Algorithmic State Machine and Memory

In this lab, we will move forward one step with our keyboard/monitor combo but with algorithmic state machine (ASM) design approach and the utilization of the memory megafunctions. The FLEX10K70 has 36Kbits of SRAM in the EABs (as you have seen in the floorplan in last lab). These EAB can be utilized using the Altera’s magefunction.

Exercise (The Video RAM)

In lab3, the pattern display on the monitor was generated in real-time, i.e., the R/G/B signals are generated by monitoring the display timing signals (H_sync and V_sync). The common approach, however, is to use a video RAM such that a location in the video RAM is mapped to a pixel on the monitor. In our case, we need three bits to represent one pixel (and thus 8 colors depth) for the VGA display size of 640X480. This means we need 307,200X3 bits of SRAM. Since we only have 36Kbits of SRAM (you can see this from the compilation report) we must reduce the resolution. Thus, in the exercise, each memory location is actually been mapped to an area of 32X32 pixels; or the display size of 20X15. The total memory usage is thus 300X3bits.

- Download the archived project from ftp://ftp.ele.uri.edu/outgoing/jcl/306/lab5.qar
- Start Quartus II and restore the project using Project->Restore Archived Project.

Examine the VHDL source file called vgamem.vhd.

Inside the VHDL file called vgamem.vhd, there is an entity declared called myram. This entity has 9 I/O ports declared. The “we” port is a write enabling port for writing to memory. The “clock” port tells the module when to commit the data on “redin”, “bluein”, and “greenin” ports to the address specified on the “address” port. The ports “redout”, “blueout”, and “greenout” supply the data at memory location specified by “address”.

To create a memory device, the on-chip memory units called EABs (embedded array blocks) are used. Interfacing an EAB is done through a megafunction, which is specific to Altera designs, but other development platforms provide similar devices. To find out this megafunction or others included with the software click on the menu Help->Megafunctions/LPM.

When declaring a megafunction instance, there are two sections: the GENERIC MAP, and PORT MAP. The GENERIC MAP allows for parameters of the megafunction to be set like in our case the data bus width, address bus width, unregistered or registered read and write access to name a few. As is explained in the Help section mentioned above, not all parameters are required and some have default values. The PORT MAP, as with any component seen in previous labs, specifies how the ports of the declared component map to the entity that declares an instance of them. As with the GENERIC MAP, not all of the elements of the megafunction require a map, and those parameters are noted in the Help section.

The megafunction called to create memory for this lab is lpm_ram_dq. This instance creates a ram of size $2^{10}$ entries (because the address bus width is 10bits) with each address containing 3 bits of data (because the data bus width is 3 bits). The data stored for this lab is a
bit for the red, blue, and green value for each block of our pseudo-screen. The pseudo-screen is an area of 20 wide by 15 high making each block 32 pixels square.

**Examine the VHDL source file called lab5.vhd.**

The section of interest is the PROCESS that is declared. This process utilizes a state machine the segment of code associated with the current state held in sthestate. Since this process is triggered by the edge of a 25MHz clock, the state machine is processed with that frequency and each state must not have an execution time or propagation delay longer than the clock period, else unwanted results could be obtained. As expected, the state machine is constructed of different states that allow for sequential execution of steps with conditional transitions.

The following flowchart showing the algorithm of which the process named “main” has implemented in the Lab5b architecture of Lab5 entity. The algorithm reads and recognizes the keyboard inputs and then responds accordingly. The main features to note:

1. The use of type declaration: one for the state signal and the other for the direction.
2. The “direct” translation from the algorithmic flowchart to the VHDL codes. Note in particular the state name marked in the flowchart.
3. The use of case-when inside the case-when to implement the multiple values checking in one step (one clock cycle).
4. The checking of condition “svertsync='0'” is to verify whether the display on the monitor is currently being updated. Check page 142 of Rapid prototyping book. After drawing each frame of 640X480 display, Vertical Sync signal goes to ‘0’ for 64µs. This is the time for the traditional CRT display to swing the scanning from the bottom right to the upper left corner. This state machine will operate only during this 64µs period. This will avoid the conflict with the “vgaout” since it also uses the video RAM: “vgamem”.
5. The checking of key input has been simplified to the checking of the last two bytes in the sequence of three bytes.
Start

- reset
  - =0
    - reset state to S1
    - reset X, Y pos.
    - reset sclock
  - =1
    - Initialize cursor
clocr, etc.

svertsync=1

sclock<=1

swe<=1

key= "F0"

key?

- no
  - s3
  - key=?
    - = "75" => s5
      - sdir<=up
      - sdir<=left
    - = "72" => s8
    - = "6B" => s8
    - = "74" => s8

svertsync=1

0 clean up;
ready for next key

s1

svertsync=1

sclock<=1

update X, Y based on sdir

ready to write to new position

write new position

s9

s5

Remove old position

svertsync=1

s6

s7

s8

s9

s1

s2

s3

s4

s1

s1

s1

s1

s1
Verify that the correct pins are assigned.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Pin Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>blue_out</td>
<td>Pin_238</td>
</tr>
<tr>
<td>clk25MHz</td>
<td>Pin_91</td>
</tr>
<tr>
<td>display1[0]</td>
<td>Pin_13</td>
</tr>
<tr>
<td>display1[1]</td>
<td>Pin_12</td>
</tr>
<tr>
<td>display1[2]</td>
<td>Pin_11</td>
</tr>
<tr>
<td>display1[3]</td>
<td>Pin_9</td>
</tr>
<tr>
<td>display1[4]</td>
<td>Pin_8</td>
</tr>
<tr>
<td>display1[5]</td>
<td>Pin_7</td>
</tr>
<tr>
<td>display1[6]</td>
<td>Pin_6</td>
</tr>
<tr>
<td>display2[0]</td>
<td>Pin_24</td>
</tr>
<tr>
<td>display2[1]</td>
<td>Pin_23</td>
</tr>
<tr>
<td>display2[2]</td>
<td>Pin_21</td>
</tr>
<tr>
<td>display2[3]</td>
<td>Pin_20</td>
</tr>
<tr>
<td>display2[4]</td>
<td>Pin_19</td>
</tr>
<tr>
<td>display2[5]</td>
<td>Pin_18</td>
</tr>
<tr>
<td>display2[6]</td>
<td>Pin_17</td>
</tr>
<tr>
<td>green_out</td>
<td>Pin_237</td>
</tr>
<tr>
<td>horiz_sync_out</td>
<td>Pin_240</td>
</tr>
<tr>
<td>keyboard_clk</td>
<td>Pin_30</td>
</tr>
<tr>
<td>keyboard_data</td>
<td>Pin_31</td>
</tr>
<tr>
<td>red_out</td>
<td>Pin_236</td>
</tr>
<tr>
<td>reset</td>
<td>Pin_41</td>
</tr>
<tr>
<td>vert_sync_out</td>
<td>Pin_239</td>
</tr>
</tbody>
</table>

Do a functional simulation, timing simulation, and then demonstrate the working results to the TA.

**Assignment**

**Part 1 (Mandatory)**
- Modify the exercise to produce a painting program.
- Utilize the keyboard to create an edit mode which is entered by hitting the “Enter” key. Once in the edit mode, you should be able to paint. When you are not in edit mode, the arrow keys will allow the user to position the cursor without painting. When the “Enter” key is hit while in edit mode, the edit mode is exited and painting is resumed.

**Part 2 (Bonus)**
- Add support for when the ‘c’ key is hit on the keyboard that the color when in paint mode is changed. The number of colors it can change to must be more than one.

**Report:**
No report is need for the exercise part. For the Assignment part: (1) algorithmic flowchart; (2) VHDL codes (including the components’ VHDL if you’ve modified them); (3) Draw a block diagram of the entire circuit. Clearly mark all the i/o names of a block and all the net (wire/connection) names. Also indicate the name and attribute (entity, component, process, etc.) of each block.