Design Exploration for SHA-3 Algorithm in FPGAs

Florêncio Natan dos Santos Gama
Departamento de Computação, Universidade Federal de Sergipe, São Cristóvão, Sergipe
florencionatan@gmail.com

Edward David Moreno
Departamento de Computação, Universidade Federal de Sergipe, São Cristóvão, Sergipe
edwdavid@gmail.com

***Abstract*—This paper studies the performance of SHA-3 (Secure Hash Algorithm version 3) hardware implementations. One of them focused on high-speed and the other for low area. So, we discuss some techniques which were chosen for their implementation and the main influences that such techniques brought to the algorithm. We show in this paper a study with statics of consumption area and performance of this algorithm (Keccak from SHA-3) using some of the most used tools in the hardware community (Quartus and Cyclone II board from Altera).**

***Index Terms*—**SHA-3, Keccak Algorithm, FPGA, Performance.

1 INTRODUCTION

Cryptographic hash is a class of cryptographic functions that are used when we are dealing with the authenticity and integrity of data. The hash can reduce a message to a single output consisting of a sequence of bits of fixed size, which is typically smaller than the original message [1]. SHA-3 was contest organized by NITS (National Institute of Standards and Technology) between 2007 and 2012 with the object of choosing the new standard cryptographic hash to be used in the coming years [2]. The algorithm chosen was the Keccak, one of its main advantages over competitors was its excellent performance in hardware [3].

2 KECCAK ALGORITHM

The Keccak algorithm was created by Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche for participation in the SHA-3 contest. The sponge Keccak is based on functions, which is a generalization of the concept of cryptographic hash function with infinite output and can perform quasi all symmetric cryptographic functions, from hashing to pseudo-random number generation to authenticated encryption[4]. The Keccak is an easily configurable algorithm, an important characteristic is the variable size of its permutation. The size of its permutation is called b, where b ∈ {25, 50, 100, 200, 400, 800, 1600} [5]. Immediately following the entry, the current data block is divided into a three-dimensional array with 5 x 5 x w, the w is the word size and equals b/25. After is performed a function called Round, it is an operation performed on the data matrix N times. The number N of times the Round runs is set by formula $12 + 2 * \log_2 W$, with 24 the highest possible value for N, this only occurs when b is equal to 1600 bits, in this case we can see that the word size is 64 bits. The function Round[b](A, RC) performs five steps called θ (theta), ρ (rho), π (pi), χ (chi) and ι(iota), [5]. The Figure 1 has Keccak pseudocode.

$Round[b](A,RC)$
θ *step*
$\forall x \text{ in } 0\ldots4$
$C[x]=A[x,0]\oplus A[x,1]\oplus A[x,2]\oplus A[x,3]\oplus A[x,4]$
$\forall x \text{ in } 0\ldots4$
$D[x]=C[X-1]\oplus ROT(C[X+1],1)$
$\forall (x,y) \text{ in } (0\ldots4,0\ldots4)$
$A[x,y]=A[x,y]\oplus D[X]$

ρ and π *steps*
$\forall (x,y) \text{ in } (0\ldots4,0\ldots4)$
$B[y,2x+3y]=ROT(A[x,y],r[x,y])$

χ *step*
$\forall (x,y) \text{ in } (0\ldots4,0\ldots4)$
$A[x,y]=B[x,y]\oplus((NOTB[x+1,y]) \text{ AND } B[X+2,y])$

ι *step*
$A[0,0]=A[0,0]\oplus RC$

returnA

Figure 1: Pseudocode of Keccak[5]

In pseudocode above AND and NOT are respectively the conjunction and negation logic boolean operation. The symbol ⊕ represents the exclusive disjunction operation in boolean algebra. ROT(M,r) is a binary circle rotation function, which shifts the bits of M r positions. The arrays of values r[x, y] and RC are constants used by the function, size RC can change according the size of the data block [5]. The A array is the current state of hash, B and C are local temporary variables.

3 HARDWARE IMPLEMENTATION

The creators of Keccak implement and provide three architectures to reflect different trade-off for it's algorithm a high-speed core, a low-area co-processor and a mid range core [5]. This study covers the high-speed and low area in its scope, and both are totally written in VHDL (Very High Speed Integrated Circuit Hardware Description Language). The two versions analyzed here are available for download on the algorithm's page.

3.1 Main Project focused in Low area

The Keccak low area co-processor is suitable for smart cards or wireless sensor networks where area is particularly important since it determines the cost of the device and there is no rich operating system allowing to run different processes in parallel [5].

It also needs external memory, the co-processor is capable to store only a small amount of data that is being used at the moment. Its design consists of six source files, below is a brief description of each:

- "fsm.vhd" - File that contains the implementation of the state machine of the chip, responsible for determining the order in which each operation will be performed;
- "pe.vhd" - Contains the 5 defined functions by Keccak θ, ρ, π, χ, ι;
- "keccak_copro.vhd" - File that defines the chip Keccak itself, it is an encapsulation of entities defined in two files above;
- "system_mem.vhd" - Contains a simple implementation of memory for the system where the algorithm will run;
- "keccak_globals.vhd" - Type definitions and global variables used by all documents;
- "tb_keccak_copro.vhd" - File that contains a test bench for Keccak the test bench works by reading data from a file that contains a number of entries to be processed.

Now we mainly analyze the components that are defined in "fsm.vhd" and "pe.vhd" because it is in them that is defined the core of SHA-3.

3.1.1 fsh.vhd

Here we can find described the state machine that controls the work of the chip. It consists of 22 states, a state of initialization, 7 states that read data from memory and perform the first half of the function θ, 7 states that perform the second half of the function θ and process ρ and π, finally more 7 states that process functions χ and ι then store the data. Also here is defined the function π.

As known by hardware community, when a state machine is too extensive it can force the algorithm to reuse the most of the circuits responsible for permutation in this way saving area, but making slower processing. Another decision that also influences this is the fact that the data is transferred by entries on the size of the word rather than the buffer size.

3.1.2 pe.vhd

This file defines the code permutation, it contains 4 of the 5 functions that compose the Keccak and they are θ, ρ, χ, ι. The communication of this module with fsm is made by a gate called command, composed of seven bits where the bit currently active in '1' indicates which operations will be performed, we can see in the Table 1 what the meaning of each bit. All functions are implemented concurrently so that the command is responsible for assigning the correct value to the output gate.

Table 1: Meaning of the command bits

ACTIVE BIT	ACTION
0	READS DATA FROM MEMORY TO AN INTERNAL REGISTER.
1	PERFORMS THE FIRST HALF OF θ.
2	PERFORMS THE SECOND HALF OF θ.
3	SWAP DATA FROM INTERNAL REGISTERS.
4	WRITE THE VALUE OF REGISTER 1 IN THE MEMORY.
5	RUN P
6	RUN X
7	RUN I

3.2 Main Project Focused in High-speed

This version of Keccak written in VHDL, was designed to have excellent execution performance. In the third round report of SHA-3 competition we can see that Keccak has the better performance than other competitors and its predecessor SHA-2 (Secure Hash Algorithm version 2), being the only one of them considerably better than SHA-2[6]. It can be used in any performance-critical applications. Different from the previous model that needs additional memory, the high-speed implementation does not need any additional storage just gets the input block and processes it. This project contains seven source code files. They will be listed below, as well as their functionality:

- "keccak.vhd" - Chip that encapsulates the full function contains the following parts keccak_round, keccak_buffer, keccak_round_constants_gen;

- “keccak_buffer.vhd” - Component that stores the buffer data for the execution of the algorithm;
- “keccak_globals.vhd” - Type definitions and global variables used in all documents;
- “keccak_round.vhd” - Here we have implemented the 5 internal functions by the algorithm;
- “keccak_round_constants_gen.vhd” - Chip that contains a function for generating the constants used during the execution of ι;
- “tb_keccak_permutation.vhd” - Test bench used to check the operation of the algorithm;
- “tb_keccak.vhd” - Another test bench that comes with the algorithm

Now we analyze the components defined in “keccak_buffer.vhd”, “keccak_round.vhd” and “keccak_round_constants_gen.vhd”.

3.2.1 keccak_buffer.vhd

This is the data buffer that feeds the algorithm in this version. It has two modes of operation: data input, where it reads data from the input until the buffer is complete and data output where it transfers its data to the Keccak round.

3.2.2 keccak_round_constants_gen.vhd

Here are generated the constants used in ι step. Its operation is very simple, given one entry that corresponds to the round number that is being executed, the number must be between 0 and 23, then it returns the value corresponding to the round constant.

3.2.3 keccak_round.vhd

Here is where the co-processor algorithm is indeed. All functions are implemented as described. In this case the output of the current function is connected to the input of the next function, thus ordering the execution of the algorithm. To create a series of parallel components the designer chose to use the construction "for - generate" so that all words were processed in parallel.

4 SIMULATION TOOLS

This section we present the tools used to measure our results. Two tools were used, the simulator provided by Mentor Graphics ModelSim to assess the performance of the algorithms on the execution time and Altera Quartus used to measure the area. The results of area were measured considering a FPGA (Field-Programmable Gate Array) model Cyclone II from Altera also.

4.1 ModelSim

ModelSim is the market leading tool for the simulation of ASIC and FPGA devices. Created by Mentor Graphics it has several versions for both Linux and windows. It has many attractive features as advanced code coverage, mixed HDL (Hardware Description Language) simulation, effective debug environment and support for the major HDL[7].

4.2 Quartus II

Quartus II is a tool for development and synthesis HDL created by Altera. The Quartus II supports CPLDs (Complex Programmable Logic Devices) and FPGAs made by Altera, allowing the creation of circuits and test their embedded simulation tools, analysis of RTL (Register-transfer Level) diagrams and finite-state machines. It is also essential to work on the boards of the company [8].

5 SIMULATION AND RESULTS

Now will be explained the simulations and results obtained with Keccak, simulations of used area were based on the FPGA model Cyclone II from Altera, while the time performance was obtained with the ModelSim tool.

5.1 Execution time

For this test simulations were performed where the algorithm had to individually apply the hash over ten different data blocks and return values obtained. Every block contains 1600 bits and therefore with a total processing 16000 bits, about 2 kilobytes of information. Values were placed in the same text file that both algorithms used, on the Table 2 we can see the comparison between the time spent to complete the task:

Table 2: Execution time

VERSION OF ALGORITHM	Execution times (ns)
LOW AREA	1042820
HIGH-SPEED	5020

As would be expected the performance difference is very large, with high-speed coming up to 200 times faster. We can see the difference better in Figure 2.

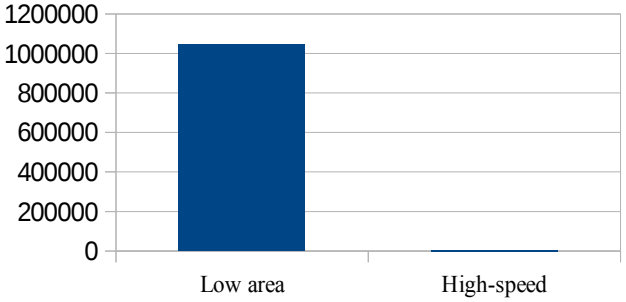

Figure2: Execution time

5.2 Area consumption

Here the tests were done to measure the area occupied by the algorithm. We did two comparisons in the first one compared the algorithms just as the area occupied by Keccak, in the second comparison we include also the area of memory used by the algorithm of low area during its processing.

The Table 3 show the obtained results of the comparison. And the Figure 3 show the graphical version of these values.

Table 3: Area Consumption

	Low area	High-speed	Low area plus additional memory
Total logic elements	1595	5191	6734
Total combinational functions	1572	3838	4579
Total registers	242	1671	4548
Total pins	140	130	130

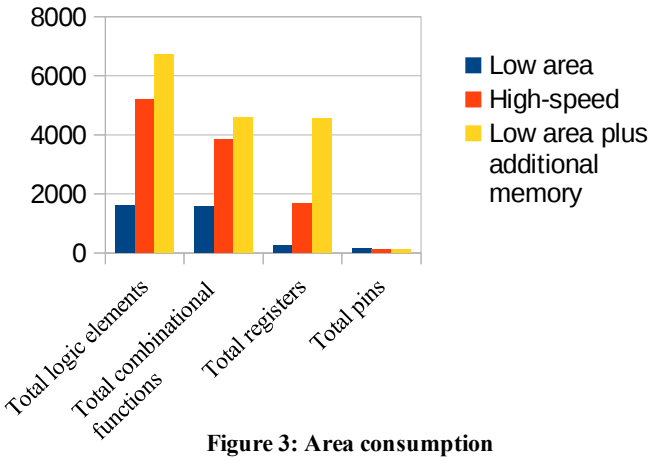

Figure 3: Area consumption

The comparison between high-speed and low area also was not too surprising. Having an area more than 3 times smaller, version low area really delivers what it promises, even losing in the time it gets an incredible reduction in area, a trade-off that can be good.

But the biggest surprise was when we analyze the low area with the additional memory it needed to run, it eventually generate a greater amount of area that high-speed. Importantly, in a real application this memory can be shared between several algorithms, other words, this version still meets the goals of their implementation, the Figure 3 shows a comparison of these results.

We can see that the number of register used in it grows considerably. Its occurs because the memory was implemented with a array of registers which has the size of a word, in this case 64 bits. In total the array has 64 registers, in other words 4096 bits. It still has 1543 logic elements more than the high-speed version.

6 CONCLUSION

The techniques that are used in the hardware implementation of an algorithm have large effects on final results. We were able to analyze two different perspectives of project that directed their results to completely different sides. With some changes in the project noted that the same algorithm can have very different results even using the same metrics to measure their performance.

How an algorithm is designed also influence techniques that we can use when we implement it. Thanks to the symmetry and simplicity of its round function, Keccak allows trading off area for speed and vice versa. Different architectures reflect different trade-offs [5].

7 REFERENCES

[1] Moreno, Edward, D. Fábio, B. Rodolfo. Criptografia em Software e Hardware. 1st ed. São Paulo – Brazil. Bless. 2003.

[2] NIST(2012, November). cryptographic hash Algorithm Competition. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

[3] NIST.(2012, October). Selects Winner of Secure Hash Algorithm (SHA-3) Competition. Available: http://www.nist.gov/itl/csd/sha-100212.cfm

[4] Bertoni, Guido; Daemen, Joan; Peeters, Michaël; Van Assche, Gilles. Keccak in a nutshell Available: http://keccak.noekeon.org/.

[5] Bertoni, Guido; Daemen, Joan; Peeters, Michaël; Van Assche, Gilles.(2012, May) Keccak implementation overview. , Available: http://keccak.noekeon.org/Keccak-implementation-3.2.pdf.

[6] Chang, Shu jen; Perlner, Ray; Burr, William E.; Turan, Meltem Sönmez; Kelsey, John M.; Paul, Souradyuti; Bassham, Lawrence E. . Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition. Avaliable: http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf.

[7] Mentor Graphics. ModelSim. Available: http://www.mentor.com/products/fpga/simulation/modelsim

[8] Altera. Quartus II web edition. Available: http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html.