A PIPELINED IMPLEMENTATION OF THE GRØSTL HASH ALGORITHM AND THE ADVANCED ENCRYPTION STANDARD

Kai Guo and Howard M. Heys
Electrical and Computer Engineering
Faculty of Engineering and Applied Science
Memorial University of Newfoundland

ABSTRACT

Grøstl is a recently proposed cryptographic hash algorithm that has common structure and features with the Advanced Encryption Standard (AES). The objective of this paper is to present the design of a high speed joint implementation of Grøstl and AES with minimal resources using a pipelining method. The advantage of this implementation is that it efficiently provides both cryptographic hash function and block cipher. The system is targeted to the Altera Cyclone IV FPGA. The paper presents a complete description of the design and implementation, as well as an analysis of the resulting synthesis and comparison to other proposed implementations of the Grøstl hash function.

Index Terms — cryptographic hash function, Grøstl, Advanced Encryption Standard (AES), FPGA, SHA-3

1. INTRODUCTION

The objective of this work was the design a high speed joint implementation of Grøstl [1] and AES [2] with minimal resources using a pipelining method. The system is targeted to the Altera Cyclone IV FPGA using the DE2-115 development board [3]. The motivation for the design is that Grøstl is AES-inspired and has common structure with AES. Hence, an implementation combining Grøstl and AES by sharing logic elements will not significantly increase the expense of area and delay. The efficiency of sharing hardware resources determines the performance of such an implementation and the analysis of the resulting system is presented.

2. BACKGROUND

2.1. Cryptographic Hash Function

A cryptographic hash function is an algorithm that processes an arbitrary block of message to a fixed–size hash code without a key [4]. A hash function has three major properties. First, it is hard to derive the message from a given hash code. Second, any change in the data will change the hash code. Third, it is hard to find two different messages that produce the same hash code.

Cryptographic hash functions are widely used in a variety of applications. For security applications, hash functions are used for authentication processes, such as digital signatures and message authentication codes (MACs). Cryptographic hash functions can also be used as ordinary hash functions to detect accidental data corruption and duplicate data.

In 2007, the National Institute of Standards and Technology (NIST) announced an open competition for a new secure hash algorithm, SHA-3 [5], to replace the older SHA-1 and SHA-2 standards after 2012, due to the discovery of serious attacks, particularly against SHA-1.

2.2. Grøstl

Grøstl [1] was one of the five finalists of the SHA-3 competition. Grøstl consists of an iterated compression function f and an output transformation Ω. They are shown in Fig. 1.

In Fig. 1, the hash function, labeled H(m), produces an n bit hash code based on *IV*, the initialization vector, and m1, … , mt which are l-bit message blocks derived from input message M such that $l \geq 2n$. In this paper, we investigate a Grøstl-256 implementation for which l is 512 and n is 256.

The f function has two l-bit permutations called P and Q with slight differences. It is defined as:

$$f(h,m) = P(h \oplus m) \oplus Q(m) \oplus h \qquad (1)$$

The construction of f function is illustrated in Fig. 2.

The output transformation Ω produces the output of the hash function by truncating and only keeping the lowest n bits of $P(x) \oplus x$. Hence,

$$\Omega(x) = \mathrm{trunc}(P(x) \oplus x). \qquad (2)$$

The P and Q permutations are based on the four transformations in AES with a few differences. For this work, the state size is 512 bits, which is four times the size of the AES block. The state may be viewed as an 8×8 array of bytes. Each round of P and Q consists of four transformations: *AddConstant*, *SubBytes*, *ShiftRow*, and *MixColumns*. These operations are described as follows.

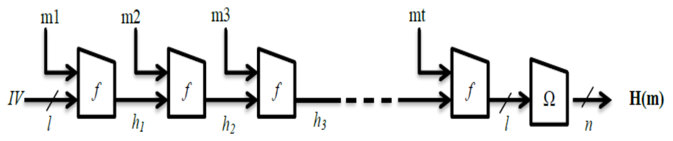

Figure 1. Overall Structure of Grøstl

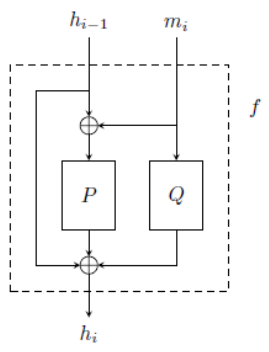

Figure 2. *f* function

(1) *AddConstant* outputs the result of exclusive-or of the input and a constant $C[i]$, which is a constant word for round i. The constants for P and Q are given in [1].

(2) *SubBytes* is identical to *SubBytes* of AES. The *SubBytes* of AES will be described in the next section.

(3) *ShiftRows* is a cyclic shift operation applied to each row by shifting a different number of bytes to the left. The shift pattern is shown in Fig. 3.

(4) *MixColumns* is a matrix multiplication applied to each column using arithmetic in $GF(2^8)$. The matrix multiplied to the columns is illustrated below:

$$\begin{bmatrix} 02 & 02 & 03 & 04 & 05 & 03 & 05 & 07 \\ 07 & 02 & 02 & 03 & 04 & 05 & 03 & 05 \\ 05 & 07 & 02 & 02 & 03 & 04 & 05 & 03 \\ 03 & 05 & 07 & 02 & 02 & 03 & 04 & 05 \\ 05 & 03 & 05 & 07 & 02 & 02 & 03 & 04 \\ 04 & 05 & 03 & 05 & 07 & 02 & 02 & 03 \\ 03 & 04 & 05 & 03 & 05 & 07 & 02 & 02 \\ 02 & 03 & 04 & 05 & 03 & 05 & 07 & 02 \end{bmatrix}$$

2.3. Advanced Encryption Standard

The Rijndael cipher was named as AES in 2001 by NIST for use by the U.S. government [2]. It has subsequently become widespread in many application environments. AES is well known as a secure block cipher based on the concepts of diffusion and confusion presented by Shannon [6]. It has a 128 bit block size and operates on a state viewed as a 4×4 array of bytes. The AES process for a 128-bit key has 10 of rounds iteration and each round has four transformations to construct diffusion or confusion. They are *AddRoundKey*, *SubBytes*, *ShiftRows* and *MixColumns* and are described as follows:

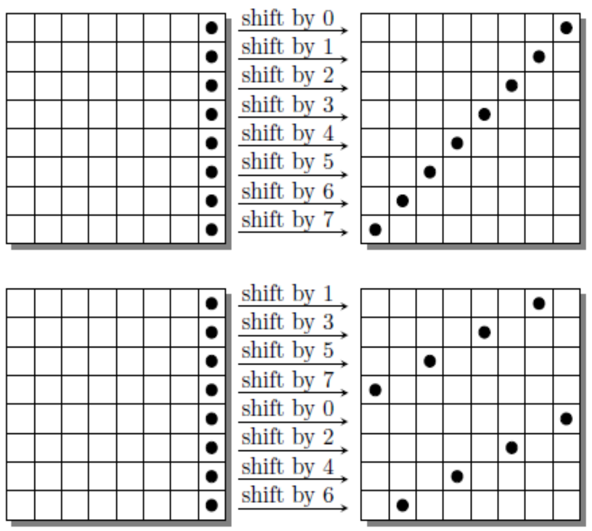

Figure 3. Shift Pattern for P (top) and Q (bottom)

(1) *AddRoundKey* performs the exclusive-or operation on a bit-by-bit basis between the round key and the 128-bit cipher state.

(2) *SubBytes* is a nonlinear substitution using a look-up table (S-Box) which is a mapping of one byte to one byte. Let the input and output byte of S-Box function be a and s, respectively. The mapping is defined by two substeps:

i. Inverse: $c = a^{-1}$, the multiplicative inverse in $GF(2^8)$ (except when $a = 0$, $c = 0$).

ii. Affine Transformation: The output byte is $s = Ac \oplus b$ where A is a constant matrix and b is a constant byte shown in (3) below.

$$\begin{bmatrix} s7 \\ s6 \\ s5 \\ s4 \\ s3 \\ s2 \\ s1 \\ s0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c7 \\ c6 \\ c5 \\ c4 \\ c3 \\ c2 \\ c1 \\ c0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad (3)$$

(3) *ShiftRows* is cyclic shift operation applied to each row by shifting a different number of bytes to the left. To be specific, shifts of 0, 1, 2 and 3 bytes to the left are applied on rows 0, 1, 2 and 3, respectively.

(4) *MixColumns* is a linear transformation function applied to each column, that can be viewed as a matrix multiplication to each column. The operation for each column $[S_{0,i}, S_{1,i}, S_{2,i}, S_{3,i}]$ is

$$\begin{bmatrix} S_{0,i} \\ S_{1,i} \\ S_{2,i} \\ S_{3,i} \end{bmatrix} \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} S'_{0,i} \\ S'_{1,i} \\ S'_{2,i} \\ S'_{3,i} \end{bmatrix} \tag{4}$$

There are various implementation methods of the S-Box. For this research, the composite field method [7] is used to minimize hardware resources. Specifically, the composite field implementation uses subfield arithmetic to find the multiplicative inverse in GF(2^8). Because it is costly in hardware to directly find the inverse in GF(2^8), taking this approach the inverse can be reduced to finding the inverse in GF(2^4) and several multiplications in GF(2^4). As well, with this S-Box structure, it is possible to split the S-Box logic, which will be discussed in the section on pipeline design.

3. DESIGN CONSIDERATIONS

The goal of the research is to design a high speed joint implementation that can do both the Grøstl and AES processes efficiently with minimal hardware resources. The Grøstl P and Q transformations and AES encryption process are denoted as P, Q and E for the following discussion. There are two ways to achieve high speed: one is to process P, Q and E in parallel and the other is to use a pipeline structure to process P/Q/E simultaneously. These two choices are shown in Fig. 4 and Fig. 5.

In Fig. 4, P, Q and E blocks are full round hardware. In Fig. 5, the sub-round pipelining is applied instead of full-round, which means a full round hardware is split into three balanced pipeline stages. The reason to choose sub-round is that P, Q and E are all independent and can simply be done in parallel. Therefore, pipelining of multiple full-round hardware is not helpful. The comparison is as follows:

1. Parallel implementation of P/Q/E needs three full rounds of hardware and 10 clock cycles for 10 round process. Hence, the total time is $10T_{round}$, where T_{round} is the duration of clock cycle required for 1 full round.
2. Sub-round pipelining requires about one full round of hardware and 32 clock cycles to produce both the encrypted and hashed result. With proper balanced pipeline stages, the clock cycle can be around $T_{round}/3$. Hence, the total time is close to $10T_{round}$.

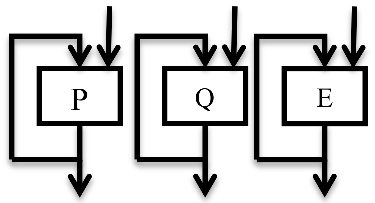

Figure 4. Parallel Implementation

Based on the above comparison, the pipelining structure is selected for this project because it requires far fewer resources than the parallel structure, but keeps almost the same throughput.

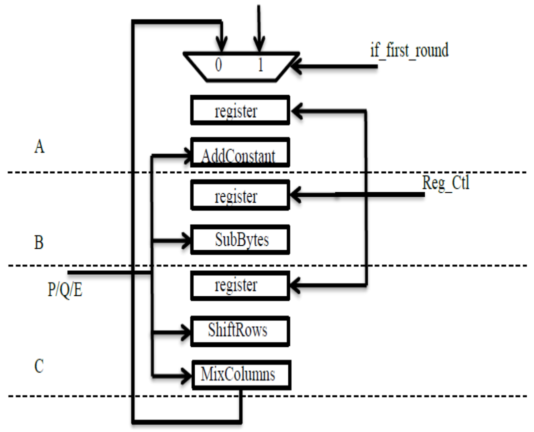

Figure 5. Pipeline Implementation

4. STRUCTURE OF THE SYSTEM

4.1. Datapath and Control Unit

The combined datapath design is based on the concept presented in [8]. Since Grøstl consists of four transformations similar to AES, the system uses joint components for each transformation that can process both Grøstl and AES. *AddConstant* and *AddRoundKey* share XOR gates. *SubBytes* is shared entirely since Grøstl uses the identical S-Box to AES. The composite field S-Box architecture is used to achieve minimal hardware resources. *ShiftRows* is constructed by swapping bytes. *MixColumns* shares the multipliers and XOR gates, but adds extra multipliers (×4, ×5, ×7) for Grøstl. The control unit provides proper control signals for each pipeline stage and transformation components. The overall design is shown in Fig. 6.

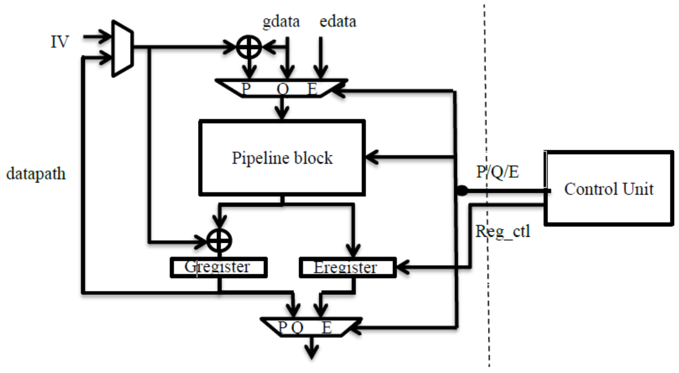

Figure 6. Datapath and Control Unit

In the Fig. 6, gdata and edata are the 512-bit inputs for Grøstl and AES, respectively. Components Gregister and Eregister are 512-bit registers. When P, Q and E are serially processed, Gregister will store the result of P and then replace with the result of Q and Eregister will store the result of E. Signals Reg_ctl and P/Q/E are control signals provided by the control unit. IV is the initial vector of Grøstl.

4.2. Pipeline Design

The pipeline structure as shown in Fig. 6 is achieved by splitting a full round hardware into three sub-rounds and adding pipeline registers between them. The pipelining block will process P/Q/E serially, which needs 32 clock cycles in total.

Under the consideration of distributing the pipeline registers more evenly, an optimized pipeline is shown in Fig. 7. Fig. 7 also shows the propagation delay of each pipeline stage from the result of timing analysis tool, TimeQuest from the Altera Quartus II development tools.

As shown in Fig. 7, the *SubBytes* transformation is split into two parts (SubBytes1 and SubBytes2) in the optimized version in order to balance the critical paths in the each pipeline stage. The first part, SubBytes1, is moved into StageA. This modification decreases the propagation delay of StageB from 5.816 ns to 4.127 ns. Since the slowest stage in the pipeline restricts the maximum frequency of the system, the optimized version can theoretically work at the frequency of 242 MHz. In comparison with 172 MHz of the non-optimized version, the improvement is significant.

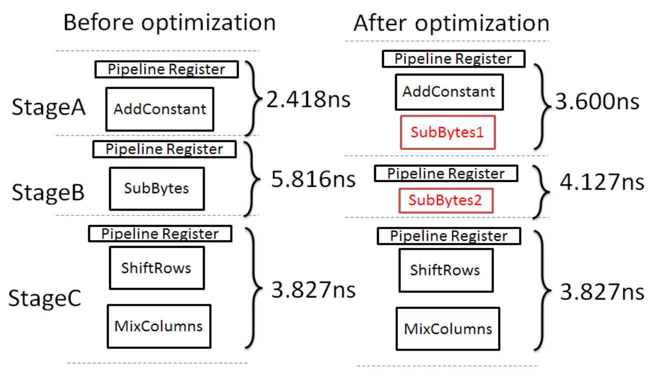

Figure 7. Pipeline optimization

5. SYNTHESIS RESULTS AND PERFORMANCE

In Table 1, the synthesis results of the proposed implementation and the results of others are presented for comparison using similar Altera FPGA technologies. Note that an extension of the design of [9] to include encryption would require an estimated 25,000 logic elements in total. In comparison to other systems, our implementation is capable of efficiently both hashing and encrypting a 512-bit block, using fewer logic elements than in [10], while producing higher throughout.

Table 1. Comparison of synthesis result
(TP = Throughput, #LE = number of logic elements, H+E = hash and encryption for each 512-bit block, H = hash only)

System	#LE	Max. Freq (MHz)	Clocks per block	TP (Mbps)
This work (Cyclone IV)	15,135	242.3	32	3877 (H+E)
[8] (Cyclone III)	13,723	56.0	30	956 (H)
[9] (Cyclone IV)	6209	149.6	21	3647 (H)
[10] (Cyclone III)	23,039	159.9	31	2640 (H+E)

6. CONCLUSION

The joint FPGA implementation of Grøstl and AES has enabled us to design an efficient high-speed system capable of simultaneously acting as a cryptographic hash function and block cipher. It can process one block of Grøstl and four parallel blocks of AES simultaneously in 32 clock cycles and is more efficient than other implementations in similar Altera FPGA technology.

11. REFERENCES

[1] P. Gauravaram, L.R. Knudsen, K. Matusievicz, F. Mendel, C. Rechberger, M. Schläffer, and S.S. Thomsen, "Grostl - A SHA-3 Candidate", submission to the NIST SHA-3 Competition, available at http://www.groestl.info/, 2011.

[2] NIST, FIPS197, "Advanced encryption standard," available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.

[3] Altera DE2-115 FPGA board user manual, available at www.terasic.com.

[4] National Institute of Standards and Technology, Cryptographic Hash Project, csrc.nist.gov/groups/ST/hash/index.html.

[5] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[6] C. E. Shannon, "Communication Theory of Secrecy Systems", Bell System Technical Journal, vol.28-4, pp. 656-715, 1949.

[7] M.M. Wong, M.L.D., Wong, A.K. Nandi, and I. Hijazin, "Composite field GF$(((2^2)^2)^2)$ AES S-box with algebraic normal form representation in the subfield inversion", IET Circuits, Devices & Systems, vol. 5, no. 6, pp. 471-476, 2011.

[8] K. Järvinen, "Sharing Resources Between AES and the SHA-3 Second Round Candidates Fugue and Grøstl", presented at the Second SHA-3 Candidate Conference, 2010.

[9] E. Homsirikamol, M. Rogawski, and K. Gaj. "Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs", IACR Eprint report 2010/445, available at eprint.iacr.org/2010/445.pdf, 2010.

[10] M. Rogawski and K. Gaj, "A High-Speed Unified Hardware Architecture for AES and the SHA-3 Candidate Grøstl", presented at Euromicro Conference on Digital System Design (DSD 2012), Izmir, Turkey, 2012.