3주차 결과보고서

전공: 아트&테크놀로지학과 학년: 4학년 학번: 20191048 이름: 김도솔

**1.**

**FPGA 동작법**은 Verilog coding, Device/Pin assignment, Run synthesis, Synthesis/Implement, Device configuration 등의 단계를 거친다. 각 단계별 세부 설명은 다음과 같다.

1. **Verilog coding**

Vivado에서 Project Manager > Add Sources > Add or create design sources/ simulation sources > Create File을 통해 design source와 simulation source 파일을 생성한다. 그 후 Sources 메뉴를 눌러 파일을 선택해 원하는 회로 설계에 맞게 코드를 작성하고 저장한다.

1. **Device assignment**

프로젝트에 적절한 Device를 assign하는 단계이다. Vivado에서 Project Manager > Settings > General > Project device를 xc7a75tfgg484-1(Artix7)로 설정하면 device assign이 완료된다.

1. **Pin assignment**

Verilog 코드의 input, output 변수에 Pin을 assign하는 단계이다. Vivado에서 Project Manager > Add Sources > Add or create constraints > Create File 을 통해 .xdc 파일을 생성한다. 다음으로 Window > I/O Ports > All ports > Scalar ports를 열어 FPGA pin list에서 할당하고 싶은 Pin을 선택해 Verilog 소스의 port와 연결한다.

ex) set\_property -dict {PACKAGE\_PIN J4 IOSTANDARD LVCMOS33} [get\_ports a]

1. **Synthesis/Implement**

Synthesis 단계에서는 디자인을 기술적으로 설계하고 논리 게이트와 레지스터의 논리 구성을 결정한다. Implement 단계에서는 FPGA의 실제 리소스와 시계 주파수 제약 조건을 고려하여 디자인을 물리적으로 배치하고 연결한다. Vivado에서 Synthesis > Run Synthesis를 완료한 후 Implementation > Run Implementation > Open Implemented Design을 완료하면 Synthesis Design과 Route Design이 생성된다.

1. **Device Configuration**

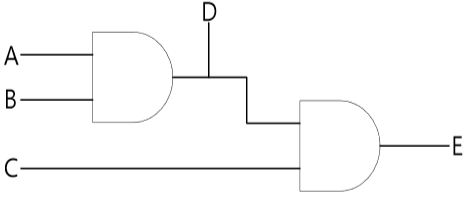
FPGA 칩에 프로그램을 로드하고, FPGA 내부 리소스를 초기화하여 디자인을 실행하는 단계를 Device Configuration이라 한다. Vivado에서 Program and debug > Generate Bitstream > Open Hardware Manager를 통해 Device configuration 단계를 완료한다.

1. **FPGA 동작**

Vivado에서 Program and debug > Open Hardware Manager > Open Target > Auto Connect 클릭 후 Project명 > Project명\_runs > impl\_1 폴더에 debug\_nets.ltx 파일을 넣는다. 다음으로 Program and debug > Open Hardware Manager > Program Device > xc7a75t\_0를 클릭하고 Debug probes file을 debug\_nets.ltx 파일로 설정하면 외부 FPGA 기기와 연결이 완료되어 FPGA 동작 결과를 확인할 수 있다.

**2.**

**3-input AND gate** **(3 input, 2 output)**



(B)

1. Boolean식

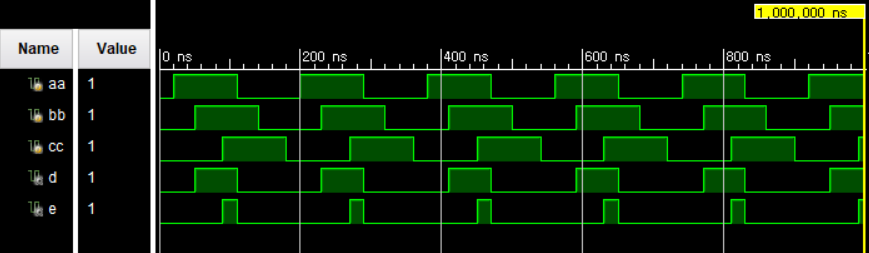
(B)의 경우 A와 B를 AND gate에 연결해 D를 출력하고, D와 C를 AND gate에 연결해 E를 출력한다. 따라서 (B)의 Boolean식은 A\*B = D, D\*C = E의 순서로 구성할 수 있다.

1. Verilog 코딩

위의 Boolean식에 따라 Verilog 코드를 작성한 결과는 다음과 같다.

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `timescale 1ns / 1ps  module inv(  input a, b, c,  output d, e  );  assign d=a&b;  assign e=d&c;  endmodule | `timescale 1ns / 1ps  module inv\_tb;  reg aa, bb, cc;  wire d, e;  inv u\_inv (  .a (aa ),  .b (bb ),  .c (cc ),  .d (d ),  .e (e ));  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  end  always begin  aa = #20 ~aa;  bb = #30 ~bb;  cc = #40 ~cc;  end  initial begin  #1000  $finish;  end  endmodule |

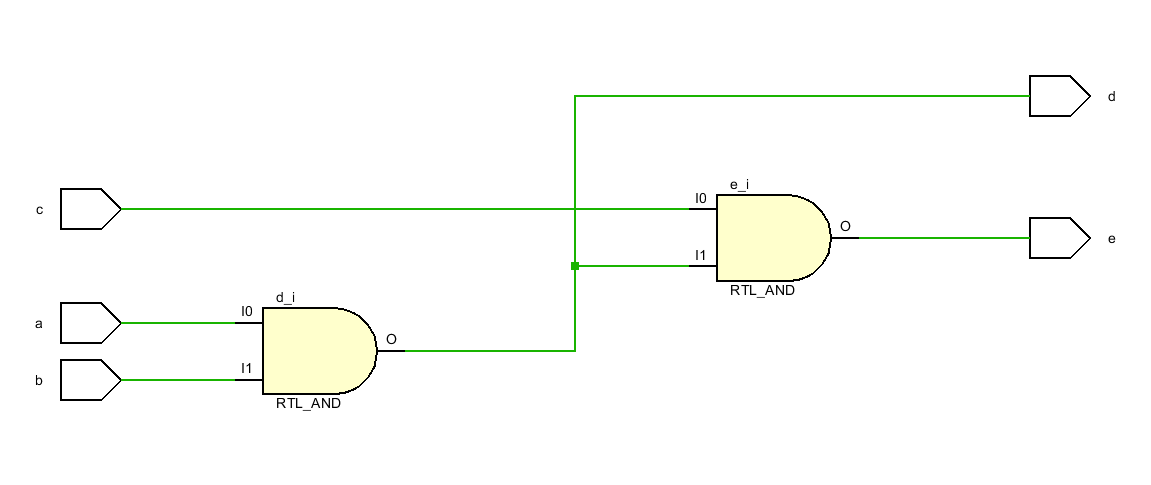
1. Simulation 결과



위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 A, B, C의 값을 바꾸면 그에 따라 출력 D, E의 값이 변화하는 것을 확인할 수 있다. 출력 D는 A와 B가 둘다 1인 경우에만 1의 값을 가진다. 출력 E는 D와 C의 값이 둘다 1인 경우에만 1의 값을 가진다. 다시 말해 출력 E는 입력 A, B, C가 모두 1인 경우에만 1의 값을 가진다. 이 시뮬레이션 결과를 통해 5)의 진리표를 작성할 수 있다.

시뮬레이션 과정은 다음과 같다. 입력 A, B, C 모두 초기값은 0이다. 시간이 지남에 따라 입력 A가 먼저 1로 변화한다. 출력 D와, E는 아무런 변화가 없다. 다음으로 입력 B도 1로 변화한다. 이때 출력 D도 1의 값을 가진다. 다음으로 입력 C도 1로 변화한다. 이때 출력 D, E 모두 1의 값을 가지게 된다. 시간이 더 지나 입력 A가 0으로 변화한다. 이때 출력 D, E 모두 값이 0으로 변환된다. 그 후 입력 B와 C의 값이 순차적으로 0으로 바뀐다. 출력값은 모두 0으로 동일하게 유지되는 것을 알 수 있다.

1. Schematic Diagram



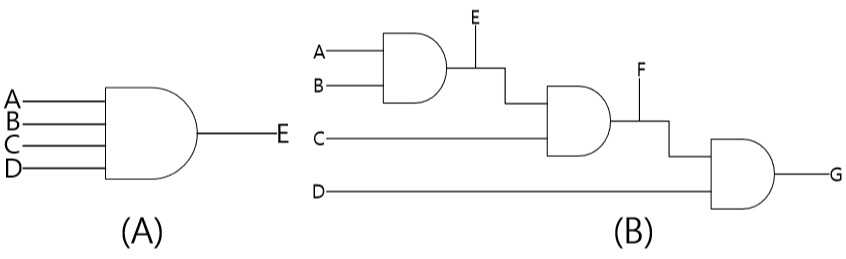
위는 Vivado에서 생성한 3-input AND gate의 Schematic Diagram이다. 입력 A와 B를 AND gate에 연결해 D를 출력하고, 입력 D와 C를 AND gate에 연결해 E를 출력한다는 점에서 (B)와 동일하게 동작하는 것을 알 수 있다.

1. 3-input AND gate의 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| In A | In B | In C | Out D | Out 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** **(4 input, 3 output)**



(B)

1. Boolean식

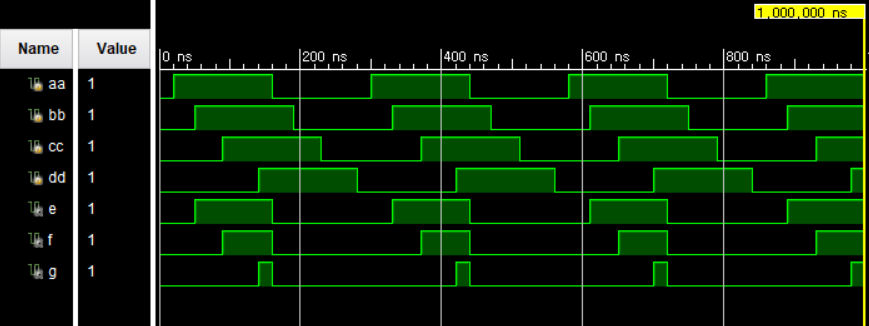
(B)의 경우 A와 B를 AND gate에 연결해 E를 출력하고, E와 C를 AND gate에 연결해 F를 출력하고, F와 D를 AND gate에 연결해 G를 출력한다. 따라서 (B)의 Boolean식은 A\*B = E, E\*C = F, F\*D = G의 순서로 구성할 수 있다.

1. Verilog 코딩

위의 Boolean식에 따라 Verilog 코드를 작성한 결과는 다음과 같다.

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `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 | `timescale 1ns / 1ps  module inv\_tb;  reg aa, bb, cc, dd;  wire e, f, g;  inv u\_inv (  .a (aa ),  .b (bb ),  .c (cc ),  .d (dd ),  .e (e ),  .f (f ),  .g (g ) );  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  dd = 1'b0;  end  always begin  aa = #20 ~aa;  bb = #30 ~bb;  cc = #40 ~cc;  dd = #50 ~dd;  end  initial begin  #1000  $finish;  end  endmodule |

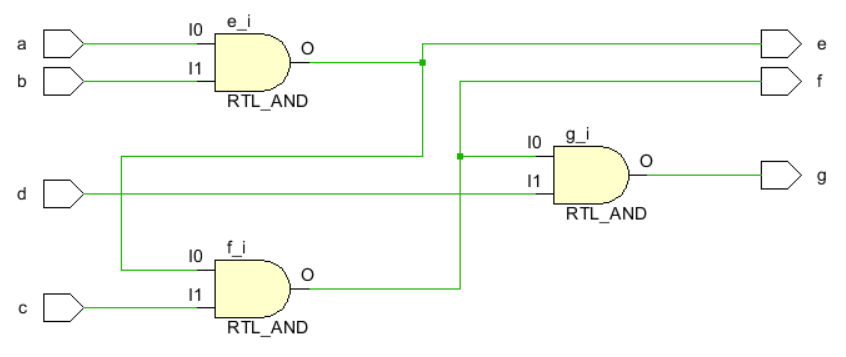
1. Simulation 결과



위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 A, B, C, D의 값을 바꾸면 그에 따라 출력 E, F, G의 값이 변화하는 것을 확인할 수 있다. 출력 E는 A와 B가 둘다 1인 경우에만 1의 값을 가진다. 출력 F는 E와 C의 값이 둘다 1인 경우에만 1의 값을 가진다. 다시 말해 출력 F는 입력 A, B, C가 모두 1인 경우에만 1의 값을 가진다. 출력 G는 F와 D의 값이 둘다 1인 경우에만 1의 값을 가진다. 다시 말해 출력 G는 입력 A, B, C, D 모두 1인 경우에만 1의 값을 가진다. 이 시뮬레이션 결과를 통해 5)의 진리표를 작성할 수 있다.

시뮬레이션 과정은 다음과 같다. 입력 A, B, C, D 모두 초기값은 0이다. 시간이 지남에 따라 입력 A가 먼저 1로 변화한다. 출력 E, F. G는 아무런 변화가 없다. 다음으로 입력 B도 1로 변화한다. 이때 출력 E도 1의 값을 가진다. 다음으로 입력 C도 1로 변화한다. 이때 출력 F도 1의 값을 가지게 된다. 다음으로 입력 D까지 1로 변화하면 출력 G도 1으로 변환돼 모든 출력 값이 1이 되는 것을 알 수 있다. 시간이 더 지나 입력 A가 0으로 변화한다. 이때 출력 E, F, G 모두 동시에 0으로 변환된다. 그 후 입력 B, C, D의 값이 순차적으로 0으로 바뀐다. 출력값은 모두 0으로 동일하게 유지되는 것을 알 수 있다.

1. Schematic Diagram



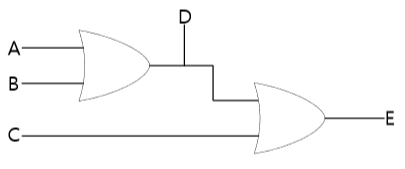
위는 Vivado에서 생성한 4-input AND gate의 Schematic Diagram이다. 입력 A와 B를 AND gate에 연결해 E를 출력하고, 입력 E와 C를 AND gate에 연결해 F를 출력하고, 입력 F와 D를 AND gate에 연결해 G를 출력한다는 점에서 (B)와 동일하게 동작하는 것을 알 수 있다.

1. 4-input AND gate의 진리표

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out E | Out F | Out 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 |

**4.**

**3-input OR gate** **(3 input, 2 output)**



(B)

1. Boolean식

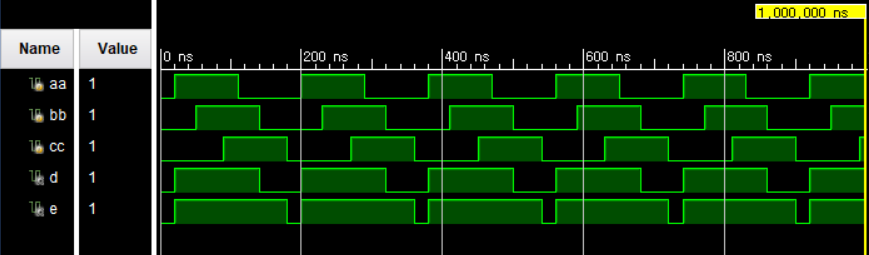
(B)의 경우 A와 B를 OR gate에 연결해 D를 출력하고, D와 C를 OR gate에 연결해 E를 출력한다. 따라서 (B)의 Boolean식은 A+B = D, D+C = E의 순서로 구성할 수 있다.

1. Verilog 코딩

위의 Boolean식에 따라 Verilog 코드를 작성한 결과는 다음과 같다.

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `timescale 1ns / 1ps  module inv(  input a, b, c,  output d, e  );  assign d=a|b;  assign e=d|c;  endmodule | `timescale 1ns / 1ps  module inv\_tb;  reg aa, bb, cc;  wire d, e;  inv u\_inv (  .a (aa ),  .b (bb ),  .c (cc ),  .d (d ),  .e (e ));  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  end  always begin  aa = #20 ~aa;  bb = #30 ~bb;  cc = #40 ~cc;  end  initial begin  #1000  $finish;  end  endmodule |

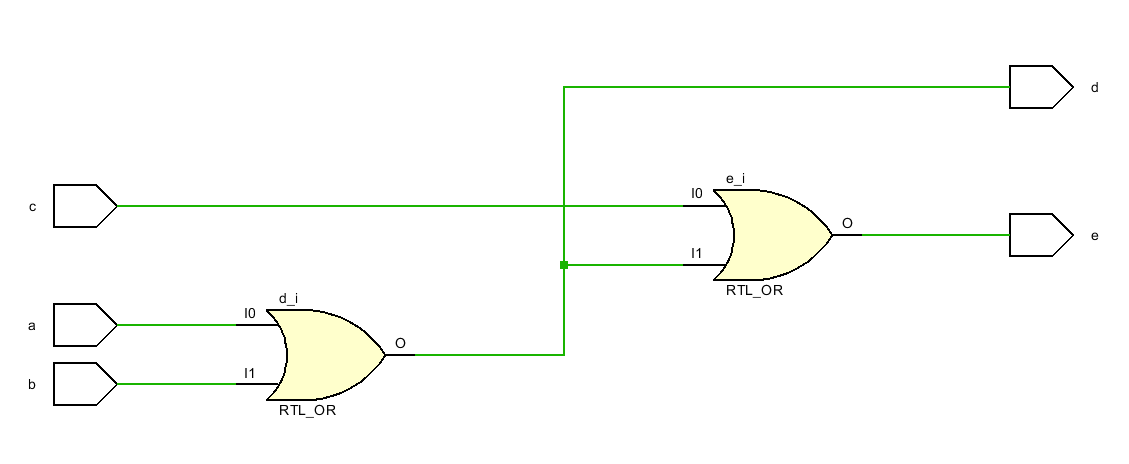
1. Simulation 결과



위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 A, B, C의 값을 바꾸면 그에 따라 출력 D, E의 값이 변화하는 것을 확인할 수 있다. 출력 D는 A와 B 중 하나만 1이어도 1의 값을 가진다. 출력 E는 D와 C 중 하나만 1이어도 1의 값을 가진다. 다시 말해 출력 E는 입력 A, B, C 중 하나만 1이어도 1의 값을 가지고 A, B, C 모두 0인 경우에만 0의 값을 가진다. 이 시뮬레이션 결과를 통해 5)의 진리표를 작성할 수 있다.

시뮬레이션 과정은 다음과 같다. 입력 A, B, C 모두 초기값은 0이다. 시간이 지남에 따라 입력 A가 먼저 1로 변화한다. 이때 출력 D와 E도 동시에 1의 값을 가진다. 다음으로 입력 B와 C의 값이 순차적으로 1로 변화한다. 출력 D와 E의 값은 동일하게 1로 유지된다. 시간이 더 지나 입력 A가 0으로 변화한다. 두 출력값은 여전히 1로 유지된다. 다음으로 입력 B도 0으로 변화한다. 이때 출력 D의 값이 0으로 변환된다. 마지막으로 입력 C까지 0으로 변화하면 출력 E도 0으로 변환돼 모든 출력 값이 0이 되는 것을 알 수 있다.

1. Schematic Diagram



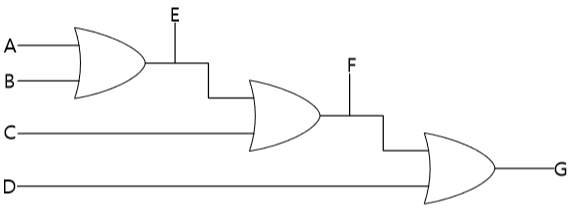
위는 Vivado에서 생성한 3-input OR gate의 Schematic Diagram이다. 입력 A와 B를 OR gate에 연결해 D를 출력하고, 입력 D와 C를 OR gate에 연결해 E를 출력한다는 점에서 (B)와 동일하게 동작하는 것을 알 수 있다.

1. 3-input OR gate의 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| In A | In B | In C | Out D | Out 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 |

**5.**

**4-input OR gate** **(4 input, 3 output)**



(B)

1. Boolean식

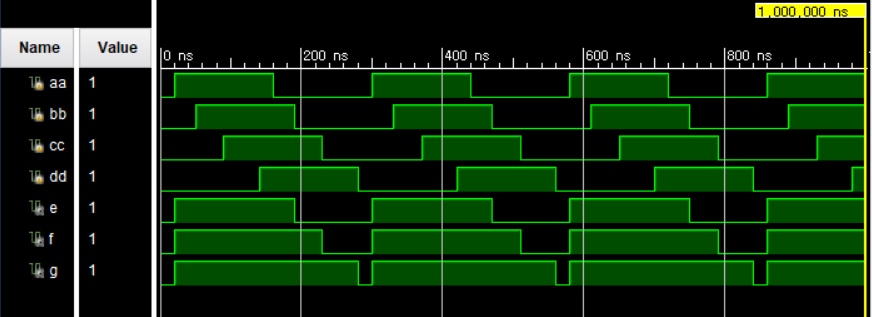
(B)의 경우 A와 B를 OR gate에 연결해 E를 출력하고, E와 C를 OR gate에 연결해 F를 출력하고, F와 D를 OR gate에 연결해 G를 출력한다. 따라서 (B)의 Boolean식은 A+B = E, E+C = F, F+D = G의 순서로 구성할 수 있다.

1. Verilog 코딩

위의 Boolean식에 따라 Verilog 코드를 작성한 결과는 다음과 같다.

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `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 | `timescale 1ns / 1ps  module inv\_tb;  reg aa, bb, cc, dd;  wire e, f, g;  inv u\_inv (  .a (aa ),  .b (bb ),  .c (cc ),  .d (dd ),  .e (e ),  .f (f ),  .g (g ) );  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  dd = 1'b0;  end  always begin  aa = #20 ~aa;  bb = #30 ~bb;  cc = #40 ~cc;  dd = #50 ~dd;  end  initial begin  #1000  $finish;  end  endmodule |

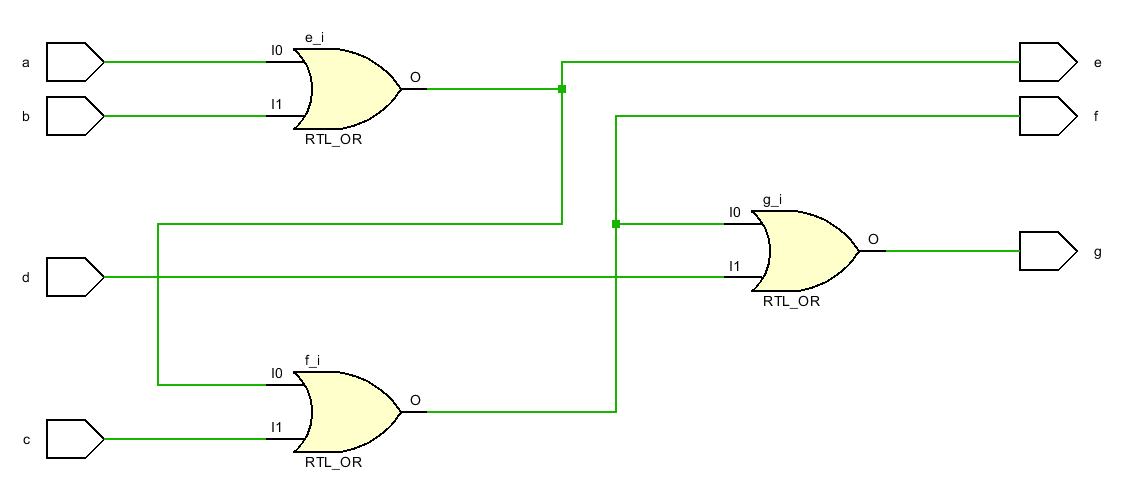
1. Simulation 결과



위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 A, B, C, D의 값을 바꾸면 그에 따라 출력 E, F, G의 값이 변화하는 것을 확인할 수 있다. 출력 E는 A와 B 중 하나만 1이어도 1의 값을 가진다. 출력 F는 E와 C 중 하나만 1이어도 1의 값을 가진다. 다시 말해 출력 F는 입력 A, B, C 중 하나만 1이어도 1의 값을 가지고 A, B, C 모두 0인 경우에만 0의 값을 가진다. 출력 G는 F와 D 중 하나만 1이어도 1의 값을 가진다. 다시 말해 출력 G는 입력 A, B, C, D 중 하나만 1이어도 1의 값을 가지고 A, B, C, D 모두 0인 경우에만 0의 값을 가진다. 이 시뮬레이션 결과를 통해 5)의 진리표를 작성할 수 있다.

시뮬레이션 과정은 다음과 같다. 입력 A, B, C, D 모두 초기값은 0이다. 시간이 지남에 따라 입력 A가 먼저 1로 변화한다. 이때 출력 E, F, G도 동시에 1의 값을 가진다. 다음으로 입력 B, C, D의 값이 순차적으로 1로 변화한다. 출력 E, F, G의 값은 동일하게 1로 유지된다. 시간이 더 지나 입력 A가 0으로 변화한다. 모든 출력 값은 여전히 1로 유지된다. 다음으로 입력 B도 0으로 변화한다. 이때 출력 E의 값이 0으로 변환된다. 다음으로 입력 C도 0으로 변화한다. 이때 출력 F의 값이 0으로 변환된다. 마지막으로 입력 D까지 0으로 변화하면 출력 G도 0으로 변환돼 모든 출력 값이 0이 되는 것을 알 수 있다.

1. Schematic Diagram



위는 Vivado에서 생성한 4-input OR gate의 Schematic Diagram이다. 입력 A와 B를 OR gate에 연결해 E를 출력하고, 입력 E와 C를 OR gate에 연결해 F를 출력하고, 입력 F와 D를 OR gate에 연결해 G를 출력한다는 점에서 (B)와 동일하게 동작하는 것을 알 수 있다.

1. 4-input OR gate의 진리표

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out E | Out F | Out 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 |

**6.**

3주차 실험을 통해, Vivado에서 논리 회로 코드를 작성하고 이를 FPGA 보드와 연결해 작동시키는 방법을 알게 되었다. 또한 FPGA 보드 동작 결과와 각 Gate별 진리표 작성 등을 통해 AND Gate와 OR Gate의 입력에 따른 출력값을 실험적으로 확인할 수 있었다. AND Gate는 input 개수에 상관 없이 모든 입력이 1이면 1을 출력한다. 즉 입력 중 하나라도 0의 값을 가지면 0을 출력한다. 반면 OR Gate는 input 개수에 상관 없이 모든 입력이 0이면 0을 출력한다. 즉 입력 중 하나라도 1의 값을 가지면 1을 출력한다.

**7.**

이번 실험에서는 우리에게 비교적 익숙한 AND/OR Gate를 사용해 다중 입력 AND/OR Gate를 구현했다. 그러나 논리 회로 실무 설계 시에는 AND/OR Gate 대신 NAND/ NOR Gate가 선호된다고 한다. 그 이유는 다음과 같다.

1. 유니버설 게이트로서의 역할: NAND 및 NOR 게이트는 유니버설 게이트로 간주된다. 이는 어떤 논리 회로도 NAND 또는 NOR 게이트만으로 구현할 수 있다는 것을 의미한다. 따라서 NAND와 NOR 게이트만으로 다양한 논리 회로를 구현할 수 있으며, 다른 게이트 종류보다 더 간단한 회로를 설계할 수 있다.
2. 일관성 및 간소화: NAND와 NOR 게이트만을 사용하면 회로가 일관성 있고 간소화된다. 이로 인해 디자인과 디버깅이 더 쉬워지며, 논리 회로의 복잡성을 낮출 수 있다.
3. 소자 수 감소: NAND와 NOR 게이트를 사용하면 디자인에서 필요한 소자(트랜지스터 등)의 수를 줄일 수 있다. 이는 회로의 크기와 전력 소비를 감소시킬 수 있다.
4. 신뢰성 향상: NAND와 NOR 게이트는 일반적으로 안정적인 논리 게이트로 간주되며, 노이즈와 같은 외부 영향에 대해 더 견고하다.

따라서 NAND와 NOR 게이트는 논리 회로 설계에서 일반적으로 더 선호되는 선택지이다. 그러나 AND 및 OR 게이트는 여전히 특정 응용 분야나 특수한 상황에서 사용될 수 있으며, 특정 설계 요구 사항에 따라 선택할 수 있다.