2005

Reconfigurable wireless sensor platform for training and research in networked embedded systems

Oscar V. Gonzalez
University of South Florida

Follow this and additional works at: http://scholarcommons.usf.edu/etd

Part of the American Studies Commons

Scholar Commons Citation
Graduate Theses and Dissertations.
http://scholarcommons.usf.edu/etd/2900

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
Reconfigurable Wireless Sensor Platform for Training and Research in Networked Embedded Systems

By

Oscar V. Gonzalez

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering Department of Electrical Engineering College of Engineering University of South Florida

Major Professor: Wilfrido A. Moreno, Ph.D.
James T. Leffew, Ph.D.
Miguel Labrador, Ph.D.

Date of Approval:
April 12, 2005

Keywords: WSN, FPGA, Motes, Remote Sensing, Hostile Environments

©Copyright 2005, Oscar V. Gonzalez
DEDICATION

I would like to thank my parents and my sisters because of their support and their motivation. My motivation was increased as they passed theirs to me. Without the energy impressed upon me by my mother I could not have reached this goal.

I want very much to express my appreciation and love for my wife and true friend. I will be forever thankful for her patience, her love and for her unwavering belief in me, even when every thing looked very dark. Her presence uplifted and sustained me.

I want to thank God and the Madonna for giving my wife and I our son. He is the best gift of our life and he has brought to us unimaginable happiness.

I also want to dedicate this thesis to my grandmother my uncles and my aunts.

I would also like to show my gratitude to all of my cousins; especially, Elver, Edgar, Miguel A and John Freddy.

I would like to recognize, in a very special way, Dr. Wilfrido Moreno for helping me to complete my studies at the Masters Level. I will always remember him in my orations to God.

I want to thank to Dr. Miguel Labrador and Dr. James Leffew for their advice and understanding.

I appreciate all the time that Mauricio Castillo and Daniel Quintela spent with me in discussions concerning the project.

I will appreciate forever the motivation of my friends Leandro, Jorge and Alex.
I want to thank Jay for helping me with the English corrections in this thesis.

Finally, I appreciate the collaboration of all of my friends here in Tampa.
# TABLE OF CONTENTS

**LIST OF TABLES**

v

**LIST OF FIGURES**

vi

**ABSTRACT**

ix

**CHAPTER 1 INTRODUCTION**

1.1 Problem Statement

1.1.1 Interoperability

4

1.1.2 Upgrading

5

1.1.3 Maintenance

5

1.1.4 Operation in Hostile and Very Hard Access Environment

6

1.1.5 Power Consumption

6

1.1.6 Low Power Reconfigurable Devices

7

1.1.7 Constraints Related to Synthesis Tools

7

1.2 Research Scope and Contributions

7

1.3 Thesis Organization

8

**CHAPTER 2 RELATED WORK**

10

2.1 Field Programmable Wireless Sensor Network Nodes

10

2.2 PicoRadio and Pico Beacon

12

2.3 iMote 10

13

2.4 Small Scale Adaptive Processing

15

2.5 Wireless Protocol Processor

15

2.6 Software Reconfigurability

16

2.6.1 Over the Air Programming (OAP)

16

**CHAPTER 3 AN OVERVIEW OF WIRELESS SENSOR NETWORKS**

18

3.1 Introduction

18

3.2 Characteristics

19

3.3 Architecture of Wireless Sensor Networks

20

3.3.1 Standard Mote Architecture

21

3.4 Physical Layer in Wireless Sensor Networks

22

3.4.1 Overview

22

3.4.2 Overview of the IEEE 802.15.4, “Zigbee” Standard

23

3.5 Mesh and Multihop Routing

24

3.5.1 Mesh Routing

24

3.5.2 Multihop Routing

25

**CHAPTER 4 REAL TIME OPERATING SYSTEMS FOR A WSN**

26

4.1 Introduction

26

4.2 TinyOS

27

4.2.1 TinyOS Concept

27
CHAPTER 4 TOSSIM: TinyOS Simulator

4.2.2 TinyOS and NesC
4.2.3 Concepts of NesC
4.2.4 Component-Based Architecture
4.2.5 Concurrency
4.2.6 Design with NesC
4.2.7 Interfaces
4.2.8 Atomicity
4.2.9 TinyOS Data Link layer

4.3 TOSSIM: TinyOS Simulator

4.3.1 Introduction
4.3.2 TOSSIM Architecture
4.3.3 TOSSIM Network Simulation
4.3.4 POWER TOSSIM

4.3.4.1 Micro-Benchmarks
4.3.4.2 Power TOSSIM Architecture

CHAPTER 5 RECONFIGURABLE DEVICES FOR THE PROPOSED ARCHITECTURE

5.1 Introduction
5.2 Types of programmable Logic
5.3 Simple Programmable Logic Devices, (SPLDs)

5.3.1 Programmable Read Only Memory(PROM)
5.3.2 Programmable Logic Array (PLAs)
5.3.3 Programmable Array Logic/Generic Array Logic(PAL/GAL)
5.3.4 Macrocell

5.4 Complex Programmable Logic Devices (CPLDs) for the Reconfigurable Platform

5.4.1 Xilinx’s CPLD: CoolRunner
5.4.1.1 Fast Zero Power (FZP) Technology
5.4.1.2 CoolRunner Architecture
5.4.2 Lattice’s CPLD: ispMACH
5.4.3 Atmel’s CPLD: ATF15xx
5.4.4 Comparison of the CPLDs for the Reconfigurable Mote

5.5 Field Programmable Gate Arrays, (FPGAs)

5.5.1 Categories of FPGAs
5.5.1.1 Static Random Access Memory, (SRAM), FPGA Based
5.5.1.2 Antifuse Based FPGA
5.5.1.3 Electrically Erasable PROM, (EEPROM), FPGA Based

5.5.2 Design of a Low Power FPGA

5.6 Field Programmable Gate Arrays, (FPGAs), for the Reconfigurable Platform

5.6.1 Quick Logic Eclipse II FPGAs
5.6.1.1 Logic Cell
5.6.1.2 RAM Modules
5.6.1.3 Embedded Computational Unit
5.6.1.4 I/O Cell Structure
5.6.1.5 Low Power Consumption

5.6.2 Altera Cyclone FPGAs
5.6.2.1 Logic Elements (LE)
5.6.2.2 Logic Array Blocks (LABs)
5.6.2.3 Embedded Memory

5.6.3 Xilinx Spartan-3L Low Power FPGAs
5.6.3.1 Configurable Logic Blocks (CLBs)
5.6.3.2 Input/Output Blocks
5.6.3.3 Block RAM
5.6.3.4 Multiplier Blocks
5.6.3.5 Digital Clock Manager (DCM) Blocks
5.6.3.6 Low Power Operation
5.6.4 Actel ProASIC Plus FPGAs
  5.6.4.1 Logic Tile 65
  5.6.4.2 Routing Resources 66
  5.6.4.3 Flash Switch 67
  5.6.4.4 Input/Output Blocks 68

5.6.5 Comparison of the FPGAs for the Reconfigurable Mote 68
5.7 Most Suitable Programmable Devices for the Reconfigurable Platform 68
  5.7.1 Capacity 69

CHAPTER 6 PROPOSED RECONFIGURABLE PLATFORM 70
  6.1 Overview 70
  6.2 Requirements for Hostile Outdoor Environment Applications 70
  6.3 Proposed Mote Architecture Concept 72
    6.3.1 Antennas 72
    6.3.2 RF Transceiver 73
      6.3.2.1 Radio Architecture 74
      6.3.2.2 Baseband Processing Block 74
      6.3.2.3 I/Q Modulator 75
    6.3.3 Processing Unit 76
    6.3.4 Sensor Signal Adaptation 77
  6.4 Software Programmability and Reconfigurability 79
    6.4.1 Software Programmability 79
    6.4.2 Software Reconfigurability 80
      6.4.2.1 OAP Strategy 81
      6.4.2.2 Operation Roles 82

CHAPTER 7 POWERING MODEL OF THE PROPOSED PLATFORM 83
  7.1 Introduction 83
  7.2 TinyOS Scheduler 83
  7.3 Duty Cycle 84
  7.4 Power Model of the Mica2 Motes 85
  7.5 Power Supply for Motes 87
  7.6 Rechargeable Batteries 87
  7.7 Power Management Strategies for the Proposed Mote 88
    7.7.1 Energy Harvesting Sources 89
  7.8 Architecture of the Proposed Powering Module 92
    7.8.1 DC-DC Converter 92
    7.8.2 Solar Panel 93

CHAPTER 8 IMPLEMENTATION AND TESTBED FOR THE PROPOSED PLATFORM 94
  8.1 Introduction 94
    8.1.1 Test and Development Configurations 95
      8.1.1.1 Reconfigurable Sensing Configuration 96
      8.1.1.2 Reconfigurable Processing and Sensing Configuration 98
      8.1.1.3 Additional Processing Configuration 99
      8.1.1.4 DSSS and Zigbee Reconfigurable Radio Configuration 100
      8.1.1.5 Fully Reconfigurable Radio Configuration 103
    8.2 Proof of Concept 105
      8.2.1 Simulation Environment 105
      8.2.2 Field Test 107
      8.2.3 FPGA Functionality 108
      8.2.4 Development Environment Setup 109

CHAPTER 9 CONCLUSION AND FUTURE WORK 110
REFERENCES

APPENDICES

Appendix A VHLD Code for the AtMega128L
Appendix B VHLD Code for Sensor Signal Adaptation Block
LIST OF TABLES

Table 1: Evolution of Sensor Nodes                                      19
Table 2: CPLD’s Power Consumption Comparison                           49
Table 3: FPGA’s Power Consumption Comparison                           62
Table 4: Required Capacity for the Reconfigurable Devices             69
Table 5: Effect Throughput and Maximum Number of Packets vs Duty Cycle 85
Table 6: Power Consumption in the Mica2                                86
Table 7: Energy-Harvesting Opportunities                              91
Table 8: Modules Activation Matrix for the Proposed Solution           104
LIST OF FIGURES

Figure 1: Evolution of the Computing Technology 3
Figure 2: Architecture of the NMRC Mote 11
Figure 3: Picture of an Actual NMRC Mote 11
Figure 4: Architecture of the PicoRadio 12
Figure 5: First Prototyping of the PicoBean Mote 13
Figure 6: Prototype of the Intel imote 14
Figure 7: TinyOS Layers Implemented in Intel iMote 14
Figure 8: A High Level Depiction of the SNAP 16
Figure 9: Architecture of a Wireless Sensor Network 21
Figure 10: Standard Mote Architecture 22
Figure 11: Coverage Map without Mesh Routing 24
Figure 12: Coverage Map with Mesh Routing 25
Figure 13: TinyOS Architecture 29
Figure 14: TinyOS Model 30
Figure 15: TinyOS Packet Format 32
Figure 16: TOSSIM Architecture: Frames, events and models 35
Figure 17: PLD Architecture 39
Figure 18: CPLD Architecture 41
Figure 19: Product Term in Traditional CPLDs 43
Figure 20: Basic Architecture of the FZP 43
Figure 21: 8-input AND Function using FZP Technology 44
Figure 22: Architecture of CoolRunner™ II CPLD 45
Figure 51: Life Time Vs Battery Capacity 88
Figure 52: Continuous Compared with Intermittent Discharge 89
Figure 53: Architecture of the Powering Module for the Proposed Mote 92
Figure 54: Solar Panel Employed for the Proposed Mote 93
Figure 55: Testbed for Integrating the Proposed Reconfigurable Platform 94
Figure 56: Architecture of the Reconfigurable Sensing Configuration 96
Figure 57: Power Consumption of a Microcontroller Without Signal Adaptation Block 97
Figure 58: Power Consumption of a Microcontroller Using a Signal Adaptation Block 97
Figure 59: Architecture of the Reconfigurable Processing and Sensing Configuration 98
Figure 60: Incorporation of an Additional Processing Unit to the Reconfigurable Mote 100
Figure 61: System Generator Design Flow 102
Figure 62: Design of a DQPSK Modulator Using the Xilinx System Generator 103
Figure 63: Architecture of the Fully Reconfigurable Mote 104
Figure 64: Simulation of a WSN of 10 motes using TOSSIM 106
Figure 65: Power Consumption Comparison of a WSN of 10 motes using TOSSIM 106
Figure 66: Field Test in Venezuela 107
Figure 67: Message Date Rate Statistics 108
Figure 68: Simulation of the Functionality of the ATMega128L Core 108
Figure 69: Development Setup Architecture 109
Figure 70: Base Architecture for the Atmega128L Written in VHDL 117
RECONFIGURABLE WIRELESS SENSOR PLATFORM FOR TRAINING
AND RESEARCH IN NETWORKED EMBEDDED SYSTEMS

Oscar V. Gonzalez

ABSTRACT

Ever since the University of California, Berkeley released the first commercial Wireless Sensor Network, (WSN), "mote", applications that employ the WSN technology have increased many fold. There are many kinds of prototypes and architectures for WSNs that are being developed by major communication companies. Due to the lack of a common standard for different physical and MAC layer protocols, interoperability among the wireless systems is constrained. Additionally, remote sensing capability, reconfigurability and interoperability have not yet been designed. In the current platforms hardware upgrading has to be performed on-site by replacing the old sensors with new ones. This procedure increases considerably the costs of deployment and maintenance. On-site upgrading also imposes serious constraints on applications that operate in very limited access environments. Due to such constraints, it is imperative that Wireless Sensor Networks platforms be developed that solve these challenges, including the remote environmental sensing.

This research focused on developing a conceptual solution using an embedded based reconfigurable platform for Wireless Sensor Networks. The design of the integrated wireless sensor network proposed in this thesis deals with features such as
ultra low power embedded solutions, software defined radio techniques and power management strategies to enable maximum system autonomy. In addition, this thesis presents the design of a development platform along with a test-bed configuration. The environmental sensing for disaster mitigation is used as a case study throughout this work. The data for this project was obtained from a project involving the use of Wireless Sensor Networks for flash-flood detection and an early-warning alert system
CHAPTER 1

INTRODUCTION

The research activity in the area of wireless sensor networks has increased considerably ever since the University of California, Berkeley’s motes were commercialized by the Crossbow Company, [1]. These motes are constantly used in current projects as well as in new applications, [2], [3]. The reason for the rapid growth in this area is attributed to the affordable price of the sensors due to the advances in major areas of the wireless sensor components such as sensing, communication and computation. The applications of sensor networks span fields such as environmental sensing, habitat monitoring, surveillance, military sensing, traffic control, manufacturing automation, inventory, distributed controls, building & structuring monitoring and disaster management & emergency response, [4].

Several wireless protocols have been employed in wireless sensor networks and each manufacturer has their preferred physical layer standard. The current Wireless Sensor Networks operate in the Industrial, Scientific and Medical, (ISM), bands at frequencies of 433 MHz, 900 MHz and 2.4 GHz. The most common modulation technique is Direct-Sequence Spread Spectrum, (DSSS), due to its high power efficiency compared to other technologies, [5]. Companies such as Crossbow and Csiro have used DSSS in their Mica2 and Fleck motes, [1], [6]. As a matter of fact, the new emerging
standards such as Zigbee and Ultrawideband, (UWB), are based on DSSS as well. However, technologies such as Frequency Hopping Spread Spectrum, (FHSS), and Bluetooth are also being used by many other companies.

The Zigbee standard presently appears to be the leader in the low-power short-range approach in Wireless Personal Area Networks, (WPAN), for indoor environments. For instance, companies such as Crossbow have launched their Zigbee mote called MicaZ and Ember has released their Zigbee platform called EM2420 into the market. There are many different WSN platform alternatives that have been developed such as UCLA’s iBadge, UCLA’s Medusa MK-II, Berkeley’s Motes, Berkeley’s Piconodes, MIT’s µAMPS and Intel’s i-Mote, [7], [9]. Every approach has its advantages and disadvantages with respect to cost, power consumption and functionality. It is not easy to identify the ideal platform for every application since different embedded processors can be and are used and their power efficiencies are application dependent. For instance, the processing and range requirements for the motes influence the overall performance of the network. Therefore, the introduction of an adaptive and reconfigurable platform is critical to ensure maximum autonomy of the motes within a deployed system. The processing capability has to be reconfigurable and the operating frequency has to be tunable. A reconfigurable WSN platform must also maximize power efficiency and facilitate maintenance and upgrade tasks.

A real time operating system is a vital component of a Wireless Sensor node. The TinyOS operating system appears to be the strongest candidate for WSN providers, [8]. Companies such as Intel and Crossbow have incorporated TinyOS in their Mica and iMote product families, [9]. Since many of the development and simulation tools are
based on TinyOS it has become a pseudo-standardization. However, other operating systems are also being used. For instance, the Swedish Institute of Computer Science has developed a new operating system called Contiki, [10]. Companies such as Rockwell, Sensoria and Ember have their own proprietary real time operating systems. Figure 1 graphically illustrates the evolution of present day computing capability.

![Figure 1: Evolution of Computing Technology](image)

Another important issue is the power consumption of the motes, which limits the processing and the signal transmission of the deployed system. Even though the design of the processors follows Moore’s Law, with respect to computing capabilities, the design of efficient batteries lags seriously. Even though new low power consumption strategies are being developed, it is still necessary, in many cases, to change the battery on-site, which is not an option for applications that operate in environments that are difficult to access. Low-power strategies such as lower duty cycle transmissions, standby modes, sleep modes and the use of ultra low power microcontrollers are being used to handle the issue of power management. In addition to power minimization the latest research, which opens other alternatives for powering the wireless sensor nodes, is being directed.
to alternative energy sources such as vibrations, acoustic noise, thermal conversion, nuclear reaction and fuel cells.

1.1 Problem Statement

The applications for WSNs are numerous. These applications range from a simple sensor network for temperature monitoring in a room to a complex sensor network for military target sensing system, [4]. The sensor motes can be deployed in indoor environments as well as in hostile and harmful outdoor scenarios such as a tropical forest, a desert or in the deep sea. Therefore, the processing capability, power consumption and data rate requirements for each kind of sensor network differ greatly. In order to meet system requirements vendors and the manufacturers of WSNs have been developing numerous application-oriented nodes to fulfill these constraints, [1], [6], [9], [11]. Several motes have been designed based on different architectures. However, issues such a as interoperability, radio adaptability, remote maintenance and upgrading have not yet been solved.

1.1.1 Interoperability

This problem arises in the most basic scenarios such as when two Mica2 motes from Crossbow need to be interconnected with one mote operating at 900 MHz and the other mote operating at 433 MHz. With the current technology it is not possible to interconnect them because of the fixed band antenna and the fixed elements used for the RF module, CC1000, from Chipcon, [12]. The situation worsens when the
communication protocols are different. Such would be the case if a Mica2 mote from Crossbow and a Fleck mote from Csiro needed to be interconnected.

1.1.2 Upgrading

The research associated with WSN devices is intensive and highly time-constrained. As a consequence, the development and release cycle of new improved devices is less than a year. For instance, the first mote from the University of California, Berkeley, called "Rene", was released in 1999 followed by Mica in 2000 and Mica2 in 2001. Each of these motes possess the same processing core but have differences related to memory and an RF stack. If the original motes had been designed based on a reconfigurable system the same hardware of the Rene platform could have been used to upgrade to the Mica and Mica2 successor motes. A reconfigurable system would have reduced the deployment and additional hardware costs.

1.1.3 Maintenance

Maintenance is one of the most important issues in the deployment of Wireless Systems. Maintenance consumes a great portion of the budget used for the operation of a network. However, this issue was not considered in the current platforms. Many of the commercial applications are deployed in outdoor environments and thousands of motes are installed to cover broad areas. In such cases the action of accessing each node becomes an expensive operation since each mote has to be reached individually. If the cost is proportional to the number of deployed nodes the maintenance costs will equal or
exceed the deployment costs. In some specific applications such as in a warfare scenarios access to the motes is restricted by safety reasons that make maintenance a very difficult task.

1.1.4 Operation in Hostile and Very Hard to Access Environments

Most of the applications for a WSN are related to environmental monitoring. For instance, in natural disaster mitigation applications the sensors have to be deployed in the river basins, in dense forests or just in a simple meadow. Therefore, the sensors must operate in areas surrounded by very dense vegetation, humidity, and fauna. Some of the sensors can get lost and others can be damaged. Therefore, access to each and every sensor becomes a very time consuming task and an expensive process that directly affects the reliability and flexibility of the network.

1.1.5 Power Consumption

Power consumption is still an open issue since there are many factors that impact this critical consideration. For instance, in the Mica2 motes, sensor signal processing is not power efficient since all the sensor signals are connected directly to the microcontroller, which increases the power consumption due to a higher number of interrupts and frequent write operations to flash memory. New strategies for reducing the power consumption have to be researched, designed and implemented.
1.1.6 Low Power Reconfigurable Devices

The major manufacturers of reconfigurable devices such as Altera, Xilinx and Actel have still not considered the Wireless Sensor Networks applications as a major market. Thus the majority of the current reconfigurable devices do not have sleep and standby modes that enable to attain the power efficiency as compared to the conventional microcontroller based devices developed for this kind of applications. For instance, while the microcontroller ATMega128L has a power consumption, on power-down mode, of 80 µW the Altera Cyclone FPGA device has a power consumption of 48 mW, which is six hundred times more than the microcontroller power consumption, [13]. Therefore, it is important to minimize the power consumption of the programmable devices in order to be used for low power Wireless Sensor Networks applications.

1.1.7 Constraints Related to Synthesis Tools

Adding to the power consumption problems of the reconfigurable systems are problems associated with synthesis tools. Synthesis tools such as Quartus and MaxPlus II from Altera and ISE from Xilinx, among others, do not consider the power as a constraint in the synthesis process. Therefore, there is no opportunity to address the problem of power optimization automatically through the synthesis tool.

1.2 Research Scope and Contributions

This research focused on the development of a conceptual solution using an embedded system based reconfigurable platform for Wireless Sensor Network
applications. The approach involved the evaluation of the requirements and the design of a suitable system based on a reconfigurable platform. The main issues dealt during this research were hardware reconfigurability, remote maintenance, system upgrading, multi-band operation and interoperability. As part of this research, a full literature review and field tests were performed to evaluate current platforms for WSN applications and to establish their limitations. In addition, an evaluation of the most suitable embedded system for the proposed concept was also performed in conjunction with identification and evaluation of different power management strategies related to the design of an effective autonomous system. The design of the development platform included a test-bed configuration for the validation of the proposed concept that used an early warning and an alert system called Rapid Organization and Situation Assessment, (ROSA), [38].

The implementation of the proposed architecture was designed and the architecture for various components determined. In addition, development and simulation tools were specified. The integration of the proposed reconfigurable platform using the standard development environment for Mica2 motes was also considered.

1.3 Thesis Organization

This thesis consists of nine chapters. The related work is presented in Chapter 2. Chapter 3 describes the basic concepts and provides an overview of Wireless Sensor Networks, (WSNs). Chapter 4 explains real time operating systems. Chapter 5 evaluates low power reconfigurable devices. Chapters 6 to 8 deal with the concept and
implementation of the proposed reconfigurable mote. Chapter 9 concludes with ideas for further expansion of this thesis.
CHAPTER 2

RELATED WORK

The concept of reconfigurability attracts a lot of attention and active research is taking place in the areas of cellular networks. In such networks the requirements of interoperability and standardization have forced researchers to specifically evaluate the reconfigurability in processors. In addition, the reconfigurability of radio devices such as radio stack using either software defined radio, (SDR), or software radio, (SW), techniques are of interest. The concept of "reconfigurable terminals" has been evaluated and the challenges associated with development have been identified in order to establish a roadmap for the implementation of reconfigurable cellular phones, [31]. In the case of wireless sensor networks research has mainly focused on developing the smallest, most affordable and power efficient motes. However, the hardware reconfigurability concept for a WSN has not been addressed. With respect to WSN networks the need of hardware reconfigurability is driving a new area of research within the WSN development groups.

2.1 Field Programable Wireless Sensor Network Nodes

The Nomad Mobile Research Centre, (NMRC), has designed Field Programable Modular Wireless Sensor Network nodes, [11]. In these motes, selected processing and
sensor signal filters are synthesized, as hardware, on an FPGA. Performance comparisons were made of this “novel mote” with the Mica2, from Crossbow, and the iMote, from Intel. The mote’s scalability and upgrading was well defined. They are based on reconfiguring the processing and sensor signal filtering according to the application and the environmental requirements. However, the radio communication of the remote was not considered since it is inflexible, which limits the radio stack and the protocol reconfigurability. In addition the remote maintenance support of the mote was not considered. Figure 2 presents the architecture of the NMRC solution and Figure 3 presents a picture the actual device.

Figure 2: Architecture of the NMRC Mote

Figure 3: Picture of an Actual NMRC Mote
2.2 PicoRadio and PicoBeacon

PicoRadio and PicoBeacon are two mote concepts presented by the University of California, Berkeley right after the development of the MICA2 motes. The PicoRadio is a meso-scale low cost radio designed for ubiquitous data acquisition, [14]. Its size is less than 1 cm$^3$ with a power limitation restricted to 100 µW for enabling energy scavenging. The architecture of the PicoRadio is depicted in Figure4. The PicoRadio integrates the concepts of reconfigurable state machines and FPGAs.

![Figure 4: Architecture of the PicoRadio](image.png)

The PicoBean platform is based on a scavenging radio that uses a patch antenna and a single solar cell. Several sources of energy were considered for this approach such as energy produced by vibration and energy produced by temperature gradients. Figure 5 presents a picture of the prototype PicoBean mote.
Since the design of these kinds of motes is oriented to indoor applications such as temperature monitoring in buildings, warehouse inventory, integrated patient monitoring in hospitals and home security among others; the employment of these kind of devices for outdoor applications such as habitat monitoring and disaster mitigation is range limited.

2.3 iMote 10

The iMote 10 is a new concept of mote for Wireless Sensor Networks developed by Intel, [9]. Development of this platform incorporated the concepts of ultra low power operation, system level integration and limited hardware reconfiguration. This mote was designed for monitoring applications such as agriculture, structural, earthquakes, industrial controls and military applications. The project involved the development of new ultra low power processors and low power reconfigurable devices. Figure 6 presents a picture of the prototype model.
This mote integrates the TinyOS operating system in its architecture and the hardware/software codesign was considered during the development of the mote. The iMote 10 uses Bluetooth in the physical layer and the radio and protocol stack were developed for Bluetooth. Figure 7 describes the TinyOS layer implemented for the iMote 10.

Since Direct Sequence Spread Spectrum has been shown to be the most power efficient physical layer the selection of Bluetooth is a drawback in the optimization of
power consumption. Since the Bluetooth standard is based on a Frequency Hopping scheme the performance with respect to low power consumption is poor.

2.4 Small Scale Adaptive Processing

The small scale adaptive processing approach is concerned with determining the most power efficient approach in reconfigurable processing for wireless sensor nodes, [15]. This approach establishes a trade off between fixed and reconfigurable logic. The power performance was measured and compared to different approaches for implementing the JPEG compression algorithm in a mote. The study concluded that excessive hardware flexibility is not power efficient. Therefore, the percentage of fixed and reconfigurable logic required to optimize power consumption must be determined.

2.5 Wireless Protocol Processor

Presently, researchers are focusing on establishing hardware-software codesign strategies for a processor that will perform better in wireless applications. For instance, a new microprocessor termed the Sensor Network Asynchronous Processor, (SNAP), has been developed, [16]. This microprocessor uses Hardware/Software codesign to optimize power consumption. The architecture for SNAP is depicted in Figure 8. In addition, an ultra low power processor, termed SNAP/LE, was introduced and yields better results, [17].
2.6 Software Reconfigurability

The main efforts associated with the reconfigurability issue is focused in software reprogrammability. These projects are developed primarily in TinyOS and their functionality supports Over the Air Programming, (OAP). In similar efforts soft processors have been implemented to enable reconfigurable hardware in wearable computing nodes for performing specific processes, [18]. In addition, a few approaches related to small scale reconfigurability have been evaluated in order to determine the most efficient trade off between power and flexibility.

2.6.1 Over the Air Programming, (OAP)

OAP is based on the concept of downloading a new program remotely in order to update an old node program. The advantage of OAP lies in the scalability and efficiency associated with programming a node remotely. The core of Over the Air Programming is based on the Deluge system, which is a resident core program of the Atmel microcontroller. The actual TinyOS application runs on top of Deluge. The
implementation of a new program is performed by dissemination through the gateway, which transmits the new program to the adjacent nodes, which transmit the new program to the other nodes.
CHAPTER 3

AN OVERVIEW OF WIRELESS SENSOR NETWORKS

3.1 Introduction

Research in Wireless Sensor Networks, (WSN), was started in the early 1980’s by the U.S defense establishment. The first "motes" were big and expensive boxes that required a truck to hold the gateway equipment. The first motes used large batteries and the sensors were not integrated into the unit.

Evolution of the technology of embedded systems and power sources enables modern motes to be much more compact. The small size of modern motes has facilitated their application to new applications and research in the development of embedded networked systems. The evolution of the WSN continues a rapid expansion and many accomplishments have been achieved. Table 1 summarizes the evolution of the WSN.
Table 1: Evolution of Sensor Nodes

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Generation 1</th>
<th>Generation 2</th>
<th>Generation 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size</strong></td>
<td>Large shoe box or bigger</td>
<td>Pack of cards</td>
<td>Dust particle</td>
</tr>
<tr>
<td><strong>Weight</strong></td>
<td>Kilograms</td>
<td>Grams</td>
<td>Negligible</td>
</tr>
<tr>
<td><strong>Node Architecture</strong></td>
<td>Separate sensing, processing and communication</td>
<td>Integrated sensing, processing and communication</td>
<td>Integrated sensing, processing and communication</td>
</tr>
<tr>
<td><strong>Topology</strong></td>
<td>Point to point, star</td>
<td>Client server, peer to peer</td>
<td>Peer to peer</td>
</tr>
<tr>
<td><strong>Power supply lifetime</strong></td>
<td>Large batteries, hours to days</td>
<td>AA batteries, days to weeks</td>
<td>Solar, months to years, Human Power, energy Scavenging</td>
</tr>
<tr>
<td><strong>Deployment</strong></td>
<td>Vehicle placed or air drop single sensors</td>
<td>Hand emplaced</td>
<td>Embedded, sprinkled left behind</td>
</tr>
</tbody>
</table>

3.2 Characteristics

The deployment of Wireless Sensor Networks has increased due to the rapid strides in technology development. New features have been incorporated in the WSN that make applications more robust and reliable. The most relevant characteristics of a WSN can be summarized as:

- Self-Organizing,
- Self-Healing,
- Scalable,
- Flexible,
- Robust,
- Reliable,
• Low cost,
• No Infrastructure,
• Frequency Reuse,
• Wide Coverage,
• Mesh networking.

3.3 Architecture of Wireless Sensor Networks

A generic wireless sensor network is composed of three parts. A front end sensor is required such as those required to sense temperature, water flow, GPS, wind speed and direction, accelerometers. An RF mote is required in order to establish the network to cover a specific area. A Gateway is required in order to collect the data acquired by each sensor. A distributed relational database engine such as Postgres or Oracle is implemented in the gateway and the monitoring center to store all the data received from each sensor. Triggers are implemented to generate alert levels and to handle the connectivity to a main backbone such as an Internet, Intranet or point-to-point connection. Figure 9 presents a typical architecture for wireless sensor networks.
3.3.1 Standard Mote Architecture

The standard platform of a wireless sensor network relies on the integration of processing, sensing and communications into a tiny device. These tiny devices are autonomous and use peer-to-peer communication to establish mesh networks that enable them to cover a broader area. Frequently, these nodes do not require an external infrastructure or power supply and can be deployed easily.

Any mote has to have a robust radio technology, ultra low power processing, flexible inputs and outputs for sensors, a life time energy source, interoperability and reconfigurability. In general, a standard mote is composed of several devices. A standard mote possesses a processing unit and memory such as SRAM, ROM or Flash memory. Additionally, the mote requires an RF radio, an antenna array and interfaces to the sensors. The mote interfaces consist of devices such as Universal Asynchronous Receive and Transmit, (UART), units, which are Serial Port Interface, (SPI), units and
Inter IC, (I2C), units. A/D conversion blocks are also required in the standard mote. Figure 10 depicts the architecture for a standard mote.

![Figure 10: Standard Mote Architecture](image)

**3.4 Physical Layer in Wireless Sensor Networks**

**3.4.1 Overview**

Several physical layers have been adopted for Wireless Sensor Networks. The most used is Direct Sequence Spread Spectrum, (DSSS). DSSS provides simplicity, low power consumption and coexistence capabilities. However, some platforms use Frequency Hopping but this approach is not power efficient due to synchronization requirements. Recently, Ultrawideband, (UWB), has appeared and seems to be a suitable technology for Wireless Sensor Networks due to its very low power consumption, penetration, simplicity and ranging characteristics. However, some challenges such as narrowband interference cancelation and limited range have to be resolved.
3.4.2 Overview of the IEEE 802.15.4, “Zigbee” Standard

Recently, the IEEE released a new standard, which is termed the IEEE 802.15.4, “Zigbee”, for the Wireless Sensor Networks Physical and Media Access Control, (MAC), Layers. The IEEE 802.15.4 standard defines the parameters required for the physical and MAC layers, which operate at 868 MHz in Europe, 915 MHz in the USA and the 2.4 GHz ISM band. Sixteen channels in the 2.4 GHz and ten channels in 915 MHz are used by this standard. Zigbee is based on DSSS Technology and uses BPSK with raised cosine pulse shaping for the lower bands and Offset-QPSK with half-sine pulse shaping, at 2.4 GHz, which is equivalent to a minimum shift keying, (MSK), scheme. Zigbee allows the use of simple, low-cost and relatively non-linear power amplifier designs. Media access employs the Carrier Sense Multiple Access Collision Avoidance, (CSMA-CA), technique. The MAC layer can support network association and disassociation and possesses an optional superframe structure with a beacon for time synchronization. Zigbee includes receiver energy detection, (ED), link quality indication, (LQI), and clear channel assessment, (CCA), and supports both contention-based and contention-free channel access methods. In the same way, warmup power loss reduction is provided.

The IEEE 802.15.4 standard uses data lengths of 64-bits and 16-bit short addresses, which can theoretically support more than 65,000 nodes per network. However, the IEEE 802.15.4 standard is controlled and defined by the Zigbee Alliance, which is concerned about network security and application layers.
3.5 Mesh and Multihop Routing

3.5.1 Mesh Routing

The main concern associated with sensor networks is the minimization of power consumption in order to maximize the lifetime of the system. Therefore, link budgets must be minimized in order to save power. However, this task is not straightforward because of unpredictable RF link condition behavior, which is affected considerably by the time of the day, multipath and external interferences. Mesh routing compensates for low power realities since dynamic links can be used when optimal routes fail by exploiting multipath effects. In addition, the low power restrictions are met by limiting the link budgets. Mesh routing improves coverage. Figures 11 and 12 present RF pattern coverage without and with mesh routing respectively.

![Coverage Map without Mesh Routing](image_url)

Figure 11: Coverage Map without Mesh Routing
3.5.2 Multihop Routing

The routing scheme used for the Mica2 motes is MintRoute. This protocol is based on optimizing the expected success rate, which is a function of link quality from the parent to base and the mote to parent. Each node monitors up to 16 neighbors and reports their receive quality. Each node broadcasts its cost, which is the addition of the parent cost and the link’s cost to parent. The target is to minimize the total cost. The data packets are acknowledged by parents and are retransmitted up to 5 times. MintRoute uses less power in order to guarantee a lifetime greater than 1 year. In this scheme the nodes are normally sleeping and wake up periodically to check the Received Signal Strength Indicator, (RSSI). Messages have a long preamble in order to get synchronized if the detected RSSI is good. Each mote checks and reports the battery voltage and consumption.
CHAPTER 4

REAL TIME OPERATING SYSTEMS FOR A WSN

4.1 Introduction

There are many real time operating systems, (RTOS), depending upon application characteristics such as PALOS, µCOS-II, eCos, VxWorks and Real-time Linuxes [1], [3]. However, the sensor network systems are unique and several specialized constraints have to be considered. The most important requirements for WSNS are:

- **Small physical size and low power consumption:** The physical size and power requirements constrain the processing, storage and interconnection capabilities of a basic device. Therefore, the software must be efficient in the utilization of memory and processing.

- **Concurrency-intensive operation:** The focus of the system is the transfer of information from node to node while minimizing the processing in real time. The system has to perform several real-time events. For instance, information can be simultaneously acquired, processed and streamed onto a network. In the same way the node can receive data from other nodes.

- **Limited physical parallelism and control hierarchy:** In a sensor network the node’s processor has to perform almost every operation. Typically, the
microprocessor has primitive interfaces to the sensor and actuators.

- Diversity in design and usage requires modularity: Sensor network devices are designed for specific applications rather than for general purposes. A device possesses only the limited hardware required for its specific application. The hardware requirements vary as a function of the application. Therefore, the system requires open and efficient software with a high degree of modularity.

- Robust Operation: The main feature of sensor networks is the large number of unattended nodes. This feature mandates that the operating system be reliable, robust and capable of providing support for distributed applications.

4.2 TinyOS

4.2.1 TinyOS Concept

TinyOS is an event-based operating system primarily intended for meeting the requirements of wireless sensor networks. It is based on a component programming model provided by the NesC language, which is a dialect of C. Formally, TinyOS is not an operating system. TinyOS is a programming framework for embedded systems with a series of components that enable the construction of an application-specific operating system.
4.2.2 TinyOS and NesC

The programming language nesC is an extension to C, which was designed to handle the structure concepts and execution model for TinyOS. The original TinyOS, which was written in C, was reimplemented in nesC, Version 1.x. TinyOS applications are written in the nesC programming language.

4.2.3 Concepts of NesC

The programming language nesC presents a programming environment based on event-driven execution, a flexible concurrent model and split-phase operations for component-oriented application design, which are:

- Based on a holistic or integral system design,
- Each mote runs a single application at a time,
- Resources are known statically,
- Hardware and software boundaries are application and platform dependent,
- Application specific-code is integrated in the TinyOS core,
- The compiler performs static component instantiation,
- Management of global variables to preserve memory,
- Uses pointers and does not copy buffers,
- Stores local variable in a stack,
- Mote applications are deeply tied to actual hardware.
4.2.4 Component-Based Architecture

The programming language nesC is based on the separation of construction and composition. Programs are built out of reusable components, which are linked, “wired”, to form the complete application. Figure 13 presents the TinyOS architecture.

![Figure 13: TinyOS Architecture](image)

4.2.5 Concurrency

There are two sources of concurrency in TinyOS, which are termed Tasks and Events. Tasks represent non-time critical operations. A task runs until completion and does not interrupt other tasks. Tasks can be posted by components to be executed at a time dictated by the TinyOS scheduler. Events are time critical operations that are activated by interruptions from environment variables or the completion of split-phase operations. Events, like tasks, run to completion. However, an event can interrupt the execution of a task or another event. Events are implemented by the users of an interface.
Other important elements of TinyOS are commands, signals and split-phase operations. Commands are typically used to perform some service such as starting a sensor reading. Commands are implemented by the providers of an interface. Signals are used to control events. Signals call commands, post tasks, pass parameters and interrupt asynchronous tasks. Split-phase operations are used in order to preclude blocking operations and allow tasks to execute in a non-preemptive way. All long-latency operations are split-phase, which means that request and completion functions are separated. The best example of a split-phase operation is the packet sending process. During a packet sending process the component invokes the send command to start the transmission of the packet. The communication component signals the send Done event when the transmission is complete. Figure 14 presents the TinyOS model for these interactions.

Figure 14: TinyOS Model
4.2.6 Design with NesC

Design with nesC is based on a concurrency model provided by events and tasks where the applications are built using reusable components with well-defined bidirectional interfaces. The nesC language does not use dynamic memory allocation and the application diagram is fully known at compilation time. These mechanisms increase simplicity and accuracy.

4.2.7 Interfaces

A component provides and uses interfaces, which are the points of communication with other components. Interfaces are bidirectional and utilize commands and events. The interfaces are closely related to the hardware and are defined in the .nc files, which are normally located in the /opt/tinyos-1.x/tos/interfaces.

4.2.8 Atomicity

In TinyOS, code executes either in an asynchronous way in response to an interrupt or within a synchronously scheduled task. In order to enable the detection of data races, which occur due to the concurrent updates to share state, nesC contains both synchronous and asynchronous code. Asynchronous code is code that is reachable by, at least, one interrupt handler. Synchronous code is code that is only reachable from tasks. The nesC language also has a special kind of code that is termed “Atomic”. Atomic code is a small code sequence that nesC ensures will run without preemption. However, atomic statements delay interrupt handlers, which makes the system less responsive. Therefore, the programmer must avoid call commands or event signaling.
4.2.9 TinyOS Data Link layer

The TinyOS networking stack uses Carrier Sense Multiple Access Collision Avoidance, (CSMA/CA), single error correction and double error detection data encoding. The packet abstraction in TinyOS is given by the Active Message, (AM), component. TinyOS provides a namespace of up to 256 AM message types, which can be linked to an independent software handler. The TinyOS packet format is depicted in Figure 15.

![TinyOS Packet Format](image)

**Figure 15: TinyOS Packet Format**

The phases for sending and receiving a packet in TinyOS can be summarized as follows:

1. The sender enters a CSMA delay period and waits for an idle channel,
2. When the channel is ready the sender sends a packet start symbol at 10 Kbps,
3. The receiver samples the packet start at 20 Kbps,
4. New Data is transmitted at 40 Kbps,
5. The receiver synchronizes to the new data rate,
6. The sender pauses a few bits after the start symbol and sends only one bit,
7. The receiver detects and synchronizes with these timing bits by adjusting its radio clock,
8. The sender begins transmission of the encoded packet,
9. The receiver decodes the packet in a packet buffer,
10. After transmitting all of the packet the sender transmits only ones in order to enable the signal strength measurement,
11. The sender and receiver shift positions,
12. The sender introduces a new pause as part of the shift phase,
13. The receiver sends a short acknowledge pattern bit. If the sender receives an acknowledge packet marking packet,
14. The receiver checks the packet CRC,
15. If the packet is for the receiver, the receiver accepts the packet.

4.3 TinyOS Simulator, (TOSSIM)

4.3.1 Introduction

Several simulation tools have been used in the behavior simulation of wireless sensor networks. The most accepted was called NS-2. However, after the development of TinyOS, a new concept of simulation, which is termed the TinyOS Simulator, (TOSSIM), was established. TOSSIM was designed with respect to hardware abstractions [19]. TOSSIM provides a scalable, precise and discrete-event simulation
environment for TinyOS Wireless Sensor Networks. TOSSIM captures the behavior and interactions of networks of thousands of nodes. TOSSIM is a structure on the top of TinyOS that eliminates the need of an extra simulation language.

4.3.1 TOSSIM Architecture

The TOSSIM Architecture is composed of five parts:

- Support for compiling TinyOS component graphs into the simulation environment,
- A discrete event queue,
- Reimplemented TinyOS hardware abstraction components,
- Mechanisms for extensible radio and ADC models,
- Communication services for interaction with other programs.

TOSSIM runs the same code as the sensor. It simply replaces the low level components linked to the hardware by their model components. TOSSIM translates hardware interrupts into discrete events. Figure 16 presents the TOSSIM architecture.

Compiler support is provided to match the simulation and deployment results. The nesC compiler was modified to support TinyOS component compilation for the simulator framework. To facilitate hardware emulation TOSSIM replaces the hardware-based component by a simulated version. TOSSIM emulates the behavior of the Analog-to-Digital converter, the clock, the transmit power control, memory, components of the radio stack and even sensors and actuators. The CORE of TOSSIM relies on a simulator event queue for execution. During execution TOSSIM calls interrupt handlers that are hardware abstractions of components, which control TinyOS events by calling the tasks.
TinyOS commands that simulate real mote operation. TOSSIM simulates the behavior of each node independently. TOSSIM supports external radio models, which are given by a directed graph of bit error probability. Each edge, \( (u,v) \), in the graph represents the error rate when mote \( u \) sends to \( v \). Link probabilities can be defined and modified at runtime. It is important to note that TOSSIM does not have an actual RF model. TOSSIM simply uses the error probability to drop the bits. TOSSIM has two special case radio models, which are simple and static. One places all the motes in a single radio cell with an error free scenario. The other represents a static undirected graph of error free connections.

TOSSIM provides TCP/IP mechanisms to enable communication with other programs such as TOSSIM Visualization Tool, (TinyViz). TOSSIM signals events to applications in order to provide results at run time. Applications can send commands to TOSSIM to activate or modify a simulation. TinyViz is a Java graphical user interface
for TOSSIM that enables visualization and debugging of simulation results. Users can trace the execution of TinyOS applications, set breakpoints, visualize radio messages and manipulate the virtual position and radio connectivity of motes. These operations rely on TinyViz plugins, which are software modules that watch for events coming from the simulation such as debug messages and radio messages. TinyViz plugins can be enabled or disabled at execution time. TinyViz possesses plugins for debug messages, breakpoints, ADC readings, sent radio packets, radio links, location, radio model and power profiling.

### 4.3.3 TOSSIM Network Simulation

TinyOS uses three sampling rates. Data is sampled at 40 kbps. The sampling rate is 20 Kbps for the receiving start symbol. The sampling rate is 10 Kbps for sending a start symbol. TOSSIM adjusts the radio bit rates by modifying the period of the radio clock events. This ability facilitates TOSSIM in performing a perfect simulation of the TinyOS stack. In a similar manner, TOSSIM can simulate the hidden terminal problem and errors in the shift phase.

### 4.3.4 POWER TOSSIM

POWER TOSSIM was created because of the need of determining, in a precise way, the power consumption of each mote. POWER TOSSIM is designed on the top of TOSSIM and is based on introducing a new hardware characterization and a model equipped with a new component termed PowerState.nc.
4.3.4.1 Micro-Benchmarks

In order to simulate the power consumption of each component, which affects the energy of the mote, the concept of micro-benchmarks was introduced in the TinyOS environment. Micro-benchmarks related to CPU instructions, microcontroller power states, radio transmissions and EEPROM read and writing operations are incorporated in TinyOS.

4.3.4.2 Power TOSSIM Architecture

In order to determine the power consumption, POWER TOSSIM tracks the power state of each hardware unit by using a new component call, PowerState, which generates power transition messages. The hardware’s drivers used by TinyOS such as CCC1000RadioIntM are replaced with the simulated version that includes the power profiling.
CHAPTER 5
RECONFIGURABLE DEVICES FOR THE PROPOSED ARCHITECTURE

5.1 Introduction

Re-configurability in Wireless Sensor Networks is a new issue in the research environment. Most of the reconfigurable devices are not designed for this kind of application. Therefore, they do not meet the applications requirements such as ultra low power and portability. In this chapter the characteristics of the most suitable reconfigurable device for each of the components of a WSN node are explored and presented. In addition, the features of the most suitable commercial products are compared.

In order to incorporate a reconfigurable device such as a Field Programmable Gate Array, (FPGA), or Complex Programmable Logic Device, (CPLD), in a wireless sensor networks mote it is necessary to determine the design parameters and characteristics required for a reconfigurable system. Thereafter, proper device selection can be performed and economical and inconvenient influences avoided.

5.2 Types of Programmable Logic

According to the application, it is possible to find a great variety of programmable devices. The spectrum ranges from small devices for implementing basic logic
operations to huge FPGAs that can hold a microprocessor core including the peripherals.

The most used types of reconfigurable devices are the Simple Programmable Logic Devices, (SPLDs), CPLDs and FPGAs, [20], [22].

5.3 Simple Programmable Logic Devices, (SPLDs)

A Simple Programmable Logic Device, (SPLD), consists of arrays of unconnected “AND” and “OR” gates. A SPLD contains an array of fuses that enable the connection of the different inputs to the AND gates. Figure 17 presents the architecture of a PLD. Typically, up to 10 to 20 logic equations can be implemented in this type of device.

![Figure 17: PLD Architecture](image)

SPLDs are classified as Programable Read Only Memory, (PROM), Programable Logic Array, (PLA), and Programmable Array Logic/Generic Array Logic, (PAL)/(GAL). In the same way, there are PLD Macrocells.
5.3.1 Programmable Read Only Memory, (PROM)

PROMs offer high speed, low cost and are suitable for small designs. They are implemented using an orthogonal arrangement of wires, which can be enabled by an optional connection or row activation. Some of them are configured by breaking connections to activate them. Some ROMs can be erased and be reprogrammed, (EPROMs).

5.3.2 Programmable Logic Array, (PLAs)

PLAs are more flexible than PROMs and enable more complex designs. PLAs have both a reconfigurable AND-plane and OR-plane. Any two-level AND-OR circuit can be implemented.

5.3.3 Programmable Array Logic/Generic Array Logic, (PAL)/(GAL)

PALs and GALs offer good flexibility and are faster and less expensive than PLAs. However, only the AND-plane can be configured. The OR-plane is fixed.

5.3.4 Macrocell

Another type of PLD is the Macrocell which incorporates flip-flops, multiplexers, buffers and a clock input. Combinational or sequential logic can be implemented in a macrocell.
5.4 Complex Programmable Logic Devices, (CPLDs), for the Reconfigurable Platform

CPLDs were pioneered by Altera. Altera’s first family of CPLDs was called Classic EPLD’s. The first series was followed by three additional series that were classified as MAX 5000, MAX 7000 and MAX 9000. Because of the rapid growing market of the CPLDs in reconfigurable devices, others manufacturers such as Xilinx, AMD, and Lattice started to produce CPLDs. A CPLD can be defined as the integration of several PLD blocks with programmable interconnections on a single chip. The inputs and outputs are connected by a global interconnection matrix that provides two levels of programmability. Each PLD block can be programmed and the interconnection of the PLDs can be programmed. Figure 18 presents the architecture for a generic CPLD.

![CPLD Architecture](image)

Figure 18: CPLD Architecture

Pin-to-Pin delays are very short, making CPLDs the most suitable device for wide decoding, state machines and counters. The design software is simple, easy to use and it compiles very fast. CPLD’s are limited in size and the number of flip-flops is reduced, [23]. However, traditional CPLDs have fairly high static power consumption, which prevents them from being used in the proposed reconfigurable platform due to the power
constraints inherent to Wireless Sensor Networks. In order to solve this challenge a new concept of Zero Power was introduced by companies such as Xilinx, Lattice and Atmel. Xilinx incorporated its technology termed Fast Zero Power, (FZP), in its family called CoolRunner™. Lattice and Atmel used the concept of Zero Power in their families called ispMACH and ATF respectively. These devices demonstrated power consumptions lower than 1 mW while in standby.

### 5.4.1 Xilinx’s CPLD, CoolRunner™

The CoolRunner™ family is based on Fast Zero Power technology, which is a patented Xilinx development for ultra low power consumption, [24].

#### 5.4.1.1 Fast Zero Power, (FZP), Technology

Traditional CPLD architectures use in the product-term array of the original bipolar PLD design technology is presented in Figure 19. In this approach, the product-term has connections for each input of the logic block, which produces a high capacitance. Since the switching is slow, linear sense amplifiers are used to improve the propagation delay. However, the use of these amplifiers incorporates continuous power consumption.
In order to solve this challenge CMOS technology was employed in order to take advantage of its low power consumption. In FZP, the AND gates in the product-term array are implemented using configurable multiplexers linked to the inputs of a normal CMOS NAND gate. Each multiplexer selects the input, its complement or Vcc. Figure20 presents the structure of this approach.

Implementation of wider AND gates is accomplished by the use of a DeMorgan tree. In this case the inverter of the Figure20 is replaced by a 2-input NOR gate. In order to extend the array a NAND gate and inverters are used to combine the outputs of the
NOR gate. Figure 21 presents the model of an 8-input AND function implemented using the FZP approach.

![8-input AND Function using FZP Technology](image)

**Figure 21:** 8-input AND Function using FZP Technology

### 5.4.1.2 CoolRunner™ Architecture

The CoolRunner™ architecture is based on macrocells, which are combined to form Function Blocks (FB), [24], [25]. These blocks are interconnected with a global routing matrix, which Xilinx terms an Advanced Interconnect Matrix (AIM). The Function Blocks use a Programmable Logic Array (PLA) configuration, which enables all product terms to be routed and changed among any of the macrocells of the FB. Figure 22 presents the CoolRunner™ II CLPD Architecture.
The function block of the CoolRunner™ II contains 16 macrocells with 40 inputs and 16 outputs, which enables 56 product terms. These macrocells support sum of products that comprises up to 40 inputs and span 56 product terms within a single function block. The logic functions can be purely combinational or registered, with the storage element operating either as a D flip-flop, a T flip-flop or a transparent latch. Each flip-flop is configurable for either single or dual edge clocking. Figure 23 depicts all the available resources in the Macrocell of the Xilinx CoolRunner™ II.
The Xilinx CoolRunner™ II CPLD provides solutions requiring from 32 to 512 macrocells. CoolRunner™ II CPLDs can operate from 1.8V to 3.3V.

5.4.2 Lattice’s CPLD, ispMACH

Lattice’s ispMACH CPLD is based on a non sense-amplifier design approach, which is very similar to the Xilinx Fast Zero Power technology. It is a total CMOS design. The ispMACH architecture is based on Generic Logic Blocks, (GLBs), which are interconnected by a Global Routing Pool, (GRP). Output Routing Pools, (ORPs), are used to connect the GLBs to the I/O blocks, which contain multiple I/O cells, [26]. Figure24 presents the architecture of the ispMACH 4000.
A GLB consists of a programmable AND array, logic allocator, 16 macrocells and GLB clock generator. The programmable AND array consists of 36 inputs and 83 output product terms that are made up of 80 logic and 3 control terms. Each of the 80 logic product terms are applied the logic allocator and the 3 control product terms control the Shared PT Clock, PT initialization and PT Output Enable. Every five product terms from the 80 logic terms form a Product Term Cluster for a macrocell.

Each macrocell of the Generic Logic Block includes a programmable XOR gate, a programmable resister latch and routing for the logic and control functions. The macrocells feed the GRP and OPR. A direct input exists from the I/O cell enable high speed input register with programmable delay. The clock input can be selected from four different blocks along with the PT clock. Figure 25 presents the Architecture of a macrocell in the ispMACH 4000 CPLD.
The Lattice ispMACH 4000 provides solutions requiring from 32 to 512 macrocells. It possesses multi-voltage operation. The Lattice ispMACH 4000 supports 3.3V in the 4000V family, 2.5 V in the 4000B family and 1.8 V in the 4000C/Z family.

5.4.3 Atmel’s CPLD, ATF15xx

The Atmel, ATF15xx CPLD family uses Zero Power technology and proven electrical erasable technology. The architecture of this CPLD is based on Logic Blocks. Each block consists of 16 macrocells, a switch matrix and regional fold backs. The switch matrix receives all the signals as inputs from the global bus and the fold back handles the buried feedback from the macrocells. The logic blocks are interconnected using a Global Bus.

The macrocell consists of five sections for product terms and product term multipliers, a flip-flop, OR/XOR/CASCADE logic, output select and output enable and
logic array inputs. Each macrocell has 5 product terms to receive signals from the global and regional busses. The flip-flop can be configured for D, T, JK and SR operation or used as a latch. The macrocell can be programmed for combinational or registered logic.

The Atmel ATF15xx Family provides solutions requiring from 32 to 128 macrocells. This CPLD’s family supports only one operational voltage. However, it is designed to operate at either 3.3V or 5V.

5.4.4 Comparison of the CPLDs for the Reconfigurable Mote

Table 2 presents a comparison of the power consumption characteristics of CPLDs that were considered for the reconfigurable mote.

Table 2: CPLD Power Consumption Comparison

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Family Name</th>
<th>Vcc(V)</th>
<th>Standby Power(mW)</th>
<th>Power-Up Current (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lattice</td>
<td>ispMACH 4128ze</td>
<td>1.8</td>
<td>0.063</td>
<td>N/A</td>
</tr>
<tr>
<td>Lattice</td>
<td>ispMACH</td>
<td>3.3</td>
<td>0.12</td>
<td>N/A</td>
</tr>
<tr>
<td>Xilinx</td>
<td>CoolRunner II</td>
<td>1.8</td>
<td>0.36</td>
<td>N/A</td>
</tr>
<tr>
<td>Xilinx</td>
<td>CoolRunner II</td>
<td>3.3</td>
<td>0.66</td>
<td>N/A</td>
</tr>
<tr>
<td>Atmel</td>
<td>ATF15xx</td>
<td>3.3</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>Altera</td>
<td>MAX3000</td>
<td>3.3</td>
<td>507</td>
<td>N/A</td>
</tr>
</tbody>
</table>

5.5 Field Programmable Gate Arrays, (FPGAs)

Due to the limitations of CPLDs with respect to implementing pipelined and very complex applications new programmable devices were introduced. The Field Programmable Gate Arrays, (FPGAs), appeared to overcome the CPLD’s limitations. A FPGA consists of an array of logic blocks, I/O block and an interconnection matrix. All the lines of the I/O blocks can be configured either as input, output or bidirectional lines. Figure 26 presents the typical structure of a FPGA.
The logic block in a FPGA is based on a Look-Up Table, (LUT), combined with a flip-flop, a multiplexer, and some logic. The combinational logic is implemented in the LUT rather than using an array of gates. The truth table is stored in memory cells. The outputs of each cell are combined and driven by multiplexers, (pass transistors), to generate the output of the block.

5.5.1 Categories of FPGAs

Depending on the storage element used in the FPGA, several categories can be identified. The most used storage devices are Static Random Access Memory, (SRAM), Erase Programmable ROM, (EPROM), and Antifuse.
5.5.1.1 Static Random Access Memory, (SRAM), FPGA Based

In this approach, each switch is a pass transistor controlled by the state of an SRAM bit. In this case, the FPGA needs to be configured when the power is turned on.

5.5.1.2 Antifuse Based FPGA

The Antifuse based FPGA is built with connections that form a low resistance path when they are electrically programmed. Once these connections are programmed they can not be modified. The Antifuse FPGA is one time programmable device. This approach has the advantage of being radiation tolerant.

5.5.1.3 Electrically Erasable PROM, (EEPROM), FPGA Based

Commonly, this category of FPGA is based in a flash Electrically Erasable PROM, (EEPROM), memory. In this architecture, each switch is a floating gate transistor that can be turned off by injecting a charge onto its gate. This category exists in the middle of the SRAM and Antifuse techniques since it provides the non-volatility of antifuse with the reprogrammability of the SRAM. Consequently, this category is the most suitable for the reconfigurable platform proposed in this research.

5.5.2 Design of a Low Power FPGA

Power consumption can be minimized at the different levels of abstraction of the Physical layer such as the Gates, Combinational logic , Register transfer level and
Architecture level, [27]. The power consumption in a FPGA is concentrated mainly in the interconnection and clock sections. Figure 27 illustrates that in a typical FPGA more than 65 % of the power is in the interconnection wires and the remaining is associated with the clock, logic, output and inputs signals.

Figure 27: Typical Energy Breakdown of a FPGA Interconnect

The interconnection design plays a very important role in the power optimization and is the main purpose for using shorter wires. The connectivity between configurable logic blocks in a FPGA can be obtained through three levels of interconnect architecture. Level-0 is Nearest Neighbor, Level-1 provides a Mesh Architecture and Level-2 is a Hierarchical Interconnect. The approach most employed is a combination of the three, [28]. The routing starts at level-0 by interconnecting the nearest neighbors. The cost of these connections increases proportionally to the number of neighbors that must be interconnected. Figure 28 depicts the connection strategy for the nearest neighbor approach.
Those connections that cannot be completed using the Nearest Neighbor Connection strategy employ a Mesh or Hierarchical Interconnect. The mesh strategy is based on a symmetric approach. Figure 29 depicts the symmetric connection approach.

The selection of either a mesh or hierarchical interconnection is driven by the Manhattan Distance. If this distance is greater than 10, a binary tree with inverse clustering is used. Otherwise, the mesh approach is employed. Figure 30 depicts the connections for an inverse clustering binary tree.
One way to optimize clock consumption is through the use of dual edge triggered flip flops for the clock. Such devices make it possible to use half of the clock frequency, which considerably reduces power consumption. One of the last approaches proposed for powering the FPGA is replacing the strategy of using a predefined dual Voltage, Vdd, for the fabric by a configurable one, [29]. In such an approach each of the blocks of the FPGA can have Vdd Low or Vdd High according to the application requirements. In this way power consumption is reduced up to 50%, [30].

5.6 Field Programmable Gate Arrays, (FPGAs), for the Reconfigurable Platform

The main characteristics to be taken into account in the selection of the FPGA to be integrated are ultra low power consumption and low cost. The candidates for the FPGA, to be utilized in the reconfigurable mote, must comply and follow the characteristics presented for a low power FPGA. In order to have a wider spectrum of possibilities, the FPGAs produced by Actel, Aldec, Cypress, DynaChip, Lattice, i-cube, Phillips, Siliconexion, Synopsys, Xanthom, Atmel, Altera, Xilinx, Quick Logic, Spec, and Vitesse were evaluated. However, due to their characteristics, capacity and power consumption features only the architectures of the FPGAs from Altera, Xilinx, Quick Logic, and Actel were studied for this research.
5.6.1 Quick Logic Eclipse II FPGA

This FPGA’s family was designed by Quick Logic to meet the power requirements of portables devices such as handhelds, phones and PDAs. The family utilizes 0.18 µm six layer metal CMOS technology. In order to reduce power consumption the core voltage is 1.8 V and the I/O block is powered at 3.3 V. The FPGA consist of a main fabric that holds the array structure, which is surrounded by Embedded RAM blocks, Phase Locked Loops, (PLL), for clock synthesis, configurable arithmetic modules and Multi-standard I/O blocks. The capacity of this family ranges from 47,000 to 320,000 gates. The architecture of this FPGA is depicted in the Figure 31.

Figure 31: Architecture of the Quick Logic Eclipse II FPGA
5.6.1.1 Logic Cell

The basic logic cell consists of two six-inputs AND gates, four two-inputs AND gates, seven two-to-one multiplexers and two D flip-flops. The cell has 17 user inputs plus the control signals for a total of 30 lines. Additionally, it has six outputs. Four outputs are combinatorial and two are registered.

5.6.1.2 RAM Modules

The Quick Logic Eclipse II family includes up to 24 modules with dual port RAM of 2,304 bits for implementing RAM, ROM and FIFO functions. Each RAM module is user configurable and can be concatenated in horizontal and vertical directions to reach wider words or longer memory blocks.

5.6.1.3 Embedded Computational Unit

One of the most interesting features of this FPGA is the incorporation of an arithmetic unit to avoid inefficient implementations of these functions in the standard programmable logic. This unit includes a 16-bit adder, an 8-bit multiplier and a 17-bit register. Accumulators, multiplexers and decoders only come with the large FPGAs of this family.
5.6.1.4 I/O Cell Structure

This block supports voltages of 1.8, 2.5 and 3.3 V. It supports I/O standards such as LVTTL, LVCMOS25, LVCMOS18, PCI, GTL+, SSTL3, and SSTL2. In addition, signals with similar functions are grouped in the same I/O Block. For instance, control signals in one block and memory signals in another.

5.6.1.5 Low Power Consumption

As presented in the previous section the interconnection network in a FPGA plays a very important role in the power consumption of the device. The clock distribution is also critical in preserving the profile. The Eclipse II family uses an H-Shaped clock network architecture to provide flexibility and power conservation. In addition to the single global clock that is routed trough the device, the clock networks are routed in such a way that there are five independently accessible networks called quad-nets. There are five quad-nets in each quadrant and each quad-net can be turned off in order to reduce power consumption. Figure 32 depicts the quad-net structure.
5.6.2 Altera Cyclone FPGA

The cyclone FPGA is the lowest power consumption family of Altera FPGA’s. It was designed mainly for data-path applications. Its architecture is based on a two dimensional row and column based approach. These rows and columns interconnect the Logic Array Blocks, (LAB), to the embedded memory blocks. The I/O elements, which go to Cyclone device I/O pin, are located at the end of each LAB row and column.

5.6.2.1 Logic Elements, (LE)

A Logic Element, (LE), is the smallest unit of logic in the Cyclone Architecture. Each LE contains a four-input LUT, which is a function generator that can implement any function of four variables. Additionally, each LE includes a programmable register and a carry chain with carry select capability. Each LE drives all types of interconnects
such as local, row, LUT chain, register chain and direct link interconnects. The complete architecture of the Cyclone LE is depicted in Figure 33.

![Figure 33: Logic Element Structure of the Altera Cyclone FPGA](image)

Each register can be configured for D, T, JK, or SR operation or can be programmed as a latch. Each LE has three outputs that drive the local row and column routing resources. In addition to these three general outputs, the LE, within a LAB, has a LUT chain and register chain outputs, which enable the cascade of several LUTs in the same LAB.

The Cyclone LE can operate in a Normal mode or a Dynamic Arithmetic mode. The normal mode is recommended for general logic and combinatorial functions. In this mode four data inputs from the LAB local interconnects are inputs to a four-input LUT, which enables a higher number of LUTs to be cascaded for combinatorial purposes. The dynamic Arithmetic mode is suitable for implementing adders, counters, accumulators,
wide parity functions and comparators. In this mode, four two-input LUTs are used as a
dynamic adder/subtractor. The first two LUTs compute two summations based on a
possible carry in of 1 or 0 and the other two LUTs generate carry outputs for the two
chains of the carry select circuitry.

5.6.2.2 Logic Array Blocks, (LABs)

A Logic Array Block, (LAB), is based on the LE. It consists of 10 LEs, LE carry
chains, LAB control signals, a local interconnect, a look-up table chain and register chain
connection lines. The LUT chain connections transfer the output of one LE to the
adjacent LE for enabling a fast cascade within the same LAB. Figure 34 depicts the
interconnection scheme and the structure of the LAB.

![Figure 34: Logic Array Block Structure of the Altera Cyclone FPGA](image)
5.6.2.3 Embedded Memory

The Cyclone FPGA includes columns of 4 Kbit memory blocks. Each memory can be configured as a different type of memory with or without parity, true and simple dual-port, single-port RAM, ROM and FIFO Buffer.

5.6.3 Xilinx Spartan-3L Low Power FPGA

The Xilinx Spartan 3 family is a new FPGA family that replaces the Spartan II series and produced the lowest cost for the Xilinx FPGAs. The Spartan-3L is a member of this family, which is a lower power version of the Spartan 3. This FPGA consumes less static current and introduces the hibernate mode in order to optimize power consumption. The Spartan 3L architecture consists of five fundamental elements. The Architecture incorporates Configurable Logic Blocks, (CLBs), Input/Output blocks, RAM, Multiplier blocks and Digital Clock Manager, (DCM), blocks.

5.6.3.1 Configurable Logic Blocks, (CLBs)

This block consists of RAM-Based Look-Up tables, (LUTs), for implementing logic functions and storage elements than can be used as flip-flops or latches. CLBs enable the implementation of synchronous and combinatorial circuits. Each CLB consists of four interconnected slices, which are grouped in pairs. Each pair is organized as a column with an independent carry chain. Figure 35 presents architecture for the CLB.
Every slice is comprised of two logic function generators, two storage elements, wide function multiplexers, carry logic, and arithmetic gates. Left-hand and right-hand slice pairs use these elements to provide logic, RAM and arithmetic functions. However, the left-hand pair supports storing data using distributed RAM and shifting data with 16-bits registers.

5.6.3.2 Input/Output Blocks

Input/Output blocks control the flow of data between internal logic blocks and the device pins. These blocks enable the configuration of each pin to several standards such as GTL, HSTL, LVCMOS, LVTTL, PCI, SSTL, LDT, LVDS, LVPECL and RSDS. Each I/O line support bidirectional data and tristate operation.
5.6.3.3 Block RAM

All the devices of the Spartan 3 family incorporate 18Kbit RAM Blocks. The memory is configurable to create wider words or longer memory. Dual and single port memory configurations are supported.

5.6.3.4 Multiplier Blocks

The Spartan 3 family includes an embedded multiplier in the FPGA that accepts two 18-bits words as inputs to produce 36-bits results. The input buses to the multiplier accept data in two’s complement form. Either 18-bit signed or 17-bit unsigned numbers are accommodated.

5.6.3.5 Digital Clock Manager, (DCM), Blocks

One of the richest features of the Spartan 3 family is the incorporation of a Digital Clock Manager, (DCM), for enabling self-calibrating and providing fully digital solutions for distributing, delaying, multiplying, dividing and phase-shifting clock signals. The Spartan 3L family includes four DCMs, which are located at the ends of the outermost block RAM column. The DCM has four functional components. The DCM contains a Delay-Locked Loop, a Digital Frequency Synthesizer, (DFS), Phase Shifter, (PS), and the Status Logic where each component has its associated signals as depicted in Figure 36.
5.6.3.6 Low Power Operation

The low power optimization in the Spartan 3 FPGA is reached by incorporating a Hibernate Mode, which is activated by pulling the input PROG_B pin low. Activation of the Hibernate Mode is presented in the Figure 37.

Figure 37: Hibernate Mode Activation in the Xilinx Spartan 3L FPGA
In this mode, the power consumption is reduced to the minimum possible. The FPGA is put into a Hibernate Mode by switching off the VCCint, (core), and VccAux, (auxiliary), power supplies.

5.6.4 Actel ProASIC Plus FPGA

The Actel ProASIC Plus FPGA is a Flash-Based FPGA Family introduced by Actel that is used in low power applications. As a matter of fact, this FPGA has been selected by the Monterey Bay Aquarium Research Institute, (MBARI), for Low Power and High-Reliability Operation. The Architecture of the Actel ProASIC Plus is based on tiles. An array of tiles is surrounded by embedded memory and Input/Outputs blocks.

5.6.4.1 Logic Tile

The basic element of the Actel ProASIC Plus FPGA is the logic tile, which has three-input and one output cell. The inputs can be inverted individually and the output can be connected locally for an efficient long-line routing path. The title can be configured as a logic device, as a latch with clear or set or as a flip-flop with clear or set. Figure 38 depicts the tile structure.
5.6.4.2 Routing Resources

The core of the Actel ProASIC Plus FPGA family relies on very well structured routing. Four hierarchical levels are incorporated. The hierarchical levels consist of ultra-fast local resources, efficient long-line resources, very long-line resources and performance global networks. The ultra-fast resources are used to connect a tile directly to each of its neighbors. The efficient long-line resources provide connectivity between titles with a variable distance from 1, 2 or 4 tiles. These resources are distributed throughout the entire FPGA in horizontal and vertical ways. The routing software controls the loading effects due to distance and fanout. For very long connections, the routing is performed by the high-speed very long-line resources, which enables coverage of the entire device with minimum delay and power consumption. Figure 39 presents the distribution of the efficient long-line resources.
In order to interconnect the external pins of the devices to the internal logic signals high performance global networks are used. These signals can be used by internal blocks and present a very low skew. These lines are used for routing clock, reset and control signals ensuring minimum skew.

5.6.4.3 Flash Switch

The ProASIC Plus FPGA uses a programming element with a live-on-power-up ISP Flash switch. It has two transistors that share the floating gate, which is the storage element. One is a sensing transistor for writing and verification and the other is a switching transistor used for erasing the gate information.
5.6.4.4 Input/Output Blocks

These devices signals can be fully configured as inputs, outputs, tristate drivers or bidirectional buffers. In addition, they can operate at 2.5 V and 3.5 V.

5.6.5 Comparison of FPGAs for the Reconfigurable Mote

Table 3 presents a power consumption comparison of the FPGA families presented and evaluated in this chapter.

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Family Name</th>
<th>$V_{cc}$(V)</th>
<th>Standby Power(mW)</th>
<th>Power-Up Current (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick Logic</td>
<td>Eclipse II QL8050</td>
<td>1.8</td>
<td>0.17</td>
<td>N/A</td>
</tr>
<tr>
<td>Xilinx</td>
<td>Spartan 3L</td>
<td>3.3</td>
<td>6</td>
<td>N/A</td>
</tr>
<tr>
<td>Actel</td>
<td>ProASIC Plus</td>
<td>2.5</td>
<td>37.5</td>
<td>N/A</td>
</tr>
<tr>
<td>Altera</td>
<td>Cyclone</td>
<td>1.5</td>
<td>48</td>
<td>N/A</td>
</tr>
</tbody>
</table>

5.7 Most Suitable Programmable Devices for the Reconfigurable Platform

The selection of a FPGA or CPLD, for each of the reconfigurable components of the mote, depends mainly on the function of the mote and the power consumption. Therefore, a selection process that considers just FPGAs or CPLDs cannot be used in this concept. Consequently, it is necessary to evaluate and determine what is more suitable for each case.

CPLD’s are selected instead of FPGAs when high-performance logic is required since it possesses a less flexible internal architecture, the delay through a CPLD is more predictable and the delay is usually shorter. On the other hand, FPGA’s are used for
register-intense and pipelined applications such as complex designs with a core processor or system-on-a-chip implementations.

In order to determine whether a CPLD or FPGA is the most suitable for the proposed reconfigurable mote it is necessary to evaluate other features such as required capacity, operating voltage, analysis tools and price. Such analysis is required since power consumption is related to the size of the device and hence it is critical to determine the correct capacity for each device.

### 5.7.1 Capacity

The programmable devices were used to implement several functions in the reconfigurable mote. Each of these functions required different resources and hence it was mandatory to determine the specific requirements for each case. Table 4 summarizes the required capacity for each block in the proposed architecture.

<table>
<thead>
<tr>
<th>Reconfigurable Block</th>
<th>Function</th>
<th>Required Capacity</th>
<th>Suitable Size of the Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Processor</td>
<td>Soft Processing</td>
<td>18,347 gates (Xilinx)</td>
<td>30,000</td>
</tr>
<tr>
<td>Sensor Interface</td>
<td>Control of A/D Signals Modem</td>
<td>1,884 L.E. (Altera) 64 Macrocells (Xilinx)</td>
<td>3,000 128</td>
</tr>
<tr>
<td>Baseband Processor</td>
<td></td>
<td>180 Macrocells (Xilinx)</td>
<td>256</td>
</tr>
</tbody>
</table>
CHAPTER 6

PROPOSED RECONFIGURABLE PLATFORM

6.1 Overview

The challenges and limitations of the current wireless sensor networks platforms were presented in the previous chapters. This chapter delves into formulating the requirements and characteristics of the proposed platform that tries to alleviate the problems posed by current technologies. Once the features of the proposed mote are determined, the most suitable architecture will be discussed.

6.2 Requirements for Hostile Outdoor Environment Applications

The quasi-dream of having very small "motes" such as Picoradio and Smart Dust, which were introduced by University of California, Berkeley, is associated with developing indoor applications and trying to solve the low power, low cost and miniaturization challenges. However, current applications such as outdoor environmental sensing, natural disaster mitigation and agriculture automation require a more flexible and robust platform to enable operations in hostile environments. Most of these applications require that the equipment be deployed in very hard to access places. Thus the WSN motes must be as robust, autonomous and versatile as possible. Many of the deployment locations could be mostly in the tropical zones since the natural habitats are mostly centralized in those areas. The motes have to perform in areas with lots of
vegetation and humidity, which leads to the complicated task of maintenance and upgrading of these motes. Therefore, the proposed platform must possess capabilities of re-configurability, healing, upgradability and self-maintenance. In order to meet these requirements, a reconfigurable wireless sensor mote should possess such features as:

- The capability of working at the different available frequencies of 433MHz, 866 MHz, and 915 MHz in the ISM bands that are assigned for wireless sensor networks: This capability will allow the required interoperability and avoid the requirement of mote replacement due to cases of interference, frequency unification or range adaptation,
- Upgrading and updating capabilities of the radio and protocol stack, baseband processing and sensor signal adaptation. These features are vital to adapt to the standard development and application requirements. They also enable motes to be upgraded with the most efficient base processing and filtering schemes,
- Reconfigurability and upgrading of the main processing unit such as the microcontroller and microprocessor,
- Capability of remote maintenance operations in order to avoid the on-site manipulation of the node,
- Ultra low power and low cost,
- Maximum power autonomy,
- Scalable according to the requirements of the application to enable broad band commercial adaptation,
- Complying with the most relevant standards used in wireless sensor networks such as Zigbee and Spread spectrum techniques,
- Reliability and security,
- Programming over the air,
- Software upgrading, updating and replacement,
- Feasibility.

### 6.3 Proposed Mote Architecture Concept

In order to meet the requirements presented in the previous section a novel concept of a re-configurable mote architecture was introduced. The development of this concept required the review of various components of a mote in order to dentify the various levels of re-configurability required to achieve the desired features. A generic mote is composed of an antenna, an RF transceiver, processing units, sensor signal interfaces and a power unit.

#### 6.3.1 Antennas

The environmental monitoring applications are often required to cover huge areas, which limits the operating frequency to the lower frequencies of the Industrial, Scientific and Medical (ISM), Bands. Since the range for 2.4 GHz transmitter is about 50 meters it does not meet the requirements for these types of outdoor applications. Hence, the reconfigurable mote must be constrained to work in the bands of 433 MHz and 900 MHz. Consequently, the features of the required antenna are:

- Dual Band antenna that operates at 433 MHz and 900 MHz,
- Unit Gain,
• Quarter wavelength antenna,
• An external antenna since the patch antennas employed in cellular systems decrease the range, [31], [32].

Figure 40 presents a picture of an antenna that meets these characteristics.

![Dual Band Antenna for the Reconfigurable Mote](image)

**Figure 40: Dual Band Antenna for the Reconfigurable Mote**

### 6.3.2. RF Transceiver

The Direct Sequence Spread Spectrum technology is the most widely used in different wireless sensor nodes due to its low power features and simple implementation scheme. As a matter of fact, the recent Standard IEEE 802.15.4/Zigbee uses DSSS in the physical layer. Consequently, this is the most suitable base technology for the proposed solution. For instance, an IEEE 802.15.4 compliant radio transceiver has already been introduced by Chipcon, [33]. However, this radio has not considered the hardware reconfigurability in transmission frequency band, baseband processing and protocol stack.
6.3.2.1 Radio Architecture

A Re-configurable radio consists of a baseband processing block, ADC and DAC Converters, an RF quadrature modulator and an RF quadrature demodulator. Figure 41 describes the architecture of such a radio. A quadrature modulator, AD8345 from Analog Devices, is used to complement to the Quadrature Demodulator, AD8348 from Analog.

![Figure 41: Architecture of the Reconfigurable Radio](image)

6.3.2.2 Baseband Processing Block

The architecture for the baseband processing block relies on performing the mapping, encoding and I/Q signal generation in the baseband using an FPGA. As presented in Figure 42, the DSSS modulator is composed of a DQPSK Modulator, which receives the pseudorandom code from the code generator, a pulse shaping block followed by an I/Q signals generator that up-converts it either to zero IF or low IF.
Figure 42: Architecture of the Direct Sequence Spread Spectrum, (DDSS), Transmitter

The DSSS receiver accepts the quadrature I/Q signals coming from the Analog to Digital Converters and decodes them in the quadrature down-converter. The received signal is passed through a Chip Matched Filter in order to perform the despreading and demodulation operations that enable symbol detection. Figure 43 presents the receiver process.

Figure 43: Architecture of the Direct Sequence Spread Spectrum (DSSS) Receiver

6.3.2.3 I/Q Modulator

The in-phase and quadrature baseband signals are formed based on the transmitted data to be spread and modulated in the digital domain. Afterwards, digital to analog processing that is followed by a smoothing low pass filter for each of the I/Q
signals is performed. Finally, they are upconverted using quadrature mixers in order to have a single-side band RF signal. Figure 44 shows the architecture for the I/Q modulator.

![Figure 44: I/Q Modulator](image)

### 6.3.3 Processing Unit

In addition to the processing operation required for the modulation and demodulation processes, the mote requires a main processor unit to control the input/output interfaces, to drive the RF transceiver, to perform the memory storage procedures and compression and routing algorithms and other basic functions. On the top of this core runs the real time operating system, (RTOS), such as TinyOS.

In order to implement a complete reconfigurable processing system it is necessary to upgrade the soft core of a microcontroller or a microprocessor to an FPGA. Due to the overhead produced by this design and the current limitations of the FPGA’s related to designing the standby and sleep modes, it would produce an inefficient power system if a fully reconfigurable system was designed. In other words, a highly flexible system with the current technology is not highly power efficient, [15]. Therefore, a hybrid
reconfigurable architecture is required, which is composed of an ultra low power microprocessor or a microcontroller designed on a Field Programmable Gate Array, (FPGA), for performing specific algorithms according to the application requirements. The FPGA should only be powered on demand in order to optimize the power consumption of the mote.

Since the technology of the Atmel’s ultra low power microcontrollers have attained very good performance in power they were used as the cores for most of the wireless sensor nodes such as Mica2 from Crossbow and Fleck from Csiro. In addition, a new microprocessor concept and architecture have been developed and improved, [17].

6.3.4 Sensor Signal Adaptation

In the MICA2 mote, from the University of California, Berkeley, the signals from the sensors are processed by the Atmel microcontroller ATMEGA 128 L. This microcontroller incorporates an 8 channel Analog to Digital Converter to handle all the incoming signals from the sensor boards. The sensor boards use the standard 51 pins DF9 connector. The lines ADC0 to ADC7 correspond to the Analog to Digital Converter channels. Depending upon the application, different sensor boards can be utilized. The sensor boards provided by Crossbow are:

- MTS101: Includes a precision thermistor, light sensor and general prototyping area. Compatible with MICA and MICA2,
- MDA300: A data acquisition board that supports 8 analog inputs, 8 digital input/output channels and 2 relay channels,
• MTS400CA: This is an Environmental Sensor Board. It includes temperature, humidity, barometric pressure and an ambient light sensor as well as a 2-Axis Accelerometer,

• MTS420CA: Possesses the same sensors as the MTS400CA and incorporates a GPS module,

• MTS300: This is multi-sensor board. It includes light and temperature sensors, a microphone and a sounder,

• MTS310. Similar to the MTS300 but includes a 2-Axis Accelerometer and a 2-Axis Magnetometer.

There are several kinds of sensors with variable operating and sampling rates. Similarly, the input signal frequencies vary due to the different data rates from different sensors. These variable input signals force the microcontroller, ATMega 128L, to adapt to the different speeds, which results in an increased number of interrupts to the microprocessor. Since low power consumption for the microcontroller depends on a low duty cycle, where most of the time is spent in a sleep mode, the power efficiency decreases considerably and requires the batteries to be replaced more frequently.

Some sensors such as acoustic sensors and microphones generate data very often, which is written continually onto the Flash Memory of the mote. Since the major power consumption of the flash memory depends on the write operations the use of such sensors depletes the power resources.

In order to solve these problems a new sensor signal adaptation block was proposed. This block handles all the signals coming from the sensor boards and then stores the data in a power efficient SRAM and generates a minimum number of interrupts
to the microprocessor to avoid frequent wake ups. The architecture of the proposed block is depicted in Figure 45.

![Diagram of Sensor Signal Adaptation Block](image)

Figure 45: Sensor Signal Adaptation Block

The control logic for this block was based on a State Machine and logic block interfaces for the microprocessor and the Analog to Digital Converter. A CPLD was used to design the system, which included the SRAM.

### 6.4 Software Programmability and Reconfigurability

#### 6.4.1 Software Programmability

TinyOS has became the most widely used Real Time Operating System, (RTOS), in Wireless Sensor networks based on tiny devices such MICA2 from Crossbow. There are a series of programming, debugging and developing tools available that enable the designer to accomplish any project. Much of the software resources for wireless sensor network applications are developed in TinyOS. Therefore, the proposed platform had to adapt to this RTOS.

The standard programming tool employed in the MICA2 motes is the MIB510 board, which is a programming and serial interface board used to upload the required software to each mote. Figure 46 depicts the MICA2 board programming board.
In order to use the same programming tool the proposed mote used the standard 51 pins DF9 connector, which preserved the signal configuration. Similarly, equivalent hardware with pin-to-pin compatibility for the Atmega 128L was implemented as a soft processor on the FPGA. It is important to emphasize that the incorporation of a new processor implies further software development.

On another hand, enabling of the signal adaptation block was necessary to modify the drivers of the sensor boards. In order to ensure interoperability with the MICA2 motes it was necessary to perform adaptations to the TinyOS Code since the code in all the motes had to be the same.

### 6.4.2 Software Reconfigurability

Since TinyOS has been adopted as a Real Time Operating System for the proposed architecture it is imperative that the software re-configurability tools designed for the system are used. The scheme utilized by MICA2, from Crossbow, is Over the air Programming, (OAP). This approach is based on downloading a new core to be updated
onto the node. The core of the air programming relies on the Deluge system, which is a resident program running in the Atmel microcontroller. The actual TinyOS application runs on the top of deluge as depicted in the Figure 47. When a new core is completely downloaded, the core is stored in the Flash Memory to be used as a boot-up during the next power up.

Figure 47: Over the Air Programming Bootloader

6.4.2.1 OAP Strategy

The reprogramming is done in several phases. The new core is first downloaded in order to propagate it later to all the nodes. The propagation is performed by announcing, requesting and sending operations. Four different images are downloaded to avoid a corrupted core. After code verification the node is reprogrammed and rebooted.

In order to transmit all the data of the new core the program images are divided into contiguous pages with each page containing N packets. Figure 48 illustrates the OAP data configuration.
6.4.2.2 Operation Roles

The reconfigurability of a mote is based on an operational mode schemes. The conventional motes use the operation modes used by the microcontrollers, which consist of Standby, Sleep and Power-Up Mode. However, for the development of the concept of a reconfigurable platform it was necessary to introduce new operating roles such as:

- Cluster Head Operation,
- Repeater Operation,
- Sensing Operation,
- Slave Operation.

These new roles operate in combination with the microcontroller or FPGA power saving modes. The purpose of these roles is to enable the activation of the required reconfigurable blocks according to the application requirements and the wireless sensor configurations. They are designed to contribute and even to enhance the lifetime of the motes by changing the roles periodically according to the available energy, position in the network, function performed, activity and multihop operation.
CHAPTER 7

POWERING MODEL FOR THE PROPOSED PLATFORM

7.1 Introduction

The mote power source is the most critical component in the architecture of a Wireless Sensor Mote. The life time, free maintenance schemes and autonomy of the mote depends on the characteristics of power supply. Therefore, it is crucial to establish a model that ensures the feasibility of required features in the proposed concept. In order to develop the powering model it was important to determine how the Real Time Operating System, which was TinyOS, manages power consumption. In addition, power source types and their limitations had to be established. After the available and appropriate power source component set was established, a power consumption model for each element of the set was developed along with a power consumption optimization scheme.

7.2 TinyOS Scheduler

Power management in the Mica2 mote is performed by the TinyOS Scheduler, which is a component of the operating system that controls the execution of tasks driven by events. Figure 49 presents the TinyOS scheduler scheme. The TinyOS scheduler takes advantage of the power-save, power-down, and standby modes of the microcontroller, AtMega128L, to ensure a low power duty cycle. The microcontroller is
in sleep mode during the major portion of its life time and wakes up only when the TinyOS Scheduler decides it should be awakened. Generally, the microcontroller only wakes up to perform tasks such as sending or receiving a radio message, getting sensor readings or accessing the memory. However, the frequency of these tasks depends on the application and the type of sensing variables.

![TinyOS Scheduler Scheme](image)

**Figure 49:** TinyOS Scheduler Scheme

### 7.3 Duty Cycle

The duty cycle of the proposed system was controlled by the TinyOS scheduler. The minimum duty cycle depended on the number of sensors, the amount of data generated by each sensor, the rate of change of the measured variables and the inherent priorities of the specific applications. In summary, the limitation of using the lowest duty cycle depends on whether the application can handle the throughput provided by each duty cycle. Table 5, presents the maximum number of packets and the effective throughput for various duty cycles.
Table 5: Effect Throughput and Maximum Number of Packets vs Duty Cycle

<table>
<thead>
<tr>
<th>Duty Cycle</th>
<th>Maximum Number of Packets/sec</th>
<th>Effective Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 percentage</td>
<td>42.93</td>
<td>12.364 kbps</td>
</tr>
<tr>
<td>35.50 percentage</td>
<td>19.69</td>
<td>5.671 kbps</td>
</tr>
<tr>
<td>11.50 percentage</td>
<td>8.64</td>
<td>2.488 kbps</td>
</tr>
<tr>
<td>7.53 percentage</td>
<td>6.03</td>
<td>1.737 kbps</td>
</tr>
<tr>
<td>5.61 percentage</td>
<td>4.64</td>
<td>1.336 kbps</td>
</tr>
<tr>
<td>2.22 percentage</td>
<td>1.94</td>
<td>0.559 kbps</td>
</tr>
<tr>
<td>1.00 percentage</td>
<td>0.89</td>
<td>0.258 kbps</td>
</tr>
</tbody>
</table>

7.4 Power Model of the Mica2 Motes

In order to develop the powering strategies it was important to review the power model of the Mica2 motes designed by UC Berkeley. Table 6 presents the power consumption of each of the components of the Mica2. It is easy to identify that the main power consumption is due to Radio transmissions, write operations in the flash memory and CPU consumption in the active state. Figure 50 displays the power consumption profile when the transmission of a single message is performed at maximum transmission power. Normally, the system is in the power-save mode and consumes just 110 µA, which corresponds only to the microcontroller since the rest of the devices are turned off. In order to leave this state, the microcontroller goes to wake up mode and powers the ADC, radio and sensors. Immediately, since the radio stack uses Carrier Sense Multiple Access Collision Avoidance, (CSMA-CA), the radio starts to listen to the radio channel to detect any potential collision before beginning to transmit the message. Once a transmission link is established and the message is sent the system goes back to the power-save mode. Similarly, the mote wakes up periodically to sniff the Received Signal Strength Indicator, (RSSI), to detect any transmission from the other nodes.
Table 6: Power Consumption in the Mica2

<table>
<thead>
<tr>
<th>Component</th>
<th>Consumption</th>
<th>Component</th>
<th>Consumption</th>
<th>Component</th>
<th>Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Active</td>
<td>8.0 mA</td>
<td>CPU Idle</td>
<td>3.2 mA</td>
<td>Radio Rx</td>
<td>7.0 mA</td>
</tr>
<tr>
<td>ADC Noise Reduce</td>
<td>1.0 mA</td>
<td>Tx (-20 dBm)</td>
<td>3.7 mA</td>
<td>Tx (-19 dBm)</td>
<td>5.2 mA</td>
</tr>
<tr>
<td>Power-down</td>
<td>103 A</td>
<td>Tx (-15 dBm)</td>
<td>5.4 mA</td>
<td>Write</td>
<td>12.9 ms</td>
</tr>
<tr>
<td>Power-save</td>
<td>110 A</td>
<td>Tx (-8 dBm)</td>
<td>6.5 mA</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>Standby</td>
<td>216 A</td>
<td>Tx (-5 dBm)</td>
<td>7.1 mA</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>Extended Standby</td>
<td>223 A</td>
<td>Tx (0 dBm)</td>
<td>8.5 mA</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>Internal Oscillator</td>
<td>0.93 mA</td>
<td>Tx (+4 dBm)</td>
<td>11.6 mA</td>
<td>Sensor</td>
<td>0.7 mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tx (+6 dBm)</td>
<td>13.8 mA</td>
<td>board LEDs</td>
<td>2.2 mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tx (+8 dBm)</td>
<td>17.4 mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tx (+10 dBm)</td>
<td>21.5 mA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 50: Power Model for the MICA2
7.5 Power Supply for Motes

Both the Mica2 and the proposed mote were powered by a battery. The Mica2 uses 2 AA Alkaline batteries that provide a 3.0V power supply for the systems. However, the operation of the mica2 requires a voltage under 2.7V, [34]. I/O devices and sensors do not operate below 2.5 V. The radio, CC1000, and microcontroller have to operate above 2.1V and 2.2V respectively. Hence, the voltage range of just 0.3 V implies earlier battery replacement, which reduces the life time of the mote. Since the proposed platform focused on incorporating the maximum autonomy it was necessary to look for alternative batteries that could provide higher voltage ranges and also to design a smart power regulating system for the mote.

7.6 Rechargeable Batteries

In order to enable the maximum autonomy of the motes it is essential to avoid frequent battery replacements. Although the capacity of the rechargeable batteries is lower than single use batteries, the use of rechargeable batteries helps in prolonging the lifetime through timely recharging. The selection of the battery depends on its size, capacity, voltage and cost. Experiments performed with AA 1.2V Rechargeable Nickel Metal Hydride batteries revealed that the average capacity of these batteries is about 2500 mAh, [35]. Such a capacity yields 15.79 months of operation for the motes with a duty cycle of 1 percent and a power supply load of 10 mA, which is the average consumption for Mica2 motes. Although this lifetime was acceptable, battery replacement has to be done anyway. Therefore, it was necessary to look for strategies to expand the batteries
lifetime. Figure 51 presents the lifetime vs battery capacity for Mica2 motes working at 10 mA.

![Lifetime vs Battery Capacity](image)

Figure 51: Lifetime vs. Battery Capacity

### 7.7 Power Management Strategies for the Proposed Mote

The primary strategy to increase the lifetime of the battery of a mote is to reduce its duty cycle. One way to reduce the duty cycle of the microcontroller is by implementing an adaptation block for reducing interrupts. Another way of reducing the duty cycle is to modify the timer of the TinyOS Scheduler. However, this approach depends on the applications and monitored variables. Finally, a way of reducing the duty cycle is to reduce the operating time of the battery. If the battery works only 50 percent
of the time it is equivalent to reducing the duty cycle by 50 percent. Figure 52 illustrates the advantage of intermittent discharge compared to the continuous discharge.

![Graph showing continuous versus intermittent discharge](image)

**Figure 52: Continuous Compared with Intermittent Discharge**

### 7.7.1 Energy-Harvesting Sources

The concepts of energy scavenging and harvesting have gained significance in recent times due to the development of portable electronic devices such as handhelds, laptops, and palmtops, [36]. Due to the limitations in battery technology, extensive research is being carried out in order to find new energy sources and develop new transducers and devices. For instance, in remote sensing applications such as disaster mitigation and environment monitoring, the alternative energy scavenging has to be from the natural environment. The available sources for power could be sunlight, rain, heat, electromagnetic fields, animal sounds, and many other such sources. For example, during the rainy season in tropical forests, the energy can be derived from the falling water
drops. Similarly, power compensation from the sun and the wind represent attractive mechanisms. Table 7 presents the energy-harvesting opportunities. The most suitable power sources for motes are solar panels and vibrational microgenerators.
Table 7: Energy-Harvesting Opportunities

<table>
<thead>
<tr>
<th>Energy</th>
<th>Performance</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambient radio frequency</td>
<td>lower than 1 W/cm²</td>
<td>Unless near a transmitter</td>
</tr>
<tr>
<td>Ambient light</td>
<td>100 mW/cm² (directed toward bright sun) / 100 W/cm² (illuminated office)</td>
<td>Common polycrystalline solar cells are 16 percent 17 percent efficient, while standard monocystalline cells approach 20 percent.</td>
</tr>
<tr>
<td>Thermoelectric</td>
<td>60 W/cm²</td>
<td>Quoted for a Thermo Life generator</td>
</tr>
<tr>
<td>Vibrational Microgenerators</td>
<td>4 W/cm³ (human motionHz) / 800 uW/cm³ (machines - KHz)</td>
<td>Predictions for 1 cm³ generators. Highly dependent on excitation and structure</td>
</tr>
<tr>
<td>Acoustic Noise</td>
<td>3 E-6 mW/cm² at 75 dB sound level</td>
<td>Calculated from acoustic theory</td>
</tr>
<tr>
<td>Passive Human Powered</td>
<td>1.8 mW (shoe insert ≤ 1 cm²)</td>
<td></td>
</tr>
<tr>
<td>Ambient Air Flow</td>
<td>1 mW/cm³</td>
<td>Demonstrated in microelectromechanical turbine at 30 liters/min.</td>
</tr>
<tr>
<td>Push Buttons</td>
<td>50 uJ/N</td>
<td>Quoted at 3 V DC for the MIT Media Lab Device.</td>
</tr>
<tr>
<td>Hand generators</td>
<td>30 W/kg</td>
<td>Quoted for Nissho Engineeings Tug Power (vs. 1.3 W/kg for a shake-driven flashlight).</td>
</tr>
<tr>
<td>Heel strike</td>
<td>7 W potentially available</td>
<td>Demonstrated systems: 800 mW with dielectric elastomer heel - 26 250700 mW with hydraulic piezoelectric actuator shoes - 24 10 mW with piezoelectric insole.</td>
</tr>
<tr>
<td>Thermal Conversion</td>
<td>0.00018 mW - 10 deg C</td>
<td></td>
</tr>
<tr>
<td>Nuclear Reaction</td>
<td>80 mW/cm²</td>
<td></td>
</tr>
<tr>
<td>Fuel Cells</td>
<td>300 - 500 mW/cm³</td>
<td></td>
</tr>
</tbody>
</table>
7.8 Architecture of the Proposed Powering Module

The powering module for the proposed architecture was based on using a DC-DC converter for voltage stabilization. The DC-DC converter can be designed to insure that the system will be powered at optimal voltage with reduced power consumption, [37]. When the solar panel provides enough energy for the mote the battery will not be used, which extends battery lifetime. Special circuitry helps in preventing overcharging of the battery in order to increase the battery longevity. Figure 66 presents a diagram of the powering module.

![Diagram of the Powering Module for the Proposed Mote](image)

Figure 53: Architecture of the Powering Module for the Proposed Mote

### 7.8.1 DC-DC Converter

The DC-DC conversion, employed in the design of the power supply module, was performed by the MAXIM 1676. The features that made the MAXIM 1676 the most suitable DC converter were:

- Operation down to 0.7V Input Supply Voltage,
• Selectable Current Limit of 0.5A or 1.0A,
• 16 µA Quiescent Current,
• Synchronous Rectification for Improved Efficiency.

7.8.2 Solar Panel

The physical dimensions of the solar panel to be integrated into the mote must be as compact as possible. The price of the solar panel represents another important decision factor. PowerFilm™ Ultra Flexible Thin Film Solar panels, produced by Sundance Solar, were used for this project due to their compact size and simple integration with the mote. The panel produces approximately 3.6 V and 100 mA under ideal conditions. Figure 54 displays this flexible solar panel.

![Solar Panel Employed for the Proposed Mote](image)

Figure 54: Solar Panel Employed for the Proposed Mote
CHAPTER 8

IMPLEMENTATION AND TESTBED FOR THE PROPOSED PLATFORM

8.1 Introduction

In order to implement the architecture presented in chapter 7, a testbed was designed using the wireless sensor network implemented at USF for Disaster Management. This testbed was comprised of a Gateway that collects the data sent by the MICA2 and MICA2motes. The application testbed was used for integrating the new platform for a real-life application development environment. Figure 55 depicts the architecture of the proposed testbed integration.

Figure 55: Testbed for Integrating the Proposed Reconfigurable Platform
8.1.1 Test and Development Configurations

The testing and the development had to be performed in several stages. Different stages were required for sensor signal adaptation, soft processing and the reconfigurable radio. The different stages allowed a seamless integration of the different components.

The development of the mote was realized in the six configurations:

- Hard Processor, (Atmega128L), + CPLD, (A/D Conversion), + Memory + CC1000 for 433 MHz + Fixed antenna + Programming using the Standard tools, (MIB510-TinyOS),
- Soft Processor in an FPGA, (ATMega128L), + CPLD, (A/D Conversion), + Memory + CC1000 for 433 MHz + Fixed antenna + Programming using the Standard tools, (MIB510-TinyOS),
- Hard Processor, (Atmega128L), + Memory + CPLD (A/D Conversion) + optional specific processing + CC1000 for 433 MHz + Fixed antenna + Programming using the Standard tools, (MIB510-TinyOS),
- Hard Processor, (Atmega128L), + Memory + Spread Spectrum Reconfigurable Radio + Dual band Antenna + Programming using the Standard tools, (MIB510-TinyOS),
- Hard Processor, (Atmega128L), + Memory + Zigbee Reconfigurable Radio Dual Band antenna + Programming using the Standard tools, (MIB510-TinyOS),
8.1.1.1 Reconfigurable Sensing Configuration

The implementation of the reconfigurable sensing configuration was based on using the standard microcontroller employed in the ATmega 128L conventional motes and an external block implementation for sensor signal adaptation. The block was implemented using a CPLD, Analog to Digital Converter and SRAM Memory as presented in the Chapter 4. Both the Xilinx CoolRunner CPLD and the Lattice ispMACH CPLD were used in order to compare their performances and power consumption. The CPLDs from Atmel and Altera were not employed in this implementation because of their high power consumption. Figure 56 presents the testbed used for testing this configuration.

![Figure 56: Architecture of the Reconfigurable Sensing Configuration](image)

From the software perspective, the ADC modules and sensor signal drivers in TinyOS should adapt to this new hardware. The internal ADC converters of the ATmega 128L microcontroller could be deactivated even though the radio stack and the protocol modules remain unchanged. Implementation of this new interface reduced the power consumption of the mote considerably since the low duty cycle was maintained for the microprocessor. Figure 57 presents the power consumption when signal block adaptation
was not used. The improvement in power consumption when the signal adaptation block is used is presented in Figure 58.

Figure 57: Power Consumption of a Microcontroller without Signal Adaptation Block

Figure 58: Power Consumption of a Microcontroller Using a Signal Adaptation Block
8.1.1.2 Reconfigurable Processing and Sensing Configuration

In this configuration, the ATmega 128L microcontroller was replaced by its software version written in VHDL. The VHDL software is presented in Appendix A. The signal adaption module was also incorporated. The soft core of the microcontroller was uploaded to both the Xilinx Spartan 3L and the Quick Logic Eclipse II FPGAs. Even though the standby power of the Quick Logic FPGA is lower than the Spartan 3L, it was necessary to perform field tests to establish the advantages of each with respect to power consumption. The Altera Cyclone and Actel ProASIC Plus were not used since they do not have a power-down mode, which means their power consumptions are much higher than the Spartan 3L and Eclipse II FPGAs. The core of the microcontroller implemented in an FPGA was interfaced to the adaptation signal block and the flash memory of the mote. The RF radio and the other components were configurable as in the traditional architecture. Figure 59 depicts the architecture for this configuration.

Figure 59: Architecture of the Reconfigurable Processing and Sensing Configuration

In order to use similar programming tools, pin-to-pin compatibility was preserved. However, in addition to the software modifications performed in previous stages it was
necessary to modify the TinyOS modules that control the sleep modes in the mote. The code had to be modified to handle the power mode hardware operation supported by the Spartan 3L and Eclipse II.

### 8.1.1.3 Additional Processing Configuration

An extra processing unit was incorporated in this configuration in order to avoid having the microcontroller perform specific time and power consuming processes that could be processed efficiently by the reconfigurable devices. In this approach, the standard microcontroller was used primarily for the sensor signal adaption block. The extra processing unit was powered by the microcontroller when specific processes needed to be performed in extra devices either because the microcontroller did not have the resources to execute or it was not a power efficient operation. The extra processor unit could perform data and image compressions to avoid an increased data rate, which would eventually increase power consumption. Similarly, more efficient transmission schemes could further reduce the power consumption. Figure 60 displays the integration of an additional processing unit in the reconfigurable mote.
8.1.1.4 DSSS and Zigbee Reconfigurable Radio Configuration

Apart from incorporating the reconfigurable processors, optimization of the RF radio would also improve performance. The interoperability feature was obtained by using a dual band antenna in the prototyping device. This antenna operated at the 433 MHz and 900 MHz frequencies of the ISM bands. The radio could operate on either one or both of the frequencies. An external antenna was chosen in order to maximize range coverage.

This proposed mote concept used a Direct Sequence Spread Spectrum, (DSSS), approach in the physical layer, which was used in both the traditional and Zigbee radios. The radio implementation was performed with the Xilinx Spartan 3L FPGA and the CoolRunner CPLD. Although the tendency in Software Defined Radio is to use FPGAs, due to their inherent flexibility, the CPLD advantages in power consumption compels a comparison of the performances of these two devices for implementation of the baseband processing. The FPGA implementation used a hardware tool from Xilinx, called System
Generator, for modeling the system. System Generator runs on top of the Matlab Simulink environment enabling the use of the development and simulation resources available in Matlab. System Generator can also be used for designing and testing DSP systems in visual flow environments. The signal processing algorithms were tested and verified for their functionalities. Once System Generator was added as a library to Matlab a system model was created in Simulink. The library included several hardware models for DSP, communications, controls and visualization. Logic, arithmetic elements and memory blocks were also available. Libraries of FIR filters, state machines, FFTs and data converters were also available and used. After the design of the desired block was performed in Matlab, System Generator produced the VHDL files, timing constraints files and test bench for the target device. In addition, System Generator produced the project files required to perform the compilation and synthesis. Figure 61 describes the design flow using the Xilinx System Generator.
Each of the blocks of the transmitter and the receiver were designed using the System Generator in Matlab in order to generate the VHDL files, which were used in the ISE Synthesis tool from Xilinx to generate the programming core for the FPGA or CPLD. ISE can link to simulation tools such as ModelSim. The modulator design for Differential Quadrature Phase Shift Keying, (DQPSK), is presented in Figure 62.
8.1.1.5 Fully Reconfigurable Radio Configuration

The final step in the proposed concept is to integrate all of the reconfigurable blocks studied in the previous configurations. The operation of the fully reconfigurable mote relies on the roles defined in Chapter 4 for sensing, cluster head, repeater and slave operations. The architecture of the fully reconfigurable mote is presented in Figure 63.
The soft processor and the reconfigurable radio blocks handle sleep, standby, and power-up modes. The additional processing, and sensor-signal adaption blocks power down in order to reduce power consumption. Depending upon their functionalities some of the blocks deactivate in accordance with to the matrix presented in Table 8.

Table 8: Modules Activation Matrix for the Proposed Solution

<table>
<thead>
<tr>
<th>Role</th>
<th>Soft Processor</th>
<th>Reconfigurable Radio</th>
<th>Signal Adaptation</th>
<th>Optional Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Head Repeater Slave</td>
<td>Normal</td>
<td>Normal</td>
<td>Off</td>
<td>Activate</td>
</tr>
<tr>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
<td>Off</td>
<td>Off</td>
</tr>
</tbody>
</table>
8.2 Proof of Concept

8.2.1 Simulation Environment

The development of the proposed architecture was performed on the TinyOS Simulator, (TOSSIM). After implementing each of the modules in hardware, the TinyOS components that represented the exact functionality of the actual hardware had to be developed. In addition, the power estimation had to be integrated. The advantage of this approach is that the TinyOS code developed for simulation can be reused in the implementation. Figure 64 presents the simulation results for a Wireless Sensor Network consisting of 10 motes that was deployed in an area of 1000 feet by 1000 feet. The Radio Model employed was "Empirical" and based on an outdoor trace of packet connectivity for RFM1000 radios. This radio model established the bit error profile of the link. In this simulation, the energy consumption was estimated in each mote for both the radio and CPU. Similarly, radio links were established and radio packets were sent and quantified.
Figure 64: Simulation of a WSN of 10 Motes using TOSSIM

Figure 65 displays the energy consumption of each mote. It is important to notice that the power consumption is evenly distributed in all the motes in order to avoid an overload on any single mote.

Figure 65: Energy Consumption Simulation of a WSN of 10 Motes using TOSSIM
8.2.2 Field Tests

Field Tests were performed in a rural zone in Maracay, Venezuela. Ten Mica2 motes were deployed randomly to cover an area of 2100 feet by 900 feet. A Surge-Reliable core was uploaded to each mote, which was configured to run at maximum power. Some of the motes were fixed at specific locations while others were mobile. The topology and the characteristics of the zone are pictured in Figure 66.

![Field Test in Venezuela](image)

Figure 66: Field Test in Venezuela

The purpose of the field test was to evaluate the capabilities of the Chipcon CC1000 radio in such an environment. The Message Rate statistics are presented in Figure 67. The message rate remained relatively constant at all times except for the peaks presented at 12:31 and 12:35. These peaks were due to reconfiguration of the network by the change of positions of some of the nodes. Therefore, the peaks are not expected if the position of the motes is fixed in a remote sensing environment.
8.2.3 FPGA Functionality

The functionality of the Atmega128L core was simulated in Altera Quartus II, which was running on the Cyclone FPGA. In order to confirm the simulation the VHDL code was uploaded to the Xilinx Spartan 3 and the functionality confirmed with the Tektronix Logic Analyzer TLA714. Simulation results are presented in Figure 68.

Figure 67: Message Rate Statistics

Figure 68: Functionality Simulation of the AT Mega128L Core
8.2.4 Development Environment Setup

The development environment setup was designed to test the proposed concept. Figure 69 displays the interoperability of the components used for developing and testing the proposed wireless sensor network modifications.

Figure 69: Development Setup Architecture
CHAPTER 9

CONCLUSION AND THE FUTURE WORK

The concept of a reconfigurable mote for Wireless Sensor Networks was designed and developed. The desired system requirements for the development of the platform were identified. The different phases for the development setup were discussed including the architecture of a fully reconfigurable platform.

The remote maintenance and operation for hostile environments was considered. A conceptual solution that complies with the requirements presented in the original problem statement was developed.

Several programable devices were evaluated including low power Complex Programmable Logic Devices, (CPLDs,) and Field Programable Gate Arrays, (FPGAs). The most suitable reconfigurable devices for each of the components of the architecture were identified. In addition, the development and simulation tools required for the design and implementation of the reconfigurable motes were identified and evaluated. The re-configurability issues associated with the radio transmissions were also addressed. Direct Sequence Spread Spectrum and Zigbee concepts were considered in the proposed solution. Software Defined Radio, (SDR), techniques were also studied and included in the proposed reconfigurable architecture.
Powering Strategies were studied and simulation results were presented for the sensor-signal adaptation scheme. The architecture for an efficient powering module, for maximum system autonomy, was presented. An application-oriented development platform was conceived and integrated with the Rapid Organization and Situation Assessment, (ROSA), testbed.

By leveraging on the existing ROSA resources and a reliable laboratory field tests can be performed for new wireless sensor network architectures. New applications and research projects can use the developed test-bed to validate the hardware mote implementation. This approach will offer actual hardware results of the implementation in addition to the software simulation tools.

In the future, a new research project that integrates Wireless Sensor Networks and Networked Embedded Systems groups can be envisioned. The motivation and the requirements for disaster mitigation and environmental applications were presented in this thesis. However, this research represents a simple first step in the development of a prototype of the proposed state-of-the-art platform that meets all the requirements in these kinds of applications. Further laboratory and field tests have to be carried out in order to determine the performance, power consumption, reliability, flexibility and interoperability issues as well as other novel concepts.
REFERENCES


Appendix A VHDL

Code for the ATMEGA128L

The VHDL code used to implement the soft core of the AtMega128L was based on the architecture presented in Figure 70. It is important to clarify that the actual architecture of the microcontroller was adapted to enable its utilization in the proposed reconfigurable platform.

Figure 70: Base Architecture for the Atmega128L Core Written in VHDL
Appendix A (continued)

The VHDL code for the core of the Atmega128L consists of the following VHDL files:

- top_avr_core_sim.vhd,
- avr_core.vhd,
- AVRuCPackage.vhd,
- alu_avr.vhd,
- pm_fetch_dec.vhd,
- bit_processor.vhd,
- CPUWaitGenerator.vhd,
- io_adr_dec.vhd,
- io_reg_file.vhd,
- RAMDataReg.vhd,
- reg_file.vhd,
- Service_Module.vhd,
- simple_timer.vhd,
- Timer.Counter.vhd,
- Uart.vhd,
- DataRAM.vhd,
- PROM.vhd.

Only the top VHDL code for the core and simulation, top_avr_core_sim.vhd

and

avr_core.vhd,

are presented in these appendices. All other files are available upon request at ovgonzalez@hotmail.com.
Appendix A (continued)

File: top_avr_core_sim.vhd

-- Top entity for the simulation of the microcontroller Atmega128L from Atmel
-- Adapted for the microcontroller Atmega128L using as a base the Atmega 103L Core
-- from www.opencores.org

library ieee;
use ieee.std_logic_1164.all;
use WORK.AVRuCPackage.all;

entity top_avr_core_sim is
  generic(InsertWaitSt: Boolean = FALSE; RAMSize: positive = 128);
  port(
    ireset: in std_logic;
    cp2: in std_logic;
    porta: inout std_logic_vector(7 downto 0);
    portb: inout std_logic_vector(7 downto 0);

    -- UART
    rxd: in std_logic;
    txd: out std_logic;

    -- External interrupt inputs
    nINT0: in std_logic;
    nINT1: in std_logic;
    nINT2: in std_logic;
    nINT3: in std_logic;
    INT4: in std_logic;
    INT5: in std_logic;
    INT6: in std_logic;
    INT7: in std_logic
  );
end entity top_avr_core_sim;

architecture Struct of top_avr_core_sim is
  component pport is
    generic(
      PORTX_Adr: std_logic_vector(IOAdrWidth-1 downto 0);
      DDRX_Adr: std_logic_vector(IOAdrWidth-1 downto 0);
      PINX_Adr: std_logic_vector(IOAdrWidth-1 downto 0)
    );
    port(
      -- AVR Control
      ireset: in std_logic;
      cp2: in std_logic;

      -- UART
      rxd: in std_logic;
      txd: out std_logic;

      -- External interrupt inputs
      nINT0: in std_logic;
      nINT1: in std_logic;
      nINT2: in std_logic;
      nINT3: in std_logic;
      INT4: in std_logic;
      INT5: in std_logic;
      INT6: in std_logic;
      INT7: in std_logic
    );
end component pport;
Appendix A (continued)

```vhdl
adr: in std_logic_vector(5 downto 0);
dbus_in: in std_logic_vector(7 downto 0);
dbus_out: out std_logic_vector(7 downto 0);
iore: in std_logic;
iowe: in std_logic;
out_en: out std_logic;

-- External connection
portx: out std_logic_vector(7 downto 0);
ddrx: out std_logic_vector(7 downto 0);
pinx: in std_logic_vector(7 downto 0));
```

```vhdl
end component pport;
```

```vhdl
component external_mux is

  port( 
    ramre: in std_logic;
    dbus_out: out std_logic_vector (7 downto 0);
    ram_data_out: in std_logic_vector (7 downto 0);
    io_port_bus: in ext_mux_din_type;
    io_port_en_bus: in ext_mux_en_type;
    irqack: in std_logic;
    irqackad: in std_logic_vector(4 downto 0);
    ind_irq_ack: out std_logic_vector(22 downto 0);
  );

end component external_mux;
```

```vhdl
component Service_Module is

  port( 
    -- AVR Control
    ireset: in std_logic;
    cp2: in std_logic;
    adr: in std_logic_vector(5 downto 0);
    dbus_in: in std_logic_vector(7 downto 0);
    dbus_out: out std_logic_vector(7 downto 0);
    iore: in std_logic;
    iowe: in std_logic;
    out_en: out std_logic;

    -- SLEEP mode signals
    sleep_en: out std_logic;

    -- SRAM control signals
    ESRAM_en: out std_logic;
    ESRAM_WS: out std_logic;
  );
```
--IRQ
ExtInt_IRQ: out std_logic_vector(7 downto 0);
ExtInt_IRQ_Ack: in std_logic_vector(3 downto 0);

-- External interrupts (inputs)
Ext_Int_In: in std_logic_vector(7 downto 0));

end component Service_Module;

component RAMDataReg is
  port(  
    ireset: in std_logic;
    cp2: in std_logic;
    cpuwait: in std_logic;
    RAMDataIn: in std_logic_vector(7 downto 0);
    RAMDataOut: out std_logic_vector(7 downto 0);
  )
end component RAMDataReg;

component Simple_Timer is
  port(  
    ireset: in std_logic;
    cp2: in std_logic;
    irqline: out std_logic;
    timer_irqack: in std_logic
  )
end component Simple_Timer;

component Timer.Counter is
  port(  
    -- AVR Control
    ireset: in std_logic;
    cp2: in std_logic;
    adr: in std_logic_vector(5 downto 0);
    dbus_in: in std_logic_vector(7 downto 0);
    dbus_out: out std_logic_vector(7 downto 0);
    iore: in std_logic;
    iowe: in std_logic;
    out_en: out std_logic;

    -- Timer/Counters
    EXT1: in std_logic;
    EXT2: in std_logic;
  )
end component Timer.Counter;
Appendix A (continued)

Tosc1: in std_logic;
OC0_PWM0: out std_logic;
OC1A_PWM1A: out std_logic;
OC1B_PWM1B: out std_logic;
OC2_PWM2: out std_logic;

--IRQ
TC0OvfIRQ: out std_logic;
TC0OvfIRQ_Ack: in std_logic;
TC0CmpIRQ: out std_logic;
TC0CmpIRQ_Ack: in std_logic;
TC2OvfIRQ: out std_logic;
TC2OvfIRQ_Ack: in std_logic;
TC2CmpIRQ: out std_logic;
TC2CmpIRQ_Ack: in std_logic;
TC1OvfIRQ: out std_logic;
TC1OvfIRQ_Ack: in std_logic;
TC1CmpAIRQ: out std_logic;
TC1CmpAIRQ_Ack: in std_logic;
TC1CmpBIRQ: out std_logic;
TC1CmpBIRQ_Ack: in std_logic;
TC1ICIRQ: out std_logic;
TC1ICIRQ_Ack: in std_logic);

end component Timer_Counter;

component uart is
  port(
    -- AVR Control
    ireset : in std_logic;
    cp2: in std_logic;
    adr: in std_logic_vector(5 downto 0);
    dbus_in: in std_logic_vector(7 downto 0);
    dbus_out: out std_logic_vector(7 downto 0);
    iore: in std_logic;
    iowe: in std_logic;
    out_en: out std_logic;

    --UART
    rxd: in std_logic;
    rx_en: out std_logic;
    txd: out std_logic;
    tx_en: out std_logic;
  )
end component;
Appendix A (continued)

```vhdl
--IRQ
txcirq: out std_logic;
txc irqack: in std_logic;
udreirq: out std_logic;
rxcirq: out std_logic);
end component uart;

component PROM is
  port (  
    address_in : in std_logic_vector(15 downto 0);  
data_out: out std_logic_vector(15 downto 0));
end component PROM;

component DataRAM is
  generic(RAMSize:positive);
  port (  
    cp2: in std_logic;
    address: in std_logic_vector(LOG2(RAMSize)-1 downto 0);
    ramwe: in std_logic;
    din: in std_logic_vector(7 downto 0);
    dout: out std_logic_vector(7 downto 0));
end component DataRAM;

component CPUWaitGenerator is
  generic(InsertWaitSt: Boolean);
  port (  
    ireset: in std_logic;
    cp2: in std_logic;
    ramre: in std_logic;
    ramwe: in std_logic;
    cpuwait: out std_logic);
end component CPUWaitGenerator;

component avr_core is
  port (  
    cp2: in std_logic;
    ireset: in std_logic;
    cpuwait: in std_logic;

    -- PROGRAM MEMORY PORTS
    pc: out std_logic_vector(15 downto 0);
    inst: in std_logic_vector(15 downto 0);

    -- I/O REGISTERS PORTS

123
```
Appendix A (continued)

    adr: out std_logic_vector(5 downto 0);
iore: out std_logic;
iowe: out std_logic;

    -- DATA MEMORY PORTS
    ramadr: out std_logic_vector(15 downto 0);
    ramre: out std_logic;
    ramwe: out std_logic;
    dbusin: in std_logic_vector(7 downto 0);
    dbusout: out std_logic_vector(7 downto 0);

    -- INTERRUPTS PORT
    irqlines: in std_logic_vector(22 downto 0);
    irqack: out std_logic;
    irqackad: out std_logic_vector(4 downto 0);

end component avr_core;

-- Signals connected directly to the core
signal sg_core_cpuwait: std_logic:= '0';

-- Program memory
signal sg_core_pc: std_logic_vector(15 downto 0):= (others => '0');
    -- PROM address
signal sg_core_inst : std_logic_vector(15 downto 0):= (others => '0');
    -- PROM data

-- I/O registers
signal sg_core_adr: std_logic_vector(5 downto 0):=(others => '0');
signal sg_core_iore: std_logic:= '0';
signal sg_core_iowe : std_logic:= '0';

-- Data memory
signal sg_core_ramadr: std_logic_vector(15 downto 0):= (others => '0');
signal sg_core_ramre: std_logic:= '0';
signal sg_core_ramwe: std_logic:= '0';
signal sg_core_dbusin: std_logic_vector(7 downto 0):= (others => '0');
signal sg_core_dbusout: std_logic_vector(7 downto 0):= (others => '0');

-- Interrupts
signal sg_core_irqlines: std_logic_vector(22 downto 0):= (others => '0');
signal sg_core_irqack : std_logic :='0';
signal sg_core_IRQackad : std_logic_vector(4 downto 0):= (others => '0');
Appendix A (continued)

-- Signals connected directly to the SRAM controller
signal sg_ram_din: std_logic_vector (7 downto 0):= (others => '0');
signal sg_ram_dout : std_logic_vector (7 downto 0):= (others => '0');

-- Signals connected directly to the I/O registers

-- PortA
signal sg_porta_dbusout: std_logic_vector(7 downto 0):= (others => '0');
signal sg_porta_out_en: std_logic:= '0';

-- PortB
signal sg_portb_dbusout: std_logic_vector(7 downto 0):= (others => '0');
signal sg_portb_out_en: std_logic:= '0';

-- UART
signal sg_uart_dbusout: std_logic_vector(7 downto 0):= (others => '0');
signal sg_uart_out_en: std_logic:= '0';
signal sg_uart_tx_en: std_logic:= '0';
signal sg_uart_rx_en: std_logic:= '0';

-- Timer/Counter
signal sg_tc_dbusout: std_logic_vector(7 downto 0):= (others => '0');
signal sg_tc_out_en: std_logic:= '0';

-- Service module
signal sg_sm_dbusout: std_logic_vector(7 downto 0):= (others => '0');
signal sg_sm_out_en: std_logic:= '0';

-- Signals connected directly to the external multiplexer
signal sg_io_port_out: ext_mux_din_type:= (others => "00000000");
signal sg_io_port_out_en: ext_mux_en_type:= (others => '0');
signal sg_ind_irq_ack: std_logic_vector(sg_core_irqlines'range):=
    (others => '0');

-- External interrupts signal
signal sg_ext_int_req: std_logic_vector(7 downto 0):= (others => '0');

-- Reset signals
signal sg_nrst_cp2: std_logic:= '0';

-- Port signals
signal PortAReg: std_logic_vector(porta'range):= (others => '0');
signal DDRAReg: std_logic_vector(porta'range):= (others => '0');
signal PortBReg: std_logic_vector(porta'range):= (others => '0');
Appendix A (continued)

signal DDRBReg: std_logic_vector(porta'range):= (others => '0');

begin
  
  TESTING_CORE: component avr_core port map
  (  
    cp2 => cp2,
    ireset => ireset,
    cpuwait => sg_core_cpuwait,

    -- PROGRAM MEMORY PORTS
    pc => sg_core_pc,
    inst => sg_core_inst,

    -- I/O REGISTERS PORTS
    adr => sg_core_adr,
    iore => sg_core_iore,
    iowe => sg_core_iowe,

    -- DATA MEMORY PORTS
    ramadr => sg_core_ramadr,
    ramre => sg_core_ramre,
    ramwe => sg_core_ramwe,
    dbusin => sg_core_dbusin,
    dbusout => sg_core_dbusout,

    -- INTERRUPTS PORT
    irqlines => sg_core_irqlines,
    irqack => sg_core_irqack,
    irqackad => sg_core_irqackad
  );

  RAM_Data_Register: component RAMDataReg port map
  (  
    ireset => ireset,
    cp2 => cp2,
    cpuwait => sg_core_cpuwait,
    RAMDataIn => sg_core_dbusout,
    RAMDataOut => sg_ram_din
  );

  -- Program memory
  PM: component PROM port map
  (  
    address_in => sg_core_pc,
    data_out => sg_core_inst
  );
Appendix A (continued)

-- Data memory

DM: component DataRAM
generic map (RAMSize => RAMSize)
port map
(
  cp2 => cp2,
  address => sg_core_ramadr(LOG2(RAMSize)-1)
downto 0),
  ramwe => sg_core_ramwe,
  din => sg_ram_din,
  dout => sg_ram_dout);

-- CPUWait generation

CPUWait_Gen: component CPUWaitGenerator
generic map (InsertWaitSt => InsertWaitSt)
port map
(
  ireset => ireset,
  cp2 => cp2,
  ramre => sg_core_ramre,
  ramwe => sg_core_ramwe,
  cpuwait => sg_core_cpuwait
);

EXT_MUX: component external_mux port map
(
  ramre => sg_core_ramre, -- ramre output of the core
dbus_out => sg_core_dbusin, -- Data input of the core
ram_data_out => sg_ram_dout, -- Data output of the RAM
io_port_bus => sg_io_port_out, -- Data outputs of the I/O
  io_port_en_bus => sg_io_port_out_en, -- Out enable
  irqack => sg_core_irqack,
  irqackad => sg_core_irqackad,
  ind_irq_ack => sg_ind_irq_ack -- Individual interrupt
  -- acknowledge for the peripheral
);

-- PORTA

PORTA_COMP: component pport generic map
(
  PORTX_Adr => PORTA_Address,
  DDRX_Adr => DDRA_Address,
  PINX_Adr => PINA_Address
)
port map
Appendix A (continued)

(  
   -- AVR Control
   ireset => ireset,
   cp2 => cp2,
   adr => sg_core_adr,
   dbus_in => sg_core_dbusout,
   dbus_out => sg_porta_dbusout,
   iore => sg_core_iore,
   iowe => sg_core_iowe,
   out_en => sg_porta_out_en,

   -- External connection
   portx => PortAReg,
   ddrx => DDRAReg,
   pinx => porta
);

-- PORTA connection to the external multiplexer
   sg_io_port_out(0) <= sg_porta_dbusout;
   sg_io_port_out_en(0) <= sg_porta_out_en;

-- Tri-state control for PORTA
   PortAZCtrl: for i in porta'range generate
      porta(i) <= PortAReg(i) when DDRAReg(i)='1' else 'Z';
   end generate PortAZCtrl;

-- PORTB
   PORTB_COMP: component pport
      generic map
         ( PORTX_Adr => PORTB_Address,
           DDRX_Adr => DDRB_Address,
           PINX_Adr => PINB_Address
         )
      port map
         (  
           -- AVR Control
           ireset => ireset,
           cp2 => cp2,
           adr => sg_core_adr,
           dbus_in => sg_core_dbusout,
           dbus_out => sg_portb_dbusout,
           iore => sg_core_iore,
           iowe => sg_core_iowe,
           out_en => sg_portb_out_en,
Appendix A (continued)

-- External connection
portx => PortBReg,
ddrx=> DDRBReg,
pxn => portb
);

-- PORTB connection to the external multiplexer
sg_io_port_out(1) <= sg_portb_dbusout;
sg_io_port_out_en(1) <= sg_portb_out_en;

-- Tri-state control for PORTB
PortBZCtrl: for i in portb'range generate
      portb(i) <= PortBReg(i) when DDRBReg(i)='1' else 'Z';
end generate;

-- Simple timer
TIMER: component simple_timer port map
      (ireset => ireset,
       cp2 => cp2,
       irqline => sg_ext_int_req(0),
       timer_irqack => sg_ind_irq_ack(0));

sg_core_irqlines(22 downto 20) <= (others => '0');
sg_core_irqlines(13 downto 10) <= (others => '0');

UART_AVR: component uart port map
      (ireset => ireset,
       cp2 => cp2,
       adr => sg_core_adr,
       dbus_in => sg_core_dbusout,
       dbus_out => sg_uart_dbusout,
       iore => sg_core_iore,
       iowe => sg_core_iowe,
       out_en => sg_uart_out_en,

--UART
rx => rxd,
rx_en => sg_uart_rx_en,
txd => txd,
tx_en => sg_uart_tx_en,
-- UART connection to the external multiplexer
  sg_io_port_out(2) <= sg_uart_dbusout;
  sg_io_port_out_en(2) <= sg_uart_out_en;

-- Timer/Counter

TIM_CNT: component Timer.Counter port map
(
  -- AVR Control
  ireset => ireset,
  cp2 => cp2,
  adr => sg_core_adr,
  dbus_in => sg_core_dbusout,
  dbus_out => sg_tc_dbusout,
  iore => sg_core_iore,
  iowe => sg_core_iowe,
  out_en => sg_tc_out_en,

  -- Timer/Counters
  EXT1 => '0',
  EXT2 => '0',
  Tose1 => '0',
  OC0_PWM0 => open,
  OC1A_PWM1A => open,
  OC1B_PWM1B => open,
  OC2_PWM2 => open,

  -- IRQ
  TC0OvfIRQ => sg_core_irqlines(15), -- Timer/Counter0
              -- overflow ($0020)
  TC0OvfIRQ_Ack => sg_ind_irq_ack(15),
  TC0CmpIRQ => sg_core_irqlines(14), -- Timer/Counter0
              -- Compare Match ($001E)
  TC0CmpIRQ_Ack => sg_ind_irq_ack(14),
  TC2OvfIRQ => sg_core_irqlines(9), -- Timer/Counter2
              -- overflow ($0014)
Appendix A (continued)

TC2OvfIRQ_Ack => sg_ind_irq_ack(9),
TC2CmpIRQ => sg_core_irqlines(8), -- Timer/Counter2
    -- Compare Match ($0012)
TC2CmpIRQ_Ack => sg_ind_irq_ack(8),
TC1OvfIRQ => open,
TC1OvfIRQ_Ack => '0',
TC1CmpAIRQ => open,
TC1CmpAIRQ_Ack => '0',
TC1CmpBIRQ => open,
TC1CmpBIRQ_Ack => '0',
TC1ICIRQ => open,
TC1ICIRQ_Ack => '0'
);

-- Timer/Counter connection to the external multiplexer
sg_io_port_out(3) <= sg_tc_dbusout;
sg_io_port_out_en(3) <= sg_tc_out_en;
Serv_Module: component Service_Module port map
  (  
    -- AVR Control
    ireset => ireset,
    cp2 => cp2,
    adr => sg_core_adr,
    dbus_in => sg_core_dbusout,
    dbus_out => sg_sm_dbusout,
    iore => sg_core_iore,
    iowe => sg_core_iowe,
    out_en => sg_sm_out_en,

    -- SLEEP mode signals
    sleep_en => open,

    -- SRAM control signals
    ESRAM_en => open,
    ESRAM_WS => open,

    --IRQ
    ExtInt_IRQ => sg_core_irqlines(7 downto 0),
    ExtInt_IRQ_Ack => sg_ind_irq_ack(7 downto 4),

    -- External interrupts (inputs)
    Ext_int_In => sg_ext_int_req
  );

-- Service module connection to the external multiplexer
sg_io_port_out(4) <= sg_sm_dbusout;

131
Appendix A (continued)

    sg_io_port_out_en(4) <= sg_sm_out_en;

-- External interrupt inputs
    sg_ext_int_req(0) <= nINT0;  -- May be uncommented only if simple_timer
        -- is disconnected from sg_ext_int_req(0)

    sg_ext_int_req(1) <= nINT1;
    sg_ext_int_req(2) <= nINT2;
    sg_ext_int_req(3) <= nINT3;
    sg_ext_int_req(4) <= INT4;
    sg_ext_int_req(5) <= INT5;
    sg_ext_int_req(6) <= INT6;
    sg_ext_int_req(7) <= INT7;
end architecture Struct;

File: avr_core.vhd

-- Top entity for the Atmega128L Code
-- Adapted for Atmega128L using as a base the Atmega 103L Core

library ieee;
use ieee.std_logic_1164.all;
entity avr_core is
    port(cp2: in std_logic;
        ireset: in std_logic;
        cpuwait: in std_logic;

        -- PROGRAM MEMORY PORTS
        pc: out std_logic_vector(15 downto 0);  -- CORE OUTPUT
        inst: in std_logic_vector(15 downto 0);  -- CORE INPUT

        -- I/O REGISTERS PORTS
        adr: out std_logic_vector(5 downto 0);  -- CORE OUTPUT
        iore: out std_logic;  -- CORE OUTPUT
        iowe: out std_logic;  -- CORE OUTPUT

        -- DATA MEMORY PORTS
        ramadr: out std_logic_vector(15 downto 0);
        ramre: out std_logic;
        ramwe: out std_logic;
        dbusin: in std_logic_vector(7 downto 0);
        dbusout: out std_logic_vector(7 downto 0);

end entity avr_core;
Appendix A (continued)

-- INTERRUPTS PORT
irqlines: in std_logic_vector(22 downto 0);
irqack: out std_logic;
irqackad: out std_logic_vector(4 downto 0);
end entity avr_core;

architecture struct of avr_core is

component pm_fetch_dec is
  port(
    -- EXTERNAL INTERFACES OF THE CORE
    clk: in std_logic;
    nrst: in std_logic;
    cpuwait: in std_logic;

  -- PROGRAM MEMORY PORTS
  pc: out std_logic_vector(15 downto 0); -- CORE OUTPUT
  inst: in std_logic_vector(15 downto 0); -- CORE INPUT

  -- I/O REGISTER PORTS
  adr: out std_logic_vector(5 downto 0); -- CORE OUTPUT
  iore: out std_logic; -- CORE OUTPUT
  iowe: out std_logic; -- CORE OUTPUT

  -- DATA MEMORY PORTS
  ramadr: out std_logic_vector(15 downto 0);
  ramre: out std_logic;
  ramwe: out std_logic;
  dbusin: in std_logic_vector(7 downto 0);
  dbusout: out std_logic_vector(7 downto 0);

  -- INTERRUPTS PORT
  irqlines: in std_logic_vector(22 downto 0);
  irqack: out std_logic;
  irqackad: out std_logic_vector(4 downto 0);

  -- END OF THE CORE INTERFACES

  -- INTERFACES TO THE ALU
  alu_data_r_in: out std_logic_vector(7 downto 0);
  alu_data_d_in: out std_logic_vector(7 downto 0);

  -- OPERATION SIGNALS INPUTS
  ide_add_out: out std_logic;
idc_adc_out: out std_logic;
idc_adiw_out: out std_logic;
idc_sub_out: out std_logic;
idc_subi_out: out std_logic;
idc_sbc_out: out std_logic;
idc_sbci_out: out std_logic;
idc_sbiw_out: out std_logic;
adiw_st_out: out std_logic;
sbiw_st_out: out std_logic;
idc_and_out: out std_logic;
idc_andi_out: out std_logic;
idc_or_out: out std_logic;
idc_ori_out: out std_logic;
idc_eor_out: out std_logic;
idc_com_out: out std_logic;
idc_neg_out: out std_logic;
idc_inc_out: out std_logic;
idc_dec_out: out std_logic;
idc_cp_out: out std_logic;
idc_cpc_out: out std_logic;
idc_cpi_out: out std_logic;
idc_cpse_out: out std_logic;
idc_lsr_out: out std_logic;
idc_ror_out: out std_logic;
idc_asr_out: out std_logic;
idc_swap_out: out std_logic;

-- DATA OUTPUT
alu_data_out: in std_logic_vector(7 downto 0);

-- FLAGS OUTPUT
alu_c_flag_out: in std_logic;
alu_z_flag_out: in std_logic;
alu_n_flag_out: in std_logic;
alu_v_flag_out: in std_logic;
alu_s_flag_out: in std_logic;
alu_h_flag_out: in std_logic;

-- INTERFACES TO THE GENERAL PURPOSE
-- REGISTER FILE
reg_rd_in: out std_logic_vector(7 downto 0);
reg_rd_out: in std_logic_vector(7 downto 0);
reg_rd_adr: out std_logic_vector(4 downto 0);
reg_rr_out: in std_logic_vector(7 downto 0);
Appendix A (continued)

reg_rr_adr: out std_logic_vector(4 downto 0);
reg_rd_wr: out std_logic;
post_inc: out std_logic; -- POST INCREMENT FOR
-- LD/ST INSTRUCTIONS
pre_dec: out std_logic; -- PRE DECREMENT FOR LD/ST
-- INSTRUCTIONS
reg_h_wr: out std_logic;
reg_h_out: in std_logic_vector(15 downto 0);
reg_h_adr: out std_logic_vector(2 downto 0); -- x, y, z
reg_z_out: in std_logic_vector(15 downto 0); -- OUTPUT
-- OF R31:R30 FOR LPM/ELPM/IJMP
-- INSTRUCTIONS

-- INTERFACES TO THE INPUT/OUTPUT
-- REGISTER FILE
adr: out std_logic_vector(5 downto 0);
iowe: out std_logic;

dbusout: out std_logic_vector(7 downto 0); -- OUTPUT
-- OF THE CORE
sreg_fl_in: out std_logic_vector(7 downto 0);
sreg_out: in std_logic_vector(7 downto 0);
sreg_fl_wr_en: out std_logic_vector(7 downto 0);

-- FLAGS WRITE ENABLE SIGNALS
spl_out: in std_logic_vector(7 downto 0);
spn_out: in std_logic_vector(7 downto 0);
sp_ndown_up: out std_logic; -- DIRECTION OF
-- CHANGING OF STACK POINTER
-- SPH:SPL 0->UP(+) 1->DOWN(-)
sp_en: out std_logic; -- WRITE ENABLE
-- (COUNT ENABLE) FOR
-- AND SPL REGISTERS
rampz_out: in std_logic_vector(7 downto 0);

-- INTERFACES TO THE INPUT/OUTPUT ADDRESS
-- DECODER
ram_data_in: in std_logic_vector(7 downto 0);
adr: in std_logic_vector(5 downto 0);
iore: in std_logic; -- CORE SIGNAL
ramre: in std_logic; -- CORE SIGNAL
dbusin: out std_logic_vector(7 downto 0); -- CORE
-- SIGNAL
-- INTERFACES TO THE BIT PROCESSOR
bit_num_r_io: out std_logic_vector(2 downto 0); -- BIT
    -- NUMBER FOR
    -- CBI/SBI/BLD/BST/SBRS/SBRC/SBIS
    -- INSTRUCTIONS
dbusin: in std_logic_vector(7 downto 0);
    -- SBI/CBI/SBIS/SBIC IN
bitpr_io_out: in std_logic_vector(7 downto 0); -- SBI/CBI
    -- OUT
branch: out std_logic_vector(2 downto 0); -- NUMBER
    -- (0..7) OF BRANCH CONDITION FOR
    -- BRBS/BRBC INSTRUCTION
bit_pr_sreg_out: in std_logic_vector(7 downto 0);
    -- BCLR/BSET/BST (T-FLAG ONLY)
sreg_bit_num: out std_logic_vector(2 downto 0); -- BIT
    -- NUMBER FOR BCLR/BSET INSTRUCTIONS
bld_op_out: in std_logic_vector(7 downto 0); -- BLD
    -- OUT (T FLAG)
bit_test_op_out: in std_logic; -- OUTPUT OF
    -- SBIC/SBIS/SBRS/SBRC
    -- OPERATION SIGNALS INPUTS

-- INSTRUCTIONS AND STATES
ide_sbi_out: out std_logic;
sbi_st_out: out std_logic;
ide_cbi_out: out std_logic;
cbi_st_out: out std_logic;
ide_bld_out: out std_logic;
ide_bst_out: out std_logic;
ide_bset_out: out std_logic;
ide_bclr_out: out std_logic;
ide_sbic_out: out std_logic;
ide_sbis_out: out std_logic;
ide_sbrs_out: out std_logic;
ide_sbrs_out: out std_logic;
ide_sbrc_out: out std_logic;
ide_brbs_out: out std_logic;
ide_brbc_out: out std_logic;
ide_reti_out: out std_logic;

-- END OF INTERFACES TO THE OTHER BLOCKS
);
end component pm_fetch_dec;
Appendix A (continued)

component alu_avr is
    port(
        alu_data_r_in: in std_logic_vector(7 downto 0);
        alu_data_d_in: in std_logic_vector(7 downto 0);
        alu_c_flag_in: in std_logic;
        alu_z_flag_in: in std_logic;
        alu_data_out: out std_logic_vector(7 downto 0);
        alu_c_flag_out: out std_logic;
        alu_z_flag_out: out std_logic;
        alu_n_flag_out: out std_logic;
        alu_v_flag_out: out std_logic;
        -- OPERATION SIGNALS INPUTS
        idc_add: in std_logic;
        idc_adc : in std_logic;
        idc_adiw: in std_logic;
        idc_sub: in std_logic;
        idc_subi: in std_logic;
        idc_sbc: in std_logic;
        idc_sbc: in std_logic;
        idc_sbc: in std_logic;
        idc_sbc: in std_logic;
        idc_sbiw: in std_logic;
        adiw_s: in std_logic;
        sbiw_st: in std_logic;
        idc_and: in std_logic;
        idc_and: in std_logic;
        idc_or: in std_logic;
        idc_ori: in std_logic;
        idc_eor: in std_logic;
        idc_com: in std_logic;
        idc_neg: in std_logic;
        idc_inc: in std_logic;
        idc_dec in std_logic;
        idc_cp: in std_logic;
        idc_cpc: in std_logic;
        idc_cpi: in std_logic;
        idc_cpse: in std_logic;
        idc_lsr: in std_logic;
        idc_ror: in std_logic;
        idc_asr: in std_logic;
        idc_swap: in std_logic;
    )
end component;
Appendix A (continued)

```vhdl
alu_s_flag_out: out std_logic;
alu_h_flag_out: out std_logic
);

dcomponent alu_avr;

ccomponent reg_file is
  generic(ResetRegFile: boolean);
  port(
    reg_rd_in: in std_logic_vector(7 downto 0);
    reg_rd_out: out std_logic_vector(7 downto 0);
    reg_rd_adr: in std_logic_vector(4 downto 0);
    reg_rr_out: out std_logic_vector(7 downto 0);
    eg_rr_adr: in std_logic_vector(4 downto 0);
    reg_rd_wr: in std_logic;
    post_inc: in std_logic; -- POST INCREMENT FOR LD/ST
      -- INSTRUCTIONS
    pre_dec: in std_logic; -- PRE DECREMENT FOR LD/ST
      -- INSTRUCTIONS
    reg_h_wr: in std_logic;
    reg_h_out: out std_logic_vector(15 downto 0);
    reg_h_adr: in std_logic_vector(2 downto 0); -- x, y, z
    reg_z_out: out std_logic_vector(15 downto 0);
      -- OUTPUT OF R31:R30 FOR
    clk: in std_logic;
    nrst: in std_logic
  );

dcomponent io_reg_file is
  port ( 
    clk: in std_logic;
    nrst: in std_logic;
    adr: in std_logic_vector(5 downto 0);
    iowe: in std_logic;
    dbusout: in std_logic_vector(7 downto 0);
    sreg_fl_in: in std_logic_vector(7 downto 0);
    sreg_out: out std_logic_vector(7 downto 0);
    sreg_fl_wr_en: in std_logic_vector(7 downto 0); --FLAGS
      -- WRITE ENABLE SIGNALS
  );
```

138
Appendix A (continued)

spl_out: out std_logic_vector(7 downto 0);
sph_out: out std_logic_vector(7 downto 0);
sp_ndown_up: in std_logic; -- DIRECTION OF
-- CHANGING OF STACK POINTER SPH:
-- SPL 0->UP(+) 1->DOWN(-)
sp_en: in std_logic; -- WRITE
-- ENABLE(COUNT ENABLE) FOR SPH
-- AND SPL REGISTERS
rampz_out: out std_logic_vector(7 downto 0)
);
end component;

component bit_processor is
    port(
        clk: in std_logic;
        nrst: in std_logic;
        bit_num_r_io: in std_logic_vector(2 downto 0); -- BIT
        -- NUMBER FOR CBI/SBI/BLD/BST/SBRS/
        -- SBRC/SBIC/SBIS
        -- INSTRUCTIONS
        dbusin: in std_logic_vector(7 downto 0);
        -- SBI/CBI/SBIS/SBIC IN
        bitpr_io_out: out std_logic_vector(7 downto 0);
        -- SBI/CBI OUT
        sreg_out: in std_logic_vector(7 downto 0);
        -- BRBS/BRBC/BLD IN
        branch: in std_logic_vector(2 downto 0);
        -- NUMBER(0..7) OF BRANCH CONDITION
        -- FOR BRBS/BRBC INSTRUCTION
        bit_pr_sreg_out: out std_logic_vector(7 downto 0);
        -- BCLR/BSET/BST(T-FLAG ONLY)
        sreg_bit_num : in std_logic_vector(2 downto 0); -- BIT
        -- NUMBER FOR BCLR/BSET INSTRUCTIONS
        bld_op_out : out std_logic_vector(7 downto 0); -- BLD
        -- OUT (T FLAG)
        reg_rd_out: in std_logic_vector(7 downto 0);
        -- BST/SBRS/SBRC IN
        bit_test_op_out: out std_logic; -- OUTPUT OF
        -- SBIC/SBIS/SBRS/SBRC

        -- OPERATION SIGNALS INPUTS

        -- INSTRUCTIONS AND STATES
        idc_sbi: in std_logic;
    );

end component;
Appendix A (continued)

```vhdl
sbi_st: in std_logic;
ide_cbi: in std_logic;
cbi_st: in std_logic;
ide_bld : in std_logic;
ide_bst: in std_logic;
ide_bset: in std_logic;
ide_bclr: in std_logic;
ide_sbic: in std_logic;
ide_sbis: in std_logic;
ide_sbrs: in std_logic;
ide_sbrsc: in std_logic;
idc_bld: in std_logic;
idc_bst: in std_logic;
idc_bset: in std_logic;
idc_bclr: in std_logic;
idc_sbic: in std_logic;
idc_sbis: in std_logic;
idc_sbrs: in std_logic;
idc_sbrsc: in std_logic;
idc_reti: in std_logic);
end component;

component io_adr_dec is
  port(
    adr: in std_logic_vector(5 downto 0);
    iore: in std_logic;
    dbusin_ext: in std_logic_vector(7 downto 0);
    dbusin_int: out std_logic_vector(7 downto 0);
    spl_out: in std_logic_vector(7 downto 0);
    sph_out: in std_logic_vector(7 downto 0);
    sreg_out: in std_logic_vector(7 downto 0);
    rampz_out: in std_logic_vector(7 downto 0));
end component;

signal sg_dbusin, sg_dbusout : std_logic_vector(7 downto 0):= (others => '0');
signal sg_adr: std_logic_vector(5 downto 0):= (others => '0');
signal sg_iowe, sg_iore:std_logic:= '0';

-- SIGNALS FOR INSTRUCTIONS AND STATES
signal sg_idc_add, sg_idc_adc, sg_idc_adiw, sg_idc_sub, sg_idc_subi,
sg_idc_sbc, sg_idc_sbcrc, sg_idc_sbcrp, sg_idc_sbcsv, sg_idc_sbcsvp,
sg_idc_and, sg_idc_andi, sg_idc_or, sg_idc_ori, sg_idc_com,
sg_idc_neg, g_idc_inc, sg_idc_dec, sg_idc_cp, sg_idc_cpe, sg_idc_cpi,
sg_idc_cpl, sg_idc_lsr, sg_idc_ror, sg_idc_asr: std_logic:= '0'

signal sg_idc_swap, sg_idc_sbi, sg_sbi_st, sg_idc_cbi, sg_cbi_st, sg_idc_bld,
sg_idc_bst, sg_idc_bset, sg_idc_bclr, sg_idc_sbic, sg_idc_sbis,
sg_idc_sbrs, sg_idc_sbrsc, sg_idc_brbs, sg_idc_brbc,sg_idc_reti: std_logic:= '0';
```

140
Appendix A (continued)

signal sg_alu_data_r_in, sg_alu_data_d_in,
  sg_alu_data_out: std_logic_vector(7 downto 0):= (others => '0');
signal sg_reg_rd_in, sg_reg_rd_out,
  sg_reg_rr out: std_logic_vector(7 downto 0):= (others => '0');
signal sg_reg_rd_addr,
  sg_reg_rr_addr: std_logic_vector(4 downto 0):= (others => '0');
signal sg_reg_h_out,
  sg_reg_z_out: std_logic_vector(15 downto 0):= (others => '0');
signal sg_reg_h_addr: std_logic_vector(2 downto 0):= (others => '0');
signal sg_reg_rd_wr, sg_post_inc, sg_pre_dec, sg_reg_h_wr: std_logic:= '0';
signal sg_reg_fl_in, sg_reg_out, sg_reg_fl_wr_en, sg_spl_out, sg_sph_out,
  sg_ramp_out: std_logic_vector(7 downto 0):= (others => '0');
signal sg_sp_ndown_up, sg_sp_en: std_logic:= '0';
signal sg_bit_num_r_io, sg_branch,
  sg_sreg_bit_num: std_logic_vector(2 downto 0):= (others => '0');
signal sg_bitpr_io_out, sg_bit_pr_sreg_out, sg_sreg_flags, sg_bld_op_out,
  sg_reg_file_rd_in: std_logic_vector(7 downto 0):= (others => '0');
signal sg_bit_test_op_out: std_logic:= '0';
signal sg_alu_c_flag_out, sg_alu_z_flag_out, sg_alu_n_flag_out,
  sg_alu_v_flag_out, sg_alu_s_flag_out, sg_alu_h_flag_out: std_logic:= '0';

begin
  main: component pm_fetch_dec port map
    (  
      -- EXTERNAL INTERFACES OF THE CORE
      clk => cp2,
      nrst => ireset,
      cpuwait => cpuwait,

      -- PROGRAM MEMORY PORTS
      pc => pc,
      inst => inst,

      -- I/O REGISTERS PORTS
      adr => sg_adr,
      iore => sg_iore,
      iowe => sg_iowe,

      -- DATA MEMORY PORTS
      ramadr => ramadr,
      ramre => ramre,
      ramwe => ramwe,
      dbusin => sg_dbusin,
    )
Appendix A (continued)

dbusout => sg_dbusout,

-- INTERRUPTS PORT
irqlines => irqlines,
irqack => irqack,
irqackad => irqackad,

-- END OF THE CORE INTERFACES

-- INTERFACES TO THE OTHER BLOCKS

-- INTERFACES TO THE ALU
alu_data_r_in => sg_alu_data_r_in,
alu_data_d_in => sg_alu_data_d_in,

-- OPERATION SIGNAL INPUTS
idc_add_out => sg_idc_add,
idc_adc_out => sg_idc_adc,
idc_adiw_out => sg_idc_adiw,
idc_sub_out => sg_idc_sub,
idc_subi_out => sg_idc_subi,
idc_sbc_out => sg_idc_sbc,
idc_sbcxi_out => sg_idc_sbcxi,
idc_sbiw_out => sg_idc_sbiw,
adiw_st_out => sg_adiw_st,
sbiw_st_out => sg_sbiw_st,
idc_and_out => sg_idc_and,
idc_andi_out => sg_idc_andi,
idc_or_out => sg_idc_or,
idc_ori_out => sg_idc_ori,
idc_eor_out => sg_idc_eor,
idc_com_out => sg_idc_com,
idc_neg_out => sg_idc_neg,
idc_inc_out => sg_idc_inc,
idc_dec_out => sg_idc_dec,
idc_cp_out => sg_idc_cp,
idc_cpc_out => sg_idc_cpc,
idc_cpi_out => sg_idc_cpi,
idc_cpse_out => sg_idc_cpse,
idc_lsr_out => sg_idc_lsr,
idc_ror_out => sg_idc_ror,
idc_asr_out => sg_idc_asr,
idc_swap_out => sg_idc_swap,
Appendix A (continued)

-- DATA OUTPUT
alu_data_out => sg_alu_data_out,

-- FLAGS OUTPUT
alu_c_flag_out => sg_alu_c_flag_out,
alu_z_flag_out => sg_alu_z_flag_out,
alu_n_flag_out => sg_alu_n_flag_out,
alu_v_flag_out => sg_alu_v_flag_out,
alu_s_flag_out => sg_alu_s_flag_out,
alu_h_flag_out => sg_alu_h_flag_out,

-- INTERFACES TO THE GENERAL PURPOSE REGISTER FILE
reg_rd_in => sg_reg_rd_in,
reg_rd_out => sg_reg_rd_out,
reg_rd_adr => sg_reg_rd_adr,
reg_rr_out => sg_reg_rr_out,
reg_rr_adr => sg_reg_rr_adr,
reg_rd_wr => sg_reg_rd_wr,
post_inc => sg_post_inc,
pre_dec => sg_pre_dec,
reg_h_wr => sg_reg_h_wr,
reg_h_out => sg_reg_h_out,
reg_h_adr => sg_reg_h_adr,
reg_z_out => sg_reg_z_out,

-- INTERFACES TO THE INPUT/OUTPUT REGISTER FILE
sreg_fl_in => sg_sreg_fl_in,
sreg_out => sg_sreg_out,
sreg_fl_wr_en => sg_sreg_fl_wr_en,
sp_out => sg_sp_out,
sph_out => sg_sp_out,
sp_ndown_up => sg_sp_ndown_up,
sp_en => sg_sp_en,
rampz_out => sg_rampz_out,

-- INTERFACES TO THE BIT PROCESSOR
bit_num_r_io => sg_bit_num_r_io,
between_io_out => sg_bitpr_io_out,
branch => sg_branch,
bit_pr_sreg_out => sg_bit_pr_sreg_out,
sreg_bit_num => sg_sreg_bit_num,
Appendix A (continued)

bld_op_out => sg_bld_op_out,
bit_test_op_out => sg_bit_test_op_out,

-- OPERATION SIGNALS INPUTS

-- INSTRUCTIONS AND STATES
idc_sbi_out => sg_idc_sbi,
sbi_st_out => sg_sbi_st,
idc_cbi_out => sg_idc_cbi,
cbi_st_out => sg_cbi_st,
idc_bld_out => sg_idc_bld,
idc_bst_out => sg_idc_bst,
idc_bset_out => sg_idc_bset,
idc_bclr_out => sg_idc_bclr,
idc_bclr_out => sg_idc_bclr,
idc_sbi_out => sg_idc_sbi,
idc_sbis_out => sg_idc_sbis,
idc_sbis_out => sg_idc_sbis,
idc_sbrs_out => sg_idc_sbrs,
idc_sbrs_out => sg_idc_sbrs,
idc_sbrs_out => sg_idc_sbrs,
idc_sbrs_out => sg_idc_sbrs,
idc_brc_out => sg_idc_brbc,
idc_brbc_out => sg_idc_brbc,
idc_brbc_out => sg_idc_brbc,
idc_reti_out => sg_idc_reti
);

general_purpose_register_file: component reg_file
generic map(ResetRegFile => TRUE)
port map
(
  reg_rd_in => sg_reg_rd_in,
  reg_rd_out => sg_reg_rd_out,
  reg_rd_adr => sg_reg_rd_adr,
  reg_rr_out => sg_reg_rr_out,
  reg_rr_adr => sg_reg_rr_adr,
  reg_rd_wr => sg_reg_rd_wr,
  post_inc => sg_post_inc,
  pre_dec => sg_pre_dec,
  reg_h_wr => sg_reg_h_wr,
  reg_h_out => sg_reg_h_out,
  reg_h_adr => sg_reg_h_adr,
  reg_z_out => sg_reg_z_out,
  clk => cp2,
nrst => ireset
);

bit_proc: component bit_processor port map
Appendix A (continued)

(  
  clk => cp2,  
nrst => ireset,  
  bit_num_r_io => sg_bit_num_r_io,  
dbusin => sg_dbusin,  
  bitpr_io_out => sg_bitpr_io_out,  
sreg_out => sg_sreg_out,  
  branch => sg_branch,  
  bit_pr_sreg_out => sg_bit_pr_sreg_out,  
sreg_bit_num => sg_sreg_bit_num,  
  bld_op_out => sg_bld_op_out,  
  reg_rd_out => sg_reg_rd_out,  
  bit_test_op_out => sg_bit_test_op_out,  
)

-- OPERATION SIGNALS INPUTS

-- INSTRUCTIONS AND STATES
  idc_sbi => sg_idc_sbi,  
  sbit => sg_sbit,  
  idc_cbi => sg_idc_cbi,  
  cbist => sg_cbist,  
  idc_bld => sg_idc_bld,  
  idc_bst => sg_idc_bst,  
  idc_bset => sg_idc_bset,  
  idc_bclr => sg_idc_bclr,  
  idc_sbic => sg_idc_sbic,  
  idc_sbis => sg_idc_sbis,  
  idc_sbrs => sg_idc_sbrs,  
  idc_sbrs => sg_idc_sbrs,  
  idc_brbc => sg_idc_brbc,  
  idc_reti => sg_idc_reti  
);

io_dec:component io_adr_dec port map  
  (  
    adr => sg_adr,  
    iore => sg_iore,  
    dbusin_int => sg_dbusin, -- LOCAL DATA BUS
      -- OUTPUT
    dbusin_ext => dbusin, -- EXTERNAL DATA BUS INPUT
    spl_out => sg_spl_out,  
    sph_out => sg_sph_out,  
    sreg_out => sg_sreg_out,  
  
145
Appendix A (continued)

    rampz_out => sg_rampz_out
    );

io_registers: component io_reg_file port map
    (    clk => cp2,
      nrst => irest,
      adr => sg_adr,
      iowe => sg_iowe,
      dbusout => sg_dbusout,
      sreg_fl_in => sg_sreg_fl_in,
      sreg_out => sg_sreg_out,
      sreg_fl_wr_en => sg_sreg_fl_wr_en,
      spl_out => sg_spl_out,
      sph_out => sg_sph_out,
      sp_ndown_up => sg_sp_ndown_up,
      sp_en => sg_sp_en,
      rampz_out => sg_rampz_out
    );

ALU: component alu_avr port map
    (    alu_data_r_in => sg_alu_data_r_in,
      alu_data_d_in => sg_alu_data_d_in,
      alu_c_flag_in => sg_sreg_out(0),
      alu_z_flag_in => sg_sreg_out(1),

      -- OPERATION SIGNALS INPUTS
      idc_add => sg_idc_add,
      idc_adc => sg_idc_adc,
      idc_adiw => sg_idc_adiw,
      idc_sub => sg_idc_sub,
      idc_subi => sg_idc_subi,
      idc_sbc => sg_idc_sbc,
      idc_sbcii => sg_idc_sbcii,
      idc_sbiw => sg_idc_sbiw,
      adiw_st => sg_adiw_st,
      sbiw_st => sg_sbiw_st,
      idc_and => sg_idc_and,
      idc_andi => sg_idc_andi,
      idc_or => sg_idc_or,
      idc_ori => sg_idc_ori,
      idc_eor => sg_idc_eor,
      idc_com => sg_idc_com,
Appendix A (continued)

```vhdl
idc_neg => sg_idc_neg,
idc_inc => sg_idc_inc,
idc_dec => sg_idc_dec,
idc_cp => sg_idc_cp,
idc_cpc => sg_idc_cpc,
idc_cpi => sg_idc_cpi,
idc_cpse => sg_idc_cpse,
idc_lsr => sg_idc_lsr,
idc_ror => sg_idc_ror,
idc_asr => sg_idc_asr,
idc_swap => sg_idc_swap,

-- DATA OUTPUT
alu_data_out => sg_alu_data_out,

-- FLAGS OUTPUT
alu_c_flag_out => sg_alu_c_flag_out,
alu_z_flag_out => sg_alu_z_flag_out,
alu_n_flag_out => sg_alu_n_flag_out,
alu_v_flag_out => sg_alu_v_flag_out,
alu_s_flag_out => sg_alu_s_flag_out,
alu_h_flag_out => sg_alu_h_flag_out);

adr <= sg_adr;
iowe <= sg_iowe;
iore <= sg_iore;
dbusout <= sg_dbusout;
end architecture struct;
```
Appendix B VHDL

Code for the Sensor Signal Adaptation Block

-- VHDL Code for the Signal Adaptation Block
-- Developed based on a template provided by Xilinx
-- Modified to work in the Signal Adaptation Block of the
-- Reconfigurable "Mote"
-- Top Level

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity TOP_LEVEL_TB is
end entity TOP_LEVEL_TB;

architecture BEHAVE of TOP_LEVEL_TB is
-- CONSTANT DECLARATION
constant HALFCLKPERIOD: time:= 200 nS;

-- Component Declaration
component TOP_LEVEL is
port(
    SP_CS0n: in std_logic;
    SP_CS1n: in std_logic;
    SP_WEn: in std_logic;
    SP_OEn: in std_logic;
    SP_D: inout std_logic_vector(15 downto 0);
    SP_A: in std_logic_vector(23 downto 1);
    SP_A_0: out std_logic;
    SPRING_RESETn: in std_logic;
    CHIP3_EN: out std_logic;
    A: out std_logic_vector(22 downto 0);
    D: inout std_logic_vector(15 downto 0);
    FLASH_CS0n: out std_logic;
    FLASH_WR_PROTECT: out std_logic;
    SRAM_CS1n: inout std_logic;
);
Appendix B (continued)

OEn: out std_logic;
RWn: inout std_logic;
IO13: out std_logic;
SRAM_UPPER_BYTEEn: out std_logic;
SRAM_LOW_BYTEEn: out std_logic;
--ADS7870 Signals
Cclk: in std_logic;
Selk: inout std_logic;
Din: out std_logic;
Dout: in std_logic;
AD_CHIP2_ENn: out std_logic;
RISE_FALL: out std_logic;
CONVERT: out std_logic;
OSC_CNTRL: out std_logic;
ADS_RESET: out std_logic;
BUSY: in std_logic;
BUSY_TEST: out std_logic;
LED1_SEL: out std_logic;
LED2_SEL: out std_logic;
LED3_SEL: out std_logic;
LED4_SEL: out std_logic;
);
end component TOP_LEVEL;

-- SIGNAL DECLARATIONS
signal SP_CS0n: std_logic;
signal SP_CS1n: std_logic;
signal SP_WEn: std_logic;
signal SP_OEn: std_logic;
signal SP_D: std_logic_vector(15 downto 0);
signal SP_A: std_logic_vector(23 downto 1);
signal SP_A_0 : std_logic;
signal CHIP3_EN: std_logic;
signal A: std_logic_vector(22 downto 0);
signal D: std_logic_vector(15 downto 0);
signal FLASH_CS0n: STD_LOGIC;
signal FLASH_WR_PROTECT: std_logic;
signal SRAM_CHIP1_ENn: std_logic;
signal OEn: std_logic;
signal RWn: std_logic;
signal SRAM_UPPER_BYTEEn: std_logic;
signal SRAM_LOW_BYTEEn: std_logic;
Appendix B (continued)

```vhdl
    signal IO13: std_logic;
signal Cclk: std_logic;
signal Sclk: std_logic;
signal Din: std_logic;
signal Dout: std_logic;
signal AD_CHIP2_ENn: std_logic;
signal RISE_FALL: std_logic;
signal CONVERT: std_logic;
signal OSC_CNTRL: std_logic;
signal ADS_RESET: std_logic;
signal BUSY: std_logic: = '0';
signal BUSY_TEST: std_logic;
signal led1_sel: std_logic;
signal led2_sel: std_logic;
signal led3_sel: std_logic;
signal led4_sel: std_logic;

begin
    -- Component Instantiation
    TOP_TEST: TOP_LEVEL
        port map(
            SP_CS0n => SP_CS0n,
            SP_CS1n => SP_CS1n,
            SP_WEn => SP_WEn,
            SP_OEn => SP_OEn,
            SP_D => SP_D,
            SP_A => SP_A,
            SP_A_0 => SP_A_0,
            CHIP3_EN => CHIP3_EN,
            A => A,
            D => D,
            FLASH_CS0n => FLASH_CS0n,
            FLASH_WR_PROTECT => FLASH_WR_PROTECT,
            SRAM_CS1n => SRAM_CHIP1_ENn,
            OEn => OEn,
            RWn => RWn,
            SRAM_UPPER_BYTEEn => SRAM_UPPER_BYTEEn,
            SRAM_LOW_BYTEEn => SRAM_LOW_BYTEEn,
            Celk => Celk,
            Sclk => Sclk,
            Din => Din,
            Dout => Dout,
        )
```
Appendix B (continued)

AD_CHIP2_ENn => AD_CHIP2_ENn,
RISE_FALL => RISE_FALL,
CONVERT => CONVERT,
OSC_CNTRL => OSC_CNTRL,
ADS_RESET => ADS_RESET,
BUSY => BUSY,
BUSY_TEST => BUSY_TEST,
IO13 => IO13,
led1_sel => led1_sel,
led2_sel => led2_sel,
led3_sel => led3_sel,
led4_sel => led4_sel
);

GEN_CLKS: process is
  begin
    cclk <= '0';
    wait for 15 uS;
    loop
      cclk <= '1';
      wait for HALFCLKPERIOD;
      cclk <= '0';
      wait for HALFCLKPERIOD;
    end loop;
  end process GEN_CLKS;

FLOW: process is
  begin
    dout <= '1';
    SP_A <= "00000000000000011111"
    SP_CS0n <= '1';
    SP_CS1n <= '1';
    SP_WEn <= '1';
    SP_OEn <= '1';
    wait for 100 us;
    SP_CS1n <= '0';
    SP_WEn <= '0';
    sp_d <= "1111111111111111"
    wait for 100 us;
    SP_WEn <= '1';
    wait for 100 us;
  end process FLOW;
Appendix B (continued)

SP_WEn <= '0';
sp_d<="0000000000000000"
wait for 100 us;

SP_WEn <= '1';
wait;
end process FLOW;

end architecture BEHAVE;