Team Leader:

Erich Viebrock

Members:

Erich Viebrock, Gilbert Rivera

**Stopwatch**

Author(s) of the Report:

Erich Viebrock

Date of Submission:

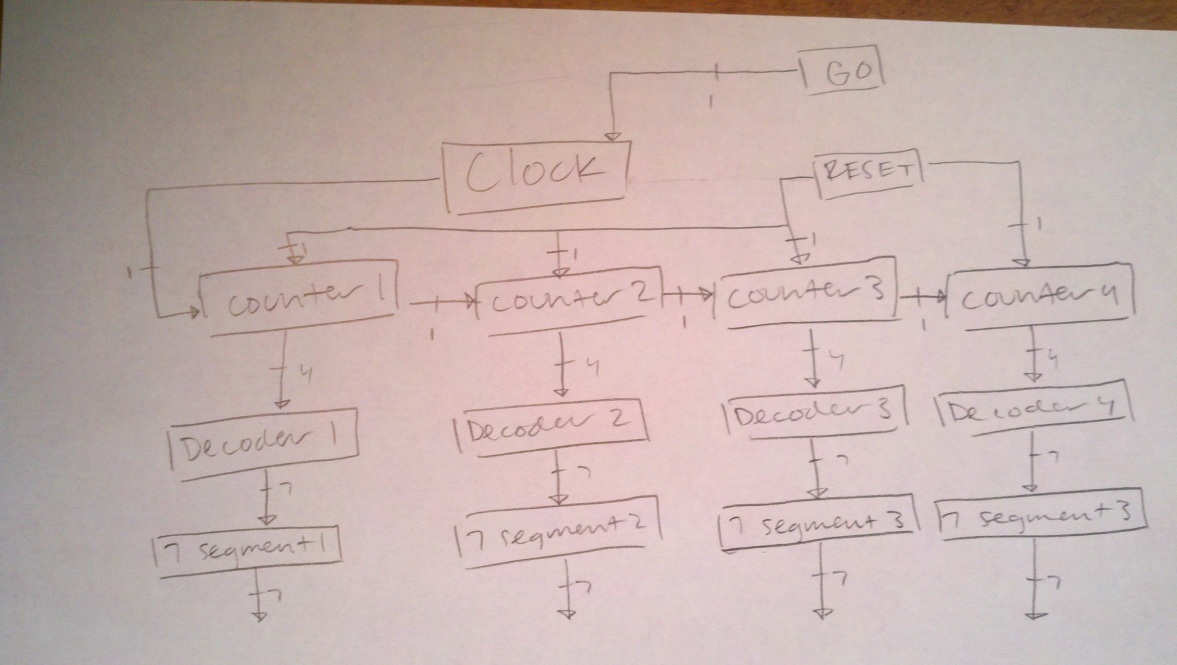
12 December 2011

1. Introduction

Stopwatches are used constantly within society, as precision timing is needed for competition, testing, and infinitely many scenarios. Our project is to recreate a digital stopwatch. Our digital stopwatch will use two toggle switches as inputs; one switch for “GO,” and one switch for “RESET.” The digital stopwatch will begin once a toggle switch is set to HIGH. Of course, if the “GO” switch is LOW the stopwatch will stop counting, but contain its current value. The time for the digital stopwatch will output to the seven segment display, utilizing all 4 seven segment displays. Once the stopwatch has reached 9959 on the seven segment display, the system will stop, as this also simulates a realistic stopwatch reaching the maximum time the stopwatch can output. The entire project will be written in VHDL.

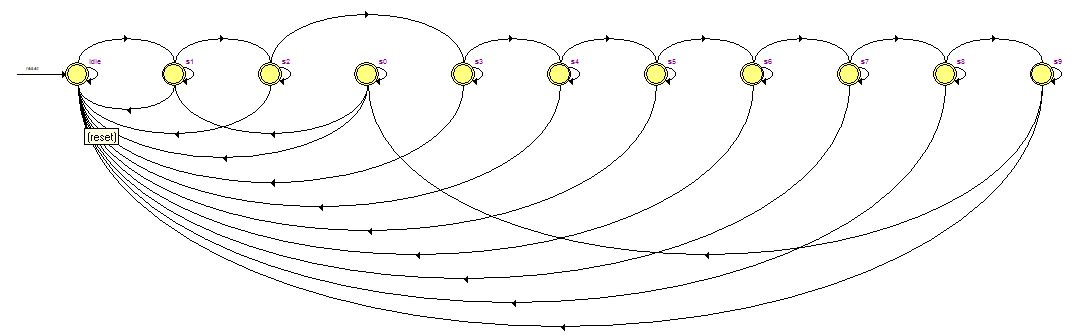
1. Theoretical Background

In order to understand how a stopwatch functions, a basic understanding of the clock inside the FPGA, and a clock divider is necessary. First, the FPGA, along with virtually all digital electronics, function within discrete time. This mode of time simply means time is sampled, at a given rate, or frequency. The frequency within digital systems varies widely, but is usually much faster than conceivable to humans. However, note the frequency of a clock signal accounts for the clock to run through two states: HIGH and LOW. Therefore, when a clock signal sees a HIGH and LOW state, this denotes a successful clock cycle. Usually, the clock changes from HIGH to LOW, and LOW to HIGH in even increments. For instance, the frequency of the FPGA is claimed to be clocked at 27 MHz. This means the FPGA refreshes the system 27,000,000 per second. For creating a project conceivable to the human eye, this clocking rate needs to be restrained. In order to restrain the clock to a reasonable clock rate, a clock divider can be written in VHDL, or Verilog. This file commands the FPGA to count however many clock edges the user demands. There is not one unique answer to a clock divider, but one method is to command the FPGA to count positive clock edges at a given increment.

1. Solution
   1. System Diagram:

The system diagram depicts the entire stopwatch in segments of logic blocks. The user has two inputs, “GO” and “RESET,” which are toggle switches. GO enables the stopwatch to begin, which starts at the clock. Once the clock is enabled, the system moves to “COUNTER1” which is the pseudo name given to the hundredths of a second counter of the stopwatch. “COUNTER2,” COUNTER3,” and “COUNTER4” are all counters to the respective portion of the stopwatch. Each counter sends information to the decoder, which assigns a numerical value to “LIGHTS,” the output of the FSM. The numerical value for LIGHTS is then sent to the respective 7 segment, which outputs the value to the FPGA. If RESET is ever used, the stopwatch reverts back to the initial state, 0000. Each counter sends 1 bit of information to the next, which is if overflow is HIGH or LOW. Each counter sends 4 bits of information to their respective decoder, for which state the FSM is currently in. Then, each DECODER sends 7 bits of information to the 7 segment, which allows the binary number to be represented as a base 10 number.

* 1. Design Description:
     1. State machine diagram



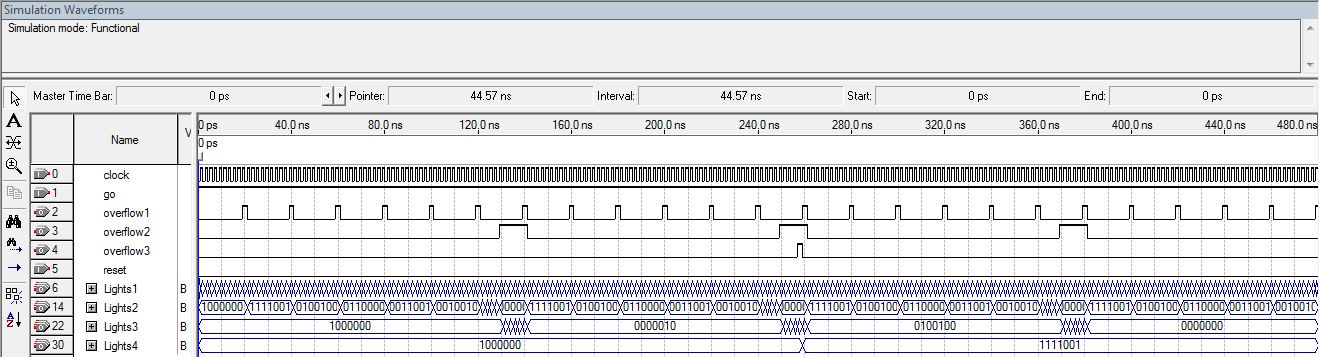
Unfortunately, the state machine viewer in Quartus only displays one FSM of the counter, all 4 FSMs are identical. The states from left to right are as follows: idle, s1, s2, s0, s3, s4, s5, s6, s7, s8, and s9. Quartus’s ordering of the states is odd, but accurately demonstrates how the FSM moves from each state, as GO is HIGH and RESET is LOW.

1. Testing Procedures

In order to test our stopwatch, the process was simple. Begin with the smallest portion of the stopwatch. Once that portion was functional, add the next piece, and so forth. Initially, the design was very hierarchical, as all the components of the stopwatch were divided into separate .vhd files. There was a separate file for the first counter, second counter, third counter, fourth counter, seven segment, clock, and comprehensive FSM. Each counter and seven segment was individually tested. Once these files were functional, the next step was to incorporate all the files into a comprehensive FSM. However, this challenge proved too difficult. At best, I could only get the first two counters to run, whereas the third and fourth were completely dysfunctional. I decided this method was troublesome and required more effort than necessary. I had trouble creating COMPONENTs and PORT MAPping all necessary pieces of the stopwatch, so I decided to incorporate all the .vhd files into one .vhd file. This solution allowed me to avoid the problem of creating PORT MAPs and COMPONENTs, as all the necessary files I needed were already in the .vhd file. The main benefit of making a project more hierarchical results in a faster clocked system. Therefore, my method of creating a single .vhd file for the stopwatch could be deemed bad practice. However, this is not an issue in reality; a stopwatch needs to run at 100 Hz, or 1/100 of a second to be fully functional, which will in no way be affected by not making a hierarchical design. When testing the single .vhd file, results were much better than before. Although, the design was not fully functional as desired. The first two counters were functioning correctly, however the third and fourth were not.

1. Results and Discussion

Vector Waveform File



This is the vector waveform file of the dysfunctional stopwatch. Notice How the first and second overflows are registering correctly, however the third and fourth FSM do not function correctly. Once the second overflow is registered, the third FSM runs at 100 Hz, rather than the desired 1 Hz. The fourth FSM runs into the same problem.

1. Task Breakdown
   1. Erich Viebrock – My task was to design the stopwatch, and any components requiring the stopwatch to run. Therefore, I needed to design the FSM for 4 counters, and change the clock divider to divide each 100 Hz.
   2. Gilbert Rivera – Gilbert’s task was to design the memory for an alarm, and to create the poster.
2. Conclusion

Concluding, designing the stopwatch was more difficult than expected. The first big hurdle was to find a way to have the first counter enable the second counter to begin counting. The solution to this problem was simply adding the output of the first FSM to the second FSM, as an input. In order to do this, each overflow required to be a buffer. By changing the overflows to buffers, they could be changed from inputs to outputs, within the system. After this problem was resolved, I believed the rest of the stopwatch would be fairly easily to solve. However, I was never able to enable the third and fourth FSM to count correctly. Hence, the stopwatch did not behave as a traditional stopwatch. If I were to create 4 different clock dividers, and port them into each FSM to allow each FSM to run independently, I believe this would resolve the issue I was facing. However, I think this is cheating the stopwatch, and should only run to one clock divider.

1. Appendix

**VHDL code:**

-- comprehensive FSM

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

ENTITY final\_project IS

PORT(go, reset, clockdiv : IN STD\_LOGIC;

overflow1, overflow2, overflow3 : BUFFER STD\_LOGIC;

Lights1, Lights2, Lights3, Lights4 : BUFFER STD\_LOGIC\_VECTOR (1 to 7));

END final\_project;

ARCHITECTURE Behavior OF final\_project IS

COMPONENT clk -- clock divider component

PORT(iclk : IN STD\_LOGIC;

oclk : OUT STD\_LOGIC);

END COMPONENT;

-- FSMs with respectable states

SIGNAL clock : STD\_LOGIC;

TYPE STATE\_TYPE\_a IS (idle, s0, s1, s2, s3, s4, s5, s6, s7, s8, s9);

SIGNAL a: STATE\_TYPE\_a;

TYPE STATE\_TYPE\_b IS (idle, s0, s1, s2, s3, s4, s5, s6, s7, s8, s9);

SIGNAL b: STATE\_TYPE\_b;

TYPE STATE\_TYPE\_c IS (idle, s0, s1, s2, s3, s4, s5, s6, s7, s8, s9);

SIGNAL c: STATE\_TYPE\_c;

TYPE STATE\_TYPE\_d IS (idle, s0, s1, s2, s3, s4, s5, s6, s7, s8, s9);

SIGNAL d: STATE\_TYPE\_d;

BEGIN

clkassignstage: clk PORT MAP(clockdiv, clock);

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- hundredths of seconds

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROCESS(clock, go, reset, a)

BEGIN

IF (clock'EVENT AND clock = '1') THEN

CASE a IS

WHEN idle =>

IF reset = '1' THEN

a <= idle;

ELSIF go = '1' AND reset = '0' THEN

a <= s1;

ELSE

a <= idle;

END IF;

WHEN s0 =>

IF reset = '1' THEN

a <= idle;

ELSIF go = '1' AND reset = '0' THEN

a <= s1;

ELSE

a <= s0;

END IF;

WHEN s1 =>

IF reset = '1' THEN

a <= idle;

ELSIF go = '1' AND reset = '0' THEN

a <= s2;

ELSE

a <= s1;

END IF;

WHEN s2 =>

IF reset = '1' THEN

a <= idle;

ELSIF go = '1' AND reset = '0' THEN

a <= s3;

ELSE

a <= s2;

END IF;

WHEN s3 =>

IF reset = '1' THEN

a <= idle;

ELSIF go = '1' AND reset = '0' THEN

a <= s4;

ELSE

a <= s3;

END IF;

WHEN s4 =>

IF reset = '1' THEN

a <= idle;

ELSIF go = '1' AND reset = '0' THEN

a <= s5;

ELSE

a <= s4;

END IF;

WHEN s5 =>

IF reset = '1' THEN

a <= idle;

ELSIF go = '1' AND reset = '0' THEN

a <= s6;

ELSE

a <= s5;

END IF;

WHEN s6 =>

IF reset = '1' THEN

a <= idle;

ELSIF go = '1' AND reset = '0' THEN

a <= s7;

ELSE

a <= s6;

END IF;

WHEN s7 =>

IF reset = '1' THEN

a <= idle;

ELSIF go = '1' AND reset = '0' THEN

a <= s8;

ELSE

a <= s7;

END IF;

WHEN s8 =>

IF reset = '1' THEN

a <= idle;

ELSIF go = '1' AND reset = '0' THEN

a <= s9;

ELSE

a <= s8;

END IF;

WHEN s9 =>

IF reset = '1' THEN

a <= idle;

ELSIF go = '1' AND reset = '0' THEN

a <= s0;

ELSE

a <= s9;

END IF;

END CASE;

END IF;

END PROCESS;

overflow1 <= '1' WHEN a = s0 ELSE '0';

PROCESS (Lights1, a)

BEGIN

IF

a = idle

THEN Lights1 <= "1000000";

ELSIF

a = s0

THEN Lights1 <= "1000000";

ELSIF

a = s1

THEN Lights1 <= "1111001";

ELSIF

a = s2

THEN Lights1 <= "0100100";

ELSIF

a = s3

THEN Lights1 <= "0110000";

ELSIF

a = s4

THEN Lights1 <= "0011001";

ELSIF

a = s5

THEN Lights1 <= "0010010";

ELSIF

a = s6

THEN Lights1 <= "0000010";

ELSIF

a = s7

THEN Lights1 <= "1111000";

ELSIF

a = s8

THEN Lights1 <= "0000000";

ELSIF

a = s9

THEN Lights1 <= "0011000";

ELSE

Lights1 <= "1111111";

END IF;

END PROCESS;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- tenths of seconds

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROCESS(overflow1, go, reset, b, clock)

BEGIN

IF (clock'EVENT AND clock = '1') THEN

CASE b IS

WHEN idle =>

IF reset = '1' THEN

b <= idle;

ELSIF go = '1' AND reset = '0' AND overflow1 = '1' THEN

b <= s1;

ELSE

b <= idle;

END IF;

WHEN s0 =>

IF reset = '1' THEN

b <= idle;

ELSIF go = '1' AND reset = '0' AND overflow1 = '1' THEN

b <= s1;

ELSE

b <= s0;

END IF;

WHEN s1 =>

IF reset = '1' THEN

b <= idle;

ELSIF go = '1' AND reset = '0' AND overflow1 = '1' THEN

b <= s2;

ELSE

b <= s1;

END IF;

WHEN s2 =>

IF reset = '1' THEN

b <= idle;

ELSIF go = '1' AND reset = '0' AND overflow1 = '1' THEN

b <= s3;

ELSE

b <= s2;

END IF;

WHEN s3 =>

IF reset = '1' THEN

b <= idle;

ELSIF go = '1' AND reset = '0' AND overflow1 = '1' THEN

b <= s4;

ELSE

b <= s3;

END IF;

WHEN s4 =>

IF reset = '1' THEN

b <= idle;

ELSIF go = '1' AND reset = '0' AND overflow1 = '1' THEN

b <= s5;

ELSE

b <= s4;

END IF;

WHEN s5 =>

IF reset = '1' THEN

b <= idle;

ELSIF go = '1' AND reset = '0' AND overflow1 = '1' THEN

b <= s6;

ELSE

b <= s5;

END IF;

WHEN s6 =>

IF reset = '1' THEN

b <= idle;

ELSIF go = '1' AND reset = '0' THEN

b <= s7;

ELSE

b <= s6;

END IF;

WHEN s7 =>

IF reset = '1' THEN

b <= idle;

ELSIF go = '1' AND reset = '0' THEN

b <= s8;

ELSE

b <= s7;

END IF;

WHEN s8 =>

IF reset = '1' THEN

b <= idle;

ELSIF go = '1' AND reset = '0' THEN

b <= s9;

ELSE

b <= s8;

END IF;

WHEN s9 =>

IF reset = '1' THEN

b <= idle;

ELSIF go = '1' AND reset = '0' THEN

b <= s0;

ELSE

b <= s9;

END IF;

END CASE;

END IF;

END PROCESS;

overflow2 <= '1' WHEN b = s0 ELSE '0';

PROCESS (Lights2, b)

BEGIN

IF

b = idle

THEN Lights2 <= "1000000";

ELSIF

b = s0

THEN Lights2 <= "1000000";

ELSIF

b = s1

THEN Lights2 <= "1111001";

ELSIF

b = s2

THEN Lights2 <= "0100100";

ELSIF

b = s3

THEN Lights2 <= "0110000";

ELSIF

b = s4

THEN Lights2 <= "0011001";

ELSIF

b = s5

THEN Lights2 <= "0010010";

ELSIF

b = s6

THEN Lights2 <= "0000010";

ELSIF

b = s7

THEN Lights2 <= "1111000";

ELSIF

b = s8

THEN Lights2 <= "0000000";

ELSIF

b = s9

THEN Lights2 <= "0011000";

ELSE

Lights2 <= "1111111";

END IF;

END PROCESS;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- single seconds

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROCESS(overflow2, go, reset, c, clock)

BEGIN

IF (clock'EVENT AND clock = '1') THEN

CASE c IS

WHEN idle =>

IF reset = '1' THEN

c <= idle;

ELSIF go = '1' AND reset = '0' AND overflow2 = '1' THEN

c <= s1;

ELSE

c <= idle;

END IF;

WHEN s0 =>

IF reset = '1' THEN

c <= idle;

ELSIF go = '1' AND reset = '0' AND overflow2 = '1' THEN

c <= s1;

ELSE

c <= s0;

END IF;

WHEN s1 =>

IF reset = '1' THEN

c <= idle;

ELSIF go = '1' AND reset = '0' AND overflow2 = '1' THEN

c <= s2;

ELSE

c <= s1;

END IF;

WHEN s2 =>

IF reset = '1' THEN

c <= idle;

ELSIF go = '1' AND reset = '0' AND overflow2 = '1' THEN

c <= s3;

ELSE

c <= s2;

END IF;

WHEN s3 =>

IF reset = '1' THEN

c <= idle;

ELSIF go = '1' AND reset = '0' AND overflow2 = '1' THEN

c <= s4;

ELSE

c <= s3;

END IF;

WHEN s4 =>

IF reset = '1' THEN

c <= idle;

ELSIF go = '1' AND reset = '0' AND overflow2 = '1' THEN

c <= s5;

ELSE

c <= s4;

END IF;

WHEN s5 =>

IF reset = '1' THEN

c <= idle;

ELSIF go = '1' AND reset = '0' AND overflow2 = '1' THEN

c <= s6;

ELSE

c <= s5;

END IF;

WHEN s6 =>

IF reset = '1' THEN

c <= idle;

ELSIF go = '1' AND reset = '0' AND overflow2 = '1' THEN

c <= s7;

ELSE

c <= s6;

END IF;

WHEN s7 =>

IF reset = '1' THEN

c <= idle;

ELSIF go = '1' AND reset = '0' AND overflow2 = '1' THEN

c <= s8;

ELSE

c <= s7;

END IF;

WHEN s8 =>

IF reset = '1' THEN

c <= idle;

ELSIF go = '1' AND reset = '0' AND overflow2 = '1' THEN

c <= s9;

ELSE

c <= s8;

END IF;

WHEN s9 =>

IF reset = '1' THEN

c <= idle;

ELSIF go = '1' AND reset = '0' AND overflow2 = '1' THEN

c <= s0;

ELSE

c <= s9;

END IF;

END CASE;

END IF;

END PROCESS;

overflow3 <= '1' WHEN c = s0 ELSE '0';

PROCESS (Lights3, c)

BEGIN

IF

c = idle

THEN Lights3 <= "1000000";

ELSIF

c = s0

THEN Lights3 <= "1000000";

ELSIF

c = s1

THEN Lights3 <= "1111001";

ELSIF

c = s2

THEN Lights3 <= "0100100";

ELSIF

c = s3

THEN Lights3 <= "0110000";

ELSIF

c = s4

THEN Lights3 <= "0011001";

ELSIF

c = s5

THEN Lights3 <= "0010010";

ELSIF

c = s6

THEN Lights3 <= "0000010";

ELSIF

c = s7

THEN Lights3 <= "1111000";

ELSIF

c = s8

THEN Lights3 <= "0000000";

ELSIF

c = s9

THEN Lights3 <= "0011000";

ELSE

Lights3 <= "1111111";

END IF;

END PROCESS;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- tens of seconds

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROCESS(overflow3, go, reset, d, clock)

BEGIN

IF (clock'EVENT AND clock = '1') THEN

CASE d IS

WHEN idle =>

IF reset = '1' THEN

d <= idle;

ELSIF go = '1' AND reset = '0' AND overflow3 = '1' THEN

d <= s1;

ELSE

d <= idle;

END IF;

WHEN s0 =>

IF reset = '1' THEN

d <= idle;

ELSIF go = '1' AND reset = '0' AND overflow3 = '1' THEN

d <= s1;

ELSE

d <= s0;

END IF;

WHEN s1 =>

IF reset = '1' THEN

d <= idle;

ELSIF go = '1' AND reset = '0' AND overflow3 = '1' THEN

d <= s2;

ELSE

d <= s1;

END IF;

WHEN s2 =>

IF reset = '1' THEN

d <= idle;

ELSIF go = '1' AND reset = '0' AND overflow3 = '1' THEN

d <= s3;

ELSE

d <= s2;

END IF;

WHEN s3 =>

IF reset = '1' THEN

d <= idle;

ELSIF go = '1' AND reset = '0' AND overflow3 = '1' THEN

d <= s4;

ELSE

d <= s3;

END IF;

WHEN s4 =>

IF reset = '1' THEN

d <= idle;

ELSIF go = '1' AND reset = '0' AND overflow3 = '1' THEN

d <= s5;

ELSE

d <= s4;

END IF;

WHEN s5 =>

IF reset = '1' THEN

d <= idle;

ELSIF go = '1' AND reset = '0' AND overflow3 = '1' THEN

d <= s6;

ELSE

d <= s5;

END IF;

WHEN s6 =>

IF reset = '1' THEN

d <= idle;

ELSIF go = '1' AND reset = '0' THEN

d <= s7;

ELSE

d <= s6;

END IF;

WHEN s7 =>

IF reset = '1' THEN

d <= idle;

ELSIF go = '1' AND reset = '0' THEN

d <= s8;

ELSE

d <= s7;

END IF;

WHEN s8 =>

IF reset = '1' THEN

d <= idle;

ELSIF go = '1' AND reset = '0' THEN

d <= s9;

ELSE

d <= s8;

END IF;

WHEN s9 =>

IF reset = '1' THEN

d <= idle;

ELSIF go = '1' AND reset = '0' THEN

d <= s0;

ELSE

d <= s9;

END IF;

END CASE;

END IF;

END PROCESS;

PROCESS (Lights4, d)

BEGIN

IF

d = idle

THEN Lights4 <= "1000000";

ELSIF

d = s0

THEN Lights4 <= "1000000";

ELSIF

d = s1

THEN Lights4 <= "1111001";

ELSIF

d = s2

THEN Lights4 <= "0100100";

ELSIF

d = s3

THEN Lights4 <= "0110000";

ELSIF

d = s4

THEN Lights4 <= "0011001";

ELSIF

d = s5

THEN Lights4 <= "0010010";

ELSIF

d = s6

THEN Lights4 <= "0000010";

ELSIF

d = s7

THEN Lights4 <= "1111000";

ELSIF

d = s8

THEN Lights4 <= "0000000";

ELSIF

d = s9

THEN Lights4 <= "0011000";

ELSE

Lights4 <= "1111111";

END IF;

END PROCESS;

END Behavior;

**Clock Divider:**

LIBRARY ieee ;

USE ieee.std\_logic\_1164.all ;

ENTITY clk IS

PORT ( iclk : IN STD\_LOGIC;

oclk : OUT STD\_LOGIC);

END clk;

ARCHITECTURE Behavior OF clk IS

CONSTANT halfcount : POSITIVE := 135000; -- 135000 counts 1/100 second

SIGNAL Count: INTEGER RANGE 0 TO halfcount-1;

SIGNAL clkstate : STD\_LOGIC;

BEGIN

PROCESS ( iclk )

BEGIN

IF (iclk'EVENT AND iclk = '1') THEN

IF Count = halfcount-1 THEN

Count <= 0;

clkstate <= not clkstate;

ELSE

Count <= Count + 1;

END IF;

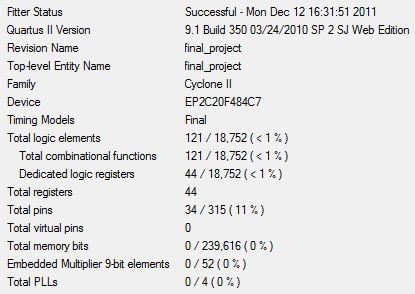
END IF;

END PROCESS;

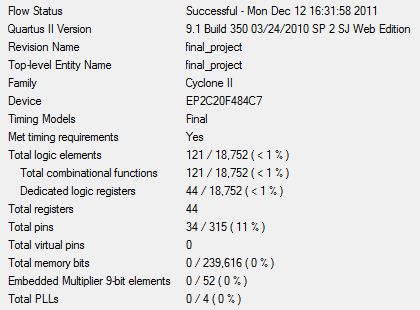
oclk <= clkstate;

END Behavior;

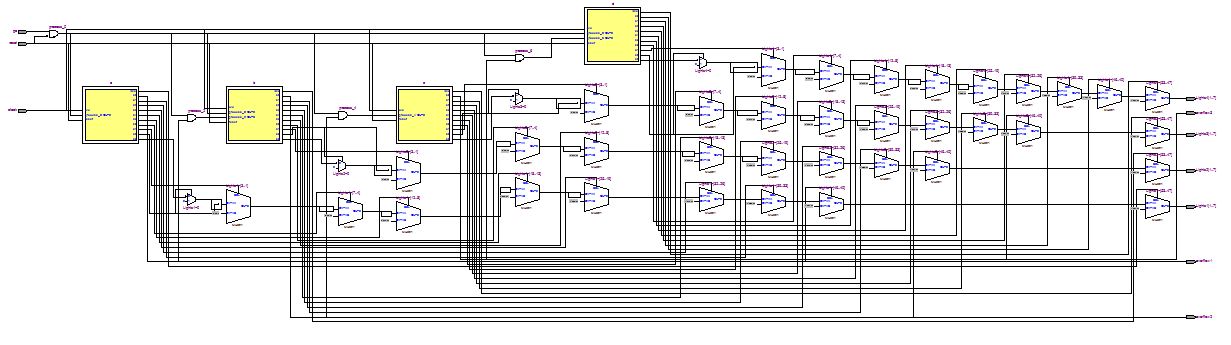
**Fitter Summary:**



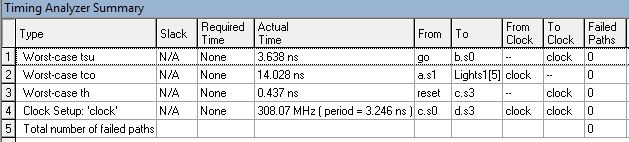
**Flow Summary:**



**RTL Viewer:**



**Timing Summary:**

****