Submission of Experiment 28
Digital Electronics with Altera
Part II: The Practical Part

Name: Zheng Sun
Group Number: 112
Lab date: 17, Nov 2017

1. The compilation result of the design in Section 2.3 [3 Marks]

Screenshot:

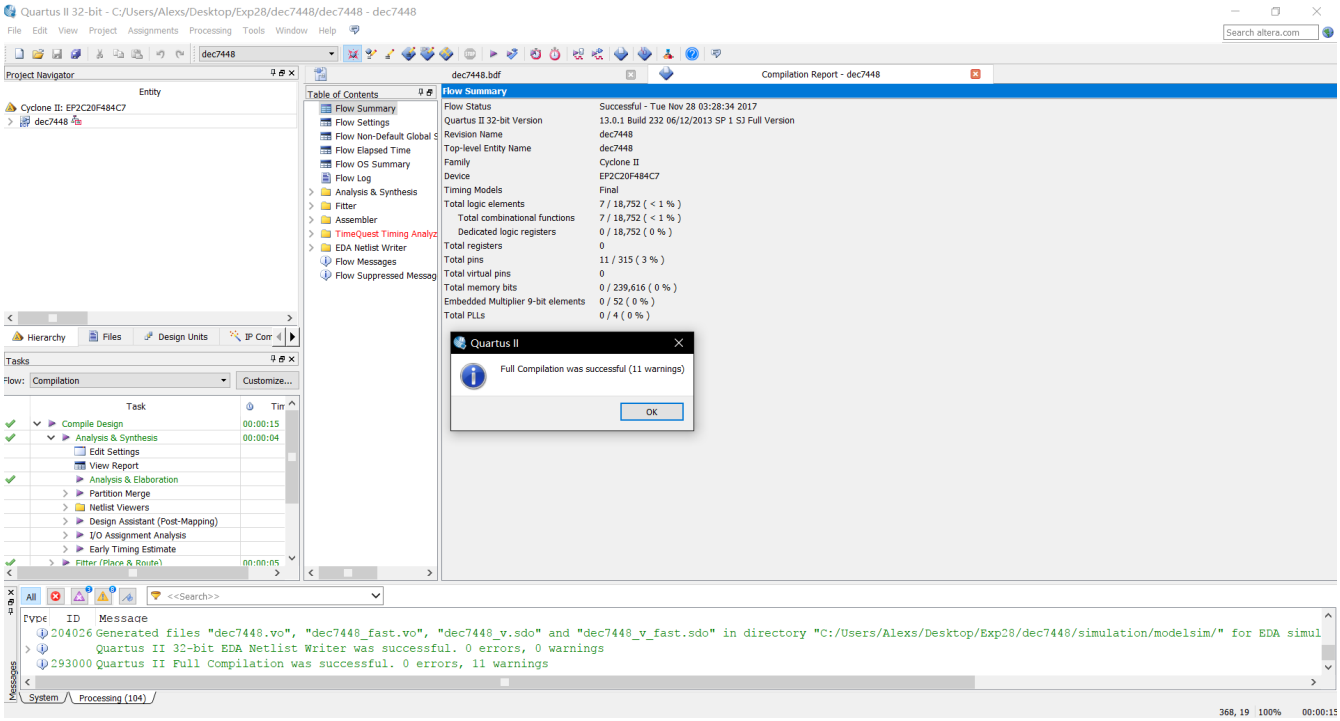


Figure 1: Compilation result of section 2.3

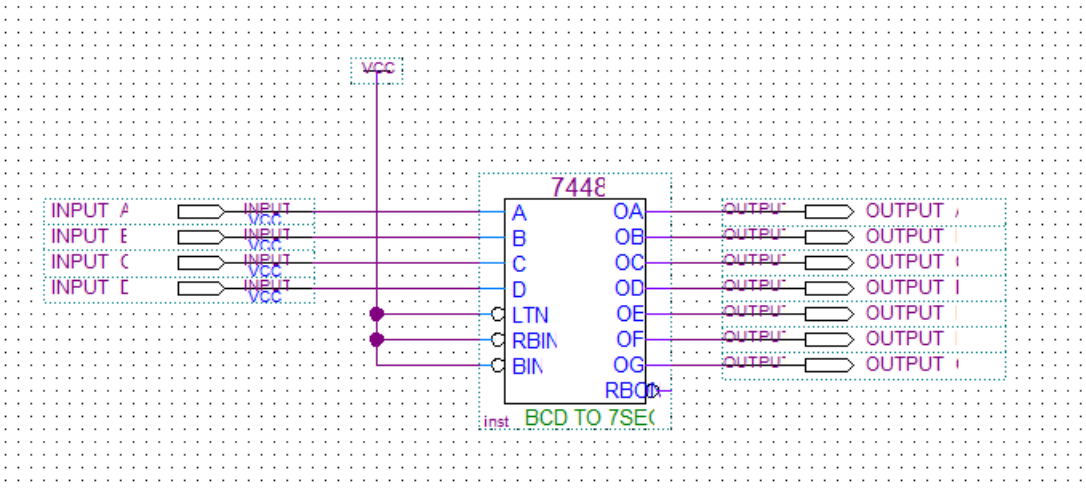


Figure 2: Schematic of section 2.3

Comment/Explanation: As can be seen in Figure 1, the compilation report did not display any error, indicating everything was done correctly. Though there are 11 warnings, however, in this project, they are trivial and can be ignored. The overall schematic capture and compilation were successful.

2. The output simulation waveform in Section 2.4 [3 Marks]

Screenshot:

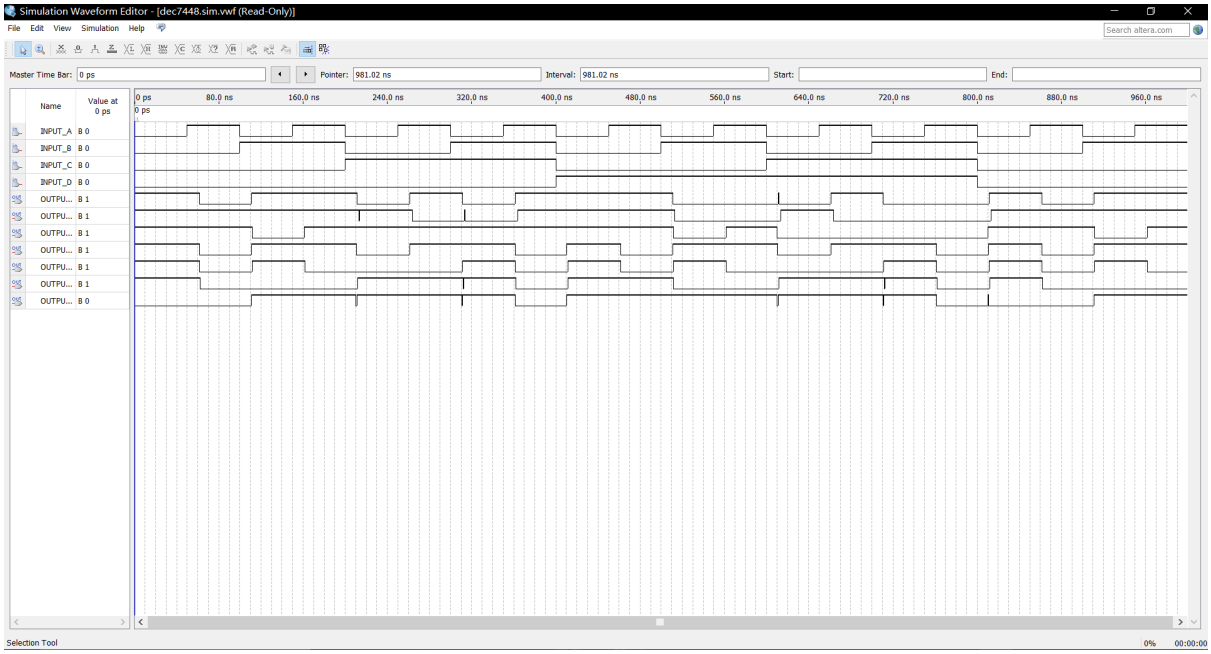
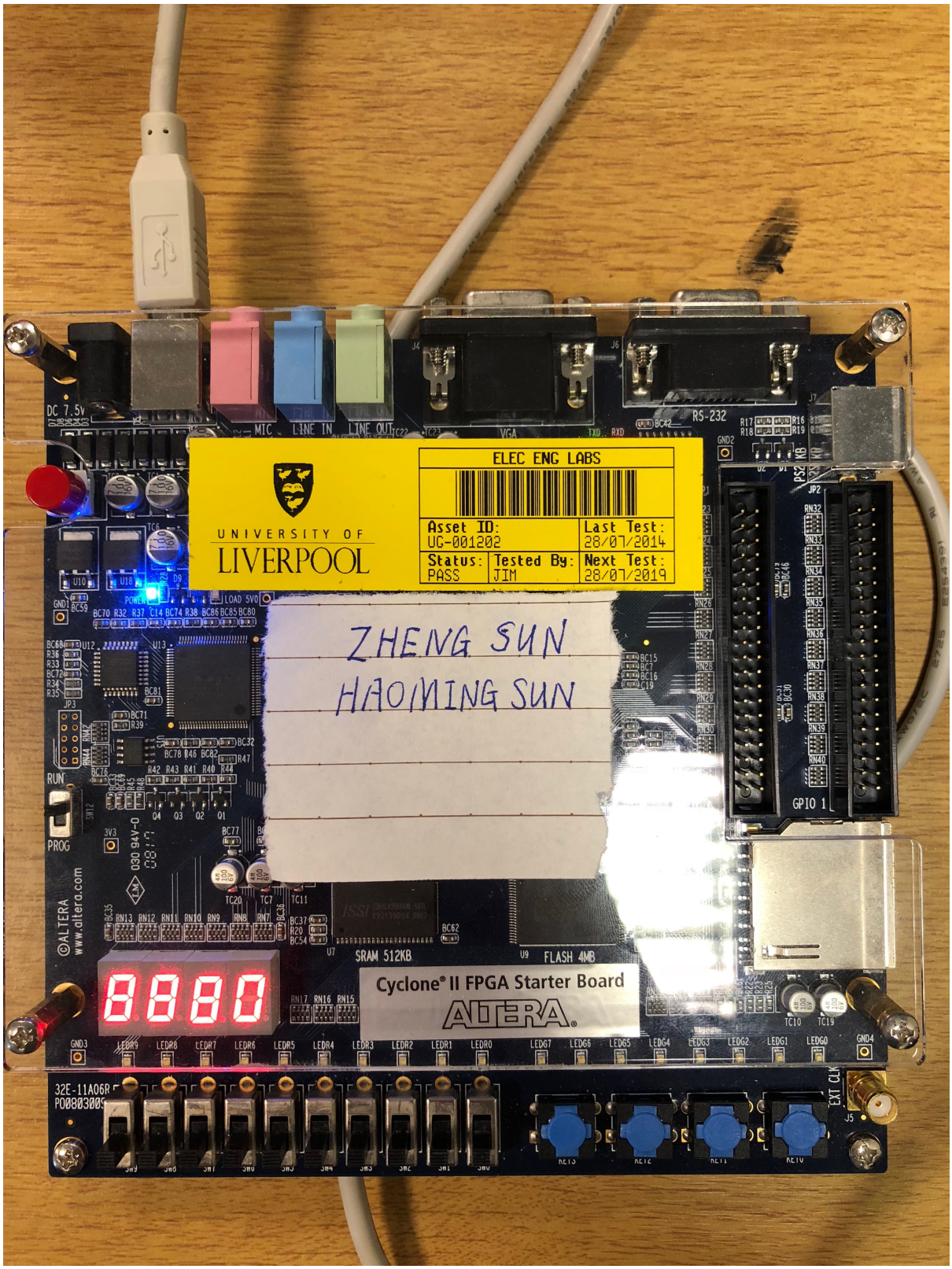


Figure 3: Simulation waveform of section 2.3

Comment/Explanation: As shown in Figure 3, the 4 input pins indicate the binary representation of input signal and the 7 output pins are decoded to represent the 7-segment LED. For example, 0000 for 0, 0111 for 7, 1111

for F. However, in this case, a high level in output voltage indicates the lights turned off, which is not usual.

3. The schematic *7SegmentDecoder.bdf* of Section 2.5 [4 Marks]



Screenshot:

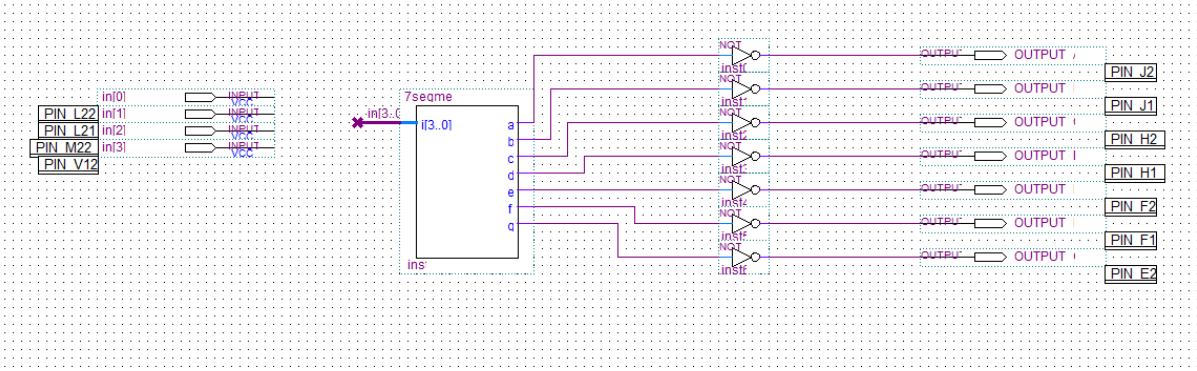


Figure 4: Schematic of the 7-Segment Decoder

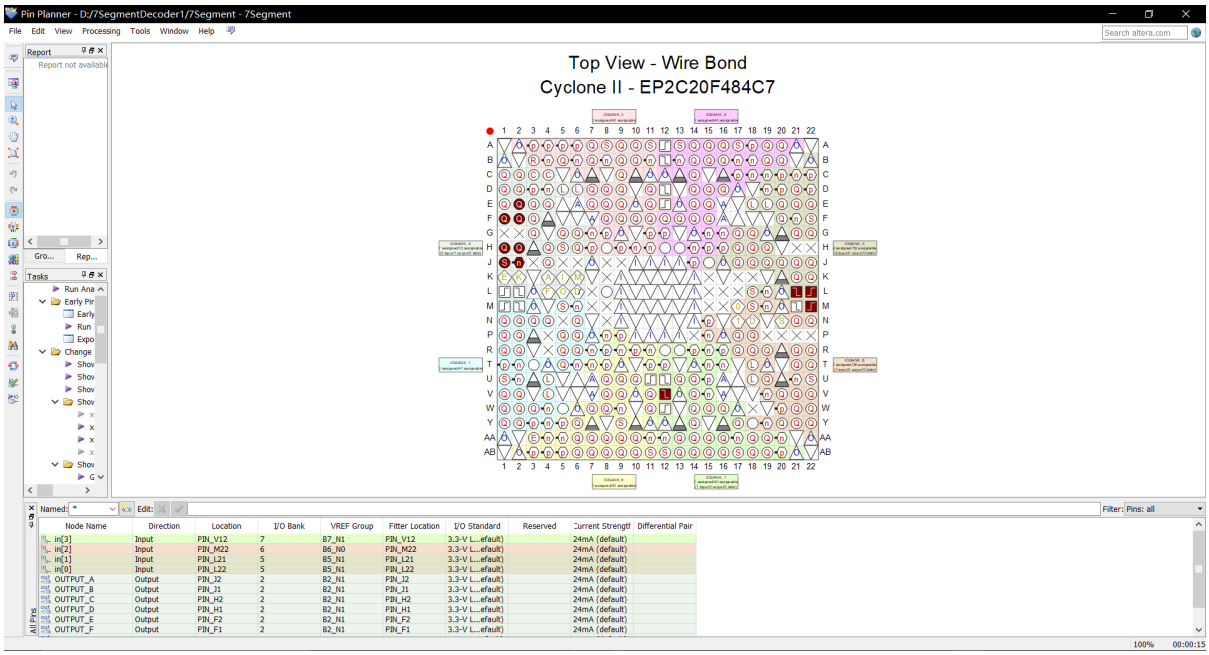


Figure 5: Pin Assignment of the 7-Segment Decoder

Comment/Explanation: This project generally does the same as that of the last section, except for that the output of the decoder were inverted to make the 7-segment LED light as usual with certain decoded input signal.

4. The simulation waveform of Section 2.6 [4 Marks]

Screenshot:

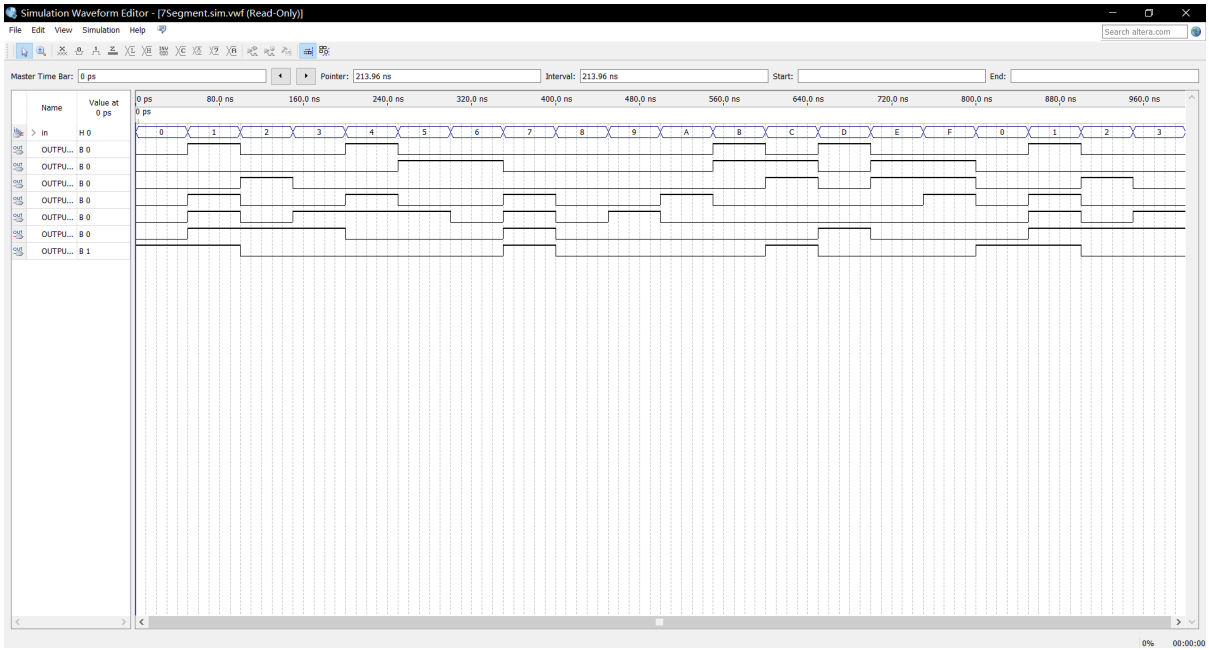


Figure 6: Simulation of the 7-Segment Decoder

Comment/Explanation: As can be seen form the simulation figure, the outputs are inverted with a voltage level LOW to signal a LED segment lighting.

5. The schematic *dec_counter.bdf* of Section **Error! Reference source not found.**.1 [4 Marks]

Screenshot:

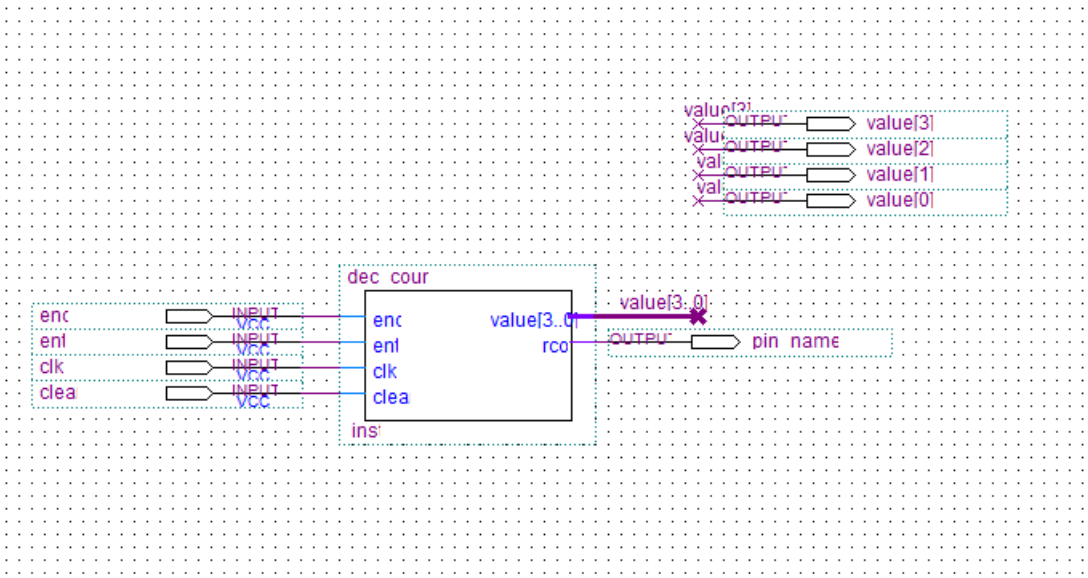


Figure 7: Schematic of the Decimal Decoder

Comment/Explanation: As can be seen in the figure, there are 4 input pins (enc, ent, clk, clear) and 5 output pins (the bus with 4 output pins value [3..0] and 1 output carry rco).

6. **The simulation waveform of Section Error! Reference source not found.3.1** [4 Marks]

Screenshot:

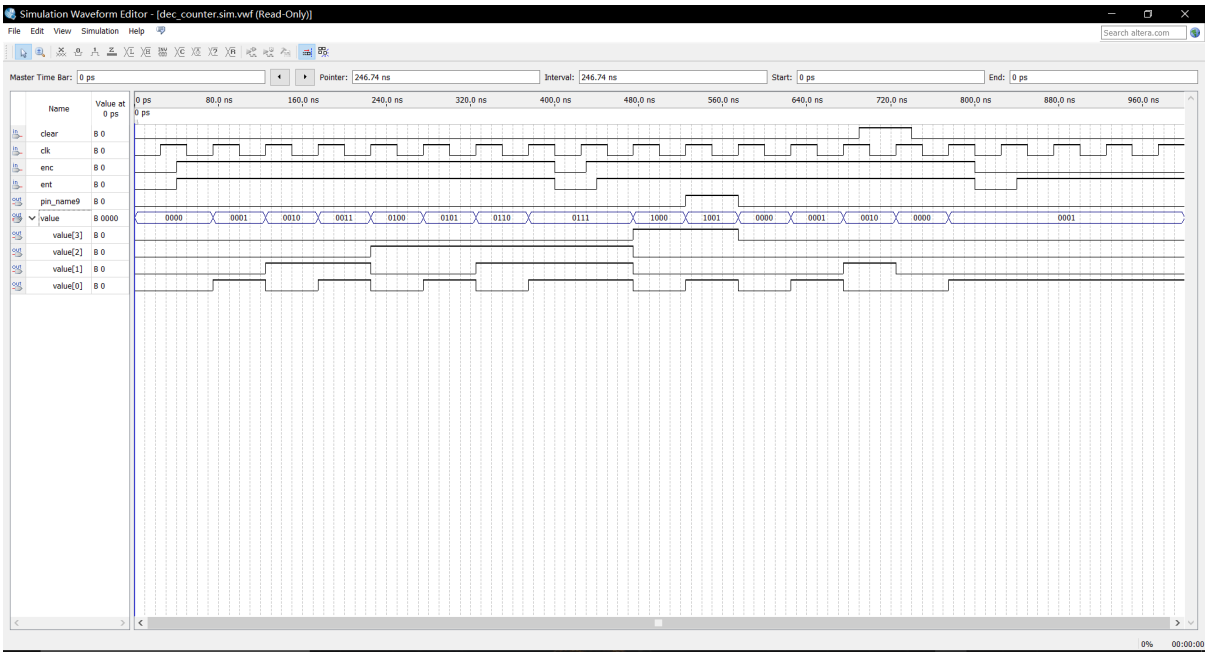


Figure 8: Simulation of the Decimal Decoder

Comment/Explanation: As can be seen in the figure, only when the 2 enable outputs, ent and enc are both HIGH, the output will rise with the clock rising.

7. The schematic of Section **Error! Reference source not found.**3.2 with pin assignment [4 Marks]

Screenshot:

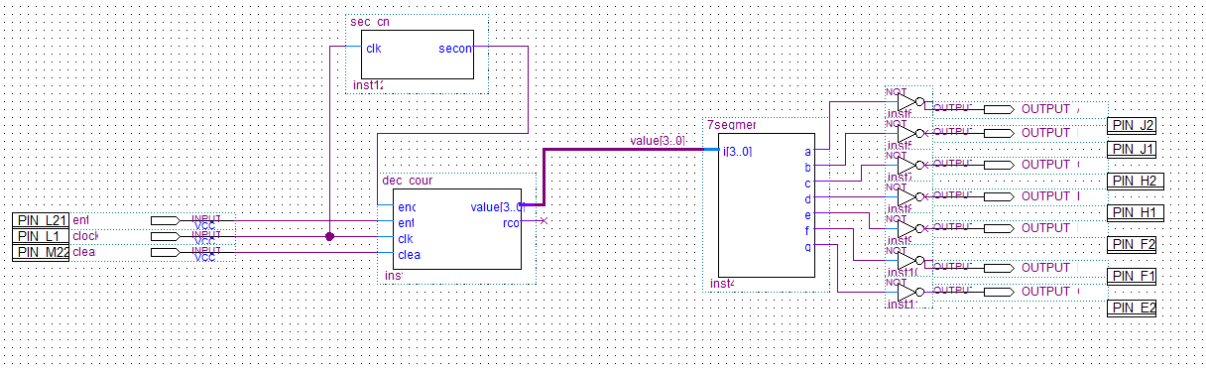


Figure 9: Schematic of the modified counter

Comment/Explanation: As can be seen in the figure, there are 3 main parts being used in the figure: the 7-segment decoder, the decimal counter and a second counter which is used to condition the frequency since the original clock frequency is too high.

8. The simulation waveform and its explanation of Section 3.2 [4 Marks]

Screenshot:

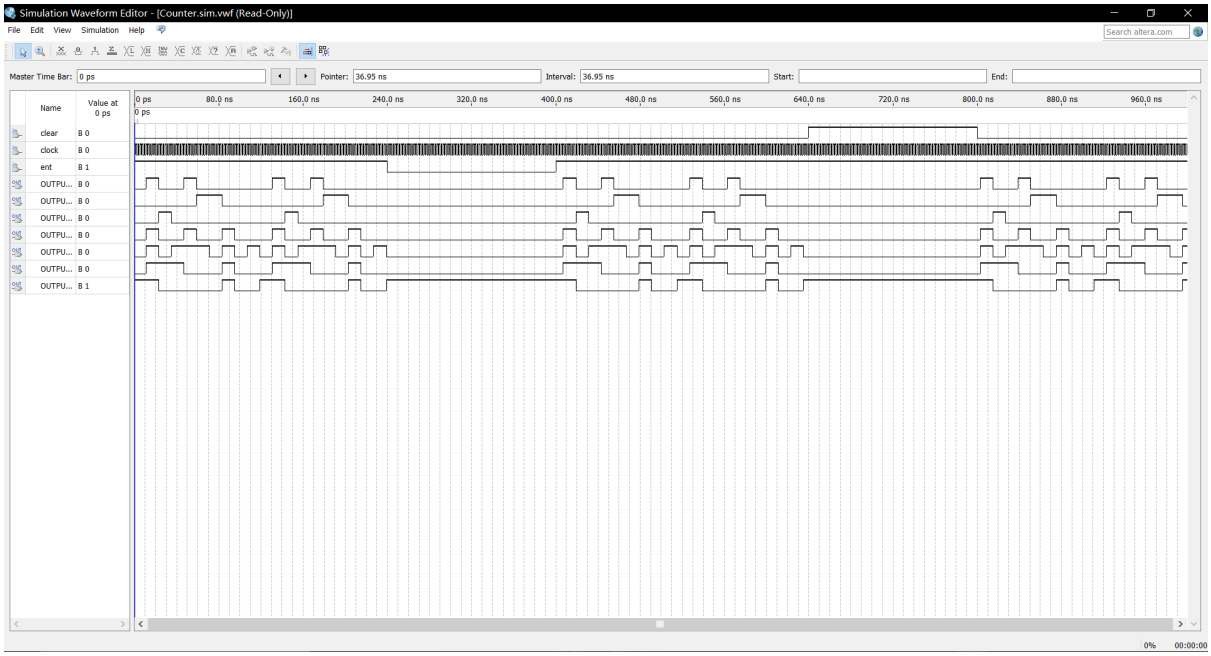


Figure 10: Simulation of the modified counter

Comment/Explanation: As can be seen in the figure, the counter pauses with the ent dropping to LOW; the counter can recover later with ent set back. A HIGH in clear will reset all the output to LOW.

9. **The schematic and simulation of the divide-by-12 and display on two Seven Segment of Section 3.2** **[10 Marks]**

Screenshot:

Comment/Explanation:

10. The code, schematic design and simulation of Section 4.1 [10 Marks]

The code (as text):

```
SUBDESIGN single_pulsar
    (
    clk, KEY0     :INPUT;
    KEY1, reset   :INPUT;
    z             :OUTPUT;
    )
VARIABLE
    y : NODE;
    ss: MACHINE OF BITS(z) WITH STATES(
     s0=0,
     s1=1,
     s2=0,
     s3=0
    );

BEGIN
    ss.clk  =  clk;
    ss.reset = reset;
    y = KEY0 or KEY1;
    TABLE
    ss,   y  =>   ss;
    s0,   0  =>    s0;
    s0,   1 =>    s2;
    s1,   0  =>    s2;
    s1,   1  =>    s3;
    s2,   0  =>    s2;
    s2,   1  =>    s1;
    s3,   0  =>    s0;
    s3,   1  =>    s3;
    END TABLE;
END;

SUBDESIGN dec_count
(
enc, ent, clk : INPUT; % two enables and the clock %
clear : INPUT; % Synchronous clear %
```

```
value[3..0] : OUTPUT; % Four output bits %
rco : OUTPUT; % ripple carry out %
)
VARIABLE
count[3..0] : DFF; % locally define 4 D-Flip-Flops for the count %
BEGIN
count[].clk = clk; % Connect the clock input to the DFF°Øs clock %
value[] = count[]; % connect the outputs of the DFFs to the outputs %
IF (clear) THEN % if clear is true clear the count i.e. %
count[].d = 0; % load the flipflops with zero %
ELSIF (enc & ent & (count[].q != 9)) THEN
 % if both enables are true and the count does not %
count[].d = count[].q + 1; % equal nine then add one to the count value %
ELSIF (enc & ent & (count[].q == 9)) THEN
% if both enables are true and the count does %
count[].d = 0; % equal nine then load the flip flops with zero %
ELSE % with no enable keep the flips flops at the same value %
count[].d = count[].q;
END IF;
rco = ((count[].q == 9) & ent);% generate the rco when the count is nine and
ent is true %
END;
```

7-segment decoder

```
%   -a-                          %
% f|   |b                        %
%   -g-                          %
% e|   |c                        %
%   -d-                          %
%                                %
% 0 1 2 3 4 5 6 7 8 9 A b C d E F  %
%                                %
```

```
SUBDESIGN 7segment
(
  i[3..0]            : INPUT;
  a, b, c, d, e, f, g : OUTPUT;
)
BEGIN
  TABLE
    i[3..0] => a, b, c, d, e, f, g;
```

```
    H"0"    => 1, 1, 1, 1, 1, 1, 0;

    H"1"    => 0, 1, 1, 0, 0, 0, 0;
    H"2"    => 1, 1, 0, 1, 1, 0, 1;
    H"3"    => 1, 1, 1, 1, 0, 0, 1;
    H"4"    => 0, 1, 1, 0, 0, 1, 1;
    H"5"    => 1, 0, 1, 1, 0, 1, 1;
    H"6"    => 1, 0, 1, 1, 1, 1, 1;
    H"7"    => 1, 1, 1, 0, 0, 0, 0;
    H"8"    => 1, 1, 1, 1, 1, 1, 1;
    H"9"    => 1, 1, 1, 1, 0, 1, 1;
    H"A"    => 1, 1, 1, 0, 1, 1, 1;
    H"B"    => 0, 0, 1, 1, 1, 1, 1;
    H"C"    => 1, 0, 0, 1, 1, 1, 0;
    H"D"    => 0, 1, 1, 1, 1, 0, 1;

    H"E"    => 1, 0, 0, 1, 1, 1, 1;
    H"F"    => 1, 0, 0, 0, 1, 1, 1;
  END TABLE;
END;
```

Screenshot of the schematic design:

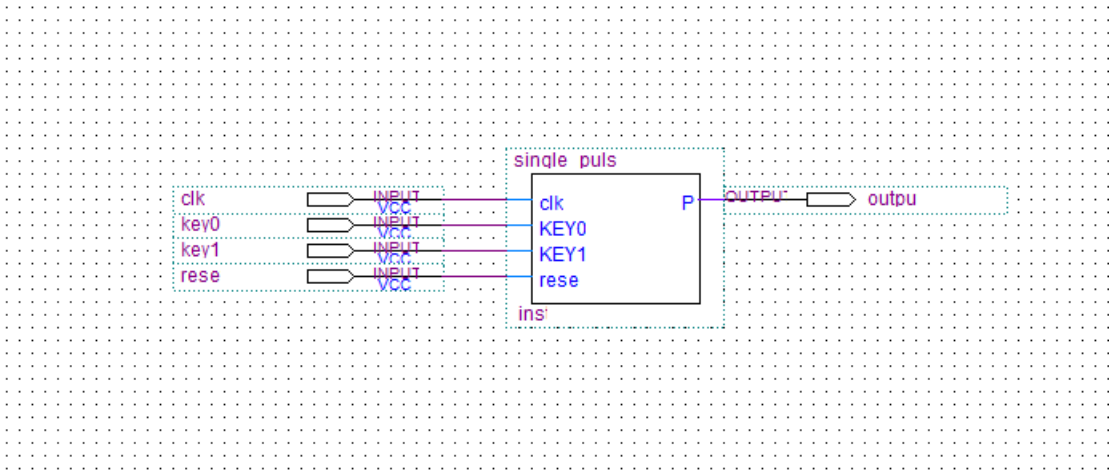


Figure 11: Schematic of the Single Pulsar

Screenshot of the simulation:

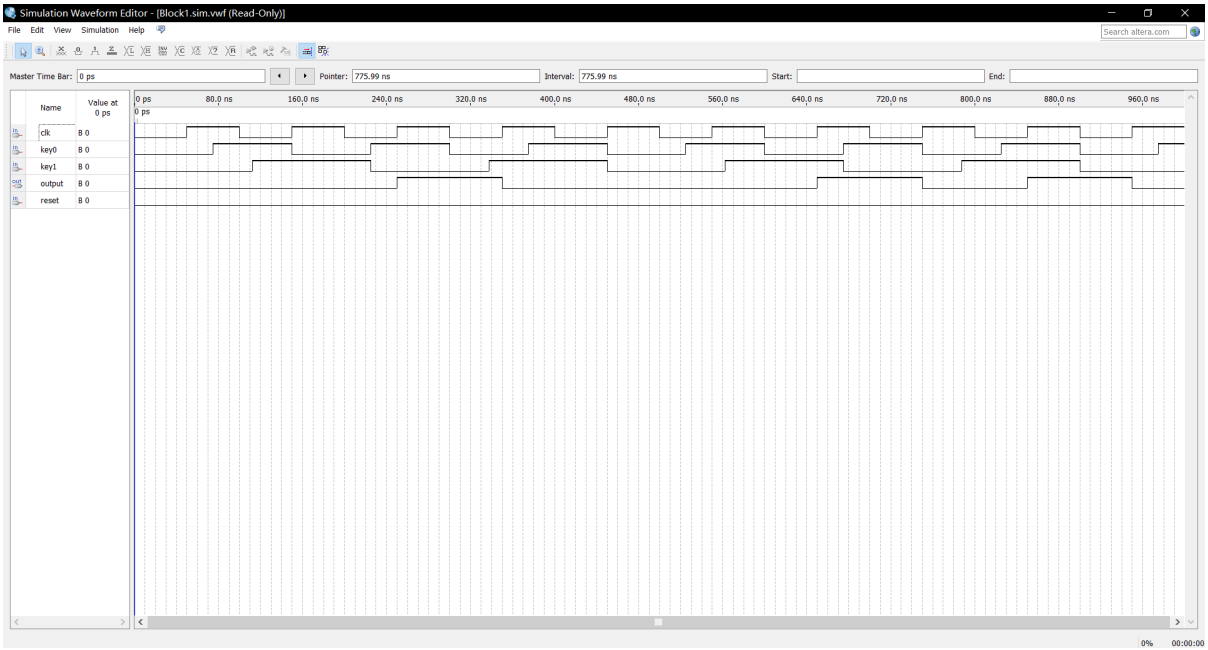


Figure 12: Simulation of the Single Pulsar

Comment/Explanation: The reset input can eliminate the output of the signal at once. Key1 and key2 are needed to enable the single pulsar.

11. The code, schematic design and simulation of Section 4.2 [10 Marks]

The code (as text):

The code (as text):

```
SUBDESIGN single_pulsar
    (
    clk, KEY0     :INPUT;
    KEY1, reset   :INPUT;
    z             :OUTPUT;
    )
VARIABLE
    y : NODE;
    ss: MACHINE OF BITS(z) WITH STATES(
     s0=0,
     s1=1,
     s2=0,
     s3=0
    );

BEGIN
```

```
    ss.clk  =  clk;
    ss.reset = reset;
    y = KEY0 or KEY1;
    TABLE
    ss,   y  =>   ss;
    s0,   0  =>    s0;
    s0,   1  =>    s2;
    s1,   0  =>    s2;
    s1,   1  =>    s3;
    s2,   0  =>    s2;
    s2,   1  =>    s1;
    s3,   0  =>    s0;
    s3,   1  =>    s3;
    END TABLE;
END;
```

Screenshot of the schematic design:

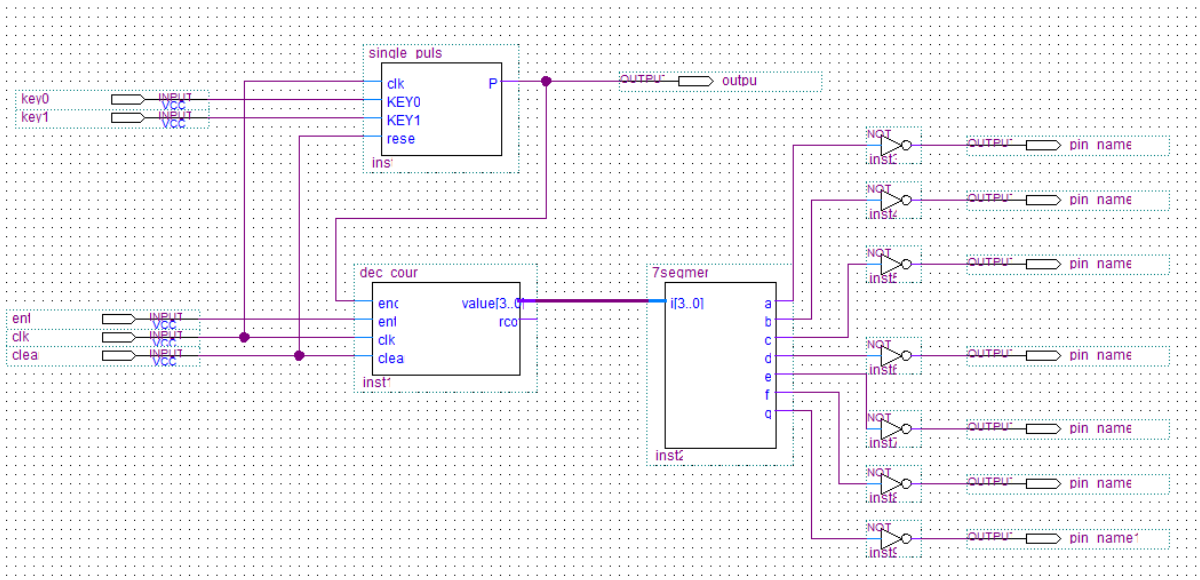


Figure 13: Schematic of the Pushbutton Enabled Counter

Screenshot of the simulation:

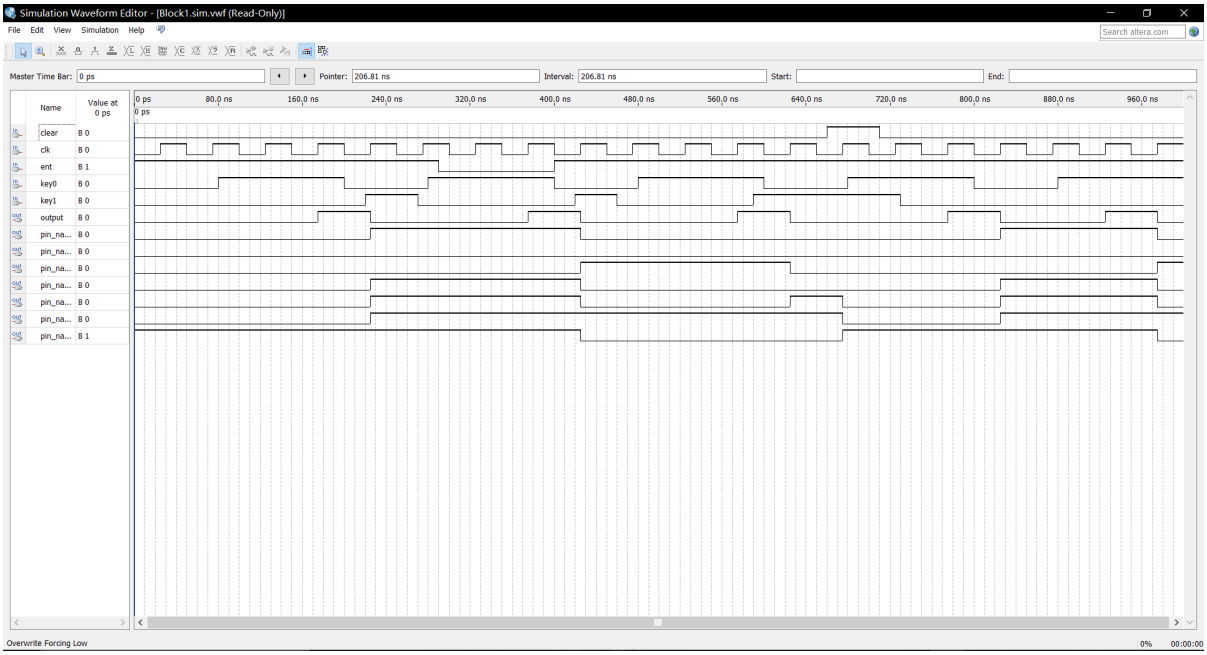


Figure 14: Simulation of the Pushbutton Enabled Counter

Comment/Explanation: As can be seen from the simulation, effective signal will be generated only when the active signal cover the rising edge; Only one signal is generated at once.

12. The code, schematic design and simulation of Section 4.3 [10 Marks]

The code (as text):

Screenshot of the schematic design:

Screenshot of the simulation:

Comment/Explanation: