Single Core Implementation of Blue Midnight Wish Hash Function on VIRTEX 5 Platform

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

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

[3]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

Abstract

This paper presents the design and analysis of an area efficient implementation of the SHA-3 candidate Blue Midnight Wish hash function with different digest sizes of 256 and 512 bits on an FPGA platform. The core functionality with finalization implementation without padding stage of BMW on Xilinx Virtex-5 FPGA requires 51 slices for BMW-256 and 105 slices for BMW-512. Both BMW versions require two blocks of memory: one memory block to store the intermediate values and hash constants and the other memory block to store the instruction controls. The proposed implementation achieves a throughput of 68.71 Mpbs for BMW-256 and 112.18 Mpbs for BMW-512.

I. Introduction

To obtain efficient and secure computerized information handling, hash functions are used in countless protocols and algorithms. Until now, two generations of SHA algorithms have been standardized and widely deployed - SHA-1, and SHA-2, and although they have some similarities, they have also significant differences [1]. SHA-1 is the most frequently 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 [2]. The discovered mathematical weaknesses which were shown to exist indicated the need for replacement with a stronger hash function [3], although there exist another family of standardized hash function called SHA-2 which officially replaced SHA-1 in 2010.

The SHA-2 family is a family of four algorithms that differ 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, their operational performance is in many settings less than desirable, and the National Institute of Standards and Technology (NIST) have felt the need for an improved new family of hash functions [4]. At the end of 2007, NIST decided to invite cryptographic algorithms designers and developers to participate in an open competition running between 2008 and 2012 for choosing a new candidate for the next cryptographic hash standard SHA-3. This work is now well underway, as the competition is about to enter into its third phase, in which five of the strongest candidates will be singled out for the final testing until a winner may be declared in 2012. The Blue Midnight Wish (BMW) hash function is one of the candidates promoted to the second round of the SHA-3 competition and implemented in software, it is one of the fastest proposed new designs running in the competition [5]. In this paper, we proposed a hardware design of BMW which is simple, area efficient and provides significant throughput improvements over previous work. The proposed BMW hash function core is implemented in FPGA using Virtex 5 XC5VLX110 device.

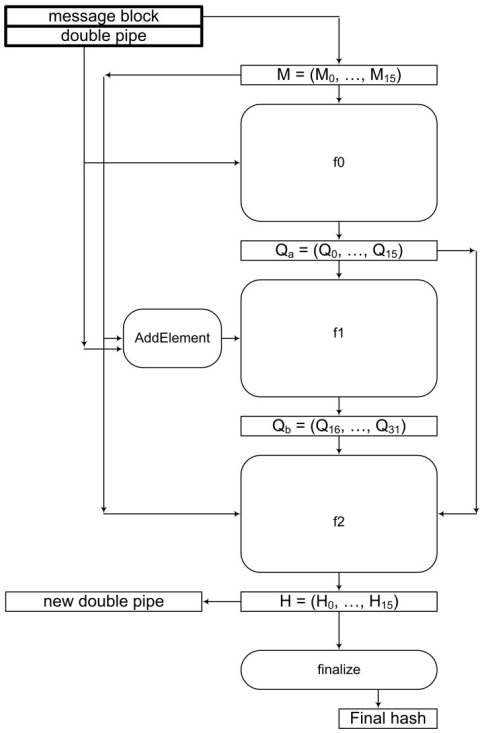

Fig. 1: Graphical representation of the hash function Blue Midnight Wish

The rest of the paper is organized as follows. In Section 2, we describe briefly the compression function of the second round version of the BMW algorithm, while Section 3 contains the architectural description of the BMW-256 design. In Section 4, the BMW hashing operations are detailed out. In section 5, the synthesis results of the FPGA implementation are given and comparisons with other related works are shown. Finally, in section 6, our conclusions are presented, and some observations and possibilities for future work are discussed.

II. The Hash Function of BLUE MIDNIGHT WISH

The BMW-n hash function is shown in Fig. 1. We refer to the variant that creates a 256 bit message digest as BMW-256 and the variant that creates a 512bit message digest as BMW-512. The basic data block which is used for BMW-256 is 32 bits long and for BMW-512 is 64 bits long. The algorithm has four different operations in the hash computation stage: bit-wise logical word XOR, word addition and subtraction, shifts (left or right), and rotate left. The BMW uses a double pipe design to increase the resistance against generic multi-collision attacks and length extension attacks. In the double pipe design, the sizes of the inputs to the compression function are twice the message digest size. The inputs to the compression function are the message blocks $M^{(i)}$, along with the initialization vector $H^{(i-1)}$ (previous double pipe) and the output is the current double pipe $H^{(i)}$.

The hash function has two main parts: 1. Message digesting part and 2. Finalization part as it is shown in Fig. 1. The first part uses three separate functions f_0, f_1 and f_2 to define the so called "compression function" of Blue Midnight Wish. The output of the compression function is $H^{(i)} = (H_0^{(i)}, H_1^{(i)}, ..., H_{15}^{(i)})$. There are two inputs for the function f_0: The first argument consists of sixteen n-bit words, which are working as initial values $H_0^{(i-1)}$, $H_1^{(i-1)}$,.., $H_{15}^{(i-1)}$. The second argument consists of sixteen n-bit words, which represent the input message block: $M_0^{(i)}$, $M_1^{(i)}$,..., $M_{15}^{(i)}$.

The function $f_0(M^{(i)}, H^{(i-1)})$ computes $M^{(i)} \oplus H^{(i-1)}$ and produces $Q_a^{(i)}$ as the first part of the extended (quadrupled) pipe, hence $Q_a^{(i)} = (Q_0^{(i)}, Q_1^{(i)}, ..., Q_{15}^{(i)})$. The inputs for the function f_1 are three different arguments, the message block $M^{(i)}$, the previous double-pipe $H^{(i-1)}$ and the value of $Q_a^{(i)}$. The function $f_1(M^{(i)}, H^{(i-1)}, Q_a^{(i)})$ computes the second part of the extended (quadrupled) pipe $Q_b^{(i)}$, hence $Q_b^{(i)} = (Q_{16}^{(i)}, Q_{17}^{(i)}, ..., Q_{31}^{(i)})$.

The third function f_2 also takes three arguments; the message block $M_{(i)}$ and the values of both $Q_a^{(i)}$ and $Q_b^{(i)}$. The function $f_2(M^{(i)}, Q_a^{(i)}, Q_b^{(i)})$ computes the new double-pipe value $H^{(i)}$, i.e. $H^{(i)} = (H_0^{(i)}, H_1^{(i)}, ..., H_{15}^{(i)})$.

The second part (finalization) contains of the same compression function defined in the message digesting part (so it uses the same functions f_0, f_1 and f_2), but instead of initial values $H_0^{(i-1)}$, $H_1^{(i-1)}$,..., $H_{15}^{(i-1)}$, it use $Constant_j^{final}$

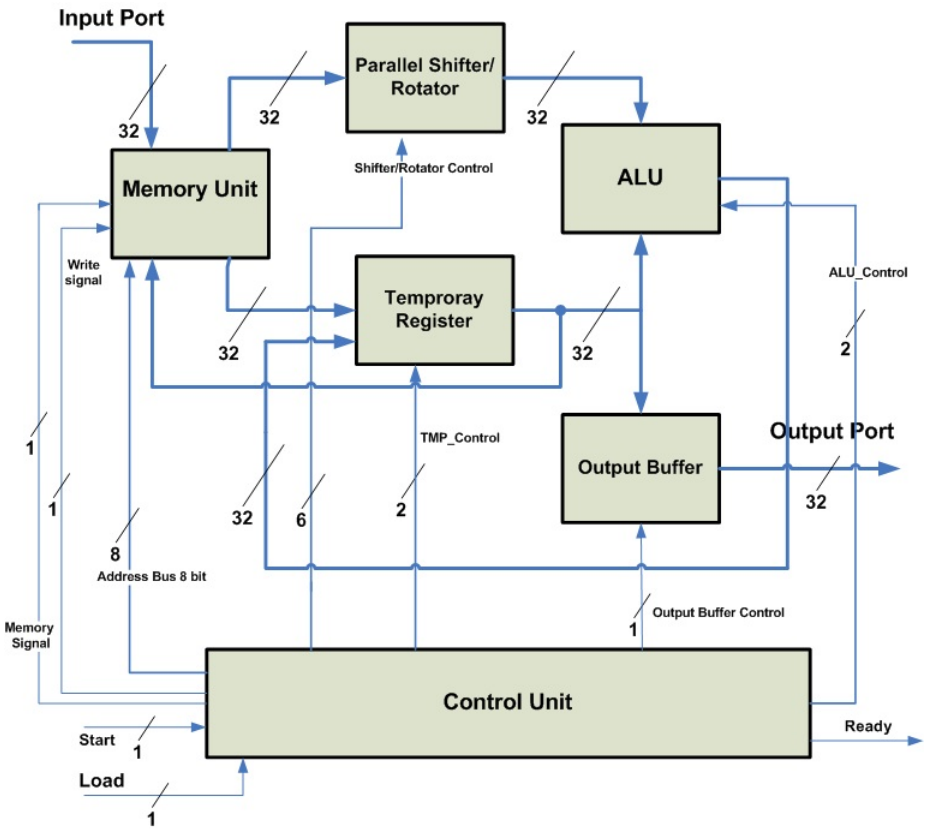

Fig. 2: BLUE MIDNIGHT WISH-256 Core Architecture

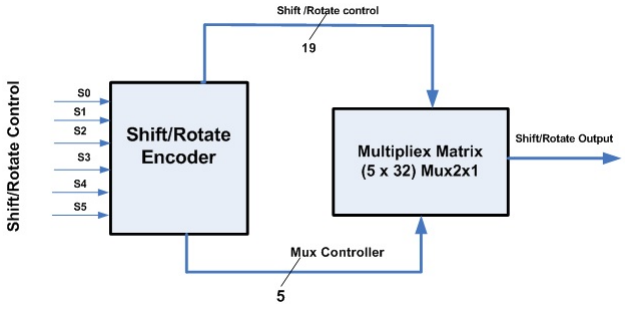

Fig. 3: Parallel Shifter/Rotator Block (BMW-256)

= ($Constant_0^{final}$, $Constant_1^{final}$,... , $Constant_{15}^{final}$) values and the role that was played by the input message block in the previous message digesting part, now will be played by the last obtained double-pipe $H^{(i)}$ = ($H_0^{(i)}$, $H_1^{(i)}$,..., $H_{15}^{(i)}$).

III. Blue Midnight Wish256 Core Architecture

Fig. 2 shows the complete architecture of the entire BMW core-256 process, which includes six main hardware operative parts, Memory unit, Parallel Shifter/Rotator, ALU (Arithmetic Logic Unit), Temporary Register, Output Buffer and Control Unit. Their operations are as follows:

Parallel Shifter/Rotator: It contains a 5 x 32 Mux matrix each one is a 2 x 1 multiplex with a large encoder (5 X 11). This component is responsible for the shift and rotation operations of the 32 bit words. It receives 32 bit parallel data from the memory Block and transmits 32 bit parallel data to the ALU. That happens dependent on the value of the shifter control word. Because we have 36 operations in the BMW hash core, the width of shifter control word is 6 control bits as shown in Fig. 3 and Appendix A.

ALU : The ALU component as shown in Appendix B offers three different operations in the hash computation stage: bit-wise logical word XOR, word addition and subtraction (modulo 2^{32}). The ALU component receives 32 bit data words from the Parallel Shifter/Rotator and the Temporary Register and transmit the output to the Temporary

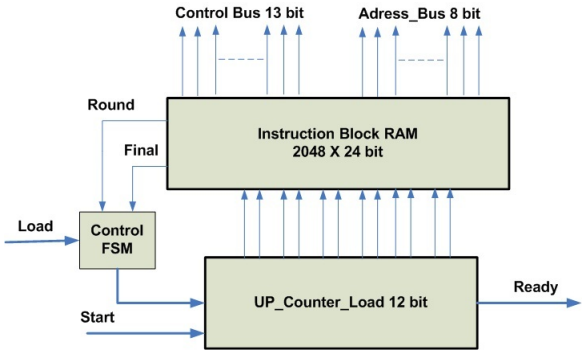

Fig. 4: BMW-256 Control Unit

Register to work as a parallel accumulator. This idea has been already applied in a similar way in the implementation of Beuchat et al., of the hash function BLAKE [9].

Temporary Register: It contains a 32 Mux 2 x 1 and a shift register. The Temporary Register works as an accumulator. It receives 32 bit words from The Memory Unit and The ALU and transmits data 32 bit words to the ALU and the output stage.

Memory unit: To implement the BMW-256 core memory block, we used an FPGA block RAM of size 256 x 32 bits. As we mentioned in section 3.1, the memory block contains a ROM to store the BMW-256 constants K_j, J=0,1,..., 15 , $H^{(i-1)}$ and the $Constant_j^{final}$. In addition, the memory block contains sufficient RAM to store the BMW-256 input message blocks ($M_0^{(i)}$, $M_1^{(i)}$,..., $M_{15}^{(i)}$), the intermediate values of the BMW hash function, and the final double pipe values $H^{(i)}$ = ($H_0^{(i)}$, $H_1^{(i)}$, $H_2^{(i)}$,...,$H_{15}^{(i)}$).

Output Buffer: After the finalization stage, the Output buffer will receive the final digest results.

Control Unit: It has been designed as a 2048 x 25 bit Instruction Block RAM, an 11 bit up_counter_load bit and a Control FSM (Finite State Machine) as shown in Fig. 4. it contains three operative parts, all of them working together to produce 8 bit memory address words to control the memory block traffic with the other BMW-256 sub-systems. Instructions Block RAM translated after placement and routing to one 36K Block RAM and one 18K block RAM. The Control Unit produces the 13 bit control word to control the data flow between the BMW-256 core sub-systems. The Control Unit subsystems are working as follows:

once the Start and Load signals becomes high, the organization of the sixteen input messages inside RAM location is started. Subsequently, the Load signal becomes low and the Instruction Block RAM starts to control the BMW hashing core to execute the f_0, f_1, and f_2 according to the BMW-256 algorithm operations which was described in section 2. Finally, the Round signal becomes high, and BMW hashing core starts to transfer the $H_0^{(i)}$,$H_1^{(i)}$,...,$H_{15}^{(i)}$ values in the message locations and and transfer $Const^{final}$ vaules in the $H_0^{(i)}$,$H_1^{(i)}$,...,$H_{15}^{(i)}$ locations. After that the Final signal becomes high and the final hash output. The Control FSM is used to organize the movement of instructions from the up_counter_load according to the value of each of the signals Load, Round and Final.

IV. Blue Midnight Wish hashing operations

In this section we describe how the computation hash core works to execute the internal functions in BMW. As an example, we will explain how to XOR two blocks of data present in locations number 4 and 5 in the Memory Unit, and write the result in location number 7. First, the Control Unit gives order to the Memory Unit to choose location number 4. Then the Control Unit asks the Temporary Register to pick up the data from the data bus and subsequently the same operation happens with location number 5. However, instead of using the Temporary Register, the Parallel Shifter/Rotator picks up the data. Now, the Control Unit asks the Shift/Rotate Encoder to give order to the ALU to add these data and store them in the Temporary Register. Finally, the Control Unit gives order to the Memory Unit to pick up the data and place them in location number 7. Because we used the Parallel Shift/Rotate, and the parallel Arithmetic Logic Unit , we succeeded to reduce the number of cycles for each operation shown in Table I (page 5). Using the BMW operations in Table I, we see that we can execute the function f_0 in 426 cycles, function f_1 in 452 cycles and finally function f_3 in 170 cycles.

V. Performance Evaluation

The BMW-n core has been designed in VHDL and it was synthesized (synthesis, placement and routing) using ISE foundation 12.3 [7] in VIRTEX 5 XC5VLX110 Xilinx device as shown in Appendix C . In Table II, we compare this implementation optimized for small FPGAs with the previous similar implementation. By using the proposed

TABLE I: BLUE MIDNIGHT WISH hashing core operations (execution times)

Operation	Proposed	BMW-256[11]
Load	1	1
XOR	1	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

TABLE II: BLUE MIDNIGHT WISH performance results

Algorithm Name	FPGA Type	Area(Slice)	Frequency [MHZ]	Throughput	Memory Blocks	Throughput/Area (Slice)
Proposed BMW-256	Virtex5 XC5VLX110	51	141	68.71 Mbps	3	1.35
Proposed BMW-512	Virtex5 XC5VLX110	105	115	112.18 Mbps	3	1.09
BMW-256 [8]	Virtex5 XC5VLX110	1980	264	5Mbps	—	0.0025

structure we have spent around 97% less area compared to previous design for BMW-256 on the same FPGA VIRTEX 5 XC5VLX110 device while increasing the measured throughput around 14 times.

VI. CONCLUSION AND FUTURE WORK

In this paper we have presented an FPGA implementation of a new BMW hashing core structure using a parallel shifter/rotator and a parallel arithmetic logic unit (ALU) . The BMW core receives 16 message words and processes them. The goal was to use as small area as possible in order to minimize the hardware cost. For the future work, we will take on the challenge to improve this design. The goal is to improve the throughput while keeping the optimized the area usage. It will certainly be beneficial in some future usage scenarios to do a full implementation in ASIC.

ACKNOWLEDGEMENT

We would like to thank Jean-Luc Beuchat for his useful comments and hints how to improve the results presented in this paper.

REFERENCES

[1] National Institute of Standards and Technology, *"Secure Hash Standard (SHS), FIPS PUB 180-3"*, Federal Information Processing Standards Publication, October 2008, http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[2] 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.

[3] NIST (2006). *"NIST Comments on Cryptanalytic Attacks on SHA-1"*. http://csrc.nist.gov/groups/ST/hash/statement.html

[4] 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

[5] D. Gligoroski, V. Klima, S. J. Knapskog, M. El-Hadedy, Jorn Amundsen and S. F. Mjolsnes, *"Cryptographic Hash Function BLUE MIDNIGHT WISH"*, Submission to NIST (Round 2) of SHA-3 Competition, September 2009

[6] D. Gligoroski, V. Klima, *"A Document describing all modications made on the Blue Midnight Wish cryptographic hash function before entering the Second Round of SHA-3 hash competition"*, http://people.item.ntnu.no/~danilog/Hash/BMW-SecondRound/Supporting_Documentation/Round2Mods.pdf

[7] Xilinx, *"Device Package User Guide"*, 2010 http://www.xilinx.com/support/documentation/user_guides/ug112.pdf

[8] 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.

[9] J.-L. Beuchat, E. Okamoto, and T. Yamazaki, *"Compact Implementations of BLAKE-32 and BLAKE-64 on FPGA"*, In Proceedings of the 2010 International Conference on Field-Programmable Technology (FPT 2010), early version available at http://eprint.iacr.org/2010/173.pdf

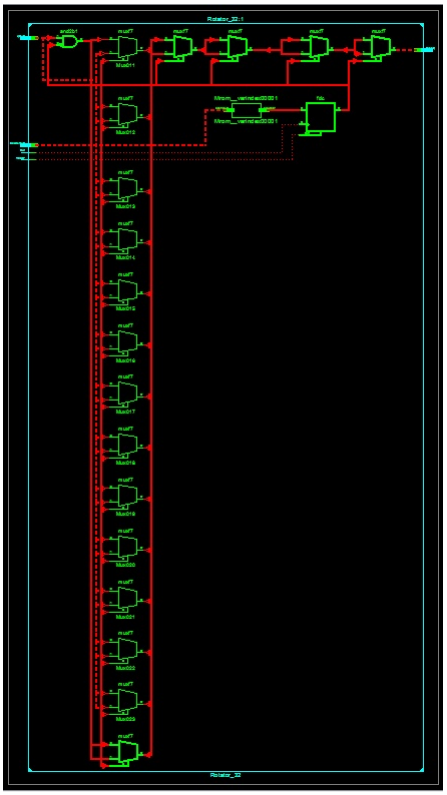

Fig. 5: Parallel Shifter/Rotator 32bits

Appendix A

We implement the Parallel Shifter/Rotator using MUXF7 matrix with Fixed ROM to control the data flow as shown in Fig.5. MUXF7 is 2-to-1 Lookup Table Multiplexer with General Output. *MUXF7 provides a multiplexer function in a full Spartan-3, Virtex-II, Virtex-II Pro, and Virtex-II Pro X CLB for creating a function-of-7 lookup table or a 16-to-1 multiplexer in combination with the associated lookup tables. Local outputs (LO) of MUXF6 are connected to the I0 and I1 inputs of the MUXF7. The S input is driven from any internal net. When Low, S selects I0. When High, S selects I1[7].*

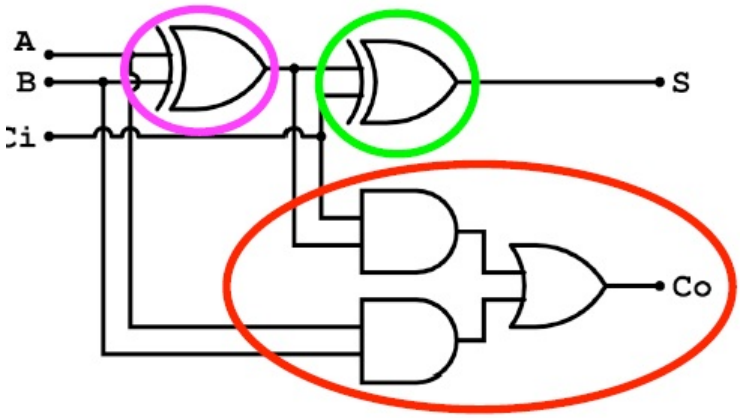

Fig. 6: One Bit Full Adder

Appendix B

It is possible to map Carry ripple adder as shown in Fig.6 onto carry chain block as shown in Fig.7. By this way we can implement the ALU component using small area and low delay as shown in Fig.8 . Each 4-bit ALU contains 4 LUT6_2 and CARRY4. For BMW-256 , the ALU will use eight 4-bit ALU and for BMW-512 will use sixteen 4-bit ALU. Table.III contains the ALU implementations results for 32 and 64 bits versions. The LUT6_2 component is a six-input, two-output Look-Up Table (LUT) that can either act as a dual asynchronous 32-bit ROM (with 5-bit addressing), implement any two 5-input logic functions with shared inputs, or implement a 6-input logic function and a 5-input logic function with shared inputs and shared logic values. The CARRY4 as shown in Fig.9 represents the fast carry logic for a slice. The carry chain consists of a series of four MUXes and four XORs that connect to the other logic (LUTs) in the slice via dedicated routes to form more complex functions. The fast carry logic is useful for building arithmetic functions like adders, counters, subtractors and add/subs, as well as such other logic functions as wide comparators, address decoders, and some logic gates (specifically, AND and OR). This is discussed in [7] and has been already applied in the implementation of Beuchat et al., of the hash function BLAKE [9].

TABLE III: ALU using Carry Chain block

Name	Area(slice)	Delay
ALU_32	8	6.930 ns
ALU_64	16	9.544 ns

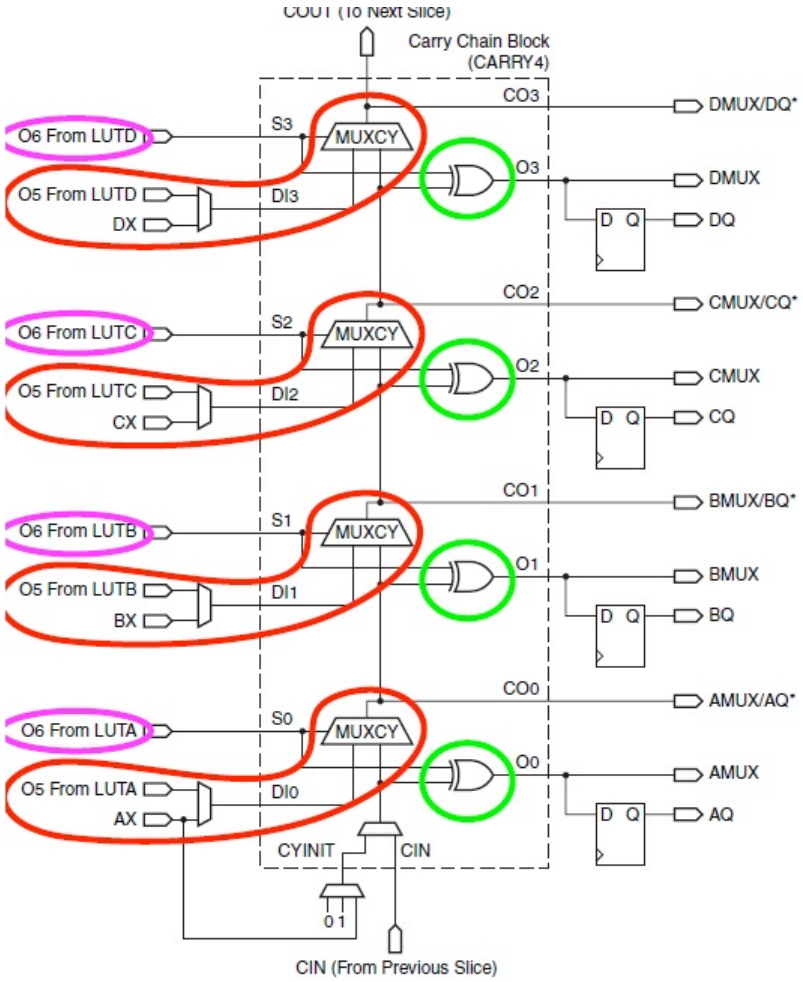

Fig. 7: Virtex 5 Micro-architecture

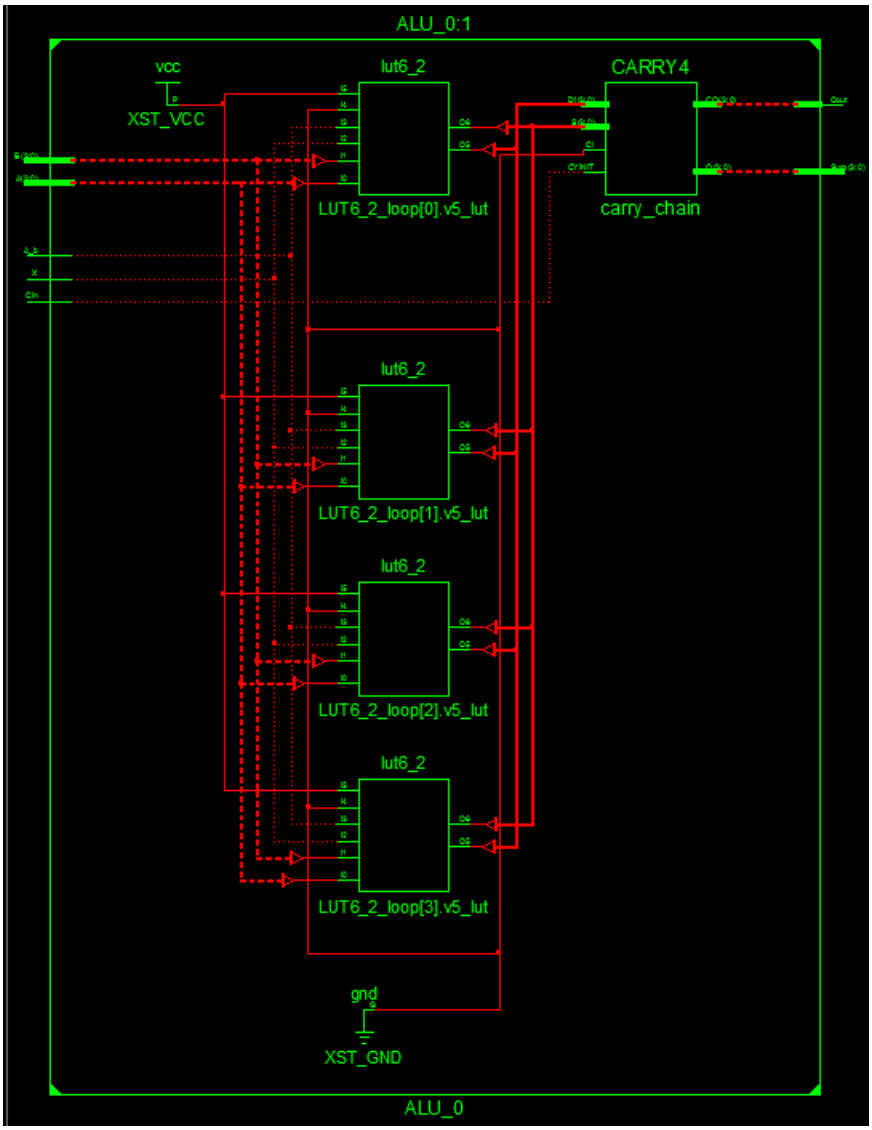

Fig. 8: 4 bit ALU

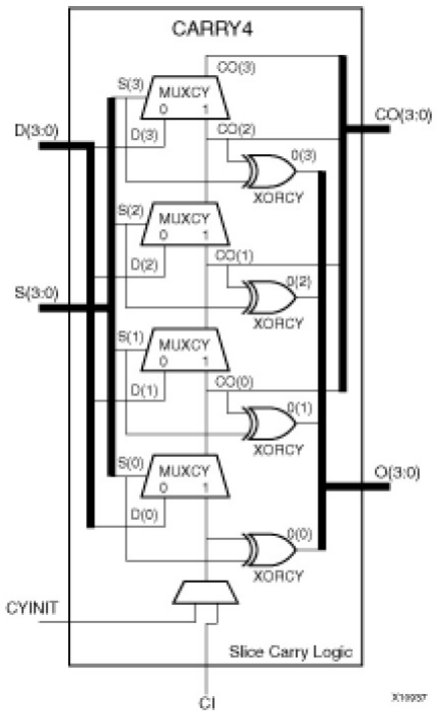

Fig. 9: Fast Carry Logic with Look Ahead

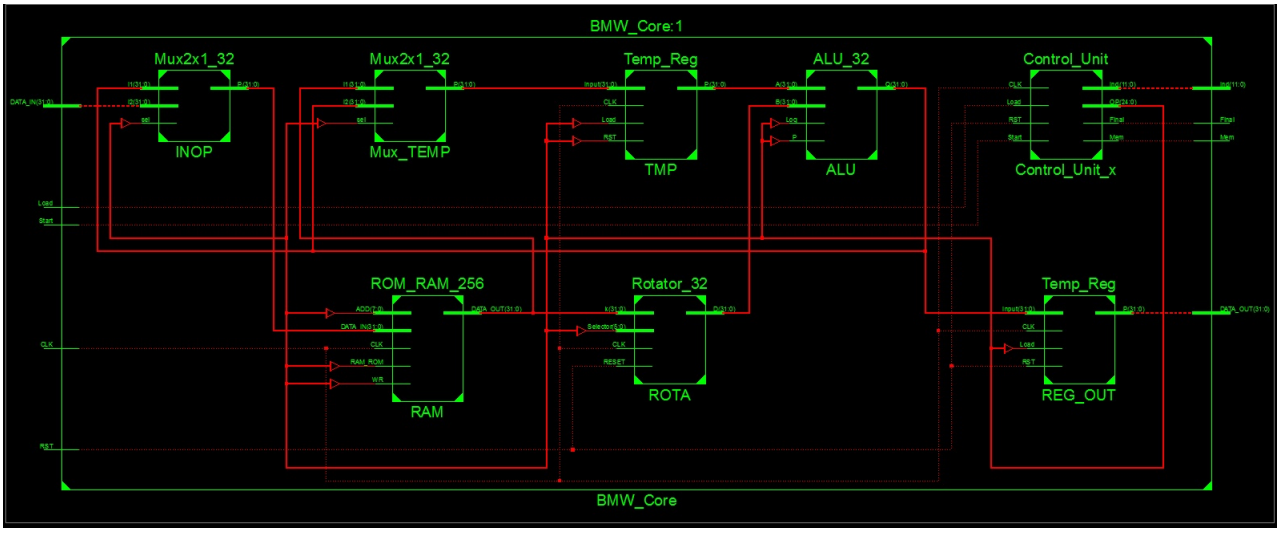

Fig. 10: RTL schematic for BMW-256

Appendix C

In this section we have the all details for BLUE MIDNIGHT WISH implementations using single Core on FPGA VIRTEX 5. Fig.10 contains the RTL schematic for BMW-256, Fig.11 contains the Device Utilization Summary for BMW-512 after placement and routing.Fig.12 contains the Device Utilization Summary for BMW-512 after placement and routing.

BMW_Core Project Status (10/08/2010 - 12:32:56)			
Project File:	BMW_256.xise	**Parser Errors:**	No Erro
Module Name:	BMW_Core	**Implementation State:**	Placed
Target Device:	xc5vlx110-3ff676	**•Errors:**	No Erro
Product Version:	ISE 12.3	**•Warnings:**	267 Wa
Design Goal:	Area Reduction	**•Routing Results:**	All Sign
Design Strategy:	Area Reduction with Physical Synthesis	**•Timing Constraints:**	All Con
Environment:	System Settings	**•Final Timing Score:**	0 (Timi

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	80	69,120	1%
Number used as Flip Flops	64		
Number used as Latch-thrus	16		
Number of Slice LUTs	203	69,120	1%
Number used as logic	202	69,120	1%
Number using O6 output only	121		
Number using O5 output only	11		
Number using O5 and O6	70		
Number used as exclusive route-thru	1		
Number of route-thrus	12		
Number using O6 output only	12		
Number of occupied Slices	51	17,280	1%
Number of LUT Flip Flop pairs used	203		
Number with an unused Flip Flop	123	203	60%
Number with an unused LUT	0	203	0%
Number of fully used LUT-FF pairs	80	203	39%
Number of unique control sets	4		
Number of slice register sites lost to control set restrictions	4	69,120	1%
Number of bonded IOBs	82	440	18%
IOB Flip Flops	44		
Number of BlockRAM/FIFO	3	128	2%
Number using BlockRAM only	3		
Number of 36k BlockRAM used	2		
Number of 18k BlockRAM used	1		
Total Memory used (KB)	90	4,608	1%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Average Fanout of Non-Clock Nets	3.17		

Fig. 11: Device Utilization Summary for BMW-256

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	124	69,120	1%
Number used as Flip Flops	97		
Number used as Latch-thrus	27		
Number of Slice LUTs	398	69,120	1%
Number used as logic	397	69,120	1%
Number using O6 output only	230		
Number using O5 output only	28		
Number using O5 and O6	139		
Number used as exclusive route-thru	1		
Number of route-thrus	28		
Number using O6 output only	28		
Number of occupied Slices	105	17,280	1%
Number of LUT Flip Flop pairs used	408		
Number with an unused Flip Flop	284	408	69%
Number with an unused LUT	10	408	2%
Number of fully used LUT-FF pairs	114	408	27%
Number of unique control sets	4		
Number of slice register sites lost to control set restrictions	7	69,120	1%
Number of bonded IOBs	146	800	18%
IOB Flip Flops	76		
Number of BlockRAM/FIFO	3	128	2%
Number using BlockRAM only	3		
Number of 36k BlockRAM used	2		
Number of 18k BlockRAM used	1		
Total Memory used (KB)	90	4,608	1%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Average Fanout of Non-Clock Nets	3.25		

Fig. 12: Device Utilization Summary for BMW-512