CHRISTMAS SINGING LIGHTS

A project by Canache Livia-Teodora

WORKING SPACE AND DEVELOPEMENT APPS USED

The „Christmas Singing Lights" Project was created using an Arduino UNO motherboard, one passive buzzer (passive to permit to modify the frequencies of the singing notes), 9 LEDs that copy the flashing lights of a christmas guirlande and each of them consists of a serial connected resistance in order to don't burn the LED, a HC-SR04 ultrasonic motion sensor that detect if there is anything in the range of 10 cm and allows the buzzer and LEDs to turn on and a button that changes through the 4 chosen christmas songs, connected to a PULL-UP resistance in order to avoid electric magnetic interferences (EMI) to interact with our button. All of these components are connected on a breadboard (see Figure 1.).

The programming environment used for this project was „Arduino IDE 2.3.4". The whole code project it's presented below.

PROJECT'S AND WIRING SCHEME

In order to create a look up of my initial project, I used a dedicated site for Arduino simulation (see „References. 1."), where I designed a prototype and tested out each of my poject's components, so I'd have a better understanding of them in my further implementation.

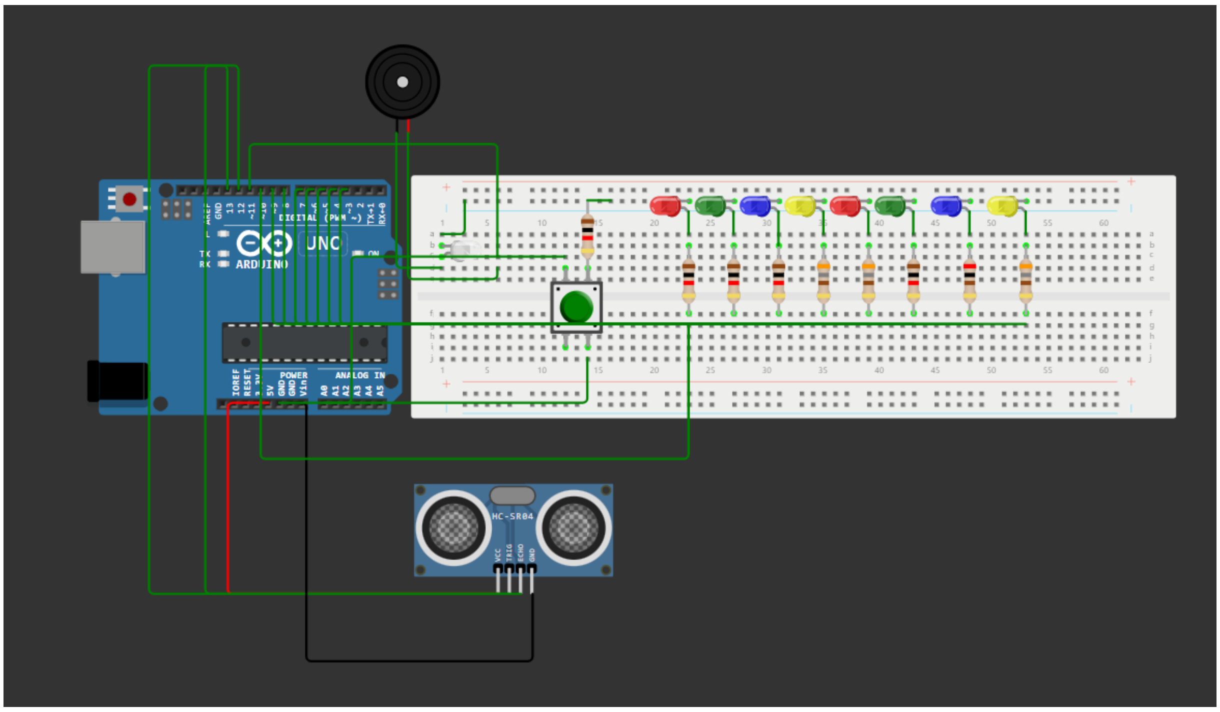
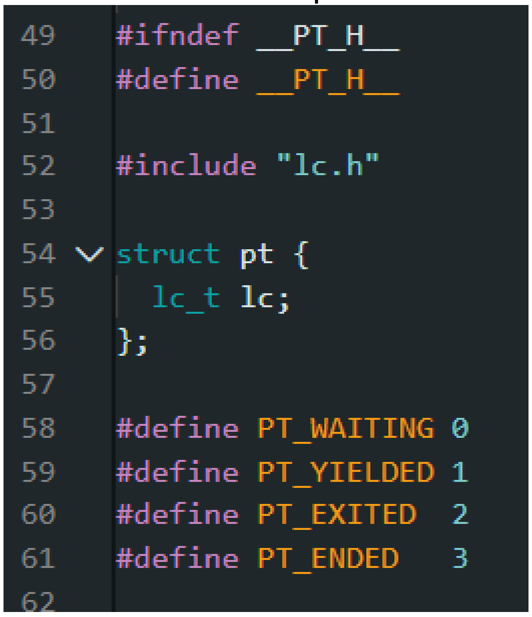


Figure 1. Project's simulation

THE CODE

Initialization:

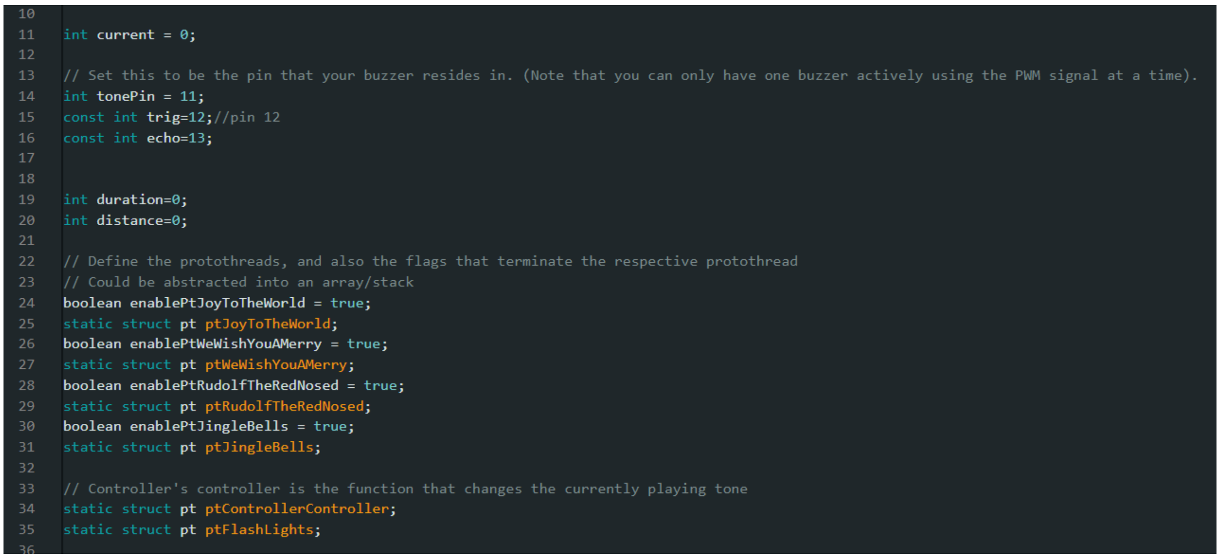
The program contains the necessary libraries **header files,** that were downloaded from the original source code for this project. Those header files contain a few protothread functions that operate the songs in the main file.



```
#ifndef __PT_H__
#define __PT_H__

#include "lc.h"

struct pt {
  lc_t lc;
};

#define PT_WAITING 0
#define PT_YIELDED 1
#define PT_EXITED  2
#define PT_ENDED   3
```

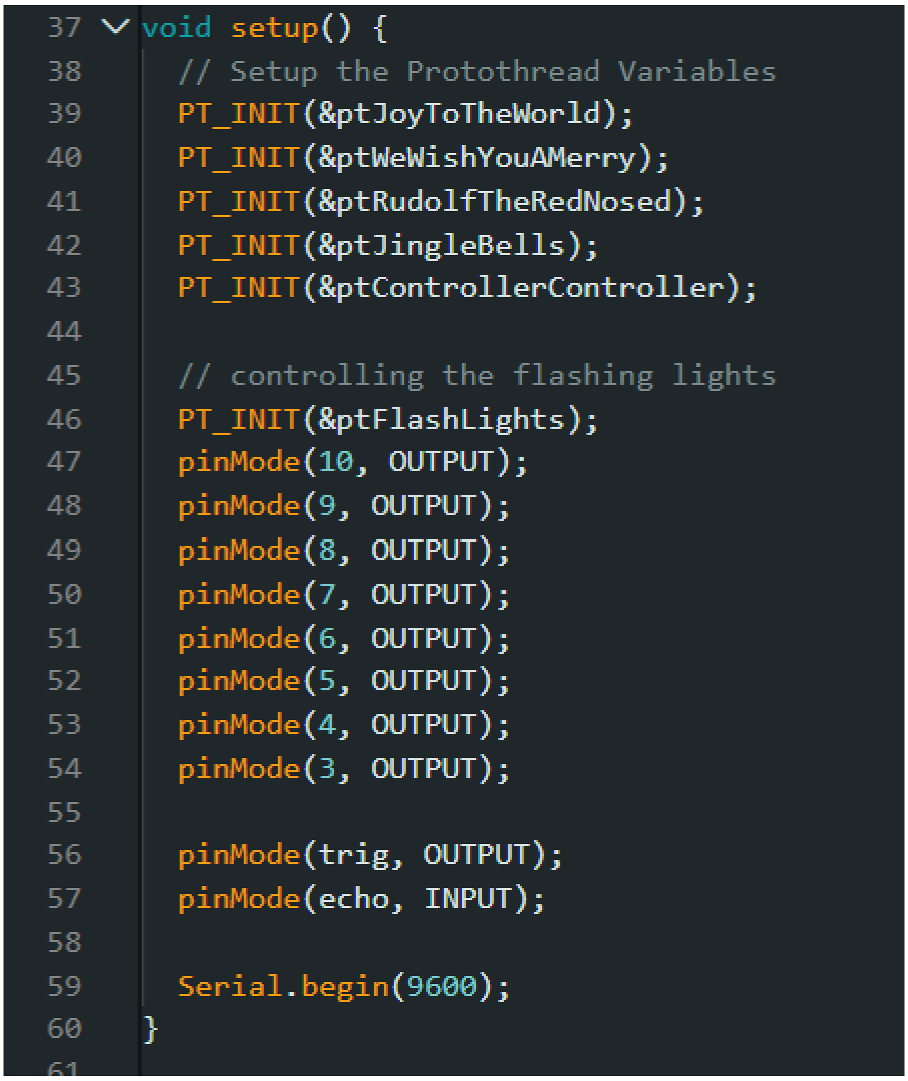
Figure 2. The definitions of the protothreads in „pt.h” file

Starting with the first lines of the code, all of the parameters used are initialised and defined.



```

int current = 0;

// Set this to be the pin that your buzzer resides in. (Note that you can only have one buzzer actively using the PWM signal at a time).
int tonePin = 11;
const int trig=12;//pin 12
const int echo=13;


int duration=0;
int distance=0;

// Define the protothreads, and also the flags that terminate the respective protothread
// Could be abstracted into an array/stack
boolean enablePtJoyToTheWorld = true;
static struct pt ptJoyToTheWorld;
boolean enablePtWeWishYouAMerry = true;
static struct pt ptWeWishYouAMerry;
boolean enablePtRudolfTheRedNosed = true;
static struct pt ptRudolfTheRedNosed;
boolean enablePtJingleBells = true;
static struct pt ptJingleBells;

// Controller's controller is the function that changes the currently playing tone
static struct pt ptControllerController;
static struct pt ptFlashLights;

```

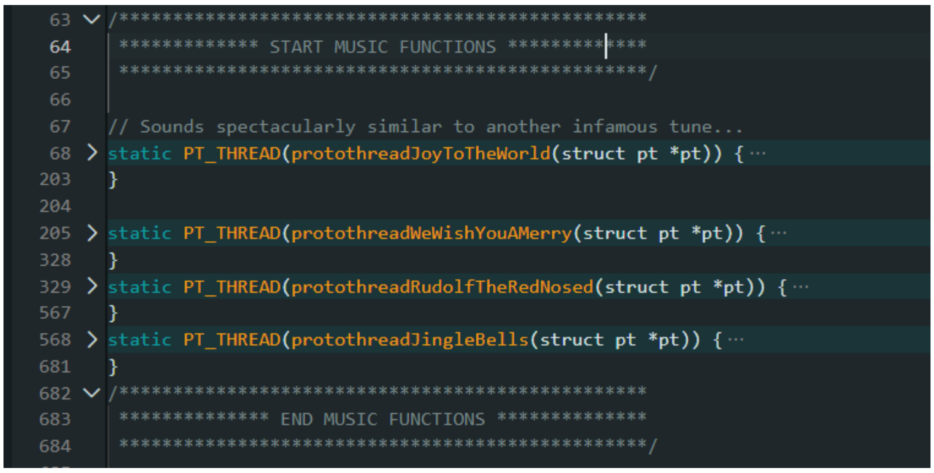
Then in the „void setup” function we set up the protothread variables and we prepare the flashing lights control.



```
void setup() {
  // Setup the Protothread Variables
  PT_INIT(&ptJoyToTheWorld);
  PT_INIT(&ptWeWishYouAMerry);
  PT_INIT(&ptRudolfTheRedNosed);
  PT_INIT(&ptJingleBells);
  PT_INIT(&ptControllerController);

  // controlling the flashing lights
  PT_INIT(&ptFlashLights);
  pinMode(10, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(3, OUTPUT);

  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);

  Serial.begin(9600);
}
```

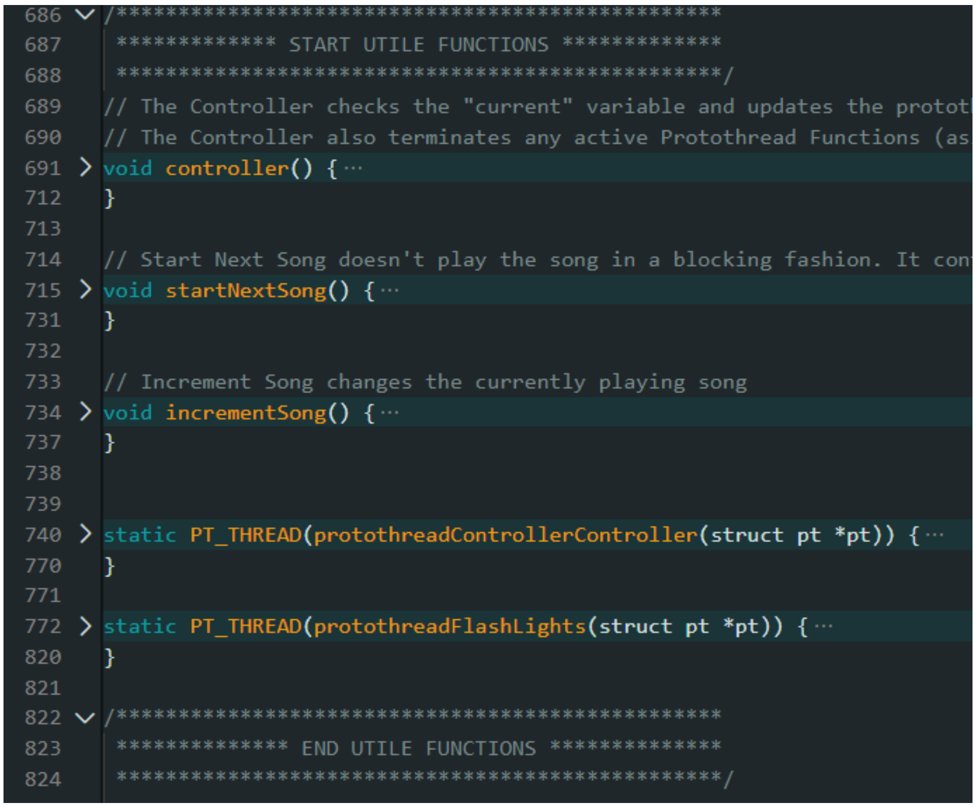
Control functions:

In order to create the christmas tracks: „Track 0: Joy to the World; Track 1: We Wish you a Merry Christmas; Track 2: Rudolf the Red Nosed Reindeer; Track 3: Jingle Bells", the original author of this project implemented for each song, a protothread function that has a static timer object to track note durations, which uses the „timer_set" and „timer_expired" functions manage delays. For the main execution, the function „PT_BEGIN(pt)" initializes the protothread and notes are played sequentially using the tone function, „tone(tonePin, frequency, duration)" generates a tone at tonePin with a specified frequency and duration, after playing a note, the code waits for the timer to expire before proceeding to the next note. Additional short delays (e.g., 5.20833333333 ms) create pauses between specific notes for better rhythm. „PT_END(pt)" marks the end of the protothread, after the melody finishes, „startNextSong()" is called to initiate another song. Because of the small capacity of an Arduino uno, the protothreads structures are used, they are lightweight, stackless threads that simplify cooperative multitasking.



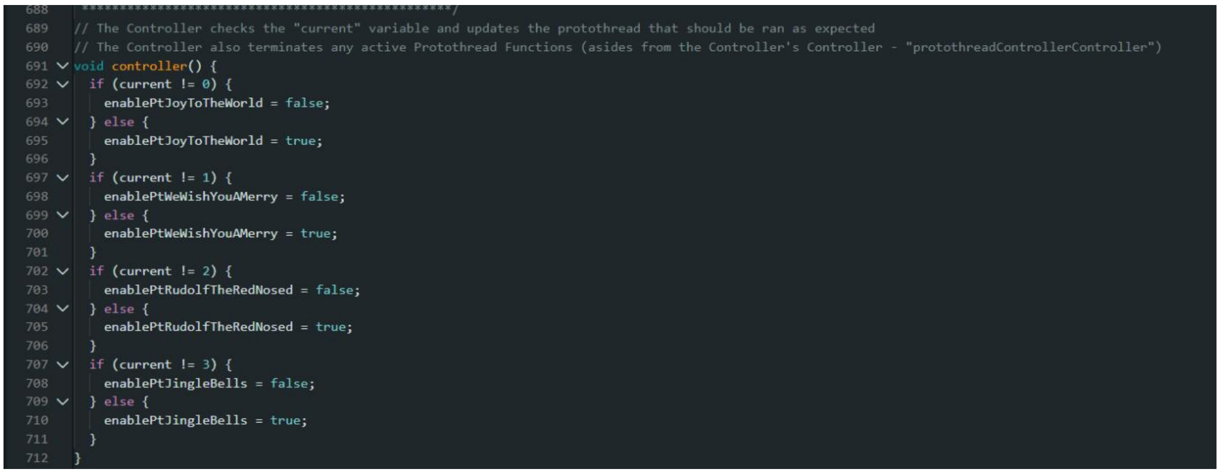
```
/**************************************************
 ************* START MUSIC FUNCTIONS *************
 **************************************************/

// Sounds spectacularly similar to another infamous tune...
static PT_THREAD(protothreadJoyToTheWorld(struct pt *pt)) {⋯
}

static PT_THREAD(protothreadWeWishYouAMerry(struct pt *pt)) {⋯
}
static PT_THREAD(protothreadRudolfTheRedNosed(struct pt *pt)) {⋯
}
static PT_THREAD(protothreadJingleBells(struct pt *pt)) {⋯
}
/**************************************************
 ************* END MUSIC FUNCTIONS *************
 **************************************************/
```

From here „the utile” functions start.



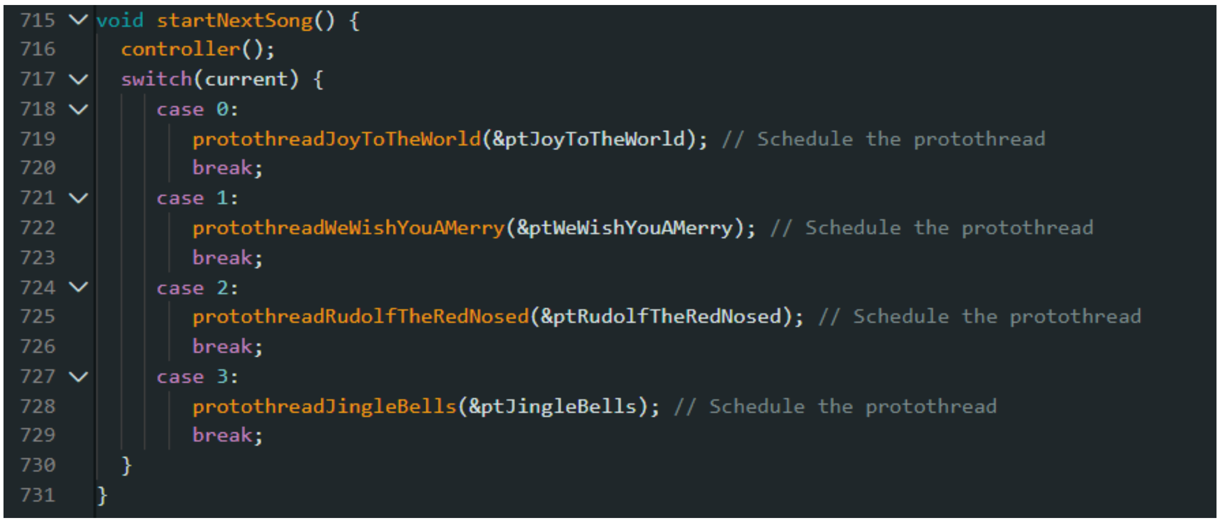
```
/**************************************************
 ************* START UTILE FUNCTIONS *************
 **************************************************/
// The Controller checks the "current" variable and updates the protot
// The Controller also terminates any active Protothread Functions (as
void controller() {⋯
}

// Start Next Song doesn't play the song in a blocking fashion. It con
void startNextSong() {⋯
}

// Increment Song changes the currently playing song
void incrementSong() {⋯
}


static PT_THREAD(protothreadControllerController(struct pt *pt)) {⋯
}

static PT_THREAD(protothreadFlashLights(struct pt *pt)) {⋯
}

/**************************************************
 ************* END UTILE FUNCTIONS *************
 **************************************************/
```

The „controller” function enables the song change through the push button, using the „current” variable and a „if else” case structure.



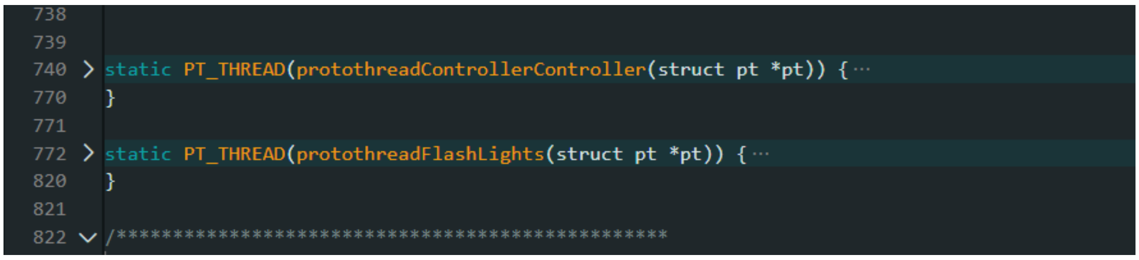
```
 **************************************************/
// The Controller checks the "current" variable and updates the protothread that should be ran as expected
// The Controller also terminates any active Protothread Functions (asides from the Controller's Controller - "protothreadControllerController")
void controller() {
  if (current != 0) {
    enablePtJoyToTheWorld = false;
  } else {
    enablePtJoyToTheWorld = true;
  }
  if (current != 1) {
    enablePtWeWishYouAMerry = false;
  } else {
    enablePtWeWishYouAMerry = true;
  }
  if (current != 2) {
    enablePtRudolfTheRedNosed = false;
  } else {
    enablePtRudolfTheRedNosed = true;
  }
  if (current != 3) {
    enablePtJingleBells = false;
  } else {
    enablePtJingleBells = true;
  }
}
```

„startNextSong” it's the function that can be found at the end of the song protothreads. This function enables the other song to start. In this function we use the protothreads and a switch case structure.



```
void startNextSong() {
  controller();
  switch(current) {
      case 0:
        protothreadJoyToTheWorld(&ptJoyToTheWorld); // Schedule the protothread
        break;
      case 1:
        protothreadWeWishYouAMerry(&ptWeWishYouAMerry); // Schedule the protothread
        break;
      case 2:
        protothreadRudolfTheRedNosed(&ptRudolfTheRedNosed); // Schedule the protothread
        break;
      case 3:
        protothreadJingleBells(&ptJingleBells); // Schedule the protothread
        break;
  }
}
```

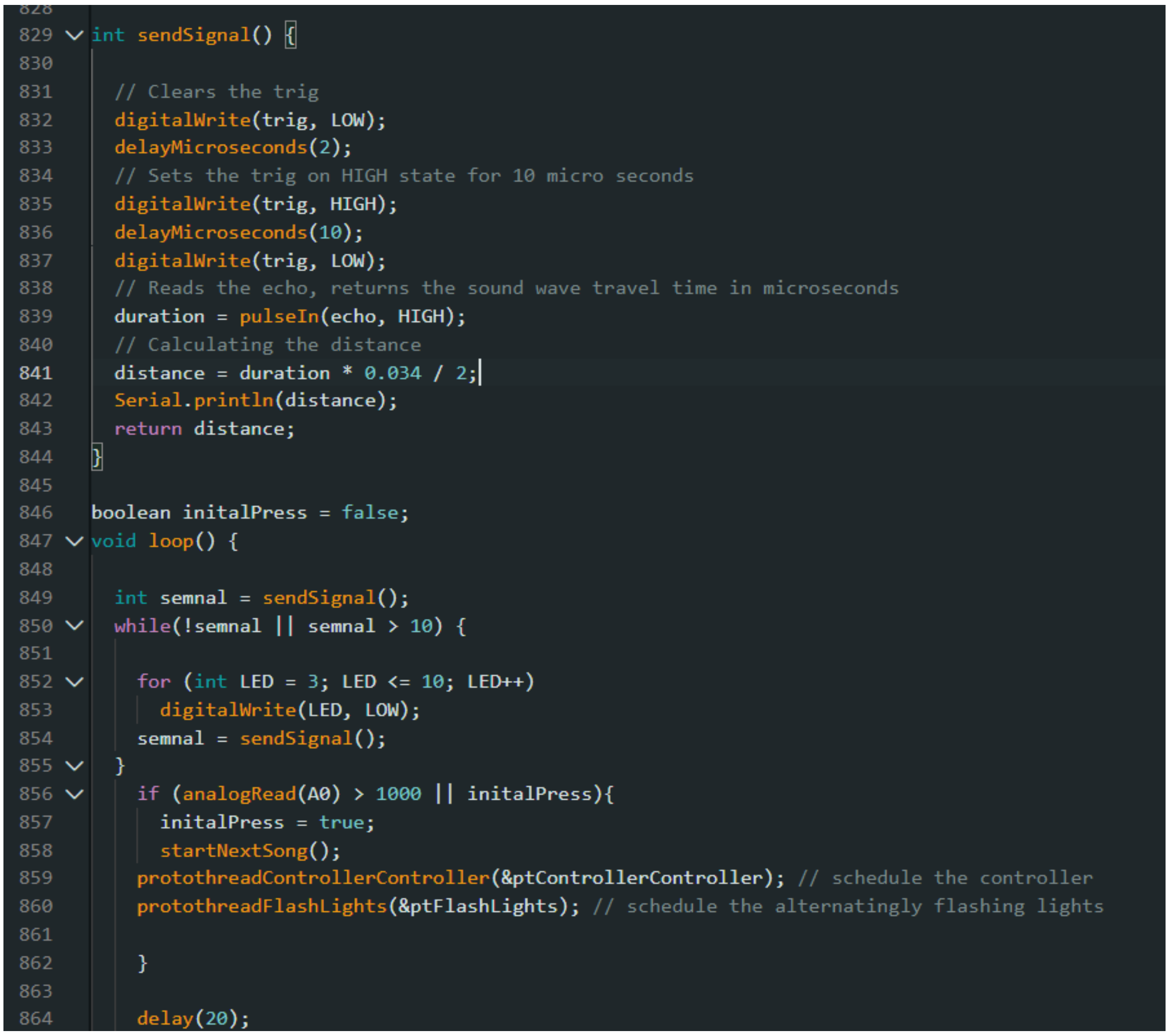
The „incrementSong” function just increments the „current” variable and divides it to 4, representing the number of the songs.



```


static PT_THREAD(protothreadControllerController(struct pt *pt)) {···
}

static PT_THREAD(protothreadFlashLights(struct pt *pt)) {···
}

/**************************************************
```

Then the other two protothread functions, „protothreadControllerController” and „protothreadFlashLights” are defined. The first one, monitors a button connected to an analog pin (A0) to change the currently playing song in a system. The thread runs in an infinite loop, waiting for a button press (detected by an analog reading exceeding a threshold of 1000). When pressed, it calls „incrementSong” to switch to the next song. To ensure a smooth transition, it introduces a blocking delay of 1 second during which music playback stops and festive lights are turned off. Finally, the protothreads for the various songs are reinitialized to reset their timing and prepare them for the next execution. The second one defines a protothread function, protothreadFlashLights, that continuously flashes a set of festive lights connected to digital pins. The thread runs in an infinite loop, toggling the states of specific pins (3–10) in a sequence to create a light pattern. After each pattern is set, a timer is initialized with a 500-millisecond delay, and the thread waits until the timer expires before proceeding to the next light configuration.

The last utile function „sendSignal” implements the ultrasonic motion sensor inputs and calculates the distance and the duration of the motion.



```
int sendSignal() {

  // Clears the trig
  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  // Sets the trig on HIGH state for 10 micro seconds
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);
  // Reads the echo, returns the sound wave travel time in microseconds
  duration = pulseIn(echo, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2;
  Serial.println(distance);
  return distance;
}

boolean initalPress = false;
void loop() {

  int semnal = sendSignal();
  while(!semnal || semnal > 10) {

    for (int LED = 3; LED <= 10; LED++)
      digitalWrite(LED, LOW);
    semnal = sendSignal();
  }
    if (analogRead(A0) > 1000 || initalPress){
      initalPress = true;
      startNextSong();
    protothreadControllerController(&ptControllerController); // schedule the controller
    protothreadFlashLights(&ptFlashLights); // schedule the alternatingly flashing lights

    }

    delay(20);
```

In the main function „loop” continuously checks for an object detected by an ultrasonic sensor (sendSignal()) within a 10 cm range. If no object is detected, all LEDs (pins 3 to 10) are turned off, and the sensor signal is rechecked. If a button press is detected (analogRead(A0) > 1000) or a flag initalPress is set, it starts a new song and schedules two protothreads: one to manage the controller and another to handle flashing lights. A 20-millisecond delay ensures the loop operates at a consistent pace.

References:

1.https://wokwi.com/projects/418606842357346305

2. https://github.com/jlunaing/Arduino-Christmas-Card-with-Music/blob/main/Christmas-Project.ino

3.https://www.youtube.com/watch?v=bJjCK7uszys

4.https://www.youtube.com/watch?v=znzHw-jDzMM&t=44s

5.https://github.com/fernandomalmeida/protothreads-arduino