컴퓨터공학실험2 4주차 결과 보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20201635 이름: 전찬

**0. 목차**

1. 실험 목적

2. 4-input NAND gate의 simulation 결과 및 과정 설명

3. 4-input NOR gate의 simulation 결과 및 과정 설명

4. 4-input XOR gate의 simulation 결과 및 과정 설명

5. 4-input AOI gate의 simulation 결과 및 과정 설명

6. 결과 검토 및 논의 사항

7. 추가 이론 조사 및 작성

**1. 실험 목적**

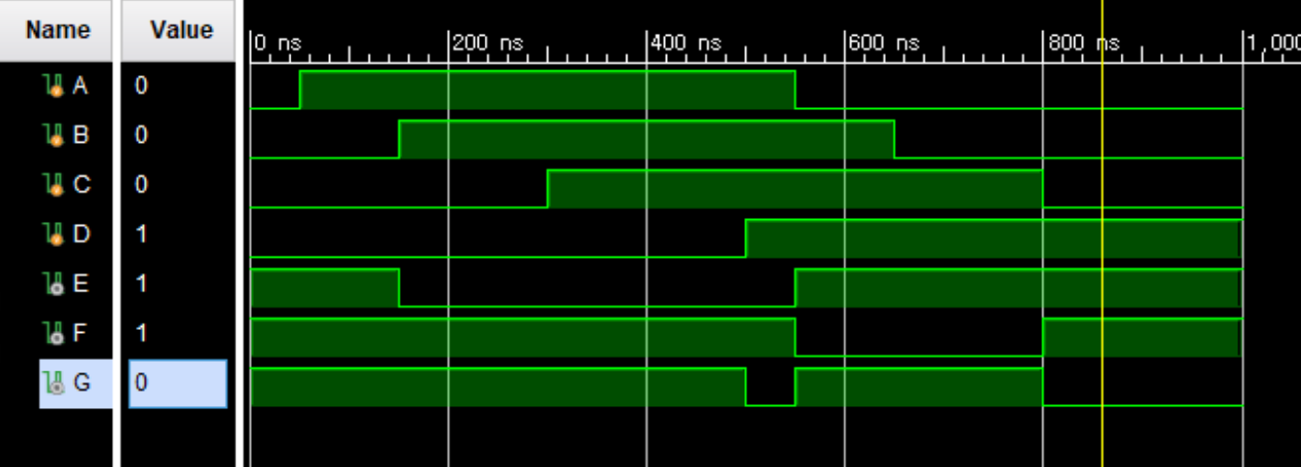
NAND / NOR / XOR / AOI gate가 각각 어떤 동작을 수행하는지 이해한다. 또한 2-input 뿐만 아닌, input을 3, 4개로 확장시킬 때도 각각의 gate가 어떤 역할을 수행하는지 실습을 통해서 확인한다. NAND / NOR / XOR gate에서 서로 다른 두 형태의 4-input gate들에 대해서 output에 차이가 존재하는지 비교한다.

**2. 4-input NAND gate의 simulation 결과 및 과정**

4-input NAND gate는 3개의 2-input NAND gate를 이용해서 구현해낼 수 있다. input A, B, C, D 에 대해서 A, B를 NAND gate의 input으로 가진 이후 나온 output에 대해 다시 C와 NAND gate 의 input으로 가지며, 이 형태를 반복해서 4-input NAND gate를 만들어낼 수 있다.

+추가적으로 NAND 3개를 통해서 만들어낸 4-input NAND gate와 4-input AND gate 의 output에 NOT gate(bubble)을 추가한 gate는 동일하지 않다. 앞의 경우에는 input에서 특별한 규칙성이 없이 output이 1 또는 0이 되지만(논리식은 (A’+B’)C + D’ 형태), 뒤의 경우는 모든 input이 1인 경우에만 0이 된다. 이는 지금까지 3-input AND, 3-input OR gate등과 같이 결합한 형태가 2-input으로 만들어낸 형태와 동일한 역할을 수행하는 gate들과는 다르다는 것을 파악할 수 있다.

이제부터 4-input NAND gate((B) 형태)에 대해 알아보자. 4-input NAND gate의 simulation 결과는 다음과 같다.



<4-input NAND gate simulation>

또한 4-input NAND gate의 진리표는 다음과 같다.

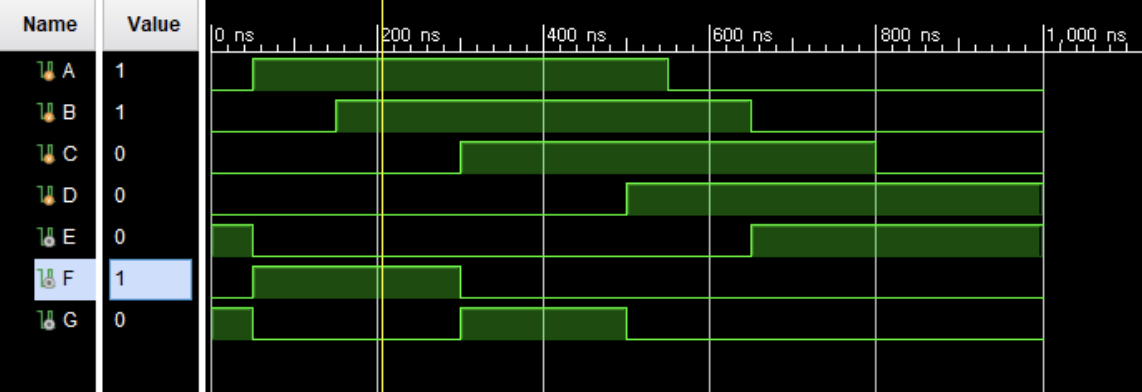
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4-input NAND gate | | | | | | |
| input | | | | output | | |
| A | B | C | D | E | F | G |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |

<4-input NAND gate 진리표>

**3. 4-input NOR gate의 simulation 결과 및 과정**

4-input NOR gate 또한 4-input NAND gate를 만드는 것처럼 두 가지 방법으로 만들어낼 수 있다. 첫 번째 방법은 4-input OR gate의 output에 NOT gate(bubble)를 합치는 것이고, 두 번째는 이번 실습에서 진행했던 2-input NOR gate 3개를 이용해서 만들어내는 것이다. 이 경우에서도 두 회로는 다른 역할을 수행한다. 첫 번째 회로는 모든 input이 0일 때만 output으로 1을 가지는 회로이다. 이와 다르게 두 번째는 일정한 규칙성 없이 input에 따른 output을 가지게 된다.(논리식은 (A’B’+C)D’ 형태이다.)

4-input NOR gate의 simulation 결과는 다음과 같다.



<4-input NOR gate simulation>

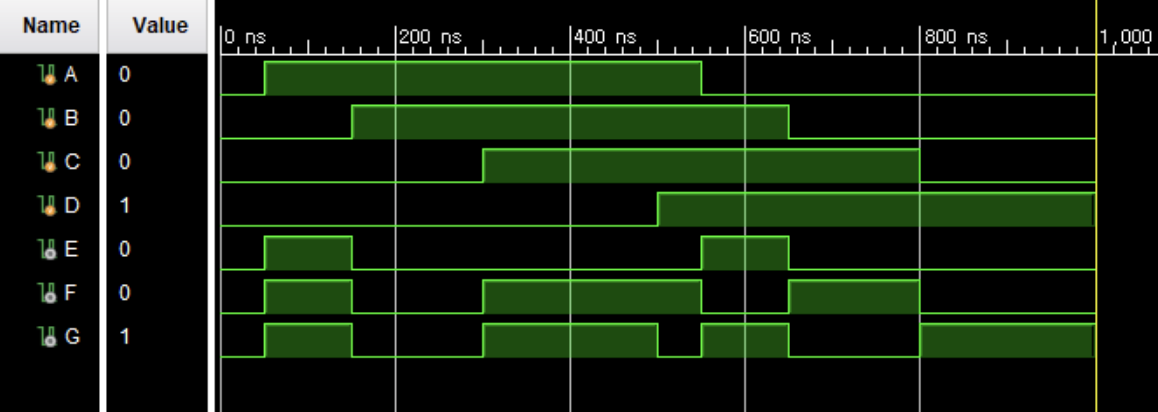
또한 4-input NOR gate의 진리표는 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4-input NOR gate | | | | | | |
| input | | | | output | | |
| A | B | C | D | E | F | G |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |

<4-input NOR gate 진리표>

**4. 4-input XOR gate의 simulation 결과 및 과정**

사실 대부분의 경우에서 XOR gate는 input이 2개인 gate로 여긴다. 하지만 특수하게 4-input XOR gate를 만들어낼 수 있다. 4-input XOR gate또한 위에서 정의했던 4-input NAND/NOR gate와 비슷하게 두 형태로 생각할 수 있다. 첫 번째로는 4-input XOR gate 또한 논리적으로 exclusive OR 이라고 여기는 것인데, 4개의 input 중 하나의 input만 High(1) 이어야 output으로 High(1)을 갖는 gate라고 생각할 수 있다. 다른 방법으로는 2-input XOR gate 3개를 연결한 형태인데, input A, B, C, D에 대해서 논리식 A’B’C’D + A’B’CD’ + A’BC’D’ + A’BCD + AB’C’D’ + AB’C’D + ABC’D + ABCD’ 형태이다. 즉, 4-input XOR gate는 input의 개수와 상관없이, input의 High(1)의 개수가 홀수라면 output으로 1을 갖는 회로이다. 이번 실습에서는 이 회로를 제작하기도 했다. 이에 대한 simulation 결과는 아래와 같다.



<4-input XOR gate simulation>

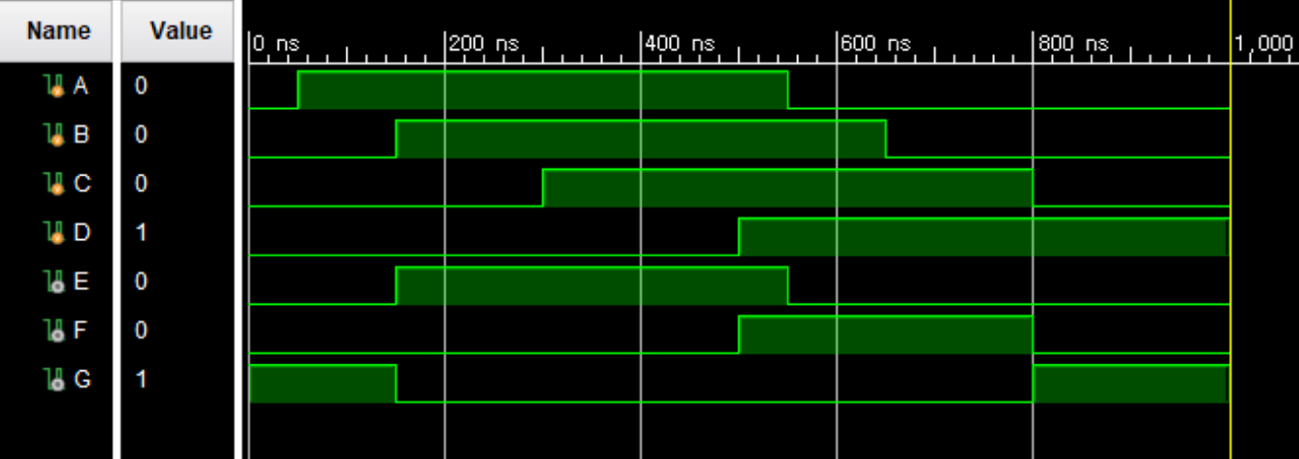
4-input XOR gate의 진리표는 아래와 같은 형태이다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4-input XOR gate | | | | | | |
| input | | | | output | | |
| A | B | C | D | E | F | G |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

<4-input XOR gate 진리표>

**5. 4-input AOI gate의 simulation 결과 및 과정**

4-input AOI gate는 input A, B, C, D를 (AB + CD)’ 형태로 만드는 gate이다. 드모르간 법칙을 이용해서 (A’ + B’)(C’ + D’) 형태가 됨을 파악할 수 있으며, A, B 에서 Low(0)이 1개 이상, C, D에서 Low(0)이 1개 이상이면 output으로 High(1)을 가짐을 파악할 수 있다. 추가로 이번 실습에서 활용한 형태는 2-2 AOI gate라고 하기도 한다. 4-input AOI gate의 simulation 결과는 아래와 같다.



<4-input AOI gate simulation>

또한 4-input AOI gate의 진리표는 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4-input AOI gate | | | | | | |
| input | | | | output | | |
| A | B | C | D | E | F | G |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |

<4-input AOI gate 진리표>

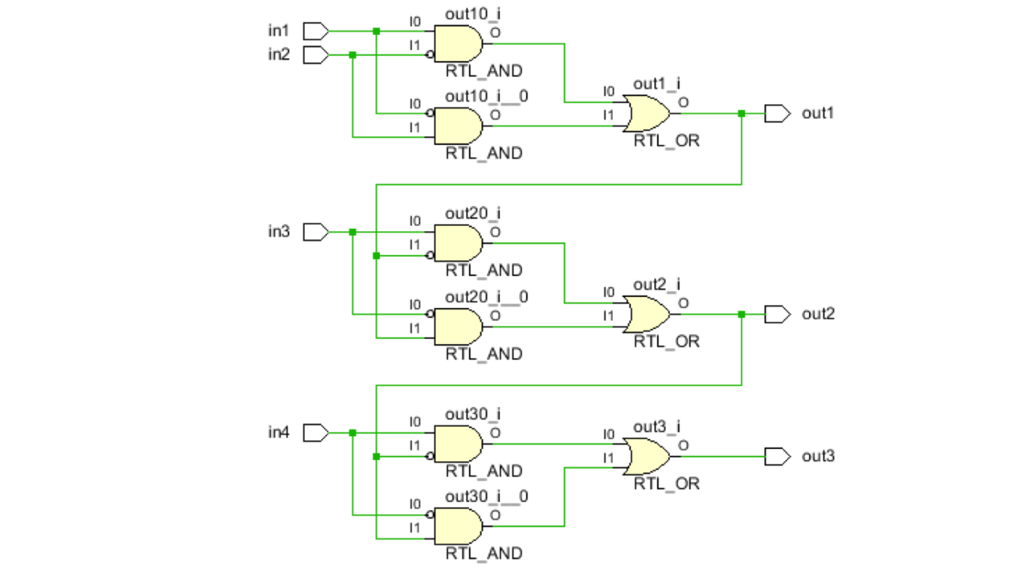
**6. 결과 검토 및 논의 사항**

- verilog에서 XOR gate의 schematic 확인

이번 실습에서는 이전 실습(4-input AND / OR gate의 구현)과는 다르게, 같은 4-input gate라도 여러 형태로 정의될 수 있음을 파악할 수 있다. 또한 4-input NAND / NOR / XOR gate같은 경우에는 2-input에서 수행하는 역할과 전혀 다른 역할을 수행하는 gate가 될 수도 있음을 파악할 수도 있었다. 추가로 개인적으로 XOR gate를 “^” 한 문자로 표현할 수 있다는 것을 알지 못해서, AB’ + A’B 형태로 표현하기도 했는데, 이번에 조교님께서 알려주셔서 더 편하게 표현할 수 있었다.

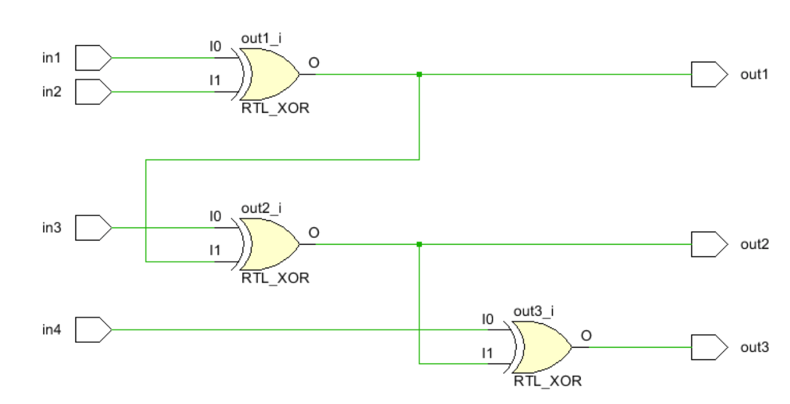
위 기호를 알려주시며, schematic을 통해서 AB’ + A’B(실제로 verilog에서는 A&(~B)|(~A)&B) 형태의 식에서 XOR gate가 작용하는지 XOR gate를 schematic으로 구현해봤을 때, 동일한 결과가 나오는지 나에게 질문을 하셨었는데, 컴퓨터에서 schematic을 바로 실행하려면 튕기는 현상이 존재해서 그때 확인을 하지는 못했다. 이를 확인해보자.

우선 첫 번째 형태(X&(~Y)|(~X)&Y) 형태로 정의했을 때는 schematic이 다음과 같다.



<X&(~Y)|(~X)&Y 형태로 정의했을 때의 schematic>

두 번째 형태(X^Y) 는 다음과 같다.



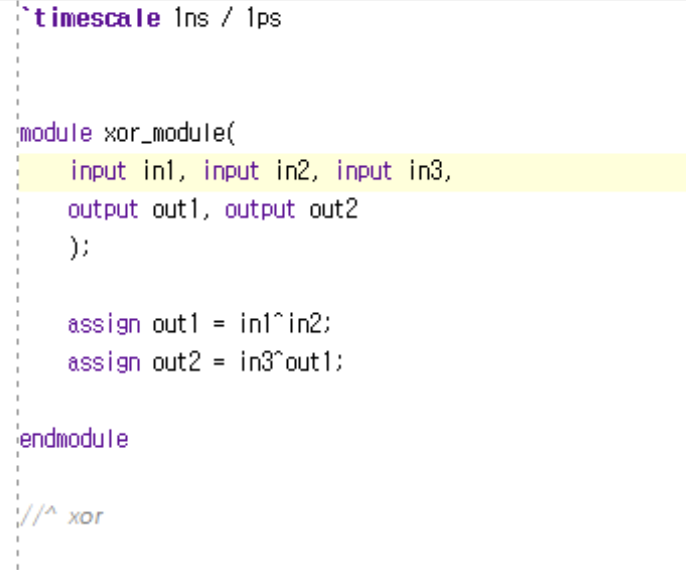
<X^Y 형태로 정의했을 때 schematic>

결과를 보면 쉽게, 첫 번째 형태로 정의했을 때, 두 번째 형태와 동일하지 않은 schematic이 만들어짐을 파악할 수 있다. 또한 첫 번째 형태에서 회로가 많은 만큼, 실행 속도도 느려짐을 예측할 수도 있다. 따라서 앞으로는 XOR gate를 “^” 기호를 사용하는 것이 코드 면에서도, 실행 속도 면에서도 효율적이다.

- 3-input XOR gate의 확인

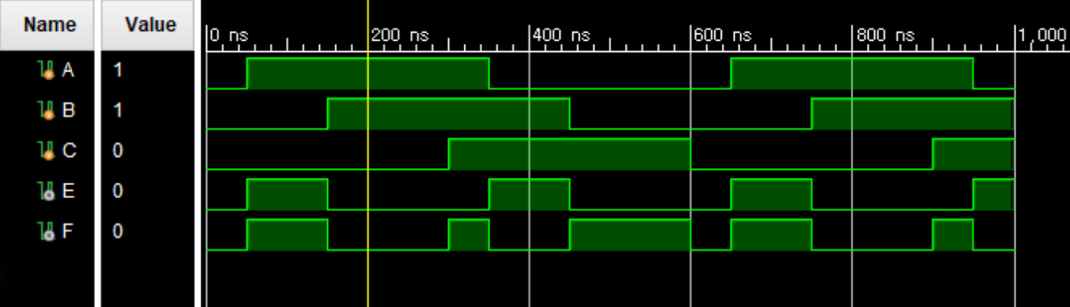
또한 지금까지 2-input, 4-input XOR gate에서는 “input의 High(1) 이 홀수일 때 output이 High(1)이다.”라는 정의가 적용했는데, input이 홀수일 때도 동일한 정의가 적용되는지 또한 필요한 과정이다. 따라서 3-input XOR gate(2-input XOR gate 2개를 연결한 gate)를 verilog를 통해서 만들어서 동일한 역할을 수행하는지 확인해보자.

우선 xor\_module.v 코드는 아래와 같이 작성했다.



<3-input XOR gate의 verilog코드>

이에 대한 simulation 결과는 아래와 같다.



<3-input XOR gate의 simulation 결과>

simulation 결과를 토대로, input의 High(1)의 개수가 홀수일 때, output(F)가 High(1)이 됨을 파악할 수 있으며, input이 홀수일 때도 동일한 정의가 작용함을 파악할 수 있다.

**7. 추가 이론 조사 및 작성**

- XOR gate의 정확한 정의

위에서 XOR gate가 두 가지 형태로 정의될 수 있다고 설명을 했었는데, 정확하게 파악하기 위해서 여러 전공 책과 문서들을 찾아봤는데, 이에 대한 정보가 거의 나와있지 않았다.(주로 XOR gate는 input이 2개인 경우만 존재한다고 결론지었다.) 결과적으로 IEEE-Std91a-1991 에서 XOR gate가 “The output stands at its 1-state if one and only one if the two inputs stands at its 1-state.”1 와 같이 정의되어 있음을 파악할 수 있었다. 하지만 1991년도의 문서라 30년동안 정의가 바뀌었을 수도 있기 때문에, XOR gate의 정확한 정의에 대해서는 2-input일 때만 정의된다고 여기기로 했다.