ELE306 Spring2008 LAB5

Algorithmic State Machines and Video RAM

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 18Kbits of SRAM in the nine EABs. These EAB can be utilized using the Altera's magefunction utility.

Exercise #1 (The Video RAM)

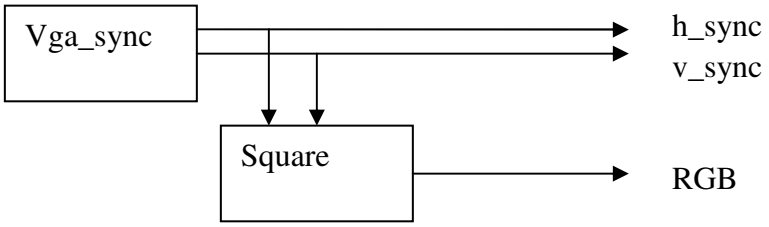


Figure 1. Paint-by-timing method used in Lab 3.

Figure 1 shows a simplified block diagram of Lab 3's approach, which is basically paint-by-timing. Essentially, the square.vhd circuit paints the square on the VGA display by outputting appropriate RGB signal at the appropriate time. Therefore, circuit "square" must constantly monitor the timing signals: horizontal sync and vertical sync. Modern day approach utilizes an organization in Figure 2.

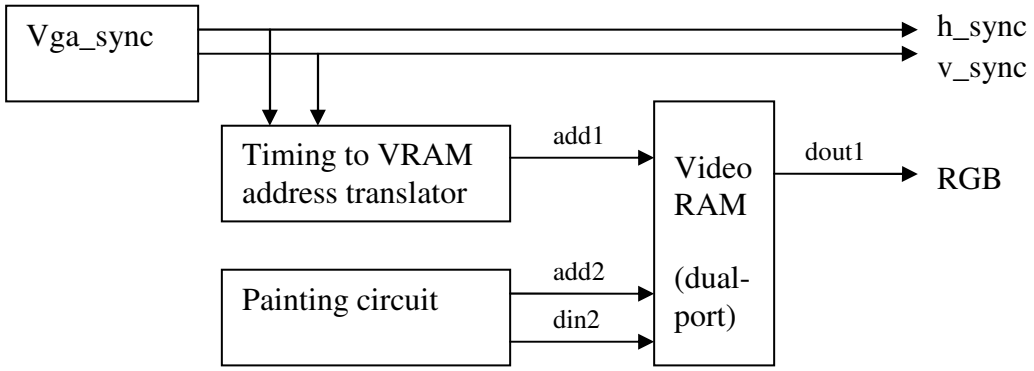


Figure 2. The use of Video RAM in display engine.

Figure 2 shows the common approach of using a video RAM (VRAM) such that a location in the VRAM is mapped to a pixel on the monitor. The VRAM has two ports (dual-port): port no. 1 is read-only and connected to add1 (address to port1) and dout1 (data out of port1); and port no. 2 is write-only and connected to add2 and din2. The timing signals, h_sync and v_sync are generated as before but the paint circuit is removed from directly checking the timing signals. Instead a circuit will translate the timing signals to the appropriate VRAM address. This will enable the contents of VRAM be read out one pixel at a time, via dout1, and be displayed on the

monitor. These circuits are completely autonomous and formed a "scanning engine" to effectively map VRAM contents to the monitor display. To paint anything on the screen, all we have to do now is to change the contents of the VRAM and the pattern(s) will show up faithfully on the screen! Port no. 2 of VRAM is to be access by the circuit that generates the pattern(s).

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 307,200 X 3 bits of SRAM. Since we only have 18Kbits 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. This is done by tweaking the timing to VRAM address translator. The effective display size in the exercise is 20X15. The VRAM size 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 called *vgamem*. 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 (the din2 as in Figure 2) to the address specified on the "address" port. The ports "redout", "blueout", and "greenout" (the dout1 as in Figure 2) supply the data at memory location specified by "address". In this exercise, we use a regular RAM to emulate the dual-port nature of the VRAM and therefore only one "address" input. The way the exercise circuit works is to multiplex between ad1 and ad2 (as in line 88 and 89 of lab5.vhd, where "spixel_row(9 downto 5) & spixel_column(9 downto 5)" is ad1 from the timing signals and "sposy & sposx" is ad2 from the painting circuit).

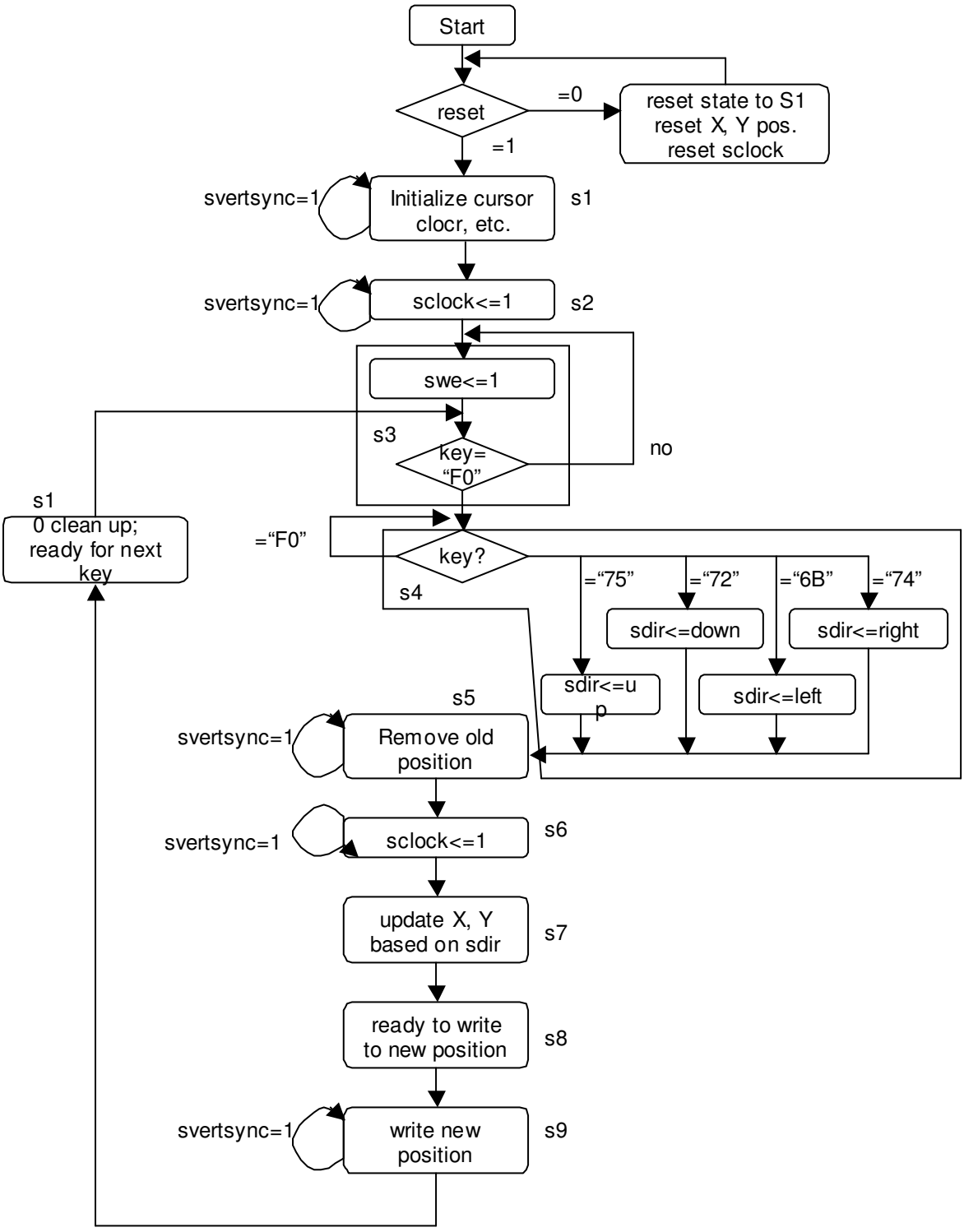
To create a memory device, the on-chip memory units called EABs (embedded array blocks) are used. Interfacing an EAB is done through a megafunction, which is specific to Altera designs, but other development platforms provide similar devices. To find out this megafunction or others included with the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

Open vgamem.vhd in the VHDL editor window. Notice on Lines 3 and 4, Altera's own library LPM is used. 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; at line 23, lpm_widthad => 10) with each address containing 3 bits of data (because the data bus width is 3 bits; at line 22, lpm_width => 3).

Examine the VHDL source file called lab5.vhd.

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 (the break code sequence) 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 above to the VHDL codes. Note in particular the state name marked in the flowchart. The design procedure is to create an algorithmic flowchart for the design and then chop it down to segments of

actions. Each segment must have either a clear defining border, such as S3 which must wait for the first byte "F0" in the break code sequence, or the actions within the state can finish within one clock cycle. The former is easier to determined while the later case usually requires some experience and/or trials-and-errors.

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 to change the contents of the VRAM. The "scanning engine" will have the VRAM for the rest of the time.
5. The checking of key input has been simplified to the checking of the last two bytes in the sequence of three bytes.

Verify that the correct pins are assigned.

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

Perform a functional simulation and then demonstrate the working results to the TA.

You may choose to work on assignment part 1 before moving on to exercise #2. Note that assignment part 1 is derived from exercise #1, while assignment part 2 is based on exercise #2.

Exercise #2 (Animation with Video RAM)

Download the archived project from ftp://ftp.ele.uri.edu/outgoing/jcl/306/lab5a.qar Restore the project and program the FPGA. Examine the VHDL code to discover how the animation was done. Essentially, a separate ROM is created to store the 8 different patterns in the animation. Each pattern is four pixels by four pixels and was placed in the ROM one row at a time for each pattern. To simplify this exercise, the keyboard has been removed.

Reconstruct the algorithmic flowchart, as shown in previous exercise, of lab5b.vhd (to be included in your lab report).

Assignments

Part 1

- Modify the exercise #1 to produce a single-color 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

- Put keyboard back to exercise #2 and modify it so you can use the arrow keys to move the 4X4 block of the animation around the screen; while the animation is still going.

Lab 5 Reports:

- No report is need for the exercise #1.
- For exercise #2, draw the algorithmic flowchart of lab5b.vhd (as written).
- For assignment part 1 and part 2, respectively:
  1. algorithmic flowchart;
  2. VHDL codes (including the components' VHDL if you've modified them);
  3. Draw a block diagram of the entire circuit. (Use RTL viewer produced diagram as a guide and draw your own more simplified version. RTL viewer may produce a huge schematic in this case.) 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.