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

I confirm that I have read and understood the University's definitions of plagiarism and collusion from the Code of Practice on Assessment. I confirm that I have neither committed plagiarism in the completion of this work nor have I colluded with any other party in the preparation and production of this work. The work presented here is my own and in my own words except where I have clearly indicated and acknowledged that I have quoted or used figures from published or unpublished sources (including the web). I understand the consequences of engaging in plagiarism and collusion as described in the Code of Practice on Assessment (Appendix L).

Important: Marking of all coursework is anonymous. Do not include your name, student ID number, group number, email or any other personal information in your report or in the name of the file submitted via Canvas. A penalty will be applied to submissions that do not meet this requirement.

Important

- All **code** should be included as **text** and **not** screenshots.
- For **every snapshot** from the screen (whether these are of schematics or simulations):
  - **FIRST** use **PrintScreen** under MS Windows, or CMD+SHIFT+3 or **Screenshot** under macOS, to show the entire desktop including time and date. Dump these full-screen screenshots at the end of your Report in an Appendix, as evidence that the work done is uniquely your own.
  - **THEN SECONDLY,** use the **Snipping** tool (MS Windows) or **Screenshot** tool (macOS) to show the relevant part of your design or simulation. This 'focussed' screenshot is the one you should include in the appropriate section of your Report.
  - If the entire desktop, including taskbar + time + date + initialled filename is not visible in a screenshot in the Appendix, the associated 'focused' screenshot in your main Report may be ignored, with that section receiving a mark of zero.
- **Throughout this assignment, please insert your last-letter initials (see p.9 of the main script) and underscore, at the beginning of all saved filenames. This should be readable from the title of the relevant window. This instruction is implied in the main script with "YI" for "your initials".** If this format is not followed, corresponding sections may receive a mark of zero.
- Including screenshots / photos of other people's work is considered as academic malpractice and will be penalised in accordance with the University Codes of Practice on Assessment.
- **Take photographs of your work in programming the DE1 board (see below.) All photographs and screenshots should have a Figure number and a brief caption.**

Section 2. Project Design based on Schematic Capture [14 Marks]

Section 2.3 Drawing Your Schematic [3 Marks]

The schematic, including the compilation result.

Focused screenshot:

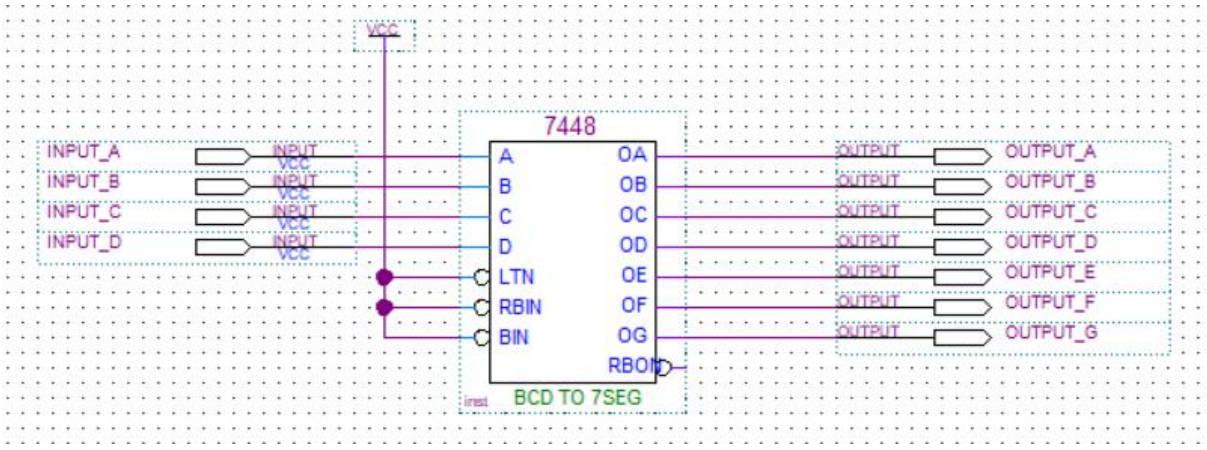


Figure 1: The focused screenshot for dec7448

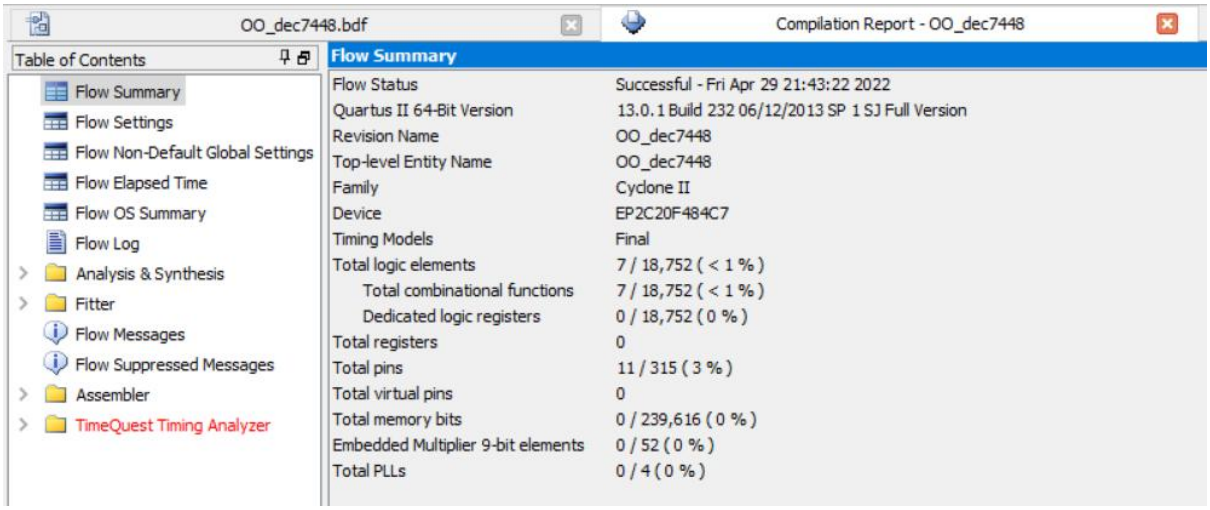


Figure 2: The compilation report corresponding to the dec7448

Comment/Explanation:

Device 7448 is made up of primitives such as logic gates and I/O pins. In my schematic, there are 4 inputs, which are. INPUT_A , INPUT_B, INPUT_C INPUT_D. Similarly, there are seven outputs in total, which are OUTPUT_A, OUTPUT_B, OUTPUT_C, OUTPUT_D, OUTPUT_E, OUTPUT_F, OUTPUT_G. To supply power to device7448, pins LTN, RBIN, BIN is connected to

the Vcc supply. After compilating, the whole circuit was compiled successfully. Despite there being eight warnings, they did not affect the subsequent experiments after checking the help messages.

Section 2.4 Waveform Entry for Simulation [3 Marks]

The output simulation waveform.

Focused screenshot:

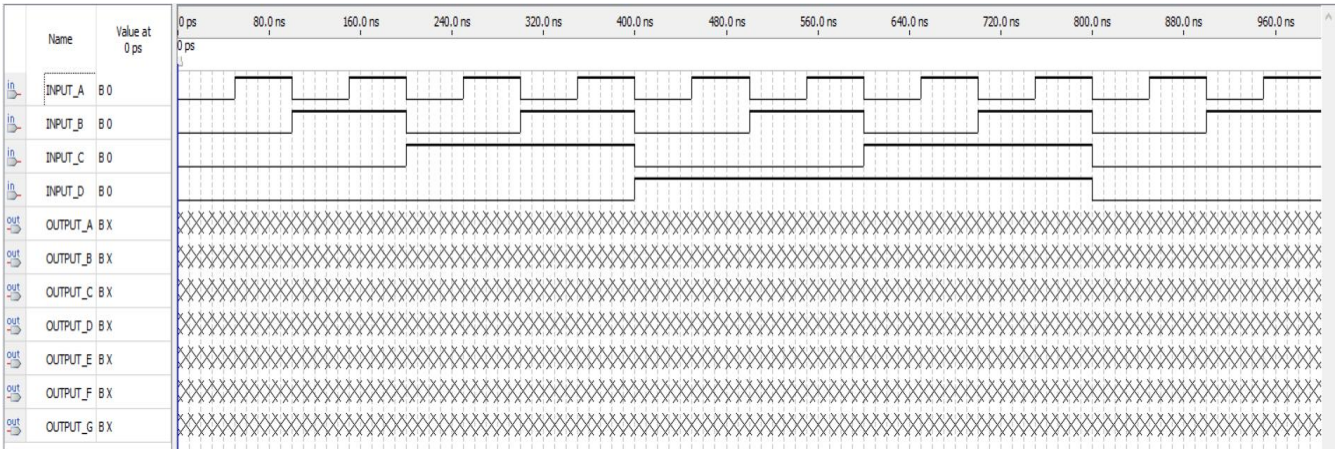


Figure 3: The input waveform for 7448

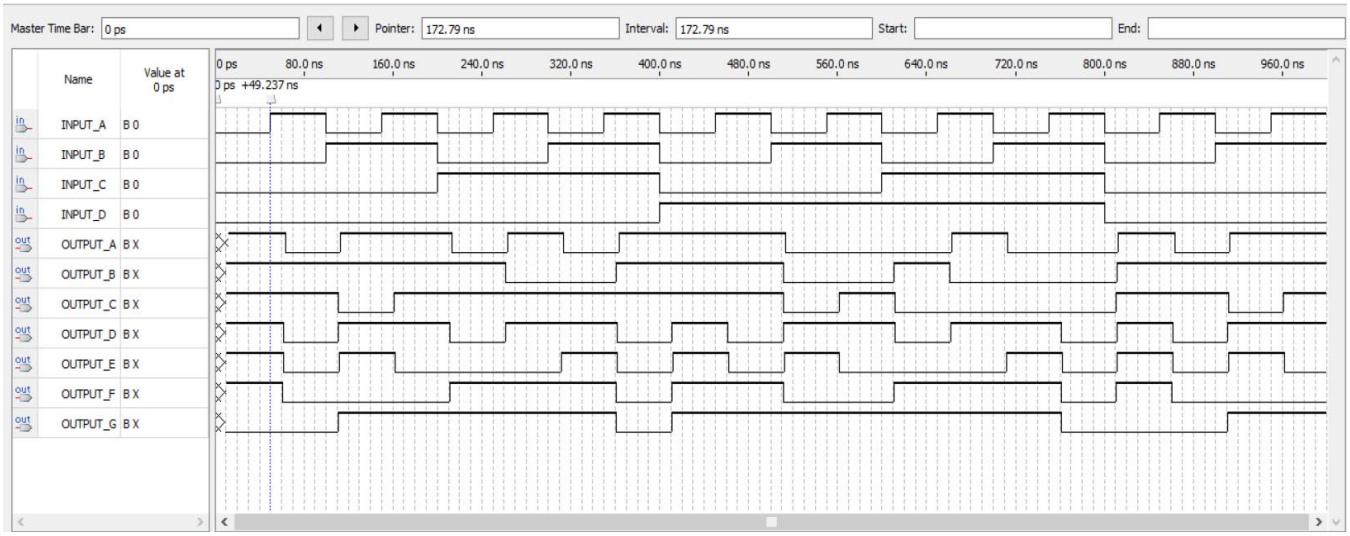


Figure 4: The simulation output file for 7448

Comment/Explanation:

As Quartus accurately simulates the propagation delay, which results in no output signal for the first 50ns. Also, the diagram above shows that when the input is initially

0000, only G is logic low in the output, which is correct. As we are using the current design as a 7 segment display, the output G is the central horizontal segment of the 7 segment display and when G is logic 0, the segment will not be on and this results in 7 segment display now displaying the number '0' as required. It can be noted that when the time is 50ns, the input signal A changes to logic 1, but the output signal does not change and remains in its original logic state, due to the propagation delay of the FPGA. However, overall, the simulated waveform meets expectations.

Section 2.5 Design Project based on Verilog Code: Decoder [4 Marks]

The schematic (with pin allocations) *YI_7SegmentDecoder.bdf*

Focused screenshot:

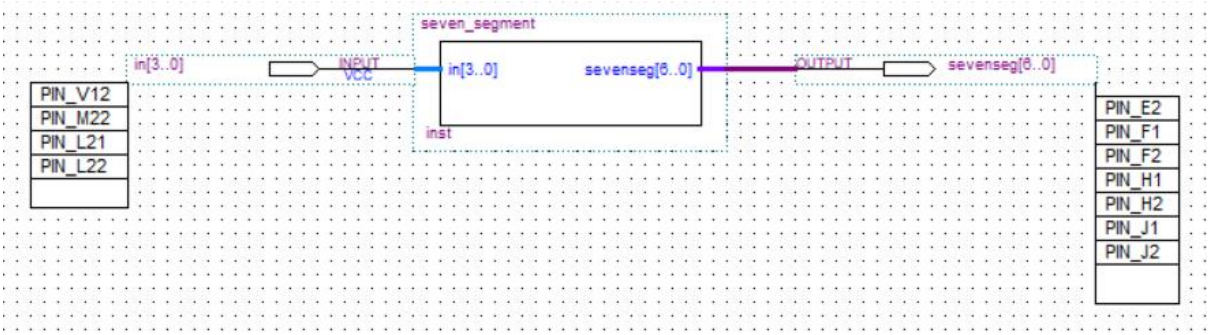


Figure 5: The focused screenshot for 7 segment decoder with pin allocations

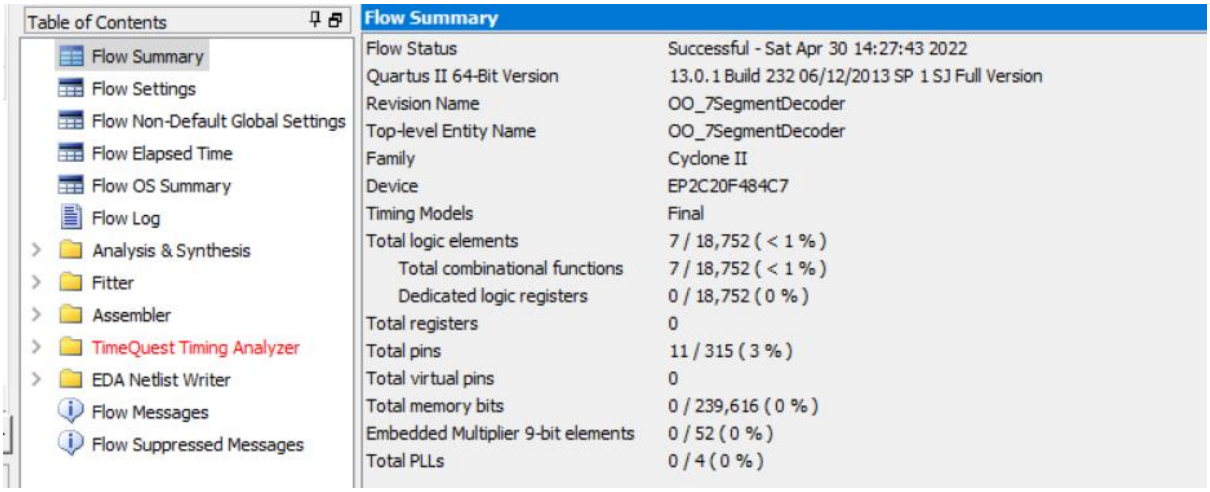


Figure 6: The compilation report corresponding to 7 segment decoder

Comment/Explanation:

Since DE1 has ten toggle switches, SW0 to SW9. The switch can provide a LOW logic level when the switch is in DOWN or OFF position, and a HIGH logic level when it is in UP position. According to this feature, the pin allocations for the input is designed. These functions will be implemented on the DE1 board. For the output pin allocations, the specific interface is correctly connected as described in the pre-lab. The specific positions corresponding to the board are as follows:

Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
in[3]	Input	PIN_V12	7	B7_N1	PIN_H3	3.3-V LV...default)		24mA (default)	
in[2]	Input	PIN_M22	6	B6_N0	PIN_J4	3.3-V LV...default)		24mA (default)	
in[1]	Input	PIN_L21	5	B5_N1	PIN_L8	3.3-V LV...default)		24mA (default)	
in[0]	Input	PIN_L22	5	B5_N1	PIN_M6	3.3-V LV...default)		24mA (default)	
sevenseg[6]	Output	PIN_E2	2	B2_N1	PIN_E2	3.3-V LV...default)		24mA (default)	
sevenseg[5]	Output	PIN_F1	2	B2_N1	PIN_F1	3.3-V LV...default)		24mA (default)	
sevenseg[4]	Output	PIN_F2	2	B2_N1	PIN_F2	3.3-V LV...default)		24mA (default)	
sevenseg[3]	Output	PIN_H1	2	B2_N1	PIN_H1	3.3-V LV...default)		24mA (default)	
sevenseg[2]	Output	PIN_H2	2	B2_N1	PIN_H2	3.3-V LV...default)		24mA (default)	
sevenseg[1]	Output	PIN_J1	2	B2_N1	PIN_J1	3.3-V LV...default)		24mA (default)	
sevenseg[0]	Output	PIN_J2	2	B2_N1	PIN_J2	3.3-V LV...default)		24mA (default)	
<<new node>>									

Figure 7: The pin allocations for the segment decoder

As can be seen from the figure, each segment [] output port and inputs have been allocated a suitable position.

Section 2.6 Input Simulation for Decoder [4 Marks]

The simulation waveform

Focused screenshot:

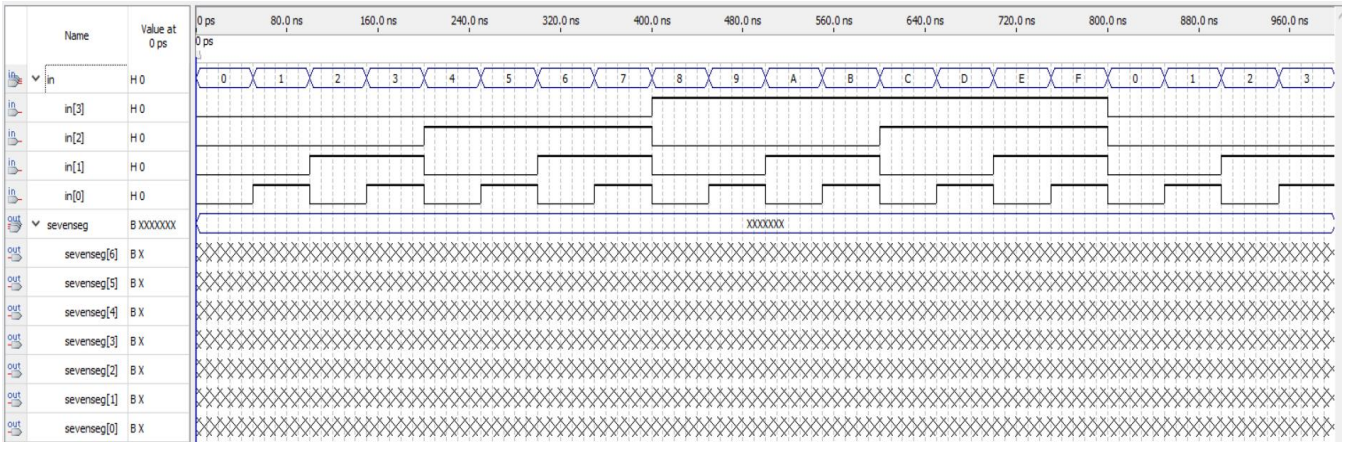


Figure 8: The input waveform for the decoder

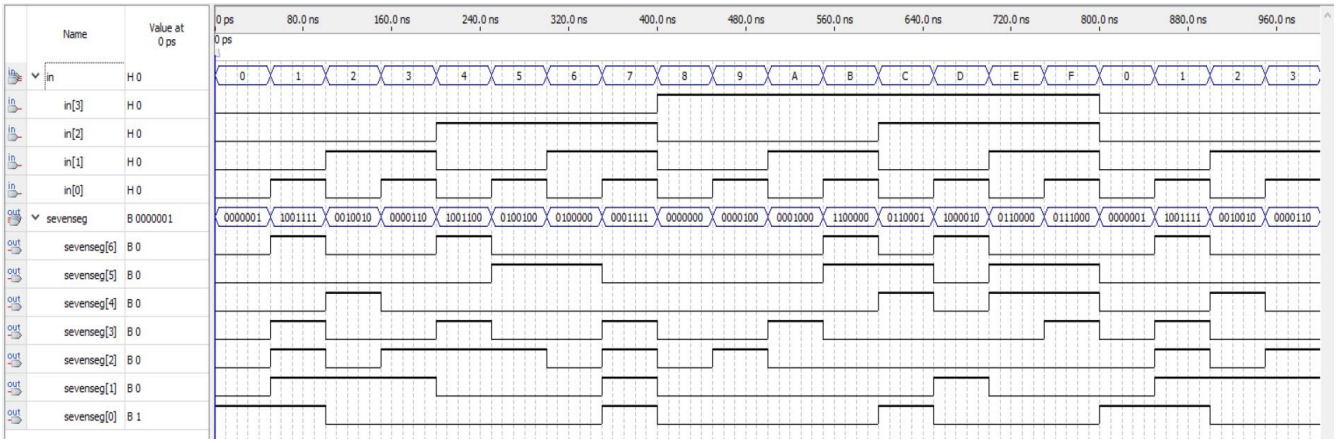


Figure 9: The simulation output file for the decoder

Comment/Explanation:

Before the simulation, I changed the input radians to hexadecimal and adjusted the Count period to 50ns. It is particularly noteworthy that the “Functional simulation” is used to replace the “Timing Simulation” in the previous simulation, the benefit of the replacement is that the approximate timings will be slightly avoided and the data from the simulation is much closer to the theoretical value.

As can be seen from the figure, only the sevenseg[0] port is at high logic level at the time corresponding to the number "0", which is as expected. segment[0] corresponds to the segment at the horizontal centre of the 7 segment display. Only When the output is at high logic level, the corresponding segment will not open. Therefore, at this time, only the central segment does not open, which corresponds to the number "0". The same rule is followed for the other digits{0-F}, the segment is only switched on when the corresponding segment [] port is logic low.

Photos of operation on the board

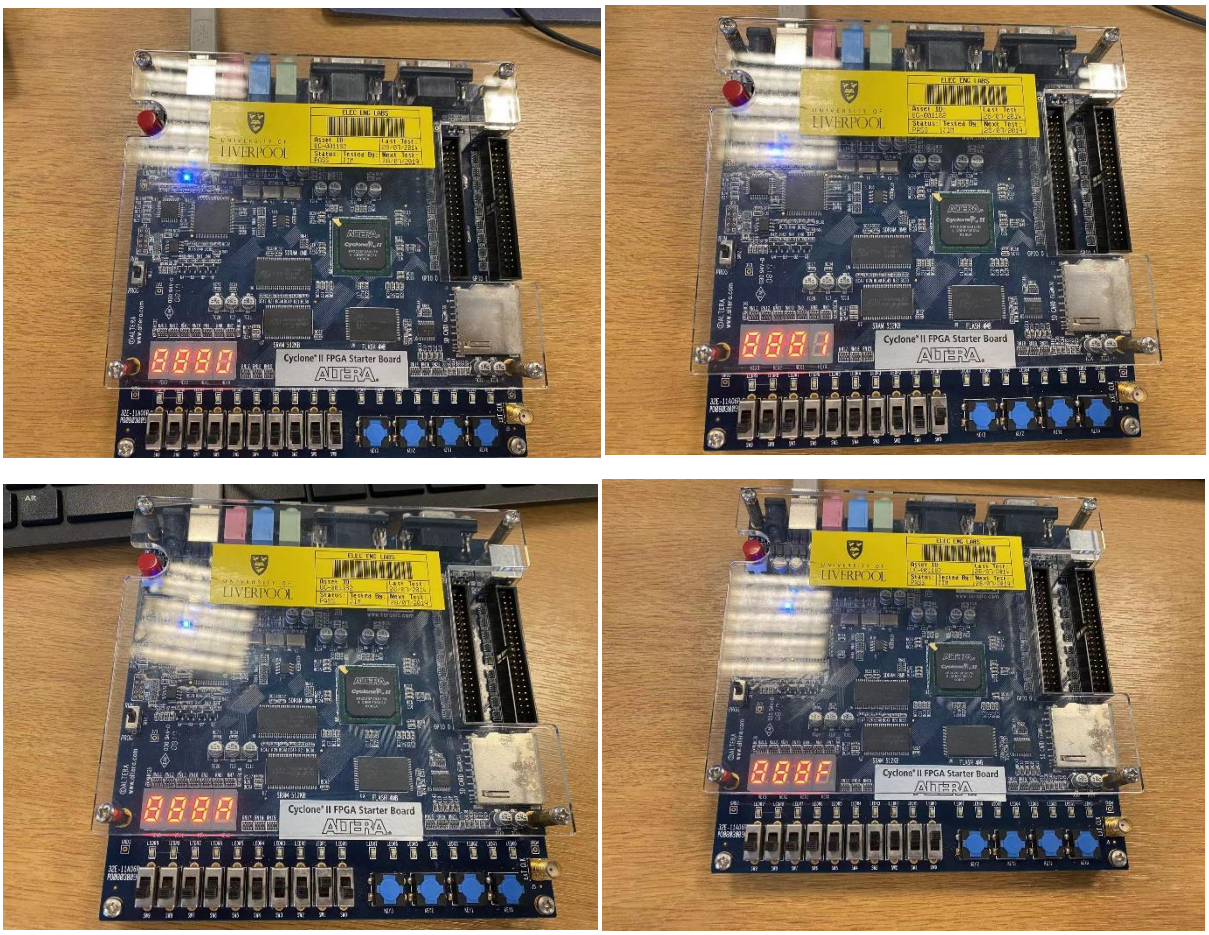


Figure 10: The real photos from the DE1 board

A total of four cases are displayed, each corresponding to a different Toggle switch case.

For the number "0", there is no switch in UP position, which corresponds to the number 0.

For the number "1", only SW0 is in the UP position, which corresponds to the displayed number 1.

For the number "A" (10), SW1 and SW4 are in the UP position, which means that 2+8=10. 7 segments display the same number and the same calculation result.

For the number "F" (15), all SW switches are in UP position, which means that 1+2+4+8=15. 7 segment display shows the same number as calculated.

Section 3. Sequential design in Verilog [26 Marks]

Section 3.1 Counter design in Verilog [8 marks]

Both schematics [**4 Marks**]

Decade counter – focused screenshot:

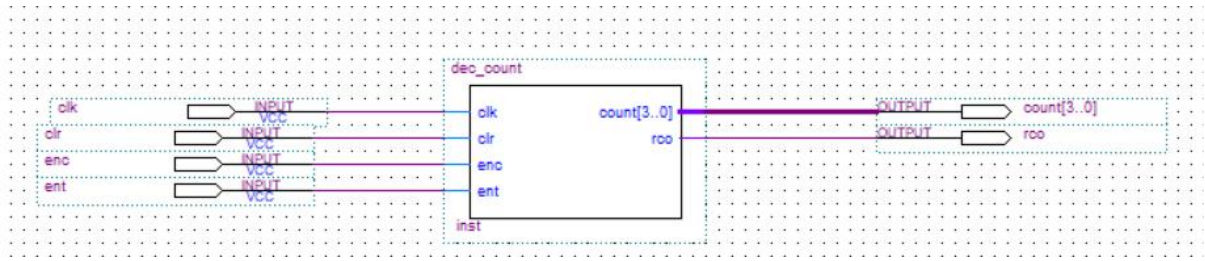


Figure 11: The schematic for decade counter

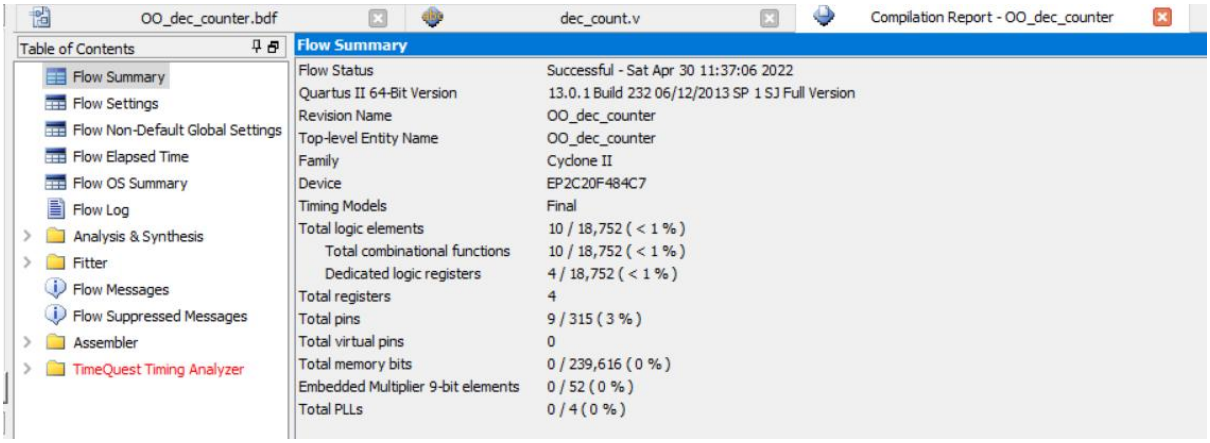


Figure 12: The compilation report for decade counter

Comment/Explanation:

In the design of the decade counter, there are four inputs: input clk, input clr, input enc and input ent. Where input clr is used as a “clear” function. In addition, there are four outputs. The program is designed so that rco is "continuously assigned", causing it to rise to the high level almost immediately when the count is 9 and ent is true. This makes rco a conditional output. In addition, the "always" block in the program is designed to be triggered on the rising clock edge, which means that it is executed whenever the clock transitions from 0 to 1.

Decade counter with 7-segment decoder – focused screenshot:

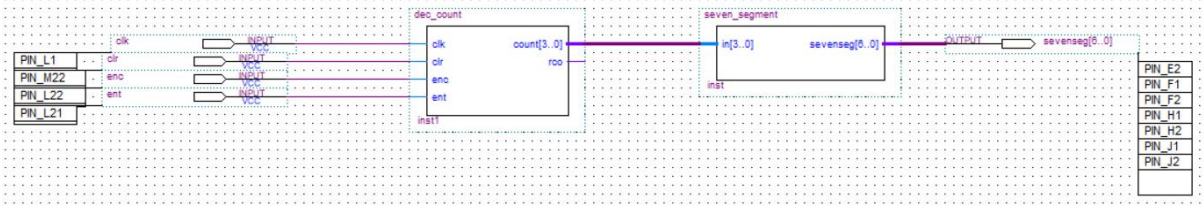


Figure 13: The schematic for decade counter with 7-segment decoder

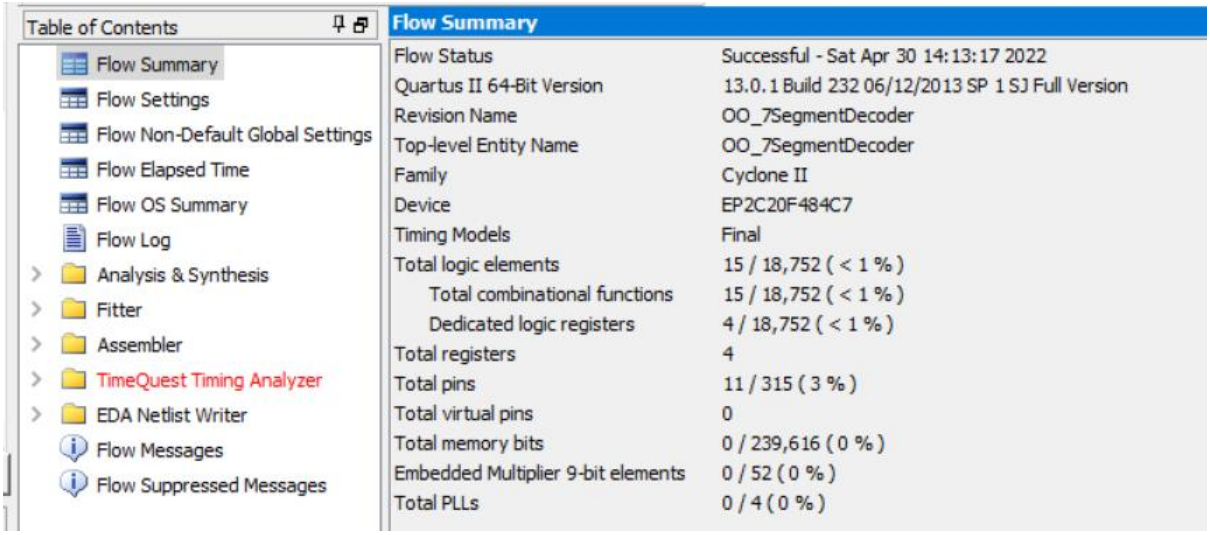


Figure 14: The compilation report for decade counter with 7-segment decoder

Comment/Explanation:

A 7-segment decoder module has been added to the previous part of the decade counter so that the numbers can be displayed on the 7-segment display. The inputs are binary digits equivalent to 0-9, which are provided manually by flipping the four toggle switches on the DE1 board. The output of the decade counter is connected to the input of the 7-segment module. The pin allocation of the outputs is the same as in the previous section, the pin allocation of the inputs has been changed and the whole pin allocations on the DE1 board are as follows:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
clk	Input	PIN_L1	2	B2_N1	PIN_L1	3.3-V LV...default)		24mA (default)	
clr	Input	PIN_M22	6	B6_N0	PIN_M22	3.3-V LV...default)		24mA (default)	
enc	Input	PIN_L22	5	B5_N1	PIN_L22	3.3-V LV...default)		24mA (default)	
ent	Input	PIN_L21	5	B5_N1	PIN_L21	3.3-V LV...default)		24mA (default)	
sevenseg[6]	Output	PIN_E2	2	B2_N1	PIN_E2	3.3-V LV...default)		24mA (default)	
sevenseg[5]	Output	PIN_F1	2	B2_N1	PIN_F1	3.3-V LV...default)		24mA (default)	
sevenseg[4]	Output	PIN_F2	2	B2_N1	PIN_F2	3.3-V LV...default)		24mA (default)	
sevenseg[3]	Output	PIN_H1	2	B2_N1	PIN_H1	3.3-V LV...default)		24mA (default)	
sevenseg[2]	Output	PIN_H2	2	B2_N1	PIN_H2	3.3-V LV...default)		24mA (default)	
sevenseg[1]	Output	PIN_J1	2	B2_N1	PIN_J1	3.3-V LV...default)		24mA (default)	
sevenseg[0]	Output	PIN_J2	2	B2_N1	PIN_J2	3.3-V LV...default)		24mA (default)	
<<new node>>									

Figure 15: The pin allocations for decade counter with 7-segment decoder

As in the previous section, the decade counter has only four inputs, input clk, input clr, input enc and input ent. Each pin is assigned a position as shown in the diagram above. Finally, the design passes compilation tests and the corresponding compilation report is as above.

Both simulation waveforms [**4 Marks**]

Simulation of counter – focused screenshot:

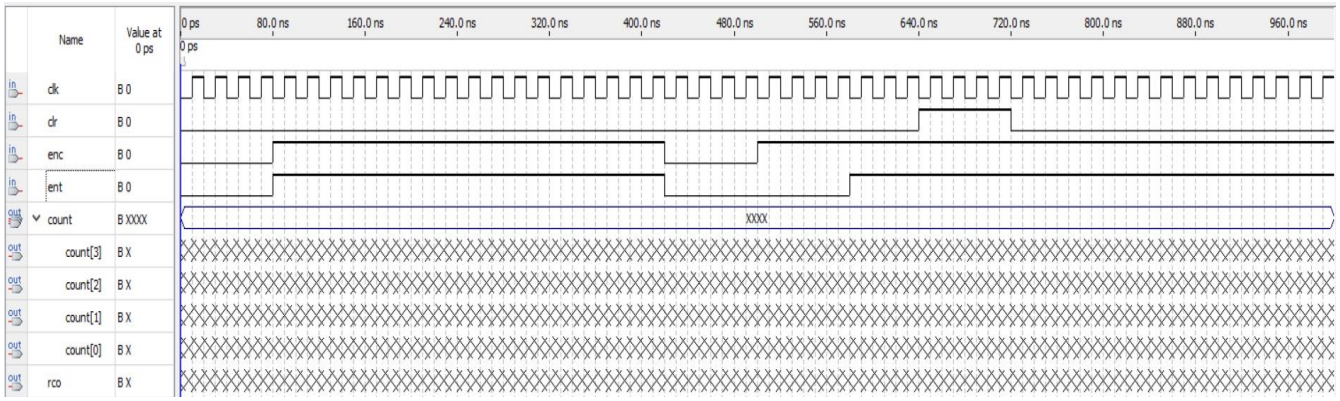


Figure 16: The simulation input waveform for decade counter

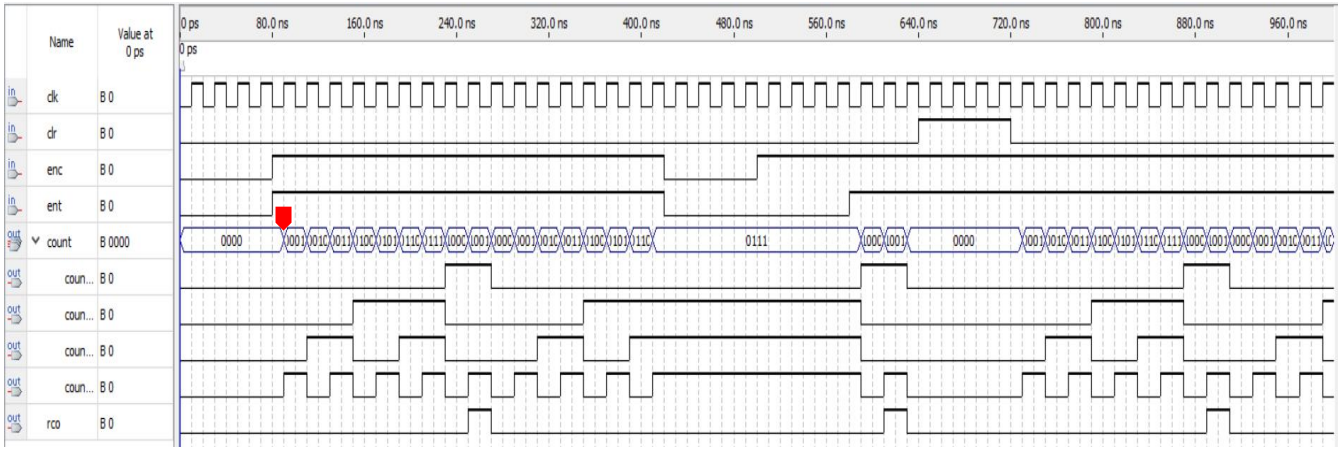


Figure 17: The simulation output file for decade counter

Simulation of counter connected to 7-segment decoder – focused screenshot:

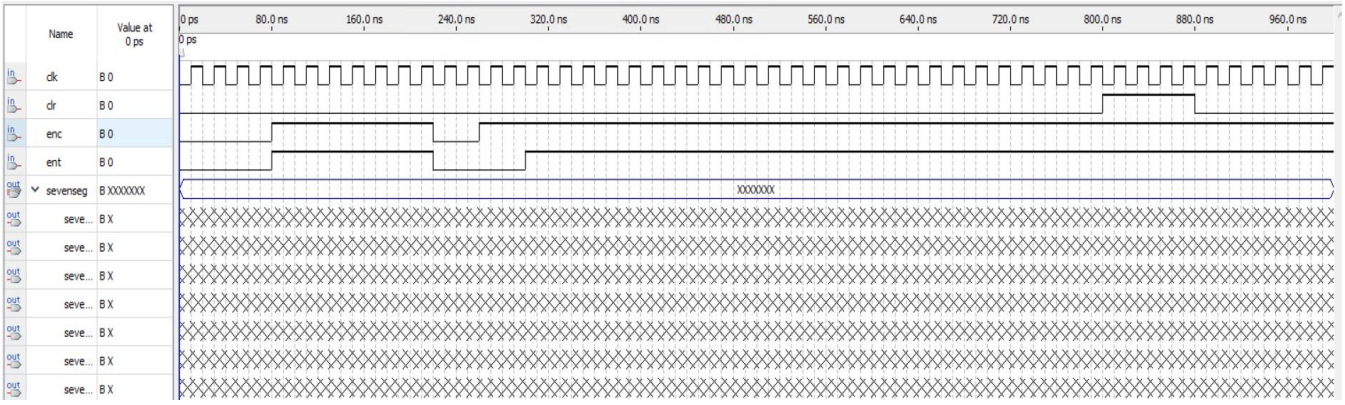


Figure 18: The simulation input waveform for decade counter with 7-segment decoder

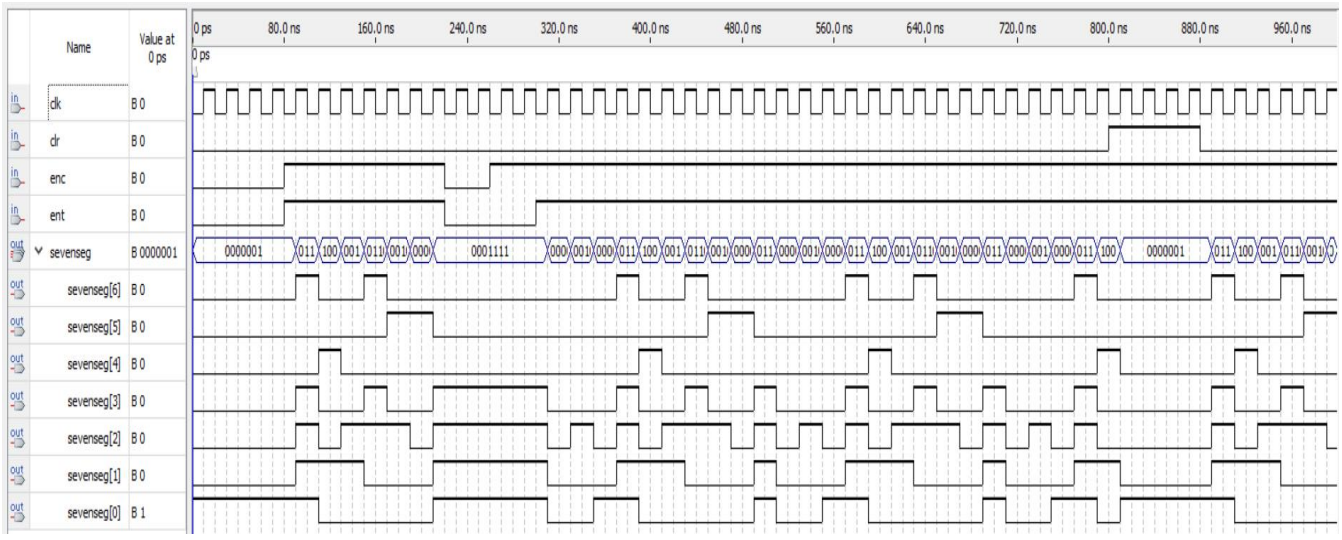


Figure 19: The simulation output file for counter with 7-segment decoder

Comment/Explanation for both:

For the decade counter, to achieve the 50MHz clock requirement, I set the period of the input clock signal to 20ns (1/50MHz = 20ns). In this simulation the functional simulation is chosen to avoid the slightly approximate timings. During the initial 0-80ns period, the output is at low level, despite the rising edge, as both “ent” and “enc” are at low level, which means that the decade counter is in a “disable” state. During the period 80-440ns, the decade counter is in normal counter operation as both ent and enc are at high level. During the 640-720ns period, because the “clr” is at high level,

the decade counter will be **cleared**, which means that the counter is in the “reset” state. In the time after clear, the counter continues to count normally again.

For the counter connected to 7-segment decoder, similarly, the period of the input clock signal is set to 20ns (1/50MHz = 20ns) and the functional simulation is chosen to avoid the slightly approximate timings. In the time period 0-80ns, since the pins “enc” and “ent” are both at low logic level, only sevenseg[0] is at high logic level. This is because sevenseg[0] corresponds to the horizontal centre of the segment of the 7-segment display, which will not turn on until the output pin is at low logic level, corresponding to the number "0". When the “enc” and “ent” is at high level, the counter is in normal operation and the number increases with time. When “clr” is high at 800-880ns, which means that the counter is in “reset” state and the number on the 7-segment display will be cleared to “0”. After the “clr” time, the counter counts normally again.

Section 3.2 Once-per-second counter [18 Marks]

The initial schematic of Section 3.2 **[4 Marks]**

Focused screenshot:

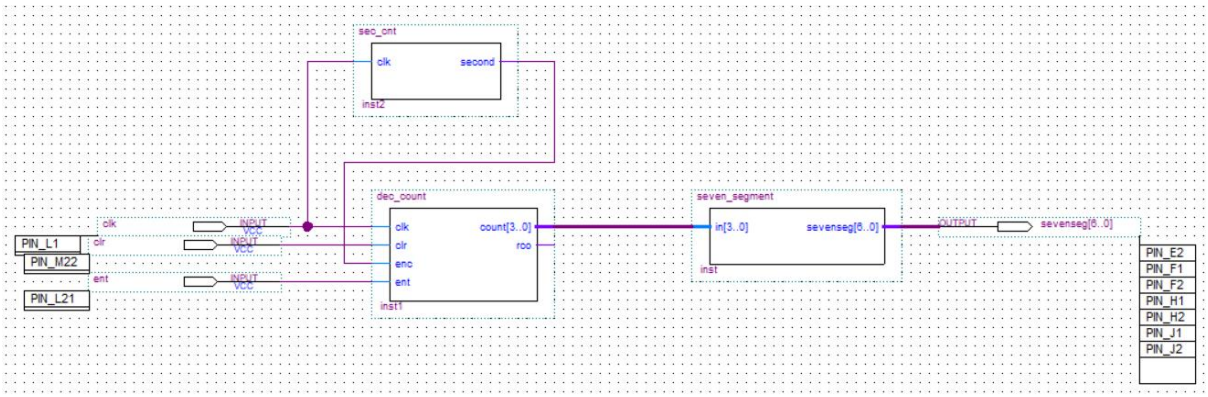


Figure 20: The focused screenshot for Once-per-second counter

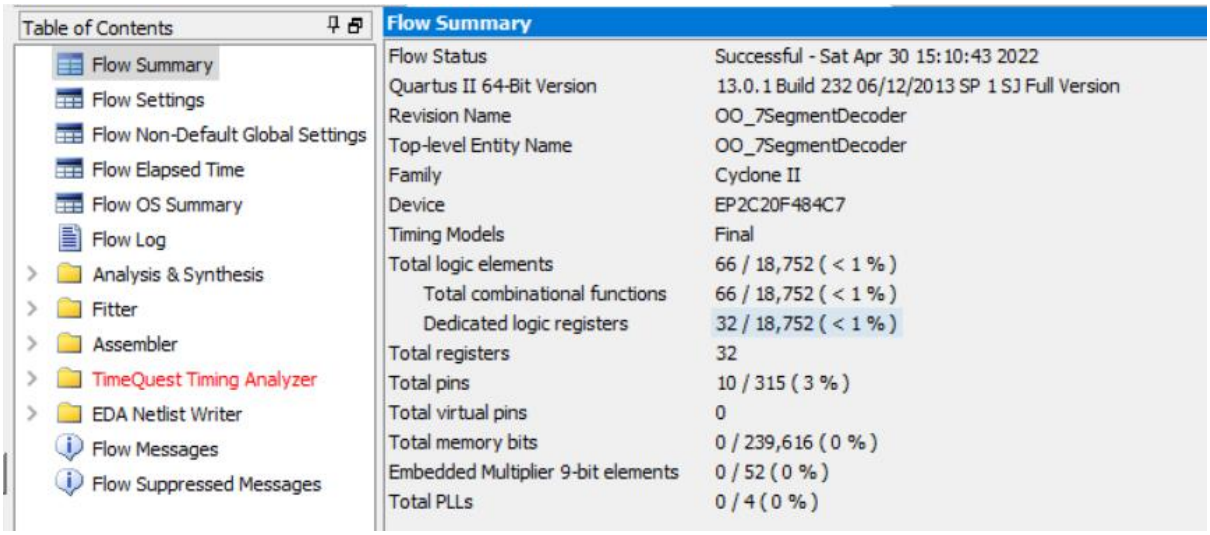


Figure 21: The compilation report for Once-per-second counter

Comment/Explanation:

In order to design the sequential circuit, a second counter module is added to the previous design and the "second" output of the second counter is connected to the "enc" input of the decade counter. Because of the **continuous assignment** of the "second" output, this ensures that the "second" immediately change to 1 in order to be available as an "enable" input to the counter at the next active edge. The counter is incremented approximately every second as required, which means that we need to enable the counter for only a single clock period every 50 million clock pulses. The "clk" input of the second counter and the "clk" input of the previous section are connected together to share the same clock signal. The position allocation of each pin remains the same.

Section 3.2 continues on the next page

(1) The simulation waveform and its explanation **[4 Marks]**

Focused screenshot:

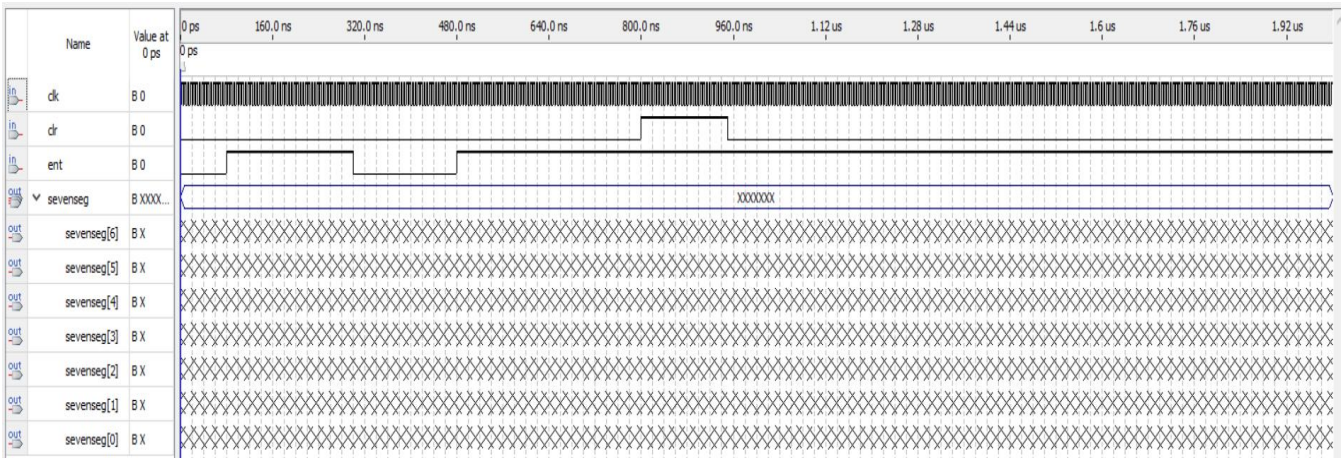


Figure 22: The input waveform for Once-per-second counter

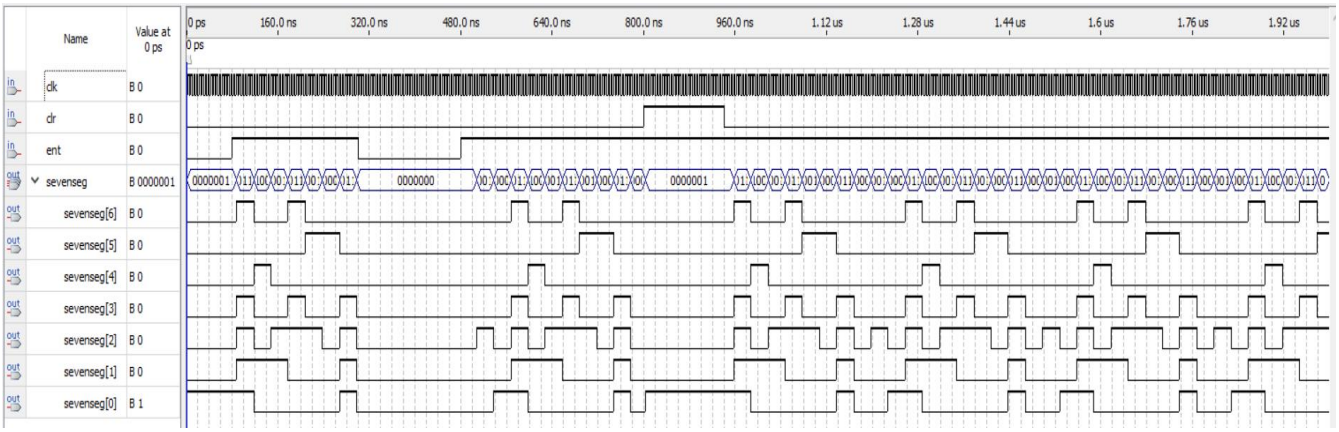


Figure 23: The simulation output file for Once-per-second counter

Comment/Explanation:

As the useful simulation period for the second counter would be far over the maximum "End Time" (100us). In order to simulate the experiment smoothly, as required by the lab script, we changed the new_count value from 49999999 to 5. However, when using the DE1 board, we have to set this value back to 49999999. To show the full cycle of relevant behaviour on simulation, the clock period is adjusted to 5ns, the End Time is adjusted to 2us with "ent" and "clr" are set to the appropriate period.

As in the previous experiment, when the input "ent" is low, only sevenseg[0] is at high logic level, corresponding to the number 0. The counter is in the "incremented" state for 80-300ns. The counter counts normally with time. When the "ent" is back at low logic level within 300-480ns, the counter is in "disabled" state. During the clear time

(800-940ns) the counter is in the "reset" state and the 7-segment display reverts to the number "0". During the 940-2000ns time period, the counter remains in the "incremented" state as the "ent" is always at high logic level and no "clear" period.

It is particularly noteworthy that the number 9 corresponds to 0000100. For example, the time period corresponding to the number 9 is 1.2-1.22us. Therefore, each number corresponds to a time period 20ns, and the time required for a complete cycle from the number 0 to the number 9 is calculated to 10*20=200ns. However, as can be seen from the simulations, the time between increments (one cycle) is closer to 300ns. This is due to the propagation delay, as the time corresponding to each digit is not strictly 20ns, and the time error is magnified when calculating the time for multiple digits, which results in more time being needed for one cycle in the simulation experiment.

(2, 3) The schematic and simulation of the divide-by-12, and of the cascading of two divide-by-12 counters **[10 Marks]**

Divide-by-12 ***code*** *– paste here as text (since the schematic would be just the same as the previous one)*

```
module dec_count (output reg[3:0] count, output rco, input clk, clr, enc, ent)
;

/* Below: a 'continuous assignment' of rco so that it goes high virtually imm
ediately
when the count is 9 and ent is true (i.e. rco goes high in a combinational wa
y, not
waiting for the active clock edge). This makes rco a conditional output.*/

assign rco = ((count == 4'd11) && ent);  // Generate the rco when
// the count is 9 and ent is true.


// Below - this 'always' block is triggered by the rising clock edge
// It is executed every time the clock transitions from 0 to 1

always @ (posedge clk)
 begin
  if (clr)     // If clr = 1,
    count <= 0;     // reset 'count' to zero.
else if (enc && ent && count != 4'd11) // Else, if enc & ent = 1,
// and count isn't B(11),
    count <= count + 1;          // increment count by 1

  else if (enc && ent && count == 4'd11)   // Else, if both enables
             // are true, and count = B(11),
    count <= 0;        // return count to 0.

  else    // Else (if one or both enables false),
    count <= count; // do NOT increment.
 end
endmodule
```

Divide-by-12 simulation – focused screenshot:

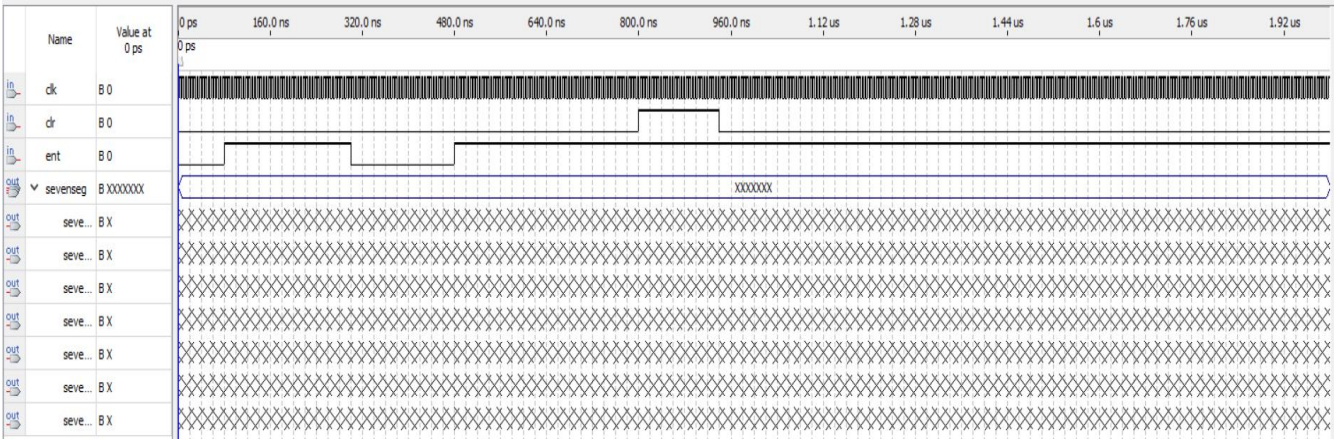


Figure 24: The simulation input waveform for Divide-by-12 simulation

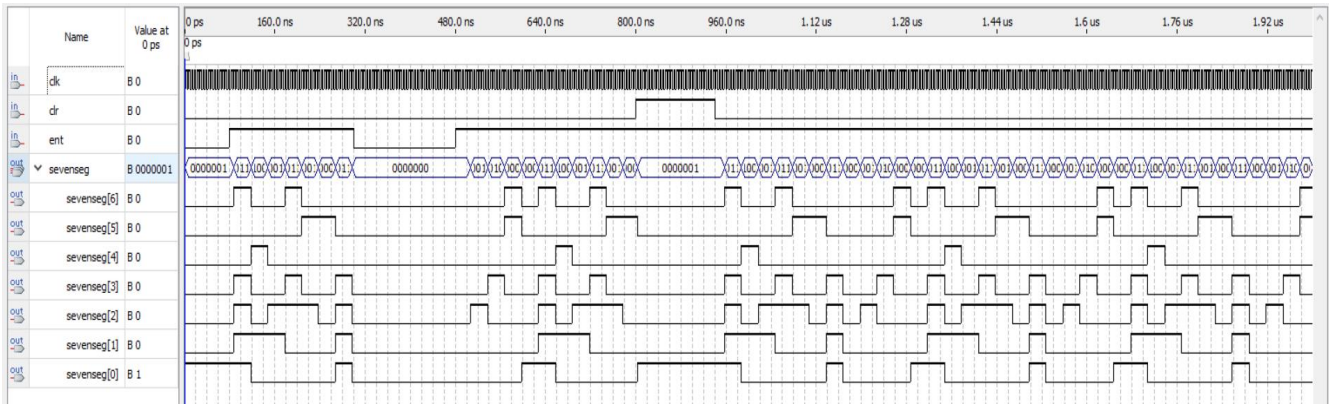


Figure 25: The output file for the Divide-by-12 simulation

Comment/Explanation for both:

For the code section, part of the code for the decade counter module needs to be adjusted to count from 0 to 12 to display the results from 0 to B. The condition of rco needs to be adjusted to 4'd11 to count to 12. Also. The two "else if" conditions also need to be adjusted to 4'd11. The rest of the code does not need to be changed. After these adjustments, the counter will be able to count to 12 and display from 0 to B.

The above experiment can be verified by simulating it. By looking at the simulated figure above, it can be seen that A (0001000) B (1100000) will appear after the number 9 (0000100). For example, the numbers A and B correspond to the time range 1.24-1.28us according to simulation waveform.

Screenshots of cascading (schematic design):

Cascading schematic – focused screenshot:

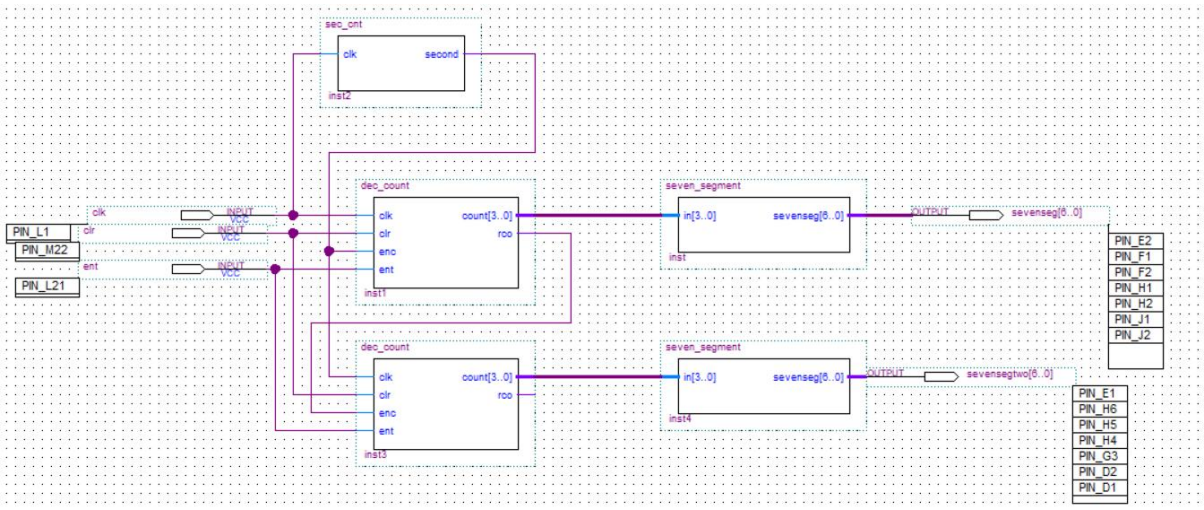


Figure 26: The schematic for cascading

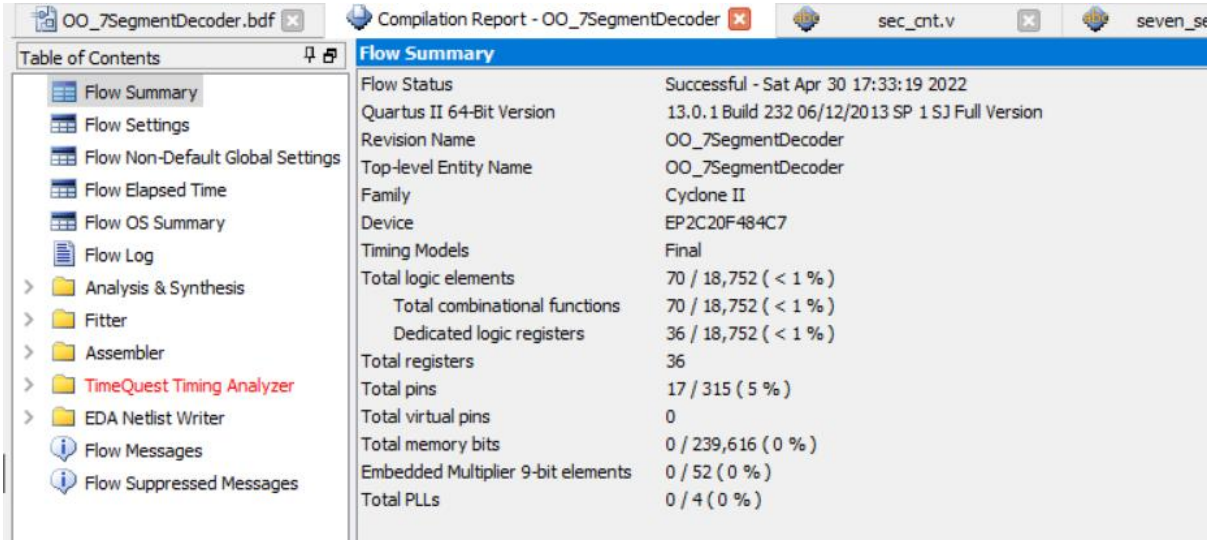


Figure 27: The compilation report for cascading simulation

Cascading simulation – focused screenshot:

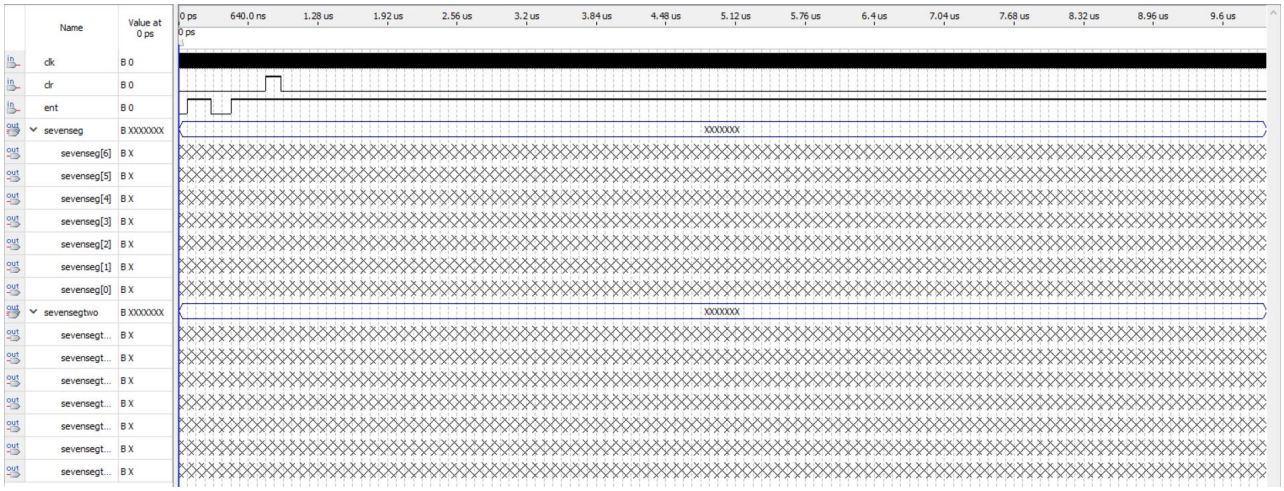


Figure 28: The simulation input waveform for cascading simulation

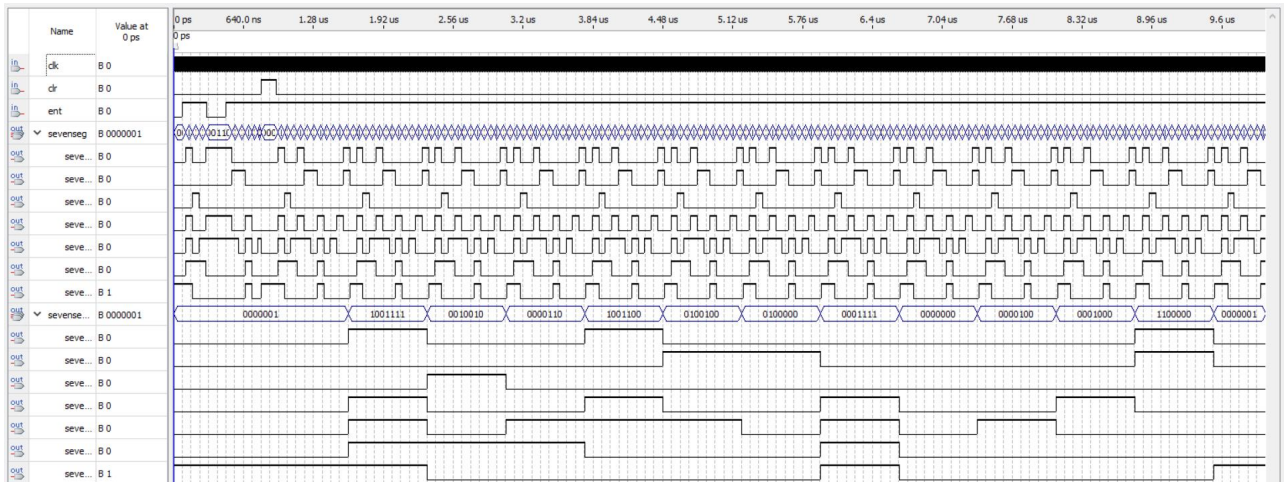


Figure 29: The output file for the cascading simulation

Comment/Explanation for both:

The schematic shows that the "clk" of the second counter is connected to the "enc" pin of the first counter. The "rco" pin of the first counter is connected to the "enc" pin of the second counter. This allows the second counter to start counting only when the first counter finishes a cycle. The "ent" inputs of both counters are connected together to obtain the same "ent" input. After checking the pin assignments, each pin of the second counter is also correctly allocated to proper position.

This time we cascade two divide-by-12 counters, as the second counter starts to change after the first count reaches B, it takes longer to simulate. Therefore, the end time is extended to 10us to show as much of the output waveforms as possible. The simulation waveform shows that the second counter changes from a 0 to a 1 at 1.6us, which is as expected. This is because the second counter only starts counting after the first counter has finished a cycle and the first counter finishes the first cycle at that time. The second counter does not affect the first counter. After cascading, the counter

can now count from 00 to BB, which corresponds to an extension of the counter's counting time range.

Photos of operation on the board

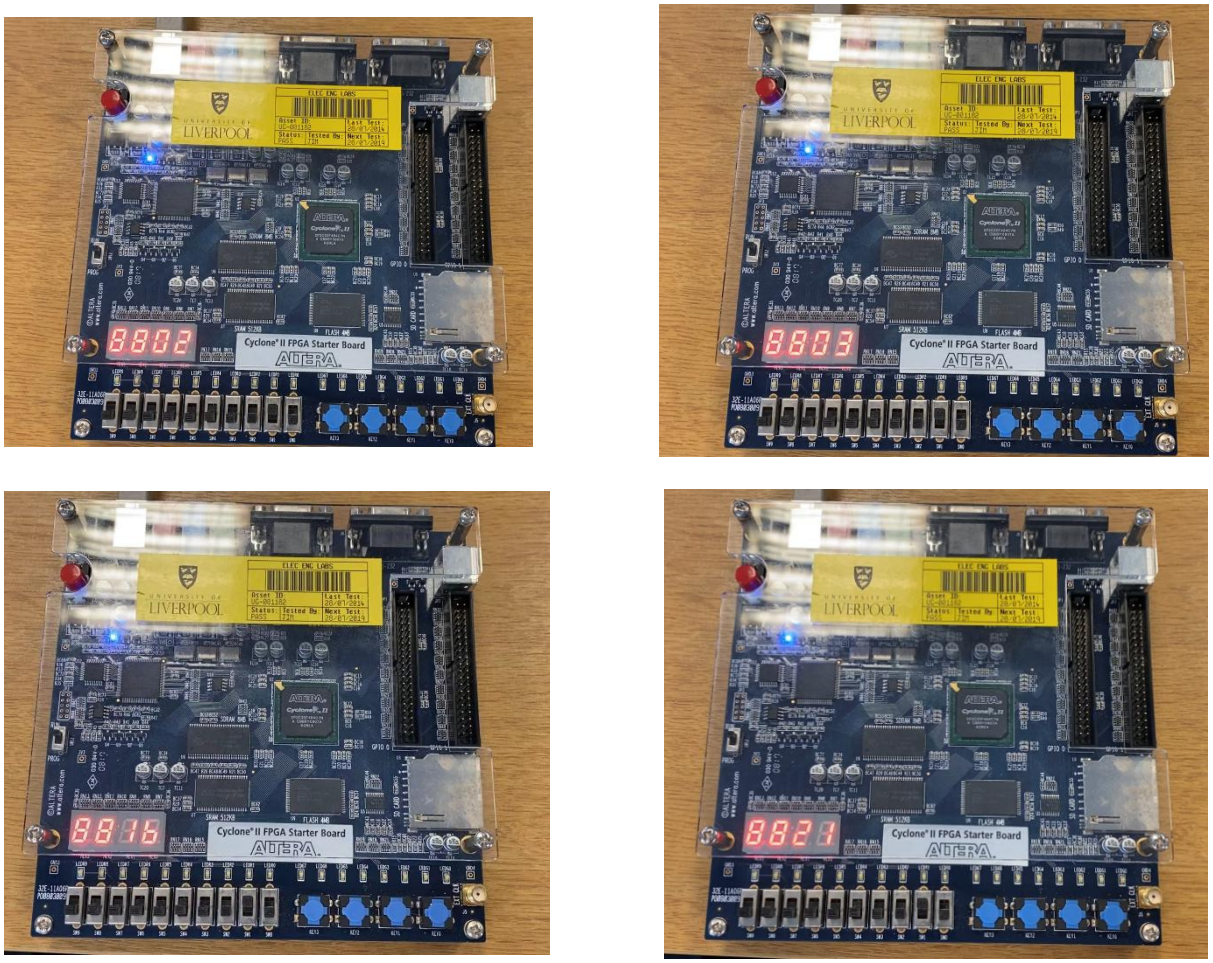


Figure 30: The photos of operation on the board

As can be seen in the figure above, a total of four states of the 7-segment display are shown. The third and fourth results prove that cascading two counters is successful, as the second counter adds 1 when the first counter ends a cycle (From 0 to B). The third figure proves that each counter can count from 0 to B.

Section 4. Design Assignments [30 Marks]

Section 4.1 Single Pulser
[10 Marks]

The Verilog code and annotated simulation

Single Pulser schematic – focused screenshot:

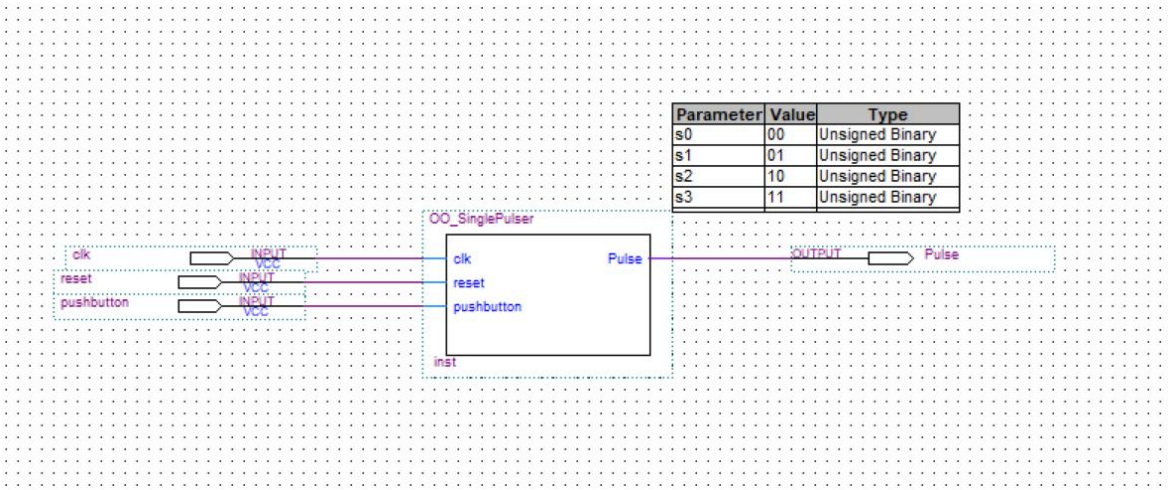


Figure 31: The schematic for single pulser

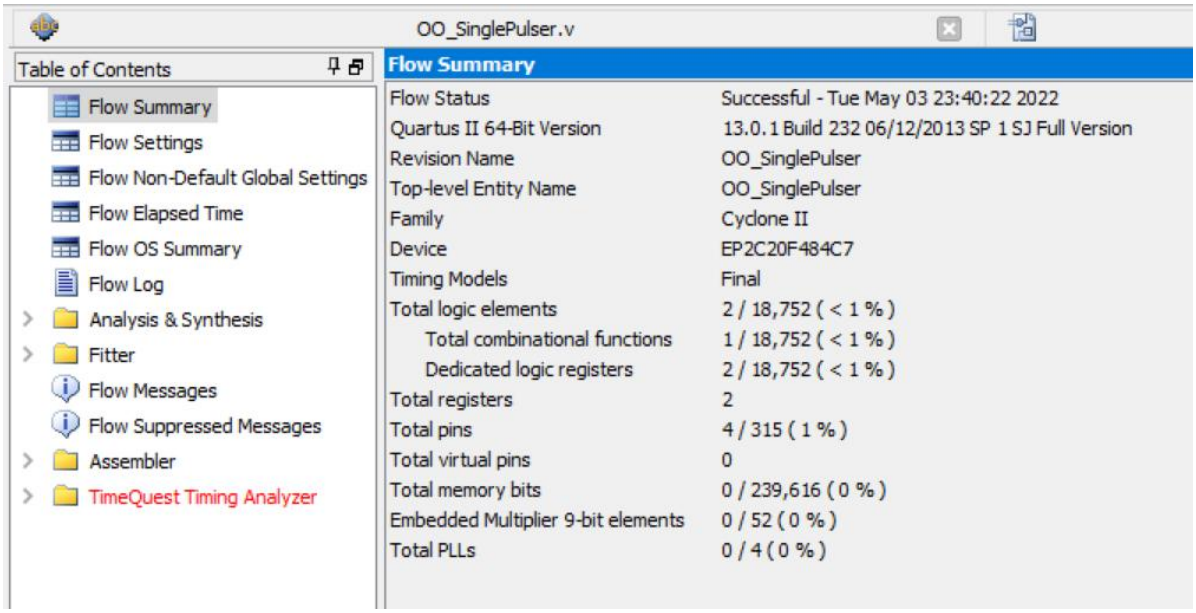


Figure 32: The compilation report for single pulser

The Verilog code (as text):

```
1.  module OO_OOSinglePulser (output reg Pulse, input clk, reset, pushbutton);
2.
3.  reg[1:0] p_state, n_state; //Present and next states
4.
5.  parameter s0=2'b00, s1=2'b01, s2=2'b10, s3=2'b11; // state variables (4 state
    s)
6.
7.  always@(posedge clk, posedge reset)  // The functions of reset
```

```
        if (reset ==1) p_state <=s0; // When reset is pressed and becomes tru
    e,
                                        //the present state returns to s0 (initi
    al state)

        else p_state <=  n_state; //The next state value becomes the present
    state

always@(p_state,pushbutton)
      begin

       Pulse=1'b0;
       n_state=p_state; // Default outputs and next state

   case(p_state)
//State s0
    s0:
     begin
       Pulse=1'b0;      //Output(Pulse)=0
         if(pushbutton) //If the push button is pressed
            n_state= s1;
         else           //If the push button is not pressed
            n_state=s0;
     end

//State s1
    s1:
     begin
       Pulse=1'b1;      //Output(Pulse)=1
         if(pushbutton) //If the push button is pressed
            n_state= s2;
         else           //If the push button is not pressed
            n_state=s0;
     end

//State s2
    s2:
     begin
        Pulse=1'b0;      //Output(Pulse)=0
          if(pushbutton) //If the push button is pressed
             n_state= s3;
          else           //If the push button is not pressed
             n_state=s0;
     end

//State s3
    s3:
     begin
        Pulse=1'b0;      //Output(Pulse)=0
          if(pushbutton) //If the push button is pressed
             n_state= s3;
          else           //If the push button is not pressed
             n_state=s0;
     end
   endcase
end
endmodule
```

Simulation – focused screenshot:

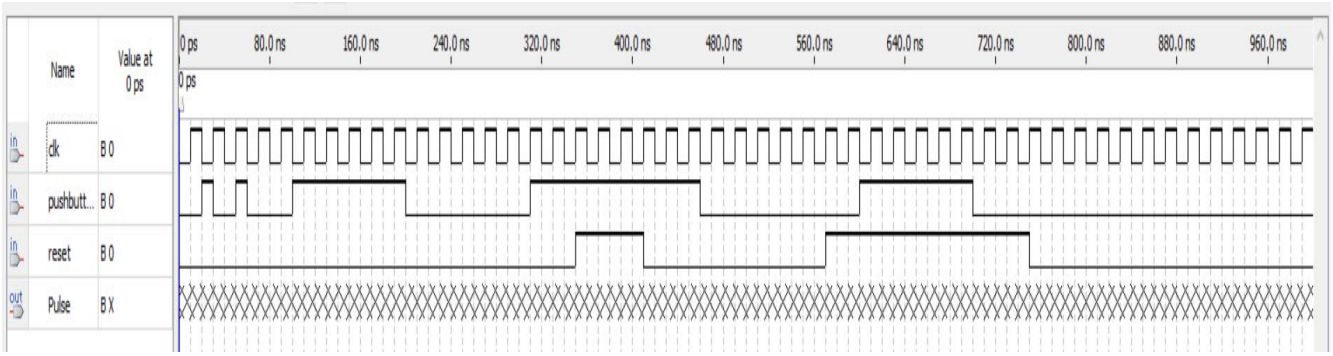


Figure 33: The simulation input waveform for the single pulser

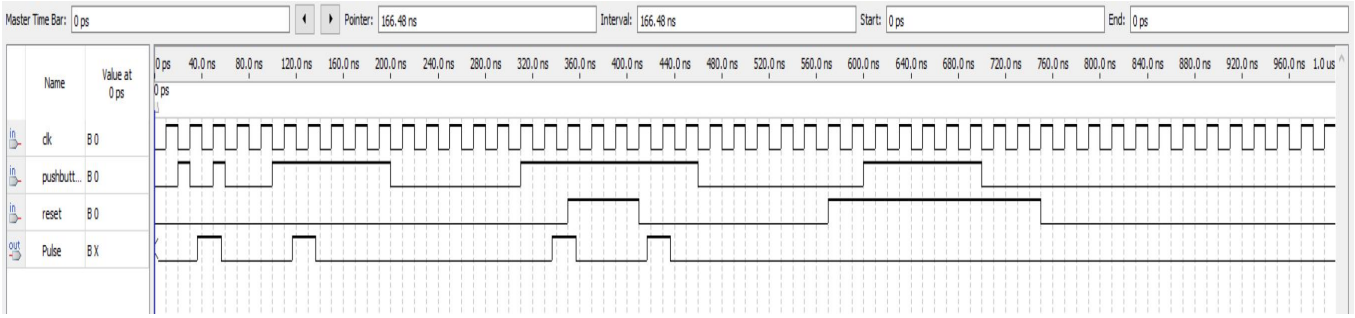


Figure 34: The simulation result for the single pulser

Comment/Explanation:

In this simulation, four states are designed s0, s1, s2, s3. s2 is the only state where the output Pulse is high (1), the other states will be low, so that only one output is generated for 1 clock period, even when the input pulse is much longer than this. In the design, a push button is selected as the input signal, and the output goes high when the push button is pressed and stays low when it is not pressed. Pulse is the output in the design.

As can be seen from the figure of the simulation results, when the time is 120-200ns, although the input (push button) is always high, there is only one output for only 1 clock period, which meets the requirements of the task. This is because the input time is set well beyond the time of a clock, but there is only one output for only 1 clock period. When reset is pressed, it can be noticed that the output is reset to low level (initial state), which is as expected. This can be seen in the last two parts of the simulation figure. When the time of the reset is greater than the time of the input, all outputs will be low. When the time of the reset is less than the time of the input, the reset eliminates the high output for this time.

Also, when the time is 20ns-30ns, the input is less than a clock, but there is still an output, which is a limitation of the design, but at 50-60ns, there is no output, even

though there is an input less than a clock. The difference between the two cases is whether the pulse input is on the rising or falling edge of the clk. For the rising edge there is no output despite a short time input. For the falling edge, the output will be high for 1 clock, despite the short time of input. This is to be expected because of the effect of active edge of the clock and the setup and hold time.

Section 4.2 Pushbutton Enabled Counter [10 Marks]

The schematic design (with pin assignments) and annotated simulation

Schematic with pin assignments – focused screenshot:

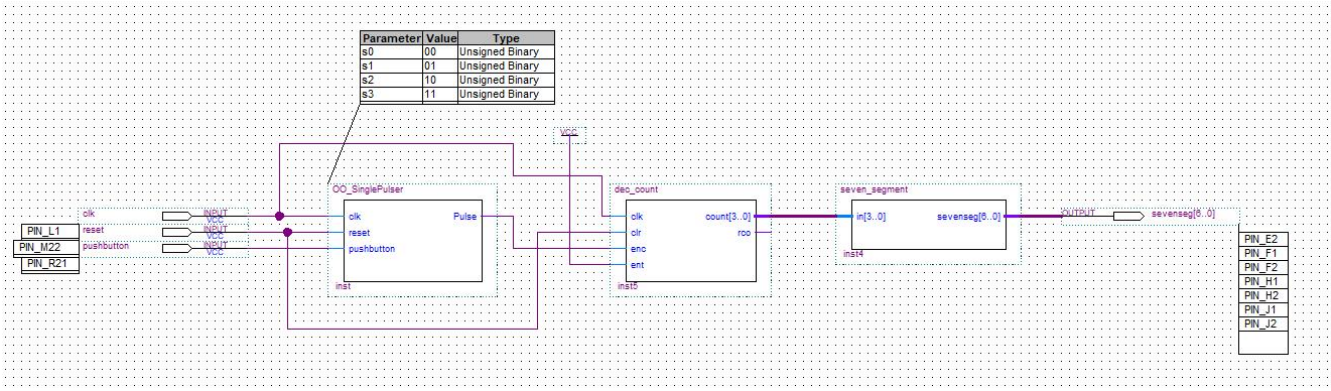


Figure 35: The schematic for the enabled counter with pin assignment

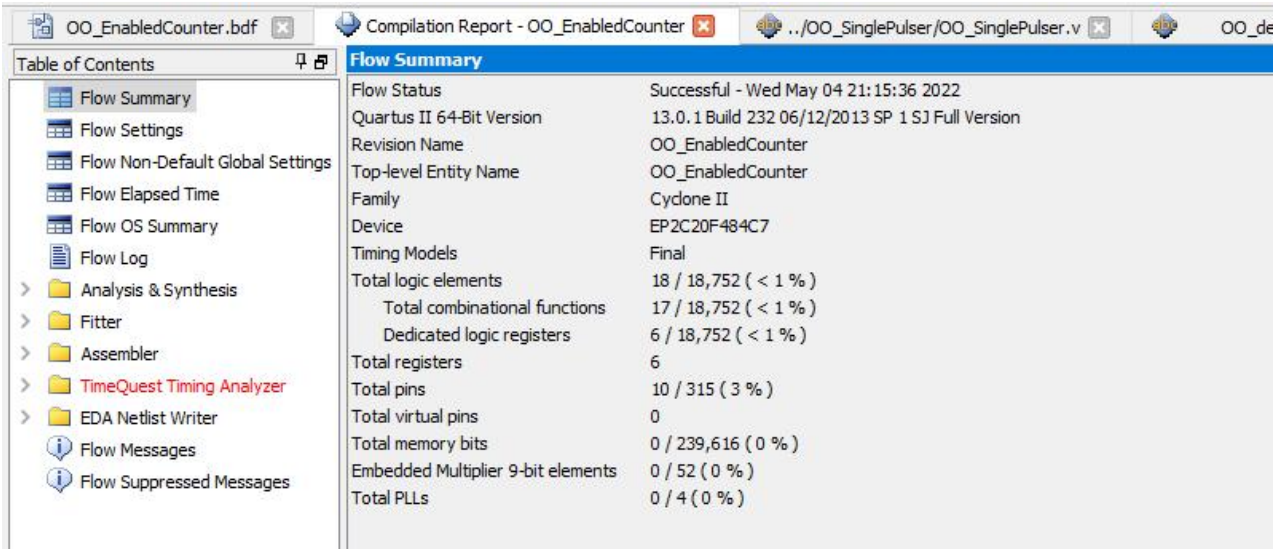


Figure 36: The compilation for the enabled counter

Annotated simulation – focused screenshot:

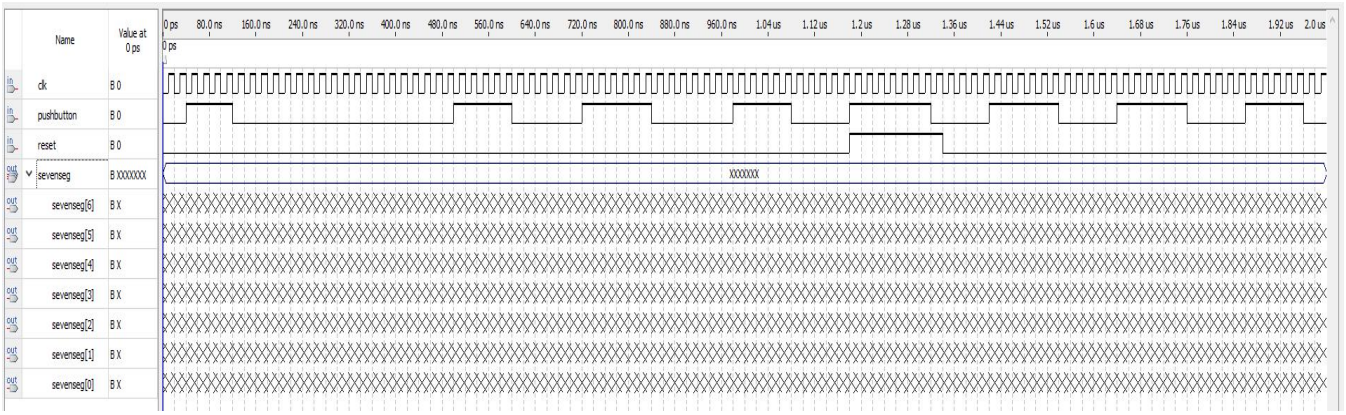


Figure 37: The simulation input waveform for the enabled counter

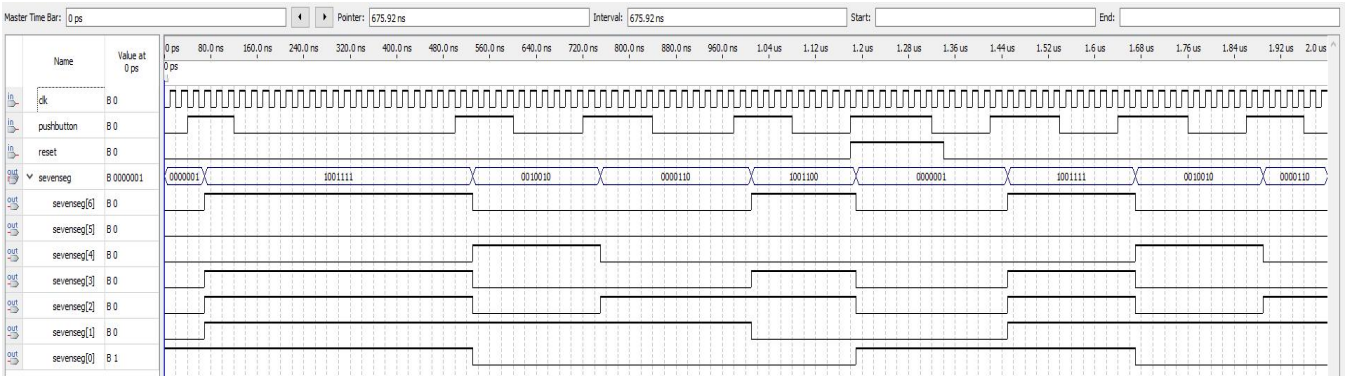


Figure 38: The simulation result for the enabled counter

Comment/Explanation for both:

In this design, the "pulse" output of the single pulser is connected to the "enc" input of the dec_counter. The final output is implemented by the seven segment module. Since the last digit of my student ID number is 9 and the binary digit of 9 is 1001. Therefore, I use key[1] as the pin of pushbutton according to the requirements of the experiment. The pin assignments for the other parts are the same as the previous parts. So, every time key[1] is pressed, the number of seven segment will increase by 1. If reset is selected, the number of output will return to show 0.

The end time of the output is set to 2us, and the period of the clock is set to 20ns according to the requirements. As can be seen from the output figure, whenever the pushbutton is high for a longer period of time, the number will increase by 1. When reset is high, the output number goes back to 0. This is in line with the experimental expectation.

Section 4.3 Adder/Subtractor Logic Design [10 Marks]

The schematic design (with pin assignments) and annotated simulation

Schematic with pin assignments – focused screenshot:

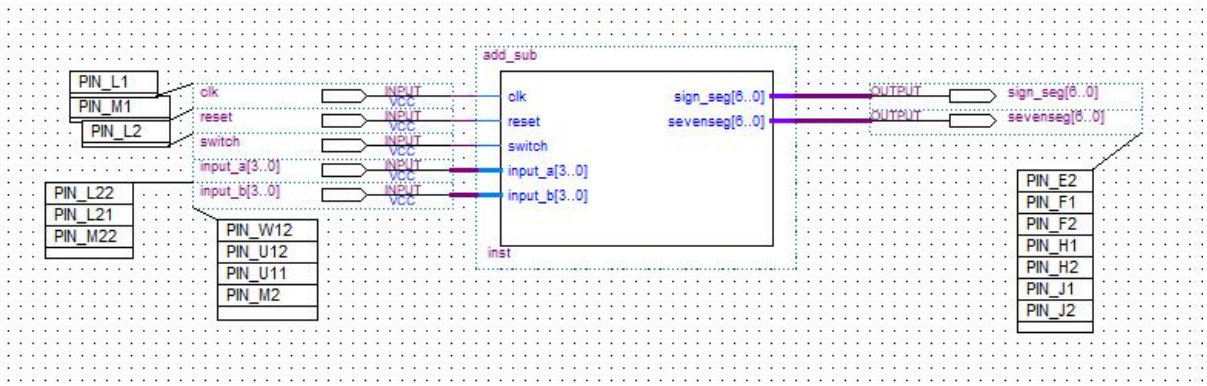


Figure 39: The schematic for the Adder/Subtractor Logic Design

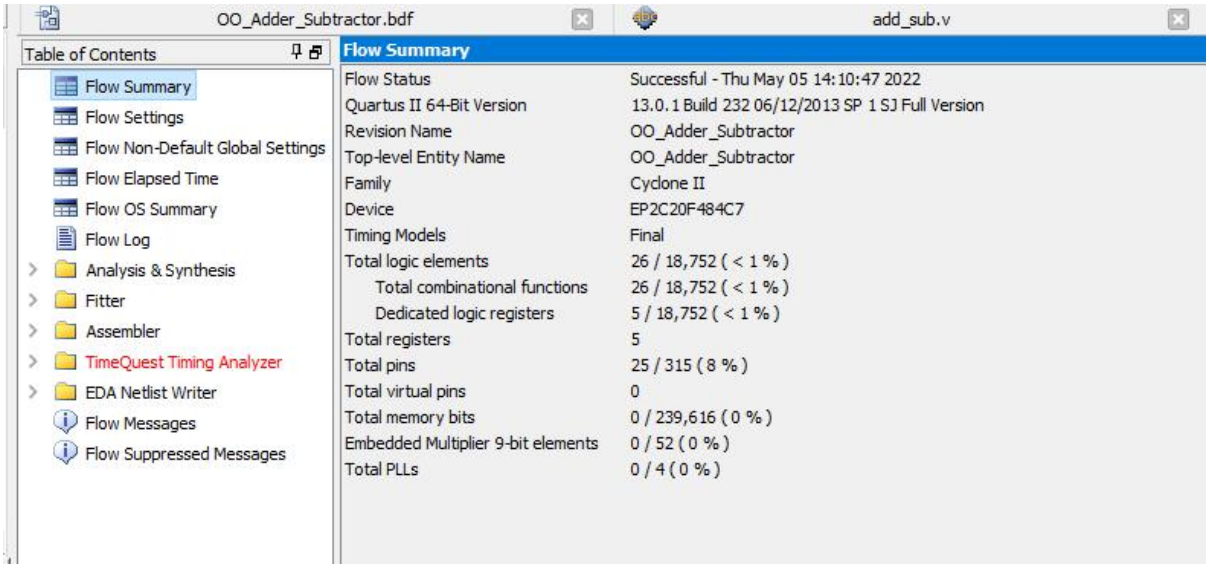


Figure 40: The compilation report for the Adder/Subtractor Logic Design

```
1.  module      add_sub(
2.      input           clk,
3.      input           reset,
4.      input           switch,             //Switch additive and subtractive, switch=1:
    additive; switch=0:subtractive, which need to connect to Toggle switch
5.      input   [3:0]   input_a,            //Input values a,b
6.      input      [3:0]    input_b,
7.      output  reg[6:0]    sign_seg,       //Display sign bit, display 1 for negative
    value, display 0 for positive value
8.      output  reg[6:0]    sevenseg        //Specific calculated values
9.      );
10.
11.
12. reg [4:0] c_reg;                //Final value of the final calculation result
13. always@(posedge clk or negedge reset) begin  //Reset seven segment display when res
    et becomes low level
```

```
14.     if(!reset)
15.         c_reg <= 5'd0;           //Decimal 0 (5 bits)
16.     else if(switch)
17.           c_reg <= {input_a[3],input_a[3:0]} + {input_b[3],input_b[3:0]};
      //additive
18.     else
19.         c_reg <= {input_a[3],input_a[3:0]} + (~{input_b[3],input_b[3:0]}+1);
20.   //subtractive
21. end
22.
23. always@(*) begin
24.     case(c_reg[4])
25.         1'b0:sign_seg = 7'b0000001; //If the calculation result is positive, the se
    venseg shows 0
26.         1'b1:sign_seg = 7'b1001111; //If the calculation result is negative, the se
    venseg shows 1
27.     endcase
28. end
29.
30. always@(*) begin
31.     if(c_reg[4]) begin
32.         case((~{c_reg[3:0]})+1'd1)  //If the sign bit is negative, it needs to be i
    nverted plus one
33.           // abcd                abcdefg
34.             4'h0: sevenseg = 7'b0000001;
35.             4'h1: sevenseg = 7'b1001111;
36.             4'h2: sevenseg = 7'b0010010;
37.             4'h3: sevenseg = 7'b0000110;
38.             4'h4: sevenseg = 7'b1001100;
39.             4'h5: sevenseg = 7'b0100100;
40.             4'h6: sevenseg = 7'b0100000;
41.             4'h7: sevenseg = 7'b0001111;
42.             4'h8: sevenseg = 7'b0000000;
43.             4'h9: sevenseg = 7'b0000100;
44.             4'hA: sevenseg = 7'b0001000;
45.             4'hB: sevenseg = 7'b1100000;
46.             4'hC: sevenseg = 7'b0110001;
47.             4'hD: sevenseg = 7'b1000010;
48.             4'hE: sevenseg = 7'b0110000;
49.             4'hF: sevenseg = 7'b0111000;
50.         endcase
51.     end
52.     else begin
53.         case(c_reg[3:0]) // If the sign digit is positive, the direct decision will
     be made
54.           // abcd                abcdefg
55.            4'h0: sevenseg = 7'b0000001;
56.             4'h1: sevenseg = 7'b1001111;
57.             4'h2: sevenseg = 7'b0010010;
58.             4'h3: sevenseg = 7'b0000110;
59.             4'h4: sevenseg = 7'b1001100;
60.             4'h5: sevenseg = 7'b0100100;
61.             4'h6: sevenseg = 7'b0100000;
62.             4'h7: sevenseg = 7'b0001111;
63.             4'h8: sevenseg = 7'b0000000;
64.             4'h9: sevenseg = 7'b0000100;
65.             4'hA: sevenseg = 7'b0001000;
66.             4'hB: sevenseg = 7'b1100000;
67.             4'hC: sevenseg = 7'b0110001;
68.             4'hD: sevenseg = 7'b1000010;
69.             4'hE: sevenseg = 7'b0110000;
70.             4'hF: sevenseg = 7'b0111000;
71.         endcase
72.     end
73. end
74. endmodule
```

Annotated simulation – focused screenshot:

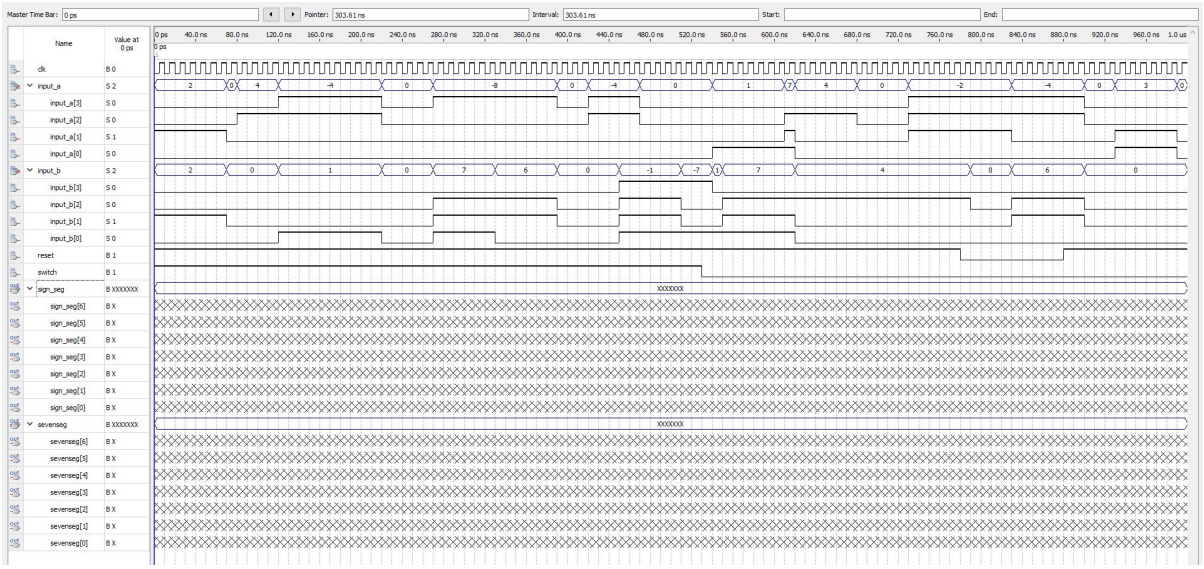


Figure 41: The simulation input waveform for the Adder/Subtractor

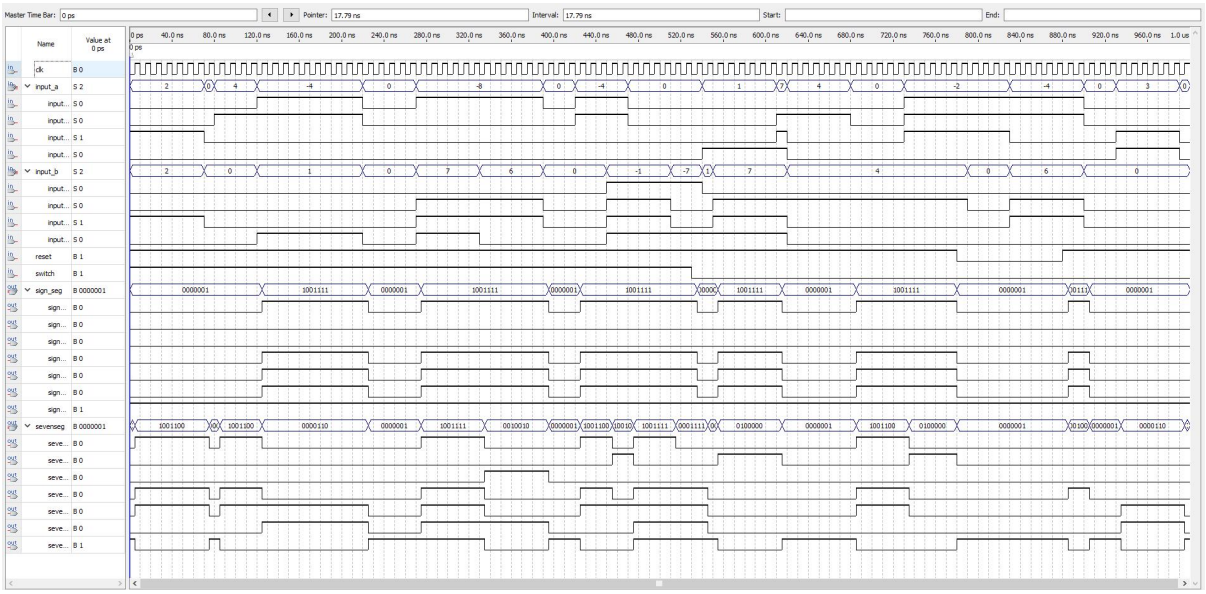


Figure 42: The simulation result for the Adder/Subtractor

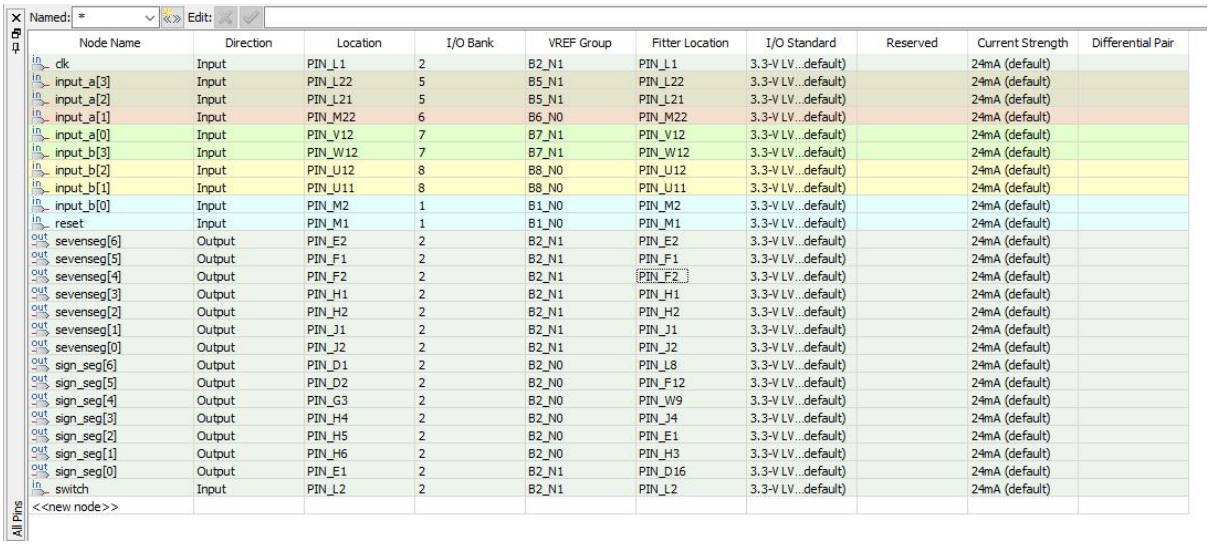
Comment/Explanation for both:

The first seven segment display is used to display the different numbers and the second seven segment display is used to display the sign bit, which displays 1 when the calculated value is negative and 0 when the calculated result is positive. For convenience, I put both seven segments into one module.

For the input method, I choose to use the toggle switches, arranging the two 4-bit inputs with different toggle switches, the exact pin assignment can be seen in the

following figure. In addition, I have also assigned the “reset” and “switch” pins to toggle switches. The pin corresponding to the reset is PIN_M1 and the pin corresponding to the switch is PIN_L2.

For the outputs, as in the previous tasks, different pins are assigned to the two seven segment displays in the following way:



Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
clk	Input	PIN_L1	2	B2_N1	PIN_L1	3.3-V LV...default)		24mA (default)	
input_a[3]	Input	PIN_L22	5	B5_N1	PIN_L22	3.3-V LV...default)		24mA (default)	
input_a[2]	Input	PIN_L21	5	B5_N1	PIN_L21	3.3-V LV...default)		24mA (default)	
input_a[1]	Input	PIN_M22	6	B6_N0	PIN_M22	3.3-V LV...default)		24mA (default)	
input_a[0]	Input	PIN_V12	7	B7_N1	PIN_V12	3.3-V LV...default)		24mA (default)	
input_b[3]	Input	PIN_W12	7	B7_N1	PIN_W12	3.3-V LV...default)		24mA (default)	
input_b[2]	Input	PIN_U12	8	B8_N0	PIN_U12	3.3-V LV...default)		24mA (default)	
input_b[1]	Input	PIN_U11	8	B8_N0	PIN_U11	3.3-V LV...default)		24mA (default)	
input_b[0]	Input	PIN_M2	1	B1_N0	PIN_M2	3.3-V LV...default)		24mA (default)	
reset	Input	PIN_M1	1	B1_N0	PIN_M1	3.3-V LV...default)		24mA (default)	
sevenseg[6]	Output	PIN_E2	2	B2_N1	PIN_E2	3.3-V LV...default)		24mA (default)	
sevenseg[5]	Output	PIN_F1	2	B2_N1	PIN_F1	3.3-V LV...default)		24mA (default)	
sevenseg[4]	Output	PIN_F2	2	B2_N1	PIN_F2	3.3-V LV...default)		24mA (default)	
sevenseg[3]	Output	PIN_H1	2	B2_N1	PIN_H1	3.3-V LV...default)		24mA (default)	
sevenseg[2]	Output	PIN_H2	2	B2_N1	PIN_H2	3.3-V LV...default)		24mA (default)	
sevenseg[1]	Output	PIN_J1	2	B2_N1	PIN_J1	3.3-V LV...default)		24mA (default)	
sevenseg[0]	Output	PIN_J2	2	B2_N1	PIN_J2	3.3-V LV...default)		24mA (default)	
sign_seg[6]	Output	PIN_D1	2	B2_N0	PIN_L8	3.3-V LV...default)		24mA (default)	
sign_seg[5]	Output	PIN_D2	2	B2_N0	PIN_F12	3.3-V LV...default)		24mA (default)	
sign_seg[4]	Output	PIN_G3	2	B2_N0	PIN_W9	3.3-V LV...default)		24mA (default)	
sign_seg[3]	Output	PIN_H4	2	B2_N0	PIN_J4	3.3-V LV...default)		24mA (default)	
sign_seg[2]	Output	PIN_H5	2	B2_N0	PIN_E1	3.3-V LV...default)		24mA (default)	
sign_seg[1]	Output	PIN_H6	2	B2_N0	PIN_H3	3.3-V LV...default)		24mA (default)	
sign_seg[0]	Output	PIN_E1	2	B2_N1	PIN_D16	3.3-V LV...default)		24mA (default)	
switch	Input	PIN_L2	2	B2_N1	PIN_L2	3.3-V LV...default)		24mA (default)	
<<new node>>									

All Pins

Figure 43: The pin allocations for the inputs and outputs

As can be seen from the code, I have defined two 4-bit inputs (input_a and input_b). In my design, I perform addition when “switch” is in high level and subtraction when “switch” is in low level. In addition, “reset” is performed when “reset” is in low level to reset the number displayed on the seven segment to zero. When the result of the calculation is a negative number, the sign bit will display 1, corresponding to the code 1001111. When the result of the calculation is a positive number, the sign bit will display a 0, corresponding to the code 0000001.

From the simulation figure, for addition, when both inputs are 2 (0-70ns), the result of the sign bit is 0000001 (0) and the result of the seven segment showing the number is 1001100 (4), which is as expected. In addition, when the input values are -4 and 1 respectively and the addition is performed (120-220ns), it can be seen that the result of the sign bit is 1001111 (1) and the result of the seven segment of the display digit is 0000110 (3), so the final result of the calculation is -3. This is correct.

When the subtraction is performed (switch goes low, 530-1000ns), it can be seen from the figure that when the inputs are 1 and 7 (550-610ns), the result of the sign bit is

1001111 (1) and the result of the seven segment of the display number is 0100000 (6), so the final calculation results in -6. This is correct. For another example, when the input is 3 and 0 (930-1000ns), the result of the sign bit is 0000001 (0) and the result of the seven segment of the display number is 0000110 (3), so the final result is 3, which is also correct. For the operation of "reset", when the "reset" is in low level (780-880ns), the result of the sign bit is 0000001 (0) and the result of the seven segment of the display number is also 0000001 (0), which shows that the operation of "reset" is successful.

See below for Appendix of full-screen screenshots.

Appendix

Section 2.3 Drawing Your Schematic

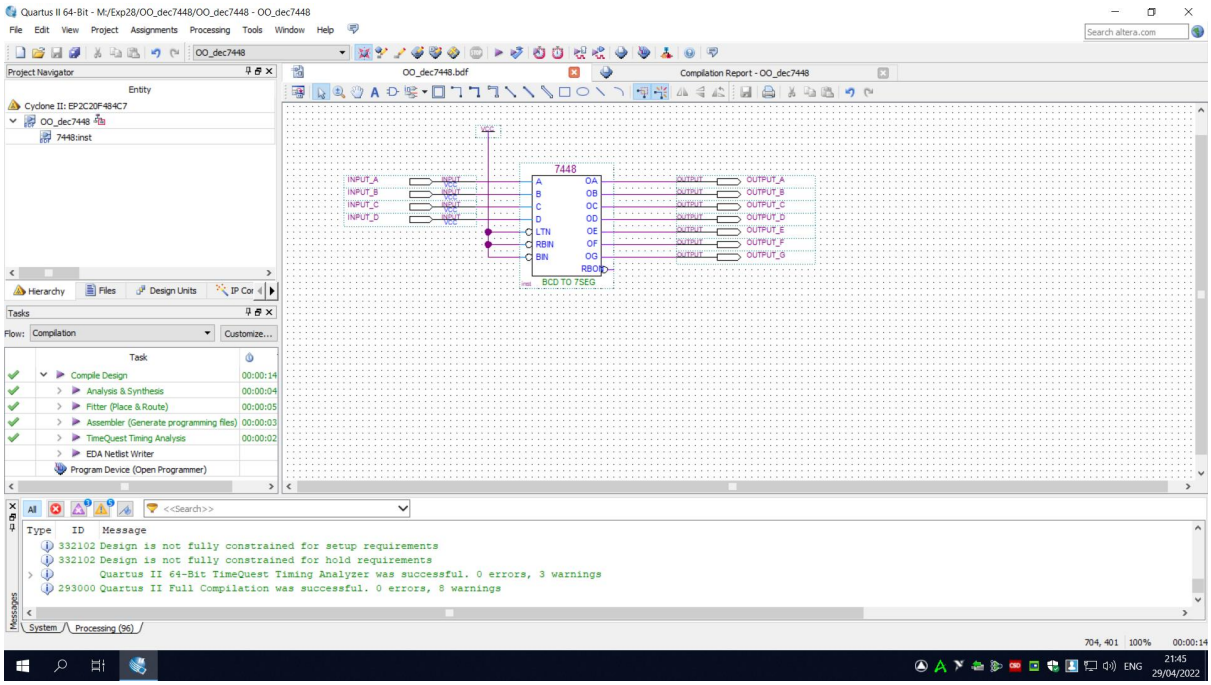


Figure 44: The full-screen screenshot for 7448

Section 2.4 Waveform Entry for Simulation

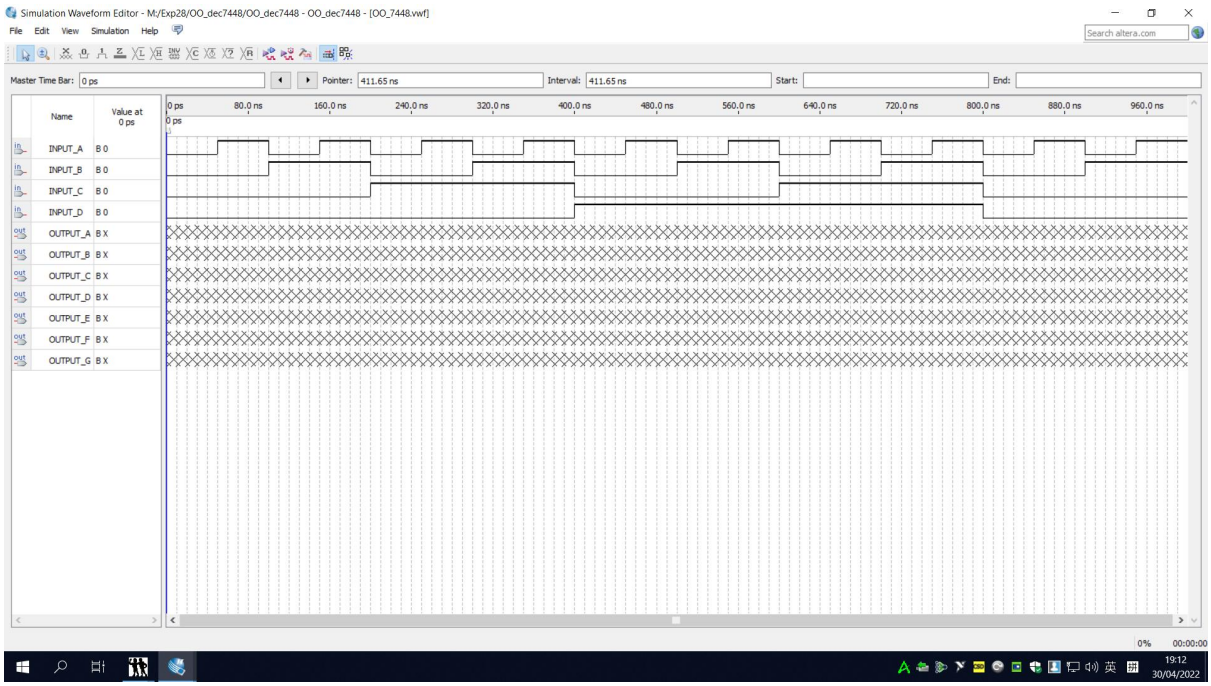


Figure 45: The full-screenshot for the input waveform of 7448

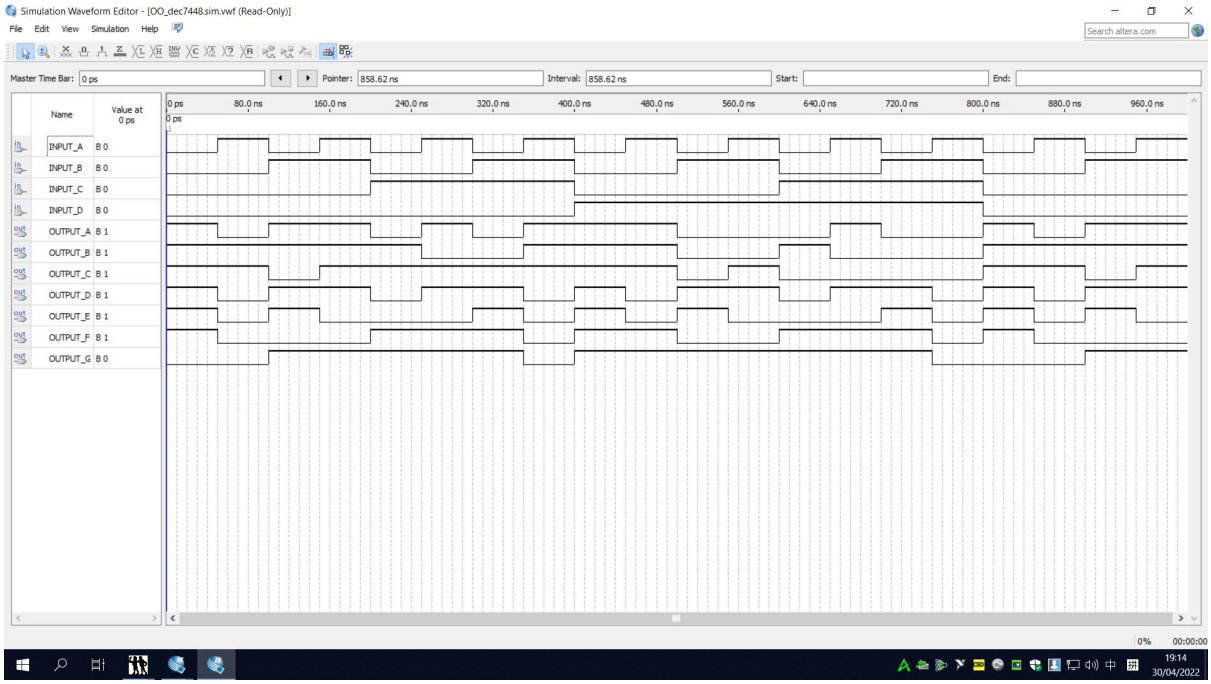


Figure 46: The full-screen screenshot for the output file of 7448

Section 2.5 Design Project based on Verilog Code: Decoder

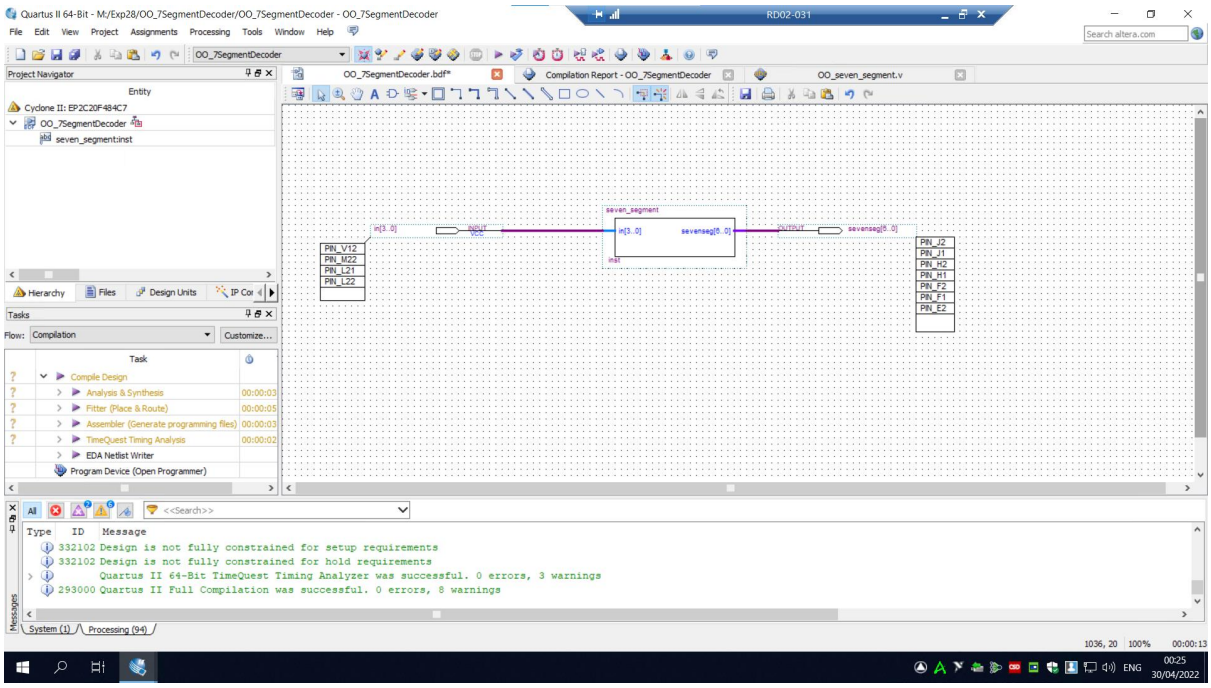


Figure 47: The full-screen screenshot for 2.5

Section 2.6 Input Simulation for Decoder

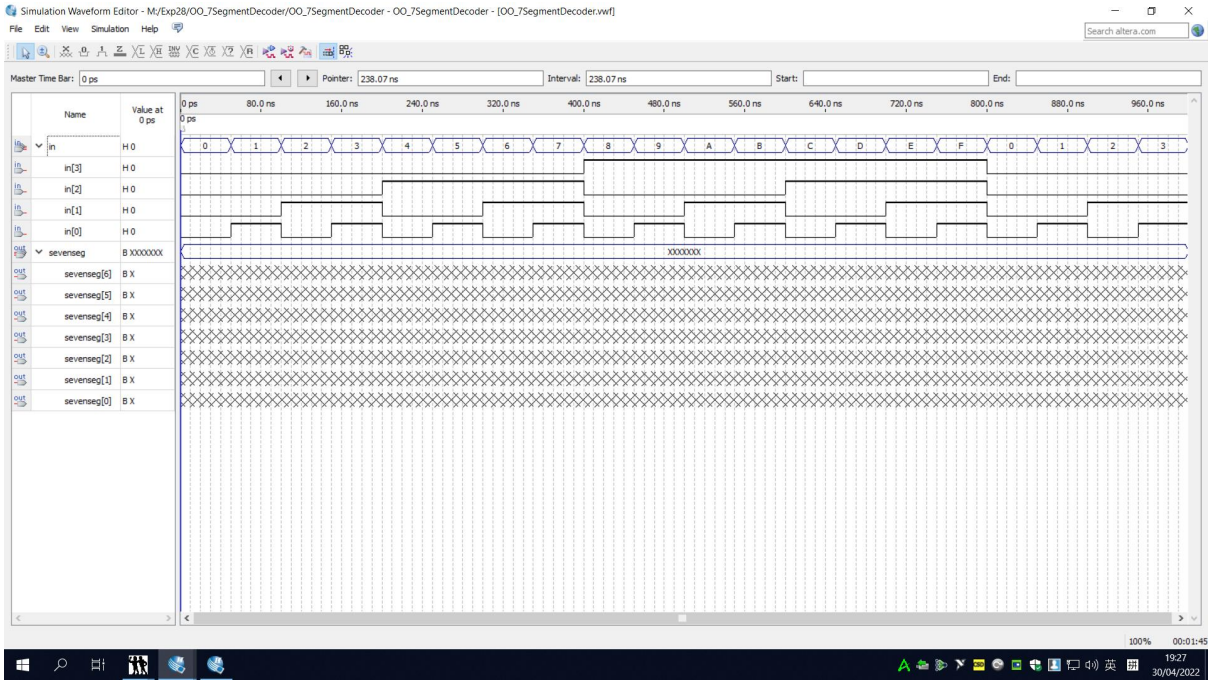


Figure 48: The full-screen screenshot for the input waveform of decoder

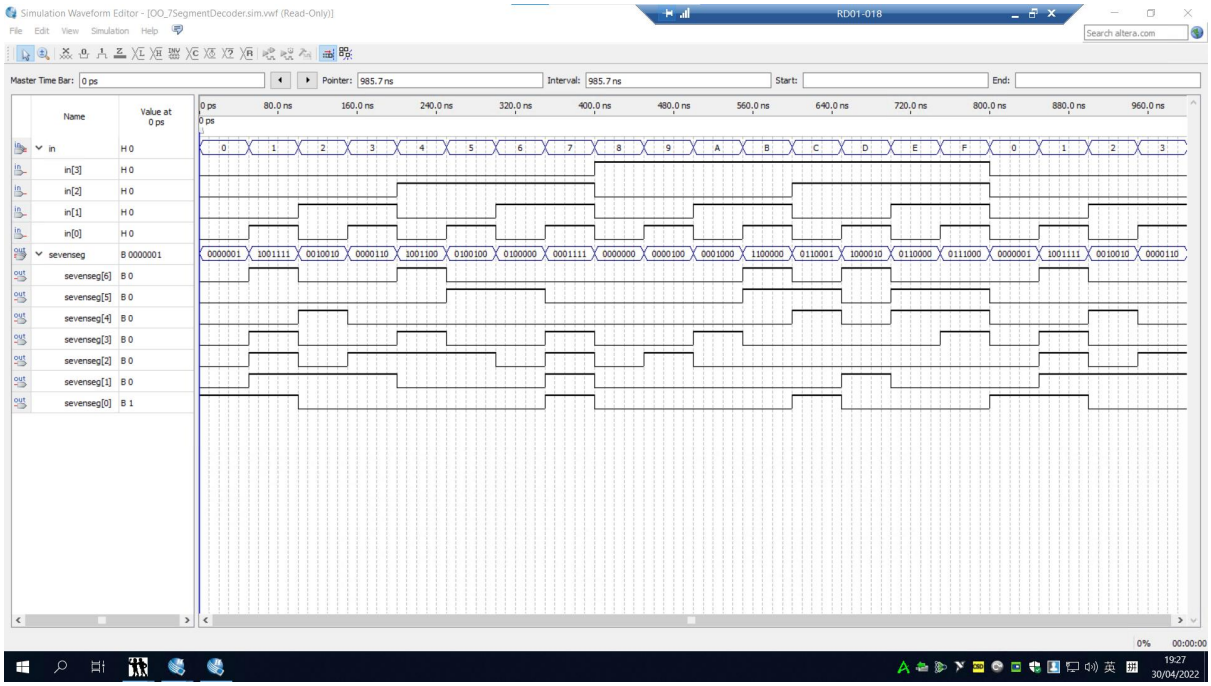


Figure 49: The full-screenshot for the output file of decoder

Section 3.1 Counter design in Verilog [8 marks]

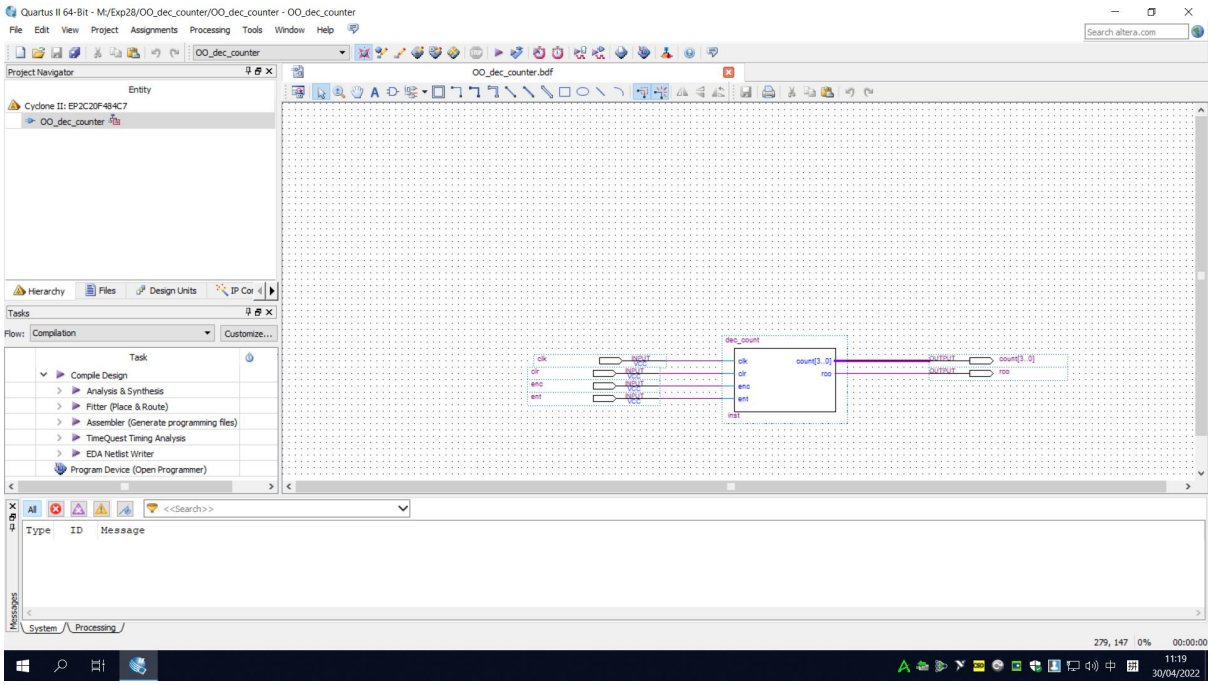


Figure 50: The full-screen screenshot for decade counter

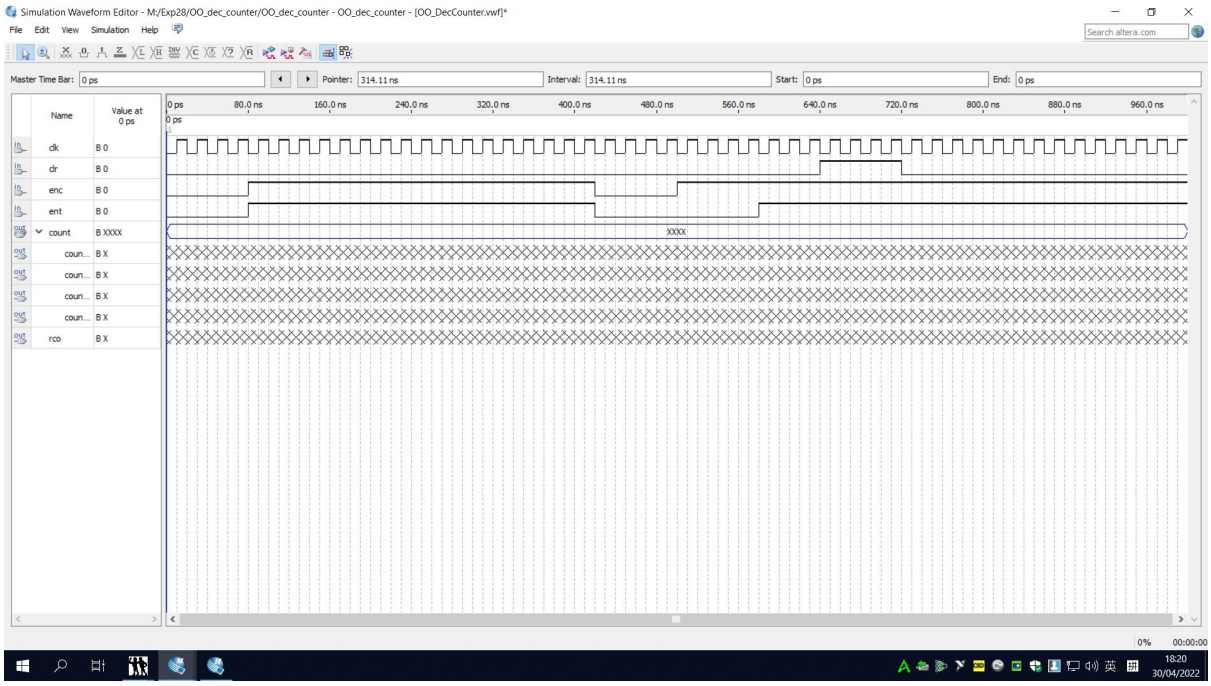


Figure 51: The full-screenshot for the input waveform of the decade counter

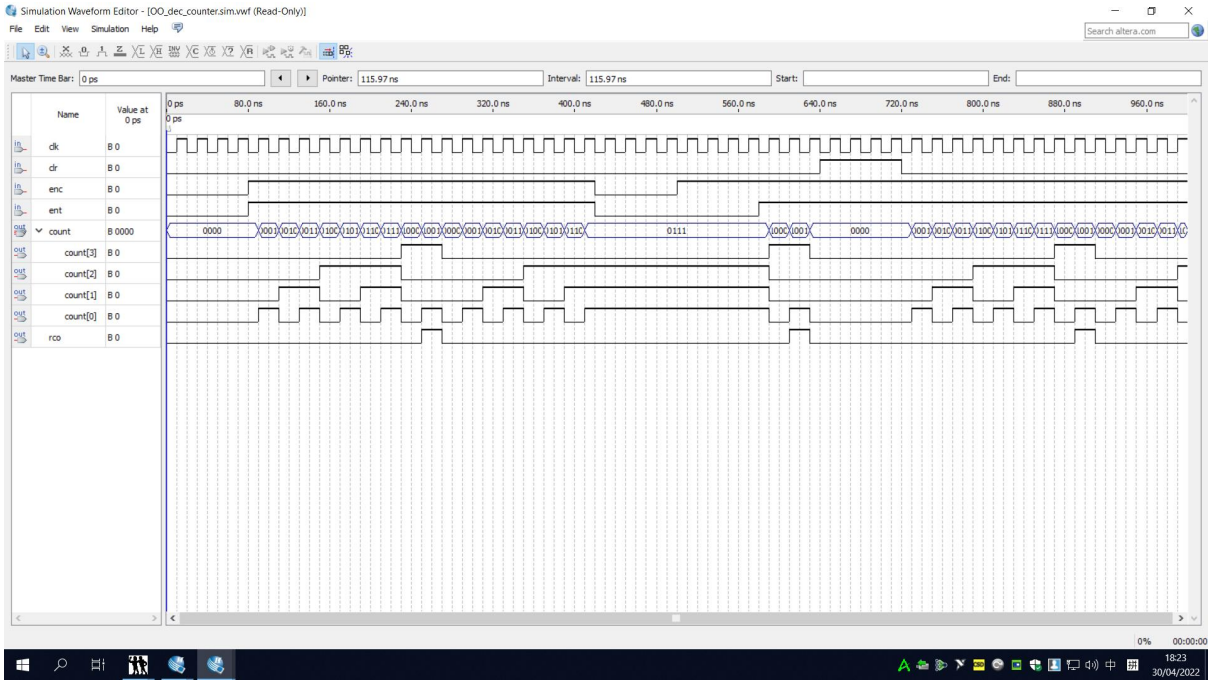


Figure 52: The full-screen waveform screenshot for decade counter

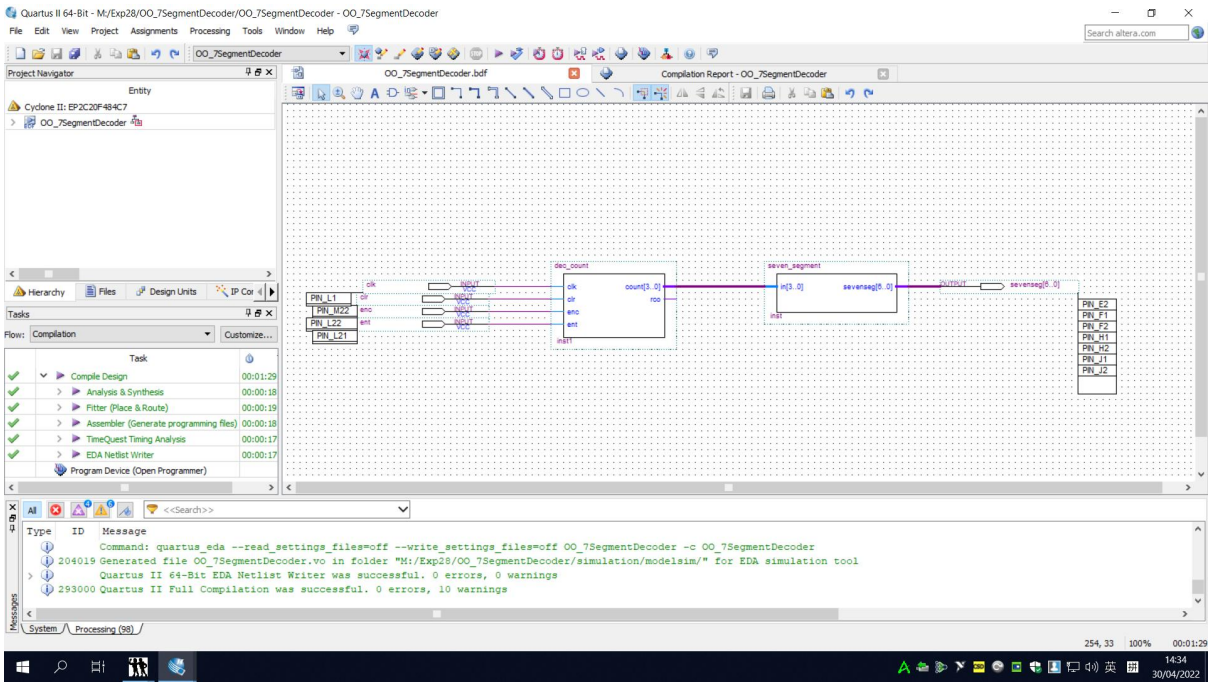


Figure 53:The full-screenshot for the decade counter with 7-segment decoder

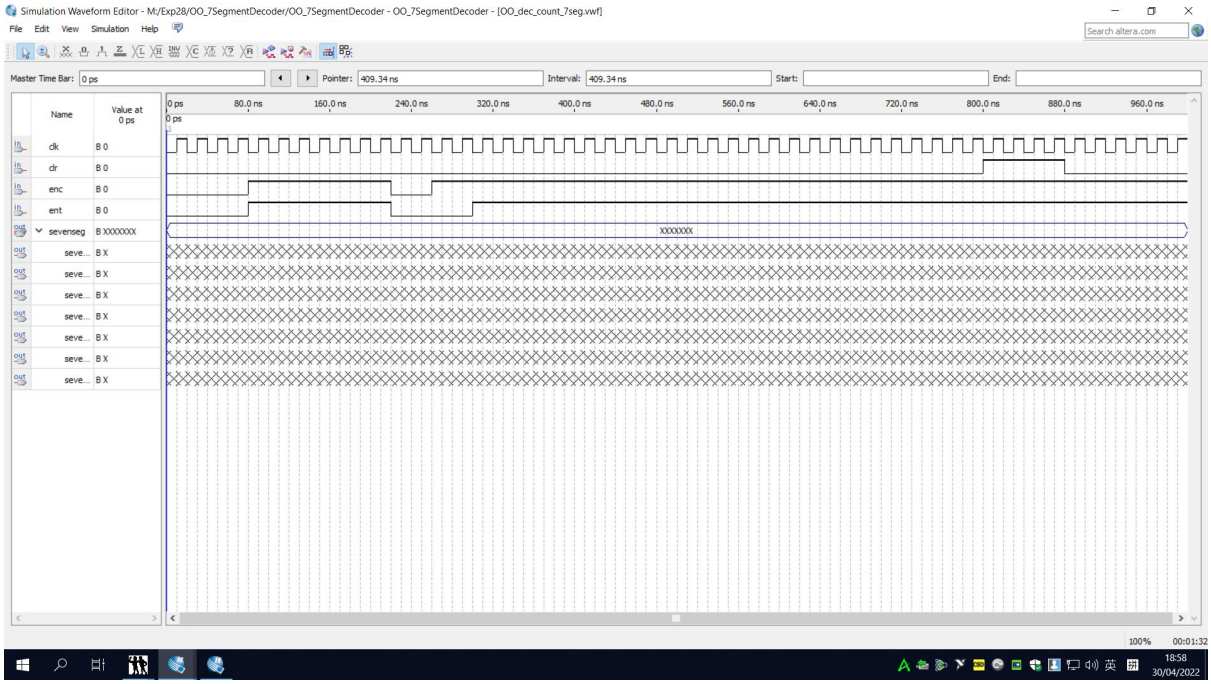


Figure 54: The full-screenshot for the input waveform of the decade counter with 7-segment decoder

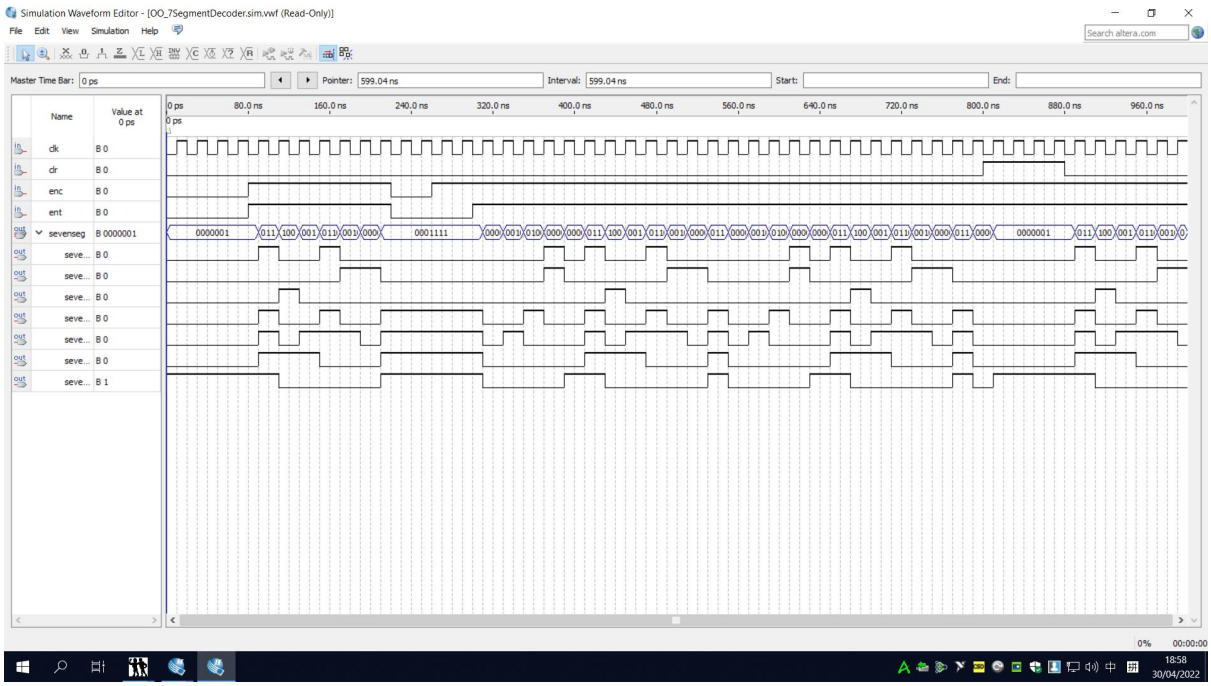


Figure 55: The full-screen waveform screenshot for decade counter with 7-segment decoder

Section 3.2 Once-per-second counter

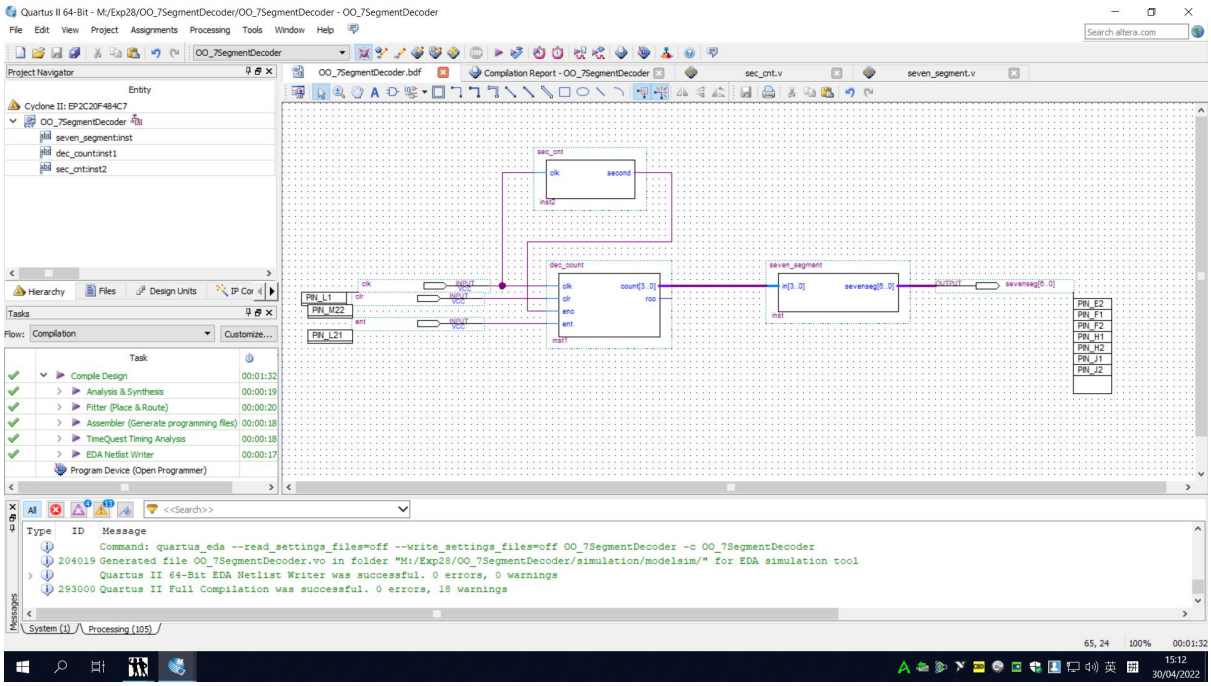


Figure 56: The full-screenshot for the Once-per-second counter

Section 3.2 continues on the next page

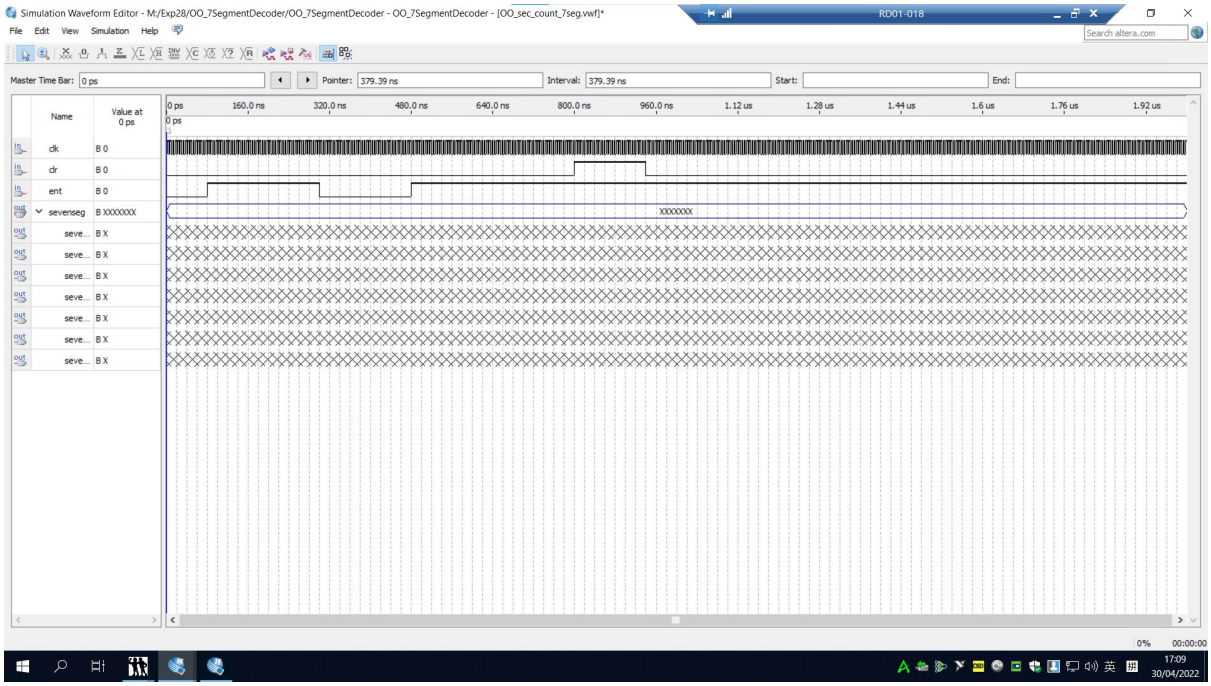


Figure 57: The full-screenshot for the waveform the Once-per-second counter

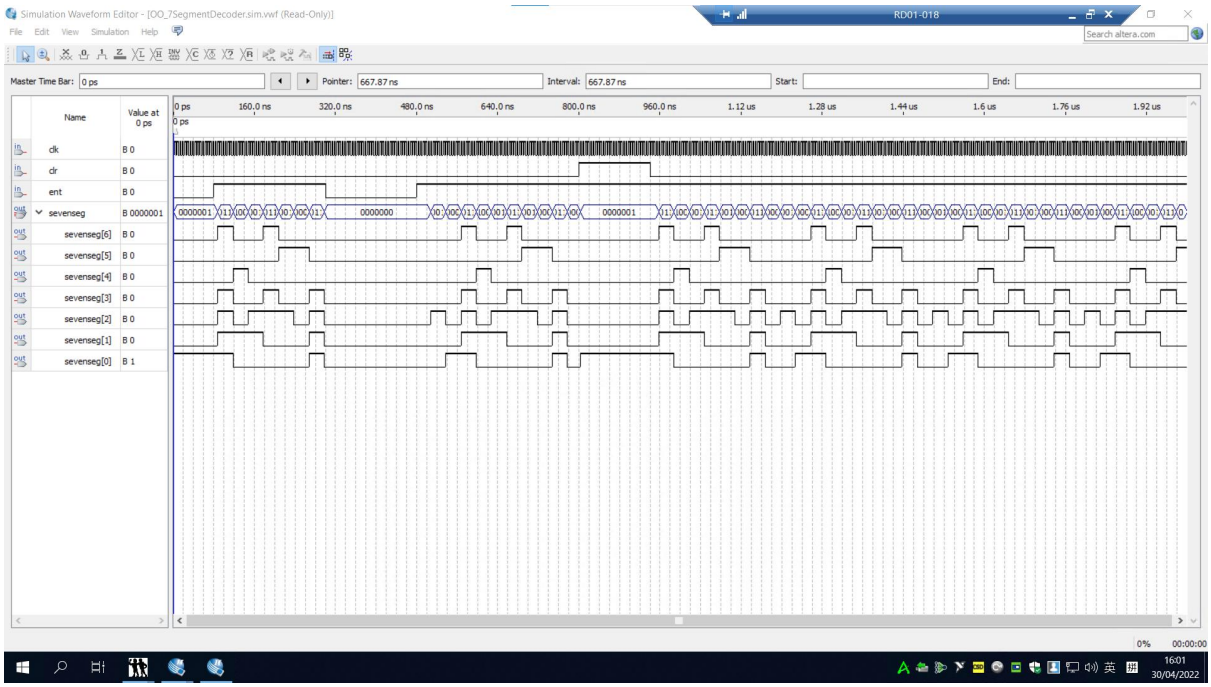


Figure 58: The full-screenshot output file screenshot for the Once-per-second counter

(2, 3) The schematic and simulation of the divide-by-12, and of the cascading of two divide-by-12 counters

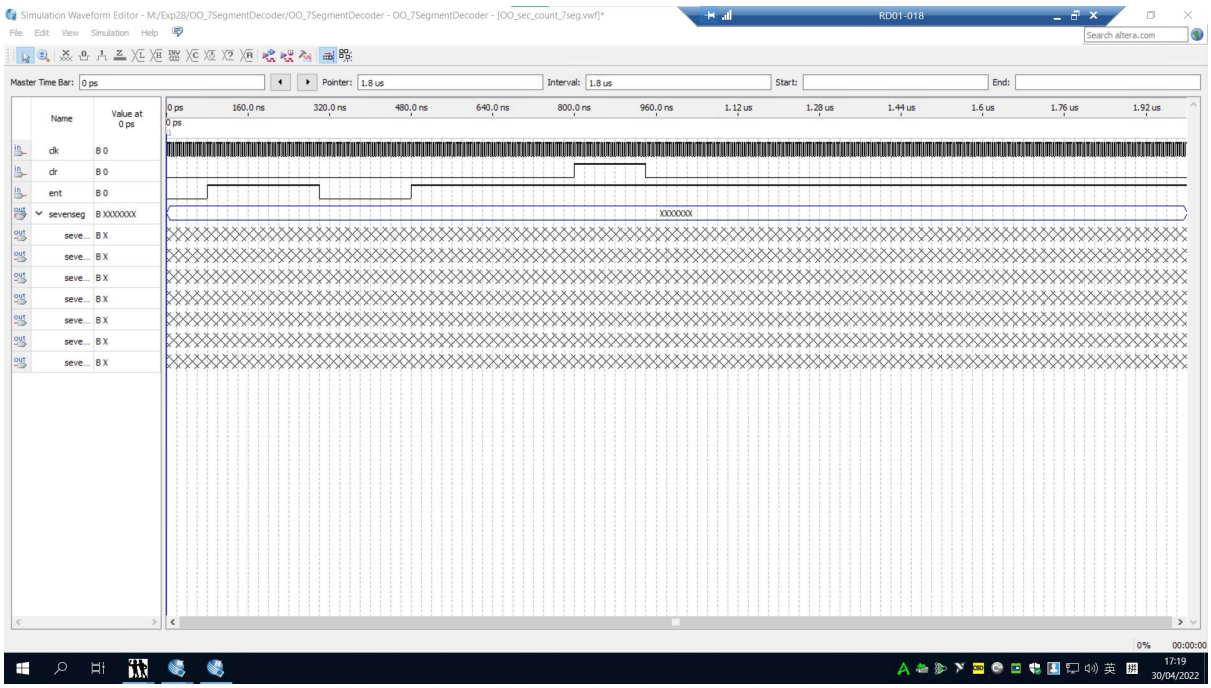


Figure 59: The full-screenshot for the waveform the divide-by-12 counter

Figure 60: The full-screenshot for the output file for the divide-by-12 counter

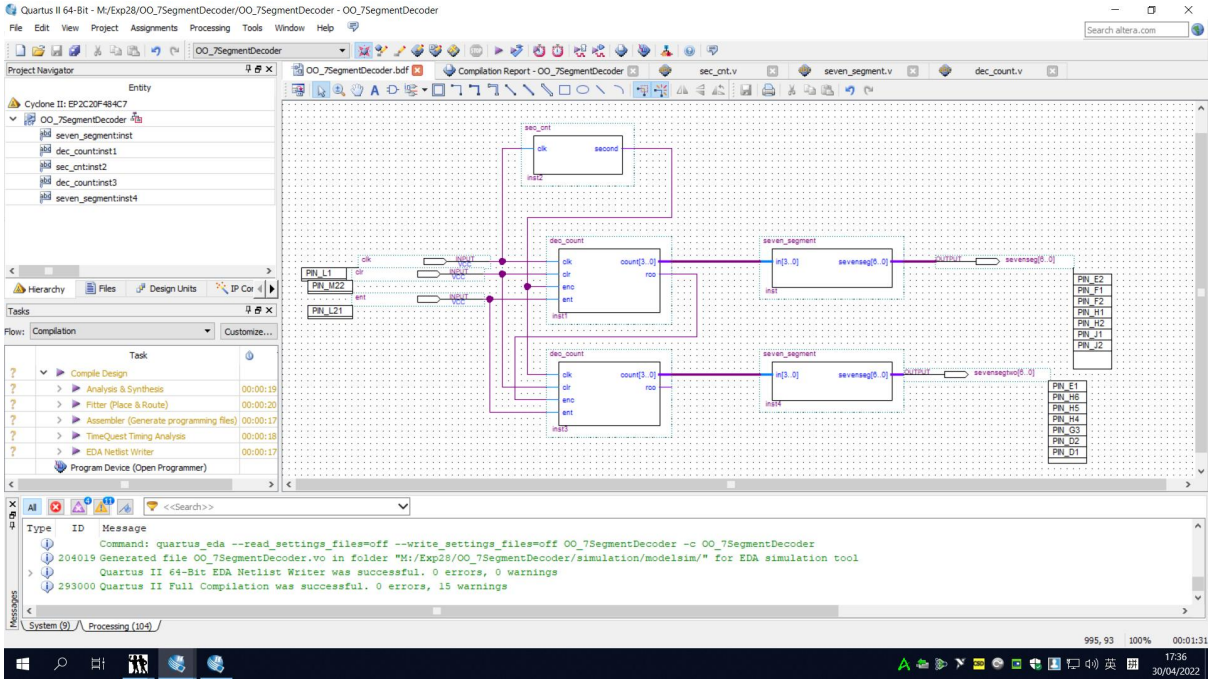


Figure 61: The full-screenshot for cascading simulation

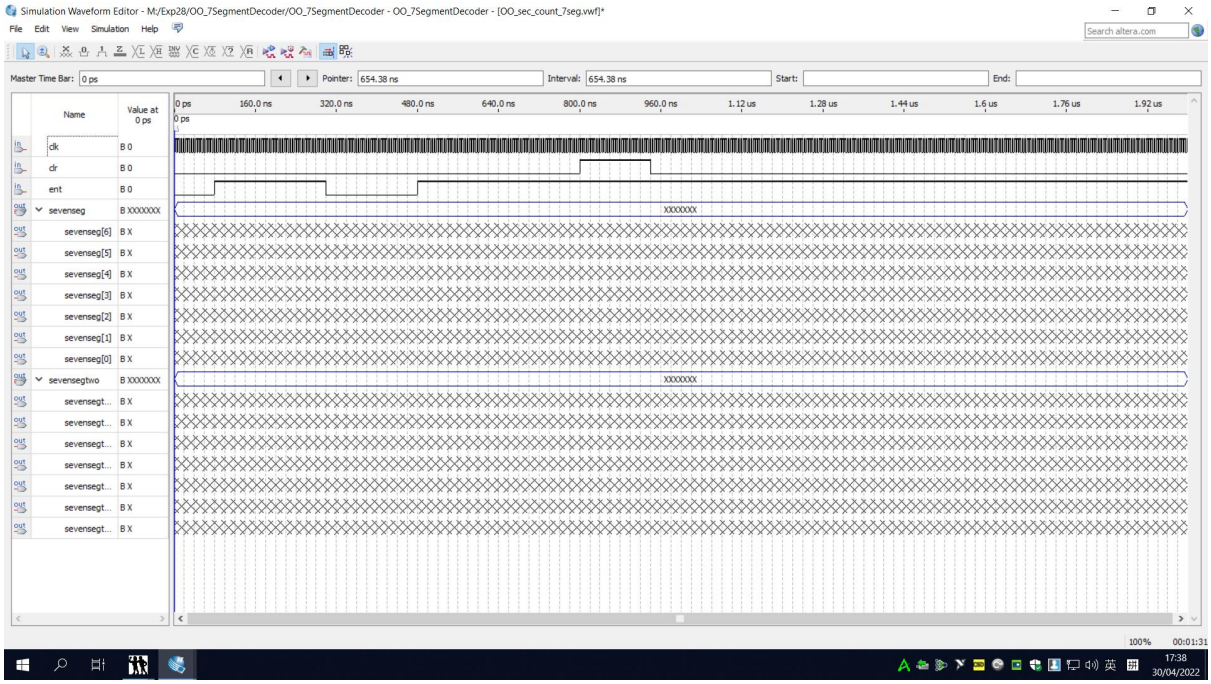


Figure 62: The full-screenshot for the waveform the cascading simulation

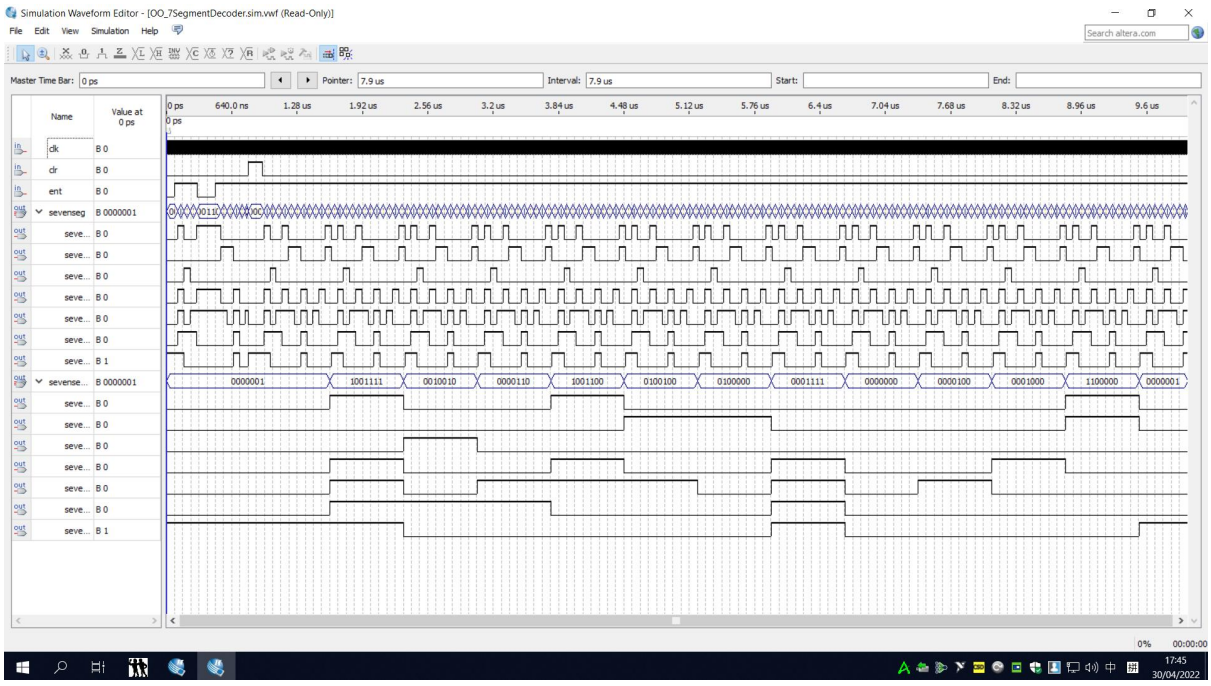


Figure 63: The full-screenshot for the output file for the cascading simulation

Section 4.1 Single Pulser

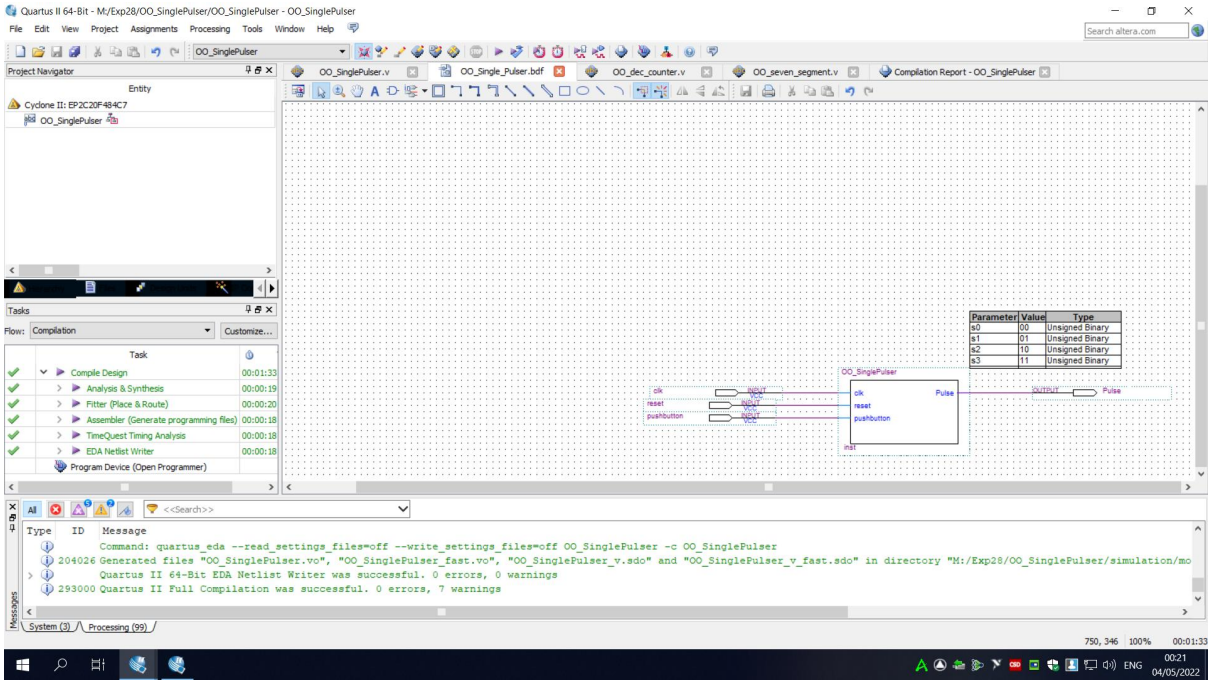


Figure 64: The full-screenshot for the single pulser

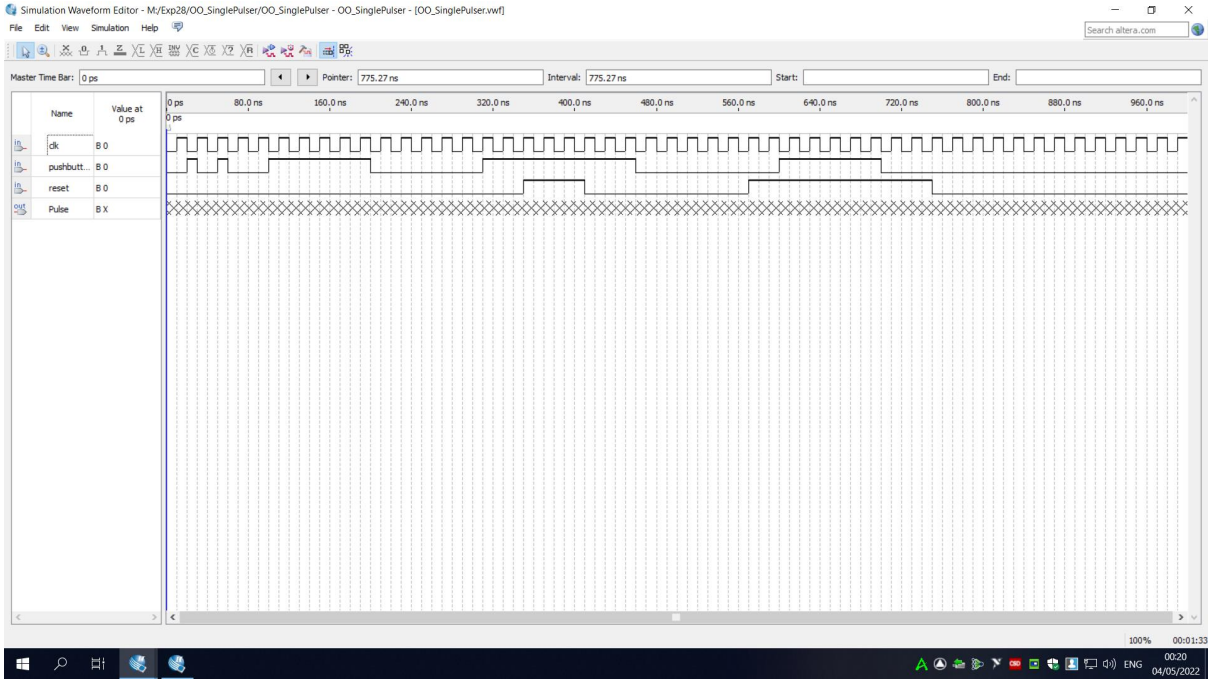


Figure 65: The full-screenshot for the output file for the single pulser

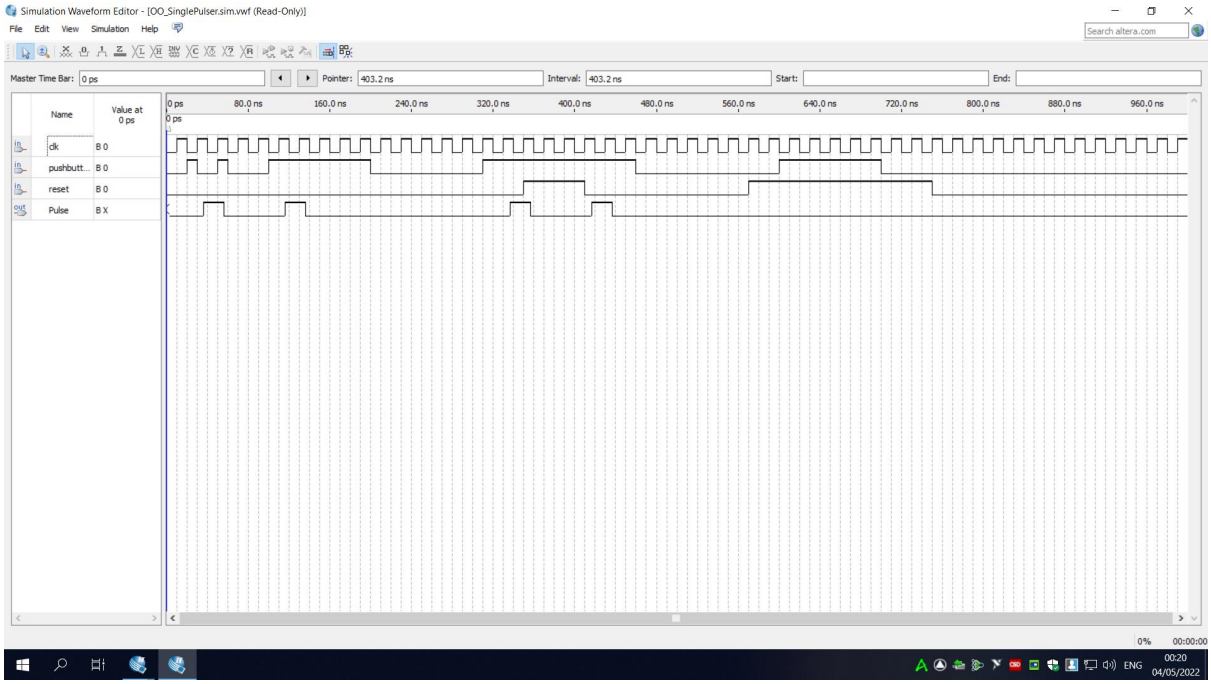


Figure 66: The full-screenshot for the waveform of the single pulser

Section 4.2 Pushbutton Enabled Counter

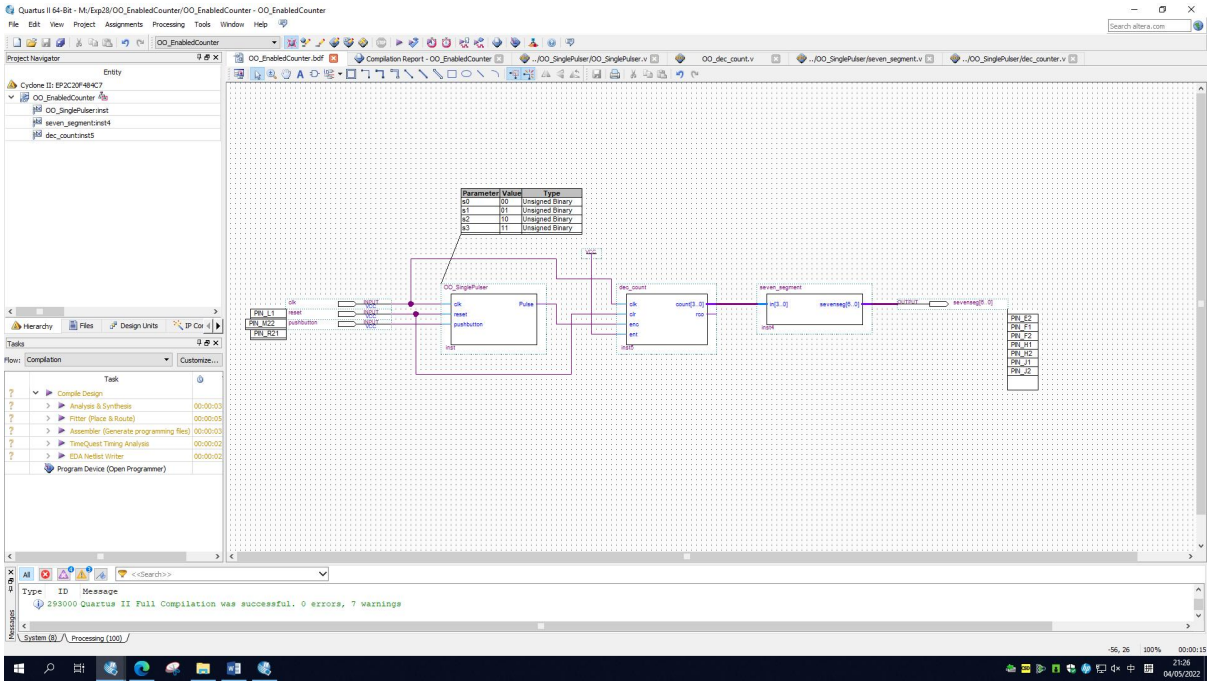


Figure 67: The full-screen-shot for the enabled counter

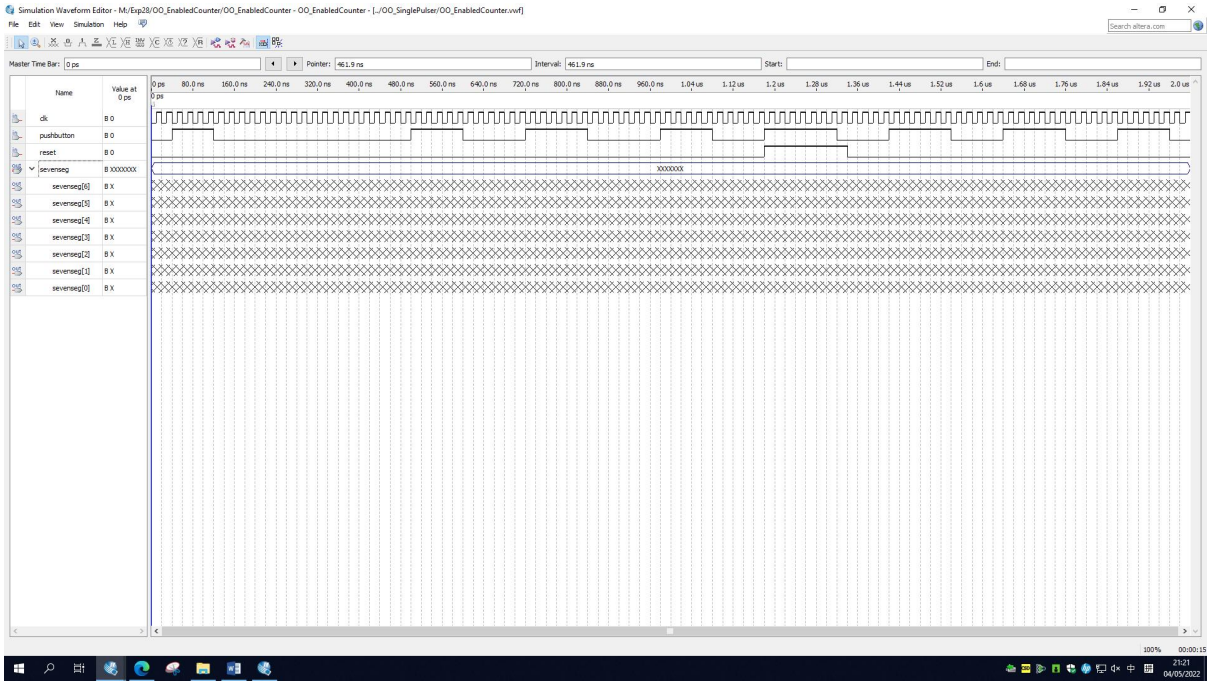


Figure 68: The full screen-shot for the waveform input

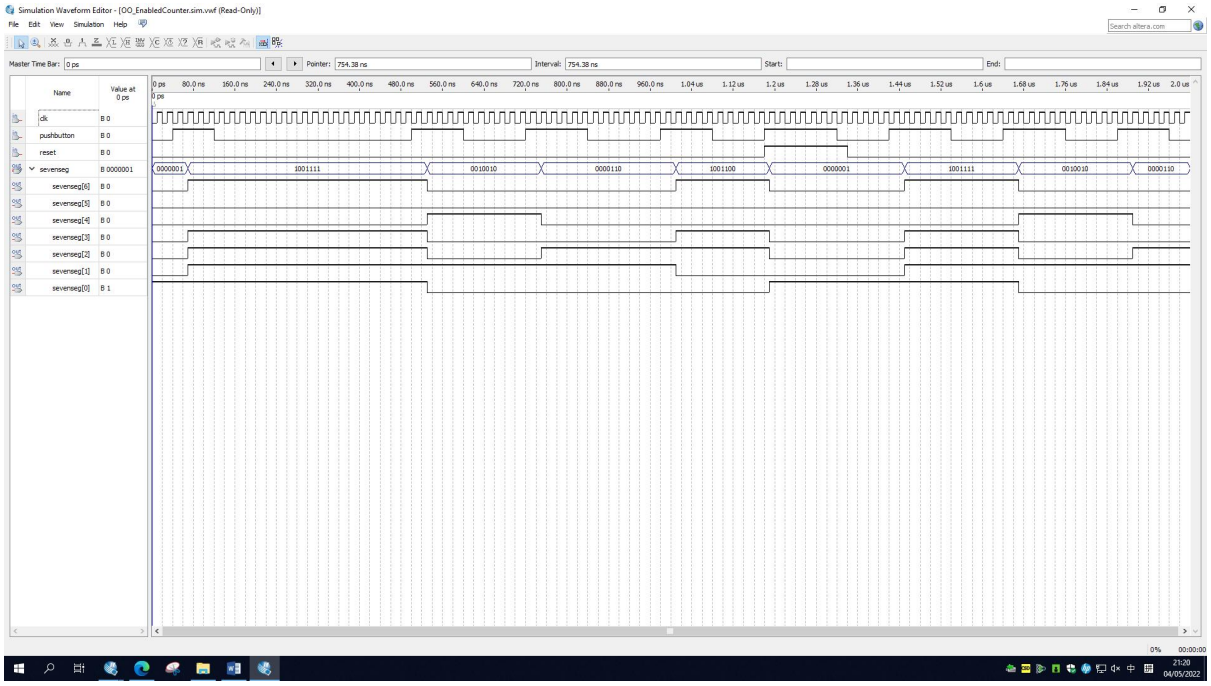


Figure 69: The full-screen for the output file

Section 4.3 Adder/Subtractor Logic Design

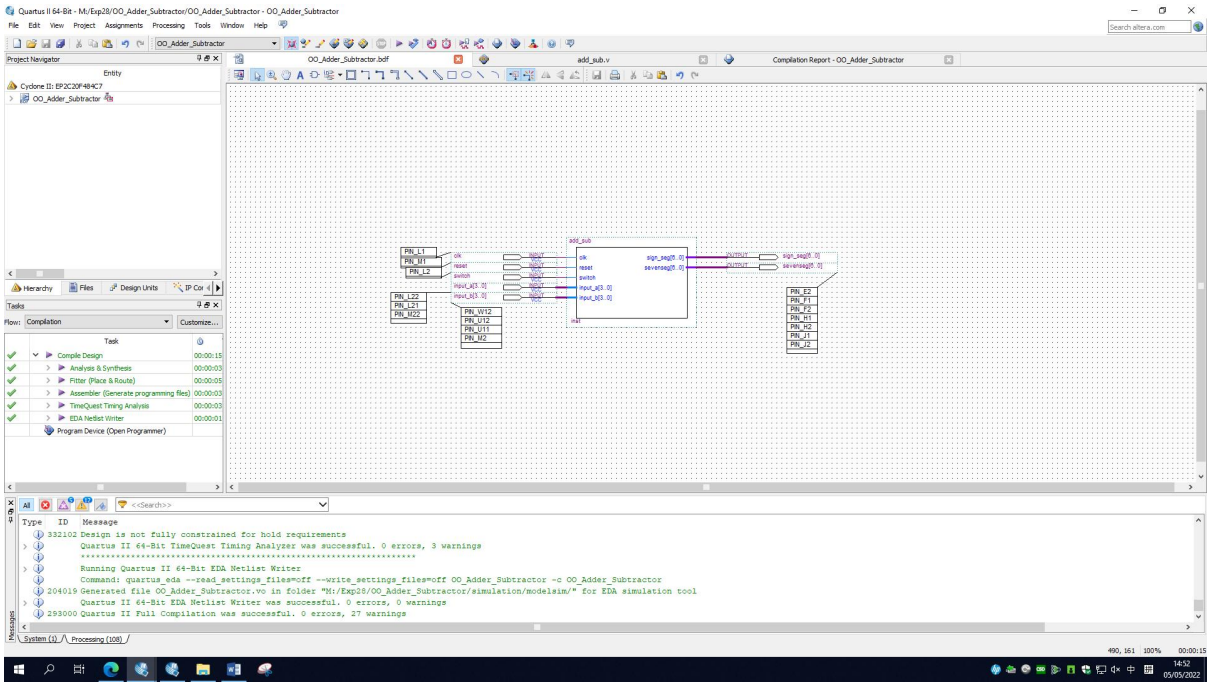


Figure 70: The full-screen-shot for the Adder/Subtractor Logic Design

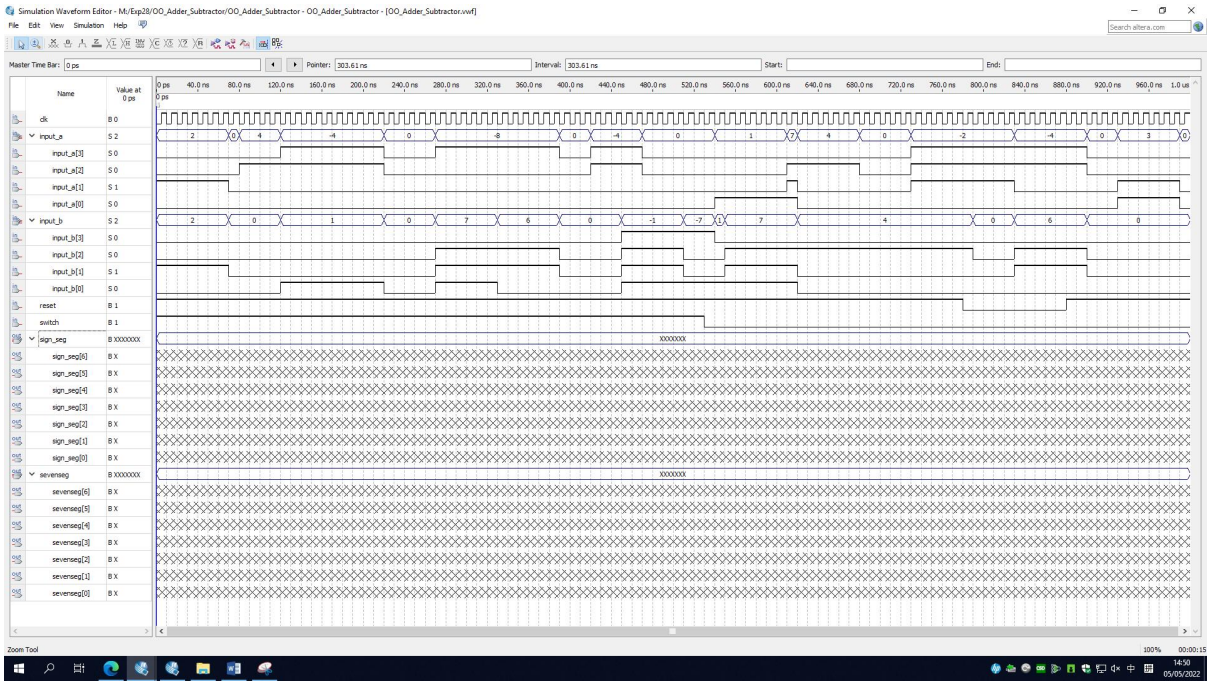


Figure 71: The full screen-shot for the waveform input

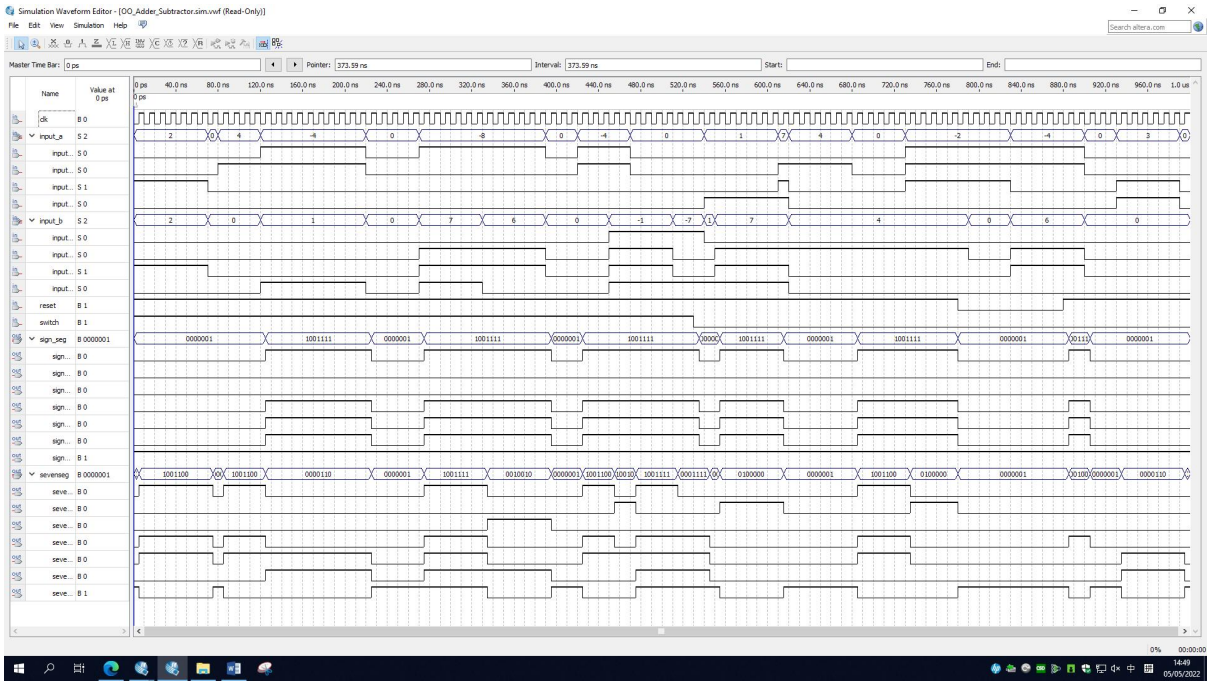


Figure 72: The full-screen for the output file