Counters

2017-11-15
2016-17101 김종범
2016-17274 이도윤

실습1. 임의의 Sequence를 가진 Counter 구현

000->010->011->101->110->000의 상태를 갖는 3-bit Counter를 구현해 보았다. 전 상태 ABC로부터 다음상태 XYZ를 구하는 카르노 맵은 다음과 같다.

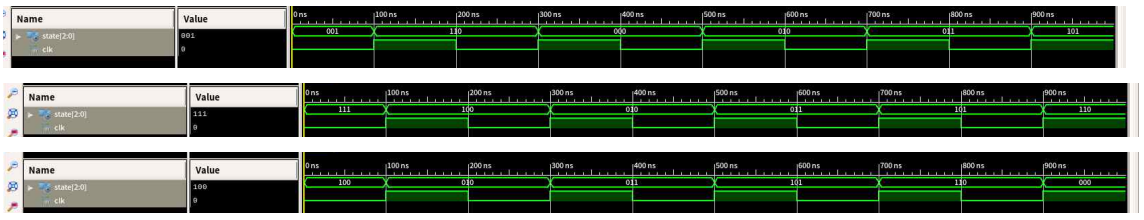
X	$\overline{B}\,\overline{C}$	$B\overline{C}$	BC	$\overline{B}C$
$\overline{A}$	0	X	1	0
A	X	1	X	0

Y	$\overline{B}\,\overline{C}$	$B\overline{C}$	BC	$\overline{B}C$
$\overline{A}$	1	X	0	1
A	X	1	X	0

Z	$\overline{B}\,\overline{C}$	$B\overline{C}$	BC	$\overline{B}C$
$\overline{A}$	0	X	1	1
A	X	0	X	0

이로부터 $X = C$, $Y = \overline{B} + \overline{A}\,\overline{C}$, $Z = \overline{A}B$라는 식을 얻을 수 있었으며, 이는 Don't care 상태에서 벗어나지 못하게 하지 않는다.

구현은 Structural하게 하였으며, 코드와 시뮬레이션 결과는 다음과 같다. 각각의 시뮬레이션은 초기 상태를 001,100, 111로 한 것이다.




```
20 ////////////////////////////////////////////////
21 module nextstate(
22     input [2:0] state_init,
23     output [2:0] state_out
24     );
25
26 wire A, B, C;
27
28 assign A = state_init[2];
29 assign B = state_init[1];
30 assign C = state_init[0];
31
32 assign state_out[0] = (!A)&B;
33 assign state_out[1] = (!B)|((!A)&(~C));
34 assign state_out[2] = C;
35
36
37 endmodule
38
```

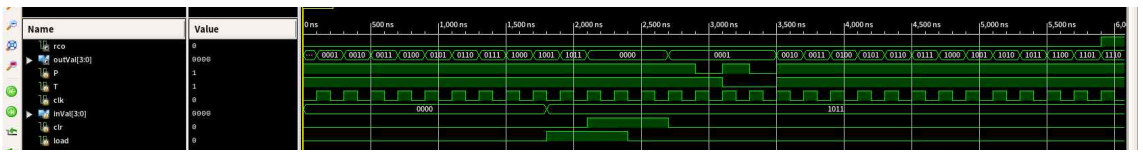
```
20 ////////////////////////////////////////////////
21 module counter1(
22     input clk,
23     output [2:0] state
24     );
25
26 wire[2:0] D,Q,nD;
27 nextstate ns1(state,nD);
28
29 D_FlipFlop1 Dflip2(nD[2],clk,state[2]);
30 D_FlipFlop1 Dflip1(nD[1],clk,state[1]);
31 D_FlipFlop1 Dflip0(nD[0],clk,state[0]);
32
33
34
35 endmodule
```

실습2. 4-bit Catalog Counter 구현

clear가 load보다 우선순위가 높고, clear와 load가 0인 경우에는 P와 T가 모두 1인 경우에면 count+1 값을 반환하게 4-bit Catalog Counter를 구현하였다.

코드와 시뮬레이션 결과는 아래와 같다.

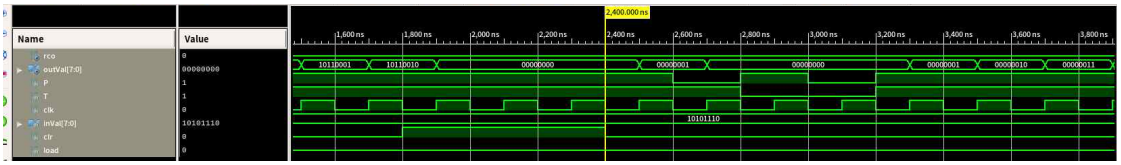
```
20 //////////////////////////////////////////////////////////////////////////////////
21 module fourBitCounter(
22 input P,
23 input T,
24 input clk,
25 input [3:0] inVal,
26 input clr,
27 input load,
28 output rco,
29 output [3:0] outVal
30     );
31
32 wire [3:0] plusOne, nextVal;
33 wire [3:0] one;
34 assign one = 1;
35
36 assign plusOne = outVal + one;
37
38 assign nextVal[3] = ((!clr)&((load&inVal[3])|((!load)&((P&T&plusOne[3]) | ((~P)&(~T)&(outVal[3])) ))));
39 assign nextVal[2] = ((!clr)&((load&inVal[2])|((!load)&((P&T&plusOne[2]) | ((~P)&(~T)&(outVal[2])) ))));
40 assign nextVal[1] = ((!clr)&((load&inVal[1])|((!load)&((P&T&plusOne[1]) | ((~P)&(~T)&(outVal[1])) ))));
41 assign nextVal[0] = ((!clr)&((load&inVal[0])|((!load)&((P&T&plusOne[0]) | ((~P)&(~T)&(outVal[0])) ))));
42
43 D_FlipFlop0 rcoooo((nextVal == 4'b1111), clk, rco);
44 D_FlipFlop0 Dflip3(nextVal[3],clk,outVal[3]);
45 D_FlipFlop0 Dflip2(nextVal[2],clk,outVal[2]);
46 D_FlipFlop0 Dflip1(nextVal[1],clk,outVal[1]);
47 D_FlipFlop0 Dflip0(nextVal[0],clk,outVal[0]);
48
49
50 endmodule
51
```

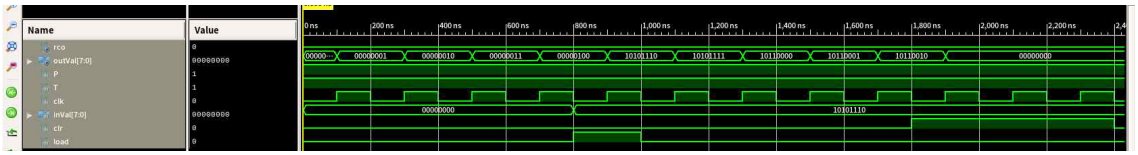


실습3. 8-bit Catalog Counter 구현

실습2에서 구현한 4-bit Catalog Counter를 2개를 이어 붙여서 8-bit Catalog Counter를 구현하였다. High 4-bit Catalog Counter의 P와 T값은 Low 4-bit Catalog Counter의 rco값을 넣어주어 구현하였다. 이렇게 해야 상위 4bit의 counter가 동작할 때만 +1가 되게 할 수 있다.

코드와 시뮬레이션 결과는 아래와 같다.






```
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module eightBitCounter(
22 input P,
23 input T,
24 input clk,
25 input [7:0] inVal,
26 input clr,
27 input load,
28 output rco,
29 output [7:0] outVal
30     );
31
32 wire rco_res;
33
34 fourBitCounter  fl(P, T             , clk, inVal[3:0], clr, load, rco_res, outVal[3:0]);
35 fourBitCounter  fh(rco_res, rco_res , clk, inVal[7:4], clr, load, rco    , outVal[7:4]);
36
37 endmodule
38
```