Lodz University of Technology
Institute of Information Technology

ENGINEER'S THESIS

Weather monitoring station constructed over the cell phone

(Unofficial translation into English by the author)

(Last update: 23 November 2026)

Faculty of Technical Physics, Information Technology and Applied Mathematics

Supervisor: Michał Morawski, PhD
Candidate: Piotr Marcińczyk
Album no: 180633
Field: Computer Science
Speciality: Software Engineering and Data Analysis

Łódź, 22 March 2016

Institute of Information Technology
90-924 Łódź, ul. Wólczańska 215, building B9
tel. (42) 631-27-97, (42) 632-97-57, fax (42) 630-34-14, e-mail: office@ics.p.lodz.pl

Contents

Nomenclature **4**

Abstract **5**

1. Introduction **6**
1.1. Background 6
1.2. Aims and scope of the thesis 7

2. Hardware and tools **9**
2.1. Microcontroller 9
2.2. Programming environment 11
2.3. Real-time operating system 12
2.4. Hardware abstraction layer 14
2.5. Weather sensors 14
2.6. Phone for communication 16

3. Practical implementation **19**
3.1. Extension board 19
3.2. Peripheral initialization 20
3.3. Communication with the sensors 21
3.3.1. Humidity sensor AM2302 21
3.3.2. Pressure sensor BMP180 22
3.4. Communication with the phone 23
3.4.1. Protocol 24
3.4.2. SMS text messages 26
3.5. Scheduled notifications 29
3.6. Text terminal and settings 30
3.7. Power management 30

4. User manual **32**
4.1. Start-up 32
4.2. Configuration and diagnostics 32
4.3. SMS messages with the current status 33

5. Conclusions and evaluation **35**

Bibliography **37**

List of Figures **39**

Nomenclature

3GPP	3rd Generation Partnership Project
AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
ARM	Advanced RISC Machine
ASCII	American Standard Code for Information Interchange
BCD	Binary-Coded Decimal
CPU	Central Processing Unit
CTS	Clear To Send
DMA	Direct Memory Access
GMT	Greenwich Mean Time
GSM	Global System for Mobile Communications
IAP	In Application Programming
IDE	Integrated Development Environment
IRA	International Reference Alphabet
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
I²C	Inter-Integrated Circuit
JTAG	Joint Test Action Group
LDO	Low-Dropout Regulator
LED	Light-Emitting Diode
MCO	Microcontroller Clock Output
MCU	Microcontroller unit
OTG	USB On-The-Go
PCB	Printed Circuit Board
PDU	Protocol Data Unit
PLL	Phase-Locked Loop
RAM	Random-Access Memory
RCC	Reset and clock control
RTC	Real Time Clock
RTOS	Real-time operating system
RTS	Ready To Send
SMS	Short Message Service
SPI	Serial Peripheral Interface
SWD	Serial Wire Debug
THT	Through-Hole Technology
UKE	Urząd Komunikacji Elektronicznej (Office of Electronic Communications)
URC	Unsolicited Result Code
USART	Universal Synchronous and Asynchronous Receiver and Transmitter
USB	Universal Serial Bus
VCO	Voltage-Controlled Oscillator

Abstract

The following engineering thesis describes realization of a weather monitoring station, communicating with the user over a cellular network. The embedded system uses **STMicroelectronics STM32F401RE** microcontroller, placed on **NUCLEO-F401RE** development board. The software has been written in C programming language, using **Ac6 System Workbench for STM32** integrated environment. **FreeRTOS** serves as a real-time operating system. Measurements are done by two sensors: **Aosong AM2302** hygrometer and **Bosch BMP180** barometer, both having built-in thermometers. Data is transferred in SMS messages, sent from **Sony Ericsson K300i** mobile phone, connected to the station with a modified **DCU-11** service cable. Communication is based on an asynchronous serial protocol using Hayes AT command set. The values measured by the station can be retrieved on demand or cyclically after configured interval. Settings can be changed using a text terminal connected to the virtual serial port over an USB cable. Created system could be used for monitoring meteorological conditions of a greenhouse or a garden. Construction of the station allows for its simple extending, adding new functionalities or using implemented way of communication in another systems.

Chapter 1

Introduction

1.1. Background

Rapid technological progress means that electronic devices offer ever greater technical and functional capabilities while becoming smaller at the same time. This is one of the reasons why so many people own a mobile phone. These are often highly advanced devices, with specifications comparable to those of personal computers. This is also one of the reasons why mobile connectivity is chosen for communicating with remote devices. Examples include alarm systems, road traffic measurement points and weather stations. Connecting to the base stations of a cellular network requires a **GSM** modem. Most such modems can send short text messages (**SMS**), make voice calls and exchange data with the Internet. Prices of new modems remain high, which may discourage the use of this technology in a project. The cost of the whole system can, however, be reduced by using an older-generation mobile phone that still works but is no longer in use – for example because its functionality is no longer sufficient for its owner. Such phones are easy to find on the second-hand market, and they give the system remote communication at a low cost.

Moreover, not all of the phone's functions are needed to use it this way. Devices with broken displays, speakers or keypads are therefore still useful in projects of this kind. Communication with the modem built into the phone is usually possible through a cable, infrared or Bluetooth. This widens the choice of devices that can

stand in for an expensive wireless cellular modem even further.

1.2. Aims and scope of the thesis

The aim of this thesis is to build a simple weather station as an example of a system that uses the kind of remote communication described in the previous section. The station will measure basic atmospheric parameters: temperature, relative humidity and air pressure. The data should then be sent as a short text message (**SMS**) to another device, usually a mobile phone. This means that the connection uses a wireless cellular network. Such a solution makes the system reachable wherever there is **GSM** coverage, that is practically anywhere in Poland and in much of the world[(1)]. As mentioned in the previous section, an attractive option is to use an older-generation phone as the device that provides the modem needed for this kind of communication.

The main control element of the proposed system will be a microcontroller, which collects data from the weather sensors and communicates with the connected mobile phone over a serial link. It will also process incoming commands, and it should detect any errors that occur and inform the user about them. The microcontroller software should be as universal as possible in terms of functionality, i.e. it should be easy to connect it to a system other than a weather station (e.g. an alarm system or a set of relays).

The thesis also covers the theory needed to build such a system, as well as an analysis of how well the chosen solutions work, including their resilience to various kinds of errors and exceptional situations. Possible alternative applications of the system and ways of developing it further are also proposed. It is also worth emphasizing the positive environmental impact of reusing devices that would otherwise be scrapped.

Chapter 2 compares the hardware and software needed to build a system of this type and explains the choice of tools for this thesis. Chapter 3 describes in detail

[(1)] Based on UKE coverage map `http://btsearch.pl/` and Mobile World Live `http://maps.mobileworldlive.com/network.php?cid=30&cname=Poland` (accessed 16.02.2016)

how the practical part of the system has been implemented. The next one, Chapter 4, contains a basic user manual, i.e. operating instructions for the completed system. The final part of the thesis presents the effectiveness of the solution and observations on the performance of the system, together with an evaluation of how well it reacts to errors.

Chapter 2

Hardware and tools

2.1. Microcontroller

Rapid progress in manufacturing technology and the wide use of these chips – and, as a result, high demand – have made 32-bit ARM processors a very attractive alternative to 8-bit devices. Their prices are comparable, while the capabilities of the former are much greater. Moreover, such a microcontroller (MCU) makes it easier to extend the project later and allows a real-time operating system (RTOS) to be used. This approach makes it easy to manage the tasks assigned to the system, to add new ones efficiently and to set their priorities.

The controller chosen for the system developed in this thesis is the **STMicroelectronics STM32F401RE**. It is an ARM Cortex-M4 processor with the following parameters [1]:

- CPU clock up to 84 MHz
- 512 kB of non-volatile FLASH memory
- 96 kB of volatile Static RAM
- supply voltage from 1.7 to 3.6 V
- 11 timers + system timer
- communication interfaces: USART, I²C, SPI
- USB 2.0 full-speed with OTG
- real-time clock (RTC) with battery backup

- 16-channel DMA controller
- debug interfaces: SWD and JTAG.

The chip is made only in surface-mount packages, which makes it hard to solder without specialized equipment. For the prototype, a ready-made development board offered by the chip manufacturer can therefore be used. The most popular kits are **DISCOVERY** and **NUCLEO**. The former comes with many components, such as buttons, LEDs, an accelerometer, a gyroscope, a display, a microphone and an audio amplifier. The latter is more universal – the board carries only the components needed to run the kit, including the programmer and headers for custom components and extension boards, which are also compatible with the **Arduino** platform. Since most of the components of the **DISCOVERY** kits would not be used in the planned system, the **NUCLEO-F401RE** development board, shown in Figure 2.1, was chosen in the end.

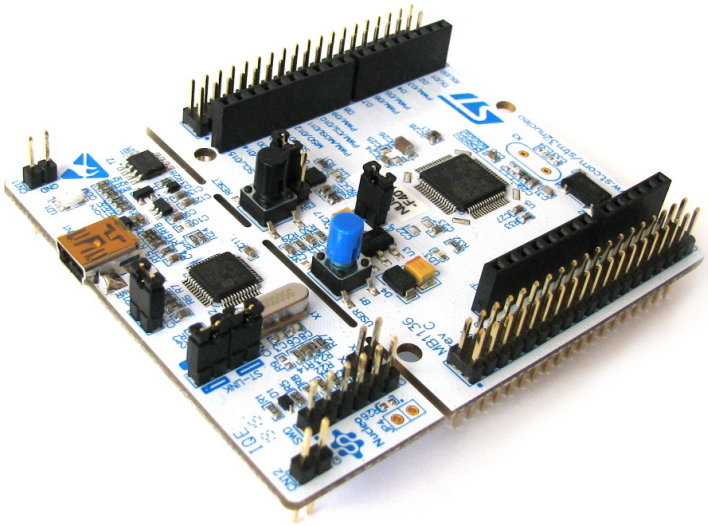

Fig. 2.1: The NUCLEO-F401RE development board

The NUCLEO kit has an attached **ST-LINK/V2** programmer, which can be separated by cutting the PCB at the marked place [3]. This makes it easy to reuse as a universal programmer for future projects. The programmer has an 8 MHz crystal resonator soldered on, which is also connected to the pins of the main MCU. A mini-USB socket is provided for connecting a computer and for power. The pro-

grammer has a built-in UART↔USB converter, which lets the debugger and serial communication with the chip work at the same time. By default, the converter's communication lines are connected internally on the board, so no manual changes are needed for this.

The NUCLEO board has two buttons, black and blue, labelled **RESET** and **USER** respectively. The first one resets the main microcontroller, which restarts the program stored in its memory. The second one is connected to the `PC13` pin of the controller and can be programmed by the user.

On both sides of the board, the microcontroller pins are brought out on goldpin headers, so that any components can be connected to them. They can be reached from both the top and the bottom.

2.2. Programming environment

There are many integrated development environments (IDE) on the market for writing software for ARM processors. A good number of them are paid, or offer free versions limited in time and in the maximum size of the compiled code. Examples of such environments are:

- **IAR Embedded Workbench for ARM**[(1)];
- **Keil MDK-ARM**[(2)];

Their main drawback is that they cannot compile code for the PC platform. This in turn makes it impossible to test custom helper functions that do not depend on the architecture and could be run locally, without the microcontroller.

An alternative to commercial solutions are open-source tools, which are therefore completely free. Most of them are so general that a lot of configuration is needed to get a fully working environment. This can be an obstacle, as it takes a considerable amount of time before software development can even begin. There

[(1)] IAR Embedded Workbench for ARM, `https://www.iar.com/iar-embedded-workbench/` (accessed 06.09.2018)
[(2)] Keil MDK-ARM, `http://www2.keil.com/mdk5` (accessed 06.09.2018)

are also solutions already configured for a specific platform, such as **Ac6 System Workbench for STM32**, chosen for this thesis and available on the **OpenSTM32 Community**[3] platform. It is based on the **Eclipse**[4] environment, so it runs on practically every popular operating system. It also offers many plugins written by the community, which let programmers adapt the environment to their needs. Code is compiled with the **GNU ARM Toolchain** for the `arm-none-eabi` architecture. The programming language used in this thesis is **C**, in the `gnu99` standard.

In addition, the **OpenOCD** software is used to communicate with the programmer. It can not only upload compiled programs, but also run them in debug mode, i.e. execute a program instruction by instruction, with a live view of all variables and processor registers and the ability to modify them manually. This makes it much easier to find bugs and to check that the programmed functions work correctly.

2.3. Real-time operating system

In small projects, an infinite loop in the main function of the program is often sufficient. Such a program continuously checks the state of the peripherals and takes the appropriate action. Interrupts can be used in addition: they are triggered immediately after a state change, which helps to meet the time limits required by the system.

The problem appears as more and more functions are required of the embedded system. The main loop of the program becomes less readable, and every new task makes the code harder to manage. The order of operations in the program becomes hard to predict, which increases the risk of mistakes. This is where a real-time operating system helps: among other things, it switches between tasks while keeping to the required time limits. There are countless such systems, including commercial

[3] OpenSTM32 Community, *System Workbench for STM32*, `http://www.openstm32.org/System+Workbench+for+STM32` (accessed 06.09.2018)

[4] *The Eclipse Foundation open source community website*, `https://eclipse.org/` (accessed 17.02.2016)

ones. Examples include: **BeRTOS**[5], **DDC-I HeartOS**[6], **FreeRTOS**[7], **SafeRTOS**[8] and **OpenRTOS**. The last two are paid derivatives of **FreeRTOS**, developed by **WITTENSTEIN** and supplied with reliability certification.

FreeRTOS V8.2.3 was chosen for this thesis because of its popularity, clear documentation and steady development – and, as a result, a low probability of faults. It allows a separate task to be created for each function, with a priority that sets its importance. A lower-priority task runs only when the more important one is sleeping or blocked (e.g. on a semaphore). **FreeRTOS** offers many configuration options in the `FreeRTOSConfig.h` header. The most important ones include:

- `USE_PREEMPTION` – preemptive switching of the processor context;
- `TICK_RATE_HZ` – frequency of the system kernel tick, i.e. of the interrupt in which preemption may take place;
- `LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY` – the highest priority of an interrupt that may call system functions – interrupts with a higher priority are independent of the system.

Apart from the headers, the system kernel needs only a few source files:

- `tasks.c` – task management and allocation of the processor context;
- `list.c` – list implementation (used, among others, by the scheduler);
- `queue.c` – implementation of synchronized queues;
- `port.c` and `portmacro.h` – all definitions that depend on the processor architecture and the compiler;
- `MemMang/heap_x.c` – one of five available heap managers, which allocate and free memory.

[5] `http://www.bertos.org/`
[6] `http://www.ddci.com/products_heartos.php`
[7] `http://www.freertos.org/`
[8] `http://www.highintegritysystems.com/safertos/`

2.4. Hardware abstraction layer

The microcontroller peripherals are managed with the **Standard Peripheral Library V1.5.1** (**StdPeriph** for short), supplied by **STMicroelectronics**. The differences between the versions for particular processor families are small, which is why the cited documentation [4] refers to **STM32F3xxx**. The library provides a map of the processor registers, including bit-level configuration values, and a set of functions that abstract away the hardware, so peripherals can be controlled without knowing the specific registers. This in turn makes the code more readable, makes changes easier and allows programs to be moved between different processor models from this manufacturer, minimizing the risk of errors caused by differences in architecture.

2.5. Weather sensors

The sensors used for the meteorological measurements are the **Aosong AM2302** and the **Bosch BMP180**. The first one is a digital capacitive hygrometer, which measures the relative humidity of the air. It is shown in Figure 2.2. It has a built-in thermometer, used for temperature compensation of the measurements, which can also be read out. The sensor belongs to the **DHT-22** family and is also known as the **RHT03**.

The operating range of the sensor, taken from its documentation [5], is given in Table 2.1. The sensor is powered with 3.3 to 5 V, so it can be connected to the board directly.

Parameter	Range	Accuracy	Resolution
humidity [%]	from **0.0** to **99.9**	±2	±0.1
temperature [°C]	from **-40** to **80**	±0.5	±0.1

Tab. 2.1: AM2302 humidity sensor parameters

The **AM2302** sensor communicates with the microcontroller over a single data line. The protocol partly resembles the **Maxim-Dallas 1-Wire** interface [7], but it does not support addressing of the connected devices, so every device of this type

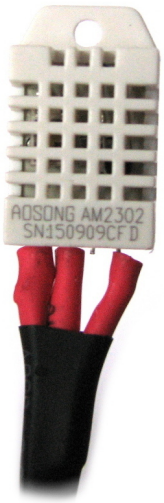

Fig. 2.2: Aosong AM2302 humidity sensor

needs its own communication line. Moreover, the individual timings required by the protocol differ somewhat between the datasheets [5, 6] of the sensors of the **DHT-22** family. A margin of error of a few percent has therefore been allowed in the implementation (it also covers the inaccuracy of the oscillators of the individual devices).

The second sensor, the **Bosch BMP180** piezoresistive pressure sensor, is mounted on a **SparkFun** board with pins for through-hole mounting (THT). The sensor itself is made only in small *Land Grid Array* packages, which are hard to solder without special equipment; it can be seen in the middle of the board in Figure 2.3.

According to the datasheet [8], the sensor's readings lie in the ranges given in Table 2.2. The sensor communicates over the standard I²C protocol, with a clock frequency of up to 400 kHz.

Parameter	**Range**	**Accuracy**	**Resolution**
pressure [hPa]	from **300** to **1100**	±0.12	±0.01
temperature [°C]	from **-40** to **85**	±0.5	±0.1

Tab. 2.2: BMP180 air pressure sensor parameters

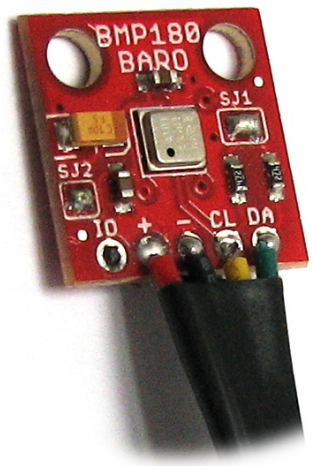

Fig. 2.3: BMP180 air pressure sensor on a SparkFun board

2.6. Phone for communication

When choosing a phone for remote connection with the system, one should make sure that the protocol it uses for communication is well documented and that the phone can be connected easily, with little intervention. Manufacturers often design their own means of communication and do not publish the necessary specifications, which makes connecting to the phone difficult. Examples of such protocols are **MBUS** and **FBUS**, designed by **Nokia**. Both are based on asynchronous serial data transmission. They differ mainly in speed and directionality: 9600 bps half-duplex and 115200 bps full-duplex, respectively. Every message is sent as a binary frame. More information can be found on the website[9] of the open-source **Gnokii** program, which lets a phone communicate with a computer using these protocols and includes their detailed specification.

A much more widespread standard is the **Hayes AT** command set (AT from *attention*), developed by Dennis Hayes for analog telephone modems[10]. It is still being developed and is used by many modems (including GSM ones). The communication is largely text-based, and commands can be sent both by a machine and

[9] Gnokii project, `https://gnokii.org/` (accessed 18.09.2018)
[10] History of computers: *Modem*, `http://history-computer.com/ModernComputer/Basis/modem.html` (accessed 18.09.2018)

manually by a human. This in turn makes it easy to analyze the course of a transmission and to detect errors quickly. The standard was originally approved by the ITU Telecommunication Standardization Sector (**ITU-T**) [9]; today the specification and new commands are maintained by the **3GPP** group [10]. The most popular phones that support this protocol include some Nokia models (e.g. 5140, 6020, 6310, 6510) and **Sony Ericsson** phones (those made before 2005 allow UART-compatible communication, while newer ones require the USB interface to be implemented). The full list of compatible devices can be found in the specification [11].

The embedded system built in this thesis uses a Sony Ericsson K300i phone, shown in Figure 2.4. Both communication and power go through the connector at the bottom of the phone, so the connection works without any further modification of the phone. In addition, to ensure better compatibility with the various devices that may be connected to the station, tests were also run with a second phone, the Sony Ericsson K700i, shown in Figure 2.5.



Fig. 2.4: Main phone – Sony Ericsson K300i



Fig. 2.5: Additional phone – Sony Ericsson K700i

Chapter 3

Practical implementation

3.1. Extension board

An extension board, shown in Figure 3.1, has been built for this thesis. It plugs into the bottom of the NUCLEO headers through two double female goldpin strips. It has three connectors: for the weather sensors and for the phone used for remote communication. This makes it easy to swap individual modules and leaves room for extending the system further without modifying the original board.

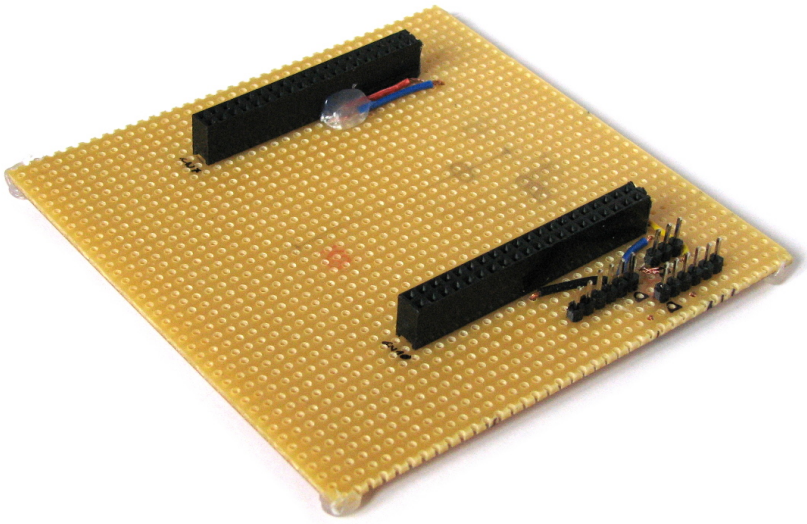

Fig. 3.1: Custom extension board with NUCLEO sockets

3.2. Peripheral initialization

The system is clocked by the **MCO (Microcontroller Clock Output)** oscillator signal coming from the **ST-LINK** programmer on the NUCLEO board. Its frequency is $f_{\text{in}} = 8$ MHz, and it originally comes from the `X1` crystal resonator [3]. Using the PLL built into the microcontroller, the system clock has been set to $f_{\text{out}} = 64$ MHz. The configuration of the registers this requires is in `system_stm32f4xx.c`, the main source file of the **StdPeriph** library (described in Section 2.4). The relationships given in the reference manual [2] are shown in equations (3.1) and (3.2):

$$f_{\text{VCO}} = f_{\text{in}} \cdot \frac{\texttt{PLL_N}}{\texttt{PLL_M}} \tag{3.1}$$

$$f_{\text{out}} = \frac{f_{\text{VCO}}}{\texttt{PLL_P}} = 64 \text{ MHz} \tag{3.2}$$

In addition, the frequencies and parameters have to stay within the ranges given in formulas (3.3) to (3.7):

$$192 \text{ MHz} \leqslant f_{\text{VCO}} \leqslant 432 \text{ MHz} \tag{3.3}$$

$$1 \text{ MHz} \leqslant \frac{f_{\text{in}}}{\texttt{PLL_M}} \leqslant 2 \text{ MHz} \tag{3.4}$$

$$192 \leqslant \texttt{PLL_N} \leqslant 432 \tag{3.5}$$

$$2 \leqslant \texttt{PLL_M} \leqslant 63 \tag{3.6}$$

$$\texttt{PLL_P} \in \{2; 4; 6; 8\} \tag{3.7}$$

The recommended frequency in (3.4) is 2 MHz, which makes the PLL, and with it the whole system, more stable. From the above conditions, the best parameter configuration can be determined; it is shown in Table 3.1.

Parameter	Value
PLL_M	4
PLL_N	192
PLL_P	6

Tab. 3.1: Selected PLL register values

The **STM32F401** microcontroller has several main buses: **AHB**, **APB1** (low speed) and **APB2** (high speed). Most peripherals communicate with each other over one of these three buses. Right after the microcontroller starts, the peripherals are disconnected from the buses to save energy. The selected peripherals therefore have to be enabled in software, by setting registers of the **RCC** group (Reset and clock control). Depending on the bus of the selected module, this is done with the function `RCC_AxBnPeriphClockCmd()`, where `x` is `H` or `P`, and `n` is a number from `1` to `3`.

3.3. Communication with the sensors

Both weather sensors are powered with 3.3 V from the low-dropout linear regulator (**LDO**) on the NUCLEO board, labelled `U4` [3]. The regulator output is marked `+3V3` in the schematic.

3.3.1. Humidity sensor AM2302

The sensor has a single communication line, pulled up to the supply voltage by default. To start communication, the microcontroller pulls the line down to ground for at least 1 ms and then releases it back to the supply voltage. After a time T_{go}, the sensor responds by holding the line low for T_{rel} and then high for T_{reh}. Next, 5 octets of data are sent (most significant bit and byte first): two for humidity, two for temperature and one checksum, i.e. the 8 lowest bits of the sum of the first 4 data bytes. The bits differ in the duration of the high state, $T_{\mathrm{H}x}$, while the low state always lasts T_{L}. The communication ends with the line held low for T_{end}. The allowed durations in microseconds (µs), according to [5, 6] and including the oscillator margin of error, are listed in Table 3.2.

The microcontroller's peripherals do not support the sensor's protocol directly in hardware. A USART module could be used to communicate in the 1-Wire standard[(1)], but a software implementation using a **GPIO** pin is sufficient in this case.

[(1)] Maxim Integrated: *Using a UART to Implement a 1-Wire Bus Master*, `https://www.maximintegrated.com/en/app-notes/index.mvp/id/214` (accessed 23.08.2020)

Time	Min	Max
T_{go}	0	200
T_{rel}	70	90
T_{reh}	70	90
T_{L}	40	70
T_{H0}	20	45
T_{H1}	50	80
T_{end}	40	60

Tab. 3.2: Allowed durations in µs for the AM2302 sensor protocol

The sensor's communication line is connected to pin `PC9` of the microcontroller. The pin is configured as an input with the internal pull-up resistor enabled, and with a low level in output mode, so that changing its direction alone produces the correct signal levels. The time taken by each stage of the protocol is measured with the 16-bit timer `TIM10`, with its prescaler set to $f_{\text{out}}/1000000 = 64$. This gives a clock period of approximately 1 µs.

The data received from the sensor is in a fixed-point, sign-magnitude format. To obtain the humidity or temperature value, it has to be divided by 10. However, to speed up processing, the original readings are stored as they are, and a custom procedure named `printx()` has been implemented to convert them into text. It takes a pointer to a function that outputs a single character (e.g. to the USART transmitter or to a text buffer), which makes it more universal than the standard `printf()`. Fixed-point numbers are printed with the "`%.#b`" flag, where "`#`" specifies the number of decimal places.

3.3.2. Pressure sensor BMP180

The sensor is read using the I²C module. The communication lines have been connected to the pins labelled `I2C1`: `PB8` (SCL) and `PB9` (SDA), and then configured in the alternate function mode (AF, *Alternate Function*). The bus clock has been set to 100 kHz to make the communication less prone to errors. The sensor address is `0xEE` (`0xEF` for reading) [8]. Sensors of this type have an internal **EEPROM** memory whose content differs from unit to unit: it holds unique calibration data needed to calculate the exact measured value. This data is read right after the system starts

and then stored in the microcontroller's RAM. The sensor starts a measurement after a defined value is written to the register at address `0xF4`. For temperature it is `0x2E`, while for pressure it depends on the required precision of the result. The system uses the standard mode with identifier `0x74`, in which a measurement takes 7.5 ms and averages two samples. The temperature is stored in a 16-bit register, multiplied by 10. The pressure, on the other hand, is given as an integer value in pascals (Pa) and has to be divided by 100 to obtain hPa, the unit more commonly used to describe the weather. These operations are performed only during the conversion to text, using the `printx()` procedure.

3.4. Communication with the phone

The most popular cable used to exchange data between a computer and a Sony Ericsson phone is DCU-11, shown in Figure 3.2. Its plug contains a UART↔USB converter, which allows data to be transferred without a serial port in the computer. The manufacturer often supplies the converter drivers with the cable, so that it can communicate properly. Once the cable is connected, data can be exchanged with the modem built into the phone, for example from a terminal, using a virtual serial port.

The pins of the phone connector are numbered from the left, looking at the front of the phone (the side with the display and the keypad), as shown in Figure 3.3. The function of each pin is explained in Table 3.3[(2)].

To adapt the cable for communication with the embedded system, the USB converter has been removed and the necessary changes have been made to the plug on the phone side. The original wiring – Rx (sending), Tx (receiving), RTS (*Ready To Send*), CTS (*Clear To Send*) and GND (ground) – did not allow an external power source to be connected, and so the phone could not be charged. Because of the limited number of wires, the optional hardware flow control (the RTS and CTS lines)

[(2)] Pinouts.ru: *Sony Ericsson R520, R310, R320, T28, T39, T68, T68i, T610, T630 and others cell phones cable connector pinout*, `http://pinouts.ru/CellularPhones-P-W/erics_t28_pinout.shtml` (accessed 23.02.2016)



Fig. 3.2: Original DCU-11 cable with a USB plug

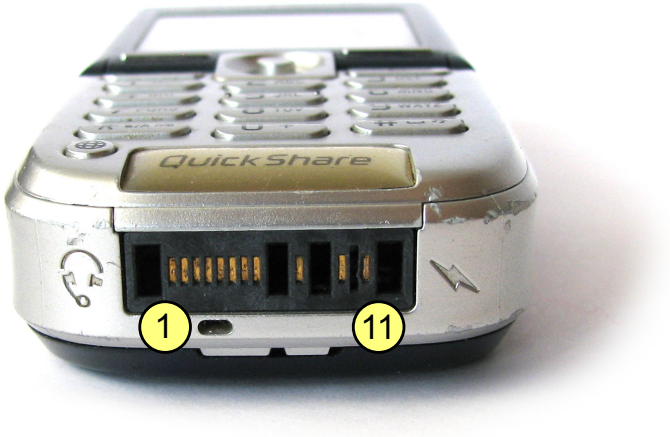

Fig. 3.3: Pin numbering of the Sony Ericsson connector

has been dropped and these wires have been used to supply voltage from the `5V` pin of the NUCLEO board. This approach also frees the pins for an external audio source, which are shared with flow control. The inside of the plug after the changes is shown in Figure 3.4. The red wire has been moved from its original pin 2 (RTS) to pin 11 (Vcc), while the green one has been left unconnected for future use.

3.4.1. Protocol

Communication with the phone is asynchronous and full-duplex, using the signal lines of the `USART6` module: Tx to `PA11` and Rx to `PA12`. The lines are pulled up to 3.3 V. The high state represents binary `0` and the low state binary `1`. The phone detects the transmission speed automatically when transmission starts. Usually it

Pin	Name	Description
1	ATMS	Audio to the phone
2	AFMS/RTS	Audio from the phone or Ready To Send
3	CTS	Clear To Send
4	DTMS (Rx)	Data sent to the phone (*to mobile*)
5	DFMS (Tx)	Data received from the phone (*from mobile*)
6	ACC in	External accessory control
7	ACC out	External accessory control
8	AGND	Audio ground
9	Flash	Service pin
10	DGND	Digital ground
11	Vcc	Phone charging +5 V

Tab. 3.3: Sony Ericsson phone connector pinout

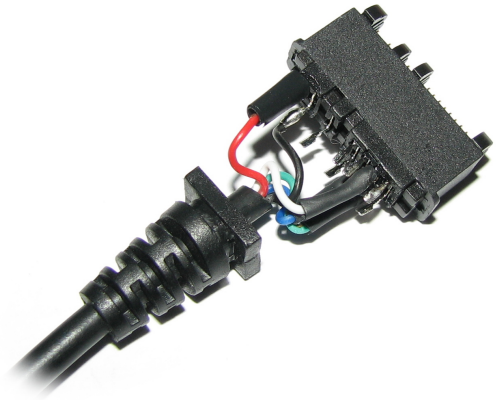

Fig. 3.4: DCU-11 cable plug after the modifications

can be anywhere from 300 to 115200 bps. In the developed system the connection parameters have been set to: 9600 bps, 8 data bits, no parity, 1 stop bit. The communication uses AT commands; the complete list, with details and the parameters supported by Sony Ericsson phones, can be found in the developer specification [11].

Commands start with the characters "`AT`" and are terminated with the carriage return character (CR – *Carriage Return*), whose ASCII/IRA value is `13`. The exception is "`A/`", which repeats the last command and does not need a carriage return. The commands include basic ones, described by the **ITU** standard, and ones defined by modem manufacturers, for example some of those starting with "`AT+C`", which are extensions for mobile telephony [9]. If a command takes parameters, they are

usually given after an equals sign (=) and separated by commas (,). Strings must be enclosed in double quotes ("). Optional parameters can be omitted by giving no argument – the default value is then used.

In case of an error, caused for example by an unsupported command or invalid parameters, the phone's modem returns the code "ERROR" preceded by the newline characters CR and LF (*Line Feed*, value 10). This gives a readable output when communicating through a text terminal. If the requested command is accepted, the phone returns a response ending with "OK".

The connected phone can also send unsolicited result codes (**URC**), i.e. data that is not related to a sent request but caused by an external event, for example an incoming call or a newly received SMS message. The developed system should handle such situations correctly. For this reason, the communication uses the phone's echo option. Every valid character sent to the modem is immediately sent back. This makes it possible to synchronize the output precisely and to decide whether a received response relates directly to the command that was sent.

Communication is handled in a separate task, PhoneTask, and in the interrupt service routine (ISR – *Interrupt Service Routine*) of the USART6 peripheral, which is triggered when a byte is received (RXNE flag) and after one is sent (TXE flag). By default, received responses are treated as URCs and processed once a CRLF sequence arrives. The same applies to data received before the echo of the transmitted command. If the echo is received correctly, the rest of the response is processed until "OK" or "ERROR" is received. If the system receives neither within a few seconds, this is treated as a timeout error. The phone buffers URCs while commands are being transmitted and sends them once it finishes processing the current command. For this reason, after each command the system waits about 150 ms before sending the next one, so that any such responses can be received.

3.4.2. SMS text messages

The AT standard provides two modes of transferring messages: text mode and **PDU** mode (*Protocol Data Unit*). The first one is more convenient for humans, as it contains all the data in a readable format, but fewer modems and phones support

it [12]. The second mode maps the low-level message protocol directly. At the same time it is much more versatile – among other things, it allows binary data to be transferred and even formats not supported by the modem to be used. The phones used in this thesis support only the **PDU** mode. It is a binary standard, and since AT communication should stay readable and easy for a human to type into a terminal, the data is transferred as strings of ASCII characters representing bytes in hexadecimal. The data frame of a typical received SMS message looks as follows [13]:

1) *SMSCA* – address (number) of the SMS Service Center:
   1.1) length of the whole address field in octets *(1 byte)*;
   1.2) address type *(1 byte)*;
   1.3) address value in BCD encoding *(variable size)*.
2) *First Octet* – message type and content *(1 byte)* – the value `0x04` denotes SMS-DELIVER, i.e. a message received from the Service Center;
3) *Originating Address (OA)* – sender's number:
   3.1) address length in digits *(1 byte)*;
   3.2) address type *(1 byte)*;
   3.3) sender's address value *(up to 12 bytes)*.
4) *Protocol Identifier (PID)* – protocol identifier *(1 byte)* – the value `0x00` denotes a regular SMS message;
5) *Data Coding Scheme (DCS)* – data coding scheme *(1 byte)* – the value `0x00` stands for 7-bit GSM encoding;
6) *Time-Stamp (SCTS)* – message time stamp *(7 bytes)* – BCD encoded, in the format (2 digits per field): year, month, day, hour, minute, second, and time zone given as the difference from GMT in quarters of an hour;
7) *User Data (UD)* – data length *(1 byte)* + data *(up to 140 bytes)* – the actual message content, encoded according to the selected DCS scheme.

Phone numbers are formatted according to the ITU-T E.164 standard[3]. The most commonly used address types are `0x91`, an international number starting with

[3] ITU-T: *E.164: The international public telecommunication numbering plan*, 11.2010, `https://www.itu.int/rec/T-REC-E.164/en` (accessed 11.03.2016)

the country code (48 for Poland – the leading plus is omitted), and `0x81`, a national number that works only within the current country. Addresses in a PDU frame are BCD encoded, with the two half-bytes (*nibbles*) of each byte swapped. Moreover, if the number of digits is odd, the number is padded with `0xF`. This means that, for example, the address +48123 is written as "`8421F3`". The time stamp field is encoded in a similar way. The zone value for Central European Time (CET = GMT+01:00), used in Poland in winter, is `0x04` and is represented by the string "`40`". The message content is encoded by default in septets (7-bit packets) using the GSM alphabet, which partly overlaps with the ASCII/IRA standard, in particular in the Latin letters and digits [14]. This allows simple messages to be sent without implementing any encoding conversion. A single message can hold up to 160 characters in 140 bytes.

The frame of a sent message differs in part from that of a received one. The Service Center address field can be omitted by giving the value `0x00` – the phone then uses the number assigned by the operator of the network it is currently connected to:

1) *SMSCA* – address of the SMS Service Center;
2) *First Octet* – message type and validity period format *(1 byte)* – the value `0x11` denotes SMS-SUBMIT, i.e. a message sent to the Service Center, with a relative validity period format;
3) *Message Reference (MR)* – identifier of the sent message, which allows it to be matched with Service Center reports *(1 byte)* – not every phone supports user-assigned identifiers, in which case the value "`0x00`" should be given;
4) *Destination Address (DA)* – recipient's number – same format as in *Originating Address*;
5) *Protocol Identifier (PID)*;
6) *Data Coding Scheme (DCS)*;
7) *Validity Period (VP)* – message validity period *(relative: 1 byte)* [13] – for example the value `143` for 12 hours after sending, `167` for 24;
8) *User Data (UD)*.

To let the system react immediately to incoming SMS messages, the URCs related to new messages are enabled [11, 15]. This is done with the command

"AT+CNMI=,1" (*New Message Indication*). The first parameter, which sets the response buffering method, is omitted, because Sony Ericsson phones support only one default method. The value 1 makes the phone report where the incoming message has been stored: the memory type (phone or SIM card) and the index. For example, the message "+CMTI: "ME",3" reports a new message stored in the phone memory (*Mobile Equipment*) at index 3. To retrieve its content, one has to make sure that the right SMS memory is selected, using the command "AT+CPMS="ME","ME","ME"" (*Preferred Message Storage*). Then "AT+CMGR=3" (*Read Message*) is sent with the index received earlier. The phone returns "+CMGR: 0," followed by the length of the whole message in octets, and then the actual content of the PDU frame in hexadecimal ASCII representation. If the communication has succeeded, the SMS message can be deleted from the phone with "AT+CMGD=3" to keep the memory from filling up.

A message is sent with the command "AT+CMGS=42", where 42 is an example number of PDU octets. The octets follow the carriage return CR, in the same format as when receiving. The frame is confirmed with [Ctrl+Z], i.e. the character with ASCII value 26. While a message is being sent, which takes a dozen or so seconds, the modem stops responding, so the timeout for this command has been increased to 30 s to avoid needless errors. If the SMS message is sent successfully, the phone responds with "OK", otherwise with "ERROR".

3.5. Scheduled notifications

To send the station's measurements at an interval set by the user, the microcontroller's built-in real-time clock **RTC** is used. The current time is retrieved from the connected phone with the command "AT+CCLK?". The response, in the form "+CCLK: "YY/MM/DD,hh:mm:ss±zz"", is parsed and written to the clock registers. The value "±zz" is the time zone, given as the difference from GMT in quarters of an hour, as in the field 6) *Time-Stamp (SCTS)* of the PDU frame described in Section 3.4.2. For the time to be correct, the phone has been set to take the time from the GSM network operator – on the K300i this function is called "Auto time zone". The

microcontroller's RTC itself is clocked by the external 32.768 kHz crystal resonator X2 on the NUCLEO board [3].

The time of the automatic sending is set in the first alarm register (ALRA) of the RTC, according to the interval set by the user. The interrupt triggered when the clock reaches that time sends an SMS message with the station's readings and moves the alarm time forward, so that notifications are sent regularly at the set interval.

3.6. Text terminal and settings

As mentioned in Section 2.1, the programmer on the NUCLEO board can be used as a UART↔USB converter. This allows the system to be connected to a computer with a USB port and to communicate over a virtual serial port. The converter is internally connected to the pins of the USART2 module: PA2 and PA3. Communication is handled by the TerminalTask task, which takes data from a buffer and carries out the operations requested by the user. The console allows the system to be configured, including the operating mode, the associated phone number and the country code. The settings are saved in the microcontroller's non-volatile memory using the In-Application Programming mechanism (IAP – *In Application Programming*). The Flash memory is divided into sectors of different sizes: 4× 16 kB, 1× 64 kB and 3× 128 kB. Data can only be erased one whole sector at a time, so, since there are only a few settings, a 16 kB sector has been chosen. To keep that space free, the linker script (the LinkerScript.ld file) has been modified to place only the bootloader data (e.g. the oscillator configuration and the interrupt vector addresses) in the first sector. The rest of the program has been moved to the 64 kB region, leaving the smallest sectors free for storing settings.

3.7. Power management

The modification of the connection cable described in Section 3.4 made it possible to power the connected phone and charge its battery. Thanks to this, a switched-off phone can be started – the supply voltage puts the phone into charging mode, in

which AT commands can then be sent. Right after the whole system is switched on and the individual peripherals are initialized, the phone's state is checked with the command "AT+CFUN?". The response "+CFUN: 1" means the phone is fully on and ready to communicate with the cellular network. Otherwise, "AT+CFUN=1" is sent, which switches the phone into the required mode.

The battery state is checked periodically with the command "AT+CBC?". The response contains two parameters. The first one gives the power state: "0" means no external power and "1" means the battery is charging. This solution also allows the whole system to run on the phone's battery and a failure of the main power source to be detected.

To save energy, the microcontroller offers 3 sleep modes, in which no instructions are executed. The first one, "Sleep mode", stops the processor core while the peripherals keep running, and waits for one of them to raise an interrupt. It is entered with the "WFI" instruction (*Wait For Interrupt*); waking up takes 4 to 6 processor cycles, after which the corresponding interrupt service routine is executed. The second mode, "Stop mode", also switches off most of the clock lines, the PLLs and the oscillators, which stops most of the peripherals except, among others, the RTC, the RAM and the microcontroller's inputs. The microcontroller's Flash memory can optionally be powered down as well, at the cost of a longer wake-up time. This mode is entered in the same way as "Sleep mode", with the SLEEPDEEP bit of the *System Control* register set in addition. The most economical mode is "Standby mode", which powers down the RAM (except for the registers kept in the *Backup Domain*), so its content is lost. It is entered by setting the PDDS bit. The MCU can be woken up by a reset, a change of state on the WKUP pin (Low-power wakeup) or an RTC alarm.

The developed system uses the first mode, "Sleep mode", which is activated whenever all the operating system tasks are sleeping or blocked. The more economical modes have not been used, because the USART module cannot be kept powered in them. That would prevent an immediate reaction to data sent by the phone, such as a received message or an incoming call.

Chapter 4

User manual

4.1. Start-up

The system is powered with 5 V from a source connected to the mini-USB socket. The source should deliver at least 500 mA so that the connected phone can be powered without problems. Examples of such a source are a computer, a USB charger or a portable battery (power bank). After start-up, the modules on the board are initialized and checked for correct operation.

4.2. Configuration and diagnostics

When the system is connected to a computer or another device that supports the programmer's driver, it can be accessed through a text terminal. Required connection settings: **115200/8-N-1** (115200 bps, 8 data bits, no parity, 1 stop bit). A screen with information about the system and the available configuration options is displayed with the [ESC] key. If the debug output is enabled ([D] key), the user is informed on the fly about changes in the state of the individual system modules and about the communication with the connected phone. The current state of the system, including the sensor readings, is displayed after pressing [S]. An example output is shown in Listing 4.1.

The report shows the battery charge level, the time taken from the phone, the

```
*****************************
 Current status: [OK]
   > Storage: [OK]
   > Phone: [OK]
     > Battery: 100 % (charging)
   > Clock: [OK]
     > 02.03.2016 21:37:13
     > Auto SMS at: 21:45:00
   > Sensors:
     > DHT-22: [OK]
       > T: 22.6 *C
       > RH: 46.6 %
     > BMP180: [OK]
       > T: 22.3 *C
       > P: 983.06 hPa
*****************************
```

Listing 4.1: Example system status report

time of the next SMS notification and the sensor measurements. It also marks each system component as failed ("`[Error]`") or working correctly ("`[OK]`").

The console allows the associated phone number, to which status reports are sent, to be changed ([P] key). When the system is moved to another country, the correct country code should also be set ([C] key); for Poland it is +48. Once adjusted, the settings can be saved to non-volatile memory with the [!] key ([Shift+1]). Otherwise, the changes will be lost after a power loss or a system restart.

4.3. SMS messages with the current status

Once the phone number used for communication has been configured, sending a text message reading "STATUS" from it makes the station reply with an SMS message containing the current readings of the weather sensors. The message is not case-sensitive. Messages encoded other than with the 7-bit GSM alphabet are rejected – they must not contain national characters or Unicode symbols. Messages can also be sent automatically at a set interval. To configure this, connect to the text terminal as described in Section 4.2 and press [A]. The system then asks for the interval in minutes at which the report should be sent. Entering "`0`" turns automatic sending off.

The message contains the current sensor readings and any malfunctions of the system components. An example is shown in Listing 4.2.

```
02.03.2016 21:33:02
Temp: 23.7 *C
RH: 43.7 %
Pres: 982.91 hPa
```

Listing 4.2: Example SMS message with measurements

Note that the reported pressure p also depends on the altitude at which the station is placed. To calculate the sea-level value p_0, the compensation formula (4.1) from the sensor's datasheet [8] can be used. The altitude h is given in meters.

$$p_0 = \frac{p}{\left(1 - \frac{h}{44330}\right)^{5.255}} \tag{4.1}$$

Chapter 5

Conclusions and evaluation

The system created in this engineering thesis is an example of a weather station that can be communicated with remotely over a cellular network. Two sensors are used for the measurements; they allow air temperature, humidity and pressure to be read with an accuracy sufficient for amateur use. Data transfer has been implemented with Hayes AT commands sent to the connected Sony Ericsson phone. Since the station has been tested with two different phone models and the protocol follows the specification published by the manufacturer, it can be expected to work with other phones of this brand as well. Keeping the phone permanently connected to an external power source charges its battery and prevents it from being fully discharged. In addition, if the phone switches off, it can be started again with a command, which would be impossible without external power. These arguments confirm the initial assumption that an older-generation mobile phone can serve as a modem, keeping its functionality while lowering the cost of the whole system.

The station could be used, for example, to monitor the meteorological conditions of plants grown in a greenhouse, or outdoors in an allotment garden. This allows unfavorable conditions to be detected early, without having to go there. An event that calls for preventive action could be insufficient air humidity or excessive temperature, and thus the need to water the soil.

The system has been designed so that extending its functionality, by adding more sensors or components that perform a given action (e.g. a relay controlling a humidifier or a heater), should not be a problem. In terms of hardware, the universal

extension board described in Section 3.1 has free space for additional components, and the goldpin headers of the **NUCLEO** kit give easy access to the required power supply and to the microcontroller pins. In terms of software, the operating system makes it easy to add new code by implementing another task (thread) or interrupt service routine. The weather station could become more universal by supporting other mobile phones, for example through the **MBUS** and **FBUS** protocols mentioned in Section 2.6.

On the other hand, a system like this, communicating over the phone network, can also be used for other purposes. One example is notifying about unwanted movement detected by a photocell, with the option of listening to the surroundings through the phone's built-in microphone at the same time.

Bibliography

[1] STMicroelectronics: *STM32F401xD STM32F401xE Datasheet*, (DocID025644 Rev 3), 01.2015, http://www.st.com/web/en/resource/technical/document/datasheet/DM00102166.pdf (accessed 15.02.2016)

[2] STMicroelectronics: *STM32F401xB/C and STM32F401xD/E Reference Manual*, RM0368, (DocID025350 Rev 4), 05.2015, http://www.st.com/web/en/resource/technical/document/reference_manual/DM00096844.pdf (accessed 15.02.2016).

[3] STMicroelectronics: *STM32 Nucleo-64 boards User manual*, UM1724, (DocID025833 Rev 9), 08.2015, http://www.st.com/web/en/resource/technical/document/user_manual/DM00105823.pdf (accessed 15.02.2016)

[4] STMicroelectronics: *Description of STM32F30xx/31xx Standard Peripheral Library*, UM1581, (DocID023800 Rev 1), 10.2012, http://www.st.com/web/en/resource/technical/document/user_manual/DM00068049.pdf (accessed 15.02.2016)

[5] Aosong: *Temperature and humidity module AM2302 Product Manual*, http://akizukidenshi.com/download/ds/aosong/AM2302.pdf (accessed 15.02.2016)

[6] Aosong: *Digital-output relative humidity & temperature sensor/module AM2303*, http://kamami.pl/dl/dht22_ds.pdf (accessed 15.02.2016)

[7] Maxim Integrated: *1-Wire Communication Through Software*, APP 126, 30.05.2002, https://www.maximintegrated.com/en/app-notes/index.mvp/id/126 (accessed 15.02.2016)

[8] Bosch: *BMP180 Digital pressure sensor Data sheet*, Revision 2.5, BST-BMP180-DS000-09, (Reference 0 273 300 244), 05.04.2013, http://cdn.sparkfun.com/datasheets/Sensors/Pressure/BMP180.pdf (accessed 15.02.2016)

[9] ITU-T: *V.250 Recommendation: Serial asynchronous automatic dialling and control*, 07.2003, https://www.itu.int/rec/T-REC-V.250/en (accessed 15.02.2016)

[10] 3GPP: *TS 27.007: AT command set for User Equipment (UE)*, ETSI TS 127 007, V12.11.0, 01.2016, http://www.etsi.org/deliver/etsi_ts/127000_127099/127007/12.11.00_60/ts_127007v121100p.pdf (accessed 15.02.2016)

[11] Sony Ericsson: *AT commands for Sony Ericsson phones*, 17th edition, 06.2010, (Publication no. 1206-6103.17), http://dl-developer.sonymobile.com/documentation/DW-65054-dg_at_2006--10_r17a.pdf (accessed 15.02.2016)

[12] J. Bogusz: *Moduły GSM w systemach mikroprocesorowych*, Wydawnictwo BTC, Warszawa 2007.

[13] 3GPP: *TS 23.040: Technical realization of the Short Message Service (SMS)*, ETSI TS 123 040, V12.2.0, 10.2014, http://www.etsi.org/deliver/etsi_ts/123000_123099/123040/12.02.00_60/ts_123040v120200p.pdf (accessed 27.02.2016)

[14] 3GPP: *TS 23.038: Alphabets and language-specific information*, ETSI TS 123 038, V12.0.0, 10.2014, http://www.etsi.org/deliver/etsi_ts/123000_123099/123038/12.00.00_60/ts_123038v120000p.pdf (accessed 27.02.2016)

[15] 3GPP: *TS 27.005: Use of Data Terminal Equipment - Data Circuit terminating Equipment (DTE - DCE) interface for Short Message Service (SMS) and Cell Broadcast Service (CBS)*, ETSI TS 127 005, V13.0.0, 01.2016, http://www.etsi.org/deliver/etsi_ts/127000_127099/127005/13.00.00_60/ts_127005v130000p.pdf (accessed 27.02.2016)

List of Figures

2.1. The NUCLEO-F401RE development board..... 10
2.2. Aosong AM2302 humidity sensor 15
2.3. BMP180 air pressure sensor on a SparkFun board..... 16
2.4. Main phone – Sony Ericsson K300i 18
2.5. Additional phone – Sony Ericsson K700i 18

3.1. Custom extension board with NUCLEO sockets..... 19
3.2. Original DCU-11 cable with a USB plug..... 24
3.3. Pin numbering of the Sony Ericsson connector 24
3.4. DCU-11 cable plug after the modifications..... 25