Zadanie inżynierskie

Odtwarzacz plików dźwiękowych MP3

Beata Posłuszny, Grzegorz Sroka

2019/2020

Cel

Mając projekt bazowy odtwarzający dźwięk w formacie .wave udostępniony przez prowadzących, dążyliśmy do przekształcenia go tak, aby odtwarzał format .mp3. Następnie w miarę możliwości mieliśmy go wzbogacić o dodatkowe funkcjonalności typowe dla współczesnych odtwarzaczy dźwięku.

Minimalne wymagania sprzętowe i przygotowanie środowiska

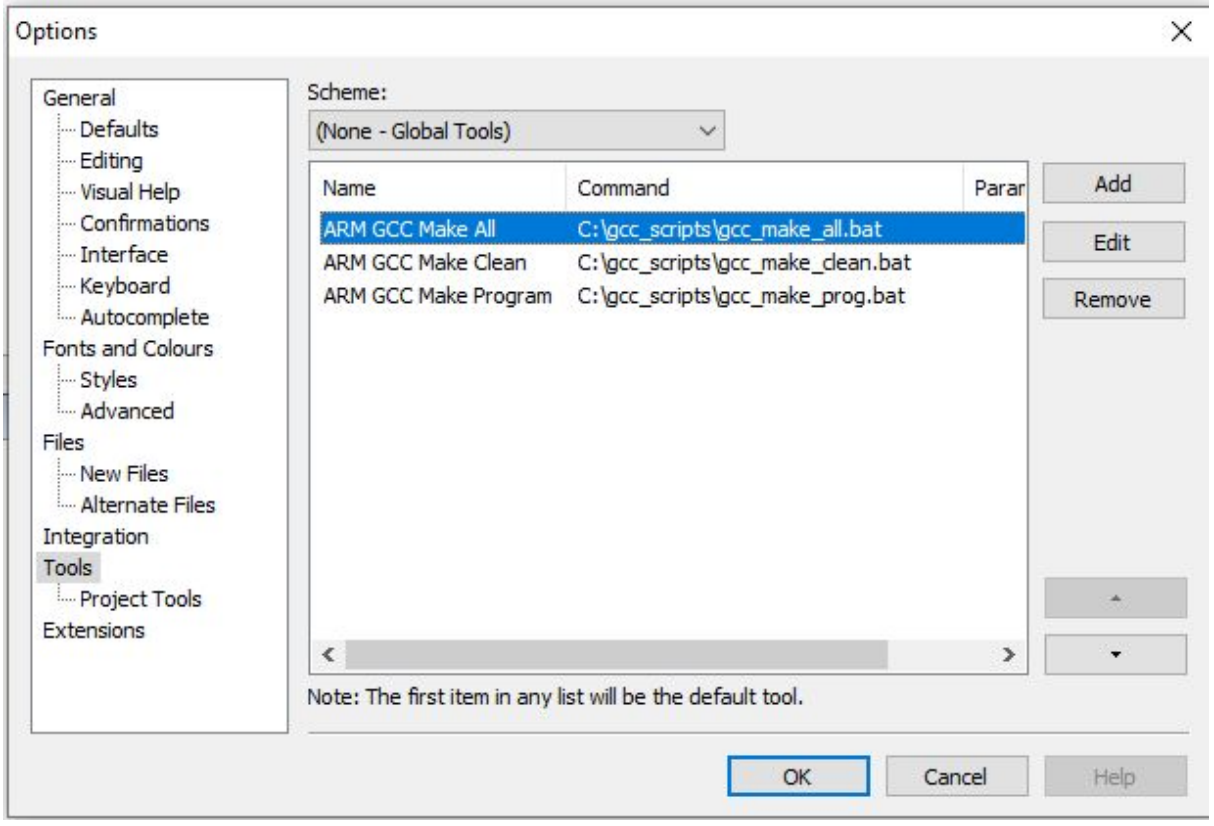
a. Sprzęt, na którym zdecydowaliśmy się wykonać ćwiczenie to płytka STM32F407. Aby połączyć komputer z płytką należy użyć przejściówki USB A do mini USB-B, a w celu połączenia płytki z pendrivem, z którego będziemy wczytywać pliki użyliśmy przejściówki USB A do micro-B. Oprócz tego w celu odsłuchania plików potrzebne są słuchawki z wtyczką mini jack 3.5mm.

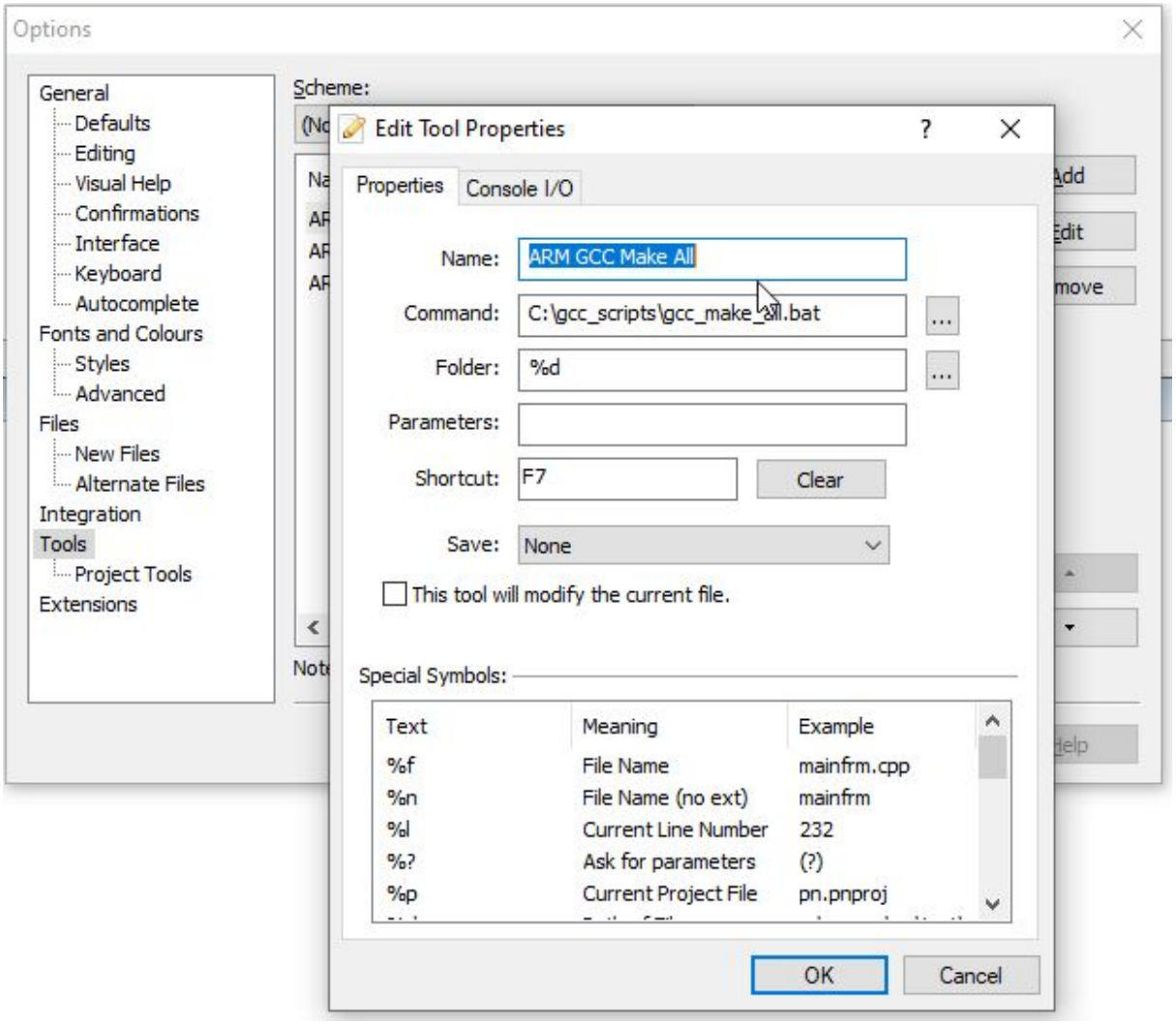
b. Do projektu dodaliśmy bibliotekę Helix[1], która udostępnia podstawowe interfejsy i algorytmy do dekodowania plików .mp3. W celu skompilowania projektu potrzebne są takie narzędzia jak:

   i. ExtraPutty, by przesyłać szeregowo znaki z terminala
   ii. kompilator arm-none-eabi-gcc w celu zbudowania projektu
   iii. OpenOCD do zaprogramowania pamięci Flash mikrokontrolera
   iv. edytor Programmer's Notepad 2
   v. Visual Studio Code - do wygodnego analizowania plików

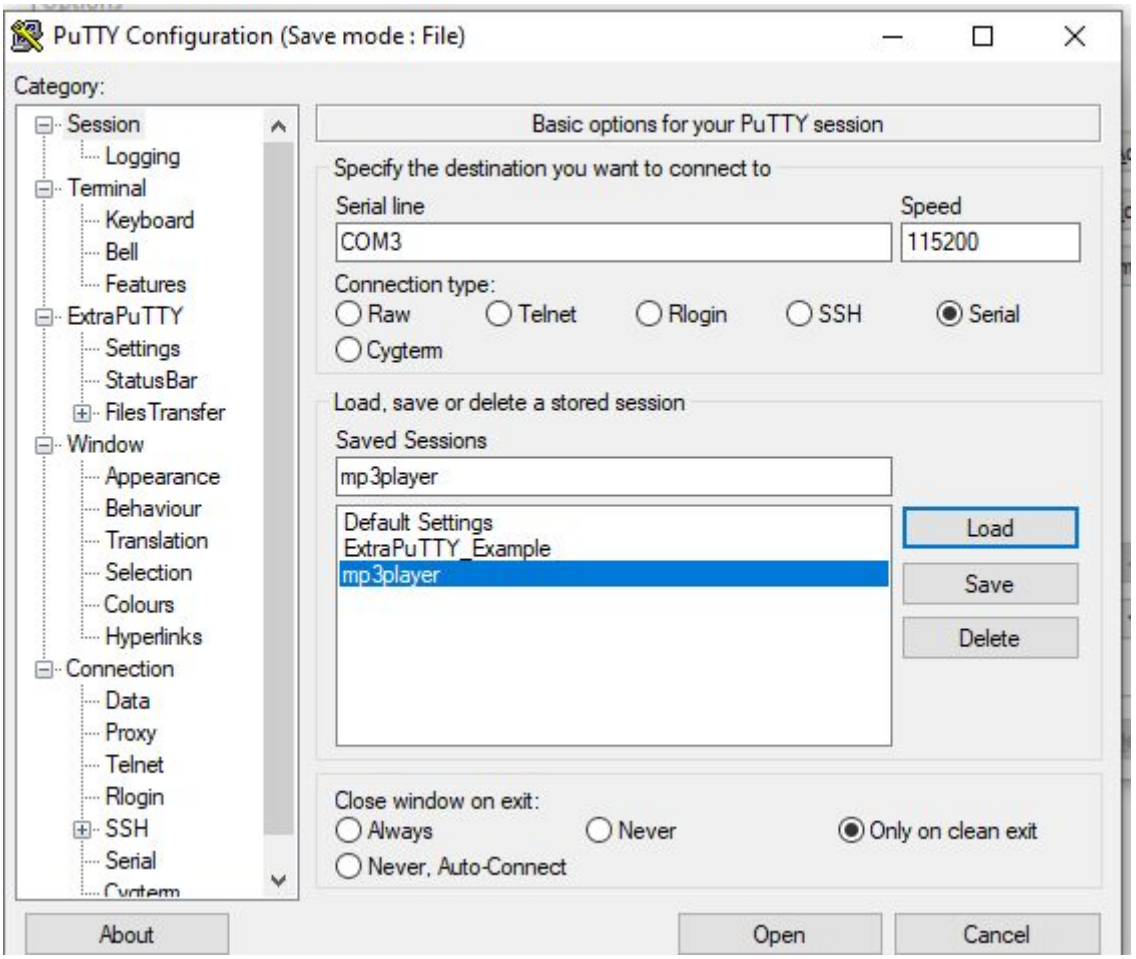
Instrukcja użytkowania

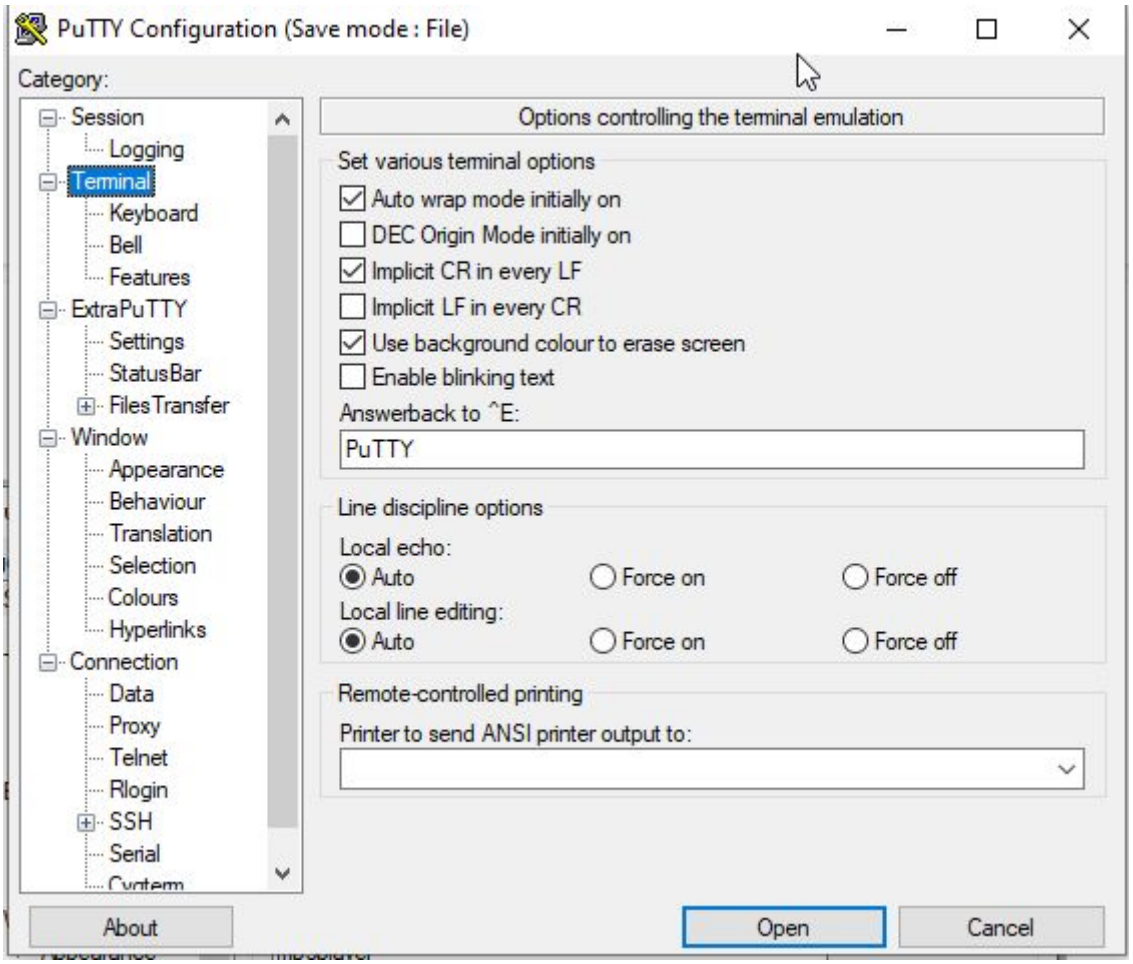
Po podłączeniu wszystkich elementów ze sobą należy przejść do kompilacji projektu[2]. Umieścić narzędzia do kompilacji wraz z projektem w pamięci komputera. W naszym przypadku foldery[5] gcc_scripts i gcc_toolchain umieściliśmy na dysku C. Ścieżki obecne w plikach będących w folderze gcc_scripts są ustawione tak, by nie trzeba było ich zmieniać, jeśli umieścimy oba foldery bezpośrednio na dysku C. W programie Programmer's Notepad 2 otworzyć plik empty.c umieszczony w głównym katalogu projektu i dodać komendy wraz ze ścieżkami do odpowiednich plików kompilatora (jak na obrazku poniżej).





W celu uruchomienia projektu należy wykonać komendę make all, następnie make program, aby zaprogramować pamięć flash mikrokontrolera. Ustawienia wprowadzane w ExtraPutty widoczne są na obrazku poniżej. Oczywiście powinniśmy sprawdzić w menadżerze urządzeń, czy przypisany został COM3 i w razie potrzeby zmienić na właściwy.


PuTTY Configuration (Save mode : File)
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
ExtraPuTTY
Settings
StatusBar
FilesTransfer
Window
Appearance
Behaviour
Translation
Selection
Colours
Hyperlinks
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Cygterm
Basic options for your PuTTY session
Specify the destination you want to connect to
Serial line
Speed
COM3
115200
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Cygterm
Load, save or delete a stored session
Saved Sessions
mp3player
Default Settings
ExtraPuTTY_Example
mp3player
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
Never, Auto-Connect
About
Open
Cancel



Skróty klawiszowe komend wprowadzane w ExtraPutty pozwalające na skorzystanie z funkcjonalności oferowanych przez odtwarzacz:

- p - pause
- r - resume
- q - previous
- w - next
- s - stop.

Po naciśnięciu klawisza stop, aby ponownie powrócić do odtwarzania, należy nacisnąć czarny przycisk na płytce (RESET BUTTON).

Sposób działania

Po inicjalizacji wszystkich podstawowych komponentów płytki, program przechodzi do zadania głównego. Tam czytana jest zawartość pendrive'a.

```
f_opendir(&directory, path)

while (1)
{
  if (f_readdir(&directory, &info) != FR_OK)
  {
```

Nazwy plików są zapamiętywane w tablicy.

```
if (strstr(info.fname, ".mp3") || strstr(info.fname, ".MP3"))
{
  paths[i] = malloc((strlen(info.fname) + 1) * sizeof(char));
  strcpy(paths[i], info.fname);
  i++;
}
```

Następnie w przypadku wybrania przez użytkownika opcji play, program przechodzi do głównej funkcjonalności. Otwierany zostaje plik .mp3.

```
res = f_open(&file, songname, FA_READ);
```

Zostaje zainicjalizowany kodek wraz z odpowiednimi parametrami, takimi jak głośność czy częstotliwość.

```
BSP_AUDIO_OUT_Init(OUTPUT_DEVICE_AUTO, 70, AUDIO_FREQUENCY_44K)
```

Kodek dostaje informację o tym, skąd pobierać gotowe dane.

```
BSP_AUDIO_OUT_Play((uint16_t *)&out_buffer[0], OUT_BUFFER_SIZE * 2);
```

Dane są nieustannie buforowane. Odczytujemy je i zapisujemy do read_buffer, który musi być co najmniej dwa razy większy niż wielkość bufora dekodera.

```
#define READ_BUFFER_SIZE 2 * MAINBUF_SIZE
```

Wartość bytes_left jest ustawiona na 0 (przypadek startowy).

```
bytes_to_read = READ_BUFFER_SIZE - bytes_left;

result = f_read(in_file, (BYTE *)read_buffer + bytes_left, bytes_to_read, &bytes_read);
```

Przesuwany jest też wskaźnik zaznaczający miejsce, z którego mają być pobierane dane do dekodowania.

```
if (bytes_read == bytes_to_read)
{
  read_pointer = read_buffer;
  offset = 0;
  bytes_left = READ_BUFFER_SIZE;
  return 0;
}
```

Znajdowany jest początek ramki mp3.

```
  offset = MP3FindSyncWord(read_pointer, bytes_left);
```

Wskaźnik przesuwany jest o tyle, ile wynosi offset. Jednocześnie aktualizujemy wartość bajtów do przeczytania.

```
read_pointer += offset;
bytes_left -= offset;
```

Następnie ramka jest dekodowana oraz zapisywana w drugim buforze - out_buffer. Jego wielkość jest dostosowana do wielkości ramki[4].
Warto tutaj wspomnieć, że bufor jest zapełniany od początku do połowy i od połowy do końca, dlatego zgłasza

zapotrzebowanie na dane dwa razy: w momencie dojścia do połowy i końca bufora, a cały proces jest nieustannie powtarzany, aż dojdziemy do końca pliku.

```
void BSP_AUDIO_OUT_HalfTransfer_CallBack(void)
{
  ON(RED);
  buf_offs = BUFFER_OFFSET_HALF;
}

void BSP_AUDIO_OUT_TransferComplete_CallBack(void)
{
  OFF(RED);
  buf_offs = BUFFER_OFFSET_FULL;
  BSP_AUDIO_OUT_ChangeBuffer((uint16_t *)&out_buffer[0], OUT_BUFFER_SIZE);
}


while (1)
{
  if (playing == SONG_IS_PLAYED_NOW)
  {

    if (buf_offs == BUFFER_OFFSET_HALF || buf_offs == BUFFER_OFFSET_FULL)
    {
      if (mp3_proccess(&file) == END_OF_FILE)
      {
```

Warto nadmienić, że gdy dekodowana jest pierwsza połowa bufora, dane są odczytywane i zapisywane do jego drugiej połowy; gdy dekodowana jest druga połowa, dane są zapisywane do pierwszej.

```
if (buf_offs == (BUFFER_OFFSET_HALF))
{
  result = MP3Decode(hMP3Decoder, &read_pointer, &bytes_left, out_buffer, 0);
  buf_offs = BUFFER_OFFSET_NONE;
}
if (buf_offs == (BUFFER_OFFSET_FULL))
{
  result = MP3Decode(hMP3Decoder, &read_pointer, &bytes_left, &out_buffer[DECODED_MP3_FRAME_SIZE], 0);
  buf_offs = BUFFER_OFFSET_NONE;
}
```

Globalnie zapamiętany jest stan, w jakim znajduje się proces odtwarzania i w przypadku interakcji ze strony użytkownika dokonywane jest porównanie nowego stanu z ubiegłym, aby poprawnie obsłużyć urządzenie - jak mówi dokumentacja w bibliotece BSP, pauza czy też start powinny być wykonywane jednorazowo (tj. mogą występować na przemian, ale nie dwa razy pod rząd).

```
case 'p':
  if (current_state == PLAY_STATE)
  {
    current_state = PAUSE_STATE;

    if (BSP_AUDIO_OUT_Pause() != AUDIO_OK)
      xprintf("Pause failed\n");

    playing = SONG_ISNT_PLAYED_NOW;
    xprintf("paused\n");
  }
  break;
```

```
case 'r':
  if (current_state == PAUSE_STATE)
  {
    current_state = PLAY_STATE;

    if (BSP_AUDIO_OUT_Resume() != AUDIO_OK)
      xprintf("BSP_AUDIO_IN_Resume failed\n");

    playing = SONG_IS_PLAYED_NOW;
    xprintf("resumed\n");
  }
  break;
 case 's':
    current_state = STOP_STATE;
    playing = SONG_ISNT_PLAYED_NOW;

    xprintf("stopped\n");
    break;
 case 'q':
    //previous
    current_state = PREVIOUS_STATE;
    playing = SONG_ISNT_PLAYED_NOW;
    break;
 case 'w':
    //next
    current_state = NEXT_STATE;
    playing = SONG_ISNT_PLAYED_NOW;
    break;
```

W przypadku wywołania stop, next lub previous należy przerwać przetwarzanie nieskończonej pętli i wywołać stop na dekoderze, aby następnie zacząć proces od nowa (tj. rozpocząć odtwarzanie od początku obecnego, poprzedniego lub następnego utworu).

```
  if (current_state == STOP_STATE || current_state == NEXT_STATE || current_state== PREVIOUS_STATE)
  {
    break;
  }
}

if (BSP_AUDIO_OUT_Stop(CODEC_PDWN_SW) != AUDIO_OK)
```

Napotkane problemy i spostrzeżenia

Pomimo posiadania projektu bazowego odtwarzającego pliki .wave, zadanie wcale nie było łatwe. Napotkaliśmy wiele trudności przy próbie zmodyfikowania kodu tak, by odtwarzał pliki mp3. Wielkość zdekodowanych już ramek okazała się być stałej długości, a nie jak zakładaliśmy na początku - zmienna. Pojawiły się też problemy z szybkością odtwarzania - była dwa razy większa, niż spodziewana. Empirycznie spostrzegliśmy, że przekazywaliśmy błędną informację o ilości danych do kodeka. Dopiero w momencie, gdy poradziliśmy sobie z błędami odtwarzania mogliśmy przejść do rozbudowy funkcjonalności: pauzy, stopu, previous, next.

Podsumowanie

Dzięki realizacji tego zadania inżynierskiego lepiej rozumiemy proces odtwarzania dźwięku oraz nabyliśmy wiedzę, dzięki której jesteśmy bardziej świadomi, w jaki sposób przebiega cały proces. Projekt można rozszerzyć jeszcze bardziej, np. dodać odtwarzanie różnych typów plików jak np. flac.

Bibliografia

[1] https://github.com/vanbwodonk/STM32F4_USB_MP3?fbclid=IwAR3rhKJAkTI1ooWEND9xT0K2JvqYOnB4Z2Xi0fXvtBqCiF2wXc-rCuVg2Ik

[2] https://github.com/sgrzegorz/STM32F4-mp3-player.git

[3] https://stm32.eu/2012/05/10/stm32butterfly-odtwarzacz-mp3/?fbclid=IwAR2uFJj5_N4XIuUdcn-lqMe6lhz735vqV1hmyp5U0vXrDrJgMk5MbSGa6Fg

[4] https://github.com/pwegrzyn/STM32F7-MP3-Player

[5] https://drive.google.com/file/d/18mRLMTOQ6li5NTuED0yK66-BWRbIh1ix/view?usp=sharing