Reprint

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

Copyright Notice: This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted or mass reproduced without the explicit permission of the copyright holder.
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.](image-url)
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.](image)

The round function R:

\[ (A, B) = R(a, b) \]
for all \( i \) do
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:

\[ A = \text{Mill}(a) \]
all indices should be taken modulo 19,
\[ x >>> y \] denotes rotation of bits within \( x \) over \( y \) positions
for all \( i \) do
\[ A[i] = a[i] \oplus a[i+1 \oplus 2] \]
end of \{\gamma: nonlinearity\}
for all \( i \) do
\[ a[i] = A[7i \oplus i(i+1)/2 \oplus 1] \]
end of \{\pi: intra-word and inter-word dispersion\}
for all \( i \) do
\[ A[i] = a[i] \oplus a[i+1] \oplus a[i+4] \]
end of \{\theta: diffusion\}
\[ A[0] = A[0] \oplus 1 \]
\{\iota: asymmetry\}

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

The input mapping \( F^i \):

\[ (a, b) \rightsquigarrow 0 \]
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 \( F^o \):

\[ z[0] = a[1] \]
Return \( z \)

2.2. The RadioGatúñ 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 RadioGatúñ is:

\[ z = \text{RadioGatúñ} [l_w](x). \]
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 \( \leq 191 \)-bit then an input block with 192-bit is constructed. If we have a message with length 192-bit \( \leq \) length \( \leq 383 \)-bit then two input blocks with 384-bit total are constructed. If we have a message with length 384-bit \( \leq \) length \( \leq 575 \)-bit then three input blocks with 576-bit total are constructed and so on.

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 \( m_1, m_2, \ldots, m_t \).
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 ≤ 191-bit then an input block RadioGatún operates for 19 rounds. If we have a message with length 192-bit ≤ length ≤ 383-bit then RadioGatún operates for 20 rounds. If we have a message with length 384-bit ≤ length ≤ 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 transformation. It is composed of 19 basic components. Inside every component, one 64-bit NOT operation in...
The outputs of each OR are combined with previous inputs through XOR operation, as shown in Figure 5. The permutation $\pi$ combines cyclic word shifts and a permutation of the word positions. The operation of $\pi$ has been implemented with wires which map each 64-bit input to the appropriate output, while the internal bits of every stream are shifted cyclic. $\theta$ 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 $\iota$ 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 $R \times 192 \times 327 \div 19 + R$, 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.

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

* 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 ≤ 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