A High-radix Online Arithmetic Verification System

Final Year Project 1800478: Interim Report

Zifan Wang, 01077639
Imperial College London

January 28, 2019

Contents

1 Introduction

With the right number representation system, it is possible to perform arithmetic operations MSD first. Consequently, these online arithmetic operators are attractive for hardware implementation in both serial and parallel forms. When computing digits serially, they can be chained such that subsequent operations begin before the preceding ones complete. Parallel implementations tend to be most sensitive to failure in their LSDs, making them more friendly to overclocking than their LSD first counterparts, for which the opposite is true.

In the past, online operators have typically been implemented in binary. Although Radix-2 modules are the simplest to design and has the shortest cycle time per digit, it has the highest online delay and requires the largest number of cycles to complete calculations [18]. As such, the choice of binary is not absolute. In this project, I will explore high-radix online operators, investigating their suitability for FPGA implementation and examining the resultant tradeoffs between performance, area and power.

2 Background Research

2.1 Online Arithmetic

Traditional arithmetic operators have two common characteristics. Firstly, their order of operation may be different depending on the operation itself. A traditional adder, parallel or serial, generates its answers from the LSD to the MSD. A traditional divider design, on the other hand, generates its answer from the MSD to the LSD [3] [14].

Due to this inconsistency, arithmetic operators may be forced to compute word-by-word, waiting for all digits to finish in the previous operator before the next can start [21]. Therefore, if a divider follows an adder, the divider has to wait until the adder has completed its computation before it can begin its own.

The other commonality of traditional designs is that their precisions are specified at design-time. Once built, a 32-bit adder always adds 32 bits together, adding 16-bit numbers usually involves masking the unused bits. A possible way of making it less inefficient would be using SIMD instructions [5], combining smaller operations into a larger one that fits the hardware. This, however, has the tradeoff of being harder to program, and the applications must have sufficient parallelism to exploit.

Online arithmetic does not suffer from the first issue as it performs all arithmetic operations from MSD first [7] [8]. Furthermore, pipelining can be used with online serial arithmetic operators. Thus the output digit of an earlier operation can be fed into the next operator before the earlier one completes its computation.

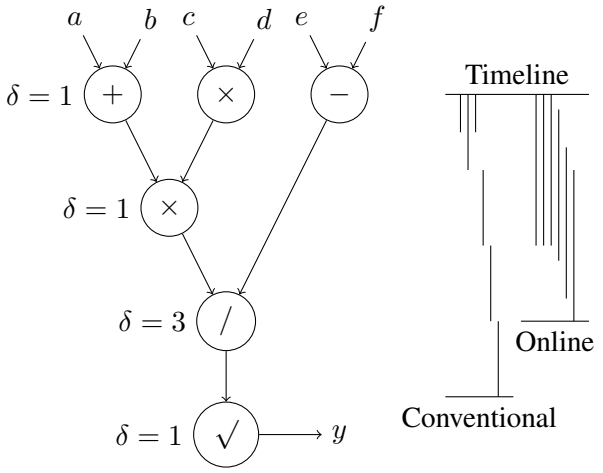

Figure 1: Computing $y = \sqrt{(a+b)cd/(e-f)}$ with serial online operators [7]

As illustrated in figure 1, while each individual operation may take longer than its conventional counterpart,

online arithmetic can provide a speedup if the operators are chained in serial. In addition to the tradeoff in time, individual online arithmetic operators also uses more memory. To perform all computation from the MSD to the LSD, the use of a redundant number system is compulsory. However, this redundancy also has its advantage in making the operators scalable. The time required per digit can be made independent of the length of the operands [19].

A recently proposed architecture allows the precision of online arithmetic to be controlled at runtime [21]. Traditionally, this runtime control was restricted due to the parallel adders present in the multipliers and dividers. This architecture reuses a fixed-precision adder and stores residues in on-chip RAM. As such, a single piece of hardware can be used to calculate to any precision, limited only by the size of the on-chip RAM.

The way online arithmetic alleviates the second problem of fixed precision falls out directly from its MSD-first nature. Suppose the output of a conventional ripple adder is sampled before it has completed its operation. In this case, the lower digits would have been completed, but the carry would not have reached the higher ones. This means the error on the result would be significant, as the top bits were still undetermined [15]. However, if the output of a parallel online adder is sampled before its completion, the lower bits would be the undetermined ones. This means the error of the operation would be small. With overclocking, online arithmetic operators fail gracefully, losing their precision gradually from the lowest bits first. Thus, it allows for a runtime tradeoff between precision and frequency [16].

2.2 High-radix Arithmetic

Conventional designs of arithmetic operators use binary representations. This was chosen as the additional complexity of a high-radix operator does not always provide justifiable speedups except in specific cases [10] [2]. However, using a high-radix representation system could still yield better numerical accuracy while reducing area cost for FPGAs. For example, a hexadecimal floating-point adder was shown to have a 30% smaller area-time product than its binary counterpart, while still delivering equal worst-case and better average-case numerical accuracy [4].

However, the savings are not without trade-offs. This trade-off can become unfavourable if the specification requires much I/O and little computation [20]. This is because the overhead of radix conversion would be significant. It is also unwise to use high-radix representations when the numbers are unusually small, thus making the savings offered by the high-radices negligible [4].

2.3 High-radix Online Arithmetic

Using high-radix number representations for online arithmetic is a relatively novel concept. While there has been some research with similar premises [12] [13], We take a more direct approach with this project by implementing custom operators made for high-radix online arithmetic on an FPGA. This will provide empirical results on the method, and will hopefully reveal practical insights along the way.

Furthermore, benchmarking this exotic arithmetic system with popular FPGA applications such as neural networks would be interesting, as there is not much precedence for it.

3 Project Specification

3.1 Project Organisation

This project is a part of a larger project investigating the effect of using high-radix number representation with online arithmetic operators. The overarching aim involves implementing such a system on an FPGA and quantifying its performance improvements. This is achieved through two individual projects, vertically split from the enveloping project. One shall design the arithmetic operator modules, while the other shall design a system from the top-level to test and evaluate these operators. This project deals with the system-level issues.

As this project progresses in parallel with the designing of the operator modules, it is necessary to decouple the two projects so that, being individual projects, they can be evaluated individually. The success of one project should not be restricted by the status of the other. To this end, the goal of the system-level design is more focussed on its functionalities and robustness. This relationship and its effect on the evaluation will be examined further in the evaluation chapter of this report.

To ensure the two products will work together once they are both complete, a common interface is agreed upon. The interface will be done using Qsys. The unit-level project will build different operators, which can have varying arithmetic functions and designs. These will be packaged into Qsys modules, which will become the DUTs of the testbench.

3.2 Deliverables

At the end of the project, the system should be able to perform the following:

1. Connect to the arithmetic modules as its input;
2. Generate and run tests on these modules;
3. Vary the frequency of the FPGA;
4. Evaluate its performance.

3.3 Hardware Choice

The system itself will be built on a Cyclone V SX SoC Development Board from Intel [30].

The 5CSXFC6D6F31C6N SoC has an Arm Cortex-A9 MPCore accompanied by Intel's 28nm FPGA fabric [23]. The FPGA is necessary for implementing the hardware design and obtaining empirical results for the project.

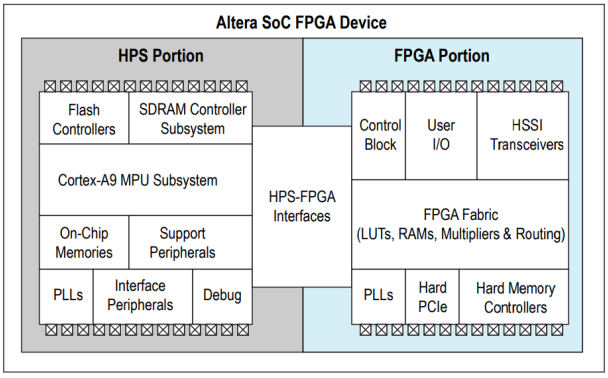

Figure 2: Structure of the System-on-Chip

While an FPGA without an embedded CPU will be enough for this project to work, having an Hard Processor System (HPS) on the same chip is useful as the test software can run on it. The HPS is a separate piece of hardware

that distinguishes itself from a soft processor, such as the Nios II, a processor programmed onto the FPGA itself. With this additional capacity, a better user interface can thus be constructed with more detailed, on-the-fly control of the FPGA. This means setting up the testbench will only require programming the design into the FPGA, followed by running the test script on the HPS. The product will thus be self-contained. It will be more accessible as no additional setup is required for the user.

It should be noted that Xilinx offers similar boards as well. Its Zynq SoC family has a very comparable structure as they too integrate the software programmability of an Arm processor with the hardware possibility of an FPGA. For example, similar to the Cyclone V SX, Zynq-7000S features an Arm Cortex-A9 coupled with a Xilinx 28nm FPGA [34]. As such, a board like the ZedBoard [35] could be just as viable for this project.

As there are very few significant functional differences between the two brands, I shall initially explore with the Intel board, simply for its availability and my familiarity with their development tools. Due to the architectural differences between the logic elements between Xilinx and Altera FPGAs [17], the performances on the two boards are not necessarily identical. Once the project has progressed to a point where the system design is mature and tested, the Xilinx alternative can be explored as an extension.

3.4 Software Choice

The software choice follows closely with the hardware choice in this project. To develop for Intel FPGAs, Quartus has to be used. The version picked is arbitrary as there are not many functional differences between the versions that will be critical to the project. As Quartus Prime 16.0 is the version installed in the computers in the department, I will use the same version simply for convenience. This naturally means the hardware system will be built with the system integration tool that comes with Quartus – Qsys.

The Qsys software is designed to be used for integrating different hardware modules into a system. As such, it will be used as the interface for the two parallel projects.

While an HLS language could be used, in this design it suffers from a few problems and does not offer enough benefits to justify its use. Usually HLS is preferred for developing complex algorithms, because compilers can optimise them into RTL much better than humans. However, the resulting RTL would be unreadable, making directly controlling or debugging at the hardware level nearly impossible. The interfaces require detailed control of the actual hardware and the rest of the testbench has a lot of control path work and direct manipulation on the data bits. It is therefore not worth it to use HLS and as such, this design will be written in Verilog.

Other than the hardware design tools, there is some freedom of choice on the HPS side of the project. The test will be built with Python, which will be running on an Ubuntu system that is installed on the HPS. This choice is made as there are previous unrelated projects on the same development board, which means a lot of time can be saved on tedious setup works such as getting an operating system booting.

Git is used as the version control system for this project. A list of repositories on GitHub holds all files related to this project. Readme files on the repositories and the commit histories will serve as digital logbooks to this project.

4 Engineering Background

4.1 Testbench Architecture

The design of the verification system is the major engineering challenge of this project. While there have been many similar performance analyses done on hybrid SoCs before, each of them used their own, usually ad hoc, testbench design [15] [11]. As such, most testbench are not designed to be scalable or portable, serving only what they are built for. In this project, I shall use a generic structure inspired by that of an agent in Universal Verification Methodology (UVM).

Before UVM, integrated circuit designs were verified with methodologies developed independently by stimulator vendors such as Cadence, Mentor Graphics, and Synopsys. In an effort to unify for greater efficiency, the standards organisation of the Electronic Design Automation (EDA) industry, Accellera, established UVM with support from multiple vendors. It provided a common structure for verification, with class libraries that made building and running a testbench a significantly smoother experience. The agent is a container in UVM that emulates and verifies DUTs [22]. While this project is in no position to achieve what UVM has done, I do hope that this testbench would have an easily modifiable structure that will make the process of testing similar future designs slightly simpler.

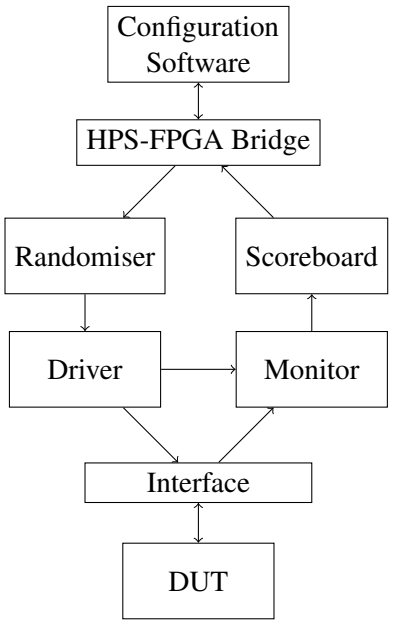

Figure 3: Block diagram of the proposed testbench

Configuration is first done from software running on the HPS, which sends the test specifications to the randomiser. The randomiser will provide a stream of random data that will be converted to meaningful test inputs by the driver. The test output will be watched by the monitor, reporting any interesting event to the scoreboard, which keeps track of them. The scoreboard feeds the information back to the software on demand. The interface is used to decouple the control logic from the DUT, allowing the frequency of the DUT to be finely controlled.

4.2 Data Transfer Rate

In order to stress the DUT, the verification system must perform at a much higher frequency than the expected frequency of the DUT. Assuming the DUT is to run at 300MHz, to fully explore the effect of overclocking, the testbench must be able to run at double the frequency or higher. This gives a target frequency of 800MHz. Assuming a data width of 32-bit, the target data transfer rate is then estimated to be 25.6Gbps.

4.2.1 HPS-FPGA Bridge

As the testing is to be controlled by the HPS, the HPS-FPGA bridge will be the immediate bottleneck if the test data is to flow from HPS to FPGA. While the HPS can easily generate test data with a piece of software, there is a large amount of overhead as data crosses from one architecture to another. This overhead exists in the form of both decreased bandwidth and increased delay. Thus, it is not be sensible for the HPS to send out data during runtime.

4.2.2 Off-chip DDR SDRAM

Another thought may be to first populate the off-chip DDR SDRAM on the FPGA side, then feed that data to the DUT during test. This is already much faster than passing the data directly from HPS. The 1GB, 32-bit wide DDR3 on the FPGA side is rated at 400MHz. With double rate transfer, this gives a maximum transfer rate of 25.6Gbps.

Although using the off-chip RAM may theoretically achieve the targets, it still has its disadvantages. Firstly, the process of filling up the memory takes time. Thus, the testing would be broken up into bursts, with time in between for checking results and filling in new data. The complexity of the SDRAM interface also requires an SDRAM controller to be used to manage SDRAM refresh cycles, address multiplexing and interface timing. These all add up to significant access latency. While it could be overcome with burst and piplined accesses, it would further complicate the SDRAM controller. A controller is provided by Intel [25], but it would consume a non-negligible amount of the limited FPGA resources while adding unnecessary complexities to the design. Customising or building a new SDRAM controller to fit this project is possible, but needlessly time-consuming.

4.2.3 On-chip Memory

The on-chip memory is much faster and simpler to use. In comparison, this memory is implemented on the FPGA itself, and thus needs no external connections for accesses. It has higher throughput and lower latency than the SDRAM. The memory transactions can also be piplined, giving one transaction per clock cycle. With an on-chip FIFO accessed in dual-port mode, the write operations at one end and the read operations at the other end can happen simultaneously. This feature is useful as tests are prepared and fed into the DUT, or when test results are collected and fed to the monitor.

On-chip memory is not without its drawbacks. It is volatile like SDRAM and very limited in capacity. SDRAMs can have store about 1GB, while on-chip memory could only hold a few MB [24]. Volatility is not exactly of concern in this project, but its small capacity means not much test data can be held before it needs more fed in.

4.2.4 Distributed RAM / Registers

On-chip memory has a minimum latency of 1 clock cycle as the R/W access gets processed. If a even faster memory is desired, we can use LUTs or registers to store them. This option would eliminate the latency but takes up much more FPGA resources. The capacity is even more limited as LUTs are usually used for logic. There will be a significant amount of data generated during testing, and the testbench should be as lightweight as possible to allow flexibility in the DUTs. As such, distributed RAM will not be used in this project for data transfer. Registers will still be used as they are essential for many other purposes.

4.3 Runtime Test Data Generation

To exploit the benefits of on-chip memory, and circumvent the drawback of buffering testing data generated from the HPS, a method for generating testing data at runtime, on the FPGA needs to be designed. As arithmetic operators have a vast set of valid inputs, it is necessary to have cost-effective test generation.

A good choice here is to use random testing. With relatively low effort, random testing can provide significant coverage and discover relatively subtle errors [6]. The main drawback of random testing is the possible lack of coverage for corner cases, for which the usual solution is to provide handwritten tests to complement it. However,

as the main goal of this testbench is gauging the performance of the module, and not necessarily verifying the correctness of the module, having uncovered testing holes is acceptable during stress testing. As the project progress, special tests could be written and run separately with a relaxed timing restriction to cover the holes. It should be noted that certain corner cases may represent critical paths in the design. To combat this, the testbench provides the option to run handwritten inputs alongside random tests.

LFSRs are a reliable way of generating pseudorandom numbers quickly with low cost [9]. They will thus form the starting point of data generation. While it is possible for data generated to be invalid as inputs to the DUT, this should not be the case for most benchmarks in this project. Even if this is the case, they can be dealt later in the process by the monitor.

By following this approach, the software would only need to configure the randomiser, and test data no longer needs to pass through the HPS-FPGA bridge. Thus, the testbench can provide fast and constant data to stress the DUT.

4.4 Clock Domains

Another concern in the system design is of the different clock domains that must exist on the FPGA. At a minimum, there need to be two clock domains: one surrounds the DUT and another supports the rest of the control logic around the DUT. These clock frequencies can be generated with PLLs, which are provided as IP Cores in the Quartus software [26]. A clock tree will distribute them to the individual modules. Data crossing clock domains will be fed through FIFOs to prevent loss.

The proposed structure will have the bulk of the control logic running in a separate clock domain to the DUT. Only an interface with FIFOs will be running in synchronicity with the DUT. Therefore, the test controls can run at a slower frequency without bottlenecking the system, allowing the DUT to be stressed further. The problem now is to ensure the monitor can handle the stream of DUT output coming in at a higher frequency that it is running at. As the monitor needs to calculate the correct data before it can check if the DUT output is correct, it cannot keep up with the speed of the DUT. This report consider three alternatives.

4.4.1 Partial Monitors

A lightweight idea is to implement a parity checker instead of a full model inside the monitors. For example, to check an adder, the monitor can just check if the final bit with a LUT acting as a XOR gate.

Although this is reasonably fast, it cannot be extended once the DUT is faster than a parity calculation followed by a comparison. More critically, it provides no additional information once the DUT fails, and it has a 50% rate of ignoring an error. If this is to be solved by increasing the number of bits checked, the problem returns back to its initial state. Thus this method will not be experimented.

4.4.2 Lazy Monitors

An more scalable alternative is to have the monitor only check a selection of data sets. For example, if the monitor is programmed to check every third test point, statistically it will make little difference to the final result. In case the DUT is aware of this and only produce correct outputs on every third operation, this process can be randomised too.

This method can be extended if the DUT get fast simply by skipping more checks, and it has the full information when it detects an error. However, this method needs the extra logic in the random controller, making the monitor slightly more complex than it probably should be.

4.4.3 Parallel Monitors

As the test data is uniform, the monitor can be parallelised in to a number of sub-monitors. The sub-monitors is connected to a arbiter that is connected to three FIFOs. The FIFOs are the inputs and the output of the DUT. A round robin arbiter distributes the data to the sub-monitors equally. The results from each sub-monitor are then

sent to a single scoreboard. To avoid potential hazards, the output from the sub-monitors will be buffered before processed by the scoreboard.

This does not have data dependency on a random controller, and it can fully guarantee the correctness of the DUT. It is also scalable as more sub-monitors can be added it the DUT fills up its output buffer. As a downside, this method takes up the most FPGA resources to implement as it scales.

Comparing across the three methods, the parallel monitors will be built first for this project, as it offers the best functionalities. Although unlikely, if a fatal issue arises in this design, or if the testbench needs to be more lightweight, then the lazy monitor will be used as the alternative.

4.5 Result Analysis

If the monitor detects an interesting event such as an error, it will send a message to the scoreboard. The scoreboard has counters tracking these events, which are exposed and can be read by the HPS.

The software can run statistics to provide further insights to the user.

5 Implementation Plan

5.1 Timeline

The project is divided into three main phases. The initial one is of research and learning. This is to lay the foundation to the project. The second phase is building the core product of the project. The final phase is enhancing the product.

In order to track the progress and success of the project, the difficulties of the deliverables need to be analysed first. Planning slacks are appended after all future tasks to reduce uncertainties. Figure 4 provides a visualisation of the project timeline. The slacks are indicated by the shaded areas in the chart.

5.1.1 Term Time and Examinations

Time available for the project varies greatly throughout the year. The greatest factors will be term time and examinations. The time needed for revision has been marked in the chart. These times will allow minimal progress of the project.

During term time, there will be coursework deadlines, which will also negatively affect the time that can be allocated to this project. One coursework module was selected for Autumn and three were picked for Summer. As such, it is expected that the progress will be somewhat slower during Autumn but significantly slower during Spring. To make up for the time lost, a part of Christmas was used and a few weeks from Easter will also be committed towards the project.

The list of tasks were then laid out onto the timeline. This was done according to their expected difficulty and the expected availability to work on those tasks. Deducting 5 weeks for revision, 15 weeks remain before the

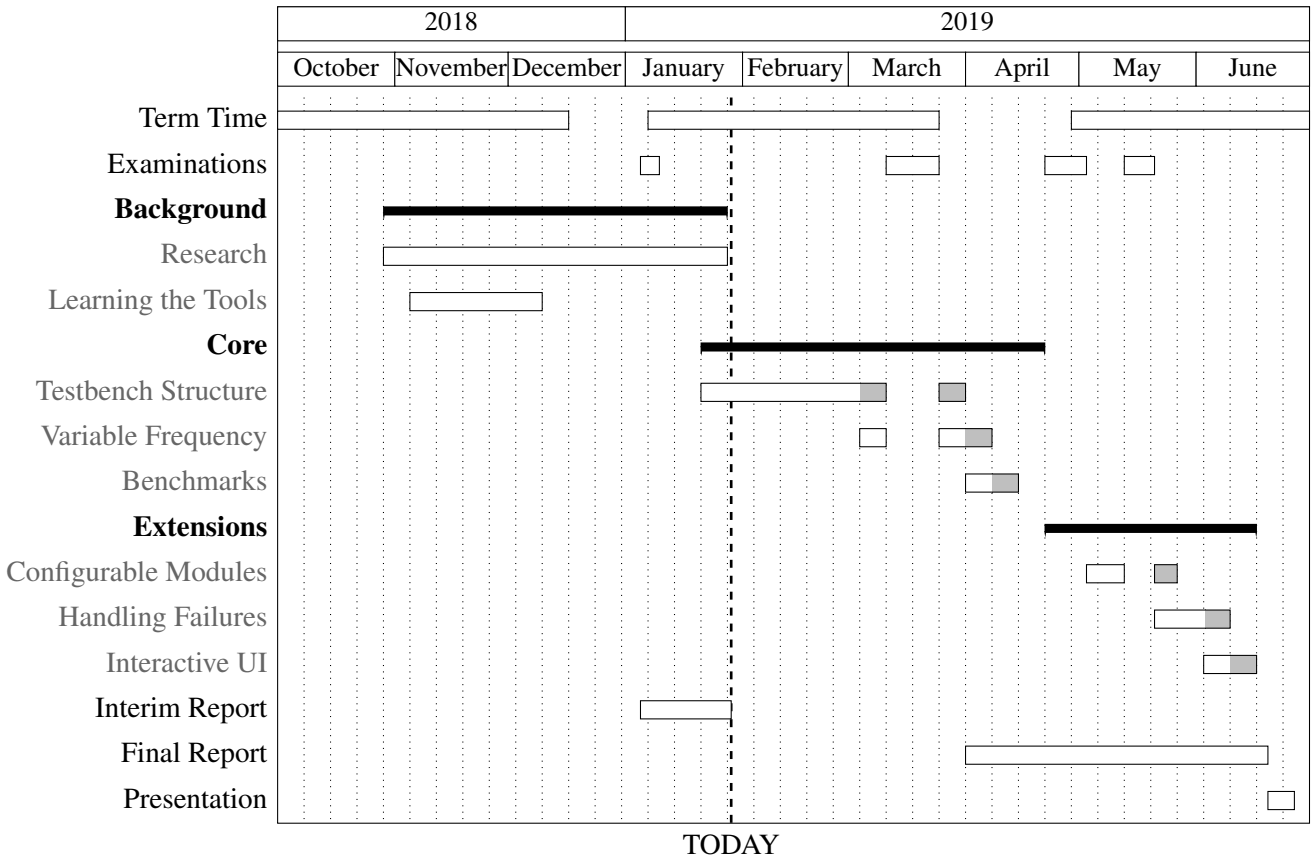

Figure 4: Project Timeline

submission of the final report. To still have enough flexibility, the tasks were planned to be possible to complete in 12 weeks. However, 12 weeks is the best case scenario. With reasonable slack for all tasks taken into account, the worst case will take 19 weeks. As such, The tasks near the end are planned so that they can be easily dropped if there time becomes an issue. This is reflected in the plan, as even in the worst case scenario, completing all core tasks takes 12 weeks. On the other hand, if all tasks were complete in 12 weeks, the remaining time can always be used to improve the product further. This can either mean going back to optimise earlier tasks, or come up with new ones.

5.1.2 Background

Research – To fill in the background knowledge required to work on this project, the first months were spent on reading textbooks and papers. The research was to provide context and motivation for the project. It also provided an overview of the field of research and some understanding towards the current state-of-the-art. Most importantly, it offered the necessary knowledge needed for this project. The result of this research was summarised in chapter 3.

Learning the Tools – Due to the lack of experience in programming hybrid SoCs and the lack of knowledge in the current state-of-the-art digital arithmetic designs, significant effort was spent on researching and learning the skills necessary to carry out the project. This involved building a small testing system on the board. The process taught me not only how to handle a hybrid SoC, but also how to handle the Quartus file system with git. Details regarding this testing system can be found in section 5.2.

5.1.3 Core Tasks

While most of the later sections of the project can be selectively added or removed from the scope relatively easily, this initial setup of the testbench structure will always remain critical to any further improvements. It is thus vital that the base system is completed early. To ensure that this happens, this task and the testbench structure will be placed in the highest priority before its completion, and any blocking issue will be discussed with the supervisor if it cannot be resolved after reasonable effort.

Testbench Structure – Once comfortable with the tools, the main design of the testbench can start. This task will be the foundation of this project, as it will provide a basis for all of the following features. A skeletal testbench should be complete and functional at the end of this task. This means an operator module matching the correct specifications can be loaded into the testbench to run basic tests. To reduce risk and uncertainty at this stage in the project, this basic testbench is not required to have the full flexibility of the final product in terms of compatibility. While it is still undesirable to hardcode parameters and features, the support for DUTs different from the one built by the other project will only be addressed later as task 7.

Variable Frequency – As the project seeks to quantify performance across a range of frequencies, the most important feature of the system will be the ability to vary this parameter. This will be done by controlling the PLLs with the HPS [26]. TimeQuest Timing Analyser in Quartus can be useful as an estimation of what is achievable. Once implemented, empirical tests will be run to explore and confirm the maximum frequency for which the testbench can remain reliable. If it does not meet the planned target, the testbench may need to be redesigned, and the project would be under some risk.

Benchmarks – Once the testbench can reach a satisfying performance, more intricate tests and benchmarks can be designed. These aims to reflect the system's performance running meaningful compute tasks. The systems enveloping the arithmetic modules could be stressed with popular algorithms to evaluate real-life obtainable speedups.

In addition to running better tests, we also aim to obtain better data from the tests. The results required here would be numerical information on power consumption, FPGA resources used, and the data throughput. FPGA resources used can be obtained from compilation report in Quartus. Data throughput can be calculated from clock frequency and data width. Power consumption can be estimated from PowerPlay Power Analyser in Quartus.

5.1.4 Extensions

The previous three tasks form the core of this project. In other words, all core tasks must be completed for a functional product. The following tasks however, will be mostly considered as useful extensions. While not as vital as the core tasks, completion of the following tasks would greatly improve usability, and thus are still relevant to the success of the final product.

Configurable Modules – So far, only modules of a specific I/O width and numerical representation can be tested. It might be interesting to explore arithmetic modules with other configurations. To allow the testbench to be used for further experiments or future projects, it will be helpful to have a configurable testbench. Qsys components can be configured with a Hardware Component Definition File [27]. The plan is to build the testbench as a Qsys component, then use Qsys Component Editor as an interface for configuration. This is because the Component Editor provides a friendly interface for configuration directly from analysing the testbench design file.

Handling Failures – Another improvement to the testbench is related to how it handles failures in the module. It would be much more insightful for the user if a more insightful failure message were provided in addition to just a simple failure rate. This could include examples of failed output against expected output, or statistical data describing the pattern of failures. This additional logic required may degrade performance of the testbench, so it would be useful for the verbosity of this information to be configurable by the user.

Interactive UI – If time allows for even greater user experience enhancements, real-time interactive graphical user interface could be constructed for the final demonstration. This would visualise the reduction of the module's accuracy as the user increases the clock rate. However, this would take significant time and effort, and this task will be re-evaluated when the project progresses to the stage. A time-saving yet functional alternative would be a command-line interface with a well-documented user guide.

5.1.5 Reports and Presentation

The reports and the presentation are the most visible in all deliverables of this entire process. As such, while not directly contributing to the progress of the project, they are still vital to its success. The reports will be written alongside the engineering process. At the end, around a week of time will be spent solely on completing and polishing up the final report. This should allow ample time for a well-organised submission.

The week after the final report will be used for the presentation. This involves preparing a slide deck, a demonstration, and a script. During the demonstration, the product will be exhibited in front of the audience. The details of the exhibition largely depends on the status of task 9.

5.2 Work to Date

Before any engineering work was done towards the final product, a small module was built to learn the environment. This module needed to be simple yet covered enough grounds to provide as much learning during the process as possible without taking up too much actual development time towards the product. As the greatest unfamiliarity was with the interaction across the HPS-FPGA bridge, a simple hardware accelerated adder was made for this training.

5.2.1 FPGA Side

Programming the FPGA to communicate with the HPS is no trivial task. Luckily, there exists a golden system reference design [33] for the board in use for this project. Unfortunately, support for certain versions of Quartus are missing from the GSRD download database, including the one used for this project, 16.0. While the design can be opened with a different version of the software, it causes a series of conflicts usually related to using IP cores that have changed over the iterations. To circumvent this issue cleanly, GSRD version 14.1 was downloaded and compiled on a separate install of Quartus II 14.1. This allowed the reference design to be studied in detail, and the sections required for this project to be rebuilt with Quartus Prime 16.0.

From the perspective of the FPGA, The HPS exposes three bridges for connections [28]. As this is a relatively simple task, the lightweight bridge was used. Module `altera_hps` exposes the master of this bridge as `h2f_lw_axi_master`. Next, the actual hardware adder was built and integrated as a hardware module in Qsys with a matching interface. A simple adder can produce its result after one clock cycle. This greatly simplifies the logic required for the Avalon slave interface. The logic for the control and data signals were thus written according to the interface specifications [32]. Following the naming conventions for the signals allows Qsys Component Editor to automatically detect the Avalon slave from this module at analysis. This saves the troubles of editing the `_hw.tcl` file. To experiment with module configuration, the adder was designed with variable width.

The adder was then instantiated and connected to the rest of the system with two clicks in Qsys. From there, Qsys could generate the HDL for the entire system, which was then compiled to a bitstream file. With the bitstream ready, the work now shifted to the HPS.

5.2.2 HPS Side

The HPS runs Ubuntu and a Bash script has been written to load the bitstream onto the FPGA. Next, a program was written in Python to test the hardware design from the HPS. The interfaces are mapped onto the physical memory, thus they can be accessed by opening `/dev/mem`. Checking against the specifications [28], the lightweight master is at `0xFF20_0000`. Qsys allocates the memory spaces of modules relatively, so when it reports that the adder has been placed at `0x0010_0000`, it is physically at `0xFF30_0000`. The adder was designed to have its two inputs at `0x00` and `0x10` and its output at `0x20`, which were assigned by Qsys relatively to `0xFF30_0000`.

With the memory mapping understood, the program was designed to closely mirror this relative relationship between the modules using classes. For example, the adder defines its output at `0x20`, but its read and write methods are inherited from an AXI class that brings it to the correct physical address by adding the address of the bridge defined in it. This parallel between software and hardware should be helpful as the product gets more complex.

To verify what I have built and learnt was correct, 1000 add operations were executed separately with and without the hardware acceleration of the FPGA. The results were compared and confirmed that this training module worked. While called hardware acceleration, the FPGA actually had worse performance than the HPS in this testing case. The CPU is reasonably efficient in calculating additions, while calculating on the FPGA requires the adder I/O data going through the HPS-FPGA bridge. This incurs a significant overhead, thus slowing it down.

6 Evaluation Plan

6.1 Product Metrics

6.1.1 Robustness

The next few measures looks at the performance of the final product. First, the maximum stress of which the testbench can provide without failing is a good metric. This can be quantitatively measured by the maximum data throughput across the DUT, and the maximum frequency that the DUT can be running where the testbench remains reliable. A robust testbench with a higher maximum frequency can reveal a wider picture in the performance of the DUT. This would hopefully allow more insights to be gained regarding the DUT, or it could mean that the testbench can be used for future designs that may be faster than the current one. As the main quantitative metric, this will be a vital indicator of the project's success.

6.1.2 Flexibility

The flexibility of the testbench is also vital to the product's performance. The testbench should suitable for a range of DUTs. This will allow the testbench to be used for future experiments. The flexibility can be measured by the number of configurable parameters that it has, and the range of which these parameters can be adjusted to.

6.1.3 User-friendliness

The ease of use of the testbench can be another evaluation point. On the hardware side, the verification system can be packaged into a Qsys module. Given the DUT is also a module with an agreed interface, it can easily be connected in Qsys for testing. On the software side, a user-friendly interface could be built. A usable command line interface may be good enough, but a simple graphic interface could make the tests much more visual and informative.

The interface could also provide information on the failure in the DUT. A better testbench will provide more insightful details when the DUT fails. This would make debugging or evaluating the design much simpler. Along with the GUI, this project has many optional extensions that would be discussed further in section 6.2. After the main goal of the project being met, the number of optional functions implemented becomes a good measure of the progress of the project.

6.1.4 Note

A noteworthy point in evaluation is regarding the progress of the sister project. The purpose of the testbench is to verify and stress arithmetic designs. If these designs are not available near the end of this project, it would be difficult to empirically prove the capabilities of the testbench and its surrounding system. Since the project comprises a verification system, the results from the benchmarks should not be used for evaluation of this project. It is not impossible, as there are still substitutes for them. For functional purposes, standard off-the-shelf arithmetic modules could be used in-lieu. For other purposes, it is possible to have a model done before the actual design starts in the paired project. While this would allow this project to progress, it would be extra work for the other project. In all, it would be nice to have a solid arithmetic module completely to run in this testbench, but without one, the system can still be built and completed, albeit generating less useful data towards the overall aim of the project.

6.2 Project Metrics

6.2.1 Implementation Plan

One natural way of measuring the success of the project is to look at the actual progress and compare it to the implementation plan. It should be noted that no plan is perfect, so some deviation is allowed. However, if there is significant delay from the plan, there must be justifications given.

6.2.2 Fallbacks and Extensions

The fallbacks and extensions were detailed in the implementation plan. The progression of these extension tasks can be used as a point of evaluation to the project. This is already reflected in the product metrics, as 6.1.2 and 6.1.3 examines mainly the success of the extension tasks, while only 6.1.1 analyses the achievements of the core task. However, this does not mean that this project will be completely sealed if all extension tasks are complete before the final deadline, there is always more potential for future work. In the very unlikely case that the project progresses to such a state, new ideas such as dynamic voltage scaling will be drafted and evaluated.

7 Ethical, Legal and Safety Plan

7.1 Ethical Considerations

Checking against the ethical issue list provided by Imperial College Research Ethics Committee [29], this project does not

- damage participants' mental or physical health;
- jeopardise the safety or liberty of the researchers;
- use any private information;
- involve sensitive subject matter or methods;
- risk any conflict of interest between the researchers and the College.

This project is thus free from significant ethical concerns.

7.2 Legal Considerations

Intel Quartus Prime software offers a variety of IP cores. These are encrypted module designs that can be integrated into the verification system of the project [31]. This project integrates some of these IP cores so that some of the basic circuit building blocks does not need to be redesigned.

While it is possible to complete this project with a free licence, Imperial has academic licences for the software allowing for faster compilation time. This licence also allows these IPs to be used for research purposes. As this product is academic, the licence is appropriate and sufficient. This project therefore, has no legal issues.

7.3 Safety Considerations

As the development board used does not have high voltages, there is no safety issues in this project. A detailed safety analysis is available as a separate document.

8 Conclusion

In this report, I have provided motivation for the project through background research. This motivation is then formulated into a organised list of deliverables. With the deliverables set, the tools required to build them are chosen with justification. Next, an initial design sketch is proposed to achieve the specifications. With this, the tasks were broken into smaller piece, which were laid onto a timeline. The tasks which were already complete, were documented; and the tasks in the future, were planned with engineering uncertainties in mind. Examining the expectations of the project together with the plan of implementation, gave rise to an evaluation plan, which will serve as targets during the engineering process, and yardsticks when the project completes. An overview of the non-technical concerns was also provided at the end.

While this report seeks to cover and plan for every aspect of the project, it cannot be perfect. There will always be unexpected changes, which will be reflected in the final report.

I thus look forward to working on this project in the next 5 months.

References

[1] I. Ahmed, S. Zhao, J. Meijers, O. Trescases and V. Betz, "*Automatic BRAM Testing for Robust Dynamic Voltage Scaling for FPGAs*", *Int. Conf. on Field-Programmable Logic and Applications*, 2018.

[2] A Amin, W. Shinwari, "*High-Radix Multiplier-Dividers: Theory, Design, and Hardware*", *IEEE Trans. Comput.*, vol. 1, no.8, 2008.

[3] R.P. Brent, "*A Regular Layout for Parallel Adders*", *IEEE Trans. Comput.*, vol. C-31, pp. 260-264, 1982.

[4] B. Catanzaro, and B. Nelson, "*Higher Radix Floating-Point Representations for FPGA-Based Arithmetic*", *Proceedings of the 51st Annual Design Automation Conference*, 2005.

[5] R. Duncan, "*A Survey of Parallel Computer Architectures*", *Computer*, vol. 23, pp. 5-16, 1990.

[6] J.W. Duran, "*An Evaluation of Random Testing*", *IEEE Trans. on Software Engineering*, vol. SE-10, no. 4, pp. 438-444, 1984.

[7] M.D. Ercegovac, "*On-line Arithmetic: An Overview*", *28th Annual Technical Symposium*, pp. 86-93, Internaltional Society for Optics and Photonics, 1984.

[8] M.D. Ercegovac, and T. Lang, "*Digital Arithmetic*", Morgan Kaufmann, 2003.

[9] S. Hazwani, et al, "*Randomness Analysis of Pseudo Random Noise Generator Using 24-bits LFSR*", *Fifth Int. Conf. on Intelligent Systems, Modelling and Simulation*, 2014.

[10] P. Kornerup, "*Reviewing High-Radix Signed-Digit Adders*", *IEEE Trans. Comput.*, vol.64, no. 5, pp. 1502-1505, 2015.

[11] H. Li, J.J. Davis, J. Wickerson and G.A. Constantinides, "*ARCHITECT: Arbitrary-precision Constant-hardware Iterative Compute*", *Int. Conf. on Field-Programmable Technology*, 2017.

[12] T. Lynch, and M.J. Schulte, "*A High Radix On-line Arithmetic for Credible and Accurate Computing*", *Journal of Universal Computer Science*, vol. 1, no. 7, pp. 439-453, 1995.

[13] T. Lynch, and M.J. Schulte, "*Software for High Radix On-line Arithmetic*", *Reliable Computing*, vol. 2, no. 2, pp. 133-138, 1996.

[14] H.R. Srinivas, and K.K. Parhi, "*High-Speed VLSI Arithmetic Processor Architectures Using Hybrid Number Representation*", *J. of VLSI Sign. Process.*, vol. 4. pp. 177-198, 1992.

[15] K. Shi, D. Boland, and G.A. Constantinides, "*Accuracy-Performance Tradeoffs on an FPGA through Overclocking*", *Proc. Int. Symp. Field-Programmable Custom Computing Machines*, pp. 29-36, 2013.

[16] K. Shi, D. Boland, E. Stott, S. Bayliss, and G.A. Constantinides, "*Datapath Synthesis for Overclocking: Online Arithmetic for Latency-Accuracy Trade-offs*", *Proceedings of the 13th Symposium on Field-Programmable Custom Computing Machines*, pp. 1-6, ACM, 2014.

[17] O. Šćekić "*FPGA Comparative Analysis*", *University of Belgrade*, 2005.

[18] A.F. Tenca, and M.D. Ercegovac, "*Design of high-radix digit-slices for on-line computations*", 2007.

[19] K.S. Trivedi, and M.D. Ercegovac, "*On-line Algorithms for Division and Multiplication*", *IEEE Trans. Comput.*, vol. C-26, no. 7, pp. 667-680, 1977.

[20] P. Whyte, "*Design and Implementation of High-radix Arithmetic Systems Based on the SDNR/RNS Data Representation*" *Edith Cowan University*, 1997.

[21] Y. Zhao, J. Wickerson, and G.A. Constantinides, "*An Efficient Implementation of Online Arithmetic*", *Int. Conf. on Field-Programmable Technology*, 2016.

[22] Accellera Systems Initiative, "*Universal Verification Methodology 1.2 User's Guide*", 2015.

[23] Altera Corporation, "*Cyclone V SoC Development Board Reference Manual*", 2015.

[24] Altera Corporation, "*Memory System Design*", *Embedded Design Handbook*, 2010.

[25] Altera Corporation, "*Introduction to Altmemphy IP*", *External Memory Interface Handbook: Reference Material*, vol. 3, 2012.

[26] Altera Corporation, "*Phase-Locked Loop Basics, PLL*,".

[27] Altera Corporation, "*Creating Qsys Components*", 2018.

[28] Altera Corporation, "*Cyclone V Hard Processor System Technical Reference Manual*", 2018.

[29] Imperial College "*An Ethics Code*", *Imperial College Research Ethics Committee*, 2013.

[30] Intel Corporation, "*Cyclone V SoC Development Kit and Intel SoC FPGA Embedded Development Suite*".

[31] Intel Corporation, "*Introduction to Intel FPGA IP Cores*", 2018.

[32] Intel Corporation, "*Avalon Interface Specifications*", 2018.

[33] RocketBoards.org, "*GSRD 14.1 User manual*", 2015.

[34] Xilinx, Inc, "*Zynq-7000 All Programmable SoC*", 2018.

[35] Xilinx, Inc, "*ZedBoard (Zynq Evaluation and Development) Hardware User's Guide*", 2012.