SIR10US: A Tightly Coupled Elliptic-Curve Cryptography Co-Processor for the OpenRISC

Michael Gautschi*, Michael Muehlberghuber*, Andreas Traber*, Sven Stucki*, Matthias Baer*, Renzo Andri*, Luca Benini*
*Integrated Systems Laboratory, ETH Zurich, Gloriastrasse 35, 8092 Zurich, Switzerland.
Email: {gautschi,mbgh,luca.benini}@iis.ee.ethz.ch, {svstucki,atraber,baermatt,andrire}@student.ethz.ch

Beat Muheim§, Hubert Kaeslin§
§Microelectronics Design Center, ETH Zurich, Gloriastrasse 35, 8092 Zurich, Switzerland
Email: {muheim,kaeslin}@ee.ethz.ch

***Abstract*—Today's embedded systems require resource-aware acceleration engines, which support advanced cryptographic algorithms such as elliptic-curve cryptography (ECC). The authors present an application-specific co-processor for digital signature verification according to the Elliptic Curve Digital Signature Algorithm (ECDSA) based on the NIST B-233 standard. A novel OpenRISC-ISA (instruction-set architecture) core featuring a high IPC rate and balanced pipeline stages has been developed to act as the main controlling unit of the accelerator. The redesigned OpenRISC core processes 67 % more instructions per second than the reference architecture and ties with a micro-controllable ECC datapath through a highly optimized interface. An ECDSA signature is verified in 11 ms, which is equal to a speedup of 15x and 3.3x with respect to a portable C implementation on the OpenRISC and an assembler-optimized implementation on an ARM7, respectively. Moreover, thanks to a tightly coupled data memory, the proposed co-processor does not block the OpenRISC during its ECC-specific operations, thereby enabling it to also support concurrent execution of other workloads and/or software-based cryptographic extension functions.**

***Keywords*—OpenRISC, elliptic-curve cryptography, ECC, finite-field arithmetic, co-processor, instruction-set extension.**

I. INTRODUCTION

Public-key cryptography based on elliptic curves has become a de-facto standard within embedded devices and resource-constrained environments. Various dedicated microprocessors, aiming at low-cost execution of secure algorithms have been presented [14], [16]. These implementations usually lack the ability of a general purpose CPU. Many embedded systems require a greater flexibility than dedicated hardware in order to support distinct security processing standards [15]. The greater flexibility of a CPU allows to incorporate different security algorithms and also allows to support new or evolving standards by changing the software.

Elliptic-curve cryptography (ECC) has been proposed to be used for public-key cryptography and typically requires rather time-consuming arithmetic in large finite fields. To speed up ECC-algorithms, several dedicated processors [14], [16] as well as various accelerators based on an 8-bit datapath [7], [4], [8] have been published. Combining the flexibility of a general purpose CPU and an application-specific accelerator for ECC permits to maintain the area and power efficiency of the dedicated block, but at the same time brings about greater flexibility. In hardware/software codesign, the designer has several possibilities to connect the accelerator to the processor. Hodjat et al. [5] presented a LEON processor with a tightly coupled AES encryption co-processor which requires 1.7 times less cycles for an AES encryption than its loosely coupled memory mapped version thanks to a direct connection of the integer unit to the co-processor. The tightly coupled approach is more efficient if the core and the accelerator have to interact and the binding to a dedicated unit can be achieved best if the source code of the processor is available. The OpenCores community [13] offers a low-area RISC processor suitable for embedded systems. This OpenRISC processor with its flat pipeline guarantees short reaction times between the co-processor and the OpenRISC. Memory-intensive applications like ECC have to access the memory numerous times for read and write operations and the execution time greatly depends on the number of memory access conflicts. Equipping the accelerator with a local memory solves the memory bottleneck [2] and allows to use the full advantage of the co-processor. Thanks to the tightly coupled integration, the full control of the OpenRISC, and the local memory, no FIFOs, domain-crossing functionality, or serializers/deserializers are required which greatly simplifies the integration.

The contributions of our work can be divided into three parts. First, we propose a redesign of the OpenRISC microprocessor, which outperforms its reference design in terms of instructions per cycle (IPC) and the maximum clock frequency, leading to a 67 % higher throughput. Second, a tightly coupled ECC co-processor has been developed, suitable to accelerate elliptic-curve operations based on the NIST B-233 standard [12]. It features a 16-bit datapath supporting both binary- and prime-field arithmetic required to complete the Elliptic Curve Digital Signature Algorithm (ECDSA) [6] conforming to the aforementioned standard. Finally, we show how to link the co-processor to the redesigned OpenRISC using a tightly coupled, memory-mapped approach. Compared to a portable, software-only implementation of an ECDSA signature verification process, the co-processor achieves a 15x speedup and is 21x more energy-efficient. The OpenRISC core requires 25 % of the area of the final layout, but offers great flexibility and can proceed with other workloads and/or execute software-based cryptographic extension functions while the co-processor executes the ECC-specific operations.

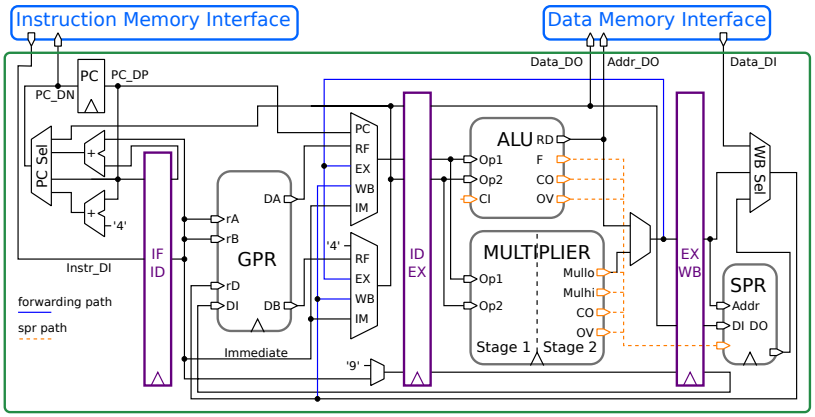

Fig. 1. OR10N micro-architecture with four pipeline stages. Multiplier and ALU share the same write port and the special purpose register (SPR) is used to store the upper 32 bit of a multiplication together with other status bits.

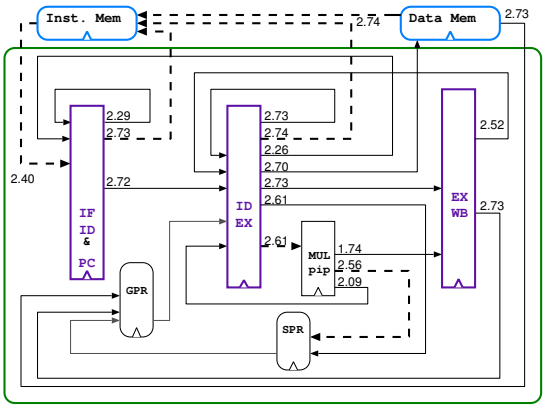

Fig. 2. Critical paths in nano seconds of all pipeline stages obtained through synthesis on a 180 nm process technology.

The remainder of this paper is structured as follows: Section II presents our implementation and improvements of the OpenRISC microprocessor. Our finite-field arithmetic co-processor and its integration into the OpenRISC is described in Section III. The results of our work are presented in Section IV, before we draw conclusions in Section V.

II. OpenRISC Micro-architecture

The OpenCores community [13] developed the OpenRISC architecture, an open-source processor with a fully working toolchain. We investigated the existing Verilog implementation, called *or1200* [9], and identified significant room for improvements with regard to its IPC performance, which amounted to approximately 0.6. The main micro-architectural weaknesses were found during the execution of multiplications, branches, and load/store operations. We redesigned the micro-architecture from scratch to achieve high IPC values, but maintained the four-stage pipeline. Moreover, we balanced the pipeline stages, which allows us to operate the core at a higher frequency and/or lower voltage for better energy efficiency. Fig. 1 shows a simplified block diagram of the pipeline organization of our architecture called OR10N. Fig. 1 illustrates the four stages of the 32-bit RISC processor: *instruction fetch (IF)*, *instruction decode (ID)*, *execute (EX)*, and *write back (WB)*. The core is attached to an instruction and data RAM.The instruction memory interface is implemented such that the instruction RAM can be replaced with a cache. The data memory interface grants incoming requests in the same cycle. It is therefore possible to stall the pipeline immediately when a request is not granted. To achieve a higher performance of the core, it is important that the critical paths between the different pipeline stages are balanced. Fig. 2 highlights all critical paths from one stage to another. The longest path, starting at the data memory, adds up to 2.74 ns and limits the maximum frequency of OR10N. Critical paths between all other stages are also in the range of 2.7 ns which illustrates that the pipeline stages are balanced.

For our RISC micro-architecture, we aimed at a near-optimal IPC performance of 1 to maximize the processor's efficiency in terms of power and performance. Therefore, we significantly improved branch execution, multiplication handling, and the load/store interface. Branch execution is done in the *ID-Stage*, which allows us to execute branches and jumps without stalling the pipeline. Load/store operations are executed in the *EX-Stage* and the result of a load-word operation is written back in the *WB-Stage*. The multiplication unit has been pipelined once in order to break the critical path of the multiplier. This was achieved by moving the output register of the multiplier into the multiplier with the use of the `optimize_register` command from Synopsys Design Compiler. As a consequence, if the result of the multiplier is used in the following cycle, the pipeline needs to be stalled. Otherwise, the result can be written back into the general purpose register (GPR) in the next clock cycle on a separate write port. However, as illustrated in Fig. 2, the paths ending at the GPR are critical, and therefore we did not add a second write port but reused the existing port whenever it is not used by an ALU- or load-word-operation. If the port is already in use, the pipeline needs to be stalled for one cycle to write back the result of the multiplier.

In order to increase the performance of our OR10N microprocessor for ECC applications, we now propose an application-specific co-processor attached to the core.

III. An ECC Co-Processor for the OpenRISC

Hardware acceleration of ECC can be achieved at different hierarchy levels. Fig. 3 illustrates the levels of abstraction, starting from the top with the actual ECC protocol, down to the finite-field arithmetics, which represent core components of an ECC protocol. We aimed to support the ECDSA algorithm, standardized by several institutions including NIST [12]. The targeted elliptic curve for our co-processor is the *B-233* curve as defined by NIST. Since the elliptic-curve operations represent the most time-consuming parts of an ECC protocol, the performance of the underlying binary finite-field arithmetic is of utmost importance. To verify a digital signature, two main building blocks are required in addition to the elliptic-curve operations (for details see [12]). First, a hash algorithm such as SHA-2 [11] or its successor, the upcoming SHA-3 [3], is required to generate a fixed-size message digest of the incoming data. Second, some operations within a prime finite field $\mathbb{F}_p$ are needed.

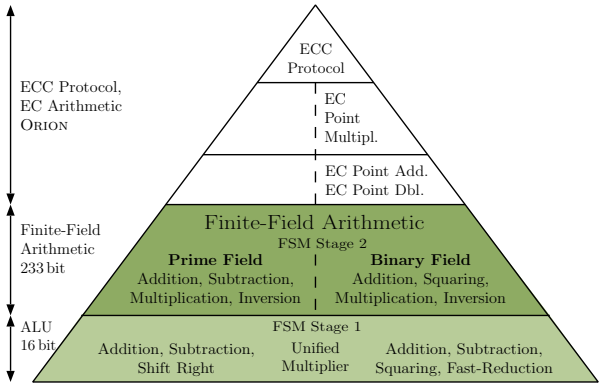

Fig. 3. ECC hierarchy. - The ECC protocol as well as the curve operations are accomplished in software on the OpenRISC utilizing the finite-field arithmetic provided by the co-processor.

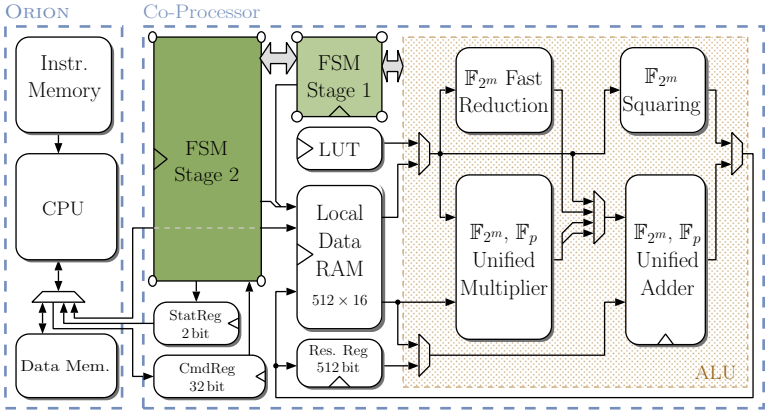

Fig. 4. Architecture of the finite-field arithmetic co-processor supporting elliptic-curve operations based on a binary finite field $\mathbb{F}_{2^m}$ (such as the underlying field of the NIST-B233 standard for the Elliptic-Curve Digital Signature Algorithm (ECDSA)).

For a suitable hardware acceleration of the hash algorithm, the actual algorithm has to be determined in advance. As we tried to keep our system as generic as possible, we decided to calculate the message digest on the OR10N, which allows us to easily adapt the employed algorithm according to our needs. In order to support the required prime-field arithmetic, only minor modifications to the architecture for the $\mathbb{F}_{2^m}$ operations are required. Hence, we decided to include both binary- and prime-field arithmetic into our co-processor. Processing the field operands at their full width of 233 bits contradicts with our initial goal of a resource-aware implementation. As a result, our final co-processor design, as shown in Fig. 4, features a unified 16-bit datapath supporting both $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ operations.

When using the *B-233* elliptic curve, a major amount of computation time has to be spent with completing the $\mathbb{F}_{2^m}$ arithmetic. Therefore, we added a dedicated circuit providing a *Fast Reduction* for the binary field defined by the irreducible polynomial $f(x) = x^{233} + x^{74} + 1$. Along with the reduction block, a *Unified Multiplier* and a *Unified Adder* represent the core-components of our ALU. Moreover, our co-processor contains a dedicated unit for fast *Squaring* in $\mathbb{F}_{2^m}$. This allows to square an operand by simply rewiring the inputs (since squaring is a linear function for binary polynomials), instead of using the multiplication circuitry. In addition, the co-processor has been equipped with a dual-port *Data RAM* to store all field operands locally. This internal memory allows to process two 16-bit words simultaneously and to write them back into the *Result Register* within a single clock cycle.

Since ECDSA as well as other ECC protocols do not only need the arithmetic for the finite-field over which the respective elliptic curve is defined, our architecture supports both binary-field and prime-field operations. As can be seen in Table I, the computation time for some of the finite-field operations adds up to several hundred cycles and beyond. Multiplication in $\mathbb{F}_{2^m}$ represents one of the most time-consuming finite-field operations within ECDSA when using projective coordinates.

A. Multi-Stage FSM Controller

Controlling the datapath of the co-processor is accomplished based on a two-stage finite-state machine (FSM) approach. While *FSM Stage 1* interacts with both the *ALU* and the local memories, the second FSM (*FSM Stage 2*) exclusively interacts with the FSM of the first stage. Moreover, the second FSM determines which 233-bit operands of the data RAM should be processed. In addition to the finite-field operations within $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$, our co-processor offers some utility functions such as moving an operand from one address to another or checking of specific bits whether they are set or not. As a result, the OR10N microprocessor does not have to read a full 16-bit word from the co-processor, but can directly communicate with the *Stage 2 FSM*, thereby saving further clock cycles when executing an ECC protocol.

B. OR10N and Co-Processor Interface

Fig. 4 shows how the tightly coupled co-processor is attached to OR10N. The local data memory and the command register are memory-mapped and reachable within a single clock cycle, allowing the processor to interact with the co-processor without stalling the pipeline. This is achieved with a multiplexer which redirects all signals to the co-processor whenever a co-processor address is decoded.

During an initial phase, a single field element of 233 bits can be copied from the data memory to the co-processor's local memory in 8 consecutive cycles as shown in Fig. 5 by utilizing both write ports of the *Data RAM*. At least one operand needs to be copied to the internal memory in order to compute something on the co-processor. Hence, the initial phase takes at least 8 cycles and the local *Data RAM* requires at least 16 entries. As read and write operations from the OpenRISC processor to our co-processor can be seen as overhead, we tried to minimize them. When performing an elliptic-curve operation based on the targeted finite field, we therefore pre-load our local *Data RAM* with all the operands required to finish the respective ECC operation. To be able to keep multiple operands locally in the co-processor and not to unload them regularly, the *Data RAM* has been designed to hold up to 512 16 bit values, allowing to keep up to 32 field operands in the *Data RAM* of which some are required for temporary variables when performing finite-field operations such as an inversion. To fully pre-load the co-processor, 256 consecutive write operations are required, which is still negligible compared to the finite-field operations which can take up to 35 kcycles.

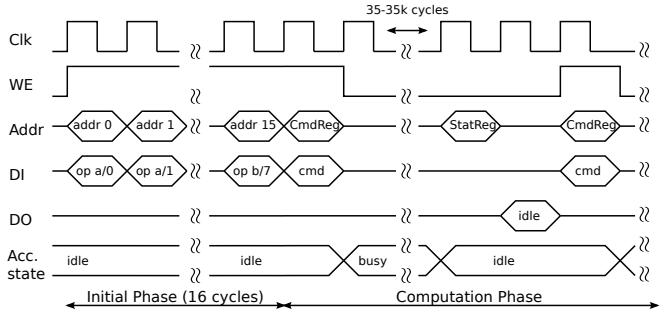

Fig. 5. Timing diagram of the initial phase and the computation phase. Operand a and b are written to the local memory in 16 cycles in the initial phase. In the computation phase, the processor issues a new command to the command register as soon as the co-processor state is idle.

In the computation phase, also illustrated in Fig. 5, the processor writes finite-field instructions to the command register, which triggers the *Stage 2 FSM*. A status register is set indicating that the co-processor is busy and not able to accept further commands. This status register can be directly accessed by the processor since it is memory mapped as shown in Fig. 4. After the instruction is processed, the status register is released and the next command can be written to the co-processor. An ECDSA signature is therefore verified by the execution of an application-specific program on the processor. While the time-consuming finite-field instructions are executed on the co-processor, OR10N can process other workloads.

IV. Results

In the following, performance improvements of our OR10N OpenRISC architecture and a comparison of a software-only implementation with the accelerated design is presented.

A. OpenRISC Performance Results

The redesigned OpenRISC was compared against the original *or1200* design in terms of IPC, critical path, and area. The core architecture of OR10N occupies 39.1 kGE, which is only 8.3 % more than the original implementation. The additional area is due to the 15.5 kGE-large multiplier, which is only pipelined once instead of three times as in the original architecture.

Synthesizing the *or1200* core to the same instruction and data memories results in a critical path of 2.96 ns. The redesign of the core led to a 7.4 % shorter critical path allowing us to operate the core at a frequency of 364 MHz. IPC performance was measured during RTL simulation using different benchmark programs including branch-, computational-, and storage-intensive applications. The benchmarks were coded in C and compiled with the OpenRISC-specific GCC compiler to generate the machine code. The total number of MOPS is the product of IPC and clock frequency and is shown in Fig. 6 for each benchmark application. OR10N processes 344 MOPS on average, which is 67 % more compared to the reference architecture. This improvement is due to a 55 % higher IPC and a 7.4 % higher frequency.

B. Co-Processor Results and Physical Layout

While we described our OpenRISC redesign using SystemVerilog, the finite-field co-processor was implemented in VHDL. Functional correctness and Synthesis was accomplished utilizing ModelSim 10.2a by Mentor Graphics® and Synopsys Design Compiler 2012.06, respectively. The co-processor runs at a maximum frequency of 166 MHz and features a 512×16 bit dual-port RAM, which can hold up to 32 finite-field elements at a time. For the physical layout, which was done using Cadence SoC Encounter, we combined both OR10N and the co-processor resulting in a single design called SIR10US. The final layout of SIR10US, targeting a mature 180 nm CMOS technology, is presented in Fig. 7. Due to limited area on the chip, the data RAM was chosen to be 2 KB. The instruction memory is 8 KB of which 3.8 KB are used to run a full ECDSA signature verification.

TABLE I. Computation time comparison

Operation	OpenRISC† [µs]	Co-Processor [µs]	Co-Processor [cycles]	Speedup
Binary Finite-Field ($\mathbb{F}_{2^m}$)				
Addition/Subtraction	0.29	0.21	35	1.4
Squaring	4.01	0.73	122	5.4
Multiplication	64.44	3.44	574	18.7
Inversion	1'818.42	209.17	34'861	8.7
Prime Finite-Field ($\mathbb{F}_p$)				
Addition§	1.81	0.28	46	6.6
Subtraction	0.97	0.23	38	4.3
Multiplication	459.79	95.57	15'929	4.8
Inversion§	915.80	198.00	33'000	4.6

† Average computation time based on several iterations with varying inputs.
§ Non-constant runtime on the co-processor.

We used C to develop a software-only implementation of the finite-field arithmetic. To compute finite-field operations on the co-processor, a set of assembly functions is provided in a library. Since the command and status register, as well as the *Data RAM* are memory mapped, these assembly instructions are all simple store-word commands. Table I lists the computation times required for the respective finite-field operations on both OR10N and the co-processor. Utilizing the co-processor to compute the finite-field operations leads to a speedup in the range of 1.4 and 18.7 by considering the two different clock frequencies. In addition, a full ECDSA signature verification, targeting the NIST B-233 elliptic curve and using López-Dahab (LD) coordinates [10] for point representation, has been implemented. The correctness of both the C and the hardware implementations of the finite-field operations and the ECDSA algorithm have been verified with the RELIC [1] and OpenSSL toolkit [17]. An ECDSA signature verification on the co-processor requires 1.9 Mcycles of which less than 1 %

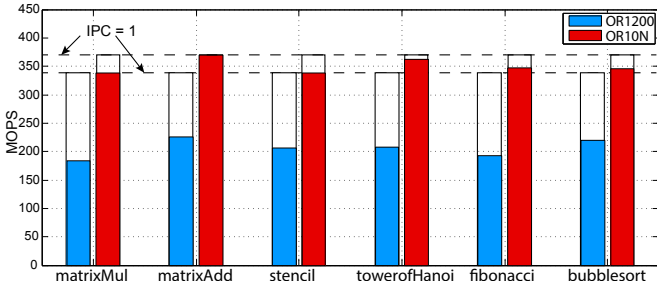

Fig. 6. Performance of OR10N in MOPS compared to the original or1200 architecture. Maximum performance depends on the maximum clock frequency and is reached when the IPC is equal to 1.

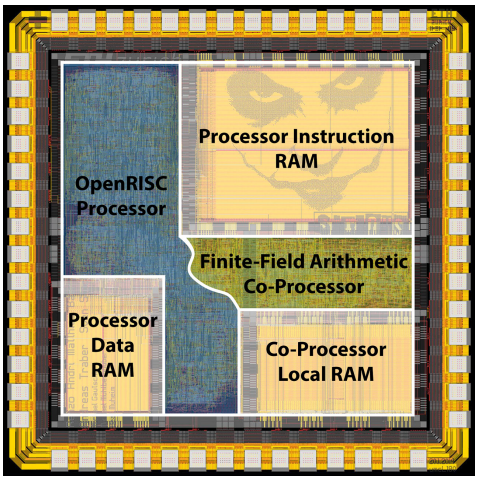

Fig. 7. Final layout of the SIR10US chip showing the different memories and computation engines.

is overhead due to the co-processor interface. At 166 MHz a signature can be verified in 11 ms, which is 15x faster than our software-only signature verification on OR10N running at 364 MHz and 12x faster than on an ARM7 core running at the same frequency. In addition, power analysis with Cadence SoC Encounter has shown that the co-processor accelerated design requires only 1.05 mJ to complete the ECDSA algorithm while a layout without the co-processor requires 21x more energy due to the longer runtime and higher frequency. Even though the combined design increases the area by 37%, it is more energy efficient and allows to further process the data on the main core concurrently.

A more efficient implementation of the ECDSA algorithm has been reported by Turan [18]. The multiplication and reduction routines have been optimized for speed with software engineering techniques and tuned with machine-specific assembler instructions. Implemented on an ARM7 processor, a signature is verified in 6.1 Mcycles. Even though, this implementation is highly optimized and makes use of the full 32 bit multiplier, our accelerated design is 3.3x faster. Extending the datapath of the co-processor to 32 bit would result in another 3-4x speedup for all finite-field operations [16], but on the other hand would also increase the size of the accelerator. Hence, the proposed solution provides a good balance between accelerator area overhead and energy efficiency boost.

TABLE II. DATA SHEET OF THE SIR10US CHIP.

Property	
Area Requirement	109.0 kGE
OR10N Processor	27.6 kGE
Instruction RAM	39.4 kGE
Data RAM	12.3 kGE
ECC Co-Processor	13.6 kGE
Co-Processor Local RAM	16.1 kGE
CMOS Fabrication Technology	180 nm
Maximum Frequency	166 MHz
Core Supply Voltage	1.8 V
Pad Supply Voltage	3.3 V
Power Consumption	93 mW
OR10N-*MOPS*	157

V. CONCLUSION

We proposed SIR10US, an improved design of the OpenRISC processor with an attached ECC co-processor. Balanced pipeline stages and an IPC-optimized implementation of the datapath led to a 67 % higher throughput in terms of MOPS compared to the original architecture. Accelerating the OpenRISC with the co-processor speeds up signature verification according to ECDSA by a factor of 15 with respect to a portable C implementation. Verification on the co-processor is not only faster but also cuts energy dissipation by a factor of 21. Compared to a highly optimized software implementation with machine specific assembler instructions [18], our design is 3.3x faster. Furthermore, our design with a local co-processor memory allows the main processor to execute other workloads during the time-consuming finite-field operations.

ACKNOWLEDGEMENTS.

The work presented in this article has been supported in part by the Swiss Commission for Technology and Innovation (CTI) under project number 13044.1 PFES-ES.

REFERENCES

[1] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. http://code.google.com/p/relic-toolkit/. Version 0.3.5.

[2] P. Biswas et al. Introduction of Architecturally Visible Storage in Instruction Set Extensions. *TCAD'07*, 26(3):435–446.

[3] S. Chang et al. Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition, Nov. 2012.

[4] X. Guo and P. Schaumont. Optimized System-on-Chip Integration of a Programmable ECC Coprocessor. *ACM Trans. Reconfigurable Technol. Syst.*, 4(1):6:1–6:21, Dec. 2010.

[5] A. Hodjat and I. Verbauwhede. Interfacing a High Speed Crypto Accelerator to an Embedded CPU. In *Proceedings of the 38th Asilomar Conf. on Signals, Systems and Computers'04*, pages 488–492 Vol.1.

[6] D. Johnson, A. Menezes, and S. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.

[7] M. Koschuch et al. Hardware/Software Co-design of Elliptic Curve Cryptography on an 8051 Microcontroller. In *CHES 2006*, volume 4249, pages 430–444.

[8] S. Kumar and C. Paar. Reconfigurable Instruction Set Extension for Enabling ECC on an 8-Bit Processor. In *Field Programmable Logic and Application*, volume 3203, pages 586–595. 2004.

[9] D. Lampret. 'OpenRISC 1200 IP Core Specification'. 2010.

[10] J. López and R. Dahab. Improved Algorithms for Elliptic Curve Arithmetic in $GF(2^n)$. In *Selected Areas in Cryptography*, volume 1556, pages 201–212. 1999.

[11] NIST. *Secure Hash Standard (SHS) (FIPS PUB 180-4)*. National Institute of Standards and Technology, Mar. 2012.

[12] NIST. *Digital Signature Standard (DSS) (FIPS PUB 186-4)*. National Institute of Standards and Technology, July 2013.

[13] OpenCores Community. OpenRISC Community Portal, 2013. [Online; accessed 6-February-2014].

[14] G. Orlando and C. Paar. A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$. In C. K. Koç and C. Paar, editors, *CHES 2000*, volume 1965, pages 41–56.

[15] S. Ravi et al. Security in Embedded Systems: Design Challenges. *ACM Trans. Embed. Comput. Syst.*, 3(3):461–491, Aug. 2004.

[16] A. Satoh and K. Takano. A Scalable Dual-Field Elliptic Curve Cryptographic Processor. *IEEE Computers*, 52(4):449–460, Apr. 2003.

[17] The OpenSSL Project. OpenSSL: The Open Source toolkit for SSL/TLS. Version 1.0.1e.

[18] E. Turan. ECDSA Optimizations on ARM Processor for a NIST Curve over GF (2m). Master's thesis, Oregon State University, 2001.