| Ousmane Toure & Jianning Chen  EECE2160 | Embedded Design: Enabling Robotics  Lab Assignment 8 |
| --- | --- |

Lab Assignment 8

Ousmane Toure & Jianning Chen

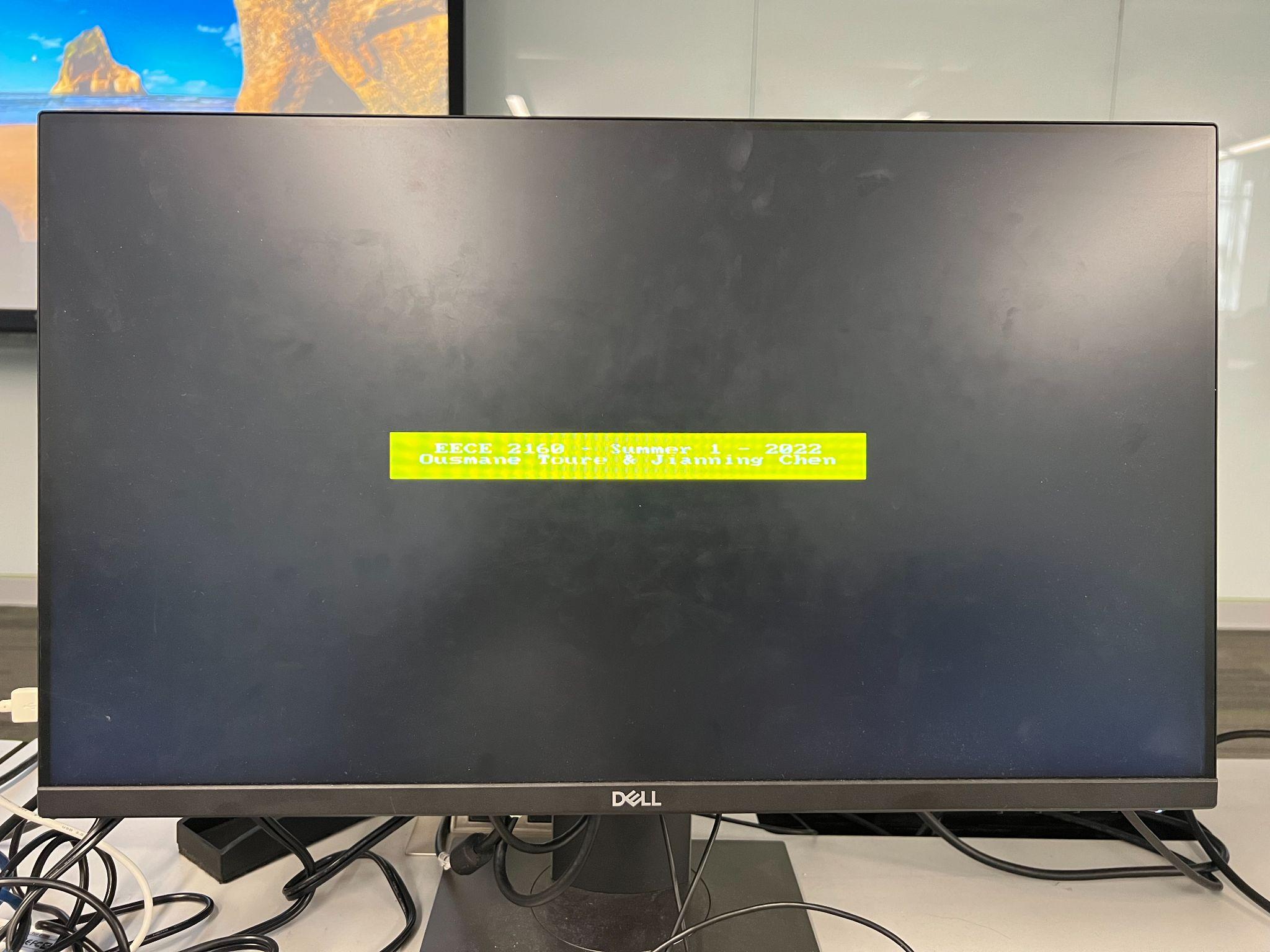
[Toure.o@northeastern.edu](mailto:Toure.o@northeastern.edu)

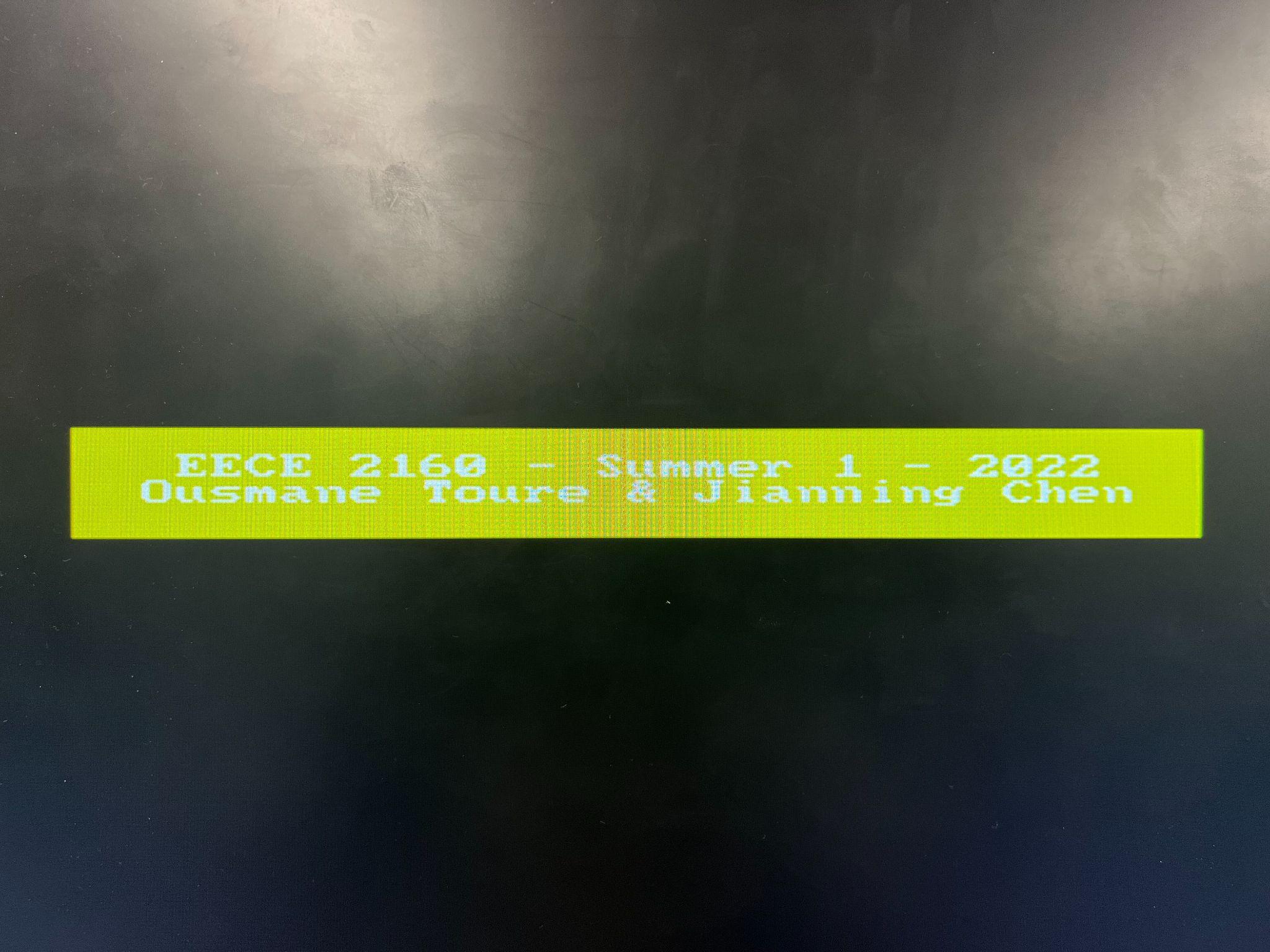
[chen.jiann@northeastern.edu](mailto:chen.jiann@northeastern.edu)

Submit date:6/23/2022

Due Date: 6/23/2022

**8.1 Running Result**





**8.2 Code**

lab8\_final.cpp

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <iostream>

#include “address\_map\_arm.h”

using namespace std;

void video\_text(int, int, char \*, char \*);

void video\_box(int, int, int, int, short,char \*);

int resample\_rgb(int, int);

int get\_data\_bits(int);

#define STANDARD\_X 320

#define STANDARD\_Y 240

#define YELLOW 0xD1D100

int screen\_x;

int screen\_y;

int res\_offset;

int col\_offset;

// Physical base/starting address of FPGA Devices

const unsigned int LW\_BRIDGE\_BASE = 0xC8000000; // Base offset

const unsigned int FINAL\_PHYSICAL\_ADDRESS = 0xFF203020; // Final offset

const unsigned int LW\_BRIDGE\_SPAN = FINAL\_PHYSICAL\_ADDRESS - LW\_BRIDGE\_BASE;

char \*Initialize(int \*fd)

{

// Open /dev/mem to give access to physical addresses

\*fd = open( "/dev/mem", (O\_RDWR | O\_SYNC));

if (\*fd == -1) // check for errors in openning /dev/mem

{

cout << "ERROR: could not open /dev/mem..." << endl;

exit(1);

}

// Get a mapping from physical addresses to virtual addresses

char \*virtual\_base = (char \*)mmap (NULL, LW\_BRIDGE\_SPAN, (PROT\_READ | PROT\_WRITE), MAP\_SHARED, \*fd, LW\_BRIDGE\_BASE);

if (virtual\_base == MAP\_FAILED) // check for errors

{

cout << "ERROR: mmap() failed..." << endl;

close (\*fd); // close memory before exiting

exit(1); // Returns 1 to the operating system;

}

return virtual\_base;

}

/\*

\* Close general-purpose I/O.

\* @param pBase Virtual address where I/O was mapped.

\* @param fd File descriptor previously returned by 'open'.

\*/

void Finalize(char \*pBase, int fd)

{

if (munmap (pBase, LW\_BRIDGE\_SPAN) != 0){

cout << "ERROR: munmap() failed..." << endl;

exit(1);

}

close (fd); // close memory

}

int main(void) {

int fd;

char \*pBase = Initialize(&fd);

volatile int \* video\_resolution = (int \*)(pBase + PIXEL\_BUF\_CTRL\_BASE - LW\_BRIDGE\_BASE);

screen\_x = \*video\_resolution & 0xFFFF;

screen\_y = (\*video\_resolution >> 16) & 0xFFFF;

volatile int \* rgb\_status = (int \*)(pBase + RGB\_RESAMPLER\_BASE - LW\_BRIDGE\_BASE);

int db = get\_data\_bits(\*rgb\_status & 0x3F);

// check if resolution is smaller than the standard 320 x 240

res\_offset = (screen\_x == 160) ? 1 : 0;

// check if number of data bits is less than the standard 16-bits

col\_offset = (db == 8) ? 1 : 0;

// create a message to be displayed on the video and LCD displays

char text\_top\_row[40] = " EECE 2160 - Summer 1 - 2022\0";

char text\_bottom\_row[40] = "Ousmane Toure & Jianning Chen\0";

char empty[50] = " \0";

// update color

short background\_color = resample\_rgb(db, YELLOW);

video\_text(20, 29, empty, pBase);

video\_text(20, 30, empty, pBase);

video\_text(25, 29, text\_top\_row,pBase);

video\_text(25, 30, text\_bottom\_row,pBase);

video\_box(0, 0, STANDARD\_X, STANDARD\_Y, 0,pBase); // clear the screen

video\_box(23 \* 4, 28 \* 4, 56 \* 4 - 1, 32 \* 4 , background\_color,pBase);

Finalize(pBase, fd);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Subroutine to send a string of text to the video monitor

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void video\_text(int x, int y, char \* text\_ptr, char \*pBase) {

int offset;

volatile char \* character\_buffer = (char \*)(pBase + FPGA\_CHAR\_BASE - LW\_BRIDGE\_BASE); // video

offset = (y << 7) + x;

while (\*(text\_ptr)) {

\*(character\_buffer + offset) = \*(text\_ptr); // write to the character buffer

++text\_ptr;

++offset;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Draw a filled rectangle on the video monitor

\* Takes in points assuming 320x240 resolution and adjusts based on differences in resolution and color bits.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void video\_box(int x1, int y1, int x2, int y2, short pixel\_color,char \*pBase) {

volatile int pixel\_buf\_ptr = \*(int \*)(pBase + PIXEL\_BUF\_CTRL\_BASE - LW\_BRIDGE\_BASE);

int pixel\_ptr, row, col;

int x\_factor = 0x1 << (res\_offset + col\_offset);

int y\_factor = 0x1 << (res\_offset);

x1 = x1 / x\_factor;

x2 = x2 / x\_factor;

y1 = y1 / y\_factor;

y2 = y2 / y\_factor;

/\* assume that the box coordinates are valid \*/

for (row = y1; row <= y2; row++) {

for (col = x1; col <= x2; ++col) {

pixel\_ptr = pixel\_buf\_ptr + (row << (10 - res\_offset - col\_offset)) + (col << 1);

\*(short \*)(pixel\_ptr + pBase -LW\_BRIDGE\_BASE) = pixel\_color; // set pixel color

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Resamples 24-bit color to 16-bit or 8-bit color

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int resample\_rgb(int num\_bits, int color) {

if (num\_bits == 8) {

color = (((color >> 16) & 0x000000E0) | ((color >> 11) & 0x0000001C) |

((color >> 6) & 0x00000003));

color = (color << 8) | color;

} else if (num\_bits == 16) {

color = (((color >> 8) & 0x0000F800) | ((color >> 5) & 0x000007E0) |

((color >> 3) & 0x0000001F));

}

return color;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Finds the number of data bits from the mode

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int get\_data\_bits(int mode) {

switch (mode) {

case 0x0:

return 1;

case 0x7:

return 8;

case 0x11:

return 8;

case 0x12:

return 9;

case 0x14:

return 16;

case 0x17:

return 24;

case 0x19:

return 30;

case 0x31:

return 8;

case 0x32:

return 12;

case 0x33:

return 16;

case 0x37:

return 32;

case 0x39:

return 40;

}

}

address\_map\_arm.h

/\* This files provides address values that exist in the system \*/

#define BOARD "DE1-SoC"

/\* Memory \*/

#define DDR\_BASE 0x00000000

#define DDR\_END 0x3FFFFFFF

#define A9\_ONCHIP\_BASE 0xFFFF0000

#define A9\_ONCHIP\_END 0xFFFFFFFF

#define SDRAM\_BASE 0xC0000000

#define SDRAM\_END 0xC3FFFFFF

#define FPGA\_PIXEL\_BUF\_BASE 0xC8000000

#define FPGA\_PIXEL\_BUF\_END 0xC803FFFF

#define FPGA\_CHAR\_BASE 0xC9000000

#define FPGA\_CHAR\_END 0xC9001FFF

/\* Cyclone V FPGA devices \*/

#define LED\_BASE 0xFF200000

#define LEDR\_BASE 0xFF200000

#define HEX3\_HEX0\_BASE 0xFF200020

#define HEX5\_HEX4\_BASE 0xFF200030

#define SW\_BASE 0xFF200040

#define KEY\_BASE 0xFF200050

#define JP1\_BASE 0xFF200060

#define JP2\_BASE 0xFF200070

#define PS2\_BASE 0xFF200100

#define PS2\_DUAL\_BASE 0xFF200108

#define JTAG\_UART\_BASE 0xFF201000

#define JTAG\_UART\_2\_BASE 0xFF201008

#define IrDA\_BASE 0xFF201020

#define TIMER\_BASE 0xFF202000

#define TIMER\_2\_BASE 0xFF202020

#define AV\_CONFIG\_BASE 0xFF203000

#define RGB\_RESAMPLER\_BASE 0xFF203010

#define PIXEL\_BUF\_CTRL\_BASE 0xFF203020

#define CHAR\_BUF\_CTRL\_BASE 0xFF203030

#define AUDIO\_BASE 0xFF203040

#define VIDEO\_IN\_BASE 0xFF203060

#define EDGE\_DETECT\_CTRL\_BASE 0xFF203070

#define ADC\_BASE 0xFF204000

/\* Cyclone V HPS devices \*/

#define HPS\_GPIO1\_BASE 0xFF709000

#define I2C0\_BASE 0xFFC04000

#define I2C1\_BASE 0xFFC05000

#define I2C2\_BASE 0xFFC06000

#define I2C3\_BASE 0xFFC07000

#define HPS\_TIMER0\_BASE 0xFFC08000

#define HPS\_TIMER1\_BASE 0xFFC09000

#define HPS\_TIMER2\_BASE 0xFFD00000

#define HPS\_TIMER3\_BASE 0xFFD01000

#define FPGA\_BRIDGE 0xFFD0501C

/\* ARM A9 MPCORE devices \*/

#define PERIPH\_BASE 0xFFFEC000 // base address of peripheral devices

#define MPCORE\_PRIV\_TIMER 0xFFFEC600 // PERIPH\_BASE + 0x0600

/\* Interrupt controller (GIC) CPU interface(s) \*/

#define MPCORE\_GIC\_CPUIF 0xFFFEC100 // PERIPH\_BASE + 0x100

#define ICCICR 0x00 // offset to CPU interface control reg

#define ICCPMR 0x04 // offset to interrupt priority mask reg

#define ICCIAR 0x0C // offset to interrupt acknowledge reg

#define ICCEOIR 0x10 // offset to end of interrupt reg

/\* Interrupt controller (GIC) distributor interface(s) \*/

#define MPCORE\_GIC\_DIST 0xFFFED000 // PERIPH\_BASE + 0x1000

#define ICDDCR 0x00 // offset to distributor control reg

#define ICDISER 0x100 // offset to interrupt set-enable regs

#define ICDICER 0x180 // offset to interrupt clear-enable regs

#define ICDIPTR 0x800 // offset to interrupt processor targets regs

#define ICDICFR 0xC00 // offset to interrupt configuration regs

# 8.3 Code Explanation

Based on “*DE1-SoC\_Computer\_System\_with\_ARM\_Cortex\_A9\_For\_C++\_Only.pdf*” (Simplified as “*DCSACAC.pdf*”), this lab made some changes on the code provided from page 63 to 65. Originally the code provided will cause segmentation fault. That is because the physical address is not mapped to the functions provided. On information provided on page 20 and page 21 of “*DCSACAC.pdf*”, the *Buffer* register, which is a part of the pixel buffer controller, is on the physical address of 0xFF203020. Therefore, similar to what has been done in lab6, the base address returned by mmap, which is named pBase, will be used to map the functions to the corresponding physical address. Also, similar to lab7, in order to control the video output, the bridge span has to be changed, which is the address of the buffer register minus the bridge base. The bridge span will be used in the initialize function to set up the video output.

In the main function, the video text function and the video box function, all base addresses declared in “address\_map\_arm.h” have been changed by adding pBase - LW\_BRIDGE\_BASE. And the pointers that point to the addresses are all declared to be volatile variables so that they cannot be optimized by the compiler by any chance. Doing this allows all the buffer pointers to point to the correct corresponding physical addresses. Once the addresses are correctly set up, a box and 2 lines of texts can be shown on the screen.

The final step is to change the texts and the color of the text box, and to place the text and the box in the middle of the screen. This is done by changing the number parameters inside the video box and video text functions. In the code provided in “*DCSACAC.pdf*”, the y-values are already centered, but the x-values have to be decreased in order to move the box and the text from the right of the screen to the middle of the screen. The color code used in the lab for the box is 0xD1D100, which is a darker yellow that shows the white texts more clearly. In addition, if a text is written in the previous code execution and on the next code execution the text placement is changed, it would cause some problems because the previous text won’t be erased. Therefore, in order to make the text showing correctly, in each execution, an empty line would be printed first in order to cover the texts in previous code execution, and then new texts will be created.

# References

1. Prof. Julius Marpaung, “*Lab Report Guide*”, Northeastern University, January 6 2020.
2. “*DE1-SoC\_Computer\_System\_with\_ARM\_Cortex\_A9\_For\_C++\_Only.pdf*”, Intel Corporation - FPGA university program, November 2017.