**3주차 결과 보고서**

20150555 남민혁

**1. 실험 목적**

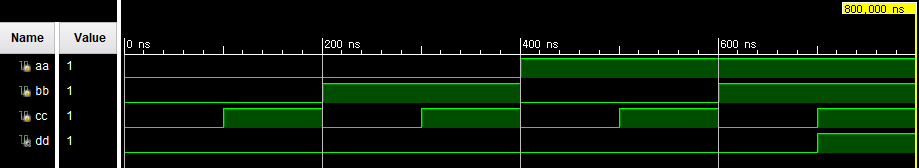
3주차 실험의 목적은 아래와 같다.

* AND/OR/NOT 게이트 동작의 이해와 확인
* Verilog를 사용하여 다중 입력 AND/OR/NOT 게이트의 구현
* 입력 신호 생성 후 Simulation를 통해 구현된 각 게이트의 동작을 확인

**2. FPGA 동작법을 설명하시오.**

FPGA는 아래 과정을 통하여 동작한다.

* Functional Architecture Design: 이론 및 스키메틱을 통해서 디자인을 설계한다.
* Design Entry: HDL을 통하여 위에서 설계한 내용을 코딩한다.
* RTL Simulation: 설계한 내용을 시뮬레이션으로 검증해본다.
* Device / Pin Assignment: 유저가 high-level에서 구현한 디자인을 적용할 특정한 FPGA 보드의 specification에 적용되도록 배정한다. 더불어 HDL에서 선언한 레지스터의 입출력 값을 보드의 특정 Pin과 매칭시켜 물리적인 입출력 값으로 적용한다.
* Synthesis / Implement: 앞에서 구현한 high-level 디자인을 FPGA 보드가 이해할 수 있는 low-level 즉, bitstream 파일을 생성한다.
* Device Configuration: FPGA보드를 사용하여 구현한 디자인이 실제로 구현되었는지 확인한다.

**3. 3-input AND gate의 simulation 결과 및 과정에 대해서 설명하시오. (3 input, 2 output)[3장 ppt 31 page 참조 , 진리표 작성]**

**-Boolean 식**

|  |  |
| --- | --- |
| (A) | (B) |
| ABC=D | (AB)C=E |

**-Verilog 코딩**

|  |  |  |
| --- | --- | --- |
|  | (A) | (B) |
| Verilog | `timescale 1ns / 1ps  module three\_input\_and\_gate\_a(  input a,  input b,  input c,  output d  );  assign d = a & b & c;  endmodule | `timescale 1ns / 1ps  module three\_input\_and\_gate\_b(  input a,  input b,  input c,  output d,  output e  );  assign d = a & b;  assign e = c & d;  endmodule |
| Test Bench | module three\_input\_and\_gate\_a\_tb;  reg aa;  reg bb;  reg cc;  wire dd;  three\_input\_and\_gate\_a u\_three\_input\_and\_gate\_a(  .a (aa),  .b (bb),  .c (cc),  .d (dd)  );  initial aa = 1'b0;  initial bb = 1'b0;  initial cc = 1'b0;  always aa = #400 ~aa;  always bb = #200 ~bb;  always cc = #100 ~cc;  initial begin  #800  $finish;  end  endmodule | module three\_input\_and\_gate\_b\_tb;  reg aa;  reg bb;  reg cc;  wire dd;  wire ee;  three\_input\_and\_gate\_b u\_three\_input\_and\_gate\_b(  .a (aa),  .b (bb),  .c (cc),  .d (dd),  .e (ee));  initial aa = 1'b0;  initial bb = 1'b0;  initial cc = 1'b0;  always aa = #400 ~aa;  always bb = #200 ~bb;  always cc = #100 ~cc;  initial begin  #800  $finish;  end  endmodule |

**-Simulation 출력 결과 비교**

|  |  |
| --- | --- |
| A |  |
| B |  |

**-진리표**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

***Note*** (B)를 기준으로 작성되었으며, (A)는 위 진리표의 마지막 열에 있는 Output을 Output D로 고려하면 된다.

**4. 4-input AND gate의 simulation 결과 및 과정에 대해서 설명하시오. (4 input, 3 output)[3장 ppt 33 page 참조 , 진리표 작성]**

**-Boolean 식**

|  |  |
| --- | --- |
| A | B |
| ABCD=E | ((AB)C)D=G |

**-Verilog 코딩**

|  |  |  |
| --- | --- | --- |
|  | (A) | (B) |
| Verilog | `timescale 1ns / 1ps  module four\_input\_and\_gate\_b(  input a,  input b,  input c,  input d,  output e  );  assign e = a & b & c &d;  endmodule | `timescale 1ns / 1ps  module four\_input\_and\_gate\_b(  input a,  input b,  input c,  input d,  output e,  output f,  output g  );  assign e = a & b;  assign f = e & c;  assign g = f & d;  endmodule |
| Test Bench | module four\_input\_and\_gate\_a\_tb;  reg aa;  reg bb;  reg cc;  reg dd;  wire ee;  four\_input\_and\_gate\_a u\_four\_input\_and\_gate\_a(  .a (aa),  .b (bb),  .c (cc),  .d (dd),  .e (ee));  initial aa = 1'b0;  initial bb = 1'b0;  initial cc = 1'b0;  initial dd = 1'b0;  always aa = #400 ~aa;  always bb = #200 ~bb;  always cc = #100 ~cc;  always dd = #50 ~dd;  initial begin  #800  $finish;  end  endmodule | module four\_input\_and\_gate\_b\_tb;  reg aa;  reg bb;  reg cc;  reg dd;  wire ee;  wire ff;  wire gg;  four\_input\_and\_gate\_b u\_four\_input\_and\_gate\_b(  .a (aa),  .b (bb),  .c (cc),  .d (dd),  .e (ee),  .f (ff),  .g (gg));  initial aa = 1'b0;  initial bb = 1'b0;  initial cc = 1'b0;  initial dd = 1'b0;  always aa = #400 ~aa;  always bb = #200 ~bb;  always cc = #100 ~cc;  always dd = #50 ~dd;  initial begin  #800  $finish;  end  endmodule |

**-Simulation 출력 결과 비교**

|  |  |
| --- | --- |
| A |  |
| B |  |

**-진리표**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |

***Note*** (B)를 기준으로 작성되었으며, (A)는 위 진리표의 마지막 열에 있는 Output을 Output E로 고려하면 된다.

**5. 3-input OR gate의 simulation 결과 및 과정에 대해서 설명하시오. (3 input, 2 output)[3장 ppt 35 page 참조, 진리표 작성]**

**-Boolean 식**

|  |  |
| --- | --- |
| A | B |
| A+B+C=D | (A+B)+C=E |

**-Verilog 코딩**

|  |  |  |
| --- | --- | --- |
|  | (A) | (B) |
| Verilog | `timescale 1ns / 1ps  module three\_input\_or\_gate\_a(  input a,  input b,  input c,  output d  );  assign d = a | b | c;  endmodule | `timescale 1ns / 1ps  module three\_input\_or\_gate\_b(  input a,  input b,  input c,  output d,  output e  );  assign d = a | b;  assign e = c | d;  endmodule |
| Test Bench | module three\_input\_or\_gate\_a\_tb;  reg aa;  reg bb;  reg cc;  wire dd;  three\_input\_or\_gate\_a u\_three\_input\_or\_gate\_a(  .a (aa),  .b (bb),  .c (cc),  .d (dd)  );  initial aa = 1'b0;  initial bb = 1'b0;  initial cc = 1'b0;  always aa = #400 ~aa;  always bb = #200 ~bb;  always cc = #100 ~cc;  initial begin  #800  $finish;  end  endmodule | module three\_input\_or\_gate\_b\_tb;  reg aa;  reg bb;  reg cc;  wire dd;  wire ee;  three\_input\_or\_gate\_b u\_three\_input\_or\_gate\_b(  .a (aa),  .b (bb),  .c (cc),  .d (dd),  .e (ee));  initial aa = 1'b0;  initial bb = 1'b0;  initial cc = 1'b0;  always aa = #400 ~aa;  always bb = #200 ~bb;  always cc = #100 ~cc;  initial begin  #800  $finish;  end  endmodule |

**-Simulation 출력 결과 비교**

|  |  |
| --- | --- |
| A |  |
| B |  |

**-진리표**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

***Note*** (B)를 기준으로 작성되었으며, (A)는 위 진리표의 마지막 열에 있는 Output을 Output D로 고려하면 된다.

**6. 4-input OR gate의 simulation 결과 및 과정에 대해서 설명하시오. (4 input, 3 output)[3장 ppt 37 page 참조 , 진리표 작성]**

**-Boolean 식**

|  |  |
| --- | --- |
| A | B |
| A+B+C+D=E | ((A+B)+C)+D=G |

**-Verilog 코딩**

|  |  |  |
| --- | --- | --- |
|  | (A) | (B) |
| Verilog | `timescale 1ns / 1ps  module four\_input\_or\_gate\_a(  input a,  input b,  input c,  input d,  output e  );  assign e = a | b | c |d;  endmodule | `timescale 1ns / 1ps  module four\_input\_or\_gate\_b(  input a,  input b,  input c,  input d,  output e,  output f,  output g  );  assign e = a | b;  assign f = e | c;  assign g = f | d;  endmodule |
| Test Bench | module four\_input\_or\_gate\_a\_tb;  reg aa;  reg bb;  reg cc;  reg dd;  wire ee;  four\_input\_or\_gate\_a u\_four\_input\_or\_gate\_a(  .a (aa),  .b (bb),  .c (cc),  .d (dd),  .e (ee));  initial aa = 1'b0;  initial bb = 1'b0;  initial cc = 1'b0;  initial dd = 1'b0;  always aa = #400 ~aa;  always bb = #200 ~bb;  always cc = #100 ~cc;  always dd = #50 ~dd;  initial begin  #800  $finish;  end  endmodule | module four\_input\_or\_gate\_b\_tb;  reg aa;  reg bb;  reg cc;  reg dd;  wire ee;  wire ff;  wire gg;  four\_input\_or\_gate\_b u\_four\_input\_or\_gate\_b(  .a (aa),  .b (bb),  .c (cc),  .d (dd),  .e (ee),  .f (ff),  .g (gg));  initial aa = 1'b0;  initial bb = 1'b0;  initial cc = 1'b0;  initial dd = 1'b0;  always aa = #400 ~aa;  always bb = #200 ~bb;  always cc = #100 ~cc;  always dd = #50 ~dd;  initial begin  #800  $finish;  end  endmodule |

**-Simulation 출력 결과 비교**

|  |  |
| --- | --- |
| A |  |
| B |  |

**-진리표**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |

***Note*** (B)를 기준으로 작성되었으며, (A)는 위 진리표의 마지막 열에 있는 Output을 Output E로 고려하면 된다.

**7. 결과 검토 및 논의사항**

Two-level로 구현한 결과와 One-level로 구현한 결과의 Simulation 결과는 다르지 않았다. 다만 중간 결과 출력을 위한 register를 추가적으로 사용하여 추가적인 결과를 출력할 수 있었다.

Verilog 코딩을 한 후 Schematic한 결과를 출력했을 때, One-level로 의도한 코드 역시 Two-level로 나타나, 이로 인하여 실험 결과가 잘못되었을 수도 있어 이 부분에 대한 추가적인논의가 필요해보인다.

**8. 추가 이론 조사 및 작성**

본 실험에서는 간단한 AND/OR 논리회로를 구현하였지만, 실제로 one-level과 two-level로 다르게 구현함에 따라 결과 출력까지 걸리는 시간이나, 제작에 필요한 비용적인 측면에서 차이가 생길 것으로 보인다. 구현 방식에 있어 차이는 있지만, 논리적으로는 모두 동일한 식임을 진리표와 Boolean 식을 통해 알 수 있었다.

**9. 참고 문헌**

강석태, “Verilog HDL Summary”, http://vlsi.hongik.ac.kr/lecture/%EC%8B%A4%ED%97%98/Verilog\_Summary.pdf.

레바스, “Verilog HDL 문법 (컴파일러 지시어, 조건문, 다중 분기, 반복문)”, https://rebas.kr/214?category=603584.

한국기술교육대학교 전기전자통신공학부, “디지털시스템 설계 및 실습” Verilog HDL 문법 강의자료, https://cms3.koreatech.ac.kr/sites/yjjang/down/dsys11/M01\_VerilogHDL01.pdf.

Treeroad, “Assignment”, https://treeroad.tistory.com/entry/Assignment.