3주차 결과보고서

전공: 경영학과 학년: 4학년 학번: 20190808 이름: 방지혁

1.
우선 첫 단계로는 design entry로 설계하는 것입니다. 크게 2가지가 가능한데, 하나는 HDL 언어를 사용하여 설계하는 것이고, 또다른 한 가지는 스케메틱으로 설계를 하는 것입니다. HDL 언어를 사용하는 방법에서는 design source code와 testbench 코드를 작성합니다. design source code는 구조적 설계의 측면으로 회로의 구조를 기술하는 것입니다. testbench 코드는 하드웨어가 어떠한 과정을 거쳐 동작해야 하는지 기술합니다.

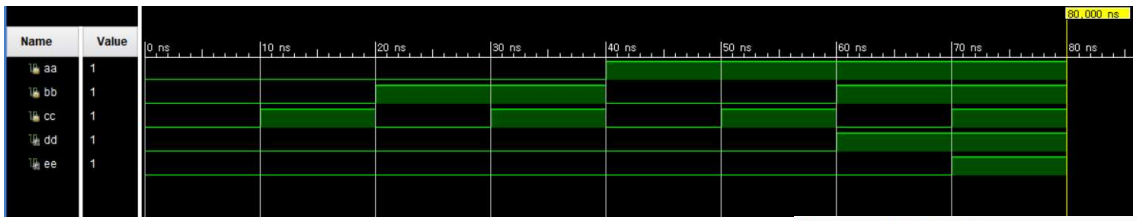
두 번째 단계로는 RTL simulation을 하는 것입니다. 이전의 단계에서 설계한 회로를 시뮬레이션하여 제대로 동작하는지 확인합니다. FPGA 보드에 직접 넣기보다는 이 방법이 시간이 덜 소요되고 디버깅이 간단하며, 이를 통해 논리적인 오류나 시간 지연 등을 조정할 수 있습니다.

세 번째 단계는 합성(Synthesis)입니다. 해당 과정은 사용자가 HDL로 작성된 코드를 FPGA 보드에서 사용할 수 있는 Low Level 수준의 netlist로 변환합니다. 이 과정을 통해 논리 회로가 netlist 파일로 저장될 수 있으며 매핑 준비가 완료됩니다.

네 번째 단계는 배치 및 배선(Place & Route)입니다. FPGA 내부의 합성된 각 논리 블록들이 실제로 어떻게 FPGA 내부의 위치들과 연결될지를 정의합니다. 이 과정에서 FPGA의 전력 소비 최소화, 지연 시간 감소 같은 최적화가 이루어집니다.

마지막 단계는 비트스트림 생성 (Bitstream Generation) 및 다운로드입니다. 우선 완료된 설계를 FPGA에 탑재할 수 있도록 변환합니다. 이 파일을 비트스트림(Bitstream)이라고 합니다. FPGA 내부의 각 구성 요소가 어떻게 설정될지를 정의한 정보가 포함되어 있습니다. 최종적으로 FPGA에 비트스트림을 다운로드하여 구현합니다. 이외 추가할 수 있는 단계로는 최종 검증과 유지보수가 있을 수 있습니다.

2.



Input A	Input B	Input C	Output D	Output E
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

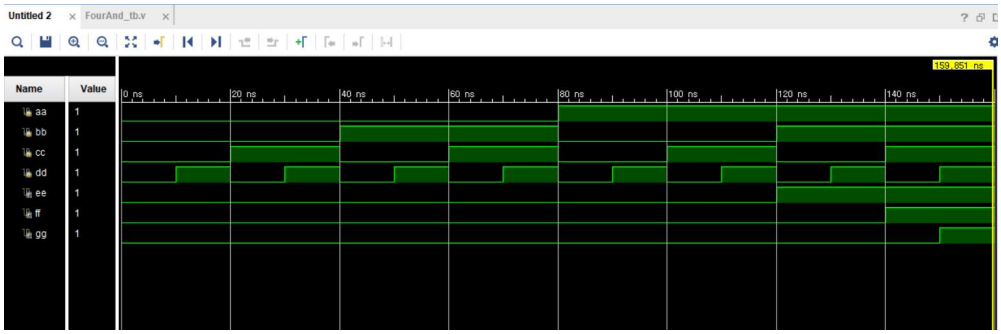
```
`timescale 1ns / 1ps

module ThreeAnd(
  input  a, b, c,
  output d, e
  );
  assign d = a&b;
  assign e = c&d;

endmodule
```

```
`timescale 1ns / 1ps

module ThreeAnd_tb;

reg aa, bb, cc;
wire dd, ee;

ThreeAnd u_test(
.a(aa),
.b(bb),
.c(cc),

.d(dd),
.e(ee)
);

initial begin
  aa = 1'b0;
  bb = 1'b0;
  cc = 1'b0;
  end

  always@(aa or bb or cc) begin
  aa <= #40 ~aa;
  bb <= #20 ~bb;
  cc <= #10 ~cc;
  end

  initial begin
  #80
  $finish;
  end


endmodule
```

3 input 및 2 output AND gate의 simulation 사진이 가장 상위의 것이며, simulation 하단에 있는 table이 truth table, 우측에 있는 코드는 design source, 가장 하단에 있는 것이 testbench입니다.

truth table에서 볼 수 있듯이 모든 input이 1이 되어야 최종 output이 1이 되는 것을 확인할 수 있습니다.

3.



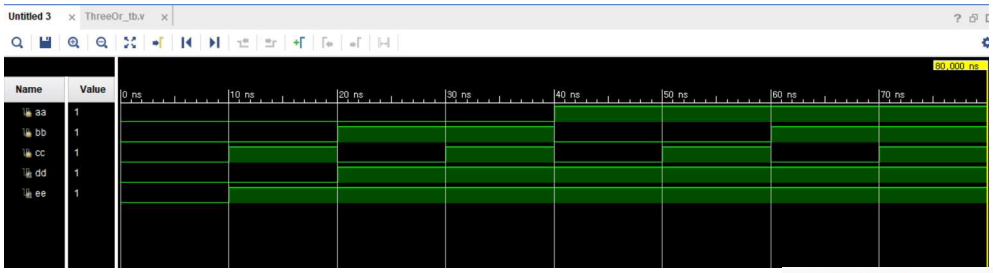
Input A	Input B	Input C	Input D	Output E	Output F	Output G
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	1	0
1	1	1	1	1	1	1

```
`timescale 1ns / 1ps

module FourAnd(
input a, b, c, d,
output e, f, g
    );

assign e = a&b;
assign f = c&e;
assign g = d&f;

endmodule
```

```
`timescale 1ns / 1ps

module FourAnd_tb;
reg aa, bb, cc, dd;

wire ee, ff, gg;

FourAnd u_test(

.a (aa),
.b (bb),
.c (cc),
.d (dd),

.e (ee),
.f (ff),
.g (gg)
);

initial begin
aa = 1'b0;
bb = 1'b0;
cc = 1'b0;
dd = 1'b0;
end

always@(aa or bb or cc or dd) begin
aa <= #80 ~aa;
bb <= #40 ~bb;
cc <= #20 ~cc;
dd <= #10 ~dd;
end

initial begin
#160
$finish;
end
endmodule
```

4 input 및 3 output AND gate의 simulation 사진이 가장 상위의 것이며, simulation 하단에 있는 table이 truth table, 좌측에 있는 코드는 design source, 우측에 있는 것이 testbench입니다. truth table에서 볼 수 있듯이 모든 input이 1이 되어야 최종 output g가 1이 되는 것을 확인할 수 있습니다.

4.



Input A	Input B	Input C	Output D	Output E
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

```
`timescale 1ns / 1ps

module ThreeOr(
  input  a, b, c,
  output d, e
  );
  assign d = a|b;
  assign e = c|d;

endmodule
```

```
`timescale 1ns / 1ps
module ThreeOr_tb;

reg aa, bb, cc;
wire dd, ee;

ThreeOr u_test(
.a(aa),
.b(bb),
.c(cc),

.d(dd),
.e(ee)
);

initial begin
  aa = 1'b0;
  bb = 1'b0;
  cc = 1'b0;
  end

  always@(aa or bb or cc) begin
  aa <= #40 ~aa;
  bb <= #20 ~bb;
  cc <= #10 ~cc;
  end

  initial begin
  #80
  $finish;
  end


endmodule
```

3 input 및 2 output OR gate입니다. simulation 사진이 가장 상위의 것이며, simulation 하단에 있는 table이 truth table, 좌측에 있는 코드는 design source, 우측에 있는 것이 testbench입니다.

d는 a와 b에 OR, e는 c와 d에 OR을 취한 것입니다. truth table에서 볼 수 있듯이 단 하나의 input만 1이여도 최종 output E가 1이 되는 것을 확인할 수 있습니다.

5.

Input A	Input B	Input C	Input D	Output E	Output F	Output G
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	1
0	1	0	0	1	1	1
0	1	0	1	1	1	1
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

```
`timescale 1ns / 1ps

module FourOr(
input a, b, c, d,
output e, f, g
);

assign e = a|b;
assign f = c|e;
assign g = d|f;

endmodule
```

```
`timescale 1ns / 1ps

module FourOr_tb;

reg aa, bb, cc, dd;
wire ee, ff, gg;

FourOr u_test(
.a(aa),
.b(bb),
.c(cc),
.d(dd),

.e(ee),
.f(ff),
.g(gg)
);

initial begin
aa = 1'b0;
bb = 1'b0;
cc = 1'b0;
dd = 1'b0;
end

always@(aa or bb or cc or dd) begin
aa <= #80 ~aa;
bb <= #40 ~bb;
cc <= #20 ~cc;
dd <= #10 ~dd;
end

initial begin
#160
$finish;
end

endmodule
```

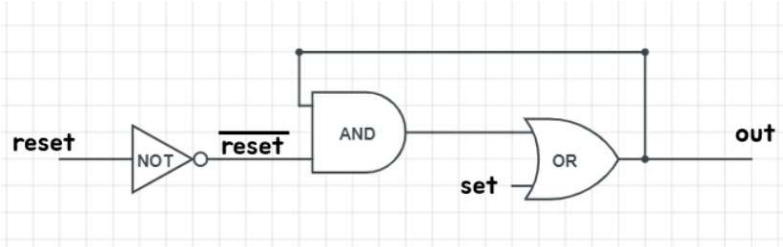
4 input 및 3 output OR gate입니다. simulation 사진이 가장 상위의 것이며, simulation 하단에 있는 table이 truth table, 좌측에 있는 코드는 design source, 우측에 있는 것이 testbench입니다. e는 a와 b에 OR, f는 c와 e에 OR, g는 d와 f에 OR을 취한 것입니다. truth table에서 볼 수 있듯이 단 하나의 input만 1이여도 최종 output g가 1이 되는 것을 확인할 수 있습니다.

6.

성공적이었지만, 강의 자료(B)란이 아닌 (A)란의 Schematic를 확인했을 때 PPT의 과제 란에 주어진 회로도와 조금씩은 차이가 존재하는 것을 확인할 수 있었습니다. input 3개의 게이트보다 2개의 게이트로 구성한다는 점이었는데, fan-out 및 propagation time 이슈 때문이 아닐까 예상해보았습니다. 또한, 효율적인 회로를 vivado에서 구성할 수 있다는 점을 알게 되었습니다.

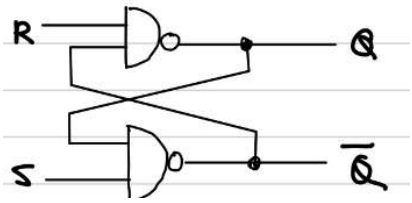
7.

NOT, AND, OR 게이트를 사용하여 만들 수 있는 래치에 대해 생각해보았습니다.



입력에는 set과 reset이 존재합니다. OR 게이트를 통과하여 그 output이 AND 게이트의 입력으로 사용됩니다. 이것을 피드백이라고 합니다, 진리표는 다음과 같이 나타낼 수 있겠습니다.

set	reset	output
0	0	1
0	1	0
1	0	1
1	1	1



set	reset	output
0	0	1
0	1	0
1	0	0
1	1	0

또한, 이처럼 SR latch에 대해 고민 해볼 수 도 있습니다. NAND 게이트로 구성되었으며 두 input이 모두 0일 때만 output(Q)이 1이 나옵니다.