**Project 3 Report**

EE 472: Spring 2013

Dylan Armstrong

Xin Chen

Youngjun Han

Michael Molina

Table of Contents

Design Procedure 2

Testing Process 2

Time Estimates 3

Diagrams 4

High-level block diagram 4

TrainCom 4

SwitchControl 5

Current Train 6

SerialCom 7

Schedule 8

Pseudo Code 8

TrainCom 9

SwitchControl 9

CurrentTrain 9

SerialCom 9

Schedule 9

Work Distribution 9

# Design Procedure

Similarly to our approach for project 2, the design for this project made use of many of the design steps, as described in the project 2 description. The basic design process was as follows:

1. Draw all UML diagrams, starting with highest level diagrams
2. Understand the functionality of each task and revise UML diagrams as needed
3. Write pseudo-code for each task, separately
4. Review program logic and control scheme
5. Write code and work through all compile time errors
6. Test and debug code
7. Polish code

While our general design approach, outlined above, was similar to our approach for project 2, the actual implementation of this process was notably different than before. For this project we decided to first understand and develop the initial high-level flow of the program as a group; following that brief, initial work, we divided what we saw as the four major areas of the program up between each group member, as indicated in the list below:

1. Dylan: Serial Com
2. Leo: Hardware interrupts
3. Youngjun: Schedule and linked list
4. Michael: Revise trainCom and combine north/east/west into currentTrain

After all members completed the initial design of their respective module, we integrated everything with the existing project 2. Following this, we implemented all of the remaining code, such as the startup function, removing unnecessary, old code, etc. Finally, we all worked together – intermittently – to complete the code for any of the above areas, before moving to the debug process, which is described in the following section of this report.

# Testing Process

Following after the initial design and coding for the project was completed, as described in the previous section of this report, the team focused its attention to testing and debugging the project. This section describes our testing and debugging process.

The testing and debugging procedure was mostly completed through use of the debugger tool in the IAR embedded workbench. Testing was completed step by step. All steps are listed below:

1. Test global count working properly triggered by ISP
2. Test push-button working properly triggered by ISP
3. Test schedule to see that tasks are being added or removed properly
4. Test as a whole program excluding hyperterm
5. Test as a whole program including hyperterm

The main problem was adjusting the logic for ‘schedule’, which adds and removes tasks. Adding and removing task from the list was not simple. We created specific functions to add or remove a task from the list. Adding and removing functions behave differently, depending on the cases such as empty list, single task list, or removing the first task in the list and etc. For this problem, the team decided to use several if cases to classify all cases and coded each case to act properly. Another problem that arose by schedule function is what parameters to use to let schedule know when to add or remove tasks. The team decided to treat each task as a resource. In other words, we created a flag variable for each task and set it to 1 when the task is to be added, and to 0 when it has to be removed. To make this work as we desired, the team modified the adding and removing function not to attempt to add a task which is already in the list and not to attempt to remove a task which is currently not in the list.

# Time Estimates

|  |  |
| --- | --- |
| Subject | Time spent |
| Design | 3 hours |
| Coding | 3 hours |
| Test / Debug | 10 hours |
| Documentation | 3 hours |

# Diagrams

## High-level block diagram



## TrainCom



## 

## SwitchControl



## Current Train



## SerialCom



## Schedule



# Pseudo Code

**header.h**

#include header files for drivers libraries needed in this project;

declare error routine function prototype;

define new variable type bool;

declare function prototypes for the entire project;

define struct TCB;

create a struct for holding task trainCom’s data as trainComData;

create a struct for holding task switchControl’s data as switchControlData;

create a struct for holding task schedule’s data as scheduleData;

create a struct for holding task serialCom’s data as serialComData;

create a struct for holding task currentTrain’s data as currentTrainData;

**constantDefinitions.h**

#define statements of constant definitions for the entire project;

**main.c**

#include "header.c" and "constantDefinition.h"

unsigned char timerState; // create a flag for timer interrupt

unsigned char State0 = 0; // create flags for button interrupt

unsigned char State1 = 0;

unsigned char State2 = 0;

unsigned char State3 = 0;

bool north = FALSE; // initialize global variables

bool west = FALSE;

bool east = FASLE;

bool south = FALSE;

bool trainPresent = FALSE

unsigned int trainSize = FALSE;

unsigned int traversalTime = 0;

unsigned int gridLockDelay = 0;

bool gridLock = FALSE;

unsigned int globalCount = 0;

unsigned int trainComFlag = 0;

unsigned int switchControlFlag = 0;

unsigned int serialComFlag = 0;

unsigned int currentTrainFlag = 0;

char gridLockMessage[]; // initialize global strings for gridlock alarm

char trainDirectionMessage[]; // and train info

char trainSizeMessage[]

char trainFromMessage[];

void main(void)

{

call startup\_function(); // initialize timer, pushbutton, serialCom, speaker

initialize trainCom TCB;

initialize serialCom TCB;

initialize switchControl TCB;

initialize currentTrain TCB;

initialize trainComData TCB; // initialize all the fields inside the structs

initialize serialComData TCB;

initialize switchControlData TCB;

initialize currentTrainData TCB;

initialize scheduleData TCB;

call schedule; // start running the linked list

}

void UARTIntHandler(void) // ISR for serialCom

{

unsigned long ulStatus;

Get the interrrupt status;

Clear the asserted interrupts;

while(UARTCharsAvail(UART0\_BASE))

read the next character from the UART and write it back to the UART;

}

void IntGPIOe(void) // ISR for buttons

{

clear the interrupt to avoid continously looping;

Set State0,1,2, or 3 accordingly when a button is pressed;

// trigger button hardware interrupt

flip the polarity of the results above;

set trainComFlag = 1;

// when a button is pressed, we need to add trainCom to generate a train

}

void IntTimer0(void)

{

Clear the interrupt to avoid continuously looping;

timerState = 1; // trigger the interrupt

}

**trainCom.c**

#include “header.h” and “constantDefinitions.h”;

void trainCom(void\* data)

{

re-cast the passed in void pointer to trainComData struct pointer type;

local variables int direction = 0; int TrainSize = 0;

if (there is no train currently passing or gridlock going on)

{

If (State0Ptr == 1) // up button pressed

Call builString to build train from north message;

Reset State0Ptr = 0; // reset interrupt

Else if (State1Ptr == 1) // down button pressed;

Call buildString to build train from south message;

Reset State1Ptr = 0;

Else if (State2Ptr ==1) // left button pressed;

Call buildString to build train to east message;

Reset State2Ptr = 0;

Else if (State3Ptr ==1) // right button pressed;

Call buildString to build train to west message;

Reset State3Ptr = 0;

TrainSize = randomInteger(2,9); // generate train size

Update trainSizeP = TrainSize;

Direction = randomInteger(0,3); // generate train to direction

If (direction == WEST)

westP = TRUE;

call buildString to build train to west message;

if (direction == NORTH)

northP = TRUE;

call buildString to build train to north message;

if (direction == EAST)

eastP = TRUE;

call buildString to build train to east message;

if (direction == SOUTH)

southP = TRUE;

call buildString to build train to south message;

call buildString to build train size message;

write TrainSize into trainSizeMessageP;

print trainToDirectionMessage;

print trainSizeMessage;

print trainFromDirectionMessage;

Reset trainPresent;

Reset trainComFlagP = 0;

}

**switchControl.c**

#include “header.h” && “constantDefinitions.h”

Void switchControl(void\* data)

{

Re-cast the passed in void pointer to switchControlData struct pointer type

Set gridLockChecked = TRUE;

Int gridLockNum; // this random number is to determine whether there is a gridlock

If (gridLockDelay == 0 && traversalTime == 0 && trainPresent == false) // no current train passing

{ // no current gridlock going on

gridLockNum = randomInteger(-2,2);

if (gridLockNum < 0) // when there is a gridlock generated

{

gridlock = TRUE;

trainPresent = FALSE;

gridLockDelay = -0.2\*gridLocknum\*120; // compute gridlock delay

call buildString to build gridlockMessage;

print gridLockMessage;

set serialComFlag = 1; // raise the serialComFlag as an interrupt to add serialCom task

} else { // when there is no gridlock generated

traversalTime = 0.1\* (trainSize) \* 120; // compute the train traversal time

} else if { (gridlock == TRUE) // if we are currently in gridlock

gridlockDelayTime decrement by 1;

if (gridLockDelayTime % 2 ==0) {

// this algorithm gives a flashing rate of 0.5 sec ON 0.5 sec OFF

print gridLockMessage;

} else

print empty string;

if (gridLockDelayTime == 0) { // this happens when gridlock is finished

set gridlock == FALSE;

trainPresent == TRUE;

call buildString to modify the global gridLockMessage to become an empty string;

print the empty string to clear the gridlockMessage;

}

set serialComFlag = 1; // when gridlock is finished we need to update the information

// on HyperTerm

} else { // if we enters here, there is currently a train passing through

decrement traversalTime;

if (traversalTime == 0) { // train is passed

trainPresent = FALSE;

trainSize = FALSE;

north = FALSE;

east = FALSE;

west = FALSE;

south = FALSE; // when the train is passed, reset all the global variables

clear OLED screen

turn off sound

set serialComFlag = 1; //when train is passed, serialCom info needs to be updated

}

switchControlFlag = 0; // remove switchControl from the list untile needed again

}

**currentTrain.c**

#include "header.h" and "constantDefinitions.h"

void eastTrain(void); // helper function prototypes

void westTrain(void);

void northTrain(void);

void southTrain(void);

static char\* trainDirectionMesssage; // local variable pointers

static char\* trainSizeMessage; // three pointers pointing to char array

static char\* trainFrom; // for display message on OLED

static unsigned int remainingtraversalTime;

void currentTrain(void\* data) {

re-cast passed in void pointer to currentTrainData pointer type

trainDirectionMessage = trainDirectionMessageP;

trainSizeMessage = trainSizeMessageP

trainFrom = trainFromP;

remainningTraversalTime = traversalTimeP; // define local variable and variable pointers

if (trainPresent == TRUE && remainingTraversalTime >0)

// there is currently a train passing

{

if (eastP == TRUE)

call eastTrain();

ease if (westP == TRUE)

call westTrain();

else if (northP == TRUE)

call northTrain();

else if (southP == TRUE)

call southTrain();

currentTrainFlagP = 0; // remove currentTrain from the list

}

}

void eastTrain()

{

creates an integer array for west train sound pattern

if (eastSoundArray[remainingTraversalTime%26] ==1) // go through the array repeatedly

turn on the sound

else

turn off the sound

if (remainingTraversalTime %8 < 4)

// this algorithm gives a flashing rate of 2 sec ON 2 sec OFF

{

print trainDirectionMessage;

print trainSizeMessage;

print trainFrom;

} else

clear the screen;

}

void westTrain()

{

creates an integer array for west train sound pattern

if (westSoundArray[remianingTraversalTime % 14] ==1)

// go through the array repeatedly

turn on the sound;

else

turn off the sound;

if (remainingTraversalTime % 4 <2) // a flashing rate of 1 sec ON 1 sec OFF

{

print trainDirectionMessage;

print trainSizeMessage;

print trainFrom;

} else

clear the screen;

}

void northTrain()

{

creates an integer array for north train sound pattern

if (northSoundArray[remainingTraversalTime % 20] ==1) {

// traverse the array element repeatedly

turn on the sound;

} else

turn off the sound;

if(remainingTraversalTime % 6 <3) // a flashing rate of 1.5 sec ON ,1.5 sec OFF

print trainDirectionMessage && trainSizeMessage && trainFromMessage;

else

clear the screen;

}

void southTrain()

{

creates an integer array for south train sound apttern

if (southSoundArray[remainingTraversalTime % 24 ==1) // traverse the array element

turn on the sound;

else

turn off the sound;

if (remainingTraversalTime % 4 < 2) // a flashing rate of 1 sec ON 1 sec OFF

print trainDirectionMessage && trainSizeMessage && trainFromMessage;

else

clear the screen;

}

**schedule.c**

#include "header.h" and "constantDefinitions.h"

TCB\* queueHead;

TCB\* currentPtr; // declare two local TCB pointers

void addNode(TCB\* node);

void removeNode(TCB\* node); // two helper function prototypes

void schedule(void\* data)

{

recast the passed in void pointer to schedulData type

queueHead = NULL; // initialize queueHead and currentPtr

currentPtr = queueHead; // the linked list is initially empty

int localTime; // three local function variables

int digit;

char a[COUNTER\_STRING\_SIZE] = INITIAL\_COUNTER\_STRING;

// char array for displaying GC

print glocalCount on the screen;

while(1)

{

if (timerState ==1 ) // timer interrupt

increment globalCount by 1;

digit = 0;

localTime = globalCount; // update localTime for local function time reference

while(localTime != 0) {

a[5-digit] = (localTime % 10) + ASCII\_ZERO;

digit = digit +1;

localTime = localtime / 10;

} // this while loop writes globalCount into the char array

print the globalCount;

if (gridLock == TRUE)

set switchControlFlag = 1;

// when there is a gridLock we need to add switchControl to manage gridLock

else if (trainPresent == TRUE) {

switchControlFlag = 1; // when there is a train passing, we need both

currentTrainFlag = 1 // switchControl and currentTrain in the list

}

timerState = 0; // reset timer interrupt

}

if (currentPtr != NULL) {

call the current task in the list;

if (next node is NULL) // we reach the end of the list, need to go back to the head

moves currPtr to point to queueHead;

else if (queueHead != NULL)

moves currentPtr to point to the next node;

if (trainComFlag == 1) // the below if statements check whether to

add trainCom; // add or remove tasks from the list

if (trainComFlag ==0)

remove trainCom;

if (switchControlFlag == 1)

add switchControl;

if (switchControlFlag ==0)

remove switchControl;

if (currentTrainFlag == 1)

add currentTrain;

if (currentTrainFlag ==0)

remove currentTrain;

if (serialComFlag == 1)

add serialCom;

if (serialComFlag ==0)

remove serialCom;

}

void addNode(TCB\* node) {

TCB\* currentNode = queueHead; // creates a pointer pointing to queueHead

bool nodeExists = FALSE;

while(currentNode != NULL) { // check if the given node is already in the list

if current Node == node

nodeExists = TRUE;

currentNode = next node;

}

points the currentNode back to queueHead;

if (the node is not in the list yet)

if the list is currently empty

set queueHead to node;

else {

while( currentNode's next node is not NULL) // traverse to the end of list

currentNode = currentNode->next;

add node to the end of the list;

}

}

void removeNode(TCB\* node) {

TCB\* currentNode = queueHead; // creates a pointer pointing to queueHead

int verifyNodeExist = 0;

while(currentNode is not NULL) { // check to see if node is in the list

if currentNode == node

verifyNodeExists = 1; // node is in the list

currentNode = currentNode->next;

}

if (node is in the list) {

if (queueHead is NULL)

do no thing;

else if queuehead is node && node-> is NULL { // this means the size of list is 1

queueHead = NULL; // set queueHead and currentPtr to NULL

currentPtr = NULL;

} else { // the size of list is at least 2

if queueHead is node

set queueHead to the second node in the list;

set node's prev to NULL;

else if (node is at the end of the list)

set node's previous node's next to NULL;

else

node -> prev -> next = node -> next;

node -> next -> prev = node -> prev;

}

}

**serialComPseudo**

#include "header.h" and "constantDefinitions.h"

voidUARTsend(const unsigned char \*pucBuffer, unsigned long ulCount)

{

while(ulCount--)

write the next character to the UART;

}

void serialCom(void\* data)

{

recast the void pointer to serialComData type

call UARTSend function to clear the screen;

call UARTSend function to display gridlockMessage && trainSizeMessage

&& trainFromMessage && trainToMessage

set serialComFlag = 0;

}

**startup\_function.c**

#include "header.h" and "constantDefinitions.h"

void startup\_function(void)

{

Set the clocking to run directly from the crystal;

Initialize the OLED display;

Clear the default ISR handler and install IntTimer0 as the handler; // initialization for timer

Enable Timer 0;

Configure Timer 0 and set the timebase to 0.5 second;

Enable interrupts for Timer0 and activate it;

Clear the default ISR handler and install IntGPIOe as the handler; // initialization for buttons

Enable GPIO port E, set pin 0,1,2,3 as inputs;

Activate the pull-up on GPIO port E

Configure GPIO port E as triggering on falling edges

Enable interrupts for GPIO port E

IntEnable(INT\_GPIOE);

Set PWM Divide Ratio to 1; // initialization for speaker

Set Device: PWM0 Enabled;

Set GPIO Port: G Enabled;

Tell Port G, Pin 1, to take input from PWM 0;

Set a SPEAKER\_FREQUENCY Hz frequency as u1Period;

Configure PWM0 in up-down count mode, no sync to clock;

Set u1Period (SPEAKER\_FREQUENCY) as the period of PWM0;

Set PWM0, output 1 to a duty cycle of DUTY\_CYCLE;

Activate PWM0;

// initialization for serialCom

Clear the default ISR handler and install UARTIntHandler as the handler;

Enable the peripherals;

Enable processor interrupts;

Set GPIO A0 and A1 as UART pins;

Configure the UART for 115,200, 8-N-1 operation;

Enable the UART interrupt;

}

# Work Distribution

|  |  |
| --- | --- |
| Task | Names |
| Design | Michael, Leo, Young, Dylan |
| UML Diagrams | Michael |
| pseudocode | Leo |
| trainCom | Michael |
| switchControl | Dylan |
| North/east/west train | Dylan, Michael |
| schedule | Young |
| Test / Debug | Michael, Leo, Young, Dylan |
| Documentation | Michael, Leo, Young, Dylan |