Efficient Hardware Implementation of LEA 논문 리뷰

https://youtu.be/S4h_96Ay3hw

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

Efficient Hardware Implementation of the Lightweight Block Encryption Algorithm LEA

Article

Efficient Hardware Implementation of the Lightweight Block Encryption Algorithm LEA

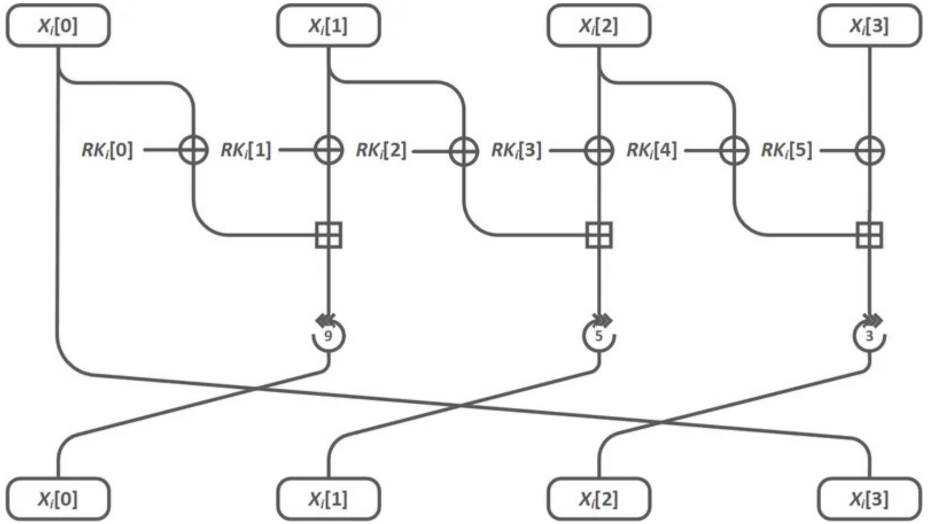
Donggeon Lee [1,*], Dong-Chan Kim [2], Daesung Kwon [2] and Howon Kim [1]

- LEA 알고리즘의 3가지 최적화 설계를 제안
  1. 속도 최적화
  2. 면적 최적화 1
  3. 면적 최적화 2

- LEA의 암호화 과정은 단순한 XOR, Addtion, Rotation으로 설계되어 있음.
  - 여기에서는 키 확장과 암호화 과정을 합쳐서 설계하였는데, 키 확장 과정 설계의 차이가 있음.

1.LEA

- 암호화 과정은 단순하고 키 길이에 영향을 받지 않음.
- 키 확장은 키 길이에 따라서 사용되는 상수 값과 라운드키로 저장되는 값의 차이가 있음



$$\delta_0 = C3EFE9DB_{16}, \delta_1 = 44626B02_{16}$$
$$\delta_2 = 79E27C8A_{16}, \delta_3 = 78DF30EC_{16}$$
$$\delta_4 = 715EA49E_{16}, \delta_5 = C785DA0A_{16}$$
$$\delta_6 = E04EF22A_{16}, \delta_7 = E5C40957_{16}$$

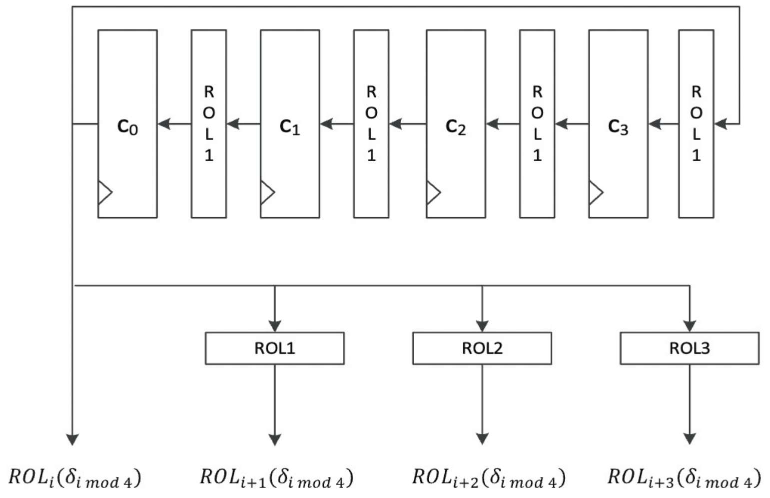
$$T_0^{i+1} \leftarrow ROL_1(T_0^i \boxplus ROL_i(\delta_{i \mod 4}))$$
$$T_1^{i+1} \leftarrow ROL_3(T_1^i \boxplus ROL_{i+1}(\delta_{i \mod 4}))$$
$$T_2^{i+1} \leftarrow ROL_6(T_2^i \boxplus ROL_{i+2}(\delta_{i \mod 4}))$$
$$T_3^{i+1} \leftarrow ROL_{11}(T_3^i \boxplus ROL_{i+3}(\delta_{i \mod 4}))$$
$$RK^i \leftarrow (T_0^i, T_1^i, T_2^i, T_1^i, T_3^i, T_1^i)$$

2. 속도 최적화 설계

3.1. Constant Value Schedule Logic for Speed-Optimized Implementation

LEA employs several constants for key scheduling. To design the constant schedule logic, the usage patterns of constants need to be analyzed. In Equation (5), the constant values used for the i-th round function are $ROL_i(\delta_{i \bmod 4})$, $ROL_{i+1}(\delta_{i \bmod 4})$, $ROL_{i+2}(\delta_{i \bmod 4})$, and $ROL_{i+3}(\delta_{i \bmod 4})$. At the i-th round, the $i \bmod 4$-th constant is chosen; in other words, constants are used in increasing order, *i.e.*, $\delta_0, \delta_1, \delta_2, \delta_3, \delta_0, \ldots$. After a constant is chosen, it is rotated i, $i+1$, $i+2$, and $i+3$ times to the left.

Figure 2. Constant scheduling logic structure for speed-optimized LEA hardware.



$$
\begin{aligned}
T_0^{i+1} &\leftarrow ROL_1(T_0^i \boxplus ROL_i(\delta_{i \bmod 4})) \\
T_1^{i+1} &\leftarrow ROL_3(T_1^i \boxplus ROL_{i+1}(\delta_{i \bmod 4})) \\
T_2^{i+1} &\leftarrow ROL_6(T_2^i \boxplus ROL_{i+2}(\delta_{i \bmod 4})) \\
T_3^{i+1} &\leftarrow ROL_{11}(T_3^i \boxplus ROL_{i+3}(\delta_{i \bmod 4})) \\
RK^i &\leftarrow (T_0^i, T_1^i, T_2^i, T_1^i, T_3^i, T_1^i)
\end{aligned}
$$

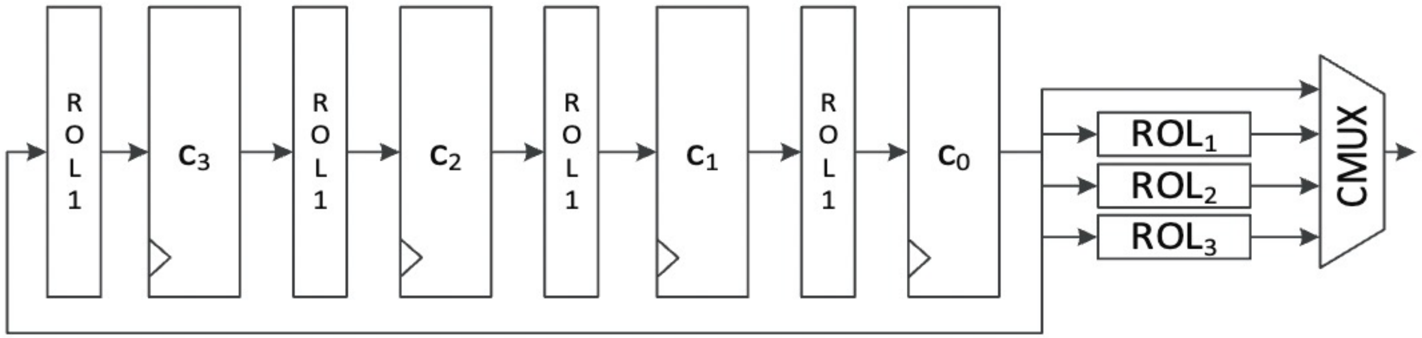
Figure 2 shows the intuitive structure of the constant schedule logic of the 128-bit speed-optimized version of LEA hardware. The speed-optimized version executes one round per clock cycle. Therefore, it should generate all four constants required for a round. Constants δ_0 to δ_3 are stored in 32-bit flip-flops c_0 to c_3. Each value in a 32-bit flip-flop moves to the next flip-flop per round. Since a constant value that is rotated i-times ($i+1$, $i+2$, and $i+3$ times) is used for the i-th round, it is rotated 1 bit left for every round. Since the constant used for the i-th round is located at the c_0 register, its value is exactly $ROL_i(\delta_{i \bmod 4})$. The remaining $ROL_{i+1}(\delta_{i \bmod 4})$, $ROL_{i+2}(\delta_{i \bmod 4})$, and $ROL_{i+3}(\delta_{i \bmod 4})$ are generated from corresponding ROL_1, ROL_2, and ROL_3 operations. In the figure, no rotation consumes any logical gates because they can be easily implemented by crossing some wires. Thus, the logic requires only 128 flip-flops.

3. 면적 최적화 설계

3.2. Constant Value Schedule Logic for Area-Optimized Implementation

To minimize the number of gates required, some logic gates are shared and iteratively used in a round. In area-optimized implementation, one round can be split into several clock cycles. Therefore, four constants must be generated one by one in a round. The intuitive structure of constant scheduling logic is depicted in Figure 3. At the beginning of a round, c_0 is fed with $ROL_i(\delta_{i \bmod 4})$ from c_1. The value is passed to the key scheduling logic through the first path of the MUX. For the remaining clock cycles of one round, $ROL_{i+1}(\delta_{i \bmod 4})$, $ROL_{i+2}(\delta_{i \bmod 4})$, and $ROL_{i+3}(\delta_{i \bmod 4})$ are fed to the key scheduling logic using the second, third, and fourth path of the MUX.

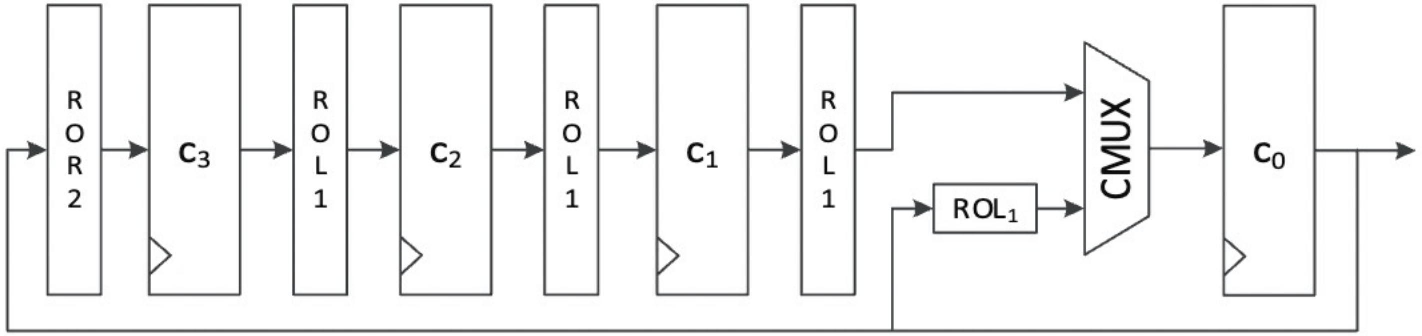
Figure 3. Intuitive constant scheduling logic structure for area-optimized LEA hardware.



3. 면적 최적화 설계

An alternative logic structure for area-optimized LEA is depicted in Figure 4. The 32-bit constant in c_0 is fed to the key scheduling logic. When the round counter is increased, the upper path of MUX is used, which leads $ROL_i(\delta_{i \bmod 4})$ at c_1 to move to the c_0 register. In a round, the remaining constant values used for the i-th round function, $ROL_{i+1}(\delta_{i \bmod 4})$, $ROL_{i+2}(\delta_{i \bmod 4})$, and $ROL_{i+3}(\delta_{i \bmod 4})$, are generated during the remaining three clock cycles using the lower path of MUX. By using this structure, the cost for the four-input MUX is reduced to that of a two-input MUX. Moreover, the rotating logic before c_3 is different from that in Figure 3. At the final state of a round, the c_0 is $ROL_i + 3(delta_{i \bmod 4})$. To make $ROL_i + 4(delta_{i \bmod 4})$ have the same value at a register after four rounds, c_0 should be rotated to the right twice. Consequently, the rotation logic before the c_3 register in Figure 3 is different from that in Figure 4.

Figure 4. Alternative constant scheduling logic structure for area-optimized LEA hardware.

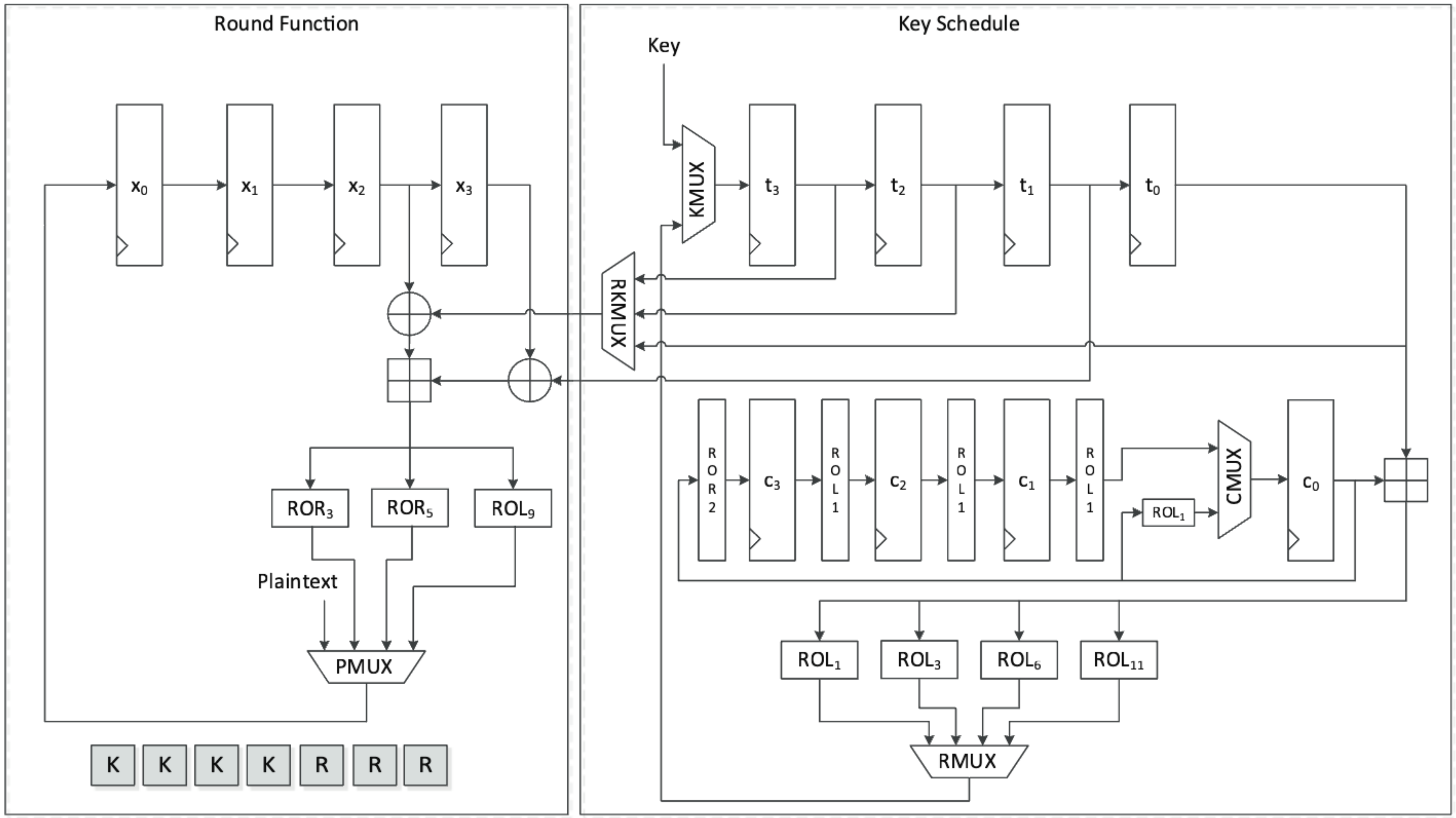


4. 전체 하드웨어 설계

$$T_0^{i+1} \leftarrow ROL_1(T_0^i \boxplus ROL_i(\delta_{i \bmod 4}))$$
$$T_1^{i+1} \leftarrow ROL_3(T_1^i \boxplus ROL_{i+1}(\delta_{i \bmod 4}))$$
$$T_2^{i+1} \leftarrow ROL_6(T_2^i \boxplus ROL_{i+2}(\delta_{i \bmod 4}))$$
$$T_3^{i+1} \leftarrow ROL_{11}(T_3^i \boxplus ROL_{i+3}(\delta_{i \bmod 4}))$$
$$RK^i \leftarrow (T_0^i, T_1^i, T_2^i, T_1^i, T_3^i, T_1^i)$$

$$X_0^{i+1} \leftarrow ROL_9((X_0^i \oplus RK_0^i) \boxplus (X_1^i \oplus RK_1^i))$$
$$X_1^{i+1} \leftarrow ROR_5((X_1^i \oplus RK_2^i) \boxplus (X_2^i \oplus RK_3^i))$$
$$X_2^{i+1} \leftarrow ROR_3((X_2^i \oplus RK_4^i) \boxplus (X_3^i \oplus RK_5^i))$$
$$X_3^{i+1} \leftarrow X_0^i$$

Figure 5. Datapath of LEA-128-AREA-1.

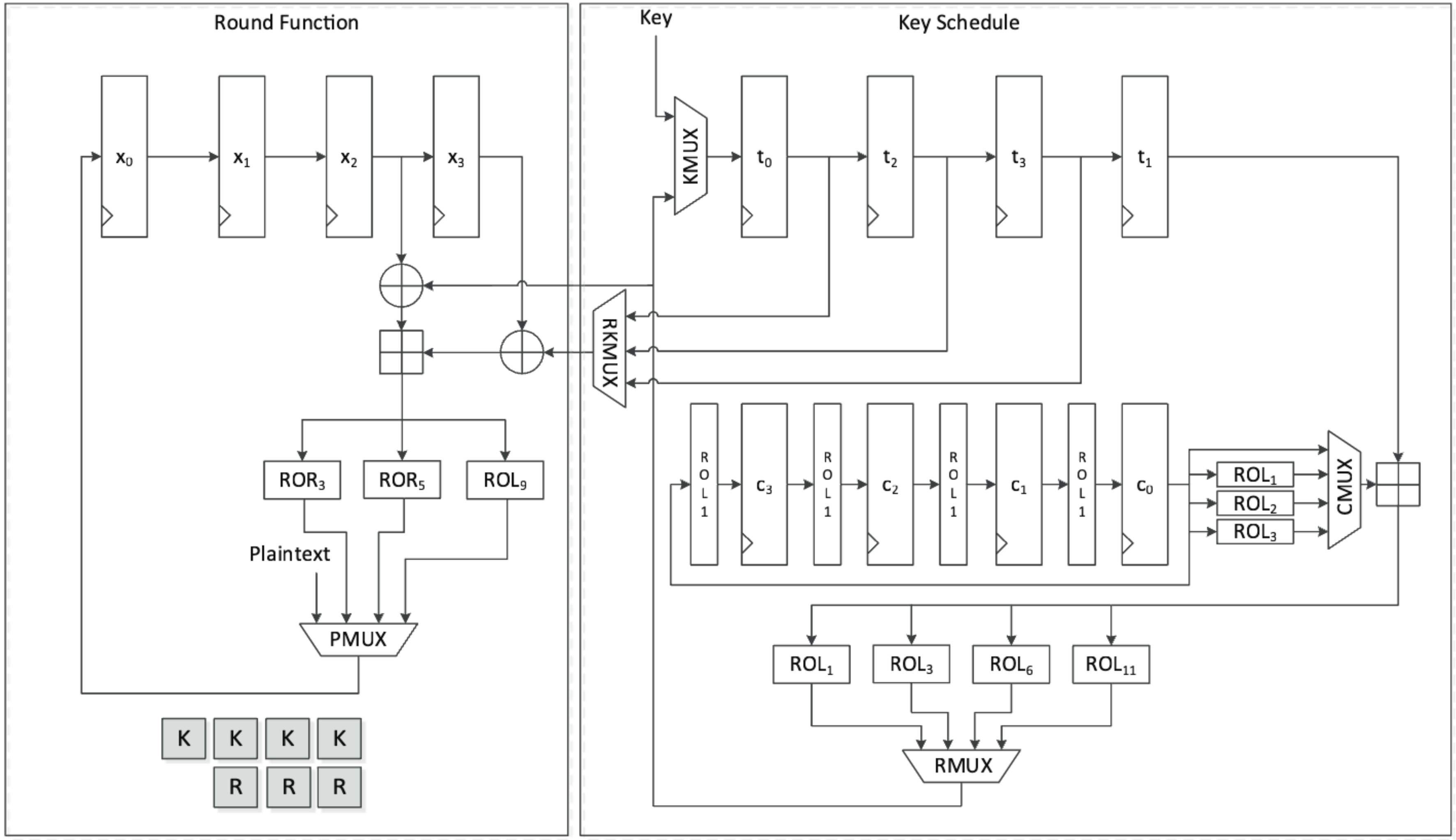


4. 전체 하드웨어 설계

$$T_0^{i+1} \leftarrow ROL_1(T_0^i \boxplus ROL_i(\delta_{i \bmod 4}))$$
$$T_1^{i+1} \leftarrow ROL_3(T_1^i \boxplus ROL_{i+1}(\delta_{i \bmod 4}))$$
$$T_2^{i+1} \leftarrow ROL_6(T_2^i \boxplus ROL_{i+2}(\delta_{i \bmod 4}))$$
$$T_3^{i+1} \leftarrow ROL_{11}(T_3^i \boxplus ROL_{i+3}(\delta_{i \bmod 4}))$$
$$RK^i \leftarrow (T_0^i, T_1^i, T_2^i, T_1^i, T_3^i, T_1^i)$$

$$X_0^{i+1} \leftarrow ROL_9((X_0^i \oplus RK_0^i) \boxplus (X_1^i \oplus RK_1^i))$$
$$X_1^{i+1} \leftarrow ROR_5((X_1^i \oplus RK_2^i) \boxplus (X_2^i \oplus RK_3^i))$$
$$X_2^{i+1} \leftarrow ROR_3((X_2^i \oplus RK_4^i) \boxplus (X_3^i \oplus RK_5^i))$$
$$X_3^{i+1} \leftarrow X_0^i$$

Figure 6. Datapath of LEA-128-AREA-2.

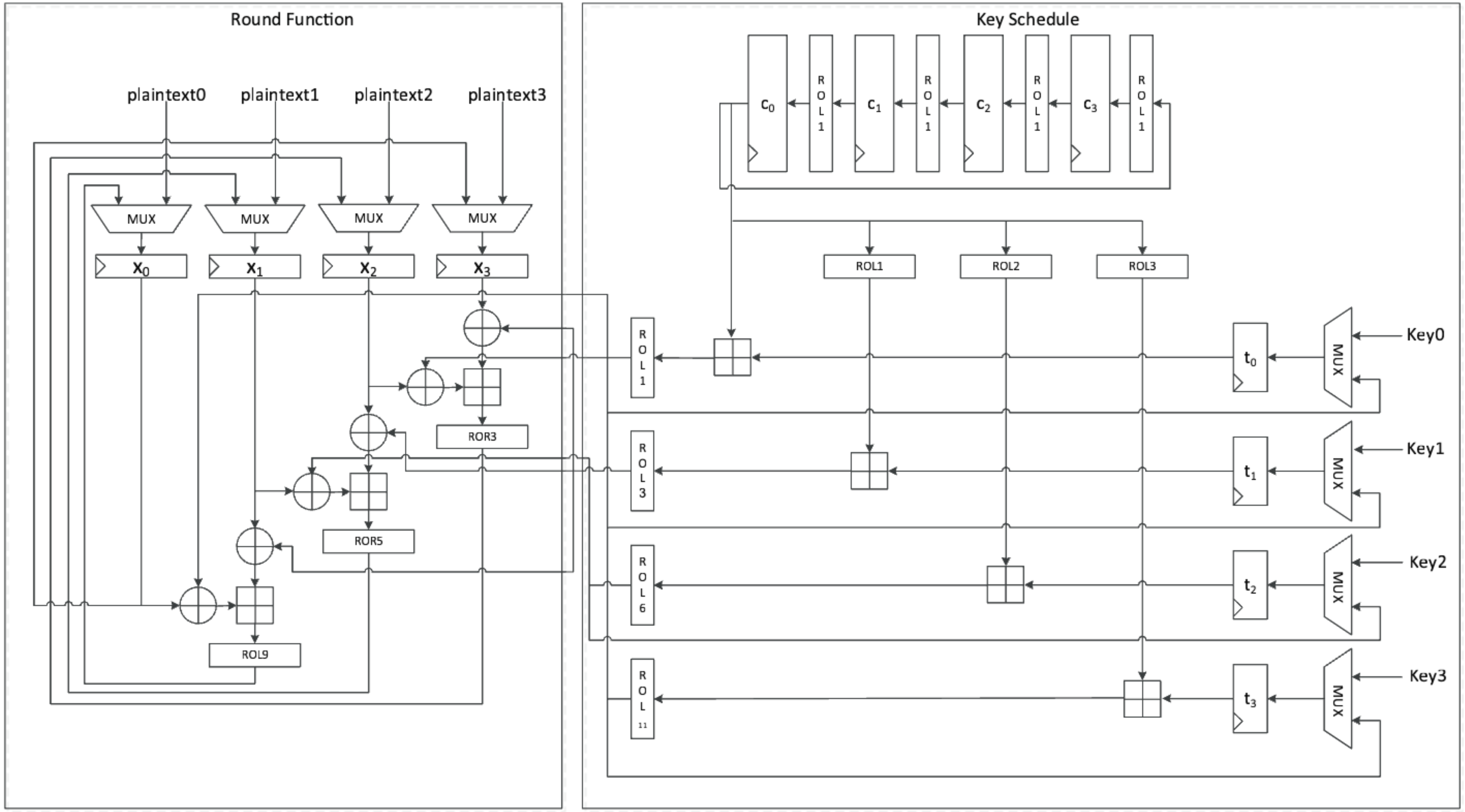


4. 전체 하드웨어 설계

$$T_0^{i+1} \leftarrow ROL_1(T_0^i \boxplus ROL_i(\delta_{i \bmod 4}))$$
$$T_1^{i+1} \leftarrow ROL_3(T_1^i \boxplus ROL_{i+1}(\delta_{i \bmod 4}))$$
$$T_2^{i+1} \leftarrow ROL_6(T_2^i \boxplus ROL_{i+2}(\delta_{i \bmod 4}))$$
$$T_3^{i+1} \leftarrow ROL_{11}(T_3^i \boxplus ROL_{i+3}(\delta_{i \bmod 4}))$$
$$RK^i \leftarrow (T_0^i, T_1^i, T_2^i, T_1^i, T_3^i, T_1^i)$$

$$X_0^{i+1} \leftarrow ROL_9((X_0^i \oplus RK_0^i) \boxplus (X_1^i \oplus RK_1^i))$$
$$X_1^{i+1} \leftarrow ROR_5((X_1^i \oplus RK_2^i) \boxplus (X_2^i \oplus RK_3^i))$$
$$X_2^{i+1} \leftarrow ROR_3((X_2^i \oplus RK_4^i) \boxplus (X_3^i \oplus RK_5^i))$$
$$X_3^{i+1} \leftarrow X_0^i$$

Figure 7. Datapath of LEA-128-SPEED.



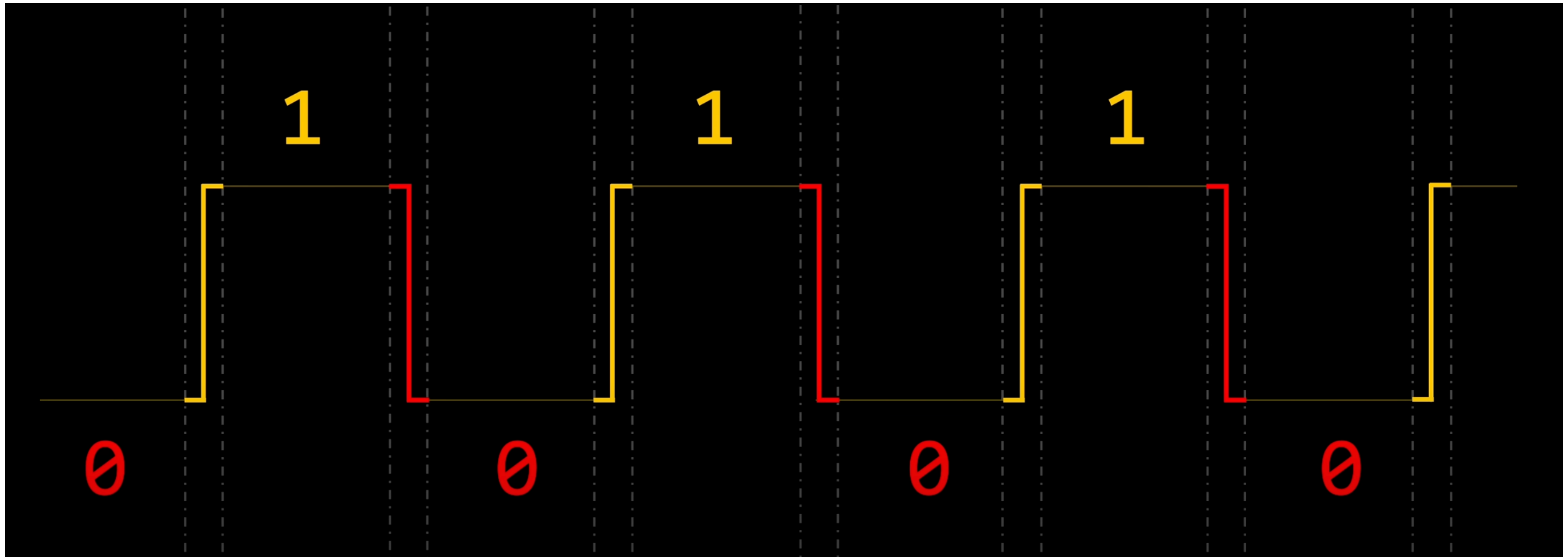
5. 성능 평가

Table 2. Comparison of implementation results using Xilinx Virtex 5.

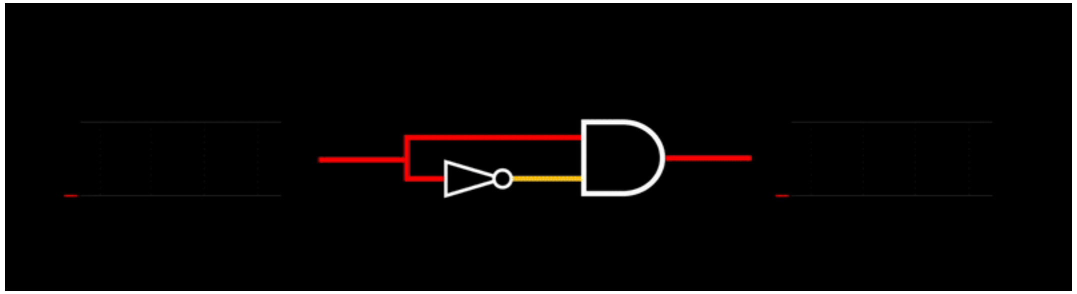
Designs	Cycles	Max. Freq.	Latency @max freq (μs)	Latency @10MHz (μs)	Throughput (Mbps)	Area (Slice Element) Reg	LUT	Total	ATP	Throughput/Area
LEA-128-AREA-1	168	269.658	0.62	16.8	205.45	392	249	503	311.86	0.41
LEA-128-AREA-2	96	163.861	0.59	9.6	218.48	388	306	559	329.81	0.39
LEA-128-SPEED	24	217.806	0.11	2.4	1,161.63	386	713	854	93.94	1.36
LEA-192-AREA-1	168	197.797	0.85	16.8	226.05	423	408	620	527	0.36
LEA-192-AREA-2	84	198.364	0.42	8.4	453.40	514	403	709	297.78	0.64
LEA-192-SPEED	28	218.250	0.13	2.8	1,496.57	508	911	1,103	143.39	1.36
LEA-256-AREA-1	288	257.652	1.12	28.8	229.02	663	713	994	1,113.28	0.23
LEA-256-AREA-2	192	169.2	1.13	19.2	225.60	649	987	1,003	1,133.4	0.22
LEA-256-SPEED	32	126.23	0.25	3.2	1,009.84	645	1,131	1,137	284.3	0.89

6. 부가 설명

- Clock cycle : 디지털 회로에서 신호가 한 번 변화하는 동안의 주기.
  - 이때마다 상태가 갱신됨.

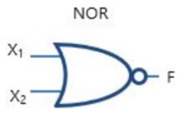


- Rising-edge detector



6. 부가 설명

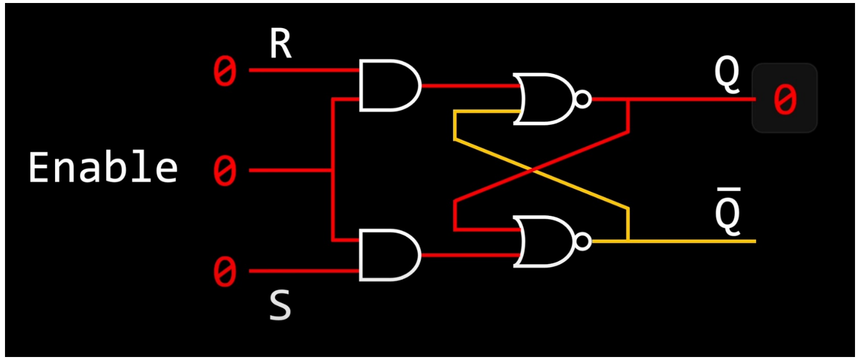
ㅇ 기호 및 진리표

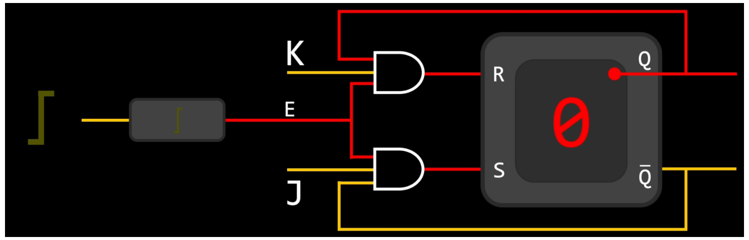


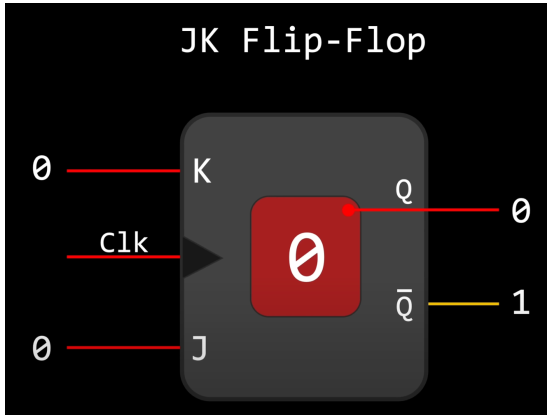
$F = \overline{X_1 + X_2} = \overline{X_1} \cdot \overline{X_2}$

X_1	X_2	F
0	0	1
0	1	0
1	0	0
1	1	0

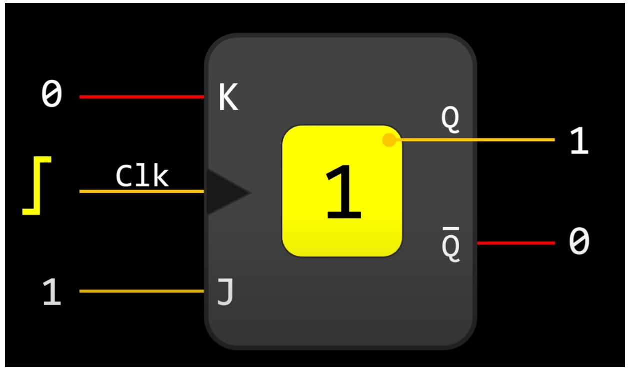
- SR-LATCH: 두 입력(SET, RESET)을 통해 비동기적으로 내부 Q 상태를 설정하거나 리셋하는 기본적인 1비트 저장 소자
- Gated SR-LATCH: Enable 신호가 활성화될 때만 set/reset을 반영해 Q를 갱신하는 latch
- JK filp-flop: J=K=1이면 Q 출력을 토글하는 기능이 추가된 동기식 SR 기반 flip-flop



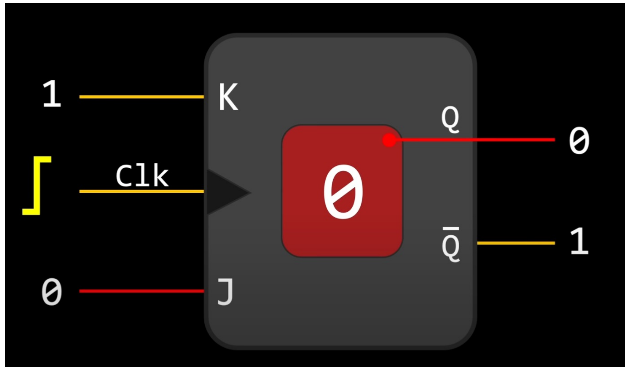




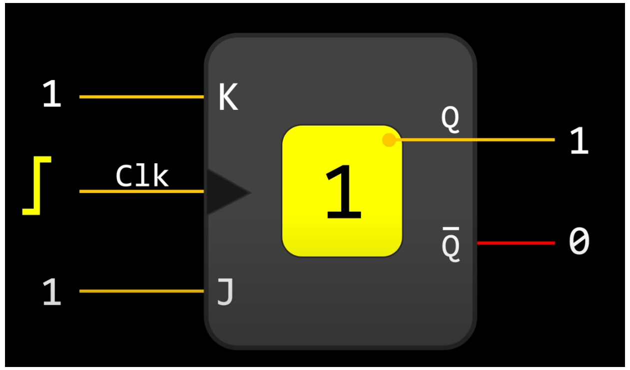
6. 부가 설명



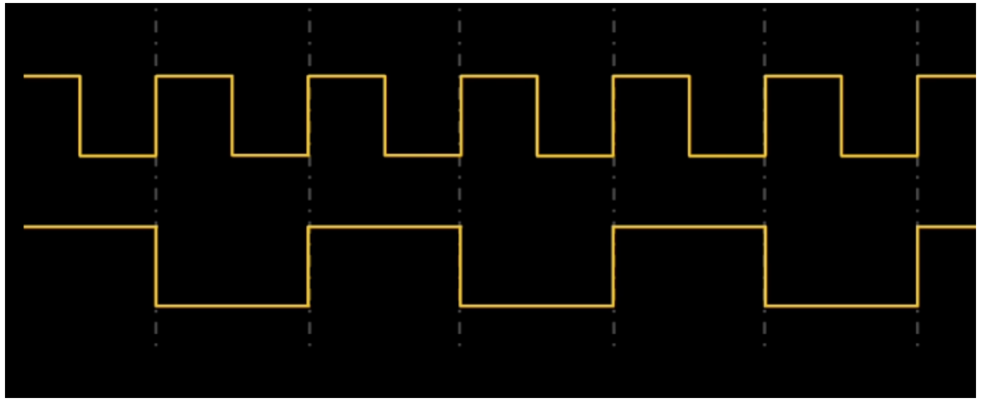
0
K
Q
1
Clk
1
Q̄
0
1
J

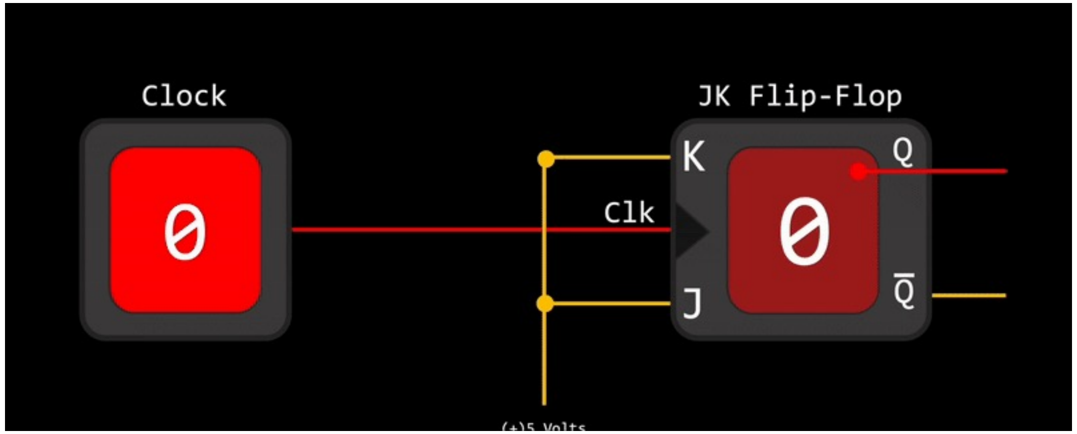


1
K
Q
0
Clk
0
Q̄
1
0
J

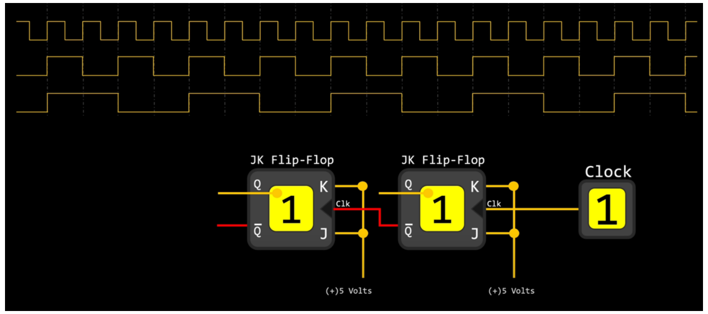


1
K
Q
1
Clk
1
Q̄
0
1
J





Clock
0
JK Flip-Flop
K
Q
Clk
0
J
Q̄



JK Flip-Flop
JK Flip-Flop
Clock

감 사 합 니 다