AFRL-RI-RS-TR-2021-101

OPTIMIZATION AND EXPLORATION OF TRUSTED LOW-POWER HIGH PERFORMANCE COMPUTER ARCHITECTURES

OKLAHOMA STATE UNIVERSITY

JUNE 2021

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2021-101 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
JONATHAN HEINER
Work Unit Manager

/ S /
GREGORY HADYNSKI
Assistant Technical Advisor,
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JUNE 2021	FINAL TECHNICAL REPORT	SEP 2017- DEC 2020

4. TITLE AND SUBTITLE

OPTIMIZATION AND EXPLORATION OF TRUSTED LOW-POWER HIGH PERFORMANCE COMPUTER ARCHITECTURES

5a. CONTRACT NUMBER
NA

5b. GRANT NUMBER
FA8750-17-1-0261

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

James Stine

5d. PROJECT NUMBER
97SP

5e. TASK NUMBER
ST

5f. WORK UNIT NUMBER
NE

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Oklahoma State University
401 Whitehurst Hall
Stillwater OK 74078-1030

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2021-101

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The research objectives of this project are to design, develop, and evaluate multi-core hardware support for secure computer architectures at then at the nanometer level. Many of these architectures are currently or will be employed in advanced architectures that may have secure capabilities within the Air Force Research Laboratory in Rome, NY. This will be accomplished by designing complete design flow integration with commercial and open-source Electronic Design Automation (EDA) tools. The design flow will take a high-level system-level architecture description as inputs along with area, critical path delay, and power dissipation constraints. Based on the System-on-Chip (SoC) architecture description and design constraints, the tools will automatically generate synthesizable hardware description language (HDL) models, embedded memories, and custom components to implement the specified very large scale integration (VLSI) architecture.

15. SUBJECT TERMS
Secure Processor, Hardware Root of Trust, Microcode

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**JONATHAN HEINER**
U	U	U	UU	30	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE **Page**

1.0 SUMMARY

The growing need for new technologies, lower power, abilities to handle smaller feature sizes, and other demanding issues need to be overcome for the United States to produce competitive designs within silicon technology. The National Academy of Engineering, Science and Medicine argue that limitations in technologies are of particular importance for the growth of new areas and better ideas to benefit society [1]. This design project aims to accelerate the pace of innovation in Micro/Nanoelectronics by significantly improving designs for computer arithmetic and low-energy systems especially for architectures that need high-performance and security, which are essential for our national security, wellbeing, and economic growth.

The difficulty of creating innovative and dynamic chip designs has become a challenge over the last several years. Moreover, Moore's law continues to drive the scaling of complementary metal-oxide semiconductor (CMOS) technology despite the feature size of the transistor having now shrunk well into the nano-scale region [2]. Companies that have significantly grown niches, such as Arm and Xilinx, now dominate in creating Intellectual Property (IP), and they typically do not allow research institutions to integrate these designs into that design process. Specifically, these companies have their own set of tools and intellectual property (IP) that have multiple layers of secrecy and underlying infrastructure. All of which make it difficult for research entities to reproduce designs in the area. More importantly, there is a large gap between what a research institution has access to and what is needed to make a design viable. This makes it very hard for researchers to produce designs that function properly.

This research proposes to generate and maintain a suite of hard IP blocks to provide the necessary building blocks for designing a working system-on-chip (SoC). This includes essential components such as the Institute of Electrical and Electronics Engineers' (IEEE) 754 floating-point, integer arithmetic units, and cryptographic generators. These parts are extremely helpful in SoC design to have a library of silicon-proven hard-IP blocks that are reliable and can be reused multiple times in future designs. More importantly, as opposed to commercial generators that create blocks that promote secrecy and hide underlying details, this research attempts to create full-functioning blocks that can allow the IP developed for the Air Force Research Laboratory (AFRL) flourish and thrive.

The major research emphasis in this proposal is placed on designing a complex Very Large Scale Integration (VLSI) multi-core architecture along with hard IP blocks using an elaborate design flow or sequence of steps. Several software tools and validation suites are also created to assist designers in specific computer architectures that are robust, have high amounts of performance, and yet are considered mobile in that they may consume small amounts of power.

Therefore, the goal of this project is to research and develop high-level synthesis tools for SoC platforms in nanometer CMOS technologies that (1) provide the ability to efficiently integrate embedded memories, low-power/high-performance circuits and processors, mixed-signal designs, and communication structures, (2) combine synthesis and layout information to accurately estimate area, delay, and power from high-level SoC architecture descriptions, (3) facilitate rapid design-space exploration of secure SoC solutions, and (4) are well documented, easy to use, and publicly available for AFRL personnel. It is anticipated that many of the outcomes of this project

will aid in the development and deployment of silicon architectures for any division that employs trusted foundry fabrication capabilities.

The hardware designs and algorithms developed are expected to be significantly faster and more efficient than existing modeling packages, not to mention accurate and repeatable. The accuracy will come from real extraction material from the actual silicon processing. This increase in performance will give researchers and programmers the ability to solve problems that were previously intractable. A better understanding of the relationship of hardware and instruction set design to processor cost, performance, and power consumption is also obtained.

2.0 INTRODUCTION

The demand for increased speed, decreased energy consumption, improved memory utilization, and better compilers for processors has become paramount to the design of the next generation of computer architectures [3]. To make matters worse, the traditional challenges of designing digital devices with semiconductor technology has drastically changed with the introduction of deep submicron technology. Designs that have long been expanding by Moore's Law have discovered that silicon technology has severe limitations within technologies below *180* nm [4]. What was once easy to improve a design by scaling the minimum feature size of a transistor can no longer be simply scaled at a lower technology.

Because silicon technologies are so small, designs can now implement billions of transistors on a reasonably small die. Unfortunately, this leads to the power density and total power dissipation that is at the limits of what packaging, cooling, and other infrastructure can support. More importantly, Complementary Metal Oxide Semiconductor (CMOS) technologies below 90nm leakage current almost matches or surpasses that of dynamic power, making power dissipation a major obstacle to designing complex SoC designs. Although power dissipation complicates the process to which integrated circuits can be produced, it does not necessarily mean that designs cannot be efficiently designed. A designer just has to be cognizant that performance does not necessarily mean that one can increase the clock rate, as technology grows smaller. This new challenge requires designers to realize that both power and speed are closely linked, and that engineering choices or sacrifices are normally required if a design requires a lower power factor or high clock rates.

To make things more challenging, processor designers have increased core counts to exploit Moore's Law that has made decisions about having multiple cores and maintaining high performance while still exhibiting low-energy dissipation [5]. More importantly, single core processor designs are the engine that ultimately makes multiple core devices work. Therefore, for virtually all applications, including general-purpose computer architectures, reducing the power consumed by SoCs is essential to allow new features and add performance to improve technology [6]. Consequently, it is extremely important to understand what, where and how power consumption affects SoC designs to improve upon it for both single and multiple-core processors. Interestingly, applications-purpose designs also affect both high-performance and low-energy consumption and utilizing intelligent and novel approaches [7].

2.1 Power Consumption Basics

The total power consumption of a digital logic circuit consists of two major portions. The first part consists of dynamic power that is the power that is consumed when a device is active. Typically, dynamic power is consumed when devices are active and are switching back and forth. That is, they are based on what is supplied at the input of a circuit. If, for example, a circuit has lots of activity (e.g., within a cell phone using the Internet), it will typically consume lots of dynamic power. Conversely, applications that only switch on during critical events (e.g., sensors within automobiles for abnormal events), typically consume very low amounts of dynamic power. Dynamic power for the longest time was the dominant element for energy dissipating in digital circuits and also the driving motivator for smaller feature sizes [4].

In addition to switching power, short-circuit power dissipation also contributes to the dynamic power in digital circuits [4]. This power dissipation occurs when a CMOS gate is suddenly turned from on to off and back to on. The switching causes both n-type metal-oxide semiconductor (NMOS) and p-type metal-oxide semiconductor (PMOS) transistors to be on momentarily resulting in a short-circuit or "crowbar" current [4]. Although the short-circuit current can be small, it can contribute to the total dynamic power if the input is ramped up too quickly [8]. Although short-circuit current does not contribute large amounts to the energy a device consumes, it is still important to make sure gates are not floating on an output when enabling power-gating to a digital circuit for lower-power consumption.

The last component for power dissipation in digital circuits is static power dissipation which is defined as the power consumed when devices are powered up and no signals are changing values [4]. In the past, static power dissipation, which is mainly dominated by leakage current in a gate, was either non-existent or did not significantly impact a design. However, as the voltage and minimum feature size of a transistor gets smaller, the pronounced effect of leakage within a gate makes static power dissipation almost equal to or greater than dynamic power below 90nm [9]. Moreover, in some designs static power can dominate a design and also provide exponential complexity as the feature sizes decrease and even cause problems related to the yield of silicon devices [10].

In summary, power dissipation is composed of three major components: dynamic, switching, and static power dissipation. Balancing power dissipation involves several strategies that include use of multiple threshold voltage transistors, sleep transistors and an assortment of similar approaches [11]. Regardless of the technique used to reduce the amount that a digital circuit turns on or off, energy dissipation approaches tend to work better when designs have smaller footprints or lower area consumption [12]. The designs presented within this work utilize some of the generic techniques for low-power dissipation, such as multi-threshold CMOS (MTCMOS) gates, but also employ techniques to efficiently design logic blocks more efficiently and with more parallelization to avoid larger area consumption.

2.2 Scientific Computing and Use of IEEE 754 Floating-Point

Computing has changed drastically since the original idea of computation was invented by Alan Turing in the 20th century [13]. Since computers have been invented, computation has involved using binary digits or bits to compute numbers [14]. Although fixed-point registers have been growing in size from 16 bits to 128 bits, the problem with dealing with precision loss has been a difficult and ongoing problem [15]. The IEEE proposed the IEEE 754 in 1985 to deal with problems associated with this loss of precision by creating a new standard dealing with floating-point or scientific notation [15], [16].

The IEEE 754 standard developed a standard approach in dealing with scientific notation by creating a uniform view of floating-point numbers. Originally, the IEEE 754 1985 standard has two basic formats single and double precision that dealt with 32 bits and 64 bits, respectively [16]. This standard assigns three fields to denote the Sign (S), Exponent (Exp) and Mantissa or Significand (f) of a number. The sign bit is always 1 bit and allows both positive and negative numbers to be easily assigned [15]. That is, the IEEE 754 standard creates a unique representation for any scientific number by using the following formula:

$$(1.0)^S \cdot 1.f \cdot 2^{exp-bias} \tag{1}$$

In order to make sure exponents are easily decoded and also can handle comparisons easily, they are organized as unsigned numbers offset by a bias defined in the standard [17]. Within the standard, all biases are computed as the half-way number based on the size of the exponent field. Moreover, the most-significant bit within the mantissa is always a 1 because the IEEE 754 standard defines values between 1 and 2 (i.e., [1,2)), therefore, this bit is never stored but implied always as a 1 [15]. Although the IEEE 754 has two basic floating-point point formats, it was expanded to handle quadruple precision as well as half precision [18]. Half-precision floating-point numbers were added to the IEEE 754 standard to deal with cases where fast computation is necessary using smaller formats, such as within machine learning applications [19]. The values for several sizes in the IEEE 754 standard are summarized in Table 1.

Table 1: IEEE 754 Bit Sizes for Different Formats

Size	Bit Length	Mantissa Size	Exponent Size	Bias
Half	16	10	5	15
Single	32	23	8	127
Double	64	52	11	1,023
Quad	128	112	15	16,383

Computation is important in limiting the number of cycles that an application executes [14]. This ultimately translates to the speed that computing improves at [20]. However, although faster clocks exhibit better performance, they come at the price at higher power dissipation and ultimately more complexity that can lead to more difficult validation [21], [22]. Therefore, it is important that computation be done quickly but also efficiently with high amounts of parallelization to promote parallelization but high throughput [23].

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 Optimization and Exploration of Trusted Low-Power High-Performance Computer Architectures

Many design flows or design scripts involve taking structural or behavioral descriptions of computer architecture and translating into a working silicon mask layer that can be fabricated. Although this process is just an evolution of what software compilers use, this process has dramatically changed from early designs involving several hundred transistors to current SoC designs that contain close to or exceed 1 billion transistors [24], [25]. To make matters worse, power and high-performance issues have complicated the entire process.

Complex arithmetic architectures typically employ commercial IP that comes from manufacturers of Electronic Design Automation (EDA) software. This IP is usually in the form of higher-level constructs that force internal data structures to generate architectures without indemnity related issues [26]. For example, Synopsys, a popular EDA company, uses something called DesignWare that creates automatic optimized implementations of popular algorithms, such as IEEE 754 arithmetic [27]. Although these elements within IP inside EDA tools are effective, they hide much of the details from a user. Subsequently, they can cause issues related to implementing architectures that use a subset of the IP they generate [26]. They can also cause much of the optimization of multiple blocks to be significantly smaller than they would separately as a complete unit can optimize all structures equally. This works attempts to address some of these issues by generating much of the IP for use with the Air Force Research Laboratory based on the expertise of the principal investigator of this project.

All of the designs are created with the cutting-edge and most-recent EDA software provided by EDA tool vendors. The process of going from design architecture to deployable silicon, or back-end process, is utilized from Cadence Design Systems separately and self-contained within one set of scripts. Cadence Design Systems® (CDS) utilizes Innovus™ for place and route data. The front-end process or the synthesis portion of the design flow employ Synopsys® *Design_Compiler*™ (*DC)* for turning the SoC architecture into an implementation based on a technology-mapped netlist (i.e., the front-end process). The simulation tool ModelSim from Mentor Graphics/Siemens is employed in each design for HDL simulation and verification, since it is rich and full featured to contain most ideas within simulation.

The following elements were developed in this work for the Air Force Research Laboratory:

- Exploration and implementation of several cryptographic implementations in Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL) and Verilog for modular arithmetic and its associated cryptographic computations.
- Create extensible test environments that allow for easy chip exploration and analysis for IEEE 754 arithmetic and other cryptographic architecture implementations.
- Full IEEE 754 floating-point and integer arithmetic implementations in VHDL and Verilog to help explore advanced computation and efficiency in implementation.

Each of the subtasks is described in the following subsections. As summarized above, the subtasks together focus on the development of high-level EDA tools for low-power SoC designs specifically designed for a high-performance computer architecture that also exhibits low power dissipation.

3.2 Modulo Reduction for Hardware

The modulo operation is becoming increasingly important in the world of computer architecture as it is used more and more in modern encryption techniques. However, while the radix of the required modulus has increased, our hardware has not been adapted to efficiently calculate high radix sizes for higher security requirements. Generally, the modulo operation is accomplished by a device that is close in size and function to a full divider in which the quotient is abandoned and the remainder is utilized for the final operation. Although this is mathematically correct it can be quite wasteful as division is a complicated arithmetic operation.

While it may seem a simple task to compute the remainder of the division operation on its own, it is actually a complicated operation to perform in hardware. While there are many methods proposed to compute such an operation, it is important to create efficient hardware to avoid wasteful energy and time dependence. Although there are methods that are quite efficient utilizing Montgomery multiplication, they tend to be more complicated and area/energy intensive [28], [29]. Other methods, such as double-add-reduce (DAR) methods are simplistic and slow [30].

One type of reduction that is quite simple is the 2^{k-a} reduction [31]. The 2^{k-a} hardware reduction algorithm is simple but requires quite a few cycles and utilizes hardware components that require a lot of time for calculation. Another reduction algorithm is Barrett reduction, the current gold standard for fast reduction. With Barrett reduction, some math is utilized in order to calculate $x \bmod m$ without the need for looping which would allow us to, in theory, make a reduction module that takes a small number of clock cycles. Unfortunately, the arithmetic operations required for the calculation of $x \bmod m$ using Barrett reduction is quite complicated and requires the use of a multiplier as well as some other slow hardware arithmetic components.

The Barrett reducer shown makes use of both a multiplier and subtractor in order to calculate the modulus of the input value x. A multiplier implementation, especially one of sizes such as 256 bits and 384 bits, requires a large amount of logic, area, routing, and delay in order to place and route into a design. Multipliers also consume large amounts of energy as they have to add multiple times to form the final product [29]. One particular alternative method for performing modular reduction is presented in [32] and utilizes only shifting, adding, and subtracting in order to accomplish the remainder calculation. Although the implementation in [32] is promising, it does not provide any hardware algorithm or implementation.

The architecture in this work utilizes the implementation in [32] as it has proven benefits over previous algorithms when it comes to software implementation. In the original paper [32], the authors compared the proposed algorithm with Barrett reduction, Montgomery reduction, fast modular reduction, and a real-world test in which the new algorithm was compared with the GNU multiple precision algorithm. When compared with Barrett reduction it was found that the proposed algorithm had a performance increase not in the number of operations required but in the complexity of the operations as Barrett reduction requires a multiplication while the [32] algorithm does not. In addition, when compared with traditional Montgomery reduction it requires more add

operations, on average, than the [32] and also sometimes requires a few "correction" steps in order to get the correct answer as it does not always get it on the first try. Most importantly, the algorithm proposed in [32] computes fast modular reduction method but only requires processing the upper half of the bits and, therefore, only requires $radix/2$ operations, but the fast modular reduction method. This ultimately translates to lower energy requirements as well as smaller area consumption.

Cryptographic systems require architectures that can do the modulo operation. Therefore, this work integrates the design from [32] using the implementation found in [33]. This datapath shown in Figure 1 has two distinct sections that are the initial shifting and adding section as well as the final adjustment section. In the initial shifting and adding section there is a multiplexer that is used to select between loading of a new input value or the output of the shift and add section. This multiplexer is subsequently used to select the input for the flop with enable. This register is only enabled when the shift_done flag is 0. This flag or enable becomes a 1 when the counter finds that the module has done a number of shifts equal to the size of the reduced output in bits. When disabled, the register will hold a steady state of the value calculated from the shift and add. When enabled, and not loading in a new value, the shifter and adder loop into the set of multiplexers that then decide the value to be loaded into the register.

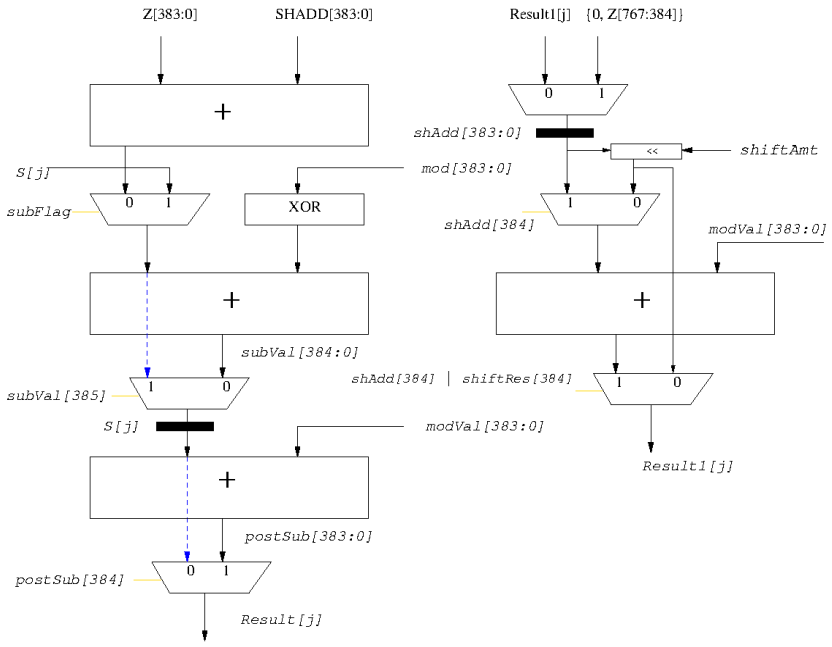

Figure 1: Mod Without Mod Architecture

An important improvement to this implementation is quite simple but can also be designed with replacing some of the basic components with more complicated ones, replacing the carry propagate adders with fast prefix adders, or adding logic to compound an operation that takes several cycles into one that takes far fewer cycles. One example that has been implemented for this paper is the use of a leading zero detector (LZD) [34] in order to shift out all of the leading zeroes of the result.

This allows the reducer to save one cycle for every recurring leading zero. For example, if the result value saved in the register has 4 leading zeros, then the implementation will save three cycles by shifting out all four of these zeros at a time. The LZD is added by using a variable shifter instead of a single shift for the shifting portion. The LZD itself resides inside the counter which passes the number of shifts needed to shift out all of the zeroes back to the datapath and subtracts either the number of zeroes or the entire remaining count from the counter. This allows for considerable speedup on a large number of operations.

The proposed reducer is compared with the Barrett reducer which is considered the best option for modular reduction units in terms of execution speed. The Barrett reducer is an efficient piece of hardware as it requires a small number of clock cycles to calculate the modular reduction operation, but this is a tradeoff as due to its use of the significant size of the multiplier required as it has both high area, energy dissipation, complexity, and critical path delay. Both the Barrett and the proposed reducer are implemented using a 768×384 bit reduction form and are verified using the modulus values for the National Institute of Standards and Technology (NIST) P-384 curve [35]. The implementations are compared in various nanometer CMOS technologies using the tools previously mentioned using ARM-based standard-cells in Global Foundries CMOS technology shown in Table 2. As expected in Table 2, the Barrett reducers clearly completes with fewer number of cycles, however, at the expense of the significant energy and area overhead.

Table 2: Comparison of the Barrett Reducer vs. Proposed Reducer with P-384 modulus

CMOS Feature Size	Barrett Reducer			Mod without Mod			Mod without Mod w/LZD		
	Area [um^2]	Delay [ps]	Total Power [mW]	Area [um^2]	Delay [ps]	Total Power [mW]	Area [um^2]	Delay [ps]	Total Power [mW]
45nm	1,226.1	1,040.0	157.8	42.4	438.0	2.9	92.1	617.0	8.6
32nm	452.6	1,018.1	104.6	16.6	359.6	4.7	38.6	515.0	13.7
14nm	186.3	1,103.4	8.9	10.4	437.4	0.3	16.3	655.2	0.6

By comparing the design with the Barrett reducer in terms of critical path delay it is easy to see that this new implementation is dramatically better with or without the leading zero detector. This is due to the large critical path caused by the 384×384 bit multiplier in the Barrett reducer. When comparing the proposed design with the proposed design with the LZD, an increase occurs due to the use of both a variable shift shifter as well as the leading zero detector itself. Overall the proposed design without the LZD is able to run at over double the speed of the Barrett reducer on average while the design with LZD is able to run approximately 25 percent faster.

By examining the area results it is once again obvious that the proposed design wins outright, this is likely again to the 384×384 bit multiplier that takes a large amount of routing and space to do the multiplication logic. When comparing the values between the proposed design without the LZD, the multiplier width causes the multiplier to unreasonably scale to larger radix sizes. That is, with a large size the area of the proposed design scales relatively linearly whereas, the Barrett reducer has trouble with the large area required to accomplish the multiplier required for Barrett reduction. When comparing the design with LZD to the Barrett reducer it is again obvious to see that the design scales much better than the Barrett reducer. Unfortunately, as a result of the logic

required for a shifter, the proposed design with LZD is quite a bit larger than the original design but this once again shows that the proposed design is able to increase its performance at the expense of area.

With the comparison of the cycles required for each reducer, it is evident that Barrett reducer still has a clear advantage over the proposed design. This is due to the Barrett reducer being able to calculate the modulus with little required arithmetic other than the multiplication which, as stated before, has a long critical path. Although the Barrett reducer is clearly faster overall, the significant power savings along with the architecture with a LZD has stronger benefits in the long run. Although the architecture with a LZD consumes more energy, it has a significant reduction in cycle counts as shown in Table 3 comparing cycle counts using NIST P-384 curve data [35].

Table 3: Average Cycle Comparison of the Mod without Mod vs. Barrett Reduction Algorithm using NIST Vectors

Vectors	Barrett Reducer	Mod without Mod	Mod without Mod w/LZD
ECP 384	6 cycles	390 cycles	104 cycles

In summary, the architectures implemented for this work based on previous work [33] demonstrate significant performance for use within crypto-based hardware. The designs are integrated into architectures utilized at the AFRL and implemented in CMOS 45nm and 14nm technologies. Optimization is also done on the software within the AFRL secure hardware to optimize the hardware and implement it in silicon.

3.3 Testing Integration for IEEE 754 and Crypto-Based Hardware

Operations that involve complex specifications, such as the IEEE 754, are hard to validate well. This is because there are lots of nuances related to the operation as well as IEEE 754 vast array of exceptions that can occur [16]. The IEEE-754 standard for binary floating-point numbers specifies invalid and valid operations. In addition, the standard specifies the capabilities of at least two different kinds of a Not-A-Number (NaN). The two different kinds that are required for correct compliance to the IEEE 754 are signaling and quiet NaNs. Signaling NaNs are used for uninitialized variables and arithmetic-like enhancements that are not part of the standard. Addition or subtraction with one or two quiet NaNs for input operands will signal no exception but deliver a quiet NaN result. In addition, any operation on a signaling NaN will issue an Invalid operation exception and deliver a quiet NaN result as specified in Section 6 of the IEEE-754 standard [16].

This work involves creating testing integration for 32-bit and 64-bit floating-point numbers for various IEEE 754 operations, such as addition, subtraction, comparisons, division, conversion and multiplication.

To simplify the testing, a heavily modified version of the floating-point validation suite written by John Hauser at the University of California at Berkeley [36] to help ensure compatibility with the IEEE-754 standard. The suites were modified in C and integrated into the Mentor Graphics/Siemens ModelSim software. Testbenchs were written in VHDL and Verilog to read in values and test them against the hardware designs for this work.

In order to make sure the testbenches work effectively, they are validated against a clock. This clock is utilized to stabilize the inputs and allow the vectors to be read from a file. The vectors come from the TestFloat suite and then get tested against each design. After testing, an output file is created to indicate what the test results are and what was an error and the vectors that passed the test. The basic suite processes a significant number of vectors but processes them intelligently and efficiently to allow the designs for the AFRL to be integrated into a secure processor design.

The basic structure is shown in Figure 2 where the positive clock is utilized for reading in the TestFloat or NIST vector suite. Then a repeat statement is utilized to delay the clock to wait for the functional unit to complete its operation. The number of clock cycles is dependent on the unit being designed. For example, some designs take only 1 cycle to complete, whereas others take multiple cycles. For example, in Figure 2 the testbench waits 15 clock cycles. After the operation completes, the testbench checks the results versus the known result and outputs the value to an output file for validation. Scripts are designed to run this automatically. Each testing suite takes approximately 60 minutes on average to run through all the tests for a given design and operation (e.g., IEEE 754 addition).

```
always @(posedge clk)
  begin
     if (~reset)
       begin
          #0; {op1, op2, yexpected, flags_expected} = testvectors[vectornum];
          #50 start = 1'b1;
          repeat (2)
            @(posedge clk);
          // deassert start after 2 cycles
          start = 1'b0;
          repeat (13)
            @(posedge clk);
          $fdisplay(desc3, "%h_%h_%h_%b_%b | %h_%b", op1, op2, AS_Result, Flags, Denorm, yexpected, (AS_Result==yexpected));
          vectornum = vectornum + 1;
       end // if (~reset)
     $display("%d vectors processed", vectornum);
  end // always @ (posedge clk)
```

Figure 2: Testing Validation Suite Specifically for Verilog

Each operation is tested for the secure processor designs found within this work. All designs passed and were validated by the test suites from NIST and TestFloat. Having a set design of test suites not only saved time but allowed more time to be given into the designs found later in this work.

3.4 Integer and IEEE 754 Floating-Point High-Performance and Low-Power Implementations

As mentioned previously, IP designs can cause significant disruptions in testing complex systems. These systems get extremely complicated when you have multiple units that rely on each other. That is, one system that utilizes IEEE 754 arithmetic may be reliant on something that includes security, therefore, making sure all the systems work together efficiently is important. With dedicated IP from companies like Cadence Design Systems® or Synopsys®, a user has no access to the underlying elements within a design causing EDA tools to possibly not optimize across a complete architecture well. For this reason, this work involves designing high-performance and energy-efficient designs that help the AFRL optimize computation but also allow the complete architecture to optimize across the whole design.

More importantly, creating an optimal unit that allows multiple units to simulate them across a design. Most IP vendors provide two different types of simulation models for a design. They typically include a behavioral and element to instantiate a design. The behavior-based design is given to be compliant with the operation of the IP that is purchased. Although this behavior-based model is accurate, running more detailed analysis along with other designs can cause difficulty with assessing the design's overall benefit or disadvantage [37]. However, when an institution has access to complete models along with a Hardware Descriptive Language design, it makes optimizing across a design easier and allows simulation to accurately model what is happening in the hardware.

This work involves several complete designs implemented by the authors of this report. The designs utilize the experience of the author to create optimal arithmetic designs for integer and IEEE 754 arithmetic that are optimized for high-performance but also take into account energy performance. The following designs are created for this work and are documented in the following subsections

- 32-bit and 64-bit Integer Divider
- Integer and IEEE 754 Floating-Point Comparator
- IEEE 754 Denormalized and Normalized Floating-Point Adder/Subtracter
- IEEE 754 Divider and Square-Root Unit

3.4.1 32-bit and 64-bit Integer Divider

Although division by recurrence [38] is an important algorithm for division it has had little use over the last couple of years. This may have something to do with issues that arose during the floating-point divide (FDIV) division bug during the early 1990s [39]. Despite utilizing architectures that employ multipliers to avoid similar FDIV verification issues and the ability to combine with existing multiplier units, division by recurrence has certain elements that may have advantages for architectures that require low-area overhead and low-energy requirements as well as the demand in multi-core processors employing multiple functional units.

Division by recurrence [40], [41] also has other unique advantages in that it can easily produce the remainder and can be easily augmented for higher radices to allow more bits to be retired every

iteration. Consequently, these architectures can lend themselves to digit-serial architectures that may be more suited towards applications such as signal processing.

Although division by recurrence also has advantages in that it can be easily implemented within VLSI designs, logic designs can get complicated with the implementation of Signed-Digit (SD) arithmetic [40], [42]. This section presents a radix 4 signed digit divider using recurrence that exhibits low-power tendencies. Results from this research indicate that division by recurrence can be broken down into clear steps that make its implementation simple and concise. Second, low-area overheads can potentially make this unit a clear winner for architectures that require division without the need for fast cycle times. Finally, these architectures clearly outperform high-performance multiplicative-division architectures in certain areas by showing approximately a 75% reduction in overall power dissipation as well as a much smaller area overhead.

Division by recurrence utilizes a simplified equation to compute the quotient every iteration. It is assumed that the input operands are the dividend (N) and the divisor (D). This equation is radix-based (r) and iterates every iteration to produce the quotient:

$$w[n+1] = r \cdot w[n] - q_{n+1} \cdot D \tag{2}$$

where $w[0] = N$. The process of selecting the quotient is called the Quotient Selection Table (QST) process. Adders with redundant digit sets are utilized to avoid carry propagation and optimize on speed [40]. The simplest method to avoid carry propagation is to utilize redundant adders, such as the Signed-Digit Adder. Although redundant adders help alleviate carry propagation, careful alignment of the carry along with the sum is necessary to produce a final sum. Signed Digit Adders also introduce an increase in area compared to methods using carry-propagate addition since the number of bits is typically doubled for a given implementation. Critically, redundant adders can also introduce errors that can be detrimental to an algorithm if not managed properly.

Integer division is somewhat difficult in that the selection function needs to have the input operands modified in order to get their values in range. Therefore, the divisor is shifted (m bits) accordingly via a leading-one detector (LOD) [34]. However, since the computation involves a higher radix version of division, the complexity is increased for integer division. That is, a correct remainder is required for the bit of the quotient so that it is aligned to the radix-r boundary. This requires extra or guard bits within the intermediate computation to allow for the computation to take place.

The baseline structure for the design is shown in Figure 3 for 32-bit division. The design is expanded to handle both 32-bit and 64-bit division based on an input selector. Registers are shown in Figure 3 as filled-in rectangles. The overall structure of this divider is built to perform a maximum of 19 iterations. The number of iterations required is calculated and passed to the finite state machine (FSM) for signal control. The additional bits for handling overflow and aligning within a radix-4 boundary amounts to 36 bits internally. A small additional carry propagate adder (CPA) before the QST, as seen in Figure 3, is utilized to combine the carry/save portions of the recurrence before the QST.

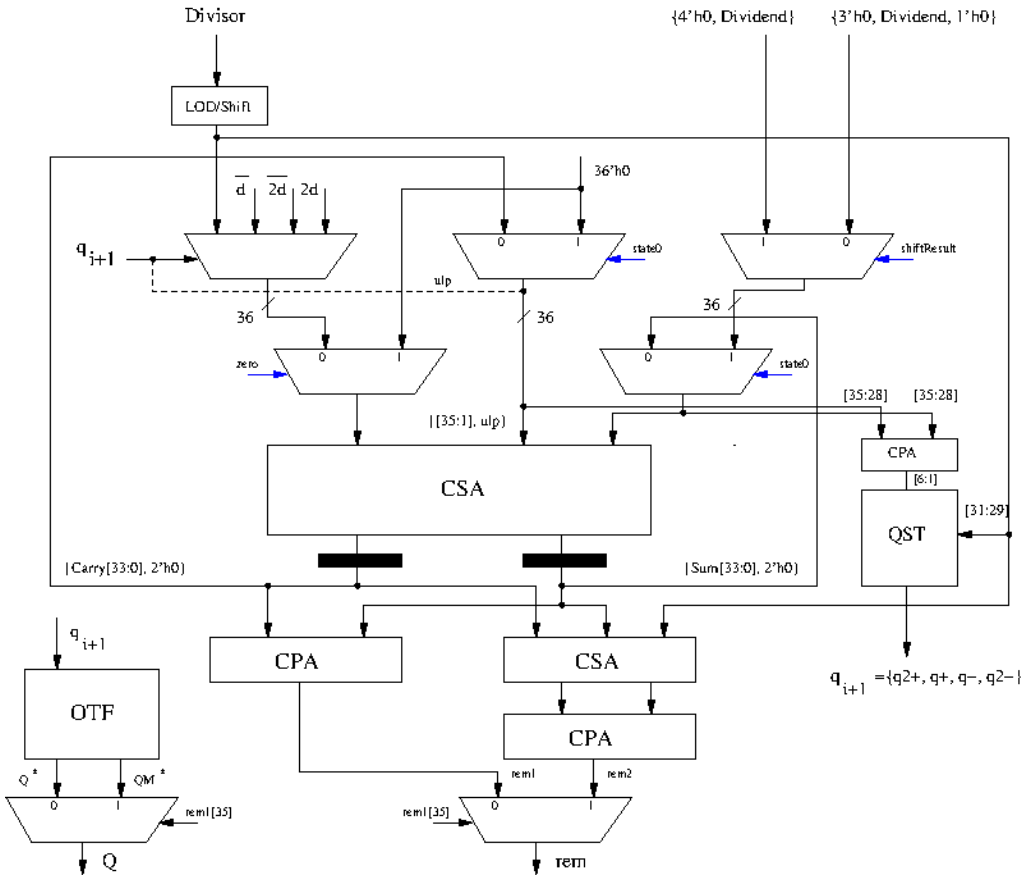

Figure 3: 32-bit Integer Divider

The on-the-fly (OTF) unit translates each quotient digit during a recurrence into its conventional representation using a technique called on-the-fly (OTF) conversion [43]. On-the-fly conversion is an extremely efficient method of converting an item from redundant notation into its conventional representation since recurrence methods for division, and even square root, are online algorithms. The quotient-digit selection uses an estimation of the remainder to obtain the next quotient digit. As mentioned previously, the quotient-digit selection is independent from the value of the divisor and to exploit the target standard-cell library in our results, the function is implemented directly from the carry-save form without assimilation. Since the QST is significantly smaller, this methodology helps the synthesis form an easy amount of combinational logic to generate the quotient correctly despite a large number of inputs.

Integer division requires a remainder to be produced, which is one advantage of using division by recurrence compared to other division methods that employ multiplication. However, since division by recurrence is implemented using a non-restoring division algorithm, the remainder needs to be modified appropriately. Consequently, the remainder needs to be shifted accordingly after N cycles. The remainder is also utilized to make sure the quotient is properly adjusted due to the partial remainder possibly being negative, as shown in Figure 3.

Since division by recurrence occupies significantly less area than other methods, such as Goldschmidt's division, it will consume a significantly smaller amount of energy. Results using a 32nm ARM-based standard-cell library indicate a 97% reduction in energy for 32-bit input operands over methods that employ multipliers (e.g., Newton-Raphson based division). That is, for 32-bit input operands only 0.717 mW is consumed over a multiplier-based divider that consumes 24.139 mW. This reduction in power is achieved due to the approximate 90% reduction in area for the 32-bit divider in this work

3.4.2 32-bit and 64-bit Integer and IEEE 754 Floating-Point Comparator

An important datapath element for any general-purpose architecture is the comparator [14]. It is also an essential device for application-specific and signal-processing architectures [44]. In order for processors to maintain high throughput with fast clock rates and low area, it is imperative that such devices be small, yet fast. Consequently, the design of efficient comparators is an important research topic.

This work integrates a high-performance 64-bit comparator that performs floating-point comparisons on single precision (32-bit) and double precision (64-bit) IEEE 754 numbers, as well as 32-bit and 64-bit two's complement numbers [17]. Although the comparator presented in this work is not pipelined, it can easily be pipelined or designed with skew-tolerant design techniques [45].

Floating-point comparisons are usually implemented using one of two methods, both of which are addressed by the IEEE 754 standard [16]. With the first method, the comparator indicates one of four mutually exclusive relations between the two input operands: Greater Than (GT), Less Than (LT), Equal (EQ) and Unordered. The last case, Unordered, arises when at least one operand is a Not-a-Number (i.e., NaN). For this case, an Invalid Operation exception is also asserted. The result of the comparison is often stored using two bits of a floating-point status and control register (FSCR), called the floating-point condition codes (FCC) [46]. With the second method, the comparator returns a True-False condition code based on a specific comparison condition (e.g., $A > B$). With both methods, comparisons are exact and never overflow nor underflow. Comparisons also ignore the sign of zero, such that positive zero is equal to negative zero. Infinite values are considered to have larger magnitudes than all other floating-point numbers.

The comparator design presented for this work performs a magnitude (unsigned) comparison based on a technique previously used for operand normalization [47]. In the first stage of the magnitude comparator, 2-bit pairs from the first operand, A, are compared in parallel with the corresponding 2-bit pairs from the second operand, B, to determine if the two bits from A are greater than or less than the two bits from B. This hierarchical-based comparator has the advantage in that it can do comparisons for both two's complement and IEEE 754 arithmetic but also is more energy efficient in that it uses less resources than traditional adder-based comparators [17].

3.4.3 IEEE 754 Floating-Point Normalized and Denormalized Adder/Subtracter

The IEEE 754 standard allows for, in non-denormalized cases, exponent values of 1 to 2046, along with a number of exception cases including NaN's (Not-a-Number) and infinity. This allows for non-denormalized values to range from 2.225×10^{-308} to 1.779×10^{308}. In the case for denormalized values, this extends the lower range to 4.941×10^{-325} [15], [48]. This is due to the exponent now able to have an effective representation of -1022. Regardless of the details of implementation, floating-point addition requires the input operands to go through a number of processes. Those processes are as follows, both input operand exponents are compared, the significands are aligned and added, the result of said addition is normalized, and finally the result is rounded [40]. Additional processes in the datapath are typically used either for format conversion or are application specific. For example, in this design, a mantissa comparison operation is required to be performed in parallel with the post-sum normalization process in order to generate denormalized exponent rounding logic.

The architecture utilized for the design in this research utilizes an IEEE 754 compliant floating-point adder/subtracter implementation that attempts to maximize the utility of the adder and still maintain relatively high performance. This design supports not only normalized IEEE 754 values but has full support for denormalized IEEE 754 values. Any combination of denormalized and normalized values can be used as inputs. It also has full support for binary 16, binary 32, and binary 64 IEEE 754 2008 values, and it offers the ability to convert between them. It is also able to convert binary 16, binary 32, and binary 64 two's complement integer values to the IEEE 754 format.

This specialized unit is important for this research as its intended application is for high-performance computing utilized by the AFRL. And denormalized IEEE 754 floating-point numbers fill the gap between the smallest normal and zero by allowing numbers with reduced precision to exist. As the value decreases so does the precision in what is called gradual underflow [49], [50]. In several applications, especially those dealing with application-specific scientific uses, gradual underflow can preserve expected program behavior, whereas truncation may be erratic. There is no situation that better demonstrates the value of denormals than the difference of two small numbers A and B. For example, if A and B are nearly equal but different, A-B may result in a number smaller than the minimum normal. Without gradual underflow A-B may yield a 0 result which possibly confuses applications that utilize comparisons. Consequently, there is a need for hardware which handles both normalized and denormalized IEEE 754 floating-point numbers.

The primary adder-subtractor structure for this design utilizes a carry-propagate (CPA) parallelized 64-bit prefix adders to simultaneously compute both possible results for the mantissas so that, regardless of the relative difference between the input mantissas, no other steps are necessary. The sign of the result from the prefix structure's adder is also computed in combination with results from the input handling section to create the corrected sign for the result. The internal structure of the prefix adders is also manually defined to keep dynamic power levels low during synthesis.

To prepare the sum of the adder-subtractor for rounding, a two-stage structure is used. The first stage uses another leading-zero detector, similar to the one used earlier in this work [34] to feed the number of leading zeros into the second stage, a left barrel shifter. That is, the number of leading zeroes is determined from looking at the result of the adder-subtractor. The result from the adder-subtractor is then left shifted by the number of leading zeroes. This produces a result within

the required fixed domain values for IEEE 754, and the shift amount required to reach this domain is sent to the rounder to adjust the exponent.

The efficiency of the parallelization of the CPA enables the design to work efficiently for the high-performance designs within the AFRL. The power consumption uses 62.4% of the power compared to the adder generated from a Synopsys® IP reference design. This is due to optimizations of the design as well as minimization of the logic through a separate adder and subtracter structure. Since most of the performance from this design comes from the extent of its parallelization, there are next to no scenarios where values are propagated through logic more than once. This minimizes the frequency at which voltage swings occur, reducing switching and internal power requirements. Parallelization also allows for exception and rounding logic for the design, resulting in similar delay and better power results. However, the additional precision and denormalized value support does require a significant amount of extra hardware to implement. As a result, compared to a design that only has IEEE 754 double-precision normalized arithmetic, this results in a 45% increase in area.

The key to this implementation is paralleling all of the adder and subtractor structures possible in the design, which allows synthesis to better optimize the critical path through this architecture [23]. Then, the corrected sign is utilized to help adjust the operation accordingly. Similar designs use the same idea within a three-operand addition unit [51]. The unit is designed completely in VHDL and thoroughly tested utilizing the TestFloat validation suite mentioned earlier in this report. Additional random denormalized vectors via a Java program were also generated to give completeness and passed completely.

3.4.4 IEEE 754 Floating-Point Divide and Square-Root Unit

One of the most challenging operations to compute is IEEE 754 division and square root. Although division occurs infrequently in most scientific computations, neglecting or using software versions can significantly degrade the overall performance of an architecture [52]. There have been several significant implementations for IEEE 754 division and square root, mostly using a multiplier [53]. Unfortunately, many of these implementations are proprietary or have limited information on their overall design.

One of the more challenging elements to process within division and square root while using a multiplier is for IEEE 754 rounding. This is because it is difficult to address what the remainder is. Researchers have suggested using a subsequent operation for multiplicative-based division and square that computes the remainder and then using this operation to adjust the rounding properly [54]. Multiplicative-based routines typically employ converging algorithms that utilize bracketing methods to make sure an approximation to the reciprocal occurs [55]. Afterwards, a multiplication is performed that completes the division or square root operation.

The challenging element in this research, although somewhat documented in [53], is to design a unit that can handle both IEEE 754 arithmetic for single and double precision numbers. This is achieved by utilizing a variant of the Newton-Raphson method but adding a constant term that properly rounds the final answer to its correct rounded result. Based on a variant of [54], the rounding function assumes that a biased intermediate result Q that has been computed with N+1 bits, which is known to have an error of $(-0.5, +0.5)$ unit-in-the-last position (ulp) with respect

to N bits. The extra bit, or guard bit, is used along with the sign of the remainder, a bit stating whether the remainder is exactly zero, and a value to choose from three possible results, Q, Q+1, and Q-1 (i.e., denoted as Q, QP=Q+1, QN=Q-1). As shown in [53], this makes the selection of the correct result trivial as shown by Table 4.

In Table 4, simple combinational logic is utilized to choose the correct IEEE 754 rounding mode: round-to-nearest even (RNE), round-to-positive infinity (RP), round-to-negative infinity (RM), and round-to-zero (RZ) or truncation [16]. Both RP and RM are difficult in that they are contingent on the sign of the result as shown in the heading (i.e., +/-). One of the interesting elements of this table is that the rows that indicate the remainder equal to zero can never occur. This can be proven by assuming the quotient is equal to the numerator divided by the denominator. If both the numerator and denominator are the same size (e.g., N bits), the result or the quotient can never occur (i.e., it can never be equal to N+1 bits). Therefore, the halfway case or the blue-colored rows in Table 4 are never considered and the selection logic becomes a simple combinational choice of the correct result.

Table 4: Action Table for Rounding IEEE 754 Numbers using Multiplicative-Based Divide and Square Root

Guard Bit	Remainder	RNE	RP (+/-)	RM (+/-)	RZ
0	**=0**	**Q**	**Q**	**Q**	**Q**
0	-	Q	Q/QN	QN/Q	Q
0	+	Q	QP/Q	Q/QP	Q
1	**=0**	**RNE**	**QP/Q**	**Q/QP**	**Q**
1	-	Q	QP/Q	Q/QP	Q
1	+	QP	QP/Q	Q/QP	Q

The designs for this work are efficiently designed to operate at 1.5 GHz (Giga Hertz) in 14nm Global Foundries CMOS using ARM-based standard-cells. The designs are created to allow for either single-precision or double-precision input operands and allows for all five IEEE 754-2008 rounding modes as well as five IEEE 754 status flags [15]. The design is coded up in VHDL and Verilog and verified using the testing suite mentioned previously.

4.0 RESULTS AND DISCUSSION

The design flow is created in such a way that environmental variables are utilized to enable it to be installed anywhere. For example, all the libraries refer to an environmental variable. The Global Foundries (GF) 45nm, 32nm, and 14nm SoC framework utilizes standard-cells and memories from ARM®. All implementations utilize MTCMOS-based standard-cell technologies. Scripts are designed to be integrated with the design flow, so that any design can be easily created from an HDL design into a mask layout. All standard cell designs and memory elements are created using scripts that either generate a placed and route design or initiate a language within the Cadence Design System® or Synopsys® EDA tools.

Power dissipation is an important element within any computer architecture; however, the computation of power can be complicated by the level of extraction that occurs for a given design. Consequently, the SoC framework and design flow presented here has several different power level estimation tools that can either estimate the power during synthesis with no parasitic extraction of the wires (e.g., through Synopsys® PrimeTime™) or computed more accurately using an extracted standard parasitic exchange format (SPEF) file. Scripts are created that allow all designs to be tested effortlessly for timing, area, and power parameters. More importantly, all the scripts can be modified, so that power, delay, or area can be targeted as an optimized constraint for a given computer architecture. All designs were created and designed effortlessly through the use of scripts designed to interface the HDL definitions.

The designs presented in this research are designed to allow a user to focus on a particular item for optimization, such as Signal Integrity, and play with design parameters to enhance its understanding and potentially go between different EDA tools. The designs are all self-contained with the directory and can be remade through their Makefiles or by typing, “make all”. Since each design is composed of smaller implementation design architectures, each design runs with minimum run time and system resources.

All designs are contained on the OSU/AFRL research server. It is also anticipated that many Air Force Research Laboratory personnel can utilize the research presented in this report for further study and other projects. All of the designs are also integrated into designs that were successfully fabricated out during the timeframe of this work.

5.0 CONCLUSIONS

This work demonstrates the research that was conducted for the Air Force Research Laboratory. With the combination of architecture and circuit-based enhancements and a common set of software design flows, allowing a cohesive SoC framework and high-performance and low-power units that produces designs that are several orders of magnitude better than previously implemented architectures. More importantly, the tools have been created that allow them to create multiple design seamlessly as well as create an agile environment for the deployment of VLSI computer architectures in silicon.

6.0 REFERENCES

[1] S. G. Narendra, "Challenges and design choices in nanoscale CMOS," *J. Emerg. Technol. Comput. Syst.*, vol. 1, no. 1, pp. 7–49, Mar. 2005.

[2] K. Zhang, "Challenges and opportunities in nano-scale VLSI design," in *2005 IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT).*, Apr. 2005, pp. 6–7.

[3] R. Sivakumar, Department of ECE, RMK Engineering College, India., and D. Jothi, "Recent Trends in Low Power VLSI Design," *Int. j. comput. electr. eng.*, vol. 6, no. 6, pp. 509–523, 2014.

[4] N. Weste and D. Harris, "Cmos Vlsi Design: A Circuits And Systems," *Perspective,. Reading, MA, USA, Addison Wesley*, 2010.

[5] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Power Limitations and Dark Silicon Challenge the Future of Multicore," *ACM Trans. Comput. Syst.*, vol. 30, no. 3, pp. 1–27, Aug. 2012.

[6] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," in *Design Automation Conference*, Jun. 2010, pp. 555–560.

[7] S. Ataei and J. E. Stine, "A 64 kB Approximate SRAM Architecture for Low-Power Video Applications," *IEEE Embedded Sys. Lett.*, vol. 10, no. 1, pp. 10–13, Mar. 2018.

[8] V. Kursun and E. G. Friedman, *Multi-voltage CMOS Circuit Design: Kursun/Multi-Voltage CMOS Circuit Design*. Chichester, England: John Wiley & Sons, 2006.

[9] J. E. Stine and J. Grad, "Low Power and High Speed Addition Strategies for VLSI," in *2006 8th International Conference on Solid-State and Integrated Circuit Technology Proceedings*, Oct. 2006, pp. 1610–1613.

[10] A. Vassighi, O. Semenov, M. Sachdev, and A. Keshavarzi, "Effect of static power dissipation in burn-in environment on yield of VLSI," in *17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings.*, Nov. 2002, pp. 12–19.

[11] N. B. Romli, K. N. Minhad, M. B. I. Reaz, and M. S. Amin, "An overview of power dissipation and control techniques in cmos technology," *J. Eng. Sci. Technol*, vol. 10, no. 3, pp. 364–382, 2015.

[12] M. Alioto, E. Consoli, and G. Palumbo, "Analysis and Comparison in the Energy-Delay-Area Domain," *Flip-Flop Design in Nanometer CMOS*. pp. 119–173, 2015, doi: 10.1007/978-3-319-01997-0_5.

[13]C. Teuscher, *Alan Turing: Life and Legacy of a Great Thinker*. Springer Science & Business Media, 2004.

[14]D. A. Patterson and J. L. Hennessy, *Computer Organization and Design ARM Edition: The Hardware Software Interface*. Morgan Kaufmann, 2016.

[15]J.-M. Muller *et al.*, *Handbook of Floating-Point Arithmetic*. Birkhäuser, Cham, 2018.

[16]American National Standards Institute, *IEEE Standard for Binary Floating-point Arithmetic*. IEEE, 1985.

[17]J. E. Stine and M. J. Schulte, "A combined two's complement and floating-point comparator," in *2005 IEEE International Symposium on Circuits and Systems*, May 2005, pp. 89–92 Vol. 1.

[18]E. Schwarz, "Revisions to the IEEE 754 standard for floating-point arithmetic," *16th IEEE Symposium on Computer Arithmetic, 2003. Proceedings.* doi: 10.1109/arith.2003.1207667.

[19]J. Dean, D. Patterson, and C. Young, "A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21–29, Mar. 2018.

[20]S. L. Harris and D. Harris, *Digital Design and Computer Architecture: ARM Edition*. Morgan Kaufmann, 2015.

[21]S. Devadas and S. Malik, "A survey of optimization techniques targeting low power VLSI circuits," in *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, San Francisco, California, USA, Jan. 1995, pp. 242–247, Accessed: May 03, 2021. [Online].

[22]J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, May 2013, pp. 1–6.

[23]B. Mathis and J. E. Stine, "A Well-Equipped Implementation: Normal/Denormalized Half/Single/Double Precision IEEE 754 Floating-Point Adder/Subtracter," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Jul. 2019, vol. 2160–052X, pp. 227–234.

[24]D. Burger and J. R. Goodman, "Billion-transistor architectures," *IEEE Ann. Hist. Comput.*, vol. 30, no. 09, pp. 46–49, 1997.

[25]J. Doweck *et al.*, "Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake," *IEEE Micro*, vol. 37, no. 2, pp. 52–62, Mar. 2017.

[26]G. Qu and M. Potkonjak, *Intellectual Property Protection in VLSI Designs: Theory and Practice*. Springer Science & Business Media, 2007.

[27]P. Kurup and T. Abbasi, "Design Re-Use using DesignWare," *Logic Synthesis Using Synopsys®*. pp. 263–278, 1995, doi: 10.1007/978-1-4757-2370-0_8.

[28]P. Kornerup, “High-radix modular multiplication for cryptosystems,” in *Proceedings of IEEE 11th Symposium on Computer Arithmetic*, Jun. 1993, pp. 277–283.

[29]A. Carter *et al.*, “Comparison of parallelized radix-2 and radix-4 scalable Montgomery multipliers,” in *2013 Asilomar Conference on Signals, Systems and Computers*, Nov. 2013, pp. 1144–1148.

[30]F. Rodriguez-Henriquez, N. A. Saqib, A. D. Pérez, and C. K. Koc, *Cryptographic Algorithms on Reconfigurable Hardware*. Springer Science & Business Media, 2007.

[31]J.-P. Deschamps, *Hardware Implementation of Finite-Field Arithmetic*. McGraw Hill Professional, 2009.

[32]M. A. Will and R. K. L. Ko, “Computing mod without mod.” https://eprint.iacr.org/2014/755.pdf (accessed May 04, 2021).

[33]R. Swann and J. E. Stine, “An Improved Hardware Architecture for modulo without Multiplication,” in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug. 2020, pp. 635–638.

[34]V. G. Oklobdzija, “An algorithmic and novel design of a leading zero detector circuit: comparison with logic synthesis,” *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 2, no. 1, pp. 124–128, Mar. 1994.

[35]National Institute National Institute of Standards and Technology, *FIPS PUB 186-4 Digital Signature Standard (DSS): Subcategory: Cryptography*. CreateSpace Independent Publishing Platform, 2013.

[36]J. Hauser, “The SoftFloat and TestFloat Validation Suite for Binary Floating-Point Arithmetic,” *University of California, Berkeley, Tech. Rep*, 1999.

[37]V. Hahanov, I. Hahanova, N. C. Umerah, and T. Yves, “Testing and verification of HDL-models for SoC components,” *2010 East-West Design & Test Symposium (EWDTS)*. 2010, doi: 10.1109/ewdts.2010.5742112.

[38]M. D. Ercegovac and T. Lang, “Simple radix-4 division with operands scaling,” *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1204–1208, Sep. 1990.

[39]D. Price, “Pentium FDIV flaw-lessons learned,” *IEEE Micro*, vol. 15, no. 2, pp. 86–88, Apr. 1995.

[40]M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Elsevier, 2004.

[41]J. E. Stine, *Digital Computer Arithmetic Datapath Design Using Verilog HDL*. Springer Science & Business Media, 2004.

[42]A. Avizienis, “Signed-Digit Numbe Representations for Fast Parallel Arithmetic,” *IRE Transactions on Electronic Computers*, vol. EC-10, no. 3, pp. 389–400, Sep. 1961.

[43]Ercegovac and Lang, “On-the-Fly Conversion of Redundant into Conventional Representations,” *IEEE Trans. Comput.*, vol. 36, pp. 895–897, Jul. 1987.

[44]J. Eyre and J. Bier, “DSP processors hit the mainstream,” *Computer* , vol. 31, no. 8, pp. 51–59, Aug. 1998.

[45]D. Harris and M. A. Horowitz, “Skew-tolerant domino circuits,” *IEEE J. Solid-State Circuits*, vol. 32, no. 11, pp. 1702–1711, Nov. 1997.

[46]S. I. Inc and D. L. Weaver, *The SPARC architecture manual*. Prentice-Hall, 1994.

[47]E. Antelo, M. Boo, J. D. Bruguera, and E. L. Zapata, “A novel design of a two operand normalization circuit,” *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 6, no. 1, pp. 173–176, Mar. 1998.

[48]M. L. Overton, *Numerical computing with IEEE floating point arithmetic*. SIAM, 2001.

[49]J. T. Coonen, “Underflow and the denormalized numbers,” *Computer* , 1981, [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.1839&rep=rep1&type=pdf.

[50]W. Kahan, “Lecture notes on the status of IEEE standard 754 for binary floating-point arithmetic,” *http://http. cs. berkeley. edu/\ wkahan/ieee754status/ieee. ps*, 1996, [Online]. Available: https://ci.nii.ac.jp/naid/10011331711/.

[51]J. Sohn and E. E. Swartzlander, “A Fused Floating-Point Three-Term Adder,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 61, no. 10, pp. 2842–2850, Oct. 2014.

[52]S. F. Oberman and M. J. Flynn, “Design issues in division and other floating-point operations,” *IEEE Trans. Comput.*, vol. 46, no. 2, pp. 154–161, Feb. 1997.

[53]S. F. Oberman, “Floating point division and square root algorithms and implementation in the AMD-K7/sup TM/ microprocessor,” in *Proceedings 14th IEEE Symposium on Computer Arithmetic (Cat. No.99CB36336)*, Apr. 1999, pp. 106–115.

[54]E. M. Schwarz, “Rounding for quadratically converging algorithms for division and square root,” in *Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, Oct. 1995, vol. 1, pp. 600–603 vol.1.

[55]S. C. Chapra and Others, *Applied numerical methods with MATLAB for engineers and scientists*. McGraw-Hill Higher Education, 2008.

7.0 LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

AFRL	Air Force Research Laboratory
ARM	Advanced RISC Machines
CDS	Cadence Design Systems
CMOS	Complementary Metal-Oxide Semiconductor
CPA	Carry Propagate Adder
DAR	Double Add Reduce
DC	Design Compiler
EDA	Electronic Design Automation
EQ	Equal
FCC	Floating-point Condition Codes
FDIV	Floating Point Divide
FSCR	Floating-point Status and Control Register
FSM	Finite State Machine
GF	Global Foundries
GHz	Giga Hertz
GT	Greater Than
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
LOD	Leading One Detector
LT	Less Than
LZD	Leading Zero Detector
MTCMOS	Multi-Threshold CMOS
NaN	Not A Number
NIST	National Institute of Standards and Technology
NMOS	N-type Metal-Oxide Semiconductor
OSU	Oklahoma State University
OTF	On-The-Fly
PMOS	P-type Metal-Oxide Semiconductor
QST	Quotient Selection Table

RM	Round to negative infinity
RNE	Round to Nearest Even
RP	Round to Positive infinity
RZ	Round to Zero
SD	Signed Digit
SoC	System-on-Chip
SPEF	Standard Parasitic Exchange Format
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VLSI	Very Large Scale Integration