3주차 결과보고서

전공: 컴퓨터공학과 학년: 4학년 학번: 20191583 이름: 김태곤

**1. FPGA 동작법을 설명하시오.**

FPGA는 아래와 같은 순서로 동작을 검증할 수 있다.

1. Verilog Coding

Verilog를 사용하여 원하는 논리회로를 코드로 작성한다. 이때 Design Sources를 통해 회로의 구조를 정의한다. Simulation의 testbench코드를 통해 회로 동작을 모델링하고 테스트하는데 사용된다.

1. Device Assignment

FPGA의 Device를 정한다. Vivado의 Project Manager – Setting – Project device를 통해 지정할 수 있다. 본 실험에서는 Device : xc7a75tfgg484(Artix7)로 지정한다.

1. PIN Assignment

Verilog 코드에서 사용된 실제 FPGA 핀 간의 연결을 설정한다. 주로 Input, Output 변수에 대한 설정을 한다. Vivado의 Project Manager – Add Source – Add or create constraints를 통해 xdc파일을 생성하며, pin list에서 할당하고 싶은 Pin과 Verilog 소스의 port를 링크하고, 여러 속성들을 정의한다.

1. Synthesis/Implementation

Vivado의 Run Synthesis, Run Implementation을 통해 코드를 합성하고, FPGA에 맞게 구현한다.

1. Device Configuration

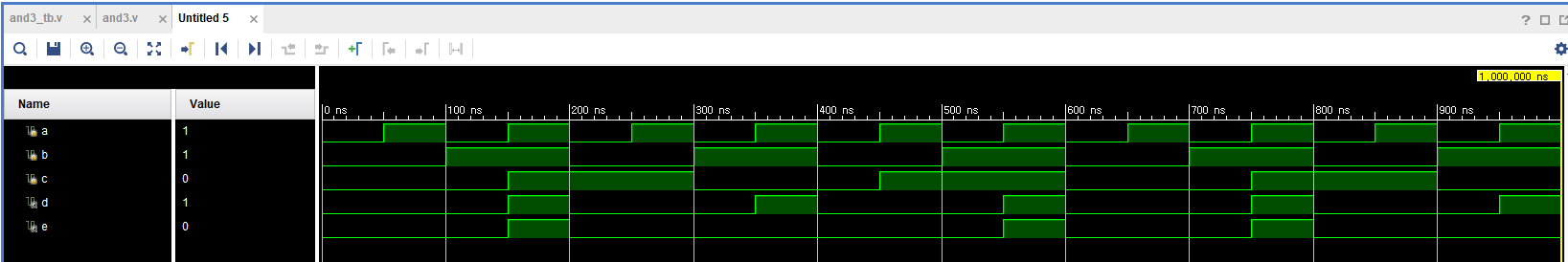
Program and Debug - Generate Bitstream – Open Hardware Manager를 통해 구성 파일을 FPGA에 로드하여 하드웨어에 구현한다.

1. FPGA 작동

Open Target – auto connect를 통해 연동을 하고, Project\_runs – impl\_1에 debug\_nets.ltx 파일을 넣는다. 이후 Program Device에서 ltx파일을 설정한 후 동작 결과를 관찰한다.

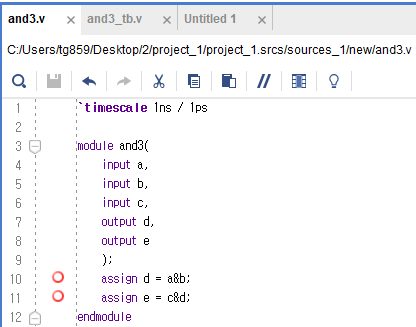
**2. 3-input AND gate의 simulation 결과 및 과정에 대해서 설명하시오.**

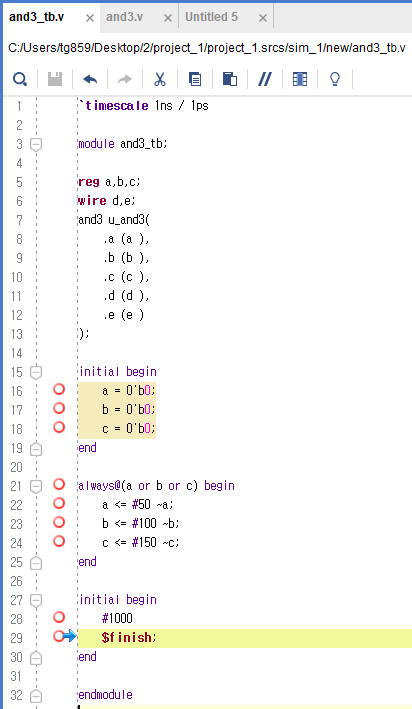
**(3 input, 2 output)[3장 ppt 31 page 참조 , 진리표 작성]**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | 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 |

Input으로 받은 a,b,c는 각각 50, 100, 150ns마다 값이 변하도록 Verilog Code를 작성하였다. d의 값은 a&b, e의 값은 a&b&c를 나타낸다. simulation 결과를 보면 알 수 있듯이 a와 b가 1인 경우 d의 값은 1이 되었고, a,b,c 모두 1인 경우 e의 값이 1이 나왔다. 즉, 이 구조는 입력 값들의 AND 연산을 단계적으로 수행하여 최종적으로 모든 입력 값이 참일 때에만 참을 출력하는 3-다중입력 AND 게이트이다.

****



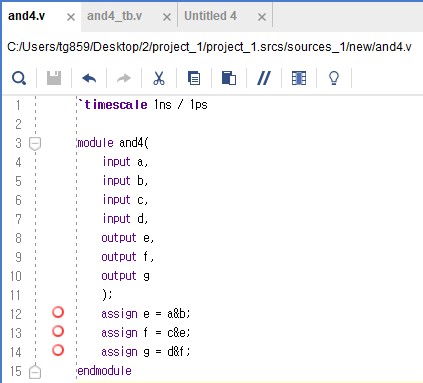
**3. 4-input AND gate의 simulation 결과 및 과정에 대해서 설명하시오.**

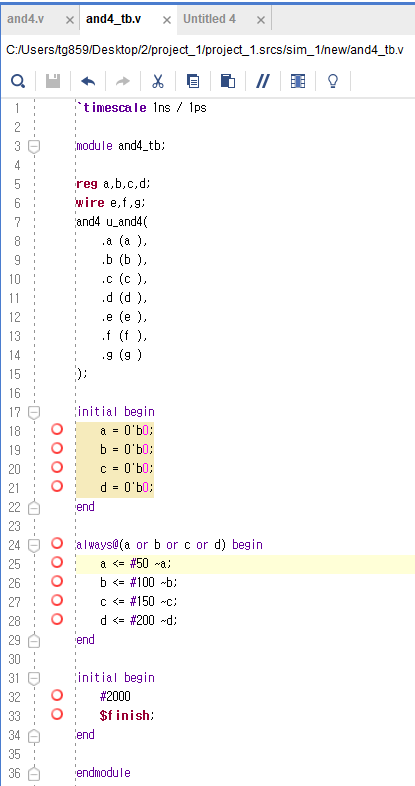
**(4 input, 3 output)[3장 ppt 33 page 참조 , 진리표 작성]**

****

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

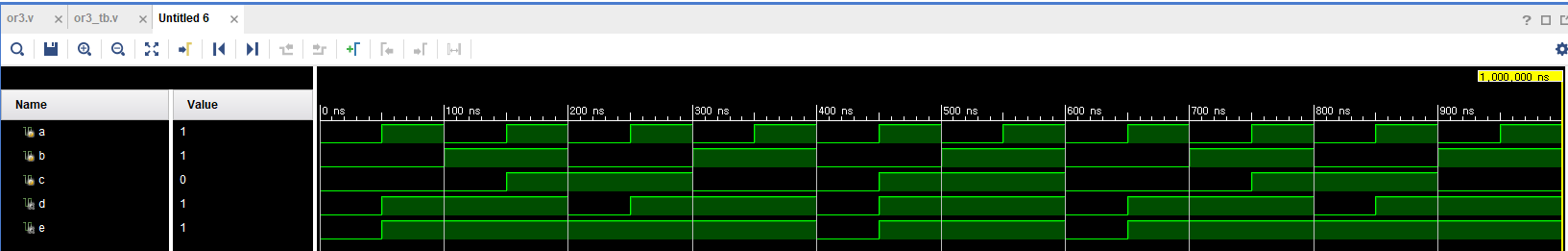
Input으로 받은 a,b,c,d는 각각 50, 100, 150, 200ns마다 값이 변하도록 Verilog Code를 작성하였다. e의 값은 a&b, f의 값은 a&b&c, g의 값은 a&b&c&d를 나타낸다. simulation 결과를 보면 알 수 있듯이 a와 b가 1인 경우 e의 값은 1이 되었고, a,b,c 모두 1인 경우 f의 값이 1이 나왔으며 a,b,c,d 모두 1인 경우에만 g의 값이 1이 되었다. 즉, 이 구조는 입력 값들의 AND 연산을 단계적으로 수행하여 최종적으로 모든 입력 값이 참일 때에만 참을 출력하는 4-다중입력 AND 게이트이다.

****

****

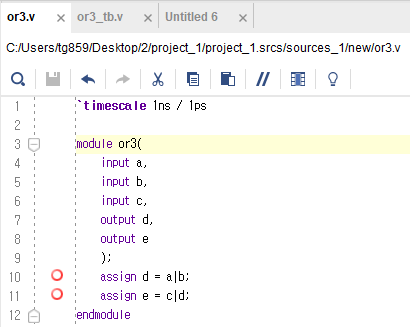
**4. 3-input OR gate의 simulation 결과 및 과정에 대해서 설명하시오.**

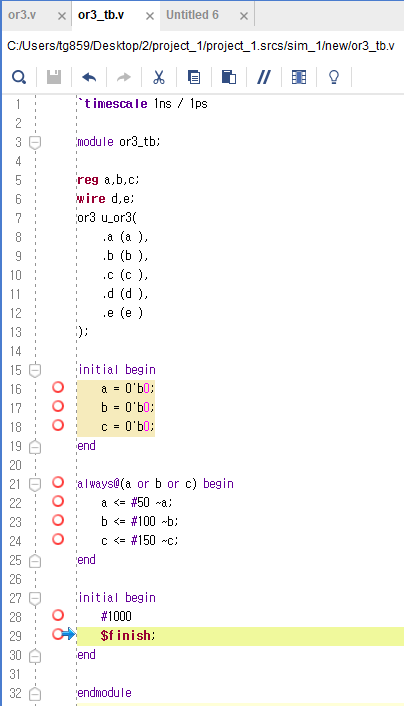
**(3 input, 2 output)[3장 ppt 35 page 참조, 진리표 작성]**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | 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 |

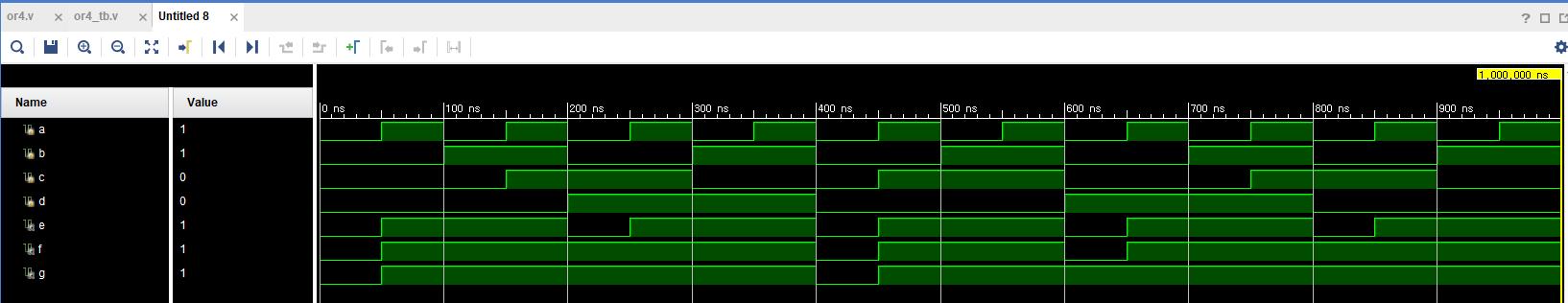
Input으로 받은 a,b,c는 각각 50, 100, 150ns마다 값이 변하도록 Verilog Code를 작성하였다. d의 값은 a|b, e의 값은 a|b|c를 나타낸다. simulation 결과를 보면 알 수 있듯이 a와 b 둘 중 하나라도 1인 경우 d의 값은 1이 되었고, a,b,c의 값 중 하나라도 1이 포함되어 있는 경우 e의 값은 1을 가졌다. 다시 말해 a,b,c 모두 0이 아닌 경우 1의 값을 갖는다. 즉, 이 구조는 입력 값들의 OR 연산을 단계적으로 수행하여 최종적으로 입력 값 중 참을 포함하는 경우 참을 출력하는 3-다중입력 OR 게이트이다.

****

****

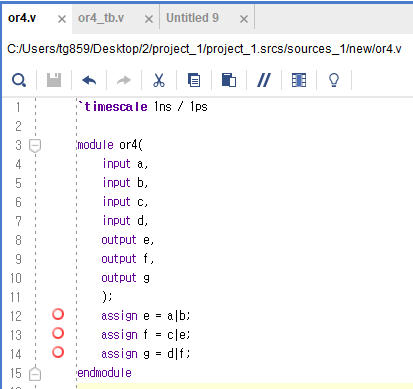
**5. 4-input OR gate의 simulation 결과 및 과정에 대해서 설명하시오.**

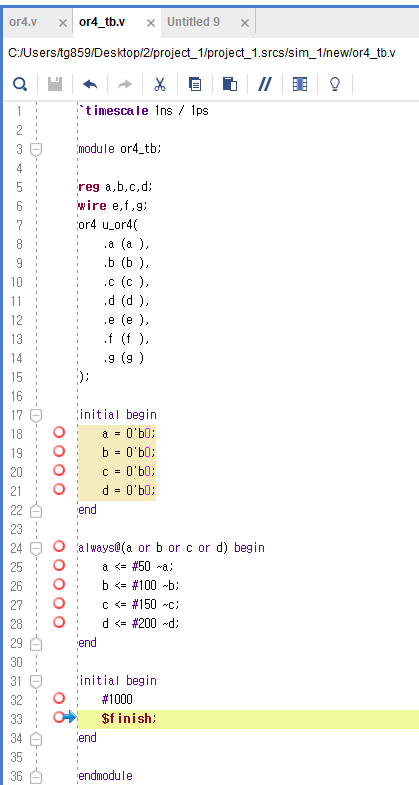
**(4 input, 3 output)[3장 ppt 37 page 참조, 진리표 작성]**

****

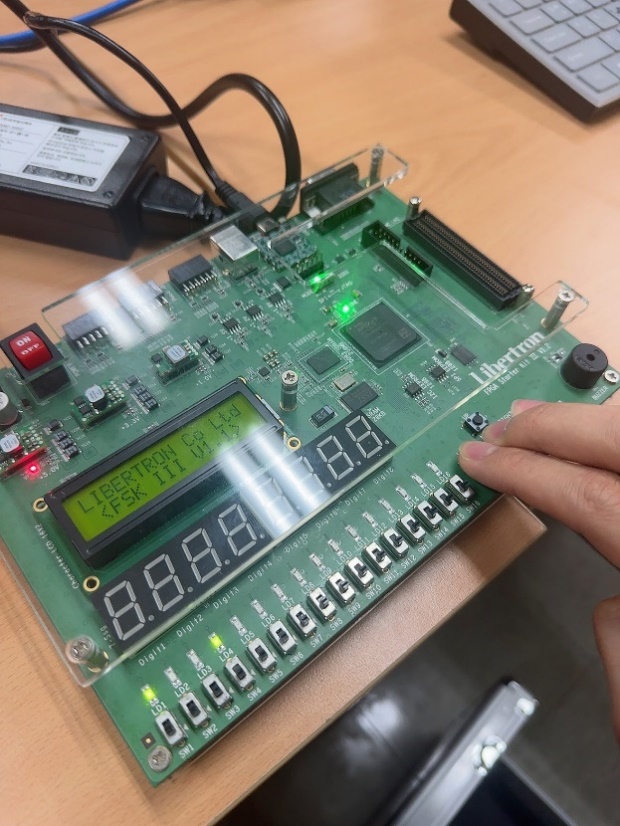
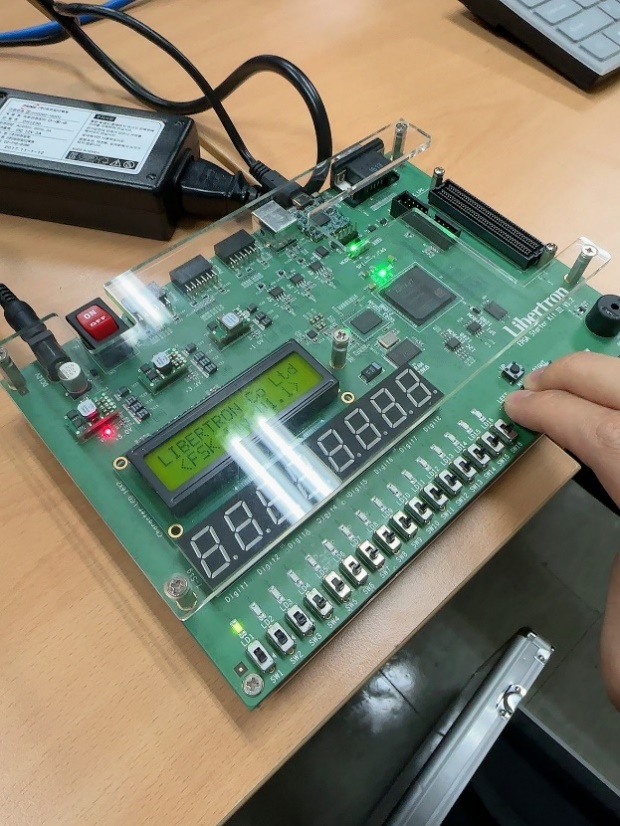
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1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

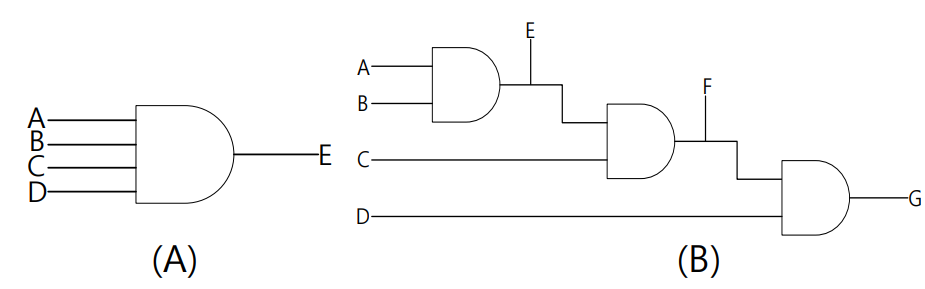
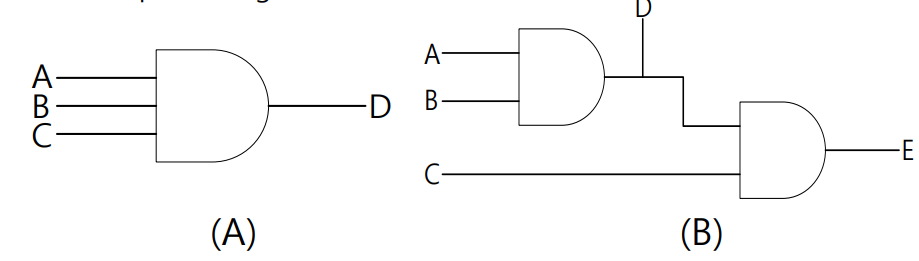
Input으로 받은 a,b,c,d는 각각 50, 100, 150, 200ns마다 값이 변하도록 Verilog Code를 작성하였다. e의 값은 a|b, f의 값은 a|b|c를 나타내며, g의 값은 a|b|c|d를 나타낸다. simulation 결과를 보면 알 수 있듯이 a와 b 둘 중 하나라도 1인 경우 e의 값은 1이 되었고, a,b,c의 값 중 하나라도 1이 포함되어 있는 경우 f의 값은 1을 가졌다. 또한 a,b,c,d중 하나라도 1을 가진 경우 g의 값은 1이 되었다. 다시 말해 a,b,c,d 모두 0이 아닌 경우 1의 값을 갖는다. 즉, 이 구조는 입력 값들의 OR 연산을 단계적으로 수행하여 최종적으로 입력 값 중 참을 포함하는 경우 참을 출력하는 4-다중입력 OR 게이트이다.

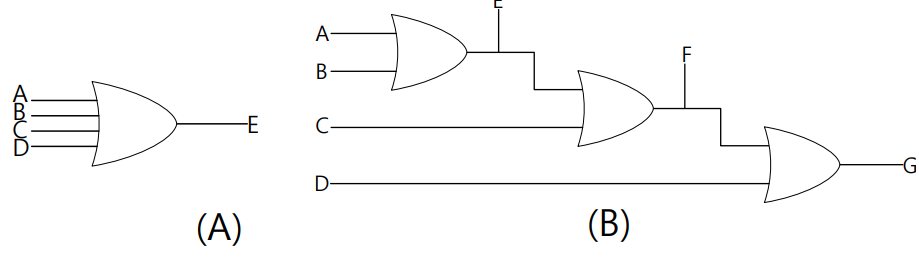
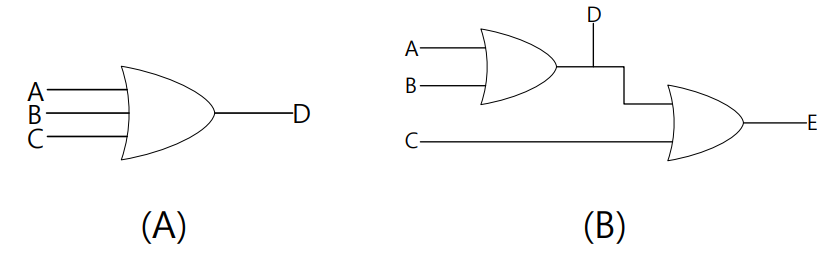
****

****

**6. 결과 검토 및 논의사항.**

****

****

시뮬레이션과 FPGA를 통해 3,4 AND, OR 게이트를 직접 살펴보았다. 그 결과 이상적인 진리표와 동일한 결과의 시뮬레이션 결과를 도출해낼 수 있었고, FPGA를 통해 직접 입력 신호를 바꾸어가며 성공적으로 동작을 비교할 수 있었다.

위 사진에 나와있는 각각의 논리회로 쌍들은 서로 비슷해 보인다. 실제로 위에 방식대로 설계된 게이트들은 A와 B 모두 같은 결과값을 가지게 된다. 서로 다른 구현 방식으로 표현 했지만 같은 결과를 가지는 A와 B는 같은 것일까?

최종 출력 결과는 같지만 내부적인 구조의 차이로 인한 다양한 특성 차이가 존재한다. A 구현은 단일 게이트로 한번에 입력을 받아 하나의 결과를 출력한다. B 구현은 입력과 과정을 여러 단계로 나누어 구현하여 최종적으로 단일 게이트와 같은 기능을 수행하게 된다. 이 두 구현은 최종적인 기능은 같지만, 내부 구조가 다르며, 이러한 차이는 전파 지연, 칩 영역의 사용, 전력 소모 등의 측면에서 다른 특성을 나타낼 수 있다. 따라서 우리는 요구사항에 따라 가장 적합한 구현 방식을 선택해야 한다.

**7. 추가 이론 조사 및 작성.**

3주차 실험에서는 AND와 OR 게이트를 이용해 실습을 하고 결과 값을 비교해 볼 수 있었다. 논리 회로 게이트에는 AND, OR 이외에도 다양한 게이트들이 존재한다.

NOT : 인버터라고도 하며, 단일 입력 신호의 논리 상태를 반전시킨다. 즉, 입력이 1이면 출력은 0이 되고, 입력이 0이면 출력은 1이 된다.

BUFFER : 입력된 신호를 그대로 출력으로 전달한다. 이 게이트는 주로 신호의 세기를 강화(버퍼링)하거나, 여러 부하(load)에 신호를 분배하기 위해 사용된다.

NAND : AND 게이트의 출력을 반전시킨 것과 같다. 모든 입력이 1일 때만 출력이 0이 되고, 그 외의 경우에는 출력이 1이 된다. NAND 게이트는 보편적 게이트로 간주되어, 다른 모든 타입의 게이트를 만드는 데 사용될 수 있다.

NOR : OR 게이트의 출력을 반전시킨 것이다. 모든 입력이 0일 때만 출력이 1이 되고, 하나라도 입력이 1이면 출력이 0이 된다. NOR 게이트 역시 다른 모든 게이트를 구성할 수 있는 보편적 게이트이다.

XOR : 입력 신호들 중에서 홀수 개의 신호가 1일 때 출력이 1이 되는 게이트이다. 즉, 두 입력이 서로 다를 때만 1을 출력한다. XOR 게이트는 덧셈 연산에서의 합(bitwise addition)을 계산하는 데 사용된다.

이처럼 다양한 게이트들이 디지털 논리 회로를 구성하고, 복잡한 데이터 처리, 컴퓨터 메모리, 마이크로프로세서 디자인 등 다양한 곳에서 필수적인 역할을 한다.