ECE3375 Project: Seasonal Temperate Ambient Light

Nicholas Pysklywec: 250196085

April 16, 2022

Contents

1 Problem Definition

1.1 Problem Description and Specifications

The proposed system will determine the temperature of the room and assign lighting equivalent to the current season combined with temperature data. Specifications to be met by such a system are as follows:

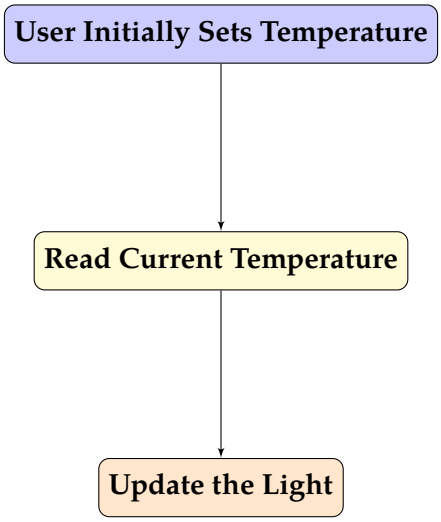
- System can Read Environment Temperature Accurately
- System can Convert Temperature to Appropriate RGB Colour Value
- System can Update Colour based On Environment at Continuous Intervals

1.2 Consumer Impact

An ambient temperature light could have a positive impact on society. Such a device provides a no-worry lighting solution to people who do not want to worry about choosing the correct specific ambient light colour for seasons, in addition to providing a very nice warm and practical range of colours to the user environment. A seasonal light would have use in patios, decks, and backyards and would be very suited to weather patterns seen in Canada. The device provides an entertaining functionality as users can get a sense of the outdoor environment from the colour of the light rather than reading a numeric value.

2 Functional Description

The user will need to configure the system season via ports on the DE10 board. After this occurs, the user interaction is completed. The light will now sense the temperature of the environment, and will combine this sensed temperature with the seasonal input by the user. The light will adjust to a specific ambient colour to fit the combined season and temperature. Suppose the environment is 10 degrees and the season is winter. The light would actually read as warmer, and would give off a more red colour. Each season of the year will have a varying range of colours related to the temperature range of the season. A simple high level diagram can be shown below:



Note that the reading of temperature, and updating light will happen continuously at a defined time interval. The most ideal would be 30 minute time intervals, as temperature generally changes significantly in that time range. Therefore, the device will read temperature every 30 minutes, and update the light value to reflect this.

3 Input/Output

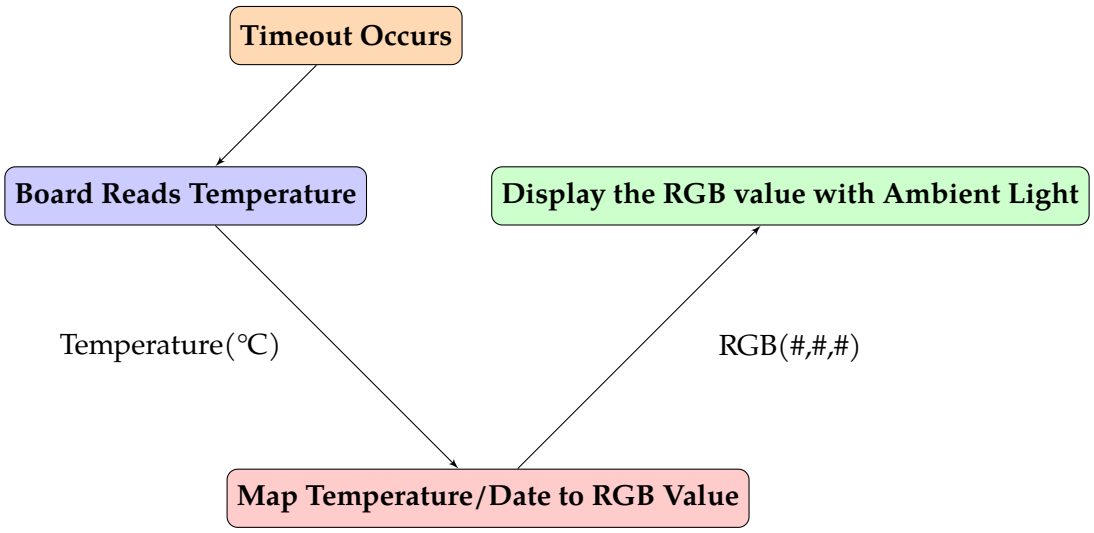
The ambient light project will have two major inputs(temperature, season), and one major output(RGB to a light component). The system will continuously read data from the environment to detect potential changes in temperature, and will update the RGB value of the light to reflect this(in combination with a season value). Temperature input will occur via ADC reading from channel 0 of the DE10. Seasonal input will be read from push buttons(1-4) on the DE10 board. Output is ideally to a bulb that can change itself to a specific RGB colour(it is difficult to find the correct device for this, but it definitely exists). The furthest level of design for this output will be a skeletal function to set this device to some RGB values. No development will be done for this interfacing as it is not known how such interfacing will be conducted.

4 Initial Software Design

4.1 High Level Design

The general design will flow in a simple manner. When the device is plugged in, the season will need to be input via a push button. After this occurs, the general flow will happen.The

board will read the temperature from the environment at specific intervals via a continuous update function. This temperature will be passed to a handler. The handler will determine an appropriate RGB value based on this temperature and the season. The RGB value produced will then be sent to the ambient light module which will display the given RGB value. Below is a quick summary of this:



The design of each functionality will be described in four subsections below.

4.2 Continuous Update

The A9 private timer will be used as the timer for the system. The main function of the timer will be done in the checkInterval() function. Main timer functionality will be allocated to various methods(setTimer, lapTimer, clearTimer, startTimer, stopTimer). The checkInterval function will check the current timer value to check divisibility by 10 with use of a %. If the current value is some multiple of 10, the getRGB() function will be called to update the RGB value based on the environment. In addition, a setColour() function will then be called to update the colour of the light. The timer will constantly be incremented.

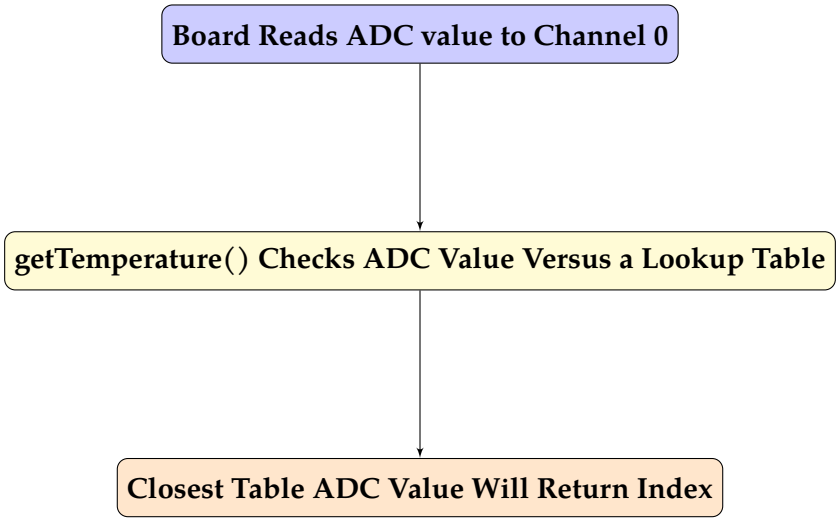
Listing 1: checkInterval()

```
//A function that checks the currentMillis to see if the timer is at intervals
    of 10 seconds
void checkInterval() {

    //Use modulo to see if remainder 10
```

```
    if (currentMillis % 10 == 0) {

                //Debug line used to check current time
            printf("Current time surpassed is %d seconds\n",currentMillis);

                //Get the current RGB colur value based on temp, season
            getRGB(getTemperature(),season);

      }

        //Increment the tiemr if the timer timeouts
        if (MILLISECOND_TIMER->status == 1){

            currentMillis++;

            //Reset
            MILLISECOND_TIMER ->status = 1;
        }

}
```

4.3 Temperature

The temperature reading function will be done by use of the ADC voltage input. The ADC channel 0 of the DE10 will be used. The value will be read, then sent to a getTemperature function. The function will compare the ADC value against a ADC/temperature lookup table. The index of the closest ADC value will notate the temperature of the environment. The design can be illustrated below:



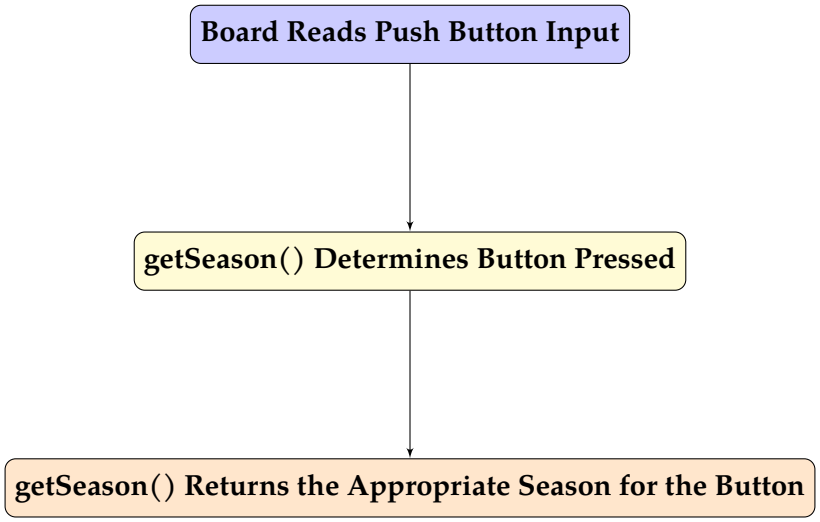
Listing 2: getTemperature()

```
//Get the temperature based on the current ADC value
int getTemperature() {

    unsigned char near1=0, near2=0;

    //Get interger value of ADC
    //Note getADC I used in the lab, so I will not put it here(it is in the full
        source code however)
   int ADC = getADC();

    //Default temperature of 20 celsius
   int celsius = 20;

   //Loop through temperature array, look for the specific ADC value
   for (int i = 1; i < 101; i++)
   {
      //If current index is the ADC value we return the index as the index is the
          celsius temperature
      if ((temperaturetable[i]>=ADC) && (ADC>=temperaturetable[i+1]) ) {

            //Check if the ADC value in bounds
         near1 = temperaturetable[i]-ADC;
         near2 = ADC-temperaturetable[i+1];

         if(near1<near2) celsius=i;
         else  celsius=i+1;
      }
```

```
    }

     //Return value in celsius
    return celsius;
}
```

4.4 Season Initialization

The season function is required to determine the specific season that the device is to be configured for. A user will be able to press a push buttons(1-4) to denote seasons(Spring, Summer, Fall, Winter). This is a simple task. The input from the push button on the DE10 will be read and parsed. It will then be converted to the specific month via a switch statement. Finally, the function will return the current season value for the entire system.



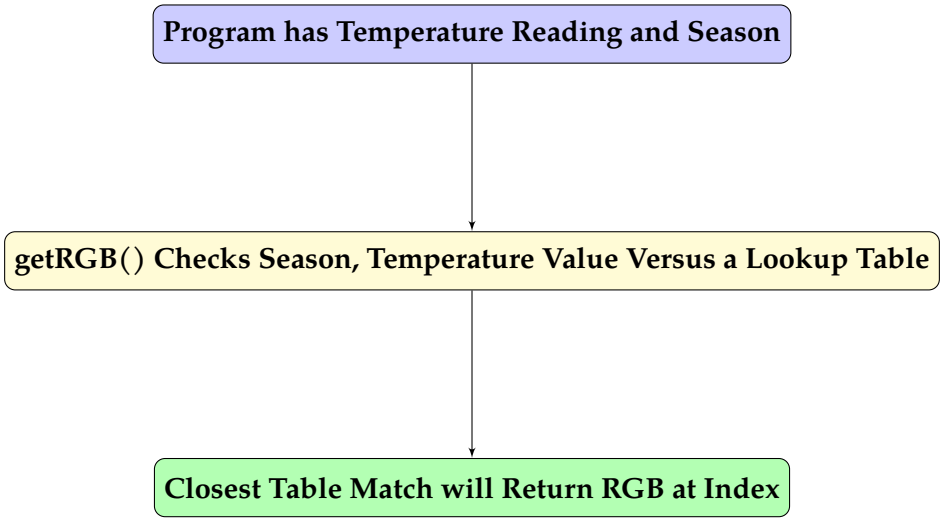
Listing 3: getSeason()

```
//Return a season based on push button input
int getSeason(){

    //Get the current value of the push buttons
    volatile int inputValue = *pushBtnInput;

    //Return appropriate season for the push buttons 1-4
    switch(inputValue){
        case 1:
            return 0;
```

```
            break;
        case 2:
            return 1;
            break;
        case 4:
            return 2;
            break;
        case 8:
            return 3;
            break;
    }

    //If not a selected choose the season Summer
   return 1;

}
}
```

4.5 Ambient Light Generation

The ambient lighting RGB value will need to be generated in some manner. This will be done via a lookup table. An array of RGB values specific from cold(0) to hottest(12) will be initialized. In addition to this, an seasonal temperature array will hold the range of season specific highs and lows. These are to be generated from online sources. This array will contain extreme low temperature(0) to extreme high temperatures(12). Four arrays of this type will be combined into a seasonal array that can be queried with the index 0 - Spring, 1 - Summer, 2 - Fall, 3 - Winter. When a function receives an input of temperature and season, it will look at this table and determine the closest RGB values to the input data. A flow chart can be seen below that outlines this graphically:



Listing 4: getRGB()

```
//Table that maps low to high temps of seasons Spring -> Winter
//0 - Spring, 3 - Winter,
//Note: temperature isfor Toronto
const unsigned short seasonalTable[4][12] =
  {{5,7,9,10,11,13,15,16,17,18,19,20},
  {16,17,18,19,20,21,22,23,24,25,26,30},
  {7,8,9,10,13,14,15,16,17,18,19,20},
  {-25,-20,-15,-13,-10,-8,-4,-1,0,1,3,5},
  };


//Return an RGB value appropriate to a specific seasonal input(0->3) in
    combination with a temperature input
int getRGB(int temp,int season) {

    //Define a struture to hold RGB values
    struct RGB *rgb = malloc(12*sizeof(struct RGB));

    //The structure is 12 size
    //There are 12 temperature settings avaliable
    //0 is white and is coldest temperature for that season
    //12 is red, and is the hottest temperature for the season
    //I personally picked each colour code
   rgb[0].R = 255;
    rgb[0].G = 230;
    rgb[0].B = 204;
```

```
    rgb[1].R = 255;
    rgb[1].G = 217;
    rgb[1].B = 179;

    rgb[2].R = 255;
    rgb[2].G = 204;
    rgb[2].B = 153;

    rgb[3].R = 255;
    rgb[3].G = 191;
    rgb[3].B = 128;

    rgb[4].R = 255;
    rgb[4].G = 179;
    rgb[4].B = 102;

    rgb[5].R = 255;
    rgb[5].G = 166;
    rgb[5].B = 77;

    rgb[6].R = 255;
    rgb[6].G = 153;
    rgb[6].B = 51;

    rgb[7].R = 255;
    rgb[7].G = 140;
    rgb[7].B = 26;

    rgb[8].R = 255;
    rgb[8].G = 128;
    rgb[8].B = 0;

    rgb[9].R = 230;
    rgb[9].G = 115;
    rgb[9].B = 0;

    rgb[10].R = 204;
    rgb[10].G = 102;
    rgb[10].B = 0;

    rgb[11].R = 204;
    rgb[11].G = 0;
    rgb[11].B = 51;

    rgb[12].R = 204;
    rgb[12].G = 0;
```

```
    rgb[12].B = 0;

    //Similar operation to getTemperature, compare if the current temperature is
        in bounds of two adjacent
   unsigned char near1=0, near2=0;
   int index = 0;

   //Loop through through the relevant seasonal RGB array
    //The seasonal temperature table defined above, the relevant row is seleceted
        using the season function input
   for (int i = 1; i < 12; i++)
   {
      //Find closest temperature value index
      if ((seasonalTable[season][i]>=temp) && (temp>=seasonalTable[season][i+1])
          ) {

            //Once again, chose a indice that is closest to current temprerature
         near1 = seasonalTable[season][i]-temp;
         near2 = temp-seasonalTable[season][i+1];
         if(near1<near2) index=i;
         else  index=i+1;
      }
   }

    //Index now has indice that will be the best suited colour to the seasonal
        temperature.

   //Return the RGB value at this index as #,#,#
    return rgb[index].R,rgb[index].G,rgb[index].B;

}
```

5 Prototyping Plan

5.1 Breakdown

The project has four general functions it will need to have implemented on the microcontroller. These are reading the environment temperature continuously, reading the season indicator at initialization of system, and updating the RGB value of a light component. In addition, each of these functions will be read at a particular interval so a timer function additionally is required. Prototyping of these functions will be further explained in their respective sections below.

Note: I mainly explain how I am to isolate each function and test individually at home on a simulator. The entire system will together be tested on the physical DE10 board in the lab, and my results will be explained in the coming sections.

5.2 Continuous Updates

The intervals of continuous updates must be prototyped before being combined with other project functions. This will be done through isolation of the function via the simulator. First, the entire function will be tested on the simulator to ensure 10 second read intervals occur successfully. A short interval for conduction of tests is important so issues can be resolved. If this function works, the getInterval function will be added to the main source code and RGB will be written every 10 seconds. This is relatively easy as we simply call the getRGB() function inside the getInterval function. Additionally, this will be tested to ensure detection occurs every 10 seconds to ensure continuous reads occur. If further success occurs, the next steps are to ensure the timer will work on the DE10 physical board, and test larger time intervals(10 minutes, 30 minutes).

5.3 Temperature

A prototyping plan will be established in the following manner for temperature generation. A function will be created that will take ADC input, and given this, will compare such input versus a lookup table. The closest ADC value in the table to the ADC input will be found, and this table value will reside at some index i. The i value will denote the closest temperature value in degrees celsius to the ADC value. To test this idea, first the code will be ran in a controlled environment, which will be an online C compiler. The function will be ran with a bogus(but realistic) fake ADC input to see how the function responds. After several inputs, if the function works as intended and the output temperature is correct, testing with the simulator will occur. The function will be ran reading from the random ADC values in a simulator. The main purpose of this is to ensure the ADC read code works with a DE10 microcontroller. Consequently, the simulator output temperature output generated will be compared vs manual hand calculations to check correctness.

5.4 Season Initialization

Some prototyping is required to ensure that the input is properly parsed and converted. A function is to be generated that will take a look at the push button input to the DE10, read this value from the register, and then convert the pushed button to a corresponding season. This will be done with a switch statement used to check the current push button. The prototyping will first commence in an isolated manner on the simulator alone with attempts to print out the current season based on the push button. If this properly works, the code will be integrated with other functions in the simulator.

5.5 Ambient Light Generation

The ambient light selection will need to be prototyped. A function will be created to take temperature, and season to generate a relevant RGB value. The closest temperature value of the specific seasonal table will be selected, and its index will be noted. The index will then determine the RGB value that is chosen. The function first will be tested in an isolated environment alone. An online C compiler will be used. The entire function will be ran with bogus temperature and season values(but realistic). The function should return a colour that would reflect how the temperature is generally regarded in the season. For instance, getRGB(26,2) should return a red value as it in summer 26 degrees is regarded as quite warm. This can easily be tested for correctness. Next in prototyping, the function will be added to the main program and tested with fake numbers once again, however it is in the simulator. If this works, finally the getRGB function will be ran with the temperature function, and season functions to try to have the entire process working.

6 Selection of Microcontroller

Many microcontrollers could be considered to improve the project. Some metrics used to compare various types of MCs difficulty of interfacing with a temperature sensor(onboard or peripheral) and a RTC(onboard or peripheral). An onboard temperature sensor will make the measurement of the environment much more accurate and effortless. Presence of an RTC onboard the system would make the project less user reliant, it would allow the date to be stored on the board so no seasonal initialization is required. Comparison will generally be done with regards to difficulty of accessing these two systems. Hard systems are general external peripherals(the function can be done, but it would take a lot of work), while easy means the function is simple to access on the microcontroller and usually onboard.

Microcontroller Feature Comparison		
Name	Temperature Sensor Interfacing	RTC(Real Time Clock)
DS83C530	Hard	Easy
ESP32	Easy	Easy
Arduino Uno	Moderate	Moderate
DE10	Hard	Hard

In the project a DE10 was used as it is the given hardware available without any purchase. Fortunately, the code used could be reused, and simply fitted to any of these MCs with not too much work. If a new choice could be made, the ESP32 would likely be chosen, as it includes the two main functions of the system on-board in discrete ways, as all other systems require at least one peripheral. Having the functions contained on a system

would make interfacing much easier from C with less error room. Additionally, measurements for time, and temperature will be accurate. Moreover, the scaling of the system to one device makes the whole design more similar to a completed product, as the end goal would be to have the entire system in one casing.

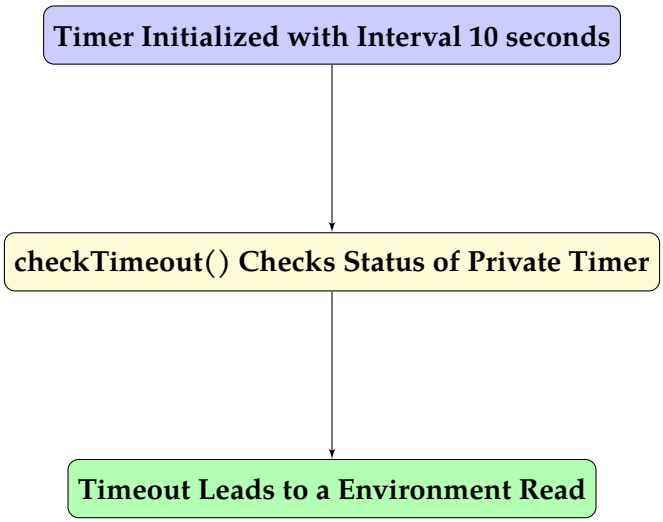
7 Revised Software Design

7.1 Revision

Revision of the timer function occurred, alongside an entire redesign of the system used to get temperature, as the ADC did not exactly work as expected on the DE10.

7.2 Timer Revised

The timer was modified from the initial design to timeout at 10 seconds rather than check for intervals of 10 seconds. There was a glaring issue not described with the modulo operation used. When initial prototyping occurred, the timer seemed to repeat the checkInterval function if condition multiple times, and hundreds of calls were made to getRGB. This initial design was clunky, and inefficient. Instead, it was decided to simply use a timeout value of 10 seconds and check for the flag to make more controllable, and consistent timer. The new flow can be seen below:



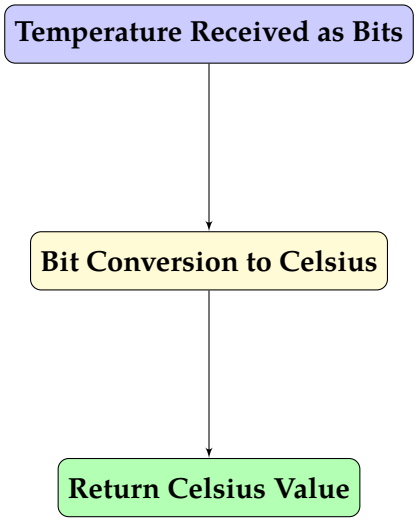
This cycle continuously repeats, with the timer being entirely reinitialized after each step. To see the functions in code:

Listing 5: Timer Functions

```
// Initialize the timer
void initTimer()
{

    //Set the time interval to 10 seconds
    int interval = 1000000000;

    //Put this into the timer laod register
    timer_1->load = interval;

    //Set control register, start timer
    timer_1->control = 3 + (1 << 8);

    //Set the status to 1 to reset timeout flag
    timer_1->status = 1;
}

// Check if the timer has timed out
void checkTimeOut()
{

    //Check for the timeout flag to show
    if (timer_1->status == 1)
    {


        //Set the current RGB value of some device to new one based on
            temperature, season
      setColour(getRGB(getTemperature(),season));

        //Reset the timer using the init interval function
        initTimer();
    }
}
```

7.3 Temperature Revised

It was decided a more accurate, and better way to read temperature was use of a peripheral for the specific purpose. The specific peripheral potentially used would be the AS621 chip. An I2C network would communicate with this device and so consequently, the previous temperature code was revisited and changed. A conversion from bits to temperature was given for the peripheral in manual and is provided in code. Generally, continuous reads were made from the AS621 and a temperature returned. The basic flow is:



Note that, I2C communication is trickier, so the code would need to rigorously tested to ensure it does indeed work with the AS621. The best effort was used to try to set up the communication however(heavy use of lecture examples).

8 Prototyping Results

8.1 Lab Experience

As the system was tested in the lab, some issues arose with the code. First, the ADC input was not working as expected. The ADC values being read were not proper, and interfacing with the ADC was incorrect. Additionally, the timer did not fire at correct intervals at all, and constantly ran. This issue was a software bug and can be fixed by correctly interfacing with the timer. The code initially used for the timer was inefficient, and understanding of the system was low, but as this course progressed, the timer system understanding grew much more, and it was decided a redesign was required.

8.2 Hardware Planning

The hardware waited on would be a light bulb device, and the AS621 temperature sensor(if it was in the lab, it was not prototyped with). This plan would only be ran after the software bugs above are resolved. First, the temperature sensor would be connected to the DE10, and the I2C communication would be tinkered with. The desired outcome is that the code could be prepared beforehand, using the AS621 manual. Reading to the console would try to be done with this component, to ensure a realistic temperature is read. The most robust testing would occur with use of the device outside, where temperature is much more variable than an indoor environment. If successful, next the light bulb

device would be tested independently with hardcoded RGB value written to it. If this is successful, the changing of the colour of the light will try to be done using two, or three different hardcoded RGB values. If this works, then the device will try to be connected to the entire system. The optimal test environment would be some dynamically changing temperature environment, so the bulb could be tested to see if it works. The tables constructed in software will be checked with the bulb colour to ensure the correct colour shows.

9 Source Code

Included below is an appendix of the source code with revised design. However in my submission I include my old design C file, along with the new file.

Listing 6: Full Source Code

```
#define SW_BASE 0xFF200040
#define KEY_BASE 0xFF200050
#define TIMER_A9_BASE 0xFFFEC600
#define I2C0_BASE 0xFFC04000


// pointers to hardware
volatile I2Cn* const I2C_ptr = ( I2Cn* )I2C0_BASE;
volatile a9_timer *const timer_1 = (a9_timer *)TIMER_A9_BASE;
volatile int* const pushBtnInput = (int*)KEY_BASE;

#include <stdio.h>
#include <math.h>

//I2c structure for the DE10 I2C
//Code from the unit 9: Communications
typedef struct _I2Cn
{
   // uses variables called 'pad' to align
   // needed registers in memory
      int control;       //0x00
      int target;        //0x04
      int slave;         //0x08
      int pad0;          //skip
      int data_cmd;      //0x10
      int std_scl_hcnt;  //0x14
      int std_scl_lcnt;  //0x18
      int fast_scl_hcnt;    //0x1C
      int fast_scl_lcnt;    //0x20
      int pad1;          //skip
```

```
    int pad2;          //skip
    int intr_status;   //0x2C
    int intr_mask;     //0x30
    int raw_intr_status; //0x34
    int rx_fifo_thr;   //0x38
    int tx_fifo_thr;   //0x3C
    int cmb_intr;      //0x40
    int rx_under_intr;    //0x44
    int rx_over_intr;  //0x48
    int tx_over_intr;  //0x4C
    int intr_read;     //0x50
    int tx_abort_intr;    //0x54
    int rx_done_intr;  //0x58
    int activity_intr;    //0x5C
    int stop_dtct_intr;   //0x60
    int start_dtct_intr; //0x64
    int gen_call_intr;    //0x68
    int enable;        //0x6C
    int status;        //0x70
    int tx_fifo_lvl;   //0x74
    int rx_fifo_lvl;   //0x78
    int sda_hold;      //0x7C
    int tx_abort_src;  //0x80
    int gen_slave_nack;   //0x84
    int dma_control;   //0x88
    int dma_tx_lvl;       //0x8C
    int rx_data_lvl;   //0x90
    int sda_setup;     //0x94
    int ack_gen_call;  //0x98
    int enable_status;    //0x9C
    int ss_fs_supp;       //0xA0
} I2Cn;


// Structure for the timer
typedef struct _a9_timer
{
    int load;
    int count;
    int control;
    int status;
} a9_timer;

int season;
int count = 0;
```

```
struct RGB {
    unsigned char R;
    unsigned char G;
    unsigned char B;
};

//Suppose we have a date value, we need to convert this date to a season
//This table will contain various seasons
const unsigned short seasonalTable[4][12] =
  {{5,7,9,10,11,13,15,16,17,18,19,20},
  {16,17,18,19,20,21,22,23,24,25,26,30},
  {7,8,9,10,13,14,15,16,17,18,19,20},
  {-25,-20,-15,-13,-10,-8,-4,-1,0,1,3,5},
  }; //try to do

//Convert the AS621 digital hex value to a celsius value
int getTemperature() {

    //Get temeprature as a hexadecimal value
    int hexTemp = getHexTemp();

    //Manual gives a conversion formula
    //0.0078125 * Hex value = complement
    //negative is (Hex value - 1)
    //Calcualte celsius using this

    //Use formula
    int celsius = hexTemp*0.0078125;

    //Return celsius value
   return celsius;
}

//Want to get internal data register value
int getHexTemp() {
    //First it says in manual to write to index register
    //Write to index register
    I2C_ptr->data_cmd = 0x46 + 1;

    //Write TVAL registervalue to let device know we want this
    I2C_ptr->data_cmd = 0x0;

    //Wait for data to arrive
    while ( I2C_ptr->rx_fifo_lvl == 0 );

    int tempHex;
```

```
 //Get first half of bits from the tempval register
    tempHex = I2C_ptr->data_cmd;

    //Wait for next bits to arrive
    while ( I2C_ptr->rx_fifo_lvl == 0 );

     //Get net half of bits from the tempval register, shift temp to properly
         reflect theri value
    tempHex = tempHex + (I2C_ptr->data_cmd << 8);

    //Return combined lsb,msbs
    return tempHex;

}

long decimalToBinary(int decimalnum)
{
    long binarynum = 0;
    int rem, temp = 1;

    while (decimalnum!=0)
    {
        rem = decimalnum%2;
        decimalnum = decimalnum / 2;
        binarynum = binarynum + rem*temp;
        temp = temp * 10;
    }
    return binarynum;
}

//Init I2C on DE10
void init_I2C()
{

    //Enable the I2C pin, disable the I2C controller
   I2C_ptr->enable = 2;

   //Looking at enable status until bit 0 is cleard
   while ( I2C_ptr->enable_status & 0x1 );


   //Here I set the appropriate control bits
   I2C_ptr->control = 0b01101101;

   //Set the specific slave address for the AS621
```

```
   I2C_ptr->target = 0x46;

   //Not 100% sure about this
   // set count for high-period of clock
   I2C_ptr->fast_scl_hcnt = 90;
   // set count for low-period of clock
   I2C_ptr->fast_scl_lcnt = 160;

   //Enable the I2C controller after it has been configured for the DE10
   I2C_ptr->enable = 1;


   //Wait until the status register is set to 1 which would mean enabled
   while (( I2C_ptr->enable_status & 0x1 ) == 0 );
}

//Initalize the AS621 device
void initAS621() {

    //Do required initialization
    //Step One as said in manual, START condition is generated by master by
        pulling SDA from lopgic high to low, while SCL is kept at high
    //Pull data line from high to low
    //Connetion 3 is chosen
   I2C_ptr->data_cmd = 0;

    //Slave address byte fro device is completed with 9th bit to itnegate a read
        1 or write 0
   //Byte + 1 to indicate a read
   I2C_ptr->data_cmd = 0x46 + 0;

}

//Return a season based on push button input
int getSeason(){

    //Get the current value of the push buttons
    volatile int inputValue = *pushBtnInput;

    //Return appropriate season for the push buttons 1-4
    switch(inputValue){
        case 1:
            return 0;
            break;
        case 2:
            return 1;
```

```
            break;
        case 4:
            return 2;
            break;
        case 8:
            return 3;
            break;
    }

    //If not a selected choose the season Summer
   return 1;

}



//Return an RGB value appropriate to a specific seasonal input(0->3) in
    combination with a temperature input
int getRGB(int temp,int season) {

    //Define a struture to hold RGB values
    struct RGB *rgb = malloc(12*sizeof(struct RGB));

    //The structure is 12 size
    //There are 12 temperature settings avaliable
    //0 is white and is coldest temperature for that season
    //12 is red, and is the hottest temperature for the season
    //I personally picked each colour code
   rgb[0].R = 255;
    rgb[0].G = 230;
    rgb[0].B = 204;

    rgb[1].R = 255;
    rgb[1].G = 217;
    rgb[1].B = 179;

    rgb[2].R = 255;
    rgb[2].G = 204;
    rgb[2].B = 153;

    rgb[3].R = 255;
    rgb[3].G = 191;
    rgb[3].B = 128;

    rgb[4].R = 255;
```

```
 rgb[4].G = 179;
 rgb[4].B = 102;

 rgb[5].R = 255;
 rgb[5].G = 166;
 rgb[5].B = 77;

 rgb[6].R = 255;
 rgb[6].G = 153;
 rgb[6].B = 51;

 rgb[7].R = 255;
 rgb[7].G = 140;
 rgb[7].B = 26;

 rgb[8].R = 255;
 rgb[8].G = 128;
 rgb[8].B = 0;

 rgb[9].R = 230;
 rgb[9].G = 115;
 rgb[9].B = 0;

 rgb[10].R = 204;
 rgb[10].G = 102;
 rgb[10].B = 0;

 rgb[11].R = 204;
 rgb[11].G = 0;
 rgb[11].B = 51;

 rgb[12].R = 204;
 rgb[12].G = 0;
 rgb[12].B = 0;

 //Similar operation to getTemperature, compare if the current temperature is
     in bounds of two adjacent
unsigned char near1=0, near2=0;
int index = 0;

//Loop through through the relevant seasonal RGB array
 //The seasonal temperature table defined above, the relevant row is seleceted
     using the season function input
for (int i = 1; i < 12; i++)
{
   //Find closest temperature value index
```

```
      if ((seasonalTable[season][i]>=temp) && (temp>=seasonalTable[season][i+1])
          ) {

            //Once again, chose a indice that is closest to current temprerature
         near1 = seasonalTable[season][i]-temp;
         near2 = temp-seasonalTable[season][i+1];
         if(near1<near2) index=i;
         else  index=i+1;
      }
   }

    //Index now has indice that will be the best suited colour to the seasonal
        temperature.

   //Return the RGB value at this index as #,#,#
    return rgb[index].R,rgb[index].G,rgb[index].B;

}

//General tranfer write to index register

// Initialize the timer
void initTimer()
{

    //Set the time interval to 10 seconds
    int interval = 1000000000;

    //Put this into the timer laod register
    timer_1->load = interval;

    //Set control register, start timer
    timer_1->control = 3 + (1 << 8);

    //Set the status to 1 to reset timeout flag
    timer_1->status = 1;
}

// Check if the timer has timed out
void checkTimeOut()
{

    //Check for the timeout flag to show
    if (timer_1->status == 1)
    {
```

```
        //Set the current RGB value of some device to new one based on
            temperature, season
       setColour(getRGB(getTemperature(),season));

        //Reset the timer using the init interval function
        initTimer();
    }
}



//This method is very dependant on what peripheral is used
//The hope is to use the rgb values to write a specific sequence of lights to be
    on
int setColour(r,g,b) {

   //We would conduct writing of generated RGB values to the specific bulb
       peripheral here
    //ATM we have a skelton method though to show where this would happen
    printf("Red Value: %ld \n", decimalToBinary(r));
    printf("Green Value: %ld \n", decimalToBinary(g));
    printf("Blue Value: %ld \n", decimalToBinary(g));

}


int main(void)
{
    //Get the current season
   season = getSeason();

    //Initialize the I2C controller for AS621
   init_I2C();

    //Initialize the AS621 slave device for temperature I2C communication
    initAS621();


    //Start timer
    initTimer();


   while(1) {
```

```
        //Continiously check the timeout
        //Update the light if interval of 10 seconds reached
      checkTimeOut();

   }

}
```

10 Conclusions

10.1 Feasibility

The project is feasible and the nature and the concept is very desirable. Ambient lights are an item that people generally really like, as seen with any smart lights such as the LIFX brand lights. A light that would automatically change would be very similar, and could potentially be marketable if simple and cheap. One can see with companies like Apple, that a simple device that looks sleek and has a great user experience will sell. The function of the system could be replicated on any devices, as existing smart lights could be programmed to do what the Temperature Ambient Light does if a device such as a phone or laptop were connected. Interestingly, the system with a temperature sensor in combination with a complex RGB light all in one device is not a common market item, therefore some success could potentially be had. The current system however would require a massive scaling down into a simple bulb so the device is contained. This would most definitely require reworking of code, and a change of all hardware to the smallest magnitude.

10.2 Experience

The project was incredibly educational in nature. The specific idea pursued had many different areas of the course involved in construction, so the practical application of concepts was refined. It was incredibly interesting that through combining all the different components of ECE3375, allows building of some really complicated systems. I could say I am much more interested in MC programming after this project than before. Initially it all seemed very difficult and confusing, but now it seems incredibly easy as you simply need to consult the devices manual to write working code. This project definitely showed me why this course is taken, alongside an introduction into designing a software system. The latter skill will be incredibly helpful in construction of a capstone project.

10.3 Sustainability

The design of the system is relatively sustainable. With presence of an RTC however, I think that such a device could be made very sustainable. The system would only turn

on at specific time intervals(5pm-10pm). This would really help minimize the wasting of power, and increase the lifespan of the bulb used. In addition, reading of temperature could be done at longer intervals of the day to minimize processing power used. The two suggestions could make the system more sustainable, and energy efficient.

10.4 Environmental Concerns

Some issues with the device could be had with the bulb used for the device. A bulb used will eventually die, so it is important that a long lifespan be considered in picking of this peripheral. The issue resides with the fact that cheaper bulbs have a lower lifespan, so a compromise will need to be made between the lifespan of bulb, and the price. The device would therefore be disposable in nature, and not the greatest for the environment. The optimal bulb to use would be an LED which can last up to 17 years if used for 8 hours per day. If the time interval lighting is implemented, such a bulb would be very good to ensure an environmentally conscious system design.