Chapter 10

VGA Text Generator

The purpose of this laboratory is to create a character generator circuit for displaying ASCII text characters on the VGA display. You will also create a simple circuit that writes characters to the screen using the switches and buttons.

Learning Objectives

After completing this laboratory you should be able to:

- Understand how to use internal BRAM memories
- Draw characters from a font ROM onto the screen

Exercises

Exercise 1 – Character Organization Overview

Displaying text is an important function of a video controller. Dedicated circuits are often used to facilitate the display of text characters on a screen. To display text on our VGA display, we will organize the 640x480 display area into larger "tiles" where each tile represents a character location. In this lab, the size of each character is 8 pixels wide and 16 pixels high. Mapped onto a 640x480 display, this font will display 80 text characters in each line (i.e., 640 pixels for each row divided by 8 columns per character) and 30 lines (i.e., 480 lines divided by 16 pixels per character, 480 / 16). The layout of the 80 x 30 character screen is shown in Figure 10.1.

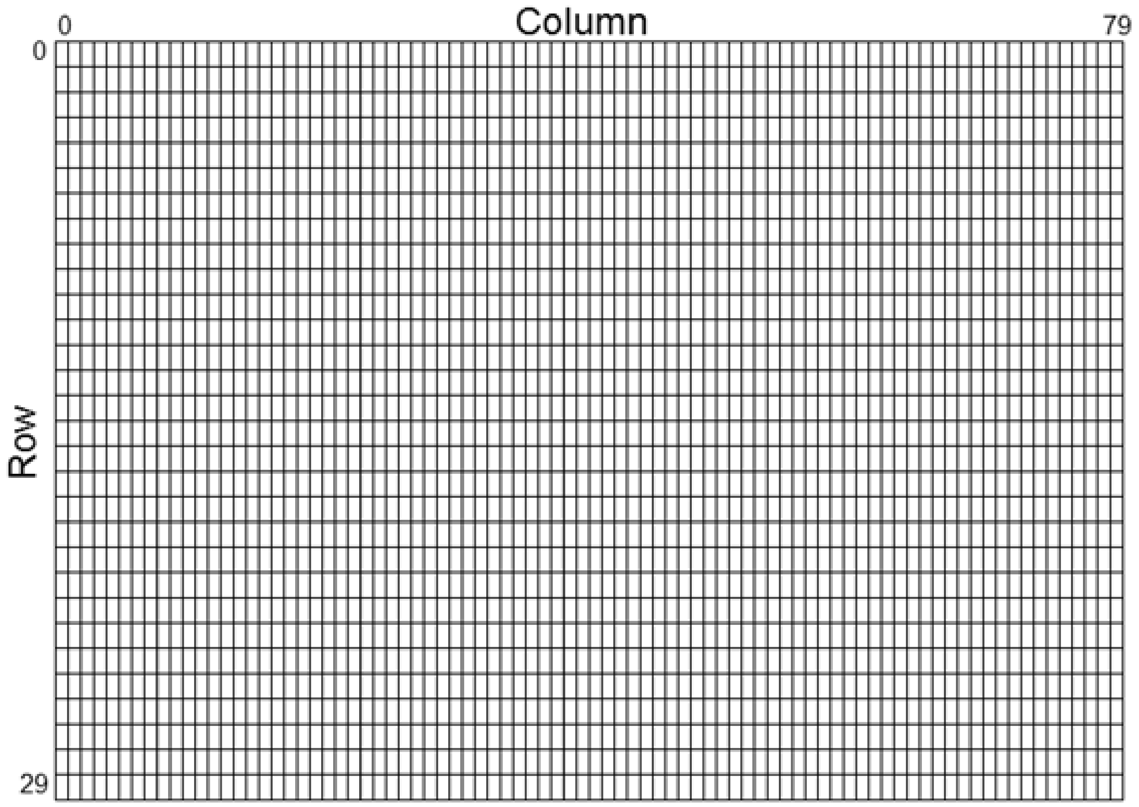

Figure 10.1: Character screen layout.

When drawing text characters on the screen, we will refer to several different coordinate systems. The first coordinate system is the "pixel" coordinate system used for locating individual pixels on the display (i.e., 640x480). The top left pixel of the display is labeled (0,0), and the lower right pixel of the display is (639,479). The pixel_x and pixel_y outputs of the VGA controller reference this display coordinate system.

There is another coordinate system, called the "character" coordinate system, for specifying a character position on the display (i.e., the 80x30 character layout shown in the previous figure). The top left character position is (0,0), and the lower right character is (79,29). A set of 8x16 pixels (i.e., 128 pixels) in the pixel coordinate system is associated with each character in the character coordinate system. For example, the character located at the character position (7,14) is made up of 128 pixels where the top left pixel of this character location is (56,224) (i.e., 7x8=56,15x16=224) and the lower left pixel is (63,239).

Every pixel in the display is mapped to one of the characters. To convert the x coordinate of a pixel in the pixel coordinate system into its corresponding x coordinate in the character coordinate system is done by dividing the x pixel value by 8 without rounding. To convert the y coordinate of a pixel into the corresponding character row is done by dividing the y pixel value by 16 (again without rounding). For example, the pixel (138, 312) in the pixel coordinate system will map to the character located at character position (17, 19) (i.e., 138/8, 312/16). The location of this character in the coordinate system is shown in Figure 10.2.

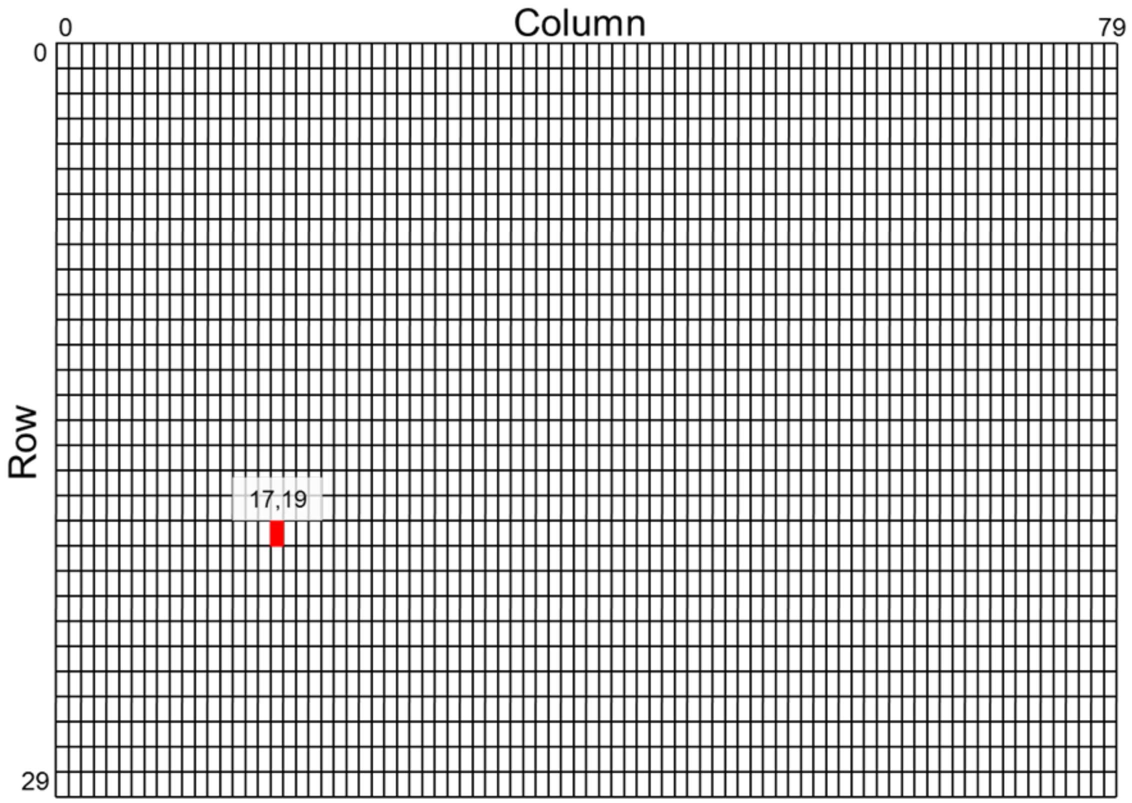

Figure 10.2: Character (17, 19).

This division can easily be done in digital hardware by taking the top seven bits of pixel_x for the x character location taking the top 6 bits of pixel_y for the y character location. The following VHDL code demonstrates how to generate the character positions from the pixel positions:

```
char_x_pos <= pixel_x(9 downto 3);
char_y_pos <= pixel_y(8 downto 4);
```

There is also a coordinate system for pixels within an individual character. As described above, each character is 8 pixels wide and 16 pixels high. The position of the top left pixel of a character is (0,0), and the lower right pixel of a character is (7,15) as shown in Figure 10.3.

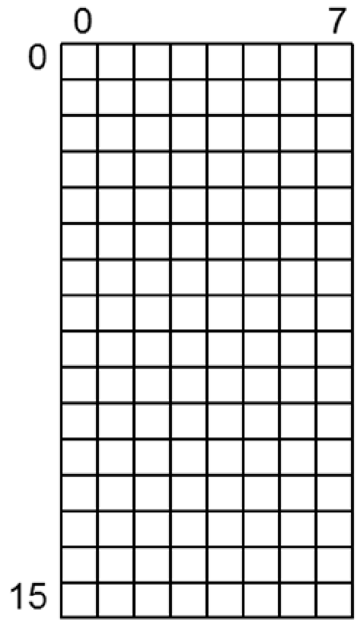

Figure 10.3: Character coordinate system.

Every pixel in the display coordinate system maps to a specific character in the character coordinate system *and* to a pixel within this character. It will be important to map pixel locations in the display coordinate system to pixels within an individual character. The x pixel location within a character can be found by computing the pixel_x mod 8. The y pixel location within a character can be found by computing pixel_y mod 16. Performing a modulus operation using powers of 2 is very simple in a digital system. This is done by simply selecting the lower three bits of the pixel_x value and the lower four bits of the pixel_y value as shown in the following VHDL code:

```
char_x_pixel <= pixel_x(2 downto 0);
char_y_pixel <= pixel_y(3 downto 0);
```

In the example described above, the pixel (138,312) in the display coordinates maps to character (17,19). Within this character, this pixel maps to (138 mod 8=2, 312 mod 16=4) as shown Figure 10.4.

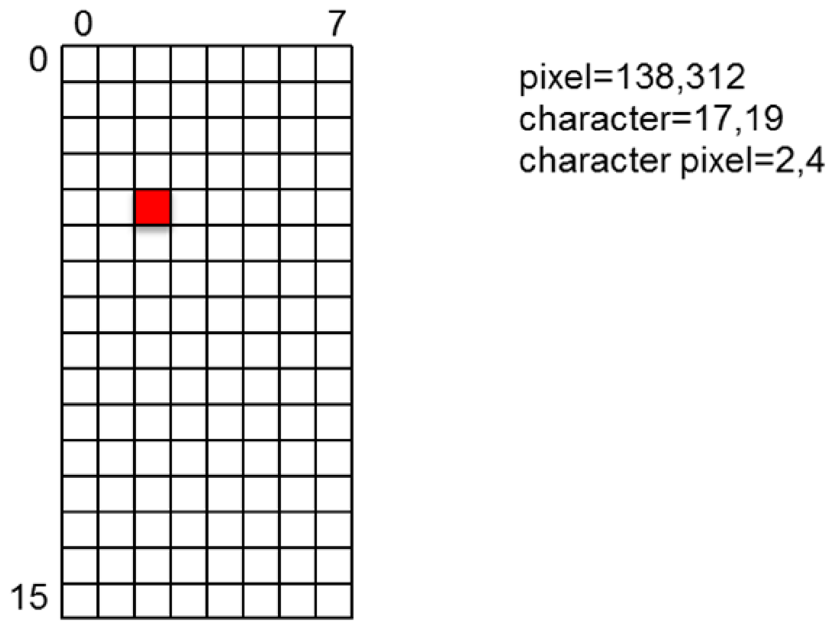

Figure 10.4: Pixel (138, 312) in character coordinate system.

Answer the following questions to demonstrate your understanding of the organization of the characters on the display:

Question: How many characters can be displayed on the 640 x 480 display using this font?

Question: What are the character coordinates of the character associated with the pixel at location (279, 173)?

Question: What is the position of this pixel within a character (i.e., what position is this pixel within a 8x16 character)?

Exercise 2 – Character Memory

The primary task of this lab will be to create a character generator module for display text characters on the VGA display. This module can be reused in future labs for displaying text on your VGA display. An important function of this module is to remember the character value at each character position on the display. A "character memory" will be used to store the character to display at each of the 80x30 locations on the character display.

The character to display at each location is specified by a 7-bit ASCII value. To support any 7-bit ASCII characterl, the minimum size of this memory is 80x30x7 bits to provide enough room to store characters (7 bits each) for each of the 80 columns and 30 rows. This memory size, however, is not an even power of two. To simplify our circuit, we will use a character memory that is sized with even powers of two. Specifically, the memory size will be 128x32x8 bits. This memory can provide enough memory for 128 columns, 32 rows, and 8 bits for each character. Seven bits are needed to determine the character column (i.e., $2^7 = 128$) and five bits are needed to determine the character row (i.e., $2^5 = 32$). Although this larger sized memory will result in wasted memory, it will significantly simplify the circuit you will create.

A character memory specified in VHDL has been provided for you in this lab. Download this memory and review its structure. The entity declaration for this memory is shown below.

```
entity char_mem is
   port(
      clk: in std_logic;
      char_read_addr : in std_logic_vector(11 downto 0);
      char_write_addr: in std_logic_vector(11 downto 0);
      char_we : in std_logic;
      char_write_value : in std_logic_vector(7 downto 0);
      char_read_value : out std_logic_vector(7 downto 0)
   );
end char_mem;
```

Download: char_mem.vhd

The character memory is organized in row order. This means that sequential characters in memory are located on the same row next to each other. The address 0x000 corresponds to the character at the location 0,0, the address 0x001 corresponds to the character at location 1,0, and the address 0x04F corresponds to the character at location 79,0 (the last visible character in row 0). Because the memory is organized with 128 character columns per row, there are an additional 48 characters in this row that are not visible (i.e., addresses 0x50 through 0x7F). The first character of the next row (i.e., the character at position 0,1) is stored at address 0x080. The character address can easily be created from the character column and character row by concatenating the column signal with the row signal as follows:

```
char_write_addr <= char_row & char_column;
```

This memory has two memory ports: a read port and a write port. The write port is used to store character values at specific locations in the character coordinate system. You will use this port to update and modify the character display. When you want to display a different character at a different location on the screen, you will write a character into the character memory at the desired location. Writing characters to the character memory will occur relatively infrequently. For this lab, you will use the switches and the buttons to write characters into the character memory (this will be described in a future exercise).

There are three signals associated with the write port: the character write address (char_write_addr), the character value to write (char_write_value), and the write enable (char_we). To store a new character in the display, you will need to set the value of the character you wish to write, determine the character address, and assert the character write enable control signal.

The read port is used by the character generator circuit to determine the character to display at a particular pixel_x and pixel_y location. There are two signals associated with the character memory read port. The first is the character read address (char_read_addr). This address is a 12 bit signal used to determine which location in the character memory to read. The second signal is the char_read_value. The character value at the location specified by the character read address would be provided on this signal. As shown in the VHDL for this memory, the character data will be provided on the clock cycle *after* the address is given (i.e., a synchronous memory).

Carefully review the contents of the character memory VHDL file and answer the following questions:

Question: How many bits are stored in the character memory?

Question: What address of the character memory holds the value for the character located on the character display at (65, 17)?

Question: What are the coordinates of the character stored at address 0xB8F in the character memory?

Question: What is the default message that will be displayed on line 0? You will need to interpret the default ASCII values for line 0 to determine the text output.

Question: Summarize the timing of a read operation.

The character that is read from the character memory will depend on the current pixel_x and pixel_y values that are generated by the VGA timing circuit. As shown in the Figure 10.5 below, you will need to generate an address for the char_read_addr. Determine how you are going to generate the 12-bit address signal that will drive the char_read_addr input of the character memory. This statement should be created by concatenating various bits from the pixel_x and pixel_y counters.

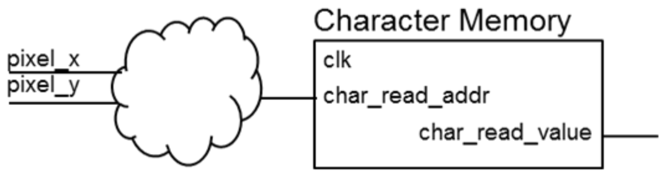

Figure 10.5: Character Memory.

Question: Provide a VHDL statement that will determine the char_read_addr input based on the pixel_x and pixel_y signals (i.e., char_read_addr <= [fill in the blank using pixel_x and pixel_y]).

You will use this statement later in the lab when you build your character generator circuit.

Exercise 3 – Font ROM

The appearance of the characters on the screen is determined by a "font ROM". The font ROM is a read only memory that contains the pattern of pixels that should be displayed on the screen when a particular character needs to be displayed. As described earlier, the size of each character in this font ROM is 8 pixels wide and 16 pixels high. The font ROM stores one bit for each of the 8x16 pixels associated with each character for a total of 128 bits. A '1' in the font ROM indicates the corresponding position within the 8x16 character should be displayed in the foreground (i.e., white for white text). A '0' in the font ROM indicates that the corresponding pixel should be blanked or put in the background. The text below demonstrates the contents of the font ROM for the upper-case character 'A'. The layout of this character in a 8x16 character grid is shown in Figure 10.6.

```
"00000000", -- 0
"00000000", -- 1
"00010000", -- 2    *
"00111000", -- 3   ***
"01101100", -- 4  ** **
"11000110", -- 5 **   **
"11000110", -- 6 **   **
"11111110", -- 7 *******
"11000110", -- 8 **   **
"11000110", -- 9 **   **
"11000110", -- a **   **
"11000110", -- b **   **
"00000000", -- c
"00000000", -- d
"00000000", -- e
"00000000", -- f
```

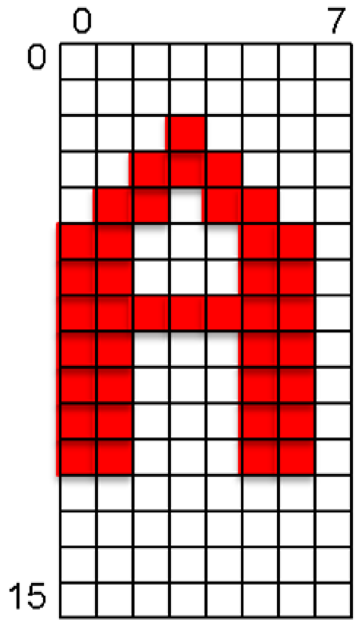

Figure 10.6: The letter A.

A font ROM has been created for you that defines the character shape for 128 different characters (all 7-bit ASCII characters). This font ROM is defined as a VHDL file that declares an array of std_logic_vector(7 downto 0) values. Each word of the ROM defines one line of a character. Since each character requires 16 lines, the total size of the ROM is 128 characters x 16 lines / character x 8 bits per character line. This ROM fits into the size of a single Xilinx BRAM and will be mapped to one of the BRAMs on your FPGA board. The initial contents of the ROM have been defined and you can view the contents of the ROM by reviewing the VHDL file.

Download: list_ch13_01_font_rom.vhd

Open this file and review its contents (you will need to add it within your project). Answer the following questions after reviewing the VHDL of this ROM:

Question: How wide, in bits, is each word of this ROM?

Question: How many words are there in this ROM?

Question: What is the total size, in bits, of this ROM?

Question: What is the value of the word at address 0x32a in this ROM?

The font ROM will be used in conjunction with the character memory to determine which pixels to highlight in order to display characters. You will need the character from the character memory and the pixel_y signal from the VGA timing controller to determine the address in the font ROM. The character value (which is 7 bits) is used to determine the starting address of the character to display. The character value should form the upper seven bits of your address when reading the font ROM. The pixel_y value is used to determine which of the 16 lines of this character to display. The pixel_y value should form the lower 4 bits of your address when reading the font ROM.

Question: What character is associated with address 0x32a in this ROM?

Question: What addresses in the font ROM store the contents for the character 'A'?

Question: Provide a VHDL statement that will determine the address input to the font ROM. Your statement should use the character read from the character memory AND the pixel_y signal. (i.e., font_rom_addr <= <fill in the blank>).

Exercise 4 – Character Generator

Once you understand the function of the character memory and the font ROM, you are ready to build the character generator. The character generator module can be used with any VGA based project to display text characters. Begin this exercise by creating a new VHDL entity named `charGen.vhd` with the following ports:

Port Name	Direction	Width	Purpose
clk	Input	1	50 MHz clock
char_we	Input	1	Character write enable. When this signal is asserted, a new character is written into the character memory.
char_value	Input	8	The 8-bit value to write into the character memory.
char_addr	Input	12	The write address of the character memory.
pixel_x	Input	10	The column address of the current pixel.
pixel_y	Input	10	The row address of the current pixel.
pixel_out	Output	1	The value of the character output pixel. A logic ‘1’ indicates that the current pixel should be displayed in the foreground and a logic ‘0’ indicates that the pixel should be blanked or put in the background.

There are three steps involved to determine the correct pixel out value based on the pixel_x and pixel_y display location. These steps are as follows:

1. Determine the current character at the character location specified by the pixel_x and pixel_y locations. This is done by properly addressing the character memory (described above) and reading the character at this memory location.
2. After obtaining the current character, the next step is to read a single line of the character font from the font ROM. The current character read from the character memory and the pixel_y value are used to determine which character line in the font ROM to read.
3. The final step is to determine which of the 8 pixels in the font ROM line to display. You will use the bottom three bits of the pixel_x signal to determine which of these 8 pixels to display. The relationship between the character memory, font ROM, and the 8:1 multiplexer is shown in Figure 10.7.

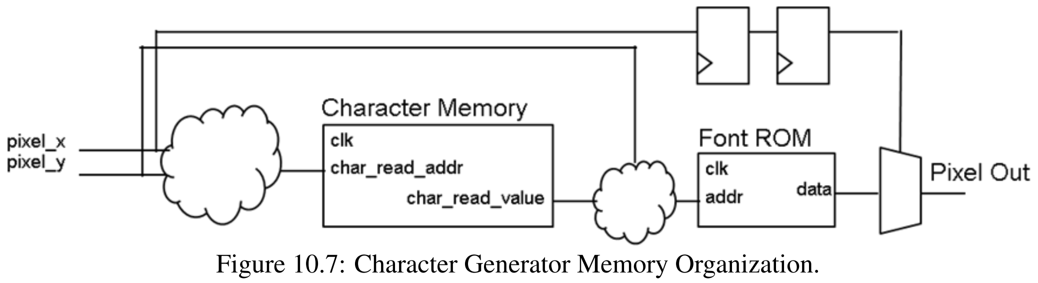

Figure 10.7: Character Generator Memory Organization.

Create your character generator circuit by instancing the character memory, the font ROM, and a pixel out multiplexer. Determine the logic for generating the character memory read address, the font ROM address, and the pixel_out select line.

Both the character memory and the font ROM are synchronous memories and require one clock cycle to access the memory contents. Since there are two memories in series, your pixel output will arrive two clock cycles after the input. By the time the value of the pixel is determined, the current value of pixel_x will have changed. The table below describes the sequence required to determine the appropriate pixel value:

A problem occurs during the last step of the process when selecting which of the 8 bits in the font ROM row to display. Since the pixel_x value changes every other clock cycle, the incorrect pixel_x value is used for the input to the 8:1 pixel multiplexer. To resolve this problem, you need to provide two pipelining registers for the pixel_x

Cycle	Pixel	Action
n	x, y	Initiate character memory read
n + 1	x, y	Character memory result is available. Start font ROM read
n + 2	x + 1, y	Font ROM result is available. Choose the appropriate bit from the font ROM

input for use by the 8:1 multiplexer. This two cycle delay will guarantee that the pixel_x address used to compute the character memory read address is the same pixel_x used for the mux input. The timing diagram shown in Figure 10.8 demonstrates the timing of this three stage process. Carefully review this timing diagram to understand how the pixel out signal is generated.

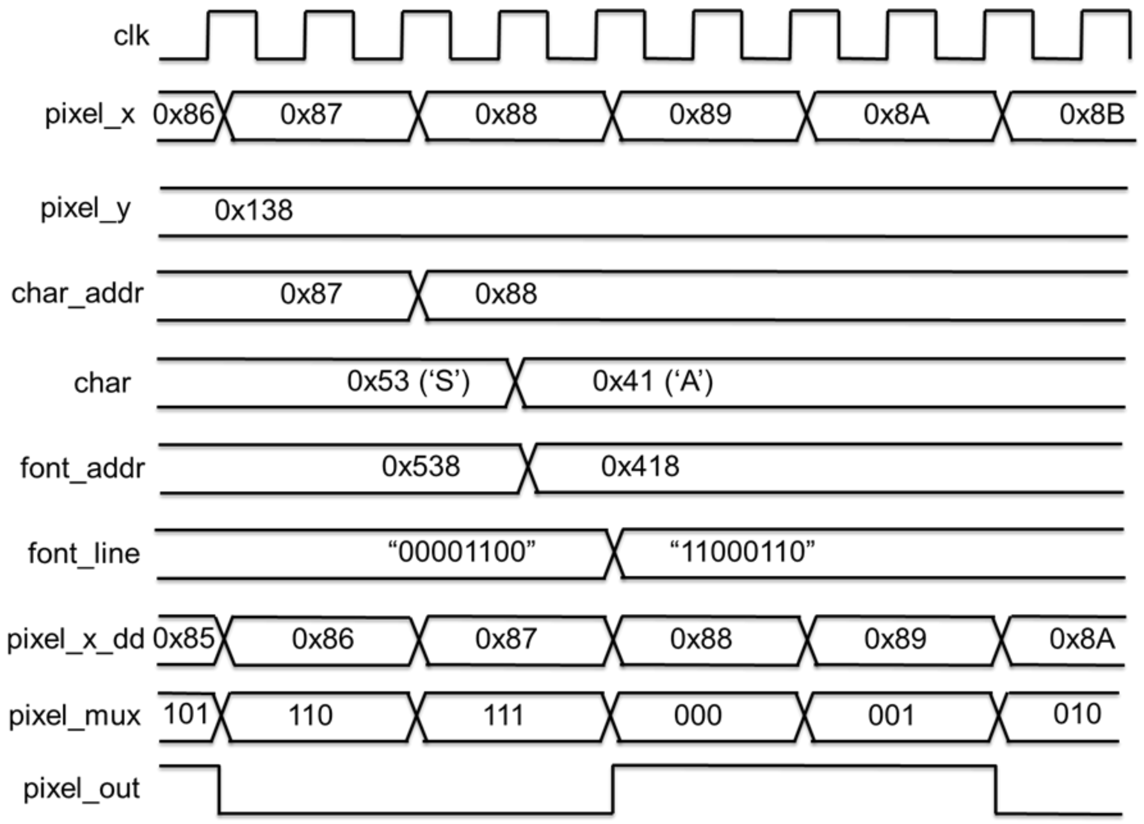

Figure 10.8: Character generator waveform.

A simple testbench has been provided for you to test your character generator. This testbench will emulate the VGA timing controller and see if your character generator circuit produces the correct output. The testbench is organized into three phases:

1. The character 0x0 is written to all of the locations in the character memory to put the memory in a known state (i.e., clear the contents of the character memory).
2. Several characters are written to the character memory into various locations
3. A complete VGA cycle is simulated and the output of your character generator is compared with the expected output

Download: tb_chargen.vhd

Carefully simulate your character generator within this testbench until you received the “done” message. This testbench will give you an error if the pixel out signal does not match the expected pixel out signal. Resolve any errors you may find in your circuit before proceeding.

Upload: Upload your character generator VHDL file after it passes all testbench cases.

Exercise 5 – Top-Level Design

The final design exercise is to integrate your VGA character controller and the VGA timing controller to create a rudimentary terminal. Your terminal will accept characters from the buttons and display them on the screen.

Follow these guidelines when creating your top-level design:

- Use button 3 as a reset button—all circuit modules should be reset when this button is pressed
- Instance your VGA timing controller circuit from a previous lab. Note that due to the delays associated with the memories in the character generator, the pixel output will be delayed by two clock cycles in comparison with the HS and VS signals. You should add two flip flops between the HS and VS outputs of the VGA timing controller and the HS and VS output signals of your top-level circuit. This will align the output pixel data with the HS and VS signals
- Instance your character generator from this lab and hook it up to the VGA timing controller (i.e., attach the pixel_x and pixel_y outputs of the VGA timing controller to the character generator)
- Attaches the switches to the char_value input of the character generator. The switches will determine the value to write into your character memory
- Instance your seven segment display and drive the right two digits of the segment with the value of the switches (blank the left two digits). This will allow you to look at the hex value of the switches before you send your character to the display
- You will need to generate the RGB signals from the pixel_out signal. The character ROM only indicates foreground vs. background (i.e., the result is only a single bit and does not store color). You will want to choose a foreground color and background color and convert the value of the pixel into your chosen foreground and background colors (i.e., if you want the foreground to be white, you will assign RGB = “11111111” when the pixel_out = ‘1’ and if you want the background to be black, you will assign RGB = “00000000” when the pixel_out = ‘0’). Make sure to blank the RGB signals when the VGA timing controller asserts the blank output
- When button 0 is pressed, write the value of the switches into the character memory (the memory location to write to will be described in the next bullet). You will need to add a delay to this button so that you are not sending many characters (similar to the delay used in the VGA display lab)
- Create a register that indicates the character column and character row where a new character should be written. These registers should be updated every time a character is written so that characters one after another, from left to right. For example, the first character location is character 0,0. After writing a character to this location, increment the column register to 1 so the next character will be written at 0,1. After writing to location 0,79, move to the start of the next row (i.e., 1, 0)

Simulate the writing of values to the character memory and verify that your column and row counters increment properly. Once your circuit has simulated properly, create a UCF file that incorporates the I/O signals used in your project and synthesize your circuit.

Question: Review you synthesis logs and copy all warnings. Summarize each warning and provide a justification for why the warning is acceptable.

After you have resolved all warnings and verified that your I/O pins were properly mapped, test your design on the board and VGA monitor.

Upload: Copy and paste your top-level design.

Personal Exploration

For your personal exploration, modify the top-level design to add some additional features. Ideas include:

- Add some objects on top of the character display (i.e., like objects you used in the object display lab)
- Support different color characters
- Modify the font ROM to create a different font or a set of symbols (i.e., rather than characters, create pictures)
- Support the movement of the cursor with other buttons

Pass Off

- Show your VGA character controller simulation in the testbench
- Demonstrate a working terminal