



Wave Analysis using FPGAs on Marine Buoys

by

Brian COLGAN

This Report is submitted in partial fulfilment of the requirements of the Honours Degree in Computer and Communications Engineering (DT081) of the Dublin Institute of Technology

May 29, 2017

Academic Supervisor: Dr. Andreas SCHWARZBACHER

Company Supervisor: Mr. Paul DRUM

Declaration

I, the undersigned, declare that this report is entirely my own written work, except where otherwise accredited, and that it has not been submitted for a degree or other award to any other university or institution.

Signed:



Date:

May 29, 2017

Acknowledgements

I would firstly like to thank everyone at CréVinn Teoranta for the opportunity to work with them. In particular I would like to thank principal engineer Mr. Paul Drum for his support throughout this project. I would also like to thank Dr. Andreas Schwarzbacher for his help and direction with this project.

Abstract

In this project an embedded system is developed to analyse wave data. The system developed uses acceleration measurements and performs a Fast Fourier Transform (FFT) on these measurements. A Nexys4-DDR development board will be used throughout the project. The board uses the Artix-7 Field Programmable Gate Array (FPGA) from Xilinx. This is the target device to implement a soft Intellectual Property (IP) core. By using Xilinx MicroBlaze Processor the Universal Asynchronous Receiver/Transmitter (UART) Soft IP Core was designed to send the calculated results of the FFT.

Table of Contents

1	Introduction	1
1.1	Objectives	1
1.2	Ethics	2
1.3	Project Organisation	3
1.3.1	Resources	3
1.3.2	Milestones	3
1.3.3	Plan	3
2	Technical Background	5
2.1	FFT	5
2.2	UART	6
2.3	VHDL	6
2.4	Xilinx Vivado	6
2.5	MicroBlaze	6
2.6	AXI	7
2.7	Xilinx SDK	8
2.8	C Programming Language	8
3	Design Procedure	9
3.1	Development Board	9
3.2	Accelerometer	10
3.3	FFT	11
3.4	UART	11
4	Implementation	13
4.1	Nexys4-DDR Advanced I/O Demo	13
4.2	MicroBlaze	14
4.3	AXI Peripheral	16
4.4	FFT	19
5	Results & Discussion	23

5.1	Nexys4-DDR Advanced I/O Demo	23
5.2	MicroBlaze	24
5.3	AXI Peripheral	24
5.4	FFT	25
6	Conclusion	26
6.1	Future Work	26
	References	30
	Appendices	31
A	C Source Code	31
B	VHDL Accelerometer Code	34
C	VHDL AXI Peripheral Code	40
D	Sinusoids M-file	41
E	Preliminary Report	43

List of Figures

1.1	Gantt Chart	4
2.1	Butterfly Diagram	5
2.2	MicroBlaze Core Block Diagram	7
2.3	AXI channel order and interfaces	8
3.1	System Overview	10
3.2	Accelerometer RTL Block Diagram	10
3.3	Data flow diagram for FFT	11
3.4	Wireless embedded system built around Artix 7 FPGA	11
3.5	Interfacing UART and GSM SIM900A Module	12
4.1	MicroBlaze implementation with UART	15
4.2	Block diagram of custom AXI Peripheral	16
4.3	MicroBlaze implementation with UART and custom AXI peripheral	18
4.4	Measured Wave Data	19
4.5	FFT of Measured Wave Data	19
4.6	Measured Wave vs. Simple Wave	20
4.7	8-points of input sinusoid	20
4.8	Flowchart for C program	22
5.1	Results of Advanced I/O Demo	23
5.2	“Hello World” from MicroBlaze UART shown on terminal	24
5.3	Custom AXI output shown on terminal	24
5.4	FFT Output from compiled source	25
5.5	Plot of FFT Results	25

Listings

4.1	“Hello World” C program	14
4.2	Instantiation of custom AXI peripheral	17
4.3	C program using custom AXI	17
4.4	Generating FFT input data in MATLAB	20
4.5	Structure in C for complex numbers	21
4.6	Example butterfly operations in C	22

Chapter 1

Introduction

The aim of this project is to design and implement a system that can mathematically process wave data and transmit this data using a Field Programmable Gate Array (FPGA).

The Nexys4 DDR board will be used throughout the project. It is a complete, ready-to-use digital circuit development platform based on the latest Artix-7 FPGA from Xilinx [1].

1.1 Objectives

The goal of this project is to determine sea state and water temperature using an FPGA. Wave power in particular is a measurement which is useful information for marine users and wave power generation research. An FPGA will be used for this project as it has many advantages. FPGAs can be programmed using both hardware description languages (HDL) and software programming languages, in this way FPGAs can offer the flexibility of software with the high performance of hardware. Another advantage is reprogrammability and reusability, hardware design faults can be resolved by modifying the HDL code and uploading a new bitstream to the device. This can all be done instantly. The availability of peripherals is also a huge advantage of using FPGAs. This FPGA development board was chosen as it has an accelerometer and thermometer, measurements from both peripherals will be used in this project.

The FPGA must read in acceleration measurements using the on-board three axes accelerometer. This is the information used to calculate magnitude of the acceleration. This displacement data should be mathematically processed to analyse the

wave. The main challenge in this project is to implement a MicroBlaze softcore processor on the FPGA which can read accelerometer measurements and to design a Fast Fourier Transform (FFT) using C on this soft-processor. The FFT results can be used in calculating wave energy.

This information must then be formed into a human-readable message which should outline sea state, weather conditions. An on-board temperature sensor is also used to measure temperature of the water in degrees Celsius. This information should be appended to the message which will be transmitted to the end-user.

1.2 Ethics

The information obtained in this project is particularly useful for two groups within society; marine users and wave power researchers. As such, the ethical requirements of this project can be divided into two parts;

- Relations with colleagues, clients, employers and society in general
- Environmental and social obligations

both of which are set out by Engineers Ireland in its Code of Ethics [2].

With regards to societal relations, the intention of the project is to provide safety information to marine users. In this context, the project is a demonstration of skills and expertise being applied to the common good as well as advancing human welfare with proper regard for the health and welfare of the public. It is important to note that this project is designed to be non-hazardous and have no military applications so as to not result in a serious detriment to any person or persons.

Environmental and social obligations are also a primary motive for this project. As outlined above wave power researchers would, understandably, have significant interest in wave energy estimations. The design of this project is intended to promote the principles and practises of sustainable development and the needs of the present and future generations. The project shall also foster environmental awareness within the profession of engineering among the public.

The reason an Artix-7 FPGA was selected was due to it being the highest performance-per-watt fabric offered by Xilinx [3]. This allows the design to accomplish the objectives outlined in the Introduction with the most efficient consumption of natural resources which is practicable economically. This includes the maximum reduction in energy usage, waste and pollution.

As this FPGA has low power consumption it would also be possible to power the design itself with a renewable energy source e.g. wave energy. Execution of the project in this way eliminates any adverse impact on the natural environment.

1.3 Project Organisation

Time management is vital within a project, as such the project is broken down into phases and then again into tasks. Each task is allotted a predicted amount of time and this allows schedules and deadlines to be made. Each phase and its corresponding tasks are input to a Microsoft Project file. This predicted timeline is displayed as a Gantt chart which allows for accurate tracking of tasks and resources.

1.3.1 Resources

Resources for this project were provided by the company sponsoring the project, CréVinn Teoranta [4], these include materials such as the FPGA as well as a computer to work on throughout the project.

1.3.2 Milestones

Milestones are the major goals within a project, these are outlined below;

- Design specification
- Register-Transfer Level (RTL) Design
- RTL Verification
- Design on FPGA

1.3.3 Plan

The Gantt chart for the project can be seen in Figure 1.1.

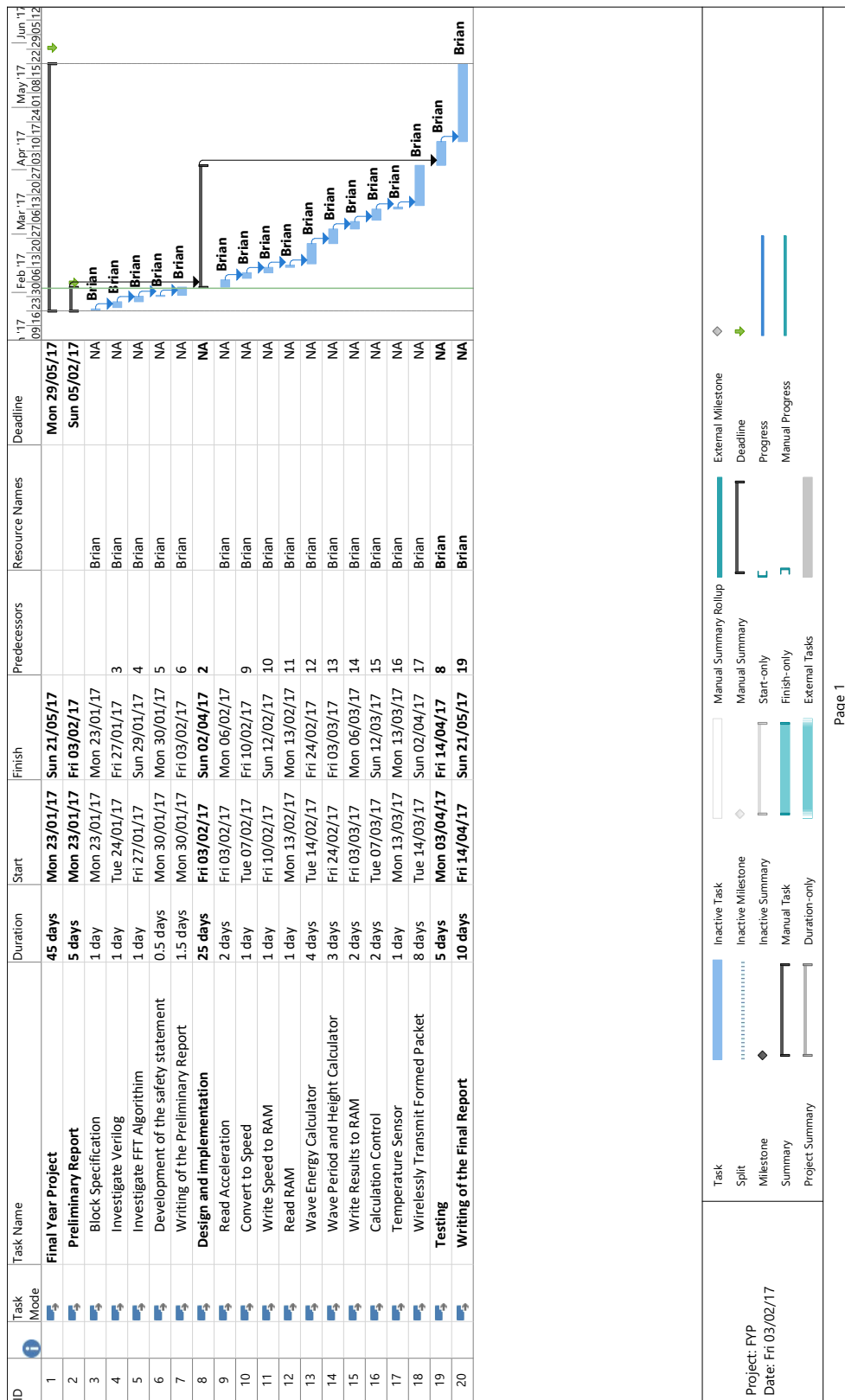


Figure 1.1: Gantt Chart

Chapter 2

Technical Background

In this chapter, the different technologies utilised throughout the project are outlined. These include the different languages, protocols and IDEs used. They will be described in a general way which does not include how they will be used in the project.

2.1 FFT

To process the data the design will perform an FFT by implementing the Cooley-Tukey algorithm, this is the most common form of FFT. It is a recursive algorithm which breaks the Discrete Fourier Transform (DFT) into smaller DFTs. This design will divide the DFT of size $n = 2m$ into two interleaved DFTs of size m , this is known as radix-2.

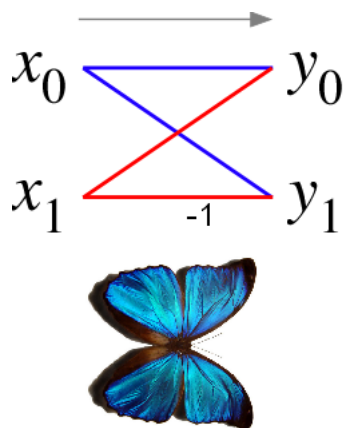


Figure 2.1: Butterfly Diagram [5]

These smaller DFTs are then combined via size-2 DFTs (known as butterflies in this

context, due to the shape of the data flow) pre-multiplied by roots of unity (which are known as twiddle factors). The butterfly operation takes the form:

$$y_0 = x_0 + x_1\omega_n^k \quad (2.1)$$

$$y_1 = x_0 - x_1\omega_n^k \quad (2.2)$$

The above form is known as Decimation In Time (DIT). However, one can have a process where the butterflies come first and are post-multiplied by the twiddle factors, which is known as Decimation In Frequency (DIF).

2.2 UART

A Universal Asynchronous Receiver/Transmitter (UART) is used for asynchronous serial input/output. Its most common protocol is RS-232. In this, the sender and receiver agree on a baud rate, bits per byte, parity and framing bits. UART is very useful for debugging and communicating with other computer systems [6].

2.3 VHDL

Very High Speed Integrated Circuit Hardware Description Language (VHDL) was defined for use in the design and documentation of electronics systems [7]. VHDL allows designers to design at various levels of abstraction when considering the application of VHDL to FPGA design, it is helpful to identify register transfer level (RTL) as one of the levels of abstraction. RTL is the input to synthesis [8].

2.4 Xilinx Vivado

The Vivado Design Suite is a software suite designed by Xilinx to increase the overall productivity for designing, integrating, and implementing systems using Xilinx devices, including the Xilinx 7 series [9]. The Vivado Design Suite replaced the existing Xilinx ISE Design Suite of tools in 2013 [10].

2.5 MicroBlaze

The MicroBlaze Soft Processor Core developed by Xilinx is a key element of their product portfolio [11]. MicroBlaze will implement a microprocessor entirely within

the Xilinx FPGA general purpose memory and logic fabric [12]. It is a highly configurable and easy to use processor and is included free with Xilinx Vivado. In terms of its instruction set architecture, it is an FPGA optimized 32-bit Reduced Instruction Set Computer (RISC) soft processor. Figure 2.2 shows a block diagram of the MicroBlaze core.

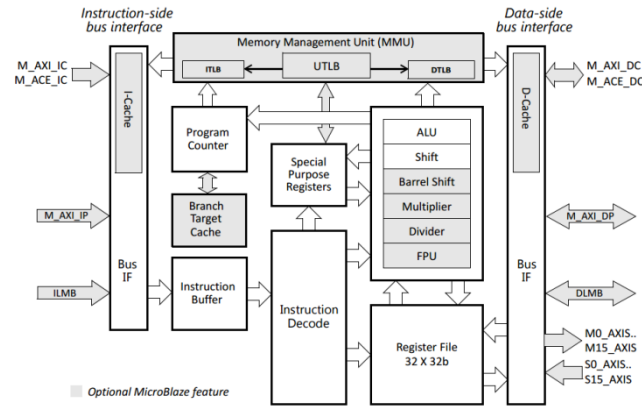


Figure 2.2: MicroBlaze Core Block Diagram [13]

2.6 AXI

The MicroBlaze supports the use Advanced eXtensible Interface (AXI) peripherals. AXI is a bus protocol which falls under the Advanced Microcontroller Bus Architecture (AMBA) specification. AXI is targeted at high performance, high clock frequency system designs and it supports the following [14];

- separate address/control and data phases
- unaligned data transfers using byte strobes
- burst-based transactions with only start address issued
- separate read and write data channels to enable low-cost DMA
- ability to issue multiple outstanding addresses
- out-of-order transaction completion
- easy addition of register stages to provide timing closure

An illustration of how the AXI protocol is intended to be implemented is shown in Figure 2.3.

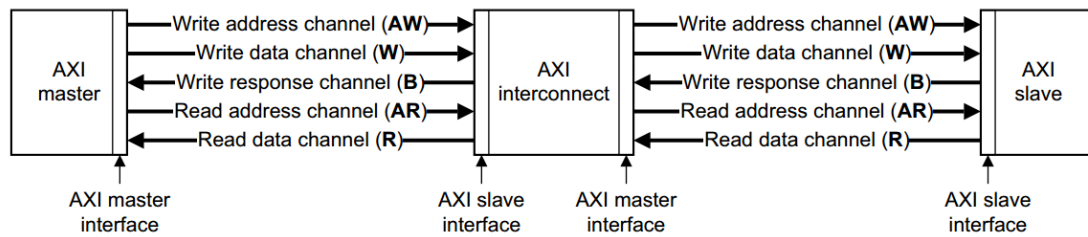


Figure 2.3: AXI channel order and interfaces [14]

2.7 Xilinx SDK

The Xilinx Software Development Kit (SDK) is the Integrated Design Environment (IDE) for creating embedded systems on the MicroBlaze. It enables programmers to write, compile, and debug C (or C++) applications for their embedded system. This software can be tested by downloading the code directly to the FPGA and executing it on the actual system. It is included free with Xilinx Vivado and is based on Eclipse 4.5.0. Like any SDK, it also contains a full suite of libraries and sample code [15].

2.8 C Programming Language

C is a general-purpose programming language that was originally designed in 1972. It was designed for writing system software and it was the language the UNIX operating system was written in. Its use has now spread way beyond that field, such as embedded systems [16].

Chapter 3

Design Procedure

In this chapter, the design of the system will be discussed. This includes how the technologies from the previous chapter will be implemented to achieve the overall project objectives.

3.1 Development Board

The Nexys4 DDR board has generous external memories and several built-in peripherals, including an accelerometer. This allows the Nexys4 DDR to be used for a wide range of designs without needing any other components.

The Nexys4 DDR is an updated version of the Nexys4 board. The major improvement is the replacement of the 16 MiB CellularRAM with a 128 MiB DDR2 SDRAM memory. This new RAM can be accessed at up to 1334 MB/s, this is particularly useful for embedded processor-based designs which would perform poorly with slow asynchronous memory access [17].

The board also includes an Analog Devices ADXL362 accelerometer. The ADXL362 is an ultralow power, 3-axis Microelectromechanical systems (MEMS) accelerometer that consumes less than 2 μ A at a 100Hz output data rate and 270nA when in motion triggered wake-up mode. In addition to its ultralow power consumption, the ADXL362 has many features to enable true system level power reduction. It includes a built-in micropower temperature sensor.

These features are what makes the board very suitable for the specified application. An overview of the proposed design of system can be seen in Figure 3.1:

For this design, the Core is the MicroBlaze. The accelerometer and temperature sensor are inputs to the system and calculations will be output to a transmitter.

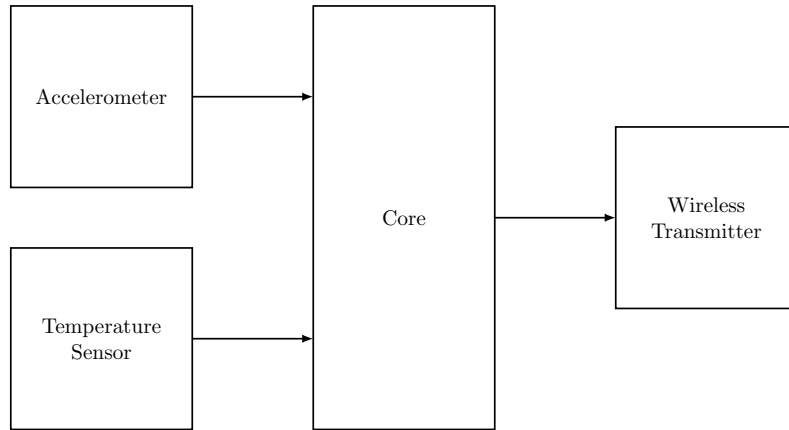


Figure 3.1: System Overview

3.2 Accelerometer

An RTL block diagram of the Nexys 4 DDR's on-board accelerometer can be seen in Figure 3.2. It has four outputs which are of interest to this project, they are;

- ACCEL_X – Acceleration in the x-axis
- ACCEL_Y – Acceleration in the y-axis
- ACCEL_Z – Acceleration in the z-axis
- ACCEL_TMP – Temperature read in by the thermometer

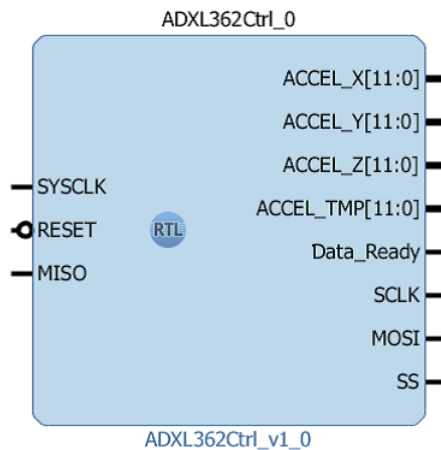


Figure 3.2: Accelerometer RTL Block Diagram

To connect this RTL module to the MicroBlaze, it must be implemented as an AXI peripheral.

3.3 FFT

The below diagram shows the data-flow for an eight-point decimation-in-time radix-2 FFT. This is the type of FFT which will be implemented on the MicroBlaze in the C programming language. A data flow diagram for this form of FFT can be seen in Figure 3.3.

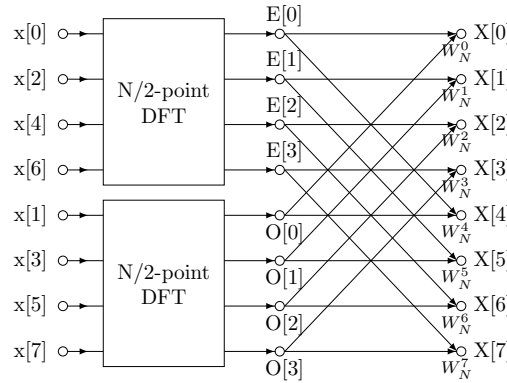


Figure 3.3: Data flow diagram for FFT

The testing of the FFT will be done by entering a sinusoid as stimulus to the design. The output of this system will be measured against expected results. Wave data will also be obtained to measure against pre-determined results.

3.4 UART

The design will have a Xilinx MicroBlaze soft-processor with a Universal Asynchronous Receiver/Transmitter (UART) port which can be used to send out the resultant FFT. The UART can be used to send Attention (AT) commands (from the Hayes command set) and drive a Global System for Mobile Communications (GSM) module. This type of design has been used in many different applications [18, 19]. A block diagram of an example design can be seen in Figure 3.4.

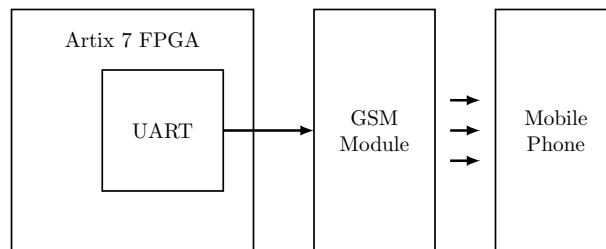


Figure 3.4: Wireless embedded system built around Artix 7 FPGA

Data will be transmitted using the RS-232 standard for serial communication. This interface can be seen in Figure 3.5

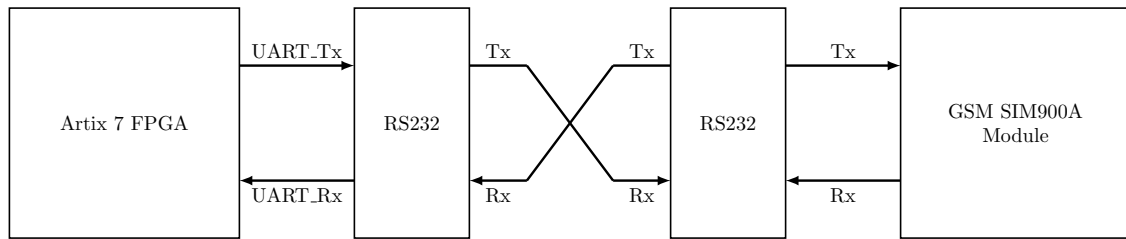


Figure 3.5: Interfacing UART and GSM SIM900A Module

Chapter 4

Implementation

This chapter will discuss the technical elements of implementing the proposed systems and issues that occurred in the process.

4.1 Nexys4-DDR Advanced I/O Demo

To ensure that all modules and components of the FPGA were functioning correctly, the *Nexys4-DDR Advanced I/O Demo* was the first thing that was loaded onto the FPGA. This was done using Vivado Design Suite. This demo intends to emulate the functionality of the built-in self-test which usually comes pre-loaded onto the FPGA, however the board being used had been previously flashed.

This demo will demonstrate usage of the ADXL362 Accelerometer and display results using the on-board Video Graphics Array (VGA) display in 1280 x 1024 mode. The behaviour of this is as follows. The demo connects to the VGA display in a 1280 x 1024 resolution and displays various items on the screen. The item of interest here is the small square representing the X and Y acceleration data from the ADXL362 onboard accelerometer. The square moves according the board's position [20].

The accelerometer display also displays the acceleration magnitude. This is calculated using the equation below.

$$Accel_{MAG} = \sqrt{x^2 + y^2 + z^2} \quad (4.1)$$

where x , y and z represent the acceleration value on the respective axes. The demo also shows the accelerometer temperature value.

This demo required some debugging as the version downloaded from GitHub did not work “out of the box”. However, changing default I/O standards on some of the

I/O ports allowed the implementation to be run within Vivado. The VHDL code for the accelerometer control module can be seen in Appendix B.

To ensure the z-axis was working correctly, the RTL module was modified so that the small square's position represent acceleration data in the X and Z axes as opposed to X and Y.

4.2 MicroBlaze

The first prerequisite in getting a MicroBlaze running on the FPGA is installing the board file for the board which will be used (in this case, the Nexys 4 DDR) [21]. A board file is essentially an Extensible Markup Language (XML) file contained within the Vivado installation directory. It describes the component parts of the board itself and gives each part a respective name and index. The specified board can then be selected when starting a new project so that everything is configured correctly.

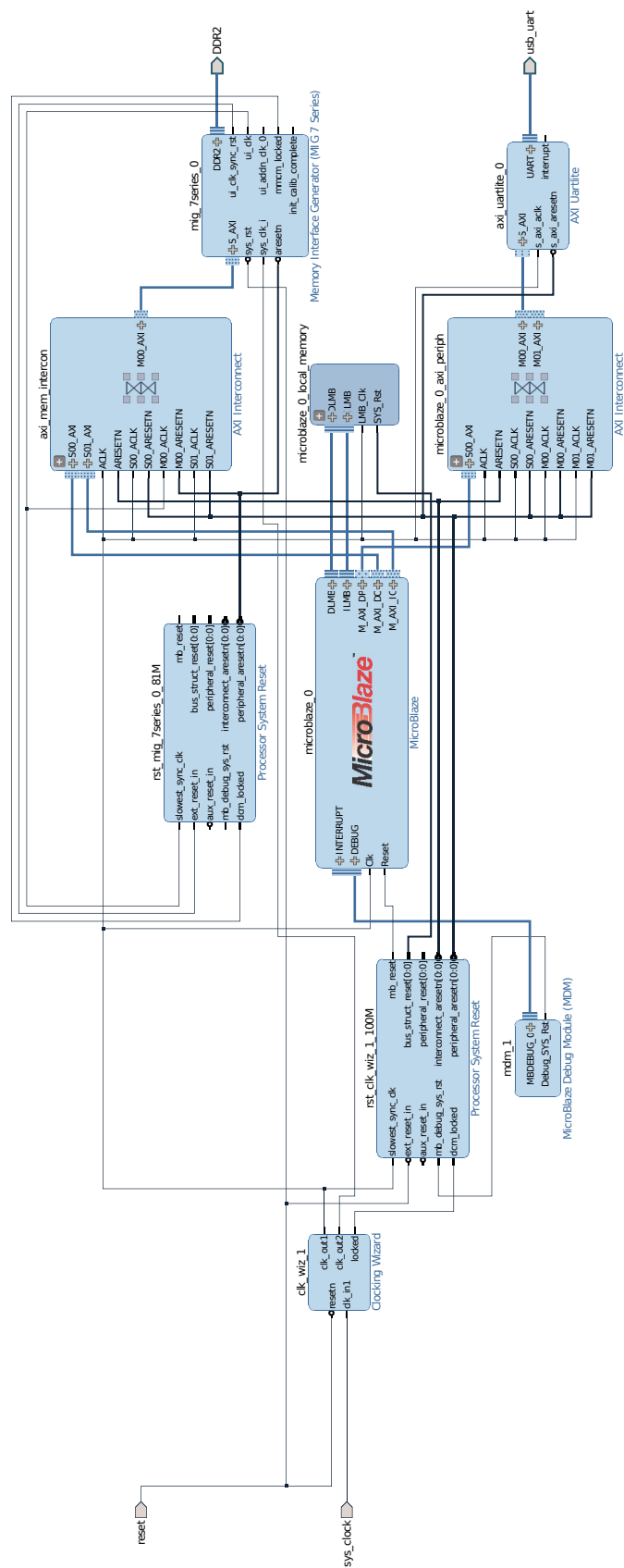
The Preliminary Report in the Appendices outlines two of the goals as having an FFT in hardware and having a UART on a MicroBlaze. These objectives were modified to incorporate the implementation of the FFT on the MicroBlaze also. This implementation of the MicroBlaze allows for output from a C program to go out over UART.

The result of implementing the MicroBlaze as above is the block design in Figure 4.1. In this design the UART IP block, AXI Uartlite, is connected to the AXI Interconnect. However, this implementation does not interface with the accelerometer module.

Once a bitstream is generated for the design, the project can be exported to Xilinx SDK. From here, a C program can be written on and UART output can be examined using the `print()` function. This can be seen in Listing 4.1.

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
int main() {
    init_platform();
    print("Hello_World\n\r");
    cleanup_platform();
    return 0;
}
```

Listing 4.1: “Hello World” C program



4.3 AXI Peripheral

The next step after creating a project with a MicroBlaze was to have it interact with the accelerometer module. This proved to be most difficult task of the project. To do this, a custom IP block needed to be created. This would be an AXI peripheral containing the RTL accelerometer module, which would then be connected to the AXI Interconnect. Tutorials for creating a custom IP block like this using VHDL can be found online [22].

The referenced tutorial describes the process of implementing a multiplier in VHDL on a MicroZed board. The boards are part of Xilinx's Zynq product family, as opposed to using soft-processors Zynq boards have both an FPGA fabric and an ARM processor. The first steps in this tutorial are common to that of a tutorial from Xilinx for creating an AXI peripheral [23].

The VHDL multiplier from *FPGA Developer* was used as a basis to get a working example of a custom AXI peripheral. This code can be seen in Appendix C.

As can be seen from Figure 4.3, there is now a second AXI peripheral connected to the AXI Interconnect, *my_multiplier_v1.0*. This peripheral allows two values to be written to memory from a C program in Xilinx SDK and then multiplies them together and stores the result in another location in memory.

Once this was achieved, a similar procedure was followed to implement the accelerometer module as an AXI peripheral. The intention of this peripheral was to store accelerometer magnitude values in memory, these values would then be used as input to the FFT in C. Unfortunately, this goal was not achieved as a result of an inability to run the implementation of the design which contained the accelerometer AXI peripheral.

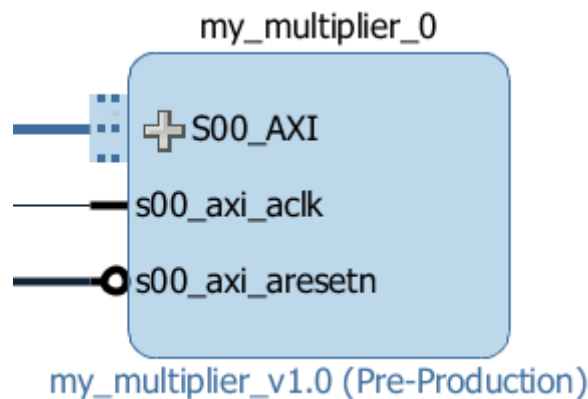


Figure 4.2: Block diagram of custom AXI Peripheral

The peripheral in Figure 4.2 can be seen as part of the full system in Figure 4.3. This multiplier module was instantiated with the VHDL in Listing 4.2.

```
multiplier_0 : multiplier
port map (
    clk => S_AXI_ACLK,
    a => slv_reg0(31 downto 16),
    b => slv_reg0(15 downto 0),
    p => multiplier_out);
```

Listing 4.2: Instantiation of custom AXI peripheral

The custom AXI peripheral was then tested from SDK with the code in Listing 4.4.

```
Xuint32 *baseaddr_p = (Xuint32*)XPAR_MY_MULTIPLIER_0_S00_AXI_BASEADDR;
int main()
{
    init_platform();
    xil_printf("Multiplier_Test\n\r");
    // Write multiplier inputs to register 0
    *(baseaddr_p+0) = 0x00020003;
    xil_printf("Wrote:_0x%08x_\n\r", *(baseaddr_p+0));
    // Read multiplier output from register 1
    xil_printf("Read:_0x%08x_\n\r", *(baseaddr_p+1));
    xil_printf("End_of_test\n\n\r");
    return 0;
}
```

Listing 4.3: C program using custom AXI

4.4 FFT

Although the accelerometer magnitude could not be used as input to the FFT due to the above issue, an FFT could still be implemented on the MicroBlaze in C. Instead of using accelerometer measurements from the board, a sample sinusoid would be used as an input to the FFT. Sample wave data was acquired from a project which also used accelerometers on a buoy [24]. This data was imported into MATLAB to produce the plot seen in Figure 4.4.

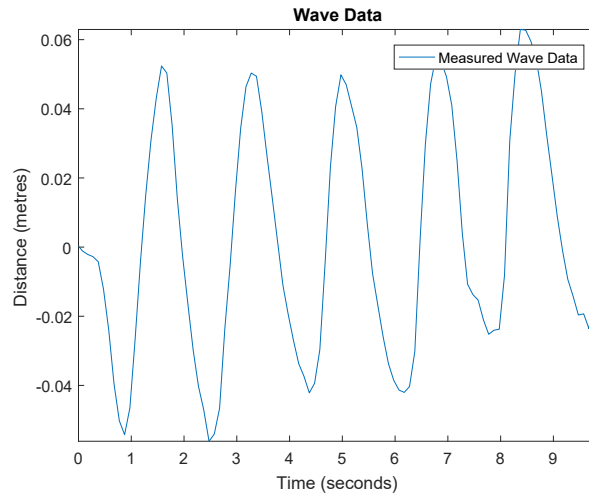


Figure 4.4: Measured Wave Data

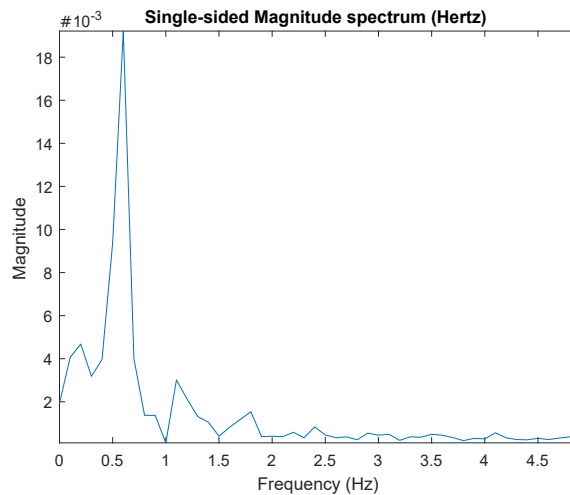


Figure 4.5: FFT of Measured Wave Data

To confirm the frequency of the sample data, an FFT was performed in MATLAB. The resultant frequency is what will be used for the sinusoid, in order to replicate real wave data. The FFT of the measurements can be seen in Figure 4.5.

The results show that the waves have a frequency of 0.6Hz and an amplitude of 0.05m. This wave was approximated in MATLAB with a single sinusoid. Plots of the measured data and simple sinusoid can be seen in Figure 4.6.

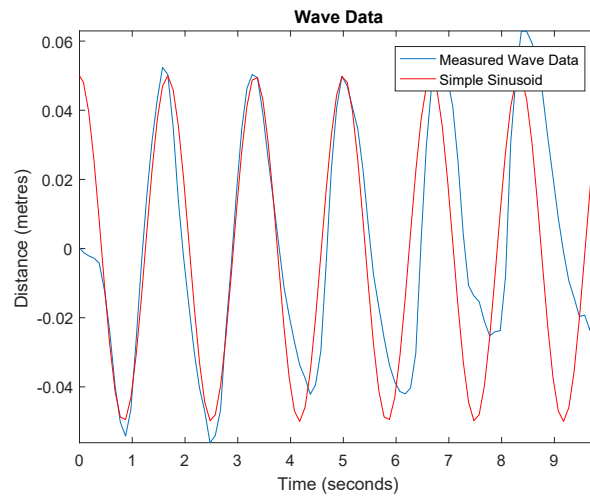


Figure 4.6: Measured Wave vs. Simple Wave

It was decided to do an eight-point FFT. As such, the input to the FFT to be done in C can be seen in Figure 4.7. The commented M-file used for the generation and plotting of these sinusoids is shown in Appendix D.

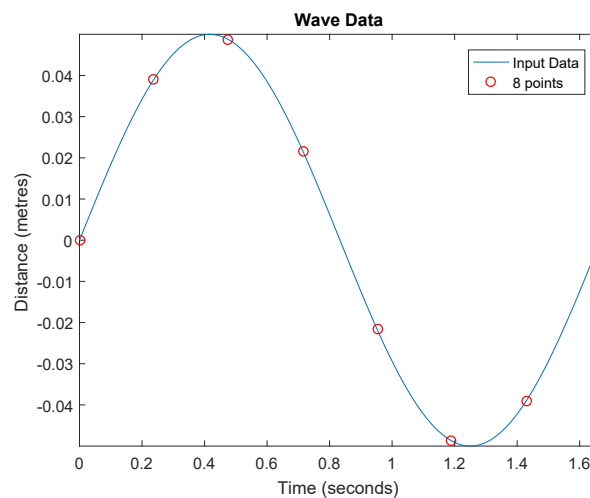


Figure 4.7: 8-points of input sinusoid

The input values for the C program were also generated in MATLAB. These include values for the input signal (x), the twiddle factors (tw), and the correct results (X).

```
% Generate input variables for C program
% x (input signal), tw (twiddle factors)
t = linspace(0, (1/0.6), 8);
```

```

x = 0.05*sin(2*pi*0.6*time);
X = fft(x);
for k = 0:3
    tw(k+1) = (exp(1))^( (-j*2*pi*k)/8);
end

```

Listing 4.4: Generating FFT input data in MATLAB

The values for x were stored in a structure which contained real and imaginary numbers. This is in order to allow both real and imaginary numbers to be multiplied by the twiddle factors. The structure in which the above MATLAB values are stored is shown in Listing 4.5. The output from C program is compared to the theoretical results which are also stored in an array.

```

//Structure to hold real and imaginary numbers
struct complex
{
    float real, img;
};
//Twiddle factors, input data, and expected results
//(calculated in MATLAB)
struct complex tw[4] = {{1,0},{0.707,-0.707},{0,-1},{-0.707,-0.707}};
struct complex data[8] = {{0,0},{0.0391,0},{0.0487,0},{0.0217,0},
                          {-0.0217,0},{-0.0487,0},{-0.0391,0},{0,0}};
struct complex gold[8] = {{0.0000,0.0000},{0.0685,-0.1653},
                          {-0.0313,0.0313},{-0.0251,0.0104},
                          {-0.0241,0.0000},{-0.0251,-0.0104},
                          {-0.0313,-0.0313},{0.0685,0.1653}};

```

Listing 4.5: Structure in C for complex numbers

As previously mentioned, the implementation will be a radix-2 Cooley-Tukey algorithm. As the input is eight samples, the amount of radix-2 stages is equal to;

$$\log_2(8) = 3 \text{ stages} \quad (4.2)$$

Each stage in the C program will be a function containing the butterfly operations on each of the input samples and the shuffling of the elements in the input array. The output of stage one will be the input to the stage two function, stage two's output is also the input the final third stage. Mathematically, the butterflies are of the form;

$$y_0 = x_0 + x_1\omega_n^k \quad (4.3)$$

$$y_1 = x_0 - x_1\omega_n^k \quad (4.4)$$

In C, this can be written as shown in Listing 4.6.

```

stage2[5].real = (s1[5].real + s1[7].real)*tw[1].real - (s1[5].img +
    s1[7].img)*tw[1].img;
stage2[5].img = (s1[5].real + s1[7].real)*tw[1].img + (s1[5].img + s1
    [7].img)*tw[1].real;
stage2[6].real = (s1[4].real - s1[6].real)*tw[2].real - (s1[4].img -
    s1[6].img)*tw[2].img;
stage2[6].img = (s1[4].real - s1[6].real)*tw[2].img + (s1[4].img -
    s1[6].img)*tw[2].real;

```

Listing 4.6: Example butterfly operations in C

The source code for the full C program can be seen in Appendix A. A flowchart outlining the functionality of the C program is shown in Figure 4.8.

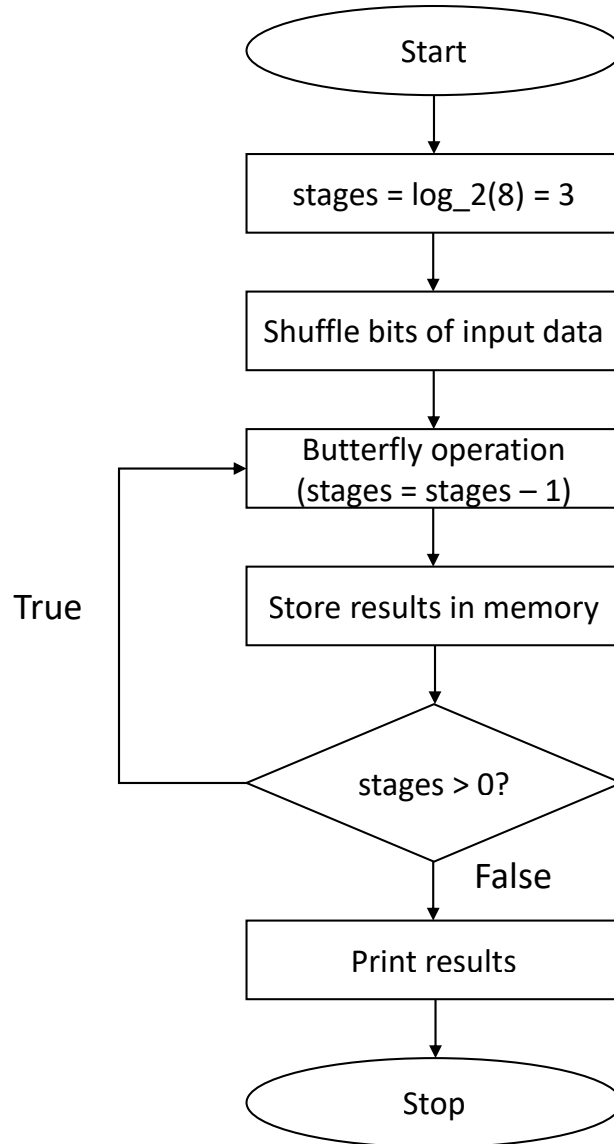


Figure 4.8: Flowchart for C program

Chapter 5

Results & Discussion

In this chapter, the results of the implementation will be discussed. These results will show the various outputs from the development board.

5.1 Nexys4-DDR Advanced I/O Demo

Figure 5.1 shows the VGA output from the Advance I/O Demo. There is a separate window shown for each measurement device. The accelerometer windows shows a green box move in its respective axis as the board moves. The temperature sensor on the accelerometer also shows the boards temperature. This demo also shows other peripherals available on the device (e.g. microphone and LED colours). From running and debugging this demo, it was confirmed that the sensors that were required throughout the project are working correctly.

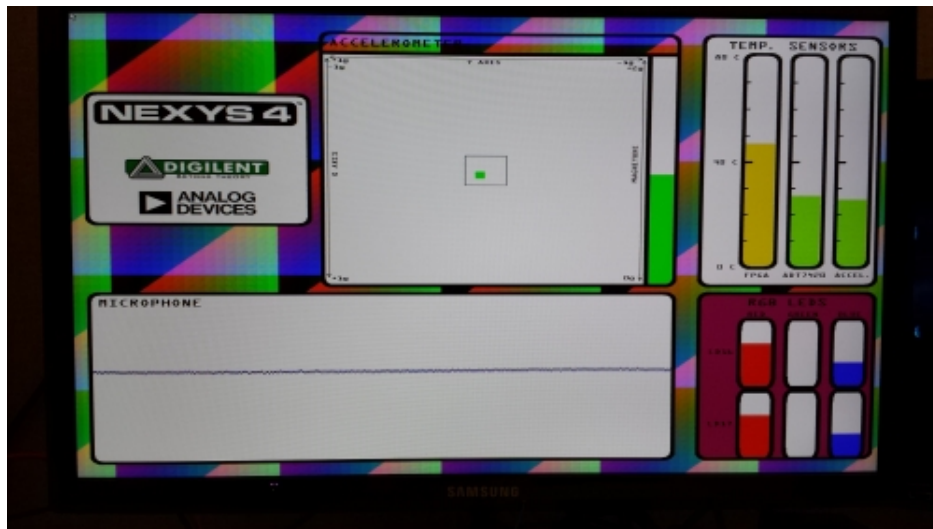


Figure 5.1: Results of Advanced I/O Demo

Once this functionality was achieved, work began on getting the MicroBlaze working on the FPGA. This began with getting a UART AXI peripheral working first, followed by implementing a custom AXI peripheral. The results of this are outlined below.

5.2 MicroBlaze

The string “Hello World” was sent from the MicroBlaze. This was monitored and tested using Tera Term on a personal computer. The output from the UART was sent out via USB to Communication port 4 (COM4). Figure 5.2 shows the screenshot of Tera Term displaying the message sent from the UART.

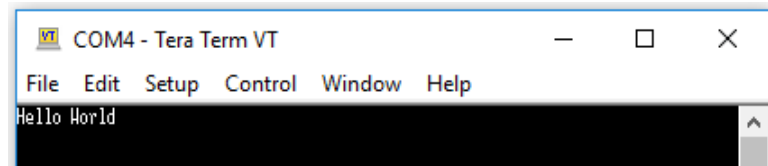


Figure 5.2: “Hello World” from MicroBlaze UART shown on terminal

5.3 AXI Peripheral

Figure 5.3 shows the AXI peripheral functioning correctly. The value 0x00090009 is written. The module then separates this into two separate values. The first product, a , is equal to bits 31 down to 16. The second product, b , is equal to bits 15 down to 0. The result, p , is in hexadecimal.

$$a \times b = p \quad (5.1)$$

$$9_{16} \times 9_{16} = 51_{16} \quad (5.2)$$

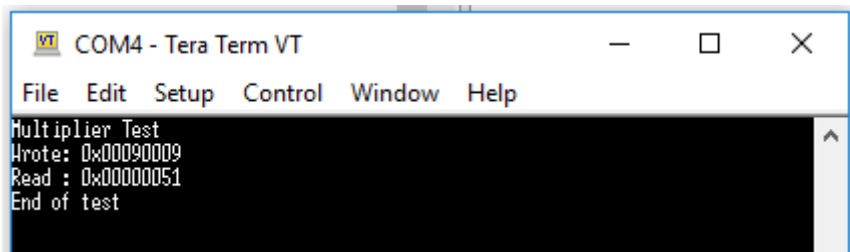


Figure 5.3: Custom AXI output shown on terminal

5.4 FFT

Figure 5.4 shows the input, output, and expected results of the FFT. The output values match the expected ones calculated by MATLAB. The absolute values of these results represent the magnitude spectrum of the wave data.

```
bcolgan@proxtwo6:~/fyp$ ./proj.o
FFT INPUT
x(0) = (0.000 0.000i)
x(1) = (0.039 0.000i)
x(2) = (0.049 0.000i)
x(3) = (0.022 0.000i)
x(4) = (-0.022 0.000i)
x(5) = (-0.049 0.000i)
x(6) = (-0.039 0.000i)
x(7) = (0.000 0.000i)
FFT OUTPUT
X(0) = (0.000 0.000i)
X(1) = (0.068 -0.165i)
X(2) = (-0.031 0.031i)
X(3) = (-0.025 0.010i)
X(4) = (-0.024 0.000i)
X(5) = (-0.025 -0.010i)
X(6) = (-0.031 -0.031i)
X(7) = (0.068 0.165i)
Expected results
X(0) = (0.000 0.000i)
X(1) = (0.068 -0.165i)
X(2) = (-0.031 0.031i)
X(3) = (-0.025 0.010i)
X(4) = (-0.024 0.000i)
X(5) = (-0.025 -0.010i)
X(6) = (-0.031 -0.031i)
X(7) = (0.068 0.165i)
PASS: FFT outputs correct results
```

Figure 5.4: FFT Output from compiled source

These results were plotted in MATLAB to show the accuracy of the results. It can be seen from Figure 5.6 that the wave period is calculated as 0.6Hz and the amplitude is calculated as 0.0447 metres. This frequency is correct but the amplitude is off by roughly 10%.

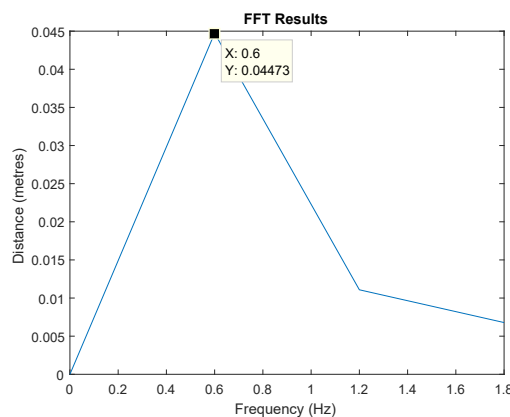


Figure 5.5: Plot of FFT Results

Chapter 6

Conclusion

The core purpose of this project was to demonstrate how FPGA technology can be used to analyse waves. This was achieved by implementing a softcore MicroBlaze processor on the FPGA which allowed an FFT to be written in C on the device. The FFT was performed on sample accelerometer measurements using the radix-2 DIT form of the Cooley-Tukey algorithm. This result was then verified against theoretical results using MATLAB. The accelerometer on the development board was tested along with the temperature sensor. An AXI peripheral was designed to show how the accelerometer could interface with the MicroBlaze processor. The MicroBlaze had a UART interface which allowed for the results to be sent from the C program over USB. In doing this, many skills were developed. This includes learning how to use some new IDEs as well as getting exposure to HDL.

6.1 Future Work

Although many of the objectives were met, further development can be done in this area to improve performance. The below roadmap shows how this may be done.

- A more accurate FFT could be used to improve the results, this would involve changing the C code to support a higher amount of points. (E.g. increase from 8-points to 32-point or 64-point). This would allow acceleration measurements to be acquired over a longer period and thus provide more accurate information on sea state.
- Integrate GSM onto the FPGA. As the FPGA already outputs via UART, it would be possible to integrate a GSM module with a SIM card to the board. This would provide the functionality of sending an SMS with calculated

information to a mobile device or third-party service, like Twitter.

- Waterproofing and providing external power to the device would allow for it to be put out on marine buoys.
- If this design was used on many devices, it would be cost effective to implement it on an application-specific integrated circuit (ASIC) as opposed to an FPGA.

References

- [1] Digilent. Nexys4 DDRTM FPGA Board Reference Manual. [Accessed 03 02 2017]. [Online]. Available: https://reference.digilentinc.com/_media/nexys4-ddr:nexys4ddr_rm.pdf
- [2] Engineers Ireland. Code of Ethics. [Accessed 03 02 2017]. [Online]. Available: <http://www.engineersireland.ie/EngineersIreland/media/SiteMedia/about/Engineers-Ireland-Code-Of-Ethics-2010.pdf>
- [3] Xilinx. Artix-7 FPGA Family. [Accessed 03 02 2017]. [Online]. Available: <https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html>
- [4] Cr Vinn Teoranta. About us | Cr Vinn. [Accessed 05 2017]. [Online]. Available: <http://crevinn.com/about-us/>
- [5] S. G. Johnson. File:Butterfly-FFT.png. [Accessed 03 02 2017]. [Online]. Available: <https://commons.wikimedia.org/wiki/File:Butterfly-FFT.png>
- [6] DIT School of Electrical and Electronic Engineering. Section 2: UART, I2C and SPI interfacing. [Accessed 05 2017].
- [7] IEEE. VHDL Language Reference Manual. [Accessed 05 2017]. [Online]. Available: <http://ieeexplore.ieee.org/document/5967868/>
- [8] Doulos. Levels of Abstraction. [Accessed 05 2017]. [Online]. Available: https://www.doulos.com/knowhow/vhdl_designers_guide/levels_of_abstraction/
- [9] Xilinx. Vivado Design Suite User Guide. [Accessed 05 2017]. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug910-vivado-getting-started.pdf
- [10] ——. ISE Design - 14.7 Full Product Installation. [Accessed 05 2017]. [Online]. Available: <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html>
- [11] ——. MicroBlaze Soft Processor Core. [Accessed 05 2017]. [Online]. Available: <https://www.xilinx.com/products/design-tools/microblaze.html>

- [12] Digilent. Nexys 4 DDR - Getting Started with Microblaze Servers. [Accessed 05 2017]. [Online]. Available: <https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-4-ddr-getting-started-with-microblaze-servers/start>
- [13] Xilinx. MicroBlaze Processor Reference Guide. [Accessed 05 2017]. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug984-vivado-microblaze-ref.pdf
- [14] ARM. AMBA 3 AXI Protocol Checker User Guide r0p1. [Accessed 05 2017]. [Online]. Available: <https://developer.arm.com/docs/dui0305/latest/glossary>
- [15] Xilinx. Xilinx Software Development Kit (XSDK). [Accessed 05 2017]. [Online]. Available: <https://www.xilinx.com/products/design-tools/embedded-software/sdk.html>
- [16] Y. Panarin. Introduction to C Language. [Accessed 05 2017]. [Online]. Available: <http://www.electronics.dit.ie:80/staff/ypanarin/Lecture%20Notes/K235-1/01%20Introduction%20to%20C.pdf>
- [17] Digilent. Nexys4-DDR introduction. [Accessed 05 2017]. [Online]. Available: https://www.youtube.com/watch?v=nk__TzBZ6_I
- [18] P. K. Gaikwad, "DEVELOPMENT OF FPGA MICROBLAZE PROCESSOR AND GSM BASED HEART RATE MONITORING SYSTEM," *International Journal of Computer Science and Mobile Applications*, vol. 1, pp. 24–29, 2013.
- [19] N. Joshi, "Development of FPGA MicroBlaze Processor and GSM based Wireless Monitoring System for Neonatal Intensive Care Unit," *International Journal of Research in Engineering & Advanced Technology*, vol. 4, 2016.
- [20] Digilent. Nexys4-DDR Advanced I/O Demo (Built-In Self-Test). [Accessed 05 2017]. [Online]. Available: <https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-4-ddr-user-demo/start>
- [21] ——. Nexys 4 DDR - Getting Started with Microblaze. [Accessed 05 2017]. [Online]. Available: <https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-4-ddr-getting-started-with-microblaze/start>
- [22] J. Johnson. Creating a custom IP block in Vivado. [Accessed 05 2017]. [Online]. Available: <http://www.fpgadeveloper.com/2014/08/creating-a-custom-ip-block-in-vivado.html>

- [23] Xilinx. Creating an AXI Peripheral in Vivado. [Accessed 05 2017]. [Online]. Available: <https://www.xilinx.com/video/hardware/creating-an-axi-peripheral-in-vivado.html>
- [24] fahrvergnuugen. Accelerometer Test. [Accessed 05 2017]. [Online]. Available: <https://docs.google.com/spreadsheets/d/1CLFJdp1atmTmEpoMR7QMtugu-Eg271VjV2CX3A7Pv58/edit?usp=sharing>

Appendix A

C Source Code

```
//Structure to hold real and imaginary numbers
struct complex
{
    float real, img;
};

//Twiddle factors, input data, and expected results (calculated in
    MATLAB)
struct complex tw[4] = {{1,0},{0.707,-0.707},{0,-1},{-0.707,-0.707}};
struct complex data[8] = {{0,0},{0.0391,0},{0.0487,0},{0.0217,0},
    {-0.0217,0},{-0.0487,0},{-0.0391,0},{0,0}};
struct complex gold[8] = {{0.0000,0.0000},{0.0685,-0.1653},
    {-0.0313,0.0313},{-0.0251,0.0104},
    {-0.0241,0.0000},{-0.0251,-0.0104},
    {-0.0313,-0.0313},{0.0685,0.1653}};

//Output structure for stage 1
struct complex stage1[8];

//First stage of butterfly operations
void stage1FFT(struct complex x[8]) {
    printf("FFT_INPUT\n");
    int i;
    for (i= 0; i < 8; i++){
        printf("x(%d) = (%.5f_%.5fi)\n",i, x[i].real, x[i].img);
    }
    stage1[0].real = x[0].real + x[4].real;
    stage1[1].real = x[0].real - x[4].real;
    stage1[2].real = x[2].real + x[6].real; //by tw0 (1 + 0j)
    stage1[3].real = (x[2].real - x[6].real)*tw[2].real; //by tw2 (0 - 1j)
    )
    stage1[3].img = (x[2].real - x[6].real)*tw[2].img; //by tw2 (0 - 1j)
```

```

    stage1[4].real = x[1].real + x[5].real;
    stage1[5].real = x[1].real - x[5].real;
    stage1[6].real = x[3].real + x[7].real; //by tw0
    stage1[7].real = (x[3].real - x[7].real)*tw[2].real; //by tw2
    stage1[7].img = (x[3].real - x[7].real)*tw[2].img; //by tw2
}

// Output structure for stage 2
struct complex stage2[8];

// Second stage of butterflies
void stage2FFT(struct complex s1[8]) {
    stage2[0].real = s1[0].real + s1[2].real;
    stage2[1].real = s1[1].real + s1[3].real;
    stage2[1].img = s1[1].img + s1[3].img;
    stage2[2].real = s1[0].real - s1[2].real;
    stage2[3].real = s1[1].real - s1[3].real;
    stage2[3].img = s1[1].img - s1[3].img;
    stage2[4].real = s1[4].real + s1[6].real;
    stage2[5].real = (s1[5].real + s1[7].real)*tw[1].real - (s1[5].img +
        s1[7].img)*tw[1].img;
    stage2[5].img = (s1[5].real + s1[7].real)*tw[1].img + (s1[5].img + s1
        [7].img)*tw[1].real;
    stage2[6].real = (s1[4].real - s1[6].real)*tw[2].real - (s1[4].img -
        s1[6].img)*tw[2].img;
    stage2[6].img = (s1[4].real - s1[6].real)*tw[2].img + (s1[4].img -
        s1[6].img)*tw[2].real;
    stage2[7].real = (s1[5].real - s1[7].real)*tw[3].real - (s1[5].img -
        s1[7].img)*tw[3].img;
    stage2[7].img = (s1[5].real - s1[7].real)*tw[3].img + (s1[5].img - s1
        [7].img)*tw[3].real;
}

// Output structure for stage 3
struct complex stage3[8];

// Final stage of butterflies
void stage3FFT(struct complex s2[8]) {
    int j;
    for (j = 0; j < 4; j++) {
        stage3[j].real = s2[j].real + s2[j+4].real;
        stage3[j].img = s2[j].img + s2[j+4].img;
    }
    int k;
    for (k=4; k < 8; k++) {
        stage3[k].real = s2[k-4].real - s2[k].real;
        stage3[k].img = s2[k-4].img - s2[k].img;
    }
}

```



```

    }
}

// Run all 3 stages of butterflies and print calculated results beside
// expected results

int main () {
    stage1FFT(data);
    stage2FFT(stage1);
    stage3FFT(stage2);
    int i;
    printf("FFT_OUTPUT\n");
    for (i= 0; i < 8; i++){
        printf("Calculated_Result:\tX(%d)_=(%.3f_%.3fi)\nExpected_Result:\tX(%d)_=(%.3f_%.3fi)\n",i, stage3[i].real, stage3[i].img,i,gold[i].real,gold[i].img);
    }
}

```

Appendix B

VHDL Accelerometer Code

```
-----  
-----  
-- Author:   Albert Fazakas  
--           Copyright 2014 Digilent, Inc.  
-----  
  
--  
-- Create Date:    15:00:45 03/04/2014  
-- Design Name:  
-- Module Name:    AccelerometerCtl - Behavioral  
-- Project Name:  
-- Target Devices:  
-- Tool versions:  
-- Description:  
--           This is the main module for the the Nexys4 onboard ADXL362  
--           accelerometer.  
--           The module consists of two components, AXDX362Ctrl and  
--           AccelArithmetics. The first one  
--           configures the ADXL362 accelerometer and continuously reads X, Y,  
--           Z acceleration data and  
--           temperature data in 12-bit two's complement format.  
--           The data read is sent to the AccelArithmetics module that  
--           formats X and Y acceleration  
--           data to be displayed on the VGA screen in a 512 X 512 pixel area.  
--           Therefore the X and Y  
--           acceleration data will be scaled and limited to -1g: 0, 0g: 255,  
--           1g: 511.  
--           The AccelArithmetics module also determines the acceleration  
--           magnitude using the  
--            $\text{SQRT}(X^2 + Y^2 + Z^2)$  formula. The magnitude value is also  
--           displayed on the VGA screen.  
--           To perform SQRT calculation a Logicore component is used.  
--
```

```

-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity AccelerometerCtl is
generic
(
    SYSCLK_FREQUENCY_HZ : integer := 100000000;
    SCLK_FREQUENCY_HZ   : integer := 1000000;
    NUM_READS_AVG       : integer := 16;
    UPDATE_FREQUENCY_HZ : integer := 100
);
port
(
    SYSCLK      : in STD_LOGIC; -- System Clock
    RESET       : in STD_LOGIC;

    -- Spi interface Signals
    SCLK        : out STD_LOGIC;
    MOSI        : out STD_LOGIC;
    MISO        : in  STD_LOGIC;
    SS          : out STD_LOGIC;

    -- Accelerometer data signals
    ACCEL_X_OUT  : out STD_LOGIC_VECTOR (8 downto 0);
    ACCEL_Y_OUT  : out STD_LOGIC_VECTOR (8 downto 0);
    ACCEL_MAG_OUT : out STD_LOGIC_VECTOR (11 downto 0);
    ACCEL_TMP_OUT : out STD_LOGIC_VECTOR (11 downto 0)
);
end AccelerometerCtl;

architecture Behavioral of AccelerometerCtl is

```

```

component ADXL362Ctrl
generic
(
    SYSCLK_FREQUENCY_HZ : integer := 100000000;
    SCLK_FREQUENCY_HZ   : integer := 1000000;
    NUM_READS_AVG       : integer := 16;
    UPDATE_FREQUENCY_HZ : integer := 1000
);
port
(
    SYSCLK      : in STD_LOGIC; -- System Clock
    RESET       : in STD_LOGIC;

    -- Accelerometer data signals
    ACCEL_X      : out STD_LOGIC_VECTOR (11 downto 0);
    ACCEL_Y      : out STD_LOGIC_VECTOR (11 downto 0);
    ACCEL_Z      : out STD_LOGIC_VECTOR (11 downto 0);
    ACCEL_TMP     : out STD_LOGIC_VECTOR (11 downto 0);
    Data_Ready   : out STD_LOGIC;

    --SPI Interface Signals
    SCLK         : out STD_LOGIC;
    MOSI         : out STD_LOGIC;
    MISO         : in  STD_LOGIC;
    SS           : out STD_LOGIC
);
end component;

component AccelArithmetics
generic
(
    SYSCLK_FREQUENCY_HZ : integer := 100000000;
    ACC_X_Y_MAX          : STD_LOGIC_VECTOR (9 downto 0) := "01" & X"FF";
    -- 511 pixels, corresponding to +1g
    ACC_X_Y_MIN          : STD_LOGIC_VECTOR (9 downto 0) := (others =>
        '0') -- corresponding to -1g
);
port
(
    SYSCLK      : in STD_LOGIC; -- System Clock
    RESET       : in STD_LOGIC;

    -- Accelerometer data input signals
    ACCEL_X_IN   : in  STD_LOGIC_VECTOR (11 downto 0);
    ACCEL_Y_IN   : in  STD_LOGIC_VECTOR (11 downto 0);
    ACCEL_Z_IN   : in  STD_LOGIC_VECTOR (11 downto 0);

```

```

Data_Ready      : in STD_LOGIC;

-- Accelerometer data output signals to be sent to the VGA controller

ACCEL_X_OUT      : out STD_LOGIC_VECTOR (8 downto 0);
ACCEL_Y_OUT      : out STD_LOGIC_VECTOR (8 downto 0);
ACCEL_MAG_OUT    : out STD_LOGIC_VECTOR (11 downto 0)
);
end component;

-- Self-blocking reset counter constants
constant ACC_RESET_PERIOD_US : integer := 10;
constant ACC_RESET_IDLE_CLOCKS : integer := ((ACC_RESET_PERIOD_US
    *1000)/(1000000000/SYSCLK_FREQUENCY_HZ));

signal ACCEL_X      : STD_LOGIC_VECTOR (11 downto 0);
signal ACCEL_Y      : STD_LOGIC_VECTOR (11 downto 0);
signal ACCEL_Z      : STD_LOGIC_VECTOR (11 downto 0);

signal Data_Ready : STD_LOGIC;

-- Self-blocking reset counter
signal cnt_acc_reset : integer range 0 to (ACC_RESET_IDLE_CLOCKS - 1) :=
    0;
signal RESET_INT: std_logic;

begin

-- Create the self-blocking reset counter
COUNT_RESET: process(SYSCLK, cnt_acc_reset, RESET)
begin
    if SYSCLK'EVENT and SYSCLK = '1' then
        if (RESET = '1') then
            cnt_acc_reset <= 0;
            RESET_INT <= '1';
        elsif cnt_acc_reset = (ACC_RESET_IDLE_CLOCKS - 1) then
            cnt_acc_reset <= (ACC_RESET_IDLE_CLOCKS - 1);
            RESET_INT <= '0';
        else
            cnt_acc_reset <= cnt_acc_reset + 1;
            RESET_INT <= '1';
        end if;
    end if;
end process COUNT_RESET;

```

```

ADXL_Control: ADXL362Ctrl
generic map
(
    SYSCLK_FREQUENCY_HZ => SYSCLK_FREQUENCY_HZ,
    SCLK_FREQUENCY_HZ   => SCLK_FREQUENCY_HZ,
    NUM_READS_AVG        => NUM_READS_AVG,
    UPDATE_FREQUENCY_HZ  => UPDATE_FREQUENCY_HZ
)
port map
(
    SYSCLK      => SYSCLK,
    RESET       => RESET_INT,

    -- Accelerometer data signals
    ACCEL_X      => ACCEL_X,
    ACCEL_Y      => ACCEL_Y,
    ACCEL_Z      => ACCEL_Z,
    ACCEL_TMP    => ACCEL_TMP_OUT,
    Data_Ready   => Data_Ready,

    --SPI Interface Signals
    SCLK         => SCLK,
    MOSI         => MOSI,
    MISO         => MISO,
    SS           => SS
);

Accel_Calculation: AccelArithmetics
GENERIC MAP
(
    SYSCLK_FREQUENCY_HZ => 100000000,
    ACC_X_Y_MAX          => "01" & X"FF", -- 511 pixels, corresponding
        to +1g
    ACC_X_Y_MIN          => (others => '0') -- corresponding to -1g
)
PORT MAP
(
    SYSCLK => SYSCLK,
    RESET  => RESET_INT,

    -- Accelerometer data input signals
    ACCEL_X_IN => ACCEL_X,
    ACCEL_Y_IN => ACCEL_Y,
    ACCEL_Z_IN => ACCEL_Z,
    Data_Ready => Data_Ready,

```

```
-- Accelerometer data output signals to be sent to the VGA display
ACCEL_X_OUT => ACCEL_X_OUT,
ACCEL_Y_OUT => ACCEL_Y_OUT,
ACCEL_MAG_OUT => ACCEL_MAG_OUT
);

end Behavioral;
```

Appendix C

VHDL AXI Peripheral Code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity multiplier is
  port (
    clk : in std_logic;
    a    : in std_logic_vector(15 downto 0);
    b    : in std_logic_vector(15 downto 0);
    p    : out std_logic_vector(31 downto 0)
  );
end multiplier;

architecture IMP of multiplier is

begin
  process (clk)
  begin
    if clk'event and clk = '1' then
      p <= a * b;
    end if;
  end process;
end IMP;
```


Appendix D

Sinusoids M-file

```
% Load in accelerometer data and timestamps from spreadsheet
% Plot in the time domain
signal = load('data.txt');
time = load('time.txt');
time = time - time(1);
figure
plot(time,signal,'DisplayName','Measured Wave Data')
ylabel('Distance (metres)')
xlabel('Time (seconds)')
title('Wave Data')
legend('show')
axis tight
% Single-sided magnitude spectrum with frequency axis in Hertz
% Each bin frequency is separated by fs/N Hertz.
figure
N = length(signal);
fs = 10;
fnyquist = fs/2;
f = (0:fs/N:round(fs/2));
X_mags = abs(fft(signal));
bin_vals = [0 : N-1];
fax_Hz = bin_vals*fs/N;
N_2 = ceil(N/2);
plot(fax_Hz(1:N_2), X_mags(1:N_2)/length(signal))
xlabel('Frequency (Hz)')
ylabel('Magnitude');
title('Single-sided Magnitude spectrum (Hertz)');
axis tight
% Plot measured data with a simple sinusoid of the same frequency
% and same amplitude plotted against it.
figure
plot(time,signal,'DisplayName','Measured Wave Data')
```

```

hold on
mysine = 0.05*cos(2*pi*0.6*time);
plot(time,mysine,'r','DisplayName','Simple Sinusoid')
ylabel('Distance (metres)')
xlabel('Time (seconds)')
title('Wave Data')
legend('show')
axis tight
% Plot simple sinusoid with 8-samples marked out on plot
figure
time = linspace(0,(1/0.6),100);
mysine = 0.05*sin(2*pi*0.6*time);
plot(time,mysine,'DisplayName','Input Data');
time = linspace(0,(1/0.6),8);
mysine = 0.05*sin(2*pi*0.6*time);
hold on
plot(time,mysine,'ro','DisplayName','8 points')
axis tight
ylabel('Distance (metres)')
xlabel('Time (seconds)')
title('Wave Data')
legend('show')

```

Appendix E

Preliminary Report



Wave Analysis using FPGAs on Marine Buoys

by

Brian COLGAN

This Report is submitted in partial fulfilment of the requirements of the Honours
Degree in Computer and Communications Engineering (DT081) of the Dublin
Institute of Technology

February 3, 2017

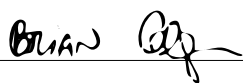
Academic Supervisor: Dr. Andreas SCHWARZBACHER

Company Supervisor: Mr. Paul DRUM

Declaration

I, the undersigned, declare that this report is entirely my own written work, except where otherwise accredited, and that it has not been submitted for a degree or other award to any other university or institution.

Signed:

A handwritten signature in black ink, appearing to read 'Brian', followed by a horizontal line.

Date:

February 3, 2017

Table of Contents

1	Introduction	1
1.1	Objectives	1
2	Research	3
2.1	Design	3
2.2	Data Processing	5
2.3	Wireless Transmission	6
2.4	Testing	7
3	Ethics	8
4	Work Plan	10
4.1	Resources	10
4.2	Milestones	10
4.3	Plan	11
	References	12
	Appendices	13
A	Project Risk Assessment Form	13

List of Figures

1.1	System Overview	2
2.1	Top Level Block Diagram	3
2.2	Butterfly Diagram	5
2.3	Data flow diagram for N=8: a decimation-in-time radix-2 FFT	5
2.4	Wireless embedded system built around Artix 7 FPGA	6
2.5	Interfacing UART and GSM SIM900A Module	6
4.1	Gantt Chart	11

Chapter 1

Introduction

1.1 Objectives

The key objectives of this project are to create a hardware design that can mathematically process wave data and wirelessly transmit this data using a Field Programmable Gate Array (FPGA). The Nexys4 DDR board will be used throughout the project. It is a complete, ready-to-use digital circuit development platform based on the latest Artix-7™ FPGA from Xilinx®.[1] The project will be completed over a 16-week period in collaboration with CréVinn Teoranta with the intention to demonstrate skills in hardware design and mathematical methods.

The FPGA must read in acceleration measurements using the on-board three axes accelerometer. This information will then be used to calculate velocity. Over a set amount of time, displacement data can be saved to a memory block in Random Access Memory (RAM). This data can then be mathematically processed to determine wave height, period and energy. The main challenge in this project is to design a Fast Fourier Transform (FFT) in the Hardware Design Language (HDL), SystemVerilog. The FFT is essential in calculating wave energy. Other algorithms to calculate wave height and period will be required, the approach to calculating this displacement data will be outlined in the Research chapter. Like the FFT, these will be implemented using HDL as opposed to implementing a soft-processor on the FPGA.

The height, period, energy from the above calculations must then be formed into a packet to be wirelessly transmitted. An on-board temperature sensor is also used to measure temperature of the water which will be appended to the final data packet in degrees Celsius. The received transmission will be a human-readable message which will outline sea state, weather conditions and water temperature based on the measured data to the end-user.

An overview of this system can be seen in Figure 1.1 below:

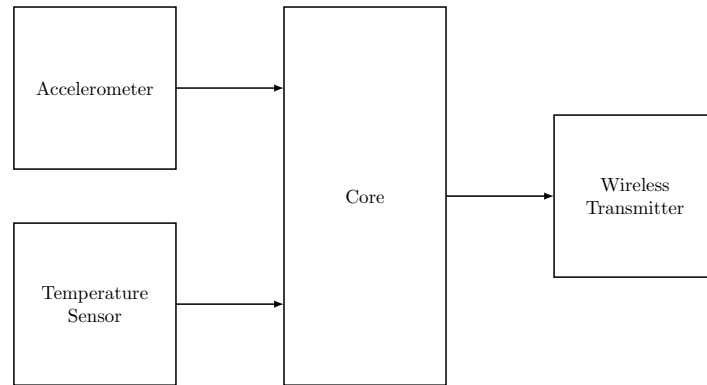


Figure 1.1: System Overview

For this design, the Core is the Intellectual Property (IP) block. The accelerometer and temperature sensor are inputs to the system and calculations will be output to the wireless transmitter.

Chapter 2

Research

2.1 Design

A Top-Level Block Diagram can be seen in Figure 2.1 below.

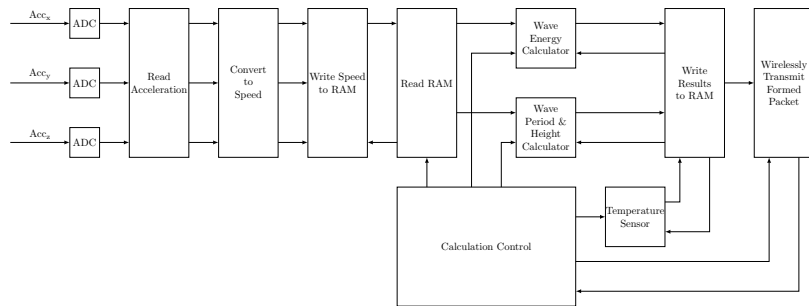


Figure 2.1: Top Level Block Diagram

ADC:

Analogue-to-digital converter to be used with accelerometer measurements as an input.

Read Acceleration:

Read in acceleration in all 3-axes

Convert to Speed:

Calculate speed from acceleration. ($v = u + at$)

Write Speed to RAM:

Store speed values in memory block until a sizeable array is built over time.

Read RAM:

When array is full, read values to be used for calculation.

Wave Energy Calculator:

Read in x, y and z measurements over time to calculate Wave Energy, this can be done with an FFT.

Wave Period & Height Calculator:

Read in x, y and z measurements to calculate displacement. ($s = ut + \frac{1}{2}at^2$) Only z-axis is required for Wave Height. Period can also be calculated with an FFT.

Temperature Sensor:

Read in water temperature to be transmitted.

Calculation Control:

Initiates calculation modules when value array is full.

Write Speed to RAM:

Energy, Height, Period and Temperature stored to a memory block.

Wireless Transmit Formed Packet:

Build data block in packet and transmit wirelessly.

2.2 Data Processing

To process the data the design will perform an FFT by implementing the Cooley-Turkey algorithm, this is the most common form of FFT. It is a recursive algorithm which breaks the Discrete Fourier Transform (DFT) into smaller DFTs. This design will divide the DFT of size $n = 2m$ into two interleaved DFTs of size m , this is known as radix-2.

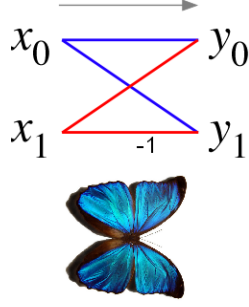


Figure 2.2: Butterfly Diagram[2]

These smaller DFTs are then combined via size-2 DFTs (known as butterflies in this context, due to the shape of the data flow) pre-multiplied by roots of unity (which are known as twiddle factors). The butterfly operation takes the form:

$$y_0 = x_0 + x_1 * tw$$

$$y_1 = x_0 - x_1 * tw$$

The above form is known as Decimation In Time (DIT). However, one can have a process where the butterflies come first and are post-multiplied by the twiddle factors, which is known as Decimation In Frequency (DIF).

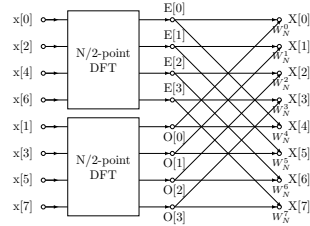


Figure 2.3: Data flow diagram for N=8: a decimation-in-time radix-2 FFT[3]

2.3 Wireless Transmission

To transmit the data wirelessly the design requires a specific wireless technology. Data will be sent via a Short Message Service (SMS). The design will have a Xilinx MicroBlaze soft-processor with a Universal Asynchronous Receiver/Transmitter (UART) port which will be used to send relevant Attention (AT) commands (from the Hayes command set) and drive the Global System for Mobile Communications (GSM) module. This design can be used in many applications.[4, 5] A block diagram of the design can be seen in Figure 2.4.

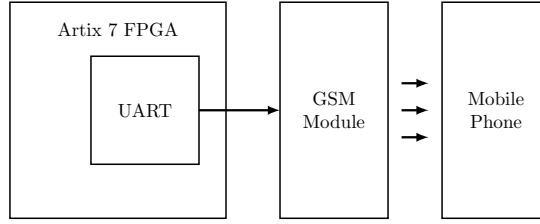


Figure 2.4: Wireless embedded system built around Artix 7 FPGA

Data will be transmitted using the RS-232 standard for serial communication. This interface can be seen in Figure 2.5.

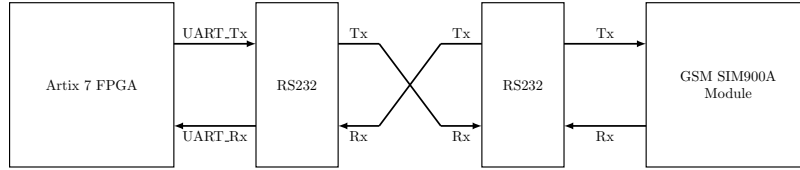


Figure 2.5: Interfacing UART and GSM SIM900A Module

2.4 Testing

The testing of the FFT will be done by entering a sinusoid as stimulus to the design. The output of this system will be measured against expected results. These test-cases will be done with a SystemVerilog test-bench. Wave data will also be obtained to measure against pre-determined results.

Initially the GSM module will be tested by sending an SMS to a personal mobile phone. Once this functionality is achieved, a new Subscriber Identity Module (SIM) card will be put into the GSM module. This allows texts to be sent to third-party services, e.g. Twitter.

Chapter 3

Ethics

The information obtained in this project is particularly useful for two groups within society; marine users and wave power researchers. As such, the ethical requirements of this project can be divided into two parts;

- Relations with colleagues, clients, employers and society in general
- Environmental and social obligations

both of which are set out by Engineers Ireland in its Code of Ethics.[6]

With regards to societal relations, the intention of the project is to provide safety information to marine users. In this context, the project is a demonstration of skill/expertise being applied to the common good as well as advancing human welfare with proper regard for the health and welfare of the public. It is important to note that this project is designed to be non-hazardous and have zero military applications so as to not result in a serious detriment to any person or persons.

Environmental and social obligations are also a primary motive for this project. As outlined above wave power researchers would, understandably, have significant interest in wave energy estimations. The design of this project is intended to promote the principles and practises of sustainable development and the needs of the present and future generations. The project shall also foster environmental awareness within the profession of engineering among the public.

The reason an Artix®-7 FPGA was selected was due to it being the highest performance-per-watt fabric offered by Xilinx.[7] This allows the design to accomplish the objectives outlined in the Introduction with the most efficient consumption of natural resources which is practicable economically. This includes the maximum reduction in energy usage, waste and pollution.

As FPGAs already have low power consumption (in comparison to microprocessors) it would also be possible to power the design itself with a renewable energy source e.g. solar energy. Execution of the project in this way eliminates any adverse impact on the natural environment.

Chapter 4

Work Plan

Time management is vital within a project, as such the project is broken down into phases and then again into tasks. Each task is allotted a predicted amount of time and this allows schedules and deadlines to be made. Each phase and its corresponding tasks are input to a Microsoft Project file. This predicted timeline is displayed as a Gantt chart which allows for accurate tracking of tasks and resources.

4.1 Resources

Resources for this project were provided by the company sponsoring the project, CréVinn Teoranta, these include materials such as the FPGA as well as a computer to work on throughout the project.

4.2 Milestones

Milestones are the major goals within a project, these are outlined below;

- Design specification
- Register-Transfer Level (RTL) Design
- RTL Verification
- Design on FPGA
- GSM Integration

4.3 Plan

The Gantt chart for the project can be seen in Figure 4.1.

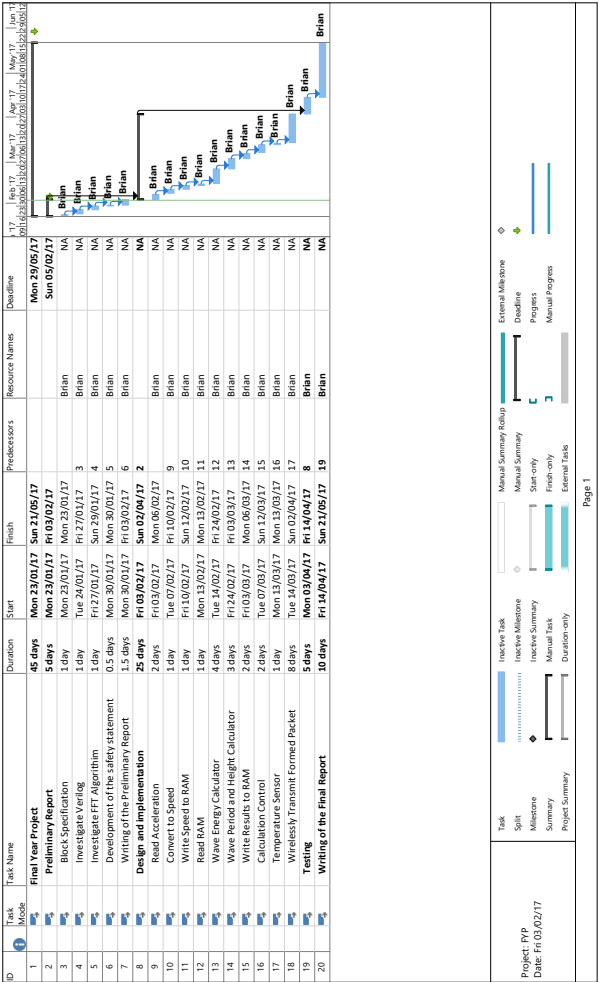



Figure 4.1: Gantt Chart

References

- [1] Digilent. Nexys4 DDR™ FPGA Board Reference Manual. [Accessed 03 02 2017]. [Online]. Available: https://reference.digilentinc.com/_media/nexys4-ddr:nexys4ddr_rm.pdf
- [2] S. G. Johnson. File:Butterfly-FFT.png. [Accessed 03 02 2017]. [Online]. Available: <https://commons.wikimedia.org/wiki/File:Butterfly-FFT.png>
- [3] Virens. File:DIT-FFT-butterfly.png. [Accessed 03 02 2017]. [Online]. Available: <https://commons.wikimedia.org/wiki/File:DIT-FFT-butterfly.png>
- [4] P. K. Gaikwad, “DEVELOPMENT OF FPGA MICROBLAZE PROCESSOR AND GSM BASED HEART RATE MONITORING SYSTEM,” *International Journal of Computer Science and Mobile Applications*, vol. 1, pp. 24–29, 2013.
- [5] N. Joshi, “Development of FPGA MicroBlaze Processor and GSM based Wireless Monitoring System for Neonatal Intensive Care Unit,” *International Journal of Research in Engineering & Advanced Technology*, vol. 4, 2016.
- [6] Engineers Ireland. Code of Ethics. [Accessed 03 02 2017]. [Online]. Available: <http://www.engineersireland.ie/EngineersIreland/media/SiteMedia/about/Engineers-Ireland-Code-Of-Ethics-2010.pdf>
- [7] Xilinx. Artix-7 FPGA Family. [Accessed 03 02 2017]. [Online]. Available: <https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html>

Appendix A

Project Risk Assessment Form



Project Risk Assessment Form

Student(s) Name

Programme Code and Year

Start Date of Project

Supervisor Name

Brain Colgan

C12880846

DT081/4

23 January 2017

Dr. Andreas Schwarzbacher

Implement Wave Analyser using Nexys 4 DDR Field Programmable Gate Array (Artix-7). Project is sponsored by Cr@Winn Tioranta and will mostly be carried out in their Ballymount office. Hardware Design Language (HDL) will be written from a laptop/desktop.

Potential Hazards

Present (Yes/No)

Details

Controls Required

Work Equipment and Electrical Shock Hazard

Yes

FPGA Board

Adhere to all relevant company policies

Hand tools / Power Tools

No

N/A

N/A

Slips, trips and falls

No

N/A

N/A

Manual Handling

No

N/A

N/A

Soldering

No

N/A

N/A

Compressed Air

No

N/A

N/A

Rotating Machinery

No

N/A

N/A

Noise

No

N/A

N/A

Computer Usage

Yes

Design work done from desktop/laptop

Take breaks at regular intervals to rest eyes and avoid being sedentary

Lone Working	Yes	Work can be done at home or in the office without supervision	Ensure contact is maintained with colleagues and academic supervisor
Sudden Illness and/or Medical Emergencies	No	N/A	N/A
Fire/Emergency Evacuation	No	N/A	N/A
Biological Agents	No	N/A	N/A
Chemical	No	N/A	N/A
Gases	No	N/A	N/A
Heat Sources/High Temperatures/Hot Surfaces	No	N/A	N/A
Lasers	No	N/A	N/A
Vibrations	No	N/A	N/A
Working off Campus	Yes	Will be working mostly in Ballymount office	Maintain contact with academic supervisor
What other Hazards could be present and what effect will they have on the student(s), fellow students or employees? Use separate sheet if required.	N/A	N/A	N/A

Category of Supervision Required		
Category	Level of Risk Present	Tick one
Category A	The risks associated with the work and/or the experience of the student(s) are such that the work must be supervised directly by a competent person (the supervisor or his/her authorized nominee), at least until the supervisor is satisfied that the student(s) can follow correctly the appropriate scheme of work. Note: this might apply only to a small section of the whole project.	<input type="checkbox"/>
Category B	The risks associated with the work and/or the experience of the student(s) are such that the work may not be started without the supervision of his/her authorized nominee's advice and approval.	<input type="checkbox"/>
Category C	The risks associated with the work requires considerable care but it is considered that the student(s) is adequately trained and competent in the procedures involved.	<input type="checkbox"/>
Category D	The risks are low and carry no special supervision requirements.	<input checked="" type="checkbox"/>

Risk Assessment Sign Off	
Student(s) Sign and Date	Supervisor Sign and Date

THE SIGNED FORM SHOULD BE SUBMITTED TO _____, THE PROJECT SUPERVISOR AND THE STUDENT(S) SHOULD RETAIN A COPY OF SAME.