Demo Video: https://youtu.be/v7jB_Q0p_W0

Code: https://github.com/Grant-Hymes/Entrance-Announcinator-2-Electric-Boogaloo

Final Project (include in a ZIP file, submit on Canvas by 5/3/2023)

- All code (all .c and .h files for project)
- Copy of the PowerPoint presentation used during the project demo
- Written documentation of the library developed
    - Intro - Description of the Library and its overall functions
    - Hardware description - what device(s), part numbers, links, etc.
    - Full documentation - All public functions (arguments and outputs)
    - Basic usage example - the bare minimum to test the functionality of the hardware
    - Advanced usage example - covering all the functions and features
- Video of the demonstration of the project

Intro

The main part of the program runs all the setup functions for the various hardware components, and has the ability to customize the following: pin connections for each device in their respective init functions, the color that the LED turns for each state (void writeColor(int r, int g, int b)), and the behavior of the device when each button is pressed.

An enum is used to run the different states that the device can be in, and the device switches between the states using the external interrupts from the buttons (located in main) or from the status polling for the PIR motion sensor in the while(1) loop. The enum has 5 different states: startup, ready, armed, tripped, and boom.

The external interrupt for the ‘self-destruct’ is set to have the highest priority, and does not allow for transitions into other states. The self-destruct brings the device into the ‘boom’ mode.

The external interrupt for the arming button will toggle the device between the ready, armed, and tripped states. The device only gets put into the tripped state when the polling from the motion sensor detects something, and this is only after the device has been put into the armed mode. Pressing the arming button can bring it out of the tripped state and into the armed state.

The output of the device can be customized for each of the states, such as changing the color of the LED and playing a Song struct-using play_music(struct Song s)-or playing a single note-using play_note(char[] note, int seconds).

Song structs can also be written here, which are gone more into detail in the full documentation.

Hardware description

PIC24FJ64GA002
- Basic 16 bit microcontroller, heart of the project

WS2812 RGB LED
- Basic RGB LED, uses proprietary communication system

LIDAR Sensor: The VL53L1X time-of-flight sensor
- Time of Flight LIDAR Sensor, uses I^2C for communication

Adafruit PIR Motion Sensor
- Analog Motion Sensor, outputs a voltage upon detection

Breadboard Compatible Headphone Jack
- Allows plugging in a 3.5 mm compatible audio device to the breadboard

Breadboard Compatible Button
- Basic button, allows current to flow when pressed.

Full documentation

- Button Library
  - void initButtons(int pin1, int pin2);
    - Initializes external interrupt of both given pins pin must be between 6 and 9
    - The interrupts are initialized to use negative edge detection and a pull up resistor is set at both pins. If the button is pressed, the interrupt is triggered.
    - The interrupts themselves and their exact usages must be setup by the user
- LED Library
  - void writeColor(int r, int g, int b);
    - Sets the LED attached to pin RA0 to the given color.
    - r, g and b values can be between 0 and 255.
    - For example writeColor(255,0,0); would set the LED to be red
- LIDAR Library
  - void lidar_init(void);
    - Initializes the LIDAR library. Should be called before any usage of the LIDAR. Starts continuous measurements every 50 ms in long distance mode. Requires the LIDAR to be connected to SCL2 and SDA2.
  - uint16_t lidar_read_distance(void);
    - Reads a distance value from the LIDAR in millimeters. Blocking. If this call fails, it will never return. In this version it will always fail, therefore it should not be called.
- PIR Motion Sensor Library
  - void initMotionSensor(int pin);
    - Initializes Input Capture on the given pin, pin must be between 6 and 9. Timer 2 is setup for this input capture and status and timeTripped variables are created.
  - int status;
    - Simple integer variable, stores a 0 if the trap has not been tripped and a 1 if it has. It is assumed that this variable is used using polling within a main function, The variable should be reset to 0 every time the polling loop activates
  - int timeTripped;
    - Simple integer variable, stores the last time that the PIR motion sensor was tripped, Stores this value in second since startup

- Music Library
    - struct Song;
        - The Song struct packages together everything needed to play a song in play_music, an integer for the tempo in bpm, an integer for the number of notes in a song, and an array of 3 length char arrays with length 128. Each 3 character char array represents a note. In order, the characters represent the number of beats to hold the note, the note letter name, and the octave of the note. Using play_music, The char representing the number of beats to hold the note must be ‘0’ through ‘9’, ‘a’ through ‘w’, or ‘A’ through ‘W’. ‘a’/’A’ both represent 10 and ‘w’/’W’ both represent 32. Anything else will default to 0 and the note will be almost instantly skipped. The chars representing note name and octave are limited as described in set_note. Setting the length higher than 128 will cause issues. The minimum tempo for the PIC24 at 16MHz is 58 bpm, the maximum tempo is just over 1 million.
    - int play_music(struct Song s);
        - Plays all the notes in the given song struct at the song tempo. Songs that do not have a rest at the end will continue to hold the last note until it is set to something else. Notes are played by calling set_note on each note. The hold amount of the previous note and the tempo determine the amount of time to wait until the next note is played.
    - void set_note(char note, int octave);
        - Sets the output compare to a square wave with the correct period for the given note and octave. Valid input for note is limited to ‘C’, ‘d’, ‘D’, ‘e’, ‘E’, ‘F’, ‘g’, ‘G’, ‘a’, ‘A’, ‘b’, ‘B’, and ‘ ‘, otherwise the period will be set to an undesired value. Natural notes are represented by capital letters, flat notes are represented by lowercase letters, and sharps must be translated to the natural or flat equivalent. Any integer can be given however, the minimum octave available is one meaning anything lower will be set to one and a high enough octave will eventually default the period to 0.
    - void init_speaker(int pin);
        - Initializes output compare on the given pin pin must be between 6 and 9. The pin must be between 6 and 9 inclusive. Timer 3 is set up for the output compare and timer 5 is set up for handling song tempo.
    - void play_note(char note[], int seconds);
        - Calls set_note(note, octave) twice, with a delay in between. First call sets the desired note to be played taken from an input of a 2-character string, looking only at the first two characters in the string. A valid input for this would look like “C3” or “d3”, with the first char being the note name and the second char being the octave number. set_note(note, octave) will take care of the inputs being validated. After the first call, delay_ms2() is called with an input of 1000 * seconds. delay_ms2() is an internal utility for the music library. set_note(‘ ‘, 3) is then called to turn off the audio.

Basic usage example

```
…

Struct Song {
    Int tempo;
    Int size;
    Char notes[128][3];
};

Struct Song tune {100,3,{"5A3","5G3","1 3"}};

void setup(void) {
    CLKDIVbits.RCDIV = 0;
    AD1PCFG = 0x9fff;
}

int main(void) {
    uint16_t distance;

    setup();
    init_speaker(8);
    initMotionSensor(9);
    lidar_init();

    writeColor(255, 0, 0);

    delay_ms(100);

    /* distance = lidar_read_distance; currently broken, don’t use */

    while (1) {
        while(status == 0);
        Status = 0;
        play_music(tune);
        writeColor(0, 0, 255);
    }
}

// the code above tests the PIR motion sensor, and the music system
// when the motion sensor detects motion, the LED color changes and a short
// tune is played, tune doesn’t hold any significance, just two random
notes
```

Advanced usage example

```
...

enum mode {
      startup = 0,
      ready = 1,
      armed = 2,
      tripped = 3,
};

Struct Song {
    Int tempo;
    Int size;
    Char notes[128][3];
};

Struct Song tune {100,2,{"5A3","5G3","1 3"}};

enum mode curMode;

void setup(void) {
    CLKDIVbits.RCDIV = 0;
    AD1PCFG = 0x9fff;
}

int main(void) {
    uint16_t distance;

    setup();

    writecolor(0,255,0);
    curMode = startup;

    init_speaker(8);
    initMotionSensor(9);
    initButtons(7,6);
    lidar_init();

    writeColor(255, 0, 0);
    curMode = ready;
```

```
    delay_ms(100);

    /* distance = lidar_read_distance; currently broken, don't use */


    while (1) {
        while(status == 0);
        Status = 0;

        if(curMode == armed && timeTripped > 10) {

            play_music(tune);
            writeColor(0, 0, 255);
        }
    }
}

void __attribute__((__interrupt__,__auto_psv__)) _INT1Interrupt(void) {
    _INT1IF = 0;

    If (curMode == ready) { curMode = armed;}

    Else if (curMode == armed;) { curMode = ready;}

    Else if (curMode == tripped;) { curMode = armed;}

    play_note("A3", 1);

}



// the code above uses each function to test their functionality, except
// set_note().
// The code implements the PIR motion sensor, button and music generator.
// The device has 4 modes, startup, ready, armed and tripped.
// When the PIR motion sensor detects motion the status variable is changed
// to a 1. If the mode is armed and its been 10 seconds since startup, the
// short tune is played, additionally a single note is played upon each
// button press to show the mode is changing.
```