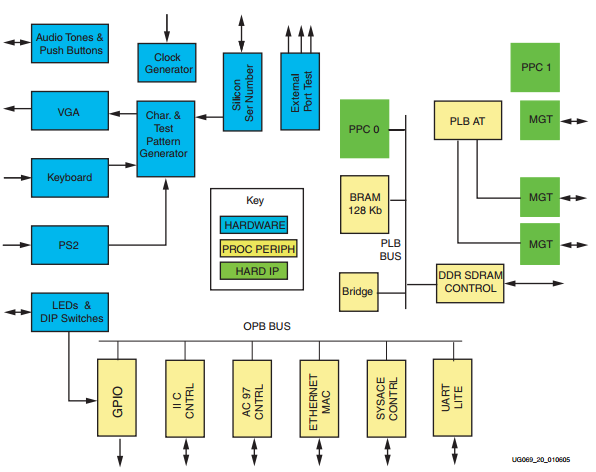
***Implementation of Low Pass Filter on The Given Image***

**Introduction**

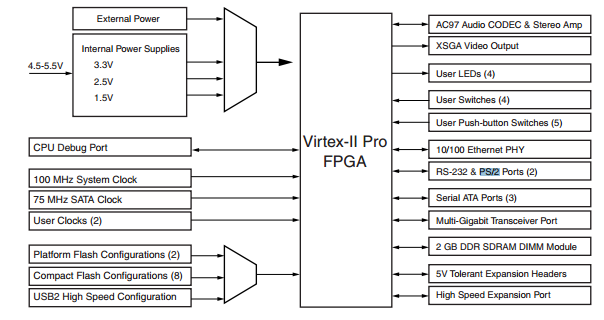
A Xilinx Virtex II Pro FPGA Board with a XC2VP30 device and 896 package has been used. The board includes a 15 pin video DAC connector to support the VGA monitor and a 6 pin PS/2 serial port to support the keyboard.

***Objective:*** The main objective is to learn to use the FPGA to display a grayscale image in raw format on the VGA with the help of storing in .coe file. Further, we learn how to implement a LBF on the given image, pressing the “0” key should pass the image through our LPF but should also retain the output so that when pressed the ‘o’ key again, it gets fed as the input to the LPF again. The character should be displayed in the top left corner of the VGA monitor.

***Background:***



**Fig. 1**: *Block Diagram of Virtex II Pro Development Board* ***[1]***

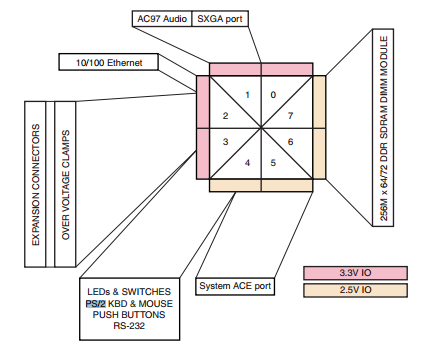


**Fig. 2**: *Block Diagram of Virtex II Pro Development Board* ***[1]***

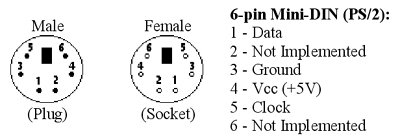
These are the onboard hardware that we are interested in for this assignment:

* PS/2 Connector
* Clock Generator

**PS/2 Connector:**



**Fig. 3***: I/O Bank Connections to Peripheral Devices showing LEDs and PS/2 on #4* ***[1]***

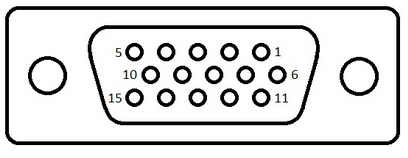


**Fig. 4***: Pins on PS/2 connector* ***[2]***

* The development kit has three serial ports on board namely:
* RS-232 ….. x 1
* PS/2 ….. x 2

RS-232 is a 9pin connector used to communicate to a host computer via COM port. There are also two PS/2 ports mainly to support the keyboard and mouse use. System. All of the serial ports are equipped with level-shifting circuits, because the Virtex-II Pro FPGAs cannot interface directly to the voltage levels required by RS-232 or PS/2. [1]

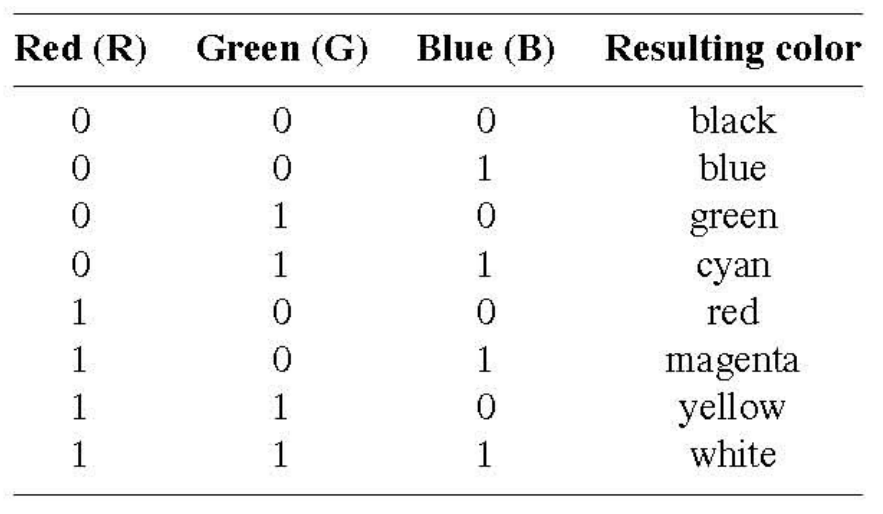
**VGA Connector:**



**Fig. 5**: *VGA Female connector* ***[3]***

* *Blank signal*: Overrides the RGB signal and blanks the display output.
* *Pixel clock*: Various clock rates can be created by the software but in this particular case, we use it at a rate of 25MHz since here we are using a 640x480@60Hz monitor.
* *RGB*: These are the three different analog voltage signals that decide the color of each

Pixel. The three colors are Red, Green and Blue.

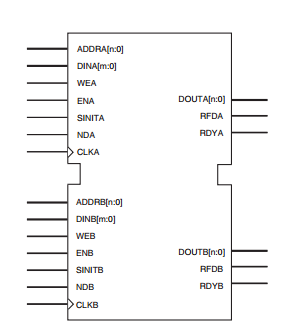


**Table 1**: *RGB Signals for 7 different colors* ***[4]***

**COE file**

COE files are memory files with extension .coe created with the help of Memory Editor - a tool that specifies memory contents and initialization values for CORE Generator memory cores. These files have one or more memory blocks defining the contents of one or more COE files. For each memory block defined in a CGF file, the Memory Editor generates a separate COE file. [5]

**Dual Port RAM**



**Fig. 6**: *Core schematic of Dual Port RAM* ***[4]***

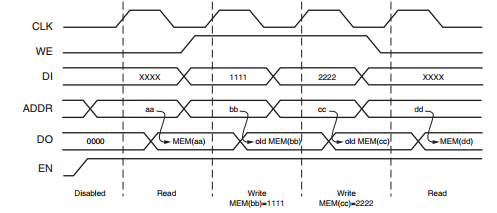
In this assignment we have used 2 Dual port RAMs, one is used to read the pixel values for display and the other is used to manipulate those values while the LPF is being implemented, once this is done, we copy these values to the first Ram so that we can read the updated image. The original image is not being retained here but we can always use another rom to retain the original image if needed.

The block memory operates synchronously to the rising edge of the clock. The enable pin affects the read, write, and SINIT functionality of the port. Ports with an inactive enable pin keep the output pins in the previous state and do not write data to the memory locations. By default, the enable pin is active high. [4]

Activating the write enable pin allows the port to write to the memory locations. When active, the contents of the DIN bus is written to memory at the address pointed to by the ADDR bus. The output latches are loaded or not loaded according to the write configuration (write first, read first, no change). When inactive, a read operation occurs, and the contents of the memory locations referenced by the address bus reflect on the DOUT bus, regardless of the write mode selected. [4]

In latch mode, the read operation uses one clock edge. The read address is registered on the read port, and the stored data is loaded into the output latches after the RAM access time. When using the output register, the read operation takes one extra latency cycle. [5]

A write operation is a single clock-edge operation. The write address is registered on the write port, and the data input is stored in memory. [5]

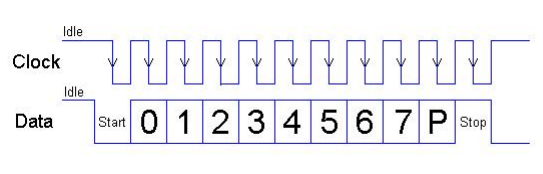


**Fig. 7**: *Read first mode wave form of Dual Port RAM* ***[5]***

**Procedure**

This section of the report provides a brief description of the steps that were followed to implement the assignment. Xilinx ISE 9.2i design software was used for Verilog code write-up, synthesize and implementation and to burn the code to the FPGA kit. We use some fragments of the codes from assignments 2,3 and 4 to synchronize the keyboard and VGA. Also, the image xco is provided by Dr. Lin.

1. Created a new project in ISE named ‘Lab5’. The logic implemented to write the code has been given below.
2. The code is been divided into 4 modules. One to synchronize the keyboard, the output of which is fed as the input to the module that is been used to synchronize the VGA monitor with FPGA development kit. We also use a module to handle the coe file, since we are using 2 RAMS, we’d have 2 modules here and lastly a top module is used to handle the signal flow within the rest of the modules.
3. *Types of signals used for Keyboard:* There are two signals involved in this procedure, one is the clock signal and the other is the data signal which is received from the keyboard. This signal is sampled at the negative clock i.e. at the falling edge.



**Fig. 7.a***: Nature of Data and clock signal* ***[4]***

1. *Reading the font Image:* The used rom has width of 8 bits and a depth of 65535 which corresponds to a 256x256 image in grayscale. This is created using the Xilinx coregent tools and the provided coe file. The clock signal and the address signal are given as inputs to the rom which in turn provides an 8 bit data which corresponds to the pixel value and based on this value we output the display color on VGA.
2. *Reading the keys:* The data signal from the keyboard is collection of 11 bits in which the main data is stored in the 8 bits starting from the 2nd bit and ending at the 9th bit as shown in the figure above. The rest of the bits indicate the start, end and parity. Each key has a unique code which is sent in the mention 8 bit data set. This data is sent continuously until the key is pressed followed by the ending bit 0.

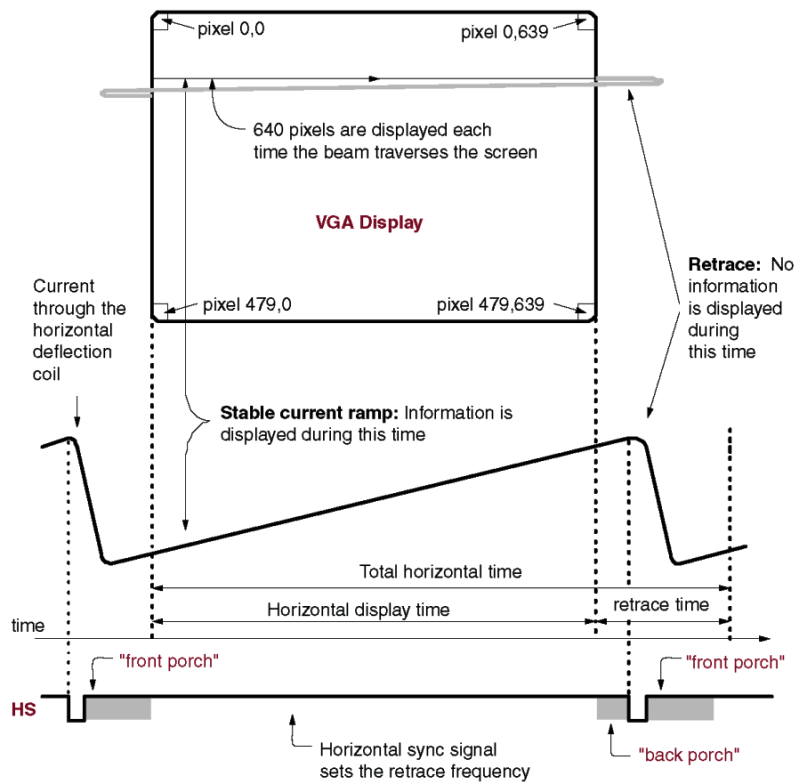
|  |  |  |
| --- | --- | --- |
| Key codes in Hexadecimal | Character  Address | KEY |
| 45  16  1E  26  25  2E  36  3D  3E  46  5A | 0000  0001  0010  0011  0100  0101  0110  0111  1000  1001  1011 | 0  1  2  3  4  5  6  7  8  9  Enter |

**Table 2**: *Keyboard scan codes [6]*

1. *Using the VGA monitor:* Here, the inputs are the two buttons, pixel clock signal and outputs are the synchronization signals and the signals for RGB.
2. We named the horizontal and vertical synchronization signals as h\_synch and v\_synch respectively, we also initialize counters for each synch signal in order to identify which pixel is being written. We start from h\_counter and v\_counter equal to zero, the h\_counter counts till the total period of 1 cycle as shown above and then is reset to zero and in the meantime v\_counter counts till the total period of 1 cycle of the vertical signal and then is reset to zero conditions are given to make the sync signals either high or low depending on which section of the cycle their respective counter is reached. The limit for each section of cycle for both horizontal and vertical signals where given by Dr. Lin in the handout for assignment 2 but since these were given in terms of seconds, we had to convert them to number of counts by multiplying the given time with pixel clock used. In my approach, a 25MHz pixel clock was used and the following counter values were implemented.

|  |  |  |  |
| --- | --- | --- | --- |
| Type of refresh cycle | Parameter | Time | count |
| HORIZONTAL | A | 31.77µs | 795 |
| B | 3.77 µs | 95 |
| C | 1.89 µs | 48 |
| D | 25.17 µs | 640 |
| E | 0.94 µs | 12 |
| VERTICAL | O | 16.6 ms | 523 |
| P | 64 µs | 2 |
| Q | 1.02 ms | 33 |
| R | 15.25 ms | 480 |
| S | 0.35 ms | 8 |

**Table 3**: *Count values obtained for the given time using a pixel clock of 25Mhz*



**Fig. 8:** *Diagram to give an idea of the logic implementation for VGA display* ***[7]***

Once the active video section of the cycle is reached, a case condition is used to display 8 different colors based on the output of the LEDs.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| LED Display | | | Decimal value | R | G | B |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 255 | 255 | 255 |
| 0 | 1 | 0 | 2 | 255 | 0 | 0 |
| 0 | 1 | 1 | 3 | 0 | 255 | 0 |
| 1 | 0 | 0 | 4 | 0 | 0 | 255 |
| 1 | 0 | 1 | 5 | 255 | 255 | 0 |
| 1 | 1 | 0 | 6 | 0 | 255 | 255 |
| 1 | 1 | 1 | 7 | 255 | 0 | 255 |

**Table 4**: *RGB Values based on the output of LEDs*

A white color background and a red color text was used in this assignment.

1. We have also used an FSM here, it is very much required to implement the wait states and to determine the right time to perform a task, we check if the key is being pressed at a given point of time to move to the next stage. We read pixels string from the left top corner and cover a 3x3 block area. Once done we add these pixel values and then average them, this number represents the pixel value for the center of that particular block. We gradually cover all such blocks on the image.
2. UCF*:* The UCF values were downloaded from [7] used.

|  |  |
| --- | --- |
| Signal | Value |
| Keyboard Clock | AG2 |
| Keyboard Data | AG1 |
| VGA clock | AJ15 |
| Horizontal synch | B8 |
| Vertical synch | D11 |
| blank | A8 |
| Comp synch | G12 |
| Pixel clock | H12 |
| red[7] | H10 |
| red[6] | C7 |
| red[5] | D7 |
| red[4] | F10 |
| red[3] | F9 |
| red[2] | G9 |
| red[1] | H9 |
| red[0] | G8 |
| green[7] | E11 |
| green[6] | G11 |
| green[5] | H11 |
| green[4] | C8 |
| green[3] | D8 |
| green[2] | D10 |
| green[1] | E10 |
| green[0] | G10 |
| blue[7] | E14 |
| blue[6] | D14 |
| blue[5] | D13 |
| blue[4] | C13 |
| blue[3] | J15 |
| blue[2] | H15 |
| blue[1] | E15 |
| blue[0] | D15 |

**Table 5**: *UCF Values [7]*

1. Next, we check the syntax form design utilities 🡪 check syntax option in the processes window.
2. We then synthesize the written code. If no errors occur, we proceed to the next step. Else, we need to debug the code accordingly.
3. Once the synthesis is finished successfully, we need to implement design.
4. Generate the programing file. The tools for these three steps can be found on the processes window.
5. A bit file is generated in the project folder. With the same name as the name of the project.
6. We now connect the FPGA kit to the system and the keyboard & VGA to the FPGA kit and turn the kit and ON.
7. We use the generated bit file and map it using configure device (iMPACT) tool available in generate programming file tool.
8. The board is now ready for testing.

***Difficult part:***

The most difficult part of the assignment for me was to implement the FSM, at first I was able to display the image but accumulation of all the neighboring pixel values was much of a task. Moreover, understanding how many stages the FSM actually requires was even more difficult. Apart from implementing the wait state, I gradually created more stages to implement the read copy, sum division, write and transfers between the RAMs in each stage.

**Results**

The results were obtained as expected. The given image was first displayed and then was sent to the LPF when the key “0” was pressed on the keyboard. The output of the LPF was displayed on the VGA monitor every time the key ‘0’ was pressed without retaining the original image.

**Conclusion**

The obtained result supports the desired objective. The ‘0’ key worked as mentioned above and the image was displayed on the left top corner of the VGA monitor. The Low Pass Filter was also working fine.

**References**

1. <https://www.digilentinc.com/Data/Products/XUPV2P/XUPV2P_User_Guide.pdf>
2. <http://www.computer-engineering.org/ps2protocol/>
3. <http://www.computer-engineering.org/ps2protocol/>
4. <http://www.xilinx.com/support/documentation/ip_documentation/dp_block_mem.pdf>
5. <http://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf>
6. <http://www.eecs.ucf.edu/~mingjie/EEL5722/EEL5722c%20lab3%20Tutorial.pdf>
7. <http://www.xilinx.com/itp/xilinx10/isehelp/cgn_r_memory_editor_overview.htm>
8. <https://eewiki.net/pages/viewpage.action?pageId=28278929>
9. <http://www.digilentinc.com/Products/Detail.cfm?NavTop=2&NavSub=453&Prod=XUPV2P&CFID=18922620&CFTOKEN=ff23df9a79e4c2ac-061E4AC0-5056-0201-02D0AEB908347149>
10. <http://ece.gmu.edu/coursewebpages/ECE/ECE448/S13/viewgraphs/ECE448_lecture7_VGA_1.pdf>
11. <http://www.cs.ucr.edu/~jtarango/cs122a_lab4.html>
12. Pong P. Chu, “FPGA PROTOTYPING BY VERILOG EXAMPLES”, WILEY PUBLICATIONS, 2008, pp. 341–352.