Compact Hardware Design of Whirlpool Hashing Core

Timo Alho and Panu Hämäläinen
Nokia Technology Platforms
Tampere, Finland
{timo.a.alho, panu.hamalainen}@nokia.com

Marko Hännikäinen and Timo D. Hämäläinen
Tampere University of Technology,
Institute of Digital and Computer Systems
Tampere, Finland
{marko.hannikainen, timo.d.hamalainen}@tut.fi

Abstract

Weaknesses have recently been found in the widely used cryptographic hash functions SHA-1 and MD5. A potential alternative for these algorithms is the Whirlpool hash function, which has been standardized by ISO/IEC and evaluated in the European research project NESSIE. In this paper we present a Whirlpool hashing hardware core suited for devices in which low cost is desired. The core constitutes of a novel 8-bit architecture that allows compact realizations of the algorithm. In the Xilinx Virtex-II Pro XC2VP40 FPGA, our implementation consumes 376 slices and achieves the throughput of 81.5 Mbit/s. The resource utilization of our design is one fourth of the smallest Whirlpool implementation presented to date.

1 Introduction

Cryptographic hash algorithms belong to the basic primitives that are employed in numerous cryptographic applications [7]. They are widely used e.g. in message authentication, digital signature, and key derivation schemes. A hash algorithm takes in an arbitrary-length data input and compresses it into a fixed-size output value called *hash*. Compared to the hash algorithms of other fields, the required properties for cryptographic hash algorithms are that they are *one-way functions* and *collision resistant* [7].

Traditionally, SHA-1 [11] and MD5 [10] have been the most utilized algorithms. However, researches have recently found weaknesses in both of them [12, 13]. This has raised a need for new designs. One of the potential alternatives for replacing the traditional choices is a hash algorithm called Whirlpool [1]. In 2003, it was selected as a part of the New European Schemes for Signatures, Integrity and Encryption (NESSIE) portfolio of cryptographic primitives as result of a research project within the Information Societies Technology (IST) Programme of the European Commission [8]. Furthermore, the ISO/IEC standard 10118-3 [4] on dedicated hash functions contains the Whirlpool algorithm.

Cryptographic algorithms are utilized for security services in various environments in which low cost and low power consumption are key requirements. Examples of such technologies are wireless local and personal area networks, sensor networks, and smart cards. In these environments security procedures are often among the tasks requiring most of the overall processing capacity. Hence, it is beneficial to implement cryptographic algorithms in hardware as significantly higher performance and lower power consumption can be achieved compared to software. For continuing the evaluation of new hash algorithms, in this paper we present a compact hardware design of the Whirlpool algorithm for low cost devices.

According to the authors' knowledge, three publications for the hardware implementation of Whirlpool have previously been published. Kitsos *et al.* [5] and McLoone *et al.* [6] present straightforward designs that utilize a full-width (512-bit) data paths for maximizing the throughput at the expense of hardware resource consumption. In order to reduce the amount of resources, Pramstaller *et al.* [9] decrease the data path width to 64 bits. In our design, we further decrease the data path width to eight bits for achieving even lower resource consumption while still providing reasonable throughput. Compared to the previous designs, our design is well-suited for low cost devices due to its significantly smaller size. Similarly to the references, we implement our design on Field Programmable Gate Arrays (FPGA) for enabling direct comparison of results.

The rest of the paper is organized as follows. Section 2 overviews the Whirlpool algorithm. Our 8-bit hardware architecture is presented in Section 3. The implementation results and comparisons are presented in Section 4. Section 5 concludes the paper.

2 Whirlpool Algorithm

The Whirlpool algorithm [1], shown in Fig. 1, is based on a dedicated block cipher W that operates on 512-bit

978-3-9810801-2-4/DATE07 © 2007 EDAA

data blocks (n_i) using a 512-bit key (H_{i-1}). Initially, the key input (H_0) is a string of zeroes. The hash function is constructed from a *compression function* using the Merkle-Damgård method [7]. The compression function has been built from the internal block cipher with the Miyaguchi-Preneel construction [7]. Before the message is subjected to hashing operation, a specific padding is appended to the message [1].

The block cipher W is composed of ten identical rounds of transformations. All basic operations are performed on 8-bit bytes and each byte is interpreted as a polynomial in the *Galois Field* GF(2^8). The cipher operates on a 512-bit *hash state* using a chained 512-bit *key state*, derived from the data input and the key input respectively. The states are internally viewed as 8-by-8 matrices of bytes.

The data path and the key schedule of W and the sequences of the round transformations are illustrated in Fig. 1. The notations used in this paper are consistent with the Whirlpool specification [1]. The operations are performed in the data path as well as in the key schedule during a round are defined as follows:

- The *non-linear layer* γ is a byte substitution operation. The substitute values for the bytes of the state are derived from the substitution box (S-box) S [1] individually:

$$\gamma(a) = b \iff b_{ij} = S[a_{ij}],\ 0 \le i, j \le 7.$$

- During the *cyclical permutation* π, each column of the state is operated independently. The column j is cyclically shifted downwards by j positions:

$$\pi(a) = b \iff b_{ij} = a_{(i-j) \bmod 8, j},\ 0 \le i, j \le 7. \quad (1)$$

- The *diffusion layer* θ transformation is a linear mapping that can be expressed as a matrix multiplication with a constant matrix in GF(2^8):

$$\theta(a) = b \iff b = a \cdot C, \text{where}$$

$$C = \begin{bmatrix} 01_x & 01_x & 04_x & 01_x & 08_x & 05_x & 02_x & 09_x \\ 09_x & 01_x & 01_x & 04_x & 01_x & 08_x & 05_x & 02_x \\ 02_x & 09_x & 01_x & 01_x & 04_x & 01_x & 08_x & 05_x \\ 05_x & 02_x & 09_x & 01_x & 01_x & 04_x & 01_x & 08_x \\ 08_x & 05_x & 02_x & 09_x & 01_x & 01_x & 04_x & 01_x \\ 01_x & 08_x & 05_x & 02_x & 09_x & 01_x & 01_x & 04_x \\ 04_x & 01_x & 08_x & 05_x & 02_x & 09_x & 01_x & 01_x \\ 01_x & 04_x & 01_x & 08_x & 05_x & 02_x & 09_x & 01_x \end{bmatrix}.$$

  The notation used in the elements of C denote polynomials in GF(2^8) [1]. The reduction polynomial for the matrix multiplications is

$$x^8 + x^4 + x^3 + x^2 + 1.$$

- The *key addition* $\sigma[k]$ is a bitwise addition (XOR) of a key matrix k with the state:

$$\sigma[k] = b \iff b_{ij} = a_{ij} \oplus k_{ij},\ 0 \le i, j \le 7.$$

The key schedule expands the initial 512-bit cipher key into a sequence of 11 round keys $K^0, K^1, \ldots K^{10}$, i.e. key states. Before the transformations of the first round, the input message block n_i is XORed with the first round key K^0, which equals to the input H_{i-1}. In the key addition phase of the key schedule, an iteration-round-dependent constant matrix c^r is XORed to the key state. The constant for the round r is defined as:

$$\begin{aligned} c^r_{0j} &\equiv S[8(r-1)+j], & 0 \le j \le 7, \\ c^r_{ij} &\equiv 0, & 1 \le i \le 7, 0 \le j \le 7. \end{aligned}$$

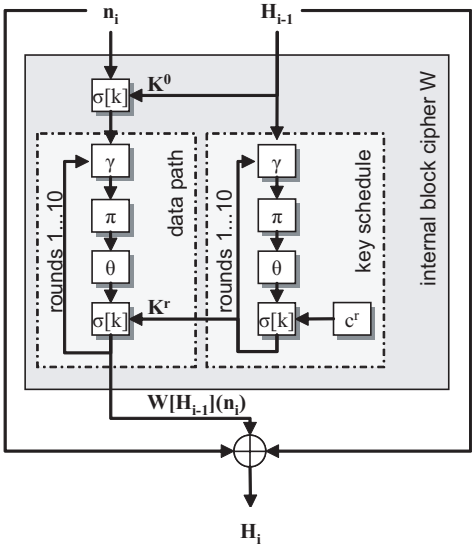

Figure 1. The data path of the Whirlpool algorithm.

3 Whirlpool Hashing Core Architecture

Whirlpool has especially been designed to be executed on 8-bit and 64-bit processors since the basic operations are performed on 8-bit bytes and the longest data dependency in one round of the algorithm is within a 64-bit block. Hence, an attractive choice of the data path width for the hardware implementation is a multiple of 8 or 64 bits. In this work we chose to design a 8-bit data path in order to minimize the hardware area. The mathematical structure of the internal block cipher W is almost identical with the structure

of the Advanced Encryption Standard (AES) algorithm [2]. Therefore, many of the design choices adopted in compact AES implementations can also be used here.

The high-level architecture of our Whirlpool core is depicted in Fig. 2. All the connections in the figure are eight bits wide. The architecture consists of two separate data paths, one for the hash state and the other one for the key state. Each data path contains a 64×8-bit Dual-Port RAM (DPRAM) (x-RAM and K-RAM), a S-box γ, a diffusion unit θ, and a XOR gate for the key addition $\sigma[k]$. As the storages for the states and the cyclical permutation transformation π are combined into the DPRAMs, the order of γ and π is reversed. The third 64×8-bit DPRAM, H-RAM, is used for storing the partial hash result H_i. The usage of DPRAMs allows reading the next byte while writing a previously processed byte, which increases performance. One additional S-box is used for generating the c^r bytes. The data paths iterate the round operations ten times.

The operation of the core can be divided into three steps:

1. *Data loading*: The 64 bytes of the data block are loaded from the input *n_i* byte by byte. At the beginning of the loading, H-RAM contains the result of the previous ten rounds of operation H_{i-1}. For the first data block, the output of H-RAM is masked to zero. During the loading, the multiplexers m_1, m_2, and m_3 are controlled so that $H_{i-1} \oplus n_i$ is loaded to H-RAM and x-RAM, and H_{i-1} to K-RAM.

2. *Data processing*: The round function is iterated ten times for both the hash state and the key state in parallel. For each round, the states are processed one row at a time. The rows are read from x-RAM and K-RAM byte by byte. Processing a row takes 16 clock cycles, from which eight cycles are used for reading of the operands and the other eight cycles for writing the result. During the first nine rounds, the multiplexers m_2 and m_3 are controlled so that the results of each round are written back to x-RAM and K-RAM. During the last round of the block cipher operation, m_1 is set so that the final result of the round ($W[H_{i-1}](n_i)$) is XORred with contents of H-RAM ($H_{i-1} \oplus n_i$) and written back to H-RAM. Therefore, at the end of the last round the contents of the H-RAM is $H_i = W[H_{i-1}](n_i) \oplus H_{i-1} \oplus n_i$. At this point, the contents of x-RAM and K-RAM are no longer needed.

3. *Data unloading*: After processing the last message block, the hash can be read from H-RAM. The 64 bytes of the result are read from the port *H_i* a byte at a time.

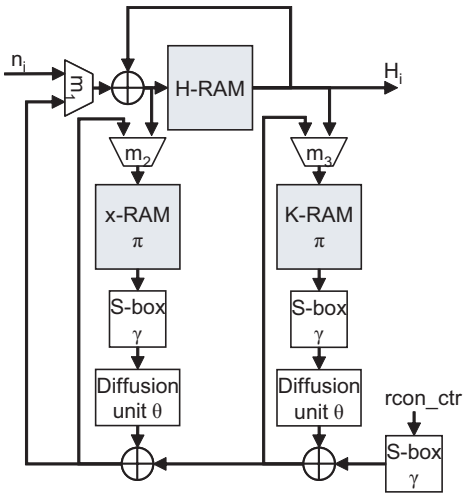

Figure 2. The high-level data path architecture of the Whirlpool hash core.

Excluding the data unloading, a block of data is processed in $64 + 10 \times 8 \times 16 = 1344$ clock cycles. The required message padding must be added outside the hardware core. The following subsections describe the design of the subcomponents.

3.1 Cyclical Permutation

Since the states are stored in memories, the permutation π can be performed with addressing logic. The data elements of the states remain at the same memory locations during the round operations but the addressing sequence differs from round to round. After the bytes of a row are processed, they are written back to the same memory locations. The 64-entry DPRAMs are organized so that the three most significant bits of a memory address indicate the row of a state matrix and the three least significant bits indicate the column.

Initially, before the first round, the bytes of the state s are organized so that the element s_{ij} is located in the memory location $[i, j]$ (the notation $[i, j]$ refers to the memory location $8i + j$). It can be seen from (1) that during the first round of operation the jth element of the row i must be read from the location

$$[(i - j) \bmod 8, j].$$

Similarly, since the bytes are stored back in the same locations, the read address for the second round is

$$[((i - j) \bmod 8 - j) \bmod 8, j] = [(i - 2j) \bmod 8, j].$$

This can be generalized for the round r, for which the address location for the jth element of the row i is

$$[(i - rj) \bmod 8, j].$$

The addressing logic satisfying this is implemented with three 3-bit counters i, j, and r and a small amount of combinatorial logic.

3.2 Substitution Box

The non-linear layer transformation γ makes use of the S-box S that has 256 8-bit entries. It can directly be implemented as combinatorial logic or ROM. However, due to the internal structure of S, it can also be constructed from 16-entry mini-boxes E, E^{-1}, and R as shown in Fig. 3. The contents of the boxes are given in [1].

Each mini-box can be implemented by using only four 4-input Look-Up-Tables (LUT), which typically are the basic building blocks of FPGA devices. Since each XOR gate consumes one 4-input LUT, the total resource consumption for a mini-box-based S-box is 32 4-input LUTs. On the contrary, an FPGA experiment of ours showed that a direct 256 entry LUT-based S-box implementation consumes 208 4-input LUTs or 2048 memory bits when implemented with combinational logic or ROM blocks, respectively. Therefore, the mini-box-based approach was selected for this work. In addition, to increase the maximum clock frequency, the S-box was pipelined into two stages.

3.3 Diffusion Unit

The constant polynomial coefficients of C in the diffusion layer transformation θ have been chosen so that they contain as few terms as possible. The needed basic operation for the matrix multiplication is a finite field multiplication in GF(2^8) with the polynomial 02_x, shown in Fig. 4(a). The multiplications by 04_x and 08_x can be calculated by successively applying the multiplication with 02_x, since $04_x \cdot d = 02_x \cdot 02_x \cdot d$ and $08_x \cdot d = 02_x \cdot 04_x \cdot d$. Similarly, the multiplications with 05_x and 09_x can be performed by using previously computed values and one addition (which is simply an XOR operation), since $05_x \cdot d = 04_x \cdot d \oplus d$ and $09_x \cdot d = 08_x \cdot d \oplus d$. The complete circuit that computes all the required multiplications for a state byte, shown in Fig. 4(b), requires 25 XOR gates in total.

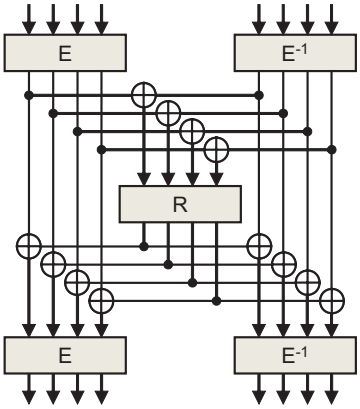

Figure 3. Structure of the Whirlpool S-box using five mini-boxes [1]. All connections are one bit wide.

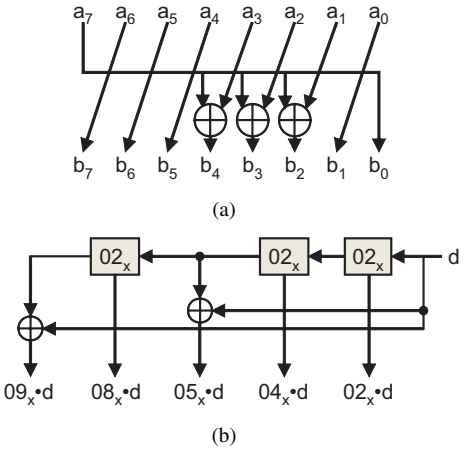

Figure 4. (a) Bit level schematic for computing $b = 02_x \cdot a$ using the reduction polynomial $x^8 + x^4 + x^3 + x^2 + 1$. The parameters a_i and b_i denote single bits. (b) Byte level schematic for computing $09_x \cdot d$, $08_x \cdot d$, $05_x \cdot d$, $04_x \cdot d$, and $02_x \cdot d$.

The diffusion unit presented in Fig. 5 processes one row of the state in 16 clock cycles. The multiplier unit exploits the circulant property of the multiplicand matrix in the same way as [3] in the implementation of the AES algorithm. Data are fed to the unit's input (*d_in*) byte by byte and the intermediate results are maintained in the eight registers. During the first eight cycles of the operation, the multiplication operation is performed by adding and cyclically shifting the intermediate results in the unit. During the

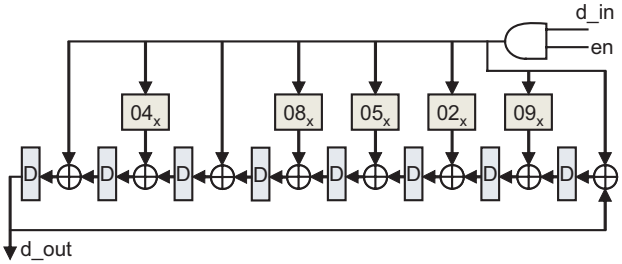

Figure 5. Diffusion layer multiplier unit. 02_x, 04_x, 05_x, 08_x, and 09_x denote field multiplications with polynomials x, x^2, $x^2 + 1$, x^3, and $x^3 + 1$, respectively. All connections are eight bits wide.

last eight cycles, the input is masked to zero with the control signal *en* and the row is read byte by byte from the output (*d_out*).

4 Results and Comparison

The Whirlpool core was described in VHDL and synthesized to two different target FPGAs, Altera Stratix EP1S40 and Xilinx Virtex-II Pro XC2VP40. The reference implementations have used Xilinx FPGAs. The basic configurable block in Altera Stratix FPGAs is a *Logic Element* (LE), which includes a 4-input LUT and a register. On the contrary, the basic block in Xilinx Virtex-II FPGAs is a *slice*, which includes two 4-input LUTs and two registers. In both the technologies, the basic blocks contain additional logic e.g. for carry propagation and register bypassing. Both the FPGA technologies provide dedicated memory blocks of different capacities (block RAM) for implementing a variety of memory functions. The LUTs of Virtex-II can also function as small single-port or dual-port memories (distributed RAM).

Table 1 lists the resource utilization, the maximum clock frequency, and the maximum throughput for our implementations and the references. In the Altera FPGA, K-RAM, x-RAM, and H-RAM were mapped to dedicated memory blocks whereas in the Xilinx FPGA slices were used as distributed RAM.

Of the reference implementations, Kitsos *et al.* [5] present two different implementation approaches. The first one contains one iterative 512-bit data path that interleaves the operations performed to the hash state and the key state. The data path is first utilized for the full key schedule and the computed round keys are stored in a RAM. The total cycle count for hashing a 512-bit block of data is 20 clock cycles. The other implementation contains two separate 512-bit data paths. The round key calculation is performed on-the-fly in parallel with the hash state computations. The resource consumption is therefore higher than in the first implementation but the cycle count is halved, resulting in higher throughput.

McLoone *et al.* [6] report results for two different 512-bit-wide architectures. The first one equals to the second implementation of Kitsos *et al.* [5] but implements the S-box as a large LUT instead of the mini-box approach. The required S-boxes are implemented in dedicated memory blocks. In the second architecture, the iteration loop is unrolled so that two rounds are performed per clock cycle. This approach considerably increases the resource consumption, yet implying only a modest increase in throughput.

The 64-bit Whirlpool implementation of Pramstaller *et al.* [9] interleaves the on-the-fly round key generation and the hash state computations. The S-box has been implemented using the mini-box approach. The cycle count for hashing a 512-bit block is 176.

Table 1 shows that the resource utilization of our design is significantly lower than that of the reference designs at the expense of lower throughput. Our maximum clock rate is considerably higher than in the reference implementations, which is due to the pipeline registers in the S-box that shorten the critical path. It can be seen that the throughput per used logic block is superior in the 512-bit implementations compared to the 64-bit and 8-bit implementations.

Table 1. Implementation results and comparison.

Design	**FPGA device**	**Data width** [bits]	**Logic blocks**	**Memory blocks**	**Cycles/ block**	**Max. freq.** [MHz]	**Throughput** [Mbit/s]
This work	Altera Stratix EP1S40	8	443 LEs	3 M512s (1.5 Kbits)	1344	169	64.3
	Xilinx Virtex-II Pro XC2VP40	8	376 slices	-	1344	214	81.5
Pramstaller *et al.* [9]	Xilinx Virtex-II Pro XC2VP40	64	1456 slices	-	176	131	382
Kitsos *et al.* [5]	Xilinx Virtex-E V1000E	512	3751 slices	-	20	93	2380
	Xilinx Virtex-E XCV1000E	512	5585 slices	-	10	87.5	4480
McLoone *et al.* [6]	Xilinx Virtex-4 XC4VLX100	512	4956 slices	68 BRAMs (1224 Kbits)	10	94.6	4790
	Xilinx Virtex-4 XC4VLX100	512	13210 slices	-	5	47.8	4896

This is not surprising as a full-data-width implementation can take full advantage of the parallelism in the Whirlpool algorithm whereas 64-bit and 8-bit implementations have to use extra effort (i.e. more control logic and multiplexers) to time share the effective computational resources. The same effect can be seen e.g. in AES implementations utilizing different data path widths [3].

5 Conclusion

In this paper we presented the design and implementation of a Whirlpool hardware core for low cost devices. The architecture was based on a 8-bit data path in order to minimize the hardware resource consumption. As a result, we achieved considerably lower hardware area compared to the reference implementations. Due to the lower utilization of parallelism, our throughput was also lower. However, we believe that our approach is well suited for environments in which low power consumption and low cost are the key design goals. A possible enhancement to increase the throughput of the core is to parallelize the memory accesses between consecutive rows (i.e. by simultaneously reading a new row and writing the previous row). This modification would almost double the throughput while only requiring modifications to the control logic and two extra 64-bit registers to temporarily store the result of the previous row.

References

[1] P. S. L. M. Barreto and V. Rijmen. The Whirlpool hashing function. http://planeta.terra.com.br/informatica/paulobarreto/whirlpool.zip, May 2003.

[2] Advanced encryption standard (AES). Federal Information Processing Standards Publication 197, 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`, cited 27th Nov 2006.

[3] P. Hämäläinen, T. Alho, M. Hännikäinen, and T. D. Hämäläinen. Design and implementation of low-area and low-power AES encryption hardware core. In *Proc. 9th Euromicro Conference on Digital System Design (DSD 2006)*, Cavtat, Croatia, Aug. 2006.

[4] Information technology–security techniques–hash-functions–part 3: Dedicated hash-functions. ISO/IEC 10118-3, 2004.

[5] P. Kitsos and O. Koufopavlou. Efficient architecture and hardware implementation of the Whirlpool hash function. *IEEE Trans. Consumer Electronics*, 50(1):208–213, 2004.

[6] M. McLoone, C. McIvor, and A. Savage. High-speed hardware architectures of the Whirlpool hash function. In *Proc. IEEE 2005 Int. Conf. Field-Programmable Technology (FPT'05)*, pages 147–153, Singapore, Dec. 2005.

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

[8] NESSIE. New European Schemes for Signatures, Integrity, and Encryption. IST-1999-12324, Apr. 2004.

[9] N. Pramstaller, C. Rechberger, and V. Rijmen. A compact FPGA implementation of the hash function Whirlpool. In *Proc. ACM/SIGDA 14th Int. Symp. Field Programmable Gate Arrays (FPGA 2006)*, pages 159–166, Monterey, CA, USA, Feb. 2006.

[10] R. Rivest. The MD5 message-digest algorithm. RFC 1321, Apr. 1992.

[11] Secure hash standard. Federal Information Processing Standards (FIPS) 180-2, 1999.

[12] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *Proc. 25th Annu. Int. Cryptology Conf., Advances in Cryptology (CRYPTO 2005)*, pages 17–36, Santa Barbara, CA, USA, Aug. 2005.

[13] X. Wang and H. Yu. How to break MD5 and other hash functions. In *Proc. 24th Int. Conf. Theory and Application of Cryptographic Techniques, Advances in Cryptology EUROCRYPT 2005*, pages 19–35, Aarhus, Denmark, May 2005.