Journal of Information Assurance and Security 5 (2010) 626-636

Compact Implementation of BLUE MIDNIGHT WISH-256 Hash Function on Xilinx FPGA Platform

Mohamed El Hadedy[1,3], **Danilo Gligoroski**[2], **Svein J. Knapskog** [1] **and Martin Margala** [3]

[1]The Norwegian Center of Excellence for Quantifiable Quality of Service in Communication Systems(Q2S),
Norwegian University of Science and Technology (NTNU),
O.S.Bragstads plass 2E, N-7491 Trondheim, Norway
mohamed.elhadedy@q2s.ntnu.no, Knapskog@q2s.ntnu.no

[2]Department of Telematics, Faculty of Information Technology, Mathematics and Electrical Engineering,
The Norwegian University of Science and Technology (NTNU),
O.S.Bragstads plass 2E, N-7491 Trondheim, Norway
danilog@item.ntnu.no

[3] Department of Electrical and Computer Engineering, University of Massachusetts Lowell,
Ball 301, One University Ave, Lowell, MA 01854,USA
Mohamed_Aly@uml.edu, Martin_Margala@uml.edu

Abstract: **– In cryptography and information security, hash functions are considered as the "Swiss army knife" - they are used in countless protocols and algorithms. In 2005, we witnessed a significant theoretical breakthrough in breaking the current cryptographic standard SHA-1. Although there is another family of standardized hash functions called SHA-2, ready to replace SHA-1 hash function, at the end of 2007, the National Institute of Standards and Technology (NIST) decided to start a 4 year world-wide development process, including a competition for the superior algorithm design, for choosing the next cryptographic hash standard SHA-3. Blue Midnight Wish is one of the proposed new designs in the SHA-3 competition that continues in the Second Round of the competition. This paper presents the design and analysis of an area efficient Blue Midnight Wish compression function with digest size of 256 bits (BMW-256) on FPGA platforms. The proposed architecture achieves significant improvements in system throughput with reduced area. We demonstrate the performance of the proposed BMW hash function core using VIRTEX 5 FPGA implementation. The new BMW hash function design allows for 16X speed up in performance while consuming significantly lower area than previously reported.**

Keywords: Hash Function Standard, SHA-2, Blue Midnight Wish

I. Introduction

Cryptographic hash functions play a fundamental role in modern cryptography. While related to conventional hash functions commonly used in non- cryptographic computer applications in both cases, larger domains are mapped to smaller ranges they differ in several important aspects. Hash functions take a message as input and produce an output referred to as a hash code, hash result, hash value, or simply hash. More precisely, a hash function h maps bit strings of arbitrary finite length to strings of fixed length, say n bits. For a domain D and range R with h: D $\rightarrow$ R and $|D| > |R|$,the function is many to one, implying that the existence of collisions (pairs of inputs with identical output) is unavoidable. Indeed, restricting h to a domain of t-bit inputs ($t > n$), if h were random in the sense that all outputs were essentially equiprobable , then about 2^{t-n} inputs would map to each output, and two randomly chosen inputs would yield the same output with probability 2^{-n}. the basic idea of cryptographic hash functions is that a hash-value serves as a compact representative image (sometimes called an imprint, digital fingerprint, or message digest) of an input string, and can be used as if it were uniquely identifiable with that string [1]. As the hash functions are widely used in many security applications, it is very important that they fulfill certain security properties. Those can be considered as follows:

- Pre-image resistance: for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any pre-image M such that H = Hash (M) when given any H for which a corresponding input is not known.
- Second pre-image resistance: It must be hard to find another pre-image for a given input, i.e., given M_0 and Hash (M_0) getting M_1 must be hard such that Hash (M_0) = Hash (M_1).
- Collision resistance: It is computational infeasible to find any two distinct inputs http M_0, M_1 which hash to the same output, i.e., such that Hash (M_0) = Hash (M_1).

A function that is characterized by the first two properties is called a one-way hash function. If all three properties are met the hash function is considered collision resistant. Finding collisions in a specific hash function is the most common

Received June 14, 2010

way of attacking it. There are a lot of hash functions; they are based on cellular automation, block ciphers, modular arithmetic, knapsack and lattice problem, algebraic matrices, etc. Recently, there have been two SHA algorithms introduced. SHA-1, and SHA-2, and although they have some similarities, they have also significant differences [1,2]. SHA-1 is the most used member of the SHA hash family, employed in hundreds of different applications and protocols. However, in 2005, we witnessed a significant theoretical breakthrough in breaking the current cryptographic standard SHA-1 [1]. Although there is another family of standardized hash function called SHA-2, ready to replace SHA-1, consequently, the discovered mathematical weakness which might exist indicates the need for using stronger hash functions [2].
The SHA-2 family is a family of four algorithms that differs from each other by different digest size, different initial values and different word size. The digest sizes are: 224, 256, 384 and 512 bits. Although no attacks have yet been reported on the SHA-2 variants, they are algorithmically similar to SHA-1, and the National Institute of Standards and Technology NIST have felt the need for and made efforts to develop an improved new family of hash functions [2, 3]. At the end of 2007, (NIST) decided to start a 4 year world-wide development process, including a competition for the superior algorithm design, for choosing the next cryptographic hash standard SHA-3. The new hash standard SHA-3 is currently under development - the function will be selected via an open competition running between 2008 and 2012. The Blue Midnight Wish (BMW) hash function is one of the candidates from Second Round of the competition and it is one of the fastest proposed new designs in the SHA-3 competition in software [4]. In this paper, we show the proposed architecture design is simple, area efficient and provides significant throughput improvements over previous works. The proposed BMW hash function core -256 is evaluated in FPGA using VIRTEX II XCV300-6PQ240 Xilinx device [5, 6]. The rest of the paper is organized as follows. In Section 2, we describe briefly the compression function of BMW-256 algorithm with new tweaks. Section3, Background of Xilinx Virtex-5 FPGA. Section4, highlights of the architecture and FPGA implementation of the proposed BLUE MIDNIGHT WISH hash function core sub-system. In Section 5, the synthesis results of the FPGA implementation are given with comparisons with other related works. Finally, in section 6, conclusions, observations and future work are discussed.

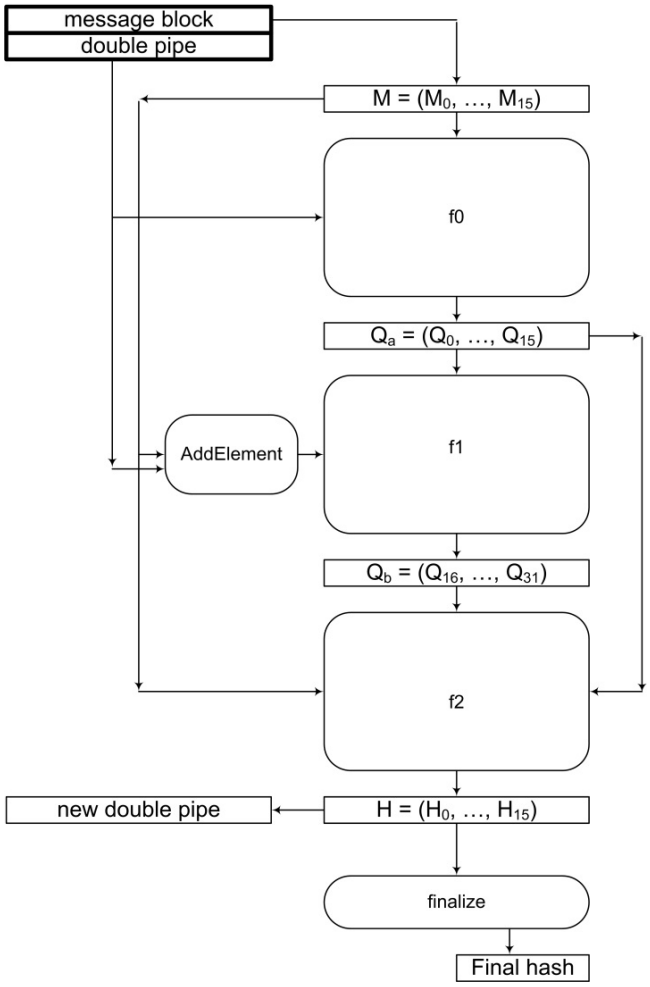

Figure. 1: Representation of the compression function in Blue Midnight Wish

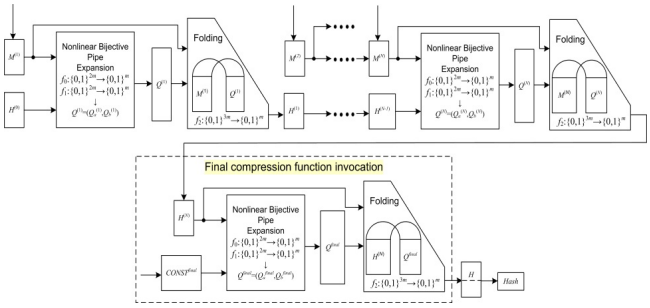

Figure. 2: A graphic representation of the Blue Midnight Wish Hash function

II. BLUE MIDNIGHT WISH (BMW)

Blue Midnight wish hash function is a one of the 14 candidates in second round of the NIST's SHA-3 competition [7]. It was tweaked in order to resist attacks by Thomsen [8]. BMW is a family of hash functions, containing four major instances, e.g., BMW-n, for n= 224, 256, 384, 512, where n is the size of hash output. BMW is a wide-pipe Merkle-Damgrd hash construction [9] with an unconventional compression function, where the nonlinearity is derived from the overlap of modular addition and XOR operations. The most innovative parts of the design are the compression function construction and the design of the permutations; much of the design is novel and unique amongst the second-round candidates. BMW has very good performance and appears to be suitable for a wide range of platforms. It has modest memory requirements. The BMW has four different operations in the hash computation stage: bit-wise logical word XOR operation, Word addition and subtraction, shift operations (left or right), and rotate left operation. BMW uses a double pipe design to increase the resistance against generic multi-collision attacks and length extension attacks [10,11]. In the double pipe design, the sizes of the inputs to the compression functions are twice the message digest size.

As shown in Fig.1, the compression function of BMW takes the chaining of 16 words $H_0^{(i-1)}$, $H_1^{(i-1)}$,....,$H_{15}^{(i-1)}$, and

Table 1: Definition of the function f_0 of BLUE MIDNIGHT WISH

1. Bijective Transform of $M^{(i)} \oplus H^{(i-1)}$
$W_0^{(i-1)} = (M_5^{(i)} \oplus H_5^{(i-1)}) - (M_7^{(i)} \oplus H_7^{(i-1)}) + (M_{10}^{(i)} \oplus H_{10}^{(i-1)}) + (M_{13}^{(i)} \oplus H_{13}^{(i-1)}) + (M_{14}^{(i)} \oplus H_{14}^{(i-1)})$
$W_1^{(i-1)} = (M_6^{(i)} \oplus H_6^{(i-1)}) - (M_8^{(i)} \oplus H_8^{(i-1)}) + (M_{11}^{(i)} \oplus H_{11}^{(i-1)}) + (M_{14}^{(i)} \oplus H_{14}^{(i-1)}) - (M_{15}^{(i)} \oplus H_{15}^{(i-1)})$
$W_2^{(i-1)} = (M_0^{(i)} \oplus H_0^{(i-1)}) + (M_7^{(i)} \oplus H_7^{(i-1)}) + (M_9^{(i)} \oplus H_9^{(i-1)}) - (M_{12}^{(i)} \oplus H_{12}^{(i-1)}) + (M_{15}^{(i)} \oplus H_{15}^{(i-1)})$
$W_3^{(i-1)} = (M_0^{(i)} \oplus H_0^{(i-1)}) - (M_1^{(i)} \oplus H_1^{(i-1)}) + (M_8^{(i)} \oplus H_8^{(i-1)}) - (M_{10}^{(i)} \oplus H_{10}^{(i-1)}) + (M_{13}^{(i)} \oplus H_{13}^{(i-1)})$
$W_4^{(i-1)} = (M_1^{(i)} \oplus H_1^{(i-1)}) + (M_2^{(i)} \oplus H_2^{(i-1)}) + (M_9^{(i)} \oplus H_9^{(i-1)}) - (M_{11}^{(i)} \oplus H_{11}^{(i-1)}) - (M_{14}^{(i)} \oplus H_{14}^{(i-1)})$
$W_5^{(i-1)} = (M_3^{(i)} \oplus H_3^{(i-1)}) - (M_2^{(i)} \oplus H_2^{(i-1)}) + (M_{10}^{(i)} \oplus H_{10}^{(i-1)}) - (M_{12}^{(i)} \oplus H_{12}^{(i-1)}) + (M_{15}^{(i)} \oplus H_{15}^{(i-1)})$
$W_6^{(i-1)} = (M_4^{(i)} \oplus H_4^{(i-1)}) - (M_0^{(i)} \oplus H_0^{(i-1)}) - (M_3^{(i)} \oplus H_3^{(i-1)}) - (M_{11}^{(i)} \oplus H_{11}^{(i-1)}) + (M_{13}^{(i)} \oplus H_{13}^{(i-1)})$
$W_7^{(i-1)} = (M_1^{(i)} \oplus H_1^{(i-1)}) - (M_4^{(i)} \oplus H_4^{(i-1)}) - (M_5^{(i)} \oplus H_5^{(i-1)}) - (M_{12}^{(i)} \oplus H_{12}^{(i-1)}) - (M_{14}^{(i)} \oplus H_{14}^{(i-1)})$
$W_8^{(i-1)} = (M_2^{(i)} \oplus H_2^{(i-1)}) - (M_5^{(i)} \oplus H_5^{(i-1)}) - (M_6^{(i)} \oplus H_6^{(i-1)}) + (M_{13}^{(i)} \oplus H_{13}^{(i-1)}) - (M_{15}^{(i)} \oplus H_{15}^{(i-1)})$
$W_9^{(i-1)} = (M_0^{(i)} \oplus H_0^{(i-1)}) - (M_3^{(i)} \oplus H_3^{(i-1)}) + (M_6^{(i)} \oplus H_6^{(i-1)}) - (M_7^{(i)} \oplus H_7^{(i-1)}) + (M_{14}^{(i)} \oplus H_{14}^{(i-1)})$
$W_{10}^{(i-1)} = (M_8^{(i)} \oplus H_8^{(i-1)}) - (M_1^{(i)} \oplus H_1^{(i-1)}) - (M_4^{(i)} \oplus H_4^{(i-1)}) - (M_7^{(i)} \oplus H_7^{(i-1)}) + (M_{15}^{(i)} \oplus H_{15}^{(i-1)})$
$W_{11}^{(i-1)} = (M_8^{(i)} \oplus H_8^{(i-1)}) - (M_0^{(i)} \oplus H_0^{(i-1)}) - (M_2^{(i)} \oplus H_2^{(i-1)}) - (M_5^{(i)} \oplus H_5^{(i-1)}) + (M_9^{(i)} \oplus H_9^{(i-1)})$
$W_{12}^{(i-1)} = (M_1^{(i)} \oplus H_1^{(i-1)}) + (M_3^{(i)} \oplus H_3^{(i-1)}) - (M_6^{(i)} \oplus H_6^{(i-1)}) - (M_9^{(i)} \oplus H_9^{(i-1)}) + (M_{10}^{(i)} \oplus H_{10}^{(i-1)})$
$W_{13}^{(i-1)} = (M_2^{(i)} \oplus H_2^{(i-1)}) + (M_4^{(i)} \oplus H_4^{(i-1)}) + (M_7^{(i)} \oplus H_7^{(i-1)}) + (M_{10}^{(i)} \oplus H_{10}^{(i-1)}) + (M_{11}^{(i)} \oplus H_{11}^{(i-1)})$
$W_{14}^{(i-1)} = (M_3^{(i)} \oplus H_3^{(i-1)}) - (M_5^{(i)} \oplus H_5^{(i-1)}) + (M_8^{(i)} \oplus H_8^{(i-1)}) - (M_{11}^{(i)} \oplus H_{11}^{(i-1)}) - (M_{12}^{(i)} \oplus H_{12}^{(i-1)})$
$W_{15}^{(i-1)} = (M_{12}^{(i)} \oplus H_0^{(i-1)}) - (M_4^{(i)} \oplus H_4^{(i-1)}) - (M_6^{(i)} \oplus H_6^{(i-1)}) - (M_9^{(i)} \oplus H_9^{(i-1)}) + (M_{13}^{(i)} \oplus H_{13}^{(i-1)})$
2. Further bijective transform of $W_j^{(i)}, j = 1, 2, 3, ..., 15$
$Q_0^{(i)} = S_0(W_0^{(i)}) + H_1^{(i-1)}; Q_1^{(i)} = S_1(W_1^{(i)}) + H_2^{(i-1)}; Q_2^{(i)} = S_2(W_2^{(i)}) + H_3^{(i-1)}; Q_3^{(i)} = S_3(W_3^{(i)}) + H_4^{(i-1)};$
$Q_4^{(i)} = S_4(W_4^{(i)}) + H_5^{(i-1)}; Q_5^{(i)} = S_0(W_1^{(i)}) + H_6^{(i-1)}; Q_6^{(i)} = S_1(W_6^{(i)}) + H_7^{(i-1)}; Q_7^{(i)} = S_2(W_7^{(i)}) + H_8^{(i-1)};$
$Q_8^{(i)} = S_3(W_8^{(i)}) + H_9^{(i-1)}; Q_9^{(i)} = S_4(W_9^{(i)}) + H_{10}^{(i-1)}; Q_{10}^{(i)} = S_0(W_{10}^{(i)}) + H_{11}^{(i-1)}; Q_{11}^{(i)} = S_1(W_{11}^{(i)}) + H_{12}^{(i-1)};$
$Q_{12}^{(i)} = S_2(W_{12}^{(i)}) + H_{13}^{(i-1)}; Q_{13}^{(i)} = S_3(W_{13}^{(i)}) + H_{14}^{(i-1)}; Q_{14}^{(i)} = S_4(W_{14}^{(i)}) + H_{15}^{(i-1)}; Q_{15}^{(i)} = S_3(W_{15}^{(i)}) + H_0^{(i-1)};$
3. S-transform used in f_0 Function
$S_0(x) = SHR^1(x) \oplus SHL^3(x) \oplus ROTL^4(x) \oplus ROTL^{19}(x)$
$S_1(x) = SHR^1(x) \oplus SHL^2(x) \oplus ROTL^8(x) \oplus ROTL^{23}(x)$
$S_2(x) = SHR^2(x) \oplus SHL^1(x) \oplus ROTL^{12}(x) \oplus ROTL^{25}(x)$
$S_3(x) = SHR^2(x) \oplus SHL^2(x) \oplus ROTL^{15}(x) \oplus ROTL^{29}(x)$

Table 2: Initial double pipe $H^{(i-1)}$ for BMW-256 (Hexadecimal values)

BLUE MIDNIGHT WISH-256			
$H_0^{(i-1)}$	0x40414243	$H_8^{(i-1)}$	0x60616263
$H_1^{(i-1)}$	0x44454647	$H_9^{(i-1)}$	0x64656667
$H_2^{(i-1)}$	0x48494A4B	$H_{10}^{(i-1)}$	0x68696A6B
$H_3^{(i-1)}$	0x4C4D4E4F	$H_{11}^{(i-1)}$	0x6C6D6E6F
$H_4^{(i-1)}$	0x50515253	$H_{12}^{(i-1)}$	0x70717273
$H_5^{(i-1)}$	0x54555657	$H_{13}^{(i-1)}$	0x74757677
$H_6^{(i-1)}$	0x58595A5B	$H_{14}^{(i-1)}$	0x88898A8B
$H_7^{(i-1)}$	0x5C5D5E5F	$H_{15}^{(i-1)}$	0x8C8D8E8F

a message 16 words $M_0^{(i)}$, $M_1^{(i)}$,....,$M_{15}^{(i)}$ as input,and produces the updated chaining $H_0^{(i)}$, $H_1^{(i)}$,....,$H_{15}^{(i)}$. The size of a word is 32 bits for BMW-224/256 and 64 bits for BMW-384/512. The new BMW compression function [5,6] uses 2 main parts as shown in Fig.2. The first one comprises three functions, called f_0, f_1, and f_2, in sequence to generate $H_{(i)}$. As shown in the following. Inputs for the function f_0 are two arguments as shown in Fig.1: The first argument consists of sixteen 32-bit words, which are working as initial double pipe values $H_0^{(i-1)}$, $H_1^{(i-1)}$,....,$H_{15}^{(i-1)}$, as shown in Table 1, for BMW-256. The second argument consists of sixteen 32-bit words, which represent the input message block: $M_0^{(i)}$, $M_1^{(i)}$,....,$M_{15}^{(i)}$. As shown in Table 2, the function $f_0(M^{(i)}, H^{(i-1)})$ computes $M^{(i)} \oplus H^{(i-1)}$, produces a temporary $W_j(i)$,$j = 1, 2, 3, ..., 15$, and $Q_a^{(i)}$ = ($Q_0^{(i)}$, $Q_1^{(i)}$,..., $Q_{15}^{(i)}$) from $W_j^{(i)}$. The second function f_1 takes the same message block $M_{(i)}$, and $Q_a^{(i)}$ (the output from f_0) as input, and generates the second part of the quadruple pipe $Q_b^{(i)}$ = ($Q_0^{(i)}$, $Q_1^{(i)}$,..., $Q_{15}^{(i)}$), through ADD_Element, EXPAND_1 and EXPAND_2 expression functions as shown in Eq.(1) and Table.3. Expansion functions make use of the s-transform along with the r-transforms as shown in Eq. (2), and Eq. (3).

For ii = 0,1 : $Q_{(ii+16)}^{(i)} = Expand_1(ii + 16)$

$$Expand_1(i) = S_1(Q_{(j-16)}^i) + S_2(Q_{(j-15)}^i) + S_3(Q_{(j-14)}^i) + S_0(Q_{(j-13)}^i) + S_1(Q_{(j-12)}^i) + S_2(Q_{(j-11)}^i) + S_3(Q_{(j-10)}^i) + S_0(Q_{(j-9)}^i) + S_1(Q_{(j-8)}^i) + S_2(Q_{(j-7)}^i) + S_3(Q_{(j-6)}^i) + S_0(Q_{(j-5)}^i) + S_1(Q_{(j-4)}^i) + S_2(Q_{(j-3)}^i) + S_3(Q_{(j-2)}^i) + S_0(Q_{(j-1)}^i) + ADD\_Element(j - 16) \quad (1)$$

For ii=2,3,4,5,...,15 : $Q_{(ii+16)}^{(i)} = Expand_2(ii + 16)$

$$Expand_2(i) = (Q_{(j-16)}^i) + r_1(Q_{(j-15)}^i) + (Q_{(j-14)}^i) + r_2(Q_{(j-13)}^i) + (Q_{(j-12)}^i) + r_3(Q_{(j-11)}^i) + (Q_{(j-10)}^i) + r_4(Q_{(j-9)}^i) + (Q_{(j-8)}^i) + r_5(Q_{(j-7)}^i) + (Q_{(j-6)}^i) + r_6(Q_{(j-5)}^i) + (Q_{(j-4)}^i) + r_7(Q_{(j-3)}^i) + S_4(Q_{(j-2)}^i) + S_5(Q_{(j-1)}^i) + ADD\_Element(j - 16)$$

Note that ADD_Element(j) index expressions involving the variable j for left rotations, M and H are computed modulo(16).

$$ADD\_Element(j) = (ROTL^{(j+1)}(M_{(j)}^{(i)}) + ROTL^{(j+4)}(M_{(j+3)}^{(i)}) + ROTL^{(j+11)}(M_{(j+10)}^{(i)}) + K_{j+16}) \oplus H_{j+7}^{(i)} \quad (2)$$

$$\begin{aligned} &r_1(x) = ROTL^3(X), r_2(x) = ROTL_7(x), \\ &r_3(x) = ROTL^{13}(x), r_4(x) = ROTL^{16}(x), \\ &r_5(x) = ROTL^{19}(x), r_6(x) = ROTL^{23}(x), \\ &r_7(x) = ROTL^{27}(x), S_4(x) = SHR^1(x) \oplus x, \\ &S_5(x) = SHR^2(x) \oplus x \end{aligned} \quad (3)$$

Table 3: K_j for BLUE MIDNIGHT WISH (Hexadecimal Values)

BLUE MIDNIGHT WISH-256			
K_0	0x55555550	K_8	0x 7ffffff8
K_1	0x5aaaaaa5	K_9	0x 8555554d
K_2	0x5fffffa	K_{10}	0x 8aaaaaa2
K_3	0x6555554f	K_{11}	0x 8ffffff7
K_4	0x6aaaaaa4	K_{12}	0x 9555554c
K_5	0x6ffffff9	K_{13}	0x 9aaaaaa1
K_6	0x7555554e	K_{14}	0x 9ffffff6
K_7	0x7aaaaaa3	K_{15}	0x a555554b

As shown in Table 4, the third function f_2 takes three arguments: Message block M(i) and the quadruple pipes $Q_a(i)$ and $Q_b(i)$ and generates the value of the double pipe H^i. The second part contains the same functions but instead of initial double pipe values $H_0^{(i-1)}$, $H_1^{(i-1)}$, $H_2^{(i-1)}$,..., $H_{15}^{(i-1)}$, it will use constant values $CONST_j^{final}$ = ($CONST_0^{final}$, $CONST_1^{final}$, $CONST_2^{final}$,..., $CONST_{15}^{final}$)as shown in Table 5 and the input message block will be the new double pipe $H^{(i)}$ = ($H_0^{(i)}$, $H_1^{(i)}$, $H_2^{(i)}$,..., $H_{15}^{(i)}$). The reason to use $CONST_j^{final}$ values is to remove one degree of freedom to the attackers who try to find pseudo collisions and pseudo-preimages. Additionally, the final invocation of the compression function is a measure for any attack whereby an attacker can find near collisions or near-pseudo-collisions of the compression function of BMW [5,6].

Table 5: $CONST_j^{final}$ for BMW-256 (Hexadecimal values)

BLUE MIDNIGHT WISH-256			
$CONST_0^{final}$	0Xaaaaaaa0	$CONST_8^{final}$	0Xaaaaaaa8
$CONST_1^{final}$	0Xaaaaaaa1	$CONST_9^{final}$	0Xaaaaaaa9
$CONST_2^{final}$	0Xaaaaaaa2	$CONST_{10}^{final}$	0Xaaaaaaaa
$CONST_3^{final}$	0Xaaaaaaa3	$CONST_{11}^{final}$	0Xaaaaaaab
$CONST_4^{final}$	0Xaaaaaaa4	$CONST_{12}^{final}$	0Xaaaaaaac
$CONST_5^{final}$	0Xaaaaaaa5	$CONST_{13}^{final}$	0Xaaaaaaad
$CONST_6^{final}$	0Xaaaaaaa6	$CONST_{14}^{final}$	0Xaaaaaaae
$CONST_7^{final}$	0Xaaaaaaa7	$CONST_{15}^{final}$	0Xaaaaaaaf

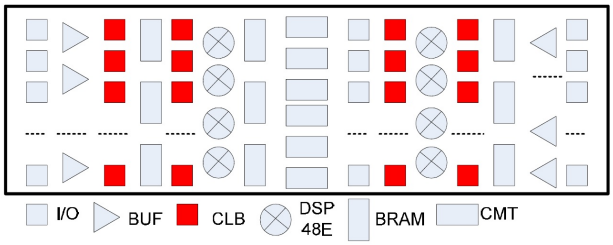

Figure. 3: Xilinx Virtex 5 FPGA components [12,13]

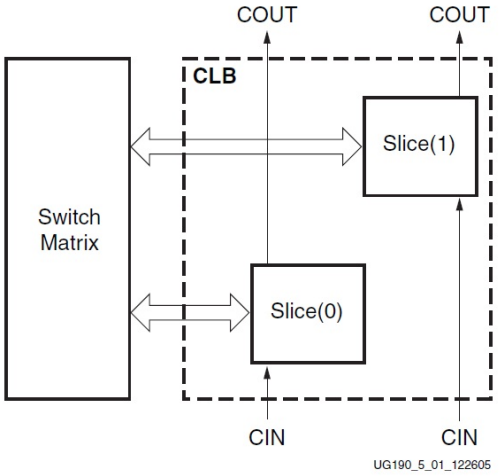

Figure. 4: Arrangement of slices in a CLB of Virtex-5 FPGA [13]

III. Background of Xilinx Virtex-5 FPGA Architecture

Virtex-5 FPGAs offer today's designers the ultimate system integration platform to solve their most demanding requirements. Virtex-5 FPGAs offer unprecedented performance and density gains - at speeds on average 30 percent faster and 65 percent increased capacity over previous generation 90nm FPGAs. Notably, this breakthrough performance has been achieved while reducing dynamic power consumption by 35 percent and consuming 45 percent less area than previous generation devices. Description of Virtex-5 FPGA architecture parts can help us to understand how to implement BMW-256 compression function. As shown in Fig.3 the major components of Xilinx Virtex-5 FPGA. The Configurable Logic Blocks (CLBs) are the main logic resources for implementing sequential as well as combinatorial circuits. Each CLB element is connected to a switch matrix for access to the general routing matrix (shown in Fig. 4). A CLB element contains a pair of slices. These two slices do not have direct connections to each other, and each slice is organized as a column. Each slice in a column has an independent carry chain. For each CLB, slices in the bottom of the CLB are labeled as SLICE (0), and slices in the top of the CLB are labeled as SLICE (1).

The Xilinx tools designate slices with the following definitions. An "X" followed by a number identifies the position of each slice in a pair as well as the column position of the slice. The "X" number counts slices starting from the bot-

Table 4: Definition of the folding function f_2 of BLUE MIDNIGHT WISH

1. Cumulative temporary variables XL and XH
$XL = Q_{16}^{(i)} \oplus Q_{17}^{(i)} \oplus Q_{18}^{(i)} \oplus Q_{19}^{(i)} \oplus Q_{20}^{(i)} \oplus Q_{21}^{(i)} \oplus Q_{22}^{(i)} \oplus Q_{23}^{(i)}$
$XH = XL \oplus Q_{24}^{(i)} \oplus Q_{25}^{(i)} \oplus Q_{26}^{(i)} \oplus Q_{27}^{(i)} \oplus Q_{28}^{(i)} \oplus Q_{29}^{(i)} \oplus Q_{30}^{(i)} \oplus Q_{31}^{(i)}$
2. The new double pipe $H^{(i)}$
$H_0^{(i)} = (SHL^5(XH) \oplus SHR^5(Q_{16}^{(i)}) \oplus M_0^{(i)}) + (XL \oplus Q_{24}^{(i)} \oplus Q_0^{(i)})$
$H_1^{(i)} = (SHL^7(XH) \oplus SHL^8(Q_{17}^{(i)}) \oplus M_1^{(i)}) + (XL \oplus Q_{25}^{(i)} \oplus Q_1^{(i)})$
$H_2^{(i)} = (SHL^5(XH) \oplus SHL^5(Q_{18}^{(i)}) \oplus M_2^{(i)}) + (XL \oplus Q_{26}^{(i)} \oplus Q_2^{(i)})$
$H_3^{(i)} = (SHR^1(XH) \oplus SHL^5(Q_{19}^{(i)}) \oplus M_3^{(i)}) + (XL \oplus Q_{27}^{(i)} \oplus Q_3^{(i)})$
$H_4^{(i)} = (SHR^3(XH) \oplus (Q_{20}^{(i)}) \oplus M_4^{(i)}) + (XL \oplus Q_{28}^{(i)} \oplus Q_4^{(i)})$
$H_5^{(i)} = (SHL^6(XH) \oplus SHR^5(Q_{21}^{(i)}) \oplus M_5^{(i)}) + (XL \oplus Q_{29}^{(i)} \oplus Q_5^{(i)})$
$H_6^{(i)} = (SHR^4(XH) \oplus SHL^6(Q_{22}^{(i)}) \oplus M_6^{(i)}) + (XL \oplus Q_{30}^{(i)} \oplus Q_6^{(i)})$
$H_7^{(i)} = (SHr^1 1(XH) \oplus SHL^2(Q_{23}^{(i)}) \oplus M_7^{(i)}) + (XL \oplus Q_{31}^{(i)} \oplus Q_7^{(i)})$
$H_8^{(i)} = ROTL^9(H_4^{(i)} + (XH \oplus Q_{24}^{(i)} \oplus M_8^{(i)}) + (SHL^8(XL) \oplus Q_{23}^{(i)} \oplus Q_8^{(i)})$
$H_9^{(i)} = ROTL^{10}(H_5^{(i)} + (XH \oplus Q_{25}^{(i)} \oplus M_9^{(i)}) + (SHR^6(XL) \oplus Q_{16}^{(i)} \oplus Q_9^{(i)})$
$H_{10}^{(i)} = ROTL^{11}(H_6^{(i)} + (XH \oplus Q_{26}^{(i)} \oplus M_{10}^{(i)}) + (SHL^6(XL) \oplus Q_{17}^{(i)} \oplus Q_{10}^{(i)})$
$H_{11}^{(i)} = ROTL^{12}(H_7^{(i)} + (XH \oplus Q_{27}^{(i)} \oplus M_{11}^{(i)}) + (SHL^4(XL) \oplus Q_{18}^{(i)} \oplus Q_{11}^{(i)})$
$H_{12}^{(i)} = ROTL^{13}(H_0^{(i)} + (XH \oplus Q_{28}^{(i)} \oplus M_{12}^{(i)}) + (SHR^3(XL) \oplus Q_{19}^{(i)} \oplus Q_{12}^{(i)})$
$H_{13}^{(i)} = ROTL^{14}(H_1^{(i)} + (XH \oplus Q_{29}^{(i)} \oplus M_{13}^{(i)}) + (SHR^4(XL) \oplus Q_{20}^{(i)} \oplus Q_{13}^{(i)})$
$H_{14}^{(i)} = ROTL^{15}(H_2^{(i)} + (XH \oplus Q_{30}^{(i)} \oplus M_{14}^{(i)}) + (SHR^7(XL) \oplus Q_{21}^{(i)} \oplus Q_{14}^{(i)})$
$H_{15}^{(i)} = ROTL^{16}(H_3^{(i)} + (XH \oplus Q_{31}^{(i)} \oplus M_{15}^{(i)}) + (SHR^2(XL) \oplus Q_{22}^{(i)} \oplus Q_{15}^{(i)})$

tom in sequence 0, 1 (the first CLB column); 2, 3 (the second CLB column); etc. A "Y" followed by a number identifies a row of slices. The number remains the same within a CLB, but counts up in sequence from one CLB row to the next CLB row, starting from the bottom. Fig.5 shows four CLBs located in the bottom-left corner of the die [13]. For the Xilinx Virtex 5, every slice contains four look-up tables (LUTs), four storage elements, wide-function multiplexers, and carry logic. These elements are used by all slices to provide logic, arithmetic, and ROM functions.

In addition to this, some slices support two additional functions: storing data using distributed RAM and shifting data with 32-bit registers. Slices that support these additional functions are called SLICEM; others are called SLICEL as shown in Fig.6, and Fig.7 respectively. The storage elements in a slice can be configured as either edge-triggered D-type flip-flops or level-sensitive latches. The D input can be driven directly by a LUT output via AFFMUX, BFFMUX, CFFMUX or DFFMUX, or by the BYPASS slice inputs bypassing the function generators via AX, BX, CX, or DX input. When configured as a latch, the latch is transparent when the CLK is Low.

In literature, LUT is a basic unit for reconfigurable logic. The LUT can implement any digital logic truth table, constrained only by the number of signal inputs and outputs. In addition to, basic LUTs, slices contain three multiplexers (F_7AMUX, F_7BMUX4, and F_8MUX). These multiplexers are used to combine up to four function generators to provide any function of seven or eight inputs in a slice.

F_7AMUX and F_7BMUX are used to generate seven input functions from LUTs A and B, or C and D, while F8MUX is used to combine all LUTs to generate eight input functions. Functions with more than eight inputs can be implemented using multiple slices. There are no direct connections between slices to form function generators greater than eight inputs within a CLB or between slices. Each CLB contains 2 slices, 8 LUTs, 8 flip flops, 2 arithmetic and carry chains, 256 bit distributed RAM and 128 bits shift registers.

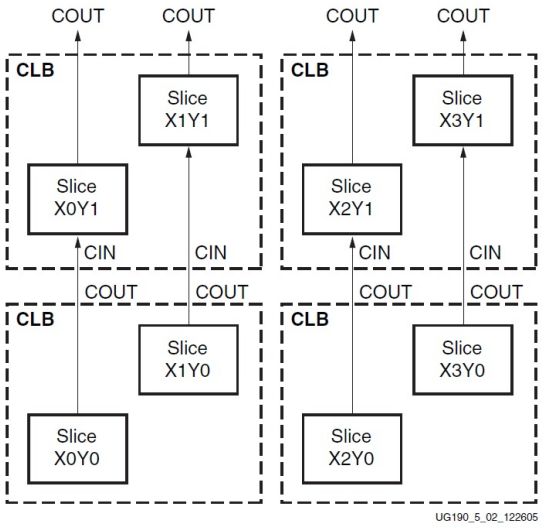

Figure. 5: The relations between slices and CLBs [13]

IV. BMW-256 core Architecture

The BMW-256 core hardware architecture can be best described in three main sections. In the first section, we give an overview of the hardware block as whole. In the second section, we describe by the details the internal components of BMW-256 core. In the third section, we describe the BMW-256 hashing core operations.

A. Hardware Overview

As shown in Fig.8, BMW-256 Core block, it comprises four blocks as the following:

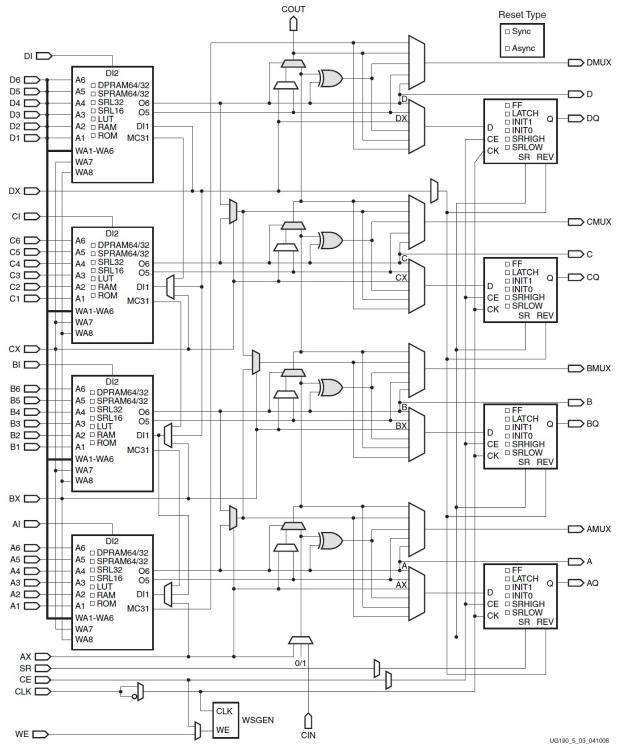

Figure. 6: SLICEM Internal Architecture [13]

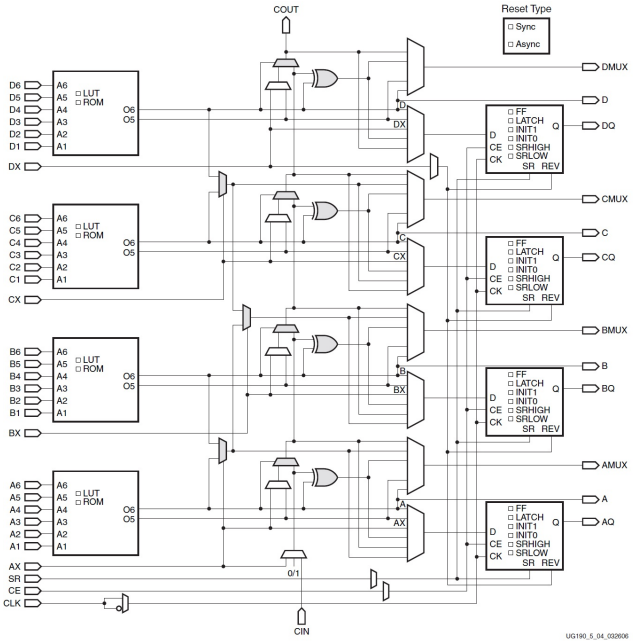

Figure. 7: SLICEL Internal Architecture [13]

- Memory block: it contains block RAM (832 byte), block ROM (192 byte) and Memory Data Bus block to organize data flows between ROM, RAM blocks and Hash Computation Core.

- Hash Computation Core: it comprises 32 bit ALU (Arithmetic Logic Unit) and 32 bit parallel Shift/Rotate Block. Hash Computation Core receives data flow from Memory block and transmits data to Memory block and Output Shift Register according to Controller block.

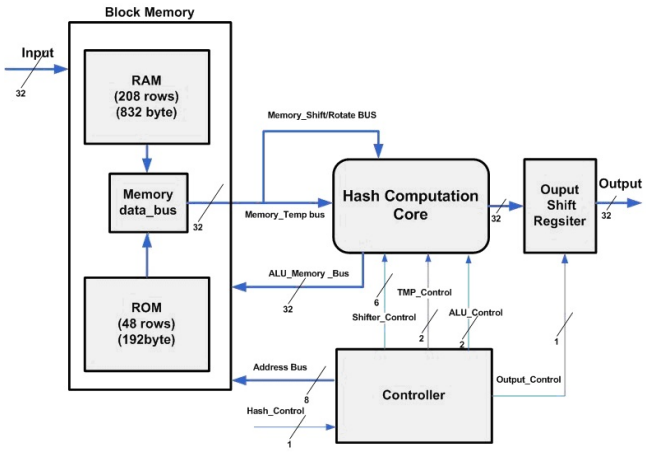

Figure. 8: BLUE MIDNIGHT WISH-256 Core Architecture

- Output Shift Register: it receive the final double pipe hash values $H_0^{(i)}$, $H_1^{(i)}$, $H_2^{(i)}$, ..., $H_{15}^{(i)}$ from Hash Computation Core according to Controller process.

- Controller : it contains Moore Finite State Machine (MFSM) instructions to control the data flow between BMW-256 Core components according to the BMW-256 mathematical algorithm [5,6]

B. BMW-256 Core internal components

In this section, we describe by details BMW-256 Core components as the following:

- Memory Block: To implement the BMW-256 Core Memory block, we used FPGA block RAM 256 x 32 bits as shown in Fig.9, the Memory block interface in Table 6, and the Memory Block Truth Table as shown in Table 7.

Table 6: BLUE MIDNIGHT WISH256 Core Memory Block Interface

signal	I/O	Description
CLK	IN	Global CLOCK
RAM_ROM	IN	Active Low , initialize RAM to write and read data , Active High, initialize ROM to read data
WR	IN	Active high, to write data in RAM rows
ADD(7:0)	IN	Data present in Address Bus
DATA_IN (31:0)	IN	Data presents in Data_RAM input 32 bit Bus
DATA_OUT(31:0)	OUT	Data presents in Data_RAM output 32 bit Bus

As we mentioned in section 3.1, Memory Block contains ROM to store the BMW-256 constants K_j, J=0,1,..., 15 , $H^{(i-1)}$ and $Constant_j^{final}$ as shown in Table.8, Table.9 and Table.10. on the other hand, Memory Block contains RAM to store the BMW-256 input block Message ($M_0^{(i)}$, $M_1^{(i)}$, ..., $M_{15}^{(i)}$) as shown in, the intermediate values of BMW hash function as shown in Table.9, and the final double pipe values $H^{(i)}$ = ($H_0^{(i)}$, $H_1^{(i)}$, $H_2^{(i)}$,...,$H_{15}^{(i)}$).

- Hash Computation Core: as shown in Fig.8, Hash Computation Core contains three operative components as the following:

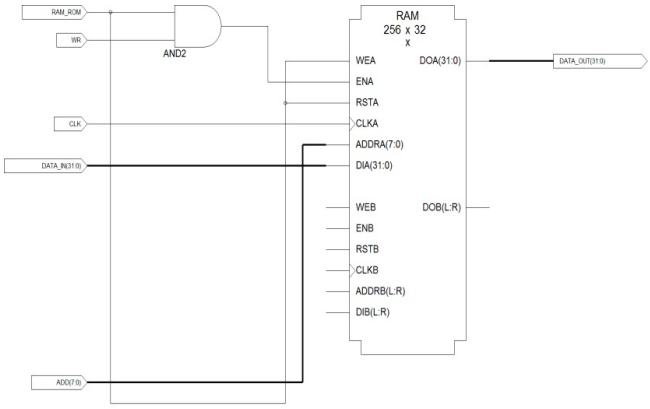

Figure. 9: BLUE MIDNIGHT WISH-256 Core Memory Block

Table 7: BLUE MIDNIGHT WISH-256 Core Memory Block Operation

WR	RAM_ROM	Operation
X (don't care)	1	ROM (Read)
0	0	RAM (Read)
1	0	RAM (Write)

- Parallel Shifter/Rotator: As shown in Fig.10 Parallel Shifter/Rotator contains a 5 x 32 Mux (Multiplex) matrix each one is Mux 2x1with big Encoder (5 X 11). This component is responsible for the shift and rotation operations of the 32 bit word. It receives 32 bit parallel data from the Memory Block and transmits 32 bit parallel data to the ALU. That happens decided by the value of shifter control word. Because we have 46 operations in BMW hash Core, the width of Shifter control word is 6 control bits as shown Table 11.

- Arithmetic and Logic Unit (ALU) component: As shown in Fig.11, Fig.12 ALU contains 32 cells all of them working together according to the truth table as shown in Table 12. ALU component uses four different operations in the hash computation stage: bit-wise logical word XOR operation, word addition and subtraction (modulo 2^{32}). ALU component receives 32 bit data word from Parallel shifter/rotator and Temporary Register and transmit the output to the Temporary Register to work as a parallel accumulator. This is decided by the value of ALU_Control word (2 bit word).

- Temporary Register contains 32 Mux2x1 and Shift register as shown in Fig.13, and Table 13. Temporary Register works as an accumulator. It receives 32 bit words from Memory unit and ALU and transmits data 32 bit word to ALU and the output stage. This happens decided by the value of TMP_Control word (2 bit word).

- Controller:Controller has been designed as Moore FSM as shown in Fig.16. it Contains, six operative parts, all of them working together to produce Memory Address word to control Memory block traffic with other BMW-256 sub-systems and on the other hand. T he controller produces the control word to control data flow between BMW-256 Core sub-systems. The controller subsys-

Table 8: BLUE MIDNIGHT WISH-256 Core ROM ($H^{(i-1)}$ locations)

RAM 256 X 32 Row Location	Double pipe initial Values	ADD (7:0)
208	$H_0^{(i-1)}$	11010000
209	$H_1^{(i-1)}$	11010001
210	$H_2^{(i-1)}$	11010010
211	$H_3^{(i-1)}$	11010011
212	$H_4^{(i-1)}$	11010100
213	$H_5^{(i-1)}$	11010101
214	$H_6^{(i-1)}$	11010110
215	$H_7^{(i-1)}$	11010111
216	$H_8^{(i-1)}$	11011000
217	$H_9^{(i-1)}$	11011001
218	$H_{10}^{(i-1)}$	11011010
219	$H_{11}^{(i-1)}$	11011011
220	$H_{12}^{(i-1)}$	11011100
221	$H_{13}^{(i-1)}$	11011101
222	$H_{14}^{(i-1)}$	11011110
223	$H_{15}^{(i-1)}$	11011111

Table 9: BMW-256 Core ROM (K_j locations)

RAM 256 X 32 Row location	Double pipe initial Values	ADD (7:0)
224	$K_0^{(i-1)}$	11100000
225	$K_1^{(i-1)}$	11100001
226	$K_2^{(i-1)}$	11100010
227	$K_3^{(i-1)}$	11100011
228	$K_4^{(i-1)}$	11100100
229	$K_5^{(i-1)}$	11100101
230	$K_6^{(i-1)}$	11100110
231	$K_7^{(i-1)}$	11100111
232	$K_8^{(i-1)}$	11101000
233	$K_9^{(i-1)}$	11101001
234	$K_{10}^{(i-1)}$	11101010
235	$K_{11}^{(i-1)}$	11101011
236	$K_{12}^{(i-1)}$	11101100
237	$K_{13}^{(i-1)}$	11101101
238	$K_{14}^{(i-1)}$	11101110
239	$K_{15}^{(i-1)}$	11101111

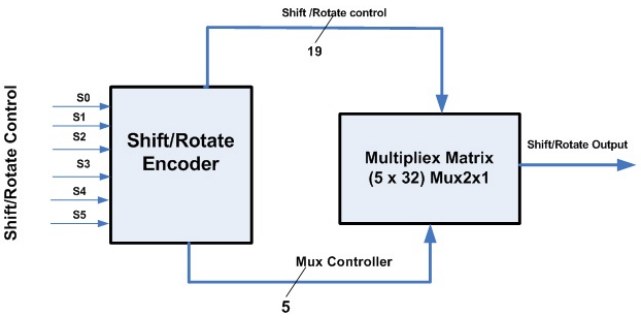

Figure. 10: Parallel Shifter/Rotator schematic diagram

Table 10: BLUE MIDNIGHT WISH-256 Core ROM ($Const_j^{final}$ locations)

RAM 256 X 32 Row location	Double pipe initial Values	ADD (7:0)
240	$Const_0^{final}$	11110000
241	$Const_1^{final}$	11110001
242	$Const_2^{final}$	11110010
243	$Const_3^{final}$	11110011
244	$Const_4^{final}$	11110100
245	$Const_5^{final}$	11110101
246	$Const_6^{final}$	11110110
247	$Const_7^{final}$	11110111
248	$Const_8^{final}$	11111000
249	$Const_9^{final}$	11111001
250	$Const_{10}^{final}$	11111010
251	$Const_{11}^{final}$	11111011
252	$Const_{12}^{final}$	11111100
253	$Const_{13}^{final}$	11111101
254	$Const_{14}^{final}$	11111110
255	$Const_{15}^{final}$	11111111

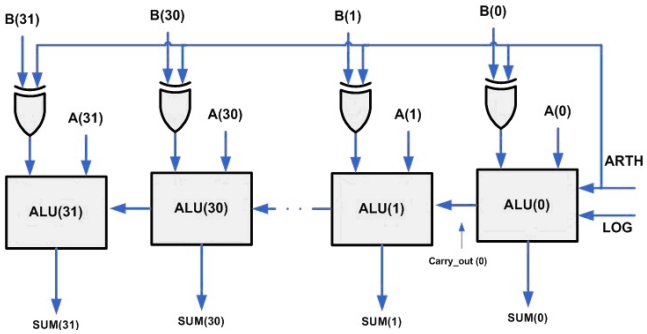

Figure. 11: Parallel 32 bit ALU (Arithmetic and Logic Unit)

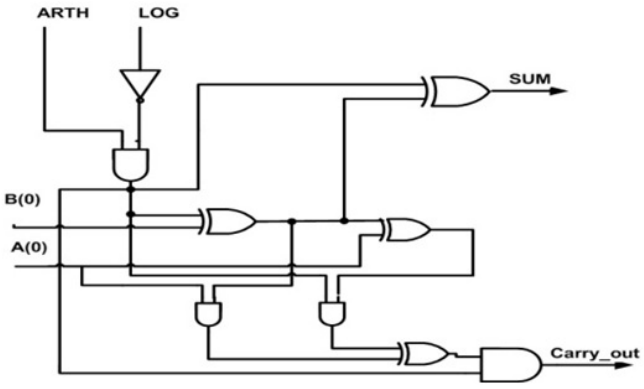

Figure. 12: ALU Cell (Arithmetic and Logic Unit)

tems are working as the following: Input Message Control, once the start signal become high, the input message Control starts to organize the sixteen input messages inside RAM locations. After that the round Enable signal become high, that will make BMW Round Control starts to execute the f0, f1, and f2 BMW according to BMW-256 Algorithm. Finally, with the final message round, with Final Round signal becomes high, the Final process Control Block starts to transfer $Const^{final}$ in Messages locations and start BMW

Table 11: Parallel Shifter/Rotator operations in BLUE MIDNIGHT WISH-256 Core

S/R control	Operation	S/R control	Operation
000000	LOAD	010111	ROL8
000001	SHl1	011000	ROL9
000010	SHl2	011001	ROL10
000011	SHl3	011010	ROL11
000100	SHl4	011011	ROL12
000101	SHl5	011100	ROL13
000110	SHl6	011101	ROL14
000111	SHl8	011110	ROL15
001000	SHR1	011111	ROL16
001001	SHR2	100000	ROL17
001010	SHR3	100001	ROL18
001011	SHR4	100010	ROL19
001100	SHR5	100011	ROL20
001101	SHR6	100100	ROL21
001110	SHR7	100101	ROL22
001111	SHR11	100110	ROL23
010000	ROL1	100111	ROL24
010001	ROL2	101000	ROL25
010010	ROL3	101001	ROL26
010011	ROL4	101010	ROL27
010100	ROL5	101011	ROL28
010101	ROL6	101100	ROL29
010110	ROL7	101101	ROL30

Table 12: Arithmetic and Logic Unit Cell (Truth Table)

LOG	ARTH	ALU Operations
1	X (Don't Care)	XOR
0	0	ADD
0	1	Subtract

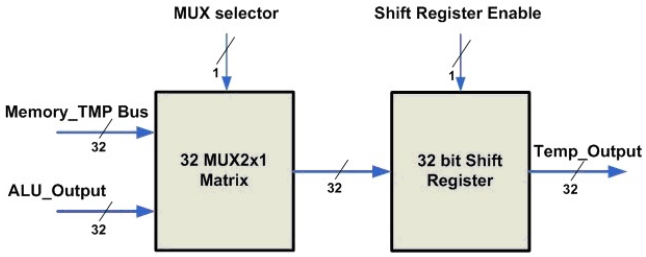

Figure. 13: Temporary Register Unit

Round Control Block to work to produce the Final Hash Output. Both Control Selector and Address Bus Selector are Multiplexers controlled by Combinational circuit block called Bus Selector Control.

C. The BMW-256 hashing core operations

In this section we describe how the Computation Hash Core works to execute the internal functions in BMW-256 as

Table 13: TMP_Register (Truth Table)

Shift_Register_Enable	MUX_Selector	Temp_Output
0	X (don't care)	Idle
1	0	Memory_TMP Bus
1	1	ALU_Output

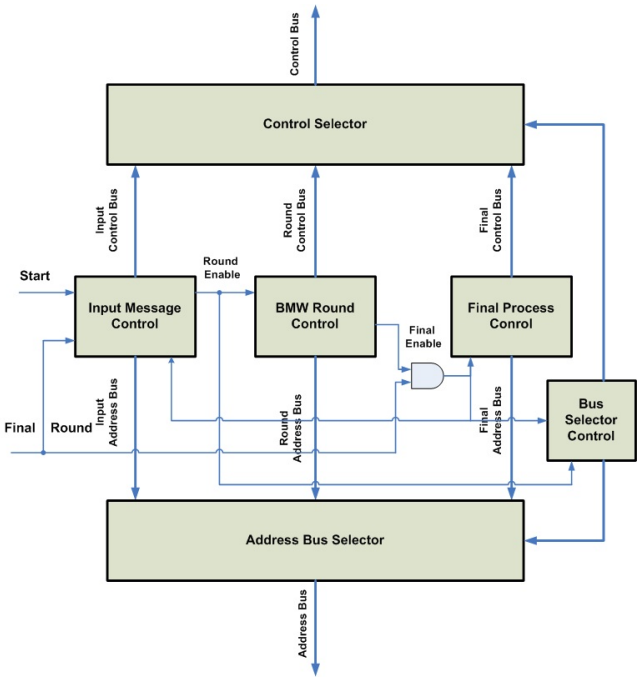

Figure. 14: Controller

shown in Table 14. For example, if we would XOR two pieces of data in locations number 4 and 5 in memory unit, and write the result in location number 7. First, Controller gives order to Memory Block to choose locations number 4. Then Controller asks Temporary Register to pick up the data from data bus and then the same operation happens with location number 5. But instead of Temporary Register, the Parallel Shifter/Rotator picks up data. Now, Controller asks operation encoder unit to give order to one bit ALU unit to add these data and save it in Temporary Register. Finally, Controller gives order to Memory Block to pick up data and located in location number 7.

Table 14: BLUE MIDNIGHT WISH internal functions (execution times)

BMW functions	No. of Cycles
F_0	413
F_1	476
F_2	171

V. Performance Evaluation

BMW 256 Core has been designed in VHDL [14] and it was synthesized (synthesis, placement and routing) using ISE foundation 10.1 [15] in VIRTEX XCV300-6PQ240 and Virtex5 XC5VLX110 Xilinx devices. As shown in Table 15, the comparison with the previous implementation for BMW-256 [16] shows improvements in number of cycles for each operation. In the proposed design we used Parrallel Shifting and rotation with parrallel 32-bit ALU instead of using single 1 bit cell , by this way , the new design has succeeded to reduce number of cycles for each operation.

Table 15: BLUE MIDNIGHT WISH-256 Hashing Core Operations (execution times)

Operation	Proposed	BLUE MIDNIGHT WISH-256[16]
Load	1	1
XOR	3	32
ADD	1	32
SUB	1	32
S_0	4	127
S_1	4	128
S_2	4	129
S_3	4	132
S_4	4	34
S_5	2	34
R_1	1	3
R_2	1	7
R_3	1	13
R_4	1	16
R_5	1	19
R_6	1	23
R_7	1	27

In Table 16, we compare the area size for different designs for SHA-2 [17,18] and with deferent candidates from SHA-3 competitions in VIRTEX and VIRTEX 5 Xilinx devices.

Table 16: BLUE MIDNIGHT WISH-256 Performance results

Algorithm Name	FPGA Type	Area (Slice)	Throughput
Proposed	Virtex XCV300	1314	6 Mbps
	Virtex5 XC5VLX110	445	21 Mbps
BMW-256 [16]	Virtex XCV300	2147	1.07 Mbps
	Virtex5 XC5VLX110	1980	5Mbps
SHA-256[17]	Virtex XCV200	4768	291Mbps
SHA-256 [18]	Virtex E XCV600	5828	———

As shown in Table 16, we have achieved around 38% lower area compared to the old design for BMW-256 with rising up throughput around 6 times compare to the old one, on the same FPGA Virtex XCV300 device and around 77% lower area compared to the old BMW-256 with rising up throughput 16 times compare to the old one on the same FPGA Virtex5 XC5VLX110.

VI. Conclusion and Future Work

In This paper we presented an FPGA implementation of a new BMW-256 Hashing core structure with 256 bits of message digest using parallel shifter/rotator and parallel 32 bit word Arithmetic logic unit (ALU). The BMW-256 core receives 16 messages words of 32 bits and processes them. The goal was to use as small area as possible in order to minimize the hardware cost. we have achieved around 72% lower area compared to SHA-256 on the same FPGA device.

References

[1] X. Wang, A. C. Yao, and F. Yao. *"Cryptanalysis on SHA-1 hash function"*. In proceeding of The Cryptographic hash workshop. National Institute of Standards and Technology, November 2005.

[2] NIST (2006). *"NIST Comments on Cryptanalytic Attacks on SHA-1"*

[3] William E. Burr, *"Cryptographic Hash Standards: Where Do We Go from Here?"*, IEEE Security and Privacy, Vol. 4, No. 2, pp. 88-91, Mar./Apr. 2006, doi:10.1109/MSP.2006.37

[4] eBACS (2010). *"ECRYPT Benchmarking of Cryptographic Systems"*

[5] D. Gligoroski, V. Klima, S. J. Knapskog, M. El-Hadedy, J.Amundsen ,*"Blue Midnight Wish"*, In proceeding of The First SHA-3 Candidate Conference, February 2009, Belgium- Leuven

[6] D. Gligoroski, V. Klima, *"A Document describing all modi_cations made on the Blue Midnight Wish cryptographic hash function before entering the Second Round of SHA-3 hash competition"*

[7] *"National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family"*. Federal Register, 27(212):6221262220, November 2007

[8] S. S. Thomsen. *"Pseudo-cryptanalysis of the Original Blue Midnight Wish"*. In S. Hong and T. Iwata, editors, Fast Software Encryption, LNCS, Seoul, South Korea, 2010. Springer. To appear.

[9] Lucks,S. *"Design Principles for Iterated Hash Functions"*, e-print (September 29, 2004)

[10] Joux, A., *"Multicollisions in iterated hash functions. Application to cascaded constructions"*.In Proceeding of CRYPTO 2004. LNCS, vol. 3152, pp. 430440, 2004.

[11] Lucks, S., *A failure-friendly design principle for hash functions*, In proceeding of ASIACRYPT,2005.

[12] M. Long, *"Implementing Skein Hash Function on Xilinx Virtex-5 FPGA Platform"*, 02-02-2009,Version 0.7

[13] Xilinx, *"Virtex-5 FPGA User Guide"*, UG190 (v5.2) November 5, 2009

[14] *"Model Sim PE/PLUS User's Manual*, Model technology, 2008

[15] Xilinx, *"Device Package User Guide"*,2010

[16] M. El Hadedy, D. Gligoroski, S. J. Knapskog, *"Low Area Implementation of the Hash Function "Blue Midnight Wish - 256" for FPGA platforms"*. In Proceedings of The International Conference on Intelligent Networking and Collaborative Systems. IEEE Computer Society 2009 ISBN 978-0-7695-3858-7.

[17] N. Sklavos, O. Koufopavlou, *"Implementation of the SHA-2 Hash Family Standard Using FPGAs"*, The Journal of Supercomputing, 31(3), pp.227-248, 2005.

[18] M. McLoone, J. V. McCanny, *"Efficient single-chip implementation of SHA-384 & SHA-512"*. In Proceedings of the International Conference on Field-Programmable Technology (FTP), pp. 311-314, 2002

Author Biographies

Mohamed El-Hadedy is a PhD student at the Centre for Quantifiable Quality of Service in Communication Systems (Q2S) at the Norwegian University of Science and Technology, Trondheim, Norway and Visiting Researcher at Electrical and computer engineering department at University of Massachusetts Lowell, Massachusetts, USA . He received his B.Sc., rated /Very Good with Honors/ (Ranked Fourth), in Electronic and Communication Engineering from the Electronic and Communications Department, Faculty of Engineering, Mansoura University, Egypt, 2002. He received his M.Sc. in Electronic and Communication Engineering from the Electronic and Communications department, Faculty of Engineering, Mansoura University, Egypt in 2006. The title of his Master Thesis is: (Improvement of digital image watermarking techniques based on FPGA Implementation). He worked as an assistant researcher in the Electronic and Communication Department, Faculty of Engineering, Mansoura University, Egypt, from 2002 to 2003, and as an assistant researcher in the Atomic Energy Authority in Cairo, Egypt, from 2004 to 2008. His research interests are cryptography, computer security, computer architecture design, FPGA and ASIC implementations, watermarking and digital rights management.

Danilo Gligoroski is professor at the Department of Telematics at Norwegian University of Science and Technology - Trondheim, Norway. He received the PhD degree in Computer Science from Institute of Informatics, Faculty of Natural Sciences and Mathematics, at University of Skopje - Macedonia in 1997. His research interests are Cryptography, Computer Security, Discrete algorithms and Information Theory and Coding.

Svein Johan Knapskog Svein J. Knapskog received his Siv.ing. degree (M.S.E.E.) from the Norwegian Institute of Technology (NTH), Trondheim, Norway in 1972. Since 2001, he has been Professor at the Norwegian University of Science and Technology (NTNU), the Department of Telematics. He is presently principal academic at the Norwegian Centre of Excellence (CoE) for Quantifiable Quality of Service in Communication Systems (Q2S). He has previously held various positions in Norwegian public sector, SINTEF and industry. From 1982 - 2000, he was Associate Professor

at NTH (later NTNU), Department of Computer Science and Telematics, where he also served a three year term as Head of Department. In the academic year 2005-2006 he has been acting Head of Department of the Department of Telematics. His field of interests includes information security and QoS as well as related communication system architectural issues. His current research focus is on information security primitives, protocols and services in distributed autonomous telecommunication systems and networks, and security evaluation there of. Prof. Knapskog has been active in a number of conference program committees and has authored/co-authored a number of technical reports, research papers and a textbook (in Norwegian)

Martin Margala (IEEE-SM04) received the M.S. degree in microelectronics from Slovak Technical University, Slovakia, in 1990 and the Ph.D. degree in electrical and computer engineering from the University of Alberta, Canada, in 1998. He is currently an Associate Professor with the Electrical and Computer Engineering Department, University of Massachusetts, Lowell. Previously, he was with the University of Rochester, Rochester, NY, and with the University of Alberta. From 1998 to 2003, he has been an adjunct scientist with the Telecommunications Research Labs, Edmonton, Canada. He holds three patents (two others pending) and is an author or coauthor of more than 120 publications in peer-reviewed journals and conference proceedings on high-frequency circuit design and test. His main research interests are energy-efficient low-voltage circuit design, high-bandwidth and data-processing architectures and adaptive built-in-self-test systems. Dr. Margala is a member of program committees of many conferences and symposia in design and test.