Extending The Instruction Set Of RISC-V Processor For ASCON Algorithm

Yunus Emre ERYILMAZ[1], Berna ÖRS[1]
[1] Istanbul Technical University
Faculty of Electric and Electronics, Department of Electronics and Communication Engineering
First.eryilmazy18@itu.edu.tr; Second.orssi@itu.edu.tr

ABSTRACT

Lightweight cryptography (LWC) is an important research area at a time when constrained devices are on the rise. The well-known asymmetric and symmetric cryptography algorithms such as AES and RSA are not suitable for these devices. So, the lightweight cryptography field has emerged. Speed and memory usage are the most important attributes of LWC algorithms. ASCON lightweight cryptography algorithm is one of the finalist algorithms in the standardization process of the NIST and CEASER competition. In this study, the instruction set architecture of Ibex core, a RV32IMC processor has been extended for the ASCON algorithm. The algorithm has been run on the processor and observed the execution time, size of instruction memory, the average energy consumption and the area coverage. The code has been profiled on the SPIKE RISC-V ISA simulator and the most frequent operations have been chosen. The chosen operations have been designed with Verilog HDL, the modules instantiated at the processor and the same measurements are done with the custom instructions. Thus, the execution time of the algorithm is decreased by 30.2%, the memory size is decreased by 2.3%, the energy consumption while running the algorithm is decreased by 41.84%, and the number of look-up tables and flip-flops in the modified processor are increased by 18.2% and 1.94% respectively.

***Keywords*:** Lightweight Cryptography, Custom Instruction, RISC-V, Open-source Hardware, Instruction Set Extension

INTRODUCTION

Usage of constrained devices such as RFID (Radio-Frequency Identification) tags, sensors, microcontrollers and smartcards is increasing every year. This situation brings new security concerns. However, applying cryptographic algorithms is challenging because conventional cryptographic algorithms are not suitable for constrained devices. The algorithms are optimized for systems with high computing power like desktop and server environments. If the conventional algorithms run on constrained devices, they work in low performance. So, lightweight cryptography has emerged. National Institute of Standards and Technology (NIST) started to research standardization of lightweight cryptographic algorithms in 2013 due to insufficiency of current NIST-approved cryptographic standards. They opened a competition for adding a public comment to the standardization process in 2018. ASCON is one of the competitor algorithms and finalists. ASCON is chosen for the project because most of its operations can be implemented on FPGA easily due to behaviour of the operations.

Speed is the most important thing about lightweight cryptography. There are several ways to speed up the process. The first of them is optimization of the algorithm, but it is not feasible and an easy way to do it. Another of them is using a more powerful processor. However, the processor requires a wider area and more power consumption if its computing power gets higher. For this environment, an application specific processor is a more appropriate way. In this study, this way has been aimed. A RISC-V based core, Ibex is selected to implement the algorithm because Ibex is a fast and small open-source processor and supports I, M and C extensions.

In the second part, the background information about ASCON and Ibex core are given. Details for the software implementation of ASCON on Ibex are given in the third part. The instruction set extension of the processor are described in the fourth part. The results are given and discussed in the fifth part and the sixth part concludes the study.

BACKGROUND INFORMATION

Ascon

ASCON is a family of cryptographic algorithms which consist of authenticated encryption and hashing. It is designed to be lightweight and easy to implement on constraint devices with countermeasures for side-channel attacks. Also, it is selected for the final portfolio of the CAESAR competition, and a finalist competitor in the ongoing NIST Lightweight Cryptography competition (Web-9).

In authenticated encryption, there are encryption and decryption algorithms. Encryption algorithm gets a maximum 160-bit key (K), 128-bit nonce (N) and associated data (A) of arbitrary length to encrypt plaintext (P) of arbitrary length and produce ciphertext (C) of the same length as plaintext and 128-bit tag (T). Decryption algorithm gets key, nonce, associated data, ciphertext and tag, which are the same length as that used in encryption, to decrypt ciphertext and produce plaintext or a verification tag (⊥) if the verification of tag fails. Encryption and decryption algorithms can be seen in Figure 1 and 2, respectively.

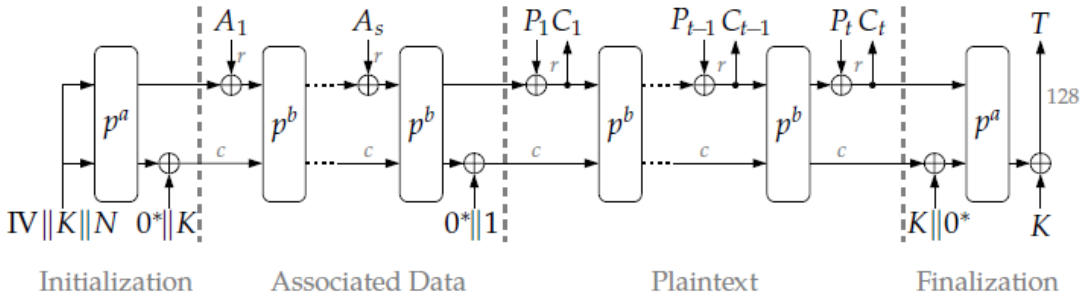

Figure 1. Block diagram of encryption algorithm (Dobraunig et al., 2019).

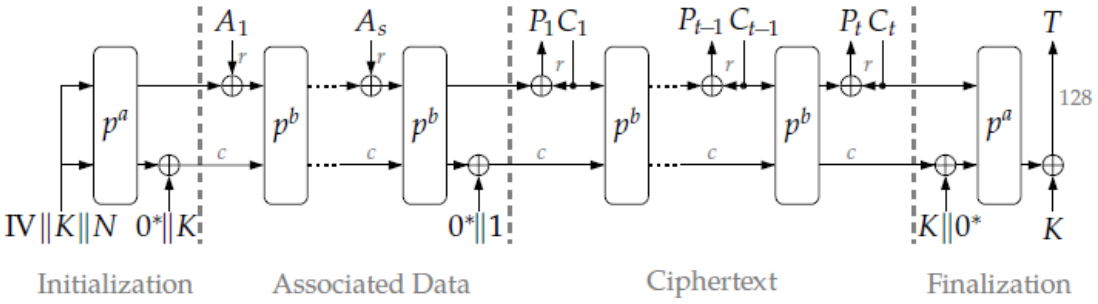

Figure 2. Block diagram of decryption algorithm (Dobraunig et al., 2019).

ASCON encryption algorithm consists of four stages: Initialization, processing associated data, processing plaintext/ciphertext and finalization. In the initialization, initialization vector (IV) and 320-bit state of the sponge construction (S) are created by using key (K) and the nonce (N). Optional padding is applied to state (S) by XORing it with associated data (A) in the processing of associated data. The padding can be skipped when the associated data is not used. The plaintext is XORed with state to produce ciphertext while processing ciphertext. In processing plaintext, ciphertext is XORed with state to produce plaintext. The key is XORed with state to generate authentication tag in the finalization stage.

Round permutation is a function which is used in every stage of the algorithm. It gets 320-bit state (S) as input parameter and splits five equal 64-bit registers as x0, x1, x2, x3, x4. The permutation consists of three stages: Addition of constants, substitution layer, linear diffusion layer. A constant value is XORed with the x2 register and assigned to the x2 in the addition of constants. State (S) splits into 64 pieces that are 5 bits long, and the pieces are updated with the s-box function in the substitution layer. Substitution layer operations can be seen in Figure 3. Linear diffusion layer updates every register by shifting left. Shifting amount is different for every register. Linear diffusion layer operations can be seen in Figure 4.

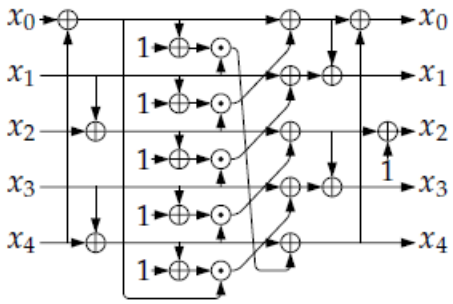

Figure 3. Logical operations of substitution layer (Dobraunig et al., 2019).

$$x_0 \leftarrow \Sigma_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)$$
$$x_1 \leftarrow \Sigma_1(x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)$$
$$x_2 \leftarrow \Sigma_2(x_2) = x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6)$$
$$x_3 \leftarrow \Sigma_3(x_3) = x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17)$$
$$x_4 \leftarrow \Sigma_4(x_4) = x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)$$

Figure 4. Linear layer operations (Dobraunig et al., 2019).

Ibex RISC-V Core

Ibex is an open-source 32-bit RISC-V core that supports I, M, and C extensions. It is written in SystemVerilog and consists of parametrizable blocks. Also, it is suitable for embedded control applications. The architecture of Ibex core is shown in Figure 5.

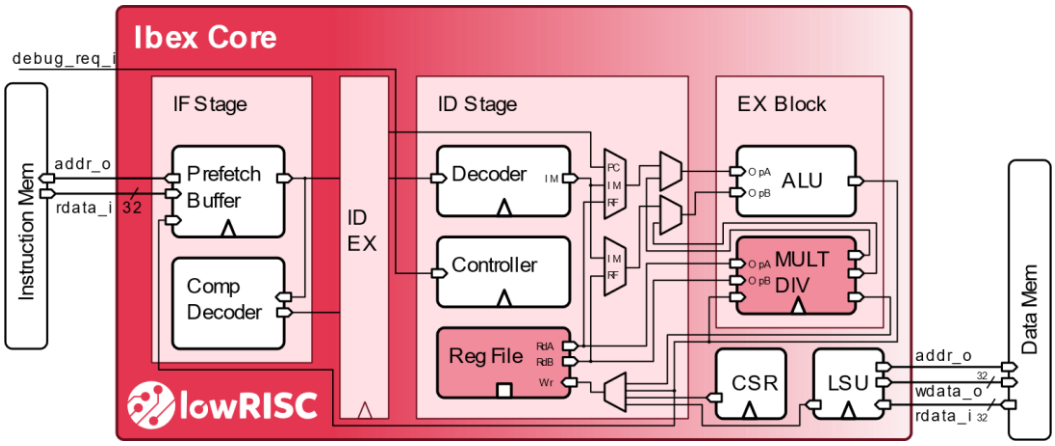

Figure 5. Architecture of Ibex with a 2-stage pipeline (lowRISC,2022).

SOFTWARE IMPLEMENTATION OF ASCON ON IBEX

The source code from the official ASCON Github repository is used in the software implementation and the profiling (Web-4). ASCON-128av variation is used because Ibex is a 32-bit processor. The key, the nonce, the tag, and the data block are 128-bit, a is 12, and b is 8 for this variation.

Software Implementation

The source files call encryption function and generates a ciphertext. Then, it decrypts the ciphertext and creates the same plaintext that used in the first place. The files are compiled for RV32I ISA with RISC-V GNU compiler (Web-6) and run a behavioural simulation on Ibex with Vivado Logic Simulator (Web-7). Artix-7 100T FPGA (Web-10) is used in the implementation and the simulation. The compiler generates .vmem file to run on processors and it contains from 4251 lines. The encryption and the decryption take 66952 and 66198 clock cycles, respectively.

Also, Ibex core covers 2977 LUTs, 1955 FFs and its maximum frequency is 124.27 MHz in this implementation.

Profiling for Finding the Most Frequent Operation

C code of the encryption algorithm is profiled by using SPIKE RISC-V ISA Simulator (Web-5) and RV32I instruction set. The decryption algorithm did not profile because it is mostly the same as the encryption algorithm. The profiling is done previously by Altinay and Ors (2021). Profiling results of the encryption algorithm is shown in Figure 6.

Optimization	Clock count of Permutations	Total Clock Count	Percentage (%)
-Os	15789	29138	54,2
-O3	10656	22695	47
-O2	10656	22694	47
-O1	10656	22684	47
-O0	26694	71877	37,1

Figure 6. Number of clock cycles of permutation and encryption functions (Altınay, 2021).

Five optimization flags are used in profiling and they are named as -O0, -O1, -O2, -O3 and -Os. The compiler makes no optimization on the code with -O0 flag. The -O1 flag makes optimization on the small functions to optimize code size and execution time. The -O2 flag makes all supported optimization for the code but does not change space-speed balance. -O3 does the same optimizations as -O2 and does more optimizations than it. -Os optimizes to only reduce the code size. According to Figure 6, the permutation function is used frequently.

The profiling results of the round permutation function is shown in Figure 7. Figure 7 states that the linear layer and the substitution layer are the most time-consuming operations in the permutation function.

Optimization	Round constant addition	Substitution layer	Linear layer
-Os	2,5%	45,2%	52,3%
-O3	2,5%	45,2%	52,3%
-O2	2,5%	45,2%	52,3%
-O1	2,5%	45,2%	52,3%
-O0	3,6%	48,7%	47,7%

Figure 7. Number of clock cycles of the layers in the permutation function (Altınay, 2021).

Thus, s-box and round functions must be designed as a custom module for the ALU.

CUSTOM INSTRUCTION SET EXTENSION OF IBEX

Custom Modules

One of the most frequent functions is round function. It shifts all bits of the number right as shifting amount and fills vacant bits with the least significant bits. It gets two parameters: 32-bit shifting amount and 32-bit number to shift. The block diagram of round function is shown in Figure 8.

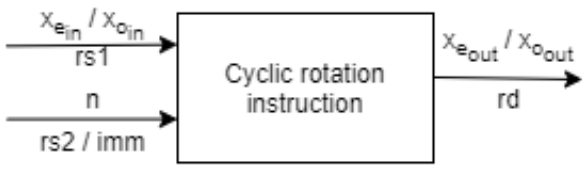

Figure 8. Round block diagram (Altınay, 2021).

The other most frequent function is the s-box function. There are two arrays that are named even and odd states in the software implementation of the ASCON algorithm. The custom module does the s-box operations to one of them. The second operand is not used and the output is always zero. The module takes the address of the first element of the input array and holds it. After that, it loads all elements of the array to the module by using the direct memory access to the block RAM and makes the s-box computations. Finally, it loads the outputs of the function to the input array. The block diagram of s-box is shown in Figure 9.

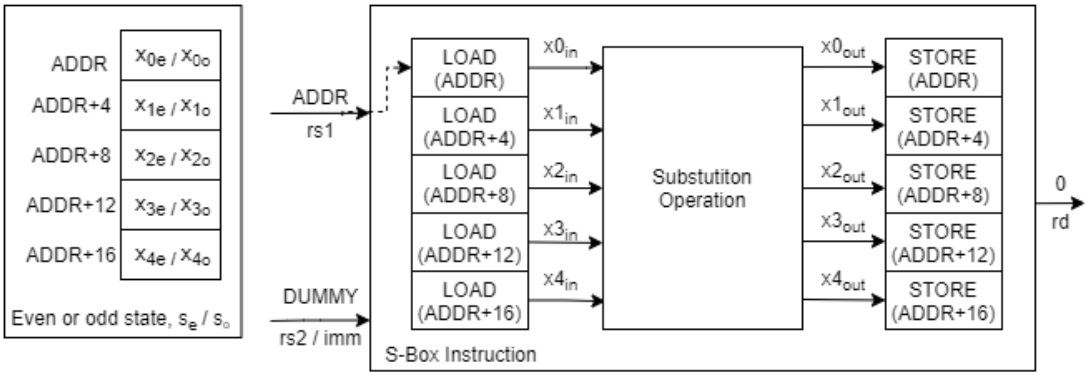

Figure 9. S-box block diagram (Altınay, 2021).

Changes in Software

While adding the instructions, the compiler and ALU are modified to detect them. The functions are named cust0 and cust2 in the compiler for simplicity. There are two files named "risc-v-opc.c" and "risc-v-opc.h" that contain instruction templates in the riscv-gnu-compiler folder. "risc-v-opc.c" and "risc-v-opc.h" files must be modified to add instructions. "risc-v-opc.c" file contains the list of opcodes and their properties. The template and properties of new instructions will be written into the riscv_opcodes array in the file. "risc-v-opc.h" contains definitions for "risc-v-opc.c" file. The MATCH and MASK values are defined and associated with the instruction in the file. Every instruction is decoded by using these values. If bitwise ANDing of the MASK value of cust2 and the incoming instruction is equal to the MATCH value of cust2, the incoming instruction is detected as cust2. After the modifications, generated compiler files must be deleted and the compiler must be reinstalled.

Also, inline assembly statements are added to the ASCON source files to call added custom instructions. The inline assembly statement to call cust2 instruction is shown in Figure 10. Inline assembly is a method to call low-level language statements in a script that is written in a high-level language (Web-1). Using inline assembly increases optimization, decreases function-call overhead, and allows to use of CPU-specific instructions. However, inline assembly makes the maintenance and the readability of the code harder. The code with inline assembly cannot be cross-platform because the used instructions can be CPU-specific, and they cannot run on the other platforms. Any compiler can handle the inline assembly differently and this can cause a decrease in performance.

```
asm volatile("cust2 %[result0], %[value1], %[value2]\n\t":
  [result0] "=r" (r0) : [value1] "r" (&s_e[0]) , [value2] "r" (1)
);
```

Figure 10. Inline assembly statement to call s-box instruction.

Changes in Hardware

Modifications in processor architecture are done in the decoder and the algorithmic logic unit (ALU) while adding the round module. The round function is a single cycle operation, so it can be instantiated in parallel to the other ALU operations. The decoder must be modified to recognize the round instruction. So, a new case was added to the case structure in the OPCODE_OP case. The round function is instantiated in ALU. operand_a_i and operand_b_i are the inputs, cust0_result is the output of the custom module. Also, the result mux is modified to conduct the result of the custom module.

The s-box module is instantiated in the execution stage, in parallel to the multiplication module because it does a multi-cycle operation. Appropriate input and output signals are created to operate the module in the execution. The inputs and outputs of the stage are changed to control block RAM and get data from the other modules. The decoder and controller in the instruction decode stage are also modified. A new case for the custom instruction is added to the decoder to enable the custom module and the register file. An output signal is added to the controller to stall the processor while waiting to finish multi-cycle operations. The core will continue to operate when the operation of the custom module is finished. If the core did not enter a waiting state, it can produce wrong results.

Source operand and enable output signals from the ID stage are connected to source operands and enable input signals of the EX stage in the ibex_core module. RAM data, RAM address and the enable signals are added as inputs and outputs of the core module to communicate with RAM. The new core signals are connected to block RAM in the top module. The modified execution block with added modules is shown in Figure 11.

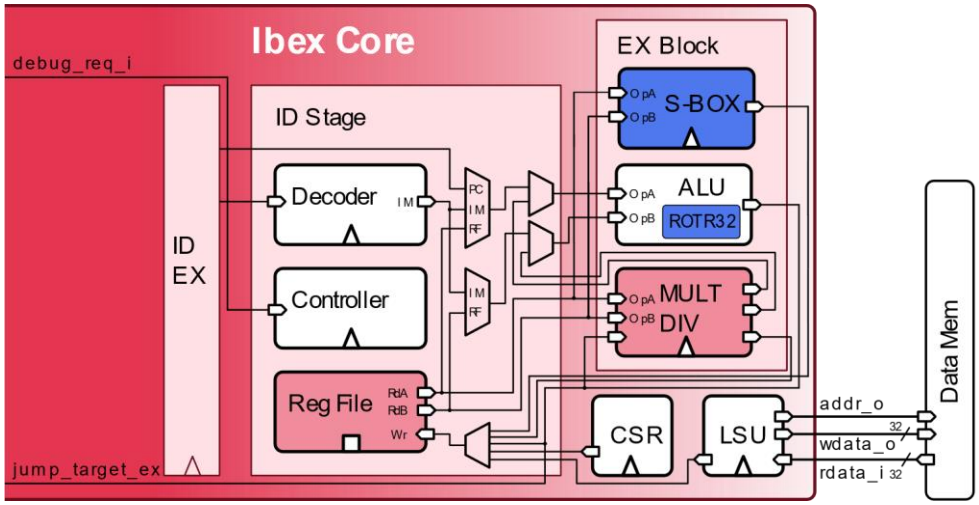

Figure 11. Modified execution block of the processor.

RESULTS

After modifying the core, the compiler and the source codes, the encryption and the decryption algorithms are run on the core and the implementation are compared. The measurements are taken on Xilinx Artix-7 100T FPGA (Web-10) with Vivado 2018.1 (Web-11).

The area measurement is made by implementing unmodified and extended Ibex core and comparing the utilization reports of them. Areas covered by unmodified and extended Ibex is shown in Table 1. The amount of FFs and LUTs are increased by 1.94% and 18.9% respectively because s-box and ROTR32 modules are added to the processor.

Table 1. Area comparison between two cores.

	Vanilla	**Extended**	**Change**
LUT	2977	3542	+18.97%
FF	1955	1993	+1.94%

Instruction memory sizes are compared by examining the number of lines in the generated vmem files. The comparison is shown in Table 2. The program with custom instruction has a smaller number of lines than the other one since the round function without custom instruction is 216 instructions and the same function with custom instruction is 59 instructions.

Table 2. Instruction memory size comparison between two cores.

	Vanilla	Extended	Change
Lines	4251	4153	-2.3%

Execution times of the programs are measured by measuring the execution of the main function takes how many clock cycles in the behavioural simulation of Vivado Logic Simulator (Web-7). The comparison between the two programs is shown in Table 3. The program with custom instructions takes less time to complete because the instruction memory size of the program with custom instructions is smaller and the custom module operations take less time than the ALU operations.

Table 3. Execution time comparison between two cores.

	Vanilla	Extended	Change
Number of clock cycles	133150	92830	-30.2%

Switching Activity Interchange Format (SAIF) file is needed to measure the average energy consumption in the processor, and it is generated after running a post-implementation timing simulation. SAIF stores data about static probability ratios and toggle rates of the wires in the device (Web-2). Static probability is the ratio of how much time the net is HIGH to the device operation time (Web-3). The power report for the processor is generated by using the SAIF file in Vivado (Web-8) and dynamic power can be learned with power reports. Average energy consumption is the multiplication of the execution time, the clock frequency, and the dynamic power of the processor. The average energy consumption comparison of the program with and without custom instructions is shown in Table 4.

Table 4. Average time consumption comparison between two cores.

	Vanilla	Extended	Extended
Average dynamic power [W]	0.228	0.196	-14.03%
Minimum clock period [ns]	7.859	7.645	-2.72%
Average energy consumption [mJ]	0.239	0.139	-41.84%

The program with custom instructions consumes less energy than the program without custom instructions because the execution time is shorter than the other program. So, switching activity and the energy consumption is lesser too.

CONCLUSION

Lightweight cryptography is an important field for security at a time when the usage of constrained devices is increasing. ASCON is one of the lightweight cryptography algorithms and is close to being a standard for LWC. It is a fast and efficient algorithm even in its software implementation and it can be more resourceful with instruction set extension. To show it, the C program for ASCON is firstly compiled and run on Ibex core. After observing the whole program running on the core, profiling of the program is made, and the most frequent operations are decided. Then, equivalent modules are designed in Verilog language and instantiated into the core. Finally, the execution time, the average energy consumption, the instruction memory size, and the area usage are measured and compared. According to the results, the area usage increased but the execution time, the average energy consumption, and the instruction memory size is decreased. Therefore, the ISA extension has increased the performance of the algorithm. The future work will be on porting the compiler for more optimized results without the need for inline assembly statements and implementing a communication module to test the system more efficiently.

REFERENCES

Altınay Ö. (2021). Instruction Extension of RV32I and GCC Back End for ASCON Lightweight Cryptography Algortihm (Master's thesis, Dept. Electron. and Comm. Eng., Istanbul Technical Univ., Istanbul, Turkey). Available from CoHE Thesis Center. (Thesis No. 682851)

Altınay Ö. and Örs, B. (2021). Instruction extension of RV32I and GCC back end for Ascon lightweight cryptography algorithm, 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), August 23-25, 2021, pp. 1-6.

Dobraunig C., Eichlseder M., Mendel F., and Schläffer M. (2019). *Ascon v1.2. Submission to the NIST Lightweight Cryptography competition*, http://ascon.iaik.tugraz.at.

lowRISC. (2022). *Ibex Documentation*. Retrieved from: https://ibexcore.readthedocs.io/downloads/en/latest/pdf/.

McKay K., Bassham L., Sönmez Turan M., and Mouha N. (2016), *Report on lightweight cryptography* (No. NIST Internal or Interagency Report (NISTIR) 8114 (Draft)), National Institute of Standards and Technology.

Turan M. S., McKay K. A., Çalik Ç., Chang D., and Bassham L. (2019), *Status report on the first round of the NIST lightweight cryptography standardization process*, National Institute of Standards and Technology, Gaithersburg, MD, NIST Interagency/Internal Rep. (NISTIR).

Waterman A. S. (2016). *Design of the RISC-V instruction set architecture* (Doctoral dissertation, Dept. Elect. Eng. and Comp. Sci. Univ. of California, Berkeley, CA, USA). Retrieved from https://digitalassets.lib.berkeley.edu/etd/ucb/text/Waterman_berkeley_0028E_15908.pdf.

Waterman A., Lee Y., Patterson D. and Asanović K. (2016). *The RISC-V Instruction Set Manual, Volume I: User-Level ISA Version 2.1*. (Technical Report UCB/EECS-2016-118). Retrieved from UC Berkeley The Department of Electrical Engineering and Computer Sciences website: http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html

Waterman A., Lee Y., Patterson D., and Asanovic K. (2019). *The RISC-V Instruction Set Manual, Volume I: User-Level ISA.* (Document Version 20191213). Retrieved from RISC-V International website: https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf.

Web sites:

Web-1: https://gcc.gnu.org/wiki/DontUseInlineAsm, accessed 19 February 2022.

Web-2: https://www.synopsys.com/designware-ip/technicalbulletin/understanding-power-profile.html, accessed 10 February 2022.

Web-3: https://d2pgu9s4sfmw1s.cloudfront.net/DITA-technicalpublications/PROD/PSG/ug-qpp-power-683174-709286.pdf, accessed 11 January 2022.

Web-4: https://github.com/ascon/ascon-c, accessed 10 January 2022.

Web-5: https://github.com/riscv-software-src/riscv-isa-sim, accessed 4 January 2022.

Web-6: https://github.com/riscv/riscv-gnu-toolchain, accessed 15 March 2022.

Web-7: https://www.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx2020_1/ug900-vivado-logic-simulation.pdf, accessed 14 February 2022.

Web-8: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug907-vivado-power-analysis-optimization.pdf, accessed 25 March 2022.

Web-9: https://csrc.nist.gov/Projects/lightweight-cryptography/finalists, accessed 11 February 2022.

Web-10: https://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf, accessed 4 January 2022.

Web-11: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug893-vivado-ide.pdf, accessed 7 February 2022.