EEL 4740 Embedded Computing Systems

Final Project

Students

Student Name
Jonathan Rene Gonzalez

Student Name
Makingston Chery

Title: Create a Tally Counter with Hardware Design Using Xilinx ISE Tools and Logi Bone with Spartan-6 xc6slx9-2tqg144 Device

Justification

Most Tally counters only count up so if a mistake is made and the button to count up is pressed when it was not meant to be pressed then the counter will have an inaccurate value. By creating a tally counter that can quickly countdown the error will be corrected and the total count will have an accurate value. Another feature that our tally counter has is that it gives you feedback when the “count up” or “count down” is pressed, we are triggering an LED for each button when it is pressed, this will create a more accurate count on whatever the user is counting.

Another use for the counter is to set it to a higher count and the user can “count down” from the total number.

Example Let's say that the user is checking how many participants are entering a room of an event, the user can press the “count up” button as people are going in. If the user presses the “count up” button by mistake it can be quickly corrected by pressing the “count down” button. Now let's assume that at this event people are asked to fill out a survey, so after all the people are inside the room the user can check how many people did not submit the survey by pushing the “count down” button as people turn in the survey. After all the people are in the room of the event the user will have an accurate number of how many people are in the room and after the event is over the user will have a count of how many people that did not turn in the survey.

Objective

To create an accurate Tally Counter that can count down and get development environment setup for Spartan-6 xc6slx9-2tqg144 device.We will apply the knowledge acquired in Embedded Computing Systems to interact with the Development board, reading inputs, setting outputs, interfacing, using peripherals, developing and programming Finite State Machine(FSM) that uses all of these.

Background

In the market place there exist tally counters that can count down but there are not to many. As a result, our research as shown this type of prototype does not inform the user when one of the button is pressed. For that reason, we decided to developed a tally counter device with an LED that will alert the user whenever one of the buttons are pressed. This will enable the user to be more accurate on their number of count.

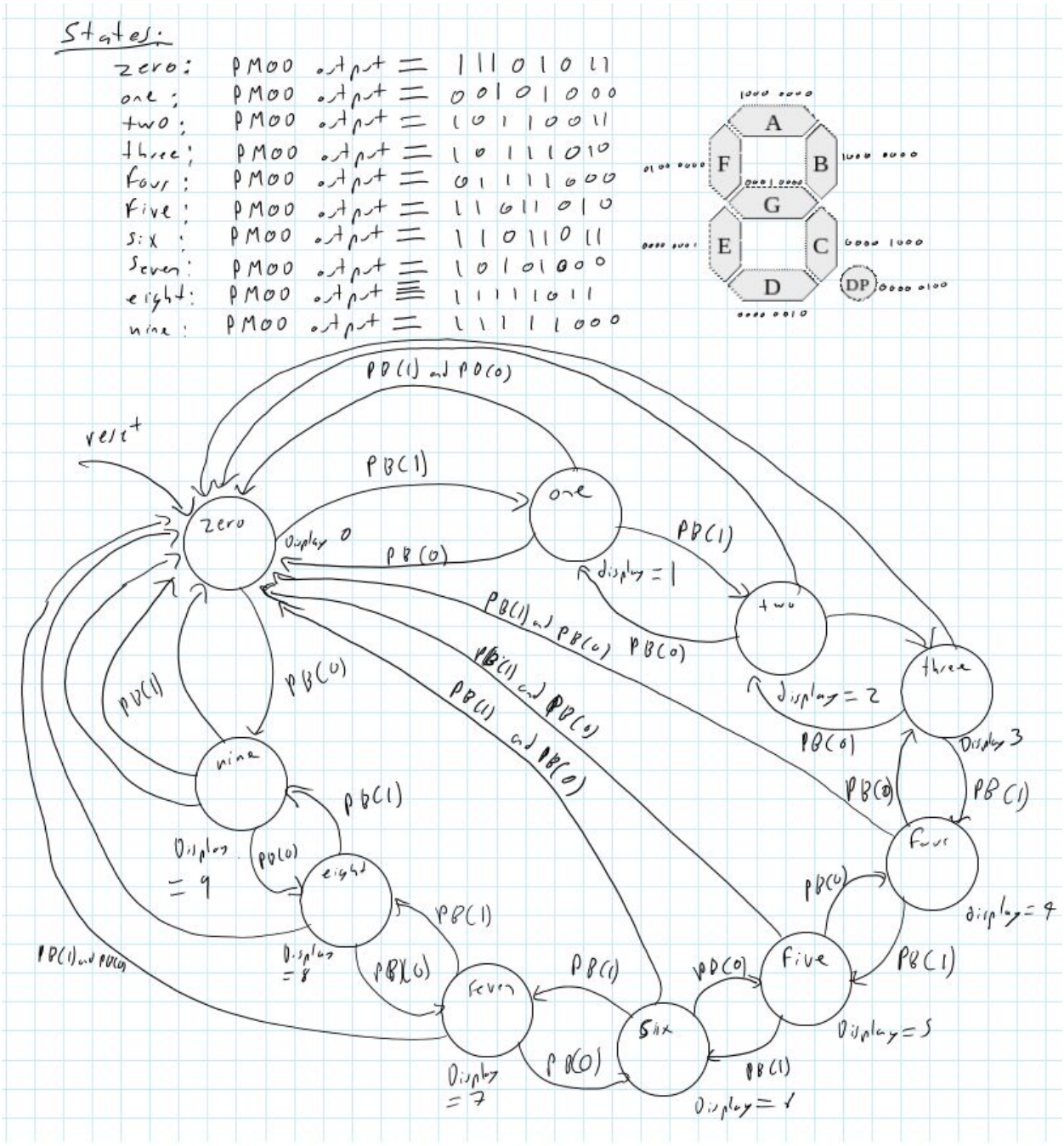
Brainstorming

The Goal is to create a Tally counter that can count up, count down, reset count, and give feedback when the count buttons are pressed. We plan to have three inputs, “count up”, “count down”, and “reset”, the tally counter will display the count on a seven segment display. Therefore an output for each segment and an output to show the feedback when the buttons are pressed. In addition, we plan to program the board by using the knowledge we acquire in class and in the lab, so we will design the Finite State Machine(FSM) with different states for each count. The count are from 0 to 10 in our prototype. We thought of two ways to programing the tally counter. First we decided to do it, by using FSM, knowing that it is not the best way to implement the counter, but we are doing it to practice and show what we have learned in class.

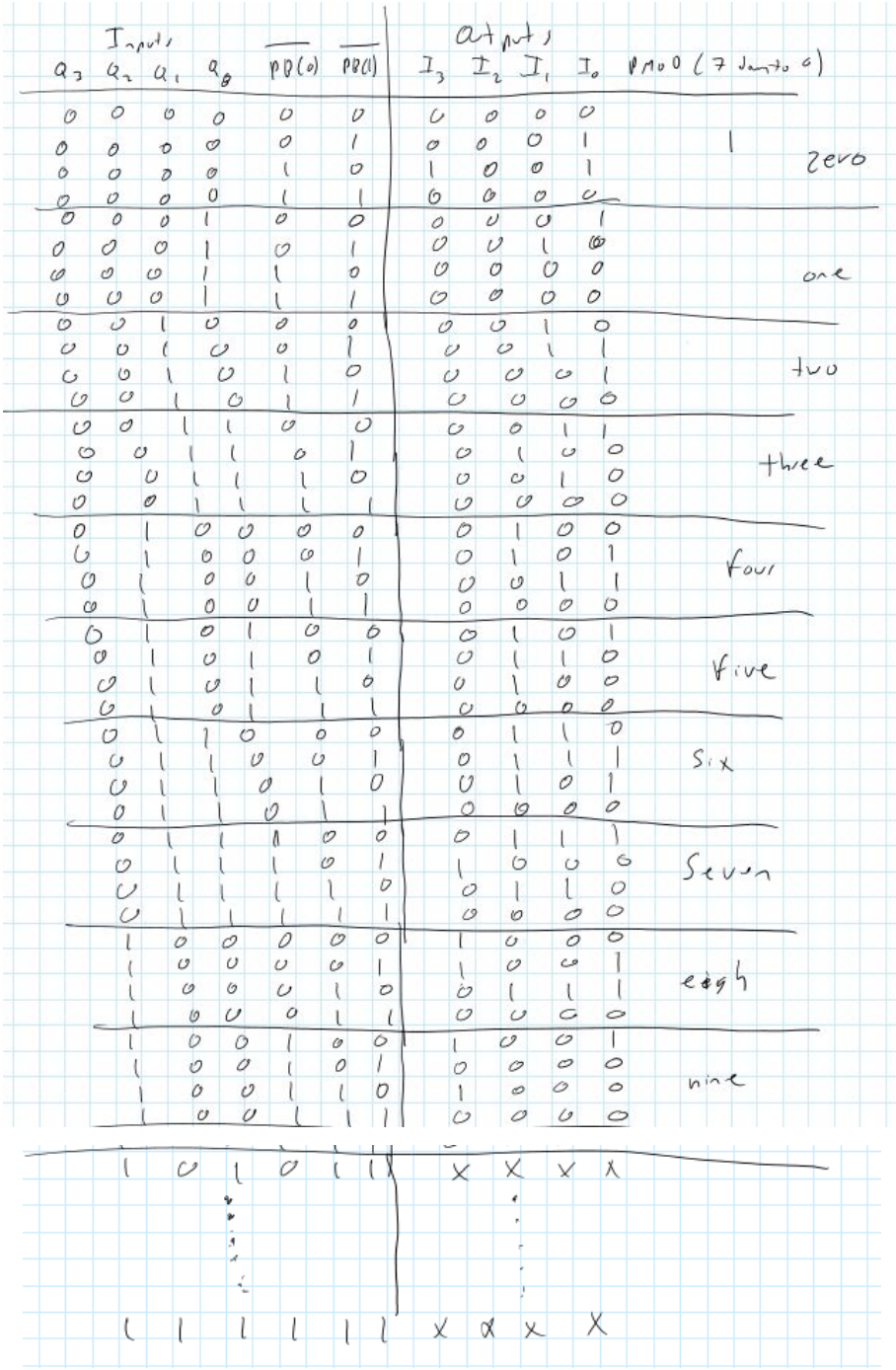
On the Power reset the FSM will set the count to ‘0’ and when the count up button is pressed it will go to the next state which will display ‘1’ . If count down button is pressed then it will go to the previous state and display the correct value. As the user presses the buttons the LED will turn on and stay on to show feedback that the button is being pressed. While the tally counter is not being pressed both LED will be blinking with different frequencies to show that the tally counter is functional.

FSM

STATES


States:
zero: PMOD output = 1110101
one: PMOD output = 0010000
two: PMOD output = 1011011
three: PMOD output = 1111010
four: PMOD output = 0111000
five: PMOD output = 1101110
six: PMOD output = 1101111
seven: PMOD output = 1010000
eight: PMOD output = 1111111
nine: PMOD output = 1111000
A
B
C
D
E
F
G
DP
reset
zero
one
two
three
four
five
six
seven
eight
nine
Display 0
Display = 1
Display = 2
Display 3
display = 4
Display = 5
Display = 6
Display = 7
Display = 8
Display = 9
PB(1)
PB(0)
PB(1) and PB(0)

Table


Inputs
Outputs
zero
one
two
three
four
five
six
Seven
eigh
nine

LOGIC using K-Map Solver

Using K-MAP solver

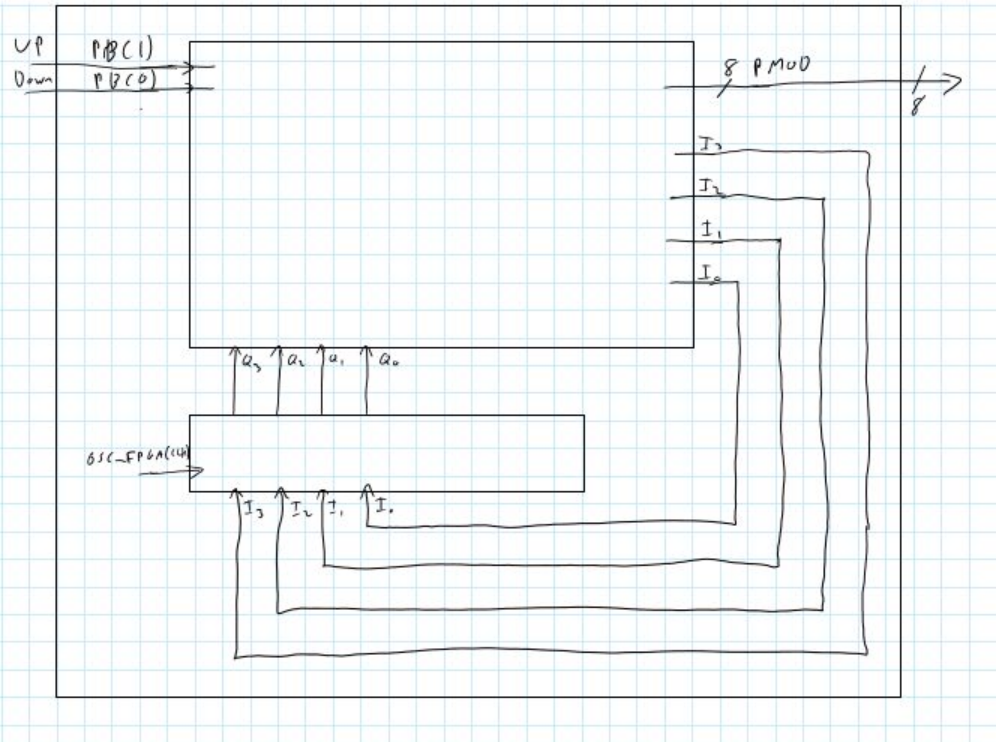
$$I_3 = Q_3\overline{Q_0}\,\overline{PB(0)} + Q_3 Q_0\,\overline{PB(1)} + Q_2 Q_1 Q_0\,\overline{PB(0)}\,PB(1) + \overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0}\,PB(0)\,\overline{PB(1)}$$

$$I_2 = Q_2\overline{Q_1}\,\overline{PB(0)} + Q_2\overline{Q_0}\,\overline{PB(0)} + Q_2 Q_0\,\overline{PB(1)} + Q_2 Q_1\,\overline{PB(1)} + \overline{Q_2}\,Q_1 Q_0\,\overline{PB(0)}\,PB(1) + Q_3\overline{Q_2}\,\overline{Q_0}\,PB(0)\,\overline{PB(1)}$$

$$I_1 = Q_1 Q_0\,\overline{PB(1)} + \overline{Q_3}\,Q_1\overline{Q_0}\,\overline{PB(0)} + Q_2\overline{Q_1}\,\overline{Q_0}\,PB(0)\,\overline{PB(1)} + Q_3\overline{Q_1}\,\overline{Q_0}\,PB(0)\,\overline{PB(1)} + \overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}\,Q_0\,\overline{PB(0)}\,PB(1)$$

$$I_0 = Q_0\,\overline{PB(0)}\,\overline{PB(1)} + \overline{Q_3}\,\overline{Q_0}\,\overline{PB(0)}\,\overline{PB(1)} + \overline{Q_3}\,\overline{Q_0}\,PB(0)\,\overline{PB(1)} + \overline{Q_2}\,\overline{Q_0}\,\overline{PB(0)}\,\overline{PB(1)} + \overline{Q_1}\,\overline{Q_0}\,PB(0)\,\overline{PB(1)}$$

Controller Design



Solution Analysis

Our solution is to develop an FSM that has a state for each count, since this is a prototype the count will only go up to 10. We have also experimented by creating the count without using FSM design to display the correct count on the 7 segment display by incrementing the count number. We decided to proceed with the design of the FSM in order to practice and show the knowledge we gained in class and in the lab.

Cost

Based on the prototype of the BeagleBone and the quantity of our technology. The unit cost is as followed:

BeagleBone…….. $135.0 per board.
The NRE (Non-Recurring Engineering cost) $135,000.0

Total cost = (NRE (Non-Recurring Engineering cost) + unit cost* #of units
$135,000 + $8 * 100,000 = 935000

$$\text{Non-Recurring Engineering cost} = \frac{\text{Total cost}}{\text{\# of units}} = \$9.35$$

The above estimate was based on the website link below.
http://www.sigenics.com/page/custom-asic-cost-calculator

Methodology

In the development and testing of our Tally Counter Prototype we will first need to get our development environment setup. To set up the development environment we used Xilinx Webpack and acquire a free license through Xilinx.

After getting our development environment setup we need to be able to use our board, Logi Bone R1.0, and we will need to be able to connect and upload our designs to the board. Our first goal is to control one LED with one Push button on the board. In order to achieve this we will need to study the Logi Bone documentation and datasheet.

Once we have our development environment setup and we are able to interact with the board we will start working with an 7 segment display peripheral because we will need it to display the count. For this reason, we will need to look into the documentation and datasheets of the 7 segment display.

After we have successfully set up the development environment and are able to communicate with the 7 segment display, we will start working on the VHDL code from our FSM Design.

Once VHDL code is ready, we will use xilinx test bench to see the outputs and set inputs. We will try all possible combinations to ensure that tally counter works. If any bugs are found or test Bench does not show the desire outputs we will adjust the code.

After we have finalize the code we will generate the bit file through xilinx and will upload the bit file to the board.

Then we will test that our prototype acts as it should.

References

Software

ISE WebPACK Design Software

https://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html

BeagleBone Black

Main Website:

https://beagleboard.org/black

Technical Reference Manual:

http://elinux.org/Beagleboard:BeagleBoneBlack#Hardware_Files
https://github.com/CircuitCo/BeagleBone-Black/blob/master/BBB_SRM.pdf?raw=true

Schematic

https://github.com/CircuitCo/BeagleBone-Black/blob/master/BBB_SCH.pdf?raw=true

LOGI-Bone R1.0

User Manual

http://valentfx.com/wiki/index.php?title=LOGI_Bone_User_Manual

Schematic

https://github.com/fpga-logi/logi-boards/blob/master/fpga-boards/logi-bone/PRJ-DOC/sch/sch-logi-bone-r1_0.PDF

Repository

https://github.com/fpga-logi

KYX-5461AS 4-digit 7-segment LED display

Schematic and pin out information used

http://thomas.bibby.ie/using-the-kyx-5461as-4-digit-7-segment-led-display-with-arduino/

Final Project prototype design and specifications

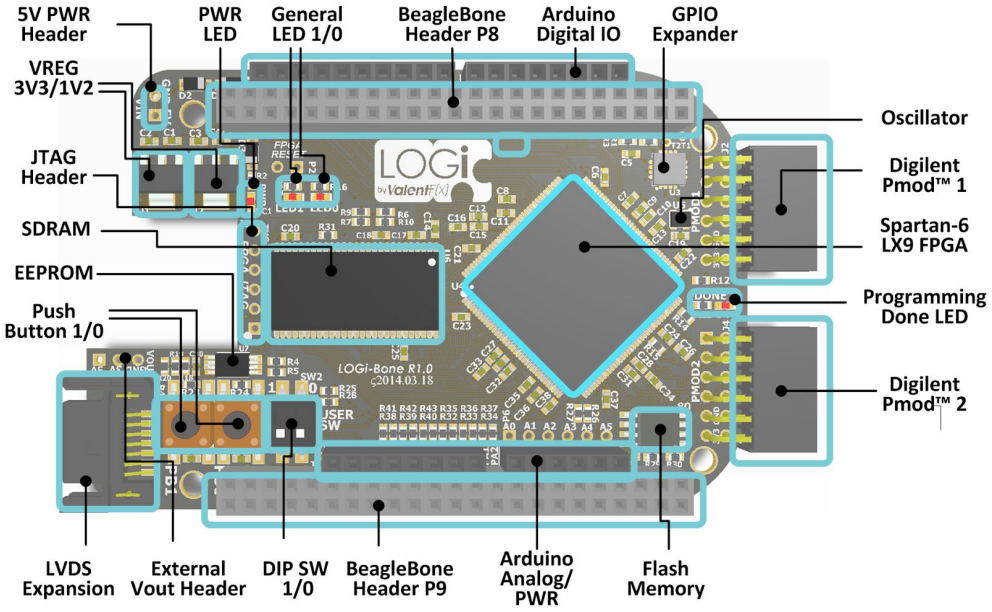
Design Specifications Needs

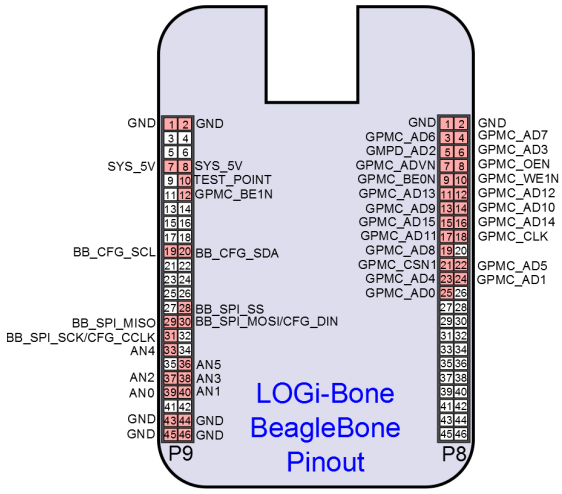
The Design specification needs for Tally Counter:

1. Tally counter needs to count up
2. Tally counter needs to count down
3. Tally counter needs to keep track of the last count
4. Tally counter needs to be able to reset the count
5. Tally counter needs to give back user feedback when an action is performed
6. Tally counter needs to let user know that it is operational (on)

Prototype Design

For the Design of the prototype we are using LOGI-Bone R0.1 FPGA board





We are also using 4 digit 7 segment display to keep track of the count

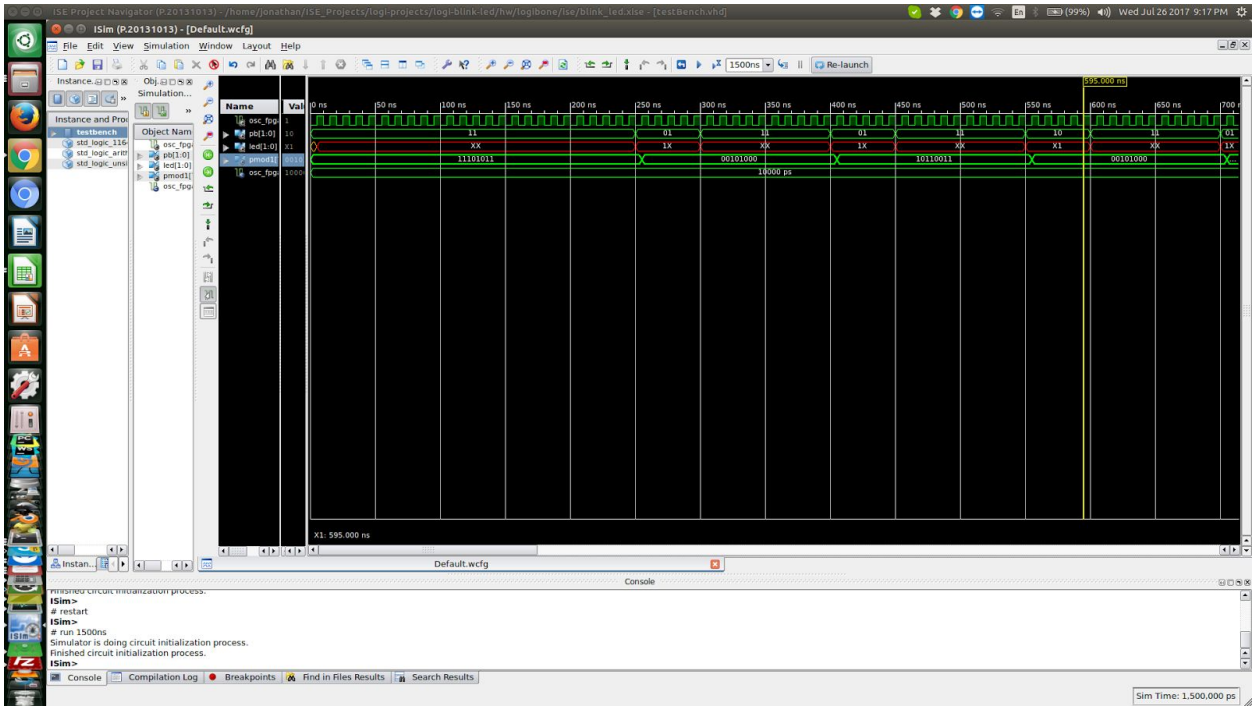
We have two push buttons on the LOGI-Bone one is used to count up the other one is used to count down. As the user counts up or down the number on the 4 bit 7 segment display will display the number of the count It also shows an LED designated to a button to let the user know that they have pressed the button. To reset the count both buttons, count up and count down have to be pressed.

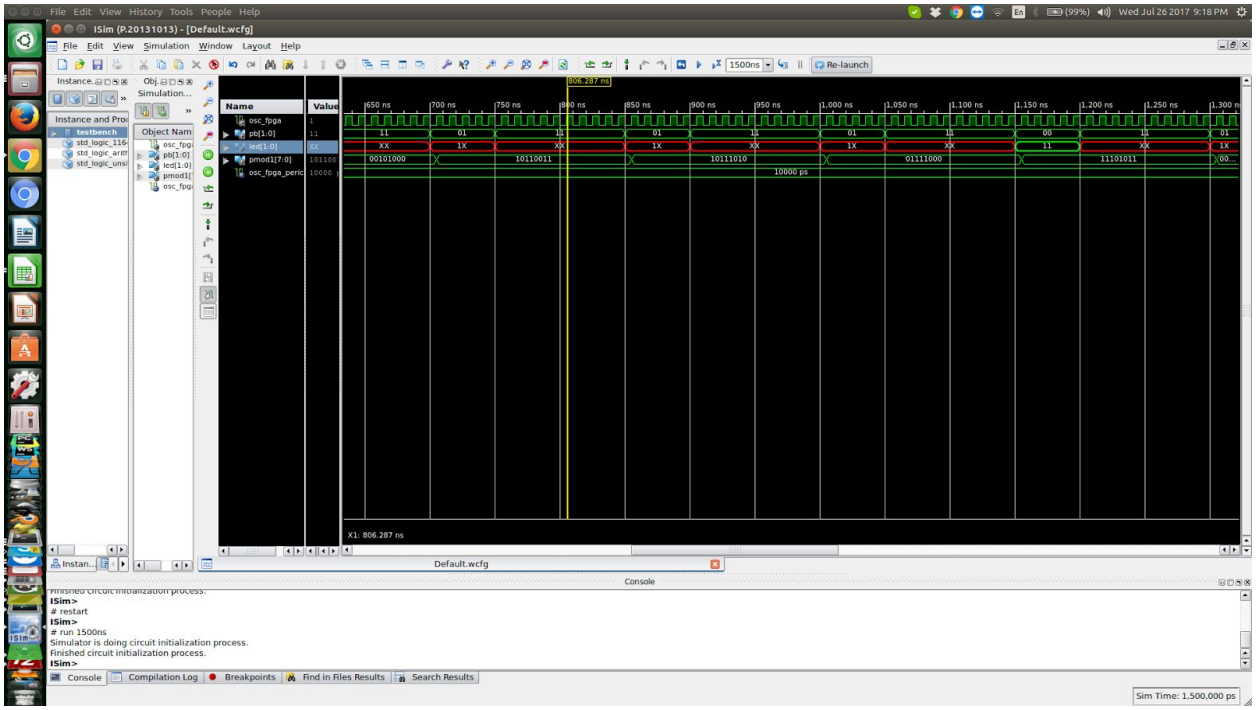
On power reset the device will be set to 0 by default.

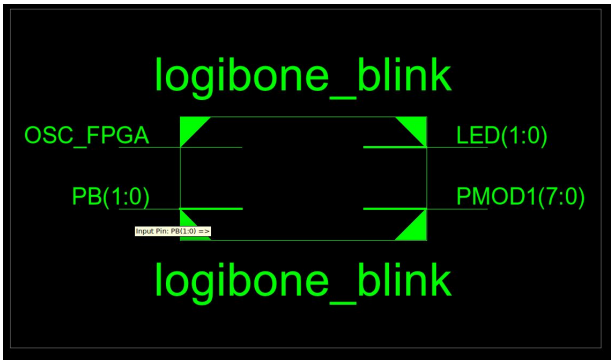
Analysis of results and test

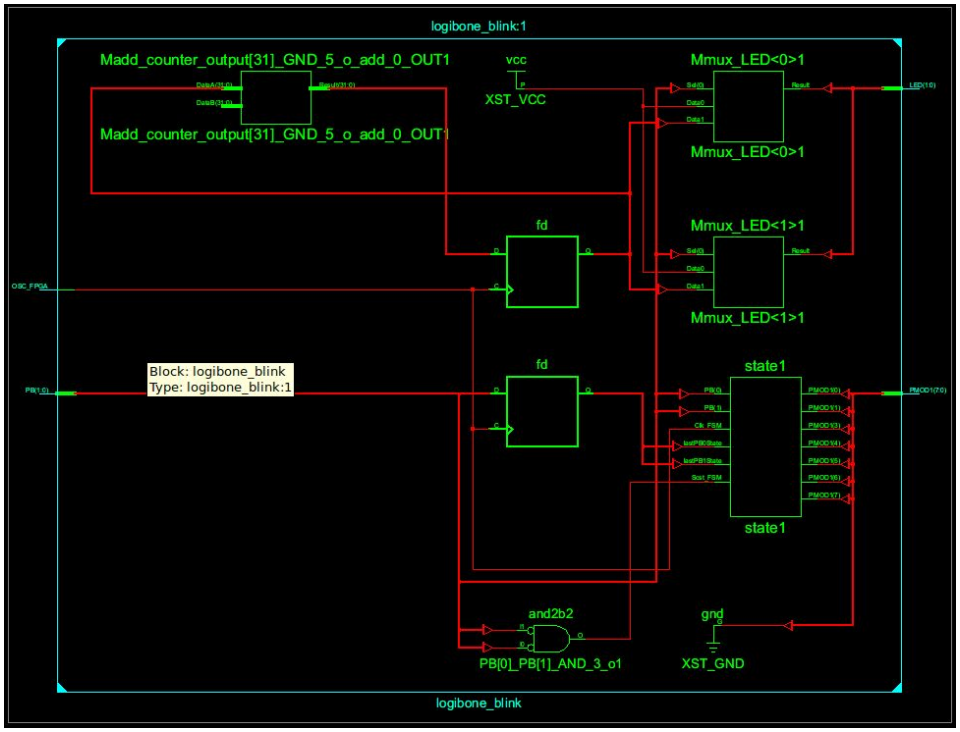
In order to test our design of the Tally counter we created a test bench giving the push buttons specific values to emulate the push of the buttons.

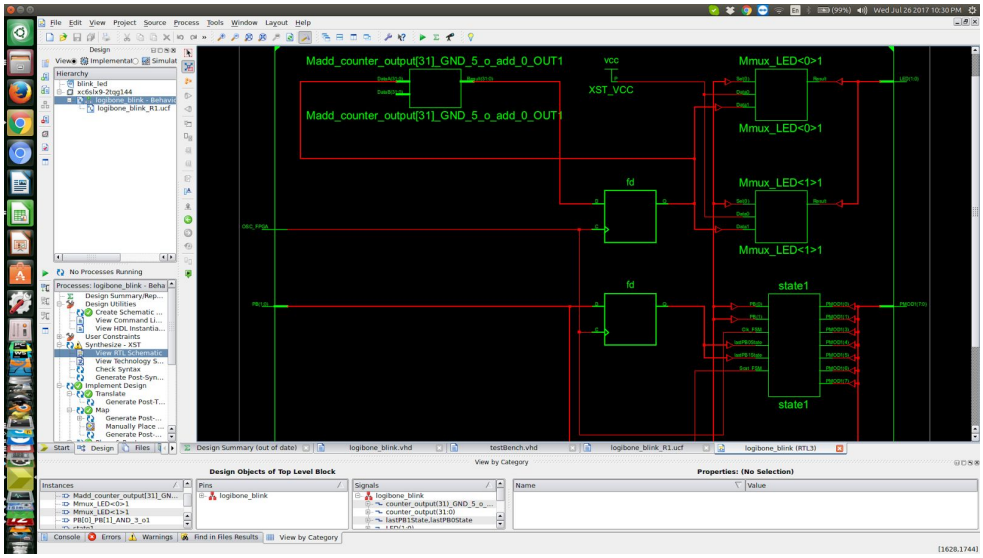
After adjusting the VHDL code to get the desire output we came up with the following results. After results were correct we uploaded the Bit file to the board.

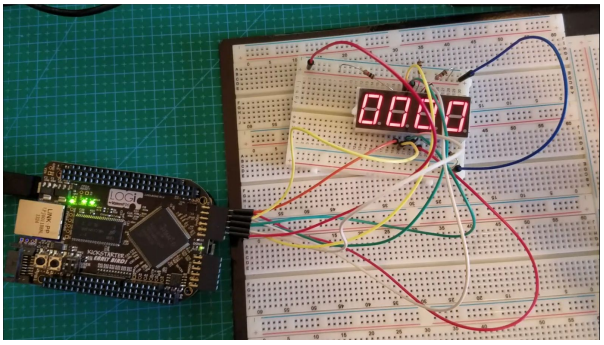


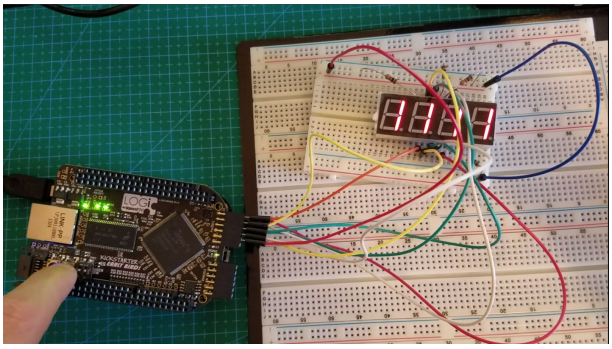


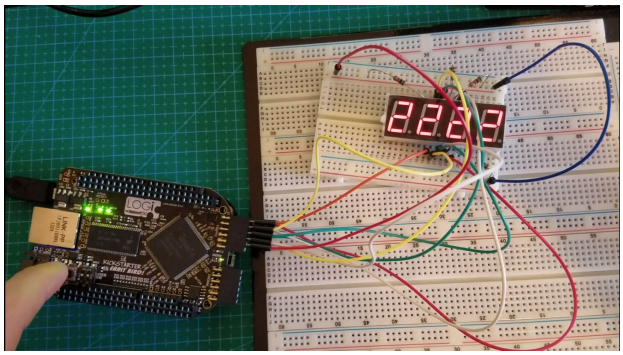

logibone_blink
OSC_FPGA
LED(1:0)
PB(1:0)
PMOD1(7:0)
logibone_blink

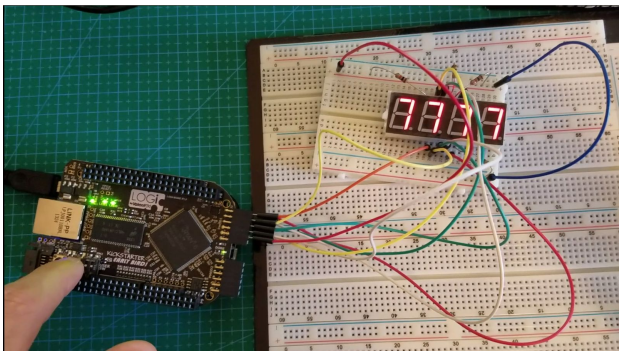

logibone_blink:1
Madd_counter_output[31]_GND_5_o_add_0_OUT1
vcc
XST_VCC
Mmux_LED<0>1
Madd_counter_output[31]_GND_5_o_add_0_OUT1
Mmux_LED<0>1
fd
Mmux_LED<1>1
OSC_FPGA
Mmux_LED<1>1
fd
state1
Block: logibone_blink
Type: logibone_blink:1
PB(1:0)
PMOD1(7:0)
state1
and2b2
gnd
PB[0]_PB[1]_AND_3_o1
XST_GND
logibone_blink

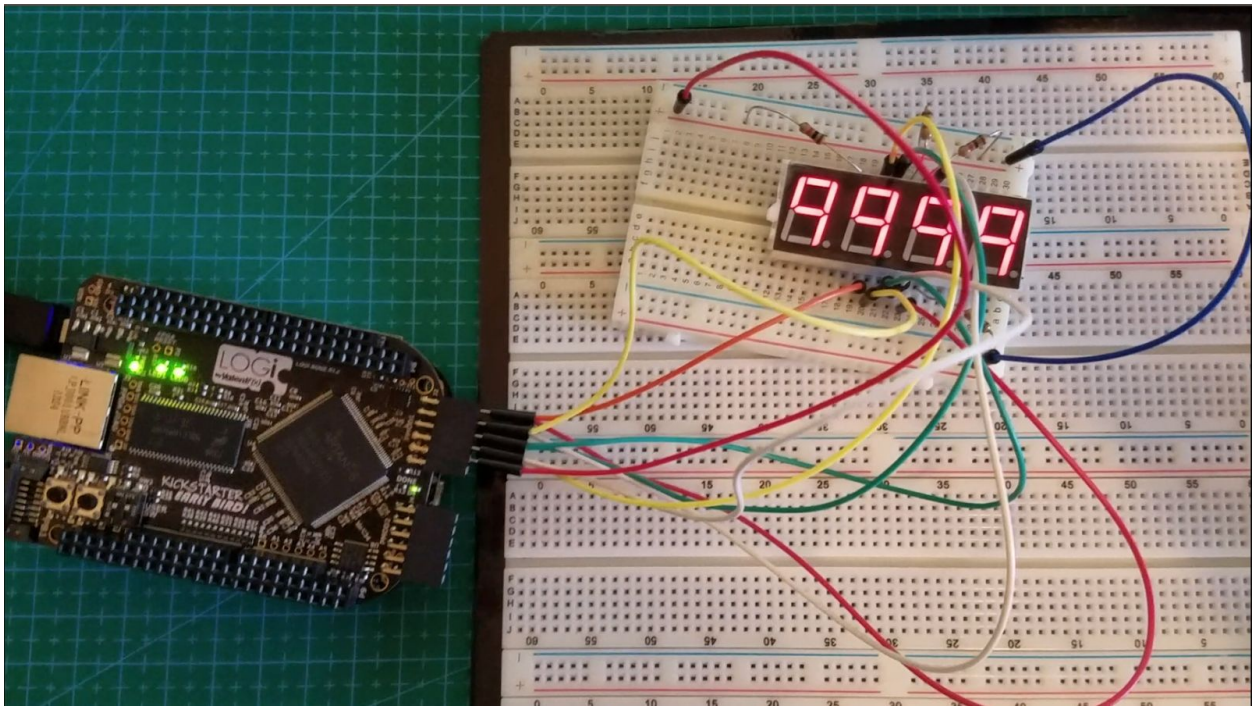

Madd_counter_output[31]_GND_5_o_add_0_OUT1
vcc
XST_VCC
Mmux_LED<0>1
Madd_counter_output[31]_GND_5_o_add_0_OUT1
Mmux_LED<0>1
fd
Mmux_LED<1>1
Mmux_LED<1>1
fd
state1
state1











Conclusions

By working on this project we were able to set up our development environment so we can now easily test and design prototypes for different projects, We learned how to communicate with the LOGI-Bone and how to assign the pins of the chip in order to use them as inputs or outputs. We were also able to interface with different internal and external peripherals by using the schematic and datasheets of the board.

Future work

The next steps of this project is to make the counter go up to the highest count on the 4 digit 7 segment display which is 9999 or change the display to an lcd display to be able to count to higher numbers. We also plan on adding a motion sensor so that the count is incremented or decremented based on which sensor is activated

Additional References

LOGI-Bone Tutorials

http://valentfx.com/wiki/index.php?title=LOGI_Guide_-_Your_First_Project_using_Xilinx_ISE

Karnaugh Map Minimizer Solver

http://www.32x8.com/var6.html

Tally Counter VHDL Code

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: Jonathan Rene Gonzalez
--
-- Create Date:    14:14:22 06/21/2012
-- Design Name:
-- Module Name:    spartcam_beaglebone - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.NUMERIC_STD.ALL;

entity logibone_blink is
port( OSC_FPGA : in std_logic;
                    PB : in std_logic_vector(1 downto 0);
                    LED : out std_logic_vector(1 downto 0);
                    PMOD1 : out std_logic_vector(7 downto 0)
);
end logibone_blink;

architecture Behavioral of logibone_blink is

        type statetype is (zero, one, two, three, four, five, six, seven, eight, nine, ten, none);
        signal state, nextstate, prevstate:statetype;


        -- Led counter
        signal counter_output : std_logic_vector(31 downto 0);

        -- signal for push buttons
        --signal push_button_0, push_button_1 : std_logic;
        -- signal for output
        --signal segment8 : std_logic_vector(7 downto 0); -- not being used not needed
```

```
	-- signal for last states of push buttons
	signal lastPB1State : STD_LOGIC := '1';
	signal lastPB0State : STD_LOGIC := '1';

begin

	process(OSC_FPGA)
	begin
		-- count up on each rising edge of the clock
		if OSC_FPGA'event and OSC_FPGA = '1' then
			-- for leds that change with counter increment counter
			counter_output <= counter_output + 1 ; -- count up on each rising edge of the
clock

			-- when push button 1 is pushed
			if(PB(1) = '0' and lastPB1State = '1') then
				state <= nextstate;
			end if;
			lastPB1State <= PB(1);
			-- when push button 0 is pushed
			if(PB(0) = '0' and lastPB0State = '1') then
				state <= prevstate;
			end if;
			lastPB0State <= PB(0);

			-- if both push buttons are pressed reet (send back to state zero)
			if(PB(0) = '0' and PB(1) = '0') then
				state <= zero;
			end if;
		end if;
	end process ;


	LED(0) <= '1' when PB(0) = '0' else counter_output(23);    --led0 will toggle every 2^(24)/50e6 =
335.5 ms = 2.98 Hz
	LED(1) <= '1' when PB(1) = '0' else counter_output(24);    --led1 will toggle every 2^(25)/50e6 =
671ms ms = 1.49 Hz

	-- next state logic
	nextstate <= one when state = zero else
					two when state = one else
					three when state = two else
					four when state = three else
					five when state = four else
					six when state = five else
					seven when state = six else
					eight when state = seven else
```

```
                                        nine when state = eight else
                                        ten when state = nine else
                                        zero when state = ten else
                                        none;

-- prevstate logic
        prevstate <= zero when state = one else
                                        one when state = two else
                                        two when state = three else
                                        three when state = four else
                                        four when state = five else
                                        five when state = six else
                                        six when state = seven else
                                        seven when state = eight else
                                        eight when state = nine else
                                        nine when state = ten else
                                        ten when state = zero else
                                        none;

        -- states logic
        PMOD1 <= "11101011" when state = zero else
                                      "00101000" when state = one else
                                      "10110011" when state = two else
                                      "10111010" when state = three else
                                      "01111000" when state = four else
                                      "11011010" when state = five else
                                      "11011011" when state = six else
                                      "10101000" when state = seven else
                                      "11111011" when state = eight else
                                      "11111000" when state = nine else
                                      "11111001" when state = ten else -- displaying A
                                      "00000100";



        -- to test just set all pmods to 1
        --PMOD1(0) <= '1' when segment8(0) = '1' else '0';
        --PMOD1(1) <= '1' when segment8(1) = '1' else '0';
        --PMOD1(2) <= '1' when segment8(2) = '1' else '0';
        --PMOD1(3) <= '1' when segment8(3) = '1' else '0';
        --PMOD1(4) <= '1' when segment8(4) = '1' else '0';
        --PMOD1(5) <= '1' when segment8(5) = '1' else '0';
        --PMOD1(6) <= '1' when segment8(6) = '1' else '0';
        --PMOD1(7) <= '1' when segment8(7) = '1' else '0';
```

end Behavioral;

Tally Counter User Constraint File (UCF)

```
##############################################################
##############################################################
## UCF Constraints file                    ##
##############################################################
##############################################################

######################
# Timing Constraints #
######################

##### Grouping Constraints #####
NET OSC_FPGA TNM_NET = clk50_grp;

##### Clock Period Constraints #####
TIMESPEC TS_PER_CLK50 = PERIOD "clk50_grp" 20.0 ns;

######################
# Pin LOC Constraints #
######################

#PMOD1#########################################################################
##########
NET PMOD1<0>          LOC = "P112" ;
NET PMOD1<1>          LOC = "P111" ;
NET PMOD1<2>          LOC = "P67" ;
NET PMOD1<3>          LOC = "P66" ;
NET PMOD1<4>          LOC = "P62" ;
NET PMOD1<5>          LOC = "P61" ;
NET PMOD1<6>          LOC = "P58" ;
NET PMOD1<7>          LOC = "P57" ;

#PUSH
BUTTONS#######################################################################
###
NET PB<0>          LOC = "P83" ; #
NET PB<1>          LOC = "P59" ; #

# LED 0 and LED 1 ###################
NET LED<0>        LOC = "P140"  ;
NET LED<1>        LOC = "P74"  ;
NET OSC_FPGA      LOC = "P85"  ;

#Dip
Switches######################################################################
##
```

```
#NET SW<0>          LOC = "P75" ;   # No used for project
#NET SW<1>          LOC = "P78" ;   # not used for project
```

Tally Counter Test Bench

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   12:45:17 07/26/2017
-- Design Name:
-- Module Name:   /home/jonathan/ISE_Projects/logi-projects/logi-blink-led/hw/logibone/ise/testBench.vhd
-- Project Name:  blink_led
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: logibone_blink
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY testBench IS
END testBench;

ARCHITECTURE behavior OF testBench IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT logibone_blink
    PORT(
        OSC_FPGA : IN  std_logic;
```

```
        PB : IN  std_logic_vector(1 downto 0);
        LED : OUT  std_logic_vector(1 downto 0);
        PMOD1 : OUT  std_logic_vector(7 downto 0)
       );
    END COMPONENT;


   --Inputs
   signal OSC_FPGA : std_logic := '0';
   signal PB : std_logic_vector(1 downto 0) := (others => '0');

          --Outputs
   signal LED : std_logic_vector(1 downto 0);
   signal PMOD1 : std_logic_vector(7 downto 0);
   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name

   constant OSC_FPGA_period : time := 10 ns;

BEGIN

          -- Instantiate the Unit Under Test (UUT)
   uut: logibone_blink PORT MAP (
        OSC_FPGA => OSC_FPGA,
        PB => PB,
        LED => LED,
        PMOD1 => PMOD1
       );

   -- Clock process definitions
   OSC_FPGA_process :process
   begin
                    OSC_FPGA <= '0';
                    wait for OSC_FPGA_period/2;
                    OSC_FPGA <= '1';
                    wait for OSC_FPGA_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin

                    PB(1) <= '1';
                    PB(0) <= '1';

                    LED(0) <= '0';
                    LED(1) <= '0';
```

```
-- hold reset state for 100 ns.
wait for 100 ns;

		LED(0) <= '1';
		LED(1) <= '1';

wait for OSC_FPGA_period*10;

-- insert stimulus here
		-- next state
		PB(1) <= '1';
		wait for 50 ns;
		PB(1) <= '0';
		wait for 50 ns;
		PB(1) <= '1';
		wait for 50 ns;

		-- nextState
		PB(1) <= '1';
		wait for 50 ns;
		PB(1) <= '0';
		wait for 50 ns;
		PB(1) <= '1';
		wait for 50 ns;

		-- go back to prev state
		PB(0) <= '1';
		wait for 50 ns;
		PB(0) <= '0';
		wait for 50 ns;
		PB(0) <= '1';
		wait for 50 ns;

		-- increase
		PB(1) <= '1';
		wait for 50 ns;
		PB(1) <= '0';
		wait for 50 ns;
		PB(1) <= '1';
		wait for 50 ns;

		PB(1) <= '1';
		wait for 50 ns;
		PB(1) <= '0';
		wait for 50 ns;
		PB(1) <= '1';
		wait for 50 ns;
```

```
PB(1) <= '1';
wait for 50 ns;
PB(1) <= '0';
wait for 50 ns;
PB(1) <= '1';
wait for 50 ns;

--reset
PB(0) <= '1';
PB(1) <= '1';
wait for 50 ns;
PB(0) <= '0';
PB(1) <= '0';
wait for 50 ns;
PB(0) <= '1';
PB(1) <= '1';
wait for 50 ns;

PB(1) <= '1';
wait for 50 ns;
PB(1) <= '0';
wait for 50 ns;
PB(1) <= '1';
wait for 50 ns;

PB(1) <= '1';
wait for 50 ns;
PB(1) <= '0';
wait for 50 ns;
PB(1) <= '1';
wait for 50 ns;

PB(1) <= '1';
wait for 50 ns;
PB(1) <= '0';
wait for 50 ns;
PB(1) <= '1';
wait for 50 ns;

PB(1) <= '1';
wait for 50 ns;
PB(1) <= '0';
wait for 50 ns;
PB(1) <= '1';
wait for 50 ns;

PB(1) <= '1';
wait for 50 ns;
```

```
                PB(1) <= '0';
                wait for 50 ns;
                PB(1) <= '1';
                wait for 50 ns;

                PB(1) <= '1';
                wait for 50 ns;
                PB(1) <= '0';
                wait for 50 ns;
                PB(1) <= '1';
                wait for 50 ns;

                PB(1) <= '1';
                wait for 50 ns;
                PB(1) <= '0';
                wait for 50 ns;
                PB(1) <= '1';
                wait for 50 ns;

                PB(1) <= '1';
                wait for 50 ns;
                PB(1) <= '0';
                wait for 50 ns;
                PB(1) <= '1';
                wait for 50 ns;

                PB(1) <= '1';
                wait for 50 ns;
                PB(1) <= '0';
                wait for 50 ns;
                PB(1) <= '1';
                wait for 50 ns;

                PB(1) <= '1';
                wait for 50 ns;
                PB(1) <= '0';
                wait for 50 ns;
                PB(1) <= '1';
                wait for 50 ns;


      wait;
   end process;

END;
```