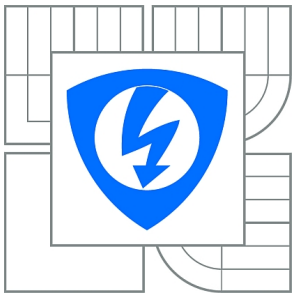
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV MIKROELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF MICROELECTRONICS

ELECTRONIC INFORMATION CARD

ELEKTRONICKÝ INFORMAČNÍ ŠTÍTEK

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. VIKTOR ŠAFÁŘ

AUTHOR

VEDOUCÍ PRÁCE — Ing. JOSEF ŠANDERA, Ph.D.

SUPERVISOR

BRNO 2011

BRNO UNIVERSITY
OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

Department of Microelectronics

Diploma thesis

master's study field
Microelectronics

Student: Bc. Viktor Šafář
Year of study: 2

ID: 78337
Academic year: 2010/11

TITLE OF THESIS:

Electronic information card

INSTRUCTION:

Introduce with principles, parameters and design of electronic information tags in the world. Design and fabricate a prototype of an electronic name tag using LED display and PIC microcontroller. Create a program to control the device via computer. Examine the possibility of the device to be powered by batteries and select a suitable battery.

REFERENCE:

According of supervision instructions

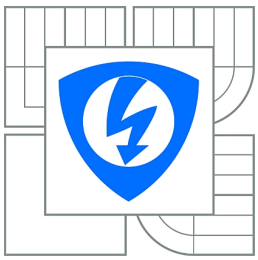
Assigment deadline: 7.2.2011

Submission deadline:26.5.2011

Head of thesis: Ing. Josef Šandera, Ph.D.
Consultant:

prof. Ing. Vladislav Musil, CSc.
Subject Council chairman

WARNING:



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav mikroelektroniky

Diplomová práce

magisterský navazující studijní obor
Mikroelektronika

Student:	Bc. Viktor Šafář	***ID:***	78337
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Elektronický informační štítek

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s principy, parametry a provedením elektronických informačních štítků ve světě. Navrhněte a realizujte vzorek elektronického informačního štítku. Pro konstrukci použijte hotový aktivní displej Z LED diod a mikrokontroler PIC. Sestavte obslužný program pro komunikaci s PC. Prověřte možnosti napájení z baterií a vyberte vhodný typ baterie.

DOPORUČENÁ LITERATURA:

According of supervision instructions

Termín zadání:	7.2.2011	***Termín odevzdání:***	26.5.2011

Vedoucí práce: Ing. Josef Šandera, Ph.D.

prof. Ing. Vladislav Musil, CSc.
Předseda oborové rady

Abstract

In this project, an electronic information card based on an LED matrix display is designed. The theoretical part shows an overview of microcontroller peripherals and interfaces and character set discussion is held. It is shown how to design such a device taking one step at a time. First, the device main features are outlined and its functionality is described. Then, component selection is done taking into consideration the requirements and an electronic schema is designed. The heart of the device is a Microchip MCU for which a piece of software is designed. Consequently, a Windows application is build to operate the device.

Key words

LED display, Microchip USB, PIC18F46J50, SPI EEPROM, encoding of Czech characters, ISO 8859-2, display character set 5×7, USB device, WinUSB

Bibliografická citace díla:

ŠAFÁŘ, V. *Elektronický informační štítek.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 53 s., 5 příloh. Vedoucí diplomové práce Ing. Josef Šandera, Ph.D..

Declaration:

I declare that I have elaborated my master's thesis on the theme of "Electronic information card" independently, under the supervision of the master's thesis supervisor and with the use of technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the master's thesis I furthermore declare that, concerning the creation of this master's thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal copyright and I am fully aware of the consequences in the case of breaking Regulation S 11 and the following of the Copyright Act No 121/2000 Vol., including the possible consequences of criminal law resulted from Regulation S 152 of Criminal Act No 140/1961 Vol.

Brno, 26/05/2011

…….

Acknowledgement:

I am thankful to my university supervisor Ing. Josef Šandera, Ph.D for his continuous support during the work on this thesis.

Contents

1 Introduction

Electronic devices presenting information in text form are widely used all over the world. Ranging from large colorful screens alongside city roads to tiny one-character displays, these devices are used on different kinds of places for advertising, time-telling, latest information update and many more. The technologies used in them vary with respect to the device working environment and the desired performance.

The goal of this thesis is to have a look at current market solutions dealing with electronic information cards, an original design of one and a practical implementation. Our main concern will be given to devices with active displays. The implementation will be based on an existing LED matrix display designed by Department of microelectronics, FEEC, BUT and a Microchip PIC microcontroller.

Usage of such device is intended on conferences and similar events, either as a general name tag or a gift from the host. This sets out two basic parameters of the device: low cost and light weighted. The device is expected to be paid for by the event sponsor so part of the device is to be designed so that it can carry the sponsor's advertisement.

The paper covers theoretical background regarding current market possibilities, electrical device design, assorted peripheral overview and character set topic. Then, the implementation is undertaken one step at a time covering the device definition, component selection and circuit design. The next steps include designing the firmware of the device and an application for PC. At the end, a user guide is available.

2 Theoretical background

Nowadays, the easiest way to find a product on the market is to look it up using a web search engine on the Internet. By searching the term “electronic information card”, “electronic name badge” or “electronic nametag” it can quickly determined that most of the currently sold name tags come from various Chinese manufacturers and the prices range from 15 USD to 40 USD per piece.

Figure 1: Electronic name badge by PromoTagZ [1]

Performance of these devices is very much determined by their price. The cheapest devices can only hold one character string [2] of 120 characters while other devices implement microphone, GPS and radio communication module [3], [4]. Table 1 shows some of current possibilities when looking for an electronic information card. It is clear that features of such a device can vary from very few to many making the device quite complex.

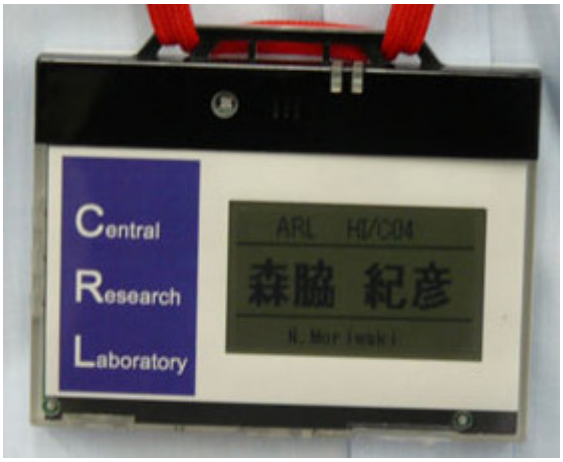


Figure 2: Electronic name badge for Hitachi employees [4]

Table 1: Some of current market possibilities regarding electronic information cards

Manufacturer – Model	**Features**
PromoTagZ – LED Badge [1]	• 8 message storage capacity (250 characters per message) • 4 Button On-Board Message programming • 168 LEDs • Weight cca 30 grams • Size: 8 cm × 4,2 cm × 6 mm • Display size: 7.3 cm × 1.2 cm • CR 2032 battery, lasts up to 45 hours
Unknown – AC-263 [2]	• 1 message of 120 characters • Weight: 33 g without battery • Size: 79 × 42 × 7 mm • Display size: 73 × 15 mm • 147 LED • 4 levels of speed to display • Data communication: input by hand • One CR2032 or two CR2016 battery • Operation time: about 20 hours
Hitachi - AirLocation Tag-w [3,4]	• Wi-Fi • RFID • emergency message function • Price: $178 • Designed for tracking employee movement within corporate area

2.1 Designing a device

When designing an electrical device, follow a set of rules and ways should be followed [5]. It begins with the device definition which means defining basic function of the device, its parameters and operation. Then there is design including component selection, theoretical design on block level and its verification (calculations and computer simulation), circuit level design, printed circuit board, mechanical parts and overall electrical and mechanical design.

The next steps include manufacturing PCBs and mechanical parts, soldering, first power-up, software design and programming (if needed), testing and completion. Once the device prototype is functional and corresponding with client's requests, the process moves to creating device documentation and release design for mass production. In this report we will go through all of these steps with an exception to the last one.

2.2 Displays

There are two basic types of displays – active and passive. As for active displays, the visual information being presented is provided by *generation* of light while passive displays provide us with information presented by *modulation* of light. Various types of displays are based on various physical effects.

LED displays, as they consist of a number of LEDs, are based on the electroluminescence effect. That means that if a forward voltage is applied on the diode generating forward current, electrons and holes entering the area of the p-n junction on different energy levels will recombine and release a photon. This photon has energy proportional to the energy difference and it varies with material used. One of the parameters of an LED is its color which is defined by the energy of the photon. Nowadays, it is possible to manufacture LEDs of almost any color in the visible part of electromagnetic spectrum and beyond (particularly in the infrared spectrum which is used in optical communications). When put together into a matrix, LEDs can prove as a nice, simple and highly performing display that is easy to drive.

Information on such a display can be presented and updated in series piece by piece, not all at once. Well known example of this behavior is a CRT monitor. This technology utilizes a human eye persistence of vision which is the phenomenon of the eye by which an afterimage is thought to persist for approximately one twenty-fifth of a second on the retina [6]. CRTs may sometimes be seen to flicker, often in a brightly lit room, and at close viewing distances. This effect is due to the greater likelihood that part of the screen will occupy the viewer's peripheral vision, where sensitivity to flickering is greater. Generally, a refresh rate of 85 Hz or above (as found in most modern monitors) is sufficient to minimize flicker at close viewing distances, and all recent computer monitors are capable of at least that rate.

2.2.1 LED matrix display

The display used in this project is an LED matrix display consisting of 8 rows and 32 columns resulting in a total of 256 LEDs. There are several aspects to driving a matrix display. If the persistence of vision is utilized it is possible to turn on one row or one column at a time which for example gives us the possibility to adjust the brightness of the display in software by adjusting the frequency of the switching.

The easiest way to drive a matrix display is to drive both columns and rows directly from the microcontroller. This approach requires significant amount of wires to interconnect the microcontroller with the display but it is the easiest one to design software for.

Another approach is to use some kind of multiplex, either for driving columns or rows or both of them. This way, the amount of interconnections is smaller but the complexity of the software is higher. To drive the matrix display, four 8-channel multiplexers (16-or-more-channel multiplexers have too many outputs that cancel out the use of multiplexer to simplify the circuit) for the columns can be used. If the microcontroller can access the information on how to create desired image on the display, then the information passed into these multiplexers would have to be encoded which increases the complexity of the software. A whole workaround would have to be created to work this way.

Another way to drive a matrix display is by using serial-to-parallel shift registers. If four 8-bit registers connected in series, a 32-bit information and a clock signal has to be supplier that will move this information trough the registers. This way, the 32 columns can be driven quite easily.

There is also a technique called Charlieplexing [7] (proposed by Charlie Allen in 1955) which pulls the number of interconnections of a direct driving to absolute minimum (see figure 3). Driving a higher number of LEDs requires complementary drive and a tri-state logic I/Os. The idea is that if LED1 wants to be turned on, Pin 1 needs to be set to H, Pin 2 to L and Pin 3 to Z (high impedance). To light up another LED, statuses of at least two of these pins need to be changed. This becomes even more complicated if larger amount of LEDs in place thus becoming a useless principle.

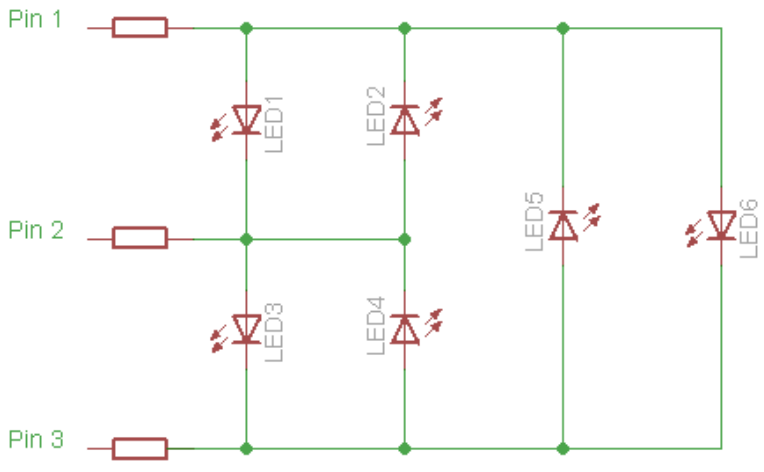


Figure 3: Charlieplexing 6 LEDs

2.3 Microcontroller unit (MCU)

An MCU is a small computer on a single integrated circuit containing a processor core, memory and programmable input/output peripherals. Program memory in some form is often included on the chip, as well as a typically small amount of RAM. In this project, we will work with an 8-bit MCU by Microchip. For its operation, MCU requires a clock signal that can be brought from outer circuitry or from an internal oscillator.

Each MCU has its own instruction set that can be found in the device's datasheet. An instruction is a built-in functionality of an MCU and by combining instructions, programs are created. It is quite difficult to design large and complex programs in this kind of code so another way has been invented. The program can be created in a higher level programming language (such as C) and then compiled into the lower form by using a suitable compiler (more on this topic in section 1.7 Development tools).

2.3.1 Program

Usually, there are two phases when running a microcontroller – a setup phase and a continuous executive phase. In the setup phase, MCU is setup by the program to correctly interface with the surrounding circuitry and its peripherals are brought to required state. In the on going executive phase, program is usually "stuck" in a never ending loop. In this loop, the program is doing what it has been designed to do (driving a display, managing buttons, communicating with other devices, etc.).

There are asynchronous events that happen rarely or occasionally (such as communication inquiry from another device or in some cases a press of a button). These events may not be caught up in the program main loop. Instead, MCUs are designed in a way that if such an event occurs, the execution of the main loop is halted and the event is brought to attention. This is called an *interrupt*. Every MCU family handles interrupts a little bit differently but the main idea is as follows. When an interrupt occurs, the current state of execution is saved and begins execution of an interrupt handler. Interrupt handler is a routine (also called Interrupt Service Routine or ISR) that takes care of whatever aspect caused the interrupt to happen. When this routine finishes, the program loads the state of execution and continues. From the logic of it, it is quite clear, that ISR can not execute time-consuming pieces of code. A good way to execute a longer piece of code is to setup a flag in the ISR and pick this flag up in the main loop after the ISR is over and the main loop continues.

2.3.2 Peripherals

There are several key peripherals implemented in every MCU. These may include timers, communication interfaces (USB, UART, SPI, I2C, Ethernet), analog-to-digital converters, pulse width modulators, real-time clock, analog comparators and others.

Timers are quite important peripherals. There are 8-bit and16-bit timers available in Microchip 8-bit microcontrollers. Timer is a register that holds a value which is incremented on raising edge of the system clock signal (that is why they are also called counters). Some timers have a prescaler that scales down the timer's input clock. A prescaler is basically another counter that is connected between the system clock and the timer and it generates output impulse when overflowed.

To communicate with other devices such as a PC, there are several communication interfaces implemented. One way to create a physical communication layer between a PC and an MCU is to use UART which is a hardware implementation of RS-232 standard (or so-called Serial port on a PC). Since Serial port is no longer a common interface on a PC, it is possible to use an RS-232 to USB converter, such as FT232. But this involves implementing another IC into the circuit which is not cheap. Microchip implements USB interface directly into some of their MCUs so advantage is taken of this feature.

SPI stands for Serial Peripheral Interface and it is a synchronous serial data link standard named by Motorola that operates in full duplex mode [8, 9]. Devices communicate in master/slave mode where the master device initiates the data transfer. Multiple slave devices are allowed with individual slave select (chip select) lines. SPI can be used in conjunction with SPI enabled memories or other serial devices.

2.4 Memory

To store user data in an electronic device, a memory chip has to be implemented. Most of the available MCUs have a built-in internal memory but this memory is not of a big capacity. In our device, the request stands at storing at least 200 strings of at least 50 characters. The actual size of these data is dependent on encoding but it can assumed that one character will take at least 1 byte of memory resulting in a total of 10 000 bytes. MCU internal memories are not this big. Thus an external memory has to be provided.

There are several types of memory for storing user data in an MCU based device. The most common one is an EEPROM (Electronically Erasable Read Only Memory) which is a

type of memory that allows its entire contents to be electrically erased and then rewritten electrically, so that it needs not to be removed from the device. It is a non-volatile memory, which means that it can hold data when power is not connected. That is a desirable behavior. Microchip manufactures an SPI enabled EEPROM which is the one that will be used.

2.5 Universal Serial Bus

USB stands for Universal Serial Bus and it is probably the most common serial bus in the world of personal computers. It was designed and it is promoted and supported by USB Implementers Forum which consists of some of the world leading companies (Apple, Microsoft, Intel, etc.) [10]. Current highest version is 3.0 but it is not too spread yet since the specification was released on November 17 2008 and the first USB 3.0 certified consumer products hit the market in January 2010. Table 2 shows USB version and their specifications.

Table 2: Main versions of USB and some of their specifications (for full specification visit http://usb.org)

USB 1.0/1.1	• Version 1.0 released in January 1996
	• data rates of 1.5 Mbps (Low-bandwidth) and 12 Mbps (Full-bandwidth)
	• version 1.1 released in September 1998
	  o fixed top problems in 1.0
	  o the earliest version to be widely adopted
	• "Low-speed" since the release of 2.0

USB 2.0	• released in April 2000
	• "High-speed" bandwidth of 480 Mbps
	• Backward compatible with 1.1
	• Several connectors
	  o Plug A, plug B
	  o Mini-A, Mini-B (October 2000)
	  o Micro-USB (April 2007)

USB 3.0	• Released in November 2008 • "Super-speed" bandwidth of 5.0 Gbps • Different connectors but compatible (older plug can be put into 3.0 receptacle)

2.5.1 USB 2.0 Overview

USB is designed to establish communication between a host (usually PC, root hub) and a device or devices (peripherals). The role of root system software is to provide a uniform view of IO system for all applications software. It hides hardware implementation details so that application software is more portable. The USB IO subsystem manages the dynamic attach and detach of peripherals. This phase, called enumeration, involves communicating

with the peripheral to discover the identity of a device driver that it should load, if not already loaded. The device holds this information in so-called descriptors. A unique address is assigned to each peripheral during enumeration to be used for run-time data transfers. During run-time the host PC initiates transactions to specific peripherals, and each peripheral accepts its transactions and responds accordingly. Additionally the host PC software incorporates the peripheral into the system power management scheme and can manage overall system power without user interaction [11].

Besides the obvious role of providing additional connectivity for USB peripherals, a hub provides managed power to attached peripherals. It recognizes dynamic attachment of a peripheral and provides at least 0.5 W of power per peripheral during initialization. Under control of the host PC software, the hub may provide more device power, up to a maximum of 2.5 W, for peripheral operation. A newly attached hub will be assigned its unique address, and hubs may be cascaded up to seven levels deep (including the root hub). During run-time a hub operates as a bi-directional repeater and will repeat USB signals as required on upstream (towards the host) and downstream (towards the device) cables. The hub also monitors these signals and handles transactions addressed to itself. All other transactions are repeated to attached devices. A 2.0 hub supports both 2.0 and 1.1 peripherals: 480 Mbps (high-speed), 12 Mbps (full-speed) and 1.5 Mbps (low-speed). [11]

All USB peripherals are slaves that obey a defined protocol. They must react to request transactions sent from the host PC. For example, the peripheral responds to control transactions that requests detailed information about the device and its configuration. The peripheral sends and receives data to/from the host using a standard USB data format. This standardized data movement to/from the PC host and interpretation by the peripheral gives USB its enormous flexibility with little PC host software changes. [11]

2.5.2 Communication

USB device communication is based on *pipes* (logical channels). A pipe is a connection from the host controller to a logical entity, found on a device, and named an *endpoint*. Because pipes correspond 1-to-1 to endpoints, the terms are sometimes used interchangeably. A USB device can have up to 32 endpoints: 16 into the host controller and 16 out of the host controller. The USB standard reserves one endpoint of each type, leaving a theoretical maximum of 30 for user use [11].

Endpoints are grouped into *interfaces* and each interface is associated with a single device function. An exception to this is endpoint zero, which is used for device configuration and which is not associated with any interface.

The USB architecture comprehends four basic types of data transfer [11]:

- Control Transfer: Used to configure a device at attach time and can be used for other device-specific purposes, including control of other pipes on the device.
- Bulk Data Transfer: Generated or consumed in relatively large quantities and has wide dynamic latitude in transmission constraints.
- Interrupt Data Transfer: Used for timely but reliable delivery of data, for example, characters or coordinates with human-perceptible echo or feedback response characteristics.
- Isochronous Data Transfer: Occupies a prenegotiated amount of USB bandwidth with a prenegotiated delivery latency. (Also called streaming real time transfers.)

A pipe supports only one of the types of transfers described above for any given device configuration.

2.5.3 USB device classes

Most USB devices have much in common with other devices that perform similar functions. All mice send information about mouse movements and button clicks. All drives transfer files. All printers receive data to print and send status information back to the host. When a group of devices or interfaces share many attributes or provide or request similar services, it makes sense to define the attributes and services in a class specification. The specification can serve as a guide for developers who design and program devices in the class and for programmers who write device drivers for host systems that communicate with the devices. Operating systems can provide drivers for common classes, eliminating the need for device vendors to provide drivers for devices in those classes [12].

2.5.4 WinUSB

WinUSB is a Windows generic driver for devices that do not fit into any defined USB class. The driver was introduced with Windows Vista and is also usable on Windows XP systems but not usable on earlier Windows editions [13]. It supports control, bulk, and

interrupt transfers. Its advantage is no need of knowledge how to write drivers on developer's part which significantly speeds up development.

Microchip has designed its C18 microcontrollers to work with the WinUSB driver as well as "Microchip USB stack" which is a ready-to-use pack of code files in C programming language.

2.6 Character set

ASCII is a character encoding scheme and it is the base for text communication. It is a 7-bit encoding resulting in 128 characters that include 33 non-printing control characters, 94 printable characters and a space [14], [15], [16]. Localized character sets are based on ASCII and they add another 128 characters to it evolving into 8-bit encoding.

Table 3: ISO 8859-2 character set in hexadecimal designation

	x0	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**	**xA**	**xB**	**xC**	**xD**	**xE**	**xF**
0x	NOT IN USE															
1x																
2x	SP *	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8x	NOT IN USE															
9x																
Ax	NBSP*	Ą	˘	Ł	¤	Ľ	Ś	§	¨	Š	Ş	Ť	Ź	SHY*	Ž	Ż
Bx	°	ą	˛	ł	´	ľ	ś	ˇ	¸	š	ş	ť	ź	˝	ž	ż
Cx	Ŕ	Á	Â	Ă	Ä	Ĺ	Ć	Ç	Č	É	Ę	Ë	Ě	Í	Î	Ď
Dx	Đ	Ń	Ň	Ó	Ô	Ő	Ö	×	Ř	Ů	Ú	Ű	Ü	Ý	Ţ	ß
Ex	ŕ	á	â	ă	ä	ĺ	ć	ç	č	é	ę	ë	ě	í	î	ď
Fx	đ	ń	ň	ó	ô	ő	ö	÷	ř	ů	ú	ű	ü	ý	ţ	˙

* SP – space, NBSP – non-breaking space, SHY – soft hyphen

To be able to process Czech diacritics a system has to use one of the encodings that support Czech characters. There are several of these and none of them is the official one. The chosen encoding has to be taken into consideration both in the device and in the PC application. There are vast discussions on the Internet as to what encoding should be used and why. The Internet Assigned Numbers Authority (entity that oversees various resource

allocations) prefers ISO 8859-2 encoding as an encoding for Eastern-European languages [15] so this is the one that will be used. As a result, each character will be encoded in one byte.

The space in table 3 between 0x81 and 0x9F is often used for various special characters that often appear in real life such as Greek letters, currency symbols, basic math characters, etc.

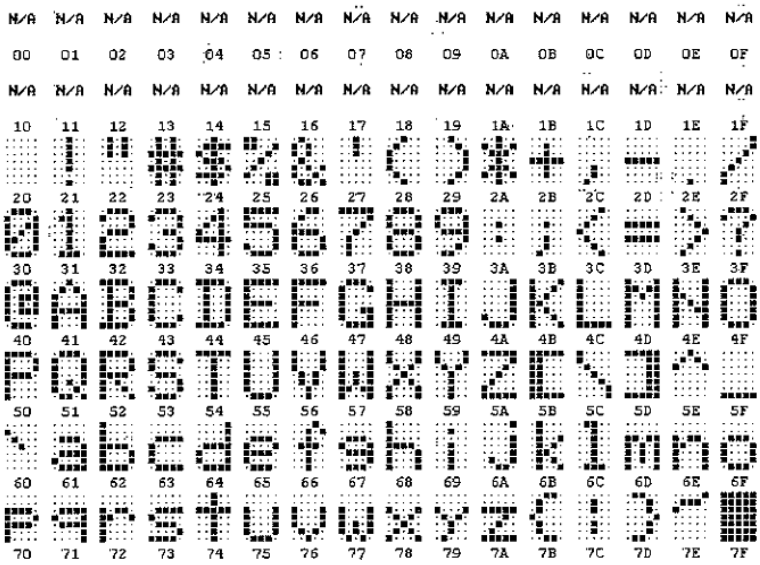


Figure 4: Standard ASCII character set [17]

Figure 5: Standard ISO 8859-2 character set [17]

The display size allows us to use standard character resolution of 5 × 7 dots per one character (5 dots wide and 7 dots high). Figure 4 shows standard ASCII character set and

figure 5 shows expansion to standard ISO 8859-2. The eighth row of the display can be used to improve the visual features of the font when displaying capital characters with diacritics. Porting the entire font to fit 8 rows for standard ASCII characters will result in undesirable character deformation.

2.7 Development tools

To program the microcontroller, the Microchip MPLAB ICD2 programmer/debugger will be used in conjunction with Microchip MPLAB IDE v8.43 (current version at the time of the beginning of this project) development environment. The program will be designed in C programming language and Microchip's MPLAB® C Compiler for PIC18 MCUs will be used to compile it. Figure 6 shows how to connect application board with the ICD2 programmer.

To design the circuit and printed circuit board, CadSoft Eagle 4.16 will be used. The university owns a license but it is also available as freeware for non-profit applications. The freeware version is called "Light Edition" and it only supports two signal layers and usable board area is limited to 100 × 80 mm which should be satisfying for the needs of this project.

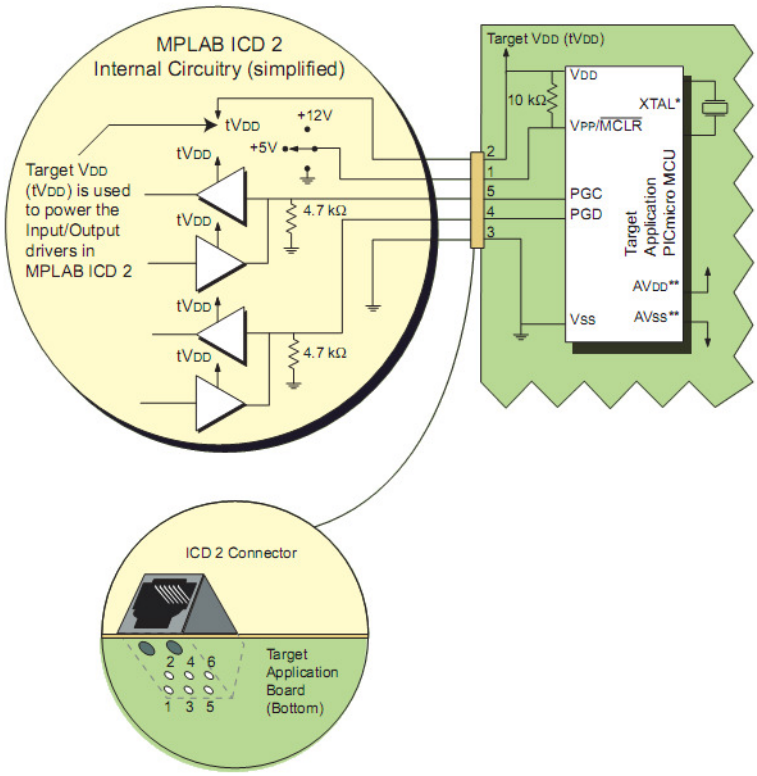


Figure 6: Connecting target application board to MPLAB ICD2 programmer/debugger [18]

To design a Windows program, Microsoft Visual Studio 2008 will be used which is available on the MSDNAA portal. The program will be designed as a .NET Winform application. .NET is a framework designed and supported by Microsoft for developing powerful Windows applications. The only aspect that could be thought of as a disadvantage is the need of .NET framework on user computer. But since .NET is included by default in Windows Vista and 7 operating systems, it is part of Windows XP service packs and it is widely used by various third party Windows applications, it is safe to build an application on it.

Nowadays, the selection of electronic devices is quite simplified by easy-to-use selection tools on the websites of device distributors [19], [20]. They can provide a very good top-level search results. The actual selection is, of course, based on a closer look at the device's properties.

3 Designing the device

3.1 Device definition

The device will be an electronic name tag thus having proportions as a regular paper name tag, meaning its size should be roughly 10 cm wide and 5 cm high. Its operation will be as such: a user will connect the device to a computer with Windows operating system via USB and uploads certain amount of data into the device by means of a piece of software that will be designed as well. The data will be consequently available to the user to display on the device.

There will be four buttons to operate the device placed in two pairs on each side of the device. The device will be powered by two button cell batteries. Since the weight of the batteries is quite unbalanced to the weight of the rest of the hardware, it needs to be kept in mind when designing the PCB. The front side of the PCB is to be without components other than display and buttons so that an advertisement of an event sponsor can be put here.

The device will be able to display selected text in various lengths of at least fifty characters and be able to store at least two hundred of these text strings reflecting usage as a nametag on events where the entire database of event participants will be uploaded to the device at once and each participant will select their name later on.

3.2 Component selection

The device will be an assembly of two boards – a board with LED matrix display and a main board with the rest of the circuitry. The LED matrix requires two sides of PCB to be used and if incorporated into the main board, there will not be enough room for other components or the size of the main board would have to be expanded which is not desirable. This way, the size of the main board is determined only by the circuitry itself. The LED board will be simply placed on top of the main board with an insulation layer in between and soldered on the edges.

The heart if the device will be a PIC microcontroller which will control the display and buttons and will communicate with a PC via USB. The device will be able to draw power from the USB when connected; otherwise it will be powered by two coin cell batteries. Figure 7 shows component schema of the device.

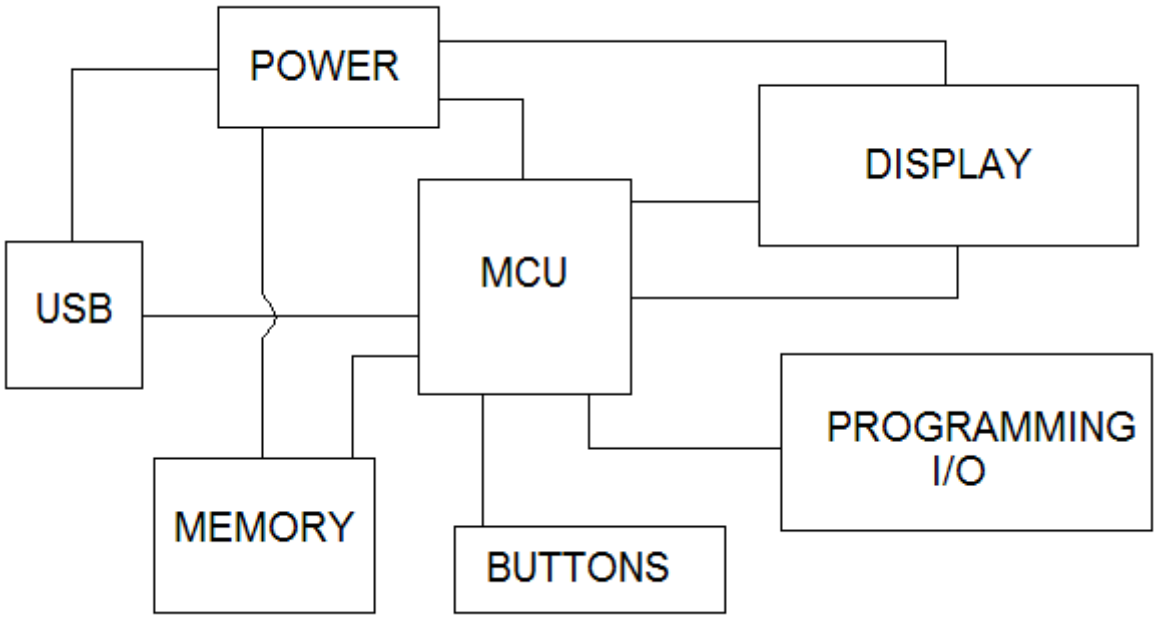


Figure 7: Component scheme of the device

3.3 Circuit design

For complete circuit design see appendix A.

3.3.1 Power block

Since the device is supposed to be portable, the only way to power the device is by batteries. There will be two coin cell batteries is parallel to provide longer working period. CR2032 is a standard lithium coin cell battery rated at 3.0 V with capacity ranging from up to 350 mAh. When the device is connected to PC via USB, it is possible to draw power from USB. Figure 8 shows how to prevent the device from drawing power from batteries when connected to USB.

Transistor Q9 is a P-channel MOSFET with ultra low on-resistance and high source current (IRLML6401) and IC2 is LDO regulator (MCP1825S). When USB is disconnected, regulator output node is on the same potential as GND and Q9 is switched on. Ultra low on-resistance means that the drain-to-source resistance is very low in this state ($R_{DS(on)} = 0.05\ \Omega$) thus creating virtually none voltage dropdown to V_{DD}. Continuous source current of the MOSFET is 1.3 A. When the device is connected, regulator output node is at $V_{LDO} = 3.3$ V and D2 forward voltage is $V_F = 0.3$ V, resulting in 3.0 V at V_{DD}. The regulator output is connected with the MOSFET gate and since potential of this node is higher than the one of

MOSFET source, the transistor will switch off thus preventing from drawing power from the batteries.

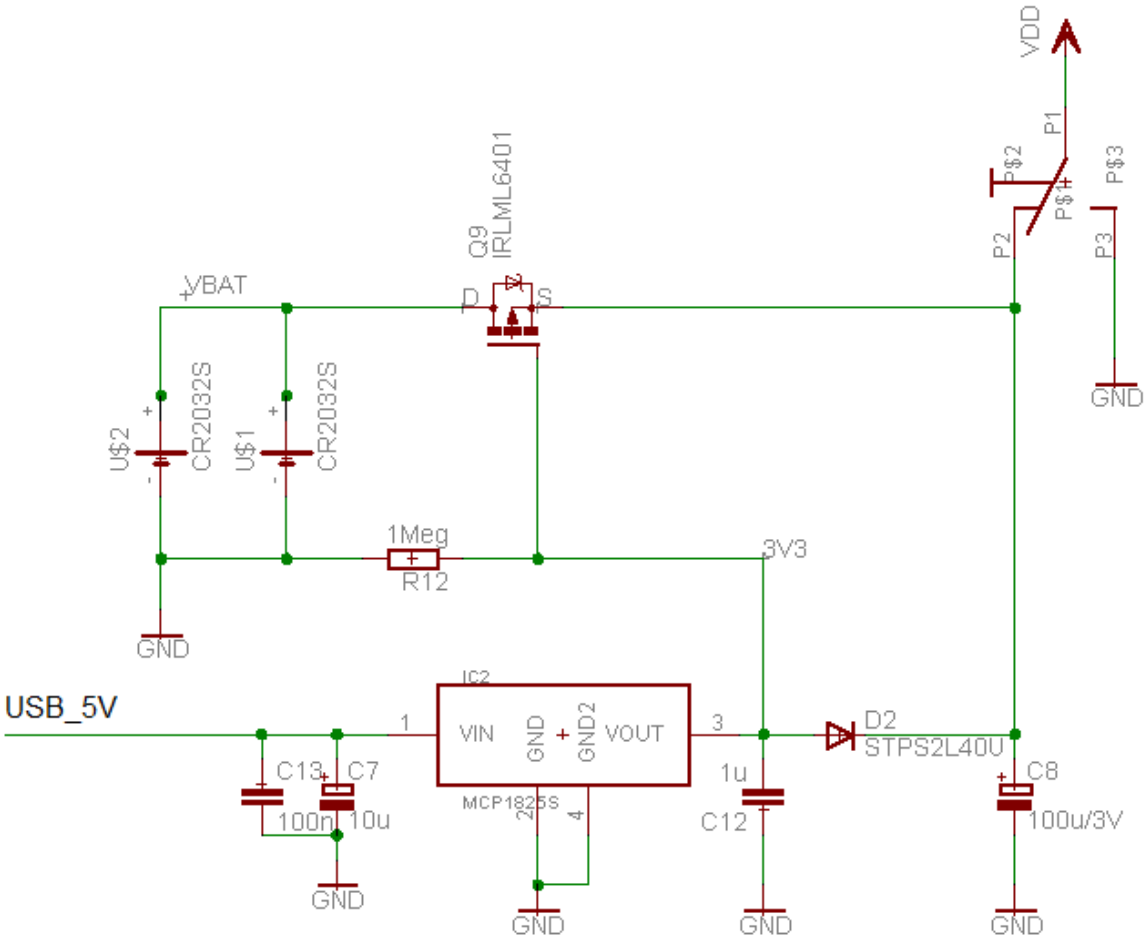


Figure 8: Power supply block

3.3.2 Display columns

The type of LED used on the matrix display is KP-1608 SRC which is a super bright red LED. Figure 9 shows relative luminous intensity of this LED. Experiments have shown that brightness of these LEDs is sufficient at very low forward currents, specifically values around 2 - 4 mA.

For driving the display, it has been decided to use the approach utilizing shift registers, specifically 74HC595, logic symbol in figure 10. This device is an 8-bit serial-in, parallel-out shift register with output latches capable of operation at supply voltage as low as 2 V and of sourcing or sinking current up to 35 mA on the parallel outputs. When the information is shifted in on the DS (serial data input) pin, an impulse on ST_CP (storage register clock

input) has to be sent to write the information into the parallel output latches. Then, when $\overline{OE}$ (output enable) is brought low, the information is available on the Q0 – Q7 parallel outputs. Q7' is a serial output and it is connected to the DS pin of the next register.

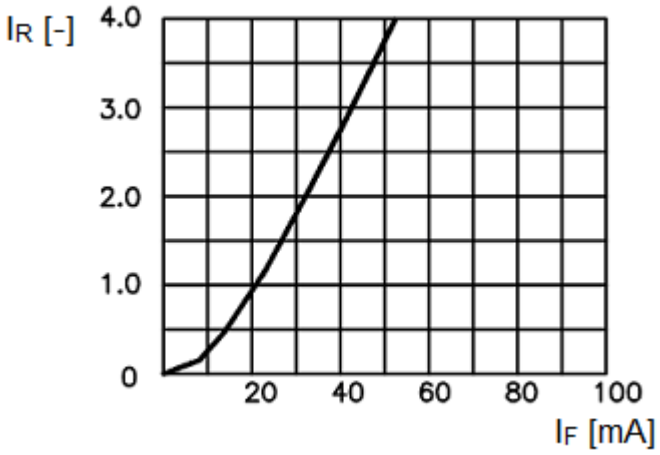


Figure 9: KP-1908 SRC super bright red LED: relative luminous intensity (I_R) vs. forward current (I_F), relative value at I_F = 20 mA [24]

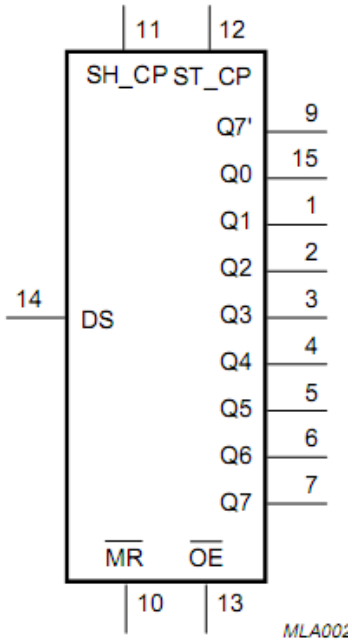


Figure 10: Logic symbol of 74HC595 shift register [25]

By connecting four of these registers in series, we get a 32-bit parallel output which is a perfect fit for 32 columns of our display. The idea is to shift 32-bit information into the registers, then move this information into parallel output latch. This will present logical 1 and

will be connected to the anodes of the LEDs. Each row of the display will be separately controlled and connected to ground when the information in the latch registers is ready. That way, the column information has to be shifted out eight times to present information on the entire display and every time, appropriate row needs to be switched on. If the rows are switched frequently enough, image on the LED matrix will appear to be displayed (see chapter 1.2 Display).

To set the forward current of the LEDs, there has to be a resistor connected in series. To calculate the value of the resistor, the forward voltage (V_F) and the forward current (I_F) of the LED need to be known as well as the supply voltage, which is determined by supply voltage of the shift registers which is determined by the supply voltage of the system which is V_{DD} = 3 V. The forward current and the related luminous intensity apply when continuous current is provided. Since each row will be swithed separately, the actual work period of each row is one eighth of the display working period. Thus, the forward current has to be adjusted:

$$I_F = 8 \cdot I_{F1} = 8 \cdot 3mA = 24mA \tag{1}$$

But this value is only theoretical; the actual one depends on the charge of the batteries and on the actual implementation of the LED. Then the resistor needed in series has a value of

$$R = \frac{V_{DD} - V_F}{I_F} = \frac{3 - 1{,}85}{0{,}024} \approx 48\Omega \tag{2}$$

Resistors of this value do not exist in common resistor sets; the closest value is 47 Ω.

3.3.3 Memory and buttons

As discussed in chapter 1.4 Memory, an external memory needs to be provided to store user data. The memory capacity is defined by the amount of data it needs to hold. When using the ISO 8859-2 encoding, one character is represented by one byte. The device is supposed to hold at least 200 text strings of at least 50 characters each resulting in a total of 10 000 bytes. 25AA256 is a 256 kb (32 kB) SPI enabled EEPROM which suits our needs.

The memory electronically consists of 64-byte pages. When writing to the memory, it is possible to write one byte or one page at a time. Page boundaries start at addresses that are integer multiple of 64. When writing a page, it is possible to start on any address but when the write cycle hits the end of the page, it will automatically continue writing from the beginning of the page. It is useful that the page size is if 64 B. This can be uses that to our advantage so

that every user data entry starts on a new page. Since the memory size is 32 kB, each byte is addressed by a 16b address. The 64-byte page system provides an ideal way of storing 512 text strings of 64 characters each.

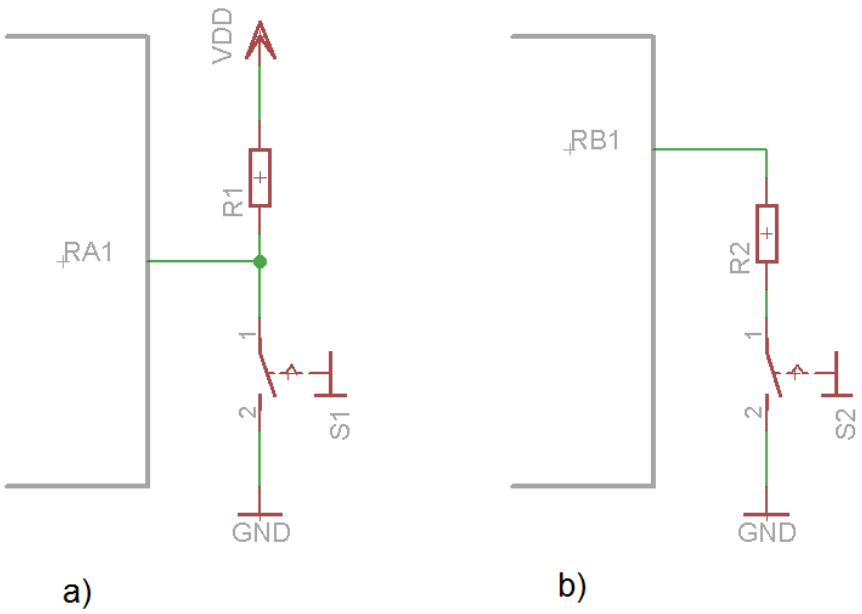


Figure 11: Two ways of connecting a button with microcontroller: a) standard, b) internal pull-up on pin

Four buttons are provided to operate the device. There are two ways of connecting buttons with an MCU (see figure 11). The standard one requires additional resistor providing logical 1 to input pin when switch is off. In the other one, the pull-up is built it the MCU itself.

3.3.4 Microcontroller

To select the appropriate MCU, we will have a look at Microchip's website [21]. The requirements are:

- Application voltage 2.5 – 3 V
- 28 general I/Os (8 for display rows, 5 for 74HC595, 4 for EEPROM, 4 for buttons, 4 for USB, 3 for programming) resulting into 44-pin package
- USB interface
- SPI interface

The search results offer 5 microcontrollers which differ in the size of program memory, ranging from 16 kB to 128 kB, and other insignificant features. Based on previous experience, the PIC18F46J50 with 64 kB of program memory is chosen. It is the flag ship of the 18F46J50 microcontroller family.

3.3.5 Display rows

When looking at the electrical characteristics of the MCU, it can be seen that maximum current sunk/sourced by PORTB and PORTC is 25 mA per pin and only 4 mA for pins of PORTA, PORTD and PORTE which means that it can not be connected to display rows directly to the MCU. The correct way to do this is to put transistor in between to work as electronic switch. Setting up the transistor's operation point is as follows.

The maximum current that can flow through the collector is current pooled from all of the LEDs in one row:

$$I_C = 32 \cdot I_F = 32 \cdot 24\,mA = 768\,mA\,. \tag{3}$$

This value is, again, theoretical. All 32 LEDs in one row would have to be switched on which never happens during normal operation since characters will be separated by a one-column space. Then the current flowing through base is

$$I_B = \frac{I_C}{h_{21E}}\ [\mathrm{A}], \tag{4}$$

where h_{21E} is DC current gain (also called h_{FE} or β) and it is to be found in the transistor datasheet as one of its parameters. Then the value of the base resistor:

$$R_B = \frac{V_{RB}}{I_{RB}} = \frac{V_{DD} - V_{BE}}{I_B} = \frac{V_{DD} - V_{BE}}{\frac{I_C}{h_{21E}}}\ [\Omega], \tag{5}$$

V_{RB} – voltage across the resistor,

I_{RB} – current flowing through the resistor,

V_{BE} – transistor base-to-emitter voltage dropdown, typically $V_{BE} = 0.65$ V.

Transistor BC817-40 has a high collector current and high DC current gain (h_{FE} = 300), the calculated value of the resistor is:

$$R_B = \frac{3 - 0{,}65}{\frac{0{,}768}{300}} \approx 918\ \Omega \tag{6}$$

The value of the resistor is going to be R_B = 1 kΩ.

3.3.6 Universal Serial Bus

Typical USB cable consists of 4 wires – V_{BUS}, D+, D- and GND. The V_{BUS} lead provides power to the connected devices, D+ and D- are data lines and GND provides common ground reference.

The 18F46J50 USB peripheral requires the following pins to be connected for the peripheral to work. D+ and D- connected with corresponding bus lines, the V_{SS} pins connected to USB bus GND line and V_{USB} pin connected to 3.0 – 3.6 V (ideally 3.3 V) voltage reference and locally bypassed to common ground with a capacitor of at least 100 nF capacity. If the device is dual powered (e.g. by the USB bus and by batteries), there has to be a sense pin determining USB attached state of the device. For this reason, a pair of resistors is provided on the V_{USB} line as a voltage divider.

3.3.7 Serial Peripheral Interface

The SPI interface of the microcontroller is used in two ways – for providing information to the shift registers and for communication with the external EEPROM. Connection with the shift registers requires 5 lines: serial data out, serial clock, a line for transferring the information into the output latches (storage register clock), master reset and output enable. For the SPI interface to work, serial data out and serial clock lines have to be connected to corresponding I/O pins, the other lines can be connected to any other I/O.

When connecting the EEPROM with the MCU, at least 4 lines are required. The actual number of lines is dependent on how the memory is used and whether one or more physical memory chips are used. The SPI bus can operate with a single master device and with one or more slave devices. A chip select line is required for each slave device and only one slave device can be active at a time. The other lines are serial data out, serial data in and clock signal. From the master's point of view, serial out is connected to serial in on the memory chip and vice versa. Other I/O pins on the memory chip include write protect, hold and power

supply pins. Write protect pin represents one of several ways of preventing accidental writes into the memory. Hold pin allows master to pause communication to the memory chip. Both pins can be connected to V_{DD} permanently if neither of these functionalities is desirable.

3.4 Printed circuit board

When designing the printed circuit board, it needs to be kept in mind that the weight of the batteries is considerable with respect to the other parts of the PCB. It also needs to be kept in mind that the front side of the device is to be without any components besides the display and the buttons. The top and bottom layer of the PCB can be found in appendices B and C, respectively.

For the purposes of this project, the main board should be as light as possible, so the thickness of the base material should be as low as possible but, on the other hand, it needs to be able to withstand some mechanical strain when connecting and disconnecting USB cable or changing batteries. The way to decide the thickness would be to manufacture several PCBs with different thickness, test them in working environment and then decide.

The PCB for this project will be manufactured by a domestic PCB producer PragoBoard s.r.o. This company offers a cheep way for manufacturing prototype PCBs called “Pool service”. There are two aspects where the prototype PCB will differ from the design and one of them is the thickness of the main board. Since PCB manufacturing is preceded by a lot of support work, it would be very expensive to do all of it for just one PCB so there are several standards set up for the Pool service which include the thickness of the manufactured PCB to be 1.5 mm.

The second aspect is the programming connector which is designed to fit the ICD2 receptacle. If the device is released for mass production, the programming will be done by pads taken out to the very edge of the main board and a suitable programming dock. The overall design of the PCB is shown in appendices C and D and photographs of the board in appendix E. The dimensions of the PCB are 103.5 × 55.2 mm. For purposes of this project, the display board is mounted on the main board via SIP sockets.

4 Software for microcontroller

As discussed earlier, Microchip MPLAB IDE in conjunction with Microchip MPLAB ICD2 is used to program the microcontroller. The code itself will be written in C programming language and compiled by MPLAB® C Compiler for PIC18 MCU (MCC18).

4.1 Microcontroller configuration

The software design starts with setting up the MCU to fit its surrounding circuitry. This can be done by setting up the configuration word (also called configuration bits or fuses). Fuses are control registers that can only be set during programming. When using the MCC18 compiler, fuses are introduced by the #pragma directive and keyword “config”.

Because of very unique requirements of the USB module for stable clock source, the clock source block of the 18F46J50 is more complicated than with USB not-enabled devices. USB module requires clock input of either 48 MHz (for full-speed device) or 6 MHz (for low-speed device). The device will be designed to comply with full-speed specification. The PIC18F46J50 clock diagram is shown in figure 12. It is not necessary to use a 48 MHz external oscillator though. The MCU is equipped with a 96 MHz PLL and a postscaler with 48 MHz output. The PLL requires a 4MHz input and it is equipped with a 1 – 12 prescaler so that it is possible to use external oscillators of various frequencies. Internal oscillator can not be used for USB module because it is not stable enough so an external oscillator of f_{OSC} = 20 MHz is provided.

There are three ways to get MCU core (also called CPU) clock. Either the 48 MHz for USB module can be taken and divided further or internal or secondary external oscillator can be used. It is not desirable to use another external oscillator and internal oscillator only generates frequencies up to 8 MHz which means the USB clock will be used.

To set up the oscillator block, PLLDIV (controlling the PLL prescaler) and OSC (controlling oscillator mode selection) need to be set accordingly:

```
#pragma config PLLDIV = 5
#pragma config OSC = HSPLL
```

OSC register controls oscillator source for the PLL block as well as for the CPU. The next step is setting up the clock source division for the CPU:

```
#pragma config CPUDIV = OSC3_PLL3
```

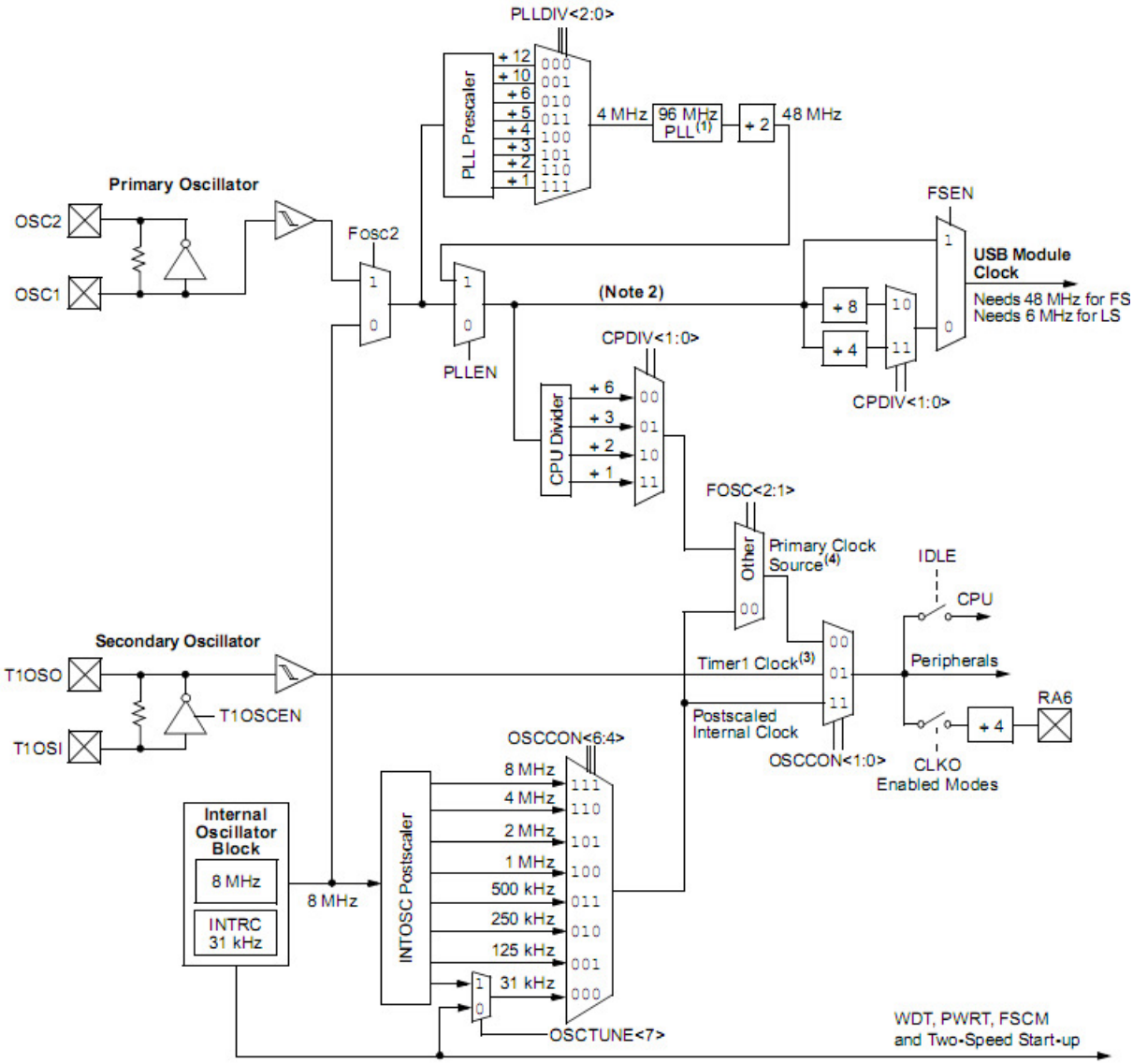


Figure 12: PIC18F46J50 clock diagram [23]

CPUDIV sets CPDIV and OSCON registers. This means that system primary oscillator frequency (external oscillator of 20 MHz) is divided by 5 to get a 4MHz input clock for PLL generating 96 MHz which is subsequently divided by 2 to get 48 MHz clock for USB module. This value is also taken to the CPU prescaler and it is divided by 3 resulting in 16 MHz CPU clock.

4.2 Program overview

Several global variables need to be defined during initialization of the system to hold various runtime values. If the value of a variable is going to be changed within the interrupt routine, it is necessary to prefix its definition with “volatile” modifier. This tells the compiler

not to optimize usage of this variable. Compilers are made to “assume” that a value of a variable can not change on its own, which is exactly what happens during ISR from the main program point of view.

Table 4: List of some of the global variables used in program referred to in the text:

Variable	Size, description
`unsigned char word_on_disp[64]`	8 b, holds string currently being displayed
`unsigned long dispBuffer[8]`	32 b, holds data retrieved from `charSet` for currently used string
`unsigned long buff`	32 b, holds data retrieved from `dispBuffer` for currently selected row
`unsigned char moveNowOneBitToLeft`	8 b, indicates it is time to move displayed string one bit to left (set up by timer)

When defining larger amount of variables, the compilation may result in an error saying that it is not possible to fit all variables into the selected memory block. MCU memory is split into several blocks called banks or databanks. When not explicitly defined, compilers use the first bank and when they run out of space, an error is issued. To fix this, linker file specified in build options needs to be edited. All available databanks are listed with defined start and end addresses and a name. Part of linker file for PIC18F46J50:

```
DATABANK   NAME=gpr0       START=0x60            END=0xFF
DATABANK   NAME=gpr1       START=0x100           END=0x1FF
DATABANK   NAME=gpr2       START=0x200           END=0x2FF
DATABANK   NAME=gpr3       START=0x300           END=0x3FF
DATABANK   NAME=gpr4       START=0x400           END=0x4FF
DATABANK   NAME=gpr5       START=0x500           END=0x5FF
DATABANK   NAME=gpr6       START=0x600           END=0x6FF
DATABANK   NAME=gpr7       START=0x700           END=0x7FF
DATABANK   NAME=gpr8       START=0x800           END=0x8FF
DATABANK   NAME=gpr9       START=0x900           END=0x9FF
DATABANK   NAME=gpr10      START=0xA00           END=0xAFF
DATABANK   NAME=gpr11      START=0xB00           END=0xBFF
DATABANK   NAME=gpr12      START=0xC00           END=0xCFF

SECTION    NAME=USB_VARS           RAM=gpr12
SECTION    NAME=CHARSET_DATA       RAM=gpr11
SECTION    NAME=VARS               RAM=gpr10
SECTION    NAME=CONVERSION         RAM=gpr9
```

When defining a large variable, for example the character set, it is necessary to allocate this variable in a separate databank. To tell the compiler where to put the variable, a section in the linker file needs to be created and `#pragma udata [section name]` directive used in code to apply the allocation.

The main idea of how the correct data for display are retrieved is as follows. The character set (as described in chapter 1.6) is going to be saved in two-dimensional array:

```
unsigned char charSet[224][8]
```

The idea is to keep 8 5-bit words reflecting 8 rows and 5 pixel width of each character. Since there are no 5-bit variable types, an 8-bit one (unsigned char) needs to be used. The first index of the variable is designation of a character while the second index is designation of selected row. The number 224 comes from the number of characters encoded by ISO 8859-2 from which the first 32 unused characters are subtracted. The content of this variable has to comply with the ISO 8859-2 table of characters.

When accessing a value from this array, only the character desired and the row information are needed.

```
charSet[(unsigned char)'A' - 32]
```

This will return 8 bytes of data and if printed out in a suitable way, the following is obtained:

```
0 0 0 0 0 0 0 0        byte 1
0 0 0 0 0 1 0 0        byte 2
0 0 0 0 1 0 1 0        byte 3
0 0 0 1 0 0 0 1        byte 4
0 0 0 1 0 0 0 1        byte 5
0 0 0 1 1 1 1 1        byte 6
0 0 0 1 0 0 0 1        byte 7
0 0 0 1 0 0 0 1        byte 8
```

This way, any character in the character set can be easily accessed as long as all parts of the system dealing with encoding remain compliant. User data in the ISO 8859-2 encoding will be send to the device and stored in the EEPROM. When a text string is required to be displayed a variable `word_on_disp` will be filled with corresponding data.

For the user to operate the device, four buttons of different functions are provided. To make the operation easy, the buttons have the following functionalities: Enter, Escape, Move

next, Move previous. From the user's point of view, the device can be in four different states or menu levels. A push of a button causes setting a change menu level flag and sets the new state. The states and the operation of the device from user's point of view are described in table 5.

Table 5: List of device states

State	Description
0	Welcome state – "Welcome" on display. If no user data present, Escape button causes to display "No data". If user data present, Enter or Escape button causes to switch to state 1.
1	Selection state – a letter of English alphabet is displayed. Enter button causes to switch to state 2. Next and Previous buttons cause change of letter.
2	Search state – search for desired character string based on current letter. Next and Previous buttons cause change of string. Escape button causes to switch to state 1. Enter button causes to switch to state 3.
3	Display state – displaying selected character string indefinitely Escape button causes to switch to state 1.

Even though user data can contain other than English alphabet characters, the selection state only supports these. It would be very long if support for all of the characters of ISO 8859-2 encoding was made. A conversion table between these sets is provided for the search. When 'A' is selected in selection state, search state will provide all of user data strings beginning with 'A', 'Á', 'Ą', 'Â', 'Ă', 'Ä' and likewise for other characters. To represent non-letter characters, symbol '#' has been implemented.

When entering state 2 and no user data is compliant with selected character, "No data" is displayed. It is possible to go to selection state by pressing the Escape button.

In the program main loop, displaying characters on the display needs to be processed as well as managing buttons and USB requests. Figure 13 shows the program workflow setup phase and figure 14 shows the loop phase.

Several internal variables are set up when the device powers up for the very first time. That is checked by reading the third byte in the EEPROM. The setup of internal variables include writing the first power-up byte and "Welcome" and "No data" strings into the EEPROM, setting currently selected flag to zero and setting no data flag.

There are also asynchronous events processed by the interrupt routine. These include USB interrupts and timer interrupts. Timer 0 is used for displaying selected string on the display (see chapter 3.4).

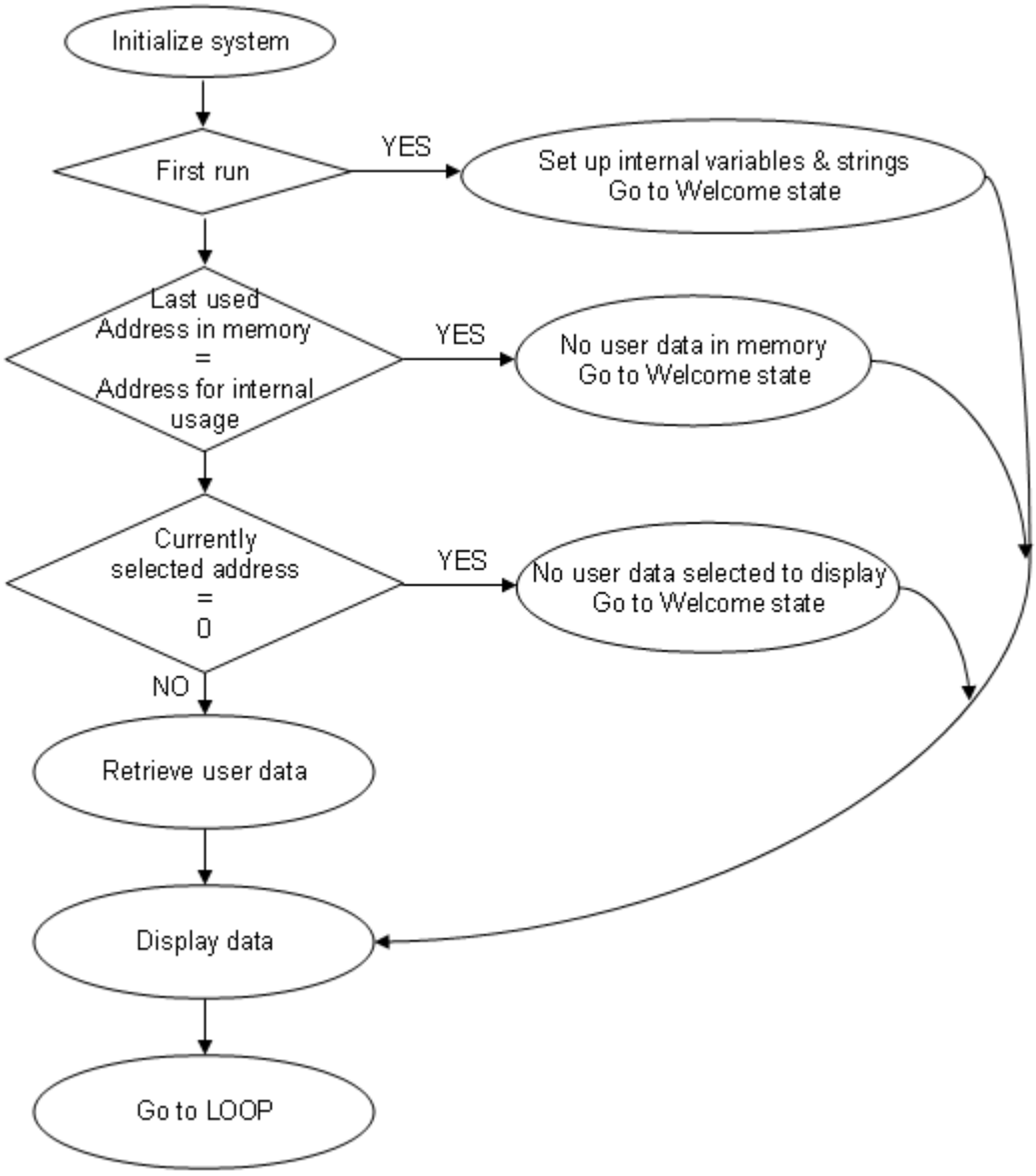


Figure 13: Program workflow diagram – setup phase

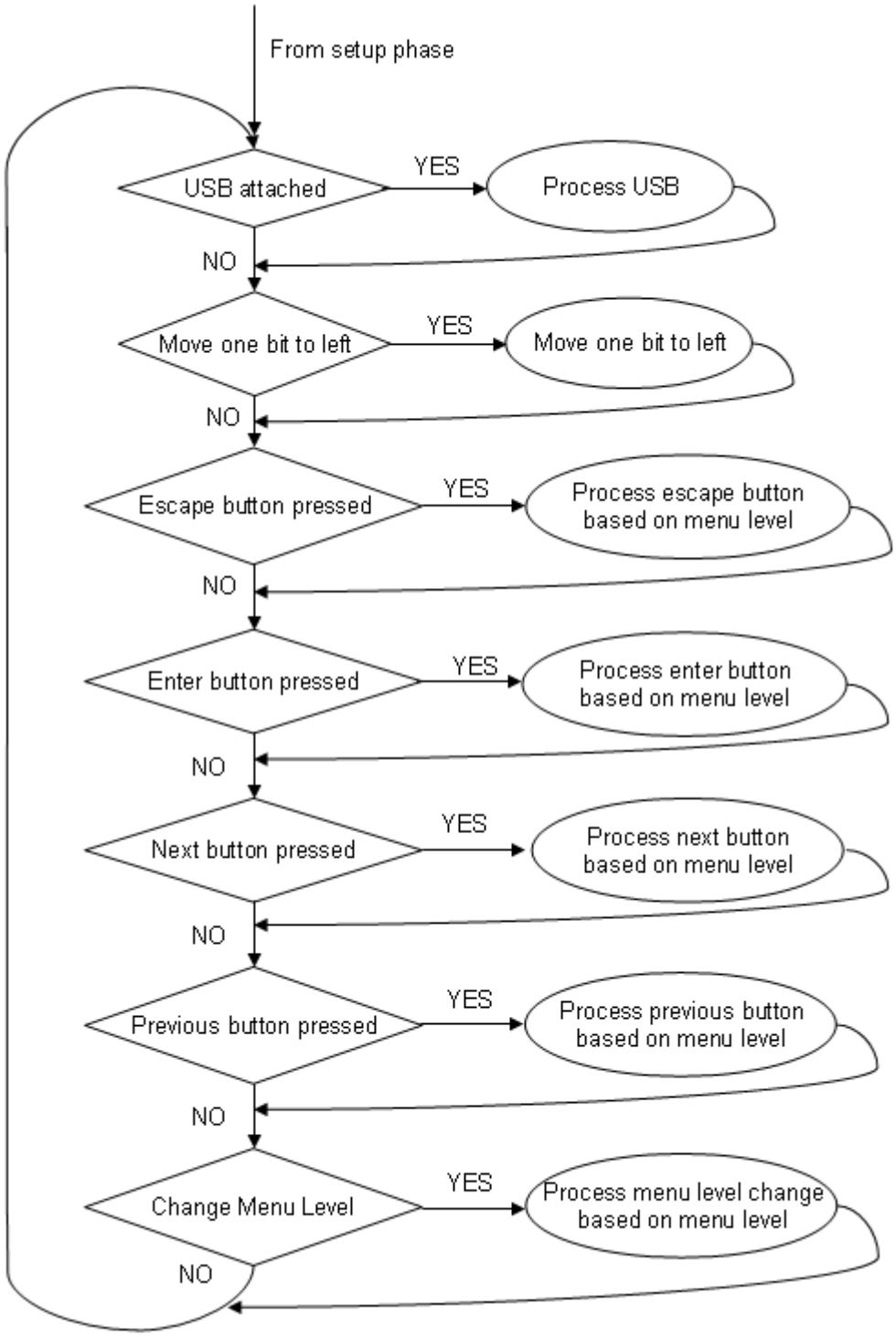


Figure 14: Program workflow diagram – loop phase

4.3 Memory overview

For the device to work independently on current power state, various states of the device and variables are going to be stored in the EEPROM. Table 6 shows organization of the first page of EEPROM.

Table 6: First page of EEPROM overview

Variable description	Physical address	Name in the program
High byte of address of last user-selected string	0x00	lastUsedAddH
Low byte of address of last user-selected string	0x01	lastUsedAddL
High byte of address of currently user-selected string	0x02	currentlyUsedAddH
Low byte of address of currently user-selected string	0x03	currentlyUsedAddL
Byte signaling the very first device power-up	0x04	firstPowerUpAdd
0x01 if user has selected a string to display or 0x00 if user has not selected one	0x05	currentSelectedAdd

Since the EEPROM capacity is 32 kB, all bytes are addressed by a 16-bit address. To store this address, it has to be written one byte at a time. Communication with the EEPROM is determined by its instruction set (table 7). There is a STATUS register reflecting the status of the memory array. All communication is on MSB-first basis.

Table 7: 25AA256 instruction set [26]

Instruction name	Instruction format	Description
READ	0000 0011	Read data from memory array beginning at selected address
WRITE	0000 0010	Write data to memory array beginning at selected address
WRDI	0000 0100	Reset the write enable latch (disable write operations)
WREN	0000 0110	Set the write enable latch (enable write operations)
RDSR	0000 0101	Read STATUS register
WRSR	0000 0001	Write STATUS register

On the MCU side of the communication procedure, Master Serial Synchronous Port is configured as an SPI master interface which consists of a serial receive/transmit buffer register SSP1BUF and a shift register SSP1SR which is not directly accessible. When a byte is written to SSP1BUF, it is automatically written to SSP1SR as well and clocked out. At the time of shifting out, another byte is clocked into the SSP1SR and rewritten to SSP1BUF when shifting in is over. The state of SSP1BUF is indicated by BF (buffer full) bit of SSP1STAT register. Before any other writing into SSP1BUF, the register has to be read.

First four pages of the memory will be allocated for internal usage, resulting in 508 pages available for user text strings.

4.3.1 Memory write procedure

As it has been said before, it is possible to write one byte or one page (64 bytes) of data at a time. The write procedure is as follows:

1. Pull chip select low
2. Issue a WREN instruction
3. Pull chip select high
4. Pull chip select low
5. Issue WRITE instruction
6. Issue a 16-bit address
7. Issue data byte
8. Pull chip select high

At point 4, it is possible to continue issuing up to 64 bytes of data. For the data to be actually written to the array, it is necessary to bring chip select high after the last byte has been clocked in. It also takes some time (maximum of 5 ms, [26]) to complete the internal write cycle after the chip select pin has been brought high. During this period, it is not possible to access the data in the memory array but it is possible to read the STATUS register. The WIP (Write-In-Progress, STATUS<0>) bit indicates the EEPROM is busy with write operation. When set to logical 1, a write is in progress. It is a read-only bit.

It has been observer that if an interrupt occurs when communicating with the EEPROM, an error occurs. Thus, prior to any attempt for communicating with the memory, all interrupts will be disabled. Following these rules, the program write procedure for writing 64 bytes of data into the EEPROM is as follows:

1. Disable GIE (Global Interrupt Enable)
2. Pull chip select low
3. Read dummy from SSP1BUF (just to make sure it is empty)
4. Write WREN instruction into SSP1BUF
5. Wait till BF is set (indicating that the byte has clocked sent)
6. Read dummy byte from SSP1BUF
7. Pull chip select high
8. Pull chip select low
9. Write WRITE instruction to SSP1BUF
10. Wait till BF is set
11. Read dummy byte
12. Write address high byte to SSP1BUF
13. Wait till BF is set
14. Read dummy byte
15. Write address low byte to SSP1BUF
16. Wait till BF is set
17. Read dummy byte
18. Write data byte to SSP1BUF
19. Wait till BF is set
20. Read dummy byte
21. Repeat 18, 19, 20 sixty three times.
22. Pull chip select high
23. Read STATUS register
24. Wait till WIP is 0 by reading the STATUS register over and over again
25. Enable GIE

When writing only one byte into the memory field, skip item 21.

Reading the status register procedure:

1. Read dummy byte from SSP1BUF
2. Write RDSR instruction to SSP1BUF
3. Wait till BF is set
4. Read dummy byte

5. Send dummy byte
6. Wait till BF is set
7. Read from SSP1BUF

Since the STATUS register is read when a write procedure is in progress, it is not necessary to disable/enable GIE.

4.3.2 Memory read procedure

To read 64 bytes from the EEPROM memory field a 64-byte buffer (`pageOfData`) is needed. The idea is to send a READ instruction, the address and then keep sending dummy data to clock in the data from the memory. The procedure for retrieving 64 B of data from SPI EEPROM is as follows:

1. Disable GIE
2. Pull chip select low
3. Read dummy from SSP1BUF (just to make sure it is empty)
4. Write READ instruction to SSP1BUF
5. Wait till BF is set
6. Read dummy byte
7. Write address high byte to SSP1BUF
8. Wait till BF is set
9. Read dummy byte
10. Write address low byte to SSP1BUF
11. Wait till BF is set
12. Read dummy byte
13. Write dummy byte to SSP1BUF
14. Wait till BF is set
15. Read byte from SSP1BUF and assign it to `pageOfData`
16. Increment `pageOfData` index
17. Write dummy byte
18. Wait till BF is set
19. Repeat 15, 16, 17, 18 sixty three times
20. Pull chip select high
21. Enable GIE

When reading only one byte from the memory field, skip item 19.

4.4 Display

There are two ways of displaying text – static and non-static (or moving). Displaying static text is only used in the selection state of the device when only one character of English alphabet is displayed. A variable called `showStatic` has been implemented to hold information about whether or not to present static text. This variable is set to 1 when entering selection stage; otherwise it is set to 0.

When only static text is shown, variable `dispBuffer` holds the same value trough the entire time before the displayed text is changed. When moving text is shown, it is necessary to update this variable periodically so that it holds correct data. For the display to present ideal visual information, the period of switching rows and moving the text has to be precise and in sync. Thus, it can not be handled in the main loop but it has to be handled by a timer. Timer 0 is a 16-bit timer/counter to which a prescaler can be assigned. When the timer overflows the interrupt flag is set and two events handled. One of them is refreshing the display (switching the rows), the other one relates to moving the text (changing the value in `dispBuffer`) if allowed by `showStatic`. When this is triggered, a flag called `moveNowOneBitToLeft` is set to 1 and it is caught up later in the main loop.

The text movement is quite a long procedure thus it can not be handled in ISR. Several variables are used: one indicating the position of the text on display, another indicating the end of the text, and another one indicating character index of the text. When `moveNowOneBitToLeft` is set to 1, the movement procedure is run in the main loop and works as follows. The existing information in `dispBuffer` is shifted to left by one bit and another bit of information is added into its place for each row. This bit is masked out from the corresponding byte in `charSet` using a nested index of text on display and current character.

The SPI interface is used to clock out information into the shift registers. Prior to switching on a new row, the current one has to be switched off so that no unwanted effects on the display occur. That is done by pulling $\overline{\text{OE}}$ pin of the registers high. Then the row is changed and correct 32-bit information is retrieved from `dispBuffer`. Then the information is clocked out 8 bits at a time using similar procedure to writing into EEPROM. Then an impulse has to be sent on the ST_CP line so that the information is moved to the output latches. The last step is switching the row on which is done by pulling $\overline{\text{OE}}$ low.

4.5 Universal Serial Bus

Setting up the USB module is very complicated so Microchip provides so-called "USB stack" which is a ready-to-use pack of code files in C for quick application development. When implemented, several adjustments are needed:

HardwareProfile.h – USB bus sense configuration and system clock frequency

usb_descriptors.c – Product string descriptor and maximum power consumption

Otherwise, the pack is ready to be used. One endpoint and one buffer handler for each direction is set up with 64-byte buffers (`INPacket` and `OUTPacket`). The buffers can be only accessed when the corresponding USB handler is not working with them. More information on buffer ownership can be found in [23]. The names of the buffers comply with the direction of the data transfer from the point of view of the USB host. If the host sends a packet of data to the endpoint OUT buffer, the USB peripheral hardware will automatically receive it and store the data. Additionally, the endpoint handler will indicate that the endpoint is no longer busy and the data can be accessed.

The communication between the host PC and the device is directed by the first byte in the transfer. The list of commands and the corresponding actions are listed in table 8. In the PC program, several actions are possible (see chapter 4) and various commands are sent to the device. The commands are standard ASCII control characters.

Table 8: List of commands and corresponding action taken by the MCU

Command	Description and/or action
0x02	Start of text (STX) – the following bytes are user data till End of text (0x03)
0x05	Enquiry (ENQ) – send firmware version to host
0x07	Bell (BEL) – respond with ACK (0x06)
0x11	Device control 1 (DC1) – memory reset
0x12	Device control 2 (DC2) – send all user data to host
0x13	Device control 3 (DC3) – send number of records in user memory to host
0x14	Device control 4 (DC4) – send available memory to host

When STX is received the following bytes are stored into another buffer and written to EEPROM. "Sending" information back to the host is done by writing into `INPacket` and assigning the handler to state of write which results in taking ownership of the endpoint buffer and writing into it the content of `INPacket`. The quotes are in place because the device can

not send anything to the host. The information is only presented in the USB module I/O buffer for the host to read which will happen the next time the endpoint is polled by the host.

In the main loop, there is a check (USB_SENSE) whether the device is attached or not to the USB bus and appropriate action is taken. The action is handled by the USB stack.

The buffer size defines the amount of data that is possible to send to and from the device at once. Since there has to be a control character at the very beginning of the packet and a termination character indicating the end of user defined text string, it is possible to send maximum of 62 bytes of user data at a time thus defining the maximum length of the user text string.

5 Windows application

A .NET Winform application has been designed to manage the device from Windows based PC. The main purpose of this application is to load data into the device, clear its memory or load the data from the device into the PC. When connected to PC, the device is recognized as a WinUSB device and appropriate driver is installed. Then, the application can be run. It comes with LibUsbDotNet library [22] which can access the device. In the operating system and all of its applications, the device is always shown as a "Microchip WinUSB device". USB devices are not directly accessible from the application environment; work with them is done via their drivers. The library provides support for writing and reading from the driver.

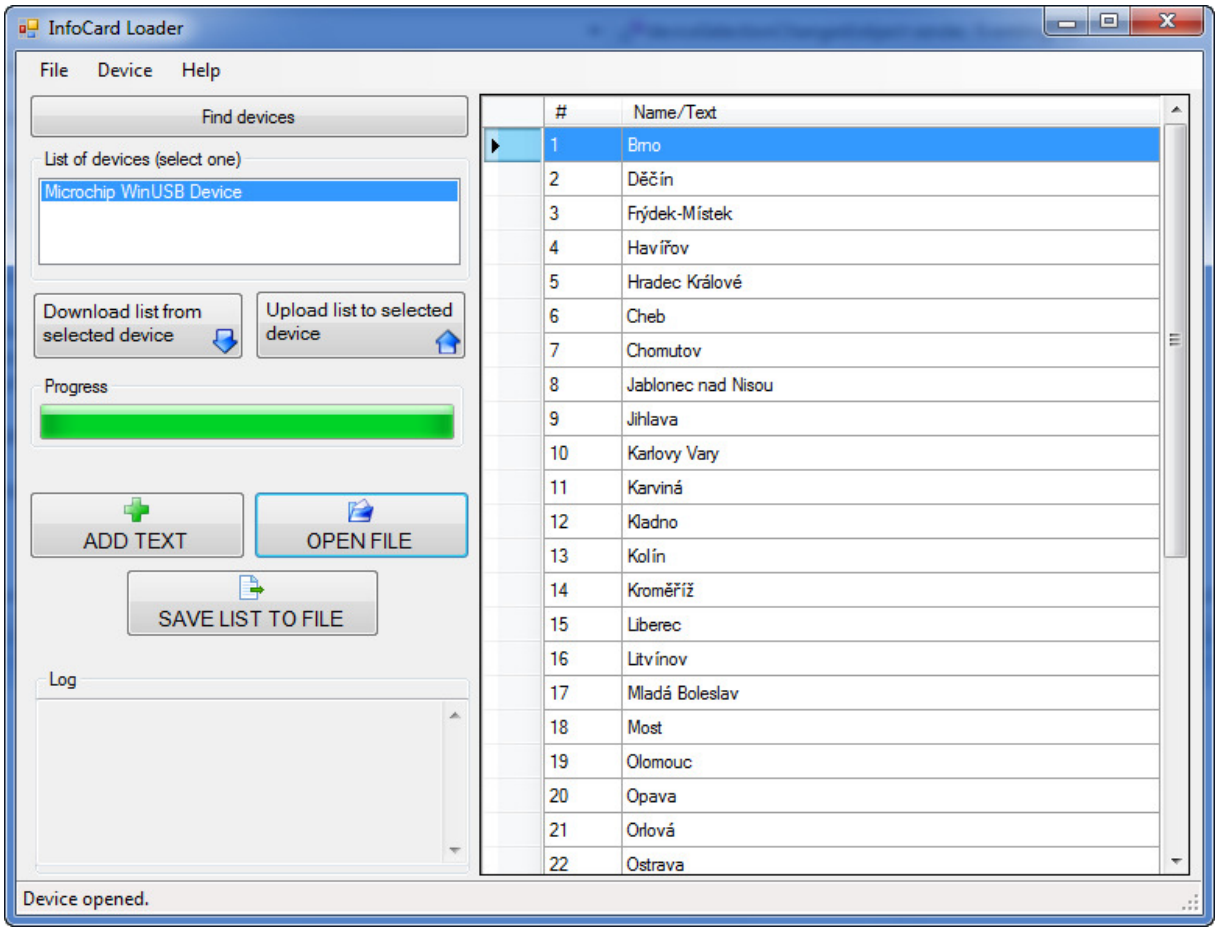


Figure 15: Supporting Windows application

When the application is started, a search for connected devices has to be issued. It is possible for several devices to be connected to the host PC but only one device can be communicated to at a time. The list of accessible devices is shown and it is necessary to select

one before proceeding. When a device is selected, the application will try to open the device – i.e. establish one IN and one OUT endpoint. All important events are logged in the text box designated “Log”, some information is presented in the application status bar.

The application is capable of opening and reading Microsoft Office Excel files (version 2003 and 2007 as these are the most common ones) and text files. Reading an Excel file is done via OLE-DB Windows API and it is necessary that an appropriate version of MS Office or MS Office expansion pack is installed on the user computer otherwise the API is not available. Reading text files is not underlined with any conditions.

When the list is not empty, it is possible to send it to the device or save it as a text file. Sending the data to the device or downloading them from the device is indicated by the progress bar. When sending is finished, it is possible to disconnect the device and use it. Items of the list can be added via opening files or by adding them directly by hand. Local context menu on the list provides delete operations on the list and matching alphabetical order of the items. Under the main menu Help option, an HTML help file is available.

Sending the list items to or downloading them from the device is done one by one. Each text string is prefixed with STX byte and suffixed with ETX byte. It is possible for the items in the list to be of 62 characters in length. This is determined by the size of the buffer which is 64 bytes but the prefix and the suffix need to be taken into consideration. A longer text strings will be automatically shortened and the user will be notified via the log.

6 User guide

When the device is turned on for the very first time, there is no user data in memory and the device is showing “Welcome!” on the display. The control buttons are: Escape (top left), Enter (bottom left), Next (bottom right), Previous (top left). If the Escape button is pressed at this point, ‘No data” will be shown on the display.

Connect the device to a Windows based computer and run the InfoCard Loader application. It does not matter whether you first connect the device or run the application. In the application, the “Find devices” button has to be hit and then the required device has to be selected in the list below. If no devices are found, a warning message is shown in the application status bar. By selecting the device in the list, the application connects to the device and it is possible to start using it. If an attempt is made to communicate with the device without it being selected in the list, an error will be shown.

The “Open file” button allows user to open a file in supported format. Supported file formats are Microsoft Office Excel 2003 and 2007 (*.xls, *.xlsx), text files (*.txt). When opened, the content of the file is loaded into the list on the right. It is possible to add items to the list manually by double clicking on the row marked * or by using the “Add text” button. Content of the item can be up to 62 characters long. When trying to add text longer than 62 characters, the application will automatically downsize the text and issues a warning in the log. It is possible to upload 508 items into the device.

When the application is connected to the device, it is possible to:

- Upload the content of the list into the device
- Download the content of user memory from the device (if any)
- Get system information from the device
- Delete user memory in the device

Uploading, downloading and file opening progress are indicated by the progress bar.

A log is provided to keep track of the operation. Application help is available under the main menu Help option or by pressing F1.

When user data is uploaded to the device, the device can be disconnected from the computer. At this point, it is still showing the welcome screen. By pressing the Escape button, the device enters its menu and letter ‘A’ is displayed. At this point, it is possible to use the Next and Previous buttons to display other letters. If the Enter button is pressed, a search is done for user data starting with the selected letter. Only letters with no diacritics are available but the search results return all corresponding characters. If no user data is found, “No data” appears on the screen. To return to the alphabet menu, press Escape.

When a letter is selected, it is possible to browse the corresponding user records by Next and Previous buttons. By pressing Enter when chosen text is being displayed, the selection is confirmed and it is no longer possible to browse the data. This state of the device is considered to be the working state in which the device stays most of the time. To return to the alphabet menu, press Escape.

7 Price summary

The price of the device is summed up in table 9.

Table 9: Device cost breakdown (17/05/2011)

Component	**Price [CZK]**
Display (FEEC, BUT)	450
SMD and THT components ([19], bulk buy)	323
Main board (PragoBoard, s.r.o, 10 boards)	236
Batteries	25 × 2
Total	1059

To this price, the cost of soldering, testing and software developing should be added. For purposes of this project, this is omitted. Usually, it is not possible to buy components by one piece but to buy them in bulk. The price has been adjusted to fit the number of components actually used but bought in bulk. The price of the PCB can differ based on the number of boards ordered. This value has been calculated on a base of ordering 10 boards with solder mask, HAL and screen printing.

8 Conclusion

The device has been designed according to assignment. From the user's point of view, the device is a battery powered electronic information card that consists of and LED display capable of showing various user defined text strings and buttons for easy operation. Resolution of the display is 32 × 8. The main application of this device is usage as a nametag. For this reason, the main board of the device is equipped with two holes and the device can hold up to 508 character strings (names) each of 62 characters long in its internal memory. The data are loaded via computer application.

The device is a high-speed WinUSB device and when connected to the computer, it requires the WinUSB driver which is included in Windows XP SP2 and higher versions of Windows operating systems. An application is provided to communicate with the device when connected. The application is designed to load data into the device as well as to retrieve them from the device.

When user data is present in the device's memory, it is possible to browse them using a simple menu and the buttons. The data is stored in an SPI enabled EEPROM. Information on the display is presented by using four 8-bit shift registers with output latches. The USB module is powered by an external oscillator on the hardware side and Microchip USB stack on the software side. The device is capable of drawing power either from two coin cell batteries or, when connected, from USB. The heart of the device is MCU PIC18F46J50.

The price of the prototype is calculated to 1059 CZK when ordering 10 boards and ordering components by bulk. The device has been tested and when powered by two coin cell batteries Energizer CR2032 (240 mAh, [30]), it will work continuously for 4 hours.

9 References

[1] *Deluxe flashing scrolling LED Name* [online]. 2007 [cit. 2011-05-15]. Available: http://www.kustomxpress.com/Deluxe-LED-Name-Badge-p/bdg-pro-rd-r.htm

[2] *Led name card* [online]. 2009 [cit. 2011-05-15]. Available: http://www.tradett.com/products/u26520p158406/led-name-card-led-name-badge.html

[3] *Hitachi's worker tracking tags* [online]. 2007 [cit. 2011-05-15]. Available: http://www.engadget.com/2007/06/26/hitachis-worker-tracking-tags/

[4] *Hitachi's employee-tracking AirLocation II Tag-w WiFi-enabled RFID tags* [online]. 2006 [cit. 2011-05-15]. Available: http://www.m-indya.com/shownews.php?newsid=2081

[5] *Návrh elektronických přístrojů* [online]. 2009 [cit. 2011-05-15]. Available: http://www.umel.feec.vutbr.cz/mnen/Files/kap_5.pdf

[6] *Persistence of vision* [online]. 2011 [cit. 2011-05-15]. Available: http://en.wikipedia.org/wiki/Persistence_of_vision

[7] *Charlieplexing LEDs- The theory* [online]. 2011 [cit. 2011-05-15]. Available: http://www.instructables.com/id/Charlieplexing-LEDs--The-theory/

[8] *SPI - Overview and Use of the PICmicro Serial Peripheral Interface* [online]. 2002 [cit. 2011-05-15]. Available: http://ww1.microchip.com/downloads/en/devicedoc/spi.pdf

[9] *Introduction to Serial Peripheral Interface* [online]. 2002 [cit. 2011-05-15]. Available: http://www.eetimes.com/discussion/beginner-s-corner/4023908/Introduction-to-Serial-Peripheral-Interface

[10] *USB.org - Documents* [online]. 2011 [cit. 2011-05-15]. Available: http://www.usb.org/developers/docs/

[11] *A Technical Introduction to USB 2.0* [online]. 2011 [cit. 2011-05-15]. Available: www.usb.org/developers/whitepapers/usb_20g.pdf

[12] AXELSON, Jan. *USB Complete: Everything You Need to Develop Custom USB Peripherals*. Madison WI, USA: Lakeview Research, 2005. 572 p.

[13] *WinUSB (Windows Driver Kit)* [online]. 2011 [cit. 2011-05-15]. Available: http://msdn.microsoft.com/en-us/library/ff540196.aspx

[14] *ISO 8859-2 Character Set* [online]. 1996 [cit. 2011-05-15]. Available: http://nl.ijs.si/gnusl/cee/charset.html

[15] *Proč právě ISO-8859-2?* [online]. 1997 [cit. 2011-05-15]. Available: http://www.cestina.cz/whyISO.html

[16] *ISO/IEC 8859-2* [online]. 2011 [cit. 2011-05-15]. Available: http://en.wikipedia.org/wiki/ISO/IEC_8859-2

[17] *PDK-USB3-XXXXXX VF POS Pole Display Kit: Installation and operating instructions* [online]. 2002 [cit. 2011-05-15]. Available: www.ieeinc.com/specs/PDK_USB3_INOPML_REVA.pdf

[18] *Using MPLAB ICD2* [online]. 2005 [cit. 2011-05-15]. Available: http://ww1.microchip.com/downloads/en/devicedoc/51265g.pdf

[19] *TME Electronic components* [online]. 2011 [cit. 2011-05-15]. Available: http://www.tme.eu/cz/

[20] *Farnell* [online]. 2011 [cit. 2011-05-15]. Available: http://cz.farnell.com/

[21] *Product Selector Tool* [online]. 2011 [cit. 2011-05-15]. Available: http://www.microchip.com/productselector/MCUProductSelector.html

[22] LibUsbDotNet C# USB Library [online]. 2011 [cit. 2011-05-15]. Available: http://sourceforge.net/projects/libusbdotnet/

[23] PIC18F46J50 Family Data Sheet [online]. 2005 [cit. 2011-05-15]. Available: http://ww1.microchip.com/downloads/en/devicedoc/39931b.pdf

[24] *KPT-1608 Datasheet* [online]. 2011 [cit. 2011-05-16]. Available: http://www.datasheetcatalog.org/datasheets2/61/612074_1.pdf

[25] *74HC595 8-bit serial-in, serial or parallel-out shift register with output latches* [online]. 2003 [cit. 2011-05-16]. Available: http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf

[26] *Microchip 25AA256 256K SPI™ Bus Serial EEPROM* [online]. 2005 [cit. 2011-05-17]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21822D.pdf

[27] HRBÁČEK, Jiří. *Moderní učebnice programování PIC : 2. díl*. Praha : Technická literatura BEN, 2007. 144 p. ISBN 978-80-7300-137-7

[28] WILMSHURST, Tim. *Designing Embedded Systems with PIC Microcontrollers: Second Edition: Principles and Applications*. Oxford, UK: Elsevier Ltd., 2010. 662 p. ISBN 978-1-85617-750-4

[29] IBRAHIM, Dogan. *Advanced PIC microcontroller projects in C: from USB to ZIGBEE with the PIC18F Series*. Oxford, UK: Elsevier Ltd., 2008. 545 p. ISBN 978-0-7506-8611-2

[30] *Energizer CR2032* [online]. 2011 [cit. 2011-05-17]. Available: http://data.energizer.com/PDFs/cr2032.pdf

10 List of acronyms

ASCII – American Standard Code for Information Interchange, character encoding scheme based on English alphabet

C – Programming language C

EEPROM – Electronically Erasable Read Only Memory

ETX – End of text, ASCII control byte (0x03)

FEEC, BUT – Faculty of electrical engineering and communication, Brno University of Technology

GIE – Global Interrupt Enable

GND – ground node

ICSP – In-circuit serial programming

IDE – Integrated Development Environment

ISR – Interrupt service routine

LDO – Low drop-out regulator

LED – Light emitting diode

MCU – Microcontroller unit

MCC18 – MPLAB® C Compiler for PIC18 MCU

MSDNAA – Microsoft Developer Network Academic Alliance

PCB – Printed circuit board

SPI – Serial Peripheral Interface

STX – Start of text, ASCII control character (0x02)

USB – Universal Serial Bus

11 Appendices

Appendix A – Full electric schema

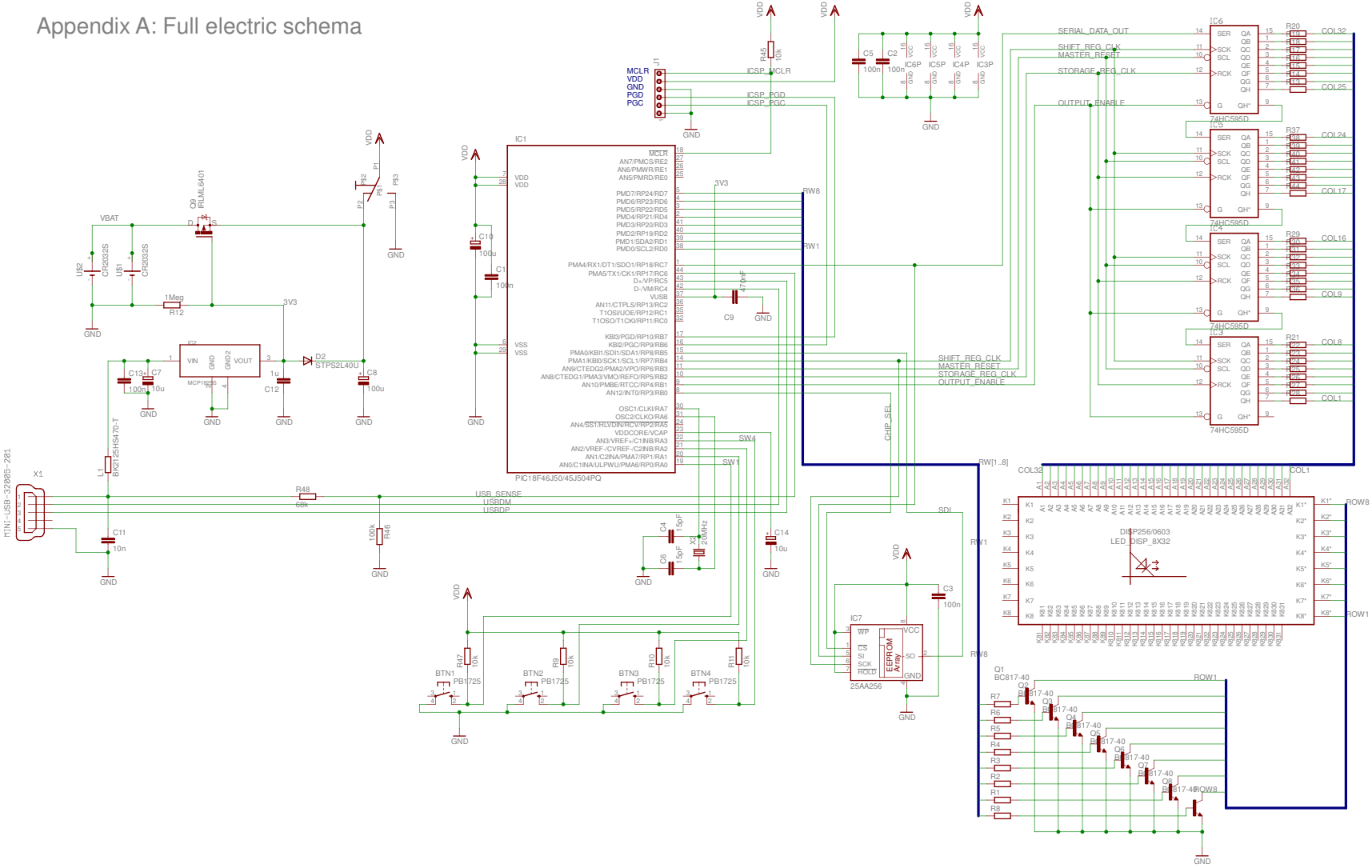
Appendix B – List of components

Appendix C – Printed circuit board

Appendix D – Component assembly

Appendix E – Photographs

Appendix A: Full electric schema



Appendix B – List of components

Part	Value	Device	Package
BTN1 – BTN4		PB1725	PB1725
C1 – C3, C5, C13	100 nF		SMD 1206
C4, C6	15 pF		SMD 1206
C7	10 µF		EIA 3528-21
C8, C10	100 µF		EIA 3528-21
C9	470 nF		SMD 1206
C11	10 nF		SMD 1206
C12	1 µF		SMD 1206
C14	10 µF		EIA 3216-18
D2		STPS2L40U	SMB
IC1		PIC18F46J50	MQFP44
IC2		MCP1825S	SOT223
IC3 – IC6		74HC595D	SO16
IC7		25AA256	SO08
J1		MTA06-100	10X06MTA
L1		BK2125HS470-T	SMD 0805
LED_DISP_8X32		DISP256/0603	
Q1 – Q8		BC817-40	SOT23
Q9		IRLML6401	MICRO3
R1 – R8	1 kΩ		SMD 1206
R9 – R11	10 kΩ		SMD 1206
R12	1 MΩ		SMD 1206
R13 – R44	47 Ω		SMD 1206
R45, R47	10 kΩ		SMD 1206
R46	100 kΩ		SMD 1206
R48	68 kΩ		SMD 1206
SW1		ESP2010	ESP2010
U$1, U$2		CR2032S	CR2032S
X1		MINI-USB-32005-201	32005-201
X2		Q 20.000MHZ	SMD

Appendix C – Printed circuit board

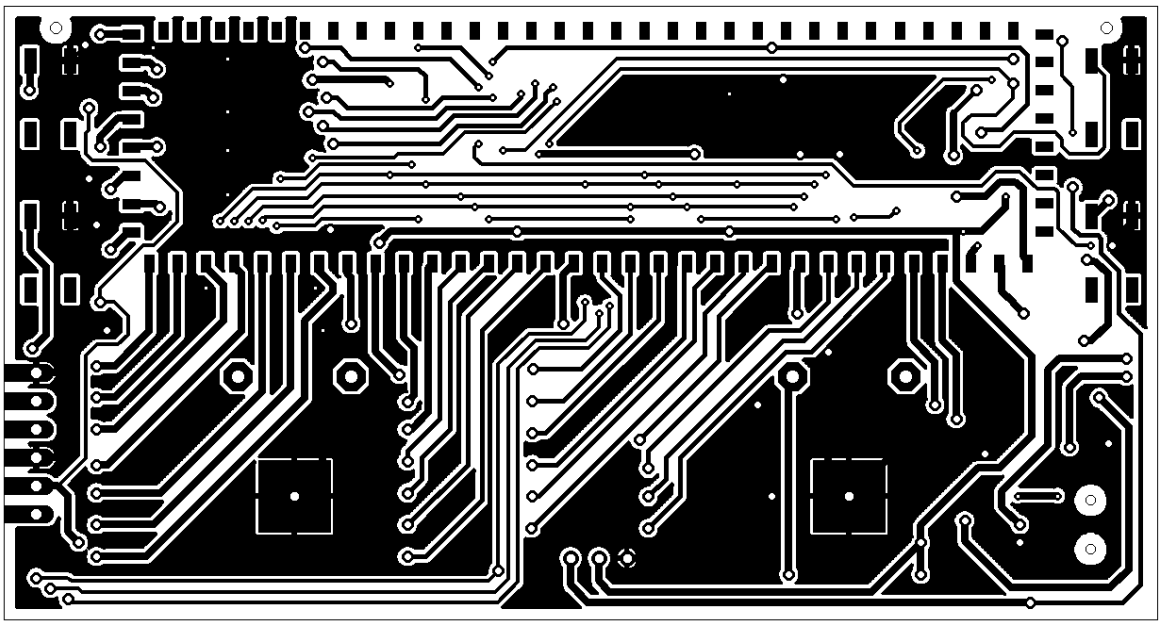


Figure C1: Top conductive layer (dimensions are 103.5 × 55.2 mm)

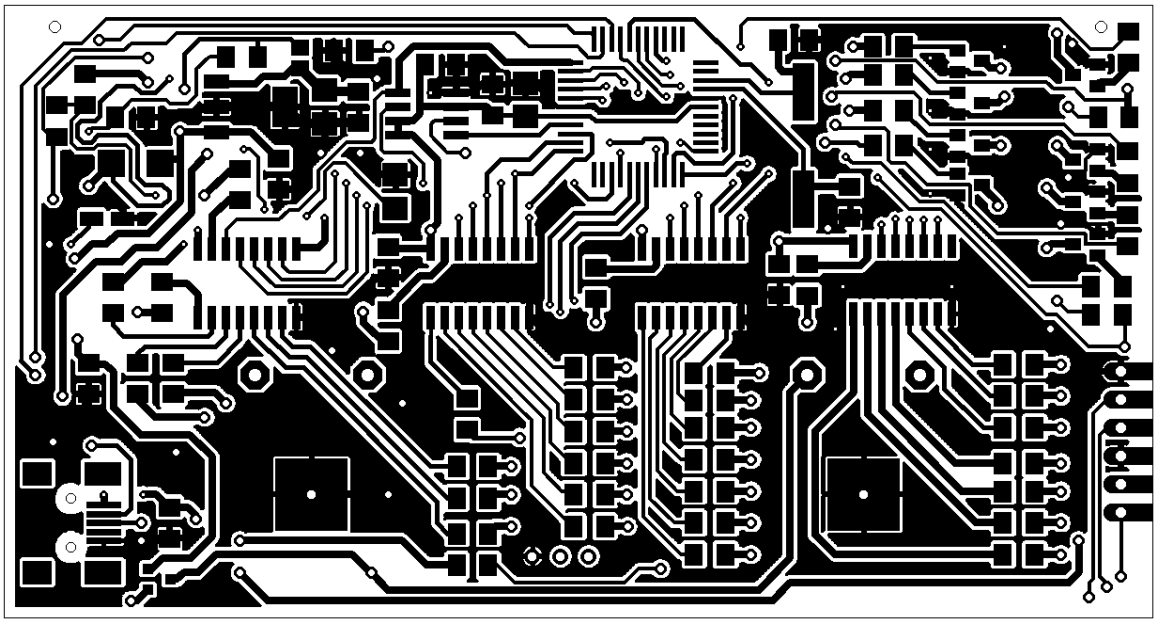


Figure C2: Bottom conductive layer

Appendix D – Component assembly

Figure D1: Top component assembly (dimensions are 103.5 × 55.2 mm)

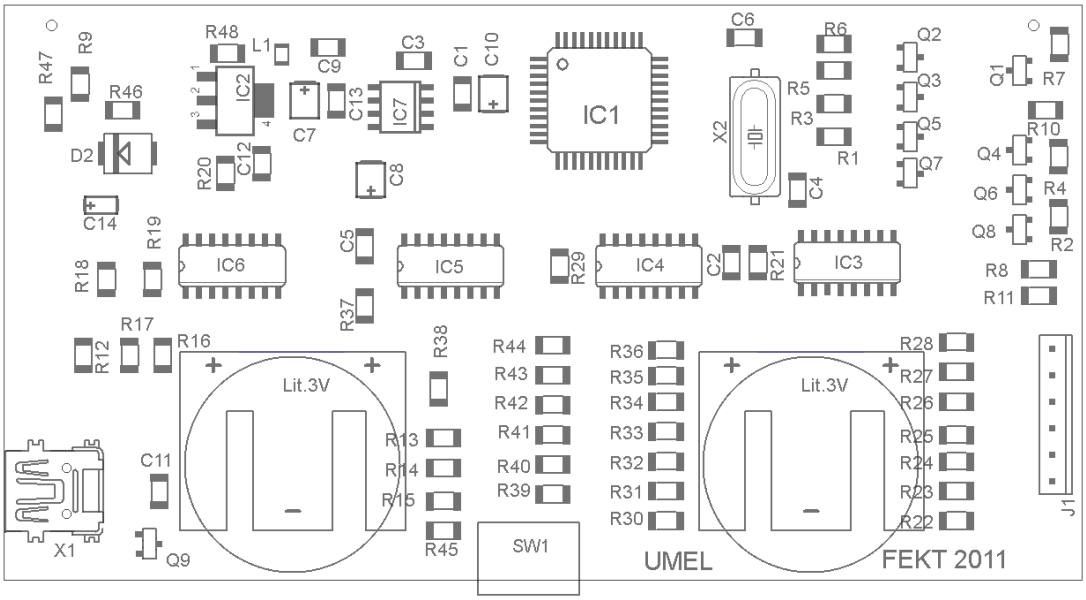


Figure D2: Bottom component assembly

Appendix E – Photographs

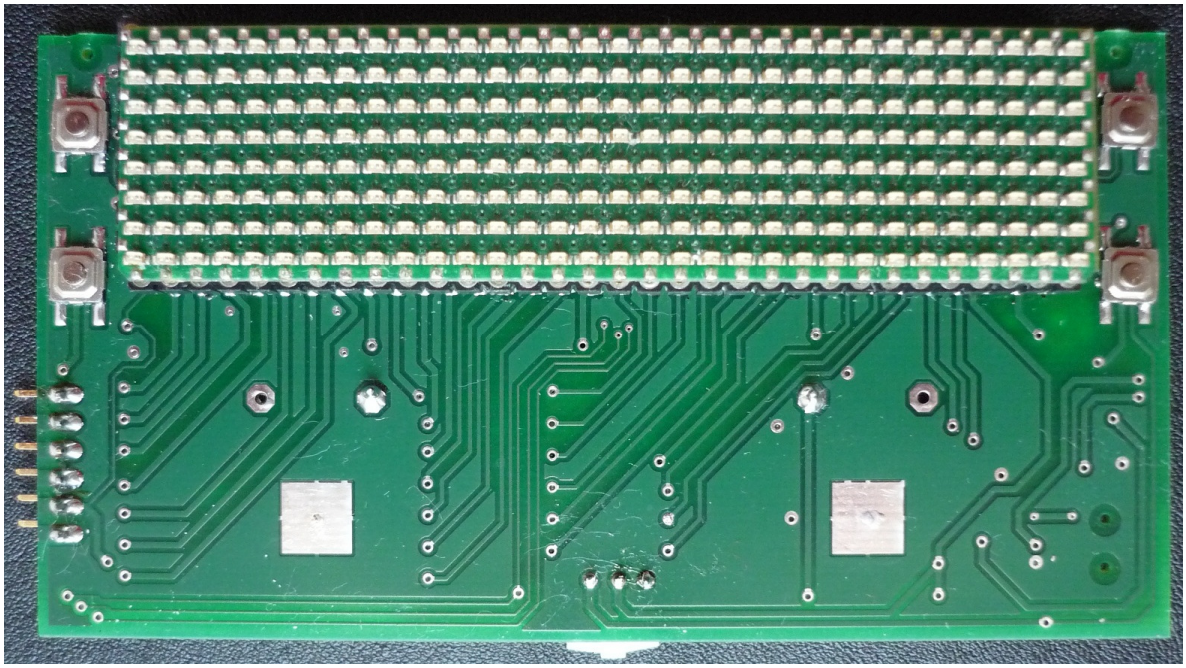


Figure E1: Top side of the device (dimensions are 103.5 × 55.2 mm)

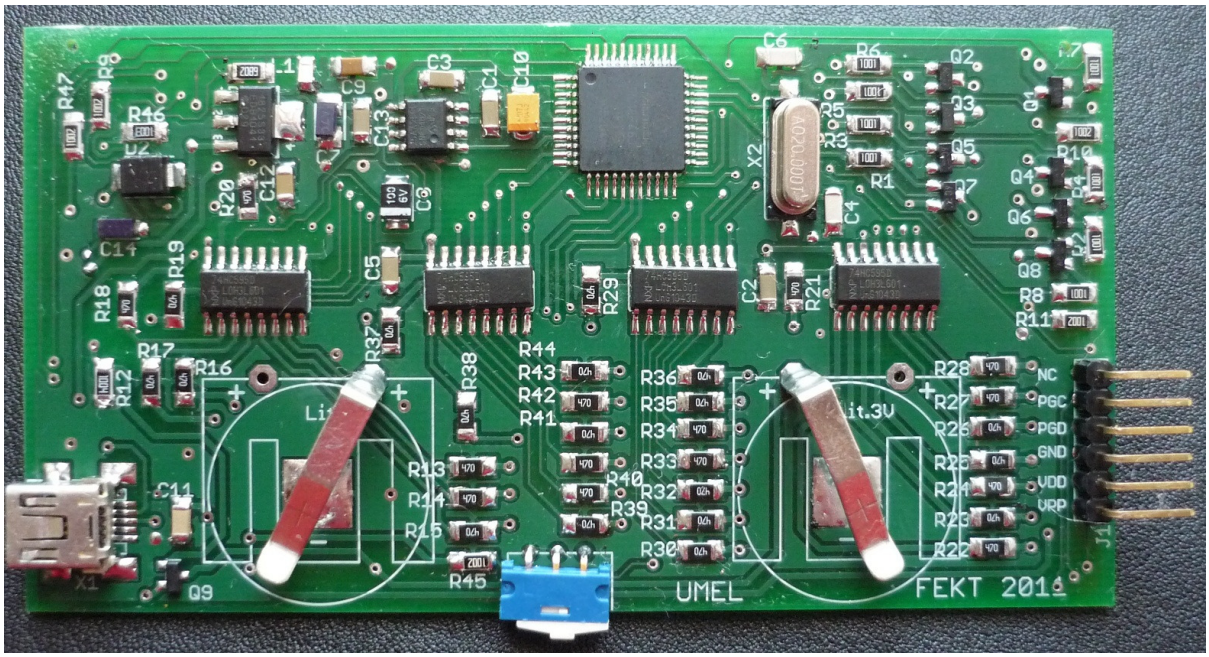


Figure E2: Bottom side of the device