합성을 위한 **VHDL Coding Style 3**

국일호
goodkook@csvlsi.kyunghee.ac.kr
http://www.csvlsi.kyunghee.ac.kr

합성을 위한 VHDL Coding Style 세 번째 연재로서 이번에는 Finite State Machine 의 설계에 대해서 살펴 보도록 한다. State Machine 은 프로세서 제어 유닛(Control Unit)의 인스트럭션 디코드 및 실행 을 위한 시퀀서(instruction sequencer)나 컨트롤러의 제어신호 생성 등 디지털 회로 설계에 있어서 매우 광범위하게 이용 된다. 컨트롤러는 바로 FSM 을 의미하는 것이라 하겠다.

1. FSM (Finite State Machine) 기본 구조

State Machine 을 구성하는 요소 들로는 현재 상태와 전이(Transition)할 다음 상태 등을 나타내는 상태(State), 전이 조건의 판단과 방향 그리고 각 상태의 입력과 출력 등 이다. 디지털 회로의 State Machine 은 보통 유한 상태 머신(FSM:Finite State Machine) 이라고 하며 상태전이는 클럭에 따라 진행되고 상태를 나타낼 플립플롭과 전이 조건 판단 및 각 상태에 따른 출력을 표현하는 조합 회로로 구성 된다. FSM 의 기본 구조는 그림 1 과 같다. FSM 을 설계할 때 상태 전이 표(State Table) 혹은 전이 도(State Diagram)를 작성하게 된다. 상태 전이도에 의한 경우 그래픽적으로 표현에 용이한 반면 상태 최적화 알고리즘을 적용할 때에는 상태 테이블을 작성하여야 한다. 최근 EDA 툴들은 상태 표 혹은 상태 전이도에 의한 FSM 의 설계 입력 방식을 모두 지원 하고 있어서 용이하게 FSM 을 설계할 수 있다. 비교적 고급의 툴들에서는 상태표 혹은 상태도를 모두 지원 하지만 상태도만을 지원하는 툴들이 비교적 수적으로 많다. PC 상에서 사용할 수 있는 툴로는 Summit Design 사의 VisualHDL 이나 Mentor 의 Renoir 가 상태도와 상태표를 모두 지원하며 StateCAD, AdctiveVHDL 등은 상태도 만을 지원한다. 이런 툴들은 상태도를 입력하면 이를 처리하여 VHDL 소스를 생성해 준다.
FSM 의 형태는 Mealy 형과 Moore 형 그리고 이들을 혼합한 형태가 있다. FSM 의 기본 구조는 현재 상태(Current State)를 저장하는 레지스터로 구성된 순차 회로, 다음 상태로의 전이를 결정(Next State Logic)하기 위하여 비교기 등을 포함하는 조합회로, 그리고 현재 상태에서 출력을 나타내기 위한 조합회로로 구성되어 있다. FSM 의 출력이 입력과 현재 상태에 의하여 결정되는 경우 Mealy Machine 이라 하며, Moor Machine 은 입력은 오직 다음 상태의 전이를 결정하고, 출력은 현재상태에 의해서만 정해지는 경우이다.

http://csvlsi.kyunghee.ac.kr

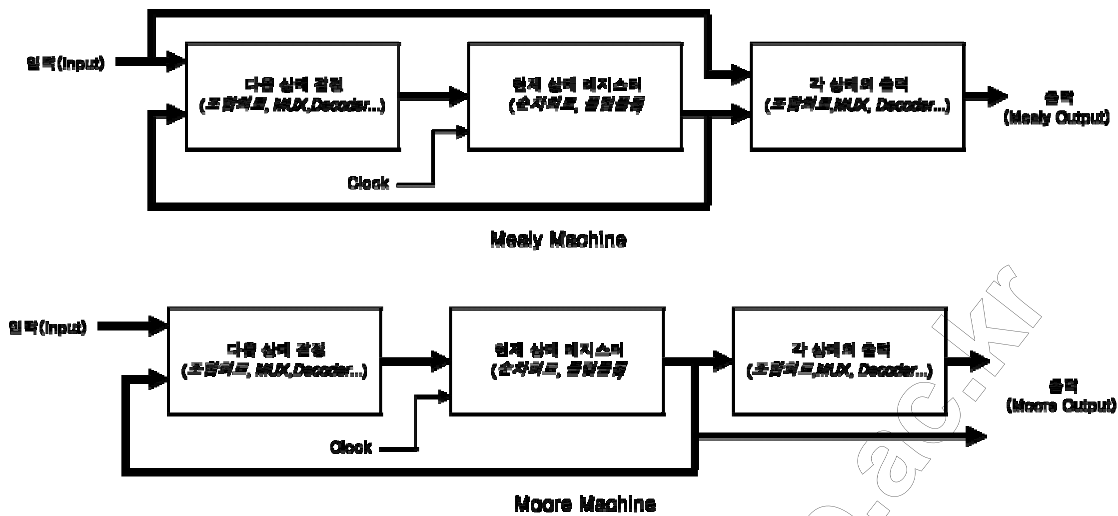


그림 **1. Mealy/Moore** 유한상태머신**(FSM: Finite State Machine)**의 기본구조

VHDL 로 FSM 을 설계하는 방법이 따로 존재하는 것은 아니다. 다만 VHDL 코딩 스타일에 있어서 다소 주의할 점이 있다. VHDL 에서 State Machine 은 if 문에 의한 클럭 트리거와 case 문에 의하여 기술된다.

```
if (clock'event and clock='1') then
        case current_state is
                when S0 =>
                        …..
                        current_state <= S1;
                when S1 =>
                …..
                        current_state <= S2;
        end case;
end if;
```

순차 회로를 기술할 때에는 가급적 리셋(reset)을 표현하는 것이 좋다. 특히 FSM 의 최초 상태를 결정하려면 리셋 조건을 반드시 포함하도록 하여야 한다. FSM 에서 리셋은 동기적(synchronous) 혹은 비동기적(asynchronous)인 방법이 있으나 가급적 비동기 리셋을 취하도록 함으로서 회로 전체적으로 초기화가 이루어 지도록 하는 것이 좋다. 그렇지 않을 경우 초기 지연에 의한 알 수 없는 출력이 발생하게 되어 다른 회로에 영향을 미치거나 디버깅에 어려움을 초래할 수 있다. 비동기 리셋을 포함하는 FSM 의 기술은 다음과 같다.

```
If reset='0' then
        Current_state <= S_INIT;
elsif (clock'event and clock='1') then
        case current_state is
                when S_INIT =>
                        …..
                        current_state <= S0;
                when S0 =>
                        …..
```

http://esvlsi.kyunghee.ac.kr

```
                    current_state <= S1;
          when S1 =>
                    …..
                    current_state <= S2;
     end case;
end if;
```

FSM 은 설계할 때 상태의 개수가 정해져 있다. 따라서 상태의 표현은 VHDL 의 enumeration type 을 이용한다. 다음과 같은 예를 생각해 보도록 하자.

```
type MONTH is (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec);
```

위에서 정의 된 MONTH 는 12 가지의 경우만을 갖는 것으로서 이를 바이너리 형태로 나타내려면 4 비트가 필요하다. 그러나 4 비트로 표현할 수 있는 경우의 수는 16 가지이므로 “0000” 에서 “1011” 까지만 사용하고 나머지 경우는 사용하지 않음을 의미한다. 이러한 바이너리 표현은 고정된 유한개의 개수 표현이 아니며 허용된 비트 수 내에서 2 의 n 승 개 만큼의 경우를 가진다. FSM 에서 각 상태의 개수는 정해져 있으며 그 이외의 상태가 있을 수 없다. 또한 FSM 에서의 각 상태란 서로 배타적인 관계에 있으며 상태를 나타내는 시그널들은 절대로 연산에 참여할 수 없다. FSM 에서 상태를 나타낼 때는 보통 One-Hot 로 인코딩(One-Hot Encoding) 한다. One-Hot 는 각각의 경우에 대하여 비트를 할당하는 것으로 위의 예로든 MONTH 의 경우 12 비트가 필요해진다. FSM 의 상태를 나타내는 방법으로 바이너리 와 One-Hot encoding 을 표 1 에 보였다. One-Hot 방법은 바이너리 방법에 비하여 더 많은 플립플롭을 필요로 하지만 상태의 표현이 명확하고 별도의 상태 디코딩할 필요가 없게 되며 이에 따른 지연이 없다.

표 **1. FSM** 의 바이너리 및 **One-Hot Encoding** 예

MONTH	Binary	One-Hot
Jan	0000	000000000001
Feb	0001	000000000010
Mar	0010	000000000100
Apr	0011	000000001000
May	0100	000000010000
Jun	0101	000000100000
Jul	0110	000001000000
Aug	0111	000010000000
Sep	1000	000100000000
Oct	1001	001000000000
Nov	1010	010000000000
Dec	1011	100000000000

앞서 언급했지만 FSM 은 컨트롤러의 단계별 동작을 제어하기 위한 신호를 발생하는 시퀀서로 이용되므로 모든 상태에서 출력은 지연이 일정해야 한다. 특정 상태의 출력이 다른 상태의 경우보다 지나치게 길게 나와서는 좋지 않다. 따라서 클럭에 의한 상태의 전이와 이에 따른 출력이 조합 회로를 통하여 이루어진다는 점을 감안하여 출력 로직은 단순한 멀티 플렉서(MUX,

Multiplexer)나 디코더(Decoder)를 사용하도록 한다. 현재 상태와 입력에 따른 출력 로직(output logic)의 기술은 다음과 같이 case 문을 이용하여 간단하게 기술 되도록 한다. 이때 주의 할 것은 모든 상태와 모든 출력을 빠짐없이 기술 함으로써 필요 없는 래치(latch)가 생기지 않도록 주의 해야 하며 그렇지 않을 경우 때에 따라 FSM 이 엉뚱한 동작을 일으킬 수도 있다. 정확한 FSM 을 기술하기 위해서는 그림 2 의 FSM 구조를 충실하게 따르도록 하는 것이 무엇 보다도 중요하다.

2. FSM 예제 : 교통 신호등 제어기

다음은 FSM 의 한 예로서 일반 차로 (High-Way)와 샛길(Farm-Road)이 교차하는 교차로의 신호등 제어기(Traffic Light Controller)이다. 예제의 교통 신호등이 설치된 교차로의 조건은 각 도로에 녹-황-적색 등이 설치되어 있고 샛길에는 교통량 감지기가 설치 되어 있는 교차로이다. 통행 우선순위는 일반 도로이므로 샛길에서 교통량이 감지될 경우에 한하여 통행을 허용한다. 신호등이 녹색에서 적색으로 바뀌기 전에 예비 신호로서 잠시동안 황색등이 켜진다. 샛길에서의 교통량이 감지 되었더라도 일정기간 후에 통행을 종료하도록 하여 일반 차로의 통행을 방해하는 일이 없도록 한다. 예제 1 의 “신호등 제어기”에서는 황색등이 켜지는 기간과 샛길 통행 허용 기간을 정하기 위하여 두개의 타이머/카운터를 사용한다. 시뮬레이션 결과는 그림 2 와 같다. State Machine (sm_tlc)는 샛길에서의 교통량 요구에 따라 신호등 On-Off 상태로 전환되고 두개의 타이머를 제어 한다. “highway_green” 상태에서 샛길의 교통량이 있을 경우 (farmroad_traffic=’1’) 타이머 계수를 시작하고 황색등 상태(highway_yellow)로 전환, “highway_yellow” 상태에서는 황색등화 구간타이머가 종료 할 때(short_timeout=’1’)까지 대기한 후 샛길에 녹색 등화를 켠다. 샛길의 녹색등화 상태(farmroad_green)는 샛길의 교통량이 없거나(not farmroad_traffic=’1’) 일정기간이 지난 후 (long_timeout=’1’) 황색등 상태(farmroad_yellow)로 바뀌고 다시 도로의 녹색등 상태로 전환된다. 각 도로에 설치된 신호등을 켜고 끄는 것은 제어기의 상태에 따른 출력으로 case 문을 이용한 디코더(out_tlc)로 기술되었다.

[예제 1] 신호등 제어기 (TLC:Traffic Light Controller)

```
entity tlc is
port ( clk, reset, farm_traffic: in std_logic;
          farmroad_green_on,
          farmroad_yellow_on,
          farmroad_red_on,
          highway_green_on,
          highway_yellow_on,
          highway_red_on: out std_logic);
end tlc;

architecture a_classic of tlc is
     type states is (highway_green, highway_yellow, farmroad_green, farmroad_yellow);
```

```
    signal tlc_state: states;
    signal short_timer, long_timer: unsigned(7 downto 0);
    signal set_timer, short_timeout, long_timeout: std_logic;

    procedure timer ( signal    set, clk, set_timer : std_logic;
                      signal    timer : inout unsigned(7 downto 0);
                      constant load : unsigned(7 downto 0);
                      signal    timeout : out std_logic) is
    begin
        if set='0' or set_timer='1' then        -- gated reset !!
            timer <= load;
            timeout <= '0';
        elsif clk'event and clk='1' then
            if timer = 0 then
                timeout <= '1';
            else
                timer <= timer - 1;
            end if;
        end if;
    end;

begin

    timer ( reset, clk, set_timer, short_timer, "00011111", short_timeout);
    timer ( reset, clk, set_timer, long_timer, "11111111", long_timeout);

    sm_tlc : process ( clk )
    begin
        if clk'event and clk='1' then
            if reset='0' then
                tlc_state <= highway_green;
                set_timer <= '0';
            else
                set_timer <= '0';
                case tlc_state is
                    when highway_green   =>
                        if farm_traffic='1' and long_timeout='1' then
                            tlc_state <= highway_yellow;
                            set_timer <= '1';
                        end if;

                    when highway_yellow =>
                        if short_timeout='1' then
                            tlc_state <= farmroad_green;
                            set_timer <= '1';
                        end if;

                    when farmroad_green =>
                        if not farm_traffic='1' or long_timeout='1' then
                            tlc_state <= farmroad_yellow;
                            set_timer <= '1';
                        end if;

                    when farmroad_yellow =>
                        if short_timeout='1' then
                            tlc_state <= highway_green;
                            set_timer <= '1';
```

http://cslsi.kyunghee.ac.kr

```
                    end if;
                when others =>
            end case;
        end if;
    end if;
end process;

-- Decode states and drive lights.
out_tlc : process(tlc_state)
begin
    case tlc_state is
        when highway_green    =>
            farmroad_green_on    <= '0';
            farmroad_yellow_on <= '0';
            farmroad_red_on       <= '1';
            highway_green_on      <= '1';
            highway_yellow_on    <= '0';
            highway_red_on         <= '0';

        when highway_yellow =>
            farmroad_green_on    <= '0';
            farmroad_yellow_on <= '0';
            farmroad_red_on       <= '1';
            highway_green_on      <= '0';
            highway_yellow_on    <= '1';
            highway_red_on         <= '0';

        when farmroad_green =>
            farmroad_green_on    <= '1';
            farmroad_yellow_on <= '0';
            farmroad_red_on       <= '0';
            highway_green_on      <= '0';
            highway_yellow_on    <= '0';
            highway_red_on         <= '1';

        when farmroad_yellow =>
            farmroad_green_on    <= '0';
            farmroad_yellow_on <= '1';
            farmroad_red_on       <= '0';
            highway_green_on      <= '0';
            highway_yellow_on    <= '0';
            highway_red_on         <= '1';

        when others =>
            farmroad_green_on    <= '-';
            farmroad_yellow_on <= '-';
            farmroad_red_on       <= '-';
            highway_green_on      <= '-';
            highway_yellow_on    <= '-';
            highway_red_on         <= '-';

    end case;

end process;

end a_classic;
```

http://vlsi.kyunghee.ac.kr

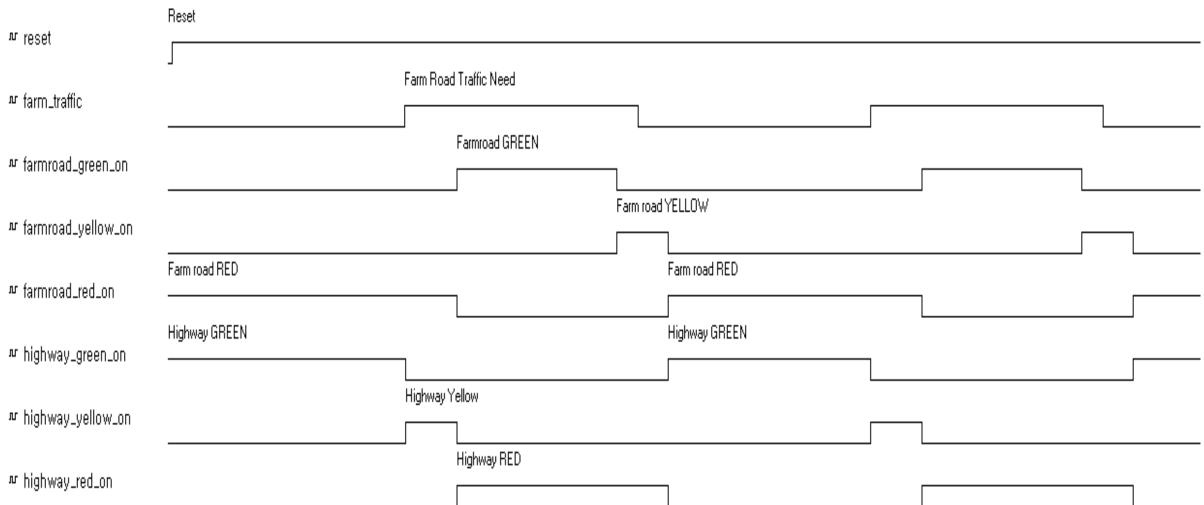
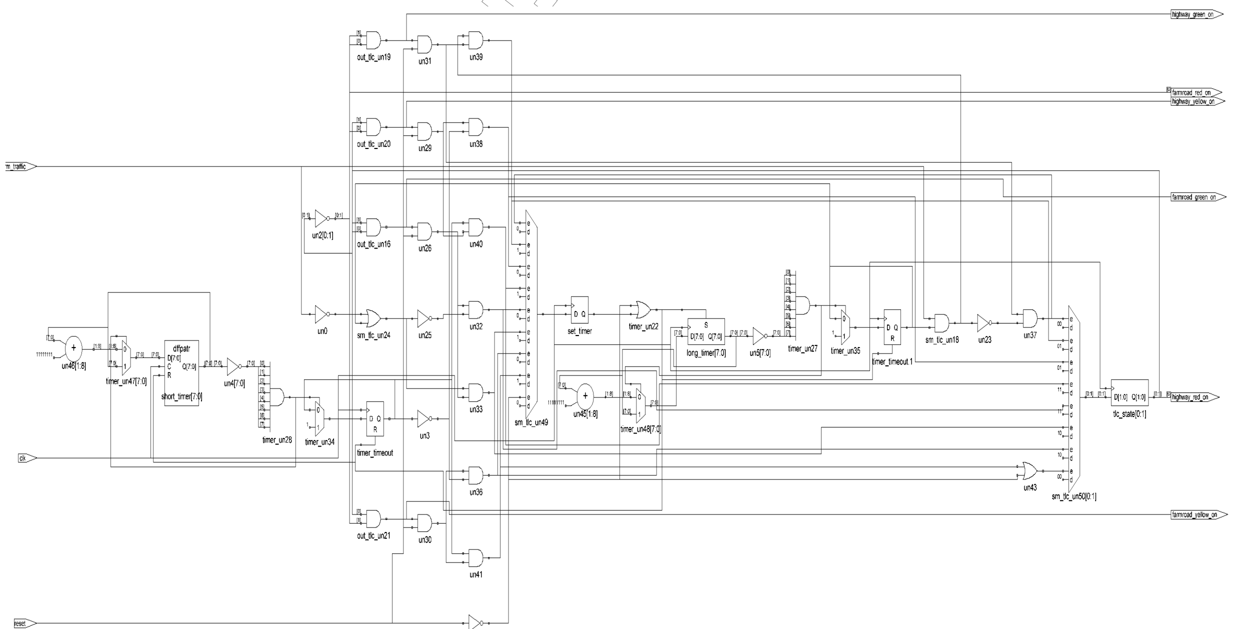
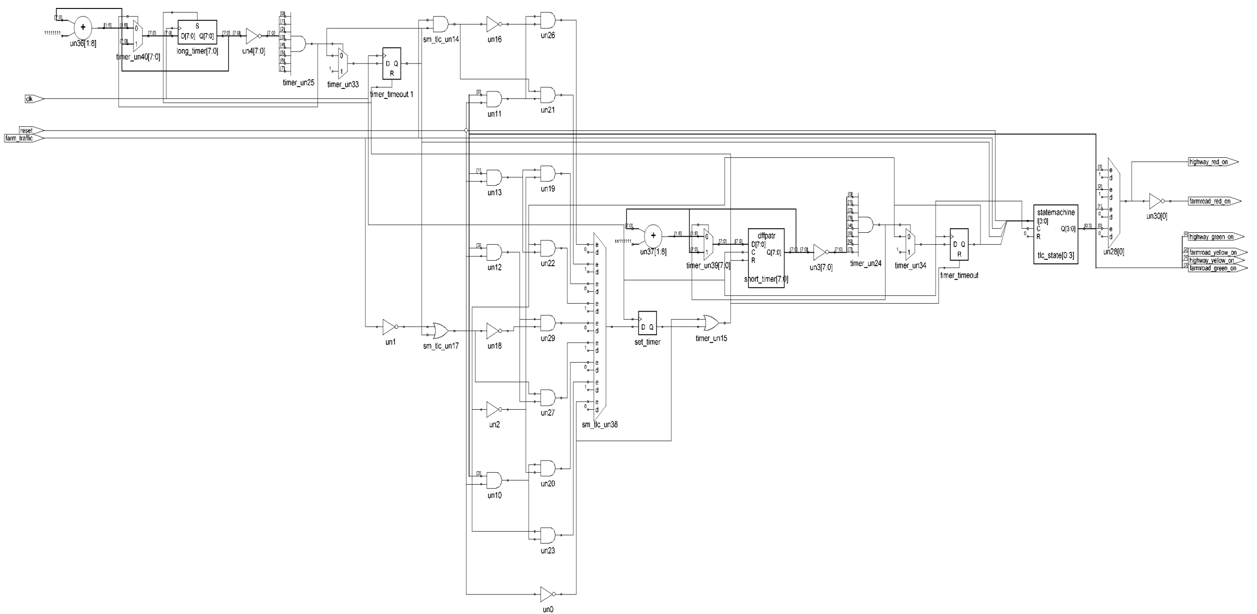


그림 **2.** [예제 **1] TLC** 의 시뮬레이션 결과

그림 3 은 [예제 1]의 합성 결과를 나타낸 것이다. 그림 3(a)의 경우 합성에 FSM 합성 옵션을 이용한 경우이고 (b)는 옵션을 주지 안고 합성한 경우이다. 두개의 결과를 비교해 보면 FSM 옵션을 준 경우 One-Hot 로 인코딩하여 상태를 나타내는 신호인 tlc_state 가 각각 4 비트로 합성되어 있고, FSM 합성 옵션을 주지 않은 경우 상태 전이에 필요한 상태 디코딩 로직이 포함되어 있는 것을 볼 수 있다.



(a) Binary Encoding



(b) One-Hot Encoding

그림 **3. [**예제 **1] TLC** 의 합성 결과

3. FSM 예제 **:** 파형 발생기

다음의 [예제 2]는 일종의 파형 발생용 FSM 의 예로서 National Semiconductor 사의 모터 드라이버 칩인 LMD18245 를 이용한 스테핑 모터 드라이브용 상(Phase)발생기 이다. FSM 을 이용하여 주기적인 신호를 발생 시킬 수 있음을 보여주는 예라 하겠다. FSM 합성 옵션을 사용하는 것은 설계자의 선택이다. 가능하면 One-Hot 로 하는 것이 좋으나 합성된 하드웨어의 크기가 너무 커서 원하는 디바이스에 Fitting 이 안될 경우 Binary Encoding 으로 합성해 볼 수 있을 것이다. 다만 이때 합성된 회로가 원하는 속도와 동작을 하는지 검증해 보아야 한다. 2 개의 [예제 2]를 One-Hot 방식으로 합성한 경우 Altera 7032 CPLD 에 Fitting 되지 않았으나 Binary Encoding 으로 합성하였는데 스테핑 모터 구동신호가 워낙 저속에서 작동하기 때문에 별다른 문제를 일으키지는 않았다. 그림 4(a)는 LMD18245 의 매뉴얼 상에 나와있는 구동 파형이며 (b)는 [예제 2]의 시뮬레이션 결과 이다.

[예제 **2]** 스테핑 모터 구동용 상**(Phase)**발생 **FSM**

```
entity PulseGen is
        port (
                dir           : in std_logic;
                clk           : in std_logic;
                reset         : in std_logic;
                direction_a : out std_logic;
                m_a421        : out std_logic;
```

```
            m_a3          : out std_logic;
            direction_b : out std_logic;
            m_b421        : out std_logic;
            m_b3          : out std_logic );
end PulseGen;

architecture behave of PulseGen is
    type states is (S0, S1, S2, S3, S4, S5, S6, S7, S8);
    signal current_state : states;
    signal t_a421, t_b421 : std_logic;

begin

        process(clk, reset)
        begin

            if (reset='0') then
                current_state <= S0;
            elsif (clk'event and clk='1') then
                case current_state is
                    when S0 =>
                        current_state <= S1;

                    when S1 =>
                        current_state <= S2;

                    when S2 =>
                        current_state <= S3;

                    when S3 =>
                        current_state <= S4;

                    when S4 =>
                        current_state <= S5;

                    when S5 =>
                        current_state <= S6;

                    when S6 =>
                        current_state <= S7;

                    when S7 =>
                        current_state <= S8;

                    when S8 =>
                        current_state <= S1;

                end case;
            end if;
        end process;

        process(current_state, dir, t_a421, t_b421)
        begin

            case current_state is
                when S0 =>
                    direction_a <= '1';
```

http://csvlsi.kyunghee.ac.kr

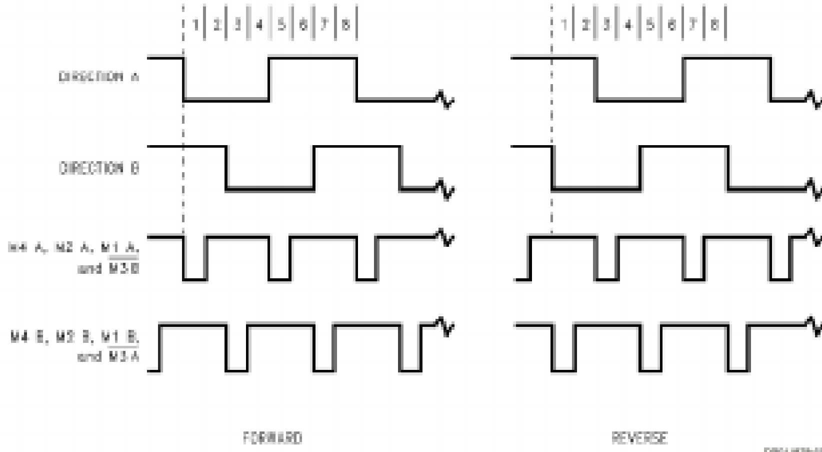
```
		direction_b <= '1';
		t_a421		<= '1';
		t_b421		<= '1';

	when S1 =>
		if dir='1' then
			direction_a <= '0';
			direction_b <= '1';
			t_a421		<= '0';
			t_b421		<= '1';
		else
			direction_a <= '1';
			direction_b <= '0';
			t_a421		<= '1';
			t_b421		<= '0';
		end if;

	when S2 =>
		if dir='1' then
			direction_a <= '0';
			direction_b <= '1';
			t_a421		<= '1';
			t_b421		<= '1';
		else
			direction_a <= '1';
			direction_b <= '0';
			t_a421		<= '1';
			t_b421		<= '1';
		end if;

	when S3 =>
		if dir='1' then
			direction_a <= '0';
			direction_b <= '0';
			t_a421		<= '1';
			t_b421		<= '0';
		else
			direction_a <= '0';
			direction_b <= '0';
			t_a421		<= '0';
			t_b421		<= '1';
		end if;

	when S4 =>
		if dir='1' then
			direction_a <= '0';
			direction_b <= '0';
			t_a421		<= '1';
			t_b421		<= '1';
		else
			direction_a <= '0';
			direction_b <= '0';
			t_a421		<= '1';
			t_b421		<= '1';
		end if;

	when S5 =>
```

http://csvlsi.kyunghee.ac.kr

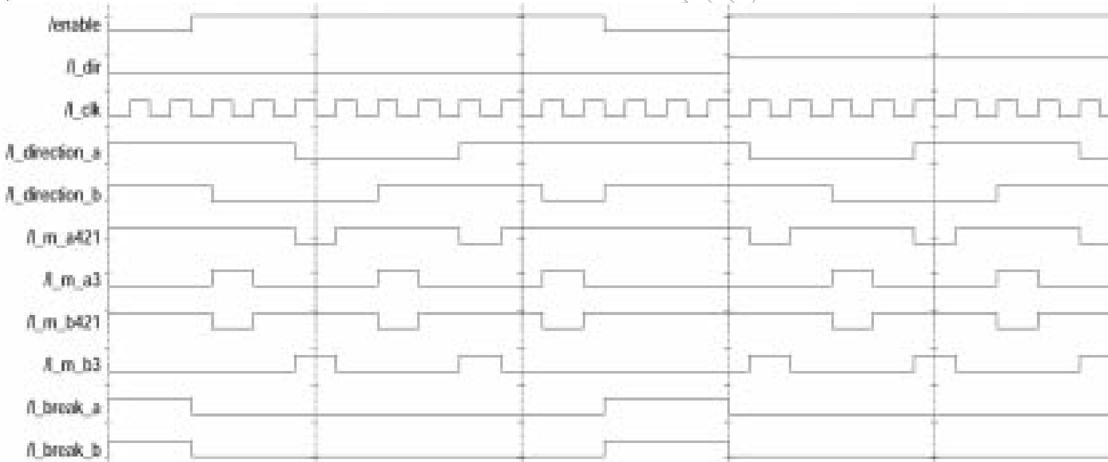
```
                    if dir='1' then
                            direction_a <= '1';
                            direction_b <= '0';
                            t_a421        <= '0';
                            t_b421        <= '1';
                    else
                            direction_a <= '0';
                            direction_b <= '1';
                            t_a421        <= '1';
                            t_b421        <= '0';
                    end if;

            when S6 =>
                    if dir='1' then
                            direction_a <= '1';
                            direction_b <= '0';
                            t_a421        <= '1';
                            t_b421        <= '1';
                    else
                            direction_a <= '0';
                            direction_b <= '1';
                            t_a421        <= '1';
                            t_b421        <= '1';
                    end if;

            when S7 =>
                    if dir='1' then
                            direction_a <= '1';
                            direction_b <= '1';
                            t_a421        <= '1';
                            t_b421        <= '0';
                    else
                            direction_a <= '1';
                            direction_b <= '1';
                            t_a421        <= '0';
                            t_b421        <= '1';
                    end if;

            when S8 =>
                    if dir='1' then
                            direction_a <= '1';
                            direction_b <= '1';
                            t_a421        <= '1';
                            t_b421        <= '1';
                    else
                            direction_a <= '1';
                            direction_b <= '1';
                            t_a421        <= '1';
                            t_b421        <= '1';
                    end if;

    end case;

    m_a421 <= t_a421;
    m_b3     <= not t_a421;
    m_b421 <= t_b421;
    m_a3     <= not t_b421;
```

http://csvlsi.kyunghee.ac.kr

```
        end process;
end behave;
```



(a)



(b)

그림 **4. [**예제 **2]** 스테핑 모터 구동용 상발생 **FSM** 의 시뮬레이션 결과

4. 그래픽 **FSM** 설계도구

컨트롤러의 FSM 을 설계하는 일은 여간 복잡한 것이 아니다. 더구나 병렬 특성의 하드웨어를 순차적인 처리에 익숙한 언어형태의 도구로서 설계한다는 것이 쉬운 일은 아니다. 이러한 점을 감안하여 최근에 그래픽 툴들이 속속 발표되고 있는 추세이다. 이는 고전적인 스키마틱 회로도 작성 툴과는 다르게 각 블록단위의 모듈들과 데이터의 흐름(data flow)을 한눈에 파악할 수 있도록 하는 등 시스템 측면의 설계 툴들이 주종을 이루고 있다.
전체적인 컨트롤 과정을 한눈에 알아볼 수 있는 그래픽 툴들을 이용하면 쉽게 FSM 을 설계할 수 있다. 그림 5 는 Aldec 사의 ActiveVHDL 에 내장되어

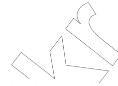
있는 FSM 편집기로 작성한 BlackJack 게임의 상태도이다. BlackJack Game 에 대해서 자세히 알 필요는 없겠으나 예로든 상태도와 이로부터 생성된 VHDL 소스를 이해하기 위해서 간단히 설명하기로 한다. BlackJack 은 일종의 카드 놀이로서 딜러로부터 받은 카드의 숫자 합 중 가장 작은 사람이 이기는 게임이다. 단 Ace 카드를 받은 경우 자신의 카드 총합에서 10 을 뺄 수 있다. 만일 자신의 카드 총합이 21 을 넘으면 그 게임에서 진다.
BlackJack Game 의 진행은 8 개의 상태가 존재하며 받은 카드의 숫자와 그 합의 조건에 따라 이루어 진다. 상태도에는 이와 같은 각 상태로 전이 및 전이 조건이 표현 되어 있으며 각 상태에서의 처리내용 그리고 입출력에 대한 정의가 나타나 있다. 그림 5 의 상태도로부터 생성된 VHDL 소스는 다음과 같다. 생성된 BlackJack 게임의 VHDL 소스를 보면 각 상태가 정의 되어 있고 이들은 One-Hot 으로 Encoding 되어 있음을 알 수 있다. 각 상태의 출력은 병렬 할당문에 의하여 기술 되었다.

```
entity bjack_c is
   port (CARD: in STD_LOGIC_VECTOR (3 downto 0);
            CLOCK: in STD_LOGIC;
            NEW_C: in STD_LOGIC;
            NEW_G: in STD_LOGIC;
            BUST: out STD_LOGIC;
            HAND: out STD_LOGIC_VECTOR (4 downto 0);
            HOLD: out STD_LOGIC;
            NEXT_C: out STD_LOGIC);
end;

architecture bjack_c_arch of bjack_c is

--diagram signal declarations
signal Total: STD_LOGIC_VECTOR (4 downto 0);

-- ONE HOT ENCODED state machine: BlackJack
type BlackJack_type is (Begin_g, BustState, Got_im, Hit_me, HoldState, TenBack, test16, Test21);

attribute enum_encoding of BlackJack_type: type is
             "00000001 " &                -- Begin_g
             "00000010 " &                -- BustState
             "00000100 " &                -- Got_im
             "00001000 " &                -- Hit_me
             "00010000 " &                -- HoldState
             "00100000 " &                -- TenBack
             "01000000 " &                -- test16
             "10000000";                  -- Test21

signal BlackJack: BlackJack_type;

begin
--concurrent signal assignments
--ACTION
DiagramActions:
HAND <= Total;
```

```
BlackJack_machine: process (CLOCK)
--machine variables declarations
variable Ace: BOOLEAN;

begin

if CLOCK'event and CLOCK = '1' then
	if NEW_G='1' then
		BlackJack <= Begin_g;
		Total <= "00000";
		Ace := false;
	else
	case BlackJack is
		when Begin_g =>
			BlackJack <= Hit_me;
		when BustState =>
		when Got_im =>
			if NEW_C='0' then
				BlackJack <= test16;
			end if;
		when Hit_me =>
			if NEW_C='1' then
				BlackJack <= Got_im;
				Total <= Total + CARD;
				Ace :=   (CARD=11) or Ace;
			end if;
		when HoldState =>
		when TenBack =>
			BlackJack <= test16;
		when test16 =>
			if Total > 16 then
				BlackJack <= Test21;
			else
				BlackJack <= Hit_me;
			end if;
		when Test21 =>
			if Total < 21 then
				BlackJack <= HoldState;
			elsif Ace then
				BlackJack <= TenBack;
				Total <= Total - 10;
				Ace :=   FALSE;
			else
				BlackJack <= BustState;
			end if;
		when others =>
			null;
	end case;
	end if;
end if;
end process;

-- signal assignment statements for combinatorial outputs
NEXT_C_assignment:
NEXT_C <= '1' when (BlackJack = Hit_me) else
           '0';
```

http://csvlsi.kyunghee.ac.kr

```
BUST_assignment:
BUST <= '1' when (BlackJack = BustState) else
            '0';

HOLD_assignment:
HOLD <= '1' when (BlackJack = HoldState) else
            '0';

end bjack_c_arch;
```



BJACK_C

CLOCK
NEW_G
NEW_C
CARD[3:0]
Total[4:0]
BUST
HOLD
NEXT_C
HAND[4:0]

BlackJack

Ace

NEW_G='1'

Total <= "00000";
Ace := false;

Begin_g
/1/

NEXT_C <= '0';
BUST <= '0';
HOLD <= '0';

Hit_me
/8/

NEXT_C <= '1';

NEW_C='1'

Total <= Total + CARD;
Ace := (CARD=11) or Ace;

Got_im
/4/

@ELSE

NEW_C='0'

test16
/64/

Total <= Total - 10;
Ace := FALSE;

Total > 16

TenBack
/32/

Ace

2

Test21
/128/

@ELSE

1

Total < 21

HoldState
/16/

HOLD <= '1';

BustState
/2/

BUST <= '1';

그림 5. State Machine 의 예 (Black Jack Game)