5주차 결과보고서

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

**1.**

5주차 실험의 목적은 드모르간의 제 1 법칙과 제 2 법칙을 이해하고 그것이 적용된 Boolean 함수가 실제로 어떻게 동작하는지 확인하는 것이다. 먼저 Vivado에서 Verilog 언어를 사용해 강의 자료에 주어진 드모르간의 정리, Boolean 함수의 동작을 구현한다. 다음으로 입력 신호를 생성한 후, 시뮬레이션 결과를 통해 실제로 어떻게 구현 됐는지 확인한다. 마지막으로 FPGA 보드에 연결하여 앞에서 구현한 회로가 어떻게 동작하는지 확인한다.

**2.**

**1) De-Morgan의 제 1 법칙**

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* (A)와 (B)의 Schematic 비교

도표, 라인, 그래프, 폰트이(가) 표시된 사진

자동 생성된 설명

(A)의 Schematic Diagram

도표, 라인, 평면도, 그래프이(가) 표시된 사진

자동 생성된 설명

(B)의 Schematic Diagram

(A)의 Schematic Diagram은 OR 게이트 출력 부분에 NOT 게이트가 연결된 형태이고, (B)의 Schematic Diagram은 AND 게이트의 입력 부분에 각각 NOT 게이트가 연결된 형태이다.

* (A)와 (B)의 Verilog 코딩

|  |  |
| --- | --- |
| (A)’s Design Source Code | (A)’s Simulation Source Code |
| `timescale 1ns / 1ps  module dm1\_a(  input a, b,  output c  );  assign c = ~(a|b);  endmodule | `timescale 1ns / 1ps  module dm1\_a\_tb;  reg aa, bb;  wire c;  dm1\_a u\_dm1\_a (  .a (aa ),  .b (bb ),  .c (c )  );  initial begin  aa = 1'b0;  bb = 1'b0;  end  always aa = #100 ~aa;  always bb = #150 ~bb;  initial begin  #1000  $finish;  end  endmodule |
| (B)’s Design Source Code | (B)’s Simulation Source Code |
| `timescale 1ns / 1ps  module dm1\_b(  input a, b,  output c  );  assign c = (~a)&(~b);  endmodule | `timescale 1ns / 1ps  module dm1\_b\_tb;  reg aa, bb;  wire c;  dm1\_b u\_dm1\_b (  .a (aa ),  .b (bb ),  .c (c )  );  initial begin  aa = 1'b0;  bb = 1'b0;  end  always aa = #100 ~aa;  always bb = #150 ~bb;  initial begin  #1000  $finish;  end  endmodule |

1. Design Source Code 비교

(A)의 모듈은 2개의 입력(a, b)과 1개의 출력(c)을 가지고 있다. c는 a와 b의 OR 연산의 결과에 NOT을 취한 것이다. (B)의 모듈도 마찬가지로 2개의 입력(a, b)과 1개의 출력(c)을 가지고 있다. c는 a와 b에 각각 NOT을 취한 다음 AND 연산한 결과이다.

2. Simulation Source Code 비교

이 코드는 (A)와 (B)가 유사하다. 먼저 각 모듈을 인스턴스화하고, 입력 및 출력 신호를 연결한다. 다음으로 초기 블록(initial)에서 입력 신호 aa, bb를 초기화한다. always 블록에서는 입력 신호 aa, bb를 일정한 주기로 반전시킨다. 마지막 initial 블록에서는 1000ns(나노초) 후에 시뮬레이션을 종료하는 $finish 명령을 실행한다.

* (A)와 (B)의 Simulation을 통해 출력 결과 비교

스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명(A)의 Simulation 결과

스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

(B)의 시뮬레이션 결과

(A)와 (B)를 각각 시뮬레이션 돌린 결과, 모든 입력 경우에 대해 동일한 출력 값을 가진다는 사실을 확인할 수 있었다. 이로부터 드모르간의 제 1 법칙이 실제 회로에 적용됨을 확인했다.

* (A) , (B)를 NOR과 비교

NOR 게이트는 OR 게이트 출력 부분과 NOT 게이트를 연결한 형태로, 이는 (A)의 구조와 동일하다. 또한 위의 시뮬레이션 결과를 통해 (A)와 (B)가 모든 입력 경우에 대해 같은 출력 값을 갖는다는 사실을 확인했다. 따라서 세 논리식 A NOR B = ~(A OR B) = ~A AND ~B가 동일하게 동작한다는 것을 알 수 있다. 추가적으로 진리표를 작성해보면 (A), (B), NOR 게이트의 진리표가 같다는 것을 확인할 수 있다.

**2) De-Morgan의 제 2 법칙**

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* (A)와 (B)의 Schematic 비교

도표, 라인, 그래프, 폰트이(가) 표시된 사진

자동 생성된 설명

(A)의 Schematic Diagram

도표, 라인, 그래프, 평면도이(가) 표시된 사진

자동 생성된 설명

(B)의 Schematic Diagram

(A)의 Schematic Diagram은 AND 게이트 출력 부분에 NOT 게이트가 연결된 형태이고, (B)의 Schematic Diagram은 OR 게이트의 입력 부분에 각각 NOT 게이트가 연결된 형태이다.

* (A)와 (B)의 Verilog 코딩

|  |  |
| --- | --- |
| (A)’s Design Source Code | (A)’s Simulation Source Code |
| `timescale 1ns / 1ps  module dm2\_a(  input a, b,  output c  );  assign c = ~(a&b);  endmodule | `timescale 1ns / 1ps  module dm2\_a\_tb;  reg aa, bb;  wire c;  dm2\_a u\_dm2\_a (  .a (aa ),  .b (bb ),  .c (c )  );  initial begin  aa = 1'b0;  bb = 1'b0;  end  always aa = #100 ~aa;  always bb = #150 ~bb;  initial begin  #1000  $finish;  end  endmodule |
| (B)’s Design Source Code | (B)’s Simulation Source Code |
| `timescale 1ns / 1ps  module dm2\_b(  input a, b,  output c  );  assign c = (~a)|(~b);  endmodule | `timescale 1ns / 1ps  module dm2\_b\_tb;  reg aa, bb;  wire c;  dm2\_b u\_dm2\_b (  .a (aa ),  .b (bb ),  .c (c )  );  initial begin  aa = 1'b0;  bb = 1'b0;  end  always aa = #100 ~aa;  always bb = #150 ~bb;  initial begin  #1000  $finish;  end  endmodule |

1. Design Source Code 비교

(A)의 모듈은 2개의 입력(a, b)과 1개의 출력(c)을 가지고 있다. c는 a와 b의 AND 연산의 결과에 NOT을 취한 것이다. (B)의 모듈도 마찬가지로 2개의 입력(a, b)과 1개의 출력(c)을 가지고 있다. c는 a와 b에 각각 NOT을 취한 다음 OR 연산한 결과이다.

2. Simulation Source Code 비교

이 코드는 (A)와 (B)가 유사하다. 먼저 각 모듈을 인스턴스화하고, 입력 및 출력 신호를 연결한다. 다음으로 초기 블록(initial)에서 입력 신호 aa, bb를 초기화한다. always 블록에서는 입력 신호 aa, bb를 일정한 주기로 반전시킨다. 마지막 initial 블록에서는 1000ns(나노초) 후에 시뮬레이션을 종료하는 $finish 명령을 실행한다.

* (A)와 (B)의 Simulation을 통해 출력 결과 비교

스크린샷, 라인, 사각형이(가) 표시된 사진

자동 생성된 설명

(A)의 Simulation 결과

스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

(B)의 Simulation 결과

(A)와 (B)를 각각 시뮬레이션 돌린 결과, 모든 입력 경우에 대해 동일한 출력 값을 가진다는 사실을 확인할 수 있었다. 이로부터 드모르간의 제 2 법칙이 실제 회로에 적용됨을 확인했다.

* (A) , (B)를 NAND와 비교

NAND 게이트는 AND 게이트 출력 부분과 NOT 게이트를 연결한 형태로, 이는 (A)의 구조와 동일하다. 또한 위의 시뮬레이션 결과를 통해 (A)와 (B)가 모든 입력 경우에 대해 같은 출력 값을 갖는다는 사실을 확인했다. 따라서 세 논리식 A NAND B = ~(A AND B) = ~A OR ~B가 동일하게 동작한다는 것을 알 수 있다. 추가적으로 진리표를 작성해보면 (A), (B), NAND 게이트의 진리표가 같다는 것을 확인할 수 있다.

**3) De-Morgan의 정리 진리표 작성**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | A+B | A∙B | A’ | B’ | (A+B)’ | A’∙B’ | (A∙B)’ | A’+B’ |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**3.**

**1) Boolean Function 1 - (A'+B')\*C' = ((A\*B)+C)'**

* (A)와 (B)의 Schematic 비교

도표, 평면도, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

(A)의 Schematic Diagram

도표, 라인, 폰트, 그래프이(가) 표시된 사진

자동 생성된 설명

(B)의 Schematic Diagram

(A)의 Schematic Diagram은 입력 a, b에 각각 NOT을 취한 다음 이를 OR 게이트로 연결하고, 그 출력 값을 입력 c에 NOT을 취한 결과와 AND 게이트로 연결해 d를 출력하는 형태이다. (B)의 Schematic Diagram은 입력 a, b를 AND 게이트로 연결하고, 그 출력 값을 입력 c와 함께 OR 게이트, NOT 게이트를 순차적으로 통과시켜 d를 출력하는 형태이다.

* (A)와 (B)의 Verilog 코딩

|  |  |
| --- | --- |
| (A)’s Design Source Code | (A)’s Simulation Source Code |
| `timescale 1ns / 1ps  module bf1\_a(  input a, b, c,  output d  );  assign d = (((~a)|(~b))&(~c));  endmodule | `timescale 1ns / 1ps  module bf1\_a\_tb;  reg aa, bb, cc;  wire d;  bf1\_a u\_bf1\_a (  .a (aa ),  .b (bb ),  .c (cc ),  .d (d )  );  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  end  always aa = #100 ~aa;  always bb = #150 ~bb;  always cc = #200 ~cc;  initial begin  #1000  $finish;  end  endmodule |
| (B)’s Design Source Code | (B)’s Simulation Source Code |
| `timescale 1ns / 1ps  module bf1\_b(  input a, b, c,  output d  );  assign d = ~((a&b)|c);  endmodule | `timescale 1ns / 1ps  module bf1\_b\_tb;  reg aa, bb, cc;  wire d;  bf1\_b u\_bf1\_b (  .a (aa ),  .b (bb ),  .c (cc ),  .d (d )  );  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  end  always aa = #100 ~aa;  always bb = #150 ~bb;  always cc = #200 ~cc;  initial begin  #1000  $finish;  end  endmodule |

1. Design Source Code 비교

(A), (B) 두 모듈 모두 3개의 입력(a, b, c)과 1개의 출력(d)을 가지고 있다. (A)의 모듈은 먼저 a와 b 각각 NOT 연산을 한 다음 그 결과를 OR 연산하고, 그 결과 값을 c에 NOT 연산한 결과와 AND 연산 시켜 d를 출력한다. 반면 (B)의 모듈은 (A)에 드모르간 법칙을 적용해 a와 b에 AND 연산을 한 다음, 이를 C와 OR 연산하고 그 결과 값에 NOT 연산을 적용해 d를 출력하도록 한다.

2. Simulation Source Code 비교

이 코드는 (A)와 (B)가 유사하다. 먼저 각 모듈을 인스턴스화하고, 입력 및 출력 신호를 연결한다. 다음으로 초기 블록(initial)에서 입력 신호 aa, bb, cc를 초기화한다. always 블록에서는 입력 신호 aa, bb, cc를 일정한 주기로 반전시킨다. 마지막 initial 블록에서는 1000ns(나노초) 후에 시뮬레이션을 종료하는 $finish 명령을 실행한다.

* (A)와 (B)의 Simulation을 통해 출력 결과 비교

스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

(A)의 Simulation 결과스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

(B)의 Simulation 결과

(A)와 (B)를 각각 시뮬레이션 돌린 결과, 모든 입력 경우에 대해 동일한 출력 값을 가진다는 사실을 확인할 수 있었다. 이로부터 드모르간의 정리가 좀 더 복잡한 논리식으로 구성된 실제 회로에도 적용됨을 알 수 있다.

* Boolean Function 진리표 작성

|  |  |  |  |
| --- | --- | --- | --- |
| IN A | IN B | IN C | OUT D |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

위의 진리표 작성을 통해 입력 a, b, c가 각각 (0, 0, 0), (0, 1, 0), (1, 0, 0)인 경우에만 출력 d가 1이 된다는 것을 알 수 있다. 따라서 boolean function (A'+B')\*C'에서 c는 무조건 0이고, ~a와 ~b의 논리합이 1인 경우, 즉 드모르간 정리에 따라 a와 b의 논리곱이 0인 경우(0, 0), (0, 1), (1, 0)에 출력 값이 1이 된다. 마찬가지로 boolean function ((A\*B)+C)'는 (A\*B)+C가 0이 되어야 하기 때문에 a와 b의 논리곱이 0이고, c도 0인 경우에 출력 값이 1이 된다. 이를 통해 두 논리식이 결국 같은 결과 값을 출력한다는 사실을 확인할 수 있다.

**2) Boolean Function 2 - (A'\*B')+C' = ((A+B)\*C)'**

* (A)와 (B)의 Schematic 비교

도표, 라인, 평면도, 그래프이(가) 표시된 사진

자동 생성된 설명

(A)의 Schematic Diagram

도표, 라인, 그래프, 평면도이(가) 표시된 사진

자동 생성된 설명

(B)의 Schematic Diagram

(A)의 Schematic Diagram은 입력 a, b에 각각 NOT을 취한 다음 이를 AND 게이트로 연결하고, 그 출력 값을 입력 c에 NOT을 취한 결과와 OR 게이트로 연결해 d를 출력하는 형태이다. (B)의 Schematic Diagram은 입력 a, b를 OR 게이트로 연결하고, 그 출력 값을 입력 c와 함께 AND 게이트, NOT 게이트를 순차적으로 통과시켜 d를 출력하는 형태이다.

* (A)와 (B)의 Verilog 코딩

|  |  |
| --- | --- |
| (A)’s Design Source Code | (A)’s Simulation Source Code |
| `timescale 1ns / 1ps  module bf2\_a(  input a, b, c,  output d  );  assign d = (((~a)&(~b))|(~c));  endmodule | `timescale 1ns / 1ps  module bf2\_a\_tb;  reg aa, bb, cc;  wire d;  bf2\_a u\_bf2\_a (  .a (aa ),  .b (bb ),  .c (cc ),  .d (d )  );  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  end  always aa = #100 ~aa;  always bb = #150 ~bb;  always cc = #200 ~cc;  initial begin  #1000  $finish;  end  endmodule |
| (B)’s Design Source Code | (B)’s Simulation Source Code |
| `timescale 1ns / 1ps  module bf2\_b(  input a, b, c,  output d  );  assign d = ~((a|b)&c);  endmodule | `timescale 1ns / 1ps  module bf2\_b\_tb;  reg aa, bb, cc;  wire d;  bf2\_b u\_bf2\_b (  .a (aa ),  .b (bb ),  .c (cc ),  .d (d )  );  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  end  always aa = #100 ~aa;  always bb = #150 ~bb;  always cc = #200 ~cc;  initial begin  #1000  $finish;  end  endmodule |

1. Design Source Code 비교

(A), (B) 두 모듈 모두 3개의 입력(a, b, c)과 1개의 출력(d)을 가지고 있다. (A)의 모듈은 먼저 a와 b 각각 NOT 연산을 한 다음 그 결과를 AND 연산하고, 그 결과 값을 c에 NOT 연산한 결과와 OR 연산 시켜 d를 출력한다. 반면 (B)의 모듈은 (A)에 드모르간 법칙을 적용해 a와 b에 OR 연산을 한 다음, 이를 C와 AND 연산하고 그 결과 값에 NOT 연산을 적용해 d를 출력하도록 한다.

2. Simulation Source Code 비교

이 코드는 (A)와 (B)가 유사하다. 먼저 각 모듈을 인스턴스화하고, 입력 및 출력 신호를 연결한다. 다음으로 초기 블록(initial)에서 입력 신호 aa, bb, cc를 초기화한다. always 블록에서는 입력 신호 aa, bb, cc를 일정한 주기로 반전시킨다. 마지막 initial 블록에서는 1000ns(나노초) 후에 시뮬레이션을 종료하는 $finish 명령을 실행한다.

* (A)와 (B)의 Simulation을 통해 출력 결과 비교

스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

(A)의 Simulation 결과스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

(B)의 Simulation 결과

(A)와 (B)를 각각 시뮬레이션 돌린 결과, 모든 입력 경우에 대해 동일한 출력 값을 가진다는 사실을 확인할 수 있었다. 이로부터 드모르간의 정리가 좀 더 복잡한 논리식으로 구성된 실제 회로에도 적용됨을 알 수 있다.

* Boolean Function 진리표 작성

|  |  |  |  |
| --- | --- | --- | --- |
| IN A | IN B | IN C | OUT D |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

위의 진리표 작성을 통해 입력 a, b, c가 각각 (0, 1, 1), (1, 0, 1), (1, 1, 1)인 경우에만 출력 d가 0이 된다는 것을 알 수 있다. 따라서 boolean function (A'\*B')+C'에서 c는 무조건 1이고, ~a와 ~b의 논리곱이 0인 경우, 즉 드모르간 정리에 따라 a와 b의 논리합이 1인 경우(0, 1), (1, 0), (1, 1)에 출력 값이 0이 된다. 마찬가지로 boolean function ((A+B)\*C)'는 (A+B)\*C가 1이 되어야 하기 때문에 a와 b의 논리합이 1이고, c도 1인 경우에 출력 값이 0이 된다. 이를 통해 두 논리식이 결국 같은 결과 값을 출력한다는 사실을 확인할 수 있다.

**4.**

**1Bit 비교기(2 input, 4 output)**

* Schematic 작성

도표, 평면도, 라인, 기술 도면이(가) 표시된 사진

자동 생성된 설명

1 Bit 비교기의 Schematic Diagram은 위와 같이 나타난다. 출력 c는 먼저 입력 a와 b를 XOR 게이트에 연결한 다음 그 결과에 NOT 게이트를 적용하여 얻어진다. 출력 d는 입력 a와 b를 XOR 게이트에 연결하여 얻어진다. 출력 e는 입력 b를 NOT 게이트에 연결한 다음 입력 a와 AND 게이트에 연결하여 얻어진다. 마지막으로 출력 f는 입력 a를 NOT 게이트에 연결한 다음 입력 b와 AND 게이트에 연결하여 얻어진다. 각 출력은 그것이 의미하는 비교 연산자에 따라 값이 다른데, 출력 c는 A=B인 경우 1이 되고, 출력 d는 A≠B인 경우 1이 되고, 출력 e는 A>B인 경우 1이 되고, 출력 f는 A<B인 경우 1이 되도록 설계했다.

* Verilog 코딩

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `timescale 1ns / 1ps  module cmp(  input a, b,  output c, d, e, f  );  assign c = ~(a^b);  assign d = a^b;  assign e = a&~b;  assign f = ~a&b;  endmodule | `timescale 1ns / 1ps  module cmp\_tb;  reg aa, bb;  wire c, d, e, f;  cmp u\_cmp (  .a (aa ),  .b (bb ),  .c (c ),  .d (d ),  .e (e ),  .f (f )  );  initial begin  aa = 1'b0;  bb = 1'b0;  end  always aa = #100 ~aa;  always bb = #150 ~bb;  initial begin  #1000  $finish;  end  endmodule |

1. Design Source Code

cmp 모듈은 2개의 입력(a, b)과 4개의 출력(c, d, e, f)을 가지고 있다. 입력 a, b를 XNOR 연산해 c를 출력하고, 출력 d는 입력 a, b를 XOR 연산한 결과이다. 출력 e는 입력 b에 NOT을 취한 다음 입력 a와 AND 연산한 결과이고, 출력 f는 입력 a에 NOT을 취한 다음 입력 b와 AND 연산한 결과이다.

2. Simulation Source Code

먼저 각 모듈을 인스턴스화하고, 입력 및 출력 신호를 연결한다. 다음으로 초기 블록(initial)에서 입력 신호 aa, bb를 초기화한다. always 블록에서는 입력 신호 aa, bb를 일정한 주기로 반전시킨다. 마지막 initial 블록에서는 1000ns(나노초) 후에 시뮬레이션을 종료하는 $finish 명령을 실행한다.

* Simulation 결과

스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a, b의 값을 바꾸면 그에 따라 출력 c, d, e, f의 값이 변화하는 것을 확인할 수 있다. 출력 c는 입력 a, b의 값이 같은 경우 1을 출력하고, 반대로 출력 d는 입력 a, b가 다른 경우 1을 출력한다. 출력 e는 입력 a가 1, 입력 b가 0인 경우, 즉 입력 a 값이 더 큰 경우 1을 출력하고, 출력 f는 입력 a가 0, 입력 b가 1인 경우, 즉 입력 b의 값이 더 큰 경우 1을 출력한다는 사실을 알 수 있다. 시뮬레이션 결과를 진리표로 정리한 결과는 다음과 같다.

* 진리표

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | A=B | A≠B | A>B | A<B |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

**5.**

이번 실험에서는 드모르간 제 1 법칙, 제 2 법칙을 실제 회로에 적용해보고, 좀 더 복잡한 논리식에 관해서도 드모르간 법칙에 따라 어떻게 동작하는지 알아보았다. 이를 통해 하나의 논리식에 드모르간 법칙을 적용하면 같은 결과 값을 내놓는 다양한 논리식으로 변형이 가능하다는 사실을 알게 되었다. 또한 1 bit 비교기 실습에서 주어진 진리표를 보고 이를 논리식으로 구현해 기능에 맞게 동작할 수 있도록 회로를 설계하는 방법을 알게 되었다. 특히 두 입력 값의 크기 비교 기능을 >, < 연산자가 아닌 논리 연산자로 구현할 수 있다는 사실이 새로웠다.

**6.**

**2 Bit 비교기**

2 비트 비교기는 두 개의 2 비트 입력을 비교하고, 입력 A와 입력 B가 서로 같은지 또는 어떤 입력이 다른 입력보다 큰지를 판별하는 논리 회로이다. 2 비트 비교기의 주요 기능은 다음과 같다.

* A와 B가 같은지 여부를 판별: 2 비트 입력 A와 B를 비교하여 두 입력이 서로 같은지 여부를 판단한다. 이는 두 입력 비트가 동일한지 비교하는 논리 연산을 수행하고 결과는 "같음" 또는 "다름"으로 나타난다.
* A가 B보다 큰지 여부를 판별: 두 2 비트 입력 A와 B를 비교하여 A가 B보다 큰지 여부를 판단한다. 이는 A와 B를 비교하여 A가 B보다 크거나 같은지를 확인하는 논리 연산을 수행하며 결과는 "A가 B보다 큼" 또는 "A가 B와 같거나 작음"으로 나타난다.

1 비트 비교기와 2 비트 비교기는 비교하는 비트의 개수와 비교 결과를 나타내는 출력 비트의 개수에서 차이점이 있다. 이 때문에 논리 회로의 복잡도도 상이한데, 2 비트 비교기는 두 개의 2 비트 입력을 동시에 처리하므로, 일반적으로 더 복잡한 논리 회로가 필요하다. 1 비트 비교기는 하나의 비트만을 처리하므로 논리 회로가 상대적으로 간단하며, 디자인과 구현이 더 쉽다. 따라서 1 비트 비교기는 두 개의 입력을 단순하게 비교할 때 사용되며, 2 비트 비교기는 더 복잡한 데이터 비교 및 분석에 사용된다.

2 비트 비교기를 구현하려면, 두 개의 2 비트 입력을 비교하기 위한 논리 회로를 설계해야 한다. 이런 비교기는 일반적으로 AND, OR, NOT, XOR 게이트 등의 기본 논리 게이트를 사용하여 구성되며, 비트별로 입력을 비교하고 결과를 조합한다는 특징이 있다. 2 비트 비교기의 진리표는 다음과 같다.

* 진리표

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 입력 | | 출력 | | | |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 0 0 | 0 0 | 1 | 0 | 0 | 0 |
| 0 1 | 0 | 1 | 0 | 1 |
| 1 0 | 0 | 1 | 0 | 1 |
| 1 1 | 0 | 1 | 0 | 1 |
| 0 1 | 0 0 | 0 | 1 | 1 | 0 |
| 0 1 | 1 | 0 | 0 | 0 |
| 1 0 | 0 | 1 | 0 | 1 |
| 1 1 | 0 | 1 | 0 | 1 |
| 1 0 | 0 0 | 0 | 1 | 1 | 0 |
| 0 1 | 0 | 1 | 1 | 0 |
| 1 0 | 1 | 0 | 0 | 0 |
| 1 1 | 0 | 1 | 0 | 1 |
| 1 1 | 0 0 | 0 | 1 | 1 | 0 |
| 0 1 | 0 | 1 | 1 | 0 |
| 1 0 | 0 | 1 | 1 | 0 |
| 1 1 | 1 | 0 | 0 | 0 |