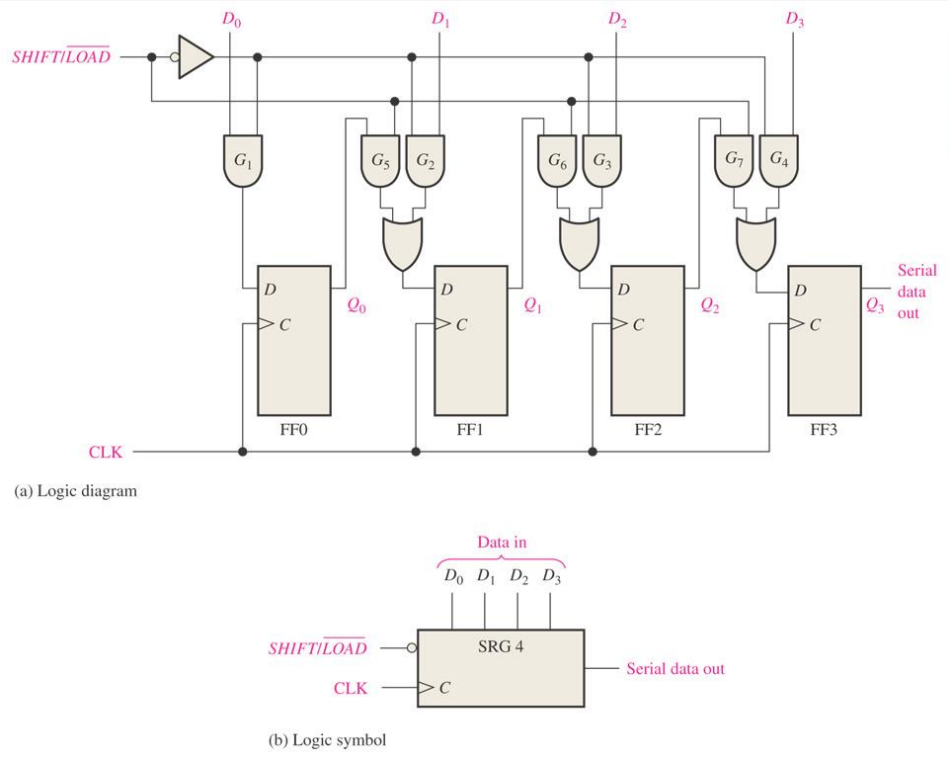
State machine을 이용한 Serial adder 설계

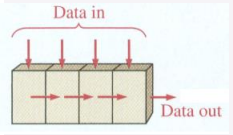
컴퓨터 로직설계

휴먼지능정보공학과
201810776 소재휘

Parallel In/Serial Out Shift Registers

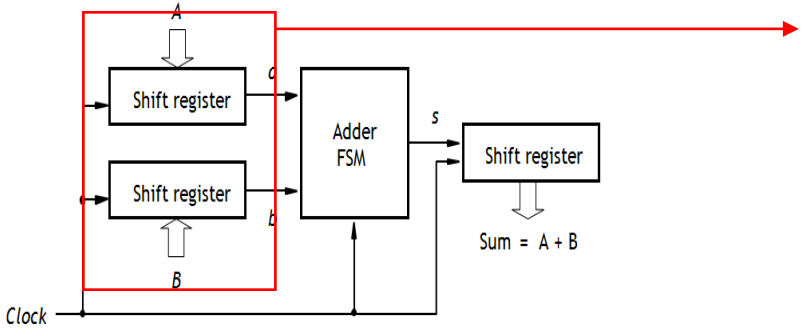
- 4-bit parallel in/serial out



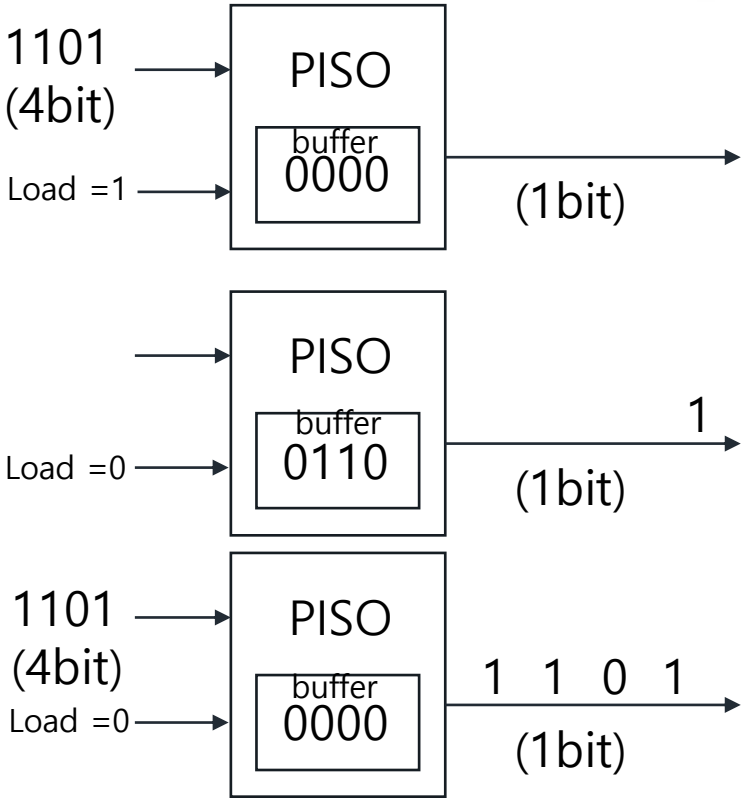


#5-1 PISO

Serial adder design



처음 구현하려는 것은 parallel in-serial out인 PISO입니다. 대략적으로 구현하려는 것을 옆에 그림으로 나타내어 보았습니다. Load가 1일 때 1101이라는 4bit가 동시에 들어와 input된 순서대로 1bit씩 나가게 하는 것을 구현해야 합니다.



#5-1 PISO

Serial adder design

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  ENTITY piso IS
4  PORT (
5  Q : BUFFER std_logic_vector(3 DOWNTO 0);
6  R : IN std_logic_vector (3 DOWNTO 0);
7  reset, L : IN std_logic;
8  CLK : IN std_logic ;
9  Y : OUT std_logic
10 );
11 END piso;
12 ARCHITECTURE BEHAVIOR OF piso IS
13 BEGIN
14    PROCESS(CLK, reset)
15    BEGIN
16    IF (reset = '1') THEN
17       Q <= (OTHERS => '0');
18    ELSIF (CLK'event and CLK='1') THEN
19       IF (L='1') THEN
20       Q <= R;
21       ELSE
22       Q(0)<=Q(1);
23       Q(1)<=Q(2);
24       Q(2)<=Q(3);
25       Q(3)<='0';
26       END IF;
27    END IF;
28       Y<=Q(0);
29 END PROCESS;
30 END BEHAVIOR;
```

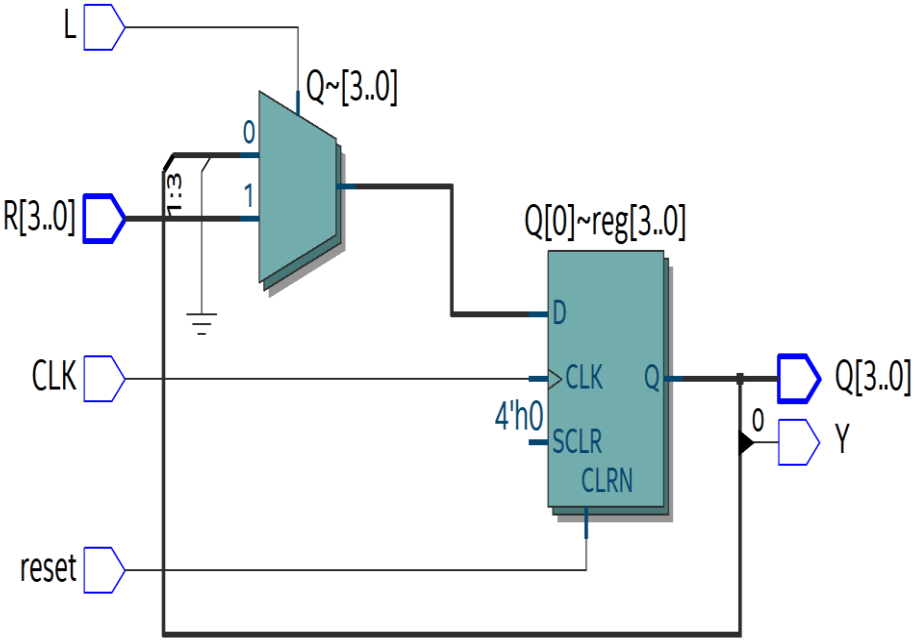
우선적으로 4bit의 input을 동시에 받아서 1bit의 output을 하나씩 내보내는 PISO를 구현하기 위해서 VHDL 코드를 만들었습니다.

Input으로는 4bit의 R, reset과 load, clk를 넣었으며 4개의 buffer와 output 하나를 port로 설정하였습니다.

Reset이 1일 때에는 모든 버퍼에 0값을 넣습니다. 나머지 경우에는 clk의 rising edge에서 Load의 값이 1이라면 input의 4bit값으로 buffer에 load합니다.
Load의 값이 0이라면 buffe의 bit를 한비트씩 뒤로 밀리게 하였습니다. 그리고 마지막 버퍼의 값을 outpu으로 나오게 하였습니다.

#5-1 PISO

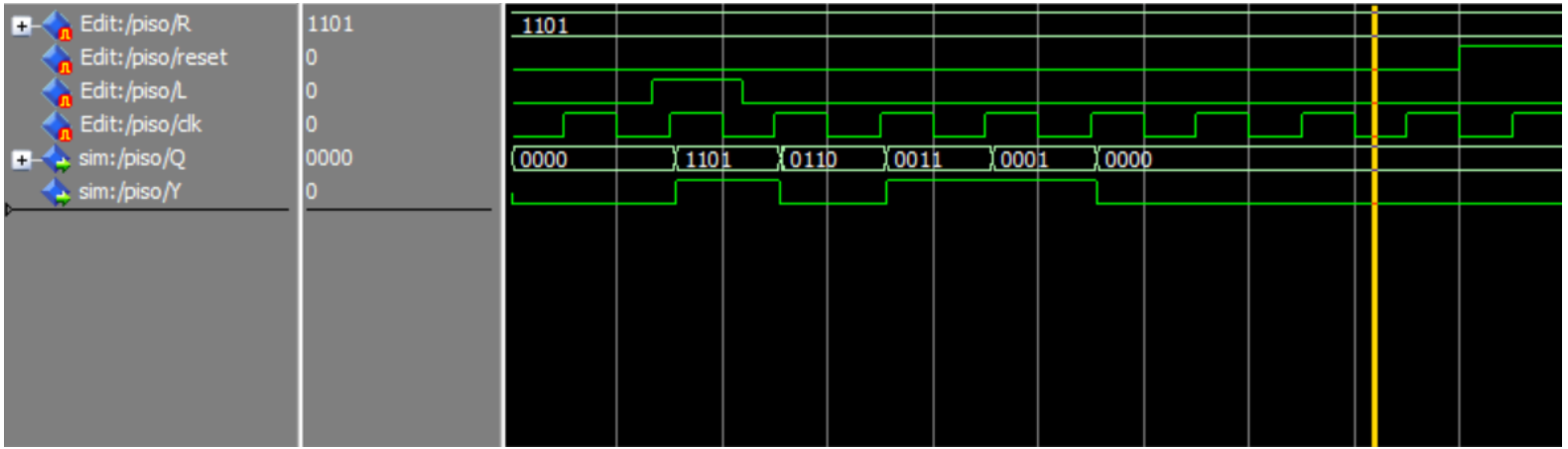
Serial adder design



VHDL 코드로 도출된 예상 회로를
RTL viewer로 확인하였습니다.

#5-1 PISO

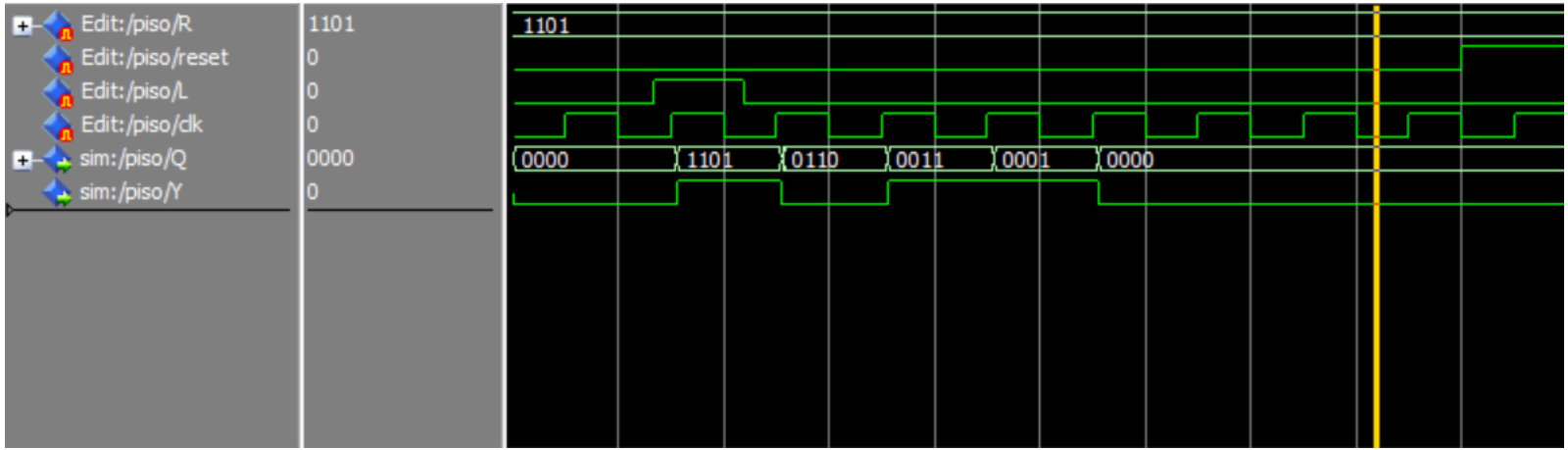
Serial adder design



다음 VHDL 코드를 검증하기 위해서 Simulating을 해보았습니다. Clock의 rising edge에서 Load가 1일 때에는 input으로 들어온 4bit가 그대로 buffer에 들어오는 것을 확인할 수 있습니다. 그리고 Load가 0일 때에는 input이 들어오지 않고 buffer에 있는 비트가 한비트씩 뒤로 밀리는 것을 확인했습니다. 또한 buffer의 마지막 bit가 output으로 도출되는 것을 확인할 수 있습니다. 즉 input이었던 1101이 차례로 1 0 1 1 순으로 ouput의 결과로 나가게 되는것을 확인할 수 있습니다.

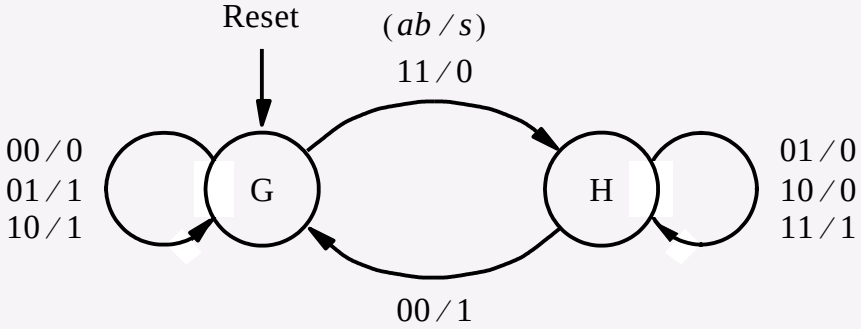
#5-1 PISO

Serial adder design



Reset이 1일 때에는 buffer가 모두 0000의 값이 들어가는 것을 확인하여 reset이 잘 작동함을 확인하였습니다. 이로서 load가 1일 때 들어온 4bit가 결국 output으로 1bit씩 나오게 된다는 것을 알 수 있었습니다.

Mealy-type machine for Serial adder



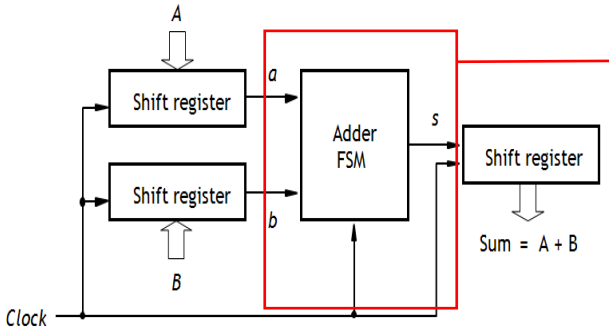
G: carry-in = 0

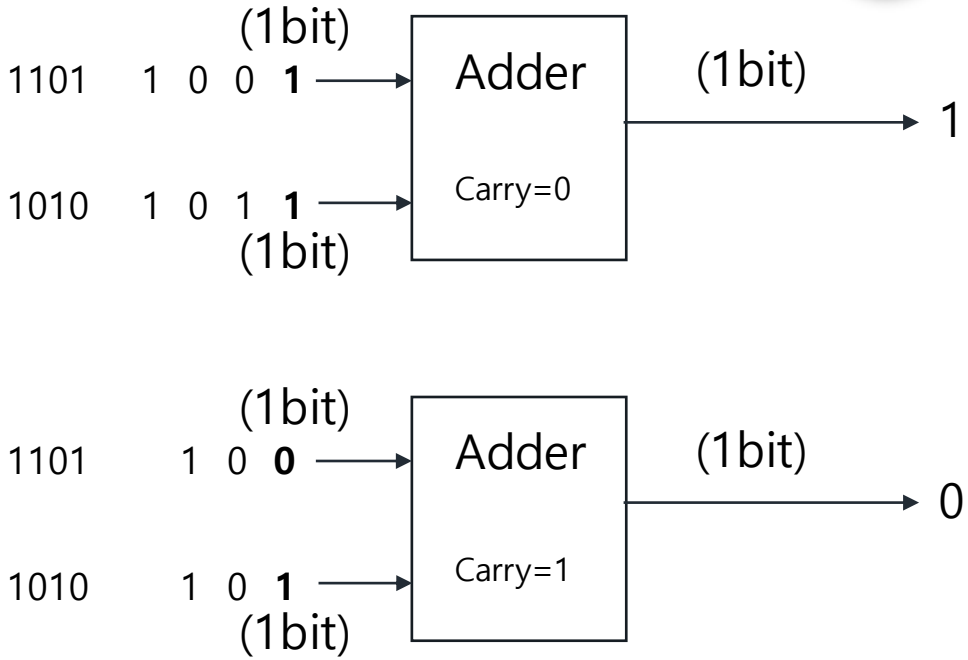
H: carry-in = 1

Present state	Next state				Output *s*			
	ab=00	01	10	11	00	01	10	11
G	G	G	G	H	0	1	1	0
H	G	H	H	H	1	0	0	1

#5-2 Adder

Serial adder design

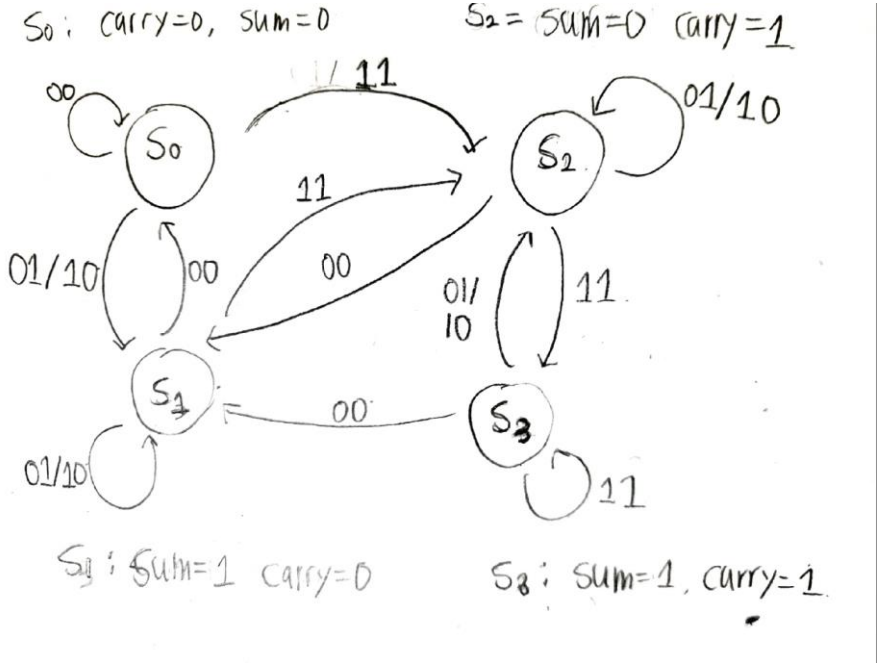




다음으로 구현할 것은 adder입니다. Adder은 PISO의 output에서 나온 결과인 1bit의 input 2개를 받아 1bit의 output을 내보냅니다. Adder은 1001+1011=10100과 같은 비트 연산을 1bit 단위로 차례로 연산합니다. 이 때 carry를 통하여 연산의 올림을 가능하도록 구현합니다.

#5-2 Adder

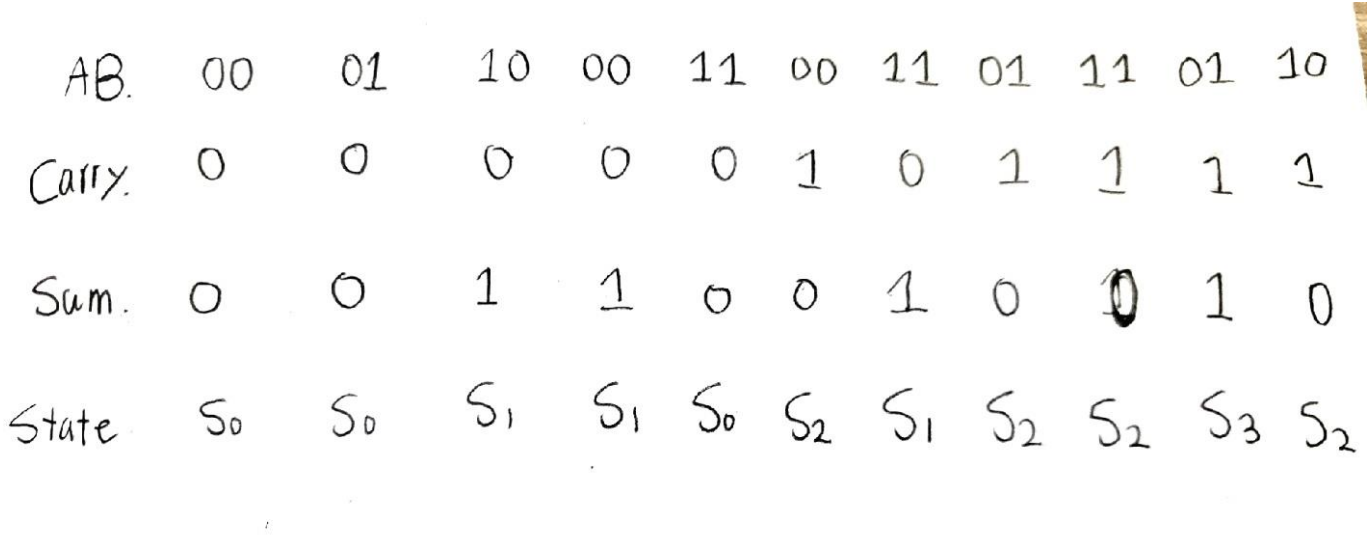
Serial adder design



Adder의 연산 구현하기 위해서 Moore Machine으로 Adder의 state diagram을 구현하였습니다. 올림을 할 것인지의 여부인 Carry와 연산 결과인 Sum의 유무에 따라서 4개의 State로의 상태 전환이 일어납니다.

#5-2 Adder

Serial adder design



AB	00	01	10	00	11	00	11	01	11	01	10
Carry	0	0	0	0	0	1	0	1	1	1	1
Sum	0	0	1	1	0	0	1	0	0	1	0
State	S_0	S_0	S_1	S_1	S_0	S_2	S_1	S_2	S_2	S_3	S_2

State diagram을 바탕으로 예상되는 Timing diagram을 도출해 보았습니다.
이를 바탕으로 시뮬레이팅을 진행할 예정입니다.

#5-2 Adder

Serial adder design

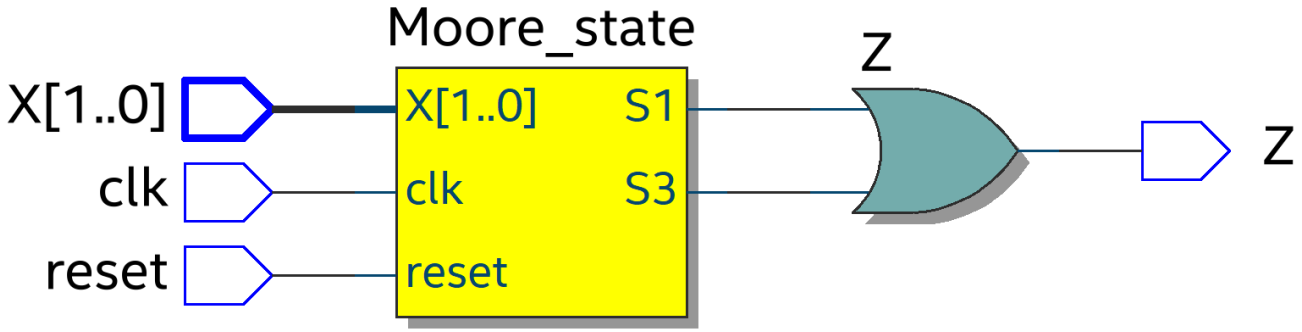
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY adder IS
5     PORT (clk, reset  : IN  STD_LOGIC ;
6           X     : IN  STD_LOGIC_VECTOR(1 DOWNTO 0) ;
7           Z       : OUT    STD_LOGIC ) ;
8  END adder ;
9
10 ARCHITECTURE Behavior OF adder IS
11    TYPE State IS (S0, S1, S2, S3) ;
12    SIGNAL Moore_state : State ;
13 BEGIN
14    PROCESS ( reset, clk )
15    BEGIN
16       IF reset = '1' THEN Moore_state <= S0 ;
17       ELSIF (clk'EVENT AND clk = '1') THEN
18          CASE Moore_state IS
19             WHEN S0 =>
20                IF X(0) = '0' AND X(1) = '0' THEN Moore_state <=S0 ;
21                ELSIF X(0) = '0' AND X(1) = '1' THEN Moore_state <=S1 ;
22                ELSIF X(0) = '1' AND X(1) = '0' THEN Moore_state <=S1 ;
23                ELSE Moore_state <=S2 ;
24
25                END IF ;
26
```

```
26             WHEN S1 =>
27                IF X(0) = '0' AND X(1) = '0' THEN Moore_state <=S0 ;
28                ELSIF X(0) = '0' AND X(1) = '1' THEN Moore_state <=S1 ;
29                ELSIF X(0) = '1' AND X(1) = '0' THEN Moore_state <=S1 ;
30                ELSE Moore_state <=S2 ;
31
32                END IF ;
33
34             WHEN S2 =>
35                IF X(0) = '0' AND X(1) = '0' THEN Moore_state <=S1 ;
36                ELSIF X(0) = '0' AND X(1) = '1' THEN Moore_state <=S2 ;
37                ELSIF X(0) = '1' AND X(1) = '0' THEN Moore_state <=S2 ;
38                ELSE Moore_state <=S3 ;
39
40                END IF ;
41
42             WHEN S3 =>
43                IF X(0) = '0' AND X(1) = '0' THEN Moore_state <=S1 ;
44                ELSIF X(0) = '0' AND X(1) = '1' THEN Moore_state <=S2 ;
45                ELSIF X(0) = '1' AND X(1) = '0' THEN Moore_state <=S2 ;
46                ELSE Moore_state <=S3 ;
47
48                END IF ;
49          END CASE ;
50       END IF ;
51    END PROCESS ;
52    Z <= '1' WHEN Moore_state=S3 OR Moore_state=S1 ELSE '0' ;
53
54 END Behavior ;
```

State diagram을 바탕으로 adder를 vhdl코드로 구현하였습니다. Sum이 1인 state일 시에는 output을 1로 나오게 합니다. Clock의 rising edge에서 상태의 전환이 일어납니다.

#5-3 SIPO

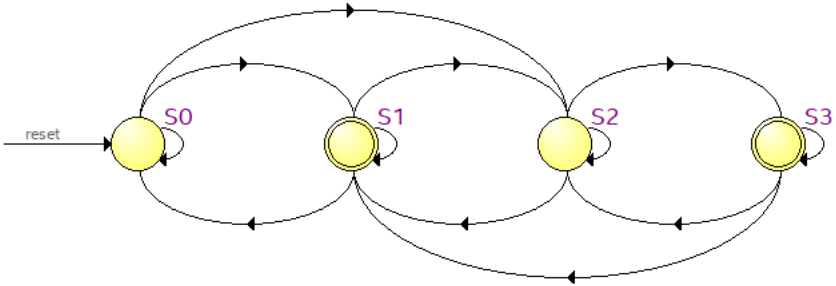
Serial adder design



RTL viewer을 통하여 예상 회로를 확인할 수 있었습니다.

#5-2 Adder

Serial adder design

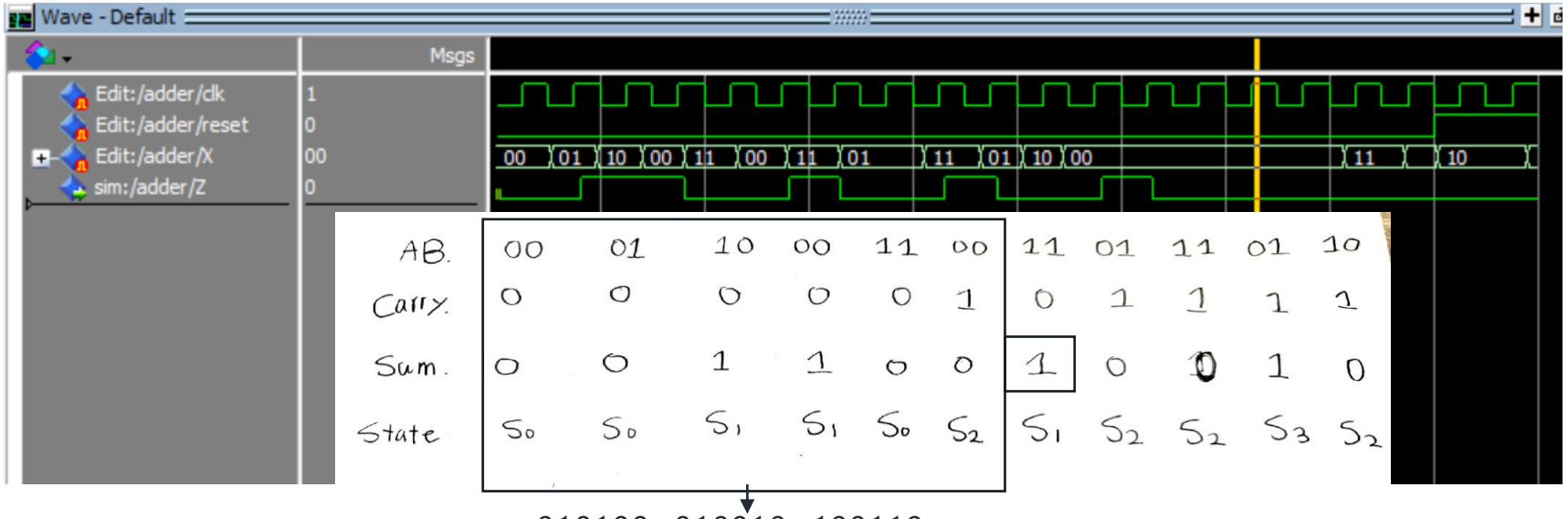


	Source State	Destination State	Condition
1	S0	S2	(X[0]).(X[1])
2	S0	S0	(!X[0]).(!X[1])
3	S0	S1	(!X[0]).(X[1]) + (X[0]).(!X[1])
4	S1	S2	(X[0]).(X[1])
5	S1	S0	(!X[0]).(!X[1])
6	S1	S1	(!X[0]).(X[1]) + (X[0]).(!X[1])
7	S2	S2	(!X[0]).(X[1]) + (X[0]).(!X[1])
8	S2	S3	(X[0]).(X[1])
9	S2	S1	(!X[0]).(!X[1])
10	S3	S2	(!X[0]).(X[1]) + (X[0]).(!X[1])
11	S3	S3	(X[0]).(X[1])
12	S3	S1	(!X[0]).(!X[1])

RTL viewer을 통하여 State diagram을 확인할 수 있었습니다.

#5-2 Adder

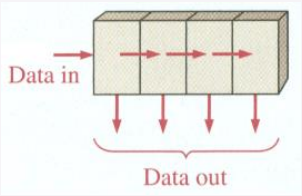
Serial adder design

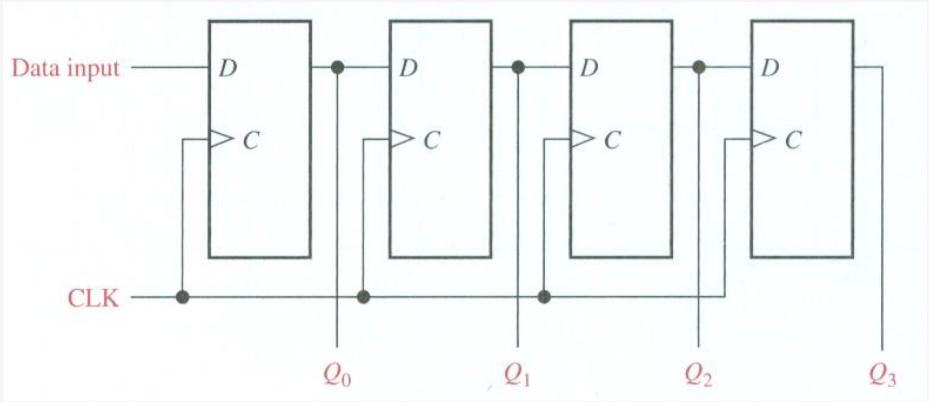


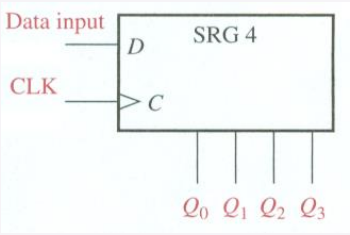
시뮬레이팅 결과 clock의 rising edge에서 state의 변화가 일어나게 됩니다. 이는 제가 예상한 Timing diagram와 일치함을 확인하였습니다.
State가 S1과 S3일 시, 즉 Sum이 1일 때 output이 1이 됩니다.

Serial In/Parallel Out Shift Registers
강의노트 VHDL 8-39

- 4-bit serial in/parallel out

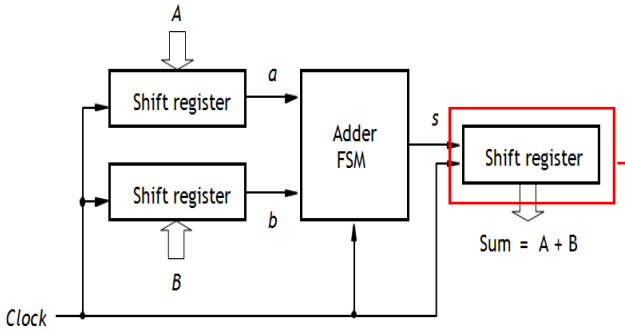




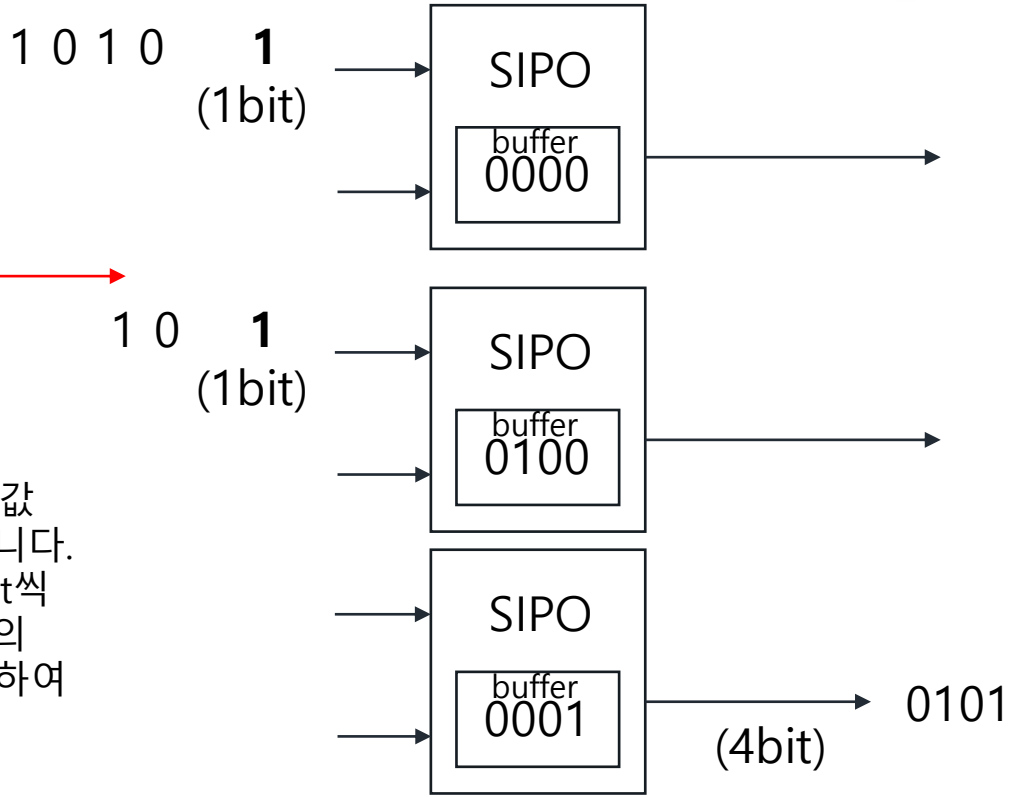


#5-3 SIPO

Serial adder design



Adder을 통해서 PISO에서 나온 1bit의 값 두개를 연산하여 1bit씩 output 시켰습니다. SIPO에서는 Adder의 1bit씩의 값을 4bit씩 묶어 output 시키려고 합니다. 즉 PISO의 input이었던 4bit 단위들을 Adder을 통하여 연산하여 SIPO를 통하여 4bit 단위로 output 시키려는 목적입니다.



#5-3 SIPO

Serial adder design

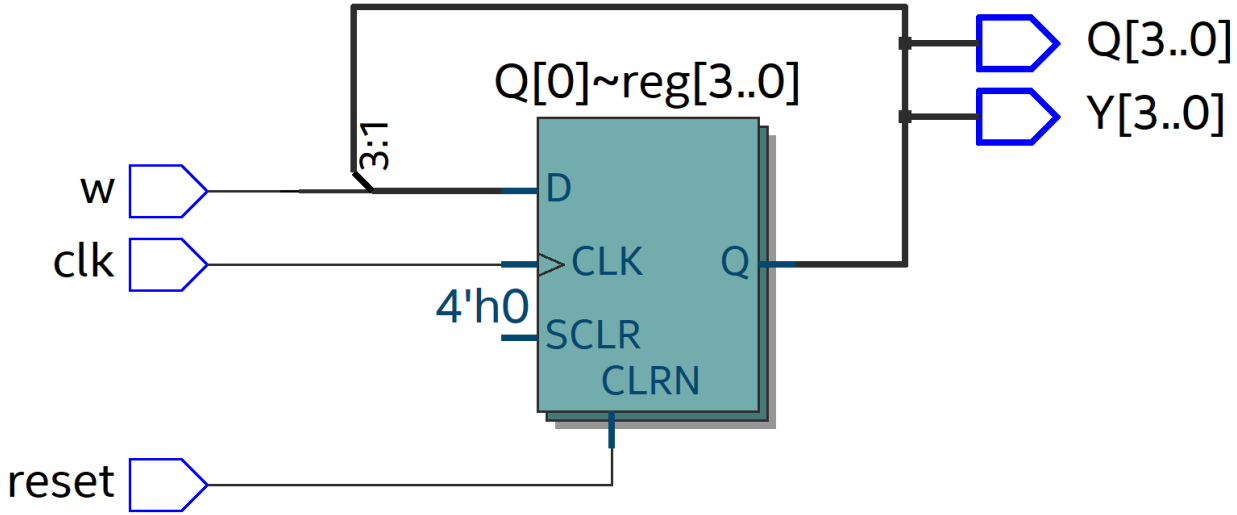
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  ENTITY sipo IS
4  PORT (
5  Q : BUFFER std_logic_vector(3 DOWNTO 0);
6  reset, w : IN std_logic;
7  clk : IN std_logic ;
8  Y : OUT std_logic_vector(3 DOWNTO 0)
9  );
10 END sipo;
11
12 ARCHITECTURE BEHAVIOR OF sipo IS
13 BEGIN
14     PROCESS(CLK, reset)
15     BEGIN
16     IF (reset = '1') THEN
17        Q <= (OTHERS => '0');
18     ELSIF (CLK'event and CLK='1') THEN
19        Q(0)<=Q(1);
20        Q(1)<=Q(2);
21        Q(2)<=Q(3);
22        Q(3)<=w;
23        END IF;
24        Y(0)<=Q(0);
25        Y(1)<=Q(1);
26        Y(2)<=Q(2);
27        Y(3)<=Q(3);
28 END PROCESS;
29 END BEHAVIOR;
30
```

Adder을 통해서 PISO에서 나온 1bit의 값 두개를 연산하여 1bit씩 output 시켰습니다. SIPO에서는 Adder의 1bit씩의 값을 4bit씩 묶어 output 시키려고 합니다. 즉 PISO의 input이었던 4bit 단위들을 Adder을 통하여 연산하여 SIPO를 통하여 4bit 단위로 output 시키려는 목적입니다.

Reset이 1이 아니라면 input w의 값을 넣으며 buffer에 저장된 값을 한 칸씩 밀리게 하여 차례로 4bit를 만들게 됩니다.

#5-3 SIPO

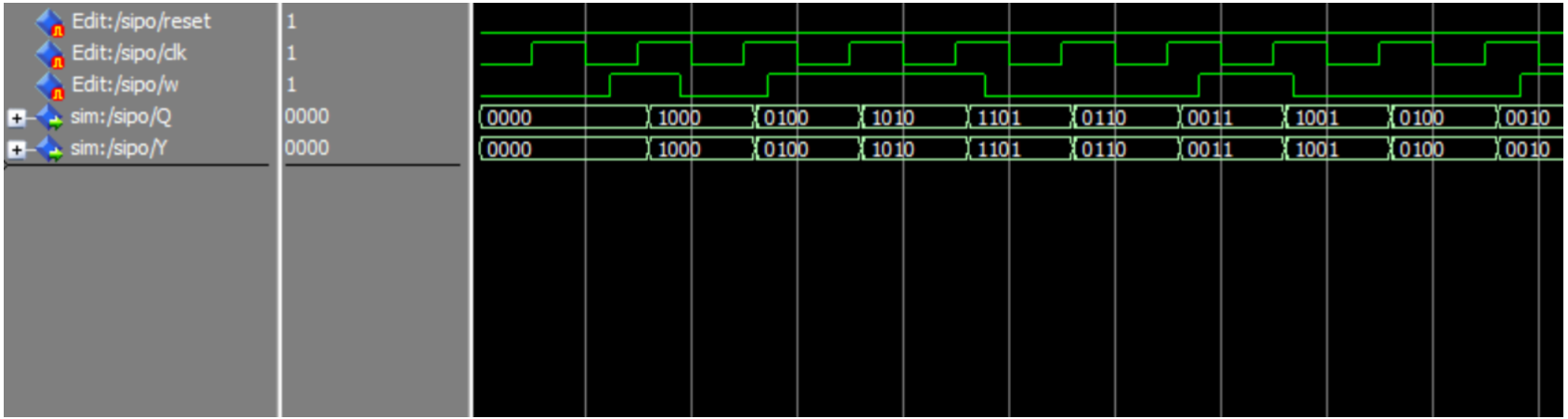
Serial adder design



RTL viewer을 통하여 예상 회로를 확인할 수 있었습니다.

#5-3 SIPO

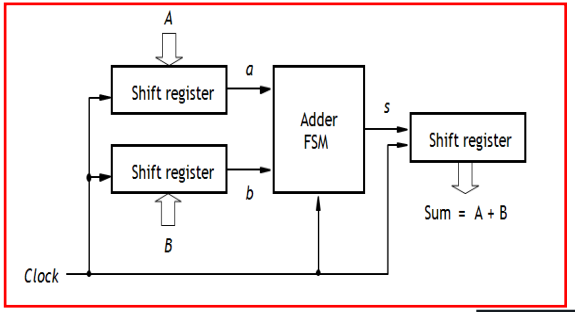
Serial adder design



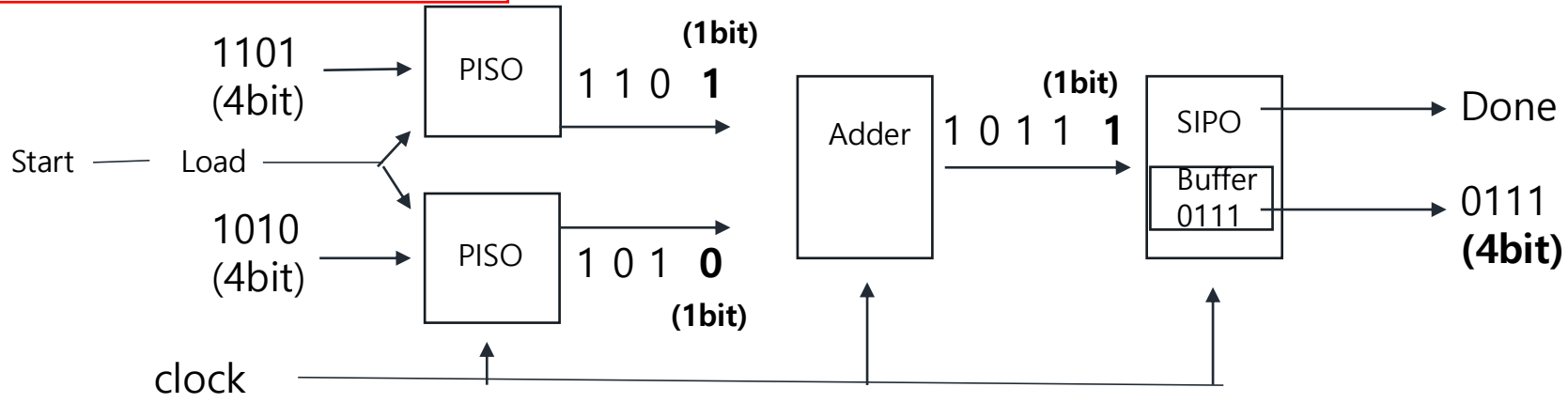
Clock의 rising edge에서 W의 값이 4bit의 첫번째 자리에 들어오게 되며 이에 따라서 이미 버퍼에 저장되어 있던 값이 한 칸씩 밀리게 됩니다. 이로서 4bit 단위의 output이 일어나게 되는 것을 검증하였습니다.

#5-4 Serial Adder

Serial adder design

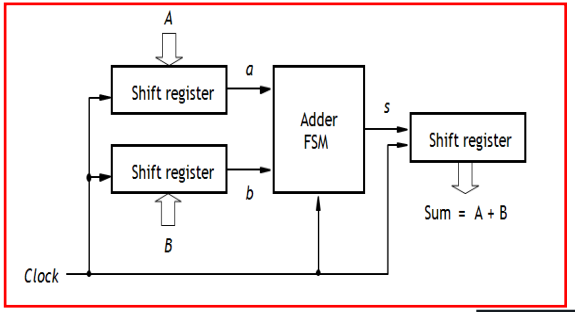


마지막으로 앞에서 구현한 PISO , Adder, SIPO를 이용하여 4bit 단위의 이진 연산을 가능하게 하는 Serial Adder을 구현할 것입니다. Start가 처음 1이 되는 순간에만 PISO에서 4bit 단위의 이진수를 load 받아 Adder를 통해 연산 후 SIPO를 통하여 4bit단위로 묶어 output 시킵니다. 그리고 4bit 단위의 연산이 되었을 때 Done을 1로 output시킵니다.

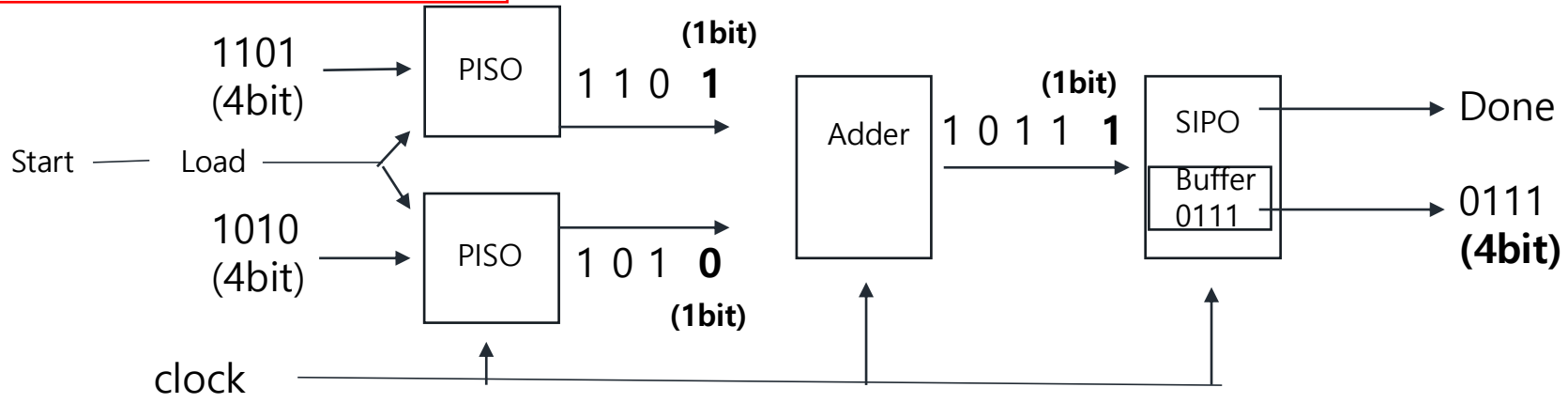


#5-4 Serial Adder

Serial adder design



PISO와 Adder, 그리고 SIPO의 작동은 clock의 rising edge에서 일어났습니다. 따라서 SIPO의 버퍼에 첫 input이 들어가는 시점은 최소 3clock 뒤입니다. 따라서 4bit의 load 후 SIPO에서 연산이 완료된 4bit가 나오게 되는 시점은 7clock 뒤입니다.



#5-4 Serial Adder

Serial adder design

Sipo, adder, piso에서 output이 제대로 나오는지 확인하려고 만든 output. 무시해도 됩니다.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY serialadder IS
PORT (
A, B : IN std_logic_vector(3 DOWNTO 0);
sreset, sclk, Start : IN std_logic;
Done : OUT std_logic;
Sum : OUT std_logic_vector(3 DOWNTO 0);
pis1out : OUT std_logic;
pis2out : OUT std_logic;
adderout : OUT std_logic;
sipout : OUT std_logic_vector(3 DOWNTO 0)
);

END serialadder;

ARCHITECTURE BEHAVIOR OF serialadder IS
   TYPE State IS (S0, S1, S2, S3, S4, S5, S6) ;
   SIGNAL Mealy_state : State ;
```

```
SIGNAL  p1 : STD_LOGIC ;
SIGNAL  p2 : STD_LOGIC ;
SIGNAL  ad : STD_LOGIC ;
SIGNAL  sip : STD_LOGIC ;
SIGNAL  outsip : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
SIGNAL Lbutton : STD_LOGIC ;
SIGNAL Sbutton : STD_LOGIC ;

COMPONENT piso
   PORT (
         Q : BUFFER std_logic_vector(3 DOWNTO 0);
         R : IN std_logic_vector (3 DOWNTO 0);
         reset, L : IN std_logic;
         clk : IN std_logic ;
         Y : OUT std_logic
        );
END COMPONENT ;

COMPONENT adder
   PORT (clk, reset  : IN  STD_LOGIC ;
         X     : IN  STD_LOGIC_VECTOR(1 DOWNTO 0) ;
         Z        : OUT    STD_LOGIC ) ;
END COMPONENT ;
```

#5-4 Serial Adder

Serial adder design

```
COMPONENT sipo
   PORT (
         Q : BUFFER std_logic_vector(3 DOWNTO 0);
         reset, w : IN std_logic;
         clk : IN std_logic ;
         Y : OUT std_logic_vector(3 DOWNTO 0)
         );
END COMPONENT ;

BEGIN

pis1out<=p1;
pis2out<=p2;
adderout<=ad;
sipout<=outsip;



   PROCESS(Start, sclk, sreset)
   BEGIN

   IF (sreset = '1') THEN
   Sum(0)<='0';
   Sum(1)<='0';
   Sum(2)<='0';
   Sum(3)<='0';
```

```
ELSE
Sum<=outsip;
END IF;

IF (Start='1' AND Sbutton='0') THEN
   Lbutton<='1';
ELSE
   Lbutton<='0';
END IF;



IF (sclk'event and sclk='1') THEN
   CASE Mealy_state IS
   WHEN S0 =>
      If(Start='1') THEN
      Sbutton<='1';
      Mealy_state <=S1;
      ELSE
      Sbutton<='0';
      END IF;

   WHEN S1 =>
```

#5-4 Serial Adder

Serial adder design

PISO와 Adder, 그리고 SIPO를 병합하기 위해서 port를 연결하였으며 start를 구현하기 위해서 Start가 처음 들어갔을 때를 기점으로 State를 변하게 함으로서 처음 Start에만 Load가 반응하도록 구현하였습니다.

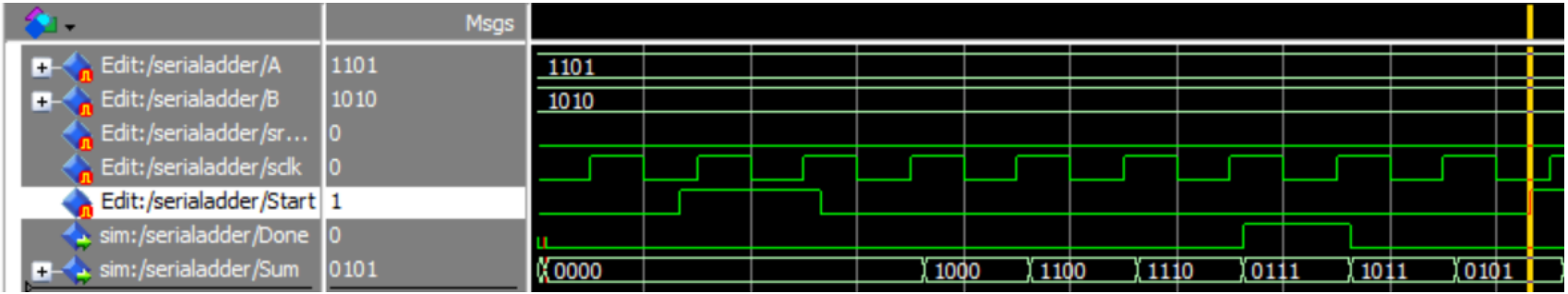
그리고 State를 clock의 rising edge마다 변화시켜 7번째 rising edge에서 Done이 나오도록 하였습니다.

```
   WHEN S1 =>

      Mealy_state <= S2;
   WHEN S2 =>

      Mealy_state <= S3;
   WHEN S3 =>

      Mealy_state <= S4;
   WHEN S4 =>

      Mealy_state <= S5;
   WHEN S5 =>

      Mealy_state <= S6;
   WHEN S6 =>
      Sbutton<='0';
      Mealy_state <= S0;



   END CASE ;
END IF;
```

```
END PROCESS;
   Done <= '1' WHEN Mealy_state=S6 AND sreset='0' ELSE '0';


   q1 : piso PORT MAP (R(0) => A(0) ,R(1) => A(1) ,R(2) => A(2) ,R(3) => A(3) ,
                       reset => sreset, L => Lbutton, clk => sclk, Y => p1);

   q2 : piso PORT MAP (R(0) => B(0) ,R(1) => B(1) ,R(2) => B(2) ,R(3) => B(3) ,
                       reset => sreset, L => Lbutton, clk => sclk, Y => p2);

   q3 : adder PORT MAP (X(1) => p1, X(0) => p2 , reset => sreset, clk => sclk, Z => ad);

   q4 : sipo PORT MAP (w => ad, reset => sreset, clk => sclk,
                       Q(0) => outsip(0), Q(1) => outsip(1), Q(2) => outsip(2), Q(3) => outsip(3
END BEHAVIOR;
```

#5-4 Serial Adder

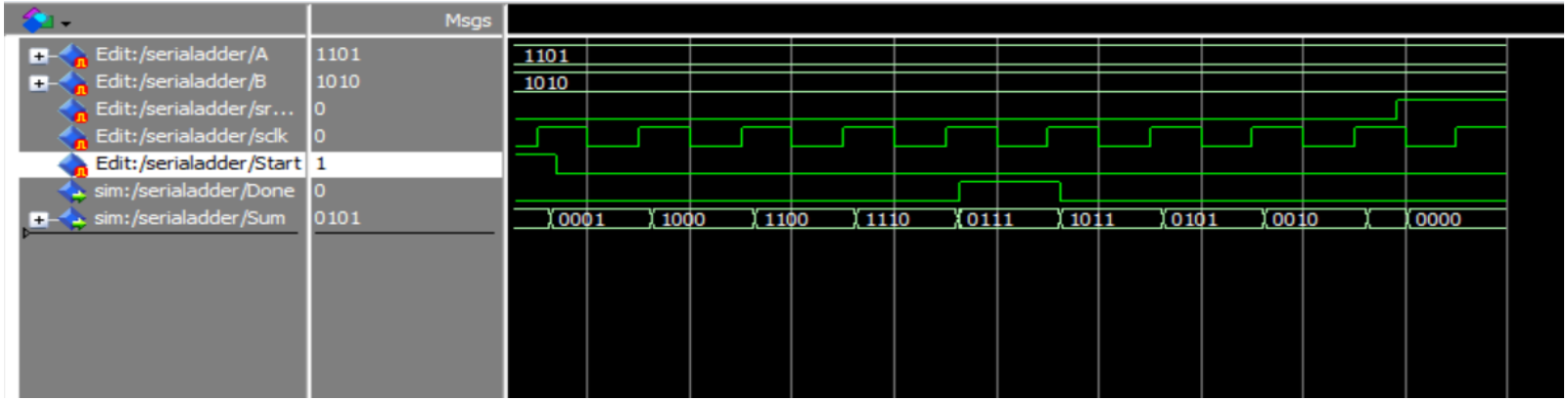
Serial adder design



Start가 두 clock 이상 지속되어도 첫 Start에서만 4bit가 두개가 load되어 PISO, Adder, SIPO를 거쳐 세 clock부터 연산된 결과가 한 비트씩 Sum으로 들어오기 시작한다. 그리고 4bit가 연산이 되었을 때 Done이 1이 되는 것을 확인할 수 있었다. 그 결과 1101+1010은 (1)0111이 되는 것을 알 수 있으며 1은 carry가 1인 상태로 남아 뒤의 결과로 들어오는 것을 알 수 있다.

#5-4 Serial Adder

Serial adder design



두번째 start에서도 앞의 결과와 마찬가지로 나오며 reset이 1이 될 때에는 Sum이 0000으로 고정되는 것 또한 확인할 수 있었다.

#5-5 Discussion

Serial adder design

이번 실습은 Serial adder을 구현하는 실습을 하였습니다. 이번 내용은 전에 배웠던 내용의 총 망라라고 볼 수 있었습니다. PISO, Adder, SIPO를 만들고 이를 병합하는 과정까지 많은 시간이 소요되었습니다.
이 실습을 통하여 알 수 있는 사실이 몇가지 있었습니다.

1. 4bit의 비트 연산이 어떠한 회로를 통해서 이루어지는지 확인할 수 있었습니다.
2. RTL의 코드들을 컴포넌트 하는 방법을 알 수 있었습니다.
3. PISO와 SIPO의 차이점을 알 수 있었습니다.
4. 이 과정은 모두 clock의 rising edge에서 일어난다는 사실을 확인할 수 있었습니다.

이번 실습을 통해서 Serial Adder을 실제로 구현하는 방법을 알 수 있었으며 논리 회로를 통해서 bit연산을 구현하는 것이 공학에서 쓰이는 회로를 구현한 것 같아서 의미가 있었습니다.