An Energy Efficient Datapath for Asymmetric Cryptography

Andrew D. Targhetta and Paul V. Gratz
Department of Electrical and Computer Engineering
Texas A&M University

***Abstract*—Public key encryption on resource constrained devices can be a burden in terms of latencies per operation but also in terms of energy consumed. For certain applications, such as WSN and RFID tags, the energy budget is quite small. Understanding the energy cost of each public key operation is important. This paper explores the energy consumption of public key cryptography by presenting a parameterizable microarchitecture for accelerating public key operations and demonstrating how the datapath width and key size effects the energy consumption per Montgomery multiplication, a key component of public key encryption algorithms. We find that for $O(n^2)$ algorithms, like Montgomery multiplication, a larger datapath provides optimal energy efficiency. Moreover, our microarchitecture is capable of reducing the energy consumption of Montgomery multiplication by a factor of 50 while providing a speed up of 10x for a key size of 192-bits over the traditional software implementation on a low-power embedded processor.**

I. Introduction

Asymmetric cryptography, also know as Public key encryption, has become an essential component in secure communications. Unlike its symmetric counterpart, asymmetric cryptography requires separate keys for the encryption and decryption operations. Asymmetric cryptography has been used to solve a variety of security challenges, ranging from session key establishment for secure communications to digital signatures for message authenticity and non-repudiation [5]. While symmetric cryptography is based on data shifts and permutations, asymmetric cryptography is based on mathematically hard problems. As a result, the computational requirements for asymmetric cryptography are far greater than that of symmetric cryptography [13].

To enable asymmetric cryptographic capabilities on resource constrained devices, acceleration hardware is often employed. Example applications include but are not limited to smart cards, Wireless Sensor Networks (WSN), and Radio Frequency Identification (RFID) tags [1], [6], [16]. Wander et al. found, in the WSN domain, fairly weak asymmetric cryptography (160-bit ECC equivalent to 1024-bit RSA) consumes approximately 72% of the energy alloted for communication handshaking. Moreover, they assume that only 5% to 10% of a WSN's energy budget is available for handshakes [23]. For RFID tags, it is difficult to quantify the energy budget for encryption; however, since most RFIDs are passive energy harvesters, it is significantly less than that of a WSN node.

Much of the research and development in the area of asymmetric cryptographic hardware acceleration has focused on minimizing both the time per cryptographic operation and the amount of hardware resources consumed, while other efforts have attempted to address reconfigurability. However, a limited amount of work has been put into understanding the power and energy impact of asymmetric cryptographic hardware acceleration. Recognizing the trades-offs between energy consumption, logic utilization, and performance in cryptographic hardware designs is important in any sort of energy constrained system. Therefore, the purpose of this paper is two fold. First, a parameterizable microarchitecture for accelerating asymmetric cryptographic operations will be introduced. Second, various configurations of this novel microarchitecture will be evaluated in terms of energy consumption. From the results presented in this paper, conclusions will be made as to which datapath width yields the most energy efficient design. Likewise, the effect of security requirements on energy consumption will be provided. To demonstrate the significance of this work, the energy consumption of our hardware will be compared to that of an ultra low power processor, an ARM Cortex-M3, performing the same operations.

II. Background

A. Underlying Mathematics

Public key encryption is based on a one-way function, a mathematical function which has a computationally feasible forward operation and a computationally infeasible reverse operation. RSA is a public key encryption scheme which uses modular exponentiation as a one-way function [17]. To perform modular exponentiation, a series of large integer multiplications are computed such that the modulo operator is used to reduce intermediate results. The reverse operation, referred to as the Discrete Logarithm Problem (DLP), is considered intractable as the size of the modulus increases [13]. It should be noted that modular multiplication is part of a larger class of finite field operations, and increasing the number of bits used to represent the modulus increases the size of the finite field.

The successor to RSA is Elliptic Curve Cryptography (ECC), which utilizes scalar point multiplication over elliptic curves [11], [14]. Scalar point multiplication involves repeated addition and doubling of points on an elliptic curve defined over a finite field and is analogous to modular exponentiation. Elliptic curve arithmetic requires modular addition, subtraction, and inversion, along with modular multiplication. Moreover, techniques involving three dimensional coordinate systems are typically employed, such that modular multiplication dominates the algorithm's execution time [3].

Using algorithm complexity approximations, we analytically calculated the percentage of time spent in modular multiplication for an ECC operation to be between 76.4% to 98.1% . Our first order approximation with various keys sizes and datapath configurations assumes the use affine-Jacobian mixed coordinates to reduce the required number of inversions to one and Fermat's Little theorem for the inversion computation [3],[4]. It should be noted, however, that the hardware described in this paper is capable of performing modular addition and subtraction along with modular multiplication.

As with modular exponentiation, the reverse operation, known as the Elliptic Curve Discrete Logarithm Problem (ECDLP), is considered intractable. The advantage to using elliptic curves for asymmetric cryptography is that the ECDLP is computationally harder than the DLP, and thus, the size of the underlying finite field is much smaller when compared to modular exponentiation schemes of equivalent security. Based on existing computational capabilities, integer computations in the range of 192-bits to 384-bits provide adequate security for ECC [9]. Other forms of finite field mathematics not involving modular arithmetic can be utilized for ECC [8]. However, this work focuses on modular arithmetic, also known as *GF(p)* mathematics, in order to support backwards compatibility with RSA.

To compute modular addition (subtraction), the reduction step, i.e. the modulo operation, consists of a simple conditional subtraction (addition) of the modulus. Multiplication, on the other hand, requires a more complicated reduction step. For brevity, the focus of this paper will be on modular multiplication. Many reduction techniques exist for modular multiplication. For software implementations, special Mersenne primes selected by the National Institute of Standards and Technology (NIST) can be used which allow modular congruency to be used in order to reduce a multiplication result using substitutions, additions, and subtractions [2]. The reduction operation is modulus dependent, requiring algorithm adjustments when changes to the ECC parameters are made. Obviously, this is not a problem for software implementations where reconfigurability is key. When modular multiplication is implemented in hardware, the preferred method of reduction is Montgomery reduction [15]. Operands are initially mapped into an isomorphic $GF(p)$, known as the Montgomery domain, where each reduction step simply requires the addition of a multiple of the modulus and a right shift. Montgomery reduction is ideal for hardware in the sense that the exact same reduction technique works on any prime modulus, and the algorithm has a limited number of conditional branches in comparison to other techniques. This equates to a reduced amount of control logic and allows for efficient pipelining.

When Montgomery reduction is combined with multiplication, the operation is referred to as Montgomery multiplication. Since Peter Montgomery's original proposal in 1985, significant improvements have been made on the algorithm, and a variety of computational methods exist for Montgomery multiplication. Mainly, the reduction steps have been interleaved with the multiplication steps, which in turn reduce the amount of unnecessary computation. Koç et al. provide a comprehensive examination of the various interleaved methods for Montgomery multiplication in software [12]. The Coarsely Integrated Operand Scanning (CIOS) algorithm was selected for this work due to the fact that the hardware implementation is only dependent on the word-size of the datapath and not on the bit-length of the integers it computes.

B. *Energy Consumption in Digital Circuits*

Modern VLSI circuits utilize CMOS technology. Understanding how energy is consumed in CMOS circuitry is key to creating energy efficient designs. CMOS dissipates energy in three different ways. First, there is static dissipation, which is caused by leakage current. Static power is typically caused by source to drain current when the transistor is turned off and can be described by the simple voltage times current equation:

$$P = V * I_{leak} \tag{1}$$

The second type of energy consumption is switching power, given by the following formula:

$$P = (1/2) * C_{load} * f * V^2 \tag{2}$$

C_{load} is the capacitance the transistors must drive and is made up of wire and gate capacitance. The frequency, f, is the rate at which the transistor switches and depends on switching activity information. The third type of power in CMOS technology is often referred to as internal power and is given by the following formula:

$$P = ((1/2) * C_{int} * V^2 * f) + (V * I_{sc}) \tag{3}$$

Internal power incorporates the charging of capacitances within a gate and the short circuit current which exists between the type N and P transistors during a logic state transition.

Well defined techniques exist to perform the above computations and one such set of techniques is described in the methodology section (IV) of this paper.

III. A Parameterizable Microarchitecture

This section of the paper discusses the design of a custom microarchitecture for computing modular arithmetic. The intent of the design is to accelerate the underlying mathematics required for ECC. In order to fully explore the design space associated with this sort of acceleration hardware, the HDL code has been written such that the width of the data path and the size of internally addressable memory can be adjusted prior to logic synthesis. Other design parameters such as the size of the microprogram can be adjusted as well. However, for the analysis provided in this paper, those design parameters are held constant unless stated otherwise.

A. *Hardware Design*

The microarchitecture depicted in Figure 1 is capable of executing the CIOS algorithm discussed in Section II along with modular addition and subtraction. From here on, the top level design will be referred to as the Finite Field Arithmetic Unit (FFAU). The CIOS algorithm consists of two nested for-loops. The first loop computes the following:

$$t = t + a * B$$

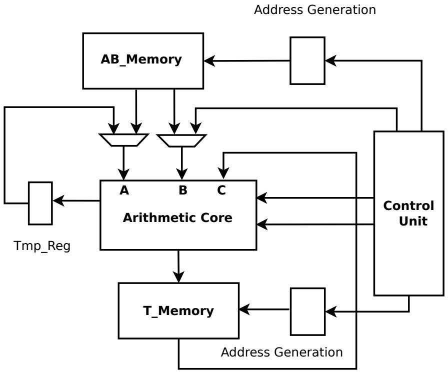

Fig. 1. Top Level Architecture of the FFAU

Multiply-Add
$(Carry, Result) \Leftarrow A * B$
$(Carry, Result) \Leftarrow A * B + C$
$(Carry, Result) \Leftarrow A * B + C + Carry$
Add-Subtract
$(Carry, Result) \Leftarrow A + B$
$(Carry, Result) \Leftarrow A + B + C$
$(Carry, Result) \Leftarrow A + B + C + Carry$
$(Carry, Result) \Leftarrow -A + B$
$(Carry, Result) \Leftarrow -A + B + C$
$(Carry, Result) \Leftarrow -A + B + C + Carry$
Clear Pipe
$(Carry, Result) \Leftarrow C + Carry$
$(Carry, Result) \Leftarrow Carry$

TABLE I
ARITHMETIC CORE COMPUTATIONAL CAPABILITIES

such that t is a vector with a word length of $k + 2$, a is a vector with a word length of k, and B is of unity word length and part of b, a vector the same size as a. Note that if l is the bit length of the finite field elements (i.e. 192, 256, or 384) and w is the bit width of the datapath, then $k = l/w$. For example, if we want to process 192-bit integers (minimal security) with a 32-bit datapath, then each integer, a and b, will be represented by $k = 6$ words. The second inner loop computes the following:

$$t = t + m * n$$

such that t is as previously defined and n is a vector of k words. m is a single word value computed just prior to its use on every iteration of the outer loop. In short, the computation in each of the inner loops involves a multiplication of a vector by a scalar and the addition of a another vector.

At the center of the FFAU is the arithmetic core. It is capable of clocking in three w-bit operands and clocking out one w-bit result on every clock cycle. For the current design, the arithmetic core has two pipeline stages and uses parallel array multiplication and Carry Save Adder (CSA) row reduction techniques. While it achieves a throughput of one operation per clock cycle, each operation has a latency of three cycles. Table I reveals a subset of the operations the arithmetic core is capable of performing.

Note that $Result$ is the lower w bits of the computation and $Carry$ is the remaining upper w bits of the computation. The arithmetic core is self draining in the sense that control bits from the control unit in addition to the store address for the corresponding result propagate through the pipeline along with the operand data. This greatly simplifies the required control logic.

The key to an efficient design is near 100% utilization of the arithmetic core. In order to avoid pipeline stalls, three w-bit operands must be fetched from internal scratchpad RAM while one w-bit result is stored on every clock cycle. To allow for the use of dual port RAMs, the memory within the FFAU is split into two memory modules. The AB memory holds the a, b, and n integers, while the T memory holds the intermediate result, t. Since the AB memory must hold three k-word integers, the minimum size of the AB memory is $3k$ words. For design simplicity and future expansion, both the AB and T memories were designed to be $4k$ deep. It should be noted that this liberal use of memory will only slightly exaggerate the energy consumption of the FFAU, but for future work, the memory size will be re-examined. The AB memory requires two read ports, and at least one of those ports must support write operations in order to load the input data. The T memory module requires only one read port and one write port for the internal FFAU architecture.

Result data from the arithmetic core can be stored in either the T memory or a temporary result register. Part of the control data which propagates with the computation is the write-enable signal for the T memory module and the load signal for the temporary result register. The A input to the arithmetic core is multiplexed to allow input from either the AB memory module or the temporary result register. The temporary result register is necessary to avoid a structural hazard that would otherwise exist during the reduction step of the CIOS algorithm when computing $t = t + m * n$. Consequently, m is stored in the temporary result register during reduction, thereby allowing the architecture to simultaneously access m and t. As with the A input, the B input of the arithmetic core is multiplexed, enabling multiplication by a value from an 8-entry, microcode selectable RAM module within the control unit. For the calculation of m, a constant must be pre-loaded into the constant RAM.

The address generation logic is responsible for addressing the read ports for both memory modules. An index register is dedicated to each read port and can be independently controlled using the binary codes found in Table II. The width of the index registers is determined by the depth of the RAM modules, i.e. $log_2(4k)$, and is automatically set prior to logic synthesis. The constant bus referenced in the table is fed by the constant RAM module within the control unit. The write port on the T memory module is addressable only by the store address pipeline within the arithmetic core. The store address along with the control data latched into the arithmetic core on every clock cycle is supplied by the control unit.

The FFAU control unit, depicted in Figure 2, is a simple microcoded state machine. It has two additional index registers for handling nested loops, a small RAM for holding constants,

Code	Operation	Description
00	Hold	no change to value
01	[reg]←const_bus	load register with value on constant bus
10	[reg]← 0	clear register value
11	[reg]← [reg]+1	increment register value

TABLE II
INDEX REGISTER CONTROL CODES

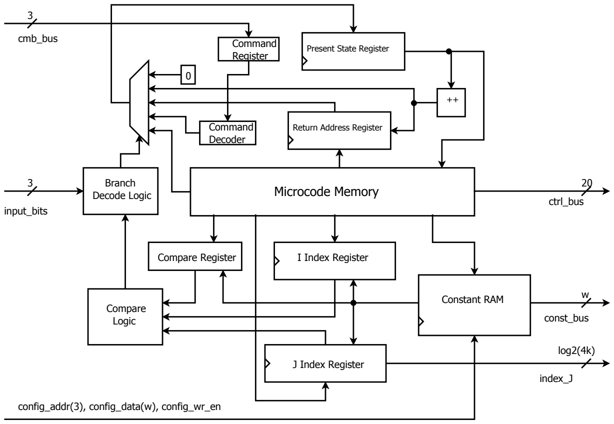

Fig. 2. The Control Unit within the FFAU

a return address register for simple subroutines, and a command decoder ROM for supporting multiple operations. The control unit is also capable of making branch decisions within a microprogram based on signals from the datapath. As seen in Figure 2, the CIOS algorithm requires a minimal amount of decision logic. Currently, the microcode ROM is 64 entries deep, which was more than enough to implement the CIOS algorithm, along with modular addition and subtraction.

B. Design Tradeoffs

Even though this study was tailored for energy efficient finite field arithmetic acceleration, many of the lessons learned here can be applied to the acceleration of other algorithms. During the design phase, a number tradeoffs were encountered. Among the most obvious is the tradeoff between reconfigurability and efficiency. Clearly, these are two conflicting design goals. As more logic is added to the design to support a wider variety of algorithms, the amount of logic being effectively utilized for a given algorithm decreases. If the accelerator is being tuned for a specific algorithm, one would want to provide just enough reconfigurability in the design to allow for a certain amount of scalability and not much more. After all, if reconfigurability is the primary design objective, the sole use of a general purpose processor should be considered.

To accommodate expansion and dynamic configuration of key size, i.e. the size of the underlying finite field, the FFAU pulls array bounds from the constant RAM within the control unit. For this design, the use of microprogramming over hardcoding the control unit is preferred in order to improve reconfigurability and reduce control unit complexity. In this case, the control complexity is moved into the microprogram; however, a good microcode assembler can help improve the situation. It should be noted that combining a microcoded control unit with a constant RAM allows for two levels of reconfigurability in addition to reducing the cost associated with the control store.

Scalability verses efficiency is another tradeoff encountered in this study. Consequently, the FFAU is only scalable up to a certain point determined by the size of scratch memories. The approach for determining memory size taken here is to look at the largest practical problem size this device might be used for. Unfortunately for cryptographic applications, the problem size grows as new attack algorithms are developed. A complexity verses performance tradeoff exists when considering the use of the temporary result register mentioned in Section III. At the cost of additional multiplexing logic and a control signal, a structural hazard is avoided, thereby reducing potential pipeline stalls. It should be noted that another solution to the aforementioned structural hazard is to add a third port to the AB memory; however, this could negatively effect the scalability of the design.

The aforementioned tradeoffs were discussed somewhat independent of the algorithm complexity. However, when considering area verses performance, it is beneficial to examine the computation time in terms of input size. For example, the number of clock cycles required to complete a CIOS operation on the FFAU is as follows:

$$cc = 2k^2 + 6k + (k+1)p + 22 \tag{4}$$

where k is as defined in part A of this section, and p is the latency of an arithmetic core operation and is directly related to the depth of the pipeline. The examination of the complexity equation above revealed to us the portions of the algorithm where a larger investment in terms of logic should be made. Optimizations that reduce the coefficients of higher ordered terms in the complexity polynomial should be prioritized for both energy efficiency and performance. Hence to reduce the coefficient of the quadratic term, the FFAU has logic for scratchpad address generation that is separate from that of the actual computation. Likewise, the memories are organized such that three operands can be fetched at once, rather than stalling while waiting for a third operand to be fetched from memory.

Another interesting tradeoff that is quantified in (4) is clock rate verses pipeline depth. In this preliminary study, this tradeoff was not really considered; instead, a depth which provided a reasonable clock rate without much logic optimization effort was chosen. However, assuming an ideal increase in clock rate due to pipelining, it is fairly straight forward to calculate an optimal pipeline depth for the FFAU using (4). Obviously, the effect p has on performance will be algorithm specific. In this situation, there is a data dependency within the outer loop of the CIOS algorithm that requires a pipeline stall, hence the coefficient, k. The $+1$ comes from the fact that the pipeline must drain before the final result is available. It should be noted that the data dependency could be removed at the cost of microcode complexity.

IV. Methodology

The previous section introduced the microarchitecture of the FFAU. This section will now discuss the pre and post synthesis evaluation of the design. Since memories are generally not synthesized directly, the test bench contains the AB and T memories. The constant RAM is on the order of a small register file and thus is implemented in logic. The estimation of energy consumed by the logic portion of the FFAU was completed entirely within the Synopsys tool chain. Synthesis is performed using the Synopsys HDL Compiler to a 45nm technology library. The Verilog Compiler Simulator (VCS) is used for both pre and post synthesis verification in addition to creating stimulus activity files for use in power estimation. Finally, average power estimation for synthesized logic was performed with PrimeTime PX using switching activity information contained within the stimulus files [19], [22].

While Synopsys provides a means for estimating average power for the logic portion of the design, the average power for the memories within the design are estimated using Cacti, a timing, power and area model for caches and simple RAM blocks [18]. Total average power estimations of each variant of the design coupled with the time required per operation taken from simulation were used to compute the energy required per operation.

To compare the design provided in this work with existing software solutions, average power estimations for an ARM Cortex-M3 were made using the STMicroelectronics STM32F2xx series datasheet, which provides the average current draw of the processor with all peripherals turned off for various clock rates [20]. The STM32F2xx series utilizes a leading-edge 90nm standard cell library and operates at a higher clock rate relative to other ultra-low power microprocessors on the market today, making it a closer match to our design. To make a more fair comparison, the datasheet power estimations are scaled to the 45nm process technology using the following methodology. Assuming dynamic power is the dominate factor for energy consumption in the processor and capacitance is proportional to the feature size of the process technology, the following formula can be utilized to scale from process a to process b:

$$scale_{atob} = (size_b/size_a) * (V_b/V_a)^2 \tag{5}$$

For the above equation, $size_a$ and $size_b$ represent the feature size of technology a and b respectively, and V_a and V_b are the supply voltages for technology a and b respectively.

Energy per operation for code executing on an ARM Cortex-M3 is estimated using the aforementioned power calculation along with execution time per operation. We used a clock cycle accurate ARM Cortex-M3 simulator built into Texas Instrument's Code Composer Studio v4 (CCStudio v4) Integrated Development Environment (IDE) to determine execution time in cycles.

V. Results

This section of the paper provides the results of our study. To demonstrate the energy efficiency of this design, average power and execution time to perform Montgomery multiplication for key sizes of 192-bit, 256-bit, and 384-bit were recorded. All of the results from our hardware assumes a 100 MHz clock and 0.9V supply voltage. Table III provides a breakdown of the average static and dynamic power consumed by the 8-bit, 16-bit, 32-bit, and 64-bit variants of our design for each of the Montgomery multiplications. In all cases, the dynamic energy is the dominate component in average power. This is primarily due to the small memories and high utilization of the arithmetic logic. The leakage power provides us some insight into how much power will be consumed if power gating is not utilized while the FFAU is idle.

Datapath Width	Area(cell units)	Static Power	Dynamic Power
Key Size: 192-bit			
8-bit	2,091	32.3 μW	166.2 μW
16-bit	4,244	59.3 μW	311.9 μW
32-bit	11,329	159.1 μW	659.9 μW
64-bit	36,582	530.6 μW	1,472.7 μW
Key Size: 256-bit			
8-bit	2,091	34.0 μW	186.2 μW
16-bit	4,244	61.6 μW	310.2 μW
32-bit	11,327	161.4 μW	684.4 μW
64-bit	36,582	532.9 μW	1,613.4 μW
Key Size: 384-bit			
8-bit	2,168	35.4 μW	197.1 μW
16-bit	4,322	65.0 μW	321.6 μW
32-bit	11,405	164.3 μW	888.5 μW
64-bit	36,664	535.7 μW	1,686.5 μW

TABLE III
Area Utilization, Static Power, and Dynamic Power vs. Datapath Width

Table IV provides the total average power along with execution time and energy per Montgomery multiplication with respect to the datapath width. When comparing integer key sizes from smallest to largest, we note that average power only increases slightly, whereas the computation time increases quadratically. The increase in average power is mainly due to a linear increase in memory, which accounts for an increase in leakage power. The significant increase in computation time is due to the $O(n^2)$ nature of the multiplication operation. When comparing datapath bit widths of the FFAU, the average power increases less than quadratically as the datapath width doubles. The net result is that the energy per CIOS operation tends to decrease as the datapath width increases. To demonstrate this, Figure 3 charts the amount of energy consumed per 192-bit, 256-bit, and 384-bit operation for each of the variants of the FFAU. Due to the fact that the CIOS algorithm is not perfectly quadratic, the decrease in energy consumed per operation does not continue. As can be seen for the 192-bit key case with a 32-bit datapath, at some point increasing the datapath width starts to increase the energy consumed, leading to an optimal datapath width in terms of energy for a given key size. We believe this trend continues for larger key sizes; however, the optimal datapath width is higher than or equal to 64-bits.

From the results above, it becomes clear that an algorithm with a $O(n^2)$ time complexity tends to favor a larger datapath when considering energy efficiency, while the energy efficiency of a $O(n)$ algorithm will not be significantly affected by datapath size. Moreover, for an algorithm exhibiting a $O(1)$

behavior, a decrease in datapath size will yield an increase in energy efficiency. The latter observation is quite obvious considering the constant portion of the time complexity equation represents the serial bottle neck within the algorithm. For our study, the time complexity has a fairly large constant coefficient which is a result of the dynamic configuration of the FFAU at which time the arithmetic logic lies idle.

In order to provide some insight into the relative energy efficiency of the FFAU, Figure 3 also includes the energy per operation estimations for the ARM Cortex-M3 operating at 100 MHz with a 0.9V supply voltage. Table V lists the energy estimations for the ARM processor since they extend beyond the scale of the graph in Figure 3. In terms of performance, the FFAU on average yields a 10x improvement over the ARM.

Width	Average Power	Ex. Time	Energy
Key Size: 192-bit			
8-bit	198.5 μW	13,920 ns	2.763 nJ
16-bit	371.2 μW	4,220 ns	1.566 nJ
32-bit	819.0 μW	1,520 ns	1.245 nJ
64-bit	2,004.3 μW	710 ns	1.423 nJ
Key Size: 256-bit			
8-bit	220.2 μW	23,510 ns	5.176 nJ
16-bit	371.8 μW	6,710 ns	2.495 nJ
32-bit	845.7 μW	2,150 ns	1.818 nJ
64-bit	2,146.3 μW	830 ns	1.782 nJ
Key Size: 384-bit			
8-bit	232.5 μW	50,550 ns	11.755 nJ
16-bit	386.6 μW	13,830 ns	5.347 nJ
32-bit	888.5 μW	4,110 ns	3.652 nJ
64-bit	2,222.3 μW	1,410 ns	3.133 nJ

TABLE IV
AVERAGE POWER, EXECUTION TIME, AND ENERGY PER OP VS. DATAPATH WIDTH

Key Size	Ex. Time	Average Power	Energy
192-bit	13,870 ns	4,500 μW	62.4 nJ
256-bit	23,010 ns	4,500 μW	103.6 nJ
384-bit	48,530 ns	4,500 μW	218.4 nJ

TABLE V
AVERAGE POWER AND ENERGY PER OP VS. KEY SIZE (ARM CORTEX-M3)

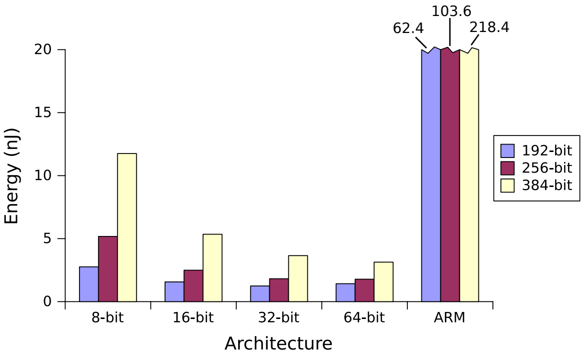

Fig. 3. Energy per Montgomery Multiplication vs. Architecture

VI. PREVIOUS WORK

The work discussed in this paper is loosely based off of our prior work performed for Sandia National Laboratories (SNL) [21]. The development in this paper is intended to target a low power ASIC or custom processor design, whereas the work for SNL targeted an FPGA.

Much effort has been dedicated to achieving significant acceleration using hardware in FPGA and ASIC designs. However, only a few publications seem to address the energy consumption aspect of public key cryptography for embedded devices. In order for public key cryptography to be viable in extremely low power applications such as RFIDs, a better understanding of the energy cost associated with asymmetric encryption in both hardware and software is necessary. Wander et al. compare the energy cost of 1024-bit RSA with that of 160-bit ECC to show that 160-bit ECC saves a significant amount of energy when executed on an 8-bit Atmel ATmega128L microprocessor. The results showed that based on an assumed battery life, the device using ECC could execute 4.2 times the number of key exchange operations [23]. Naturally, this is a very compelling argument for using elliptic curves instead of modular exponentiation based cryptosystems.

Keller et al. have examined the public key energy consumption for FPGAs. First, the design of an entire asymmetric cryptographic processor is explained. Then, the design is implemented on an Xilinx Spartan 3E FPGA, and it is characterized in terms of its energy consumption. The processor is capable of utilizing binary or prime finite fields. For prime field mathematics, the authors used 192-bit integers, while for binary mathematics, 163-bit polynomials were used. For energy consumption characterization, the authors kept the bit lengths the same but made various algorithmic changes. What they found was that the power consumption of the FPGA remained quite constant throughout their experimentation, and thus, the speed of the operation had more bearing on energy consumption. The system configuration which performs the fastest ends up yielding the more energy efficient design [10].

Goodman et al. compared public key cryptographic operations on a domain-specific reconfigurable cryptographic processor (DSRCP) with previously reported FPGA implementations and a software only implementation on a strongARM. The DSRCP was implemented in a 0.25 μm processes technology, and the energy consumption numbers were true measurements. The authors report orders of magnitude lower energy consumption for the DSRCP in comparison to the software and FPGA implementations. They claim the DSRCP is just as reconfigurable as software, yet they only consume half the energy of previously reported non-reconfigurable hardware solutions [7].

VII. CONCLUSION

This paper provides a unique microarchitecture for computing prime finite field arithmetic. The FFAU is a parametrizable architecture such that the datapath width and maximum integer width can be set prior to logic synthesis. The design is simulated and synthesized in order to verify correctness and estimate the energy required for Montgomery multiplication of key sizes from 192-bit to 384-bit. Moreoever, we have shown that the majority of the time for an ECC scalar point multiplication is spent in the modular multiplication operation. Thus, optimizing a design for modular multiplication is quite useful. The CIOS algorithm, however, is not perfectly quadratic; thus, depending on the size of integers being computed, there is a sweet spot where energy consumption is minimized.

Figure 3 demonstrates a sweet spot for energy efficiency at a 32-bit datapath width for a 192-bit computation. As the computation size increases, it should be noted that the graph appears to shift towards the larger datapath widths. With 256-bit math, the 64-bit width appears to be optimal. Expanding our results might show that a 128-bit datapath would provide optimal energy efficiency for 384-bit security. Likewise, we expect an examination of 160-bit integer computation would show the 16-bit datapath to be optimal in terms of energy. Another conclusion seen here is that the choice to utilize an 8-bit microprocessor over a 32-bit microprocessor for energy efficiency reasons may not be the best one, despite the lower power utilization.

The results provided in this paper also shed some light on how much additional security can cost in terms of energy. These are important considerations that should be taken into account early on in the design phase. Obviously, the required security has a large influence on the amount of energy a device will consume. If a particular device does not have an energy budget large enough to support pure software based encryption, other alternatives, such as the hardware acceleration shown here, exist.

For future work, it would be interesting to compare the energy consumption of this work with that of a large bit serial multiplier. Additionally, the FFAU should be included in a larger system capable of computing complete ECC algorithms for a true estimation on the energy cost of security.

VIII. Acknowledgments

We would like to thank Boris Grot and Hyungjun Kim for their contributions to this paper.

References

[1] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede, “Public-Key Cryptography for RFID-Tags,” *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, .pp 217-222, 2007.

[2] M. Brown, D. Hankerson, J. López, A. Menezes, “Software Implementation of the NIST Elliptic Curves Over Prime Fields,” *Topics in Cryptography - CT-RSA 2001*, LNCS, vol. 2020, pp. 250-265.

[3] H. Cohen, A. Miyaji, T. Ono, “Efficient elliptic curve exponentiation using mixed coordinates,” *Advances in Cryptography: ASIACRYPT’98*, vol. 1514, pp. 51-65.

[4] A. Daly, W. Marnane, T. Kerins, E. Popovici, “An FPGA implementation of a GF(*p*) ALU for encryption processors,” *Microprocessors and Microsystems*, vol. 28, pp. 253-260, 2004.

[5] W. Diffie and M. E. Hellman, “New Directions in Cryptography,” *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644-654, 1976.

[6] G. Gaubatz, J. Kaps, E. Öztürk, B. Sunar, “State of the Art in Ultra-Low Power Public Key Cryptography for Wireless Sensor Networks,” *Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pp. 146-150, 2005.

[7] J. Goodman, A.P. Chandrakassan, “An Energy-Efficient Reconfigurable Public-Key Cryptography Processor,” *IEEE Journal of Solid-state Circuits*, vol. 36, no. 11, 2001.

[8] D. Hankerson, J. Hernandez, A. Menezes,“Software Implementation of Elliptic Curve Cryptography over Binary Fields,” *Cryptographic Hardware and Embedded System:CHES 2000*, LNCS, vol. 1965, pp. 1-24.

[9] D. Johnson, A. Menezes, S. Vanstone,“The Elliptic Curve Digital Signature Algorithm (ECDSA),” Technical report CORR 99-34, Dept. of C&O, University of Waterloo, 1999.

[10] M. Keller, A. Byrne, W. P. Marnane, “Elliptic Curve Cryptography on FPGA for Low-Power Applications,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, no. 1, 2009.

[11] N. Koblitz, ”Elliptic Curve Cryptosystems,” *Mathematics of Computation*, vol. 48, no. 177, .pp 203-209, 1987.

[12] Ç.K. Koç, T. Acar, and B.S. Kaliski Jr.,“Analyzing and Comparing Montgomery Multiplication Algorithms,” *IEEE Micro*, vol. 16, no. 3, pp. 26-33, 1996.

[13] A. Menezes, P.C. van Oorschot, S.A. Vanstone,*Handbook of Applied Cryptography*, CRC Press, 1997.

[14] V.S. Miller, “Use of Elliptic Curves in Cryptography,” *Advances in Cryptography*, LNCS, pp. 417-426, 1986.

[15] P.L. Montgomery, “Modular Multiplication Without Trial Division,” *Mathematics of Computation*, vol.44, no. 170, pp. 519-521, 1985.

[16] D. Naccache and D. M’Raïhi, “Cryptographic Smart Cards,” *IEEE Micro*, vol. 16, no. 3, pp.14-24, 1996.

[17] R.L. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, 1978.

[18] P. Shivakumar N.P. Jouppi, “CACTI 3.0: An Integrated Cache Timing, Power, and Area Model,” Compaq: Western Reaserch Laboratory, 2001.

[19] Synopsys, *Expanding the Synopsys Prime Time Solution with Power Analysis*, online resource,
http://www.synopsys.com/Tools/Implementation/SignOff/CapsuleModule/ptpx_wp.pdf, 2006.

[20] STMicroelectronics, *STM32F215xx Datasheet*, onlie resource,
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00263874.pdf November, 2010.

[21] A.D. Targhetta, “Elliptic Curve Cryptographic Acceleration Unit,” Sandia National Laboratories, SAND2010-2697, available upon request, May 2010.

[22] Virginia Tech, “Synopsys Tutorial: Power Estimation,” online resource,
http://computing.ece.vt.edu/wiki/Synopsys_Tutorial:_Power_Estimation, 2008.

[23] A.S. Wander, N. Gura, H. Eberle, V. Gupta, S.C. Shantz, “Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks,” *IEEE International Conference on Pervasive Computing and Communications*, vol. 0, pp. 324-328, 2005.