EE2361 Final Project Library Documentation

Touchpad Library:

void chipWrite(unsigned char reg, unsigned char data);
Write to the CAP1188 by sending the start bit, slave address, register address location, data, and stop bit. The register address corresponds to the register that the user would like to change the setting of. The data is the value that the user wants to change the setting to (see page 38 of the CAP1188 datasheet for register addresses and data values).

void chipRead(unsigned char reg);
Send the start bit, then write the slave address. Following up with the register that the user would like to read the data from (see page 38 of the CAP1188 datasheet for register address), repeat the start sequence, and send the slave address with the read bit set to 1. Enable the receiver and read the first byte. Send the stop bit.

void configTouchpad(void);
Configure the CAP1188. First disable the multi-touch feature, as we do not need to detect multi-touches. Select the specific leds to pull high. We changed the sensitivity of each of the touchpads to be 1x's the default amount since the default sensitivity made touch detection too sensitive. The final configuration setting controls which capacitive touch pad is initialized and turned on.

Stepper Library:

void stepper_setup();
Sets up the necessary ports and latches needed to set the motors up, as well as sets the timer1 and timer2 to have a period of 1 second.

void drive_motor1() and void drive_motor2();
Both functions drive a PWM signal; when timer1 reaches its maximum period, the motor is driven off. drive_motor1() is responsible for driving the water gun and drive_motor2() is used to drive the lockbox motor.

void reset_motor1() and reset_motor2();
Perform a similar task as the drive functions, but drive the motors in reverse.

void sleep();
Set the motors to logic low to turn off each motor.

iLED library:

void writeColor1-4(int r, int g, int b);

The writeColor1-4 functions take three 8-bit RGB values. It first converts them to binary, from which the individual bits are stored within an array of size 8. From the array, the function checks for each individual bit that corresponds to a write_0 or write_1.

LCD Screen Library:

void lcd_init(void);

Within the library, the first function is the void lcd_init(void); function. In this function, several commands are executed in order to set up the oscillator frequency trim (internal LCD frequency), enabling the display driver, setting up temperature controlled follower, and finally adjusting the contrast by sending bit packages to the lcd_cmd function.

lcd_cmd(char Package);

Within the void lcd_cmd(char Package) function, it receives a write package, then setting the Start bit and continuously clearing the IFS3bits.MI2C2 interrupt flag before moving to the next step, setting the slave address and the R/nw bit. It continues to reset the IFS3bits.MI2C2 interrupt flag before moving to the next step. Following that, the next step is to send a control byte, then the data byte, then finally sending the Stop bit when complete.

lcd_setCursor(char x, char y);

lcd_setCursor(char x, char y) is fairly simple, it uses an equation where it multiplies the row by 0x40 then adds by the column then ors the result with 0x80 to set the MSB to 1 before sending out the result to lcd_cmd(result) in order to determine where the character or string is initially printed out on the LCD display.

void lcd_printChar(char myChar);

It receives the character that we want to print as an argument and begins similarly to the lcd_cmd function where it sends a Start bit before setting up the slave address as well as the write command to I2C2TRN. Afterwards within the control bit it differs by having the RS bit set to 1 (writing) instead of 0 like from the lcd_cmd function. Afterwards, it takes the inserted character as a package and sends it as a Package data byte before sending the Stop bit to signal that it is done. Like the lcd_cmd function, between each step, there is a step of waiting for the MI2C2IF to go to 1 before clearing the interrupt flag before going to the next step.

void lcd_printStr(const char s[]);

This function works about the same as the print char command except that it obviously needs several writes. This command begins by first storing the length of the string with strlen(s) in order to determine how many writes must be needed . After sending the Start bit, it also has to receive the slave address and the R/nW bit as well. Afterwards, within a while loop it sends a

control bit where both Co = 1 and RS = 1 as the Co bit indicates whether this is the first of consecutive writes or a series of writes while the RS bit tells that we are writing. Afterwards in the while loop, it takes the element of the array s[i] then sends it in as a data byte before incrementing i and checking the condition of the while loop to make sure that it is not the last element of the array. When it is the last byte/element of the array, it has to send the control byte once again but this time, the value of Co is set to 0, indicating that this data byte is the last of several writes while RS remains 1. After sending the last element of the array s[i] as the data byte, it sends the Stop bit as well indicating that it is done.