4주차 결과보고서

전공: 아트엔테크놀로지 학년: 3학년 학번: 20191172 이름: 함승우

1.

AND/OR/NOT 연산에 이어서 NAND, NOR, XOR 게이트의 동작을 이해하고, Verilog와 FPGA를 통해서 그 동작을 눈으로 확인할 수 있다.

2. 4-input NAND gate

도표, 라인, 폰트, 평면도이(가) 표시된 사진

자동 생성된 설명

* Boolean식

(B)의 경우의 Boolean식을 먼저 설명하자면, NAND의 연산은 NOT(A AND B)으로 연산할 수 있기 때문에 아래와 같다.

e = ~(a & b)

f = ~(c&e);

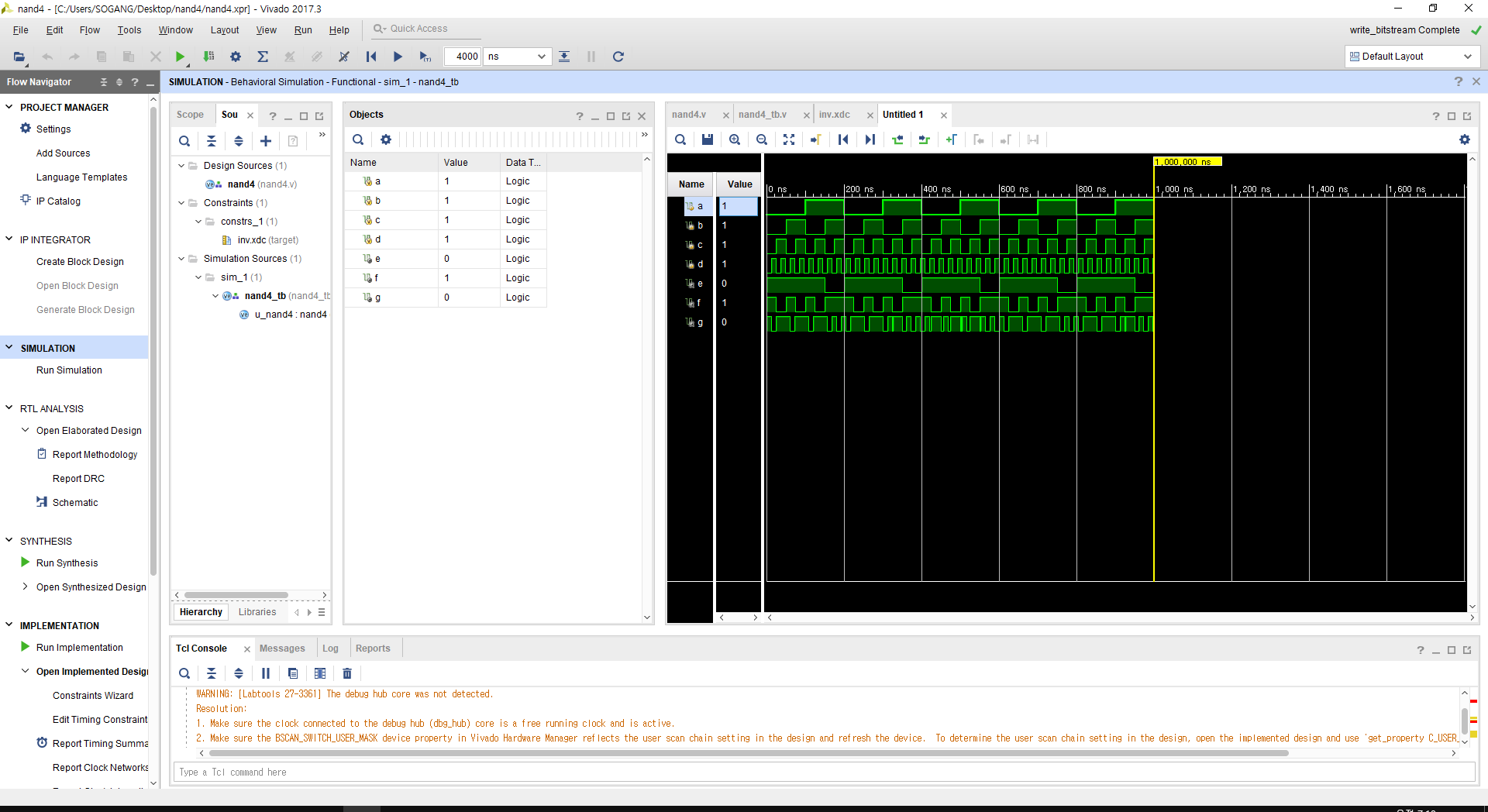
g = ~(d&f);

* Verilog 코딩

HDL을 사용해서 회로를 만들기 위해서는 v파일과 이 v파일을 simulation할 수 있는 testbench 파일을 코딩했다.

|  |  |
| --- | --- |
| Nand 코딩 | Tb 코딩 |
| ‘timescale 1ns / 1ps  module nand4(  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 nand4\_tb;  reg a, b, c, d;  wire e, f, g;  nand4 u\_nand4(  .a(a), .b(b), .c(c), .d(d),  .e(e), .f(f), .g(g)  );  initial a = 1’b0;  initial b = 1’b0;  initial c = 1’b0;  initial d = 1’b0;  always a = #100 ~a;  always b = #50 ~b;  always c = #25 ~c;  always d = #12 ~d;  initial begin  #1000  $finish  end  endmodule |

Simulation 결과



위 v파일은 4개의 Input값 a, b, c, d가 입력이 되고, output e, f, g가 출력이 된다. 이 때 a, b 사이의 NAND의 연산을 통해서 e가 출력된다. 이 출력 값 e와 c 사이의 NAND 연산을 통해서 f가 출력되고, 마찬가지로 이 f와 d의 NAND 연산을 통해서 g가 출력된다.

위 테스트벤치를 해석하자면, 4개의 레지스터를 설정한다(reg a, b, c, d), 그 후 세 개의 와이어를 정의한다 (wire e, f, g), 이 와이어는 보통 출력 값을 저장하고, 이는 시뮬레이션 테스트 결과값으로 사용된다. initial a = 1’b0 ~ initial d = 1’b0는 a, b, c, d의 초기값을 0으로 설정하고, 그 밑의 always a = #100 ~ a ~ always d = #12 ~d의 구문 각각 레지스터의 루프를 정의한 부분인데, 각각 쓰인 시간 딜레이 후에 해당 입력을 반전시킨다. 예를 들자면 a의 경우 #100(100ns) 후에 반전시킨다. 마지막으로 1000ns까지 진행하고 종료한다.

진리표 작성

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out E | Out F | Out G |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

(A) 즉 4개의 input을 가지지만 1개의 output을 갖는 경우에는 위 3가지의 output을 갖는 경우와 결과가 다르다. (A)의 경우를 Boolean 식으로 나타내자면,

assign e = ~(a&b&c&d);이다.

(A)의 경우에는 NOT(abcd)이다. 이는 NOT a + NOT b + NOT c + NOT d이다.

(B)의 경우에는 NOT(NOT(NOT(ab)c)d)이다. 이는 NOT(NOT(NOT a + NOT b) + NOT c) + NOT d)이다. (A)의 경우에 진리표만 작성해보자면,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out E |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 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 | 0 |

그러므로 AND 연산과는 다르게 3개의 output을 가지는 경우와, 1개의 output만 가지는 경우는 다르다.

3. 4-input NOR gate

도표, 스케치, 라인, 그림이(가) 표시된 사진

자동 생성된 설명

* Boolean 식

assign e = ~(a|b);

assign f = ~(c|e);

assign g = ~(d|f);

NOR의 식의 경우에는 OR 연산을 취한 뒤에 NOT 연산을 취한 것과 같기 때문에 위와 같이 나타낼 수 있다.

* Verilog 코딩

위의 테스트벤치와 동일한 코드를 사용할 것이기 때문에 테스트벤치의 설명은 생략하고 v파일에 대하여 작성하자면

‘timescale 1ns / 1ps

module nor4(

input a,b,c,d,

output e,f,g

);

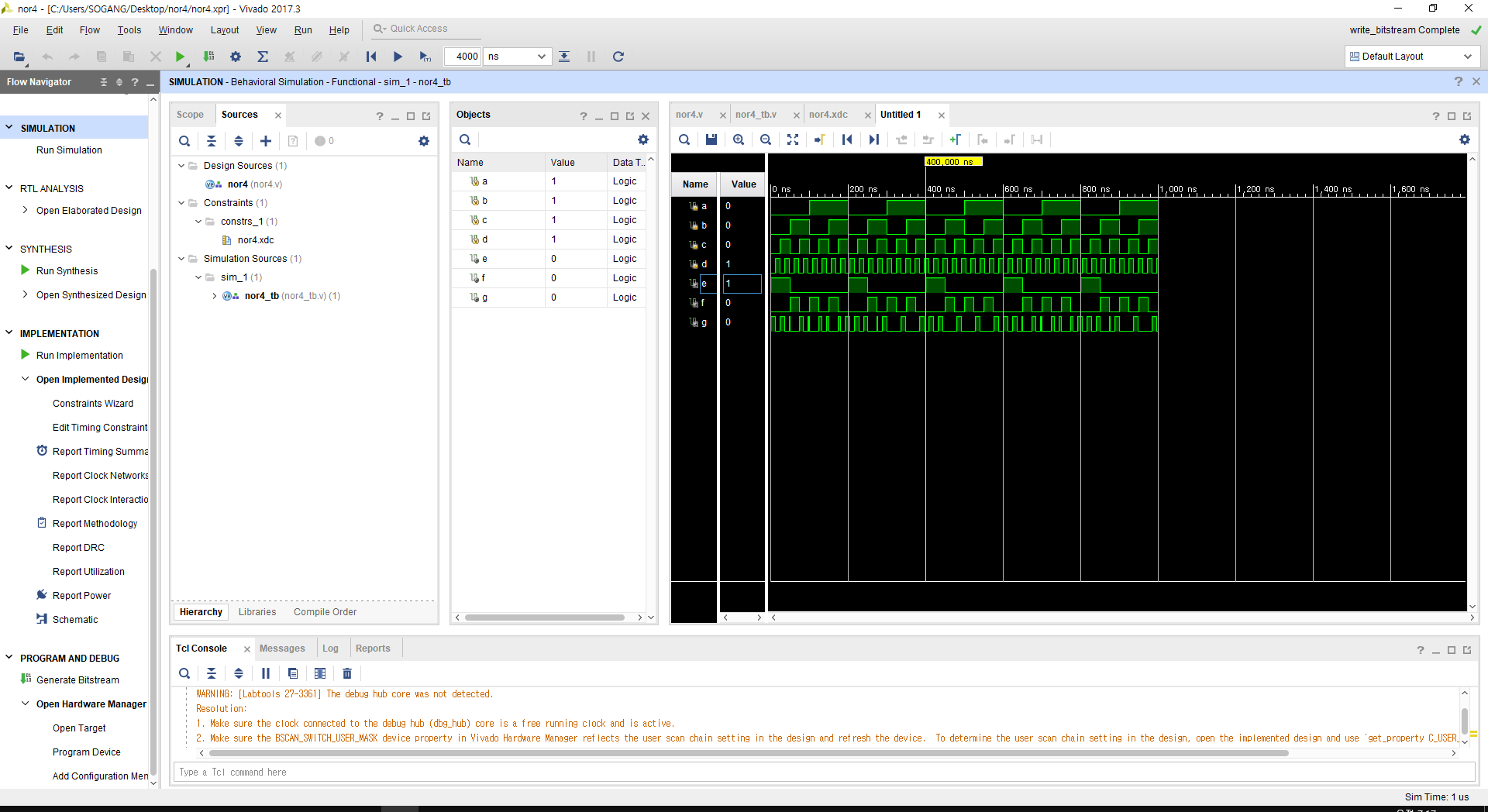
assign e = ~(a|b);

assign f = ~(c|e);

assign g = ~(d|f);

endmodule

* simulation 결과



위 v파일은 4개의 Input값 a, b, c, d가 입력이 되고, output e, f, g가 출력이 된다. 이 때 a, b 사이의 NOR의 연산을 통해서 e가 출력된다. 이 출력 값 e와 c 사이의 NOR 연산을 통해서 f가 출력되고, 마찬가지로 이 f와 d의 NOR 연산을 통해서 g가 출력된다.

* 진리표 작성

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out E | Out F | Out G |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 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 | 1 |
| 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 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

(A)의 경우에는 NOT(a|b|c|d)이고, (B)의 경우에는 NOT(NOT(a|b)|c)|d)이다.

(A)의 결과를 진리표로 작성해보았다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out E |
| 0 | 0 | 0 | 0 | 1 |
| 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 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

4. 4-input XOR gate

도표, 스케치, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

위와 마찬가지로 (A)의 경우에는 A,B,C,D를 한번에 입력받고 하나의 output인 E를 출력한다. (B)는 A,B,C,D를 입력 받지만, A,B를 XOR 연산을 통해 E를 출력하고, 이 출력 값 E와 C를 XOR 연산을 통해 F를 출력한다. 이 F는 다시 D와 XOR 연산을 통해서 G라는 최종 출력 값을 출력한다.

* Boolean 식

(A)의 경우에는 assign e = a^b^c^d;

(B)의 경우에는

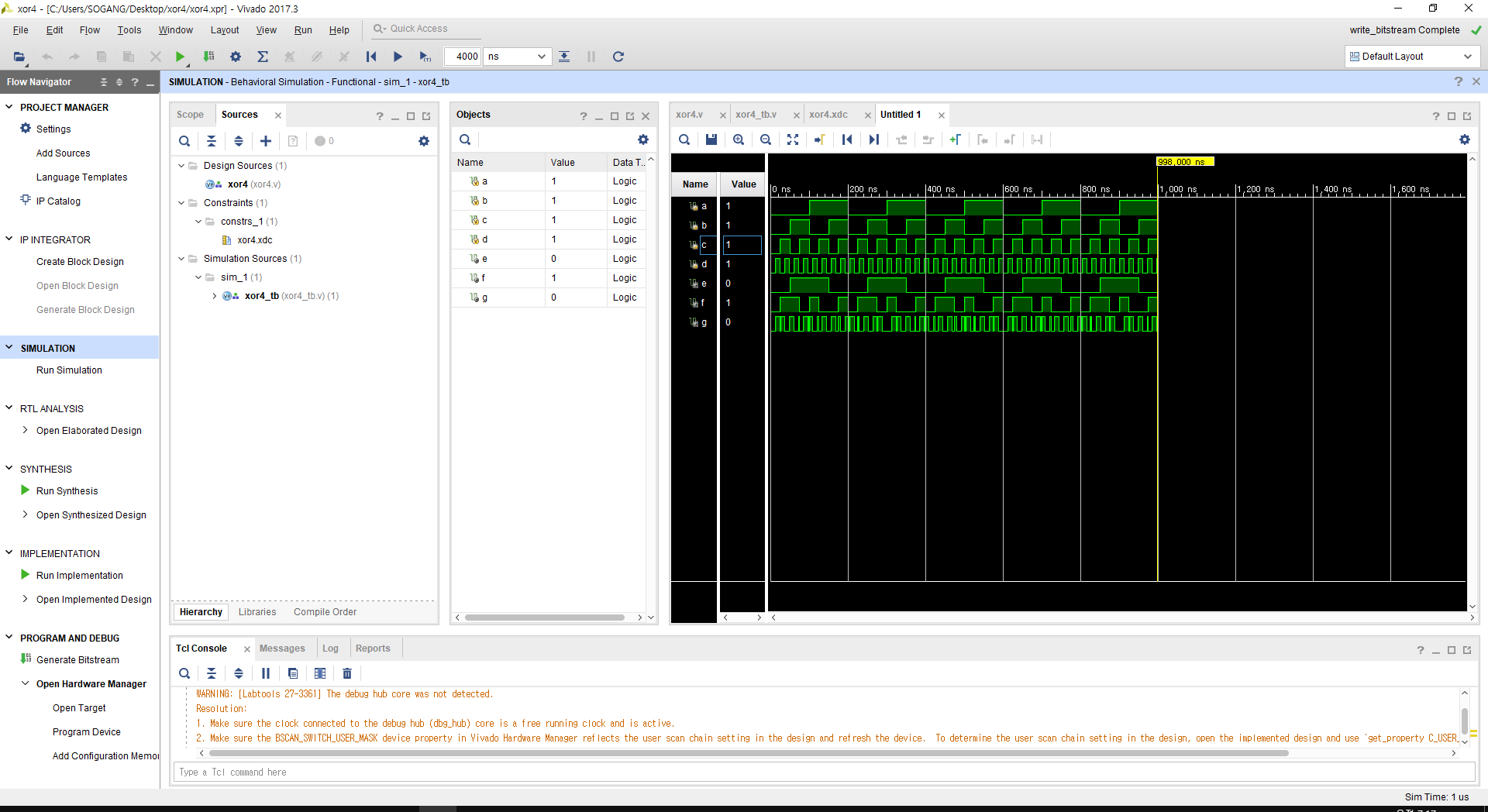
assign e = a^b;

assign f = e^c;

assign g = f^d;

이다.

* Verilog 코딩
* ‘timescale 1ns / 1ps
* module xor4(
* input a,b,c,d,
* output e,f,g
* );
* assign e = a^b;
* assign f = c^e;
* assign g = d^f;
* endmodule
* Simulation 결과



NAND, NOR 연산과 같은 형식의 테스트벤치 파일을 사용했기 때문에, a는 100ns, b는 50ns, c는 25ns, d는 12ns 마다 값이 반전되는 것을 알 수 있다. 이 시뮬레이션 결과에 따라서 진리표 작성을 했다.

* 진리표 작성

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

(A)의 경우에는 a^b^c^d를 한번에 연산하게 되는데, 이를 진리표로 간단하게 나타낸다면

- (A)의 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out E |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

XOR 연산의 경우에는 input에서 1의 개수가 홀수면 Output이 1이되고, 1의 개수가 짝수면 0을 출력하는 것을 알 수 있다.

5. 4-input AOI gate

도표, 스케치, 라인, 기술 도면이(가) 표시된 사진

자동 생성된 설명

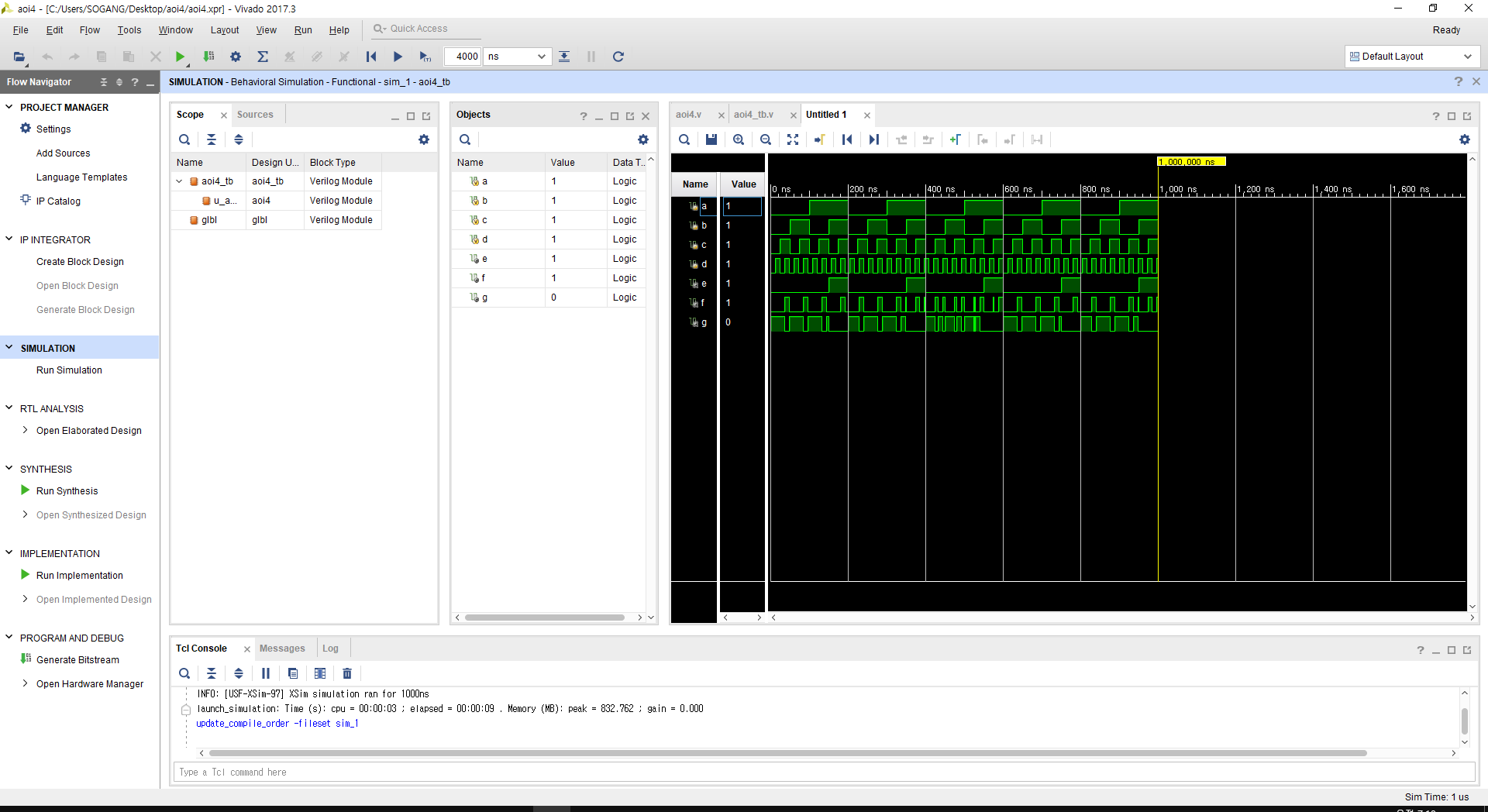
AOI gate란 여러 개의 입력을 받아서 그 입력들을 AND 연산 후, 그 결과를 OR 연산하고, 마지막으로 인버트(NOT) 연산을 하는 게이트이다.

실습을 진행한 경우를 예를 들어본다면, 위 그림도 마찬가지로 a,b를 AND 연산하고 e를 출력한다. c와 d또한 AND 연산하고 f를 출력한다. 이 e와 f를 OR 연산후 Invert한다.

* Verilog 코딩
* ‘timescale 1ns / 1ps
* module aoi4(
* input a,b,c,d,
* output e,f,g
* );
* assign e =a&b;
* assign f = c&d;
* assign g = ~(e|f);
* endmodule

e와 f는 각각 a, b의 and 연산, c와 d의 and 연산, g는 이 e와 f의 OR 연산 후 NOR 연산을 취했다.

* Simulation 결과



위 결과를 토대로 진리표를 작성했다.

* 진리표 작성

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

6. 결과 검토 및 논의사항.

첫번째로, NAND와 NOR 게이트는 각각 AND연산, OR 연산 후에 NOT(인버트)를 취했지만, NAND과 NOR 게이트는 AND와 OR 게이트와 3 output, 1 output의 경우에 각각 달랐다.

AND와 OR gate의 경우에 boolean 식으로 정리해보겠다.

3 output일 때의 AND 연산

e = a &b, f = e & c, g = f & d인데, g = ((a & b) & c) & d라고 할 수 있다.

1 output일 때의 AND 연산

e = a & b & c &d이다.

따라서 3 output일 때와 1 output일 때의 AND 연산은 최종적으로 출력하는 값에는 차이가 없다.

3 output일 때의 OR 연산

e = a | b, f = e | c. g = f | d인데, g= ((a | b) | c) | d라고 할 수 있다.

1 output일 때의 OR 연산

e = a|b|c|d이다.

마찬가지로 3 output일 때와 1 output일 때의 OR 연산은 최종적으로 출력하는 값에는 차이가 없다.

하지만, 3 output일 때의 NAND 연산은

e = ~(a&b) , f = ~(e&c), g = ~(f & d)이다.

g = ~((~((~(a&b))&c)) &d)

g = ~(~(~a + ~b) + ~c) + ~d)이다.

이는 1 output일 때의 NAND인

e = ~(a&b&c&d)

e = ~a+ ~b + ~c + ~d와는 다르다.

NAND와 NOR 연산은 우리가 기존에 알고있던 논리 구성 방식이 다르기 때문에 논리 회로 설계 과정에서 충분히 유의하여 작성하여야 한다. 하지만, XOR의 경우에는 차이가 없었다. 이는 AND, OR, XOR 게이트만을 사용하여는 모든 논리회로를 나타내는 것이 불가능하지만, NAND, NOR 같은 모든 논리회로를 표현 가능한 범용 게이트를 사용하여 정밀하고 세밀한 논리회로를 설계할 수 있다.

7. 추가 이론 조사 및 작성

XOR의 경우에는 ^라는 연산자를 사용하여 표현할 수 있었는데, 논리합과 논리곱만을 사용하여 표현할 수도 있다. XOR이 X와 Y를 연산한다고 했을 때, X ^ Y라고 할 수도 있고, XY’ + X’Y라고 표현할 수도 있다. NOR과 NAND와 마찬가지로 Inverter를 씌운 XNOR 연산을 생각해 볼 수 있는데, XNOR의 경우에는 논리 연산식으로 표현했을 경우에는

X XNOR Y = (X XOR Y)’ = (XY’ + X’Y)’ = (X’ + Y)(X + Y’) = XY + X’Y’이다. XNOR의 경우에는 입력 값이 모두 동일해야만 1을 출력한다.