EE4306 VHDL Assignment

Reaction Timer

Sven Richter (0416977629)

September 13, 2004

Contents

1 Introduction

The task was to design a reaction timer which has to fit into the MA4A5 Lattice development boards. The particular features of these boards are that they have three buttons and four seven-segment-displays which can be used to realize the project.

The button SW1 should clear the display and has to start the process. After the RESET button has been pressed the system is supposed to have a four to eight seconds delay (during that the displays stays unchanged) before actually starts counting. As soon as the STOP button (SW3) is pushed the counter freezes the display to show the reaction time.

The display is driven by a 2 kHz clock but the reaction will be shown in milliseconds. If it happens that no one pushes the STOP button the timer tops up to "9999" and after that goes back to "0000".

2 Principles of operation

I decided to use states to realize the reaction counter. To design the program in such a manner gave me the opportunity to define specific events in every state which will trigger a change to the next state without having any effect on it . This rule forces every function to run only in the current state and prevents it from interaction with any other part.

In the next section I will give a short overview over each state and describe his basic functionality and the transition needed to change to the next state.

2.1 The states

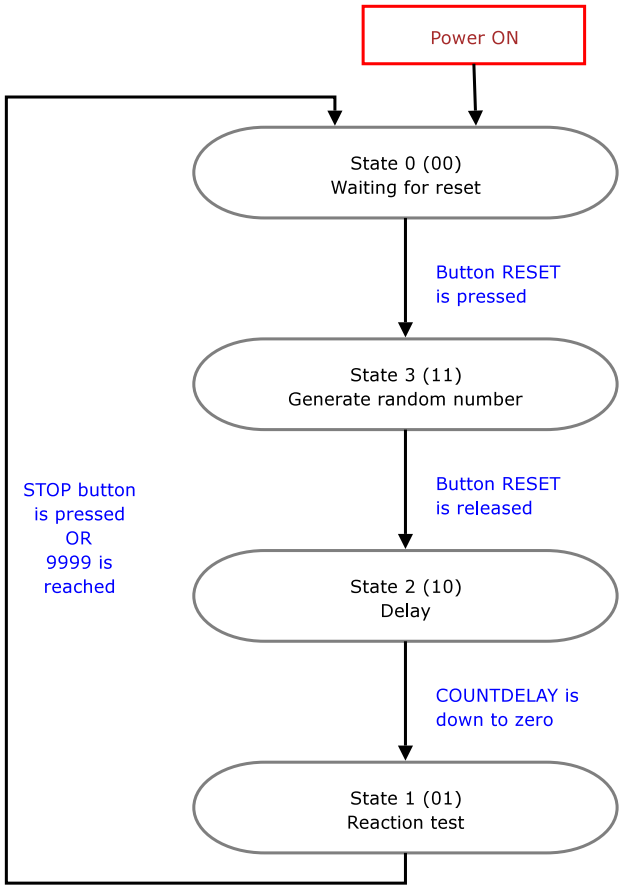

Figure 1: States

2.1.1 State 0 - Waiting for reset

In this state the board will wait till a user presses the RESET button to start the whole cycle. State zero is the first state the board will be in after it is powered on and it will be the state the program changes to after the reaction test is finished.

The only thing which does happen in this state is to show the reaction time result in the seven-segment-display. If it has just been powered on it will show "0000" because nobody did a reaction test and therefore the measured time is zero. The same result will appear if nobody stops the timer before it reaches then seconds.

2.1.2 State 3 - Generate random number

This state measures the time the RESET button is pressed and uses it as a random delay in the next state. Therefore the system will jump into state three after the RESET button is released. Furthermore, the display will be reset to "0000" during this state if any different time is shown on it.

2.1.3 State 2 - Delay

In this state the timer (COUNTDELAY) which was destined in state three will count down. This produces a delay until the test actually starts. When the COUNTDELAY reaches zero it changes to state one.

2.1.4 State 1 - Reaction test

This is the state for the test itself. During the test the seven-segment-display will count up in milliseconds to show the time which has passed. If the user presses STOP the display will freeze and the program will change to state zero.

If the user does not hit the STOP button during the time the four displays need to count up to nine (9.999 seconds) the reaction test will also be stopped and the display will be reset to "0000" and the state will change to zero.

2.2 The program

In figure 2 you can see the schematic design of the VHDL program. The source-code of all three modules is attached in the appendix.

2.2.1 Main module

The main module ReactionTimer (section A.1) does the main function of the VHDL program. It switches between the states which are briefly described in section 2.1. To check the conditions for a change from one state to another it uses the the RESET and STOP input which are connected to pin 31 (SW3 - STOP) and pin 9 (SW1 - RESET). It also uses the clk input (pin 11) to measure the time the RESET button (SW3) is pressed (in state 3, section 2.1.2) and to count the delay (state 2, section 2.1.3) down.

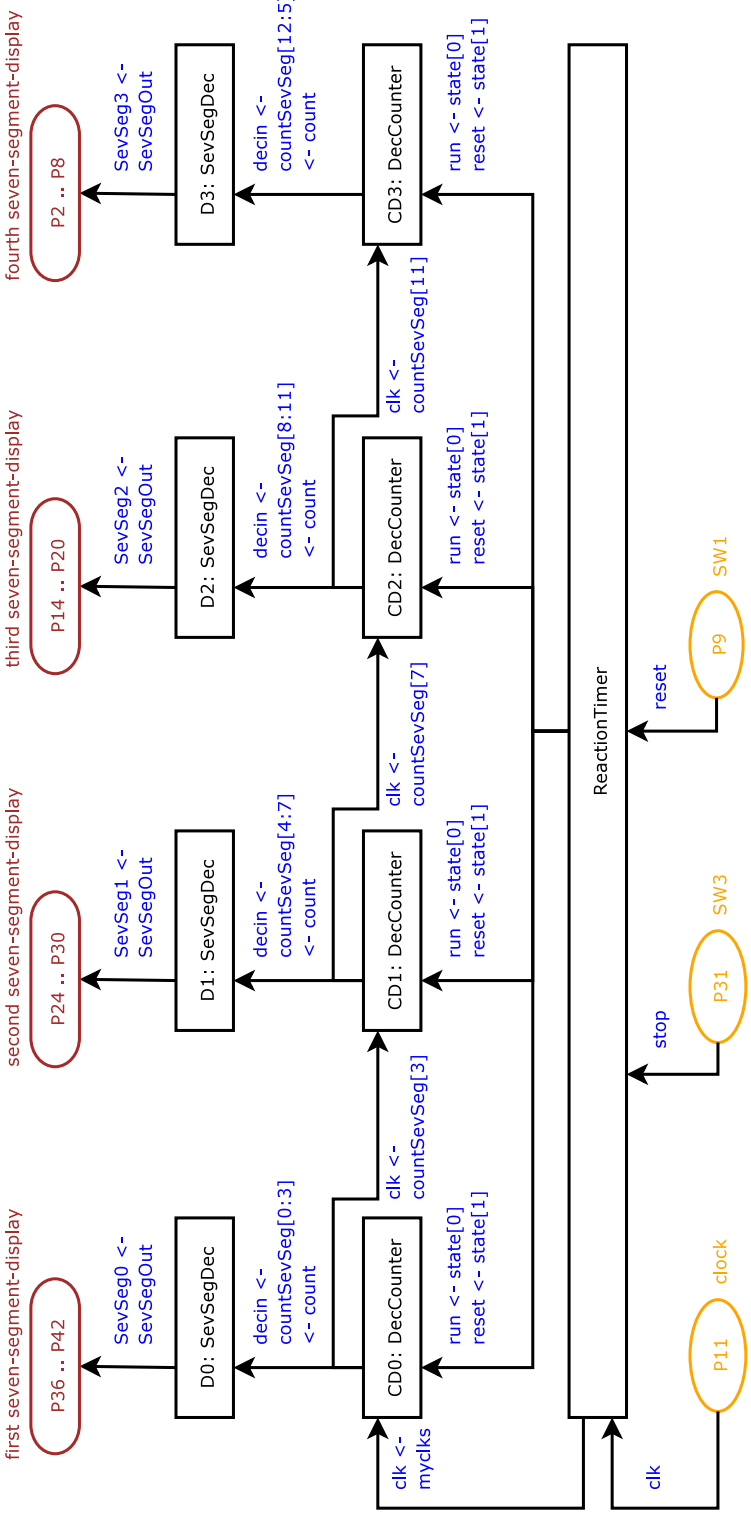

Figure 2: Program design

To generate a random number for the delay the time the RESET button is pressed is meassured. To do this the bits 4 to 12 of COUNTDELAY are used. Because this is a 9 bit counter it needs about $\frac{1}{4}$ of a second ($\frac{1}{2kHz}(2^9 - 1) = 0.255sec$) to be back at zero. As the maximum time the RESET button is pressed is assumed to be about 0.5 seconds, it should give us a random number of a haphazard value. Any difference of 0.5 milliseconds in the pressing time will thus result in a change of the delay time of 16 milliseconds.

To ensure a minimum delay of about four seconds the most significant bit of COUNTDELAY is set to one ($\frac{1}{2kHz}2^{13} = 4.096sec$). Then in state two (section 2.1.3) after the RESET button is released COUNTDELAY starts counting down till it reaches zero and initiate the start of the reaction test.

2.2.2 Decimal counter and seven-segment-display

The decimal counter, DecCounter module (section A.2), is driven by a 1 kHz clock (MYCLK) which is simply the system clock (CLK) divided by 2. Therefore MYCLK is inverted every time CLK has a rising edge. In contrast to most of the common counters the decimal counter does not work on the rising edge of a clock signal but counts up on the falling edge. This makes it possible to connect the next decimal counter on the most significant bit of the four bit count output.

The decimal counter has also a run and reset input. If the reset is equal one the counter will reset its output to zero even if the clock signal does not change. But if the run signal is not set to true the counter will freeze the current counted number even if the clock signal changes. These inputs make it possible to connect the counter to a steady clock signal and direct to the corresponding seven segment display drivers. Every DecCounter module is plugged to one SevSegDec module (section A.3) which itself is connected to a seven-segment-display, therefore all counted values will be directly shown on a display. Because the counter is driven by an 1 kHz clock the displayed time is given in milliseconds.

The run and reset inputs of the counter are equal to the current state so they are connected and produce the expected result. In state three (section 2.1.2) and state two (section 2.1.3) the display is reset, in state one (section 2.1.4) the clock runs and in state zero (section 2.1.1) it is stopped and shows the measured time.

3 Results of simulation

To ensure that the program is working I first tested both single modules and afterwards the complete program. To produce these waveforms the code was changed twice to make the delay displayable. Both changes are done in the ReactionTimer module (section A.1) and commented. First, not tie bit 4 to 12 of the COUNTDELAY but the bit 0 to 9 are used to measure the time RESET is pressed. Second, the most significant bit of COUNTDELAY was not set to one.

3.1 SevSegDec

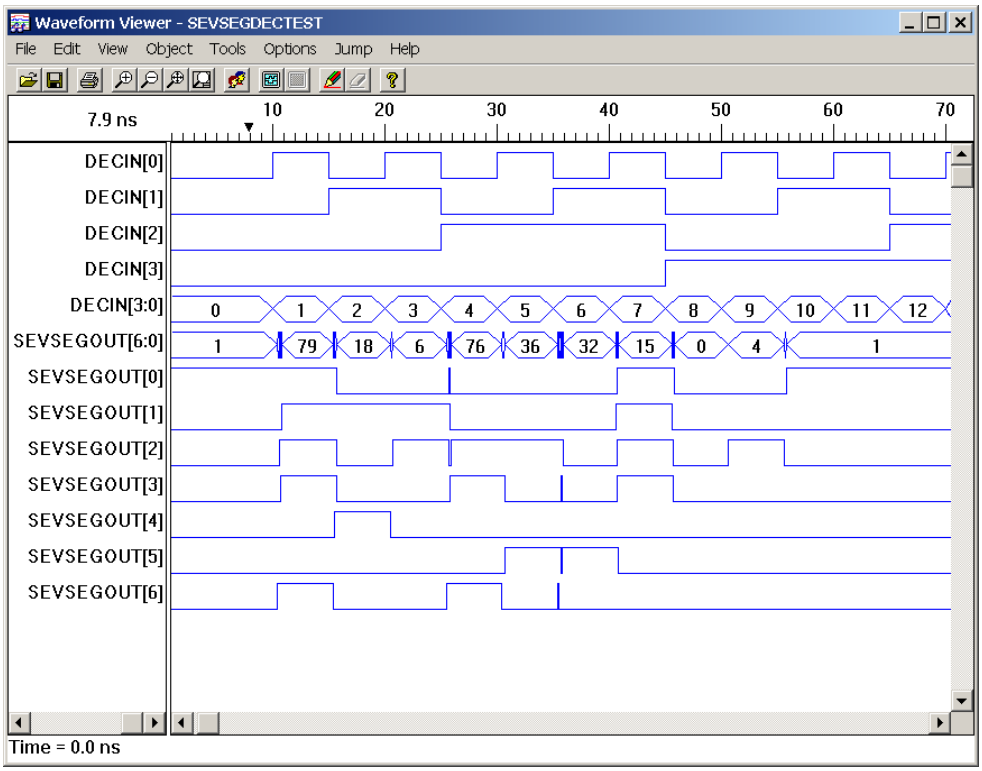

Figure 3: SevSegDec waveform

The waveform of the seven-segment-display driver is shown in figure 3. For each value of the 4 bit input the display driver gave an output value. Because of a decimal bus radix it showed the decimal representation of the output bit sequence which will show the number in the display (see following table)

input	bit sequence	decimal value
0	0000001	1
1	1001111	79
2	0010010	18
3	0000110	6
4	1001100	76
5	0100100	36
6	0100000	32
7	0001111	15
8	0000000	0
9	0000100	4

It should never occur in the running program that the input becomes greater than 9 but I wanted to check if the display will, as expected, show the zero in such a case.

3.2 DecCounter

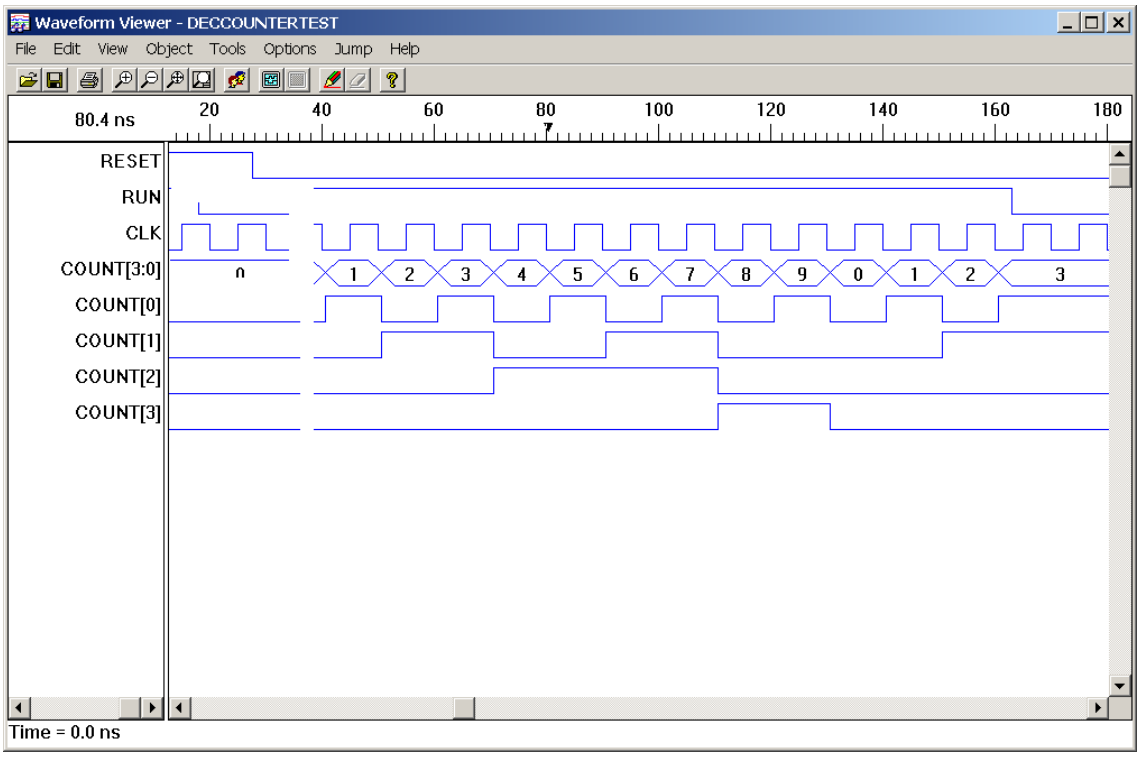

Figure 4: DecCounter waveform

The resulting waveform for the decimal counter test is shown in figure 4. It behaved like expected: counted up at the falling edge of the clock from 0 to 9, reset if reset is equal one and only counted if run is true. If run and reset were zero it held the current value.

3.3 ReactionTimer

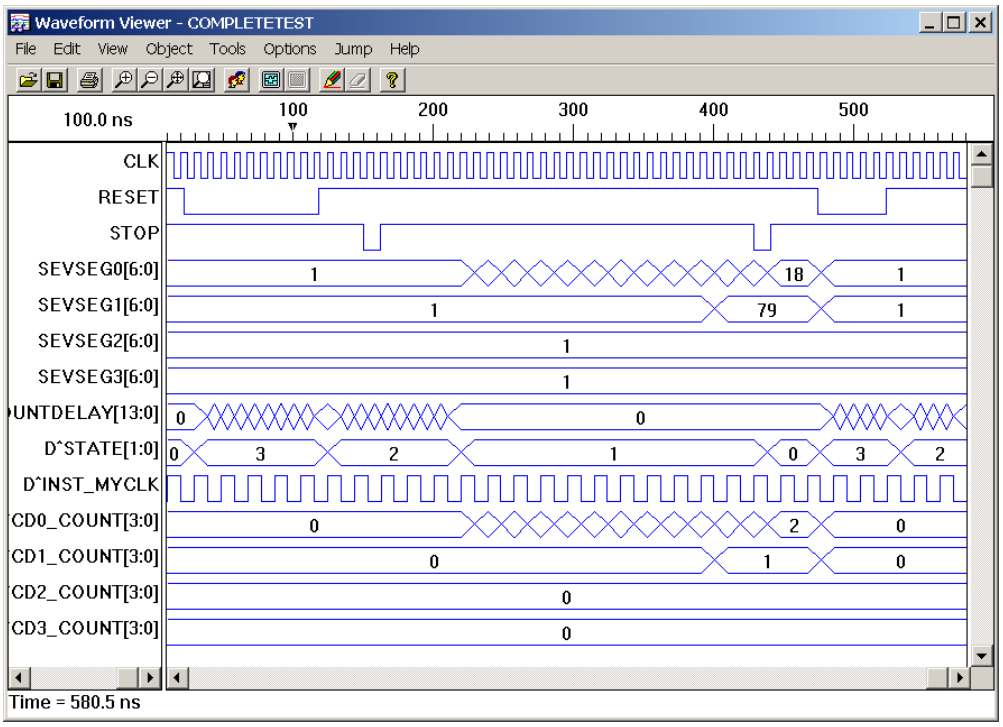

Figure 5: ReactionTimer waveform - complete run

In figure 5 the complete cycle is shown. You can see how the change from one state to another is performed if the conditions complies.

In figure 6 the random number generation process is shown. It started when RESET was pressed and changed the state from state zero to state 3. Ones sees how COUNTDELAY counted up as long as the RESET button is pressed and how the state changed from three to two when it was released. In state two COUNTDELAY counted down. How it reached zero is shown in figure 7. I have added a low pulse for the STOP button to simulate that the user presses this button during the delay to test if it has any effect. As expected this does not change anything.

The real program would need much more clock cycles to reach this state but because two lines of code were modified for the waveform test it was displayed.

When the system reached state one the test started till the RESET button was pressed. One can see how the seven-segment-display starts counting up. I have stopped the test very soon because the SevSegDec module was already been tested but one can see how the divided clock signal triggered the first counter and the first counter the second one.

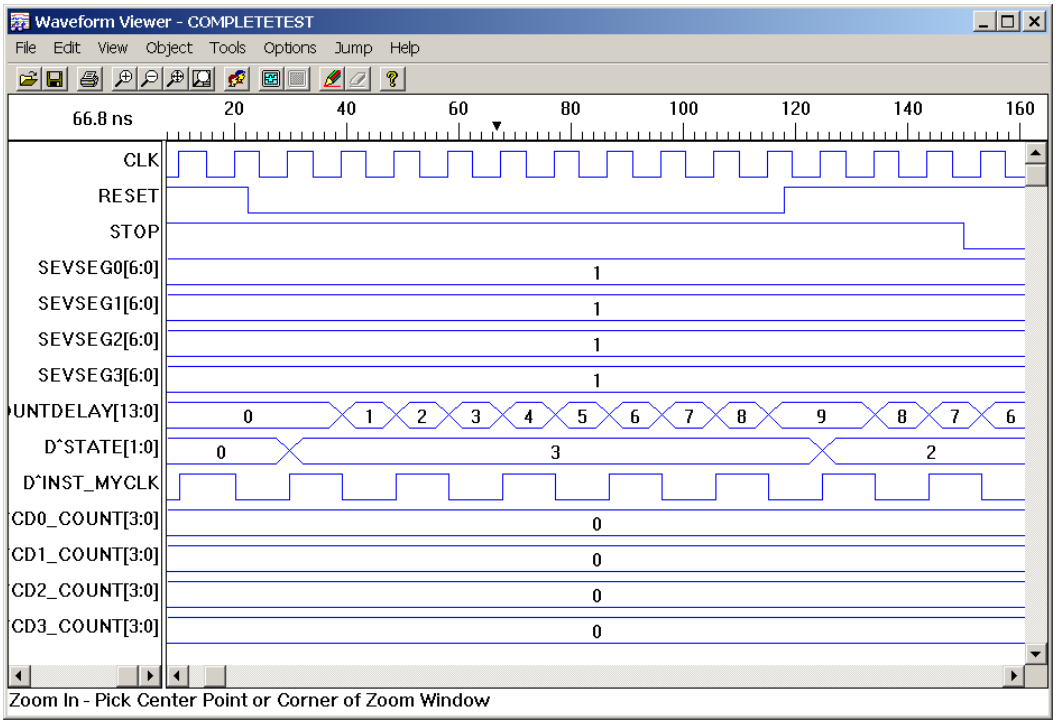

Figure 6: ReactionTimer waveform - countdelay counts up

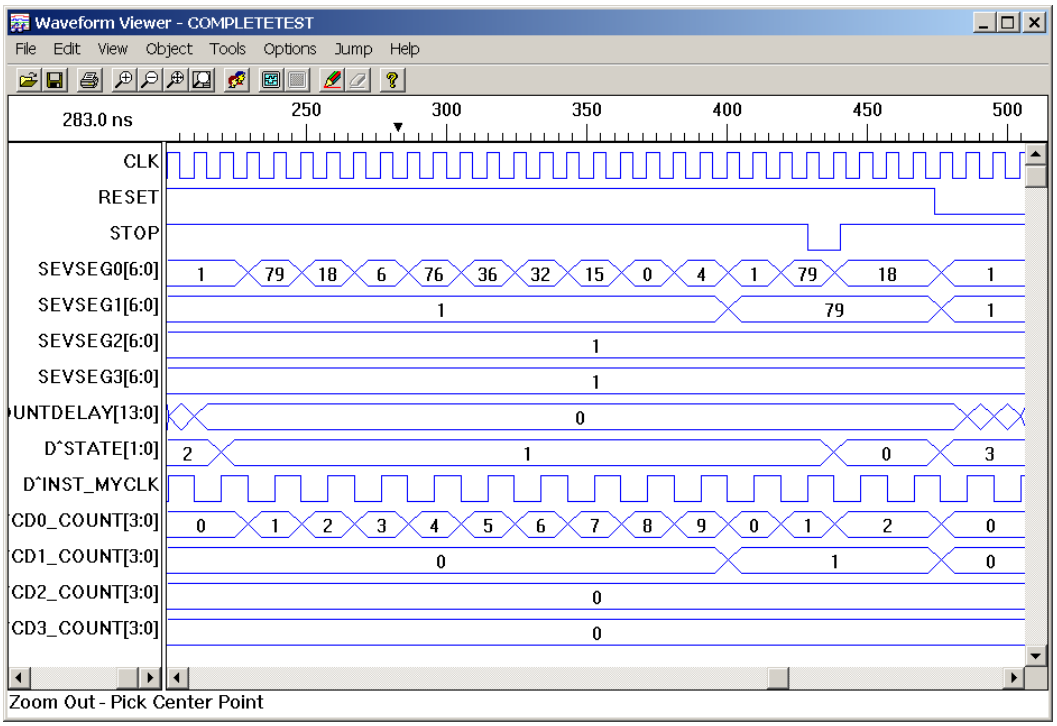

Figure 7: ReactionTimer wavefrom - delay and test

After the STOP button was pressed the system entered state zero again and the display still showed the measured time. I did not want to test if the display reset to zero after 10 seconds as a waveform because this would need to many clock cycles to be reasonable. It was quite better to test it on the Lattice test board itself, because there I could let the timer run to 9900 and then changed to a slower clock frequency. And it did work as expected.

A Source code

The source code compiles perfectly on the **Simplify** compiler and just returns an expected warning that a few inputs are set to "don't care" and therefore are not set with a discrete value. But I have expected it because I defined that it should do that in the optimization constrains.

A.1 ReactionTimer

```
-- ------------------------------------------------------

--    File: reactiontime.vhd
--
--    Assignment 2004 Semester 2
--    Sven Richter (0416977629)
--
--    Reactioncounter .. after reset (sw1) starts
--    the clock till sw3 is pressed and shows the
--    passed miliseconds in the seven segment
--    display (decimal)
--
--    It uses deccounter for decimal counting
--     and sevensegdec for dispalying
--
-- ------------------------------------------------------


    library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

    entity ReactionTimer is
   port (clk: in std_logic; -- 2 kHz clock
      reset :in std_logic; -- the reset button
      stop :in std_logic; -- reaction meassure button
      SevSeg0 :out std_logic_vector (6 downto 0);
      SevSeg1 :out std_logic_vector (6 downto 0);
      SevSeg2 :out std_logic_vector (6 downto 0);
      SevSeg3 :out std_logic_vector (6 downto 0)
   );


    attribute loc: string;
    attribute loc of clk:   signal is "P11";
    attribute loc of stop:   signal is "P31"; -- button SW3
    attribute loc of reset:   signal is "P9"; -- button SW1
    attribute loc of SevSeg3:   signal is "P2 P3 P4 P5 P6 P7 P8";
    attribute loc of SevSeg2:   signal is "P14 P15 P16 P17 P18 P19 P20";
    attribute loc of SevSeg1:   signal is "P24 P25 P26 P27 P28 P29 P30";
    attribute loc of SevSeg0:   signal is "P36 P37 P38 P39 P40 P41 P42";

end;
```

```
architecture ReactionTimer_arc of ReactionTimer is

  component DecCounter
    port (clk :in std_logic;    -- the clock signal
       run : in std_logic;
       reset: in std_logic;
       count :inout std_logic_vector (3 downto 0)    -- the output value
    );
  end  component;


  component SevSegDec
    port (decin:in std_logic_vector (3 downto 0);
       SevSegOut :out std_logic_vector(6 downto 0)
    );
  end  component;


   signal state : std_logic_vector (1 downto 0) := "00"; -- tells which state it is in and
sets the
                        -- the run and reset of the deccounter
   signal countdelay: std_logic_vector (13 downto 0); -- counter of the delay should be
enough for max 8 second on 2khz
    signal countSevSeg: std_logic_vector (15 downto 0); -- the counter result for the
cascaded decimal counter

       signal myclk: std_logic; -- the clock for the decimal counter 1/2*clk = 1 khz ..
for useless count result

begin

      -- map the dec counter to the seven segment displays
  D0: SevSegDec port map (countSevSeg(3 downto 0), SevSeg0);
  D1: SevSegDec port map (countSevSeg(7 downto 4), SevSeg1);
  D2: SevSegDec port map (countSevSeg(11 downto 8), SevSeg2);
  D3: SevSegDec port map (countSevSeg(15 downto 12), SevSeg3);

  CD0:DecCounter port map (myclk, state(0), state(1), countSevSeg(3 downto 0));
  CD1:DecCounter port map (countSevSeg(3), state(0), state(1), countSevSeg(7 downto
4));
  CD2:DecCounter port map (countSevSeg(7), state(0), state(1), countSevSeg(11 downto
8));
  CD3:DecCounter port map (countSevSeg(11), state(0), state(1), countSevSeg(15 downto
12));

       run: process begin
      wait until rising_edge(clk);

          -- produce the clock for the Decimal Counter
      -- os one clock puls is a millisecond
      myclk <= not myclk;

          -- decide what to do dependend on the current state
      case0: case state is
```

```
        when "11" => -- we produce the random number
          if (reset = '0') then -- reset button pressed
            -- counter needs 1/4 second to recycle
            -- should be randon enough
            countdelay(12 downto 4) <= countdelay(12 downto 4) + "000000001";
            --countdelay(9 downto 0) <= countdelay(9 downto 0) + "0000000001";
-- use for testing
          else
            -- put a one into the most significant bit to
            -- ensure at least a 4 seconds delay
            countdelay(13) <= '1'; -- remove for testing
            state <= "10"; -- next step
          end if;
        when "10" => -- use the random number and count down
          if (countdelay > "000000000000") then
            countdelay <= countdelay - "0000000000001";
          else
            state <= "01"; -- next step
          end if;
        when "01" => -- measure the time and show it
          if (countSevSeg(15) = '0' and countdelay(0) ='1') then
            -- if the counter reaches 0000
            state <= "00";
          elsif ( stop = '1' ) then
            -- to remember is countSevSeg was 999
            if countSevSeg(15) = '1' then
              countdelay(0) <= '1';
            else
              countdelay(0) <= '0';
            end if;
          else
            -- if the users presses stop
            state <= "00";

          end if;
        when others =>
          if (reset = '0') then -- wait until rest ist pushed
            state <= "11"; -- start again
          end if;

      end case case0;
    end process run;

    end ReactionTimer_arc;
```

A.2 DecCounter

```
-- ------------------------------------------------------------

--   File: deccounter.vhd
--
--   Assignment 2004 Semester 2
--   Sven Richter (0416977629)
--
--   a decimal counter
--
-- ------------------------------------------------------------


    library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

    entity DecCounter is
   port (clk :in std_logic;    -- the clock signal
      run : in std_logic;    -- 1 - run, 0 - stop
      reset : in std_logic;    -- if '0' reset the counter
--       carry: out std_logic;    -- carry out if back to zero
      count :inout std_logic_vector (3 downto 0)    -- the output value
   );
end;

    architecture DecCounter_arc of DecCounter is
begin
   run: process(run, clk, reset) begin
      if (reset = '1') then
            count <= X"0";
      elsif ( falling_edge(clk)) then
         -- had to cascade because leonardo wants the
         -- clk statement allone
         if (run = '1') then
            if ( count < X"9") then
               count <= count + X"1";
            else
               count <= X"0";
            end if ;
         end if;
      end if;
   end process run;

    end DecCounter_arc;
```

A.3 SevSegDec

```
-- ------------------------------------------------------------

--   File: sevsegdec.vhd
--
--   Assignment 2004 Semester 2
--   Sven Richter (0416977629)
--
--   just a simple Module to show a decimal
--   value at the Seven Segment Display
--
-- ------------------------------------------------------------


    library ieee;
use ieee.std_logic_1164.all;

    entity SevSegDec is
   port (decin:in std_logic_vector (3 downto 0);
      SevSegOut :out std_logic_vector(6 downto 0)
   );
end;

    architecture SevSegDec_arc of SevSegDec is
begin
   process (decin) begin
      case0: case Decin is
         -- need to have inverse output because
         -- the driver negates it
         when X"1" => SevSegOut <= "1001111"; -- 1
         when X"2" => SevSegOut <= "0010010"; -- 2
         when X"3" => SevSegOut <= "0000110"; -- 3
         when X"4" => SevSegOut <= "1001100"; -- 4
         when X"5" => SevSegOut <= "0100100"; -- 5
         when X"6" => SevSegOut <= "0100000"; -- 6
         when X"7" => SevSegOut <= "0001111"; -- 7
         when X"8" => SevSegOut <= "0000000"; -- 8
         when X"9" => SevSegOut <= "0000100"; -- 9
         when others => SevSegOut <= "0000001"; -- 0
      end case case0;
   end process ;
end SevSegDec_arc;
```