Hardware Design and Performance Estimation of The 128-bit Block Cipher CRYPTON

Eunjong Hong, Jai-Hoon Chung, and Chae Hoon Lim

Information and Communications Research Center
Future Systems, Inc.
372-2 Yangjae-Dong, Seocho-Ku, Seoul, Korea 137-130
E-mail: {ejhong, jhoon, chlim}@ future.co.kr

Abstract

CRYPTON is a 128-bit block encryption algorithm proposed as a candidate for the Advanced Encryption Standard (AES), and is expected to be especially efficient in hardware implementation. In this paper, hardware designs of CRYPTON, and their performance estimation results are presented. Straightforward hardware designs are improved by exploiting hardware-friendly features of CRYPTON. Hardware architectures are described in VHDL and simulated. Circuits are synthesized using 0.35 μm gate array library, and timing and gate counts are measured. Data encryption rate of 1.6 Gbit/s could be achieved with moderate area of 30,000 gates and up to 2.6 Gbit/s with less than 100,000 gates.

1. Introduction

The explosive growth in computer systems and their interconnections via networks has changed the way we live. From politics to business, our lives depend on the information stored and communicated using these systems. This in turn has led to a heightened awareness of the need of the information security. To enforce information security by protecting data and resources from disclosure, a secure and efficient encryption algorithm is needed. Since the widely used encryption algorithms, DES or Triple DES, are no more considered secure enough or efficient for future applications, a new block encryption algorithm with a strength equal to or better than that of Triple DES and significantly improved efficiency is needed.

Recently very high bandwidth networking technologies such as ATM and Gigabit Ethernet are rapidly deployed [1]. Network applications such as virtual private network [2] need high-speed executions of encryption algorithms for high-speed networks. In order to utilize the high-speed networks at link speed, encryption speed as fast as 1 Giga bits per second is required. According to our experiment, the execution speed of Triple DES on Intel Pentium-II, 333MHz is only 19.6Mbps, and the highest performance of Triple DES from the commercially available encryption hardware is about 200 Mbps [3, 4].

CRYPTON [5, 6] is a 128-bit block encryption algorithm proposed as a candidate for the Advanced Encryption Standard (AES) [7]. In the evaluation performed by NIST, its software implementation on Pentium-Pro, 200MHz showed about 40Mbps, the best encryption and decryption speeds among the AES candidates' [8]. Hardware

implementations of CRYPTON are expected to be more efficient than software implementations because it was designed from the beginning with hardware implementations in mind. The encryption and decryption use the identical circuitry, and there needs no large logic for S-boxes. Moreover it does not use addition or multiplication but only uses exclusive-OR operations, and the exclusive-OR operations can be executed in parallel. CRYPTON is considered as the most hardware-friendly AES candidate on several researches [9-11].

In this paper, hardware designs of CRYPTON Version 1.0 [6], which were optimized by exploiting the hardware-friendly features of CRYPTON, are presented. To maximize operation parallelism, key generation and data encryption operations are executed simultaneously. Some round loops can be unrolled for speedup without increasing the quantity of logic. S-boxes used for both key scheduling and data encryption are shared to minimize the area. Hardware architectures are described in VHDL and simulated using Synopsys Compiler and Simulator. Circuits are synthesized using 0.35 μm gate array library, and timing and gate counts are measured.

This paper is organized as follows. In Section 2, CRYPTON algorithm is briefly introduced. In Section 3, our design considerations of CRYPTON hardware are described. In Section 4 and 5, the detailed hardware designs of CRYPTON and estimation results are presented respectively. Concluding remarks are made in Section 6.

2. CRYPTON Algorithm

In CRYPTON, each data block of 128 bits is processed in the form of a 4 × 4 byte array. The round transformation of CRYPTON consists of four steps: byte-wise substitutions, column-wise bit permutation, column-to-row transposition, and then key addition. The encryption process involves 12 repetitions of the same round transformation. The decryption process can be made identical to the encryption process with a different key schedule. Figure 1 shows the high level structure of CRYPTON.

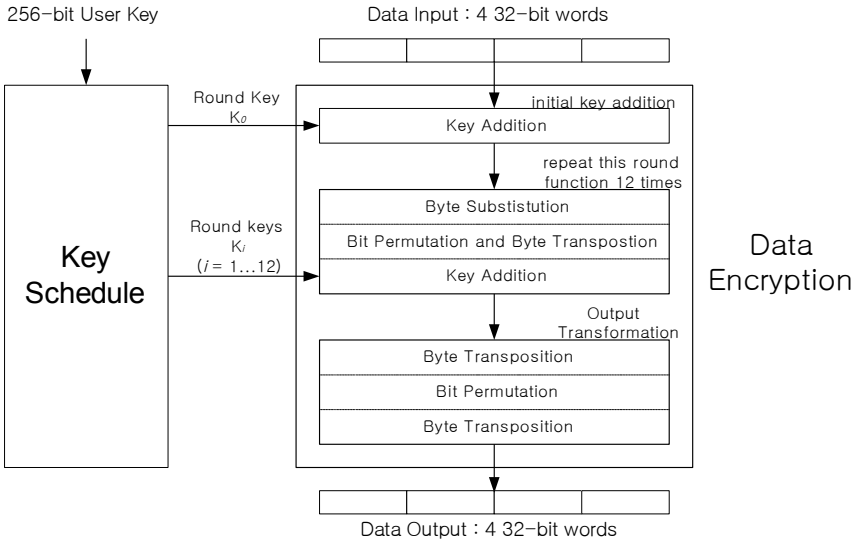

Figure 1. The Structure of CRYPTON.

The block cipher CRYPTON has the following features:

- 12-round self-reciprocal cipher (with identical processes for encryption and decryption) with block length 128 bits and key length up to 256 bits.

- Strong security against existing attacks.
- Efficiency in both SW and HW: thanks to the high degree of parallelisms and the use of only very simple operations (logical ANDs/XORs, rotates, and table lookups), CRYPTON runs very fast on most platforms in software as well as in hardware.
- Wide tradeoffs between speed and memory: It can be implemented using the storage of 32, 256, 1K, 4K, 16K or 512K bytes, etc. Its speed on Pentium Pro 200 MHz ranges from 54 Mbps (in C) to 64 Mbps (in in-line ASM).
- Fast key scheduling: The encryption key schedule runs much faster than one-block encryption, so it is very efficient for use in the application requiring frequent key changes (e.g., in hashing mode).

2.1 Basic Building Blocks

2.1.1 Byte Substitution γ

CRYPTON uses a nonlinear byte substitution using four 8 × 8 S-boxes, S_i ($0 \le i \le 3$). These S-boxes are derived from one 8 × 8 involution S-box S (i.e., $S = S^{-1}$) and satisfy the inverse relationships $S_2 = S_0^{-1}$ and $S_3 = S_1^{-1}$. The involution S-box S was designed to enable efficient logic implementation (see [6] for details).

The S-box transformation γ consists of byte-wise substitutions on a 4 × 4 byte array. Two different transformations are used alternatively in successive rounds: γ_o in odd rounds and γ_e in even rounds. They are depicted in Figure 2. Observe that the four S-boxes are arranged so that the following holds for any 4 × 4 byte array A:

$$\gamma_o\ (\gamma_e(A)) = \gamma_e(\gamma_o(A)) = A.$$

This property is used to derive the identical process for encryption and decryption.

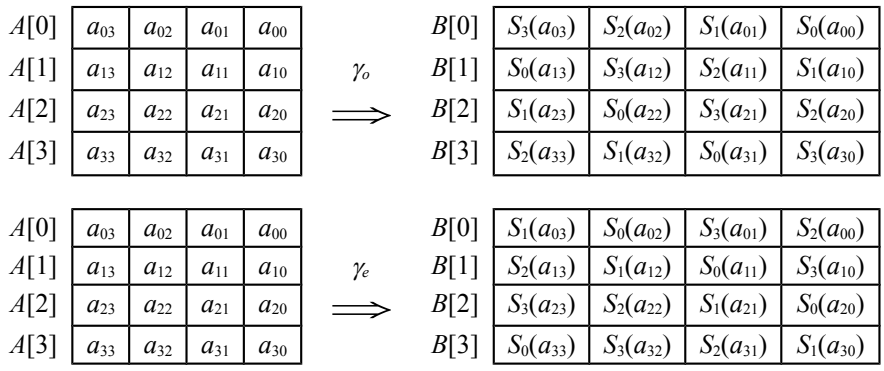

Figure 2. The Byte Substitution γ.

2.1.2 Bit Permutation π and Byte Transposition τ

As linear transformations, CRYPTON uses a sequence of bit permutation and byte transposition. The bit permutation π mixes each byte column in 4 × 4 byte array and the byte transposition τ transposes the resulting byte columns into byte rows.

Let us first define four masking vectors (M_3, M_2, M_1, M_0) for bit extraction used in π as

$$M_0 = m_3 \,||\, m_2 \,||\, m_1 \,||\, m_0 = \texttt{0x3fcff3fc},$$
$$M_1 = m_0 \,||\, m_3 \,||\, m_2 \,||\, m_1 = \texttt{0xfc3fcff3},$$
$$M_2 = m_1 \,||\, m_0 \,||\, m_3 \,||\, m_2 = \texttt{0xf3fc3fcf},$$
$$M_3 = m_2 \,||\, m_1 \,||\, m_0 \,||\, m_3 = \texttt{0xcff3fc3f},$$

where m_0 = 0xfc, m_1 = 0xf3, m_2 = 0xcf, m_3 = 0x3f. To make the encryption and decryption processes identical, two slightly different versions of a bit permutation is used; π_o in odd rounds and π_e in even rounds. They are defined as follows:

- Bit permutation π_o for odd rounds: $B = \pi_o(A)$ defined by

$$B[0] \leftarrow (A[3] \wedge M_3) \oplus (A[2] \wedge M_2) \oplus (A[1] \wedge M_1) \oplus (A[0] \wedge M_0),$$
$$B[1] \leftarrow (A[3] \wedge M_0) \oplus (A[2] \wedge M_3) \oplus (A[1] \wedge M_2) \oplus (A[0] \wedge M_1),$$
$$B[2] \leftarrow (A[3] \wedge M_1) \oplus (A[2] \wedge M_0) \oplus (A[1] \wedge M_3) \oplus (A[0] \wedge M_2),$$
$$B[3] \leftarrow (A[3] \wedge M_2) \oplus (A[2] \wedge M_1) \oplus (A[1] \wedge M_0) \oplus (A[0] \wedge M_3).$$

- Bit permutation π_e for even rounds: $B = \pi_e(A)$ defined by

$$B[0] \leftarrow (A[3] \wedge M_1) \oplus (A[2] \wedge M_0) \oplus (A[1] \wedge M_3) \oplus (A[0] \wedge M_2),$$
$$B[1] \leftarrow (A[3] \wedge M_2) \oplus (A[2] \wedge M_1) \oplus (A[1] \wedge M_0) \oplus (A[0] \wedge M_3),$$
$$B[2] \leftarrow (A[3] \wedge M_3) \oplus (A[2] \wedge M_2) \oplus (A[1] \wedge M_1) \oplus (A[0] \wedge M_0),$$
$$B[3] \leftarrow (A[3] \wedge M_0) \oplus (A[2] \wedge M_3) \oplus (A[1] \wedge M_2) \oplus (A[0] \wedge M_1).$$

Note that in hardware each bit of π outputs requires only exclusive-ORs of three different bits.

The byte transposition τ simply rearranges a 4 × 4 byte array by moving the byte at the (i, j)-th position to the (j, i)-th position (See Figure 3 for $B = \tau(A)$).

$A[0]$	a_{03}	a_{02}	a_{01}	a_{00}		$B[0]$	a_{30}	a_{20}	a_{10}	a_{00}
$A[1]$	a_{13}	a_{12}	a_{11}	a_{10}	τ	$B[1]$	a_{31}	a_{22}	a_{11}	a_{01}
$A[2]$	a_{23}	a_{22}	a_{21}	a_{20}	$\Longrightarrow$	$B[2]$	a_{32}	a_{22}	a_{12}	a_{02}
$A[3]$	a_{33}	a_{32}	a_{31}	a_{30}		$B[3]$	a_{33}	a_{23}	a_{13}	a_{03}

Figure 3. The Byte Transposition τ.

2.1.3 Key Addition σ

Key mixing is performed by simply exclusive-ORing round keys with data words. That is, for a round key $K = (K[3], K[2], K[1], K[0])^t$, $B = \sigma_K(A)$ is defined by $B[i] = A[i] \oplus K[i]$ for i = 0, 1, 2, 3.

2.1.4 Round Transformation ρ

One round of CRYPTON consists of applying in sequence S-box transformation, bit permutation, byte transposition and key addition. That is, the encryption round functions used for odd and even rounds are defined (for round key K) by

$$\rho_{oK} = \sigma_K \circ \tau \circ \pi_o \circ \gamma_o \qquad \text{for } r = 1, 3, \text{ ... etc.},$$
$$\rho_{eK} = \sigma_K \circ \tau \circ \pi_e \circ \gamma_e \qquad \text{for } r = 2, 4, \text{ ... etc.}$$

2.2 Encryption and Decryption

The encryption transformation E_K of r-round CRYPTON under key K consists of an initial key addition and $r/2$ times repetitions of ρ_o and ρ_e and then a final output transformation. Encryption and decryption can be performed by the same code (logic) if different round keys are applied.

2.3 Key Scheduling

r-round CRYPTON requires total $4(r + 1)$ round keys each of 32-bit length. These round keys are generated from a user key of $8k$ (k = 0,1,...,32) bits in two steps: first nonlinear transform the user key into 8 expanded keys and then generate the required number of round keys from these expanded keys using simple operations. This two-step generation of round keys is to allow efficient on-the-fly round key computation in the case where storage requirements do not allow storing the whole round keys.

For details of CRYPTON algorithm, please refer to [6].

3. Design Considerations

3.1 Parallel Execution of Key Generation and Encryption

Some block ciphers pre-compute round keys and store them. Then the stored keys are used repeatedly while encrypting. This style needs extra cycles for key setup, and large storage if all 12 round keys of 128-bit length are to be stored. However, CRYPTON can generate keys simultaneously with encryption. The time it takes to generate a round key of CRYPTON is insignificant compared to the time it takes for a round transformation, and this makes it possible to generate round keys with the encryption proceeding. The parallel operation of hardware CRYPTON is illustrated in Figure 4.

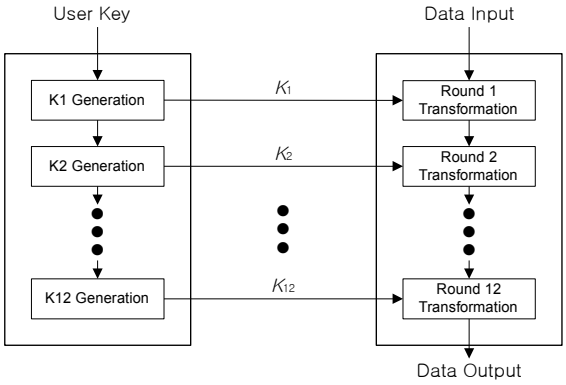

Figure 4. Concurrent Operation of Hardware CRYPTON.

3.2 Loop Unrolling

Because CRYPTON consists of 12 repetitions of round transformation, iteration is an inevitable choice for small area designs. Rather than building every round transformation separately, a data flow is made to pass the same hardware block repeatedly. We can build the whole encryption with only a small, basic building block by exploiting iteration. The block diagram for 12-cycle iteration is shown in Figure 5(a).

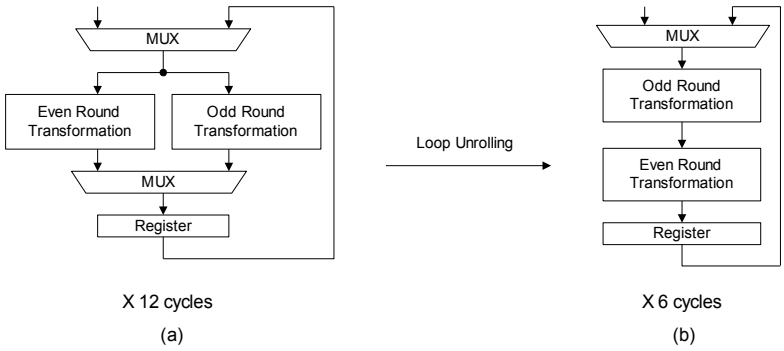

Figure 5. Loop Unrolling.

Although iteration results in small area designs, it accompanies additional path delay taken from multiplexer and register. To reduce the number of pass through multiplexer and register, we can unroll the loop so that one cycle contains double or many times of the round transformation logic. In Figure 5(b), two round transformations are performed in a cycle, but the number of components included in the design is one multiplexer less than that of Figure 5(a). We can find that the design in Figure 5(b) has better speed and area than the design in Figure 5(a). CRYPTON has a good tradeoff between speed and area when 2, 3, 4, 6, or 12 rounds are concatenated in a loop.

3.3 Common Logic Sharing

CRYPTON has only a few simple operations as its components. Because the components appear in several different parts of the algorithm, we can make those parts share a common logic block rather than building many separate logic blocks. Common logic sharing will contribute to area reduction only if the common components have smaller area than the multiplexer added. Otherwise it will only result in increased control burden and longer path delay.

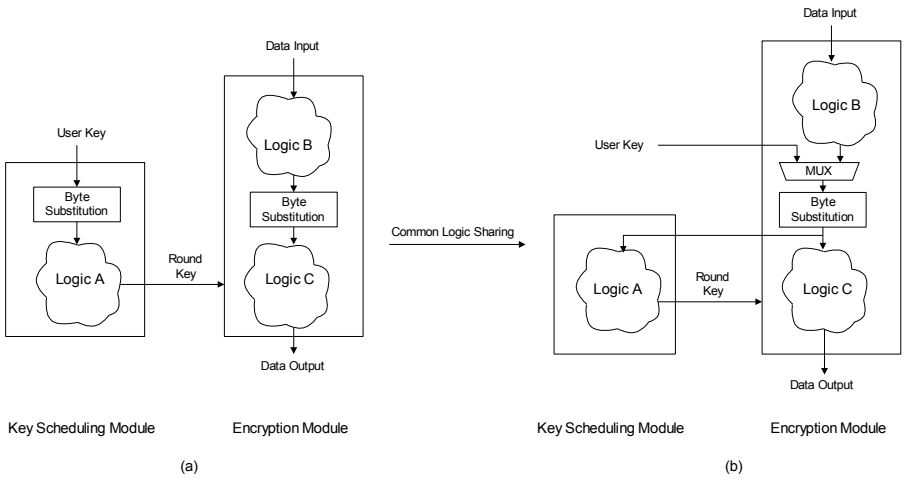

Figure 6. Common Logic Sharing.

CRYPTON has byte substitution operation both in key scheduling module and encryption module. Because byte substitution charges a significantly larger area than 128 2:1 multiplexers, there is a benefit of adopting common logic sharing as shown in Figure 6.

4. Hardware Design of CRYPTON

In this section we propose two hardware designs of CRYPTON.

4.1 Two-Round Model

Two-round model performs two successive transformations for even and odd rounds within a clock cycle. Two-round model is efficient in area-speed tradeoff as we saw in Section 3.2, and it has smaller area than designs with 3, 4, 6, or 12 rounds concatenated.

Two-round model consists of three modules: encryption module, key scheduling module, and control module as shown in Figure 9. The following scheme describes how two-round model works.

(1) Load 128-bit input data at “DATA_IN” port, and 256-bit user key at “USER_KEY” port.
(2) If decryption is to be performed, apply logic high on “DECR” port until the encryption is finished.
(3) Start encryption by applying logic high pulse for one clock cycle at the “START” port.
(4) Check the output port “DONE” to see if the encryption is completed.
(5) If “DONE” port outputs logic high, read the encryption result through output port “DATA_OUT”.

The 12-round encryption process needs 13 round keys; from the round 0 key to the round 12 key. Generating 13 round keys will take 7 clock cycles because two-round model computes two keys within a clock cycle. Thus the result is available 7 clock cycles later after logic high on “START” port is latched at the rising edge of clock.

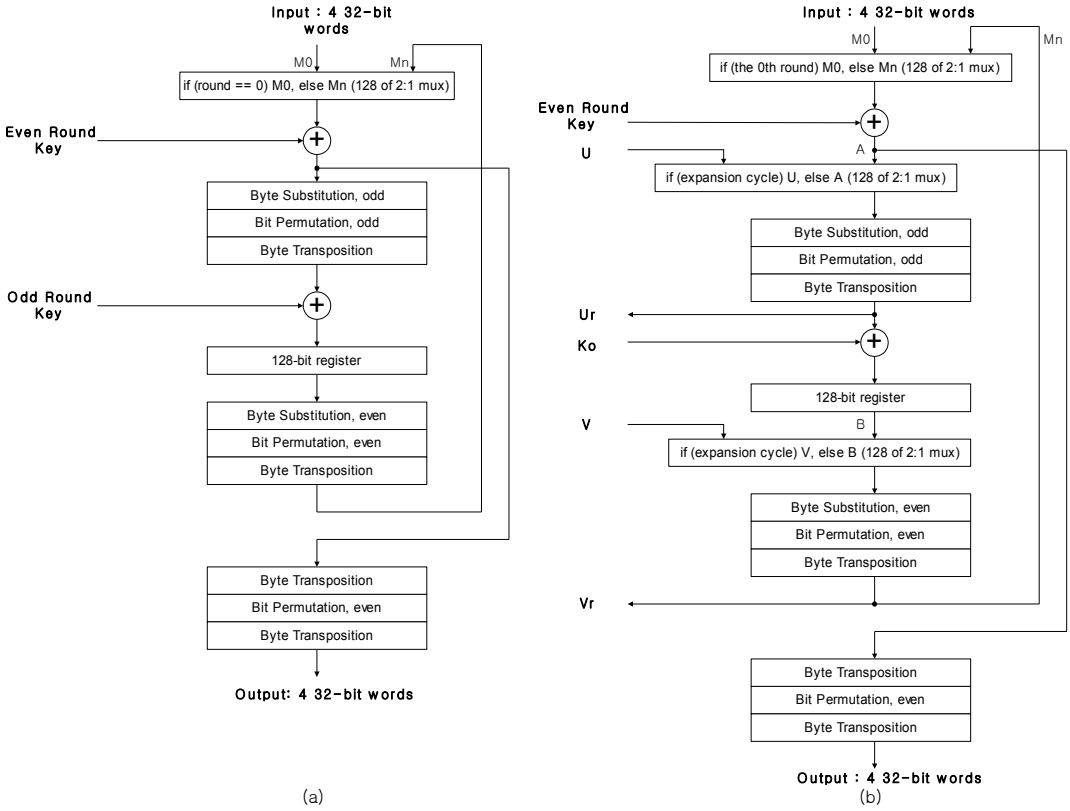

Figure 7. Encryption Module of Two-Round Model

Figure 7(a) shows the encryption module of two-round model. In encryption module, two blocks for even round transformation and odd round transformation are cascaded serially, but the sequence of blocks in Figure 7(a) is somewhat different from that of Figure 1. Because round 0 has uniqueness that only key addition is performed in it, it will require extra 128 two-input exclusive-OR gates if we didn't take the sequence shown in Figure 7(a).

The block diagram for key scheduling module is depicted in Figure 8. Key scheduling module conveys two successive round keys to the encryption module at every clock cycle. One of our design principles is parallel execution mentioned in Section 3.1, and we don't adopt any round-key pre-computation style for speed enhancement.

There is design concern in the final stage of round-key computation. A round key is exclusive-OR sum of a round-constant, masking constants, and expanded keys. There are only 4 masking constants that can be easily built into combination logic, and expanded keys are unknown before the round, thus we have no design choices about them. But there are as many as 13 round-constants of 32-bit length, and one should decide if he will build wired logic out of pre-computed round-constants, or make the design compute the constants from the specified equation at each clock cycle. Computation of constants in fully parallelized design like Figure 8 needs at least 4 32-bit adders and two 32-bit registers. On the other hand using wired logic for the 13 constants introduces a little longer propagation delay in key computation. Because two-round model aims at small area, implementation with wired logic is chosen.

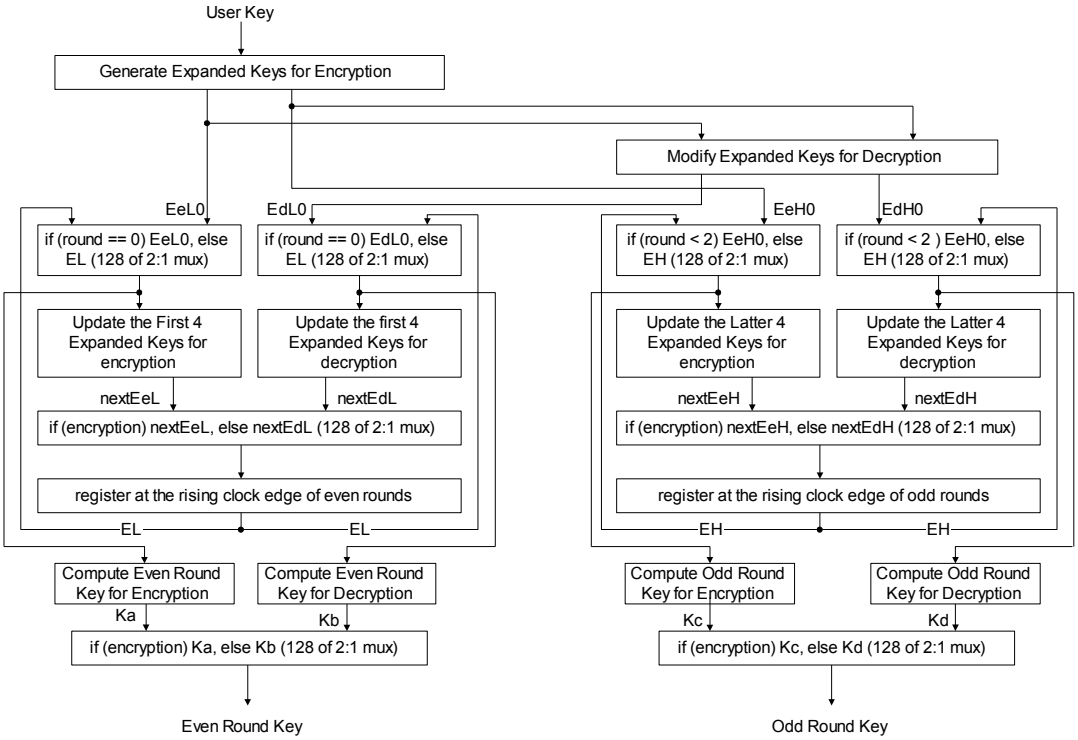

Figure 8. Key Scheduling Module of Two-Round Model.

The most unwieldy part in CRYPTON might be the nonlinear substitution, γ. γ consists of byte-wise substitution on a 4×4 byte array, in other words 16 256-entry tables. As Figure 7(a) shows, encryption module must have two γ's because γ is distinct for odd rounds and even rounds. In addition, key scheduling module has two γ's for user key expansion, which is used only once when a new user key is set. Seeing that most of the other operations takes comparatively small area, incorporating separate γ's for key expansion looks very mismatched for its rare usage. This lack of balance can be corrected by sharing γ's between the expansion block of key scheduling module and round transformation block of encryption module. This scheme effectively reduces the total area but needs one more clock cycle only for key expansion when new expanded keys are to be computed. Figure 7(b) shows the new block diagram for encryption module with γ sharing. In Figure 7(b), outputs "Ur" and "Vr" are fed to the inputs of two 128-bit registers and stored in them. Those stored values in the registers are used repeatedly for generation of round keys unless a new user key is set. The key scheduling area also has change in expansion block. Key scheduling area takes $U'$and $V'[6]$ – the outputs of expanded key registers – as inputs, and performs operations later than γ.

Combining encryption module and key scheduling module, the whole encryption block is built as shown in Figure 9. In Figure 9, "Cycle0" is a signal indicating the key expansion cycle, and "Cycle1" tells the first cycle among 7 is going on. Both signals are used as control inputs for multiplexers.

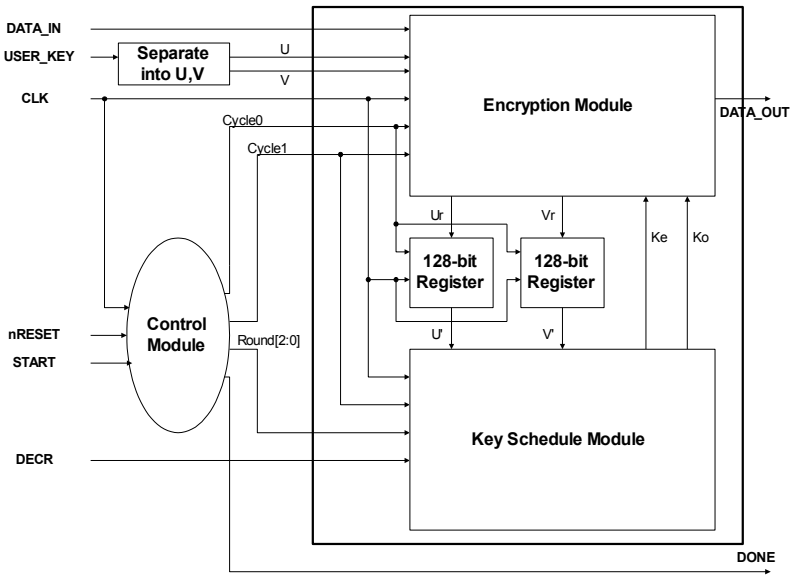

Figure 9. Hierarchical View of Two-Round Model.

4.2 Full-Round Model

To make a fast design, pipelining or loop unrolling can be generally applied. In pipelining, the algorithm is partitioned into several stages, and this enables several data blocks to be encrypted simultaneously. This is possible in ECB mode because each result of block encryption is independent from others. But modes except ECB require the previous result of encryption to be available to complete the present encryption, which makes pipelining useless. Since most of the recent application of block ciphers use chaining or feedback mode, speed enhancement through pipelining is not considered here. Instead we make full-round model with 12 rounds fully unrolled. This full-round model computes 12 rounds of transformation without looping, and it is the fastest but the largest design among those exploiting loop-unrolling. The loop unrolling of 4 or 6 will have just intermediate values of area and speed between those of two-round model and full-round model. Block diagrams for full-round model are straightforward and shown in Figure 10, Figure 11 and Figure 12.

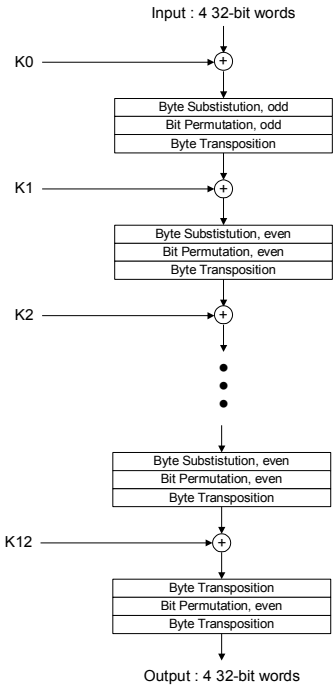

Figure 10. Encryption Module of Full-Round Model.

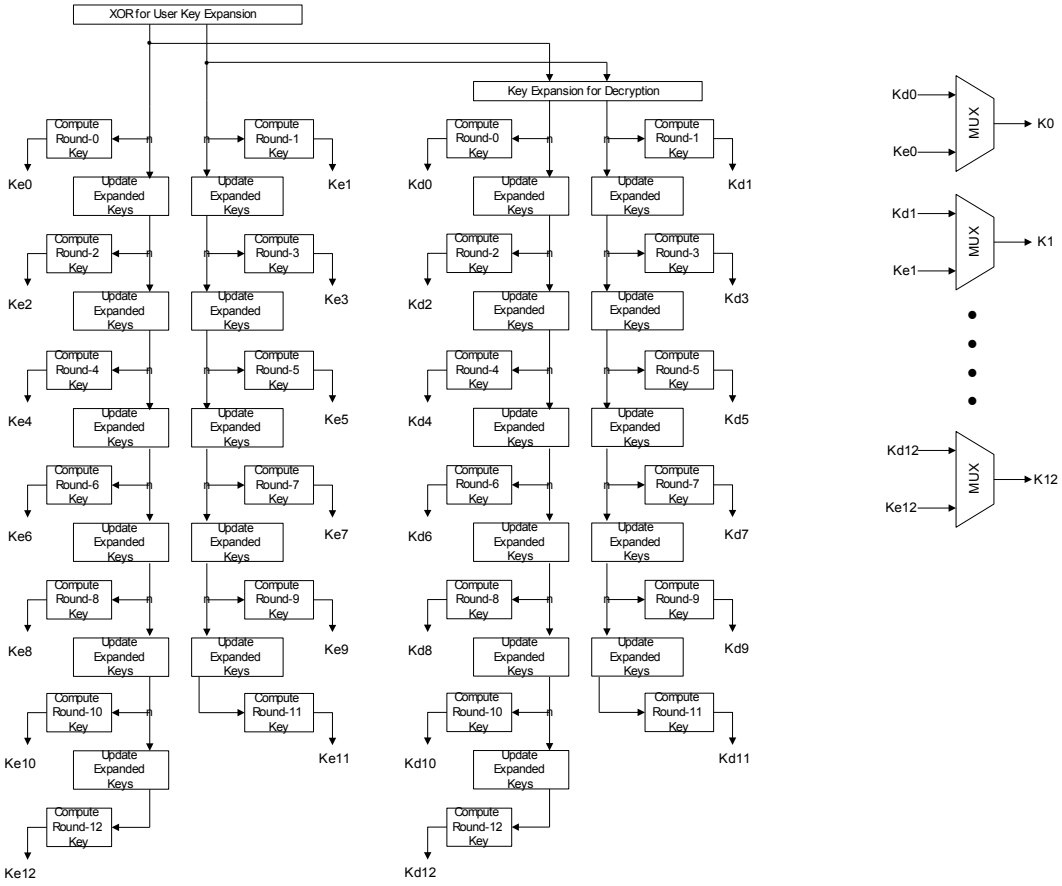

Figure 11. Key Scheduling Model of Full-Round Model.

In key scheduling module of Figure 11, round-key output Kn for the nth round will be Ken if encryption is performed, and Kdn if decryption is performed. To achieve high speed, common logic sharing in expansion block is not adopted here. Full-round model tells us area cost of the nearly pure algorithm because it doesn't need any registers or control unit. But the 12 multiplexers in key scheduling area could not be removed.

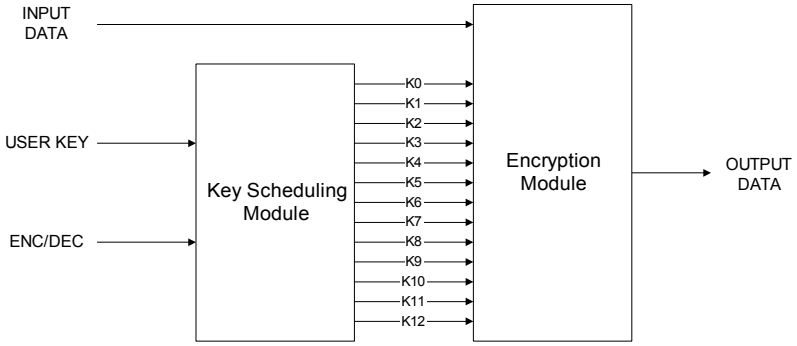

Figure 12. Hierarchical View of Full-Round Model.

5. Estimation Results

Our estimations are all based on Synopsys DesignCompiler and Hyundai 0.35 μm gate array library. Table 1 shows area and speed of two-round model and full-round model each with two speed-area tradeoffs. In this paper, all estimations are for typical cases. "Gate" means a 2-input NAND gate, equivalent to 4 transistors.

Table 1. Speed and Area Estimations of Hardware CRYPTON.

Estimated items		Two-round model		Full-round model	
		Area critical	Speed critical	Area critical	Speed critical
Gate count (Cell Area)	total gates	18,322	28,179	46,259	93,929
	encryption module	8,267	17,958	33,598	74,857
	key scheduling module	7,930	8,078	12,661	19,072
Minimum clock period (T_{clk})		18.97 ns	10.23 ns	74.03 ns	44.30 ns
Key setup time		on the fly	on the fly	10.13 ns	7.91 ns
Time to switch keys		18.97 ns	10.23 ns	10.13 ns	6.13 ns
Time to encrypt one block		132.79 ns ($7 \times T_{clk}$)	71.61 ns ($7 \times T_{clk}$)	74.03 ns	44.30 ns
Throughput		898Mbps*	1.66Gbps**	1.61Gbps**	2.69Gbps**

* 1Mbps = 1,024 × 1,024 bps = 1,048,576 bps

** 1Gbps = 1,024 × 1,024 × 1,024 bps = 1,073,741,824 bps

The two results for two-round model have identical logic design, but are different only in optimization strategies. The one indicated as "Area Critical" was optimized to reduce as much area as possible. On the other hand, the result of "Speed Critical" was obtained by minimizing the clock period. As shown in "Time to switch keys", one clock is spent on expanding the user key when the user key is switched to a new value. Since the design uses one clock source for its whole area, the time to switch keys will be at least "minimum clock period", but the actual propagation delay in key expansion is less than the minimum clock period. Round keys are

generated on the fly, and if the user keys remain same, 7 clock cycles are needed for both encryption and decryption.

The same optimization strategies were applied to full-round model. However, in speed critical optimization, the path from data input to the final output was optimized rather than minimizing the clock period as in two-round model.

Although results of full-round model were entered into the same format of table with two-round model, the numbers should not be compared directly because the two designs have different architectures. The following three explanations qualify the meaning of each item for full-round model.

- *Time to switch keys*: time from the insertion of a new user key to the generation of the round 0 key (this is because time taken to compute the next round key is much shorter than time to compute one round transformation. By the time γ operation of 1^{st} round has been performed, all round keys from 1^{st} round to 12^{th} round are available).
- *Key setup time*: time needed to compute the whole 12 round keys.
- *Time to encrypt one block*: The longest path delay from data input to the final data output (it is the time taken to encrypt one block when all round keys are set up and ready to be used).

In two-round model, "Total Area" is larger than the sum of "encryption module" and "key scheduling module" because area of buffer for expanded keys and control unit is missing. Full-round model doesn't have any extra logic except encryption module and key scheduling module, and thus the sum is exactly matched.

Two main issues in logic design are speed and area. Because speed and area are objects of tradeoff in most cases, we can get the best results on one criterion by optimizing for it while ignoring the other. On the other hand, the result is the worst one for the ignored criterion. We can find the approximate upper and lower bounds of speed and area of CRYPTON by once optimizing for area and then for speed.

The main trade-off between space and time takes place in optimization of S-box. We could obtain as high speed as 2.6Gbps by growing the area of S-box, but the total number of gates was over 90,000. Resorting to speed-area tradeoff, two-round model faster than full-round model was possible contrary to our initial scheme of using loop unrolling to make a fast design. In our estimation, two-round model was found more moderate and practical both in area and speed than full-round model. Although optimization in the synthesis tool was very useful to achieve a goal in speed or area, a better result was possible by modifying the design itself.

6. Conclusions

In this paper, we designed and proposed two hardware architectures of CRYPTON exploiting the inherent hardware-friendly features of CRYPTON. The architectures were described in VHDL and circuits were synthesized using 0.35 μm gate array with several speed-area tradeoffs. 0.9 Gbps with the smallest area of 18,000 gates and the fastest speed of 2.6 Gbps with less than 100,000 gates could be achieved. The speed of 2.6 Gbps is faster than the commercially available fastest Triple-DES chip with an order of magnitude. This is enough speed to support the Gigabit networks. Since CRYPTON has good scalability in gate count, a designer can select a proper speed-area tradeoff from the large set choices.

References

1. L. Geppert and W. Sweet, “Technology 1999 Analysis & Forcast – Communications”, *IEEE Spectrum*, January 1999, pp.29-34.
2. D. Kosiur, *Building and Managing Virtual Private Networks*, Wiley, 1998.
3. Analog Devices, ADSP-2141L SafeNet DSP Preliminary Technical Data Sheet, REV.PrA, February 1999.
4. VLSI Technology, VMS115 IPSec Coprocessor Data Sheet, Rev2.0, January 1999.
5. C.H. Lim, “CRYPTON: A New 128-bit Block Cipher,” *Proceedings of the First Advanced Encryption Standard Candidate Conference*, (Ventura, California), National Institute of Standards and Technology (NIST), August 1998.
6. C.H. Lim, “A Revised Version of CRYPTON – CRYPTON Version 1.0,” *Proceedings of the 1999 Fast Software Encryption Workshop*, March 1999.
7. NIST, Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES), Federal Register, Vol.62, No.177, September 12, 1997.
8. M. Smid and E. Roback, “Developing the Advanced Encryption Standard,” *Proceedings of the 1999 RSA Conference*, January 1999.
9. E. Biham, “A Note on Comparing the AES Candidates,” *Proceedings of the Second Advanced Encryption Standard Candidate Conference*, (Rome, Italy), National Institute of Standards and Technology (NIST)", March 1999.
10. C.S.K. Clapp, “Instruction-level Parallelism in AES Candidates,” *Proceedings of the Second Advanced Encryption Standard Candidate Conference*, (Rome, Italy), National Institute of Standards and Technology (NIST)", March 1999.
11. B. Schneier, et. al., “Performance Comparison of the AES Submissions,” *Proceedings of the Second Advanced Encryption Standard Candidate Conference*, (Rome, Italy), National Institute of Standards and Technology (NIST)", March 1999.