Audio Player

Report and documentation

Information Technology 4th Semester
Thursday 10:00 Am
Group ID: F3

Team members:
Szymon Ruciński 217860 **(Team leader)**
Anna Preczyńska 217858
Iga Wójcik 217867

Table of contents

Task assignment to team members:

Team member	Assigned task
Szymon Ruciński (Team Leader)	- Memory card reader - Loudspeaker integration - Testing - Writing code - Creating report
Anna Preczyńska	- LCD display - Testing - Creating report
Iga Wójcik	- Bluetooth module - Testing - Creating report

Components and peripherals:

- Arduino Uno Rev3
- i2C converter integrated into Display LCD 1602
-HC-05 Bluetooth module
-AVR DSP 51 Card reader
-2Gb SD card
-8 Ohm, 0.5 Watt speaker

1.Project description

1.1 General

The main goal of the project was to install appropriate card in the SD module, use its contents to play audio file and display its name on the LCD screen. Additional feature is the ability to control the device via telephone connected to the device using a bluetooth module.

1.2 User's Guide

1.2.1 Setting up a memory card

In order to play music directly from an SD card one has to format it to an operable filesystem. In our case this will be FAT32 and FAT16 file format. Only usage of those formats guarantees it's full functionality and capabilities. It's not desirable to format SD cards frequently, as it shortens their life span. Format can be done via a computer no matter what OS system is being used.

1.2.2 Uploading songs

To make sure that any user will not lose his sleep over technical configurations, we were doing our best to provide them with possibly the most enjoyable experience. Selecting your favourite music and downloading should be your biggest concern. Right after doing so you just need to simply drag&drop it onto the SD card.

1.2.3 Supported audio formats

To play your favourite music on Audio Player your audio files have to meet some specific technical requirements. Due to Arduino technical limitations the only supported audio format is:
-Uncompressed 8bit Microsoft Wave file with „.WAV” file extension.
Our device supports 8000Hz sampling rate and mono sound settings.
Uncompressed audio format with reduced bitrate automatically eliminates decoding process which demands huge computational power.

1.2.4 Bluetooth pairing

Bluetooth pairing takes place using mobile application on the mobile phone with bluetooth module. To set up the connection there is necessary to select the HC-05 device from the list of available devices. The connection key is either „0000” or „1234”. After proper connection there is a difference in the functioning of the diode located on the bluetooth module. With continuous blinking, indicating the lack of a paired control device, the diode began to blink in the sequence of two consecutive blinks, repeated every 2 seconds. To control the device using a mobile application, all you have to do is enter the right letters in the application terminal. There is no specific application, which will be compatible with the device, because most of applications recommended to control Arduino using bluetooth module and proposed by GooglePlay are matching with the requirements of the device.

1.2.5 Interface

If the properly set up SD card with chosen music is connected to the device via SD card module and if the device is connected to the mobile phone using bluetooth connection, open the application to operate the device and choose terminal mode or input from keyboard.

Operation of keys to control the device:

Key	Action	Description of the action
n	next	play next song from the SD card
p	play/pause	play/pause current song or first from the SD card
r	remove	remove current song from SD card
q	quiet	adjust volume to lower level
l	louder	adjust volume to higher level

In our implementation, the initial value of volume level is 6 while the speaker is available to play audio in volume level between 1 and 7.

To control our device, we used the application provided by *Guiding Apps* „Arduino bluetooth controller”. Below we attach the example of using commands controlling the functioning of the device using the above-mentioned application in *terminal mode*.

The mobile application is used only as a device controller, the name of the played track is displayed on the LCD display.

1.2.6 Display

The display displays the name of the currently played song. If the name of the song is too long and it does not fit the screen, the text line is divided into two parts, displayed one after the other.

1.2. Functions

Our implementation assumes using the following functions for the device:

- reading data from SD module
- displaying information from the card on the screen
- displaying current song
- playing music through out the speaker
- controlling device via mobile phone connected using bluetooth module (play, pause, next, previous, higher volume, lower volume)
- removing songs from SD card

2. Static program analysis

2.1 Software design

Complexity of our device required us to divide our code into functions. Each function has it's own task that is to be executed.

- ***a_Global.ino***

In this part we define global and static variables that have to be accessible to each component of program. It also contains libraries definitions.

- ***b_Setup.ino***

To make code cleaner we have decided to divide typical sketch file with it's „Setup" and „Loop" functions into the separate files. In setup part we define audio settings, pins that are responsible for audio output, lcd display then it checks whatever an SD card works correctly and if the songs are present on the SD card. At the very end we also initialize a bluetooth module. Program also checks files naming and calculates amount of songs.

- ***c_loop.ino***

After proper configuration in **b_Setup.ino** loop part of code is executed. In this segment program is waiting for an input that is supposed to be send via a bluetooth. Program is responding only to specific characters. „**P**" that stands for play or pause song (depending on current state), „**N**" that stands for next (song) , „**R**" stand for remove current file, „**Q**" - reduce volume , „**L**" - increase volume. Right after analyzing input it evokes d_reactionToInput function and does whatever it was „asked" to do.

- ***d_reactionToInput.h***

In this part of code we execute specific tasks using high-level programming structure and external libraries. Task like displaying information about currently played song on the display, removing songs from an SD card, increasing and decreasing volumes, updating global information.

- ***e_assignNumericalValues.h***

This function is searching an SD card for audio files with „.WAV" extensions and then it places a number corresponding to the file into the array. Later we iterate through this array to play songs or remove them etc.

- ***f_sdCardFailure.h***

This function checks if an SD card is connected to the SD card reader and returns error in case it's not.

- ***i_sizeOfArray.h***

This function is searching an SD card for audio files with „.WAV" extensions and returns amount of songs currently present on the SD card.

2.2 Brief explanation of mechanics

In loop section of our program, Arduino keeps waiting for a user input. If it's different than the established one it will be ignored. If the requirements of „If" statement are met then program sends parameters like selected character, actual amount of songs on the SD card, Trmpcm object (responsible for audio playback), current loudness level and an lcd object responsible for displaying information.

```
char mychar;                        //character used for audio playing
if(mySerial.available()>0)mychar =  mySerial.read();  //GETTING CHAR FROM KEYBOARD

// mychar=Serial.read();

    if(mychar=='n' || mychar=='r' || mychar =='p' ||mychar =='q' || mychar =='l'){

 if(indexCounter>=HowManySongs)
    indexCounter=0; currentSong=*array_pointer;

      if(mychar=='n'){
        lcd.backlight();
      currentSong=*(array_pointer+indexCounter);
              indexCounter++;
      Serial.println(indexCounter);
            Serial.println(currentSong);
```

The following parameters are send to the function reaction to input. It's the backbone of our program. All of the commands, procedures are implemented inside this function.

```
  inputReact( mychar, HowManySongs,&currentSong,tmrpcm,&loudnessLevel,lcd);    //MAGIC HAPPENS HERE
}
```

3. „How it's made"

3.1 Playing and pausing audio

We are able to control audio playback using components of tmrmpc library. In this case if users sends „p" character to Arduino then tmrpcm.play(fileName) method initializes audio playback. However if audio is already playing than we evoke method tmrpcm.play(fileName) and pause audio playback. So that it could work exactly the same way as it does in professional hi-fi's.

```
    if(mychar == 'p' && tmrpcm.isPlaying()){
       tmrpcm.pause();    //PAUSE SONG
     }
     else if(mychar == 'p'){
       tmrpcm.play(buff);       //PLAY SONG
```

3.2 Displaying current song

To display information about currently played song we had to create an array of 32 characters. Arduino has got a problem with coping with String and they are also too memory demanding. We set cursor at the 0,0 position and use a built-in function to obtain data from a special segment of WAVE audio file and save it to the info array. Then we display characters from array in the lower and higher part of the screen.

```
    else if(mychar == 'n'){
      lcd.clear();
      tmrpcm.play(buff);    //PLAY NEXT SONG

char info[32];
char bottom[16];
  lcd.setCursor(0, 0);
  tmrpcm.listInfo(buff,info,0);
  lcd.print(info);//Serial.println(":");
  for(int i=16;i<32;i++)
  {
    bottom[i%16]=info[i];

    }
  lcd.setCursor(0,1);
  lcd.print(bottom);


      }
```

2.3 Removing song

Before removing a file from an SD card we halt audio-playback, delete it from an SD card. Rerun setup to recalculate initial parameters and start loop procedure.

```
     else if(mychar == 'r'){
       tmrpcm.disable();
     SD.remove(buff);
     setup();
     loop();
    //REMOVE SONG
      }
```

2.4 Volume adjustment

To adjust volume user has to send one of the corresponding characters. After doing so we adjust volume parameters inside global Trmcpcm object. There is also a mechanism implemented to make sure that user won't be able to decrease or increase level beyond or upon the given boundary.

```
  else if(mychar == 'q'){

    if(*loudnessLevel<=1)
    {*loudnessLevel=1;
    }
    else
    *loudnessLevel=*loudnessLevel-1;
tmrpcm.setVolume(*loudnessLevel);

    }

  else if(mychar == 'l'){
      if(*loudnessLevel>=7)
    {*loudnessLevel=7;
    }
    else
    *loudnessLevel=*loudnessLevel+1;
tmrpcm.setVolume(*loudnessLevel);

  }
}
```

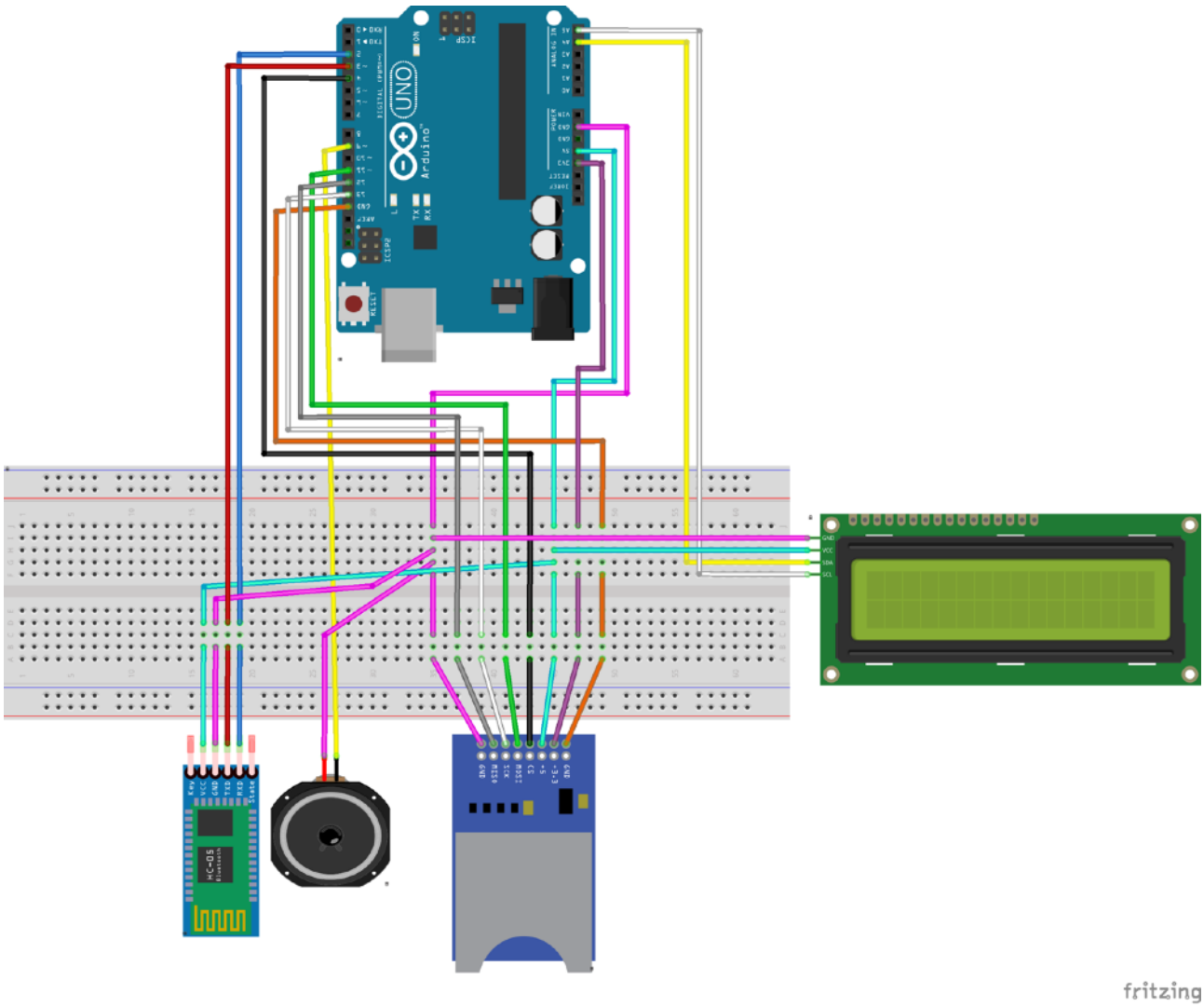
2.5 Wireless communication

A library called <SoftwareSerial.h> provided us with tools which we could use to seamlessly integrate HC-05 module with our device. When mySerial.available() method returns value greater than 0 it means we have successfully connected our Android smartphone with Arduino and it is capable of receiving character from Bluetooth terminal in our device and we can save it to a char in our device

```
char mychar;                    //character used for audio playing
if(mySerial.available()>0)mychar =  mySerial.read();  //GETTING CHAR FROM KEYBOARD
```

4. Adapting external components into the ES

4.1 GPIO scheme and explanation



Pin number	Target
13	SCK on SD card module
12	MISO on SD card module
~11	MOSI on SD card module
~9	Speaker
4	CS(Chip Select) on SD card module
~3	TXD on bluetooth module
2	RXD on bluetooth module
A4	SDA on LCD display
A5	SCL on LCD display

4.2 Speaker

We used 8 Ohm, 0.5 Watt speaker in our application.

TMRpcm library is an Arduino library for asynchronous playback of PCM/WAV files direct from SD card, it utilises standard Arduino SD library, SD card and output device - in our case Speaker.

We initiate the speaker in our code:

```
TMRpcm tmrpcm;

tmrpcm.speakerPin = 9;  //SPEAKER PIN
tmrpcm.volume(1);
tmrpcm.quality(0);
tmrpcm.setVolume(6);
initialAudioSettings=true;
}
```

In Arduino Uno the speaker is initialised for the pin 9 directly.

- tmrpcm.play("filename") - plays a file
- tmrpcm.pause() - pauses/unpauses playback
- tmrpcm.setVolume(6) - 0 to 7. Set volume level
- tmrpcm.volume(1) - 1(up) or 0(down) to control volume

4.3 Memory card

The memory card is 2 GB micro SD placed in adapter to SD.
Memory card is initialised by the SD card module at the Chip Select (CS) pin – in our case, pin 4.
In our implementation the assigning Chip Select to the proper pin is applied in the function responsible for checking if the SD card is present, by calling the *begin()*

```
void sdCardFailure(const int SD_ChipSelectPin){
 if (!SD.begin(SD_ChipSelectPin)) {  // see if the card is present and can be initialized:
   Serial.println("SD fail");
   return;   // nothing can be done
 }
```

function from the „SD" library.

4.4 Memory card reader

The SD library allows for reading from and writing to SD cards. The communication between the microcontroller and the SD card uses SPI, which takes place on digital pin. Additionally, another pin must be used to select the SD card. The File class allows for reading from and writing to individual files on the SD card.

SPI (*Serial Peripheral Interface*) is protocol of a synchronous serial data. In our implementation it takes place on digital pins 11, 12, and 13.

SPI is a synchronous communication protocol, which means that devices connected by this protocol share a clock signal. The clock signal synchronises the data bits sent from the master to the sampling of bits by the slave. The speed of data transfer is determined by the frequency of the clock signal (one bit of data per each clock cycle).
With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices:

- MISO (Master In Slave Out) - The Slave line for sending data to the master,
- MOSI (Master Out Slave In) - The Master line for sending data to the peripherals,
- SCK (Serial Clock) - The clock pulses which synchronize data transmission generated by the master

and one line specific for every device:

- SS (Slave Select) - the pin on each device that the master can use to enable and disable specific

In the setup(), we call sdCardFailure(), naming pin 4 as the CS pin

```
void sdCardFailure(const int SD_ChipSelectPin){
 if (!SD.begin(SD_ChipSelectPin)) {
   Serial.println("SD fail");
   return;   // nothing can be done
 }

 };


   root = SD.open("/");
   HowManySongs=SizeOfArray(root,0);
   root.close();
```

Creating a new file with SD.open();

4.5 Display

Displaying currently played song in our player is the task for the I2C 1602 LCD module. It is constructed of 2 lines by 16 characters. To use the device, we used its documentation and basing on it we used the recommended library “LiquidCrystal_I2C”. Due to the display needs to be implemented first, in a_GLOBAL.h we created constructor. As it needs only two data connections, constructor of the display is exactly as follows:

LiquidCrystal_I2C lcd(0x27,16,2);

Values in the constructor have been selected to set the LCD address to 0x27 and the number of rows and columns.
In the library implementation, the constructor is defined as:

LiquidCrystal_I2C(uint8_t lcd_addr, uint8_t lcd_cols, uint8_t lcd_rows, uint8_t charsize = LCD_5x8DOTS);

,where

lcd_addr	I2C slave address of the LCD display.
lcd_cols	Number of columns.
lcd_rows	Number of rows.
charsize = LCD_5x8DOTS	The size in dots that the display has, set in library as default

To work properly, the display uses ***I2C serial communication protocol***.

To transmit data, this Inter-Integrated Circuit uses two wires - SDA and SCL.

- *SDA* (Serial Data) – The line for the master and slave to send and receive data.
- *SCL* (Serial Clock) – The line that carries the clock signal.

The data is transfer bit by bit, using clock signal controlled by a master.
In this protocol, data is transform in messages. They always have “start” and “stop” bits at the beginning and at the end respectively.

- *START* condition is defined as the decrease from the high voltage level to the low voltage level of the SDA line, before the SCL switch from high to low.
- *END* condition is defined as the increase from low voltage level to high voltage level of the SDA line, before the SCL switch from high to low.

The transmitted message is broken into more frames. First 7 to 10 bits are dedicated for the *address* of the slave device. The next bit is reserved for the *write/ read bit*. Master specifies here whether he wants to write data or receive it from the device. Write bit indicates the low voltage level on the SDA line. After this and every next frame, all frames are followed by the *ACK/NACK bit.* Acknowledge bit (low voltage level) is returned to sender when the address frame or data frame was successfully received. In other case, no-acknowledge bit is returned.

The next are data frames, which are always 8-bits long and sent with the most significant bit first.
After all frames, end condition must be send by master device.

The LCD module is built in a LSI controller, which has two 8-bits registers:

- Instruction register (**IR)**
- Data register **(DR**)

Instruction register stores instruction codes and address information for displaying data RAM (DDRAM) and character generator (CGRAM).
Data register temporarily stores data to be written or read from DDRAM or CGRAM.

- *Display Data RAM (**DDRAM**)* - used to store the display data represented in 8-bit characters.
- *Character Generator RAM (**CGRAM**)* - here, the user can rewrite character. For 5x8 dots, eight character patterns can be written and for 5x10 dots, four character patterns can be written.
- *Character Generator ROM (**CGROM**)* - generates 5x8 dot or 5x10 dot character patterns from 8-bit character codes.

Interface Pin Function:

Pin No.	Symbol	Level	Description
1	VSS	0V	Ground
2	VDD	5V	Supply voltage for logic
13	SDA	H/L	Serial Data
14	SCL	H/L	Serial Clock

Instruction table:

Instruction	Instruction Code										Description
	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	
Clear display	0	0	0	0	0	0	0	0	0	1	Write „01H” to DDRAM and set the DDRAM address to „00H” from Address Counter
Set Cursor	0	0	0	0	0	1	S/C	R/L	X	X	Set cursor without changing of DDRAM data
Backlight		0	0	0	0	0	1	0	0	0	Write „08H” to DDRAM to turn on the backlight
Read data	1	1	MSB ASCII code or CG bit pattern data LSB								Read Data from CG or DD RAM:

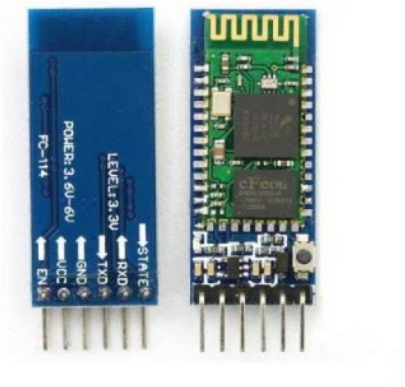
S/C=0 means move the cursor.
S/C=1 means shift display.
R/L= 0 means shift to the left.
R/L= 1 means shift to the right.

4.6 Bluetooth module

Our HC-05 Bluetooth Module can operate as either a slave device or a master device. As a slave it can only accept connections. As a master it can initiate a connection. The HC-05 module is a Bluetooth SPP (Serial Port Protocol) module, which means it communicates with the Arduino via the Serial Communication.

Specification:

- Model: HC-05
- Input Voltage: DC 5V
- Communication Method: Serial Communication
- Master and slave mode can be switched



PIN	DESCRIPTION	FUNCTION
GND	Ground	Connect to Ground
VCC	+5V	Connect to +5V
TXD	UART_TXD, Bluetooth serial signal sending PIN	Connect with the MCU's (Microcontroller and etc) RXD PIN.
RXD	UART_RXD, Bluetooth serial signal receiving PIN	Connect with the MCU's (Microcontroller and etc) TXD PIN.
KEY	Mode switch input	If it is input low level or connect to the air, the module is at paired or communication mode. If it's input high level, the module will enter to AT mode.

This Bluetooth module has a LED at the top left of the board. The onboard LED shows the current state of the module.

- rapid flash (about 5 times a second) – module is on and waiting for a connection or pairing
- single quick flash once every 2 seconds – the module has just paired with another device
- double quick flash every 2 seconds – connected to another device

The HC-05 remembers devices it has paired with and after cycling the power you do not need to re-pair.

SoftwareSerial Library supports serial communication on pins 0 and 1.

Firstly we set the data rate in bits per second (baud) for serial data transmission

```
mySerial.begin(9600);
```

It opens serial port and sets data rate to 9600 bps.

In line below we check whether data is coming from the serial port and then read the data from the serial port.

```
char mychar;
if(mySerial.available()>0)mychar =  mySerial.read();
```

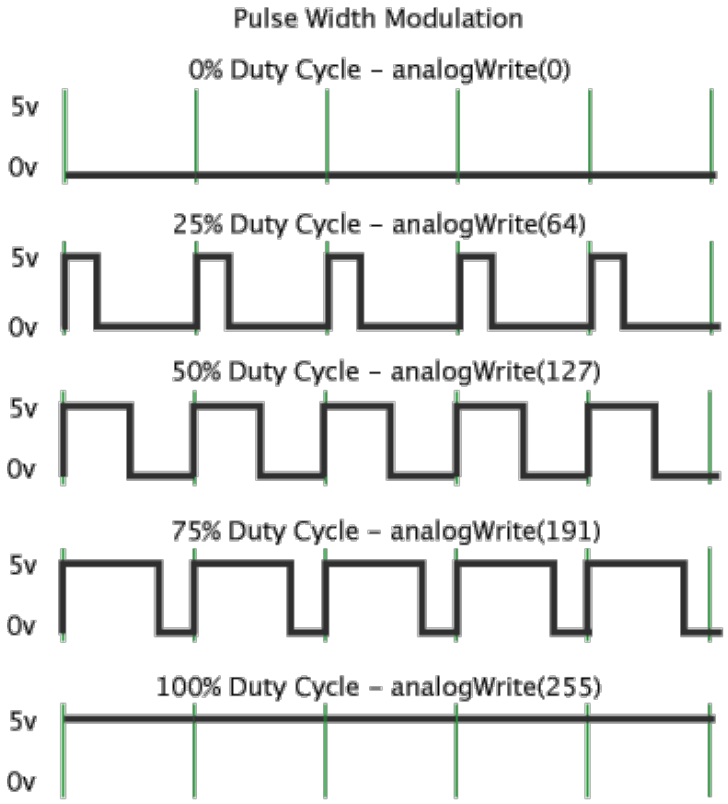
Function serial.println(). prints data to the serial port as human-readable ASCII text followed by a carriage return character and a newline character. This command takes the same forms as Serial.print().

```
Serial.println(indexCounter);
      Serial.println(currentSong);
```

4.7 Audio-player low level explanation.

In order to convert digital values into the digital form we had to use Pulse Width Modulation, or PWM. This is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. This duration or period is the inverse of the PWM frequency. A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always

on), and analogWrite(127) is a 50% duty cycle (on half the time) for example. Voltage of the final output signal is set by the PWM duty cycle percentage. Microcontroller updates the duty cycle to build the waveform. Basically the PWM function would act as a digital to analog converter (D/A or DAC).



Our device is capable of playing 8-bit PCM audio on pin 9 using pulse-width modulation (PWM). It basically uses two timers. The first timer is responsible for changing he sample value 8000 times a second. The second holds pin in state HIGH high for 0-255 ticks out of a 256-tick cycle,depending on sample value. The second timer repeats 62500 times per second (16000000 / 256), much faster than the playback rate (8000 Hz), so it almost sounds halfway decent, just really quiet on a PC speaker.
Takes over Timer 1 (16-bit) for the 8000 Hz timer. This breaks PWM
 * (analogWrite()) for Arduino pins 9 and 10. Takes Timer 2 (8-bit)
 * for the pulse width modulation, breaking PWM for pins 11 & 3.
Since we are reading WAV files directly from SD, we can save as many as the SD card will allow, with general disregard for file size.
The library uses timers and interrupts to create a signal that runs at 16000 cycles/second. The signal is controlled by the variables we read in from a file. The file is read into a small buffer, and playback is started. While interrupts control the playback, the second buffer starts filling up with data, using the spare cpu cycles between interrupts, and resumes playback once the first buffer is 'emptied'. Then the first buffer starts loading data again, while the second is 'emptied' and so on. This allows a continuous stream of data to be available for playback. Testing seems to indicate a minmum requirement for about 400 bytes of total memory for a steady stream and/or reasonable sound quality. (soundBuff = 200)

5. Failure mode and effect analysis

Our project is based on many crucial elements. They have to work properly in order to fulfill basic functionalities. Some of them are more important and others are optional and would not affect operation of our device. Items and functions which will be considered in failure analysis are presented in the table below along with the gravity of this item's failure.

Item/functionality	Severity
Microcontroller	critical
Power Source	critical
Card reader	critical
Speaker	critical
Bluetooth module	negligible
Display LCD	high

The microcontroller and power supply are an integral part of the device. The Arduino UNO can be powered from a USB cable coming from a computer. The USB connection is also how the code is loaded onto the Arduino board. Program logic and components are managed by the microcontroller, therefore without its proper operation the functionality would not be fulfilled.
Card reader, and speaker are also essential components of the device. Without Failure of the card reader would make the device unusable because it provides access to the songs from the SD card. Without a working card reader there is no real way to read the information from the SD card. As for the speaker, its failure would disable basic function of the device, which is playing the music through out the speaker.

Bluetooth module and LCD Display are non-critical component, but have great contribution to the project. Their malfunction would only partially render the device useless. Without LCD Display device would still be able to fulfil the function of displaying information about current song but only on the computer screen. As for the Bluetooth module, its failure deactivate communication via mobile phone, but then computer keyboard can be used.

Microcontroller is the heart of our applications. Failure related to it would probably be very apparent and easy to identify, as it would most likely be a chip failure.

LCD Display's failure would manifest itself as backlight problems or doesn't display or displays wrong/random characters.

Bluetooth module failure would be very easy to spot. Smartphone cannot find the HC-05 Bluetooth module or when the onboard LED is not flashing.

As for the speaker, its failure would be easy to detect. Generated sound would be significantly quieter and if the circuit was not done properly the component itself would be hot to touch.

Failure with Card reader would be very noticeable. The SD card will not be initialised (assuming that the card is fully functional).

Replacement of the components in the device would be quite easy. Because they are connected to the breadboard changing them for a substitute would take no time and effort at all. The micro-controller is placed on the board, so its replacement proves to be easy as well. In case of power supply failure replacing the cable, power brick or the whole element would could be a solution. If the problem were with the breadboard the quickest solution would be to replace it as whole.

6. References

• LiquidCrystal library documentation
https://arduinoliquidcrystal.readthedocs.io/en/latest/liquidcrystal.html#description

• LiquidCrystal_I2C.h library documentation
http://arduino.idsl.pl/index.php/biblioteki-arduino-ide/13-liquidcrystal-i2c-h

• SoftwareSerial library documentation
https://www.arduino.cc/en/Reference/SoftwareSerial

• TMRpcm library documentation
http://domoticx.com/arduino-library-tmrpcm/
https://github.com/TMRh20/TMRpcm/wiki

• SPI library
https://opencircuit.nl/ProductInfo/1000061/I2C-LCD-interface.pdf

• SD library
https://www.arduino.cc/en/Reference/SD

• HC-05 Bluetooth module documentation
http://www.electronicaestudio.com/docs/istd016A.pdf
https://www.gme.cz/data/attachments/dsh.772-148.1.pdf

• I2C protocol explanation
http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

• LCD I2C documentation
https://opencircuit.nl/ProductInfo/1000061/I2C-LCD-interface.pdf
http://www.microcontrollerboard.com/lcd.html

7. Other information

What had been promised for midterm evaluation:
• reading data from SD module
• displaying information from the card on the screen
• removing songs from SD card
What was eventually done for midterm evaluation:
• reading data from SD module
• displaying information from the card on the screen
• removing songs from SD card
• play music through out the speaker
• set of buttons to control played audio (play, pause, next, previous)
What is done apart from that:
• Communication with the device via mobile phone and Bluetooth module
• displaying information from the card on the LCD screen
• Volume adjustment