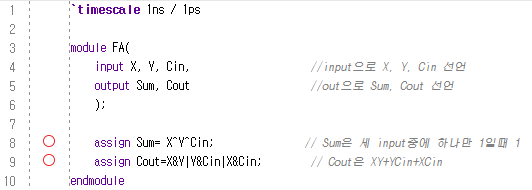
**Lab9.**

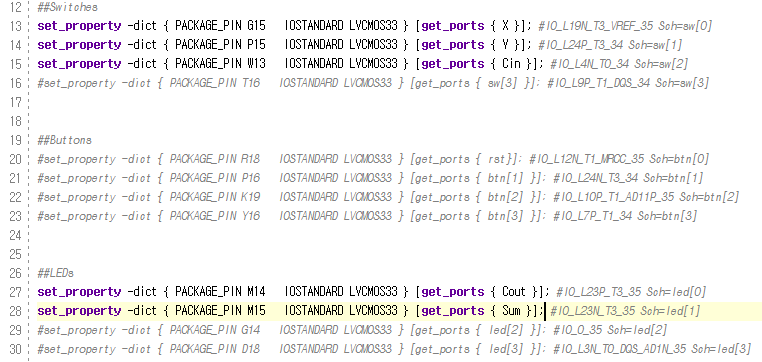
# **Homework1. Full Adder**

**Verilog Code / 주석**

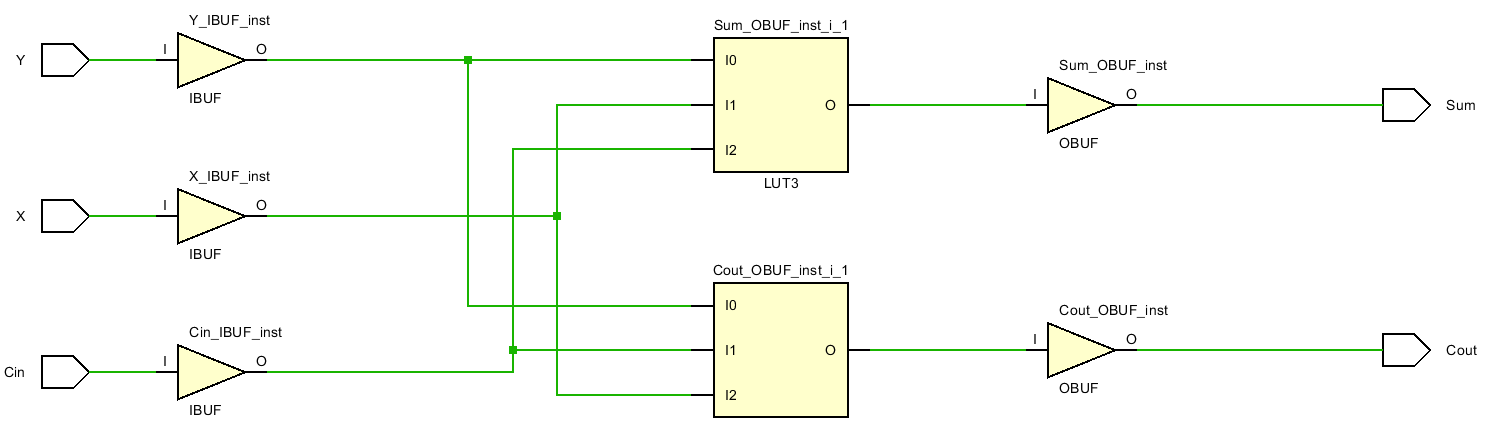
FA.v



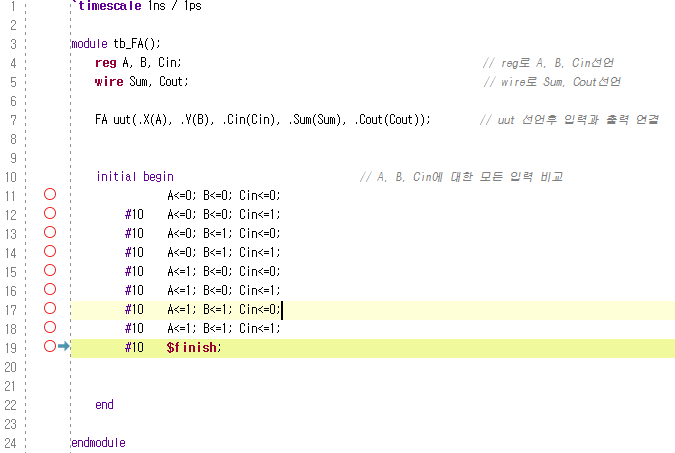
Zybo-Z7-Master.xdc



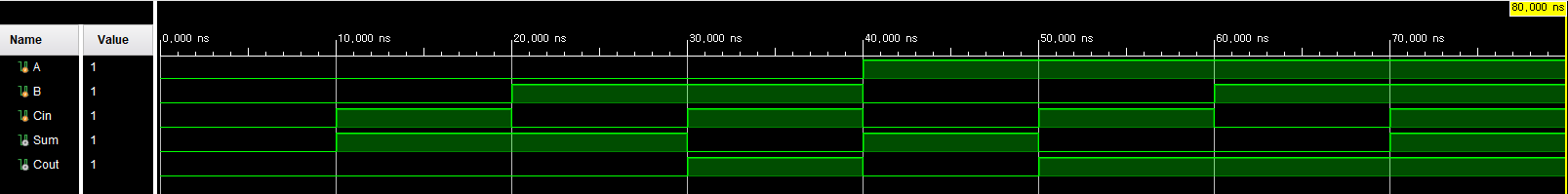
**Synthesis Report**

****

**Test Bench Code**



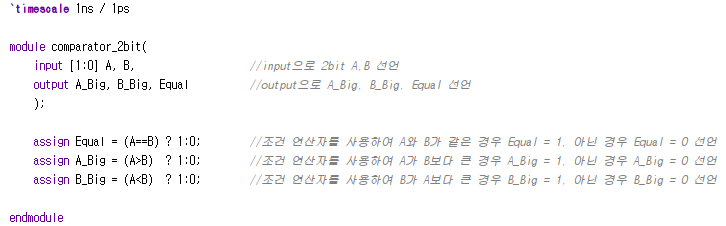
**Simulation Result**



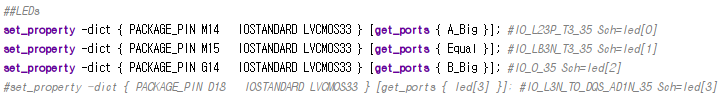
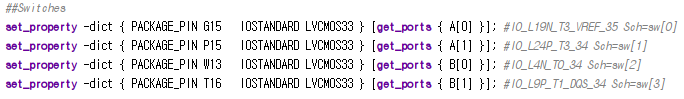
|  |
| --- |
| **Discussion** |
| . - Verilog Coding을 시작하기 전 작성한 Block Diagram      - 작성한 Verilog Module에 대한 설명  FA 모듈은 input으로 X, Y, Cin을 받고 출력으로 Sum과 Cout을 출력하도록 설정하였다.  입력에 대해서 X는 G15, Y는 P15, Cin은 W13으로 mapping하였다.  출력에 대해서 Cout은 M14, Sum은 M15로 mapping하였다.  - 작성한 Code가 어떤 동작을 하는지? 왜 그런 동작을 하는지?  작성한 Code는 X, Y, Cin의 관계를 이용하여 Sum과 Cout을 표시하는 동작을 하였다.  Sum은 X^Y^Cin으로 세 input중에 하나만 1일때 1의 값을 갖도록 설정하였다.  Cout은 X, Y중 하나가 1의 값을 가지고 Cin이 1이거나 X, Y가 모두 1인 경우 carry가 발생하므로 이와 같이 설정하였다.  - 작성한 Test Bench Code가 어떤 동작을 하는지?  Test Bench는 FA모듈을 불러오고 입력과 출력을 연결한 후 입력에 대해 출력이 어떻게 변하는지 확인하는 코드로 작성하였다.  이때 주어진 표에서 A, B, Cin을 입력으로 하여서 A와 X, B와 Y로 연결하였다.  이후 8개의 모든 경우의 수에 대하여 test하였다.  - Simulation 파형이 왜 그렇게 나온것인지?  simulation 파형은 이론적인 내용과 동일하게 측정되었다.  A<=0; B<=0; Cin<=0; ->Sum : 0 Carry : 0  A<=0; B<=0; Cin<=1; ->Sum : 1 Carry : 0  A<=0; B<=1; Cin<=0; ->Sum : 1 Carry : 0  A<=0; B<=1; Cin<=1; ->Sum : 0 Carry : 1  A<=1; B<=0; Cin<=0; ->Sum : 1 Carry : 0  A<=1; B<=0; Cin<=1; ->Sum : 0 Carry : 1  A<=1; B<=1; Cin<=0; ->Sum : 0 Carry : 1  A<=1; B<=1; Cin<=1; ->Sum : 1 Carry : 1  이와 동일하게 simulation도 나타났다.  - 작성한 Code가 잘 동작하지 않는다면 어디서 문제가 발생한 것인지?  오류 없이 동작하였다.  - 오류를 해결했다면, 어떤 부분에서 해결을 했고, 왜 오류가 발생하였는지?  오류 없이 동작하였다. |

# **Homework2. 2bit magnitude comparator**

**Verilog Code / 주석**

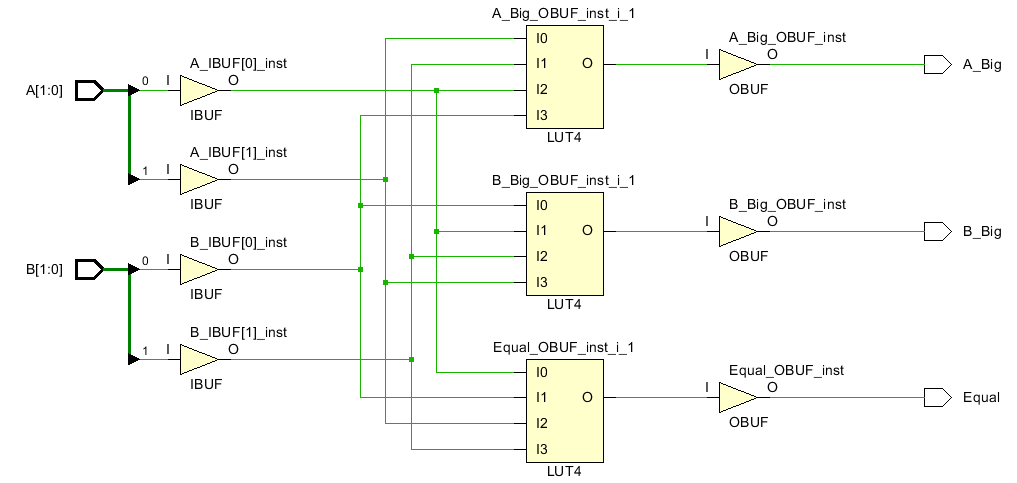


COMPARATOR MODULE

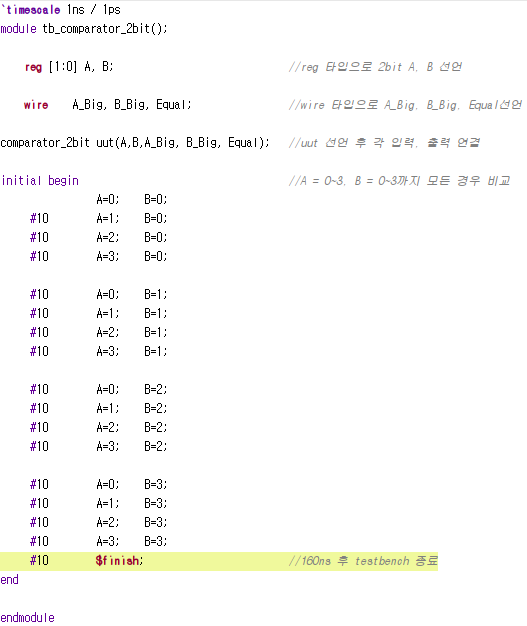


CONSTRAINTS FILE

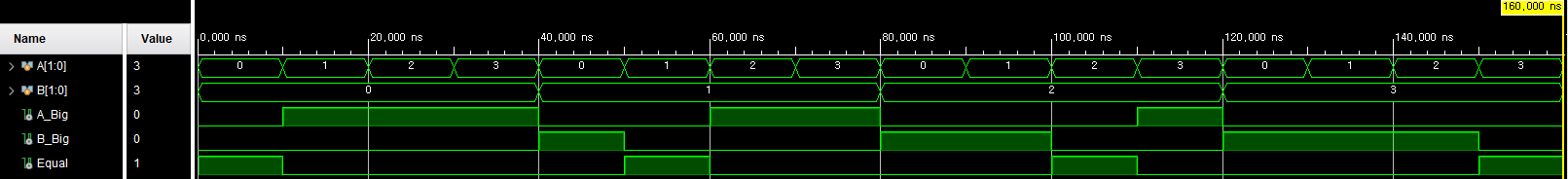
**Synthesis Report**

****

**Test Bench Code**



**Simulation Result**



|  |
| --- |
| **Discussion** |
| . - Verilog Coding을 시작하기 전 작성한 Block Diagram    - 작성한 Verilog Module에 대한 설명  위의 그림과 같이 2bit의 A와 B가 입력된다. 부호가 없는 이진수 2bit의 A와 B의 크기를 비교해서 출력으로 A가 B보다 큰 경우 A\_Big, B가 A보다 큰 경우 B\_Big, A와 B가 같은 경우 Equal이 출력된다. 입력인 A를 slide switch[0]과 slide switch[1]에 mapping 해주었고 B는 slide switch[2]과 slide switch[3]에 mapping해 주었다. 출력인 A\_Big, Equal, B\_Big은 각각 LED[0],[1], [2]에 mapping해 주었다.  - 작성한 Code가 어떤 동작을 하는지? 왜 그런 동작을 하는지?  assign 문과 조건 연산자(?)를 통해 Equal의 경우 A와 B가 같을 경우 1, 같지 않을 경우 0을 출력하도록 해주었고, A\_Big의 경우는 A가 B보다 클 경우 1, 크지 않을 경우 0을 출력하도록 해주었다. 마지막으로 B\_Big도 A\_Big과 유사하게 B가 A보다 클 경우 1, 크지 않을 경우 0을 출력하도록 설계하였다.  - 작성한 Test Bench Code가 어떤 동작을 하는지?  reg 타입으로 input을, wire 타입으로 output을 선언한 후 uut를 선언해 각 input과 output을 연결해 주었다. initial문을 사용하여 10ns마다 A와 B의 값을 바꾸어 주면서 결과값을 관찰하였다. A가 0일 때 B를 0~3, A가 1일 때 B를 0~3, A가 2일 때 B를 0~3, A가 3일 때 B를 0~3으로 바꾸어 주면서 확인할 수 있는 모든 경우의 수에 대해 test해 보았다. 위 16개의 경우의 수를 검증하고 160ns후에는 $finish문을 통해 simulation을 종료하도록 코드를 작성하였다.  - Simulation 파형이 왜 그렇게 나온것인지?  A=0; B=0; -> Equal  A=1; B=0; ->A\_Big  A=2; B=0; ->A\_Big  A=3; B=0; ->A\_Big    A=0; B=1; ->B\_Big  A=1; B=1; ->Equal  A=2; B=1; ->A\_Big  A=3; B=1; ->A\_Big    A=0; B=2; ->B\_Big  A=1; B=2; ->B\_Big  A=2; B=2; ->Equal  A=3; B=2; ->A\_Big    A=0; B=3; ->B\_Big  A=1; B=3; ->B\_Big  A=2; B=3; ->B\_Big  A=3; B=3; ->Equal  작성한 코드를 보았을 때 위와 같은 결과가 나와야 한다. simulation 결과는 10ns마다 값이 바뀌면서 위와 같은 결과가 나옴을 확인할 수 있었고 160ns후에는 코드에서 보았듯이 simulation을 종료하는 것까지 확인할 수 있었다. |