| Ousmane Toure & Jianning Chen  EECE2160 | Embedded Design: Enabling Robotics  Lab Assignment 7 |
| --- | --- |

Lab Assignment 7

Ousmane Toure & Jianning Chen

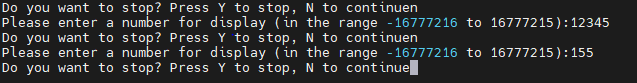
[Toure.o@northeastern.edu](mailto:Toure.o@northeastern.edu)

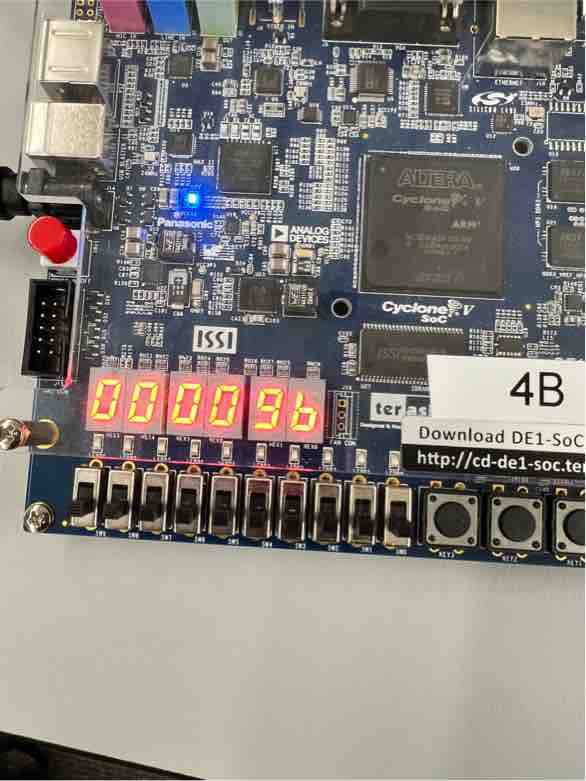
[chen.jiann@northeastern.edu](mailto:chen.jiann@northeastern.edu)

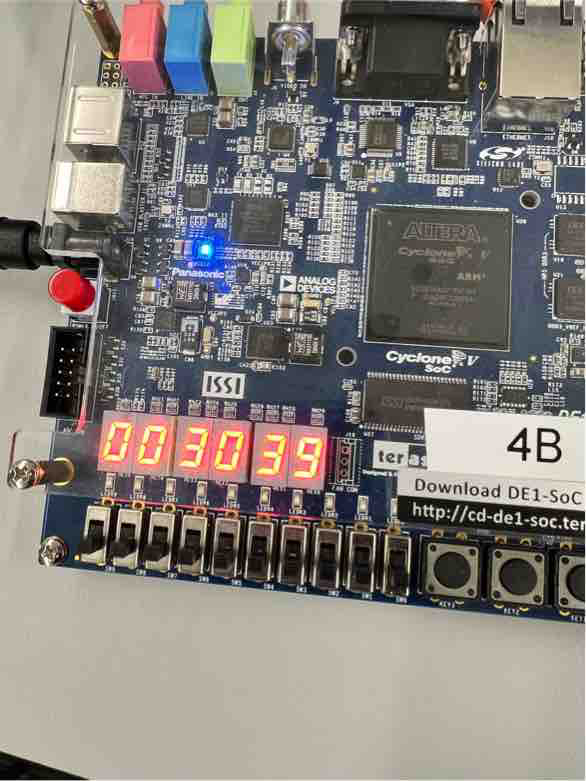
Submit date: 6/17/2022

Due Date: 6/17/2022

**7.1-7.2 Section 1 of Lab 7**

****





**7.3 Section 3 of Lab 7**

**7.3 code:**

DE1SoCfpga.h

#ifndef DE1SOCFPGA\_H

#define DE1SOCFPGA\_H

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <iostream>

const unsigned int LW\_BRIDGE\_BASE = 0xFF200000;

const unsigned int FINAL\_PHYSICAL\_ADDRESS = 0xFFFEC700;

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

const unsigned int SW\_OFFSET = 0xFF200040 - LW\_BRIDGE\_BASE;

class DE1SoCfpga{

protected:

char \*pBase;

int fd;

public:

DE1SoCfpga();

~DE1SoCfpga();

//int ReadAllSwitches();

void RegisterWrite(unsigned int offset, unsigned int value);

int RegisterRead(unsigned int offset);

};

#endif //DE1SOCFPGA\_H

DE1SoCfpga.cpp

#include "DE1SoCfpga.h"

DE1SoCfpga::DE1SoCfpga(){

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

if(fd == -1){

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

exit(1);

}

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

if(virtual\_base == MAP\_FAILED){

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

close(fd);

exit(1);

}

pBase = virtual\_base;

}

DE1SoCfpga::~DE1SoCfpga() {

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

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

exit(1);

}

close(fd);

}

int DE1SoCfpga::RegisterRead(unsigned int offset) {

return \*(volatile unsigned int \*)(pBase + offset);

}

void DE1SoCfpga::RegisterWrite(unsigned int offset, unsigned int value) {

\*(volatile unsigned int \*)(pBase + offset) = value;

}

SevenSegment.h

#ifndef SEVENSEGMENT\_H

#define SEVENSEGMENT\_H

#include "DE1SoCfpga.h"

const unsigned int bit\_values[16] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07, 0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71};

const unsigned int HEX3\_0\_OFFSET = 0xFF200020 - LW\_BRIDGE\_BASE;

const unsigned int HEX5\_4\_OFFSET = 0xFF200030 - LW\_BRIDGE\_BASE;

const unsigned int MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET = 0xDEC600; // Points to LOAD

const unsigned int MPCORE\_PRIV\_TIMER\_COUNTER\_OFFSET = 0xDEC604; // Points to COUNTER

const unsigned int MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET = 0xDEC608; // Points to CONTROL

const unsigned int MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET = 0xDEC60C; // Points to INTERRUPT

const unsigned int letter\_values[9] = {0x0E,0x5C,0x74,0x54,0x00,0x5E,0x5C,0x79,0x00};

class SevenSegment: public DE1SoCfpga{

public:

SevenSegment();

~SevenSegment();

void Hex\_ClearAll();

void Hex\_ClearSpecific(int index);

void Hex\_WriteSpecific(int index, int value);

void Hex\_WriteNumber(int number);

void Write\_Letter(int index);

private:

unsigned int reg0\_hexValue;

unsigned int reg1\_hexValue;

int initialvalueLoadMPCore;

int initialvalueControlMPCore;

int initialvalueInterruptMPCore;

};

#endif //SEVENSEGMENT\_H

SevenSegment.cpp

#include "SevenSegment.h"

SevenSegment::SevenSegment(){

reg0\_hexValue = 0;

reg1\_hexValue = 0;

RegisterWrite(HEX3\_0\_OFFSET,reg0\_hexValue);

RegisterWrite(HEX5\_4\_OFFSET,reg1\_hexValue);

initialvalueLoadMPCore = RegisterRead(MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET);

initialvalueControlMPCore = RegisterRead(MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET);

initialvalueInterruptMPCore = RegisterRead(MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET);

}

SevenSegment::~SevenSegment() {

Hex\_ClearAll();

RegisterWrite(MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, initialvalueLoadMPCore);

RegisterWrite(MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, initialvalueControlMPCore);

RegisterWrite(MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, initialvalueInterruptMPCore);

}

void SevenSegment::Hex\_ClearAll() {

RegisterWrite(HEX3\_0\_OFFSET,0);

RegisterWrite(HEX5\_4\_OFFSET,0);

}

void SevenSegment::Hex\_ClearSpecific(int index) {

if(index == 0){reg0\_hexValue = reg0\_hexValue & 0xFFFFFF00;}

else if(index == 1){reg0\_hexValue = reg0\_hexValue & 0xFFFF00FF;}

else if(index == 2){reg0\_hexValue = reg0\_hexValue & 0xFF00FFFF;}

else if(index == 3){reg0\_hexValue = reg0\_hexValue & 0x00FFFFFF;}

else if(index == 4){reg1\_hexValue = reg1\_hexValue & 0xFFFFFF00;}

else if(index == 5){reg1\_hexValue = reg1\_hexValue & 0xFFFF00FF;}

else{

std::cerr << "Index exceeding 5 or less than 0!" << std::endl;

exit(1);

}

}

void SevenSegment::Hex\_WriteNumber(int number) {

int temp;

if(number >= 0) {temp = number;}

else {temp = 0xFFFFFF+number+1;}

temp = number;

int store[6] = {0,0,0,0,0,0};

int remainder;

int count = 0;

while(temp > 0){

remainder = temp%16;

store[count] = remainder;

count++;

temp = temp/16;

}

reg0\_hexValue = 0;

reg1\_hexValue = 0;

for(int i = 0; i < 6; i++){Hex\_WriteSpecific(i,store[i]);}

}

void SevenSegment::Hex\_WriteSpecific(int index, int value) {

Hex\_ClearSpecific(index);

int temp;

temp = value;

if(index < 4){reg0\_hexValue = reg0\_hexValue | (bit\_values[temp] << 8\*index);}

else{reg1\_hexValue = reg1\_hexValue | (bit\_values[temp] << 8\*(index-4));}

RegisterWrite(HEX3\_0\_OFFSET,reg0\_hexValue);

RegisterWrite(HEX5\_4\_OFFSET,reg1\_hexValue);

}

void SevenSegment::Write\_Letter(int index){

int temp\_low = reg0\_hexValue >> 24;

reg0\_hexValue = reg0\_hexValue << 8 | letter\_values[index];

reg1\_hexValue = reg1\_hexValue << 8 | temp\_low;

RegisterWrite(HEX3\_0\_OFFSET,reg0\_hexValue);

RegisterWrite(HEX5\_4\_OFFSET,reg1\_hexValue);

}

main.cpp

#include <iostream>

#include "SevenSegment.h"

int main(){

/\*

SevenSegment \*display = new SevenSegment;

std::cout << "Program Starting...!" << std::endl;

bool condition = true;

int hex\_value;

while(condition){

std::cout << "Please enter a number for display (in the range -16777216 to 16777215):";

std::cin >> hex\_value;

display->Hex\_WriteNumber(hex\_value);

std::cout << "Do you want to stop? ";

char v;

std::cin >> v;

if(v == 'y'){condition = false;}

}

std::cout << "Terminating...!" << std::endl;

delete display;

return 0;

\*/

SevenSegment \*display = new SevenSegment;

std::cout << "Program Starting...!" << std::endl;

int counter = 100000000; // timeout = 1/(200 MHz) x 200x10^6 = 1 sec

display->RegisterWrite(MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter);

display->RegisterWrite(MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3);

int count = 0;

while(display->RegisterRead(SW\_OFFSET)%2 == 0){

if(display -> RegisterRead(MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0){

display -> RegisterWrite(MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET,1);

display -> Write\_Letter(count);

count++;

if(count == 9){count = 0;}

}

}

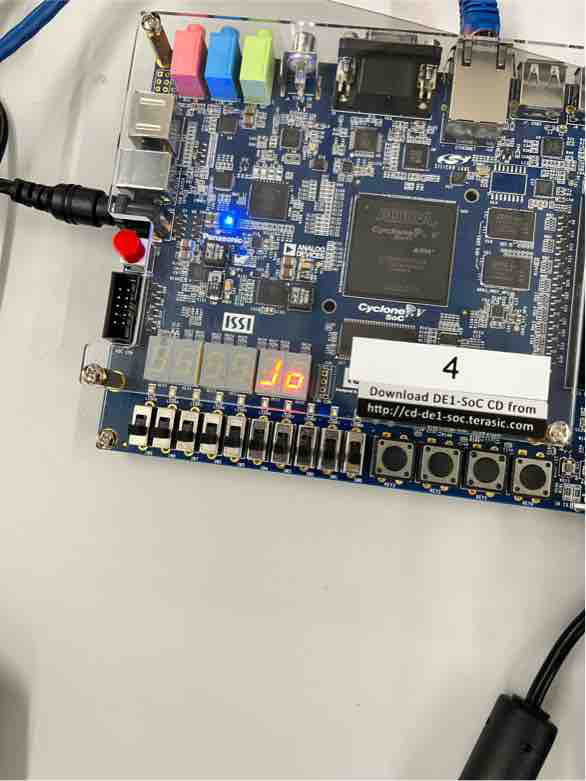
delete display;

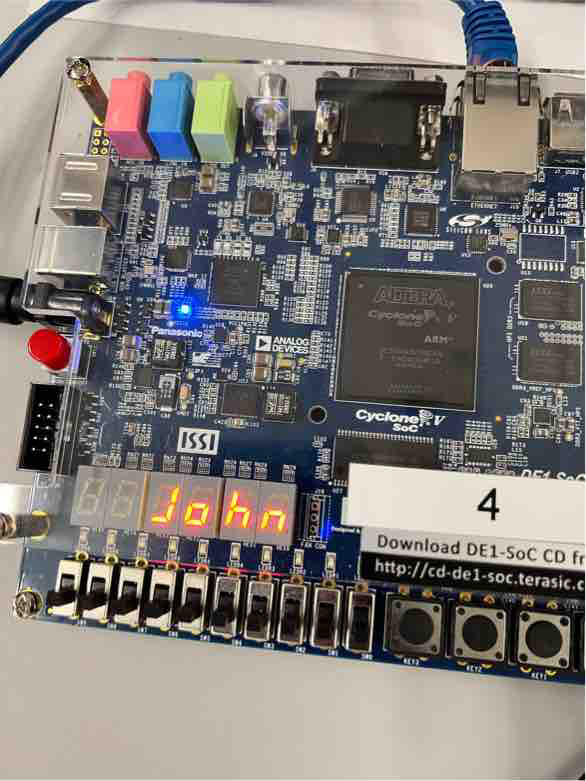
std::cout << "Terminating...!" << std::endl;

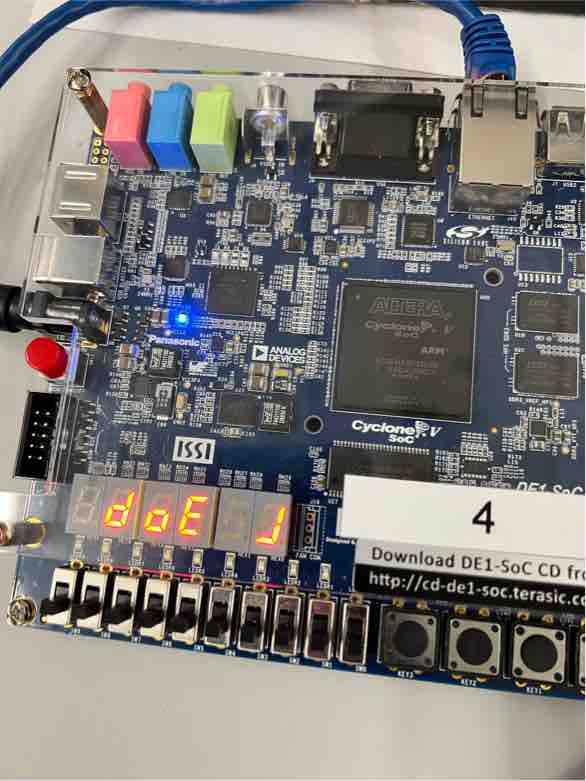
return 0;

}

**Running Result**







**Code Explanation**

In part 3 of the lab, the goal is to make a name rolling to the left using the timer of the DE1SoC board. Different from the prelab assignment where the seven-segment display is supposed to show numbers and letters from A to F, the name display is going to show “J”, “o”, “h”, “n”, “d”, “o”, “E” in order with space in between. Therefore, a new array is made that contains the corresponding hex value for these letters in 7-segment displays.

To make the letters roll to the left, simply change the timer load offset to 100000000 which means the timer will count every 0.5 seconds. After each 0.5 seconds, the while loop in the main.cpp will be called. In the seven-segment class, a member function is constructed, which shifts the hex value by 8 bits in order, and makes the decision to change the hex0value or the hex1value. The loop stops if switch 0 is turned on, this is done by register read.

# References

1. Prof. Julius Marpaung, “*Lab Report Guide*”, Northeastern University, January 6 2020.