Department of Computer Science, FEECS,
VŠB - Technical University of Ostrava, 2019

Computer Architectures and Parallel Systems
(APPS - Architektury počítačů a paralelních systémů)

Laboratory Exercises

Ing. Petr Olivka, Ph.D., email: petr.olivka@vsb.cz
Ing. David Seidl, Ph.D., email: david.seidl@vsb.cz

Contents

List of Figures

Part I

Microcomputer Programming

Chapter 1

Overview of Used HW & SW

1.1 Microcomputer and Development Platform

Microcomputer (Microcontroller, Microprocessor, Monolitic Computer) is a small computer implemented on single chip. It must contains processor (CPU), RAM and Flash memories and peripheral devices.

On the market is nowadays in addition to well-known Intel and AMD producers of CPUs for personal computers also many well-known producers of microcomputers, e.g. Atmel, Microchip, NXP, STMicroelectronics, Texas Instruments, Fujitsu, ARM (not manufacturer), etc. The assortment of product is very wide and it covers requirements of industry.

In past ten years 32-bits processors ARM started to promote. Their massive production very decreased their price and it slowly relegate older 8 and 16-bit microcomputers to edge of interest.

Therefore the first half of laboratory exercises in the subject Computer Architectures and Parallel Systems will be aimed to programming a ARM microcomputer.

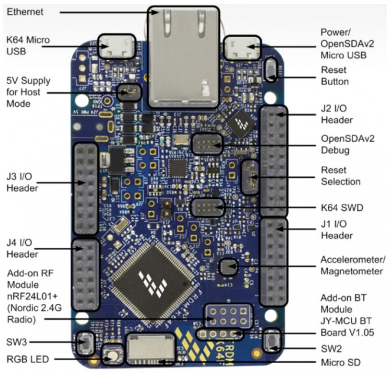

Figure 1: FRDM-K64F overview

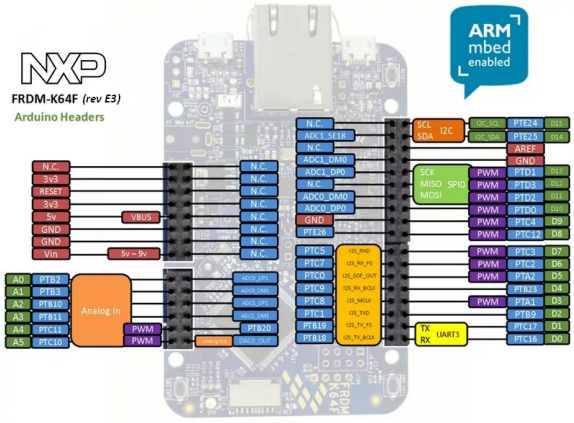

Figure 2: FRDM-K64F pinout (http://development.mbed.org)

1.1.1 Kit FRDM-K64F

The high-end microcomputer NXP from K64 family was selected for laboratory exercises. It contains 32-bits CPU core ARM Cortex M4 with 1 MB Flash memory and 256 KB RAM. It is working at frequency 120 MHz. This microcomputer is part of development kit FRDM-K64F, see figure 1, together with addition peripheral devices and programming/debugging interface.

The Kit FRDM-K64F is depicted in figure 1. The pinout of this kit is visible in figure 2. Names of pins is very important for subsequent programming.

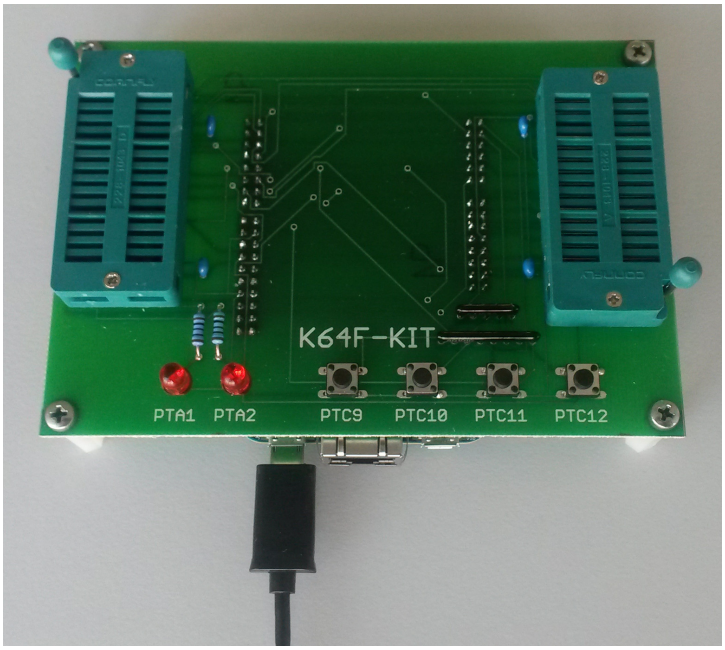

Figure 3: Extension K64F-KIT

An inseparable part of this text is also following datasheets:

- `frdm-k64f-usrguide.pdf` - description of FRDM-K64F
- `k64f-refman.pdf` - K64 Sub-Family Reference Manual

More information about this products is available at `http://www.nxp.com`.

1.1.2 Extension K64F-KIT

The FRDM-K64F Kit is for laboratory exercises extended by K64F-KIT. This kit is depicted in figure 3. The FRDM-K64F is under this kit.

This kit contains four buttons connected to pins PTC9, PTC10, PTC11 and PTC12. Names of pins is also visible on the K64F-KIT. Two addition LEDs is connected to pins PTA1 and PTA2.

For laboratory exercises is very important two ZIF (Zero Insertion Force) sockets on the top. They are used for additional modules used for individual laboratory exercises. These modules will be described later.

The full scheme of K64F-KIT is shown in figure 4.

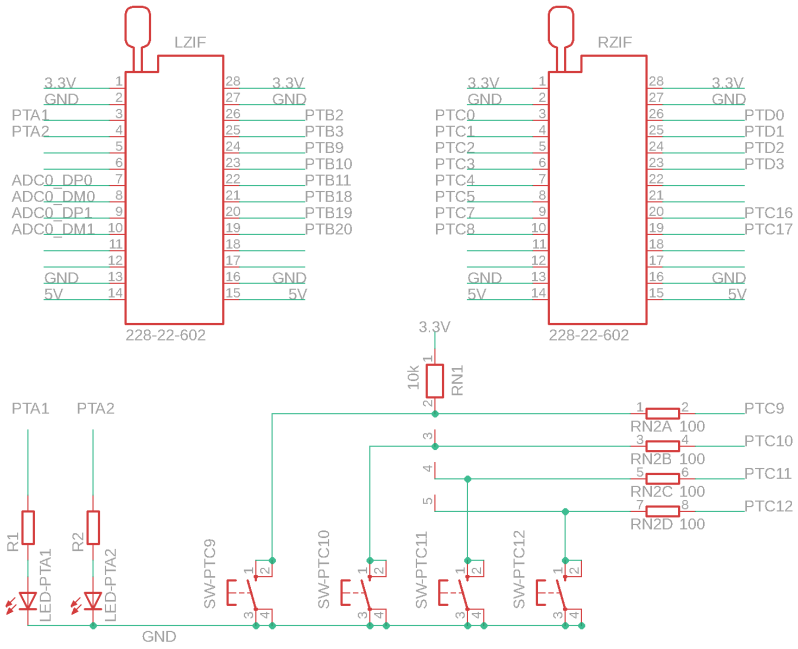

Figure 4: K64F-KIT Scheme

1.1.3 FRDM-K64F Interfacing with Computer

The interconnection between FRDM-K64F and a computers is realized by an USB cable. On the FRDM-K64F must be the USB cable connected to the left micro USB connector, see figure 3

1.2 Programming Framework Mbed

The project Mbed https://www.mbed.com is widely supported by many big international manufacturers. Its aim is to create universal programming framework which can be used on a large number of development platforms: https://developer.mbed.org/platforms/. Many of these boards have Arduino compatible connectors to be possible use Arduino shields. Mbed also suports modern technology IoT.

1.2.1 Online Compiler

The web page Mbed https://developer.mbed.org/compiler/ offers for developers online compiler with simple integrated development environment (IDE), see figure 5.

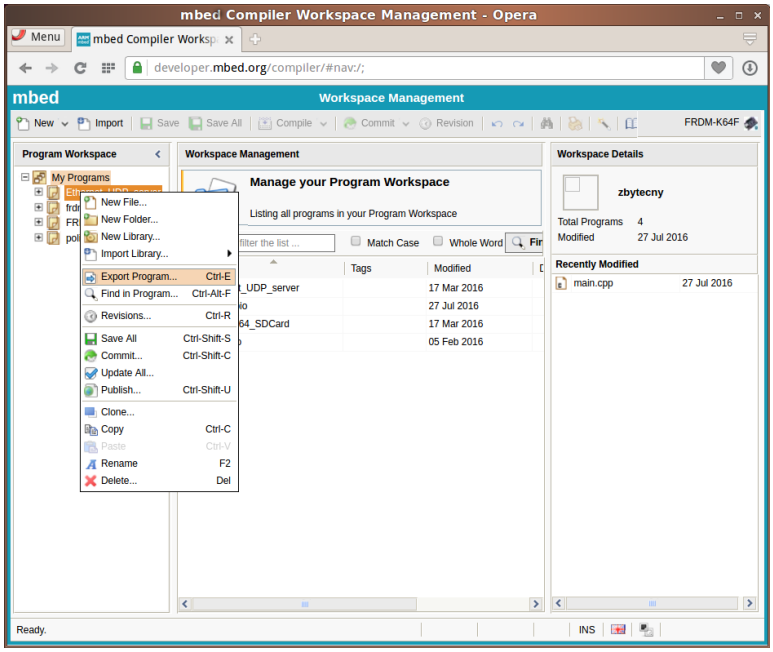

Figure 5: Mbed online compiler

This compiler is neither designed for professional work not for education. It can be used to compose required project from many available modules and then it is very usual procedure to export this project for other IDE.

During laboratory exercises this online compiler will not be used.

1.2.2 Mbed API (Application Programming Interface)

During programming in laboratory exercises only a small fraction of the whole API will be used. Programming will be mainly focused to programming ports, which are usually known as GPIO (General Purpose Input and Output). Also a delay function will be necessary to use.

1.2.3 GPIO (Ports) – Classes DigitalIn and DigitaOut

The API for GPIO programming contains three classes: `DigitalIn, DigitalOut and DigitalInOut`. Short description of these three classes is below:

```
class DigitalIn
{
  public:
    // constructor
    DigitalIn( PinName pin );
    // read pin state
    int read();
    operator int ();
};

class DigitalOut
{
  public:
    // constructor
    DigitalOut( PinName pin );
    // read pin state
    int read();
    operator int ();
    // change output value
    void write( int value );
    operator= ( int value );
};

class DigitalInOut
{
  public:
    // constructor
    DigitalInOut( PinName pin );
    DigitalInOut( PinName pin, PinDirection direction,
                  PinMode mode, int value)
    // set pin I/O direction
    void output();
    void input();
    // read pin state
    int read();
    operator int ();
    // change output value
    void write( int value );
    operator= ( int value );
};
```

A fragment of code how to use these classes in program to control LEDs and read buttons follows:

```
DigitalOut g_led1( PTA1 );  // LED
DigitalOut g_led2( PTA2 );  // LED
DigitalIn  g_sw1( PTC9 );   // Button
DigitalIn  g_sw2( PTC10 );  // Button

...
if ( g_sw2 ) g_led2 = 1;    // check buttons state
else         g_led2 = 0;

g_led1 = !g_led1;           // invert state of led1

while ( g_sw1 == 0 );       // wait while button is pressed
...
```

1.2.4 Time Control – Functions wait

Mbed API contains two functions for delay:

```
void wait( float t_sec );        // delay seconds
void wait_ms( int t_msec );      // delay miliseconds
```

The both functions delay a program for a required time. The disadvantage of these functions is that the time is measured directly by the CPU. Thus no other part of the program can be executed.

1.2.5 Time Control – Class `Ticker`

The Mbed API contains a few classes to control a timing and time events. One of those classes is class `Ticker`. This class is aimed to control periodic events in program. A short introduction of this class follows:

```
class Ticker {
public:
    Ticker() { ... }

    // attach callback function
    void attach_us(Callback<void()>func,us_timestamp_t t);

    // attach callback object
    template<typename T, typename M>
    void attach_us(Callback<void()>(T *obj, M method), us_timestamp_t t)

    // attach callback function
    void attach(Callback<void()>func,float t);

    // attach callback object
    template<typename T, typename M>
    void attach(Callback<void()>(T *obj, M method), float t)

    // destructor
    virtual ~Ticker()
    {
        detach();
    }

    // detach callback
    void detach();
  ...
};
```

Class `Ticker` allows to attach only one callback function or one object and its method. A short fragment of following code introduce how the `Ticker` can be used:

```
DigitalOut g_led1( PTA1 );
DigitalOut g_led2( PTA2 );

void fun_blink()
{
    g_led1 = !g_led1;
}

class Blinker
{
public:
    Blinker( DigitalOut &t_digout ) : m_digout( t_digout ) {}
    void blink() { m_digout = !m_digout; }
protected:
    DigitalOut &m_digout;
};

int main()
{
    Ticker l_t1, l_t2;
    Blinker l_blinker( g_led2 );
    l_t1.attach_us( callback( fun_blink ), 100000 );
    l_t2.attach_us( callback( &l_blinker, &Blinker::blink ), 500000 );
    ...
    while( 1 );
}
```

The advantage of class `Ticker` is that the timing is CPU independent. Callbacks are periodically called using interrupt.

Caution! Callback functions and methods must not use functions wait and printf.

1.3 MCUXpresso IDE

NXP, the producer of FRDM-K64F, supports for the development its own IDE - MCUXpresso. This IDE is based on Eclipse and it is freely available from the web site http://www.nxp.com. The MCUXpresso is modern IDE fulfilling all requirements for the professional programming and its usage is intuitive as usage of many others known IDEs.

The main window of MCUXpresso is visible in figure 6. On the left side is visible workspace with project. In the middle is visible source code in the editor window.

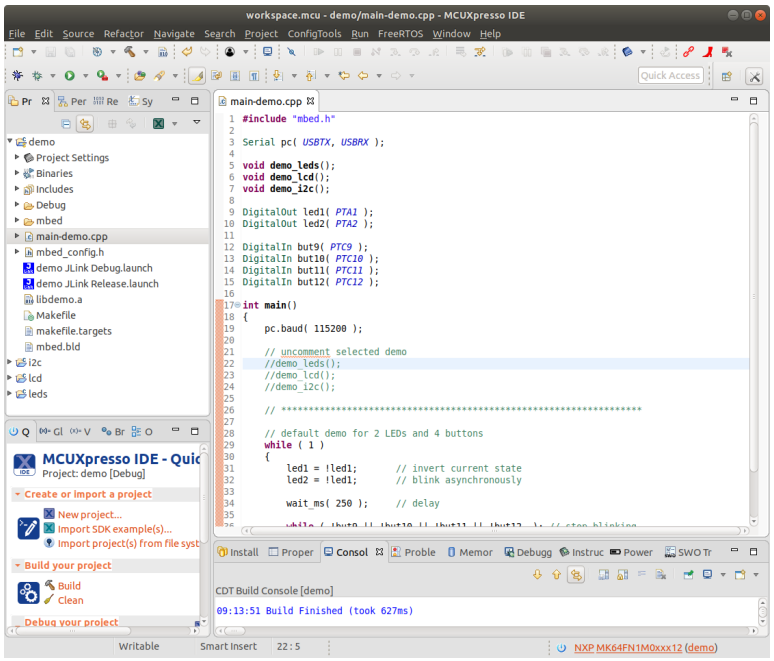

Figure 6: Main window of MCUXpresso

1.3.1 Program Examples - Archive `apps-labexc.zip`

The archive with program examples is prepared for laboratory exercises. It contains simple source code to easily start programming in individual exercises. This archive `apps-labexc.zip` contains the whole workspace in directory `workspace.mcu`.

Content of workspace `workspace.mcu`

```
./workspace.mcu       -> Workspace for MCUXpresso
./workspace.mcu/leds  -> Directory with project for LEDs
./workspace.mcu/lcd   -> Directory with project for LCD module
./workspace.mcu/i2c   -> Directory with project for I2C bus
./workspace.mcu/demo  -> Demo examples for individual exercises
```

Content of directory `./workspace.mcu/leds`:

```
./main-leds.cpp       -> Source code for LEDs
```

Content of directory `./workspace.mcu/lcd`:

```
./main-lcd.cpp        -> Source code for LCD
./lcd-lib.cpp         -> Source code with functions for LCD
./lcd-lib.cpp         -> Header file with functions for LCD
./font8x8.cpp         -> Source code with fixed size font 8x8
```

Content of directory `./workspace.mcu/i2c`:

```
./main-i2c.cpp       -> Source code for I2C bus
./i2c-lib.cpp        -> Source code with functions for I2C bus
./i2c-lib.h          -> Header files with functions for I2C bus
./si4735-lib.cpp     -> Function for SI4735 initialization
./si4735-lib.h       -> Header file with function for SI4735
```

Content of directory `./workspace.mcu/demo`:

```
./main-demo.cpp      -> Demo for individual exercises
```

1.3.2 Run and Debug Programs in MCUXpresso

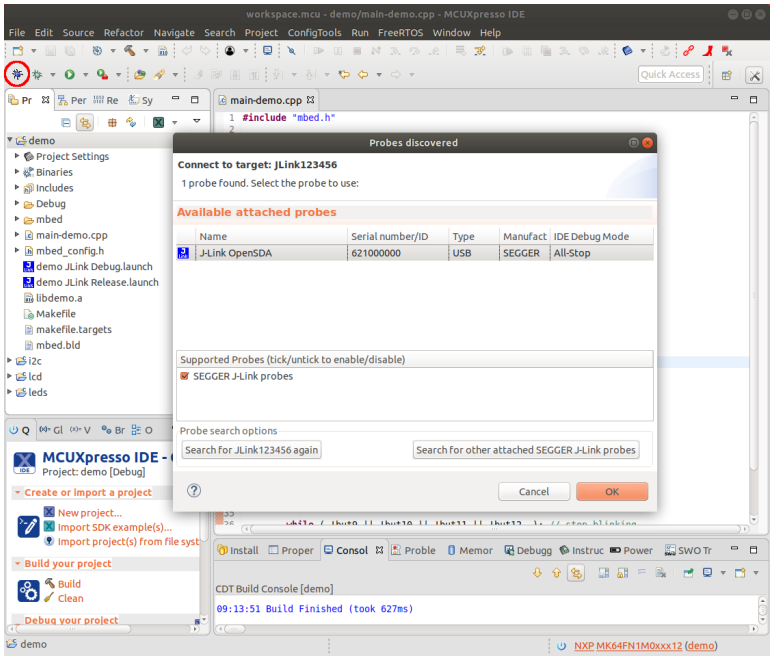

Figure 7: Start debugging in MCUXpresso

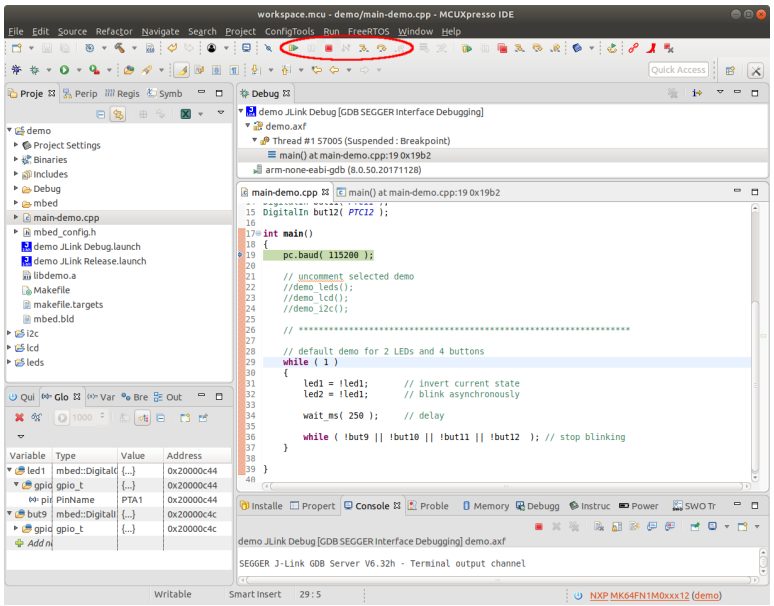

Figure 8: Run or debug program in MCUXpresso

To run or debug programs in MCUXpresso it is necessary to perform

two steps.

The first step is to select Debug from the toolbar of MCUXpresso, in figure 7 is marked with the red circle. The new layout of MCUXpresso is visible in figure 8.

Now it is possible to perform the second step: run or debug a program. All possible selections are marked in figure 8 with the red ellipse.

1.3.3 Output on Serial Line

In a program for FRDM-K64F it is possible to use method `printf` of class `Serial` to display some information when program is running.

```
Serial g_pc(USBTX, USBRX);
...
  g_pc.baud(115200);
  g_pc.printf("Hello␣world!\r\n");
...
```

The output is redirected via serial line into computer with the MCUXpresso. This output can be displayed by `minicom` started in a terminal window:

```
$ minicom -D /dev/ttyACM0
```

To get help press the `CTRL-A-Z` or quit by CTRL-A-Q. The speed must be set to **115200 bauds** and **hardware flow control disabled**.

An another possibility is to set speed directly by command `stty` and then display data from FRDM-K64F by program `cat`:

```
$ stty -F /dev/ttyACM0 115200 raw -crtscts
$ cat /dev/ttyACM0
```

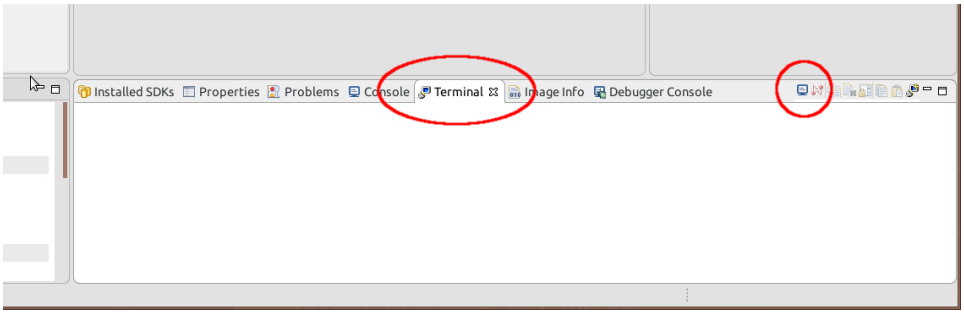

Figure 9: Terminal TAB in MCUXpresso

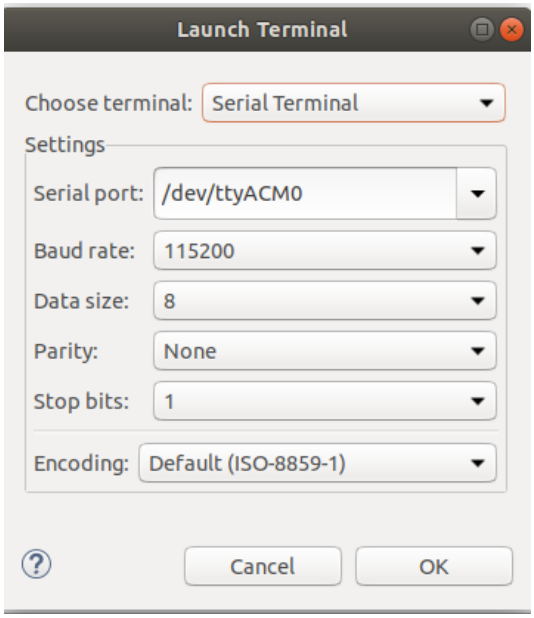

Figure 10: Set terminal in MCUXpresso

The third alternative is the use of a terminal directly in MCUXpresso.

The tab Terminal must be selected at the bottom of MCUXpresso. In figure 9 it is marked by the red ellipse.

The connection with proper terminal will be open using the Terminal icon marked in figure 9 by red circle. The proper settings of terminal is visible in figure 10.

1.4 GCC/G++ Compiler - short overview

The GNU compiler GCC and G++ is open source project developed to compile C of C++ source code for many CPUs, including ARM. This compiler is integrated in MCUXpresso as well as in many others IDEs.

In this chapter the brief summary of features of C language will be performed.

1.4.1 Data Types

The following data types can be used in programs:

- signed char: -128 ÷ 127
- signed short: -32768 ÷ 32767
- signed int, signed long: -2147483648 ÷ 2147483647
- unsigned char: 0 ÷ 255
- unsigned short: 0 ÷ 65535
- unsigned int, unsigned long: 0 ÷ 4294967295

The keyword `unsigned` have to be used for all unsigned variable. The keyword `signed` is optional.

1.4.2 Number Format

Numbers can be written in four formats:

```
Decimal:
   int i   =  10578;
   char ch = -10;
   float f =  13.54;

Hexadecimal:
  int i   = 0x12B7;
  char ch = 0xF1;

Binary:
  int i   = 0b1101001010010110;
  char ch = 0b10001101;
```

```
Octal:
  int i   = 0737;
  char ch = 015;
```

One number can be written in four formats: 10 = 0xA = 0b1010 = 012. It is still one and the same number!

1.4.3 Arithmetical, logical and binary operators

1.4.4 Arithmetical + - * / %

The first four operators are well-known from an elementary school. The fifth operator is the modulo and the result is a remainder of division, e.g: `11 % 2 = 1, 13 % 5 = 3, 31 % 31 = 0`.

Short form of arithmetic operations:

```
x++, ++x     x = x + 1
x--, --x     x = x - 1
x += y       x = x + y
x -= y       x = x - y
x *= y       x = x * y
x /= y       x = x / y
x %= y       x = x % y
```

1.4.5 Logical && || !

```
Logical conjunction (AND)     Logical disjunction (OR)
 false && false  => false      false || false    => false
 false && true   => false      false || true     => true
  true && false  => false       true || false    => true
  true && true   => true        true || true     => true

Logical negation
         !true   => false              !false    => true
```

In C language is `true` any nonzero value, `false` is zero.

1.4.6 Binary & | ^ ~ << >>

```
Binary AND       Binary OR
0 & 0 => 0       0 | 0 => 0
0 & 1 => 0       0 | 1 => 1
```

```
1 & 0 => 0        1 | 0 => 1
1 & 1 => 1        1 | 1 => 1

Binary XOR        Binary negation
0 ^ 0 => 0        ~1 => 0
0 ^ 1 => 1        ~0 => 1
1 ^ 0 => 1
1 ^ 1 => 0

Bit shift
0b01 << 1 => 0b10
0b10 >> 1 => 0b01
```

Examples:

```
5 & 3 = 1 because 0b101 & 0b011 = 0b001
5 | 3 = 7 because 0b101 | 0b011 = 0b111
5 ^ 3 = 6 because 0b101 ^ 0b011 = 0b110
~5 = 250  because ~0b101 = 0b11111010,
          if 5 is 8-bit unsigned number

5 << 3 = 40 because 0b101 << 3 = 0b101000
5 >> 3 = 0  because 0b101 >> 3 = 0b000000
```

Short form of binary operations:

```
x >>= y   x = x >> y
x <<= y   x = x << y
x &= y    x = x & y
x |= y    x = x | y
x ^= y    x = x ^ y
```

The logical and binary operators are very often mistaking:

```
char x = 5;
char y = 2;

if (x && y) {...} //true
if (x & y)  {...} //false
```

Chapter 2

Laboratory Exercises with Microcomputer

2.1 LEDs (Light-Emitting Diodes)

The first laboratory exercise is aimed to control LEDs. In this exercise is used an extension module with LEDs. The K64F-KIT with the mounted LED module is depicted in figure 11. This module contains eight red LEDs and two RGB LEDs.

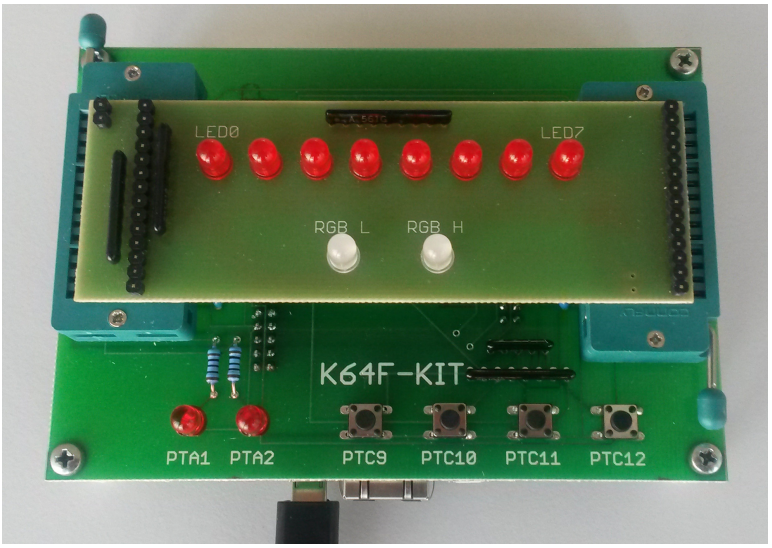

Figure 11: K64F-KIT with LEDs module

2.1.1 Connection of Red LEDs to Microcomputer

Microcomputer GPIO pins allow direct connection of a LED through a resistor. The resistor limits the flowing current through LED and it also prevents an overloading of GPIO pins.

The full scheme of red LEDs connection is visible in figure 12. From this scheme it is clear that all LEDs are activated (switched on) by logical level 1 on any GPIO pin. Eight LEDs LED0÷7 are connected to eight pins PTC0÷PTC8 (not PTC6!).

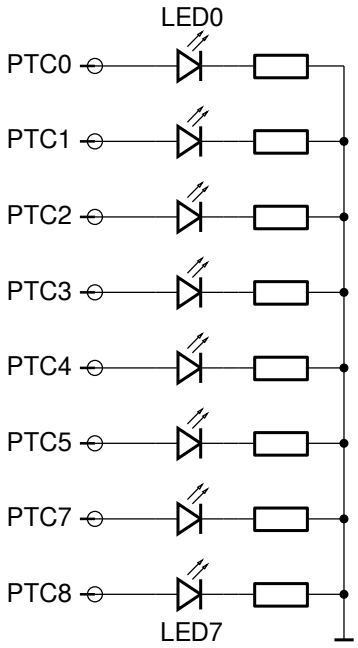

Figure 12: Scheme of red LEDs connection

2.1.2 LED Brightness Control by PWM

2.1.2.1 Description

The pulse width modulation (PWM) is the simplest method of signal conversion from a digital to analog form. The GPIO pin allows to set output state only to logical level 0 and 1. But the periodic alternation offers the possibility to set a different timing of both output levels. Finally this irregular alternation, well-known as duty cycle, may control output power (current or voltage).

Therefore the PWM can be also used to control the brightness of LEDs. The size of duty cycle is expressed as a part of time period T in percents. A few examples of signals with different duty cycle are visible in figure 13.

2.1.2.2 Task Examples

- Control brightness of all LEDs, set different brightness for every LED.
- Slowly, one by one, switch on all LEDs (slowly switch on first LED, then second, etc.). Then slowly switch off LEDs.
- 4 LEDs slowly switch on and parallelly 4 LEDs slowly switch off.

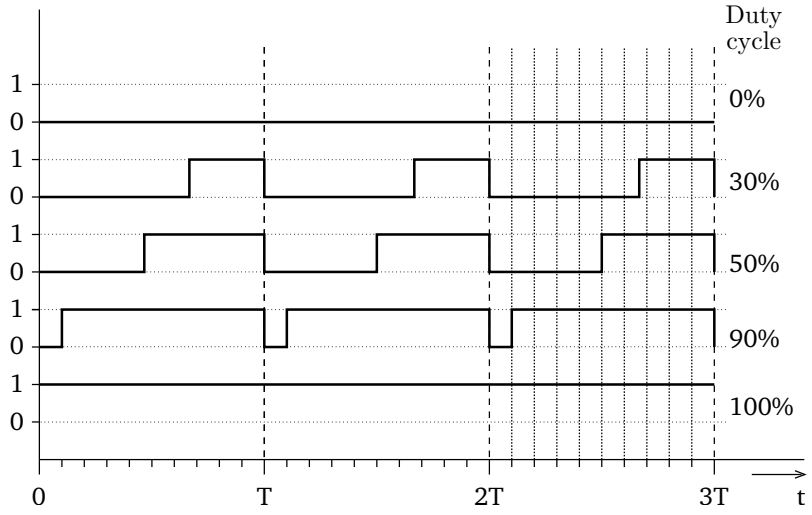

Figure 13: Timing of signals with different duty cycle

- Use two buttons to select a LED and by two buttons set its brightness.

2.1.2.3 Steps of Solutions

- Blinking with one selected LED.
- Blinking with selected LED with duty cycle 50%..
- Find time period $T = 1/f$ when blinking is not visible.
- Control brightness of single LED, duty cycle is set in percents.
- Control brightness of two LEDs parallelly, duty cycles for both LEDs is different. Split a time period to individual milliseconds and check the duty cycle of all LEDs.
- Finish task.

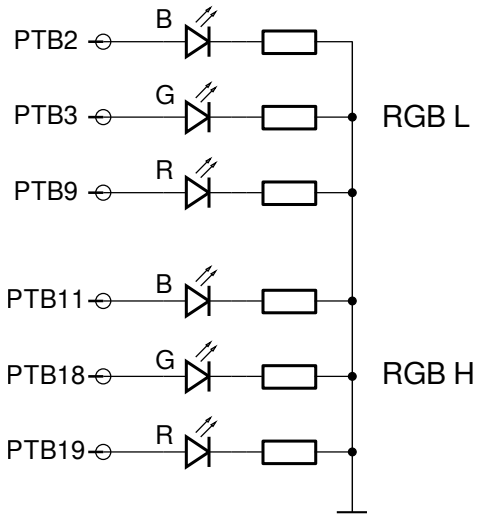

Figure 14: Scheme of RGB LEDs connection

2.1.3 Mixing of Colors - RGB LEDs

2.1.3.1 Description

A computer technology uses for visualization in most cases an additive principle of a color mixing. This principle is based on mixing of three base colors: red, green and blue (RGB). These three colors can compose any color in a visible spectrum.

The additive color mixing can be realized by RGB LED, where brightness of all colors are controlled by PWM (principle described above). The scheme of the RGB LEDs connections to microcomputer is depicted in figure 14. The first RGB-L LED is connected to pins PTB2, PTB3 and PTB9. The seconsd RGB-H LED is connected to pins PTB11, PTB18 and PTB19.

2.1.3.2 Task Examples

- Switch on selected colors with different brightness.
- Slow transition between two colors.
- Switch on RGB LED with color according to HTML value #RRGGBB
- Using buttons select and set individual colors of RGB LEDs.

2.1.3.3 Solution

Solution is similar to PWM control.

2.1.4 Control of blinking frequency

2.1.4.1 Description

The aim of this task is to control LED blinking with given frequency. The required time period for given frequency is $T = 1/f$. Caution! This time period must be split to two parts, where one half of period T will be LED on and the second half of T will be LED off.

2.1.4.2 Task examples

Tasks for frequency control is similar as for PWM. But aim is to control frequency, not duty cycle.

2.1.4.3 Solution

There are two possibilities.

The first solution is to use one counter for one LED. This counter is incremented every millisecond and it is checked for overflow.

The second solution is to use one counter for all LEDs. The modulo will be computed for all LEDs and their time periods. If the modulo is 0, then state of LED will be changed.

The second solution is little bit easier, the first solution also allows to control phase of blinking.

2.2 LCD - RGB Graphic Display

Graphic LCD displays are nowadays used in many devices and they slowly crowding out text LCD displays. They offer better interface for user, but they required higher performance of connected microcomputer.

The extension module for laboratory exercise implements small graphic LCD display with resolution 240×320 points, 16-bit color depth, SPI interface and ILI9341 controller. This module is visible in figure 15. More information about this LCD module can be found in datasheet `ili9341.pdf`.

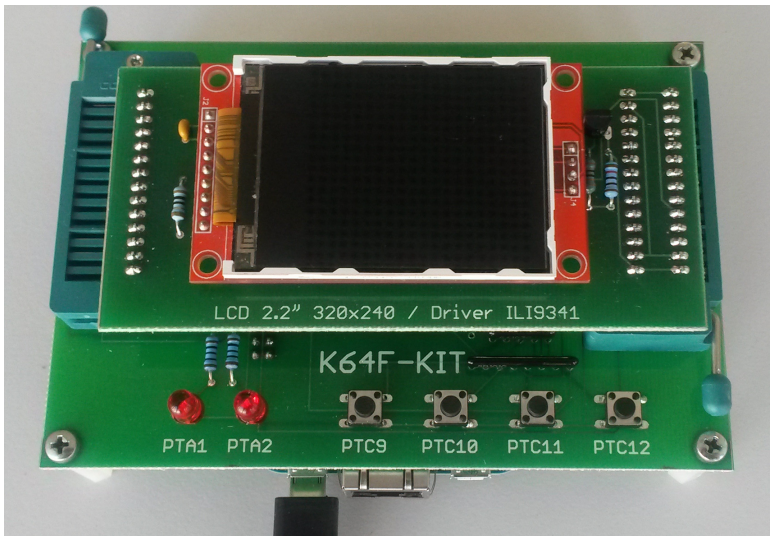

Figure 15: K64F-KIT with LCD module

2.2.1 LCD Module Connection

The interconnection between LCD module and microcomputer is depicted in figure 16.

The SPI interface SCK/MISO/MOSI is directly connected to SPI interface of microcomputer. The signal RESET is used to reset LCD controller and it is connected to PTC1. The signal chip select CS, connected to PTC0, is used to activate SPI interface of LCD controller. The signal D/C determines between data and commands on SPI. The backlight of LCD module is controlled by pin PTC3 through a transistor.

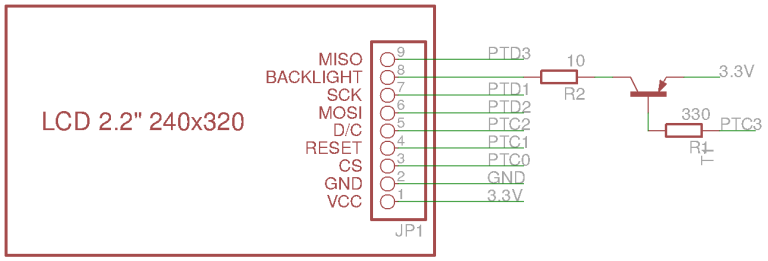

Figure 16: Scheme of LCD module connection to microcomputer

2.2.2 LCD Programming Interface

The programming interface contains only three functions:

```
// LCD controller initialization
void lcd_init();

// draw one pixel to LCD screen
void lcd_put_pixel( int x, int y, int color );

// clear screen
void lcd_clear();

// HW reset of LCD controller
void lcd_reset();
```

The first function `lcd_init` have to be called once at begin of program. This function initializes ILI9341 controller. The function `lcd_clear` clears the screen.

The function `lcd_put_pixel` draw pixel at position `[x,y]` with a given color. The color is 16-bit value in RGB form 5-6-5. This form is for clarity in more details described in table 2.1.

Table 2.1: 16-bit RGB color format 5-6-5

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
color	R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B

Function `lcd_reset` is used only internally to perform HW reset of LCD controller.

Program examples are in archive `apps-labexc.zip`.

2.2.2.1 Matrix Font

The controller of graphic LCD is not able in most cases to display any text information. To display text information is is necessary to program it.

For the laboratory exercise the fixed size font 8×8, in file font8x8.cpp (from https://github.com/dhepper/font8x8), is prepared. In the source file the font is stored as two dimensional array 256×8 bytes. The form of this array is in following example:

```
uint8_t font8x8[ 256 ][ 8 ] = {
    ....
  {0x3E, 0x63, 0x73, 0x7B, 0x6F, 0x67, 0x3E, 0x00}, //U+0030 (0)
  {0x0C, 0x0E, 0x0C, 0x0C, 0x0C, 0x0C, 0x3F, 0x00}, //U+0031 (1)
  {0x1E, 0x33, 0x30, 0x1C, 0x06, 0x33, 0x3F, 0x00}, //U+0032 (2)
  {0x1E, 0x33, 0x30, 0x1C, 0x30, 0x33, 0x1E, 0x00}, //U+0033 (3)
  {0x38, 0x3C, 0x36, 0x33, 0x7F, 0x30, 0x78, 0x00}, //U+0034 (4)
  {0x3F, 0x03, 0x1F, 0x30, 0x30, 0x33, 0x1E, 0x00}, //U+0035 (5)
  {0x1C, 0x06, 0x03, 0x1F, 0x33, 0x33, 0x1E, 0x00}, //U+0036 (6)
  {0x3F, 0x33, 0x30, 0x18, 0x0C, 0x0C, 0x0C, 0x00}, //U+0037 (7)
  {0x1E, 0x33, 0x33, 0x1E, 0x33, 0x33, 0x1E, 0x00}, //U+0038 (8)
  {0x1E, 0x33, 0x33, 0x3E, 0x30, 0x18, 0x0E, 0x00}, //U+0039 (9)
  ....
  };
```

The form of stored characters are depicted in figure 17.

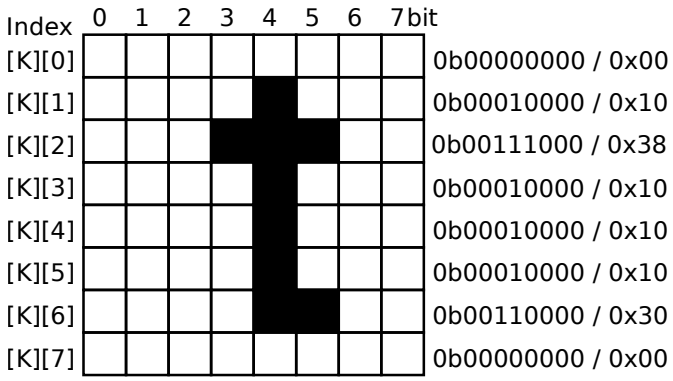

Figure 17: Format of matrix font

2.2.2.2 Task Examples

- Program function that displays one character at position [x,y].
- Program function that displays string horizontally/vertically.
- Slowly move text on the screen.
- Display time in format HH:MM:SS with blinking separator ':'. Time will be set using buttons.
- Stopwatches.
- Countdown.
- Draw line/rectangle/circle.
- Regulate backlight level using PWM.

Tasks will be in laboratory combined and modified.

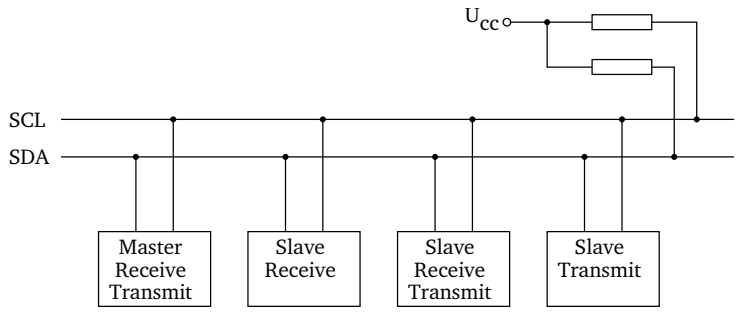

Figure 18: Typical configuration of I^{2}C bus

2.3 I^{2}C Bus

2.3.1 Introduction

I2C (Inter-Integrated Circuits) bus was introduced by Philips Semiconductor for communication between integrated circuits and microcomputers.

I2C is bidirectional bus realized by two lines (wires). Sometimes it is incorrectly called a serial line.

2.3.1.1 Description

The I2C bus consists of two lines (wires). The line **SDA** is used for bidirectional data transfer. The line **SCL** is the clock signal for synchronization. The both lines are operating in two logical levels 0 and 1.

In a idle state the both lines are in level 1. This state is kept by two pull-up resistors. Integrate circuits outputs are designed as *open collector* (open drain). This design guarantees the bidirectional behaviour of lines.

The base schema of I^{2}C is shown in figure 18. On the bus can be connected more devices. Usually one device is marked as *Master*, which control communication. Other devices are *Slaves*.

Slave devices, after call from master, receive or transmit data. On the bus can be connected more master devices. But this configuration is not simple and it is not suitable for laboratory exercises.

2.3.1.2 Transmission of Bits

Transmission of bits on the bus is synchronous. The synchronization is performed by clock signal. Just one bit is transferred in one clock cycle. The transfer of value 0 and 1 is depicted in figure 19a. From this figure is clear, that data signal SDA can be changed only when clock signal SCL is

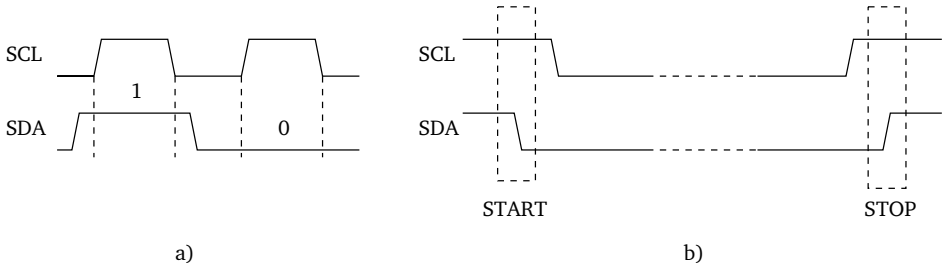

Figure 19: Course of signals on I^2C

in level 0. If SCL signal is 1 and the SDA signal is changed, it is control signal.

2.3.1.3 Control Signals START and STOP

Because of limited number of wires the I^2C bus can has only two control signals. These signals are START (S) and STOP (P). The both signals are depicted in figure 19b. The falling edge of SDA is the signal START and the rising edge of SDA is the signal STOP. The I^2C bus is after STOP signal in idle state.

2.3.1.4 Transmission of Bytes

The amount of data transmitted between START and STOP signal is not limited. But the transmission between ICs is not continuous bit stream. The transmission is split to bytes. The diagram of one byte transmission is depicted in figure 20. From this figure is clear that MSB bit is transmitted as the first one and LSB is the last one.

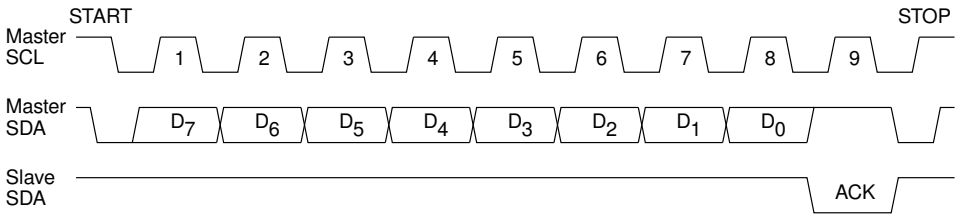

Figure 20: Transmission of one byte with acknowledgement

2.3.1.5 Acknowledgement

The reliability of a communication is very important. Thus the I2C standard defines a confirmation of every transmitted byte. It is visible in figure 20. Every circuit receives after one byte transmission an acknowledgement. The acknowledgement is transferred as ninth bit in opposite direction than the byte was transmitted. The acknowledgement can be positive ACK, defined as 0, or negative NACK, defined as 1.

2.3.1.6 Simplified scheme of communication

The simplified scheme of communication will be used in following text. In figure 21 the transmission of one byte from master to slave is depicted together with the acknowledgement from slave. Symbols S and P represent START and STOP signal.

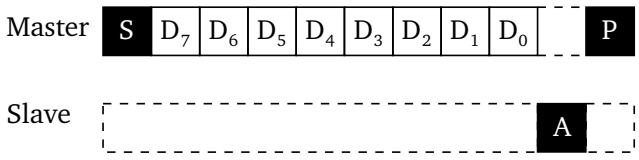

Figure 21: Simplified scheme of communication on the I^2C bus

2.3.1.7 Addressing

I2C bus allows connect more ICs parallely. To be possible communicate directly with selected IC it is necessary to assign unique address to every IC. The IC address must be transmitted from master as the first byte after START signal. The format of address byte is depicted in figure 22.

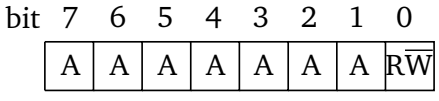

Figure 22: Address format of any I^2C device

The lowest (LSB) bit in address byte defines direction of communication and 7 upper bits is address. Therefore the number of addresses on I^2C bus is limited to range $0 \div 127$.

The direction of communication is defined by $R/\overline{W}$ bit in a following way. If $R/\overline{W} = 0$, then the master will write following bytes to selected slave. If $R/\overline{W} = 1$, then the master expects data from an addressed slave.

2.3.1.8 Address Verification

As was described above, the first byte after START is the address of selected IC and every byte transmission on I^2C is followed by acknowledgement. According this principle an addressed IC must answer the positive ACK (=0) when it recognizes its own address. If the master receives NACK (=1), then on the I^2C bus is not any IC with a given address.

The example of addressing the circuit PCF8574 is depicted in figure 23. The IC PCF8574 will be in more details described below. The part of PCF8574 address, marked in figure as $A_0 \div A_2$, is configurable.

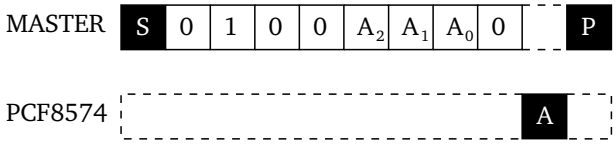

Figure 23: Example of addressing of PCF8574

2.3.2 I^2C Programming Interface

The archive `apps-labexc.zip` for laboratory exercises contains programming interface for I^2C bus. It implements following set of functions:

`void i2c_init()` - initialization of I^2C bus. Signals SDA and SCL are set to 1.

`void i2c_start()` - START signal.

`void i2c_stop()` - STOP signal.

`void i2c_ack()` - send positive ACK.

`void i2c_nack()` - send negative NACK.

`unsigned char i2c_getack()` - read ACK.

`uint8_t i2c_output( uint8_t t_value )` - send one byte and get acknowledgement.

`uint8_t i2c_input()` - receive one byte without acknowledgement.

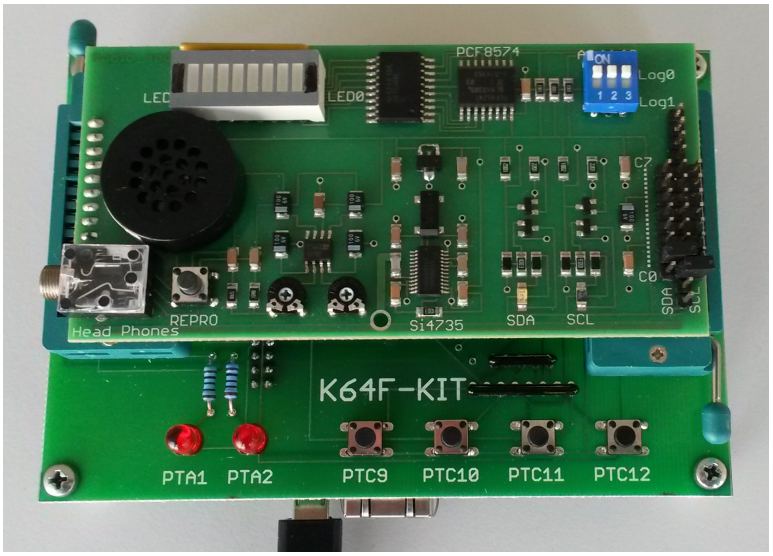

Figure 24: K64F-KIT with radio module and PCF8574 expander

2.3.3 FM/AM Radio Module

A FM/AM radio module is prepared for I^2C bus laboratory exercise. This module is depicted in figure 24. It contains two IC connected to I^2C bus. The first IC is the simplest IC for I^2C – 8-bit expander PCF8574 and the second IC is the FM/AM radio module Si4735. This radio module is usually used in mobile phones.

2.3.4 PCF8574 – 8-bit Expander

The inseparable part of this documentation is datasheet `PCF8574.pdf`.

The 8-bit expander PCF8574 was selected for radio module with a concrete goal. It is the simplest IC available on the market for I2C bus and it is a good opportunity to get the first experiences with I^2C programming in very easy way. This IC do not have any configuration registers and it contains only one 8-bit data register whose bits are directly lead out to its package. The value of this register can be changed by a single writing or its value can be also obtained by a single reading.

The scheme of PCF8574 with connected LEDs is visible in figure 25

2.3.4.1 PCF8574 Addressing

The scheme visible in figure 25 depict PCF8574 and its three address pins $A_0 \div A_2$ allow configure I^2C address of this IC. These three address pins are

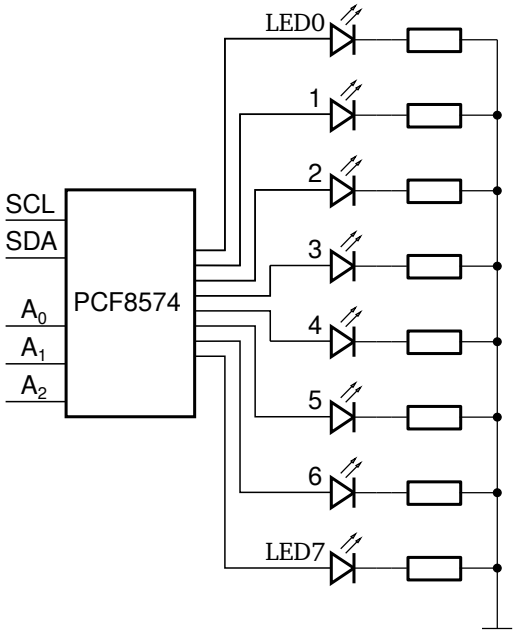

Figure 25: PCF8574 with connected LEDs

connected to switches and programmer must use them to set a concrete address. The scheme of switches connection is shown in figure 26.

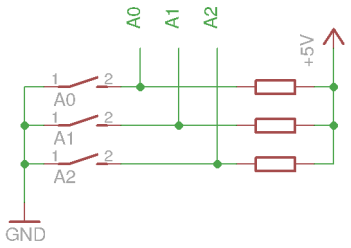

Figure 26: PCF8574 address settings

The address of PCF8574 is described in datasheet at page 9. The address is composed from three parts and these parts are visible in figure 27.

The highest four bits $H_0 \div H_3$ are hardwired by manufacturer and according to the datasheet they are set to value 0b0100 (0x4). The middle part of address $A_0 \div A_2$ is set by switches. The LSB bit $R\overline{W}$ defines direction of communication (described earlier).

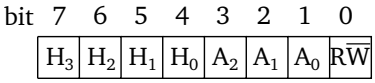

Figure 27: Three parts of PCF8574 address

2.3.4.2 PCF8574 Communication

The communication with PCF8574 is described in details in datasheet at page 10 and 11.

The simplified diagram of communication is visible in figure 28. It is example of one byte writing.

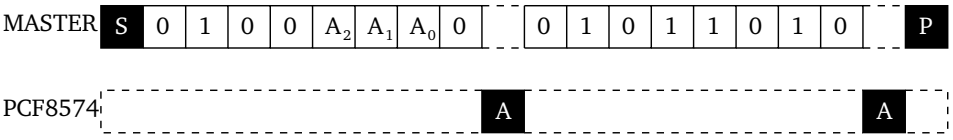

Figure 28: Writing one byte to PCF8574

2.3.5 Si4735 – FM/AM Radio-module

Inseparable parts of this text are manufacturer programmers guide for Si4735 `SI4735-PG.pdf` and text of European standard EN50067, the specification of Radio Data System (RDS).

The IC Si4735 contains all necessary parts for receiving FM (Frequency Modulation) and AM (Amplitude Modulation) broadcasting. This chip is used in many mobile phones as a miniature radio receiver. In laboratory exercise only the FM broadcasting will be received. The scheme of radio module wiring is visible in figure 29.

The voice output is after radio station tuning directed to headphone jack or it is possible to redirect the voice to the speaker by pressing button.

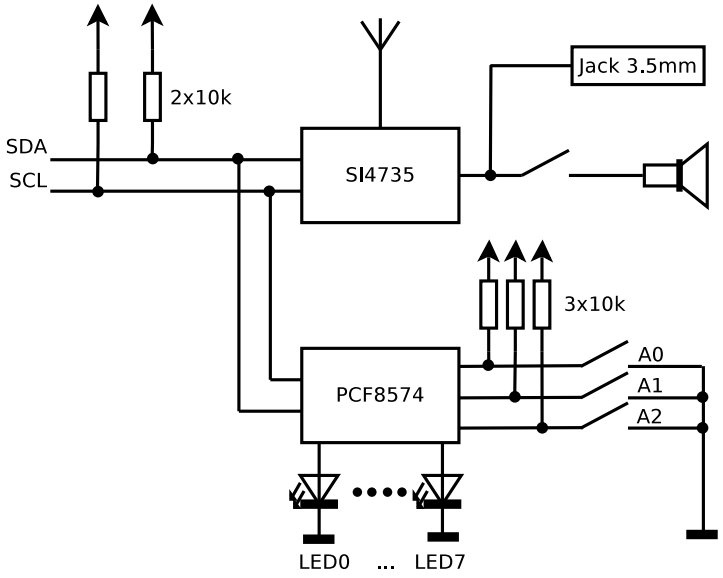

Figure 29: Scheme of Si4735 wiring

2.3.5.1 Si4735 Control

The complete documentation of Si4735 control is described in Programming Guide from producer.

The address of Si4735 on the I^2C bus is 0x22. This address can't be changed.

2.3.5.2 Volume Control

The volume of a voice output must be set using a command SET_PROPERTY 0x12. This command must be followed by the argument 0x00 and then by two-byte subcommand RX_VOLUME 0x4000. Next byte is 0x00 and the last byte is a level of volume in range 0÷63. More information is in the programming documentation on page 65 and 117.

Command

0	1	2	3	4	5
0x12	0x00	0x40	0x00	0x00	0b0VVVVVVV*

∗Bits V specify volume level

2.3.5.3 Tuning of Radio Station

The radio station can be tuned in two ways. The first possibility is the direct tuning of a known frequency using a command FM_TUNE_FREQ 0x20. This command has four byte arguments. The first and the last byte can be 0x00. The second and the third byte is the required frequency in MHz multiplied by 100.

More information about this command is in the programming documentation on page 67 and 68.

Command

0	1	2	3	4
0x20	0x00	FreqHI	FreqLO	0x00*

∗More information about these bits is in the documentation

The second possible way to tune a station is the automatic tuning using command FM_SEEK_START 0x21. This command requires only one argument whose second bit set to 1 defines automatic wrap of seeking when the end of frequency range is achieved. The third bit specify direction of seeking up or down.

The response to this command contains only one byte. More information about this command is in the programming documentation on page 69.

Command

0	2
0x21	0b0000SW00*

∗ S - Seek up/down 0/1
W - Wrap seeking on the end of frequency range

2.3.5.4 Status of receiver

The command FM_TUNE_STATUS 0x22 has only one byte argument which can be set to 0x00. Response to this command contains eight bytes. The second and third byte is currently tuned frequency. The fourth byte is strength of signal and the fifth byte is a quality of signal.

More information about this command is in the programming documentation on page 70 and 71.

Command

0	1
0x22	0x00*

∗More information in documentation

Response

0	1	2	3	4	5	6	7
S1*	S2*	FreqHI	FreqLO	RSSI	SNR	MULT*	CAP*

∗More information in documentation

2.3.5.5 Signal Quality

The command FM_RSQ_STATUS 0x23 has only one byte argument which can be set to 0x00. Response to this command contains eight bytes. The LSB bit of second byte inform whether some radio station was recognized. The fourth byte is strength of signal and the fifth byte is a quality of signal. Ve čtvrtém bajtu je hodnota síly signálu a v pátém bajtu kvalita signálu.

More information about this command is in the programming documentation on page 72 and 73.

Command

0	1
0x23	0x00*

∗More information in documentation

Response

0	1	2	3	4	5	6	7
S1*	S2*	0bxxxxxxxV*	STBL*	RSSI	SNR	MULT*	FREQ*

∗More information in documentation

2.3.5.6 Radio Data System (RDS)

The RDS is a system for embedding text information to FM radio broadcasting. The transmitted information can be e.g. a name of radio station, name of currently broadcasted program, alternative frequency of current station or information about traffic accidents in surroundings.

RDS data can be obtained from Si4735 using command FM_RDS_STATUS 0x24. This command has only one byte argument where the LSB bit enables an internal RDS interrupt to activate RDS receiving.

The response to FM_RDS_STATUS command contains 13 bytes.

The LSB bit of the second byte informs whether RDS information are synchronized. If this bit is not set, then the rest of response is invalid.

The most important data for following processing are marked as BxHI and BxLO, where x is 1÷4. They are block of data transmitted in one group.

More information about this command is in the programming documentation on pages 74-76.

Dotaz

0	1
0x24	0b00000SMI*

∗ bit I switch on internal interrupt for RDS receiving,
bits S and M are described in documentation

Response

0	1	2	3	4	5	
RDS0*	RDS1*	0bxxxxxxxS*	RDS3*	B1HI	B1LO	
6	7	8	9	10	11	12
B2HI	B2LO	B3HI	B3LO	B4HI	B4LO	0baabbccdd**

∗More information in documentation
∗∗ Bits aa, bb, cc a dd define state of all blocks

Data are transmitted in RDS in groups. Every group is split to four blocks, see figure 30. Content of all blocks are defined by standard EN50067. Bits Checkword are processed directly by Si4735 and they are not included in response to command 0x24.

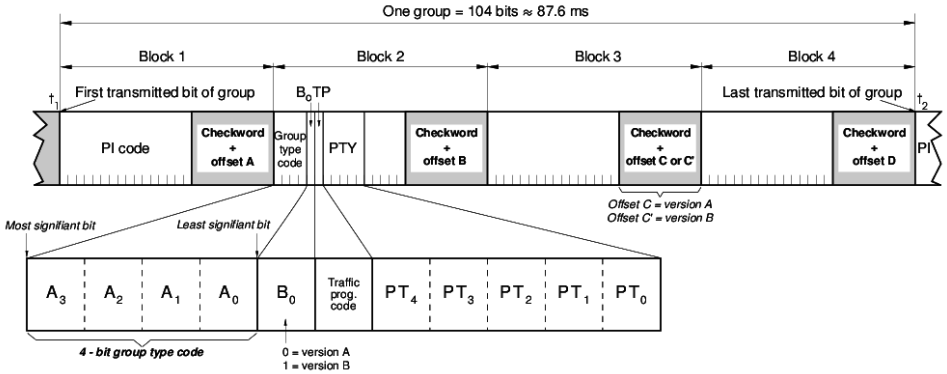

Figure 30: General form of RDS data

In figure 30 it is visible that the first block contains information marked as PI - Program Identification. It is unique number for every radio station. Bits $A_0 \div A_3$ in the second block specify a group number. The bit B_0 defines group version A or B. The bit TP indicate traffic reports. Bits $PT_0 \div PT_4$ specify type of broadcasted program (speech, music, etc.). The meaning of other bits depends on a group number.

2.3.5.7 Program Service Decoding

The Program Service (name of radio station) are transmitted in group 0 in version A and B. The text in the both versions can be different.

The structure of bits in group 0A and 0B is depicted in figure 31 and 32.

The name of radio station is composed from 8 characters, but only one pair of characters is in one response. Thus the name of station is split to four character pairs8. This character pair is in block 4. The position of this character pair in station name is specified by two lowest bits of block 2.

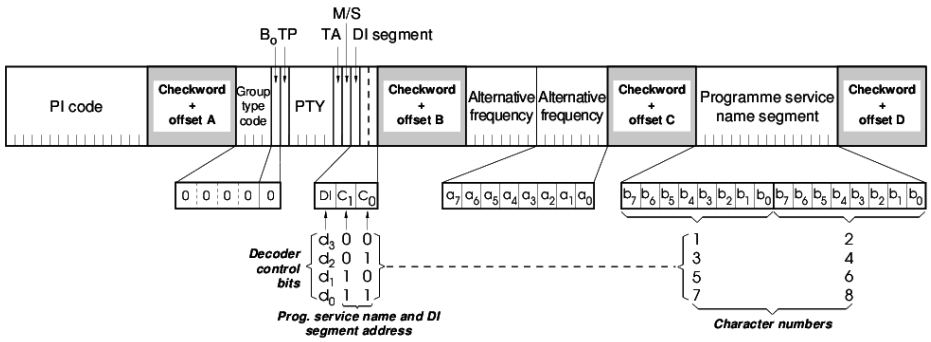

Figure 31: Group 0A

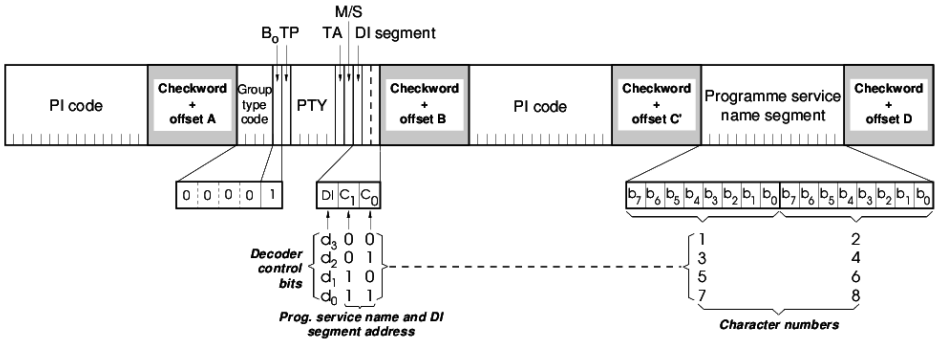

Figure 32: Group 0B

2.3.5.8 Alternative Frequency Decoding

The alternative frequency is transmitted at part of Program Service in version A, block 3. The position of this frequency is also visible in figure 31. The value 0b00000001 corresponds to frequency 87.6 MHz and value 0b00110011 corresponds to frequency 107.9 MHz.

2.3.5.9 Radiotext Decoding

Radiotext service is transmitted in similar way as Program Service. It is composed from 64 characters split to character quartets. These characters are in block 3 and 4 and they are transmitted only in the group 2A. The lowest four bits in group 2 specify position of every character quartet. The format of group 2A is for clarity depicted in figure 33.

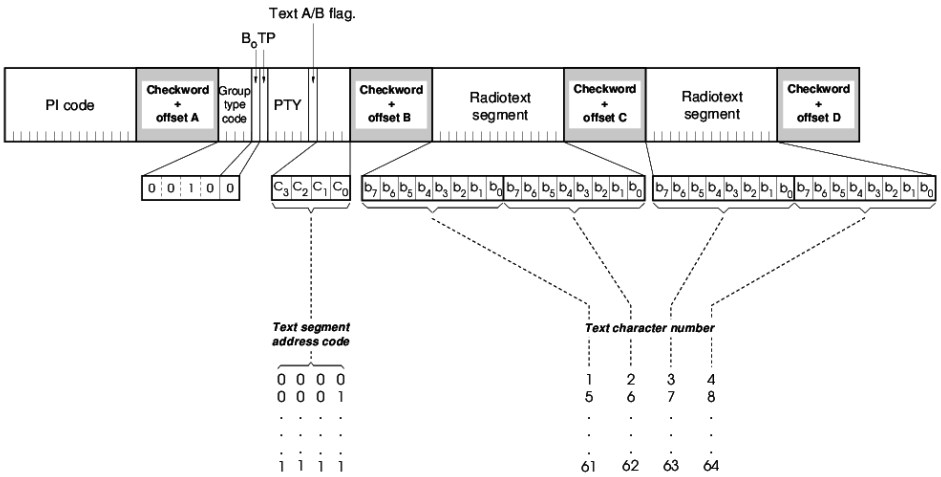

Figure 33: Skupina 2A

2.3.5.10 Task Examples

- Study all datasheets.
- Set address for PCF8574, read and write data, implement light bar.
- Tune concrete radio station.
- Using two buttons set volume level.
- Using two buttons change tuned frequency.
- Using light bar display level of signal.
- Using LED detect stereo signal.
- Display name of station (Program Service).
- Display Radiotext.

Part II

Parallel Systems Programming

Chapter 3

Threads

3.1 Motivation for Utilization of Multi-Core CPU Computing Power

During past years producers of CPUs started to increase the CPU performance by multi-core architecture. Computing power is increasing as many times as a number of processor cores. But the problem is with software which lags behind the advanced level of computers. The programs and operating systems are behind computers many years. It turned out during the last 30 years several times. The example is 64 bit architecture. When AMD introduced the first 64 bit CPU? When 64 bit operating system startet do be common on desktops? The answer can be easily found on the internet.

The current problem is development of programs which are able to use more processors and their cores parallely. The professor David Patterson from department of computer science at University of California published in 2010 article "The Trouble with Multicore". In this article is one very interesting paragraph:

"*One of the biggest factors, though, is the degree of motivation. In the past, programmers could just wait for transistors to get smaller and faster, allowing microprocessors to become more powerful. So programs would run faster without any new programming effort, which was a big disincentive to anyone tempted to pioneer ways to write parallel code. The La-Z-Boy era of program performance is now officially over, so programmers who care about performance must get up off their recliners and start making their programs parallel.*"

So if we want to our programs work faster, there is no other way than program them parallely.

3.2 Programing with Threads

Threads are parallely and independently running subroutines (functions) inside process. This technology is implemented in Unix and Windows systems more than 20 years. It is very usefull and it help to program more comfortable regardless of number of CPUs and cores. However the higher computing performance can be utilized only when computer has two or more cores or CPUs.

How to use threads in own program in Linux will be described in following text. The first step is to program function that will represent a thread. This function must have precisely defined format:

```
void *thread_function_name( void *argument ) { /* code */ }
```

The argument can be used to pass to thread any data type. When it is necessary to pass more argument then they have to be encapsulated to array, structure or class.

When the function for thread is programed, then it is possible to create thread from it. In the Linux is for this purpose designed function:

```
int pthread_create( pthread_t *thread,
                    const pthread_attr_t *attr,
                    void *(*start_routine) (void *),
                    void *arg );
```

The first parameter `thread` is pointer to thread ID. This ID can be used later e.g. for synchronization of threads. The second parameter `attr` allows to define some parameters of new thread. For normal usage can be used as NULL pointer. The third parameter `start_routine` is a function which will be run as thread. The last parameter `arg` is an argument for new thread. It is used to pass data to new thread.

Waiting for the completion of thread must be performed by function:

```
int pthread_join(pthread_t thread, void **retval);
```

The first parameter `thread` is ID of existing (running) thread and the second parameter `retval` can be used to get return value of thread.

More information about thread programming is available in manual pages, see `man pthreads`, `man pthread_create` and `man pthread_join`, or the internet.

The example of program which creates a thread and then it waits until its completion is in following code:

```
void *simple_thread( void *str )
{
    printf( "Thread created with argument '%s'.\n", ( char * ) str );
    sleep( 5 );
    return NULL;
}

int main()
{
    pthread_t tid;
    pthread_create( &tid, NULL, simple_thread, ( void * ) "Testing" );
    pthread_join( tid, NULL );
    printf( "Done\n" );
}
```

Another example of thread usage is in archive threads-demo.zip, that is part of this study material. This archive contains example where the maximum number is searched in array using two threads. The both threads search only in half of array and the resulting time of search is half of time required by single thread search. More information are in comments in source code.

3.2.0.1 Preparation for Laboratory Exercises

- Study this text and demo program.
- Repeat basic sorting algorithms – Insertion, Selection and Bubble sort.
- Prepare source code of algorithms to be able sort part of array in ascending and descending order.
- Prepare source code to be able sort any type of numbers (int, float, double, etc.).
- Think and test generation of random numbers within specified range.
- Test programming with threads.

3.2.0.2 Task Examples

- Generate random numbers into arrays using more threads.

- Sort selected part of array and verify the result.
- Merge two sorted array.
- Sort array parallely.
- Measure sort time.

Chapter 4

CUDA

Study materials and examples are in separate files.