Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

#74

학생 : 김시윤

강사 : Innova Lee(이 상훈)

Verilog 를 이용한 디지털 시스템 설계

(논리회로를 숙지한 상태임을 가정하여 진행하였다.)

1.Verilog 를 이용한 조합회로 기술 방법

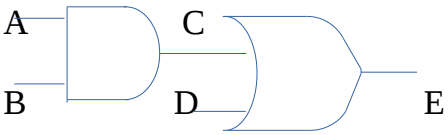
베릴로그는 동시진행문 혹은 연속 진행 지정문을 이용하여 조합회로의 동작을 모델링 한다.
And + Or 두개의 회로는 2 개의 베릴로그 문장으로 A,B,C,D,E 로 표현이 가능하다.

assign #5 C = A && B;
assign #5 E = C || D;

assign 은 값을 할당하는 데 사용한다.
#5 는 5 ns 의 지연을 갖고있다는 뜻이며 && 는 And 게이트를 말하고 || 는 Or 게이트를 의미한다.

출력 C = A and B 이고 출력 E = (A and B) or D 로 표현할 수 있다.

결국 위의 식을 그림으로 표현하면 다음과 같다.



1_1 . clock pulse 생성

assign #10 CLK = ~CLK;

10 ns 마다 반전하는 회로가 만들어진다.(여기서 주의할 점 베릴로그는 대소문자를 구별하기 때문에 대문자 소문자에 주의!!)

---

module and_gate   (A,B,C);
output C;

input A.B;

wire B[3:0];
wire C[3:0];
wire A[3:0];

assign C[3] = A[3] && B[3];
assign C[2] = A[2] && B[2];
assign C[1] = A[1] && B[1];

=

assign C = A & B;

```
assign C[0] = A[0] && B[0];
endmodule
```

전가산기 설계 예제

```
module Full_Adder( X, Y, Cin, Cout, Sum);
output Cout, Sum;

intput X, Y, Cin;

assign #10 Sum = X ^ Y ^ Cin;
assign #10 Cout = (X && Y) || (X && Cin) || (X && Cin);
endmodule
```

4 비트 전 가신기 일 경우

```
module Adder4 (S, Co, A,B,Ci);
output [3:0] S;
output Co;
input [3:0] A,B;
input Ci;

wire [3:1] C; // C= clock
```

나머지는 가산기 설계데로 하나하나 비트별 요소를 넣어주면 된다.

1_2 initial 과 always

```
initial
begin
        sequential-statements
end
```

always 문

```
always @(sensitivity-list)
        sequential-statements
end
```

순차적 할당방법.

Initial 은 시간이 0 인 순간에 한번 실행되는 것.

always 는 루프를 돌면서 실행된다.

```
Always @(A,B,C)
begin
        C = A && B;
        E = C || D;  // non block <=
```

end
A → B → D 순으로 진행한다.
위의 예제는 블로킹 문이다 A , B 가 끝나지 않으면 진행하지 않는다.

블로킹과 논블로킹 지정문

```
module sequential_module (A, B, C, D, Clk);
input Clk;
output A, B, C, D;
reg A, B, C, D;

always @(posedge Clk)
begin
        A = B;
        B = A;
end
/* blocking*/

always @(posedge Clk)
begin
        C <= D;
        D <= C;
end
/*non block*/

endmodule
```

always 블록을 이용한 플립플롭 모델링

J-K 플립플롭

```
module JKFF (SN, RN, J, K, CLK, Q, QN);
input SN, RN, J, K, CLK;
output Q, QN;

reg Qint;

always @(negedge CLK or RN or SN)
begin
        if(~RN)
        #8 Qint <= 0;
        else if (~SN)
        #8 Qint <= 1;
        else
        Qint <= #10 (( J && ~Qint) || (~K && Qint));
end

assign Q = Qint;
```

```
assign QN = ~Qint;

endmodule
```

Set 과 Reset 이 달린 JK 플립플롭의 모델링이다.
Set , Reset, ClockPulse 는 모두 Low Active 이다.

Clock 과 상관없이 Set , Reset 은 동작하기 때문에 Always 를 통해 항상 상태를 확인한다.
Reset 이 0 이면 출력 Q 는 0 이 되고 Set 이 0 이되면 출력 Q 는 1 이 된다.
둘다 1 일 경우 CLK 가 Low 일 때만 동작하게 된다.

JK 플립플롭의 일반화 식은

Q = J and ~Q or ~K and Q 이므로 CLK 에 의해 동작하는 if 문에 대입한다.

위의 플립플롭 예제를 통해 쉬운 D 플립플롭을 설계해본다.

우선 설계하기 전에 입력과 출력을 정해준다.
D 플립플롭의 동작은 Low Active 로 설정한다.
Clock , Set , Reset 이 존재하며
출력은 Q , ~Q 두개의 출력을 가진다.

진리표를 그리면 다음과 같다.

Clock Pulse	Set	Reset	D	Q	~Q
High	0	1	1	1	0
High	1	0	0	0	1
Low	0	0	0	0	1
Low	0	0	1	1	0

```
module D_FF (SN, RN, D, Q, QN, CLK);

input SN, RN, D, CLK;
output Q, QN;

reg Qint;

always @(negedge CLK or SN or RN)
begin
        if(~RN)
        #8 Qint <= 0;
        else if(~SN)
        #8 Qint <= 1;
        else
```

```
        #10 Qint <= D;

end

assign Q = Qint;
assign QN = ~Qint;

endmodule
```

/* 여기서 시간지연을 어떤상황에 얼만큼 해야하는지 모르겠다.
우리가 사용하는 보드에 어처피 클럭 주파수가 존재하는데 설정을 해줘야하는지
의문이다. */