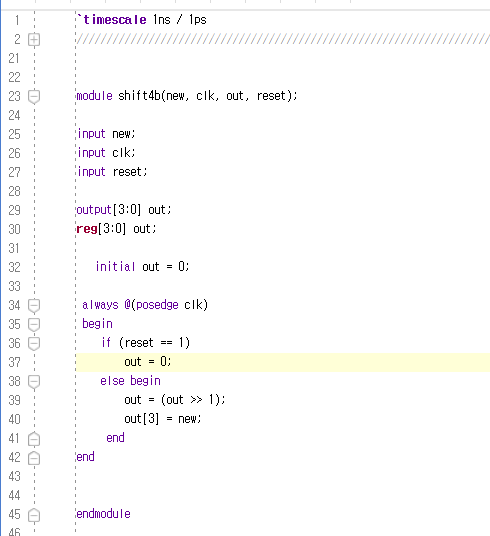
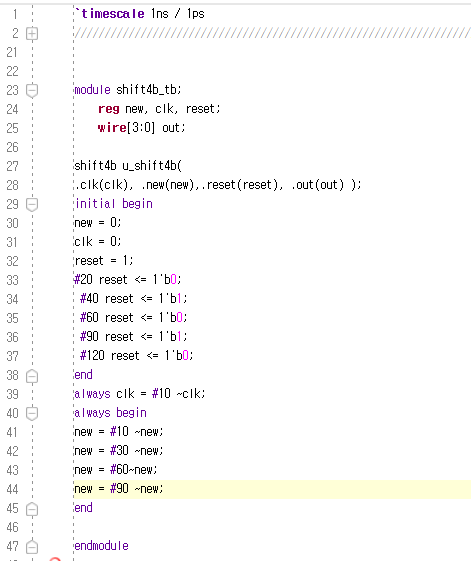
13주차 결과보고서

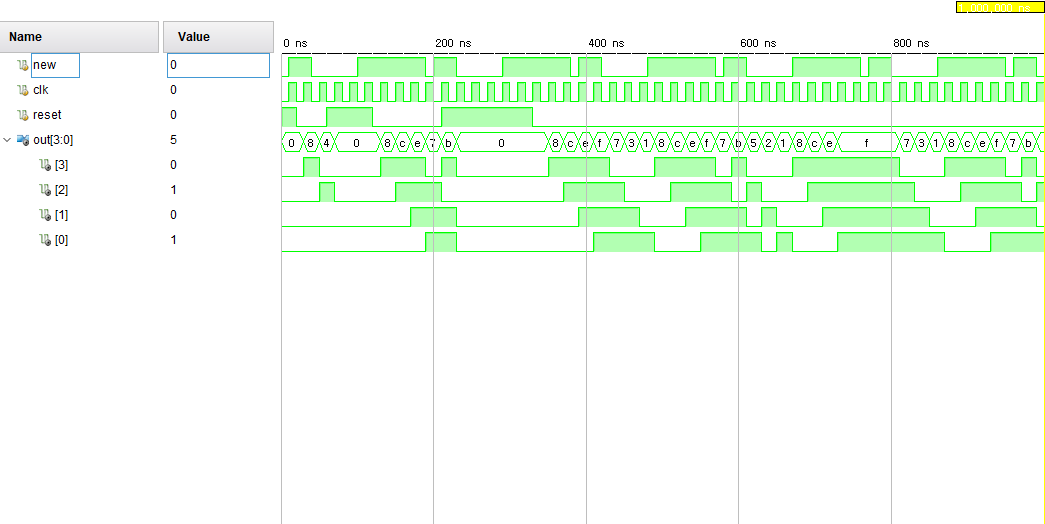
전공 : 아트앤테크놀로지 학년 : 3학년 학번 : 20191098 이름 : 백승주

1. 4bit Shift register는 네 개의 플립 플롭을 연결해 4비트의 정보들을 클럭에 따라 계속 왼쪽이나 오른쪽으로 1비트씩 옮겨 저장하는 회로로 비트를 오른쪽으로 옮겨가게 한 4bit Shift register의 코드를 구성하면 다음과 같다.  
   

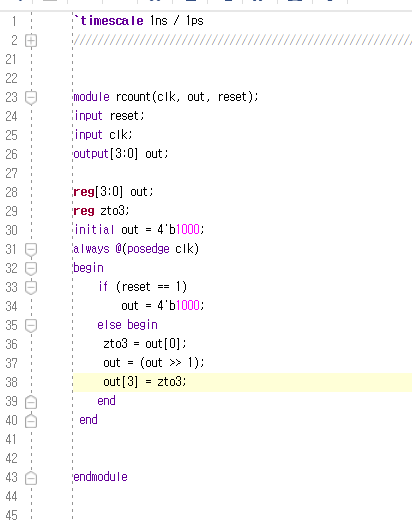
위 코드에서는 if else 문을 이용해 reset이 1일 경우 결과값이 0으로 초기화하도록 했고 reset이 1이 아니라면 out[3], out[2], out[1], out[0]의 값들이 out>>1 에 따라 한 칸씩 오른쪽으로 이동하게 된다. 오른쪽으로 이동한다면 맨 왼쪽의 값인 out의 값을 다시 할당해야 한다. 이 out[3]에는 회로의 input값인 new를 할당해준다. 즉 만일 out값들이 0001이고 새로운 new값이 계:속 1이었다면 0001, 1000,1100, 1110, 1111 과 같은 식으로 out 값들이 변화해갈 것이다. 이에 따른 testbench 코드는 다음과 같다.



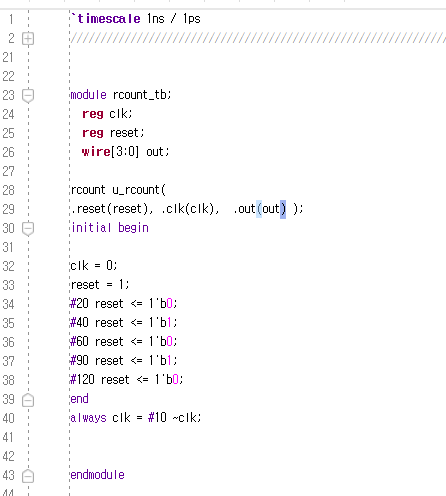
위 코드에서 reset 값을 변화시키며 reset 값에 따른 변화를 관찰하고자 했고 new값 역시 일정하게 변화를 줘 그 변화를 관찰하고자 했다. 이에 따른 시뮬레이션 결과는 다음과 같다.

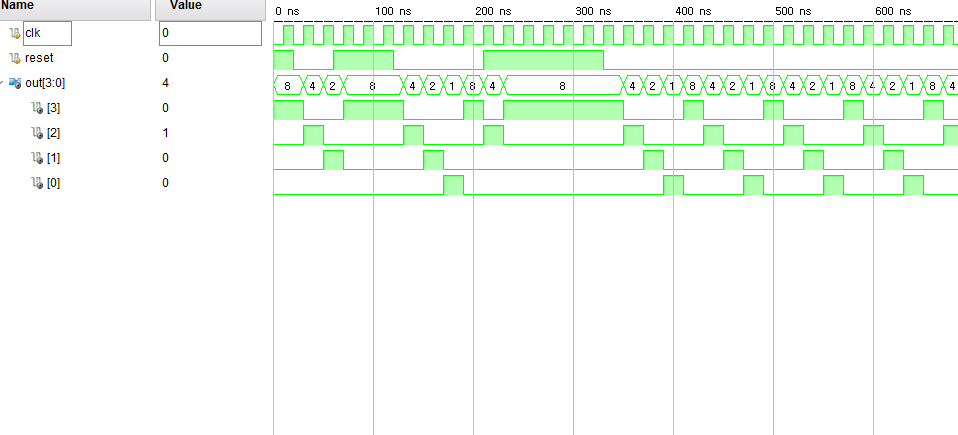


위 시뮬레이션 결과를 살펴보면 먼저 reset값이 1일 때는 초기화되어 out 값이 0, 즉 0000의 결과를 출력하는 것을 확인할 수 있다. 그리고 reset이 0이 아닐 때 new값이 0일 때는 7 ->3 -> 1, 즉 0111 -> 0011 -> 0001 로 계속 오른쪽으로 shift 되고 MSB에는 new 값인 0이 들어가고 있음을 확인할 수 있다.   
new값이 1일 때 맨 처음 1000, 즉 8일 때에는 값들이 오른쪽으로 시프트되고 MSB에 1이추가되어 1100, 즉 c를 출력하는 것을 확인할 수 있고 1100에서 다시 같은 계산이 반복되어 1110, 즉 e를 출력하고 있는 것을 확인할 수 있다. 마지막으로 1110, 즉 e에서 new가 1일 때 다시 같은 과정을 반복하여 1111, 즉 f값을 출력하는 것을 시뮬레이션 결과에서 확인할 수 있었다.   
 또한 이러한 out 값들의 변화는 positive edge에서 나타난 것 또한 확인할 수 있었다.

1. Ring counter는 마지막 플립 플롭의 값이 맨 처음 플립 플롭의 입력값이 되는 형태로 앞선 shift register와 그 구조는 비슷하다. 맨 처음 플립 플롭의 입력값을 따로 input해줄 필요 없이 순환하는 형태인 ring counter의 Verilog 코드는 다음과 같다.  
     
   

위 코드는 앞선 shift register 코드와 상당히 유사한데 input 값이 없는 대신 zto3 변수를 통해 배열의 LSB값을 임시로 저장한 뒤 배열의 MSB에 저장한다는 점과 초기화할 때 0이아닌 1000으로 초기화한다는 점이 다르다.   
zto3 = out[0] 을 통해 배열의 첫 번째 값을 zto3 변수에 저장하는데 이후 shift 작업을 할 예정이라 현재의 out[0]은 지워지기 때문이다. Shift한 이후 이전에 저장해두었던 zto3을 통해 out[3]=zto3로 이전 out[0] 값을 out[3]으로 전달하게 되어 결과적으로 이 회로는 순환하는 ring counter가 된다. 단, 이때 reset이 1일 때 reset 값은 0이 아닌 1000으로 설정했는데 그 이유는 이 회로는 순환 형태이기 때문에 reset으로 0000이 된다면 아무리 shift를 하며 순환해도 항상 0000이라 그 변화를 관찰할 수 없게 되기 때문이다. 이에 따른 testbench코드는 다음과 같다.

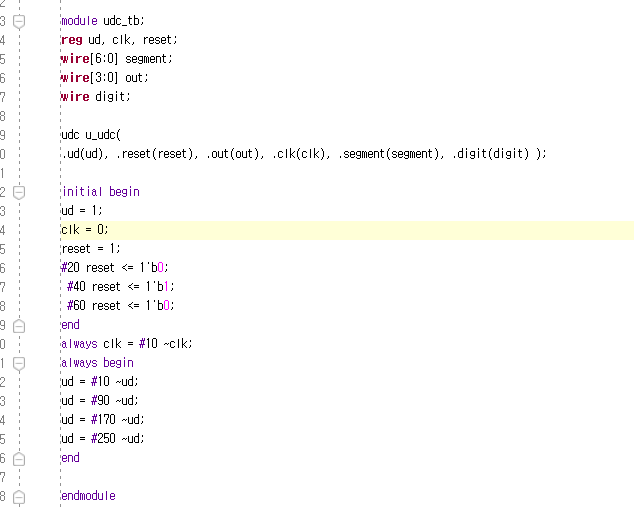
  
앞선 shift register와 다르게 input 값이 필요가 없으므로 클럭 값을 제외하고reset 값만 변화를 줘 그 변화를 관찰하고자 했다. 그 시뮬레이션 결과는 다음과 같다.

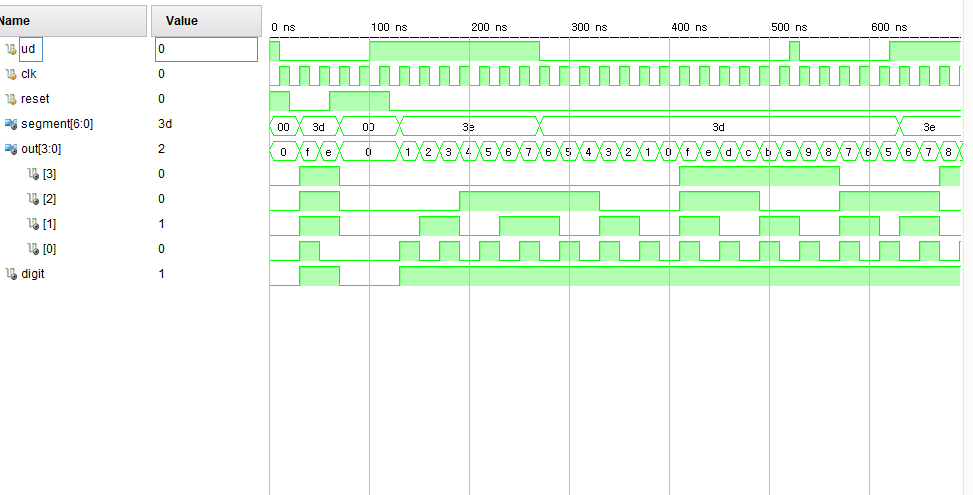


위 시뮬레이션 결과를 살펴보면 먼저 reset 했을 경우 설정한대로 out값이 8, 즉 1000으로 초기화된 걸 확인할 수 있다. Reset 이후 out 값의 변화를 살펴보면 8 -> 4 -> 2 -> 1 -> 8 -> 4 -> 2 -> 1 로 8, 4, 2, 1 이 계속 반복되는 것을 확인할 수 있는데 이를 자세히 살펴보면 1000, 0100, 0010, 0001 의 값들이 순서대로 반복되는데 이 값들은 오른쪽으로 shift된 결과라는 것을 알 수 있다.

1. 4 비트 up/down counter는 입력 값이 1이면 up counter로 인식해 상태에 계속 1을 더하고 입력 값이 0 이면 down counter로 인식해 상태에 계속 1을 빼게 한 counter다. 이러한 up/down counter의 Verilog 코드는 다음과 같다.  
   

위 코드에서 segment 변수들은 fpga 상에서 현재 up count를 하고 있는지, down count를 하고 있는지 나타내기 위해 설정했다. 0111110은 U를 나타내고 0111101은 D를 디스플레이를 나타낸다. 위 소스는 하나의 input인 ud값을 가지고 있는데 이 ud 값이 1이라면 up count 라는 뜻이고 0이라면 down count라는 뜻이다. 따라서 ud ==1 일 때 if문을 이용해 out < = out+1을 통해 out값에 1을 더하고 ud == 0 이라면 else문을 이용해 out <= out -1을 통해 out 값에 1을 빼준다. Reset 값이 1일 때는 0으로 초기화되도록 했다. 위 코드에 대한 testbench 코드와 시뮬레이션 결과는 다음과 같다.





위 시뮬레이션에서 ud = 1일 때 out 값들의 변화를 살펴보면 0001, 0010, 0011, 0100, 0101, 0110, 0111, 다시 말해 1, 2, 3, 4, 5, 6, 7 로 1씩 증가하고 있는 것을 확인할 수가 있다. 이러한 증가는 ud 값이 1에서 0으로 바뀌며 사라지는데 ud 값이 0으로 되면 증가 대신 out 값이 오히려 감소되고 있는 것을 확인할 수 있다. 이 때 out 값은 7부터 0까지 다시 말해 0111에서 0000까지 계속해서 감소하며 0 다음에는 1111, 즉 f값으로 넘어간 후 다시 감소하는 것을 확인할 수 있다. 그 외에도 reset 값이 1일 때는 이전 값에 관계없이 out 값이 0으로 초기화된 것을 확인할 수 있었다.

1. 실습을 통해 4 bit shift Register, 4 bit Ring counter, 4 bit Up down counter의 소스 코드를 구현하고 그 시뮬레이션 결과들을 확인했다. 이 세 회로는 그 원리나 소스 코드가 비슷했으나 차이점이 존재했는데 먼저 4 bit shift Register는 결과값들을 계속 오른쪽으로 shift시키고 맨 왼쪽의 비트에는 input 값을 집어넣는 회로였다. 4 bit Ring counter는 출력값이 계속 오른쪽으로 shift 시킨 결과라는 점에서는 앞선 회로와 동일하지만 맨 왼쪽의 비트에 input 값 대신 맨 오른쪽의 비트를 집어넣는다는 점에서 차이를 보였다. 4 bit Up down counter의 경우 shift 시키는 대신 1씩 증가시키거나 감소시켰다. 입력값이 1일 경우에는 up으로 받아들여 값들을 증가시켰고 입력값이 0일 경우에는 down으로 받아들여 감소시켰다.

  
7 segment로 숫자뿐만 아니라 알파벳 역시 나타낼 수가 있다. 직관적으로 알아볼 수 있는 숫자들과는 달리 알파벳의 경우에는 7 segment로 나타냈을 때는 일반적으로 한 번에 알아볼 수 없는 알파벳들도 존재한다. 만일 7 segment로 모든 알파벳들을 나타낸다면 이러한 모양들에 대해서도 숙지해야될 것이다. 이번에 나타내고자 한 u와 d의 경우 모양이 직관적이기 때문에 바로 0111110와 0111101로 표현할 수 있었다. A부터 F까지의 알파벳은 자주 쓰이는 16진수를 나타내기 위해서는 숙지할 필요성이 있다. A부터 F까지의 2진수 값들은 다음과 같다.  
A : 0111 1111

B : 0111 1100

C : 0101 1000

D : 0101 1110

E: 0111 1001

F : 0111 0001