Compact Hardware Implementation of the Hash Function AURORA-256

Toru Akishita[1], Tadaoki Yamamoto[2], and Hiroyuki Abe[2]

[1] Sony Corporation
[2] Sony LSI Design Inc.
{Toru.Akishita,Tadaoki.Yamamoto,HiroyukiE.Abe}@jp.sony.com

Abstract. This report notes compact hardware implementation of the hash function AURORA-256.

1 Introduction

AURORA is a hash function family which is a first round candidate of SHA-3 competition [1]. AURORA consists of six hash algorithms including AURORA-224 and AURORA-256 whose hash size are 224 bits and 256 bits, respectively. AURORA-224/256 are constructed from the secure and efficient compression function using a security-enhanced Merkle-Damgård transform, i.e., the strengthened Merkle-Damgård transform with the finalization function. The compression function is designed based on the well-established design techniques for blockciphers, and uses the Davies-Meyer construction.

AURORA-224/256 achieves high efficiency both in software implementations and in hardware implementations. Especially, in hardware implementations, AURORA-256 enables a variety of implementations, from high-speed to area-restricted implementations. Using a $0.13 \mu m$ CMOS ASIC library, AURORA-256 achieves the highest throughout of 10.4 Gbps. In an area-optimized implementation, AURORA-256 can be implemented with only 11.1 Kgates, which is achieved using AURORA-256 Type-H4 implementation with area optimization. In the implementation, two F-function circuits are used and the data path width is 128 bits.

In this report, we aim more compact implementation of AURORA-256. By using only one F-function circuit and 64-bit data path width, we can reduce gate size to about 8.9 Kgates, which is the smallest area among so-called "fully autonomous implementations" in SHA-3 hardware implementations on the SHA-3 zoo website [2].

2 AURORA-256 Type-H5 implementation

In [1], four types of hardware implementations were considered. Among them, AURORA-256 Type-H4 architecture achieved the smallest area. It processes a round of the AURORA architecture in one of the message scheduling module

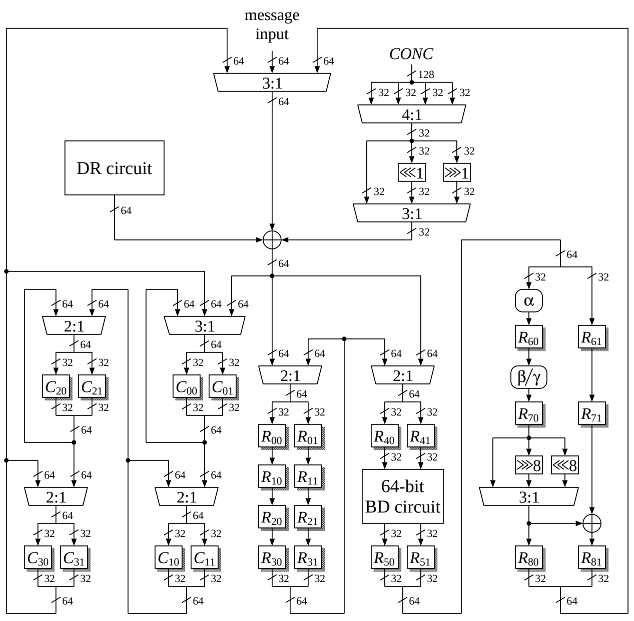

Fig. 1. Data path architecture of AURORA-256 Type-H5 implementation

MSM or in the chaining value processing module *CPM* mutually in every two clock cycles. It requires two F-function circuits and takes 72 cycles for both the compression function *CF* and the finalization function *FF*.

In order to achieve smaller area, we design another type of implementation, AURORA-256 Type-H5 implementation. AURORA-256 Type-H5 implementation processes a round of the AURORA architecture in one of *MSM* or in *CPM* mutually in every four clock cycles. It requires only one F-function circuit and takes 144 cycles for both *CF* and *FF*. Fig. 1 shows the data path architecture of AURORA-256 Type-H5 implementation, where the data path width is 64 bits. In the figure, all registers represented by a box with shadow are composed of registers without enable signal.

In the design, the circuits required for multiplication by the matrix $\mathcal{M}_0$ and $\mathcal{M}_1$ are merged. For an input vector (x_0, x_1, x_2, x_3) and an output vector (y_0, y_1, y_2, y_3), the multiplication by $\mathcal{M}_i$ $(i = 0, 1)$ can be computed through the following equations.

$$\begin{cases} a_0 = \{02\} \times x_0 \\ a_1 = \{02\} \times x_1 \\ a_2 = \{02\} \times x_2 \\ a_3 = \{02\} \times x_3 \end{cases} \begin{cases} b_0 = a_1 \oplus x_0 \\ b_1 = a_2 \oplus x_1 \\ b_2 = a_3 \oplus x_2 \\ b_3 = a_0 \oplus x_3 \end{cases} \begin{cases} c_0 = b_0 \oplus a_3 \\ c_1 = b_1 \oplus a_0 \\ c_2 = b_2 \oplus a_1 \\ c_3 = b_3 \oplus a_2 \end{cases} \begin{cases} d_0 = a_3 \oplus x_0 \\ d_1 = a_0 \oplus x_1 \\ d_2 = a_1 \oplus x_2 \\ d_3 = a_2 \oplus x_3 \end{cases}$$

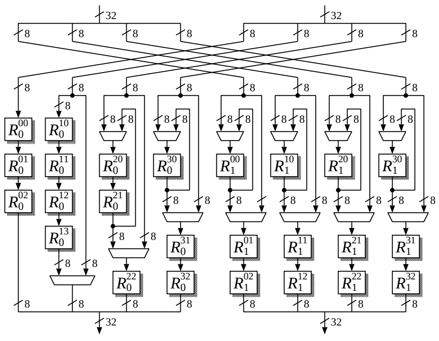

Fig. 2. 64-bit Byte Diffusion (*BD*) circuit

$$\begin{cases} e_0 = \{\texttt{04}\} \times b_0 \\ e_1 = \{\texttt{04}\} \times b_1 \\ e_2 = \{\texttt{04}\} \times b_2 \\ e_3 = \{\texttt{04}\} \times b_3 \end{cases} \begin{cases} f_0 = (i == 1)?\ e_1 : d_3 \\ f_1 = (i == 1)?\ e_2 : d_0 \\ f_2 = (i == 1)?\ e_3 : d_1 \\ f_3 = (i == 1)?\ e_0 : d_2 \end{cases} \begin{cases} y_0 = c_0 \oplus f_0 \\ y_1 = c_1 \oplus f_1 \\ y_2 = c_2 \oplus f_2 \\ y_3 = c_3 \oplus f_3 \end{cases}$$

$\oplus$ denotes an addition of two elements in GF(2^8), which costs 8 XOR logic gates and $\times$ denotes a multiplication in GF(2^8). For an element a in GF(2^8), $\{\texttt{02}\} \times a$ and $\{\texttt{04}\} \times a$ require 3 and 5 XOR logic gates, respectively. The total number of XOR gates and 2:1 selector gates required for multiplication by $\mathcal{M}_0/\mathcal{M}_1$ are 160 and 32.

In a 64-bit data path architecture, the 256-bit byte diffusion function *BD* cannot be implemented only by simple wiring of byte data. We utilize the 64-bit byte diffusion (*BD*) circuit, as shown in Fig. 2. It consists of byte wiring, twenty-five 8-bit registers and thirteen 8-bit 2:1 selectors. 256-bit data, which are input into the 64-bit *BD* circuit in four clock cycles, are output in the order corresponding to *BD* by controlling selectors.

The round function circuit is shared for *MSM* and *CPM*, and processed by repeating the following steps:

$$MS_L\ (MSF_L) \rightarrow CP\ (CPF) \rightarrow MS_R\ (MSF_R) \rightarrow CP\ (CPF) \rightarrow \cdots$$

The left 256-bit M_L of a 512-bit message block is input in 64-bit blocks from the 1st cycle to the 4th cycle, and then intermediate values of MS_L (MSF_L) are stored in registers by repeating the following steps:

$$\{R_{00}, R_{01}\} \rightarrow \{R_{10}, R_{11}\} \rightarrow \{R_{20}, R_{21}\} \rightarrow \{R_{30}, R_{31}\} \rightarrow \{R_{00}, R_{01}\} \rightarrow$$
$$\{R_{10}, R_{11}\} \rightarrow \{R_{20}, R_{21}\} \rightarrow \{R_{30}, R_{31}\} \rightarrow \{R_{40}, R_{41}\} \rightarrow \text{64-bit BD circuit} \rightarrow$$
$$\{R_{50}, R_{51}\} \rightarrow \{R_{60}, R_{61}\} \rightarrow \{R_{70}, R_{71}\} \rightarrow \{R_{80}, R_{81}\} \rightarrow \cdots$$

The right 256-bit M_R of a 512-bit message block is input in 64-bit blocks from the 9th cycle to the 12th cycle, and then intermediate values of MS_R (MSF_R) are stored in registers by repeating the same order as MS_L.

On the other hand, the chaining value stored in two 32-bit registers $\{C_{30}, C_{31}\}$, $\{C_{20}, C_{21}\}$, $\{C_{10}, C_{11}\}$ and $\{C_{00}, C_{01}\}$ is loaded via $\{C_{30}, C_{31}\}$ from the 5th cycle to the 8th cycle, and then 256-bit intermediate values of CP (CPF) are stored in registers by repeating the following steps:

$$\{R_{40}, R_{41}\} \rightarrow \text{64-bit BD circuit} \rightarrow \{R_{50}, R_{51}\} \rightarrow \{R_{60}, R_{61}\} \rightarrow$$
$$\{R_{70}, R_{71}\} \rightarrow \{R_{80}, R_{81}\} \rightarrow \cdots$$

Note that the chaining value to be fed forward is XORed into intermediate values of MS_R (MSF_R) in advance from the 137th cycle to the 140th cycle, which can reduce four cycles for updating the chaining value.

3 Evaluation Results

We show our evaluation results of AURORA-256 Type-H5 implementation. The environment of our hardware design and evaluation is as follows.

Language	Verilog-HDL
Design library	0.13 μm CMOS ASIC library
Simulator	VCS version 2006.06
Logic synthesis	Design Compiler version 2007.03-SP3

One gate is equivalent to a 2-way NAND and speed is evaluated under the worst-case conditions. Table 1 represents the evaluation results of AURORA-256 Type-H5 implementation together with those of AURORA-256 Type-H4 implementation and the best known results of SHA-224/256 [3]. For each implementation two types of circuits are synthesized by specifying either area or speed optimization. In addition, we investigate the condition to maximize "Efficiency" that indicates "Throughput" per area, which we call efficiency optimization.

The gate size of AURORA-256 Type-H5 implementation (8,870 gates) in area optimization is about 20% and 23% smaller than that of AURORA-256 Type-H4 implementation (11,111 gates) and SHA-224/256 (11,484 gates). This is the smallest gate size among so-called "fully autonomous implementations" in SHA-3 hardware implementations on the SHA-3 zoo website [2].

References

1. T. Iwata, K. Shibutani, T. Shirai, S. Moriai, and T. Akishita, "AURORA: A cryptographic hash algorithm family." In *Submission to NIST*, 2008.
2. SHA-3 Hardware Implementations on The SHA-3 Zoo website, `http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations`
3. A. Satoh and T. Inoue, "ASIC-hardware-focused comparison for hash functions MD5, RIPEMD-160, and SHS." *Integration, the VLSI Journal*, Vol. 40, No. 1, pp. 3–10, Jan. 2007.

Table 1. Evaluation Results of AURORA-256 Type-H5 implementation

	Data Path Architecture	Cycles	Area [gates]	Frequency [MHz]	Throughput [Mbps]	Efficiency [Kbps/gate]
AURORA-256 ($0.13\mu m$)	Type-H5	144	8,870	304.8	1,084	122.2
			9,970	509.3	1,811	181.6
			9,970	509.3	1,811	181.6
	Type-H4	72	11,111	306.4	2,179	196.1
			14,255	475.3	3,380	237.1
			12,257	423.6	3,012	245.7
SHA-224/256 ($0.13\mu m$) [3]	-	72	11,484	154.1	1,096	95.4
			15,329	333.3	2,370	154.6

For each implementation, the 1st row and the 2nd row show the results of the synthesized circuits by area and speed optimization, respectively. The 3rd row also shows the results by efficiency optimization for each implementation of AURORA-256.