|  |  |
| --- | --- |
| *Title:* | ***Lab #8: FPGA PONG – Score Keeper*** |
| *Name:* |  |

# INTRODUCTION

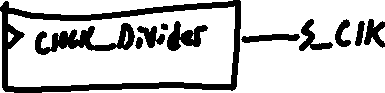
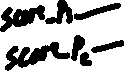
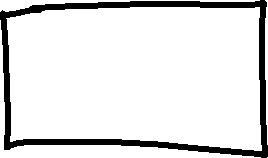
You need to use Xilinx Vivado.

In this series of labs covering VHDL knowledge, we will work step by step to design and play FPGA pong. In this lab, we will design a score keeper module that will display a score on the seven segment display.

Start by reading Section 8.1 of the BASYS Reference Manual to gain an understanding of how the seven segment display works.

Also look at the lecture notes on how 7-seg displays work.

Here is a quick overview of what we are trying to accomplish:



# Start Xilinx Vivado

1. Create a new project for this lab.
2. Create a new design file called score\_keep.vhd. Define inputs as shown in the above image.
3. Also create a file called clock\_divider.vhd with the inputs shown in the above image
4. Import the VHDL file score\_keep\_test.vhd into your design sources when creating the project, and set it as the top.
5. Import the constraint file for the Basys 3 board.
6. You will need to uncomment the following portions of the constraint file.
   * Uncomment the two ‘clk’ lines.
   * Uncomment all lines in the ‘7 segment display’ section, except for the one by itself (port dp)
     1. Should be seg[6:0] and an[3:0]
   * Uncomment the buttons btnL and btnR

**Clock Divider**

The onboard clock on the FPGA runs at 100MHz, while the seven segment display runs significantly slower, around 1.5kHz. To make up for this, we must divide the clock on the FPGA. While we do have a module called clock\_divider, we will only use this as a divide by 4 module, while the score keeper itself will divide the rest of the way from the slow clock to the 1.5khz clock. This is because the VGA controller we will be designing later runs at around 25MHz, and so we want to have a main clock running the entire board at 25MHz.

In the clock\_divider module, we are implementing a divide by 4 circuit. Create a 2 bit counter, and in a PROCESS have it increment by 1 for each clock of the FPGA board. When the signal reaches 3, reset the counter (0, 1, 2, 3, 0…). Outside of the PROCESS block, assign s\_clk such that it has a 50% duty cycle (on for the first half, off for the second).

**Score Keep**

A general overview of the workings of score\_keep are as follows:

1. A PROCESS block is used as a counter, dividing the incoming 25MHZ clock by . The 2 MSBs will be used to determine the digit\_sel signal.
2. A PROCESS block is used to assign ‘an’ depending on digit\_sel.
3. 2 PROCESS block are used to ‘see’ the rising edge of the score signals, and assign it to a register.
4. A PROCESS block is used to detect a reset, and reset scores if so. Additionally, if no reset is detected, and if there is a rising edge of a score signal, it will increment that player’s score by 1.
5. A PROCESS block is used to determine what digit\_val is depending on which digit\_sel is active, and whatever the associated player’s score is.
6. A final PROCESS block is used to assign ‘seg’ depending on what digit\_val is.

Start by creating signals:

signal counter : unsigned (15 downto 0) := (others => '0');

signal score\_p1\_val : integer := 0;

signal score\_p2\_val : integer:= 0;

signal digit\_sel : unsigned (1 downto 0);

signal score\_p1\_prev, score\_p2\_prev : std\_logic := '0';

signal score\_p1\_rise, score\_p2\_rise : std\_logic := '0';

signal digit\_val : integer range 0 to 9 := 0;

* Create your first process block, triggered on the clock. This is almost the exact same as our clock divider block, except instead of resetting ‘counter’ at 3, it will reset at . Outside of this process block, assign ‘digit\_sel’ to the two MSBs of the ‘counter’ signal. This digit\_sel will increment from 00 to 11 corresponding to the 4 digits of the 7 segment display.
* Create a second process block triggered by digit\_sel. ‘an’ is an active low one-hot signal select. This means that setting an to ‘1111’ means “no digits active”, and inversely ‘0000’ means “all digits active”. Using a case statement, set an to the proper signals using a case statement.

|  |  |
| --- | --- |
| **digit\_sel** | **an** |
| 00 | 1110 |
| 01 | 1101 |
| 10 | 1011 |
| 11 | 0111 |

* Your third process block is also triggered by the (fast) clock, and follows some odd logic. VHDL cannot by default detect the rising edge of any signal besides clock signals. To work around this, each clock cycle we will set score\_p1\_prev <= score\_p1; In the fourth process block, also triggered by the clock, we will set score\_p1\_rise <= '1' when score\_p1 = '1' and score\_p1\_prev = '0' else '0'; score\_p2 can be calculated in the same way in the same process blocks.
* In the fifth process block, clock triggered, first start by checking for a high reset. If it is high, set the score values to 0. Otherwise, if the previously calculated score rise is 1, then we can increment the score val by 1 for the corresponding player:

Clock triggers

Reset?

Scores = 0

Else:

If score rise

Score\_val + 1

* In the sixth process block, triggered by the digit\_sel, we will use a case statement to determine what digit should actually be displayed. In the case, sensitive to digit\_sel, assign digit\_val as follows:

|  |  |
| --- | --- |
| **digit\_sel** | **digit\_val** |
| 00 | score\_p2\_val mod 10 |
| 01 | (score\_p2\_val / 10) mod 10 |
| 10 | score\_p1\_val mod 10 |
| 11 | (score\_p1\_val / 10) mod 10 |

* In the final process block, triggered by a change in digit\_val, we will use a case statement to assign seg to be corresponding to the digit val. Look back at the lecture notes to understand again how the 7 segment display shows its values, and assign the case statement as follows (some of this is done for you, since it is very repetitive)

case digit\_val is

when 0 => seg <= "1000000";

when 1 => seg <= "1111001";

when 2 => seg <= " ";

when 3 => seg <= "0110000";

when 4 => seg <= "0011001";

when 5 => seg <= "0010010";

when 6 => seg <= " ";

when 7 => seg <= "1111000";

when 8 => seg <= " ";

when 9 => seg <= "0010000";

when others => seg <= " ";--you choose here!

end case;

# Compile and Test it!

You should now be able to generate a bitstream and program your FPGA. If your code works, you will be able to increment the score on the 7 segment display by pressing the left and right buttons.

Show your TA and have him/her sign the checkoff sheet.

**Troubleshooting**

If you are having trouble, run through the following troubleshooting steps:

* Does your seven segment display have a “ghost” number?
  1. Your clock signal is too fast, Look back through and ensure that you are dividing by 4 in the clock\_divider, then by 65535 in the score\_keep module
* Does pressing the buttons increment the wrong number?
  1. In your 6th process block, you are likely running the wrong calculations to determine what digit you are showing. Make sure that you are displaying tens, ones, tens, ones.
  2. If it is not the calculations, check your rising edge code. Make sure that the player one score triggers player 1 val, etc.
* Do you have no display on your seven segment?
  1. Remember the digit select should show something for “others”. Make sure that there is at least something shown here, so you can tell if it is failing.
  2. If you still have no output, it is possible that you have set an or seg incorrectly. Check carefully through your code, make sure that only one an is 0 at a time, and make sure that your seg is active low.
  3. If you still have no output, check your input and output names, especially in reference to the test module. If they do not match exactly, it will not see your module.
* Does pressing the button increment the numbers continuously?
  1. You are incorrectly detecting the rising edge of the score signals. Check back carefully and ensure that it only actually increments for a single clock cycle. (make sure your code has “score\_px *and not* score\_px\_prev”)