A Programmable Galois Field Processor for the Internet of Things

Yajing Chen Shengshuo Lu Cheng Fu David Blaauw Ronald Dreslinski Jr
Trevor Mudge Hun-Seok Kim
University of Michigan, Ann Arbor
yajchen@umich.edu,luss@umich.edu,fuch@umich.edu,blaauw@umich.edu,rdreslin@umich.edu
tnm@umich.edu,hunseok@umich.edu

ABSTRACT

This paper investigates the feasibility of a unified processor architecture to enable error coding flexibility and secure communication in low power Internet of Things (IoT) wireless networks. Error coding flexibility for wireless communication allows IoT applications to exploit the large tradeoff space in data rate, link distance and energy-efficiency. As a solution, we present a light-weight Galois Field (GF) processor to enable energy-efficient block coding and symmetric/asymmetric cryptography kernel processing for a wide range of GF sizes (2^m, $m = 2, 3, ..., 233$) and arbitrary irreducible polynomials. Program directed connections among primitive GF arithmetic units enable dynamically configured parallelism to efficiently perform either four-way SIMD 5- to 8-bit GF operations, including multiplicative inverse, or a wide bit-width (e.g., 32-bit) GF product in a single cycle. To illustrate our ideas, we synthesized our GF processor in a 28nm technology. Compared to a baseline software implementation optimized for a general purpose ARM M0+ processor, our processor exhibits a $5-20\times$ speedup for a range of error correction codes and symmetric/asymmetric cryptography applications. Additionally, our proposed GF processor consumes $431\mu W$ at $0.9V$ and $100MHz$, and achieves $35.5pJ/b$ energy efficiency while executing AES operations at $12.2Mbps$. We achieve this within an area of $0.01mm^2$.

CCS CONCEPTS

• **Computer systems organization** → **Special purpose systems**;

KEYWORDS

Domain Specific Hardware Acceleration, Internet of Things, Galois field, security and information coding

ACM Reference format:
Yajing Chen Shengshuo Lu Cheng Fu David Blaauw Ronald Dreslinski Jr Trevor Mudge Hun-Seok Kim University of Michigan, Ann Arbor . 2017. A Programmable Galois Field Processor for the Internet of Things. In *Proceedings of ISCA '17, Toronto, ON, Canada, June 24-28, 2017,* 14 pages.
https://doi.org/10.1145/3079856.3080227

1 INTRODUCTION

Envision the future Internet of Things. We envision a future world with a Trillion of IoT devices communicating with other (often heterogeneous) devices. The devices can be configured to address many standards, non-standard, custom specified communication schemes for various use-case scenarios. The communication between things will be enabled at anytime, anywhere among anything, from the Internet of Things (IoT) toward Internet of Everything (IoE).

Today, IoT networks are often fragmented, having nodes with different physical layer protocols such as Bluetooth Low Energy (BLE) [5] or IEEE802.15.4 [51]. In the future, direct connectivity should no longer be limited by a single protocol or one specific physical layer ASIC.

The value of flexible baseband processing has been demonstrated in previous works [9, 32, 57]. By dynamically changing configurations (signal bandwidth, modulation parameters, etc.) on the Software-Define Radio (SDR) platform, IoT devices can adjust the data rate to extend the operating distance with the same power budget. SDR can enable higher energy efficiency when the operating scenarios require shorter distance and/or lower data rates than that are defined in a particular wireless standard. Rapid innovation is enabled by the nature of SDR as new communication modulation schemes and protocols can be easily adopted with a simple software update. Enhanced spectral efficiency can be achieved via dynamic adaptations in frequency planning, pulse shaping, and bandwidth allocation.

Flexible radio solution is realizable without increasing the overall power consumption [9]. In fact, better energy efficiency can be achieved via operating the SDR at an energy-optimal configuration that might be outside the scope of a specific physical layer protocol [8, 24]. In this chapter, we augment the benefit obtained from SDR baseband signal processing flexibility by also adding flexibility to the information / error-correction coding in IoT communication.

1.1 Error Correction Coding Flexibility

Error correction coding is a very effective way to enhance reliability in IoT communications. The optimal energy efficiency, data rate, and link distance tradeoff can be obtained by adjusting the error correction coding rate and/or the information encoding schemes. There are many error correction codes that have been proposed for low power wireless. Today, each physical layer protocol specifies an error correction scheme that is optimized for a representative scenario (combination of packet length, data rate and link distance). For example, Bose-Chaudhuri-Hocquenghem (BCH) coding is the error correction method used in Body Area Networks [52], that allows 2 or 3-bit error correction in a 63-bit codeword. Reed Solomon (RS) codes also have been proposed for low power communication because they work within the hardware complexity restrictions for Low Data Rate Wireless Personal Networks (LDR WPAN) [13, 16].

A single fixed error correction code is suboptimal, given that baseband processing is migrating towards a more versatile SDR with flexible data rates, bandwidths and modulation schemes to satisfy heterogeneous IoT operating scenarios. The information coding flexibility, therefore, plays an important role in data-rate, distance, and energy-efficiency tradeoffs for IoT communications. Ideally, flexible coding should support a broad class of parameterized coding schemes for various standards or non-standard communications to adapt to different channel conditions. A low rate encoding with strong error correction capability should be applied in a noisy channel, while in a clean channel we should be able to pack more information bits in a codeword. A flexible coding scheme should also be able to address different error patterns, for example, be robust enough to handle both uniformly distributed and burst bit errors. Unlike convolutional, turbo, LDPC and polar codes that are widely used in high performance broadband access, low power IoT communications, with relatively low data rates and short packet lengths, do not employ a long codeword. In this work, we address a broad class of Galois Field (GF) based block coding schemes with a short ($< 100s$ bits) codeword as they are more common for low power IoT wireless connectivity. By varying codeword length n, information length k and GF size (2^m), various block codes are supported that can optimally adapt to different channel conditions and application scenarios.

1.2 Cryptography

While the proposed flexible error correction coding schemes provide robustness against noise and interference, IoT wireless communication that exchanges a plaintext through a shared medium is fundamentally insecure to malicious attacks that eavesdrop and impersonate in the middle of the network. Our proposed GF processor addresses the security aspect of the IoT communication by providing a unified architecture to perform not only error correction coding but also popular cryptography kernel functions computed in a finite GF.

Asymmetric Cryptography. Asymmetric cryptography such as Elliptic Curve Cryptography[1] (ECC_l) [22], is widely employed in an authentication process to exchange a shared private key. However, because of its extremely large GF (e.g., 2^{233}), ECC_l complexity is orders of magnitude higher than that of the symmetric cryptography process. Although, the asymmetric cryptography process is called only once when a new secure session is established, ECC_l software implementation to perform long bit length GF operations is very inefficient and typically incurs excessive latency when realized on a low power general purpose processors (analysis provided in Section 3.3.4). Because it is used infrequently, the area of an ECC_l solution is more critical than its energy efficiency provided it meets the latency requirement. Our proposed GF processor architecture provides an area efficient solution for ECC_l without incurring a significant overhead to the baseline architecture in order to support both Error Correction Codes (ECC_r) and ECC_l.

Symmetric Cryptography. Once a secure session is established through an ECC_l based key exchange process, symmetric cryptography using a shared private key is applied on a packet-by-packet basis to encrypt / decrypt a plaintext / ciphertext message pair. A widely used method for symmetric cryptography is the Advanced Encryption Standard (AES). Unlike asymmetric cryptography ECC_l, throughput of the symmetric cryptography should be matched to the data rate of the underlying physical layer.

Traditionally, the coding and security modules are implemented as dedicated accelerators [33, 35, 37] because they are computationally intensive and do not efficiently map onto a general purpose processor architecture. Moreover, due to lack of programmability in the legacy physical layer, coding flexibility has not been widely explored. With the move towards SDR, this opens a new research possibility on microarchitectures to enable coding flexibility and to inherently address the security aspect of IoT connectivity as well.

1.3 Feasibility to Address Coding Flexibility and Cryptography on the Same Piece of Hardware

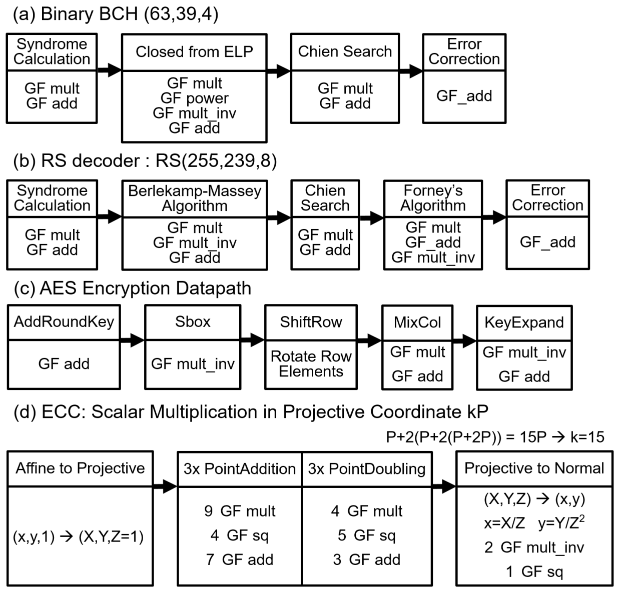

Figure 1: Dataflow for ECC_r, AES, and ECC_l. They share the same underlying GF arithmetic operations.

Coding flexibility allows IoT applications to exploit the large tradeoff space in data rate, link distance and energy-efficiency. However, realizing coding flexibility on a general purpose processor is too inefficient, and it easily leads to 100× worse energy efficiency compared to dedicated hardware accelerators, offsetting the benefit of flexible coding. On the other hand, employing multiple dedicated accelerators is expensive in terms of design effort and production cost. Addressing this challenge, we propose a unified architecture that is area- and energy-efficient to support not only flexible information coding but also secure wireless communication via asymmetric and symmetric cryptography.

[1]In this paper, we use ECC_l to refer Elliptic Curve Cryptography and ECC_r to refer Error Correction Code.

To demonstrate feasibility of the proposed solution, we first briefly describe the datapath involved in example error correction coding and cryptography processes. Fig. 1 (a) show the datapath of the binary BCH code [42, 56], which has three coding parameters n, k, t; n represents the codeword bit length, k represents the information bit length and t indicates the number of bit errors correctable within a codeword. A binary BCH code n, k, t is constructed from a Galois field GF(2^m), where $n = 2^m - 1$. The decoding datapath of binary BCH consists of four kernels. The first kernel—Syndrome Calculation—evaluate the received codeword. Non-zero syndromes indicate that errors exist in the received codeword. The kernel—Closed Form ELP [25]—solves the coefficients of the error locator polynomial. Finally, the Chien search finds the roots of the error locator polynomial, and the roots reveal the error locations. Bit errors at these locations are corrected via GF additions. The BCH code is favored by many low power devices [39, 46, 47] for its concise decoding process and effectiveness in correcting uniformly distributed random errors.

Reed Solomon (RS) codes [27, 56] also have n, k, t parameters and operate on GF(2^m), where $n = 2^m - 1$. Unlike binary BCH, in an RS code, n is the number of symbols in a codeword, where each symbol has m bits. k and t are the number of information symbols and the number of correctable error symbols in a codeword, respectively. The decoding datapath is very similar to binary BCH. The Berlekamp-Massey-Algorithm(BMA) is a generalized method to solve the coefficients of the error location polynomial. RS decoding requires an additional kernel—Forney Algorithm—to evaluate the actual m-bit error values after the error symbols are located via the Chien search algorithm. RS codes are suitable for correcting multiple-burst bit errors (up to m error bits within a symbol). A generalized RS decoder datapath is shown in Fig. 1 (b).

A variety of BCH and RS codes can be created with different coding rates (k/n), GF sizes (2^m) and the error correcting ability represented by the parameter t. As Fig. 1 (a) and (b) indicate, BCH and RS codes share an identical datapath at a high level. One of technical challenges involved in flexible block coding is efficient handling of various GF sizes (2^m) and irreducible polynomials associated with each GF [12]. Dedicated hardware accelerators address this issue by designing a hard-wired GF arithmetic unit that is specific to a given GF size and an irreducible polynomial pair.

The datapaths of AES and ECC$_l$ shown in Fig. 1 (c) and (d) respectively, which are distinct from that of the error correction codes. Our proposed architecture exploits the fact that each step involved in AES and ECC$_l$ can be converted to equivalent finite Galois field arithmetic operations. For example, the S-box operation in AES is equivalent to GF(2^8) inverse operation [7]. Similarly, the point addition operation in ECC$_l$ can be mapped to a series of GF multiplications, additions, and division in GF(2^m), where typically $m > 100$.

Fig. 1 (c) illustrates the AES encryption datapath for a single round of processing. Different key lengths (128, 192 or 256-bit) determine the number of rounds for a single encryption. Decryption is very similar to encryption except that the kernels process in a reverse direction with different constants to implement inverse.

The ECC$_l$ datapath is shown in Fig. 1 (d). The key operation in ECC$_l$ is scalar multiplication (kP) performed on an elliptic curve [30, 34], where k is the scalar and $P(x_p,y_p)$ is a point on the elliptic curve. The scalar multiplication is further decomposed into two operations–point addition (P_1+P_2) and point doubling ($2P$). Both point addition and point doubling are computed via several GF multiplication, square and multiplicative inverse operations. Depending on the ratio between multiplication and multiplicative inverse, transforming points to a different coordinate (e.g., the projective coordinate) may be necessary to reduce the complexity.

We make an important observation that all coding and cryptography processes under consideration (binary BCH, RS, AES and ECC$_l$) were derived from GF mathematics, therefore they share the same underlying operations. However, their GF sizes and irreducible polynomials can be very different. Addressing this challenge leads to our proposed GF processor architecture described in Section 2. We propose to implement the Galois field arithmetic unit as a dedicated GF functional unit of the proposed processor to provide efficient computation of complicated GF operations.

1.4 Contribution

To address generality in both coding and cryptography, we need to support arithmetic flexibility in different bit-widths and arbitrary irreducible polynomials associated with each Galois field. Meanwhile, the cost of arithmetic flexibility needs to be affordable because IoT devices are limited by strict power budget and area constraints. The contributions of this work are summarized as follows:

- We analyze several error correction coding schemes and cryptography kernels to identify the shared fundamental operations.
- We propose a flexible bit-width Galois field arithmetic unit consisting of two types of primitive arithmetic units: multiplication and square.
- These primitives are combined to support single cycle complicated instructions: multiplicative inverses and long bit-width GF products using 32-bit segmentation.
- We exploit the parallelism in coding and cryptography datapaths, and employ (configurable) four-way 8-bit SIMD operations to improve performance and functional unit reuse. The SIMD computation datapath is shared among the multiplicative inverse and wide bit-width GF product instructions.
- We illustrate how to efficiently handle extremely wide bit-width ($>$ 100bit) multiplication by iteratively using the SIMD-optimized single-cycle 32-bit products.
- The GF arithmetic unit is integrated into a two-stage in-order processor. Several coding kernels and cryptography kernels are evaluated using the proposed processor architecture.

We demonstrate the feasibility of a unified processor architecture to enable coding flexibility and secure communication in low power IoT wireless networks. The proposed processor enhances wireless communication energy-efficiency by adapting the coding scheme to various noise/interference conditions. At the same time, secure communication is realized via symmetric and asymmetric cryptography processes that are also efficiently implemented on the proposed processor.

2 THE PROGRAMMABLE LIGHT WEIGHT GALOIS FIELD PROCESSOR

2.1 Processor Architecture

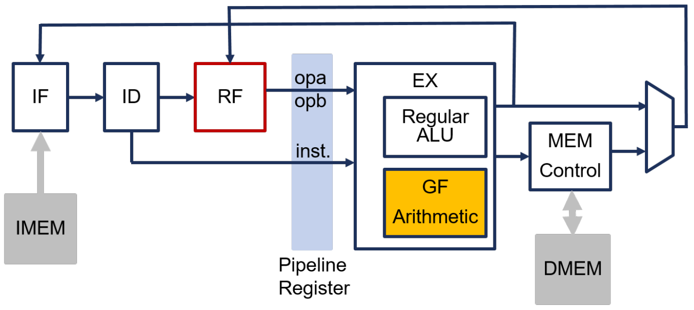

Figure 2: Proposed Galois Field Processor

The system architecture of the programmable Galois field processor is illustrated in Fig. 2. Control related computation, integer arithmetic operations and memory operations are conducted by regular functional units. Rather than implementing the full instruction set of a Cortex M0+, we profile the workloads and identify the subset of control instructions, integer arithmetic operations and memory operations needed. We only implement this subset to conserve area. We use a 32-bit datapath, and 16 entry 32-bit register file. We add a Galois field arithmetic unit and associated instructions to optimize GF operations. All SIMD GF instructions, including multiplication, square/power, multiplicative inverse, and a 32-bit partial product are single cycle instructions. Arbitrary irreducible polynomials with degree smaller or equal to 8-bit are supported with the dedicated configuration register (inside the GF Arithmetic Unit). The microarchitecture of GF arithmetic unit is discussed in Section 2.2 - 2.4.

Our instruction set architecture (ISA) is a combination of GF instructions and a subset of Cortex M0+ instructions, which conduct all non-GF instructions including control-related, memory and integer arithmetic operations. The GF instructions have a 26-bit format, consisting of a 10-bit opcode and a 16-bit register field. Their operation is illustrated in Table 1. The GF arithmetic instructions support a complete set of GF arithmetic operations.

2.2 The Galois Field Arithmetic Unit

In this design, we restrict our design to a binary Galois field (2^m), where m is the bit width to represent an element in the Galois field. In this arithmetic, m-bit inputs produce m-bit output without a carry. Addition and subtraction are identical; they are implemented via bitwise exclusive-OR. Multiplication is performed as multiplication modulo an irreducible polynomial. Since a GF(2^m) can have many irreducible polynomials, the selection of the irreducible polynomial will affect the multiplication/multiplicative inverse operations. Flexibility in GF arithmetic implies that we need to address various bit-widths and different irreducible polynomials.

2.3 Design Challenges

In this section, we identify design challenges that are specific for a flexible Galois Field arithmetic unit.

Format	Description	Operations
Category-I: SIMD Instr. Each RF holds four 8-bit GF values. All operations are in Galois field.		
gfMult_simd R_{s1},R_{s2}, R_d	Multiplication	$R_{s1} \otimes R_{s2} \to R_d$
gfMultInv_simd R_s, R_d	Multiplication Inverse	$R_s^{-1} \to R_d$
gfSq_simd R_s, R_d	Square	$R_s^2 \to R_d$
gfPower_simd R_{s1},R_{s2}, R_d	Power	$R_{s1}^{R_{s2}} \to R_d$
gfAdd_simd R_{s1},R_{s2}, R_d	Addition	$R_{s1} \oplus R_{s2} \to R_d$
Category-II: single cycle 32-bit Partial Product		
gf32bMult R_{s1},R_{s2}, R_d^h,R_d^l	32-bit Carryfree Multiplier	$R_{s1} \times R_{s2} \to R_d^h, R_d^l$
Category-III: Configuration		
gfConfig #Address	Load 56-bit coef. to field config. register	$*Address \to R_{config}$

Table 1: Galois Field Instructions.

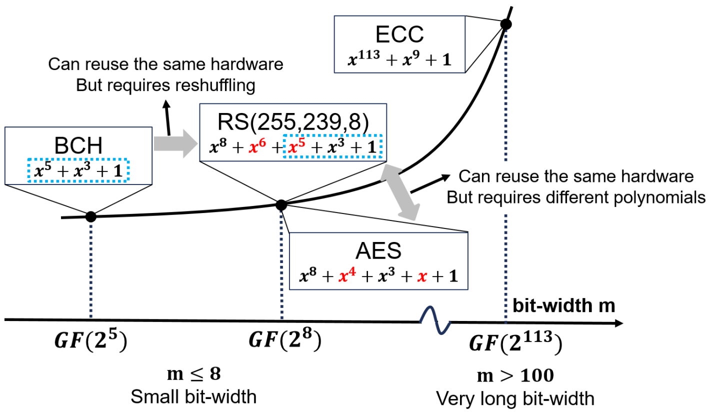
ECC
$x^{113} + x^9 + 1$
Can reuse the same hardware But requires reshuffling
BCH
$x^5 + x^3 + 1$
RS(255,239,8)
$x^8 + x^6 + x^5 + x^3 + 1$
Can reuse the same hardware But requires different polynomials
AES
$x^8 + x^4 + x^3 + x + 1$
bit-width m
$GF(2^5)$
$GF(2^8)$
$GF(2^{113})$
$m \leq 8$
Small bit-width
$m > 100$
Very long bit-width

Figure 3: GF Arithmetic Design Challenge. To support a wide range of bitwidths and arbitrary polynomials.

Most of the error correction codes and AES only require small bit-widths to represent a GF element, specifically $m \leq 8$ as shown on the left side of Fig. 3. In contrast, asymmetric cryptography, ECC_l on the right side of Fig. 3, requires very long bit-width fields. Several standard binary curves are recommended for ECC_l [6, 43], the smallest being 113 bits and the largest being 571 bits. Although these two requirements differ widely in bit width range, we show that they can be implemented efficiently with the same underlying hardware.

Challenges arise because of the fact that, in contrast to integer arithmetic operations, a smaller GF bit-width ($m \leq 8$) operation cannot directly use a larger GF bit-width datapath by simply setting the most significant bits to zeros. Galois fields with distinct sizes have completely different irreducible polynomials in terms of both degrees and coefficients. Setting the most significant bits to zeros does not work even if the smaller bit-width irreducible polynomial may be a subset of the larger bit width irreducible polynomial. For example, BCH(31,26,1) on GF(2^5) is a subset of RS(255,239,8) on GF(2^8), as shown in Fig. 3. However, even in such cases, setting the most significant 3 bits to zero will not work because Galois field requires a polynomial modulo operation. Fig.5 (b) will illustrate this with an example.

Even the same bit-width datapaths cannot directly share the same multiplication and multiplicative inverse units. For example, RS(255,239,8) and AES both in GF(2^8) cannot use the same multiplication and multiplicative inverse units becasue of the different coefficients in their irreducible polynomials. All in all, specific hardware has to be designed to support variable bit-width GF operations.

2.4 The GF Arithmetic Unit Microarchitecture

The Galois field arithmetic unit microarchitecture is illustrated in Fig. 4. It is composed of several basic units, including 16 8-bit GF multiplication units and 28 8-bit GF square units. It also includes one dedicated configuration register to support arbitrary irreducible polynomial and control logic to selectively connect basic units to execute different instructions.

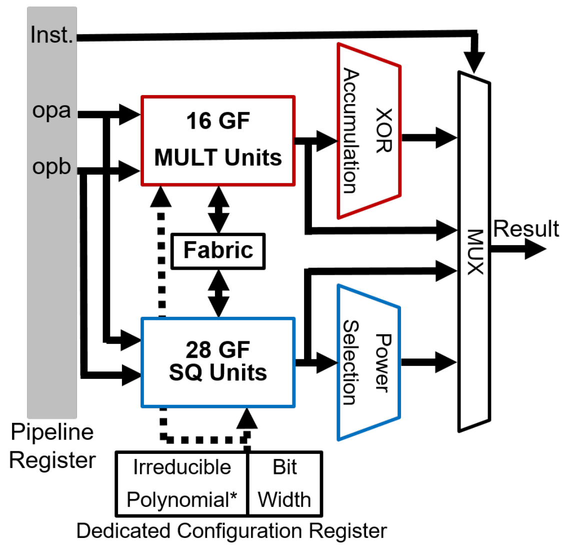

Figure 4: Galois Field Arithmetic Unit. It supports SIMD instructions: multiplication, square, and multiplicative inverse, as well as 32-bit partial products. It has a dedicated configuration register shared among the ALUs. The pipeline register is shared between GF arithmetic units and the regular arithmetic logic units.

2.4.1 *Primitive Units.*

In this design, we employ two primitive computation units: multiplication and square.

Multiplication. We employ the compact GF multiplier method introduced in [53]. It decomposes multiplication into a carryless multiplier and a polynomial reduction module. The multiplier first computes a $(2m-1)$ bit full product in GF(2) from two m-bit inputs, Fig. 5 illustrates this. The polynomial reduction module performs $modulo(\mathbf{c},\mathbf{r})$, where $\mathbf{c}$ is the carryless multiplier output and $\mathbf{r}$ is the vector representation of the irreducible polynomial.

The polynomial reduction module is implemented as a linear transformation by implementing a GF(2) matrix vector multiplication. The reduction is done using a reduction matrix ($\mathbf{P}$), which is derived by a transformation of the irreducible polynomial ($\mathbf{r} \rightarrow \mathbf{P}$), and can be determined a priori. The reduction matrix is programmable by writing the values of $\mathbf{P}$ to the dedicated configuration register.

The default Galois field operation in our datapath is GF(2^8)–an 8-bit datapath. The polynomial reduction circuits include an 8-by-7 matrix and a reduction vector of 7-bits as outlined by the green dashed box. The remaining vector shown by the red dashed box fills out the datapath bit width.

We deploy a new method, which selectively maps the full product $\mathbf{c}$ to the polynomial reduction module based on a GF size dependent pattern. The bit-width signal from the configuration register directs the mapping circuit to split the full product $\mathbf{c}$ into the reduction vector (green dashed box in Fig. 5) and the remaining vector (red dashed box in Fig. 5).

The example of operating on a smaller bitwidth is illustrated in Fig. 5 (b). This is done by first setting the most significant bits to zeros and then changing the mapping of $\mathbf{c}$ to the polynomial reduction step. The mapping of $\mathbf{c}$ depends on the GF size, with each size having a unique mapping. As mentioned in Section 2.3, if only the significant bits were set to zero without updating the mapping, the c_2 bit in the partial product would be mapped to the wrong position. Our hardware contains a configuration register which specifies how the mapping of partial products to the polynomial reduction should occur, enabling us to support smaller bit-widths. As a result, we reuse the same size computation resource in the polynomial reduction module. The control overhead to support 5-, 6-, and 7-bit using the 8-bit computation unit is small: 8% of the entire arithmetic units.

Our method is in contrast to prior approaches for supporting smaller bit-widths [53, 54], which instead increase the programmable part of the reduction matrix-vector operation. An increased size of the reduction matrix would require more computation resources. To support 5-, 6-, and 7-bit multiplication, one option [54] is to add a 5-by-3 matrix vector operation, incurring more than 26% additional hardware overhead. The other option [53] is to include a triangular matrix, which requires more control when mapping the full product to the linear transformation and several additional computations.

We compare our implementation with another configurable GF multiplication—the popular systolic solution [19, 20, 45]. The multiplication resource comparison with a LSB method in [20] is shown in Table 2.

The difference in combinational logic between the two methods is small, because it is determined mostly by the computational complexity. The systolic method processes one bit polynomial multiplication and reduction at a time. To achieve the same throughput, one sample per cycle, a bit-level pipeline method is required, which results in a very short critical path at the cost of significantly more area and power. As we target the IoT application domain which does not require a very high throughput, our proposed design is a preferred choice. We support wide bit-width products (> 8-bits) by connecting multiple basic multiplication units together. This will be elaborated on in Section 2.4.3.

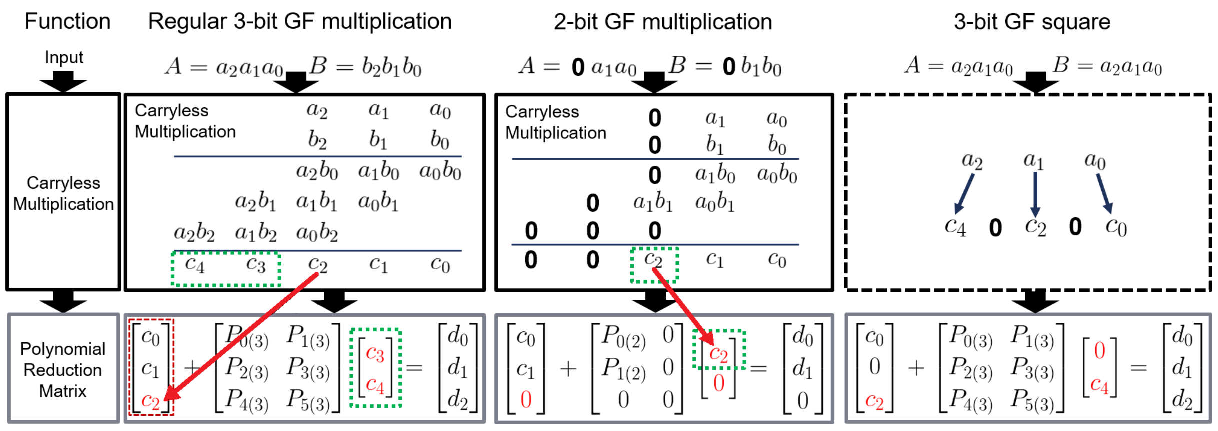

Figure 5: GF Multiplication Unit. Illustrating regular multiplication, smaller bitwidth multiplication and squaring.

		Systolic	This work
Polynomial Modulo		Bit-pipelined	Single Step Linear Transform
Comb.	AND	$2m^2$	$2m^2 - m$
	XOR	$2m^2$	$2m^2 - 3m + 1$
FF	opa	$(m-1)m$	Pure combinational logic
	opb	$(m-1)m/2$	
	intermediate	$(m-1)m$	
Total area*		$16.5m^2 - 10m$	$6.5m^2 - 7.75m$
Configuration Datapath (shared by multiple ALUs)			
FF	p or S	m	$m(m-1)$
*AND : MUX : XOR : FF = 1 : 2.25 : 2.25 : 4 in a 28nm technology			

Table 2: Resource Comparison for Different Multiplication Methods

28nm	GF mult	GF square
m=5,6,7,8 ; arbitrary polynomial		
# of cells	263	73
Area (um^2)	199.59	63.48
Critical Path (ns)	0.4	0.2
Configuration of GF Arithmetic Unit		
# of primitive arithmetic units	16	28

Table 3: Comparison between Multiplication and Square

Square. The square operation is the other basic unit in our GF arithmetic unit. Although it is a specialized multiplication, we still implement it as a separate unit for two reasons. Firstly, square is much simpler than multiplication in Galois field. Mathematically, the full product of a square only spreads the input and inserts zeros in the odd positions as shown in Fig. 5 (c). Thus, a square operation only needs a polynomial reduction module. The hardware comparison between multiplication and square is illustrated in Table 3. Secondly, square is a heavily utilized operation to realize the complicated multiplicative inverse instruction. We will elaborate on this in Section 2.4.3.

Our preferred design includes 16 GF multiplication units and 28 GF square units. The four-way SIMD multiplicative inverse and 32-bit partial product both require 16 GF multiplication units for single cycle operations. 28 GF square units are needed to support a single cycle SIMD multiplicative inverse. More multiplication units would benefit elliptic curve cryptography applications, but result in more idle hardware for error correction codes due to limited parallelism (discussed in Section 3.1).

2.4.2 Centralized Dedicated Configuration Register.

We employ a dedicated centralized configuration register to support various bit-widths and arbitrary polynomials. It is set via a special configuration instruction when a Galois field is decided. Since it is shared among all arithmetic units, the overhead of supporting variable bit-widths and arbitrary polynomials are amortized.

The centralized configuration register is also used in data-gating almost half of the combinational logic when the 32-bit partial product instruction is executed, resulting in a 33% power reduction.

2.4.3 Complicated Instructions.

Complicated instructions such as multiplicative inverse, 32-bit partial product, and SIMD instructions are realized by different connections among the primitive units in the interconnect fabric shown in Fig. 4. Computation units are reused among different instructions, reducing area overhead.

Multiplicative Inverse. In Galois field, multiplicative inverse can be computed by $\alpha^{-1} = \alpha^{2^m-2}$, where α is an element in GF(2^8). This can lead to a large power depending on m. To reduce the number of power computations involved in this process, we propose using the Itoh-Tsujii algorithm (ITA) introduced in [15]. The multiplicative inverse instruction is implemented by connecting a series of multiplications and squares. Four multiplications and seven squares are concatenated to realize one 8-bit multiplicative inverse. Fig. 6 presents an example for multiplicative inverse in GF(2^4). Multiplicative inverse over GF(2^3) is realized by muxing out the A^6 power.

We also considered other systolic methods [26, 31, 58], based on Euclid Algorithms (EA), which implement multiplicative inverse at a higher throughput. The hardware resource comparison between EA [26] and our method is illustrated in Table 4. We implemented the ITA method for two reasons: first, we are not targeting a high

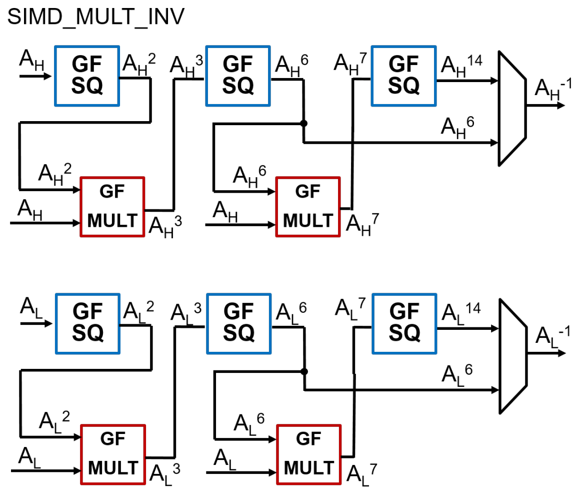

Figure 6: Multiplicative Inverse. Interconnecting primitive units to perform the multiplicative inverse.

	Systolic Euclidean Algorithm (pipelined)	This work ITA
Comb.	$m(6m+3)XOR+$ $m(6m+7)AND+$ $m(6m+5)MUX$	$(15m^2-11m)AND+$ $(15m^2-13m+4)XOR$
FF	$m(6m+4)$	No need
Total area* **	$57m^2$	$48.75m^2$
Configuration Datapath (shared by multiple ALUs)		
FF	m	$m(m-1)$
* AND : MUX : XOR : FF = 1 : 2.25 : 2.25 : 4 in a 28nm technology **only m^2 item is considered, which overestimate the area of our work		

Table 4: Resource Comparison for Different Multiplicative Inverse Methods

throughput application domain; second, the ITA method can use existing multiplication/square hardware, requiring no extra area.

Single Cycle 32-bit Partial Product. A single cycle 32-bit partial product utilizes all 16 of the 8-bit GF multiplication units and a simple GF(2) addition (exclusive-OR). An example of 16-bit partial product is illustrated in Fig. 7. Very wide bit-width (>100bit) GF multiplication benefits from this single cycle 32-bit partial product by iteratively using this product and performing a reduction step on the CPU. An example of efficiently supporting GF(2^{233}) multiplication is detailed in Section 3.3.4.

SIMD Instruction. We support SIMD instructions for multiplicative inverse (Fig. 6), multiplication and square (Fig. 8). Since there are 28 square units, the even-power instructions can be carried out in SIMD fashion with a few muxes, as shown in Fig. 8 (top). We implement a four-lane 8-bit SIMD datapath for two reasons. First, the gain of a wider SIMD datapath is negligible because of limited parallelism in the applications, which will be discussed in Section 3.1. Second, we want to match the number of 8-bit GF multiplier involved in a 32-bit partial product and a SIMD multiplicative inverse. Both a 32-bit partial product and a four-way 8-bit SIMD multiplicative inverse can be realized with 16 8-bit full multipliers. Hence, a majority of the computation resources are reused by different instructions.

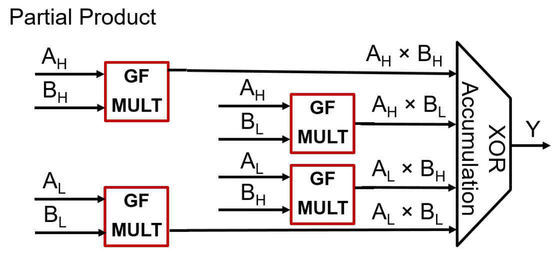

Figure 7: 16-bit Partial Product Example. The 32-bit case follows an analogous interconnection of primitive units.

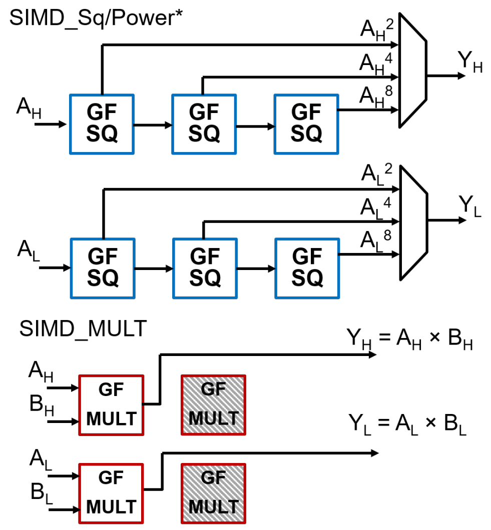

Figure 8: 2-way SIMD GF Multiplication and Square/Power Example. The 4-way case follows an analogous interconnection of primitive units.

Data Gating of Idle GF Arithmetic Unit. The Galois field arithmetic unit is not always active. The GF instructions are interleaved with other instructions such as integer arithmetic and control instructions. Therefore, data gating is applied on the Galois field arithmetic unit by feeding zeros as default inputs, so the updating on the pipeline register will not incur switching activities in the GF arithmetic unit, resulting in a 77% dynamic power savings.

3 EVALUATION

3.1 Application Kernels

Table 5 illustrates the kernels we are evaluating for both cryptography and coding applications—here coding refers to the decoding process, while encoding is also feasible with the proposed architecture. We select the most common algorithms that works for all different types of RS/BCH codes. Even without any modification, the coding algorithms and AES reveal some degrees of parallelism [27]. We employ a four way SIMD to take advantage of this.

3.2 Kernel Mapping

Kernel mapping is performed by converting each step in ECC$_r$, AES, and ECC$_l$ to a series of GF arithmetic operations and datapath control instructions.

The rest of this subsection is an example for syndrome calculation in the RS/BCH decoder datapath. This is the first and the only kernel that cannot be avoided in the decoding datapath. Because the decision whether to perform the rest of the decoding datapath relies on the value of syndromes. If syndromes are all zero, this indicates that no error occurred and the decoding will terminate.

As illustrated in Table 5, there are $2t$ independent syndromes to compute for t error correcting ability. Syndrome computation is parallelized in a straightforward manner. Each syndrome S_i is typically computed using Horner's rule via the recursive calculation in the form of $S_{i,j} = S_{i,j-1}\alpha^i + R_{n-j}$ for $j = 1, 2, ..., n$ where $S_{i,j}$ is the computed syndrome after $j-$rounds of recursion, α is a primitive element in Galois field, and R_{n-j} is the received symbol in the codeword. $S_i = S_{i,n}$ holds when n is the number of symbols in a codeword. The inner loop for syndrome calculation requires one GF multiplication and one GF addition. On a general purpose processor, the multiplication is typically optimized by performing the calculation in the log domain [38] that requires table lookup operations, as shown in left side of Table 6. With our architecture, the GF operations will directly call the corresponding instruction, as shown in the right side of Table 6.

The left column of Table 6 indicates that each inner loop of syndrome calculation involves one integer, one modulo, one bitwise exclusive-OR, and two multi-cycle table lookup operations. In contrast, in our architecture, the inner loop involves just two single cycle GF operations.

Because the inner loop computation on our architecture avoids table lookup operations, we also achieve reduced data memory footprint. Furthermore, because we have less variables required for each inner loop, our register files have less pressure. Finally, the $2t$ syndrome computations can be vectorized in a straightforward manner, as discussed in the beginning of this section. The cumulative effect of these improvements results in an over 20× speedup.

3.3 Performance

3.3.1 Methodology.

The baseline for our evaluation is obtained by running the kernels from Table 5 in Keil uVision 5, an ARM integrated development environment. It includes both a compiler and cycle accurate simulator , in which we can observe the executed assembly code. The target platform is a Cortex M0+, a two-stage in-order processor [1]. To obtain performance results for our architecture, we take the assembly code that performs a Galois field operation, for example the left column in Table 6, and replace it by our GF instructions. The control code and other non-GF arithmetic operations are kept the same. The reduced register pressure allows us to hold more data elements in the unoccupied registers. The cycles to compute each GF instruction using our architecture are deterministic and added to the simulated runtime.

3.3.2 BCH/RS Decoder.

We implemented a decoder consisting of syndrome calculation, Berlekamp-Massey-algorithm (BMA), Chien search and Forney's algorithm. The baseline implementation on the M0+ is optimized by conducting GF multiplication / multiplicative inverse in the log domain [38].

On our architecture, we do not need to operate in the log domain because all GF operations are directly performed on the GF arithmetic unit. A binary BCH (31,11,5) decoder is realized by combining the first three kernels and setting $n = 31$, $k = 11$, and $t = 5$ in the algorithm. An RS (255,239,8) decoder is realized by utilizing all four kernels (syndrome calculation, BMA, Chien search, Forney's algorithm) and setting the parameters to $n = 255$, $k = 239$, and $t = 8$. The speedup is shown in Fig. 9.

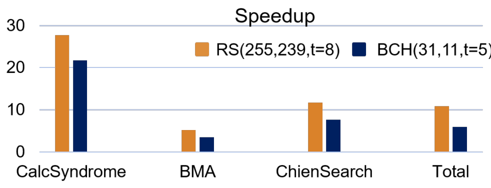

Figure 9: ECC$_r$ Decoder Speedup over M0+. Specifically comparing against RS on GF(256) and BCH on GF(32).

Computation dominant kernels, such as syndrome calculation, are very efficient on the proposed architecture. RS (255, 239, 8) has a better speedup compared to BCH (31,11,5) because RS ($t = 8$) has 16 independent syndromes, a perfect match for our 4-way SIMD, while BCH ($t = 5$), with 10 independent syndromes, looses two lanes in the last round.

BMA experiences the least speedup because it is an iterative algorithm with limited parallelism in the inner loop. In this evaluation, we only exploit the most straightforward parallelism in computing several intermediate t-degree polynomial coefficients. As discussed in Table 5, Chien search has many independent elements to evaluate. But for each element, the computation complexity is proportional to the degree t. Comparing it to the complexity n ($> t$) of syndrome computation, the control overhead of Chien search is larger.

Forney's algorithm for RS codes has over a 10× speedup. Forney's algorithm evaluates each error location using the GF multiplicative inverse and multiplication operations, which all map to single cycle operations on our architecture. We are able to calculate four independent errors in parallel.

The overall speedup for RS is more than 10×, which is better than the binary BCH decoder speedup. Forney's algorithm is not required in the binary BCH decoder because the error is binary. However, the binary BCH exposes another level of parallelism—inter-codeword parallelism. One packet contains multiple shorter codewords that can be decoded independently. Potentially, the inter-codeword parallelism embedded in binary BCH could lead to a higher performance gain if we had attempted to optimize for it.

Application	Kernel	Function Description	Parallelism Description
RS/BCH Decoder (n,k,t) $n=2^m-1$ GF(2^m)	Syndrome Computation	Reflect the errors of the received message	Explicit vectorizable with 2t independent syndromes. Each syndrome can be computed in parallel. Each requires n-steps of recursive computation.
	Berlekamp-Massey Algorithm	Find the coefficients of error locator polynomial (ELP) $\Lambda x = 1+\Lambda_1 x+\cdots\Lambda_t x^t$	Small and implicit parallelism. 2t steps of iterative computation to solve each coefficient of t-degree ELP. Dependency among coefficients limits parallelism.
	Chien Search	Find the root of ELP, which indicates the exact location of errors	Explicit vectorizable with 2^m independent elements to evaluate the t-degree ELP.
	Forney Algorithm	Calculate the error values at the error locations	Parallelism depends on the number of errors. Vectorizable with the computation of errors.
AES	AddRoundKey	Combine the state of subkeys (in this round)	Vectorizable with 16 independent state bytes.
	Sbox	Multiplicative inverse of each state bytes	
	ShiftRow	Scramble along the row	Nonvectorizable data movement
	MixColumn	Scramble along the column	Vectorizable with 4 of 4-by-4 matrix vector operation
	Key Expansion	Generate the next subkey	Vectorizable with 4 (a row). Sbox the first row, then xor with all other rows.
ECC$_l$	GF(2^m) multiplication	Fundamental operations of point addition/doubling over binary GF(2^m) NIST standard curves have $m > 100$ with sparse irreducible polynomials. Example: NIST Koblitz curve GF(2^{233}) with $x^{233}+x^{74}+1$	Utilize the single cycle 32-bit partial product. Benefit from sparse polynomial reduction
	GF(2^m) Squaring		

Table 5: ECC$_r$, AES, and ECC$_l$ Kernel Algorithms with Levels of Parallelism.

General Purpose Processor	This work
For j = 1,2,...,N	
$sumIdx = BIN2Idx[sum]$ $sumIdx = (sumIdx+i)\%fieldsize$ $sum = Idx2BIN[sumIdx]\oplus R_j$	$sum = sum \otimes \alpha^i$ $sum = sum \oplus R_j$

Table 6: Syndrome Computation Comparison between Embedded GPP and Our Architecture

3.3.3 *Symmetric Cryptography: AES.*

Several open source benchmarks for AES [44] [3] [4] are compared. The implementation in [44] achieves the best performance on the baseline ARM M0+ platform.

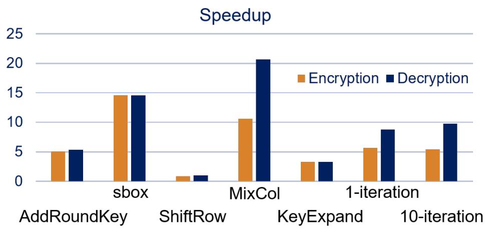

Figure 10: AES Speedup over M0+

Across all the kernels, S-Box and MixCol/invMixCol show the best speedup. Traditionally, S-Box is implemented as a table lookup operation while in our architecture it is realized directly with the multiplicative inverse operation. Similarly, MixCol/invMixCol in our architecture are also realized directly by performing inner products with Galois field arithmetic. The MixCol and invMixCol can use the same code but with different coefficients.

We profiled these open source benchmarks, and observed that MixCol/invMixCol is the hotspot. The selected baseline benchmark on the ARM M0+ obtains the best performance mostly because of good MixCol/invMixCol optimization. MixCol shows good optimization because the coefficients for MixCol are 0x02, 0x03, 0x01, and 0x01. GF multiplication with these values can be optimized into a few exclusive-ORs and shifts. While for invMixCol, the coefficients are 0x11, 0x13, 0x09, and 0x14. As a result, the data dependent optimizations on the ARM M0+ are not as efficient as MixCol. Our system obtains a more than a 10× speedup on MixCol, and, because our system is agnostic to the values of the coefficients, we achieve an even higher 20× speedup on invMixCol kernel. Overall, gains of more than 5× for encryption and more than 10× for decryption are achieved, mostly from improvements in MixCol/invMixCol and S-Box.

3.3.4 *Asymmetric Cryptography.*

Multiplication/Squaring on GF(2^{233}). Very wide bit width (e.g. 233-bit) GF multiplication and square are two essential operations for ECC$_l$. The literature states that GF multiplication typically accounts for most of the execution time [50], approximately 80% [10, 18]. We utilize the same ITA method described in Section 2.4.3 to

Operation	Load (LD)	Store (ST)	GF 32b Partial Product	ALUs	Total Cycle
233-bit multiplication					
Full Product	72	71	64	112	462
Rearrange	8	–	–	29	45
Polynomial Reduction	8	8	–	60	92
Total Operation	88	79	64	201	–
Total Cycle	334		64	201	599
233-bit squaring					
High Full Product & Rearrange	5	2	5	30	49
Low Full Product & Rearrange	5	8	3	58	87
Total Operation	10	10	8	88	–
Total Cycle	40		8	88	136
* ALUs include bitwise ops such as AND, SHIFT, XOR. ** GF addition is identical to BITXOR, sorted in ALUs. *** LD/ST has 2 cycles; all other operations are single cycle.					

Table 7: Operation Cycle Count Breakdown of ECC$_l$ GF(2^{233}) Multiplication/Squaring on NIST Koblitz Curve $x^{233}+x^{74}+1$

	Erdem [14]		Clercq [11]	This work
Platform	ARM7TDMI		Cortex M0+	2-stage pipeline in-order proc. with GF unit
GF(*)	2^{228}	2^{256}	2^{233}	
Mult.	4359	5398	3672	599
Square	348	389	395	136

Table 8: ECC$_l$ GF Multiplication/Squaring on Different Platforms

realize the multiplicative inverse operation. We hand-code these operations on GF(2^{233}) with a NIST Koblitz curve $x^{233}+x^{74}+1$ [43].

The multiplication on GF(2^{233}) is achieved by performing two steps. First, the full partial product of two 233-bit inputs (8 words with 32bits/word) is computed and stored back to memory (16 words) after this procedure.

The second step is to perform the polynomial reduction on the partial product in the CPU. We rearrange the partial product into the reduction vector and the remaining vector. We place the lowest 233 bits (c_0-c_{232}) into the first 8 words of memory and pad it with 23 zeros. The remaining bits (c_{233}-c_{464}) are padded with 24 zeros to create the reduction vector in the last 8 words of memory. The polynomial reduction operates on one 32-bit word at a time with only a few shift and exclusive-OR operations. The CPU cycles required for reduction is relatively low because the standard Koblitz polynomial has only a few non-zero coefficients. The cycle breakdown of a 233-bit multiplication for ECC$_l$ is illustrated in Table 7.

The square operation follows the same procedure as multiplication. We also leverage the simple structure of the square discussed in Section 2.4.1. Only 8 32-bit partial products are necessary to achieve a square partial product. Memory operations to store intermediate results are largely avoided. We interleave the full partial product operations and then rearrange results together for square. The cycle breakdown of a 233-bit square operation is presented in Table 7.

Table 8 illustrates the performance comparison to other works. We achieve a 6.1× speedup on multiplication and a 2.9× speedup on squaring for the same GF(2^{233}) field. Typical software optimizations involve pre-computed tables in memory, incurring a large storage overhead, which is undesirable for low power devices. For example, at least 4KB table storage is required in [11]. In our implementation, table storage is entirely avoided.

GF(2^{233}) Multiplication with Karatsuba Software optimization. We provide architectural augmentation on the arithmetic level for Galois field operations, which is orthogonal to register scheduling optimizations or other pure software optimizations. We have applied the Karatsuba algorithm [23] as a pure software optimization to compute the product of two large numbers. In contrast to the direct product of two w-bit numbers which requires four $w/2$-bit multiplications and four $w/2$-bit additions, the Karatsuba algorithm employs three $w/2$-bit multiplications and six $w/2$-bit additions. Performance improves when the cost of one multiplication is greater than the cost of two additions. In this work, we employ a two-level Karatsuba algorithm to perform the product of two GF(2^{233}) numbers and use the same method for sparse polynomial reduction. The Karatsuba optimized GF(2^{233}) multiplication obtains a speedup of 1.4× comparing to the direct method on our architecture, and a speedup of 8.4× compared to the baseline implementation.

Point Addition/Point Doubling. We implemented point addition and point doubling on projective coordinates [34]. They require only GF multiplication, square and addition operations. However, translating points from affine coordinates (in which coordinates directly map to a point on an elliptic curve) to projective coordinates requires the GF inverse operation. This translation is called once per scalar multiplication. The cycle counts to perform point addition/point doubling and multiplicative inverse are given in Table 9. The comparison point, Clercq [10, 11] in Table 9, is implemented on an ARM M0+ platform. Our architecture with the direct product method achieves a 5.1× improvement for point addition, and a 3.5× improvement for GF(2^{233}) multiplicative inverse. Our architecture with the Karatsuba software optimization achieves a 6.5× improvement for point addition, and a 3.6× improvement for GF(2^{233}) multiplicative inverse.

Scalar Multiplication and ECC$_l$ Applications. Scalar multiplication on an elliptic curve is the operation of adding a point on the elliptic curve to itself repeatedly, i.e. $kP = P+P+P+...+P$ for a scalar k and a point $P = (P_x, P_y)$ on the elliptic curve. We employ the *double-and-add* method to implement scalar multiplication. For example, $k = 15$, kP is computed as $15P = 2[\ 2\ (2P+P)+P]+P$, which requires 3 Point Additions and 3 Point Doublings. The Elliptic Curve Diffie Hellman (ECDH) key exchange protocol [28] is one of the most popular ECC$_l$ applications. It requires one scalar multiplication per session. The control operations to execute an ECDH key exchange protocol are negligible compared to the workload to compute a scalar multiplication. We employ a standard GF(2^{233}) Koblitz curve and a 112-bit level security, in which the highest bit of the scalar k is one and the remaining 112 bits consist of 56 zeros and 56 ones. The resulting 56 Point Additions and 112 Point Doublings take

NIST Koblitz Curve-K233		Clercq [10, 11]	This Work Direct Prod.	This Work Karatsuba
Level 1	Mult.	3672	599	439
	Mult. Pre-comp.	675	N/A	
	Addition	68	66	66
	Square	395	136	136
Level 2	Point Addition	34426	6742	5302
	Point Doubling	No report	3499	2859
	Inverse	139000	39972	38372

Table 9: Cycle Counts of ECC$_l$ Point Addition/Point Doubling

m=5,6,7,8; arbitrary polynomial Configuration of GF Arithmetic Unit			
28nm	GF mult	GF sq	Inst. Control
# of primitive unit	16	28	–
Single primitive unit area (um^2)	199.59	63.48	–
Total Area (um^2)	3193	1777	1005
	5760		
Critical path (ns)	2.91ns @ GF multiplicative_inverse		

Table 10: Power and Area of the Galois Field Arithmetic Unit

617,120 cycles to perform. There are additional supporting functions, including two multiplicative inverses to perform coordinate projection, which consume 157,442 cycles. Running at 100MHz, our GF processor takes 7.75ms for one scalar multiplication and the ECDH key exchange protocol finishes within 8ms. Note that this latency is acceptable in majority of IoT applications because it is only incurred during the initial key exchange process.

3.4 Power and Area

We implement the RTL-level design of our Galois field arithmetic unit and the two-stage in-order processor. The design is synthesized in a 28nm technology. Prime-Time/Power is used to estimate critical path delay and power consumption, respectively. The instruction sequences contain both program control instructions (from the ARM compiler) and hand-coded GF instructions. Data inputs are random generated. The switching activity is dumped from the testbench and fed into Prime-Time/Power to generate a power/time estimation using the synthesized netlist.

3.4.1 Galois Field Arithmetic Unit.

The 16 GF multiplier has 3193um^2 area, 28 GF square has 1777um^2 area, the design total area is 5760um^2. The critical path is 2.91ns, which happens at GF multiplicative inverse. The results are illustrated in Table 10.

The results indicate that our Galois field arithmetic unit is compact, less than 6000μm^2, and fast enough with a critical path of 3ns to support applications in the IoT domain. We believe this Galois field arithmetic unit is a basic building block that can be integrated into many embedded processors as a hardware accelerator.

28nm @0.9V 100MHz		Gate count	Area (um^2)	Power (uW)
2-stage Processor Shell	Comb.	3482	2258	–
	Register File	694	2254	–
	Total	4176	4512	279
GF Arithmetic Unit		7494	5760	152
Design total		11670	10272	431

Table 11: Proposed GF Processor Characteristics

3.4.2 The Galois Field Processor.

As an example, we integrate the GF arithmetic unit into a two stage in-order general purpose processor with a 16-entry 32-bit register file. The total area is 10,272μm^2 and consumes 431 μW at nominal voltage 0.9V while running AES at 100MHz. The GF processor can run at a maximum clock upto 300MHz, which is more than enough for the low power, low bandwidth IoT application domain that we considered in this work. Thus a deeper pipeline, which can provide more throughput with a higher frequency and consumes more energy per instruction, wasn't considered as our control logic architecture.

We conduct SPICE simulation on this technology and apply voltage scaling optimizations to our design. The results show that the design can be voltage scaled down to 0.7V for a target frequency of 100MHz. When scaling, we leave a more than 50% time margin to account for increased variability at lower voltages. This time margin is more than that of previously fabricated designs in 28nm [21] that employ voltage scaling. At 0.7V, the GF arithmetic unit consumes 75μW, and the processor consumes 231μW in total. Voltage scaling will improve our energy efficiency by 1.86×.

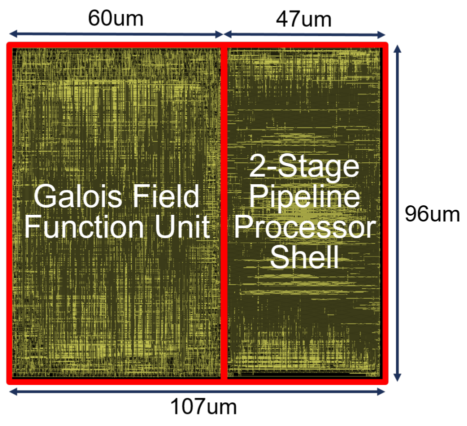

Figure 11: Layout. The area is 10,272um^2.

3.5 Comparison to ASICs

Comparing to AES. Compared to the smallest AES ASIC from Intel [41], our Galois field arithmetic unit, which supports both encryption and decryption, is smaller than the total area of the encryption and decryption datapath combined, as illustrated in Table 12. With 63.5% additional area in total, our processor can support not only

Area (um^2) Scaled to 28nm	Intel [41]	Encryption	Decryption	Total
		2800	3482	6282
	This work	GF arithmetic unit		Total
		5760		10272

Table 12: Area Comparison with Smallest AES ASIC

Scaled to 28nm 0.9V 100MHz	Power (uW)	Throughput (Mbps)	Energy Efficiency (pJ/b)
Zhang [59]	236	38	6.21
This work	431	12.2	35.5

Table 13: Comparison with the Most Energy Efficiency Compact AES ASIC

the different modes of AES, but also various error correction codes and efficient arithmetic support for asymmetric cryptography–ECC$_l$.

On the other hand, programmable solutions are known to consume more power than their ASIC counterpart. Our design consumes 6× more energy per bit compared to the most energy efficient compact AES ASIC design [59]. For systems that are extremely power constrained (i.e., 100s of μW), the ASIC remains a more suitable choice. However, if the system needs flexibility, then our design remains a better choice than using the CPU to support that flexibility. To achieve flexibility with an ASIC would require placing multiple ASIC's on the die, consuming area and increasing cost. Moreover, our solution allows flexible coding and cryptography algorithms to be updated. So for flexible designs concerned with area constraints, our design is preferred.

We do not present any other complete energy efficiency and area comparison at the application level. Other coding and asymmetric cryptography ASIC solutions [29, 33, 35, 37] include implementations with very different features that are either not reported or ambiguous. However, this does not affect the conclusion that a multiple ASIC solution will consume more area than our solution, given the smallest AES [41] is over 60% of our total area.

Comparison to GF Multiplier ASIC. The work in [40] presents a single cycle 64-bit Galois field multiplier for on-die acceleration of public key encryption for a high throughput processor. After scaling it down to 28nm, this GF multiplier accelerator consumes 1.25 mW at 0.9V and 100 MHz, while our GF processor including both arithmetic unit and control logic consumes 0.43 mW under the same condition. The area of our GF processor is 77% of this 64-bit GF multiplier accelerator. In addition, our solution can be programmed to support smaller bit-width operations, which are critical for flexible coding support. Our proposed processor consumes 4× energy per 64b multiplication operation partially due to the register file and control overhead and to the extra cycles for accumulation.

In summary, comparing to ASIC solutions, we provide programmability to support various algorithms in different applications. The unified underlying datapath allows us to address the flexibility within a very small area. The efficient arithmetic support maintains the cost of this flexibility within an affordable region for an IoT device.

4 RELATED WORKS

In previous works, flexible designs were only employed within a particular application (RS, AES, or ECC$_l$), and not optimized to support functionality across the entire GF application space. Previous work [19, 45] focused on binary Galois field arithmetic optimizations for very wide bit widths, targeting for only the ECC$_l$ domain. Moreover, they focus on achieving very high throughput, which is different from the IoT space that this work focuses on. Previous works, typified by [49], propose configurable and high throughput solutions for symmetric cryptographic algorithms with efficient table lookup support. [17] addresses both AES and ECC$_l$ in the same processor. In their work, AES and ECC$_l$ have an individual computation datapath, while the control and memory are shared among two cryptographic methods. This is different from our implementation, where AES and ECC$_l$ utilize the same computation datapath by sharing small bit-width SIMD operations and long bit-width operations. Compared to the work of [2, 36, 48], which supports RS coding with various parameters, we can perform not only RS codes but also other coding schemes in Galois fields. The work in [55] presents optimization for ECC$_l$ on both binary and prime domain, but only on large bitwidths. The low power IoT devices typically prefer binary Galois Field due to its neat implementation, thus we don't consider to support a prime Galois Field. However, we are not just designing in the large GF for ECC$_l$ applications. We design for a broader application domain that includes both small and large bitwidth for both ECC$_l$, ECC$_r$, and AES applications.

5 CONCLUSION

In this work, we presented a light-weight Galois field processor, providing efficient support for both flexible information coding and secure connectivity on IoT devices. We described the design challenges and proposed a flexible bitwidth Galois field arithmetic unit composed of two types of primitive units. Complicated instructions, including multiplicative inverse and wide bit-width partial product were supported by selectively connecting these primitive units. Configurable SIMD instructions were implemented to exploit the parallelism in coding and cryptography. The arithmetic unit was integrated into a two-stage in-order processor. Several coding and symmetric/asymmetric cryptography kernels were mapped onto our architecture, exhibiting a $5-20\times$ speedup across kernels. We demonstrated the effectiveness of our GF processor by synthesizing it in a 28nm technology, where it consumed just $431\mu W$ at 0.9V and 100MHz, while only requiring a place/route area of 0.0103 mm^2.

Finally, we demonstrated the feasibility of a unified architecture to enable coding flexibility and secure wireless communication in the low power IoT domain. The energy efficiency of IoT wireless communication can be enhanced by enabling dynamic adaptation to the error coding schemes in various channel conditions on the proposed architecture. Meanwhile, secure communication was realized via efficient support for both asymmetric and symmetric cryptography on the same hardware platform.

6 ACKNOWLEDGEMENT

This research is supported by ARM Ltd.

REFERENCES

[1] ARM-M0+. Retrieved Oct 2016 from. http://www.arm.com/products/processors/cortex-m/cortex-m0plus.php

[2] Lilian Atieno, Jonathan Allen, Dennis Goeckel, and Russell Tessier. 2006. An Adaptive Reed-Solomon Errors-and-erasures Decoder. In *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays (FPGA '06)*. ACM, New York, NY, USA, 150–158. https://doi.org/10.1145/1117201.1117224

[3] NXP AES benchmarks (Feb. 2013). Retrieved Sep, 2016 from. https://www.lpcware.com/content/project/software-encryption-nxp-arm-microcontrollers

[4] ARM AES benchmarks (Mar. 2015). Retrieved Sep,2016 from. https://www.ietf.org/proceedings/92/slides/slides-92-lwig-3.pdf

[5] Bluetooth SIG. 30 June 2010. Bluetooth Specification Version 4.0. In *The Bluetooth Special Interest Group*.

[6] Daniel R. L. Brown. 2010. Standards for Efficient Cryptography SEC 2: Recommended Elliptic Curve Domain Parameters. (2010).

[7] D. Canright. 2005. A Very Compact S-box for AES. In *Proceedings of the 7th International Conference on Cryptographic Hardware and Embedded Systems (CHES'05)*. Springer-Verlag, Berlin, Heidelberg, 441–455. https://doi.org/10.1007/11545262_32

[8] Yajing Chen, Nikolaos Chiotellis, Li-Xuan Chuo, Carl Pfeiffer, Yao Shi, Ronald Dreslinski Jr, Anthony Grbic, Trevor Mudge, David Wentzloff, David Blaauw, and Hun Seok Kim. 2016. Energy-Autonomous Wireless Communication for Millimeter-Scale Internet-of-Things Sensor Nodes. *IEEE Journal on Selected Areas in Communications* 34, 12 (2016), 3962 – 3977. https://doi.org/10.1109/JSAC.2016.2612041

[9] Yajing Chen, Shengshuo Lu, Hun Seok Kim, David Blaauw, Ronald Dreslinski Jr, and Trevor Mudge. 2016. A low power software-defined-radio baseband processor for the Internet of Things. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 40–51. https://doi.org/10.1109/HPCA.2016.7446052

[10] P.J. de Clercq. 2014. Public Key Cryptography in 32-bit for Ultra Low-Power Microcontrollers. Master's thesis. Katholieke Universiteit Leuven (KU Leuven), Flanders, Belgium.

[11] Ruan de Clercq, Leif Uhsadel, Anthony Van Herrewege, and Ingrid Verbauwhede. 2014. Ultra Low-Power Implementation of ECC on the ARM Cortex-M0+. In *Proceedings of the 51st Annual Design Automation Conference (DAC '14)*. ACM, New York, NY, USA, Article 112, 6 pages. https://doi.org/10.1145/2593069.2593238

[12] Jean-Pierre Deschamps. 2009. *Hardware Implementation of Finite-Field Arithmetic* (1 ed.). McGraw-Hill, Inc., New York, NY, USA.

[13] Ahmed O. El-Rayis, Xin Zhao, Tughrul Arslan, and Ahmet T. Erdogan. 2008. Dynamically programmable Reed Solomon processor with embedded Galois Field multiplier. In *ICECE Technology, 2008. FPT 2008. International Conference on*. 269–272. https://doi.org/10.1109/FPT.2008.4762395

[14] Serdar Süer Erdem. 2012. Fast software multiplication in F_2 [x] for embedded processors. *Turkish Journal of Electrical Engineering & Computer Sciences* 20, 4 (2012), 593–605. https://doi.org/doi:10.3906/elk-1009-756

[15] Jorge Guajardo. 2011. *Itoh–Tsujii Inversion Algorithm*. Springer US, Boston, MA, 650–653. https://doi.org/10.1007/978-1-4419-5906-5_34

[16] Sheryl L. Howard, Christian Schlegel, and Kris Iniewski. 2006. Error Control Coding in Low-power Wireless Sensor Networks: When is ECC Energy-efficient? *EURASIP J. Wirel. Commun. Netw.* 2006, 2 (April 2006), 29–29. https://doi.org/10.1155/WCN/2006/74812

[17] Michael Hutter, Martin Feldhofer, and Johannes Wolkerstorfer. 2011. A Cryptographic Processor for Low-resource Devices: Canning ECDSA and AES Like Sardines. In *Proceedings of the 5th IFIP WG 11.2 International Conference on Information Security Theory and Practice: Security and Privacy of Mobile Devices in Wireless Communication (WISTP'11)*. Springer-Verlag, Berlin, Heidelberg, 144–159. http://dl.acm.org/citation.cfm?id=2017824.2017839

[18] Michael Hutter and Erich Wenger. 2011. Fast Multi-precision Multiplication for Public-key Cryptography on Embedded Microprocessors. In *Proceedings of the 13th International Conference on Cryptographic Hardware and Embedded Systems (CHES'11)*. Springer-Verlag, Berlin, Heidelberg, 459–474. http://dl.acm.org/citation.cfm?id=2044928.2044968

[19] Atef Ibrahim and Fayez Gebali. 2016. Low Power Semi-systolic Architectures for Polynomial-Basis Multiplication over GF(2^m) Using Progressive Multiplier Reduction. *Journal of Signal Processing Systems* 82, 3 (2016), 331–343. https://doi.org/10.1007/s11265-015-1000-x

[20] Surendra K Jain, Leilei Song, and Keshab K. Parhi. 1998. Efficient semisystolic architectures for finite-field arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 6, 1 (March 1998), 101–113. https://doi.org/10.1109/92.661252

[21] Dongsuk Jeon, Michael B. Henry, Yejoong Kim, Inhee Lee, Zhengya Zhang, David Blaauw, and Dennis Sylvester. 2014. An Energy Efficient Full-Frame Feature Extraction Accelerator With Shift-Latch FIFO in 28 nm CMOS. *IEEE Journal of Solid-State Circuits* 49, 5 (May 2014), 1271–1284. https://doi.org/10.1109/JSSC.2014.2309692

[22] Vivek Kapoor, Vivek Sonny Abraham, and Ramesh Singh. 2008. Elliptic Curve Cryptography. *Ubiquity* 2008, May, Article 7 (May 2008), 8 pages. https://doi.org/10.1145/1378355.1378356

[23] A. Karatsuba and Y. Ofman. 1963. Multiplication of Multidigit Numbers on Automata. In *Soviet Physics–Doklady*, Vol. 7. 595–596.

[24] Hun Seok Kim and Babak Daneshrad. 2010. Energy-Constrained Link Adaptation for MIMO OFDM Wireless Communication Systems. *IEEE Transactions on Wireless Communications* 9, 9 (September 2010), 2820–2832. https://doi.org/10.1109/TWC.2010.062910.090983

[25] Hun Seok Kim, Seok-jun Lee, and Manish Goel 2013. Method, device, and digital circuitry for providing a closed-form solution to a scaled error locator polynomial used in BCH decoding (Mar 2013). US Patent 8,392,806, Filed 29 Jul, 2010, Issued 5 Mar, 2013.

[26] Katsuki Kobayashi and Naofumi Takagi. 2009. Fast Hardware Algorithm for Division in GF(2^m) Based on the Extended Euclid's Algorithm With Parallelization of Modular Reductions. *IEEE Transactions on Circuits and Systems II: Express Briefs* 56, 8 (Aug 2009), 644–648. https://doi.org/10.1109/TCSII.2009.2024253

[27] Akash Kumar and Kees van Berkel. 2008. Vectorization of Reed Solomon Decoding and Mapping on the EVP. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '08)*. ACM, New York, NY, USA, 450–455. https://doi.org/10.1145/1403375.1403483

[28] Christian Lederer, Roland Mader, Manuel Koschuch, Johann Großschädl, Alexander Szekely, and Stefan Tillich. 2009. *Energy-Efficient Implementation of ECDH Key Exchange for Wireless Sensor Networks*. Springer Berlin Heidelberg, Berlin, Heidelberg, 112–127. https://doi.org/10.1007/978-3-642-03944-7_9

[29] Jen-Wei Lee, Yao-Lin Chen, Chih-Yeh Tseng, Hsie-Chia Chang, and Chen-Yi Lee. 2010. A 521-bit dual-field elliptic curve cryptographic processor with power analysis resistance. In *2010 Proceedings of ESSCIRC*. 206–209. https://doi.org/10.1109/ESSCIRC.2010.5619893

[30] Chae Hoon Lim and Hyo Sun Hwang. 2000. *Speeding Up Elliptic Scalar Multiplication with Precomputation*. Springer Berlin Heidelberg, Berlin, Heidelberg, 102–119. https://doi.org/10.1007/10719994_9

[31] Wen-Ching Lin, Ming-Der Shieh, and Chien-Ming Wu. 2010. Design of high-speed bit-serial divider in GF(2^m). In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. 713–716. https://doi.org/10.1109/ISCAS.2010.5537479

[32] Yuan Lin, Hyunseok Lee, Mark Woh, Yoav Harel, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti, and Krisztian Flautner. 2006. SODA: A Low-power Architecture For Software Radio. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA '06)*. IEEE Computer Society, Washington, DC, USA, 89–101. https://doi.org/10.1109/ISCA.2006.37

[33] Yi-Min Lin, Hsie-Chia Chang, and Chen-Yi Lee. 2013. Improved High Code-Rate Soft BCH Decoder Architectures With One Extra Error Compensation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21, 11 (Nov 2013), 2160–2164. https://doi.org/10.1109/TVLSI.2012.2227847

[34] Julio López and Ricardo Dahab. 1999. *Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n)*. Springer Berlin Heidelberg, Berlin, Heidelberg, 201–212. https://doi.org/10.1007/3-540-48892-8_16

[35] Shengshuo Lu, Zhengya Zhang, and Marios Papaefthymiou. 2015. 1.32GHz high-throughput charge-recovery AES core with resistance to DPA attacks. In *2015 Symposium on VLSI Circuits (VLSI Circuits)*. C246–C247. https://doi.org/10.1109/VLSIC.2015.7231274

[36] Yung-Kuei Lu and Ming-Der Shieh. 2012. Efficient architecture for Reed-Solomon decoder. In *VLSI Design, Automation, and Test (VLSI-DAT), 2012 International Symposium on*. 1–4. https://doi.org/10.1109/VLSI-DAT.2012.6212603

[37] Yung-Kuei Lu, Ming-Der Shieh, and Wen-Hsuen Kuo. 2009. Design of high-speed errors-and-erasures Reed-Solomon decoders for multi-mode applications. In *2009 International Symposium on VLSI Design, Automation and Test*. 199–202. https://doi.org/10.1109/VDAT.2009.5158129

[38] Jianqiang Luo, Kevin D. Bowers, Alina Oprea, and Lihao Xu. 2012. Efficient Software Implementations of Large Finite Fields GF(2N) for Secure Storage Applications. *Trans. Storage* 8, 1, Article 2 (Feb. 2012), 27 pages. https://doi.org/10.1145/2093139.2093141

[39] Priya Mathew, Lismi Augustine, Sabarinath G., and Tomson Devis. 2014. Hardware Implementation of (63, 51) BCH Encoder and Decoder For WBAN Using LFSR and BMA. *CoRR* abs/1408.2908 (2014).

[40] Sanu Mathew, Michael Kounavis, Farhana Sheikh, Steven Hsu, Amit Agarwal, Himanshu Kaul, Mark Anders, Frank Berry, and Ram Krishnamurthy. 2010. 3GHz, 74mW 2-level Karatsuba 64b Galois field multiplier for public-key encryption acceleration in 45nm CMOS. In *ESSCIRC, 2010 Proceedings of the*. 198–201. https://doi.org/10.1109/ESSCIRC.2010.5619895

[41] Sanu Mathew, Sudhir Satpathy, Vikram Suresh, Mark Anders, Himanshu Kaul, Amit Agarwal, Steven Hsu, Gregory Chen, and Ram Krishnamurthy. 2015. 340 mV-1.1 V, 289 Gbps/W, 2090-Gate NanoAES Hardware Accelerator With Area-Optimized Encrypt/Decrypt GF($(2^4)^2$) Polynomials in 22 nm Tri-Gate CMOS. *IEEE Journal of Solid-State Circuits* 50, 4 (April 2015), 1048–1058. https://doi.org/10.1109/JSSC.2014.2384039

[42] P Mozhiarasi, C Gayathri, and V Deepan. 2015. An Enhanced (31, 11, 5) Binary BCH Encoder and Decoder for Data Transmission. *International Journal of Engineering Research and General Science* 3, 2 Part 2 (2015).

[43] National Institute of Standards and Technology. Feb 2000. Digital Signature Standard, FIPS Publication. (Feb 2000).

[44] TI opensource AES (Jan. 2014). Retrieved Sep 2016 from. http://www.ti.com/tool/AES-128

[45] Jeng-Shyang Pan, Chiou-Yng Lee, and Pramod Kumar Meher. 2013. Low-Latency Digit-Serial and Digit-Parallel Systolic Multipliers for Large Binary Extension Fields. *IEEE Transactions on Circuits and Systems I: Regular Papers* 60, 12 (Dec 2013), 3195–3204. https://doi.org/10.1109/TCSI.2013.2264694

[46] Hua Qian, Shengchen Dai, Kai Kang, and Xudong Wang. 2016. Efficient coding schemes for low-rate wireless personal area networks. *IET Communications* 10, 8 (2016), 915–921. https://doi.org/10.1049/iet-com.2015.0699

[47] Fernando Rosas, Glauber Brante, Richard Demo Souza, and Christian Oberli. 2014. Optimizing the code rate for achieving energy-efficient wireless communications. In *2014 IEEE Wireless Communications and Networking Conference (WCNC)*. 775–780. https://doi.org/10.1109/WCNC.2014.6952166

[48] Cecilia Esperanza Sandoval Ruiz. 2013. RS Decoder (255, k) in Reconfigurable Hardware Oriented Towards Cognitive Radio. *Ingenieria y Universidad* 17, 1 (2013), 77–92.

[49] Gokhan Sayilar and Derek Chiou. 2014. Cryptoraptor: High throughput reconfigurable cryptographic processor. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 155–161. https://doi.org/10.1109/ICCAD.2014.7001346

[50] Hwajeong Seo and Howon Kim. 2013. Optimized multi-precision multiplication for public-key cryptography on embedded microprocessors. *International Journal of Computer and Communication Engineering* 2, 3 (2013), 255–259. https://doi.org/10.7763/IJCCE.2013.V2.183

[51] IEEE Standard 802.15.4. 16 June 2011. Part 15.4: Low-Rate Wireless Personal Area Networks.

[52] IEEE Standard 802.15.6. 29 February 2012. Part 15.6: Wireless Body Area Networks.

[53] Yosef Stein and Joshua A. Kablotsky 2007. Compact Galois field multiplier engine (Feb. 2007). US Patent 7,177,891, Filed Mar 24, 2003, Issued Feb 13, 2007.

[54] Yosef Stein, Haim Primo, and Joshua A. Kablotsky 2004. Galois field multiplier system (Jul. 2004). US Patent 6,766,345, Filed Jan 30, 2002, Issued Jul 20, 2004.

[55] Andrew D. Targhetta, Donald E. Owen, Francis L. Israel, and Paul V. Gratz. 2015. Energy-efficient implementations of GF (p) and GF(2^m) elliptic curve cryptography. In *2015 33rd IEEE International Conference on Computer Design (ICCD)*. 704–711. https://doi.org/10.1109/ICCD.2015.7357184

[56] Stephen B. Wicker. 1995. *Error Control Systems for Digital Communication and Storage*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[57] Mark Woh, Yuan Lin, Sangwon Seo, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti, Richard Bruce, Danny Kershaw, Alastair Reid, Mladen Wilder, and Krisztian Flautner. 2008. From SODA to scotch: The evolution of a wireless baseband processor. In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*. 152–163. https://doi.org/10.1109/MICRO.2008.4771787

[58] Chien-Hsing Wu, Chien-Ming Wu, Ming-Der Shieh, and Yin-Tsung Hwang. 2004. High-speed, low-complexity systolic designs of novel iterative division algorithms in GF(2^m). *IEEE Trans. Comput.* 53, 3 (Mar 2004), 375–380. https://doi.org/10.1109/TC.2004.1261843

[59] Yiqun Zhang, Kaiyuan Yang, Mehdi Saligane, David Blaauw, and Dennis Sylvester. 2016. A compact 446 Gbps/W AES accelerator for mobile SoC and IoT in 40nm. In *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*. 1–2. https://doi.org/10.1109/VLSIC.2016.7573553