**Traffic Light System**

CS 435 Embedded Systems Final Project

Samir Asfirane  
Erik Kalan  
Javier Valerio

December 15th, 2016

Final Report

1. ABSTRACT OR EXECUTIVE SUMMARY *(200 – 300 words)*

The purpose of this project is to implement and to simulate a 4-way traffic control system using commercial embedded components, applying digital electronic circuit concepts as well as software engineering principles, and designing our application using an ARM-based ST Nucleo Board. The basis of such project is resembling a real-time traffic-signal system with the aid of embedded components such as LEDs, buttons, speakers, potentiometer, and temperature sensor.

The first phase of the project involved assembling the circuits and wiring, specifying input and output ports and interconnecting these components using a single breadboard and a microcontroller. The final phase of our project consisted in compiling and programming our code into the microcontroller to test our design.

By identifying the limitations of our embedded components, we modified our project’s specifications as needed. In the design and analysis of our embedded application, we will discuss how we reduced the number of I/O ports used in order to accommodate more components and thus, more features. Additionally, through demonstration of our application in class, we highlighted the features of our project and provided a more accurate hindsight of what goes behind the scenes.

Our application is more like a responsive traffic control system where the different input/output ports connected to our microcontroller reflect the current conditions of the present traffic in 4 lanes. It then uses the incoming data to display on the PC the current state of our intersection.

1. INTRODUCTION *(clearly stating aims and objectives and specification of the activity)*

The primary objective of our embedded project is simulating a 4-way traffic control system using basic embedded components and practical concepts learned in class. Our traffic system will behave and react to different inputs (for example, when a sensor changes temperature, or a button is pressed). We aim to get all the components to work collectively and simultaneously using threads.

This report outlines and emphasizes how our team planned, designed, and implemented an efficient traffic control system such that traffic flows through a fairly amount of time while keeping the waiting times of the other lights at a minimum.

1. CONCEPTUAL DESIGN OF THE SYSTEM *(interpretation of the specification, top level flow-charts, associated supporting analysis, etc.)*

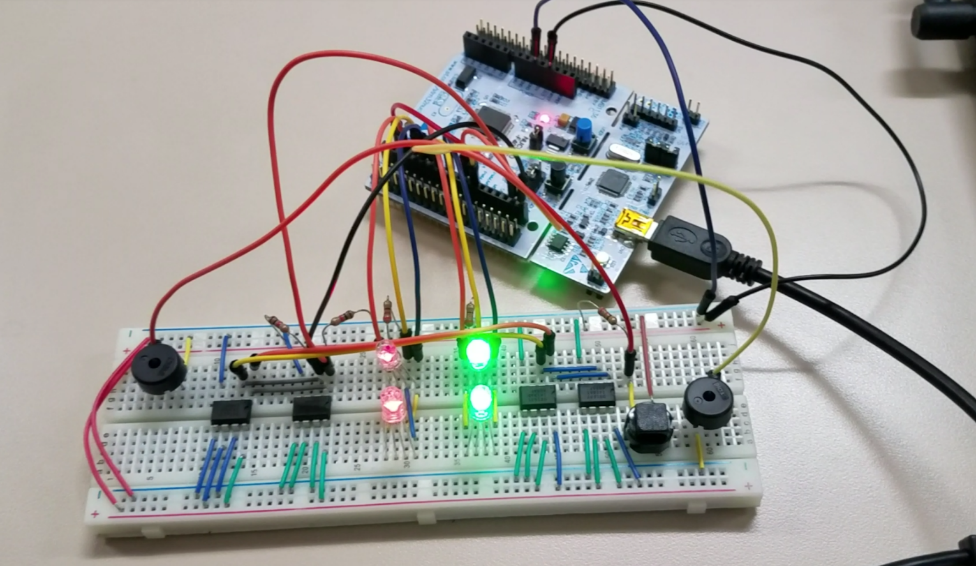
During the initial stage of the design process, our primary focus was finding the most suitable technological solution for a Traffic Control System that incorporated many features and most of the embedded components used throughout the labs in class. The microcontroller operates on a 3.3 V battery voltage which is connected to the top row (indicated with the ‘+’ sign) in our breadboard. The horizontal row of our breadboard (indicated with the ‘-’ sign) is connected to one of the GND ports in our microcontroller.

Next, we have two pairs of LEDs, each of them running in parallel (simulating a 4-way intersection) and each being connected to an input port in our board (pins PB\_3, PB\_4, PB\_5 for the first pair of LEDs; PB\_10, PA\_9, PA\_8 for the second one). The first pair of LEDs controls the traffic going north- and southbound while the second pair controls the traffic going east- and westbound. As indicated in one of the first labs for this course, pin 2 from each pair is connected to a resistor, and then ground.

Our code implements a button, which invokes the *traffic\_interruption ()* when it is pressed down. This button triggers an interrupt to the current traffic. When pressed down, all 4 LEDs start blinking red for a specific amount of time (similar to for instance, when the traffic lights at a given intersection stop working). This button is connected to GND and to input pin PA\_10 on our board.

Additionally, we use four DS1631 temperature sensors, where each sensor controls the amount of traffic going through the intersection; in other words, 1 sensor for going north-south, 1 going south-north, 1 going one east-west, and 1 more going west-east. This illustrates almost a race condition where the sensor with the highest temperature will cause its pair of LEDS to turn green and the other pair turn red for a specific amount of time. A higher temperature indicates heavier traffic. To use them, we set up the address for the sensor by connecting input pins 5, 6, 7 to ground and thus, the temperature address will be 0x90. Two 1kΩ pull-up resistors are also used to keep the SDA and SCL lines in a logic-high while the bus is idle.

1. DETAILED DESIGN AND IMPLEMENTATION OF THE CODE *(flow charts of major routines, application code)*.





**START**

Infinite loop – continue reading temperatures

If button is pressed

Run  
*traffic\_interrupt ()*

Run threads in parallel

**NO**

**YES**

Is *temp1* < *temp2*

Run *routineBLight1*&& *routineBLight2*

Run *routineALight1*&& *routineALight2*

Run   
*pedSoundBlight1 ()*&& *pedSoundBlight2 ()*

Run   
*pedSoundAlight1 ()*&& *pedSoundAlight2 ()*

1. DISCUSS THE MAIN CHALLENGES OF IMPLEMENTATION

One of our initial challenges was planning how many components our project would need without us running out of I/O pins. At first, we brainstormed about what other features we could add after building the initial design but given the limited number of free pins, we modified our approach. Similarly, another challenge was using four separate temperature sensors at the same time but this was later resolved by giving/accessing each sensor using different addresses (0x00, 0x02, 0x04, and 0x06).

1. TESTING (DESCRIPTION OF TESTS, RESULTS)

Our testing was carried out in a sequential manner; this ensured that each component behaved as expected as specified and implemented in our code and ports wiring. We first started testing only the 2 pairs of LEDs without Threads functionality, aiming that only one pair was lit green at a time and when this was accomplished, Threads were added to it. Temperature sensors were up next. We initially started with one sensor (initially, we planned to use a single temperature sensor for both pair of LEDs) but later in the design, 3 more temperature sensors were added. We tested them first without threads; shortly after, with Threads. The speakers were added next to simulate pedestrian crossing. A button was then added to simulate a broken traffic control system (all LEDs turned into blinking LEDs for a given period of time). By using this waterfall model in our project, we spent less time testing and debugging since each feature was first tested in isolation and then collectively. Using the PC was also really helpful when debugging because it helped us detect the current state of each of the components.

1. SUMMARY

At the end of this project, we were able to assemble a project which involved the many features learned on labs during this course. Most importantly, not only did we implement the initial basic design but we also added more logic and components to it. We applied a practical experience to design, build, and especially troubleshoot the different portions of our application code. Through experiments and trial and error, we obtained a better understanding of the principles of digital and embedded systems. Even though the number of components used in this project were not many, our code and design demonstrated a running, error-free embedded application which met all of our objectives.

1. REFERENCE/BIBLIOGRAPHY

Ahmadinia, Ali. “[11 - Real Time Operating SystemFile](file:///C:\Users\Samir\Downloads\11%20-%20Real%20Time%20Operating%20SystemFile)” [PowerPoint slides]. Retrieved from <https://cc.csusm.edu/>.

Ahmadinia, Ali. “[Lab 3 - Digital Input and OutputFile](https://cc.csusm.edu/mod/resource/view.php?id=133024)” [PowerPoint slides]. Retrieved from <https://cc.csusm.edu/>.

Ahmadinia, Ali. “[Lab 4 - Mbed-Digital-IOFile](file:///C:\Users\Samir\Downloads\Lab%204%20-%20Mbed-Digital-IOFile)” [PowerPoint slides]. Retrieved from <https://cc.csusm.edu/>.

Ahmadinia, Ali. “[Lab 5 - Analog Input and OutputFile](https://cc.csusm.edu/mod/resource/view.php?id=75173)” [PowerPoint slides]. Retrieved from <https://cc.csusm.edu/>.

Ahmadinia, Ali. “[Lab 6 - Serial CommunicationFile](https://cc.csusm.edu/mod/resource/view.php?id=75175)” [PowerPoint slides]. Retrieved from <https://cc.csusm.edu/>.

Ahmadinia, Ali. “[Lab 7 - RTOSFile](https://cc.csusm.edu/mod/resource/view.php?id=75177)” [PowerPoint slides]. Retrieved from <https://cc.csusm.edu/>.

1. CODE LISTINGS

/\*----------------------------------------------------------------------------

Final Project - TRAFFIC SIGNAL

PROGRAMMING USING MBED API- Samir Asfirane, Erik Kalan, Javier Valerio.

------------------------------------------------------------------------

Final Project

\*----------------------------------------------------------------------------\*/

**#include** "mbed.h"

**#include** "rtos.h"

**#include** "DS1631.h"

**#include** "pindef.h"

**#include** <stdio.h>

**#include** <time.h>

DigitalIn btn\_intrpt(*PA\_10*);

DigitalOut myledRED1(*PB\_3*); /// Connect LED cathode to pin PB\_3

DigitalOut myledGREEN1(*PB\_4*); /// Connect LED cathode to pin PB\_4

DigitalOut myledORANGE1(*PB\_5*); /// Connect LED cathode to pin PB\_5

DigitalOut myledRED2(*PB\_10*); /// Connect LED cathode to pin PB\_10

DigitalOut myledGREEN2(*PA\_9*); /// Connect LED cathode to pin PA\_9

DigitalOut myledORANGE2(*PA\_8*); /// Connect LED cathode to pin PA\_8

//I2C interface

I2C temp\_sensor(I2C\_SDA, I2C\_SCL);

Serial pc(UART\_TX, UART\_RX);

PwmOut speaker1(*PA\_7*); /// Connect speaker to pin PA\_7 (this speaker sounds when light1 is green)

PwmOut speaker2(*PB\_6*); /// Connect speaker to pin PB\_6 ( one of the led should be changed so the speaker works) this speaker sounds when light2 is green

// Struct for temp sensor

**typedef** **struct** {

**int** id;

**int** addr;

**int** temp;

} TempData;

**float** temp1 = 0, temp2 = 23.6; // two floats represeting each the temperatures of the sensors

Mutex light1\_mutex; // mutex for traffic light 1

Mutex light2\_mutex; // mutex for traffic light 2

Mutex sound\_mutex; // mutex for pedestrian signal

Mutex serial\_mutex; // mutex for serial transmision

//I2C address of temperature sensor DS1631

**const** **int** temp\_addr = 0x90;

**char** cmd[] = { 0x51, 0xAA };

**char** read\_temp[2];

**int** seconds = 10;

time\_t endwait = **time** (NULL) + seconds ;

// function that takes a float argument and turns LED GREEN

// for whatever amount of time passed as argument

**void** **lightGREEN1**(**float** time)

{

myledGREEN1 = 1;

wait(time);

myledGREEN1 = 0;

}

// function that takes a float argument and turns LED orange(blue)

// for whatever amount of time passed as argument

**void** **lightORANGE1**(**float** time)

{

myledORANGE1 = 1;

wait(time);

myledORANGE1 = 0;

}

// function that takes a float argument and turns LED red

// for whatever amount of time passed as argument

**void** **lightRED1**(**float** time)

{

myledRED1 = 1;

wait(time);

myledRED1 = 0;

}

// function that takes a float argument and turns LED GREEN

// for whatever amount of time passed as argument

**void** **lightGREEN2**(**float** time)

{

myledGREEN2 = 1;

wait(time);

myledGREEN2 = 0;

}

// function that takes a float argument and turns LED orange(blue)

// for whatever amount of time passed as argument

**void** **lightORANGE2**(**float** time)

{

myledORANGE2 = 1;

wait(time);

myledORANGE2 = 0;

}

// function that takes a float argument and turns LED red

// for whatever amount of time passed as argument

**void** **lightRED2**(**float** time)

{

myledRED2 = 1;

wait(time);

myledRED2 = 0;

}

// routine A for the thread on the first light

**void** **routineALight1**()

{

// routine that light red for 10 seconds, orange for 2 and then red for 2 seconds

light1\_mutex.lock();

**if** (temp1 >= temp2) {

lightGREEN1(10.0);

lightORANGE1(2.0);

lightRED1(5.0);

}

light1\_mutex.unlock();

}

// routine A for the thread on the second light

**void** **routineALight2**()

{

// routine that light red for 12 seconds, green for 3 and then orange for 2 seconds

light2\_mutex.lock();

**if** (temp1 >= temp2) {

lightRED2(12.0);

lightGREEN2(3.0);

lightORANGE2(2.0);

}

light2\_mutex.unlock();

}

// routine B for the thread on the first light

**void** **routineBLight1**()

{

// routine that light red for 12 seconds, green for 3 and then orange for 2 seconds

light1\_mutex.lock();

**if** (temp1<temp2) {

lightRED1(10.0);

lightGREEN1(3.0);

lightORANGE1(2.0);

lightRED1(2.0);

}

light1\_mutex.unlock();

}

// routine B for the thread on the second light

**void** **routineBLight2**()

{

// routine that light red for 10 seconds, Orange for 2 and then red for 5 seconds

light2\_mutex.lock();

**if** (temp1<temp2) {

lightGREEN2(8.0);

lightORANGE2(2.0);

lightRED2(7.0);

}

light2\_mutex.unlock();

}

// A sound signal for pedestrian to cross takes two arguments a float

// for the time it needs to stay on and a speaker to use for the sound

**void** **pedestrianSound**(**float** time, PwmOut speaker)

{

// lock the mutex for the pedestrian signal

sound\_mutex.lock();

**while** (time>0) {

**int** i = 0;

// generate a short 150Hz tone using PWM hardware output

// something like this can be used for a button click effect for feedback

**for** (i = 0; i<10; i++) {

speaker.period(1.0 / 150.0); // 500hz period

speaker = 0.01; //1% duty cycle - High range volume

wait(.02);

speaker = 0.0; // turn off audio

wait(0.5);

time = time - (**float**)0.5;

}

}

// unlock the mutex for the pedestrian signal

sound\_mutex.unlock();

}

// runs pedestrian signal for a while then waits for a time

// before running again in an infinite loop

**void** **pedSoundAlight1**(**void** **const** \*args)

{

// routine that light red for 10 seconds, Orange for 2 and then red for 5 seconds

**if** (temp1 >= temp2) {

**while** (**true**) {

pedestrianSound(10.0, speaker1);

wait(7.0);

Thread::*wait*(200);

}

}

}

// runs pedestrian signal for a while then waits for a time

// before running again in an infinite loop

**void** **pedSoundAlight2**(**void** **const** \*args)

{

// if temp1 >= temp2 means light one is green for 10 seconds

**if** (temp1 >= temp2) {

**while** (**true**) {

wait(12.0);

pedestrianSound(3.0, speaker2);

wait(2.0);

Thread::*wait*(200);

}

}

}

// runs pedestrian signal for a while then waits for a time

// before running again in an infinite loop

**void** **pedSoundBlight1**(**void** **const** \*args)

{

**if** (temp1 < temp2) {

**while** (**true**) {

wait(10.0);

pedestrianSound(3.0, speaker1);

wait(4.0);

Thread::*wait*(200);

}

}

}

// runs pedestrian signal for a while then waits for a time

// before running again in an infinite loop

**void** **pedSoundBlight2**(**void** **const** \*args)

{

// if temp1 >= temp2 means light one is green for 10 seconds

**if** (temp1<temp2) {

**while** (**true**) {

pedestrianSound(8.0, speaker2);

wait(9.0);

Thread::*wait*(200);

}

}

}

// runs routine A for light 1 in a loop

**void** **runRoutineALight1**(**void** **const** \*args)

{

// routine that light red for 10 seconds, Orange for 2 and then red for 5 seconds

**while** (**true**) {

routineALight1();

Thread::*wait*(200);

}

}

// runs routine A for light 2 in a loop

**void** **runRoutineALight2**(**void** **const** \*args)

{

// routine that light red for 10 seconds, Orange for 2 and then red for 5 seconds

**while** (**true**) {

routineALight2();

Thread::*wait*(200);

}

}

// runs routine B for light 1 in a loop

**void** **runRoutineBLight1**(**void** **const** \*args)

{

// routine that light red for 10 seconds, Orange for 2 and then red for 5 seconds

**while** (**true**) {

routineBLight1();

Thread::*wait*(200);

}

}

// runs routine B for light 2 in a loop

**void** **runRoutineBLight2**(**void** **const** \*args)

{

// routine that light red for 10 seconds, Orange for 2 and then red for 5 seconds

**while** (**true**) {

routineBLight2();

Thread::*wait*(200);

}

}

// temperature thread reads the temperature every second and updates the temp1

**void** **temp\_thread**(**void** **const** \*args)

{

//write your code here

**while** (1) {

//Sensor temperature reading the temperature every second

wait(1.0);

temp\_sensor.write(temp\_addr, &cmd[0], 1);

temp\_sensor.write(temp\_addr, &cmd[1], 1);

temp\_sensor.read(temp\_addr, read\_temp, 2);

temp1 = (**float**((read\_temp[0] << 8) | read\_temp[1]) / 256);

temp\_sensor.write(temp\_addr + 6, &cmd[0], 1);

temp\_sensor.write(temp\_addr + 6, &cmd[1], 1);

temp\_sensor.read(temp\_addr + 6, read\_temp, 2);

temp1 += (**float**((read\_temp[0] << 8) | read\_temp[1]) / 256);

temp1 /= 2;

temp\_sensor.write(temp\_addr + 2, &cmd[0], 1);

temp\_sensor.write(temp\_addr + 2, &cmd[1], 1);

temp\_sensor.read(temp\_addr + 2, read\_temp, 2);

temp2 = (**float**((read\_temp[0] << 8) | read\_temp[1]) / 256);

temp\_sensor.write(temp\_addr + 4, &cmd[0], 1);

temp\_sensor.write(temp\_addr + 4, &cmd[1], 1);

temp\_sensor.read(temp\_addr + 4, read\_temp, 2);

temp2 += (**float**((read\_temp[0] << 8) | read\_temp[1]) / 256);

temp2 /= 2;

serial\_mutex.lock();

pc.printf ("t1: %f\n\r", temp1);

pc.printf ("t2: %f\n\r", temp2);

serial\_mutex.unlock();

wait(2.0);

Thread::*wait*(600);

}

}

**void** **traffic\_interrupt** ( )

{

time\_t endwait = **time** (NULL) + seconds ;

**if** (!btn\_intrpt) {

pc.printf ("State: %s\n\r", "Interrumpted");

**while** (**time** (NULL) < endwait) {

myledRED1 = 1;

myledRED2 = 1;

wait (0.6);

myledRED1 = 0;

myledRED2 = 0;

myledGREEN1 = 0;

myledGREEN2 = 0;

myledORANGE1 = 0;

myledORANGE2 = 0;

wait (0.6);

}

}

}

**int** **main**()

{

pc.printf ("Starting %s\n\r", "Application");

Thread tempThread (temp\_thread); // thread that will calculate the temperature every second

Thread firstlight1 (runRoutineALight1); // thread that will run light 1 if temp1>=temp2

Thread soundAlight1 (pedSoundAlight1);

Thread secondlight2 (runRoutineALight2); // thread that will run light 2 if temp1>=temp2

Thread soundAlight2 (pedSoundAlight2);

Thread firstlight3 (runRoutineBLight1); // thread that will run light 1 if temp1<temp2

Thread soundBlight1 (pedSoundBlight1);

Thread secondlight4 (runRoutineBLight2); // thread that will run light 2 if temp1<temp2

Thread soundBlight2 (pedSoundBlight2);

**while** (1) {

traffic\_interrupt ();

}

}