*Lab Report #3*

*G07*

*Paul Doumet ID#260226189  
Simon Foucher ID#260223197*

**Part 1: ERROR FLASHER**  
  
**1.Goal**  
  
Our Goal in this lab is to design a error flasher circuit, but in this part we are asked to design a timer circuit that can generate a square-waveform at four different frequencies : 1Hz, 2Hz, 4Hz and 8Hz.  
  
**2. Description  
  
**  
  
This circuit represents the flasher circuit. It has three components in it: a frequency divider circuit, a T flip-flop and a multiplexer.

*The frequency divider component* :   
[file: g07\_Flasher\_Timer.vhd]  
  
We have 4 modes of operation: 1, 2, 4 and 8Hz. In order to have ‘Err’ blinking on the Led displays at 1 Hz, the signal needs to toggle at 2 Hz (one time to light up the LED, a second time to blank them out)

This component has 4 inputs:   
  
1) Freq\_Sel is used as a select line for a multiplexor selecting between 4 constancies. The constancies have been calculated such that they reduce the Altera board’s lowest available clock’s frequency (27MHz) to 2, 4, 8 and 16Hz. These frequencies are needed to have the LED segments blink in human time.

If Freq\_Sel is 00 we will have a period of 1/2 (2 pulses/s) which will be representing an ‘Err’ flashing frequency of 1Hz. If Freq\_Sel is 01 we will have a period of 1/4 (4 pulses/s) which will be representing an ‘Err’ flashing frequency of 2Hz. If Freq\_Sel is 10 our period will be 1/8 (8 pulses/s) which will be representing an ‘Err’ flashing frequency of 4Hz and finally when Freq\_Sel is 11 our period will be 1/16(16 pulses/s) which will be representing an ‘Err’ flashing frequency of 8Hz.  
  
2) Enable: It is used as an enable line for our counter. When enable =1 the counter counts and when enable =0 the counter stops counting. Since the counter stops counting the LED display stops flashing. The LED does not blank out when enable equals 0; It only freezes in its present state.   
  
3) Reset: This input is not used due to the fact that our system resets itself naturally at least twice per second. It is present because its part of the library components ports, but it is not used.  
  
4) Clock: Expects a 27 MHz clock signal.

*The flasher multiplexer*:   
[g07\_Flasher\_Mux.vhd]  
The component "G07\_Flasher\_Mux" is composed of 2 multiplexors. Both MUX use the signal from the T flip Flop as a select line and have a 7 bit output.

The first Mux toggles between '1111111' and '14' (in binary) and the second multiplexor selects between '1111111' and '29' (in binary).

They are both synchronized, such that they either both output '1111111' or '14' and '29' respectively.

The output of the multiplexors are then fed to 3 segment decoders as designed in Lab 2. Each segment decoder is connected to a LED on the Altera board. The left most LED decoder's input is connected to the MUX outputting 29, which is decoded into 'E'. The second and third decoders have their output connected to the second and 3rd LED displays and take their input from the '14' Multiplexor. We also used a 4th decoder, connected to the 4th LED segments displays which takes '1111111' as an input. This blanks out the 4th LED segment decoder permanently.

As the counter counts down, whenever it reaches 0, it toggles the flip flop, which toggles the select line from the Multiplexor. This makes the Multiplexors toggle at a frequency 1/2 of the frequency of their inputs. As the Multiplexors toggle, the LEDs will display a flashing 'Blank' and 'Err'.  
  
*The T Flip-Flop*[g07\_TFF.vhd]We use this component as a finite state machine which has 2 states: output ‘Err’ and output ‘blank’. This is necessary because we want state transitions to have a 50% duty cycle for our flashed message using a constant 37ns pulse. This pulse is used to trigger a state transition.

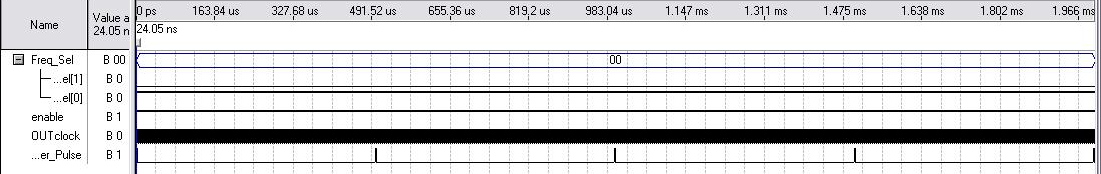
The counter output’s en[0] line gets high only when the counter reaches 0. This forces the T flip-flop to change its state. The T flip flop’s clock is also connected to the 27MHz counter clock.

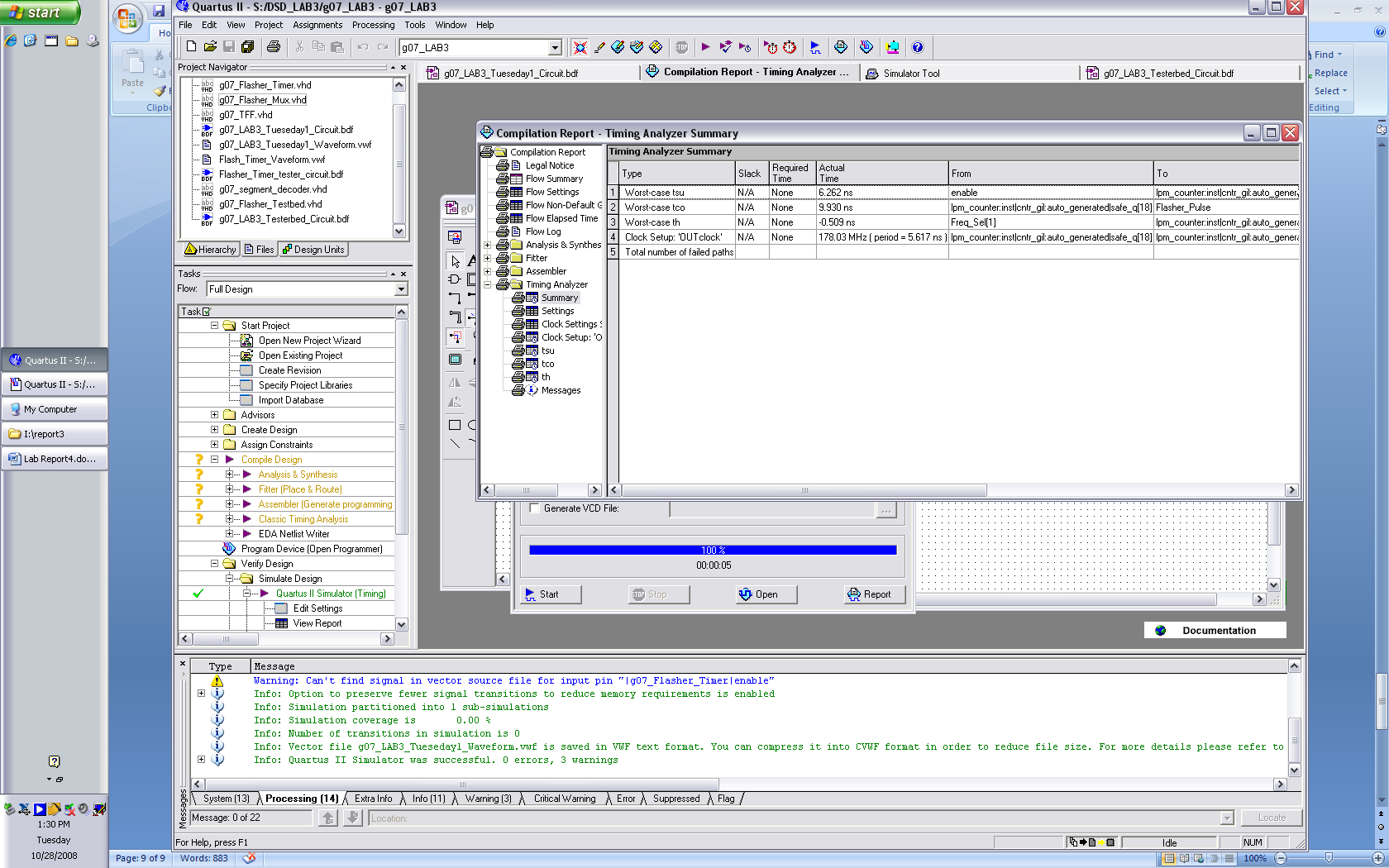
**3.Simulation**

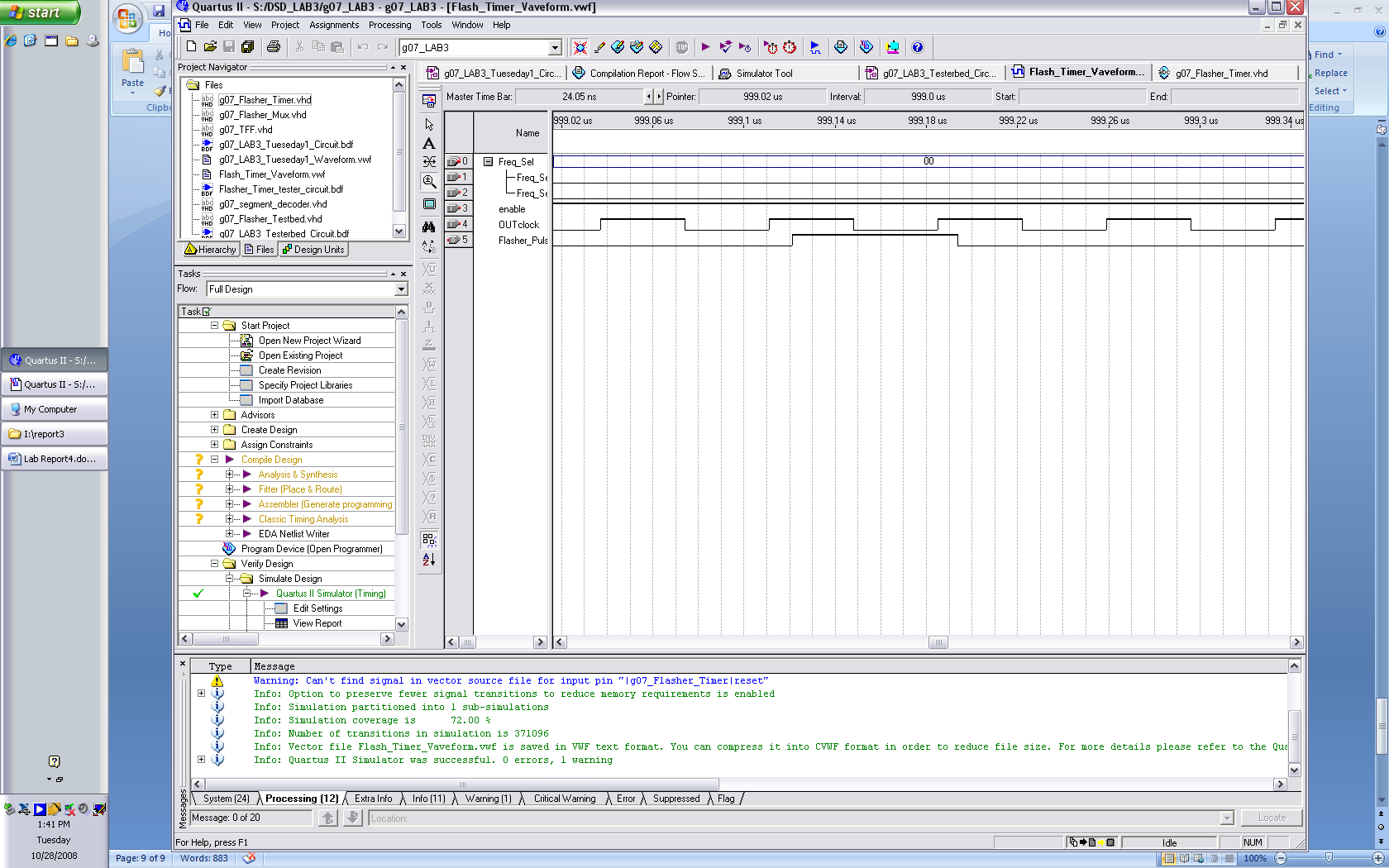
We started by simulating the Flasher\_Timer component. To do so, we had to scale down the constancies by a factor of 1000 such that, we could observe in 1ms what should appear on 1 second of real time.

We tested every 4 Freq\_Sel, and made sure that our constancies were correct. For example, in the following waveform, using mode 00, we should expect ‘Err’ to blink at 1Hz. This means that the Flasher pulse should occur twice a second. In our simulation, we should expect 2 pulses per ms (since we are scaled down by a factor of 1000), which is what we have.

[file: Flasher\_Timer\_Waveform.vwf]



We also observed a worst case set up time of 6.3 ns, which is much smaller than our clock pulse (37ns), so the circuit should behave properly.

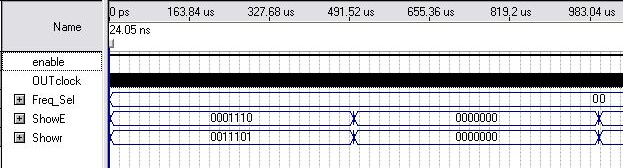


Once all 4 Freq\_Sel were successfully simulated, we built the whole flasher circuit for a single LED cell using our Flasher\_Timer, a T\_FlipFlop and Flasher Multiplexor.

[file: g07\_LAB3\_Tueseday1\_Circuit.bdf]

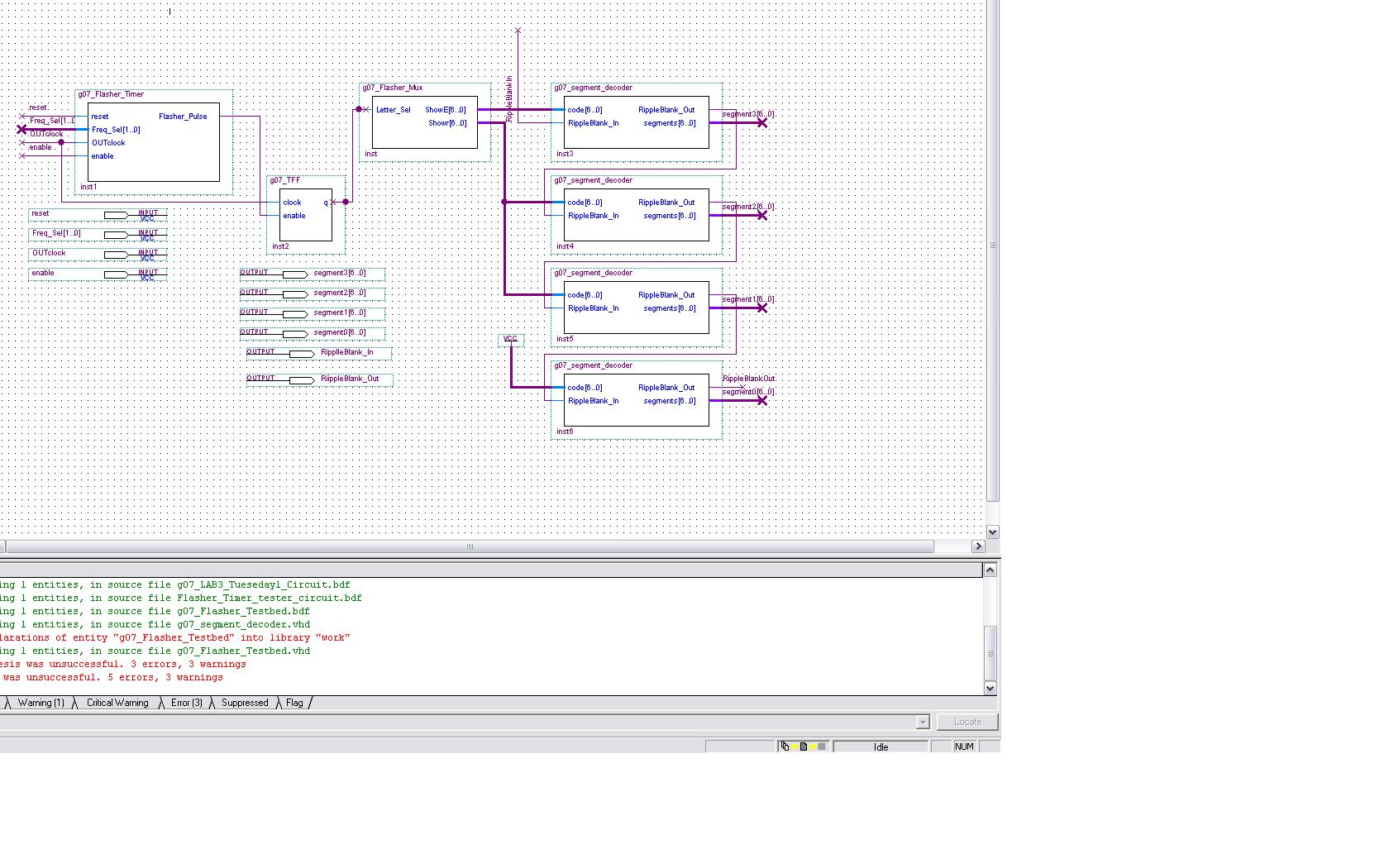
We repeated a waveform simulation, this time observing the output of the Multiplexors to make sure that they displayed ‘14’ and ‘29’ simultaneously and at the required frequencies.

[g07\_LAB3\_Tueseday1\_Waveform.vwf]



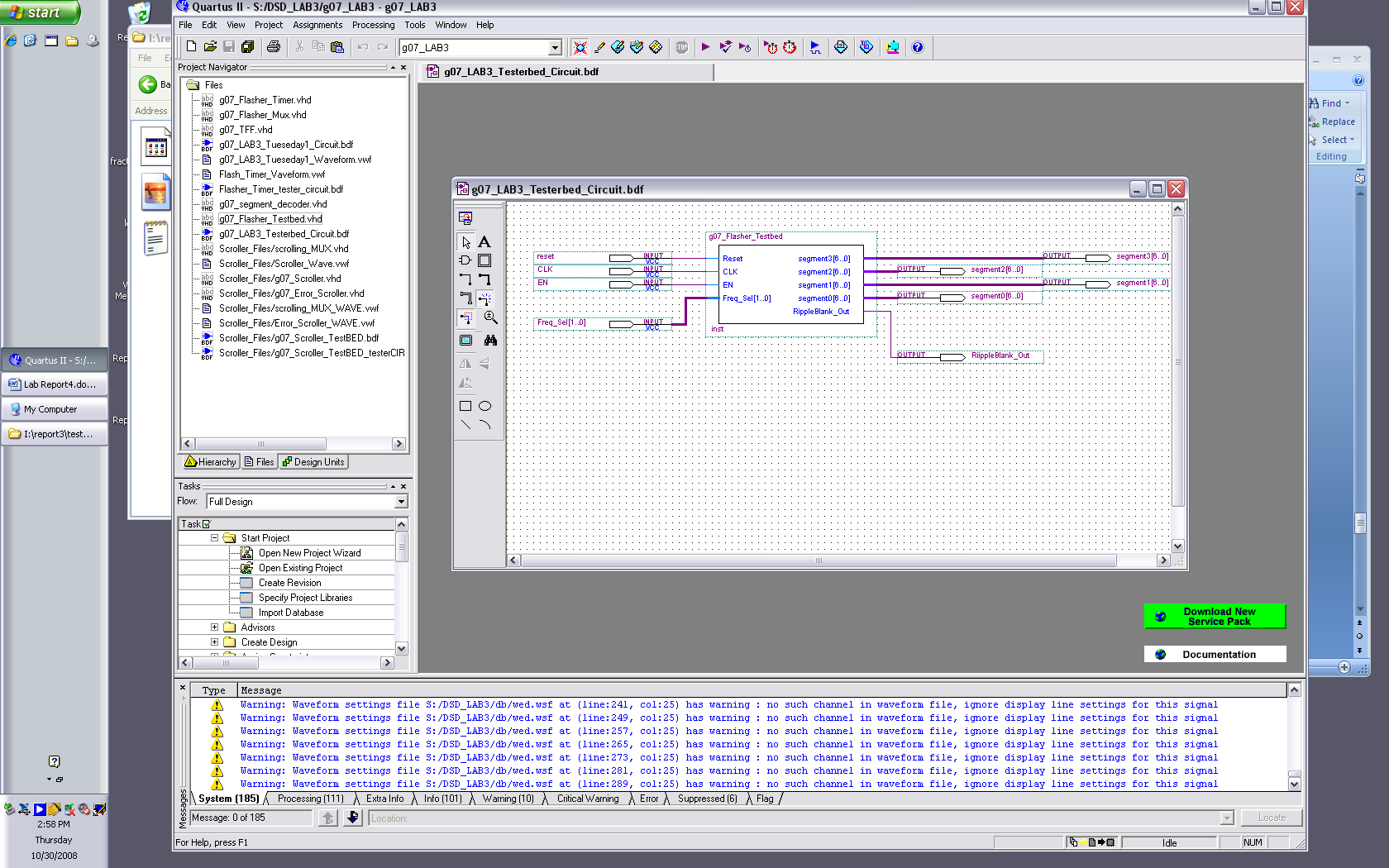
[File: g07\_Flasher\_Testbed]

At this point, since the component behaved perfectly in the simulation we created a circuit using the component 4 times to connect to all 4 LEDs on the board. [file: Flasher\_Testbed.vhd].



We also prepared a circuit with ports to output our worn onto the FPGA board.

[file: g07\_LAB3\_Testerbed\_Circuit.bdf]



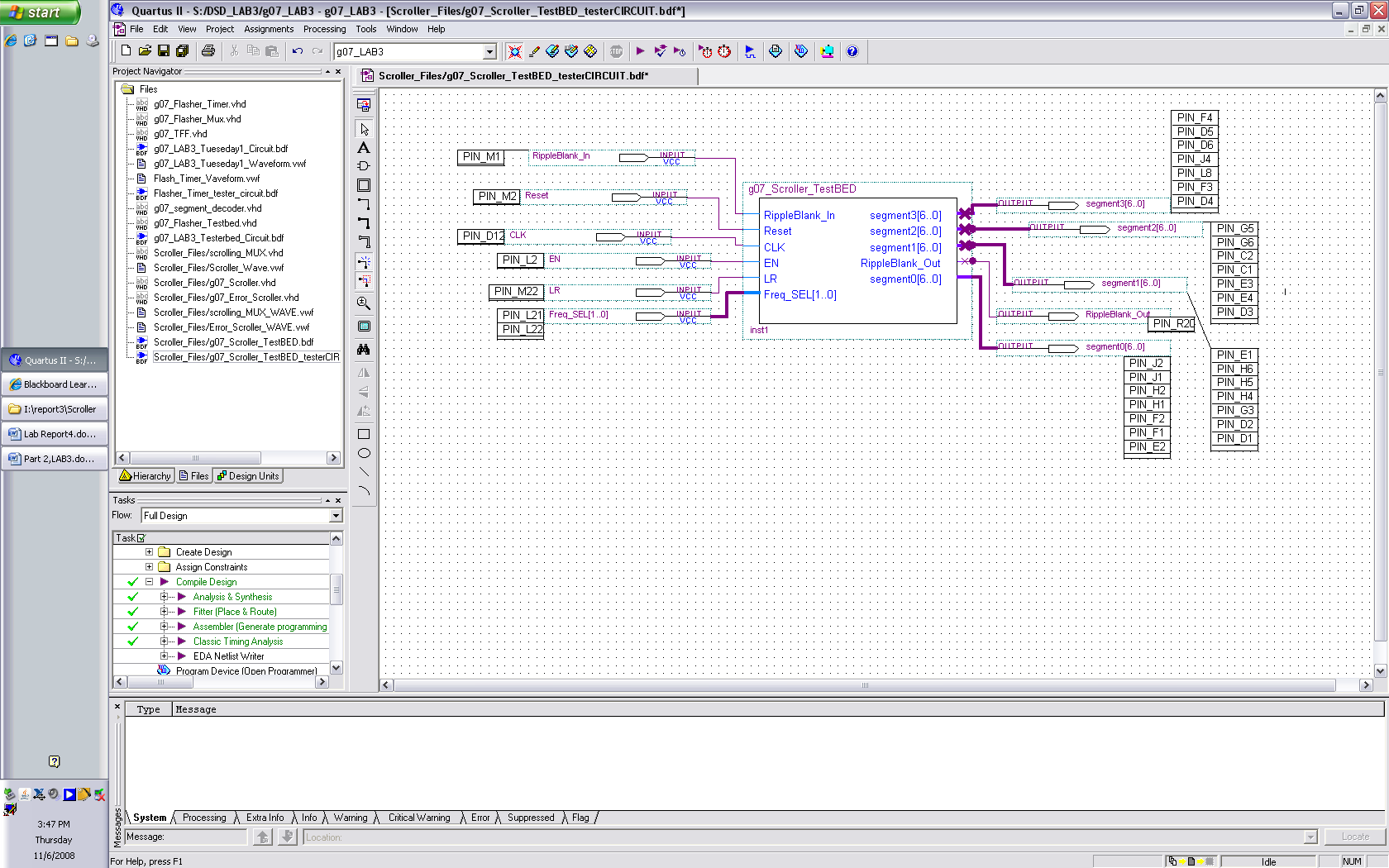
Using the appropriate pin assignment [File: g07\_Lab3\_Pin\_Assignment.csv], we mapped our circuit onto the board and were able to fully test its functions.

We were able to observe every mode of operation with the LED segments displaying intermittent ‘Err’ at the appropriate frequencies.

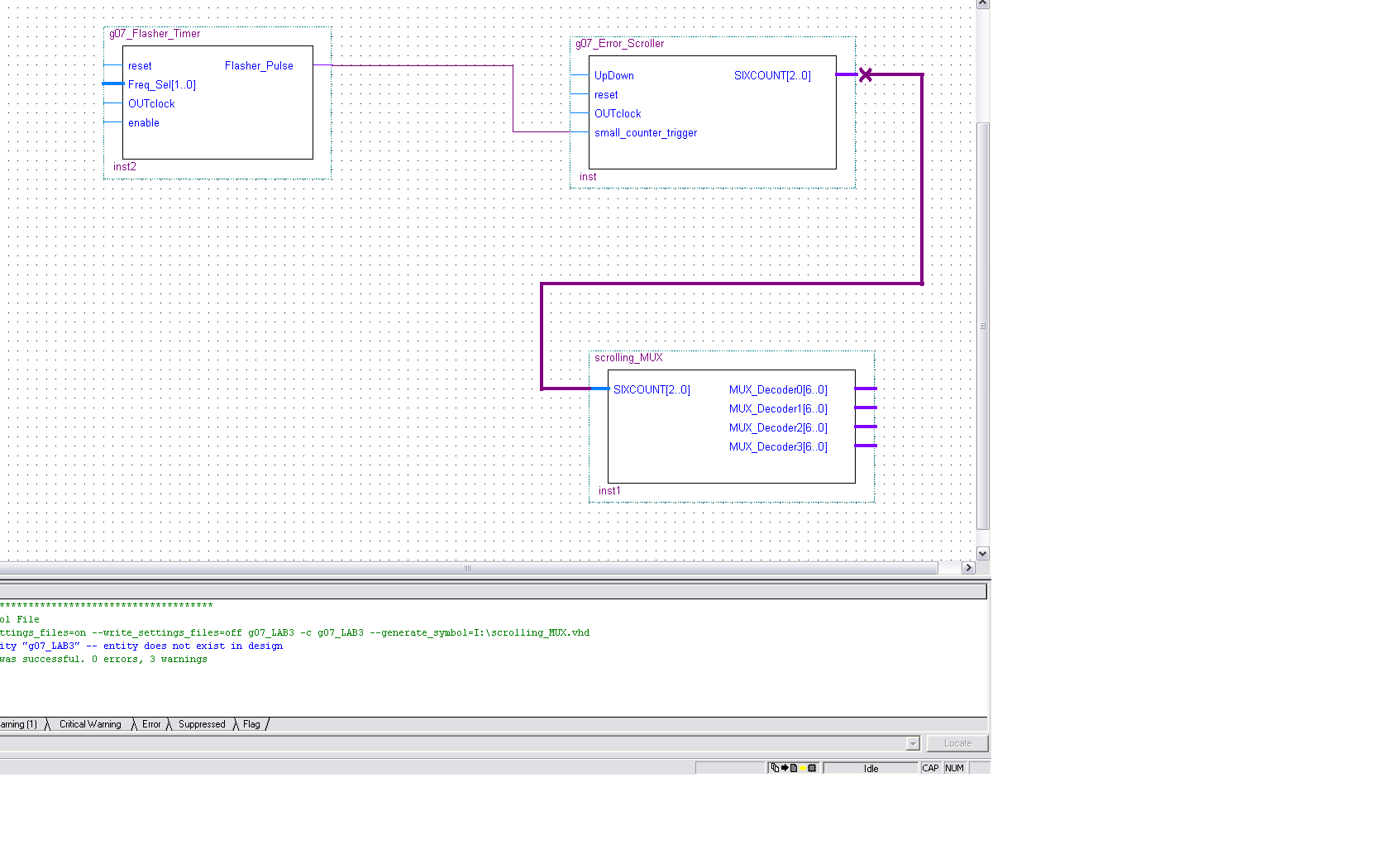
**Part 2: Error Scroller Circuit**  
  
**Goal:**   
  
Our goal in this part is to create a scrolling error message on the LED’s. The speed of the scrolling message is determined by our flasher timer circuit which sets one of the respective four frequencies: 1,2,4 or 8Hz. The direction of the scroll message will be adjustable using a 1 bit input.

**Description**:

Pinout diagram



\*\*The full scroller circuit is called ‘scroller’. Note that ‘error scroller’ is an internal component of ‘scroller’; and they are not the same entity.

Internally, G07\_scroller is composed of 3 main components:  


A – G07\_Flasher\_Timer  
B – G07\_Error\_Scroller (Do not confuse with ‘G07\_scroller’)  
C – scrolling\_MUX

(All the relevant files to the second part of the lab – the scroller- are located in /G07\_DSD\_LAB3/Circuit\_files/Scroller\_Files/)

A - The flasher time is identical to the one described in part 1 of the lab report

B - Our error scroller circuit has 4 inputs:

1)UpDown : UpDown determines the direction of the scrolling message, if UpDown = 1 the message scrolls from right to left and the counter starts counting from 0 to 5. If UpDown = 0 the message scrolls from left to right and the counter starts counting from 5 to 0.

2) Reset: Manually resets the value of the counter. If counting up, resets the counter to 0. If counting down, resets the counter to 5. (The reset switch needs to be turned off in order for the counter to be able to count)  
  
  
3)OUTclock: 27 MHz Clock with a 37 nS period. Since all of our settle times are lower than 10 nS, the logic has time to settle to a stable output in-between clock pulses.  
  
  
4) small\_counter\_trigger: input to the counter. Every events on this line causes the counter to count either 1 up or down (depending on the counting mode set by UpDown)

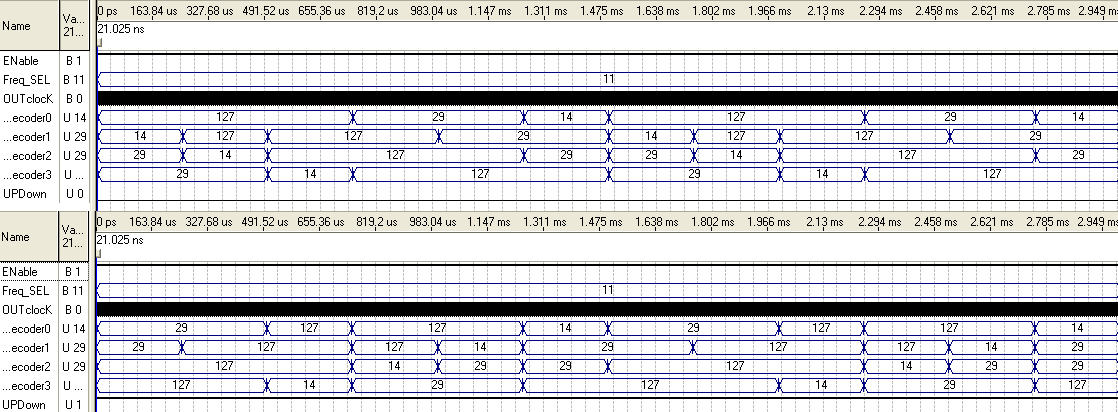
Outputs: The error scroller has a single output which is connected to the input of the MUX decoder.

C – scrolling\_MUX

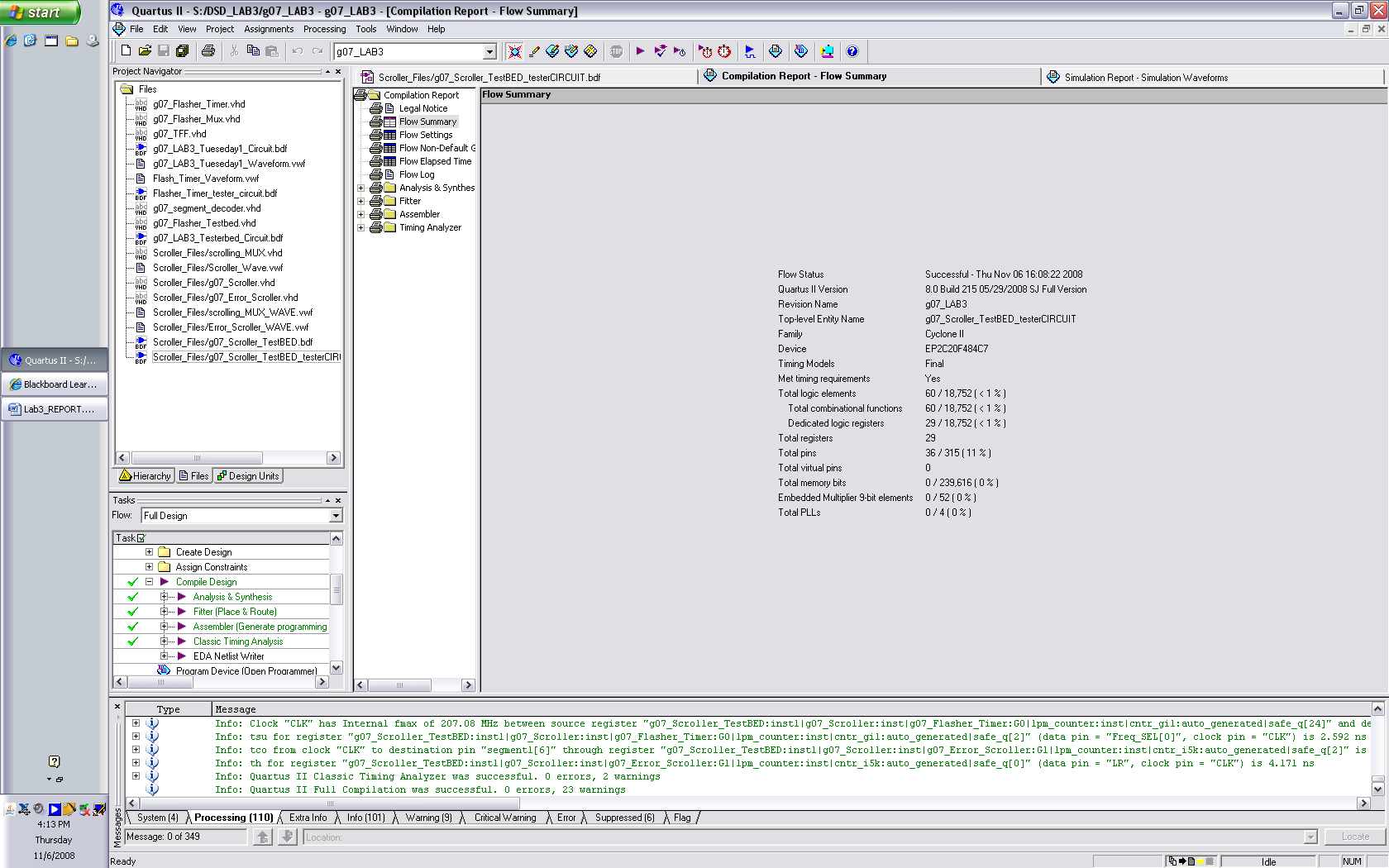
The decoder has 4 outputs, each of which will be connected to a LED digit on the board. Internally, they are very similar.

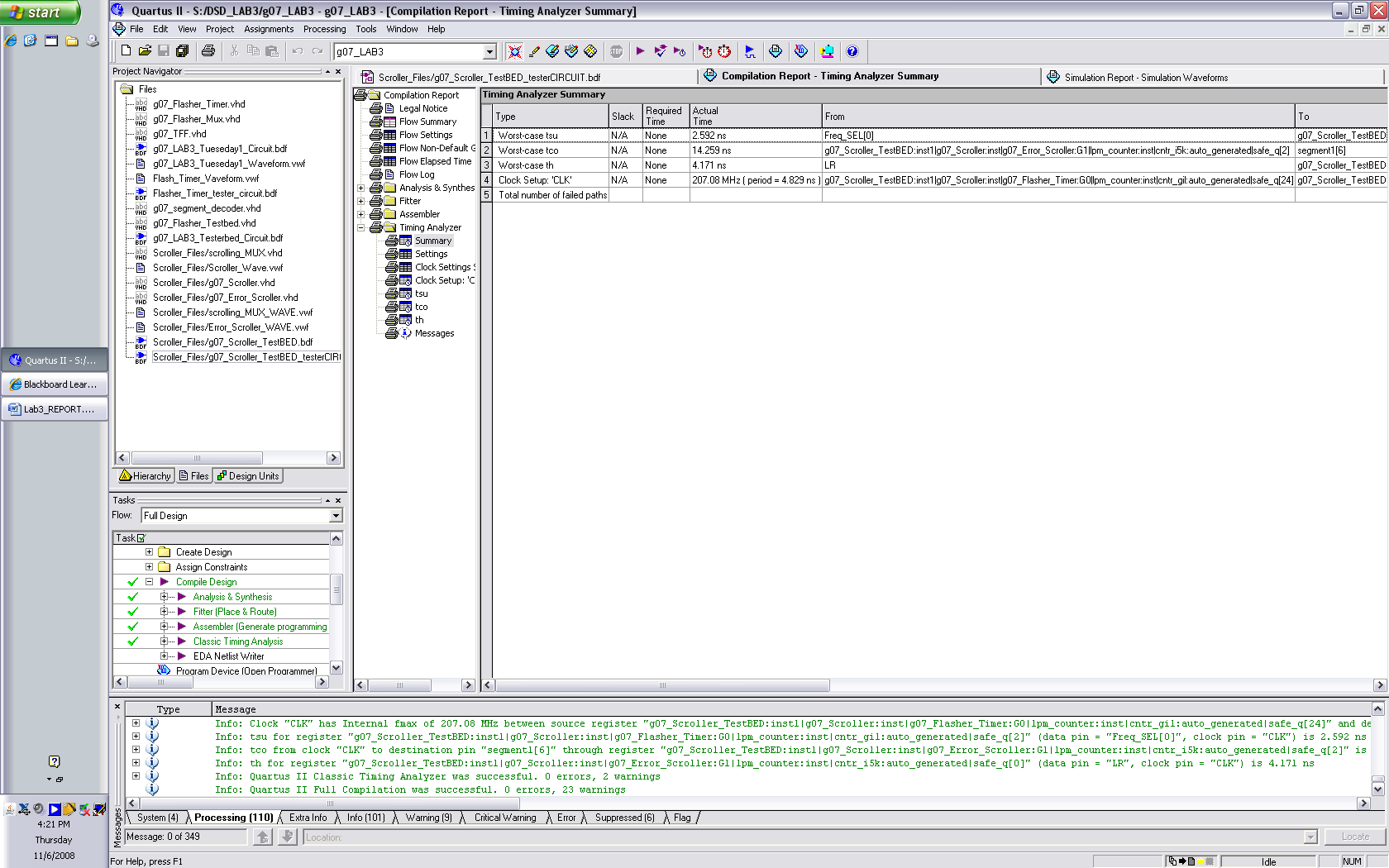
In order to determine the outputs, we created a state diagram for every LED, to map what it should output at what count.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| COUNT | LED0 | LED1 | LED2 | LED3 |
| 0 | E | R | R | BLANK |
| 1 | BLANK | E | R | R |
| 2 | BLANK | BLANK | E | R |
| 3 | BLANK | BLANK | BLANK | E |
| 4 | BLANK | BLANK | BLANK | BLANK |
| 5 | R | BLANK | BLANK | BLANK |

Based on this information, we designed 4 multiplexors, taking the count from the counter as a select line and outputting 14 where the LED should display E, 29 where the LED should display R and 127 where the LED should display blank.  
  
 **Simulation:**   
  
  
  
In the first simulation, UPDown = 0 and we can see that our decoder0 which is connected to the first LED gets assigned: Blank, R, E, Blank, R, E…  
decoder 1 gets assigned: E, Blank, R, E, Blank, R…   
decoder 2 gets assigned : R, E, Blank, R, E, Blank,R…  
decoder 3 gets assigned : R, R,E, Blank, R, E, Blank…  
  
in this case we can clearly see that it is a right shift BUT once plugged in the altera board, we realized that our message was spelled backwards. In order to fix this problem, we reversed the pin assignments such that segment3 was pointing at LED 0. Therefore when UPDown = 0 it is actually a left shift.  
  
In the second simulation, UPDown = 0 we see that:   
 decoder0 gets assigned : R, Blank, Blank, E, R, Blank, Blank, E…  
decoder 1 gets assigned :R, Blank, Blank, E, R, Blank, Blank, E, R…  
decoder 2 gets assigned : Blank, E, R, R, Blank, E, R ,R…  
decoder 3 gets assigned : Blank, E ,R , Blank, E, R , Blank…  
  
This here is clearly a left shift but for the same reasons as stated earlier it is a right shift.   
Just by looking at the simulation we can tell that the Error scroller is correct.

In the second simulation, UPDown = 0 we can observe the opposite result, so the scrolling happens the other direction.

Simulation results:  


We can see from our Flow summary that 60 logic elements have been used which is less that 1% of the board’s capacity, and the total pins used is 36 which represents 11% of the board’s pin capacity.  
  
We can also see from our timer analyzer that our propagation delay is 14.259 nS. Our worst case set up time is 2.592 nS and our worst case hold time is 4.171 nS.

We were able to successfully implement this on the board. The .sof file included in this submission contains the adequate binary.