전공: 컴퓨터공학 학년: 2학년 학번: 20161603 이름: 신민준

1. 실험 목적

이 실험에서는 NAND, NOR, XOR, AOI gate를 구현하고, 각각의 동작을 이해하며 확인하는데 목표가 있다. 먼저, Verilog로 다중입력 NAND, NOR, XOR, AOI gate를 구현하고, 입력 신호를 생성하는 simulation을 돌려 구현한 결과를 확인한다. 이후, 실제 FPGA를 통해 구현한 회로가 정상적으로 동작함을 확인한다.

1. 4-input NAND gate의 simulation 결과 및 과정에 대해서 설명하시오(4 input, 3 output).

NAND gate를 위해 생성한 코드는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module **nandga**(      input a, b, c, d,      output e      );      assign e = ~(a&b&c&d);  endmodule  module **nandgb**(      input a, b, c, d,      output e, f, g      );      assign e = ~(a&b);      assign f = ~(e&c);      assign g = ~(f&d);  endmodule |

이를 RTL Schematic으로 확인하면 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

Simulation을 위해 생성한 simulation code는 다음과 같이 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module **nandsim**;      reg [3:0] in;      wire out, e, f, g;      nandga gateA(in[0], in[1], in[2], in[3], out);      nandgb gateB(in[0], in[1], in[2], in[3], e, f, g);      initial in = 4'b0000;      always in = #50 in + 1;      initial begin          #20000  **$finish**;      end  endmodule |

Simulation 결과는 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

이를 바탕으로 진리표를 작성하면 다음과 같은 진리표가 완성된다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

1. 4-input NOR gate의 simulation 결과 및 과정에 대해서 설명하시오(4 input, 3 output).

NOR gate를 위해 생성한 코드는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module **norga**(      input a, b, c, d,      output e      );      assign e = ~(a|b|c|d);  endmodule  module **norgb**(      input a, b, c, d,      output e, f, g      );      assign e = ~(a|b);      assign f = ~(e|c);      assign g = ~(f|d);  endmodule |

이를 RTL Schematic으로 확인하면 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

Simulation을 위해 생성한 simulation code는 다음과 같이 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module **norsim**;      reg [3:0] in;      wire out, e, f, g;      norga gateA(in[0], in[1], in[2], in[3], out);      norgb gateB(in[0], in[1], in[2], in[3], e, f, g);        initial in = 4'b0000;      always in = #50 in + 1;      initial begin          #20000  **$finish**;      end  endmodule |

Simulation 결과는 다음과 같다.

스크린샷, 전자기기이(가) 표시된 사진

자동 생성된 설명

이를 바탕으로 진리표를 작성하면 다음과 같은 진리표가 완성된다.

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

1. 4-input XOR gate의 simulation 결과 및 과정에 대해서 설명하시오(4 input, 3 output).

XOR gate를 위해 생성한 코드는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module **xorga**(      input a, b, c, d,      output e      );      assign e = a^b^c^d;  endmodule  module **xorgb**(      input a, b, c, d,      output e, f, g      );      assign e = a^b;      assign f = e^c;      assign g = f^d;  endmodule |

이를 RTL Schematic으로 확인하면 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

Simulation을 위해 생성한 simulation code는 다음과 같이 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module **xorsim**;      reg [3:0] in;      wire out, e, f, g;      xorga gateA(in[0], in[1], in[2], in[3], out);      xorgb gateB(in[0], in[1], in[2], in[3], e, f, g);        initial in = 4'b0000;      always in = #50 in + 1;      initial begin          #20000  **$finish**;      end  endmodule |

Simulation 결과는 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

이를 바탕으로 진리표를 작성하면 다음과 같은 진리표가 완성된다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

1. 4-input AOI gate의 simulation 결과 및 과정에 대해서 설명하시오(4 input, 3 output).

AOI gate를 위해 생성한 코드는 다음과 같다.

|  |
| --- |
| `timescale 1ns / 1ps  module **aoi**(      input a, b, c, d,      output e      );      assign e = ~((a&b)|(c&d));  endmodule |

이를 RTL Schematic으로 확인하면 다음과 같다.

스크린샷, 노트북, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

Simulation을 위해 생성한 simulation code는 다음과 같이 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module **aoisim**;      reg [3:0] in;      wire out;      aoi gateA(in[0], in[1], in[2], in[3], out);        initial in = 4'b0000;      always in = #50 in + 1;      initial begin          #20000  **$finish**;      end  endmodule |

Simulation 결과는 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

이를 바탕으로 진리표를 작성하면 다음과 같은 진리표가 완성된다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 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 |

1. 결과 검토 및 논의 사항.

결과는 모두 다 예상했던 대로 나와주어, Verilog code로 구현한 방식이 옳았음을 알 수 있었다.

특이한 점은, NAND, NOR, XOR 연산 중 Associative law가 성립하는 연산은 XOR 뿐이었다는 점이다. NAND, NOR은 , 인 반면, XOR Logic은 가 성립했다. 따라서, XOR Logic을 정리할 때 유용하게 이 성질을 사용할 수 있을 것이다.

1. 추가 이론 조사 및 작성.

AND-OR-Invert logic은 다수의 AND gate의 output들을 NOR gate로 묶어놓는 회로이다. 이 회로는 CMOS 구현 방식에서 각 게이트들을 따로 구현하고 합치는 것 보다 전체 회로에서 사용하는 트랜지스터의 수가 적다는 점에서 장점이 있다.