High-Speed MARS Hardware

Akashi Satoh[†], Nobuyuki Ooba[†], Kohji Takano[†], Edward D'Avignon[††]
[†]IBM research, Tokyo Research Laboratory, IBM Japan Ltd., 1623-14, Shimotsuruma,
Yamato-shi, Kanagawa 242-8502, Japan
{akashi, ooba, chano}@jp.ibm.com
[††]IBM Corporation, Poughkeepsie, NY 12601, USA
davignon@us.ibm.com
March 15, 2000

Abstract. High-speed MARS encryption/decryption hardware was developed using a 0.18μm IBM CMOS technology. In order to boost performance, a special adder and multiplier was designed by optimizing the adder block structure and interconnections between adder cells using signal delay profiles. A description of the hardware including block diagrams and data flow diagrams is presented. One of the most critical portions of the design is the special adder and multiplier. The design philosophy and tradeoffs used in these pieces are discussed. Finally, performance and size estimates are presented along with the rationale behind them. The design achieves 677Mbit/s data rate for encryption when using cipher block chaining and 1.28Gbit/s for decryption and other encryption modes in 13.8Kgates + 2.25Kbyte SRAM.

1. Introduction

MARS [1] is a symmetric-key block cipher, supporting 128-bit blocks and a variable key size. It is designed to take advantage of the powerful operations supported by today's computers, resulting in a much improved security/performance tradeoff over existing ciphers. We developed high-speed MARS hardware for use when additional performance or security is required over a software implementation. Since MARS uses 32-bit multiplications and additions in conjunction with S-box lookups, it is essential for MARS hardware to have a high-speed multiplier and adder. The key to realizing high-speed arithmetic circuits is to first break one operation into parallel sub-operation blocks, then precisely adjust and control the number of signal delays from each block. We developed an automatic circuit generation program, which optimizes the parallel block structure and the wiring interconnection by using the signal delay profiles. A high-speed adder with the combination of carry-skip [2] and carry-select [3] techniques designed for an RSA encryption LSI [4] was implemented in the final stage of the multiplier. These arithmetic circuits boost the speed of MARS hardware while maintaining compact silicon area.

In this paper, we first show the data path level design of the MARS hardware with an overview of the MARS algorithm and how encryption and decryption are performed. Next, we discuss the techniques that apply to the adders and multipliers to realize the high-speed MARS computation. Finally, we give estimated performance results and the size of the MARS hardware.

2. MARS Algorithm and Hardware Architecture

2.1. Hardware Block Diagram

We designed the MARS hardware entirely from the gate level to the chip level, so that it is ready for chip fabrication. Figure 1 shows the block diagram of the hardware. It has a chip external bus, which consists of a 32-bit data bus, a 10-bit address bus, four control signals, and a clock, to interface with external logic, such as a CPU. Through the bus, the external logic will read and write message data and the key. The hardware has a forward/backward mixer, a cryptographic core for MARS encryption/decryption, and a key expander for key setup. During those operations, two S-boxes and key storage are accessed. Each S-box is a 32-bit × 256-word SRAM. The key storage is a 32-bit × 64-word SRAM.

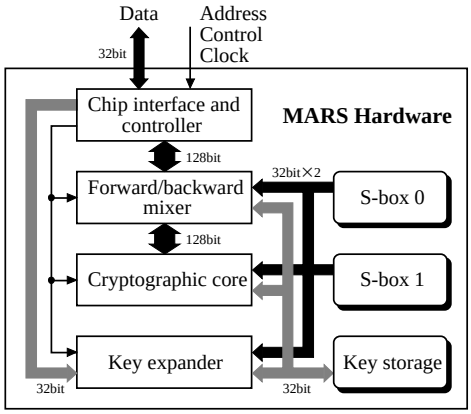

Figure 1. Block diagram of MARS hardware.

2.2. Encryption Procedure

The MARS encryption procedure has three phases: 8-round "forward mixing," 16-round "cryptographic core," and 8-round "backward mixing," as shown in Figure 2. Figure 3 shows the type-3 Feistel network structure of MARS. A 128-bit plain text block is divided into four 32-bit data words M_0, M_1, M_2, M_3, and encrypted as four words D_0, D_1, D_2 and D_3. In the figure, ⊕ denotes XOR, "<<<*n*" and ">>>*n*" denote *n*-bit cyclic left and right rotations, respectively. The lower 32 bits of the results of addition, subtraction and multiplication are used; the higher bits are discarded. MARS uses S-box (32-bit × 512-word table) lookups in the key setup, encryption, and decryption procedures. The S-box is composed of two 256-entry tables S0 (the first 256 words) and S1 (the last 256 words), used in the forward and backward mixing phases. The decryption procedure is the inverse of the encryption operation, and all circuits shown in this paper are used for both procedures by switching selectors in the data paths.

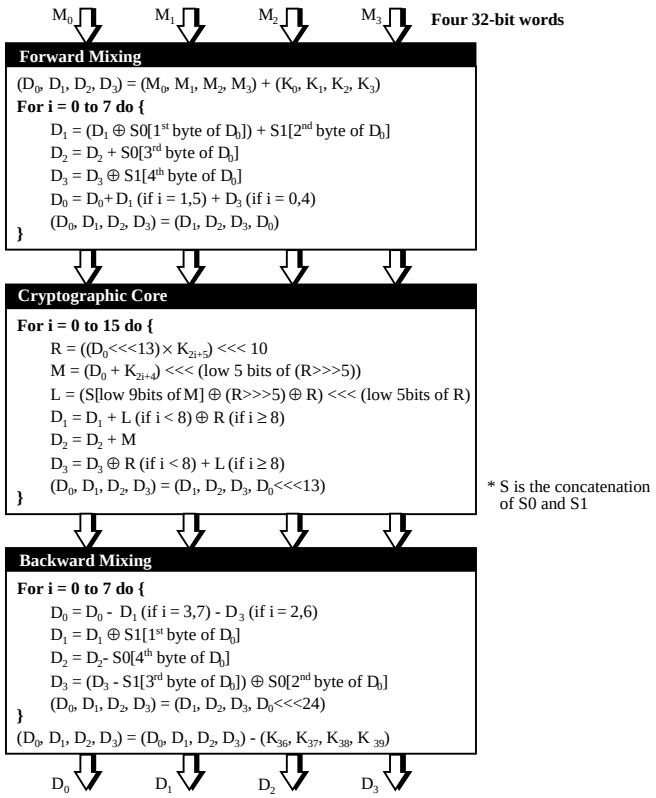

Figure 2. MARS encryption procedure.

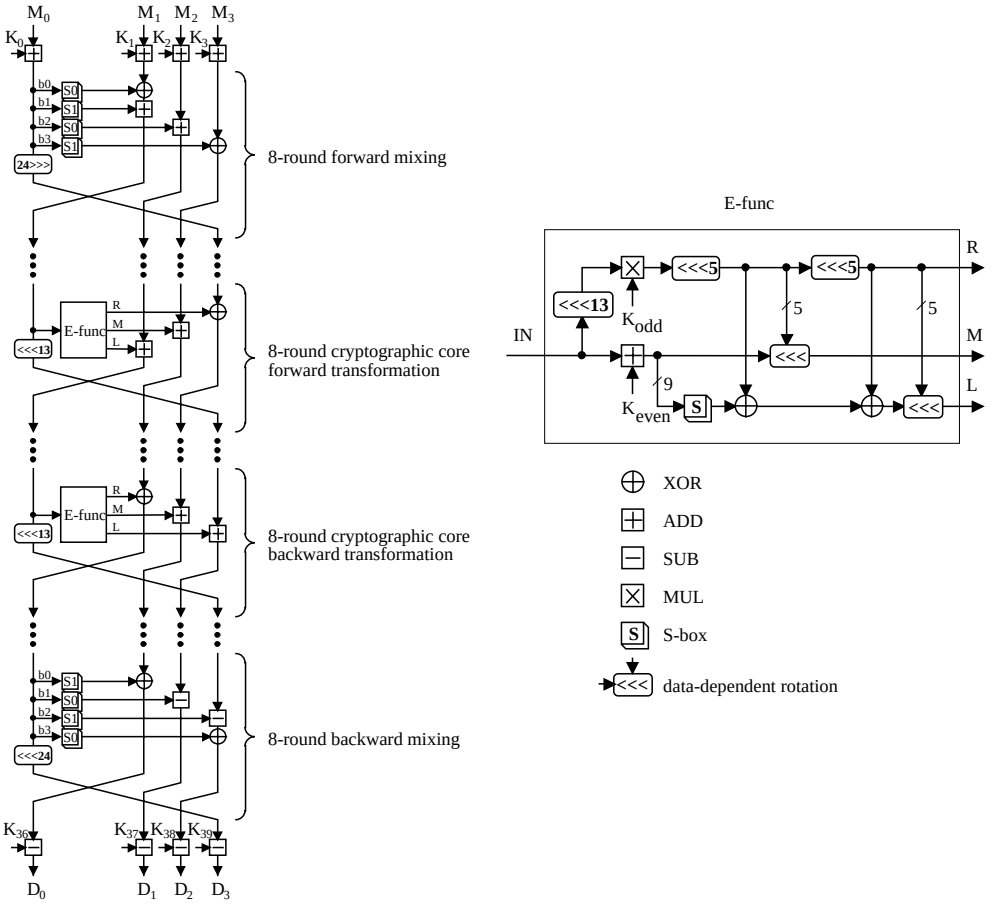

Figure 3. Type-3 Feistel network structure.

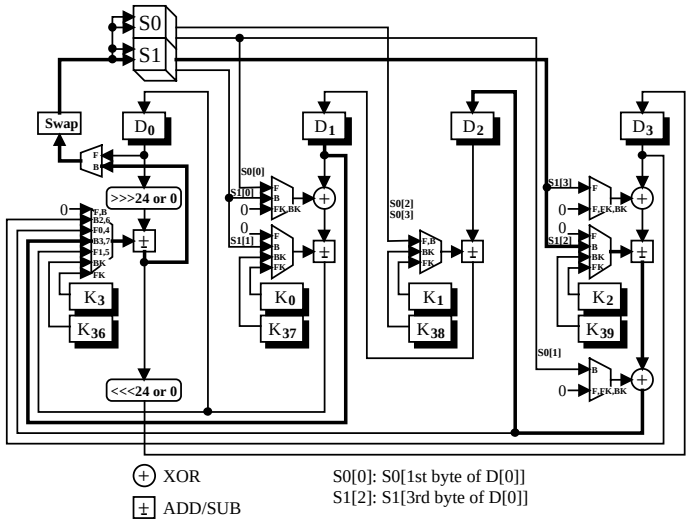

Figure 4. Forward / backward mixing data paths.

Figure 4 shows the circuit block diagram of the forward and backward mixing data paths. This circuit is also shared by the encryption and decryption procedures. Switching the selectors changes the order of the operations. The S-boxes, S0 and S1, are implemented by three-port SRAMs, one port for the write and two ports for the read operations. The thick lines show the critical path for the backward mixing process, which contains subtraction, S-box, subtraction, and XOR operations in order. Two sets of key registers $K_{0\text{-}3}$ and $K_{36\text{-}39}$ are dedicated to this mixing operation, and eight key words are copied from the 32-bit × 40-word expanded keys stored in SRAM "K".

This circuit block is used 9 times in the forward mixing mode, then one cycle is required to add the sub-keys $K_{0\text{-}3}$ to the data $D_{0\text{-}3}$, and then 8 times in the rounds of mixing operation. The backward mixing operation takes 9 cycles.

Figure 5 is the block diagram of the cryptographic core (Feistel network) data path. The thick lines specify the critical path. It consists of a multiplier, two XORs, a conditional rotator, an adder and a selector. The S-box read operation is executed in parallel with the multiplication, so that the memory access time does not affect the critical path. The S-box shares the SRAM used for the forward and backward phases shown in Figure 4. The cryptographic core operation uses this circuit in the 8-round keyed forward transformation followed by the 8-round keyed backward transformation. The cryptographic core requires 16 cycles for each 128-bit block encryption.

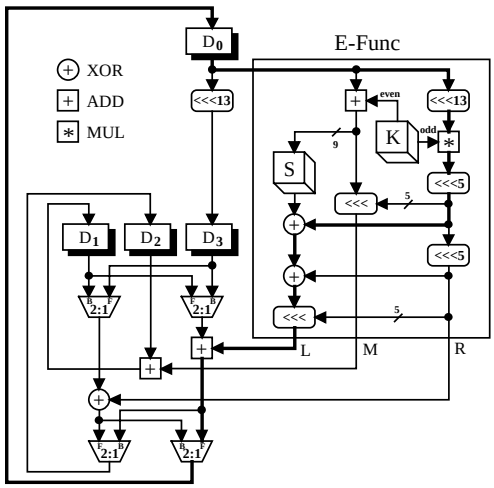

Figure 5. Cryptographic core data path.

The cryptographic core and the forward/backward mixer can operate simultaneously on separate 128-bit blocks when four-port (one for write and three for read) SRAM is used as the S-box. A 128-bit bus connection can swap data between these two circuits without additional cycles. If we share the circuits of Figure 4 with forward and backward mixing operations to save hardware resources, 18 cycles are required for one set of encryption procedures. A timing chart for this case, which is suitable for electronic codebook (ECB) encryption mode and all decryption modes, is given in Figure 6 (a). The data throughput of this architecture is 128 bits / 18 cycles. For cipher block chaining (CBC) encryption mode, the encrypted data D in the previous cycle is required before starting the current encryption of block M. In this case, the mixing phases cannot be pipelined with the cryptographic core. CBC operations require 34 cycles, with the throughput becoming 128 bits / 34 cycles. The timing chart for cipher block chaining encryption mode is shown in Figure 6 (b).

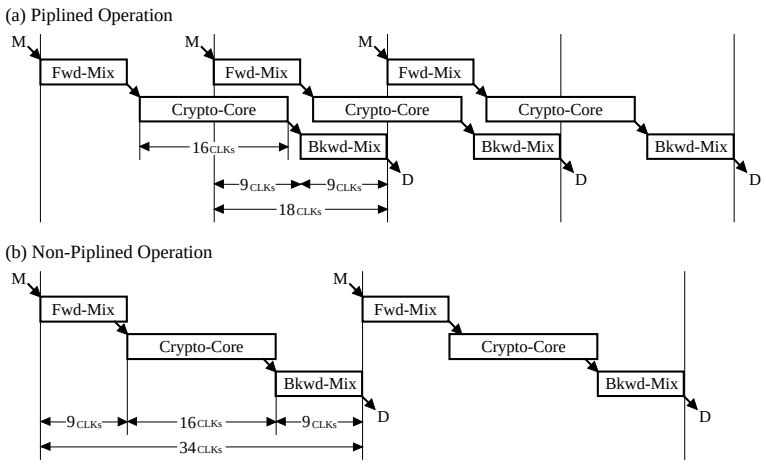

Figure 6. Timing chart of MARS encryption.

2.3. Key Expansion

The key expansion procedure, shown in Figure 7, expands the user-supplied key array, k_0, …, k_{n-1}, into a 40-word internal key array, K_0, …, K_{39}. The range of *n* is from 4 to 14 32 bit words, that is, MARS supports user key lengths from 128 bits to 448 bits. In the figure, bit-wise OR and AND are denoted by $\vee$ and $\wedge$, respectively. The block diagram of the key expander data paths is shown in Figure 8. The major components of the key expansion circuit are a barrel rotator, two registers, an adder, and multiplexers. The key storage "K" is implemented using a three-port SRAM. It is capable of one write and two read operations in parallel. We designed the key expander with a small number of latches in order to keep it small in size. The temporary storage T, which is used during the key expansion procedure, is implemented in the SRAM. For this reason, the key storage has 64 entries of 32 bits data. Key expansion takes 752 to 848 cycles depending on the value of the key.

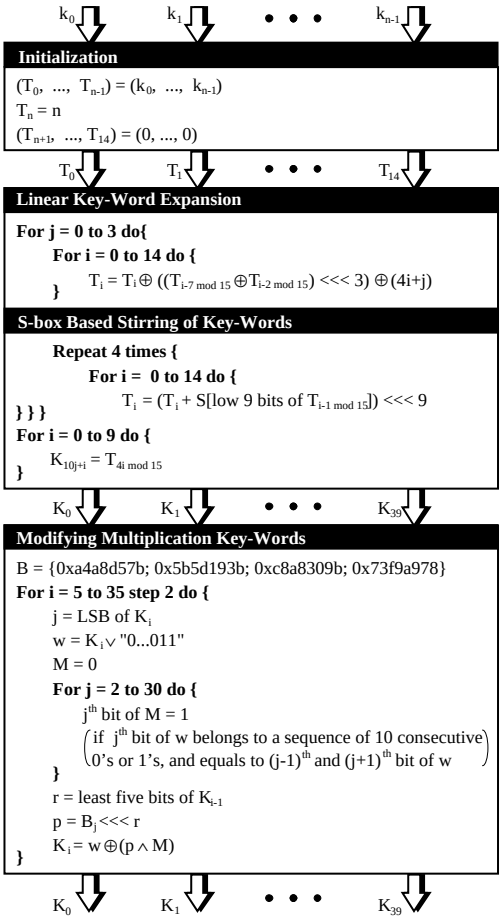

Figure 7. Key expansion procedure.

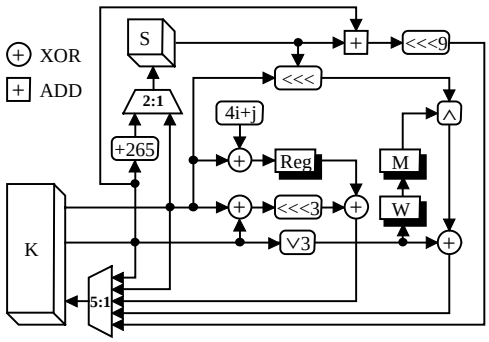

Figure 8. Key expander data path.

3. High-Speed Adder and Multiplier

3.1. High-Speed Adder

In this section, we first explain the design of a high-speed adder employing a combination of carry-skip [2] and carry-select [3] techniques used in the RSA encryption LSI [4]. This adder is used in the E-function and in the last stage of the multiplier. It is one of the most critical parts affecting MARS hardware performance.

Figure 9 shows the basic structure of the adder. It consists of ripple-carry adder blocks where each successive block is one bit longer than the block immediately below along with a carry-skip path jumping over each adder block. The delays in the ripple-carry adders and the carry-skip path are well balanced so that every carry propagates from the LSB to the MSB without waiting for the results from the other blocks. To simplify the figure, a full adder cell FA is used in the first bit of each adder block. It can be replaced by a half adder cell in the actual implementation.

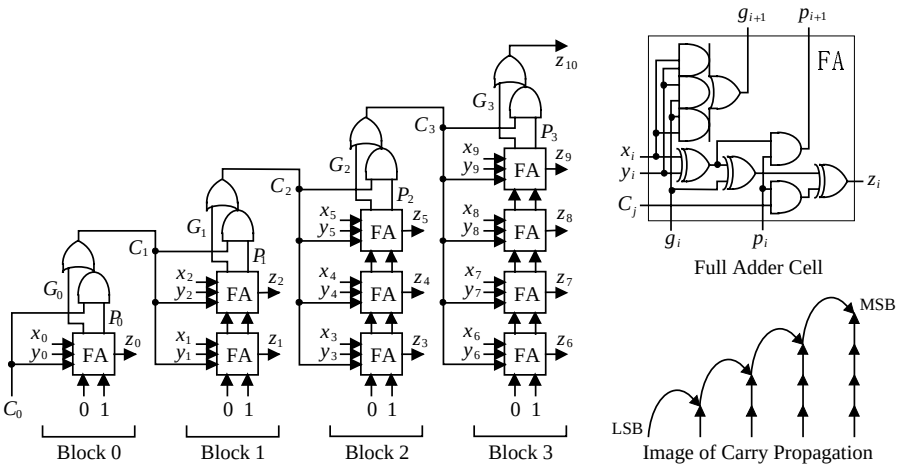

Figure 9. High-speed adder.

If two or more bits of x_i, y_i and g_i are '1' in the i-th full-adder cell, carry $g_{i+1} = 1$ is generated and fed to the next cell. The cell never generates a carry if both x_i and y_i are '0', regardless of the input g_i. If $g_{i+1} = 0$, either x_i or y_i is '0' and the other is '1', it will generate a carry if carry $C_j = 1$ comes up from the lower ripple-carry adder block j-1. For example, when (x_3, x_4, x_5) = (1, 1, 1) and (y_3, y_4, y_5) = (0, 0, 0), block 2 does not generate carries g_4, g_5, g_6 (= G_2). However, if the carry $C_2 = 1$ reached the block, the carry output C_3 immediately becomes '1.' This means that the carry C_j can skip over the blocks one after another by pre-calculating a condition between x_i and y_i in each adder block j. The condition is defined by

$$P_j = \prod_i x_i \oplus y_i = 1, \text{ where } \oplus \text{ is XOR.}$$

By making the adder block size bigger toward the MSB side, the propagation time of P_j and C_j are equalized, and therefore the total delay time is minimized. Output z_i initially holds a sum as if the block carry C_j is 0, and is inverted by the XOR gate if $C_j = 1$ comes up later.

3.2. High-Speed Multiplier

A standard n-bit × n-bit multiplier gives a 2n-bit result by repeatedly summing up the n-bit partial product rows. The multiplier used in MARS is not required to calculate the higher half of the result, as shown in Figure 10, so it is faster and smaller than standard multipliers. The high-speed techniques described in this section, however, can be applied to any multiplier. Figure 11 shows a Wallace tree [5], which is an adder cell array commonly used in a multiplier to reduce the number of partial product rows. The tree takes three rows and produces one carry row and

one sum row, so the full adder cell, FA, is called "3:2 compressor." This reduction is repeated until there are only two partial product rows, which are added together with a high-speed carry-propagation adder. Several tree architectures, which use 4:2, 6:2 and 9:2 compressors, were proposed [6][7] to optimize the critical path of this tree, but these compressors basically consist of 3:2 compressors. Booth encoding [8] is widely used to reduce the number of partial products, but it is a kind of 4:2 compression technique and does not change the tree structure. Oklobdzija et al [9] suggested that not all inputs and outputs from a compressor contribute equally to the delay, and the difference in using 4:2 and higher order compressors is not in the structure of the compressor but in the way they are interconnected.

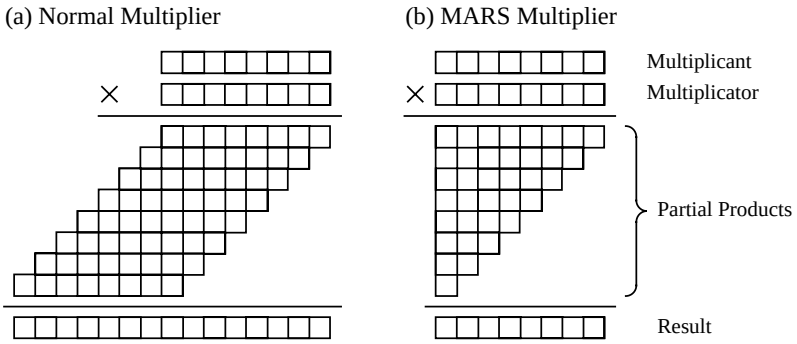

Figure 10. Partial products in MARS multiplier (n = 8).

In Figure 11, the input signals x and y of the full adder FA pass through two XORs to the output s, but the input c_i goes through only one XOR gate. The full-adder FA and half-adder HA located at the later stage of the tree are shaded in the figure. The delay profile of the tree is shown with the same shading. Here, all the XOR, NAND and AND gates are assumed to have the same propagation delay. The two signals fed into the adder at the bit-5 location come from the third-stage half adders marked with '*,' but the right signal arrives earlier than the left one. In addition, the propagation delay from an input to an output varies with the types of gate and input pin locations. For example, AND usually operates faster than XOR, and NAND is faster than AND. For that reason, we developed an optimal Wallace tree generation program in consideration of six delay propagation paths of a full adder (combination of the three inputs to two outputs) and four paths of a half adder, based on a 0.18μm IBM CMOS standard cell library.

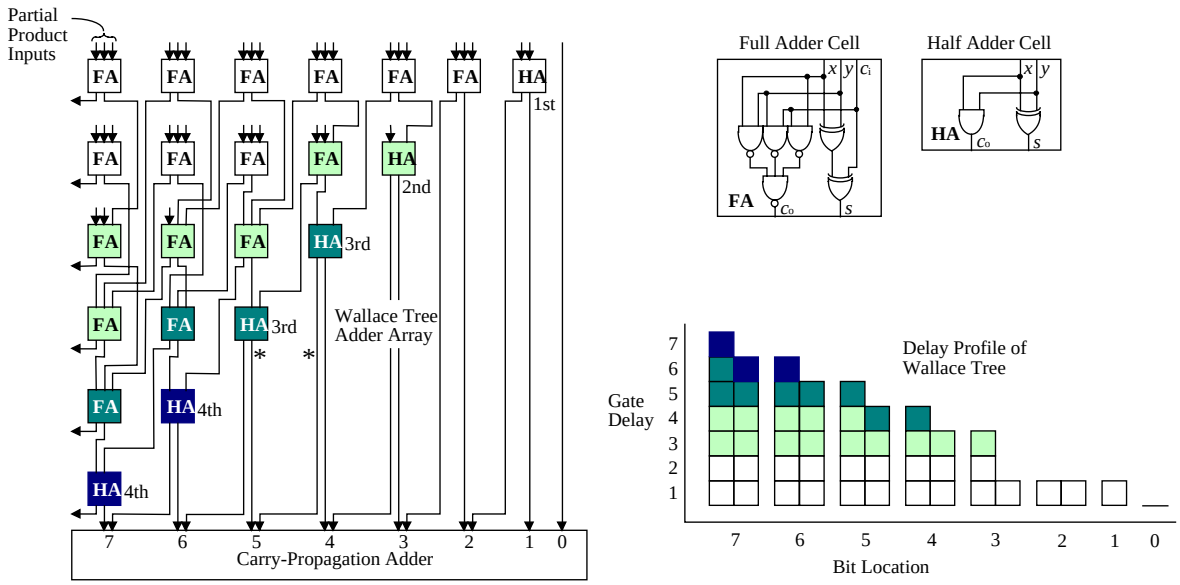

Figure 11. MARS multiplier using Wallace tree and its delay profile (n = 8).

Since the delay of the final carry-propagation adder is an addendum to the Wallace tree delay, the adder should have the optimized carry-propagation path for the tree delay profile. At the same time, we should consider the adder structure to determine the tree interconnection. Figure 12 shows the carry-propagation path in high-speed adders with equal and non-equal input signal arrival profiles. Both adders are identical to the one shown in Figure 9. The adder exhibits the best performance for the equal input profile (a). In case (b), the carry skipping over the adder blocks, though carry generator $C_{GEN,}$ has to wait until the carries propagates from the ripple-carry adder blocks. This is due to the slow input signals. In other words, an adder which is faster than the input delay slope is not needed. A simple ripple-carry adder can run fast enough in this case. The input signals at bit 4, 8 and 9 arrive very quickly, but these fast inputs also waste time waiting for the carry propagation from the next adder cells. To optimize performance of the multiplier, we have to make the positive delay slope gentle, and make the top of the hill as low as possible in the Wallace tree. This is achieved by optimizing the connection between the full adder and half adder gates according to their pin-to-pin internal delay profiles.

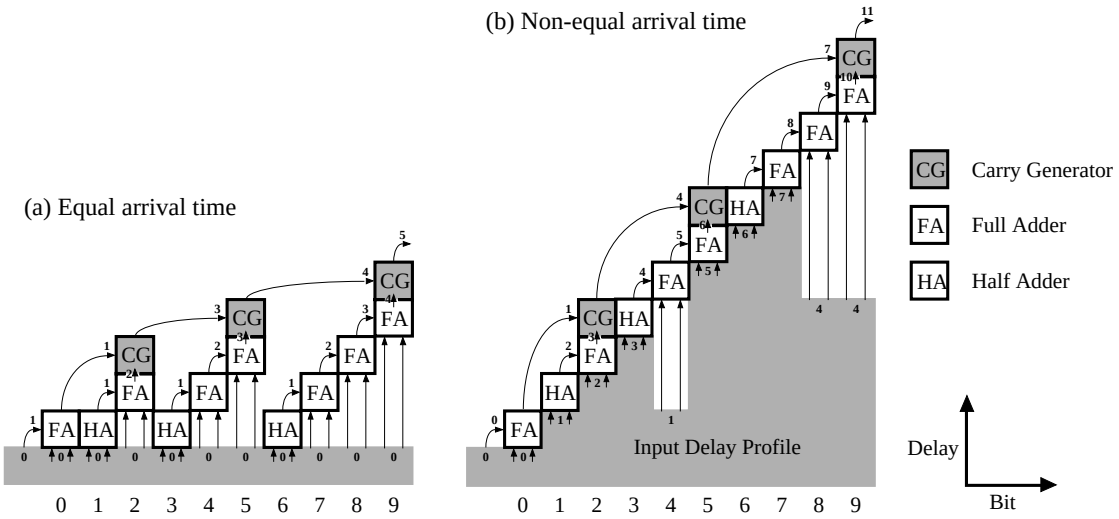

Figure 12. Carry propagation in high-speed adder on equal and non-equal input signal arrival profiles.

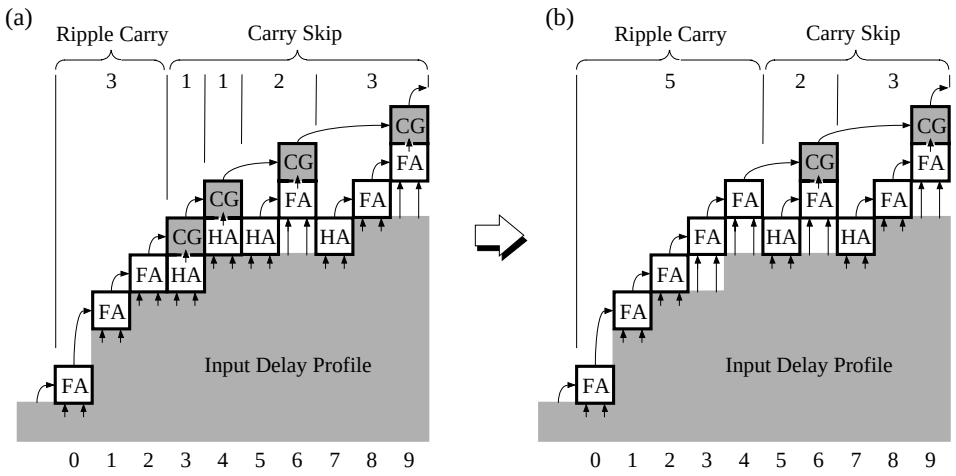

Figure 13. Adder selection over input delay profile.

Figure 13 shows an example adder structure with a positive delay slope profile. From bits 0 to 2, the slope is steeper than one FA delay, so a ripple carry adder is chosen for this part. A carry skip adder with bit blocks 1-1-2-3 is used after bit 3, in example (a). The operation of one half adder HA with a carry generator C_{GEN} is identical to that of one full adder FA, so they are replaced in example (b) to simplify the structure.

In Figures 12 and 13, the input delay time in each bit location is defined by a multiple of one adder cell delay, thus it is not difficult to optimize the adder structure. Actually, as shown in Figure 14(a), there is slack time between the input signals C and P into C_{GEN} at the 9th bit. In case (a), C is generated earlier than P, and waits for the arrival of P at C_{GEN}. When we move the location of C_{GEN} one bit left (to bit 8) so that the carry C does not waste time, the signal P has to wait instead. It is not clear which choice is better until the final adder cell is placed, and we have to choose the right combination of bit locations where C_{GEN}s are placed. In case (a), the output signal delay from the C_{GEN} is longer than that of case (b), but the carry C reaches a higher bit. We should keep the slope of the carry path over the adder blocks as gentle as possible. Therefore, the Wallace tree generator should employ a structure that has smaller value of delay/bit shown as the slope of triangles in the figure. If the structures (a) and (b) have the same slope, then we chose the former because it has higher probability to have fewer C_{GEN} cells.

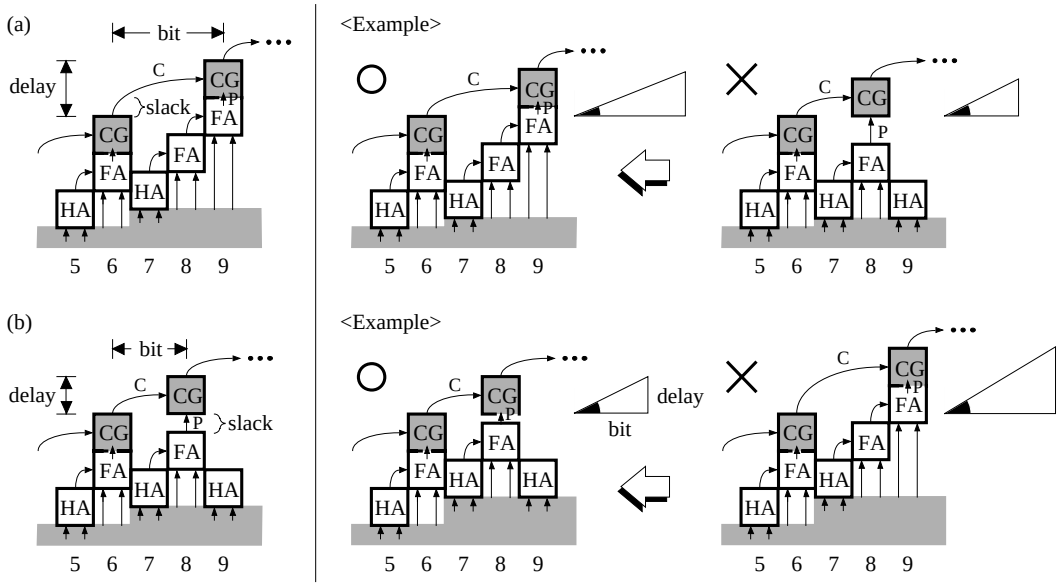

Figure 14. Carry propagation block design.

Figure 15 shows the actual delay profile of the MARS multiplier using 0.18μm IBM copper CMOS technology under nominal (V_{DD}=1.8V, Temp=25°C and L_{eff}=0.11μm) and worst case (V_{DD}=1.65V, Temp=125°C and L_{eff}=0.14μm) conditions. The output delay from the Wallace tree has an almost perfect gentle positive slope. The delay line that looks like a saw blade shows the ripple carry adder blocks. The carry skips over them smoothly. As a result, using the techniques described in this chapter realize a 2.32ns (nominal) to 3.41ns (worst) operation for the 32-bit MARS multiplier with a compact size of 3.2Kgates.

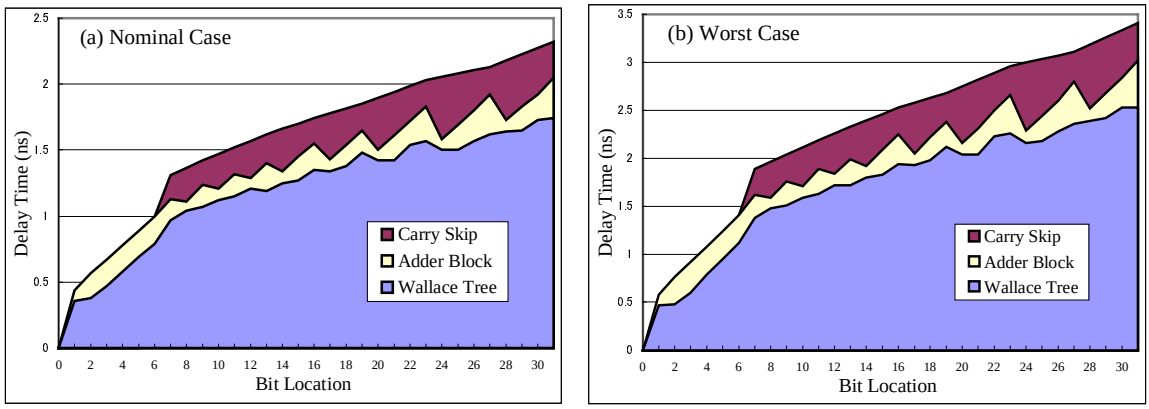

Figure 15. Actual delay profile of the MARS multiplier.

4. Performance Evaluation

We designed MARS hardware including an external interface, the control logic, and the calculation core. All the design files are written in VHDL 93. We synthesized the design using a 0.18μm IBM copper CMOS standard cell technology and evaluated its performance and size.

Table 1 shows the gate size of the logic where one gate is the size of a 2-input NAND. The memory area for the key register and S-box is shown in Table 2. We get data throughputs of 128bits / 34cycles for CBC encryption and 128bits / 18cycles for all decryption and other encryption modes, assuming a four-port SRAM implementation. However, the area of a four-port SRAM becomes larger than that of the logic part. If we do not need the high data rate, the area can be greatly reduced by using fewer-port SRAMs and a mask ROM. A two-port SRAM (one for read and one for write) for the key register halves the memory area while adding only one additional cycle for one 128-byte block encryption process. If the S-box is implemented with a single-port memory or a ROM, the cycles for the forward/backward mixing increase to 34, then the throughput of all encryption and decryption modes becomes 128 bits / 50 cycles.

The critical path delays in the forward/backward mixer and the cryptographic core under nominal and worst case conditions are shown in Figure 16. The longer delay of the cryptographic core, 5.57ns, determines the operation frequency of 180MHz (= 1 / 5.57ns) (122MHz worst case). As a result, we get a maximum data throughput of 677Mbit/s (459Mbits/s worst case) for CBC encryption and 1.28Gbit/s (867Mbits/s worst case) for other modes. All decryption modes achieve maximum throughput of 1.28Gbit/s (867Mbits/s worst case). The throughput and gate sizes for other memory implementations are summarized in Table 3.

Table 1. Logic area

Circuit Block	Gate Size
Key Expansion	2.2K
Enc/Dec Controller	4.5K
Enc/Dec Data path	6.1K
Interface + Memory Controller	1.0K
Total	13.8K

Table 2. Memory area

Function	Type	Gate Size
Key Register (256bytes)	3-port SRAM	6.8K
	2-prot SRAM	4.8K
S-box (2Kbytes)	4-port SRAM	46.2K
	3-port SRAM	30.8K
	1-port SRAM	15.4K
	ROM	6.3K

Table 3. Performance of each implementation

Memory Type		Total Gate Size (Mem+Logic)	Throughput			
Key	S-box		CBC Encryption		Other Encryption and All Decryption Modes	
			Nominal Case	Worst Case	Nominal Case	Worst Case
3-port	4-port	66.8K	677Mbit/s (34cycles)	459Mbit/s	1.28Gbit/s (18cycles)	867Mbit/s
3-port	3-port	51.4K	677Mbit/s (34cycles)	459Mbit/s	677Mbit/s (34cycles)	459Mbit/s
3-port	1-port	36.0K	460Mbit/s (50cycles)	312Mbit/s	460Mbit/s (50cycles)	312Mbit/s
2-port	1-port	34.0K	451Mbit/s (51cycles)	306Mbit/s	451Mbit/s (51cycles)	306Mbit/s
2-port	ROM*	24.9K	263Mbit/s (51cycles)	263Mbit/s	263Mbit/s (51cycles)	263Mbit/s

* 105MHz operation limited by ROM performance

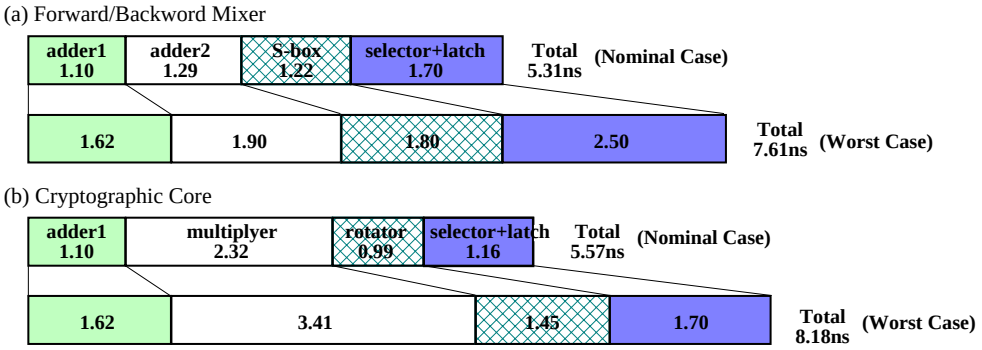

Figure 16. Critical path delay.

The technology chosen for the above estimations is a low cost copper CMOS technology several generations behind the state of the art CMOS technology. As such the performance cannot be directly compared with that of software running on today's high performance microprocessors. If built using a newer CMOS technology the performance can be expected to improve by approximately 60%.

5. Conclusion

We designed MARS hardware and estimated its size and performance. Since the MARS algorithm uses 32-bit additions and multiplications, its performance is highly dependent on the hardware design of the adder and multiplier. We designed multipliers and adders, which fully take into account the carry propagation delay. This work demonstrates that MARS can be implemented efficiently in hardware, both in terms of area and performance. We believe the design point chosen is a reasonable tradeoff of area vs. performance. We do not claim that this is the highest performance MARS design possible. Other tradeoffs may yield faster hardware implementations. Considering the size and performance in both hardware and software along with the very high security, we believe MARS is well suited to serve as the Advanced Encryption Standard algorithm.

Acknowledgement

The authors are grateful to Ms. Carolynn Burwick, Dr. Don Coppersmith, Dr. Mike Matyas, Mr. Hideto Niijima, and Mr. Nev Zunic for their reviews and comments. We also would like to thank Mrs. Akemi Ogura for supporting us generously with CAD tool operation.

References

[1] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. M. Matyas, L. O'Connor, M. Peyravian, D. Safford and N. Zunic: "MARS – a candidate cipher for AES," http://csrc.nist.gov/encryption/aes/round2/AESAlgs /MARS/mars.pdf, Aug. 1999.

[2] M. Lehman and N. Burla: "Skip Techniques for High-Speed Carry Propagation in Binary Arithmetic Units," IRE Trans. Elec. Comput., vol. EC-10, pp. 691-698, Dec. 1961.

[3] O. J. Bedrij: "Carry-Select Adder," IRE Trans. Elec. Comput., vol. EC-11, pp. 340-346, June 1962.

[4] A. Satoh, Y. Kobayashi, H. Niijima, N. Ooba, S. Munetoh, and S. Sone: "A High-Speed Small RSA Encryption LSI with Low Power Dissipation," LNCS 1396, pp. 174-187, 1997.

[5] C. S. Wallace: "Suggestion for a Fast Multiplier," IEEE Trans. Computers, vol. 13, no. 2, pp.14-17, Feb. 1964.

[6] A. Weinberger: "4:2 Carry Save Adder Module," IBM Technical Disclosure Bulletin, vol. 23, Jan. 1981.

[7] P. Song and G. De Michelli: “Circuit and Architecture Trade-Offs for High Speed Multiplication,” IEEE J. Solid State Circuits, vol. 26, no. 9, Sept. 1991.

[8] A. D. Booth: “A Signed Binary Multiplication Technique,” Quarterly J. Mechanical Applications in Math, vol. 4, part 2, pp. 236-240, 1951.

[9] V. G. Oklobdzija, D. Villeger and S. S. Liu: “A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach,” IEEE Trans. on Comp., vol. 35, no. 3, pp. 294-305, Mar. 1996.