A REPORT ON AN INDUSTRIAL ATTACHMENT WITH GRIT SYSTEMS ENGINEERING, VICTORIA ISLAND, LAGOS

Obafemi Awolowo University, Ile-Ife, Osun State.

Yusuf Taiwo Hassan

02 02 1998

Abstract

Abstract goes here

Dedication

To mum and dad

Declaration

I declare that..

Acknowledgements

I want to thank...

Contents

Chapter 1

Introduction

This report is a short description of my six month internship carried out as a compulsory component of the BSc. Electronic and Electrical Engineering. The internship was carried out at Grit Systems Engineering, Victoria Island, Lagos. This internship report contains my activities that have contributed to achieving a successful program. In this chapter, a description of Students Work Experience Program (SWEP) and the Company is given. The second chapter contains the theoretical background of the activities carried out during the course of the training, followed by the third chapter which discusses the activities. The technical experience and skills acquired during the internship are described in the fourth chapter. Finally I give a conclusion on the internship experience.

1.1 Student Work Experience Scheme. *SWEP*

The SWEP is a program of Student Industrial Work Experience Scheme (SIWES) designed for students in their third and fourth year from the faculty of Technology and Environmental Design and Management. The program aims at inculcating practical, scientific, social, and entrepreneurship skills needed to face the challenges of modern day graduate while also contributing to the overall development of undergraduates in these faculties. SIWES is highly recognized by the Nigerian University Commission (NUC), National Board for Technical Education (NBTE) and National Commission for Colleges of Education (NCCE), which makes her join forces with

Industrial Training Fund (ITF) in making this program attainable.

1.2 Objective of SIWES

The goals that the SIWES seek to accomplish are as follows

- Bridge the gap between the theoretical work/knowledge acquired in the classroom and real practical experience offered in the industries. This enables the students to appreciate/value in so many ways, the theories learnt in class.
- To enlist and strengthens employers involvement in the entire educational process of preparing graduate for employment in the industry.
- To provide students with an opportunity to apply their theoretical knowledge in real work situation, thereby bridging the gap between university work and actual practice.
- To prepare the student for the challenges in the industries and prepare them psychologically for work after school.

1.3 GRIT Systems

GRIT Systems, formally DawnFuel Limited, was founded by Mr. Ifedayo Oladapo in 2011 after he identified the challenge of under electrification in Nigeria. He demonstrated that solar power costs less than running a generator does, demonstrated it and used it as economic reality and use it as the foundation for the case he made for DawnFuel. As well as in the company's marketing campaign. The company after a while settled for installation and maintenance of imported solar products, with its custom built components, as the market did not appreciate locally built inverters.

GRIT Systems got into consultancy and monitoring of energy utilization with its new line of products.These products include The GRIT Energy and Power Monitor (GEPM) and G1 (A utility metering device). The company started with just

2 members (including the founder) and grew into a multi-departmental company with more than 15 workers. The devices were tailored to the unique requirements of under electrified communities and requires a first-hand installation by a trained Grit System's personnel. Once installed, users can remotely view graphs, receive notifications and generate simple language reports about an arbitrarily complex power supply mix.

Some of the functions of the Grit meter are:

- Reduced energy cost - Ensuring generator only runs when it is really needed.
- Multisource energy optimization – Increase in the time spent on cost effective sources like inverter while reducing the time spent on expensive sources like your generator.
- Cost-Benefit Balance - Using data from the metering devices to run energy balance simulations to help determine if and how alternative power sources would save the user money.

Chapter 2

LITERATURE REVIEW.

2.1 Embedded Systems Design

An embedded system is a microprocessor-based system that is built to control a function or range of functions and is not designed to be programmed by the end user in the same way that a PC is. A user can make choices concerning functionality but cannot change the functionality of the system by adding/replacing software. With a PC, this is exactly what a user can do: one minute the PC is a word processor and the next it's a games machine simply by changing the software. An embedded system is designed to perform one particular task albeit with choices and different options.

The last point is important because it differentiates itself from the world of the PC where the end user does reprogram it whenever a different software package is bought and run. However, PCs have provided an easily accessible source of hardware and software for embedded systems and it should be no surprise that they form the basis of many embedded systems.

2.2 Printed Circuit Board

Printed Circuit Boards (PCBs) are thin, rigid, and usually rectangular, with components attached to one or both surfaces. The top and bottom are generally coloured in dark blue or green. Lines running between the components have a slightly dif-

ferent colour. In addition to the top and bottom sides, modern circuit boards have internal planes called layers. Internal layers don't have components but may contain metal lines that carry electricity to and from the components on the top and bottom. For example, the circuit board in the iPhone 4 handset has 10 layers. Layers are critically important in PCB design, so circuit boards are commonly divided into three categories: single-sided, double-sided, or multilayer.

At the very least, a circuit board serves two purposes:

1. Provides mechanical support for a set of components

2. Provides electrical connections between the components

A picture of an populated and unpopulated Printed Circuit Board are shown in Figures 2.1 and 2.2 respectively.

2.3 Electrical Components

An electronic component is any basic discrete device or physical entity in an electronic system used to affect electrons or their associated fields. Electronic components are mostly industrial products, available in a singular form and are not to be confused with electrical elements, which are conceptual abstractions representing idealized electronic components.

Electronic components have a number of electrical terminals or leads. These leads connect to other electrical components, often over wire, to create an electronic circuit with a particular function (for example an amplifier, radio receiver, or oscillator). Basic electronic components may be packaged discretely, as arrays or networks of like components, or integrated inside of packages such as semiconductor integrated circuits, hybrid integrated circuits, or thick film devices. The following list of electronic components focuses on the discrete version of these components, treating such packages as components in their own right.

Components can be classified as passive, active, or electromechanic. The definitions of each class of electrical components are as follows

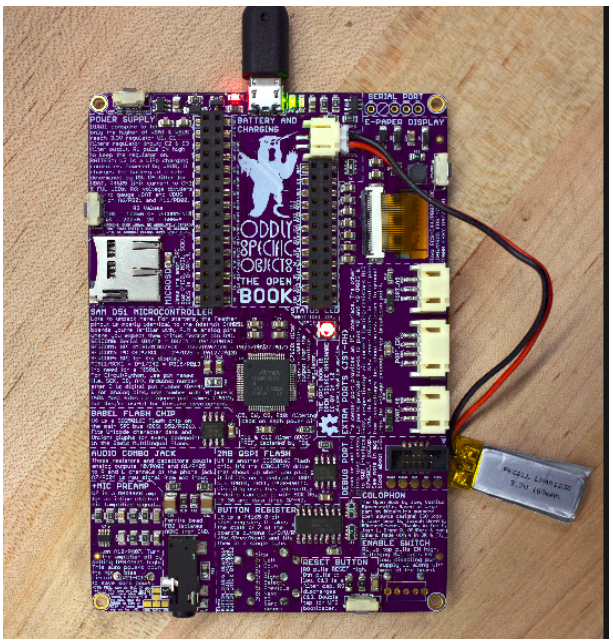

Figure 2.1: A Populated Printed Circuit Board.

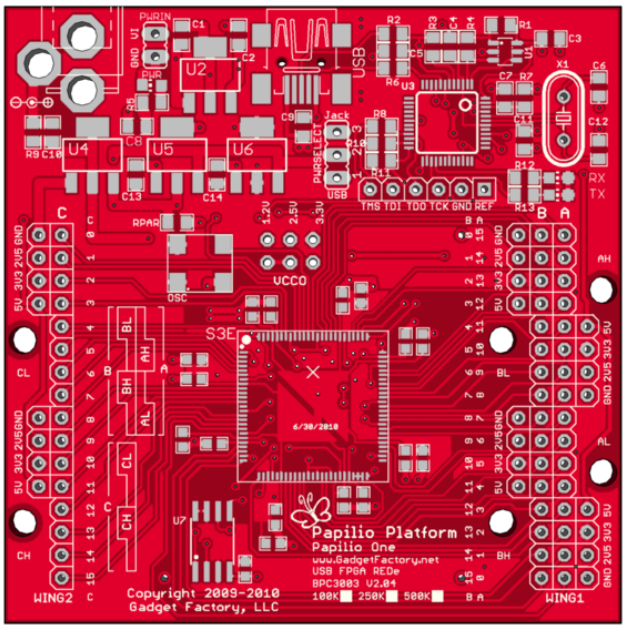

Figure 2.2: An Unpopulated Printed Circuit Board.

1. **Active :** These class of electrical components rely on a source of energy to function and usually can inject power into a circuit. Active components include amplifying components such as transistors, triode vacuum tubes (valves), and tunnel diodes.

2. **Passive :** These class of electrical components can't introduce net energy into the circuit. They also can't rely on a source of power, except for what is available from the circuit they are connected to. As a consequence they can't amplify (increase the power of a signal), although they may increase a voltage or current (such as is done by a transformer or resonant circuit). Passive components include two-terminal components such as resistors, capacitors, inductors, and transformers.

3. **Electromechanic :** can carry out electrical operations by using moving parts or by using electrical connections

A lead is an electrical connection consisting of a length of wire or a metal pad that is designed to connect two locations electrically. Electrical Components are available in three popular forms or technology. This form differ from one another by the form of their leads and method of mounting the Components. They include:

1. **Through Hole Technology:** This refers to the mounting scheme used for electronic components that involves the use of leads on the components that are inserted into holes drilled in printed circuit boards (PCB) and soldered to pads on the opposite side either by manual assembly (hand placement) or by the use of automated insertion mount machines. It is the oldest form of electrical components and is fast becoming obsolete with improving technology.

2. **Surface-mount technology (SMT):** is a method for producing electronic circuits in which the components are mounted or placed directly onto the surface of printed circuit boards (PCBs). An electronic device so made is called a surface-mount device (SMD). In industry, it has largely replaced the through-hole technology construction method of fitting components with wire leads into

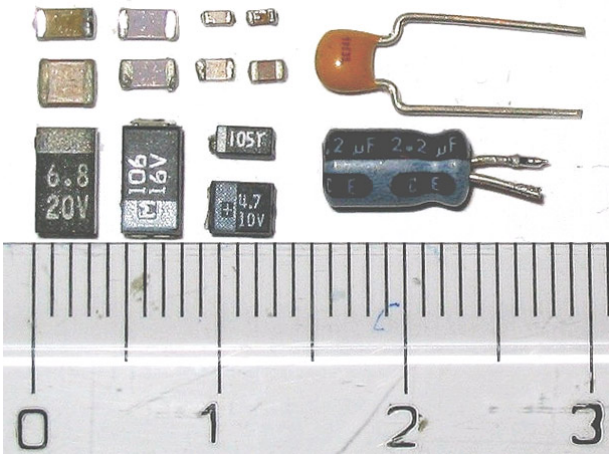

Figure 2.3: A collection of SMD Components.

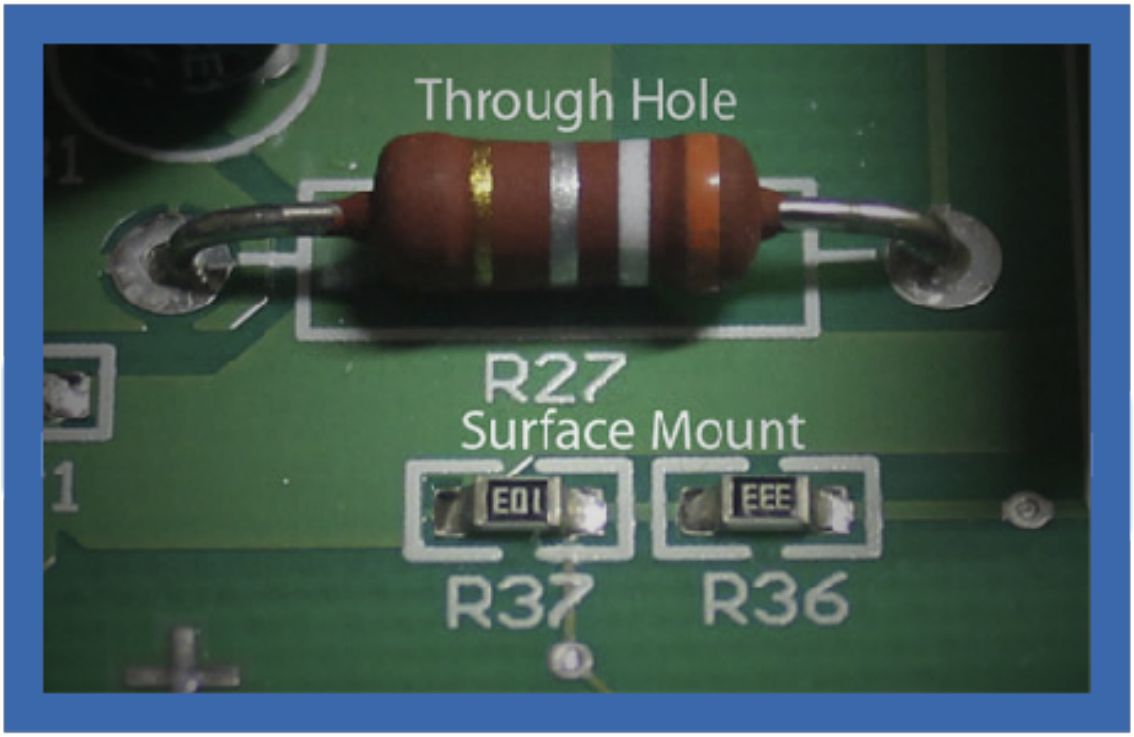

Figure 2.4: A board containing THT and SMD Components.

holes in the circuit board. Both technologies can be used on the same board, with the through-hole technology used for components not suitable for surface mounting such as large transformers and heat-sinked power semiconductors. An SMT component is usually smaller than its through-hole counterpart because it has either smaller leads or no leads at all. It may have short pins or leads of various styles, flat contacts, a matrix of solder balls (BGAs), or terminations on the body of the component.

2.4 LoRa (Long Range)

LoRa (**Lo**ng **Ra**nge) is a low-power wide-area network (LPWAN) technology. It is based on spread spectrum modulation techniques derived from chirp spread spectrum (CSS) technology.[1]It was developed by Cycleo of Grenoble, France and acquired by Semtech, the founding member of the LoRa Alliance. Semtech's LoRa devices and wireless radio frequency technology is a long range, low power wireless platform that has become the de facto technology for Internet of Things (IoT) networks worldwide.

LoRa uses license-free sub-gigahertz radio frequency bands like 433 MHz(Nigeria), 868 MHz (Europe) and 915 MHz (Australia and North America). LoRa enables long-range transmissions (more than 10 km in rural areas) with low power consumption. The technology covers the physical layer, while other technologies and protocols such as LoRaWAN (Long Range Wide Area Network) cover the upper layers.

Semtech's LoRa devices have amassed several hundred known uses cases for smart cities, smart homes and buildings, smart agriculture, smart metering, smart supply chain and logistics, and more. The technology can be utilized by public, private or hybrid networks and provides greater range than Cellular networks. LoRa Technology can easily plug into existing infrastructure and enables low-cost battery-operated IoT applications. The figur 2.5 shows how Lora compares to conventional form of data transmission.

2.4.1 Chirp Spread Spectrum (CSS)

Chirp Spread Spectrum (CSS) is a spread spectrum technique that uses wideband linear frequency modulated chirp pulses to encode information.[1] A chirp is a sinusoidal signal of frequency increase or decrease over time (often with a polynomial expression for the relationship between time and frequency). In 2.6 is an example of an upchirp in which the frequency increases linearly over time. Sometimes the frequency of upchirps increase exponentially over time.

As with other spread spectrum methods, chirp spread spectrum uses its entire allocated bandwidth to broadcast a signal, making it robust to channel noise. Further, because the chirps utilize a broad band of the spectrum, chirp spread spectrum is also resistant to multi-path fading even when operating at very low power. However, it is unlike direct-sequence spread spectrum (DSSS) or frequency-hopping spread spectrum (FHSS) in that it does not add any pseudo-random elements to the signal to help distinguish it from noise on the channel, instead relying on the linear nature of the chirp pulse. Additionally, chirp spread spectrum is resistant to the Doppler effect, which is typical in mobile radio applications.

Chirp spread spectrum was originally designed to compete with ultra-wideband for precision ranging and low-rate wireless networks in the 2.45 GHz band. However, since the release of IEEE 802.15.4a (also known as IEEE 802.15.4a-2007), it is no longer actively being considered by the IEEE for standardization in the area of precision ranging.

Chirp spread spectrum is ideal for applications requiring low power usage and needing relatively low data rates (1 Mbit/s or less). In particular, IEEE 802.15.4a specifies CSS as a technique for use in Low-Rate Wireless Personal Area Networks (LR-WPAN).

2.4.2 Advantages of LoRa (Long Range)

The advantages of LoRa includes the following

1. **Long Range :** LoRa devices are known to be able to transmit for very large distance. This makes them perfect for use in very remote and rural areas.

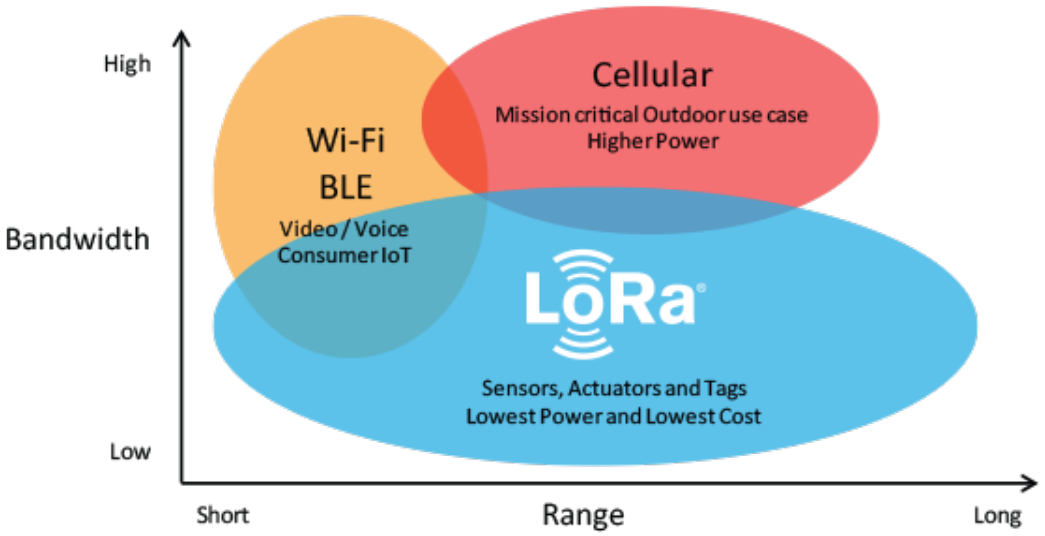

Figure 2.5: A dipiction of Lora compared to other networks.

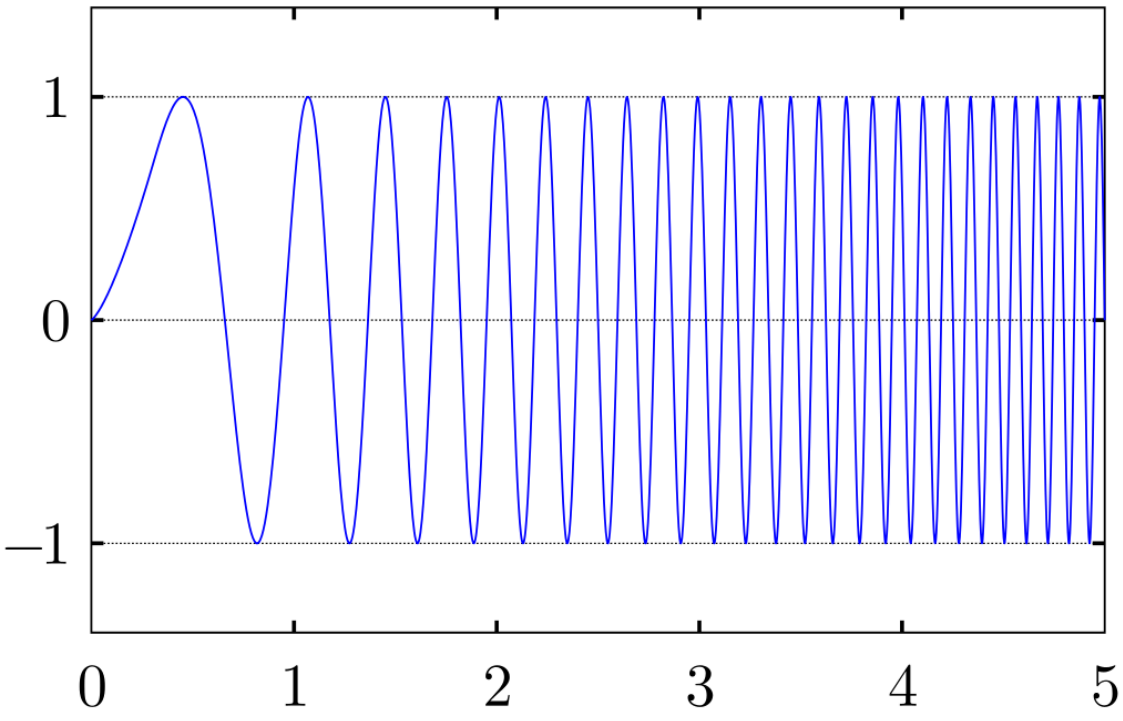

Figure 2.6: A linear frequency modulated upchirp in the time domain.

The maximum number of distance a LoRa device can transmit depends on its power requirement and presence or absence of a Low Noise Amplifier (LNA). Commercially available Lora modules are known to transmit for up to three to eight Kilometers depending on the surrounding.

2. **Low Power :** LoRA devices pride themselves with the abilty to last for a long time while running on batteries. This is due to their very low rate of power consumption. A Lora device requires minimal energy, with prolonged battery lifetime of up to 10 years, minimizing battery replacement costs

3. **License-free radio frequency bands :** Lora Devices make use of radio frequency bands like 433 MHz(Nigeria), 868 MHz (Europe) and 915 MHz (Australia and North America). These frequency bands are available in everywhere and free to be used without a transmitting license.

4. **GeoLocation :** Enables GPS-free tracking applications, offering unique low power benefits untouched by other technologies. Lora Devices can be located by certain triangularization techniques. This means that we can know what device sent a payload and where such device is stationed.

2.4.3 Disadvantages of LoRa (Long Range)

The disadvantages of LoRa includes the following

1. **Low bit Rate :** LoRa devices are unable to transmit data with high bit rate. This is a basic trade off for Low power and long range.The maximum bit rate you'll see is 50kpbs. This makes them useful in systems that dont require transmission of large data within a short period. An example of such system is a sensor mesh

2. **Latency :** Even though it is a low latency modulation it's not low enough for applications that demand very responsive real time communications.

2.5 Field-Programmable Gate Array (FPGA)

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks. In most FPGAs, logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory. The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). Circuit diagrams were previously used to specify the configuration, but this is increasingly rare due to the advent of electronic design automation tools.

2.5.1 Brief History of FPGAs

The FPGA industry sprouted from programmable read-only memory (PROM) and programmable logic devices (PLDs). PROMs and PLDs both had the option of beingprogrammed in batches in a factory or in the field (field-programmable). However, programmable logic was hard-wired between logic gates. Altera was founded in 1983 and delivered the industry's first reprogrammable logic device in 1984 – the EP300 – which featured a quartz window in the package that allowed users to shine an ultra-violet lamp on the die to erase the EPROM cells that held the device configuration. Xilinx produced the first commercially viable field-programmable gate array in 1985 – the XC2064. The XC2064 had programmable gates and programmable interconnects between gates, the beginnings of a new technology and market. The XC2064 had 64 configurable logic blocks (CLBs), with two three-input lookup tables (LUTs).The 1990s were a period of rapid growth for FPGAs, both in circuit sophistication and the volume of production.

In the early 1990s, FPGAs were primarily used in telecommunications and networking. By the end of the decade, FPGAs found their way into consumer, auto-

motive, and industrial applications. Companies like Microsoft have started to use FPGAs to accelerate high-performance, computationally intensive systems (like the data centers that operate their Bing search engine), due to the performance per watt advantage FPGAs deliver. Microsoft began using FPGAs to accelerate Bing in 2014, and in 2018 began deploying FPGAs across other data center workloads for their Azure cloud computing platform. Figure 2.7 shows a picture of a FPGA development board.

2.5.2 FPGA Design Automation Tools

Popular manufacturers of FPGAs include Intel, Xilinx and Lattice. Each manufacturer, in a bid to reduce development time provides certain DESIGN AUTOMATION TOOLS for their customers. Intel boards use Quartus and Xilinx boards use Vivado. Lattice maintained an opensource approach by enabling programming of their boards by the popilar GNU platform. Quartus and Vivado are similar as they allow logic function synthesis by schematic drawing and Hardware Descriptive Language (HDL). The popular HDLs include VHDL and Verilog.

These HDLs, although popular, are quite hard to learn and easy to make mistakes in. For this reason and many more, better languages have emerged. Such languages are either new entirely or based on popular mainstream languages. Although they solve the existing problem of steep learning curves of popular HDLs, they trade it off for a little bit of low level abstractions. Examples of these new languages are Scala, MyHDL and Migen.

Migen

Migen is a Python-based tool for automating VLSI design processes in Field-programmable Arrays (FPGAs). Migen allow application modern software concepts such as object-oriented programming and metaprogramming to design hardware. This results in the reduction of the incidence of human errors and more elegantly structured and maintainable designs. An important advantage of Migen as a Python-based tool is its abilty to benefit from Python's rich features and immense library. The price to

pay is a slightly cluttered syntax, seldomly, when writing descriptions in FHDL, but I believe that this, when compared to VHDL, is considerably acceptable and better.

Few reasons why Migen is better than the conventional VHDL and verilog are as follow:

1. The "event-driven" paradigm of today's dominant hardware descriptions languages (Verilog and VHDL) is often too general. Today's FPGA architectures are optimized for the implementation of fully synchronous circuits. This means that the bulk of the code for an efficient FPGA design falls into three categories:

   (a) Combinatorial statements

   (b) Synchronous statements

   (c) Initialization of registers at reset

   Verilog and VHDL do not follow this organization. This means that a lot of repetitive manual coding is needed, which brings sources of human errors, petty issues, and confusion for beginners.

2. VHDL and Verilog support for procedurally generated logic is extremely limited. The most advanced forms of procedural generation of synthesizable logic that VHDL and Verilog offers are CPP-style directives in Verilog, combinatorial functions, and *generate* statements.

3. VHDL and Verilog support for composite types is very limited. Signals having a record type in VHDL are unidirectional, which makes them clumsy to use e.g. in bus interfaces. There is no record type support in Verilog, which means that a lot of copy-and-paste has to be done when forwarding grouped signals.

4. Testbenches are written in pure Python. This, when used with the large library of python modules, give immense flexibility during simulation. A lot of scenarios can be created and simulated without a lot of stress.

Migen is made up of several related components

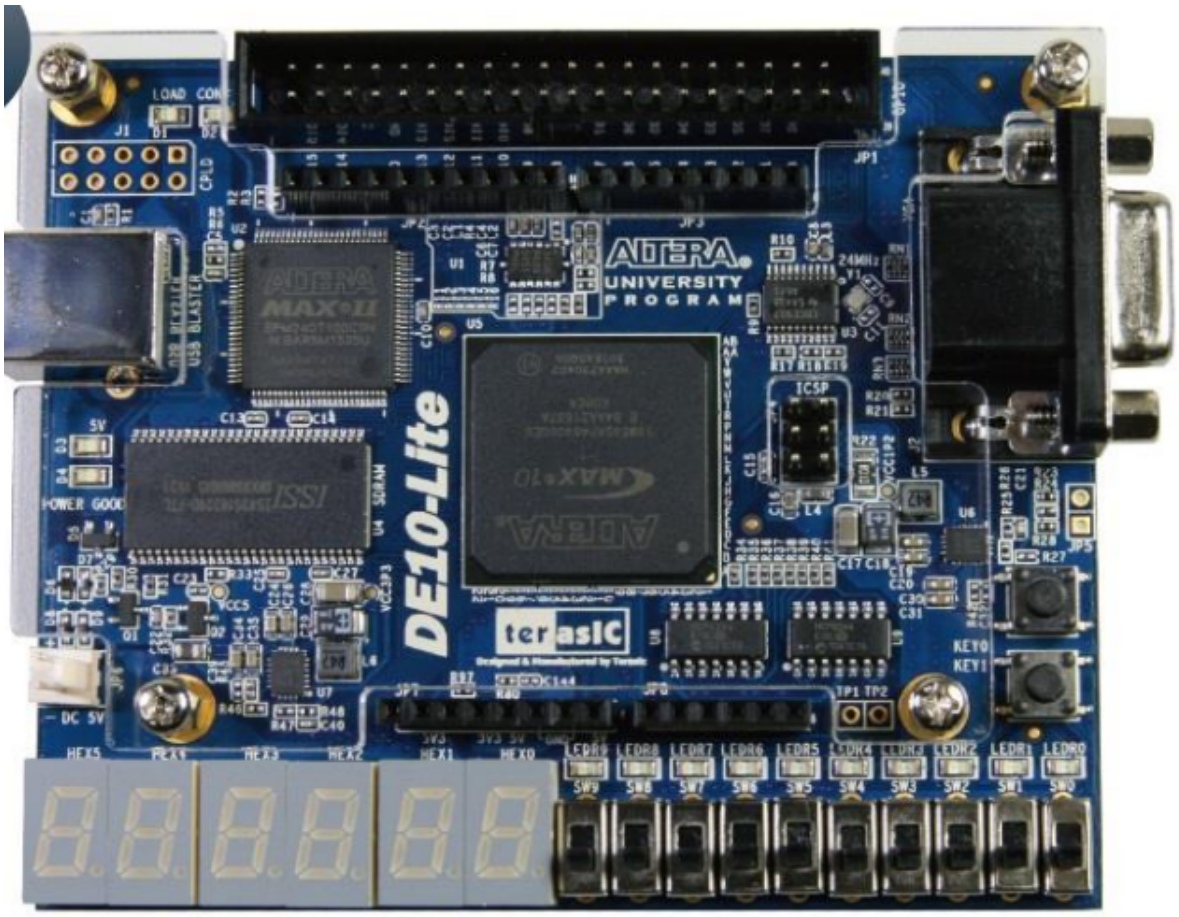

Figure 2.7: A picture of an Intel FPGA Development board.

1. the base language, FHDL
2. a library of small generic cores
3. a simulator
4. a build system

2.6 Central Processing Unit (CPU)

Chapter 3

SIWES ACTIVITIES

3.1 Design of a 16 bit Central Processing Unit (CPU)

A Central Processing Unit (CPU) is the electronic circuit withing a computer that executes instructions that make up a computer.The CPU, by all means, is one of the most important component of a computer system. Since computers vary from one another by size and complexity, so does there CPUs vary. In fact, the power of a computer is a measure of the efficiency and speed of its processor. CPUs can be found in every computer. From the cheap microcontrollers found in toys to the multi-million dollar, number-crunching Super Computers used in large organizations. The CPU performs basic arithmetic, logic, input/output and controlling operations as specified by the instruction. The CPU is to a computer what a brain is to a living organism.

The concept of a CPU and a computer as a whole were explained in a course I took in my third year in University. The course (Computational Structures) explained in details how a computer system was designed from scratch. The course explained the design of a computer system from the designing of an Instruction Set Architecture (ISA) to implementing the design in hardware. The course was taught in a very theoretical way (as expected) and there was a little bit of paucity of depth. This project was undertaken with the hope that this will be remedied.

This is not a super scalar processor. It is written in Hardware Description Language (HDL), flashed unto a Field Programmable Gate Array (FPGA) and tested. This CPU takes multiple instructions to execute the simplest of instructions. It's aimed at a level to educate myself.

I hope to implement the following.

1. A 16-bit CPU Modified Harvard Architecture core
2. 16 16-bit general registers.
3. Basic arithmetic operations, shifts, additions.
4. Branching.
5. A control unit to keep everything in lock-step.
6. Synthesized RAM for instruction and data storage.
7. An in-order CPU with no real pipelining as such

3.1.1 Choice of ISA

An Instruction Set Architecture (ISA) is a very important feature of a processor. It is an abstract model of a computer. An ISA defines the supported data types, the registers, the hardware support for managing main memory fundamental features and the input/output model of an ISA. The ISA of a processor cannot be changed once created because it basically dictates the hardware implementation of the processor. ISAs are classified based on architectural complexity into two.

1. **Complex Instruction Set Computer (CISC):** This ISA has many specialized instructions, some of which may only be rarely used in practical programs.
2. **Reduced Instruction Set Computer (RISC):** This ISA has a small set of simple and general instructions, rather than a large set of complex and specialized instructions. It allows a CPU to have fewer cycles per instruction (CPI) than a complex instruction set computer (CISC)(ref).

The design of an Intruction Set Architecture is not an easy feat. For this reason and time constraints, I opted to design the CPU based on a scratch built ISA designed by Colin "Domipheus" Riley called the Test Processing Unit (TPU). It has a intermediate level of complexity and difficulty to implement. It also gives room for restructuring as the designer is easily accessible for questions and help. The ISR fufils all the requirements stated above and more.

Migen was chosen as the HDL of choice because of its simplicity and Python-like syntax. The ISA supports 14 different instructions. They are the basic and the most found instructions in similar RISC ISAs. The instructions are of three categories

1. Arithmetic Instructions : These instructions are basically for basic numeric operations. This include Addition(ADD) and Subtraction(SUB). All other arithmetic instructions such as Division and Multiplication can and would be implemented as a function of some basic instructions

2. Logical Instructions : These instructions are basically for basic Logical operations. They are carried out both on numerical data and logical data.

3. Conditional and Branching Instructions

4. Memory accessing Instructions:These instructions are very important in the storage and retrieval of data in the memory of a system. They include SAVE, LOAD and WRITE instructions.

There is a need to define the structure of instructions for the ISA. It is a common thing to put the OPCODE first before the ARGUEMENTS of the particular command. An intruction can have a number of arguments or inputs depending on it structure. There are quite a number of ways by which instructions are structured in this regard. This is shown in the Figure 3.1. Some instruction take, as arguements, the regidters number. This means that the operation is performed on the content of the registers and stored in a particular output register. Some other instruction takes the constant inside the instruction and directly perform operation on them. All these depend on the kind of operation to be carried out on the data. An instruction is basically a 16-bit nuumber and it contains both the operation and the

OPERATION	FUNCTION
Add	D = A + B
Subtract	D = A – B
Bitwise Or	D = A or B
Bitwise Xor	D = A xor B
Bitwise And	D = A and B
Bitwise Not	D = not A
Read	D = Memory[A]
Write	Memory[A] = B
Load	D = 8-bit Immediate Value
Compare	D = cmp(A, B)
Shift Left	D = A << B
Shift Right	D = A >> B
Jump/Branch	PC = A Register or Immediate Value
Jump/Branch conditionally	PC = Register if (condition) else nop

Figure 3.1: The Instructions supported by the TPU ISA

operands(Register number or constant). The OPCODE is 4-bit wide and is enough for our 14 operations with space left for two more in the future. The OPCODE starts from the 15th bit and ends on the 12th bit. This is consistent across all instructions . The other forms of instruction bit assignment are shown in Figure 3.3.

A very important thing to note at this point is the fact that the load value is unable to load into a register a 16-bit value at once. This is because the instruction itself is 16 bit and still has to contain other things like the OPCODE and other parameters just as important as the data. This is remedied by allowing the processor to load into the upper and lower part of a register independently. Thsi is controlled by the bit number eight. This parameter is found in all instructions but used only by very few ones.

It is also important to know that our registers are demoted by three bits of the instruction. This allows for referencing only eight registers. This is a core and intrinsic part of the ISA and cannot be changed wwithout changing the ISA itself. The processor does not possess a special register for special numbers and special functions such as the Stack pointer in the Beta machine. This greatly simplifies the complexity of the system at the expense of its performance.

3.1.2 Designing the Register File

The Register File is a part of the processor that contains the registers. There are eight 16-bit individual registers that make up the Register File. The Register File is the storage medium for the processor. It is where data is stored during operation. They are usually made of the fastest kind of memory element to reduce access time during operation.

The Register Files takes in one 16-bit data inputs and two 16-bit data outputs. These outputs allow data from two different register to be available at the output buses for the ALU. The choice of register is made by the output bus select inputs. These inputs are three bits and are just enough to adderess the eight registers of the register files. The destination of the data on the input bus is also controlled by

FORM	OVERVIEW	EXAMPLE INSTRUCTION
RRR	OpCode rD, rA, rB	Add
RRd	OpCode rD, rA	Not
RRs	OpCode rA, rB	Memory Write
R	OpCode rA	Branch to Register Value
RImm	OpCode rD, Immediate	Load
Imm	OpCode Immediate	Branch to Immediate Value

Figure 3.2: Instructions Structure.

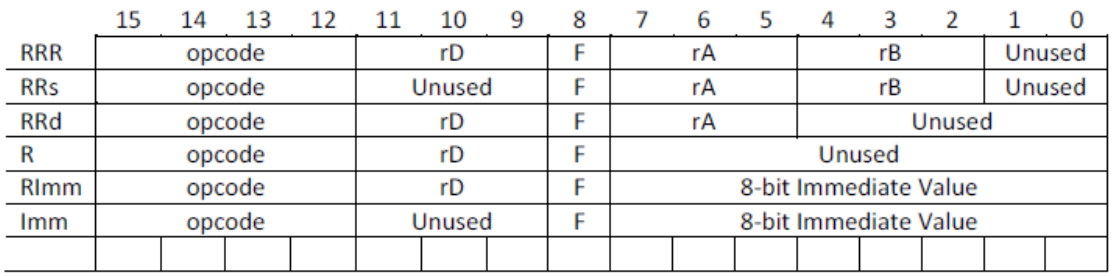

Figure 3.3: Instructions bit Assignments.

a write enable bit and a three bit select input. Therefore, data can be written to the Register files when it is needed. Finally, there is a clock input and an enable bit to activate the whole Register file as a whole.

During the implementation of this module, I wrote the program in such a way that the output from the Register File is held until the select pin changes the choice of register. In other words, if the Register File is enabled , it will always output the content of two registers based on their respective select input. I wrote a testbench to simulate every possible scenarios and I was able to make the Register File perform as follows.

3.1.3 Designing the Instruction Decoder

The decoder is a very important section of a microprocessor. it is responsible for extracting necessary data and commands from an instruction. In other words, it is responsible for pulling bits out from the instruction stream and sending them on to the other units we connect to: The Register file, ALU, and Control unit(s).

The Instruction decoder require the fillowing inputs.

- Clock input
- An enable input
- Instruction input

The outputs include the following

- the Alu OPCODE.
- Selection lines for the Register file (rD, rA and rB).
- Write enable to enable writing into the Register file.
- The Immediate(Constant) value from the instruction.

The ALU OPCODE also contains the flag. The flag signifies different things for different commands. The instructions that require the flag bit make use of it in the code and others that don't simply discard and ignore its existence. For the case of

memory based commands like LOAD and WRITE, it is for putting the immediate value in the upper or lower eight bit part of the selected register rD. For that of JUMP, it is for signaling if the processor should jump to the immediate value or to the value stored in the given register. Figure 3.5 shows the list of the commands, their OPCODES, and the status of the write enable and flag. Figure 3.6 shows a diagrametric representation of an Instruction decoder.

3.1.4 Designing the Arithmetic and Logical Unit

In TPU, we are only doing single cycle operations. Whilst a whole instruction will take multiple cycles to complete, the ALU itself will output a result in a single clock cycle. The other cycles are due to decoding, fetching and any writebacks of results which are required. The ALU takes in two 16-bit data input, an ALU opcode and produces the output of the operation after a number of clock cycles. Other inputs include clock, enable bit, write-enable bit, 16-bit program counter and eight bit immediate value. The program counter and the immediate value inputs are for storing important data required for loading data into the register file. in the case of the program counter, the important data is the next address to go to. This is an important reason for outputing a shouldBranch output bit. This helps to signify when a jump is required during the execution of a particular instruction.

Our ALU process which runs on a rising clock edge, when enable bit is active, will immediately enter a case statement dependent on the alu operation forwarded from the decoder and inputed though the ALUop input. Each if block within the case statement will write to elements of an internal register sresult, which is 18 bits wide, to accommodate carry or overflow status. There is also an internal signal for the shouldBranch output.The shouldBranch output is not computed but inferred from the opcode. This is because of the fact that we already know the number of instructions that requires branching. The ALU code was written to be as optimized as possible as it - just as others - can cause a serious bottleneck in the operation of the microprocessor.

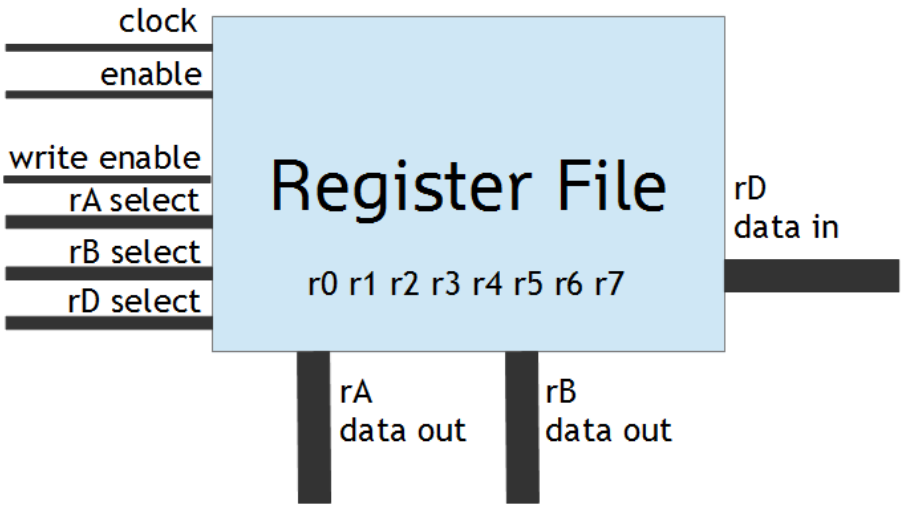

Figure 3.4: Diagramatic representation of an Register File

OPCODE	OPERATION	FORM	WRITESREGISTER?	COMMENTS
0000	ADD	RRR	Yes	
0001	SUB	RRR	Yes	
0010	OR	RRR	Yes	
0011	XOR	RRR	Yes	
0100	AND	RRR	Yes	
0101	NOT	RRd	Yes	
0110	READ	RRd	Yes	
0111	WRITE	RRs	No	
1000	LOAD	RImm	Yes	Flag bit indicates high or low load
1001	CMP	RRR	Yes	Flag bit indicates comparison signedness
1010	SHL	RRR	Yes	
1011	SHR	RRR	Yes	
1100	JUMP	R, Imm	No	Flag bit indicates a jump to register or jump to immediate
1101	JUMPEQ	RRs	No	
1110	RESERVED	–	–	
1111	RESERVED	–	–	

Figure 3.5: OPCODES and Instruction Structure.

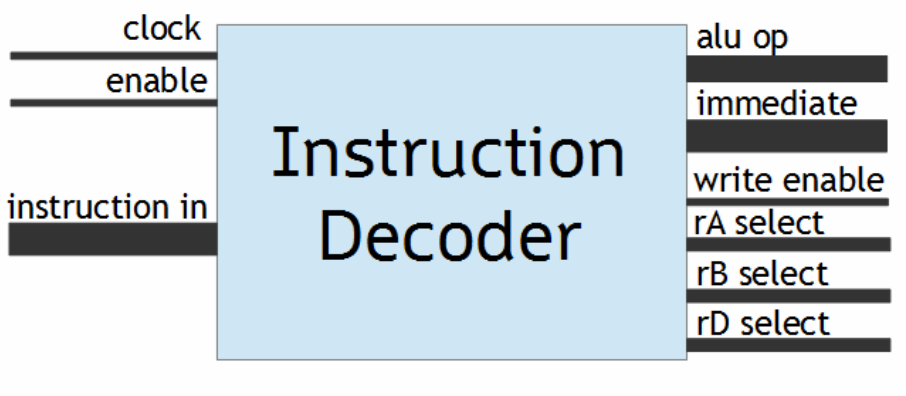

Figure 3.6: Diagramatic representation of an Instruction Decoder

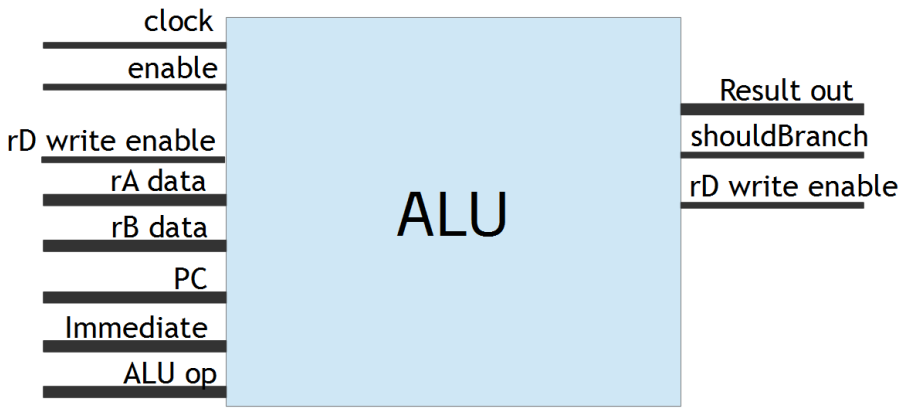

Figure 3.7: Diagramatic representation of the Arithmetic and Logical Unit.

3.1.5 Designing the Control Unit

During the implementation of the other modules, you will notice the presence of enable bits in all of the modules. This gives us the ability to switch our modules inputs on at will. Each module has a register at its output and it will preserve the output until the enable bit is asserted and the computation is carried out. The control unit basically controls the enable bit of every module of the micro processor. it helps in the crude pipelining of data in the processor. The control unit is technically a state machine, but for now it's simple enough we can just classify it as a counter. It is a simple six bit counter. The code was written to shift a bit right through its output and activate the neccessary module. It activates the RAM first since it is where the instruction is fetched and sent to the decoder.

3.1.6 Designing the Program Counter

The Program Counter (PC) is a very important part of a microprocessor. It contains the address (location) of the instruction being executed at the current time. It helps to simplify the process of jumping and branching to any instruction by allowing this to be done by changing its value. In the TPU, our PC unit will obviously hold the current PC address, and on command increment it. It has an input for setting the next PC value, and also the ability to stop – stay at the same location – which we need due to our pipeline being several cycles long.

Our PC is basically required to perform one of the following things at a particular time.

1. Increment PC
2. Set PC to new value
3. Do nothing (halt)
4. Set PC to our reset vector, which is 0x0000.

This means that two bit is enough to tell the PC what to do. This input is the PCin input. The normal state of the PC is when PCin is equal to 0. At this point,

the PC simply increments its current value by one as it is the normal feature of programs to execute from top to bottom.

3.1.7 Simulation and Testing of Different Modules

During the process of implementing the ISA in migen, modularity was paramount. This led to the independent development of individual modules albiet the fact that they were designed to work together. The pipeline was established and data is meant to progress or move from the program counter to the RAM to the decoder to the register file to the ALU and back to the register file. This flow is controlled by the control unit. The individual modules were tested independently and then chained together for a joint test. The testing takes the form of subjecting each module to certain scenarios and checking and confirming that they react and respond in the appropriate ways. The test cases are written in form of testbenchs. The testbenches are written in python and thus inherits the flexibility of python. This is only hindered by the limitation of FHDL itself.

The first module to be tested was the register file. The module was subjected to the following test:

1. A constant clock input was fed into the module
2. The enable pin was asserted and numbers were stored in different momory location sequentially
3. The numbers were read out of the memory location and compared to the initial stored ones
4. The enable pin deasserted and random numbers were fed into the input in different memory location.
5. The numbers were read out of the memory location and compared to the initial stored ones.
6. Data was stored and retrieved form the register file at the same time in one clock cycle.

At the end of the testing, all scenarios were handled well and the module performed all operation in one clock cycle as intended. The register file passed all the test.

The decoder was tested next. Since it is basically meant to extract the embedded data from the instruction and pass them to the neccessary quarters. The module was subjected to the following test:

1. A constant clock input was fed into the module
2. The enable pin was asserted and i6-bit numbers were passed into the input of the decoder.
3. The individual outputs are manually inspected to confirm that the output
4. The enable pin deasserted and the same operation tried.
5. I generated a VCD file and displayed the result of the test. This is shown in Figure **??**.

The decoder operation was not up to par during the first test. This warranted the need to rewrite the code from scratch after which it passed the test. The instruction decoder performs its operation in one clock cycle.

The Arithmetic and Logical Unit was tested next. The job of the ALU is to take in opcode and perform the action intended on the given operands and output the result. Sometimes, the ALU is simply a transparent gateway through which data pass through to be stored in the register file.

The module was subjected to the following test:

1. A constant clock input was fed into the module
2. The enable pin was asserted and numbers and certain opcodes were passed into the ALU inputs.
3. The outputs were read out of the ALU and compared to the required response or result
4. The enable pin deasserted and random numbers were fed into the input.

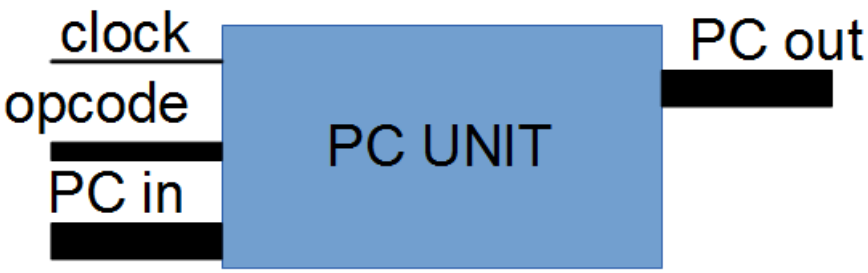

Figure 3.8: Diagramatic representation of the Program counter.

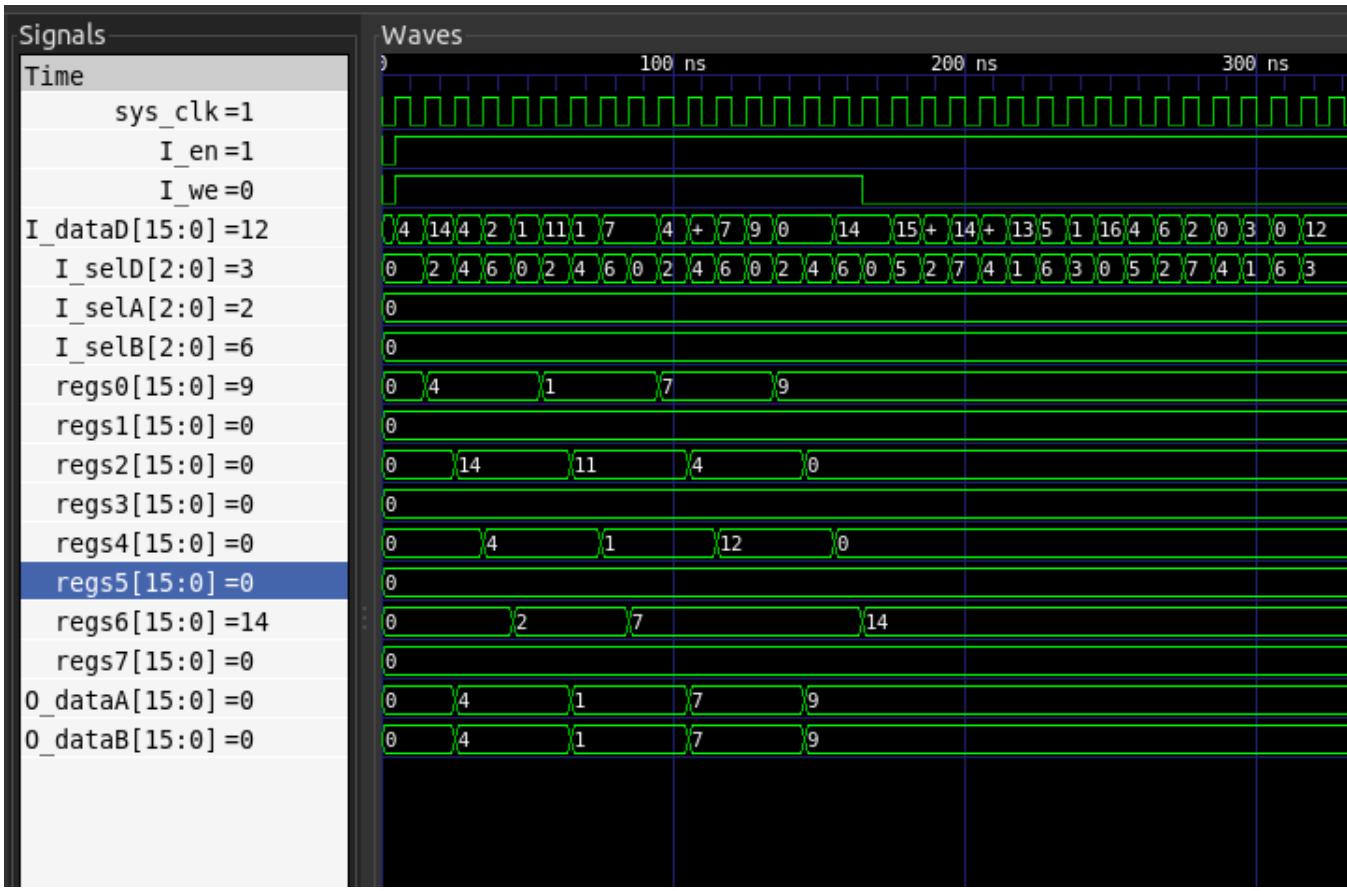

Figure 3.9: Simulation Diagram for an Register FIle.

5. The outputs were read out of the memory location and compared to the initial stored ones.

This operation was iterated for specific instruction opcodes such as ADD, SUB, OR, AND, LOAD, SHL and SHR. Each operation succesfully executed in a clock cycle, maintaining the consistency we have built up with other modules.

The operation of the program counter and the control unit are quite simple and do not really require much testing. The were fed with clock inputs and their outputs monitored. At the end of the tests, they gave the expected response in exactly one clock cycle. This would have meant that we have a throughput of a clock cycle.

The individual modules were connected as shown in Figure 3.10.

Since the testbench is to ba as automated as possible, there became a need for a kind of RAM for storage of instructions. I implemented a scratch pad RAM simply for storage of instructions during initialization of the whole CPU instance.The RAM is 16-bit in width but can take as much as 1024 inputs. This allows for continous operation of the CPU simplifies the test bench to just a series of clock ticks.

Several instructions were saved in the crude RAM of the processor and allow to run until the end. The first set of instructions are as follows:

`0x0902`

`0x0670`

`0x0902` means `LOAD 0x02` into register `0x04`.

`0x0670` means `ADD` the content of Register `0x03` into the content of register `0x04` and store it in register `0x04`.

The result of the testbench as generated by the CPU is as shown in Figure 3.11. A very surprising thing that I noticed while going through the simulation diagram is the fact that it takes 6 clock cycles from feeding the pipeline an instruction and getting its output. This came as a shock as I thought that the processor will be outputing data per clock cycle. After looking through the whole architecture, I found out that the states are controlled by the six bit control unit. This means that if there is need to add a new stage to the pipeline, the module to change is the control unit.

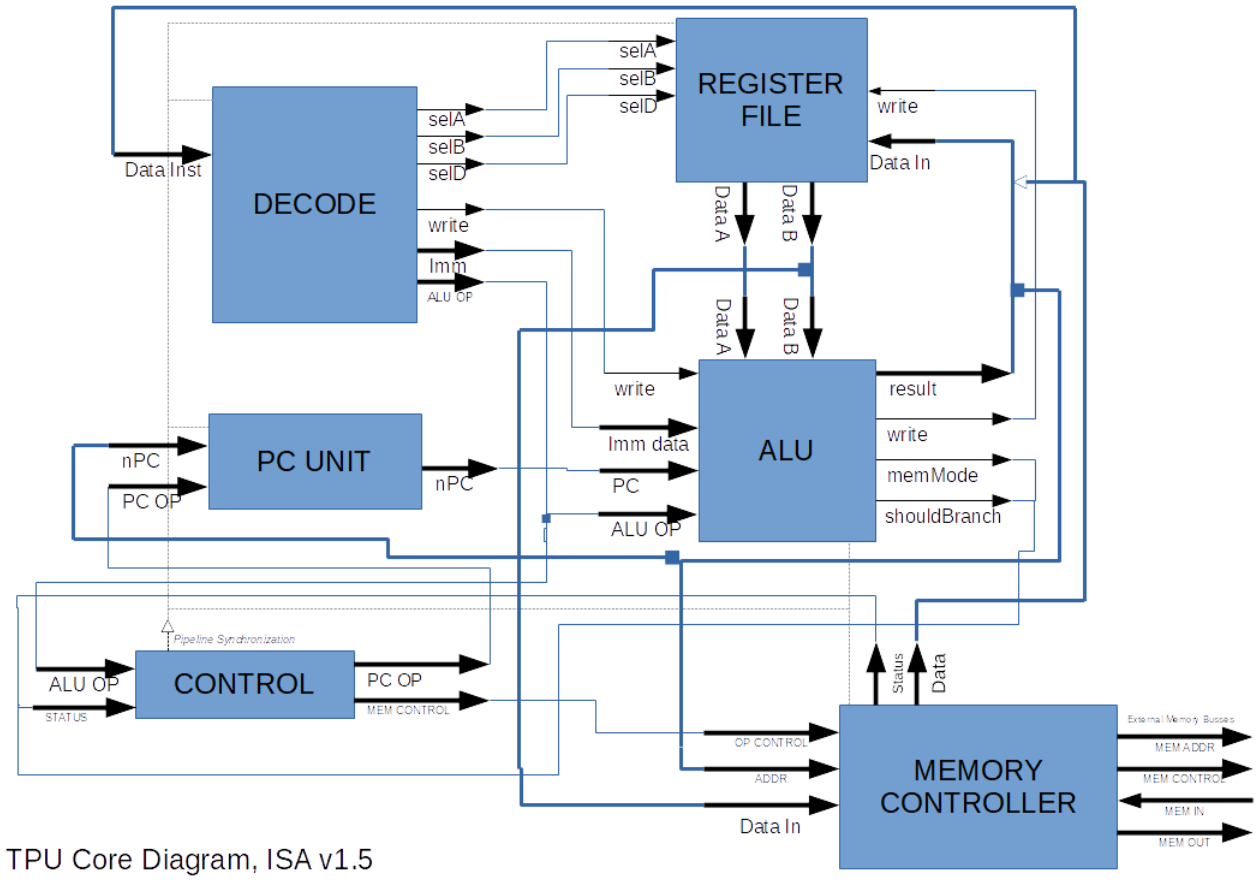

Figure 3.10: Diagram for an overall connection.

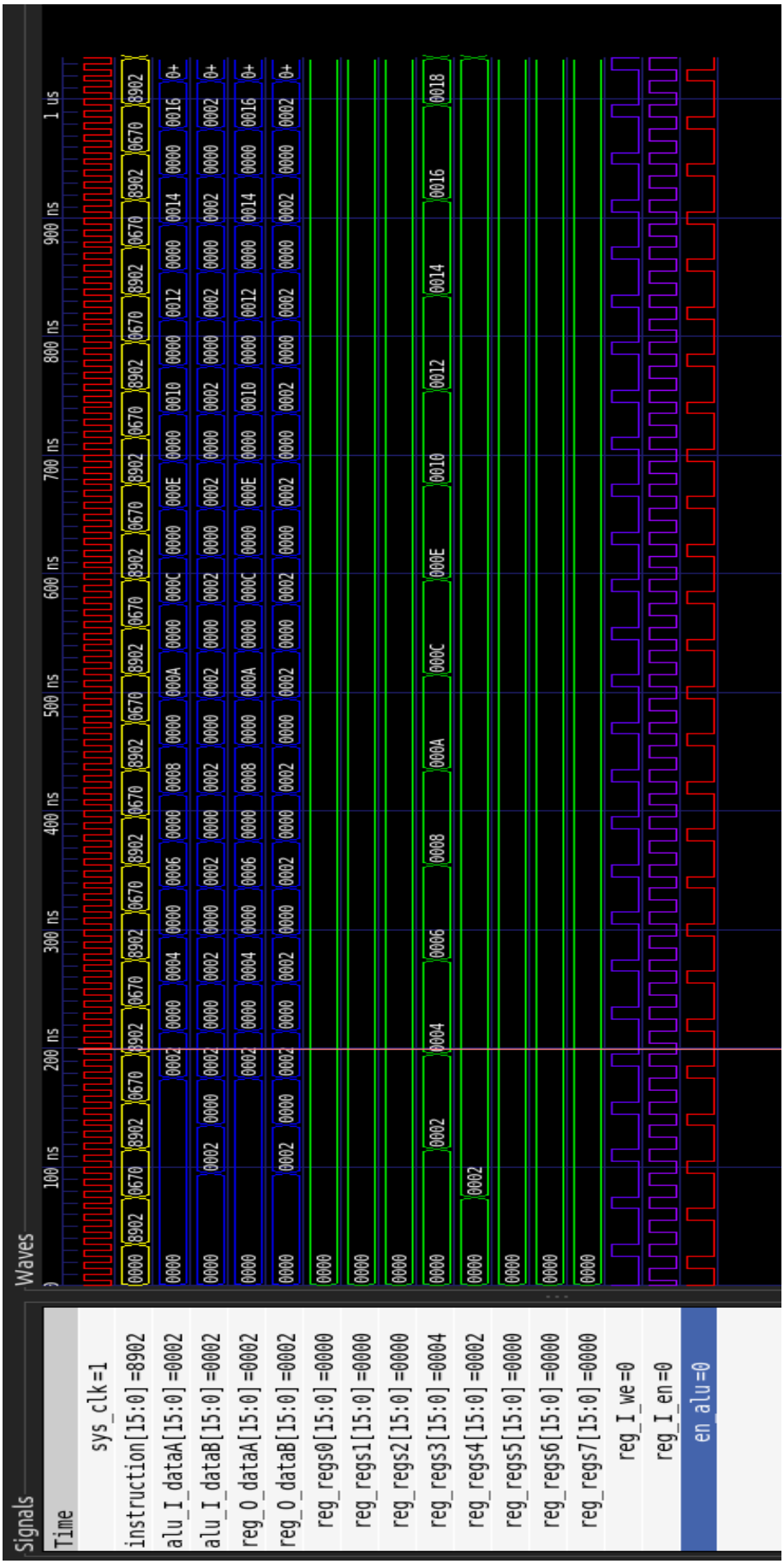

Figure 3.11: Simulation Diagram for MicroProcessor.

Another round of instructions were loaded in the crude RAM. This time, the instructions were as follows:

`0x8902`

`0x0670`

`0x2A0C`

`0x2A0C` means `ADD` the content of Register `0x02` into the content of register `0x05` and store it in register `0x03` while the other instruction mean the same as the previous test

The result of the testbench as generated by the CPU is as shown in Figure 3.12. After this test, I noticed that there were some times when the ALU writes to the Register file without the assertion of the Instruction. This lead to a couple of times when the neccessary writebacks dont occur in time and vital info is lost. After a long time of tracking and debugging the code implementation of the TPU, I solved the problem by always keeping the enable bit of the ALU high and adding a latent clock cycle to the pipeline. This means that the TPU will heavily rely on the efficiency of its other modules to make sure the right data is written at the right time.

A much elaborate test was carried out on the processor with the following instructions:

`0x80FE`

`0x83ED`

`0x2404`

`0x8701`

`0x8902`

`0x0670`

`0x2A0C`

`0xC105`

`0x80FE`

The Figure 3.13 shows the instruction loaded inside the RAM and the PC incrementing as required. The activation bits for each stage of the pipeline can also

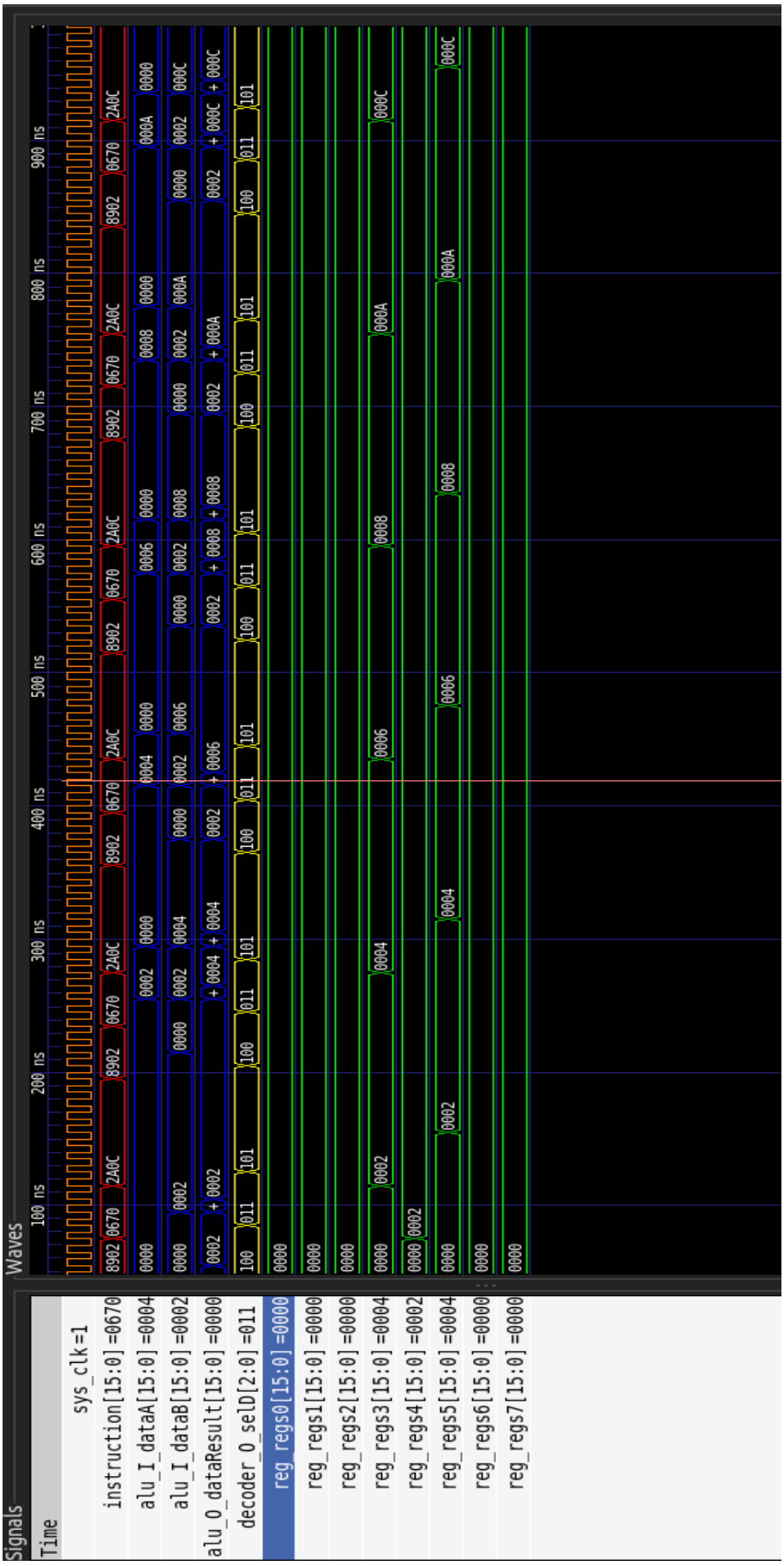

Figure 3.12: Simulation Diagram for MicroProcessor instructions Test2.

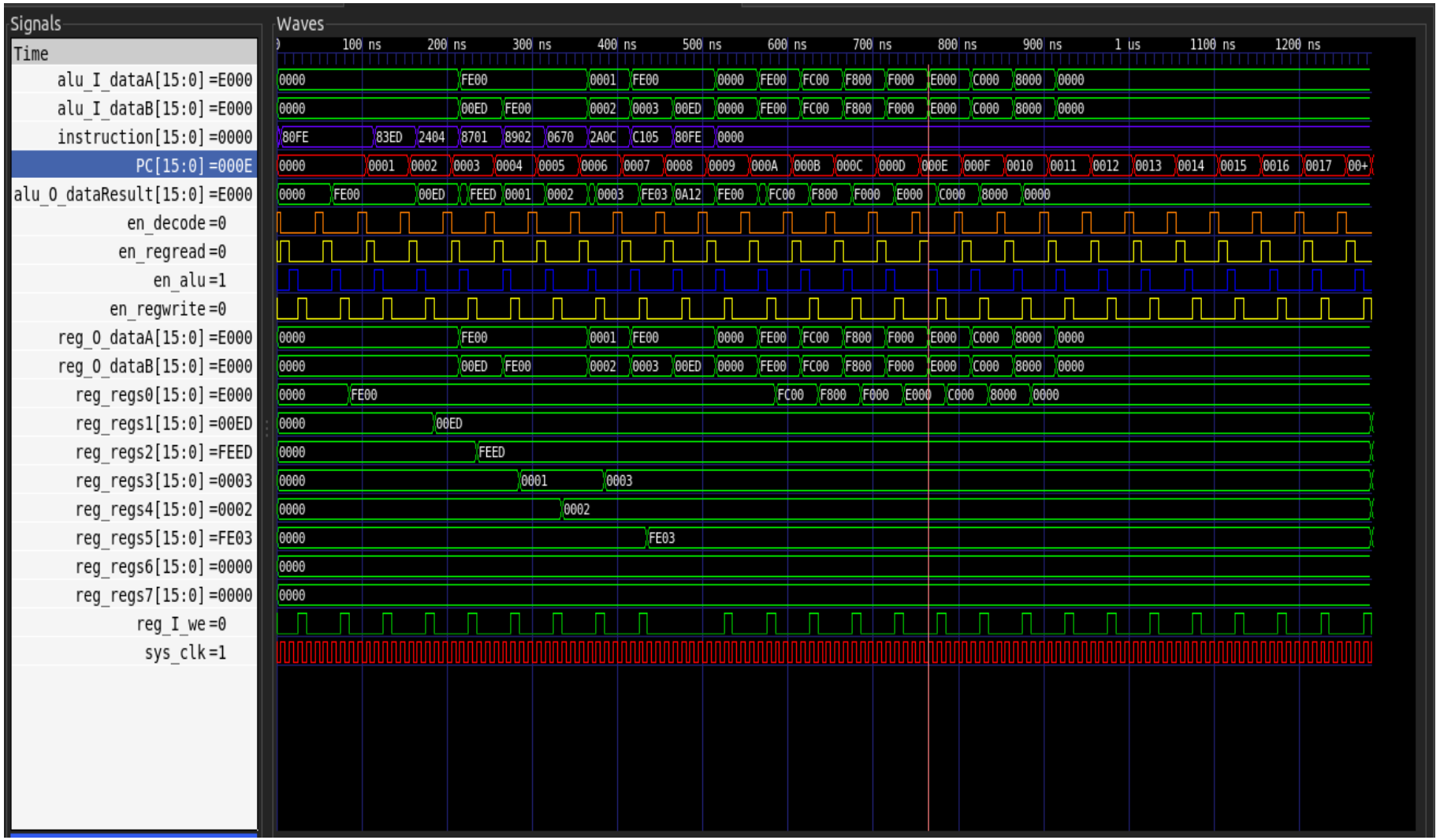

Figure 3.13: Simulation Diagram for MicroProcessor instructions full Test.

be seen. Although there was a minor glitch in the ALU output at some point, the system rectified this before the activation bit for writeback came around.

3.1.8 Testing on an FPGA

After testing the modules with testbenches, I flashed the code unto a DE10-Lite Field Programmable Gate Array(FPGA). I wrote a program to count from one to a thousand, display it on a seven-segment display and light up a LED if the number is prime. This is a considerably tough test for the processor and it is a worthy test of efficiency and validity of most of its commands. The microprocessor, albeit being slow, was able to do this. I also implemented on the FPGA a VGA module and UART module through which the processor communicates with the outside world.

3.2 Building of a Data Concentration Unit

GRIT Engineering Limited makes smart utility meters. The current flagship of the company is the G1. This is a smart meter that takes realtime data from the house's electrical network and transmits it to the internet. It is quite efficient but has a few problems.

One of these problems stem from the fact that it makes use of the popular SIMCOMM's SIM800 module to transmit data to the internet. The availability of network in any particular environment drastically affects the operations of the meter and eventually leads gaps in the data collected from the house. It also defeats the realtime purpose of the smart meter its self. Various network providers have blindspots in their coverage of the entire nation. Altough quite rare, it is possible to have no network coverage in a particular location for hours. The module is also quite expensive as every meter must possess one for it to function as expected.

Also, the G1 has eight channel for current monitoring and no direct voltage monitoring feature. This leads to the need to fix the voltage for the computations carried out by the device. This fixation of the voltage causes an accumulative error in the data collected as it is quite normal for the line voltage to vary from time to

time. The eight channel was also noticed to be quite an overkill as most bulidings do not require more than four to five channels to effectively monitor their useage.

Moreover, the G1 makes use of the popular RaspberryPi Single Board Computer(SBC) for its "heavylifting". The processes that the meter performs are literarily not hard enough to require such complex system as a raspberrypi or any SBC to be frank. Microcontrollers from vendors like STelectronics, Microchip and Espressif have a great reputation for handling these tasks easily and without breaking a sweat while also costing way less than the Raspberrypi. They can even be programmed with Python which is the primary language used in the development of the G1. There is simply no reason to use the raspberrypi anymore.

The need to solve these problems and reduce production cost led to the company's decision to make a new revision of the G1 called the G0. The G0 has the following features.

1. STM32F407VE Microcontroller based design with low power consumption.
2. Three current sensor port and one voltage sensor port
3. Local Radio transciever for transmitting of data to a data pooling device.

These solved all of the problems enumerated above and also reduced the total cost of making one unit of the device. The device had being designed and made before I resumed work as an intern in the company. The only unfinished job was the communication with a data pooling device which is meant to send the data to the internet.

As a member of the hardware team, I, along with another member of the team, was saddled with the responsibility of designing this important piece of device. It was called a Data Concentrator Unit (DCU). The DCU is basically a custom RF gateway device. It has one RF channel and is to serve all G0 boards in its vicinity. It then will relay the data collected to the internet where it can be viewed and used.

3.2.1 Hardware Design

The requirements given to be satisfied by the DCU is that

1. It must be power efficient
2. Serve devices within a radius of 700m
3. Internet connected
4. collect data periodically.

Due to the following requirements, We decided to use the following components

1. **STM32l4 Microcontroller**: The STM32L4 chip is a STElectronic microcontroller that is know for its ability to perform excellently while use=ing very low power. It contains peripherals like Serial peripheral interface (SPI), UART, I2C and a substantial amount of general purpose input/output pins. All these coupled with the fact that it is quite cheap and the company having a substantial amount in stock made the chip the go to for this project. The microcontroller is programmed in C.
2. **SX1735** The need for radio communication between the DCU and the G0 meters led to the need to use this paricular RF chip. The meter possesses one of this chips and it was only natural for the DCU to have one too. The SX1785 is a radio Frequency Tranciever that communicates via the LoRa modulation technique. This allows for long range of communication with substantial reduction in the power required while also staying unaffected by most barriers. The LoRa transciever is very cheap and requires an omni-directional antenna to perform effectively. It communicates with the host microcontroller via SPI with speeds up to 4Mhz and supports interrupts on reception of data packets. This means we can easily attend to recieved packets at anytime during operation.
3. **FerroMagnetic Random Access Memory (FRAM)**: The FRAM is meant to store data recieved from the meters in the case their is no network. Storing the data in the internal flash memory of the microcontroller will only degrade it and make it quite unprdictable in its aged form. The MB16H is a cheap non-volatile memory chip. It has up to Four million write cycles compare with

the ten thousand write cycles of Flash. It uses very low power as it does not need to refresh itself once the data is stored. The chip used in this case was MBR16871. Although it was more expensive than flash memory of the same size, it has faster access time and will last longer in operation.

4. **18650 Li-ion Battery**: These batteries are very important to the operation of the device. It serves as a power backup in case there is loss of electricity during operation. The 18650 battery is a very common battery found in a lot of products and devices. Its volumetric power ratio is higher than most portable batteries can offer while still being the cheapest of them all. Charging a 18650 Li-ion battery has become very simple and can be done without monitoring due to the emergence of overcharging and short-circuitry protection chips. They are quite cheap too.

5. **SIM800 GSM Module**: The need for this device to pipeline or route data collected from various meters to the internet were they are used. SIM800 was a natural choice as the company's webserver has been optimized for recieving requests from them. This is also due to the fact that the 2G network is has quite a wide coverage and is far from being scrapped in a country like Nigeria. It is also quite cheap and has a lot of resources available online and quite easy to get it into operation.

All the components listed above influenced the decision made during the design of the device. The micro-controller is responsible for directing the operation. It collects data from the LoRa Chip and sends it to the internet via the SIM800 chip. It accesses the FRAM via I2C and stores data unto it in the case of inability to send the data to the internet probably due to network unavailability. The schematic diagram is as shown in Figure **XXXXXXX**.

3.3 Design of a Gas Monitoring Device

3.4 PCB Making

3.5 Building of a LoRa Data Concetration Device

Chapter 4

Chapter Four Title

Chapter 5

Conclusion

Appendix A

Appendix Title