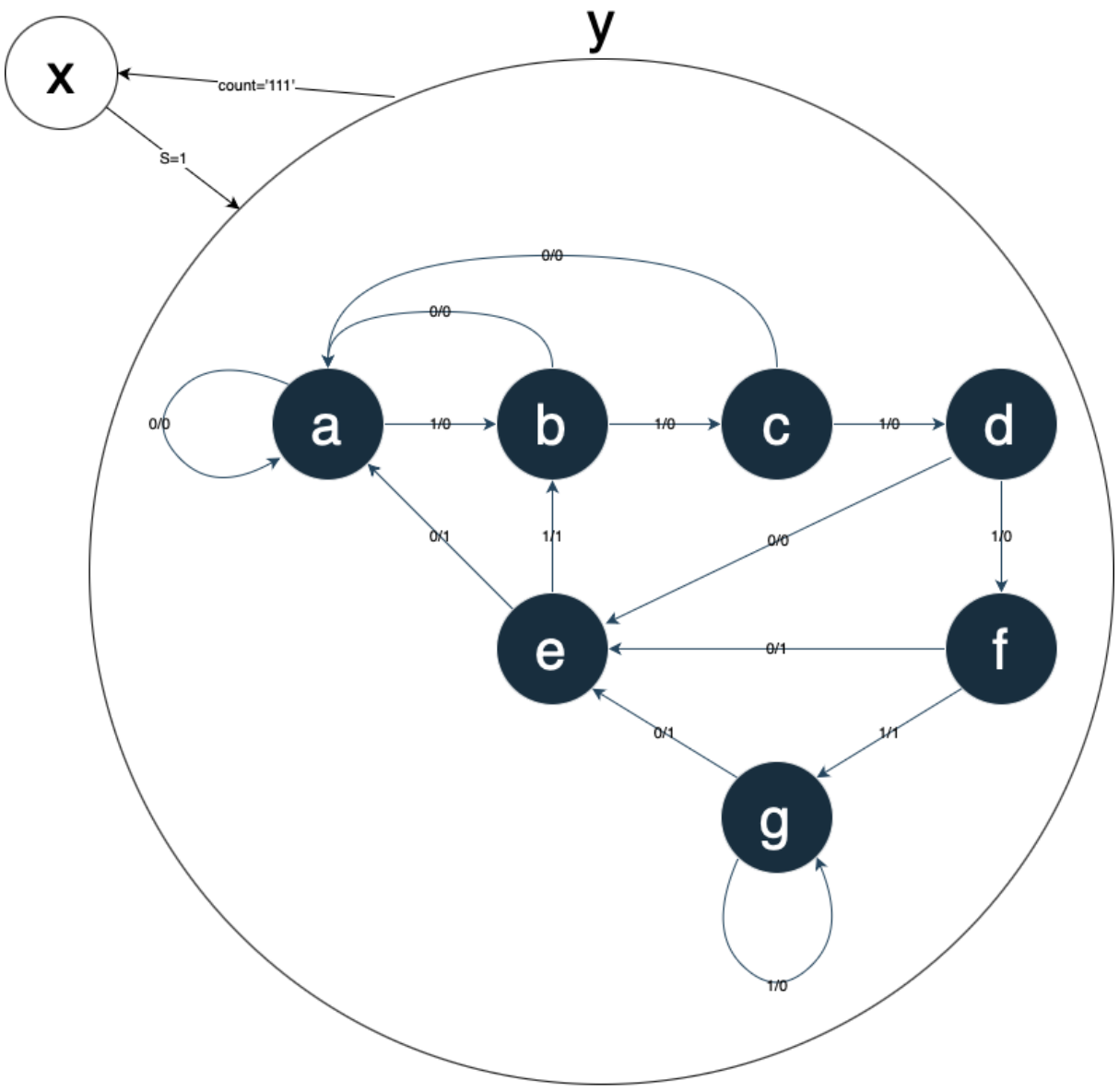
111 Detector

20181536 엄석훈

1. State diagram 및 설명



위의 그림은 직접 그린 111 detector의 state diagram이다. 그리고 x, y는 나중에 설명하고 111 detector의 실제 기능을 하는 a부터 g까지의 state diagram을 설명하면 다음과 같다. 우선 mealy type으로 설계하였고 clock latency는 2이다. 즉 111이 detect 되고 2 클럭 뒤에 1이 출력된다.

초기에는 state a에서 시작한다.

Input이 1이 0개일 때 : state는 a -> a 로 진행된다.

Input이 1이 1개일 때 : state는 a -> b -> a 로 진행된다.

Input이 1이 2개일 때 : state는 a -> b -> c -> a 로 진행된다.

Input이 1이 3개일 때 : state는 a -> b -> c -> d -> e -> a 또는 b로 진행된다. e에서 a또는 b로 갈 때 출력이 1이 총 1번 나온다.

Input이 1이 4개일 때 : state는 a -> b -> c -> d -> f -> e -> a 또는 b로 진행된다. f에서 e로 갈 갈 때, e에서 a 또는 b로 갈 때 출력이 1이 총 2번 나온다.

Input이 1이 5개일 때 : state는 a -> b -> c -> d -> f -> g -> e -> a 또는 b로 진행된다. f에서 g로 갈 때, g에서 e로 갈 때, e에서 a 또는 b로 갈 때 출력이 1이 총 3번 나온다.

Input이 1이 6개일 때 : state는 a -> b -> c -> d -> f -> g -> g -> e -> a 또는 b로 진행된다. f에서 g로 갈 때, g에서 e로 갈 때, e에서 a 또는 b로 갈 때 출력이 1이 총 3번 나온다.

Input이 1이 6개 이상일 때 : state는 a -> b -> c -> d -> f -> g -> g -> ... -> g -> e -> a 또는 b로 진행된다. f에서 g로 갈 때, g에서 e로 갈 때, e에서 a 또는 b로 갈 때 출력이 1이 총 3번 나온다.

따라서 이 state diagram을 따라서 진행되면 111이 처음 detect되고 2클럭 뒤에 1이 나오고 마지막 2번의 111에 대해서도 2클럭 뒤에 1이 나오게 된다. Clock latency를 2로 한 것은 아무래도 마지막 111이 나오는 것을 확인해야만 마지막에서 2번째의 111에 대한 출력도 1을 낼 수 있기 때문에 2의 clock latency를 주었다.

그리고 다음으로 옵션을 구현하기 위해서 카운터를 설계하여 위의 방식대로 111이 detect 될 때마다 카운터를 1씩 증가시켜 카운터가 7이 되면 detector를 reset을 시키고 state를 x로 바꾸어 더 이상 detect를 하지 않도록 설계하였다. 또한 다시 S = 1이 들어오게 되면 y로 state를 바꿔서 detector의 기능을 수행하도록 설계해주었다.

2. VHDL code 첨부 및 설명

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Detector is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        S : in STD_LOGIC;
        input : in STD_LOGIC;
        output : out STD_LOGIC
    );
end Detector;

architecture Behavioral of Detector is
        type st_mealy is (a,b,c,d,e,f,g);
        signal state : st_mealy := a;

        type st_mealy2 is (x,y);
        signal state2 : st_mealy2 := x;

        signal count : std_logic_vector(2 downto 0) := "000";
        signal stop : std_logic := '1';

begin
    process(clk, reset)
    begin
        if reset = '1' then
            output <= '0';
            state <= a;
        elsif stop = '1' then
            output <= '0';
            state <= a;
            count <= "000";
        elsif rising_edge(clk) and state2 = y then
            case state is
                when a =>
```

```
                output <= '0';
                if input = '0' then
                    state <= a;
                else
                    state <= b;
                end if;
            when b =>
                output <= '0';
                if input = '0' then
                    state <= a;
                else
                    state <= c;
                end if;
            when c =>
                output <= '0';
                if input = '0' then
                    state <= a;
                else
                    state <= d;
                end if;
            when d =>
                output <= '0';
                if input = '0' then
                    state <= e;
                else
                    state <= f;
                end if;
            when e =>
                output <= '1';
                count <= std_logic_vector(unsigned(count) + 1);
                if input = '0' then
                    state <= a;
                else
                    state <= b;
                end if;
            when f =>
                output <= '1';
                count <= std_logic_vector(unsigned(count) + 1);
                if input = '0' then
```

```
                    state <= e;
                else
                    state <= g;
                end if;
            when g =>
                if input = '0' then
                    state <= e;
                    output <= '1';
                    count <= std_logic_vector(unsigned(count) + 1);
                else
                    output <= '0';
                    state <= g;
                end if;
            when others =>
                output <= '0';
                state <= a;
        end case;
    end if;
end process;

process(clk, reset)
begin
    if rising_edge(clk) then
        case state2 is
            when x =>
                if S = '1' then
                    state2 <= y;
                    stop <= '0';
                else
                    state2 <= x;
                end if;
            when y =>
                if count = "111" then
                    state2 <= x;
                    stop <= '1';
                else
                    state2 <= y;
                end if;
        end case;
```

```
        end if;
    end process;
end Behavioral;
```

이번 과제를 위해 작성한 VHDL 코드는 위와 같다. 먼저 detector의 port에는 클럭인 clk, 리셋 신호인 reset, 옵션을 위한 S, 입력인 input신호가 각각 입력 포트이고, output신호가 출력 포트이다. 그리고 내부 signal에는 detector의 state에는 a, b, c, d, e, f, g가 있고 옵션 기능을 위한 state2가 x, y가 존재한다. 또한 옵션 기능을 구현하기 위해서 111이 7번 나온 것을 감지하기 위해 count라는 3개의 비트를 가지는 STD_LOGIC_VECTOR를 선언하고 detector가 동작 중인지 멈춰 있는지 표시하는 stop signal을 선언해주었다.

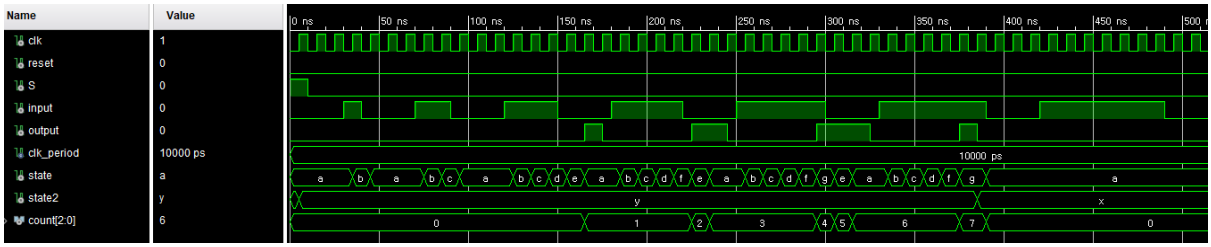
그리고 process는 총 2개 존재하는데 첫 번째 process는 detector의 기능을 구현한 process이다. process에서 먼저 reset신호가 1로 들어오면 state를 a로 초기화 해주고 0이 출력되도록 하였고 stop신호가 1로 들어오면 reset과 마찬가지로 state를 a로 초기화 해주고 0이 출력되며 count또한 000으로 초기화 되도록 해주었다. 이 두 신호의 차이점은 reset은 언제든지 detector가 초기 상태로 돌아가는 동작이고 stop은 count가 7이 되어서 detector기능 자체를 멈추는 동작이기 때문에 추가적으로 count도 000으로 바꿔주었다. 그리고 이 두가지 경우가 아닐 때 클럭의 rising edge이고 state2가 y로 detector가 동작 중 이라면 입력에 따라 state가 변화면서 출력이 발생한다.

여기서부터는 위에서 설명한 state diagram의 동작에 따라서 코드를 짜주었다. 먼저 state가 a인 경우는 입력이 1이면 b로 가고 입력이 0이면 다시 a로 가며 출력은 0이 되도록 해주었다. 다음으로 state가 b인 경우는 출력은 항상 0이고, 입력이 1이면 c로 입력이 0이면 a로 가게 했다. 다음으로 state가 c인 경우는 출력이 항상 0이고, 입력이 1이면 state를 d로 하고, 입력이 0이면 a로 가게 했다. 다음으로 state가 d인 경우는 출력이 항상 0이고, 입력이 1이면 f로 가고, 입력이 0이면 e로 가도록 해주었다. 다음으로 state가 e인 경우는 출력은 항상 1이고 입력이 1이면 b로, 입력이 0이라면 a로 가게 해주었다. 다음으로 state가 f인 경우도 항상 출력이 1이고 입력이 1이면 g로, 입력이 0이면 e로 가게 해주었다. 마지막으로 state가 g인 경우는 입력이 0이면 e로 가면서 출력을 1을 내고, 입력이 1이면 다시 g로 가면서 출력은 0이 되게 했다. 이 과정에서 출력이 1이 나올 때는 항상 count를 하나 더해주고 count가 111이 되었다면 혹시 모를 경우로 stop이 1로 활성화되어 있으면 state를 a로 옮겨주며 count를 초기화 해주었다.

그 다음 process는 옵션을 위한 동작으로 클럭이 rising edge일 때 state2의 상태가 x로 detector가 멈춘 상태라면 S로 1이 들어오면 state2를 y로 하고 stop을 0으로 해서 detector를 활성화시키고, state2의 상태가 y였다면 count가 111이 되면 state2를 x로 바꾸고 stop을 1로 해서 detector를 멈추고 count가 111이 아니면 state2를 y로 유지하는 역할을 하는 process이다.

따라서 코드가 길어 보이지만 사실은 각 state별로 input에 따라 state를 바꿔가며 출력을 내는 간단한 회로이다. 또한 카운터 옵션을 추가해 detector를 멈출지 말지 결정해주는 간단한 회로를 옵션으로 추가해 주었다.

3. 시뮬레이션 결과 첨부 및 분석



처음 시뮬레이션 시작하면 S = 1을 주어서 detector를 활성화한다. 그 결과 state2가 y로 활성화된 것을 볼 수 있다. 그리고 input에는 순서대로 1을 1번, 2번, 3번 ... 7번까지 연속으로 주었다. 그 각각의 결과는 다음과 같다.

10을 입력한 경우 state가 a -> b -> a로 갔으며 output은 계속 0이었다.

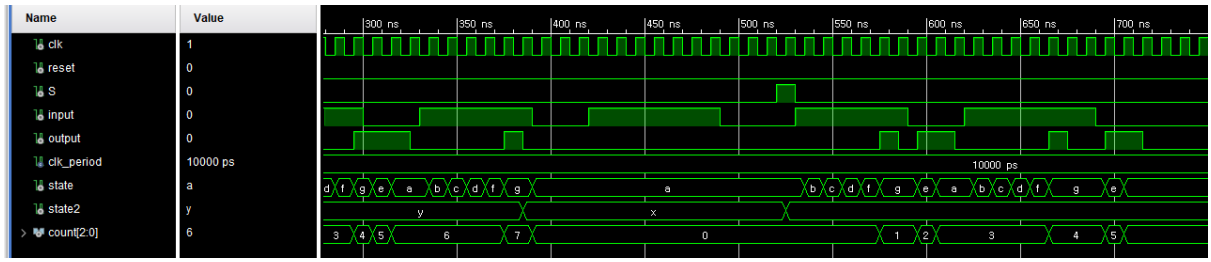
110을 입력한 경우 state가 a -> b -> c -> a로 갔으며 output은 계속 0이었다.

1110을 입력한 경우 state가 a -> b -> c -> d -> e -> a로 갔으며 state가 e에서 a로 넘어가는 과정에서 output으로 1이 한번 나오고 count는 1이 된 것을 확인할 수 있다.

11110을 입력한 경우 state가 a -> b -> c -> d -> f -> e -> a로 갔으며 state가 f에서 e로 넘어갈 때와 e에서 a로 넘어갈 때 각각 output이 1로 나와 두 클럭 연속으로 output이 1이 나왔으며 count도 2를 거쳐 3까지 진행되었다.

111110을 입력한 경우 state가 a -> b -> c -> d -> f -> g -> e -> a로 갔으며 f에서 g로 넘어갈 때와 g에서 e로 넘어갈 때, e에서 a로 넘어갈 때 각각 output이 1로 나와 세 클럭 연속으로 output이 1이 나왔으며 count도 4, 5를 거쳐 6이 되었다.

1111110을 입력한 경우 state가 a -> b -> c -> d -> f -> g -> a로 갔는데 이는 f에서 g로 넘어갈 때 1이 한번 출력되며 count가 6에서 7이 되었고 그 결과 state2가 x로 가면서 count는 0으로 초기화, state는 a로 초기화되었고 따라서 그 이후에는 detector가 비활성화 되어서 output이 계속 0이 나왔다.



그리고 아까 상황에 이어서 시뮬레이션이 계속 진행되었다. 400ns인 현재 상태는 state2가 x로 detector가 멈춰 있는 상태이다. 따라서 S = 1을 넣어 주어 먼저 state2를 y로 바꿔 detector를 활

성화했다. 그리고 이전 상황에 이어서 다시 input을 다시 넣어주었다.

다시 1111110을 입력한 경우 state가 a -> b -> c -> d -> f -> g -> g -> e -> a로 갔으며 f에서 g로 넘어갈 때와 g에서 e로 넘어갈 때, e에서 a로 넘어갈 때 각각 output이 1로 나왔으며 count는 다시 0부터 1, 2를 거쳐 3이 되었다.

다음으로 11111110을 입력한 경우 state가 a -> b -> c -> d -> f -> g -> g -> g -> e -> a로 갔으며 f에서 g로 넘어갈 때와 g에서 e로 넘어갈 때, e에서 a로 넘어갈 때 각각 output이 1로 나왔으며 count는 3부터 4, 5를 거쳐 6이 된 것을 확인할 수 있다.

이번 시뮬레이션을 통해서 다양한 횟수로 1이 연속으로 나오는 경우의 출력을 직접 테스트해보고 옵션 기능이었던 카운터 기능까지 잘 구현된 것을 확인할 수 있었다.