Hardware Implementation of the Salsa20 and Phelix Stream Ciphers

Junjie Yan and Howard M. Heys
Electrical and Computer Engineering
Memorial University of Newfoundland
Email: {junjie, howard}@engr.mun.ca

Abstract— **In this paper, we present an analysis of the digital hardware implementation of two stream ciphers proposed for the eSTREAM project: Salsa20 and Phelix. Both high speed and compact designs are examined, targeted to both field programmable (FPGA) and application specific integrated circuit (ASIC) technologies.**

The studied designs are specified using the VHDL hardware description language, and synthesized by using Synopsys CAD tools. The throughput of the compact ASIC design for Phelix is 260 Mbps targeted for 0.18μ CMOS technology and the corresponding area is equivalent to about 12,400 2-input NAND gates. The throughput of Salsa20 ranges from 38 Mbps for the compact FPGA design, implemented using 194 CLB slices to 4.8 Gbps for the high speed ASIC design, implemented with an area equivalent to about 470,000 2-input NAND gates.

1. INTRODUCTION

The eSTREAM project [1] is a multi-year effort aimed at finding new stream ciphers that are suitable for widespread adoption. More than 30 new stream cipher primitives have been proposed. The project is composed of two phases to evaluate the performance of software and hardware implementations respectively.

At the end of phase one, the software performances have been analyzed comprehensively. However, the hardware implementation results are far from sufficient. Compared with software implementations, hardware implementations are less flexible but can typically be more compact and fast. As well, encryption hardware cores are usually isolated from other hardware in a system, so hardware implementation for a cipher is often more secure.

The research of this paper examines designs of Salsa20 [2] and Phelix [3], both of which are claimed suitable for software and hardware implementation. The needs of different applications in communication systems demand different structures for cryptographic algorithms in hardware. For example, in wireless applications like cell phones, compactness and low power consumption are critical because of battery limitations and portability, while for virtual private network (VPN) applications and secure e-commerce web servers, demand for high-speed encryption is rapidly increasing.

When considering implementation technologies, normally, the ASIC approach provides better performance in density and throughput, but an FPGA is reconfigurable and more flexible. In our study, several schemes are used, catering to the features of the target technology.

2. PHELIX

2.1 Short Description of the Phelix Algorithm

Phelix is claimed to be a high-speed stream cipher. It is selected for both software and hardware performance evaluation by the eSTREAM project. The cipher supports an 8-bit to 256-bit length key and a 128-bit nonce to generate the keystream bits with a built-in Message Authentication Code (MAC).

Phelix is targeted at 32-bit platforms. It is composed of a serial of simple operations: addition modulo 2^{32}, exclusive or, and rotation by a fixed number of bits. There are 5 words that are updated during each round, and 4 "old" words are stored in memory to be used in the keystream output function.

One block that produces one word of keystream consists of two "half-block" functions H, which is defined as:

Function H (w_0,w_1,w_2,w_3,w_4, K_0,K_1)
Begin

$w_0 := w_0 \boxplus (w_3 \oplus K_0)$;	$w_3 := w_3 <<< 15$;
$w_1 := w_1 \boxplus w_4$;	$w_4 := w_4 <<< 25$;
$w_2 := w_2 \oplus w_0$;	$w_0 := w_0 <<< 9$;
$w_3 := w_3 \oplus w_1$;	$w_1 := w_1 <<< 10$;
$w_4 := w_4 \boxplus w_2$;	$w_2 := w_2 <<< 17$;
$w_0 := w_0 \oplus (w_3 \boxplus K_1)$;	$w_3 := w_3 <<< 30$;
$w_1 := w_1 \oplus w_4$;	$w_4 := w_4 <<< 13$;
$w_2 := w_2 \boxplus w_0$;	$w_0 := w_0 <<< 20$;
$w_3 := w_3 \boxplus w_1$;	$w_1 := w_1 <<< 11$;
$w_4 := w_4 \oplus w_2$;	$w_2 := w_2 <<< 5$;

Return (w_0,w_1,w_2,w_3,w_4);

0840-7789/07/$25.00 ©2007 IEEE

End.

The bitwise exclusive-or of two words, denoted as "⊕", is the sum of the words with carries suppressed. The symbol "<<<" represents left rotation, and "⊞" represents addition modulo 2^{32}. Further details of the algorithm can be found in [3].

2.2 Compact ASIC Structure of Phelix

The Phelix stream cipher can be implemented in many ways. The proposed compact structure focuses on function sharing to optimize the area. Figure 1 illustrates a minimal ASIC implementation consisting of one round of encryption and a memory recording the four old states.

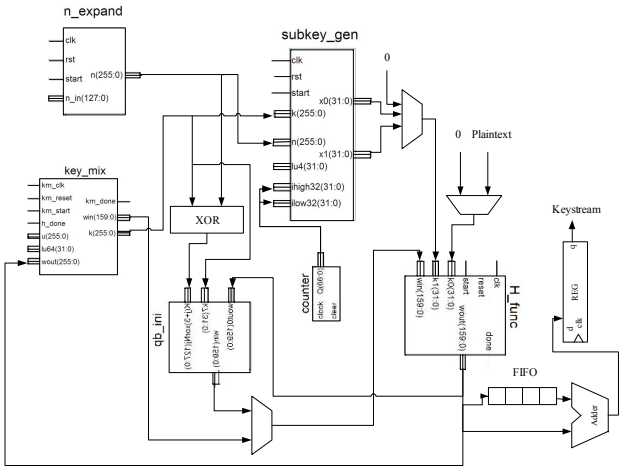

Figure 1. Phelix Compact Structure

The specifications of the main blocks are given below:

- n_expand: converts a variable-length input nonce to the fixed-length working nonce.
- key_mix: converts a variable-length input key to the fixed-length working key.
- subkey_gen: generates subkeys $X_{i,0}$, $X_{i,1}$ for each block.
- ini_dp: decides the input of H_func. For the first eight blocks (initialization phase), the generated keystream is discarded.
- H_func: performs function H (w_0,w_1,w_2,w_3,w_4, K_0,K_1).
- FIFO: the "first in, first out" memory that stores the old states.

To improve the performance, the subkey generation is on the fly: during each block, two words of subkeys are produced and used as parameters in H function block that generates the 32-bit key stream later.

To increase the speed of the encryption, we could design additional logic to perform the H function. It would require more adders and 32-bit exclusive-or function blocks that can work in parallel. However, it will dramatically increase the size of the H function circuit since the adder is the largest component compared with other simple function blocks, such as a 32-bit register. So far, we have not investigated the exact area penalty inferred by this option.

Also, many multiplexers can be removed to shorten the critical path. But the gained benefit is limited. Implemented by using 0.18μ CMOS technology, the multiplexers have a 0.37～0.70 ns range of critical path, depending on the number of the input ports, while the overall critical path that contains only two multiplexers is around 7 ns. Only a 10% speed increase can be gained by this method.

Furthermore, it is possible to compute the subkeys off-chip, and then download them into the circuit memory to save time. However, this may affect the security of the device.

3. SALSA20

3.1. Short Description of the Salsa20 Algorithm

As one of the stream cipher candidates of the eSTREAM project, Salsa20 is claimed to provide high security, and is composed of several simple operations that are similar to Phelix. The core of Salsa20 is a hash function, encrypting a 512-bit block of plaintext by hashing the key (128-bit), nonce (64-bit), and a sequence number (64-bit) to a 512-bit output.

The main function in the Salsa20 core is called quarterround function. Define y as a 4-word sequence then quarterround(y) is a 4-word sequence.

If $y = (y_0;y_1;y_2;y_3)$, then quarterround(y) = $(z_0;z_1;z_2;z_3)$, where y_i and z_i are 32-bit words, $i \in \{0,1,2,3\}$ and

$z_1 = y_1 \oplus ((y_0 \boxplus y_3) <<< 7)$

$z_2 = y_2 \oplus ((z_1 \boxplus y_0) <<< 9)$

$z_3 = y_3 \oplus ((z_2 \boxplus z_1) <<< 13)$

$z_0 = y_0 \oplus ((z_3 \boxplus z_2) <<< 18)$

If we consider the 64-byte input block as a 4×4 matrix of 32-bit words, the four elements in each row and each column will be modified by quarterround function ten times, respectively. After that, the output is added with the original values, producing a 4-word keystream.

3.2. Compact ASIC Structure of Salsa20

The compact structure of Salsa20 mainly contains two 32-bit ×16 memory blocks, a controller and one quarterround function block.

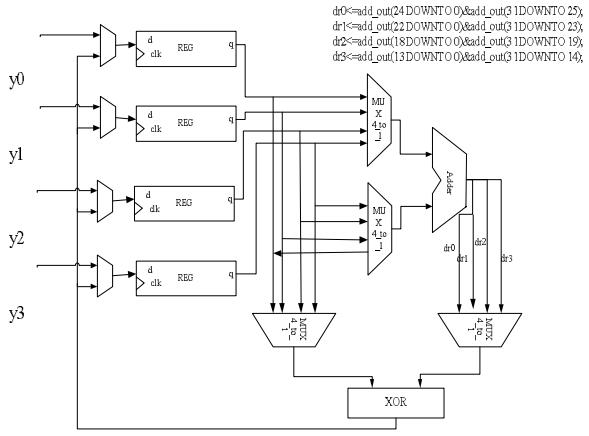

Figure 2. Datapath of Quarterround Block

The design of quarterround block is straightforward. If the *reset* signal in the controller occurs, the contents of all registers are formatted to zero. If the *start* signal occurs, the inputs are loaded into the registers in parallel, and then the core performs addition, rotation, and XOR sequentially. After that, the modified data is loaded into the registers again.

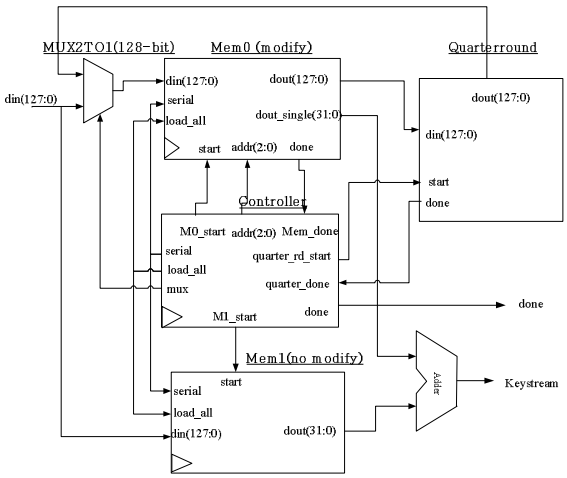

Figure 3. Salsa20 Compact ASIC Structure

The specifications of the main blocks are given below:

- Quarterround: performs quarterround function.
- Mem0(modify): stores the 16 original 32-bit words, and its contents will be modified after each quarterround function.
- Mem1(no modify): stores the 16 original 32-bit words, which will not be touched until the quarterround function block has been used for twenty times.

The critical paths of subcomponents are quite unbalanced. For example, the quarterround block is more than two times faster than the memory. Thus, we use a frequency divider in the circuit. In that way, the global frequency is 250 MHz, and the frequency for the quarterround block is 125 MHz.

3.3. Basic Iterative ASIC Structure of Salsa20

The datapath of an iterative structure consists of four quarterround function blocks, since the four rows or the four columns are encrypted independently. The control unit is simply a combination of a counter and a comparator.

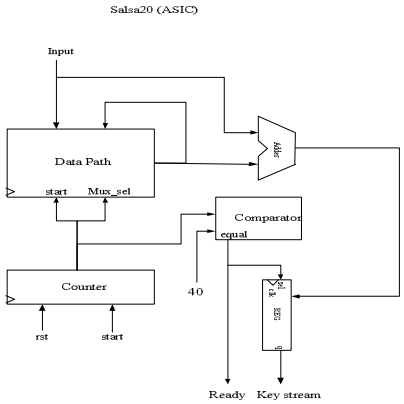

Figure 4. Salsa20 Basic Iterative ASIC Structure

After the start signal is asserted, the quarterround function begins to work on the 4×4 data matrix during the odd clock cycles. The data matrix performs a transpose function during every even clock cycle. It takes 40 clock cycles to finish the whole data encryption.

3.4. Fast ASIC Structure of Salsa20

There is minimal serialization of blocks in Salsa20. This feature can be considered in two ways: (1) during a single round, there is no communication between columns or rows; (2) each 64-byte block is encrypted independently. This gives a chance to implement a parallel structure, which can dramatically increase speed by handling several blocks at the same time.

Employing the iterative structure of Salsa20 as a pipeline stage, it is easy to build a pipelined structure of variable stages. A full pipelined structure consists of twenty stages arranged in a sequence. Adding pipelining affects latency, but compared with the overall improvement on performance, the latency is not dramatic.

3.5. Compact FPGA Structure of Salsa20

Usually, implementation on a Field Programmable Gate Arrays (FPGA) is reconfigurable and more flexible due to its specific properties. For example, memory is often a significant expense in most applications. Today's advanced FPGAs provide rich on-chip memories, which are maturely designed for compactness and speed. If properly employed, it can lead to a significant improvement in the latency of the overall design.

The proposed FPGA structure of Salsa20 takes advantage of the predefined memory and makes the use of a microprogramming controller.

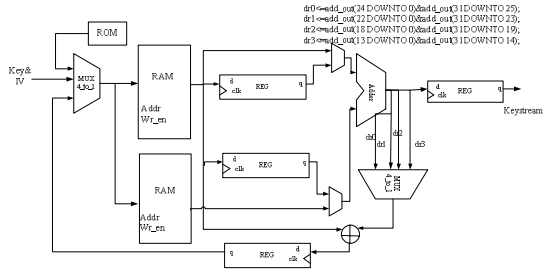

Figure 5. Compact FPGA Structure of Salsa20

Since the FPGA structure employs two 32×16 bits predefined memory and reuses the adder for compactness, the number of states required to produce different control signals for the datapath is largely increased. In a traditional controller consisting of a Finite State Machine (FSM) and combinational output logic, a large number of states can dramatically impact the logic equations, number of gates, and clock rate. To avoid such problems, the proposed design uses a microprogramming controller [4]. The control information is loaded into the memory in the initialization phase, and the controller sends out a microinstruction to the datapath every clock cycle. It is a flexible method, especially for further improvement, because program changes only cause slight difference in the memory, which contains control information.

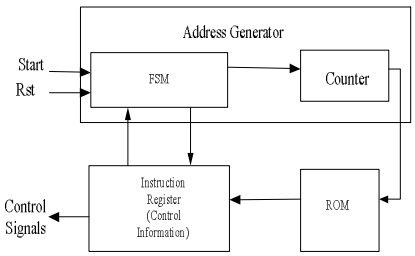

Figure 6. Controller of Salsa20

4. SYNTHESIS RESULTS

All designs are simulated and synthesized by using Synopsys CAD tools. Synthesis results for Phelix are illustrated in Table 1, and table 2 is for different structures of Salsa20.

Table 1. ASIC Implementation Results of Phelix

ASIC Device	Throughput (Mbps)	# of 2-input Nand gates
0.18 μ CMOS	260.0	12,400

To our knowledge, there are no published ASIC implementations results for the Phelix, but a rough estimation from the authors of [3], is that the cipher can achieve speed of at least 200 MBps with 20,000 gates for the area. The targeted technology is not specified. We are not aware of any published results on the hardware design of Salsa20.

Table 2. Implementation Results of Salsa20

Structure	Compact (ASIC)	Basic Iterative (ASIC)	Fast (ASIC)	Compact (FPGA)
Device	0.18μ CMOS	0.18μ CMOS	0.18μ CMOS	Xilinx FPGA 2V250fg256 [5]
Throughput	71.2 Mbps	255 Mbps	4.8 Gbps	38 Mbps
Area	14,100 2-input Nand gates	23,408 2-input Nand gates	470,000 2-input Nand gates	194 CLB slices + 4 Block RAMs

5. CONCLUSIONS

In this paper, two stream cipher candidates, Phelix and Salsa20 of eSTREAM project are implemented in hardware and compared in terms of performance and consumed area.

It is an unsurprising result that Salsa20 based on a compact context is slower and consumes more area than Phelix, since it performs a large number of invertible modifications, each of which changes one word of the matrix in a sequential manner.

A microprogrammed control unit and a predefined memory were presented in a compact FPGA structure of Salsa20 to save the area and improve the performance. It should be notice that high speed adders often consume more area in the chip. A more considerate choice of adders should be made with the requirements of the specific application.

REFERENCE

[1] Homepage for the eSTREAM project: www.ecrypt.eu.org/stream.

[2] D. Bernstein, "The Salsa20 Stream Cipher", presented at Symmetric Key Encryption Workshop, Aarhus, Denmark, May, 2005. Also available at www.ecrypt.eu.org/stream/salsa20.html.

[3] D. Whiting, B. Schneier, and S. Lucks , "Phelix - Fast Encryption and Authentication in a Single Cryptographic Primitive", presented at Symmetric Key Encryption Workshop, Aarhus, Denmark, May, 2005. Also available at www.ecrypt.eu.org/stream/phelixp2.html.

[4] B.W. Bomar, “Implementation of Microprogrammed Control in FPGAs”, IEEE Transactions on Industrial Electronics, vol. 49, pp. 415-422, April, 2002.

[5] Xilinx Inc., San Jose, Calif., “Virtex, 2.5 V Field Programmable Gate Arrays,” 2003, www.xilinx.com.