4주차 결과보고서

전공 : 국제한국학과 학년 : 4학년 학번 : 20181202 이름 : 김수미

**1. 실험 목적**

- NAND/NOR/XOR 게이트의 동작을 이해하고 직접 Verilog 코드를 작성해 봄으로써  
동작 원리를 이해한다.

- Vivado 프로그램에서 Verilog 언어를 사용하여 직접 논리식을 이용한 코드를 작성해  
봄으로써 다중입력 NAND/NOR 게이트 및 XOR 게이트를 구현해 본다.

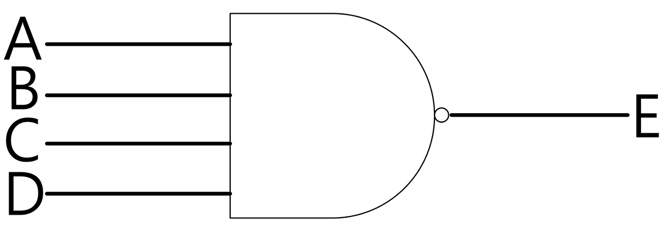
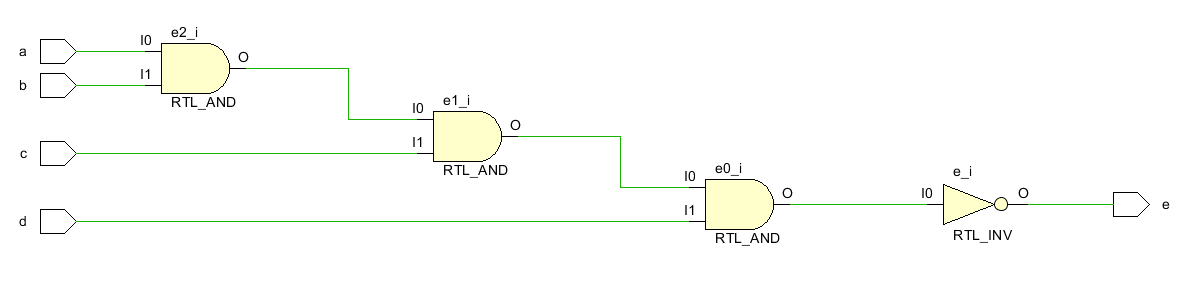
- 입력 신호 생성 후 Simulation을 통해 구현된 각 게이트가 여러 input에 대해 어떻게 동작하는지를 확인한다.

- FPGA를 통해 Verilog로 구현된 회로의 동작을 확인한다. (강의자료에는 실험 목표로써 기재되어 있으나 코로나 비대면 수업으로 인해 FPGA 실습은 진행하지 않았다).

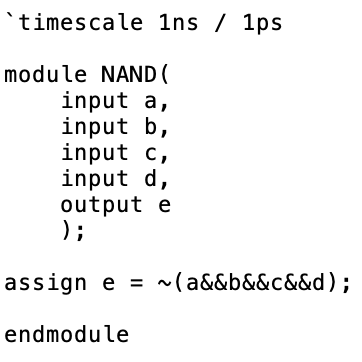
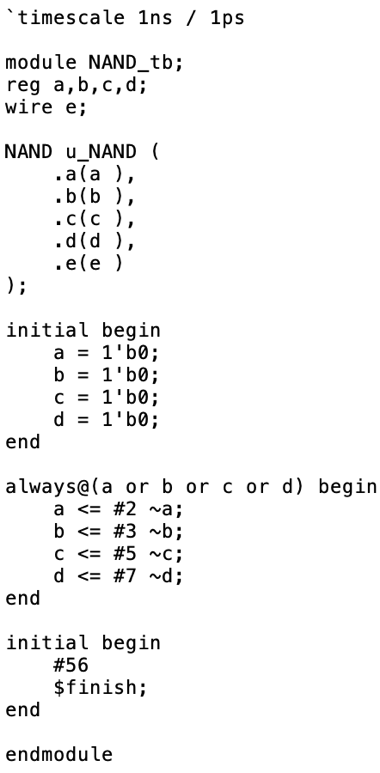
**2. 4-input NAND gate의 simulation 결과 및 과정에 대해서 설명하시오.**

**(A) 4-input, 1-output NAND gate**

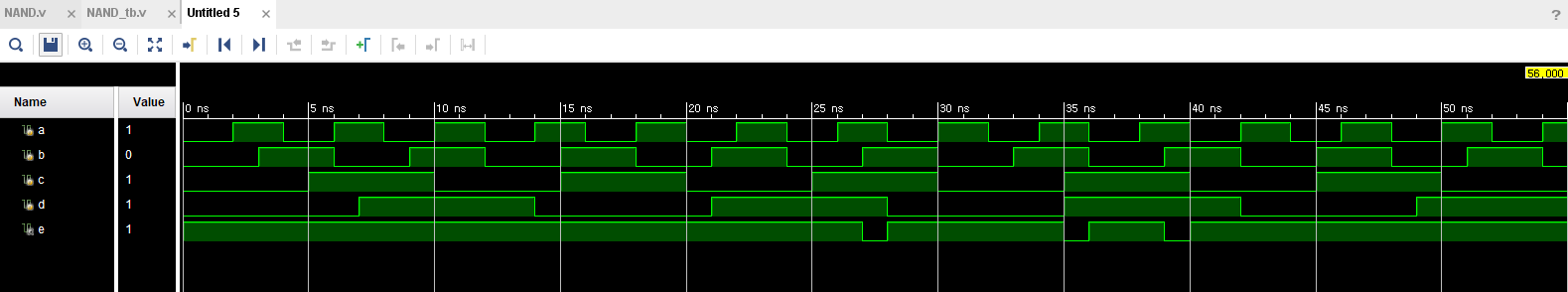
**1) Schematic (강의자료 / Vivado)**



**2) Source Code (Design Source / Test Bench)**



**3) Simulation**



**4) Truth Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input A | Input B | Input C | Input D | Output 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 |

**5) 구현 과정 설명**

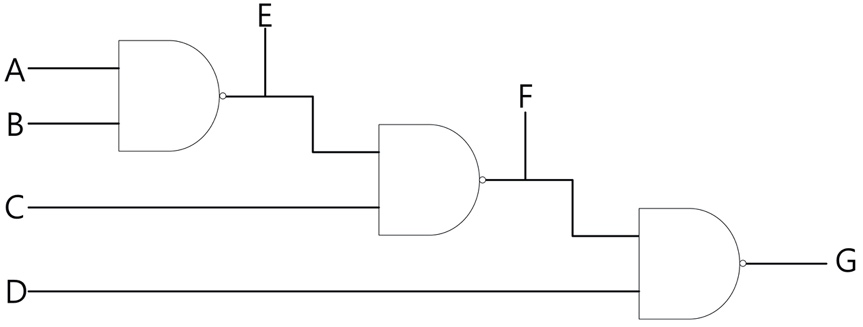
4-input, 1-output NAND 게이트의 경우 Input A,B,C,D 와 output E가 있을 때 A(nand)B(nand)C(nand)D 의 값이 Output E에 할당된다. 이때 4개의 input이 모두 1일 경우 0이 Output으로 반환되고, 그 외의 경우에는 모두 1이 Output으로 반환된다. 이는 4개의 input이 모두 1일때만 1을 반환하고 그 외의 경우에는 0을 반환하는 AND 게이트와 정 반대의 양상을 보이는데, NAND 게이트는 AND 게이트에 NOT 게이트를 붙인 것이기 때문이다. 따라서 AND 게이트와 비교했을 때 NAND 게이트의 truth table은 Output E 부분이 정 반대의 값을 가지는 것을 확인할 수 있다.

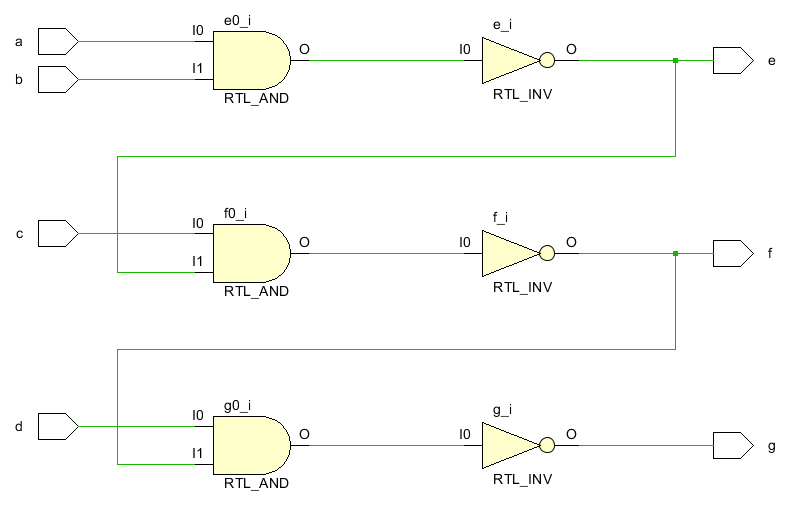
Verilog 코드는 모든 input을 AND로 연결한 다음 바깥에 not을 붙이는 식으로 작성하였다 : assign **e** = ~(**a**&&**b**&&**c**&&**d**);

강의자료의 Schematic과 Vivado에서 생성된 Schematic의 모양이 다른데, 강의자료의 Schematic에서는 4개의 input이 하나의 NAND 게이트를 통과하지만 Vivado에서 생성한 Schematic에서는 4개의 input이 차례대로 3개의 AND 게이트를 통과한 다음 마지막으로 NOT 게이트를 통과하는 것을 확인할 수 있다. 즉 직접 프로그래밍을 통해 NAND 게이트를 구현하는 경우 위와 같은 절차로(AND 게이트를 3번 지난 다음 NOT 게이트를 통과) NAND 게이트가 작동한다는 것을 알 수 있다.

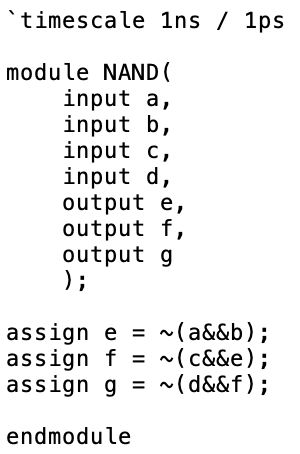
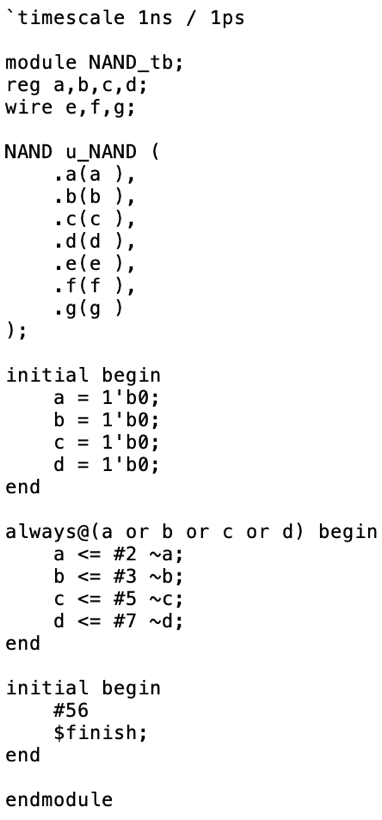
**(B) 4-input, 3-output NAND gate**

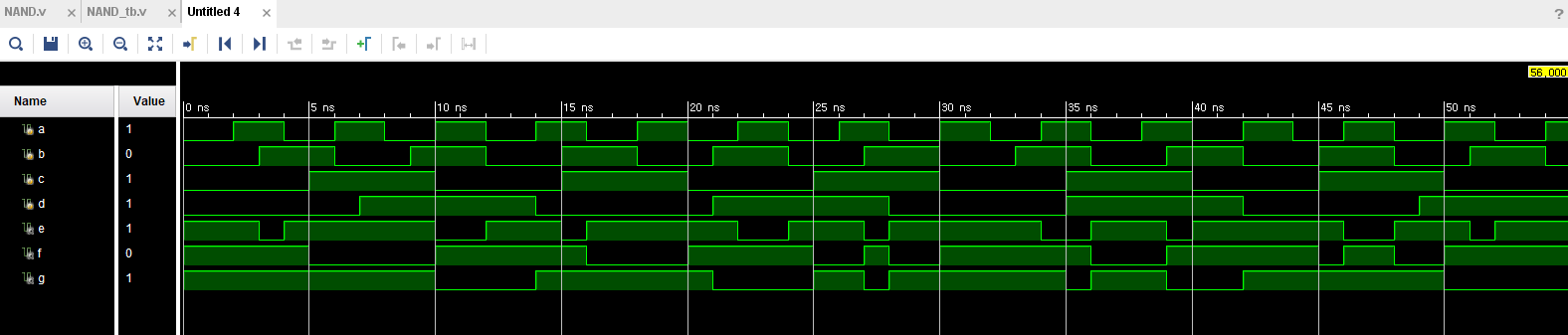
**1) Schematic (강의자료/Vivado)**





**2) Source Code**



**3) Simulation**

**4) Truth Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input A | Input B | Input C | Input D | Output E | Output F | Output G |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 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 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

**5) 구현 과정 설명**

4-input, 3-output NAND 게이트의 경우 Input A,B,C,D 와 output E,F,G가 있을 때 A(nand)B의 값이 Output E에 할당되며, C(nand)E의 값이 Output F에 할당되고, D(nand)F의 값이 Output G에 할당된다. 값을 2개씩 비교하기 때문에, 각 경우 2개의 값이 모두 1일 때에만 0의 output이 반환되며, 나머지의 경우에는 모두 1의 output이 반환된다.

4-input, 3-output NAND 게이트는 지난 주차에서 살펴본 4-input, 3-output AND 게이트에서와는 다르게, 4-input, 1-output의 최종 Output과 4-input, 3-output의 최종 Output 값이 동일하지 않다. 이는 위 Truth Table과 Simulation결과에서 확인할 수 있는데, 4-input, 1-output NAND 게이트에서는 4개의 input이 모두 1인 경우(1111)에만 최종 Output E의 값이 0이 나왔으나, 4-input, 3-output NAND 게이트의 경우에는 4개의 input이 (0001), (0101), (1001), (1101), (1111) 인 경우에도 모두 최종 Output G의 값으로 0을 반환한다.

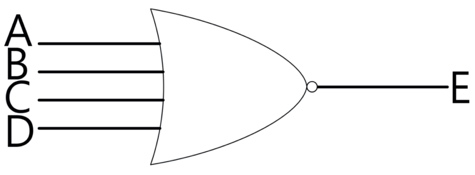
Verilog 코드는 두 input씩 AND로 연결한 다음 바깥에 not을 붙이는 과정을 총 세번 반복하여 구현하였다 : assign **e** = ~(**a**&&**b**); / assign **f** = ~(**c**&&**e**); / assign **g** = ~(**d**&&**f**);

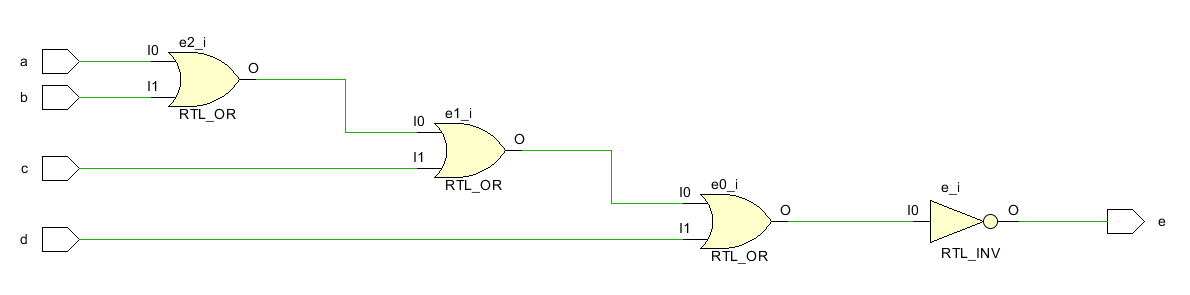
Schematic의 경우 Vivado에서 생성된 Schematic의 모양이 강의자료의 것 보다 조금 더 꼬여 있긴 하지만, 전체적으로 비슷한 모습을 보인다. 또한 강의자료의 Schematic에서는 NAND 게이트의 모양이 AND 게이트에 원(O)모양이 붙어있는 모양의 기호로 그려져 있으나 Vivado에서 생성된 Schematic을 보면 연속된 AND 게이트와 NOT 게이트로 이루어져 있음을 확인할 수 있다.

**3. 4-input NOR gate의 simulation 결과 및 과정에 대해서 설명하시오.**

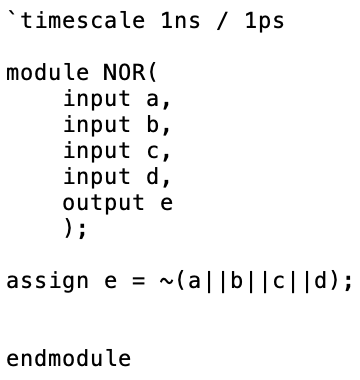
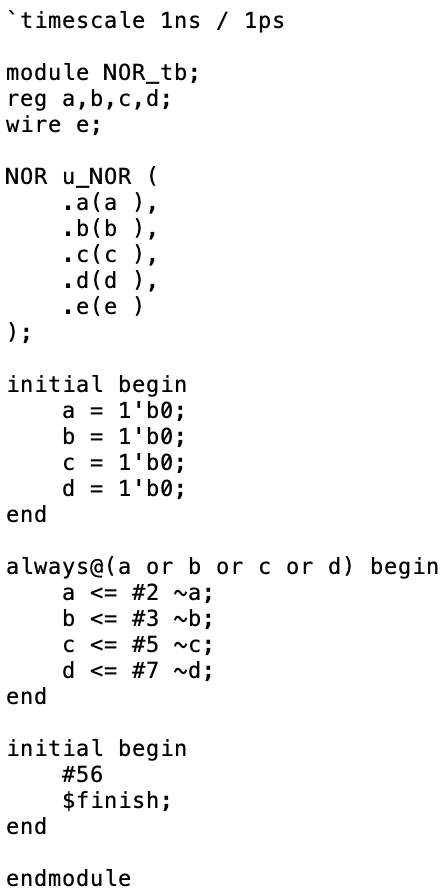
**(A) 4-input, 1-output**

**1) Schematic (강의자료/Vivado)**

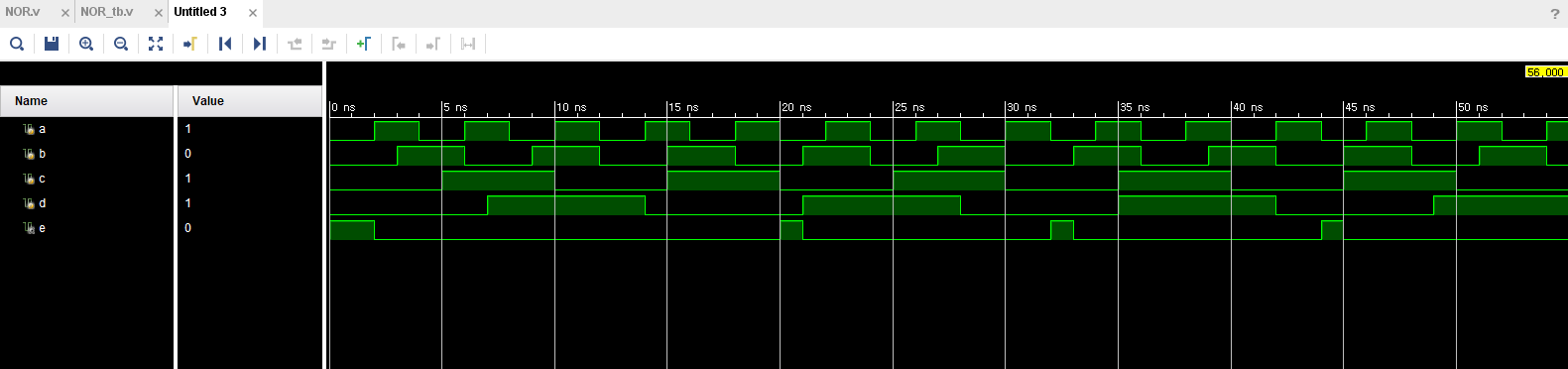




**2) Source Code**



**3) Simulation**



**4) Truth Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input A | Input B | Input C | Input D | Output 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 |

**5) 구현 과정 설명**

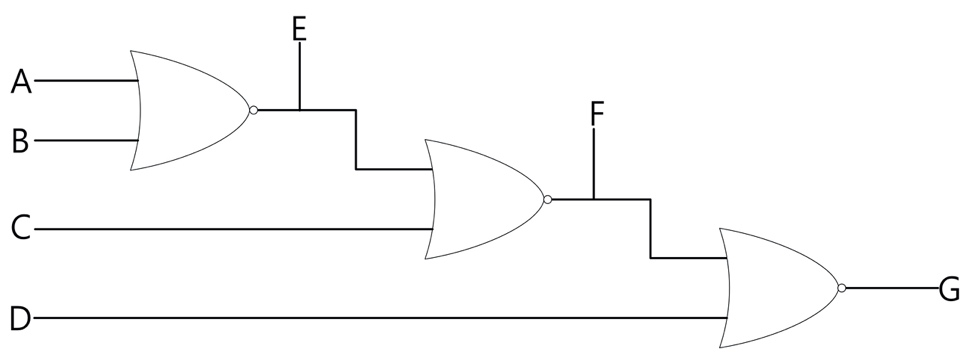
4-input, 1-output NOR 게이트의 경우 Input A,B,C,D 와 output E가 있을 때 A(nor)B(nor)C(nor)D 의 값이 Output E에 할당된다. 이때 4개의 input이 모두 0일 경우 1이 Output으로 반환되고, 그 외의 경우에는 모두 0이 Output으로 반환된다. 이는 4개의 input이 모두 0일때만 0을 반환하고 그 외의 경우에는 1을 반환하는 OR 게이트와 정 반대의 양상을 보이는데, NOR 게이트는 OR 게이트에 NOT 게이트를 붙인 것이기 때문이다. 따라서 OR 게이트와 비교했을 때 NOR 게이트의 truth table은 Output E 부분이 정 반대의 값을 가지는 것을 확인할 수 있다.

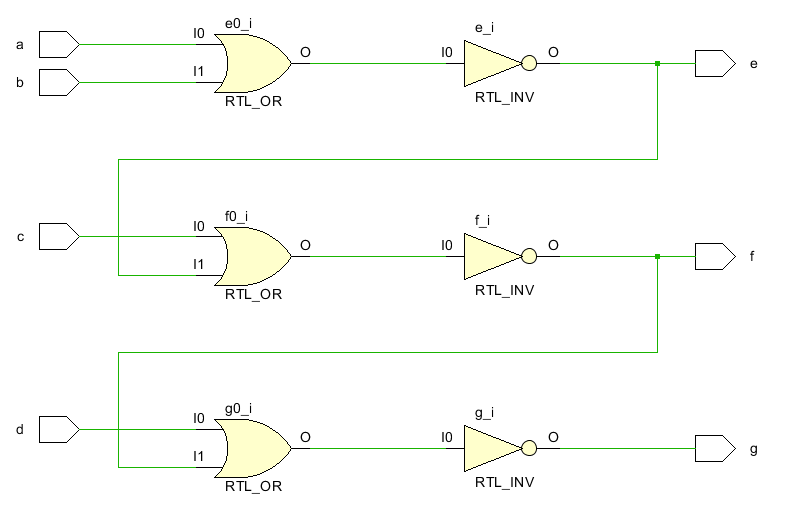
Verilog 코드는 모든 input을 OR로 연결한 다음 바깥에 not을 붙이는 식으로 작성하였다 : assign **e** = ~(**a** || **b** || **c** || **d**);

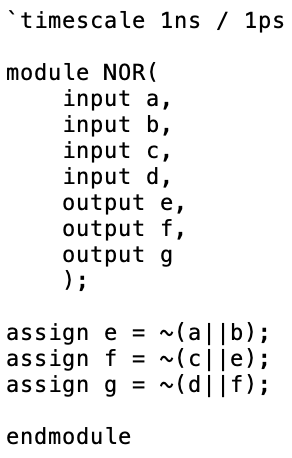
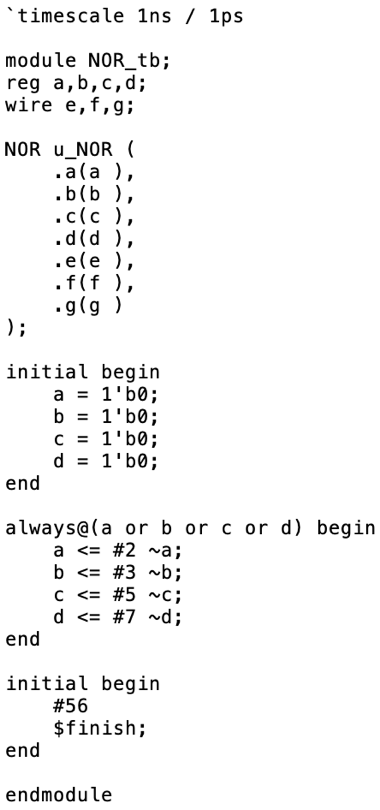
강의자료의 Schematic과 Vivado에서 생성된 Schematic의 모양이 다른데, 강의자료의 Schematic에서는 4개의 input이 하나의 NOR 게이트를 통과하지만 Vivado에서 생성한 Schematic에서는 4개의 input이 차례대로 3개의 OR 게이트를 통과한 다음 마지막으로 NOT 게이트를 통과하는 것을 확인할 수 있다. 즉 직접 프로그래밍을 통해 NOR 게이트를 구현하는 경우 위와 같은 절차로(OR 게이트를 3번 지난 다음 NOT 게이트를 통과) NOR 게이트가 작동한다는 것을 알 수 있다.

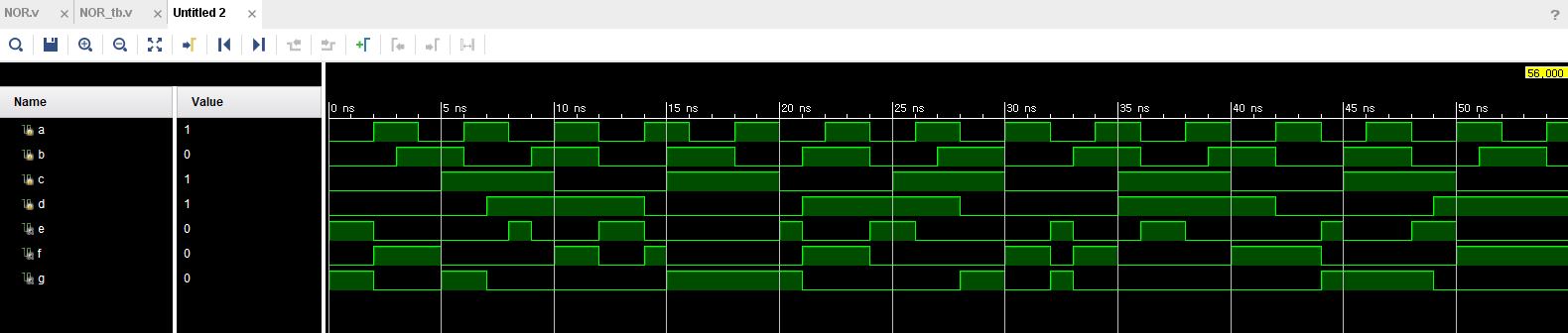
**(B) 4-input, 3-output**

**1) Schematic (강의자료/Vivado)**





**2) Source Code**

**3) Simulation**

**4) Truth Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input A | Input B | Input C | Input D | Output E | Output F | Output G |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

**5) 구현 과정 설명**

4-input, 3-output NOR 게이트의 경우 Input A,B,C,D 와 output E,F,G가 있을 때 A(nor)B의 값이 Output E에 할당되며, C(nor)E의 값이 Output F에 할당되고, D(nor)F의 값이 Output G에 할당된다. 값을 2개씩 비교하기 때문에, 각 경우 2개의 값이 모두 0일 때에만 1의 output이 반환되며, 나머지의 경우에는 모두 0의 output이 반환된다.

4-input, 3-output NOR 게이트는 지난 주차에서 살펴본 4-input, 3-output OR 게이트에서와는 다르게, 4-input, 1-output의 최종 Output과 4-input, 3-output의 최종 Output 값이 동일하지 않다. 이는 위 Truth Table과 Simulation결과에서 확인할 수 있는데, 4-input, 1-output NOR 게이트에서는 4개의 input이 모두 0인 경우(0000)에만 최종 Output E의 값이 1이 나왔으나, 4-input, 3-output NAND 게이트의 경우에는 4개의 input이 (0000), (0010), (0110), (1010), (1110) 인 경우에도 모두 최종 Output G의 값으로 1을 반환한다.

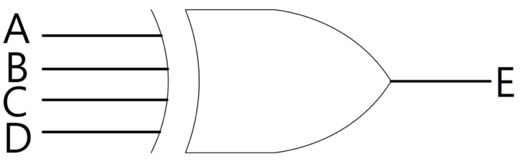
Verilog 코드는 두 input씩 OR로 연결한 다음 바깥에 not을 붙이는 과정을 총 세번 반복하여 구현하였다 : assign **e** = ~(**a** || **b**); / assign **f** = ~(**c** || **e**); / assign **g** = ~(**d** || **f**);

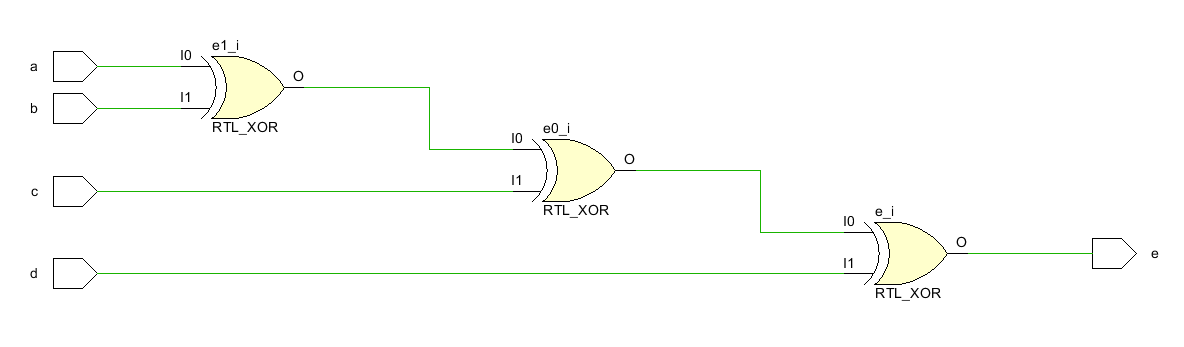
Schematic의 경우 Vivado에서 생성된 Schematic의 모양이 강의자료의 것 보다 조금 더 꼬여 있긴 하지만, 전체적으로 비슷한 모습을 보인다. 또한 강의자료의 Schematic에서는 NOR 게이트의 모양이 OR 게이트에 원(O)모양이 붙어있는 모양의 기호로 그려져 있으나 Vivado에서 생성된 Schematic을 보면 연속된 OR 게이트와 NOT 게이트로 이루어져 있음을 확인할 수 있다.

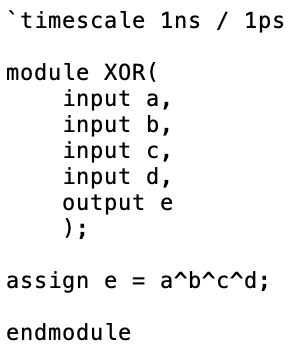
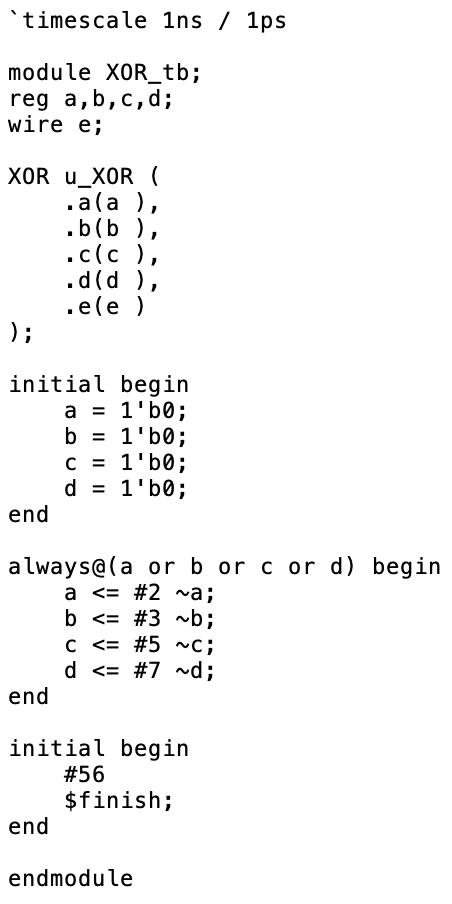
**4. 4-input XOR gate의 simulation 결과 및 과정에 대해서 설명하시오.**

**(A) 4-input, 1-output**

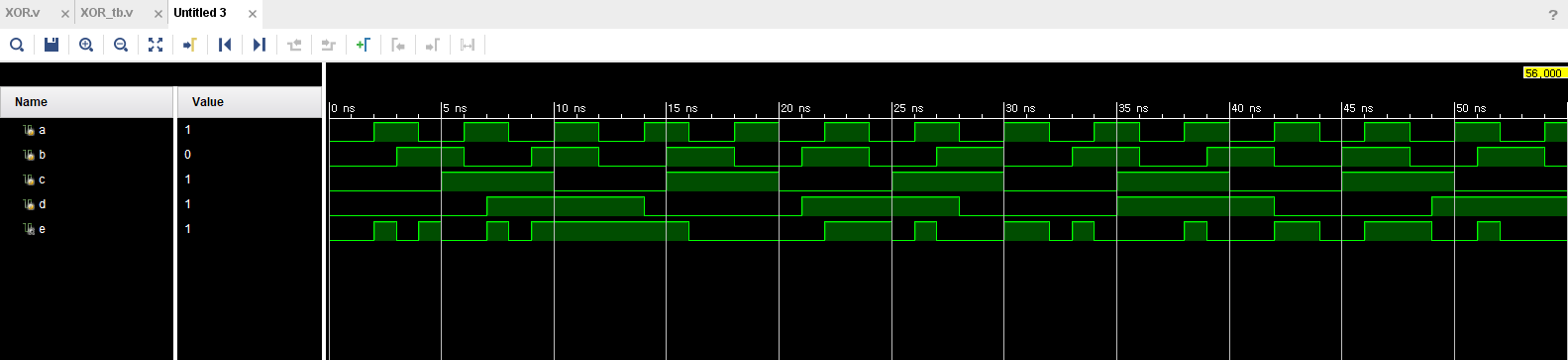
**1) Schematic (강의자료/Vivado)**





**2) Source Code**

**3) Simulation**



**4) Truth Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input A | Input B | Input C | Input D | Output 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 |

**5) 구현 과정 설명**

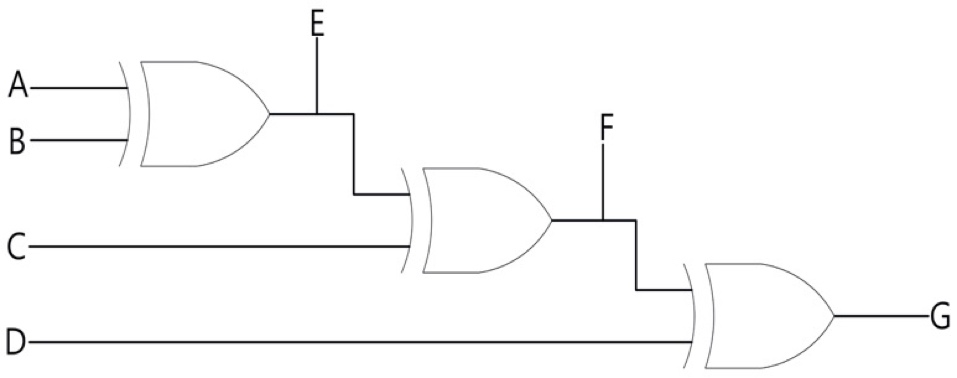
4-input, 1-output XOR 게이트의 경우 Input A,B,C,D 와 output E가 있을 때 A(xor)B(xor)C(xor)D 의 값이 Output E에 할당된다. 이때 4개의 input 중에서 1의 개수가 홀수인 경우 1이 Output으로 반환되고, 1의 개수가 짝수인 경우에는 0이 Output으로 반환된다.

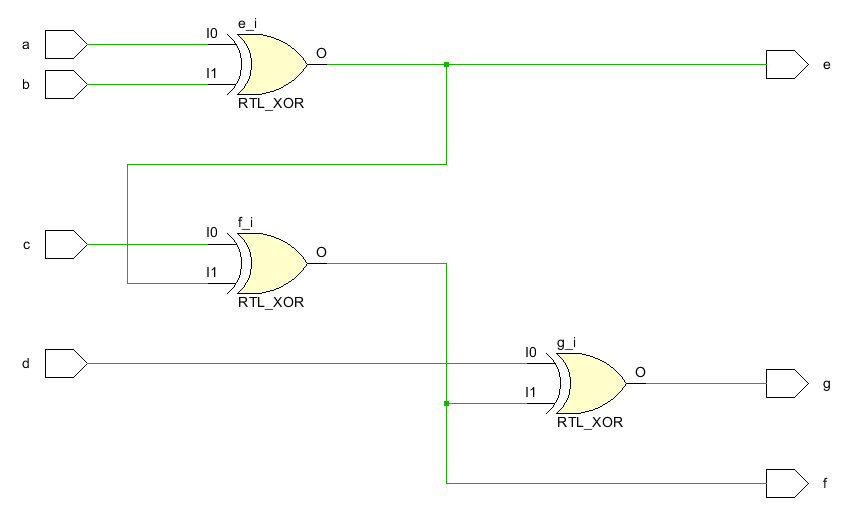
Verilog 코드는 모든 input을 XOR로 연결하는 식으로 작성하였다 : assign **e** = **a^b^c**^**d**;

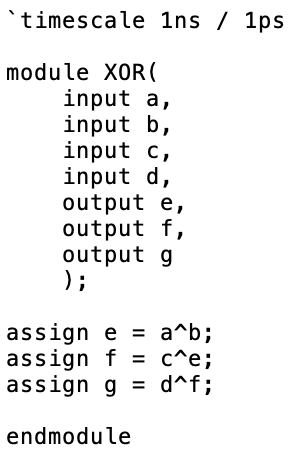
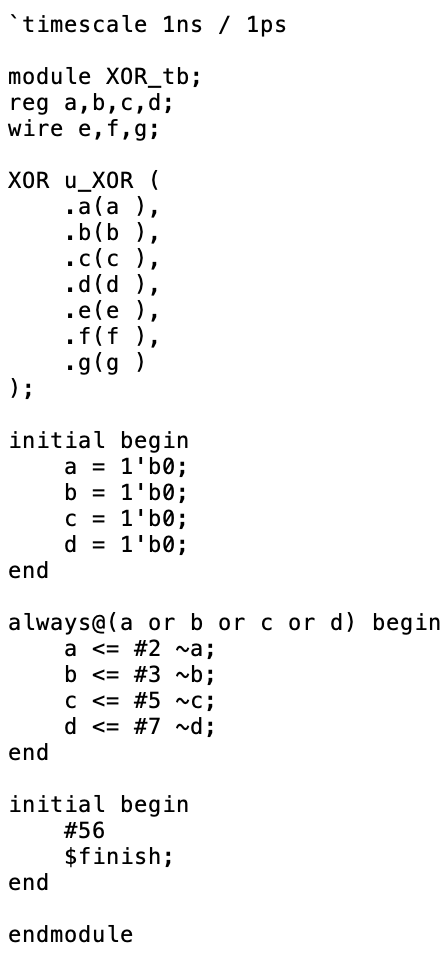
강의자료의 Schematic과 Vivado에서 생성된 Schematic의 모양이 다른데, 강의자료의 Schematic에서는 4개의 input이 하나의 XOR 게이트를 통과하지만 Vivado에서 생성한 Schematic에서는 4개의 input이 차례대로 3개의 XOR 게이트를 통과하는 것을 확인할 수 있다. 즉 직접 프로그래밍을 통해 XOR 게이트를 구현하는 경우 위와 같은 절차로(4개의 input들이 차례대로 XOR 게이트를 3번 지남) XOR 게이트가 작동한다는 것을 알 수 있다.

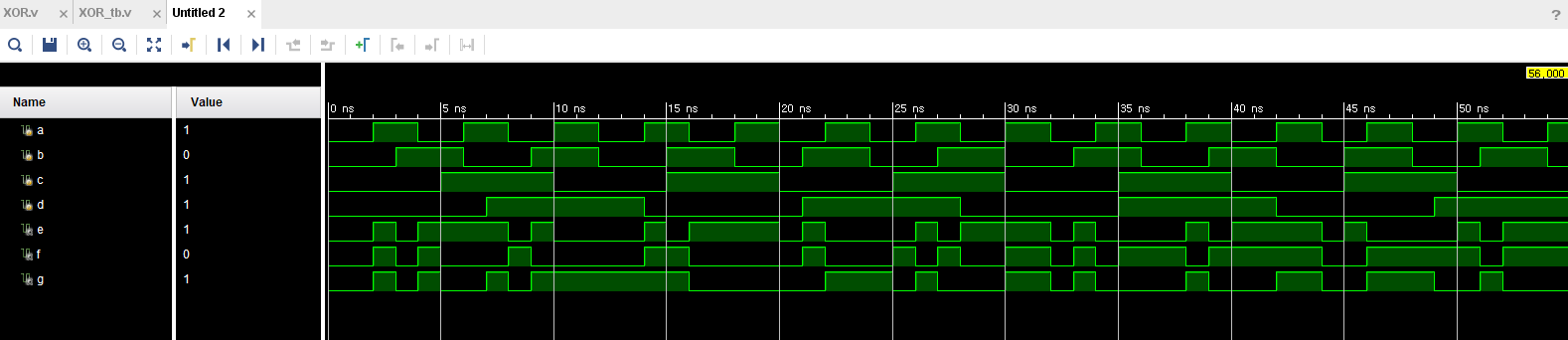
**(B) 4-input, 3-output**

**1) Schematic (강의자료/Vivado)**





**2) Source Code**

**3) Simulation**

**4) Truth Table**

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

**5) 구현 과정 설명**

4-input, 3-output XOR 게이트의 경우 Input A,B,C,D 와 output E,F,G가 있을 때 A(xor)B의 값이 Output E에 할당되며, C(xor)E의 값이 Output F에 할당되고, D(xor)F의 값이 Output G에 할당된다. 값을 2개씩 비교하기 때문에, 각 경우 2개의 값이 같으면(00 또는 11의 경우) 0의 output이 반환되며, 2개의 값이 다르면(10 또는 01의 경우) 1의 output이 반환된다.

4-input, 3-output XOR 게이트는 위에서 살펴본 4-input, 3-output NAND, NOR 게이트와는 달리, 4-input, 1-output의 최종 Output E와 4-input, 3-output의 최종 Output G값이 동일하다. 이는 위 Truth Table과 Simulation결과에서 확인할 수 있다. 두 경우 모두 4개의 input 중에서 1의 개수가 홀수인 경우 1이 최종 Output으로 반환되고, 1의 개수가 짝수인 경우에는 0이 최종 Output으로 반환된다.

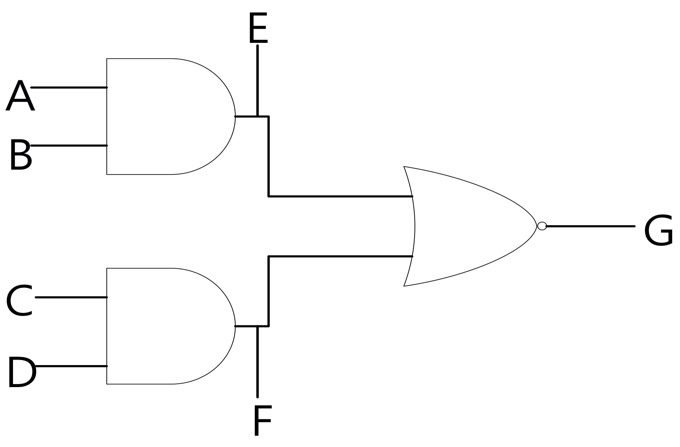
Verilog 코드는 두 input씩 XOR로 연결하는 과정을 총 세번 반복하여 구현하였다.  
: assign **e** = **a**^**b**; / assign **f** = **c**^**e**; / assign **g** = **d**^**f**;

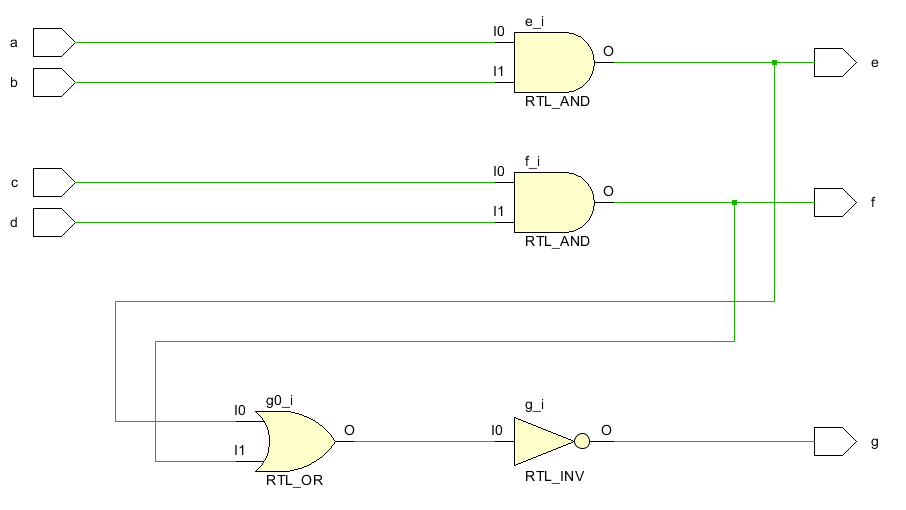
Schematic의 경우 Vivado에서 생성된 Schematic의 모양이 강의자료의 것 보다 조금 더 꼬여 있긴 하지만, 전체적으로 비슷한 모습을 보인다.

**5. 4-input AOI gate의 simulation 결과 및 과정에 대해서 설명하시오.**

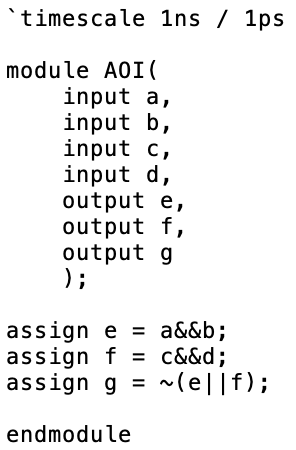
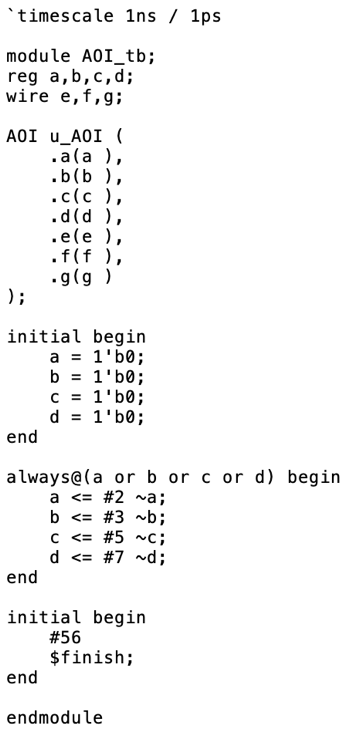
**(A) 4-input, 3-output**

**1) Schematic (강의자료/Vivado)**

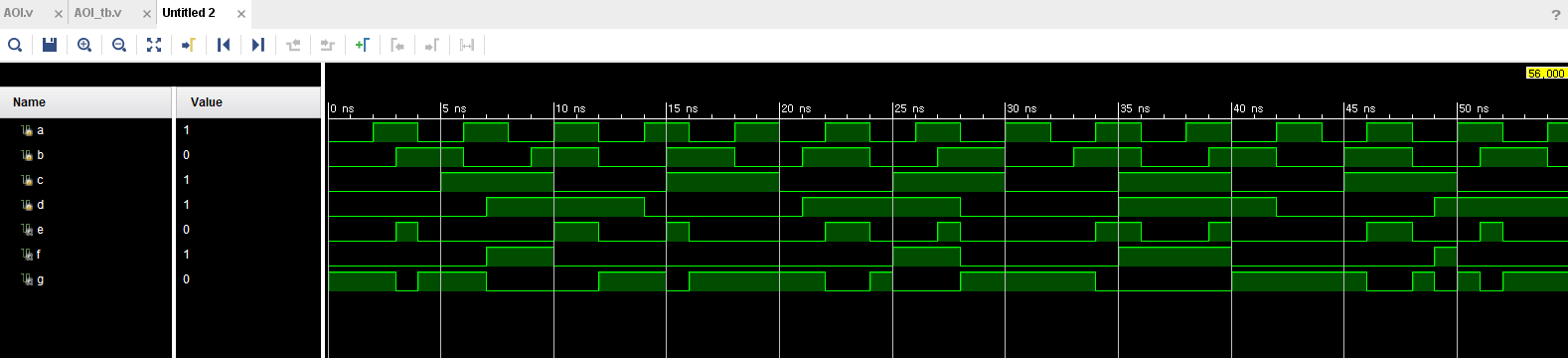




**2) Source Code**



**3) Simulation**



**4) Truth Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input A | Input B | Input C | Input D | Output E | Output F | Output 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 |

**5) 구현 과정 설명**

AOI란 AND-OR-Invert의 줄임말이다. 4-input, 3-output AOI 게이트의 경우 Input A,B,C,D 와 output E,F,G가 있을 때 A(and)B의 값이 Output E에 할당되며, C(and)D의 값이 Output F에 할당되고, E(nor)F의 값이 Output G에 할당된다.

Verilog 코드는 두 input씩 and로 연결하는 과정을 두 번, 그리고 두 input을 or로 연결한 다음 바깥쪽에 not을 붙여주는 방식으로 nor를 구현하였다.  
: assign **e** = **a**&&**b**; / assign **f** = **c**&&**d**; / assign **g** = ~(**e** || **f)**;

Schematic의 경우 Vivado에서 생성된 Schematic의 모양이 강의자료의 것 보다 조금 더 꼬여 있긴 하지만, 전체적으로 비슷한 모습을 보인다. 또한 강의자료의 Schematic에서는 마지막 NOR 게이트의 모양이 OR 게이트에 원(O)모양이 붙어있는 모양의 기호로 그려져 있으나 Vivado에서 생성된 Schematic을 보면 연속된 OR 게이트와 NOT 게이트로 이루어져 있음을 확인할 수 있다.

**6. 결과 검토 및 논의사항**

NAND 게이트와 NOR 게이트는 AND, OR, XOR 게이트와 달리 4개의 input에 동시에 논리식을 적용하였을 때와 2개의 input씩 나누어서 논리식을 적용하였을 때의 최종 Output 값이 다르게 출력되었다. 이 점이 예상과는 다른 부분이었기에, 실제 회로 설계에 있어서 이 점을 주의할 필요가 있겠다는 생각이 들었다.

**7. 추가 이론 조사 및 작성**

**1) XOR 게이트의 논리식 표현**

4주차 실습에서 XOR 게이트의 경우 코드 작성에 있어서 논리식을 X^Y 와 같이 작성했지만, 실제 XOR의 논리식은 즉 와 같이 표현되기 때문에 코드 작성에 있어서 X^Y 대신 (~X&&Y) || (X&&~Y) 식을 사용하여도 Simulation 결과 및 진리표에서 동일한 결과값을 확인할 수 있다. 대신 Schematic이 (^)기호를 사용할 때 보다 훨씬 복잡하게 그려진다는 단점이 있다.