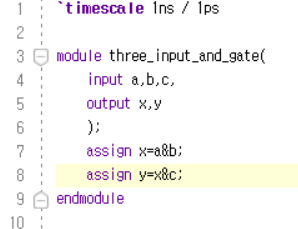
3주차 결과보고서

전공: 심리학과 학년: 3학년 학번: 20190345 이름: 김동현

**1.**

FPGA 동작 방법은 총 5단계로 나뉘어 verilog coding -> run synthesis -> device/pin assignment -> synthesis/implement -> device configuration 순서로 진행된다. 첫번째, Verilog coding 단계는 베릴로그를 이용해 코드를 작성하는 단계이다. 입력과 출력, test-bench을 구현하며, simulation을 통해 결과를 예측해볼 수 있다. 두번째, run synthesis 단계는 합성을 진행하는 단계이다. 합성이란 인간이 이해하기 쉬운 고수준의 언어인 베릴로그로 구현된 design 소스 파일을 fpga 보드가 이해할 수 있는 저수준의 언어로 변환하는 과정이다. C에서의 컴파일 과정(c언어를 어셈블리어, 기계어로 바꾸는 과정)과 유사하다. 세번째, device/pin assignment과정은 저수준으로 합성된 베릴로그 코드를 fpga 보드에 할당하는 과정이다. 프로젝트 생성 시 입력하는 “xc7a75tfgg484”가 할당하는 device를 나타내는 번호이다. 뿐만 아니라 I/O port에 원하는 pin list를 할당한다. 네번째, synthesis/implement는 합성을 진행한 뒤, implementation과정을 진행한다. 마지막으로 device configuration단계에서는 비트 스트림을 생성하여 fpga로 정보를 넘긴다. Hardware manager를 통해 target을 오픈하고, auto connect를 실시하여 구현한다. Project name\_runs/impl\_l 경로에 debug\_nets.ltx 파일을 넣은 뒤, “xc7a75t\_0”으로 program device를 설정한다. 이후 fpga의 동작을 검증한다.

**2.**

****

<그림1. 3-input and gate design code>

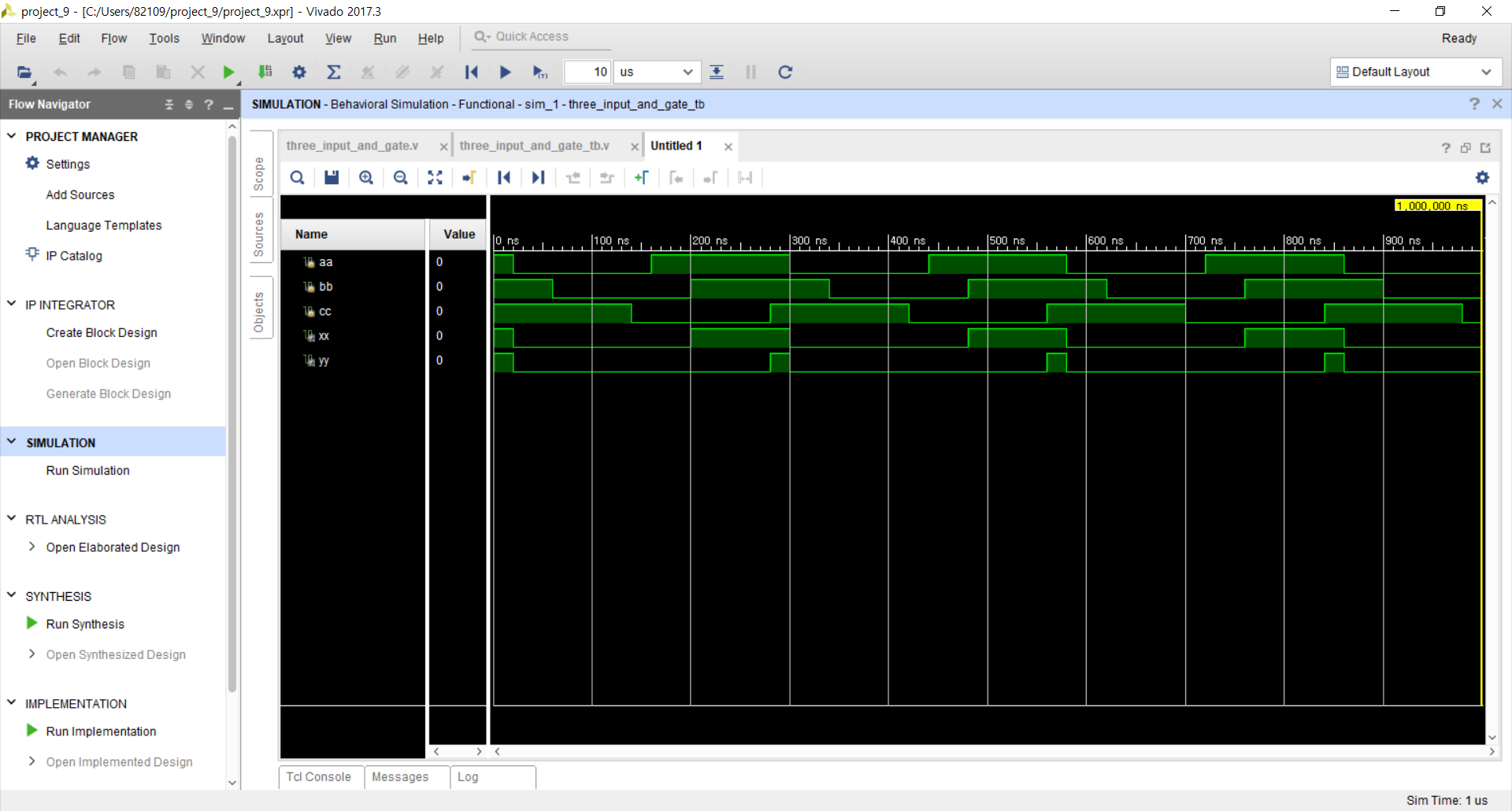
위 그림1은 3개의 입력을 가진 AND Gate의 디자인 소스 파일이다. 입력 변수 a,b,c, 출력 변수 x,y가 선언되었다. Assign문을 통해 x에 a와 b의 and연산 결과가 y에 x(a&b)와 c의 and 연산 결과가 저장된다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

<그림2. 3-input and gate simulation code>

위 그림2는 3개의 입력을 가진 AND Gate의 시뮬레이션 소스 파일이다. Reg변수 aa, bb, cc, wire변수 xx, yy가 선언되었고, 디자인 소스파일의 변수와 각각 매칭하였다. Initial문을 통해 입력 변수와 매칭된 aa, bb, cc에 1의 값을 초기값으로 부여하였다. 이후 aa는 20ns마다, bb는 40ns마다, cc는 80ns마다 값을 변화하도록 하였다. 이러한 반복을 1000ns만큼 실행한 뒤, 프로그램을 종료한다.

****

<그림 3. 3-input and gate simulation>

디자인 소스파일과 시뮬레이션 소스 파일을 바탕으로 run simulation을 한 결과 그림 3과 같은 시뮬레이션 결과가 나왔다. Aa와 bb 모두 1의 값을 가지는 경우에는 xx또한 1을 출력하며, aa, bb, cc 모두 1의 값을 가지는 경우에는 yy까지 1의 값을 출력한다. 그렇지 않은 경우에는 0의 값을 출력한다. 이로서, and gate는 모든 경우가 1인 경우에 1을, 하나라도 1이 아닌 경우 0을 출력함을 알 수 있다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input a | Input b | Input c | Output x | Output y |
| 0 | 0 | 0 | 0 | 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 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

<table 1. 3-input and gate truth table>

위 표는 3개의 입력을 가진 AND Gate의 진리표를 나타낸다.

3.

테이블이(가) 표시된 사진

자동 생성된 설명

<그림4. 4-input and gate design code>

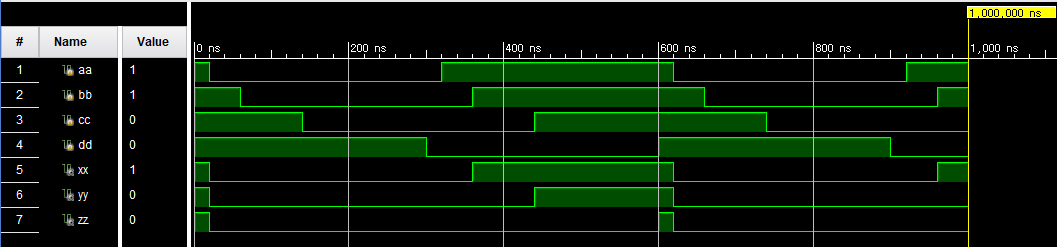
위 그림4은 4개의 입력을 가진 AND Gate의 디자인 소스 파일이다. 입력 변수 a,b,c,d 출력 변수 x,y,z가 선언되었다. Assign문을 통해 x에 a와 b의 and연산 결과가, y에 x(a&b)와 c의 and 연산 결과가, z에 y(a&b&c)와 d의 and 연산 결과가 저장된다.

테이블이(가) 표시된 사진

자동 생성된 설명

<그림5. 4-input and gate simulation code>

위 그림5는 4개의 입력을 가진 AND Gate의 시뮬레이션 소스 파일이다. Reg변수 aa, bb, cc, dd wire변수 xx, yy, zz가 선언되었고, 디자인 소스파일의 변수와 각각 매칭하였다. Initial문을 통해 입력 변수와 매칭된 aa, bb, cc, dd에 1의 값을 초기값으로 부여하였다. 이후 aa는 20ns마다, bb는 40ns마다, cc는 80ns마다, dd는 160ns 마다 값을 변화하도록 하였다. 이러한 반복을 1000ns만큼 실행한 뒤, 프로그램을 종료한다.



<그림6. 4-input and gate simulation>

디자인 소스파일과 시뮬레이션 소스 파일을 바탕으로 run simulation을 한 결과 그림 6과 같은 시뮬레이션 결과가 나왔다. Aa와 bb 모두 1의 값을 가지는 경우에는 xx또한 1을 출력하며, aa, bb, cc 모두 1의 값을 가지는 경우에는 yy까지 1의 값을 출력하며, aa, bb, cc, dd 모두 1의 값을 가지는 경우에는 zz까지 1의 값을 출력한다. 그렇지 않은 경우에는 0의 값을 출력한다. 이로서, and gate는 모든 경우가 1인 경우에 1을, 하나라도 1이 아닌 경우 0을 출력함을 알 수 있다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input a | Input b | Input c | Input d | Output x | Output y | Output z |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 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 | 0 |
| 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 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

<table 2. 4-input and gate truth table>

위 표는 4개의 입력을 가진 AND Gate의 진리표를 나타낸다.

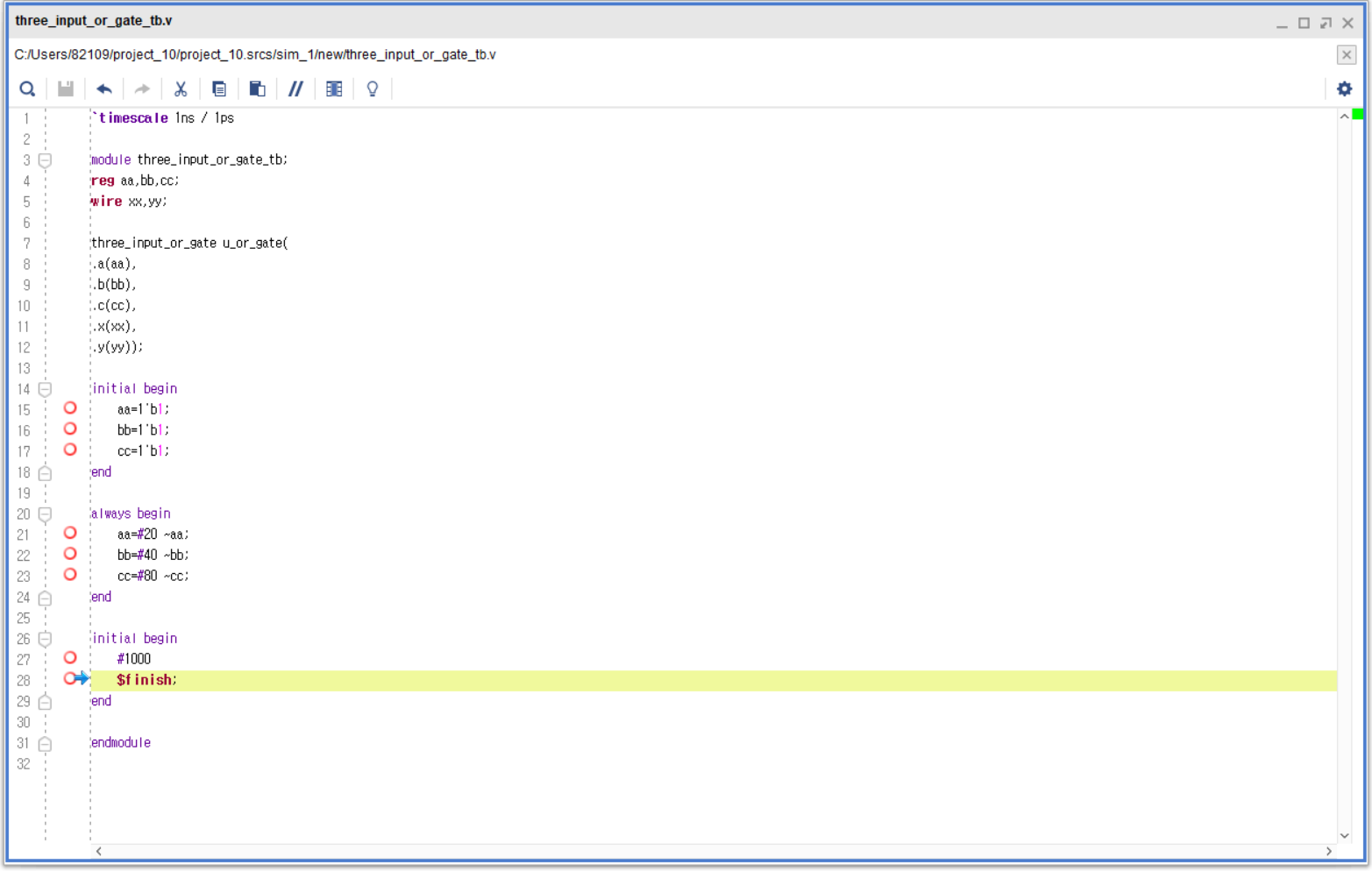
4.

텍스트이(가) 표시된 사진

자동 생성된 설명

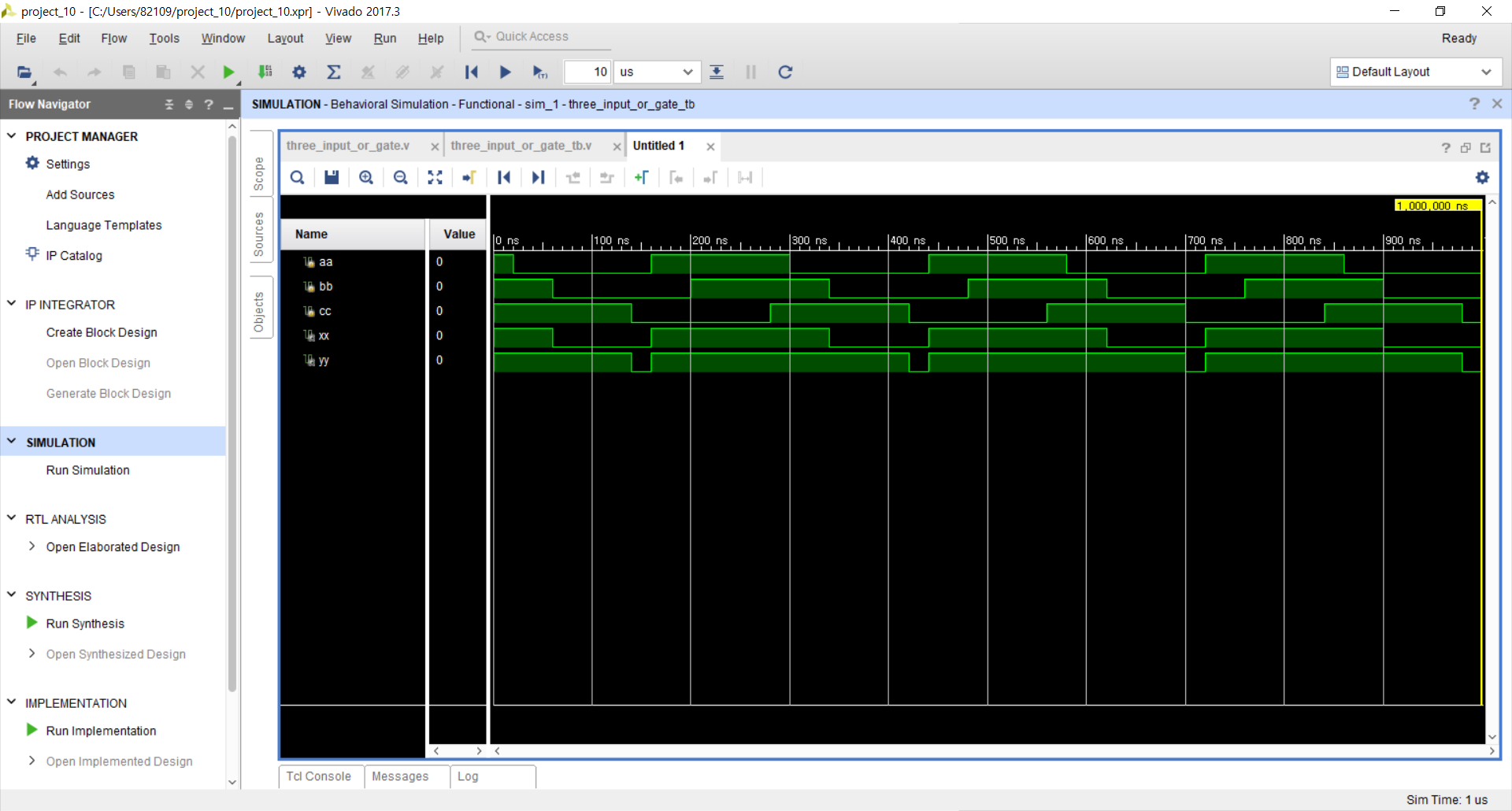
<그림7. 3-input or gate design code>

위 그림7은 3개의 입력을 가진 OR Gate의 디자인 소스 파일이다. 입력 변수 a,b,c, 출력 변수 x,y가 선언되었다. Assign문을 통해 x에 a와 b의 or연산 결과가 y에 x(a|b)와 c의 or 연산 결과가 저장된다.



<그림8. 3-input or gate simulation code>

위 그림8는 3개의 입력을 가진 OR Gate의 시뮬레이션 소스 파일이다. Reg변수 aa, bb, cc, wire변수 xx, yy가 선언되었고, 디자인 소스파일의 변수와 각각 매칭하였다. Initial문을 통해 입력 변수와 매칭된 aa, bb, cc에 1의 값을 초기값으로 부여하였다. 이후 aa는 20ns마다, bb는 40ns마다, cc는 80ns마다 값을 변화하도록 하였다. 이러한 반복을 1000ns만큼 실행한 뒤, 프로그램을 종료한다.



<그림9. 3-input or gate simulation>

디자인 소스파일과 시뮬레이션 소스 파일을 바탕으로 run simulation을 한 결과 그림 9과 같은 시뮬레이션 결과가 나왔다. Aa와 bb 중 하나라도 1의 값을 가지는 경우에는 xx또한 1을 출력하며, aa, bb, cc 중 하나라도 1의 값을 가지는 경우에는 yy까지 1의 값을 출력한다. 모든 수가 0의 값을 가지는 경우에는 0의 값을 출력한다. 이로서, or gate는 모든 경우가 0인 경우에 0을, 하나라도 0이 아닌 1인 경우 경우 1을 출력함을 알 수 있다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input a | Input b | Input c | Output x | Output y |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 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 | 1 | 1 |

<table 3. 3-input or gate truth table>

위 표는 3개의 입력을 가진 OR Gate의 진리표를 나타낸다.

5.

테이블이(가) 표시된 사진

자동 생성된 설명

<그림10. 4-input or gate design code>

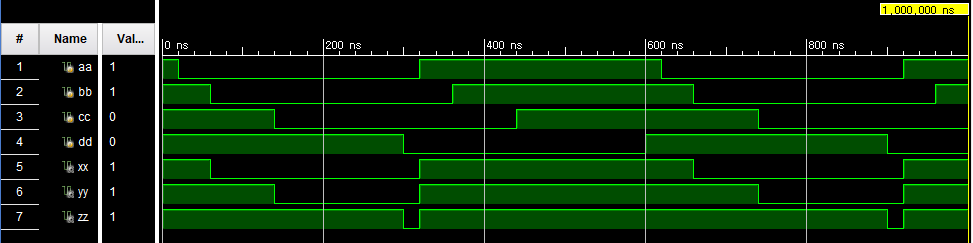
위 그림10은 4개의 입력을 가진 OR Gate의 디자인 소스 파일이다. 입력 변수 a,b,c,d 출력 변수 x,y,z가 선언되었다. Assign문을 통해 x에 a와 b의 or연산 결과가, y에 x(a|b)와 c의 or 연산 결과가, z에 y(a|b|c)와 d의 or 연산 결과가 저장된다.

테이블이(가) 표시된 사진

자동 생성된 설명

<그림11. 4-input or gate simulation code>

위 그림11는 4개의 입력을 가진 OR Gate의 시뮬레이션 소스 파일이다. Reg변수 aa, bb, cc, dd wire변수 xx, yy, zz가 선언되었고, 디자인 소스파일의 변수와 각각 매칭하였다. Initial문을 통해 입력 변수와 매칭된 aa, bb, cc, dd에 1의 값을 초기값으로 부여하였다. 이후 aa는 20ns마다, bb는 40ns마다, cc는 80ns마다, dd는 160ns 마다 값을 변화하도록 하였다. 이러한 반복을 1000ns만큼 실행한 뒤, 프로그램을 종료한다.



<그림12. 4-input or gate simulation>

디자인 소스파일과 시뮬레이션 소스 파일을 바탕으로 run simulation을 한 결과 그림 12과 같은 시뮬레이션 결과가 나왔다. Aa와 bb 중 하나라도 1의 값을 가지는 경우에는 xx또한 1을 출력하며, aa, bb, cc 중 하나라도 1의 값을 가지는 경우에는 yy까지 1의 값을 출력하며, aa, bb, cc, dd 중 하나라도 1의 값을 가지는 경우에는 zz까지 1의 값을 출력한다. 모든 수가 0의 값을 가지는 경우에는 0의 값을 출력한다. 이로서, or gate는 모든 경우가 0인 경우에 0을, 하나라도 0이 아닌 1인 경우 경우 1을 출력함을 알 수 있다.

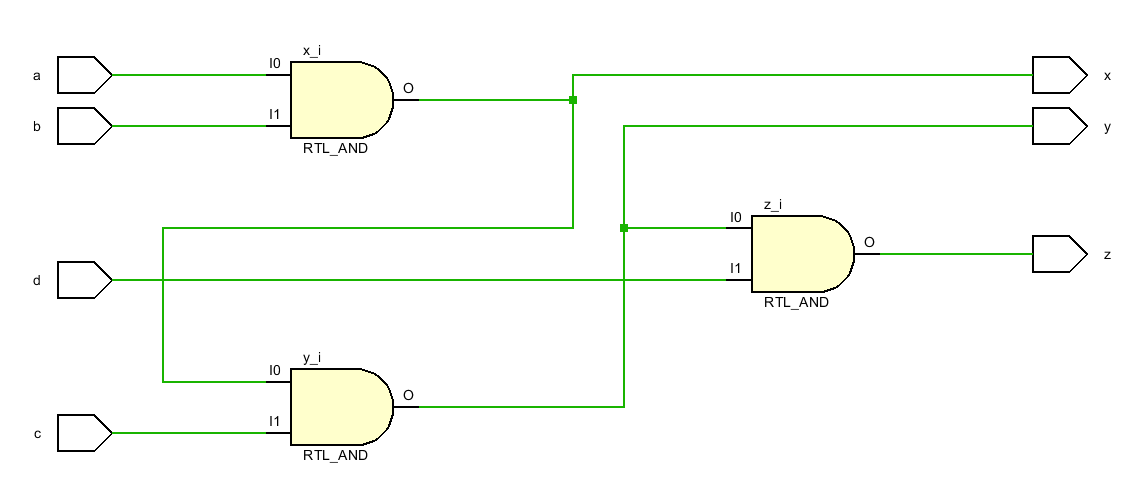
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input a | Input b | Input c | Input d | Output x | Output y | Output z |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

<table 4. 4-input or gate truth table>

위 표는 4개의 입력을 가진 OR Gate의 진리표를 나타낸다.

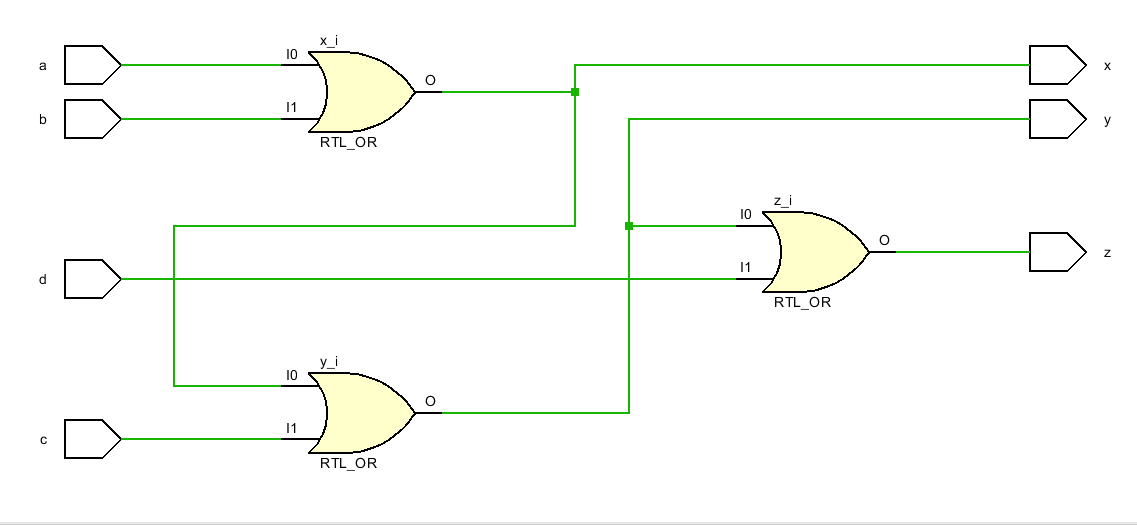
6.

실습을 통해, AND Gate와 OR Gate를 구현해보았다. 뿐만 아니라 AND와 OR 게이트를 중첩하여 사용하여 3개 이상의 입력값에 대한 연산 결과도 출력하였다. AND Gate는 모든 입력이 1인 경우 1을 출력하며, 그렇지 않은 경우에는 0을 출력한다. 아래 그림은 4개의 입력을 가지는 AND Gate의 Schematics이다.



<그림 13. 4-input AND gate schematics>

OR Gate는 모든 입력이 0인 경우 0을 출력하며, 그렇지 않은 경우에는 1을 출력한다. 아래 그림은 4개의 입력을 가지는 OR Gate의 Schematics이다.



<그림 14. 4-input OR gate Schematics>

7.

구현한 NOT, AND, OR 게이트 이외에도 XOR, NOR, NAND와 같은 논리 게이트가 존재한다.

XOR은 배타적 논리합이라고 하며, 2개의 입력이 같은 경우 false(0)을, 다를 경우 true(1)을 반환한다. 다음은 XOR에 대한 진리표이다.

|  |  |  |
| --- | --- | --- |
| Input a | Input b | Output c(a XOR b) |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

<table5. 2-input xor-gate truth table>

NOR은 부정 논리합이라고 하며, OR의 부정형이다. 따라서 모든 입력이 false(0)이면, true(1)가, 그렇지 않은 경우 false(0)을 반환한다. 다음은 NOR에 대한 진리표이다.

|  |  |  |
| --- | --- | --- |
| Input a | Input b | Output c(a XOR b) |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

<table6. 2-input nor-gate truth table>

NAND는 AND의 부정형으로, 모든 입력이 true(1)이면, false(0)을, 그렇지 않은 경우 true(1)를 반환한다. 다음은 NAND에 대한 진리표이다.

|  |  |  |
| --- | --- | --- |
| Input a | Input b | Output c(a XOR b) |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

<table7. 2-input nand-gate truth table>

8. 참고 문헌

컴퓨터공학실험2 교재