Implementing Skein Hash Function on Xilinx Virtex-5 FPGA Platform

Men Long, Intel Corporation, 02-Feb-09, Version 0.7
men.long@intel.com

1 Introduction

This paper describes an implementation of the Skein hash function based on a Xilinx FPGA. Section 2 gives a high-level overview of the Xilinx FPGA architecture needed to understand the implementation. Section 3 describes the implementation details. Section 4 provides a performance analysis of the implementation.

2 Background of Xilinx Virtex-5 FPGA Architecture

We first describe the parts of the FPGA architecture that can help us understand how to implement the arithmetic used by the Skein hash function. Fig.1 shows the major components of a Xilinx Virtex-5 FPGA. The configurable logic block (CLB) is used to implement combinational or sequential logic. A modern FPGA also incorporates certain ASIC circuits for the high-speed I/O and the common logic such as digital signal processing (DSP) of adders/multipliers. In addition, since memory is usually an indispensable component in any digital system, the FPGA chip also includes the block RAM (BRAM). Fig. 1 also illustrates the common digital design components of buffer and clock management tile (CMT).

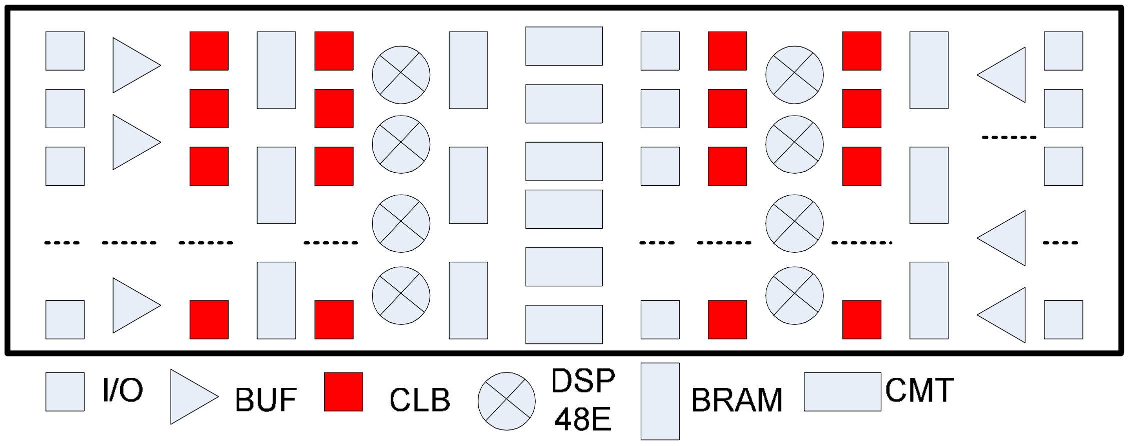


Fig. 1: The simplified diagram on the components of a Xilinx Virtex-5 FPGA. Note that we omit the on-chip switch matrix that interconnects those components.

The CLB is the most important component of an FPGA for implementing reconfigurable logic. Fig. 2 shows the internal arrangement of CLB, where two slices (labeled 0 and 1) form a CLB. In terms of the circuit wire routing, the on-chip switch matrix can interconnect multiple CLBs to construct a digital system that exceeds the logic capacity of a single CLB. In addition, each slice within a CLB has a fast path on the carry in (CIN) and carry out (COUT) for basic arithmetic

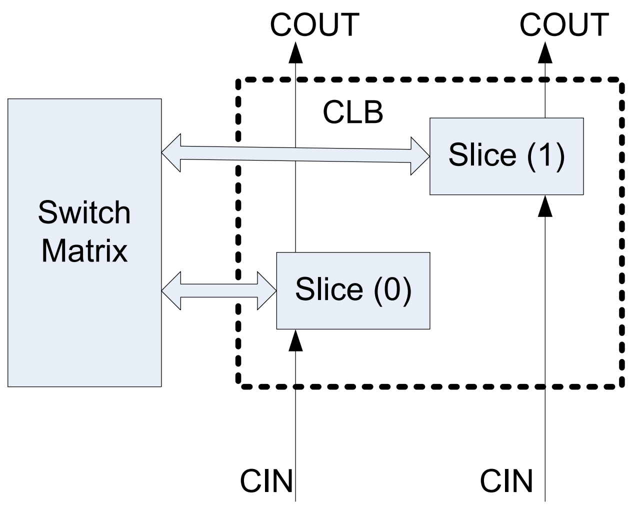


Fig. 2: Arrangement of slices in a CLB of Xilinx Virtex-5 FPGA. CIN and COUT are for the carry-in and carry-out, respectively. The switch matrix interconnects different CLBs to construct a digital system exceeding the capacity of one CLB.

A slice can be further decomposed into lookup tables (LUT) [1]. For the Xilinx Virtex 5, a slice consists of 4 such LUTs and flip-flop pairs, as shown in Fig. 3. In literature, LUT is a basic unit for the reconfigurable logic. The LUT can implement any digital logic truth table, constrained only by the number of signal inputs and outputs. In the example of Virtex-5, the LUT could implement any logic equations with the 6-bit input and the 1 bit-output. The flip-flops in a slice can register the output for the sequential logic.

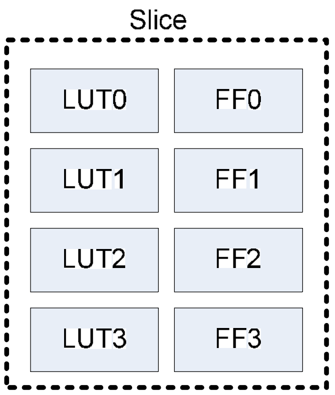


Fig. 3: Four pairs of Lookup table (LUT) and Flip-Flop (FF) in a CLB slice of Xilinx Virtex-5 FPGA.

3 Hardware Architectures for Skein Implementation

In this section, we first briefly review the algorithmic aspect of Skein hash function. Next, we describe several architectures for the Threefish cipher in the skein hash implementation. Then, we present a detailed description on what have been implemented in our FPGA platform. We conclude this section by discussing certain optimizations for the arithmetic in a round operation of the Threefish cipher.

3.1 *Skein Algorithm*

This section identifies the high-level overview of the Skein design needed to describe the implementation. For a more detailed description, we refer readers to [2]. From an implementation perspective, Skein can be decomposed into the Unique Block Iteration (UBI) construction and the Threefish Block Cipher, as shown in Fig. 4.

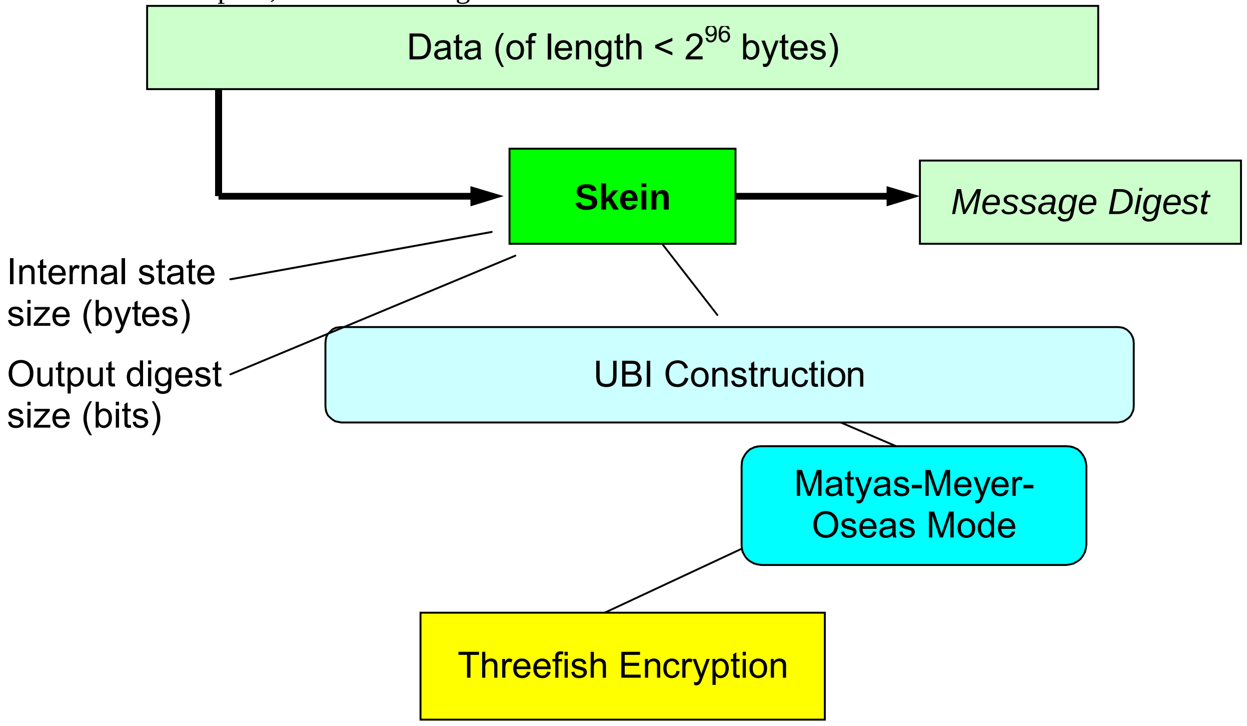

Data (of length < 2^{96} bytes)
Skein
Message Digest
Internal state size (bytes)
Output digest size (bits)
UBI Construction
Matyas-Meyer-Oseas Mode
Threefish Encryption

Fig. 4: the algorithmic decomposition of skein hash operation.

Threefish is a wide-block tweakable block cipher with a standard iterative structure. Three variants of Threefish, using three different block sizes, are defined: Threefish-256, Threefish-512, and Threefish-1024 (the suffix gives the block size in bits). Note that the Threefish key size is the same as its block size. Threefish has the architecture of round operation: Threefish-256 and Threefish-512 consist of 72 round operations while the Threefish-1024 requires 80 rounds. Each Threefish round consists of three operations: mixing, permuting, and adding round key (optional). The mix operation consists of 64-bit adds, xors, and left-rotates. The permute operation reorders 64-bit words constructed from a Threefish block. The Threefish key schedule constructs round keys from the key and a 128 bit tweak. A round key is added to the data before the first round and after each 4 rounds thereafter. Fig. 5 depicts this round architecture for the Threefish cipher.

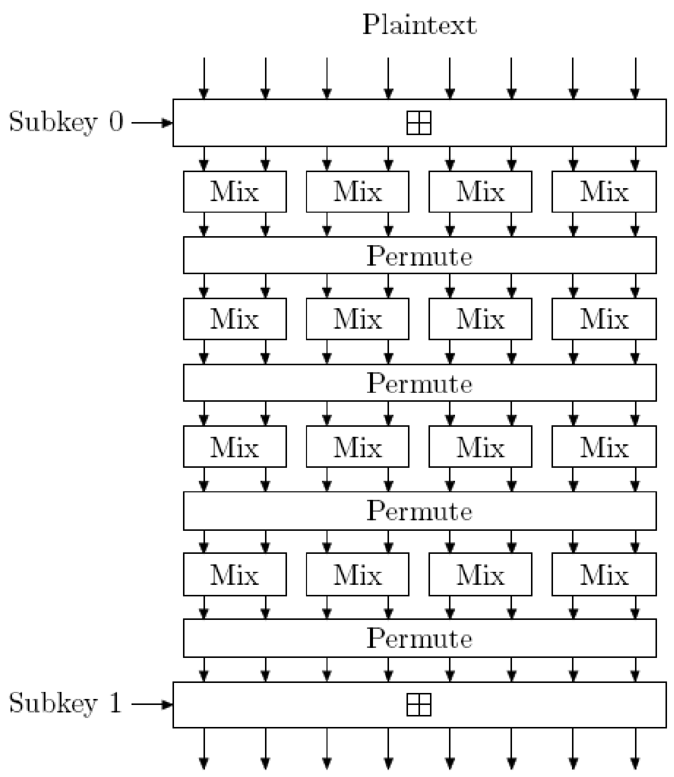


Fig. 5: Threefish round operation on mix and permute in the example of Skein-256. The diagram depicts the first 4-round operations of the Threefish cipher.

The UBI construction is a variant of the Cascade or (Merkle-Damgård) construction. It uses a tweakable block cipher in Matyas-Meyer-Oseas mode to form a compression function, and uses the bit offset of the block being hashed as the tweak. As an example, Fig. 6 depicts how UBI is used to hash a 166 byte message using the Skein-512 hash function.

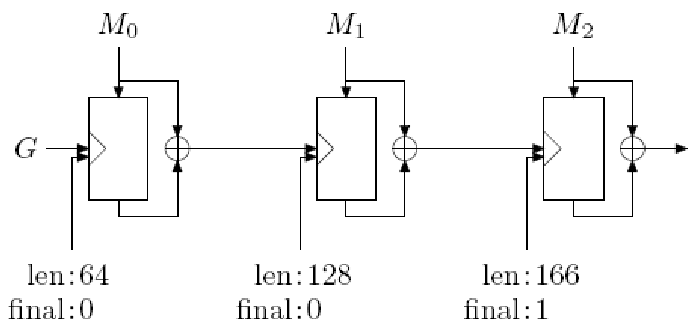


Fig. 6: the UBI (unique block iteration) construct in Skein-512 where a message of 166-byte is hashed. The length numbers 64, 128, and 166 are part of the tweaks used by the Threefish cipher. The flag (setting the final "1" for the last message block) can indicate the end of the hash processing.

3.2 *Design for the Threefish Round Implementation*

At its core the Skein hash function uses UBI to iterate the Threefish cipher. The design focus therefore on techniques to implement Threefish, which itself consists of 72 or 80 rounds. A silicon area/latency tradeoff is the metric used to evaluate the design. In this subsection, we outline several methods of implementing the round architecture for the Threefish.. It is worth pointing out that, in this subsection, we treat the arithmetic in a Threefish round operation as a black-box.

3.2.1 Architecture 1: Iterative Round Architecture

In this design, there is a 1:1 mapping between the round circuit in hardware and the round operation defined in Skein hash. In other words, one round described by the Skein algorithm is implemented in the round circuit. Fig. 7 depicts the design for the architecture. For instance, Skein-512 requires 72 rounds of the Threefish cipher. Therefore, there are 72 times of iteration for the hardware round circuits.

In addition to the datapath, there is a key scheduling block, which takes the key, tweak, and round number to produce a round key. The input is fed through a multiplexer into a round block implementing a Threefish round. The round block stores the result in a register. The register is either fed back through the multiplexer into the round itself, or, upon completion of a Threefish operation, Xored with the input to implement the Matyas-Meyer-Oseas feed-forward.

The implementation uses an on-the-fly key scheduling circuit to avoid precomputing round keys. The "rotates" used by the Threefish MIX function and round permutations are implemented through routing. Assume one round of the circuit is evaluated in exact one clock cycle. Therefore, if the algorithm uses *n* rounds, this design requires *n* clock cycles to perform one Threefish encryption over one block input. The output of the Threefish cipher then is Xorred with the input block to obtain the output of the compression function. These steps are repeated for each block of the message being hashed. For each input block, there is a new key—the chaining variable, which is the compression function output from the prior block—and tweak—the offset of the new block from the start of the message.

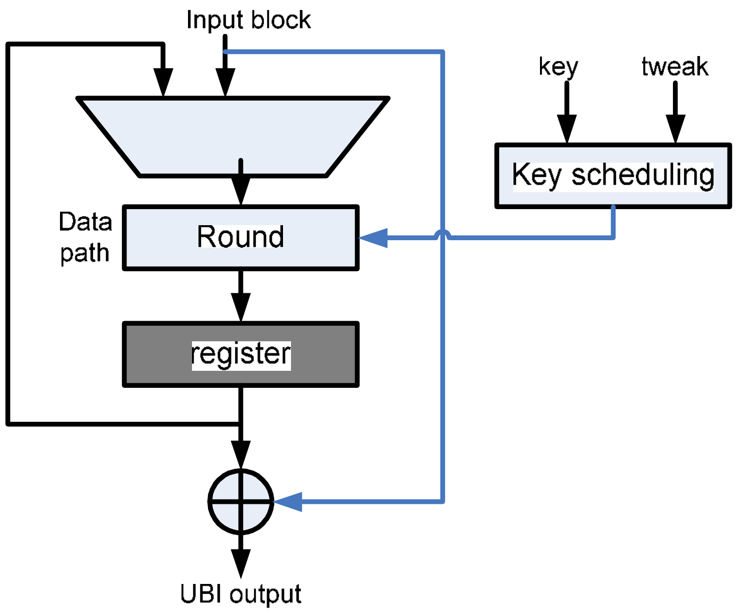


Fig. 7. Iterative round architecture in hardware for UBI in Skein. We abstract the Threefish operation in the "Round" box whose details will be described later in the text. For instance, in Skein-512 of 72-round Threefish cipher, there will be 72 iterations of the round circuit execution.

3.2.2 Architecture 2: Loop Unrolling Architecture

In contrast to the iterative round architecture, k copies of a round could be used before registering an output, where k is a divisor of the number of rounds [2]. All k rounds would then be implemented as a single combinational logic. Fig. 8 illustrates this design architecture. .

A rationale for the loop unrolling design is that the VLSI synthesis tools or the circuit mask engineers might do a better job of optimization the hardware resources. For the k round operation in iterative architecture (Section 3.2.1), the latency would be $T_{iter} = k$ cycles × clock period assuming 1 cycle for a round operation. In contrast, the latency T_{loop} of the loop unrolling architecture for k round operations will be less than T_{iter}. However, the empirical data [3] have shown that the area requirement usually increases more than the increase in throughput, resulting in lower efficiency of the loop unrolling architecture. In contrast to the folklore, it is worthy pointing out that our experimental results actually showed a comparable performance between the loop unrolling and round iteration architectures in our Skein-512 implementation.

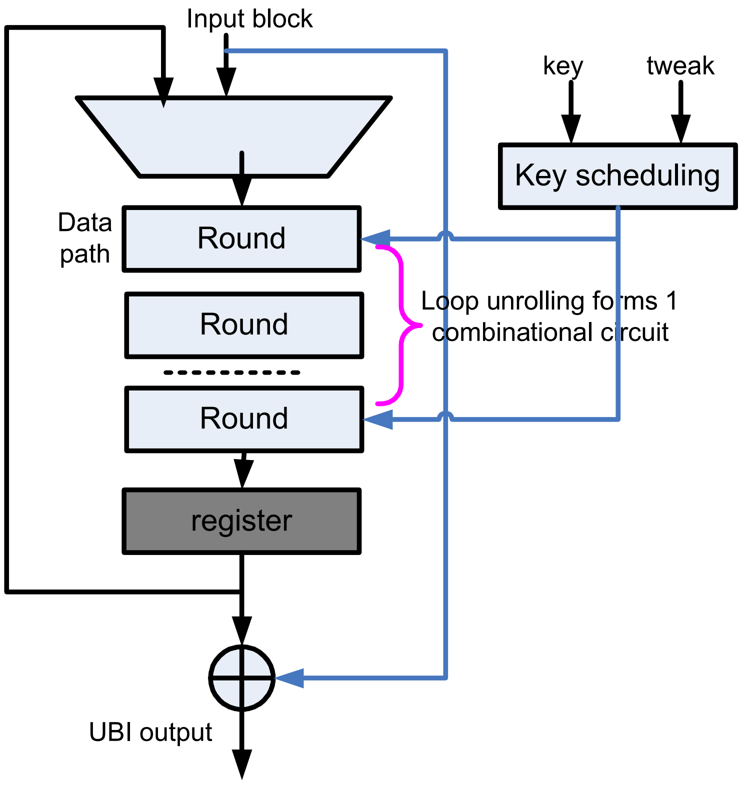


Fig. 8: Loop unrolling architecture for the UBI construct in the Skein hash function, where k copies of the round circuits are used between the input mux and the register. In the example of Skein-512 of 72-round Threefish cipher, assuming we have 4 copies of the round circuit, we incur only 18 iterations over the register output.

3.2.3 Architecture 3: Pipeline Architecture

In many cases, a pipeline architecture, such as depicted in Fig. 9, will improve throughput, as demonstrated by AES counter mode. The salient feature of pipeline design would be it inserts a register between consecutive rounds. However, a hashing algorithm such as Skein enforces data serialization. This means that hashing block n of a message cannot begin until the hashed result of the block n–1 becomes available. Consequently, most round circuits in the pipeline engine would be idle in such a design, which inefficiently utilizes hardware resources. It is worth pointing out that Skein enables the tree hashing schemes, where the blocks for the input message can be fed into the circuit in a pipeline manner. Nevertheless, this usage model of hash tree is not expected to be the primary application of a hash function submitted to the NIST competition.

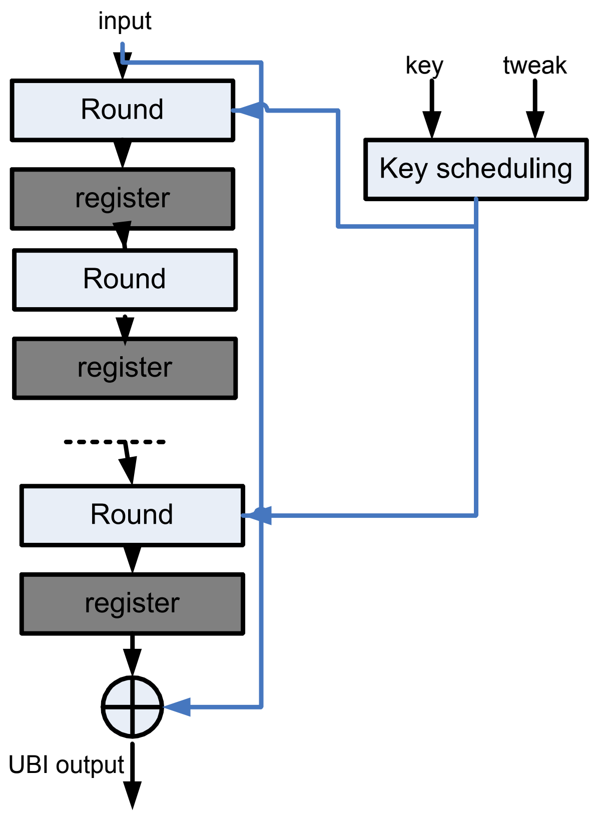


Fig. 9: pipeline architecture might be employed for the case of tree hashing in the Skein. Registers are inserted between the rounds. For normal hash operation, pipeline architecture is seldom used as most circuits are idle in waiting for the output.

3.3 *Baseline Implementation of Arithmetic in Threefish Round*

In this section, we detail the arithmetic operations inside a round of Threefish cipher. For simplicity of presentation, we describe the Skein-256 implementation as an example; implementation for Skein-512 or Skein-1024 is similar.

Fig. 10 shows the major components of the round operation. We use “add64” to denote the addition on two 64-bit numbers with carry. The shift and permute operation can be done through the wiring placement, and we denote it as “wireshift”. As the round key generation will be different according to the individual round, we use the blackbox “MuxSwitch” to reflect the nature of the wiring of the round key scheduling. As shown in Fig. 10, it is a design to maximize the throughput, because all the input of a block will be fed into the adders at the same time. Fig. 11 shows the preprocessing circuit for the key scheduling, and the output of the Fig. 11 circuit will be used to compute the round keys for the Threefish operation.

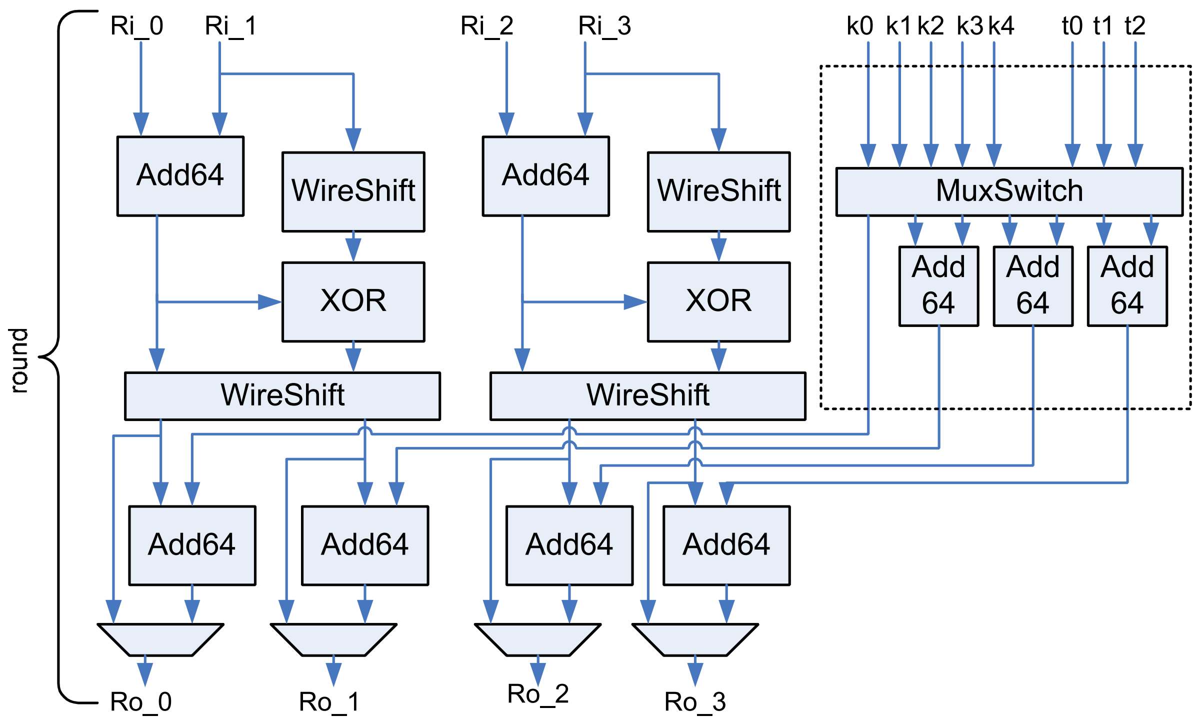


Fig. 10: Baseline for the Threefish round operation in Skein. Ri_0, Ri_1, …, Ri_3 are the 64-bit inputs for the round while Ro_0, Ro_1, …, Ro_3 are the round outputs. k0, k1, …, k4 are the preprocessed keys, and t0, t1, t2 are the preprocessed tweak. Note that the round circuit in hardware directly maps to the round concept in Skein algorithm. The multiplexers at the output handle the situation of adding the round key at every 4 rounds.

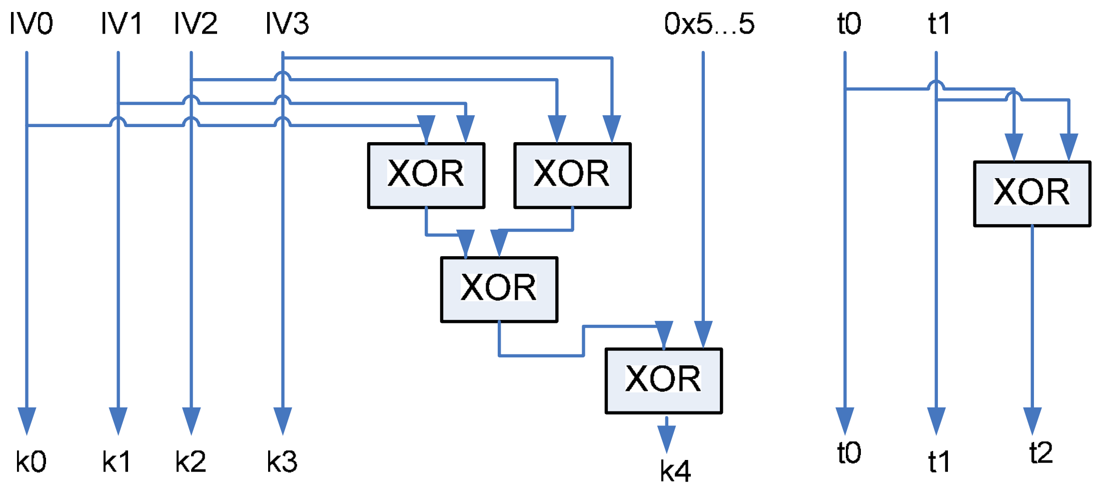


Fig. 11: The key preprocessing circuit. The IVs, t0, and t1 are the seed for the key scheduling, specified by message input. The preprocessing circuit yields k0,…, k4, and t0, …, t2. Then each round key will be generated based on k0,…, k4, and t0, …, t2 (as shown in Fig. 10).

In summery, the design shown in Figs 10-11 assumes the maximal parallelism within a round of Threefish operation. The critical path is the two 64-bit adders, one XOR gate, and some wire delay. The major component for the area for Skein-256 is thus the nine 64-bit adders. By the same token, it will be fifteen 64-bit adders for one round of Threefish in Skein-512 and twenty seven 64-bit adders for one Threefish round in Skein-1024.

3.4 *Optimization for Arithmetic Implementation in Threefish Round*

We describe several optimization techniques to the arithmetic implementation within a Threefish round. One flavor is to share the resource (e.g. adders) to reduce the area cost, and the other is to reduce the level of logics within one round circuit to increase the clock frequency. The FPGA implementation for these techniques is a work in progress.

3.4.1 Sharing adders for the key scheduling circuit

Skein adds a round key every 4 rounds. Because of this, we do not need the 3 parallel adders shown in the dashed box of Fig. 10. For instance, if each adder requires 1 cycle, the key scheduling circuit only needs 1 adder, which can compute all the round keys during the 4-cycle budget. As a result, we could save 2 adders for the key scheduling circuit without any penalty on the latency or critical path. The schematic of sharing adders in the key scheduling circuit is shown in Fig. 12.

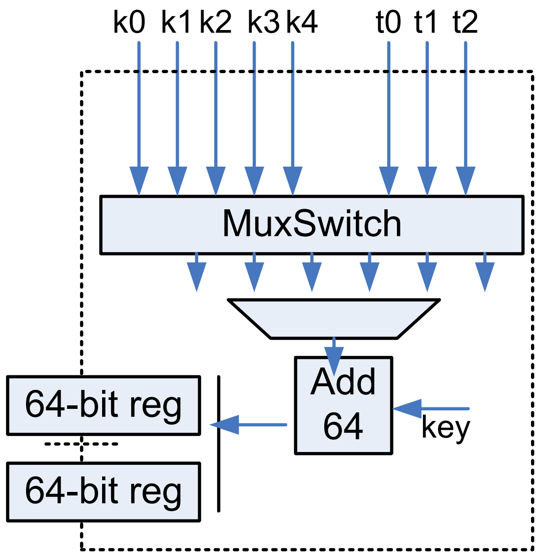


Fig. 12: Using one adder in key scheduling circuit. It can be iterated in 4 cycles to yield all the 64-bit round keys that are needed for the addition operation.

3.4.2 Sharing the output adders for the datapath

Since adding the round key occurs at every 4 rounds, output adders are not used by every round. As a result, Fig. 10 shows a multiplexer on output to account for certain rounds that do not need the round key addition.

As an optimization, we could reduce the number of the adders in the output datapath. For instance, in a Skein-256 implementation, there are 4 adders in the output stage. Fig. 13 shows that we could use only two adders instead. So, for the round with the key addition, in the first cycle we get one 128-bit output and in the second cycle the other 128-bit output. Given that only 18 of 72 rounds require the key addition, those 18 rounds incur additional latency. This gracefully increased latency is offset by an area savings of two 64-bit adders.

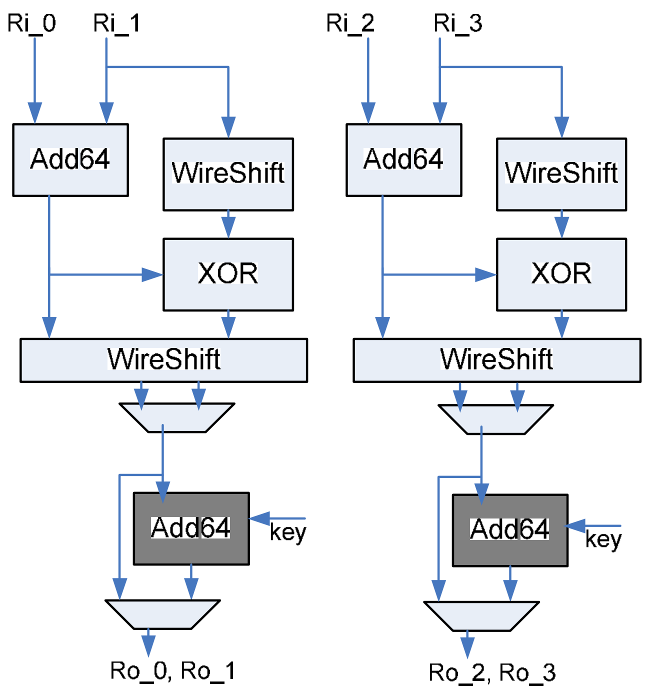
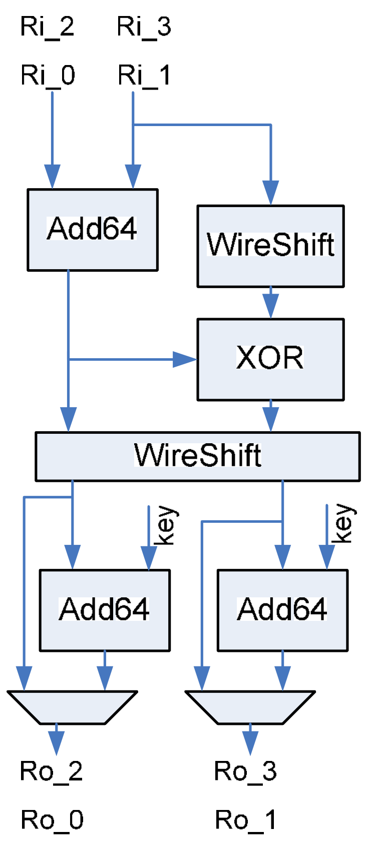


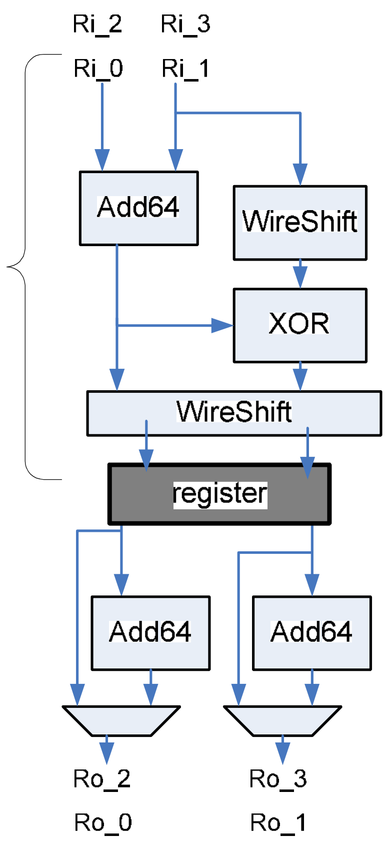
Fig. 13: Using one adder in the output for key addition. For the key-addition round, it will iterate through the adder (shaded box) in two cycles to produce the round output.

3.4.3 Sharing the input adders for the datapath

This optimization is similar to the one of sharing the output adders. There are two 64-bit adders for the input mixing. We could use only one adder in the implementation and iterate in two cycles to perform a full Threefish MIX operation, as depicted in Fig. 14(a). In contrast to the technique of sharing the output adders, the approach will incur the latency penalty for every round, which would limit the benefit of the optimization. A further optimization is shown in Fig. 14(b), where we insert a register within the round. In essence, this splits the round operation into two halves. The subsequent advantage is to reduce the logic levels of critical path by 50%, which could double the clock frequency in the final circuit.



(a)



(b)

Fig. 14: (a) sharing the input adder for the 64-bit pairs of input. Assume an example of Skein-256. In cycle *n*, Ri_0 and Ri_1 are fed into the circuit, In the next cycle, Ri_2 and Ri_3 are fed into the circuit. (b) Split the internal round operation by inserting a register. This will halve the logic levels of the critical path, and essentially double the clock frequency.

3.4.4 Overlap MIX and key addition

This approach borrows the aforementioned idea of inserting one more level of register within a round operation to reduce the critical path and therefore increase the clock frequency. Fig. 15 shows the optimization. With some additional control circuit, we could overlap the circuit between the MIX and key addition. In the example of Skein-256, for rounds 1-3, only one adder is activated for one 64-bit pair of the input block—it takes one cycle to execute each round. Round 4 will take 2 cycles, and both adders will be activated in the cycle No. 2.

In this optimization, the work of 2 rounds will be equivalent to the one-round work shown in Fig. 10. In a concrete example of Skein-256, the baseline design of Fig. 10 incurs 72 round with a round latency denoted as T_1 for one Threefish operation. In contrast, the optimization will cost 90 rounds (64+18×2) with a round latency denoted as T_2. The advantage of splitting the round operation is that T_2 might be about half of T_1. Therefore, in terms of latency at the level of Threefish (and hence UBI and Skein), the optimized design may shorten the total latency by about 37%. In addition, this approach reduces the area by about 30% (only four adders are used compared to the six in the baseline design in Fig. 10).

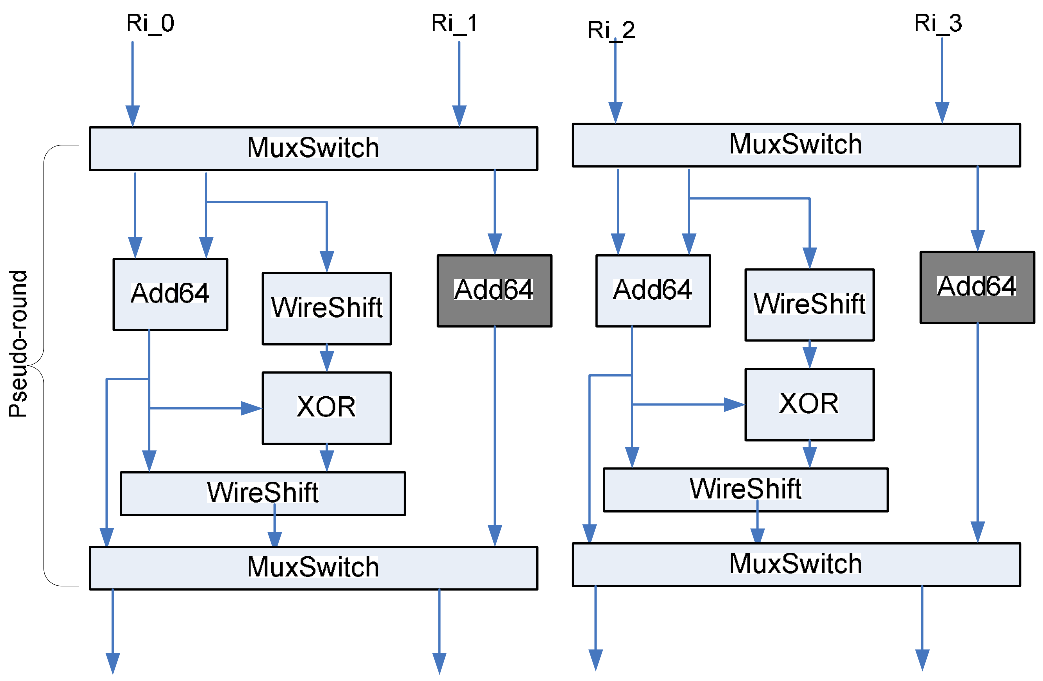


Fig. 15: Overlap circuit for MIX and key addition. The pseudo-round will be equivalent to the non-key-addition round in Skein algorithm. For the round of key addition, the circuit will iterate in two cycles to do the MIX and key addition. Benefits of this implementation include the reduction in the critical path latency and the silicon area.

4 Experimental Results

4.1 *Experiment Setup*

We use the Xilinx virtex-5 LX50T device with the speed of "-3" (550 MHz). The synthesis and place-route tool is the Xilinx ISE 9.1.03i with the default compiler options. We use the test vectors submitted by the authors of Skein hash as the reference to validate the correctness of the implementation.

4.2 *Quantitative Results*

We implemented the UBI component for Skein-256, 512, and 1024 based on the architecture defined in Fig. 7. The arithmetic within a round is implemented based on Figs. 10 and 11. Table 1 shows the results reported by the Xilinx ISE tool after the place-and-route.

Table 1: Performance of UBI one round operation

FPGA performance	UBI (Skein 256)	UBI (Skein 512)	UBI (Skein 1024)
# of flip-flops (FF)	330	586	1106
# of logic LUT	3723	7029	14237
# of LUT-FF pairs	4003	7508	14623
Minimum clock period (ns)	8.7	8.7	11.79

Most of flip-flops are spent on datapath registers and key scheduling. We use the metric (# of logic LUT-FF pairs) as the area cost. Note that the LUT-FF pair could be regarded as superset of the number of logic LUT. In this context, the number of LUT-FF pairs is greater than the number of logic LUT because some of the extra LUTs are used for the route-thrus (or wire routing) purpose.

We extrapolate the Skein throughput from the measured results in Table 1. Under the definition of Skein, we exclude the first UBI operation on the configuration value as this is a static value that could be precomputed. In addition, we exclude the last UBI operation on the output. Processing a 256-bit message in Skein 256 requires the latency of $8.7 \times 72 = 626.4$ ns. Thus the throughput for Skein-256 is (256/626.4) $\times 10^9 = 408.7$ Mbps. Since it is deterministic in the hardware design, the results apply to larger messages as well. By the same token, we can compute the throughput for Skein 512 and 1024 as 817.4 Mbps and 1.1Gbps, respectively. Table 2 summarizes the throughput data for the Skein hash family. We also include the metric of throughput per LUT-FF pair, which measure the performance per FPGA silicon unit area. We believe this might server a good measure over the tradeoff between area and throughput of a specific implementation architecture.

Table 2: Extrapolated results in Skein throughput

FPGA performance	Skein 256	Skein 512	Skein 1024
throughput (Gbps)	0.409	0.817	1.1
throughput per LUT-FF pair (Mbps)	0.1	0.11	0.075

4.3 *Discussion*

The Xilinx documentation on Virtex-5 reports the 2.5 ns for a 64-bit adder [4]. In the design of Fig. 10, the critical path has 2 adders in serial. Thus the lower bound for the latency is 5 ns for one round of UBI. Our synthesis result reports 8.7 ns for one round of Skein-256 and Skein-512. Thus about 3 ns in the round latency are spent on the wire or routing delay. Based on this finding, we expect that the optimized approach of splitting round operation outlined in Section 3.4.4 might give a substantial improvement in overall latency reduction as the clock period can be decreased and the wire delay might be dropped. Our future work may provide the quantitative results to verify our guess.

We also implemented the loop unrolling architecture shown in Fig. 8, where we use 4 copies of the Threefish round before a register. The choice of 4 copies fits well with the skein design of adding the round key for every 4 rounds. In the iterative round case, the latency for 4 rounds will be $8.7 \times 4 = 34.8$ ns. However, when we did the loop-unrolling of 4 copies of round, the latency is 24.5 ns. This supports that the idea that the synthesis tool may do a better job in the loop-unrolling case to reduce the wire/routing delay. Furthermore, it is evident that the area in the 4-round loop unrolling is far smaller than the 4 times of area in the iterative round. When we use the metric of throughput per LUT-FF pair, the results by these two architectures are quite comparable.

Table 3: UBI in Skein-512 under two different round implementation architectures. In the loop unrolling architecture, 4 copies of a round circuit are used.

FPGA performance	Loop-unrolling (in Fig. 8)	Iterative round (in Fig. 7)
# of flip-flops (FF)	520	586
# of logic LUT	10188	7029
# of LUT-FF pairs	10648	7508
Minimum clock period (ns)	24.5	8.7
72-round latency (ns)	441	626.4
Skein throughput per LUT-FF pair (Mbps)	0.11	0.11

5 Conclusion

In this note, we implemented the Skein family hash functions (Skein-256, Skein-512, and Skein-1024)—in particular, the UBI with Threefish construct—on Xilinx Virtex-5 FPGA. The design guidance is to maximize the throughput, in which we tried to parallelize the arithmetic inside a round. In a higher-level architecture, the iterative round approach is the common method to implement the round-based cryptographic function, where it tries to balance the area cost with the throughput. The loop unrolling method collapses the algorithm of several rounds into the digital logic of one round, and aims to reduce the overall latency for a cryptographic operation. One interesting experimental finding is that those two architectures yield similar performance in terms of throughput per LUT-FF pair.

We also outline several optimized approaches to improve the implementation for the arithmetic inside a Threefish round. Based on our initial synthesis results on the wire delay, we expect that some of those optimization methods (e.g. splitting the round operation) might yield non-negligible gain in terms of throughput improvement.

6 References

[1] Xilinx, "Virtex-5 User Guide, UG190 (v3.0)", February 2, 2007

[2] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback, "Report on the development of the advanced encryption standard (AES)," October 2000

[3] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker, "The Skein hash function family," Oct 2008

[4] A. Cosoroaba and F. Rivoallon, "Achieving higher system performance with the virtex-5 family of FPGAs," Xilinx WP245 V1.1.1, July 2006

7 Acknowledgement

Jesse Walker provided explanations of Skein as well as suggestions for optimizating the hardware implementation. We also thank Doug Whiting and Niels Ferguson for their feedbacks on the Threefish round implementations