ECE 385

Fall 2022

Experiment #7

Lab 7

VGA Text Mode Controller with Avalon-MM Interface

Haocheng Yang((hy38)

Yicheng Zhou(yz69)

Introduction:

a.) In this lab we designed a rudimentary graphic card to store colors and display texts to the screen, the color and the text is controlled by C code.
For Lab 7.2, the VRAM is upgraded to on-chip-memory, which gives us much more space. Thus, we expended the text string data and enable it to choose its foreground color and background color from a 16-color pallet, which can be updated via C code. The text can also be controlled by C code.

b.) Compared to the design in Lab 6.2, the designed in Lab 7 is more software based. In other word, the content displayed by lab 6.2 design is logic based, and can not be changed in runtime. Any modification needs to happen on a hardware level. While the design in Lab 7 is software based and the content displayed can be modified on runtime and is controlled by C code.

Written Description of Lab 7 System:

a.) Week 1 (Monochrome Text Display)

1. For Lab 7.1, the VRAM is implemented as registers thus resulting in extremely long compilation time. The functionality is limited by routing and number of registers; thus, it can only display a single foreground color and a single back ground color for any given frame. The text displayed is unlimited.
   For Lab 7.2, the VRAM is upgraded to on-chip-memory, which gives us much more space. Thus, we expended the text string data and enable it to choose its foreground color and background color from a 16-color pallet, which can be updated via C code. The text can also be controlled by C code.

2. The VGA controller is synchronous with the processor clock, however, the onboard pixel clock is at 50% clock speed. The controller generates the horizontal sync and vertical sync signals. It also generates the blanking interval indicator. To aid the frame generation at the interface, it also generates a DrawX

& DrawY coordinate for pixel positioning. This module only takes the RGB values generated at processor for each pixel.

3. Data “Latches” are used to implement read/write operation of the registers. Due to VGA wiring can only take 8-bit data, thus data latches are used to read/write 32-bit (4x8). The latches are written when loading from vram with the address from processor, which returns 4 bytes(words).

4. Each character is stored in an 8-bit format, 7 bits if for the indexing for font-rom, which takes in index and translate it into 8x16 pixels maps. 1 bit is to indicate if the character’s color is inverted. An additional register is used to store the global foreground and background color’s RGB values. On a hardware level, the address of each pixel is determined as follows. First we get the column coordinate from DrawX and row coordinate from DrawY. Using (Col+Row*80)/4 we can determine the address for the register in which the current character is stored, and then using (Col+Row*80) we have the current character. Using the first 2 bits of the current character we can determine the specific bits inside the register the data for the character is stored, which is an 8-bit value. The first 7 of which is the font-rom index mentioned before and the 8 bit is the invert indicator.

5. The format of the control register is as follows:

```
        [[31-25]  [24-21]    [20-17][16-13][ 12-9][ 8-5 ][ 4-1 ][   0    ]
[[RS][FGD_R][FGD_G][FGD_B][BKG_R][BKG_G][BKG_B][Rs]
```

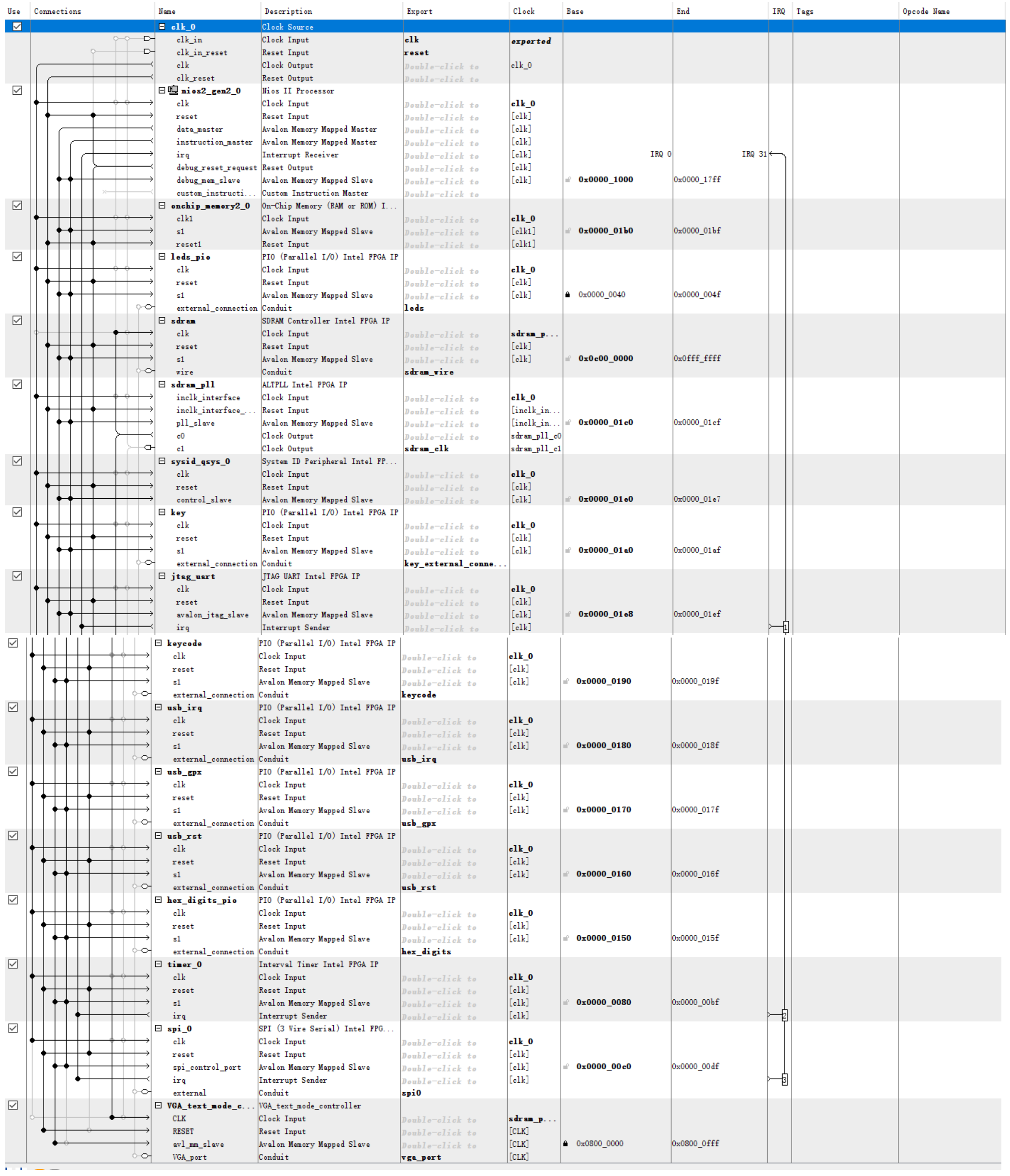
Each color is stored as 12bits data, 4 bit for RGB each. When the font-rom return is 0 and the invert-indicator is 0 or the font-rom return is 1 and the invert-indicator is 1, the interface will send the background colors’ RGB values to the controller, else it will output the foreground colors’ RGB values to the controller.

b.) Week 2 (Color Text Display)

1. To upgrade from monochrome display to color display, a few hardware changes have to be implemented. First we implemented the 16 color pallet using registers for fast accessing. Then we changed the data structure for the characters to 16-bit, 4bit is for the pallet index for foreground and 4bit for the pallet index for background, 1 bit for invert indicator and 7 bit for font-rom index. Due to the expression of data size, the previous register based vram implementation is no longer feasible. Thus an on-chip-memory based vram is implemented using Quartus IP. A 2 parallel R/W port ram, which is then instantiated in the control interface.

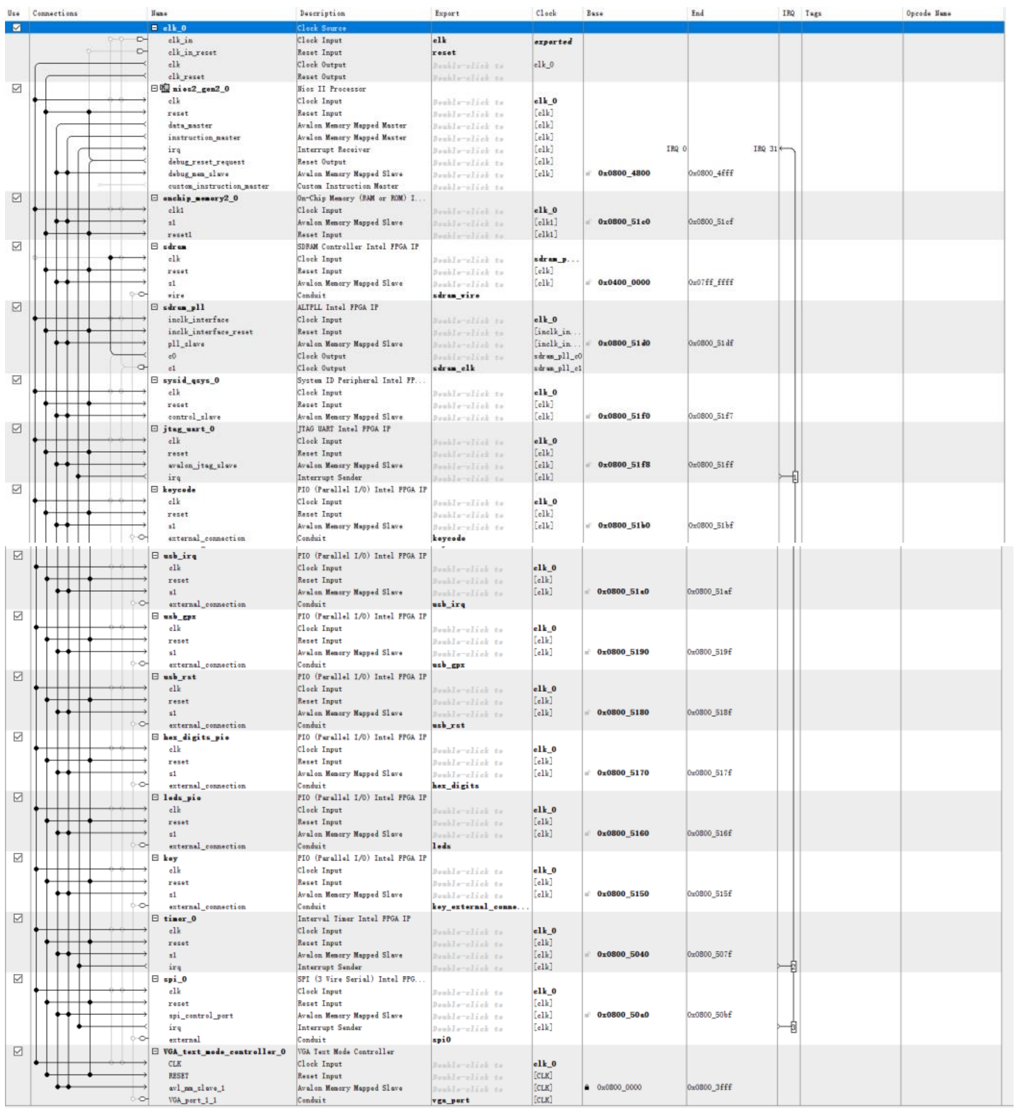
2. Modifications of platform designer IP:

   The platform designer of lab 71 and 72 are shown below:



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
✓		**clk_0**	Clock Source							
		clk_in	Clock Input	**clk**	***exported***					
		clk_in_reset	Reset Input	**reset**						
		clk	Clock Output	*Double-click to*	clk_0					
		clk_reset	Reset Output	*Double-click to*						
✓		**nios2_gen2_0**	Nios II Processor							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		data_master	Avalon Memory Mapped Master	*Double-click to*	[clk]					
		instruction_master	Avalon Memory Mapped Master	*Double-click to*	[clk]					
		irq	Interrupt Receiver	*Double-click to*	[clk]	IRQ 0	IRQ 31			
		debug_reset_request	Reset Output	*Double-click to*	[clk]					
		debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_1000**	0x0000_17ff			
		custom_instructi...	Custom Instruction Master	*Double-click to*						
✓		**onchip_memory2_0**	On-Chip Memory (RAM or ROM) I...							
		clk1	Clock Input	*Double-click to*	**clk_0**					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk1]	**0x0000_01b0**	0x0000_01bf			
		reset1	Reset Input	*Double-click to*	[clk1]					
✓		**leds_pio**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	0x0000_0040	0x0000_004f			
		external_connection	Conduit	**leds**						
✓		**sdram**	SDRAM Controller Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**sdram_p...**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0c00_0000**	0x0fff_ffff			
		wire	Conduit	**sdram_wire**						
✓		**sdram_pll**	ALTPLL Intel FPGA IP							
		inclk_interface	Clock Input	*Double-click to*	**clk_0**					
		inclk_interface_...	Reset Input	*Double-click to*	[inclk_in...					
		pll_slave	Avalon Memory Mapped Slave	*Double-click to*	[inclk_in...	**0x0000_01c0**	0x0000_01cf			
		c0	Clock Output	*Double-click to*	sdram_pll_c0					
		c1	Clock Output	**sdram_clk**	sdram_pll_c1					
✓		**sysid_qsys_0**	System ID Peripheral Intel FP...							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		control_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_01e0**	0x0000_01e7			
✓		**key**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_01a0**	0x0000_01af			
		external_connection	Conduit	**key_external_conne...**						
✓		**jtag_uart**	JTAG UART Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_01e8**	0x0000_01ef			
		irq	Interrupt Sender	*Double-click to*	[clk]			1		
✓		**keycode**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0190**	0x0000_019f			
		external_connection	Conduit	**keycode**						
✓		**usb_irq**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0180**	0x0000_018f			
		external_connection	Conduit	**usb_irq**						
✓		**usb_gpx**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0170**	0x0000_017f			
		external_connection	Conduit	**usb_gpx**						
✓		**usb_rst**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0160**	0x0000_016f			
		external_connection	Conduit	**usb_rst**						
✓		**hex_digits_pio**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0150**	0x0000_015f			
		external_connection	Conduit	**hex_digits**						
✓		**timer_0**	Interval Timer Intel FPGA IP							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_0080**	0x0000_00bf			
		irq	Interrupt Sender	*Double-click to*	[clk]			2		
✓		**spi_0**	SPI (3 Wire Serial) Intel FPG...							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		spi_control_port	Avalon Memory Mapped Slave	*Double-click to*	[clk]	**0x0000_00c0**	0x0000_00df			
		irq	Interrupt Sender	*Double-click to*	[clk]			3		
		external	Conduit	**spi0**						
✓		**VGA_text_mode_c...**	VGA_text_mode_controller							
		CLK	Clock Input	*Double-click to*	**sdram_p...**					
		RESET	Reset Input	*Double-click to*	[CLK]					
		avl_mm_slave	Avalon Memory Mapped Slave	*Double-click to*	[CLK]	0x0800_0000	0x0800_0fff			
		VGA_port	Conduit	**vga_port**	[CLK]					

Platform designer for lab71



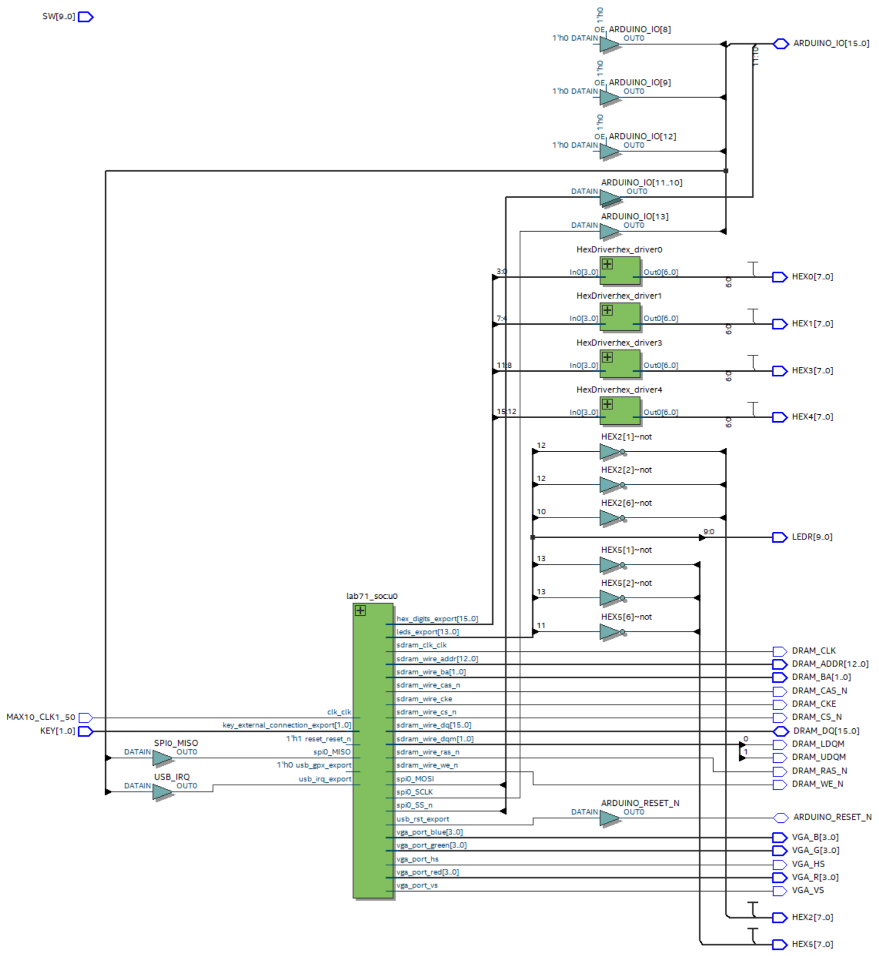
Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
☑		⊟ **clk_0**	Clock Source							
		clk_in	Clock Input	**clk**	***exported***					
		clk_in_reset	Reset Input	**reset**						
		clk	Clock Output	Double-click to	clk_0					
		clk_reset	Reset Output	Double-click to						
☑		⊟ **nios2_gen2_0**	Nios II Processor							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		data_master	Avalon Memory Mapped Master	Double-click to	[clk]					
		instruction_master	Avalon Memory Mapped Master	Double-click to	[clk]					
		irq	Interrupt Receiver	Double-click to	[clk]		IRQ 0	IRQ 31		
		debug_reset_request	Reset Output	Double-click to	[clk]					
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_4800**	0x0800_4fff			
		custom_instruction_master	Custom Instruction Master	Double-click to						
☑		⊟ **onchip_memory2_0**	On-Chip Memory (RAM or ROM) I...							
		clk1	Clock Input	Double-click to	**clk_0**					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk1]	**0x0800_51c0**	0x0800_51cf			
		reset1	Reset Input	Double-click to	[clk1]					
☑		⊟ **sdram**	SDRAM Controller Intel FPGA IP							
		clk	Clock Input	Double-click to	**sdram_p...**					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0400_0000**	0x07ff_ffff			
		wire	Conduit	**sdram_wire**						
☑		⊟ **sdram_pll**	ALTPLL Intel FPGA IP							
		inclk_interface	Clock Input	Double-click to	**clk_0**					
		inclk_interface_reset	Reset Input	Double-click to	[inclk_in...					
		pll_slave	Avalon Memory Mapped Slave	Double-click to	[inclk_in...	**0x0800_51d0**	0x0800_51df			
		c0	Clock Output	Double-click to	sdram_pll_c0					
		c1	Clock Output	**sdram_clk**	sdram_pll_c1					
☑		⊟ **sysid_qsys_0**	System ID Peripheral Intel FP...							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		control_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_51f0**	0x0800_51f7			
☑		⊟ **jtag_uart_0**	JTAG UART Intel FPGA IP							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_51f8**	0x0800_51ff			
		irq	Interrupt Sender	Double-click to	[clk]			1		
☑		⊟ **keycode**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_51b0**	0x0800_51bf			
		external_connection	Conduit	**keycode**						
☑		⊟ **usb_irq**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_51a0**	0x0800_51af			
		external_connection	Conduit	**usb_irq**						
☑		⊟ **usb_gpx**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_5190**	0x0800_519f			
		external_connection	Conduit	**usb_gpx**						
☑		⊟ **usb_rst**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_5180**	0x0800_518f			
		external_connection	Conduit	**usb_rst**						
☑		⊟ **hex_digits_pio**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_5170**	0x0800_517f			
		external_connection	Conduit	**hex_digits**						
☑		⊟ **leds_pio**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_5160**	0x0800_516f			
		external_connection	Conduit	**leds**						
☑		⊟ **key**	PIO (Parallel I/O) Intel FPGA IP							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_5150**	0x0800_515f			
		external_connection	Conduit	**key_external_conne...**						
☑		⊟ **timer_0**	Interval Timer Intel FPGA IP							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_5040**	0x0800_507f			
		irq	Interrupt Sender	Double-click to	[clk]			2		
☑		⊟ **spi_0**	SPI (3 Wire Serial) Intel FPG...							
		clk	Clock Input	Double-click to	**clk_0**					
		reset	Reset Input	Double-click to	[clk]					
		spi_control_port	Avalon Memory Mapped Slave	Double-click to	[clk]	**0x0800_50a0**	0x0800_50bf			
		irq	Interrupt Sender	Double-click to	[clk]			3		
		external	Conduit	**spi0**						
☑		⊟ **VGA_text_mode_controller_0**	VGA Text Mode Controller							
		CLK	Clock Input	Double-click to	**clk_0**					
		RESET	Reset Input	Double-click to	[CLK]					
		avl_mm_slave_1	Avalon Memory Mapped Slave	Double-click to	[CLK]	0x0800_0000	0x0800_3fff			
		VGA_port_1_1	Conduit	**vga_port**	[CLK]					

Platform designer for lab72

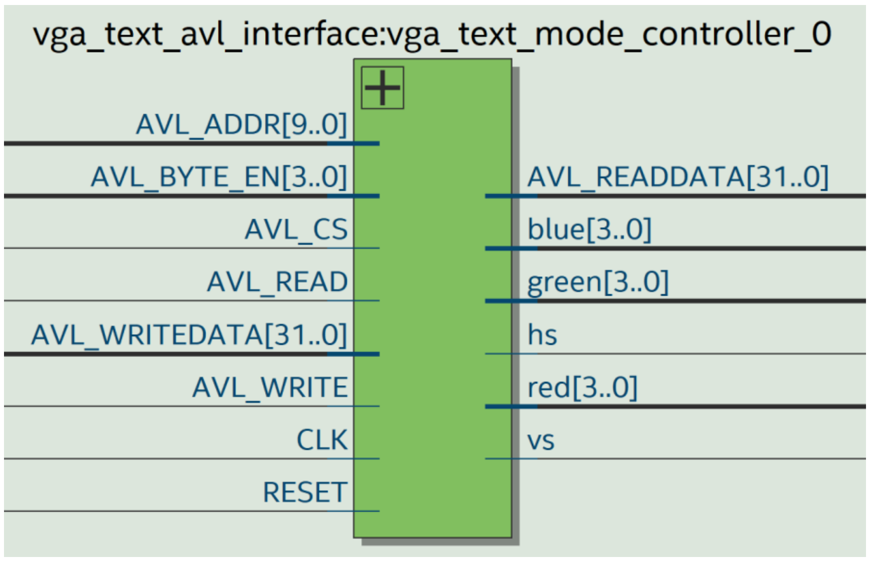
3. The pixel coordinate interpretation functions is also modified to accommodate the expanded data size. The new formula of register address is: 2*((DrawX%16)/8*8) +16. The RGB value generator is now changed to read the foreground and background pallet index from the character index and load in their respective RGB values to implement color display.

Block Diagram

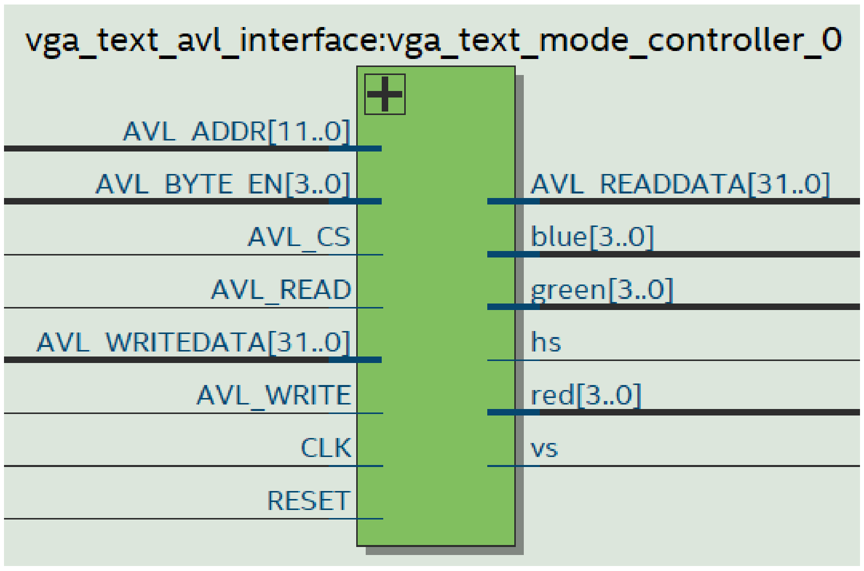
The RTL diagrams are shown below



Top level rtl diagram.



RTL Diagram for lab7.1



RTL Diagram for lab7.2

Module Descriptions

1.) vga_text_avl_interfave.sv:

Inputs: CLK, RESET, AVL_READ, AVL_WRITE, AVL_CS, [3:0] AVL_BYTE_EN, [11:0] AVL_ADDR, [31:0] AVL_WRITEDATA.

Outputs: hs, vs, [3:0] red, green, blue, [31:0]AVL_READDATA.

Description: the Avalon interface -mm. where VRAM are instantiated, and the hardware-based processor.

Purposes: this module takes in signals from Avalon interface, vram and rom to calculate the correct output to vga controller. This is also the gateway to and from the on-chip memory.

2.) VGA_controller.sv

Inputs: clk, Reset

Outputs: hs, vs, pixel_clk, blank, sync, [9:0] DrawX, [9:0]DrawY

Description: this is the VGA controller which sends signals through VGA port to the monitor.

Purpose: This is the module which determines the boundary of the screen, with it's on board clock, generates the pixel clock. It also generates the horizontal and vertical sync.

3.) Lab7.sv:

Inputs: MAX10_CLK1_50, [1:0] KEY, [9:0]SW, [15:0]DRA,_DQ, [15:0] ARDUINO_IO, Arduino_reset_n.

Outputs: [9:0] LEDR, [7:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, DRAM_CLK, DRAM_CKE, [12:0] DRAM_ADDR, [1:0] DRAM_BA, DRAM_LDQMM DRAM_UDQM, DRAM_CS_N, DRAM_WE_N, DRAM_CAS_N, DRAM_RAS_N, VGA_HS, VGA_VS, [3:0]VGA_R, [3:0]VGA_G, [3:0] VGA_B

Description: This is the top level entity of lab 7

Purpose: this calls on all other modules for functions and bridge data between other modules.

4.) HexDriver.sv:

Inputs: [3:0] In0

Outputs: [6:0] Out0

Description: This module converts 4bit-binry input and convert it to 7 bit for hex-display.

Purpose: This module converts 4bit binary numbers from registers into hexadecimal so LEDs can display outputs.

5.) font_rom.sv:

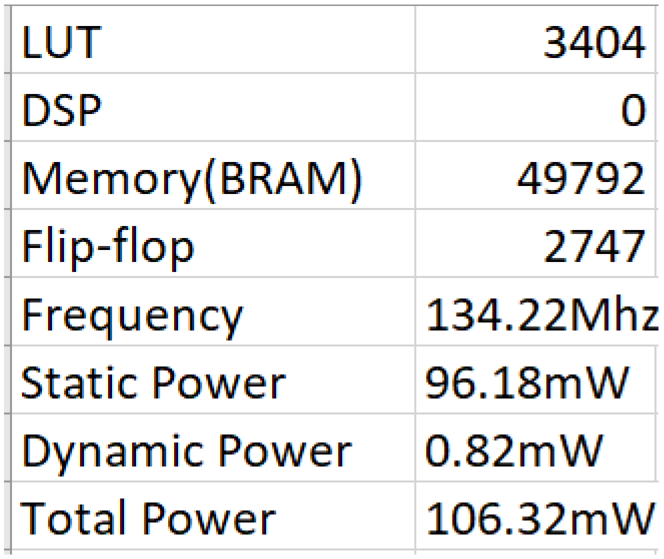
Input: [10:0] addr

Outputs: [7:0] data

Description: this is the rom where the font data is stored.

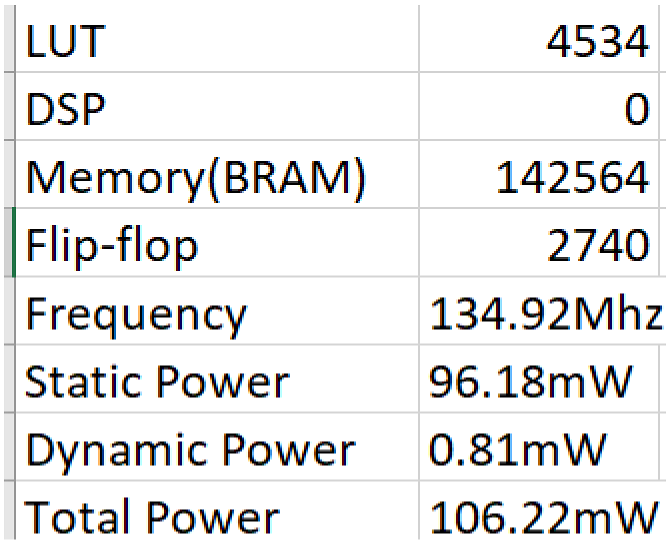
Purpose: This module is where the font data is stored and be loaded by index when a character need to be printed.

Document the design resources and statistics



LUT	3404
DSP	0
Memory(BRAM)	49792
Flip-flop	2747
Frequency	134.22Mhz
Static Power	96.18mW
Dynamic Power	0.82mW
Total Power	106.32mW

Table of data for lab 7.1



LUT	4534
DSP	0
Memory(BRAM)	142564
Flip-flop	2740
Frequency	134.92Mhz
Static Power	96.18mW
Dynamic Power	0.81mW
Total Power	106.22mW

Table of data for lab 7.2

The register vs on-chip memory implementation is that register have parallel read/write capability for every register simultaneously, meaning that all 602 register can be accessed at the same time, while the on-chip memory have only 2 channel, in our implementation both channel are read/write capable, but we can also use a write only channel and read only channel. However, the trade off for register based designed is that it uses up a lot of internal wiring, leading to routing issue and extremely long compilation time, where on-chip memory, although not as fast, but saves a lot of resources on LUT.

Conclusion:

We didn't encounter much difficulties on week 1, however, on lab2 we encountered many. First of which is that we tried to implement the on-chip memory with an sv file, but the compiler failed to recognize that, which led to multiple routing errors, we fix this issue using the altera IP GUI. Then we encountered many data format issues (due to this lab being written by 2 people) which only needed careful debugging. Then we encountered issue where while trying to write into the pallets, the both of the colors stored in the register got changed, which was solved by storing only 1 color in each register.

Extension for this lab is that we can store a sequence of images in format in the font_rom and make animation work. Currently only menu for our final project need system from this lab, but further use might be developed.

Combined with the lecture and Q/A posted on the wiki page, the information given are sufficient. However I do think that the manual made us misjudge the difficulty and time required for lab 7.2