Compact Implementation of Skein-256 Hash Function on FPGA

Dure-Shahwar Kundi
Department of Electrical Engineering
National University of Sciences & Technology
H-12, Islamabad, Pakistan
durshi1@gmail.com

Arshad Aziz
Department of Electrical Engineering
National University of Sciences & Technology
H-12, Islamabad, Pakistan
arshad@nust.edu.pk

Abstract— **In this paper we have presented a low-area design of Skein-256 hash function on Xilinx Virtex-5 FPGA. Skein is one of the 5 finalists of SHA-3 competition. Our implementation is utilizing only 527 slices and shows a throughput of 1.295 Gbps. Comparing our results with the previous published FPGA implementations of Skein-256, our design reports the best TPS of 2.457. High value of TPS shows the efficiency of our design.**

Keywords- Cryptography, Hash, SHA-3

I. INTRODUCTION

A hash algorithm takes a variable size data as input and gives its equivalent compressed representation as output. But a cryptographic hash algorithm is different, in that it is designed to serve for the computer security purposes such as Message authentication and integrity, secure web connections and password Logins [1].

Skein is one of the five cryptographic hash algorithms that have successfully proceeded to the final round of the SHA-3 cryptographic hash algorithm public competition that started in 2010 [2] and will be finalized in the end of 2012. The need for a new cryptographic hash algorithm was deemed necessary after the previous standard hash algorithms were successfully attacked by cryptanalysts rendering the information technology industry at stake.

II. SKEIN

Skein is a group of cryptographic hash functions for the three internal state sizes: 256, 512 and 1024 bits. It consists of three components [3]:

A. Threefish Block Cipher

Threefish is the tweakable block cipher at the core of Skein, defined with a 256-, 512-, and 1024-bit block size.

B. Unique Block Iteration (UBI)

UBI is a chaining mode that uses Threefish to build a compression function that maps an arbitrary input size to a fixed output size.

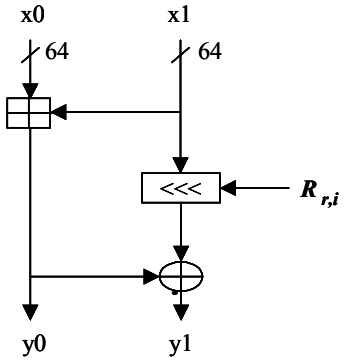

Figure 1. The MIX Function

C. Optional Argument System

This allows Skein to support a variety of optional features without imposing any overhead on implementations and applications that do not use the features.

The core of Skein is built upon the tweakable block cipher [4]. Threefish works towards the building of the compression function of Skein. It is designed for input block sizes of 1024, 512 and 256-bits. For every input message, it divides the block into equal sizes of 64-bit words (***Nw***) and performs a simple non - linear MIX operation and permutation for every pair of word. The MIX function consists of single addition, shift-and- rotate operation and XOR as shown in Fig.1. The *x0* and *x1* represents the two 64-bit input words while *y0* and *y1* the corresponding 64-bit output words.

TABLE I. ROTATION CONSTANT FOR SKEIN-256

Nw		**4**	
i		**0**	**1**
r =	0	14	16
	1	52	57
	2	23	40
	3	5	37
	4	25	33
	5	46	12
	6	58	22
	7	32	32

978-1-4577-1964-6/12/$31.00 ©2012 IEEE

TABLE II. VALUES FOR THE WORD PERMUTATION

		i =															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Nw =	4	0	3	2	1												
	8	2	1	4	7	6	5	0	3								
	12	0	9	2	13	6	11	4	15	10	7	12	3	14	5	8	1

$R_{r,i}$ is the rotational constant that repeats after every eighth round and is different for the two sets of 64-bit words in every round. The number of bits for shifting and rotation depends on the rotation constants as given in Table I.

The two output words as depicted in Fig.1 go under the permutation operation, which is same for every round. The values for the word permutation are fixed and are given in the Table II for the all Threefish variants. Since Threefish is iterative in nature, the MIX and permutation operations are performed repetitively in 72 rounds for state sizes 256 and 512 bits and in 80 rounds for state size 1024 bits. Threefish-1024 is similar, except that it has eight MIX functions per round and 80 rounds total. The rotation constants and round permutations are different for each Threefish version, and were selected to maximize diffusion across the entire Threefish block.

Fig.2. illustrates the example of Threefish-256. After every fourth round, it injects a subkey. The 18 subkeys are derived through a combination of cipher keys and tweak values by a separate module of key scheduling. To create the key schedule, the key and tweak are each extended with one extra parity word that is the XOR of all the other words. Each subkey is a combination of all but one of the extended key words, two of the three extended tweak words, and the subkey number.

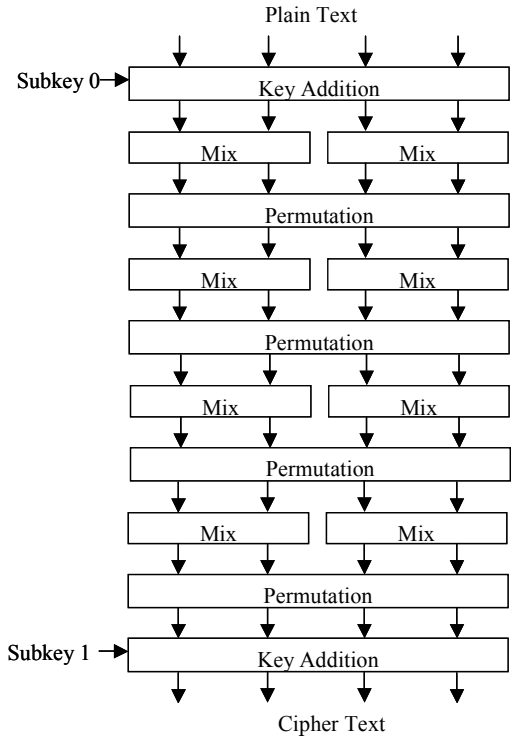

Figure 2. Four of the 72 Rounds of Skein 256 Block cipher

III. IMPLEMENTATION

In this work, we take in account the implementation of Sequential Design. Our design can be divided into two parts, control and data paths. The control path consists of clock, counter; Finite State Machine and State register while the data path consists of Input and Output registers to store the input message and the final hash values respectively, Input Key, Round_A and Round_B incorporating the mix and permute functions and Add Subkey module for the addition of key with the input message in each round. The complete module design is shown in Fig. 3.

Round_A and *Round_B* modules as shown in Fig. 3 are same, except the value of left shift constant R involved in Mix operation is different. In *Round_A* module round 1-4 is implemented completely with their rotational constants while in *Round_B* module round 5-8 are implemented. As described in Section II each round consists of Mix and Permutation operations. The complete design for both the *Round_A* and *Round_B* modules are given in Fig.4. Initially the plaintext message has been added to the input key before provided to the *Mux-1*. The first multiplexer *Mux-1* selects between the input data only for the first clock cycle and the feedback data for the remaining 72 rounds. So the logic 0 at select input *S2* will select the input data while it will select the feedback data when the logic is 1. The resulting output is then passes to the *Demux_1*. Both the *Demux_1* and *Mux_2* are control by the same select input S1. If S1 is at logic 1, the data path through module *Round_A* will be selected otherwise the data path will be all the way through module *Round _B*. The output of *Mux_2* is then fed to the Add_Subkey module to add it with subkey1.

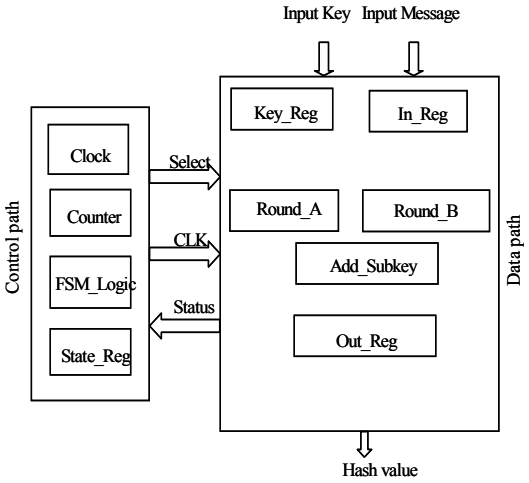

Figure 3. Seqential Design

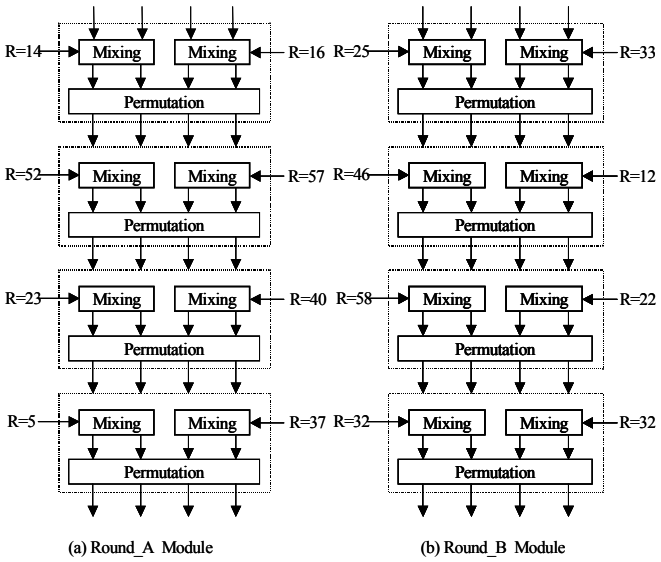

Figure 4. Round_A and Round_B Modules

The Add_Subkey module is a 256-bit adder having input keys from key scheduler but here in our design we calculate the subkeys on-the-fly.

The execution of both the modules A and B occurs on the positive edge of each clock pulse and the next subkey is available on negative edge of the same clock. In this way the complete four rounds module and subkey addition executed in one clock cycle. Therefore to complete 72 rounds and 19 Subkey addition of Skein-256, 18 clock cycles will be required instead of 72 clock cycles. Final hash value will be available after the latency of 18 cycles at the output of the XOR gate. Complete Hardware architecture using sequential technique is depicted in Fig. 5.

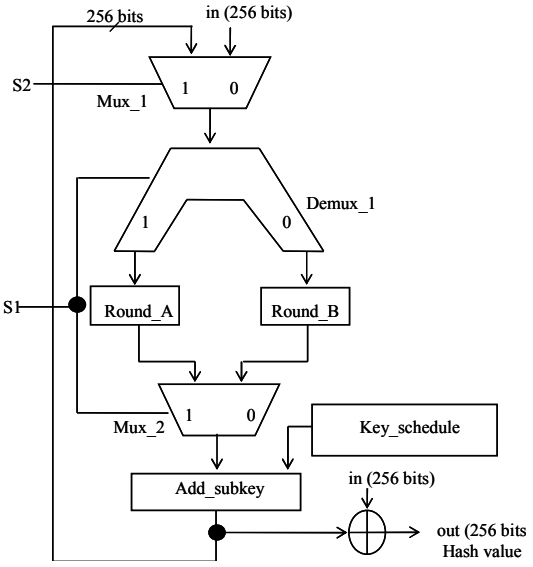

Figure 5. Hardware Architecture of Skein-256

TABLE III. THROUGHPUT AND TPS OF OUR SEQUENTIAL DESIGN OF SKEIN-256

FPGA	Area (Slices)	Frequency (MHz)	Throughput (Gbps)	TPS (Throughput/ Area)
Virtex5	527	91.116	1.295	2.457

IV. IMPLEMENTATION RESULTS

For synthesis we have used the ISE tools (v12.3) targeting the Xilinx FPGAs. The target device was a Xilinx Virtex 5 (xc5vlx110-3ff1760). The device resource usage in terms of number of slices and clock frequency estimation after Place and Route for the above FPGA architecture, are reported in Table III.

The throughput of a given design can be calculated by (1).

$$Throughput\,(TP) = \frac{BlockSize}{T} \tag{1}$$

Where *BlockSize* is the size of message in bits, which is 256 bits for Skein-256 in our case. *T* is the total time required to calculate hash value, which is given by (2).

$$T = TimeDelay(t) \cdot ClockCycles \tag{2}$$

As mentioned above total of 18 clock cycles are required to calculate the final hash value. Table III shows the Area, Frequency, Throughput per area (TPS) in terms of slices and Throughput (TP) results for our Skein-256 design.

V. COMPARISON WITH PREVIOUS WORK

There are only a few existing low area FPGA implementations of the Skein-256 up to now. Table V shows the comparison of results with available low area implementations in terms of area, TP and TPS. Our first primary focus was the area utilization with sufficient TP taking in account the best TPS ratio. There is a tradeoff between the area and throughput, so TPS is the parameter to judge the efficiency of the design.

TABLE IV. RESULTS COMPARISON OF SKEIN-256

	Area (Slices)	TP (Gbps)	TPS
A. H. Namin [5]	1385	0.161	0.116
S. Kerckhof [6]	291	0.223	0.766
M. Long [7]	7029	0.409	0.058
B. Baldwin [8]	3027	0.973	0.321
S. Matsuo [9]	854	0.393	0.460
E. Homsirikamol [10]	1364	1.307	0.958
K. Gaj [11]	843	1.567	1.859
Our Design	**527**	**1.295**	**2.457**

The design given by S. Kerckhof [6] is utilizing minimal resources because of the data path of 128-bit instead of 256-bits and reports a latency of 466 clock cycles, which results in poor TP as compared to other designs. Also the TPS of his design is 0.766 that is less than the last three results reported in the comparison table.

The TP reported by E. Homsirikamol [10] and K. Gaj [11] is better but at the cost of more resources than ours. The TPS of their designs are 0.958 and 1.859 respectively but their slice utilization is more than our compact design. The comparison table clearly shows the effectiveness of our design with best TPS of 2.457 as compared to all other implementations reported here in this paper.

VI. CONCLUSION

In this work we presented the design of our low-area hardware implementation of the cryptographic hash function Skein-256. We present the performance figures of our implementation in terms of throughput, area and throughput/area and also we compare our results with available results. Achieved results are beyond the various published renowned implementations because of our design methodology.

REFERENCES

[1] J. Menezes, P. C. Van Oorschot, S. A. Vanstone: 'Handbook of Applied Cryptography'. The CRC Press series on discrete mathematics and its applications, pp.321-383, 1997.

[2] National Institute of Standard and Technology (NIST): Cryptographic Hash Algorithm Competition Website: http://csrc.nist.gov/groups/ST/hash/sha-3/.

[3] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, J. Walker: 'The Skein Hash Function Family Version 1.3', http://www.skein-hash.info/sites/default/files/skein1.3.pdf, Oct 2010, pp. 1-100

[4] M. Liskov, R. Rivest, and D. Wagner: 'Tweakable Block Ciphers', Advances in Cryptology, CRYPTO 2002 Proceedings, Springer-Verlag, 2002, pp. 31-46

[5] H. Namin and M. A. Hasan: 'Hardware implementation of the compression functions for selected sha-3 candidates'. CACR 2009-28, 2009. http://www.vlsi.uwaterloo.ca/~ahasan/hasan_report.html.

[6] S. Kerckhof, F. Durvaux, N. V. Charvillon, F. Regazzoni, G. M. de Dormale, F. X. Standaert: 'Compact FPGA Implementations of the Five SHA-3 Finalists'. CARDIS 2011: 217-233

[7] M. Long: 'Implementing Skein Hash function on Xilinx Virtex-5 FPGA platform', http://www.skeinhash. info/sites/default/files/skein_fpga.pdf, Feb 2009, pp. 1-15

[8] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. Neill and W. P. Marnane: 'FPGA Implementations of the Round Two SHA-3 Candidates', 2nd SHA-3 Candidate Conference, Santa Barbara, August 23-24, 2010, pp. 1-18

[9] S. Matsuo, M. Knezevic, P. Schaumont, I. Verbauwhede, A. Satoh, K. Sakiyama and K. Ota: 'How Can We Conduct Fair and Consistent Hardware Evaluation for SHA-3 Candidate?', 2nd SHA-3Candidate Conference, Santa Barbara, August 23-24, 2010, pp. 1-15

[10] E. Homsirikamol, M. Rogawski, and K. Gaj: 'Comparing hardware performance of round 3 SHA-3 candidates using multiple hardware architectures in Xilinx and Altera FPGAs', May, 2011, ECRYPT II Hash Workshop 2011.

[11] K. Gaj, E. Homsirikamol, and M. Rogawski: 'Comprehensive Comparison of Hardware Performance of Fourteen Round 2 SHA-3 Candidates with 512-bit Outputs Using Field Programmable Gate Arrays', 2nd SHA-3 Candidate Conference, Santa Barbara, August 23-24, 2010, pp. 1-14.