

```
// COMMAND CENTER PT. I: STATUS
// Jacqueline Yu and Aleenah Ansari
// Last Updated 6/2/2018
// HCDE 439

// This is the first part of the COMMAND CENTER site of the
PARTY MACHINE system

// This code allows the arduino to display the status
updates sent by the bubble machine
// and the control glove. A button is used to flip through
different status screens on an
// LCD screen. RGB lights also change according to the
screen being viewed.

// Button code:
// https://learn.adafruit.com/adafruit-arduino-lesson-6-digital-inputs/arduino-code
// To get everything running at the same time using
"Update" instead of delays:
// https://learn.adafruit.com/multi-tasking-the-arduino-part-1/all-together-now

// LCD characters:
// http://omerk.github.io/lcdchangen/
// RGB lights and colors:
// https://learn.adafruit.com/adafruit-arduino-lesson-3-rgb-leds?view=all
// XBee Parsing Code written by Mike Eacker
// Additional XBee code provided by Blake on Canvas

#include "Wire.h"
#include "Adafruit_LiquidCrystal.h"
#include <LiquidCrystal.h>
#include <SoftwareSerial.h> // for XBee

// Connect via i2c, default address #0 (A0-A2 not jumpered)
```

```
Adafruit_LiquidCrystal lcd(0);

const int buttonPin = 7; // button for flipping through the
status screens
const int RGBPinR = 13; // red
const int RGBPinG = 12; // green
const int RGBPinB = 11; // blue
const int numberOfScreens = 6; // total number of screens
to rotate through, doesn't include initial start screen
const bool DEBUG = true; //set debug mode for XBee messages

// indicates which status screen is currently being viewed
// number increases as button is pressed, determines which
screen to display, "current screen number"
int screen;
// to dim the RGB lights if needed
bool dim;
// to see if lcd needs to be updated
bool newUpdate;

// initialize the serial port for the XBee with whatever
pins are used for it
SoftwareSerial xBeeSerial(2, 3); // (TX, RX) : pins on XBee
adapter
int messages = 0; // counts the number of messages
that have been received

// the following two global variables are used by the XBee
reading method 'checkMessageReceived()' .
bool msgComplete = false; // true when an incoming message
is complete
String msg = ""; // buffer to collect incoming
message

RGB lights used are common anode
#define COMMON ANODE
```

```
//////////  
//////////  
  
//      Controller Variables  
  
// Indicates what kidn fo gesture is being made with the  
control glove  
int gestMode = 0; //gesture mode from controller:  
// 0 = none  
// 1 = "jitterbug" a swish of the hand, which causes the  
bubble machine  
// to "jitterbug" back and forth  
  
//  
  
// Indicates the number of bubble machines being used  
// Is controlled by flex sensors in the control glvoe  
// More flex = more bubble guns  
int bubMode = 0; //bubble output from controller:  
// 0 = none  
// 1 = level 1 (one bubble gun)  
// 2 = level 2 (two bubble guns)  
  
int sysPwr = 0; //system power, stop/start bubbles and  
rotation  
  
// Indicates the color of the lights on the bubble machine  
// Initialized to the random String/letter "a"  
String ledClr = "a";  
// roygbpw, red, orange, yellow, green, blue, pruple, white  
  
// Indicates the light patter being displayed by teh bubble  
machine  
// Initialized to -1  
int ledMode = -1;  
// -1 = none, initial, for debugging
```

```

// 0 = none
// 1 = move theatre lights
// 2 = flashing
// 3 = random lights

// Indicates the brightness of the LEDs on the bubble
machine
// Brightness is determined by a number between 0 and 100.
int ledBright = 0;
// 0 to 100

////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

// sets the color of the RGB lights
void setColor(int red, int green, int blue)
{
    // dims lights by 75%, if desirable
    if (dim)
    {
        red = red * 0.75;
        green = green * 0.75;
        blue = blue * 0.75;
    }
#endif COMMON_ANODE
    red = 255 - red;
    green = 255 - green;
    blue = 255 - blue;
#endif
    analogWrite(RGBPinR, red);
    analogWrite(RGBPinG, green);
    analogWrite(RGBPinB, blue);
}

```



```

};

byte block4[8] = {
    0b11110,
    0b11110,
    0b11110,
    0b11110,
    0b11110,
    0b11110,
    0b11110,
    0b11110
};

byte block5[8] = {
    0b11111,
    0b11111,
    0b11111,
    0b11111,
    0b11111,
    0b11111,
    0b11111,
    0b11111
};

////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

// parses rx data from controller into key-value pairs and
populates the appropriate variables
void parseRXData() {
    bool idOK = true; //is message sent from the desired
device ID?
    bool isKey = true; //tracks whether the item is a key or
value
    String device = "bc"; //ID this device accepts messages
from

```

```

String device2 = "bm"; //ID this device accepts messages
from

Serial.println("Parsing RX packet...");

String key = ""; //store temp key data
String value = ""; //store temp value data

//while serial port contains bytes (characters),
//add each char to string
while (xBeeSerial.available() > 0) {
    char c = xBeeSerial.read(); //read character and adds
to char

    //if message from the correct device ID (or not known
yet), parse data
    if (idOK == true) {
        //key data parsing
        if (isKey == true) {

            //if value data ends, switch to value data parsing
            if (c == ':') {
                isKey = false; //prepare to parse value for key
            }
            else {

                //if char is alphanumeric, add to string
                if (isAlphaNumeric(c) || c == ' ') {
                    key = key + c;

                }
            }
        }

        //process value data
    else {

```

```

        //if key data ends, switch to value data parsing
and update variable
    if (c == '|') {
        isKey = true; //prepare to parse next key

        //for ID, check if message from correct device
        if (key.equals("id")) {

            //++++++ DEBUG ++++++
            if (DEBUG == true) {
                Serial.print("key: "); Serial.println(key);
                Serial.print("value: "); Serial.

println(value);
            }

            //if invalid device ID received, set idOK to
false which will effectively trash the message
            if (!value.equals(device) && !value.
equals(device2)) {
                idOK = false; //id not okay, invalid device ]
            }
            //if valid device ID received
            else {
                // clear key-value to prepare for next
key-value pair
                key = "";
                value = "";
            }
        }

        //for all other key-value pairs
    else {
        updateParams(key, value); //update variables

        //++++++ DEBUG ++++++
    }
}

```

```

        if (DEBUG == true) {
            Serial.print("key: "); Serial.println(key);
            Serial.print("value: "); Serial.
            println(value);
        }

        // clear key-value to prepare for next
        key-value pair
        key = "";
        value = "";
    }
}
else {

    //if char is alphanumeric, add to string
    if (isAlphaNumeric(c) || c == ' ') {
        value = value + c;
    }
}
}

//process final key-value pair only if message from
correct device
if (idOK == true) {
    updateParams(key, value); //update final variable in
the message

    //+++++ DEBUG ++++++
    if (DEBUG == true) {
        Serial.print("key: "); Serial.println(key);
        Serial.print("value: "); Serial.println(value);
    }
}
}

```

```
//updates key parameters for the device with key-value
pairs received
void updateParams(String key, String value) {
    newUpdate = true;
    //***** change variables and value datatypes to fit
your needs *****
    if (key == "bubMode") {
        bubMode = value.toInt();
    }

    if (key == "gestMode") {
        gestMode = value.toInt();
    }

    if (key == "bmPause") {
        sysPwr = value.toInt();
    } //      Controller Variables

    if (key == "ledClr")
    {
        ledClr = value;
    }

    if (key == "ledMode")
    {
        ledMode = value.toInt();
    }

    if (key == "ledBrght")
    {
        ledBright = value.toInt();
    }
}
```

```
//////////  
//////////  
  
// Receives button input and changes the screen on the LCD  
if:  
// the button has been pushed  
// a designated amount of time has passed  
class buttonStatus  
{  
    int updateInterval;  
    unsigned long lastUpdate;  
  
public:  
    buttonStatus(int interval)  
    {  
        updateInterval = interval;  
    }  
  
    void Update()  
    {  
        if ((millis() - lastUpdate) > updateInterval)  
        {  
            lastUpdate = millis();  
            if (digitalRead(buttonPin) == LOW)  
            {  
                screen += 1;  
                if (screen > numberOfScreens) // reset screen to  
0 as soon as it goes through all screens  
                {  
                    screen = 0;  
                }  
            }  
            // Serial.println(screen);  
        }  
    }  
};
```

```
//////////  
//////////  
  
// controls the LCD display  
class lcdDisplay  
{  
    // variables keep track of when the LCD was last  
    updated, or the last  
    // time the button was pushed so the screen is not  
    constantly refreshing  
    int updateInterval;  
    unsigned long lastUpdate;  
    int lastButton;  
  
public:  
    lcdDisplay(int interval)  
    {  
        // Initializes variables. Last button is set to -1,  
        since it hasn't been pushed yet.  
        updateInterval = interval;  
        lastButton = -1;  
    }  
  
    void Update()  
    {  
        // LCD screen only updates/refresches if the button  
        has been pushed anew or  
        // if a certain amount of time has passed.  
        if ((millis() - lastUpdate) > updateInterval)  
        {  
            if (newUpdate || screen - lastButton != 0)  
            {  
                lastButton = screen;  
                lastUpdate = millis();  
                lcd.setCursor(0, 0);  
            }  
        }  
    }  
}
```

```

        lcd.clear();
        newUpdate = false;

        // method takes in the screen number being viewed
        Screens(screen);
    }
}

// manages the screens of the LCD, also changes the
// LED lights accordingly
void Screens(int screenNumber)
{
    Serial.println(screenNumber);
    if (screenNumber == 0)
    {
        // Displays general welcome screen on the LCD
        // prompts user to push the button
        setColor(255, 255, 255); // white
        lcd.setCursor(0, 0);
        lcd.print("Press Button to");
        lcd.setCursor(0, 1);
        lcd.print("View Status");
    } else if (screenNumber == 1)
    {
        // Displays the power setting of the bubble machine
        // either on or off
        setColor(0, 255, 0); // green
        lcd.setCursor(0, 0);
        lcd.print("Power:");
        // if/else
        if (sysPwr == 1)
        {
            lcd.setCursor(0, 1);
            lcd.print((char)5);
            lcd.print((char)5);

```

```
    lcd.print((char)5);
    lcd.print((char)5);
    lcd.print((char)5);
    lcd.print((char)5);
    lcd.print((char)5);
    lcd.print("ON");
    lcd.print((char)5);
    lcd.print((char)5);
    lcd.print((char)5);
    lcd.print((char)5);
    lcd.print((char)5);
    lcd.print((char)5);
    lcd.print((char)5);
    lcd.print((char)5);
    lcd.print((char)5);
} else {
    lcd.setCursor(0, 1);
    lcd.print("-----OFF-----");
}
} else if (screenNumber == 2)
{
    // Displays the type of gesture being
    // performed by the glove
    setColor(255, 0, 0); // red
    lcd.setCursor(0, 0);
    lcd.print("Gesture:");
    if (gestMode == 1)
    {
        lcd.setCursor(0, 1);
        lcd.print("    JITTERBUG    ");
    } else {
        lcd.setCursor(0, 1);
        lcd.print("    NO GESTURE    ");
    }
} else if (screenNumber == 3)
{
    // Displays how many bubble guns are in use
    setColor(0, 255, 255); // aqua
```

```
lcd.setCursor(0, 0);
lcd.print("Bubble Machines:");
if (bubMode == 1)
{
    lcd.setCursor(0, 1);
    lcd.print("        ONE        ");
} else if (bubMode == 2)
{
    lcd.setCursor(0, 1);
    //                lcd.print(bubMode);
    lcd.print("        TWO        ");
} else {
    lcd.setCursor(0, 1);
    lcd.print("        NONE       ");
}
} else if (screenNumber == 4)
{
    // Displays (in words) the current color of the
bubble
    // machine's lights
    setColor(225, 0, 225); // purple
    lcd.setCursor(0, 0);
    lcd.print("Light Color:");
    Serial.print(ledClr);
    if (ledClr == "r")
    {
        lcd.setCursor(0, 1);
        lcd.print("        RED        ");
    } else if (ledClr == "o")
    {
        lcd.setCursor(0, 1);
        lcd.print("        ORANGE      ");
    } else if (ledClr == "y")
    {
        lcd.setCursor(0, 1);
        lcd.print("        YELLOW      ");
    }
}
```

```

} else if (ledClr == "g")
{
    lcd.setCursor(0, 1);
    lcd.print("      GREEN      ");
} else if (ledClr == "b")
{
    lcd.setCursor(0, 1);
    lcd.print("      BLUE       ");
} else if (ledClr == "p")
{
    lcd.setCursor(0, 1);
    lcd.print("      PURPLE     ");
} else if (ledClr == "w")
{
    lcd.setCursor(0, 1);
    lcd.print("      WHITE      ");
} else {
    lcd.setCursor(0, 1);
    lcd.print("      NONE       ");
}

} else if (screenNumber == 5)
{
    // Displays the LED light pattern
    // being used by the bubble machine
    setColor(0, 0, 255); // blue
    lcd.setCursor(0, 0);
    lcd.print("Light Mode:");
    if (ledMode == 0) {
        lcd.setCursor(0, 1);
        lcd.print("      ALWAYS ON    ");
    } else if (ledMode == 1)
    {
        lcd.setCursor(0, 1);
        lcd.print("      MOVIE THEATER  ");
    } else if (ledMode == 2)

```

```

{
    lcd.print("      FLASHING      ");
} else if (ledMode == 3)
{
    lcd.print("      RANDOM      ");
} else {
    lcd.setCursor(0, 1);
    lcd.print("      NONE      ");
}
} else if (screenNumber == 6)
{
    // Displays the LED brightness as a bar
    // Brightness is measured on a scale of 0 to 100
    // That number is then converted into blocks, with
    // full block ((char)5) representing 10, and smaller
    // blocks representing 2, 4, 6, and 8
    // If there is no light, displays "No LIGHT"
    setColor(255, 255, 0); // yellow
    lcd.setCursor(0, 0);
    lcd.print("LED Brightness: ");
    // led bright
    int tens = ledBright / 10;
    int ones = ledBright % 10;

    if (ledBright < 1)
    {
        lcd.setCursor(0, 1);
        lcd.print("      NO LIGHT      ");
    } else {
        lcd.setCursor(0, 1);
        lcd.print("      ");
        lcd.setCursor(0, 1);

        for (int i = 0; i <= tens; i++)
        {
            lcd.print((char)5);

```

```
        }

        if (ones == 1)
        {
            lcd.print((char)1);
        } else if (ones == 2)
        {
            lcd.print((char)2);
        } else if (ones == 3)
        {
            lcd.print((char)3);
        } else if (ones == 4)
        {
            lcd.print((char)4);
        }
        lcd.setCursor(16, 1);
        lcd.print(ledBright);
    }

}

};

// calls the methods handling the button and LCD
// update interval set to 150
buttonStatus buttonPress(150);
lcdDisplay lcdStatus(150);

void setup() {
    // initialize button and RGB lights
    pinMode(buttonPin, INPUT_PULLUP);
    pinMode(RGBPinR, OUTPUT);
```

```

pinMode(RGBPinG, OUTPUT);
pinMode(RGBPinB, OUTPUT);
// start with prompt screen on LCD
screen = 0;
dim = false;
// set up the LCD's number of rows and columns:
lcd.begin(16, 2);
// create custom characters
lcd.createChar(1, block1);
lcd.createChar(2, block2);
lcd.createChar(3, block3);
lcd.createChar(4, block4);
lcd.createChar(5, block5);
//lcd.write((uint8_t)0);
lcd.setCursor(0, 0);
// print to serial
Serial.begin(9600);

// XBee
// initialize our own serial monitor window
Serial.println("\n\nxBeeReceive");

// initialize and set the data rate for the
SoftwareSerial port -- to send/receive messages via the xBee
xbeeSerial.begin(9600);
}

///////////////////////////////
/////////////////////////////
void loop() {

    // if XBee's serial port contains bytes (characters),
    send data to parse function
    if (xbeeSerial.available() > 0) {
        parseRXData();
}

```

```
//+++++ DEBUG ++++++
//***** change variables to fit your needs
*****
if (DEBUG == true) {
    Serial.println("Data parsing complete, updated
values:");
    Serial.print("Bubble Mode: "); Serial.
println(bubMode);
    Serial.print("Gesture Mode: "); Serial.
println(gestMode);
    Serial.print("System Status: "); Serial.
println(sysPwr);
}
}

// methods check to see if the button has received a new
signal,
// or if the LCD screen needs to be changed/refreshed
buttonPress.Update();
lcdStatus.Update();
}
```

```
// COMMAND CENTER PT. II: MUSIC
// Jacqueline Yu and Aleenah Ansari
// Last Updated 6/2/2018
// HCDE 439

// This is the second part of the COMMAND CENTER site of
the PARTY MACHINE system

// This code allows the Arduino to play music by using an
mp3 shield.
// It is not directly related to the command center-- we
just added it
// because we thought music would be fun.

// Music shield code:
// https://learn.adafruit.com/adafruit-music-maker-shield-vs1053-mp3-wav-wave-ogg-vorbis-player
// LCD characters:
// http://omerk.github.io/lcdchangen/

// include SPI, MP3 and SD libraries
#include <SPI.h>
#include <Adafruit_VS1053.h>
#include <SD.h>
#include "Wire.h"
#include "Adafruit_LiquidCrystal.h"

// Connect via i2c, default address #0 (A0-A2 not jumpered)
Adafruit_LiquidCrystal lcd(0);

// These are the pins used for the music maker shield
#define SHIELD_RESET -1           // VS1053 reset pin (unused!)
#define SHIELD_CS    7            // VS1053 chip select pin
(output)
#define SHIELD_DCS   6            // VS1053 Data/command select
```

```
pin (output)

// These are common pins between breakout and shield
#define CARDCS 4      // Card chip select pin
// DREQ should be an Int pin, see http://arduino.
cc/en/Reference/attachInterrupt
#define DREQ 3        // VS1053 Data request, ideally an
Interrupt pin

Adafruit_VS1053_FilePlayer musicPlayer =
    // create shield-example object!
    Adafruit_VS1053_FilePlayer(SHIELD_RESET,  SHIELD_CS,
SHIELD_DCS,  DREQ,  CARDCS);

// custom character is a "play" symbol (triangle)
byte customChar[8] = {
    0b01000,
    0b01100,
    0b01110,
    0b01111,
    0b01111,
    0b01110,
    0b01100,
    0b01000
};

// button is used to start the music
const int buttonPin = 7;
const int totalSongs = 10;

// structs used to store the names of the songs in the SD
card
// titles must be 8 characters long, followed by .mp3
char songs[totalSongs][15] =
{
    {"1this is.mp3"},
```

```

        { "bodakyel.mp3" },
        { "despacit.mp3" },
        { "electric.mp3" },
        { "gods_pl_.mp3" },
        { "KOD_____.mp3" },
        { "never_be.mp3" },
        { "proof___.mp3" },
        { "readyfor.mp3" },
        { "sweater_.mp3" },
    } ;

// variables for if the music is on or off, as well as if
the current
// playlist is finished playing.
bool onOff;
bool done;

////

void setup() {
    // serial monitor for debugging
    Serial.begin(9600);
    // set up lcd and custom character
    lcd.begin(16, 2);
    lcd.createChar(0, customChar);
    lcd.setCursor(0, 0);
    pinMode(buttonPin, INPUT_PULLUP);

    Serial.println("Adafruit VS1053 Simple Test");

    // making sure shield + SD card are connected properly
    if (! musicPlayer.begin()) { // initialise the music
player
        Serial.println(F("Couldn't find VS1053, do you have the
right pins defined?"));
        while (1);
    }
}

```

```

}

Serial.println(F("VS1053 found"));

if (!SD.begin(CARDCS)) {
    Serial.println(F("SD failed, or not present"));
    while (1); // don't do anything more
}

// list files
// printDirectory(SD.open("//"), 0);

// Set volume for left, right channels. lower numbers ==
// louder volume!
musicPlayer.setVolume(20, 20);

// Timer interrupts are not suggested, better to use DREQ
interrupt!
//musicPlayer.useInterrupt(VS1053_FILEPLAYER_TIMER0_INT);
// timer int

// If DREQ is on an interrupt pin (on uno, #2 or #3) we
can do background
// audio playing
musicPlayer.useInterrupt(VS1053_FILEPLAYER_PIN_INT); // // DREQ int

// initialize variables: music is off (false) and the
playlist
// is not done (false)
onOff = false;
done = false;

// print instructions to LCD
lcd.print("-Press to Start-");
Serial.println("-Press to Start-");
}

```

```
void loop() {  
  
    // button press starts the music  
    if (digitalRead(buttonPin) == HIGH) {  
        onOff = false;  
    }  
    if (digitalRead(buttonPin) == LOW) {  
        onOff = true;  
    }  
  
    if (onOff)  
    {  
        // plays each song in order  
        // prints the title of the .mp3 file  
        // to the LCD  
        for (int i = 0; i < totalSongs; i++)  
        {  
            Serial.print("playing ");  
            Serial.println(songs[i]);  
            //lcd stuff  
            lcd.clear();  
            lcd.setCursor(0, 0);  
            lcd.print(" ");  
            lcd.print(songs[i]);  
            lcd.print(" ");  
  
            lcd.setCursor(0, 1);  
            lcd.print(" ");  
            lcd.print((char)0);  
            lcd.print(" ");  
  
            musicPlayer.playFullFile(songs[i]);  
        }  
    }  
}
```

```

        done = true;
    }

    // if the playlist is finished playing,
    // reset the LCD screen to prompt the user
    // to start the music over
    if (done) {
        done = false;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("-Press to Start-");
        Serial.println("-Press to Start-");
    }

    delay(100);
}

// File listing helper
// can be used to debug, prints the names of the
// files in the SD card to Serial
void printDirectory(File dir, int numTabs) {
    while (true) {

        File entry = dir.openNextFile();
        if (! entry) {
            // no more files
            //Serial.println("**nomorefiles**");
            break;
        }
        for (uint8_t i = 0; i < numTabs; i++) {
            Serial.print('\t');
        }
        Serial.print(entry.name());
        if (entry.isDirectory()) {
            Serial.println("/");
        }
    }
}

```

```
    printDirectory(entry, numTabs + 1);
} else {
    // files have sizes, directories do not
    Serial.print("\t\t");
    Serial.println(entry.size(), DEC);
}
entry.close();
}
}
```