**Team Multiplexers Project 4 Lab Report**

**Proposal: Hangman**

Alexander Graening (804732740), Xilai Zhang (804796478), Joseph Miller (504744848)

**Overview:**

We will create a version of the game “hangman” on an FPGA. We will use keyboard input and VGA output alongside a button for resetting the game.

**Game Description:**

Upon starting the game, user will be prompted to enter a word with up to 10 letters which will be displayed on the screen. If the user does not use all 10 letters, the unused letters will be treated as empty and not counted in the rest of the game. When starting the game, one user will enter letters for the word which will be displayed on the screen. After hitting the correct key to submit the word, the letters will be hidden as a placeholder character. A second user will then have a chance to guess the letters one at a time according to traditional hangman rules.

**Grading Rubric:**

VGA output (15%) - Game is capable of being rendered to the screen through the VGA output on the FPGA.

Graphics (15%) - Features all 26 letters of the english alphabet alongside graphics for the gallows, and hanged man as well as a few special characters to indicate unfilled spaces and unused spaces.

New Word Functionality (15%) - Players are capable of entering a new word up to 10 characters long that will be used for the duration of the current game.

Input from keyboard (20%) - Keyboard input is delivered utilizing the USB-HID interface to receive and interpret PS2 scancodes.

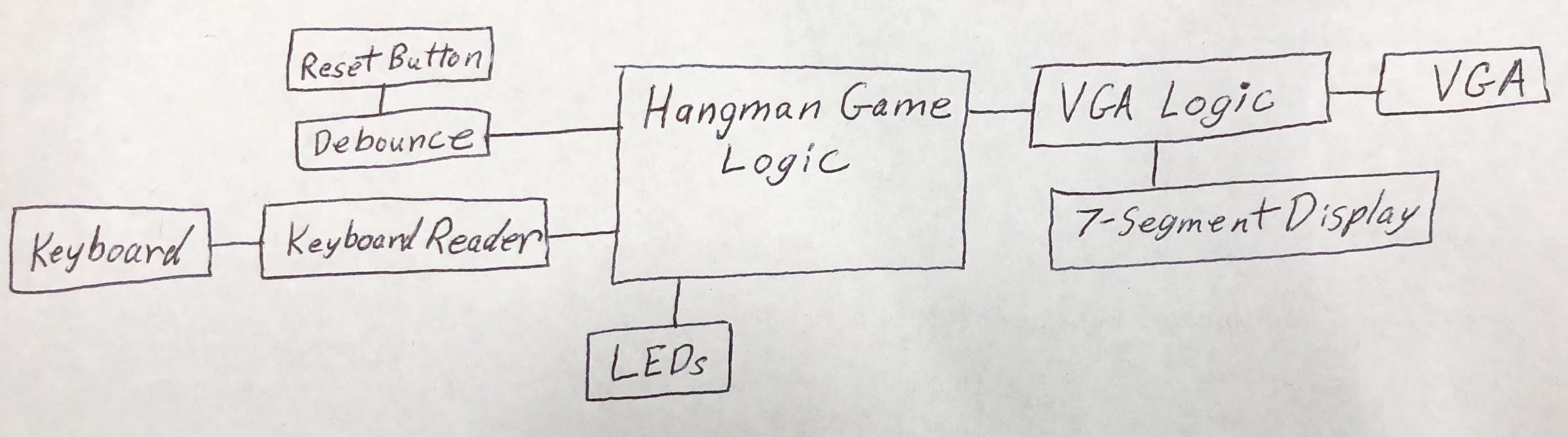
Winning/Losing Display (10%) - Upon running out of guesses the player is informed of having lost. Upon correctly finishing the word, the player is told they have won. Each are accompanied by different graphics.

Word Setting (10%) - Upon start of the game, input from the keyboard is displayed on the current letter being modified. Upon receipt of an enter that letter is set and the next letter can be modified. Upon receipt of a ‘\’ or when 10 letters have been entered, the word is set.

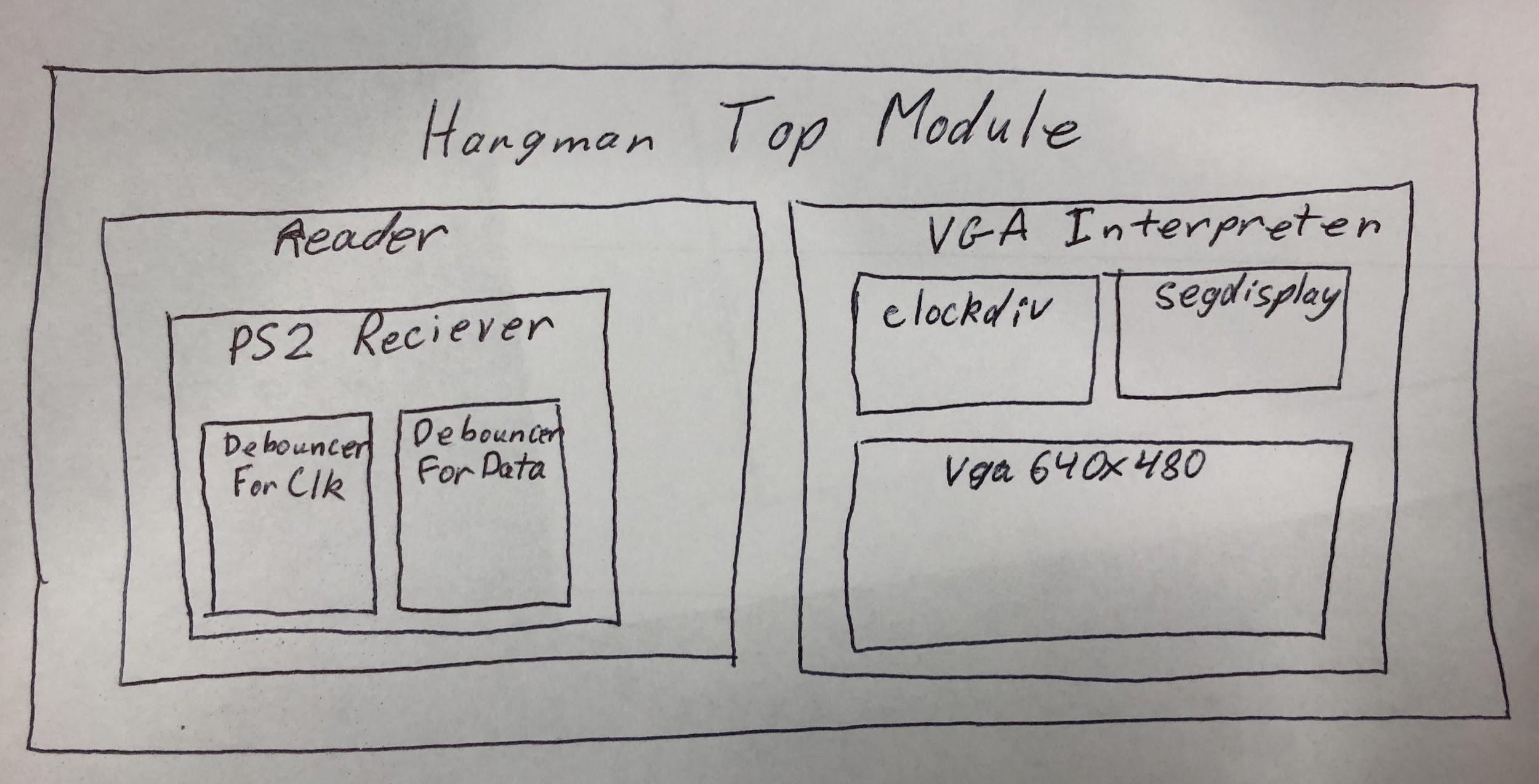
Word Guessing (10%) - When entering a new key on the keyboard, all letters associated to that keypress are revealed on the bottom. If no new letters get revealed, a new segment of the hanged man is added.

New Game/Reset Functionality (5%) - Upon completion of a game or when the reset button is pressed, a player is brought back to the word setting phase of the game.

**High-Level Schematic:**



**Figure 1:** Input and Output Relationships of Our Project. The blocks named Reset Button, Keyboard, LEDs, 7-Segment Display, and VGA refer to the physical devices and are not included in verilog modules. Keyboard Reader handles PS2 logic, VGA creates and displays graphics using hardcoded bitmaps, and Hangman Game Logic handles operation of the game and signal routing between modules.



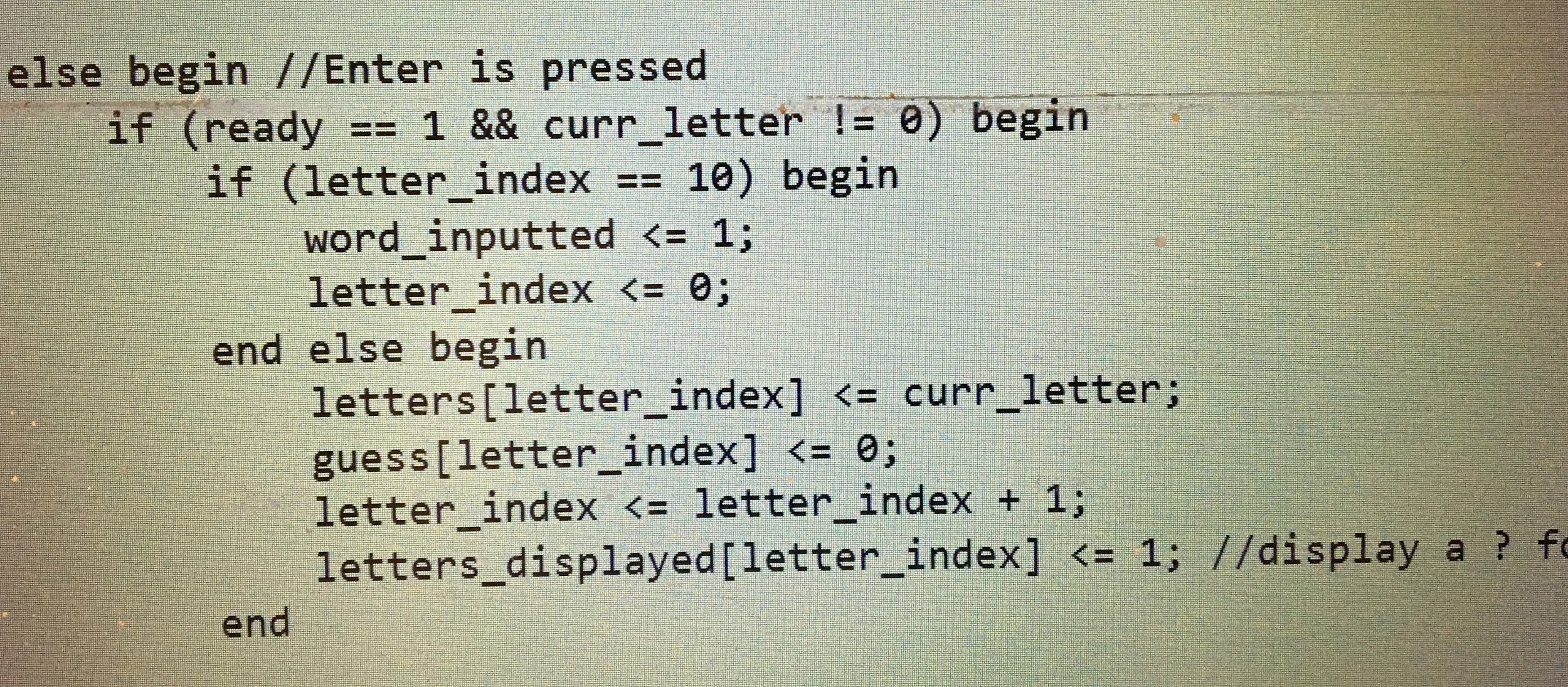
**Figure 2:** Module Hierarchy. Each of the higher level modules contains both logic and submodules.

**Hangman Top Module:**

This top module is named hangman.v. The top module initializes instances of the vga module and the keyboard\_reader module. This module also contains the main game logic.

**Main Game Logic:**

The scancode output from the keyboard reader module is stored in an eight bit register. Single bit registers are used to store game state for inputting a word, guessing mode, storing game result, and storing whether or not each of the ten letters has been guessed. We use a 10 8-bit registers to store the scan code values for each of the letters in the correct answer. When storing these values, we used a state machine to iterate through the letter positions and store each value. The letter input is not final until the user selects “Enter”. To finalize the input word, the user selects “\”, which was chosen as it is an arbitrary key that was not used for any other game functions. In guessing mode, the number of incorrect guesses are counted and checked against the registers determining whether or not each letter has been guessed. If all letters are guessed before 6 incorrect guesses, “YOU WIN” is displayed on the monitor. If 6 incorrect guesses are used before the letters are correctly guessed, “YOU LOSE” is displayed on the monitor. Each time the user makes a mistake we display a part of the hangman figure by passing the number of incorrect guesses to the VGA module.



**Figure 3:** Sample Code Snippet for Processing Input to Store the Correct Answer.

**VGA Module:**

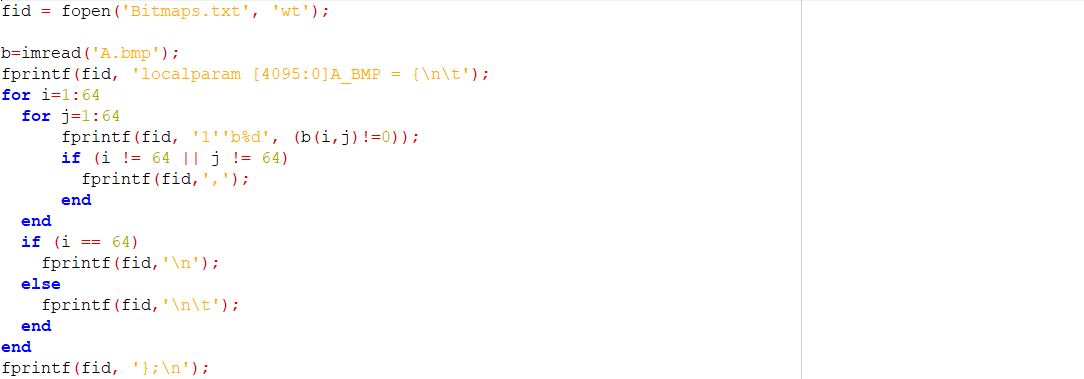
We adapted this module from the NERP Demo linked in the CCLE documentation. (<https://www.element14.com/community/thread/23394/l/draw-vga-color-bars-with-fpga-in-verilog>) The vga module switches data values to the display with the correct clock cycle. This includes constant graphics, such as the gallows, background, and letter regions as well as the logic for switching constant parameter bitmaps to the letter regions. This module takes input from a 25 MHz clock, a reset signal, and ten letter value signals for display. It outputs the rgb values for each pixel, as well as the hsync and vsync signals. Since the demo contained code to display the letters NERP on the 7-segment display, we adapted this to display HANG to be consistent with our game.

**VGA640x480:**

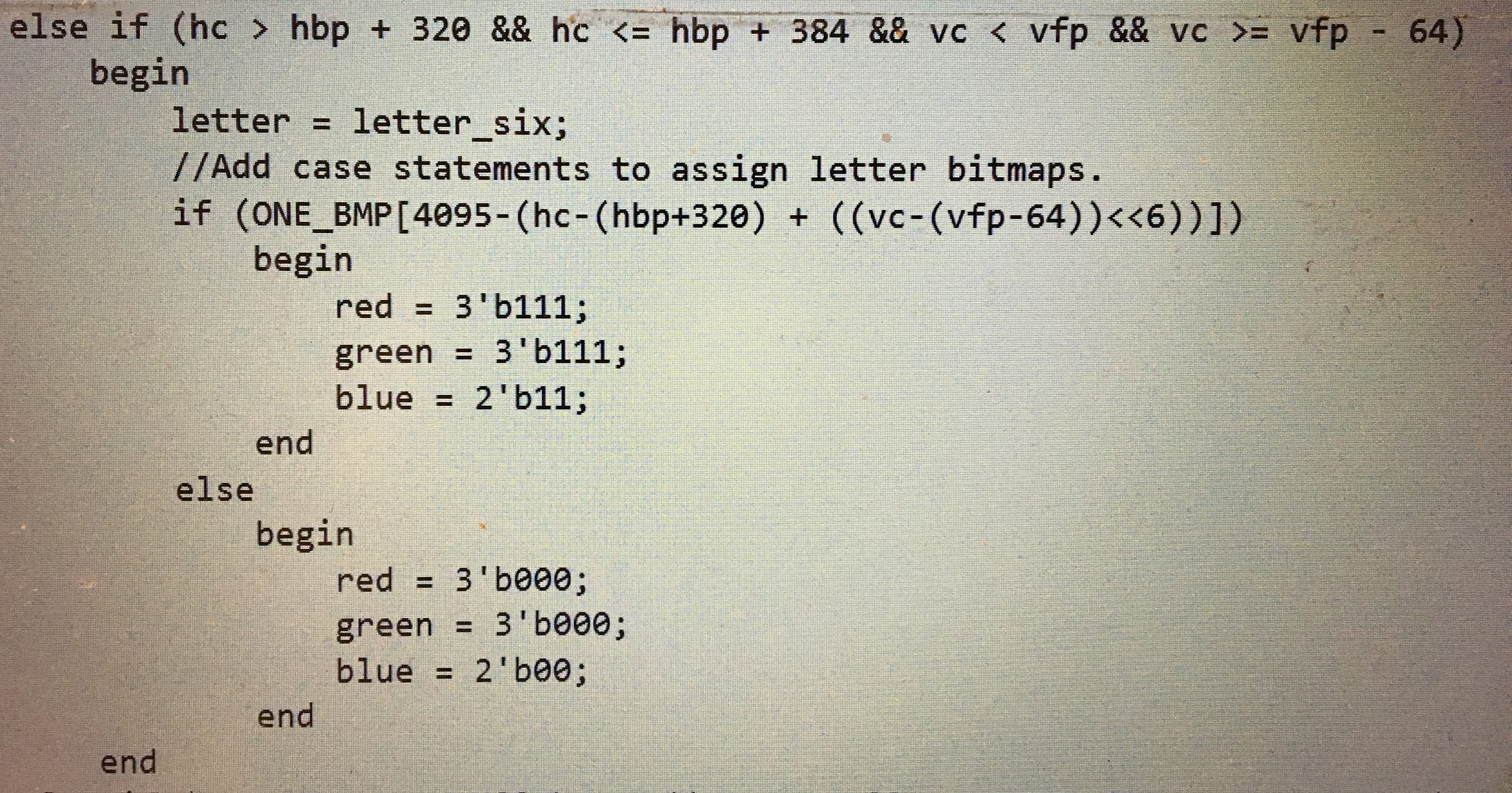
This module contains the default constants for the writable region of the VGA display as well as the sync constants. To display letters, we create 28, 4096 bit register arrays to store constant bitmaps for each character: the 26 letters of the alphabet, plus a dash and a question mark. The other temporary figures we displayed on the screen were the segments of the stick figure. We used the equation of circle, namely, x^2+y^2=r^2, to set inner and outer radius of the line for the head of the stick figure. Arms and body were simple vertical and horizontal ranges written in black. The legs were created by using equations for lines with a slope of +/- 2 contained within two parallel lines for each leg contained by upper and lower limits. For the letters, we defined 10 64x64 pixel regions. Whenever we entered these regions, we assigned pixels as either black or white based on the correct bitmap for the letter set by the inputs from the main logic module. We also created a constant gallows symbol on a light blue background.

**Note on Code Generation for Letter Bitmaps:**

Creating bitmaps for the alphabet and correctly routing them to the intended pixels in a way that did not slow down synthesis too much was challenging. We used a free bitmap generator application (Codehead’s Bitmap Font Generator) to generate 64x64 pixel maps for the letters we wished to display in a .bmp format, then we used a matlab script we wrote to convert this into more than 1000 lines of constant declarations that fit Verilog syntax.



**Figure 4:** Screenshot of the MATLAB Script Used to Generate the Bitmap Constants for Letter A. This code is included in the submission and provided an easy way to update the appearance of the fonts if we wanted to. We could simply generate new .bmp files using a new font from the bitmap generator, then run this script and copy and paste the result into our files.



**Figure 5:** Code Snippet Showing Indexing into Bitmaps to Display Text.

**Clockdiv:**

This module was included in the Nerp Demo. We modified the clock rate since the clock rate of the board the demo was written for was slower than that of the Nexys3.

**Segdisplay:**

We only modified this module to write the new message on the 7-segment displays.

**Keyboard Reader:**

This module along with the other keyboard modules are adaptations of a git repository responsible for allowing keyboard inputs for a basys 3 FPGA. This repository can be found here <https://github.com/Digilent/Basys-3-Keyboard>. The keyboard reader module takes the PS2 clock and PS2 data values and converts them into 16 bit PS/2 scancodes. This module contains a PS2\_receiver module that outputs the 16 bit PS/2 scancode. The first 8 bits of the scancode are used for determining various commands associated with the key such as it being held down, released, combined with shift, caps lock being active and so on. The last 8 bits refer to the key itself. In order to make the output from this module useable in the game logic finite state machine, the first 8 bits are checked to see if they correspond to a keyrelease. In these cases the outputted key scancode would be 0 and in all other cases the key that was received is outputted.

**PS2Receiver Module (Keyboard Module):**

This module is responsible for interpreting the PS2 data and clock values received from the USB-HID interface. It first debounces the data and clock inputs to obtain clean signals that can be used to fill a data buffer corresponding to the inputted scancode. Negedge values are used on the clock since according to the Nexys 3 manual, values read from PS2Data are only valid upon the negative edges of PS2Clk.

**Debouncer Module:**

We used this unmodified debouncer module from the example program to debounce the keyboard clock and the signals from the keyboard.