Ultrasound Imaging System for Educational Purposes

A THESIS SUBMITTED

BY

YAZAN BARHOUSH

TO

THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

BACHELOR OF COMPUTER ENGINEERING

IN THE SUBJECT OF

COMPUTER ENGINEERING

UNION COLLEGE

SCHENECTADY, NEW YORK

JUNE 2017

Faculty advisor: Professor Takashi Buma Yazan Barhoush

Ultrasound Imaging System for Educational Purposes

Abstract

An alternative to increase the effectiveness of in-class delivery methods can be hands-on educational tools. This senior project is an upgrade of an older version of an Ultrasound imaging system that will be used for educational purposes by college students. The system will allow students to gain access to a tool that can produce better understanding of ultrasound technology, the process of capturing images and electronics involved. Certain parts of the older system were replaced to achieve improved flexibility, modularity, and better control. Field-Programmable Gate Array (FPGA) and Microcontroller (MCU) were considered as valid options for improving the system. Various aspects between MCU's and the FPGA's were compared, but results were close and preliminary testing was required to make the final decision. Altera DEo-naon FPGA was chosen to produce high precision synchronous signals and replace the older Timing Electronics.

Contents

Listing of figures

List of Tables

1 Introduction

While the balance between theory and practice of engineering curriculum has dramatically changed over time[7], the effectiveness of delivery methods in many engineering schools today is relatively the same: different methods try to find an efficient technique for delivering large quantities of analytical information.[14] Limits of such an approach have been indicated by studies in behavioral psychology[13] and cognitive research[15]. An alternative to increase the effectiveness of these delivery methods can be hands-on educational tools. However, these tools have to be visual, engaging and encourage active and cooperative learning[10]

In this thesis report, we construct an ultrasound imaging system that would serve as an example topic for such a tool. The system should be interesting to many students and cover certain engineering concepts that include the nature of ultrasound and its physics, ultrasonic echo imaging technique and its features, different image modes, transducers and real-life applications.

The project can be integrated to augment a practice-based curriculum. This can provide a new engineering educational experience: a combination of visual curriculum with opportunities for application and hands-on experience, thereby erasing the traditional boundaries between theory and practice of engineering curriculum.

2 | Background

2.1 Principles of Ultrasound Imaging

Sound waves are longitudinal waves involving the alternating compression and rarefaction of a medium, with a power determined by the wave's amplitude. The speed of sound waves is a constant independent of frequency, wavelength and direction in fluids and many materials. Pulse-echo imaging works when a pulse of sound is emitted, which then travels outward at the speed of sound, v, of the surrounding medium. If the sound wave is reflected from an object, it then travels back to the source and is detected as an echo at the source. See Figure 1.1, A. Thus, the sound wave travels a distance equal to twice the distance from the source to reflecting object, z, in a time T, related by the equation 1.1, The factor of 2 accounts for the round-trip taken by the sound pulse.[12]

$$z = v \times \frac{T}{2} \tag{2.1}$$

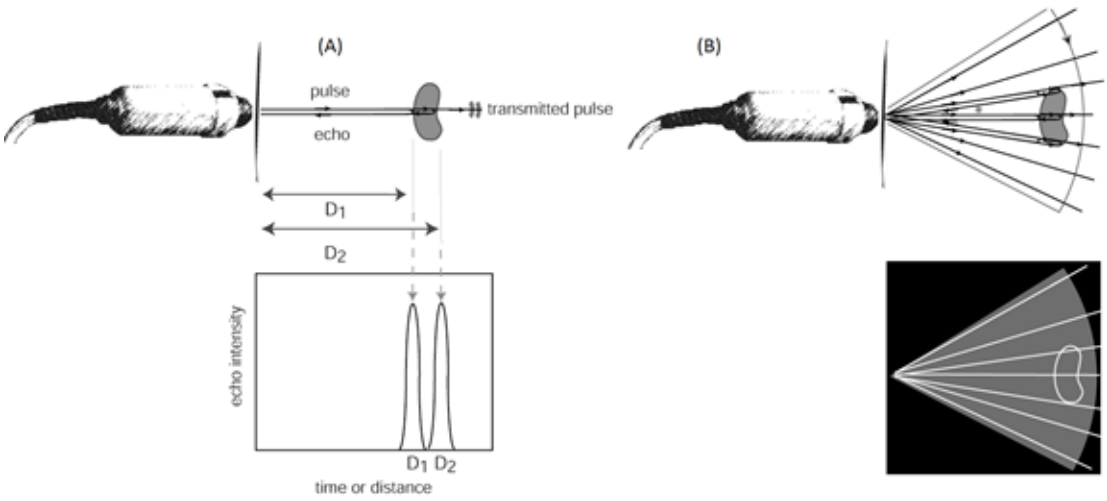

Figure 2.1: Illustration of the operation of an ultrasound pulse-echo scanner. Edited[12]

2.1.1 Ultrasound transducers

Ultrasound transducers convert electrical energy into sound waves using piezoelectricity. The black working end of the transducer is a piezoelectric crystal that undergoes mechanical deformations when a voltage is applied across it. The same transducer emits brief ultrasound pulses, then acts as a receiver to detect the returning echoes. See Figure 1. The scanner records the time required for each pulse to return, then uses the speed of sound to convert that into a distance to the object producing the echo. (Figure 1, A) shows results displayed when the echo intensity is plotted vs. the time for the echo to return. Peaks in the echo intensity occur at those times corresponding to distances at which reflective interfaces occur in the body. Such a plot can be converted into a plot of echo intensity vs. depth. However, in order to reveal the anatomy of objects, the transducer can be moved back and forth, and the two-dimensional (2D) locations of objects can be measured: the depth (y) comes from sonar ranging and the transducer's position supplies a second (x) dimension.

2.1.2 Ultrasound imaging resolution depends on frequency

The frequency of a sound wave determines its wavelength, ”λ, through the equation:

$$\lambda = v \times f \tag{2.2}$$

The wavelength of a sound wave also determines how small a feature will produce a distinct reflection. In other words, the higher the frequency, the better the spatial resolution –the smaller the separation at which one can distinguish echoes from two distinct, nearby objects.

2.1.3 Ultrasound imaging absorption depends on frequency

The absorption of ultrasound increases with frequency. As a result, ultrasound imaging is a trade-off between spatial resolution and achievable depth of imaging.

2.2 System Goals

This senior project is an upgrade of an older version of an Ultrasound imaging system system. The system will produce better understanding for ultrasound technology, the process of capturing images and electronics involved. We also aim to unify the programming environment for both Data acquisition and Data processing. The constructed ultrasound imaging system will be used for educational purposes by college students. Therefore, the newly introduced hardware and software implements must provide a hands-on educational demonstration techniques for mediating learning.

2.3 Old system and its limitations

This senior project is an upgrade of an older version of an ultrasound imaging system. The older version of the system attempted to produce a 10 kHz trigger and a 5 Hz square wave using three different designs: 555 timers, an Arduino micro-controller and a frequency crystal oscillator, divided down by a synchronous counter.

The first two designs had some accuracy issues, including delay accumulation and did not produce very accurate signal. Reasonable success was achieved with a highly accurate crystal oscillator and a clock divider: images produced were pretty good, but the hardware implementation was inconvenient and inflexible. For this project, we are planning on replacing the timing electronics to achieve improved flexibility, modularity, and better control.

2.4 Important issues in terms of goals and the effects on society

The project aims to produce a hands-on educational tool. It augments the effectiveness of delivery methods in a class room. Thereby, it may face education-related issues. These issues can be social or political. To elaborate on these issues, we will take Palestine, a third world country, as an example scenario:

The development of Higher Education (HE) in Palestine is of relatively recent date[9]. There is an increased interest from the Palestinian government organizations and higher education institutions to improve existing Science, Engineering, Technology, and Innovations (SETI) education[16].However, HE institutions face two obstacles in responding to the demand for higher education: they need to invest in upgrading and expanding their

faculties and their facilities, and overcome the many severe problems created by the Israeli occupation[6]. There also exists equity problems in HE in the West Bank and Gaza related to socioeconomic status, gender, and region of residence . Additionally, pattern comparisons of the percentage of HE students and graduates by field of study show underproduction to some degree in Natural Sciences, Engineering, and Agriculture[11]. Qualitatively, Palestinian employers view HE graduates as too theoretical, as having weak educational preparation[5].

The suggested system can contribute to the discussion of how educational technology can create tools to inform, attract, and instill engineering concepts and skills in SETI education in a developing country like Palestine.

The constructed ultrasound imaging system would serve as an educational tool. It will include hardware and software implements as well as demonstration techniques for mediating learning. It could further enhance learning, promote hands-on experiences, and increase interest in engineering. The system can be practical solution, thereby creating a learning experience while overcoming obstacles faced by the Palestinian HE institutions and SETI graduates.

3 | Design Requirements

The system is composed of six blocks: A transducer, motor, pulser-receiver, data acquisition device, data processing unit and timing electronics. See Figure 3.1.

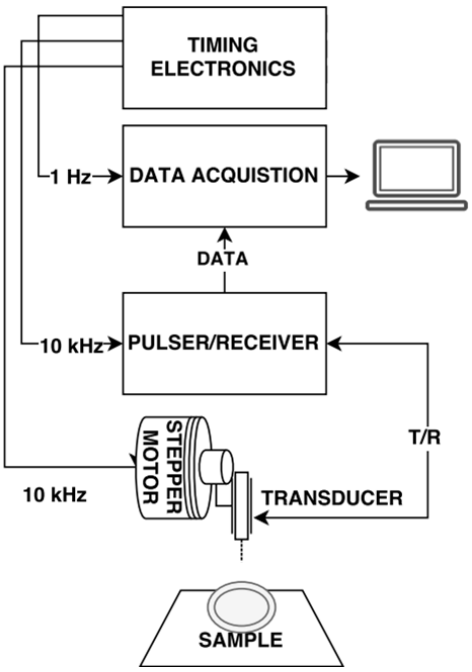

Figure 3.1: Block diagram of the ultrasound system

The pulser-receiver[1] employed with ultrasonic transducer are the main building blocks of any ultrasonic test system. The pulser section produces an electrical pulse to excite the transducer that converts the electrical input to mechanical energy, creating an ultrasonic wave.[2]

In pulse-echo applications, as in this project, the transducer reconverts the mechanical reflected pulse into an electrical signal that is then amplified and conditioned by the receiver section. The resulting Radio-frequency (RF) waves are then acquired using the data acquisition device. The data acquisition device makes the waves data available for further analysis using the computer –the Computer processes the

data and displays ultrasound images using a programing environment.

The Stepper Motor sweeps the transducer back and forth to scan across the sample. The Timing Electronics generate synchronous signal that triggers various parts of the system and controls the motor. Figure 2.1 shows a case were three signals leave the Timing Electronics: a 1 Hz signal that triggers data acquisition, a 10 kHz signal that triggers the pulser-receiver and a third signal the controls a stepper motor.

3.1 Detailed design specifications

Table 3.1: Detailed design specifications

Specification	Requirement
Frame rate	1 frames per second
Number of lines per frame	400 A-lines
Transducer pulsing frequency	10 kHz
A/D sampling rate	100 MS/s

The newly built system uses parts of the old system, but replaces the timing electronics. In order for it to be functional and achieve its goal; it has to meet the old system requirements as well as improved flexibility, modularity, and better control. Therefore, the new system has to be able to generate Signals while meeting the accuracy requirement, acquire and process data to reconstruct ultrasound images and control a motor.

For data acquisition, we are using the old digitizer (USB Oscilloscope Device), which operates at a 100 MS/s sampling rate. MATLAB will be used for ultrasound image processing and reconstruction and we decided to go with a stepper motor to sweep across the sample. (Part choice/decision will be explained later on)

3.1.1 Generate Signals

Since the ultrasound images do not have to be displayed real-time, we aim for 1 frame per second, where each frame consists of about 400 A-lines (enough data to construct an image). See Table 2.1. Additionally, the transducer is pulsed at a 10 kHz repetition rate, where stepper motor sweeps across the sample accordingly. Pulse signals output has to be adjustable so that the transducer can emit the incoming electrical energy as sound waves – reflected echoes are the data that needs to be collected and processed using a user interface.

The digitizer is triggered every 1 second so that it collects the data and makes it available for the computer to analyze it. Ultrasound images are then displayed by processing/reconstructing this data in a programing environment.

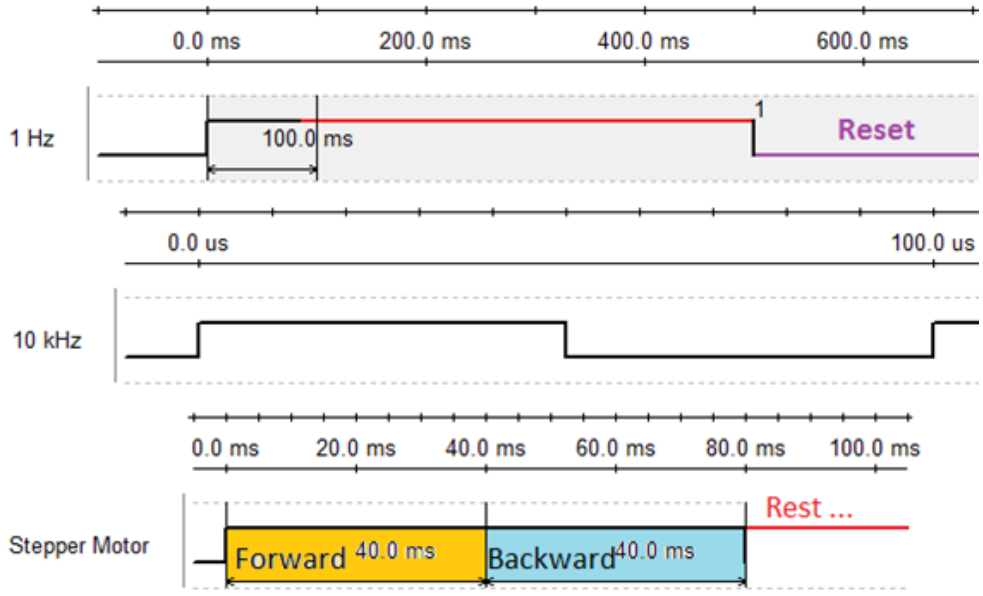

Figure 3.2: Timing Diagram

- 1 Frame Per Second = 1 Hz pulses, where 1 sweep = 400 micro-Steps: 1 frame per second can be achieved by generating 1 Hz pulses. Digitizer can also be triggered every 1

second using a 1 Hz pulse train.

- Pulsing 400 A-Lines where 1 A-Line is pulsed per micro-step = 10 kHz pulses: In the first 400 pulses: the 400 A-lines are pulsed, in 400 micro-Steps (1 per micro-Steps). The next 400 micro-Steps sweep the motor back.

  There is a 1000 10-kHz pulse in 100ms. In the first 80 ms, the stepper motor sweeps forward for 40ms and backward for the next 40ms. For the rest of "high", the stepper motor rests. When the 1 Hz is "low" the stepper motor is reset, and gets ready for the next sweep (when the 1 Hz is high again).

3.2 Accuracy Requirement

In addition to generating the signals, the system has to meet a certain accuracy requirement; worst-case delay error has to be less than 1 us in 400 pulses (less than 0.75mm shift in ultrasound image displayed)

From design specifications, the digitizer operates at 100 MS/s: this means that the sample is recorded every 10 ns. From Eq. 1.1 and knowing that the speed of sound is about 1.5 mm/us in tissue:

(Assuming that we are imaging a flat object at depth z_0, so all A-lines should have the same echo at the exact same depth z_0)

Suppose that the first A-line shows an echo at

$$z_0 = v \times \frac{t_0}{2}$$

and that the last A-line shows an echo at

$$z = v \times \frac{t_0 + \Delta T}{2}$$

The depth error between the first and last A-line is therefore

$$z - z_0 = v \times \frac{\Delta T}{2}$$

A depth error of less than one ultrasound wavelength is desirable (0.75mm for 25 MHz Ultrasonic Transducers). Therefore, $v \times \frac{\Delta T}{2}$. Thus, $\Delta T < 1$ us, where ΔT is delay time between first and last A-line.

3.3 Old system final design versus Suggested system design

Previous system's final design used a high precision crystal oscillator to provide the clocking signals for the entire system. The crystal oscillator provided a 10 kHz signal. The signal was divided down to a 5 Hz using a synchronous counter. The 10 kHz crystal fed three inputs: a synchronous counter that was implemented to correct digitizer's timing, a pulser-receiver that was used to trigger transducer's pulses, and a synchronous counter that was used to produce the 5 Hz needed to control the motor.

See Table 2.2 for side-by-side comparison of the final old system (using crystal oscillator and CMOS counters) and the proposed NEW system.

Similar to the old system, the new system will be able to generate accurate signals for imaging. However, the new system offers better control, flexibility and modularity: it will be able to generate various sets of desired signals instead of one for replacing parts to satisfy

Table 3.2: Side-by-side comparison between final old and proposed new systems.

Satisfied Requirement	Old System	New system
Accuracy	Produces Accurate Signals	Produces Accurate Signals
Control	Limited control	Better control
Flexibility	Limited flexibility	Improved flexibility
User-friendliness	No	Yes
Convenience	Convenient enough	More convenient

certain design requirements (Table 2.3). Additionally, the new system will be more convenient since many tasks will be achieved using a single block only; the timing electronics will be used for triggering data acquisition and pulser-receiver, and controlling the transducer's movement. Hardware like binary counters, level shifter and amplifiers will not be needed anymore. Moreover, the new system promises to be more software oriented than hardware; many tasks of the older system are replaced by "chunks of code" rather than hardware blocks, and all these tasks are going to be completed using the timing electronics (e.g. FPGA). The programming environment for data acquisition and processing will be the same. The code will be modular. It will make the system more user-friendly, convenient and easy to use.

Table 3.3: Parts replaced to satisfy certain design requirements.

Satisfied Requirement	Replaced Part	
	Timing Electronics	Data acquisition and processing Environment
Accuracy	Produce Accurate Signals	
Control	Control Motor	Desired signal
Flexibility	Various signal sets	Modular Code
User-friendliness		User friendly
Convenience	Compact/one block	Unified for Both

4 | Design Alternatives

4.1 Replacing Timing electronics

Timing electronics are required for the synchronization of the operations between various components of the system, and provide the timing for the execution of these operations. This is done by generating pulses using an oscillator. Clock Pulses are used to keep track of time, and controlling the speed of execution.

There are two schemes to generate signals. They differ from an algorithmic viewpoint:

- (Option 1) 5 Hz triggers a 10 kHz pulse train

  This implies that the two signals are generated separately using two separate blocks (two different functions in the case of MCU), and then matched together using a delay so that the 5 Hz would trigger a 10 kHz.

- (Option 2) 10 kHz divided down to 5 Hz

  This implies that the two signals output from the same block (same method in the case of MCU), where the 10 kHz is divided down into 5 Hz using a counter: number of faster clock pulses is counted until half of the slower clock period has passed.

Both MCUs (Microcontrollers) and FPGAs (Field-programmable gate arrays) contain oscillators that are used to provide stable clock pulses to digital circuits. Moreover, both options will ease manipulating values to change desired frequency of signals. They also can be used to control the motor that will be used for controlling the transducer's movement.

For the scope of this project, we decided to go immediately with some specific devices (e.g. C8051 and Altera DE2 board), since this is what we had available. However, concrete values for will be need for making comparisons in the future.

4.1.1 Micro-controller (C8051F020)[4]

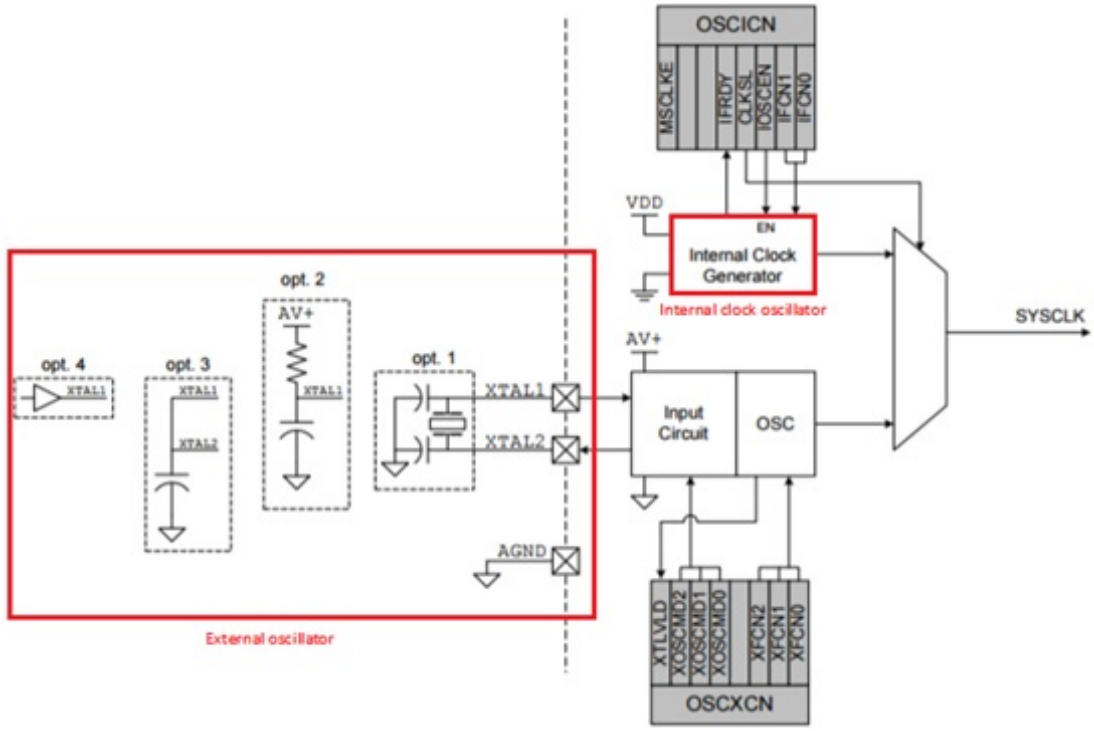

Figure 4.1: System Clocks: Internal clock oscillator: default at reset, 2 MHz default frequency and configured by SFR OSCICN. External oscillator: installed on board, 22.1184 MHz frequency and configured by SFR OSCXCN.

Each MCU includes an internal oscillator and an external oscillator drive circuit, either of which can generate the system clock. The MCUs operate from the internal oscillator after any reset. This internal oscillator can be enabled/disabled and its frequency can be set using the Internal Oscillator Control Register (OSCICN) as shown in Figure 3.1.

The external oscillator requires one of the four options: external resonator, crystal, capacitor, or RC network to be connected to the XTAL1/XTAL2 pins. The oscillator circuit must be configured for one of these sources in the OSCXCN special function register (SFR). An external CMOS clock can also provide the system clock; in this configuration, the XTAL1 pin is used as the CMOS clock input.

4.1.2 FPGA (Cyclone II EP2C35F672C6)[8]

A photograph of the DE2 board is shown in Figure 3.2. It depicts the layout of the board and indicates the location of the connectors and key components.

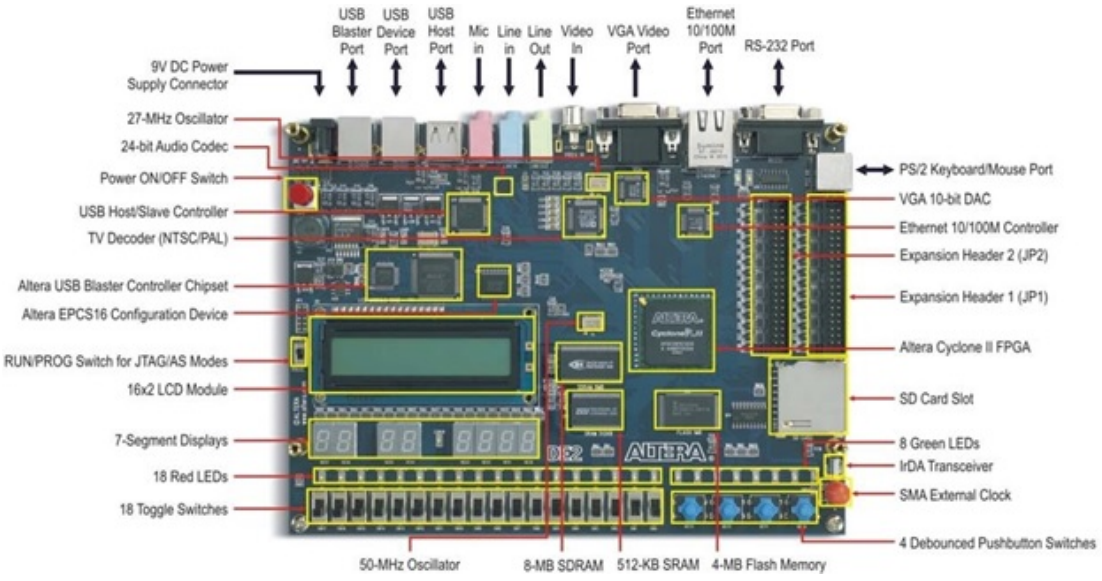

Figure 4.2: Photograph of DE2 board.

The DE2 board includes two oscillators that produce 27 MHz and 50 MHz clock signals. The board also includes an SMA connector which can be used to connect an external clock source to the board.

The schematic of the clock circuitry is shown in Figure 3.3, and the associated pin assignments appear in Table 3.1.

Table 4.1: Associated pin assignments with DE2 clock

Signal Name	FPGA Pin No.	Description
CLOCK_27	PIN_D13	27 MHz clock input
CLOCK_50	PIN_N2	50 MHz clock input
EXT_CLOCK	PIN_P26	External (SMA) clock input
Clock Speed	22.118MHz	50Mhz

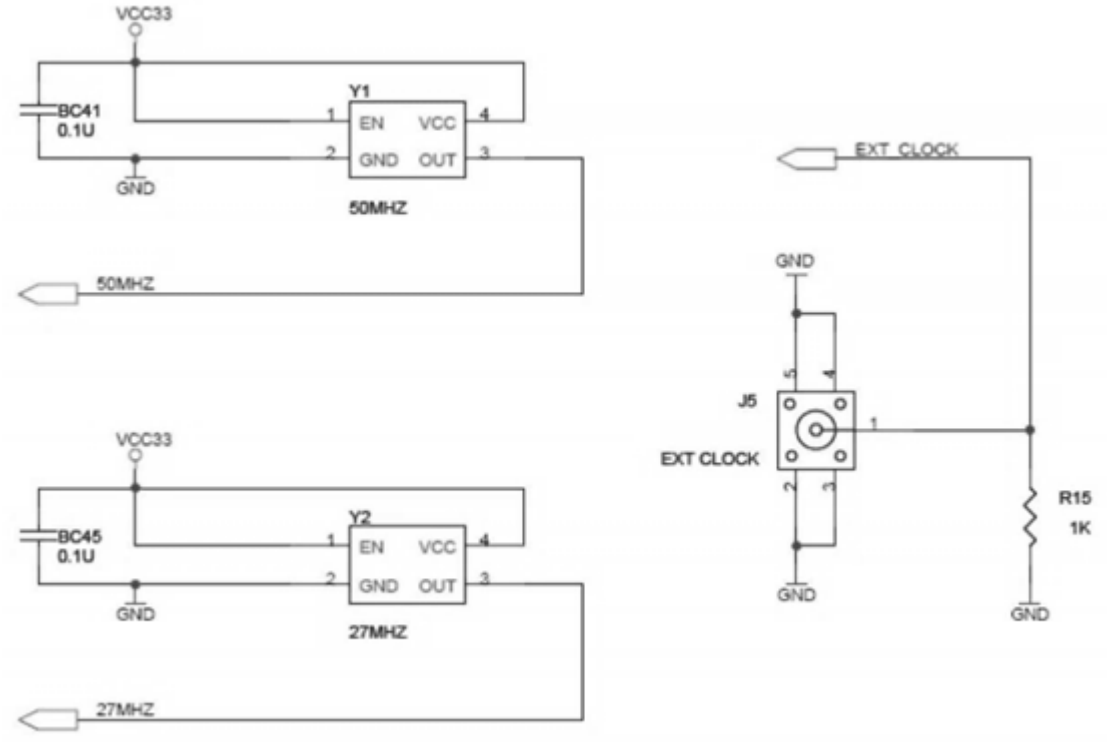

Figure 4.3: Schematic diagram of the DE2 clock circuit.

4.1.3 c8051 MCU vs DE2 FPGA

Table 3.2 compares various aspects between the MCU and the FPGA. They use different programming languages, execute code in different ways and have different limitations of processing power. However, for the task they are used for, that is generating signals, we are mainly interested in clock speed*. Although a decision matrix, Table 3.3†, was calculated to make a final decision, results were close and preliminary testing was required to make the decision.

Table 4.2: Micro-controller (c8051) vs FPGA (Altera DE2 board).

Aspect	Micro-controller (C8051)	FPGA (Altera DE2 board)
Programing language	C and Assembly	VHDL Hardware Description Language
Application Suitability	Serial execution: speed depends on instruction execution time	Parallel execution: Hardware
Processing Power	Time-limited: depends on processer	Space-limited: depends Logic circuits number
Clock Speed	22.118MHz	50Mhz

Hardware: Parallel execution Clock Speed: 50MHz. Interfaces include: External GPIO headers; On-board memory devices; General user peripheral with LEDs, Switches and push-buttons.

*Higher frequency clock signal will have more number of pulses per second. Hence, faster execution.

†The DE2 board clock was given the highest rating. This because 50 MHz is a round number which can be divided to exactly 10 kHz.

Table 4.3: A Weighted Decision Matrix for FPGA vs MCU.

CONCEPTS					
		C8051		DE2	
Criteria	Wgt	Rating	Score	Rating	Score
Programing language	.10	2	.20	1	.10
Application Suitability	.10	1	.10	1	.10
Processing Power	.10	1	.10	1	.10
Clock Speed	.20	2	.40	3	.60
TOTAL			.70		.90
RANK			2		1
CONTINUE?			Yes		Yes

4.2 Motor Driving

A motor is needed to control transducer's movement. Three different types of motors are compared to see what suits the design best. See Table 3.4 below.

Table 4.4: Comparison between DC, servo and stepper motors

Aspect	DC Motors	Servo Motors	Stepper Motors
Speed	Fast	Fast	Slow
Description	Continuous rotation motors	High torque rotary/linear actuator	Discrete steps DC motors
Advantage	High RPM	Accurate rotation within a limited angle	Easy set-up Position control
Example Application	Car wheels Fans	Robotic arms Rudder control	3D printers

DC motors are fast, continuous rotation motors. They are usually used for anything that

needs to spin at a high RPM. Servo motors are fast, high torque, accurate rotation motors within a limited angle. Generally used as a high-performance alternative to stepper motors, but more complicated setup with PWM tuning. Stepper motors are slow, precise rotation, easy set up and control motor. While servo servos require a feedback mechanism and support circuitry to drive positioning, a stepper motor has positional control via its nature of rotation by fractional increments. It is suited for designs where position is fundamental. Hence, for the application we have, and the performance needed, we decided to go with a stepper motor.

5 | Preliminary Proposed Design

We evaluated the two approaches (C8051 MCU and Altera DE2 FPGA board) to determine which one is the better way to go. The external 22.1184 MHz clock oscillator option 1 (external resonator) was chosen to generate the required 10 kHz pulse on port 3 pin 0 using interrupts (not polling). It was used for the system clock. Additionally, timer 0 in the 8-bit automatic reload mode with interrupts enabled (Mode 2: 8-bit Counter/Timer with Auto-Reload). Its clock source was set to be the system clock divided by 12 and the reload values needed for timer 0 to provide the 10 kHz signal we calculated (See Appendix).

Option 1 was used because an external CMOS clock for the system clock is not an option. The MCU was used for its clock. The external 22.1184 MHz clock was the best option for being on board and having the highest frequency –a higher frequency clock signal will have more number of pulses per second. Hence, faster execution.

5.1 Circuit Design using C8051 Microcontroller

Auto-reload mode was used to make the code easily modifiable: Mode 2 provides short repetitive intervals, which might be needed to generate a different signal than 10 kHz.

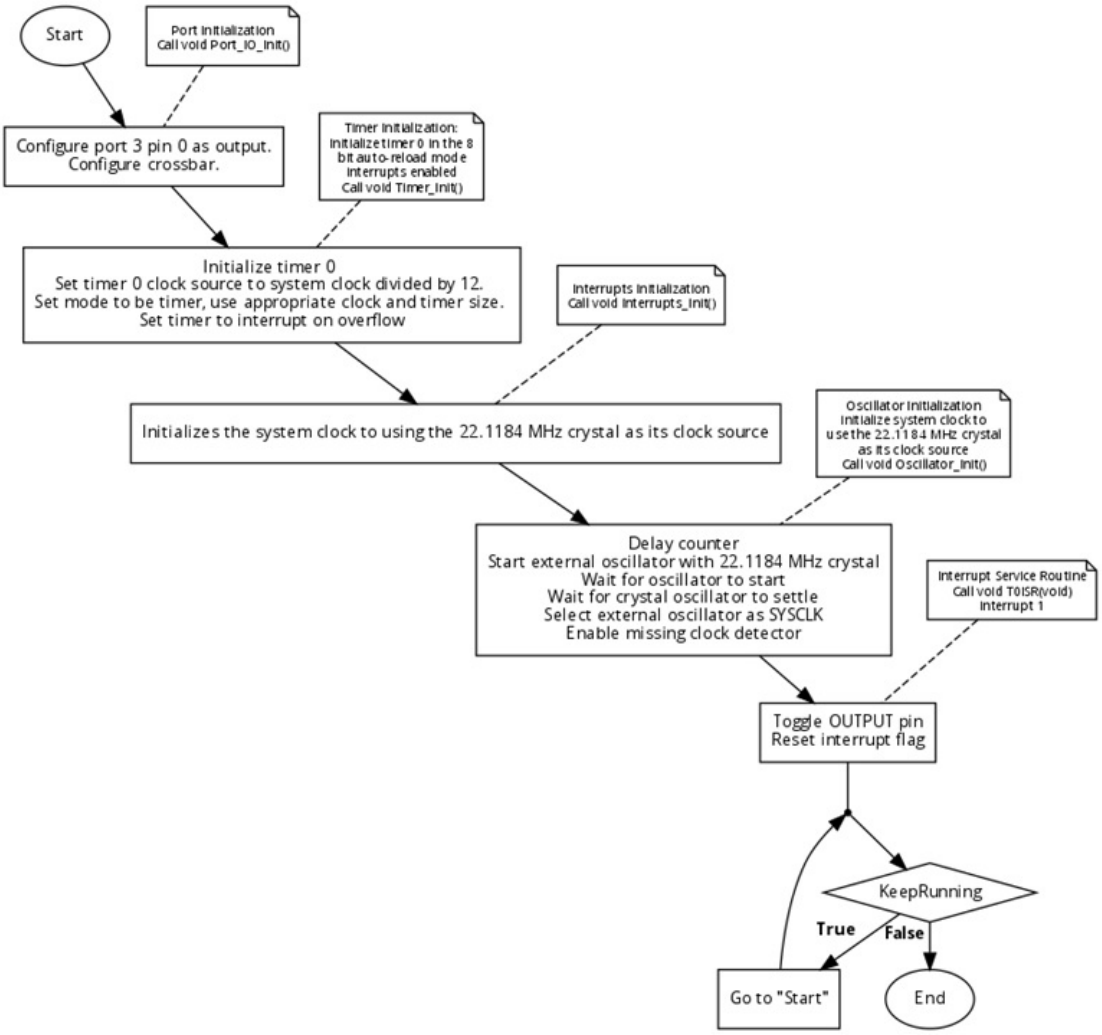

Figure 5.1: Flowchart for generating 10 kHz using c8051.

When the timer overflows, the 8-bit value is automatically reloaded with a defined value, rather than rolling over to zero –we do not have to check in code to see if the timer had overflowed and, if so, reset the timer to the defined value. This saves precious instructions of execution time to check the value and reload it.

The interrupt[3] method was used instead of polling. Although polling can monitor the

status of several devices and serve each of them as certain conditions are met, it is not an efficient use of the MCU. The advantage of interrupts over polling is that the second wastes much of the MCU's time by polling devices that do not need service. So in order to avoid tying down the MCU, interrupts are used.

5.2 Circuit using Altera DE2 board FPGA

We need to use a counter to count the number of faster clock pulses until half of the slower clock period has passed. For example, in the case of 10 kHz, the number of fast clock pulses that makes up one clock period of a slow clock cycle is 50000000/10000 = 5000. Since we want half a clock period, that's 5000/2 = 2500 for each half-cycle, the duration of each high or low.

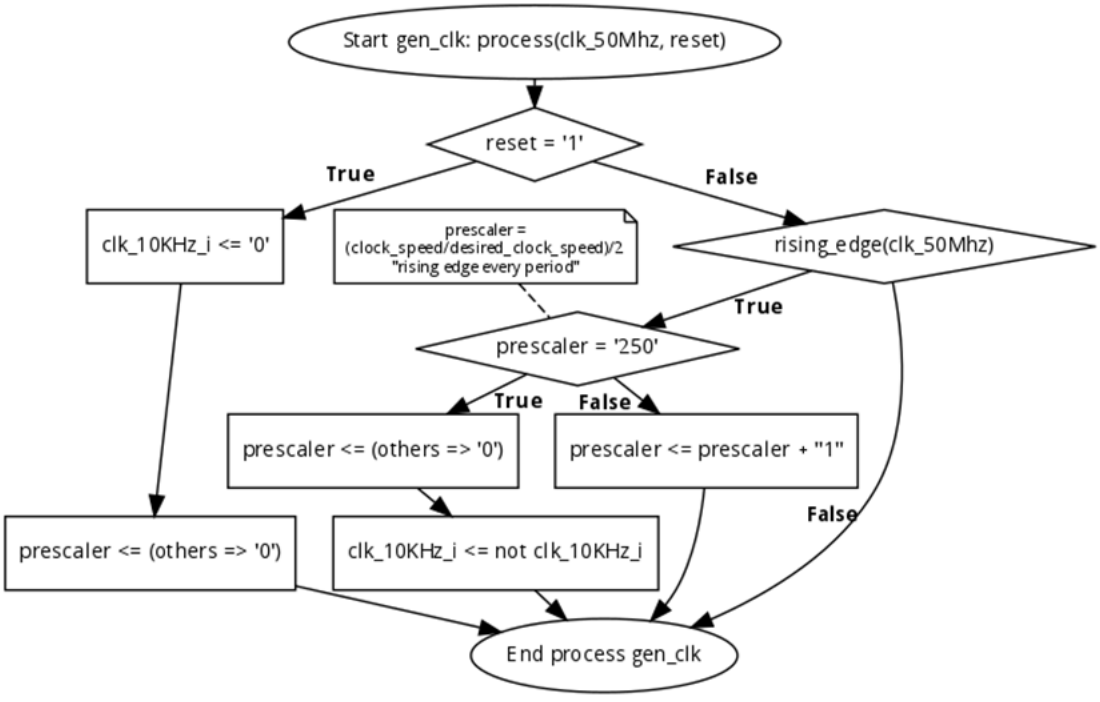

Figure 5.2: Flowchart for generating 10 kHz using DE2 board..

5.2.1 Compared Results (Preliminary data)

Figure 4.3 shows output results of generating 10 kHz square wave using the C8051 (top) and DE2 board (bottom). Flowcharts in figures 4.1 and 4.2 show the algorithms used for generating both signals.

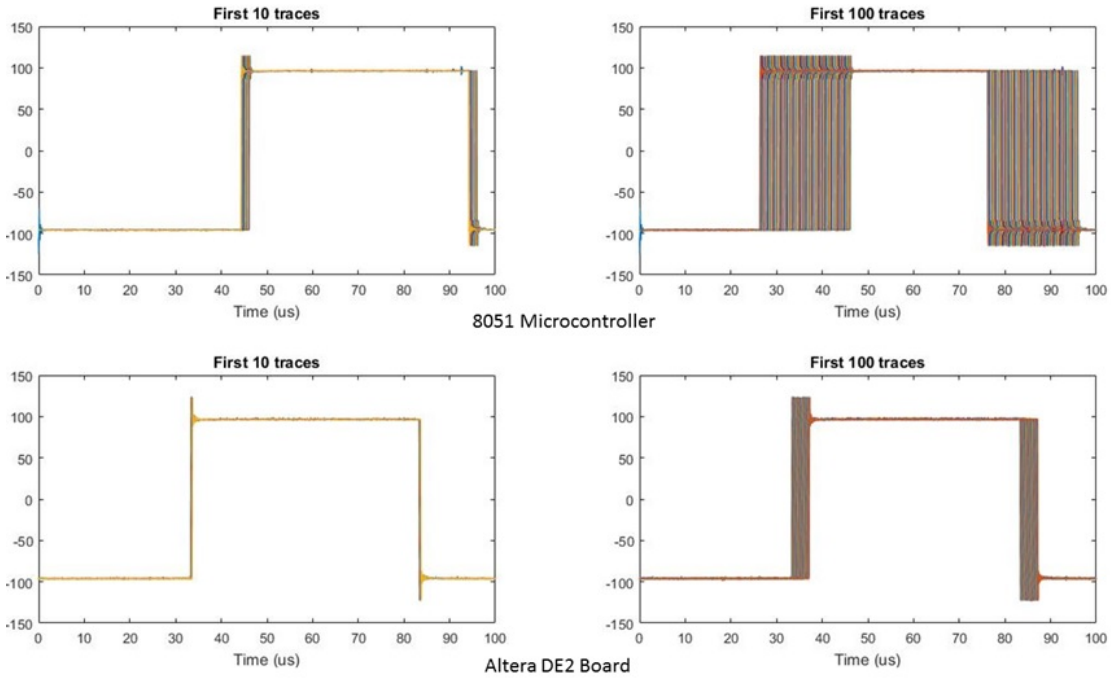

Figure 5.3: Compared Results (Preliminary data)

Although it is not obvious when looking at the "first 10 traces" of the FPGA, all the other traces show accumulation in delay. However, when zoomed in, delay seemed to increment by a constant value in the case of FPGA, while it was random in the case of the MCU. This can be due code implementation (in the case of the FPGA) and hardware limitation (in the case of the MCU). MUC's oscillator was not able to maintain generating a signal with a fixed value. This was also observed using the oscilloscope when probe was triggered

at a certain value: jitter was observed. Moreover, in the case FPGA, when Fourier transform was applied to data, output signals were not exactly 10 kHz. This might to do with the calculated counting value: it might need to be incremented/decremented by one, in order to hit the exact 10 kHz required value. Furthermore, errors where calculated: error in the case of 8051 Microcontroller was 0.16µs, and around 40ns in the case of FPGA. Therefore, we decided to proceed with the FPGA. We are planing on purchasing a new FPGA, with more modest specification, but same oscillator speed.

6 | Final Design and Implementation

For the final design, we decided to proceed with the FPGA. We purchased a new Altera board, DEo-nano with more modest specification, but same oscillator speed.

The DEo-Nano board is a compact-sized FPGA development platform. The board is designed to implement the Cyclone IV device With up to 22,320 LEs. It has a collection of interfaces including two external GPIO headers, on-board memory devices including SDRAM and EEPROM for larger data storage and frame buffering, as well as general user peripheral with LEDs and push-buttons.

6.1 Circuit using Altera DEo-nano board FPGA

The FPGA code was written in VHDL. QuartusII environment was used. Symbol block diagrams were produced out of the VHDL code, and large .bdf schematic was created.

Looking at the schematic in Figure 6.1, there are four blocks:

1. 1 Hz block: a Clock Divider that has one input: the 50MHz System Clock, and one output: 1 Hz. The block is used to trigger the stepper motor every one second.
2. Another 1 Hz bloc that is used to trigger the stepper motor every on 1 second. While the second 1 Hz is simply a copy if first, the output triggers the digitizer

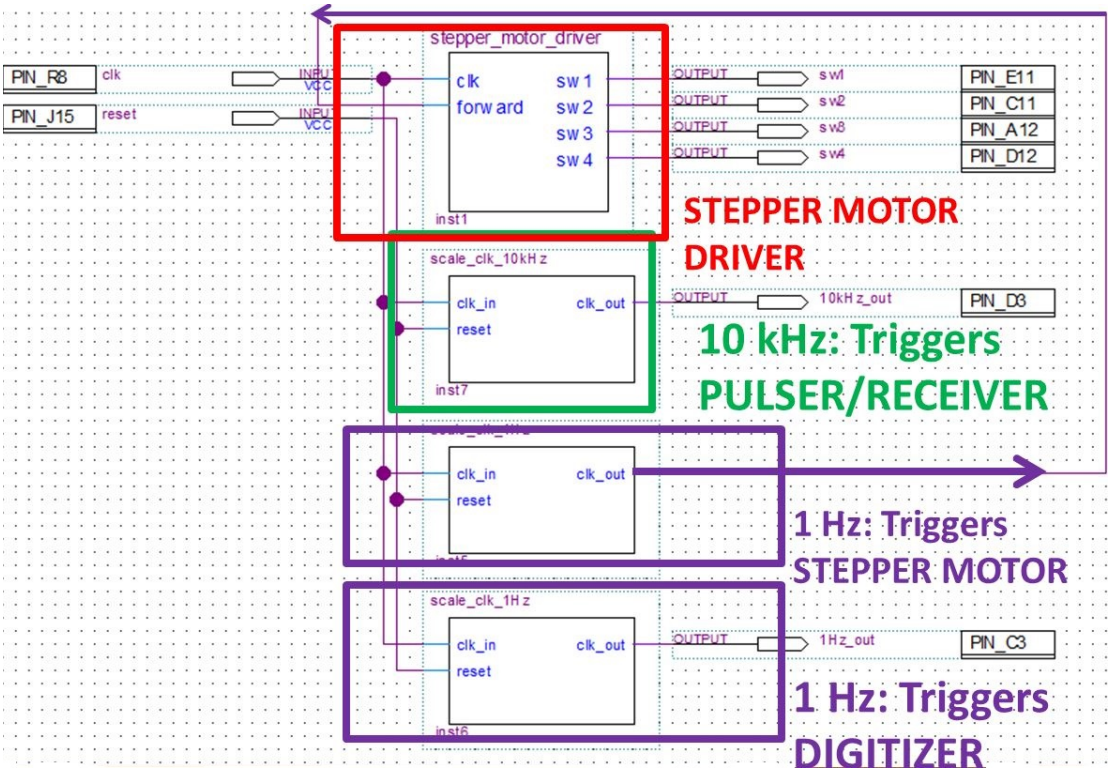

Figure 6.1: FPGA code schematic

–we can see modularity here: since FPGA is hardware oriented, parallel execution of code is easy.

3. The 10 kHz VHDL code was "copied and pasted", and only one variable was changed in the code (the increment value for the counter) –We can see how flexible it is to change values; we only need to change variables.
4. A simple bipolar stepping motor control unit

The simple bipolar stepping motor control unit is composed of multiple blocks. See Appendix for details. The main block is UPDOWNcounter (acts as an edge triggered pulse generator). It increments or decrements state if forward is '1' or '0' respectively. The rate

at which state is changed depends on the values of speed and the frequency of the system's base clock. The output converts the variable state to relevant control bits tied directly to the stepper motor driver.

6.2 SEQUENTIAL LOGIC IMPLEMENTATION FOR THE EDGE TRIGGERED PULSE GENERATOR

The design of sequential circuits in VHDL requires the use of the process statement. Process statements can include sequential statements that execute one after another as in conventional programming languages. However, for the logic synthesizer to be able to convert a process to a logic circuit, the process must have a very specific structure. We used process statements in this project to generate memory elements (flip-flops or registers).

```
process (clk, reset) is //sensitivity list
begin
    if reset = '0' then
        //-- do reset things
    elsif rising_edge(clk) then
        //-- read some signals, assign some outputs
    end if;
end process;
```

Figure 6.2: process in VHDL

In the process, an asynchronous reset was implemented. The reset signal was added in the sensitivity list, and design checked its value every time it changes, irrespective of what the clock is doing. Additionally, the expression clk'event is true when the value of clk has changed since the last time the process was executed. This is how "memory" is modeled in VHDL. The output is only assigned a value if clk changes and the new value is 1. When clk = 0 the output retains its previous value –It's necessary to check for clk=1 to distinguish

between rising and falling edges of the clock. The reset was toggled every one second using the 1 Hz block (which makes more of a synchronized reset).

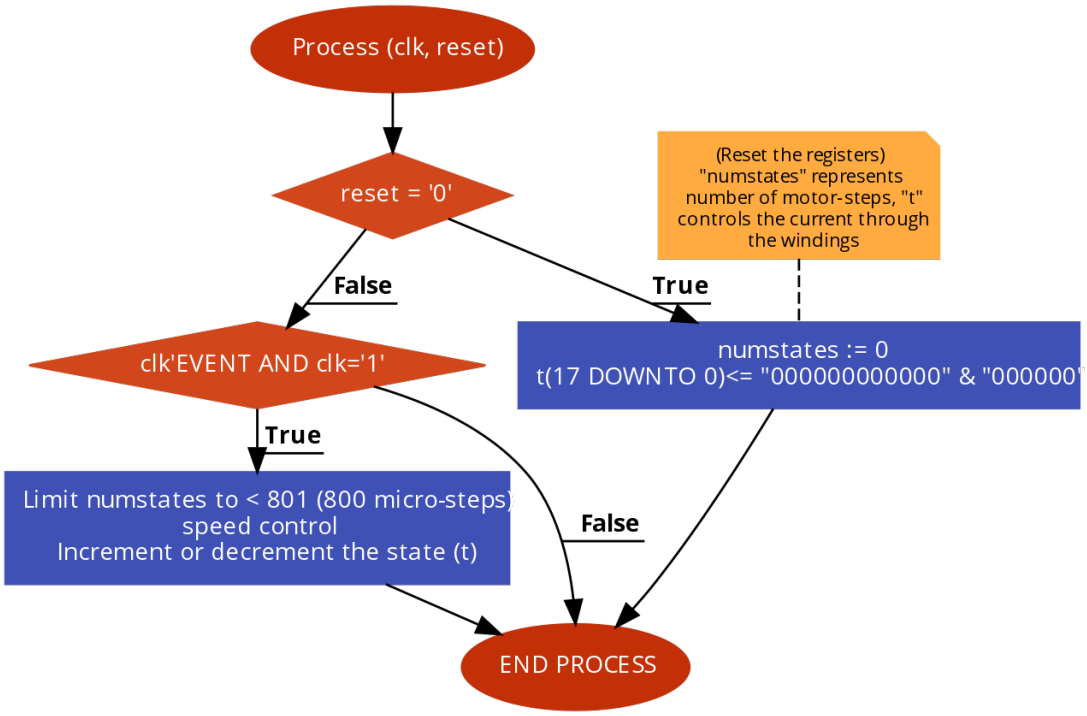

Figure 6.3: UPDOWNcounter block flowchart

6.3 The scaling factor

The frequency divider is a simple component which objective is to reduce the input frequency. The component is implemented through the use of the scaling factor and a counter. The scaling factor is the relation between the input frequency and the desired output frequency (knowing that the input frequency is 50MHz and provided we need an output frequency of 1Hz):

$$scale = \frac{f_{in}}{f_{out}} = \frac{50MHz}{1Hz} = 50000000$$

Therefore, the counter of the frequency divider generates the output signal of 1Hz each 50000000 cycles.

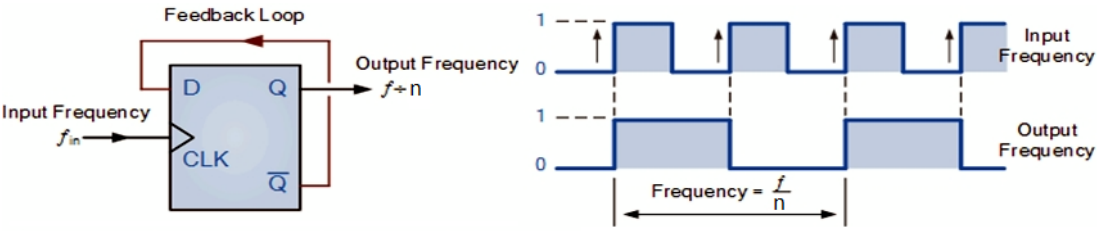

Figure 6.4: Frequency divider

The frequency divider process, generates the 1Hz signal by using a counter from 1 to 25000000 because a clock signal is a square wave with a 50 percent duty cycle. Since the counter begins at zero, the superior limit is 25000000 − 1. The reset signal is an essential part its function in this component is to restart the counter. Same process can be applied for generating other signals, simply by changing the scaling factor.

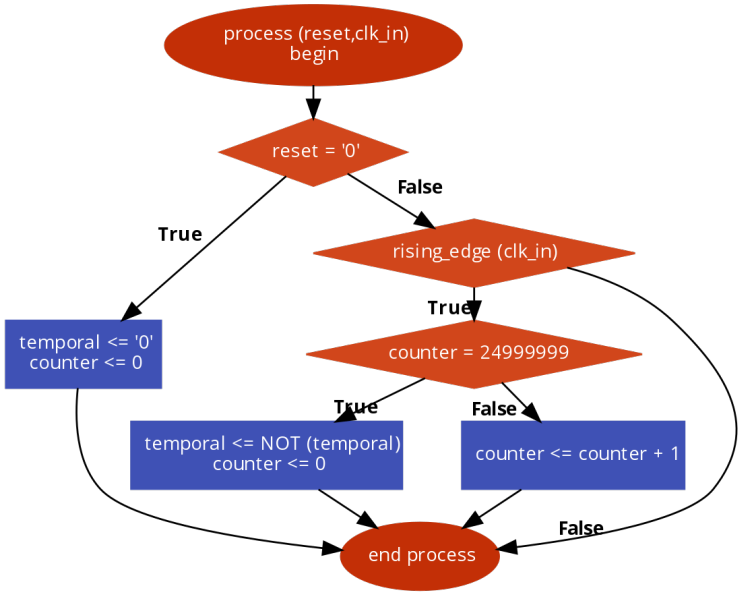

Figure 6.5: Frequency divider process, 1 Hz flowchart

6.3.1 Stepper Motor

There are several reasons to drive a stepping motor via micro-stepping, better accuracy and reduced effects of resonance (the inhibiting effect of matching stepping frequencies with mechanical load vibration frequencies) are just two of them. With a few additions to the UPDOWNcounter, the design elements can employ micro-stepping quite simply.

The FPGA code was built to control a 200 step per revolution two winding bi-polar motor (ROB-09238) via 64 states per step micro-stepping. It is a simple, but very powerful stepper motor with a 4-wire cable attached. It is a Bipolar Motor and features a 1.8 Step Angle (degrees), 2 Phase Rated Voltage of 12V, Rated Current of 0.33A and a Holding Torque of $2.3kg * cm$

Stepper Motor driver

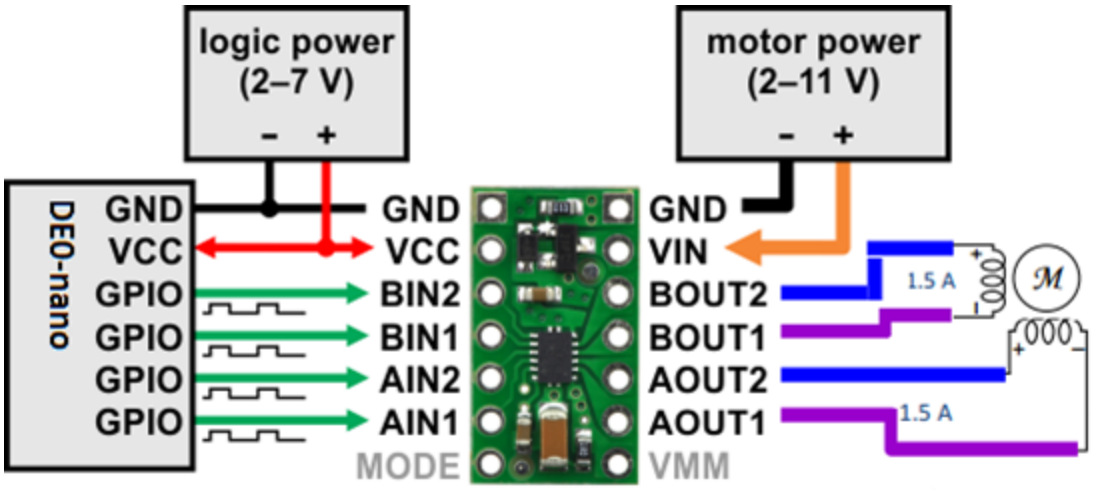

Figure 6.6: Stepper Motor driver

Figure 6.2 shows TI's DRV8835 dual motor driver connected to the circuit. It can deliver 1.2A per channel continuously (1.5 A peak) to a stepper motor. This driver is a great solution for powering up to two small, low-voltage motors, or one stepper motor. Operating voltage ranges from 0 V to 11 V. It has built-in protection against reverse-voltage, under-voltage, over-current, and over-temperature.

6.3.2 MATLAB

A script for using NI digitizer with MATLAB was written. It fetches Waveform through NI-SCOPE MATLAB Instrument Driver. The following flow chart shows how to acquire digital waveform from two channels of a National Instruments® NI-SCOPE driver. The full code can found in Appendix.

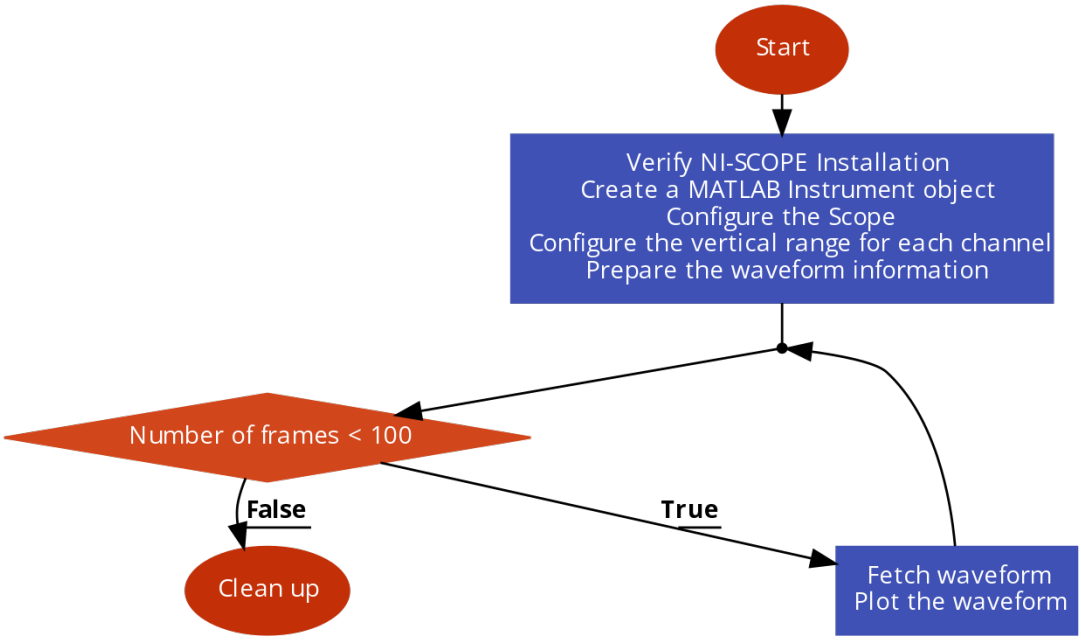

Figure 6.7: MATLAB flowchart

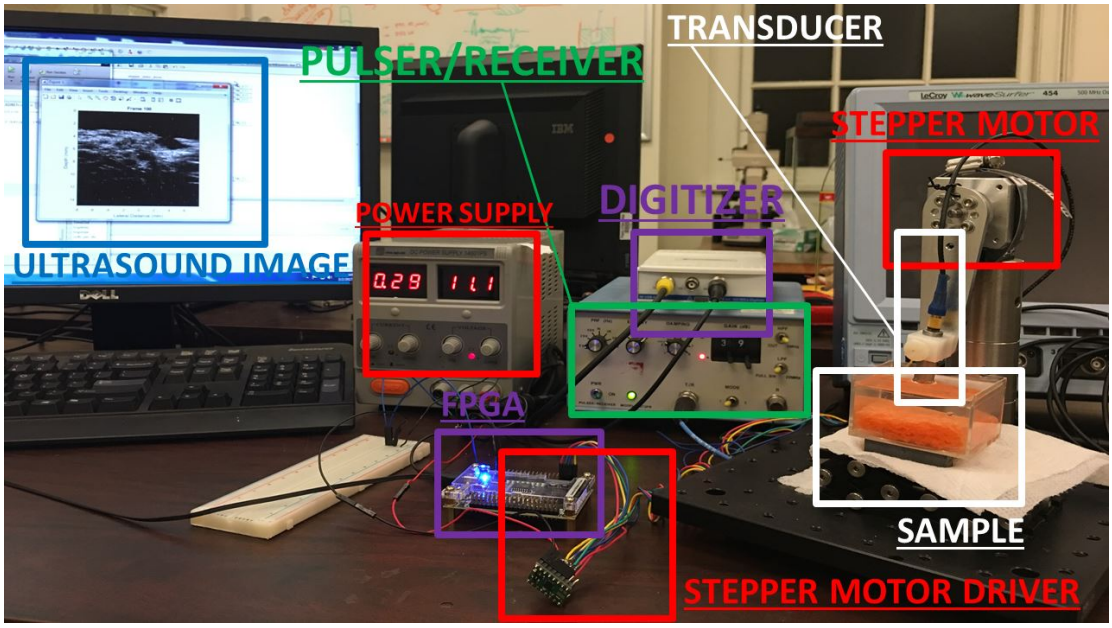

Figure 6.8: Setup of the system

In the final setup, the transducer reconverts the mechanical reflected pulse into an electrical signal that is then amplified and conditioned by the receiver section. The resulting waves are then acquired using the data acquisition device. The digitizer makes the waves data available for further analysis using the computer. The Computer processes the data and displays ultrasound images using a MATLAB. The Stepper Motor sweeps the transducer back and forth to scan across the sample.The Timing Electronics generate synchronous signal that triggers various parts of the system and controls the motor.

7 | Performance estimates and results

7.1 Power Consumption Estimated performance and its results

The pulser-receiver has a power cord that is directly connected to a socket. The Digitizer is USB powered. The DE0-Nano has many power scheme options including a USB mini-AB port, 2-pin external power header and two DC 5V pins, so we decided to go with the USB option.

The bi-polar motor (ROB-09238) requires a 2 Phase Rated Voltage of 12V and Rated Current of 0.33A. The TTI's DRV8835 stepper motor can deliver 1.5 A peak and has an operating voltage range from 0 V to 11 V.

While it could easily provide the motor with the needed current, the motor driver could barely support the motor with the needed voltage, due to its low operating voltage range –this was evident when we tried triggering the motor at high frequency signals from the FPGA. However, there were no problems when 1 Hz was used as a trigger.

Figure 4.3 shows output results of generating 10 kHz square wave using DEo-nano board, acquired using the Digitizer and processed using MATALAB.

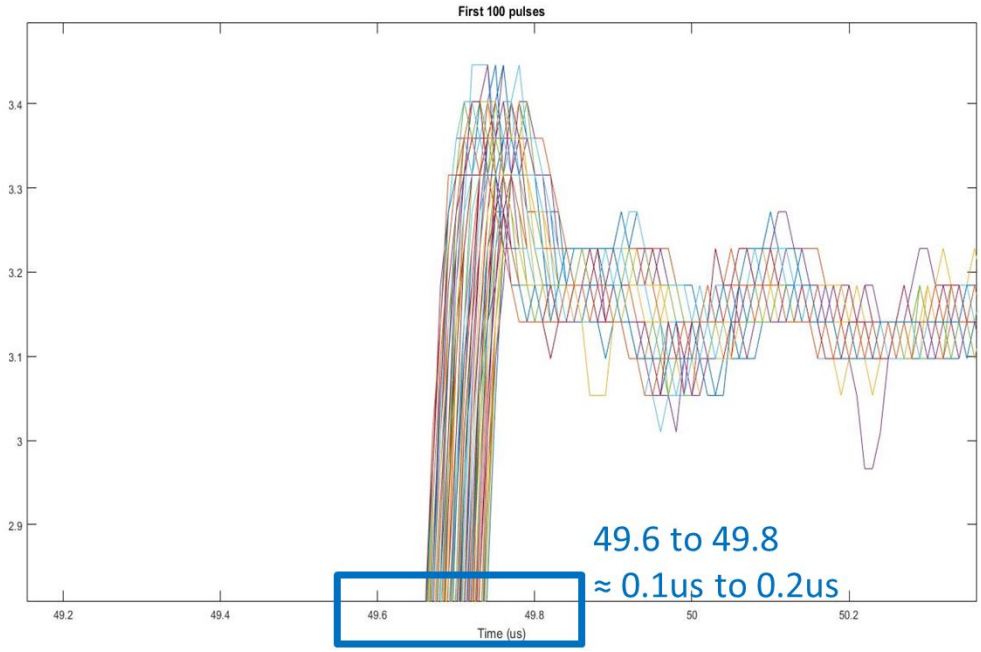

Figure 7.1: Generated Pulses Accuracy zoom

When zoomed in using MATALAB, the first 100 traces generated show accumulation in delay. This can be due to hardware limitation (the FPGA's oscillator) or the digitizer. The delay time between the first and last A-line, for the first 100 traces, was observed and turned out to be 0.1 to 0.2us. Which would be 0.4us to 0.8us in the case of 400 traces. We recall that a delay error of 1 us that results of 0.75mm shift (which is acceptable). See calculations earlier. Thus, error is within the accepted range.

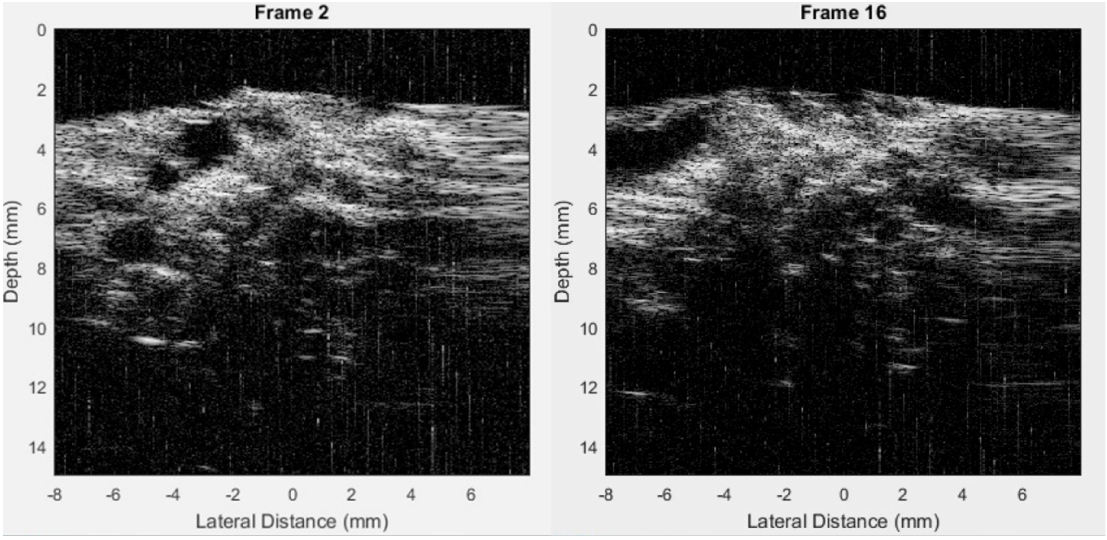

Figure 7.2: Ultrasound image of a sponge, obtained using the system

8 | Discussion, Conclusions, and Recommendations

This senior project was a successful upgrade of an older ultrasound imaging system. The system will allow student to gain access to a hands-on tool that can produce better understanding of ultrasound technology, the process of capturing images and electronics involved. We successfully manage to replace the timing electronics and trigger Data Acquisition, pulser-receiver and control the Stepper-motor. The accuracy requirement was met; the system generated signals were within Error limit (0.8us < 1us).

The system was also packed into a much smaller size. The modularity specification was met. Due to the use of FPGA we were able to produce adaptable Blocks and Reusable code. Finally, data was processed and reconstructed into ultrasound images using MATLAB.

Future work may involve implementing different Modes for triggering and acquiring data, and improve stepper motor driver by supporting other micro stepping modes.

A | Appendix

A.1 Cost Analysis

Tabulate component costs and compare to estimated cost and market cost where appropriate.

DEo-nano Development and Education Board http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=364	$120
EasyDriver - Stepper Motor Driver https://www.sparkfun.com/products/12779	$14.95
Stepper Motor with Cable https://www.sparkfun.com/products/9238	$14.95
Total	$150

Additional components are already provided by the faculty advisor overseeing this project: The Altera DE2 Development, Olympus 5073PR Pulser-Receiver, NI USB-5133 acquisition board, c8051 microcontroller and a computer.

A.2 Production schedule

Discuss the phases of the design and implementation of your project. (Pert charts may be appropriate in the discussion)

Recommend any improvements that could have been made in the scheduling and planning.

Fall Term	Winter Tern
1. Testing and adjusting design	1. Continued development with further testing
2. Algorithm development	2. Final project complete with desired testing outcomes
3. Finalize and order all parts	3. Incorporate final design into System
4. Detailed design description Update	4. Demo: Start imaging
5. web page	5. Final design report

A.3 FPGA code

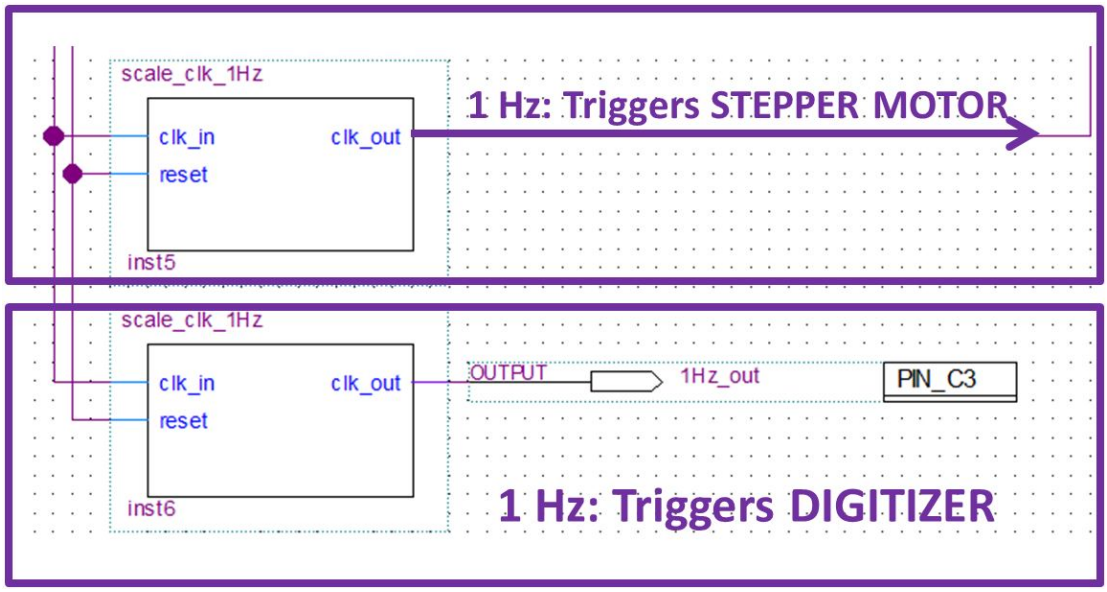

Figure A.1: 1 Hz Schematic; Clock Divider: 50MHz System Clock in, 1 Hz out.

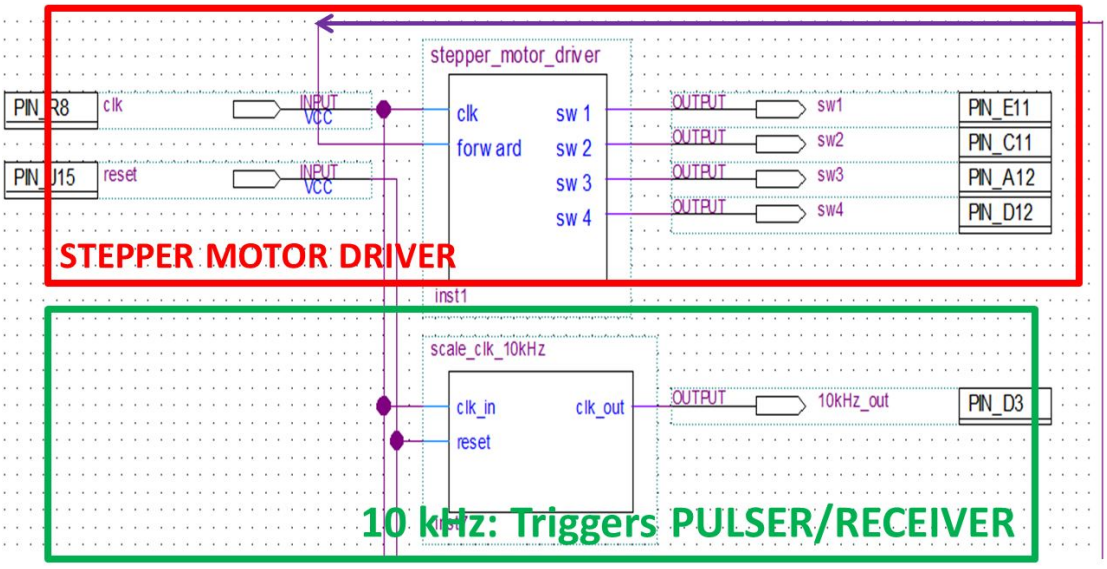

Figure A.2: 10 kHz and Stepper-Motor Driver Schematics

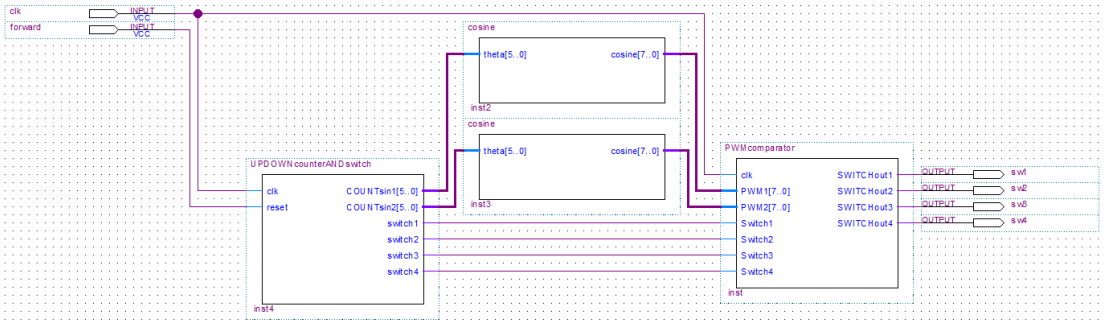

Figure A.3: stepper motor driver FPGA code

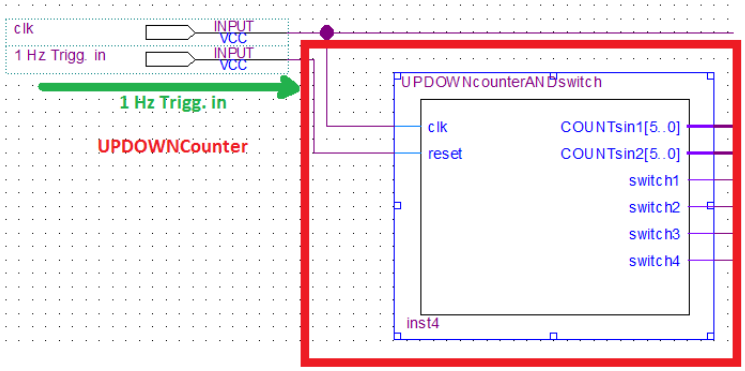

Figure A.4: UPDOWNCounter Schematic

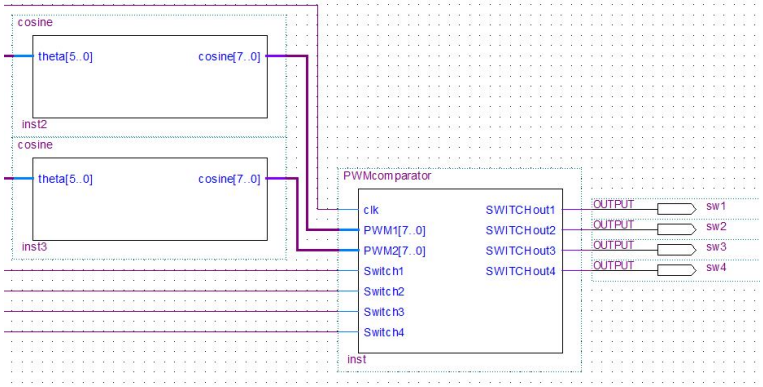

Figure A.5: Other parts for stepper motor control

```
1  --------------------------------------------------------------------------------
2  --
3  --   FileName:        UPDOWNcounterANDswitch.vhd
4  --
5  --   clk       - base clock for the system
6  --   COUNTsin1,
7  --   COUNTsin2 - two 6-bit counters whose values are bound at a 64 bit
8  --               difference (90 degree phase shift).
9  --   switch1,
10 --   switch2,
11 --   switch3,
12 --   switch4   - 4 bits that represent the polarity of COUNTsin1 and COUNTsin2,
13 --               these bits can be used directly to control a motor by half-stepping
14 --
15 --    numstates*0.028125 = Desired Angle (one forward sweep)
16 --   (dividend/256)*(clkfreq)*(numstates) = total_run_time
17 --    one_uStep_time = total_run_time\#ofuSteps = total_run_time\(Desired Angle*64)
18 --    200 step per revolution motor
19 --    micro-stepping (64)
20 --    The base clock is 50 MHz.
21 --
22 --   t - the internal counter
23 --   rightmost 8 bits: linear output to the sine table
24 --   ... the whole signal is a more accurate representation of the motor's actual position
25 --
26 --       for half-stepping, the rightmost 3 bits are all that are needed
27 --       for full-stepping, only the rightmost 2 bits determine output switches
28 --
29 --------------------------------------------------------------------------------
30 LIBRARY ieee;
31 USE ieee.std_logic_1164.all;
32 USE ieee.std_logic_unsigned.all;
33 ------------------------------------------
34 ENTITY UPDOWNcounterANDswitch IS
35 PORT(
36       clk       :IN  STD_LOGIC;
37       reset     :IN  STD_LOGIC;
38       COUNTsin1 :OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
39       COUNTsin2 :OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
40       switch1   :OUT STD_LOGIC;
41       switch2   :OUT STD_LOGIC;
42       switch3   :OUT STD_LOGIC;
43       switch4   :OUT STD_LOGIC
44     );
45 END UPDOWNcounterANDswitch;
46 ------------------------------------------
47 ARCHITECTURE arx OF UPDOWNcounterANDswitch IS
48    SIGNAL t :STD_LOGIC_VECTOR (17 DOWNTO 0):= "000000000000" & "000000";
49    SIGNAL speedreg  :STD_LOGIC_VECTOR (7 DOWNTO 0);
50    SIGNAL speedResolution        :STD_LOGIC_VECTOR (7 DOWNTO 0):= "11111111";
51
52 BEGIN
53    PROCESS (clk, reset)
54       VARIABLE timechunk :INTEGER RANGE 0 TO 2097151:= 0;
55       VARIABLE numstates :INTEGER RANGE 0 TO 2097151:= 0;
56       CONSTANT dividend  :INTEGER RANGE 0 TO 33554432:= 1280000;
57    BEGIN
58    IF reset = '0' THEN
59       numstates := 0;
60       t(17 DOWNTO 0)<= "000000000000" & "000000";
61    ELSIF (clk'EVENT AND clk='1') THEN
62       speedreg <= speedResolution(7 DOWNTO 0);--speed- control input directly proportional
```

```
      to the motor's speed
63          IF (numstates < 801) THEN--1600 ==> Desired Angle (one forward sweep)
64             IF (speedreg > 0) THEN
65             --timechunk increments at the same frequency of clk and corresponds with
66             --how much time (in periods of clk) is taken before state (t) is
67             --incremented. Speed, in short, increases how fast speed*timechunk
68             --increments and changes how quickly states are stepped through.
69                timechunk := timechunk + 1;
70                IF (CONV_INTEGER(speedreg)*timechunk >= dividend) THEN
71                   timechunk := 0;
72                   numstates := numstates + 1;
73                   --The following 5 code lines either increment or decrement the state (t).
74                   --needed to control the current through the windings.
75                   --The bit forward represents which direction the motor should turn.
76                   IF (numstates < 401) THEN
77                      t <= (t + 1);
78                   ELSE
79                      t <= (t - 1);
80                   END IF;
81                END IF;
82             END IF;
83          END IF;
84       END IF;
85       END PROCESS;
86       switch1   <= '1' WHEN t(7 DOWNTO 0) < 64 OR t(7 DOWNTO 0) > 191 ELSE '0';
87       switch2   <= '0' WHEN t(7 DOWNTO 0) < 64 OR t(7 DOWNTO 0) > 191 ELSE '1';
88       switch3   <= '1' WHEN t(7 DOWNTO 0) < 128 ELSE '0';
89       switch4   <= '0' WHEN t(7 DOWNTO 0) < 128 ELSE '1';
90
91       COUNTsin1 <= t(5 DOWNTO 0) WHEN t(6) = '0' ELSE NOT t(5 DOWNTO 0);
92       COUNTsin2 <= NOT t(5 DOWNTO 0) WHEN t(6) = '0' ELSE t(5 DOWNTO 0);
93    END arx;
94    --------------------------------------------------------------------------------
95    --
96    --   FileName:        cosine.vhd
97    --   Dependencies:    UPDOWNcounterANDswitch.vhd
98    --
99    --   theta - input/argument of cosine function
100   --   cos   - value of cosine function
101   --
102   --   This sinusoid lookup table is only a quarter of the symmetric sinusoid it
103   --   represents. It has also been distorted to match the motor.
104
105   -- sinusoid function or lookup table.
106   -- a lookup table saves time, processing power, and possible chip space for low resolutions
107   -- that are more than adequate for micro-stepping motor control.
108   --
109   -- The input resolution determines how many states there are per sinusoid
110   -- the output resolution is the same as the resolution of the pulse width.
111   -- This element's output is a sinusoidal value for the comparator process in PWM.
112   --
113   --------------------------------------------------------------------------------
114   LIBRARY ieee;
115   USE ieee.std_logic_1164.all;
116   USE ieee.std_logic_unsigned.all;
117   USE ieee.std_logic_arith.all;
118   ------------------------------------------
119   ENTITY cosine IS
120       PORT
121       (
122         theta  :IN  STD_LOGIC_VECTOR (5 DOWNTO 0);
123         cosine :OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
```

```
124       );
125   END cosine;
126   ----------------------------------------
127   ARCHITECTURE arx OF cosine IS
128     SIGNAL cos :INTEGER RANGE 0 TO 255;
129   BEGIN
130     cosine <= CONV_STD_LOGIC_VECTOR(cos, 8);
131     WITH CONV_INTEGER(theta) SELECT
132       cos <= 255 WHEN 0,
133              255 WHEN 1,
134              255 WHEN 2,
135              255 WHEN 3,
136              254 WHEN 4,
137              254 WHEN 5,
138              253 WHEN 6,
139              252 WHEN 7,
140              252 WHEN 8,
141              251 WHEN 9,
142              250 WHEN 10,
143              248 WHEN 11,
144              247 WHEN 12,
145              246 WHEN 13,
146              244 WHEN 14,
147              243 WHEN 15,
148              241 WHEN 16,
149              239 WHEN 17,
150              237 WHEN 18,
151              235 WHEN 19,
152              233 WHEN 20,
153              231 WHEN 21,
154              229 WHEN 22,
155              226 WHEN 23,
156              224 WHEN 24,
157              221 WHEN 25,
158              218 WHEN 26,
159              215 WHEN 27,
160              212 WHEN 28,
161              209 WHEN 29,
162              206 WHEN 30,
163              203 WHEN 31,
164              199 WHEN 32,
165              196 WHEN 33,
166              192 WHEN 34,
167              188 WHEN 35,
168              184 WHEN 36,
169              180 WHEN 37,
170              176 WHEN 38,
171              172 WHEN 39,
172              168 WHEN 40,
173              163 WHEN 41,
174              159 WHEN 42,
175              154 WHEN 43,
176              149 WHEN 44,
177              144 WHEN 45,
178              139 WHEN 46,
179              134 WHEN 47,
180              128 WHEN 48,
181              123 WHEN 49,
182              117 WHEN 50,
183              111 WHEN 51,
184              105 WHEN 52,
185              98 WHEN 53,
```

```
186             92 WHEN 54,
187             85 WHEN 55,
188             78 WHEN 56,
189             71 WHEN 57,
190             63 WHEN 58,
191             55 WHEN 59,
192             46 WHEN 60,
193             36 WHEN 61,
194             25 WHEN 62,
195             12 WHEN OTHERS;
196   END arx;
197   --------------------------------------------------------------------------------
198   --
199   --   FileName:        PWMcomparator.vhd
200   --   Dependencies:    cosine.vhd
201   --                    UPDOWNcounterANDswitch.vhd
202
203   -- fixed frequency pulse width modulation (PWM) base counter
204   --
205   -- The frequency at which the counter resets is the primary PWM frequency while the
      frequency at which it increments is the base frequency.
206   -- The primary frequency should be limited to somewhere between 10KHz and 30KHz.
207   -- The base frequency should be equal to the primary PWM frequency * resolution of the
      pulse width.
208   -- This frequency may not be fast enough to match the stepping of states through the
      sinusoid,
209   -- but at such high frequencies the advantages of switching to full-stepping the motor
      should be considered.
210   -- The output of the counter serves as an input to a comparator ...
211   --    This controls switching based on whether the PWM value is greater than or less than
      another input value.
212   --
213   --   clk         - base clock for the system
214   --   PWM1,
215   --   PWM2        - 8-bit values from entity cosine that determine pulsewidth
216   --   Switch1,
217   --   Switch2,
218   --   Switch3,
219   --   Switch4     - 4 bits that represent the polarity of COUNTsin1 and COUNTsin2
220   --   SWITCHout1,
221   --   SWITCHout2,
222   --   SWITCHout3,
223   --   SWITCHout4  - 4 pulse width modulated bits that control a unipolar motor
224   --
225   --   COUNTPWM    - 8-bit PWM count to act as a sawtooth function for PWM
226   --
227   --   dividend - constant that determines frequency of PWM
228   --     dividend = (clkfreq)/
229   --                (PWMfreq*PWresolution)
230   --
231   --     In this example: a 24 KHz PWM is intended with 8 bits of resulution for
232   --     the pulse's width. The base clock runs at 50 MHz.
233   --
234   --     dividend = 50,000,000/(24,000*256) = 8
235   --------------------------------------------------------------------------------
236   LIBRARY ieee;
237   USE ieee.std_logic_1164.all;
238   USE ieee.std_logic_unsigned.all;
239   ------------------------------------------
240   ENTITY PWMcomparator IS
241   PORT(
242         clk        :IN  STD_LOGIC;
```

```
243         PWM1       :IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
244         PWM2       :IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
245         Switch1    :IN  STD_LOGIC;
246         Switch2    :IN  STD_LOGIC;
247         Switch3    :IN  STD_LOGIC;
248         Switch4    :IN  STD_LOGIC;
249         SWITCHout1 :OUT STD_LOGIC;
250         SWITCHout2 :OUT STD_LOGIC;
251         SWITCHout3 :OUT STD_LOGIC;
252         SWITCHout4 :OUT STD_LOGIC
253       );
254   END PWMcomparator;
255   -----------------------------------------
256   ARCHITECTURE arx OF PWMcomparator IS
257     SIGNAL COUNTPWM :STD_LOGIC_VECTOR (7 DOWNTO 0);
258   BEGIN
259
260     PROCESS (clk)
261       VARIABLE timechunk :INTEGER RANGE 0 TO 2097151:= 0;
262       CONSTANT dividend  :INTEGER RANGE 0 TO 1048575:= 8;
263     BEGIN
264       IF (clk'EVENT AND clk='1') THEN
265         timechunk := timechunk + 1;
266         IF (timechunk >= dividend) THEN
267           timechunk := 0;
268           COUNTPWM <= (COUNTPWM + 1);
269         END IF;
270       END IF;
271     END PROCESS;
272
273     SWITCHout1 <= '1' WHEN (COUNTPWM <= PWM1 AND Switch1 = '1') ELSE '0';
274     SWITCHout2 <= '1' WHEN (COUNTPWM <= PWM1 AND Switch2 = '1') ELSE '0';
275     SWITCHout3 <= '1' WHEN (COUNTPWM <= PWM2 AND Switch3 = '1') ELSE '0';
276     SWITCHout4 <= '1' WHEN (COUNTPWM <= PWM2 AND Switch4 = '1') ELSE '0';
277   END arx;
278   --------------------------------------------------------------------------------
279   --
280   --   FileName:        scale_clk_10kHz.vhd
281   --
282   --   clk       - base clock for the system
283   --
284   --------------------------------------------------------------------------------
285   library IEEE;
286   use IEEE.STD_LOGIC_1164.ALL;
287
288   entity scale_clk_10kHz is
289       Port (
290           clk_in : in  STD_LOGIC;
291           reset  : in  STD_LOGIC;
292           clk_out: out STD_LOGIC
293       );
294   end scale_clk_10kHz;
295
296   architecture Behavioral of scale_clk_10kHz is
297       signal temporal: STD_LOGIC;
298       signal counter : integer range 0 to 2499 := 0;
299   begin
300       frequency_divider: process (reset, clk_in) begin
301           if (reset = '0') then
302               temporal <= '0';
303               counter <= 0;
304           elsif rising_edge(clk_in) then
```

```
305                 if (counter = 2499) then
306                     temporal <= NOT(temporal);
307                     counter <= 0;
308                 else
309                     counter <= counter + 1;
310                 end if;
311             end if;
312         end process;
313 
314         clk_out <= temporal;
315     end Behavioral;
316     ----------------------------------------------------------------------------------
317     --
318     --   FileName:        scale_clk_1Hz.vhd
319     --
320     --   clk       - base clock for the system
321     --
322     ----------------------------------------------------------------------------------
323     library IEEE;
324     use IEEE.STD_LOGIC_1164.ALL;
325 
326     entity scale_clk_1Hz is
327         Port (
328             clk_in : in  STD_LOGIC;
329             reset  : in  STD_LOGIC;
330             clk_out: out STD_LOGIC
331         );
332     end scale_clk_1Hz;
333 
334     architecture Behavioral of scale_clk_1Hz is
335         signal temporal: STD_LOGIC;
336         signal counter : integer range 0 to 24999999 := 0;
337     begin
338         frequency_divider : process (reset, clk_in) begin
339             if (reset = '0') then
340                 temporal <= '0';
341                 counter <= 0;
342             elsif rising_edge(clk_in) then
343                 if (counter = 24999999) then
344                     temporal <= NOT(temporal);
345                     counter <= 0;
346                 else
347                     counter <= counter + 1;
348                 end if;
349             end if;
350         end process;
351 
352         clk_out <= temporal;
353     end Behavioral;
```

```
% test_niscope_011817.m
% test script for using NI digitizer with MATLAB (2 channels)
% Can adjust sampling rate (max=1e8) and number of samples (max=2e6)
% NOTE: This assumes the 'niscope.mdd' driver has already been created
% in the default MATLAB directory using the "makemid('niscope');" command.

% Create Instrument Object
ictObj=icdevice('niscope.mdd','DAQ::Dev1','optionstring','simulate=false');
connect(ictObj);
disp('Device connected!');
%disp(ictObj);

% Configure the scope
configuration=ictObj.Configuration;
invoke(configuration,'autosetup');

% Configure the vertical range for each channel
volts_per_div=0.02;    % use 1 for digital pulses, use 0.02 for ultrasound
Range = 10*volts_per_div;
Offset = 0;
Coupling = 1;
ProbeAttenuation = 1;

% Configure Channel 0
invoke(ictObj.Configurationfunctionsvertical,'configurevertical','0', ...
    Range,Offset,Coupling,ProbeAttenuation,true);

% Configure the horizontal parameters
SamplingRate = 100e6;
numSamples = 4e6;
N=10000;        % number of samples per a-line
M=400;          % number of a-lines
c=1.5;          % speed of sound (mm/us)
dt=1/SamplingRate*1e6; % sampling interval in us
dz=c*dt/2;
dx=.050;

% set up the digital trigger
% configuretriggerdigital(source = 'VAL_PFI_1', edge = 1 (pos), delay = 0,
% offset = 0);
trigslope=1;
trigdelay=0.005;
invoke(ictObj.Configurationfunctionstrigger,'configuretriggerdigital', ...
    'VAL_PFI_1',trigslope,0,trigdelay);
refPos=0;
invoke(ictObj.Configurationfunctionshorizontal, ...
    'configurehorizontaltiming',SamplingRate,numSamples,refPos,1,1);

% Prepare the waveform information
numChannels = 1;
channelList = '0';
```

```
% parameters for dB image
nstart=5000+1500;
nstop=nstart+2000;
deltan=nstop-nstart;
env_dB=zeros(deltan,0.8*M);

figure;
colormap(gray);
z= [0:(nstop-nstart)-1]'*dz;
x= ([0:(0.8*M-1)]-0.4*M)*dx;

% waveform parameters
waveformInfo=[struct];
ch0_data = zeros(numChannels * numSamples,1,'int8');
TimeOut = 10;   % seconds

% start loop here ========================
for m=1:20
% fetch the ultrasound data
invoke(ictObj.Acquisition,'initiateacquisition');
[ch0_data, waveformInfo]=invoke(ictObj.Acquisition,'fetch', ...
    channelList,TimeOut,numSamples, ch0_data, waveformInfo);

% reshape into 2D matrix
ch0_data=reshape(ch0_data,numSamples,numChannels);
ch0_data=reshape(ch0_data,N,M);

% select portion of ch0_data for envelope detection ========
env_dB=20*log10(abs(hilbert(ch0_data((1+nstart):nstop,0.1*M:(0.9*M-1)))));

% display dB ultrasound image ==========
max_dB=max(max(env_dB));
imagesc(x,z,env_dB,[-30,0]+max_dB);
title(sprintf('Frame %d',m));
xlabel('Lateral Distance (mm)');
ylabel('Depth (mm)');
axis equal;     % equal tick mark spacing
axis image;     % crop
drawnow;

ch0_data=zeros(numChannels * numSamples,1,'int8');
% end loop here? ==========================
end

% Clean Up ========================
disconnect(ictObj);
delete(ictObj);
clear ictObj;
clear waveformInfo;
```

References

[1] 5072pr, 5073pr, 5077pr. http://www.olympus-ims.com/en/5072pr/#!cms[tab]=%2F5072pr%2Foverview. [Online; accessed 19-Nov-2016].

[2] 5072pr, 5073pr, 5077pr. http://www.olympus-ims.com/en/5072pr/. [Online; accessed 19-Nov-2016].

[3] 8051 interrupts. http://what-when-how.com/8051-microcontroller/8051-interrupts/.

[4] Support documents: (technicaldocs) c8051f02x manual. https://www.silabs.com/Support%20Documents/TechnicalDocs/C8051F02x.pdf. [Online; accessed 19-Nov-2016].

[5] Abdel-Karim, R. & Helou, S. H. (2013). The future of engineering education in palestine. *Procedia-social and behavioral sciences*, 102, 482–489.

[6] Abu-Duhou, I. (1996). Problems and perspectives for the palestinian educational system. http://www.pij.org/details.php?id=566. [Online; accessed 19-Nov-2016].

[7] Cortada, J. (2016). *All the Facts: A History of Information in the United States Since 1870*. Oxford University Press.

[8] Cyclone, I. (2006). Fpga starter development board. *Reference manual. San-Jose.: Altera*, (pp. 1–6).

[9] EACEA (2008). Bworld robot control software. http://eacea.ec.europa.eu/tempus/participating_countries/overview/oPt.pdf. [Online; accessed 19-Nov-2016].

[10] Felder, R. M. & Silverman, L. K. (1988). Learning and teaching styles in engineering education. *Engineering education*, 78(7), 674–681.

[11] Hashweh, M. Z., Hashweh, M., & Berryman, S. E. (2003). *An Assessment of Higher Education Needs in the West Bank and Gaza*. United States Agency for International Development.

[12] Kane, S. A. (2009). *Introduction to physics in modern medicine*. Taylor & Francis.

[13] Koen, B. V. (1994). Toward a strategy for teaching engineering design. *Journal of Engineering Education*, 83(3), 193–201.

[14] Lamancusa, J., Torres, M., Kumar, V., & Jorgensen, J. (1996). Learning engineering by product dissection. *age*, 1, 1.

[15] Mestre, J. P. (2001). Cognitive aspects of learning and teaching science.

[16] MOHE (2015). The ministry launched the third strategic plan 2014-2019. https://www.mohe.pna.ps/news?p=articles&news=135. [Online; accessed 19-Nov-2016].