3주차 결과보고서

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

* 1. **Verilog 코딩**

Verilog 문법에 따라 자신이 구현하고자 하는 회로의 design source 코드와 testbench 코드를 프로그래밍한다.

* 1. **설계 보드 선택**

Project manager의 setting이라는 아이콘을 눌러 general란의 project device에서 xc7a75tfgg484-1로 device를 지정해준다.

* 1. **I/O 포트 설정**

Add sources – Add or create constraints – Create file 을 통해 xdc 파일을 생성해 변수와 원하는 port번호를 다음의 코드에 입력해 연결한다.

set\_property -dict {PACKAGE\_PIN 포트번호 IOSTANDARD LVCMOS33} [get\_ports 변수]

* 1. **Run Synthesis and run implement**

Run Synthesis 와 run implement 를 클릭하고 Program and Debug에서 Generate Bitstream을 클릭한다.

* 1. **FPGA 연결 및 파일추가**

먼저 FPGA를 컴퓨터와 연결한다. 그 후 Program and debug – open hardware manager – open target – auto connect를 클릭하고 Project 파일 안의 impl\_1 폴더에 강의자료에서 다운받은 debug\_nets.ltx 파일을 추가한다.  
그 후 Program and debug – Program Device 에서 추가했던 파일을 선택해 확인을 눌러 FPGA 기기와 연결한다.

**2. 3- input And gate**



위의 3-input And gate는 A, B, C 라는 3개의 input을 가지고 2개의 output을 가지는 게이트다. 이 게이트는 먼저 A, B 의 input 값을 받고 output인 D를 출력하고 이 output D를 다시 C와 함께 input으로 받아 최종 결과값인 output E를 출력한다. 이에 대한 Verilog코드는 다음과 같다.

*`timescale 1ns / 1ps*

*module and\_3(*

*input a, b, c,*

*output d, e*

*);*

*assign d = a & b;*

*assign e = d & c;*

*endmodule*

assign 문을 이용해 output d에 a와 b의 AND 값을 할당하고 e에 d와 c의 And 값을 할당한다. 다음의 코드는 이에 대한 testbench 코드이다.

`timescale 1ns / 1ps

module and\_3\_tb;

reg a, b, c, d, clk;

wire e, f, g;

and\_3 u\_and\_3 (

.a(a), .b(b), .c(c), .d(d),

.e(e), .f(f), .g(g)

);

initial begin

a = 1'b1;

b = 1'b0;

c = 1'b0;

clk = 0;

end

always clk = #10 ~clk;

always @(posedge clk) begin

a <= #20 ~a;

b <= #30 ~b;

c <= #50 ~c;

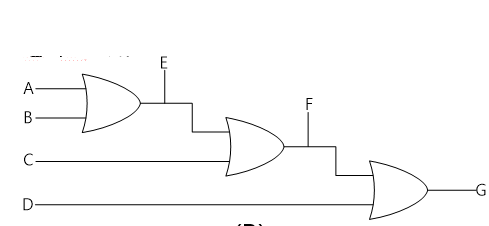
end

endmodule

여기서 always clk = #10 ~clk; 는 clk값이 10ns마다 변한다는 것을 의미한다. 이 코드에 의한 시뮬레이션 결과는 clk 값이 0에서 1로 변할 때마다 a는 20ns을 간격으로 두고 값이 바뀌고 b는 30ns, c는 50ns 간격으로 값이 바뀐다. 따라서 이 코드로 시뮬레이션을 돌리면 3 input And gate의 모든 input a, b, c 의 모든 경우의 수와 이에 따른 output들의 결과값을 얻을 수 있을 것이다. 이에 따라 예상되는 진리표는 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

**3. 4- input And gate**



위의 4-input And gate는 A, B, C, D 라는 4개의 input을 가지고 3개의 output을 가지는 게이트다. 이 게이트는 먼저 A, B 의 input 값을 받고 output인 E를 출력하고 이 output E를 다시 C와 함께 input으로 받아 output F를 출력하고 이 output F를 받아 D와 함께 최종 결과값인 G를 출력한다. 이에 대한 Verilog코드는 다음과 같다.

*`timescale 1ns / 1ps*

*module inv(*

*input a, b, c,d*

*output e, f, g;*

*);*

*assign e = a & b;*

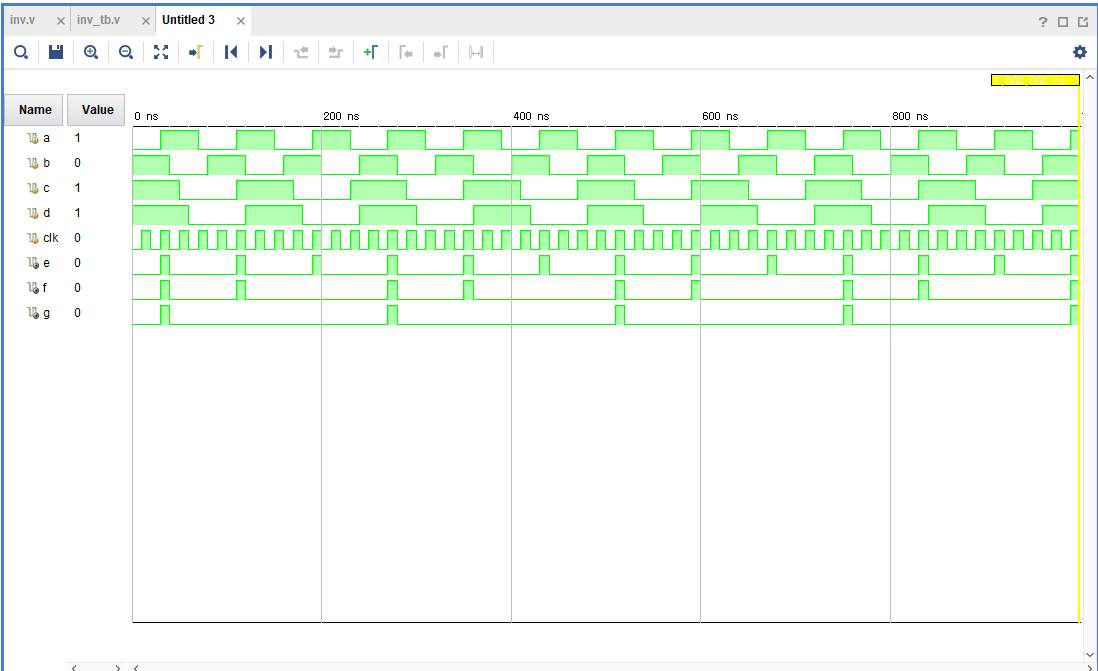
*assign f = d & c;*

*assign g = f & d*

*endmodule*

또한 이에 대한 testbench 코드는 위의 3 input And gate처럼 input output을 추가하고 시간차를

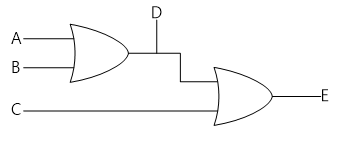
두고 output이 출력되도록 코드를 짜면 된다. 이에 다음과 같은 시뮬레이션이 나타남을 알 수 있다.



이 시뮬레이션을 토대로 진리표를 만들면 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

2-3. 3- input OR gate



위의 3-input OR gate는 3 input AND gate처럼 A, B, C, 라는 3개의 input을 가지고 2개의 output을 가지는 게이트다.. 이에 대한 Verilog코드는 다음과 같다.

*`timescale 1ns / 1ps*

*module inv(*

*input a, b, c,*

*output d,e;*

*);*

*assign d = a | b;*

*assign e = d | c;*

*endmodule*

또한 이에 대한 testbench 코드는 위의 3 input And gate처럼 input output을 추가하고 시간차를

두고 output이 시뮬레이션을 돌리면 3 input Or gate의 모든 input a, b, c 의 모든 경우의 수와 이에 따른 output들의 결과값을 얻을 수 있을 것이다. 이에 따라 예상되는 진리표는 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

2-3. 4- input OR gate



위의 4-input OR gate는 4-input AND gate처럼 A, B, C, D 라는 4개의 input을 가지고 3개의 output을 가지는 게이트다.. 이에 대한 Verilog코드는 다음과 같다.

*`timescale 1ns / 1ps*

*module inv(*

*input a, b, c, d*

*output e,f,g;*

*);*

*assign e = a | b;*

*assign f = e | c;*

*assign g = f | d;*

*endmodule*

또한 이에 대한 testbench 코드는 위의 3 input And gate처럼 input output을 추가하고 시간차를

두고 output이 출력되도록 코드를 짜면 된다. 이에 다음과 같은 시뮬레이션이 나타남을 알 수 있다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

5.

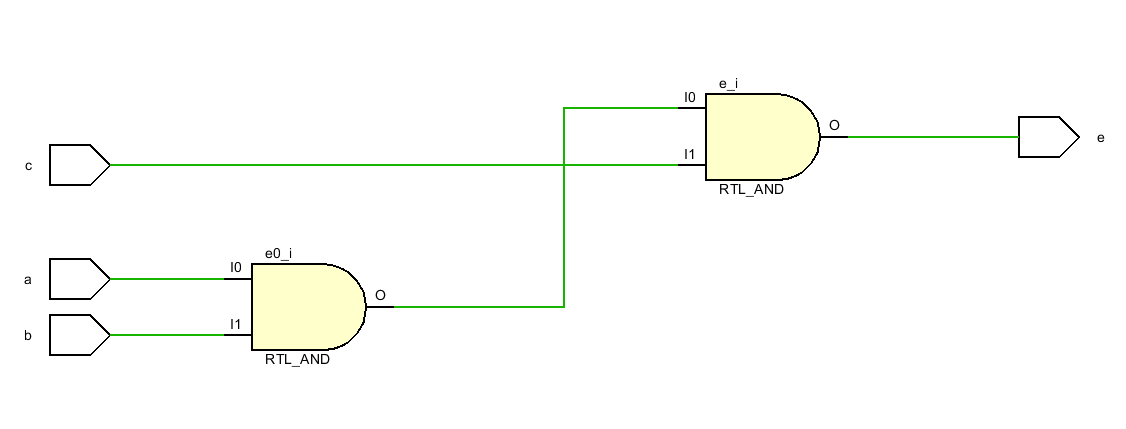
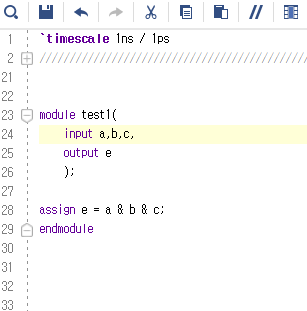
**5-1. 결과 검토**

결과적으로 3-input And gate와 4-input And gate는 입력 값이 모두 1일 때에만 최종 결과값으로 1을 가진다. 즉 input 중 0이 하나라도 존재한다면 결과값으로 0을 출력한다. 반대로 3-input Or gate는 입력 값이 모두 0일 때만 최종 결과값으로 0을 출력하고 하나라도 1이 존재한다면 결과값으로 1을 출력한다.

**5-2. 논의 사항**



위의 회로는 3 input And gate를 이전에 3개의 input과 두 개의 output으로 나타낸 것과 다르게 3개의 input과 하나의 output으로 표현한 것이다.



1. (B)

이러한 다중 입력 게이트를 (A)와 같은 코드를 통해 앞선 두 개의 output을 갖는 AND 게이트와는 다르게 표현했다. 그런데 이 식의 schematic diagram을 확인해보면 앞서 구현한 두 개의 output을 가지는 AND gate와 그 형태가 같은 것을 알 수 있다. 따라서 이 식의 시뮬레이션 결과도 앞선 시뮬레이션 결과와 동일할 것이라 예상할 수 있다.  
 또한 마찬가지로 4-input AND gate, 3-input OR gate, 4-input OR gate 모두 다중 입력게이트로 회로를 설정하더라도 그 schematic diagram이 동일하며 실제로는 같은 회로일 것이다.

**7.** 회로가 복잡해지면 이를 회로로 표현하기 어려워질 수 있다. 따라서 복잡한 회로를 간단하게 알기 위해서 Boolean algebra 의 법칙들을 사용하며 수식을 간단하게 만들어야 한다. 알아둬야할 Boolean algebra의 기본 성질로는 다음의 법칙들이 있다.

#A + B = B + A #AB = BA 교환법칙

#A + (B + C) = (A + B) + C #A(BC) = (AB)C 결합법칙

#A + 0 = A #Aㆍ1 = A 항등법칙

#A(B + C) = AB + AC # A + BC = (A + B)(A + C) 분배법칙

#A + AB = A # A(A + B) = A 흡수법칙

#(A + B)’ = A’B’ # (AB)’ = A’ + B’ 드 모르간의 법칙

AB + AB’ = (A + B)(A + B’) = A 인접 법칙