

임베디드 하드웨어

< 논리회로 >

1. 논리회로 기초

1) 디지털 시스템의 정의

- 디지털 시스템 : 정보를 디지털 데이터로 처리하는 시스템을 말한다. 이러한 시스템은 이산적인 값, 즉 디지털 데이터를 사용하여 연산을 수행하고, 정보를 저장하며, 통신을 진행한다. 대부분의 경우, 이러한 디지털 데이터는 0(LOW)과 1(HIGH)의 두 가지 상태만을 가진 이진수 형태로 표현된다.
- 디지털 시스템의 핵심 구성 요소
 - (1) 입력 장치 : 키보드, 마우스, 스캐너 등으로부터 사용자 입력이나 센서 데이터를 받아들인다.
 - (2) 처리 장치 : 중앙 처리 장치(CPU)와 같은 컴퓨터 하드웨어가 프로그램 명령어에 따라 데이터를 처리한다.
 - (3) 메모리 : RAM, ROM, 하드 드라이브 등에서 데이터와 명령어를 저장하고 검색한다.
 - (4) 출력 장치 : 모니터, 프린터 등으로 결과물을 출력한다.
 - (5) 제어 장치 : 다른 컴포넌트 사이의 작업 흐름을 조정하고 관리하는 역할을 한다.
- 디지털 시스템의 특징
 - (1) 잡음에 강하다 : 아날로그 시스템은 외부 잡음이나 온도 변화, 부품의 사용 기간 등에 민감하지만 디지털 시스템은 내·외부 잡음의 영향을 줄일 수 있다.
 - (2) 시스템 설계가 용이하다 : 디지털 시스템에서는 'ON'과 'OFF'의 두 가지 상태만이 존재하는 스위칭 회로를 사용하기 때문에 설계가 용이하다.
 - (3) 유연성이 높다 : 디지털 시스템은 프로그래밍으로 전체 시스템을 제어할 수 있기 때문에 새로운 규격 변화에 대해 유연성이 높다.
 - (4) 구성이 단순하다 : 디지털 시스템은 집적 회로로 구성되기 때문에 아날로그 회로에 비하여 구성이 단순하다.
 - (5) 정보의 저장과 가공이 용이하다 : 아날로그 시스템에서는 연속적인 값으로 계산을 하기 때문에 오차가 발생하기 마련이다.

그러나 디지털 시스템에서는 각각의 입력을 디지털 데이터로 변환하기 때문에 출력 데이터도 디지털 데이터로 정확하게 산출된다.

(6) 소형화와 저렴한 가격으로 구성이 가능하다 : 복잡한 회로를 반도체 집적회로를 사용하여 작은 크기로 제작할 수 있고, 대량 생산이 용이하기 때문에 가격이 낮아진다.

(7) 일반적으로 자연의 아날로그 신호를 받아들여 디지털 신호로 바꿔야 하기 때문에 A/D변환기와 D/A변환기가 필요하다.

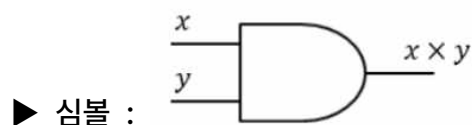
2) 불 대수

- 불 대수의 정의 : 불 대수(Boolean algebra)는 19세기 수학자 조지 부울이 개발한 논리 수학의 한 분야로, 이산적인 두 가지 값(0과 1, 또는 참과 거짓)을 다룬다. 이것은 컴퓨터 과학에서 매우 중요한 역할을 한다. 왜냐하면 디지털 로직 회로와 컴퓨팅 연산은 모두 불 대수를 기반으로 동작하기 때문이다.

- 불 대수의 주요 연산

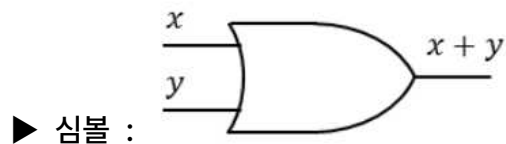
(1) AND : 모든 입력이 '참'일 때만 결과가 '참'이 되며, 식에서는 곱으로 표현된다.

X	Y	$X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1



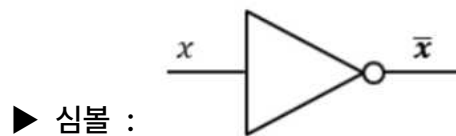
(2) OR : 적어도 하나의 입력이 '참'일 때 결과가 '참'이 되며 식에서는 덧셈으로 표현된다.

X	Y	$X + Y$
0	0	0
0	1	1
1	0	1
1	1	1



(3) NOT : 입력값을 반전시키며, 식에서는 컴포넌트(요소)위에 bar를 씌우거나 '를 씌운다.

X	\bar{X}
0	1
1	0



- 불 대수의 기본 정리

(1) 연산

▶ $X+0=X$ / $X+1=1$

▶ $X \cdot 1=X$ / $X \cdot 0=0$

(2) 멱등 법칙

▶ $X+X=X$ / $X \cdot X=X$

(3) 누승의 법칙 : $\bar{\bar{X}}=X$

(4) 상보의 법칙 : $X+\bar{X}=1$ / $X\bar{X}=0$

(5) 교환 법칙 : $XY=YX$ / $X+Y=Y+X$

(6) 결합 법칙 : $(XY)Z = X(YZ) = XYZ$

$$(X+Y) + Z = X+(Y+Z) = X+Y+Z$$

(7) 부울대수에서의 분배법칙(중요)

▶ 제 1법칙 : $X(Y+Z) = XY + XZ$

▶ 제 2법칙 : $X+(YZ) = (X+Y)(X+Z)$

(8) 간략화 정리

▶ $XY+X\bar{Y} = X$ / $(X+Y)(X+\bar{Y}) = X$

▶ $X+XY = X$ / $X(X+Y) = X$

▶ $(X+\bar{Y})Y = XY$ / $X\bar{Y}+Y = X+Y$

(9) 드모르간(De Morgan)의 법칙

▶ $\overline{X+Y} = \bar{X}\bar{Y}$

▶ $\overline{XY} = \bar{X}+\bar{Y}$

(10) 부울식의 쌍대

▶ AND → OR

▶ OR → AND

▶ 1 → 0

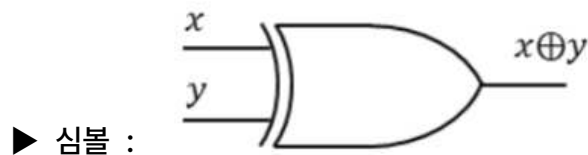
▶ 0 → 1

ex) $\overline{AB} + C$ 의 쌍대를 구하라.

→ ans) $(\overline{AB} + C)' = (\overline{AB})'C' = (\overline{A+B})C'$

(11) Exclusive OR(배타적 OR) : 두 변수가 같지 않을 때 참을 출력하는 연산이다. 기호로는 \oplus 를 사용한다.

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0



▶ 배타적 OR에 대한 정리

$$\rightarrow X \oplus 0 = X$$

$$\rightarrow X \oplus 1 = X'$$

$$\rightarrow X \oplus X = 0$$

$$\rightarrow X \oplus X' = 1$$

$$\rightarrow X \oplus Y = Y \oplus X \text{ (교환법칙)}$$


$$\rightarrow X \oplus Y \oplus Z = (X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) \text{ (결합법칙)}$$

$$\rightarrow X(Y \oplus Z) = XY \oplus XZ \text{ (분배법칙)}$$

$$\rightarrow (X \oplus Y)' = X \oplus Y' = X' \oplus Y = XY + X'Y'$$

(12) Exclusive NOR(배타적 NOR) : 두 변수가 같을 때 참을 출력하는 연산이다. 기호로는 \equiv 를 사용한다.

X	Y	$X \equiv Y$
0	0	1
0	1	0
1	0	0
1	1	1

▶ 심볼 : 

▶ $(X \oplus Y)' = (X \equiv Y)$

(13) 합의 정리

▶ $XY + X'Z + YZ = XY + X'Z$ (YZ를 중복되는 항으로 소거하여 정리)

→ proof) $XY + X'Z + YZ$

$$= XY + X'Z + (X+X')YZ$$

$$= (XY + XYZ) + (X'Z + X'YZ)$$

$$= XY(1 + Z) + X'Z(1 + Y)$$

$$= XY + X'Z$$

3) 논리식 간소화

- 논리식 간소화 : 주어진 논리식을 더 간단한 형태로 변환하는 과정
- 카르노 맵 : 부울 대수 위의 함수를 단순화 하는 방법이다.

▶ 2변수 카르노 맵 예시

$$F = A'B' + A'B$$

a \ b	0	1
0	1	1
1	0	0

: 0이 부정(')이고, 1이 원래 값이다.(0은 a',b'로 표현) 수식을 보면 a'b'과 a'b로 이루어져 있으므로 00인 자리와 01인 자리에 1을 채워 넣는다. 나머지 자리는 0을 채운다. 인접해 있는 두 '1'을 묶으면 다음과 같다.

a \ b	0	1
0	1	1
1	0	0

묶음 : $A'B'$, $A'B$ 로 표현할 수 있다.

A' 는 고정이고, B' 와 B 는 변하는 값임을 알 수 있다. 여기서 고정된 값만 남기면 A' 만 남게 되고 $F = A'$ 가 된다.

▶ 3변수 카르노 맵 계산 순서

① 먼저 카르노맵 표를 만든다.

② 함수에서 사용될 최소항들을 표의 각 칸안에 표시한다. 예를 들어 간소화할 항의 변수가 ABC 라면 A 를 1개항, BC 를 1개항으로 나누어 가로 세로 칸에 배분한다.

③ 변수의 개수가 2,3,4개인 경우에 주로 사용하도록 한다.

④ 2의 지수승으로 묶는다(2,4,8...개)

⑤ 바로 이웃해 있는 항들끼리 묶는다.

⑥ 반드시 직사각형이나 정사각형 형태로 묶는다.(최대한 크게 묶는다)

⑦ 묶은 항들의 변수가 변화했는지 확인하고 변화하지 않은 변수를 뽑는다.

(묶음 내에서 변수간에는 논리곱(\cdot), 묶음항간에는 논리합($+$)을 한다.)

▶ 3변수 카르노 맵 예시

$$F = \overline{A} \overline{B} \overline{C} + \overline{A} B \overline{C} + A \overline{B} \overline{C} + A \overline{B} C + AB \overline{C} + ABC$$

① 카르노 맵의 표를 만든다.

		$\overline{B} \overline{C}$	$\overline{B} C$	$B C$	$B \overline{C}$
\overline{A}	0	0	1	3	2
	1	4	5	7	6

: 빨간 숫자는 각 칸의 이진수로 표시하는데, 특별한 의미는 없으며 위 표대로 고정라는 것이 3변수 카르노맵의 규칙이다.

② 표에 나타낼 식을 카르노 맵에 적어 준다.

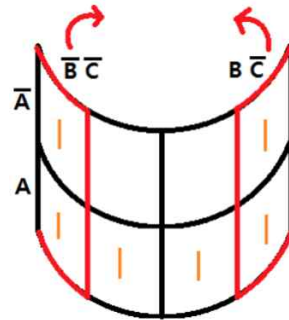
		$\overline{B} \overline{C}$	$\overline{B} C$	$B C$	$B \overline{C}$
\overline{A}	0	1			1
	1	1	1	1	1

: F식에 존재하는 해당 칸에 '1'을 적는다.

③ 2의 지수승으로 최대한 큰 사각형으로 묶는다.

		$\overline{B}\overline{C}$	$\overline{B}C$	BC	$B\overline{C}$
\overline{A}	0	1			1
A	1	1	1	1	1

		$\overline{B}\overline{C}$	$\overline{B}C$	BC	$B\overline{C}$
\overline{A}	0	1			1
A	1	1	1	1	1



: 양쪽 인접한 부근은 표를 동그란 구로 보기 때문에 정사각형으로 묶을 수 있다.

④ 묶은 항들의 변수가 변화했는지 확인하고 변화하지 않은 변수를 뽑는다.

첫 번째 묶음 : $A\overline{B}\overline{C}, A\overline{B}C, ABC, AB\overline{C}$

두 번째 묶음 : $\overline{A}\overline{B}\overline{C}, \overline{A}B\overline{C}, A\overline{B}\overline{C}, AB\overline{C}$

첫번째 묶음			두번째 묶음		
A	B	C	A	C	C
1	0	0	0	0	0
1	0	1	1	0	0
1	1	1	0	1	0
1	1	0	1	1	0

: 첫 번째 묶음에서는 A가 '1'로서 변하지 않았고, 두 번째 묶음에서는 C가 '0'으로서 변하지 않았다. 따라서 전체식은 다음과 같이 간소화 될 수 있다.

$$\rightarrow F = A + \overline{C}$$

4) 수의 표현

- 수 체계와 변환

- ▶ 2진수 : 0과 1만을 사용하는 2진 체계의 숫자
- ▶ 8진수 : 0부터 7까지 여덟 가지 숫자를 사용
- ▶ 10진수 : 0부터 9까지 열 가지 숫자를 사용
- ▶ 16진수 : 0부터 F까지 열여섯 가지의 숫자를 사용
- ▶ 수의 변환

- ① R진수에서 10진수로 변환 : 2진수로 변환 후, 2진수에서 10진수로 변환
- ② 2진수에서 8진수, 16진수로 변환 : 8진수는 3자리씩 묶고, 16진수는 4자리씩 묶어서 2진수로 변환

▶ 2진법 산술연산

① 2진법의 덧셈

$$\begin{array}{r}
 0+0=0 \\
 0+1=1 \\
 1+0=1 \\
 1+1=0 \rightarrow \text{carry (자리올림적용)}
 \end{array}
 \qquad
 \begin{array}{r}
 13_{10} = 1101 \\
 11_{10} = 1011 \\
 \hline
 11000_2 = 24_{10}
 \end{array}$$

② 2진법의 뺄셈

$$\begin{array}{r}
 0-0=0 \\
 0-1=1 \rightarrow \text{borrow (자리내림 적용)} \\
 1-0=1 \\
 1-1=0
 \end{array}
 \qquad
 \begin{array}{r}
 13_{10} = 1101 \\
 - 11_{10} = 1011 \\
 \hline
 1010_2
 \end{array}$$

③ 2진법의 곱셈

$$\begin{array}{r}
 13_{10} = 1101 \\
 11_{10} = 1011 \\
 \hline
 \begin{array}{r}
 1101 \\
 1101 \\
 1101 \\
 1101 \\
 \hline
 1001111 \\
 0000 \\
 \hline
 1001111_2 = 143_{10}
 \end{array}
 \end{array}$$

④ 2진법의 나눗셈

$$\begin{array}{r}
 1101 \\
 1011 \overline{) 10010001} \\
 \underline{1011} \\
 1110 \\
 \underline{1011} \\
 1101 \\
 \underline{1011} \\
 10
 \end{array}$$

- 음수의 표현

▶ 컴퓨터는 덧셈밖에 하지 못하기 때문에 '보충해주는 수'의 의미인 보수를 도입하여 뺄셈을 할 수 있도록 방법을 만들었다. 보수는 1의 보수와 2의 보수가 있다.

▶ 컴퓨터에서 음수를 표현하는 방법은 MSB(Most Significant Bit)를 부호로 사용하여 MSB가 0이면 '+', 1이면 '-'부호를 갖게 된다.

▶ 1의 보수

① 1의 보수의 표현 : MSB를 1로 변경 후 MSB를 제외한 모든 비트를 1→0, 0→1로 변경한다.

(n비트에서 양의 정수 N에 대한 2의 보수

$$\overline{N} = (2^n - 1) - N$$

② 1의 보수의 덧셈

case1>양수와 그보다 절대값이 작은 음수의 덧셈(올림수 발생 +1)

$$\text{ex) } 10 - 5 = 00001010 - 00000101$$

$$= 00001010 + 11111010$$

$$\begin{array}{r}
 00001010 \\
 + 11111010 \\
 \hline
 \boxed{1}0000100 + 1
 \end{array}
 \Rightarrow 00000101 = 5$$

자리올림 방법

case2>양수와 그보다 절대값이 큰 음수의 덧셈(올림수 발생 X)

$$\begin{aligned} \text{ex) } 5 - 10 &= 00000101 - 00001010 \\ &= 00000101 + 11110101 \end{aligned}$$

$$\begin{array}{r} 00000101 \\ + 11110101 \\ \hline 11111010 = -5 \end{array} \rightarrow -5 \text{ 임을 더 빨리 쉽게 알기 위해 1의 보수를 취해줌}$$

$$\sim 00001010 = -5$$

부호개조

case3>overflow가 발생하는 경우(n 비트의 두 음수의 덧셈 $\geq 2^{n-1}$ 인 경우)

$$\begin{array}{r} -5 = 1010 \\ + -6 = 1001 \\ \hline \boxed{1}0011 \end{array}$$

: overflow 발생 오류

▶ 2의 보수

① 2의 보수의 표현 : 1의 보수를 취하고 +1을 해준다.

(n 비트에서 양의 정수 N 에 대한 2의 보수)

$$N^* = 2^n - N = (2^n - 1 - N) + 1 = \overline{N} + 1$$

② 2의 보수의 덧셈

case1>양수와 그보다 절대값이 작은 음수의 덧셈(올림수 발생 → 무시)

$$\begin{aligned} 10 - 5 &= 00001010 - 00000101 \\ &= 00001010 + (11111010 + 1) \end{aligned}$$

$$\begin{array}{r} 00001010 \\ + 11111011 \\ \hline \boxed{1}0000101 \Rightarrow 00000101 = 5 \end{array}$$

무시

case2>양수와 그보다 절대값이 큰 음수의 덧셈(올림수 발생 X)

$$\begin{aligned}
 5-10 &= 00000101 - 00001010 \\
 &= 00000101 + (11110101+1) \\
 &\quad \begin{array}{r} 00000101 \\ + 11110101 \\ \hline 11111011 \end{array} \rightarrow \text{이기 쉽게 하기 위해 2의 보수를 취함} \\
 10000100+1 &= 10000101 \\
 -0101 &= -5
 \end{aligned}$$

case3>overflow가 발생하는경우(n 비트의 두 음수의 덧셈 $\geq 2^{n-1}$ 인 경우)
: overflow발생 오류

- 2진 코드

- ▶ 각 10진수 숫자들을 동등한 2진수로 대체시키는 것을 부호화(coding)이라고 한다. 2진 코드의 조건은 각 10개의 10진수에 대해 서로 다른 2진 숫자들의 조합으로 표현되어야 한다는 것이다.
- ▶ 2진 코드의 종류
 - ① 8-4-2-1 BCD Code : 각 자리가 8-4-2-1의 값을 갖는 2진 코드
 - ② 6-3-1-1 Code : 각 자리가 6-3-1-1의 값을 갖는 2진 코드
 - ③ Excess-3 Code : XS-3 Code라고도 하며 각 10진수 숫자를 그에 대응하는 4비트 2진수 값에 3을 더한 값으로 표현한다.
 - ④ 2-out-of-5 Code : 5비트 중에서 2비트만 1을 갖도록 표현한 2진코드(오류 감지에 유용하며, 특히 통신에서 오류를 감지하거나 수정하는데 사용된다.)
 - ⑤ Gray Code : 연속하는 두 수 사이에 발생하는 변화는 항상 한 비트에 대한 것만을 포함하도록 한 숫자 체계이다.(디지털 시스템에서 아날로그 값을 디지털 형태로 변환할 때 발생할 수 있는 오류를 줄여주며 주로 회전 인코더와 같은 장치에서 사용된다.)

10진수	BCD Code	2-out-of-5 Code	Gray Code	Xs-3 Code	6-3-1-1 Code
0	0000	00011	0000	0011	0000
1	0001	00101	0001	0100	0001
2	0010	00110	0011	0101	0011
3	0011	01001	0010	0110	0100
4	0100	01010	0110	0111	0101
5	0101	01100	0111	1000	0111
6	0110	10001	0101	1001	1000
7	0111	10010	0100	1010	1001
8	1000	10100	1100	1011	1011
9	1001	11000	1101	1100	1100

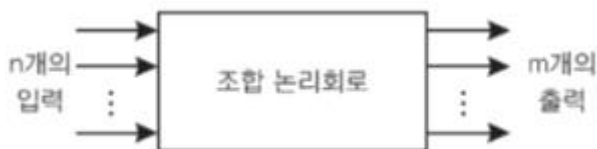
2. 조합논리회로(Combination logic circuit)

1) 각종 조합논리회로

- 조합 논리회로(Combination logic circuit) : 임의의 시점에서의 출력 값이 그 시점의 입력 값에 의해서만 결정되는 논리회로(내부 기억능력 즉, 메모리를 갖지 않는다.)

ex) NOT, AND, OR, XOR, NOR, NAND, 반가산기, 전가산기, 디코더, 인코더, 멀티플렉서, 디멀티플렉서 등

▶ 조합 논리회로의 블록도



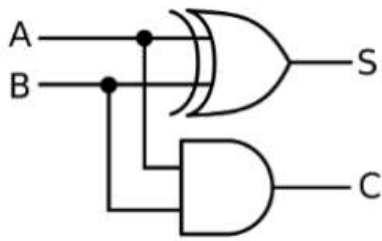
- 가산기

▶ 반가산기(Half Adder) : 2진수 1자리(1bit)의 A와 B를 더하여 합(sum)과 자리올림수(carry)를 산출하기 위한 조합 논리회로로 2개의 입출력 값을 가진다.

[진리표]

A	B	합(S)	자리 올림수(C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1
		XOR게이트	AND게이트

< 진리표 >



논리식

$$\text{Carry} = A \cdot B$$

$$\text{Sum} = A'B + AB' = A \oplus B$$

< 회로도 와 논리식 >

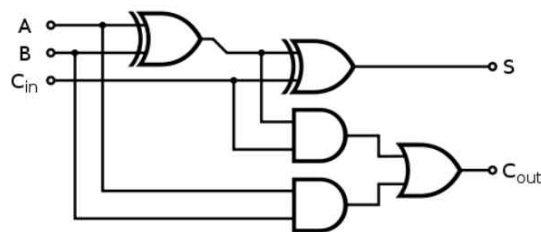
▶ 전가산기(Full Adder) : 두 비트(A,B)와 전 상태의 자리 올림수(C0)를 더해 합(S)과 최종 자리 올림수(C1)를 얻는 회로이다. 3개의 입력과 2개의 출력을 갖는 회로이며, 2개의 반가산기와 1개의 OR게이트로 구성되어 있다.

[진리표]

A	B	C0	합(S)	자리 올림수 (C1)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

< 진리표 >

[회로도]



[논리식]

$$S = A(XOR)B(XOR)C$$

$$C1 = A*B+(A(XOR)B)*C0$$

< 회로도와 논리식 >

- 디코더(Decoder 해독기)

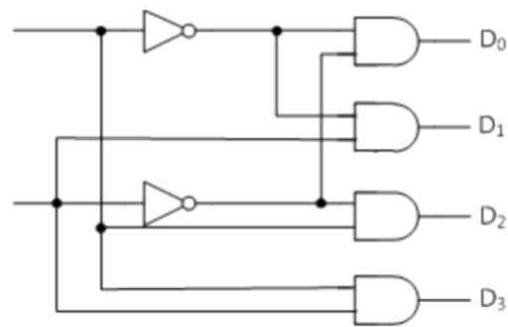
▶ 디코더는 AND게이트로 구성되며, n개의 입력을 받아들여, 2^n 개의 데이터를 출력한다.

[진리표]

A	B	D0	D1	D2	D3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

< 진리표 >

[회로도]



< 회로도 >

$$D_0 = \overline{A}\overline{B}$$

$$D_1 = \overline{A}B$$

$$D_2 = A\overline{B}$$

$$D_3 = AB$$

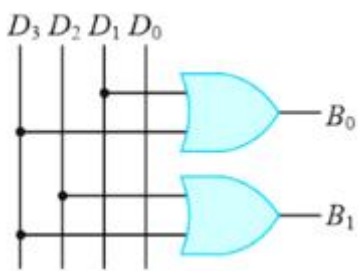
< 논리식 >

- 인코더(Encoder 부호기)

▶ 인코더는 OR게이트로 구성되며, 2^n 개의 입력을 받아들여 n개의 데이터를 출력한다. 특정 값을 여러 자리 2진수로 변환하거나, 특정 장치로부터의 신호를 여러 개의 2진 신호로 변환시키는 장치이다.

D3	D2	D1	D0	X	Y
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

< 진리표 >



< 회로도 >

$$B_0 = D_1 + D_3$$

$$B_1 = D_2 + D_3$$

< 논리식 >

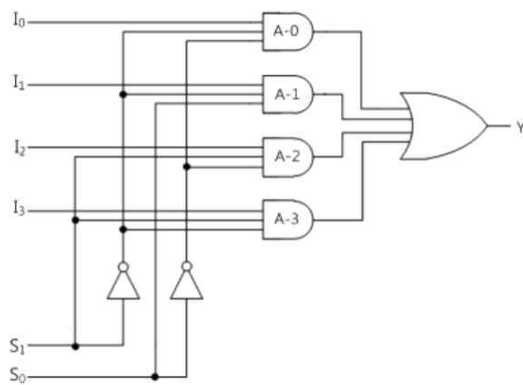
- 멀티플렉서(Multiplexer MUX)

▶ 멀티플렉서는 2^n 개의 입력선과 n개의 선택선들을 입력으로 받아 하나의 출력선으로 정보를 출력하는 논리회로이다.

S_0	S_1	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

< 진리표 >

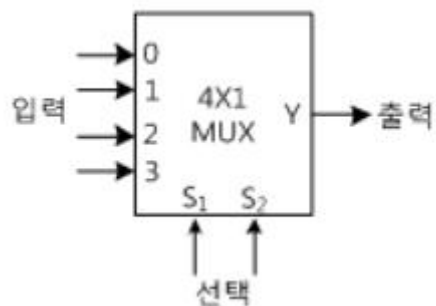
[회로도]



< 회로도 >

$$Y = (\overline{S_1} \overline{S_0} I_0) + (\overline{S_1} S_0 I_1) + (S_1 \overline{S_0} I_2) + (S_1 S_0 I_3)$$

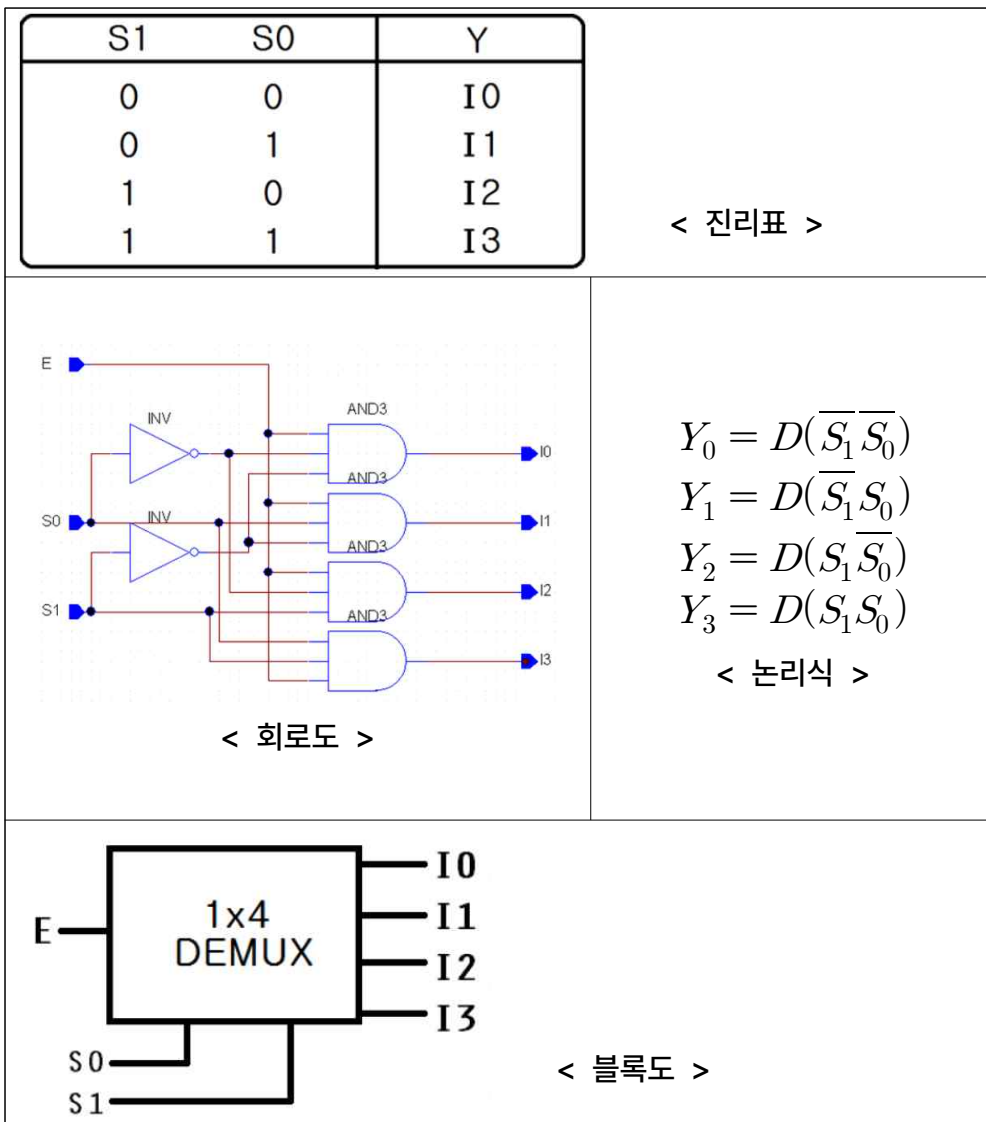
< 논리식 >



< 블록도 >

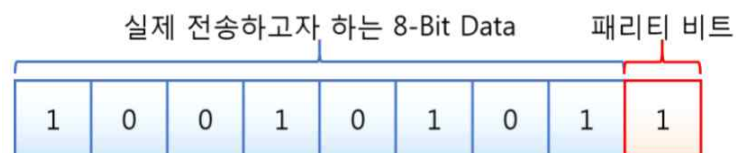
- 디멀티플렉서(Demultiplexer)

▶ 1개의 입력을 받아 2^n 개의 출력선 중 하나의 선을 선택하여 출력하는 회로이다.



- 패리티 비트(Parity Bit)

▶ 패리티 비트는 정보의 전달 과정에서 오류가 생겼는지를 검사하기 위해 추가된 비트이다. 전송하고자 하는 데이터의 끝에 1비트를 더하여 전송하는 방법으로 2가지 종류의 패리티 비트(홀수, 짝수)가 있다.



: 실제 전송하고자 하는 8-bit의 데이터 외에 추가적으로 패리티 비트를 하나 추가하여 송수신을 하게 되는데, 추가적으로 붙게되는 패리티 비트는 짝수 패리티(Even Parity)와 홀수 패리티(Odd Parity)가 있다.



: 실제 전송하고자 하는 8-bit Data에 추가적으로 붙게되는 패리티 비트를 짝수 패리티로 설정할 것인지 홀수 패리티로 설정할 것인지에 따라 붙게되는 패리티 비트의 값이 달라지게 된다. 꼭 짝수 패리티가 '0'이고 홀수 패리티가 '1'인 것이 아니다. 짝수 패리티는 실제 송신 데이터의 각 비트의 값 중에서 1의 개수가 짝수가 되도록 패리티 비트를 정하는 것인데, 만약 데이터 비트에서 1의 개수가 홀수이면 패리티 비트를 1로 정한다. 마찬가지로 홀수 패리티는 송신 데이터의 1의 개수가 홀수가 되도록 패리티 비트를 정한다.

▶ 패리티 비트의 사용 용도 : 시리얼 통신은 데이터를 송수신하는 과정에서 각 비트를 단위시간당 하나씩 보내게 되어있는데, 이 때 알 수 없는 요인에 의해 비트의 값이 틀어져 0과 1이 바뀌거나 손실될 수 있다. 이러한 경우 패리티 비트를 적용하여 오류가 있는지 없는지 확인(오류의 유무만 확인 가능)할 수 있고, 오류가 발생하면 재전송을 요구할 수 있다. 패리티 비트는 주로 시리얼 통신의 거리가 멀 경우에 사용(전송시간이 길어짐)하고, 거리가 가까운 경우는 CheckSum 데이터를 추가하는 방법으로 오류를 검출한다.

- 오류 수정 코드

▶ 해밍 코드(Hamming Code) : 특정 패턴의 패리티 비트를 원래의 데이터 비트에 삽입하여, 전체 데이터 세트에서 각 비트의 위치가 고유한 패턴(해밍 비트)을 갖도록 한 뒤, 데이터 수신 시 동일한 패턴으로 해밍 비트를 계산하고 원래의 전송된 해밍 비트와 대조한다. 이 과정에서 차이가 발견되면, 어떤 위치의 비트에 오류가 있는지 정확히 알 수 있으며 해당 위치의 비트 값을 반전시켜 원래의 올바른 값으로 복구까지 할 수 있다. RTOS(Real Time Operating System)에서와 같이 실시간으로 데이터가 전송되어야 하는 경우 데이터 전송과 동시에 에러정정이 수행되어야 하기 때문에 대부분의 MCU 디바이스에 사용되고 있다.

▶ 패리티 비트의 삽입 원리

$$\rightarrow 2^p \geq d + p + 1 \quad (d: \text{데이터비트}, p: \text{패리티비트})$$

: 위 식을 만족하는 p의 최소값만큼 데이터 비트가 필요하다.

→ 패리티 비트는 $2^n (n = 0, 1, 2, \dots)$ 의 위치로 순서대로 삽입된다.

비트 위치	1	2	3	4	5	6	7	8	9	10	11	12
기호	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	D12

▶ 패리티 비트의 체크 범위

→ n번째 패리티 비트는 n번째부터 체크를 시작하며, n비트 만큼씩 n비트를 건너뛰어 패리티 비트를 지정하고 각 패리티 비트를 결정한다.

비트 위치	1	2	3	4	5	6	7	8	9	10	11	12
기호	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	D12
P1 영역	*		*		*		*		*		*	
P2 영역		*	*			*	*			*	*	
P4 영역				*	*	*	*					*
P8 영역								*	*	*	*	*

→ 패리티 비트의 값 결정

$$P1의\ 영역 = D3 + D5 + D7 + D9 + D11$$

$$P2의\ 영역 = D3 + D6 + D7 + D10 + D11$$

$$P4의\ 영역 = D5 + D6 + D7 + D12$$

$$P8의\ 영역 = D9 + D10 + D11 + D12$$

→ 예시

기호	D3	D5	D6	D7	D9	D10	D11	D12
데이터	0	1	0	1	1	0	1	1

$$P1 = D3 + D5 + D7 + D9 + D11 = 0 + 1 + 1 + 1 + 1 = 0$$

$$P2 = D3 + D6 + D7 + D10 + D11 = 0 + 0 + 1 + 0 + 1 = 0$$

$$P4 = D5 + D6 + D7 + D12 = 1 + 0 + 1 + 1 = 1$$

$$P8 = D9 + D10 + D11 + D12 = 1 + 0 + 1 + 1 = 1$$

: 전송할 데이터 비트가 01011011이고 짝수 패리티 비트를 사용한다면 위와 같이 해밍 코드가 형성된다.

▶ 오류 검출 및 오류 비트 수정 방법

→ 해밍 코드를 통해 오류 검출 및 오류 비트를 수정하려면
신드롬(Syndrome) 값을 구해야 한다.

→ 신드롬(Syndrome) : 패리티 비트를 포함하여 얻은 2진수 값으로 0~15 사이의 값으로 만들어지는 오류 값이다. 만약 신드롬 값이 0이라면 오류가 없는 것이며, 6이면 6번째 비트인 D6에서 오류가 발생한 것이다.

→ 신드롬 값 구하기

비트 위치	1	2	3	4	5	6	7	8	9	10	11	12
기호	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	D12
데이터			0		1	0	1		1	0	1	1
해밍코드	0	0	0	1	1	0	1	1	1	0	1	1

$$C1 = P1 + D3 + D5 + D7 + D9 + D11 = 0 + 0 + 1 + 1 + 1 + 1 = 0$$

$$C2 = P2 + D3 + D6 + D7 + D10 + D11 = 0 + 0 + 0 + 1 + 0 + 1 = 0$$

$$C4 = P4 + D5 + D6 + D7 + D12 = 1 + 1 + 0 + 1 + 1 = 0$$

$$C8 = P8 + D9 + D10 + D11 + D12 = 1 + 1 + 0 + 1 + 1 = 0$$

: 위의 예시에서 신드롬 값을 계산하면 위와 같다. 신드롬 값은 패리티 비트를 포함하여 검사하며 짝수 패리티 비트이기 때문에 신드롬 또한 짝수 규칙을 따라 계산한다.

$$0bC1C2C4C8 = 0b0000 = 0$$

: 신드롬을 2진수값으로 위와 같이 변환한다. 값이 0이기 때문에 위의 예제는 오류가 없다.

2) 조합논리회로 분석, 설계

▶ 조합 논리회로를 설계할 때의 순서

- (1) 입력과 출력 조건에 적합한 진리표를 작성
- (2) 진리표에 해당하는 논리식 도출
- (3) 도출한 논리식을 OR연산자로 결합
- (4) 카르노 맵을 통해 논리식을 간단하게 정리한다.

→ ex) 두 입력 A, B를 비교하여 $A > B$ 또는 $A = B$ 이면 출력이 '1'이고, $A < B$ 이면 출력이 '0'이 되는 논리회로를 설계할 때 논리식을 구하여라.

- ① 진리표 작성 : 모든 가능한 입력 조합에 대해 원하는 출력을 나타내는 진리표를 작성한다.

A	B	Output
0	0	1
0	1	0
1	0	1
1	1	1

- ② 논리식 도출 : 위 진리표에서 출력이 '1'인 경우에 해당하는 논리식을

도출한다.

- 첫 번째 행(A=0, B=0)에서 출력은 'A'와 'B'가 모두 거짓일 때 참이다. 따라서 이 경우의 논리식은 NOT A AND NOT B (A'B') 이다.
- 세 번째 행(A=1, B=0)에서 출력은 'A'가 참이고 'B'가 거짓일 때 참이다. 따라서 이 경우의 논리식은 A AND NOT B (AB') 이다.
- 네 번째 행(A=1, B=1)에서 출력은 'A'와 'B'가 모두 참일 때 참이다. 따라서 이 경우의 논리식은 A AND B (AB) 이다.

③ 논리 연산자 OR로 결합 : 각각의 조건들을 OR연산자로 결합하여 최종 논리식을 얻는다.

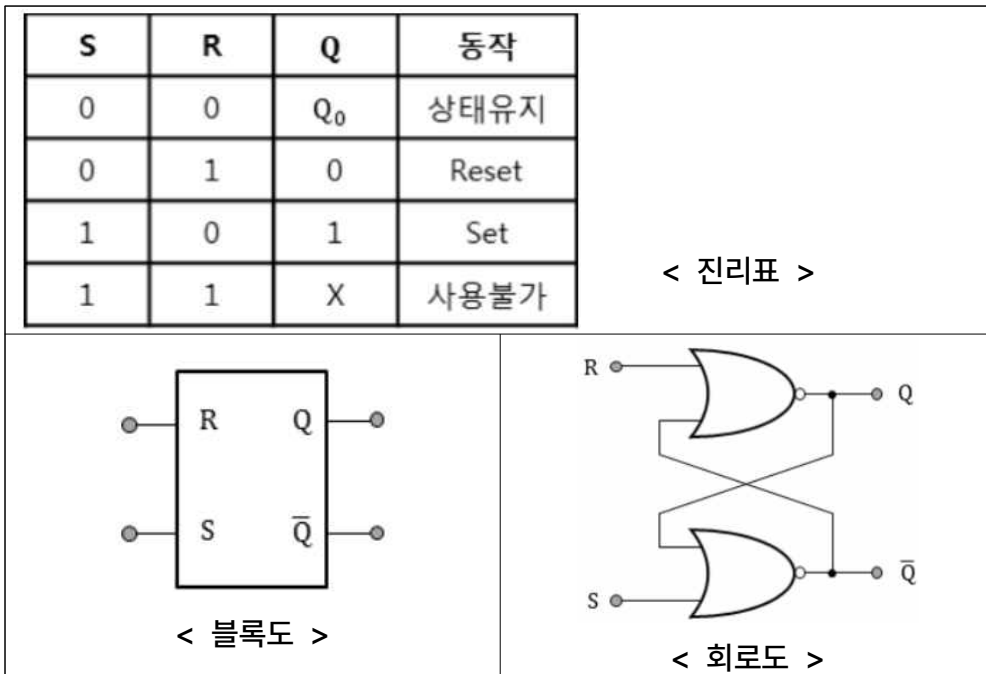
$$\cdot \text{Final Output} = (A'B') + AB' + AB$$

④ 카르노 맵을 이용해 논리식을 간소화 한다.

3. 순서논리회로

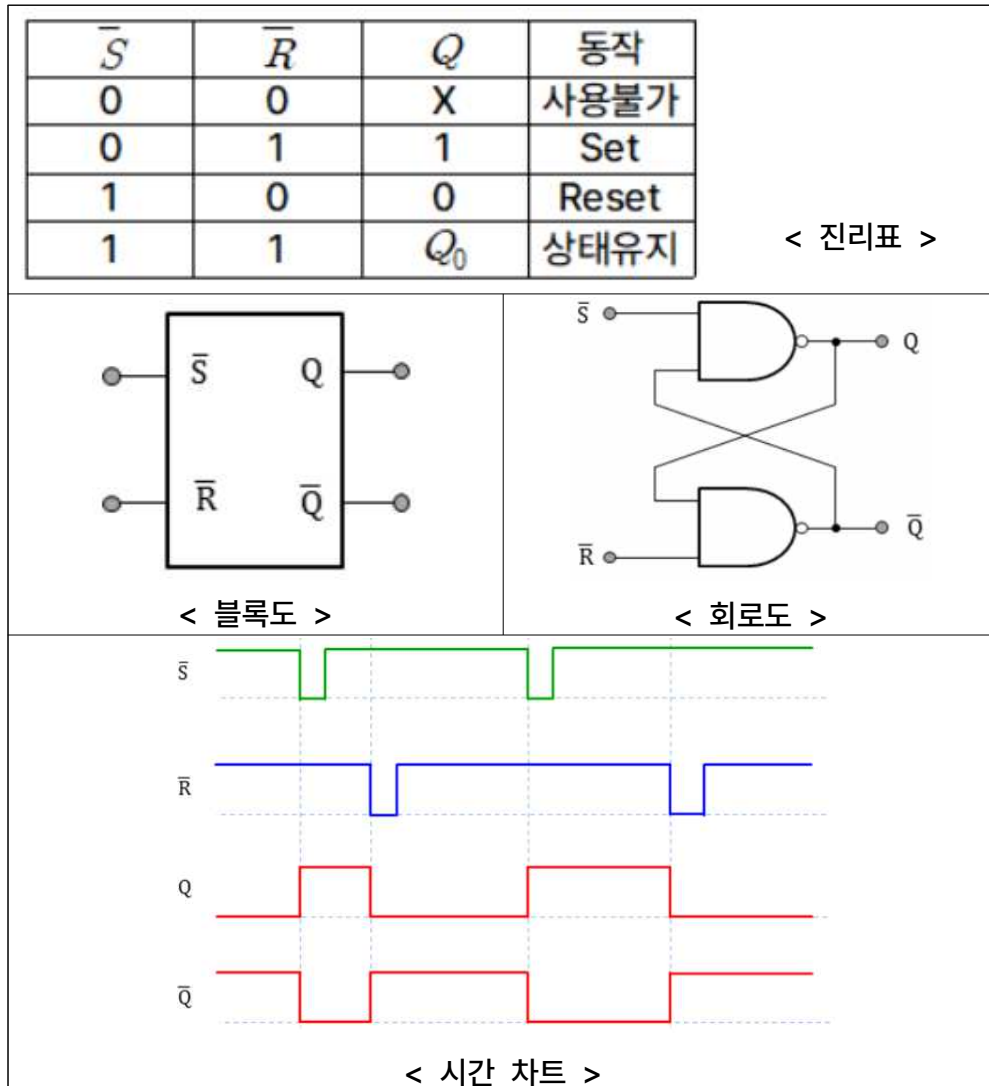
1) 래치와 플립플롭

- 순서논리회로 : 외부로부터의 입력과 현재 상태에 따라 출력이 결정되는 회로이며(출력이 입력에 영향을 주는 피드백 회로), 메모리가 존재해 기억 기능이 있다. 회로구성은 플립플롭과 논리 게이트로 구성되며, 신호의 타이밍에 따라 동기식과 비동기식으로 나뉘어진다.
 - ▶ 동기식 정보전달 : 컴퓨터의 모든 구성 요소들이 일정한 시간 간격을 두고 클록 펄스(CP, Clock Pulse)를 발생시켜서 타이밍을 맞추어 동기화 되어 동작하는 것이다.
 - ▶ 비동기식 정보전달 : 논리회로가 규칙적인 시간과 타이밍에 관계없이 입력 신호가 들어오면 곧바로 동작하는 것을 말하며, 클록 펄스가 없기 때문에 수신된 정보의 시작과 끝을 식별하기 위한 표시 데이터가 필요하다.
- 래치(Latch) : 전원이 공급되는 동안만 상태의 변화를 위한 신호가 발생할 때까지 현재의 상태를 그대로 유지하는 논리회로 기억장치로서 메모리 소자(휘발성)로 활용한다(1비트 '0' or '1'의 정보를 기억할 수 있는 최소의 기억소자). 클록 입력을 갖지 않는 쌍안정 회로이며 플립플롭과 유사하나 클록이 없으므로 비동기식 순차 논리회로 소자이다.
 - ▶ SR(Set-Reset) Latch
 - ① SR NOR Latch



: 가장 단순한 순차논리회로이며, S는 set을 의미하고 R은 reset을 의미한다. S=1,R=0 또는 S=0,R=1 일 때 출력값이 변화하며, S=0,R=0 이면 출력(Q) 상태가 피드백 입력이 되어 이전 상태를 그대로 유지할 수 있다. S=1,R=1의 입력은 출력이 모두 0이 되기 때문에 사용하지 않는다.

② SR NAND Latch



: SR NOR Latch와 원리는 같지만 입력이 inverting되어 있는 회로이다.

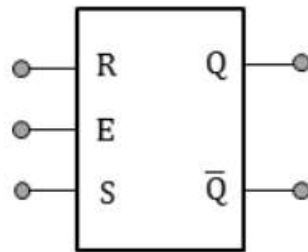
$\bar{S}=0, \bar{R}=1$ 또는 $\bar{S}=1, \bar{R}=0$ 일 때 출력값이 변화하며

$\bar{S}=1, \bar{R}=1$ 이면 이전 상태를 그대로 유지한다. $S=0, R=0$ 일 경우 출력이 모두 0이 되므로 사용하지 않는다.

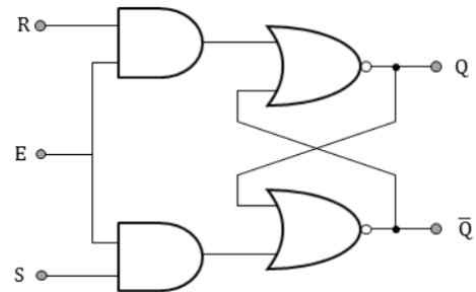
③ Gate SR NOR Latch

S	R	E	Q	동작
0	0	0	Q_0	상태유지
0	0	1	Q_0	상태유지
0	1	0	Q_0	상태유지
0	1	1	0	Reset
1	0	0	$Q_0(0)$	상태유지
1	0	1	1	Set
1	1	0	$Q_0(1)$	상태유지
1	1	1	X	사용불가

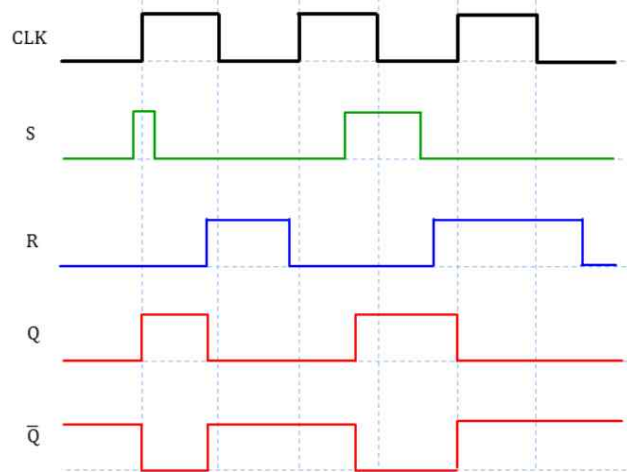
< 진리표 >



< 블록도 >



< 회로도 >



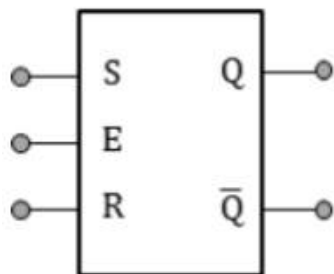
< 시간 차트 >

: NOR Latch 회로에 Enable신호를 사용할 수 있게 만든 회로이다.
 Enable신호가 LOW이면 무조건 이전상태를 유지하게 되며, Enable신호가 HIGH일 때, S,R의 상태에 따라 출력이 나타난다. 따라서 S,R 상태보다 우선하여 Enable의 상태에 따라 동기화되어 출력이 나타나므로 동기화 회로라고 한다.

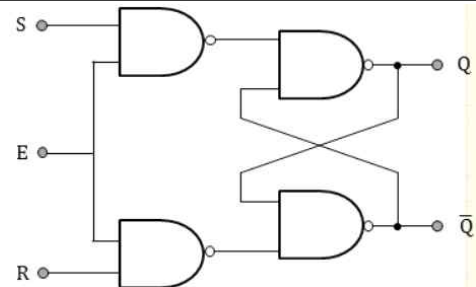
④ Gate SR NAND Latch

S	R	E	Q	동작
0	0	0	Q_0	상태유지
0	0	1	Q_0	상태유지
0	1	0	Q_0	상태유지
0	1	1	0	Reset
1	0	0	$Q_0(0)$	상태유지
1	0	1	1	Set
1	1	0	$Q_0(1)$	상태유지
1	1	1	X	사용불가

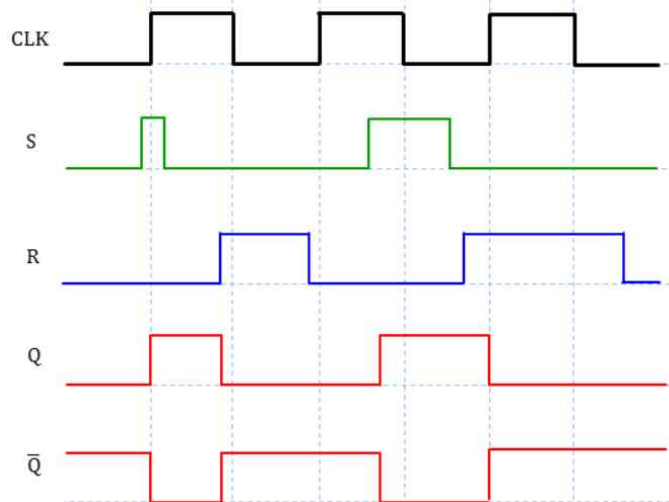
< 진리표 >



< 블록도 >



< 회로도 >



< 시간 차트 >

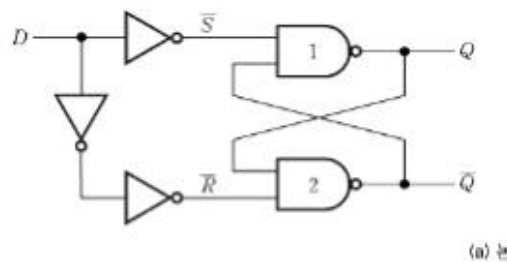
: NAND Latch회로에 Enable신호를 사용할 수 있게 만든 회로이다. NOR Latch회로와 S와 R의 위치가 뒤바뀌어 있으며, 진리표는 NOR Latch와 동일하다.

▶ D(Data) Latch

① D형 Latch

D	Q	\overline{Q}	동작
0	0	1	Reset
1	1	0	Set

< 진리표 >



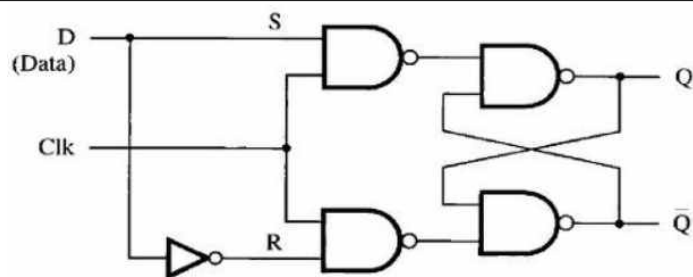
< 회로도 >

: Data값의 입력을 임시저장하고 출력하는 회로로서 D가 0일 때 LOW, 1일 때 HIGH를 출력하는 회로이다.

② 클록형 D Latch

D	CLK (enable)	Q	\overline{Q}	동작
0	1	0	1	Reset
1	1	1	0	Set
X	0	Q_0	$\overline{Q_0}$	상태유지

< 진리표 >



< 회로도 >

: Enable신호가 HIGH일 때, D값이 출력되고, Enable신호가 LOW일 때, 이전 값을 출력하는 회로이다.

▶ JK Latch

J	K	Q	\overline{Q}	동작	< 진리표 >
0	0	Q_0	$\overline{Q_0}$	상태유지	
0	1	0	1	Reset	
1	0	1	0	Set	
1	1	$\overline{Q_0}$	Q_0	상태반전 (Toggle)	

< 회로도 >

: SR래치의 확장개념이다. SR래치의 정의되지 않은 입력 문제를 해결하기 위해 고안되었으며, SR래치와는 다르게 $J=K=1$ 의 상태에서도 동작이 가능하다.

▶ T(Toggle) Latch

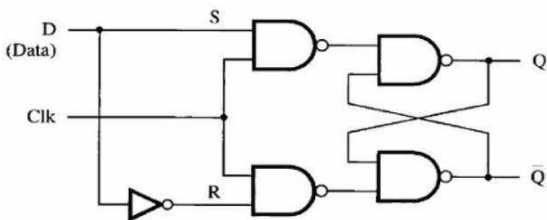
T	Q	\overline{Q}	동작	< 진리표 >
0	Q	\overline{Q}	상태유지	
1	\overline{Q}	Q	상태반전 (Toggle)	

< 회로도 >

: SR래치의 확장 개념이라고 볼 수 있다. T신호 하나로 S와 R의 신호를 번갈아가며 입력되게 한다. 이를 위해서 두 개의 논리곱 게이트(AND)를 추가하였고, SR래치의 정의되지 않은 입력 문제를 해결하기 위해 고안되었다.

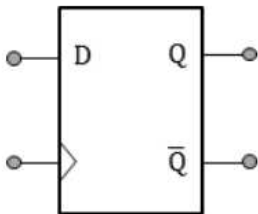
- 플립플롭(Flip-Flop) : 입력값을 임시로 저장하고, 특정 조건에서 그 값을 유지하는 회로이다. 특히, 플립플롭은 클록 신호의 상태 변화에 따라 동작하므로, 시간에 따른 데이터의 변경을 제어하는데 사용된다.(래치에서 클록신호를 추가하여 동작하는 회로이다.)

▶ D(Data) Flip-Flop



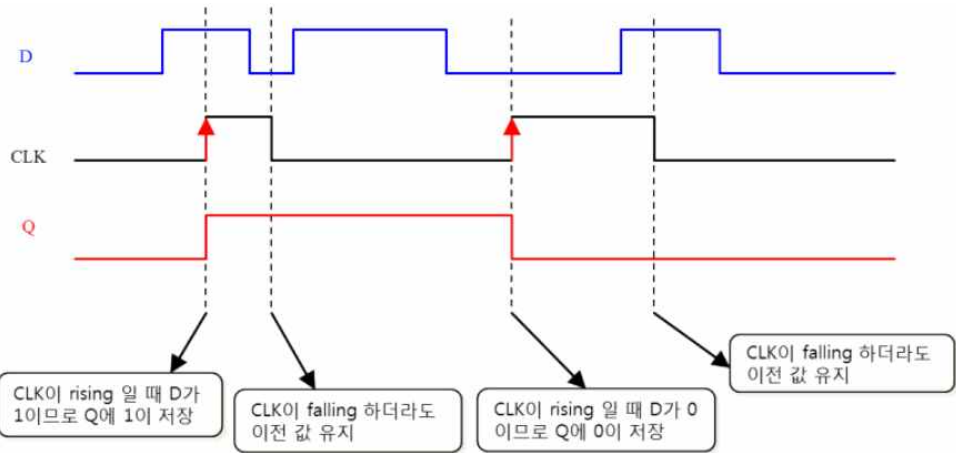
D	CLK (enable)	Q	\overline{Q}	동작
0	1	0	1	Reset
1	1	1	0	Set
X	0	Q_0	$\overline{Q_0}$	상태유지

< 진리표 >



< 블록도 >

< 회로도 >



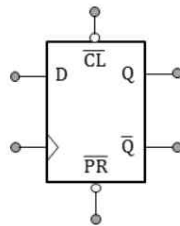
< 시간 차트 >

: 클록형 D래치와 같다.

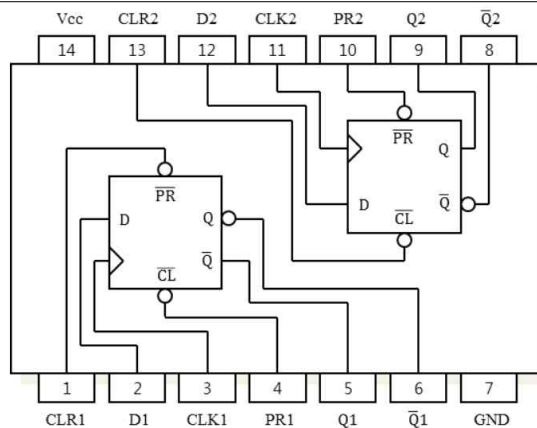
▶ 74LS74 – Dual D-type Flip-Flops(D Flip-Flop의 대표적인 소자)

입력				출력	
\overline{PR}	\overline{CLR}	CLK	D	Q	\overline{Q}
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	1	1
1	1	↑	1	1	0
1	1	↑	0	0	1
1	1	0	X	Q_0	\overline{Q}_0

< 진리표 >



< 블록도 >



< 회로도 >

→ 비동기적 입력 신호 : 클럭 에지와 상관없이, 언제든지 상태를 변경시킬 수 있는 신호이며, PR, CLR 등이 있다.

→ PR(Preset) : 출력을 강제로 HIGH로 설정하는 역할을 한다.

→ CLR(Clear) : 출력을 강제로 LOW로 설정하는 역할을 한다.

→ PR(Preset)과 CLR(Clear)이 모두 HIGH일 때 CLK에 의해 동작

→ 위 소자는 Preset과 Clear가 모두 LOW일 때는 사용하지 않는다.

Preset과 Clear가 모두 HIGH라면 이 때는 CLK의 Rising edge부분을 검출하여 Data의 값을 그대로 출력한다. 만약 CLK이 rising edge외에 HIGH, LOW, falling edge인 경우에는 현재 D값에 상관없이 이전 D값을 유지한다.

▶ T(Toggle) Flip-Flop

T	CLK	Q	\bar{Q}	동작
0	1	Q_0	\bar{Q}_0	상태유지
1	1	\bar{Q}_0	Q_0	상태반전
X	0	Q_0	\bar{Q}_0	상태유지

< 진리표 >

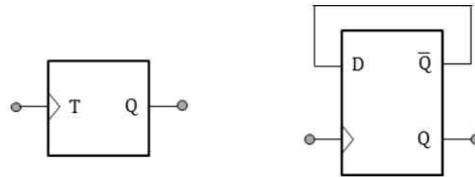
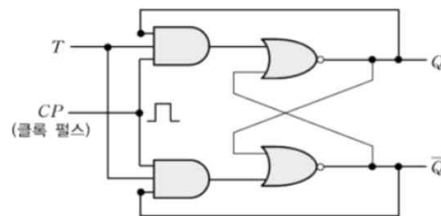
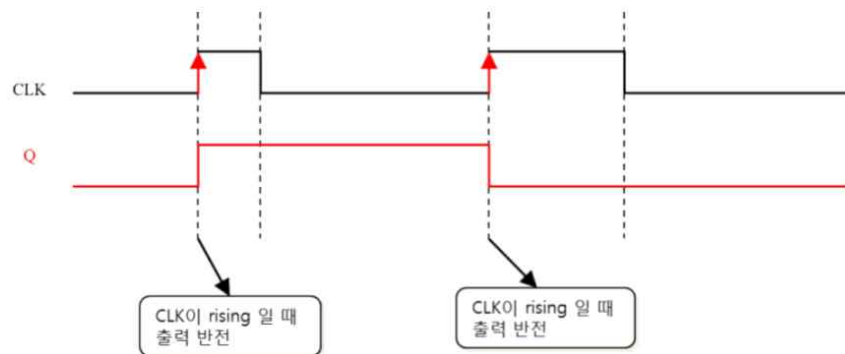


그림. 1 T 플립플롭의 기호

< 블록도 >



< 회로도 >



< 시간 차트 >

$$Q^+ = T \bar{Q} + \bar{T} Q$$

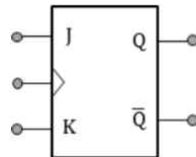
< T-FF의 특성식 >

: 입력단에 신호가 입력될 때마다(HIGH) 출력이 반전(Toggle)되는 회로이다.
D플립플롭의 반전 출력단을 입력에 연결하면 항상 반대 상태의 입력이 들어가게 되어 CLK에 의해 활성화될 때마다 반대 상태가 출력된다. 따라서 D플립플롭을 이용하여 만들 수 있다.

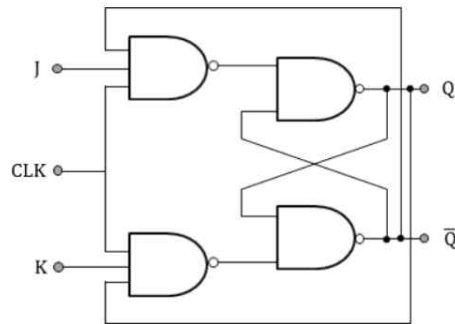
▶ JK Flip-Flop

입력			출력	
J	K	CLK	Q	\bar{Q}
0	0	↑	0	1
1	0	↑	Toggle	
0	1	↑	Q_0	\bar{Q}_0
1	1	↑	1	0
X	X	0	Q_0	\bar{Q}_0

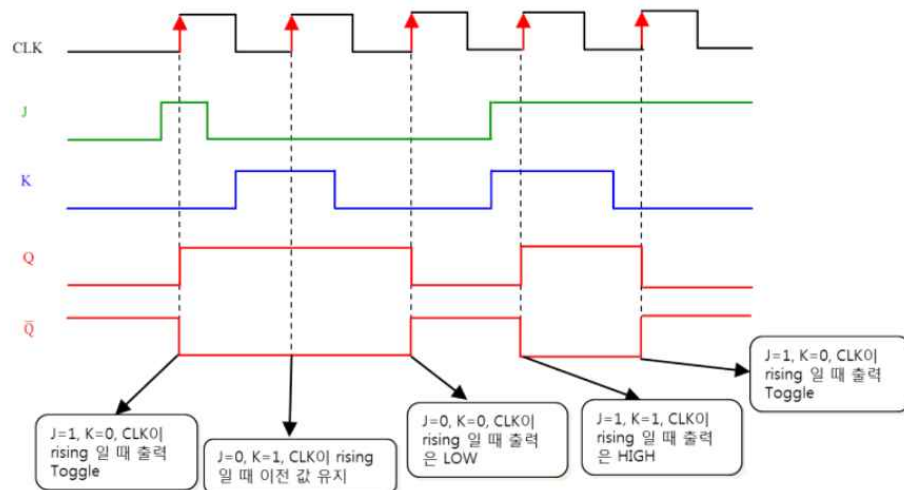
< 진리표 >



< 블록도 >



< 회로도 >



< 시간 차트 >

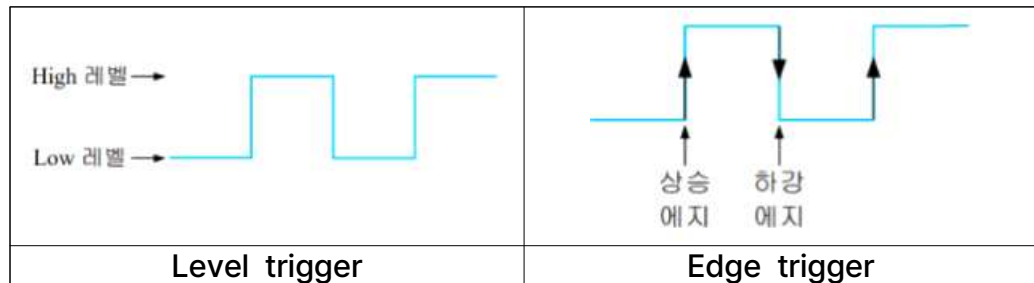
$$Q^+ = J \bar{Q} + \bar{K} Q$$

< JK-FF의 특성식 >

: JK플립플롭은 이전 상태를 유지시켜주는 기능을 가지는 D플립플롭과, 이전 상태를 반전 시켜주는 T플립플롭의 기능을 모두 가진 회로이다.

- Latch와 Flip-Flop의 차이

- ▶ Latch는 레벨 트리거(Level trigger)에 대해 동작한다.
- ▶ Flip-Flop은 에지 트리거(Edge trigger)에 대해 동작한다.
- ▶ 레벨 트리거(Level trigger) : 상태 변수에 해당하는 현재 상황을 기준으로 동작하는 방식이다.
- ▶ 에지 트리거(Edge trigger) : 상태 변수가 변화하는 순간을 기준으로 동작한다.
 - Rising edge(Positive edge) : 클록 펄스 값이 LOW에서 HIGH로 변화하는 순간을 의미한다.
 - Falling edge(Negative edge) : 클록 펄스 값이 HIGH에서 LOW로 변화하는 순간을 의미한다.



2) 각종 순서논리회로

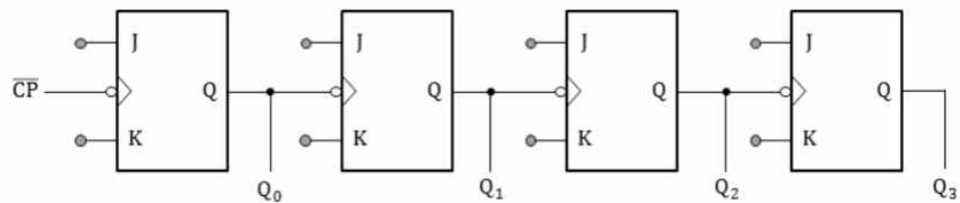
- 카운터(Counter) : 카운터는 일련의 디지털 신호를 계산하고, 그 결과를 저장하는 데 사용되는 전자 회로이다. 카운터는 일반적으로 순차적인 숫자를 생성하며, CLK의 Pulse edge에 따라 플립플롭들에 의해 2진수의 숫자가 하나씩 증가한다. 이를 '업 카운터'라고 하며, 반대로 최대값에서 시작해서 0까지 작동하는 '다운 카운터'도 있다. 카운터는 동기식 카운터(synchronous counter)와 비동기식 카운터(asynchronous counter)로 구분할 수 있다. 동기식 카운터는 클록이 모든 플립플롭에 동시에 가해지는 카운터이며, 비동기식 카운터는 플립플롭의 출력에 의해 전달되는 방식이다.

▶ 카운터의 주요 기능

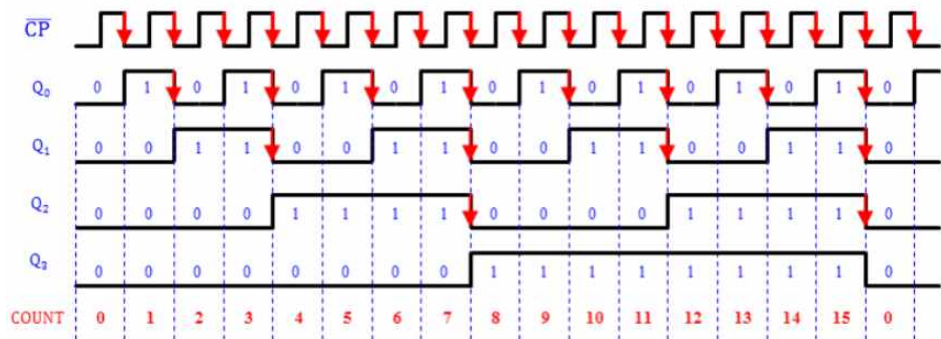
- ① 이벤트 계산 : 특정 이벤트가 발생한 횟수를 계산한다. 예를 들어, 센서에서 발생하는 펄스의 수를 세거나, 버튼이 몇 번 눌렸는지 등을 계산할 수 있다.
- ② 시간 측정 : 특정 시간 간격 동안 얼마나 많은 클록 사이클이 발생했는지 측정하여 시간을 측정할 수 있다. 타이머나 딜레이 생성에 유용하다.
- ③ 주파수 분주 : 고속 클록 신호를 낮은 주파수로 변환하는데 사용된다.

주파수를 변환하여 서로 다른 속도로 동작하는 하드웨어 간에 동기화를 할 수 있다.

▶ 비동기식 카운터(Asynchronous Counter) : 리플 카운터(Ripple Counter)라고도 한다. 각 플립플롭은 이전 플립플롭의 출력에서 클럭 입력을 받는다. 첫 번째 플립플롭만이 외부 클럭 신호를 직접 받으며, 이 구조로 인해 각 플립플롭의 출력이 순차적으로 변경된다. 이로 인해 전체 카운트 값이 '리플'처럼 전달된다. 비동기식 카운터는 설계가 간단하고, 매우 높은 입력 주파수를 다룰 수 있지만, 큰 데이터 비트 크기에 대해서는 전달 지연시간(Propagation delay time) 때문에 문제가 될 수 있다. 리플 카운터의 기본적인 구성은 JK플립플롭을 일렬로 연결하거나 T플립플롭을 일렬로 연결하여 Toggle기능을 활용한다.



< Up Counter의 회로도 >

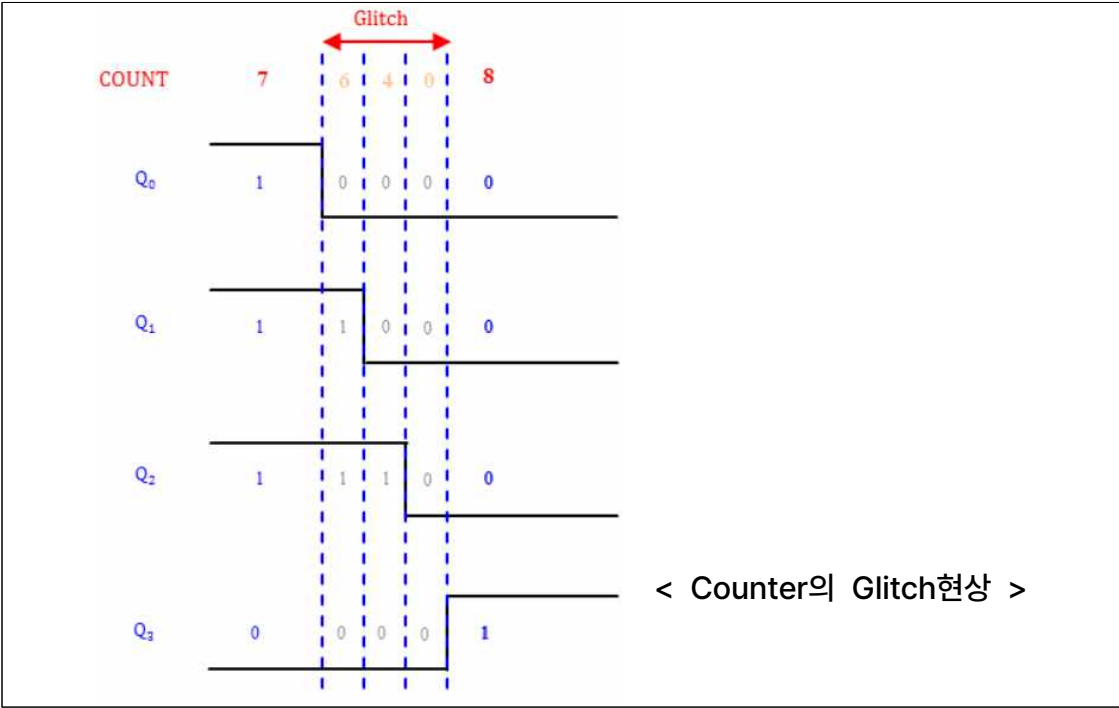


< Up Counter의 시간 차트 >

COUNT	OUTPUTS			
	Q ₃	Q ₂	Q ₁	Q ₀
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H
14	H	H	H	L
15	H	H	H	H

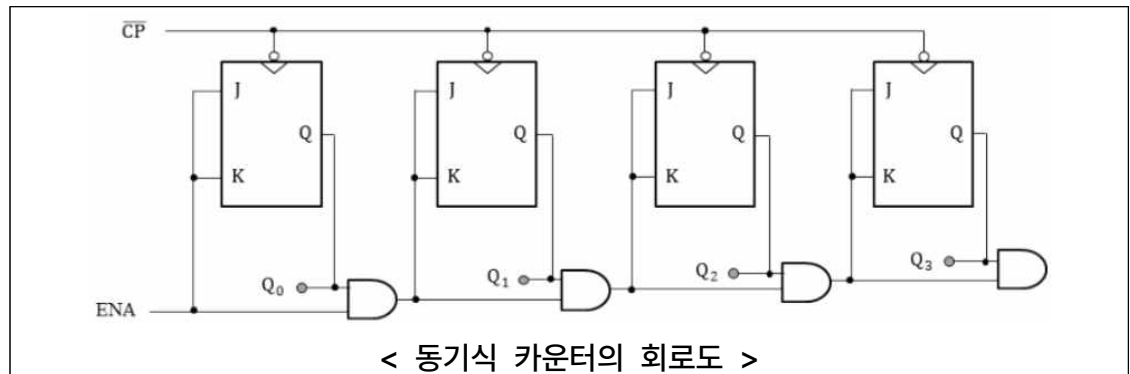
< Up Counter의 진리표 >

▶ 글리치(Glitch) / 리플(Ripple) 현상 : 카운터에서 숫자가 변환될 때 글리치 또는 리플 현상이 발생하는 경우가 있다. 예를 들어, 7에서 8로 넘어가는 순간 출력 4개가 완벽하게 동시에 변환되는 것이 아니기 때문에 중간 과정의 다른 출력값들이 생성된다. 이러한 글리치 현상은 아주 짧은 순간 발생하기 때문에 정밀하지 않은 회로에서는 사용에 지장이 없지만 정밀한 시간 단위로 작동하는 회로에는 영향을 줄 수 있으므로 글리치가 발생할 기간을 예측하여 delay를 주어 글리치를 방지해야 한다.



< Counter의 Glitch현상 >

▶ 동기식 카운터(Synchronous Counter) : 모든 플립플롭들이 동일한 클럭 입력을 공유한다. 즉, 모든 플립플롭들이 동시에(동기적으로) 상태를 바꾼다. 복잡한 로직 회로가 필요하지만, 크게 확장된 경우에도 리플 지연 문제가 없으므로 성능이 좋고 빠르며, 글리치 현상을 방지할 수 있다.



- 레지스터(Register) : 컴퓨터의 중앙 처리 장치(CPU) 내부에 있는 작은 저장 공간으로, 일반적으로 CPU가 요청을 처리하는데 필요한 데이터를 임시로 저장하거나 이동하는데 사용된다. 레지스터는 한 번에 처리할 수 있는 데이터의 양을 결정하는 CPU의 워드 크기와 일치하는 비트 수를 가진다. 예를 들어, 32비트 CPU는 32비트 레지스터를 가진다.
 - ① 데이터 레지스터(Data Registers) : 일반적으로 데이터를 임시로 보관하는데 사용되며, 산술 및 논리 연산에서 피연산자를 저장하는 데 주로 사용된다.
 - ② 주소 레지스터(Address Registers) : 주소 레지스터는 메모리 주소를 저장하는데 사용된다. CPU가 메모리의 특정 위치에 접근할 때 사용된다.
 - ③ 카운트 레지스터(Count Registers) : 명령어 순서 등을 추적하기 위해 숫자를 저장하고 업데이트한다.
 - ④ 명령어 레지스터(Instruction Registers) : 현재 실행중인 명령어를 보관한다.
 - ⑤ 범용 레지스터(General Purpose Registers) : 프로그램에서 필요에 따라 여러 용도로 사용할 수 있는 레지스터이다.
 - ⑥ 상태 플래그(Status Flags) : 이전 연산의 결과에 대한 정보와 상태(ex. 오버플로우, 언더플로우, 제로 등)를 보관한다.
 - ⑦ 특수 목적 레지스터(Special Purpose Registers) : 스택 포인터(Stack Pointer), 프로그램 카운트(Program Counter)와 같이 특정 기능을 수행하기 위해 설계된 레지스터이다.
 - ⑧ 스택 포인터(Stack Pointer) : 스택의 최상위 위치를 가리키는

레지스터이다. 스택은 컴퓨팅에서 매우 중요한 데이터 구조로, 함수 호출과 반환, 지역 변수의 저장 등에 사용된다. 함수가 호출되면 반환 주소와 함께 필요한 정보가 스택에 'push'되고, 함수가 종료되면 그 정보가 'pop'된다. 이 모든 동작에서 스택 포인터는 현재 스택의 상태를 추적하며 언제나 가장 최근에 추가된 항목을 가리킨다.

㉑ 프로그램 카운터(Program Counter) : 프로그램 카운터(PC) 또는 명령어 포인터라고도 부르며, 다음에 실행할 명령어의 주소를 저장하는 레지스터이다. 일반적으로 각 명령어가 실행된 후 PC값은 자동으로 증가하지만, 분기(branch)나 점프(jump)와 같은 제어 전달 명령이 있을 경우 PC값이 변경된다.

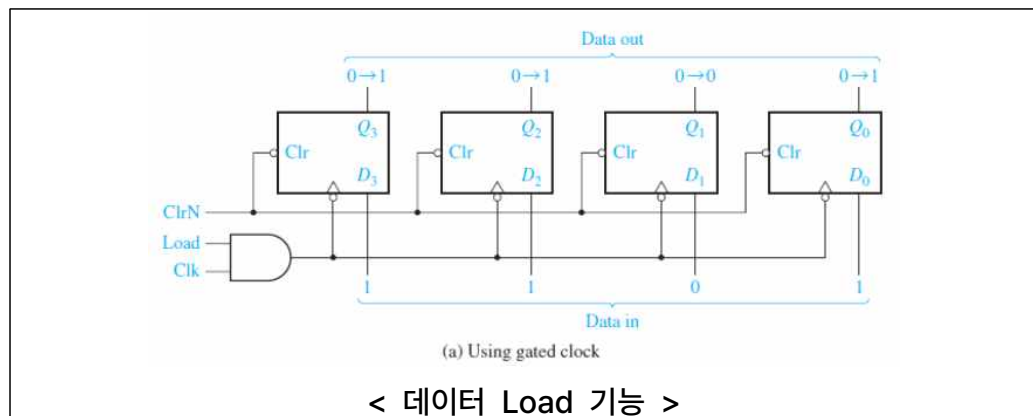
㉒ 링크 레지스터(Link Register) : 주로 서브루틴이나 인터럽트 처리를 수행할 때 복귀에 필요한 반환 주소를 저장하는데 사용된다.

▶ 레지스터의 논리구조

: 레지스터는 여러 개의 플립플롭을 연결하여 구성되어있으며, 주로 D F/F나 JK F/F을 연결하여 구성한다.

n비트 레지스터는 n개의 플립플롭으로 구성되어 있으며, n비트의 2진 정보를 저장한다.

㉑ 레지스터의 데이터 저장 기능

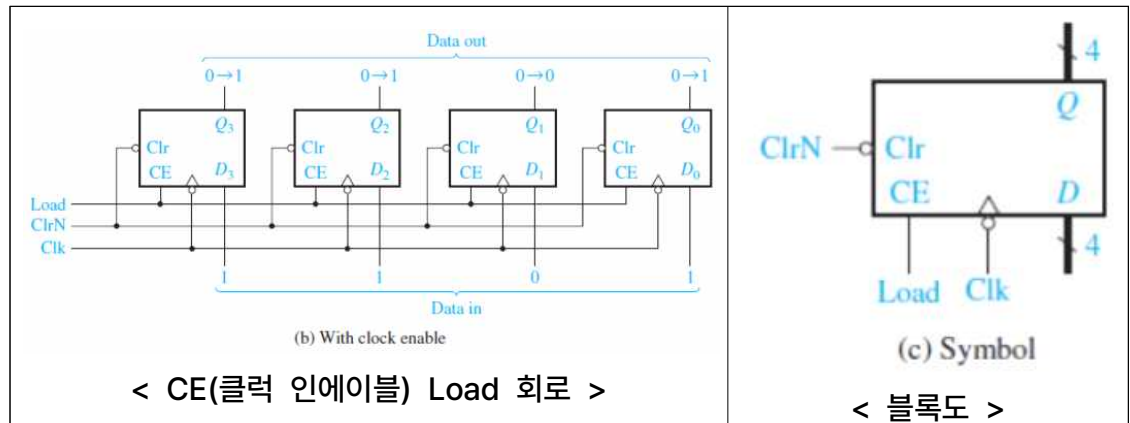


→ 위 회로는 한 비트씩 저장할 수 있는 D F/F 4개가 모여 하나의 레지스터를 이루고 있다.

→ CLK에 버블기호가 붙어 있으므로 하강엣지트리거이며, Load신호와 CLK신호가 AND Gate로 연결되어 있는 F/F이다.

→ $Load = 0$ 일 때, CLK와 상관 없이 AND Gate의 출력은 0이 되고 레지스터는 이전의 값을 그대로 유지한다.

→ $Load = 1$ 일 때, AND Gate의 출력은 CLK가 되고 입력으로 들어온 데이터는 CLK의 하강엣지에서 플립플롭으로 로드된다.



→ 위 회로는 기존 데이터 Load회로에 CE(클럭 인에이블)을 추가한 것이다.

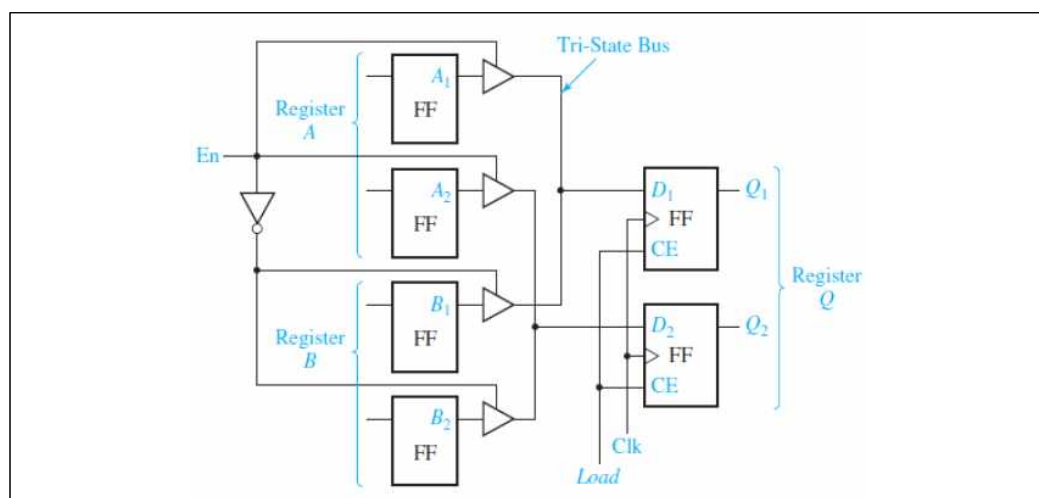
Load 회로의 값에 따라 CLK의 상태를 전환할 수 있다.

→ $Load = 0$ 일 때, $CE = 0$ 이 되고 CLK를 무시하게 되어 레지스터의 값은 이전의 값을 유지한다.

→ $Load = 1$ 일 때, $CE = 1$ 이 되고, 정상작동하여 D입력에 인가된 데이터는 CLK의 하강엣지에서 플립플롭으로 로드된다.

→ 블록도에서 데이터선(D)과 출력선(Q)에 표시된 '4'는 4비트 레지스터임을 나타낸다.

② 레지스터의 데이터 선택 기능



→ 위 회로는 F/F가 두 개로 이루어진 레지스터가 총 3개이다.

→ $En = 1$ 일 때, 3상버퍼에 의해 레지스터 B와 연결이 끊어지고 레지스터

A와 연결된다. 따라서 플립플롭 A_1 의 값이 플립플롭 Q_1 의 D_1 으로

입력되고, 플립플롭 A_2 의 값이 플립플롭 Q_2 의 D_2 로 입력된다.

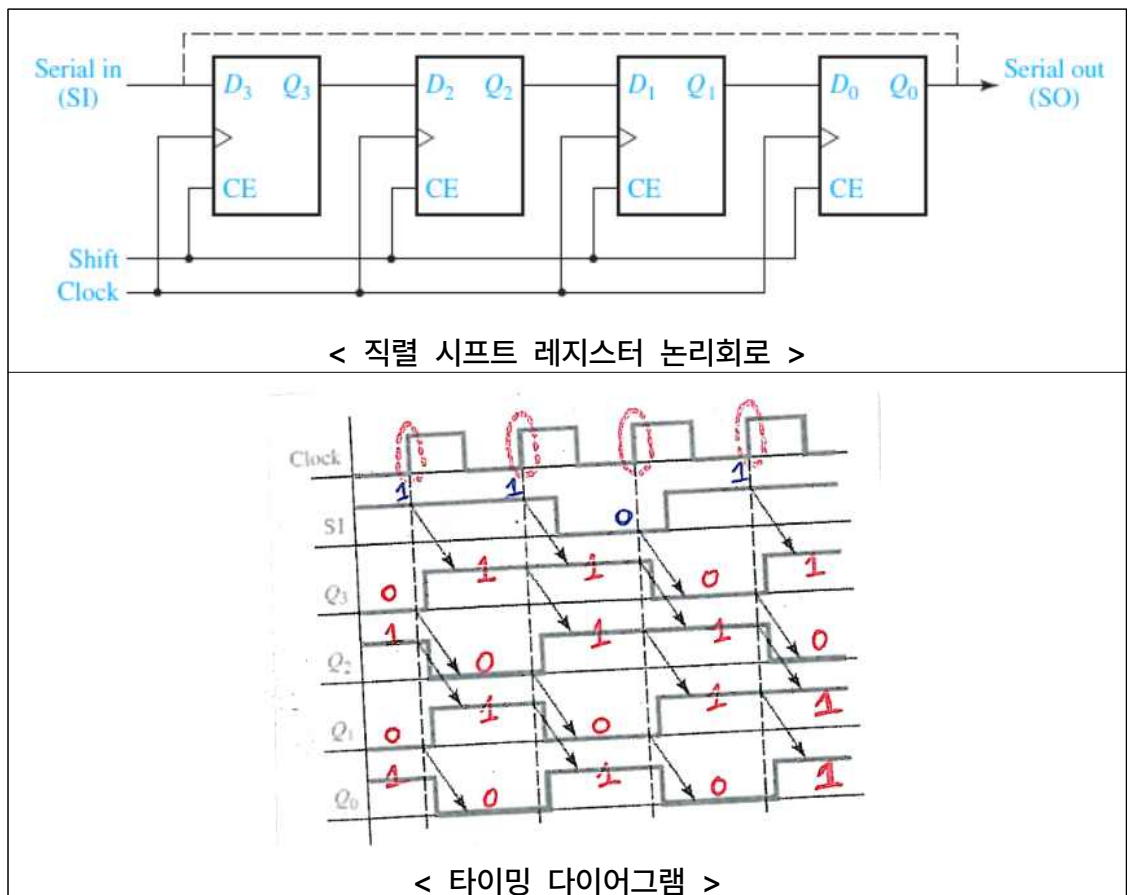
→ 따라서 레지스터 A의 데이터는 CLK의 상승엣지에서(버블 기호가 없으므로) 레지스터 Q로 전달되고 Q_1 , Q_2 값으로 출력된다.

→ $En = 0$ 이면, 레지스터 A와는 연결이 끊기고 레지스터 B와 연결된다.

- 시프트 레지스터(Shift Registers)

: 2진 데이터를 저장하여 시프트 신호가 인가될 때 이 데이터를 오른쪽 또는 왼쪽 방향으로 시프트(1비트씩 이동)할 수 있는 레지스터이다.

▶ 직렬 시프트 레지스터(Serial Shift Registers)



→ 위 회로는 버블기호가 없으므로 상승엣지를 갖는 플립플롭으로 구성되어 있다.

→ $Shift = 1$ 이면 $CE = 1$ 이 되고, 상승엣지에서 직렬 입력(SI)이 첫 번째 플립플롭 D_3 로 입력된다. 동시에 첫 번째 플립플롭의 출력은 두 번째

플립플롭으로 입력되고, 두 번째 플립플롭의 출력은 세 번째 플립플롭으로 입력되고, 세 번째 플립플롭의 출력은 마지막 플립플롭으로 입력된다.

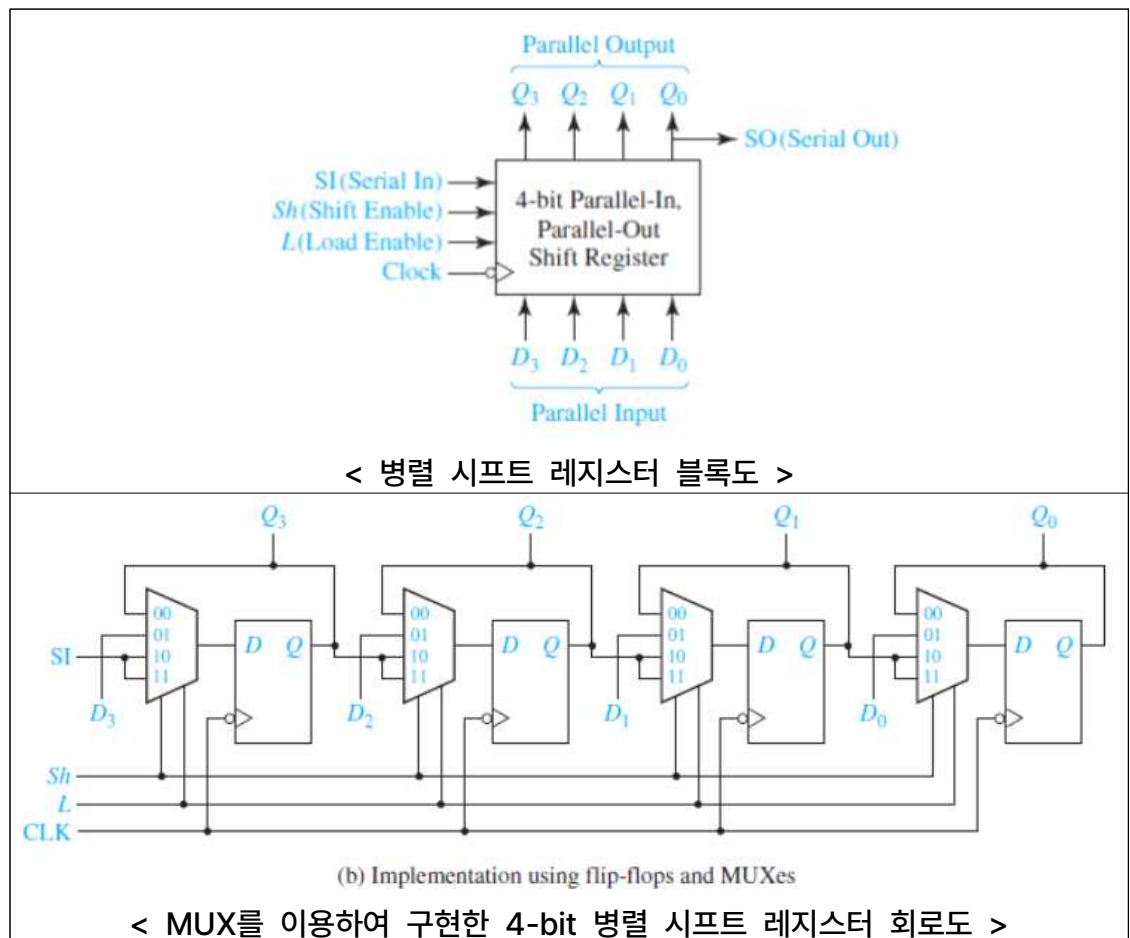
이를 바로 '시프트가 일어난다'라고 표현한다.

→ $Shift = 0$ 이면 $CE = 0$ 이 되고 CLK신호를 무시하기 때문에 시프트 동작은 일어나지 않고 이전 상태를 유지한다.

→ 타이밍 다이어그램에서 레지스터의 초기 값이 0101이고, 직렬입력(SI)에 1,1,0,1을 차례로 넣어준다고 가정했을 때, 상승엣지가 발생하면 SI입력은 Q_3 로 전달되고 $Q_3 = 0$ 의 값은 Q_2 로 전달되고 $Q_2 = 1$ 의 값은 Q_1 으로 전달되고 $Q_1 = 0$ 의 값은 Q_0 로 전달된다.

→ 이와 같이 시프트가 반복하여 일어나면 레지스터는 0101/1010/1101/0110/1011이 순환하여 반복되는 순환형 시프트 레지스터가 된다.

▶ 병렬 시프트 레지스터(Parallel Shift Registers)



→ 블록도를 보면 병렬입력(PI)으로 다수의 비트를 동시에 받아들이고 직렬출력(SO)으로 한 비트씩 내보내거나, 외부에서 직렬입력(SI)으로 한 비트씩 데이터를 받은 뒤 그 값을 한꺼번에 병렬출력(PO)으로 내보낼 수 있다.

→ Sh(Shift enable) : 시프트 허용으로 $Sh = 1$ 일 때 시프트 된다.

L(Load enable) : 로드 허용으로 $L = 1$ 일 때 데이터가 들어온다.

→ MUX(Multiplexer)를 보면 00,01,10,11 숫자가 쓰여 있는데, 앞 숫자는 Sh값, 뒤의 숫자는 L값을 의미한다.

→ $Sh = L = 0$ 이면 $Q_3 = Q^+$ 으로 현재값이 유지된다.

$Sh = 0, L = 1$ 이면 4개의 데이터들이 플립플롭에 로드된다.

$Sh = 1$ 이면 L값과 관계없이 직렬입력(SI)이 플립플롭으로 로드되면서 오른쪽으로 시프트된다.

입력		다음 상태				동작
Shift	Load	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
0	0	Q_3	Q_2	Q_1	Q_0	불변
0	1	D_3	D_2	D_1	D_0	Load
1	X	SI	Q_3	Q_2	Q_1	오른쪽 Shift

< 4-bit 병렬 시프트 레지스터 동작 진리표 >

$$Q_3^+ = \overline{Sh} \times \overline{L} \times Q_3 + \overline{Sh} \times L \times D_3 + Sh \times SI$$

$$Q_2^+ = \overline{Sh} \times \overline{L} \times Q_2 + \overline{Sh} \times L \times D_2 + Sh \times Q_3$$

$$Q_1^+ = \overline{Sh} \times \overline{L} \times Q_1 + \overline{Sh} \times L \times D_1 + Sh \times Q_2$$

$$Q_0^+ = \overline{Sh} \times \overline{L} \times Q_0 + \overline{Sh} \times L \times D_0 + Sh \times Q_0$$

< 4-bit 병렬 시프트 레지스터 차기 상태 방정식 >

< 4-bit 병렬 시프트 레지스터의 타이밍 다이어그램 >

3) 순서논리회로의 분석과 설계

▶ 순서논리회로 분석

- (1) 회로 구성 요소 파악 : 회로가 어떤 논리 게이트와 플립플롭으로 구성되어 있는지 파악한다.
- (2) 입력 및 출력 신호 이해 : 각각의 입력 신호와 출력 신호가 무엇을 의미하는지 확인한다.
- (3) 상태 전이도 작성 : 각 가능한 상태에서 다음 상태로 어떻게 전환되는지를 나타내는 상태 전이도(State Transition Diagram)를 작성한다.
- (4) 진리표 생성 : 각 상태에서 입력에 대한 출력을 나타내는 진리표를 만든다.

▶ 순서논리회로 설계

- (1) 문제 정의 : 설계하려는 회로가 해결해야 할 문제나 수행해야 할 기능을 명확하게 정의한다.
- (2) 상태 정의 및 전이도 작성 : 필요한 모든 상태를 정의하고, 각 가능한 입력 조합에 대해 다음 상태와 출력을 결정하는 상태 전이도를 작성한다.
- (3) 진리표 및 카르노맵 생성 : 설계된 기능에 대해 진리표와 카르노맵을 생성하여 최소화된 불 연산식을 찾는다.
- (4) 회로 구현 : 마지막으로, 찾아낸 최소화된 불 연산식과 플립플롭 타입(D-FF, JK-FF 등)으로 실제 회로를 구현한다.

4. 메모리

1) 각종 메모리

▶ RAM(Read-only Memory)

- 메모리의 주소에 해당하는 위치에 데이터를 읽거나 쓰는 것을 액세스(access)라고 한다.
- RAM은 프로그램이 실행되는 동안 필요한 정보를 저장하는 컴퓨터 메모리(휘발성 메모리)이다. 저장된 데이터를 순차적이 아닌 임의의 순서로 액세스 할 수 있는 저장소이므로 접근시간은 어느 위치나 동일하기 걸리는 메모리이다.
- SRAM(Static RAM) : 일반적으로 2진 정보를 저장하는 내부 플립플롭으로 구성되며, 저장된 정보는 전원이 공급되는 동안 보전된다. DRAM과 비교했을 때 사용하기 쉽고 읽기와 쓰기 사이클이 더 짧다.
- DRAM(Dynamic RAM) : 2진 정보를 커패시터에 공급되는 전하 형태로 보관한다. 그러나 커패시터에 사용되는 전하는 시간이 지나면 방전되므로 일정한 시간 안에 재충전(refresh)해야 한다. SRAM보다 전력소비가 적고 메모리 칩에 더 많은 정보를 저장할 수 있다(집적도가 높다).

▶ ROM(Read-only Memory)

- 판독만 가능한 메모리로, 기록된 정보는 전원이 꺼져도 지워지지 않으므로(비휘발성 메모리) 프로그램이나 문자 패턴 등 고정된 정보의 기억에 사용된다. 단, EEPROM이나 Flash Memory는 ROM의 종류이나 필요할 때마다 다시 프로그래밍하여 데이터를 변경 가능하다.
- EEPROM(Electically Erasable Programmable ROM) : 전기적인 방법으로 데이터를 지우고 다시 기록할 수 있는 ROM의 한 종류이다.

바이트 당 삭제 및 재작성 기능을 제공한다. 즉, 개별 바이트를 독립적으로 수정할 수 있다.

- Flash Memory : EEPROM에서 발전한 형태로서, 블록 당 삭제 및 재작성 기능을 제공한다. 따라서 대량의 데이터를 더욱 횡수와 속도 면에서 유용하며 USB드라이브나 SD카드 등에 널리 사용된다.
- NAND Flash Memory : 데이터를 블록 단위로 읽고 쓰며, 대용량 데이터 저장에 적합하다. NAND 타입은 셀당 비트 수가 많아질수록 용량 대비 가격이 낮아져서 대부분의 SSD, USB, SD카드 등에 널리 사용된다. 그러나 비교적 느린 읽기 속도와 한정된 쓰기/지우기 사이클 때문에 문제가 발생하여 이를 보완해야 한다.
- NOR Flash Memory : 바이트 단위로 데이터를 읽고 쓸 수 있는 메모리로 실행 가능 코드(XIP : eXecute In Place)저장에 주로 사용된다. 이는 CPU가 직접 NOR 플래시에서 코드를 실행할 수 있음을 의미한다. 따라서 NOR 타입은 부팅 로더나 임베디드 시스템에서 BIOS 등의 중요한 코드 저장 공간으로 사용된다. 그러나 NAND 타입에 비해 용량당 가격이 상대적으로 높고 전력 소비도 크다.

2) 메모리의 동작

▶ 메모리 동작의 기본 개념

: 컴퓨터는 이진 1과 0으로 값을 저장하기 때문에 기억 소자는 상태가 ON인지 OFF인지 수준으로 기억한다. 반도체 기억장치의 기본 요소는 기억 소자(memory cell)이며, 모든 반도체 기억 소자들이 가지는 공통적인 성질이 있는데, 0과 1의 두 개의 안정된 상태를 갖는다는 것이다. 1과 0이다. 각 기억소자는 상태를 세팅할 수 있도록 쓰여질 수 있다. 반대로 상태를 감지할 수 있도록 읽혀질 수도 있다. 쓰는 동작인 경우는 기억 소자의 상태를 1또는 0으로 만들어 준다는 의미이고 읽기 동작이라는 것은 그 소자가 갖는 현재 상태가 어떠한 상태인지를 알아오는 개념이다.

▶ 메모리 동작의 원리

: 메모리는 정보를 저장(write)하거나 저장된 정보를 읽기(read) 위하여 바둑판과 같이 열(raw, 가로 줄)과 행(column, 세로 줄)으로 구성된 matrix(행렬) 구조의 주소(address)를 가지고 있다. 이를 두고 CAS(Column Address Strobe)와 RAS(Row Address Strobe)라 부르는데 프로세서가 메모리에 있는 정보를 읽거나 메모리에 정보를 기록할 때는 먼저 가로줄에

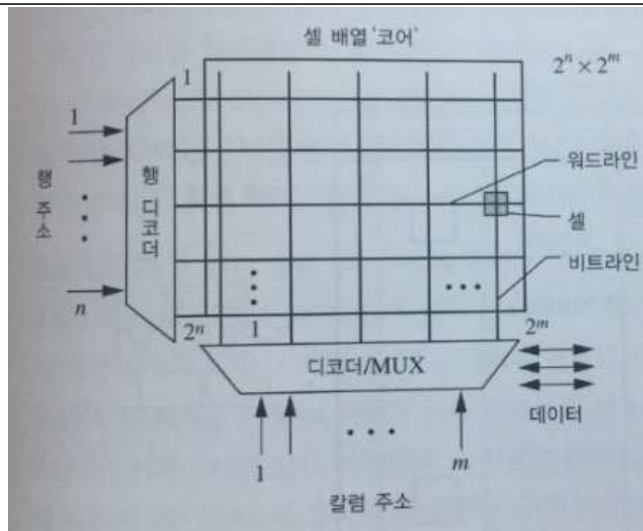
신호(RAS)를 보내고 나서 세로줄에 신호(CAS)를 보내어 주소를 확인한다. 어떤 주소에 데이터가 들어 있는지 아니면 비어 있는 지는 CAS가 담당하며 CAS 신호가 없어지면 그 주소에 다시 새로운 정보를 저장한다.

▶ 메모리 동작 과정

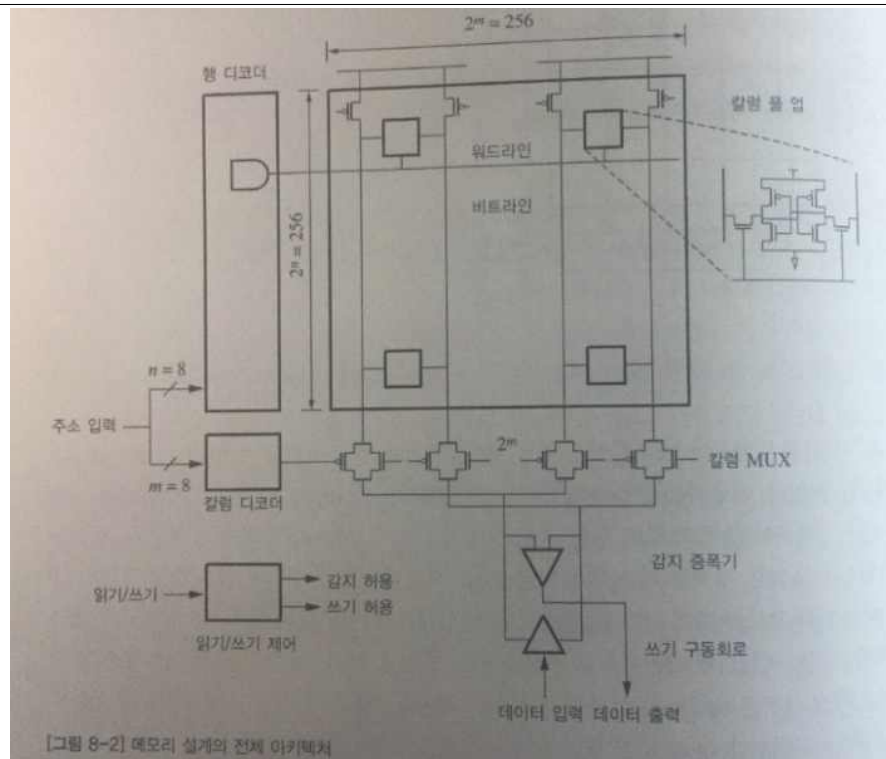
- (1) CPU가 메인보드 칩셋에 데이터를 요청하면 메인보드 칩셋은 그 데이터가 있는 곳의 행(row, 가로줄) 주소를 메모리로 보낸다. 이 동작에는 각각 1-Cycle(1Hz)이 걸린다.
- (2) 행 주소가 메모리의 행 주소 버퍼로 들어오면 센스 앰프(sense amp)가 그 행에 들어있는 모든 셀을 읽어낸다. 이렇게 행 부분을 읽는 신호는 RAS(row address strobe, 행 주소 스트로브)라고 부르고, 읽는데 걸리는 시간은 RAS-to-CAS delay(RAS와 CAS 사이의 지연 시간)라고 한다. 이 과정은 2~3 Cycle이 걸린다.
- (3) 행 주소만으로는 필요한 데이터가 어디 있는지 알 수 없으니 이번에는 열(세로 줄) 주소를 받는다. 그러면 CAS(column address strobe, 열 주소 스트로브) 신호가 일어나 정확한 열을 찾아낸다. 이 때 걸리는 시간을 CAS latency(CAS 지연 시간)라고 한다. 이것을 하는데도 역시 2~3 사이클이 걸린다.
- (4) 정확한 행과 열을 찾으면 필요한 데이터를 얻을 수 있고, 메모리 셀에 있는 내용이 출력 버퍼(output buffer)로 옮겨진다. 이 동작에는 1-Cycle이 걸린다.
- (5) 마지막으로 메인보드 칩셋이 출력 버퍼의 내용을 읽고 CPU로 전달한다. 이 때 각각 1-Cycle씩 모두 2-Cycle이 걸린다.

3) 메모리 회로 분석 및 설계

▶ RAM 회로 구성



< 메모리 시스템의 기본 구성 >

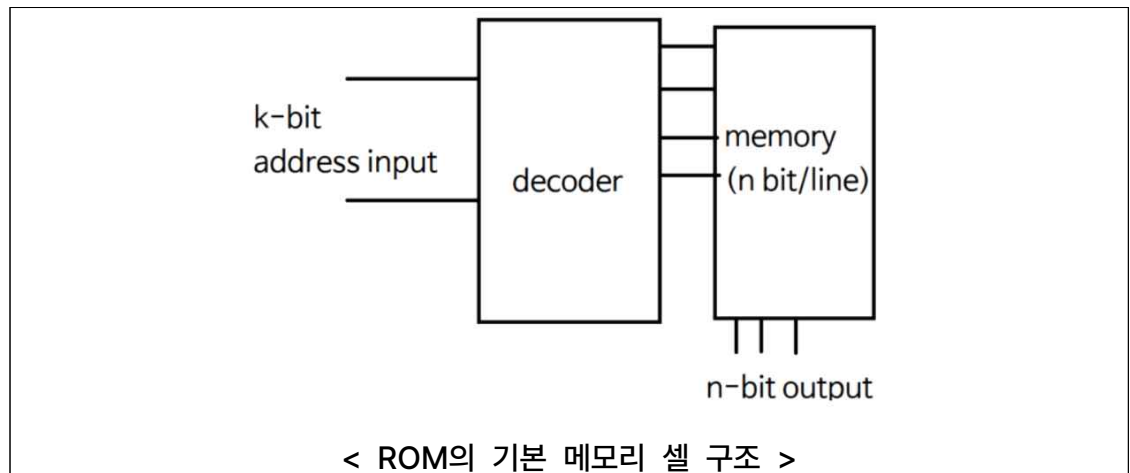


< 64KB RAM Architecture >

- 기본적인 RAM의 회로 구성은 행 선택을 위한 n비트 디코더와 열 선택을 위한 m비트 칼럼 디코더를 가지고 비트라인 접근신호를 생성한다.
- 64KB RAM은 $256 \times 256 = 65536$ 개의 셀을 보유하고 있다. 이 메모리는 1비트의 출력을 위해 16비트 주소를 사용한다.
- 데이터의 저장 기능은 꼭 유지해야 하며, 진폭 양자화/로직 레벨의 재생/입·출력의 분리/팬 아웃 구동 능력과 같은 다른 특성은 셀의 간략화를

위해 희생될 수 있다.

▶ ROM회로 구성



- ROM은 K개의 주소 입력과 N개의 데이터 출력 라인을 갖는다. 즉, ROM에는 2^k 개의 n비트 데이터를 저장하는 데이터 라인을 가질 수 있다. ($2^k \times n$ 크기의 행렬)
- k개의 입력 신호(주소)가 들어오면, 먼저 Address Decoder를 거쳐 2^k 개의 데이터 라인 중 하나로 신호가 전달되고, 적절한 메모리 상의 데이터 라인이 출력된다.

5. HDL

- : 하드웨어 설계를 기술하기 위해 사용되는 컴퓨터 언어이다. 예시로 VHDL과 Verilog가 있으며 이들 언어를 사용하여 FPGA나 ASIC등의 디지털 시스템을 설계 및 모델링을 할 수 있다.
- ASIC : Application Specific IC의 약자로 임베디드 시스템 분야에서 양산된 제품, 특히 소형기기를 중심으로 임베디드에 요구되는 비용제약에 대응하기 위해서 반도체 소자로서 많은 시스템의 요소를 단일 칩으로 통합하여 SoC(System on a Chip)화 하는 경향이 있고 이런 전용 반도체를 ASIC라고 한다.
- ASIC의 SoC화가 가능한 이유
 - (1) 반도체의 소형화가 진행되어 칩에 많은 회로를 탑재할 수 있다.
 - (2) 설계 지원 툴의 발전으로 복잡한 대규모 회로도 실현할 수 있다.
 - (3) 고속 클럭 회로는 별개의 칩으로 분할하면 배선이 길어지고 동작하지 않는다.
 - (4) 고속 회로의 전원/신호전압은 저전압화되어 있으므로 같은 칩에 설계한다.

(5) 실제 기기의 시뮬레이션 개발/검증도 가능하다.

(6) 설계 툴에 축적된 과거의 설계 자산을 적극적으로 재사용하도록 한다.

1) 프로그래머블 로직, FPGA

▶ 프로그래머블 로직(Programmable Logic)

: 사용자가 특정 작업을 수행하도록 하드웨어 로직을 설정하거나 변경할 수 있는 전자장치를 의미한다.

→ FPGA(Field-Programmable Gate Array) : 많은 수의 논리게이트를 포함하고 있으며 사용자가 필요에 따라 연결 및 구성할 수 있는 반도체 장치이다. 이는 사용자가 프로그래밍 가능한 로직 블록과 회로를 필요에 따라 구성함으로써 ASIC와는 달리 제작과정이 필요하다. FPGA의 큰 장점은 하드웨어를 직접 변경하지 않고도 새로운 기능을 추가하거나 기존 기능을 수정할 수 있다는 점이다.

2) Verilog, VHDL

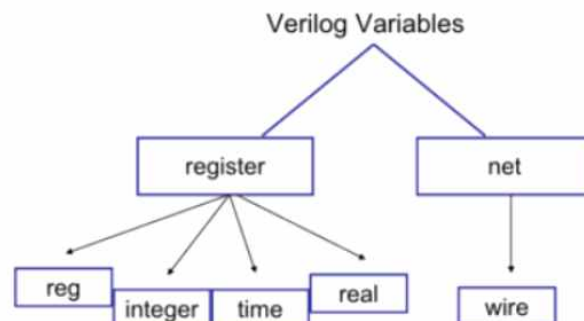
▶ Verilog

: 전자 회로 및 시스템에 쓰이는 하드웨어 기술 언어 중 하나이며, 회로를 설계할 때 사용하는 언어이다. 하드웨어는 순차적으로 돌아가지 않고 Clock에 따라서 동시에 동작하므로 시간과 동시성을 표현할 수 있고 컴파일 과정이 일반적인 프로그래밍 언어와 다르다.

▶ Verilog의 기본 문법

: 기본적인 문법은 C언어와 유사하다. if, for, while과 같은 제어문을 사용할 수 있고 세부적인 문법에서는 차이점이 존재하지만 대부분 유사하게 사용가능하고 연산자도 거의 비슷하게 사용할 수 있다. 한 가지 차이점은 종괄호 대신에 begin ~ end를 사용하여 묶어준다.

→ 데이터 타입



: 베릴로그의 데이터 타입으로는 기억소자인 Register와 연결하는 선인 Net으로 구성되어 있다. Register는 값을 받아서 저장할 때 사용되고 Net은 신호를 연결할 때 사용한다. 일반적으로 Input에는 reg타입을

사용하고 Output에는 wire타입을 사용한다.

```
wire A;
wire [7:0] databus;
wire [31:0] busA, busB, busC;
reg [0:31] address;

reg [0:7] B1, b2, b3, b4;
B1 = address[0:7];
B2 = address[8:15];
B3 = address[16:24];
```

: 벡터형으로 변수 앞에 [7:0], [0:31]과 같이 표현할 수 있다.

wire [7:0] databus 변수의 경우 8-bit wire타입이고 변수명은 databus이다.

reg [0:31] address 변수의 경우에는 32-bit이고 reg타입의 변수명은 address이다. [7:0] bus와 [0:7] bus의 차이점은 각각 상위비트(MSB)가 bus[7], bus[0]이다. reg [0:7] B1, B2, B3를 보면 address의 벡터 중에서 일부를 골라서 사용할 수 있다는 것을 의미한다.

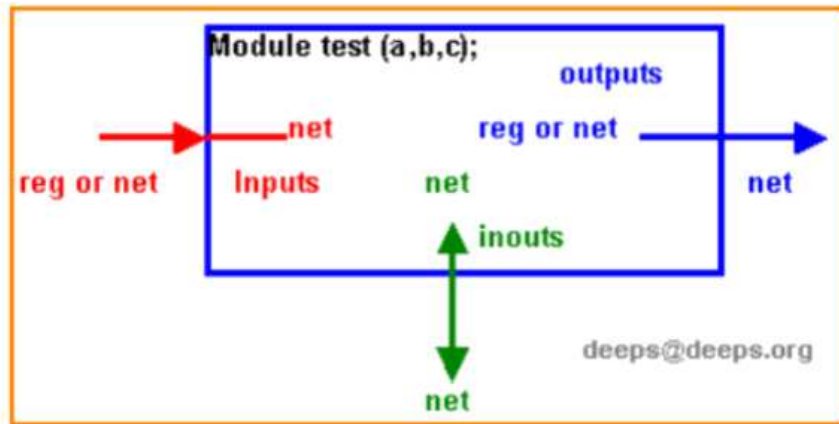
```
module adder(
    output sum,
    output c_out,
    input a, b, c_in
);
```

```
module adder(
    sum,
    c_out,
    a, b, c_in
);
input a, b, c_in;
output sum, c_out;
```

: 각 변수들을 입출력으로 사용하는 방법은 변수명 앞에 input과 output을 넣어주면 된다. 입출력을 동시에 사용하는 변수에 경우 inout을 사용하면 된다.

위 코드는 adder라는 이름의 모듈을 선언한 뒤 괄호 안에 있는 변수들은 이 모듈의 입출력 포트를 나타낸다. 그 뒤 세 개의 입력 포트와 두 개의 출력 포트를 선언한다.

▶ Verilog의 Port Declaration



: input의 경우 외부에서 들어오는 형태는 상관 없지만, 내부에서는 net(wire)타입이어야 한다.

inout의 경우 외부에서 net타입으로 들어와야 하고 모듈 내부에서도 net타입이어야 한다.

output의 경우 내부에서는 reg나 net 상관 없지만 외부에서는 net으로 받아야 한다.

► Verilog Port Connection Rules

```
//Position Association
adder adder1(sum, c_out,a, b, c_in);
//Named Association
adder adder2(.sum(sum), .c_out(c_out), .a(a), .b(b), .c_in(c_in));
```

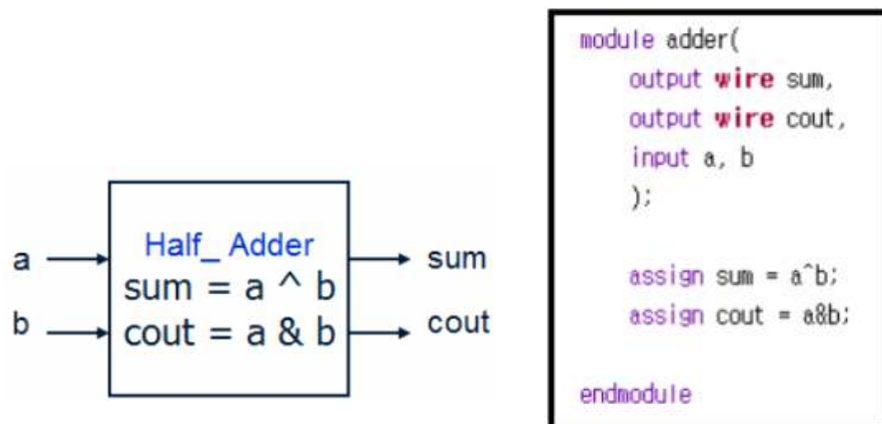
: 각각의 모듈(Component)을 가져오는(Instantiation) 방법에는 2가지가 존재한다.

Positional Association: 내연적인(Implicit) 방법으로 각 변수들의 위치에 따라서 연결하는 방법이다.

Named Association: 외연적인(Explicit) 방법으로 이름에 따라서 연결하는 방법이다.

위의 예시에서 볼 때, adder 모듈에는 sum, c_out, a, b, c_in 5개의 변수가 존재하는데 순서대로 변수를 넣을 경우 Positional Association(adder1)이고 이름에 따라서 넣을 경우 Named Association(adder2)이다.

► Verilog의 모듈



: 모듈은 특정한 기능을 하는 프로그램으로 생각해도 좋고 프로그래밍에서의 함수와 비슷한 개념이다.

베릴로그에는 함수(function)과 테스트(task)와 서브모듈(submodule)이 존재하므로 실제로는 함수, 테스트, 서브모듈 -> 모듈 -> 프로그램(FPGA)라고 생각하면 된다.

위의 예시는 반가산기 모듈을 만든 것이다.

모든 변수의 경우 wire타입으로 구성되어 있고 수식에 맞게 출력 값에 입력 값을 연산하여 대입해주었다.

위 모듈이 맞게 동작하는지 확인하는 방법으로는 테스트벤치를 이용하여 시뮬레이션 하는 방법이 있다.

▶ VHDL(VHSIC Hardware Description Language)

: 디지털 시스템과 통합 회로를 설계하고 모델링하는데 사용되는 언어이다. 이 언어는 고성능 통합 회로(VHSIC, Very High Speed Integrated Circuit)프로그램을 위해 개발되었다.

▶ VHDL의 특징

- (1) 강력한 모델링 기능: VHDL은 비동기 및 동기 디지털 시스템, 그리고 병렬 및 순차적 로직 등 다양한 유형의 하드웨어를 모델링할 수 있다.
- (2) 타입 검사: VHDL에는 강력한 타입 검사 기능이 있으며, 이는 설계 오류를 조기에 발견하고 수정하는 데 도움이 된다.
- (3) 표준화: VHDL은 IEEE 1076 표준으로 정의되었으며 이는 전세계적으로 인정받는 표준이다.
- (4) 대규모 설계 지원: VHDL에서 제공하는 패키지, 구성 요소 및 구성 등의 개념을 사용하여 대규모 설계를 쉽게 관리할 수 있다.
- (5) 시뮬레이션과 합성 지원: VHDL 코드는 논리 회로나 디지털 시스템의 동작을 시뮬레이션하거나 합성하기 위해 사용된다. 합성 도구는 VHDL 코드를 읽고 해당하는 하드웨어 (예: FPGA 또는 ASIC)에 매핑할 수

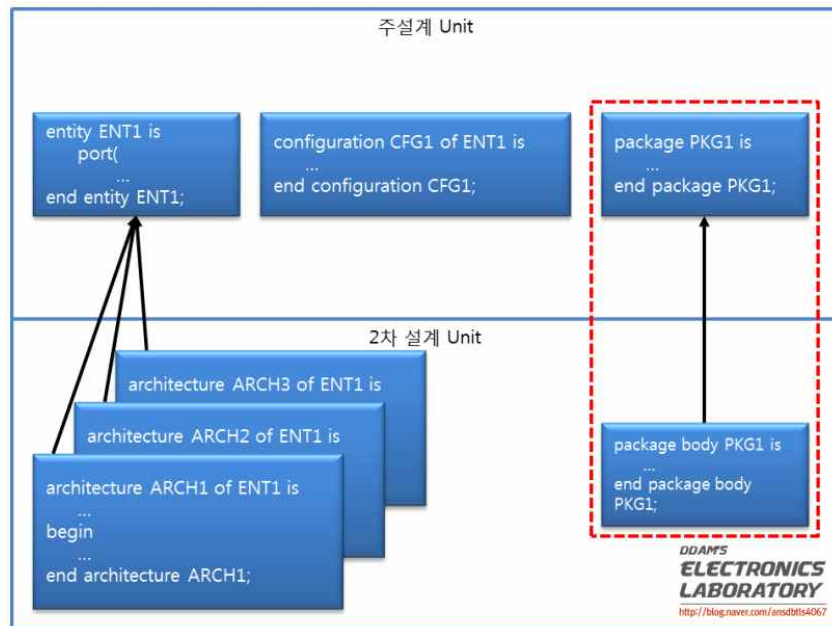
있는 게이트 레벨 설명을 생성한다.

- (6) 가독성과 이식성: 실제 하드웨어와 유사한 형태로 로직을 기술할 수 있으므로 가독성이 좋다. 또한, 어떤 종류의 칩으로도 이식 가능하므로 장비 간 호환성 문제를 최소화 할 수 있다.

▶ VHDL 프로그래밍의 기본구조

→ VHDL은 DSP나 MCU 제어를 목적으로 설계되는 임베디드 C언어의 코딩에서와 비슷한 프로그래밍의 기본 구조가 있다.

(1) Entity 설계



: VHDL로 디지털 시스템을 설계할 때에는 컴포넌트라고 정의할 수 있는 블록으로 나누어 설계할 필요가 있다. 각 컴포넌트는 설계 Entity의 오브젝트를 참조한 것이며, 일반적으로 별도의 파일로 나누어 시뮬레이션과 합성 툴에서 각각 독립적으로 컴파일할 수 있게 한다.

예를 들어 전체 회로를 계층구조로 설계했다고 하면, 하나의 설계 Entity에서 참조한 하위의 Entity들을 부설계 Entity라고 한다.

설계한 디자인 Entity를 합성할 때, 설계자는 시스템을 적당한 크기의 Entity로 분류하여야 하며, 합성 시 2입력 NAND 게이트를 기준으로 최대 5000 Gate 정도 이내인 경우 최적으로 합성될 것이다.

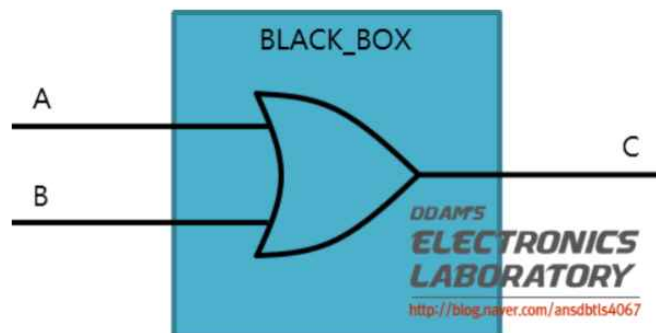
(2) 주석문(Comments)

```
-- Company:
-- Engineer:
--
-- Create Date:    16:51:19 01/11/2018
-- Design Name:
-- Module Name:    TEST02_MUX - Behavioral
-- Project Name:
```

: VHDL에서 주석문을 코드 내에 삽입하고자 할 때는 주석문 앞에 두 개의 하이픈('--')을 쓴다. 이와 같이 하이픈 두 개가 연속하여 있는 경우 컴파일러는 하이픈이 있는 곳부터 그 줄의 끝까지 주석문으로 간주하고, 컴파일 시에 무시하고 넘어가게 된다.

주석문은 가독성을 증가시키고, 이식성을 좋게 하기 위해서는 가능한 한 많은 설명문을 삽입하는 것이 좋다.

(3) Entity



: Entity는 디지털 회로를 설계하는데 있어서 가장 기본이 되는 정의를 담고 있으며 이는 설계할 디지털 회로를 블랙박스로 놓았을 때 최외각에서 보이는 모양을 기술하는 것으로 보면 이해하기 좋다. 예를 들어 A라는 것을 B라는 것과 논리합을 하여 C로 출력하는 논리회로를 설계한다고 가정해보면, Entity는 논리회로 내부가 어떻게 생겼는지는 관심이 없고 오직 그 외부에 어떤 것이 보이는가만 중요하게 여긴다.

즉, Input Port인 A,B와 Output Port인 C에 관심을 가진다.

Entity의 정의에서는 설계하고자하는 디지털 회로의 블록의 이름 즉 Entity이름과 외부와의 인터페이스인 Port를 표시한다.

```

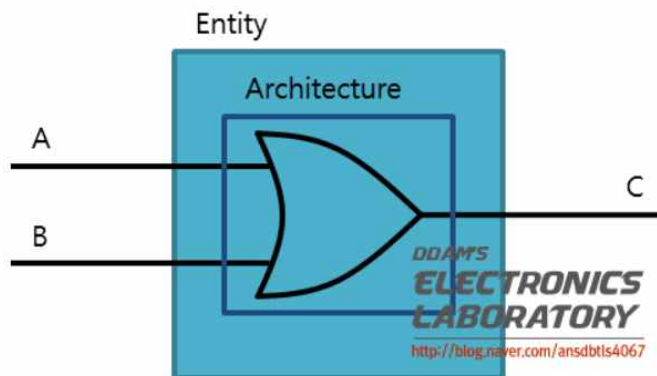
entity BLACK_BOX is
    port(A, B : in  std_logic;
          C      : out std_logic);
end entity BLACK_BOX;
entity 'Entity 이름' is
    port('Port 이름' : '모드' '데이터형';
          'Port 이름' : '모드' '데이터형');
end entity 'Entity 이름';

```

: Entity의 이름은 entity “이름” is로 정의한다. Port의 정의는 port()로 정의하며 Port의 특성을 Port 이름에 연이어 콜론(:) 다음에 기술하며 각기 다른 특성의 Port는 세미콜론(;)로 구분한다. 입력 Port는 'in', 출력 Port는 'out'으로 모드에 정의해 준다. 모드는 in(입력), out(출력), Inout(양방향), buffer(출력으로 나가는 신호를 내부에서 다시 입력으로 피드백), linkage 총 5개가 있다.

Entity 이름은 첫 자가 반드시 영문자로 시작하여야 하고, 나머지는 영문자, 숫자 또는 밑줄('_')이 사용될 수 있다. 단 밑줄은 이름의 마지막에 사용할 수 없으며, 또한 두 개 이상의 밑줄이 연속되어도 안된다. 이러한 이름의 규칙을 식별자 작성규칙이라고 한다. 여기서 식별자는 대문자와 소문자의 구별이 없다.

(4) Architecture



: Entity는 단순히 껍데기이며, 그 세부동작을 기술하는 부분이 Architecture이다. Architecture는 실질적인 내부 동작 및 부품간의 연결 구조를 기술한다. Entity 하나에 여러 Architecture가 존재할 수 있다.

```

-- BLACK_BOX에 대한 VHDL 코드 구현
-- library 선언
library ieee;
use ieee.std_logic_1164.all;
-- entity 선언
entity BLACK_BOX is
    port(A, B : in  std_logic;
         C    : out std_logic);
end entity BLACK_BOX;
-- architecture 선언
architecture Behavioral of BLACK_BOX is
begin
    process(A, B)
    begin
        C <= A or B;
    end process;
end architecture Behavioral;

```

: 위 코드에서 BLACK_BOX Entity의 실제 동작을 하는 것은 Architecture인 Behavior이다. 이 Architecture의 begin-end 사이에는 실제 논리회로의 동작을 기술한다. process 문의 내부는 순차적인 동작이 기술되며, process문의 괄호 내에 존재하는 A, B는 논리회로 동작 시 내부 신호값을 변화시키는 요인이 되는 신호이다.

이와 같이 process 문에서 동작에 있어서 내부 신호값을 변화시키는 요인이 되는 신호들을 감지리스트라고 정의하며, 이는 시뮬레이션 시에 내부 신호의 변화를 판단하는 구간이 되므로 반드시 process 문에 명시되어야 한다. 위의 예에서 C는 A 또는 B의 변화에 따라 그 값이 변하므로 process 문의 감지리스트에 명시하였음을 유의한다.

이러한 감지리스트에서 필요 시그널을 누락시키고 명시하지 않은 경우에는 회로합성의 결과에는 전혀 문제를 야기시키지 않지만, 시뮬레이션 결과에서 그 변화가 관찰되지 못하여 잘못된 결과를 야기시킬 수 있다. 따라서 클록 기반의 동기회로는 process문 내에서 구현되며, 출력 및 신호의 변화가 클록 변화에 의해 야기되는 경우에는 내부 조합회로의 입력신호를 감지리스트에 명시할 필요가 없을 것이며, 단지 클록 또는 비동기 리셋이 감지리스트에 추가되면 될 것이다. 반면에 조합회로만으로 구성된

process 문에서는 모든 입력신호는 반드시 감지리스트에 명시되어야만 한다.

```
architecture 「Architecture 이름」 of 「Entity 이름」 is
「선언문」
begin
    「Architecture의 동작을 기술」
end architecture 「Architecture의 이름」;
```

: 위에서 나타낸 문법 중 「선언문」에는 동작기술시 사용되는 데이터형(Type), 내부 함수(Function), 내부 신호(Signal), 부품(Component), 상수 등이 선언되는 부분이다.

```

-- FOUR_INPUT_OR에 대한 VHDL 코드 구현
-- library 선언
library ieee;
use ieee.std_logic_1164.all;

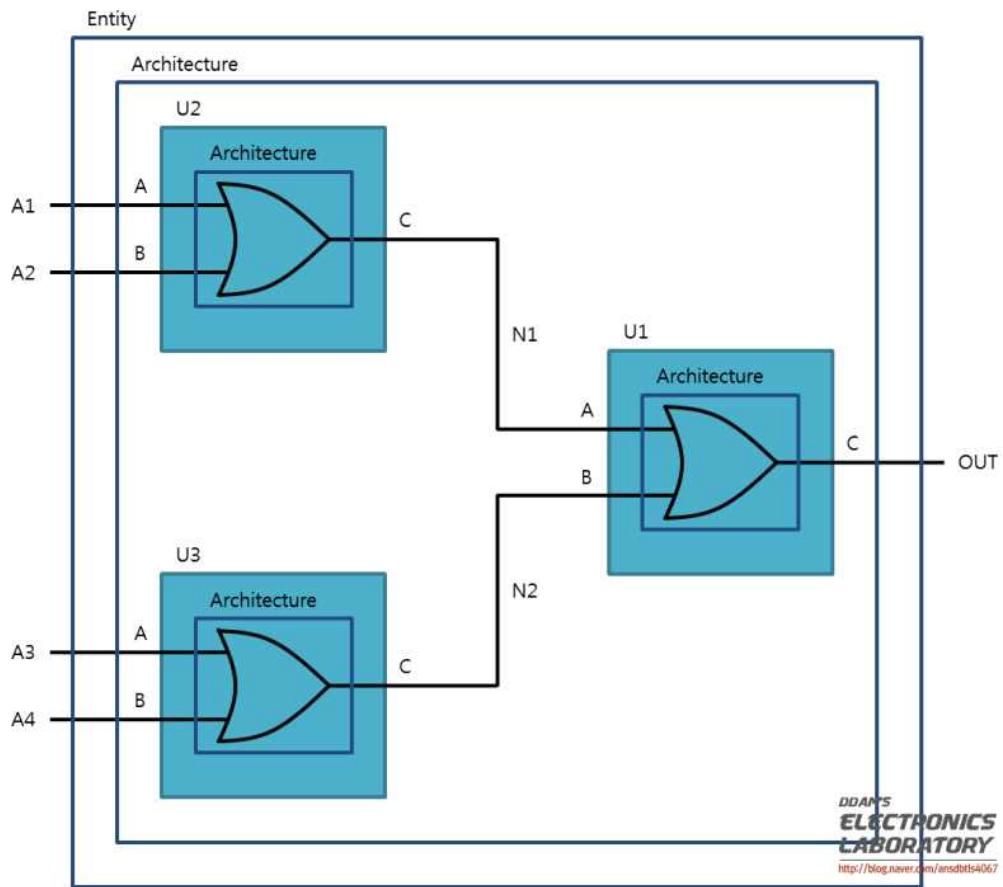
-- entity 선언
entity FOUR_INPUT_OR is
    port(A1, A2, A3, A4 : in  std_logic;
          OUT           : out std_logic);
end entity FOUR_INPUT_OR;

-- architecture 선언
architecture Behavioral of FOUR_INPUT_OR is
-- component 생성
component BLACK_BOX
    port(A, B : in  std_logic
          C   : out std_logic);
end component;

    -- 각 component를 연결하는 내부 신호선 선언
    signal N1, N2 : std_logic;
begin
    U1 : BLACK_BOX port map(A => N1, B => N2, C => OUT);
    U2 : BLACK_BOX port map(A => A1, B => A2, C => N1);
    U3 : BLACK_BOX port map(A => A3, B => A4, C => N2);

    -- 위치연결 방식을 사용한 경우
    -- U1 : BLACK_BOX port map(N1, N2, OUT);
end architecture Behavioral;

```



(5) Entity-Architecture의 Configuration

```

configuration 「Configuration 이름」 of 「Entity1 이름」 is
    for 「Architecture 이름」
        {for 「Component 라벨 이름」 : 「Component 이름」
            use entity 「Library 이름」.「Entity2 이름」(「Architecture
이름」);}
        end for;
end architecture 「Configuration 이름」;

```

-- BLACK_BOX에 대한 VHDL 코드 구현

-- library 선언

library ieee;

use ieee.std_logic_1164.all;

-- entity 선언

entity BLACK_BOX is

port(A, B : in std_logic;

C : out std_logic);

end entity BLACK_BOX;

-- architecture 선언

architecture Behavioral1 of BLACK_BOX is

begin

process(A, B)

begin C <= A or B;

end process;

end architecture Behavioral1;

architecture Behavioral2 of BLACK_BOX is

begin

C <= '0' when (A = '0' and B = '0') else
'1';

end architecture Behavioral2;

configuration Config of BLACK_BOX is

-- entity BLACK_BOX와 Behavioral2를 연결시킴

for Behavioral2

end for;

end configuration Config;

: Configuration은 하나의 Entity가 여러 Architecture를 가지고 있을 때, 특정 Architecture를 지목하여 연결해주며, 계층적 설계에서 특정 Entity와 부품 개체(Component Instances)를 연결해주는데 사용한다. 일반적으로 하나의 Entity에 하나의 Architecture를 사용하므로, 별도로 Entity-Architecture의 Configuration을 사용할 필요는 없지만, 테스트 벤치와 같은 것을 작성할 때 유용하게 사용할 수 있다. 위의 예시는 두 개의 Configuration 중 두 번째와 Entity를 연결하는 예시이다.

```
configuration 「Configuration 이름」 of 「Entity1 이름」 is
  for 「Architecture 이름」
    {for 「Component 라벨 이름」 : 「Component 이름」
      use entity 「Library 이름」.「Entity2 이름」
      (「Architecture 이름」);}
    end for;
end architecture 「Configuration 이름」;
```

: 만일 계층 설계를 사용하여 설계하였는데, 현재 레벨에서 개체로 사용한 하위레벨의 Entity가 두 개 이상의 Architecture를 가지고 있어서 하위 레벨의 Entity와 Component를 상위 레벨인 현재 레벨에서 Configuration으로 설정해야 하는 경우 {...} 내의 문을 추가하여 목적을 달성할 수 있다.

(6) Package 및 Package Body

```

package DEF_CONST_PKG is
    constant Width    : integer := 8;

    -- No MSBs 상수 정의 미루기
    constant No_MSBs : integer;
    function AND_MSBs(A : unsigned(Width - 1 downto 0))
return std_logic;
end package DEF_CONST_PKG;

package body DEF_CONST_PKG is
    -- 실제 No_MSBs 상수 정의
    constant No_MSBs : integer := 3;

    function AND_MSBs(A : unsigned(Width - 1 downto 0))
return std_logic is
    variable V : std_logic;

    begin
        V := '1';

        for N in 7 downto 8 - No_MSBs loop
            V := V and A(N);
        end loop;

        return V;
    end function AND_MSBs;
end package body DEF_CONST_PKG;

```

: Package는 일반적으로 데이터형(Type)과 서브프로그램(Subprogram)을 정의하여 놓은 것으로, 데이터형(Type), 함수(Function), 프로시저(Procedure) 등을 여러 설계에서 중복하여 정의하지 않고 공통으로 사용할 수 있도록 만들어 놓은 코드이다. Package Body에는 Package 선언에서 선언한 서브프로그램인 함수, 프로시저의 몸체부분 다시 말해 동작부분을 정의하는 방식으로 사용할 수 있다.

(7) Function 및 Procedure

```
package USER_FUNCTION is
    function Two_Input_Or(X, Y : bit) return bit;
end package USER_FUNCTION;

package body USER_FUNCTION is
    function Two_Input_Or(X, Y : bit) return bit is
        variable OUT : bit;
    begin
        OUT := X or Y;
        return OUT;
    end function Two_Input_Or;
end package body USER_FUNCTION;

package USER_PROCEDURE is
    procedure Two_Input_Or(signal X, Y : in bit;
                           signal OUT : out bit);
end package USER_PROCEDURE;

package body USER_PROCEDURE is
    procedure Two_Input_Or(signal X, Y : in bit;
                           signal OUT : out bit); is
    begin
        OUT <= X or Y;
    end procedure Two_Input_Or;
end package body USER_PROCEDURE;
```

: Function과 Procedure는 주로 Package에 포함되어 사용되는 서브프로그램이다.

(8) 변수(Variable)와 신호(Signal)

```
-- 변수 선언
variable 「변수 이름」 : 「데이터형」 := 「초기값」
-- 변수 할당
「변수 이름」 := 「할당값」

-- 신호 선언
signal 「신호 이름」 : 「데이터형」 := 「초기값」
-- 신호 할당
「신호 이름」 <= 「할당값」
```

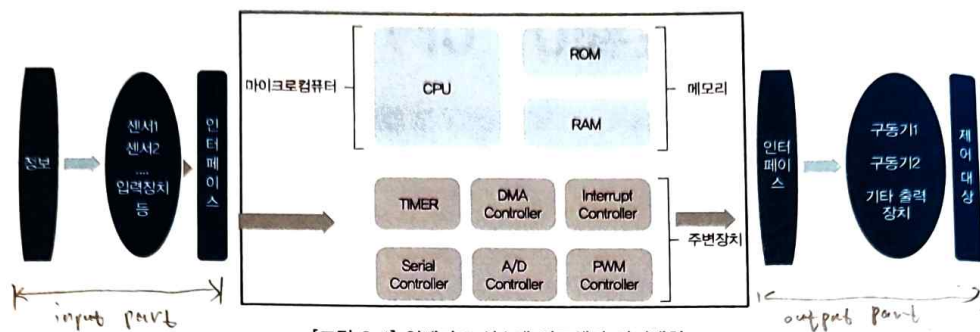
: 변수는 프로세스 또는 함수, 프로시저와 같은 서브프로그램 내에서만 선언가능한 값의 기억장소이며, 값의 할당 시에는 할당과 동시에 즉시 값이 갱신된다. 반면에 신호는 Entity, Architecture, Package, Block에서 선언되며, Port와 같이 값의 할당 시에는 일정 지연 후에 값이 갱신된다.

< 컴퓨터 구조와 마이크로프로세서 >

1. CPU(중앙처리장치) 구조

1) CPU/마이크로프로세서의 구조

▶ 마이크로 프로세서의 구조

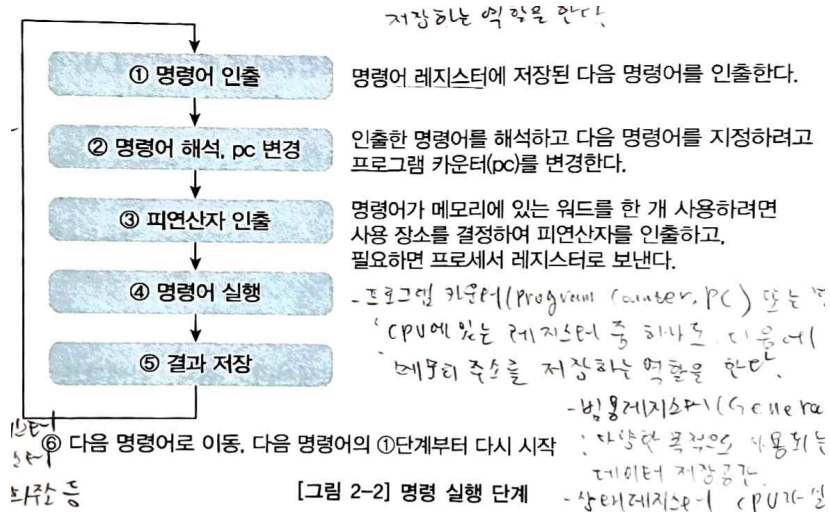


[그림 2-1] 임베디드 시스템 하드웨어 아키텍처

각 구성 요소

- CPU : 마이크로프로세서로서 연산과 제어의 기능을 수행한다.
- 메모리 : 수행중인 프로그램, 데이터, 화면 데이터들이 저장되어 마이크로 프로세서와의 정보를 주고 받는 기억장치이다.
- 보조 기억장치 : 메모리의 보완으로써, 수행중인 임시 자료의 저장이나, 비휘성의 프로그램 데이터 등을 저장하는 대용량의 기억장치이다.
- 입출력 장치 : 외부의 데이터를 받거나, 내부의 데이터를 외부로 저장하거나 출력 입력하는 장치이다.
- 입력부(Input part) : MPU가 연산 처리하기 위한 정보를 마이크로컴퓨터로 입력하는 부분으로 입력 포트라고도 한다. 기본 입력 부분은 MPU와의 타이밍 조정과 포트의 교체 등에 관한 기능을 하고 있으며, 센서는 물리적인 변화를 전기적인 변화로 출력해준다.
- 출력부(Output part) : MPU가 연산 처리한 결과를 사용하여 구체적으로 물리적인 동작을 시키기 위해 마이크로컴퓨터부에서 정보를 출력하는 것으로, 출력 포트라고도 한다. MPU와의 타이밍 조정과 포트의 교체 등에 관한 기능이 있으며 액추에이터는 전기량의 변화를 기계적인 동작으로 변환하는 부분이다.

▶ CPU 명령어 처리 단계



: MPU는 명령을 메모리에서 읽어내는 인출(fetch), 명령을 해독하는 해독(decode), 명령을 실행하는 실행(execute)라는 작업을 반복하여 프로그램을 처리한다.

- (1) 명령어 인출(fetch) : 명령어 레지스터에 저장된 다음 명령어를 인출한다.
- (2) 명령어 해석, PC변경 : 인출한 명령어를 해석하고 다음 명령어를 지정하려고 프로그램 카운터(PC)를 변경한다.
- (3) 피연산자 인출 : 명령어가 메모리에 있는 워드를 한 개 사용하려면 사용 장소를 결정하여 피연산자를 인출하고, 필요하면 프로세서 레지스터로 보낸다.
- (4) 명령어 실행
- (5) 결과 저장
- (6) 다음 명령어로 이동, 다음 명령어의 (1)단계부터 다시 시작

▶ MPU 내부 구성 방식에 따른 CISC와 RISC

- CISC(Complex Instruction Set Computer)

: 연산을 처리하는 복잡한 명령어들을 수백 개 이상 탑재하고 있는 프로세서이다. CISC의 명령어 개수가 증가함에 따라 프로세서 내부 구조가 매우 복잡해지고, 고속으로 작동되는 프로세서를 만들기 힘들어진다.

- CISC의 특징

- (1) 명령어의 개수가 많다.
- (2) 명령어 길이가 다양하며, 실행 사이클도 명령어마다 다르다.
- (3) 회로 구성이 복잡하다.
- (4) 프로그램을 만들 때 적은 명령어로 구현 가능하다.
- (5) 다양한 명령어를 사용하기 때문에 컴파일러가 복잡하다.

- RISC(Reduced Instruction Set Computer)

: 핵심적인 명령어를 기반으로 최소한의 명령어 세트를 구성함으로써 파이프라이닝이라는 획기적인 기술을 도입할 수 있어 빠른 동작 속도와 하드웨어의 단순화 및 효율화가 가능했고, 가격 경쟁력에서도 우위를 점했다.

- RISC의 특징

- (1) 명령어 세트가 적다.
- (2) 명령어가 간단하고 실행속도가 빠르다.
- (3) 명령어 길이가 고정적이다.
- (4) 워드, 데이터 버스 크기가 같고 실행 사이클도 모두 동일하다.
- (5) 회로 구성이 단순하다.
- (6) 프로그램을 구성할 때 상대적으로 많은 명령어가 필요하다.
- (7) 파이프라이닝을 사용한다.
- (8) 명령어 개수가 적어서 컴파일러가 단순하게 구현된다.

비교	CISC	RISC
하드웨어/소프트웨어 상호보완	하드웨어 및 펌웨어 중시	하드웨어 간소화, 소프트웨어 중시
명령 종류	다수	가능한 한 소수
어드레스 수식방식	다수	소수
명령 형식	가변, 복수	고정, 단일
명령 길이	가변, 복수	고정, 단일
명령 실행 사이클 수	2~15	1(목표)
주 기억 액세스	다수의 명령 (메모리-메모리, 레지스터-메모리)	load, store, 레지스터-레지스터
제어논리의 구성방식	마이크로프로그램 제어	하드웨어 논리(결선 논리)제어
범용 레지스터의 수	적음(8~24)	많음(32~192)

2) 버스 시스템

- ▶ 버스(Bus) : 버스는 장치 사이에 정보를 교환하기 위한 통로이며, 사용되는 장소에 따라 내부 버스와 외부 버스로 나뉜다.
- ▶ 내부 버스 : 마이크로컴퓨터를 구성하고 있는 MPU, 메모리 기본, 기본I/O, 사이에서 정보를 교환하기 위한 공통된 신호선의 묶음이다.
- ▶ 외부 버스 : 마이크로컴퓨터부와 주변기기 등과의 사이에서 정보교환을 위한 신호선 묶음으로 주변기기의 종류와 목적에 따라 구분하여 사용한다.
- ▶ 내부 버스를 구성하는 신호선 묶음은 기본적으로 주소 버스, 데이터 버스, 제어 버스가 있다. 주소 버스에 의해 정보를 교환하는 상대를 지정하고 데이터

버스를 사용하여 그 상대와 정보를 교환하며, 제어 신호는 제어 버스를 사용하여 실행한다.

▶ 주소 버스의 버스 폭에 의해 그 버스에서 지정할 수 있는 주소공간의 크기가 결정되며, ROM에 고정하는 처리 프로그램의 크기와 RAM을 사용하여 처리하는 스택과 변수영역, 데이터 버퍼들의 크기를 다루는 주소 공간이 필요하게 된다.

▶ 데이터 버스는 버스 폭이 넓을수록 한 번에 전송할 수 있는 비트 수가 증가하기 때문에 전송횟수를 줄여 전송속도를 향상시킬 수 있다. 또한 IC칩의 단자 수 제약에 MPU 내부 처리 단위로서의 비트 수와 데이터 버스의 비트 수가 다른 것도 있다. 따라서 데이터 버스의 비트 폭은 MPU의 처리성능을 좌우하는 요소이다.

3) 명령어(Instruction) 집합 구조

▶ 명령어 : CPU가 수행할 동작을 2진수 코드로 정의한 것이다. 일반적으로 연상부호를 사용한 어셈블리형태로 표현한다.

▶ 명령어 집합 구조(Instruction Set Architecture, ISA) : 컴퓨터 아키텍처의 한 부분으로, 하드웨어와 소프트웨어 사이의 인터페이스를 정의한다. 즉, 프로그래머가 사용할 수 있는 기계어 명령들을 정의하며, 이는 CPU가 어떤 종류의 연산을 수행할 수 있는지 결정한다.

▶ ISA의 요소

(1) 명령어와 데이터 형식: ISA는 사용 가능한 명령어들과 그 형식을 정의하며, 각 명령어가 어떤 연산을 수행하는지를 설명한다. 또한 데이터 타입(예: 정수, 실수 등)과 이들이 메모리에서 어떻게 표현되고 처리되는지도 정의한다.

(2) 레지스터 세트: 레지스터는 CPU 내부에 있는 작은 저장 공간으로서, 매우 빠른 접근 속도를 가진다. ISA는 사용 가능한 레지스터들과 그 용도를 정의한다.

(3) 주소 지정 모드: 주소 지정 모드는 메모리에서 데이터를 가져오거나 저장하는 방법을 결정하는 규칙이다.

(4) 제어 흐름 명령: 제어 흐름 명령은 프로그램 실행 순서를 변경하는 연산이다. 예로 들면 분기(branch), 점프(jump), 함수 호출(call), 반환(return) 등이 있다.

(5) 입출력(I/O) 관리 방법: 일부 ISA에서는 입출력 장치와 상호작용하는 방법에 대해 직접적인 지원을 할 수 있다.

▶ 일반적으로 ISA는 RISC와 CISC라는 두 가지 범주에서 컴퓨터 시스템 설계와 프로그래밍에 있어서 요구사항에 따른 적절한 선택을 해야한다.

4) 어드레싱 모드

▶ 어드레싱 모드(Addressing Mode) : 어드레싱 모드(Addressing Mode)는 컴퓨터 아키텍처에서 명령어가 피연산자를 참조하거나 접근하는 방식을 결정하는 메커니즘이다. 다시 말해, 어드레싱 모드는 CPU가 데이터를 메모리에서 어떻게 찾아올 것인지를 정의한다.

▶ 어드레싱 모드의 종류

(1) 직접 어드레싱(Direct Addressing): 이 모드에서는 명령어 자체에 데이터의 메모리 주소가 직접 포함된다. CPU는 이 주소로 직접 접근하여 데이터를 읽거나 쓴다.

(2) 간접 어드레싱(Indirect Addressing): 이 모드에서는 명령어에 포함된 주소가 실제 데이터의 위치를 가리키는 포인터로 사용된다. CPU는 먼저 이 포인터를 사용하여 메모리에서 실제 데이터의 주소를 가져온 후, 그 주소로 접근하여 데이터를 읽거나 쓴다.

(3) 즉시 어드레싱(Immediate Addressing): 이 모드에서는 명령어 자체에 피연산자(데이터)가 직접 포함된다.

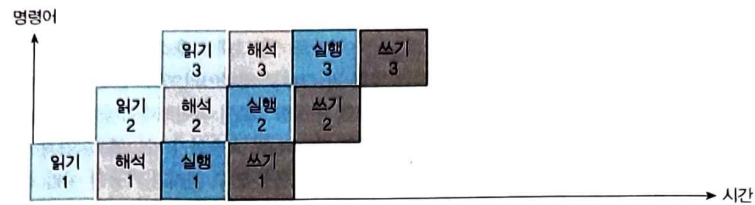
(4) 베이스 레지스터(Base Register) 또는 기준치 상대(Based Relative) 어드레싱: 베이스 레지스터와 함께 오프셋 값을 사용하여 실제 메모리 위치를 계산한다.

(5) 상대(Relative) 어드레싱: 프로그램 카운터와 함께 오프셋 값을 사용하여 실제 메모리 위치를 계산한다.

(6) 색인(Indexed) 어드레싱: 인덱스 레지스터와 함께 오프셋 값을 사용하여 실제 메모리 위치를 계산한다.

5) 마이크로 아키텍처

▶ 파이프라인(Pipelining) : 파이프라인은 컴퓨터 아키텍처에서 명령어 처리를 여러 단계로 나누어 각 단계를 동시에 실행할 수 있도록 하는 기법이다. 이는 제조 공정에서의 조립 라인과 유사하게 작동한다.



(a) 파이프라인이 동작하는 경우



(b) 파이프라인이 동작하지 않는 경우

[그림 2-3] 파이프라인 방식

▶ 슈퍼스칼라(Superscalar) : 슈퍼스칼라는 컴퓨터 CPU 디자인의 한 종류로, 하나의 클럭 사이클 내에서 여러 개의 명령을 동시에 실행할 수 있는 기능을 가진 것을 말한다. 이를 위해 CPU는 여러 개의 ALU(Arithmetic Logic Unit) 또는 다른 함수 유닛을 포함하며, 복잡한 디코딩 로직을 사용하여 가능한 많은 명령들이 동시에 실행될 수 있도록 한다.

▶ 분기(Branch) : 컴퓨터 프로그래밍에서 프로그램의 실행 흐름을 변경하는 것을 말한다. 조건부 분기(Conditional Branch)와 무조건부 분기(Unconditional Branch)로 나뉘며 조건부는 if, switch, loop, for, while 등의 제어구조를 사용하여 특정 조건이 참인 경우에만 수행되며, 무조건부는 주로 goto문을 이용하여 조건 확인 없이 실행 흐름을 변경한다. 이때 분기 명령어는 CPU의 명령어 세트에 포함되며, 프로그램 카운터를 변경하여 다음에 실행할 명령어의 위치를 변경한다.

▶ 분기 예측(Branch Prediction) : 프로그램의 제어 흐름을 미리 예측하여 분기 후에 실행될 명령어들을 미리 가져오고 준비하는 방식이다. 이를 통해 프로세서는 파이프라인을 효과적으로 유지하고, 분기 지연을 최소화하여 전체적인 성능 향상을 도모한다.

→ 작동 원리 : 프로그램의 실행 중에 조건부 분기가 발생하면, 해당 분기가 참인지 거짓인지 알 수 없는 경우가 많다. 이러한 상황에서, 프로세서는 일반적으로 분기 결과를 알게 될 때까지 다음 명령어를 가져올 수 없으며, 이로 인해 파이프라인이 비워져 성능 저하가 발생할 수 있다. 분기 예측 기법은 이러한 문제를 해결하기 위해 개발되었으며 프로세서가 미래의 분기 결과를 예측하고 그에 따라 실행할 명령어들을 사전에 준비하도록 한다.

- (1) 정적 분기 예측(Static Branch Prediction) : 조건부 분기마다 고정된 규칙에 따라 예측하며, 모든 분기가 taken or not taken 즉 점프하거나 점프하지 않는 것으로 가정한다.
- (2) 동적 분기 예측(Dynamic Branch Prediction) : 실제 실행 결과 및 패턴 등을 추적하여 동작한다. 주요한 동적 방식 중 하나인 "양자화(Two-bit Predictor)"는 2비트 상태 정보를 사용하여 각각의 조건부 분기마다 가능성 여부를 추정한다.
- (3) 추론(Branch Target Prediction) : 무조건부 점프에서 다음 목표 주소(target address)를 추론하여 사전에 적재한다.
- (4) 동적 인출(Speculative Fetch) : 조건부 점프 직전까지 계속해서 다음 명령어들을 가져오고 준비한다.

6) ARM CPU

- ▶ ARM CPU : ARM Holdings에서 개발한 프로세서 아키텍처이다. 이는 저전력 소비에 초점을 맞추고 있으며, 주로 스마트폰, 태블릿, 웨어러블 기기 등의 모바일 디바이스에 널리 사용된다.
- ▶ ARM 아키텍처의 특징
 - (1) RISC 기반: ARM은 Reduced Instruction Set Computing(RISC) 아키텍처를 따른다. 이는 간단하고 고정 길이의 명령어 집합을 사용하며, 각 명령어가 하나의 CPU 사이클 내에서 실행될 수 있도록 설계되어 있다. 이로 인해 하드웨어가 단순화되고 성능 향상 및 전력 소모 감소에 도움이 된다.
 - (2) 저전력 소비: ARM 프로세서는 저전력 소비를 중요한 설계 목표로 삼아왔다. 이는 배터리를 사용하는 모바일 기기에서 매우 중요한 요소이다.
 - (3) 라이선스 모델: ARM Holdings는 실제 칩을 제조하지 않는다. 대신 회사는 ARM 아키텍처에 대한 라이선스를 다른 회사들에게 판매하며, 그런 후 그 회사들이 자체적으로 칩을 제조한다.
 - (4) 버전과 프로파일: ARM 아키텍처는 여러 버전(예: ARMv7, ARMv8 등)으로 나뉜다. 각 버전은 일련의 기능과 명령 집합을 정의한다. 덧붙여서 각 버전 내에서도 '프로파일'별로 세분화되며, 이들은 특정 시장 세그먼트(예: 실시간 시스템용 'R 프로파일', 애플리케이션용 'A 프로파일' 등)를 위해 설계된다.
 - (5) 64비트 지원: 초기의 ARM 프로세서들은 32비트였지만 최근 버전인

ARMv8부터는 64비트 연산도 지원한다.

- (6) 코어 디자인(고집적도): ARM은 다양한 코어 디자인을 제공한다. 예를 들어, Cortex-A 시리즈는 고성능 모바일 애플리케이션에 사용되며, Cortex-M 시리즈는 마이크로컨트롤러 등의 저전력 임베디드 시스템에 사용된다.

▶ ARM CPU의 주요 명령어

(1) 데이터 처리(Data Processing) 명령어: 이 범주의 명령어들은 레지스터에 저장된 데이터를 처리한다.

- ADD (덧셈), SUB (뺄셈), MUL (곱셈), DIV (나눗셈), AND, ORR, EOR 등의 논리 연산, MOV (이동), MVN (부정 이동) 등

(2) 메모리 액세스(Memory Access) 명령어: 이 범주의 명령어들은 메모리에서 데이터를 읽거나 쓰는 작업을 수행한다.

- LDR(Load Register): 메모리에서 레지스터로 데이터 로드

- STR(Store Register): 레지스터에서 메모리로 데이터 저장

(3) 제어 흐름(Control Flow) 명령어: 이 범주의 명령어들은 프로그램 실행 흐름을 제어한다.

- B(Branch): 분기

- BL(Branch with Link): 서브루틴 호출 및 링크 레지스터에 반환 주소 저장

- BX(Branch and Exchange): 주소로 분기하며 실행 상태 변경(ARM 상태와 Thumb 상태 사이 전환)

(4) 비교(Comparison) 및 분기(Branching) 명령어: 비교 결과에 따라 프로그램 흐름을 제어하는데 사용된다.

- CMP(Compare): 값을 비교하고 상태 레지스터 설정

- BEQ(Branch if Equal), BNE(Branch if Not Equal), BGT(Branch if Greater Than), BLT(Branch if Less Than) 등: 조건부 분기

(5) 시스템(System) 명령어: 운영 체제와 관련된 작업을 수행하는데 사용된다.

- SVC(Supervisor Call): 운영 체제 서비스 요청

▶ ARM의 명령어 세트 기능

(1) ARM 명령어 세트: 이것이 ARM 프로세서의 원래 명령어 세트입니다. 이 명령어들은 대부분 32비트 길이를 가지며, 고정된 길이를 가진다.

(2) Thumb 명령어 세트: Thumb는 ARMv4T에서 처음 도입된 16비트 명령어 세트이다. Thumb는 코드 크기와 전력 소모를 줄여주므로 저전력 또는 메모리 제한 환경에서 유용하다.

(3) Thumb-2 확장: Thumb-2는 ARMv6T2에서 도입되었으며 16비트 및

32비트 Thumb명령을 혼합하여 사용할 수 있게 한다. 이 확장을 통해 개선된 코드 밀도와 실행 효율성을 얻을 수 있다.

(4) Jazelle 확장: Jazelle DBX(Direct Bytecode eXecution) 확장은 Java 바이트코드를 직접 실행할 수 있는 하드웨어 지원을 추가한다.

(5) NEON SIMD(Single Instruction Multiple Data)확장: NEON확장은 고속 멀티미디어 및 신호 처리 애플리케이션에 유용한 벡터 연산 지원을 추가한다.

(6) VFP(Vector Floating Point)확장: VFP확장은 부동소수점 연산에 대한 하드웨어 지원을 추가한다.

(7) AArch64/ARMv8-A ISA(Instruction Set Architecture): AArch64는 ARMv8-A 아키텍처에서 도입된 새로운 64비트 명령어 세트이다. 이는 기존 ARM과 Thumb 명령어 세트를 확장하고, 64비트 레지스터와 메모리 주소 지원을 추가한다.

2. 메모리 시스템

1) 메모리 계층구조



: 메모리 계층구조는 메모리 관련 3가지 주요 특성인 용량, 접근 속도, 비용간의 절충 관계를 파악해 필요에 따라 채택할 수 있게 나타낸 구조이다. 위로 갈수록 속도가 빠르고 비싸며, 아래로 갈수록 용량이 커진다. 하드디스크는 보조기억장치를 의미한다.

- 레지스터 : CPU 내부의 작은 메모리로, 휘발성이며 속도가 가장 빠르고, 기억 용량이 가장 적다.
- 캐시 : L1,L2,L3 캐시를 지칭하고 휘발성이며, 속도가 빠르나 기억 용량이 적다.
- 주기억장치(메모리) : RAM을 말하며 휘발성이고, 속도와 기억 용량은

보통이다.

- 보조기억장치(디스크) : HDD, SSD를 일컬으며 비휘발성이다. 속도는 낮으며 기억 용량이 많다.

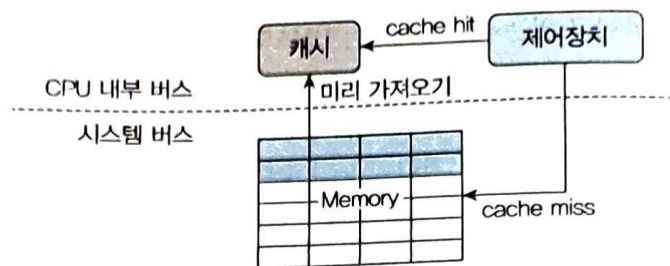
▶ 메모리 계층구조의 필요성

: 메모리 계층구조는 속도와 용량의 상호보완을 위해 고안한 방법이며, 프로세서가 필요로 하는 데이터를 최대한 가까운 곳에 위치시켜 속도를 향상시키는 것을 목표로 한다. 또한 경제성을 고려하기도 하였다.

2) 캐시 메모리

▶ 캐시(cache) : 메모리와 CPU간의 속도 차이를 완화하기 위해 메모리의 데이터를 미리 가져와(prefetch) 저장해두는 임시 장소이다. 캐시는 CPU 안에 있으며, CPU 내부 버스의 속도로 작동하고, 메모리는 시스템 버스의 속도로 작동하기 때문에 캐시가 메모리보다 빠르다.

▶ 캐시의 구조



[그림 2-10] 캐시 메모리의 동작

: CPU는 메모리에 접근하기 전 캐시를 먼저 방문하여 원하는 데이터가 있는지 찾아본다. 캐시에서 원하는 데이터를 찾으면 캐시 히트(cache hit)라고 하며, 그 데이터를 캐시에서 가져와서 바로 사용한다. 그러나 원하는 데이터가 캐시에 없으면 메모리로 가서 데이터를 찾는데, 이를 캐시 미스(cache miss)라고 한다. 캐시 히트가 되는 비율을 캐시 적중률(cache hit ratio)라고 한다.

▶ 캐시 적중률 계산 방법

: (캐시 적중률) = (캐시 적중 수) / (캐시 접근 수)

▶ 캐시의 적중률 증가 방법

- (1) 캐시의 용량 증가 : 캐시의 용량을 키우면 더 많은 데이터가 들어와서 캐시 히트 확률이 높아진다.

- (2) 앞으로 많이 사용될 데이터 가져오기 : 현재 위치에 가까운 데이터가 멀리 있는 데이터보다 사용될 확률이 더 높다는 지역성 이론에 따라 데이터를 미리 가져온다.

▶ 메모리 기록 방식

- (1) 직접 기록 : 한 워드가 메인 메모리와 캐시 메모리 양쪽에 모두 기록되는 방식으로 보통 양쪽의 내용이 일치한다. 기록하는 속도는 느리지만 고장에 대한 데이터 기록 오류가 없으며, 캐시 미스 때문에 메모리를 치환할 때 데이터 복구가 필요없다. 넓은 어드레스 공간을 여기저기 액세스 하는 경우 유리하다.
- (2) 기록 복귀 : 캐시 메모리가 변하더라도 메모리는 갱신되지 않고 대신 블록이 캐시 메모리로부터 제거될 때, 메모리를 갱신하는 방법이다. 메인 메모리의 트래픽 측면에서 직접 기록 방식보다 우수한 성능을 가지고 캐시 메모리에만 동작을 하므로 고속으로 행할 수 있다. 하지만 논리적인 복잡성이 따르고, 캐시 메모리와 메인 메모리의 내용이 항상 일치하지 않아 주의가 필요하다. 좁은 어드레스 공간을 빈번하게 액세스 하는 경우에 유리하다.
- (3) 기록 사이클의 캐시 무효화 : 워드는 직접 기록시키지만 캐시는 갱신시키지 않는 방법이다. 여러 프로세서가 동일한 메모리 위치에 대한 복사본을 갖고 있는 상황에서, 한 프로세서가 해당 위치를 수정하면 다른 모든 캐시의 복사본이 유효하지 않게된다. 이러한 경우 해당 메모리에 해당하는 모든 캐시 항목을 무효화 한다. 따라서 프로세서는 워드를 메인 메모리로부터 호출하는 것이 된다. 그 캐시 무효화를 플러싱(flushinh) 또는 퍼지(purge)라고 한다.
- (4) 기록 버퍼 : 직접 기록 방식의 변형으로 메인 메모리가 기록 버퍼를 통해서 기록되는 방법이다. 버퍼에 데이터를 기록한 후 메인 메모리에 저장하면, 디스크와 같은 슬로우 I/O장치와 같이 액세스가 오래걸리는 경우에 유리하다.

▶ 메모리 사상 방식

: 메인 메모리에서 데이터 참조는 주소로 이루어지고, 캐시 메모리를 사용하기 위해서는 메인 메모리의 주소를 캐시 메모리의 워드에 사상시켜야 한다.

- (1) 직접 사상 방식(direct mapping) : 가장 간단한 방법으로 메인 메모리의 블록을 캐시 메모리의 블록 프레임에 직접 사상시키는 방법이다. CPU가 메모리의 참조를 요청하면, 인덱스 필드(태그 필드 or 오프셋 필드 :

데이터 베이스의 특정 테이블의 column 데이터)가 캐시 메모리를 호출하기 위한 번지로 변환할 때 쓰인다. CPU 번지의 태그 필드와 캐시로부터 읽힌 워드의 태그 필드를 비교해서 원하는 데이터가 캐시에 있음을 나타내는 것이다. 즉, 메인 메모리를 블록으로 분할하여 각각의 블록과 캐시 메모리와의 대응 관계가 고정되어 있다. 이 방식은 데이터와 태그에 대한 동시 호출이 허용되고 하드웨어가 간단하다는 장점이 있지만, 매핑의 자유도가 낮아 캐시 메모리 접근에 대한 실패율이 높다.

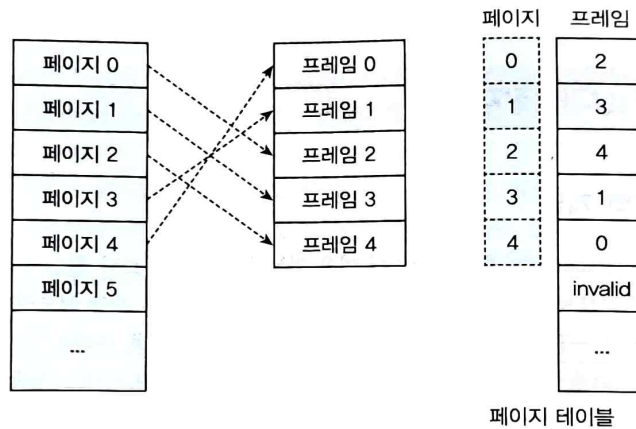
- (2) 완전 연관(연상) 사상 방식(fully associative mapping) : 메인 메모리를 블록으로 분할할 때, 모든 캐시 메모리로부터 어느 블록이나 액세스할 수 있게 되어 있다. 따라서 매핑의 자유도는 높아지지만 시스템의 가격이 올라간다. 메모리 워드의 번지와 데이터를 캐시 메모리의 블록 프레임 내에도 동시에 저장하는 방법이며, 높은 블록 경쟁을 제거할 수 있으나 호출 시간이 길어진다.
- (3) 세트 연관(집합-연상) 사상 방식(set-associative mapping) : 직접 사상 방식과 완전 연관 사상 방식을 결합한 방법으로 캐시 메모리의 각 워드를 같은 인덱스 번지 아래서 두 개 이상의 메모리 워드를 저장할 수 있도록 함으로써 직접 사상의 단점을 보완한 방법이다.

3) MMU와 가상메모리 시스템, 페이징

- ▶ MMU (Memory Management Unit, 메모리 관리 유닛): MMU는 컴퓨터의 하드웨어 구성 요소로, 가상 메모리를 물리적 메모리에 매핑하는 역할을 한다. 즉, 프로세스에서 생성되는 가상 주소를 실제 하드웨어 메모리 주소, 즉 물리 주소로 변환하는 기능을 수행한다. 이를 통해 프로세스가 독립적인 공간에서 실행되도록 보장하며, 또한 각 프로세스가 서로 격리되어 메모리 보호가 가능해진다.
- ▶ 가상 메모리(virtual memory) : 주기억장치의 이용 가능한 기억 공간보다 훨씬 큰 주소를 지정할 수 있도록 한 개념이다. 가상 메모리를 사용하면 사용자는 실제 주소 공간의 크기에 구애받지 않고 보다 큰 가상 주소 공간에서 프로그래밍을 할 수 있을 뿐만 아니라, 주기억장치보다 크기가 큰 프로세스를 수행시킬 수 있다.
- ▶ 가상 메모리의 메모리 분할 방식 : 실제 메모리에 있는 물리 주소 0번지는 운영체제 영역으로 일반 프로세스가 사용할 수 없다. 따라서 가상 메모리 시스템에서는 운영체제를 제외한 나머지 메모리 영역을 일정한 크기로 나눠 일반 프로세스에 할당한다. 메모리 분할 방식은 크게 가변 분할

방식(세그멘테이션)과 고정 분할 방식(페이징)으로 나뉜다.

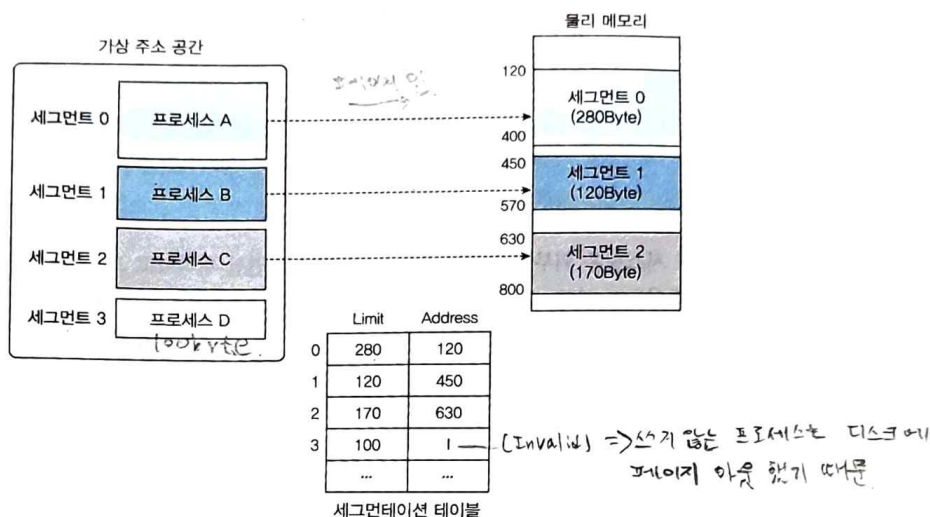
▶ 페이징 기법의 구현



[그림 2-11] 페이징 기법의 구현

: 페이징 기법은 고정 분할 방식을 이용한 가상 메모리 관리 기법으로, 물리 주소 공간을 같은 크기로 나눠 사용한다. 가상 주소의 분할된 각 영역은 페이지라고 부르며 번호를 지정해 관리한다. 물리 메모리의 각 영역은 가상 주소의 페이지와 구분하기 위해 프레임(frame)이라고 부른다. 페이지와 프레임의 크기는 같으며, 물리적 메모리에서 디스크로 페이지를 보내는 것을 페이지 아웃(스왑), 그 반대를 페이지 인이라고 한다.

▶ 세그멘테이션 기법의 구현



[그림 2-12] 세그멘테이션 기법

: 가변 분할 방식을 사용하며, 페이징 기법과 마찬가지로 세그멘테이션 기법도

매핑 테이블을 사용한다. 이를 세그멘테이션 테이블이라고 한다. 세그멘테이션 테이블에는 세그멘테이션 크기를 나타내는 limit와 물리 메모리의 시작 주소를 나타내는 address가 있다.

▶ 페이지 부재(Page Fault)

: 가상 메모리 시스템에서 발생하는 현상으로, 프로세스가 접근하려는 페이지가 메인 메모리에 없을 때 발생한다. 이는 페이지가 아직 메모리에 로드되지 않았거나, 이전에 사용되었으나 메모리 부족 등의 이유로 제거되었을 때 일어난다.

▶ 페이지 부재 시 운영체제의 처리 방법

- (1) 운영체제는 먼저 페이지 테이블을 확인하여 요청된 페이지가 메모리에 있는지 확인한다.
- (2) 페이지가 메모리에 없다면, 운영체제는 해당 페이지를 디스크에서 찾아 메모리에 로드한다. 이 과정에서 필요하다면, 다른 페이지를 디스크에 다시 저장(스왑 아웃)하여 메모리 공간을 확보할 수 있다.
- (3) 페이지를 메모리에 로드한 후, 운영체제는 페이지 테이블을 업데이트하여 해당 페이지의 새로운 메모리 위치를 반영한다.
- (4) 마지막으로, 운영체제는 원래의 메모리 접근 연산을 다시 시작하여 프로세스가 계속 실행될 수 있도록 한다.

▶ 내부 단편화

: 내부 단편화는 할당된 메모리 영역 내부에 발생하는 낭비된 공간을 의미한다. 예를 들어, 메모리가 1000byte의 블록으로 나누어져 있고, 프로세스가 1000byte 만큼의 메모리를 모두 사용하지 않는 경우, 해당 블록 내에 남은 공간(1000byte)이 내부 단편화를 발생시킨다. 이는 고정된 크기의 블록을 사용하는 메모리 할당 방식에서 흔히 발생한다.

▶ 외부 단편화

: 외부 단편화는 메모리의 할당과 해제 과정에서 발생하는, 사용할 수 없는 작은 메모리 조각을 의미한다. 메모리가 동적으로 할당되고 해제되는 과정에서, 연속된 메모리 공간이 여러 조각으로 나뉘어져서 하나의 큰 메모리 블록을 할당할 수 없게 되는 현상이 외부 단편화이다. 이는 가변 파티션 메모리 할당 방식에서 주로 발생한다. 예를 들어, 메모리 동적 할당을 위해 30byte, 40byte, 100byte의 세 개의 블록이 생성되고, 메모리를 모두 사용 후에 이 블록들은 계속 남게되어 더 큰 메모리 공간들이 필요할 때 블록들이 사용되지 못한다. 이 때 외부 단편화를 발생시킨다.

4) Thrashing(스레싱)과 Working Set(워킹셋)

▶ Thrashing(스레싱)

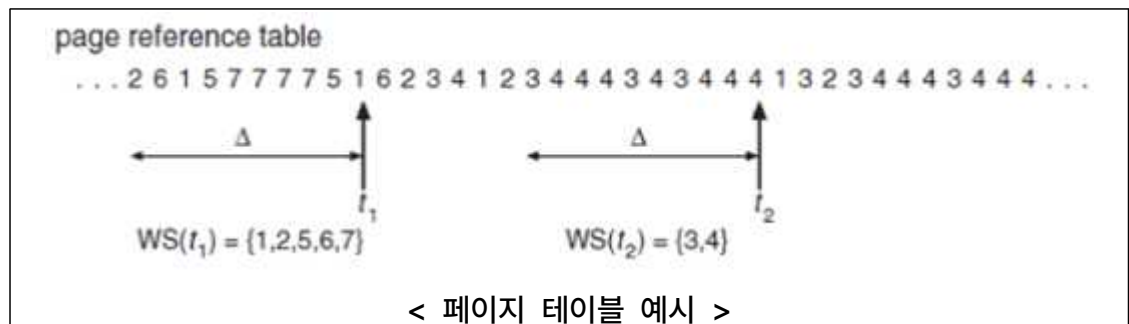
: 스레싱은 페이지 부재가 빈번하게 발생하여 실제 CPU 사용 시간보다 페이지 교체하는 시간이 많아지는 현상을 뜻한다. 주기억장치의 모든 페이지가 활발히 사용되고 있어서 어떤 페이지가 교체된 후 곧바로 반복적인 페이지 부재가 발생하면서 프로세스의 실행보다 페이지징을 위해 더 많은 시간을 소요하는 경우이다.

- 스레싱 방지 방법: 스레싱을 방지하는 방법은 프로세스의 메모리 요구량을 적절하게 예측하고, 충분한 메모리를 할당하는 것이다. 또한, 시스템의 메모리 사용량을 모니터링하여 메모리가 과도하게 사용되는 것을 미리 감지하고, 필요한 조치를 취하는 것도 중요하다.

▶ Working Set(워킹셋)

: 프로세스가 실행되는 동안 실제로 사용하는 페이지들의 집합을 의미한다. 이는 프로세스의 메모리 접근 패턴을 나타내며, 자주 사용되는 페이지들이 모여 있고, 가상 메모리 시스템에서 페이지 교체 알고리즘의 일부로 사용된다.

- 워킹셋 모델: 지역성을 기반으로 가장 많이 쓰이는 페이지 집합을 메모리 공간에 계속 상주 시켜 스레싱을 줄이는 방법
- 워킹셋 윈도우 : 주로 델타로 표시하며 고정된 숫자
- 워킹셋 사이즈 : 각 프로세스 별로 가장 많이 쓰이고 있는 페이지의 갯수



: 위의 예시에서 워킹셋 윈도우가 10으로 고정되어 있고 페이지를 10개로 잡으면 $WS(t_1)$ 이 된다. t_1 에서 워킹셋은 1,2,4,5,6,7이고 워킹셋 사이즈는 5이다. 마찬가지로 t_2 에서 워킹셋은 3,4이고 사이즈는 2이다.

각 프로세스의 워킹셋 사이즈의 총합을 D 라고 칭하고 m 은 메모리의 크기라고 할 때, D 가 m 보다 크면 스레싱이 발생하게 된다.

3. I/O 인터페이스

1) 입출력장치의 매핑

▶ 컴퓨터 시스템에서 입출력 장치(I/O devices)는 CPU와 메모리에 데이터를 전송하거나 그들로부터 데이터를 받아오는 역할을 한다. 이러한 장치들을 효율적으로 관리하기 위해, 시스템은 입출력 장치를 주소 공간에 매핑한다. 이렇게 하면 CPU가 입출력 장치를 메모리의 일부처럼 취급하여 접근할 수 있다.

▶ 입출력장치 매핑 방식

(1) 메모리 매핑 입출력 (Memory Mapped I/O): 이 방식에서는 입출력 장치가 메인 메모리의 일부분처럼 취급된다. 즉, 각각의 입출력 장치에 대응하는 메모리 주소가 있으며, 해당 주소로 데이터를 읽고 쓸 수 있다. CPU가 그 주소로 읽기 또는 쓰기 명령을 내릴 때마다 실제로는 해당하는 I/O 디바이스와 통신하게 된다.

(2) 포트 매핑 입출력 (Port-Mapped I/O or Isolated I/O): 이 방식에서는 특정 명령어들(예: IN, OUT 등)이 사용되어 별도의 '입력/출력 공간'에 있는 포트와 직접 통신한다. 각각의 포트 번호가 다른 하드웨어 장치와 연결되어 있으며, 이들은 별도의 이름 공간(address space)을 차지한다.

▶ I/O 모듈

(1) 입출력 장치의 제어와 타이밍 조정: 입출력 모듈은 특정 입출력 장치를 제어하고, 데이터 전송을 위한 적절한 타이밍을 조정한다.

(2) 데이터 버퍼링: 데이터 버퍼링은 CPU와 입출력 장치 간의 속도 차를 보완하기 위해 사용된다. CPU가 처리할 수 있는 속도와 입출력 장치에서 데이터를 전송하는 속도 사이에는 큰 차이가 있을 수 있으며, 이 때문에 느린 입출력 작업으로 인해 CPU가 대기 상태에 빠지는 것을 방지하기 위해 데이터 버퍼링을 사용한다.

(3) 주변장치와 중앙처리장치 간 전송 속도 차이 극복: 이 역시 데이터 버퍼링과 관련된 기능으로, CPU와 주변장치 사이에서 발생할 수 있는 속도 차를 보완한다.

2) 폴링, 인터럽트

▶ 폴링 방식(Polling) : 컴퓨터 과학에서 특정 장치의 상태를 주기적으로 확인하여 요청을 처리하는 방법을 의미한다. 이는 주로 입출력 장치에서 사용되며, 컴퓨터가 데이터를 전송하거나 받을 준비가 되었는지 확인하기 위해

사용된다.

- 장점 : 구현이 단순하고 직관적이다.
- 단점 : CPU가 지속적으로 장치의 상태를 확인해야 하므로 자원과 시간을 낭비한다.

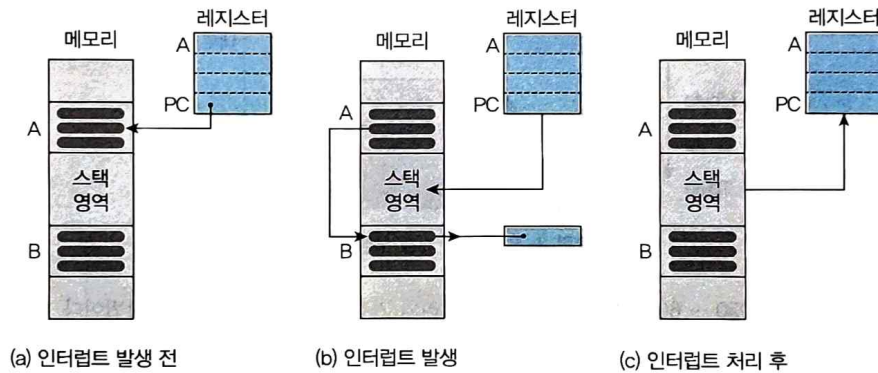
▶ 인터럽트(Interrupt) : 컴퓨터 시스템에서 매우 중요한 개념으로, CPU가 현재 실행 중인 작업을 일시적으로 중단하고, 시스템의 다른 부분에서 발생한 사건을 처리할 수 있게 하는 메커니즘이다.

▶ 인터럽트의 종류

- (1) 입출력(I/O) 인터럽트 : 해당 입출력 하드웨어가 주어진 입출력 동작을 완료하였거나 입출력의 오류 등이 발생했을 때 CPU에 요청하는 인터럽트이다.
- (2) 외부 인터럽트 : 시스템 타이머에서 일정한 시간이 만료된 경우나 오퍼레이터가 콘솔 상의 인터럽트 키를 입력한 경우 또는 다중 처리 시스템에서 다른 처리기로부터 신호가 온 경우 등에 발생한다.
- (3) SVC(SuperVisor Call) 인터럽트 : 사용자 모드에서 실행 중인 프로그램이 커널 모드의 서비스나 기능을 요청할 때 발생한다.
- (4) 기계 검사 인터럽트 : 컴퓨터 자체의 기계적인 장애나 오류로 인한 인터럽트이다.
- (5) 프로그램 오류 인터럽트 : 주로 프로그램 실행 오류로 발생한다.
- (6) 재시작 인터럽트 : 오퍼레이터가 콘솔의 재시작 키를 누를 때 발생한다.

▶ 인터럽트의 동작 과정

- (1) CPU가 입출력 관리자에게 입출력 명령을 보낸다.
 - (2) 입출력 관리자는 명령받은 데이터를 메모리에 가져다 놓거나 메모리에 있는 데이터를 저장장치로 옮긴다.
 - (3) 데이터 전송이 완료되면 입출력 관리자는 완료 신호를 CPU에 보낸다.
- * 인터럽트 방식에는 많은 주변장치 중 어떤 작업이 끝났는지를 CPU에 알려주기 위해 인터럽트 번호(IRQ Interrupt ReQuest)를 사용한다.

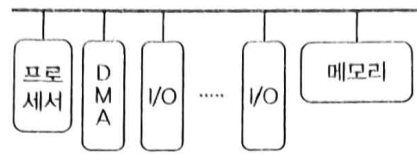


[그림 2-6] 인터럽트 처리 과정

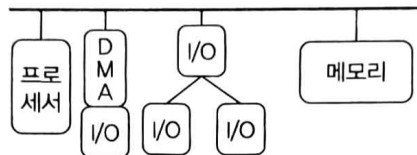
- (a) 인터럽트가 도달하기 전에 프로그램 A를 실행한다고 가정하자. 프로그램 카운터는 현재 명령어를 가리킨다.
- (b) 프로세서에 인터럽트 신호가 도달하여 현재 명령어를 종료하고 레지스터의 모든 내용을 스택 영역에 보낸다. 그리고 프로그램 카운터에서는 인터럽트 처리 프로그램(프로그램B)의 시작 위치를 저장하고 제어를 넘긴 프로그램 B를 실행한다.
- (c) 인터럽트 처리 프로그램을 완료하면 스택 영역에 있던 내용을 레지스터에 다시 저장하며, 프로그램 A가 다시 시작하는 위치를 저장하고 중단했던 프로그램 A를 재실행한다.

3) DMA

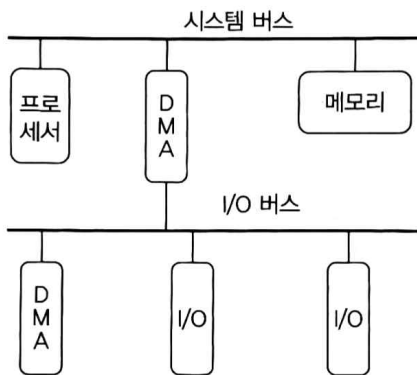
- ▶ 직접기억장치접근(DMA, Direct Memory Access) : MPU에 의한 프로그램 처리에 개입하지 않고 하드웨어 만으로 입출력장치와 메모리, 또는 메모리와 메모리 사이의 데이터를 교환하는 방식이다.



(a) 단일 버스, 분리식 DMA



(b) 단일 버스, 통합 DMA-I/O



(c) I/O 버스

[그림 2-8] 여러 가지 DMA 구성방법

: (a)는 모든 모듈이 한 개의 시스템 버스를 공유하며, 프로세서의 기능을 대신하는 DMA 모듈은 기억장치와 I/O 모듈 사이에 데이터 교환을 위해 프로그램 I/O를 사용한다.

(b)의 전송에 필요한 버스 사이클 수는 DMA와 I/O 기능들을 통합함으로써 많이 감소시킬 수 있다. 이 방법에서는 DMA 모듈과 I/O모듈들 사이에 별도의 경로가 존재하며, 시스템 버스는 사용하지 않는다.

(c)는 DMA모듈이 가지고 있는 I/O 인터페이스의 수를 한 개로 줄여주며, 시스템 확장을 쉽게 해주는 구성 방법이다.

(b),(c)의 시스템 버스는 DMA모듈이 기억장치와 데이터를 교환할 때만 사용된다. DMA와 I/O모듈 사이의 데이터 교환은 시스템 버스가 아닌 다른 통로를 통해 이루어진다.

▶ DMA 컨트롤러

: 일반적으로 컴퓨터 분야에서 데이터가 통과하는 토오를 채널이라고 하며,

MPU 대신 메모리와 직접적으로 데이터 교환을 하기 위한 하드웨어를 DMA 채널장치라고 한다.

(1) 인터럽트 요구 처리의 흐름

: MPU에서 DMA에 데이터 전송을 지시하면 DMA는 MPU의 동작과 독립하여 지시된 데이터 전송을 시작한다. 한편 MPU도 DMA의 동작과는 관계없이 다음 처리를 할 수 있다. 이처럼 DMA를 사용하여 데이터의 전송속도를 향상시켜 MPU의 부담을 줄일 수 있다.

DMA요청 → 버스요청 → 버스허가 → DMA확인 → 데이터 전송 → 인터럽트 요청 → 인터럽트 확인

(2) DMA 전송모드

: 단일 전송모드(바이트 전송모드)는 1바이트 전송할 때마다 버스의 사용을 요구하고 전송 후에 버스를 개방하여 전송하는 데이터를 한 번에 준비할 수 없는 경우에 채택하는 모드이다.

연속 전송모드(블록 전송모드)는 설정된 데이터 수의 전송이 종료될 때까지 버스를 점유한다. 연속하여 데이터 전송을 하는데 전용 제어 신호선의 상태에 따라 버스를 점유하고 개방한다. 미리 데이터 버퍼 등에 전송하는 데이터를 모아 준비할 수 있는 경우에 채택한다.

4) 입출력 버퍼링

▶ 입출력 버퍼링(I/O Buffering)은 입출력 작업의 효율성을 높이기 위한 기술로, 데이터를 일시적으로 저장하는 중간 메모리 영역인 '버퍼'를 사용한다.

▶ 단일 버퍼링 (Single Buffering): 한 번에 하나의 버퍼만 사용한다. 데이터는 이 버퍼에 쓰여지고, 이후에 목적지로 전송된다. 단일 버퍼링은 구현이 간단하지만, 데이터가 전송되는 동안 CPU나 I/O 장치 중 하나는 대기 상태가 될 수 있다.

▶ 이중 버퍼링 (Double Buffering): 두 개의 버퍼를 사용하여 한 쪽에서 데이터를 읽고 다른 쪽에서 데이터를 쓰는 동시 진행을 가능하게 한다. 이 방식은 그래픽스 렌더링과 같이 많은 양의 데이터 처리가 필요한 상황에서 유용하다.

▶ 순환 버퍼링 (Circular Buffering): 고정된 크기의 버퍼가 순환하는 형태로 사용된다. '머리'와 '꼬리'라는 두 포인터가 각각 입력 위치와 출력 위치를 추적하며, 둘 사이에 있는 공간에 데이터가 저장된다.

< 주변장치 >

1. 입출력 포트

1) GPIO의 설정과 이용

▶ GPIO : GPIO는 General Purpose Input/Output의 약자로, 일반적인 목적으로 사용되는 입력/출력 포트를 의미한다. GPIO 핀은 주로 마이크로컨트롤러나 마이크로프로세서 등에서 제공되며, 이들은 다양한 전자 장치와 상호작용하는 데 사용된다.

GPIO 핀은 하드웨어에 따라 그 수가 다르며, 각 핀은 프로그램을 통해 입력 모드 또는 출력 모드로 설정될 수 있다.

▶ GPIO의 변수형

(1) 불리언(Boolean) : GPIO 핀의 상태를 나타내기 위해 가장 많이 사용되는 데이터 타입이다. 'True' 또는 'False', 혹은 'HIGH'와 'LOW'로 표현되며, 이들은 각각 전류가 흐르고 있음(일반적으로 3.3V 또는 5V), 전류가 흐르고 있지 않음(0V)을 의미한다.

(2) 정수(unsigned Integer) : 일부 상황에서 GPIO 핀의 상태를 정수로 표현하기도 한다. 이 경우, '1'과 '0'이 각각 HIGH와 LOW 상태를 나타낸다.

▶ GPIO의 기능

(1) 디지털 입력: GPIO 핀을 입력 모드로 설정하여 외부 장치로부터 디지털 신호를 읽을 수 있다. 예를 들어, 버튼이 눌러졌는지 아닌지 확인하거나 스위치의 상태를 체크하는 등의 작업에 사용된다.

(2) 디지털 출력: GPIO 핀을 출력 모드로 설정하여 외부 장치에 디지털 신호를 보낼 수 있다. 예를 들어, LED를 켜고 끄거나, 릴레이를 제어하는 등의 작업에 사용된다.

(3) PWM(Pulse Width Modulation): 일부 GPIO 핀은 PWM 신호 생성이 가능하다. PWM은 디지털 출력을 가변적으로 제어해 아날로그와 유사한 효과를 내는 방법이다. 예를 들어, LED의 밝기 조절이나 서보 모터 제어 등에 주로 사용된다.

(4) 통신 인터페이스: 일부 GPIO 핀은 I2C, SPI, UART와 같은 통신 프로토콜을 지원한다. 이들 인터페이스는 센서나 다른 마이크로컨트롤러와 같은 외부 장치와 데이터 통신하는데 사용된다.

(5) 인터럽트 생성: 일반적으로 많은 마이크로컨트롤러에서 GPIO 핀은

'인터럽트'라는 특별한 신호 생성을 지원한다. 이 인터럽트 메커니즘은 CPU가 중요한 이벤트(예: 버튼 클릭 등) 발생 시 즉시 반응할 수 있도록 해준다.

- (6) ADC (Analog to Digital Converter): 몇몇 마이크로컨트롤러들에서 몇몇 GPIO 핀들은 아날로그 값을 디지털 값으로 변환하는 ADC 기능을 가진다.

▶ GPIO 기본 함수

- (1) `gpio_request()`: 리눅스 커널에서 GPIO 핀을 사용하기 전에 호출해야 하는 함수이다. 이 함수를 통해 GPIO 핀의 사용 권한을 요청하고, 시스템이 해당 핀을 안전하게 사용할 수 있도록 준비한다.
- (2) `gpio_free()`: `gpio_request()`로 요청한 GPIO 핀의 사용 권한을 해제하는 함수이다. 이를 통해 다른 드라이버나 프로세스가 해당 핀을 사용할 수 있게 된다.
- (3) `gpio_direction_input()` / `gpio_direction_output()`: GPIO 핀의 방향(입력 또는 출력)을 설정하는 함수이다.
- (4) `gpio_set_value()` / `gpio_get_value()`: 출력 모드에서 GPIO 핀의 상태를 설정하거나 입력 모드에서 현재 상태를 읽어오는 함수이다.
- (5) `gpio_to_irq()` / `irq_to_gpio()`: GPIO와 인터럽트 사이의 매핑 관계를 설정하는데 쓰이는 함수이다.

2) 입·출력 레지스터(Command/Status)

: 컴퓨터 시스템에서 중앙 처리 장치(CPU)와 주변 장치 간에 데이터를 전송하는데 사용되는 특수한 메모리 위치이다. 이들 레지스터는 CPU가 주변 장치에 명령을 내리거나 상태를 확인하는데 사용되며, 일반적으로 Command Register와 Status Register로 나뉜다.

▶ Command Register: Command Register는 CPU가 주변 장치에 작업을 지시하기 위해 사용한다. 예를 들어, 디스크 드라이브에 데이터를 읽거나 쓰라고 지시하거나, 프린터에 출력을 시작하라고 명령하는 등의 작업이 여기에 해당한다.

▶ Status Register: Status Register는 주변 장치의 현재 상태를 보고하기 위해 사용된다. 예를 들어, 디스크 드라이브가 현재 동작 중인지 아닌지, 데이터가 준비되었는지 아닌지, 오류 조건이 있는지 없는지 등의 정보가 여기에 포함될 수 있다.

3) 입·출력 포트 멀티플렉싱(I/O port multiplexing)

: 하나의 입·출력 포트를 여러 개의 장치와 공유하도록 하는 기술이다. 이는

효율적인 리소스 관리를 가능하게 하며, 전체 시스템의 복잡성과 비용을 줄이는데 도움이 된다.

▶ 작동 원리 : 입·출력 포트 멀티플렉싱은 여러 데이터 소스에서 입력을 받거나, 여러 대상에 출력을 보낼 수 있게 해주는 회로를 사용한다. 이 회로는 일반적으로 멀티플렉서(multiplexer) 또는 디멀티플렉서(demultiplexer)라고 부른다.

▶ 응용 사례

- 마이크로 컨트롤러: 마이크로 컨트롤러에서 입출력 포트 멀티플렉싱을 사용하면 한정된 수의 GPIO(General Purpose Input/Output) 핀을 최대한 활용할 수 있다.
- 네트워크 스위칭: 네트워크 스위칭 장비에서 데이터 패킷들을 다수의 입력 채널들 중 하나에서 받아들여, 다른 적절한 출력 채널로 보내기 위해 많이 사용된다.
- 다중 사용자 시스템: 다중 사용자 컴퓨터 시스템에서도, 서버가 동시에 많은 클라이언트 액세스 요청들을 처리하기 위해 입·출력 포트 멀티플렉싱을 사용할 수 있다.

4) 데이터시트의 개념

: 데이터시트는 전자 부품, 반도체, IC 등의 제품에 대한 기술적인 세부 사항을 제공하는 문서이다. 제조업체가 제공하며, 해당 부품이나 장치의 작동 방식, 사양, 핀 구성, 시간 특성 등에 대한 중요한 정보를 포함한다.

- (1) 일반 설명: 제품에 대한 간략한 개요와 주요 특징을 나열한다.
- (2) 핀 배치 및 기능: 각 핀의 이름과 그 기능을 설명한다.
- (3) 전기적 특성: 동작 전압 범위, 최대/최소 동작 온도 등의 주요 전기적 파라미터를 명시한다.
- (4) 동작 원리: 장치가 어떻게 작동하는지에 대한 상세 설명이 포함될 수 있다.
- (5) 응용 회로 예제: 일반적인 사용 사례 또는 응용 회로를 보여준다.
- (6) 패키징 정보: 패키지 형태와 크기 등을 나타내며, PCB(Printed Circuit Board) 설계에 필요할 수 있다.

2. 주요 주변장치

1) 시리얼 포트

▶ 시리얼 포트(Serial Port)는 데이터를 시리얼(즉, 한 번에 한 비트씩)로

전송하는 컴퓨터 하드웨어 인터페이스이다. 이는 비트를 나란히 배치하여 한 번에 전송하는 병렬 포트와 대조적이다.

▶ 시리얼 포트 유형

- (1) RS-232: 이것은 오래된 유형의 시리얼 포트로, 흔히 DB9 또는 DB25 커넥터를 사용한다. RS-232는 예전의 모뎀, 마우스, 프린터 등과 같은 장치와 컴퓨터 사이에서 널리 사용되었다.
- (2) USB (Universal Serial Bus): USB는 현대적인 형태의 시리얼 포트로, 많은 종류의 장치와 연결할 수 있다. USB 허브를 통해 여러 장치를 동시에 연결할 수 있으며 전원 공급도 가능하다.

▶ 시리얼 인터페이스

- (1) UART: UART 통신은 비동기식으로 작동하며 전용 데이터 선을 갖고 있어 디바이스 간 직접 통신을 가능하게 한다.
- (2) SPI: SPI 인터페이스는 동기식 직렬 인터페이스로서 주로 마이크로컨트롤러와 그 주변 장치간에 데이터를 고속으로 교환하는 데 사용된다.
- (3) I2C: I2C 인터페이스 역시 동기식 직렬 인터페이스지만 SPI보다 느리다. 그러나 이 방법은 하나의 클럭 신호선과 하나의 데이터 신호선만 필요하기 때문에 회로 설계가 단순해진다.

2) 타이머

: 타이머는 임베디드 시스템에서 매우 중요한 역할을 하는 주변 장치이다. 특정 시간 간격에 따라 작업을 수행하거나, 정확한 지연을 생성하거나, 이벤트의 타이밍을 측정하는 등 다양한 용도로 사용된다.

▶ 기본 구조

: 대부분의 타이머는 기본적으로 카운터와 비슷하다. 내부 클럭 신호에 의해 주기적으로 증가(또는 감소)하며, 특정 값에 도달하면 오버플로우(또는 언더플로우) 상태를 생성하고 인터럽트를 발생시킬 수 있다.

타이머 레지스터는 보통 다음과 같은 것들을 포함한다.

- (1) 카운트 레지스터: 실제로 값을 증가시키거나 감소시키는 레지스터이다.
- (2) 제어 레지스터: 타이머의 동작 방식을 제어하는 데 사용되며, 시작/중지, 카운트 방향(증가/감소), 인터럽트 활성화 등의 설정을 포함할 수 있다.
- (3) 비교 또는 캡처 레지스터: 특정 값과 현재 카운트 값을 비교하거나, 외부 이벤트 발생 시점의 카운트 값을 캡처하는 등 다양한 용도로 사용된다.

▶ 주요 용도

- (1) 일정 시간 간격으로 작업 수행: 타이머 인터럽트를 사용하여 일정 시간

간격으로 코드를 실행할 수 있다.

(2) 정확한 지연 생성: 코드 실행 사이에 정확한 지연을 만드는 것은 소프트웨어만으로는 어렵다. 하드웨어 타이머를 사용하여 정확한 지연을 만들 수 있다.

(3) 입력 신호 캡처 및 PWM 출력: 외부 이벤트(예: 센서 입력)와 연관된 시간을 측정하거나, PWM(Pulse Width Modulation) 신호와 같은 타이밍 기반 출력을 생성하는 데 사용할 수 있다.

▶ 사용 예시

: 실제 임베디드 시스템에서 타이머는 다양한 방식으로 사용된다. 예를 들어, 마이크로컨트롤러에서 LED를 1초마다 깜빡이게 하려면, 타이머를 설정하여 1초 간격으로 인터럽트를 발생시키고 인터럽트 서비스 루틴에서 LED 상태를 토글하는 코드를 실행하면 된다.

다른 예로, 초음파 거리 센서는 보통 PWM 신호로 거리 정보를 출력한다. 이 경우, 타이머의 입력 캡처 기능을 사용하여 PWM 신호의 펄스 폭을 측정하고, 이 값을 거리 정보로 변환할 수 있다.

3) A/D, D/A 변환

▶ A/D 변환: 이는 연속적인 아날로그 신호를 이산적인 디지털 값으로 변환하는 과정이다. 예를 들어, 마이크로폰은 소리를 아날로그 전기 신호로 바꾸는데, 이 신호를 컴퓨터에서 처리하려면 디지털 데이터로 바꿀 필요가 있다. A/D 컨버터(또는 ADC, Analog-to-Digital Converter)는 이런 작업을 수행한다.

▶ A/D 변환기

- (1) 계수비교형 ADC (Counter-ramp ADC): 이 방식은 입력 아날로그 신호를 디지털로 변환하기 위해 카운터를 사용한다. 카운터는 일정 시간 동안 증가하고, 그 값이 입력 아날로그 신호와 일치할 때 변환을 중지한다. 이 방식은 구현이 간단하지만, 변환 속도가 느리다.
- (2) 축차비교형 ADC (Successive Approximation Register, SAR ADC): 이 방식은 바이너리 검색 알고리즘을 사용하여 아날로그 입력값을 디지털 출력값으로 변환한다. SAR ADC는 빠르고 정확하며, 대부분의 일반적인 응용 분야에 적합하다.
- (3) 이중 적분형 ADC (Dual Slope ADC): Dual Slope 방식은 보통 DMMs(Digital Multimeters)와 같이 비교적 저속이면서 정밀도가 요구되는 애플리케이션에서 사용된다.
- (4) 병렬비교형 ADC (Flash or Parallel ADC): Flash ADC는 모든 가능한

비트 조합에 대해 별도의 컴퍼레이터가 있는 가장 빠른 A/D 변환 방법이다. 이러한 속도는 초고속 응용 분야에서 필요하지만, 하드웨어 복잡성과 비용 때문에 일반적으로 비트 수가 작은 경우에만 사용된다.

▶ D/A 변환: 반대로 D/A 변환은 디지털 값을 아날로그 신호로 바꾸는 과정이다. 예를 들어, 디지털 오디오 파일을 스피커에서 재생하려면 그 파일의 디지털 데이터를 아날로그 오디오 신호로 바꿀 필요가 있다. D/A 컨버터(또는 DAC, Digital-to-Analog Converter)가 이 작업을 한다.

▶ D/A 변환기

(1) 무계저항형 DAC (Weighted-resistor DAC or Binary-scaled DAC):

이 방식에서 각 디지털 입력 비트는 해당하는 저항 네트워크를 통해 전류나 전압을 제어하며, 각 저항 값은 2진수 가중치를 나타낸다.

(2) 사다리형 저항형 DAC (R-2R ladder DAC or Resistor ladder DAC): R-2R 사다리 회로 구조를 활용하여 다양한 출력 값을 생성하는 것으로 구조상의 장점으로 인해 가장 많이 사용되는 DAC 유형인 것으로 알려져 있다.

(3) 펄스 폭 변조를 이용한 DAC (Pulse Width Modulation, PWM DAC): PWM 방식은 디지털 신호의 듀티 사이클을 조절하여 아날로그 값에 대응시키는 방식이다. 이 방법은 간단하고 저렴하지만, 고주파 필터가 필요하며 노이즈에 취약할 수 있다.

▶ ADC (Analog-to-Digital Converter): ADC는 연속적인 아날로그 신호를 이산적인 디지털 값으로 변환한다. 이 과정은 주로 샘플링, 양자화, 부호화의 세 단계로 이루어진다.

- 샘플링: 연속적인 아날로그 신호에서 일정 간격으로 값을 추출한다.

- 양자화: 샘플링된 각 값에 가장 가까운 가능한 디지털 값(레벨)을 할당한다.

- 부호화: 양자화된 값을 해당하는 디지털 코드로 변환한다.

▶ n비트 ADC : "n비트"는 ADC가 출력할 수 있는 디지털 값의 해상도를 나타낸다. "n"이 클수록 ADC는 더 많은 수준의 아날로그 값을 구별할 수 있으며, 따라서 더 높은 해상도와 정밀도를 가질 수 있다.

예를 들어, 3비트 ADC는 $2^3 = 8$ 개의 다른 아날로그 값을 구별할 수 있다.

즉, 가능한 디지털 출력 값은 000부터 111까지이다. 반면에 8비트 ADC는 $2^8 = 256$ 개의 다른 아날로그 값을 구별할 수 있으므로 훨씬 더 많은 정보를 캡처하고 더 정확한 디지털 표현을 제공할 수 있다.

ex) 3비트 출력 ADC에서 기준전압=4.2V DC일 때 입력 2.4V 전압에 대한

디지털 출력 중 옳은 것은?

- ① 100 ② 110 ③ 011 ④ 101

sol) 3비트 ADC (Analog-to-Digital Converter)의 동작을 이해하는 것이 중요하다. 3비트 ADC는 아날로그 입력을 8개(2^3)의 다른 디지털 레벨로 변환할 수 있다.

기준 전압이 4.2V인 경우, 각 디지털 레벨은 기준 전압을 8로 나눈 값, 즉 $4.2V / 8 = 0.525V$ 가 된다.

입력 전압이 2.4V인 경우, 이를 각 레벨 값으로 나누면 몇 번째 레벨에 해당하는지 알 수 있다.

$$2.4V / 0.525V \approx 4.57$$

여기서 중요한 점은 ADC가 반올림하거나 올림하지 않고 '버림'을 사용한다는 것이다 (즉, 소수점 아래를 모두 제거). 그러므로 여기서는 "레벨 4"(100(2))에 해당하게 된다.

▶ DAC (Digital-to-Analog Converter): DAC는 디지털 값을 연속적인 아날로그 신호로 변환한다. DAC 작동 원리는 ADC와 반대 방향으로 진행되며, 입력 받은 이진 코드에 대응하는 전압 또는 전류를 출력하여 아날로그 신호를 생성한다.

▶ n비트 DAC : "n비트"는 DAC의 해상도를 나타낸다. "n"이 클수록 DAC는 더 많은 수준의 디지털 값을 구별할 수 있으며, 따라서 더 높은 해상도와 정밀도를 가질 수 있다.

예를 들어, 3비트 DAC는 $2^3 = 8$ 개의 다른 디지털 값을 구별할 수 있다. 즉, 가능한 디지털 입력 값은 000부터 111까지이다. 반면에 8비트 DAC는 $2^8 = 256$ 개의 다른 디지털 값을 구별할 수 있으므로 훨씬 더 많은 정보를 캡처하고 더 정확한 아날로그 출력을 제공할 수 있다.

ex) 5비트 D/A 변환기가 0.1V의 분해능을 갖는다고 할 때,

최대출력전압(V)과 백분율 분해능(R)은 약 얼마인가?

- ① $V=2.8V$, $R=3.23\%$
② $V=3.1V$, $R=3.23\%$
③ $V=2.8V$, $R=6.43\%$
④ $V=3.1V$, $R=6.46\%$

sol) 5비트 D/A 변환기는 총 $2^5 = 32$ 개의 다른 출력 레벨을 가질 수 있다. 각 레벨의 분해능이 0.1V라고 주어졌으므로, 최대 출력 전압은 각 레벨 분해능과 총 레벨 수를 곱하여 계산할 수 있다.

$$V_{\max} = \text{분해능} * \text{총 레벨 수}$$

$$= 0.1V * 32$$

$$= 3.2V$$

따라서 최대 출력 전압은 약 3.2V이다.

백분율 분해능(R)은 ADC나 DAC의 해상도를 백분율로 표현한 것으로, 일반적으로 아래와 같이 정의된다.

$$R = (1 / \text{총 레벨 수}) * 100\%$$

$$= (1 / 32) * 100\%$$

$$\approx 3.125\%$$

그러므로 백분율 분해능은 약 3.125%이다.

그러므로 이 문제에 대한 정답은 V=3.2V, R=3.125%에 가장 근접한

㉔ 'V=3.1V, R=3.23%'가 된다.

4) 각종 센서(초음파, 적외선, 온도)

: 임베디드 시스템에서 사용하는 센서는 환경에서 필요한 데이터를 수집하고, 이를 전기 신호로 변환하여 시스템이 이해할 수 있는 정보로 만드는 역할을 한다.

▶ 초음파 센서

: 초음파 센서는 소리의 반사 원리를 이용하여 물체와의 거리를 측정한다. 이러한 종류의 센서는 일반적으로 두 개의 요소로 구성되어 있다. 하나는 초음파를 발생시키고 다른 하나는 반사된 신호를 감지한다.

· 작동 원리

(1) 초음파 발생기가 고주파 음파(사람 귀에 들리지 않을 정도로 높은 주파수)를 방출한다.

(2) 이 음파가 물체에 부딪치면 반사되어 동일한 방향으로 돌아온다.

(3) 감지기가 반사된 음파를 감지하면 타이머 기능을 사용하여 원래의 신호가 발송된 시간과 반사된 신호가 수신된 시간 사이의 차이(즉, 지연시간)를 계산한다.

(4) 속도 = 거리 / 시간 공식을 사용하여 이 지연시간을 거리 정보로 변환한다.

▶ 적외선(IR) 센서

: 적외선 센서는 적외선 방출 및/또는 감지 장치이다. IR 센서의 종류와 작동 원리에 따라 다양한 용도로 사용된다.

· 적외선 근접 스위치: 특정 거리 내에 물체가 있으면 출력을 변경하는 스위치이다.

· 적외선 리모트 컨트롤: TV 리모컨 등에서 별도의 인코딩/디코딩 회로와 함께

사용된다.

- 열 이미징 카메라: 서로 다른 온도 값을 가진 객체들에서 방출되는 적외선 에너지 패턴을 분석함으로써 온도 분포를 시각화한다.

▶ 온도 센서

: 온도 센서는 환경의 온도를 측정하는 장치이다. 여러 종류의 온도 센서가 있지만, 가장 일반적인 형태는 다음과 같다.

- 열전대(Thermocouple): 두 가지 다른 금속을 연결하면 그 접점에서 발생하는 전압이 온도에 따라 변화하는 원리를 이용한다.
- RTD(Resistance Temperature Detector): 저항이 온도에 따라 변화하는 특성을 이용한다. 대부분의 RTD는 플래티넘 와이어로 만들어져 있으며, 매우 정확한 측정 결과를 제공한다.
- NTC/PTC 열선항(Thermistor): RTD와 유사하게 저항이 온도에 따라 변하지만, 보통 반응 속도가 빠르고 비용이 낮다.
- 반도체 기반 센서: 직접적으로 전기 신호로 변환하여 디지털 값으로 출력할 수 있는 IC 형태의 센서이다.

5) 모션 센서(가속, 자이로, 지자기)

: 임베디드 시스템에서 사용하는 센서는 환경에서 필요한 데이터를 수집하고, 이를 전기 신호로 변환하여 시스템이 이해할 수 있는 정보로 만드는 역할을 한다.

▶ 가속도 센서

: 가속도 센서는 움직임, 진동 또는 중력에 의한 기울기와 같은 가속을 감지한다. 이러한 센서들은 주로 모바일 장치, 드론, 게임 컨트롤러 등에서 사용되며 MEMS(Micro-Electro-Mechanical Systems) 기술을 기반으로 한다.

- 작동 원리

- (1) 가속도센서의 내부에 작은 무게가 부착된 빔이 있다.
- (2) 장치가 움직일 때마다 무게가 이동하고 이때 발생하는 운동량 변화를 측정한다.
- (3) 이 운동량 변화를 전기적 신호로 변환하여 디지털 값으로 출력한다.

▶ 지자기 센서(자력계)

: 지자기 센서(또는 자력계)는 주변의 자기장 강도와 방향을 측정하는 장치이다. 스마트폰의 나침반이나 컴패스 앱에서 일반적으로 사용된다.

- 작동 원리

- (1) 지자기센서 내부에 작은 마그네틱 플럭스 콘덴서가 있으며 주변의 자기장에

반응한다.

(2) 자기장의 영향으로 인해 플렉스 콘덴서의 전압이 변경되고,
그 변경된 전압을 디지털 값으로 출력한다.

▶ 자이로 스코프

: 자이로 스코프(또는 짧게 '자이로')는 회전 운동을 측정하기 위해 사용된다.
드론 제어나 모바일 장치에서 화면 방향 조절 등 다양한 곳에서 사용된다.

· 작동 원리

(1) MEMS 기반이라면 작은 진동 패들이 있는데, 이 패들이 회전할 때
발생하는 코리올리스 효과를 측정한다.

(2) 이 효과를 전기적 신호로 변환하여 디지털 값으로 출력한다.

6) 입출력 버스(I2C, SPI등)

▶ I2C (Inter-Integrated Circuit)와 SPI (Serial Peripheral Interface)는
모두 마이크로컨트롤러와 주변 장치들 간에 데이터를 전송하기 위한 직렬 통신
프로토콜이다. 두 프로토콜은 비슷한 목적을 가지고 있지만, 사용하는 방식과
기능에서 차이가 있다.

▶ I2C(Inter-Integrated Circuit) : Philips Semiconductor (현재의 NXP
Semiconductors)가 1980년대에 개발한 버스 시스템이다. I2C는 주로
저속의 짧은 거리 통신에 사용되며, 하나의 마스터 장치와 여러 개의 슬레이브
장치 간에 데이터를 전송할 수 있다.

I2C 통신에서는 두 개의 선인 SDA (Serial Data Line)와 SCL (Serial
Clock Line)이 필요하다. SDA 선은 실제 데이터 비트를 전송하는데 사용되며,
SCL 선은 클럭 신호를 제공하여 데이터 비트가 언제 읽혀야 하는지를
지정한다.

▶ SPI(Serial Peripheral Interface) : SPI는 Motorola가 처음으로
개발하였으며, 상대적으로 빠른 속도로 데이터를 전송할 수 있는 직렬
인터페이스이다. SPI도 마찬가지로 하나의 마스터 장치와 하나 이상의 슬레이브
장치 사이에서 데이터를 교환할 수 있다.

SPI 인터페이스에서는 일반적으로 네 개의 선인 MOSI (Master Out Slave
In), MISO (Master In Slave Out), SCLK (Serial Clock), 그리고 SS
(Slave Select) 혹은 CS(Chip Select)라 부르기도 하는 슬레이브 선택선을
사용한다.

MOSI 선은 마스터에서 슬레이브로 데이터를 보내기 위해 사용되고, MISO
선은 반대 방향으로 데이터를 보내기 위해 사용된다. SCLK 선은 마스터에서

생성된 클럭 신호를 전달하며, SS 선은 특정 슬레이브 장치를 선택하기 위해 사용된다.

7) 통신장치(Ethernet, Wifi)

▶ Ethernet: Ethernet은 유선 네트워크 기술(무선은 대역폭을 사용하므로 초광대역인 UWB를 사용하지 않음)로, 1970년대에 Xerox PARC에서 처음 개발되었다. Ethernet은 컴퓨터를 서로 연결하여 로컬 영역 네트워크(LAN)를 구성하는 데 주로 사용된다. 데이터를 전송할 때 CSMA/CD 알고리즘을 사용한다. 이 기술은 전용 물리적 연결을 통해 고품질의 안정적인 연결을 제공하며, 대체로 Wi-Fi보다 빠른 데이터 전송 속도를 제공한다. Ethernet은 RJ-45 커넥터와 카테고리 5(Cat5) 이상의 케이블을 사용하여 장치들을 연결한다.

▶ Wi-Fi: Wi-Fi는 무선 네트워크 기술로, 1990년대에 처음 개발되었다. Wi-Fi는 컴퓨터, 스마트폰, 태블릿 등의 장치를 인터넷에 연결하는데 주로 사용된다. 이 기술은 물리적인 케이블 없이 자유롭게 이동하면서 인터넷 접속이 가능한 편리함을 제공하지만, 일반적으로 Ethernet보다 상대적으로 낮은 데이터 전송 속도와 신호 강도 문제가 있다.

▶ 두 기술 모두 IP(Internet Protocol)와 같은 공통의 네트워크 프로토콜을 사용하여 데이터 패킷을 교환하며 동작한다. 그러나 각각 다른 링크 계층 프로토콜(Ethernet에서는 Ethernet 프레임, Wi-Fi에서는 IEEE 802.11 프레임)을 사용하여 실제 데이터 비트를 전송한다.

일반적으로 사무실 환경에서는 안정성과 고품질의 접속성능 때문에 Ethernet가 선호되며, 가정 환경이나 카페 등에서 편리함과 이동성 때문에 Wi-Fi가 많이 사용된다.

8) USB

▶ USB (Universal Serial Bus)는 컴퓨터와 주변기기를 연결하는 표준 연결 방식이다. USB는 데이터 전송과 함께 전력 공급도 가능하므로, 많은 기기들이 USB를 통해 충전되거나 작동한다.

▶ USB의 특징

(1) Plug-and-Play: USB 장치는 시스템에 연결되면 자동으로 인식되고 설정된다. 이를 통해 사용자가 별도의 드라이버 설치나 재부팅 없이 새로운 장치를 쉽게 추가할 수 있다.

(2) Hot Swappable: USB 장치는 컴퓨터가 실행 중일 때도 안전하게

연결하거나 분리할 수 있다.

- (3) Power Supply: USB 포트는 일반적으로 5V의 전력을 제공하여, 여러 가지 기기들을 충전하거나 작동시킬 수 있다.
- (4) Data Transfer Speed: 원래 USB 1.0 버전은 최대 12Mbps의 속도를 제공했다. 그러나 기술 발전에 따라 이후 버전에서는 더 높은 데이터 전송 속도가 가능해졌으며, 현재까지 개발된 최신 버전인 USB 4는 최대 40Gbps까지 지원한다.
- (5) Versatility: 다양한 유형의 데이터 (예: 오디오, 비디오, 파일 등)을 전송하기 위한 프로토콜을 지원한다.
- (6) Compatibility and Connectors: 다양한 종류의 커넥터 타입 (예: Type-A, Type-B, Mini-USB, Micro-USB, Type-C 등)이 있으며 이들 간에 호환성을 유지하는 어댑터 또한 존재한다.

▶ USB에서 사용되는 신호의 인코딩/디코딩 방법

: USB(Universal Serial Bus)는 데이터 전송을 위해 NRZI (Non-Return to Zero Inverted) 인코딩 방식을 사용한다. 이 방식은 비트 스트림에서 '0'이 있을 때만 신호를 토글(전환)하는 방식이다. 즉, '1'이 연속으로 오면 신호 상태는 변하지 않지만, '0'이 오면 신호 상태가 전환된다. 하지만 이러한 인코딩 방식은 연속된 1의 경우에 클럭 정보를 잃게 된다. 따라서 USB에서는 비트 스테핑(bit stuffing) 기법을 추가로 사용한다. 비트 스테핑은 일정 개수의 연속된 '1'(USB 2.0에서는 6개) 이후에 강제로 '0'을 삽입하여 클럭 복구를 돕는다.

9) 전원제어 인터페이스

: 전원 제어 인터페이스는 임베디드 시스템에서 중요한 역할을 한다. 이 인터페이스는 전원 공급 장치와 시스템 사이의 통신을 가능하게 하며, 전원 상태를 모니터링하고, 필요에 따라 전원을 켜거나 끄는 등의 제어 기능을 수행한다.

▶ 주요 기능

- (1) 전원 상태 모니터링: 시스템은 전압 레벨, 전류 소비, 온도 등의 파라미터를 모니터링하여 정상 범위 내에 있는지 확인한다. 이 정보는 성능 최적화, 오류 진단 및 예방 유지 보수에 사용될 수 있다.
- (2) 전원 관리: 시스템은 절전 모드로 들어가거나 깨어나기 위해 필요한 조건을 감지하고 반응한다. 이는 배터리 구동 장치에서 특히 중요하며, 대기 시간과 작동 시간을 최대화하는 데 도움이 된다.
- (3) 보호 기능: 과부하, 과열, 단락 등의 위험 조건이 감지되면 시스템은

자체를 보호하기 위해 즉시 반응할 수 있다.

▶ 구현 방법

(1) 하드웨어 방식: 일반적으로 전용 IC(예: PMIC - Power Management Integrated Circuit)를 사용하여 직접적인 전력 관리 및 모니터링 기능을 수행한다.

(2) 소프트웨어 방식: 마이크로컨트롤러 또는 프로세서가 ADC(Analog-to-Digital Converter), GPIO(General Purpose Input/Output) 등과 같은 내장 주변장치를 사용하여 전력 상태를 모니터링하고 제어 신호를 출력한다.

10) 칩 셀렉트 로직(Chip Select Logic)

: 칩 선택 로직(Chip Select Logic)은 주로 버스 시스템에서 특정 장치를 활성화하거나 비활성화하는 데 사용되며, 여러 개의 장치가 동일한 데이터 및 주소 버스에 연결되어 있을 때 특히 중요하다.

▶ 작동 원리

(1) 마이크로프로세서 또는 마이크로컨트롤러는 데이터를 읽거나 쓸 때 해당 장치의 칩 선택 신호를 활성화한다.

(2) 이 신호는 해당 장치가 버스에서 데이터를 수신하거나 전송할 수 있도록 한다.

(3) 다른 모든 장치의 칩 선택 신호는 비활성화 상태이므로, 이들 장치는 버스에서 분리된다.

▶ 구현 방법

· 디코더: 디코더 ICs (Integrated Circuits)는 입력선에 대응하는 출력선을 활성화하여 여러 개의 칩 선택 신호를 생성하는데 사용된다.

· 마이크로컨트롤러/마이크로프로세서 내장 기능: 일부 마이크로컨트롤러와 마이크로프로세서에는 내장된 메모리 관리 유닛(MMU) 또는 하드웨어 칩 선택 기능이 있다.

▶ 중요성

· 버스 충돌 방지: 여러 개의 장치가 동시에 같은 버스에 데이터를 전송하려고 하면 '버스 충돌'이 발생할 수 있다. 칩 선택 로직은 한 번에 하나의 장치만 버스에 접근하도록 함으로써 이 문제를 방지한다.

· 주소 공간 관리: 각각 별도의 주소 공간을 가진 여러 개의 메모리 ICs가 있는 경우, 칩 선택 신호를 사용하여 필요한 메모리 IC만 접근할 수 있다.