Programmable Processing Element for Crypto-Systems on FPGAs

Mohamed El-Hadedy[*]
mea4c@virginia.edu

Kevin Skadron[†]
skadron@virginia.edu

Hristina Mihajloska[‡]
hristina.mihajloska@finki.ukim.mk

Danilo Gligoroski[§]
danilog@item.ntnu.no

ABSTRACT

This paper presents the design and analysis of an area-efficient programmable processing element (PPE) for implementing diverse cryptographic systems and diverse bit-widths (currently 16, 32, and 64). To evaluate the effectiveness of our design, we implement π-Cipher and BMW on the PPE. π-Cipher is a new algorithm for authenticated encryption that offers advantages over AES-GCM and is a candidate in the CAESAR competition. BMW is a SHA-3 candidate and is used for the QuarkCoin crypto-currency. The design of the programmable processing element PPE requires the use of on-chip memory for storing the internal structure of one round of the π-function as well as for the PPE instruction logic. With the new processing element, on Xilinx Virtex-5, we implemented the PPE in just 227 slices, achieving a throughput of 1.17 Gbps/block for the π-Cipher 64-bit version and 256 Mbps/block for BMW at 250 MHz. The PPE is designed to be modular, for inclusion in larger FPGA designs or SoCs, and is also easily extended to wider bit-widths.

Keywords

PiCipher, Crypto-systems, CAESAR, FPGA

1. INTRODUCTION

Cryptography is essential to the modern IT economy. In 2013, the National Institute of Standards and Technology NIST funded a new Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [3] to identify a portfolio of authenticated ciphers that offer advantages over AES-GCM and are suitable for widespread adoption. In addition to security considerations, availability of an efficient hardware implementation will be a factor in the CAESAR selection. Yet algorithm-specific, fixed-logic processing elements cannot support the Babel of cryptographic standards already present today or diverse data-path widths, let alone the rapid innovation in cryptographic standards. We therefore design a new programmable crypto-processor. In this paper, we present the design and analysis of an area-efficient Programmable Processing Element (PPE) for use in cryptographic systems. The introduced PPE can be used to implement all the crypto-systems that are based on ARX (Adding, Rotation and Xor'ing) operations and supports multiple data-path widths. In addition, the PPE is designed to be modular, for use in more complex systems.

As a proof of concept, we used the PPE to introduce the first hardware implementation of the π-Cipher algorithm. π-Cipher is one of CAESAR candidates which core permutation function is based on ARX operations. This cipher is parallelizable sponge-based design with associated data. π-Cipher is designed for different word sizes, different block sizes and different security levels [9]. The modes that are supported are for small word sizes on low-power micro controllers (16-bit registers) π16-Cipher, 32-bit platform π32-Cipher and for 64-bit platform π64-Cipher. In addition, we used the proposed PPE for the implementation of the Blue Midnight Wish (BMW) hash function. BMW was one of the second-phase SHA-3 competition candidates [5]. At the moment the BMW algorithm is one of the selected hash algorithms for the crypto-currency QuarkCoin [1]. The PPE architecture is evaluated in an FPGA implementation using a Xilinx Virtex-5 FPGA.

2. RELATED WORK

Several architectures have been proposed for compact crypto-systems.In 2010, Beuchat et al. [4] presented a compact architecture of one of the last SHA competition's candidates, BLAKE. Blake-32 is implemented in just 56 slices with two block memories, achieving 115 Mbps. In 2011, El-Hadedy et al. [6] introduced a low area processor to be used for implementing different versions of BMW (BMW-256, BMW-512). BMW-256 is implemented in just 51 slices, achieving a throughput of 68.71 Mbps, and BMW-512 in just 105 slices achieving a throughput of 112.18 Mbps. Both BMW implementations need two block memories to store the hash function internal values as well as the instruction sets. Even

[*]Dr.Mohamed El-Hadedy is a research associate with the Department of Computer Science at the University of Virginia, 85 Engineer's Way, P.O.Box 400740, Charlottesville, Virginia

[†]Professor. Kevin Skadron is the chair of the Department of Computer Science at the University of Virginia, , 85 Engineer's Way, P.O.Box 400740, Charlottesville, Virginia, USA

[‡]Hristina is a Phd student with the Faculty of Computer Science and Engineering at Ss. Cyril and Methodius University, Rugjer Boshkovikj, 16, P.O. Box 393, Skopje, Macedonia

[§]Professor Danilo Gligoroski is with the Department of Telematics at the Norwegian University of Science and Technology(NTNU), O.S.Bragstads plass 2B, N-7491, Trondheim, Norway

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2015) Boston, MA, USA, June 1-2, 2015.

though previous implementations [4, 6] offer a small area on the FPGA platform with reasonable throughput, the entire program must be loaded at once (overlapping of compute with streaming in of new instructions is not supported), requiring a much larger instruction memory (see Section 5.2 for details). Our approach avoids this, allowing a much smaller instruction memory, significantly reducing the size of the processing element. Furthermore, their processors cannot process different data sizes using the same design. This is important for supporting the widest variety of algorithms.

3. ARCHITECTURE OF THE PPE

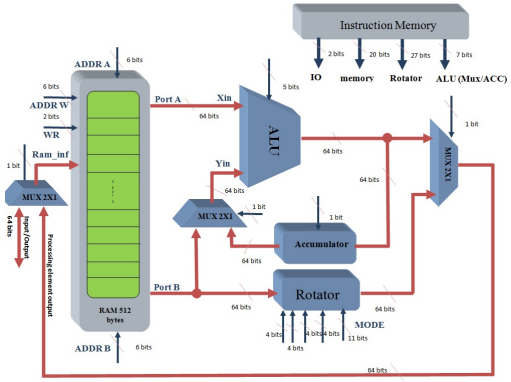
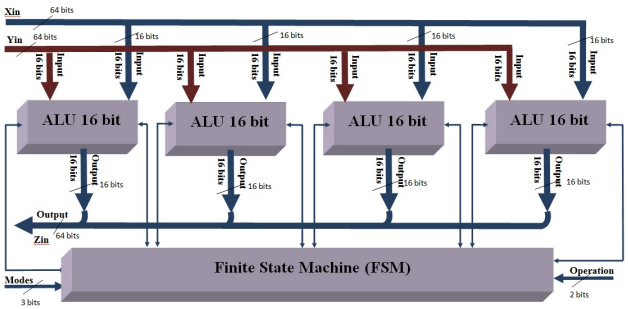


Figure 1: Top level architecture of the programmable processing element

The proposed PPE is based on a very long instruction word VLIW architecture as shown in Fig. 1. It consists of an Arithmetic and Logic Unit ALU, Rotator block, intermediate 512 byte coefficient memory, and multiplexers to organize the data traffic from Input/Output ports and the feedback from the computational part to the coefficient memory. A VLIW architecture was chosen as an efficient way to issue multiple operations in each clock cycle. Changes of control-flow (e.g., branches) are not supported (or needed, for ciphers we have examined). We currently create programs by hand; a compiler is future work, but the limited set of operations and the regular structure of the VLIW architecture should make this straightforward.

The PPE proposed here implements the ARX engine; it assumes the input text has already been padded if necessary.

3.1 ALU



ALU can process different word sizes as one 64-bit, two 32-bit, or four 16-bit operations per clock cycle

Figure 2: Programmable architecture of the ALU

Although ripple carry adders RCA is the most area-efficient adder, its worst-case propagation delay can be severe. We use a carry lookahead adder (CLA), which solves the carry-delay problem by calculating the carry signals in advance, based on the input bits [10]. The Virtex-5 FPGA has LUTs with six inputs and one output; a single LUT can output one bit of data from six-bit input data. If this resource is properly used for the circuit design, it is possible to obtain a higher performing circuit. We use these capabilities to build the CLA ALU as shown in Fig. 2 based on the CLA. The ALU offers three different operations: word XOR, word addition, and subtraction (modulo 2^n). The ALU can process data either from both coefficient memory ports (Port A and Port B) or from xin port and the accumulator register as shown in Fig. 1. In each cycle, the ALU can process 4 data words of 16 bits, 2 data words of 32 bit, or 1 data word of 64 bit, based on the value of the control mode bits For instance, the mode control bits have to be 111 and the operation 01 if we would like to add two 64 bit words.

3.2 Rotator

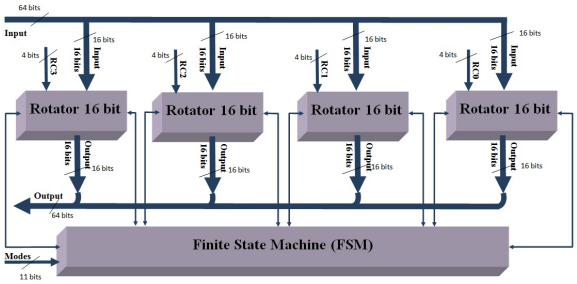


Figure 3: The Rotator

The Rotator consists of four 16-bit sub rotations, controlled by the finite state machineFSM as shown in Fig. 3. Each 16 bit Rotator consists of a 64 X 4 array of 2 X 1 multiplexers, controlled by four, 4-bit RC control bits. Based on the value of the mode control bits, the Rotator can rotate to the right four 16-bit words, two 32-bit words, or one 64-bit words. For instance, to rotate the 64 bit input data 12 times to the right, the Mode signal has to be 00001010101, and RC0, RC1, RC2, and RC3 have the same value 1100 (0xC). On the other hand, if we have four data words of 16 bits, to rotate the first word 12 times to the right, the second word 5 times to the left, the third word 7 times to the right, and the fourth word 10 times to the left, the mode signal is 00000000000, and RC0 = 1100, RC1 = 1011 , RC2 = 0111, and RC3 = 0110.

3.3 Coefficient Memory

As shown in fig. 1,the coefficient memory has two output ports of 64-bits (Port A and Port B). There are two address ports, ADDRA and ADDRB, to control the value of Port A and Port B. ADDRW is responsible for writing data into the memory through ram_inf. The memory size is 512 bytes.

3.4 Instruction Memory

The dual-ported instruction memory of 448 bytes holds the program that implements the chosen cipher algorithm, expressed as a sequence of VLIW instructions. The instruction memory has two address ports; the first 6-bit port acts as an instruction fetch address. The other 6-bit port specifies a write address, to change the values of the instruction memory. There are two enable bits: one permits read for instruction fetch and the other port is used to change the values of the instruction memory.

This dual-ported design allows a new cipher program to be streamed in while the current program progresses, or an early start on a new program while the rest of the program is still loading, or seamless processing of a program that cannot

fit entirely into the on-chip memory. The instruction-fetch ADDR works as a pointer moving sequentially and wrapping around. While this processing is happening, the write ADDR is loading the new code. Because only one read and write are allowed per cycle, conflicts are not possible, as long as the read and write addresses are not initialized to the same value, which we verify. Figure 4 shows a snapshot of each row in the instruction memory. Each instruction consists of 2 bits to control the input/output interface, 20 bits for controlling the coefficient memory, 27 bits for controlling the different sections of the Rotator, and 7 bits for controlling the ALU.

4. DEMONSTRATION ALGORITHMS

Many cryptographic algorithms are ARX based, but the algorithms vary widely in the number and sequence of ARX operations. In this paper, we evaluate two ARX algorithms as case studies. We introduce the first hardware implementation of the π-Cipher, as well as an efficient implementation of the BMW hash function.

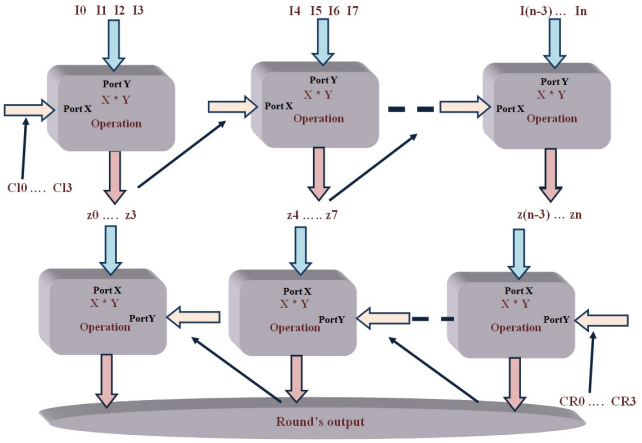
4.1 π-Cipher

π-Cipher is a nonce-based authenticated encryption cipher with associated data. Its design is based on several canonical cryptographic concepts but has some intrinsic differences. In all cases, the independent processing of the message blocks provides straightforward parallelism and incrementality, and two important mechanisms to protect against forgeries, the inclusion of the ordinal number of the message blocks, and a publicly-known nonce. Furthermore, the π-cipher also provides an option of using a secret nonce in addition to the public one. The π permutation is based on ARX operations and uses a two pass counter based sponge component that is denoted as a *triplex component*. More details can be found in the official documentation [8].

The core part of every sponge construction is the permutation function, and the whole security of the primitive relies on it. π-Cipher has an ARX based permutation function which is denoted as the π function. The permutation operates on b bits of state and updates the internal state through a sequence of R successive rounds. The state IS can be represented as a list of N 4-tuples, each of length ω-bits ($\omega = 16$, 32 or 64), where $b = N \times 4 \times \omega$. The general permutation function π consists of three main transformations $\mu, \nu, \sigma : \mathbb{Z}_{2^\omega}^4 \rightarrow \mathbb{Z}_{2^\omega}^4$, where $\mathbb{Z}_{2^\omega}$ is the set of all integers between 0 and $2^\omega - 1$. These transformations perform diffusion and nonlinear mixing of the input. It uses the following operations: addition + modulo 2^ω; left rotation (circular left shift) $ROTL^r(X)$, where X is an ω–bit word and r is an integer, $0 \leq r < \omega$; and bitwise XOR operation $\oplus$ on ω–bit words. An algorithmic description of the $*$ operation over two 4–dimensional vectors $\mathbf{X}$ and $\mathbf{Y}$ for ω-bit words, together with the values of the rotation vectors and of the constants of μ and ν transformations, are given in the official documentation of the π-Cipher [8]. One round of the permutation function is depicted in Figure 5.

4.2 Blue Midnight Wish Hash Function (BMW)

BMW is a wide-pipe Merkle-Damgård hash construction with an unconventional compression function. The nonlinearity in BMW is derived from the overlap of modular XOR and addition operations. The BMW-n hash function family contains four instances for n =224, 256, 384, and 512, where n is the size of the hash value. BMW performs four different operations in the hash computation stage: bit-wise logical



I_0, I_1, ..., I_n are the input message blocks. z_0, z_1, ..., z_n are the intermediate values. Cl_0, Cl_1, .., Cl_3 are the left constant values. CR_0, CR_1, .., CR_3 are the right constant values. In this paper, input message blocks are 16 ($n = 15$), which means one round requires 8 blocks of ARX operations

Figure 5: π-Cipher (one round of the permutation function)

word XOR, word addition and subtraction, shift operations (left or right), and rotate left operations. The size of a word is 32 bits for BMW-224/256 and 64 bits for BMW-384/512. The computation engine of the BMW consists of three subfunctions called f_0, f_1, and f_2, in sequence to generate the chaining value. More details can be found in [7].

5. IMPLEMENTATION OF THE PPE

The functionality of the proposed PPE was verified on the Xilinx Virtex-5 XC5VLX110 device. The PPE has been described on the FPGA platform in VHDL and was synthesized using ISE design suite 14.7. The PPE has been implemented in just 227 slices, including the 512-byte coefficient memory, 448 byte instruction memory, and the computational unit (Rotator and ALU), and can be clocked at 250 MHz.

5.1 π-Cipher on PPE

$$Throughput = \frac{Number\ of\ input\ bits \times Max frequency}{Number\ of\ clock\ cycles\ per\ block}$$

As discussed in Section 4.1, the π-function consists of four rounds, and each round has eight ARX operation blocks. The throughput of the design is given in Equation 2. For 64-bit operation, implementing each operation block costs only 371 bytes in the instruction memory and 384 bytes in the coefficient memory, requiring 54 cycles to process 256 input bits (64 X 4) per block, achieving a throughput of 1.17 Gbps/block at 250 MHz for the 64-bit version. For the 32-bit version, each block costs 189 bytes in the instruction memory, executed in 27 cycles, to process 128 input bits (32 X 4) per block, achieving a throughput of 1.15 Gpbs/block at 250 MHz. And for the 16-bit version, each block costs 91 bytes in the instruction memory executed in 13 cycles to process 64 input bits, achieving a throughput of 1.2 Gpbs/block at 250 MHz. For example, implementing one full round of the *pi*-function (64-bit version) with eight blocks used 512 bytes for storing the coefficients and intermediate values, and the instruction memory will be reprogrammed eight times. As mentioned before, the reconfiguration time is overlapped with the processing time, so for one round of π-function

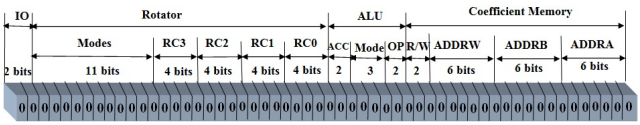


Figure 4: The instruction set

Table 1: BMW-512 performance using the PPE

Algorithm	FPGA Type	Area(slice)	Frequency (MHz)	Throughput (Mbps)	Memory Blocks	Throughput/Area(slice)
PPE (BMW-512)	XC5VLX110	227	250	256	—	1.3
BMW-512 [6]	XC5VLX110	105	115	112.18	3	1.09
BMW-512 [2]	XC5VLX330T	9810	10.004	287	—	0.028

on PPE, the throughput is 150 Mbps.

5.2 Blue Midnight Wish on PPE

For using the PPE to implement BMW-512, f_0, f_1, and f_2 can be executed in just 1000 cycles. The throughput/area ratio for the using the PPE to implement BMW-512 is higher than previous implementations, as shown in Table 1. We have not used the any block memories at all, compared to the implementation in El-Hadedy et al. [6], which used 3 block memories for storing the intermediate coefficients and the instruction sets. That means that, in terms of area, the size of the BMW-512 processor introduced by El-Hadedy et al. [6] is larger than our proposed PPE. For instance, El-Hadedy et al. [6] store the intermediate coefficients of the BMW-512 in 2048 bytes, while our proposed PPE just uses 512 bytes. Meanwhile, they use 4864 bytes for storing the instructions, while we use in total just 1792 bytes.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented the FPGA implementation of a new, area-efficient, programmable, VLIW processing element to be used in cryptographic systems based on ARX (add, rotate, xor) operations. We have implemented the ARX engine and demonstrated its effectiveness for one of the CAESAR competition candidates, π-Cipher, by using the PPE. This implementation is the first hardware implementation of the π-Cipher 16, 32, and 64 bit versions. Also, the BMW-512 hash function has been implemented and the result compared with previous implementations in terms of throughput, area (number of slices), frequency, and the throughput/area.

The next step in this work is to evaluate the effectiveness of this PPE with more cryptographic systems. Other avenues for future work are to develop a compiler for this architecture, and to explore new system architectures that use the PPE. For example, by replicating the PPE, we can implement higher throughput, either by parallelizing the processing of a single stream, or allowing processing of multiple steams concurrently. Furthermore, because each processing element is independently programmable, so these independent concurrent streams can even use different cryptographic systems. For example, we estimate that Virtex-5 (XC5VLX110) can hold at least 50 concurrent PPEs, (although 50 independent input streams would not be feasible due to I/O limitations).

7. ACKNOWLEDGEMENTS

This work was supported in part by the Center for Future Architectures Research (C-FAR), one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

8. REFERENCES

[1] *Bitcoins, litecoins, what coins?: A global phenomenon.* Stevenson, J., 2013.

[2] B.Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. ONeill, and W. P. Marnane. FPGA implementations of the round two SHA-3 candidates, santa barbara. In *In Proceedings of the NIST SHA-3 Conference*, 2010.

[3] D. J. Bernstein. Caesar: Competition for authenticated encryption: Security, applicability, and robustness. CAESAR web page, 2013. `http://competitions.cr.yp.to/index.html`.

[4] J.-L. Beuchat, E. Okamoto, and T. Yamazaki. Compact implementations of BLAKE-32 and BLAKE-64 on FPGA. In *Proceeedings of the Field-Programmable Technology (FPT), 2010 International Conference*, pages 170–177, Dec 2010.

[5] U.D. Commerce. *Nist Interagency Report 7764: Status Report on the Second Round of the SHA-3 Cryptographic Hash Algorithm Competition.* Createspace Independent Pub, 2014.

[6] M. El-Hadedy, D. Gligoroski, and S.J. Knapskog. Area Efficient Processing Element Architecture for Compact Hash Functions Systems on VIRTEX5 FPGA Platform. In *Proceedings of the Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference*, pages 240–247, June 2011.

[7] D. Gligoroski, V. Klima, S. J. Knapskog, M. El-Hadedy, and J.Amundsen. Blue Midnight Wish. In *Proceedings of The First SHA-3 Candidate Conference*, Feb 2009.

[8] Danilo Gligoroski, Hristina Mihajloska, Simona Samardjiska, Hakon Jacobsen, Mohamed El-Hadedy, and Rune Erlend Jensen. π-cipher v1. Cryptographic Competitions: CAESAR, 2014. `http://competitions.cr.yp.to/caesar-submissions.html`.

[9] Danilo Gligoroski, Hristina Mihajloska, Simona Samardjiska, Håkon Jacobsen, Rune Erlend Jensen, and Mohamed El-Hadedy. π-cipher: Authenticated Encryption for Big Data. In Karin Bernsmed and Simone Fischer-Hübner, editors, *Secure IT Systems - 19th Nordic Conference, NordSec 2014, Tromsø, Norway, October 15-17, 2014, Proceedings*, volume 8788 of *Lecture Notes in Computer Science*, pages 110–128. Springer, 2014.

[10] S. Kao, R. Zlatanovici, and B. Nikolic. A 240ps 64b Carry-Lookahead Adder in 90nm CMOS. In *Proceedings of the Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers. IEEE International*, pages 1735–1744, Feb 2006.