High radix On-line Arithmetic Verification System

Final Year Project 1800478: Interim Report

Zifan Wang, 01077639
Imperial College London

CONTENTS

I. Introduction

With the right number representation system, it is possible to perform arithmetic operations MSD-first. Consequently, these on-line 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 LSD, making them more friendly to overclocking than their LSD-first counterparts, for which the opposite is true.

In the past, on-line operators have typically been implemented in binary. Along with its sister project, This project will explore high radix on-line operators, investigating their suitability for FPGA implementation and examining the resultant trade-offs between performance, area and power.

II. Background Research

A. On-line Arithmetic

Traditional designs to achieve a faster and more efficient arithmetic operator have two common characteristics. One, their order 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 answers from the MSD to the LSD [1] [10].

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 moving on to the next. [15] This means if a divider follows an adder of the same width, the divider has to wait until the adder complete its computation before it can start its own.

The other commonality of traditional designs is that their precisions are specified design-time. Once built, a 32-bit adder always adds 32 bits together, as the hardware is fixed at run-time. A possible way of making it more efficient would be using SIMD instructions [3], trying to combine smaller operations into a larger one that fits the hardware. This requires more control circuits in the hardware, or a more complex compiler.

On-line arithmetic does not suffer from the first issue as it performs all arithmetic operations with MSD first [5] [6]. Pipelining can be used with on-line serial arithmetic operators. This means the output digit of an earlier operation can be fed into the next operator before the earlier operation been fully complete.

As illustrated in figure 1, while each individual operation may take longer than its conventional counterpart, on-line arithmetic could provide a speed up if the operators are in serial. Individually, on-line arithmetic also sacrifices in terms of memory. As all computation are made MSD to 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. [13]

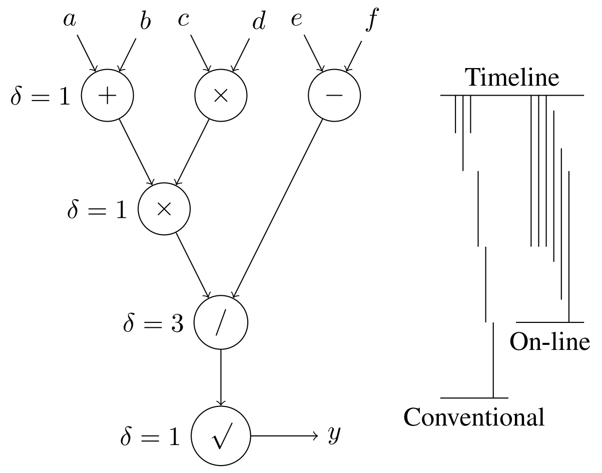

Fig. 1. Computing $y = \sqrt{(a+b)cd/(e-f)}$ with serial on-line operators [5]

A recent architecture proposal allows the precision of on-line arithmetic to be controlled at run-time [15]. Traditionally, this run-time 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 on-line arithmetic alleviates the 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 [11].

However, if the output of a parallel on-line 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, on-line arithmetic would fail gracefully, losing its precision gradually from the lowest bits first. Thus, it allows for a run-time trade-off between precision and frequency [12].

B. High radix Arithmetic

Conventional designs of arithmetic operators use binary representations. This was chosen four decades ago to maximise numerical accuracy per bit of data. However, using a high radix representation system could yield better numerical accuracy while reducing area cost of FPGAs. For example, a hexadecimal floating-point adder has a 30% smaller area-time product than its binary counterpart, while still delivering equal worst-case and better average-case numerical accuracy [2].

However, the savings are not without trade-offs. This trade-off can become unfavourable if the specification requires much I/O and little computation [14]. 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 radix negligible [2].

C. *High radix On-line Arithmetic*

Using high radix number representations for on-line arithmetic is a relatively novel concept. While there have been some research in this field [8] [9], this project takes a more direct approach by implementing custom operators made for high radix on-line arithmetic on a FPGA. This would allow for empirical results to be obtained, hopefully revealing practical insights to the method.

As an potential extension to the project, optimising this exotic arithmetic on a system level for popular FPGA accelerations such as neural networks would be innovative as well.

III. Project Specification

A. *Project Organisation*

This project is a part of a larger project investigating the effect of using high radix number systems with online arithmetic operators. The overarching aim would involve implementing such a system on FPGA and quantifying its performance improvements. This aim would be achieved through two vertically split individual projects. One would design and verify the arithmetic operator modules, while the other would 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 project so that as individual projects, they are individually evaluated. 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 product will work together once they are both complete, a common interface needs to be agreed upon. This falls off directly from the use of the hardware and as such it will be explained in the corresponding section.

B. *Deliverables*

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

1) Takes in the arithmetic modules designed by the sister project as its input;
2) Generate and run tests on these modules;
3) Vary the frequency and voltage of the FPGA;
4) Evaluate its performance.

C. *Hardware Choice*

The system itself will be built on a Cyclone V SX SoC Development Board from Intel [23].

The 5CSXFC6D6F31C6N SoC has an Arm Cortex-A9 MPCore accompanied by Intel's 28nm FPGA fabric [16]. The FPGA is necessary for implementing the hardware design and obtaining empirical results for the project.

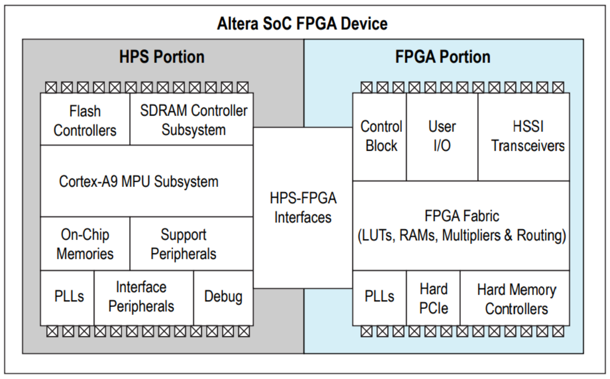

Fig. 2. Structure of the System-on-Chip

While a plain FPGA chip would be enough for this project to work, having an HPS on the same chip is useful as the test software can run on it. While allows a better user interface to be constructed with more detailed, on-the-fly control to the FPGA. This means to setup the testbench would only require programming in the FPGA with the design, and running the test script on the HPS. The product would be self-contained. It would 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, like the Cyclone V SX, Zynq-7000S also features an Arm Cortex-A9 coupled with a Xilinx 28nm FPGA [27]. As such, a board like the ZedBoard [28] could be just as viable.

As there are very few significant functional differences between the two brands, this project will initially explore with the Intel board, simply for its availability and the personal familiarity with their development tools.

Once the project has progressed to a point where the system design is mature and tested, the Xilinx alternative could be explored as an extension.

D. *Software Choice*

The software choice follows closely with the hardware choice in this project. To develop for Intel FPGA, Quartus needs to be used. The version picked is arbitrary as there is not much functional difference between the versions that would be critical to the project. As Quartus Prime 16.0 is the version installed in the computers in the department. The project will be using the same version for the convenience. This naturally means the hardware system will be build with the system integration tool that comes with Quartus – Qsys.

Other than the hardware design tools, there are some freedom of choice in the HPS side of the project. The test will be build with Python, which will be running on an Ubuntu system that is installed in the HPS. This choice is made for there are previous unrelated projects on the same develop board that uses the same configuration, which means a lot of time could be saved on getting an operating system booting.

IV. Engineering Background

A. Target

The design of the verification system is the major engineering challenge of this project. 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 data width of 32-bits, the target data transfer rate is then This required data transfer rate is estimated to be 25.6Gbps.

B. Data Transfer Rate

As the test is to be run on the HPS, the HPS-FPGA bridge will be the immediate bottleneck if the test data is to flow from HPS to FPGA. While HPS is able to easily generate test data, there is a large amount of overhead as data crosses from one architecture to another. This overhead exists in terms of both a lowered bandwidth and a high delay. Thus, it would not be sensible for HPS to send out data during run-time.

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

While using the off-chip RAM may theoretically achieve the targets, it still has its disadvantages. First, the process of filling up the memory and then using them for the tests takes time. This means the test would be broken up into bursts with time in between for checking the results and filling up with new data. The complexity of the SDRAM interface also requires a SDRAM controller to be used to manage SDRAM refresh cycles, address multiplexing and interface timing. These all adds up to a significant access latency. While this could be overcame with burst accesses and piplined accesses, it would further complicate the SDRAM controller. While this controller is provided by Altera [18], it is consumes a non-negligible amount of the limited FPGA resources, while adding unnecessary complexity to the design. Customising the controller to fit this project may also be time-consuming.

The on-chip memory is much faster and simpler to handle. This memory is implemented on the FPGA itself, and thus there is no external connections for access to this memory. It has the highest possible throughput, with the lowest possible latency in an FPGA-based system. 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 at one end and the reads at the other end can happen simultaneously. This effective doubling of the bandwidth is useful as tests are prepared and fed into the DUT, or when test results are collected and fed to a checker.

On-chip memory is not without its drawbacks. It is volatile and very limited in capacity. While the off-chip can have its storage reaching 1GB, that of the on-chip memory could only reach a few MB [17]. 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.

C. Run-time Test Data Generation

To exploit the benefits of using on-chip memory, a way of generating test data at run-time 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 the use of random testing. With relatively minimum effort, random testing can provide significant coverage and discover relatively subtle errors [4]. The main drawback of random testing is the possible lack of coverage on extreme cases, and the usual solution is to provide handwritten tests to complement random testing. However, as the main goal of this testbench is mainly gauging the performance of the module and not necessarily verifying the correctness of the arithmetic module, this could be ignored during stress testing. If logic correctness testing is later required, these special tests could be written and run separately at a slower speed.

LFSRs are a reliable way of generating pseudo random numbers fast with minimum cost [7]. They will thus form the starting point of data generation. While this should not be the case for most benchmarks, if some of the data is invalid, they will be dealt with at the monitor.

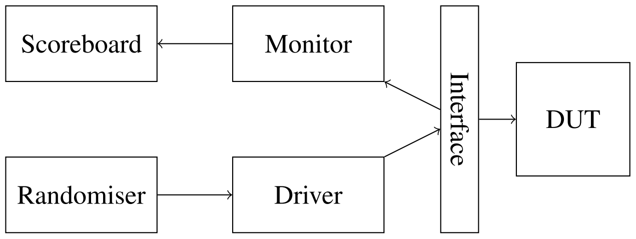

Fig. 3. Testbench block diagram

D. Clock Domains

Another concern in the system design is of the different clock domains that must exist on the FPGA. At a minimum, there needs 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 and distributed with PLLs provided as an IP in the Quartus software [19].

E. Testbench Architecture

Chapter 4 of Shi's paper [11] provides the structure for its verification system.

V. Implementation Plan

A. Milestones

The initial deliverable for the engineering side involves running a simple program on the FPGA through the HPS with the FPGA frequency being controllable. After this, the next critical step would be making sure the modules under test will be the point of failure and not the testbench. This would include some research on ways in improving the speed of feeding inputs to the arithmetic units, and checking its outputs. Once this could be confirmed, we can start adding a selection of different functionalities.

1) Running standard benchmarks;
2) Running key algorithms or their components;
3) Experiment with other power efficiency improving techniques, such as undervolting;
4) Add support for configurable radix arithmetic;
5) Allow graceful failures for the testbench in case of unintended behaviour for the arithmetic modules;
6) Add an interactive UI to control the voltage and frequency at run time and examine the DUT's behaviour;

Depending on the time situation, more or less items on this list may be fulfilled. The method of evaluation will be discussed later in the evaluation chapter.

B. Timeline

In order to track the progress and success of the project, the difficulties of the deliverables need to be analysed first. Figure 4 provides a visualisation of the project timeline.

1) ***Term Time and Examinations:*** Time available for the project varies greatly throughout the year. The greatest factors would be the term time and the examinations. The time needed for revision has been marked in the chart. These times will give minimum progress to the project.

During term time, there would be coursework deadlines, which would also negatively affect the time that could be allocated to this project. One coursework module was selected for Autumn and three was picked for Summer. As such, it is expected that the progress would be somewhat slower during Autumn but significantly slower during Spring. To make up for the time lost, a part of Christmas were 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 is done according to its expected difficulty and the expected availability to work on these tasks.

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 is to provide context and motivation to the project. It also provided an overview of the field of research and some understanding towards the current state-of-art. Most importantly, it offered the necessary tools and knowledge needed for this project to be successful.

3) ***Learning the Tools:*** Due to the lack of experience in programming in a hybrid SoC and the lack of knowledge in the current state-of-art digital arithmetic designs, a significant portion of the effort was spend on researching and learning the skills necessary to carry out the project. This involved building a small testing system on the board. Details regarding this testing system can be found in the Work to Date section.

4) ***Testbench Structure:*** Once comfortable with the tools, The main design of the testbench can start. This task would be the foundation of this project, as it would provide a basis for all following features. A skeletal testbench should be complete and functional at the end of this task. This means the an operator module matching the correct specifications can be loaded into the testbench, and compute some non-trivial tests.

5) ***Variable Frequency:*** As the project seeks to quantify the performance across a range of frequencies, the most important feature of the system would be the ability to vary this parameter. This would be done by controlling the PLLs with the HPS [19]. Once implemented, some kind of test will be run to test and confirm the maximum frequency the testbench would remain reliable. If it does not meet the planned target, the testbench may need to be redesigned, and this project would be under some risk.

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 bare minimum system gets done early. To ensure this happens, this task and the testbench structure will be placed in the highest priority before its completion, and any blocking issue should be discussed with the supervisor if it could not be resolved after reasonable effort.

6) ***Benchmarks:*** With a promising base, more intricate tests and benchmarks can be designed. These would aim to reflect the system's performance running meaningful compute tasks. The systems enveloping the arithmetic modules could be optimised for popular algorithms to evaluate real-life obtainable speed up. In addition to better tests, this task also aims to obtain better data from these tests. The minimum result required here would be numerical information on power consumption, FPGA resources required, and the data throughput.

The last three tasks forms the core of this project. In other words, all three tasks must be completed for a minimum functional product. The following tasks would be mostly considered as useful extensions. While not as vital as the core tasks, the following tasks greatly improves the quality of life and usability, and are thus equality critical to the success of the final product.

7) ***Configurable Modules:*** So far, only modules of a specific I/O width and numerical representation could be tested. It might be interesting to explore arithmetic modules with other configurations. To allow the testbench be used for further experiments or future projects, it is helpful to have a configurable testbench. Qsys components can be configured with the Hardware Component Definition File [20]. The plan is to build the testbench as a Qsys components, then use Qsys as an interface for configuration.

8) ***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 is 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 during run-time may degrade performance of the testbench, so it would be useful for the verbosity of this information to be configurable by the user.

9) ***Interactive UI:*** If time allows for even greater user experience enhancements, an real-time interactive graphical user interface could be constructed for the final demonstration. This would visualise the reduction of the module's precision

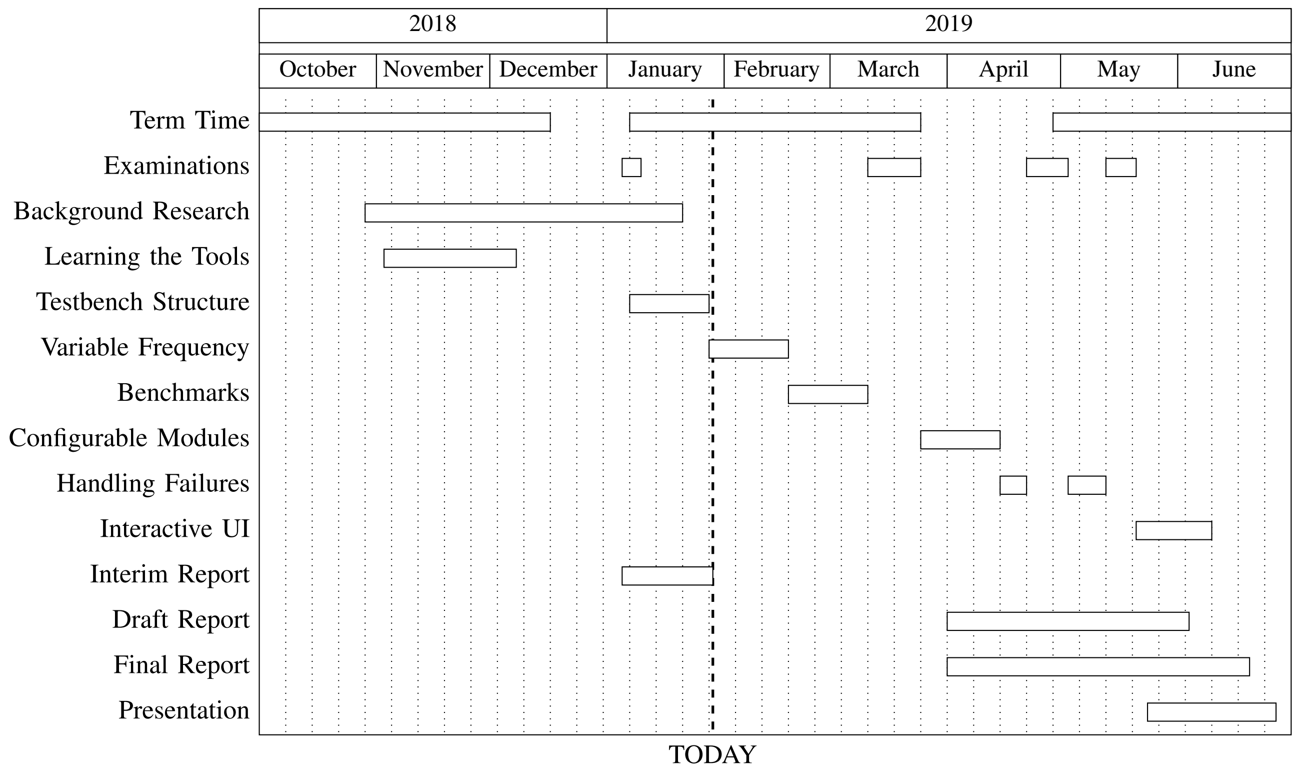

Fig. 4. Project Timeline

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. An time-saving yet functional alternative would be a command-line interface with a well documented user guide.

*10) **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. They will be written alongside with the engineering process. At the end, around a week of time will be spent solely on completing and polishing up the report for submission. This should allow ample time for a well-organised report to be written.

The week after the final report will be used for the presentation. This would involve preparing a slide deck, a demo, and a script.

C. Work to Date

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

*1) **FPGA Side**:* Programming the FPGA to communicate with the HPS is no trivial task. Luckily, there exist a golden system reference design [26] 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 could be opened with a different version of the software, it would cause 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 a three bridges for connections [21]. As this is a relatively simple task, the lightweight bridge is used. Module `altera_hps` exposes the master of this bridge as `h2f_lw_axi_master`. Next, the actual hardware adder needs to be built and integrated as a hardware module in Qsys with a matching interface. A simple adder can produce the result after one clock cycle. This greatly simplifies the logic required for the Avalon slave interface. The logic for the control and data signals are then written according to the interface specifications [25]. 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 is designed with variable width.

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

2) ***HPS Side:*** The HPS runs Ubuntu and a bash script has been written to load the bitstream onto the FPGA. Next, A program is 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 [21], 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 is then designed to have its two inputs at `0x00` and `0x10` and its output at `0x20`, which has been assigned by Qsys relatively to `0xFF30_0000`.

With the memory mapping understood, the script can be designed to closely mirror this relative relationship between the modules using classes. For example, this allows the adder to still define its output at `0x20` in the adder class, and then initialised with an axi module bringing it to the correct physical address. This parallel between software and hardware should be helpful as the product gets more complex.

For testing, 1000 add operations are executed separately with and without the hardware acceleration of the FPGA. While called hardware acceleration, it is not expected for the FPGA to have a higher performance than the HPS in this testing case. The CPU is reasonably efficient in calculating additions, while to calculate on the FPGA has a large overhead cost as the data transfers across the bridge back and forth.

VI. Evaluation Plan

A. Metrics

One natural way of measuring the success of the project is to look at the actual progress and comparing it to the plan given in 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 implementation plan, there must be justifications given.

The next few measures looks at the performance of the final product. First, the maximum stress of that the testbench and provide without failing can be used as a success measure. A 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 would be a vital indicator of the project's success.

The Robustness of the testbench is also vital to the product's performance. The testbench should be free from errors within a reasonable operating range. If the testbench becomes unreliable with some minor changes to the system, the data that can be obtained would be very limited. The failure of the DUT can no longer be confirmed, as the error may be in the testbench instead of the DUT.

The ease of use of the testbench could 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, they could be easily connected in Qsys for testing. For example, the DUT may be written in VHDL for its deterministic nature, but the testbench maybe written in Verilog for its simplicity, but as both can be synthesised into a module, they would still be compatible in Qsys. On the software side, a user-friendly interface could be built. A usable command line interface maybe good enough, but a simple graphic interface could make the tests much more visual and interesting.

The interface could also provide information on the failure in the DUT. A better testbench would 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 the corresponding section. After the main goal of the project being met, the number of optional functions implemented would become a good measure of the progress of the project.

Since the project is of the verification system, the results from the benchmarks should not be used for evaluation of this project.

B. Risks and Fallbacks

The other major risk has to do with the progress of the sister project. The purpose of the testbench is to verify and stress the arithmetic designs. If these designs would not be available near the end of this project, it would be difficult to empirically prove the capabilities of the testbench and its surrounding system. It is not impossible, as there are still substitutions 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 easier, it would be extra work for the other project, which is ultimately up to the decision of the other student. 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.

C. Extensions

VII. Ethical, Legal, and Safety Plan

A. Ethical Considerations

Checking against the ethical issue list provided by Imperial College Research Ethics Committee [22], this project

- does not damage participants' mental or physical health;
- does not jeopardise the safety and liberty of the researchers;
- does not use any private information;
- does not involve sensitive subject matter or methods;
- does not risk any conflict of interest between the researchers and the College.

This project is thus free from significant ethical concerns.

B. Legal Considerations

Intel Quartus Prime software offers a variety of IP cores. These are encrypted module designs that would be integrated into the verification system of the project [24]. This project would integrate some of these IP cores so that some of the basic circuit building blocks would not need to be redesigned.

While it would be possible to complete this project with a free license, Imperial has academic licenses for the software allowing for faster compilation time.

C. Safety Considerations

As the project is done mainly on a computer with minimum physical aspects, there is no major safety concern. For the minor concerns associated with the project, the physical development board will be handled with care, and the desk works will be interleaved with breaks.

VIII. Conclusion

References

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

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

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

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

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

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

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

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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