This is to certify that the
thesis entitled

APPLICATION SPECIFIC PROGRAMMABLE PROCESSOR
FOR SENSOR BASED NETWORKS

presented by

KARTIK VAIDYANATHAN

has been accepted towards fulfillment
of the requirements for the

MS degree in
Electrical and Computer Engineering

Major Professor's Signature

5/19/03

Date

APPLICATION SPECIFIC PROGRAMMABLE PROCESSOR FOR SENSOR BASED NETWORKS

By

Kartik Vaidyanathan

A THESIS

Submitted to
Michigan State University
In partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Electrical and Computer Engineering

2003

ABSTRACT

APPLICATION SPECIFIC PROGRAMMABLE PROCESSOR FOR SENSOR BASED NETWORKS

By

Kartik Vaidyanathan

The growing complexity of smart sensor systems has increased demands on its control electronics to improve the overall efficiency of the system. General-purpose controllers, though robust and cost-effective, are not optimized to meet the range of requirements for smart sensor systems. A thorough understanding of the architectural challenges of sensor-based systems such as wireless environmental monitoring sensors and biomedical sensors will greatly benefit the design of their control electronics. An application specific programmable processor for sensor-based systems has been designed. The prominent features of the processor are a 16-bit RISC core, supported by a flexible instruction set, 512-byte on-chip sensor data memory and a power management unit that implements a low power sleep mode. A special port has been designed, to interface with the network modified version of the IEEE 1451.2 standard serial bus for smart transducer interface of sensors and actuators. A top-down design flow methodology has been adopted using synthesis and automatic layout tools that explore designs for low power optimizations. Results that verify the operation of the processor in a simulated sensor environment are presented.

This thesis is dedicated to my family.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Andrew Mason for laying the foundation of this research and providing valuable support through the completion of this thesis. I would also like to thank my committee members, Dr. Anthony Wojcik, and Dr. Erik Goodman for their guidance and constructive comments in the revision and correction of this thesis. Finally, I would like to thank my friends and colleagues from the AMSaC Research Laboratory for their support and help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

In the rapidly growing field of microelectronics, a promising field has carved a niche for itself under the name of microsensors. Simply put, microsensors deals with the design and fabrication of sensors in the domain of silicon-based microelectronics. Microsensors are devices that convert analog parameters from the world around us into electrical signals that can be read, calibrated and stored for a wide variety of applications. Semiconductor sensors (another term used to describe microsensors) are sensors fabricated using integrated circuit (IC) fabrication techniques and inherit from ICs features such as small size, low power consumption and low cost through batch fabrication. These sensors are further classified based on the type of data measured, some of the categories being acoustic, magnetic, electrical and chemical sensors. Microsensors are being used in a wide array of applications ranging from sensors in automobiles that trigger the release of airbags to biomedical applications that monitor vital signs inside the human body. Any application that has to be programmed to react to some action without human intervention would have to use a sensor of sorts. The rapid growth of the semiconductor industry in the last twenty years that symbolized the Integrated Circuit revolution has laid the foundation for future growth in the field of microsensors [14]. The scope of research in this field arises from the fact that all microsensors are application specific and the list of applications is growing steadily.

1.1 Smart Sensors

Recent research in the area of sensors has led to what are known as "smart sensors", which are more than just dumb nodes that perform analog to digital conversion. These nodes are packed with more power and intelligence that not only collect sensor data but also make logical decisions based on the environment. These sensors are capable of processing signals to produce outputs that can be read directly by microcontrollers and computers. These smart sensors are capable of being networked, perform sensor readout control and perform calibration and compensation of sensor readings. They distribute the load on the central processor that would normally process the raw data coming in from an analog sensor by sharing some of the burden and performing some of this processing at the sensor node. As a result, the size, power consumption and cost of the sensor node increases.

Typically these features of the smart sensor are implemented on an ASIC and the sensor itself is placed off chip to account for environmental factors [1]. The growth of smart sensor systems has been strongly aided by developments in the semiconductor industry; the increasing growth of logic and memory technologies in microelectronics has resulted in, highly reliable and performance-driven systems [2]. The driving factors of the semiconductor industry - size, power and speed, have been coupled with traditional sensors to bring about what is known as the smart sensor revolution [15]. Based on the IEEE 1451.2 standard [4]; this microelectronic interface circuitry is referred to as Smart Transducer Interface Modules (STIMs). Another feature of the smart sensor is the capability to process sensor data and to share the data across a network with other sensor nodes. This networking capability of the sensor further increases the complexity of the

sensor node. These networked sensors are implemented on a sensor bus, based on the IEEE 1451.2 standard for sensor bus communication. A microcontroller or processor is needed to host these smart sensor nodes and to arbitrate the mutual functioning of the sensor interface modules. These microcontrollers, also known as Network Capable Application Processors (NCAP), interface with the STIMs using a standard that has been defined in the industry. The basic architecture of the smart sensor node and the network capable application processor is shown in Figure 1.1.

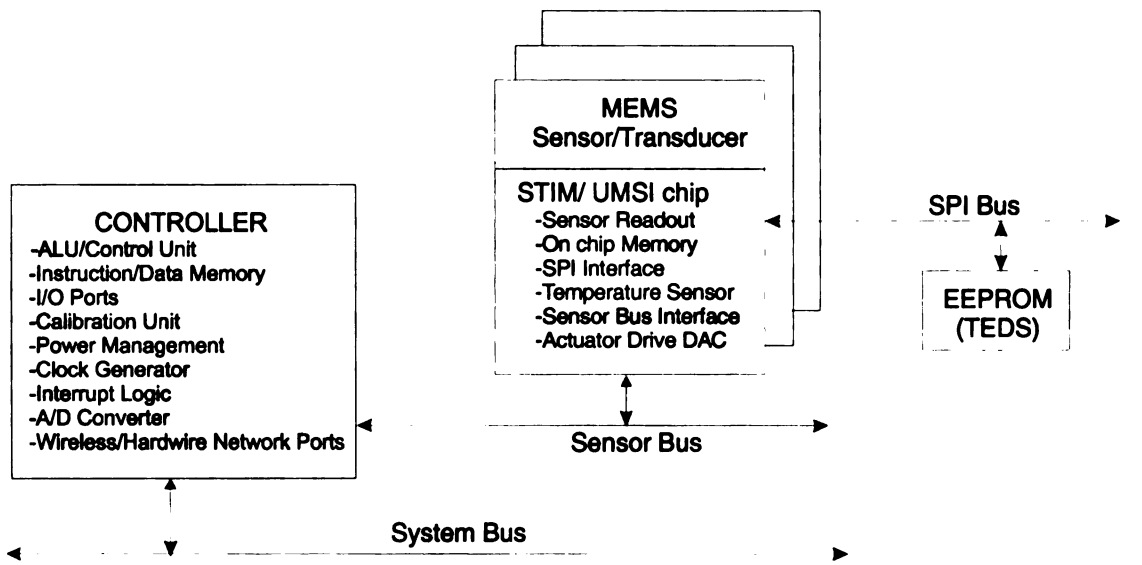


Figure 1.1: Architecture of the Smart Sensor Node with Interface Circuitry

The sensor is fabricated using Micro-Electro-Mechanical Systems (MEMS) technologies that are based on the integration of sensors, actuators, mechanical elements and electrical circuitry on the same silicon substrate. The fabrication process in MEMS-based systems could implement any integrated circuit process (CMOS, BiCMOS or Bipolar) wherein components, are fabricated by selectively etching parts of the silicon

wafer and adding new structures to form highly complex electromechanical systems. MEMS are fabricated using IC fabrication techniques that make them highly reliable, low cost and extremely high density or small size. A smart sensor STIM has been designed fabricated and tested by the AMSAC research group at Michigan State University under the name of Universal Micro Sensor Interface (UMSI) chip. This chip interfaces with a wide variety of sensors with capacitive, voltage, digital and resistive outputs [3]. In addition, UMSI provides a digital I/O port for communication with the microcontroller, an SPI interface that connects the chip to the Transducer Electronic Data Sheet (TEDS), SRAM that interfaces with the analog circuitry and the multi range/resolution temperature sensor that provides temperature compensation [3]. The sensor bus is connected to the controller using the IEEE 1451.2 [4] standard that is used by all vendors making possible the networking of the STIMs independent of the vendor. The IEEE 1451.2 also defines the Transducer Electronic Data Sheet that is stored on the sensor module to define sensor specific information. In the defined architecture, the microcontroller (or NCAP) will serve as a processing host to an array of smart sensors receiving raw data from each of these nodes.

The microcontroller used to implement the sensor bus interface performs a wide variety of operations. The need for more application specific processing is increasing steadily as applications demand higher performance and lower power consumption. The microcontroller performs operations such as sensor read, to read information from the TEDS, and sensor write, where certain error correcting codes may be written to the TEDS. A bus communication protocol is implemented that has Ack / Nacks, clock and data signals for data communication. The microcontroller receives the raw data from the

sensor and may process this data. Currently, research is underway to develop a dedicated calibration engine that processes the raw data to calibrate it to a desired range compensating cross-parameter sensitivities. While testing the UMSI chip, a general-purpose microcontroller was used that had a core-processing unit, I/O ports to implement the sensor bus and a power management unit. As smart sensors continue to find newer applications, the demands on the controller call for a shift from the use of general-purpose processors to application specific programmable processors (ASPP). These application specific processors, in addition to having core-processing capabilities, interface with specific integrated circuit blocks implemented to meet demands of smart sensors. In this thesis a sensor network processor is presented that meets the processing requirements of the host controller and interfaces with other blocks that constitute the architecture of smart sensor based systems.

Sensor based systems are used in many different walks of life – healthcare (diagnostic and prosthetics), weather and environmental monitoring systems, reconnaissance for defense and military purposes and automation of industrial processes [5]. Many of these applications have demands on different controller parameters ranging from low power, high speed, high processing power and size of the chips designed. This sensor network microcontroller has been designed to meet low power and small size constraints of microsensors used in portable wireless biomedical applications. Several integrated circuit design techniques were studied and implemented in the design to meet the desired requirements of the controller.

1.2 Background Work and Motivation

Research in the area of microsensors and sensor-based systems is being widely pursued. The microprocessor industry has been following Moore's Law, from the first Intel 4004 to the present Pentium processors, which drive the semiconductor industry to call for a continuous influx of cutting edge technology. Microcontrollers, an application specific, control-oriented version of the microprocessor, are probably the most widely used integrated circuits in the semiconductor industry. Almost all ASICs designed call for an integrated processing core or might require a separate microcontroller to supervise, control and clock the operation of the chip. Smart sensor based systems have been designed using many of the general-purpose microcontrollers available in the market from such vendors as Texas Instruments, Motorola and Hitachi, to name a few. Most of these controllers meet the generic requirements of sensor-based systems and even provide very reliable performance and low power operation.

The MS8 low power microcontroller was designed at the University of Michigan and is used as a generic interface for multiple sensors [6]. The MS8 is a mixed signal microcontroller that has voltage, capacitive and current interfaces, signal amplifier and an analog to digital converter. The processor core consists of an 8-bit controller, 40-bit accumulator and 16-bit hardware multiplier. The MS8 has a 512b boot ROM, 4Kb program memory and a 512 byte RAM for data storage. To save power, the MS8 has a programmable clock manager capable of clocking different blocks at different frequencies and a watchdog timer that powers up the processor from sleep mode. Other features of the core include a universal synchronous/asynchronous receiver transmitter

(USART), parallel I/O ports, multifunction timer and capability to interface with off chip memory.

In the AMSAC research lab at Michigan State University, the Texas Instruments low power generic controller MSP430 was used to test the UMSI chip. The MSP430 is a 16-bit RISC controller with 27 core instructions, a 3.3 MHz clock and an extremely low power consumption of 4.2nW per instruction. It wakes up in normal mode from the power saving sleep mode within 6μs [7]. This core is based on the von Neumann architecture where the address and memory bus are shared. It has clock distribution circuitry to generate a low frequency auxiliary clock besides the normal master clock. The peripheral circuitry includes a hardware multiplier, USART, 16/8 bit timers, high performance ADC and supply voltage supervisors (SVS). CALMRisc is a low power 8-bit microcontroller with a coprocessor interface designed by Samsung semiconductor and KAIST [10]. This controller uses many novel techniques for low power VLSI design and later versions of this RISC core are being used in cell phones where power consumption is a key parameter.

The goal of this research project was to design an application specific programmable processor that meets the requirements of sensor based systems in terms of power, speed and area. Emphasis was placed on implementing various architectural and circuit level design techniques for a low power processor. The processor interfaces with other design blocks such as the calibration engine, which will be used to process the data received from the sensor. The core has a defined I/O port that interfaces with the sensor bus (based on the IEEE 1451.2 standard) [4] between the UMSI chip and the controller. The implementation of the chip was done based on a top down design flow. The initial

specification was done in Verilog, followed by synthesis using Ambit Buildgates and then place and route using Silicon Ensemble. The tools were setup to implement this project and a flow developed that catered to implementing the processor with maximum efficiency. The process of this design flow and the effect of the tools on the design of the processor will be discussed.

1.3 Thesis Overview

Chapter 2 of the thesis explains the architecture of the processor. The basic block diagram, instruction set and specifications of the processor are described. Chapter 3 introduces the design flow for implementing the processor. The verification process of the processor is discussed followed by results describing the operation of the processor. Chapter 4 describes the synthesis and layout process in the development of the processor. The various simulation results and chip layouts are also presented. Chapter 5 summarizes the research presented in the thesis and describes possible future work to further improve the performance of the processor.

2. System Architecture

The microprocessor industry caters to a wide array of applications ranging from high-end general purpose processors that are used in modern day PCs to application specific processors that are used in automobiles, robots, consumer electronics and satellites. The factors taken into consideration before the design of a processor are the speed, area or size of the processor and power consumed by the processor. All design parameters of a processor can be derived from these factors. Supercomputers, desktop machines and space stations need high-speed processors that can process large volumes of data in minimum time. Laptops, handheld computers or cell phones are restricted by their portability and hence need to make efficient use of power. Microsensors or nano-robots are so small that they have to obtain processing capability in minimum area. However, all these factors are interdependent and are important in microsensor systems. An extremely high-speed processor will have an affect on the size and power consumed by the processor and vice versa. Hence, different architectural techniques and circuit design methodologies are used based on the requirement of the processor.

In this chapter, section 2.1 describes the factors taken into consideration while making the initial specification of the processor. Sections 2.2 – 2.8 describe the processor core, instruction set, memory, sleep mode, I/O ports and other features of the processor.

2.1 System Specification and Design

The processor was designed to work in conjunction with an array of sensors having certain characteristics that define the features of the processor. These sensors are used to measure analog data from the environment that are then quantified into measurements of pressure, temperature, humidity, etc. The role of the processor is to

monitor these sensor nodes and to arbitrate the processing of the data received from these nodes. Taking into account this application of the processor, several parameters or features were considered in the initial specification of the processor.

The first decision made was to use a conventional Reduced Instruction Set Computer (RISC) load store processor over a Complex Instruction Set Computer (CISC) processor. RISC cores are based on transferring the burden to software and keeping the complexity of hardware to a minimum. The time taken to execute a RISC instruction will be a fixed one-clock cycle irrespective of the type of instruction versus CISC systems that encode multiple instructions to form a complex instruction that performs a specific task. The Application Specific Programmable Processor (ASPP) designed is not going to run in a computationally intensive environment, making the choice of a RISC architecture ideal [17]. To improve the performance of the processor the core was divided into a three-stage pipeline.

The key factor taken into consideration was the low power operating requirements of the controller. A typical application of this processor will be in wireless systems - for example, an environmental monitoring system that runs off a battery and calls for minimum power wastage. Several techniques can be used to exploit low power operation such as architectural methods, circuit level techniques and special EDA tools that help save power through the design process. Besides this, sensor-based systems perform sensing sporadically, making a power-saving standby mode very practical. The processor supports gated clocks and a sleep mode where the processor moves into a standby mode, when no processing is required. The processor wakes up on an interrupt from the sensor front end and starts processing the data received. EDA tools used in the design process

provide the capability of designing the processor for low power operation and include methods to estimate the power early in the design to facilitate design changes based on requirements. The implementation and usage of these tools will be discussed in chapter 4 of the thesis.

The Harvard architecture, with independent program and data memory, was chosen over the von Neumann architecture. This allowed simultaneous memory access from both the instruction and data memory thereby increasing the memory bandwidth. It also simplified memory access by keeping the instruction and data memory separate. This feature helped improved the speed of the processor and removed the need for dual ported memory.

2.2 Architecture

The architecture of the processor is based on a RISC structure that implements an instruction set tailored to sensor applications. The instructions are either register-register or register-memory instructions, thereby reducing the bandwidth of instructions between the CPU core and the memory. The following sections of the processor briefly describe each architectural component (shown in Figure 2.1).

2.2.1 Block Diagram of the Processor

2.2.1.1 Central Processing Unit (CPU)

The CPU has a 16-bit arithmetic logic unit and a register file that consists of 8 registers for data. The CPU has additional registers such as the program status register and condition code register that are used to set flags. To control the flow of execution, the CPU uses a 10-bit program counter and 10-bit memory address register. The CPU is

pipelined into three stages to improve the throughput of the processor and control the sequence of operations in the processor.

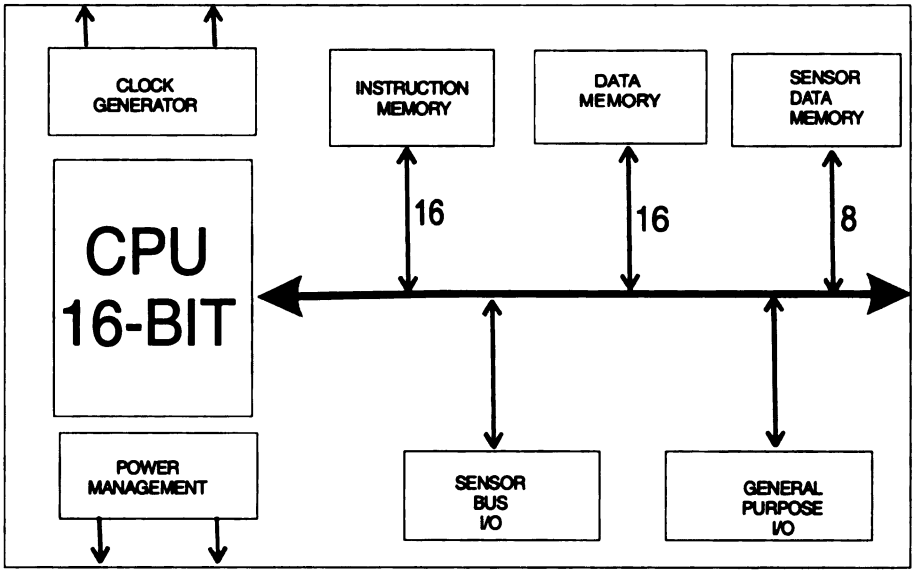


Figure 2.1: Block Diagram of the Sensor Network Processor

2.2.1.2 Clock Divider

The clock generator is assumed to be an off chip block that provides a constant frequency to the clock divider circuitry. The clock divider uses the off-chip clock signal to generate clocks of different frequencies that are used to control different stages of the pipeline. Two clock frequencies are used; one to clock each stage of the pipeline and the other to clock blocks within a particular stage of the pipeline.

2.2.1.3 Memory

The processor consists of three separate memory blocks. The conventional instruction and data memories are independent; the instruction memory is synthesized to a 2Kbyte (1024 x 16bit) static RAM and the data memory a 512 byte (256 x 16bit) RAM block. In addition a separate 256 byte (256 x 8bit) sensor data memory has been designed to store sensor data information and interface with a calibration unit.

2.2.1.4 Input/Output Ports

The I/O ports are divided into general purpose I/O that are used to interface with blocks such as the Direct Memory Access (DMA) controller and a specific sensor bus port designed to implement the Intra Module Multielement (IM^2) bus. The IM^2 bus is based on a modified version of the IEEE 1451.2 standard and has been designed specifically for the microsensor to communicate with a host controller during operation [4]. The sensor bus port implements certain complex instructions that are used by the processor to communicate with the sensor nodes that it controls. These complex instructions contrary to the other RISC instructions of the processor take more than one clock cycle to complete.

2.2.1.5 Power Management and Sleep Unit

The sleep unit in the processor is used to shut down the processor to save power. The basic application of the processor is to receive requests from a sensor node and then to process data received from the sensor. In the sleep mode the processor clock is turned off and the processor monitors the sensor ports for an interrupt to turn the processor back on. After receiving an interrupt the processor shifts back into normal mode and starts executing instructions.

2.2.2 States of the Processor

The processor has three states of operation as shown in Figure 2.2

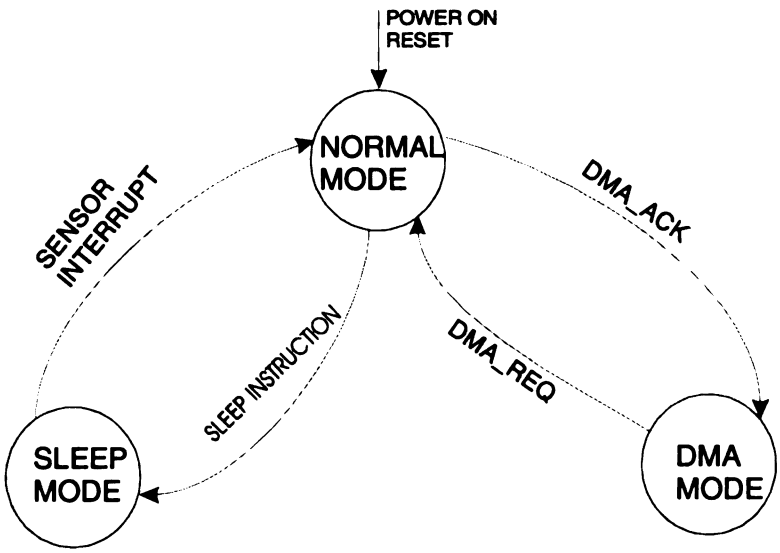


Figure 2.2: States of the Processor

1. Normal Mode

   In the normal mode of operation the processor executes the instruction sequence stored in the program memory.

2. Sleep Mode

   In the sleep mode the processor clock is shut off, basically reducing the power consumed by the processor to static power dissipation of the processor. The processor monitors the sensor bus, waiting for an interrupt from the sensor to resume operation.

3. DMA mode

   In this mode the processor is set up for Direct Memory Access (DMA) operations. The data and address are loaded onto the I/O ports of the processor; after receiving a dma_ack, the DMA process is initiated.

2.2.3 Pipeline Stages of the Processor

As shown in Figure 2.3 the processor data path is divided into a three-stage pipeline to control the flow of data during execution.

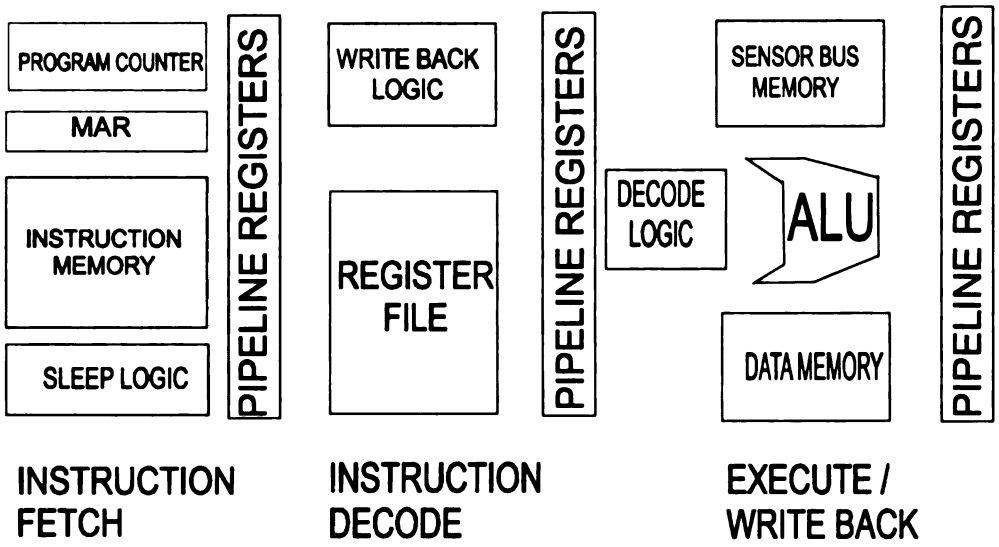


Figure 2.3: Pipeline Stages of the Processor

The three stages of the processor are

1. Instruction Fetch

   After power on reset (POR) the program counter is reset to start from instruction memory location 0x000h. There are two clocks that control the flow of data. The internal clock1 controls the flow of data within a stage, and clock2 controls the flow of data from one pipe stage to the next. On the positive edge of clock1, data is loaded into the program counter and then on the next clock, data moves to the memory address register. The program counter is incremented to hold the address of the next instruction of execution. The instruction memory is decoded (using a 10:1024 decoder) and the instruction is loaded onto a buffer register. On the negative edge of clock2, data is loaded into a 16-bit pipeline register.

2. Instruction Decode

   The instruction is loaded from the pipeline register and is broken down

into individual components - namely the opcode, register address, data memory address or immediate data. The second stage of the pipeline consists of the register file that has 8, 16-bit registers. The register addresses are used to decode the register file and load the data onto pipeline registers.

3. Execute / Write back

   The final stage of the pipeline consists of an arithmetic logic unit that executes the instruction and the necessary write back logic to store the result. All inputs to the ALU are fed in from the previous stage irrespective of the type of instruction and the data is then redirected from here to the final destination. Based on the type of instruction the data may be written back to the original register, stored in data memory or written to the sensor data memory. Two-byte instructions that require two clock cycles are also handled by this stage of the pipeline.

All the stages of the pipeline are of equal length i.e. they take the same clock cycle length to complete execution. The length of this clock cycle is limited by the slowest stage namely the Execute/Write back stage of the pipeline.

2.3 Processor Core

The core of the processor consists of a group of registers, the arithmetic logic unit and control logic that directs the flow of data through the data path. The instruction set architecture forms a major part of the processor core and defines the capabilities of the processor. The next few sections describe in detail the CPU registers, different addressing modes and instruction set architecture. A broad overview of the instruction set architecture will be described in this chapter and a description with opcodes is provided in Appendix A.

2.3.1 CPU Registers

Figure 2.4 shows the different registers in the core of the processor. The CPU registers consist of the data registers in the register file, special registers (condition code register and program status word) and pipeline registers. The register file provides the working registers for the processor and all register-register, register-immediate, register-memory instructions are executed using these registers. The condition code register is used to set carry, zero, negative, equal, greater than and less than flags based on the output from the processor. The program status word is used to indicate the state of the processor. The states are based on the sensor interrupts generated on the sensor bus port of the processor. This port holds the status of NINT, NSDET and NACK signals that will be discussed in detail when discussing the IM^2 bus. In addition to these signals there is also a flag for the DMA operation that is set by the DMA controller to request the start of the DMA process. The program counter and memory address register are 10 bit registers that are reset during the power on reset.

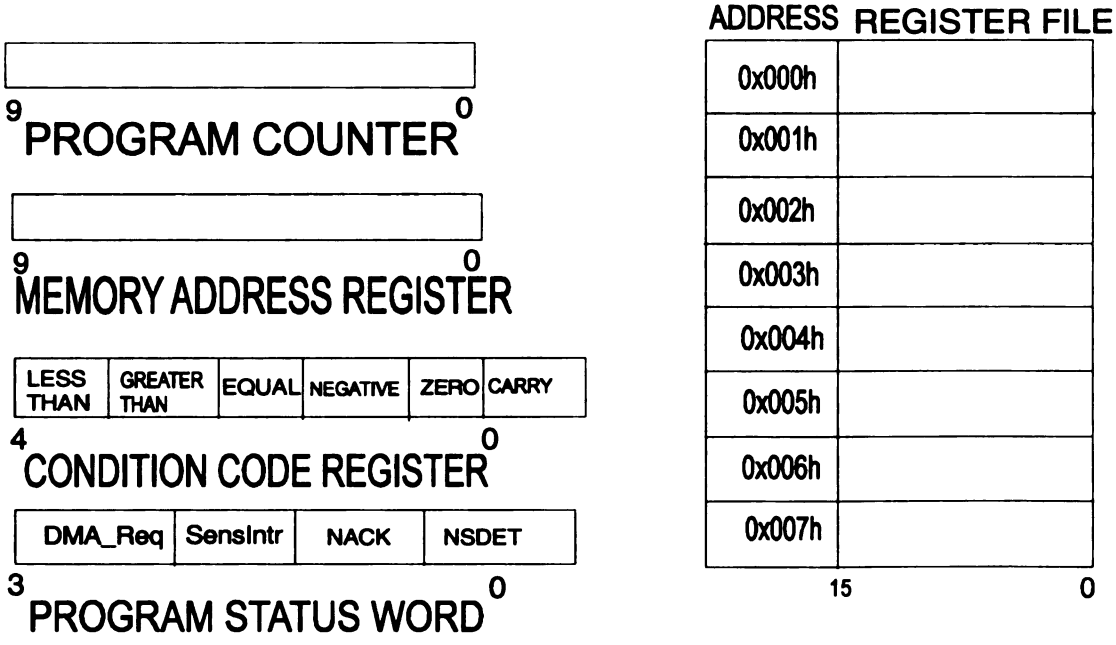


Figure 2.4: CPU Registers

2.3.2 Addressing Modes

The three addressing modes supported are register mode, indirect mode and immediate mode. Tables 2.1 – 2.3 describe each addressing mode with an example instruction.

Table 2.1: Register Addressing Mode.

	Assembler Code	**Opcode**
	Mov Reg1 Reg2	0x6350h
Length	One word instruction	
Description	Moves the content of Reg2 to Reg1. Both registers are a part of the register file. The 3 bits [10:8] are used to address register 1 and the 3 bits [7:5] are used to address register 2.	
Example	Mov (R3) (R5)	

Before

R2	0x00ffh
R3	**0x13ach**
R4	0x02d3h
R5	**0xfa34h**
R6	0x1234h

After

R2	0x00ffh
R3	**0xfa34h**
R4	0x02d3h
R5	**0xfa34h**
R6	0x1234h

Table 2.2: Indirect Addressing Mode.

	Assembler Code	**Opcode**
	Load Reg1 &(dmem)	0x0B0fh
Length	One word instruction	

Table 2.2 (cont'd).

Description	Loads the contents of Reg1 with the data stored in the data memory address location. PC is incremented and continues to execute instructions normally.
Example	Load (R3) &(0x0f)

Before

Register File

R2	0x00ffh
R3	**0x0001h**
R4	0x02d3h
R5	0xfa34h
R6	0x1234h

Data Memory

0x0d	0xaaf1h
0x0e	0x12fah
0x0f	**0x000ah**
0x1f	0x0001h
0x2f	0x00ddh

After

Register File

R2	0x00ffh
R3	**0x000ah**
R4	0x02d3h
R5	0xfa34h
R6	0x1234h

Data Memory

0x0d	0xaaf1h
0x0e	0x12fah
0x0f	**0x000ah**
0x1f	0x0001h
0x2f	0x00ddh

Table 2.3: Immediate Addressing Mode

	Assembler Code	**Opcode**
	OR Reg1 #Immediate	0x5B00h 0x00deh
Length	Two Word Instruction	

Table 2.3 (cont'd).

Description	The contents of Reg1 are ORed with the following 16-bit immediate data. The result is stored in Reg1. PC is incremented and continues to execute instructions normally. These types of instructions are two byte instructions and take two clock cycles.
Example	OR (R3), &(0x0f)

Before
Register File

R2	0x00ffh
R3	**0x13ach**
R4	0x02d3h
R5	0xfa34h
R6	0x1234h

Before
Instruction Memory

0x0d	0xaaf1h
0x0e	0x12fah
0x0f	**0x00deh**
0x1f	0x0001h
0x2f	0x00ddh

After
Register File

R2	0x00ffh
R3	**0x13feh**
R4	0x02d3h
R5	0xfa34h
R6	0x1234h

After
Instruction Memory

0x0d	0xaaf1h
0x0e	0x12fah
0x0f	**0x00deh**
0x1f	0x0001h
0x2f	0x00ddh

2.3.3 Instruction Set Architecture

The major types of instructions implemented in the processor are Arithmetic

- Load/Store
- Logical
- Control/Branch
- Rotate/Shift
- Sensor Bus

The instruction set architecture was developed based on the instruction set of the Texas Instruments processor (MSP430) [7].

2.3.3.1 Arithmetic Instructions.

The arithmetic instructions implemented in the processor with their operations are listed in Table 2.4. The arithmetic operations can support either the register-addressing mode or the immediate-addressing mode. The condition code register is used to reflect the conditions generated by the result of the arithmetic operation. The zero, negative, carry, equals, greater than and less than flags are set in the condition code register. The result of the arithmetic operation is stored in the target register and hence results in the change of data of the operands; the exception to this being the compare instruction (CMP) where the two operands are compared and the flags are set based on the comparison.

Table 2.4: Arithmetic Instructions

Instruction	Description	Operation
Add RegX, RegY	Add contents of RegX and RegY; Result stored in RegX; Flags Set: C, Z, N	RegX ← RegX + RegY
Sub RegX, RegY	Subtract contents of RegY from RegX; Result stored in RegX; Flags Set: C, Z, N	RegX ← RegX – RegY
Add RegX, #data	Add contents of RegX to immediate value; Result stored in RegX; Flags Set: C, Z, N	RegX ← RegX + #data
Addc RegX, RegY	Add with carry RegX and RegY; Result stored in RegX; Flags Set: C, Z, N	RegX ← RegX + RegY + C
Cmp RegX, RegY	Compares the contents of RegX and RegY; No result stored; Flags Set: C, Z, N, GT, LT	Temp ← RegX – RegY
Inc RegX	Increments the contents of RegX by 1; Result stored in RegX; Flags Set: C, Z, N	RegX ← RegX + 1
Dec RegX	Decrements the contents of RegX by 1; Result stored in RegY; Flags Set: C, Z, N	RegX ← RegX – 1

2.3.3.2 Load/Store Instructions

The load – store instructions implemented in the processor are listed in Table 2.5. As described previously the architecture of the processor is based on a load-store RISC structure with very limited direct memory operations. Data has to be loaded

onto the register file and then stored back in the data memory using the load store operation. These instructions are implemented using the register-addressing, indirect-addressing and the immediate addressing mode. The load instruction is a two-byte instruction where the next word holds the immediate data.

Table 2.5: Load/Store Instructions.

Instruction	Description	Operation
LDA RegX, #data	Load the Immediate data into RegX; Flags Set: None	RegX ← #data
STA RegX, &mem	Stores the contents of RegX in mem location specified by Inst [7:0]. Flags Set: None	Dmem[&addr] ← RegX
Mov RegX, RegY	Move contents of RegY to RegX. Flags Set: None	RegX ← RegY
Mov RegX, #data	Moves the immediate data to RegX; Flags Set: None	RegX ← #data

2.3.3.3 Logical Instructions

The logical instructions implemented in the processor are listed in Table 2.6. Logical instructions are implemented using the register-addressing and immediate-addressing mode. All results of logical operations are stored in the source operand. The logical instructions implemented using immediate addressing mode are all two-word instructions.

Table 2.6: Logical Instructions.

Instruction	Description	Operation
Inv RegX	Invert contents of RegX; Result stored in RegX	RegX ← ~(RegX)
AND RegX, RegY	AND contents of RegX and RegY; Result stored in RegX	RegX ← RegX & RegY
OR RegX, RegY	OR contents of RegX and RegY; Result stored in RegX	RegX ← RegX | RegY
XOR RegX, RegY	XOR contents of RegX and RegY; Result stored in RegX	RegX ← RegX ^ RegY
AND RegX, #data	AND contents of RegX with immediate data; Result stored in RegX	RegX ← RegX & #data

Table 2.6 (cont'd).

OR RegX, #data	OR contents of RegX with immediate data; Result stored in RegX	RegX ← RegX | #data
XOR RegX, #data	XOR contents of RegX with immediate data; Result stored in RegX	RegX ← RegX ^ #data
TWO RegX	Two's complement of RegX; Result stored in RegX	RegX ← ~(RegX) + 1

2.3.3.4 Control/Branch Instructions

The control / branch instructions implemented in the processor are listed in Table 2.7. The control instructions are implemented using the indirect addressing mode and are all single byte instructions. The BRA EN and BRA GL are actually four separate instructions. Based on the bit set in the instruction sequence, the branch instruction will check the equal, negative, greater than and less than flags. The branch instructions are evaluated in the Execute stage of the pipeline thereby generating a two-cycle delay in evaluation of the branch. Hence there is a delay-slot immediately following the branch instruction that can be utilized by the programmer to schedule instructions.

Table 2.7: Control/Branch Instructions

Instruction	Description	Operation
BRA #Imm	Branch to the location specified in the 8-bit address encoded in the instruction; Change PC to new value.	PC ← &BrAddr
BRA EN #Imm	Branch if EQ or N flag are set to address encoded in the instruction; Change PC to new value	PC ← &BrAddr
BRA GL #Imm	Branch if GT or LT flags are set to address encoded in the instruction; Change PC to the new value.	PC ← &BrAddr

2.3.3.5 Rotate/Shift Instructions

The rotate / shift instructions implemented in the processor are listed in Table 2.8. There are four rotate instructions that are all implemented in the register-addressing mode.

Table 2.8: Rotate / Shift Instructions

Instruction	Description	Operation
RRA RegX	Rotate right the RegX through no. of bits specified; Store Result in RegX	RegX ← Rt(RegX)
RLA RegX	Rotate left the RegX through no. of bits specified; Store Result in RegX	RegX ← Rt(RegX)
RRC RegX	Rotate right the RegX through the Carry flag in the condition code register; Flags Set: C	RegX ← Rt(RegX)
RLC RegX	Rotate left the RegX through the Carry flag in the condition code register; Flags Set: C	RegX ← Rt (RegX)

2.3.3.6 Sensor Bus Instructions

The sensor bus instructions were designed to implement the sensor bus communication protocol called the Intramodule Multielement Microsystem (IM^2) bus. These instructions are discussed in section 2.6 along with detailed description of their functions.

2.4 Memory

The memory of the processor as in most integrated chips occupies the majority of the area in the chip. This processor has been designed with three individual blocks of memory each designed to serve a specific purpose. The entire processor has been designed in RTL code, which will be discussed in the next chapter. The RTL code for the memory was written to synthesize the memory as a volatile Static RAM circuit with the necessary read write circuitry. The following sections describe the three memory blocks in the processor.

2.4.1 Program Memory

Figure 2.5 shows the block diagram of the program memory. The memory cells are word aligned and are addressed by 10-bit memory address register. The instruction sequence is stored in the program memory and is decoded in the first stage of

the pipeline. The total number of cells addressed through this decoding process is 1024; i.e. the memory has 1024 cells of 2-bytes each. The program memory also supports the Direct Memory Access (DMA) from an off chip DMA controller through the I/O ports of the processor. To start a DMA process the controller sends a DMA request to the processor. Upon receiving the request, the processor sends a DMA acknowledge signal and then the data transfer is initiated. The DMA controller sends a read/write signal based on the request through the dma_rw line. Then it sends the data and address through the DMA ports of the processor.

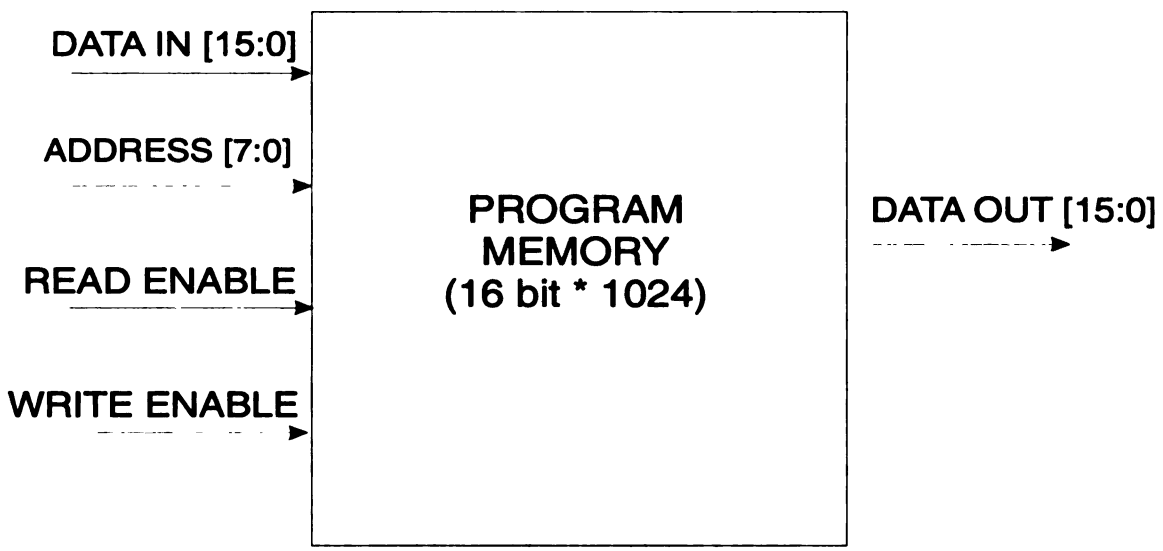


Figure 2.5: Program Memory

2.4.2 Data Memory

Figure 2.6 shows a block level diagram of the data memory. The data memory in the processor is used for temporary data storage during program execution. The data memory is 512 bytes in size and is addressed by an 8-bit address register. The memory cells are word aligned i.e. each memory cell is 1 word in length. There are a total of 256

cells of 2-bytes each. The data memory is used in the write-back stage of the pipeline and during the load/store instructions.

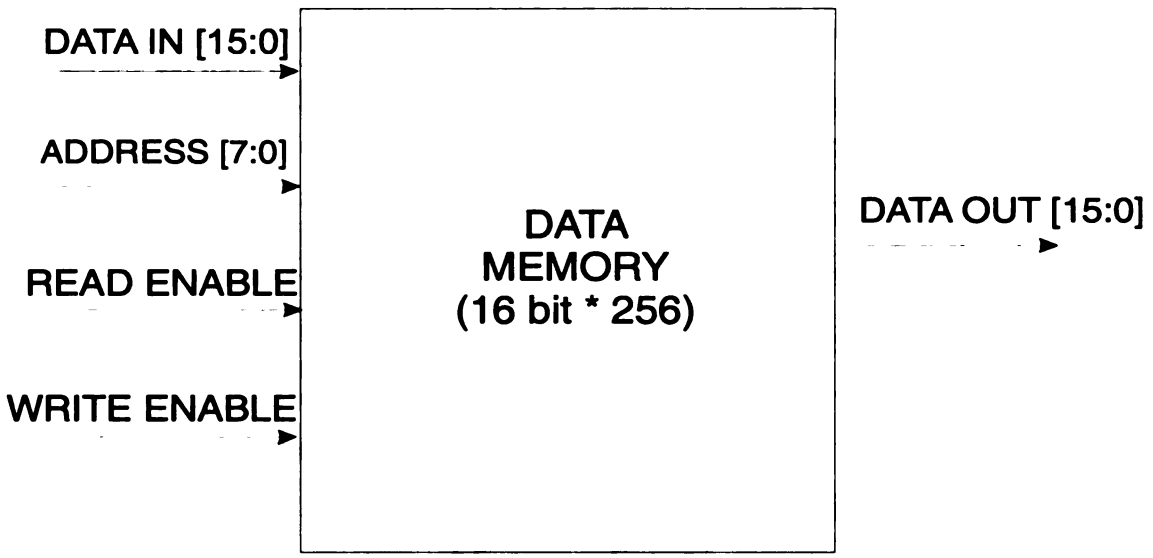


Figure 2.6: Data Memory

2.4.3 Sensor Data Memory

The sensor data memory was specifically implemented to interface with a calibration engine that is being designed to process data received from the sensor nodes. The processor serves as a host controller that receives the data over the sensor bus and stores the data in the sensor data memory. This memory block is dual ported to enable the calibration engine to read data from the memory. The size of this memory block is 256 bytes and is byte aligned, i.e., each cell in the memory unit addressed is 1-byte long. Furthermore the memory will be segmented into two areas; one to store the permanent sensor TEDS information (loaded when a sensor node is added to the network) and the second memory segment to store sensor data readings.

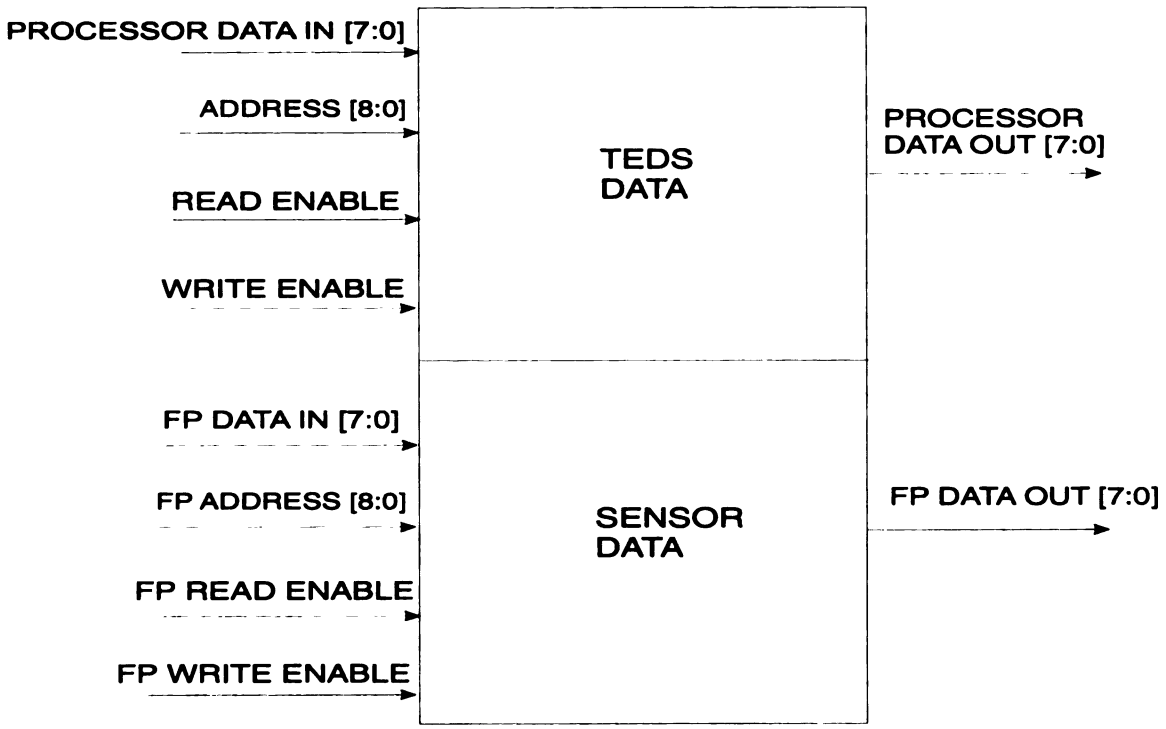


Figure 2.7: Sensor Data Memory

2.5 Sleep Mode

CMOS circuits attribute their power consumption to two types of power dissipations - static power and dynamic power dissipation. Static power dissipation is the power consumed by a circuit when there is no change in the inputs. It is usually the power dissipated due to the leakage currents in the transistors. The dynamic power consumption, on the other hand, is the power dissipated when the inputs are changing and is characterized by three components.

$$P_{dynamic} \propto f \text{ (frequency, supply voltage, capacitance in the circuit)}$$

The frequency corresponds to the frequency at which the inputs of the circuit are changing. In the case of the processor, the frequency of the input is dependent on the clock frequency. Thus by shutting off the clock the frequency component of the dynamic power dissipation function is brought to zero, reducing the dynamic power to zero.

The sleep mode was designed to move the processor into a low-power sleep state when the processor did not need to run. The typical application of the processor in controlling the sensor nodes over the sensor bus (IM^2) is very sporadic. It can be characterized as a burst of data processing requirements interspersed with dormant time when the processor has to monitor the bus waiting for sensor interrupts. To save power during these dormant periods, the sleep mode was implemented as an instruction so that the programmer can preprogram the processor to go into sleep mode after sensor data processing. Once the processor goes into sleep mode the clock is shut down, hence turning off the processor.

Once in the sleep mode the processor monitors the sensor bus for an interrupt signal from any of the sensor nodes. When the processor receives an interrupt it moves back into normal mode and the clock is turned on. The processor resumes operation from the next instruction that could branch back to the top of the program to repeat the sensor bus implementation.

2.6 Input / Output Ports

There are two basic I/O ports in the processor - the general-purpose I/O ports and the sensor bus I/O port.

2.6.1 General-Purpose I/O ports

The general-purpose I/O port consists of a 16-bit data line and a 10-bit address line. The processor supports direct memory access (DMA) operations and has signals such as dma_req, dma_ack and dma_rw on the I/O port. These ports are used to perform DMA operations and can read and write data to the memory block. The DMA operation

is also flagged in the program status register to indicate that a DMA operation is in progress.

2.6.2 Sensor Bus Input/Output port

A special port was implemented in the processor to communicate with the STIM (Smart Transducer Interface Circuit) also called the UMSI (Universal Microsensor Interface Chip) implemented in [10]. The processor designed will serve as a host controller that arbitrates the operation of a network of sensor nodes. The IM^2 (Intra-Module Multielement) bus is used to communicate between the processor and any sensor node. The IM^2 bus is based on the IEEE 1451.2 standard [11] extended to meet certain requirements of the sensor interface communication. The IM^2 bus supports both digital and analog data and has the controllable power supply lines that facilitate normally off operation. The IM^2 bus was developed in the AMSAC laboratory at Michigan State University described in [12].

The descriptions of the IM^2 signals are listed in Table 2.9. This table lists each signal, basic function and the driver. All the signals except the supply voltage and the controllable power supply are implemented in the sensor bus on the processor. The normal sequence of operations starts with an interrupt from the sensor node by pulling the NINT line low. After receiving this interrupts the processor, when ready, pulls the NIOE line low, which signals the beginning of the data transfer. The processor then sends the data over the DIN line to the sensor node. The data sent over this line is a series of instructions to control the sensor node and perform readout. The NTRIG is used to send a trigger to the UMSI chip to trigger the shift-in process and control the counter of the

temperature sensor on the UMSI chip. The DOUT line is used to send sensor data back to the processor where it can be calibrated and processed.

Table 2.9: IM^2 Bus Signals [12]

Signal	Description	Driver
DIN	Address and data transmitted from microcontroller to interface module	Processor
DOUT	Data transmitted from interface module to microcontroller	Sensor Node
DCLK	Positive-going edge latches data on DIN and DOUT	Processor
NIOE	Signals that data transport is active and delimits data framing	Processor
NTRIG	Performs triggering function	Processor
NACK	Trigger acknowledge and data transport acknowledge	Sensor Node
NINT	Used by interface module to request service from microcontroller	Sensor Node
NSDET	Used by microcontroller to detect the presence of new interface module	Sensor Node
Power	Normal 3-V power supply	
Vswitch	Controllable power supply. It will be at low when system in sleep mode.	

The signals that are driven by the processor are DCLK, DIN, NIOE and NTRIG. This communication is a serial communication protocol where 8 bit data is sent serially over the DIN bus to the UMSI chip. A separate processor instruction has been implemented to transfer data over the DIN bus. The processor loads the specified data and serially transmits the data over the bus. Another instruction has been implemented to pull the NTRIG and NIOE line high or low. Finally an instruction has been implemented to receive and store the data sent from the sensor node to the processor and store it in the sensor data memory. Table 2.10 lists the sensor bus signals implemented in the processor with a brief description of each instruction.

Table 2.10: Sensor Bus Instructions

Instruction	Description
Snd #data	Sends the data over the DIN line of the sensor bus; 1 byte of data is sent over 8 clock cycles on the DCLK line;
Rcx #data	Receives data over the DOUT line of the sensor bus; 1 byte of data is received over 8 clock cycles on the DCLK line;
Str #addr	Received data is stored in #address of the sensor data memory;
Pull #sig	Pull the NIOE/NTRIG lines high/low

The Snd and Rcx instruction take two clock cycles to complete, so there is a stall in the pipeline that will float through after any of these instructions are executed. The Str instruction has to be executed immediately after the Rcx to store the received data in the sensor data memory. The Pull instruction can chooses between NIOE and NTRIG line by setting the bits [10:8] of the instruction and chooses between pulling high and low by setting the bit [0] of the instruction.

2.7 Clock Module

The clock fed to the processor is passed through a clock divider circuit to generate two clocks of different frequencies. The clock with lower frequency is used to clock each stage of the pipeline while the higher frequency clock is used to clock blocks within a particular stage of the pipeline. A simple clock divider circuit based on JK flip-flops was implemented for this purpose. Figure 2.8 shows the schematic of this clock dividing circuitry.

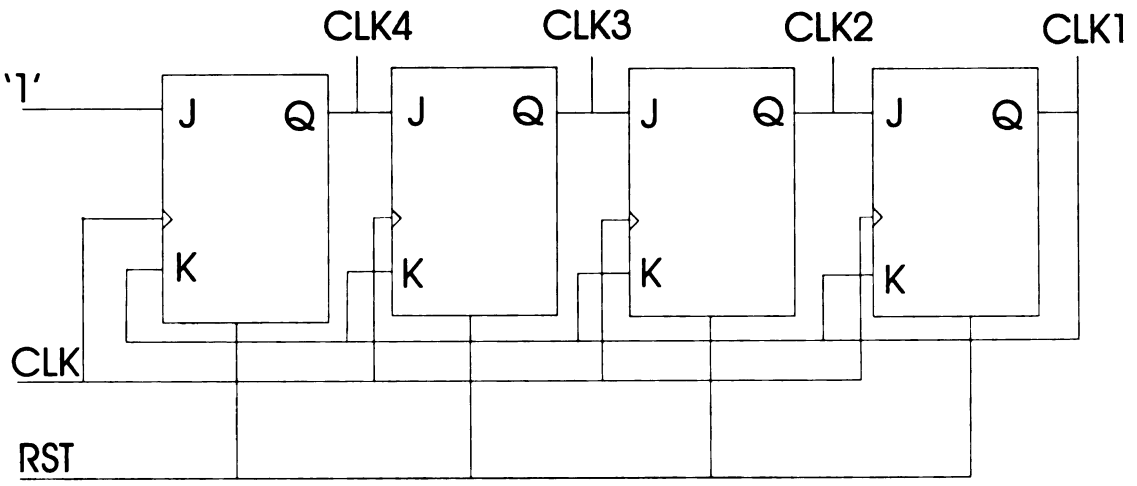


Figure 2.8: Clock Divider Circuitry

3. Design Flow, Verification and Results

A wide variety of CAD tools were used to implement the design of the processor. The number of transistors on modern-day processors is touching the one billion mark, calling for complex CAD tools to support such designs. The design approach chosen also plays a role in the performance of the chip designed. The initial part of this chapter describes the design flow, followed by a description of the tools used for verification. The last section describes the results of the verification process followed by an example for sensor node control using the designed processor.

3.1 Design Flow

The design flow implemented in Integrated Circuit design and fabrication is shown in Figure 3.1. This typical approach is most widely used both in the industry and in academia. There are basically two approaches one could adopt based on the size of the design, the granularity of parameters to be controlled and the time to market. They are the top-down and bottom-up design flow.

The first step in the design flow involves developing a system idea, in this case the design of an application specific programmable processor for sensor network data processing. The idea has to be verified at the system level to check if it meets the requirements, i.e., does it meet requirements of speed, data storage, programmability and interfacing capabilities. Then the designer has to choose between the top-down (standard cell) and bottom-up (full custom) approach.

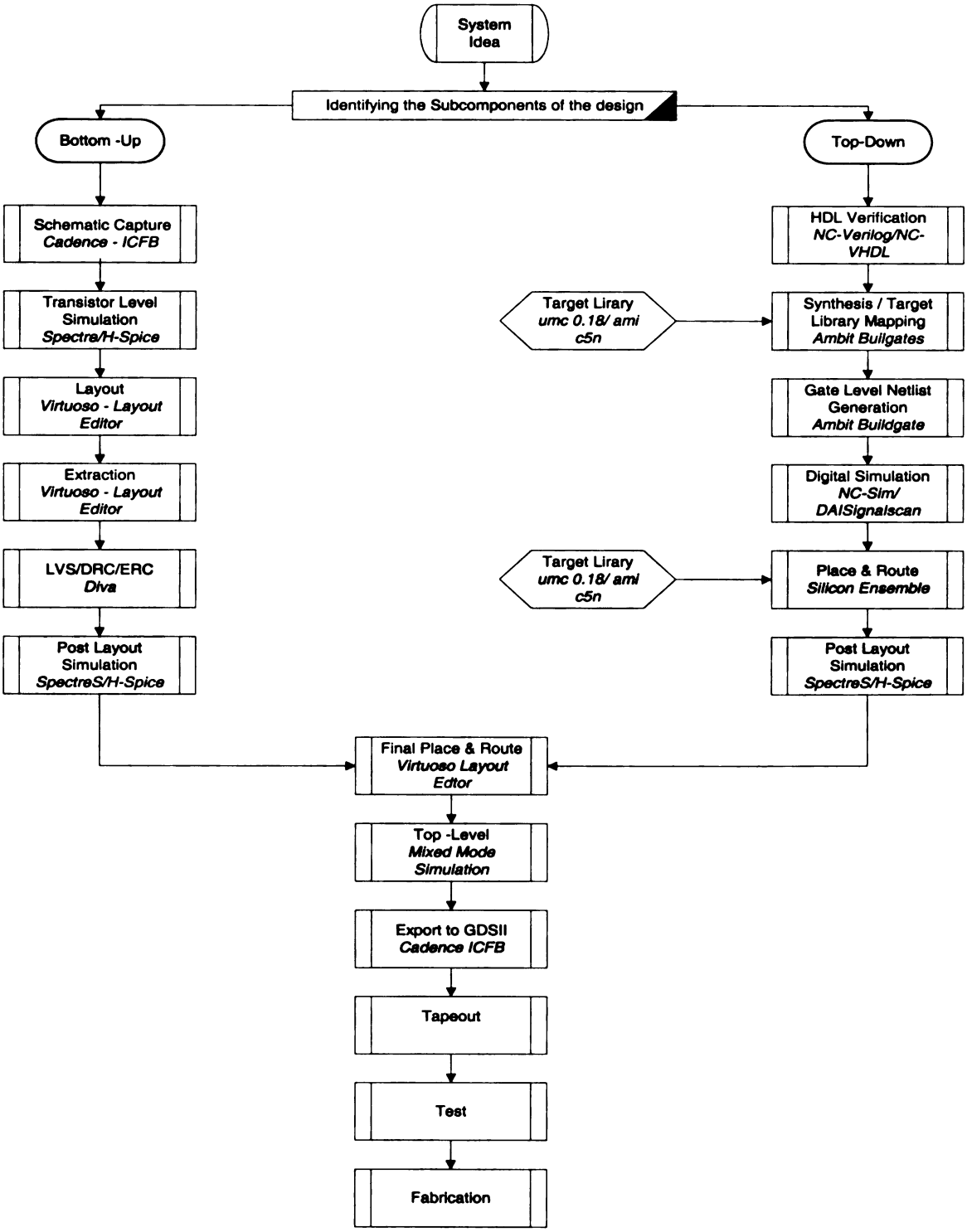


Fig 3.1: VLSI Design Flow

A full custom approach is chosen for the following reasons –

- Relatively small design size.
- Requirement for tailor made blocks that could not be synthesized by standard cells.
- To meet specific design requirements such as speed, power and size.

The first step involves designing the schematic for the circuit and verifying the functionality using transistor level simulation (spice models) for a particular target library such as AMIC5N (3 metal process) or TSMC 0.25μm (5 metal process). The layout of the schematic is generated and extracted to include the parasitic parameters. The extracted design is passed through a layout versus schematic check where a one-to-one correspondence is checked between the layout and the schematic to make sure that the nets and devices match. The post layout simulation is performed to verify the functionality of the layout and to obtain timing and power characteristics.

The standard cell approach or top-down design flow (adopted in this thesis) starts with the RTL (Register Transfer Level) specification of the design in a hardware description language such as Verilog or VHDL (Very High Speed Integrated Circuits Hardware Description Language). This description is synthesized using a synthesis tool such as Ambit Buildgates (from Cadence Design Systems) or Synopsis Design Compiler. A target library is provided to the synthesis tool to map the design to a particular technology such as AMIC5N, TSMC 0.25μm and UMC 0.18μm. The synthesis process results in a gate level netlist mapped to the target design library. Post synthesis simulation can be done to verify the functionality of the processor. The gate level netlist and the library files are the inputs to an automatic place and route layout tool that generates the

layout of the design. Silicon Ensemble from Cadence Design Systems was used for this process. In designing the processor Ambit Buildgates and a library developed by the VTVT group at Virginia Polytechnic Institute and State University were used [8].

The output of the layout tool can be merged with any blocks designed by the full custom method using a conventional layout editor. The target library has to be attached to this tool as well and will specify information such as the layer map table (defines each layer in the layout) and the spice models for the transistors. The design imported into the conventional full custom layout tool is then checked for design rule check (DRC) errors and layout versus schematic (LVS) errors. The final simulation is then performed to validate the functionality of the design known as the top-level verification. The design is then exported to a GDSII format that is a universal format accepted by all foundries and then sent for fabrication. The fabricated chip is then tested using either an automatic tester or manual-test equipment

3.2 Design Verification

Verification corresponds to the first stage of the top-down design flow. The initial specifications are used to write HDL code to implement a design. This process includes the functional simulation, which verifies the functionality of the RTL specification. This verification is not an exhaustive (fault-tolerant) testing process of the design but tests its functionality for a sequence of instructions specified in a testbench. The next few sections briefly describe the tools used for the verification process of the designed processor.

3.2.1 NC Verilog

NC Verilog developed by Cadence Design Systems is a Verilog digital logic simulator based on the Interleaved Native Compiled Code Architecture (INCA) designed

by Cadence Design Systems. The NC Verilog compiler was used to parse code written in Verilog, which is then fed to an elaborator similar to the linking process in normal programs [9]. The elaborator generates a single executable code stream that can be fed to a simulator. NC Verilog is compliant with the IEEE 1364 standard described in the IEEE standard Hardware Description Language based on Verilog HDL.

3.2.2 Signalscan

Signalscan is a powerful waveform-viewing tool developed by Cadence Design Systems to view waveforms and analyze simulation data generated by the design. Signalscan is used in the SimVison analysis environment to either view the simulation as it is generated or from a database. Signalscan generates SST2 database of the simulation or can convert a VCD (Value Change Dump) file into a SST2 design [9]. A VCD file records the changes in the signals specified in the testbench of the design that can be used to obtain characteristics about the input data. This VCD file will be used in the synthesis tool for power estimation. Additionally Signalscan has many features that help a user easily analyze the generated waveform, such as markers and radix converters (permits viewing results in hexadecimal, decimal and binary format).

3.3 Results

This section describes results to verify the operation of the processor. The digital waveforms were obtained using Signalscan waveform viewer.

3.3.1 Instruction Sequence 1

Figure 3.2 shows the first instruction sequence verifying the functionality of *Invert* (opcode 0), *Add* (opcode 3), *And* (opcode 5), *Xor* Immediate (opcode 11). The *Invert, Add, And* operations use the register addressing mode and the *XOR* is

implemented in the immediate addressing mode. Data1 [15:0] and Data2 [15:0] hold the two operands (only Data1 is used for immediate addressing mode) and Dataout [15:0] holds the result of the operation. Writebck_sig is used to indicate a write back to the register file, while the imm_out signal is used to indicate an immediate operation where the successive word holds the immediate data. The ConditionReg sets any flags resulting from the operation. In the XOR instruction (opcode 11) the operands are 0x000Ah and 0xEE00h resulting in 0xEE0Ah as the result.

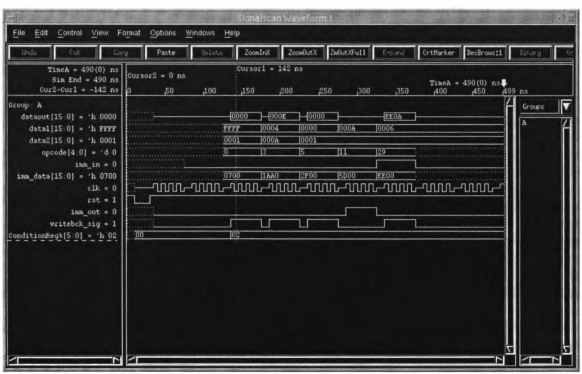


Figure 3.2: Simulations of Instruction Sequence 1

3.3.2 Instruction Sequence 2

Figure 3.3 shows the second instruction sequence verifying the functionality of *Two's complement* (opcode 14), *Add with carry* (opcode 15), *Rotate left through accumulator* (opcode 19) and *Rotate right specified number of bits* (opcode 17). The signals are the same as those described in the previous section. In the Add with carry

instruction the flag is set in the ConditionReg and is used in the rotate through carry instruction. There is a bubble in the pipeline with no output as shown between outputs 0x000Dh and 0x0001h. This occurs because of the two-byte instruction *Rotate right specified number of bits* (opcode 17) that has immediate data to wait for, leading to a delay in the output of the pipeline.

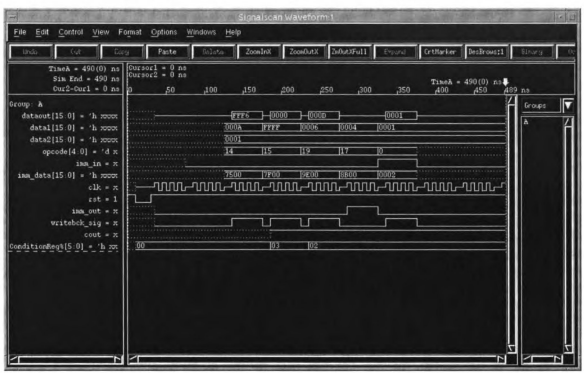


Figure 3.3: Simulations of Instruction Sequence 2

3.3.3 Instruction Sequence 3

This instruction sequence is a combination of the previous two sequences with a branch instruction inserted. The new opcode sequence of the combination is (0,3,5,**21**,11,29,14,15,19,17) with 21 being the opcode for the branch immediate instruction and a branch to opcode 19. So as shown in Figure 3.4, the opcode sequence after 21 goes to 19. Opcode 11 with its immediate data is executed between the branch

because the branch instruction is resolved in the execute stage of the pipeline leading to a two clock cycle delay.

Figure 3.4: Simulations of Instruction Sequence 3

3.3.4 Sensor Bus Instruction Sequence

The sensor bus implementation was described in section 2.6.3 along with a description of the special instructions designed. This section shows results verifying the functionality of those instructions and describes a sequence of instructions that could be used in a network controlling an array of sensors. Figure 3.5 shows the implementation of the *sensor bus write* (opcode 0x1Dh), *sensor bus read* (opcode 0x1Ch), *store sensor data* (opcode 0x1Bh) and *pull NIOE / NTRIG signal* (opcode 0x1Eh). The instruction sequence, shown in Table 3.1, starts with pulling NIOE low and then sends data on the sensor bus. The data sent on the bus are the chip ID of the UMSI chip, instruction opcode, address and data.

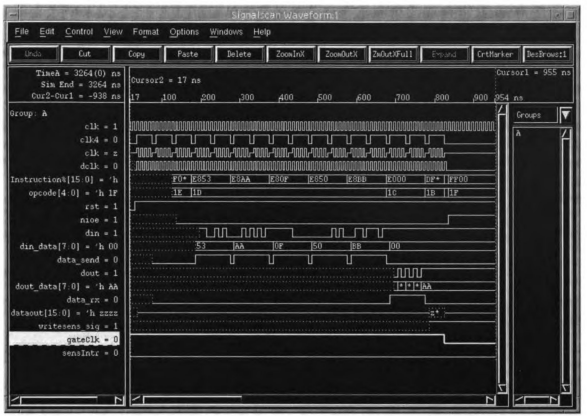


Figure 3.5: Sensor Bus Instruction Sequence 1

Table 3.1: Description of Sensor Bus Instruction Sequence 1

Instruction	Opcode	Description
0xF000h	0x1Eh	Pull NIOE signal on bus low
0xE853h	0x1Dh	Send Data 0x0101b (UMSI chip ID) and 0x0011b (Write memory instruction to UMSI chip) on Din port.
0xE8AAh	0x1Dh	Send Data 0x10101010b (Address to be written to UMSI chip) on Din port.
0xE80Fh	0x1Dh	Send Data 0x00001111b (Data to be written to UMSI chip) on Din port.
0xE850	0x1Dh	Send Data 0x0101b (UMSI chip ID) and 0x0000b (Read memory instruction to UMSI chip) on Din port.
0xE8BBh	0x1Dh	Send Data 0x10111011b (Address of Memory location to be read on UMSI chip) on Din port.
0xE000h	0x1Ch	Receive Data on DOUT port of processor
0xDF01h	0x1Bh	Store Received Data from DOUT port on Sensor Memory at address location (0x01h)
0xFF00h	0x1Fh	Put the processor in Sleep mode (power save mode)

In the sequence described, the processor first identifies the chip ID of the UMSI chip it is talking with and then instructs it to write to memory. The processor then sends the instruction to read data from the memory of the UMSI chip. The next instruction is to receive the read data received from DOUT port of the UMSI chip and store it in the sensor data memory. The processor then goes to sleep, thereby moving into low power sleep mode as shown when the gateClk signal is pulled low to turn off the clock. The signal Din and din_data show the data sent on the din port and dout_data shows the data received on the Dout port.

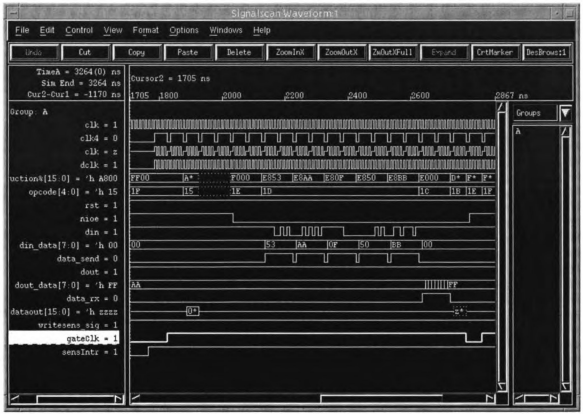


Figure 3.6: Sensor Bus Instruction Sequence 2

Figure 3.6 describes the instruction sequence after the processor receives the interrupt from the UMSI chip. This interrupt wakes up the processor from sleep mode as shown when sensIntr signal is pulled high. This in turn pulls the gateClk signal high

turning on the processor clock and executing the first instruction, which is a branch immediate instruction. Then control shifts back to the top of the code and repeats the entire program. A detailed description of all sensor bus instruction and the IM^2 bus is given in section 2.6.2.

4. Synthesis, Place and Route Results

This chapter deals with the implementation of the processor after system level specification and logic verification. The next two steps to implement the design of the processor are *synthesis* (the HDL code is converted to a gate level netlist) and *place and route* (the gate level netlist is used to generate the layout of the chip). The tools used to implement these two steps were studied and a design flow developed to use them at Michigan State University. Finally the results obtained at the end of each step are described.

4.1 Synthesis

Synthesis is the process of converting a Register-Transfer Level (RTL) description to a gate-level netlist. This mapping converts all the logic developed in the hardware description language to a netlist that has a one-to-one correspondence to a set of logic cells described in a library.

4.1.1 Need for Synthesis

The architectural description of any system specifies the set of inputs and expected outputs from the system without delving into the details of the circuit implementation of the system. Architectural descriptions are specified as a combination of behavioral, structural and logical descriptions that define a system. Synthesis is used to map these descriptions into a physical entity that will actually be used to implement the system. A synthesis tool converts the description into a netlist of combinational gates and registers that will implement the specified architecture. This netlist should meet the requirements of the system functionally and performance, such as speed (timing), power

and area. Figure 4.1 shows a simple example of synthesis of a behavioral description to a synthesized netlist. The behavioral description performs the addition of two 16-bit numbers; the synthesized netlist has to map this addition to an appropriate 16-bit ripple carry adder. It could also map, for example, to a carry lookahead adder or a carry save adder depending on the timing and area requirements specified by the designer. The specification of the adder to be synthesized is made in the Verilog code written by the designer. The synthesis tool has the capability of generating the adder based on this specification. Besides the physical implementation, the synthesis tool also performs various optimizations such as removal of redundant logic, exploiting don't care conditions, detecting unused states and making state assignments.

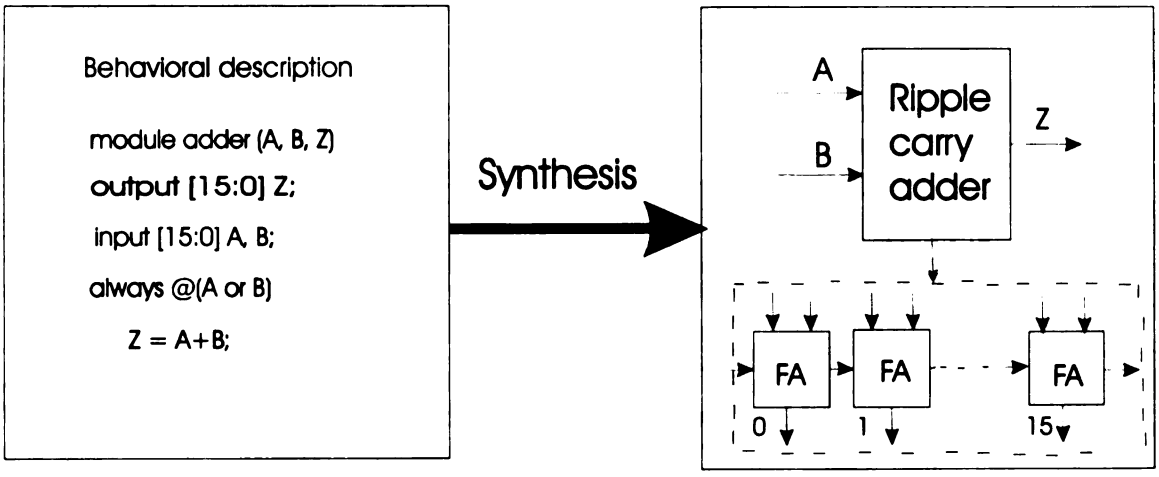


Figure 4.1: Synthesis of a 16-bit Adder

4.1.2. Tools and Requirements

Ambit Buildgates from Cadence Design Systems was used as the synthesis tool for this thesis. This industry standard tool supports rapid synthesis of multimillion gates with very high efficiency.

Features [9]

- Buildgates Extreme version of Ambit supports physically knowledgeable synthesis of chips by integrating datapath synthesis, static timing analysis and low power synthesis.
- Buildgates performs automatic partition of the datapath and control logic in a design when fed in as one single piece of code.
- Supports Verilog, VHDL and mixed language synthesis
- Performs operator merging, critical path optimization and removal of redundancy to improve the overall efficiency of the design.
- Performs RTL power optimizations using accurate timing and power information from the library files. This helps make power estimations early in the design that facilitates design changes to meet power and speed requirements.
- Performs clock-gating and sleep mode logic optimization by automatically inserting the necessary logic to further reduce power consumption.
- Integrates with place and route tools to complete the design flow.

The necessary files required to perform synthesis using Ambit Buildgates are

- Verilog Code: The RTL level description of the design to be synthesized
- TLF File: Timing Library Format file provided by the library vendor. In this thesis the VT library based on the TSMC 0.25um process [8] was used.
- TCF File: Toggle Count Format file contains information about the switching activity of the nets throughout the system. The TCF file is generated from a Value Change Dump (VCD) file during the logic simulation of the design.

4.1.3 Design Flow and Results

Figure 4.2 describes the design flow to synthesize a netlist. This section will describe each step in the design flow with the command to execute the instruction.

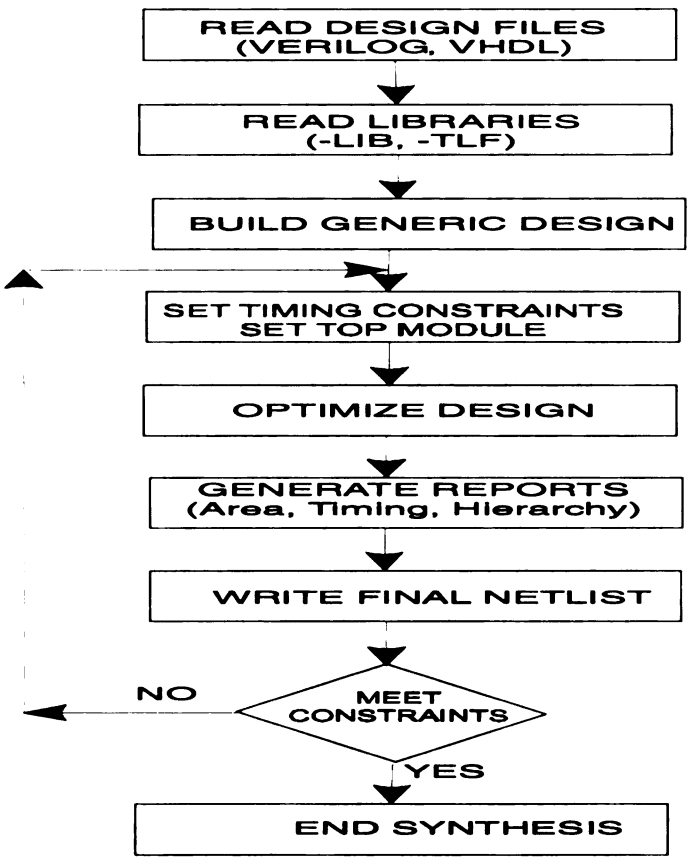


Figure 4.2: Synthesis Design Flow

Read the Library and Design Files

The first two steps in the synthesis design flow are to read the design and the library files. The design files used are usually RTL hardware language descriptions used to design a chip. The design files used were the Verilog description of the processor described in the previous section. The design files are read using the commands:

read_verilog designfile.v

read_vhdl designfile.vhd

The design files could also be of EDIF format - Exchange Design Interchange Format that is mainly used to exchange gate level designs between EDA tools. The library files used can be either the Cadence Timing Library Format (TLF) file or a Synopsys .lib format file. In the VT library the .lib file was provided and a syn2tlf converter was used to convert a .lib file to a .tlf file:

syn2tlf synopsys.lib -output cadencetlf.tlf

The files are read in using either of the commands:

read_tlf cadencelibraryfile.tlf

read_lib synopsyslibraryfile.tlf

Build a Generic Netlist

The design read in is mapped to a technology independent hierarchical gate level netlist based on generic ATL (Ambit Technology Library) and XATL (extended ATL) logic components.

do_build_generic

Set Constraints on the Design

In this step timing constraints can be defined on the design by setting an ideal clock for the design. The top-level module is also defined using the ***set_top_timing_module*** filename command.

Optimize

The optimization is the final step of the synthesis process. The *do_optimize* command starts the optimization process by invoking a series of *do_xform* commands. In the optimization process the tool defines the structure, removes redundancy, resizes cells, adds buffers and fixes design rule violations. ***do_optimize***

Generate Reports

In this step the tool generates reports that provide information about the design. The tool can generate reports about timing, area, library information, hierarchy of the design and design rules. The synthesis reports for the area and hierarchy of the design for the processor are listed in section B.1 of Appendix B.

Check Constraints

Based on the reports, if the synthesized design meets the required design criteria the designer proceeds to the next stage. If not, the constraints have to be changed or the core design files have to be changed iteratively until the design requirements are met.

4.1.4 Low Power Synthesis

Low Power Synthesis was performed to see if this feature of the tool could help in reducing the area and power consumed by the processor. The Synthesis tool explores the possibility of introducing clock-gating and sleep-mode logic in the design to reduce the power consumption. Besides this, the tool also explores the option of removing redundant blocks in the design to further reduce the area of the chip.

Design Flow and Results

Figure 4.3 shows the basic design flow using the low power synthesis option in Ambit Buildgates. The first few steps are the same as running Buildgates in normal mode.

Read in the Verilog Design and Library

This is the same process as described in the previous section 4.1.3. The library files must have the necessary power models required to estimate the power. This can be verified using the ***check_library -power*** command.

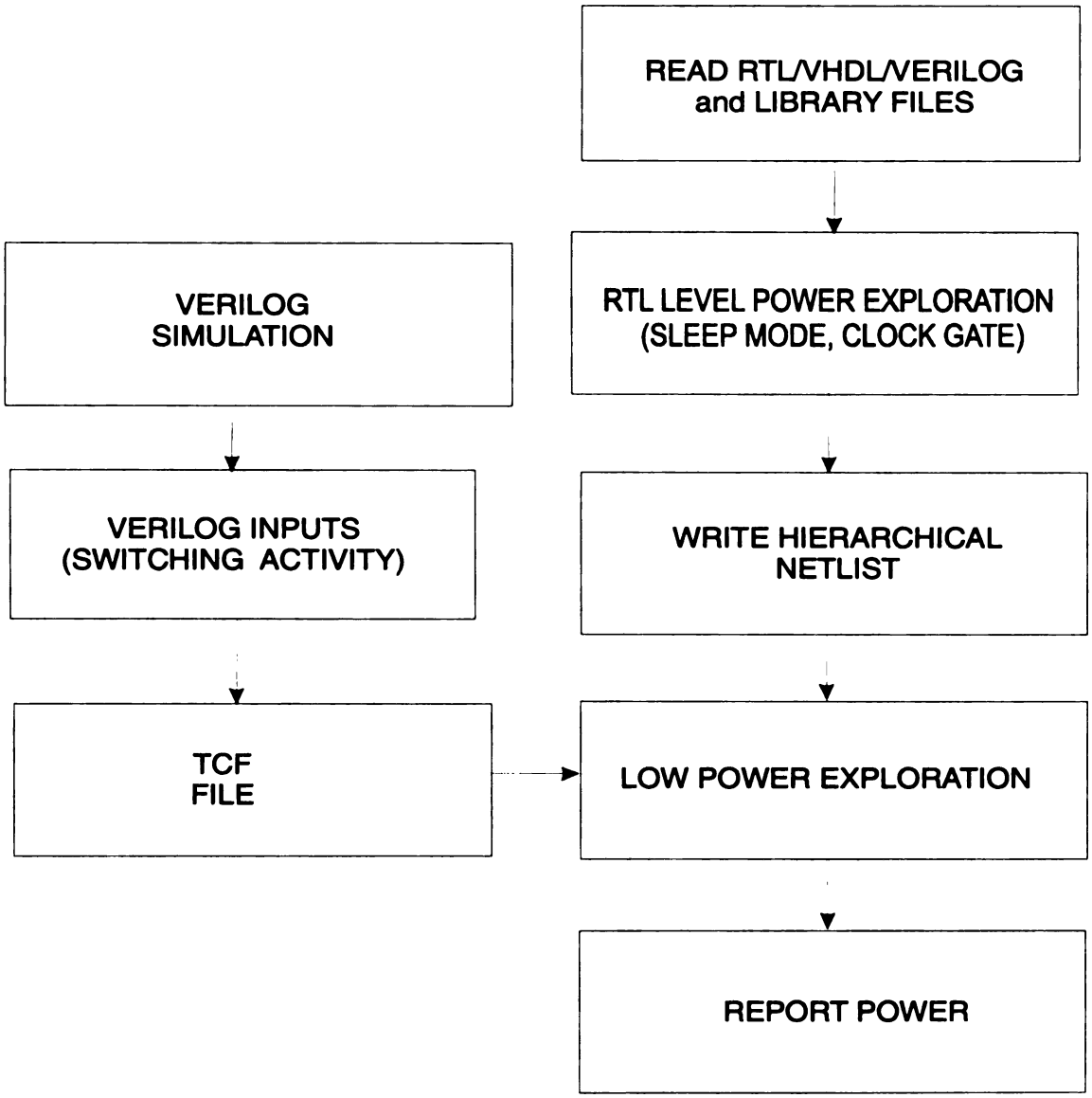


Figure 4.3: Low Power Synthesis Design Flow

RTL Level Synthesis and Power Exploration

The synthesis is started with the *do_build_generic -sleepmode* command that synthesizes the netlist while simultaneously exploring the possibility of inserting sleep mode logic into the design. The next command is the *do_xform_optimize_generic -clockgate* command that explores the possibility of inserting clock-gating logic to save power.

Read in the Toggle Count Format File

This step reads in the TCF file that describes the RTL level switching activity, which is needed to estimate the power of the design. The TCF file is generated from a Value Change Dump (VCD) file using a program called *lpsvcd2tcf* converter. The VCD file is generated during logic simulation by using certain directives in the testbench to record the change in activity of the signals in the design. The toggle count format file is read in using the command:

read_tcf tcffilename.tcf

Committing Logic and Optimizing the design

The last step is to commit the clock gating and sleep mode logic that was generated. This is done using the command:

do_optimize -power

Generate Reports

The power estimated in the synthesis process can be generated using the command. All other reports described in the previous design flow can also be generated using the same procedure. The reports for the low power synthesis are listed in section B.2 of the appendix.

report_power

4.1.5 Power and Area Results from Synthesis

The synthesis tools are used to estimate the power and area of the design. This early estimation helps the designer to see if the results obtained meet the design criteria. It facilitates making changes early in the design cycle to improve the performance and

reduce the design time. Table 4.1 lists the area and power estimates of the top module *micro* of the design obtained through both low power synthesis and normal synthesis.

Table 4.1: Area and Power Results of the Processor

Synthesis Type	Area	Power
Normal Synthesis	5343866.24 sq.microns	5.3417 mW
Low Power Synthesis	5061774.73 sq.microns	3.6241 mW

The results obtained here are an estimate using the Toggle Count Format (TCF) file discussed earlier. The power consumed by a circuit is a function of the frequency of change of inputs (discussed in section 2.5) and the TCF file records the activity of the input signal. The power estimates made above are for a random sequence of instructions chosen to test the working of the processor.

4.2 Place and Route

Place and Route is the process of automatic layout generation from the synthesized netlist. The place and route tool used the netlist generated from Builgates along with library files to automatically place and route the cells thus saving time involved in manual layout.

4.2.1 Need for Place and Route Tools

Place and Route tools form an integral part of the top down design flow methodology. The main advantage of using place and route tools is the significant design time saved when designing chips with a high transistor count. This layout process uses the netlist mapped to a target technology library, which is generated by a synthesis tool, such as Ambit Buildgates. The tool also uses another file from the target technology library vendor that maps all the cells defined in the .TLF file to their corresponding layouts. The place and route tool has the capability of laying out power and ground rails,

input/output cells of the chip, as well as the core cells of the design. The tool also reports statistics such as area of the chip, the number of wires used, number of pins and percentage utilization of the defined chip area. These features help improve the overall efficiency of the design and reduce the time to market.

4.2.2 Tools and Requirements

Silicon Ensemble version 5.3 from Cadence Design Systems was used as the place and route tool to implement the complete physical layout of the processor. This tool is currently being used in the industry and can support large designs with up to one million transistors.

Features

- Provides the basic place and route features – floor planning, placement, routing, clock tree generation, extraction and timing analysis.
- Performs restructuring of sub-optimal netlists from synthesis tools and transforms to an optimized netlist based on actual physical information.
- Silicon Ensemble includes advanced features to meet requirements of up to 130nm technology.

4.2.3 Design Flow and Results

This section describes the step-by-step process of the Silicon Ensemble design flow. The results obtained during the physical layout of the processor will also be described.

Import Data

The following data has to be imported into Silicon Ensemble:

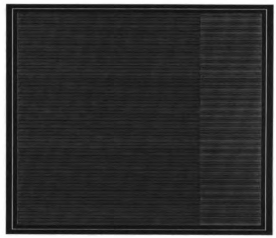
- LEF- Import the Library Exchange Format file that contains the necessary technology and cell information
- Verilog Description- Import the verilog netlist that was generated by Ambit Buildgates after the synthesis process.
- DEF- Read in any extra information such as corner cell information in the design exchange format.

Initialize the Design

Initializes the floorplan based on the design that was read in the previous step. Silicon Ensemble creates a core area (height and width), defines the number of rows, I/O to core spacing and also calculates the core utilization. Figure 4.4 shows the Initialized floorplan for the processor. The row defined is a geographical boundary (that corresponds to the pitch of the standard cells) used to place cells from the library. The percentage utilization of the cell defines the area of the row that is actually occupied by cells from the library. The rows are also flipped and abutted against each other to share the power and ground lines thereby reducing the area required. Table 4.2 shows the information obtained after initializing the floorplan. The results define an aspect ratio of 1.0 (square chip) with an area of 5955015.28 sq.microns.

Table 4.2: Parameters specified in the Initialize Floorplan step

Aspect Ratio:	1.00
Width:	2440.29 microns
Height:	2440.29 microns
Core row utilization	85.13%
Chip Area	5955015.28 sq. microns.
IO to Core Distance (microns)	X: 50.00 Y: 50.00
Number of Standard Cell Rows	188



7

(a) View of the entire Chip **(b) Zoomed in View**

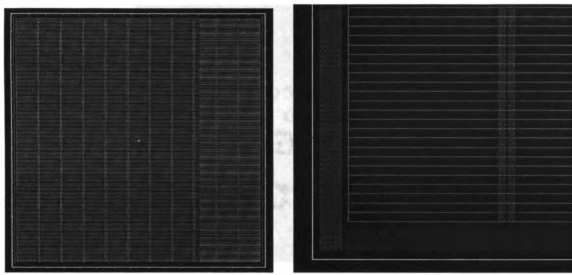
Figure 4.4: Initialize the Floorplan to Define the Chip Area

Place I/Os and Blocks

The I/O pads are placed before the blocks and can be placed based on constraints specified in the I/O constraints file. Figure 4.4 shows the floorplan after the placement of the I/O pads.

Plan Power Routing

In this step the power grid is placed both as a ring around the entire chip and as stripes in the core of the design. Figure 4.5 shows the core after power routing was completed.

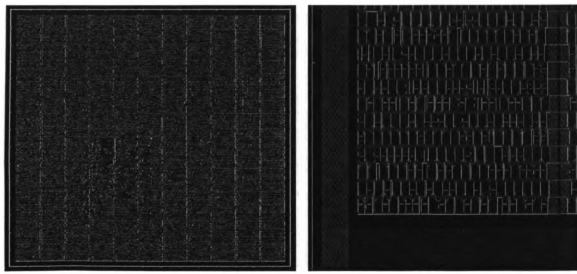


(a) View of the entire Chip **(b) Zoomed in View**

Figure 4.5: Power Planning to Place the Vdd and Gnd Rings

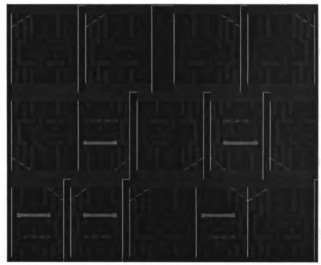
Place Cells

This step places all the core cells of the design. If the placement is unable to fit in the given size, changes have to be made to the floorplan to complete the placement. Figure 4.6 shows the chip after the cells have been placed.



(a) View of the entire Chip **(b) Zoomed in to show Power Lines and Cells**

Figure 4.6: Placing the Cells in the Design

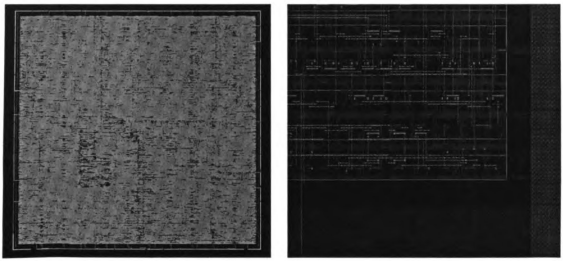


(c) Zoomed in to show Internal Cell View

Figure 4.6 (cont'd): Placing the Cells in the Design

Route

After the placement process is completed we have a chip with all the cells in the design; however, no connections have been made between the cells. First the power rings are connected to the power lines of the cells. Then the cells are routed using the *wroute* command. Figure 4.7 shows the chip after the routing process has completed.



(a) View of the entire Chip **(b) Zoomed in View**

Figure 4.7: Routing the Design by Physically Connecting the Placed Cells

Export Design

The routed design is then exported to either a GDSII format or DEF (Design Exchange Format) so that it can migrate to a custom chip design tool for final testing.

This completes the usage of the place and route tools. The design when exported to a full custom design tool such as Virtuoso Layout editor from Cadence Design Systems can be used to perform the analog simulations of the chip. The analog simulation will provide the necessary timing information of the chip that can be used to estimate the speed rating or MHz rating of the processor.

5. Conclusion

This chapter summarizes the research and the results obtained in the thesis. Future work in this area is proposed to further improve the overall design of the processor.

5.1 Summary

The research in this thesis studies and implements the architecture of an application specific programmable processor for sensor network data processing. The work describes the application of the processor with respect to sensor networks and identifies the motivation behind this research. A 16-bit RISC processor with a flexible instruction set architecture was presented. The features and processing capabilities of the processor were described. A special sleep mode was implemented to reduce the overall power consumption of the processor. Further, the Intramodule Multielement Microsystem bus used in sensor networks was interefaced to a special port on the processor. Instructions to send and receive data over the bus were implemented.

The implementation of the processor was based on a top-down design flow that reduced the design time and complexity of the project. The tools used in this research were from Cadence Design Systems. The register transfer level description of the processor was done using the NC Verilog compiler and simulation was done using the NC Sim package. Results were provided verifying the instruction set of the processor, the sleep mode and the special port design to communicate with the sensor nodes. The design was then synthesized using Ambit Buildgates to generate a netlist. Finally Silicon Ensemble was used to generate the layout of the processor. Detailed design flows were developed to setup and run these tools in Michigan State University labs.

5.2 Future Work

The processor designed meets the basic requirements of a programmable processor for sensor-based networks. Future improvements in the following areas will improve the functionality and performance of the processor.

The data received from the sensor nodes, i.e., the raw sensor readings, has to be processed through a certain calibration and compensation scheme. The need for this calibration engine arises from the non-linearity's and cross sensitivities that are introduced into the transfer curve of signals that are transformed from one domain to another. This block is currently under development at the AMSAC Research Laboratory at Michigan State University and will, in the future, be integrated onto the same chip as the processor. Design changes should be made to the processor to operate and control this block that processes the data received by from the sensor nodes.

The 3-stage pipeline designed in the processor does not account for data and control hazards that could affect the operation of the processor. At present the processor would generate an error in results if affected by data hazards. The processor being a single-issue processor does not have any structural hazards. In the future data hazards such as read after write (RAW) and write after read (WAR) should be accounted for. A register-renaming scheme could be adopted with a pool of registers to keep track of those used thereby avoiding the mentioned hazards [24].

APPENDICES

APPENDIX A

A.1 Instruction Set Architecture

This table lists the instructions of the processor along with the opcode

Table A.1: Instruction Set with Opcodes

Opcode (hex)	Instruction	Description
0x00h	Inc RegX	Increments the contents of RegX by 1; Result stored in RegX; Flags Set: C, Z, N
0x01h	LDA RegX, #data	Load the Immediate data into RegX; Flags Set: None
0x02h	STA RegX, &mem	Stores the contents of RegX in mem location specified by Inst [7:0]. Flags Set: None
0x03h	Add RegX, RegY	Add contents of RegX and RegY; Result stored in RegX; Flags Set: C, Z, N
0x04h	Sub RegX, RegY	Subtract contents of RegY from RegX; Result stored in RegX; Flags Set: C, Z, N
0x05h	AND RegX, RegY	AND contents of RegX and RegY; Result stored in RegX
0x06h	XOR RegX, RegY	XOR contents of RegX and RegY; Result stored in RegX
0x07h	OR RegX, #data	OR contents of RegX with immediate data; Result stored in RegX
0x08h	Add RegX, #data	Add contents of RegX to immediate value; Result stored in RegX; Flags Set: C, Z, N
0x09h	AND RegX, #data	AND contents of RegX with immediate data; Result stored in RegX
0x0Ah	XOR RegX, #data	XOR contents of RegX with immediate data; Result stored in RegX
0x0Bh	OR RegX, #data	OR contents of RegX with immediate data; Result stored in RegX
0x0Ch	Mov RegX, RegY	Move contents of RegY to RegX. Flags Set: None
0x0Dh	Mov RegX, #data	Moves the immediate data to RegX; Flags Set: None
0x0Eh	TWO RegX	Two's complement of RegX; Result stored in RegX
0x0Fh	Addc RegX, RegY	Add with carry RegX and RegY; Result stored in RegX; Flags Set: C, Z, N
0x10h	Cmp RegX, RegY	Compares the contents of RegX and RegY; No result stored; Flags Set: C, Z, N, GT, LT
0x11h	RRA RegX	Rotate right the RegX through no of bits specified; Store Result in RegX

Table A.1 (cont'd): Instruction Set with Opcodes

Opcode (hex)	Instruction	Description
0x12h	RLA RegX	Rotate left the RegX through no of bits specified; Store Result in RegX
0x13h	RLC RegX	Rotate left the RegX through the Carry flag in the condition code register; Flags Set: C
0x14h	RRC RegX	Rotate right the RegX through the Carry flag in the condition code register; Flags Set: C
0x15h	BRA #Imm	Branch to the location specified in the 8-bit address encoded in the instruction; Change PC to new value.
0x16h	BRA EN #Imm	Branch if EQ or N flag are set to address encoded in the instruction; Change PC to new value
0x17h	BRA GL #Imm	Branch if GT or LT flags are set to address encoded in the instruction; Change PC to the new value.
0x18h	Inc RegX	Increments the contents of RegX by 1; Result stored in RegX; Flags Set: C, Z, N
0x19h	Dec RegX	Decrements the contents of RegX by 1; Result stored in RegY; Flags Set: C, Z, N
0x1Ah	Noop	No Function Performed; Flags Set: None
0x1Bh	Str #addr	Received data is stored in #address of the sensor data memory;
0x1Ch	Rcx #data	Receives data over the DOUT line of the sensor bus; 1 byte of data is received over 8 clock cycles on the DCLK line;
0x1Dh	Snd #data	Sends the data over the DIN line of the sensor bus; 1 byte of data is sent over 8 clock cycles on the DCLK line;
0x1Eh	Pull #sig	Pull the NIOE/NTRIG lines high/low
0x1Fh	Sleep Mode	Changes to Sleep Mode; Clock Turned off; waits for SensIntr

APPENDIX B

Synthesis Results

This appendix briefly describes the results obtained from the synthesis. They are summarized reports of area and hierarchy of the cells. All units are in size are in microns and area in square microns.

B.1 Ambit Synthesis Results

The following report shows a summary of the area information obtained from the Ambit Buildgates. The report shows the area of the top most module micro to be **5343866.24 sq. microns** and lists the area of each sub module within the topmost module. A sample report after that also list the number of instances within a particular module, area of each instance, number of combinational and non-combinational instances. This was done hierarchically for all the modules in the design .

```
+--------------------------------+
| Report  | report_area          |
|---------+----------------------|
| Options | -summary             |
+---------+----------------------+
| Date    | 20030426.195408      |
| Tool    | ac_shell             |
| Release | v4.0-s008            |
| Version | Apr 20 2001 04:20:50 |
+---------+----------------------+
| Module  | micro                |
+--------------------------------+

Summary Area Report
-------------------
+-------------------------------------------------------------------------+
|               Module |   Wireload |  Cell Area |   Net Area | Total Area |
|----------------------+------------+------------+------------+------------|
|                micro |       NONE | 5343866.24 |       0.00 | 5343866.24 |
|               clkGen |       NONE |    1707.61 |       0.00 |    1707.61 |
|           pipeStage1 |       NONE | 4024317.93 |       0.00 | 4024317.93 |
|           pipeStage2 |       NONE |   87172.07 |       0.00 |   87172.07 |
```

Figure B.1: Area Report of the Processor in Normal Mode

pipeStage3	NONE	1229884.81	0.00	1229884.81
pswreg	NONE	307.93	0.00	307.93
sleepmode	NONE	475.89	0.00	475.89
buff	NONE	0.00	0.00	0.00
jkff_0	NONE	447.90	0.00	447.90
jkff_1	NONE	447.90	0.00	447.90
jkff_2	NONE	447.90	0.00	447.90
jkff_3	NONE	363.92	0.00	363.92
dmasig	NONE	643.85	0.00	643.85
imem	NONE	4007045.88	0.00	4007045.88
memaddRegister	NONE	3373.23	0.00	3373.23
pipereg11	NONE	4926.87	0.00	4926.87
programCounter	NONE	7250.34	0.00	7250.34
ps1Control	NONE	923.79	0.00	923.79
AWMUX_1024_16	NONE	1713992.14	0.00	1713992.14
AWACL_UNS_INC_10_C	NONE	2617.40	0.00	2617.40
pipereg21_data1_0	NONE	4297.02	0.00	4297.02
pipereg21_data1_1	NONE	4297.02	0.00	4297.02
pipereg21_data1_2	NONE	4297.02	0.00	4297.02
pipereg22_opcode	NONE	1371.69	0.00	1371.69
pipereg23_data2	NONE	2169.50	0.00	2169.50
pipereg24_addr	NONE	839.81	0.00	839.81
pipereg26_immin	NONE	685.84	0.00	685.84
regfile	NONE	69060.21	0.00	69060.21
AWMUX_8_16_0	NONE	11533.36	0.00	11533.36
AWMUX_8_16_1	NONE	11253.43	0.00	11253.43
alu	NONE	190566.43	0.00	190566.43
din_comm	NONE	4898.88	0.00	4898.88
dmem	NONE	1007993.54	0.00	1007993.54
dout_comm	NONE	11295.42	0.00	11295.42
mux3to1	NONE	4129.06	0.00	4129.06
pipereg31	NONE	2225.49	0.00	2225.49
pipereg32_0	NONE	4297.02	0.00	4297.02
pipereg36	NONE	4325.01	0.00	4325.01
AWACL_UNS_ADD_16_C_0	NONE	14948.58	0.00	14948.58
AWACL_UNS_ADD_16_C_1	NONE	14290.73	0.00	14290.73
AWACL_UNS_DEC_16_C	NONE	4940.87	0.00	4940.87
AWACL_UNS_GT_16_C	NONE	4884.88	0.00	4884.88
AWACL_UNS_INC_16_C_0	NONE	4786.91	0.00	4786.91
AWACL_UNS_INC_16_C_1	NONE	5794.68	0.00	5794.68
AWACL_UNS_INC_16_C_2	NONE	5766.68	0.00	5766.68
AWACL_UNS_LT_16_C_0	NONE	4311.01	0.00	4311.01
AWACL_UNS_SUB_16_C	NONE	16670.19	0.00	16670.19
AWMUX_256_16	NONE	417734.50	0.00	417734.50
mux	NONE	335.92	0.00	335.92

Figure B.1 (cont'd): Area Report of the Processor in Normal Mode

The Toggle Count Format was used to estimate the power of the systems in normal mode without any low power optimizations. Figure B.2 shows the power estimated in the normal mode to be **5.3417 mW** for a random sequence of instructions.

Table B.1: Power Estimated using the Synthesis tool in Normal Mode

Micro				
	Internal Cell	**Leakage**	**Net**	**Total**
Module	**Power (mW)**	**Power (mW)**	**Power (mW)**	**Power (mW)**
	3.6654	6.285e-08	1.6763	**5.3417**
PipeStage1	0.0318	3.000e-12	923103	0.0410
PipeStage3	1.4527	1.279e-08	0.6400	2.0927
PipeStage2	1.7804	6.990e-10	0.7922	2.5726
PipeStage1	0.2965	4.934e-08	0.1108	0.4074
SleepModule1	0.0142	3.000e-11	0.0798	0.0940
Clkmodule	0.0898	1.600e-11	0.0000	0.0898

This report shows the hierarchy of the design from the topmost module to each module within.

```
|-micro(m)
||-clkGen(m)
|||-buff(m)
|||-jkff_0(m)
|||-jkff_1(m)
|||-jkff_2(m)
|||-jkff_3(m)
||-pipeStage1(m)
|||-dmasig(m)
|||-imem(m)
||||-AWMUX_1024_16(m)
|||-memaddRegister(m)
|||-pipereg11(m)
|||-programCounter(m)
||||-AWACL_UNS_INC_10_C(m)
|||-ps1Control(m)
||-pipeStage2(m)
|||-pipereg21_data1_0(m)
|||-pipereg21_data1_1(m)
|||-pipereg21_data1_2(m)
|||-pipereg22_opcode(m)
|||-pipereg23_data2(m)
|||-pipereg24_addr(m)
|||-pipereg26_immin(m)
|||-regfile(m)
||||-AWMUX_8_16_0(m)
||||-AWMUX_8_16_1(m)
||-pipeStage3(m)
|||-alu(m)
||||-AWACL_UNS_ADD_16_C_0(m)
||||-AWACL_UNS_ADD_16_C_1(m)
||||-AWACL_UNS_DEC_16_C(m)
||||-AWACL_UNS_GT_16_C(m)
||||-AWACL_UNS_INC_16_C_0(m)
||||-AWACL_UNS_INC_16_C_1(m)
||||-AWACL_UNS_INC_16_C_2(m)
||||-AWACL_UNS_LT_16_C_0(m)
||||-AWACL_UNS_SUB_16_C(m)
```

Figure B.2: Hierarchical Report of the Processor in Normal Mode

```
|||-din_comm(m)
|||-dmem(m)
||||-AWMUX_256_16(m)
|||-dout_comm(m)
|||-mux3to1(m)
|||-pipereg31(m)
|||-pipereg32_0(m)
|||-pipereg36(m)
||-pswreg(m)
||-sleepmode(m)
|||-mux(m)
```

Figure B.2 (cont'd): Hierarchical Report of the Processor in Normal Mode

B.2 Low Power Synthesis Results

This section reports the area and hierarchy of the design when optimized for low power synthesis. In the low power synthesis mode sleep mode and clock gating logic was inserted into the design. Besides that the tool also optimizes the design by removing redundant blocks to further reduce the area of the design. The area of the processor now after low power synthesis optimization is **5061774.73 sq. microns.**

```
+--------------------------------+
| Report  | report_area          |
|---------+----------------------|
| Options | -summary             |
+---------+----------------------+
| Date    | 20030426.195450      |
| Tool    | ac_shell             |
| Release | v4.0-s008            |
| Version | Apr 20 2001 04:20:50 |
+---------+----------------------+
| Module  | micro                |
+--------------------------------+

Summary Area Report
-------------------
+-------------------------------------------------------------------------+
|               Module |   Wireload |  Cell Area |   Net Area | Total Area |
|----------------------+------------+------------+------------+------------|
|                micro |       NONE | 5061774.73 |       0.00 | 5061774.73 |
|               clkGen |       NONE |    1707.61 |       0.00 |    1707.61 |
|           pipeStage1 |       NONE | 3821182.37 |       0.00 | 3821182.37 |
|           pipeStage2 |       NONE |   96381.97 |       0.00 |   96381.97 |
|           pipeStage3 |       NONE | 1141718.96 |       0.00 | 1141718.96 |
|              pswreg |       NONE |     307.93 |       0.00 |     307.93 |
|            sleepmode |       NONE |     475.89 |       0.00 |     475.89 |
|                 buff |       NONE |       0.00 |       0.00 |       0.00 |
|               jkff_0 |       NONE |     447.90 |       0.00 |     447.90 |
```

Figure B.3: Area Report of the Processor in Low Power Mode

jkff_1	NONE	447.90	0.00	447.90
jkff_2	NONE	447.90	0.00	447.90
jkff_3	NONE	363.92	0.00	363.92
dmasig	NONE	643.85	0.00	643.85
imem	NONE	3803098.51	0.00	3803098.51
memaddRegister	NONE	4059.07	0.00	4059.07
pipereg11	NONE	4772.91	0.00	4772.91
programCounter	NONE	7488.29	0.00	7488.29
ps1Control	NONE	909.79	0.00	909.79
AWMUX_1024_16	NONE	1609673.99	0.00	1609673.99
AWACL_UNS_INC_10_C	NONE	2197.50	0.00	2197.50
pipereg21_data1_0	NONE	4297.02	0.00	4297.02
pipereg21_data1_1	NONE	4297.02	0.00	4297.02
pipereg21_data1_2	NONE	4297.02	0.00	4297.02
pipereg22_opcode	NONE	1371.69	0.00	1371.69
pipereg23_data2	NONE	2169.50	0.00	2169.50
pipereg24_addr	NONE	839.81	0.00	839.81
pipereg26_immin	NONE	685.84	0.00	685.84
regfile	NONE	78270.11	0.00	78270.11
AWMUX_8_16_0	NONE	11155.45	0.00	11155.45
AWMUX_8_16_1	NONE	11155.45	0.00	11155.45
alu	NONE	156890.13	0.00	156890.13
din_comm	NONE	4101.06	0.00	4101.06
dmem	NONE	954497.77	0.00	954497.77
dout_comm	NONE	11701.33	0.00	11701.33
mux3to1	NONE	3821.13	0.00	3821.13
pipereg31	NONE	1931.56	0.00	1931.56
pipereg32_0	NONE	4297.02	0.00	4297.02
pipereg36	NONE	4325.01	0.00	4325.01
AWACL_UNS_ADD_16_C_0	NONE	11575.35	0.00	11575.35
AWACL_UNS_ADD_16_C_1	NONE	11575.35	0.00	11575.35
AWACL_UNS_DEC_16_C	NONE	4115.06	0.00	4115.06
AWACL_UNS_GT_16_C	NONE	4227.03	0.00	4227.03
AWACL_UNS_INC_16_C_0	NONE	4269.02	0.00	4269.02
AWACL_UNS_INC_16_C_1	NONE	4269.02	0.00	4269.02
AWACL_UNS_INC_16_C_2	NONE	4269.02	0.00	4269.02
AWACL_UNS_LT_16_C_0	NONE	4227.03	0.00	4227.03
AWACL_UNS_SUB_16_C	NONE	11981.26	0.00	11981.26
AWMUX_256_16	NONE	401512.21	0.00	401512.21
mux	NONE	335.92	0.00	335.92

Figure B.3 (cont'd): Area Report of the Processor in Low Power Mode

Table B.2: Power Estimated in Low Power Synthesis Mode

	Micro			
Module	**Internal Cell**	**Leakage**	**Net**	**Total**
	Power (mW)	**Power (mW)**	**Power (mW)**	**Power (mW)**
	2.5151	6.280e-08	1.1090	**3.6241**
PipeStage1	0.0166	3.000e-12	4.759e-03	0.0213
PipeStage3	0.9301	1.277e-08	0.3554	1.2855
PipeStage2	1.2352	6.780e-10	0.5759	1.8111
PipeStage1	0.2361	4.934e-08	0.0864	0.3224
SleepModule1	7.321e-03	3.000e-12	0.0424	0.0498
Clkmodule	0.0898	1.600e-11	0.0000	0.0898

Figure B.4 shows the power estimated in the low power synthesis mode where the synthesis tool perform optimizations such as removal of redundancy and resizing of gates to reduce the power consumed in the design. The total power of the processor in this mode was estimated to be **3.6241 mW** for the same random sequence used in the normal mode. The hierarchical report has is the same as the report in the normal.

Appendix C

C.1 Place and Route Results

This section briefly describes the reports obtained from Silicon Ensemble – the place and route tool. Figure C.1 describes the Silicon Ensemble Design summary report that lists number of components, pins and nets used.

```
*********************SILICON_ENSEMBLE DESIGN SUMMARY REPORT ************
Time: 11:18:31, 16 April 2003
Design name: micro
Report file name: microFroute.summary
                                                              page 1
Number of macros: 74
Number of components: 47327
Number of pins: 256654
  Number of regular pins: 162112
  Number of special pins: 94542
  Number of unused pins: 0
Number of nets: 47325
Average number of pins per net: 5.42
Number of subnets: 0
Number of routing tracks available: 4706
Number of GCELLS per layer: 56169
```

Figure C.1: Silicon Ensemble Design Summary Report

Figure C.2 describes the wiring report of the design that lists features such as total wire length, number of vias and type of metal used for wires.

```
*********************SILICON_ENSEMBLE WIRING REPORT*********************
Time: 11:22:34, 16 April 2003
Design name: micro
Report file name: ./rpts/micro1/microFroute.wires
                                                              page 1
** (only DETAILED wiring are reported for REGULAR nets)

Total vias in regular wiring: 448642
Total segments in regular wiring: 373604
Total vias in special wiring: 2132
Total segments in special wiring: 472
```

Figure C.2: Silicon Ensemble Wiring Report

```
LAYER name: metal1
  Total wire length: 567171.06 microns
    Length of regular wires: 94638.24 microns
    Length of special wires: 472532.82 microns

LAYER name: metal2
  Total wire length: 1191706.80 microns
    Length of regular wires: 1136079.00 microns
    Length of special wires: 55627.80 microns
LAYER name: metal3
  Total wire length: 1775969.28 microns
    Length of regular wires: 1775969.28 microns
    Length of special wires: .00 microns
LAYER name: metal4
  Total wire length: 1580730.12 microns
    Length of regular wires: 1580730.12 microns
    Length of special wires: .00 microns
LAYER name: metal5
  Total wire length: 321681.24 microns
    Length of regular wires: 321681.24 microns
    Length of special wires: .00 microns

Total wirelength in regular wiring: 4909097.88 microns
Total wirelength in special wiring: 528160.62 microns
Total wirelength in regular+special wiring: 5437258.50 microns
```

Figure C.2 (cont'd): Silicon Ensemble Wiring Report

Figure C.3 describes the routing layers used such as the metal layers, poly layers, the number of vias etc.

```
********************SILICON_ENSEMBLE DESIGN SUMMARY REPORT*******
Time: 11:18:32, 16 April 2003
Design name: micro
Report file name: microFroute.summary
                                                          page 10
** LAYER INFORMATION

Total layers: 27
  Routing layers: 5

Layer information by layer number:
  1 ==> metal1   prefers horizontal routing
  2 ==> metal2   prefers vertical routing
  3 ==> metal3   prefers horizontal routing
  4 ==> metal4   prefers vertical routing
  5 ==> metal5   prefers horizontal routing
  6 ==> nwell   can't route
  7 ==> active   can't route
  8 ==> nactive   can't route
```

Figure C.3: Silicon Ensemble Layer Information Report

```
  9 ==> pactive   can't route
  10 ==> tactive   can't route
  11 ==> nselect   can't route
  12 ==> pselect   can't route
  13 ==> poly   can't route
  14 ==> glass   can't route
  15 ==> pad   can't route
  16 ==> sblock   can't route
  17 ==> text   can't route
  18 ==> res_id   can't route
  19 ==> cap_id   can't route
  20 ==> metalcap   can't route
  21 ==> nodrc   can't route
  22 ==> cc   can't route
  23 ==> via   can't route
  24 ==> via2   can't route
  25 ==> via3   can't route
  26 ==> via4   can't route
  27 ==> VIRTUAL   can't route

Layers in process order (top to bottom):
  nodrc
  metalcap
  metal5
  via4
  metal4
  via3
  metal3
  via2
  metal2
  via
  metal1
  cc
  cap_id
  res_id
  text
  sblock
  pad
  glass
  poly
```

Figure C.3 (cont'd): Silicon Ensemble Layer Information Report

REFERENCES

REFERENCES

[1] G. C. M. Meijer, J. van Drecht, P.C. de Jong and H.Neuteboom, "New concepts for smart signal processors and their application to PSD displacement transducers," *Sensors and Actuators* A, vol. 35, pp. 23030, 1992.

[2] K.D. Wise, "VLSI circuit challenges for sensing Systems," *Digest of Technical Papers Symposium on VLSI Circuits*, pp 10-22, June 1990.

[3] J. Zhang, K. Zhang, Z. Wang, and A. Mason, "A Universal Micro-Sensor Interface Chip with Network Communication Bus and Highly Programmable Sensor Readout," *Conf. Proc. 45th IEEE MWSCAS*, Tulsa, OK, 2002.

[4] IEEE Instrumentation and Measurement Society, "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators-Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats," *IEEE Std 1451.2*, 1997.

[5] A. Mason, N. Yazdi, A. V. Chavan, K. Najafi, and K. D. Wise, "A Generic Multielement Microsystem for Portable Wireless Applications," *(Invited) Proc. IEEE*, vol. 86 (8), pp. 1733-1745, August 1998.

[6] K.L. Kraver, M.R. Guthaus, T.D. Strong, P.L. Bird, G.S. Cha, W. Hold, and R.B. Brown, "A Mixed-Signal Sensor Interface Microinstrument," *Technical Digest, Solid-State Sensor and Actuator Workshop*, pp. 14-17, June 2000.

[7] Texas Instruments, THE MSP430x1xx Architecture Family User's Guide. Texas Instruments literature number SLAU049.

[8] J.B. Sulistyo and D.S. Ha, "Developing Standard Cells for TSMC 0.25um Technology under MOSIS DEEP Rules," Department of Electrical and Computer Engineering, Virginia Tech, Technical Report VISC-2002-02, April 2002.

[9] Cadence Design Systems Inc, Cadence Online Documentation, Product Version 1.0. Copyright © 2001.

[10] K.M. Lim; S.W. Jeong; Y.C. Kim; S.J. Jeong; H.K. Kim; Y.H. Kim; B.Y. Chung; H.L. Roh and H.S. Yang, "CalmRISCTM: a low power microcontroller with efficient coprocessor interface," *IEEE Int. Conference on Computer Design*, pp 299-302, 1999

[11] N. Yazdi, A. Mason, K. Najafi, and K.D. Wise, "A Smart Sensing Microsystem with a Capacitive Sensor Interface," *Digest, IEEE Int. Symposium on Circ. and Systems*, Atlanta GA, vol. IV, pp. 336-339, May 1996.

[12] J. Zhou and A. Mason, "Communication Buses and Protocols for Sensor Networks," *Sensors 2002*, ISSN 1424-8220, 2002

[13] S.M. Sze. Semiconductor Sensors. John Wiley & Sons Inc, 1994.

[14] K.D. Wise and N. Najafi, "The coming opportunities in Microsensor Systems," *Digest IEEE Int. Conf. Solid-State Sensors and Actuators* (Transducers'91), San Francisco, CA, pp. 2–7, June 1991.

[15] N. Najafi and K.D. Wise, "An Architecture and Interface for VLSI Sensors," *Dig. IEEE Solid-State Sensor Workshop*. pp. 76-79, 1990.

[16] A.P. Chandrakasan, S. Sheng and R. W. Brodersen, "Low Power CMOS Digital Design," *IEEE Journals of Solid-State Circuits*, pp. 473-484, vol. 27, no. 4., April 1992

[17] T. Burd and R. Brodersen, "Energy Efficient CMOS Microprocessor Design," *28th Hawaii Int'l Conf. on System Science*, Vol. 1, pp. 288-297, Jan 1995.

[18] K.D. Wise, "Integrated Microinstrumentation Systems: Smart Peripherals for Distributed Sensing and Control," *Dig. IEEE Int. Solid-State Circ. Conf.*, San Francisco, CA, pp. 126–127, Feb 1993.

[19] C. Piguet et al, "Low-Power Design of 8-b Embedded CoolRisc Microcontroller Cores," *IEEE J. of Solid-State Circuits*, Vol. 32, No 7, pp. 1067-1078, July 1997.

[20] M. Olivieri, A. Trifiletti and A. De Gloria, "A low-power microcontroller with on-chip self-tuning digital clock-generator for variable-load applications," IEEE Int. Conference on Computer Design, pp 482- 488, 1999.

[21] D.A. Protopapas, Microcomputer Hardware Design. Pretence – Hall, Inc. Englewood Cliffs, NJ, 1998.

[22] B. Parhami, Computer Arithmetic Algorithm and Hardware Design. Oxford University Press, NY, 2000.

[23] D.E. Thomas, P.R. Moorby, The Verilog Hardware Description Language. Kluwer Academic Publishers, MA, 1995.

[24] D.Patterson and J.Hennessy, Computer Architecture a Quantitative Approach, 2nd Edition. Morgan Kaufmann Publishers, Inc, CA, 1996.