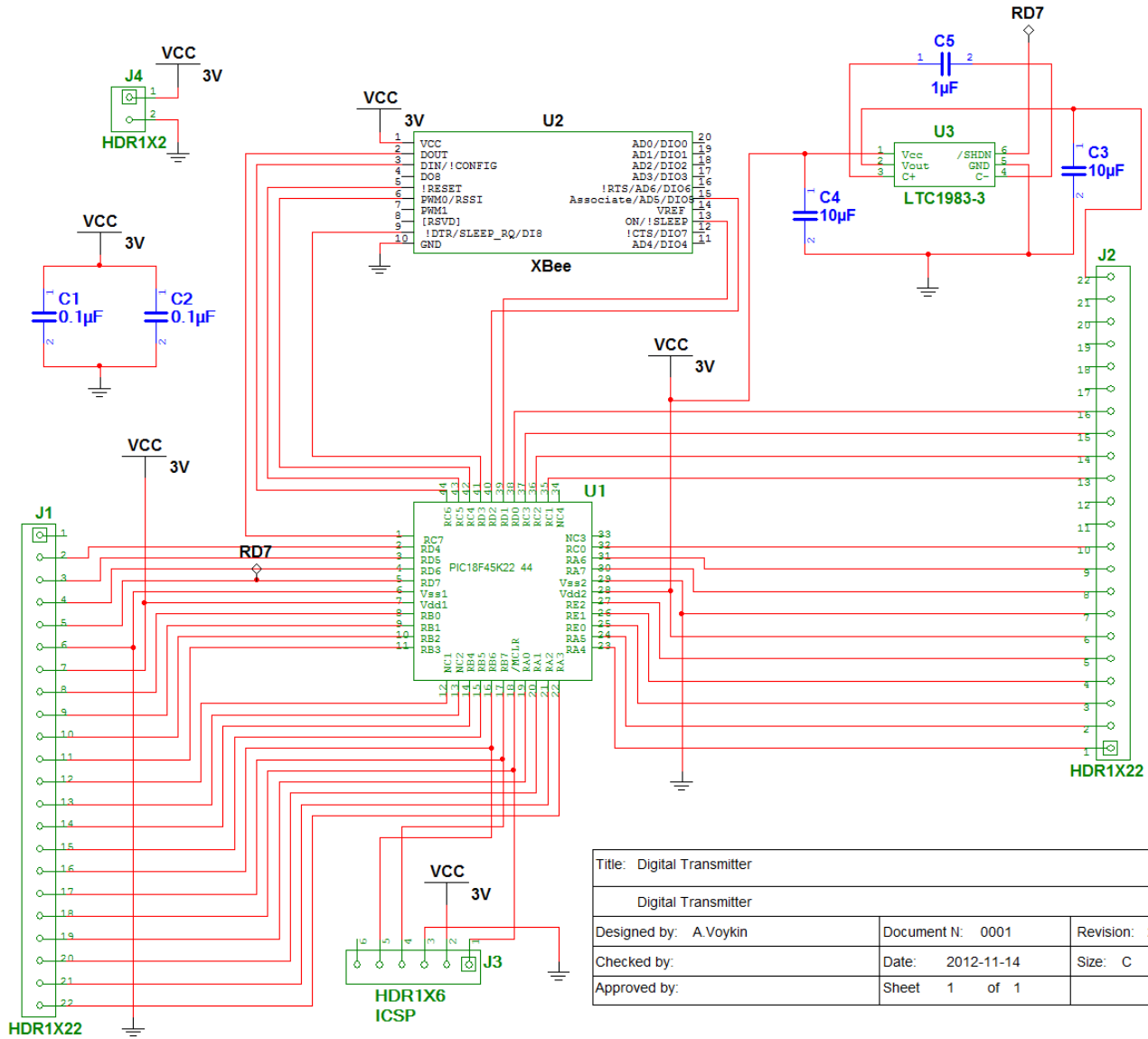


Figure 4.10 Digital Transmitter Block Diagram

primary function of the 2x22 pin Expansion Header is to couple the Digital Transmitter with sensor boards, such as the ECG AFE. The Programming Connector is a 6 pin modular connector used for In Circuit Serial Programming (ICSP) of the microcontroller. The PIC18LF45K22 microcontroller is responsible for analog to digital conversion of the sensor signals, preparation of the data for transmission and interfacing with the XBee Radio. The XBee radio is included to enable wireless communications with the FPGA Server. Many signal conditioning circuits require both positive and negative voltages for dual supply amplifier circuits so a -3V power supply is included on the Digital Transmitter. Two AAA lithium batteries, not included in the block diagram in Figure 4.10, power the circuit and are connected via the 2 Pin Power Connector (+3V).

4.2.2 Schematic

Figure 4.11 provides the circuit level schematic diagram of the Digital Transmitter. As already discussed, the Digital Transmitter is powered by two AAA batteries. These batteries power the microcontroller, XBee module and LTC1983 -3V DC to DC converter. The batteries are attached to the hardware through J4.



Title: Digital Transmitter		
Digital Transmitter		
Designed by: A.Voykin	Document N: 0001	Revision: 2.0
Checked by:	Date: 2012-11-14	Size: C
Approved by:	Sheet 1 of 1	

Figure 4.11 Digital Transmitter Schematic

The 2x22 pin female expansion headers (J1 & J2) are connected to 31 I/O pins on the microcontroller, 2 ground pins, 2 +3V power pins and 1 -3V power pin. The remaining 8 pins on the expansion headers are unconnected. The microcontroller is connected to such a large number of pins on the expansion header to allow for flexibility when interfacing with sensor boards, other than the ECG AFE.

The 6 pin modular connector (J3) is connected to +3V and ground as well as master clear, RB6 and RB7 on the microcontroller. Master clear is used for programming only and cannot be used for I/O. RB6 and RB7 are the programming clock and data lines respectively and can be used for I/O following programming the microcontroller.

The PIC18LF45K22 microcontroller was selected to provide a high level of flexibility when designing and pairing with sensor/daughter boards. This device is manufactured by Microchip Technology Inc. and commonly used in industry. Microchip Technology Inc. also supplies a free and supported development environment, MPLAB Integrated Development Environment (IDE).

The PIC18LF45K22 microcontroller has 44 pins on its Thin Quad Flat Pack (TQFP) footprint. 35 of these pins can be used as I/O and 1 can be used for input only. 28 of the pins can be configured as analog input channels to a 10 bit successive approximation ADC with a configurable internal voltage reference source. The internal oscillator can be configured to operate from 31kHz up to 64MHz. Peripheral options include dual Universal Asynchronous Receiver Transmitter (UART) ports, dual Master Synchronous Serial Ports (MSSP) (used for I2C and Serial Peripheral Interface (SPI) buses), Capture/Compare/PWM modules and comparators. The microcontroller has 4 16 bit timers and 3 8 bit timers. The microcontroller is capable of low power modes (i.e., sleep current down to 20nA) making it suitable for long term use in a BAN application. Many more features are available on this microcontroller and can be identified in the PIC18LF45K22 datasheet [53].

The microcontroller is interfaced with the XBee RF module. The XBee modules meet the IEEE 802.15.4 standard, operate in the 2.4GHz ISM band and offer a range

of up to 30m indoors. These modules operate with a 3V power supply and draw 50mA when not in sleep mode. The modules draw $50\mu\text{A}$ in sleep mode 2, as used in this design. Full technical details about the module can be found in the datasheet [54]. This particular ZigBee module was selected due to the transparent mode operation and the ease of configuration. The transparent mode operation means that pairing of two modules in a peer to peer architecture creates a wireless serial line replacement. The interface method to the XBee module is a 3V logic-level serial port that enables direct connection to the microcontroller on the Digital Transmitter (and FPGA on the FPGA Server).

The XBee modules can be configured through a software graphical user interface (GUI) or using Hayes commands, also known as Attention (AT) commands, which were traditionally used to configure computer modems. The XBee modules contain non-volatile memory, which enables them to be configured out of circuit. Appendix C provides the XBee configuration details for this system.

From the perspective of the XBee module, DIN and DOUT are connected to the microcontroller TX1 and RX1 respectively. TX1 and RX1 on the microcontroller are configured to use the UART. This connection carries all of the digitized ECG samples from the microcontroller to the XBee module. The microcontroller is also connected to the !SLEEP and SLEEP_RQ pins on the XBee module. !SLEEP is an output that identifies when the XBee module is in sleep mode. A high signal from the microcontroller on the SLEEP_RQ pin will place the XBee module into 1 of 3 pre-configured sleep modes: (1) pin hibernate, which consumes the lowest current ($<10\mu\text{A}$) but has longest wake up time (13.2ms); (2) pin doze, which has a low current ($<50\mu\text{A}$) and fast wake up (2ms); and (3) cyclic sleep, which allows periodic checking for RF data. In this work sleep mode 2, pin doze was used due to the low current and fast wake up time.

The !RESET, Associate and RSSI pins on the XBee radio are also connected to the microcontroller and available for future designs but un-implemented in this work. A low on !RESET will reset the XBee module. Associate is an output that identifies

when two XBee modules are logically paired. RSSI is a pulse width modulated output that varies the width with received signal strength of the last RF packet received.

Finally, in order to accommodate signal conditioning circuits the Digital Transmitter is designed to supply both positive and negative voltages to the expansion headers using an LTC1983-3 integrated circuit. The LTC1983-3 is a charge pump DC/DC converter capable of -3V output using only three external surface mount capacitors and a +3V supply, as described in the datasheet [55]. Up to 100mA can be supplied by the converter. The microcontroller has one output pin connected to the /SHDN pin on the -3V DC to DC converter. The /SHDN control pin can be used to place the device into shutdown mode, limiting its current draw to less than $1\mu\text{A}$.

Once the hardware is designed and a prototype built, the next step is to program the microcontroller.

4.2.3 Software

The Digital Transmitter software was developed using Microchip MPLAB v8.80 IDE with the Microchip C18 Compiler v3.40. The overall purpose of the main program is to convert the analog signal from the ECG AFE to digital format and send the digital samples to the XBee module for transmission. The software consists of the main function and functions for initialization of the microcontroller and for servicing interrupts. The flow chart in Figure 4.12 outlines the overall program operation and will be used to support the software description in the following paragraphs.

The main function calls 4 other functions prior to entering an endless while loop: `port_init()`, `serial_init()`, `ADC_init()` and `timer0_init`. In the `port_init()` function the internal oscillator is configured to operate at 16MHz and ports are initialized as analog or digital, inputs or outputs, corresponding to their function in the circuit. In the `serial_init()` function the UART is configured with a BAUD rate of 19k2 with no parity, 8 data bits and one stop bit. The `ADC_init()` function configures the ADC with an internal voltage reference of 2.048V and resolution of 10b, which is the maximum resolution available in this microcontroller. Using a 2.048V reference and

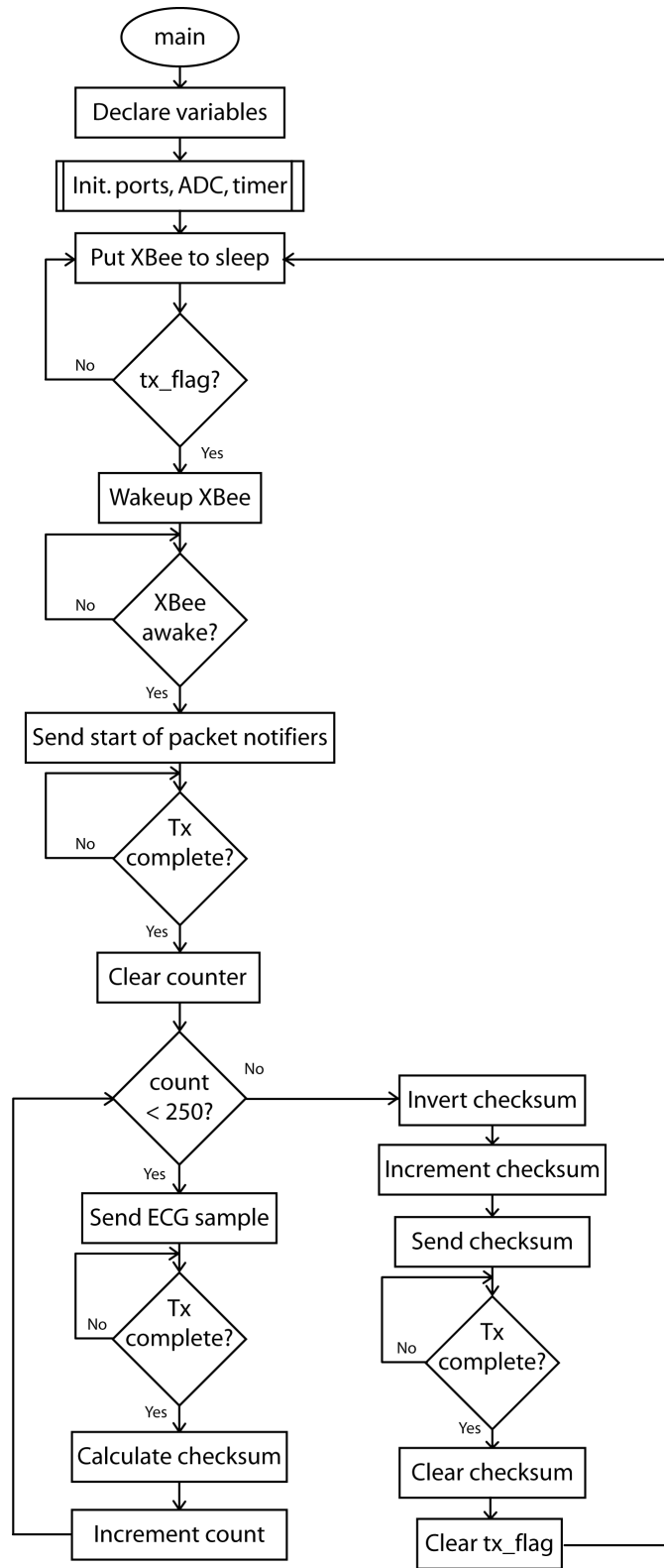


Figure 4.12 Digital Transmitter Flow Chart - main program

10b resolution the ADC step size is 2mV/step. The `timer0_init()` function initializes `timer0` to cause an interrupt every 3.99975ms. This timer is used as the sample clock for the ADC. Following the initialization functions the main function enters the while loop. This while loop has two states: (1) data acquisition; and (2) data transmission. Specific details regarding the selections identified in this chapter are discussed in Section 4.2.4.

In the data acquisition state the main loop holds the XBee module in sleep mode to limit power consumption. In addition, whenever `timer0` expires an interrupt is generated and the interrupt service routine (ISR) requests an analog to digital conversion. The ISR flow chart is provided in in Figure 4.13. Once a conversion is complete the ISR stores the result in a sample buffer. It takes 250 samples to fill the sample buffer. Once the sample buffer is full the ISR copies the sample buffer to a transmit buffer and sets a flag (`tx_flag`) to signal that its time to change to the data transmission state and begin sending data to the FPGA Server. The data is copied to the transmit buffer because while the data is being transmitted samples are still being acquired by the ADC. Placing these samples into the buffer that is being transmitted would corrupt the data.

When the transmit flag (`tx_flag`) is set the main function responds by moving into the data transmission state. First the XBee radio is taken out of sleep mode. Once the radio is ready to receive data (i.e., the status pin, pin 13 on the XBee radio goes high) the microcontroller sends 2 bytes to indicate the start of a packet, 0xaa followed by 0x55. Following this preamble, 250 samples of ECG data are sent to the XBee module for transmission, one sample at a time. Following the transmission of each sample the sample value is added to the checksum. When all 250 samples have been sent, the final checksum calculations are completed and the checksum is sent to the XBee module. Once the entire packet including preamble, sample data and checksum is sent to the XBee module the main function transitions back to the data acquisition state.

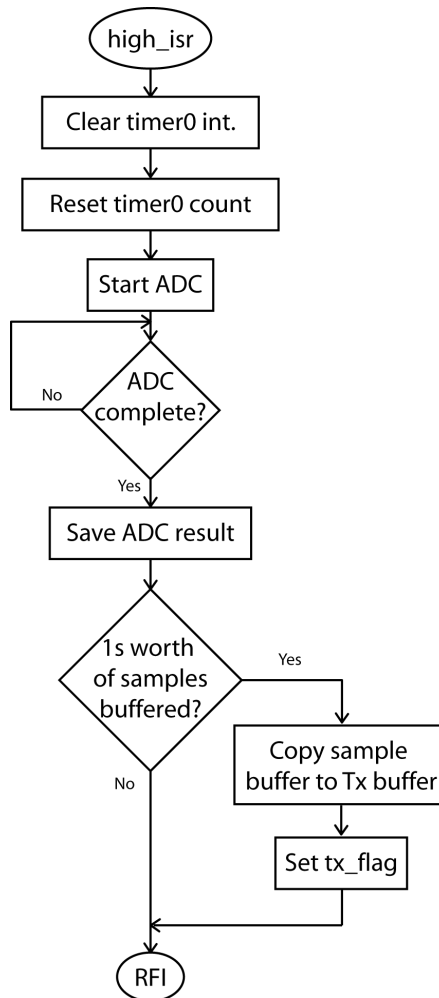


Figure 4.13 Digital Transmitter Flow Chart - interrupt service routine

4.2.4 Design Considerations

At this point several very important details from the preceding software description require clarification: (1) UART BAUD rate; (2) system sample rate; (3) preamble bytes; (4) ECG data packet; and (5) checksum;

UART BAUD Rate

In order to keep the current drawn by the XBee radio low the data transfer rate between the microcontroller and the XBee radio should be as high as possible, without compromising data integrity . In the ZigBee standard the wireless data rate is fixed but the communication rate between processor and radio is not. Anytime the

processor and the radio are communicating the radio must be activated, regardless of whether or not data is being wirelessly transmitted. Getting the data to the radio as fast as possible will allow the sleep duty cycle of the radio to be relatively high, extending battery life. Many data rates were tested but the BAUD rate 19k2 was the highest rate that allowed reliable communications between the ZigBee Mote and the FPGA Server. Hardware flow control was attempted but not successfully implemented in the final design.

System Sample Rate

In the initial software design 4ms was the target sample rate, which equates to 250 samples/s. Referring back to Table 4.1, the upper frequency limit for diagnostic quality ECG is 150Hz. According to Nyquist, the sample rate should be at least 300Hz in order to accurately reproduce the original signal. However, using a sample rate of 250 samples/s allowed the use of the sleep function of the microcontroller and hence very low current draw between analog to digital conversions, extending battery life of the ZigBee Mote. Reducing the system bandwidth in order to extend battery life was considered an acceptable tradeoff. In addition, the diagnostic quality bandwidth of the system was already compromised by the high pass filter in the ECG AFE circuit, as discussed in Section 4.1.2. Therefore, setting the bandwidth to 250 samples/s did not have a direct impact on reduction of the system quality.

The PIC18LF45K22 datasheet suggests that the microcontroller has a programmable sleep period of 4ms to 131s. However, this does not include the wake up time. In Figure 4.14 both rising and falling edges occur when the microcontroller wakes from sleep. The width of ether level is approximately 4.7ms, which is obviously longer than the expected 4ms. The reason for this is because after waking from sleep the microcontroller waits for the clock to stabilize prior to executing any instructions. Rather than compromise the system bandwidth further, in the final implementation the system sample rate was set to 3.99975ms using an interrupt from timer0 not the sleep function.

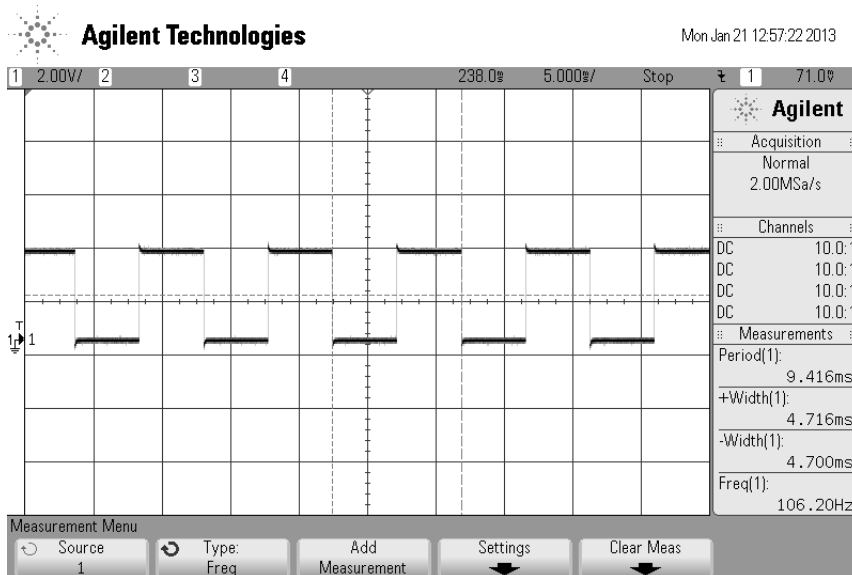


Figure 4.14 PIC18LF45K22 sleep period

Preamble Bytes

The sequence 0xaa followed by 0x55 was selected because it will never appear as a sample in normal ECG data and is easily identified while debugging. For example, the ADC in the PIC18LF45K22 microcontroller produces a 10b result therefore the largest sample, setting all 10b high, is 0x03 followed by 0xff.

ECG Data Packet

Each data packet sent from the ZigBee Mote to the FPGA Server consists of 503B. Figure 4.15 shows a diagram the data packet structure. Each 503B data packet contains 2 start of packet notifier bytes (i.e., 0xaa and 0x55), 1s worth of ADC samples (i.e., 500B), according to Equation (4.6)

$$\text{Payload Time} = \frac{500\text{B}/\text{packet}}{2\text{B}/\text{sample}} * 3.99975\text{ms}/\text{sample} = 999.9\text{ms} \quad (4.6)$$

and a single byte checksum. A packet containing 500B of ECG data was selected for 2 reasons: (1) 500B corresponds to 1s worth of ECG data. In the event of a dropped or corrupted packet 1s of data was is considered an acceptable loss; and (2) the default block size for the SD card on the FPGA Server is 512B. Using a data packet of less than 512B facilitates a single write to memory for each packet. The SD card on the

FPGA Server will be discussed further in Section 4.3.

With the UART configured with a BAUD rate of 19k2, the transmission of 503B takes 262ms according to

$$\text{Transmit Time} = \frac{503\text{B}/\text{packet} * 10\text{b}/\text{B}}{19.2\text{kb}/\text{s}} = 262\text{ms}. \quad (4.7)$$

In Equation (4.7) the 10b/B factor is used because RS-232 uses a start and stop bit for every byte so the effective throughput is only 80%.

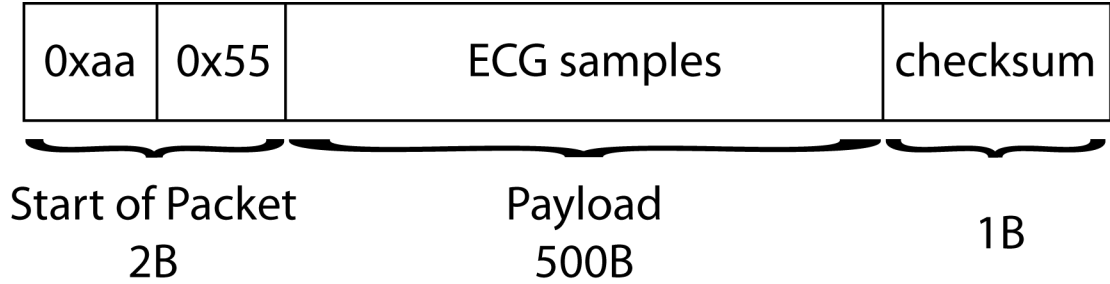


Figure 4.15 ECG Mote Packet Design

The microcontroller continues to interrupt the transmission to take ADC samples so the transmission time is actually slightly longer. The ISR takes 103μs including the analog to digital conversion and it occurs every 3.99975ms. Dividing the time to transmit 503B at 19k2 BAUD by the sample rate produces the number of interrupts per packet transmission time

$$\text{Interrupts Per Packet Tx Time} = \frac{262\text{ms}}{3.99975\text{ms}/\text{interrupt}} = 65.5 \text{ interrupts}. \quad (4.8)$$

Finally, adding the time required to service 65 interrupts to the time required to transmit 503B at 19k2 produces the overall transmission time for each packet

$$\text{Total Packet Transmit Time} = 65 * 103\mu\text{s} + 262\text{ms} = 268.7\text{ms}. \quad (4.9)$$

Checksum

A modular sum checksum was included to detect errors in transmission between the Digital Transmitter and the FPGA Server. At the Digital Transmitter the checksum is calculated by adding the value of all the bytes in the payload, ignoring overflow,

and then taking the two's complement of the result. At the receiving end the payload bytes are again added together and also added to the checksum. If the result is not zero, bit errors have occurred. An example is shown in Figure 4.16.

Transmitted			
	Byte 1		11101011
	Byte 2		11001010
	Sum		10110101
	checksum		01001011
	Received (no errors)		Received (error in Byte 1)
Byte 1	11101011	Byte 1	1110101 0
Byte 2	11001010	Byte 2	11001010
Sum	10110101	Sum	10110100
checksum	01001011	checksum	01001011
checksum + Sum	00000000	checksum + Sum	11111111

Figure 4.16 Modular Checksum Example

This method of error detection is very efficient but has a relatively high probability of not detecting multiple bit errors when compared with other error detection techniques, such as cyclic redundancy check.

Microcontroller Clock Rate

The internal oscillator frequency in the PIC18LF45K22 microcontroller is tunable through software. Not tuning the oscillator frequency or incorrect tuning of the oscillator frequency will affect the ADC sample rate. Details on how to tune the oscillator can be found in the PIC18LF45K22 datasheet [53].

Following the complete programming of the ZigBee Mote hardware and configuration of the XBee radio, sensor data can be sent to the FPGA Server.

4.3 Field Programmable Gate Array Server

The FPGA Server is the most complex component in the BAN system due to both hardware and software design considerations. As already mentioned, the FPGA Server includes the DE0-Nano Development and Education Board and a custom daughter board called the DE0 Nano Daughter Board. A photograph of the 2 parts of the FPGA Server is provided in Figure 4.17.

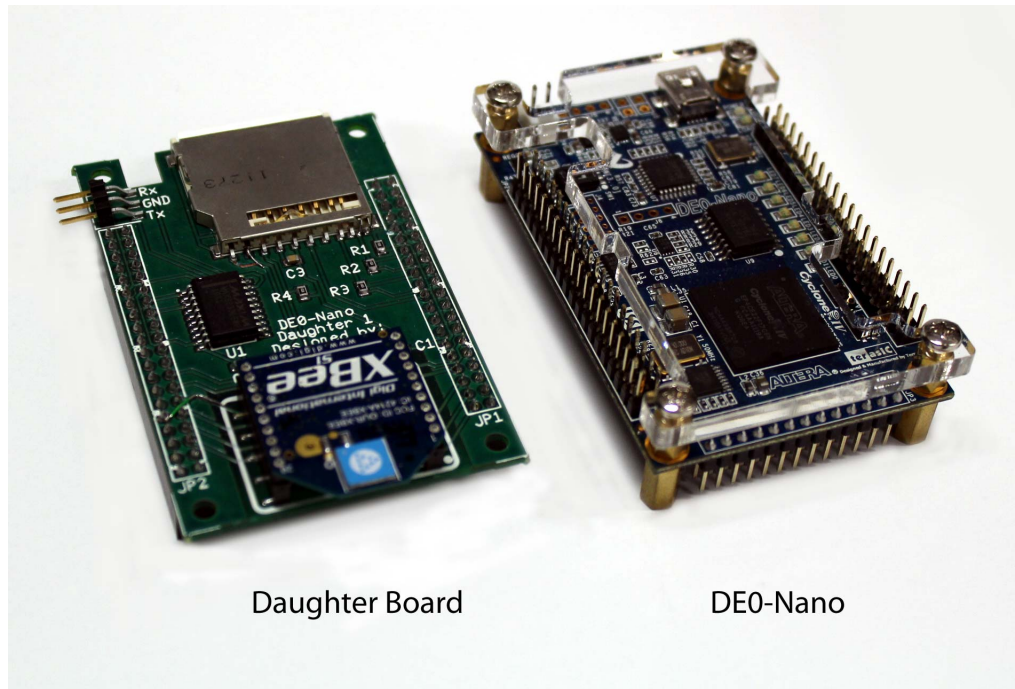


Figure 4.17 FPGA Server photo

The FPGA Server is responsible for storage and processing of the sensor data received from the ZigBee Mote(s). Once the sensor data from the ZigBee Mote(s) is received and processed, it is stored on an SD card. The SD card also contains user programs that dictate how the received data will be processed.

As mentioned, the design of the FPGA Server is complex. Therefore, for clarity, the following sections describe the FPGA Server design using a top-down approach. A block diagram of the complete system is covered first. Design decisions with a major impact on the outcome of the final system are described next. A description of the DE0-Nano Daughter Board schematic follows. Once all of the hardware has

been discussed the configuration of the complete system is addressed. The section concludes with a description of the user software that runs on the completed system.

4.3.1 Block Diagram

The block diagram is separated into 2 major blocks and several minor blocks, as shown in Figure 4.18. The major blocks represent the individual pieces of hardware that make up the FGPA Server. The DE0-Nano Development and Education Board was purchased from Terasic Technologies Inc. [56]. The DE0-Nano Daughter Board is a custom add-on board that mates with the DE0-Nano Development and Education Board and provides the additional peripherals used in this system.

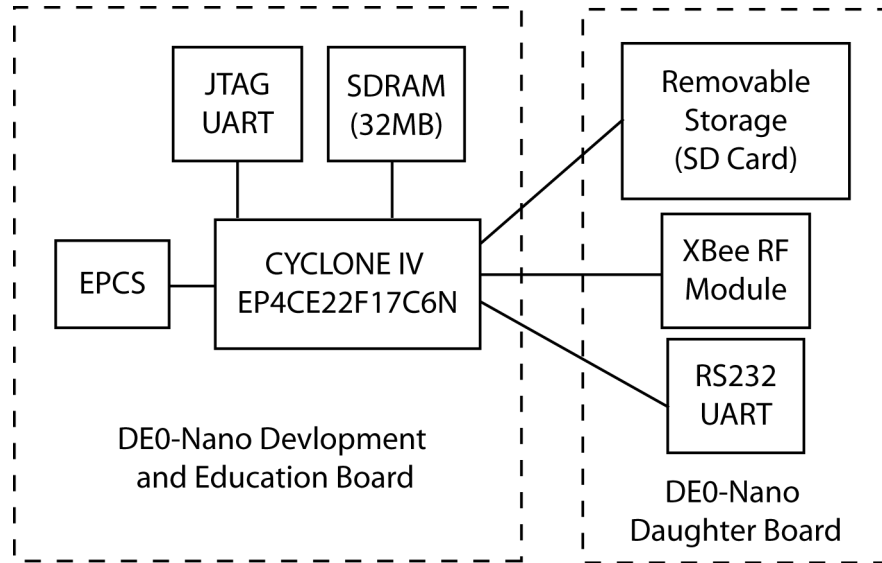


Figure 4.18 FPGA Server Block Diagram

The DE0-Nano Development and Education Board was identified as an option for this research due to its small size (7.5cm x 5cm) and because it uses an Altera FPGA. Altera hardware is popular in research and commercial products. A review of the DE0-Nano Development and Education Board hardware revealed that this platform would be suitable for a BAN master node. The review considered the FPGA manufacturer, the available memory and peripheral devices and the ability of the FPGA to run a soft core processor.

Table 4.2 summarizes key parts of the DE0-Nano Development and Education Board that are significant to this work. These include the FPGA, EPCS, Synchronous Dynamic Random Access Memory (SDRAM) and Joint Test Action Group (JTAG) UART. The FPGA is an Altera Cyclone IV EP4CE22F17C6N device. The EPCS is an Altera type code that refers to serial configuration devices. The EPCS is used to configure the FPGA and load the operating system (OS) into SDRAM at power up. The JTAG UART is used to configure and debug the system.

Table 4.2 DE0-Nano Development and Education Board Hardware

Component	Detail
FPGA	Altera Cyclone IV EP4CE22F17C6N
Configuration	USB-Blaster and EPCS
Memory	32MB SDRAM and 2Kb I2C EEPROM
Clock	External 50MHz oscillator
Power	USB or 2-pin External Power Header (5v)
Peripherals	2x40 Pin Expansion Headers for 72 GPIO pins 8 LEDs, 2 Pushbuttons, 4 DIP switches 8-channel 12b ADC (200ksps) 3-axis Accelerometer

The additional peripherals provided by the DE0-Nano Daughter Board are the removable storage, the XBee RF module and the RS-232 UART. The removable storage is used for storing user data and user programs, the XBee RF module is used to receive data from the ZigBee Mote and the RS-232 UART is used for debugging. The DE0-Nano Daughter Board will be discussed in further detail in the following section.

4.3.2 DE0-Nano Daughter Board Schematic

The DE0-Nano Daughter Board schematic is provided in Figure 4.19. The DE0-Nano Daughter Board has two 40 pin connectors (JP1 and JP2) that are used to

couple to the DE0-Nano Development and Education Board. When the daughter board is coupled with the DE0-Nano Development and Education Board these connectors attach to 72 GPIO pins on the FPGA. 16 of the 72 GPIO pins are used for I/O; 6 pins are connected to the SD Card socket, 8 pins are connected to the XBee RF module and 2 pins are used for the transmit and receive lines of the RS-232 UART.

Removable Storage

USB flash drive and SD card were the two main choices considered for removable storage. Of course USB memory has a much larger storage capacity and faster data rate than SD memory. When selecting removable storage for BAN applications these are not the only factors that require consideration. The physical nature of a USB memory device can be cumbersome. There are some very low profile USB memory sticks but even the smallest of these typically exceed the boundaries on a printed circuit board (PCB). The SD card was selected because it offers a smaller footprint when using the micro size (15mm x 11mm x 1mm) and it can be placed on a PCB such that it has little to no protrusion. In addition, the SD card consumes less power than a USB flash memory device [57,58]. Both draw approximately 30-35mA during read and write cycles but USB uses 5V power and the SD card is a 3.3V device. USB devices also draw significantly more current in standby than SD cards.

Three SD card sizes are available: standard, mini and micro. This project used the standard size SD card (32mm x 24mm x 2.1mm) to ease prototyping. The electrical interface is the same for all sizes of SD cards allowing for seamless transition to a smaller footprint card in the future.

There are three ways to communicate with SD cards: 1 bit SD mode, 4 bit SD mode and SPI. Use of both 1 and 4 bit SD modes requires licensing from the SD Card Association according to [59]. In order to avoid licensing fees SPI was selected for the communication protocol.

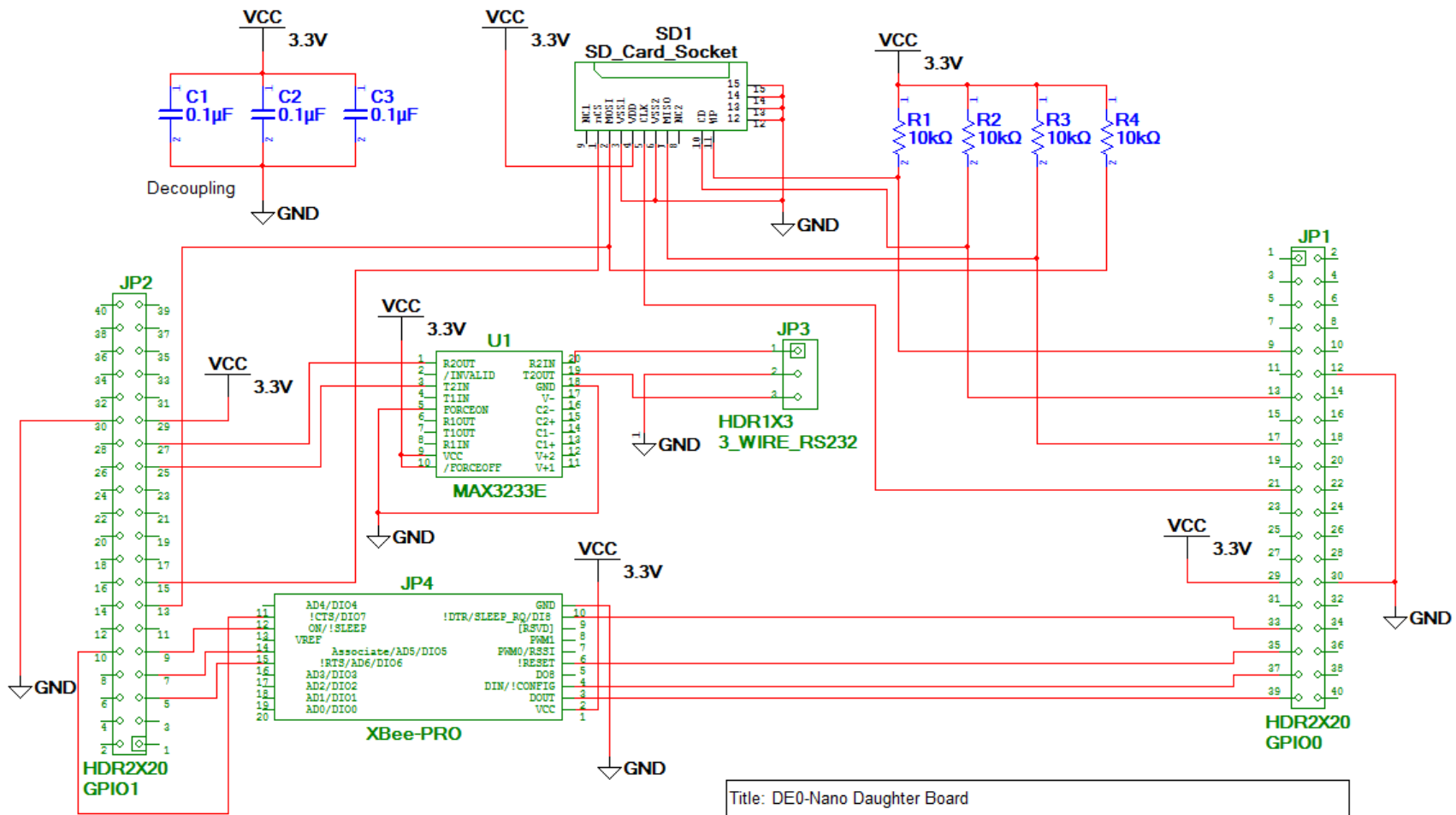


Figure 4.19 DE0-Nano Daughter Board Schematic

The SD Card Association standard specifies rates up to 25MB/s. In this work the SOPC Builder configuration is set to specify the SPI data rate at 128kbps, which is sufficient to save 1 packet of ECG data quickly enough to avoid interfering with the next packet. The default read/write block size is 512B according to Part 1 of the SD Specifications [60]. An ECG packet size of 503B, as identified in Section 4.2.3, fits into a 512B block. Therefore, saving 1 ECG packet takes only 32ms, according to Equation (4.10).

$$\text{Time to save 1 block} = \frac{1}{128\text{kbps}} * 8\text{b/B} * 512\text{B/block}. \quad (4.10)$$

As mentioned, the SD card communicates with the microcontroller using SPI. SPI uses a master/slave communications mode where one device initiates and has control over data transfers with all other devices. In this system the SD card is the slave and the Nios II processor is the master. Master In Slave Out (MISO) and Master Out Slave In (MOSI) are the data output and data input pins respectively, from the perspective of the SD card. On the SD card MOSI is pin 2 and MISO is pin 7. The MISO and MOSI pins have 10k pull up resistors. The MISO pull up resistor is required so that the pin does not float when an SD Card is not inserted in the card holder. The pull up on the MOSI pin is required only if the SPI pins are to be used in a bus architecture with other slaves, and could be eliminated in this design. The Card Select pin, pin 1, is active low and does not require a pull up resistor because SD Cards have an internal pull up on this pin. The CLK pin, pin 5, is driven by the FPGA and therefore does not require a pull up. Pins 11 and 12 on the SD card connector are optional connections for detecting when the SD card is plugged in and whether or not it is write protected. These pins are implemented in the SD card holder as single pole single throw switches with one terminal grounded. The pins are unused in this work.

XBee Module

As already mentioned the XBee module is connected to the FPGA using 8 pins; DIN, DOUT, Associated, !SLEEP, SLEEP_RQ, !RESET, !CTS and !RTS. DIN and

DOUT are the only pins used in the final implementation. All of the pins listed have been discussed in Section 4.2.2, with the exception of !RTS and !CTS. These 2 pins can be used for RS-232 hardware flow control.

RS-232 UART

The RS-232 UART was added for debugging. The primary reason was because the documentation on the Altera Forum [61] suggested that using the USB-Blaster JTAG UART would cause the uClinux OS not to boot from memory. This situation was tested and proved to be false. This allowed all testing and debugging of the hardware and OS to be done via the USB-Blaster JTAG UART. The RS-232 UART is included in the final hardware but unused in the system configuration. To complete the hardware description, the RS-232 transceiver is included.

The RS-232 standard specifies voltage levels of 3 to 15V for a logic zero and -3 to -15V for a logic 1. The standard also specifies that RS-232 devices must be able to withstand short circuits and voltage levels of $\pm 25V$. Since the FPGA I/O pins cannot support this directly, a special transceiver chip is used. The MAX3233E is a dual RS-232 transceiver with internal capacitors. This particular transceiver was selected because the IC requires no external components and because it operates with a 3.3V supply, which is the same voltage level as the FPGA GPIO pins.

When the design of the daughter board is complete and a prototype is built and connected to the FPGA Server the complete system can be configured.

4.3.3 System Configuration

This section will discuss the process of configuring the FPGA Server. Appendix A provides a step by step guide, which includes all of the design flow aspects from building the development environment on a PC to automatically running custom user programs through the uClinux kernel boot sequence when the FPGA Server powers up. An initial design should follow the steps in Appendix A in the order they are provided.

In order to complete the steps in Appendix A many software programs were required, these are listed in Table 4.3 and discussed in the following paragraphs.

Table 4.3 List of Software Used to Develop BAN System

Package	Version	Use
Quartus II	11.1 Build 173	FPGA Configuration
Altera SOPC Builder	11.1 Build 173	System Configuration
Nios2 Command Shell	11.1 Build 173	FPGA Configuration, Programming Flash Memory Terminal Emulator
uClinux-dist	20100621	Kernel Customization
uClinux kernel	2.6.30	Embedded OS
X-CTU	5013	RF Module Configuration
Ubuntu	11.10	Development PC

The development tool supplied by Altera is called Quartus II. Quartus II enables design, analysis and synthesis of FPGA or Complex Programmable Logic Device (CPLD) systems. Quartus II contains tools for system development, functional and timing simulation, power analysis, timing analysis, verification and more. Quartus II allows Register Transfer Level (RTL) entry via Verilog, VHDL, and block/schematic diagram. In addition to the core functions of the Quartus II software there are many built-in software packages available. Of particular importance to this work is a programmable system development tool called System on a Programmable Chip (SOPC) Builder.

SOPC Builder is a system development tool that allows specifying the components of an FPGA based hardware system via a Graphical User Interface (GUI). Examples of available components include memories, processors, serial interfaces (i.e., USB, UART, SPI), General Purpose Input Outputs (GPIOs), PLLs, timers and more. Once a design is finalized SOPC Builder automatically generates all of the interconnection logic required to realize the system. SOPC Builder also generates HDL files for each

of the hardware modules specified in the design, as well as an example instantiation of the overall system. These files are used to generate configuration files for an Altera FPGA. A comprehensive library related to designing systems using Quartus II and SOPC Builder can be found at the Altera website [62]. Specific information related to the use of embedded peripherals in FPGA designs can be found in Altera's Embedded Peripheral User Guide [63].

Once the Nios II system is configured, included in a Quartus II project and compiled, Nios II Command Shell is used to configure and communicate with the hardware. The following functions of the Nios II Command Shell are particularly important to this work: configuring an FPGA, loading uClinux into SDRAM, converting uClinux image files and FPGA configuration files into flash files that are suitable to load into the EPCS, programming the EPCS, configuring Joint Test Action Group connections and opening a terminal with uClinux. Further details regarding Nios II Command Shell can be found in Chapter 4 of the Altera Embedded Design Handbook [64]. Once the FPGA hardware has been configured to run the Nios II processor uClinux can be run.

A uClinux distribution is referred to by uClinux-dist. Each uClinux-dist contains the operating system kernel as well as libraries, including a C library called uClibc for developing C programs, and the toolchain required for building a custom uClinux kernel and developing user software. Many different uClinux distributions can be downloaded but care must be taken to acquire the distribution that is ported for Nios II processor. Appendix A provides the appropriate location for this download and the following references provide an excellent starting point for running uClinux in an embedded system, [61,65,66]. It is recommended to develop a uClinux system using a PC running a Linux OS. Ubuntu was used in this work. Ubuntu was selected because it is a free and open source operating system.

Finally, the XBee radio is configured to communicate with the ZigBee Mote XBee radio. X-CTU is a GUI software package from Digi International that enables configuration and testing of XBee RF modules. Appendix C provides the details on how

to use X-CTU to configure the XBee modules used in this system.

Once all of the software packages from Table 4.3 have been acquired, installed and configured, according to Appendix A, the FPGA Server can be configured. The general outline of the process is shown in Figure 4.20. A discussion of each of the steps follows.

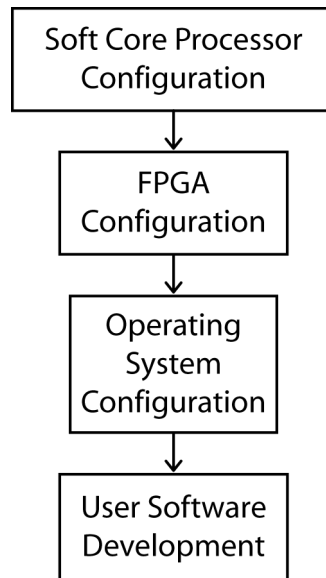


Figure 4.20 FPGA Server Configuration Flow Diagram

Soft Core Processor Configuration

Altera FPGAs can emulate many 3rd party soft core processors, as noted on the Altera website [67]. The Nios II processor was selected because it supports all of Altera’s FPGAs and it is developed and supported by Altera. This allows quick and easy access to design resources and allows seamless transitions to other FPGAs should the application demand it.

Following a conceptual hardware design the first step in the system development cycle is to configure the soft core system. This involves identifying all of the components required in the system, including the Nios II processor, memory and peripherals, and identifying how the components are logically connected together. SOPC Builder is used to complete this step.

Once the system is fully configured, SOPC Builder will automatically generate all of the files required to include the design in a Quartus II project. SOPC Builder generates the following; Verilog (.v) files for all of the modules in the system, an example Verilog instantiation for the system, a Quartus II Intellectual Property (.qip) file, a Synopsis Design Constraints (.sdc) file and a Peripheral Template File (.ptf). Page 1-6 in the SOPC User Guide [68] further describes the outputs upon generating a system with SOPC Builder. The .ptf file is used to configure the uClinux kernel and will be discussed with the Operating System Configuration. The .v, .sdc and .qip files are required to configure the FPGA and hence all need to be added to the Quartus II project.

FPGA Configuration

The next step is to configure the FPGA. The example Verilog instantiation file is copied to the top level design entity in the Quartus II project. The .qip file provides path information to the Quartus Intellectual Property files included in the design, such as the Nios II processor. The .sdc file is used to set timing constraints prior to building the Quartus II project. Once the SOPC generated files have been added to the Quartus II project and the system instantiation is included in the top level design entity, the I/O declarations can be completed. Following the I/O declarations a preliminary compilation is run. After successful compilation all of the I/O will appear in the Assignment Editor and can be logically connected to pins on the FPGA. This is followed with a full compilation that outputs an SRAM Object File (.sof).

The .sof file is used in conjunction with the USB Blaster and Nios II Command Shell to configure the FPGA. Since the Cyclone IV FPGA uses volatile Static Random Access Memory (SRAM) cells to store configuration data, the device must be reconfigured every time the power is cycled. The .sof file can also be used to create an FPGA configuration flash file. This flash file is then programmed into the serial configuration device (i.e., EPCS flash memory). By programming the EPCS flash memory with the FPGA configuration flash file, the FPGA is configured automati-

cally at power up. The configuration flash file has a .flash file extension.

There are many file extensions used in the configuration of the system, for convenience these file extensions are summarized in Table 4.4.

Table 4.4 File Extensions Used in System Development

File Name	Extension	Purpose
SRAM Object File	.sof	FPGA Configuration
Quartus II Intellectual Property	.qip	Path to Quartus II IP
Verilog	.v	RTL Programming
Synopsis Design Constraints	.sdc	Timing Analysis
Peripheral Template File	.ptf	Hardware Abstraction
Flash	.flash	EPCS Programming

Operating System Configuration

In order to use the uClinux toolchain to compile the uClinux kernel for a Nios II system a .ptf file is required. The .ptf file contains a full system description, including clock rates, naming conventions and logical connections. Appendix A.5 provides the details on how to introduce the .ptf to the uClinux toolchain.

Once the uClinux toolchain has knowledge of the .ptf file the kernel can be customized. This is done using a menu configuration GUI that enables inclusion of all the necessary operating system features, including the drivers for the system hardware. Upon successful compilation of the uClinux kernel, a kernel image called “zImage” is created. Using the zImage file, the base address of the EPCS, a bootloader and the name of the .sof created by compiling the Quartus II project, a .flash file can be created. This .flash file is programmed into the EPCS flash memory and allows automatic booting of the uClinux OS at power up. This .flash file must be programmed into the EPCS after the FPGA configuration flash file has been programmed.

Following the successful completion of the system configuration, the user software is written.

4.3.4 User Software

User programs process the incoming data from the XBee radio (i.e., the sensor data from the motes). These programs can be automatically launched by the OS during the boot sequence by adding instructions to the kernel run command (rc) file. Appendix A.7 provides the location of the rc file in the uClinux-dist file structure and a description of how to edit the file.

In this system the user programs are copied from the SD Card to the OS file system during the OS boot sequence. The user programs are written in C and compiled with a cross compiler. The programs are cross-compiled so that they can be launched by a uClinux OS running on a Nios II processor. The uClibc library is used during compilation. The cross-compiler is a GNU Compiler Collection (GCC) compiler and is installed during the Quartus II installation as part of the Nios II Embedded Design Suite (EDS).

In this work the user software is responsible for receiving data from the XBee radio, verifying the validity of the transmission via a modular checksum, processing the ECG data in order to extract interesting events and storing the ECG data and interesting events to an SD Card. The user software is located in Appendix G.4, entitled `serial_test`. A discussion of the software flow follows and is supported by the flow chart in Figure 4.21.

Following variable declarations, the first job of the user software is to create a file on the core file system that will be used to save ECG data and interesting events. This is accomplished using a function called `fopen()` with the `w+` option. The `w+` option creates a file and opens it for editing. If the file already exists the command overwrites it. In this particular application the purpose of this command is to ensure the system starts with a new copy of the file every time the OS boots.

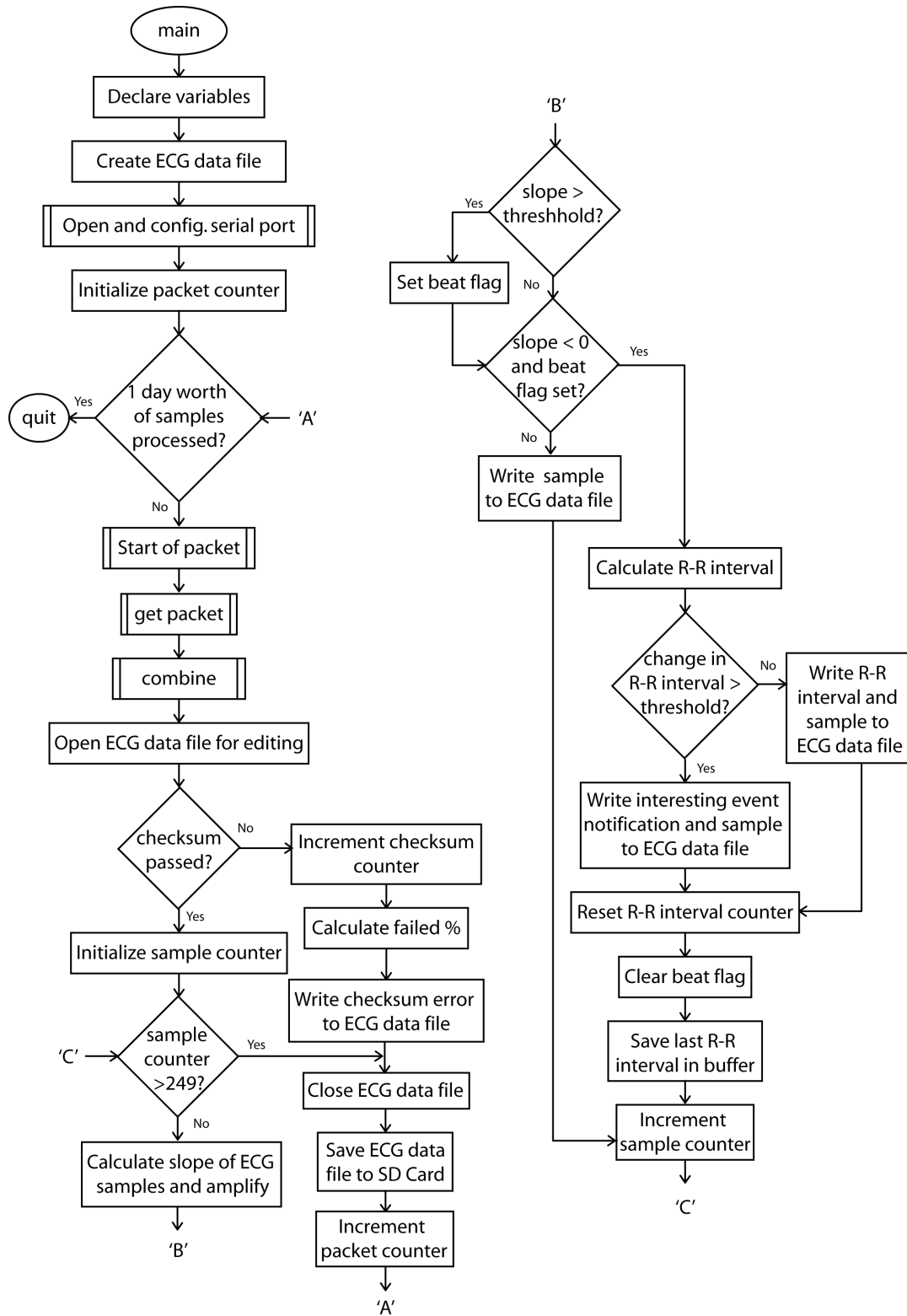


Figure 4.21 FPGA Server User Software Flow Chart - main()

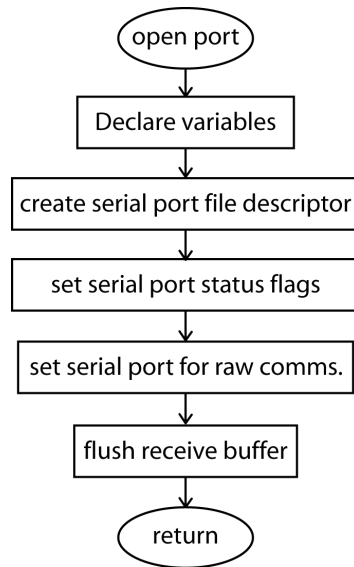


Figure 4.22 open_port() Flow Chart

Function: open_port()

Next the serial port is configured by calling the function open_port(). The flow chart for the open_port() function is shown in Figure 4.22. The open_port() function begins by declaring a termios data structure. The termios structure contains flags used to define the communication and interface parameters of a serial device. Next a file descriptor is declared and used to open a serial port by calling a standard UNIX system call, open(). The function open() requires the path to the serial port, "/dev/ttyS0" and configuration options. In this case the serial port is opened for reading and writing (O_RDWR), does not become the controlling terminal for the process (O_NOCTTY) and is opened in non-blocking mode (O_NDELAY) (i.e., does not make the calling process wait under any circumstances). When the serial port is successfully opened the port attributes are set to defaults using the system call fcntl(fd, F_SETFL, 0).

In the communications of ECG data, all byte values from 0-0xFF are possible. The default set up for the serial port is to use canonical mode, which processes the data before releasing it to the user program. In canonical mode American Standard Code for Information Interchange (ASCII) codes such as carriage return, line feed and

tab will be processed and parsed out of the ECG data. Of course this is unacceptable and so the port needs to be configured to allow all data to pass from the serial port to the user program, unprocessed. This is done using the function `cfmakeraw()` with a pointer to the `termios` structure as the argument.

In order to make the changes to the `termios` structure flags affect the serial port attributes the function `tcsetattr()` is called. Finally, any unread data in the serial port input buffer is flushed and the file descriptor for the open serial port is returned to `main`.

At this point the main program can start receiving serial data. The program is set up to continuously receive and process ECG data for 1 day. This is done by initializing a for loop to 86400, which equals the number of seconds per day. Each cycle of the for loop processes one packet and each packet contains 1 second of ECG data.

Function: `start_of_packet()`

The reception of data starts by waiting for two start of packet identifiers to enter the serial port. The flow chart for the `start_of_packet()` function is shown in Figure 4.23. This is a blocking function, meaning that it will continue to loop until the bytes received are `0xaa` followed by `0x55`. Once the `0xaa` and `0x55` identifiers

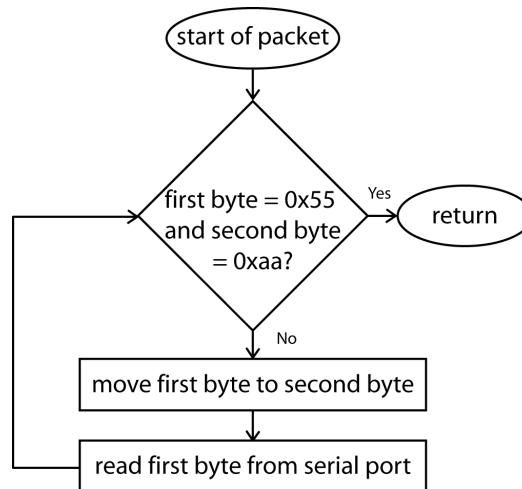


Figure 4.23 `start_of_packet()` Flow Chart

have been received, 500B of ECG data are read from the serial port into an unsigned character buffer (i.e., 8b wide).

Function: get_packet()

The flow chart in Figure 4.24 shows the process for receiving the remainder of the data packet. The incoming data is checked for validity using a modular checksum and the result of the checksum is returned to the main function.

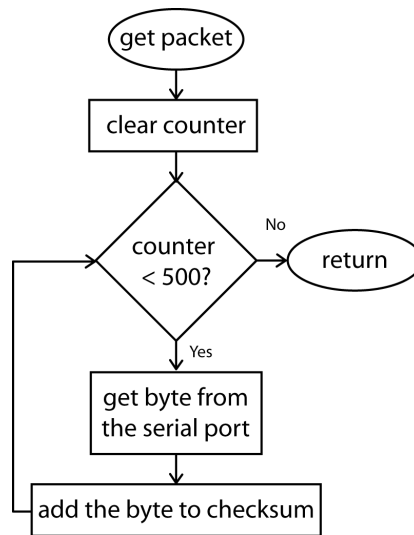


Figure 4.24 get_packet() Flow Chart

Function: combine()

The nature of RS-232 communications forces the data to be sent 1 byte at a time, however the ECG ADC samples are 10b long and therefore require 2 bytes per sample. Prior to processing the samples to identify interesting events a combine function is called. The flow of this function is shown in Figure 4.25.

This function reads two bytes from the received data buffer and combines them into a 16b word, most significant byte first. Then the next two bytes are combined. This continues until all 500B in the received data buffer have been combined and stored into a 250 element unsigned character buffer (i.e. 16b per element).

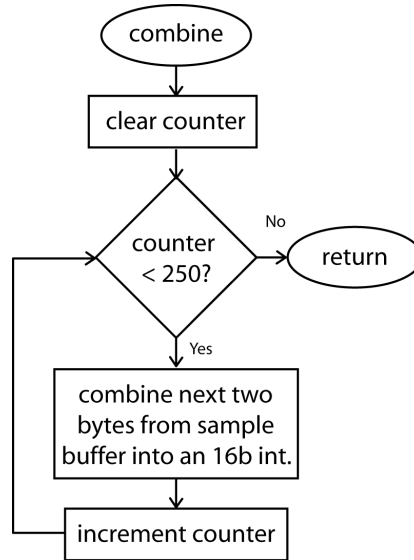


Figure 4.25 combine() Flow Chart

Prior to processing the received packet, the checksum is evaluated. If the checksum identifies data errors then a checksum error is written to the ECG data file and the main loop restarts, waiting for a new packet of data. If the checksum passes the packet is processed and the ECG data is written to the ECG data file.

R-R Interval Identification

Prior to proceeding with the description of how the system identifies R-peaks and calculates R-R intervals, terminology specific to this software needs to be defined. In Appendix G.4 the variable “beat_flag” is used to identify the onset of a QRS complex. The “beat_flag” is set when an ECG data sample exceeds the “R-peak threshold”.

Identifying an R-peak is accomplished using the following sequence: (1) pass the ECG data ($x[n]$) through a software differentiator to produce $y[n]$,

$$y[n] = 5 * (x[n] - x[n - 1]); \quad (4.11)$$

(2) set the “beat_flag” when $y[n]$ exceeds the “R-peak threshold”; and (3) identify the sample immediately preceding where $y[n]$ (i.e., the slope of $x[n]$) changes from positive to negative. This is original work however, this algorithm shares similarities with the QRS detection algorithm proven by Pan and Tompkins in [69].

The gain of 5 in Equation (4.11) was originally thought to provide greater flexibility when selecting the “R-peak threshold” but could be removed without impacting the operation of the overall system. It should be noted that if the gain is removed from Equation (4.11), the “R-peak threshold” also needs to be divided by 5.

Table 4.5 and Figure 4.26 provide example data used to locate an R-peak. In Table 4.5, n is the sample number, $x[n]$ is the ECG data and $y[n]$ is the data processed using Equation (4.11). Sample number 10 is the R-peak. In Figure 4.26, (a) is the sample in $y[n]$ that causes the “beat_flag” to be set, (b) is identified as the R-peak and (c) is the negative sample in $y[n]$ used to signal the software that the R-peak occurred on the previous sample.

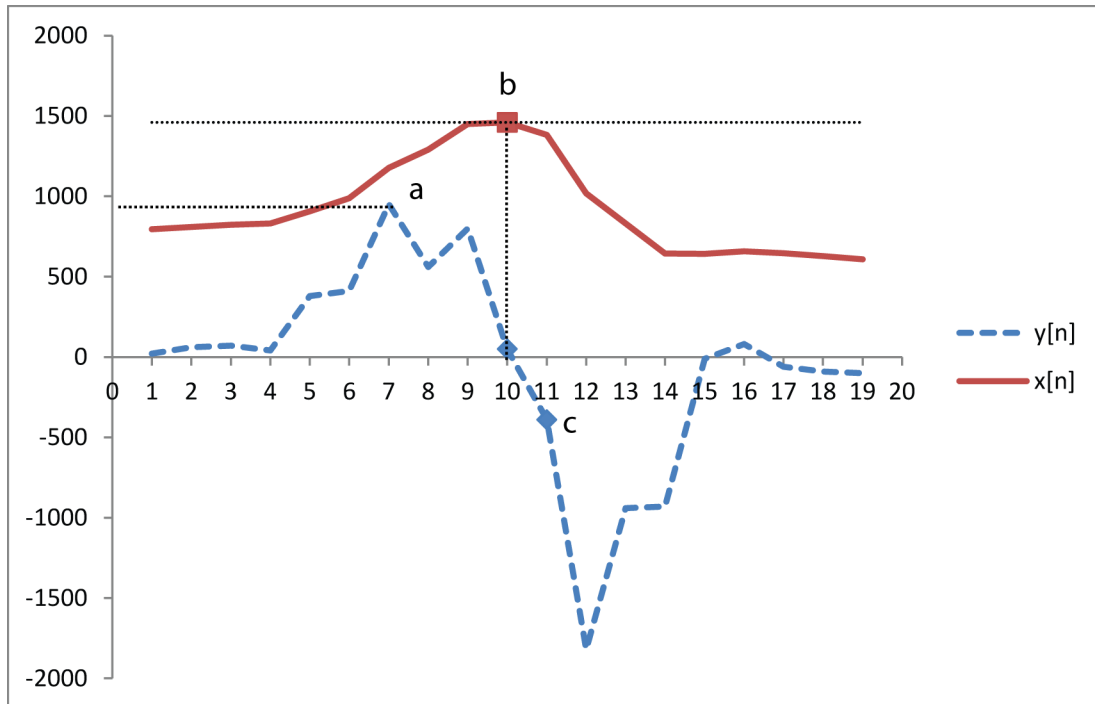


Figure 4.26 Illustration of R-peak Detection

Knowledge of the relative time an R-peak occurs as well as the sample rate facilitates a simple division in order to calculate the R-R interval

$$\text{R-R Interval} = \frac{\text{Number of samples between R-peaks}}{\text{Sample Rate (samples/s)}}. \quad (4.12)$$

Table 4.5 Data Supporting R-peak Identification Example

n	$x[n]$	$y[n]$
0	792	-
1	796	20
2	808	60
3	822	70
4	830	40
5	906	380
6	988	410
7	1178	950
8	1290	560
9	1450	800
10	1460	50
11	1382	-390
12	1018	-1820
13	830	-940
14	644	-930
15	642	-10
16	658	80
17	646	-60
18	628	-90
19	608	-100

Having knowledge of the R-R intervals, interesting events such as PVCs and PACs can be identified. In this work PVCs are identified by monitoring the ratio of adjacent R-R intervals and setting a flag when the ratio exceeds a preset limit. In this software the limit is called the “R-R interval ratio threshold”. In order for this algorithm to successfully identify a PVC the peak of the PVC waveform can not be classified as an R-peak, as discussed in Section 2.1.2. Figure 4.27 shows an example of a PVC alongside normal sinus rhythms from the MIT-BIH Arrhythmia database file MIT-BIH

223. The resultant waveform $y[n]$ produced using Equation (4.11) is also shown. It is obvious that the PVC has a much lower positive slope than the normal sinus rhythms. For this reason $y[n]$ of the PVC will not exceed the R-peak threshold. The result of this is a calculated R-R interval approximately twice as long as the previous. When the ratio of adjacent R-R intervals exceeds the “R-R interval ratio threshold”, an interesting event is detected and a resulting message is written to the ECG data file. PACs are detected in the same fashion.

In order to detect a PAC the ratio of adjacent R-R intervals must also exceed the “R-R interval ratio threshold”. The key difference is that a PAC will have a more normal QRS complex and hence will be classified as having a normal R-peak. PACs and PVCs can be discriminated by adjusting the “R-R interval ratio threshold”. PVCs will have a ratio of approximately 2 while PACs will be considerably lower, but still higher than 1, as shown in Section 2.1.2.

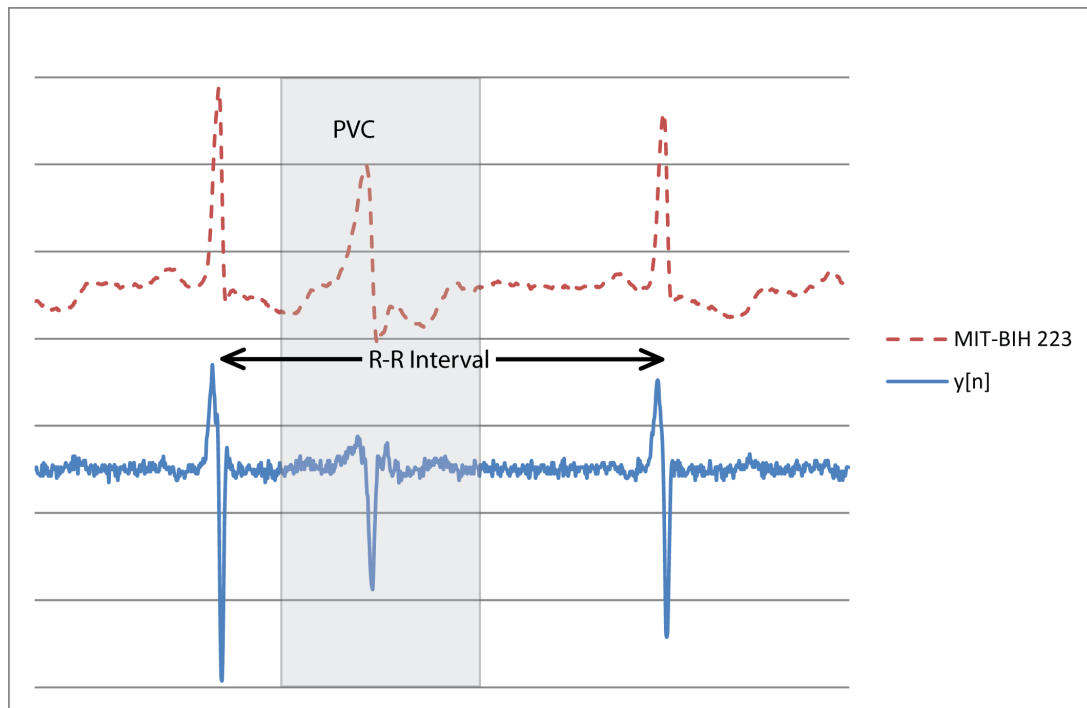


Figure 4.27 PVC and Normal Sinus Rhythms

Storing Results

At the completion of every 1s packet of ECG data processed, the ECG data file is moved from core memory to the SD Card so that in the event of a power loss or removal of the SD card, loss of data is kept to a minimum. The ECG data file includes the raw data samples, R-R intervals, checksum errors (if any) and interesting event notifications.

4.3.5 User Software Drawbacks and Benefit

This section will describe known issues with this user software as well as an additional benefit.

R-peak Threshold

The “R-peak threshold” in the R-peak detection algorithm was determined empirically. However, not all subjects have identical ECG patterns. Therefore this method is vulnerable to the following: missed R-peaks that may have a lower amplitude values at the output of the differentiator $y[n]$, and/or acting on data that is not an R-peak but has an amplitude greater than the triggering threshold. A more suitable method would use a training algorithm to set and adapt the triggering threshold to varying data sets.

Sample Rate

This software uses a sample rate of 4.012036 ms/sample, which is obviously different the sample rate used in the Digital Transmitter software (3.99975 ms/sample). The reason for this is because the microcontroller on the test system that was used to convert the raw MIT-BIH Arrhythmia Database ECG data samples to analog had a slightly faster clock rate than expected. This rate is tunable through software, however the cause of the difference in sample rate was only identified at the time of writing. Details on tuning the internal clock rate of the microcontroller can be found in the datasheet [53].

Removal of Baseline Drift

A result of the algorithm used to identify R-R intervals is its ability to alleviate baseline drift. Baseline drift is caused by variable electrode-skin impedance, which appears as a low frequency variation of the mean of the ECG signal. Since the differentiating filter in Equation (4.11) has a high pass response and the baseline drift is inherently a low frequency signal it is effectively removed from $y[n]$. Figure 4.28 provides an example. Chart a) is an example of an ECG record from the MIT-BIH Arrhythmia Database with a significant baseline drift and chart b) shows the filtered output with the drift effectively removed.

As shown in the previous paragraphs, the final user program is designed to record raw ECG data, detect R-peaks and PVCs and calculate R-R intervals. Each of these functions was added to the software iteratively, demonstrating reconfigurability of the user program. Other aspects of the overall system are also reconfigurable, these aspects are described in the following section.

4.4 System Reconfigurability

Reconfigurability is built-in to several layers of this BAN system and reconfigurability options have varying levels of complexity. The parts of the BAN system that can be reconfigured are listed below in order of increasing complexity; FPGA Server User Software, Network topology, Digital Transmitter Firmware, ECG Mote sensor and FPGA Server hardware. A discussion of each of these follows.

4.4.1 FPGA Server User Software

Modifying the user software on the FPGA Server enables the user to change how the received data is processed. For example, in this research three versions of user software were used to: (1) record raw ECG data; (2) measure and record R-R intervals; and (3) identify PCVs. In other applications the user software could be modified to signal an insulin pump when blood sugar reaches dangerous levels or to

classify movements in a person wearing accelerometers. The key advantage that makes reconfiguration of the user software on the FPGA Server relatively straight-forward is the use of an operating system.

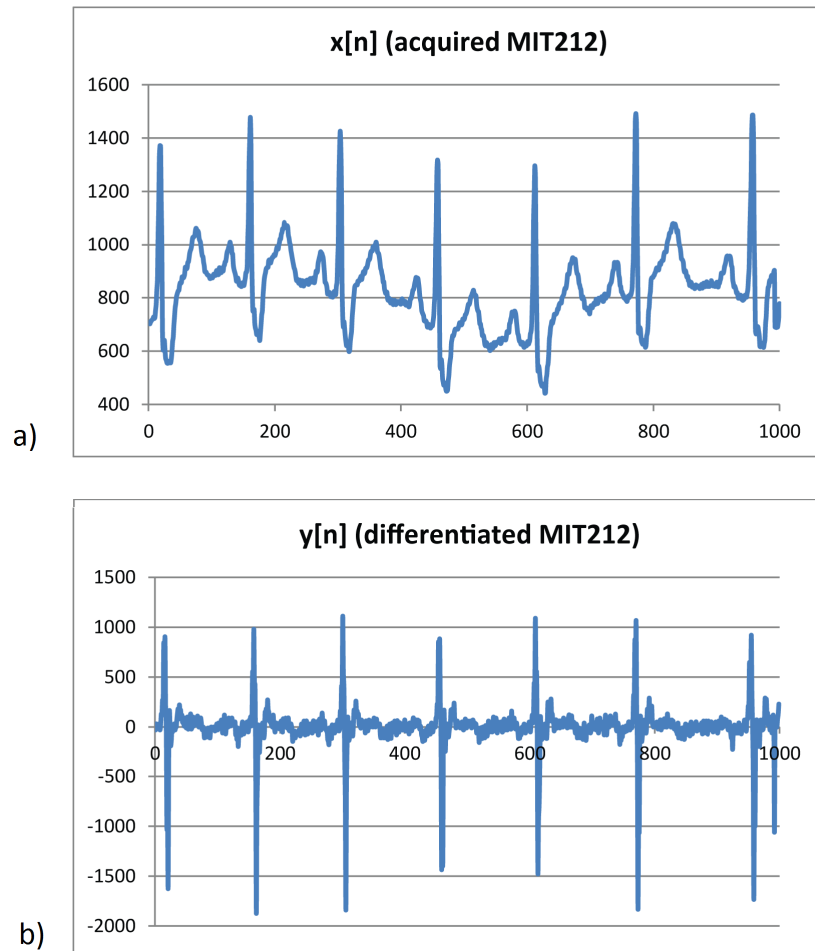


Figure 4.28 Removal of Baseline Drift in $y[n]$

The operating system provides a layer of abstraction so that the user does not need to know all of the system details in order to change the way the system processes the data. In order to modify the user software the user requires: (1) a cross-compiler for compiling C programs that will be launched by a uClinux OS running on a Nios II processor; (2) the ability to program using the C language; and (3) basic knowledge of Linux system calls. Once the user software is compiled the executable file is placed on the FPGA Server SD card and run automatically during the uClinux boot sequence.

4.4.2 Network Topology

In the current design there is one mote and one server. These network nodes are connected in a point-to-point topology and ZigBee is used for RF communications. The addition of motes may demand the use of a mesh network. Migrating to a mesh network would require reconfiguration of the XBee radio and the user software on the FPGA Server. This reconfiguration option requires an additional step compared to reconfiguring user software but remains relatively straight forward.

4.4.3 Digital Transmitter Firmware

Modifying the Digital Transmitter firmware enables: (1) the use of different ports or peripherals on the microcontroller; (2) the use of different types of sensors and/or actuators; (3) changes to sample rates and/or communication rates; and (4) preprocessing of sensor data. Examples of why each of these are a benefit follows.

(1) and (2) are benefits because they allow the system to be reconfigured for other applications. For example, body temperature may be used to identify the onset of infection. There are common integrated circuit temperature sensors that use I2C or SPI to interface to a microcontroller. The current firmware design of the Digital Transmitter does not allow the use of I2C or SPI but can be configured to use these ports.

(3) The ability to adjust the sample rate of an ADC is obviously very important when changing the sensing application. For example the current design uses a sample rate of 250 samples/s. A sensor that measures blood sugar may require a much lower sample rate. Timer settings on the microcontroller can be adjusted to accommodate this.

(4) Allowing for preprocessing of the sensor data adds value in several areas: For example, data compression could be applied to applications that require higher data rates, such as video or audio; advanced error detection and correction algorithms could be added; and interesting events could be identified at the mote, reducing the

transmission bandwidth. These are just a few of the possibilities made available by preprocessing the data.

In order to modify the Digital Transmitter firmware, in-depth knowledge of the microcontroller is required. In addition the designer would require knowledge of C programming and other technical aspects, such as the details of the I2C communication protocol, for example.

4.4.4 ECG Mote Sensor

The ZigBee Mote was made to be modular to allow for different sensor boards to be designed and used with this BAN system. Depending on the design and application, changing the sensor board may or may not also require changes to the Digital Transmitter firmware. Knowledge of sensor processing circuitry as well as Printed Circuit Board design are both required to develop new sensor boards.

4.4.5 FPGA Server Hardware

Allowing for reconfigurability at this level facilitates significant changes to the BAN system. For example, the RF communications technology could be changed to Bluetooth, upgraded to include WiFi or the external storage media could be changed to a USB memory stick.

Reconfiguring the FPGA Server hardware is a complex endeavour but not nearly as difficult as it would be without the use of an operating system. For example, suppose that the removable storage media is to change to USB memory. The following steps are required:

1. Create a modified schematic for the DE0-Nano Daughter Board, including the USB memory.
2. Layout and build the PCB.
3. Populate the PCB.

4. Reconfigure the Nios II processor in SOPC Builder to include a port for the USB memory.
5. Modify the Quartus II project to include the instantiation of the USB memory and pin assignments.
6. Recompile the uClinux distribution after including the drivers for the USB memory port.
7. Modify the uClinux start sequence to mount the USB memory during the boot sequence.

In order to accomplish the above without significant difficulty the designer should have a solid understanding of the overall system, including the software packages required to develop the system. Even then, reconfiguring the FPGA Server hardware is complex. However, if the FPGA Server was designed a built without an operating system this task would be much more involved. USB is a complicated communication standard that requires a significant amount of study to become proficient in. The OS successfully abstracts the details of configuration, read, write and protection of the external memory making this task much easier than without the OS. The same is true for other peripherals such as SPI, I2C and RS232. Appendix A should be used as a guide to modify the FPGA Server hardware.

Practically speaking, all of these reconfigurability options could not be fully demonstrated in this work. Chapter 5 includes a description of the verification results as the user software was reconfigured.

5. Test Results

This chapter begins with a description of the data source that was used to verify the operation and demonstrate reconfigurability of the BAN system. This is followed by a description of the custom built test platform that was used. The chapter concludes with a description of the test results.

5.1 MIT-BIH Arrhythmia Database

PhysioNet [70] was established in 1999 for the purpose of supporting biomedical and physiological research. The project is funded by the National Institute of Biomedical Imaging and Bioengineering as well as the National Institute of General Medical Sciences. This particular database was selected to verify the BAN system because it is well respected in the research community and is the de facto standard for databases of this type, according to Moody et. al. [71].

PhysioNet offers free access to many databases containing data files of physiological signals as well as open source software related to the databases. Relevant to this work is a database called the MIT-BIH Arrhythmia Database. The MIT-BIH Arrhythmia Database contains a large selection of annotated digital electrocardiogram (ECG) data files. These data files are downloadable in a raw sample format from the PhysioBank Automated Teller Machine (ATM) [72].

Each data file downloaded from the PhysioBank ATM contains three columns: (1) the sample number; (2) a column of ECG data acquired using a Modified Limb Lead II (MLII) placement; and (3) a column of ECG data acquired using a V5 lead placement. A MLII placement is such that the electrodes are placed on either side of

the chest vs. a classic limb lead placement where the leads are placed on the wrists and ankles. The MLII placement was used to develop and debug the BAN system prototype. Therefore this column of data was used to verify the operation of the final BAN system. The column of data containing V5 lead placement information was not used in this research.

The raw sample data is recorded as 11 bits per sample, 360 samples per second, taken over a $\pm 5\text{mV}$ range, as identified in the MIT-BIH Arrhythmia Database Directory Introduction [73]. The samples range from 0 to 2047, with 1024 representing 0V. The specific MIT-BIH Arrhythmia Database data files used for verification are 212, 213 and 223. These data files are comprised mainly of normal sinus rhythms but also include premature ventricular contractions (PVCs) and premature atrial contractions (PACs). A fraction of data file 223 is shown in Figure 5.1 for example.

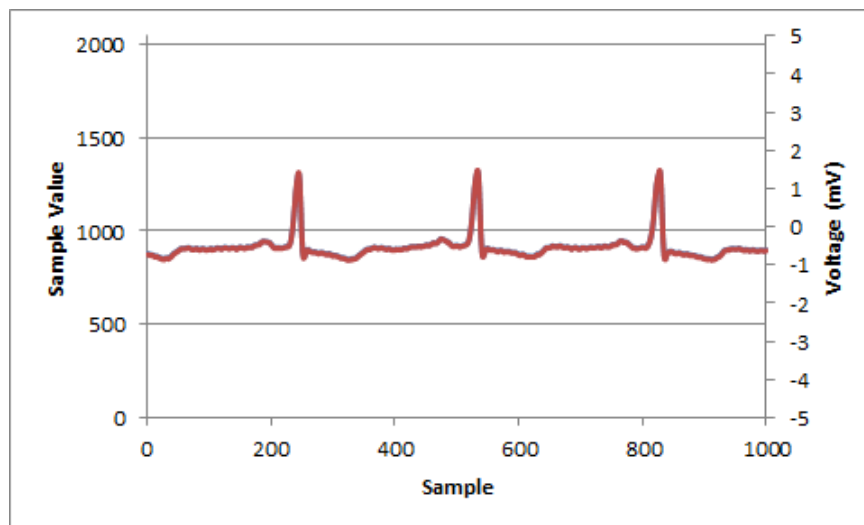


Figure 5.1 Typical MIT-BIH Arrhythmia Database Data file

5.2 Test Platform

This section will describe the design details of the test platform used to verify the operation of the BAN system beginning with a high level block diagram.

5.2.1 Block Diagram

Three options were considered prior to selecting how the MIT-BIH Arrhythmia Database data files would be injected into the BAN system.

1. Place raw samples directly onto the FPGA Server SD card. The implementation of this test platform would have been relatively straight forward, however this setup would have only tested the software on the FPGA Server and not the Digital Transmitter or RF communications.
2. Place the MIT-BIH Arrhythmia Database samples into the microcontroller memory on the Digital Transmitter. This is a better option because it would have included RF communications in the tests but it still would not have included all aspects of the Digital Transmitter.
3. Build a hardware test platform that will convert the MIT-BIH Arrhythmia Database data files to analog and connect the analog signal directly to the Digital Transmitter ADC. In this scenario only the ECG AFE is omitted from the system, as shown in Figure 5.2. This option tests as much of the system as possible without using a human subject.

With the goal being to verify the operation of the entire system, option 3 was selected.

As shown in Figure 5.2, the test platform DAC connects directly to the Digital Transmitter. The Digital Transmitter converts the analog signal to digital, packetizes a group of digital samples and then sends the packets to the FPGA Server. The FPGA Server processes the packets in order to identify interesting events and store relevant ECG data on an SD card. Once the data is on the SD card it can be compared with the original MIT-BIH Arrhythmia Database.

5.2.2 Hardware

The test platform hardware employs a Microchip PICDEM 2 PLUS DEMO BOARD, shown in Figure 5.3. The board is populated with a PIC18F46K22 microcontroller.

The PIC18F46K22 microcontroller was selected due to its relatively large program memory (64KB). The MIT-BIH Arrhythmia Database samples are programmed into the microcontroller program memory during testing. A MCP4922 12-bit Digital Analog Converter (DAC) with Serial Peripheral Interface (SPI) is added to the prototyping area. Figure 5.4 contains the schematic diagram of the test platform.

Once the test platform hardware was complete the microcontroller needed to be programmed to move the samples from the program memory to the DAC.

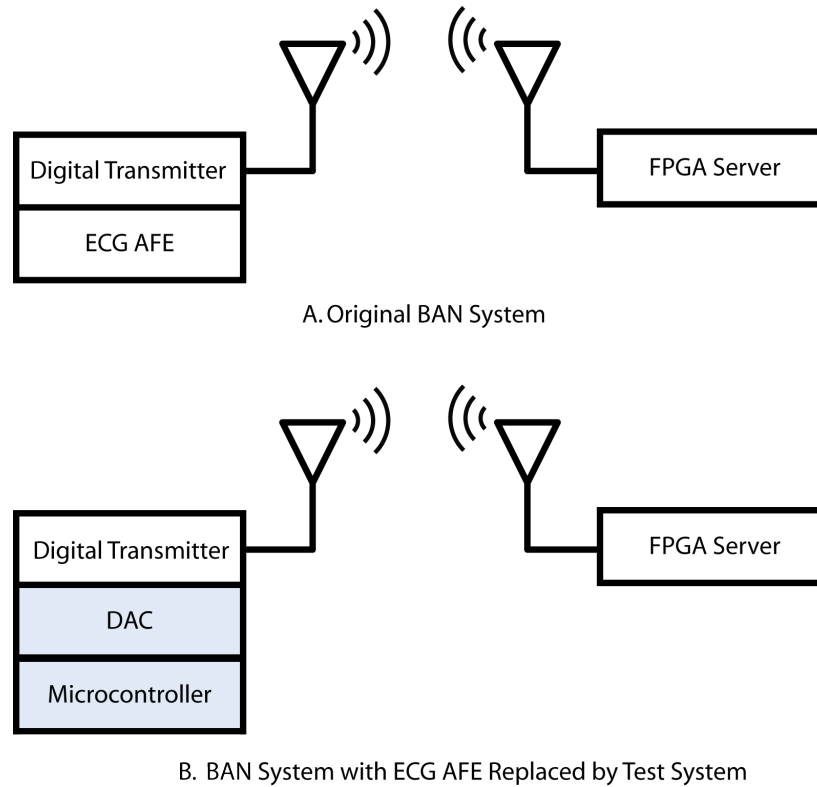


Figure 5.2 Test System Block Diagram

5.2.3 Software

The test platform software was developed using MPLAB IDE v8.80, written in C and compiled using the MPLAB C18 C compiler v3.40. Appendix G.2 provides a listing of the C programs used by the test platform microcontroller.

The purpose of the test software is twofold: (1) adjust the amplitude and offset

of the digital MIT-BIH Arrhythmia Database samples (the details of which are included in Section 5.2.4); and (2) send the adjusted values to a DAC via SPI. To do this the MIT-BIH Arrhythmia Database samples are placed in a large array in the microcontroller program memory. One at a time the samples are sent to the DAC via SPI. Once all the samples have been sent, the process starts over.

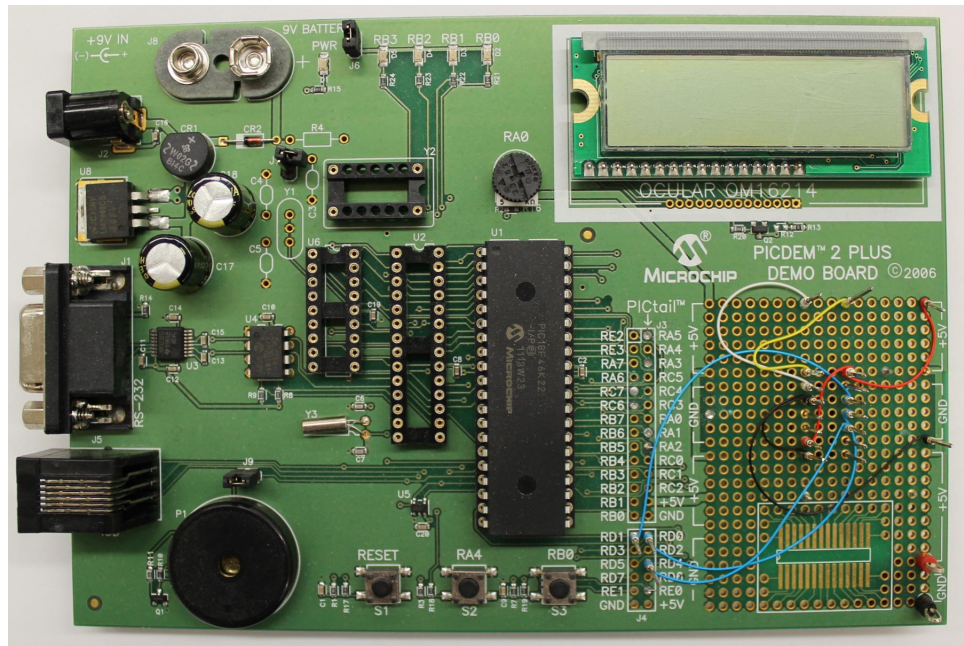
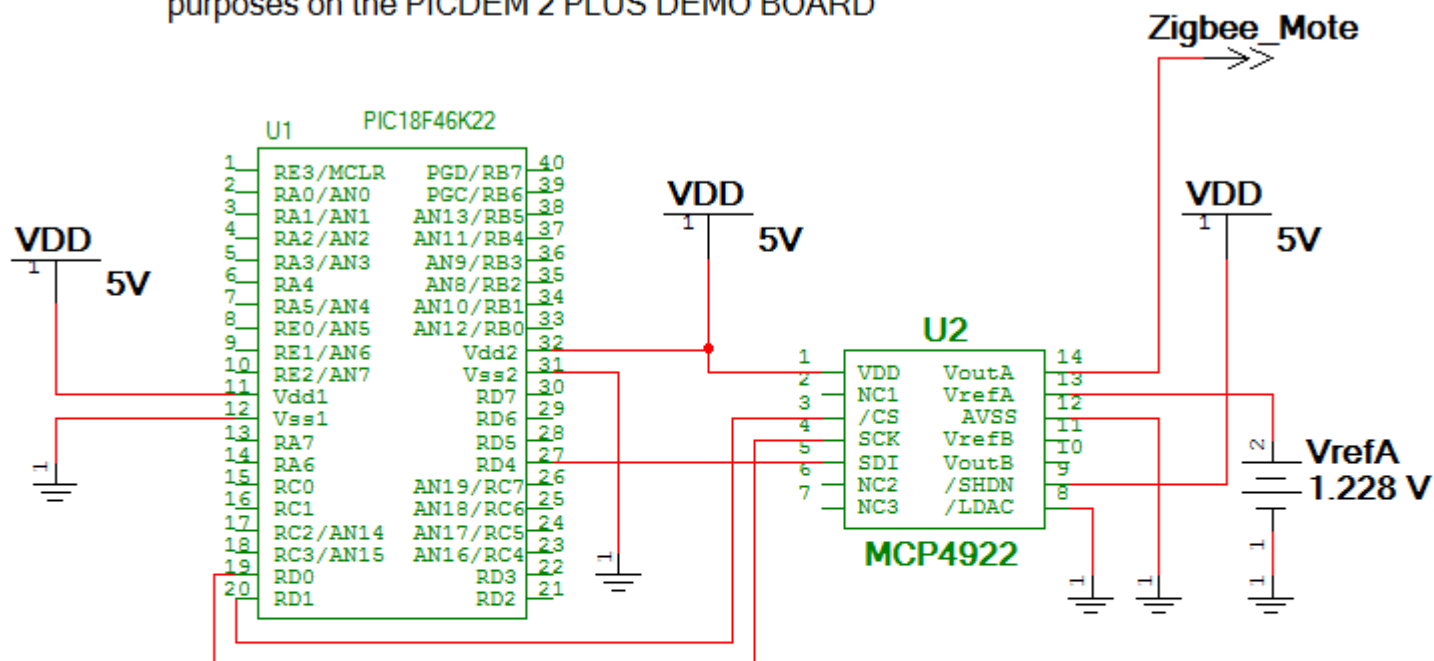


Figure 5.3 PICDEM 2 PLUS DEMO BOARD

Referring to the flow chart in Figure 5.5, the program begins with the declaration and initialization of a relatively large array called “ecg_data”. The initialization of the array uses the “rom” command to store the array to program memory rather than data memory or EEPROM. Using the program memory allows for larger sample sizes to be saved (i.e., the program memory in the PIC18F46K22 is 32KB, the data memory is less than 4kB and the EEPROM is only 1KB). Unfortunately the format of the data downloaded from the MIT-BIH Arrhythmia Database is not the same as that used by “ecg_data” so a C program was used to manipulate the data, further details are included in Section 5.2.5. After “ecg_data” is created the required ports and peripherals are initialized prior to starting the main loop.

Note: Unused pins on PIC18F46K22 are used for other purposes on the PICDEM 2 PLUS DEMO BOARD



Title: MIT_TEST_PLATFORM		
MIT_TEST_PLATFORM		
Designed by: A. Voykin	Document N: 0001	Revision: 1.0
Checked by:	Date: 2012-11-14	Size: A
Approved by:	Sheet 1 of 1	

Figure 5.4 Test Platform Schematic

The main part of the program consists of an infinite loop that waits for a sample timer to expire. The sample timer is set to expire at the sample rate of the MIT-BIH Arrhythmia Database data, 360 times per second. When the sample timer expires the program sends a word corresponding to one ECG sample to the DAC. Prior to sending a word to the DAC, the amplitude and offset of the MIT-BIH Arrhythmia Database samples are adjusted.

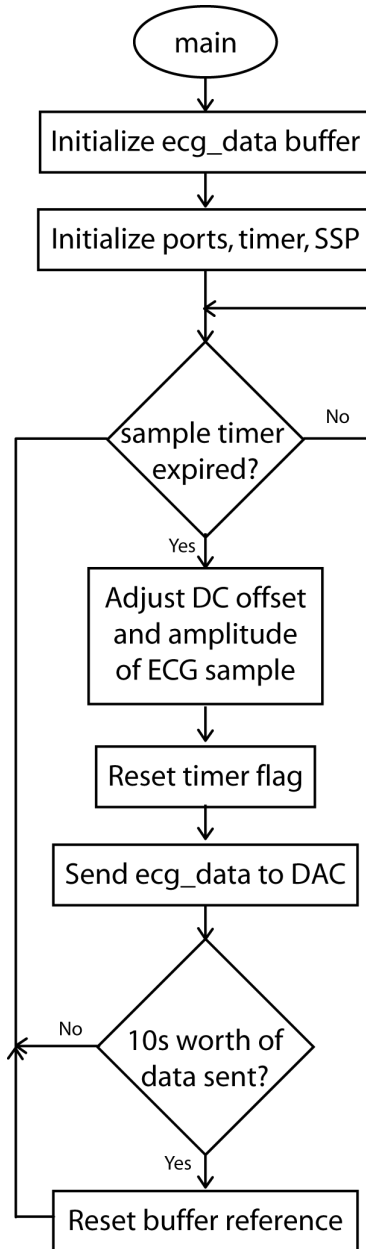


Figure 5.5 Test Platform Software Flow Chart

Once the main loop has sent all samples from the `ecg_data` array the loop starts over. This continues until the power to the test platform is removed.

As mentioned at the beginning of the software description the sample require amplitude and offset adjustment to work with this system. This is clarified in the following section.

5.2.4 Adjustment of the MIT-BIH Database Samples

In order to connect the analog output generated by the test platform to the Digital Transmitter ADC the voltage must have approximately the same amplitude and DC offset that would be generated from a human subject using the ECG AFE. To achieve this the MIT-BIH Arrhythmia Database samples required amplitude and DC offset adjustments. Amplitude adjustments are made for two reasons: (1) the MIT-BIH samples do not span the entire 11b range used to acquire them; and (2) the MIT-BIH samples consist of 11b and the test system DAC is a 12b device. The DC offset required adjustment because the MIT-BIH samples span 0–2048 and 1024 to represent 0V where the test system DAC uses decimal 0 to represent 0V.

To determine the appropriate adjustments, a baseline was required using a human subject and the ECG AFE. The primary researcher was used to generate the baseline. Figure 5.6 provides an example of the baseline measurement at the output of the ECG AFE. In order for the MIT-BIH Arrhythmia Database samples to match this baseline, each sample is adjusted by subtracting 600 to remove offset and then multiplying by 3.25.

Following the adjustments made to the MIT-BIH Arrhythmia Database samples by the test platform software, the test platform output and baseline had very similar voltage characteristics. An example of this using MIT-BIH Arrhythmia Database data file 213 is shown in Figure 5.7.

The method used to match the test system output with the baseline adds a small amount of noise to the DAC output. The reason is because in the MIT-BIH samples

are multiplied by 3.25. In order to do this the samples must be cast to a float. The result of the multiplication is recast to an int and any fractional data is truncated. A brief analysis of the resulting noise added to data file 223 is included in Appendix I.

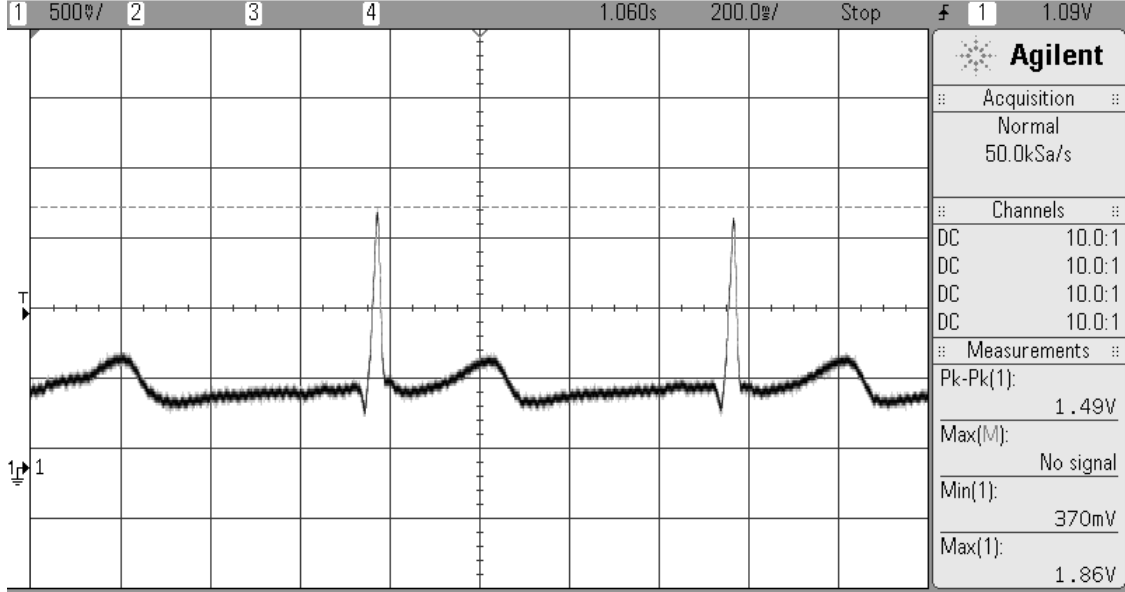


Figure 5.6 Baseline Generated Using Human Subject and ECG AFE

5.2.5 Preparation of the Sample Buffer

As already mentioned, the MIT-BIH Arrhythmia Database samples are 11-bits wide. Therefore, in the test program each sample consumes 2B of memory. The test program, excluding the the MIT-BIH samples, consumes approximately 1KB of program memory. The program memory in the PIC18F46k22 is 32KB. Therefore, at 360 samples per second, the program memory in the selected microcontroller is theoretically capable of storing over 43 seconds worth of MIT-BIH Arrhythmia Database samples, according to Equation (5.1)

$$\text{PIC18F46K22 storage (s)} = (32\text{KB}-1\text{KB}) * \frac{1\text{s}}{360 \text{ samples}} * \frac{1 \text{ sample}}{2\text{B}}. \quad (5.1)$$

The tests in this work were limited to 10s in duration for practical purposes.

The MIT-BIH Arrhythmia Database samples are downloaded as a text file. The file is delimited with spaces between the columns. Prior to placing the MIT-BIH

Arrhythmia Database samples into the microcontroller C project, the data required formatting. Each test iteration contained 10s worth of data, which equates to 3600 samples. Figure 5.8 shows an example of the downloaded MIT-BIH Arrhythmia Database text file and the format of the data used by the microcontroller. Since several test iterations were planned, manual formatting of the data was not practical. Appendix G.1 provides a C program used to automatically adjust the MIT-BIH Arrhythmia Database data files into the format shown in Figure 5.8.

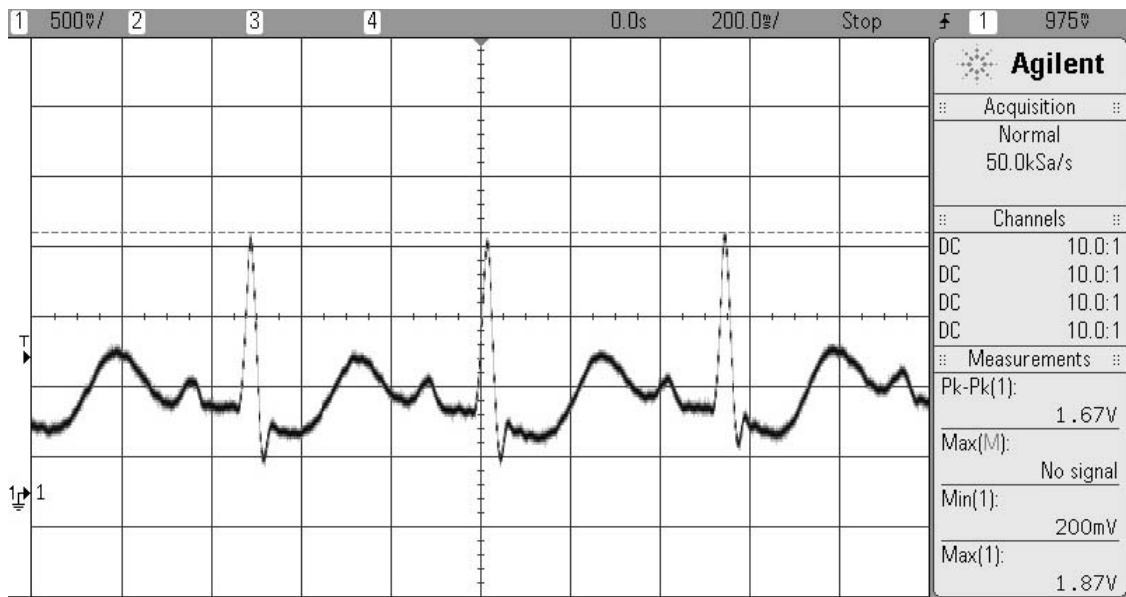


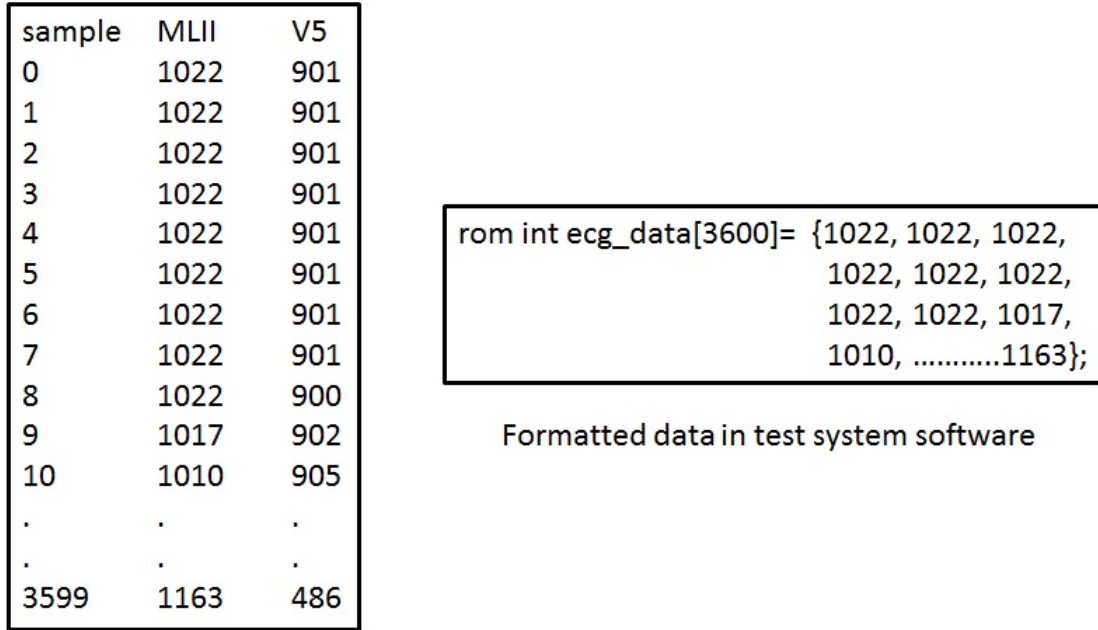
Figure 5.7 MIT-BIH Arrhythmia Database 213 Output From Test Platform

5.3 Test Results

The following section will show that the BAN system can acquire raw ECG data, identify R peaks, PVCs and PACs and calculate R-R intervals.

5.3.1 Acquired ECG Data

In order to illustrate that the BAN system is capable of acquiring raw ECG data an example is shown in Figure 5.9. The figure compares raw MIT-BIH Arrhythmia Database samples, Figure 5.9 (a), to the acquired data pulled directly from the ECG data file saved by the user software on the FPGA Server, Figure 5.9 (b).



Formatted data in test system software

MIT-BIH (213)

Figure 5.8 MIT-BIH Arrhythmia Database vs. Data in Test System Software

Of course the amplitude and offset cannot be compared between the waveforms due to the different systems used to acquire the samples, however the features of the waveforms are nearly identical. This chart is provided to show that the BAN system is capable of acquiring a complete, single lead ECG waveform. The following section will discuss the identification of R-peaks and calculation of R-R intervals.

5.3.2 R-R Intervals

The first 10s of data files 212, 213 and 223 from the MIT-BIH Arrhythmia Database were used to verify the system operation when calculating R-R intervals. Each of the data files was tested twice, resulting in 6 distinct tests. In order to validate the results, the difference between the MIT-BIH Arrhythmia Database R-R intervals and the acquired R-R intervals was calculated. The MIT-BIH Arrhythmia Database R-R intervals can be downloaded using the PhysioBank ATM [72].

The BAN system produced favourable results in detecting R peaks and calculating R-R intervals for all MIT-BIH Arrhythmia Database data files tested. Over 85% of

the acquired R-R intervals were within $\pm 3\text{ms}$ of the MIT-BIH Arrhythmia Database R-R intervals. Table 5.1 provides the data for one test iteration of each of the three data files, 212, 213 and 223. This table shows that the majority of the acquired R-R intervals are within 3ms of the published intervals from the MIT-BIH Arrhythmia Database. The worst case error in the calculated R-R intervals was 6ms and occurred using data file 213.

Following verification the software was reconfigured to detect interesting cardiac events.

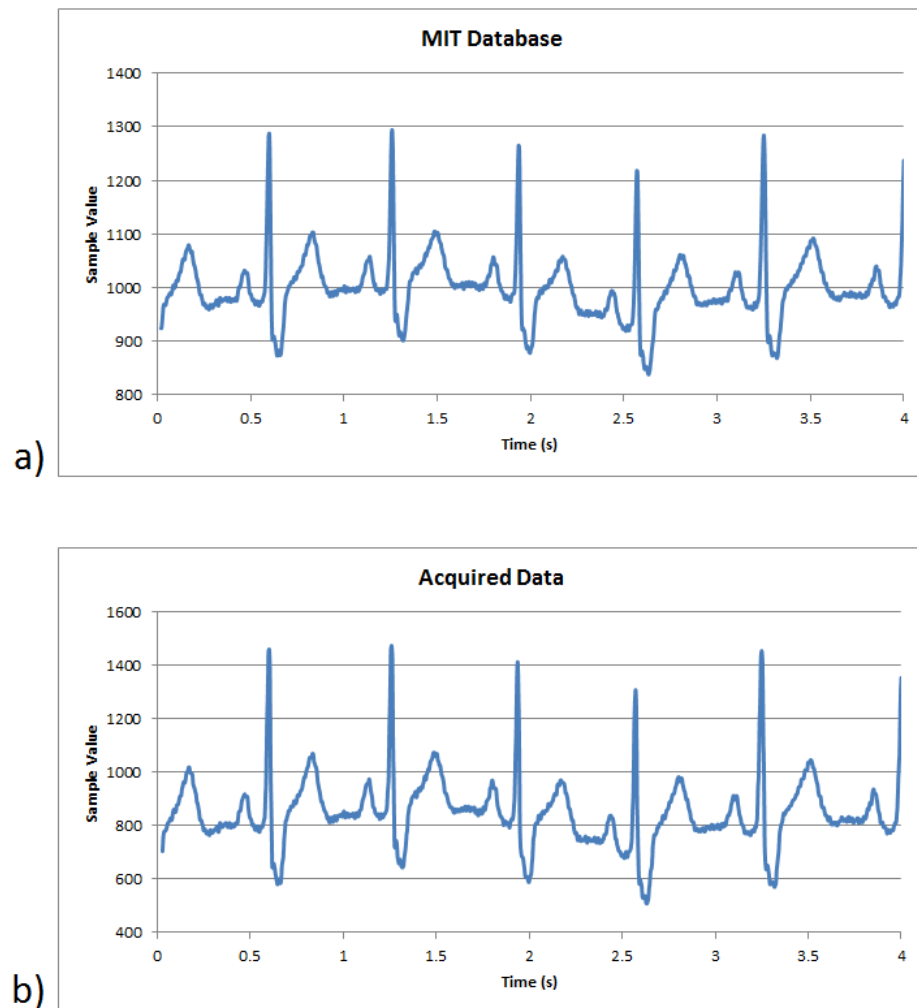


Figure 5.9 MIT212 Acquired vs Raw Samples

Table 5.1 R-R Interval Test Data

MIT-BIH Database RR Intervals (s)			Acquired Results RR Intervals (s)			MIT RR Int. -Acq. RR Int. Interval (ms)		
212	213	223	212	213	223	212	213	223
0.658	0.439	0.767	0.658	0.437	0.77	0	2	-3
0.681	0.625	0.775	0.678	0.626	0.774	3	-1	1
0.633	0.542	0.764	0.634	0.542	0.762	-1	0	2
0.678	0.556	0.756	0.678	0.554	0.758	0	2	-2
0.756	0.564	0.744	0.754	0.562	0.746	2	2	-2
0.742	0.544	0.781	0.742	0.55	0.782	0	-6	-1
0.683	0.539	0.75	0.682	0.538	0.746	1	1	4
0.661	0.525	0.717	0.658	0.522	0.718	3	3	-1
0.572	0.539	0.731	0.574	0.542	0.734	-2	-3	-3
0.575	0.55	0.728	0.574	0.546	0.73	1	4	-2
0.617	0.525	0.742	0.618	0.53	0.742	-1	-5	0
0.619	0.547	0.747	0.618	0.546	0.75	1	1	-3
0.642	0.553		0.642	0.55		0	3	
0.744	0.556		0.746	0.558		-2	-2	
	0.525			0.526			-1	
	0.55			0.55			0	
	0.525			0.526			-1	

5.3.3 Interesting Event Detection

In order to test the reconfigurability of the FPGA Server user software the algorithms were modified so that PVCs could be detected. MIT-BIH Arrhythmia Database 223 from 1:00 minute to 1:20 seconds was used. In this data file there are 2 PVCs and 1 PAC, as shown in Figure 5.10. The BAN system successfully classified each of the PVCs as interesting events and all of the normal sinus rhythms

were ignored. Due to the characteristics of PACs as discussed in Chapter 2, the same algorithm used to identify PVCs also identified the lone PAC in the data files tested. However, PACs could be excluded by increasing the ratio of adjacent R-R intervals that triggers the identification of a PVC.

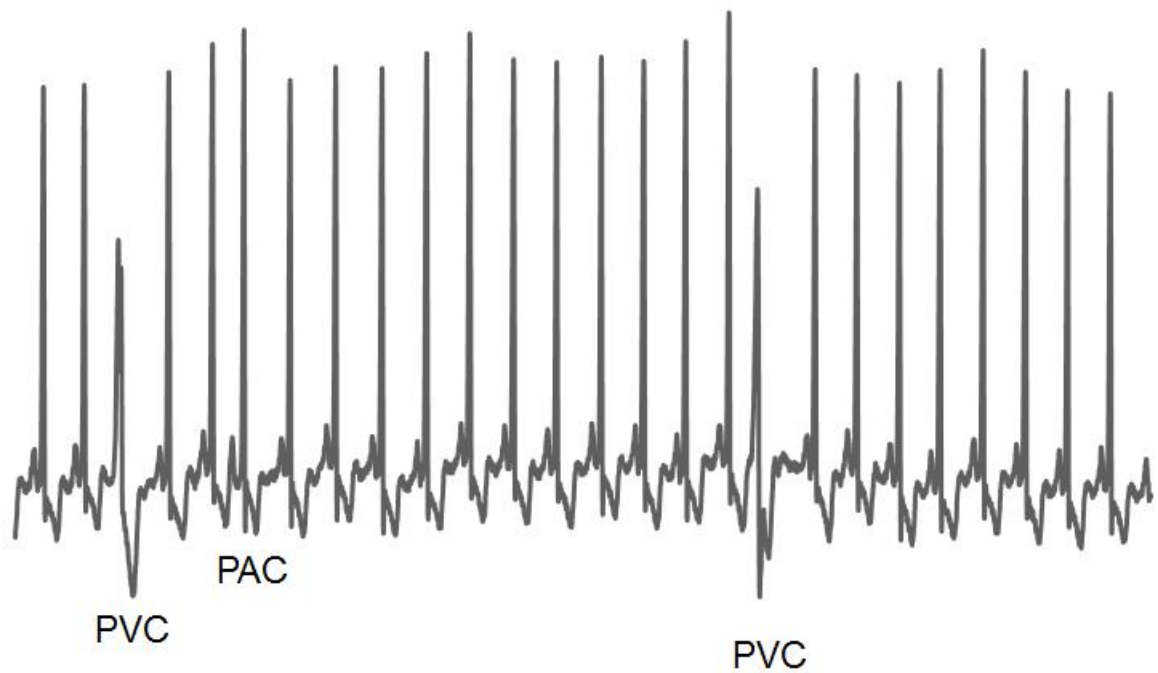


Figure 5.10 MIT223 Interesting Events 1:00 to 1:20

The following chapter will summarize the previous chapters, provide opportunities for improvement of the current system and conclude this research.

6. Summary and Contributions

The following paragraphs will discuss how each of the objectives identified in Chapter 1 were accomplished. The chapter will close with the research contributions provided by this work.

To recap, the objectives set in Chapter 1 are as follows:

1. Review the current state of Body Area Network (BAN) research through a literature review and identify opportunities to contribute to this research.
2. Determine the most appropriate wireless protocol and network, hardware and software architectures to use in a reconfigurable BAN.
3. Identify a suitable application to demonstrate a reconfigurable BAN system.
4. Verify the BAN system operation using a standardized data source.
5. Demonstrate reconfigurability of the final BAN system.

6.1 Summary

Following the introductory chapter, the electromechanical operation of the heart is discussed in Chapter 2. This chapter describes how blood moves throughout the body when the heart beats due to electrical stimulus. These electrical impulses can be viewed and analyzed with the aid of an electrocardiogram (ECG). The fiducial points of an ECG correspond to the stages of a heart beat. Of particular importance to this work is the fiducial point called the R-peak. The R-peaks can be used to calculate heart rate and identify abnormal waveforms within an ECG. Following

this description, Chapter 3 reviews the current state of Body Area Network (BAN) research.

In Chapter 3 typical nodes in a BAN are defined as master and slave. The master collects and processes the data from the slave nodes. The slave nodes acquire and transmit sensor data and depending on the application, may also process the data prior to transmission. Chapter 3 also identified several challenges associated with designing BAN systems. These challenges include power consumption, size, weight, ease of use, cost, privacy, security and flexibility. All of these challenges have been considered in current BAN system research but limited attention has been given to designing flexible systems and therefore, this was identified as an area that would benefit from further study.

This work has focused on designing a highly reconfigurable system that will accommodate different BAN applications, architectures and hardware. Challenges that were identified as being especially relevant to the design of a reconfigurable BAN system include: (1) planning the network architecture; (2) selecting the most appropriate hardware; (3) planning the software architecture; and (4) selecting the wireless protocol. Each of the following 4 paragraphs respectively summarize these challenges.

(1) Planning of the network architecture was approached with the goal of determining the most appropriate location in the BAN to store and process the sensor data. As noted in Chapter 3, current research includes many different architectures. Building on this knowledge, this work identifies the slave nodes as simple devices responsible for data acquisition and forwarding, with very little data processing. The master node processes the data to identify fiducial points and interesting events and to store the results.

(2) In the selection of the core the hardware, microcontrollers and FPGAs were considered as the most appropriate options. In the final design an FPGA running a soft core microprocessor was selected for the master node and a microcontroller was selected for the slave node. In addition, to make overall hardware used on the nodes

highly reconfigurable, both master and slave have modular functionality.

(3) In the study of the software architecture 2 approaches were investigated: (1) operating system (OS); and (2) custom firmware. An operating system was identified as a very useful method to abstract the underlying hardware details from the high level programmer. This resulted a platform that allows straight forward reconfiguration of how the sensor data is processed. Based on this, an operating system was used on the master node. On the slave node the custom firmware approach was selected for the following reasons: the slave node hardware does not require relatively complex drivers; data processing on the slave node is intended to be limited and not highly reconfigurable; low power sleep modes are straight forward to implement without the overhead of an OS; the additional services offered by the OS are not required on the slave node.

(4) Finally, many wireless protocols were investigated with two being identified as viable options for a current implementation of a BAN system; Bluetooth and ZigBee. ZigBee was selected as the most suitable option because it meets the requirements of BANs in that it is efficient, secure, has a moderate data rate and is adaptable to varying network topologies. In addition, the engineering effort required to configure a ZigBee network is relatively low.

Once a suitable architecture for a reconfigurable BAN system was identified, a prototype system was designed and built. Chapter 4 describes the system designed in this research in full detail. In general, this system consists of a master node and a slave node. The slave node is called the ZigBee Mote and the master node is called the Field Programmable Gate Array (FPGA) Server. The ZigBee Mote acquires electrocardiogram (ECG) data and transmits it to the FPGA Server for processing. The FPGA Server processes the received data and stores both raw and processed data to external memory. Highlights of each nodes are briefly discussed in the following paragraph.

The ZigBee Mote uses a Microchip microcontroller programmed in C. The FPGA

Server uses an Altera Field Programmable Gate Array (FPGA) configured as a NIOS II processor. The NIOS II processor is programmed to run uClinux. Data received from the ZigBee Mote(s) is stored to an SD card residing on the FPGA Server. User programs that determine how the sensor data will be processed are also stored on the SD card. These programs are written in C and automatically run as part of the uClinux boot sequence. Based on the design challenges and decisions discussed in Chapters 3, the overall system has many layers of reconfigurability. These are discussed at the end of Chapter 4. Once the BAN system prototype was designed and built, a test system was required to verify its operation.

Chapter 5 discusses the test system used as well as the test results. The purpose of the test system was to allow known data to be injected into the BAN system. The MIT-BIH Arrhythmia Database offered by PhysioNet was used as the known data source. In order to get the MIT-BIH Arrhythmia Database files into the BAN system the test system was designed to replace the sensor and associated electronics on the ZigBee Mote. Raw ECG sample data was converted to analog by the test system and the analog signal was connected to the ADC on the ZigBee Mote. The ZigBee Mote converted the signal back to digital and sent the samples to the FPGA Server for processing. Once the test system was functional the BAN system could be verified and reconfigurability could be demonstrated.

Three different test scenarios were used to verify the BAN system operation and demonstrate reconfigurability of the user software. In the first scenario the ZigBee Mote acquired ECG data from the test system and transmitted it to the FPGA Server. The FPGA Server received the data and stored it to external memory. The external memory was removed from the BAN system and accessed with a personal computer. The sample data was plotted and visually inspected. In order to verify the validity of the data the next test scenario reconfigured the user software on the FPGA server to detect R-peaks and calculate R-R intervals. Of the 3 different MIT-BIH Arrhythmia Database data files tested 100% of the R-peaks were identified by the FPGA Server and 85% of R-R intervals calculated were within 3ms of the intervals published in

the MIT-BIH Arrhythmia Database. In the final test scenario the FPGA Server user software was reconfigured to identify premature ventricular contractions.

The following section will describe the topics that are considered to be significant contributions of this research.

6.2 Contributions

The contributions that this work has made to BAN research include:

1. A prototype BAN system capable of digitizing an electrocardiogram (ECG) signal, wirelessly transmitting the digitized data to a remote server, saving the data on a removable storage device and processing the data to detect fiducial points as well as abnormal beats.
2. Complete design details for a flexible BAN system that facilitates reconfiguration of data processing algorithms, system hardware and network topology. The design details include: block diagrams, schematics, bill of materials, printed circuit board layouts, configuration details, software and flowcharts.
3. A step by step guide for compiling, configuring and programming an FPGA based system to automatically run user software via uClinux on the Nios II processor. Appendix A contains all of the details required to: (1) create a Linux development system on a PC, including the operating system, uClinux distribution and Quartus II Electronic Design Automation software; (2) configure the Nios II processor; (3) configure and compile the uClinux kernel (4) program a serial flash device to automatically configure the FPGA and load the uClinux kernel image into SDRAM when the power is turned on; (5) automatically run user programs as part of the uClinux boot sequence; and (6) cross compile user programs to be run by uClinux on a Nios II processor. This is a complicated procedure that will save time for designers and researchers seeking a similar outcome.

7. Future Work and Conclusions

This chapter will close the main body of the thesis with a discussion of opportunities for future work and the conclusions identified as a result of this research.

7.1 Future Work

One of the major shortcomings with this BAN system is its inability to communicate beyond the BAN, which does not allow escalation to a health care professional. Future work includes the investigation and addition of wireless communication hardware that will enable information to be shared beyond the BAN.

Other aspects of the current design that require further study are: compression of the ECG data in order to increase the efficiency of wireless data transfer, hence reduce power consumption; methods and modifications required to increase the BAUD rate between processors and XBee radios, also reducing power consumption; and finally, an investigation of the limitations of the processing capabilities of the FPGA Server.

Throughout the research, design and verification of this BAN system several key findings were identified, these are described in the following conclusions.

7.2 Conclusions

In order for a BAN system to effectively satisfy the wide range of potential applications it must be designed to be highly reconfigurable. Design and verification of a reconfigurable BAN is a complex undertaking that requires considerable planning and attention to detail. In order to realize a reconfigurable BAN system, this thesis

shows that master and slave nodes have significantly different design requirements and hence these two nodes should not be considered equal.

A suitable master node is identified as requiring advanced processing capabilities and removable storage. Based on these requirements an FPGA configured as a soft core processor, running an operating system (OS), provides many benefits to the overall BAN. These include: efficient data processing, hardware flexibility, a migration path to an Application Specific Integrated Circuit (ASIC) and the ability to relocate critical software functions into hardware. In addition, the use of an OS abstracts the complexities of the system hardware from the programmer, which allows for rapid reconfiguration of the programs that process the BAN sensor data. The OS also provides access to services unavailable to a system programmed using custom firmware drivers and peripherals.

Centralizing storage of sensor data on the master node also provides many benefits to the overall system. Data fusion is easier because all of the sensor data is contained in one location. The overall BAN hardware model is simplified because not all nodes require external storage. The slave node firmware is not complicated with drivers for external memory devices and the slave nodes do not require extra power to operate external memory devices.

A suitable slave node is intended to be very efficient and have the ability to quickly and easily adapt to new applications. Based on these requirements, a modular system with the ability to exchange sensor hardware without redesigning the entire system and a microcontroller that orchestrates the acquisition and transmission of sensor data provides many benefits. These include: low power sleep features, adaptation to different sensor modules as well as straight forward changes to peripheral modules and processing software. Together the master and slave nodes form a highly reconfigurable BAN.

In order to realize the true value of the system designed in this work many diverse applications require experimentation. For example, a direct opportunity exists in

the analysis of heart rate variability (HRV). A HRV test measures changes in R-R intervals and uses the acquired information to assess heart health. HRV testing is being researched in many health care applications [74] [75]. Another application well suited to exercise the reconfigurability of this system is biometric recognition for the purpose of data protection. Bui [76] proposes the use of ECG signals to implement security within the BAN. Each of these applications exploit the ECG electronics included in this system. What are the benefits gained by reconfiguring the system to use other sensors?

A BAN that includes sensors monitoring respiration rate, body temperature, spirometry, and blood pressure, could be used to detect declining health in patients with chronic obstructive pulmonary disease or transplanted lungs. A system with similar sensors and the ability to upload data to a health care network could be used to monitor a large population for influenza outbreaks. Of course, this would require widespread use of BANs.

In order for widespread use to become a reality BAN research needs to continue in areas specifically focused on power consumption as well as the physical size of the BAN nodes. Scientists at Seoul University [77] are working toward making BAN systems much more comfortable, promoting widespread and long-term use. This group has developed pressure sensitive electronic skin that theoretically could be attached to the wrist to measure heart rate.

A highly reconfigurable BAN system will satisfy the requirements of all of the applications mentioned and of course others. In the near future reconfigurable BANs will revolutionize healthcare, sport, entertainment, safety and more.

References

- [1] “Human Heart.” [http://commons.wikimedia.org/wiki/File:Diagram_of_the_human_heart_\(cropped\).svg#filelinks](http://commons.wikimedia.org/wiki/File:Diagram_of_the_human_heart_(cropped).svg#filelinks). Accessed: Mar., 2013.
- [2] “Conduction System of the Heart.” <http://upload.wikimedia.org/wikipedia/commons/d/d5/Conductionsystemoftheheart.png>. Accessed: Mar., 2013.
- [3] A. Atkielski, “Normal ECG Signal.” <http://en.wikipedia.org/wiki/File:SinusRhythmLabels.svg>. Accessed: Aug., 2012.
- [4] “Your Heart’s Electrical System.” <http://www.bostonscientific.com/lifebeat-online/heart-smart/electrical-system.html>. Accessed: Mar. 2012.
- [5] E. Company-Bosch and E. Hartmann, “ECG Front End Design is Simplified with MicroConverter,” *Analog Dialogue*, vol. 37, pp. 1–5, Nov. 2003.
- [6] Karthik Soundarapandian and Mark Berarducci, “Analog Front-End Design for ECG Systems Using Delta-Sigma ADCs,” *Application Report SBAA160A*, Apr. 2010.
- [7] Ajay Bharadwaj and Umanath Kamath, “Accurate ECG Signal Processing,” *EE Times Design*, Feb. 2011.
- [8] R. E. Klabunde, “Cardiovascular Physiology Concepts.” <http://www.cvphysiology.com/Arrhythmias/A009.htm>. Accessed: Mar. 2012.
- [9] J. E. Keany, D. F. Brown, and A. D. Desai, “Premature Ventricular Contraction.” <http://emedicine.medscape.com/article/761148-overview>. Accessed: Mar. 2012.
- [10] H. Zlotnik, D. Bloom, and E. Jimenez, “Seven Billion and Growing: The Role of Population Policy in Achieving Sustainability,” *United Nations Population Division*, no. 2011/3, p. 9 and 25, 2011.

- [11] V. Auteri, L. Roffia, and T. Cinotti, “ZigBee-based wireless ECG monitor,” in *Computers in Cardiology*, pp. 133 –136, Oct. 2007.
- [12] R. Fensli, E. Gunnarson, and T. Gundersen, “A wearable ECG-recording system for continuous arrhythmia monitoring in a wireless tele-home-care situation,” in *18th IEEE Symposium on Computer-Based Medical Systems*, pp. 407 – 412, June 2005.
- [13] Y. Kim and I. yeon Cho, “Wearable ECG Monitor: Evaluation and Experimental Analysis,” in *International Conference on Information Science and Applications (ICISA)*, pp. 1 –5, April 2011.
- [14] S.-H. Toh, S.-C. Lee, and W.-Y. Chung, “WSN Based Personal Mobile Physiological Monitoring and Management System for Chronic Disease,” in *Third International Conference on Convergence and Hybrid Information Technology (ICCIT)*, vol. 1, pp. 467 –472, Nov. 2008.
- [15] K. Janani, V. Dhulipala, and R. Chandrasekaran, “A WSN Based Framework for Human Health Monitoring,” in *International Conference on Devices and Communications (ICDeCom)*, pp. 1 –5, Feb. 2011.
- [16] F. Adochiei, C. Rotariu, R. Ciobotariu, and H. Costin, “A wireless low-power pulse oximetry system for patient telemonitoring,” in *7th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, pp. 1 –4, May 2011.
- [17] A. Pantelopoulos and N. Bourbakis, “A survey on wearable sensor-based systems for health monitoring and prognosis,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 40, pp. 1 –12, Jan. 2010.
- [18] M. Chen, S. Gonzalez, A. Vasilakos, H. Cao, and V. C. Leung, “Body Area Networks: A Survey,” *Mobile Network Applications*, vol. 16, pp. 171–193, Apr. 2011.

- [19] J. Ko, C. Lu, M. Srivastava, J. Stankovic, A. Terzis, and M. Welsh, “Wireless sensor networks for healthcare,” *Proceedings of the IEEE*, vol. 98, pp. 1947–1960, Nov. 2010.
- [20] T. J. Dishongh and M. McGrath, *Wireless Sensor Networks for Healthcare Applications*. Artec House, 2011.
- [21] S. Patel, K. Lorincz, R. Hughes, N. Huggins, J. Growdon, D. standaert, J. Dy, M. Welsh, and P. Bonato, “A body sensor network to monitor Parkinsonian symptoms: extracting features on the nodes,” *5th International Workshop on Wearable Micro and Nanosystems for Personalised Health*, pp. 21–23, May 2008.
- [22] A. Burns, B. Greene, M. McGrath, T. O’Shea, B. Kuris, S. Ayer, F. Stroiescu, and V. Cionca, “SHIMMER: A Wireless Sensor Platform for Noninvasive Biomedical Research,” *IEEE Sensors Journal*, vol. 10, pp. 1527–1534, Sept. 2010.
- [23] A. Casson and E. Rodriguez-Villegas, “Signal agnostic compressive sensing for Body Area Networks: Comparison of signal reconstructions,” in *Annual International Conference of the IEEE on Engineering in Medicine and Biology Society (EMBC), 2012*, pp. 4497–4500, 2012.
- [24] R. McSweeney, C. Spagnol, and E. Popovici, “Comparative study of software vs. hardware implementations of shortened Reed-Solomon code for Wireless Body Area Networks,” in *27th International Conference on Microelectronics Proceedings (MIEL), 2010*, pp. 223–226, 2010.
- [25] C.-Y. Chiang, H.-H. Chen, T.-C. Chen, C.-S. Liu, Y.-J. Huang, S.-S. Lu, C.-W. Lin, and L.-G. Chen, “Analysis and design of on-sensor ECG processors for realtime detection of VF, VT, and PVC,” in *IEEE Workshop on Signal Processing Systems (SIPS)*, pp. 42–45, Oct. 2010.
- [26] B. Massot, C. Gehin, R. Nocua, A. Dittmar, and E. McAdams, “A wearable, low-power, health-monitoring instrumentation based on a programmable system-

- on-chip™,” in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 4852 –4855, Sept. 2009.
- [27] T. Ahola, P. Korpinen, J. Rakkola, T. Ramo, J. Salminen, and J. Savolainen, “Wearable FPGA Based Wireless Sensor Platform,” in *29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS)*, pp. 2288 –2291, Aug. 2007.
- [28] S. Borromeo, C. Rodriguez-Sanchez, F. Machado, J. Hernandez-Tamames, and R. de la Prieta, “A Reconfigurable, Wearable, Wireless ECG System,” in *29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS)*, pp. 1659 –1662, Aug. 2007.
- [29] M. Jenihhin, M. Gorev, V. Pesonen, D. Mihhailov, P. Ellervee, H. Hinrikus, M. Bachmann, and J. Lass, “EEG Analyzer prototype based on FPGA,” in *7th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pp. 101 –106, Sept. 2011.
- [30] H. Dubois-Ferriere, R. Meier, L. Fabre, and P. Metrailler, “TinyNode: a comprehensive platform for wireless sensor network applications,” in *The Fifth International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 358 –365, 2006.
- [31] L. Au, W. Wu, M. Batalin, D. McIntire, and W. Kaiser, “MicroLEAP: Energy-aware Wireless Sensor Platform for Biomedical Sensing Applications,” in *IEEE Biomedical Circuits and Systems Conference (BIOCAS)*, pp. 158 –162, Nov. 2007.
- [32] R. Rashid, M. Rahim, M. Sarijari, and N. Mahalin, “Design and implementation of Wireless Biomedical Sensor Networks for ECG home health monitoring,” in *International Conference on Electronic Design (ICED)*, pp. 1 –4, Dec. 2008.
- [33] A. Nassir and O. Barnea, “Wireless body-area network for detection of sleep

- disorders,” in *IEEE 27th Convention of Electrical Electronics Engineers in Israel (IEEEI), 2012*, pp. 1–5, 2012.
- [34] J. R. Frigo, E. Y. Raby, S. M. Brennan, C. Wolinski, C. Wagner, F. Charot, E. Rosten, and V. K. Kulathumani, “Energy efficient sensor node implementations,” in *18th annual ACM/SIGDA international symposium on Field Programmable Gate Arrays, FPGA '10*, (New York, NY, USA), pp. 37–40, ACM, 2010.
- [35] M. Healy, T. Newe, and E. Lewis, “Security for wireless sensor networks: A review,” in *IEEE Sensors Applications Symposium (SAS)*, pp. 80–85, Feb. 2009.
- [36] R. Lysecky and F. Vahid, “A study of the speedups and competitiveness of FPGA soft processor cores using dynamic hardware/software partitioning,” in *Proceedings of Design, Automation and Test in Europe, 2005.*, pp. 18–23 Vol. 1, 2005.
- [37] J. Wei, L. Wang, F. Wu, Y. Chen, and L. Ju, “Design and implementation of wireless sensor node based on open core,” in *IEEE Youth Conference on Information, Computing and Telecommunication (YC-ICT)*, pp. 102–105, Sept. 2009.
- [38] D.-H. T. That, A.-V. Dinh-Duc, and K. Phan-Dinh, “Implementation of TinyOS on FPGA system,” in *TENCON IEEE Region 10 Conference*, pp. 1456–1459, Nov. 2010.
- [39] L. Peng, L. Ping, L. A. Cai, and M. Y. Yi, “Design and implementation of safety monitoring and warning system of toxic and flammable gas for municipal sewer pipe based on Nios-II and GPRS,” in *6th IEEE Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, vol. 2, pp. 394–397, Aug. 2011.
- [40] S. Lam, K. L. Wong, K. O. Wong, W. Wong, and W. H. Mow, “A smartphone-centric platform for personal health monitoring using wireless wearable biosen-

- sors,” in *7th International Conference on Information, Communications and Signal Processing (ICICS)*, pp. 1–7, Dec. 2009.
- [41] H. Powell, M. Hanson, and J. Lach, “On-body inertial sensing and signal processing for clinical assessment of tremor,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 3, pp. 108–116, April 2009.
- [42] K. Wac, R. Bults, B. van Beijnum, I. Widya, V. Jones, D. Konstantas, M. Vollenbroek-Hutten, and H. Hermens, “Mobile patient monitoring: The mobihealth system,” in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 1238–1241, Sept. 2009.
- [43] S. Hu, Z. Shao, and J. Tan, “A real-time cardiac arrhythmia classification system with wearable electrocardiogram,” in *International Conference on Body Sensor Networks (BSN)*, pp. 119–124, May 2011.
- [44] W.-Y. Chung, Y.-D. Lee, and S.-J. Jung, “A wireless sensor network compatible wearable u-healthcare monitoring system using integrated ECG, accelerometer and SpO₂,” in *30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS)*, pp. 1529–1532, Aug. 2008.
- [45] T. Gao, T. Massey, L. Selavo, D. Crawford, B. rong Chen, K. Lorincz, V. Shnyder, L. Hauenstein, F. Dabiri, J. Jeng, A. Chanmugam, D. White, M. Sarrafzadeh, and M. Welsh, “The advanced health and disaster aid network: A light-weight wireless medical system for triage,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 1, pp. 203–216, Sept. 2007.
- [46] G.-Z. Yang, *Body Sensor Networks*. Springer, 2006.
- [47] G. Blumrosen, M. Uziel, B. Rubinsky, and D. Porrat, “Tremor acquisition system based on uwb wireless sensor network,” in *International Conference on Body Sensor Networks (BSN)*, pp. 187–193, June 2010.
- [48] “IEEE 802.15 WPAN Task Group 6 (TG6) Body Area Networks.” <http://www.ieee802.org/15/pub/TG6.html>. Accessed: Mar. 2012.

- [49] B. Yu, L. Xu, and Y. Li, "Bluetooth Low Energy (BLE) based mobile electrocardiogram monitoring system," in *International Conference on Information and Automation (ICIA)*, pp. 763 –767, June 2012.
- [50] ZigBee Alliance, "ZigBee Specification Document 053474r17," Jan. 2008.
- [51] R. Sahandi and Y. Liu, "Channel Overlap Problems of ZigBee Networks for Remote Patient Monitoring on General Hospital Wards," in *2010 International Conference on Communications and Mobile Computing (CMC)*, vol. 3, pp. 259 –263, april 2010.
- [52] A. Y. K. Chan, *Biomedical Device Technology: Principles and Applications*. Charles C Thomas Publisher Ltd., 2008.
- [53] Microchip Technologies Inc., "PIC18(L)F2X/4Xk22 Data Sheet Document DS41412F," Jun 2012.
- [54] Digi International Inc., "XBee/XBee-PRO Modules Product Manual v1.xEx - 802.15.4 Protocol For RF Module Part Numbers: XB24-A...-001, XBP24-A...-001," Sept. 2009.
- [55] Linear Technology, "LTC1983-3/LTC1983-5 100mA Regulated Charge-Pump Inverters in ThinSOT Document 1983fa," 2002.
- [56] "DE0 Nano Development and Education Board." <http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English &CategoryNo=139&No=593>. Accessed: Dec., 2012.
- [57] K. O'Brien, D. Salyers, A. Striegel, and C. Poellabauer, "Power and performance characteristics of USB flash drives," in *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 1 –4, June 2008.
- [58] "Lexar." <http://www.lexar.com/>. Accessed: July 2012.
- [59] "SD Association." <https://www.sdcard.org/home/>. Accessed: Mar. 2012.

- [60] Panasonic, SanDisk, Toshiba, “SD Specifications Part 1 Physical Layer Simplified Specification Version 3.01,” May 2010.
- [61] “Altera Forum.” <http://www.alteraforum.com/>. Accessed: Feb. to Jun. 2012.
- [62] “Altera Corporation.” <http://www.altera.com/>. Accessed: 2012.
- [63] Altera Corporation, “Embedded Peripherals IP User Guide Document UG-01085-11.0,” June 2011.
- [64] Altera Corporation, “Embedded Design Handbook ED_HANDBOOK-2.9,” July 2011.
- [65] “uClinux Embedded Linux/Microcontroller Project.” <http://www.uclinux.org/>. Accessed: Jan. to Jun. 2012.
- [66] “Analog Devices Open Source Koop.” <http://blackfin.uclinux.org/gf/>. Accessed: Feb. to Apr. 2012.
- [67] “Processors from Altera and Embedded Alliance Partners.” <http://www.altera.com/devices/processor/emb-index.html>. Accessed: Jan. 2012.
- [68] Altera Corporation, “SOPC Builder User Guide Document UG-01096-1.0,” Dec 2010.
- [69] J. Pan and W. J. Tompkins, “A Real-Time QRS Detection Algorithm,” *IEEE Transactions on Biomedical Engineering*, vol. BME-32, pp. 230 –236, march 1985.
- [70] “Physionet.” <http://www.physionet.org/>. Accessed: Aug., 2012.
- [71] G. Moody, R. Mark, and A. Goldberger, “Physionet: Physiologic signals, time series and related open source software for basic, clinical, and applied research,” in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 8327 –8330, Sept. 2011.

- [72] “PhysioBank ATM.” <http://www.physionet.org/cgi-bin/atm/ATM>. Accessed: Jun. to Aug., 2012.
- [73] G. Moody, “MIT-BIH Arrhythmia Database Directory (Introduction).” <http://www.physionet.org/physiobank/database/html/mitdbdir/intro.htm>. Accessed: Aug. 2012.
- [74] X. Li, X. Wang, F. Wang, Z. Wang, R. Xue, X. Dong, and P. Zhou, “Measurement of anesthesia depth based on heart rate variability,” in *IEEE International Conference on Virtual Environments Human-Computer Interfaces and Measurement Systems (VECIMS)*, pp. 186–189, July 2012.
- [75] J. Valencia, M. Vallverdu, R. Schroeder, I. Cygankiewicz, R. Vazquez, A. de Luna, A. Porta, A. Voss, and P. Caminal, “Heart rate variability characterized by refined multiscale entropy applied to cardiac death in ischemic cardiomyopathy patients,” in *Computing in Cardiology*, pp. 65–68, Sept. 2010.
- [76] F. M. Bui, *Signal Processing Methodologies for Resource-Efficient and Secure Communications in Wireless Networks*. PhD thesis, University of Toronto, 2009.
- [77] D. McCormick, “Cheap, Pressure-Sensing Electronic Skin,” *IEEE Spectrum Inside Technology*, 2012.
- [78] “Install Nios II Linux.” http://www.alterawiki.com/wiki/Install_Nios_II_Linux. Accessed: Feb. 2012.
- [79] “Running uClinux on Terasic DE0-Nano Altera Board.” <http://www.ccm.ece.vt.edu/twiki/bin/view/Main/LinuxOnNIOSS2InstallationDE0Nano>. Accessed: Feb. 2012.
- [80] “Download Ubuntu for your Desktop.” <http://www.ubuntu.com/download/desktop>. Accessed: Jan. 2012.
- [81] Parallax Inc., “XBee USB Adaptor Board (#32400) v1.0,” Jan. 2010.

A. FPGA Server System Development

This Appendix describes the step-by-step process required to:

- install Ubuntu on a Personal Computer (PC)
- download and install the uClinux distribution ported for the Nios II processor
- install Quartus II version 11.1 on a PC running Ubuntu
- configure a Nios II system using Quartus II and SOPC Builder
- program an EPCS flash memory to load FPGA configuration files
- build a customized uClinux boot image
- program an EPCS flash memory to boot uClinux on power-up
- automatically launch user programs through the uClinux boot sequence
- compile user programs to run using uClinux

Information within this chapter was compiled using online references [61, 62, 78–80].

A.1 Ubuntu Installation

1. Download Ubuntu Desktop from <http://www.ubuntu.com/download/desktop>. The download is a disc image file (.iso). Note: this work used v11.10.
2. Create a bootable CD. A disc image file can be burned to CD in Windows simply by right clicking on the .iso image and selecting 'Open with...' 'Windows Disc Image Burner'.

3. Insert the CD in your PC, cycle the power and boot from the CD.
4. Follow prompts to load Ubuntu.
5. Once installation is complete update the local package index with latest repositories. From the desktop press `crtl-alt-t` to launch a console window then enter the following command: `$sudo apt-get update`

6. Add some packages necessary for the system.

```
$sudo apt-get install git-core git-gui make gcc ncurses-dev bison flex gawk get-
text ccache zlib1g-dev libx11-dev texinfo liblz2-dev pax-utils uboot-mkimage
corkscrew
```

7. Change shell to bash. (Required for Altera software)

```
$sudo rm /bin/sh
```

```
$sudo ln -s bash /bin/sh
```

A.2 uClinux Distribution

1. From the console window, navigate to the directory where you would like to install the uClinux-dist, this work used the home directory, and download the Nios II uClinux distribution tarball.

```
$wget http://sopc.et.ntust.edu.tw/pub/linux/nios2-linux-20100621.tar
```

2. Extract files from the tarball.

```
$sudo tar -xf nios2-linux-20100621.tar
```

3. Checkout the source files for uClinux and GNU tools.

```
./checkout
```

4. Switch to Nios II without MMU branch. Run the following command from `/nios2-linux/linux-2.6` and `/nios2-linux/uClinux-dist`.

```
$git checkout test-nios2
```

The command should respond with "Switched to branch 'test-nios2'"

5. Download the prebuilt toolchain. Note: the toolchain can be built manually but this is recommended.

```
$wget http://sopc.et.ntust.edu.tw/pub/gnutools/nios2gcc-20080203.tar.bz2
```

6. Extract files from tarball.

```
$sudo tar -jxf nios2gcc-20080203.tar.bz2 -C ~/nios2-linux
```

7. Add the path to the cross compiler to your `.bashrc` file. Using your favorite text editor, add the following lines at the end of the file.

```
export PATH=$PATH:~/nios2-linux/opt/nios2/bin
```

A.3 Quartus II Installation on Ubuntu

1. From the console window, navigate to the directory where you would like to install Quartus, this work used the home directory, and download Quartus II Linux and Quartus Devices Linux from <https://www.altera.com/download/dnl-index.jsp>

2. Invoke Quartus install from command line. Note: this work used Quartus 11.1 build 173.

```
$/bin/bash 11.1_173_quartus_linux.sh
```

This should open the Altera installer window - follow the prompts.

The installer will install Quartus II, Nios SBT, Nios II GCC4 Toolchain.

3. Repeat previous step for the Quartus Devices installation script.

```
$/bin/bash 11.1_173_devices_cyclone_max_legacy_linux.sh
```

Note: This work only included support for the Cyclone IV E family of FPGAs.

4. Add path information to `.bashrc`. This run command script is located in your home directory. Using your favorite text editor, add the following lines at the end of the file.

```
export PATH=$PATH:~/altera/11.1/Quartus/bin
```

```

export PATH=$PATH:~/altera/11.1/nios2eds/bin
export PATH=$PATH:~/altera/11.1/sopc_builder/bin
export PATH=$PATH:~/altera/11.1/nios2eds/bin/gnu/H-i686-pc-linux-gnu/bin
SOPC_KIT_NIOS2=~/altera/11.1/nios2eds
export SOPC_KIT_NIOS2

```

5. Configure the Joint Test Action Group (JTAG) USB Blaster. Navigate to the etc directory: `$cd ../../etc` from your home directory. Using your favorite text editor, add the following lines at the end of the file, `rc.local`. Note: you will need to do this as root, for example `$sudo vi rc.local`.

```

mount -bind /dev/bus /proc/bus

ln -s /sys/kernel/debug/usb/devices /proc/bus/usb/devices

```

6. Create a rules file for the USB blaster. Using your favorite text editor, create a file, `/etc/udev/rules.d/51-usbblaster.rules` and add the following.

```

# Altera USB-Blaster rule to set mode to 666.

SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", SYSFS{idVendor}=="
"09fb", SYSFS{idProduct}=="6001", MODE=="0666", NAME="bus/usb/$env
{BUSNUM}/$env{DEVNUM}", RUN+=" /bin/chmod 0666 %c"

```

7. Confirm that the USB Blaster is working. Plug the DE0-Nano Development and Education Board into a free USB port on your PC and run the following from the command prompt.

```

$jtagconfig

```

You should see something like this, if successful.

- 1) USB-Blaster [5-2] 020F30DD EP3C25/EP4CE22

A.4 System Development with Quartus II and SOPC Builder

1. Launch Quartus II from the command prompt. `Ctrl-alt-t` for a console.

\$quartus

2. Click "Create New Project (New Project Wizard)" at the "Getting started" window.
3. Click "Next" at the introduction screen.
4. On page 1 of the project wizard "Directory, Name Top-Level Entity (Page 1 of 5)",
 - Select a location to store the project. This work used the home directory. A better suggestion would be to create a "projects" directory within home.
 - Insert a project name. This work used "zxcv".
 - DO NOT change the Top-Level design entity name i.e. keep it the same as the project name.
 - click "Next".
5. Click "Next" at the "Add Files (Page 2 of 5)" screen.
6. At the "Family and Device Settings (Page 3 of 5)" screen, select the EP4CE22F17C6 device and click "Next".
7. Click "Next" at the "EDA Tool Settings (Page 4 of 5)" screen.
8. Review the summary and click "Finish" at the "Summary Page (Page 5 of 5)" screen.
9. Launch SOPC Builder. From the main toolbar: "Tools ->SOPC Builder".
10. A window should appear called "Create New System".
 - Give your system a name that is DIFFERENT from your Quartus project name. This work used "zxcv-system".
 - Select "Verilog" as the target HDL.
 - Click "OK".

11. Add components and configure the system using the "Component Library" in the left hand pane of the main SOPC Builder window. The components can be added to your system by simply double clicking the component title.
 - Select the "SDRAM Controller" from "Memories and Memory Controllers" ->"External Memory Interfaces" ->"SDRAM Interfaces" and then select the parameters as shown in Figures A.1 and A.2.
 - Select the "EPCS Serial Flash Controller" from "Memories and Memory Controllers" ->"External Memory Interfaces" ->"Flash Interfaces" and leave parameters as default.
 - Select the "NIOSE Processor" from "Processors". Set parameters as shown in Figure A.3. Leave all others as default.
 - Select "JTAG UART" from "Interface Protocols" ->"Serial". Leave all parameters as default.
 - Select "Interval Timer" from "Peripherals" ->"Microcontroller Peripherals". Leave all parameters as default.
 - Select "Avalon PLL" from "PLL". Select parameters as shown in Figures A.4, A.5 and A.6 and leave parameters as default on all other configuration pages.
 - Select "SPI (3 Wire Serial)" from "Interface Protocols" ->"Serial". Set parameters as shown in Figure A.7.
 - Select "UART (RS-232 Serial Port)" from "Interface Protocols" ->"Serial". Change BAUD Rate to 19k2 and leave all other parameters as default.
12. Rename the clocks to suit the design. In the "Clock Settings" window, double click the clocks to alter the names. Change the external clock to "clk_50", the "alt_pll0.c0" clock to "pll_sys" and the "alt_pll0.c1" clock to "pll_sdram", Figure A.8.
13. Reassign clocks to the system components as shown in Figure A.8 by clicking the clock name in the clock column and selecting the appropriate clock.

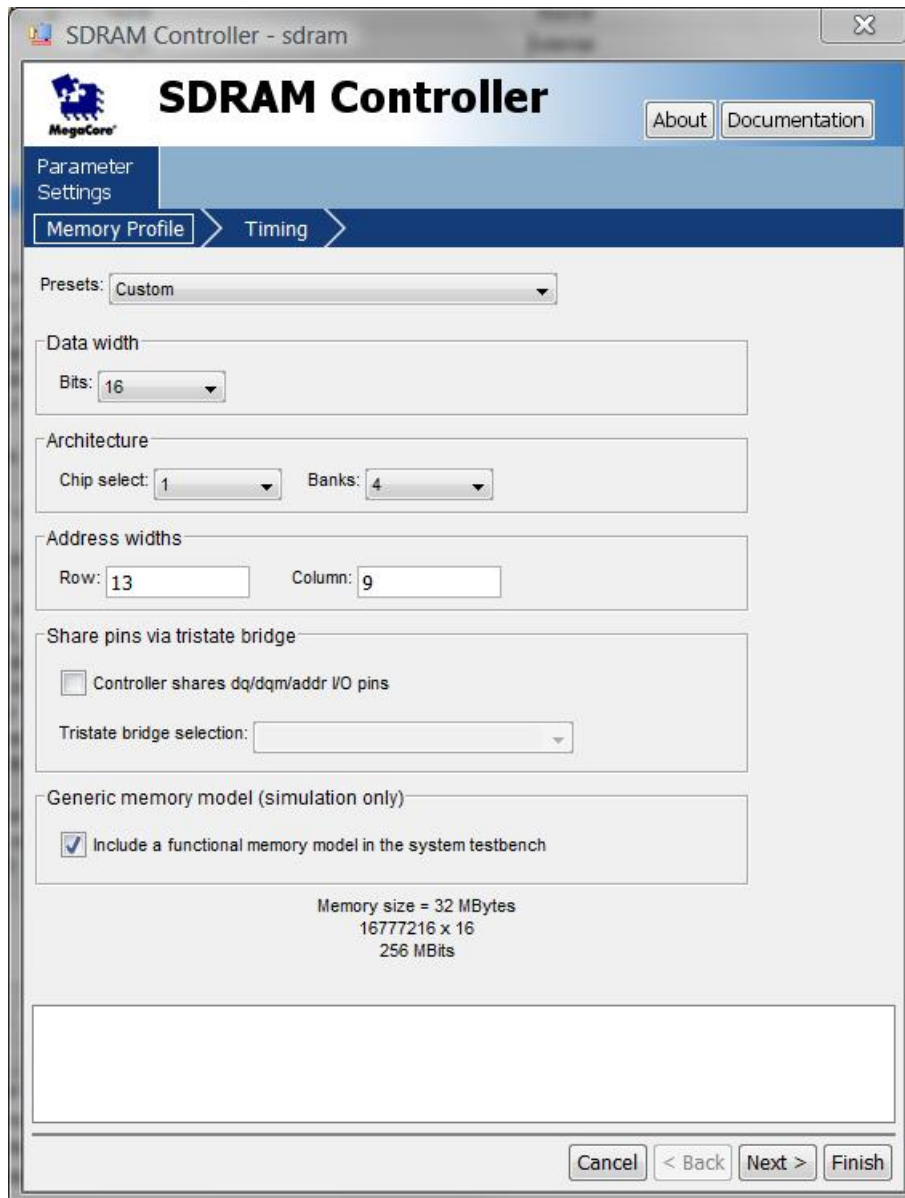


Figure A.1 SDRAM Memory Profile

14. Rename "spi_0" to "mmc_spi" to ensure uClinux-dist driver compatibility.
15. On the main toolbar, select "System ->" Assign Base Addresses" and then "Assign Interrupt Numbers".
16. Click "Generate" at the bottom of the main window, to build the system. Keep same system name when prompted to save. After build, SOPC Builder should respond with "System generation was successful".

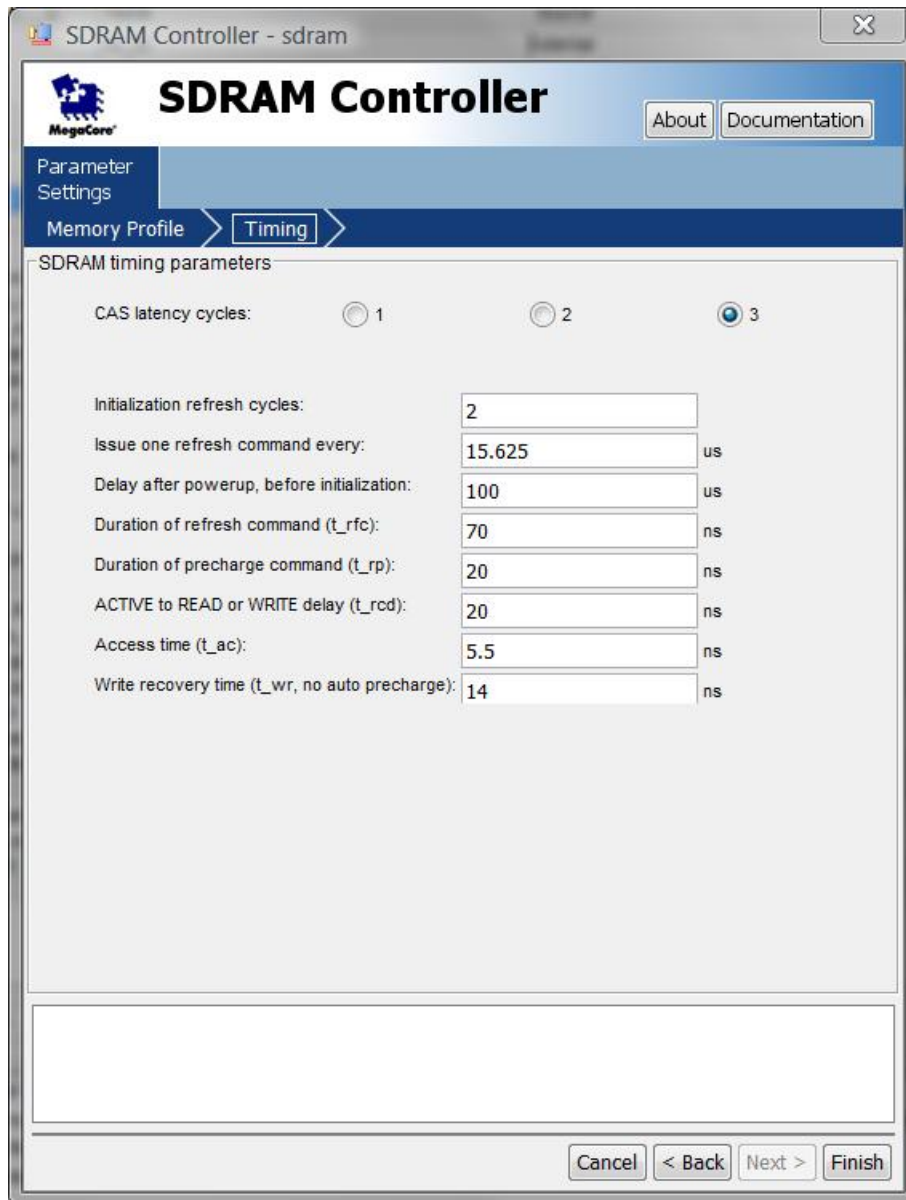


Figure A.2 SDRAM Timing

17. Take note of the EPCS base address as it will be required for programming the FPGA configuration data and the uClinux kernel image into the flash memory.
18. Exit SOPC Builder.
19. Add all SOPC Builder Verilog generated files to your Quartus project, except for the sample instantiation file, which will be called <your system name>_inst.v.
 - On the Quartus toolbar, select "Project" -> "Add/Remove Files in Project..."

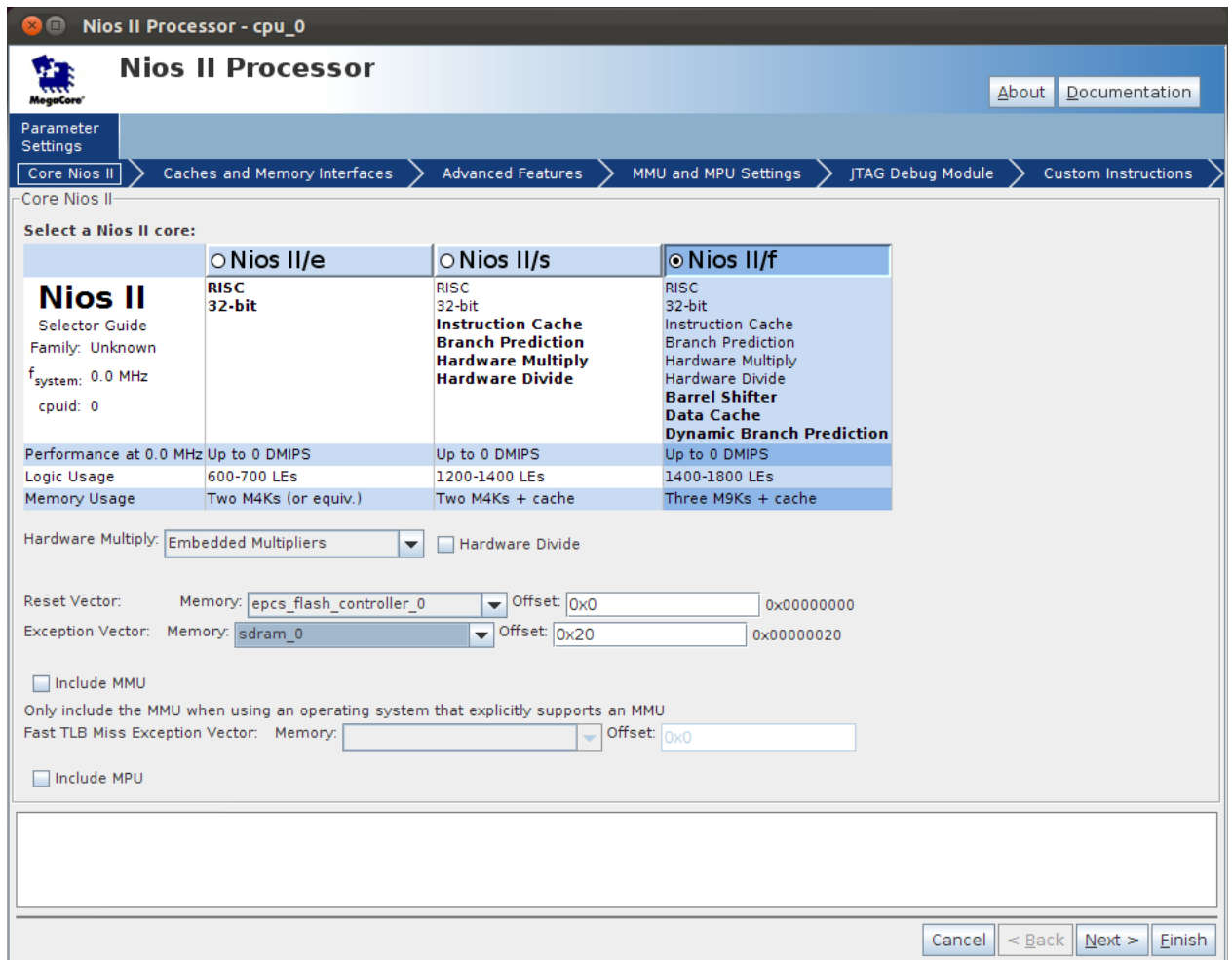


Figure A.3 Nios II Configuration

- Click the ellipsis, "...".
- Select all .v files (expect the sample instantiation).
- Click "Open".
- Click "OK"

20. Instantiate the system in the Quartus project. A sample is provided in Appendix G.3.

- Create a .v file with the same name as your Quartus project. This is your Top Level Design Entity.
- Add the file to the Quartus project, as in the previous step.

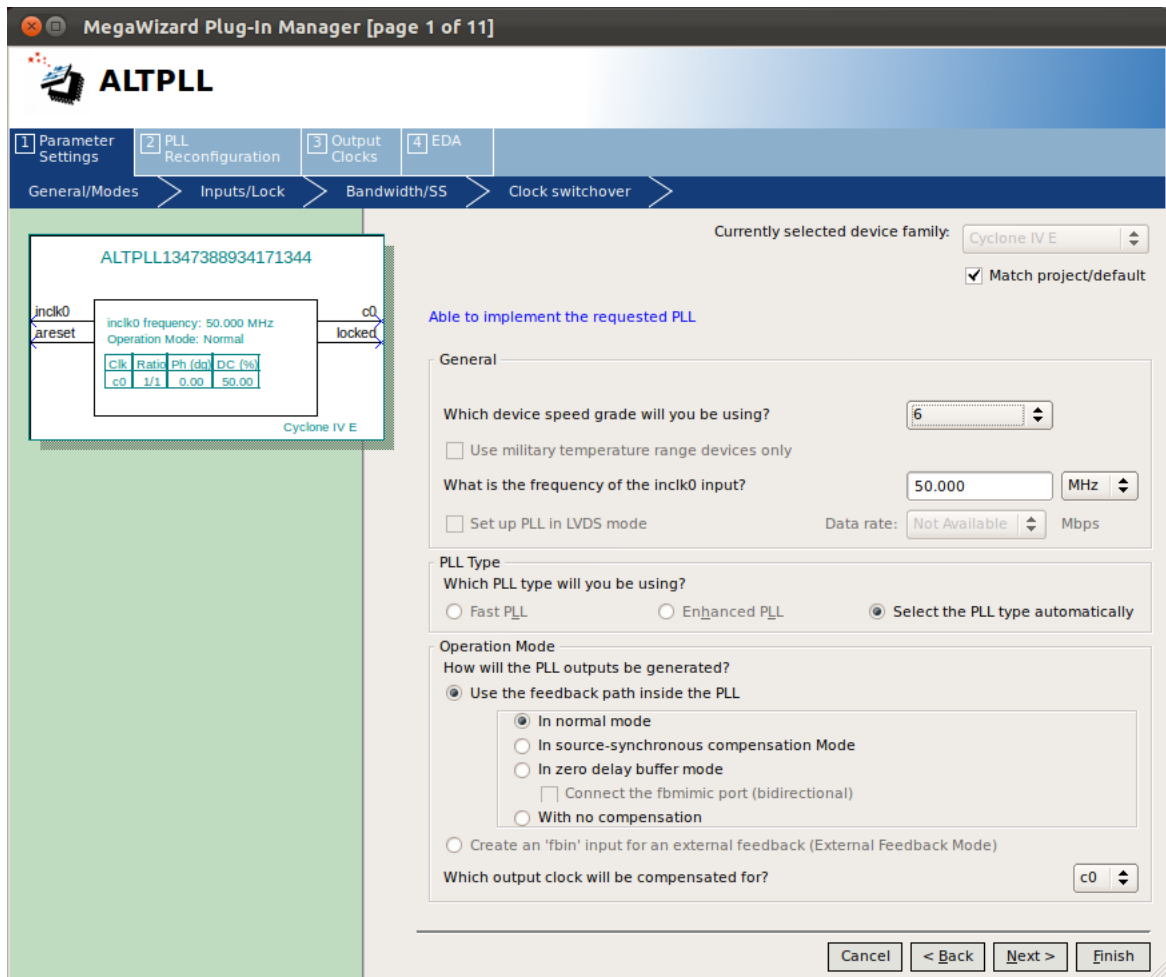


Figure A.4 PLL Configuration page 1

- Add the system instantiation to this file i.e. open the sample instantiation and copy the contents into the Top Level Design Entity.
 - Add all I/O declarations.
21. Set pin options so that the EPCS will work with the Cyclone IV device. Note: this is required for Cyclone III and IV devices only.
- On the Quartus toolbar, select "Assignments" -> "Device..."
 - Click "Device and Pin Options" and select "Dual Purpose Pins".
 - Set the following pins to "Use as regular I/O".

– DCLK

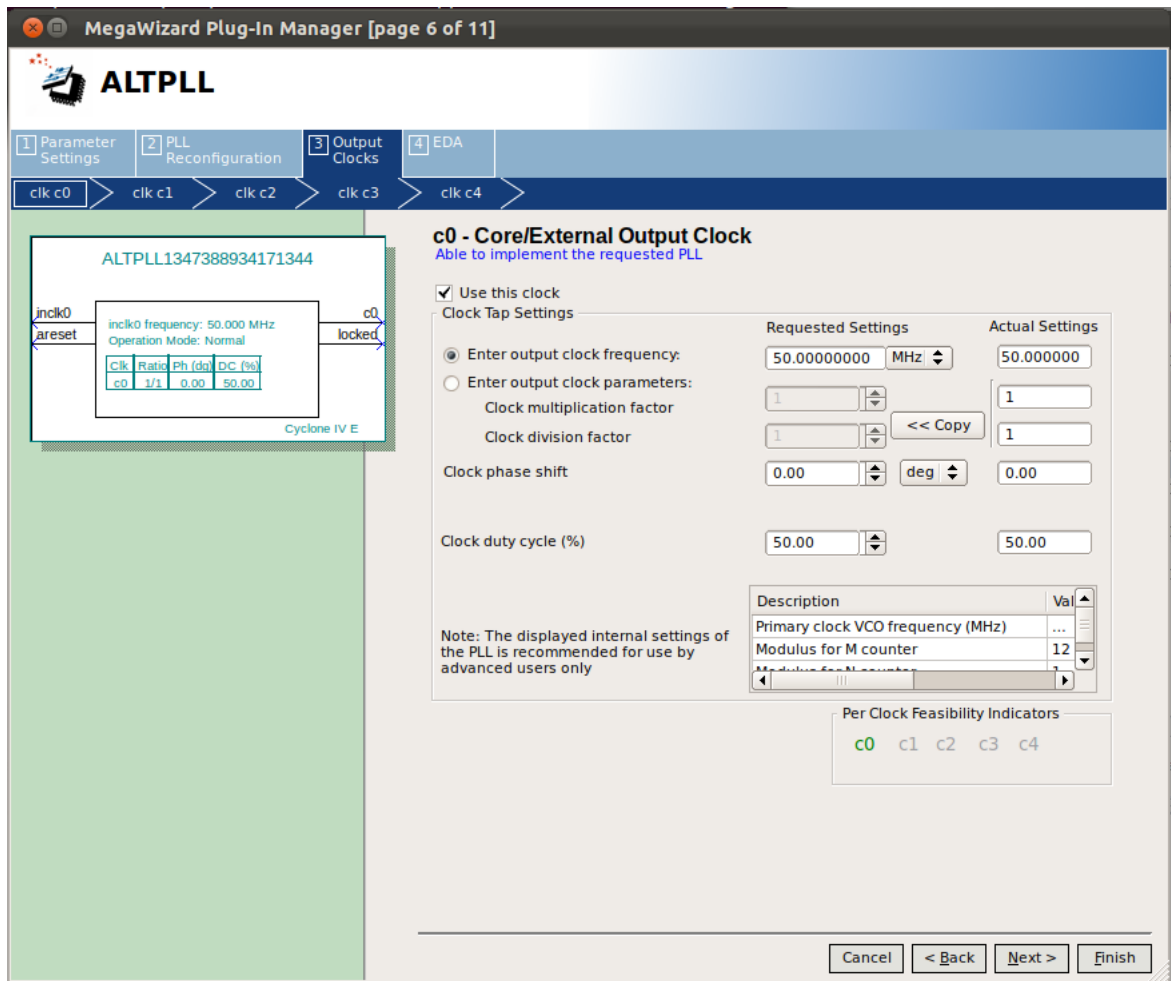


Figure A.5 PLL Configuration page 6

- Data[0]
 - Data[1]/ADSO
 - FLASH_nCE/nCSO
 - Click "OK".
22. From the main Quartus window, start "Analysis and Synthesis" by pressing ctrl-k.
 23. Assign pins. On the Quartus toolbar, select "Assignments" → "Pin Planner". The pin assignments used in this work are provided in Appendix B.
 24. Open the system design constraints file (cpu_0.sdc) and add the following lines of code to the end of the file.

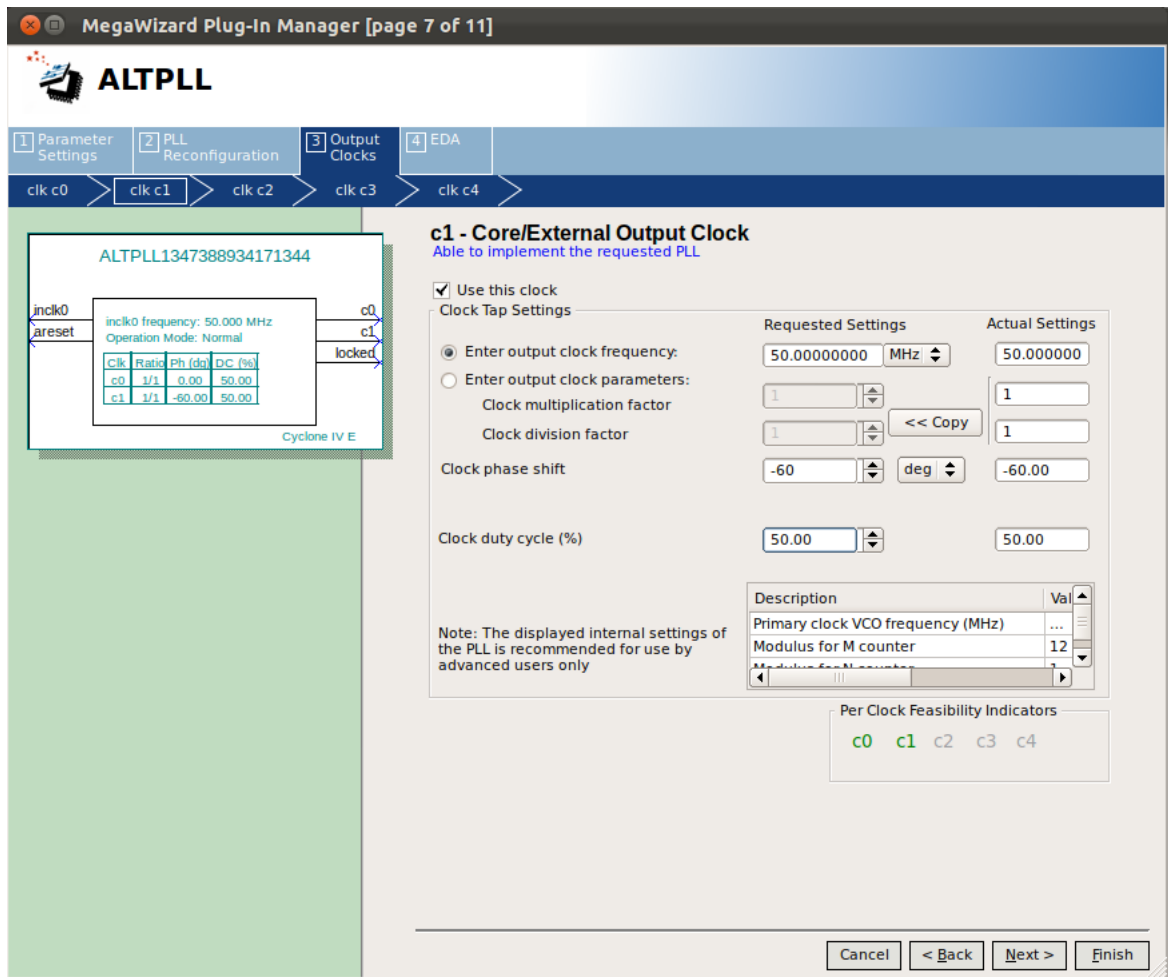


Figure A.6 PLL Configuration page 7

```
create_clock -name "clk_50" -period 20 [get_ports{clk_50}]
```

```
derive_pll_clocks
```

```
derive_clock_uncertainty
```

25. Compile the design.

A.5 Building the uClinux Boot Image

1. Open a console window, ctrl-alt-t, and navigate to nios2-linux/uClinux-dist.
2. Begin customizing the uClinux kernel image to your hardware. Run the following from the command prompt, which will run the menu configuration GUI.

```
$make menuconfig
```

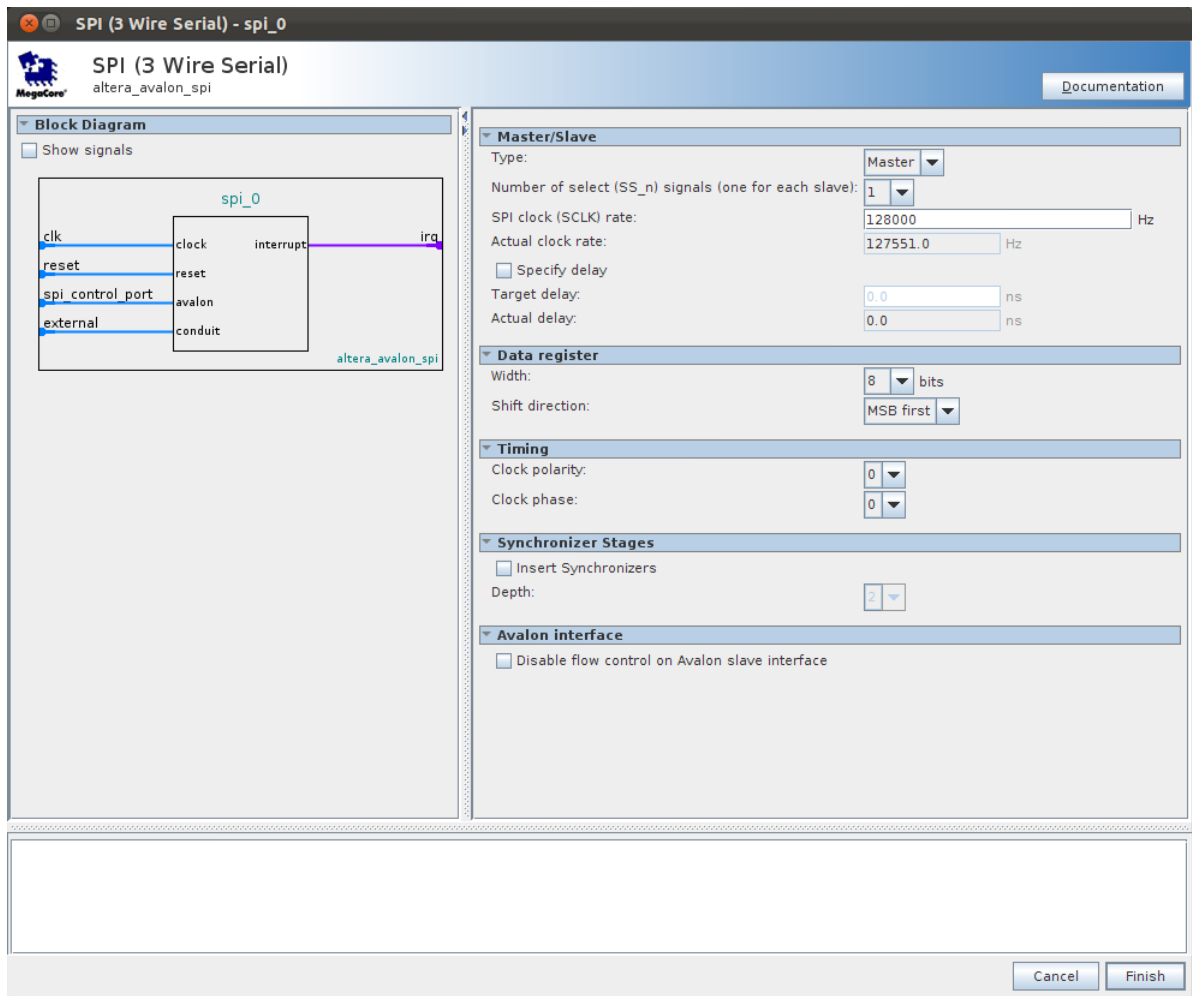


Figure A.7 SPI (3 Wire Serial) Configuration

3. Select "Vendor/Product Selection" ->"Vendor" ->"Altera Products" and select "Altera" as the vendor and "NIOS2" as the processor. Exit the menu configuration GUI.
4. Identify your hardware so that it can be used by the toolchain during the kernel build. Run the following from the command prompt and select cpu_0 (as the CPU to build the kernel against) and sdram_0 (as the device to execute the kernel from).

```
$make vendor_hwselect SYSPTF=~/.zxcv_system.ptf
```

5. Enter menuconfig again.

```
$make menuconfig
```

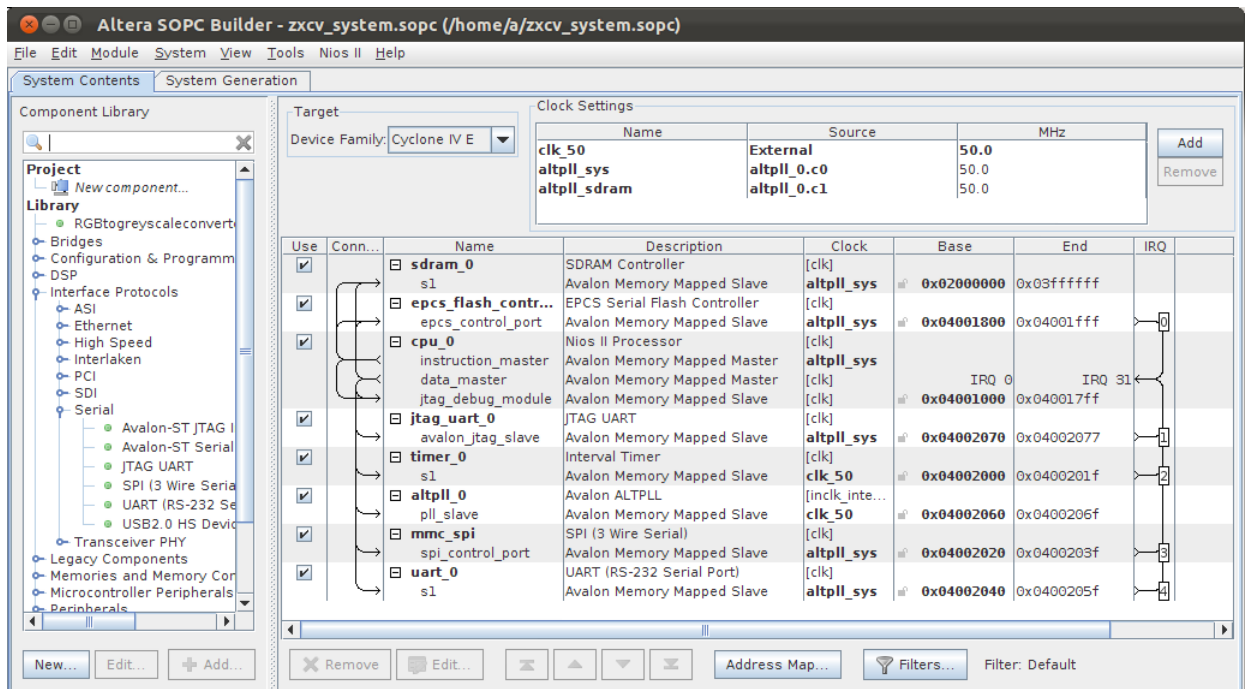


Figure A.8 SOPC Builder System Configuration

6. Enter the "Linux 2.6.30 Kernel Configuration".

Kernel/Library/Defaults - - ->

[*] Customize Kernel Settings

Then exit menuconfig and select "Yes" when prompted to save.

7. This will open "Linux 2.6.30 Kernel Configuration". From here make the following selections.

- Uncheck "Network Support"
- Add support for SD Card.

Device Drivers - - ->

[*] SPI Support

[*] Altera SPI Controller

Exit

[*] MMC/SD/SDIO Card Support

[*] MMC/SD/SDIO Over SPI

```

Deselect bounce buffer
[ ] Use bounce buffer for simple hosts
Exit
Exit
File Systems - - ->
[*] Ext Journaling file system support (This will automatically select Ext
extended attributes)
Exit
DOS/FAT/NT Filesystems - - ->
[*] VFAT (Windows 95) fs support
Exit
-*- Native Language Support
[*] Code Page 437 (United States, Canada)
[*]NLS ISO 8859-1 (Latin 1; Western European Languages)

```

8. Exit the configuration GUI and select "Yes" when prompted to save.
9. Build the image. The resulting image will be placed into uClinux-dist/images as "zImage".

```
$make
```

A.6 Configuring the FPGA and Writing the Configuration to Flash Memory

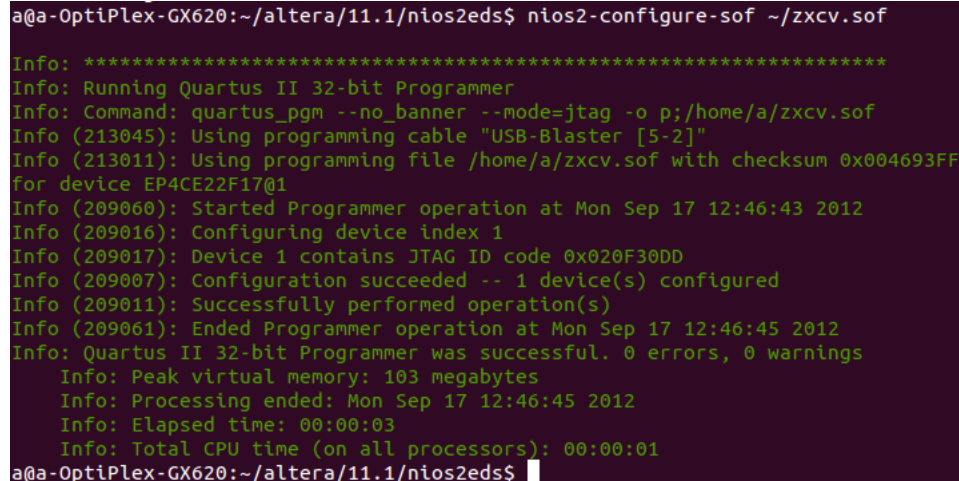
1. Open a console window, ctrl-alt-t, navigate to altera/11.1/nios2eds and launch the Nios2 command shell. Note: The Nios2 command shell must be running to use commands: nios2-configure-sof, nios2-terminal, nios2-download, nios2-flash-programmer, sof2flash and elf2flash.

```
$/nios2_command_shell.sh
```

2. Navigate to your Quartus project folder and then run the following command.

If successful the response will be as shown in Figure A.9.

```
$nios2-configure-sof zxcv.sof
```



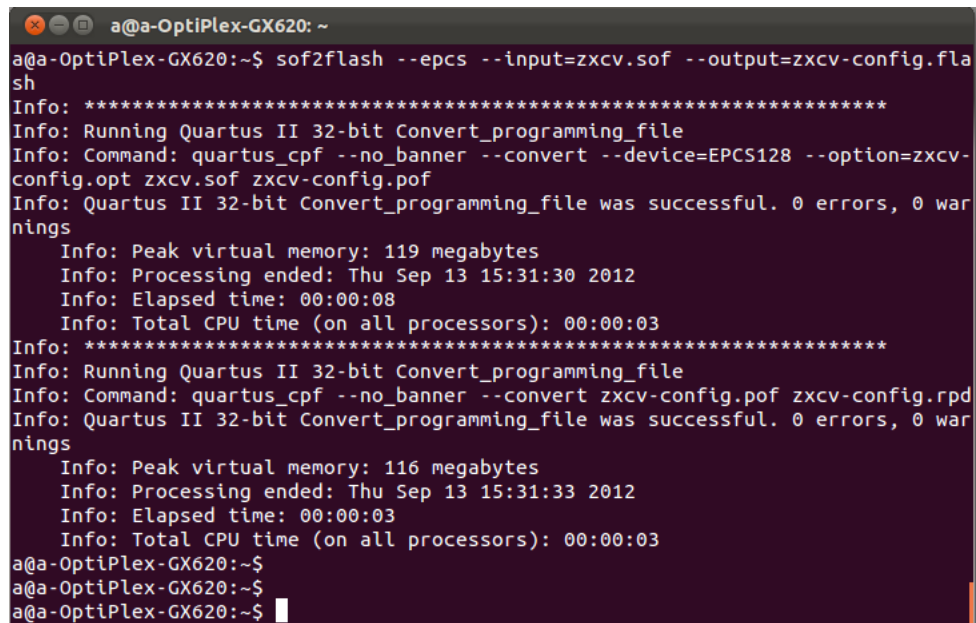
```
a@a-OptiPlex-GX620:~/altera/11.1/nios2eds$ nios2-configure-sof ~/zxcv.sof
Info: *****
Info: Running Quartus II 32-bit Programmer
Info: Command: quartus_pgm --no_banner --mode=jtag -o p;/home/a/zxcv.sof
Info (213045): Using programming cable "USB-Blaster [5-2]"
Info (213011): Using programming file /home/a/zxcv.sof with checksum 0x004693FF
for device EP4CE22F17@1
Info (209060): Started Programmer operation at Mon Sep 17 12:46:43 2012
Info (209016): Configuring device index 1
Info (209017): Device 1 contains JTAG ID code 0x020F30DD
Info (209007): Configuration succeeded -- 1 device(s) configured
Info (209011): Successfully performed operation(s)
Info (209061): Ended Programmer operation at Mon Sep 17 12:46:45 2012
Info: Quartus II 32-bit Programmer was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 103 megabytes
Info: Processing ended: Mon Sep 17 12:46:45 2012
Info: Elapsed time: 00:00:03
Info: Total CPU time (on all processors): 00:00:01
a@a-OptiPlex-GX620:~/altera/11.1/nios2eds$
```

Figure A.9 nios2-configure-sof Output

3. Create the flash file.

```
$sof2flash --epcs --input=zxcv.sof --output=zxcv-config.flash
```

Response should be as in Figure A.10.



```
a@a-OptiPlex-GX620: ~
a@a-OptiPlex-GX620:~$ sof2flash --epcs --input=zxcv.sof --output=zxcv-config.flash
Info: *****
Info: Running Quartus II 32-bit Convert_programming_file
Info: Command: quartus_cpf --no_banner --convert --device=EPCS128 --option=zxcv-config.opt zxcv.sof zxcv-config.pof
Info: Quartus II 32-bit Convert_programming_file was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 119 megabytes
Info: Processing ended: Thu Sep 13 15:31:30 2012
Info: Elapsed time: 00:00:08
Info: Total CPU time (on all processors): 00:00:03
Info: *****
Info: Running Quartus II 32-bit Convert_programming_file
Info: Command: quartus_cpf --no_banner --convert zxcv-config.pof zxcv-config.rpd
Info: Quartus II 32-bit Convert_programming_file was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 116 megabytes
Info: Processing ended: Thu Sep 13 15:31:33 2012
Info: Elapsed time: 00:00:03
Info: Total CPU time (on all processors): 00:00:03
a@a-OptiPlex-GX620:~$
a@a-OptiPlex-GX620:~$
a@a-OptiPlex-GX620:~$
```

Figure A.10 sof2flash Output

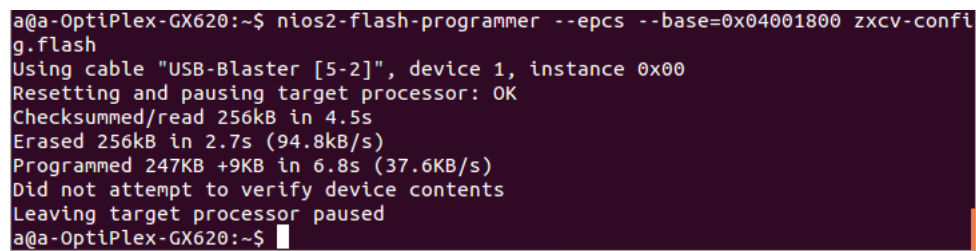
4. Program the EPCS flash memory.

```
$nios2-flash-programmer --epcs --base=0x04001800 zxcv-config.flash
```

Note: the hex address 0x04001800 is the base address of the EPCS as defined in SOPC Builder.

Response should be as in Figure A.11.

5. Cycle the power and your FPGA will automatically configure from EPCS flash memory.



```
a@a-OptiPlex-GX620:~$ nios2-flash-programmer --epcs --base=0x04001800 zxcv-config.flash
Using cable "USB-Blaster [5-2]", device 1, instance 0x00
Resetting and pausing target processor: OK
Checksummed/read 256kB in 4.5s
Erased 256kB in 2.7s (94.8kB/s)
Programmed 247KB +9KB in 6.8s (37.6KB/s)
Did not attempt to verify device contents
Leaving target processor paused
a@a-OptiPlex-GX620:~$
```

Figure A.11 nios2-flash-programmer Output

A.7 Customizing the uClinux Boot Sequence

1. Navigate to nios2-linux/uClinux-dist/vendors/Altera/nios2.
2. Using your favorite text editor, add the following lines at the end of the rc file.

```
#create a directory for use with the sd card
mkdir /mnt/sd
#mount the sd card filesystem
mount -t vfat /dev/mmcblk0p1 /mnt/sd #copy user files from the sd card to
the kernel file system
cd /bin
cp /mnt/sd/serial_test ./
cd ..
```

Note: serial_test is the name of the user software used in this work.

3. Build the kernel from uClinux-dist.

```
$cd ../../..
```

```
$make
```

A.8 Setting up the System to Boot from EPCS Flash Memory

1. Place the zImage file, located in nios2-linux/uClinux-dist/images, and the system configuration file, zxcv-config.sof, into the same directory.

2. Launch the nios2 command shell and convert the zImage to a flash file.

```
./nios2_command_shell.sh
```

```
$elf2flash --epcs --after=zxcv-config.flash --input=zImage --output=zxcv-image.flash  
--boot=$SOPC_KIT_NIOS2/components/altera_nios2/boot_loader_epcs.srec
```

3. Program the EPCS flash memory.

```
$nios2-flash-programmer --epcs --base=0x04001800 zxcv-image.flash
```

4. Cycle the power to the DE0-Nano Development and Education Board and then run nios2-terminal from the command prompt.

```
$nios2-terminal
```

This should boot the uClinux kernel. The response should be as in Figure A.12.

A.9 Developing User Software

1. Using your favorite text editor, create a C program.

2. Cross compile the program. The following command will generate an object file suitable for loading into the uClinux file system. Note: this command is native to the uClinux distribution.

```
$nios2-linux-uclibc-gcc serial_test.c -o serial_test -elf2ft
```

3. Add the user program to the kernel boot sequence as in Section A.7.

B. FPGA Pin Assignments

This appendix contains the pin assignments for the FPGA Server system.

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard
associate_xbee	Input	PIN_R13	4	B4_NO	3.3-V LVTTTL
clk_50	Input	PIN_R8	3	B3_NO	3.3-V LVTTTL
data0	Input	PIN_H2	1	B1_NO	3.3-V LVTTTL
dclk	Output	PIN_H1	1	B1_NO	3.3-V LVTTTL
din_xbee	Output	PIN_A12	7	B7_NO	3.3-V LVTTTL
dout_xbee	Input	PIN_D12	7	B7_NO	3.3-V LVTTTL
reset_xbee	Output	PIN_C11	7	B7_NO	3.3-V LVTTTL
sce	Output	PIN_D2	1	B1_NO	3.3-V LVTTTL
sd_clk	Output	PIN_C8	8	B8_NO	3.3-V LVTTTL
sd_cmd	Bidir	PIN_T10	4	B4_NO	3.3-V LVTTTL
sd_data	Bidir	PIN_B7	8	B8_NO	3.3-V LVTTTL
sd_data3	Bidir	PIN_P11	4	B4_NO	3.3-V LVTTTL
sdo	Output	PIN_C1	1	B1_NO	3.3-V LVTTTL
sdram_clk	Output	PIN_R4	3	B3_NO	3.3-V LVTTTL
sdram_wi...addr[12]	Output	PIN_L4	2	B2_NO	3.3-V LVTTTL
sdram_wi...addr[11]	Output	PIN_N1	2	B2_NO	3.3-V LVTTTL
sdram_wi...addr[10]	Output	PIN_N2	2	B2_NO	3.3-V LVTTTL
sdram_wire_addr[9]	Output	PIN_P1	2	B2_NO	3.3-V LVTTTL
sdram_wire_addr[8]	Output	PIN_R1	2	B2_NO	3.3-V LVTTTL
sdram_wire_addr[7]	Output	PIN_T6	3	B3_NO	3.3-V LVTTTL
sdram_wire_addr[6]	Output	PIN_N8	3	B3_NO	3.3-V LVTTTL
sdram_wire_addr[5]	Output	PIN_T7	3	B3_NO	3.3-V LVTTTL
sdram_wire_addr[4]	Output	PIN_P8	3	B3_NO	3.3-V LVTTTL
sdram_wire_addr[3]	Output	PIN_M8	3	B3_NO	3.3-V LVTTTL
sdram_wire_addr[2]	Output	PIN_N6	3	B3_NO	3.3-V LVTTTL
sdram_wire_addr[1]	Output	PIN_N5	3	B3_NO	3.3-V LVTTTL
sdram_wire_addr[0]	Output	PIN_P2	2	B2_NO	3.3-V LVTTTL
sdram_wire_ba[1]	Output	PIN_M6	3	B3_NO	3.3-V LVTTTL
sdram_wire_ba[0]	Output	PIN_M7	3	B3_NO	3.3-V LVTTTL
sdram_wire_cas_n	Output	PIN_L1	2	B2_NO	3.3-V LVTTTL
sdram_wire_cke	Output	PIN_L7	3	B3_NO	3.3-V LVTTTL
sdram_wire_cs_n	Output	PIN_P6	3	B3_NO	3.3-V LVTTTL
sdram_wire_dq[15]	Bidir	PIN_K1	2	B2_NO	3.3-V LVTTTL
sdram_wire_dq[14]	Bidir	PIN_N3	3	B3_NO	3.3-V LVTTTL
sdram_wire_dq[13]	Bidir	PIN_P3	3	B3_NO	3.3-V LVTTTL
sdram_wire_dq[12]	Bidir	PIN_R5	3	B3_NO	3.3-V LVTTTL
sdram_wire_dq[11]	Bidir	PIN_R3	3	B3_NO	3.3-V LVTTTL
sdram_wire_dq[10]	Bidir	PIN_T3	3	B3_NO	3.3-V LVTTTL
sdram_wire_dq[9]	Bidir	PIN_T2	3	B3_NO	3.3-V LVTTTL
sdram_wire_dq[8]	Bidir	PIN_T4	3	B3_NO	3.3-V LVTTTL
sdram_wire_dq[7]	Bidir	PIN_R7	3	B3_NO	3.3-V LVTTTL
sdram_wire_dq[6]	Bidir	PIN_J1	2	B2_NO	3.3-V LVTTTL
sdram_wire_dq[5]	Bidir	PIN_J2	2	B2_NO	3.3-V LVTTTL
sdram_wire_dq[4]	Bidir	PIN_K2	2	B2_NO	3.3-V LVTTTL
sdram_wire_dq[3]	Bidir	PIN_K5	2	B2_NO	3.3-V LVTTTL
sdram_wire_dq[2]	Bidir	PIN_L8	3	B3_NO	3.3-V LVTTTL
sdram_wire_dq[1]	Bidir	PIN_G1	1	B1_NO	3.3-V LVTTTL
sdram_wire_dq[0]	Bidir	PIN_G2	1	B1_NO	3.3-V LVTTTL
sdram_wire_dqm[1]	Output	PIN_T5	3	B3_NO	3.3-V LVTTTL
sdram_wire_dqm[0]	Output	PIN_R6	3	B3_NO	3.3-V LVTTTL
sdram_wire_ras_n	Output	PIN_L2	2	B2_NO	3.3-V LVTTTL
sdram_wire_we_n	Output	PIN_C2	1	B1_NO	3.3-V LVTTTL
sleep_rq_xbee	Output	PIN_E11	7	B7_NO	3.3-V LVTTTL

Figure B.1 FPGA Server Pin Assignments

C. XBee Configuration

This Appendix provides XBee configuration details using a USB to serial adaptor board and a custom terminal program called X-CTU, which is run on a PC with a Microsoft Windows OS. The XBee configuration can also be implemented using a terminal emulator, in Linux for instance or through user software in an embedded system.

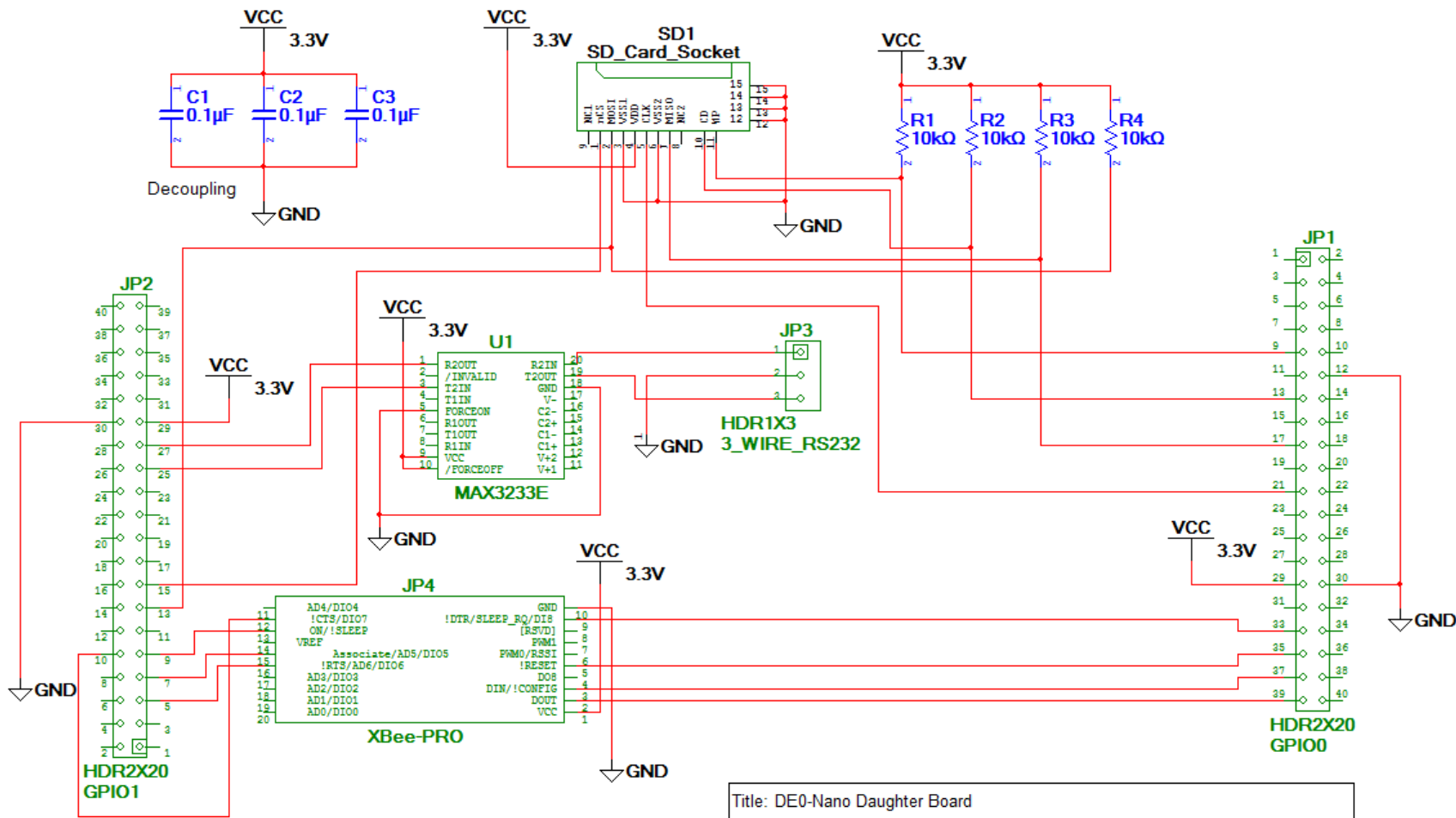
1. Acquire an XBee USB Adaptor Board. This part is available from Digikey (<http://www.digikey.ca/>), part number 32400-ND, or Parallax (<http://www.parallax.com/>), part number 32400.
2. Follow the assembly and installation instructions in [81]. Be sure to download and install the USB drivers, as noted on page 4.
3. Plug an XBee RF module into the pin header on the adaptor board and the plug the adaptor board into a free USB port on your computer.
4. Download and install the X-CTU Software from <http://www.digi.com/support/productdetail?pid=3352>. X-CTU is a gui terminal program that enables quick configuration and firmware upgrades of XBee modules.
5. Run the X-CTU Software.
6. Select the appropriate COM port and configure the port for 9600 BAUD, No Flow Control, 8 Data Bits, No Parity and 1 Stop Bit. Note: the default BAUD rate for the XBee modules is 9600 BAUD. If you have modified the non-volatile

memory in the XBee module to select a different BAUD rate, enter that rate here.

7. Select the "Modem Configuration" tab.
8. Click "Read". Your device should respond with a list of configuration parameters.
9. Modify the configuration parameters as follows:
 - Channel = 10
 - PAN ID = DEED
 - Destination Address High = 0
 - Destination Address Low = 68 (set paired device 16b Source Address to match this)
 - 16b Source Address = 69 (set paired device Destination Address to match this)
 - End Device Association = 0 - 000B
 - Sleep Mode = 2 (ZigBee mote), 0 (FPGA Server)
 - Power Level = Lowest
 - Interface Data Rate = 4 - 19200
10. Click "Write". This will write your configuration parameters to non-volatile memory.

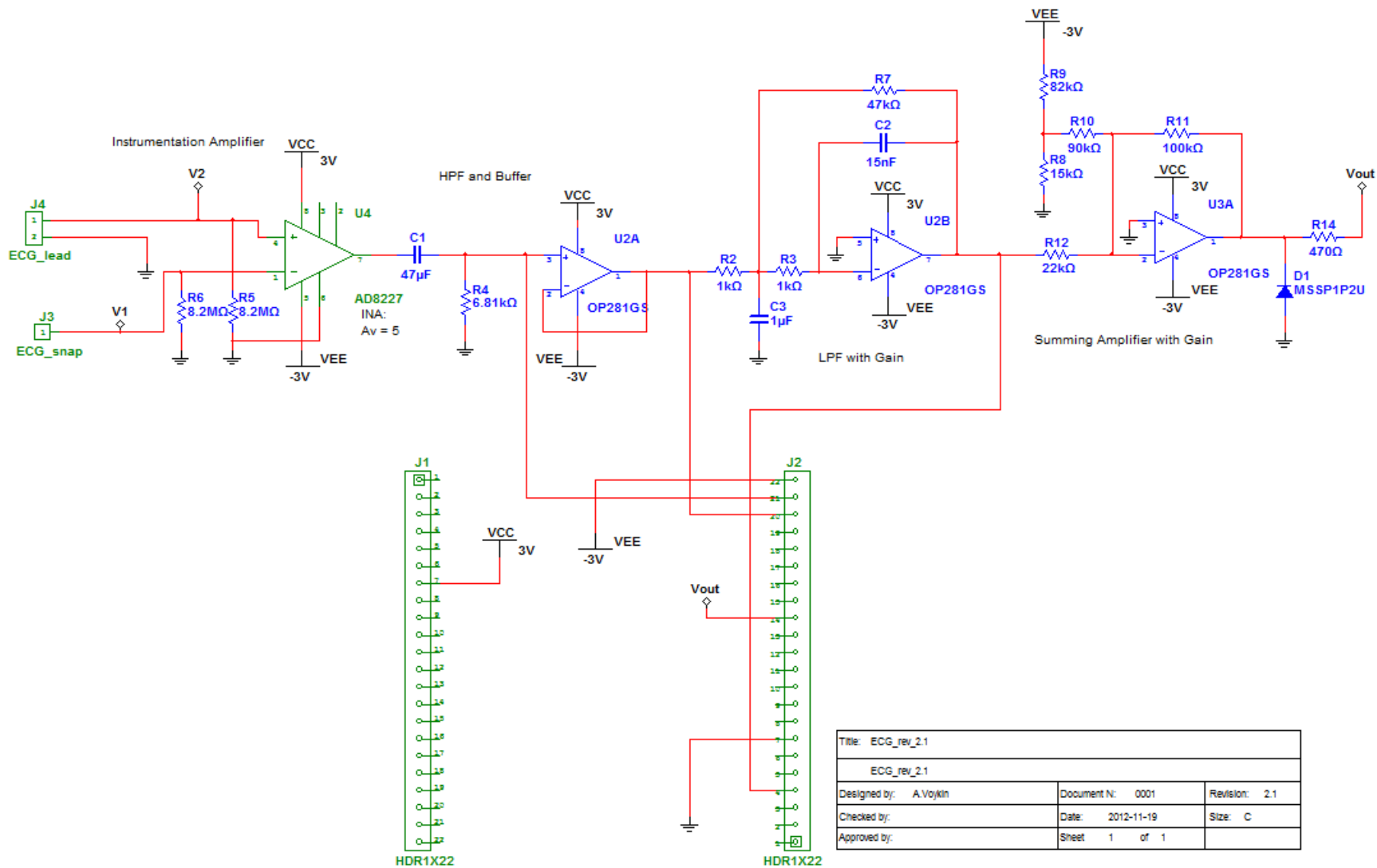
D. Schematics

This appendix contains four schematics. Figure D.1 is the daughter board designed to connect to the DE0-Nano Development and Education Board from Terasic Technologies (<http://www.terasic.com/>). Figure D.2 is analog daughter board, which connects to the Zigbee mote. Figure D.3 is the base board for the mote, which contains an XBee radio from Digi International Inc. (<http://www.digi.com/>) and a microcontroller from Microchip Technology Inc. (<http://www.microchip.com/>). Finally, D.4 is a schematic of the test platform used to verify the BSN system operation.



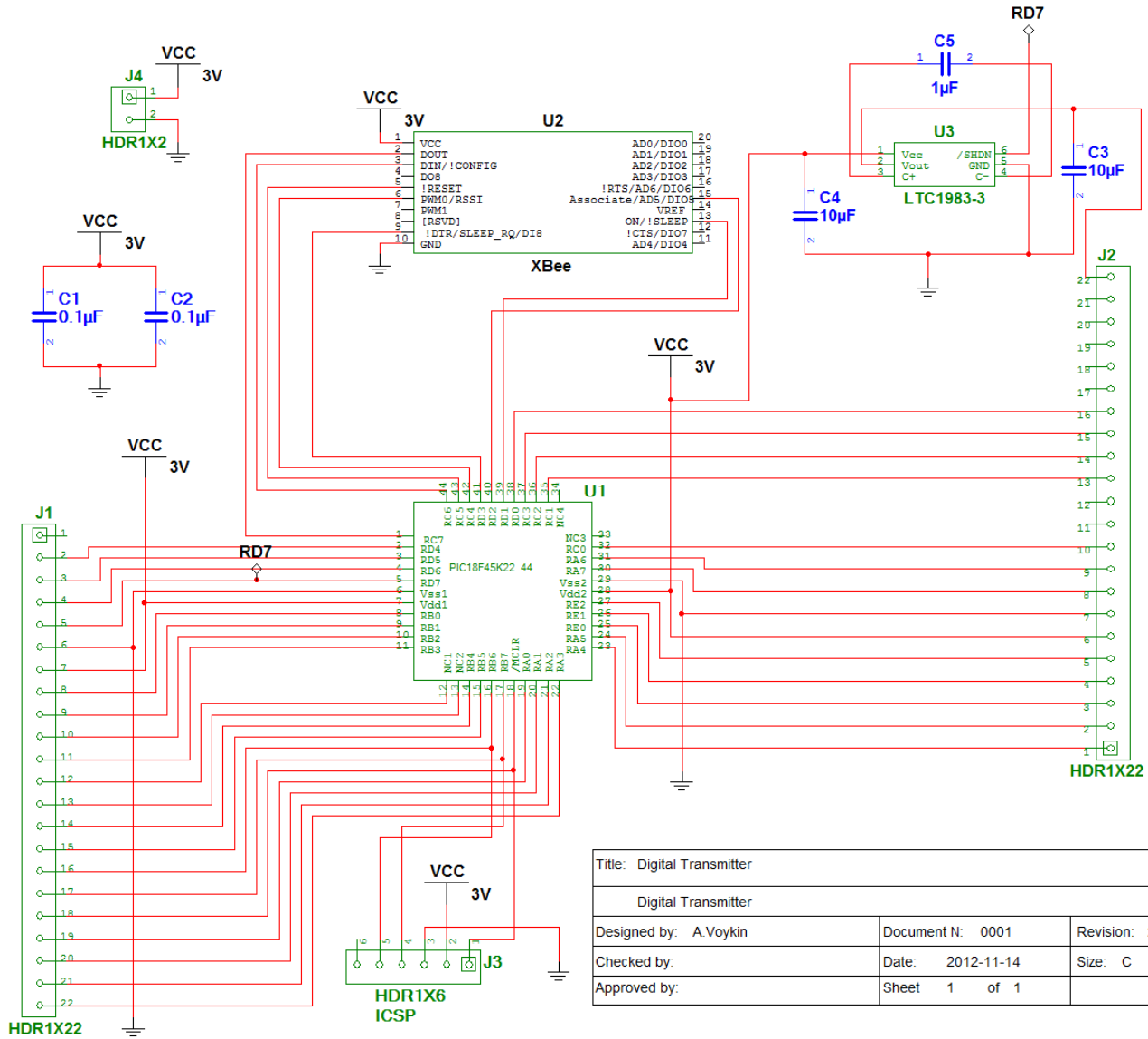
Title: DE0-Nano Daughter Board		
Zigbee, RS232 and SD Card		
Designed by: A. Voykin	Document N: 0001	Revision: 1.1
Checked by:	Date: 2012-11-14	Size: A
Approved by:	Sheet 1 of 1	

Figure D.1 DE0-Nano Daughter Board Schematic



Title: ECG_rev_2.1		
ECG_rev_2.1		
Designed by: A.Vojtkin	Document N: 0001	Revision: 2.1
Checked by:	Date: 2012-11-19	Size: C
Approved by:	Sheet 1 of 1	

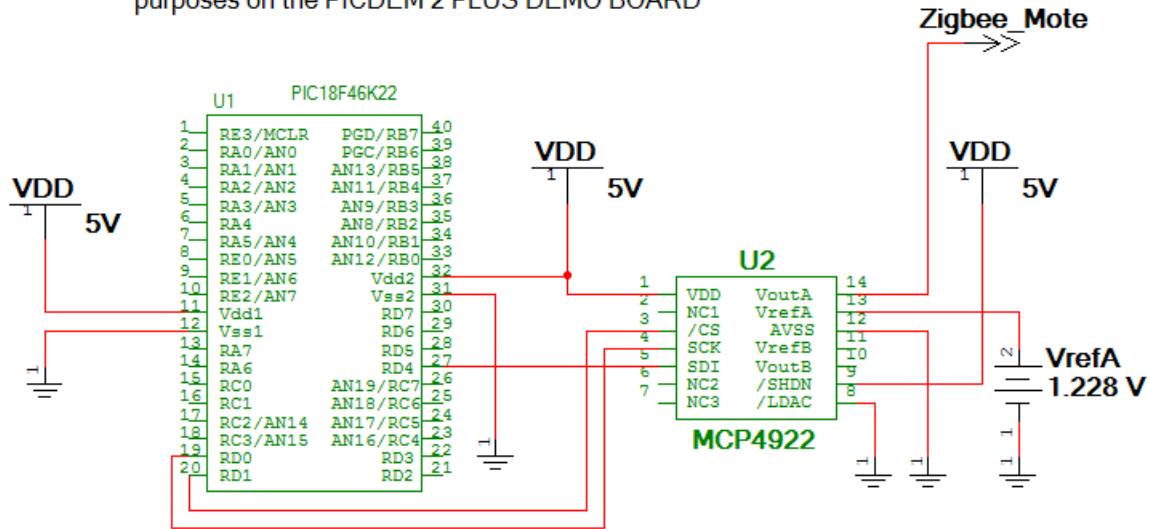
Figure D.2 ECG AFE Schematic



Title: Digital Transmitter		
Digital Transmitter		
Designed by: A.Voykin	Document N: 0001	Revision: 2.0
Checked by:	Date: 2012-11-14	Size: C
Approved by:	Sheet 1 of 1	

Figure D.3 Digital Transmitter Schematic

Note: Unused pins on PIC18F46K22 are used for other purposes on the PICDEM 2 PLUS DEMO BOARD



Title: MIT_TEST_PLATFORM		
MIT_TEST_PLATFORM		
Designed by: A. Voykin	Document N: 0001	Revision: 1.0
Checked by:	Date: 2012-11-14	Size: A
Approved by:	Sheet 1 of 1	

Figure D.4 Test Platform Schematic

E. Bill of Materials

The following three tables provide a listing of parts used to build the Electrocardiogram (ECG) Analog Front End (AFE), Digital Transmitter and DE0-Nano Daughter Board. In addition to the parts in the coming tables the DE0-Nano Development and Education Board was purchased from Terasic Technologies Inc. for \$59USD.

Table E.1 DE0-Nano Daughter Board BOM

Qty.	Description	RefDes	Type	Vendor	Vendor Part No.	Unit Price	Ext. Price
2	HDR2X20	JP1, JP2	Header	Digi-Key	S9200-ND	2.26	4.52
2	Xbee Connector	JP4	Header	Digi-Key	S9259-ND	3.03	6.06
1	CAP, 0.1 μ F	C1, C2, C3	Decoupling	Digi-Key	478-1395-1-ND	0.11	0.11
1	HDR1X3	JP3	Header	Digi-Key	S7001-ND	0.65	0.65
4	RES, 10k	R1, R2, R3, R4	Resistor	Digi-Key	P10KACT-ND	0.05	0.2
1	MAX3233E	U1	IC	Digi-Key	MAX3233EEWP+G36-ND	13.92	13.92
1	SD_Card_Socket	SD1	Socket	Digi-Key	A101492CT-ND	3.7	3.7
1	Xbee Module	na	RF Module	Digi-Key	XB24-ACI-001-ND	18.65	18.65
TOTAL COST							47.81

Table E.2 ECG AFE BOM

Qty	Description	RefDes	Vendor	Vendor Part No.	Unit Price	Ext. Price
1	IA, AD8227	U4	Digikey	AD8227BRZ	5.77	5.77
1	RES, 6.81k Ω	R4	Digikey	RR12P6.81KDCT-ND	0.13	0.13
2	RES, 8.2M Ω	R5, R6	Digikey	P8.2MACT-ND	0.05	0.1
1	CAP, 47 μ F	C1	Digikey	587-1779-1-ND	0.94	0.94
1	RES, 47k Ω	R7	Digikey	P47KACT-ND	0.05	0.05
1	CAP, 15nF	C2	Digikey	445-4435-1-ND	0.16	0.16
1	CAP, 1F	C3	Digikey	490-1695-1-ND	0.08	0.08
2	OPAMP, OP281GS	U2, U3	Digikey	OP281GSZ-ND	6.43	12.86
1	RES, 15k Ω	R8	Digikey	P15KACT-ND	0.11	0.11
1	RES, 82k Ω	R9	Digikey	P82KACT-ND	0.11	0.11
1	RES, 90k Ω	R10	Digikey	RR12P90.9KDCT-ND	0.13	0.13
1	RES, 100k Ω	R11	Digikey	P100KACT-ND	0.11	0.11
1	RES, 22k Ω	R12	Digikey	P22KACT-ND	0.11	0.11
2	RES, 1k Ω	R2, R3	Digikey	P1.0KACT-ND	0.11	0.22
1	RES, 470 Ω	R14	Digikey	P470ACT-ND	0.11	0.11
1	Diode	D1	Newark	MSSP1P2U	0.156	0.156
1	ECG_lead	J4	na	na	0	0
1	ECG_snap	J3	na	na	0.65	0.65
2	HDR1X22	J1, J2	Digikey		0.5	1
TOTAL COST						22.80

Table E.3 Digital Transmitter BOM

Qty	Description	RefDes	Type	Vendor	Vendor Part No.	Unit Price	Ext. Price
2	Xbee	U2	Header	Digi-Key	S9259-ND	3.03	6.06
1	PIC18LF45K22-44	U1	Microcontroller	Digikey	PIC18LF45K22-1/PT	3.56	3.56
1	HDR1X6	J3	Programming Header	Digikey	A31448-ND	3.13	3.13
2	CAP, 0.1 μ F	C1, C2	Decoupling Capacitor	Digikey	478-1395-1-ND	0.11	0.22
2	HDR1X22	J1, J2	0.1 Pitch Single Row	Digikey		0.5	1
1	HDR1X2	J4	0.1 Pitch Single Row	Digikey		0.1	0.1
2	CAP, 10 μ F	C3, C4	Filter Capacitor	Digikey	587-1344-1-ND	0.46	0.92
1	CAP, 1 μ F	C5	Filter Capacitor	Digikey	445-1354-1-ND	0.17	0.17
1	LTC1983-3	U3	DC to DC Converter	Digikey	LTC1983ES6-3	4.57	4.57
1	Xbee Module			Digi-Key	XB24-ACI-001-ND	18.65	18.65
TOTAL COST							38.38

F. PCB Artwork

The following three figures include the PCB artwork for the DE0-Nano Daughter Board, ECG AFE and Digital Transmitter. Each of the figures includes copper top and bottom as well as silkscreen top and bottom layers.

Note: the ECG AFE PCB artwork is for v2.0 but the final ECG AFE schematic is v2.1. In the final schematic resistors R1 and R13 were omitted and resistor R14 and diode D1 were added.

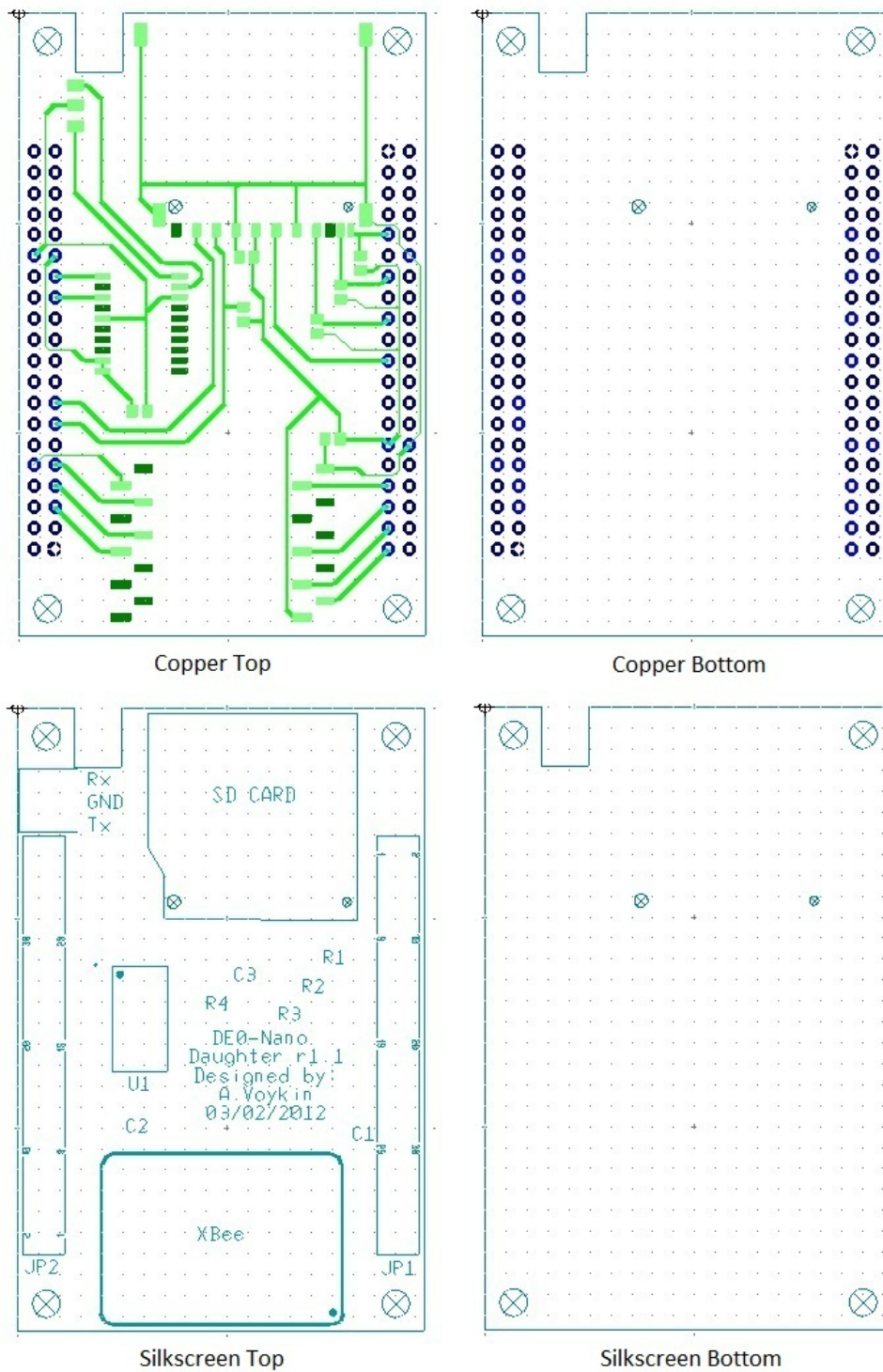
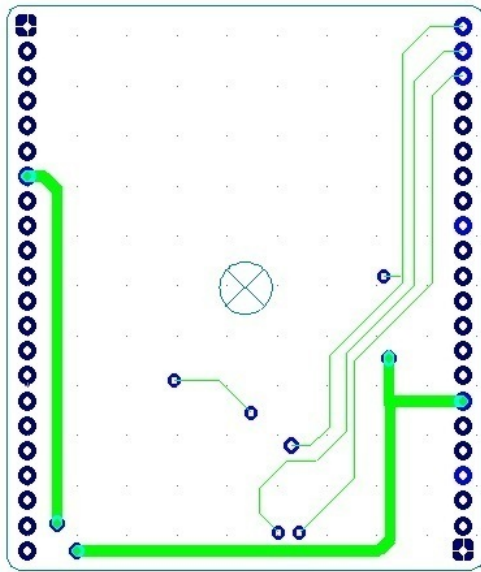
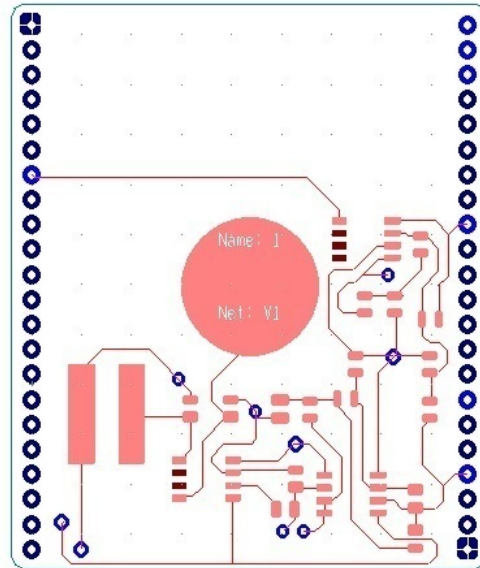


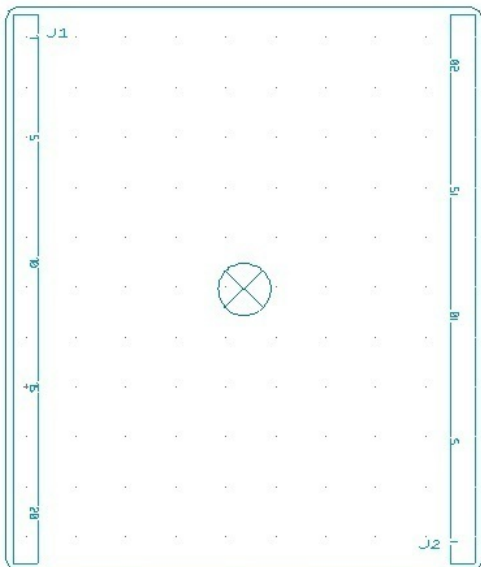
Figure F.1 DE0-Nano Daughter Board



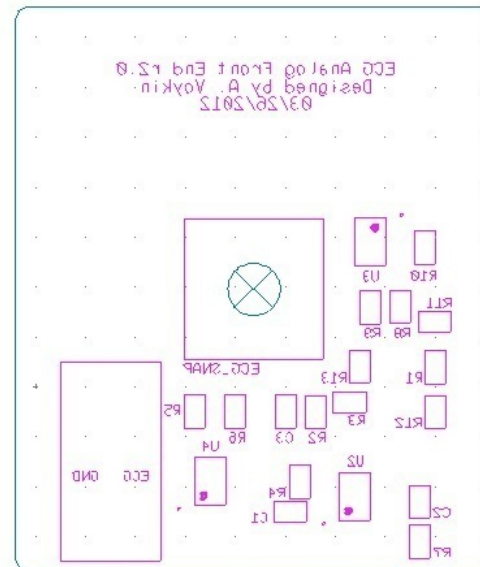
Copper Top



Copper Bottom



Silkscreen Top



Silkscreen Bottom

Figure F.2 ECG AFE

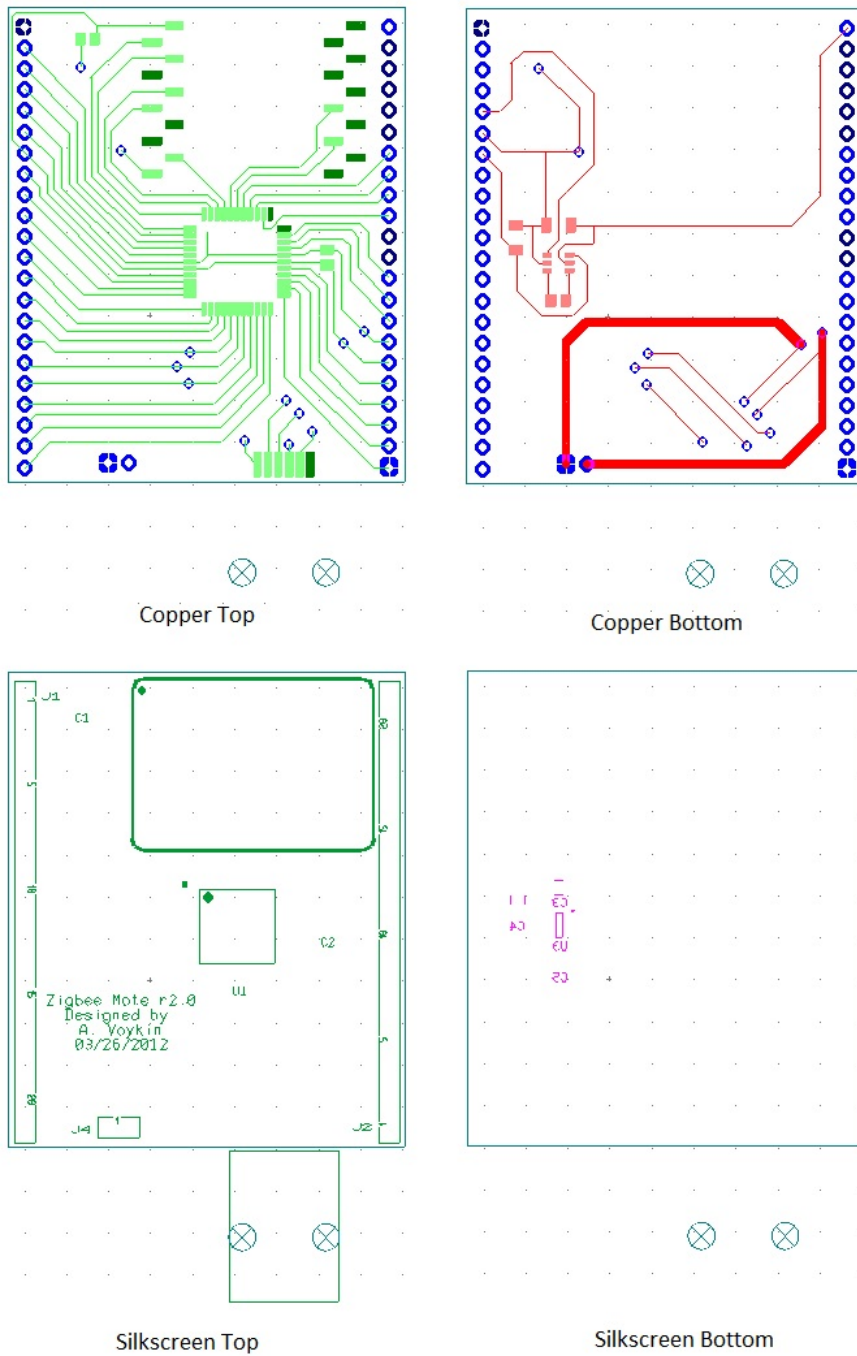


Figure F.3 Digital Transmitter

G. Software

This appendix contains all of the software used in this research.

G.1 MIT-BIH Parsing Engine

```

/*****
Written by: Anthony Voykin
Date: Jun 2012
*****/

USAGE:
Compile this C program on a PC running Windows.
From the Physiobank ATM:
-select MIT-BIH Arrythmia Database
-samples
-raw ADC units
-Export signal as CSV
Download and open the CSV file, remove the top line (the
text), save as samples.txt in the same folder as the
executable and exit. Run the parsing engine to create format
for microchip project. The parsing engine "uses" MLII and
discards the remainder of each row.The output is stored as
output.txt in the same folder as the executable.
*****/
#include <stdio.h>
```

```

void main(void)
{
    int temp,i,n;
    FILE *data, *output;
    data=fopen("samples.txt","r"); //open file for reading
        //open output file for reading/writing (overwrite
        //contents), create if doesn't exist
    output=fopen("output.txt","w+");
    for(n=0;n<225;n++)
    //for(n=0;n<1350;n++)
    {
        for(i=0;i<16;i++)
        {
            fscanf(data,"%i", &temp); //ignore sample number
            fscanf(data,"%i", &temp);
            fprintf(output,"%i",temp);
            fscanf(data,"%i", &temp); //ignore second lead in
            // MIT DB file
        }
        fprintf(output,"\n");
    }
    fclose(data);
    fclose(output);
}

```

G.2 Test Platform C Code

```

////////////////////////////////////
//                               main.c
// This program is used to test the operation of the BSN
// system designed and built by Anthony Voykin as a partial

```

```

// fulfilment of the M.Sc. program at the Univ. of
// Saskatchewan. The purpose of the program is send MIT-BIH
// samples to a 12 bit DAC via SPI. MIT-BIH samples are
// placed into the array ecg_data. Modify the array length
// according to the number of samples. Also modify the line
// of code if(i==3600) to match the number of samples.
////////////////////////////////////
#include <p18F46K22.h>//Includes microcontroller specific
                        //predefines
#include "prototypes.h" //All function prototypes
#include "mydefines.h" //Includes my defines, fuses and
                        //global variables

#include <stdio.h>
#include <timers.h>
#include <stdlib.h>
#include <delays.h>

int timer0_flag;
rom int ecg_data[3600]= {910,913,915,914,916,917,916,915,
//add MIT-BIH samples
//to this buffer, only part is shown for practicality
.
.
.
888,884,885,884,886,885,888,890,888,889,889,890,888,888};
void main(void)
{
    int i=0,temp;
    unsigned char high_byte=0b01010000,low_byte;
    port_init();

```

```

timer0_init();
SSP_init();
INTCONbits.GIE = 0b1;
while(1)
{
    if(timer0_flag)
    {
        //if timer0 has expired,send a sample
        //(timer0 is set to expire 360 times per second)
        //adjust MIT-BIH samples to match baseline
        temp = ecg_data[i]-600; //remove some DC offset
        temp = (float)temp * 3.25; //adjust amplitude
        temp = temp & 0x0f00; //mask to prepare for SPI DAC
        temp = temp >> 8;
        high_byte = high_byte | temp;
        temp = ecg_data[i++]-600;
        temp = (float)temp * 3.25;
        temp = temp & 0x00fe;
        low_byte = temp;
        timer0_flag = 0;          //reset timer flag
        CS = 0;                  //enable DAC
        SSP2BUF = high_byte;     //send high byte to DAC
        while(!SSP2STATbits.BF); //wait for Tx to complete
        SSP2BUF = low_byte;     //send low byte to DAC
        while(!SSP2STATbits.BF); //wait for Tx to complete
        if(i==3600){i=0;}
        //when all samples have been sent, start over
        CS = 1;                  //disable DAC
        high_byte = 0b01010000;
        //prepare for next time through the loop
    }
}

```



```

        }
    }
}
#include "portinit.c"
#include "interrupts.c"
#include "timer0_init.c"
#include "SSP_init.c"

////////////////////////////////////
//                               portinit.c
////////////////////////////////////
//*****//
// This will initialize the Ports for the
// PIC18F46K22 MIT TEST PLATFORM
// Pin out is for a 40 Pin Dip Package
// Internal Oscillator at 16 MHz
//*****//
void port_init(void)
{
    OSCCON = 0x72; //Default oscillator speed is 1 MHz,
                 //sets to 16 MHz
                 //B'01110010
                 //Internal Oscillator block, 16 MHz

    ANSELA = 0x00; //All digital
    ANSELB = 0x00;
    ANSELC = 0x00;
    ANSELD = 0x00;
    ANSELE = 0x00;
    //PORTA Unused
    PORTA = 0x00;
}

```

```

TRISA = 0x00;
//PORTB Unused
PORTB = 0x00;
TRISB = 0x00;
//PORTC Unused
PORTC = 0x00;    //B'00000000
TRISC = 0x00;    //B'00000000
/*
Pin # 19 - RD0 Output - SCK2
Pin # 20 - RD1 Output - /CS (MCP4922)
Pin # 27 - RD4 Output - SDO2
*/
PORTD = 0xff;    //B'11111111
TRISD = 0x00;    // B'00000000
//PORTE Unused
PORTE = 0x07;    //B'111'
TRISE = 0x07;    //B'111'
//Pin # 1 Reset input pin active low - S1
} // End of portinit.c

```

```

/////////////////////////////////////////////////////////////////
//                                     timer0_init.c
/////////////////////////////////////////////////////////////////
void timer0_init(void)
{
    T0CON = 0b10001000; //enable T0, 16 bit, internal clk,
                       // prescaler not assigned
    TMR0H = 0xd4;      //360Hz interrupt, timer0 reg counts
                       //up every instruction
                       //cycle (4xclk cycle)
}

```

```

    TMR0L = 0xb1;    //when timer0 overflows (0xFFFF to
                    //0x0000), timer0 flag is set
    INTCON2bits.TMR0IP = 1;
    INTCONbits.TMR0IE = 1;
}

void SSP_init(void)
{
    unsigned char temp;
    SSP2STAT = 0xC0;    //SMP = 1, CKE = 1
    SSP2CON1 = 0x21;    //Enable, CKP = 0,
                        //Master Fosc/16
}

////////////////////////////////////
//                                interrupts.c
////////////////////////////////////
#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
    _asm GOTO high_isr _endasm
}
#pragma code
#pragma interrupt high_isr
void high_isr (void)
{
    INTCONbits.GIE = 0b0;
    if(INTCONbits.TMR0IF)
    {
        timer0_flag = 1;
    }
}

```

```

        INTCONbits.TMR0IF = 0;
        TMR0H = 0xd4; //reset to 360Hz interrupt
        TMR0L = 0xb1;
    }
    INTCONbits.GIE = 0b1;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                     prototypes.c
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Start of Function Prototypes
void timer0_init(void);
void high_isr (void);
void port_init(void);
void SSP_init(void);
// End of Function Prototypes

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                     mydefines.h
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#define CS PORTDbits.RD1
//Configuration Bits Set in code
//You find the definitions in MPLab only
//Help -> PIC18 Config Settings -> PIC18F45K22
#pragma config FOSC = INTIO7 //Internal oscillator block,
                                //CLKOUT function on RA6,
                                //port function on RA7
#pragma config PLLCFG = OFF //Oscillator not multiplied by 4
#pragma config PRICLKEN = ON //always enabled
#pragma config FCMEN = OFF //Fail-Safe Clock Monitor disabled

```

```

#pragma config PWRTEN = OFF //Power up timer enabled
#pragma config BOREN = OFF //(SBOREN is disabled)
#pragma config BORV = 285 //VBOR set to 2.85 V nominal
#pragma config WDTEN = OFF //Watch dog timer is always
                        //disabled. SWDTEN has
                        //no effect.

#pragma config CCP2MX = PORTC1 //CCP2 input/output is
                        //multiplexed with RC1

#pragma config PBADEN = ON
#pragma config HFOFST = OFF //HFINTOSC output and ready
                        //status are delayed by the oscillator stable status
#pragma config MCLRE = EXTMCLR //MCLR pin enabled, RE3 input
                        //pin disabled

#pragma config STVREN = OFF //Stack full/underflow will
                        //cause Reset

#pragma config LVP = OFF //NO Low Voltage Program,
                        //Single-Supply ICSP disabled

#pragma config XINST = OFF //Instruction set extension
                        //and Indexed Addressing
                        //mode disabled (Legacy mode)

#pragma config DEBUG = ON //Background debugging on RB7 & RB6
#pragma config CP0 = OFF //BLOCK 0 NOT CODE PROTECTED
#pragma config CP1 = OFF //BLOCK 1 NOT CODE PROTECTED
#pragma config CP2 = OFF //BLOCK 2 NOT CODE PROTECTED
#pragma config CP3 = OFF //BLOCK 3 NOT CODE PROTECTED
#pragma config CPB = OFF //BOOT BLOCK NOT CODE PROTECTED
#pragma config CPD = OFF //DATA EEPROM NOT CODE PROTECTED
#pragma config WRT0 = OFF //BLOCK 0 NOT WRITE PROTECTED
#pragma config WRT1 = OFF //BLOCK 1 NOT WRITE PROTECTED
#pragma config WRT2 = OFF //BLOCK 2 NOT WRITE PROTECTED

```

```

#pragma config WRT3 = OFF //BLOCK 3 NOT WRITE PROTECTED
#pragma config WRTB = OFF //BOOT BLOCK NOT WRITE PROTECTED
#pragma config WRTC = OFF //CONFIGURATION REGISTER NOT
                        //WRITE PROTECTED
#pragma config WRTD = OFF //DATA EEPROM NOT WRITE PROTECTED
#pragma config EBTR0 = OFF //BLOCKS NOT PROTECTED FROM
                        //TABLE READS FROM OTHER BLOCKS
#pragma config EBTR1 = OFF
#pragma config EBTR2 = OFF
#pragma config EBTR3 = OFF
#pragma config EBTRB = OFF

```

G.3 FPGA Server Top Level Design Entity

```

module zxcv(
    input  clk_50,
    input  dout_xbee,
    input  associate_xbee,
    input  data0,
    output dclk,
    output sce,
    output sdo,
    output din_xbee,
    output reg reset_xbee,
    output reg sleep_rq_xbee,
    output [12:0]  sdram_wire_addr,
    output [1:0]  sdram_wire_ba,
    output  sdram_clk,
    output sdram_wire_cas_n,
    output sdram_wire_cke,
    output sdram_wire_cs_n,

```

```

        output [1:0] sdram_wire_dqm,
        output sdram_wire_ras_n,
        output sdram_wire_we_n,
        output sd_clk,
        inout [15:0] sdram_wire_dq,
        inout sd_data,
        inout sd_data3,
        inout sd_cmd
    );

reg reset_n;
always @ *
    reset_n = 1'b1;
always @ *
    reset_xbee = 1'b1;
always @ *
    sleep_rq_xbee = 1'b0;
//Example instantiation for system 'zxcv_system'
zxcv_system zxcv_system_inst
(
    .reset_n            (reset_n),
    .altpll_sdram       (sdram_clk),
    .altpll_sys         (),
    .areset_to_the_altpll_0  (),
    .clk_50             (clk_50),
    .MISO_to_the_mmc_spi  (sd_data),
    .MOSI_from_the_mmc_spi (sd_cmd),
    .SCLK_from_the_mmc_spi (sd_clk),
    .SS_n_from_the_mmc_spi (sd_data3),
    .data0_to_the_epcs_flash_controller_0 (data0),
    .dclk_from_the_epcs_flash_controller_0 (dclk),

```

```

.locked_from_the_altpll_0      (),
.phasedone_from_the_altpll_0  (),
.sce_from_the_epcs_flash_controller_0  (sce),
.sdo_from_the_epcs_flash_controller_0  (sdo),
.rxd_to_the_uart_0           (dout_xbee),
.txd_from_the_uart_0         (din_xbee),
.zs_addr_from_the_sdram_0    (sdram_wire_addr),
.zs_ba_from_the_sdram_0      (sdram_wire_ba),
.zs_cas_n_from_the_sdram_0   (sdram_wire_cas_n),
.zs_cke_from_the_sdram_0     (sdram_wire_cke),
.zs_cs_n_from_the_sdram_0    (sdram_wire_cs_n),
.zs_dq_to_and_from_the_sdram_0  (sdram_wire_dq),
.zs_dqm_from_the_sdram_0     (sdram_wire_dqm),
.zs_ras_n_from_the_sdram_0   (sdram_wire_ras_n),
.zs_we_n_from_the_sdram_0    (sdram_wire_we_n),
);
endmodule

```

G.4 FPGA Server User Software

```

////////////////////////////////////
////main.c
////By: Anthony Voykin M.Sc. Candidate University of
////Saskatchewan, Date: 04/2012
////////////////////////////////////
#include <stdio.h> /* Standard input/output definitions*/
#include <string.h> /* String function definitions */
#include <unistd.h> /* UNIX standard function definitions*/
#include <fcntl.h> /* File control definitions */
#include <errno.h> /* Error number definitions */

```



```

#include <termios.h> /*POSIX terminal control definitions*/

int open_port(void);
void start_of_packet(int fd);
char get_packet(unsigned char buffer[],int fd);
void combine(unsigned char buffer[], int sample[]);
int main(void)
{
    int fd,fd2,fd3,n=0,i,k;
    int HPF[250],sample[250];
    int count=0,crossover_sample=500;
    int checksum_counter=0,beat_flag=0;
    float HR[2],checksum_percentage;
    unsigned char buffer[501];    //Input buffer
    char checksum=0;
    FILE *ECG_file = NULL; //file to save ECG data
    //create asdf file
    //(actually used to delete if already exists on the sd card)
    ECG_file = fopen("/mnt/sd/asdf","w+");
    fclose(ECG_file);
    //open and configure the serial port for raw data comms.
    fd = open_port();
    //ECG_file = fopen("/mnt/sd/asdf","a+");
    //a+ option: open file for appending
    //fprintf(ECG_file,"sample(mV) HR(BPM)\n");
    //fclose(ECG_file);

    for(k=0;k<86400;k++)
        //k> x; x = number of seconds of data
        //to gather before program exit

```

```

//86400s=1day
{
    start_of_packet(fd); //wait for the packet preamble
    //get a packet and compute checksum
    checksum = get_packet(buffer,fd);
    //combine data into 250 value int array
    combine(buffer,sample);
    ECG_file = fopen("/mnt/sd/asdf","a+");
    //a+ option: open file for appending
    if(checksum == 0) //process the ECG data
    {
        for(i=0;i<250;i++)
        {
            if(i==0)
            {HPF[0] = 5*(sample[0]-crossover_sample);}
            else {HPF[i] = 5*(sample[i]-sample[i-1]);}
            //R-peak threshold (650)
            if(HPF[i] > 650) {beat_flag=1;}
            if(beat_flag && HPF[i] <0)
            {
                HR[1]=(float)count*0.004012036; //RR interval
                //(actual sample rate 0.00399975s)
                //0.004012036 used due to untuned
                //oscillator in PIC micro

                //PVC detection algorithm
                //R-R interval ratio threshold (1.4)
                if((HR[1]/HR[0]) > 1.4)
                {
                    //PVC detection algorithm
                    fprintf(ECG_file,"%i, Interesting Event

```

```

        Detected!,\n",sample[i]);
    }
    else
    {
        fprintf(ECG_file,"%i, %5.3f,\n",
            sample[i],HR[1]);
    }
    beat_flag=0;
    count=0;
    HR[0]=HR[1];
}
else fprintf(ECG_file,"%i, \n",sample[i]);
count++;
}
crossover_sample = sample[249];
}
else
//record error in ECG file for every packet checksum error
{
    checksum_counter++;
    checksum_percentage = (float)checksum_counter/
        (float) (k+1)*100;
    fprintf(ECG_file,"***INVALID CHECKSUM
        #%i out of %i (%2.0f%%),\n",
            checksum_counter,k+1,checksum_percentage);
}
fclose(ECG_file);
fd2 = open("/mnt/sd/asdf", O_RDWR);
//open fd2 in order to flush core memory to disk
fsync(fd2);

```

```

        //synchronize core memory with disk (i.e. sd card)
        close(fd2);
    }
}
//*****
//This function reads 2B from the serial port into an array
//called temp. The function continues to read bytes until the
//sequence 0x55aa is located. This sequence represents the
//start of a packet. This is a blocking operation. Inputs:
//file descriptor for an open serial port. Outputs: nil
//*****
void start_of_packet(int fd)
{
    unsigned char temp[2]={0,0};
    while(temp[0]!=0x55 || temp[1]!=0xaa)
        //wait for a start of packet notifier
        //before receiving valid data
        {
            temp[1]=temp[0];
            read(fd, &temp[0], 1);
        }
}
//*****
//This function reads 501B from the serial port into an array
//called buffer. The function also computes a 1B checksum of
//the received packet.
//Inputs: address of buffer, file descriptor for an open
//serial port
//Outputs: checksum result
//*****

```

```

char get_packet(unsigned char buffer[],int fd)
{
    int i;
    char checksum=0;
    for(i=0;i<=500;i++)
        //read 1s (1 packet) of samples plus a
        //checksum byte
        {
            read(fd, &buffer[i], 1);
            checksum = buffer[i]+checksum;
        }
    return checksum;
}

//*****
//The ecg data is transmitted in bytes, however each sample
//consists of two bytes. This function sequentially combines
//bytes, 2 at a time, from the 500B unsigned char buffer and
//places the result into a 250 value integer buffer.
//Inputs: address of unsigned char buffer, address of
//int buffer
//Outputs: nil
//*****
void combine(unsigned char buffer[], int sample[])
{
    int i,n=0;
    for(i=0;i<250;i)
        //combine buffer into sample
        {
            sample[i] = (int)buffer[n]<<8 | (int)buffer[n+1];
            //combine

```

```

        sample[i] = sample[i]*2; //2mV per step
        i++;
        n=i*2;
    }
}
//*****
//This function opens serial port 1 and also configures
//the port for raw data communications using the termios
//structure.
//Inputs: nil
//Outputs: Returns the file descriptor on success or -1 on
//error.
//*****
int    open_port(void)
{
    struct termios config;
    int fd; // File descriptor for the port
    fd = open("/dev/ttyS0", O_RDWR | O_NOCTTY | O_NDELAY);
    if (fd == -1)
    {
        //Could not open the port.
        perror("open_port: Unable to open /dev/ttyS0 - ");
    }
    else fcntl(fd, F_SETFL, 0);
    cfmakeraw(&config);
    //configure the serial port structure for raw
    //data comms. i.e. no character processing
    tcsetattr(fd, TCSAFLUSH, &config);
    //update the previous selected
    //serial port attributes

```

```

    tcflush(fd, TCIFLUSH);
    //flush data received but not read
    return (fd);
}

```

G.5 Digital Transmitter Firmware

```

/////////////////////////////////////////////////////////////////
////main.c
////By: Anthony Voykin M.Sc. Candidate University of
//// Saskatchewan, Date: 04/2012
/////////////////////////////////////////////////////////////////
////Description:
////This project is to be used with the XBee mote designed
//// and built for my masters thesis. At a high level, this
//// project will:
//// -take an A/D sample every 4ms and place the result
////      (2B - high and low) into two, 250B sample buffers
//// -Tx a 1 second packet of samples (250 samples, 500B)
////      at 115k2 BAUD to another XBee module
//// -During Tx, samples will still be gathered every 4ms
////      and placed in a temp. buffer, the sample clock will
////      be interrupt driven via TMR0
//// -After completion of Tx, the temp. buffer will be
////      copied to the sample buffer
/////////////////////////////////////////////////////////////////
#include <p18LF45K22.h> //Includes microcontroller specific
                        //predefines
#include "prototypes.h" //All function prototypes
#include "mydefines.h" //Includes my defines, fuses and
                        //global variables

```

```

//Global variables
int t=0,rx_flag=0,copy_done=0,timer0_flag=0,n=0,noflow=0,
    tmr0_flag=0,tx_flag=0;
char rx_buffer;
#pragma udata gpr0 //Each user data page is limited to 256B,
char buffer_L[250]; //use pragma directive to force data
                    //to a particular page
#pragma udata      //6 pages are available in this processor
#pragma udata gpr1
char buffer_H[250];
#pragma udata
#pragma udata gpr2
char temp_buffer_H[250];
#pragma udata
#pragma udata gpr3
char temp_buffer_L[250];
#pragma udata
//END Global variables
void main(void)
{
    int i;
    char checksum;
    port_init();
    serial_init(); //initialize the UART
    ADCinit(0x0e); //init ADC with channel 14 (RC2)
    timer0_init();
    INTCONbits.PEIE = 1; //enable interrupts so that comms.
    //with XBee module during initialization can be moderated
    INTCONbits.GIE = 1;

```



```

while(1)
{
    PORTDbits.RD3 = 1; //put XBee module to sleep
    //Communication section
    //Send 1s worth of samples to another XBee module
    if(tx_flag)
    {
        //wake up XBee module
        PORTDbits.RD3 = 0;
        //wait for xbee to wake up
        while(!PORTDbits.RD1){;}
        TXREG1 = 0xaa;
        //send a start of packet notifier to server
        while(!TXSTA1bits.TRMT){;}
        TXREG1 = 0x55;
        while(!TXSTA1bits.TRMT){;}
        for(i=0;i<=249;i++) //send samples to server
        {
            TXREG1 = temp_buffer_H[i];
            while(!TXSTA1bits.TRMT){;}
            TXREG1 = temp_buffer_L[i];
            while(!TXSTA1bits.TRMT){;}
            checksum = temp_buffer_H[i] + temp_buffer_L[i]
                + checksum;
        }
        checksum = ~checksum;
        checksum++;
        TXREG1 = checksum; //send checksum
        while(!TXSTA1bits.TRMT){;}
        checksum=0;
    }
}

```

```

        tx_flag=0;
    }
}

#include "serial_init.c"
#include "portinit.c"
#include "interrupts.c"
#include "timer0_init.c"
#include "ADConvert.c"

////////////////////////////////////
////                               interrupts.c
////////////////////////////////////
#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
    _asm GOTO high_isr _endasm
}
#pragma code
#pragma interrupt high_isr
void high_isr (void)
{
    int i;
    INTCONbits.GIE = 0;
    if(INTCONbits.TMR0IF)
    {
        INTCONbits.TMR0IF = 0; //clear int.
        TMR0H = 0xe0;
        //set to 4.014750ms second interval
        TMR0L = 0xcf;
    }
}

```

```

        ADCON0bits.GO = 1; //AD conversion
        while(ADCON0bits.NOT_DONE == 1);
        //wait for completion
        buffer_H[t] = ADRESH;
        //put AD result in temp. buffer packet
        buffer_L[t] = ADRESL;
        t++;
        //count the number of values in temp. buffer
        if(t==250)
        {
            for(i=0;i<250;i++)
            {
                temp_buffer_H[i] = buffer_H[i];
                temp_buffer_L[i] = buffer_L[i];
            }
            t=0;
            tx_flag=1;
        }
    }

    INTCONbits.GIE = 1;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                                     portinit.c
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//*****
// This will initialize the Ports for the PIC18F45K22
//      (TQFP)
// Pin out is for a 44 Pin QFP Package
// Internal Oscillator at 16 MHz

```

```

//*****
void port_init(void)
{
    OSCCON = 0b01110010;//oscillator speed is 16 MHz
                //B'01110010
                //Internal Oscillator block, 16 MHz

    ANSELA = 0x00; //all digital
    ANSELB = 0x00; //all digital
    ANSELC = 0b00000100;//RC2 analog, rest digital
    ANSELD = 0x00; //all digital
    ANSELE = 0x00; //all analog
    PORTA = 0xff; //B'11111111'
    TRISA = 0x00; //B'00000000'
    PORTB = 0xff; //B'11111111'
    TRISB = 0x00; //B'00000000'
    //Pin # 36 - RC2 Analog Input from ECG Analog Front End
    //Pin # 42 - RC4 Input - Xbee - RSSI
    //Pin # 43 - RC5 Output - Xbee - !Reset
    //Pin # 44 - RC6 Input - Serial TX - Xbee DIN
    //Pin # 1 - RC7 Input - Serial RX - Xbee DOUT
    PORTC = 0x2e; //B'00101110'
    TRISC = 0xd5; //B'11010101'
    //Pin # 39 - RD1 Input - Xbee - Sleep status (!Sleep/ON)
    //Pin # 40 - RD2 Input - Xbee - Associate
    //Pin # 41 - RD3 Output - Xbee - Sleep Request
    PORTD = 0xf1; //B'11110001'
    TRISD = 0x06; // B'00000110'
    PORTE = 0x03; //B'111'
    TRISE = 0x04; //B'100'
} // End of portinit.c

```

```

////////////////////////////////////
////                               serial_init.c
////////////////////////////////////
// No autobaud and normal operation, Fosc = 16 MHz
void serial_init(void)
{
    BAUDCON1 = 0x08;
    //0X001X00 Autobaud off, regular polarity, 16 bit BRGenerator
    RCSTA1 = 0x90;
    //10X1XXXX SPEN, 8 bits, Continuous receive
    // SPBRG1 = 0x22;
    //Baud rate = Fosc/(4(n+1)); n = (16MHz/(115k2 * 4)) - 1
    // SPBRG1 = 0x44;
    //Baud rate = Fosc/(4(n+1)); n = (16MHz/(57k6 * 4)) - 1
    // SPBRG1 = 0x67;
    //Baud rate = Fosc/(4(n+1)); n = (16MHz/(38k4 * 4)) - 1
    SPBRG1 = 0xcf;
    //Baud rate = Fosc/(4(n+1)); n = (16MHz/(19k2 * 4)) - 1
    SPBRGH1 = 0x0;
    TXSTA1 = 0x24;
    //X01001XX 8 bit, enable, Async, High Baud rate
    PIE1bits.RC1IE = 1;
    //enable interrupt - int. when input buffer is full
}

```

```

////////////////////////////////////
////                               timer0_init.c
////////////////////////////////////
void timer0_init(void)

```

```

{
    T0CON = 0b10000000;
    //enable T0, 16 bit, internal clk, prescaler = 1/2
    TMR0H = 0xe0;
    //set to 4ms interval (4cycles/inst * 1/16MHz *2(prescaler)
    // * (0xffff-0xe0b1))
    TMR0L = 0xcf;
    INTCONbits.TMR0IP = 1;
    INTCONbits.TMR0IE = 0;//do not enable
    INTCONbits.TMR0IF = 0;
    timer0_flag = 0;
    INTCONbits.TMR0IE = 1;
}

```

```

////////////////////////////////////
////                                     ADConvert.c

```

```

////////////////////////////////////
//ADCinit will set the ADC channel and turn on the ADC.

```

```

//The calling function will send the channel.

```

```

//i.e. for AN7 send 0x07

```

```

void ADCinit(int channel)

```

```

{
    ADCON1bits.PVCFG = 0b10; //set pos vref to FVR BUF2
    ADCON1bits.NVCFG = 0b00; //set neg vref to AVSS
    VREFCON0bits.FVRS = 0b10; //Set FRV to 2.048V
    VREFCON0bits.FVREN = 1; //Enable Fixed Voltage Reference
    ADCON2bits.ADFM = 1; //right justified
    channel = channel << 2;
    //channel occupies bits b6-b2 of ADCON0
    ADCON0 = 0b00000001 | channel;
}

```

```

        //set AD channel and enable ADC
        PIE1bits.ADIE = 0;

        //AD conversion interrupt not used for this project
        ADCON2bits.ADCS = 0b110; //set ADC clock to Fosc/64
    }

//ADConvert starts a conversion and waits for completion
//prior to proceeding
int ADConvert(void)
{
    ADCON0bits.GO = 1;
    while(ADCON0bits.NOT_DONE == 1){;} //blocking code!
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////                                     mydefines.h
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Configuration Bits Set in code
//Help - PIC18 Config Settings - PIC18F45K22
#pragma config FOSC = INTIO67    //Internal oscillator block,
//port function on RA6 and RA7
#pragma config PLLCFG = OFF //Oscillator not multiplied by 4
#pragma config PRICLKEN = OFF
//Primary clock is always enabled
#pragma config FCMEN = OFF
//Fail-Safe Clock Monitor disabled
#pragma config PWRTEN = OFF //Power up timer disabled
#pragma config BOREN = OFF //SBORDIS
//Brown-out Reset enabled in hardware only
#pragma config BORV = 190    //VBOR set to 1.90 V nominal

```

```

#pragma config WDTCN = OFF
//Watch dog timer is always disabled
#pragma config WDTPS = 1
#pragma config CCP2MX = PORTC1
//CCP2 input/output is multiplexed with RC1
#pragma config PBAEN = OFF
#pragma config HFOFST = OFF
//HFINTOSC output and ready status are delayed by the
//oscillator stable status
#pragma config MCLRE = EXTMCLR
//MCLR pin enabled, RE3 input pin disabled
#pragma config STVREN = OFF
//Stack full/underflow will cause Reset
#pragma config LVP = OFF
//NO Low Voltage Program, Single-Supply ICSP disabled
#pragma config XINST = OFF
//Instruction set extension and Indexed Addressing
//mode disabled (Legacy mode)
#pragma config DEBUG = ON
//Background debugging on RB7 and RB6
#pragma config CP0 = OFF //BLOCK 0 NOT CODE PROTECTED
#pragma config CP1 = OFF //BLOCK 1 NOT CODE PROTECTED
#pragma config CP2 = OFF //BLOCK 2 NOT CODE PROTECTED
#pragma config CP3 = OFF //BLOCK 3 NOT CODE PROTECTED
#pragma config CPB = OFF //BOOT BLOCK NOT CODE PROTECTED
#pragma config CPD = OFF //DATA EEPROM NOT CODE PROTECTED
#pragma config WRT0 = OFF //BLOCK 0 NOT WRITE PROTECTED
#pragma config WRT1 = OFF //BLOCK 1 NOT WRITE PROTECTED
#pragma config WRT2 = OFF //BLOCK 2 NOT WRITE PROTECTED
#pragma config WRT3 = OFF //BLOCK 3 NOT WRITE PROTECTED

```



```
#pragma config WRTB = OFF //BOOT BLOCK NOT WRITE PROTECTED
#pragma config WRTC = OFF
//CONFIGURATION REGISTER NOT WRITE PROTECTED
#pragma config WRTD = OFF //DATA EEPROM NOT WRITE PROTECTED
#pragma config EBTR0 = OFF
//BLOCKS NOT PROTECTED FROM TABLE READS FROM OTHER BLOCKS
#pragma config EBTR1 = OFF
#pragma config EBTR2 = OFF
#pragma config EBTR3 = OFF
#pragma config EBTRB = OFF
```

```
////////////////////////////////////
////                                     prototypes.h
////////////////////////////////////
void timer0_init(void);
void port_init(void);
void serial_init(void);
void high_isr (void);
int ADConvert(void);
void ADCinit(int channel);
```

H. System Specifications

This Appendix highlights the key specifications of the complete BSN system designed in this work.

FPGA Server:

- DE0-Nano Development and Education Board
 - USB Blaster for programming/debugging
 - EPCS serial configuration flash memory device
 - 32MB SDRAM
 - 2kb I2C EEPROM
 - 12b ADC (50ksps to 200ksps)
 - 8 LEDs, 2 debounced push buttons, 4 position DIP switch
 - 3-axis accelerometer
 - 50MHz crystal oscillator
 - USB and external header for power (3.6 to 5.7V range)
 - 2x40 pin and 1x26 pin expansion headers
 - Altera Cyclone IV EP4CE22F17C6N FPGA
 - * 22320 Logic Elements
 - * 594k bits of embedded memory
 - * 66 multipliers (18x18bit)
 - * 4 PLLs

- * 153 user I/O pins
- * 20 global clock networks

- NIOS II Processor

- NIOS II/f core (1400-1800 logic elements, 3x9k embedded SRAM and cache)
- Configuration loads from EPCS flash memory
- Core clock rate of 50MHz
- 32MB SDRAM controller
- JTAG UART
- Interval timer (1ms timeout, 32b counter)
- SPI for SD Card @ 128kHz clock rate
- PLL for phase shifted SDRAM clock
- RS232 UART @ 19k2 BAUD rate, 8N1 (XBee RF module interface)
- Runs uClinux 2.6.30

ZigBee Mote:

- PIC18LF45K22 Microchip microcontroller

- 44 pin Thin Quad Flat Pack
- Programmed in C
- 32kB program memory
- 16MHz clock
- 3V power supply
- 35 I/O pins plus 1 input only pin
- UART set to 19k2 BAUD (XBee RF module interface)
- 10b ADC

- XBee Radio
 - Meets IEEE 802.15.4 standard
 - Operates within ISM band (2.4GHz)
 - 30m range indoors, 90m outdoors
 - 3V operating voltage
 - 50mA Rx current, 45mA Tx current
 - Sleep current of 50uA using pin doze function (2ms wake up time)

- LTC1983-3
 - -3V DC to DC converter
 - programmable on/off for low power

I. Truncation Noise

This Appendix provides an example of the truncation noise added to the output of the test system Digital to Analog Converter when using MIT-BIH data file 223 from 1:00 minutes to 1:10 minutes.

The analysis of data file 223 from 1:00 minutes to 1:10 minutes produces a signal-to-noise ratio (SNR) of 76.1dB. This was calculated by determining the mean squared error (MSE) between the actual floating point number given by F , and the truncated result given by T . The MSE is the noise power resulting from truncation and is given by

$$\text{MSE} = \tau^2 = \frac{\sum_{n=1}^N (F_n - T_n)^2}{N} \text{ (volts)}. \quad (\text{I.1})$$

The signal voltage is calculated using the root mean squared (RMS) equation

$$\text{RMS} = \sqrt{\frac{\sum_{n=1}^N T_n^2}{N}} \text{ (volts)}. \quad (\text{I.2})$$

Equation I.2 The square of the RMS value is the signal power. Finally, the SNR in decibels was calculated using

$$\text{SNR} = 10 \log_{10} \frac{\text{RMS}^2}{\tau^2} \text{ (dB)}. \quad (\text{I.3})$$