3주차 결과보고서

전공 : 컴퓨터공학과 학년 : 3학년 학번 : 20211558 이름 : 윤준서

**1. FPGA 동작법**

FPGA의 동작은 크게 6단계로 나눠진다. 단계별 과정은 다음과 같다.

1 - **Verilog Coding** : Vivado를 이용해 디자인 소스 코드와 testbench 코드를 작성한다. 이 때 변수 값과 switch의 관계는 switch on = 1, switch off = 0이다.

2 - **Device Assignment** : Vivado 내의 Project Manager -> Settings -> Project Device 를 통해 Project에 Device를 assign한다. 이때 Device는 xc7a75tfgg484-1(Artix-7) 로 지정하는데, Artix-7 FPGA는 로직과 신호 처리, 임베디드 메모리, 메모리 인터페이스 등에 최적화된 성능을 제공하는 FPGA Device 이다.

3 - **Pin Assignment** : . Vivado 내의 Project Manager -> Add Sources -> Add or create constraints -> Create File 을 통해 .xdc 파일을 생성하고, 해당 파일에 FPGA pin list에서 할당할 Pin과 Verilog 소스의 port를 연결한다. 이때 Pin은 Verilog의 input, output 변수와 같다.

4 - **Synthesis/Implement** : Vivado 내의 Synthesis -> Run Synthesis, Implementation -> Run Implementation -> Open Implemented Design 을 통해 Synthesis Design과 Route Design을 생성한다.

5 - **Device Configuration** : Vivado 내의 Program and debug -> Generate Bitstream -> Open Hardware Manager를 통해 Device Configuration을 실행한다.

6 - **FPGA 연결 후 동작** : Vivado 내의 Program and Debug -> Open Hardware Manager -> Open Target -> Auto Connect 후 Project[이름] -> Project[이름]\_runs -> impl\_1 폴더에 debug\_nets.ltx 파일을 추가한 다음, Program and Debug -> Program Device -> Debug Probes File 에서 직전에 추가한 debug\_nets.ltx 파일을 선택하고, Program Device를 지정하여 외부 FPGA 기기에 연결할 수 있다.

**2. 3-input AND gate**

3-input 2-output AND gate의 Boolean 식은 다음과 같다.

**D = A • B**

**E = C • D**

**테이블이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**Vivado를 통해 작성한 디자인 소스와 testbench 코드, simulation 결과는 다음과 같다.

**텍스트, 스크린샷, 전자기기, 컴퓨터이(가) 표시된 사진

자동 생성된 설명**

이를 통해 작성한 진리표는 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IN A** | **IN B** | **IN C** | **OUT D** | **OUT E** |
| **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** |

**3. 4-input AND gate**

4-input 3-output AND gate의 Boolean 식은 다음과 같다.

**E = A • B**

**F = C • E**

**G = D • F**

테이블이(가) 표시된 사진

자동 생성된 설명Vivado를 통해 작성한 디자인 소스와 testbench 코드, simulation 결과는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 전자기기, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

이를 통해 작성한 진리표는 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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** | **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** |

**4. 3-input OR gate**

3-input 2-output OR gate의 Boolean 식은 다음과 같다.

**D = A + B**

**E = C + D**

**테이블이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**Vivado를 통해 작성한 디자인 소스와 testbench 코드, simulation 결과는 다음과 같다.

**텍스트, 스크린샷, 전자기기, 컴퓨터이(가) 표시된 사진

자동 생성된 설명**

이를 통해 작성한 진리표는 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IN A** | **IN B** | **IN C** | **OUT D** | **OUT E** |
| **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** |

**5. 4-input OR gate**

4-input 3-output OR gate의 Boolean 식은 다음과 같다.

**E = A + B**

**F = C + E**

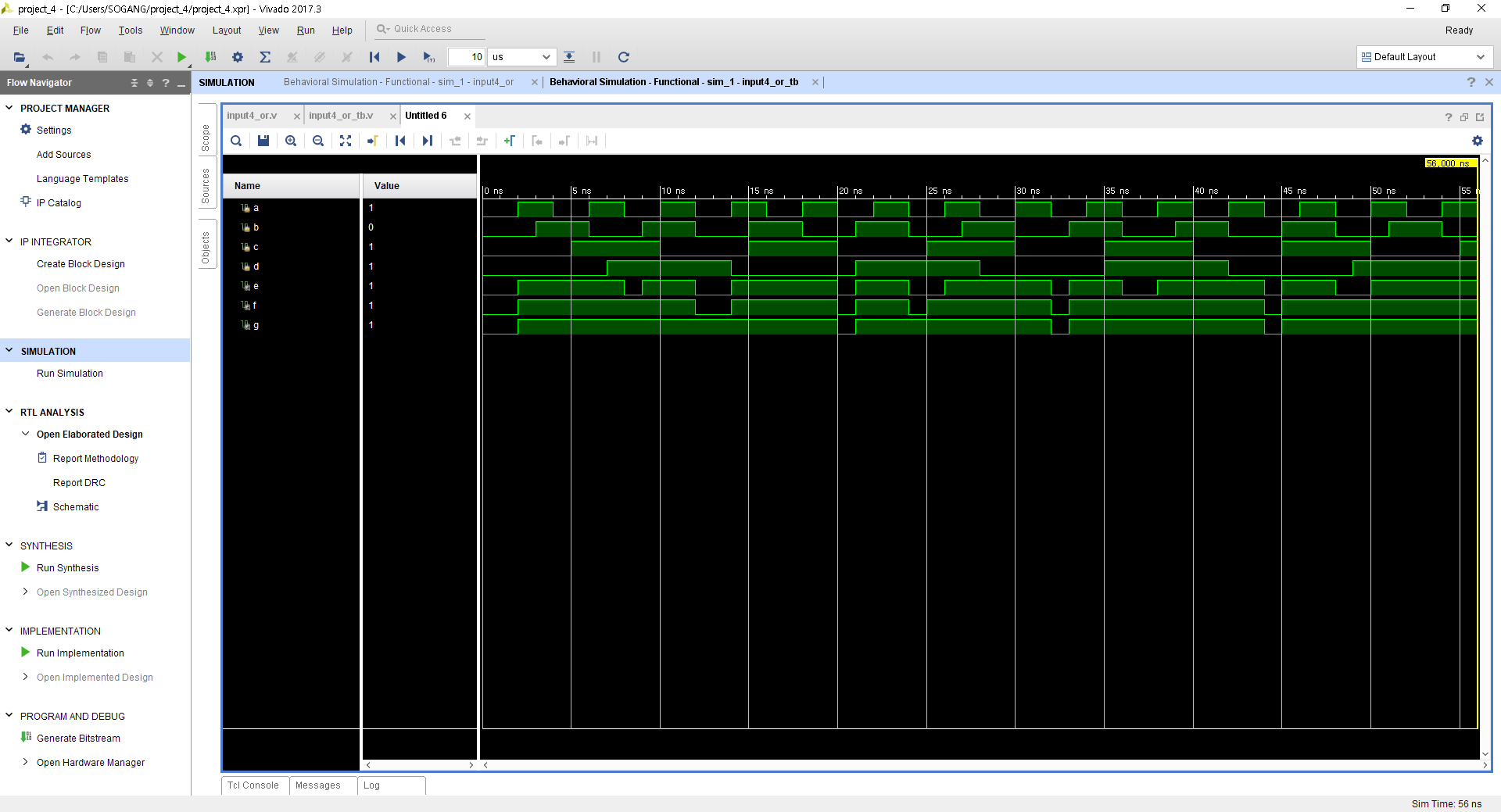
**G = D + F**

Vivado를 통해 작성한 디자인 소스와 testbench 코드, simulation 결과는 다음과 같다

**텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명**텍스트이(가) 표시된 사진

자동 생성된 설명



이를 통해 작성한 진리표는 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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** | **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** |

**6. 결과 검토 및 논의 사항**

**AND vs OR**

AND 연산은 피연산자가 단 하나라도 값이 0이면 결과값이 0이 나온다. 위의 3-input과 4-input의 경우 진리표의 값이 연산을 거듭할 수록 0의 빈도가 증가함을 알 수 있다.

반면 OR 연산은 피연산자가 단 하나라도 값이 1이면 결과값이 1이 나온다. 위의 3-input과 4-input의 경우 진리표의 값이 연산을 거듭할 수록 AND와 반대로 1의 빈도가 증가함을 알 수 있다.

**AND, OR 연산의 성질**

AND 연산과 OR 연산은 교환법칙과 결합법칙이 성립한다.

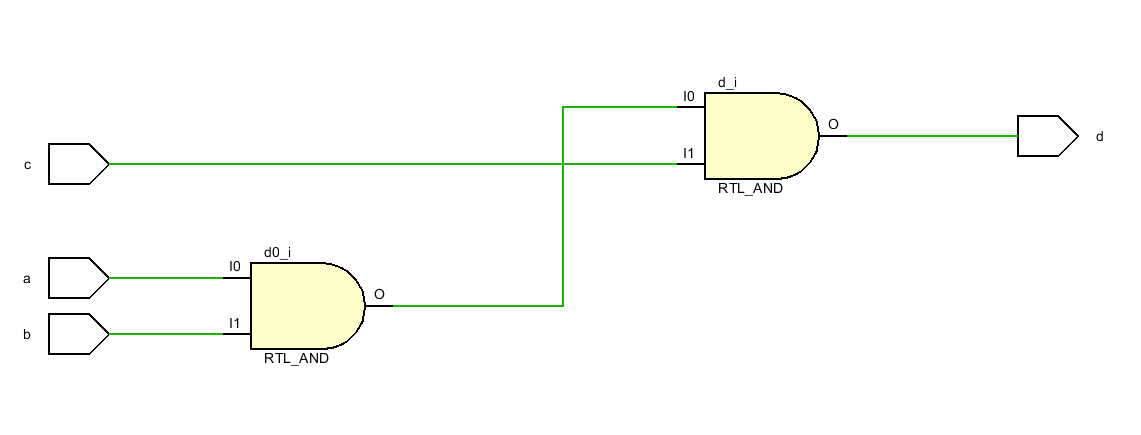
**3-input AND** 연산의 경우,

**D = A • B**

**E = C • D**

위의 식의 E의 값이 **E = A & B & C**과 같음을 알 수 있다. 이는 OR 연산에도 똑같이 성립한다.

하지만 실제 회로에서는 두 개의 output을 이용하는 것을 Schematic Diagram을 통해 알 수 있다.



**4-input OR** 연산의 경우,

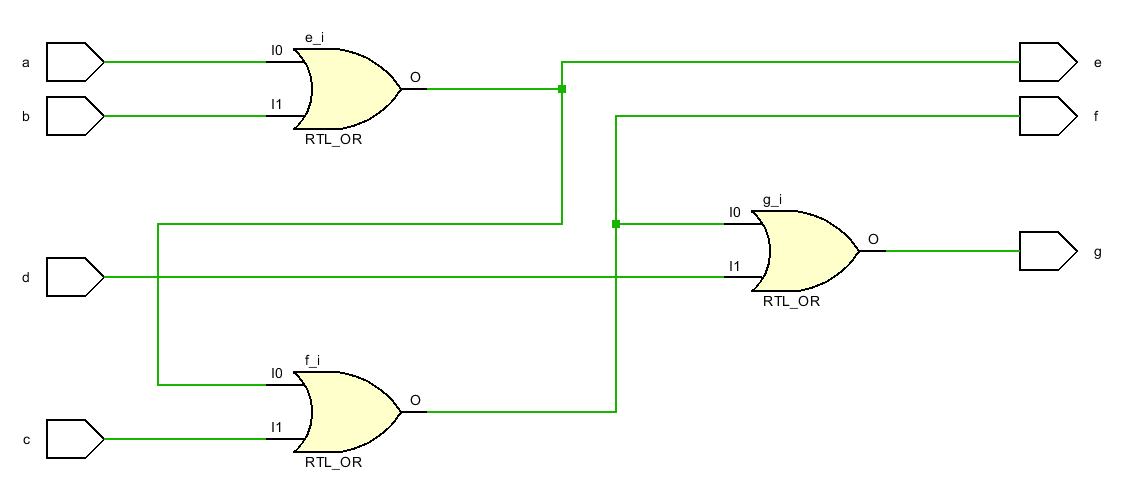
**E = A + B**

**F = C + E**

**G = D + F**

위의 식의 G의 값이 G = A | B | C과 같음을 알 수 있다. 이는 AND 연산에도 똑같이 성립한다.

하지만 실제 회로에서는 세 개의 output을 이용하는 것을 Schematic Diagram을 통해 알 수 있다.



**7. 추가 이론**

**AND 게이트**는 논리곱을 구현하는 디지털 논리 게이트다. AND 게이트 대신 NAND, NOR 게이트를 이용하여 AND 게이트의 역할을 대신할 수 있다..

**OR 게이트**는 논리합을 구현하는 디지털 논리 게이트다. OR 게이트 대신 NAND, NOR 게이트를 이용하여 OR 게이트의 역할을 대신할 수 있다..

지금까지는 AND와 OR 게이트의 연산을 다루었지만, 논리 게이트는 이외에도 여러가지가 있다.

다른 논리 게이트들의 정의와 논리식은 다음과 같다.

**NOT** - 입력 값을 반전시킨다. 0을 입력하면 1이, 1을 입력하면 0이 출력된다. **Y = A'**

**BUFFER** - 입력 값을 그대로 출력한다. **Y = A**

**NAND** - AND의 NOT이다. **Y = (A** • **B)'**

**NOR** - OR의 NOT이다. **Y = (A + B)'**

**XOR** - 배타적 논리합, 두 입력 값의 값이 다르면 1이, 같으면 0이 출력된다. **Y = (A**•**'B) + (A**•**B')**

**XNOR** - XOR의 NOT이다. **Y = ( (A** • **'B) + (A** • **B') )'**