Dokumentacja - mp3 player

Magdalena Kozub, Natalia Organek

Grudzień 2019

https://github.com/oreganko/MP3_STM32

1 Opis zadania inżynierskiego

Celem zadania była implementacja odtwarzacza plików mp3 na płytce STM32 z ekranem LCD oraz wyjściem audio. Zadanie zostało zaimplementowane w języku C przy użyciu biblioteki Helix.

Głównym problemem w tym zadaniu było odpowiednie zdekodowanie pliku mp3, który mógł składać się z ramek o różnej długości, oraz przekazywanie go na bieżąco do odtwarzacza.

2 Tworzenie playlisty

Utwory do odtwarzania umieszczone są na pendrivie. Przed rozpoczęciem odtwarzania program tworzy playlistę.

```
f_result = f_findfirst(&directory, &file_info, DISK, "*.mp3");
while(f_result == FR_OK && file_info.fname[0] && songs < SONGS) {
    char path[50];
    strcpy(path, DISK);
    strcpy(path, "/");
    strcpy(path, file_info.fname);
    strcpy(titles[songs], path);
    f_result = f_findnext(&directory, &file_info);
    songs++;
}
f_closedir(&directory);
draw_stopped_background();
xprintf("SONGS %d\n", songs);
for(int i = 0; i<songs; i++) {
    xprintf("SONG no %d: %s\n", i, titles[i]);
}
```

DISK jest zahardkodowaną ścieżką do nośnika pendrive, natomiast *titles[ ]* jest tablicą przechowującą całą playlistę. Używamy jej później do wyświetlania tytułu (który przesuwa się na ekranie tak, żeby można było przeczytać cały tytuł).

3 Działanie zadania - odtwarzanie

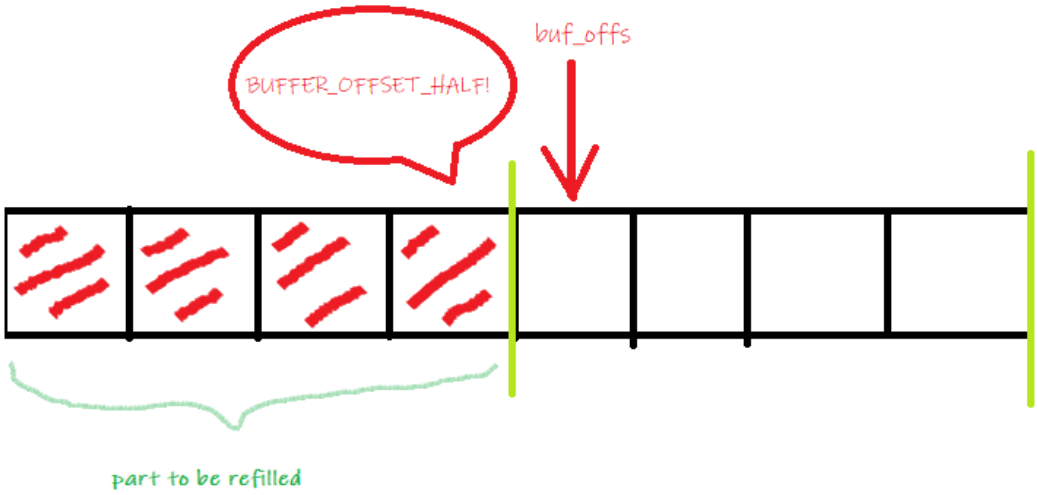
Aby poprawnie zdekodować i odtworzyć plik mp3 są potrzebne 3 bufory. Bufor, w którym trzymamy dane do zdekodowania (*input_buffer*), bufor pośredni trzymający zdekodowane dane (*spare_buffer*) i bufor danych przygotowanych już do odtwarzania (*play_buffer*). Po otwarciu pliku dekodowane jest tyle pliku mp3 aby zapełnić cały bufor dma (dwie połowy) i włączane jest odtwarzanie muzyki.

```
BSP_AUDIO_OUT_Play((uint16_t*)&play_buffer[0],AUDIO_OUT_BUFFER_SIZE);
decode(0);
decode(AUDIO_OUT_BUFFER_SIZE/2);
```

Przerwania przy odtworzeniu połowy bufora (zarówno pierwszej jak i drugiej) sygnalizują potrzebę zdekodowania kolejnego fragmentu.

```
if (buf_offs == BUFFER_OFFSET_HALF) {
    decode(0);
```

```
    }
    if (buf_offs == BUFFER_OFFSET_FULL) {
        decode(AUDIO_OUT_BUFFER_SIZE / 2);
    }
```



Samo dekodowanie można podzielić na trzy etapy:

3.1 Odczyt danych z pliku

```
if (f_read(&file, input_buffer_pointer,
AUDIO_OUT_BUFFER_SIZE - bytes_in_input_buffer, (void *) &bytes_read) != FR_OK) {
    BSP_AUDIO_OUT_Stop(CODEC_PDWN_SW);
    xprintf("file reading error\n");
                }
bytes_in_input_buffer += bytes_read;
input_buffer_pointer = input_buffer;
        }
```

input_buffer_pointer jest w tym momencie wskaźnikiem na koniec danych w *input_buffer* - gdzie możemy wklejać następne dane. Odczytywane jest tyle bajtów, żeby zapełnić cały *input_buffer*.

3.2 Dekodowanie danych

Do dekodowania danych służą funkcje z biblioteki Helix.

```
while(spare_buffer_offset < AUDIO_OUT_BUFFER_SIZE / 4){
    int offset = MP3FindSyncWord(
        (unsigned char *) input_buffer_pointer, bytes_in_input_buffer);
        if(offset == -1){
```

```
                xprintf("Offset is -1 :(");
        }
        //We move pointer to find first frame
        bytes_in_input_buffer -= (offset);
        input_buffer_pointer += (offset);
        //We decode first frame
        //- use input buffer for this. Decoded data goes to spare_buffer
        int is_error = MP3Decode(decoder, (unsigned char**) &input_buffer_pointer,
        (int*) &bytes_in_input_buffer, spare_buffer_pointer,0);
        if(is_error){
                xprintf("Error while decoding %d.\n", is_error);
                break;
        }
        //Need to know how many bits were processed

        MP3GetLastFrameInfo(decoder, &mp3_frame_info);
        spare_buffer_offset += mp3_frame_info.outputSamps;
        spare_buffer_pointer = spare_buffer + spare_buffer_offset;

}
```

spare_buffer jest w formacie wymaganym przez zdekodowane dane biblioteki Helix - short, natomiast *play_buffer* w formacie wymaganym przez bibliotekę obsługi dma - char. W związku z tym *spare_buffer* jest dwa razy mniejszy niż *play_buffer*. Aby wypełnić połowę bufora *play_buffer*, należy wypełnić *spare_buffer* również do połowy (czyli do 1/4 bufora *play_buffer*).

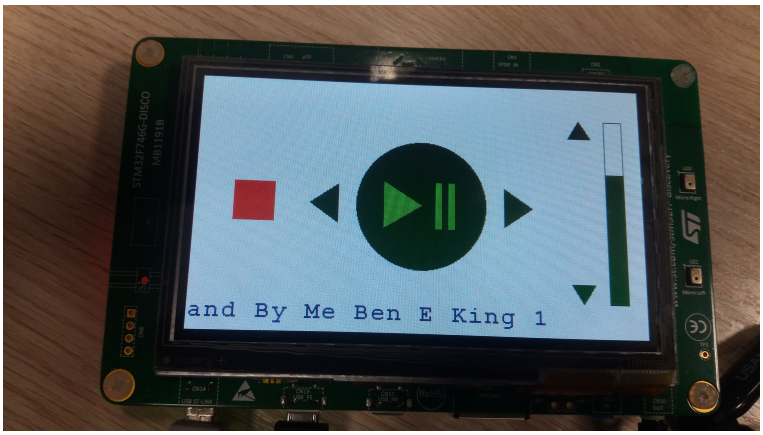
3.3 Porządkowanie buforów

```
//move data in input buffer to its beginning
memcpy(input_buffer, input_buffer_pointer, bytes_in_input_buffer);
//input_buffer_pointer has to show place where new data can be put
input_buffer_pointer = input_buffer + bytes_in_input_buffer;

//We want to play music so we fullfill half of play_buffer
and argument play_offset shows which one half
memcpy(play_buffer + play_offset, spare_buffer, AUDIO_OUT_BUFFER_SIZE/2);
//We have copied audio_out_buffer_size/4 of spare_buffer to play_buffer
//so we delete it
memcpy(spare_buffer, &spare_buffer[AUDIO_OUT_BUFFER_SIZE/4],
(spare_buffer_offset - AUDIO_OUT_BUFFER_SIZE/4) * 2);
spare_buffer_offset -= AUDIO_OUT_BUFFER_SIZE/4;
spare_buffer_pointer = spare_buffer + spare_buffer_offset;
buf_offs = BUFFER_OFFSET_NONE;
```

4 Interfejs użytkownika

W projekcie do implementacji interfejsu użytkownika został wykorzystany ekran LCD z panelem dotykowym. Ekran LCD wyświetlał przyciski przedstawione za pomocą figur geometrycznych - prostokątów, trójkątów oraz kół.



Aby kod był bardziej czytelny dodano funkcje pomocnicze rysujące te figury. Kiedy użytkownik naciskał przycisk funkcja odpowiedzialna za obsługę dotyku wywoływała odpowiednie funkcje kontrolujące odtwarzacz.

```
void detect_touch(TS_StateTypeDef touch){
    unsigned x = touch.touchX[0];
    unsigned y = touch.touchY[0];
    if(inRange(x0, x1, x) && inRange(y3, y4, y)){
        //stop_song();
        xprintf("song stopped\n");
        draw_stopped_background();
        end_song();
        current_song = 0;
        player_state1 = 2;
    }
    else if(inRange(x2, x3, x) && inRange(y3, y4, y)){
        //play_prev_song();
        current_song = current_song > 0 ? current_song - 1 : songs - 1;
        end_song();
        play_new();
        xprintf("prev song\n");
    }
    else if(inRange(x4, x5, x) && inRange(y1, y5, y)){
        if(player_state1 == 2){
            play_new();
            draw_playing_background();
            xprintf("song started\n");
        }
```

```
        else if(player_state1){ //pause_song();
            BSP_AUDIO_OUT_Pause();
            draw_stopped_background();
            player_state1 = 0;
            xprintf("song paused\n");
        }
        else { //continue_song();
            BSP_AUDIO_OUT_Resume();
            draw_playing_background();
            player_state1 = 1;
            xprintf("song continued\n"); }
        }
    else if(inRange(x6, x7, x) && inRange(y3, y4, y)){
        //play_next_song();
        current_song = (current_song + 1) % songs;
        end_song();
        play_new();
        xprintf("next song\n");
    }
    else if(inRange(x8, x9, x)){
        if(inRange(y0, y2, y)){
            volume_up();
            xprintf("volume up\n");
        }
        else if(inRange(y6, y7, y)){
            volume_down();
            xprintf("volume down\n");
        }
    }
}
```

5 Funkcjonalności

Odtwarzacz pozwala użytkownikowi na następujące czynności:

- Przełączanie pliku na poprzedni/kolejny
- Zwiększanie/zmniejszanie głośności

```
int volume_up(){
    if (BSP_AUDIO_OUT_SetVolume(volume < 91 ? volume += 10 : volume) != AUDIO_OK){
        xprintf("ERROR: Failed to set audio volume\n");
        return -1;
    }
    xprintf("v: %d", volume);
    draw_current_volume(volume);
    return 0;
}
```

```
int volume_down(){
    if (BSP_AUDIO_OUT_SetVolume(volume > 9 ? volume -= 10 : volume) != AUDIO_OK){
        xprintf("ERROR: Failed to set audio volume\n");
        return -1;
    }
    xprintf("v: %d", volume);
    draw_current_volume(volume);
    return 0;
}
```

- Zatrzymanie (pause oraz stop) oraz wznawianie utworu

```
void end_song() {
    BSP_AUDIO_OUT_Stop(CODEC_PDWN_SW);
    f_close(&file);
    printf("EndOfSong");
    player_state1 = 0;
    spare_buffer_offset = 0;
    bytes_in_input_buffer = 0;
    buf_offs = BUFFER_OFFSET_NONE;
}
```

- Wyświetlanie tytułu

```
void print_title() {
    if(strlen(titles[current_song])>=22){
        uint8_t title[22];
        memcpy(title, titles[current_song] + title_offset, 22);
        BSP_LCD_SelectLayer(1);
        BSP_LCD_SetTextColor(LCD_COLOR_DARKBLUE);
        BSP_LCD_DisplayStringAt(0, LCD_Y_SIZE - 30, title, LEFT_MODE);
        title_offset = (title_offset + 1)%(strlen(titles[current_song])-21);
    }
    else {
        uint8_t title[22] = "                      ";
        memcpy(title, titles[current_song], strlen(titles[current_song]));
        BSP_LCD_SelectLayer(1);
        BSP_LCD_SetTextColor(LCD_COLOR_DARKBLUE);
        BSP_LCD_DisplayStringAt(0, LCD_Y_SIZE - 30, title, LEFT_MODE);
    }
}
```

6 Źródła

- Projekt bazowy pobrany został ze strony upel.agh.edu.pl z sekcji Systemy Wbudowane
- Pomysł na wykonanie projektu oraz biblioteka Helix ze strony *https://stm32.eu/2012/05/10/stm32butterfly-odtwarzacz-mp3/*
- Użycie transferowego bufora - projekt Julii Dutkiewicz oraz Justyny Zawalskiej