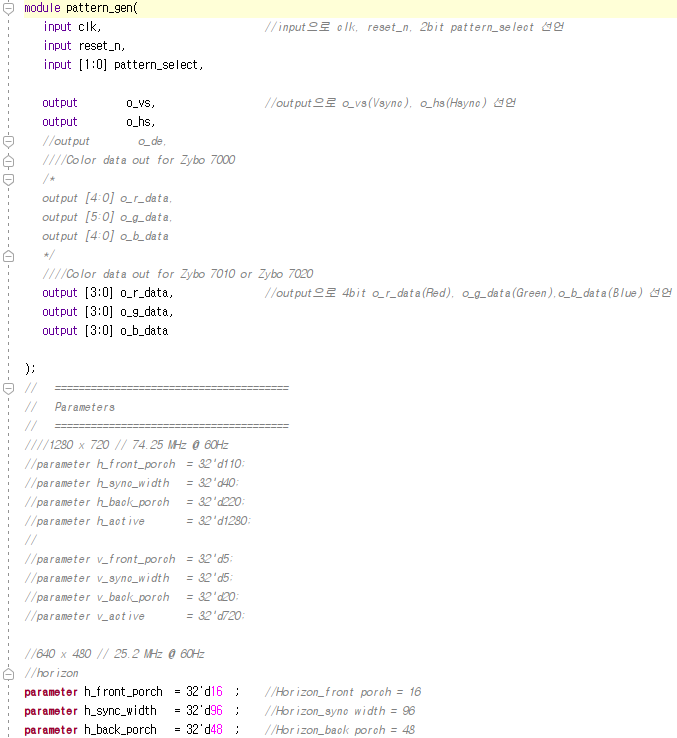
**Lab1.**

# **Homework1. video pattern generator**

**Verilog Code / 주석**

Pattern\_gen



텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트, 테이블이(가) 표시된 사진

자동 생성된 설명

Top\_mod

텍스트이(가) 표시된 사진

자동 생성된 설명

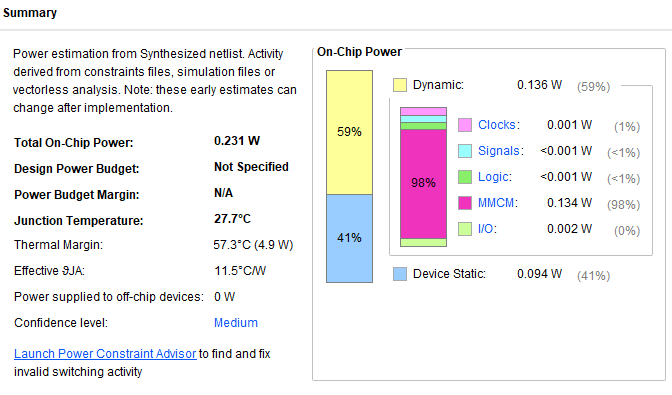
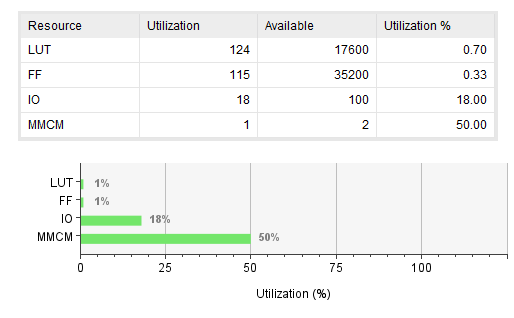
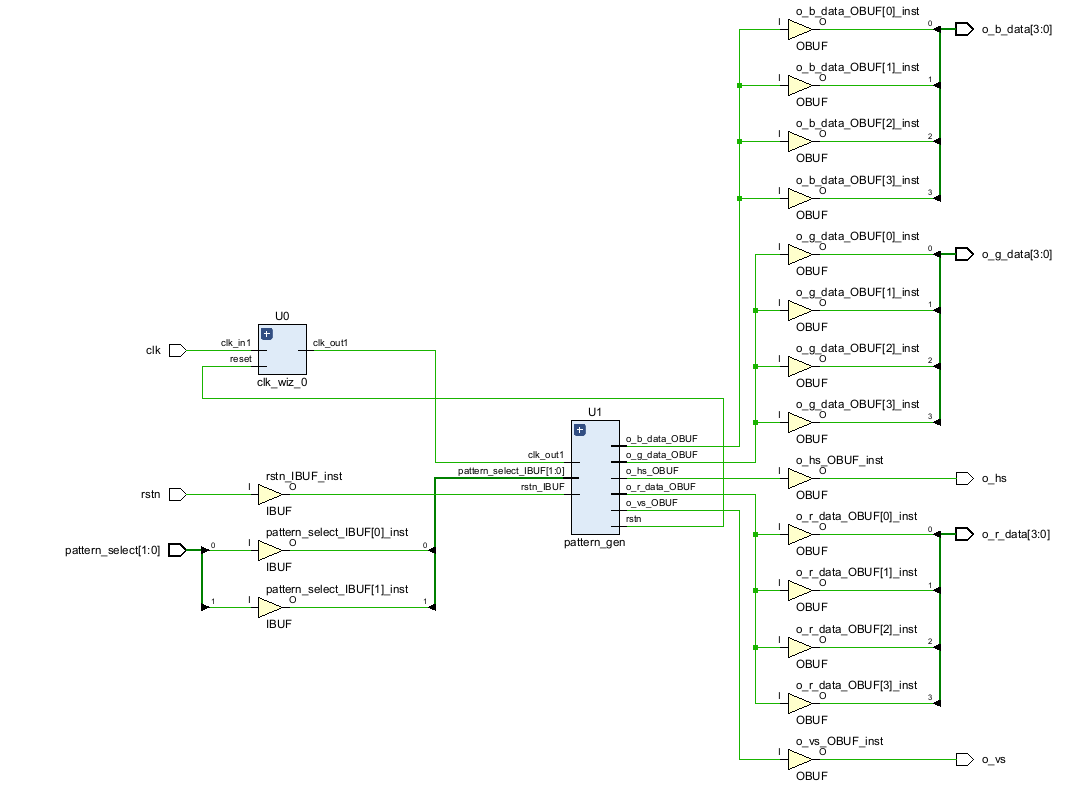
XDC

텍스트이(가) 표시된 사진

자동 생성된 설명테이블이(가) 표시된 사진

자동 생성된 설명

**Synthesis Report**

테이블이(가) 표시된 사진

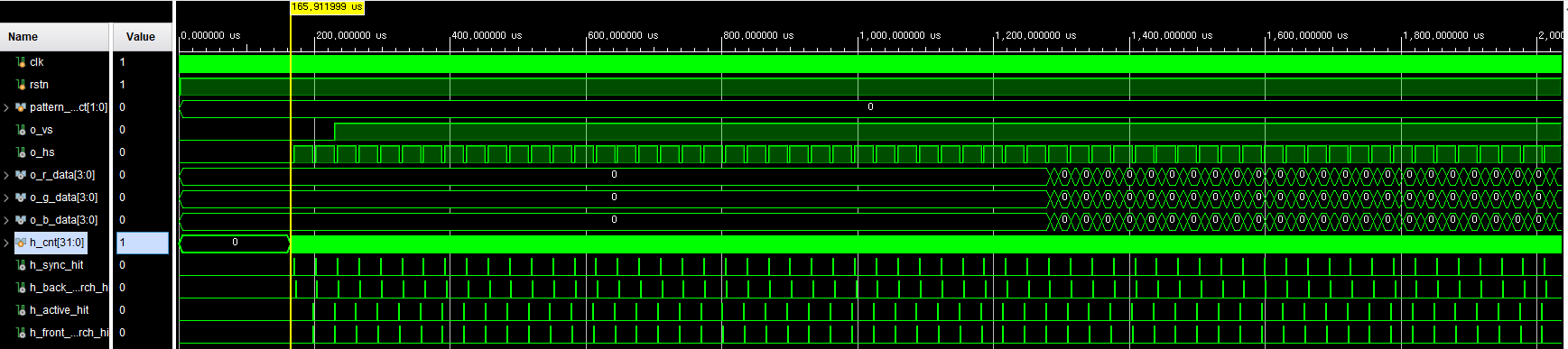
자동 생성된 설명

**Test Bench Code**

텍스트이(가) 표시된 사진

자동 생성된 설명

**Simulation Result**



|  |
| --- |
| **Discussion** |
| . - Verilog Coding을 시작하기 전 작성한 Block Diagram    - 작성한 Verilog Module에 대한 설명  Clock Generator로 FPGA 내부 클럭 125MHz를 25.2MHz로 바꾸어 Pattern Generator로 입력한다. 실험에서 사용하는 VESA(Video Electronics Standards Association) 표준 640 x 480을 60Hz로 나타내기 위해 800 \* 525 \* 60의 식을 통해 25.2MHz의 주파수로 신호를 보내기 때문이다.  2bit의 Pattern\_select를 입력하여 그에 맞는 pattern을 출력하기 위해 pattern generator를 통해 pattern을 만들어 RGB에서 각각에 해당하는 신호와 vsync, hsync을 출력한다. 이신호를 FPGA 보드에서 VGA 모듈을 통해 모니터에 연결하여 pattern을 출력한다.  - 작성한 Code가 어떤 동작을 하는지? 왜 그런 동작을 하는지?    먼저 parameter를 통해 왼쪽의 표와 같이 VESA 표준 명칭과 각각 해당하는 값을 맵핑 하였다. 또한 init\_cnt\_top을 생성하여 여유값을 만들었다.  always문을 통해 초기화가 된 이후 init\_cnt가 init\_cnt\_top까지 증가해야 하므로 약간의 시간을 delay 시키도록 하였다.  Horizon Count  화면이 출력되는 과정은 가로 한 줄을 왼쪽부터 오른쪽까지 차례대로 출력하고 한 줄의 출력이 완성되면 다음줄의 왼쪽에서부터 오른쪽의 순서로 화면을 출력하는 동작을 한다. 따라서 매 줄마다 Horizon count는 0으로 초기화를 해야 한다. Init\_cnt이Init\_cnt\_top일 때(reset을 하지 않았을 경우) h\_cnt를 1씩 증가하여 왼쪽에서 오른쪽으로 이동하게 된다.  Horizon hit point  assign문을 통해 sync, back porch, active, front porch의 가장 마지막 부분에서 각각의 hit 가 1을 출력하도록 설계하였다.  Vertical Count  Vertical Count는 Horizon Count와는 다르게 매 줄마다 초기화를 하지 않고 모든 신호를 출력한 뒤 (가장 마지막 줄의 horizon front porch끝) 0으로 초기화 하여야 한다. 또한 한 줄의 출력을 끝낸 뒤 1을 증가시켜야 하기 때문에 한 줄의 가장 마지막 부분인 front porch의 마지막 부분에서 1을 더하여 줄 바꿈을 실행한다.  Vertical active line  Vertical 에서도 active 영역을 나타내며 v\_back\_porch다음부터 v\_back\_porch+v\_active(480)까지 1을 유지하고 이외의 경우에는 0을 출력한다.    Make VSYHC  위의 그림을 참조하여 line\_cnt = v\_sync\_width - 1 이고 h\_front\_porch\_hit = 1일 때, 즉 v\_Back\_Porch가 시작할 때부터 v\_front\_porch가 끝날 때까지 1을 출력하도록 설계하였다.  Make HSYHC  마찬가지로 위의 그림을 참조하여 매 줄마다 h\_Back\_Porch가 시작할 때부터 h\_front\_porch가 끝날 때까지 1을 출력하도록 설계하였다.  Make Data Enable Signal  active\_line일 때 h\_Back\_Porch가 끝날 때부터 h\_active 가 끝날 때까지 1을 출력하도록 설계하였다.  Assign문을 이용해 wire vs와 hs를 output o\_vs, o\_hs에 연결하였고 Data Enable Signal이 1인 경우 de\_cnt값을 1씩 상승시키고 0인 경우 0으로 초기화하였다.  Pattern 0 (White / Black)    h\_active영역의 1/2, 즉 좌반부에 RGB를 각각 출력하여 흰색을 만들었고, 우반부에 RGB를 모두 출력하지 않아 검은색을 만들었다.  Pattern 1  h\_active영역 중 0 ~ 1/2의 영역에는 R을 출력하고 0~1/4, 1/2~3/4의 영역에서는 G, 0~1/4, 3/4~1의 영역에서는 B를 출력하여 최종적으로 (White / Red / Green / Blue)가 출력되도록 만들었다.  Pattern 2  h\_active영역 중 0 ~ 1/5, 4/5~1의 영역에서는 R, 1/5~2/5, 4/5~1의 영역에서는 G, 2/5~3/5, 4/5~1의 영역에서는B를 출력하여 최종적으로 (Red / Green / Blue / Black / White)가 출력되도록 만들었다.  Pattern 3  Horizon의 성분은 de\_cnt, vertical의 성분은 line \_cnt를 이용하여 영역을 구분하였다. Line\_cnt는 sync pulse, back porch, active, front porch를 모두 표현하므로 v\_sync+back proch+(1/2)active를 이용하여 v\_active 영역을 반으로 나누었다. h\_active를 1/2로 나누어 총 4개의 면을 만들었고 각 면들에 알맞은 빛을 출력하도록 만들어서 주어진 pattern을 만들었다.  pattern\_select의 값을 비교하고 그에 맞는 RGB 신호를 각각 enable 시켜 패턴을 출력하도록 하였다. 또한 r\_data는 zynq 7010의 datasheet에 맞추어 각 15를 출력하도록 하였다.  Pattern generator 모듈을 보면 reset신호가 0인 경우 reset을 한다. 이는 clk\_wiz\_0와 반대로 reset이 동작한다. 따라서 clk\_wiz\_0에는 반전된 reset 신호를 입력하였다.  - 작성한 Test Bench Code가 어떤 동작을 하는지?  Clk를 4ns마다 반전시켜 125MHz의 클럭을 만들었고 rstn을 0-> 1로 반전시켜 pattern generator와 clk\_wiz\_0를 동작시키고 pattern\_select에 00을 입력하여 pattern 0에 해당하는 신호를 출력하도록 만들었다. 예상했던 동작을 한다면 RGB는 각각 0,15가 계속 하여 반복되며 나올 것이고 o\_vs와 o\_hs는 위에서 참조했던 그림 과 같이 출력될 것으로 예상할 수 있다.  - Simulation 파형이 왜 그렇게 나온것인지?  예상과 같이 125MHz의 클럭에서 Clk\_wiz\_0를 거쳐 25.2MHz의 클럭으로 변환하였고 reset이 1로 입력된 후 약간의 시간 후에 pattern generator가 동작하였다. 실제 모니터가 출력하는 active 영역에서 RGB는 각각 0과 f를 반복하여 출력해 모니터에 연결할 경우 좌반부는 흰색, 우반부는 검은색으로 구성된 pattern 0를 출력할 것이라 예상할 수 있다.  - 작성한 Code가 잘 동작하지 않는다면 어디서 문제가 발생한 것인지?  잘 동작하였습니다.  - 오류를 해결했다면, 어떤 부분에서 해결을 했고, 왜 오류가 발생하였는지?  오류 없이 동작하였습니다. |