4주차 결과보고서

전공: 경영학과 학년: 4학년 학번: 20190808 이름: 방지혁

1.

해당 실험의 목적은 NAND, NOR, XOR, AOI 게이트의 개념과 특징에 대해 이해하는 것입니다. 이를 위해 Verilog 언어를 이용해 직접 구현해보면서 결과를 검토해봅니다. 또한, 앞서 배운 AND, OR, NOT 게이트와 앞서 언급한 게이트들과의 관계에 대해서도 이해합니다.

무엇보다도, Verilog를 통해 설계한 논리 회로를 합성하고, 비트스트림을 생성한 후, 이를 FPGA 보드에 직접 올리는 과정을 거칩니다. 이 과정을 통해 시뮬레이션에서 그치지 않고 실제 하드웨어 구현까지의 전 과정을 경험할 수 있습니다. FPGA 보드에 회로를 올려 직접 스위치 동작을 통해 확인하면서, 설계한 논리 회로가 하드웨어적으로 어떻게 구현되고 동작하는지 경험합니다.

2.


```
`timescale 1ns / 1ps

module FourNand_b(
  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 FourNand_b_tb;

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

FourNand_b 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
```

Input a	Input b	Input c	Input d	Output e	Output f	Output g
0	0	0	0	1	1	1
0	0	0	1	1	1	0
0	0	1	0	1	0	1
0	0	1	1	1	0	1
0	1	0	0	1	1	1
0	1	0	1	1	1	0
0	1	1	0	1	0	1
0	1	1	1	1	0	1
1	0	0	0	1	1	1
1	0	0	1	1	1	0
1	0	1	0	1	0	1
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	0	1	0
1	1	1	0	0	1	1
1	1	1	1	0	1	0

우선 과정을 요약하자면, output E는 input A와 input B의 nand 값이며, output F는 C와 E의 nand 값이고, output G는 D와 F의 nand 값입니다. 이 nand 게이트를 통과할 때, input의 값이 둘 다 1이면 output은 0이 나옵니다. 이를 위해 Verilog 코드는 두 input을 AND로 연결한 다음 바깥에 not을 붙여 구현하였습니다다. assign e = ~(a&b);와 같은 방식이 될 수 있겠습니다.

3.

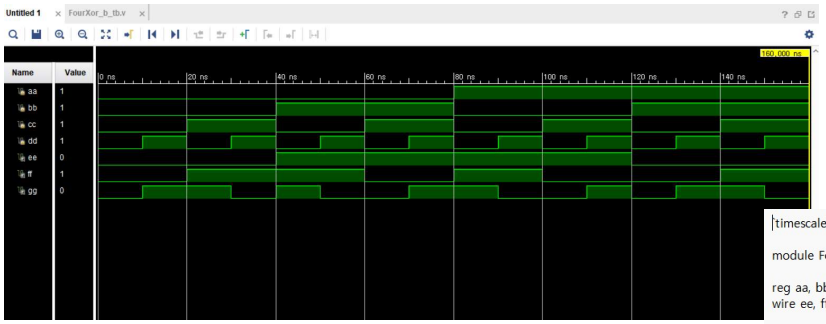

```
`timescale 1ns / 1ps

module FourNor_b(
    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 FourNor_b_tb;

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

FourNor_b 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
```

Input a	Input b	Input c	Input d	Output e	Output f	Output g
0	0	0	0	1	0	1
0	0	0	1	1	0	0
0	0	1	0	1	0	1
0	0	1	1	1	0	0
0	1	0	0	0	1	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	0	0	1
1	0	1	1	0	0	0
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	0	1
1	1	1	1	0	0	0

우선 과정을 요약하자면, output E는 input A와 input B의 nor 값이며, output F는 C와 E의 nor 값이고, output G는 D와 F의 nor 값입니다. 이 nor 게이트를 통과할 때, input의 값이 둘 다 0이면 output은 1이 나옵니다. 이를 위해 Verilog 코드는 두 input을 OR로 연결한 다음, not을 붙여 구현하였습니다. assign e = ~(a|b);와 같은 방식이 될 수 있겠습니다.

4.



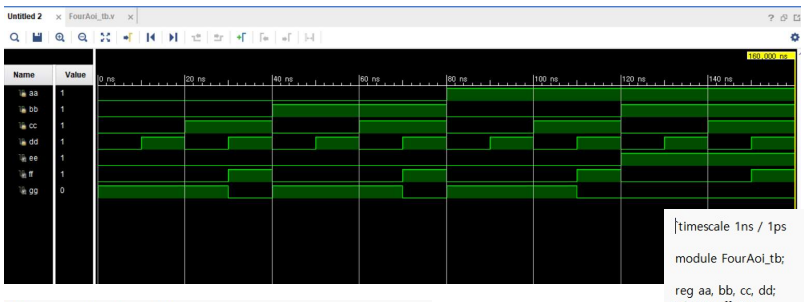

```
`timescale 1ns / 1ps

module FourXor_b(
  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 FourXor_b_tb;

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

FourXor_b 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
```

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	0
0	1	0	0	1	1	1
0	1	0	1	1	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	1
1	0	0	0	1	1	1
1	0	0	1	1	1	0
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	0	1	0	0	1
1	1	1	0	0	1	1
1	1	1	1	0	1	0

우선 과정을 요약하자면, output E는 input A와 input B의 xor 값이며, output F는 C와 E의 xor 값이고, output G는 D와 F의 xor 값입니다. 이 xor 게이트를 통과할 때, input 값 홀수 개가 1이면 output은 1이 나옵니다. 반면, input 값 짝수 개가 1이면 output은 0이 나옵니다. 이를 위해 Verilog 코드는 두 input을 XOR로 연결하여 구현하였습니다. assign e = a^b;와 같은 방식이 될 수 있겠습니다.

5.




```
`timescale 1ns / 1ps

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

    assign e = a&b;
    assign f = c&d;
    assign g = ~(e|f);

endmodule
```

```
timescale 1ns / 1ps

module FourAoi_tb;

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

FourAoi 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
```

Input a	Input b	Input c	Input d	Output e	Output f	Output g
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	0	0	1
0	1	1	0	0	0	1
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	1	1	0

AOI 게이트의 경우 Output E에 A와 B의 and 값이 할당되며, Output F에 C와 D의 and 값이 할당되고, 앞서 언급했던 E와 F에 NOR을 취한 값이 Output G에 할당됩니다. 즉 정리하자면, AOI 게이트는 a와 b, c와 d를 각각 AND 연산한 두 개의 결과를 NOR 연산한 것입니다. 수식으로 나타내면 out g=~((a&b)|(c&d))입니다.

6.

NAND, NOR 게이트는 각각 AND에 NOT, OR에 NOT이 합성된 게이트로 교환법칙과 결합법칙이 성립하지 않습니다. 이는 5주차 때 배울 드모르간의 법칙을 이용해 증명할 수 있습니다.

반면, XOR 게이트는 교환법칙과 결합법칙이 성립한다는 걸 알 수 있습니다. XOR에 NOT이 합성된 XNOR 또한 교환법칙과 결합법칙이 성립합니다.

예) ~(~(X&Y)&Z)≠~(X&~(Y&Z)), ~(~(X|Y)|Z)≠~(X|~(Y|Z)), (X^Y)^Z==X^(Y^Z)

7.

반가산기 : 2개의 input을 더할 때 carry와 sum이 출력되는 회로입니다. 반가산기의 진리표는 다음과 같습니다.

a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

sum은 a^b, carry는 a&b로 나타낼 수 있습니다. 반가산기는 입력으로 들어오는 carry를 받지 않기 때문에 최하위 비트에서만 사용할 수 있습니다. carry를 받는 가산기를 전가산기라고 합니다. 전가산기는 반가산기 두 개와 논리곱 게이트를 이용해 구현할 수 있습니다.

반면, 전가산기는 세 개의 입력 A, B, carry-in 및 두 개의 출력 Sum. Carry-out으로 구성됩니다. 진리표는 다음과 같습니다

a	b	carry in	sum	carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1