Musical notation player — an embedded system project

Zawartość

1 The aim of the project

The project "Musical Notation Player" aimed to create a hardware solution for playing musical pieces stored as a predefined sheet music format in Flash memory. Each piece is inputted according to the methodology developed by the project authors.

One of the goals of the project was to learn the specifics of embedded systems development and programming in C language, using elements characteristic of low-level programming, such as operations on registers.

2 General description of the project

2.1 External hardware components

In the project, we used components such as:

- Open1768 development board with LPC1768 microcontroller, with an adapter that allows communication with a computer via UART (using UART0 input).
- A simple speaker connected via a DAC, powered from the +5V rail on the board (using an additional wire), used to play the recorded melody.
- Flash memory AT45DB041D connected via SPI (SPI/SSP0 input) for storing piece musical notation.
- ILI9325 LCD display connected through the display input on the board, for displaying the user interface.

UART communication is necessary to use the device in **service mode**, in case the user wants to save a new song. Otherwise, if there are songs stored in Flash memory already, the connection is not necessary. UART's baud rate equals to 115200.

The photo on the next page shows the complete device with all necessary hardware components connected.

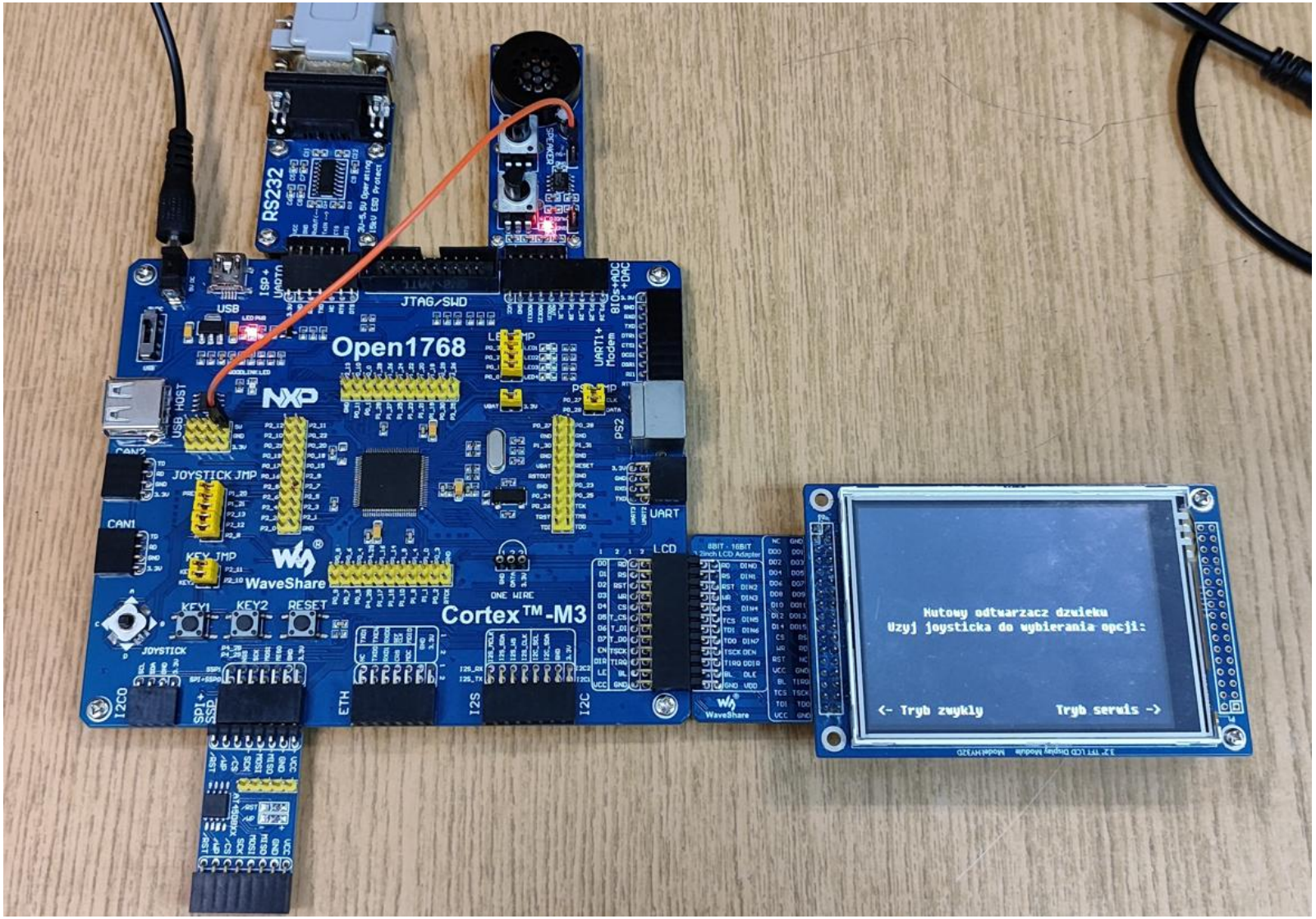

Fig. 1 Board with all hardware components connected and program running

2.2 Microcontroller peripherals

The following microcontroller peripherals (with their configuration) were used for the project:

- SysTick timer used to generate interrupts with delay() procedure, configured to milliseconds (SystemCoreClock/1e6), which interrupts are handled with the SysTick_Handler() procedure,
- Joystick (on the board), configured with the default procedure Joystick_Initialize() from the Board_Joystick.h library available in the Keil environment,
- LCD input configured with the default procedure lcdConfiguration() from the Open1768_LCD.h library, provided by the Course Instructor,
- An ILI9325 LCD display configured with the default procedure init_ILI9325()
- from the LCD_ILI9325.h library,
- UART0 for communication with the computer, activated by setting pins P0.2 and P0.3 to functions corresponding to the UART, and then configured by activating the divider flip-flops in the LCR register, writing the value of divider 27 (for a baud rate of 115,200 baud) in the DLL/DLM register, and deactivating the divider flip-flops,
- DAC, configured by setting pin P0.26 to the appropriate function, according to the documentation provided,
- SPI, for which the ARM_DRIVER_SPI* SPIdrv pointer variable set to the Driver_SPI2 was declared, on which initialization, power supply, and configuration is performed: setting master mode, setting the polarity and the phase of the clock (data transfer to the MOSI I/O on the falling edge of the clock, sampling on the rising edge), setting the most-significant-bit first (MSB), setting the eight-bit word and the transmission rate of 1 Mbps.

3 End-user application description

The application allows to play and save new songs to Flash memory. A simplified flowchart of the application logic is shown below.

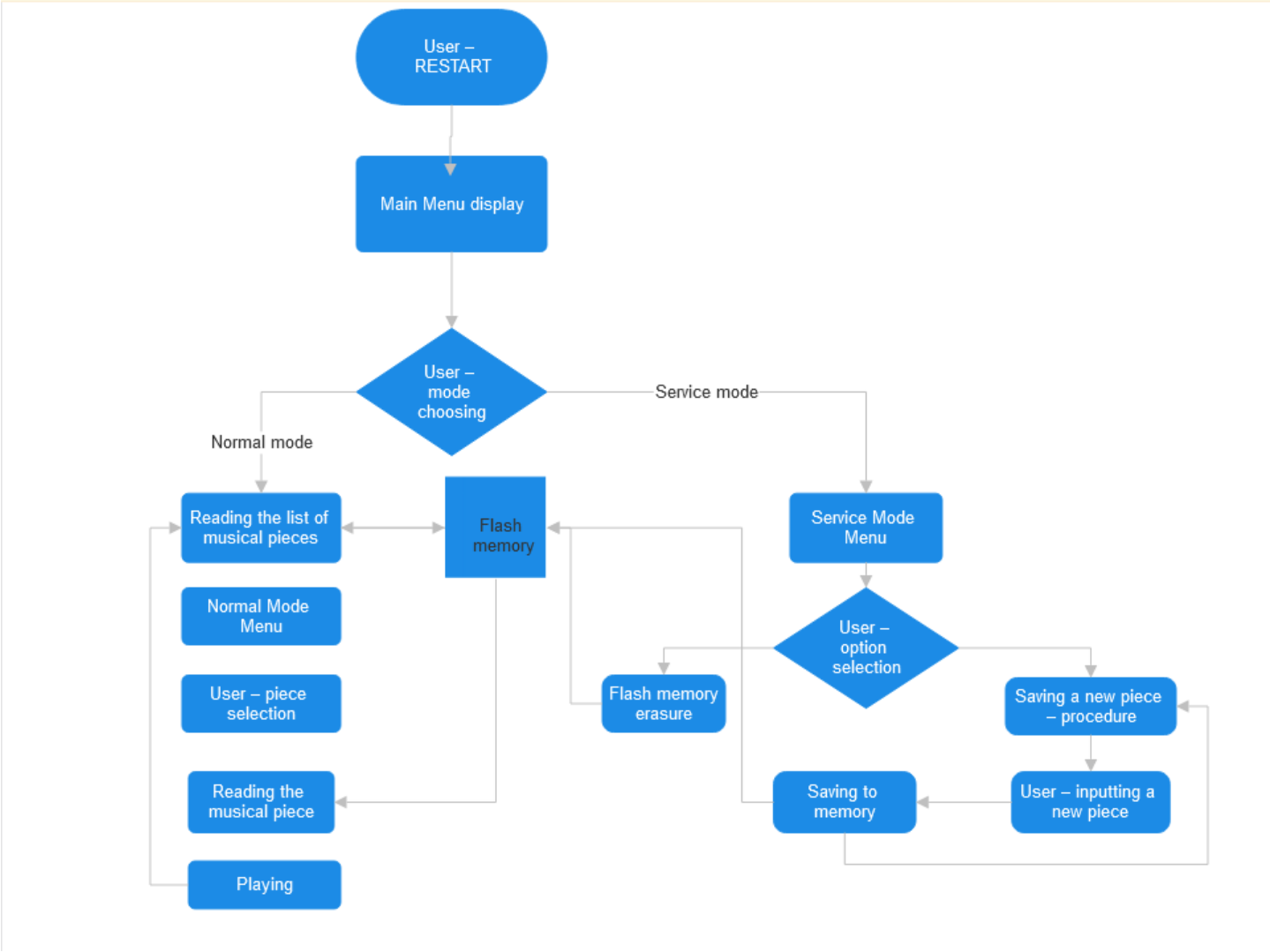

Fig. 2: Application logic flowchart

3.2 General remarks

The end-user interface is displayed on the LCD and it is the main communication channel between the application and the user in the normal mode. The user communicates with the application using a joystick located on the development board. The functions activated by moving the joystick in the appropriate direction are shown on the device display.

Communication in service mode is handled by the LCD (displaying basic diagnostic messages), as well as by a terminal listening to a serial port transmitting via UART (e.g., Tera Term or PuTTY). After choosing to save a new musical piece, the communication is performed by the user's keyboard.

3.3 The Main Menu

After restarting, the application asks the user to choose a mode: either a normal mode or a service mode. The selection is made using the joystick.

Fig. 3: The Main Menu. (Middle: "Musical notation player / Use the joystick to choose an option"; Left: "Normal mode"; Right: "Service mode")

3.4 Normal mode

Upon entering regular mode, the user is presented with a screen that allows selecting a song from the list. Scroll through the song list by moving the joystick left or right, play the selected song by moving the joystick up. The track list is a circular doubly linked (excluding the first item) list with a maximum of 20 items.

Fig. 4: The beginning of the list. (Middle: "Piece number 0 / 'A kitty climbed the fence' (Polish Children's Song)/ ^Play"; Right: "Next piece")

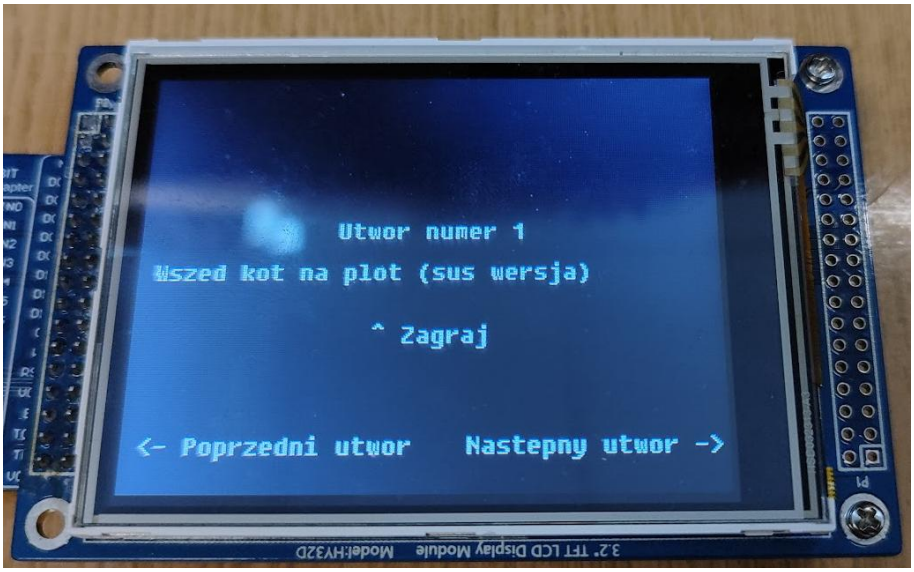

Fig. 5: Following pieces on the list. (Middle: "Piece number 1 / 'A cat came up the fence' (suspicious version)/ ^Play"; Left: "Previous piece"; Right: "Next piece")

Fig. 6: Playing the musical piece (Middle: "Piece number 0 / 'A kitty climbed the fence' / Playing...")

If the program determines that the Flash memory does not contain song data, a corresponding message will be displayed, at the same time the user is allowed to enter the service mode by moving the joystick to the right.

Fig. 7: No pieces found message. ("No pieces in memory. / Go to service mode")

3.5 Service mode

The service mode allows modifying the contents of the Flash memory, i.e. clearing it or saving a new song. The selection of the appropriate option is made by moving the joystick in the appropriate direction at the Main Menu.

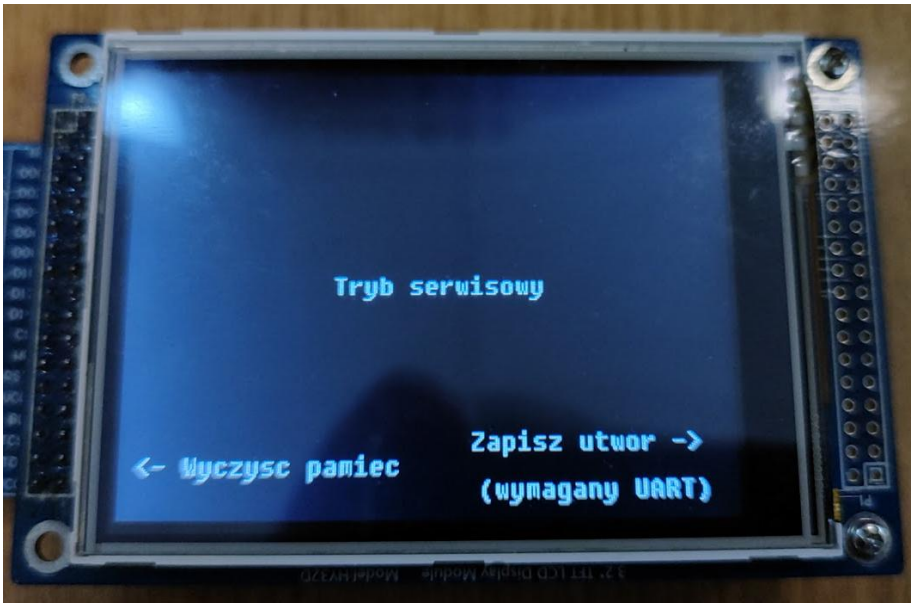

Fig. 8: Service Mode Menu (Middle: "Service mode"; Left: "Clear memory"; Right: "Save a piece (UART required)")

When you start saving a new piece, a corresponding message appears on the screen, indicating the ID of the song being recorded.

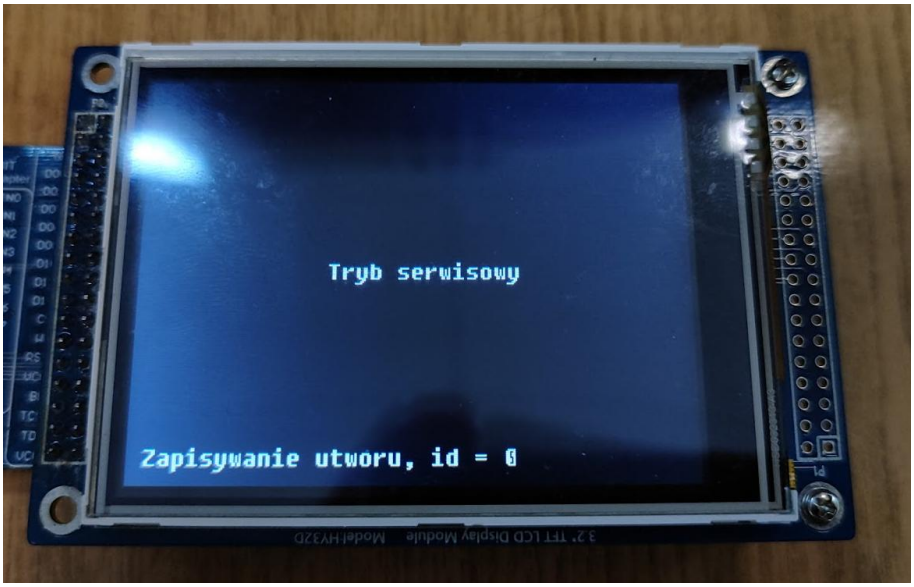

Fig. 9: Saving a new piece (Left: "Saving a piece, id = 0")

Additional information appears on the serial port console and prompts for more information about the song: its name, the playback tempo (in beats per minute) and the following notes.

```
Sprawdzanie pamieci flash...
Flash OK
Zapisywanie utworu, id = 0Zapisywanie utworu...
Nazwa utworu:...
Gama C-dur
Podaj tempo:
120
Wpisuj nuty...
C044 D044 E044 F044 G044 A044 H044 C052 H044 A044 G044 F044 E044 D044 C044
Nuty zapisane.
```

Fig. 10: Interface of service mode in the terminal ("Checking flash memory... / Flash OK / Saving the piece, id=0Saving the piece... / Name of the piece: ... / 'C major scale' / Tempo: / 120 / Enter notes... / 'C044 D044 E044 F044...' / Notes saved.")

The notes are entered according to the logic described later in the project description, in the section on the algorithms used (Chapter 4).

Attempting to save a piece, if Flash memory is full or not detected at all, will result in an error.

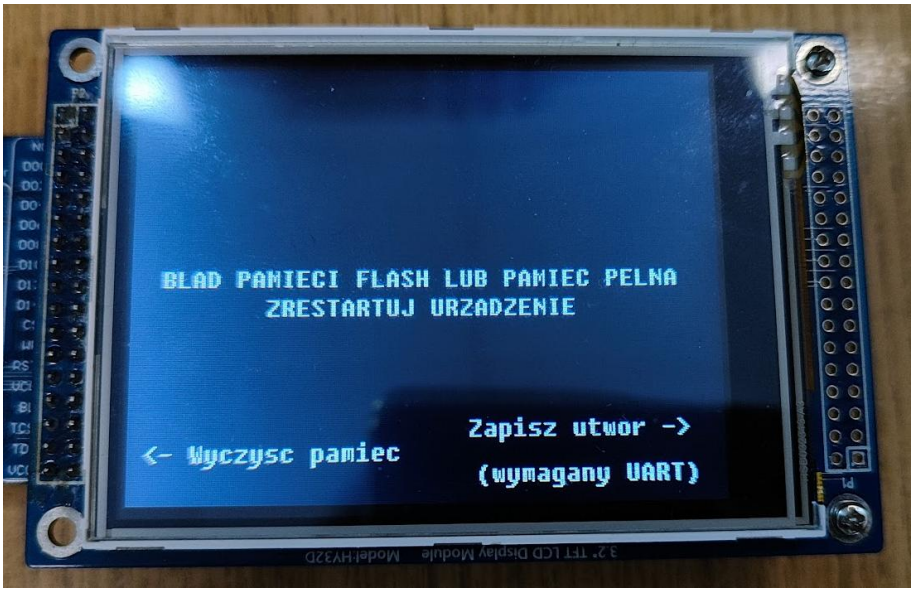

Fig. 11: Memory error (Middle: "FLASH MEMORY ERROR OR THE MEMORY IS FULL / RESTART THE DEVICE")

A maximum of 20 tracks of 4508 notes each are allowed to be recorded. These values are due to limitations on the size and number of pages available in Flash memory. You can change these values by making appropriate adjustments to the code (for example: more tracks but with fewer notes, or less tracks with more notes).

Successful piece saving is communicated to the user through the screen:

Fig. 12: Successful saving message (Middle: "Service mode"; Bottom: "RESET the device to return to normal mode/ Joystick Right to save another piece -> / The piece has been saved.")

Additionally, the user may erase the data in Flash memory:

Fig. 13: Memory erasure message ("Cleaning OK. Reset the device / Cleaning flash memory...")

4 Application algorithm

4.1 Initialization and main loop

In the main body of the program (main.c), after proper activation and configuration of the peripherals, the existence of Flash memory is checked using the function ***flash_check_present(),*** which performs the check not only of the physical presence of the hardware, but also the very fact that the peripheral responds to signals. This is done by sending the appropriate data over the SPI interface to the Flash memory register.

The program then falls into an infinite loop, which implements the main part of the user interface user interface using the LCD screen. Drawing on the said screen is done primarily primarily using two functions:

- ***void rysujprostokat (uint16_t x, uint16_t y, uint16_t xx, uint16_t yy, uint16_t color)*** and
- ***void rysuj_tekst(const char* tekst, int x, int y).***

The ***rysujprostokat (...)*** function is used to draw a rectangle with dimensions (xx, yy) on the screen, starting from the initial x and y coordinates. The function iterates through the area of the rectangle and, using ***the lcdWriteReg()*** function, inserts the appropriate data into the registers of the of the display, effectively drawing a rectangle with the given parameters.

The ***rysuj_tekst(...)*** function, in turn, is used to draw a string of characters. By iterating over the string, it calls the ***rysuj() (char literka, uint16_t x, uint16_t y)*** function for each character

up to the end-of-chain character (\0), effectively writing out all the text at the intervals specified by the function's intervals (the y parameter increases by 8 every character).

The ***rysuj*** (...) function draws individual letters on the screen, based on the data contained in the in the asciiLib.h library, it takes the character and coordinates of the starting point (x, y) as arguments.

Then the state of the joystick is retrieved continuously - in this way we are able to go to the menu we want. When the joystick position "to the right" (JOYSTICK_RIGHT) is detected, we go to the service menu, detection of the left position (JOYSTICK_LEFT) ensures the transition to the normal menu.

4.2 Service menu

Before the description of the service menu, it is necessary to know the structure of each piece.

The application provides exactly 100 pages of flash memory for each piece of information about some selected song. According to our memory configuration, we have access to a maximum of 2048 pages. In view of this, we decided to store a maximum of 20 songs (a maximum of 2000 occupied pages).

4.2.1 Structure of the piece

Structure of the piece in Flash memory:

- Page id-100 + 1 - musical piece title, maximum 63 characters - typing more than that results in undefined behavior
- Page id-100 + 2 - tempo of the piece
- Page id-100 + 3 - duration of the piece (number of notes)
- Page from id-100 + 4 to id-100 + 100 - individual notes written as a string of characters, where 48 notes are provided for one page.

In order to simplify the handling of notes, a special mini library notes.h was created, which consists of the Note structure (consisting of two fields: frequency of float type and length of type int) and the Note procedure getNote(const char[4]), which wraps the received strings into a structure.

4.2.2 Algorithm for translating a string into notes

Each note consists of four characters, and notes are separated by a white character.

- The first character of this sequence defines the sound being played according to standard international musical nomenclature: C, D, E, F, G, A, and B (or H).
- The second character of the string defines whether the sound played is "natural," i.e. without chromatic signs, or with them:
    - 0 - the sound is played naturally,
    - b - the sound is lowered by half a tone,
    - # - the sound is raised by half a tone.
- The third character of the string tells the octave in which the note is played. It is possible to enter any number from 0 to 9, the assumption being, that according to standard musical nomenclature, the octave marked with the number 4 is the middle (one-lined) octave and the other octaves are tuned to it. It is important to note that the frequency of a note in the higher octave is twice as high as that in the in the lower octave – this made it possible to write into the library code only 15 predefined frequency values, and the rest is calculated dynamically.
- The last character is the length of the song, expressed as the denominator of a fraction that is the proportion of the length of a note to the length of a bar of 4/4 meter:
    - 1 - whole note
    - 2 - half note
    - 4 - quarter note
    - 8 - eighth note
    - 6 - sixteenth note (1 truncated due to one allowed character).

Such a string is translated into a Note structure.

For example, for Eb58 such steps are performed:

- the first switch statement reads the character representing the note E and falls into the next switch,
- the second switch reads the character representing the flat ('b'), that is, lowering by half a tone, from the above data, the frequency is extracted: 311.13 Hz,
- frequency is multiplied by 2 to the power of the octave minus 4, in this particular case 25-4 = 2, giving a final frequency of 622.26 Hz, this value is written to the Note structure in the freq field,

- the last switch reads the character representing the length of the note and writes it into the structure into the duration position

The structure prepared in this way is then used to play the following notes of the song. The function responsible for this is **void playnote1(Note note, int tempo),** which is described later in the documentation.

4.2.3 Memory erasing

The service menu can be used to clear the memory. The following is used function from the imported flash_erase_all() library, which clears all pages of the memory flash. At the same time, it writes to each possible page with id*100 + 3 the value of the numeric value -1, as a character, which will later be used to determine the free space for the song.

4.2.4 Writing to memory

Writing data to a specific page is implemented using the library function **flash_write_page(uint16_t page_no, uint8_t *buffer, size_t len, uint8_t erase_first),** where:

- page_no - page number
- *buffer - pointer to an array of characters
- len - length of the data string
- erase_first - whether we want to clear the page first before saving the data

To save the song, we use the void function **check_to_save_utwor()**, which retrieves the information from flash memory using the library function

void flash_read_page(uint16_t page_no, uint8_t *buffer, size_t len), which gets the *len* length information from the page number *page_no* into the buffer pointed to by *buffer*.

The function **check_to_save_utwork()** retrieves such an *id* for which the length of the piece is outside the allowed range [1, 4656] (e.g. -1). The new song is written only after a free id is found, or not at all if a no such id is found. The message in Fig. 11 is then displayed.

The **int save_utwor (int id)** procedure is used for saving. It saves the piece to Flash memory, retrieving the musical piece name, tempo and note data according to the formula above from the UART port. Each of these input data is separated by the Enter button. It uses the **flash_write_page()** function to write this information to the appropriate Flash memory pages.

To switch to the normal menu from this position, it is necessary to restart the board.

4.3 Normal menu

4.3.1 Defined procedures and variables

In the global area we included two system counters msTicks:

- volatile int msTicks = 0;
- volatile int msTicks2 = 0;

We have defined an interrupt originating from the system clock **(void SysTick_Handler(void))** as one that, increases the msTicks counters by 1 each millisecond.

We defined the **void delay(int t)** function, which introduces a delay in the operation of the program, expressed in milliseconds.

Such functions were also defined:

- **void f(int freq)** – generates a delay based on the given frequency freq. It uses the delay function, converting the frequency to a period in milliseconds.
- **void print(const char* a)** – prints a string of characters to the UART, often used for debugging, but also when writing new songs. The function iterates over each character of the string, sending it to the UART port using registers LPC_UART0.
- **void rysuj_tytul(int k)** - reads the title of a song with id ***k*** from Flash memory and prints it on the LCD.
- **int check_to_play_utwor()** - returns the first id on which a piece exists.
- **int is_utwor_present(int id)** - checks (based on the stored length of the song), whether there is a song stored on the given id. If there is, it returns 1, if there is not, it returns 0.

The implementation of the normal menu interface and menu navigation was largely based on the 3 functions above.

4.3.2 Navigating the normal mode menu.

In the normal menu, it is possible to move between available pieces using the joystick (left or right). The menu is drawn using the procedures rysujprostokat(...) and rysuj_tekst(...) with the additional procedure described above rysuj_tytul(...).

4.3.3 Playing the piece

To play a musical piece, place the joystick in the upper position (JOYSTICK_UP). Then, after confirming the existence of the flash memory with the **flash_check_present()** procedure, the **void play_utwor(int id)** procedure is executed on the current id.

This is one of the key routines of the program, it writes out the title, tempo and length of the of the song on the UART (optional functionality), using **flash_read_page(...).** The piece is played by writing a string of 5 characters to the Note structure and using the procedure:

- **void playnote1(Note note, int tempo)** extracts from the Note structure the information about the given note. A note of a certain frequency is obtained on the speaker by changing the intensity of the signal (the DACR register of the DAC converter) every appropriate period. We obtain it by using the function **f (int freq).** We obtain the length of the note by using the **msTicks2** variable accordingly

5 Sources

To create the project, the following were used:

- LCD libraries Open1768_LCD.h/c, LCD_ILI9325.h/c and asciiLib.h/c, provided by the Course Instructor,
- Libraries to support the SPI periphery and Flash memory: spi_master.h/c and at45db041x.h/c made available by user eziya on GitHub, modified accordingly to suit LPC1768,
- Libraries containing APIs for handling microcontroller peripherals on the board Open1768 evaluation board available in the Keil environment.

Other code snippets, algorithms and data structures are the authors' own work of the project or come from the public domain (CC0 license).