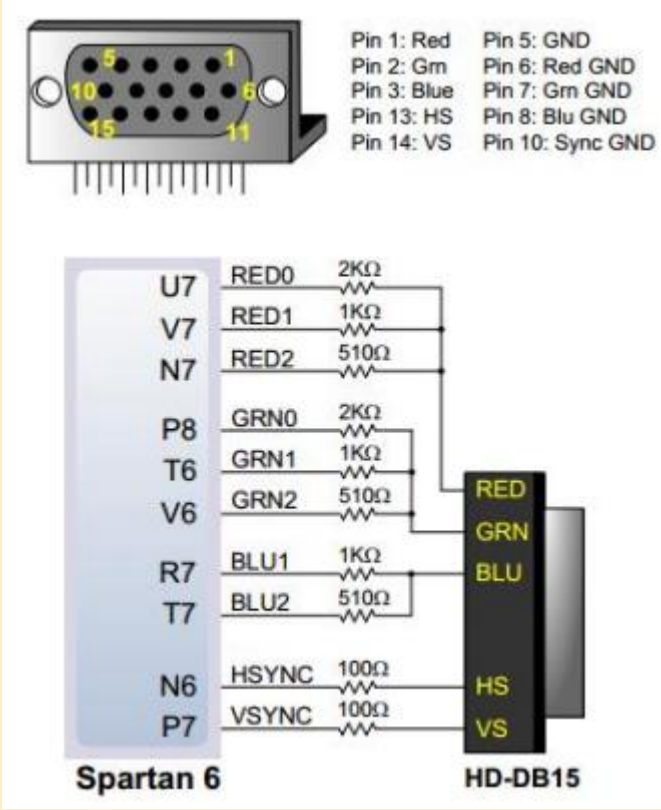
Part A

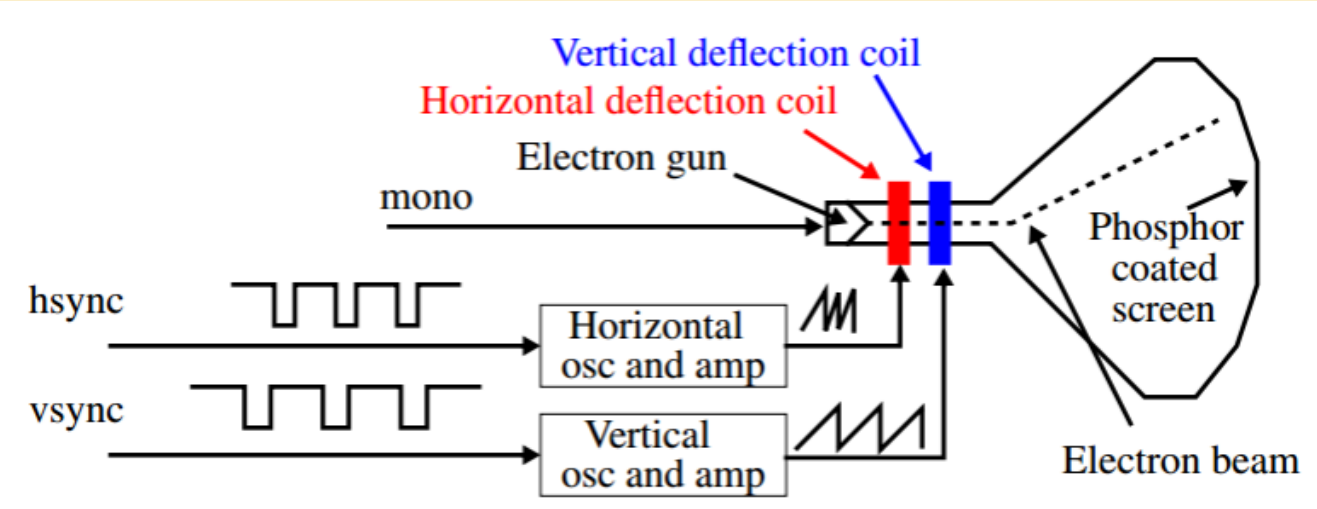
VGA INTERFACE

- Nexys3 board uses 10 FPGA signals to create 8-bit color and two standard sync signals (HS- Horizontal Sync, and VS- sync).



VGA (VIDEO GRAPHICS ARRAY)

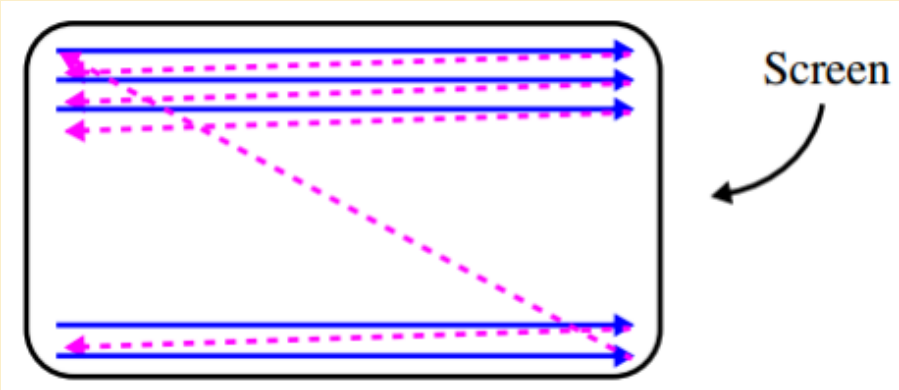
- Here we consider an 8 color 640-480 pixel resolution interface for the CRT.



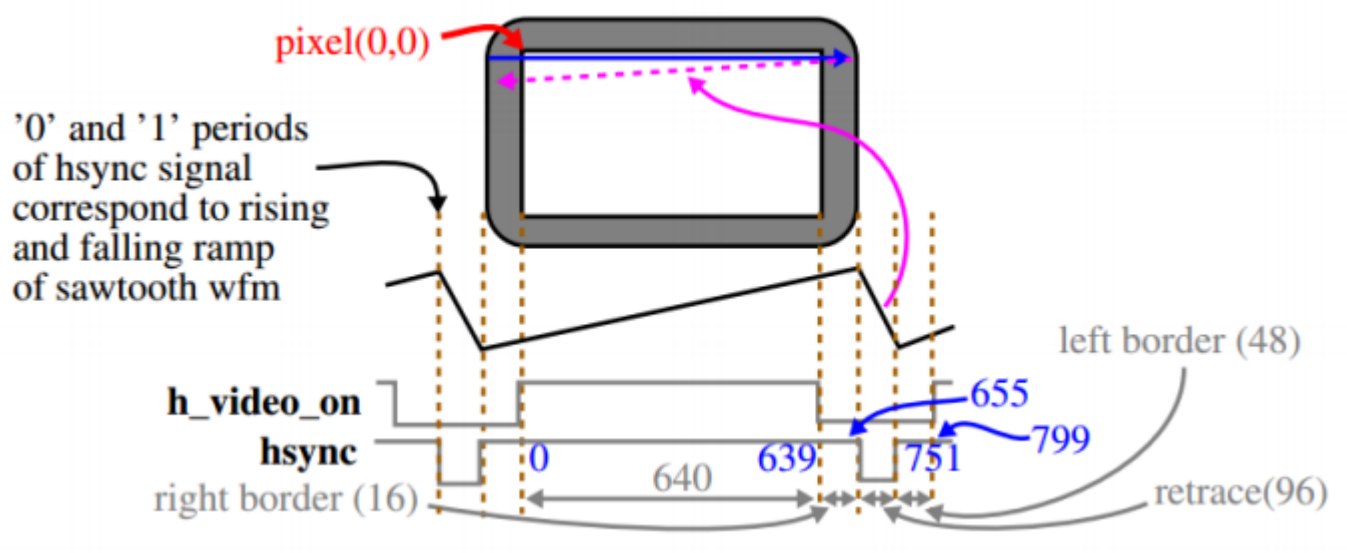
- The electron gun generates a focused electron beam that strikes the phosphor screen.
- The intensity of the electron beam and the brightness of the dot are determine by the voltage level of the external video input signal (mono signal).
- The mono signal is an analog signal whose voltage level is between 0 and 0.7 V.
- The horizontal and vertical deflection coils produce magnetic fields guide the electron beam to points on the screen.

VGA (VIDEO GRAPHICS ARRAY)

- The electron beam scans the screen systematically in a fixed pattern.



- The horz and vert. osc. and amps gen. saw tooth waveforms to control the deflection coils.

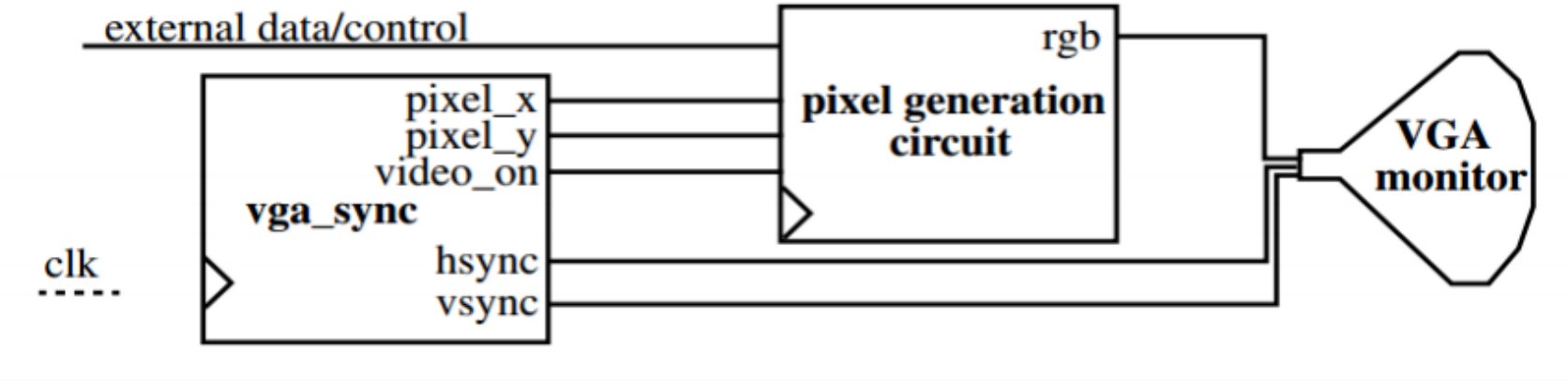


VGA (VIDEO GRAPHICS ARRAY)

- A color CRT is similar except that it has three electron beams, that are projected to the red, green and blue phosphor dots on the screen.
- The three dots are combined to form a pixel.
- The three voltage levels determine the intensity of each and therefore the color.
- The VGA port has five active signals, hsync, vsync, and three video signals for the red, green and blue beams.
- They are connected to a 15-pin D-subminiature connector.
- The video signals are analog signals -- the video controller uses a D-to-A converter to convert the digital output to the appropriate analog level.
- If video is represented by an N-bit word, it can be converted to 2 Nanalog levels.
- Three video signals can generate 2 3N different colors (called 3N-bit color).
- If 1-bit is used for each video signal, we get 2 3or 8 colors.
- If all three video signals are driven from the same 1-bit word, we get black & white

VIDEO CONTROLLER

Red (R)	Green (G)	Blue (B)	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White



VIDEO CONTROLLER

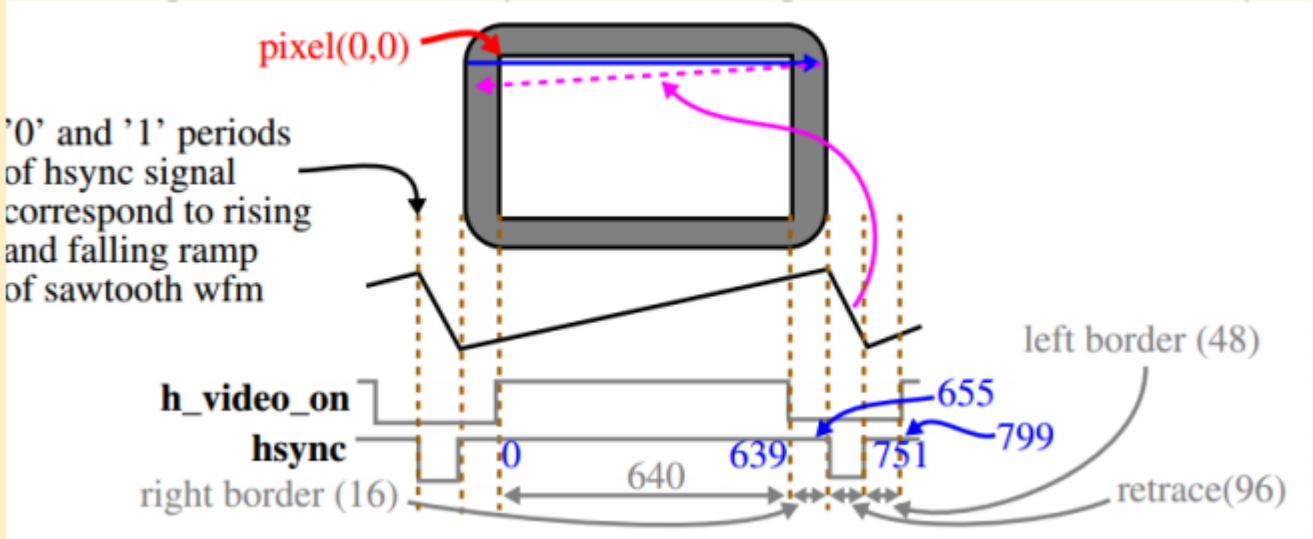
- The vga_sync generates the timing and synchronization signals.
- The hsync and vsync are connected directly to the VGA port.
- These signals drive internal counters that in turn drive pixel_x and pixel_y.
- The video_on signal is used to enable and disable the display.

- Pixel_x and pixel_y indicate the relative positions of the scans and essentially specify the location fo the current pixel.
- The pixel generator circuit generates three video signals -- the rgb signal.
- The color value is derived from the external control and data signals.
- The vga_sync circuit generates the hsync signal, which specifies the time to traverse (scan) a row, while the vsync signal specifies the time to traverse the entire screen.
- Assume a 640x480 VGA screen with a 25-MHz pixel rate (known as VGA mode).

- The screen usually includes a small black border around the visible portion.
- The top-left is coordinate (0, 0) while the bottom right is coordinate (639,479)

VIDEO CONTROLLER

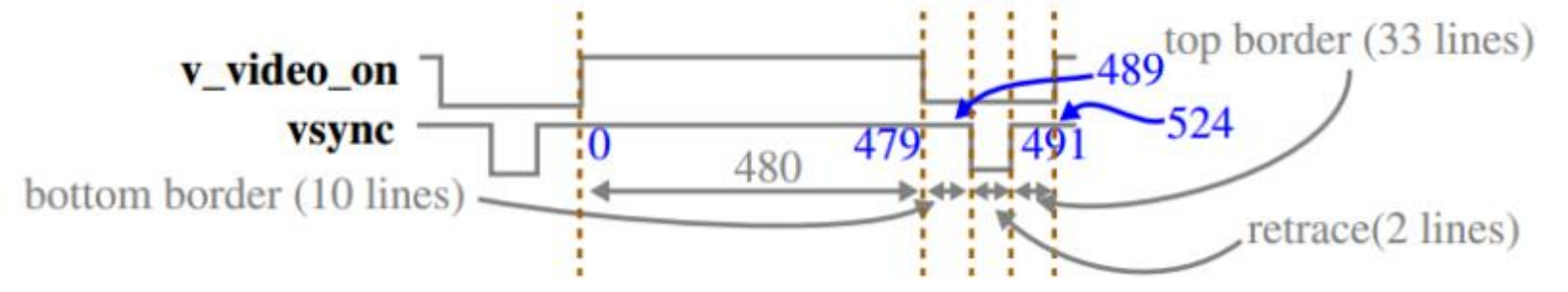
- One full period of the hsync signal contains 800 pixels and is divided into 4 regions:



- Display: Visible region of screen -- 640 pixels.
- Retrace: Region in which the electron beam returns to left edge. Video signal is disabled and its length is 96 pixels.
- Right border: Also known as the front porch (porch before retrace). Video signal is disabled and its length is 16 pixels (may differ depending on monitor).
- Left border: Also known as the back porch. Video signal is disabled and its length is 48 pixels (may differ depending on monitor).

VIDEO CONTROLLER

- The hsync signal is obtained by a special mod-800 counter and a decoding circuit.
- The counter starts from the beginning of the display region.
- This allows the counter's output to be used as the x-axis coordinate or pixel_x signal.
- The hsync is low for the counter interval 656 (640+16) to 751 (640+16+96-1).
- The h_video_on signal is used to ensure that the monitor is black in the border regions and during retrace. It is asserted when the counter is smaller than 640.



- The time unit of the movement is in terms of the horizontal scan lines.
- One period of the vsync signal is 525 lines, and has a corresponding set of four regions.

WORKSHEET

Modify the given VHDL file such to:

- Create a horizontal strip of width 40 pixels in the middle with blue color.
- Create a vertical strip of width 20 pixels in the middle with green color.
- Set the background color to be black.

Part B

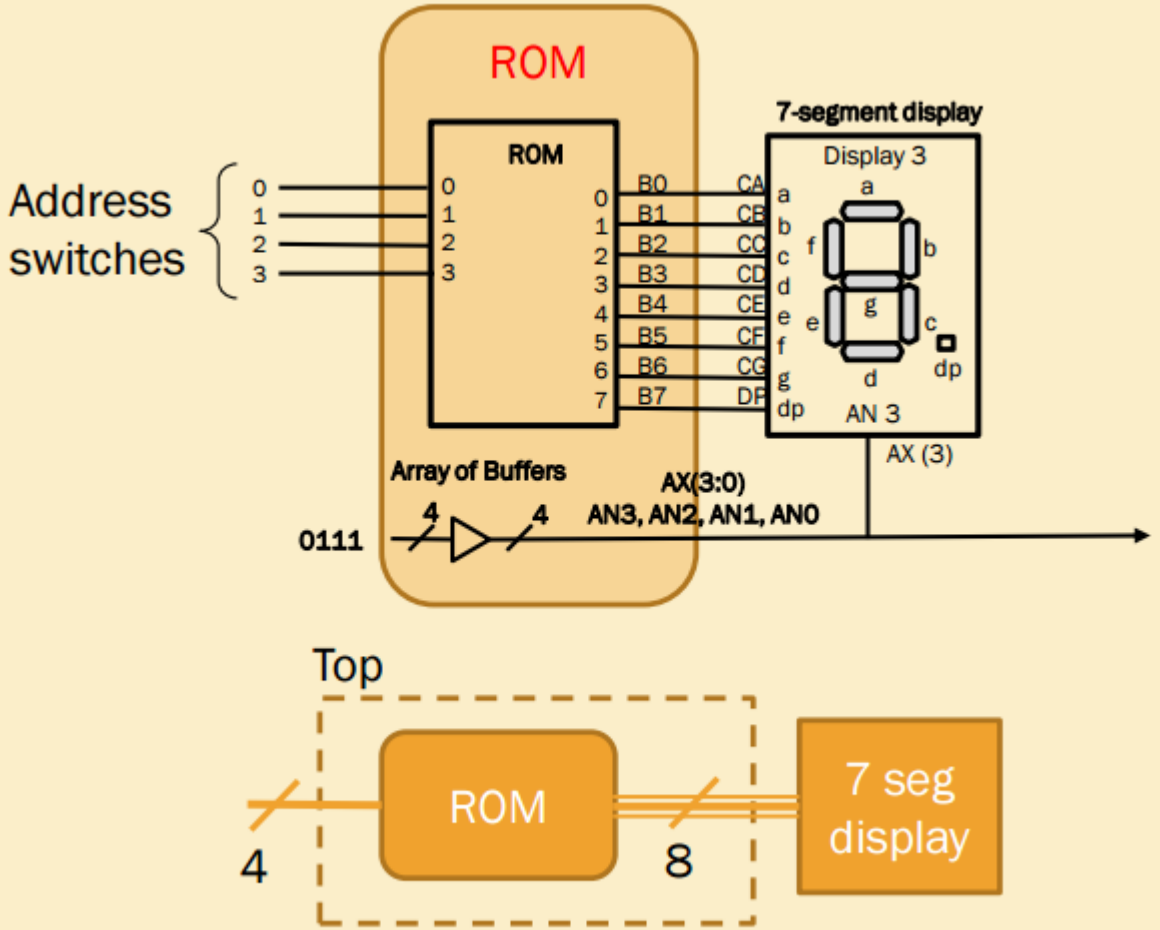
DESIGNING AND TESTING A "DIGITAL STOP WATCH"

Project DigitalClock (Top Module)

* Read Only Memory

This lab shows another way to implement a seven segment display decoder using a memory map.

The main advantage is that you can change the assignment of display values without having to redo the VHDL code



A QUICK INTRO TO COE FILES

- Most IP Core generated files use a coefficients file, COE file, to set the memory or default state of some generated core. We will be using one to set up our Read Only Memory (ROM).
- The COE file is a plain text file with a radix instantiation and a list of values for the memory to have at a given spot.
- All values in the COE file should be in the radix that is stated in the start of the file.

COE FILE EXAMPLE

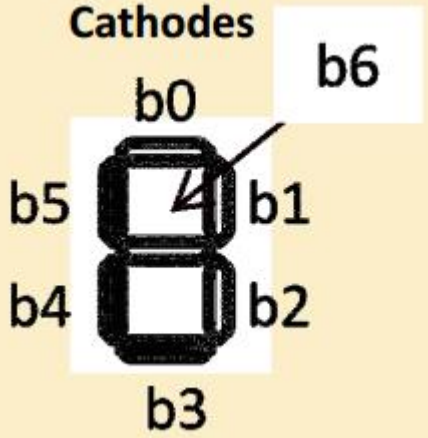
- An Example of a COE file for a ROM is shown on the right.
- Please open a text editor such as notepad to create this file.
- We want to load it with our pre-determined memory. The bit-patterns for the Seven Segment Display.

```
; This .COE file is for use with a ROM
; memory of depth=4, width=8.
; Values specified in hexadecimal format.
memory_initialization_radix=16;
memory_initialization_vector=
80,
F9,
03,
00;
```

COMPUTING COE CHARACTER CODES

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	F_1	F_0	COE Values
0	0	0	1	1	1	1	1	1	3	F	3F
1	0	0	0	0	0	1	1	0	0	6	06
2	0	1	0	1	1	0	1	1	5	B	5B
3	0	1	0	0	1	1	1	1			
4	0	1	1	0	0	1	1	0			
5	0	1	1	0	1	1	0	1			
6	0	1	1	1	1	1	0	1			
7	0	0	0	0	0	1	1	1			
8	0	1	1	1	1	1	1	1			
9	0	1	1	0	0	1	1	1			
A	0	1	1	1	0	1	1	1			
b	0	1	1	1	1	1	0	0			
C	0	0	1	1	1	0	0	1			
d	0	1	0	1	1	1	1	0			
E	0	1	1	1	1	0	0	1			
F	0	1	1	1	0	0	0	1			

- Please complete the table before opening the COE file these are the values you will need to enter in the file.

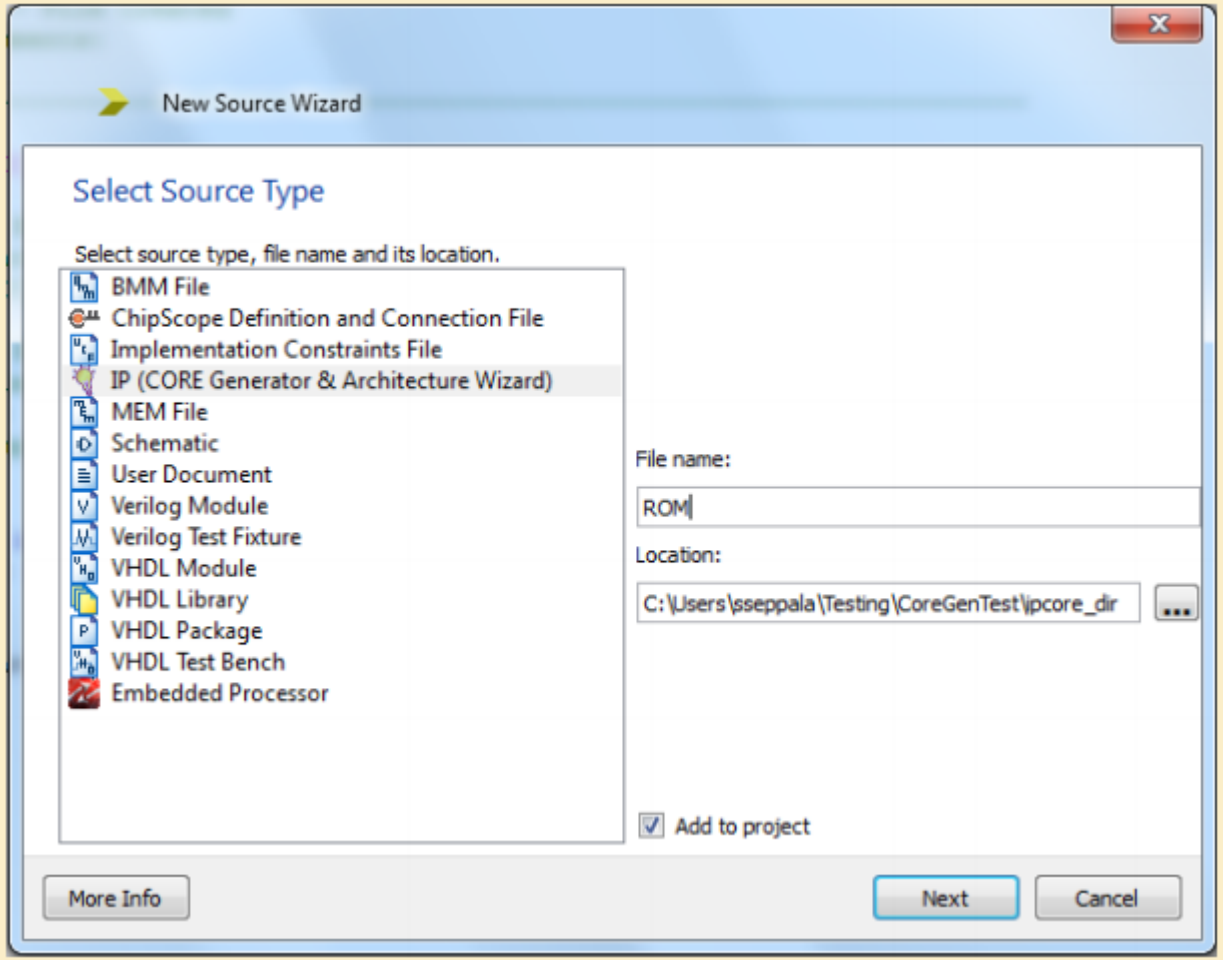


CREATE A COE FILE

- Now you must create a COE file with Seven Segment Display (SSD) patterns listed in which SSD patterns are listed in descending order, starting from zero.
- It is recommended that all values are inserted in hexadecimal.
- Remember that the order in which values are listed is the order in which they will be addressed. For example the first value will be at the lowest address, and the next value will be one more than that address, etcetera.

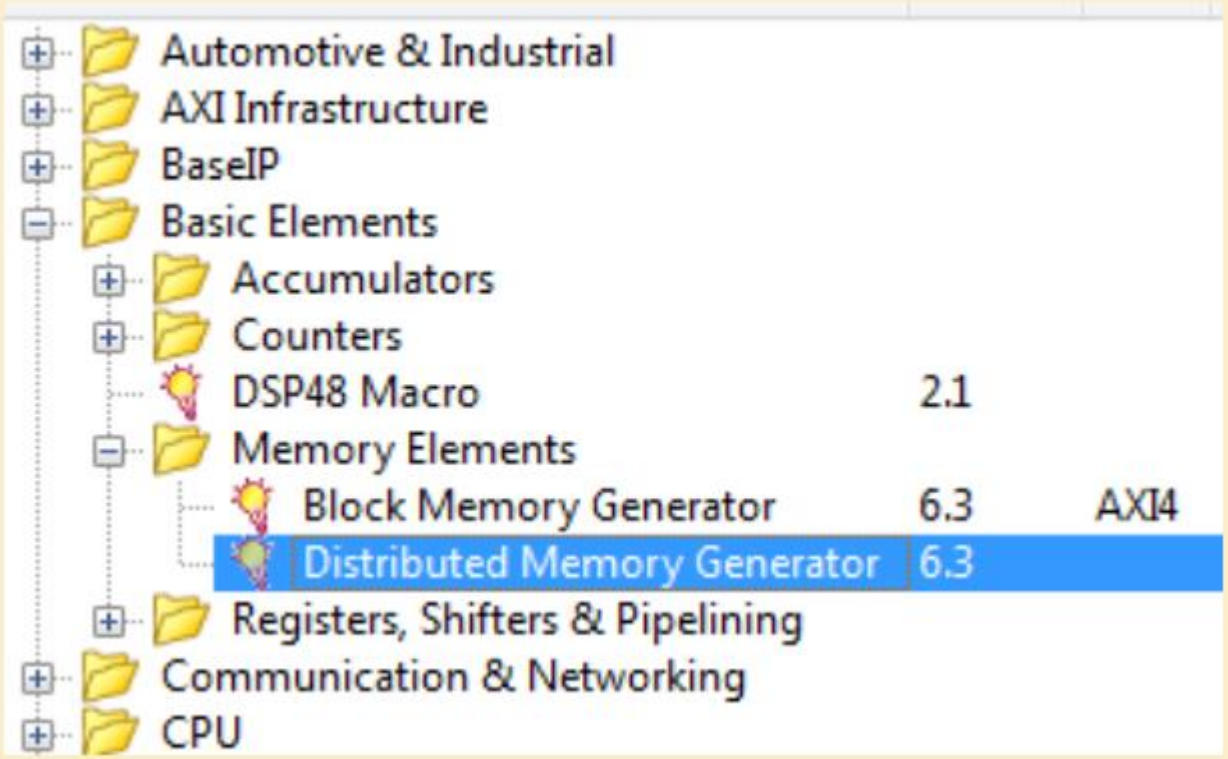
GENERATING A CORE FILE

- Ensure that you have at least one VHDL module in your project before you begin. As seen here, I have a module that I decided to name “Top”.
- Now we need to add a new source, right click in your design hierarchy and select “New Source”.
- Once “New Source” has been selected, choose “IP Core” (The lightbulb icon) and give it a name. In this example I will be calling it “ROM”.

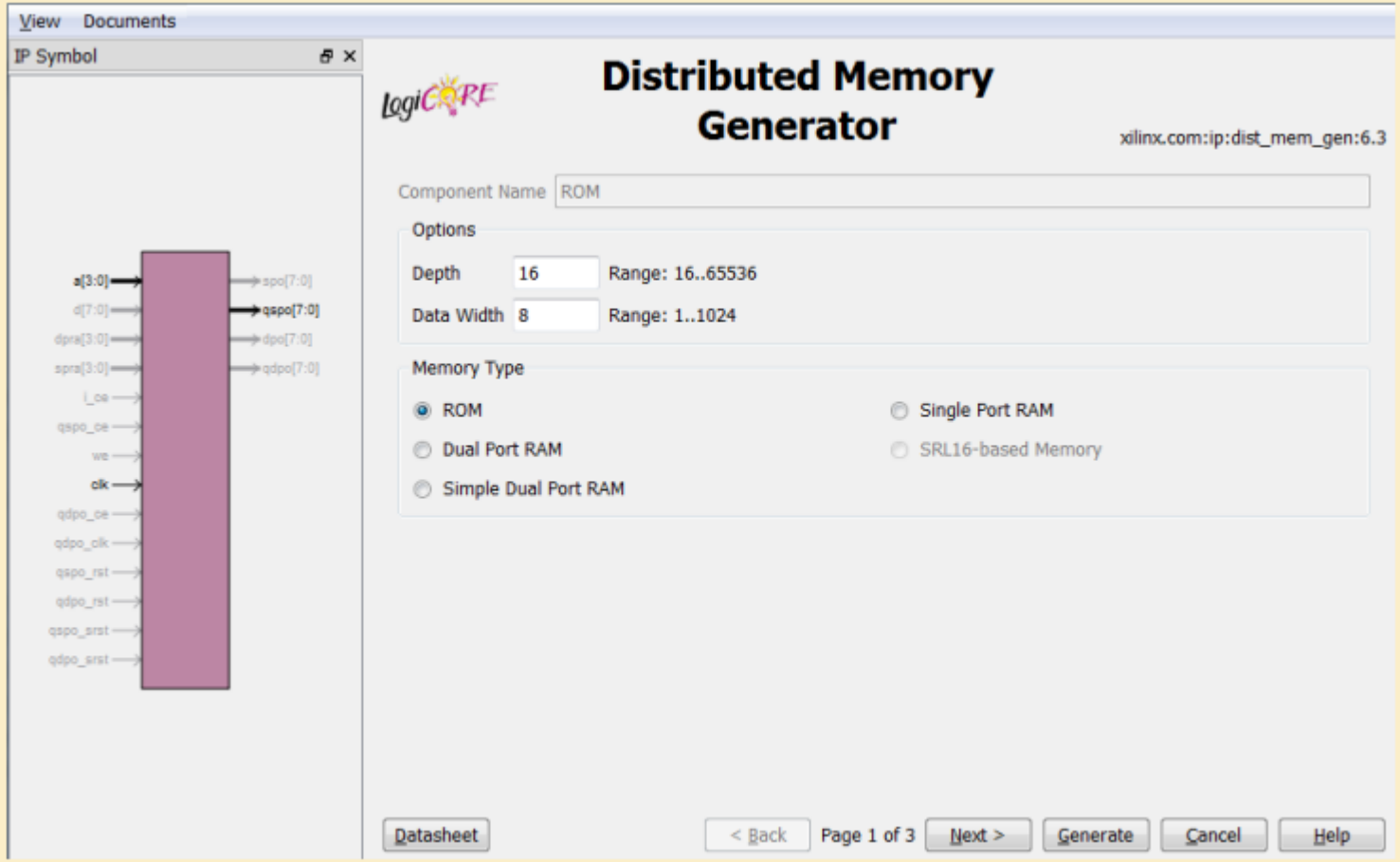


CHOOSING OUR IP-CORE

- For this lab, we want to use a ROM (Read Only Memory). To get to this, select Basic Elements > Memory Elements > Distributed Memory Generator

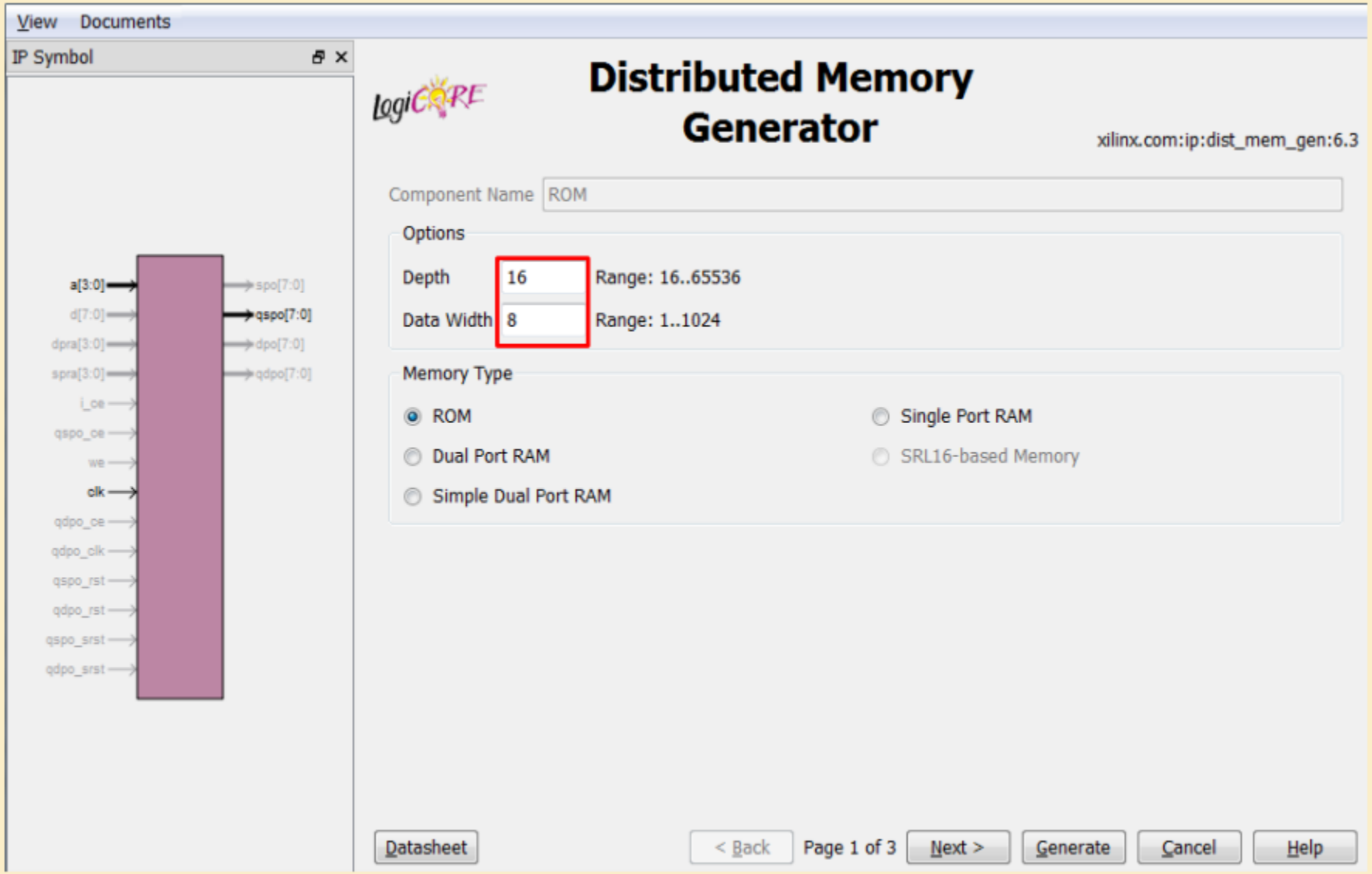


DESIGNING THE CORE



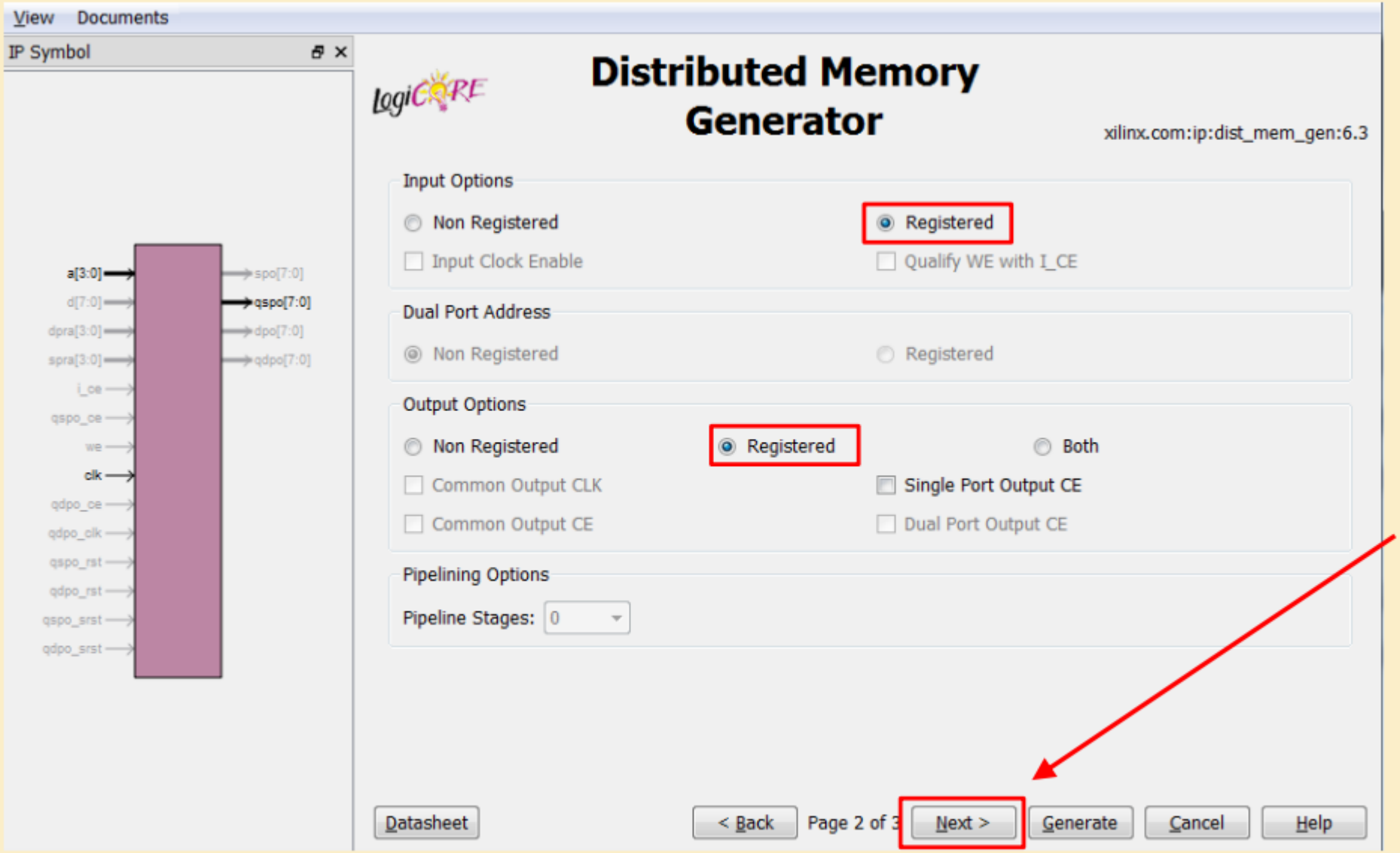
- We will now be presented with a GUI and a block diagram of the IP Core that we are trying to generate. It is up to us to build it to our desired specifications. This is where all the digital logic comes together.

DESIGN GUIDANCE FOR THE CORE



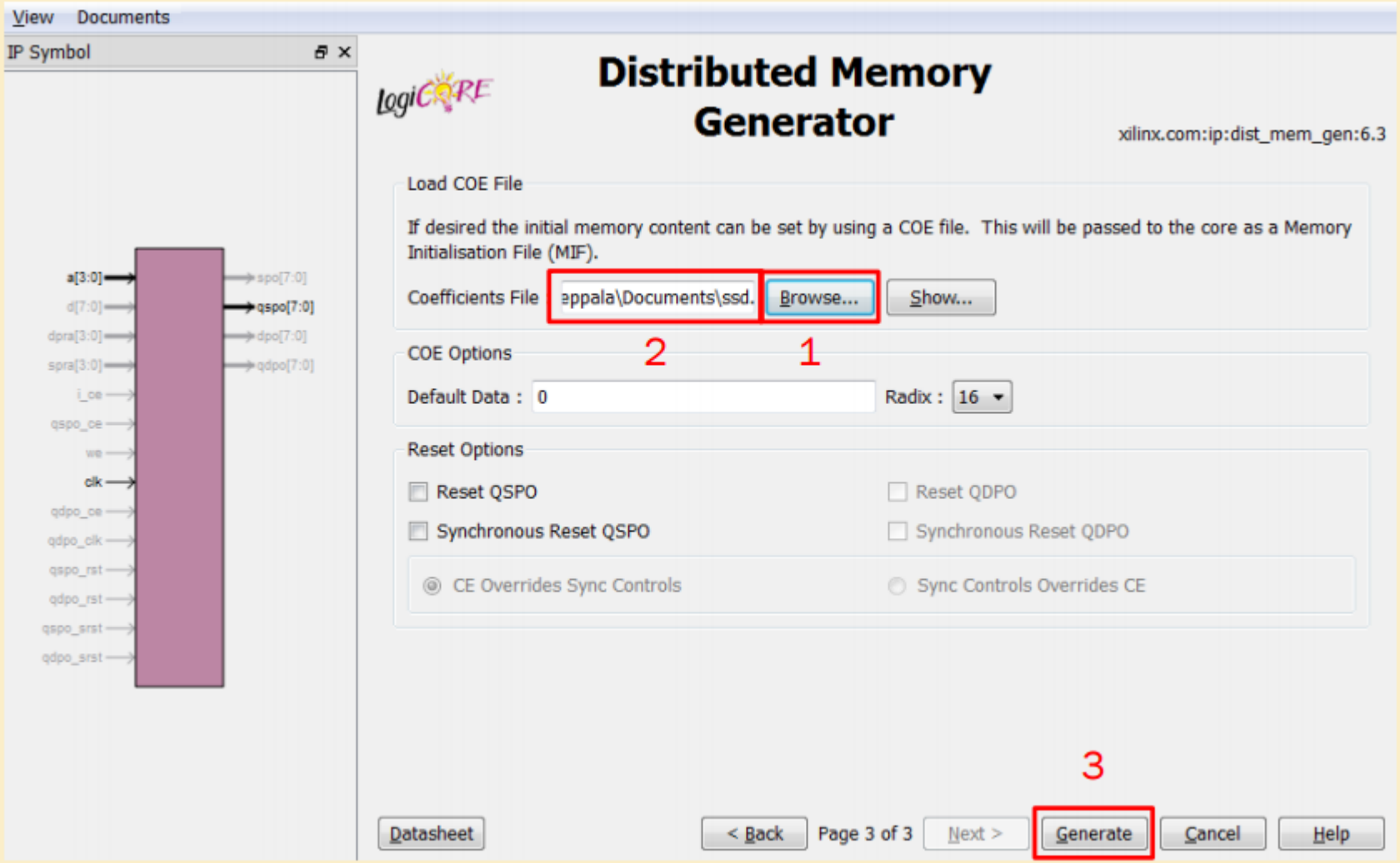
- The characteristics that we want to utilize is a ROM with a data width of 8 bits and a depth of 16 (the closest power of 2 that gives us all the values that we want to use).

I/O OPTIONS



- We want all of our I/O to be registered (synchronous).
- When you are done click on the next button to move to the next screen.

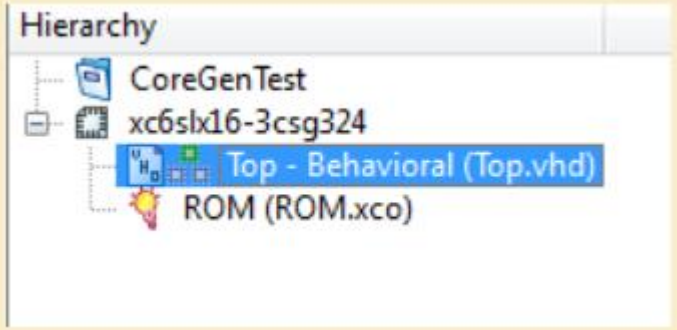
GENERATING THE COE FILE



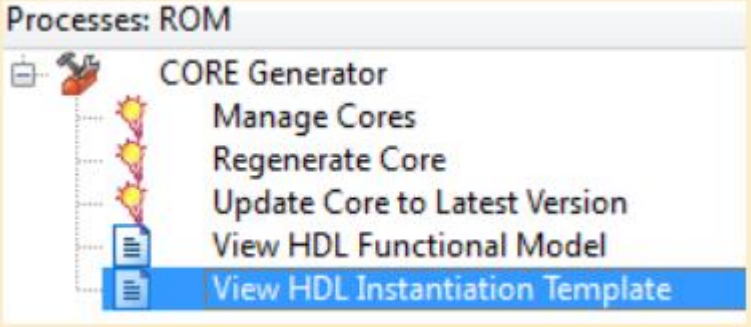
- 1. Select the COE file from the work area.
- 2. When the COE file is formatted correctly, it will appear in black. Otherwise it will be red.
- 3. Click “Generate” to continue.

USING THE GENERATED CORE

- Once the core has been generated we will have a new file in the design list.



- Select this file, and then in the window below, select “View HDL instantiation template”.



ATTACH TO THE TOP MODULE

- Now all that we have to do is copy and paste the component declaration and the portmap into the appropriate places.
- The last thing we want to do, as a safety measure, is uncomment the “UNISIM” libraries. They will be commented out by default.
- An example of the ROM module being used in a project is shown on the next slide.
- All of the cores that are generated are self-contained and can be used within any module, they do not need their own file to be used, this was done for demonstration purposes only.

USING THE ROM IN TOP MODULE

- Final Example VHDL Module

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  library UNISIM;
4  use UNISIM.VComponents.all;
5
6  entity Top is
7      Port (  clk    : in  STD_LOGIC;
8              ins    : in  STD_LOGIC_VECTOR (3 downto 0);
9              outs   : out STD_LOGIC_VECTOR (7 downto 0));
10 end Top;
11
12 architecture Behavioral of Top is
13
14 COMPONENT ROM
15   PORT (
16     a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
17     clk : IN STD_LOGIC;
18     qspo : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
19   );
20 END COMPONENT;
21
22 begin
23
24 your_instance_name : ROM
25   PORT MAP (
26     a => ins,
27     clk => clk,
28     qspo => outs
29   );
30
31 end Behavioral;
```

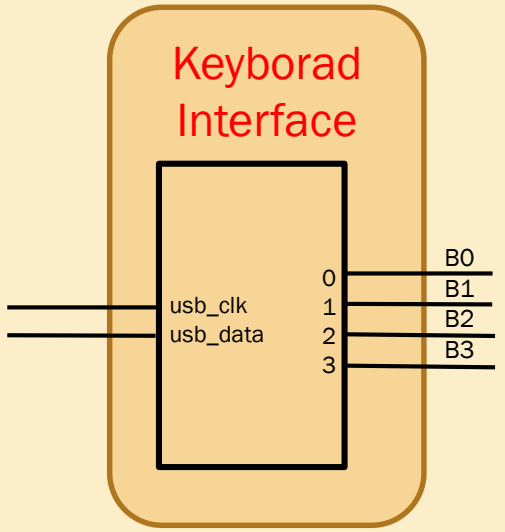
Part C

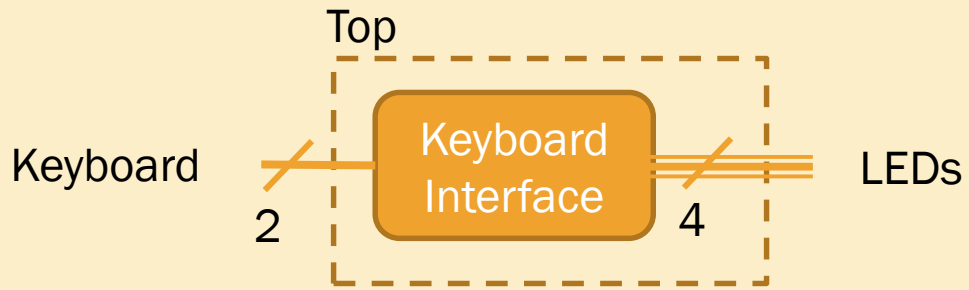
DESIGNING AND TESTING A "SERIAL INTERFACE"

Project Keyboard

- Keyboard Interface

This lab will focus on developing a USB serial interface for reading keyboard scan codes.

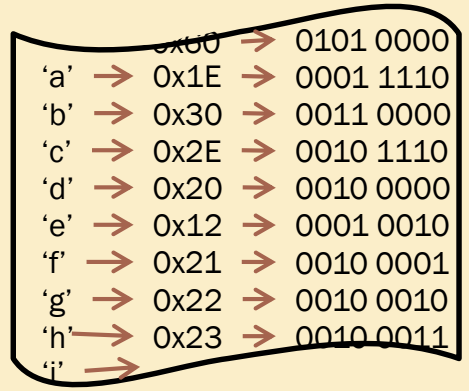




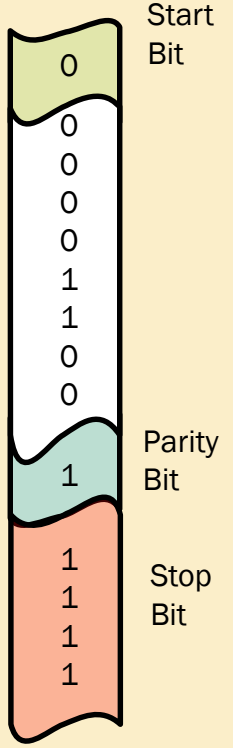
READING DATA FROM KEYBOARD

- Each key pressed is sent as a scan code or a 2 digit hexadecimal.
- The number is then sent through the USB port to the FPGA board.
- The number is encoded as an asynchronous serial sequence

Keystrokes are processed one character at a time



Each Character is converted to binary form.

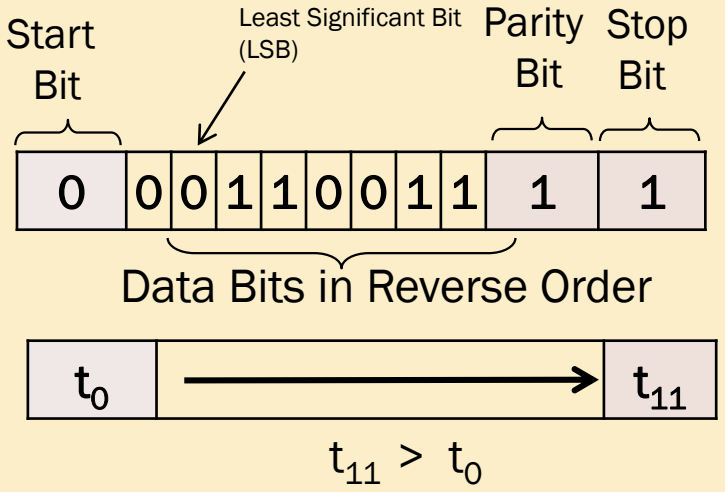


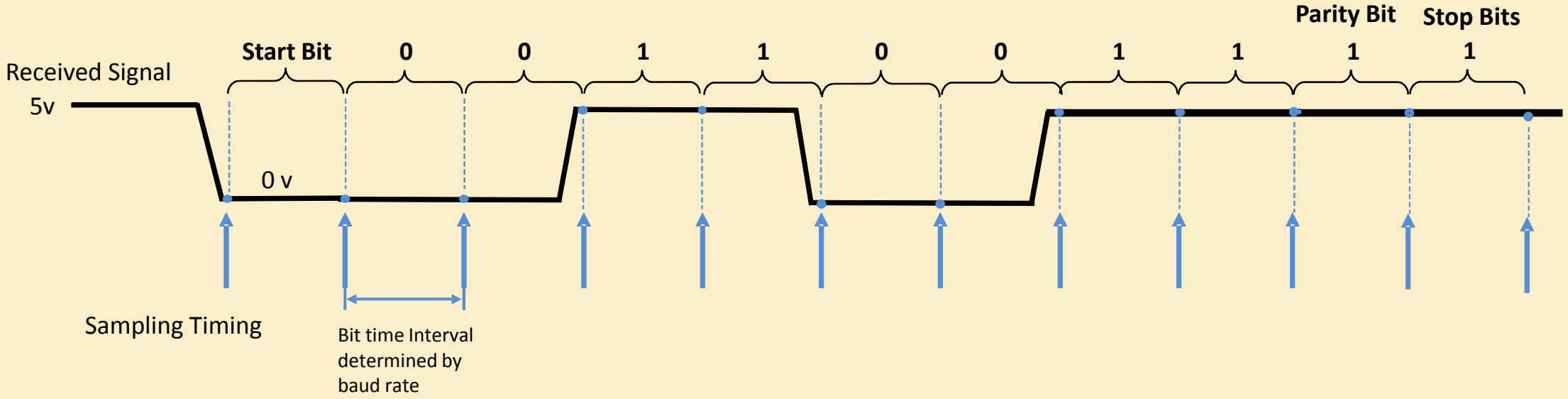
Character "b" wave form is composed

Note: Least Significant Bit (LSB) first

SERIAL COMMUNICATIONS

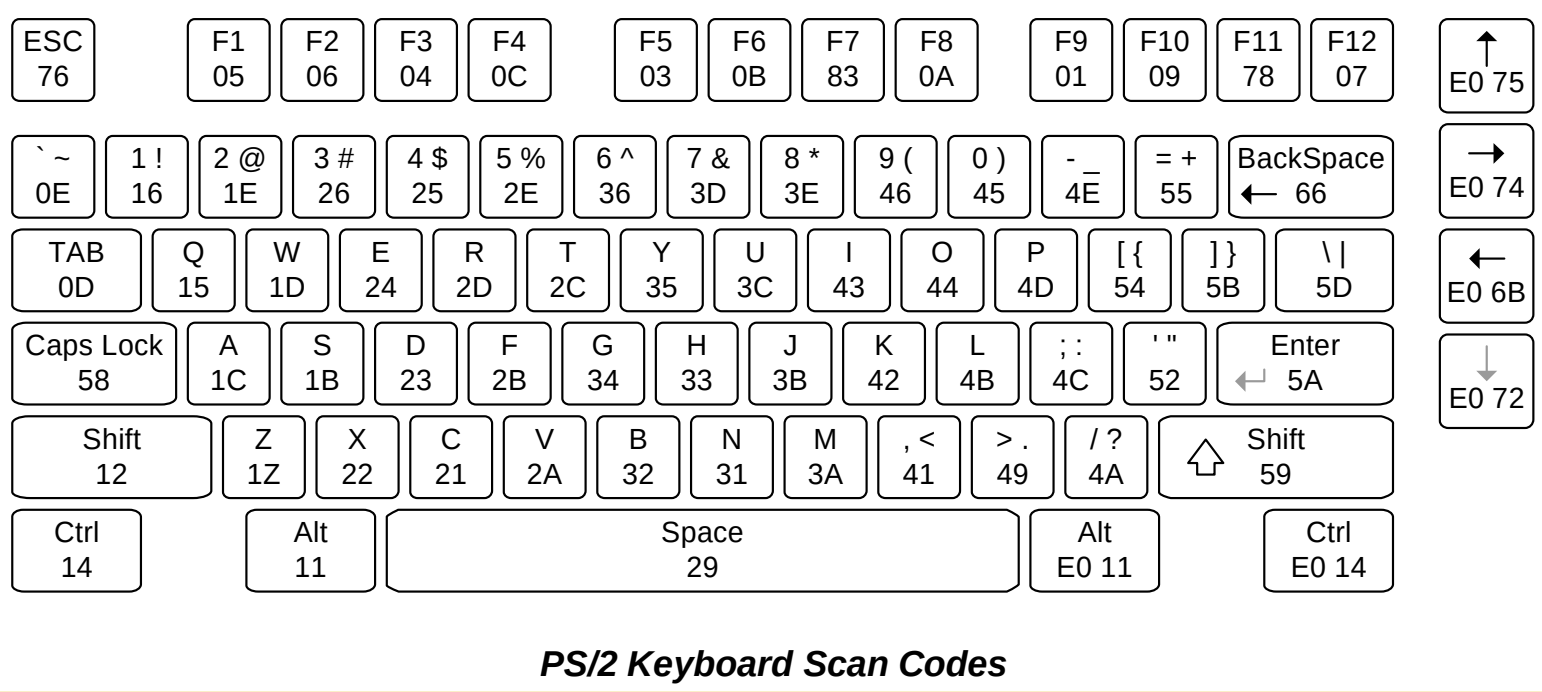
- The UART packet structure consists of 10 bits- 1 START bit, 8 DATA bits (one byte), and 1 STOP bit. This structure is shown below, and is commonly referred to as N 8 1 (8 data bits, no parity, 1 stop bit).
- Sometimes an additional bit is included for parity checking (a simple form of error detection)
- The data byte is sent Least Significant Bit (LSB) first, which means we effectively send the byte in reverse order. For instance, if we wanted to send the byte, 11001100, we would transmit the following 10-bit sequence: 0001100111 (one start bit (0), the 8 data bits LSB to MSB (00110011), and then one stop bit (1)).





KEYBOARD SCAN CODES

- This is an example set of scan codes for the keyboard in the lab
- These codes are in Hexadecimal. The UART will be using the hexadecimal values.



PS/2 Keyboard Scan Codes

USING THE USB PERIPHERAL.

- The Universal Asynchronous Recevier/Transmitter (UART) will manage receiving the keyboard data. It sends a break code of 0xF0 between each scan code. The sample code is counting on this and you will need to include this in the simulation file. Send the first scan code, followed by 0xF0 scan code (break code) and then the next scan code.
- The data will be stored in its data register.
- We need two process's.
  - Reading in our scancodes and assigning them to a vector that we will call scancode.
  - The other to perform some operation based on the scancode.
- There are two std_logic bits needed for this.

usb_clk	pin L12
usb_data	pin J13

Hint: In the UCF File these pins are accessed using the following. Other wise the USB interface will not receive scan codes

```
NET "KeyboardData" LOC = "J13" | PULLUP;
NET "KeyboardClock" LOC = "L12" | PULLUP;
```

EXAMPLE OF READING UART FOR SCANCODE

```
keyboard_scan_ready_enable : process(KeyboardClock) is

begin
    if falling_edge(KeyboardClock) then
        if bitCount = 0 and KeyboardData = '0' then      --keyboard wants to send data
            scancodeReady <= '0';
            bitCount <= bitCount + 1;
        elsif bitCount > 0 and bitCount < 9 then         -- shift one bit into the scancode from the left
            scancode <= KeyboardData & scancode(7 downto 1);
            bitCount <= bitCount + 1;
        elsif bitCount = 9 then                                -- parity bit
            bitCount <= bitCount + 1;
        elsif bitCount = 10 then                               -- end of message
            scancodeReady <= '1';
            bitCount <= 0;
        end if;
    end if;
end process keyboard_scan_ready_enable;
```

EXAMPLE OF SCANCODE PROCESSING

- For this lab you will need to replace the code in the box with a piece of code that sends the lower half (least significant bits (3 down to 0)) of the scan code to light up 4 LEDs on the board.

```
scan_keyboard : process(scancodeReady, scancode) is

begin
    if rising_edge(scancodeReady) then
        -- breakcode breaks the current scancode
        if breakReceived = '1' then
            breakReceived <= '0';
        elsif breakReceived = '0' then
            -- scancode processing
            if scancode = "11110000" then
                -- mark break for next scancode
                breakReceived <= '1';
            end if;
        end if;

        [                                    ]

    end if;
end process scan_keyboard;
```

NEEDED SIGNALS

- These are the signals needed, and example scancodes.

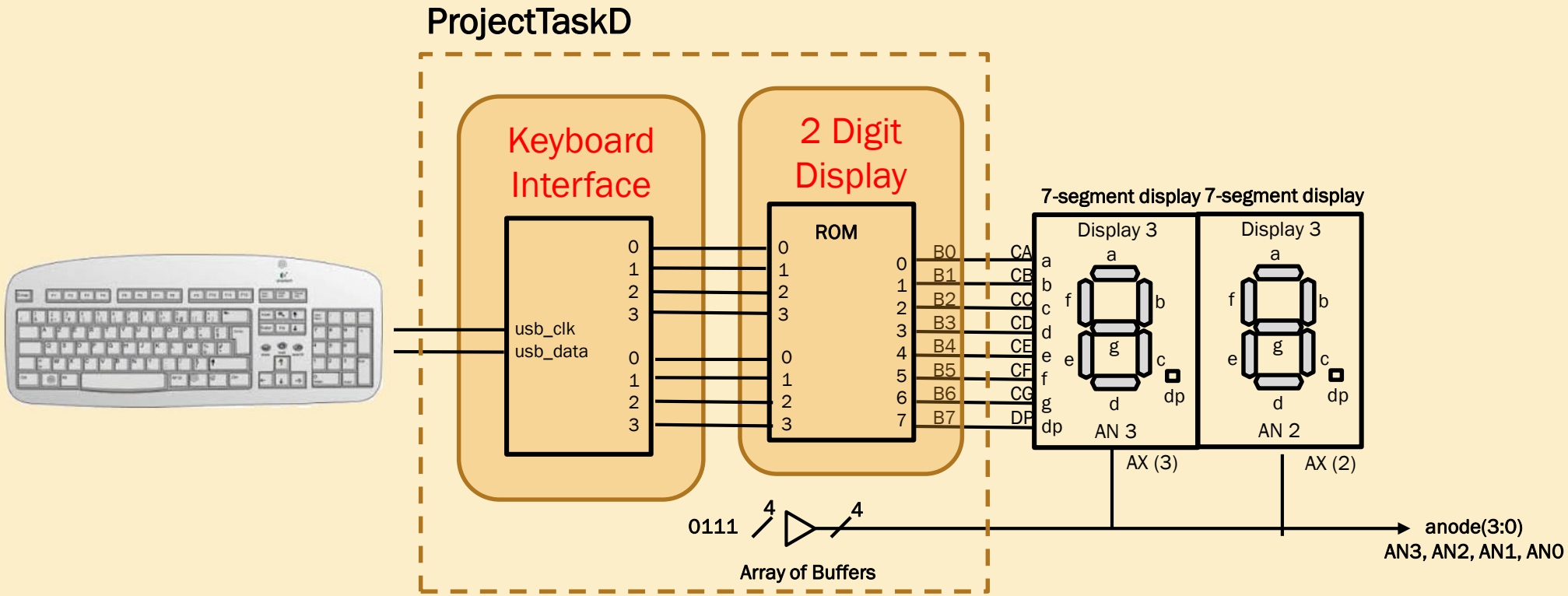
```
architecture Behavioral of KeyboardController is

    signal bitCount      : integer range 0 to 100 := 0;
    signal scancodeReady : STD_LOGIC := '0';
    signal scancode      : STD_LOGIC_VECTOR(7 downto 0);
    signal breakReceived : STD_LOGIC := '0';
```

Part D

TASK OVERVIEW

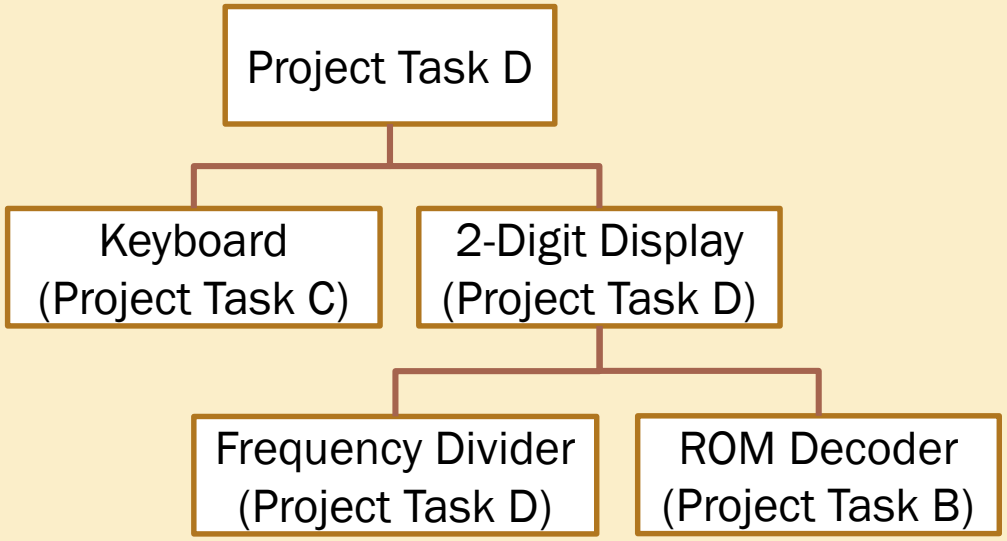
Modules

- ROM 7-Segement Decoder (Project Task B)
- Keyboard Interface (Project Task C)
- 2 Digit Display Driver



STRUCTURAL DESIGN

- This must be a structural design with a top module using the Keyboard (Project Task C) and the seven segment ROM decoder (Project Task B) and 2-Digit Display.



Note:

1. Separate frequency divider to drive the 2 digit display (use a 16-bit counter for frequency divider.)
2. ROM decoder from project B
3. Need to modify project C to output 2 4-bit hexadecimal digits.

2-DIGIT DISPLAY MODULE PROTOYPES

The entity definition for this module is given on the right please note that the module expects 2 digits as 2 4-bit vectors or busses.

```
entity 2_DigitDisplay is
  Port ( firstDigit : in STD_LOGIC_VECTOR (3 downto 0);
         secondDigit : in STD_LOGIC_VECTOR (3 downto 0);
         clkIn : in STD_LOGIC;
         cathodes : out STD_LOGIC_VECTOR (7 downto 0);
         anodes : out STD_LOGIC_VECTOR (3 downto 0));
end 2_DigitDisplay;
```

```
process (ClkIn) begin
  if (rising_edge(ClkIn)) then
    if (state = "1110") then
      displayDigit <= firstDigit;
      state <= "1101";
      anodes <= state;
    elsif (state = "1101") then
      displayDigit <= secondDigit;
      state <= "1110";
      anodes <= state;
    else
      displayDigit <= secondDigit;
      state <= "1110";
      anodes <= state;
    end if;
  end if;
end process;
```

The process on the left manages the seven segment displays so that we can see 2 digits appear. It is essentially a state machine with 2 states one for displaying the first digit and one for the second.

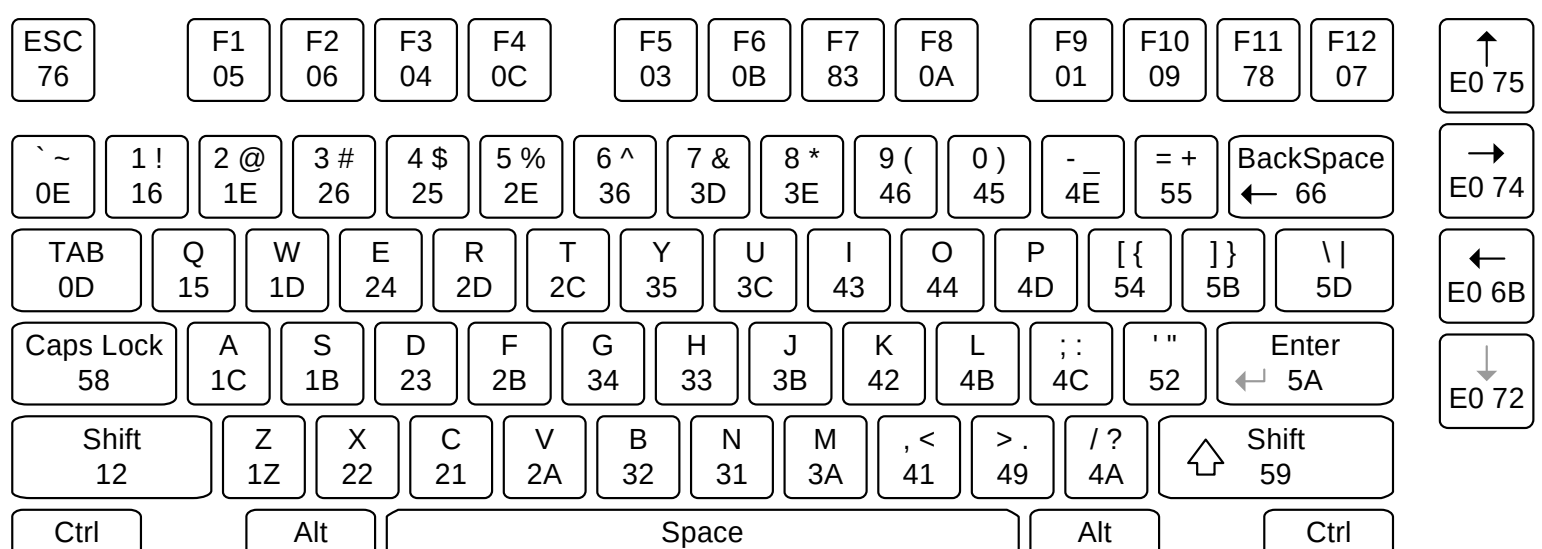
Hint: You will need three port maps that look like this,

```
frgd: freqDiv  port map (systemClk, slClk);
dds:  digitSelector port map (firstDigit, secondDigit, slClk, displayDigit, anodes);
rom1: ROM port map (displayDigit, systemClk, cathodes);
```

You will need to create signals to make these work.

VERIFY KEYBOARD SCAN CODES

- This is an example set of scan codes for the keyboard in the lab
- These codes are in Hexadecimal. The UART will be using the hexadecimal values.



PS/2 Keyboard Scan Codes

GRADING SCHEME AND REPORT REQUIREMENTS

- This task is 20 points :
  - Attempting the task: 5 points
  - Showing working boards to TA : 5 points
  - Task Report : 10 points (Maximum)

- Cover page:
  - Your name
  - Course Title, Task Number
- Project Task
  - VHDL Module and Test Bench
  - Simulation Waveform
  - UCF
- Summary paragraph
  - Work completed
  - Any problems
  - Helpful Hints
  - Suggested Improvements