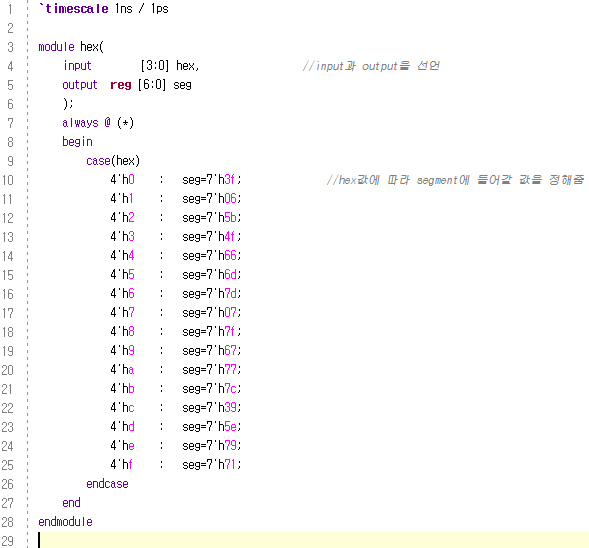
**Lab1.**

# **Homework1.**

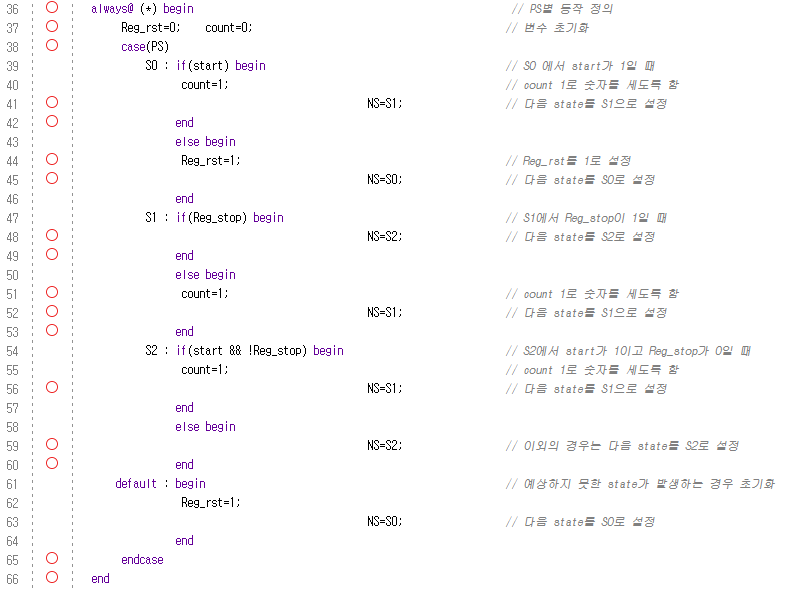
**Verilog Code / 주석**

Hex.v



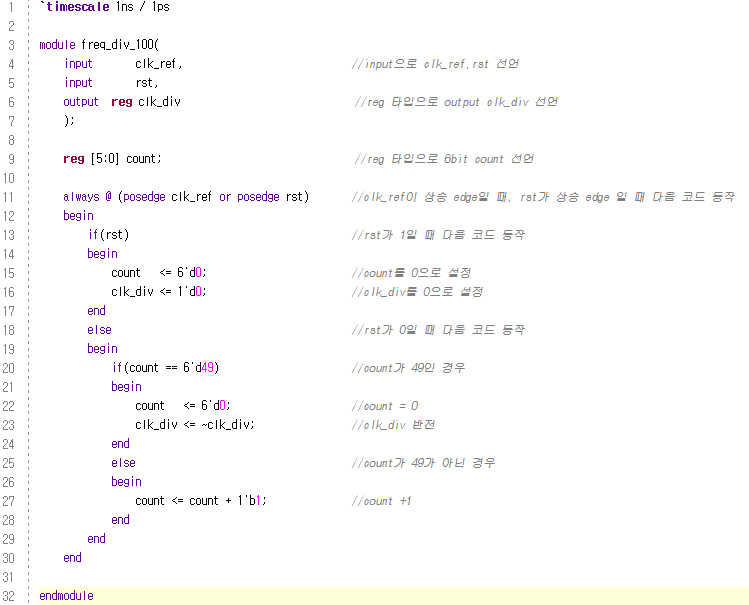
Timer.v



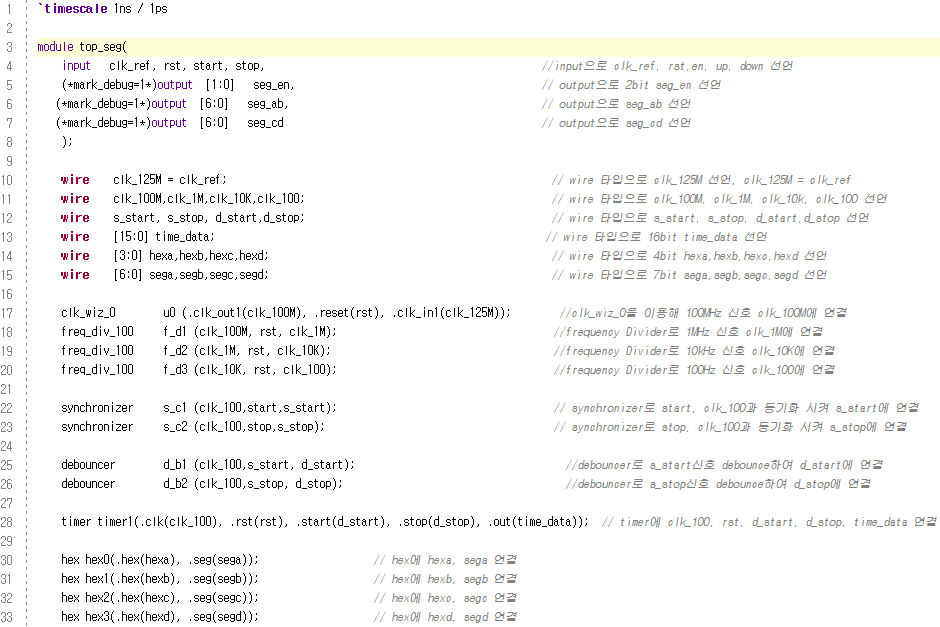


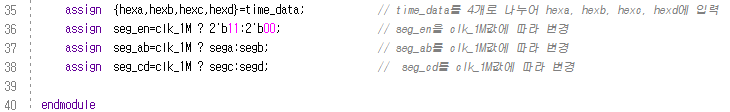


freq\_div\_100.v

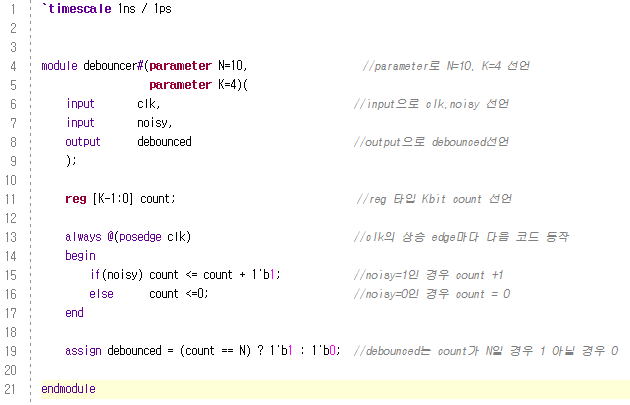


top\_seg.v





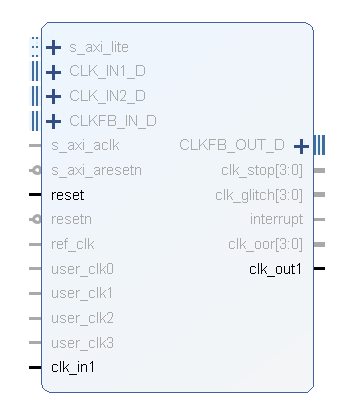
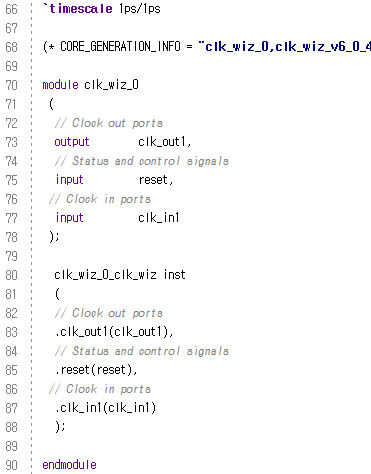
Debouncer.v



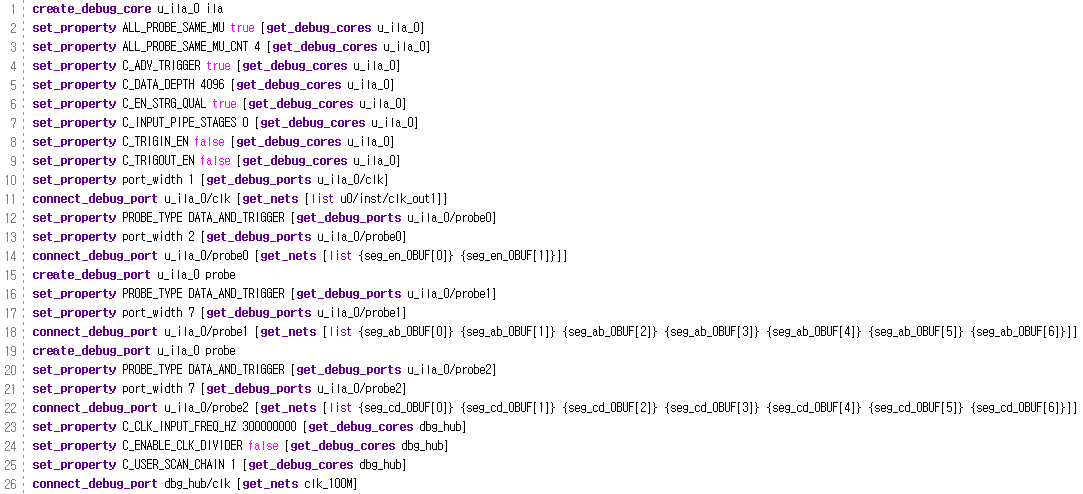
Synchronizer.v



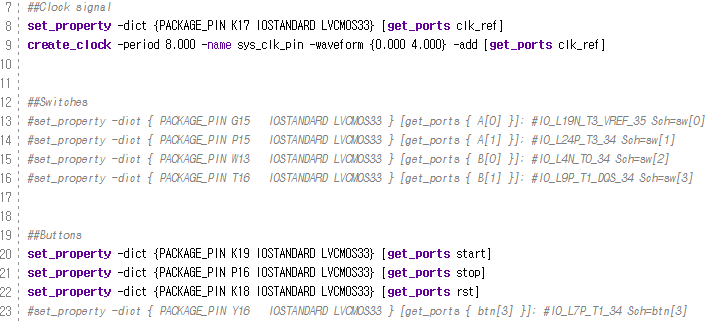
clk\_wiz\_0.v

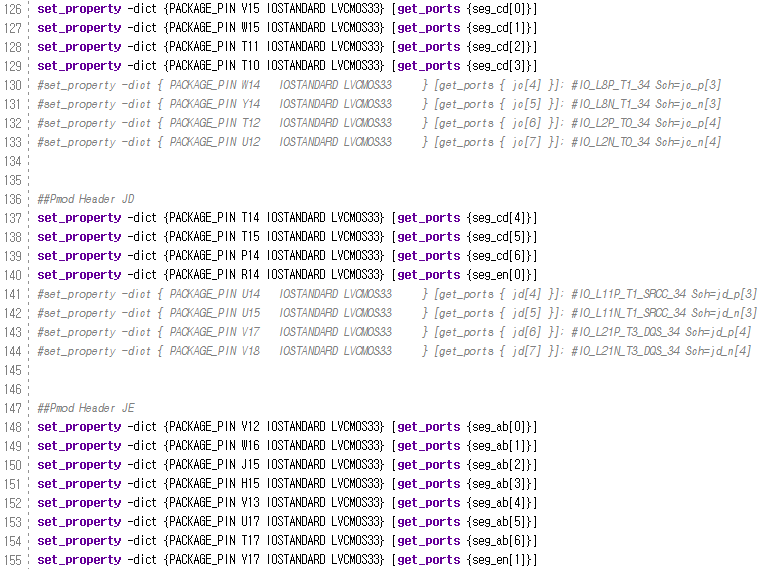


Seg\_debeg.xdc

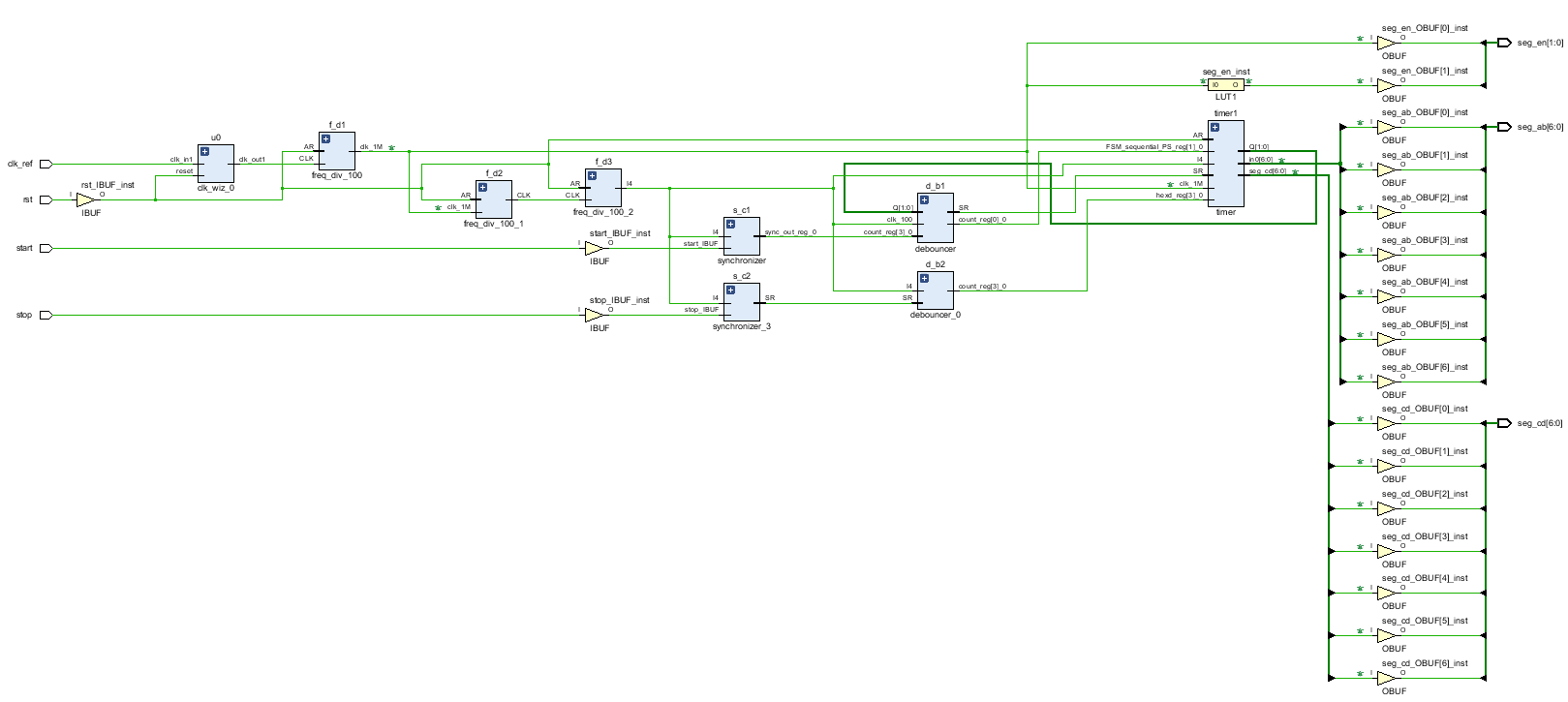


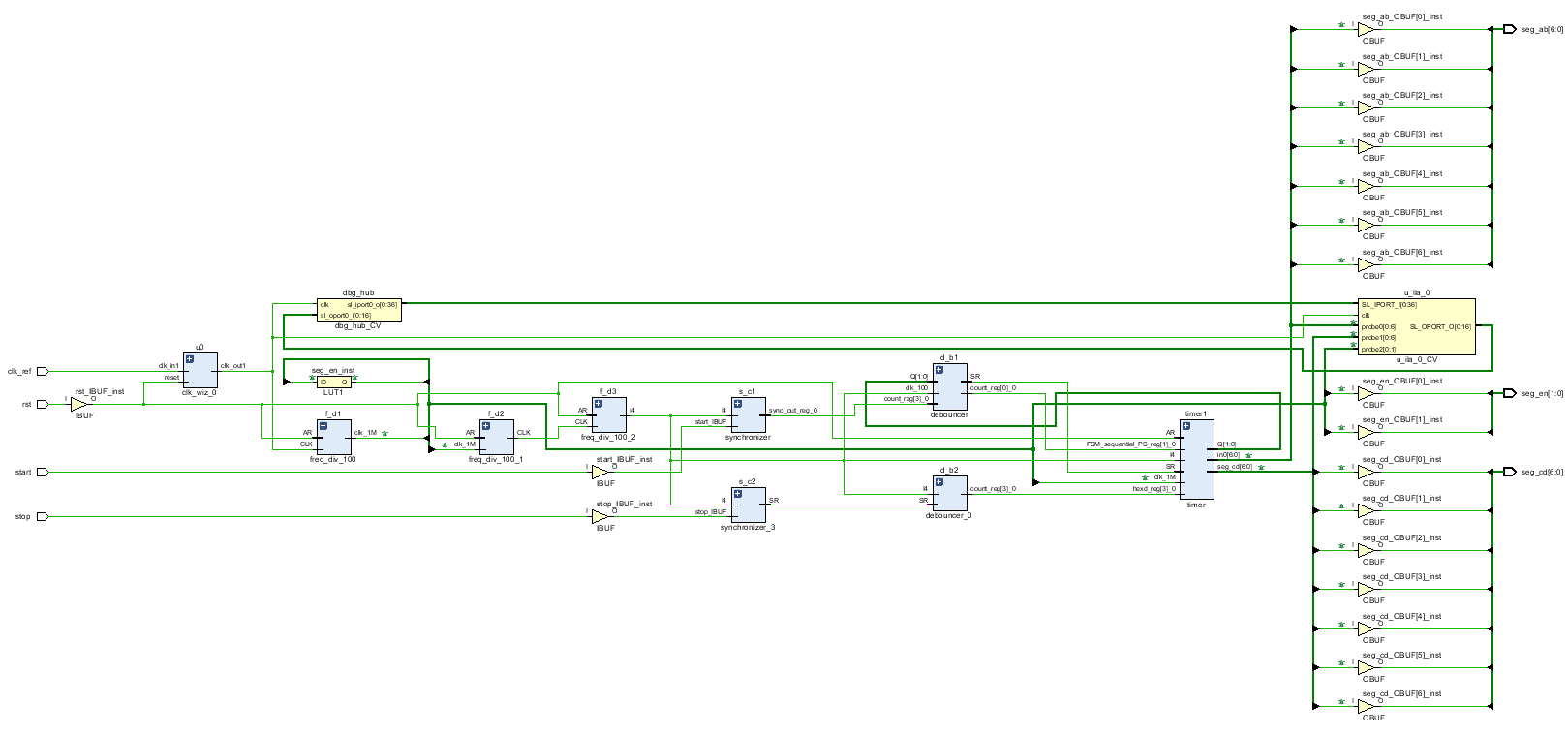
Zybo-Z7-Master.xdc





**Synthesis Report**



****

**-Debug**

**Test Bench Code**

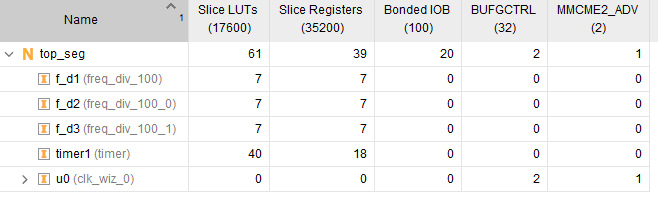
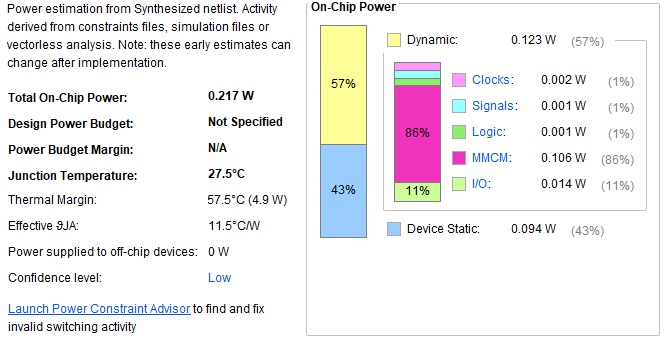


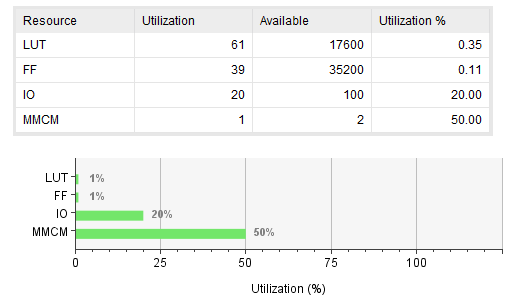
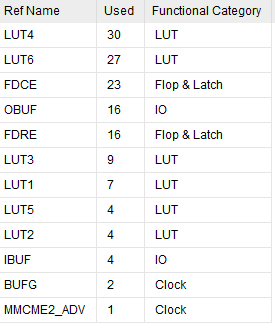
**Simulation Result**

-debug를 제외한 결과

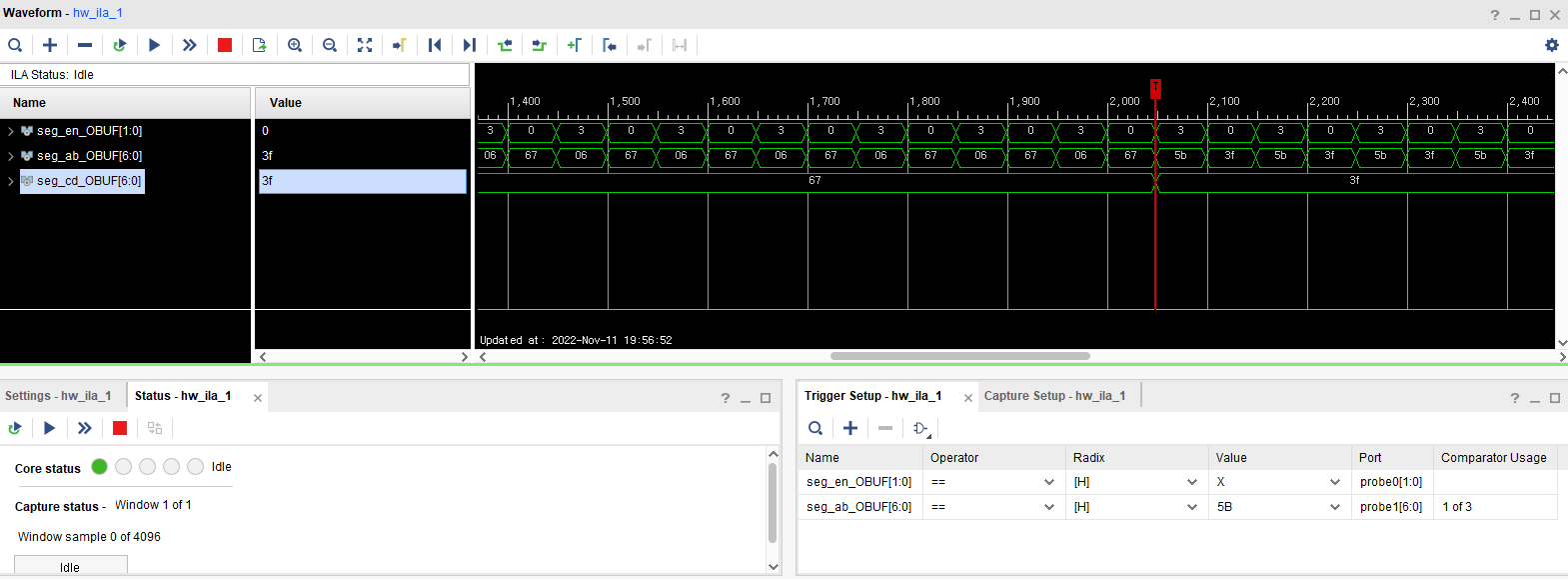
텍스트, 블라인드, 스크린샷, 옅은이(가) 표시된 사진

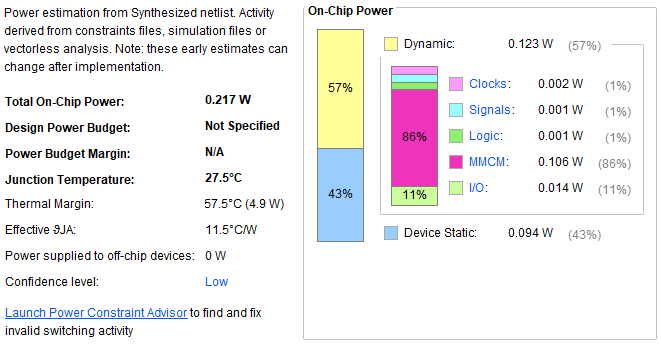
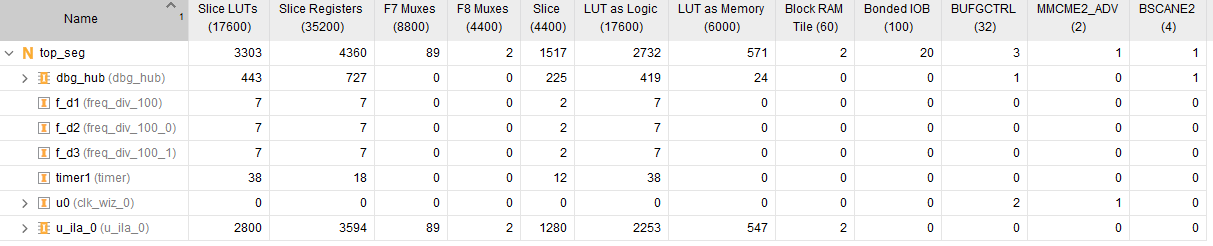
자동 생성된 설명

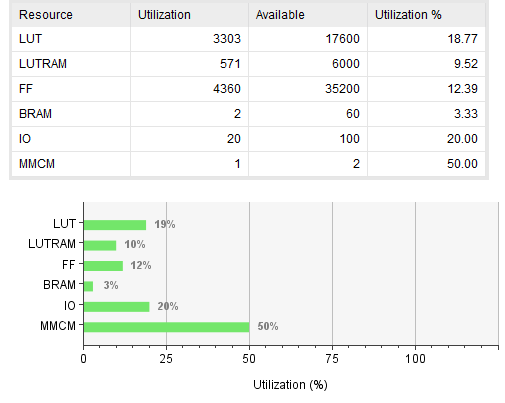
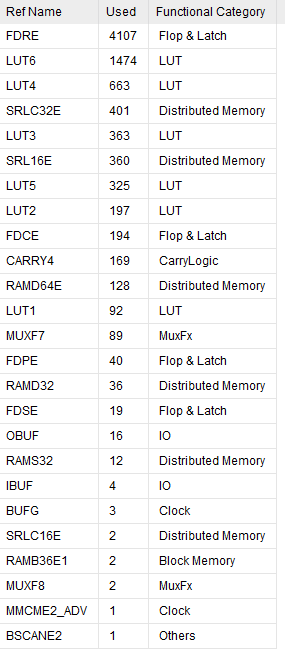


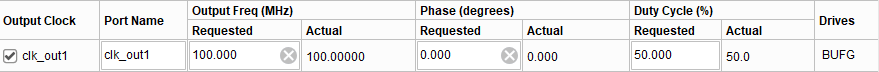


-debug 포함 결과



****







|  |
| --- |
| **Discussion** |
| . - Verilog Coding을 시작하기 전 작성한 Block Diagram      - 작성한 Verilog Module에 대한 설명  주요동작은 timer 모듈에서 진행하였으며 코드는 3개의 state로 나누어 구성하였다.  S0 : 동작이 시작하기 전으로 start 신호가 들어오기 전이다.  만약 rst가 들어오거나 rst와 start가 모두 들어오지 않을 때 S0의 상태를 유지한다.  rst신호가 들어오지 않으면서 start신호가 들어오면 S1으로 넘어가고 count를 1로 변경하여 시간이 증가할 수 있도록 카운팅을 진행한다.  S1 : 수를 증가시키는 상태이다.  카운팅을 진행하고 있는 state로 rst가 들어오면 S0로 돌아가고 segment에 입력될 숫자를 0으로 초기화한다.  Rst가 들어오지 않으면서 Reg\_stop이 들어오면 S2로 넘어간다.  이외의 경우에는 S1을 유지한다.  S2 : S2는 수가 증가하지 않는 상태이다.  Rst를 넣으면 Reg\_rst를 출력하며 S0로 초기화된다.  Rst, reg\_stop이 들어오지 않으면서 start가 들어오면 S1으로 넘어가고 count가 1이 되어 수를 다시 셀 수 있도록 하였다.  이 상태에서 변화하는 값을 네 비트씩 쪼개서 hex 모듈에 넣었다.  hex모듈에서는 이 비트를 16비트로 보았을 때 segment에 입력되는 값으로 환산한다.  이때 주기는 clk\_wiz와 freq\_div\_100모듈을 통하여 100Hz로 맞췄다.  이를 이용하여 최소 0.01단위로 시간을 변경하였고 이때 timer 모듈에서 자리 수 변경이 필요한지 확인하여 자리수 변경을 진행하였다.  - 작성한 Code가 어떤 동작을 하는지? 왜 그런 동작을 하는지?  전체적인 코드는 top\_seg 에서 module을 implementation으로 불러와서 구현하였다.  Top\_seg 모듈에서 clk\_wiz\_0, freq\_div\_100, timer, hex 모듈을 implementatio하였고 assign을 통해 time\_data를 분해하고 seg\_en에 11또는 00을 넣고 seg\_ab에는 sega와 b중 선택해서 넣었다.  Sega는 십의 자리 이고 segb는 일의 자리이다.  Seg\_cd는 segc와 segd중에 선택해서 값을 넣었다.  Segc는 소수점 아래 첫째 자리이고 segd는 소수점 아래 둘째 자리이다.  이 값들은 1MHz마다 반복되어 눈으로 보기에는 항상 동작하는 것으로 보이게 된다.  작성한 모듈은 앞선 실험에서 사용하였던 freq\_div\_100, debouncer, synchronizer, clk\_wiz\_0모듈을을 재활용하여 구성하였다.  나머지 hex 모듈과 timer는 이번 실습에서 처음 설계하였다.  먼저 hex모듈은 16진수의 값을 segment에 들어가야하는 값으로 변경하여 주는 모듈이다.  case문을 이용하여 값을 변경해주는 역할을하였다.  Timer 모듈은 앞선 문항에서 작성하였던 state에 따라 다른 동작을 진행하고 count가 1인 경우 시간을 증가시킨다.  이때 9가되면 다음 자리수를 1 더해주는 방식으로 구성하였다.  9가 될 때 다음 자리 수에 1을 더하는 이유는 0부터 시작하기 때문에 9까지 9번의 상승이 있어야 하고 다음 클럭에는 10번째 상승이 되므로 9일 때를 기준으로 값을 변경하도록 설정하면 된다.  Zybo\_Z7\_Master.xdc에 연결한 핀은 아래 datasheet를 참조하여 설정하였다.    Seg\_debeg.xdc파일에 나타난 코드는 디버그 동작을 통해서 작성된 코드이다.  - 작성한 Test Bench Code가 어떤 동작을 하는지?  Rst를 처음에 1로 설정한 후 rst를0으로 만들고 start를 1로 설정하여 값이 변화하는지 확인하였다.  HW debuging은 seg\_ab=7’h06일 때 trigger하는 방식으로 설정하였다.  - Simulation 파형이 왜 그렇게 나온것인지?  Simulation 파형은 입력이 바로 들어오지 않기 때문에 일정 시간 이후 time\_data 값이 변경되는지 확인하였다.  Clk\_100이 상승 edge가 되었을 때 값이 1 상승하는 것을 확인할 수 있었다.  HW debuging은 seg\_ab=7’h5B로 trigger하였다.  즉 소수점 위자리 중 2가 되는 부근을 debug하도록 trigger하였다.  시간을 진행하면서 19.99->20.00으로 변하는 지점을 trigger하기 위하여 start 신호를 인가하여 먼저 시간이 10여초 지난 후 trigger를 시작하였다.        결과적으로 trigger의 왼쪽에는 seg\_en이 3일 때 seg\_ab=7’h06, seg\_cd=7’h67로 19를 나타냈다.  Seg\_en이 0일 때 seg\_ab=7’h67, seg\_cd=7’h67로 99를 나타내었다.  이를 표로 나타내면 아래와 같다.   |  |  |  | | --- | --- | --- | | Seg\_en | 3 | 0 | | seg\_ab | 1 | 9 | | seg\_cd | 9 | 9 |   즉 19.99초가 원하는 대로 관측되었다.        Trigger의 오른쪽에는 seg\_en이 3일 때 seg\_ab=7’h5b, seg\_cd=7’h3f로 20을 나타냈다.  Seg\_en이 0일 때 seg\_ab=7’h3f, seg\_cd=7’h3f로00을 나타내었다.  이를 표로 나타내면 아래와 같다.   |  |  |  | | --- | --- | --- | | Seg\_en | 3 | 0 | | seg\_ab | 2 | 0 | | seg\_cd | 0 | 0 |   즉 20.00초로 원하는 대로 관측되었다.  - 작성한 Code가 잘 동작하지 않는다면 어디서 문제가 발생한 것인지?  정상적으로 동작하였다.  - 오류를 해결했다면, 어떤 부분에서 해결을 했고, 왜 오류가 발생하였는지? |

정상적으로 동작하였다.