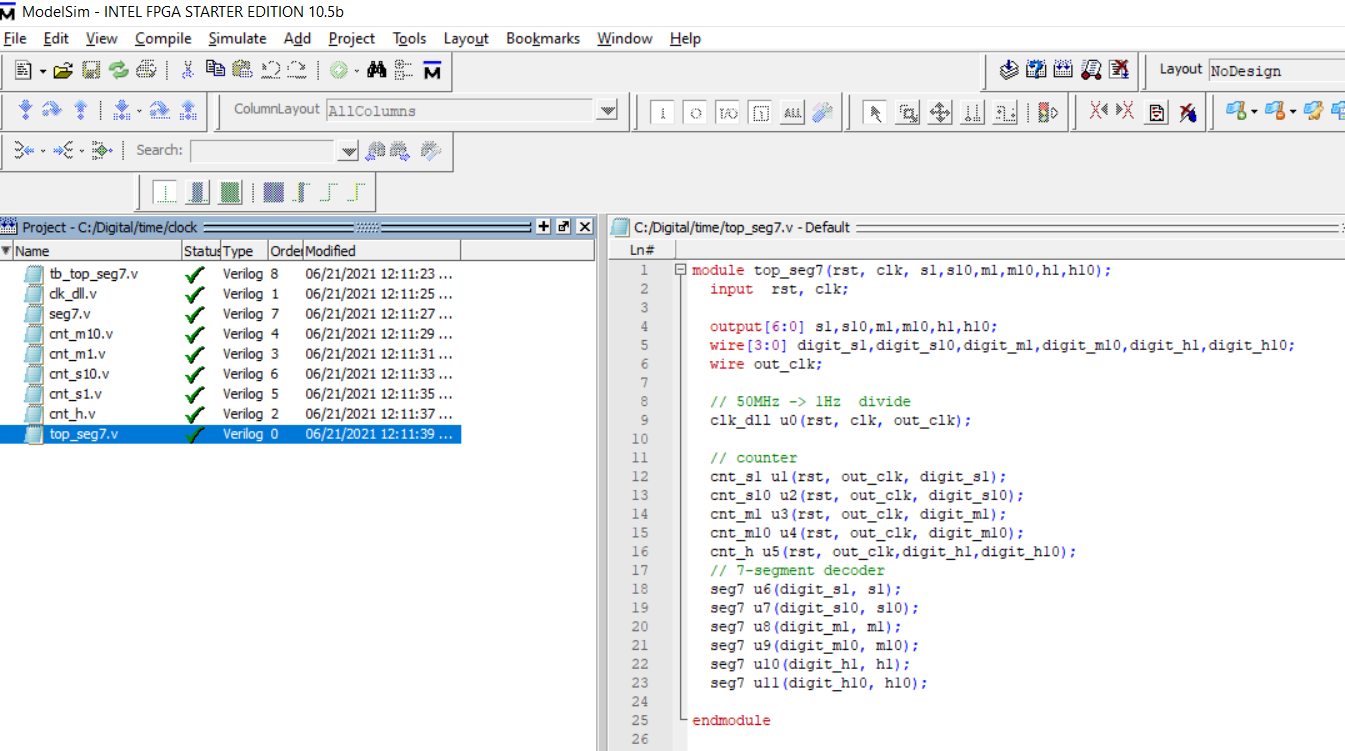
|  |  |
| --- | --- |
| 논리설계  Digital Clock 과제 제출 | |
| 설계보드 : DE10 보드 | |
| 팀 번호 : 22 | 번호 43  2016320267 이주명 |
| 번호 44  2017320122 김정규 |

1. **설계 코드 및 결과(코드 , 시뮬레이션 , 사진 등)**
   1. 코드

아래와 같이 modelsim으로 먼저 설계했습니다.



|  |
| --- |
| cllk.dll.v  50Mhz 기준 50M분주를 통해 1hz로 만들어주는 분주기 입니다.  다만 원활하게 시뮬레이션 결과를 보기 위해, 시뮬레이션시에는  분주를 //if(cnt\_clk == 24) 를 if(cnt\_clk == 24999999) 대신 사용하였습니다. |
| module clk\_dll(rst, clk, out\_clk);  input clk, rst;  output out\_clk;  reg out\_clk;  reg [24:0]cnt\_clk;  always@(posedge clk or negedge rst)  begin  if(!rst)  begin  out\_clk = 0;  cnt\_clk = 0;  end    else  begin  cnt\_clk <= cnt\_clk+1;    if(cnt\_clk == 24999999) // 50m 분주  //if(cnt\_clk == 24)  cnt\_clk <=0;  else if(cnt\_clk==0)  out\_clk = !out\_clk;  end    end  endmodule |

|  |
| --- |
| cnt\_s1.v // 1초 단위의 초에 대한 카운터 입니다. (0 ~ 9)까지 카운팅하며  분주기를 통해 나온 클럭을 기준으로 count 하게 됩니다. |
| module cnt\_s1(rst, clk, q);  input rst, clk;  output [3:0] q;  reg [3:0] tmp\_q, q;  always @(posedge clk or negedge rst)  begin  if(!rst)  begin  q <= 4'b0000;  tmp\_q <= 4'b0000;  end  else  begin  if(tmp\_q+1 == 10) // 10 -> 0 //10이 되었을떄 다시 0으로  begin  tmp\_q <= 0;  end  else  begin  tmp\_q <= tmp\_q + 1;  end  q <= tmp\_q;  end  end  endmodule |

|  |
| --- |
| cnt\_s10.v // 10초 단위의 초에 대한 카운터 입니다. (0 ~ 5)까지 카운팅하며  분주기를 통해 나온 클럭을 10분주하여 count 하게 됩니다. |
| //\_s10 , 10 division counter  module cnt\_s10(rst, clk, q);  input rst, clk;  output [3:0] q;  reg [3:0] tmp\_q, q;  reg clk1;  reg[2:0] tmp1;  //10 division  always@(posedge clk or negedge rst) begin  if(!rst)  begin  clk1 <= 0;  tmp1 <= 0;  end  else  begin  tmp1 <= tmp1+1;  if(tmp1 == 4) //10분주  tmp1 <=0;  else if(tmp1==0)  clk1 <= ~clk1;  end  end    always @(posedge clk1 or negedge rst)  begin  if(!rst)  begin  q <= 4'b0000;  tmp\_q <= 4'b0000;  end  else  begin  if(tmp\_q+1== 6) /// 6 ->0 6을 0으로 변환  begin  tmp\_q <= 0;  end  else  begin  tmp\_q <= tmp\_q + 1;  end  q <= tmp\_q;  end  end  endmodule |

|  |
| --- |
| cnt\_m1.v // 1분 단위의 카운터 입니다. (0 ~ 9)까지 카운팅하며  분주기를 통해 나온 클럭을 60분주하여 count 하게 됩니다. |
| //\_m1 , 60 division counter  module cnt\_m1(rst, clk, q);  input rst, clk;  output [3:0] q;  reg [3:0] tmp\_q, q;  reg clk1;  reg[4:0] tmp1;  //60 division  always@(posedge clk or negedge rst) begin  if(!rst)  begin  clk1 <= 0;  tmp1 <= 0;  end  else  begin  tmp1 <= tmp1+1;  if(tmp1 == 29) //60 division 60분주  tmp1 <=0;  else if(tmp1==0)  clk1 <= ~clk1;  end  end    always @(posedge clk1 or negedge rst)  begin  if(!rst)  begin  q <= 4'b0000;  tmp\_q <= 4'b0000;  end  else  begin  if(tmp\_q+1== 10) /// 10 ->0 10을 0으로 변경  begin  tmp\_q <= 0;  end  else  begin  tmp\_q <= tmp\_q + 1;  end  q <= tmp\_q;  end  end  endmodule |

|  |
| --- |
| cnt\_m10.v // 10분 단위의 카운터 입니다. (0 ~ 5)까지 카운팅하며  분주기를 통해 나온 클럭을 600분주하여 count 하게 됩니다. |
| //\_m10 , 600 division counter  module cnt\_m10(rst, clk, q);  input rst, clk;  output [3:0] q;  reg [3:0] tmp\_q, q;  reg clk1;  reg[8:0] tmp1;  //600 division  always@(posedge clk or negedge rst) begin  if(!rst)  begin  clk1 <= 0;  tmp1 <= 0;  end  else  begin  tmp1 <= tmp1+1;  if(tmp1 == 299) //600 division , 600분주  tmp1 <=0;  else if(tmp1==0)  clk1 <= ~clk1;  end  end    always @(posedge clk1 or negedge rst)  begin  if(!rst)  begin  q <= 4'b0000;  tmp\_q <= 4'b0000;  end  else  begin  if(tmp\_q+1== 6) /// 6 ->0 6을 0으로 변경  begin  tmp\_q <= 0;  end  else  begin  tmp\_q <= tmp\_q + 1;  end  q <= tmp\_q;  end  end  endmodule |

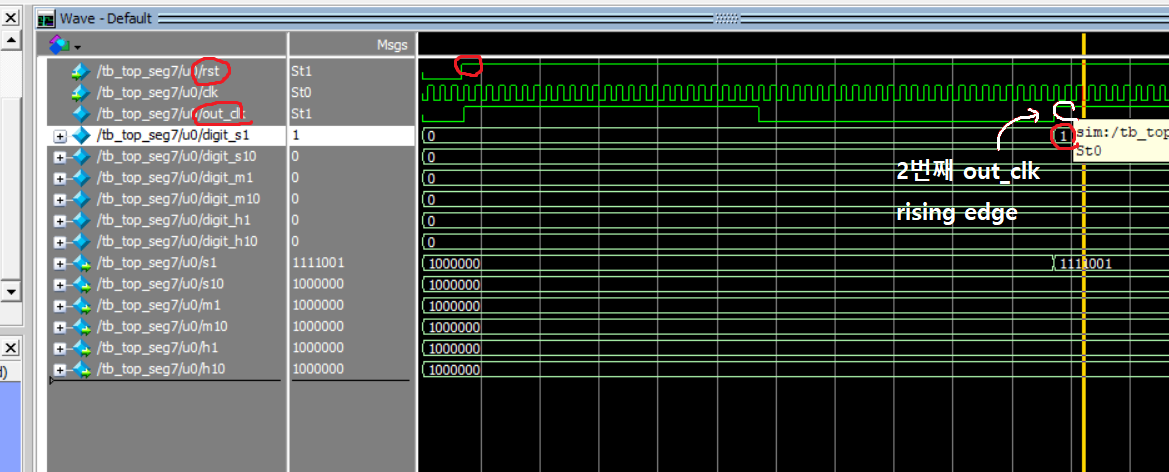
|  |
| --- |
| cnt\_h.v // 1시간 단위의 카운터 입니다. (0 ~ 23)까지 카운팅하며  분주기를 통해 나온 클럭을 3600분주하여 count 하게 됩니다.  이 떄 십의자리는 q\_h10 일의 자리는 q\_h1로 각각 output이 나오게 됩니다. |
| //h1 , h10, 3600 division counter  module cnt\_h(rst, clk,q\_h1,q\_h10);  input rst, clk;  output [3:0] q\_h1,q\_h10;  reg [4:0] tmp\_q;  reg [3:0] q\_h1,q\_h10;  reg clk1;  reg[10:0] tmp1;  //3600 division  always@(posedge clk or negedge rst) begin  if(!rst)  begin  clk1 <= 0;  tmp1 <= 0;  end  else  begin  tmp1 <= tmp1+1;  if(tmp1 == 1799) //3600 division , 3600 분주  tmp1 <=0;  else if(tmp1==0)  clk1 <= ~clk1;  end  end    always @(posedge clk1 or negedge rst)  begin  if(!rst)  begin  q\_h1 <= 4'b0000; // q\_h1 0으로 초기화  q\_h10 <= 4'b0000; // q\_h10 0으로 초기화  tmp\_q <= 4'b0000;  end  else  begin  if(tmp\_q+1== 24) /// 24 ->0 24를 0으로 변경  begin  tmp\_q <= 0;  end  else  begin  tmp\_q <= tmp\_q + 1;  end  if(tmp\_q >=0 && tmp\_q <=9) //0에서 9사이 일 때  begin  q\_h10 <= 0; // q\_h10은 0  q\_h1 <= tmp\_q; //q\_h1 은 count한 값 그대로  end  else if(tmp\_q>=10 && tmp\_q <=19) //10에서 19사이 일 때  begin  q\_h10 <= 1; //q\_h10은 1  q\_h1 <= tmp\_q - 10; // q\_h1 은 10을 뺀 count의 일의 자리  end  else if(tmp\_q >=20 && tmp\_q <= 23) //20에서 23사이 일떄  begin //24이상일 경우 이미 0으로 초기화 되어있어 고려 x  q\_h10 <=2; //q\_h10 은 2  q\_h1 <= tmp\_q-20; //q\_h1은 20을 뺀 count의 일의 자리  end  end  end  endmodule |

|  |
| --- |
| seg7.v // 각각의 카운터의 값의 결과로 나오게 된 숫자(digit)을  fpga의 7 segment의 각 신호로 변환 하는 과정입니다.  0,1,2,3,4,5,6,7,8,9인 경우 각각 모양에 맞도록 seg를 출력하게 됩니다. |
| module seg7(digit, seg);  input [3:0] digit;  output [6:0] seg;  reg [6:0] seg;  always@(digit)  begin  case(digit)  4'b0000 : seg = 7'b1000000;  4'b0001 : seg = 7'b1111001;  4'b0010 : seg = 7'b0100100;  4'b0011 : seg = 7'b0110000;  4'b0100 : seg = 7'b0011001;  4'b0101 : seg = 7'b0010010;  4'b0110 : seg = 7'b0000010;  4'b0111 : seg = 7'b1111000;  4'b1000 : seg = 7'b0000000;  4'b1001 : seg = 7'b0010000;  endcase  end    endmodule |

|  |
| --- |
| top\_seg7.v //  탑 모듈로서 , 분주의 과정 , 카운트 과정 이를 seg\_7로 변화하는 과정입니다.  왼쪽에서 오른쪽으로 h10,h1,m10,m1,s10,s1 의 순으로 실제 fpga 의 동작에 영향을 주게 됩니다. |
| module top\_seg7(rst, clk, s1,s10,m1,m10,h1,h10);  input rst, clk;  output[6:0] s1,s10,m1,m10,h1,h10;  wire[3:0] digit\_s1,digit\_s10,digit\_m1,digit\_m10,digit\_h1,digit\_h10;  wire out\_clk;    // 50MHz -> 1Hz divide  clk\_dll u0(rst, clk, out\_clk);    // counter  cnt\_s1 u1(rst, out\_clk, digit\_s1);  cnt\_s10 u2(rst, out\_clk, digit\_s10);  cnt\_m1 u3(rst, out\_clk, digit\_m1);  cnt\_m10 u4(rst, out\_clk, digit\_m10);  cnt\_h u5(rst, out\_clk,digit\_h1,digit\_h10);  // 7-segment decoder  seg7 u6(digit\_s1, s1);  seg7 u7(digit\_s10, s10);  seg7 u8(digit\_m1, m1);  seg7 u9(digit\_m10, m10);  seg7 u10(digit\_h1, h1);  seg7 u11(digit\_h10, h10);    endmodule |

|  |
| --- |
| tb\_top\_seg7.v  //top 모듈 테스트 벤치 입니다.  실제 50M 분주를 하게 되면 시간이 오래 걸리므로 (clk\_dll.v 도 simulation 적용시에는 50분주를 사용하였습니다. )  Simulation 파형 캡쳐시 , 50분주를 사용하였습니다. |
| `timescale 1ns/1ns  module tb\_top\_seg7();  reg rst, clk;  wire [6:0] seg;  top\_seg7 u0(rst, clk, seg);  initial  begin  rst=0;  #33; rst=1;  end  initial  begin  clk=0;  forever #5 clk=~clk;  end  endmodule |

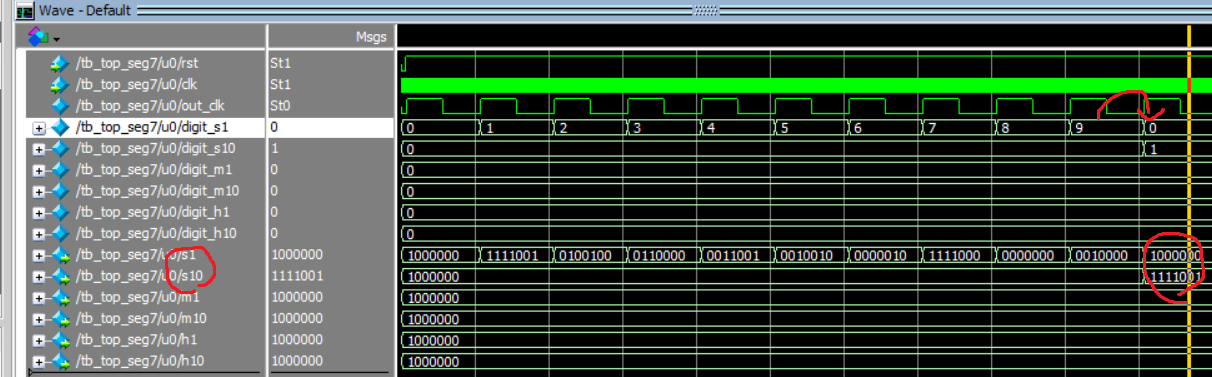
* 1. **시뮬레이션 파형 캡쳐**



rst rising edge 이후

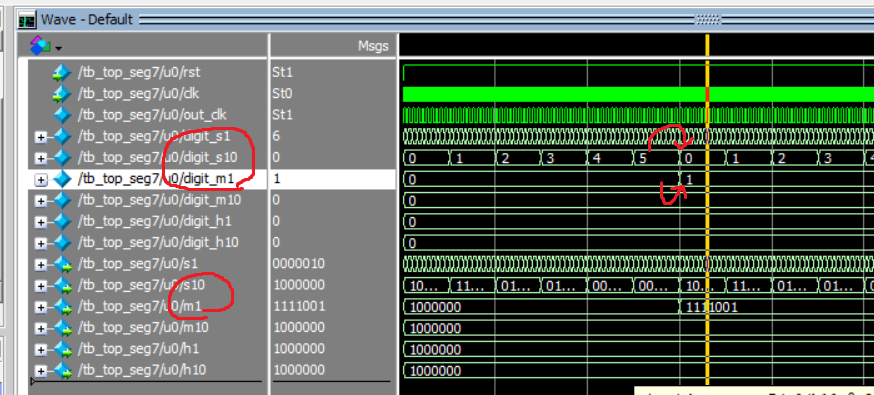
out\_clk 이 1번째 rising edge에는 digit\_s1 이 0

out\_clk이 2번째 rising edge 부터 digit\_s1 이 1이되며 count가 올라감을 확인했습니다.



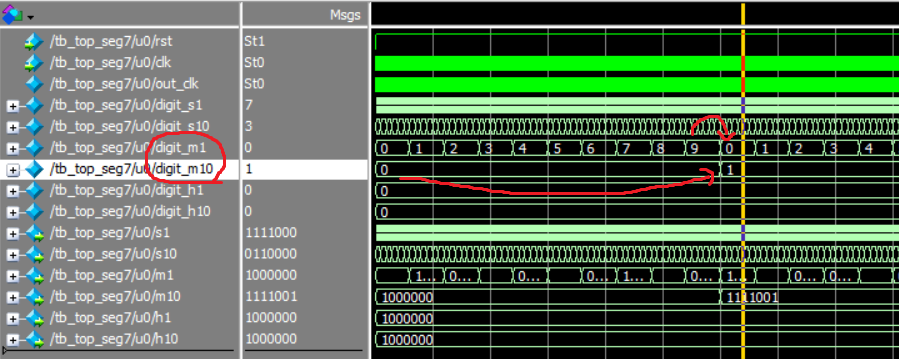
digit\_s1이 9->0으로 바뀌는 것을 볼 수 있었고 이 때 digit\_s10의 값이 1증가함을 확인할 수 있었습니다.

.



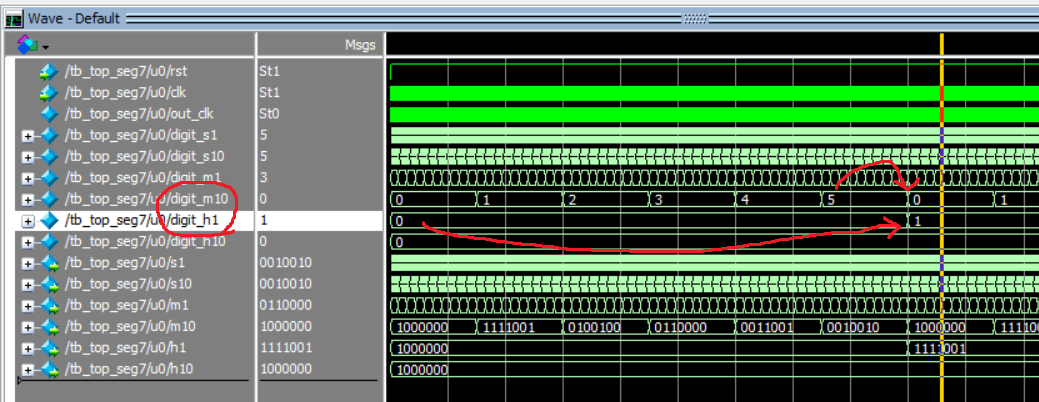
digit\_s10이 5->0으로 바뀌는 것을 볼 수 있었고 이떄 digit\_m1의 값이 1증가함을 확인할 수 있었습니다.

7\_segment의 경우 digit의 값에 따라 올바른 값으로 항상 나왔기 때문에 다음부터는 생략하도록 하겠습니다.

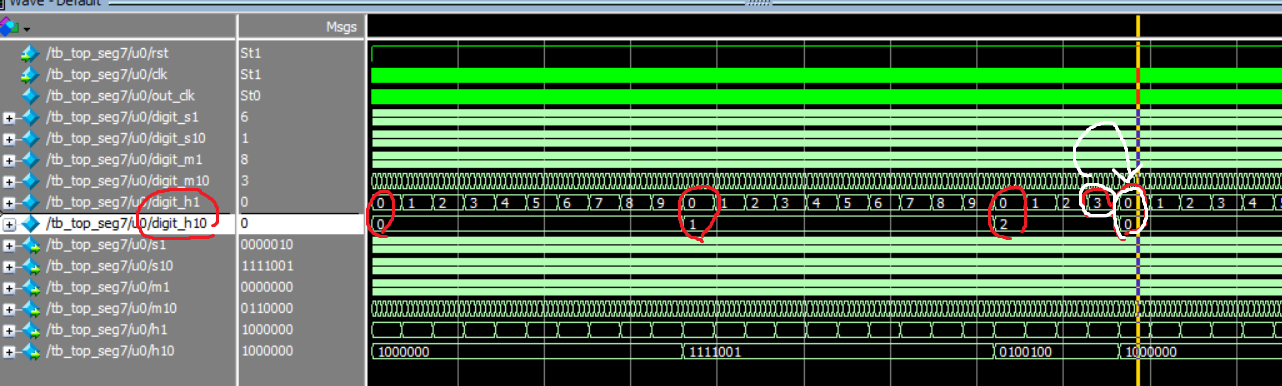


digit\_m1이 9->0으로 바뀌는 것을 볼 수 있었고 이 때 digit\_m10의 값이 1증가함을 확인할 수 있었습니다.

.



digit\_m10이 5->0으로 바뀌는 것을 볼 수 있었고 이 때 digit\_h1의 값이 1증가함을 확인할 수 있었습니다.



h10 (10의자리), h1(1의 자리) 의 값이 0 ~ 23까지 증가 후 다시 0으로 돌아가는 것을 확인 할 수 있었습니다.

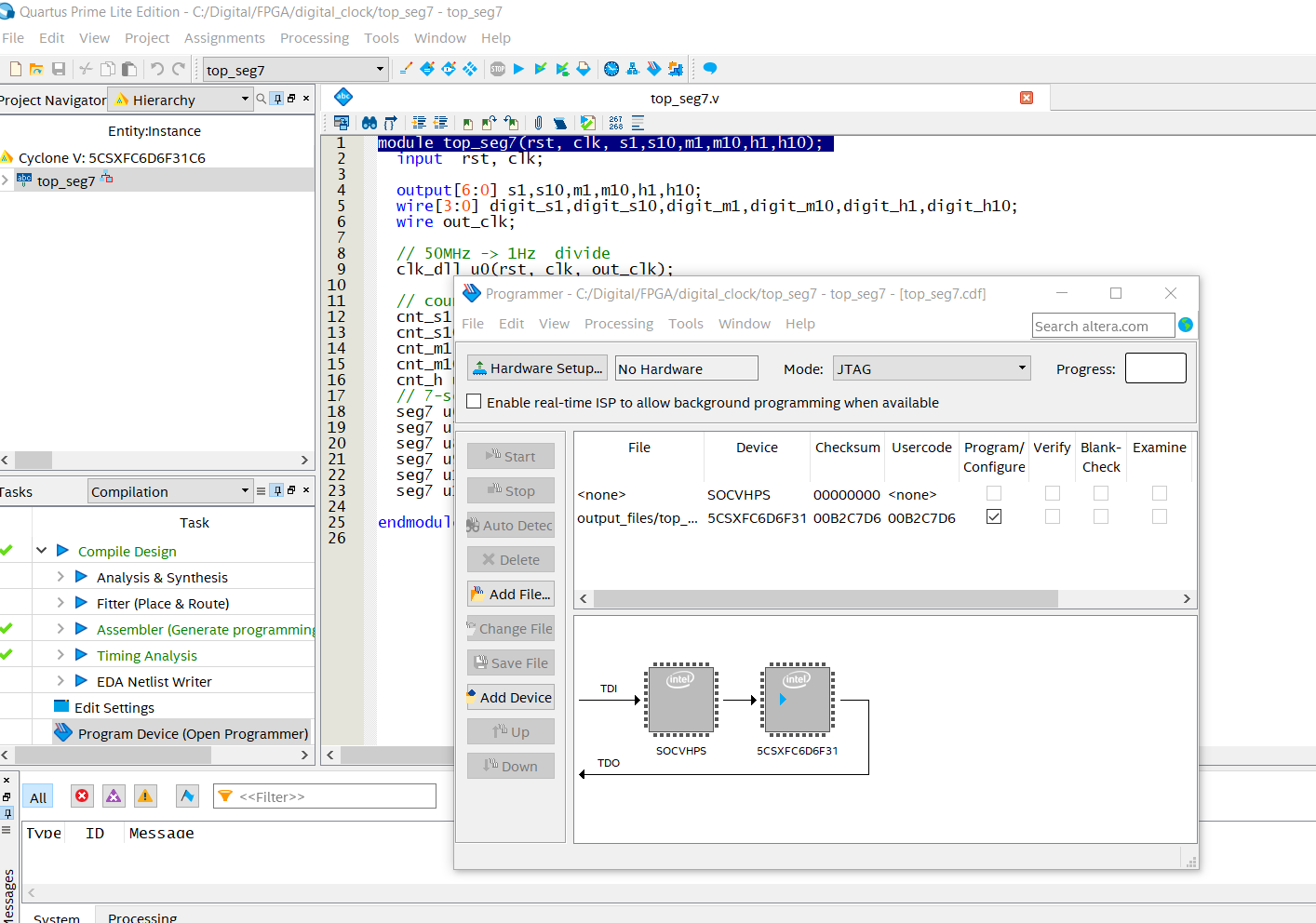
* 1. **Fpga 이용**

Modelsim 소스에서 test bench 코드를 제외하여 fpga에 소스코드를 넣었고

Clk\_Dll의 분주를 50M로 맞춘 후 , 파일을 만들어 fpga 로 테스트 해보았을 때

1초에 1씩 카운터가 오름을 확인 할 수 있었고

효과적으로 숫자가 변하는 것을 보기 위해 각 분주를 조절해가며 영상을 촬영하였습니다.

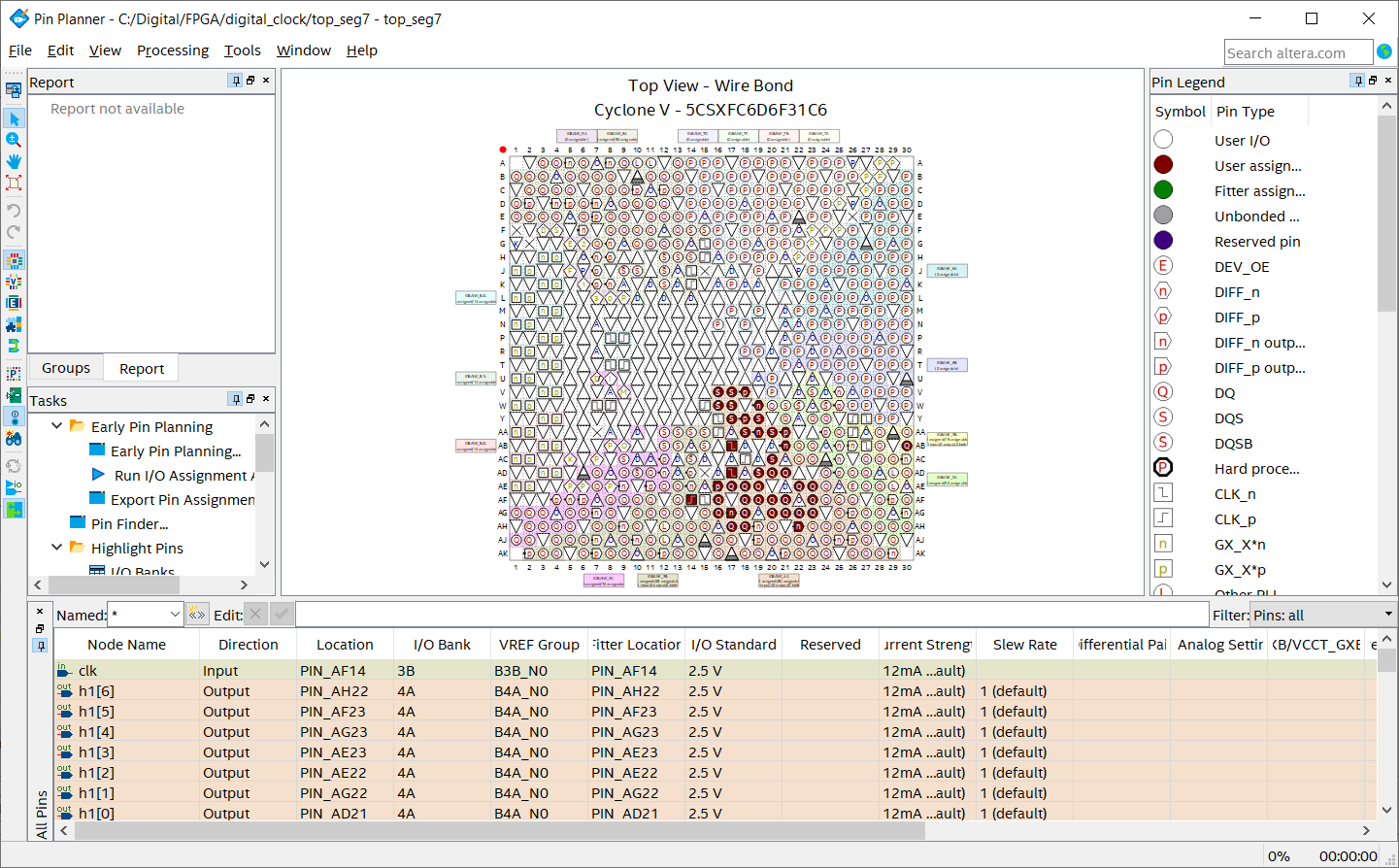


(컴파일을 마친후 , 보드 연결전 스크린 샷입니다.)

핀 설정과 관련된 부분 은 다음과 같은 내용을 파일에 직접 넣어서 저장하였습니다.

Fpga 핀 연결은 다음과 같이 의도대로 설정하였습니다.

|  |
| --- |
| set\_location\_assignment PIN\_AF14 -to clk  set\_location\_assignment PIN\_AB30 -to rst  set\_location\_assignment PIN\_W17 -to s1[0]  set\_location\_assignment PIN\_V18 -to s1[1]  set\_location\_assignment PIN\_AG17 -to s1[2]  set\_location\_assignment PIN\_AG16 -to s1[3]  set\_location\_assignment PIN\_AH17 -to s1[4]  set\_location\_assignment PIN\_AG18 -to s1[5]  set\_location\_assignment PIN\_AH18 -to s1[6]  set\_location\_assignment PIN\_AF16 -to s10[0]  set\_location\_assignment PIN\_V16 -to s10[1]  set\_location\_assignment PIN\_AE16 -to s10[2]  set\_location\_assignment PIN\_AD17 -to s10[3]  set\_location\_assignment PIN\_AE18 -to s10[4]  set\_location\_assignment PIN\_AE17 -to s10[5]  set\_location\_assignment PIN\_V17 -to s10[6]  set\_location\_assignment PIN\_AA21 -to m1[0]  set\_location\_assignment PIN\_AB17 -to m1[1]  set\_location\_assignment PIN\_AA18 -to m1[2]  set\_location\_assignment PIN\_Y17 -to m1[3]  set\_location\_assignment PIN\_Y18 -to m1[4]  set\_location\_assignment PIN\_AF18 -to m1[5]  set\_location\_assignment PIN\_W16 -to m1[6]  set\_location\_assignment PIN\_Y19 -to m10[0]  set\_location\_assignment PIN\_W19 -to m10[1]  set\_location\_assignment PIN\_AD19 -to m10[2]  set\_location\_assignment PIN\_AA20 -to m10[3]  set\_location\_assignment PIN\_AC20 -to m10[4]  set\_location\_assignment PIN\_AA19 -to m10[5]  set\_location\_assignment PIN\_AD20 -to m10[6]  set\_location\_assignment PIN\_AD21 -to h1[0]  set\_location\_assignment PIN\_AG22 -to h1[1]  set\_location\_assignment PIN\_AE22 -to h1[2]  set\_location\_assignment PIN\_AE23 -to h1[3]  set\_location\_assignment PIN\_AG23 -to h1[4]  set\_location\_assignment PIN\_AF23 -to h1[5]  set\_location\_assignment PIN\_AH22 -to h1[6]  set\_location\_assignment PIN\_AF21 -to h10[0]  set\_location\_assignment PIN\_AG21 -to h10[1]  set\_location\_assignment PIN\_AF20 -to h10[2]  set\_location\_assignment PIN\_AG20 -to h10[3]  set\_location\_assignment PIN\_AE19 -to h10[4]  set\_location\_assignment PIN\_AF19 -to h10[5]  set\_location\_assignment PIN\_AB21 -to h10[6] |



(pin 연결 스크린샷)

이후 보드 연결 후 run을 통해 파일을 다운 받고 실행과정을 fpga에서 살펴보았습니다.

아래와 같이 fpga에서 digital clock이 잘 작동함을 확인하였으며 관련된 영상은 mp4파일로 첨부했습니다 텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

1. **결론 및 검토**
   1. 목표대비 달성도  
      목표 : Digital Clock의 시 , 분 ,초 단위가 올바르게 동작하도록 설계하고 fpga로 검증해보기  
      달성도: 원하던 목표대로 100%을 달성하였습니다.
   2. Review

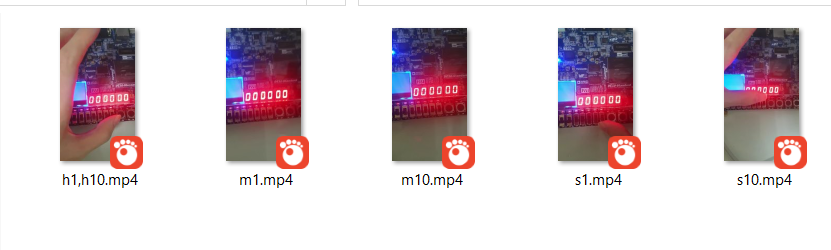
fpga 점검도중 마지막 h1,h10의 동작에 있어 0~23까지 표현되지 않고

0~15까지 나오는 현상이 있었습니다. 0~15 까지의 범위로 볼 때 4bit의 저장공간을 사용하여 이후 크기가 초과되어 다시 초기화 된다고 파악하였고

이후 관련된 변수를 5bit로 설정하여 문제를 해결하여 모두 정상 작동하게 되었습니다.

설계코드가 직접 fpga 보드로 작동함을 보면서 뿌듯하기도 하고 실제 동작함을 확인 할 수 있어서 좋았습니다.

1. DE10 보드 영상은 동영상 파일로 첨부했습니다.

  
h1,h10.mp4에서 시간 h1 , h10의 동작을 영상으로 동시에 나타내었습니다.

이후 m1,m10,s1,s10 동영상 파일에서 각각의 동작을 mp4 파일로 첨부했습니다.