Cs122a\_p2\_final\_project.ino

// Blynk device template information

#define BLYNK\_TEMPLATE\_ID "TMPL2FEVyWppL"

#define BLYNK\_TEMPLATE\_NAME "Quickstart Device"

#define BLYNK\_AUTH\_TOKEN "3e8SJ427t5bXDpBCkXc1ni6jjobGn-lp"

// Blynk prints, can be commented out.

#define BLYNK\_PRINT Serial

//Pins for the LEDs

#define BLUE 5

#define RED 6

#define WHITE 7

#define NOTE\_B0 31

#define NOTE\_C1 33

#define NOTE\_CS1 35

#define NOTE\_D1 37

#define NOTE\_DS1 39

#define NOTE\_E1 41

#define NOTE\_F1 44

#define NOTE\_FS1 46

#define NOTE\_G1 49

#define NOTE\_GS1 52

#define NOTE\_A1 55

#define NOTE\_AS1 58

#define NOTE\_B1 62

#define NOTE\_C2 65

#define NOTE\_CS2 69

#define NOTE\_D2 73

#define NOTE\_DS2 78

#define NOTE\_E2 82

#define NOTE\_F2 87

#define NOTE\_FS2 93

#define NOTE\_G2 98

#define NOTE\_GS2 104

#define NOTE\_A2 110

#define NOTE\_AS2 117

#define NOTE\_B2 123

#define NOTE\_C3 131

#define NOTE\_CS3 139

#define NOTE\_D3 147

#define NOTE\_DS3 156

#define NOTE\_E3 165

#define NOTE\_F3 175

#define NOTE\_FS3 185

#define NOTE\_G3 196

#define NOTE\_GS3 208

#define NOTE\_A3 220

#define NOTE\_AS3 233

#define NOTE\_B3 247

#define NOTE\_C4 262

#define NOTE\_CS4 277

#define NOTE\_D4 294

#define NOTE\_DS4 311

#define NOTE\_E4 330

#define NOTE\_F4 349

#define NOTE\_FS4 370

#define NOTE\_G4 392

#define NOTE\_GS4 415

#define NOTE\_A4 440

#define NOTE\_AS4 466

#define NOTE\_B4 494

#define NOTE\_C5 523

#define NOTE\_CS5 554

#define NOTE\_D5 587

#define NOTE\_DS5 622

#define NOTE\_E5 659

#define NOTE\_F5 698

#define NOTE\_FS5 740

#define NOTE\_G5 784

#define NOTE\_GS5 831

#define NOTE\_A5 880

#define NOTE\_AS5 932

#define NOTE\_B5 988

#define NOTE\_C6 1047

#define NOTE\_CS6 1109

#define NOTE\_D6 1175

#define NOTE\_DS6 1245

#define NOTE\_E6 1319

#define NOTE\_F6 1397

#define NOTE\_FS6 1480

#define NOTE\_G6 1568

#define NOTE\_GS6 1661

#define NOTE\_A6 1760

#define NOTE\_AS6 1865

#define NOTE\_B6 1976

#define NOTE\_C7 2093

#define NOTE\_CS7 2217

#define NOTE\_D7 2349

#define NOTE\_DS7 2489

#define NOTE\_E7 2637

#define NOTE\_F7 2794

#define NOTE\_FS7 2960

#define NOTE\_G7 3136

#define NOTE\_GS7 3322

#define NOTE\_A7 3520

#define NOTE\_AS7 3729

#define NOTE\_B7 3951

#define NOTE\_C8 4186

#define NOTE\_CS8 4435

#define NOTE\_D8 4699

#define NOTE\_DS8 4978

#define REST 0

//Baud rate for ESP8266 ((data rate in bits per second))

#define ESP8266\_BAUD 115200

#include <ESP8266\_Lib.h>

#include <BlynkSimpleShieldEsp8266.h>

#include <SoftwareSerial.h>

#include "ZeldaLullaby.h"

//WIFI Credentials

char ssid[] = "V-WIFI";

char pass[] = "veronica91";

// change this to make the song slower or faster

int tempo = 108;

// change this to whichever pin you want to use

int buzzer = 9;

// notes of the moledy followed by the duration.

// a 4 means a quarter note, 8 an eighteenth , 16 sixteenth, so on

// !!negative numbers are used to represent dotted notes,

// so -4 means a dotted quarter note, that is, a quarter plus an eighteenth!!

int melody[] = {

// Zelda's Lullaby - The Legend of Zelda Ocarina of Time.

// Score available at https://musescore.com/user/12754451/scores/2762776

NOTE\_E4,2, NOTE\_G4,4,

NOTE\_D4,2, NOTE\_C4,8, NOTE\_D4,8,

NOTE\_E4,2, NOTE\_G4,4,

NOTE\_D4,-2,

NOTE\_E4,2, NOTE\_G4,4,

NOTE\_D5,2, NOTE\_C5,4,

NOTE\_G4,2, NOTE\_F4,8, NOTE\_E4,8,

NOTE\_D4,-2,

NOTE\_E4,2, NOTE\_G4,4,

NOTE\_D4,2, NOTE\_C4,8, NOTE\_D4,8,

NOTE\_E4,2, NOTE\_G4,4,

NOTE\_D4,-2,

NOTE\_E4,2, NOTE\_G4,4,

NOTE\_D5,2, NOTE\_C5,4,

NOTE\_G4,2, NOTE\_F4,8, NOTE\_E4,8,

NOTE\_F4,8, NOTE\_E4,8, NOTE\_C4,2,

NOTE\_F4,2, NOTE\_E4,8, NOTE\_D4,8,

NOTE\_E4,8, NOTE\_D4,8, NOTE\_A3,2,

NOTE\_G4,2, NOTE\_F4,8, NOTE\_E4,8,

NOTE\_F4,8, NOTE\_E4,8, NOTE\_C4,4, NOTE\_F4,4,

NOTE\_C5,-2,

};

// sizeof gives the number of bytes, each int value is composed of two bytes (16 bits)

// there are two values per note (pitch and duration), so for each note there are four bytes

int notes = sizeof(melody) / sizeof(melody[0]) / 2;

// this calculates the duration of a whole note in ms

int wholenote = (60000 \* 4) / tempo;

int divider = 0, noteDuration = 0;

int ledState = 0;

//Software Serial for Uno R3

SoftwareSerial EspSerial(2, 3);

//Define baud rate

ESP8266 wifi(&EspSerial);

BlynkTimer timer;

typedef struct task {

int state;

unsigned long period;

unsigned long elapsedTime;

int (\*TickFct)(int);

} task;

int delay\_gcd;

const unsigned short tasksNum = 2;

task tasks[tasksNum];

enum SM1\_States { SM1\_INIT, SM1\_S0, SM1\_S1};

int SM1\_Tick(int state){

switch (state) { // State transitions

case SM1\_INIT:

state = SM1\_S0;

break;

case SM1\_S0:

state = SM1\_S0;

break;

case SM1\_S1:

break;

}

switch (state) { // State Action

case SM1\_INIT:

break;

case SM1\_S0:

Blynk.run();

break;

case SM1\_S1:

break;

}

return state;

}

enum SM2\_States { SM2\_INIT, SM2\_S0, SM2\_S1};

int SM2\_Tick(int state){

switch (state) {

case SM2\_INIT:

state = SM2\_S0;

break;

case SM1\_S0:

state = SM1\_S0;

break;

case SM2\_S1:

break;

}

switch (state) {

case SM2\_INIT:

break;

case SM2\_S0:

timer.run();

break;

case SM2\_S1:

break;

}

return state;

}

BLYNK\_WRITE(V1) {

ledState = param.asInt();

ledControl();

Serial.print("ledState after digitalWrite: ");

Serial.println(ledState);

}

void ledControl(){

if (ledState == 0) {

digitalWrite(RED,LOW);

digitalWrite(BLUE,LOW);

digitalWrite(WHITE,LOW);

} else {

digitalWrite(RED,HIGH);

digitalWrite(BLUE,HIGH);

digitalWrite(WHITE,HIGH);

song();

}

}

void song(){

// iterate over the notes of the melody.

// Remember, the array is twice the number of notes (notes + durations)

for (int thisNote = 0; thisNote < notes \* 2; thisNote = thisNote + 2) {

// calculates the duration of each note

divider = melody[thisNote + 1];

if (divider > 0) {

// regular note, just proceed

noteDuration = (wholenote) / divider;

} else if (divider < 0) {

// dotted notes are represented with negative durations!!

noteDuration = (wholenote) / abs(divider);

noteDuration \*= 1.5; // increases the duration in half for dotted notes

}

// we only play the note for 90% of the duration, leaving 10% as a pause

tone(buzzer, melody[thisNote], noteDuration\*0.9);

// Wait for the specief duration before playing the next note.

delay(noteDuration);

// stop the waveform generation before the next note.

noTone(buzzer);

}

}

void lightShow()

{

unsigned long previousMillis = 0; // Variable to store the previous time

const unsigned long interval = 500; // Interval between LED state changes (in milliseconds)

int ledStat = LOW; // Initial LED state

while (ledState == 1) {

unsigned long currentMillis = millis(); // Get the current time

if (currentMillis - previousMillis >= interval) {

// It's time to change the LED state

previousMillis = currentMillis; // Save the current time

// Toggle the LED state

ledStat = (ledStat == LOW) ? HIGH : LOW;

// Set the LED state for all three LEDs

if (ledState == 1) {

digitalWrite(RED, ledStat);

digitalWrite(BLUE, ledStat);

digitalWrite(WHITE, ledStat);

}

}

}

}

void playZeldaLullaby() {

for (int thisNote = 0; thisNote < notes \* 2; thisNote = thisNote + 2) {

divider = melody[thisNote + 1];

if (divider > 0) {

noteDuration = (wholenote) / divider;

} else if (divider < 0) {

noteDuration = (wholenote) / abs(divider);

noteDuration \*= 1.5;

}

tone(buzzer, melody[thisNote], noteDuration \* 0.9);

delay(noteDuration);

noTone(buzzer);

}

}

void setup()

{

Serial.begin(115200);

EspSerial.begin(ESP8266\_BAUD);

delay(10);

Blynk.begin(BLYNK\_AUTH\_TOKEN, wifi, ssid, pass);

unsigned char i = 0;

tasks[i].state = SM1\_INIT;

tasks[i].period = 500;

tasks[i].elapsedTime = tasks[i].period;

tasks[i].TickFct = &SM1\_Tick;

i++;

tasks[i].state = SM2\_INIT;

tasks[i].period = 500;

tasks[i].elapsedTime = tasks[i].period;

tasks[i].TickFct = &SM2\_Tick;

delay\_gcd = 500;

}

void loop()

{

unsigned char i;

for (i = 0; i < tasksNum; ++i) {

if ( tasks[i].elapsedTime >= tasks[i].period) {

tasks[i].state = tasks[i].TickFct(tasks[i].state);

tasks[i].elapsedTime = 0;

}

tasks[i].elapsedTime += delay\_gcd;

}

delay(delay\_gcd);

}

Wiring Diagram

