System Integration of Elliptic Curve Cryptography on an OMAP Platform

Sergey Morozov
ECE Department
Virginia Tech
Blacksburg, VA 24061
Email: morozovs@vt.edu

Christian Tergino
ECE Department
Virginia Tech
Garden City Park, NY 11040
Email: ctergino@vt.edu

Patrick Schaumont
ECE Department
Virginia Tech
Blacksburg, VA 24061
Email: schaum@vt.edu

***Abstract*—Elliptic Curve Cryptography (ECC) is popular for digital signatures and other public-key crypto-applications in embedded contexts. However, ECC is computationally intensive, and in particular the performance of the underlying modular arithmetic remains a concern. We investigate the design space of ECC on TI's OMAP 3530 platform, with a focus on using OMAP's DSP core to accelerate ECC computations for the ARM Cortex A8 core. We examine the opportunities of the heterogeneous platform for efficient ECC, including the efficient implementation of the underlying field multiplication on the DSP, and the design partitioning to minimize the communications overhead between ARM and DSP. By migrating the computations to the DSP, we demonstrate a significant speedup for the underlying modular arithmetic with up to 9.24x reduction in execution time, compared to the implementation executing on the ARM Cortex processor. Prototype measurements show an energy reduction of up to 5.3 times. We conclude that a heterogeneous platform offers substantial improvements in performance and energy, but we also point out that the cost of inter-processor communication cannot be ignored.**

I. Introduction

Modern mobile devices and embedded systems are designed to provide an ever increasing number of services for the end-user. Secure communication in particular is in demand for mobile phones, cars, remote operation and sensing, and more. A vital component of a secure communications software stack includes signatures and key-exchange, requiring public-key cryptographic algorithms. In contrast to symmetric-key cryptography, the computational costs associated with public key cryptography are high - typically three orders of magnitude above symmetric-key cryptography. Public-key algorithms like RSA, DSA, and elliptic curve cryptography (ECC) all deal with operands hundreds to thousands of bits in size and use modular arithmetic for the underlying computations. In particular, modular multiplications are a critical underlying part for public-key cryptography, and it can be prohibitively expensive on smaller embedded processors.

While algorithmic-level improvements are continually proposed, efficient mapping in hardware or software is always important. In an embedded context, where the application has a strong influence on the system architecture, dedicated hardware solutions are appealing. This includes for example, expanding the capabilities of an existing processor with specialized instructions or designing completely new co-processors. Researchers often approach the problem by coming up with an HDL description of the new co-processor or datapath, and conclude the research with a simulation or an FPGA prototype of the design.

We approached this problem from another angle. We have observed that recent trends in mobile and embedded computing have pushed increasingly complex System-on-Chip designs into heterogeneous platforms such as TI's OMAP and DaVinci chips. These SoCs now frequently include multiple heterogeneous cores: a standard RISC core could be augmented with DSP, SIMD extensions, and a graphics processing unit. Our research specifically addresses the use of a VLIW DSP processor for cryptography. Previous research has shown that such a processor could efficiently implement modular arithmetic [1], [2]. In addition, the side-channel characteristics of ECC on a DSP have been investigated [3]. Our objective was to study the integration costs that arose from ECC in a heterogeneous, DSP-enabled SoC.

The contributions of this paper are two-fold. First, we present an original implementation of binary-field modular arithmetic on a VLIW DSP processor. Our solution implements multi-precision arithmetic that utilizes the DSP's XOR-multiply instructions, and also exploits multiple concurrent datapaths to achieve a significant speed-up of binary field multiplication relative to the ARM RISC core on the same chip.

Second, we examine the design space of elliptic curve cryptography systems on a typical heterogeneous multicore platform, in this case the TI OMAP 3530. This chip contains an ARM Cortex A8 and a C64x+ DSP core. We present a solution for high speed ECC cryptography services under varying ECC cryptographic parameters. We show that the multicore implementation demonstrates a significant speed-up over existing production-grade ECC libraries for a general-purpose processor. We then make a strong effort to create a fair comparison between our heterogeneous ARM+DSP implementation, and OpenSSL ARM-only implementation. The comparison covers performance as well as energy cost of the resulting design.

The remainder of this paper is presented as follows. We first review the basis of elliptic-curve cryptography, enumerating the computational requirements. We then discuss an imple-

mentation of binary-field modular arithmetic on a VLIW DSP. In Section 4, we integrate the resulting library in a system-library for elliptic-curve cryptography on the OMAP platform. We analyze the results in Section 5 and conclude.

II. Computational Cost of Elliptic Curve Cryptography

The basics of elliptic curve cryptography are extensively discussed in literature; see for example Hankerson et al. [4]. In this section, we will only highlight the computational characteristics of ECC.

ECC is defined using Elliptic Curves over prime or binary finite fields. Prime fields $GF(p)$ use integers less than a p-bit prime; binary fields $GF(2^m)$ use polynomials with binary coefficients of degree less than an m-degree irreducible polynomial. The elliptic curve domain parameters, as well as the field definitions, are standardized, such as by SEC [5] or by NIST [6]. Our experiments were performed using prime curves on 160 and 224 bit fields, and using binary curves on 163, 283, and 409 bit fields. These curves are called $sec160pr1$, $secp224r1$, and $sect163r1$, $sect283r1$, $sect409r1$ for prime-field and binary-field SEC curves respectively. The large field-sizes used in ECC imply that the fundamental operations have to be implemented using multi-precision arithmetic. For example, the prime-field $secp160r1$ curve requires 160-bit operands, which each take 5 words on a 32-bit processor. Each operation in 160-bit arithmetic needs to be expressed in terms of a 32-bit datapath.

The central operation in ECC is the point multiplication, which multiplies a point on a curve with a scalar number k. Cryptographic protocols are constructed based on the point multiplication [6]; for example, generating a digital signature requires one point multiplication, while verifying it requires two point multiplications. The point multiplication is computationally expensive. A straightforward implementation of a point multiplication on the $secp160r1$ curve requires 1600 modular multiplications and 960 modular squarings, each of them in a 160-bit field. Hence, the crypto-engineering community makes a significant research effort to optimize the elliptic-curve algorithms. For example, the Explicit-Formulas Database [7] is an excellent source to find the latest optimizations to the ECC algorithms. Regardless of this optimization effort, however, it is clear that optimized mapping of the existing algorithms to processors is important. In this paper, we focus on this aspect, starting with efficient mapping of binary-field modular multiplication.

III. Efficient Modular Multiplication on a VLIW DSP

In this section, we describe an efficient implementation of modular arithmetic on a VLIW DSP processor, using the TI C64x+ in the OMAP as an example. We focus on the case of binary-field arithmetic; a discussion on efficient modular arithmetic on a VLIW DSP for prime field may be found in [1].

TABLE I
SEC Curve parameters used in this research

Curve	Field Size for $GF(p)$
secp160r1	$2^{160} - 2^{31} - 1$
secp224r1	$2^{224} - 2^{32} - 2^{12} - 2^{11} - 2^9 - 2^7 - 2^4 - 2 - 1$
Curve	Irreducible Polynomial for $GF(2^m)$
sect163r1	$x^{163} + x^7 + x^6 + x^3 + 1$
sect283r1	$x^{283} + x^{12} + x^7 + x^5 + 1$
sect409r1	$x^{409} + x^{87} + 1$

$GF(2^m)$ relies on binary polynomial multiplications. A large majority of processors do not have support for this arithmetic. For this reason, software implementations of ECC often stick to prime field ($GF(p)$). However, the C64x+ *does* have support for binary polynomial multiplication, an often-used operation in the domain of error control coding [8]. In our binary-field operations, we make use of this feature. In addition, we target an implementation that makes optimal use of the instruction-level parallelism in the VLIW DSP.

A. Binary Field Arithmetic

Binary field numbers are within $GF(2^m)$ and are represented in the form

$$A(x) = a_0x^0 + a_1x^1 + ... + a_{m-1}x^{m-1}, \tag{1}$$

where each coefficient $a_i = \{0, 1\}$. A binary field also needs an irreducible polynomial $f(x)$:

$$\begin{aligned} f(x) &= T(x) + x^m \\ T(x) &= x^0 + t_1x^1 + t_2x^2 + .. + t_{m-1}x^{m-1} \end{aligned} \tag{2}$$

where every coefficient, $t_i = \{0, 1\}$. All operations in $GF(2^m)$ are performed modulo $f(x)$. Addition and subtraction are equivalent to performing a bitwise Exclusive-OR between the two operands. To multiply two polynomials, one will first perform a polynomial multiplication, and next reduce the result using the irreducible polynomial that defines the field. When multiplying two polynomials $A(x)$ and $B(x)$ of degree $m-1$, the result is a polynomial $C(x)$ of degree $2m-2$.

$$\begin{aligned} C(x) &= c_0x^0 + c_1x^1 + ... + c_{2m-2}x^{2m-2}, \\ &with \\ &\quad c_0 = a_0.b_0, \\ &\quad c_1 = a_1.b_0 + a_0.b_1 \\ &\quad c_2 = a_2.b_0 + a_1.b_1 + a_0.b_2 \\ &\quad ... \\ c_{2m-2} &= a_{m-1}.b_{m-1} \end{aligned} \tag{3}$$

The result of the modular multiplication of $A(x)$ and $B(x)$ is $C(x)\ mod\ f(x)$. We next describe an efficient implementation of the $mod\ f(x)$ operation.

B. Unbalanced Exponent Modular Reduction

The binary polynomials in the fields we used ($sect163r1$, $sect283r1$, $sect409r1$) are unbalanced: this means that $T(x)$ in (2) is of low degree and much smaller than m. An efficient

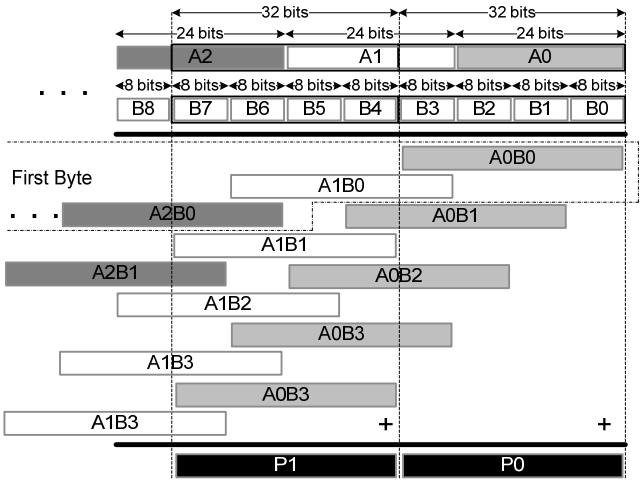

Fig. 1. Multi-precision binary polynomial multiplication: Partial products from 24-bit by 8-bit XORMPYs are XORed into 32-bit products.

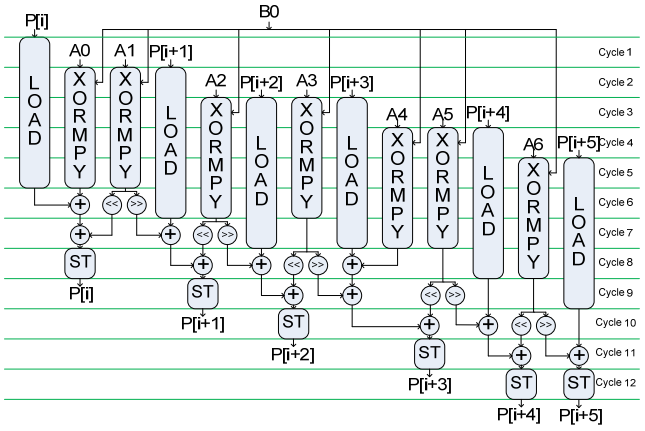

Fig. 2. Execution diagram of a multiply between a 24-bit word of operand A and a byte of operand B in $GF(2^m)$.

method for reduction over binary fields, which exploits this unbalanced character, was introduced in [9]. To reduce $C(x)$, it is first divided into two parts $C_h(x)$ and $C_l(x)$, both of which have a degree lower than m. Then, the reduction can be formulated as follows.

$$\begin{aligned} Since\ C(x) =& C_l(x) + C_h(x)x^m \\ and\ T(x) =& x^m\ mod\ f(x) \\ then\ C(x)\ mod\ f(x) =& C_l(x) + C_h(x)T(x)\ mod\ f(x) \end{aligned} \tag{4}$$

In case the product $C_h(x)T(x)$ still has coefficients with a degree higher than m, the reduction step in Equation (4) is applied repeatedly. For the polynomials used in our experiments, no more than two iterations are needed for a complete reduction. We can now map the efficient multiplication and reduction on the OMAP's DSP processor as follows.

C. Binary-Field Multiplication on TI C64x+ DSP

The C64x+ is a VLIW processor. It contains two identical data paths, each with four functional units. Each instruction is 32-bits and the processor can execute 8 instructions every cycle. The processor contains two register files, one for each data path. Each register file has 32 32-bit registers. Of particular importance for this paper is the XORMPY instruction on the C64x+. This instruction is identical to a normal multiplication except that the partial products are XORed together instead of added. This is ideal for binary polynomial multiplication. The operand sizes for XORMPY are limited to 32-bits and 9-bits and the product size is limited to 32-bits.

When implementing long-word arithmetic such as for $GF(2^m)$ multiplication, we need to use all output bits of XORMPY, in order to minimize the total number of XORMPY per modular multiplication. Therefore, we limit the input operand sizes to 24-bit and 8-bit respectively. Figure 1 shows an operand, "A", being multiplied by an operand, "B." Operand "A" must be shifted into 24-bit subwords. Each subword can then be multiplied by each byte of the "B" operand. This produces partial products which can then be XORed together to form a complete product, which still needs to be reduced.

Creation of the partial products, shifting, and accumulating them, is the most computationally intensive part of the $GF(2^m)$ modular multiplication. Given the C64x+'s parallelized architecture, this series of instructions can be computed quickly. Every cycle, the DSP can perform two XORMPYs, two shifts along with four XOR operations. Figure 2 shows the resulting execution diagram. In this figure, the rows represent cycles and the columns show concurrent operations. While only two XORMPYs can be dispatched every clock cycle, the datapath is internally pipelined, so that up to 8 XORMPYs can concurrently execute. Our $GF(2^m)$ fully exploits the parallelism offered by the DSP. The final step is the modular reduction, illustrated next.

D. Modular Reduction on the TI C64x+ DSP

After the full polynomial product is formed, it must be reduced into $GF(2^m)$. For the standard curves we studied, the unbalanced exponent modular reduction is applicable. However, depending on the specific form of $T(x)$, we use two different methods for implementing the unbalanced exponent reduction. Figure 3 illustrates both cases. If three of four or more terms in $T(x)$ are within 9 bits of each other, then it is beneficial to use XORMPYs, shifts and XORs to reduce the product. Otherwise, it is more efficient to shift and add for reduction. If most of terms in $T(x)$ are within a 9-bit window, then these terms can fit in the 9-bit operand of XORMPY. Therefore, XORMPY can be used by multiplying $T(x)$ by $C_h(x)$ using 24-bits at a time. For example, in the case of $GF(2^{163})$, the irreducible polynomial is: $f(x) = x^{163} + x^7 + x^6 + x^3 + x^0$. So $T(x) = x^7 + x^6 + x^3 + x^0$ or 0xC9. Performing the first iteration of reduction requires $T(x)$ be multiplied with seven 24-bit words. The next and final iteration of the reduction requires $T(x)$ be multiplied with one 24-bit word. An example that illustrates the other case is

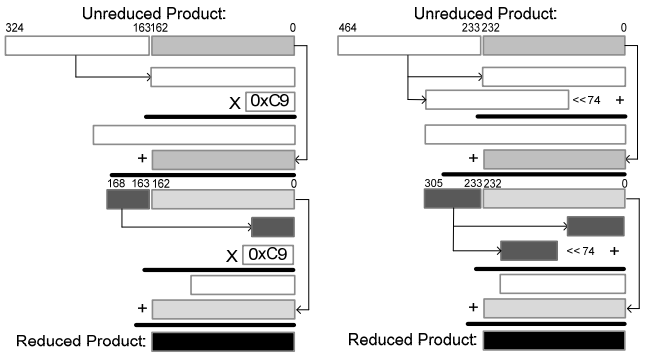

Fig. 3. Unbalanced Exponent Modular Reduction. Left, using XORMPY. Right, using shift-XOR.

reduction in $GF(2^{233})$. In this case, $f(x) = x^{233} + x^{74} + x^0$. A product in this case is 465 bits long. To reduce, we shift all bits above bit 232 to the right 233, then XOR them with the bits occupying bits 232-0. We XOR that result with $C_h(x)$ shifted to the right 233-74 = 159 bits. Since the result extends out over 233 bits, these steps must be repeated once more.

E. Performance Evaluation

We implemented the binary field modular multiplication described above for five different field sizes on an OMAP processor. We mapped the proposed algorithms in C. We compiled this code with the TI C6x C compiler under high optimization (-O3). By inspecting the assembly code generated by the compiler, and carefully rewriting the C code, we were able to achieve a parallel schedule exactly as shown in Figure 2. Our prototyping platform is a Beagle Board (http://beagleboard.org); the DSP processor executes at 360MHz while the ARM processor executes at 500 MHz.

We compare the results of the DSP to those achievable with the optimized openSSL cryptographic library for ARM. Table II shows the execution times and cycle counts, achieved for the DSP and the ARM. The table shows both binary field and prime field implementations. The DSP implementation for binary field follows the design explained above, while the DSP implementation for the prime field used the library developed by Yan et al in [1].

The DSP implementations outperform the ARM implementations significantly, even though the DSP runs at a lower clock frequency. This is clearly the result of instruction-level parallelism. The binary-field implementations show a higher speedup then the prime-field implementations. This is because the ARM does not have an XORMPY instruction, and hence has to emulate binary-field multiplication with shifts and XORs.

Our strategy to implement unbalanced exponent reductions is demonstrated in Table III. Using the shift-and-add method works better for reductions in $GF(2^{233})$ and $GF(2^{409})$, while XORMPY works better for reductions in $GF(2^{163}$ and $GF(2^{283})$. This supports the claim that if most set bits in the irreducible polynomial fit in a 9-bit window, it is more effective to reduce using XORMPYs.

TABLE II
PERFORMANCE COMPARISON OF MODULAR MULTIPLICATION

Field Size	ARM A8 500MHz		C64x+ DSP 360MHz		Speedup
	Time (μs)	Cycles	Time (μs)	Cycles	
$GF(2^{163})$	7.32	3660	0.97	349	7.5
$GF(2^{283})$	17.4	8700	2.35	846	7.4
$GF(2^{409})$	30.4	15200	3.29	1184	9.24
$GF(P160)$ [1]	3.34	1670	0.83	299	4.09
$GF(P224)$ [1]	7.81	3905	1.57	565	4.97

TABLE III
CYCLE COUNTS FOR UNBALANCED EXPONENT MODULAR REDUCTION

Field Size	Irreducible Polynomial	Shift-Add Method (Cycles)	XORMPY Method (Cycles)
$GF(2^{163})$	$x^{163} + x^7 + x^6 + x^3 + x^0$	28	19
$GF(2^{233})$	$x^{233} + x^{74} + x^0$	33	42
$GF(2^{283})$	$x^{283} + x^{12} + x^7 + x^5 + x^0$	90	64
$GF(2^{409})$	$x^{409} + x^{87} + x^0$	52	66

IV. ELLIPTIC-CURVE SYSTEM INTEGRATION

We next discuss how to integrate this fast modular arithmetic into Elliptic Curve Cryptography, executing on a heterogeneous platform that includes the ARM as well as the DSP. We first introduce the target platform, and then discuss the proposed partitioning.

A. TI OMAP

We use the TI OMAP processor as a driving example, but we note that many similar architectures exist. OMAP is popular among cell-phone manufactures such as Nokia and Samsung, and has also found its way into internet tablets and smartphones. Though many variations of the device exist, most versions combine an ARM CPU with a DSP, packaged together with a myriad of other peripherals [10].

Figure 4 shows the organization of the relevant subsystems. The central part of OMAP3530 is the microprocessor with the ARM Cortex-A8 core. This subsystem itself includes multiple submodules: processor's level 1 and level 2 caches, interrupt controller, and bridge modules to the main bus referred to as L3 interconnect. The C64x+ DSP is located in a separate subsystem, with its own caches, interrupt controller, and bus bridge/memory management unit. The L3 Interconnect bus is implemented using Open-core protocol (OCP), and connects the ARM, DSP, SDRAM controller and other subsystems together. There is also an additional bus referred to as the L4 Interconnect to connect additional peripherals.

To facilitate interprocessor communication the OMAP3530 chip includes an interprocessor communication (IPC) module attached to the L4 Interconnect. The IPC consists of a memory-mapped mailbox which contains small hardware FIFOs and connections to the interrupt controllers of both ARM and DSP subsystems. These FIFOs store 4 messages of 32-bits in size, and are typically used to exchange memory addresses. The actual sharing of data is done through a shared memory, typically a predefined section on the RAM on the L3 bus.

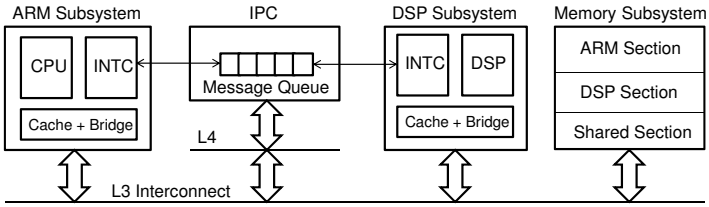

Fig. 4. OMAP 350 ARM/DSP System Overview.

TABLE IV
INTERPROCESSOR COMMUNICATION OVERHEAD

Library	Message Size (bytes)	Round-Trip Delay (μs)	ARM Cycles per byte (ARM @500MHz)
DSPLink	128	153	597
	1024	300	146
DSPBridge	128	168	656
	1024	254	124

The above implies that interprocessor communication between the ARM and DSP carries significant overhead. A typical sequence requires that the sending processor prepares the shared memory region, resolves cache coherency issues, transmits a message to the IPC queue, and triggers an interrupt for the reading processor. The reading processor has to either be polling the IPC queues or process the interrupt, translate the transmitted addresses to its own memory mapping, and read the data as necessary. Table 1 shows the overhead as measured using two communication libraries provided by TI. The fourth column shows the number of ARM processor cycles for the round-trip transfer of a byte. We note that there is a strong dependency on the message size, and that the communication overhead is over a hundred clock cycles per byte, even when optimized libraries are used. This observation will strongly influence the partitioning of ECC functionality over the ARM and the DSP.

B. ECC partitioning

Since the ARM in the OMAP has a central control function, it is natural to implement ECC as a service in terms of the system control software on this ARM. The ARM will thus initiate the ECC point multiplication, and spawn off computations to the DSP as needed.

In ECC, a distinction must be made between the low-level modular arithmetic in terms of binary-field and prime-field operations, and the point operations, which implement the scalar multiplication of EC points. A straightforward partitioning would map the modular arithmetic onto the DSP, and keep the point operations on the ARM. However, given the communication-overhead numbers cited earlier, this can be an unfavorable choice. Even for the largest field size, a field multiplication is only of the order of 30 microseconds, while the round-trip time to the DSP is at least 150 microseconds. Hence, the speedup provided by the DSP in executing modular arithmetic is completely lost in interprocessor communication.

We therefore migrated the entire point multiplication to the DSP. The ARM only handles the EC domain parameters, communicates the EC point and the scalar to the DSP, and retrieves the result of the point multiplication. An additional argument in favor of this partitioning is that the ARM is free for other tasks while the DSP is executing the point multiplication.

In the following section, we discuss the performance of this design as compared to a design that only uses the ARM.

TABLE V
ELLIPTIC CURVE IMPLEMENTATION

Binary-field Design	
EC Curve and Parameters	sect163r1, sect283r1, sect409r1
Coordinate System	Lopez-Dahab Projective
Point Multiplication	Algorithm 2P [11]
Point Doubling	Mdouble Algorithm in [11]
Point Addition	Madd Algorithm in [11]
Prime-field Design	
EC Curve and Parameters	secp160r1, secp224r1
Coordinate System	Jacobian Projective
Point Multiplication	Algorithm 3.31 in [4]
Point Doubling	Algorithm 3.21 in [4]
Point Addition	Algorithm 3.22 in [4]

V. RESULTS

We implemented all of the designs discussed so far on TI's Beagle Board. This board provides accurate performance evaluation, and also enabled us to obtain power- and energy-measurements. We first discuss the characteristics of the select EC point multiplications, then present performance evaluation, and finally provide analysis of power and energy consumption.

A. Selected Elliptic Curve Designs

Table V lists our implementation choices for binary-field and prime-field ECC implementation. Projective coordinates are used to reduce the amount of modular arithmetic required during point-multiplication. As mentioned earlier, the prime-field ECC implementation is ported from the work reported in [1].

An important objective of this work was to provide a fair comparison for elliptic-curve cryptography between a single-core ARM design, and a DSP-accelerated design. We use the implementation used in the popular OpenSSL crypto library as the baseline ARM implementation. By default, the algorithms in the library differ slightly from the variants listed in Table V. However, we made sure that the point multiplication on the ARM (OpenSSL) and the ARM+DSP (this paper) was identical in terms of computational complexity.

B. Performance

Table VI shows the resulting execution time of a point multiplication for different field sizes for a design using only the ARM versus a design using the ARM with the DSP. In the case of the DSP accelerated design, we measure the time from just prior to sending the operands to DSP, to right after the ARM receives the reply containing the computed point. In the case of OpenSSL ARM implementation, we measure the time for the point multiplication function to return.

The results indicate that our ARM-DSP implementation is able to preserve a significant part of the speed-up gained through accelerated field arithmetic, although it is less than

TABLE VI
EC POINT MULTIPLICATION TIMINGS ON THE TI OMAP3530

Elliptic Curve	ARM only Time (μs)	ARM + DSP Time (μs)	Speedup
Binary Field			
sect163r1	9674	2106	4.59
sec283r1	35034	7965	4.4
sect409r1	82611	16083	5.14
Prime Field			
secp160r1	8700	2929	2.97
secp224r1	15609	6043	2.58

TABLE VII
POWER CONSUMPTION DURING EC POINT MULTIPLICATION

Mode	Current	Power	Relative
ARM Idle	0.132	0.676	1
ARM Active	0.15	0.771	1.141
ARM + DSP Active	0.155	0.795	1.177

the ideal speedup achievable for field multiplications on the DSP. This is explained as follows. When a complex EC Point multiplication executes on a DSP, there is a certain amount of utility tasks, during which the DSP datapath cannot be fully utilized. Such additional tasks include fetching operands from memory, and creating and zeroing out temporary variables. Hence, the utilization of the DSP over the entire EC point multiplication is lower than when doing only modular multiplications. This observation is supported by the fact that a larger performance drop is seen for larger field sizes.

C. Power Efficiency

The Beagle Board's PCB design includes a test point for power measurement. Two exposed pins enable measurement of the voltage drop across a 0.3 ohm resistor. The resistor is located right between the board's power supply and the primary power regulator chip TPS65950. We measured the power by bringing out a trigger signal to one of the input/output pins on the PCB, and using this trigger to capture the voltage drop across the resistor with an oscilloscope. By computing the current through the resistor and multiplying that current by the power supply voltage we were able to get a precise estimate of the power draw by the entire board. We motivate the use of power numbers for the entire board by the fact that the OMAP chip by itself does not represent a complete system, while the purpose of our experiments is to make a system-level analysis.

We measured the power during the idle section of our program (with ARM executing a sleep() instruction), during ARM-only execution of point multiplication, and during ARM+DSP execution. The results are presented in Table VII.

We observe a 14% increase in the power draw of the device during active computation phase compared to idle phase. Using the DSP for the majority of computation, with the ARM waiting for DSP to reply, increases the power draw by 17% relative to idle. We did not attempt to use any kind of power saving methods on the ARM while it is waiting for the reply from the DSP. For instance, if we were to dynamically reduce ARM's clock frequency during the waiting period it may be possible to achieve lower overall power draw without affecting performance.

TABLE VIII
ENERGY COST OF AN EC POINT MULTIPLICATION

Curve	ARM (mJ)	ARM+DSP (mJ)	Energy Savings
Binary Field			
sect163r1	7.69	1.62	4.75
sect283r1	27.87	6.14	4.54
sect409r1	65.71	12.4	5.3
Prime Field			
secp160r1	6.92	2.26	3.06
secp224r1	12.42	4.66	2.67

Even with the increased power due to DSP utilization, we save considerable power with the ARM+DSP scheme once we take the execution time into account. Table VIII shows the energy cost of a single point multiplication for both designs. These numbers represent the energy consumed by the Beagle Board platform rather than the OMAP 3530 chip.

VI. CONCLUSION

Our objective in this research was to study heterogeneous multicore ECC design on a current-generation SoC platform. Our results show that there are clear and tangible performance and power efficiency benefits to heterogeneous design even after all the overhead and tradeoffs are taken into account. Our research shows that it is worthwhile exploring complex SoC platforms for cryptographic acceleration.

ACKNOWLEDGMENT

We thank the Security and Architecture lab at University of Connecticut for their input on the implementation of prime-field arithmetic on the OMAP 3530.

REFERENCES

[1] H. Yan, Z. Shi, and Y. Fei, "Efficient implementation of elliptic curve cryptography on dsp for underwater sensor networks," in *Proceedings of Workshop on Optimizations for DSP and Embedded Systems*, 2009. Online at http://www.imec.be/odes/.

[2] C. Tergino, "Efficient binary field multiplication on a vliw dsp," 2009. Master's Thesis Virginia Tech, ETD-06222009-150103.

[3] C. H. Gebotys and R. J. Gebotys, "Secure elliptic curve implementations: An analysis of resistance to power-attacks in a dsp processor," in *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, (London, UK), pp. 114–128, Springer-Verlag, 2003.

[4] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[5] D. Brown, "Sec 2: Recommended elliptic curve domain parameters," 2010. Certicom Research.

[6] N. F. 186-2, "Digital signature standard (dss)," 2000. National Institute of Standards and Technology.

[7] D. Bernstein and T. Lange, ""explicit-formulas database"." Online at http://www.hyperelliptic.org/EFD/.

[8] J. Sankaran, "Reed solomon decoder: Tms320c64x implementation." Digital Signal Processing Solutions Application Report, 2000.

[9] S. Haibin, Y. Jin, and Y. You, "Unbalanced exponent modular reduction over binary field and its implementation," in *Innovative Computing, Information and Control (ICICIC)*, pp. 190–193, 2006.

[10] T. Instruments, "Omap3530/25 applications processor (rev. f)," 2010. http://www.ti.com/lit/gpn/omap3530.

[11] J. López and R. Dahab, "Fast multiplication on elliptic curves over gf(2m) without precomputation," in *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '99, (London, UK), pp. 316–327, Springer-Verlag, 1999.