Chapter 2

Background and Motivation

2.1 Applications of DNA Sequencing

There are a large number of motivations for sequencing DNA. In the medical field, genomics has applications in both research and applied medicine. One example where DNA sequencing has been used productively in applied medicine is for screening for the BRCA mutations. Through testing the BRCA1 and BRCA2 genes for mutations, it is possible to accurately predict if a patient is likely to develop cancer later in life. The US Preventative Services Task Force currently recommends screenings for women who are known to have a family history of breast cancer, around 2% of the population [10]. This type of application relies on exome sequencing, where only a small subsection of DNA is sampled to identify a risk factor for a certain disease.

The research applications of sequencing have been clear for some time with the Human Genome Project launching in 1990 [11]. With it's primary goals accomplished in 2003, this project has opened numerous areas for research, showing possible applications such as understanding genotypes of viruses for targeted treatments and also the advancement of other less related areas such as forensics. The sequencing of a full human genome also has significant impact on the nature of DNA sequencing, as it allows a much wider area of mapping sequencing. Mapping sequencing uses the fact that the structure of a piece of DNA is known in order to simplify the sequencing process. Exome sampling mentioned earlier is an example of this.

DNA sequencing also plays an important role in ecology research where bio-diversity and historical biological characteristics can both be indicated by DNA samples and used to further current research. This field is particularly cost sensitive due to the enormous variety of DNA sampling that needs to be carried out and also presents some of the most complex problems for DNA sampling [12].An example of this is the Polychaos Dubium which has 670 Giga base pairs (Gbp), dwarfing the size of the human genome at 3Gbp [13]. Due to the complexity of the operations we will see in the DNA sequencing process this presents important problems in sequencing [14]. Often this area often requires de novo sequencing techniques, using no prior knowledge of the genome in contrast to human samples where the general structure of the genome is known and can be used to simplify sequencing. These de novo techniques present a significantly different problem set to exome sequencing where comparison against existing data is impossible.

Ecology research has significant practical applications within agriculture, where

studies such as the sequencing of the Oil Palm genome are currently being undertaken in order to identify important genetic traits such as drought resistance and yield. It is in areas like these where fast sequencing of particularly long genetic sequences have practical applications, which acts as a driving motivation for producing a scalable architecture for this problem. Genetically Modified Organisms (GMO) are currently a growing trend in agriculture, with some companies in this area yielding over $2Billion operating income per year [15].

In order to satisfy the growing demand for both full de novo sequencing and exome sequencing, there has been a large amount of research in the area. The result of this research is a large array of techniques and technologies designed to lower the cost and raise the speed of sequencing. In the last decade this has produced a significant advance with a set of Next Generation Sequencing technologies.

2.2 The DNA Sequencing Process

2.2.1 The Structure of DNA

Deoxyribonucleic acid is a molecule formed of four nucleotides (guanine, adenine, thymine, and cytosine), the structure of the molecule is a double helix with bonds formed between complementary pairs of nucleotides (guanine bonding with cytosine, adenine bonding with thymine) [16]. These bonds are referred to as base pairs (bps) and are commonly used to quantify the size of the molecule. For example, the human genome is 3 billion base pairs long. The structure of DNA tends to be extremely long sequences. This structure has important ramifications on the sequencing techniques as discussed below.

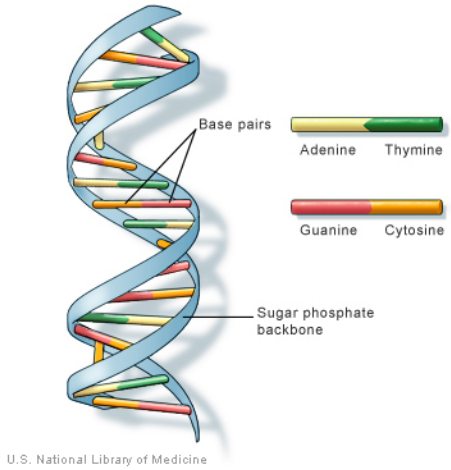

Figure 2.1: The Structure of DNA [1]

2.2.2 DNA Detection Technologies

One of the most commonly used DNA sequencing technologies is pyroseqeuncing, with commercial tools already in use such as the Roche GS20 sequencer [17]. Pyrosequencing is a synthesis method, this means that the method involves recombining a single strand of DNA with it's complementary base pairs. The full process requires a double helix of DNA first to be split into 2 single strands. Once this is done, one of the strands is taken and used to recombine to the full double helix. This involves the use of a primer, containing the four nucleotides in a sequence being introduced along the single strand. When one of the bases in the primer interacts with its complementary pairing a reaction occurs. If a base repeats consecutively, the reaction occurs twice and the byproducts of the reaction are magnified in amplitude. An example of this process is shown in figure 2.2.

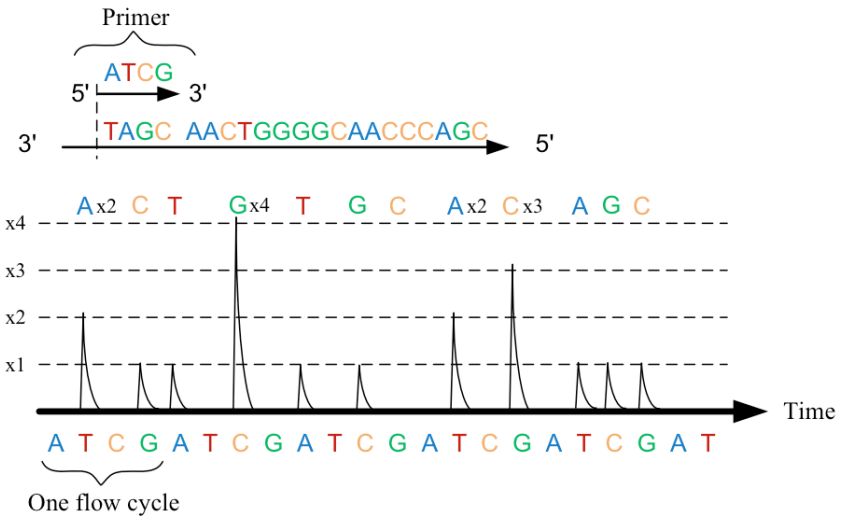

Figure 2.2: Pyrosequencing, Diagram by Y Hu [2]

This reaction releases both pyrophosphate and a positively charged hydrogen ion. In some approaches the pyrophosphate is detected using optical methods and the timing of the reaction could be used to identify which base. In newer technologies such as that introduced by Rothberg in 2011, non-optical techniques have been developed which allow much denser packing of detectors on a single chip [8]. These techniques use transistor technologies called ion-sensitive field-effect transistors (ISFETS). These ISFET devices can sense a change in the pH of a surrounding solution, the pH of the solution fluctuates when a recombination of a base with its complementary pair happens. The change in pH is sensed by the ISFET and as the supplied base is known, the base in the sequence can be identified [18].

The advantage of methods such as these is the density of detectors, allowing massively parallel sequencing to take place, Rothberg's application had 1.2 million wells on a

single chip, each able to detect a nucleotide in parallel. The disadvantage of this system however, is the detection time. The synthesis process involves the introduction of chemicals to the DNA sequence and then for a reaction to occur and the resultant chemicals to reach the detector. This is relatively slow process, taking up to 4 seconds for a single detection cycle to occur [8]. As a result, any practical application of this approach would need to use a massively parallel short read length approach. Due to this fact, a chemical process has been adopted to duplicate the DNA multiple times, cut it into small chunks and then use the overlaps of these chunks to recombine the sub-sequences into the full sequence. All NGS technologies share this limitation of read lengths being orders of magnitude shorter than even the shortest genome [19]. This is not a trivial computational problem and a variety of assembly algorithms have been proposed as a solution to this.

2.2.3 Assembly Algorithms

The problem of recombining sub-sequences into a single full sequence is computationally expensive as all overlaps of a sequence must be identified. A number of methods have been used for this, however there are three main groups.

The Overlap-Layout-Concensus (OLC) algorithm does an all against all comparison of the sub-sequences and produces a graph of where the sequences overlap, a graph can then be produced organising the sub-sequences into order by how they overlap, at this point it is relatively computationally simple to produce a best-guess full sequence. With enough data it is possible to confidently produce a full sequence that is known to be correct. This method is computational expensive due to the all-against-all comparison required at the beginning of the process. Once the comparison is done, the recombination is a relatively trivial problem to solve.

The Greedy Graph algorithm shares the first step with the OLC method, using an all-against-all comparison to build a graph to be solved. The disadvantage of the Greedy algorithm however, is that at each stage it picks a locally optimal solution, this means that the order of detection can determine the likelihood getting a given result [2].

A significantly different approach to assembly is the De Bruijn method. Used in the Solexa and SOLiD platforms [19], it produces a graph of sequences where each sequence is a node on the graph and any overlaps above a threshold are directed links on the graph. The advantage of this system is that it does not require the same all against all comparison, as overlaps are built up iteratively, however due to the fact the algorithm srotes a large amount of data on sub-sequences it uses a large amount of memory relative to the other algorithms.

The problem of assembly algorithms is therefore a matter of both approach and implementation. It is difficult however, to minimise the problem of memory usage in

Read Length	Genome Length	Pixel Count	Library Size	Logic Utilisation	Memory Utilisation (Bits)	Frequency (MHz)
50	100	9	3	9,916	4,096	38.5
50	100	9	5	10,597	4,096	39.7
50	100	9	7	11,295	4,096	39.6
50	100	9	10	12,324	4,096	40.6
50	200	5	5	5,896	4,096	40.2
50	200	10	5	11,744	4,096	40.1
50	200	20	5	23,590	4,096	39.5
50	200	40	5	50,513	4,096	34.1
50	100	5	5	5,896	4,096	40.2
50	1,000	5	5	5,896	4,096	40.2
50	10,000	5	5	5,896	4,096	40.2
100	2,000	5	5	5,957	4,096	41.7
200	2,000	5	5	6,005	4,096	41.6
400	2,000	5	5	6,047	4,096	41.6
800	2,000	5	5	6,067	4,096	41.7
1,600	2,000	5	5	6,110	4,096	41.7

Table 2.1: Design Size of Y Hu's Comparison Engine

genome will cause more overlaps. Other variables have a much more limited change; the memory utilisation stays constant, as does the frequency. This is because the longest path in the design is held within a single pixel.

In terms of absolute resources the control logic has very little overhead and the memory utilisation is very low. For any given FPGA, memory generally outnumbers logic elements significantly. In this design, the memory utilisation is very low however, and a fixed level so in this design the logic utilisation will be the limiting factor when fitting the algorithm on to a device.

2.4 Hardware Acceleration

The scope of this project was to build an FPGA accelerator for the comparison process, the reason for the choice of FPGA as the hardware for this project is several fold. The first and most important motivating factor is the reconfigurability of FPGAs, these devices are designed to be reprogrammed many times, allowing the DNA detection hardware to drive the configuration of the comparison engine. This is important due to the many possible ways the DNA detection could be done and the paralellisation of the task. The hardware for this project must also be highly parallelisable in order to take advantage of the pixel based design, a traditional central processing unit (CPU) is by nature serial and therefore could not accelerate the process in the same way.

General purpose Graphics Processing Units (GPUs) were an alternative to FPGAs in terms of parallelisation, however these are generally not used as stand alone devices and communication between processing units in a GPU can be relatively expensive in terms of performance due the design of the GPU and the hierarchical nature of it's scheduling. Fitting the algorithm to different streaming multi-processors in a GPU would be a necessary complication of that approach. GPUs also have very limited scope for acceleration using additional hardware without complex cluster designs, communicating via protocols such as PCI-Express and this does not favour a ring network design that was discussed earlier. It is expected that FPGA development is a longer process, however the FPGA hardware is tailored perfectly to support the ring network design, allowing massive parallelisation and communication. The FPGA device is also normally a standalone device allowing it to communicate directly with the DNA detection hardware rather than incurring the cost and delay of a host computer.

The primary motivation for using an FPGA is therefore speed, it has been shown in several papers that FPGAs are capable of signficantly speeding up parallel processess [3]. For suitably parallel applications with large enough problem sizes FPGAs have been shown to be the best solution in terms of performance. This can be seen in figure 2.4 where for Sum of Absolute Differences (SAD), convolution and Correntropy the FPGA significantly outperforms CPUs and GPUs for most problems [3]. Shown below green represents FPGA, light blue represents GPU accelerated, red represents basic GPU, and dark blue represents CPU.

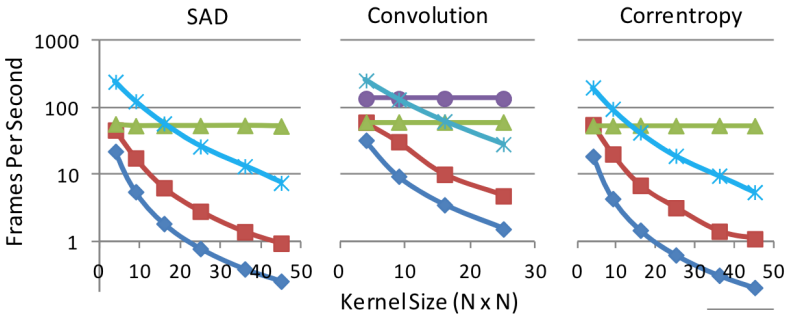

Figure 2.4: Performance of all implementations in frames per second for SAD, 2D Convolution, and correntropy at all kernel sizes tested on 720p images. [3]

The only alternative that shares all the benefits of an FPGA based implementation would be developing an application specific integrated circuit (ASIC). An ASIC would not be easily reconfigurable but would be truly parallel in the ring network's design and could be built on the same chip as the detection hardware. The draw back of this approach however, is the enormous cost and time associated with developing an ASIC). The design of an FPGA based system also leaves the possibility of an ASIC based implementation at a later point based on the FPGA design.

Chapter 4

Design

4.1 Overview

The structure of this project is split into three sections; the device hardware, the device software, and host software. The device hardware is the organisation of the pre-built FPGA devices into a cluster configuration using custom inter-connectors for the cluster interface. The device software is the program that will be written onto each device and dictates their functionality. The device software comprises the bulk of the project, partitioning the computation into separate devices and handling all communication protocols between the devices. The host software runs on a host machine connected to the cluster and provides all source data for the system as well as receiving all the output data from the cluster system.

In this project the available hardware has been the motivating design limitation, and so discussion of the hardware configuration is discussed first, the characteristics of which will motivate the design decisions made in the rest of the design.

4.2 Hardware

The motivation of this project was a scalable system and the design of the HDL is intended to be built such that it can be implemented on any FPGA hardware with the relevant ports. For the purposes of testing in this project three DE0 FPGAs will be used (shown in Figure 4.1).

The Terasic DE0 board contains an Altera CYCLONE III FPGA with 15,408 logic elements and 512kb of memory. This chip is capable of hosting around 9 pixels from the original algorithm within around 10 000 logic elements. This board has a relatively large number of general purpose input/output ports with 2 40-pin connectors on the side of the board. It also has a 5-pin RS232 specification UART which may be used for host computer to device simulation.

In order to ensure that the design of the system is scalable to other architectures and interfaces, the UART controller can be logically partitioned and replaced in other applications as will be discussed in the HDL design. First estimates of the number of interconnect ports needed are around 40 pins minimum with the scaling as the design parameters change. As these FPGA development boards are available freely within the

department they will be used for the prototype system design. It is important to note however, that this should not impact the design in such a way that stops the design from being easily moved to other boards when necessary. A secondary benefit of this board is the large number of user input/output ports such as 4 7-segment displays which can be used for debugging and progress checking. This board was freely available within the department and therefore made a good low cost prototyping unit, however these boards can be bought at a retail price of $120.

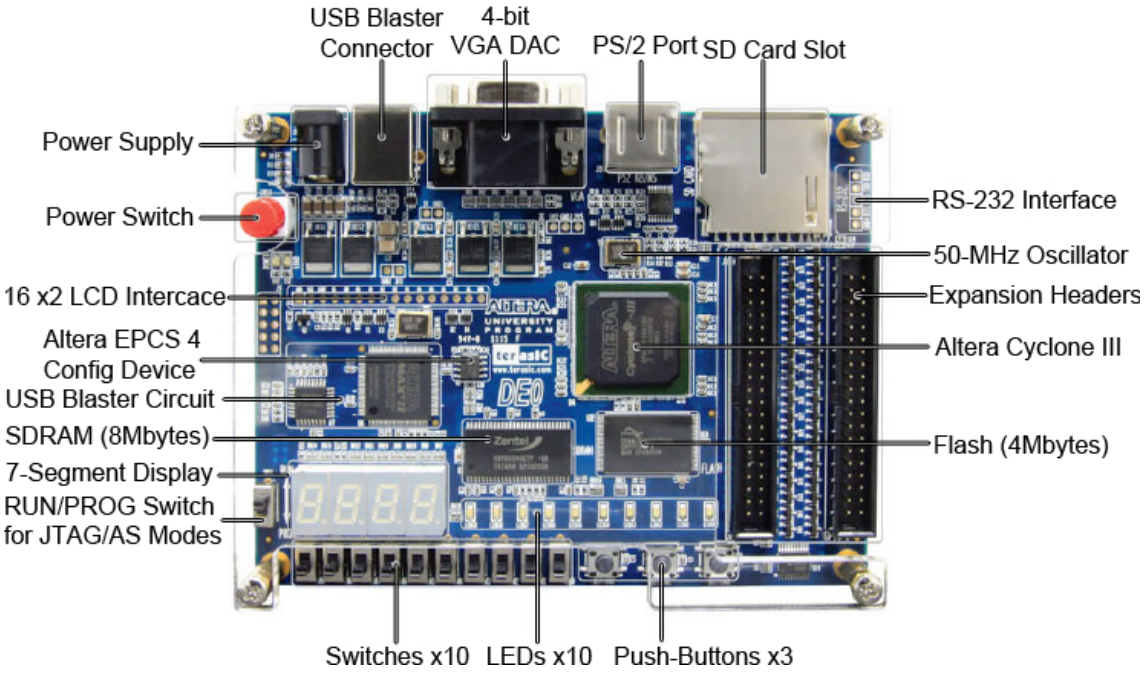

Figure 4.1: Terasic DE0 FPGA Board

An important decision to be made in the design of the hardware was whether to use a common clock across all the devices, or to use separate clocking. The advantage of the single clock is that the processors can work in lock-step reducing communication delays and simplifying the interconnect protocol. However, a common clock places other limitations on the design. The common clock design limits the clock frequency of all the devices to the speed at which the inter-FPGA communication can happen, this may lower the theoretical maximum clock speed, however the design already runs at a relatively low clock speed and the intentional is to overcome this limitation with parallelisation. The common clock will need to be considered in the testing stage, as identifying the clock speed where correct operation is primarily done through trial and error with a trade-off between bit-error rates and speed. The ring network design of this system favours the common clock approach as all communication is exclusively done with nearest neighbours, limiting the length of the inter-FPGA communications.

Chapter 5

Implementation

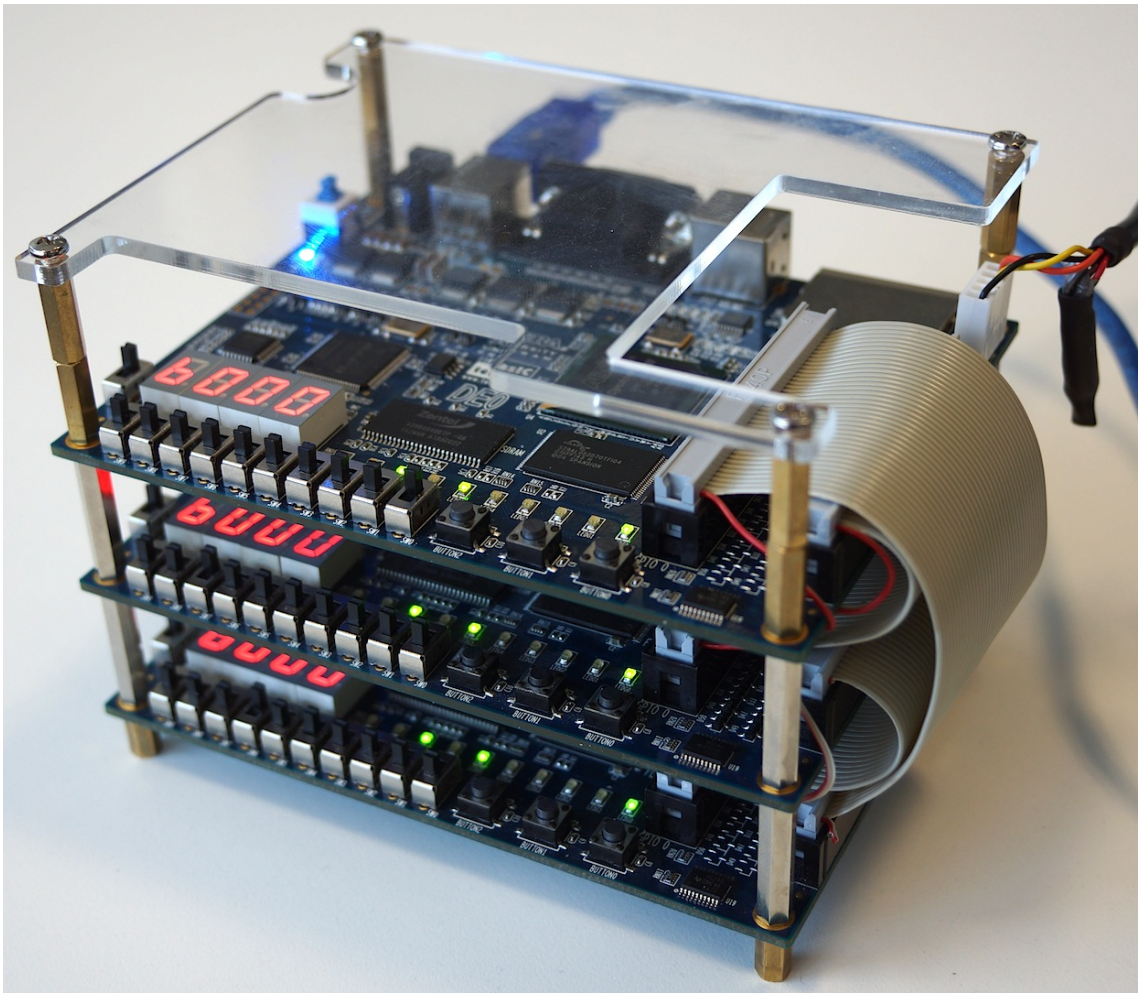

Figure 5.1: Hardware Implementation

As discussed in the design section, one of the main features of this design was organising the processors to run in lockstep. This meant there were no internal data transfer protocols other than simple ready/available data signals. As a result the vast majority of the design translated from simulation to device implementation in a straight forward way. The only non-deterministic element of the design was the shared clock across multiple devices, this element required some simple analysis to identify the limits. The clock frequency needs fine control in order to limit the error rates in the data transfer. This was done by using a phase locked loop (PLL) on the master device, this takes the

local 50 MHz clock as an input and outputs a compile time set frequency. This clock is then put out across 1 GPIO pin. As described in the design section this is then used as the clock input of all the devices.

The clock frequency maximum set by the on board logic was 31MHz, ideally the inter-FPGA communication also fits this limit, allowing maximum speed of operation. The velocity of propagation for the ribbon cable is 4.53 nsM^{-1} [39]. Assuming a total cable length of 15cm, including the propagation from chip to connector (it is safe to assume the PCB is a better propagator than the cable), the total latency of a signal is

$$\begin{aligned}
\text{Propagation Delay} &= \text{Propagation Velocity} \times \text{distance} \\
&= 4.53 \times 0.15 \\
&= 0.6795 \text{ ns} \\
\text{Fmax} &= \frac{1}{\text{Propagation Delay} + \text{Input Delay} + \text{Output Delay}} \\
\text{Fmax} &\approx \frac{1}{0.680 \times 10^{-9} + 2.19 \times 10^{-9} + 0.915 \times 10^{-9}} \\
&= 260 \text{ MHz}
\end{aligned}$$

These calculations use IO delay characteristics as detailed in the Altera CYCLONE III datasheet [40].

As can be seen from these estimations, as long as the inputs and outputs are registered, there should be enough slack to run the processor at the limit the pixels will allow. It is important having made these estimations to test these in a practical setting.

Another implementation problem to consider is the clock skew. The clock can take a variable amount of time to travel both across chip and between chips. This is easy to model within a single device, using the toolchains provided by FPGA manufacturers. However, across chips there are no modelling tools. In a similar way to the Fmax calculation, the clock skew will be effected by the number of devices and their distance apart. However, it is possible to try and control the clock skew by ensuring that the clock source is connected to all the devices in a similar fashion. On the DE0 FPGA this has been achieved by sending the clock out of the master device and then splitting the signal from a shared point to go to each device with an identical connection. This still requires testing in a practical environment however. For larger clusters of FPGAs on more expensive technologies there are different systems that can be used. The ribbon cable that is used in this implementation isn't a particularly high quality transmitter for this purpose, other devices have serial digital interfaces which may be used for clock sharing, this provide much higher quality channels across which the clock can be shared.

There are also steps that can be taken within the compilation of the HDL for each

Appendix A

User Guide

The comparison engine project consists of 3 parts; hardware, device logic, and host software. Using this system involves a number of steps initially setting up each part, and then a simple command line program to run the comparison.

A.1 Getting the Code

All code and resources for this project are available on request from jcr09@ic.ac.uk.

A.2 Setting up the Hardware

The first step required in using the comparison engine is to set up the hardware. This involves connecting the FGPAs together, this is done as detailed in the hardware section of the report (Section 1, Chatpter 4). This requires connecting each FPGA to the next with ribbon cable as shown in the figures on the next page. Shown on the next page is the setup for the DE0 cluster. A UART must be connected to 1 FPGA, the master device. This is most commmonly done with a UART-USB connector such as the FTDI USB-RS232 cable listed in the bibliography here as reference [41]. Only the TX and RX connectors are needed, the flow control signals can be left unconnected.

A.3 Programming the FPGA Devices

The FPGA device images is the setup information that dictates how the FPGA functions. In order to do this, 2 images must be compiled. One image will go on the slave, one on the master. The master is the device that produces the clock and is connected to the UART as detailed earlier. This is done in the Altera Quartus II software with the two projects made available on request.

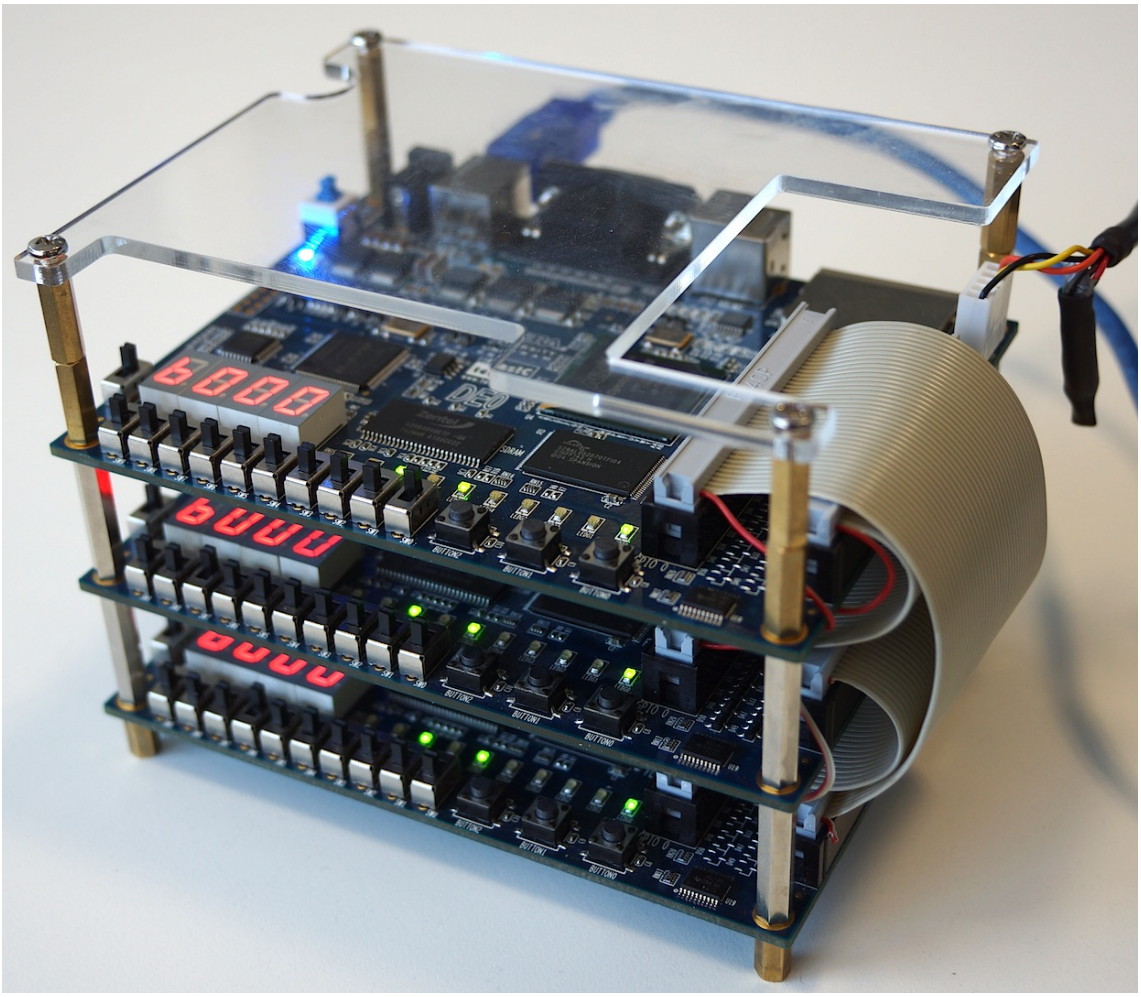

Figure A.1: Hardware Implementation

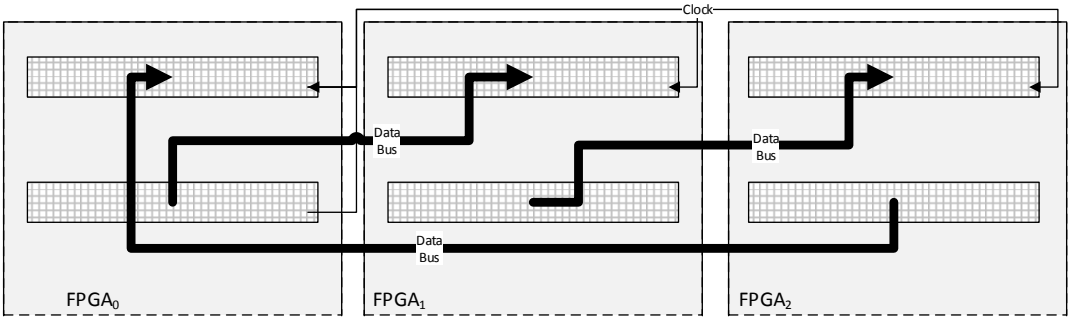

Figure A.2: Inter-FPGA Communication Hardware Setup