High radix On-line Arithmetic Verification System

Final Year Project 1800478: Interim Report

Zifan Wang, 01077639
Imperial College London

I. Introduction

This project is a part of a larger project investigating the effect of using high radix number systems with on-line 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.

II. Project Specification

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.

A. Project Interfacing

A number representation system should be decided before detailed design starts.

The modules would be compiled hardware in

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 [1].

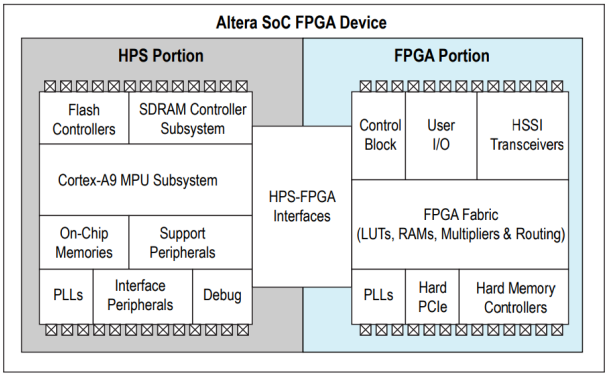

Fig. 1. Structure of the System-on-Chip

The 5CSXFC6D6F31C6N SoC has a Am Cortex-A9 MP-Core accompanied by Intel's 28nm FPGA fabric [2]. The FPGA is necessary for implementing the hardware design and obtaining empirical results for the project. Having an HPS on the same chip is useful as the test software can run on it, allowing a better user interface to be constructed with more detailed, on-the-fly control to the FPGA.

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 a Arm Cortex-A9 coupled with a Xilinx 28nm FPGA [3].

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.

E. Deliverables

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.

III. Background

A. On-line Arithmetic

Traditional proposals to achieving 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 [4] [5].

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. [6] 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 [7], 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 [8]. Pipelining can be used with on-line 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.

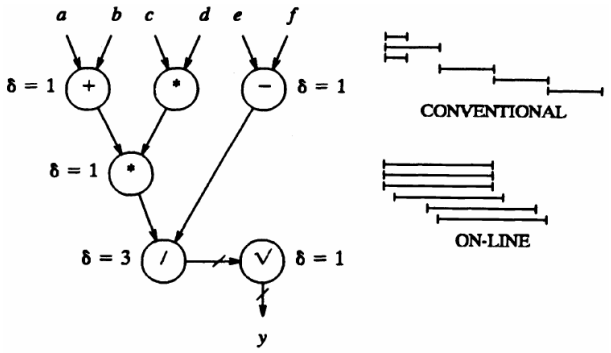

Fig. 2. Computing $y = \sqrt{(a+b)cd/(e-f)}$ [8]

As illustrated in figure 2, 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. [9]

A recent architecture proposal allows the precision of on-line arithmetic to be controlled at run-time [6]. 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.

Another way that 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 [10].

However, if the output of an 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 [11].

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 [13].

However, the savings are not without trade-offs. This trade-off can become unfavourable if the specification requires much I/O and little computation [12]. 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 [13].

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, they are

Compared to existing research, the combination of high radix and on-line is relatively novel, and to optimising it on a system level for popular FPGA accelerations such as neural networks would be innovative as well.

With respect to the testbench, figuring out a system to obtain the optimal point of operation of the FPGA, in terms of its voltage and frequency, would also require some advanced research.

IV. Implementation Plan

A. Risks

A major risk of this project is related to its schedule and the existence of an initial blocking task. While most of the later sections of the project can be selectively added or removed from the scope relatively easily, the initial setup of the testbench will always remain critical to any further improvements. It is thus vital that the bare minimum system gets done early. To ensure this happens, it 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 significant effort.

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 adders and multipliers 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.

V. Evaluation Plan

VI. Ethical, Legal, and Safety Plan

VII. Conclusion

Appendix A
Formulae

$$1 + 1 = 2$$

Appendix B
Data

Sum	1	2	3	4
1	2	3	4	5
2	3	4	5	6

References

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

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

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

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

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

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

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

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

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

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

[11] K. Shi, D. Boland, E. Stott, S. Bayliss, 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.

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

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

[14] T. Lynch, 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.

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