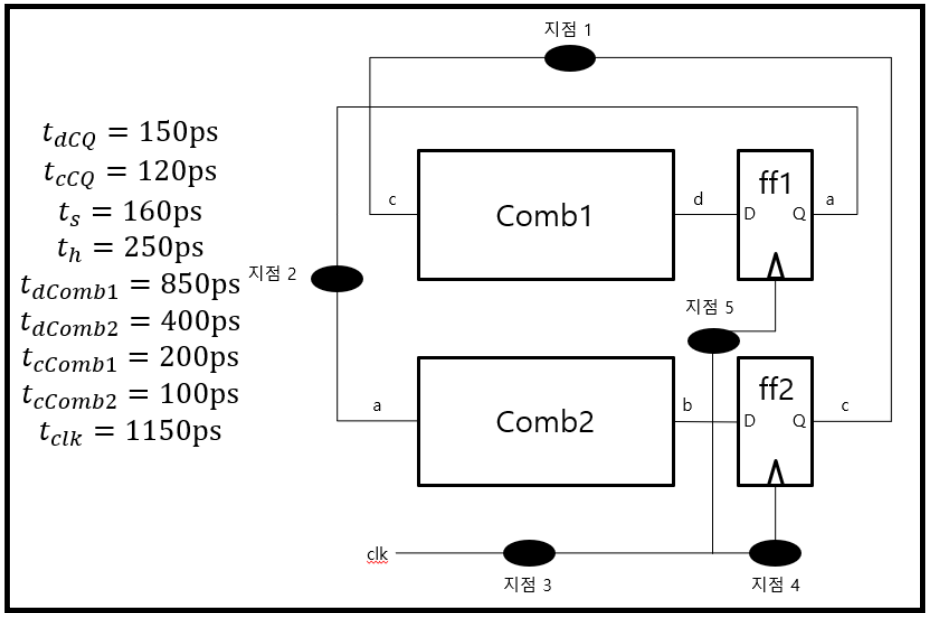
Problem Set 1 Answers – Timing

Problem 1

- Timing 관련 문제이다 아래 회로에 대한 질문에 답하여라. t_{dCQ}, t_{cCQ}, t_s, t_h는 각각 Flip-flop의 clock-to-Q propagation delay, clock-to-Q contamination delay, setup time, hold time을 의미하며 t_{dComb}, t_{cComb} 는 각각 Combinational logic의 propagation delay, contamination delay 그리고 t_{clk}는 clock period를 의미한다. (단, 현재 회로 상에서 Flip-flop ff1, ff2에 도착하는 clock signal time은 같다고 가정한다.)

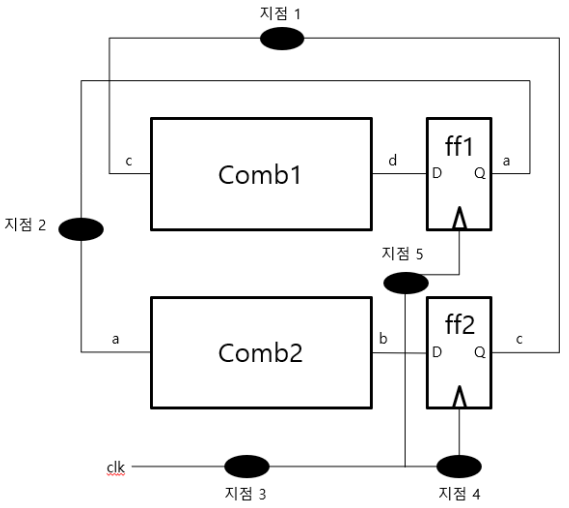
$t_{dCQ} = 150\text{ps}$
$t_{cCQ} = 120\text{ps}$
$t_s = 160\text{ps}$
$t_h = 250\text{ps}$
$t_{dComb1} = 850\text{ps}$
$t_{dComb2} = 400\text{ps}$
$t_{cComb1} = 200\text{ps}$
$t_{cComb2} = 100\text{ps}$
$t_{clk} = 1150\text{ps}$



Problem 1.1

- 아래 회로에 대해 가장 작은 hold time slack과 setup time slack 값을 각각 구하여라.

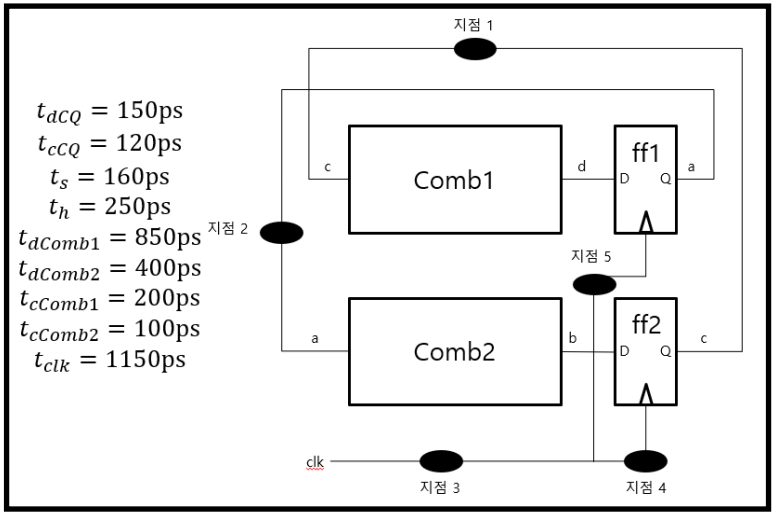
$t_{dCQ} = 150\text{ps}$
$t_{cCQ} = 120\text{ps}$
$t_s = 160\text{ps}$
$t_h = 250\text{ps}$
$t_{dComb1} = 850\text{ps}$
$t_{dComb2} = 400\text{ps}$
$t_{cComb1} = 200\text{ps}$
$t_{cComb2} = 100\text{ps}$
$t_{clk} = 1150\text{ps}$



정답: c → d setup time slack = $t_{clk} - (t_{dCQ} + t_{dComb1} + t_s)$ = -10ps
c → d hold time slack = $t_{cCQ} + t_{cComb1} - t_h$ = 70ps
a → b setup time slack = $t_{clk} - (t_{dCQ} + t_{dComb2} + t_s)$ = 440ps
a → b hold time slack = $t_{cCQ} + t_{cComb2} - t_h$ = -30ps
따라서 minimum hold time slack = -30ps, minimum setup time slack = -10ps

Problem 1.2

- 1.1에서 negative slack 이 발생할 경우, timing violation 이기 때문에 이를 해결해야 한다. 최소 개수의 buffer를 이용하여 timing violation을 완전히 해결하고자 할 때, 위 회로의 어느 지점(들)에 몇 개의 buffer를 삽입하면 되는가? 이유를 적어 설명하여라.(단, 사용할 buffer의 delay는 20ps이다.)



정답: 1. 지점 5에 buffer 1개 삽입 (clock skew로 인해 a → b hold, c → d setup time slack 각각 +20ps), 지점 2에 buffer 1개 삽입 (combination logic의 delay를 늘려 a → b hold time slack +20ps)
2. 또는 지점 5에 buffer 2개 삽입 (clock skew로 인해 a → b hold, c → d setup time slack 각각 +40ps)

Problem Set 2 Answers – Meta-stability and number systems

Problem 1.1

- 어떤 회로에 대하여, 해당 회로 외부에서 비동기 입력 signal a가 들어온다고 가정하자. 또한 이 회로의 clock period는 1ns이며, signal a을 입력으로 바로 받는 회로 내부의 flip-flop f에 대한 setup time = 30ps, hold time = 15ps이라고 가정하자. 이 경우, signal a의 state transition에 대해 flip-flop f의 출력 state가 meta-stability state에 빠질 확률은 몇 % 인가?

정답: Meta-stability state에 빠질 확률 = $\frac{t_s+t_h}{t_{clk}}$ = (30+15) / 1000 = 4.5%

Problem 1.2

- 1.1 에서 구한 확률이 너무 클 경우, 해당 회로를 개선할 필요가 있다. 이때 해결할 수 있는 방안으로 강의 시간에 Brute-Force Synchronizer를 소개하였다. 이것의 구조는 어떠한 것이며, 어떤 원리로 확률을 줄이는지 설명하여라.

정답 :

Flip-flop 여러 개를 연달아 붙임으로써 cycle 내에 meta-stability를 해결할 수 있도록 함.

확률을 줄이는 방법으로는 meta-stability 회복까지 기다리는 시간을 늘려주면 되는데, 이를 위해서는 clock enable을 사용하거나 serial FF 개수를 많이 두는 방법이 있음

Problem 2.1

- 십진법으로 나타내어진 숫자 -1.46을 s2.2 fixed-point format으로 변환하여라. 이 때, absolute error와 relative error를 각각 구하여라.

정답: -1.46 x 4 = - 5.84 → (round) -6 → (2s complement) 1010 → (sign bit) 110.10 (-1.5)
Absolute error : -1.46 + 1.5 = 0.04
Relative error : (0.04) / 1.46 = 0.0273 or 2.73 %

Problem 2.2

- 이진법 5E3 floating-point format(bias = 0)으로 나타내어진 숫자 01010110를 십진법으로 변환하여라.

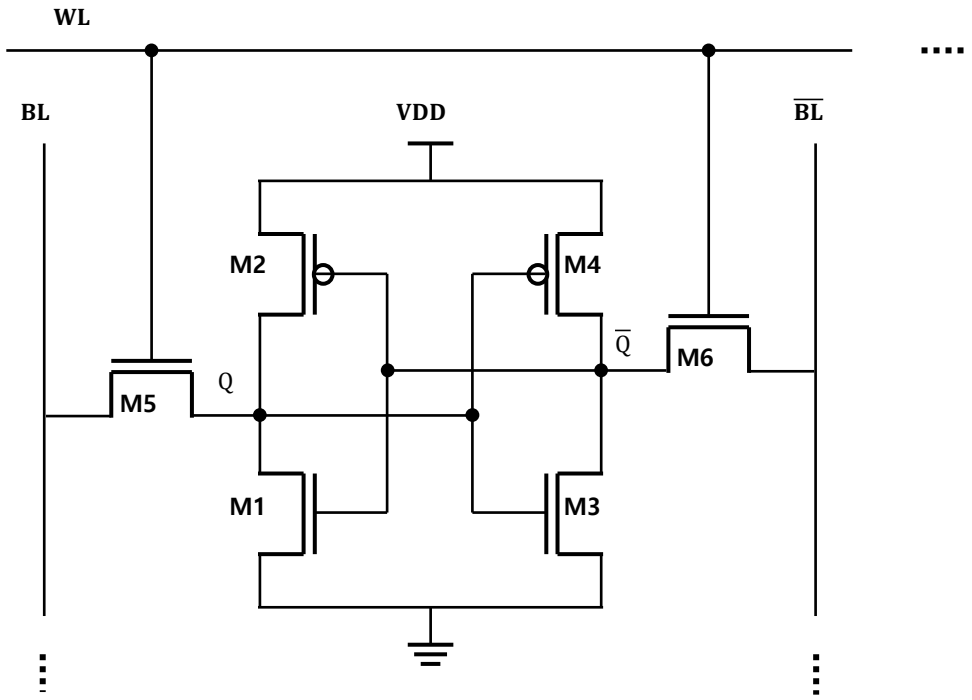
정답 :

010E10110
10110 → (decimal) 22
010 → (decimal) 2, bias = 0
22 x (1/32) x 2^(2-0) = 2.75

Problem Set 3 Answers – Memory cells and Huffman encoding

Problem 1

- 메모리 셀에 대한 문제이다. 아래 문제에 나온 칸을 채워라

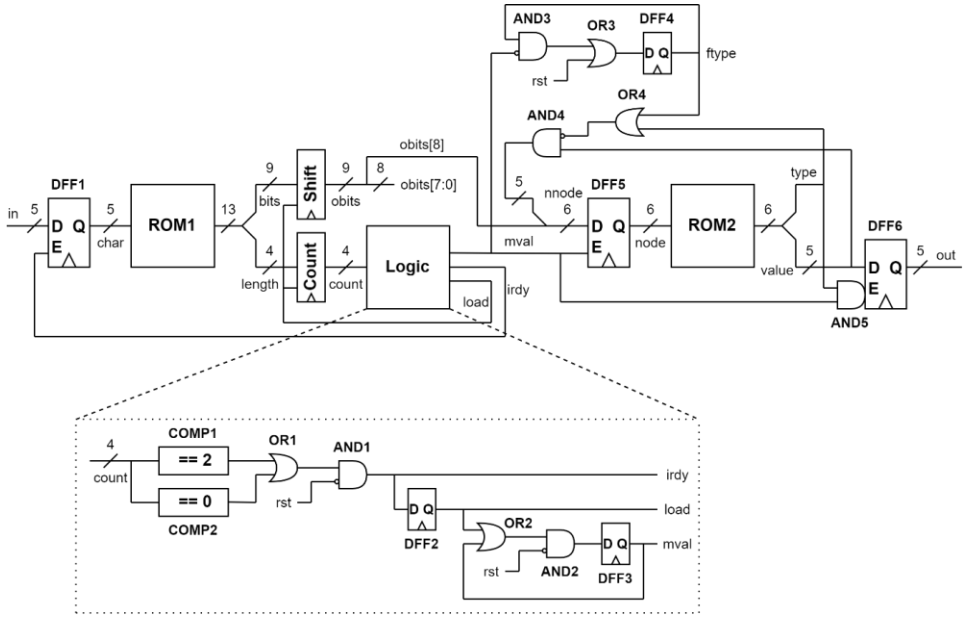


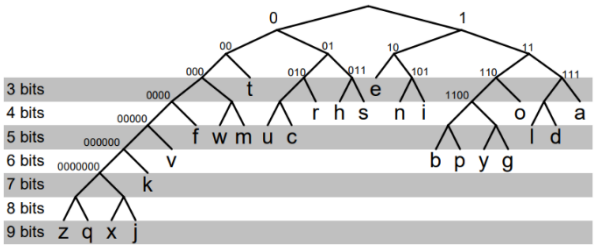
Problem 1

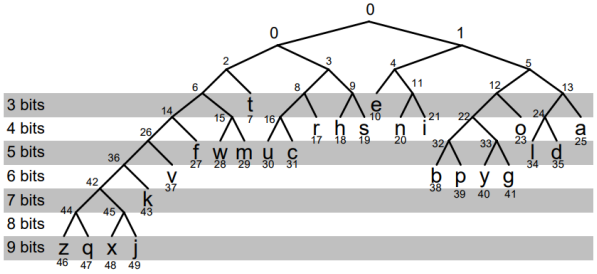
- SRAM write는 아래 2 단계로 작동된다.
  (단계 1) 저장하려는 data에 맞게 _BL, $\overline{BL}$_ 설정
  (단계 2) _WL_ 을 __high(1)__ 로 설정
- SRAM read는 아래 3 단계로 작동된다.
  (단계 1) __BL, $\overline{BL}$___ 을 _high(1)로 charge 시킴_
  (단계 2) ___WL____ 을 _high(1) _ 로 설정
  (단계 3) __BL, $\overline{BL}$__ 의 한쪽이 _discharge 되는__것을 감지
- Flash write는 gate에 강한 전압을 걸어 주어 전자가 Oxide를 통과하는 원리를 이용한 것이다. 즉, 전자가 floating gate에 들어가면 cell이 program이 되었다고 하며, 반대로 빠져나오면 erase 되었다고 한다. Program 된 cell은 threshold voltage가 높아지고 Erase 된 cell은 threshold voltage가 낮아진다.

Problem 2

- 다음 그림은 Huffman encoder의 output이 Huffman decoder의 input으로 들어가는 회로이다. Encoder의 input은 5-bit code이며, 여기서 A = 1, B = 2, ⋯, Z = 26으로 대문자 및 소문자 알파벳에 대한 ASCII code의 하위 5-bit와 일치한다. 좌측 하단의 그림은 code를 tree 형태로 보여주며, tree의 root에서 leaf로 가는 경로는 해당 문자에 대한 code를 제공한다. 예를 들어, 문자 E는 오른쪽, 왼쪽, 왼쪽으로 이동함으로써 3-bit 문자열 100으로 표현된다. 우측 하단의 그림은 decoding을 용이하게 하기 위해 labelling한 tree의 node들을 보여준다. 각 node에는 주소 역할을 하는 정수가 할당된다.







Problem 2.1

- **ROM1**은 각 문자의 length와 문자열을 저장한다. Input에 맞는 output을 작성하라.

정답 :

Input: 5'b01011 → ROM1 → Output: 13'b 0 1 1 1 0 0 0 0 0 0 1 0 0 (K)

Input: 5'b01001 → ROM1 → Output: 13'b 0 1 0 0 1 0 1 1 0 0 0 0 0 (I)

Input: 5'b01101 → ROM1 → Output: 13'b 0 1 0 1 0 0 0 1 1 0 0 0 0 (M)

Problem 2.2

- **ROM2**는 tree를 저장한다. Input에 맞는 output을 작성하라.

정답 :

Input: 6'b000001 → ROM2 → Output: 6'b 0 0 0 0 1 0 (node: 1)

Input: 6'd000101 → ROM2 → Output: 6'b 0 0 0 1 1 0 (node: 5)

Input: 6'd001100 → ROM2 → Output: 6'b 0 0 1 0 1 1 (node: 12)

Input: 6'd010111 → ROM2 → Output: 6'b 1 0 1 1 1 1 (node: 23)

Problem 2.3

- Huffman encoder의 input에 문자가 계속 입력된다면 obits[8] 신호는 일정 시간 이후로 항상 의미 있는 값만을 출력한다. 이를 가능케하는 이유를 clock cycle 단위로 자세히 설명하고 이와 가장 관련 있는 **module 이름** (볼드 처리된 것) **하나**를 적어라.

정답 :

카운터가 count==2(마지막에서 두 번째 bit)에 도달하면 irdy가 다음 문자를 DFF1에 로드하고

카운터가 count==1(마지막 bit)에 도달하면 load가 다음 문자의 length와 문자열을

counter 및 shift register에 로드한다.

가장 관련 있는 module 이름: COMP1

Problem 2.4

- Huffman decoder에 ftype 신호가 필요한 이유를 설명하라.

정답 :

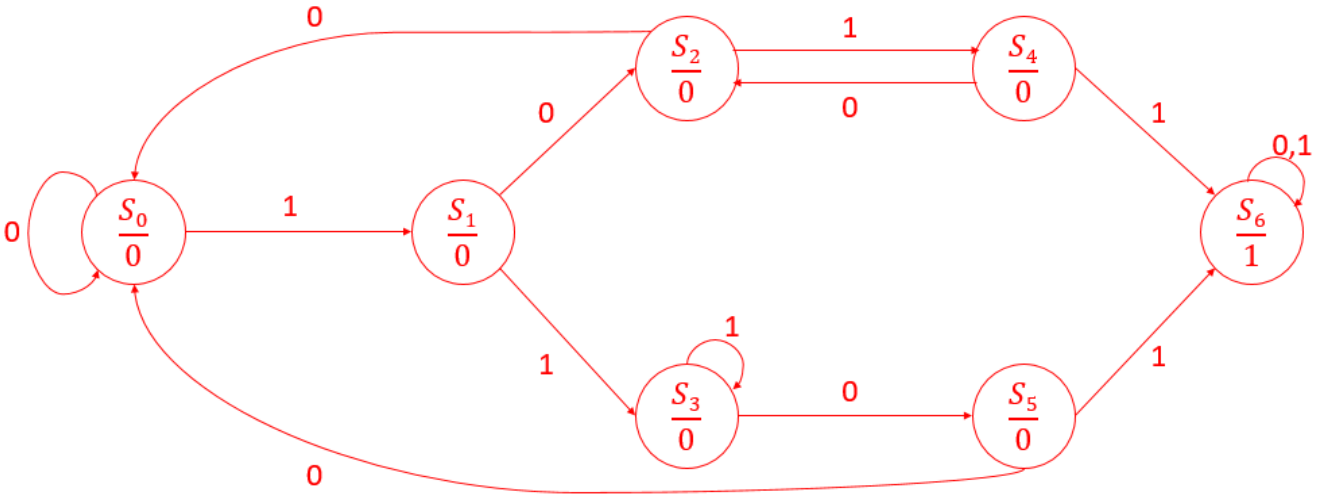
type 신호가 1이 되면 다음 문자의 첫 번째 bit에 따라 root의 두 children 중 하나에서 search를 다시 시작한다. 하지만 첫 번째 피드백이 이루어지기 전까지는 type의 값을 모르기 때문에 ftype 신호를 사용하여 reset 이후 첫 번째 valid input이 입력될 때 머신이 root에서 시작하도록 설정한다.

Problem Set 4 Answers –
FSM design and counter design

Problem 1.1

Input의 X(1-bit)와 Output Z(1-bit)가 존재한다. Input의 순서가 1011 혹은 1101경우일 때를 포함 그 이후의 Output은 1을 출력한다. 처음 State는 S_0에서 시작한다.

- 위의 조건을 만족하는 State transition diagram을 Moore Machine으로 나타내고, State transition table을 작성하시오. (단, state의 개수가 최소가 되도록 할 것)



Problem 1.1

Input의 X(1-bit)와 Output Z(1-bit)가 존재한다. Input의 순서가 1011 혹은 1101경우일 때를 포함 그 이후의 Output은 1을 출력한다. 처음 State는 S_0에서 시작한다.

- 위의 조건을 만족하는 State transition diagram을 Moore Machine으로 나타내고, State transition table을 작성하시오. (단, state의 개수가 최소가 되도록 할 것)

Current State	Next State		Output(Z)
	X = 0	X = 1	
S_0	S_0	S_1	0
S_1	S_2	S_3	0
S_2	S_0	S_4	0
S_3	S_5	S_3	0
S_4	S_2	S_6	0
S_5	S_0	S_6	0
S_6	S_6	S_6	1

Problem 1.2

- 1.1에서 작성한 내용을 바탕으로 State를 3개의 D-Flip Flop을 이용하여 implement한다고 할 때, 각 state와 output에 대한 나올 수 있는 간소화된 Sum Of Product(SOP) Boolean 식 중 가장 간소화 된 SOP Boolean 식을 K-map을 이용하여 나타내시오. (단, 가장 간소화된 SOP 식이란 가장 적은 개수의 sum과 가장 적은 개수의 product를 사용한 것)

정답 : 앞의 state transition table 이용

S_0 = 000, S_1 = 001, S_2 = 010, S_3 = 011, S_4 = 101, S_5 = 100, S_6 = 110

S_4 = 100, S_5 = 101로 할 경우 간단한 식이 되지 않음

a^+ == x'bc + ab + xbc' + xa

b^+ == b'c + ab + xc + xa

c^+ == xa', Z = ab

a^+

bc \ xa	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	X	X	0
10	0	1	1	1

b^+

bc \ xa	00	01	11	10
00	0	0	1	0
01	1	1	1	1
11	0	X	X	1
10	0	1	1	0

c^+

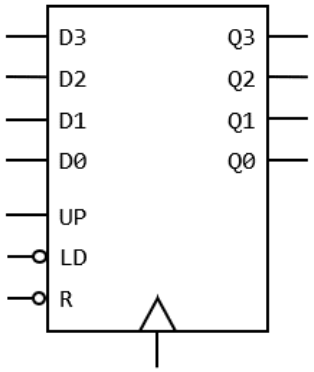
bc \ xa	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	0	X	X	1
10	0	0	0	1

Z

bc \ a	0	1
00	0	0
01	0	0
11	0	X
10	0	1

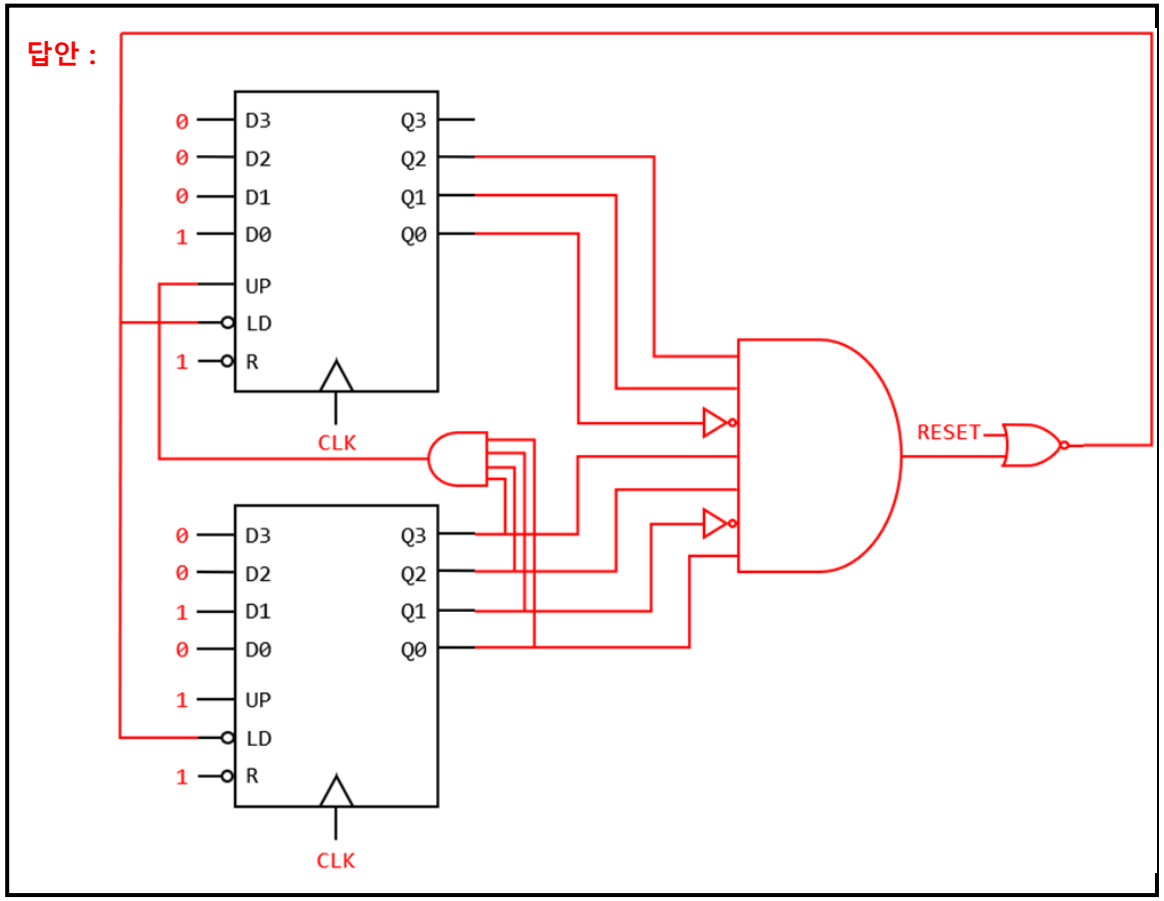
Problem 2

- 아래 주어진 4-bit synchronous up-counter 2개와 최소 개수의 logic gate를 사용하여 0010010에서 1101101까지 count up하고 반복하는 7-bit counter module을 구현하여라. 이때, 외부 RESET 신호가 들어올 때 카운터는 0010010 상태에서 시작한다. (단, LD는 D값을 Q값으로 load, UP은 count up, R은 Q값을 0으로 reset하며 우선순위는 R > LD > UP이다. 사용 가능한 logic gate는 NAND, NOR, AND, OR, XOR, XNOR, INV로 가정하고 input 개수에는 제한이 없다.)



[4-bit up-counter]

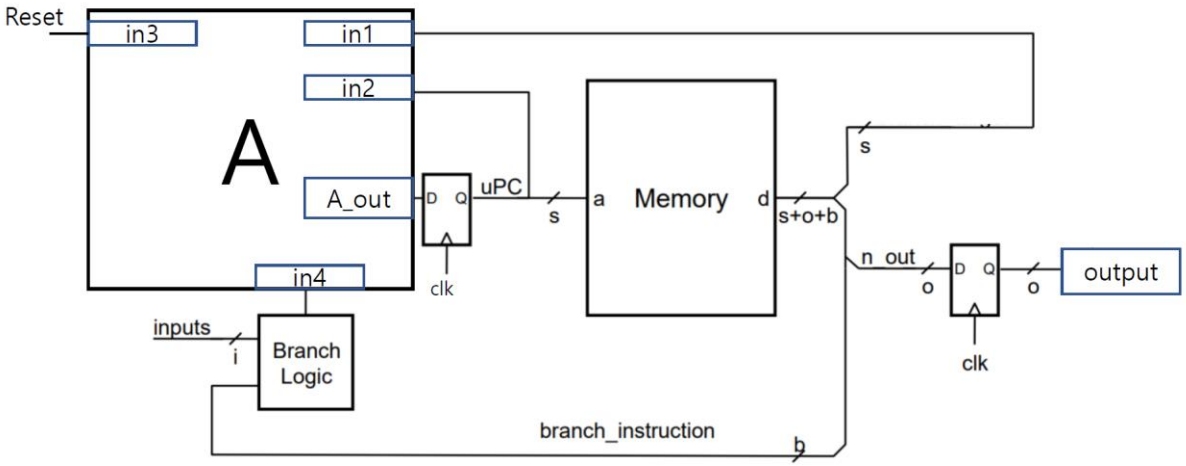
답안 :


0
0
0
1
D3
D2
D1
D0
Q3
Q2
Q1
Q0
UP
LD
1
R
CLK
RESET
0
0
1
0
D3
D2
D1
D0
Q3
Q2
Q1
Q0
1
UP
LD
1
R
CLK

Problem Set 5 Answers – Microcoded FSM

Problem 1

- 아래의 그림은 **Instruction sequencing**을 활용한 microcoded FSM을 구현한 회로이다.



Problem 1.1

- Instruction sequencing이 올바르게 동작할 수 있도록, 주어진 회로의 A 부분을 verilog로 구현하시오.(state의 bit수는 s이며, in1, in2, in3, in4, A_out으로 표현할 것. Reset 신호를 받으면 A_out은 0으로 초기화된다.)

```
정답 :  assign A_out = in3 ? {s{1'b0}} : (in4 ? in1 : in2 + 1'b1);
```

Problem 1.2

- 주어진 회로처럼 Instruction sequencing을 활용하여 microcoded FSM을 구성하였을 때에, memory의 size를 줄일 수 있는 이유가 무엇인가?

정답 : 대부분의 상황에서 sequence를 따라가고, 특별한 input일 때만 가끔 branch하기 때문에, 모든 state와 input의 경우에 대한 정보를 저장할 필요가 없기 때문이다.

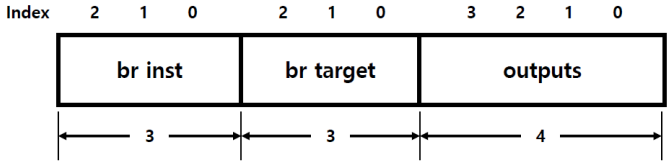
Problem 1.3

- 아래 table은 주어진 회로의 block diagram 속 memory table이다. 조건들을 참고하여 timing diagram 속 (A)~(G)를 올바르게 표현하시오.

Conditions

i. 주어진 회로의 branch logic은 ((brinst[0] & in[0]) | (brinst[1] & in[1])) ^ brinst[2]로 정의되어 있다.

ii. 아래 그림은 memory에 저장되는 micro instruction 구조이다.

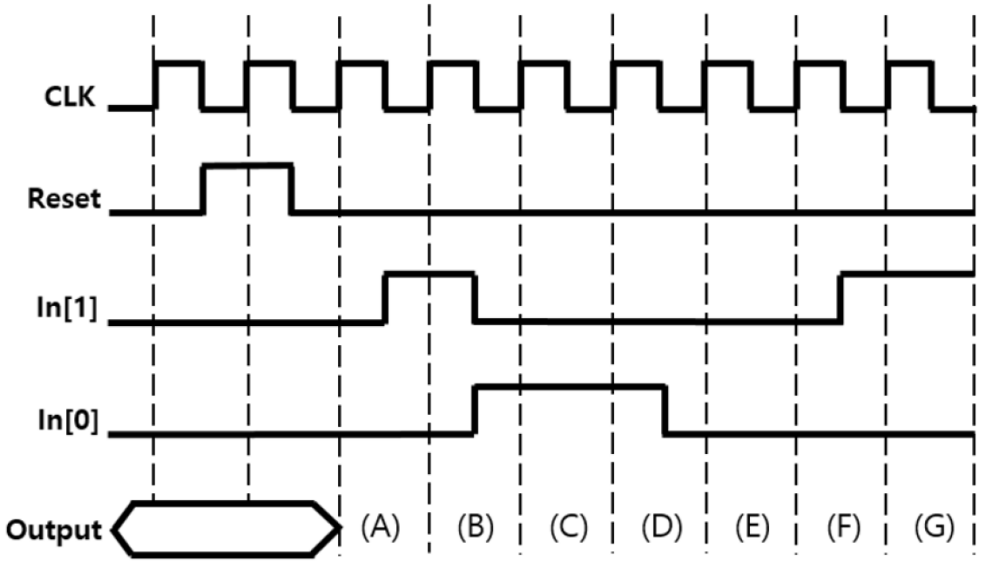


iii. Posedge triggered Flip-Flop을 이용한다.

iv. Output은 binary representation으로 표현된다.

Address	Data
000	1110000100
001	0010100010
010	0100010001
011	1000001000
100	0010100101
101	1000001111

Problem 1.3



addr	input	Br inst	Br target	output	Next addr	Branch
000	-	111	000	0100	-	(reset)
000	00	111	000	0100	000	Branch
000	10	111	000	0100	001	upc + 1
001	01	001	010	0010	010	Branch
010	01	010	001	0001	011	upc + 1
011	00	100	000	1000	000	Branch
000	00	111	000	0100	000	Branch

정답 :

(A) : 4'b0100 - reset

(B) : 4'b0100

(C) : 4'b0100

(D) : 4'b0010

(E) : 4'b0001

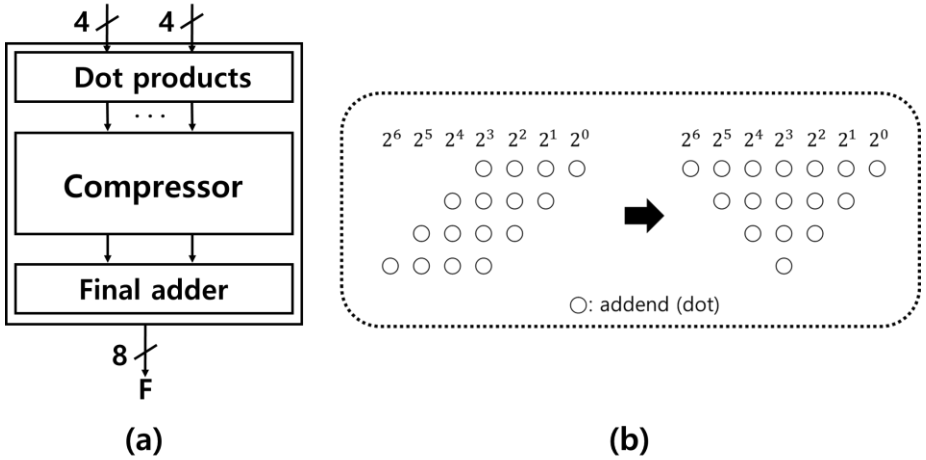
(F) : 4'b1000

(G) : 4'b0100

Problem Set 6 Answers – Parallel multiplier and adder

Problem 1

- 4-bit 입력 A(3:0) = [A3, A2, A1, A0]와 B(3:0) = [B3, B2, B1, B0]를 곱하는 parallel multiplier F(7:0) = A(3:0) * B(3:0)를 아래의 그림과 같이 구현하고자 한다. 아래 그림 (a)는 parallel multiplier의 작동을 그림으로 표현한 것이며 그림 (b)는 dot product의 결과를 addend(dot)로 표현한 것이다.



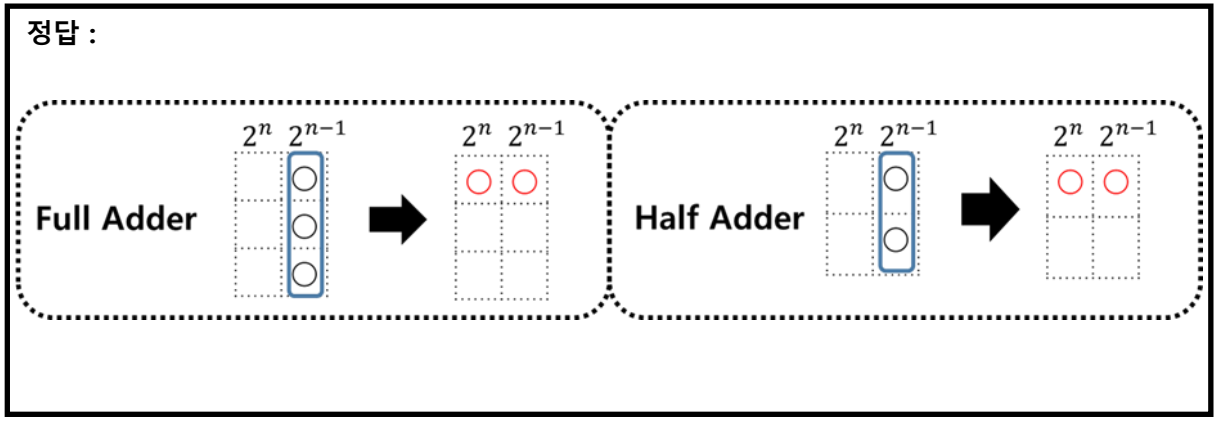
Problem 1.1

- 앞의 그림 (b)에서 compressor의 역할을 설명하여라.

정답 : row를 2개 이하로 만든다

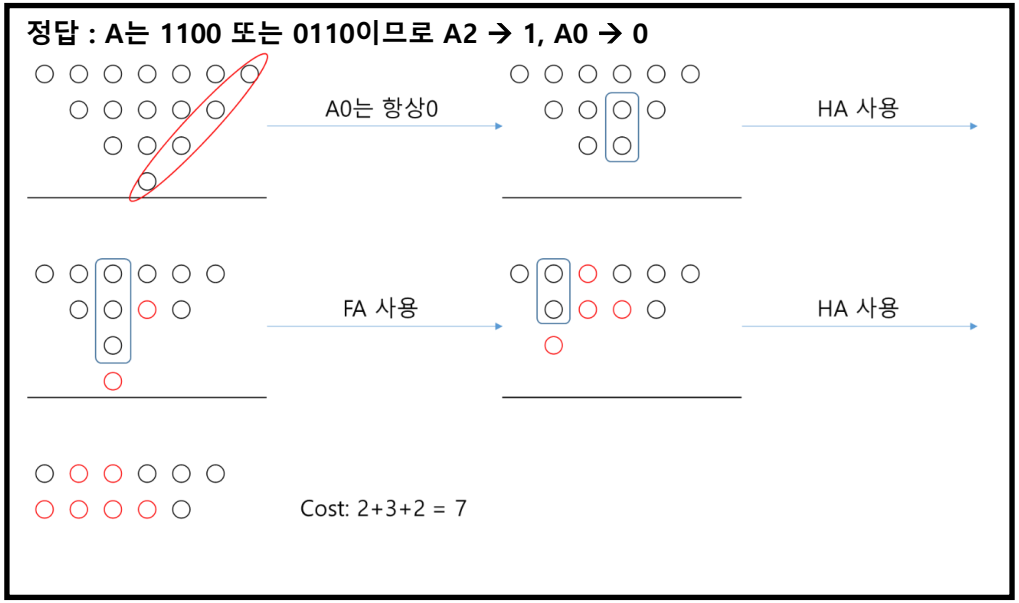
Problem 1.2

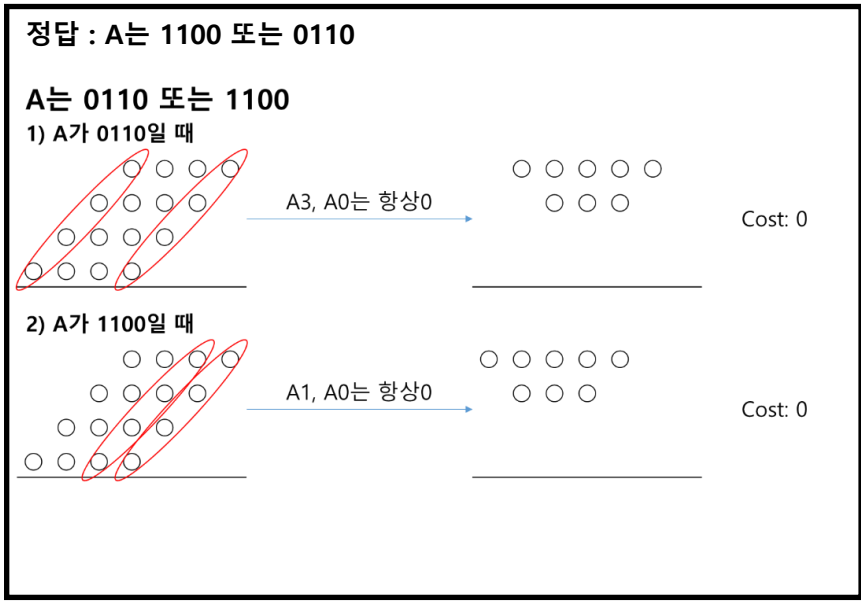
- Compressor에서 Full Adder(FA)와 Half Adder(HA)를 이용할 때의 결과를 그려보아라.



Problem 1.3

- A(3:0)가 0이 아닌 6의 배수일 때 (2)의 FA와 HA를 이용해 compressor를 구현하고자 한다. 이때 FA의 cost는 3, HA의 cost는 2일 때 cost가 최소가 되도록 compressor의 작동 과정을 그림으로 보이고 최소 cost를 구하라. (FA와 HA를 사용할 때 그림에 명시하도록 할 것)





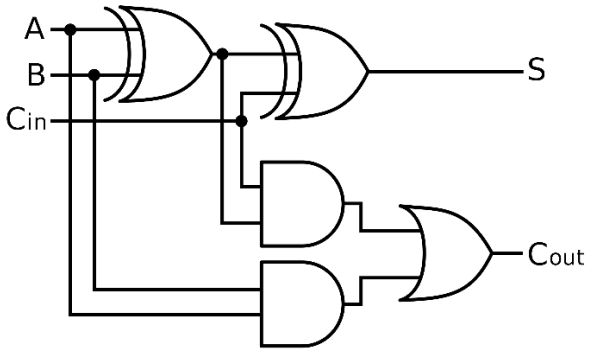
Problem 1.4

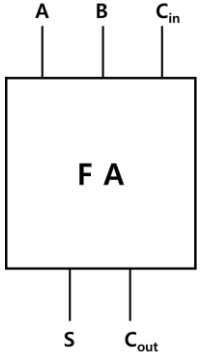
- Compressor를 구현할 때 Carry Save Adder(CSA) 방법을 사용할 때의 **장점**을 설명하여라

정답 : interconnect, wiring, layout이 규칙적이다.

Problem 2.1

- 왼쪽의 logic gate들은 오른쪽과 같은 full adder(FA)를 표현한다. 오른쪽의 **FA만을** 이용하여 12-bit의 ripple carry adder(RCA)를 그려보시오. 아래의 표를 참고하여 해당 adder의 area와, input A[0] 혹은 B[0]부터 마지막 sum bit인 S[11]까지의 delay를 구하시오.

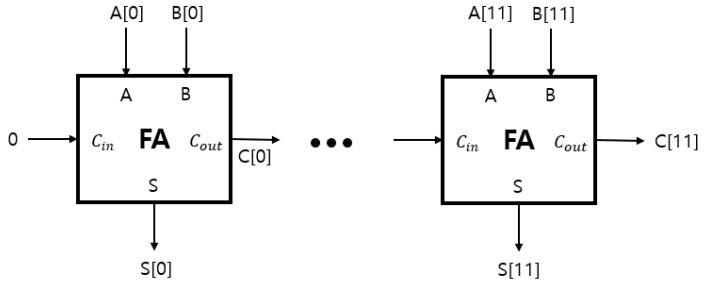




	Area (μm^2)	Delay (ns)
2-input XOR gate	8	60
2-input AND gate	6	30
2-input OR gate	6	40

정답 :

logic gate :

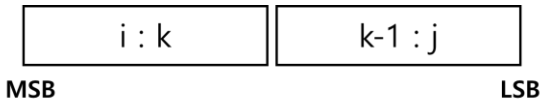


area : (6*3+8*2)*12 = $408(\mu m^2)$

path delay : 130+(70*10)+60 = 890(ns)

Problem 2.2

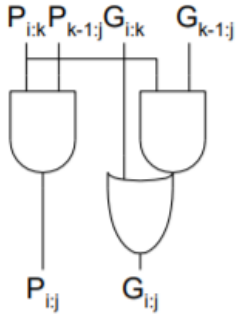
- Parallel prefix adder의 group propagate $P_{i:j}$와 group generate $G_{i:j}$를 $P_{i:k}$, $P_{k-1:j}$, $G_{i:k}$, $G_{k-1:j}$로 표현하고 logic gate로 표현하시오. 이때, **2-input logic gate만을** 이용하여라. ($P_{n:m}$은 n번째에서 m번째의 propagate를 의미한다.)



정답 :

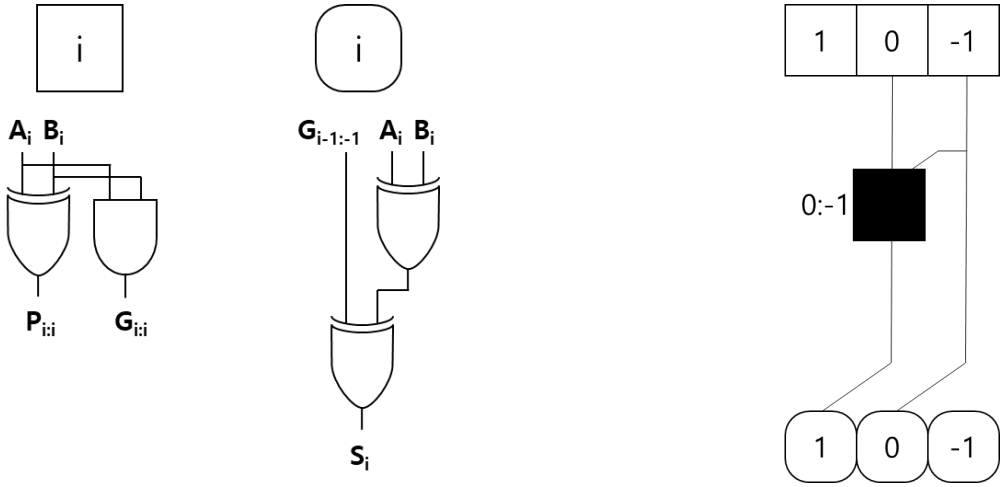
$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j}$$

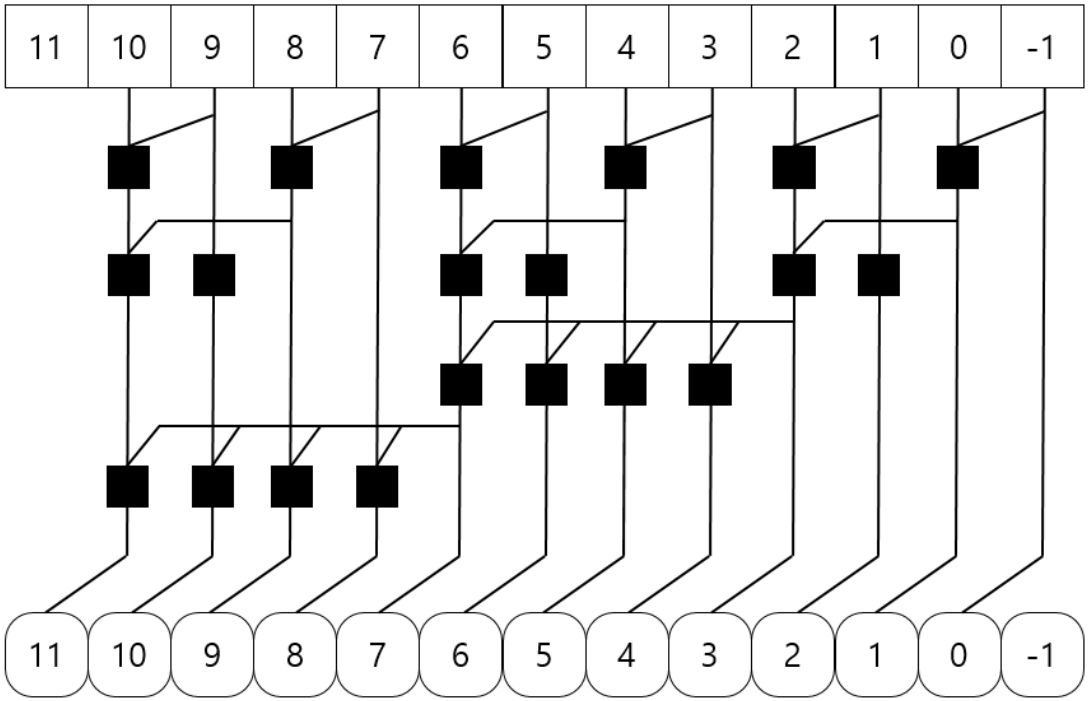
$$P_{i:j} = P_{i:k} \cdot P_{k-1:j}$$



Problem 2.3

- 12-bit parallel prefix adder를 2)에서 구한 logic gate를 이용하여 아래의 tree structure에 표현하여라. 이때 2.2 에서 구한 logic gate는 검은색 상자로 대체하여 그리시오. tree structure의 일부 그림이 표기하는 바는 왼쪽 아래와 같으며, 검은색 상자를 사용하는 예시는 오른쪽 아래와 같다.




11
10
9
8
7
6
5
4
3
2
1
0
-1
11
10
9
8
7
6
5
4
3
2
1
0
-1

Problem 2.4

- 2.3 에서 만든 parallel prefix adder의 area와, input A[0] 혹은 B[0]부터 마지막 sum bit인 S[11]까지의 delay를 아래의 표를 참고하여 구하시오.

	Area (μm^2)	Delay (ns)
2-input XOR gate	8	60
2-input AND gate	6	30
2-input OR gate	6	40

정답 :

area : (14*11)+(18*20)+(16*12)=$706(\mu m^2)$

path delay : 60+70*4+60=400(ns)