A System-on-Chip Design of the RadioGatún Hash Function

Paris Kitsos and Bhanu Prasad

International Conference on High Performance Computing, Networking and Communication Systems,

9-12 of July 2007 in Orlando, FL, USA

A System-on-Chip Design of the RadioGatún Hash Function

Paris Kitsos
Computer Science
School of Science and Technology
Hellenic Open University, Patras, Greece
E-mail: pkitsos@eap.gr

Bhanu Prasad
Department of Computer and Information Sciences, Florida A&M University,
Tallahassee, FL, USA
E-mail: bhanu.prasad@famu.edu

Abstract

Recent VLSI hash function designs allow incorporating features such as more complex input expansion schedules, more rounds and bigger states into these functions. As a result, these hash functions are slower and bulkier when compared with previous ones such as MD4, MD5, SHA-1, RIPEMD, etc. An architecture and VLSI implementation of a new hash function called RadioGatún, that achieves high-speed performance, is presented in this paper. Three different FPGA devices are used for the demonstration of this new hash function. In the first device, the hash function achieves throughput up to 1145 Mbps at 85 MHz; in the second device it achieves throughput 1819 Mbps at 135 MHz; in the third device, it reaches a throughput 4406 Mbps at 327 MHz for one data block for hashing. Since no other previous RadioGatún implementations exist in the literature, the comparison with previous implementations of hash families such as MD5, SHA-1, SHA-2, Whirlpool, etc. are provided.

1. Introduction

Hash functions are symmetric primitives that are used in many cryptographic protocols and functions. In general, a hash function maps a bitstring of arbitrary length to a fixed-length output that is called as the hash value or message digest.

The most commonly known hash function is the Secure Hash Algorithm-1 (SHA-1) [1]. However, the security level of SHA-1 does not match the security level guaranteed by the newly announced AES encryption standard [2] that is specified using 128-bit, 192-bit and 256-bit keys. The National Institute of Standards and Technology (NIST) has announced the updated Federal Information Processing Standard (FIPS 180-2) [3], which has introduced three new hash functions referred to as SHA-2 (256-bit, 384-bit and 512-bit).

More recent designs attempt to inspire more confidence by including more complex input expansion schedules, more rounds and a bigger state, resulting in hash functions that are slower and bulkier. As a result, the New European Schemes for Signatures, Integrity and Encryption (NESSIE) project [4] was responsible to introduce a hash function with similar security level. As a result, the hash function included in the NESSIE portfolio is Whirlpool [5].

NIST is also initiating an effort to develop one or more additional hash algorithms through a public competition [6]. Until now, the RadioGatún hash function [7] has been proposed for hardware oriented platforms.

In this paper, a system-on-chip design of RadioGatún hash function is proposed. Since no other RadioGatún implementation is available, comparisons with the implementations of other hash families [8-13] are provided.

The rest of the paper is organized as follows: Section 2 briefly described the RadioGatún hash function and in Section 3, the proposed architecture is presented. The hardware synthesis results and comparison analysis with other works are reported in Section 4 and finally Section 5 concludes the paper.

2. RadioGatún Hash Function Specifications

The RadioGatún function is an alternating-input Iterative Mangling Function (IMF) with the belt-and-mill structure. A general diagram of the IMF structure is provided in Figure 1.

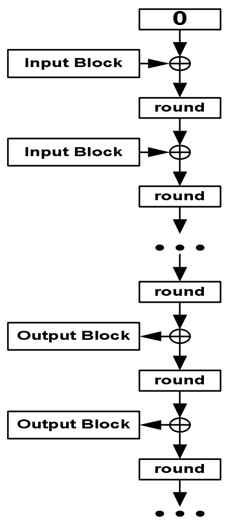

Figure 1: The structure of IMF.

It consists of the alternation of input injection and a simple invertible round function, followed by a fixed number of rounds without input or output, followed by the iterated application of rounds while returning part of the state. Also, there is an input mapping that maps the bits of an input block to bits of the state and the output mapping that maps bits of the state to the bits of an output block. Both the input mapping and output mapping operations are linear.

2.1. The belt-and-mill structure

The round function is the central component of any alternating-input IMF. The state consists of two parts: the belt and the mill, and the round function that treats the belt and the mill very differently. The state consists of four operations that can take place in parallel. *Mill function* is an invertible nonlinear function applied to the mill. *Belt function* is an invertible simple linear function applied to the belt. In *Mill feedforward*, some bits of the mill are fed to the belt in a linear way. In *Bell feedforward*, some bits of the belt are fed to the mill in a linear way. More details are presented.

The mill consists of 19 words a[i], the belt b of 13 stages b[i] of 3 words b[i, j] each. An input block p consists of 3 words p[i], an output block z consists of 2 words z[i]. The indexing starts from 0. The round function is specified by the following algorithm and a general diagram is provided in Figure 2.

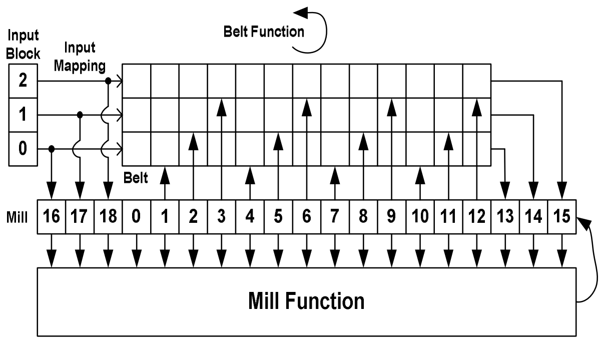

Figure 2: Diagram of the round function R.

The round function R:

```
(A, B) = R(a, b)
for all i do
B[i] = b[i + 1 mod 13]
End of {Belt function: simple rotation}
for i = 0 to 11 do
B[i + 1, i mod 3] = B[i + 1, i mod 3] XOR a[i + 1]
end of {Mill to belt feedforward}
A = Mill(a) {Mill function}
for i = 0 to 2 do
A[i + 13] = A[i + 13] XOR b[12, i]
end of {Belt to mill feedforward}
```

The round function R makes use of the mill function that is specified in the following algorithm:

The mill function Mill:

```
A = Mill(a)
all indices should be taken modulo 19,
x >>> y denotes rotation of bits within x over y positions
for all i do
$A[i] = a[i] \oplus \overline{\overline{a[i+1]}a[i+2]}$
end of {γ: nonlinearity}
for all i do
a[i] = A[7i]>>>i(i + 1)/2
end of {π: intra-word and inter-word dispersion}
for all i do
A[i] = a[i] ⊕ a[i + 1] ⊕ a[i + 4]
end of {θ: diffusion}
A[0] = A[0] ⊕ 1 {ι: asymmetry}
```

The input mapping (Input Block) is specified in the following algorithm:

The input mapping F_i:

```
(a, b) ß 0
for i = 0 to 2 do
b[0, i] = p[i]
a[i + 16] = p[i]
end of
Return (a, b)
```

The output mapping is specified in the following algorithm:

The output mapping F_o:

```
z[0] = a[1]
z[1] = a[2]
Return z
```

2.2. The RadioGatún message digest

Let x denotes the input that can be a bitstring of any length and l_w be the parameter word length that can have any value from 1 to 64. Then the message digest (*hash value z*) of the RadioGatún is:

$$z = \text{RadioGatún}[l_w](x).$$

Each value of l_w defines another function. The word length is set to 64 by default: RadioGatún means RadioGatún[64].

It is claimed that RadioGatún[lw] offers a security level indicated by a capacity lc = 19lw. For the 64-bit version RadioGatún this is a capacity of 1216-bit, for the 32-bit version and 16-bit version this gives 608-bit and 304-bit respectively.

3. Proposed Architecture

The objective of the research described in this paper is to ascertain how fast the RadioGatún hash function can operate on synchronous hardware devices. As such, the architecture which has been developed is implemented using the XILINX Virtex-E, Virtex-II and Virtex-4 FPGAs. The architecture that performs the RadioGatún hash function is shown in Figure 3. The word length (l_w) of this architecture is specified as equal to 64.

In this architecture, one round is used in an iterative scheme in order to operate the alternating-input IMF function. Also, some multiplexers are necessary in order to combine the input block data with the previous block.

As this figure shows, the main parts of the proposed architecture are the Round, the Padder, the Input Function, one 2x1 1216-bit multiplexer with the corresponding register, one 2x1 2496-bit multiplexer with the corresponding register and a 128-bit output register. The Mill and Belt parameters are set to 0. Finally, the Control Unit synchronizes the overall hash function operation.

The *Padder* pads the input data (message) by appending a single bit equal to 1 and zeroes until the length of the result is a multiple of the round input block length (192-bit). If we have a message with length ≤ 191-bit then an input block with 192-bit is constructed. If we have a message with length 192-bit ≤ length ≤ 383-bit then two input blocks with 384-bit total are constructed. If we have a message with length 384-bit ≤ length ≤ 575-bit then three input blocks with 576-bit total are constructed and so on.

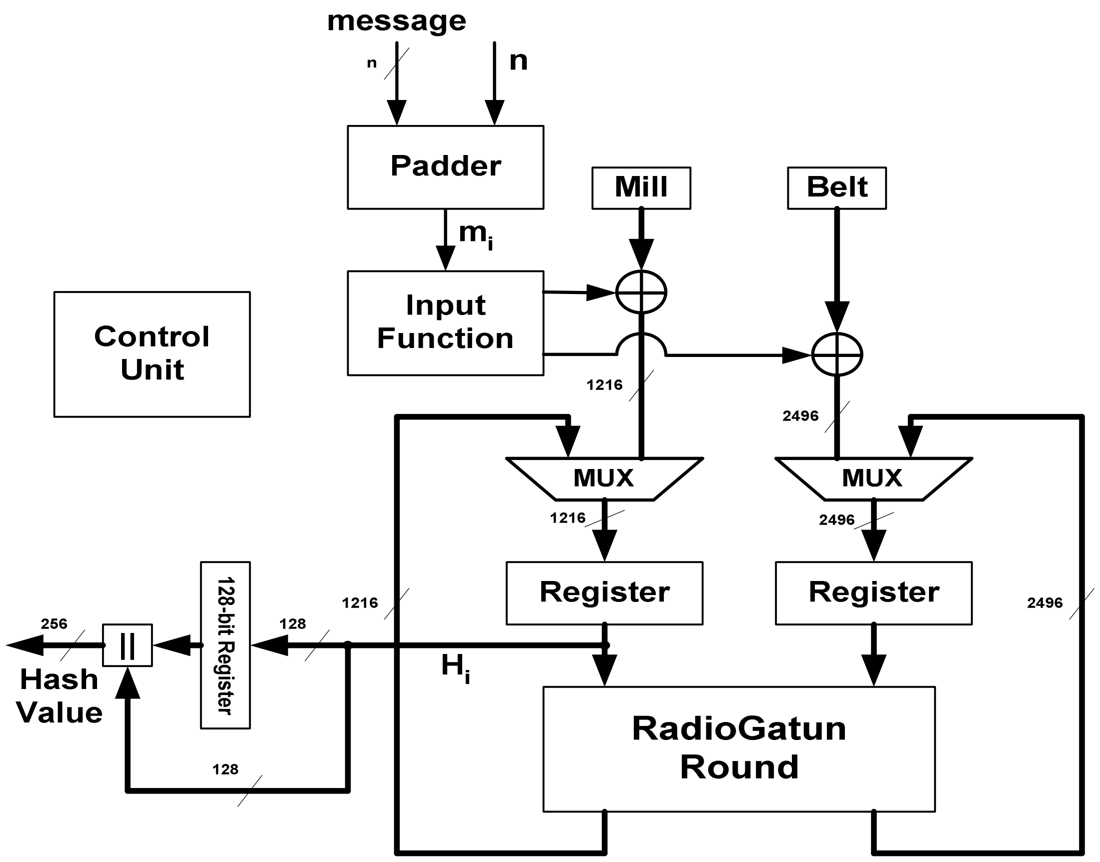

Figure 3: Hardware architecture of the RadioGatún hash function.

In the proposed architecture, an interface with a 64-bit input for Message is considered. The input *n* specifies the total length of the initial message. The padded message is partitioned into a sequence of *t* 192-bit blocks *m1, m2, …, mt*.

This sequence is then used in order to generate a sequence of two 128-bit strings H1 and H2, in the following way (the blocks H1 and H2 are used in order to produce the final Hash Value). Initially each *mi* block is processed through Input Function. For each *mi* block, one round is used. Then 16 blank rounds are followed without any input or output. After that, two 128-bit strings H1 and H2 are produced through another two rounds. The following round (after the blank rounds) generates the H1 block. This block is temporarily stored in the 128-bit output register. The H2 block is generated during the next round. Then both H1 and H2 blocks are concatenated (|| module) in order to produce the final Hash Value.

As a result, if we have a message with length $\leq$ 191-bit then an input block RadioGatún operates for 19 rounds. If we have a message with length 192-bit $\leq$ length $\leq$ 383-bit then RadioGatún operates for 20 rounds. If we have a message with length 384-bit $\leq$ length $\leq$575-bit then RadioGatún operates for 21 rounds and so on.

The basic module of the RadioGatún hash function (Figure 3) is the RadioGatún round function. The architecture of the round function (and described by the *round function, R.* algorithm) is depicted in Figure 4. This function consists of four components that are connected as shown in Figure 4. The *Mill to belt feedforward* and *Belt to mill feedforward* transformations are simple XOR operations, as described by the corresponding algorithm. In addition, the belt function is a simple rotation from left to right by one byte and implemented with wired technique. Finally, the mill function is an array of three transformations each with its own specific contribution. The aim of these transformations is to introduce high diffusion and distributed nonlinearity to system.

Due to the fact that one round is executed in a single clock cycle, the whole system has a very high throughput rate.

The VLSI implementation of the transformation mill function is shown in Figure 5. γ is an invertible nonlinear transformation. It is composed of 19 basic components. Inside every component, one 64-bit NOT operation in combination of one 64-bit OR operation is executed.

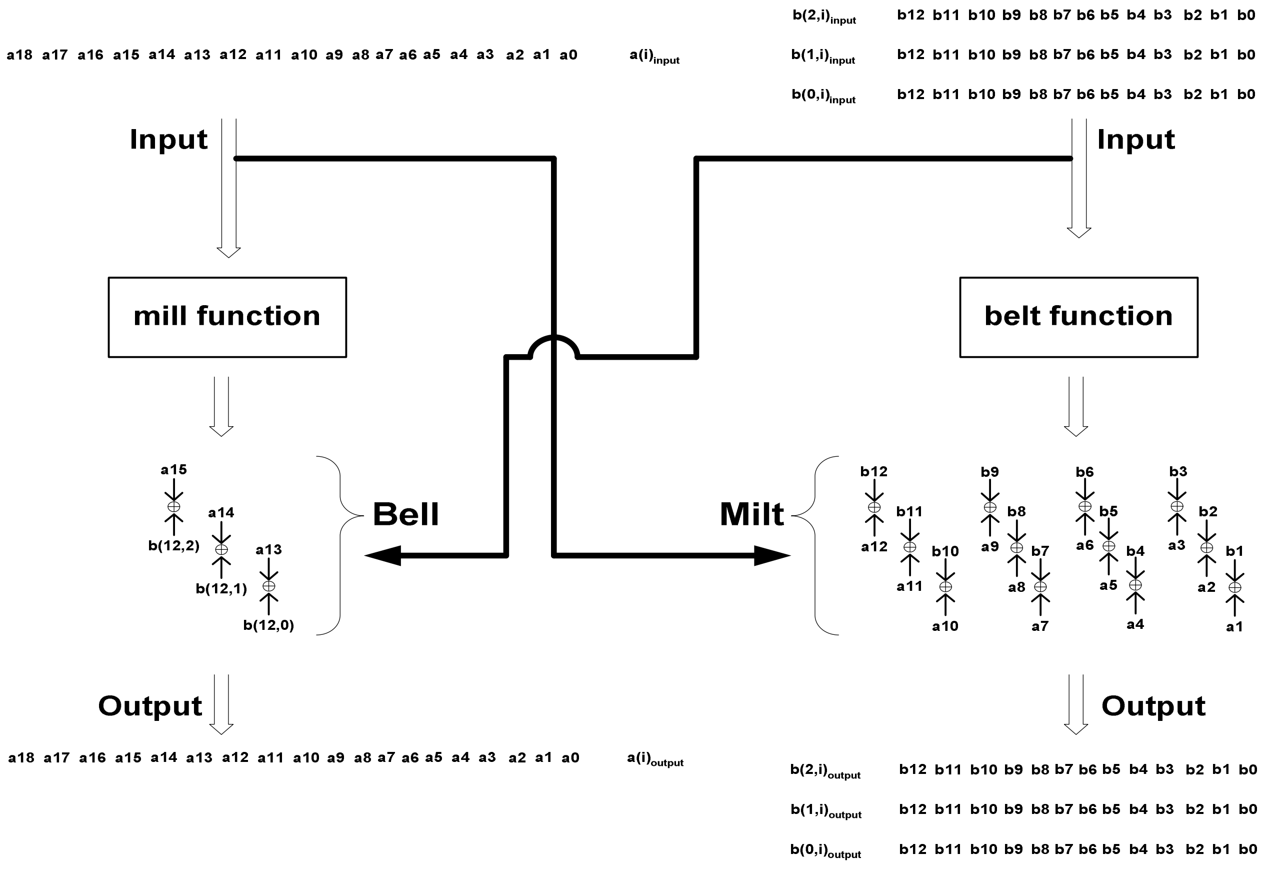
b(2,i)input b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
a18 a17 a16 a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0
a(i)input
b(1,i)input b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
b(0,i)input b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
Input
Input
mill function
belt function
a15 b(12,2) a14 b(12,1) a13 b(12,0)
Bell
Milt
b12 a12 b11 a11 b10 a10 b9 a9 b8 a8 b7 a7 b6 a6 b5 a5 b4 a4 b3 a3 b2 a2 b1 a1
Output
Output
a18 a17 a16 a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0
a(i)output
b(2,i)output b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
b(1,i)output b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
b(0,i)output b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0

Figure 4: Architecture of the round function.

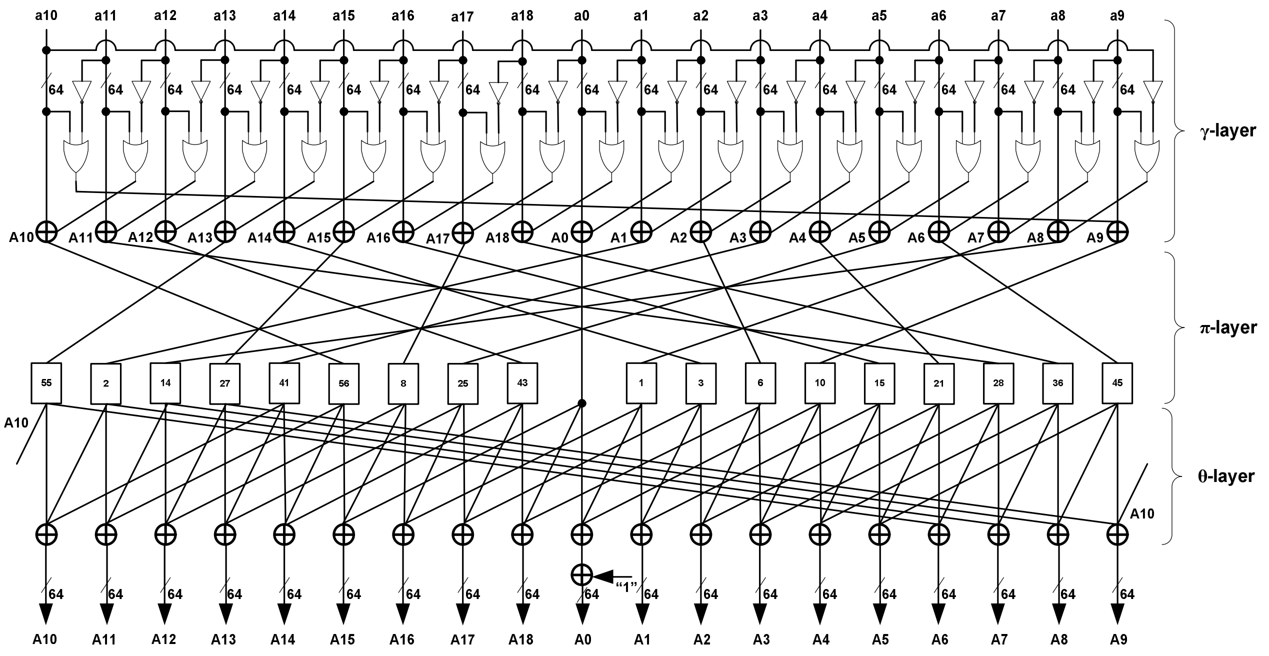

Figure 5: Hardware architecture of the mill function.

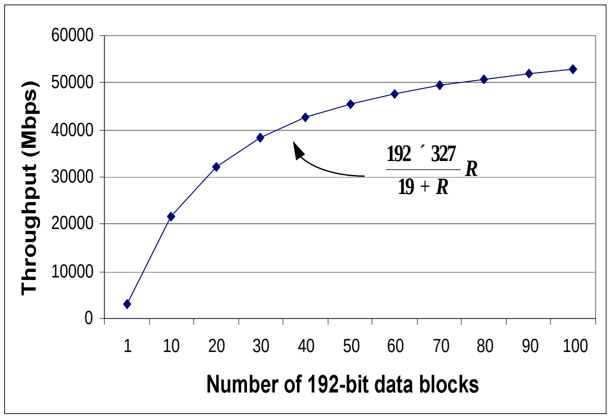

Figure 6: Throughput vs. number of data packets.

The outputs of each OR are combined with previous inputs through XOR operation, as shown in Figure 5. The permutation π combines cyclic word shifts and a permutation of the word positions. The operation of π has been implemented with wires which map each 64-bit input in to the appropriate output, while the internal bits of every stream are shifted cyclic. θ is an invertible linear transformation. It is also composed of 19 main components. Each one of these executes three 64-bit XORs, as shown in Figure 5. Finally, the transformation ι executes a 64-bit XOR operation between the first 64-bit block with the "1" in order to increase the asymmetry of the round function.

4. Hardware Synthesis Results

The proposed architecture has been captured by using VHDL in structural description. All the internal components of the design were synthesized, placed and routed by using XILINX Virtex-E, Virtex-II and Virtex-4 FPGA devices. How many rounds are executed each time depends on the initial message length. The hash operation demands 19 rounds to eject the first hash value. As a result, the throughput depends on the initial message length. In this case, throughput is computed by the function $\frac{192*F}{19+R}R$, where R is the number of 192-bit blocks which are injected into the system. It is easy to prove that the throughput value increases when the value of the parameter R increases.

For example, in XILINX 4VLX25FF668 device, the operation frequency is equal to 327 MHz. Figure 6 shows the system's throughput in relation to the number of data packets applied for hashing.

If only one block (R=1) is injected for data hashing, the throughput is 3.1 Gbps. For R=100 data blocks, the throughput can reach the value of 52.7 Gbps. Similar measurements for the rest of FPGA devices can be computed.

Table 1: Synthesis results and comparisons.

Implementation	FPGA Device	CLB Slices	Frequency (MHz)	Throughput (Mbps)
MD5 [8] (Iterative arch.)	Xilinx V1000FG680-6	880	21	165
SHA-1 [9]	Xilinx V300PQ240-6	2606	37	237
SHA-2 (256) [10]	Xilinx V300E	1004	42.9	77
RIPEMD-160 [10]	Xilinx V300E	1004	42.9	89
SHA-2 (256/384/512) [11]	Xilinx V200PQ240-6	1060/1966/2237	74/72/75	326/350/480
Whirlpool BB * [12]	Xilinx V1000EFG1156-8	5713	72	3686
Whirlpool LB * [12]	Xilinx V1000EFG1156-8	5585	87.5	4480
Whirlpool [13]	Xilinx XC4VLX100	4956	144	4790
Proposed #1	Xilinx V400EBG560	3033	85	(R=1, 816 / R=100, 13714)
Proposed #2	Xilinx 2V1000FF896	3099	135	(R=1, 1296 / R=100, 21781)
Proposed #3	Xilinx 4VLX25FF668	3031	327	(R=1, 3139 / R=100, 52760)

***** BB (Boolean expressions Based) represents the S-boxes implemented using Boolean expressions and LB (LUT Based) represents the S-boxes implemented using LUTs.

The synthesis results are shown in Table 1. Since the RadioGatún is a new hash function, no other implementations are available. As a result, comparisons with the implementations of other hash families [8-12] are provided in order to have a fair and detailed comparison with the proposed implementations.

RadioGatún function's time performance depends on the initial message. So, throughput measurements for one data block (R=1 and message with length $\leq$ 191-bit needs 19 clock cycles) and hundred data blocks (R=100 and message with length > 19200-bit needs 119 clock cycles) are computed.

The proposed implementation was realized by three different FPGA devices. At the Virtex-E (V400EBG560), FPGA achieves a throughput up to 816 Mbps at 85 MHz for one data block. Also, at the Virtex-II (2V1000FF896), FPGA achieves a throughput up to 1296 Mbps at 135 MHz for one data block and finally at Virtex-IV (4VLX25FF668), FPGA reaches a throughput equals to 3139 Mbps at 327 MHz.

As the Table 1 shows, the RadioGatún operation frequency is competitive and in many cases it is much better than the previous implementations. Also, it achieves better time performance than previous hash functions (like MD5, SHA-1, SHA-2, RIPEMD, etc.) for small message lengths and much better time performance for large message lengths.

5. Conclusion

An efficient architecture for the new hash function, RadioGatún, and its VLSI implementation in different FPGA devices are presented in this paper. In the first device, it achieves a throughput value equals to 816 Mbps at 85 MHz, in the second device it achieves a throughput 1296 Mbps at 135 MHz and finally in the third, FPGA achieves a throughput 3139 Mbps at 327 MHz for one data block for hashing. The synthesis results and comparisons prove that RadioGatún implementation is a flexible solution in the applications with ultra high-speed specification demands.

6. References

[1] SHA-1 Standard, National Institute of Standards and Technology (NIST), Secure Hash Standard, FIPS PUB 180-1, available online at: www.itl.nist.gov/fipspubs/fip180-1.htm. Accessed on February 27 2007.

[2] "Advanced encryption standard", available online at http://csrc.nist.gov. Accessed on February 27 2007.

[3] SHA-2 Standard, National Institute of Standards and Technology (NIST), Secure Hash Standard, FIPS PUB 180-2, available online at: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf. Accessed on February 27 2007.

[4] "NESSIE. New European scheme for signatures, integrity, and encryption", http://www.cosic.esat.kuleuven.ac.be/nessie. Accessed on February 27 2007.

[5] P. S. L. M. Barreto and V. Rijmen, "The Whirlpool hashing function". Primitive submitted to NESSIE, September 2000, revised on May 2003, http://planeta.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html. Accessed on February 27 2007.

[6] National Institute of Standards and Technology (NIST), Information Technology Laboratory, available online at: http://www.csrc.nist.gov/pki/HashWorkshop/index.html. Accessed on February 27 2007.

[7] Guido Bertoni, Joan Daemen, Gilles Van Assche, Micha¨el Peeters, "RadioGatún, a belt-and-mill hash function", Second Cryptographic Hash Workshop, Santa Barbara, CA, August 24-25, 2006.

[8] Janaka Deepakumara, Howard M. Heys and R. Venkatesam, "FPGA Implementation of MD5 hash algorithm", Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2001), Toronto, Ontario, May 2001.

[9] N. Sklavos, P. Kitsos, K. Papadomanolakis and O. Koufopavlou, “Random number generator architecture and VLSI implementation”, Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS 2002), USA, 2002.
[10] Sandra Dominikus, “A hardware implementation of MD4-Family algorithms”, Proceedings of IEEE International Conference on Electronics Circuits and Systems (ICECS 2002), Croatia, September 2002.
[11] N. Sklavos and O. Koufopavlou, “On the hardware implementation of the SHA-2 (256, 384, 512) hash functions”, Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS 2003), May 25-28, Bangkok, Thailand, 2003.
[12] P. Kitsos and O. Koufopavlou, “Whirlpool Hash Function: Architecture and VLSI Implementation”, In proc. of IEEE International Symposium on Circuits & Systems (ISCAS'04), Canada, May 23-26, 2004.
[13] M. McLoone, C. McIvor and A. Savage, “High-Speed Hardware Architectures of the Whirlpool Hash Function”, In proc. of IEEE 2005 Conference on Field-Programmable Technology (FPT’05), Singapore, December 11-14, 2005.