ECE 385

Fall 2023

Experiment #7

HDMI Text Mode Controller with AXI4 Interface

Ziyi Lin
Zhuoyang Lai
TA: Jerry Wang

1.Introduction

a.

HDMI interface read and write through AXI4 bus, and drawX, drawY signal. We use two different kinds of storage methods, in particular, registers and on-chip-mem.

b.

The logic condition to set the color has changed, now color is dependent on the control register of color and font ROM to determine the background or foreground color.

Advantage:
We use packaged IP which can be repeatedly reused once being functional.
It can save a lot of time on design.
AXI bus is more suitable for large data transmission.
Disadvantage: Implementation of AXI bus IP is complicated, compared to lab 6.2.

2.Written Description of Lab 7 System

i.Written Description of the entire Lab 7 system

Create a monochrome graphics controller for the MicroBlaze to be connected via AXI4-Lite bus

ii.Describe at a high level your HDMI Text Mode controller IP.

HDMI_Text_Controller contains 600 words of VRAM, 1 control register, corresponding Font_ROM and our logic map draw_x, draw_y to char, the vga sycronize generator. The input port is Clock_signal, Reset signal, Output port is HDMI output.

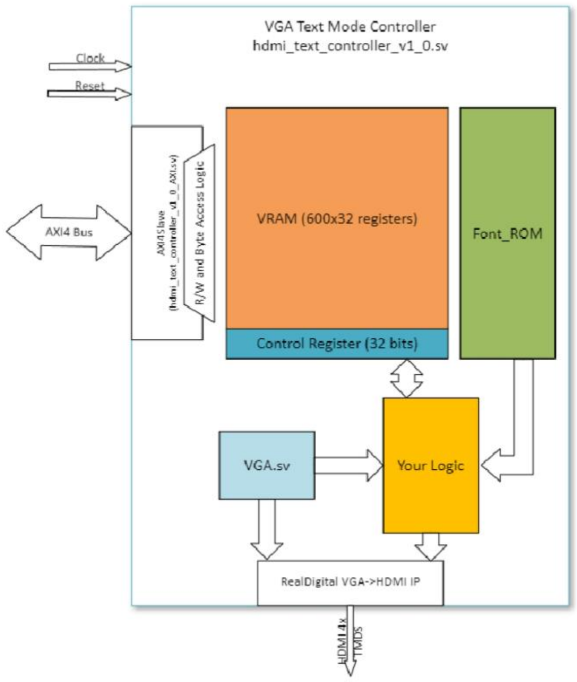
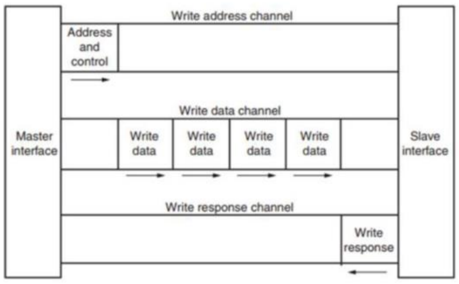


Fig1. HDMI Text Mode controller

iii.Describe the logic used to read and write your HDMI AXI registers.

The AXI4-Lite interface consists of five channels: Read Address, Read Data, Write Address, Write Data, and Write Response. An AXI4 read transaction using the Read Address and Data channels is shown in Figure 3. Address and control signal tells slave which register to write and write data tells the slave what data should be write into register.

Similarly, an AXI4 write transaction using the Write Address, Data, and Response channels is shown in Figure 2. Address and control signal tells the slave which register to read and data read from register will be returned back on Read data Channel.



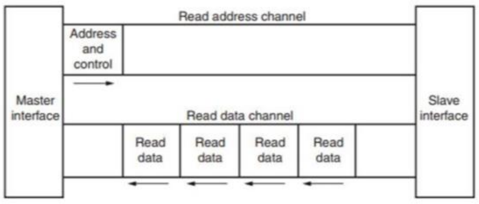


Fig 2. AXI Write

Fig3. AXI Read

iv.Describe the algorithm used to draw the text characters from the VRAM and font ROM

First, map draw_x, draw_y to corresponding register.

```
Reg_id = (DrawY/16)*20 + DrawX/32
```

Second, calculate the corresponding top left pixel of register block.

```
Top_x = 32*(reg_id % 20)
Top_y = 32*(addr_reg % 20)
```

Third, find which four parts of register draw_x and draw_y belongs to

```
If DrawX - shape_x >= 24://draw_x belongs to last part of reg
Find the corresponding row in font_rom
row= DrawY - shape_y + 16*slv_regs[addr_reg][30:24]
Call the module font_rom(.addr(row),.data(sprite_data))
    Value stored in sprite_data is exactly the char we need to print
If DrawX - shape_x >= 16 && DrawX - shape_x <24://draw_x belongs to third part of reg
Follow the previous pseudocode
If DrawX - shape_x >= 8 && DrawX - shape_x <16://draw_x belongs to second part of reg
Follow the previous pseudocode
Else:  // draw_x belongs to last part of reg
Follow the previous pseudocode
```

Fourth, after getting the value of sprite_data, visit the control register, the following implementation is described in v.

v.Describe your implementation of the inverse color bit, as well as the implementation of the control register.

Visit the control register in axi file and find the corresponding RGB color.

Bit	31-25	24-21	20-17	16-13	12-9	8-5	4-1	0
Function	UNUSED	FGD_R	FGD_G	FGD_B	BKG_R	BKG_G	BKG_B	UNUSED

If inverse color bit in vram reg is 1, then reverse the background and foreground color.

Bit	31	30-24	23	22-16	15	14-8	7	6-0
Function	IV3	CODE3	IV2	CODE2	IV1	CODE1	IV0	CODE0

Following is psudocode

```
if((sprite_data[index])^iv_bit)
    Then color = foreground.
    Else color = background.
```

b.Week 2 (Color Text Display)

i.Describe the hardware changes:

1.We remove the registers and instantiate on chip memory. The on-chip memory blocks (BRAM) blocks on the FPGA have two ports, each of which may be read or write. We use true dual port which both ports can read and write. one side interact with axi bus, the other port is read only which returns word address of draw_x and draw_y which will be further used for font display.

2.Corresponding modifications to the IP Editor.

We remove the registers and instantiate on chip memory and change the color mapper according to the new memory map

3.Modified sprite drawing algorithm with the updated indexing equations from on-screen pixels to VRAM.

First find the correspond word adder of draw_x and draw_y

```
Word_id = (DrawY/16)*40 + DrawX/16
```

Second find the top left pixel of corresponding word_id

```
top_y = 16*(DrawY/16)
top_x = 16*(DrawX/16)
```

Third identify draw_x, draw_y belongs to the first char or the second char.

```
DrawX - shape_x >= 8://belong to second char
Find row in font_rom file:
sprite_addr = DrawY - shape_y + 16*reg_data[30:24]
Else://belongs to first char
sprite_addr = DrawY - shape_y + 16*reg_data[14:8]
```

Table 7. Bit Encoding for VRAM (Color Mode, Word Addresses 0x000-0x4AF)

Bit	31	30-24	23-20	19-16	15	14-8	7-4	3-0
Function	IV1	CODE1	FGD_IDX1	BKG_IDX1	IV0	CODE0	FGD_IDX0	BKG_IDX0

Forth after find the sprite_adder in rom file, call the font_rom mudule and return the content of char

```
font_rom font_rom(.addr(sprite_addr),.data(sprite_data))
```

color selection will be describe in part4

4.modifications to support multicolored text

Find the color index from the word adder, which is 4 bits foreground color and 4 bits background color. Visit the slv_reg (extended to 8 registers which hold 16 kinds of color) and according to the 4 bits index, find the corresponding RGB.

Address	31-25	24-21	20-17	16-13	12-9	8-5	4-1	0
0x800	UNUSED	C1_R	C1_G	C1_B	C0_R	C0_G	C0_B	UNUSED

If inverse bit in char is 1 then inverse the foreground and background color.

5.Additional hardware/code to draw paletted colors.

To differentiate whether we should visit the paletted colors reg or memory on chip, we should assert if adder is larger than 0x800. After then we should minus the adder by 2048(0x800).

3.Block Diagram.

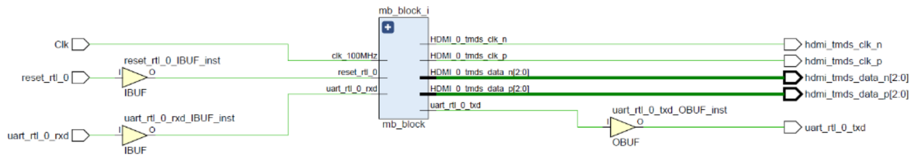


Fig4. Top level for 7.1

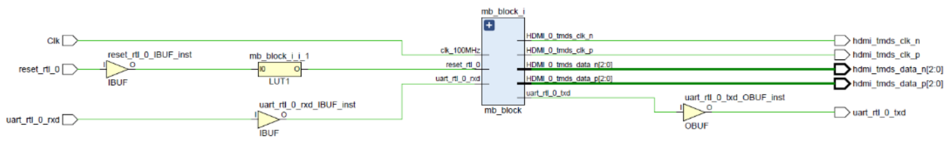


Fig5. Top level for 7.2

b.Internal block diagram of the IP you designed for each week.

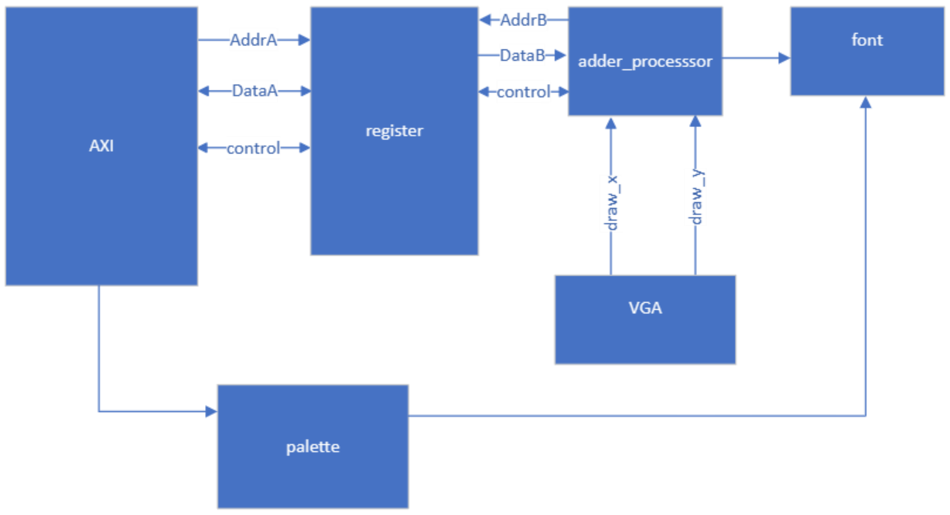


Fig6. Summary of internal block diagram for lab7.1

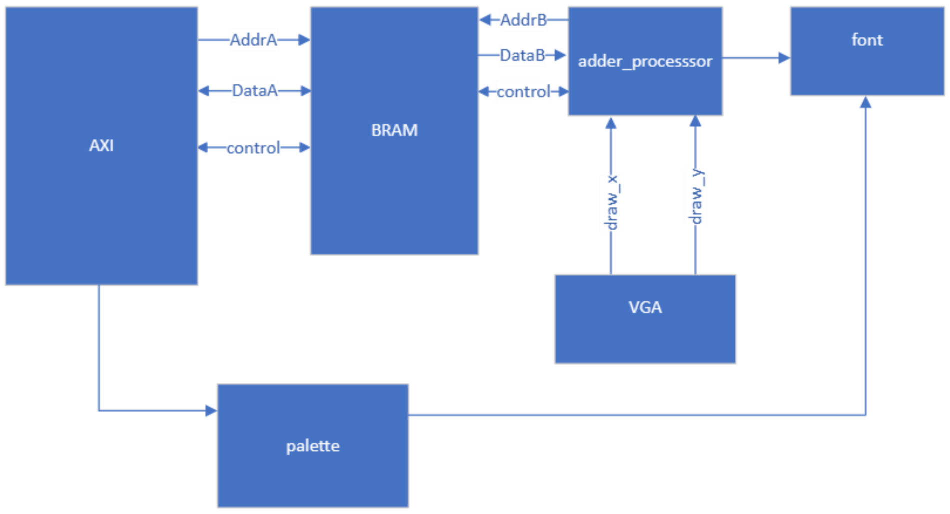


Fig7. Summary of internal block diagram for lab7.2

In the control signal we set a FSM that wait for 3 clock to read the data from BRAM

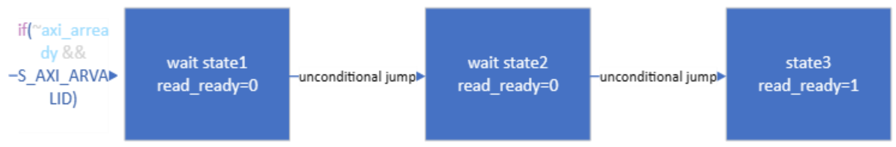


Fig8. FSM for lab7.2

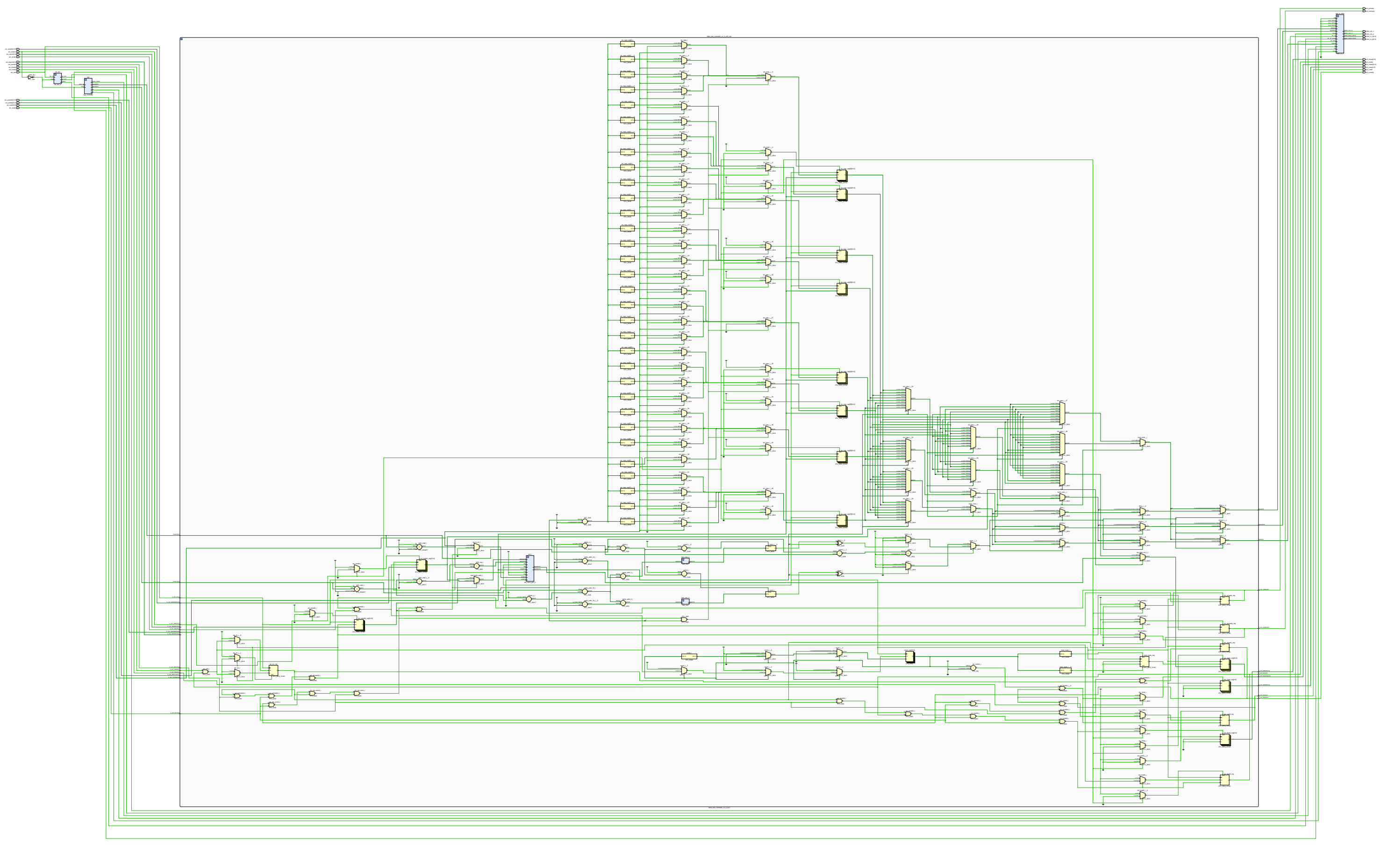


Fig9. internal block diagram of IP for lab7.2 from vivado

4.Module Descriptions.

Module: lab7_toplevel.sv
Input: reset_rtl_0, uart_rtl_0_rxd, Clk
Output: hdmi_tmds_clk_n, hdmi_tmds_clk_p, [2:0] hdmi_tmds_data_n, hdmi_tmds_data_p
Description: The lab7_toplevel module is designed for a project involving both serial communication (UART) and video output (HDMI).
Purpose: Acts as a bridge between various peripherals and a core processing module

Module: clk_wiz_1
Input: clk_100MHZ, reset
Output: clk_out1, locked
Description: The Clocking Wizard enables designers to generate clock signals with specific frequencies, can be reset via reset signal.
Purpose: setting the desired clock frequency without the need for manually coding

Module: mdm (Micro blaze debug module)
Input: debug signal
Output: debug data
Description: providing debugging and monitoring capabilities for MicroBlaze-based FPGA designs.
Purpose: facilitate the debugging and monitoring of MicroBlaze soft processors embedded in FPGA designs.

Module: MicroBlaze
Input: M_AXI_DP (Peripheral Data Interface, AXI4-Lite or AXI4 interface)
DLMB(Data interface, Local Memory Bus (BRAM only))
M_AXI_IP (Peripheral Instruction interface, AXI4-Lite interface)
ILMB (Instruction interface, Local Memory Bus (BRAM only))
S0_AXIS..S15_AXIS (AXI4-Stream interface slave direct connection interfaces)
M_AXI_DC (Data-side cache AXI4 interface)
M_ACE_DC (Data-side cache AXI Coherency Extension (ACE) interface)
M_AXI_IC (Instruction-side cache AXI4 interface)
M_ACE_IC (Instruction-side cache AXI Coherency Extension (ACE) interface)
Output: M_AXI_DP (Peripheral Data Interface, AXI4-Lite or AXI4 interface)
DLMB (Data interface, Local Memory Bus (BRAM only))
M0_AXIS..M15_AXIS(AXI4-Stream interface master direct connection interface)
M_AXI_DC (Data-side cache AXI4 interface)
M_ACE_DC (Data-side cache AXI Coherency Extension (ACE) interface)
Description: The MicroBlaze core is organized as a Harvard architecture with separate bus interface units for data and instruction accesses.
Purpose: compute, modify and read memory and finish tasks according to the instruction.

Module: axi_uart
Inputs: S_AXI_ACLK,S_AXI_ARESETN,Freeze,
  S_AXI_AWADDR[C_S_AXI_ADDR_WIDTH-1:0], S_AXI_AWVALID,
  S_AXI_WDATA[C_S_AXI_ DATA_WIDTH - 1: 0],
  S_AXI_WSTB[C_S_AXI_ DATA_WIDTH/8-1:0] ,
  S_AXI_WVALID, S_AXI_BREADY,
  S_AXI_ARADDR[C_S_AXI_ADDR_WIDTH[1:0], S_AXI_ARVALID,
  S_AXI_RREADY
Outputs: IP2INTC_Irpt, S_AXI_AWREADY, S_AXI_WREADY, S_AXI_BRESP[1:0], S_AXI_BVALID, S_AXI_ARREADY, S_AXI_RDATA[C_S_AXI_DATA_WIDTH [1:0], S_AXI_RRESP[1:0], S_AXI_RVALID //from datasheet
Description: UART is a hardware module that provides asynchronous serial communication. It is used for transmitting and receiving data between devices in a serial fashion. By using AXI4-Lite Interface Signals we can control UART.
Purpose: facilitate serial communication between digital devices.

Module: microblaze_0_axi_intc
Inputs: s_axi_i, interrupt_address
Outputs: s_axi_o, processor_ack
Description: generate interrupt requests that need to be efficiently managed.
Purpose: managing interrupts in complex SoC designs

Module: microblaze_0_axi_intc
I/O: Read Address Channel Signals
  Read Data Channel Signals
  Write Address Channel Signals
  Write Data Channel Signals
  Write Response Channel Signals
Description: connects one or more AXI memory-mapped Master devices to one or more memory-mapped Slave devices.
Purpose: manages the AXI transactions between AXI masters and AXI slaves.

Module: hdmi_text_controller_0
Inputs: axi_i, axi_aclk, axi_aresetn
Outputs: axi_o, HDMI bus
Description: HDMI_Text_Controller contains 600 words of VRAM, 1 control register, corresponding Font_ROM and our logic map draw_x, draw_y to char, the vga sycronize generator. The input port is Clock_signal, Reset signal, Output port is HDMI output.
Purpose: a map that store what char should draw at corresponding draw_x and draw_y
the internals for each week's version of the HDMI controller IP.
Both for week1 and week2

Module: hdmi_text_controller_v1_0.sv
Input: axi_aclk, axi_aresetn, axi_awaddr[11:0], axi_awprot[2:0], axi_awvalid, axi_awready, axi_wdata[31:0], axi_wstrb[3:0], axi_wvalid, axi_wready, axi_bresp[1:0], axi_bvalid, axi_bready, axi_araddr[11:0], axi_arprot[2:0], axi_arvalid, axi_arready, axi_rdata[31:0], axi_rresp[1:0], axi_rvalid, and axi_rready.
Output: hdmi_clk_n, hdmi_clk_p, hdmi_tx_n[2:0], and hdmi_tx_p[2:0]
Description: This module essentially acts as a display controller, taking input coordinates and generating RGB color values based on the drawn content and configured colors. The AXI4-Lite interface allows for communication with a master device, making it suitable for integration into larger systems.
Purpose: interfaces with an HDMI display

Module: VGA_Controller.sv
Inputs: pixel_clk, reset
Outputs: hs, vs, active_nblank, sync, [9:0] drawX, [9:0] drawY
Description: The vga_controller manages VGA display synchronization for a 640x480 resolution. It counts pixel positions and creates sync signals based on VGA timing. The module uses a 25MHz clock and use counter to generate horizontal sync pulse and vertical sync pulse.
Purpose: To produce control and sync signals for interfacing with a VGA display.

Module: font_rom.sv
Input: [10:0] addr
Output: [7:0] data
Description: The font_rom module in Verilog is a read-only memory (ROM) specifically designed for storing font data, used in a VGA display system.
Purpose: To display a character, the system would access the corresponding set of addresses in this ROM to retrieve the pixel rows for that character.

Module: hdmi_text_controller_v1_0_AXI
Input: DrawX, DrawY, Red, Green, Blue, S_AXI_ACLK, S_AXI_ARESETN, S_AXI_AWADDR, S_AXI_AWPROT, S_AXI_AWVALID, S_AXI_WDATA, S_AXI_WSTRB, S_AXI_WVALID, S_AXI_RREADY, S_AXI_ARADDR, S_AXI_ARPROT, S_AXI_ARVALID
Output: S_AXI_AWREADY, S_AXI_WREADY, S_AXI_BRESP, S_AXI_BVALID, S_AXI_ARREADY, S_AXI_RDATA, S_AXI_RRESP, S_AXI_RVALID.
Description: The vga_controller manages VGA display synchronization for a 640x480 resolution. This file specify under what condition should salve and master start to read and write data and when to load data from the bus.
Purpose: To produce control and sync signals for interfacing with a VGA display.
Specific for lab7.2

Module: blk_mem_gen_0
Input: clka, addra, dina, ena, wea, rsta, regcea, clkb, addrb, dinb, enb, web, rstb, regceb
Output: douta, doutb,
Description: Block RAMs are used for storing large amounts of data inside of FPGA.
Purpose: to extend the functionality and capability of a single primitive to memories of arbitrary widths and depths.

5. Design Resources and Statistics

LUT	15688
DSP	3
Memory(BRAM)	32
Flip-Flop	21167
Frequency	71.154MHZ //WNS<0
Static Power	0.482w
Dynamic Power	0.074w
Total Power	0.408w

Lab7.1

LUT	2664
DSP	3
Memory(BRAM)	34
Flip-Flop	1821
Frequency	102.438MHZ
Static Power	0.074w
Dynamic Power	0.378w
Total Power	0.452w

Lab7.2

the difference: if we implement register, then we can get access to all 601 registers at the same time, but on_chip memory only has two ports and two memory locations may be accessed simultaneously, also, there is latency if we want to read data from on chip memory.
The lab 7.2 is more efficient, because the LUT is less and the WNS is positive, the tradeoff for 7.2 is there is latency in read data, we need to wait for 3 clock cycle.

6.Conclusion

a. Create a monochrome graphics controller for the MicroBlaze to be connected via AXI4-Lite bus.
b. This lab teaches us set up a graphics controller, if we are developing a game, it provides us with example to set up on chip memory and how to show them on the VGA display.
c. the lab is overall helpful.