Brno University of Technology

Faculty of Information Technology

Microprocessors and Embedded Systems

Programming of Embedded Application

Encoder of Morse Code

December 19, 2018

Stupinský Šimon, (xstupi00)

1 Introduction

The main purpose of this project is implementing the *embedded application* in *C Language* for platform *Kinetis K60* on the development board *FITkit3*[1]. This application will be receiving the entered characters from the keyboard and subsequently encoding this characters to **Morse Code**. Characters input will be performed interactively via the keyboard attached to the *FITkit3* development board. Since the keyboard contains only 12 buttons, the way of entering the chars will be implemented such as on older mobile phones at writing the SMS.

The output of this application will be realized with using the *piezo* loudspeaker, which is obviously presented on the development board *FITkit3*. For better differentiation of the encoded characters, respectively its part according to **Morse Code**, are sound effects enlarged by blinking of four LEDs on this board. The application allows entering the characters even if the previous character is still being played by the buzzer. The **all** characters, for which exists the **Morse Code**, is supported. The button $\star$ serves to the *reduction* of the frequency the encoding characters and button # to its *accelerated*.

The application was developed in **MCUXpresso IDE v10.2.1**.

2 Hardware Keyboard Plugging

The only hardware component which was necessary to connect to development board *FITkit3* is external **Keyboard**. As you can see in Figure 1, keyboard contains **12** buttons, which are located to **three** columns and **four** rows. In the summary, the keyboard contains the **seven** different *pins*, which represents the concretely relevant *row*, respectively *column*.

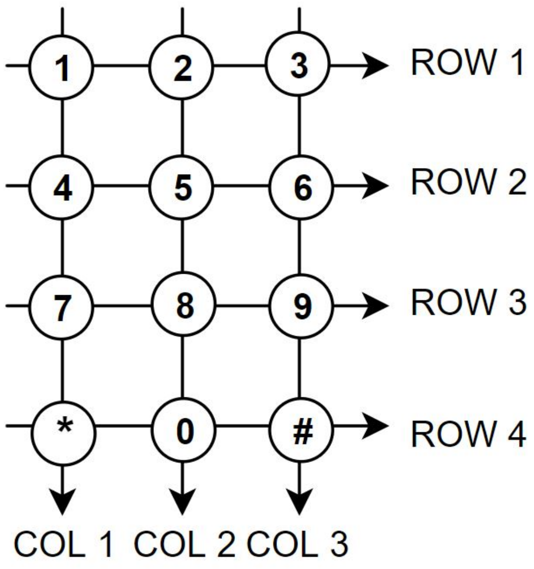

Figure 1: Circuit Diagram of Keyboard

OUTPUT ARRANGEMENT		
OUTPUT PIN NO.	**SYMBOL**	**FITkit3**
1	**COL 2**	**PTA 25**
2	**ROW 1**	**PTA 24**
3	**COL 1**	**PTA 28**
4	**ROW 4**	**PTA 26**
5	**COL 3**	**PTA 29**
6	**ROW 3**	**PTA 27**
7	**ROW 2**	**PTA 7**

Table 1: Output Arrangment of Keyboard

These pins was necessary to connect to some free pins at the development board *FITkit3*. Although the MCU itself offers it dozens of *GPIO* outlets (pin field P1 - see kit diagram[2], page

[1]https://minerva.php5.cz/
[2]https://minerva.php5.cz/doc/schemata/Minervafull.pdf

6 and 8), universal use it on FITkit3 only **19** outlets. Wires described by **PTAx**, *PTEx* and six another outlets which name starting with *MCU_I2C0*, respectively *MCU_SPI2*. Specifically in our case was selected the wires **PTA7, PTA24-29**. On these *pins* of **PORT A**, on the board **FITkit3**, was connected the individual pins of keyboard subsequently.

As you can see in Table 1, all rows, and columns of the keyboard are assigned the relevant wires. In this order was gradually connected on the individual selected pins at the development board, according to its physic location.

3 Application Control

The control of application takes place exclusively through the keyboard. The users have available all buttons, which have specific functionality. The buttons can be divided into groups in which will be buttons, which have the same functionality. The individual buttons will be numbered according to numbers in Figure 1. The first group contains the button **1** and **11** and the functionality of these buttons are the simplest. Each time you press these buttons, the relevant key button (number 0 or 1) will be stored to buffer for encoding to *Morse Code*.

The second group contains the remaining buttons, which allows entered the characters to encode to *Morse Code*. This group includes the buttons **2-9**. The functionality of these buttons is the same as on the older mobile keyboards. Each button allows entered **three**, respectively **four** letters and **one** number. **Four** letters only in the case of buttons **7** and **9**. The input method is simple, the first pressing means the first **letter** and until to the last pressing, which means the **number**. Between the individual press is a timeout, which determines when the character will be recognized. In this application was this timeout, after experimental verification, set to a value of **1 second**. Except for standard letters, which are viewing on the individual buttons, the buttons **7** and **9** contains on the **fourth** position letter **Q**, respectively letter **Z**. The capacity of these buttons are **five** and capacity of remaining buttons are **four**.

The last group contains the buttons **10** and **12**. As was mentioned in the Introduction, these buttons serve for changing the speed of reproduction frequency of individual encoding characters. The first button from this group serves for decreasing the frequency speed and the second serve for the increasing this frequency speed. However, the user does not have full authority over these changes, because reproduction would consequently not make sense in some situations. The range of allowed changes it represents by the change in both directions on **11 levels**, compared to the initial level.

4 Implementation

In the introductory section of the program is occurs the hardware initialization. Among the most interesting parts includes connection the wires of the keyboard on the relevant pins. In our program, we chose to configured the **columns** as **input pin**. As was indicated in the previous section, columns are connected on the pins of *PORT A*, concretely **25**, **28**, **29**. On all these pins was *cleared* Interrupt Status Flag, *enabled* interrupt on *failing edge*, *selected* option for *GPIO* pins, *enabled* pull resistor and subsequently selected *Pull-up* resistor variant on these pins. On the contrary to columns, were the **rows** configured as the **output** pins. The individual pins, again on the *PORT A* (**7**, **24**, **26**, **27**), was configured as *GPIO* outputs and for *high drive strength* according to recommendation in[3].

[3]https://cache.freescale.com/files/microcontrollers/doc/data_sheet/K60P144M100SF2.pdf

The next section that will be introduced is the processing the *interrupts* from the keyboard and subsequently recognition and processing of the individual characters, entered from this keyboard. Since the columns were configured as input pins when the button is pressed the relevant column caused *IRQ interrupt*. After the detection of the concrete column, on which was the button pressed, follows the scanning of all rows. It is performed by adjusting the individual pins to **active** level and subsequently the tracking of the changes on the current column. In the next step, is known the pressed button and can be processing.

In the case of the button with capacity equal to **one**, so the buttons **1**, **10**, **11** and **12** can be immediately performing the relevant action of processing. When it is button **1** or button **11**, it will be immediately after the detection stored to the buffer of received characters. If the case of two remaining buttons, will be changed the corresponding rate of the reproduction of the characters. In the case of the buttons, which have the capacity equal to **four** or **five** will be the current characters stored to the buffer after the **expiration of timeout** or after the **pressing** of another button. The mentioned **timeout** is solved by **Periodic Interrupt Timer**[4] on the *PIT Channel 1*.

The last part to which we look in the frame of implementation itself is the *reproduction* of the characters. To the *buffer* of received characters not stored the characters itself but only the indexes to the *map*, which contains the relevant *Morse code* for these characters. In the case, when the buffer contains some characters for reproducing the individual parts of **Morse code** of some character are gradually stored to the *reproducing cache*, from where it will be subsequently reproduced.

The *Morse encoding* is adheres to internationally valid rules. It means, that in the program is established one **unit**, according to which all components of **Morse code** are controlled. The length of a **dot** is **one** unit. The length of the **slash** is **three units**. The space between parts of the same character is **one** unit. The space between characters is **three** units. When using the buttons for change the rate of frequency of reproduction the chars, (buttons **10** and **12**) is this **unit** decreasing or increasing. All these parts of **Morse Code** and its sounds are accompanied by flashing LEDs for the duration of the sound. The **dot** represents by the LED **D9**, the **slash** by the LED **D10** and the **both kind of spaces** gradually represents by LEDs **D11**, respectively **D12**.

5 Conclusion

The designed and implemented **Embedded Application** is fully functionally. Meets **all** the requirements that were put before its creation. Moreover, users have a chance encode **not only** letters, but they can encode **all the numbers** too. Functionality is complemented by **flashing of LEDs** that help beginners to distinguish individual parts of the code. The sound output of this application is fully compatibles with requirements of **International Morse Code**.

[4]http://mcuxpresso.nxp.com/api_doc/dev/210/group__pit.html