**<Full Adder>**

1)full adder code

텍스트이(가) 표시된 사진

자동 생성된 설명

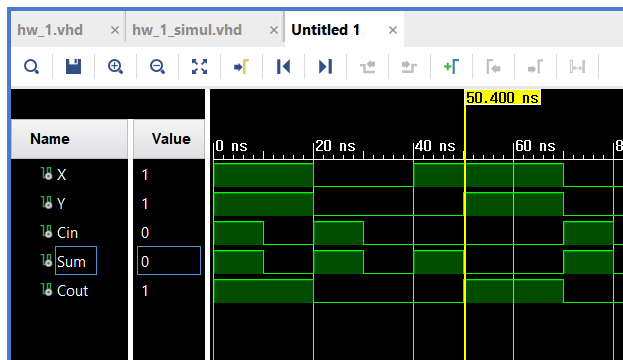
테이블이(가) 표시된 사진

자동 생성된 설명설계 핵심 : sum과 cout의 식을 도출하여 architecture 내부에 코드 작성

설계 방법 : 입출력을 정의해주고 architecture 내부에 sum과 cout의 식을 각각 작성하였다. x, y, cin의 값이 변할 때마다 해당하는 statement가 실행되면서 sum과 cout의 값이 변하도록 설계하였다. 하지만 여기에서는 2개의 statement에 x,y,cin이 모두 존재하기 때문에 입력 중 하나가 바뀌더라도 2개 모두 실행된다.

Full adder 동작 원리 : 1 bit의 x,y,cin이 입력되었을 때 그 값에 따라 sum과 cout이 바뀌도록 동작한다. 예를 들어 x=0, y=1, cin=1이면 sum=0이고, cout=1이 된다. 이처럼 총 2\*2\*2=8가지의 경우에 따라 출력 값이 변화한다. (위의 표에서 확인 가능하다.)

2) full adder simulation 결과



Process문 때문에 wait statement가 나오기 전까지 초기화가 진행된다. 따라서 X=0, Y=1로 처음에 초기화를 시켰더라도 X <= '1', Y <= '1'로 초기화가 된다.(Cin=’1’) 10ns까지 X=1, Y=1로 유지되는 것을 확인할 수 있다.

그 다음 Cin=0이 되었고,

sum <= cin xor x xor y;

cout <= (x and y) or (x and cin) or (y and cin); 은

concurrent statement이기 때문에 Cin의 값이 변하면 sum과 cout의 값이 바로 변한다. 따라서 Cin=0이 되지만 Sum=0, Cout=1로 값이 바뀐다. 그리고 10ns 동안 기다리고 X=0, Y=0, Cin=1의 값이 입력되어 이 값에 따라 Sum과 Cout의 값이 바뀌게 된다. (Cin, X, Y에 따라 Sum과 Cout의 값이 바로 변한다.)

이 원리를 적용하여 각각의 경우에 대해 생각해보면

X=1 Y=1 Cin=1 ->Sum=1, Cout=1

X=1 Y=1 Cin=0 ->Sum=0, Cout=1

X=0 Y=0 Cin=1 ->Sum=1, Cout=0

X=0 Y=0 Cin=0 ->Sum=0, Cout=0

X=1 Y=0 Cin=0 ->Sum=1, Cout=0 이라는 결과가 나타난다는 것을 확인할 수 있다.

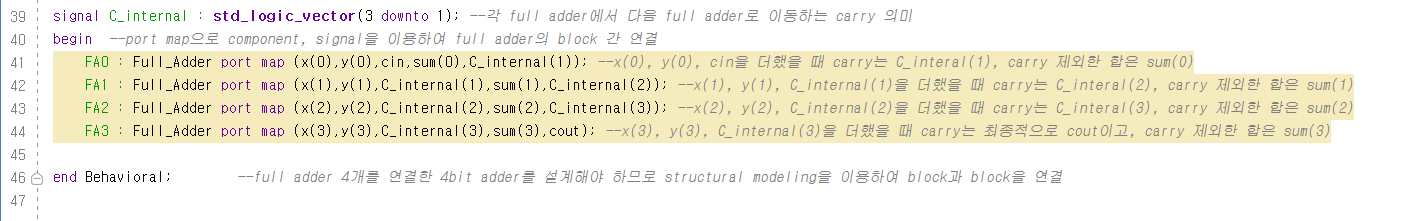
한 번의 process가 끝나게 되면 다시 process 처음으로 돌아가 실행되므로 50ns 이후에도 시뮬레이션이 되는 것을 알 수 있다.

**<4 bit adder>**

1)4 bit adder code

텍스트이(가) 표시된 사진

자동 생성된 설명



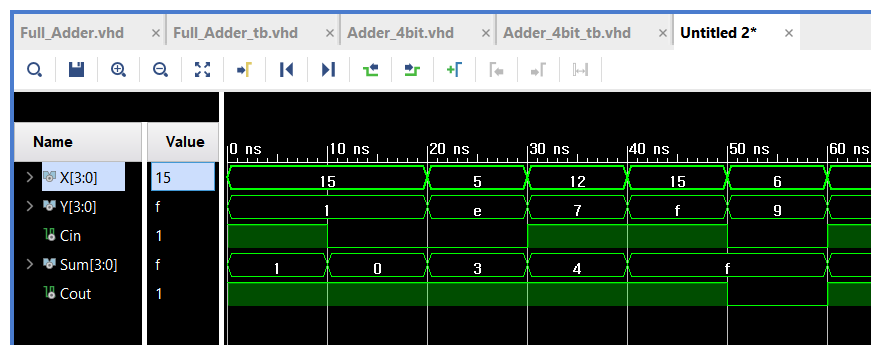
설계 핵심 : 앞서 설계한 Full adder 4개를 structural modeling을 이용하여 Full adder 간 block을 연결한 것

설계 방법 : 4 bit adder는 똑같은 full adder 4개를 직렬로 연결하여 structural model로 설계했다. 즉 full adder를 component로 생각하고 설계한 것이다. 여기서는 C\_internal이라는 signal이 각 block의 연결고리 역할을 한다고 할 수 있다.

4 bit adder 동작 원리 : 4 bit의 x,y 입력과 1bit의 cin 입력에 값이 입력되면 FA는 concurrent statement으로 설계되었기 때문에 바로 4bit의 두 수의 덧셈 결과가 나오는 것이 아니라 우선 각 FA에서의 입력, x(n), y(n), cin 혹은 C\_internal(n)에 의해 carry와 sum의 결과가 나타난다.(이때 C\_internal의 값은 모두 0이고 n은 0~3의 숫자이다.) 그 결과에 의해 FA가 concurrent하게 동작하므로(event 발생에 의해 동작) 다른 FA의 C\_internal 값이 변하게 되면 그 FA가 덧셈을 하게 되고 그 FA의 carry와 sum의 결과를 바꾼다. 또한 이에 의해 바뀐 C\_internal의 값에 의해 바로 앞에서 했던 동작과 똑같이 동작하고 이것이 계속 반복된다.(4bit adder 코드에서 FA0, FA1, FA2, FA3이 concurrent statement) 그리고 더 이상 C\_internal 값의 변화가 없을 때 비로소 4bit 덧셈 결과를 얻을 수 있다.

\*시뮬레이션은 최종 덧셈 결과만 나타난다.

2) Adder\_4bit simulation



\*wait for 10ns이므로 시뮬레이션 결과는 10ns 동안 유지된다.

처음 process문 때문에 X=1111, Y=0001, Cin=1로 초기화되고 10ns동안 기다린다.

그때 X(1111)+Y(0001)+Cin(1)-> Cout=1, Sum=0001(1)로 concurrent statement에 의해 출력값이 변화한다.

그 다음 Cin에만 0을 넣어줬기 때문에

X(1111)+Y(0001)+Cin(0)-> Cout=1, Sum=0000(0)이다.

계속 같은 방식으로 계산을 해주면

X(0101)+Y(1110)+Cin(0)-> Cout=1, Sum=0011(3)

X(1100)+Y(0111)+Cin(1)-> Cout=1, Sum=0100(4)

X(1111)+Y(1111)+Cin(1)-> Cout=1, Sum=1111(15)

X(0110)+Y(1001)+Cin(0)-> Cout=0, Sum=1111(15)이고

이는 시뮬레이션 결과와 완벽히 일치하는 것을 확인할 수 있다.

마찬가지로 process문이 끝났으므로 다시 process 처음으로 돌아가서 시뮬레이션이 진행된다.

**<고찰>**

1)vivado 사용 미숙

Full adder의 코드와 testbench 코드를 작성하여 시뮬레이션을 돌리고, 후에 소스를 추가하여 4 bit adder의 코드와 testbench 코드를 작성하여 시뮬레이션을 돌렸으나 full adder의 시뮬레이션만 보여졌다. 이는 Adder\_4bit\_tb를 우클릭하여 set as top을 선택하여 해결할 수 있었다.

2) 여러 오류

텍스트이(가) 표시된 사진

자동 생성된 설명

std\_logic\_vector를 사용했어야 했는데 std\_logic만 써서 오류가 발생하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

downto0을 띄어쓰기 안 해서 오류가 발생했다.