Vanity

CS122A: Fall 2017

Wendy Li

Table of Contents

Introduction

I want to work on something that is simple to use and solves a problem I have on a day to day basis as a college student with limited funds. The six dollar lamp I found in the sale section of ikea usually does not usually provide enough light for me to get ready at my desk. When I get ready to go out, I want to be able to see what I am actually putting on my face. I decided to created a vanity that has smart features to brighten my room when I need it.

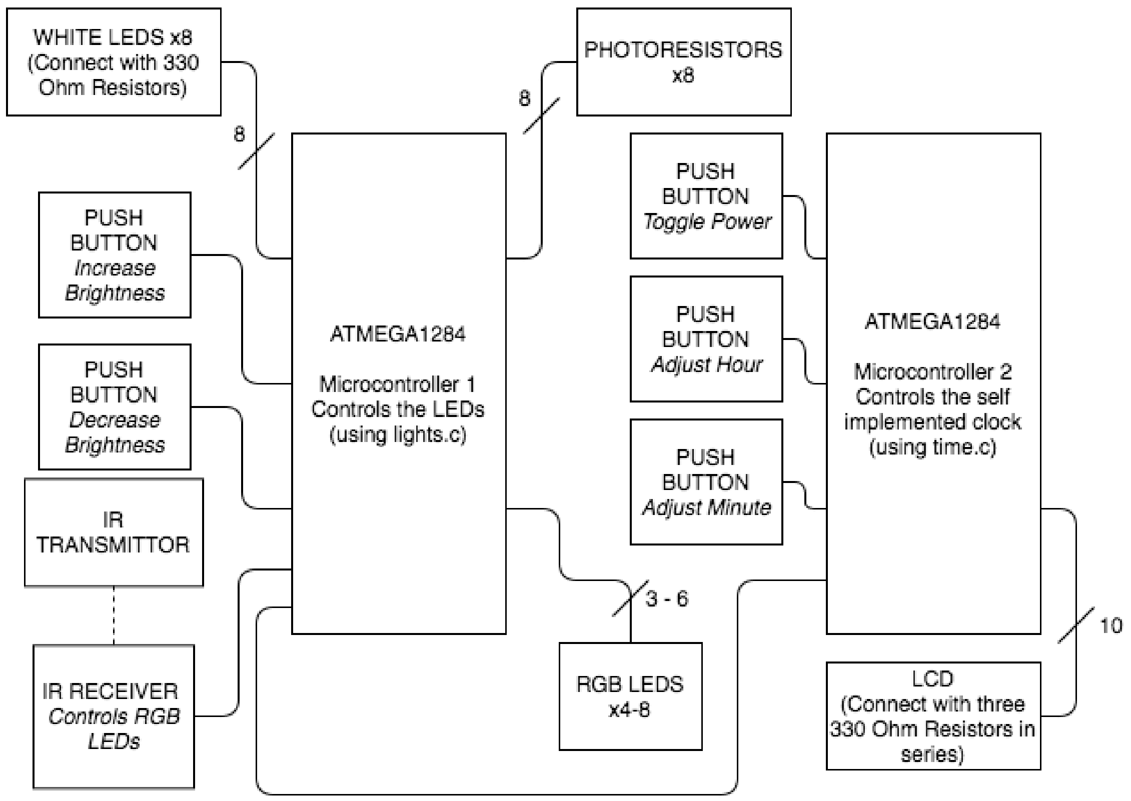
Using photoresistors I am assessed the brightness of the room. Using that information, I adjust the brightness of connected LEDs using pulse with modulation. I place the LEDs around a see through mirror to create a vanity that is responsive to the brightness from the room. Using a LCD screen, I display the time and a welcome message. For added fun, I also implemented features that make colorful lights dance. These lights are controlled with a remote.

Hardware

Parts List

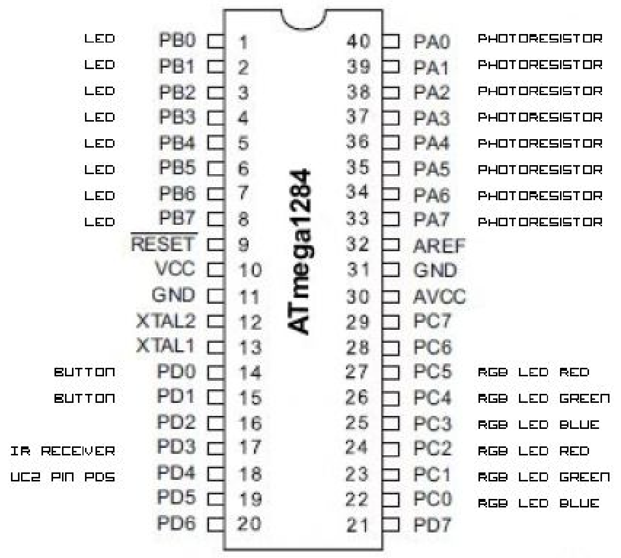
Part	Part #	Quantity	Price (optional)	Link
ATMega1284	ATMega1284	2		
Photoresistors		8	$3.99 (for 20)	here
White or Warm White LEDs 5mm		8	$8.49 (for 350)	here
RGB LEDs (Cathode)		4-8	(included above)	here
Push Buttons		5	$1 (for 4)	
IR Receiver	VS1838 TL1838 VS1838B	1		
IR Transmitter		1		
LCD Screen (White on Black)	YB1602A (black)	1	$3.99	here
10k potentiometer		1		
330Ohm Resistors		11		
See through Mirror	n/a	1	$16.99	here
Wires & connectors		alot		
		Total	$34.46	

Block Diagram

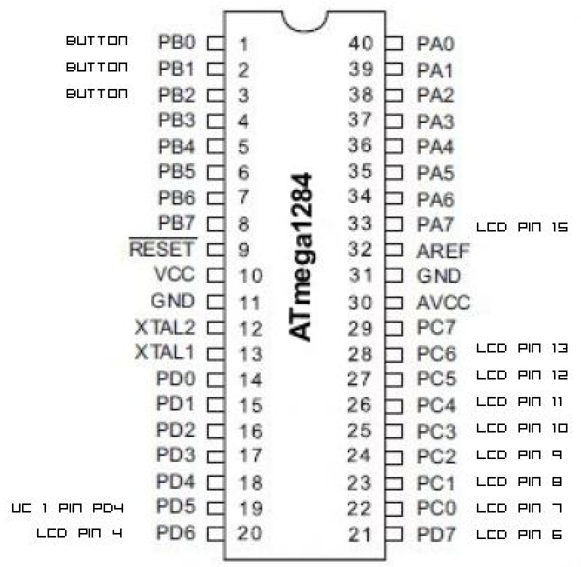

WHITE LEDS x8 (Connect with 330 Ohm Resistors)
8
PUSH BUTTON Increase Brightness
PUSH BUTTON Decrease Brightness
IR TRANSMITTOR
IR RECEIVER Controls RGB LEDs
ATMEGA1284
Microcontroller 1 Controls the LEDs (using lights.c)
8
PHOTORESISTORS x8
3 - 6
RGB LEDS x4-8
PUSH BUTTON Toggle Power
PUSH BUTTON Adjust Hour
PUSH BUTTON Adjust Minute
ATMEGA1284
Microcontroller 2 Controls the self implemented clock (using time.c)
10
LCD (Connect with three 330 Ohm Resistors in series)

Pinout (For each microcontroller/processor)

UC 1: LIGHTS.C

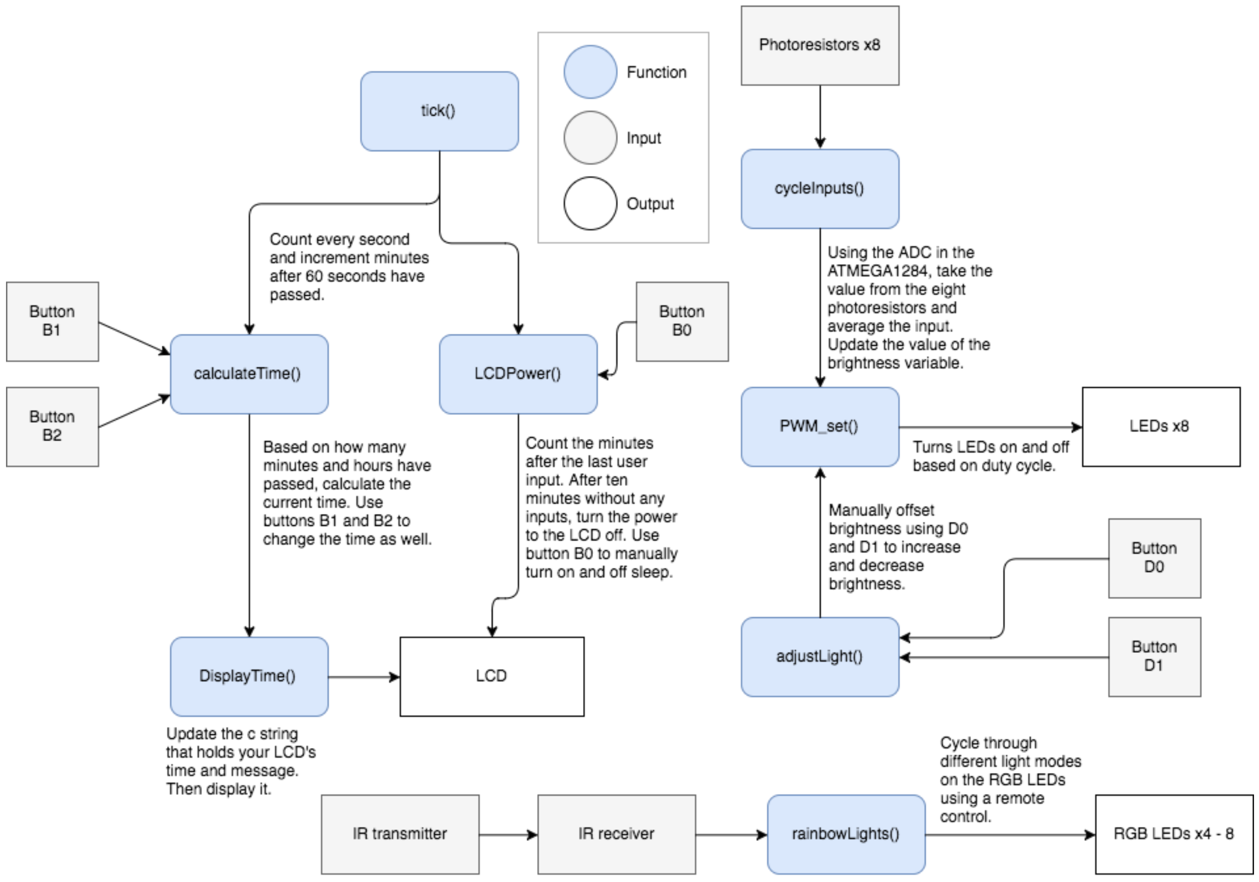


UC 2: TIME.C



Software

The software designed for this project was implemented using the PES standard. The overall design as a task diagram is included below. Detailed state machine diagrams included at the end in the appendix.


Function
Input
Output
tick()
Photoresistors x8
cycleInputs()
Count every second and increment minutes after 60 seconds have passed.
Using the ADC in the ATMEGA1284, take the value from the eight photoresistors and average the input. Update the value of the brightness variable.
Button B1
Button B2
calculateTime()
LCDPower()
Button B0
PWM_set()
Turns LEDs on and off based on duty cycle.
LEDs x8
Based on how many minutes and hours have passed, calculate the current time. Use buttons B1 and B2 to change the time as well.
Count the minutes after the last user input. After ten minutes without any inputs, turn the power to the LCD off. Use button B0 to manually turn on and off sleep.
Manually offset brightness using D0 and D1 to increase and decrease brightness.
Button D0
Button D1
adjustLight()
DisplayTime()
LCD
Update the c string that holds your LCD's time and message. Then display it.
Cycle through different light modes on the RGB LEDs using a remote control.
IR transmitter
IR receiver
rainbowLights()
RGB LEDs x4 - 8

tick()	**Handles ticking the clock once every second and increments minutes after 60 seconds.**
calculateTime()	**Calculates the time using the tickFSM and uses B1 to increment minutes, B2 to increment hours.**
LCDPower()	**Turns on and off LCD display (controlled by PB0). Also sends signal to other microcontroller to turn off LEDs. (set brightness to 0)**
DisplayTime()	**Displays the time based on variables "hours" and "minutes" as**

	"hours":"minutes". Displays in military time along with a greeting message based on the time of day it is.
PWM_set()	**Pulse with modulation turns LEDs on and off for varying duty cycles to adjust brightness.**
adjustLight()	**Adjusts brightness of LEDs using buttons D0 and D1. Changes a signed variable manualOffset.**
rainbowLights()	**Controls the RGB LEDs. Has three modes: red(night mode), blue flashing (soothing mode), and rainbow (fun mode).**
cycleInputs()	**Takes input from photoresistors and averages them. Then updates brightness values to change brightness of LEDs after input has been taken from all eight photoresistors. Then, cycle through and update again.**

Implementation Reflection

Using other components
The LEDs I used in the project are definitely not bright enough to light up a person while they look into it. I wish i could have used brighter lights but I didn’t have enough time to research other options. If i were to make this again, I would think about using a different kind of LCD screen because I only want light to show through where the text is. However, in the implementation, the LCD I used was slightly too bright and could be seen through the mirror.

Assembly - Putting it All Together
I spent a lot of time on the code for the project and implementing parts. However, I underestimated the amount of time it would actually take for me to put all my parts in place with connectors. I ended up with a jumbled mess of wires with connections that would pop out constantly causing issues. These issues could be confused for issues in the programming which would cause me to go back and forth assembling and disassembling parts when it was unnecessary. I should have taken more time to carefully tape down wires and find breadboards that were smaller and did not take up as much space.

I like how my project turned out. The mirror really brings it all together and it ended up having a clean look to it even though the inside was a mess of wires.

Milestone

I want to be done with converting the analog data of the photosensors to digital and using PWM to dim the lights on the LEDs according to the amount of light input that was read. Inputs from buttons will also turn lights on/off and change brightness.

I was able to complete my milestone in time. However, I still had a few bugs with adjusting the brightness. The issues came from trying to add signed and unsigned numbers together. I had to run through and make sure that variables were casted as unsigned and signed at the appropriate times to make the addition and subtraction work properly. I think I picked an appropriate milestone.

Completed components

- **Automatically adjusts lights**
    - **Takes in data for light input**
        - **Uses eight photosensors**
    - **Outputs appropriate light levels for brightness of the room**
        - **20 levels of brightness**
        - **Brighter in a dark room, darker in a bright room**
        - **Dim lights with PWM**
- **Sleep Mode**
    - **After 10 minutes from last input, lights will turn off (sleep mode)**
- **LCD Displays (connected on second microcontroller)**
    - **Display 1: Time (military time)**
        - **Set the time using two buttons**
    - **Display 2: Welcome message based on time of day**
        - **6:00AM - 11:59AM : “Good morning!” (06:00 - 11:59)**
        - **12:00PM - 5:59 PM : “Good Afternoon!” (12:00 - 17:59)**
        - **6:00PM - 9:59 PM : “Good Evening!” (18:00 - 21:59)**
        - **10:00PM - 5:59AM: “Good Night!” (22:00 - 5:59)**
- **Controls**
    - **On/Off - Button**
        - **Toggles LEDs, RGB LEDs, and LCD in and out of sleep mode**
    - **Manually adjusts lights**
        - **Uses two buttons to change light output**
        - **Offsets the brightness up to +10 and -10 brightness levels**
    - **Remote control RGB Lights**
        - **IR sensor to turn RGB lights on and cycles between sleep mode, relax mode, rainbow mode, and off.**
        - **Doubles as a night light that automatically goes to sleep after 10 minutes**
            - **Turn it on from your bed at night**

        - **Set Time - Buttons, change the hour and change the minute**
        - **On/Off - Remote control (Night mode)**
            - **IR sensor also turns LCD display on/off (sleep mode)**
        - **Sleep Mode**
            - **After 10 minutes from last input, displays will turn off (sleep mode)**
            - **Sleep mode preserves the set time on the LCD Screen**

Incomplete components

- **Incomplete: I did not program my IR transmitter to control the LEDs, LCD, and RGB LEDs. I thought that it did not make sense to need remote access to the mirror for functions that the user would use in proximity to the mirror.**
    - **Complete: It controls the RGB LEDs only and cycles through different outputs.**

- **Incomplete: I want the LEDs to change brightness and color.**
    - **Complete: I separated these two tasks to different LEDs.**
        - **The white LEDs are controlled using PWM to change the brightness.**
        - **The RGB LEDs are controlled using a FSM to change the mode (night mode, relax mode, rainbow mode).**

Youtube Links

- **Short video:** link **https://www.youtube.com/watch?v=2kJp44RgkWY**
- **Longer video:** link **https://www.youtube.com/watch?v=5a2piSyVOfY**

Testing

White LEDs
The LEDs are supposed to display a certain amount of brightness based on the input it read from eight photoresistors. The ATMEGA1284's on board ADC returns values of up to ten bit resolution. The value that was returned from the ADC was a number between 0 and 1023. The value of my variable *brightness* **was a number between 0 and 20 based on the value from the ADC.**

During testing, to see what range of values I was getting, I connected another set of LEDs to another port that was empty and set that port's value to *brightness*. **It was difficult to see what values of brightness correlated to what amount of light was in the room since I can not physically control photons. However, it did help me notice an issue I was having with overflow causing my lights to turn off or become extremely bright.**

Clock Displayed on LCD

While writing the code for my time code, I has issues with the LCD screen. Also during certain times of the day, the LCD would just begin to display garbage. Initially, the screen would constantly flash and the display did not look very good. When I had a friend test the progress I had, he said that he had a hard time deciphering what the display said.

I suspected that it was because the display was updating too often. I thought that since none of the characters were changing, that the display would not flicker so much. Instead, I used a variable to tell if during the current cycle and if it was, then update the display. This made the display much clearer.

Button combinations
While letting a friend test my project, he was extremely enthusiastic about pressing buttons. So much so that he was trying to press all five of them at once. This not only led to the buttons falling out of place but also to the LEDs to not work.

To solve this issue, I specified what to do when more than one button is pressed in situations that would lead to issues. Instead, if more than one button was pressed at a time, the state machine would stay in the same state. A null transition staying in the wait state was better than the entire program not working.

Basic tests

- **Testing how well the IR sensor worked with a simple program that turned on and off one LED when input was received.**
- **Testing the brightness of LEDs using PWM by setting fixed brightness levels each time the microcontroller was programmed.**
- **Testing the LCD screen worked by displaying a simple message such as "hello world!"**

Known Bugs

Lights.c :

- **When manually adjusting light down, even if brightness is the minimum value, manual offset continues to change.**
    - **You needs to up manual offset as much as you lowered it**
    - **Nothing is wrong with the way its implemented, it is just a poor design choice on my part**
- **Occasionally, LEDs turn off during change in environment lighting, if light does not turn back on after a few seconds, restart power.**
    - **Probably has to do with an addition error. It is possible that somewhere in the arithmetic, something is causing the value of brightness to become negative.**
    - **Also, certain points in the code, I add signed and unsigned numbers which may have caused issues in the arithmetic. I tried to cast the variables but it did not fully resolve the issue.**

        - **I continue to work on this issue because it is one of the main functionalities of the project. I want it to work smoothly without issues. I plan on using a set of tester LEDs to check the value that brightness is being set to. That way I can see how the brightness value in relation to what is being displayed on the dimming LEDs.**
    - **When the input from the other microcontroller comes in on PD4, there is a delay from the time that the lights should turn on and when they actually do, its approx a 300ms delay**
        - **Also, sometimes the lights wont turn on or off when they are supposed to from pressing the button on PB0 on the uc connected to the LCD**
            - **It helps to hold the button for a little longer till the lights turn on**

Time.c

- **Clock may be a bit off**
    - **Most likely just very small errors in how time is counted, it should only be milliseconds off. This would become more of an issue when the power has been on for longer periods of time.**
    - **I plan on using an actual timer to test if the clock becomes inaccurate over time. I also plan on cleaning up the time code to make sure it runs as smoothly as possible**

Resume/Curriculum Vitae (CV) Blurb

Self Adjusting Vanity

- **A vanity that automatically adjusts the brightness of the lights based on its environment**
- **Using an ATMEGA1284 microcontroller, photoresistors, IR sensor, and more**
- **Components controlled with eight different synchronous state machines**

Future work

- **More modes for RGB Lights**
- **Brighter LEDs**
- **Better LCD screen**
- **Sturdy Casing made from wood**
- **Organize wires**
- **Smaller breadboards, reduce use of excess space and reduce weight**
- **Lots of debugging and testing**

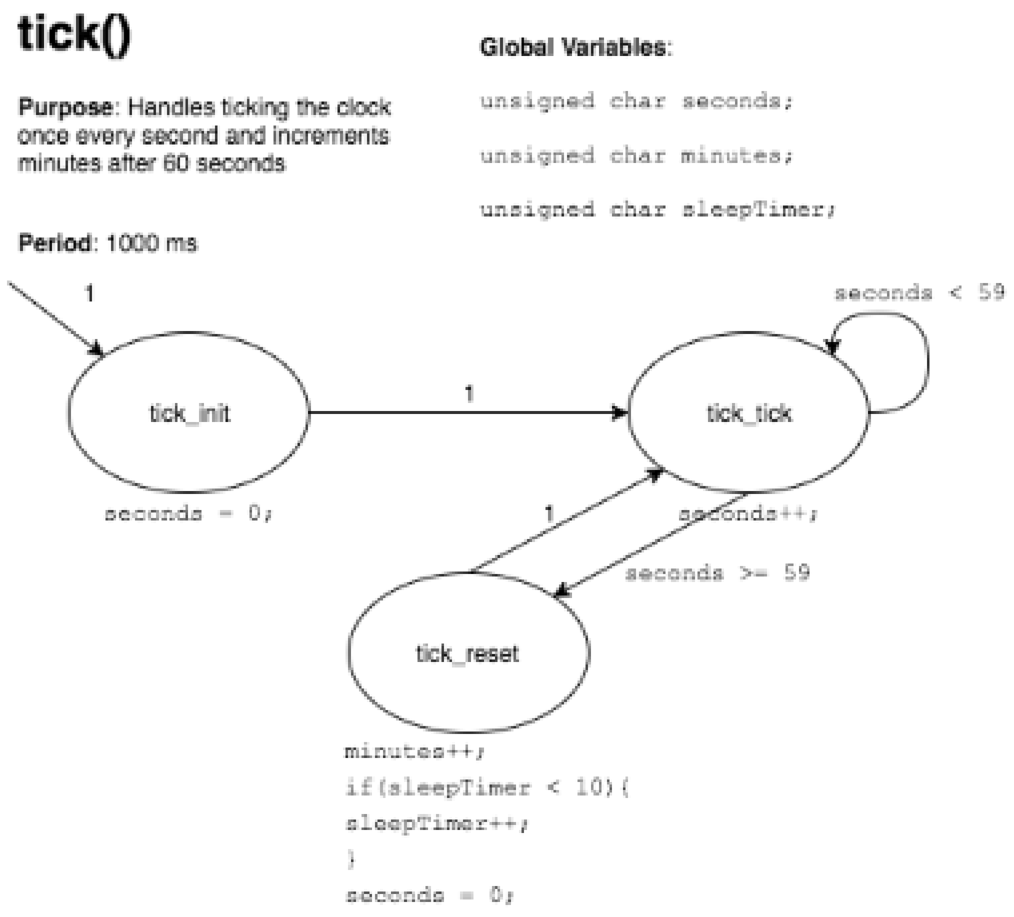
Extra Credit

A link to a public DIY for your project: link (Work in progress)

References

Inspiration for smart mirror : https://www.youtube.com/watch?v=J2S75AhPqnM

Appendix



tick()

Purpose: Handles ticking the clock once every second and increments minutes after 60 seconds

Period: 1000 ms

Global Variables:

```
unsigned char seconds;
unsigned char minutes;
unsigned char sleepTimer;
```

```
minutes++;
if(sleepTimer < 10){
sleepTimer++;
}
seconds = 0;
```

calculateTime()

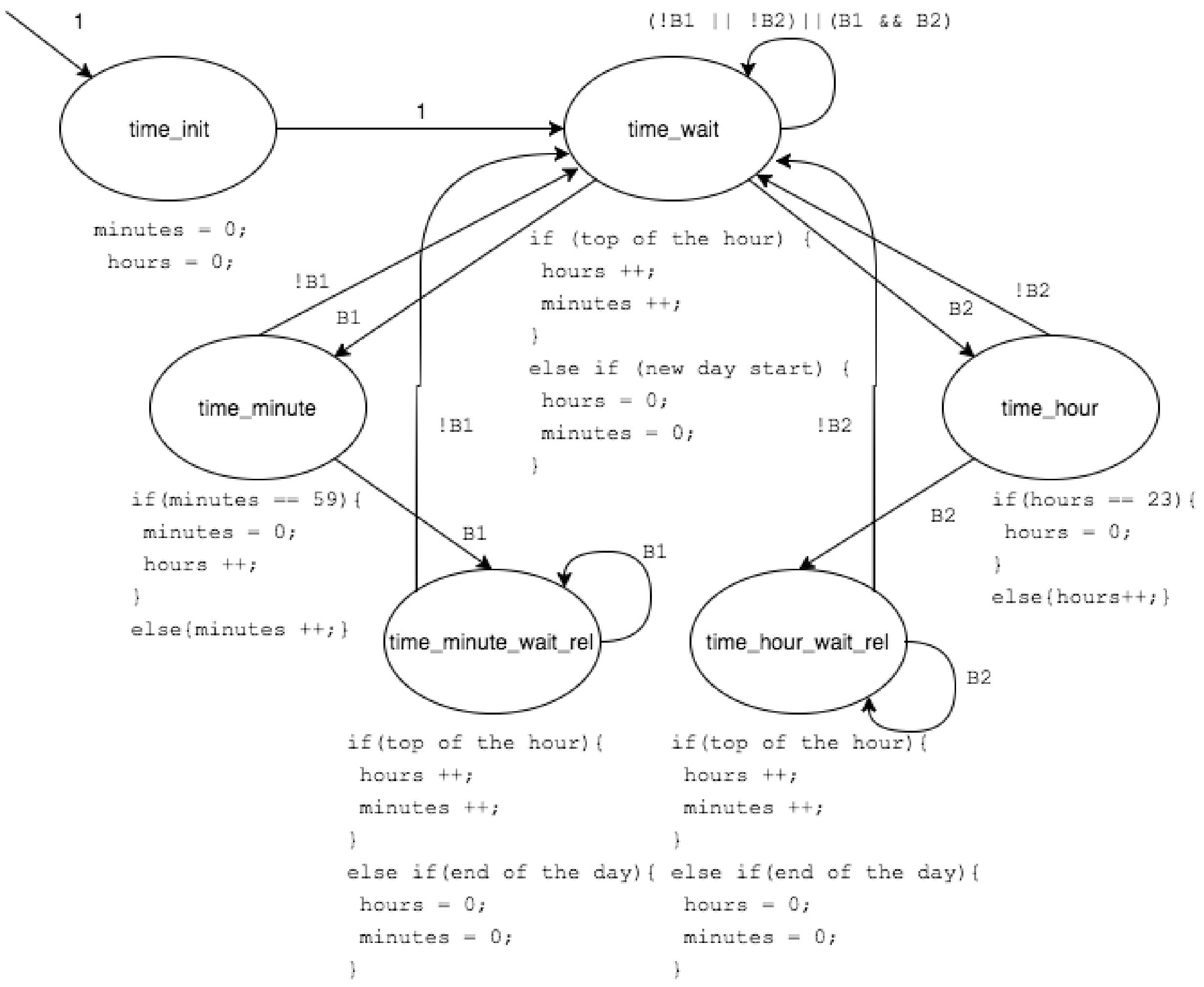
Purpose: Calculates the time using the tickFSM and uses B1 to increment minutes, B2 to increment hours

Period: 50 ms

Global Variables:

```
char* timeStr =

"00:00          Good Night!       ";
```



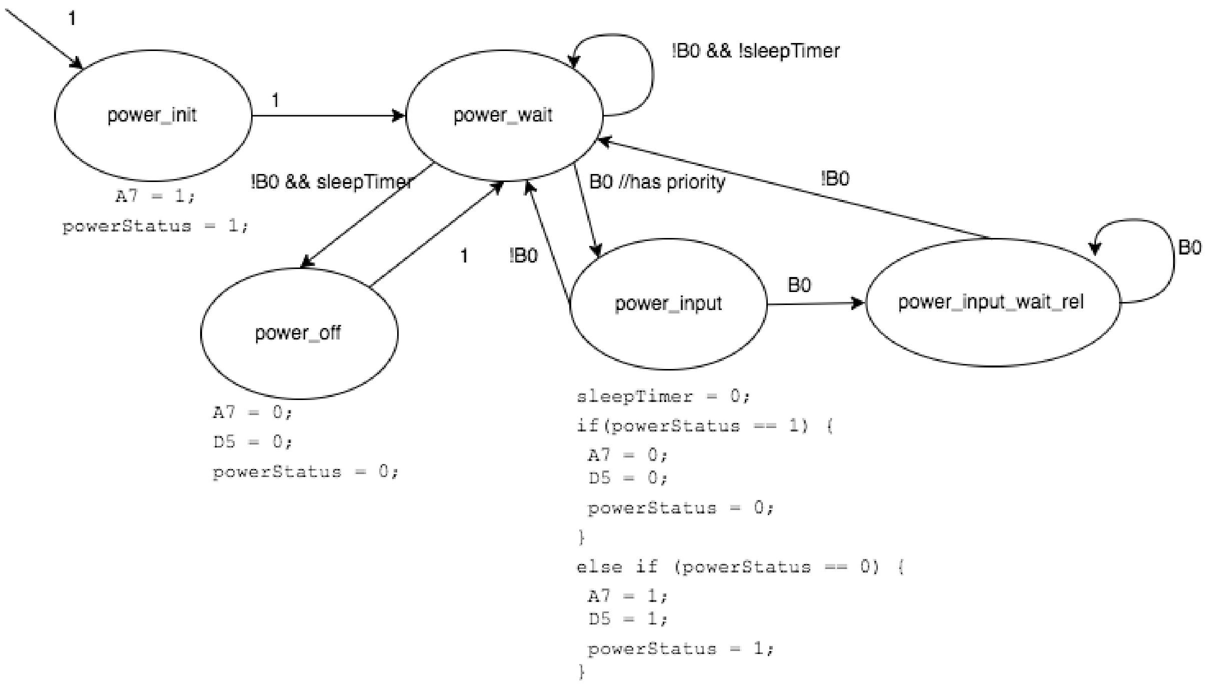
LCDPower()

Purpose: Turns on and off LCD display (controlled by PB0)

Global Variables:

```
unsigned char powerStatus;
unsigned char tempOff;
```

Period: 50 ms



```
A7 = 1;
powerStatus = 1;
```

```
A7 = 0;
D5 = 0;
powerStatus = 0;
```

```
sleepTimer = 0;
if(powerStatus == 1) {
 A7 = 0;
 D5 = 0;
 powerStatus = 0;
}
else if (powerStatus == 0) {
 A7 = 1;
 D5 = 1;
 powerStatus = 1;
}
```

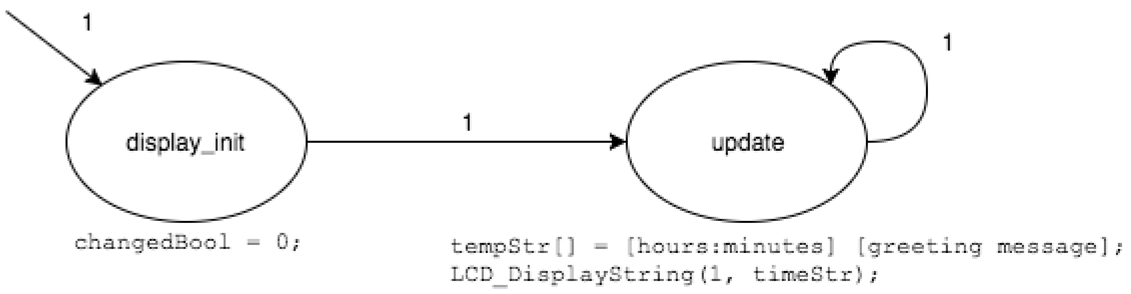
displayTime()

Purpose: Displays the time based on variables "hours" and "minutes" as "hours":"minutes"

Global Variables:

```
char changedBool;
char tempchar;
```

Period: 50 ms



```
changedBool = 0;
```

```
tempStr[] = [hours:minutes] [greeting message];
LCD_DisplayString(1, timeStr);
```

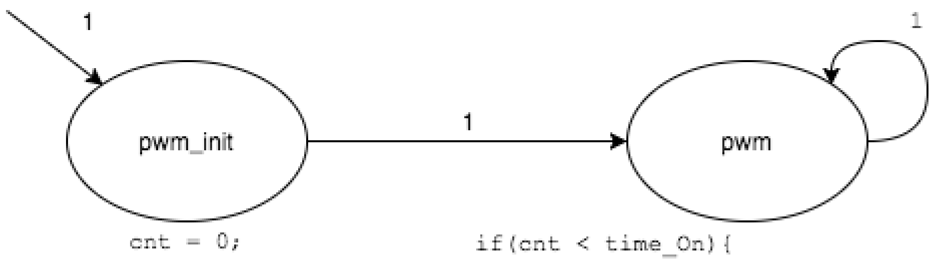
PWM_set()

Purpose: Pulse with modulation turns LEDs on and off to adjust brightness

Period: 1 ms

Global Variables:

```
unsigned short cnt;
```



```
cnt = 0;
```

```
if(cnt < time_On){
 B = 0xFF;
 cnt++;
}
else if( (cnt >= time_On) && (cnt < time_Off)){
 B = 0x00;
 cnt++;
}
else if(cnt == time_Off){
 B = 0x00;
 cnt=0;
}
```

adjustLight()

Purpose: Adjusts brightness of LEDs using buttons.

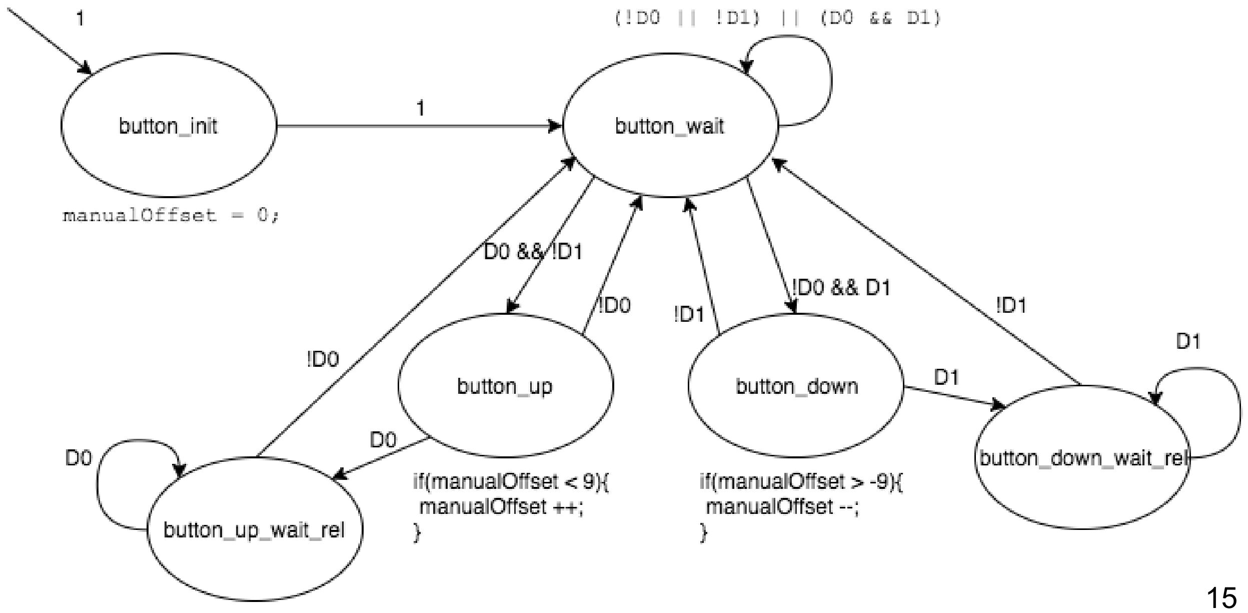
Period: 100 ms

Global Variables:

```
unsigned char seconds;

unsigned char minutes;

unsigned char sleepTimer;
```



```
manualOffset = 0;
```

```
if(manualOffset < 9){
 manualOffset ++;
}
```

```
if(manualOffset > -9){
 manualOffset --;
}
```

rainbowLights()

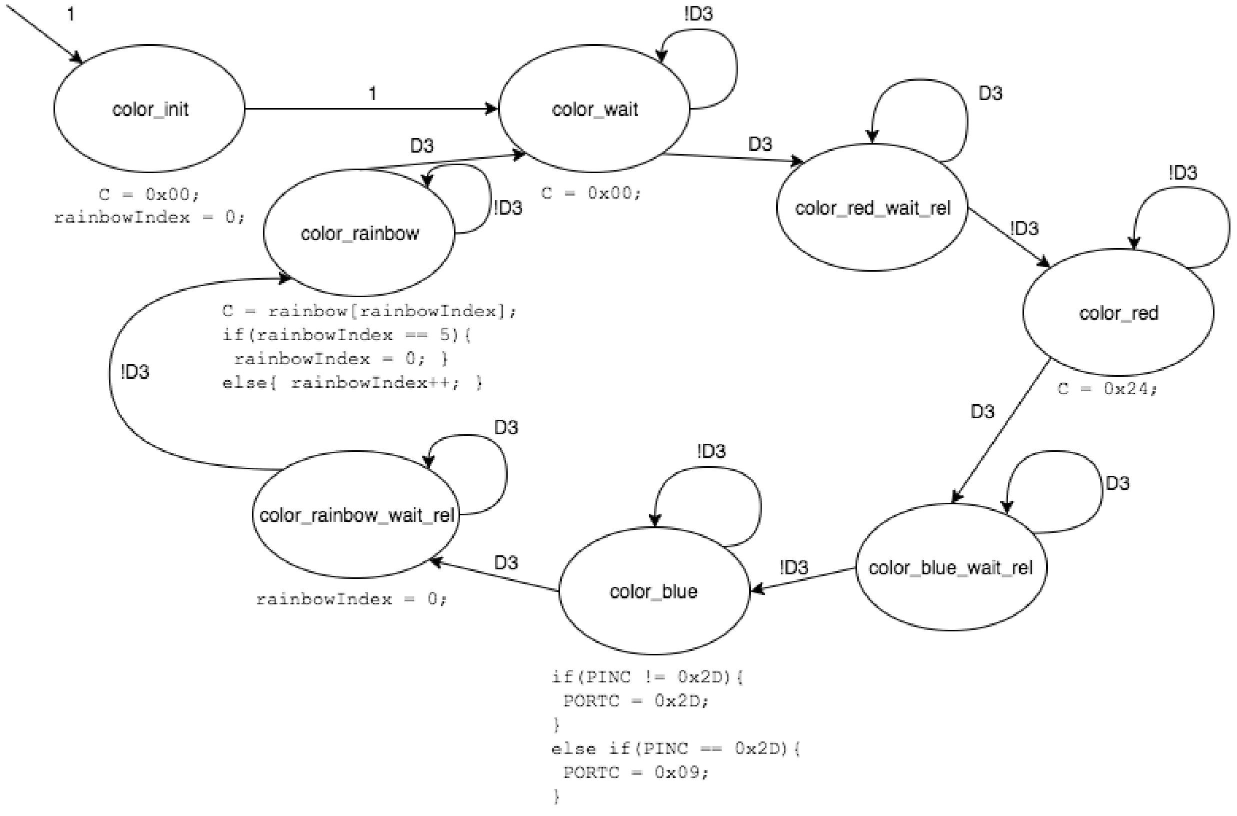
Purpose: Controls the RGB LEDs. Has three modes.

Period: 100 ms

Global Variables:

```
const char rainbow[6] = {0x24, 0x36,
0x12, 0x1B, 0x09, 0x2D};

unsigned char rainbowIndex;
```



cycleInputs()

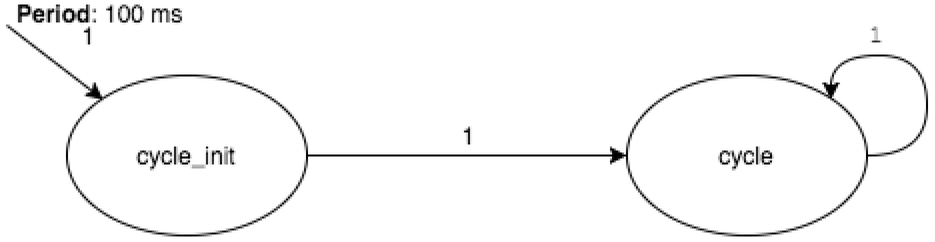
Purpose: Takes input from photoresistors and averages them. Then updates brightness values to change brightness of LEDs.

Global Variables:

```
unsigned short current_val;

unsigned short my_short;
```

Period: 100 ms



```
brightness = 1;
time_On = brightness;
time_Off = (21-brightness);
currentPin = 0x00;
Set_A2D_Pin(currentPin);
my_short = 0;
```

```
Set_A2D_Pin(currentPin);
current_val = ADC;
my_short = my_short  + (current_val / 8);

if(currentPin = 7){ //last pin
 brightness = ((my_short + 50) / 51);
 brightness += manualOffset;
 if(!D4){
  brightness = 0;
 }
 //time_on and time_off control the PWM
 time_On = brightness;
 time_Off = (21-brightness);
 my_short = 0;
 currentPin = 0; //cycle LDRs again
}
```

Github Repo: https://github.com/wendingoli/Mirror