LC-008

Three stages pipelined MD5 implementation on FPGA

Hoang Anh Tuan† Katsuhiro Yamazaki† and Shigeru Oyanagi†

1. Introduction

The hash algorithm or message digest algorithm is used to generate a unique message digest for an arbitrary message. The digest must have three characteristics of easily to compute, hardly to inversely compute message from the digest, and hardly to find other message with same message digest [1].

Several message digest algorithms have been developed such as the SHA or MD5 algorithms. They are used to guarantee the correctness of the transmitted data. Some implementations of MD5 on the FPGA were introduced. The SIG-MD5 [2] was introduced with parallel and pipeline implementation patterns to increase the throughput to 725 Mbps with 7997 slices on Xilinx Vertex-II XC2V4000-6 FPGA chip. Other full-loop-unrolling and iterative looping architecture were introduced with 351 Mbps and 165 Mbps of throughput respectively [3].

In this paper, we introduce our work of 3 stages pipeline MD5 implementation on FPGA, called PPMD5. This work breaks the data dependency of a single step inside the main loop of the MD5 algorithm into 3 stages and makes them pipelined. The implementation achieves 704 Mbps and uses 1010 hardware slices with no BRAM on Xilinx Vertex-II XC2V4000-6 FPGA chip.

2. The MD5 algorithm

The MD5 algorithm is widely used in public key cryptographic algorithm and internet communication to guaranty the message's integrity and authentication. It calculates 128 bit digest from an arbitrary message through 4 steps of appending padding bits (Padding), appending length (Length), initialization (Init), and message compression (H_{MD5}) as can be seen in Figure 1.

(1) Appending padding bits is used to guarantee that the length of the appended message is able to divide into 512-bit blocks. It adds a single 1-bit and multiple 0s into the original b-bit message until the length of the message is congruent to 448, modulo 512.

(2) A 64-bit representation of the length of the original message is added into the result of (1) in this step. After this, the final length of the message will be a multiple of 512 bits.

(3) The initialization step starts the 128-bit message digest as 4 words of 32-bit, called A, B, C, D. At the beginning, they are initiated as constants.

$$A = 0x67452301 \quad B = 0xefcdab89 \quad C = 0x98badcfe \quad D = 0x10325476 \qquad [1]$$

(4) Message compression (H_{MD5}) is the heart of the MD5 algorithm. It processes all 512-bit blocks (Y_0 to Y_{L-1}) of the padded message in sequence form the first one, compresses them into 128 bit message digest. The H_{MD5} step handles the 512-bit input data as 16-bit words, represent for 16 keys (X_0.. X_{15}).

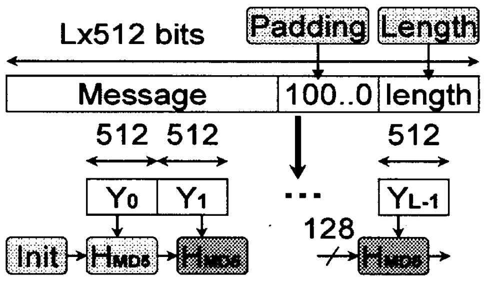


Figure 1. MD5 algorithm.

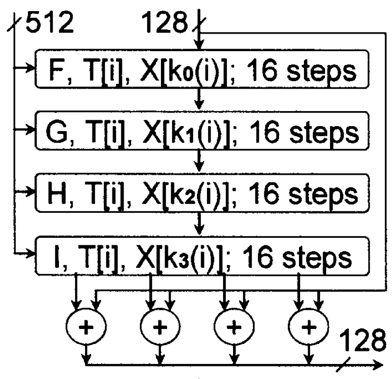

512
128
F, T[i], X[k0(i)]; 16 steps
G, T[i], X[k1(i)]; 16 steps
H, T[i], X[k2(i)]; 16 steps
I, T[i], X[k3(i)]; 16 steps
+
128

Figure 2. Compression algorithm.

The initial values of 128-bit message digest of A, B, C, D are stored as AA, BB, CC, and DD for the later usage. The algorithm consists of four rounds, and comprises 16 steps each as shown in Figure 2. Hence, 64 steps are performed in the algorithm, denoted by i. Inside the H_{MD5}, functions (Func) F, G, H, and I represent four functions, specified for the four rounds:

$$\begin{aligned} F(B, C, D) &= (B \wedge C) \vee (\sim B \wedge D) \\ G(B, C, D) &= (B \wedge D) \vee (C \wedge \sim D) \\ H(B, C, D) &= B \oplus C \oplus D \\ I(B, C, D) &= C \oplus (B \vee \sim D) \end{aligned} \qquad [2]$$

T is a 64-elemrnt constant table and used by 64 steps. X is the 16-word keys with the location relies on the round and the internal step. Address of the key in the 16-element key table (X_0..X_{15}) is calculated depend on the round, named k_0, k_1, k_2, and k_3 respectively:

$$\begin{aligned} k_0(i) &= i \\ k_1(i) &= (1+5i) \bmod 16 \\ k_2(i) &= (5+3i) \bmod 16 \\ k_3(i) &= 7i \bmod 16 \end{aligned} \qquad [3]$$

A single step of the compression algorithm is represented by the equation [4]

$$A = B + ((A+Func(B,C,D)+X[k]+T[i]) <<< s) \qquad [4]$$
$$A \leftarrow D;\ B \leftarrow A;\ C \leftarrow B;\ D \leftarrow C$$

†Ritsumeikan University, Graduate School of Science and Engineering

in which, <<< represents a circle shift left operation. The bit-shift value s depends on the step number and is extracted from a 16-element table.

The final values A, B, C, D of the message compression function H_{MD5} is generated by adding the results of 64 internal loops with the initial values of AA, BB, CC, and DD, respectively as can be seen in equation [5] and Figure 2.

$$\begin{aligned} A &= A + AA \\ B &= B + BB \\ C &= C + CC \\ D &= D + DD \end{aligned} \qquad [5]$$

3. An architecture for the MD5 algorithm

Figure 3 shows the proposed architecture for the MD5 system. The heart of the architecture is the H_{MD5} module, which performs all equations [1] to [5]. The architecture in Figure 3 performs steps 3 to 5 of the MD5 algorithm shown in section 2. The message padding and length appending are not included in this design, because they can be carried out simply by hardware or software before writing into the memory.

Inputs of the design are as follows: the message is given to the design as data in the memory (MEM). The data is divided into blocks of 512-bit, written into continuous location. The addresses of that data are controlled in the MD5 by a register. Number of those blocks is known as init_number and provided from step 2. That 64-bit number shows how many times the H_{MD5} module must perform the calculation in order to finish the compression for a long message. It is provided at the start time, together with the start_new signal, and saved into register for later access. Start address of the data is also provided as start_addr signal at this time. The start_new signal will reset all registers into the initial or provided values.

The readX signal shows that some of the 32-bit data (among 512-bit key) are still waiting in the memory. It is controlled by a count-down register. It guarantees that 16 data of X are written into H_{MD5} module each time.

The H_{MD5} module itself contains memory for the 16 32-bit keys or X reg. Therefore, after the keys are read, this module can

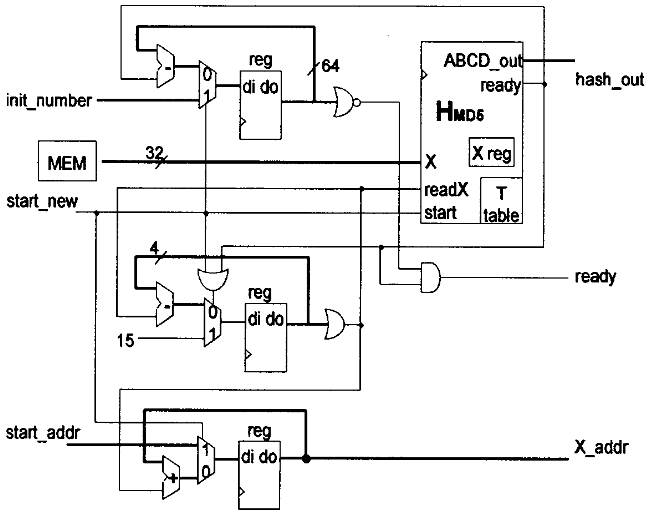


Figure 3. An architecture for MD5 algorithm.

work separately with the memory, and the memory can be accessed by other processes such as processes for steps 1 and 2 in section 2. The readX signal is understood as data of new block is available by the H_{MD5} and the H_{MD5} will start its process to read keys for 16 times. The start signal simply reloads the initial values shown in equation [1] into A-D. Details of the H_{MD5} module in this implementation are shown in section 5.

4. Data dependency, data movement and pipelining methodology among steps

4.1 Data dependency and data movement

The tree in Figure 4 shows the data dependency in computing the new value of B between two continuous steps. As can be seen, the new value of B, calculated by equation [4] relies on previous values of A, B, values of T, X, s and Func. X itself relies on its location denoted by k, which must be calculated from the step number i. Func depends on the previous values of B, C, D and current step i. Figure 4 also shows that the new value of B of the current step completely depends on the step and previous values of A~D. However, the internal values of T, k, X can be pre-computed because they rely on the step number i only. A can also be predicted within some steps before.

Figure 5 shows the data movement of A within 2 steps before from the current step, in which u shows the values that changes at each step. In order to compute the required B value at current step, A is required at current step. That value is transferred from D and C at 1 and 2 steps before respectively, which means the value of A at current step can be defined at 2 steps before as C.

4.2 Pipelining methodology

In this PPMD5 implementation, we manage to implement a single step of MD5 into pipeline based on the data dependency in equation [4]. As can be seen in Figure 4, the computation of B

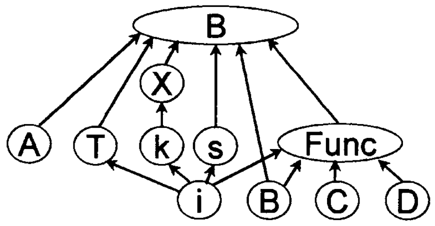
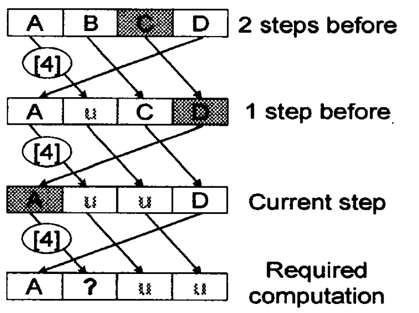


Figure 4. Data dependency in single step.



u: unknown values calculated at other steps
?: required computation value at current step

Figure 5. Trace of A within 2 steps.

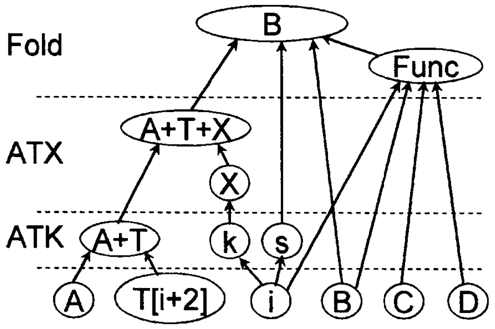


Figure 6. Pipelining methodology.

will require a huge sequenced computation of k, X, Func and 3 units of 32-bit adder. This generates an enormous latency. However, if we re-write the equation [4] into

$$A = B + ((A+T[i]+X[k]+Func(B,C,D)) <<< s) \quad [6]$$
$$A \leftarrow D;\ B \leftarrow A;\ C \leftarrow B;\ D \leftarrow C$$

while giving a look to the trace of A, that latency can be divided into smaller stages. Address of the key of the current step can be pre-computed several steps before, because it relies mainly on the step number i. The trace of A in Figure 5 allows us to define value of A at current step as D or C at 1 or 2 steps before, respectively. All that make A+T+X be able to pre-compute up to 3 steps before.

Figure 6 shows the methodology to divide the computations of equation [6] into 3 pipeline stages of ATK, ATX and Fold. The ATK stage computes values of A plus T (in equation [6]) and address of the key (in equation [3]). The ATX stage receives results of ATK stage in order to compute A+T+X[k] value. Finally, all values are gathered to form B in the Fold stage. The biggest latency occurs at Fold stage with the Func, shift and plus operations.

Figure 7 shows the operations of those pipeline stages inside 64 steps of the H_{MD5} algorithm. When a compression of a key is computed, it first goes to the ATK stage for preparation of A plus constant before moving to the second stage of ATX. In ATX

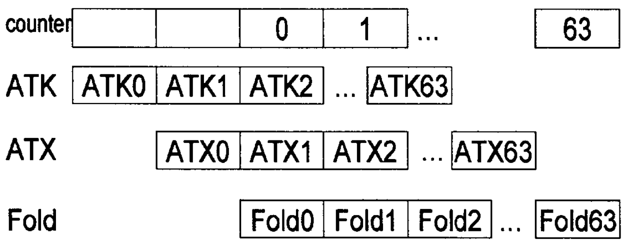


Figure 7. Pipeline operations.

stage, the data given by ATK stage is added with the required key. At this time, the ATK module is used for the next data preparation. Same operation occurs when data moves into the Fold stage, the final result of required step is given while the ATX and ATK modules are used to prepare values for the next and 2 steps latter compression computations, respectively. In order to complete the compression of one block of data, 65 clocks are required from the starting time.

5. Implementation

5.1 Data implementation

There are 4 different data with different length involved in the design of the MD5 algorithm. They include the keys, constant values, shift values, and the message digest data. In this implementation, the keys and digest data are designed as matrixes of registers that contain 16 and 4 elements respectively. The constant is designed as a 64-element hardware look-up-table. The s value used for the shift operation is specified as 16 cases shift unit. Other intermediate data are specified as 32-bit registers.

5.2 Pipeline implementation of the compression function

Figure 8 shows the pipeline implementation of the compression module H_{MD5}. It contains 3 stages of ATK, ATX, and Fold as described in section 4. Those stages contact to the other through intermediate register or pipeline registers AT, k and ATX. Register AT is used to store C+T value, which means

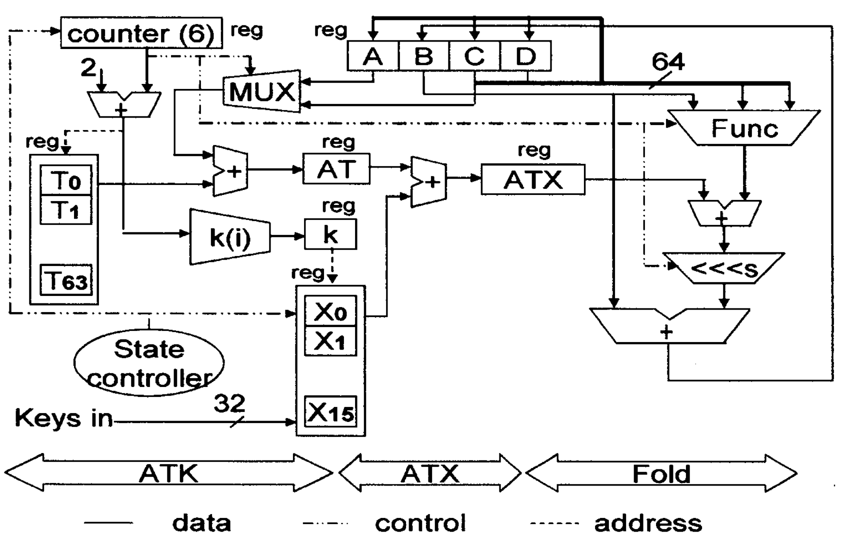


Figure 8. Pipeline design of H_{MD5}.

A+T at Fold stage. Location of the key is stored in k register, which means k(i+2) at the ATK stage. ATX register represents the addition of that 3 data for the current step. The registers AT and k are used to connect ATK and ATX stage while the register ATX is used to connect ATX and Fold stage.

The ATK stage contains two computation units of plus (+) and k(i). The plus module simply adds two 32-bit data of A and constant T[i] in equation [6] together. The k(i) module is the implementation of equation [3]. However, this stage occurs at two steps (clocks) before the current Fold stage. Hence, the counter is i+2 in the T[i] and k(i) module while A is taken as C following the trace shown in Figure 5. After this stage, value of A+T and address of the corresponding key are stored and transferred to the ATX stage. The plus module in ATX takes value of key at address shown by k and value of AT register before adding them together and stores into the ATX register for the Fold stage. The Fold stage completes the computation of equation [6] using Func module that represents operation of equation [2], <<<s module that specifies the circle shift for s bit, and two plus modules. Result of this is written back to the digest value A~D following equation [6].

6. Implementation results and discussion

The pipelined MD5 design was implemented on the Virtex-II XC2V4000-6 devices and compiled by Xilinx ISE 8.1 version. Table 1 shows the results of the PPMD5 pipelined implementation. The throughput of the design is calculated by ((frequency*block size) / clocks per block) achieves 704.3 Mbps. In this design, all the memory is used as registers inside the hardware. Therefore, the hardware size becomes large with 1010 hardware slices in use. The hardware size can be reduced significantly if large memory such as constant table (T) or the key table can be used as memory in BRAM.

There are some other implementations, which achieve very high throughput, up to 2Gbps and 5Gbps by parallel execution such as SIG-MD5-HT4p and SIG-MD5-HT10i [2]. However, their pipeline implementation methodology is completely different from this research. They make pipeline by dividing the 64-step operation into 2, 4, 32 or 64 stages. This helps to reduce difficulties inside a single step but increases the hardware size significantly in order to complete all 64 steps. The high throughput will be achieved by compressing several messages at the same time in parallel. On the contrary, this design tries to increase throughput by breaking a single step into 3 stages that can be managed in pipeline. It makes each of them simpler and faster. Therefore, the PPMD5 implementation achieves a good tradeoff between hardware size and throughput in comparison with others. This design divided a single step into smaller stages; hence has no conflict with the previous implementations such as SIG-MD5 systems. Those techniques of SIG-MD5 can also be applied together with PPMD5 to make two-layer pipeline systems.

Table 1. Implementation results of the compression function

Name	Device	Slices	LUTs	Flip Flops	BRAMs	Clock (MHz)	Throughput (Mbps)
PPMD5	Virtex-II XC2V4000-6	1010	1887	863	0	88	704.3

Table 2. Delay of the modules

Stage	ATK	ATX	Fold	PPMD5	Normal MD5
Delay (ns)	9.2	9	11.25	11.35	17.38

Effectiveness of the pipeline architecture in this implementation can be seen in Table 2. Delay of PPMD5 shows the logic delay in pipeline implemented H_{MD5} module, while delay of Normal MD5 shows the logic delay of the implemented H_{MD5} module without pipeline technology. Breaking the single step shown in Figure 4 into 3 small stages helps to decrease the delay of normal implementation into 2/3 in the PPMD5 implementation. However, the hardware size slightly increases from 972 slices of the normal implementation to 1010 slices due to the overhead of pipelining in PPMD5 implementation.

As can be seen in Figure 8 and Table 2, the Fold stage is not equal in hardware complexity and delay time in comparison with other stages. Hence, the main delay of the system occurs in this stage. Breaking the Fold stage into 2 not only makes equal in computation time among stages but also helps to increase frequency. However, one more clock will be required for every step computation. In this case, throughput can be increased if parallel mechanism is used to compute MD5s of 2 messages simultaneously. Besides, the combination of separating the PPMD5 architecture into steps and making parallel message input in the same manner with SIG-MD5 system is also another method to increase the throughput for PPMD5 architecture.

7. Conclusion

This paper described the implementation of the core of MD5 algorithm into 3-stage pipeline. The results show that this architecture achieves good tradeoff between hardware and throughput. The pipeline implementation achieves 704.3 Mbps. It required 1010 slices on the XC2V4000 device. The implementation also should be improved with more stages pipelining in combination with parallel message compression in order to get higher hardware/throughput tradeoff.

Reference

[1] RFC 1321 – The MD5 Message-Digest Algorithm

[2] K.Jarvinen et al.: Hardware Implementation Analysis of the MD5 Hash Algorithm, Proc 38th IEEE International Conference on System Sciences-2005.

[3] J. Deepakumara et al.: FPGA Implementation of MD5 Hash Algorithm, Proc of the Canadian Conference on Electrical and Computer Engineering, CCECE 2001, Vol.2, pp.919-924, 2001.