Versatile RISC-V ISA Galois Field arithmetic extension for cryptography and error-correction codes

Yao-Ming Kuo
Francisco Garcia-Herrero
Juan Antonio Maestro
{ykuo1,fgarciahe,jmaestro}@nebrija.es
ARIES Research Center, Universidad Antonio de Nebrija
Madrid, Spain

ABSTRACT

With the rise of Edge Computing and the advancement of cryptography and error correction codes, portable device processors must handle a wide range of algorithms based on Galois field arithmetic. A tradeoff between speed and power consumption is required to optimize the execution time of a general-purpose processor without reducing efficiency, if these operations are included in the processor's instruction set. In this work, a RISC-V instruction set for multiplication of Galois field arithmetic is implemented and validated using SweRV-EL2 1.3 on a Nexys A7 FPGA. Performance in some algorithms like AES and Reed-Solomon codes has been improved at the expense of a slight increase in logic utilization.

CCS CONCEPTS

• **Computer systems organization → Reduced instruction set computing**; • **Security and privacy** → *Cryptography*; *Hardware security implementation*; • **Hardware** → *Communication hardware, interfaces and storage.*

KEYWORDS

RISC-V, ISA, Galois Field Arithmetic

ACM Reference Format:

Yao-Ming Kuo, Francisco Garcia-Herrero, and Juan Antonio Maestro. 2021. Versatile RISC-V ISA Galois Field arithmetic extension for cryptography and error-correction codes. In *Proceedings of CARRV '21: Workshop on Computer Architecture Research with RISC-V (with ISCA 2021) (CARRV '21).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In recent decades, with the rise of small portable devices for applications such as the Internet of Things [14] and CubeSats [8], as well as the concepts of Edge Computing [22], Industry 4.0 [12], and Big Data, part of the information must be processed on the client device, alleviating the network's load.

To achieve this goal, processors must be more efficient in executing specific operations and attending tasks with lower latency. Since most portable devices have a general-purpose processor, it is not possible to run specific applications efficiently. Therefore, some solutions must be found to improve its performance. One option is the customization of the processor, identifying the algorithms that are most used in each case. For example, cryptography and error-correction codes are present in almost any portable device today, and many of these algorithms are based on finite field arithmetic. Because of this, the acceleration of finite field arithmetic $GF(2^m)$ proceeds to have a significant role [4]. It is generally used to encode and decode in a communication channel and detect errors in the transmitted data (BCH, Reed-Solomon codes [25]). It is also used in asymmetric cryptography, such as Elliptical Curve Cryptography [24], which is extensively used in the authentication process to exchange private keys or symmetric cryptography inside a secure communication channel as Advanced Encryption Standard (AES) [9].

Furthermore, post-quantum cryptography algorithms (PQC) [10] have been developed in recent years with the rise of quantum computing research. Some of the survivors of the third round of the NIST's PQC competition [21] use $GF(2^m)$ arithmetic (Classic McEliece [2], Rainbow [7], HQC [20], and GeMSS [3]). These algorithms require high computing power to generate the keys and encrypt/decrypt the data, becoming the main bottleneck in processing for small devices, such as IoT end-nodes [19] and Cubesats [15].

Traditionally, there are three ways to optimize a processor. The first solution is adding coprocessors to the system to perform those specific operations. The second is expanding the base instructions of the computer architecture. Moreover, the third solution is to make a hybrid system [23] between the first and the second solution, adding coprocessors and specific instructions depending on the case. We focus on the second solution, expanding the RISC-V base instruction set (RV32I) due to its simplicity to integrate within a core and its low area utilization.

This work aims to propose a flexible instruction set for RV32 cores capable of accelerating any algorithm based on finite field arithmetic $GF(2^m)$, thus improving processor performance. A trade-off between the base instruction set and specific custom instructions can be found for applications that require high flexibility and, at the same time, acceleration in the CPU execution time. For example, a portable device generally contains different error-correction codes and cryptographic standards or proprietary ciphers.

The rest of the paper is organized as follows. Section 2 describes the mathematical basis of $GF(2^m)$, previous related works, and contributions of this work. Section 3 introduces the instruction set extension. Section 4 shows the simulations and experimental results in a RISC-V SoC (SweRVolf). Finally, section 5 finishes with the conclusion of this work.

2 BACKGROUND

This section is divided into three subsections. The first describes the fundamentals of finite field arithmetic. Then, the second subsection details the previous related works. And finally, in the last subsection, the contributions of this work are listed.

2.1 Finite field arithmetic

In this subsection, a quick review of the concepts related to Galois Field operators is made (for more details, refer to [6]), focused on GF(2^m). This field is an extension of GF(2), where the elements that make it up are zero and one. All finite fields have a unit element (α^0), a zero element ($\alpha^{-\infty}$), a primitive element (α), and at least one irreducible polynomial $p(x) = x^m + p_{m-1}x^{m-1} + ... + p_1 x + p_0$. The primitive element α is the root of the irreducible polynomial and generates all the GF(2^m) nonzero elements.

There are two ways to represent the elements in GF(2^m): exponential and polynomial. In the exponential representation, the parts are defined as powers of α, i.e.

$$GF(2^m) = \{0, \alpha^0, \alpha^1, \alpha^2, ..., \alpha^{2m-2}\} \tag{1}$$

While the polynomial representation has the following form:

$$\begin{aligned} P(\alpha) &= a_{m-1}\alpha^{m-1} + ... + a_1\alpha + a_0; \\ a_i &\in GF(2), 0 \le i \le m-1 \end{aligned} \tag{2}$$

Polynomial representation is beneficial for doing arithmetic operations. The definitions of addition and multiplication of finite fields are given below.

2.1.1 *GF Addition.*

Let's consider two elements of $a(x)$ and $a(x)$ (Eq. 3), both belonging to the field $GF(2^m)$.

$$\begin{aligned} a(x) &= a_{m-1}x^{m-1} + ... + a_1 x + a_0 \\ b(x) &= b_{m-1}x^{m-1} + ... + b_1 x + b_0 \\ a_i \wedge b_i &\in GF(2), 0 \le i \le m-1 \end{aligned} \tag{3}$$

The sum $s(x)$ (Eq. 4) is directly the XOR operation of each of its coefficients. The result belongs to the same field.

$$s(x) = (a_{m-1} \oplus b_{m-1})x^{m-1} + ... + (a_0 \oplus b_0) \tag{4}$$

2.1.2 *GF Multiplication.*

There are different ways to multiply two polynomials in $GF(2^m)$. This paper focuses on two-step multiplication. As its name implies, this method separates multiplication into two steps: carry-less multiplication and polynomial reduction.

The first step is carry-less multiplication. The product $d(x)$ of the polynomials $a(x)$ and $b(x)$, is a polynomial of degree $2m-2$. This operation can be represented in matrix form as:

$$\begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_{m-1} \\ d_m \\ d_{m+1} \\ \vdots \\ d_{2m-2} \end{pmatrix} = \begin{pmatrix} a_0 & 0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 & a_0 \\ 0 & a_{m-1} & a_{m-2} & \cdots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & \cdots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & a_{m-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-2} \\ b_{m-1} \end{pmatrix} \tag{5}$$

After the carry-less multiplication, the next step is the polynomial reduction based on an irreducible polynomial $f(x)$. In modular reduction $c(x) = d(x) mod f(x)$, the degree of $d(x)$ is reduced by the degree of the irreducible polynomial $f(x)$, resulting in a degree less than $m\check{}1$. The matrix form of the polynomial reduction is showed in Equation 6.

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 & r_{0,0} & \cdots & r_{0,m-2} \\ 0 & 1 & \cdots & 0 & r_{1,0} & \cdots & r_{1,m-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & r_{m-1,0} & \cdots & r_{m-1,m-2} \end{pmatrix} \begin{pmatrix} d_0 \\ \vdots \\ d_{m-1} \\ d_m \\ \vdots \\ d_{2m-2} \end{pmatrix} \tag{6}$$

The matrix R in Equation 6 depends exclusively on the irreducible polynomial $f(x)$. The coefficients r can be calculated as follows:

$$r_{j,i} = \begin{cases} f_j; j = 0, \ldots, m-1; i = 0 \\ r_{j-1,i-1} + r_{m-1,i-1}; j = 0, \ldots, m-1; i = 1, \ldots, m-2 \end{cases} \tag{7}$$

2.2 Previous related works

In this subsection, related works by other authors are presented.

The RISC-V community has proposed a scalar cryptographic extension [26]. This instruction set accelerates various cryptographic algorithms, such as AES [17], SHA-256, SHA-512, SM3, and SM4. Although it achieves a considerable speedup, this proposal does not contemplate post-quantum (PQC) algorithms and error-correction codes. In this way, the instruction set is not flexible and cannot accelerate other algorithms or proprietary ciphers.

It can be observed that cryptography and error-correction codes share the same operations, such as bit manipulation (i.e. rotations, permutations, carry-less multiply) and finite field arithmetic. These operations can be defined in the instruction set in order to accelerate a wider range of algorithms. For example, a finite field $GF(q)$ ISA extension was proposed in Alkim's [1] work to accelerate lattice-based PQC cryptography (Kyber, NewHope). The same can be done for the $GF(2^m)$ fields to speed up code-based PQC, error-correction codes, and basic operations of classical cryptography [13].

2.3 Contributions

The contribution of this work is a solution between the RISC-V base ISA and the scalar cryptographic K extension [26], resulting

	31 ... 25	24 ... 20	19 ... 15	14 ... 12	11 ... 7	6 ... 0
clmul	0000101	rs2	rs1	001	rd	0110011
clmulh	0000101	rs2	rs1	011	rd	0110011
ffwidth	0000111	rs2	rs1	000	unused	0110011
ffred	0000111	rs2	rs1	001	rd	0110011

Figure 1: The instruction format for the custom Galois field arithmetic instructions.

in an intermediate performance between the two. However, with greater flexibility in the protocols that it can process.

The RISC-V ISA K extension can accelerate SHA2, AES, SM3, and SM4 using dedicated instructions, and they plan to support more algorithms in the future (Aria, Camelia, NIST LWC, and PQC algorithms). As this extension implements dedicated hardware for each algorithm, the execution time is faster, but the area is much larger than that of our proposal.

Our proposed ISA extension is capable of processing the following algorithms:

- Non-binary error-correction codes (i.e., Non-binary low density parity check (LDPC), BCH, RS codes, ...)
- Pre-quantum cryptography (i.e., AES, Elliptic Curve, ...)
- PQC cryptography (i.e., McEliece, Rainbow, HQC, ...)

3 PROPOSED ISA EXTENSION

This section describes the ISA extension for RISC-V processors proposed in this work. The selection criteria of the opcodes were based on the SweRV-EL2 microarchitecture [16].

The operation added in this extension is the multiplication of finite fields since the sum of two numbers in $GF(2^m)$ is only an XOR operation, and it is already defined in RV32I. The multiplication is done in three different instructions (See Section 2.1.2): carry-less multiplication (CLMULH and CLMUL) and polynomial reduction (FFRED). The carry-less operation requires two instructions since multiplying two 32-bit numbers results in a 64-bit number, and the size of the RV32 registers is 32-bit.

To correctly multiply two numbers in $GF(2^m)$, it is also necessary to indicate the irreducible polynomial and the degree to the processor. Therefore, additional instruction is required to pass these parameters (FFWIDTH). These parameters are stored in two internal registers at the input of the reduction module. The block diagram is shown in Fig. 2.

In some processors, such as SweRV-EL2, carry-less multiplication is already implemented in the processor as an extension. Hence, the opcode is kept for a compatibility and resource sharing issue.

We decided not to implement the square function and the inverse of $GF(2^m)$ since they are modules that require much logic and can also be calculated using finite field multiplication. In more specific computers, these two instructions can be added to improve the system's efficiency at the expense of increasing logic utilization.

Figure 1 shows the formats of the custom instructions proposed in this work. As we can see here, the carry-less multiplication keeps the RISC-V extension B format.

The parameters that receive and return the instructions are:

- **FFWIDTH**: It receives in RS1 the degree of the polynomials, and in RS2, the irreducible polynomial. These two parameters are stored in internal parameters. Since RV32I registers are 32-bits, the most significant bit of the irreducible polynomial is assumed to be logical 1 when the degree of the input polynomials is 32.
- **FFRED**: It receives the polynomial to be reduced as a parameter. In RS1, it receives the high part, and in RS2, the low part of the polynomial. This instruction returns the reduced polynomial $c(x)$.
- **CLMULH & CLMUL**: The parameters are the same as extension B.

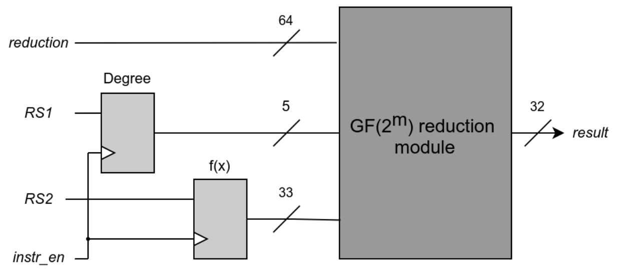

Figure 2: FFWIDTH internal registers.

```
li       a5,  8       % Polynomial degree
li       a4,283       % Primitive poly
ffwidth  a5,a5,a4
...
lbu      a4,0(a0)     % Operand A
lbu      a7,0(a1)     % Operand B
clmul    a4,a4,a7     % CL multiplication
li       a5,0
ffred    a7,a5,a4     % Reduction
```

Figure 3: GF multiplication for AES.

Instruction	Inputs	Destination register	Description
li a5, 8	8	a5	Load the value 8 in register a5
li a4, 283	283	a4	Load 283 (primitive poly.) in a4
ffwidth	a4, a5	a5	Store 8 and 283 in internal registers
⋮	⋮	⋮	⋮
lbu	0(a0)	a4	Load operand A in a4
lbu	0(a1)	a7	Load operand B in a7
clmul	a4, a7	a4	CL multiply, result in a4
li	0	a5	Load 0 in a5
ffred	a4, a5	a7	Poly. reduction, result in a7

Table 1: Detailed description of the code in Fig. 3 (AES example).

An example of the multiplication of finite fields using the custom instructions is shown in Fig. 3. This example is a part of the AES code. AES employs the following reducing polynomial for multiplication:

$$f(x) = x^8 + x^4 + x^3 + x + 1 \tag{8}$$

This primitive polynomial belongs to $GF(2^8)$. Therefore, the degree is 8. Additionally, this polynomial can represent in binary, hexadecimal, or decimal form (see Eq. 9).

$$100011011_2 = 11B_{16} = 283_{10} \tag{9}$$

The number 283 is the value loaded in the second instruction of Fig. 3, representing the primitive polynomial in base 10.

The FFWIDTH instruction appears only for the first time to tell the hardware the degree and the irreducible polynomial. These two received parameters are stored in internal registers. Additionally, this instruction receives a value in the result in a5 since the format of FFWIDTH corresponds to an operation instruction in RISC-V. This instruction is not used in any other part of the code.

The register RS1 passed to FFRED is 0 because AES uses polynomials that belong to $GF(2^8)$ and the bits 63-32 will always be zero after carry-less multiplication. In this case, the compiler used the instruction LI assigning a logical zero to RS1 instead of using CLMULH (see Fig. 3).

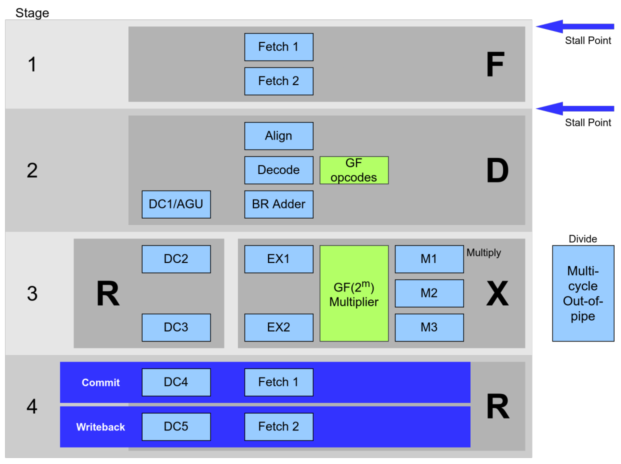

Figure 4: SweRV EL2 Core Pipeline [5].

A detailed description of each instruction in this example can be seen in Table 1.

4 IMPLEMENTATION RESULTS

The custom instructions are implemented and validated with Verilator v4.032 using the SweRV-EL2 v1.3 core.

The first step is adding the logic in the decoding stage to recognize the opcode of the custom instructions. Espresso logic minimizer [18] is used in this stage, which is a logic synthesis computer program capable of reducing the complexity of digital logic circuits.

Then, the corresponding logic is added in the execution stage. Since carry-less multiplication and binary multiplication share the same module within the core, the polynomial reduction module is also implemented in the same block. The block diagram of the SweRV-EL2 core is shown in Fig. 4.

Once the logic is implemented, the assembly module (binutils) is modified. Thus, the toolchain can recognize the opcodes of the custom instructions. Binutils is a collection of binary tools and part of the RISC-V GNU toolchain, including the assembler and the linker. In this way, we can run tests of different algorithms using the C language and compare the efficiency between the RV32IMC base and custom instructions. In this work, a performance evaluation for AES and Reed-Solomon codes is made.

4.1 AES performance

The C code was generated for the different key sizes (AES128, AES192, and AES256) and encryption schemes (CBC, CTR, and ECB). Another version was created with the custom instructions replacing the code segments where the GF multiplication appears. Then, they were compiled with the following flags:

-O3 -fomit-frame-pointer -fPIC -no-pie

	RS(255,247)		RS(255,239)	
	Encode	Decode	Encode	Decode
standard	154,003	151,681	300,831	303,289
out proposal	29,006	22,648	58,660	45,237
Reduc. %	81.17%	85.07%	80.50%	85.08%

Table 2: Number of clock cycles required for RS codes.

AES128	CBC Enc.	CBC Dec.	CTR Enc.	CTR Dec.	ECB Enc.	ECB Dec.
standard	197,920	198,240	198,208	198,197	50,641	50,726
our proposal	38,328	39,303	39,033	38,995	10,854	11,011
Reduc. %	80.63%	80.17%	80.31%	80.33%	78.57%	78.29%

AES192	CBC Enc.	CBC Dec.	CTR Enc.	CTR Dec.	ECB Enc.	ECB Dec.
standard	242,573	242,695	242,839	242,828	62,572	62,661
our proposal	47,016	48,019	47,637	47,617	13,764	13,939
Reduc. %	80.62%	80.21%	80.38%	80.39%	78.00%	77.75%

AES256	CBC Enc.	CBC Dec.	CTR Enc.	CTR Dec.	ECB Enc.	ECB Dec.
standard	285,245	285,593	285,439	285,425	72,331	72,416
our proposal	53,396	54,548	54,054	54,036	14,462	14,660
Reduc. %	81.28%	80.90%	81.06%	81.07%	80.01%	79.76%

Table 3: Number of clock cycles required for AES (RV32IMC vs custom).

SweRV-EL2	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Slice	LUT as logic
standard	18,605	7651	341	74	5,329	18,605
our proposal	19,974	7688	413	80	5,699	19,974
Inc. %	7.36%	0.48%	21.11%	8.11%	6.94%	7.36%

Table 4: Logic utilization of SweRV-EL2 core, implemented on a Nexys A7 FPGA.

The **typical_pd** and **fpga_optimize** settings were used for the SweRV-EL2 core. These configurations create a lightweight processor to implement on the Nexys A7 FPGA.

Table 3 shows the number of clock cycles required for the base and custom instructions for each encryption method. It can be seen that for all of the cases, a significant reduction of 77.75% can be reached. For example, for AES256, the clock cycles needed for CBC encryption is 285,245, while using our proposal only needs 53,396.

Regarding the code size reduction, it was observed that it is greater than 35% for all cases.

4.2 Reed-Solomon performance

The same was done for the Reed-Solomon code performance comparison. A program in C code has been created with the encryption and decryption routine for RS(255,247) and RS(255,239). And then, another version was created with the custom instructions, replacing the multiplication of finite fields.

The same SweRV-EL2 configurations as for AES were kept, and they were also compiled with the same GCC flags.

Table 2 shows the number of clock cycles required to encrypt and decrypt the RS(255,247) and RS(255,239) codes. As we can see here, for example, the clock cycles required to encode in RS(255,247) have been reduced by 81.17%, being 154,003 for RV32IMC and 29,006 for our proposal.

4.3 Logic utilization

In order to implement in an FPGA (Nexys A7), the SweRV-EL2 core was integrated into SweRVolf [11] SoC, which consists of the SweRV CPU with a boot ROM, AXI4 interconnect, UART, SPI, RISC-V timer, and GPIO.

Two different SoC versions were created, one with the base instructions and the other with the custom instructions, both with the extension Zbc [5] enabled.

Table 4 shows the logic utilization for the standard and custom version. It can be seen that there is a 6.94% increase in the number of slices.

Regarding the clock frequency, both work at 50 MHz. There was no decrease in frequency due to the addition of the extra logic for the custom instructions.

5 CONCLUSIONS

Due to the rise of Edge Computing [22], small portable devices have to process efficiently different error correction codes, cryptography, and in some cases, proprietary protocols.

Finite field $GF(2^m)$ arithmetic has been identified to be used in many applications, such as cryptography and error-correction codes. Therefore, an extension of the instructions is proposed in this work. It is oriented to applications where it is necessary to process different protocols that use finite field arithmetic.

From the previous section, it can be seen that for AES and Reed-Solomon, a reduction of 77.75% was achieved in the number of clock cycles at the expense of a 6.94% increment in logic utilization.

As future work, we are using these instructions for post-quantum cryptography. The Classic McEliece and HQC algorithms are adapted to the RISC-V architecture. We are in the stage of evaluating its performance and extracting preliminary results.

REFERENCES

[1] Erdem Alkim, Hülya Evkan, Norman Lahr, Ruben Niederhagen, and Richard Petri. 2020. ISA Extensions for Finite Field Arithmetic: Accelerating Kyber and NewHope on RISC-V. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020, 3 (Jun. 2020), 219–242. https://doi.org/10.13154/tches.v2020.i3.219-242

[2] Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. 2017. Classic McEliece: conservative code-based cryptography. *NIST submissions* (2017).

[3] Antoine Casanova, Jean-Charles Faugere, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. 2017. *GeMSS: a great multivariate short signature.* Ph.D. Dissertation. UPMC-Paris 6 Sorbonne Universités; INRIA Paris Research Centre, MAMBA Team ....

[4] Yajing Chen, Shengshuo Lu, Cheng Fu, David Blaauw, Ronald Dreslinski, Trevor Mudge, and Hun-Seok Kim. 2017. A programmable Galois Field processor for the Internet of Things. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA).* 55–68. https://doi.org/10.1145/3079856.3080227

[5] Western Digital Corporation. 2020. *RISC-V SweRV EL2 Programmer's Reference Manual.* https://github.com/chipsalliance/Cores-SweRV-EL2/blob/branch-1.3/docs/RISC-V_SweRV_EL2_PRM.pdf

[6] Jean-Pierre Deschamps, Jose Luis Imana, and Gustavo D Sutter. 2009. *Hardware implementation of finite-field arithmetic.* McGraw-Hill Education.

[7] Jintai Ding and Dieter Schmidt. 2005. Rainbow, a New Multivariable Polynomial Signature Scheme. In *Applied Cryptography and Network Security*, John Ioannidis, Angelos Keromytis, and Moti Yung (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 164–175.

[8] Hank Heidt, Jordi Puig-Suari, Augustus Moore, Shinichi Nakasuka, and Robert Twiggs. 2000. CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation. (2000).

[9] Simon Heron. 2009. Advanced encryption standard (AES). *Network Security* 2009, 12 (2009), 8–12.

[10] A. Khalid, S. McCarthy, M. O'Neill, and W. Liu. 2019. Lattice-based Cryptography for IoT in A Quantum World: Are We Ready?. In *2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI).* 194–199. https://doi.org/10.1109/IWASI.2019.8791343

[11] Olof Kindgren. [n.d.]. *SweRVolf Github repository.* https://github.com/chipsalliance/Cores-SweRVolf

[12] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. 2014. Industry 4.0. *Business & information systems engineering* 6, 4 (2014), 239–242.

[13] Wei-Ming Lim and M. Benaissa. 2003. Design Space Exploration of a Hardware-Software Co-Designed GF(2<sup>m</sup>) Galois Field Processor for Forward Error Correction and Cryptography. In *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (Newport Beach, CA, USA) *(CODES+ISSS '03).* Association for Computing Machinery, New York, NY, USA, 53–58. https://doi.org/10.1145/944645.944659

[14] Lu Tan and Neng Wang. 2010. Future internet: The Internet of Things. In *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, Vol. 5. V5–376–V5–380. https://doi.org/10.1109/ICACTE.2010.5579543

[15] M. R. Maheshwarappa, M. D. J. Bowyer, and C. P. Bridges. 2017. Improvements in CPU FPGA Performance for Small Satellite SDR Applications. *IEEE Trans. Aerospace Electron. Systems* 53, 1 (2017), 310–322. https://doi.org/10.1109/TAES.2017.2650320

[16] Ted Marena. 2019. RISC-V: high performance embedded SweRV™ core microarchitecture, performance and CHIPS Alliance. *Western Digital Corporation* (2019).

[17] Ben Marshall, G. Richard Newell, Dan Page, Markku-Juhani O. Saarinen, and Claire Wolf. 2020. The design of scalar AES Instruction Set Extensions for RISC-V. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 1 (Dec. 2020), 109–136. https://doi.org/10.46586/tches.v2021.i1.109-136

[18] P.C. McGeer, J.V. Sanghavi, R.K. Brayton, and A.L. Sangiovanni-Vicentelli. 1993. ESPRESSO-SIGNATURE: a new exact minimizer for logic functions. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 1, 4 (1993), 432–440. https://doi.org/10.1109/92.250190

[19] J. M. McGinthy and A. J. Michaels. 2018. Lightweight Internet of Things Encryption Using Galois Extension Field Arithmetic. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData).* 74–80. https://doi.org/10.1109/Cybermatics_2018.2018.00046

[20] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and IC Bourges. 2018. Hamming quasi-cyclic (HQC). *NIST PQC Round* 2 (2018), 4–13.

[21] Dustin Moody. 2016. Post-quantum cryptography: NIST's plan for the future. In *The Seventh International Conference on Post-Quntum Cryptography, Japan.*

[22] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

[23] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha. 2007. A Synthesis Methodology for Hybrid Custom Instruction and Coprocessor Generation for Extensible Processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 11 (2007), 2035–2045. https://doi.org/10.1109/TCAD.2007.906457

[24] Lawrence C Washington. 2008. *Elliptic curves: number theory and cryptography.* CRC press.

[25] Stephen B Wicker and Vijay K Bhargava. 1999. *Reed-Solomon codes and their applications.* John Wiley & Sons.

[26] Alexander Zeh, Andy Glew, Barry Spinney, Ben Marshall, Daniel Page, Derek Atkins, Ken Dockser, Markku-Juhani O Saarinen, Nathan Menhorn, Richard Newell, et al. [n.d.]. RISC-V Cryptographic Extension Proposals Volume I: Scalar & Entropy Source Instructions. ([n. d.]).