ENEL 373 Project Report

FPGA Programmable Counter & PWM Waveform Generator.

Andy Everitt – 147 319 94

Greg Bates – 728 761 22

A. Introduction

This report outlines the design process and implementation required to create a user programmable down counter and PWM generator on a Nexys-4DDR FPGA board. To achieve this the software tool Vivado was used which allowed for synthesis and analysis of VHDL code. The first milestone for this assignment was to develop the 16-bit down counter which counted at a frequency of 1000 Hz. Furthermore, the counter was made to be reprogrammable for the first milestone with 16 input switches altering the value at which the counter counts down from.

Achieving the entire assignment meant building upon Milestone 1. This meant modifying the down counter so that it could be used as a PWM generator. A Finite State Machine (FSM) module was also implemented and other more minor changes to improve functionality. The final product also allowed four different clock speeds to be used by the counter; the user was also given the ability to select the desired clock speed, which varied the rate of counting. For our final design, three FSM were applied, which allowed the user to cycle through clock speed and counter output settings, as well as programming either the period or duty cycle of the counter using the same 16 input switches. All the design choices and technical information will be further explained throughout this report.

B. Top Level Design

Initially the design was set on completing milestone 1 which mainly focused upon the 16-bit down counter. As milestone 1 was completed the attention then shifted to the side components needed to complete the assignment. As seen from Figure 1 the seven-segment display was used to display the count value. Although this was not a part of the criteria it was useful for debugging purposes as the current counter value could be seen when testing the implementation of the code on the FPGA.

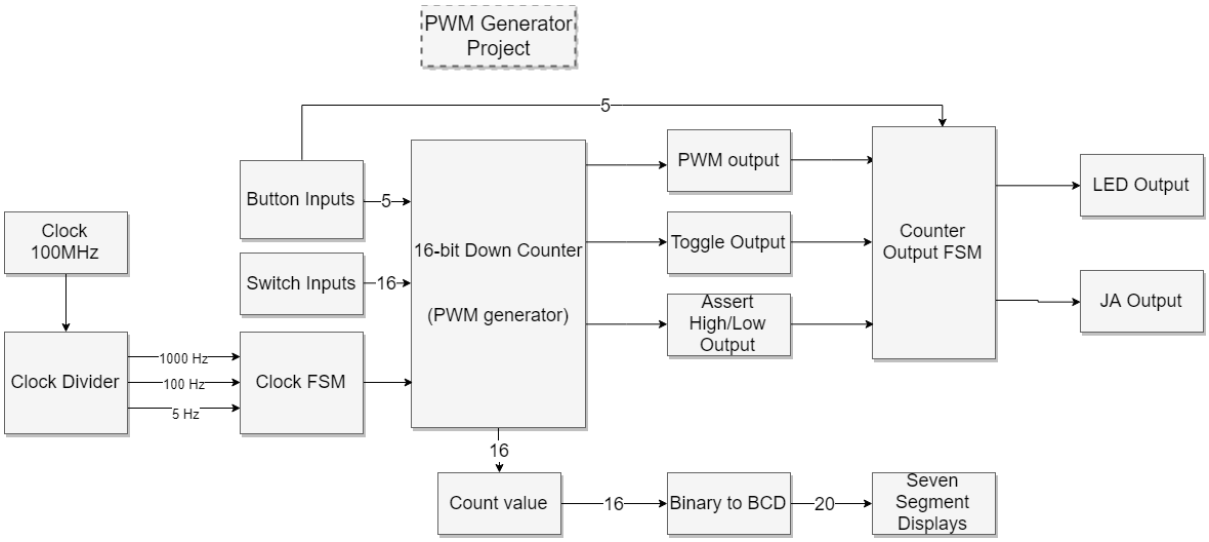

Figure 1 - Top level design.

C. Design Summary

C.1. Top Level Design

The top-level module, called “main_pwm”, contains the main structure for generating the 16-bit down counter and PWM generator. This module was structurally coded to allow simple components to be used repeatedly with changes to the generic and port mapping, such as the clock divider which is used three times to create each of the programmable clock speeds. Structural coding connects all the different components together in a simple way. Through using a variety of signals generated by each implementation of the clock divider, we were able to select the desired speed using a FSM that cycled upon a button press. This in turn was sent into the 16 bit-down counter which included the PWM generator. The count out value is sent to both the binary to binary coded decimal (BCD) module and the counter output FSM before their signals are outputted on the seven-segment displays and RGB LEDs respectively.

The counter out FSM was implemented as a process in the top level using behavioural coding, despite being bad practice to combine structural and behavioural code in the same module, in this situation it was not practical to implement the assert High/Low output in the counter module since the signal would be reset when the counter value got reloaded. The amount of behavioural code in the top-level file was kept to a minimum.

The user inputs, and outputs of the project are shown in Figure 2. The “left” button (BTNL) cycles through the available clock speeds using another FSM, this clock is then fed into the counter to control the count rate. A constant 1000 Hz clock is used to sync all the other processes within every other module such as button debouncing and displaying the counter value on the seven segment displays, it was found that if the clock speed was too high or low for these processes then they did not operate reliably, multiple button presses would be registered, and the display would flicker for example. The “down” button (BTND) cycles through the possible outputs generated by the counter module. The “right” button (BTNR) asserts a reset signal HIGH in the top level that is propagated through all the components and primarily resets the counter to zero which allows it to immediately load the currently stored period value on the next clock cycle. LEDs were used to give a visual indication of the current states the project is in and its output, however a more accurate signal is provided by ports JA(1) for the clock and JA(2) for the counter output.

The different modules previously mentioned get explained in depth through the following subsections.

C.2. 16-Bit Down Counter (PWM generator)

The 16-bit down counter was implemented through behavioural programming. This module not only preformed the counting required for the assignment but also had the function to change duty cycle and period upon receiving user inputs illustrated in Figure 2. These processes were introduced into the counter module instead of being in the top level or another module because having all the signals influenced by the switches within the same component of the circuit simplifies the design.

To be able to change both the period and duty cycle with the same switches a two-variable state machine was introduced into the module. This was controlled by the up button (BTNU) selecting states and the centre button (BTNC) setting the duty cycle or period upon pushing. At any given time, the period and duty cycle can be seen from the row of 16 LEDs and the state from the blue LED 17.

Figure 2 - FPGA user input and output diagram.

The counter module outputs four main signals, the PWM waveform, a toggle output that inverts every time the counter reaches zero, and assert HIGH/LOW that sets the output HIGH/LOW respectively after the counter reaches zero and stays there indefinitely until the output state is changed.

The PWM output is calculated from the period, duty cycle, and current count value using the following formula and asserting the output HIGH is the condition is true.

$$count < \frac{period * duty}{100}$$

This was implemented in VHDL as shown below. The divide by 100 was needed to be moved to the left-hand side as it is much easier to do multiplication on binary than division. The "11111111" represents the desired value for 100% duty cycle, i.e. 255.

```
pwm_out: process (count)
begin
   if ("11111111" * count) < (period * duty(7 downto 0)) then
      pwm <= '1';
   else
      pwm <= '0';
   end if;
   output <= pwm;
end process pwm_out;
```

C.3. Finite State Machine

The Finite State Machine contains four states denoted from “00” through to “11” counted in binary. Three FSMs are used in this project, one for changing the clock speed and the other for counter output. The clock speed can vary between 5Hz, 100Hz, 1000Hz and pause through pressing the left button. It is shown what the current state is from the rate at which the counter counts down as shown on the seven-segment display.

For the counter, the FSM allows for the output state to switch between the PWM, assert high, toggle, and assert low. The current state is displayed through LED 16 which changes colour from Red, Blue, Green and all colours in respect to the order of state previously mentioned.

One-hot encoding was used when programming the FSMs as the FPGA has enough flip flops to allow for one per state. Enumeration and case statements were used to implement the FSM which optimises the schematic by utilising a single multi-input multiplexer and three registers for the current state, next state, and output state as shown in Figure 7. Another possible way to create a FSM is to use “if” statements. This would be inefficient as VHDL would create a series of two input multiplexers which increases the propagation delay of the circuit.

The FSM is shown Figure 3.

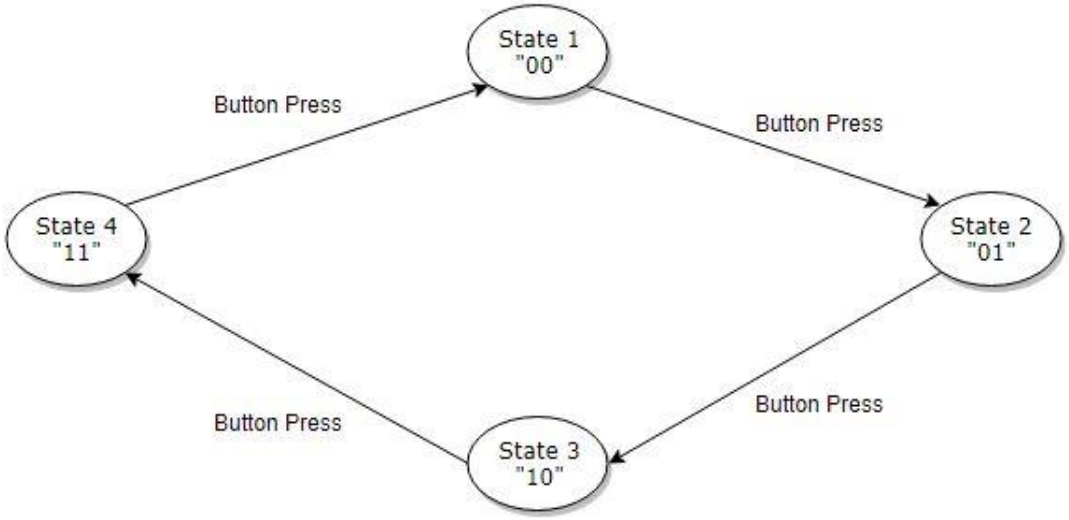

Figure 3 – 4 input Finite State Machine.

For the clock FSM, since four states were available and only three clock speeds were required for the project specification, a fourth clock speed was included that used the 100 MHz hardware clock. This had the side effect of pausing the counter, it was suspected that this was due to the propagation delay of the counter circuit preventing it from updating at the clock rate and therefore freezing. This was however a useful feature as it allowed for some debugging of other components such as loading the period and duty cycle without having the counter decrement. Another possible explanation would be that the seven segment displays were only being updated at 1000 Hz so could not update in time however the output read from the JA(2) pin remained constant indicating that this is not the case.

C.4. Button Debounce

Button debouncing is achieved through using a clock to check for a rising edge and three buffer signals. The button press is then assigned to these signals one at a time over an interval of 3ms (storing each value in a flip flip). Each value can then be put through a logic circuit to ensure the button has in fact been pressed instead of noise or other interruptions leading to an erroneous button press.

This code was obtained from the Vivado Language Templates.

C.5. Clock Divider

The clock divider used was Blair Bonnetts. This module was used as it allowed easy manipulation of the input frequency, set by the hardware clock on the FPGA, and the output frequency using generic mapping in the top-level file. Three clock speeds were created that could be selected by the user. These were 1000 Hz, 100 Hz, and 5 Hz; as mentioned previously, the 100 MHz hardware clock could also be selected however this causes the counter to pause.

C.6. BCD & Seven Segment Display

The BCD module was based off code obtained from an online source [1]. It converts the 16-bit binary counter value into a 20-bit binary coded decimal (BCD) value, this can then be split into 5x4 bits that can easily be displayed on the seven segment displays using the "multiplex_seven_seg" module. This module cycles through each anode, display, and writes the relevant BCD bits to it before moving onto the next display. An update rate of 1000 Hz was found to work well for this, if anything higher was used then the display flicker. Both the binary to BCD module and the BCD to seven segment display module were required to show the count value on the seven-segment displays which was extremely useful for debugging purposes.

D. Testing

Testing and debugging was made significantly easier with the use of checking values or certain expected events with LEDs. By simply assigning signals to LEDS or turning them on when an "if" statement has occurred made stepping through and seeing how the code is working much easier. An oscilloscope was used later in the testing to ensure the correct duty cycle and frequency were being used for the PWM output.

One module which was tested through simulation was the FSM, and the down counter in milestone 1. Through coding a Testbench, we could see how the waveforms looked and compare them to the expected. It is shown from Figure 4 that the state changes are working correctly with each of the four states being selected sequentially upon a button press.

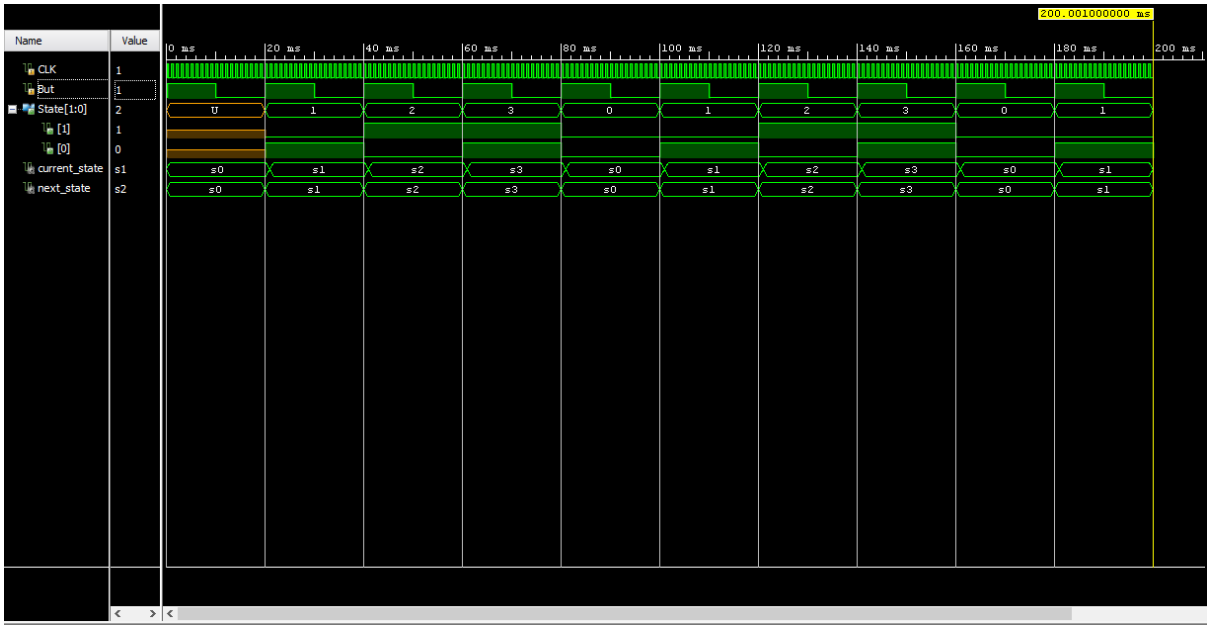

Figure 4 - Testbench waveform for the FSM.

E. Problems

A problem experienced within the 16-bit down counter section of the program was making it so that only the first 8 switches can influence the duty cycle. On top of this the duty cycle had to be inputted as a percentage instead of an 8-bit binary. To fix the switches problem, only switches 7-0 were read when in the state for changing the duty cycle. This worked effectively as there was no chance in a value greater than 8-bits being inputted. Since the LEDs above switches 15-8 would remain unchanged when the duty cycle signal was assigned to them, 8 zeros were concatenated with the duty cycle to make it a 16-bit signal which ensured LEDs 15-8 never displayed an erroneous value. Using the duty cycle as a percentage was achieved as described in section C.2. It was decided that 255 would represent 100% and 0 would represent 0% as this allowed for the highest resolution of programmable duty cycles. The LEDs above the switches were programmed to update when the switch state changed, this is shown below.

```
-- Shows the currently loaded period or duty on the LEDs above the switches.
-- LED17_B is on when showing duty, and off when showing period.
show_PWM: process(SW_State, BTNC)
   begin
      if (rising_edge(clk1000hz)) then
         if SW_State = '0' then
            LED <= period;
            LED17_B <= '0';
         elsif SW_State = '1' then
            LED <= duty;
            LED17_B <= '1';
         end if;
      end if;
   end process show_PWM;
```

A number of problems occurred with the counter outputs. It was found that the toggle could only be achieved if the signal was inverted in the same process as the counter as shown below.

```
if count = 0 then
   count <= period; -- restarts counter after reaching 0
   count_zero <= '1'; -- HIGH only while count = 0
   toggle <= not toggle; -- T flip flop to output toggle waveform
else
   count <= count - 1; -- decremment counter
   count_zero <= '0';
end if;
```

The count_zero signal was used to implement the assert HIGH/LOW output. This was achieved using a latch in the top level, if this was included in the counter module then the output would only pulse HIGH/LOW while count = 0 instead of remaining HIGH/LOW.

```
-- Creates a latch that asserts the output HIGH indefinitely until PWM_State changes after the counter --
reaches 0
if (count_zero = '0' and output_tmp = '0') then
   output_tmp <= '0';
   LED17_R <= '0';
   JA(2) <= '0';
else
   output_tmp <= '1';
   LED17_R <= '1';
   JA(2) <= '1';
end if;
```

F. Project Feedback

VHDL was an interesting computing tool as it showed how computing processes are implemented at a hardware level. Taking a top down approach towards this project increased our understanding as coding was something which we felt more comfortable doing on a larger level rather than designing a circuit of this scale. It is fascinating seeing how closely the hardware and software are linked within an FPGA something that we don't get the same sense of appreciation for working on a PC.

In future we would recommend the code for using the seven-segment display is given as this was incredibly useful for debugging, for having another physical thing to see change and have an influence of increases the experience. On top of this having Testbenches being a priority back in Milestone 1 may encourage people to use this tool at an early stage instead of waiting to make it in the report writing process for this defeats the purpose of testing if it's to be done once completed.

G. Conclusion

Designing a programmable 16-bit down counter and PWM waveform generator posed some challenges but was an ultimately rewarding experience. This project involved learning VHDL code and how to operate Vivado to successful implement a working program onto Nexys 4DDR FPGA board. Through modularising the code into multiple sections debugging was made easier and a simpler RTL schematic are created. This tactic ultimately improved the learning experience as having a top level design shows how all the different components combine to form the 16-bit counter.

H. References

[1] Admin, 'VHDL Code for Binary to BCD converter', *All About FPGA*, 30-Nov-2014. [Online]. Available: https://allaboutfpga.com/vhdl-code-for-binary-to-bcd-converter/. [Accessed: 22-May-2018].

I. Appendices

I.1. Testbench Code.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FSM_TB is
--  Port ( );
end FSM_TB;

architecture Behavioral of FSM_TB is

component Finite_State_Machine is
   Port ( CLK : in STD_LOGIC;
          But : in STD_LOGIC;
          State : out STD_LOGIC_VECTOR(1 downto 0)
          );
end Finite_State_Machine;

type state_type is (s0, s1, s2, s3); --Four states

signal CLK : STD_LOGIC := '0';
signal But : STD_LOGIC;
signal State : STD_LOGIC_VECTOR(1 downto 0);

begin

UUT : Finite_State_Machine
   port map(CLK => CLK, But => But, State => State);

process is
begin

   CLK <= not CLK after 1ms;

   But <= not But after 10ms;

end process;

end Behavioral;
```

I.2. RTL Schematic.

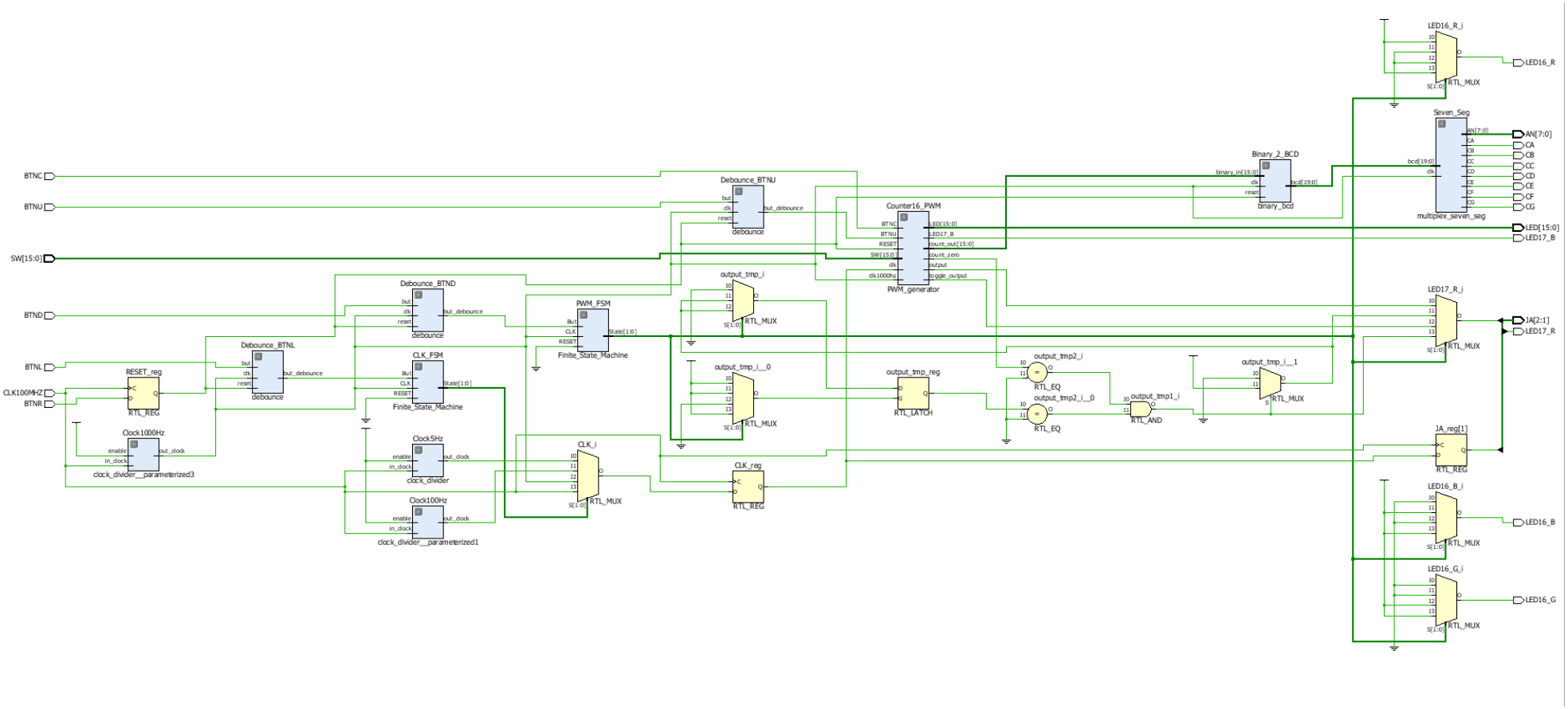

Figure 5 - main_pwm.vhd top level structural schematic.

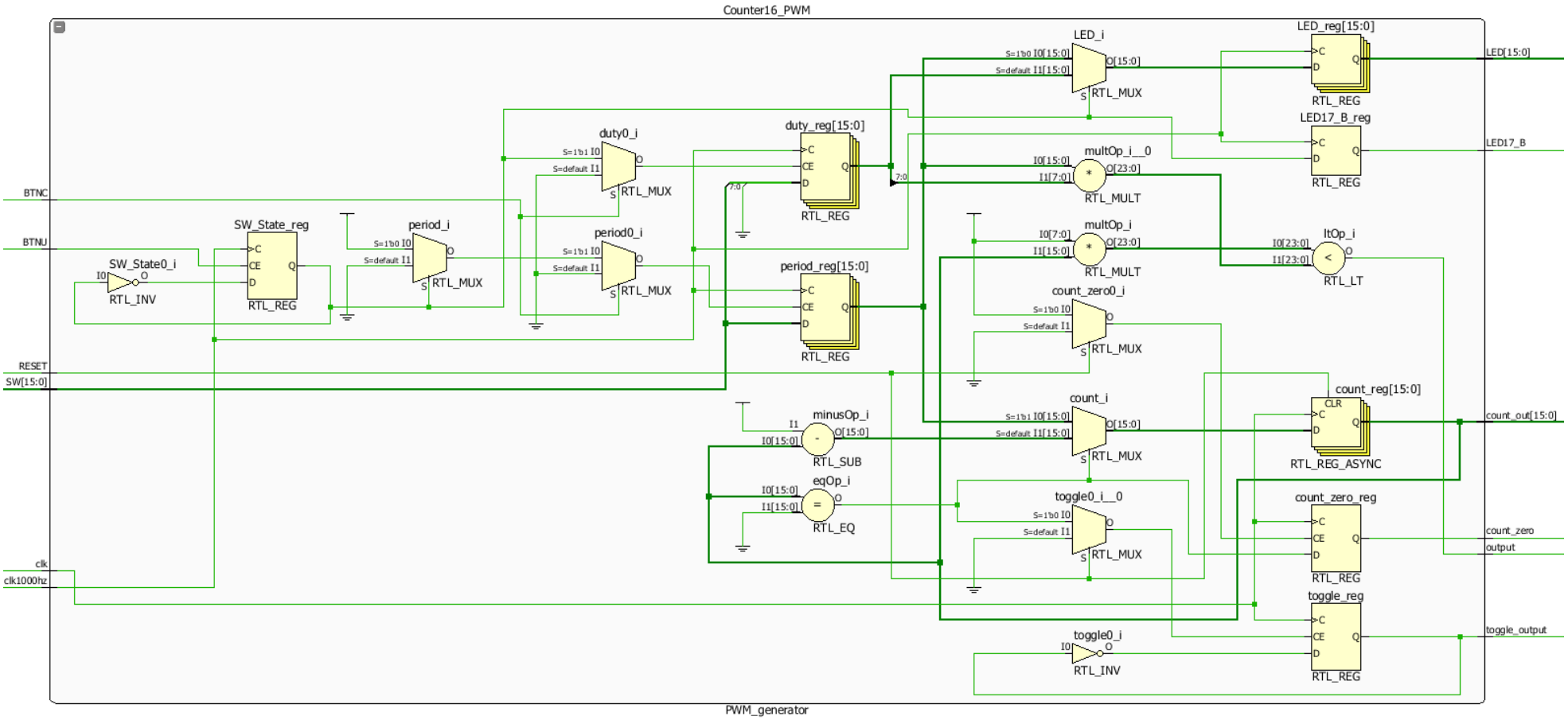

Figure 6 - PWM_generator.vhd 16 bit down counter with PWM output.

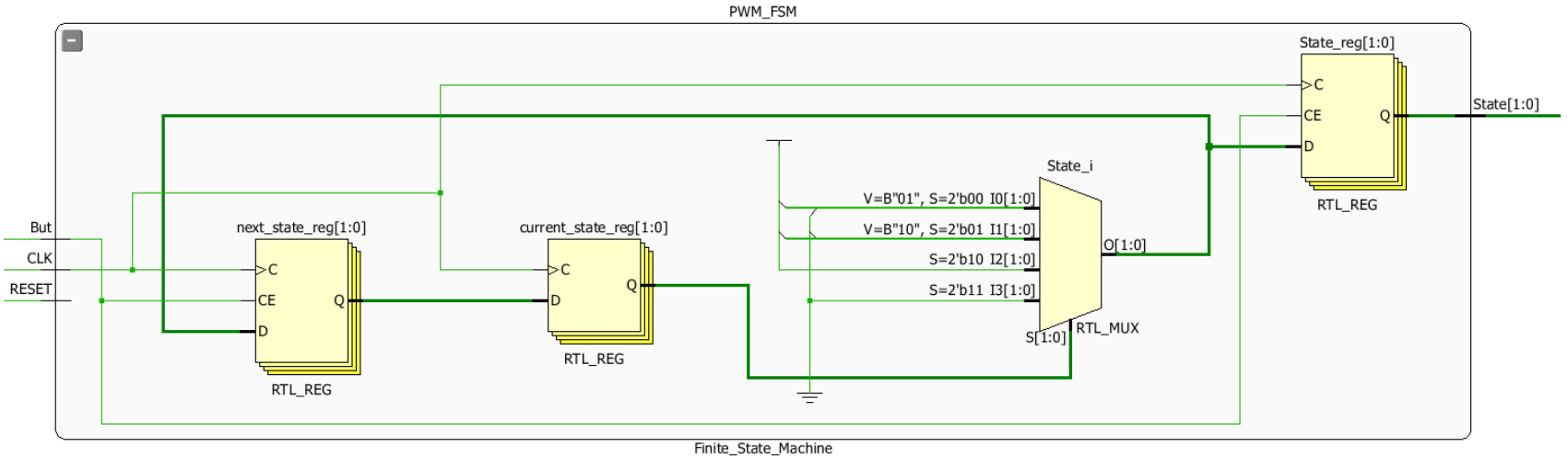

Figure 7 - Finite_State_Machine.vhd 4 state FSM.

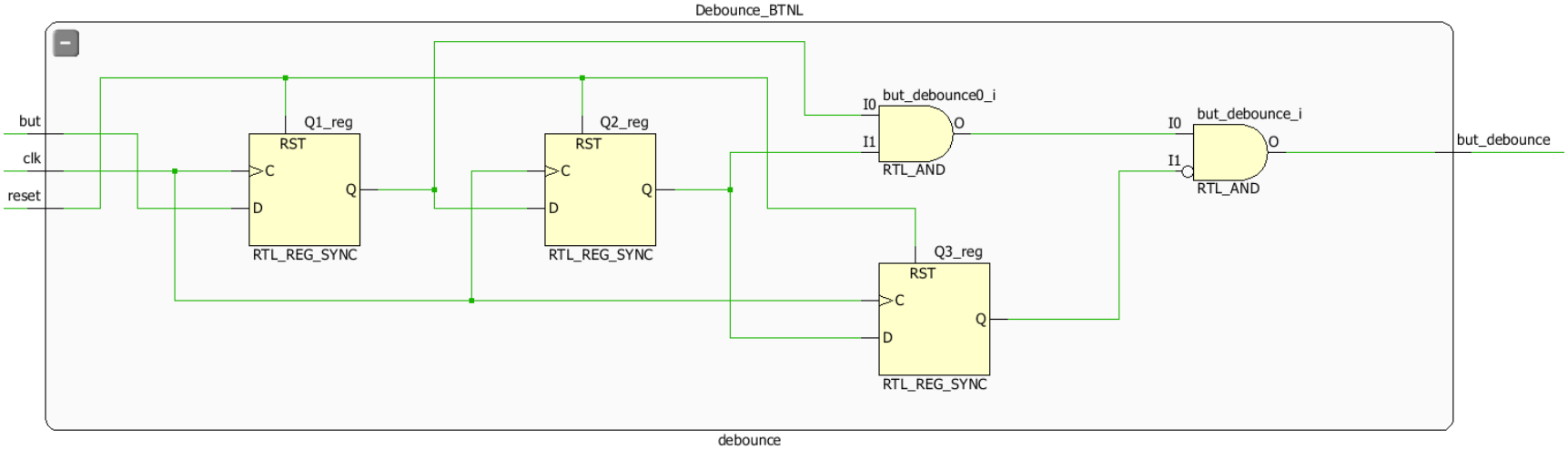

Figure 8 - debounce.vhd VHDL example button debounce code.

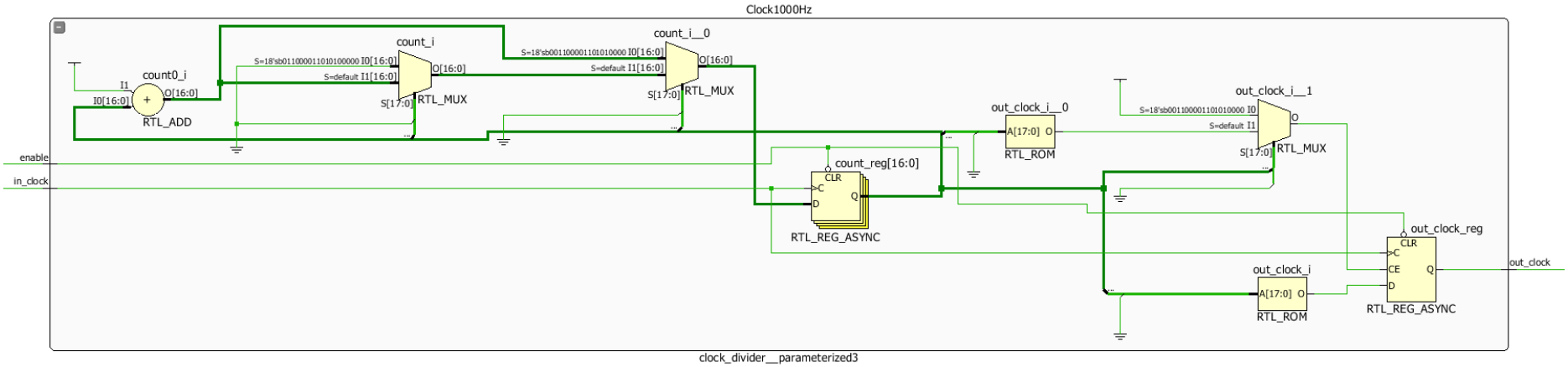

Figure 9 - clock_divider.vhd Blairs clock divider.

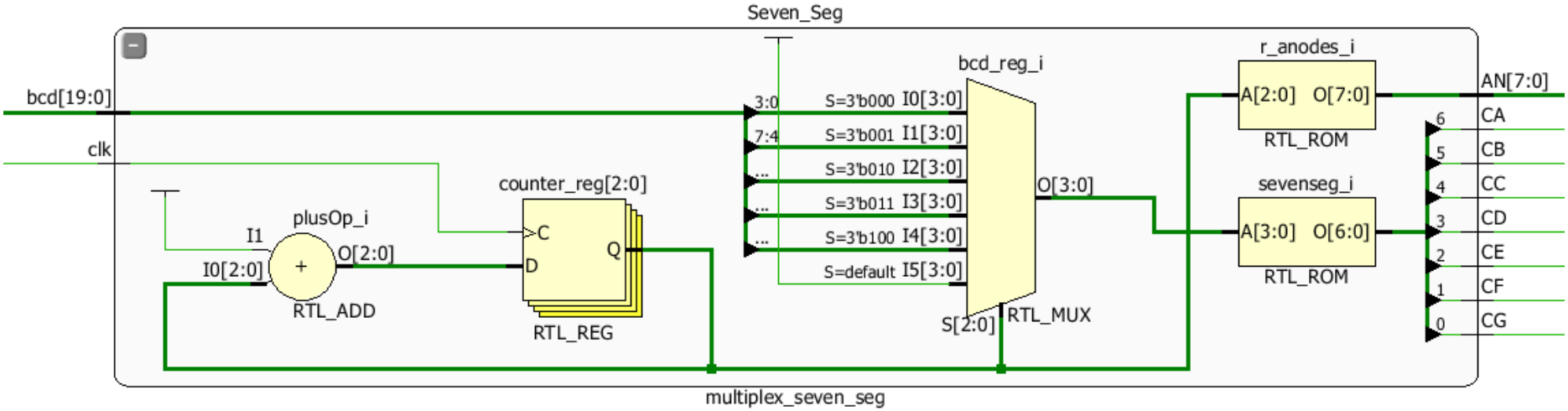

Figure 10 - multiplex_seven_seg.vhd BCD input goes to 5 seven segment displays.

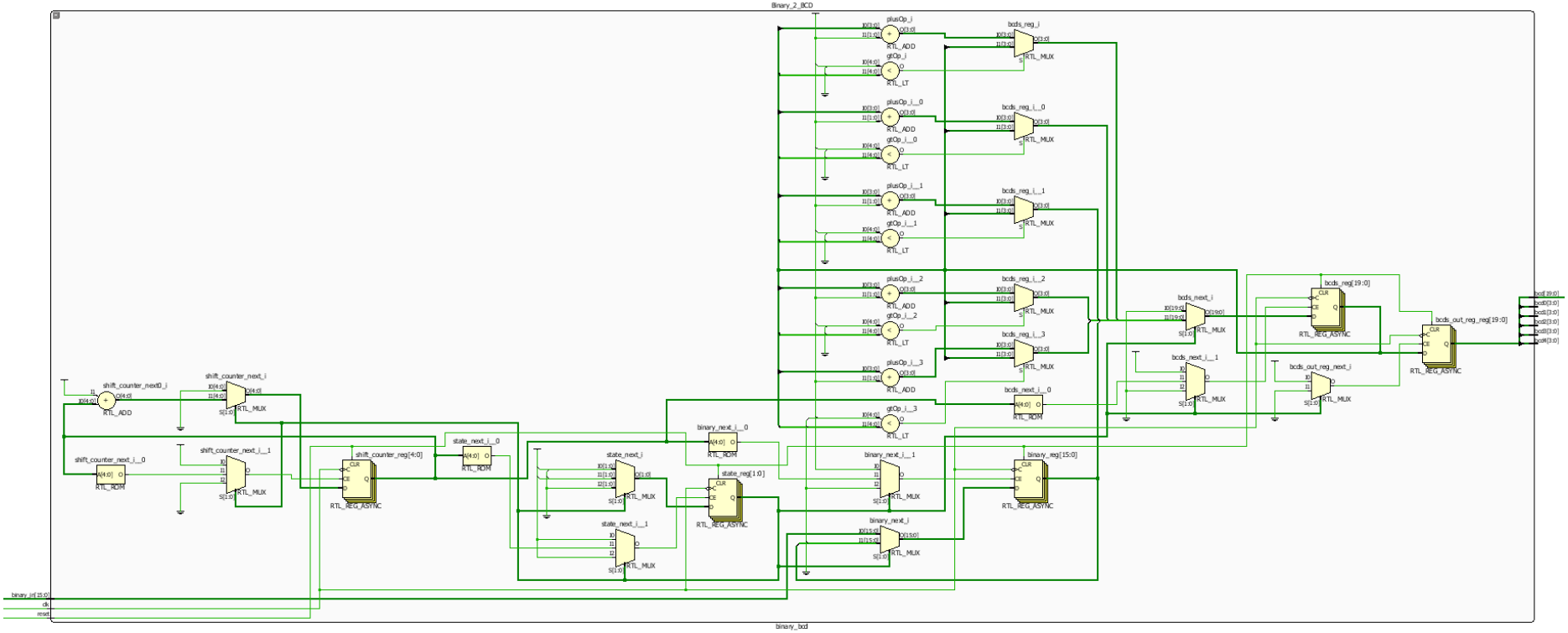

Figure 11 - binary_bcd.vhd converts 16 bit binary into 20 bit BCD.