NUCLEO STM32F103 BLUETOOTH COMMUNICATION

Programming of Microcontrollers – Final Project

University of Bern / Berner Fachhochschule

Author: Hector Alvarez Marquez
hector.alvarezmarquez@students.unibe.com

Contents

Introduction

Wireless communications have transformed health care sector by integrating real-time activity monitoring of patients. Additionally, wireless systems are portable and flexible, for this reason clinical intervention is not required; hence, patients are free to move and continue with their daily activities. Rehabilitation, cardiovascular, and diabetes treatments rely on the implementation of medical devices that assist patients either during short or extended periods of time. Medical implants placed inside the human body, known as percutaneous implants, involve a high-risk level if the patient completely depends on the device function; therefore, the implant must comply with a set of regulations to guarantee the safety, and the effectiveness of the device. Due to response variation from one patient to another, a communication mechanism that notifies an external control system should be integrated to provide the means for monitoring, calibrating, and adjusting the operation conditions.

The IEEE 802.15.4 standard defines the low-rate Wireless Personal Area Networks protocols, which include, Zigbee, Bluetooth (BT), and Bluetooth Low Energy (BTLE). The Zigbee protocol is intended to use for less expensive applications that require short-range low-rate wireless data transfer. In contrast, the BT protocol provides a high-rate data transfer, with a maximum data rate of 1 Mb/s; furthermore, the standard defines 40 channels to communicate in the operating frequencies avoiding data corruption, and interference. The BTLE is the state of the art BT protocol that includes the requirements to achieve low power consumption. Additionally, the communication frequency is harmless for the human body; hence, BTLE is an optimal solution implemented in medical devices.

Mobile applications are a complementary tool that supply flexibility and mobility by means of BTLE. The information transmitted contains the performance of the implant, suitable for critical event detection. Moreover, corrective actions can be implemented by reading patient records preventing critical issues. Nowadays, medical applications for the smartphone have been integrated with a variety of devices to generate reports containing information about health conditions, and activity levels of individual users. Furthermore, over 2 billion new devices integrating BT are produced every year, meaning that the market is continuous expansion, and BT incorporation has almost become a requirement for any technological development. The aim of this project is to provide an understanding of the principles of data transmission, data encryption, and data protection through the development of a typical BT application using established guidelines by the Bluetooth 4.0 specification.

Bluetooth communication Protocol

Data transfer requires two components: a transceiver, and a receptor. The transceiver is the component which starts the communication whenever information from the receptor needs to be acquired. The receptor replies only when a connection request is received; through this process, energy consumption is reduced, extending the life of the battery; hence, the durability of the device is prolonged. The Bluetooth core specification (BCS) states the modulation process for data transmission, which is done through service advertising. The operating frequencies established by the specification cover the range from 2.40 GHz to 2.48 GHz. Additionally, a flexible communication is accomplished by subdividing the frequency range into 40 channels. Before two devices start the transmission process, authentication must take place. iPhone operating systems (iOS) use universally unique identifiers UUID to authenticate a device, and their respective services.

Universally Unique Identifier

In its canonical form, a UUID is represented by 32 lowercase hexadecimal digits, displayed in five groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters (32 alphanumeric characters and four hyphens). For general representation, xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx, the most significant bits of N indicate the variant (depending on the variant; one, two, or three bits are used). The variant covered by the UUID specification is indicated by the two most significant bits of N being 1 0 (i.e., the hexadecimal N will always be 8, 9, a, or b). The variant covered by the UUID specification (RFC 4122) has five versions. For this variant, the four bits of M indicate the UUID version (i.e., the hexadecimal M will be either 1, 2, 3, 4, or 5).

Each version uses different information to generate the UUID and thus some may be more appropriate than the others in specific use cases. According to the specification, Version 1 UUIDs are generated from date-time and MAC address, version 2 UUIDs are generated from group or user id and date-time, versions 3 and 5 produce deterministic UUIDs generated from a user-specified namespace and user-supplied data, and version 4 UUIDs are generated using a random or pseudo-random number.

During development, the device UUIDs where created using an online UUID generator (https://www.uuidgenerator.net/version1). Table 1 shows the UUIDs and their respective functions.

UUID	Function
5f298c8e-c851-11e6-9d9d-cec0c932ce01	Device ID
5f298f18-c851-11e6-9d9d-cec0c932ce01	Service ID I
5f2994a4-c851-11e6-9d9d-cec0c932ce01	Service ID II
5f2995ee-c851-11e6-9d9d-cec0c932ce01	Service ID III
5f2996ca-c851-11e6-9d9d-cec0c932ce01	Service ID IV
5f299792-c851-11e6-9d9d-cec0c932ce01	Service ID V
5f29985a-c851-11e6-9d9d-cec0c932ce01	Service ID VI

Table 1. Universally Unique Identifiers and Functions

Hardware components – Receptor (Peripheral System)

The peripheral system, illustrated in Figure 2, comprises a MBED application shield module, a RN42XV module, and a STM32F103 Cortex M3 processor; furthermore, the system transmits encoded packages advertising a set of wireless services.

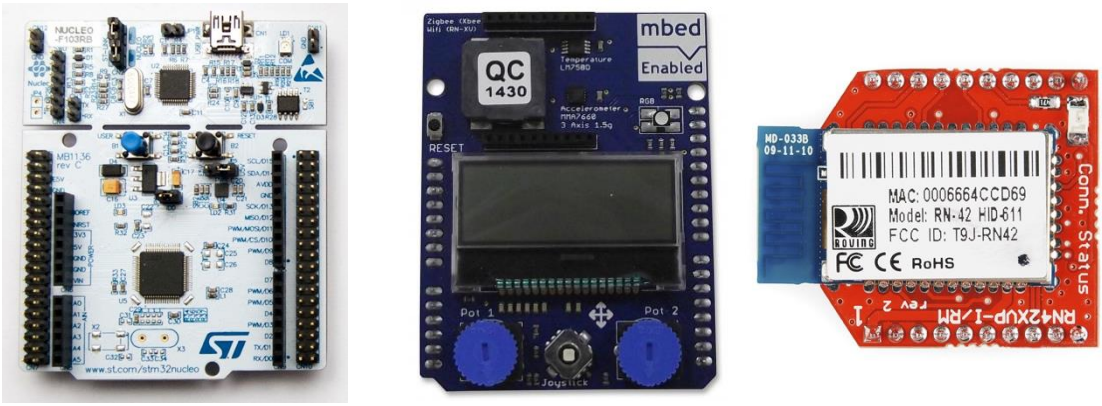

Figure 1. Peripheral system components: (a) Nucleo STM32F103RB Cortex, (b) MBED application shield, (c) RN42XV module.

RN42XV Module

The RN42XV are small form factor, low-power Bluetooth radio modules offering plug-in compatibility for the widely used 2 x 10 (2-mm) socket typically used by 802.15.4 radio modules. These modules are simple to design in and are fully certified, making them a complete embedded Bluetooth solution. Table 2 presents the module pin description simplified with the pins used during development.

Pin Number	Signal Name	Description	Direction
1	VDD_3V3	3.3 V regulated power input to the module	Power
2	TXD	UART TX, 8 mA drive, 3.3 V tolerant	From module
3	RXD	UART RX, 3.3 V tolerant	To module
5	RESET_N	Optional module reset signal (active low), 100 k pull up, apply	Input

		pulse of at least 160 µs, 3.3 V tolerant	
10	GND	Ground	Ground
12	RTS	UART RTS flow control, 3.3 V tolerant.	From module
16	CTS	UART CTS flow control, 3.3 V tolerant.	To module

Table 2. RM42XV Pin Description

The most basic application using a RN42XV module, requires the connection of the pins voltage supply (VDD), transmitter (TX), receiver (RX), reset (NRST), and ground (GND). Additionally, the pins ready to send (RTS), and clear to send (CTS) control the flow during transmission. The schematic of the module is illustrated in Figure 4.

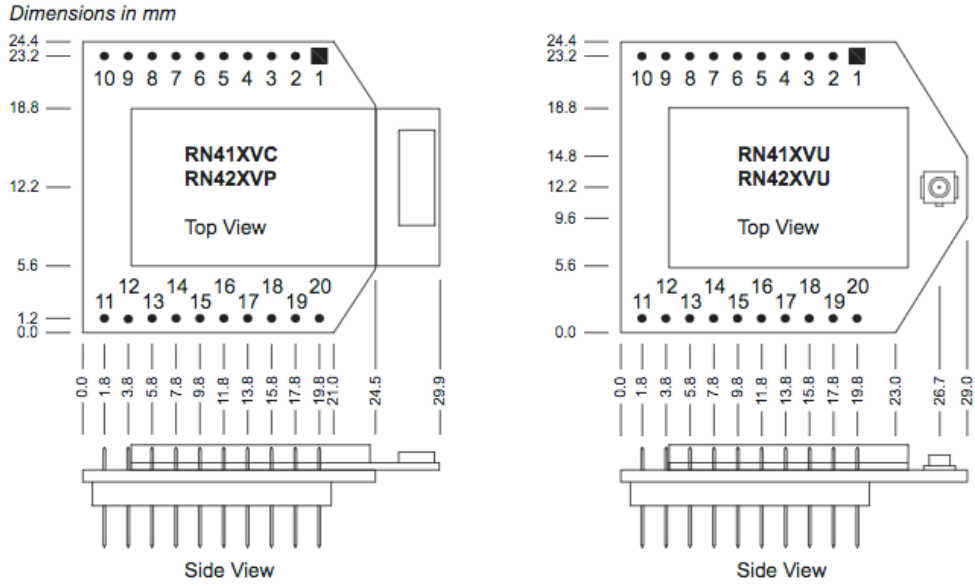

Figure 2. RN42XV schematics.

MBED application shield

The mbed application shield, illustrated in Figure 3, is a printed circuit board (PCB) compatible with Nucleo STM32F103 microcontrollers used as an expansion board for fast-prototyping with an ease of design. The MBED application shield acquires data from the temperature sensor, the potentiometers,

and the RN42XV module; simultaneously, the information is sent to the STM32F103 Cortex M3 processor.

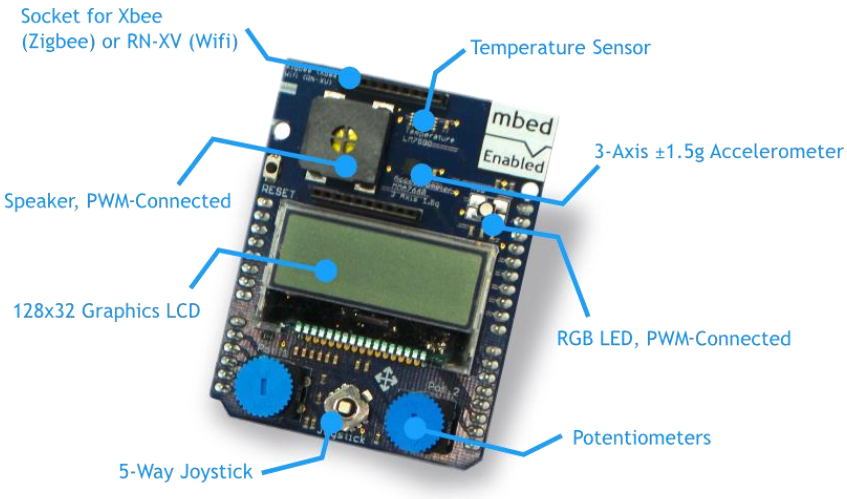

Figure 3. MBED application shield components.

The dedicated pins for each component are depicted in Figure 4. During project development, most of the pins, with the exception of the SW_UP, SW_DOWN, SW_LEFT, and SW_RIGHT pins, were used to activate each of the components.

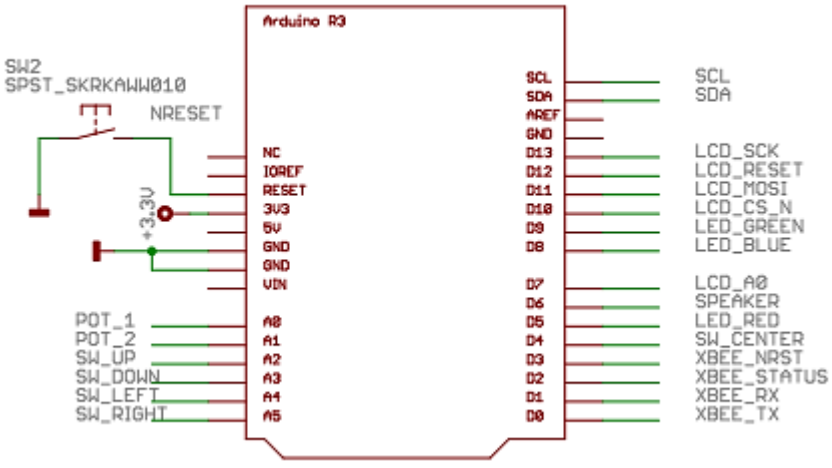

Figure 4. MBED schematics.

STM32 Nucleo Board

The STM32 Nucleo board provides a flexible way for building prototypes with the STM32 microcontroller, choosing from the various combinations of performance, power consumption, and features. The STM32 microcontroller is responsible for analyzing, and processing the information received through the general-purpose input/output (GPIO) pins, followed by a response generation. Additionally, the microcontroller is the interface for software programming integration, which enables algorithm implementation to control the hardware, and the overall behavior of the system. Figure 5 illustrates the STM32 board configuration when a mbed expansion board is integrated.

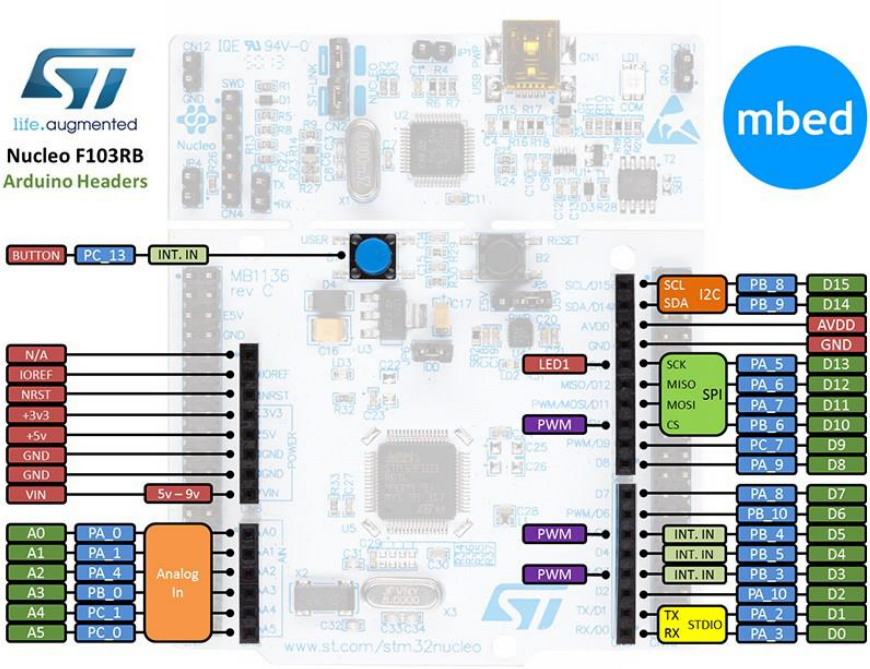

Figure 5. STM32 board schematics.

Finally, the overall process executed by the peripheral system is presented in Figure 6.

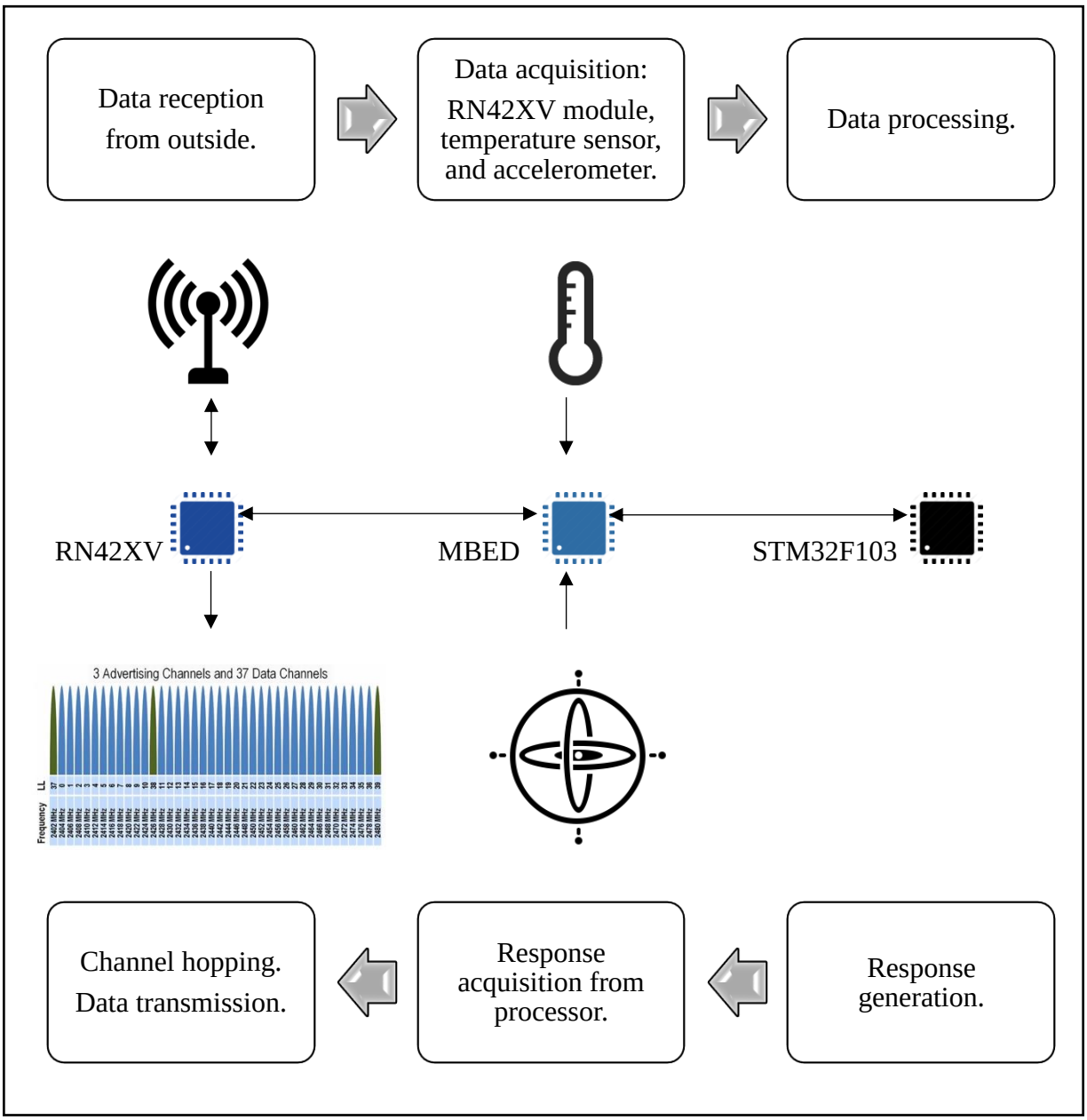

Figure 6. Peripheral system data transmission schematic.

Bluetooth – Communication Validation

Before code development, the RN42XV module was tested to guarantee an effective device performance. For this process, the device was connected using the voltage, ground, transmission, and reception pins. Additionally, an external terminal connected through Bluetooth enabled bidirectional data transmission. The Coolterm software used for testing purposes is a serial communication emulation software, which provides an interactive interface by standardized configuration settings, such as, baud rate, parity check, RTS mode, CTS mode, encryption mode, etcetera. The Bluetooth configuration settings are displayed in Table 3 including control parameters, and their respective values.

Control Parameter	Description	Value
Baud rate	Rate at which information is transferred in a communication channel.	115200 Bps
Parity check	Bit added to a string of binary code to ensure that the total number of 1-bits in the string is even or odd. Parity bits are used as the simplest form of error detecting code.	None
Data	UART RX, 3.3 V tolerant	8
Stop bits	Indicates the end of each package sent to differentiate, and recognize data transmission start/stop	1
RTS/CTS flow control	Handshake method, indicates when data is ready to be sent or received	Not implemented
Port	Port used for communication	BT module

Table 3. Serial communication settings.

Figure 7 shows the terminal configuration implemented during validation where the control parameters were assigned using the values presented in Table 3. Notice that the name of the port selected corresponds to the RN42 module generic name.

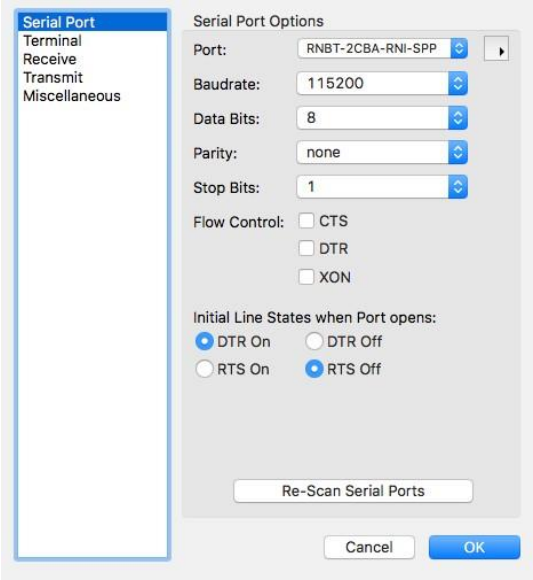

Figure 7. Coolterm configuration.

Command mode is established when the RN42XV module replies with the string CMD after typing $$$, which indicates that the connection and terminal settings are correct. While command mode is enabled, the module accepts ASCII bytes as commands. Table 4 shows a list of commands used to validate the module performance.

Command	Description
$$$	Enable command mode.
h <cr>	Displays a list of commands.
x <cr>	Confirms that command mode is selected; in addition, this command shows the summary of the module's current settings, such as Bluetooth name, device class, and serial port settings.
GN,	Get device name
SN,	Set device name

Table 4. Coolterm commands.

The results obtained during BT validation, presented in Figure 8, demonstrate that both transmission, and reception efficiently work; thus, the device performance has been proven to be optimal.

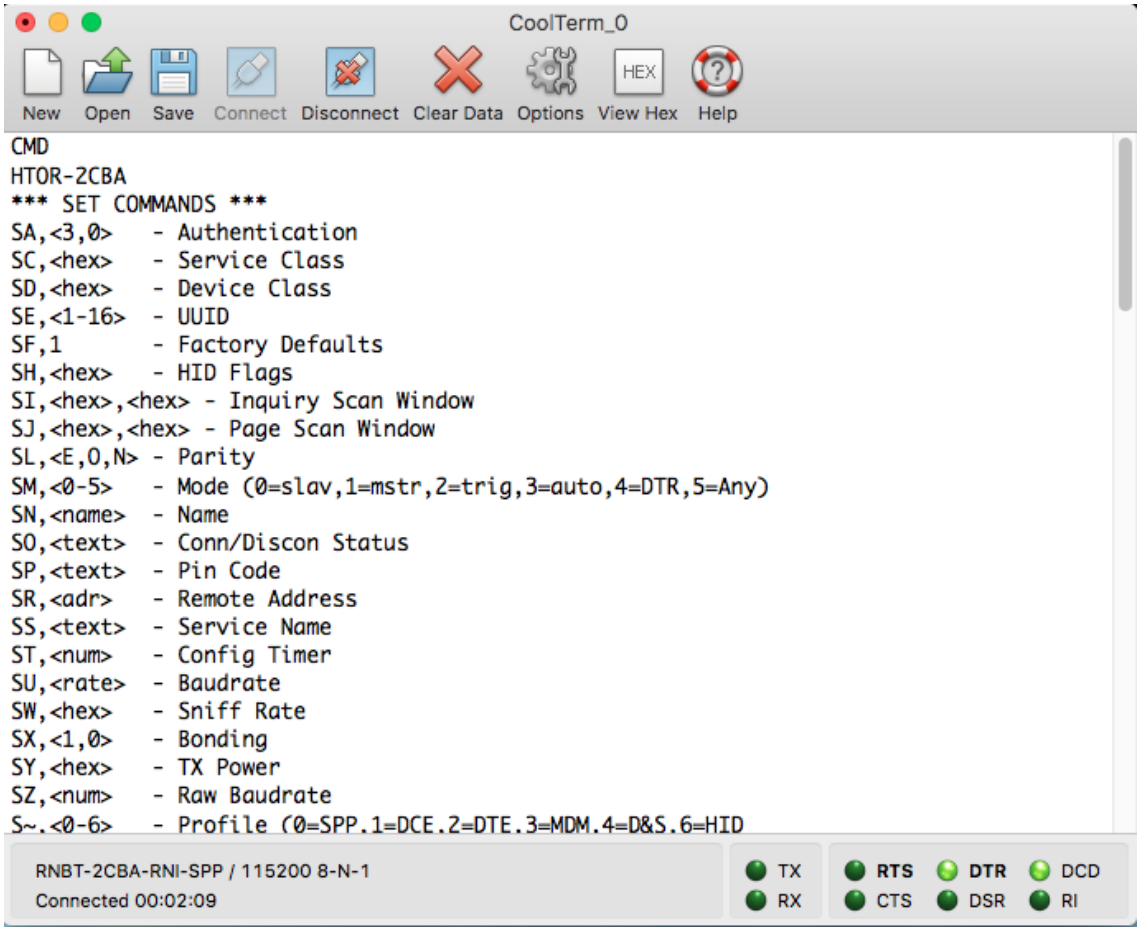

Figure 8. BT validation results.

STM32 architecture

After component validation, an interface diagram was generated to define a control mechanism for each component. The STM32 processor architecture, is comprised by hardware components which fulfil specific functions. The interface diagram, illustrated in Figure 9, provides a clear description which defines the interaction between hardware, and software components.

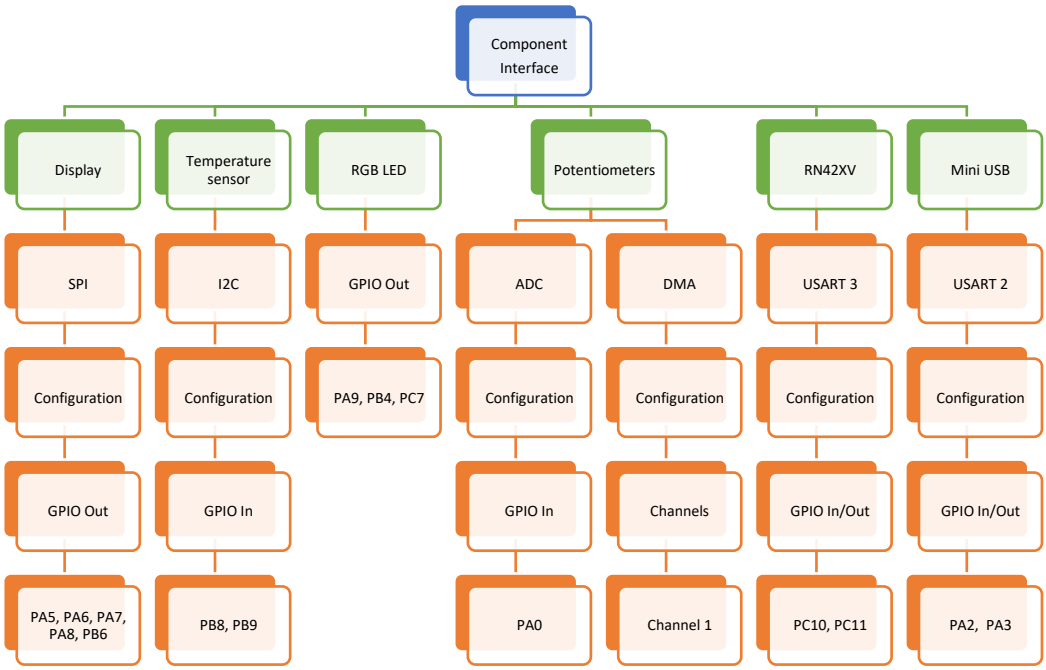

Figure 9. Hardware-Software Interface Diagram.

SPI Bus

The Serial Peripheral Interface (SPI) bus is a high-speed synchronous serial communication interface specification used for short distance communication. SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. The code, shown in Figure 10, corresponds to the SPI configuration used during development.

```
103  void SPI_configuration(void)
104  {
105    SPI_InitTypeDef SPI_InitStructure;
106    SPI_I2S_DeInit (SPI1);
107
108    SPI_InitStructure.SPI_Direction = SPI_Direction_1Line_Tx;
109    SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
110    SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
111    SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
112    SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
113    SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
114    SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_32;
115    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
116    SPI_InitStructure.SPI_CRCPolynomial = 7;
117
118    /*INITIALIZE*/
119    SPI_Init (SPI1, &SPI_InitStructure);
120
121    /*ENABLE SPI1*/
122    SPI_Cmd (SPI1, ENABLE);
123  }
```

Figure 10. SPI configuration code.

I2C

The Inter-Integrated Circuit (I2C) is a multi-master, multi-slave, single-ended, serial computer bus. It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication. The code, presented in Figure 11, corresponds to the I2C configuration used during development.

```
126  void I2C_configuration(void)
127  {
128    I2C_InitTypeDef  I2C_InitStructure;
129
130    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
131    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
132    I2C_InitStructure.I2C_OwnAddress1 = 0x3C;
133    I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
134    I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
135    I2C_InitStructure.I2C_ClockSpeed = 400000;
136
137    I2C_Init(I2C1, &I2C_InitStructure);
138    I2C_Cmd(I2C1, ENABLE);
139  }
140
```

Figure 11: I2C configuration code.

ADC

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 18 multiplexed channels allowing it measure signals from 16 external and two internal sources. A/D conversion of the various channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored in a left-aligned or right-aligned 16-bit data register. It is possible to organize the conversions in two groups: regular and injected. In continuous conversion mode ADC starts another conversion as soon as it finishes one. The ADC 1 with the respective channels CH0, and CH1, has been used during development to acquire data from the mbed board potentiometers. Figure 12 shows the ADC configuration code implemented.

```
141  void ADC_configuration(void)
142  {
143    ADC_InitTypeDef  ADC_InitStructure;
144    RCC_ADCCLKConfig(RCC_PCLK2_Div6);
145
146    ADC_DeInit(ADC1);
147
148    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
149    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
150    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
151    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
152    ADC_InitStructure.ADC_ScanConvMode = ENABLE;
153      //For DMA
154     ADC_InitStructure.ADC_NbrOfChannel = 2;
155
156    /*ENABLE ADC1*/
157    ADC_Init (ADC1, &ADC_InitStructure);
158    ADC_Cmd(ADC1,ENABLE);
159
160    ADC_ResetCalibration(ADC1);
161    while(ADC_GetResetCalibrationStatus(ADC1));
162    ADC_StartCalibration(ADC1);
163    while(ADC_GetCalibrationStatus(ADC1));
164
165     ADC_SoftwareStartConvCmd(ADC1,ENABLE);
166
167     ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_71Cycles5);
168     ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_71Cycles5);
169
170     /* Enable ADC1 DMA */
171     ADC_DMACmd(ADC1, ENABLE);
172  }
173
```

Figure 12. ADC configuration code.

US ART

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a fractional baud rate generator.

The USART is the main component used for serial communication in this project. The STM32 processor includes 3 USARTs for development processes. The USART 2 is dedicated to exchange information through the mini-USB port. This port communicates with a terminal to inform any operation executed by the device operating as an observer, which controls and supervises the device performance. The integration of the mbed application board restricts the use of USART 1, due to pin usability; hence, the USART 3 was selected for controlling Bluetooth transmission. Although the predefined pins for USART 3 are already in use, same issue as for USART 1 due to the extensive use of hardware, an available pin remapping solves this problem. The USART configuration code is presented in the following figure.

```
283   /* Configure USART2 RX/TX */
284   USART_InitStructure.USART_BaudRate            = 115200; //
285   USART_InitStructure.USART_WordLength          = USART_WordLength_8b;
286   USART_InitStructure.USART_StopBits            = USART_StopBits_1;
287   USART_InitStructure.USART_Parity              = USART_Parity_No;
288   USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
289   USART_InitStructure.USART_Mode                = USART_Mode_Tx | USART_Mode_Rx;
290   /* Initialize USART2*/
291   USART_Init(USART2, &USART_InitStructure);
292   USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
293   USART_Cmd(USART2, ENABLE);
294
295   /* Configure USART3 RX/TX */
296   USART_InitStructure.USART_BaudRate            = 115200; //
297   USART_InitStructure.USART_WordLength          = USART_WordLength_8b;
298   USART_InitStructure.USART_StopBits            = USART_StopBits_1;
299   USART_InitStructure.USART_Parity              = USART_Parity_No;
300   USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
301   USART_InitStructure.USART_Mode                = USART_Mode_Tx | USART_Mode_Rx;
302   /* Initialize USART3*/
303   USART_Init(USART3, &USART_InitStructure);
304   USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);
305   USART_Cmd(USART3, ENABLE);
```

Figure 13. USART configuration code.

Core Bluetooth Functions

A communication diagram, illustrated in Figure 14, was developed to provide a general idea of the operations required to comply with the Core Bluetooth specification. iOS applications are compatible with the BT specification; thus, following the standards guarantee an effective integration between the mobile application, and the peripheral system.

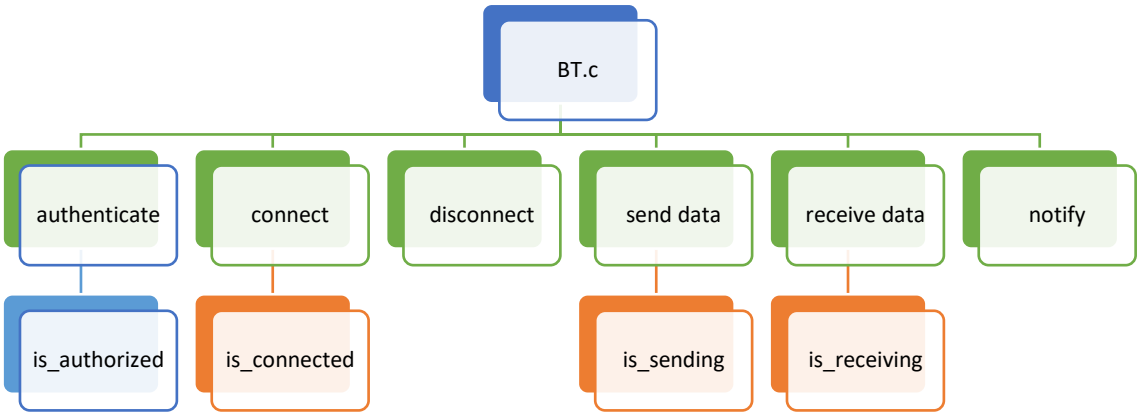

Figure 14. Core Bluetooth functions.

The main functions provided by the peripheral system should include: authenticate, connect, disconnect, notify, and send/receive data. The function description is presented in table 5.

Function	Description	Input	Output
is_authorized	Higher security system before data transmission is enabled	Void	Boolean
is_connected	Verifies the connection status	Void	Boolean
disconnect	Ends BT communication	Void	void
send_data	Configured by retarget, sends information through iterations	printf(…)	Uint_16 ASCII
is_sending	Verifies TX	Void	Boolean
receive_data	Activates interruption handler when information is received	Uint_16 ASCII	RXE
is_receiving	Verifies RX	Void	Boolean
notify	Notifies the observer when an action has been executed	String	Void

Table 5. Core Bluetooth function description.

Mobile App Development

The mobile application was created using Xcode, Graphical User Interface (GUI) for apple developers. The iPhone Operating System (iOS) application includes the main classes, and the templates used during development, AppDelegate, and ViewController. In addition, a controller scene provides interactive development, which is a useful tool visualizing the output preview. The CoreBluetooth framework offers predefined classes, functions, and objects to develop applications incorporating Bluetooth communication; thus, design is easy, robust, and most important complies with the Bluetooth Core Specification. The framework contains defined functions to induce sleep mode when there is no interaction between the devices; this mode is required to achieve low energy consumption, which is a key feature in medical devices.

The application view, illustrated in Figure 15, displays the components that execute the specific actions to control the overall system. The connect button initiates the synchronization between the mobile application and the peripheral. When the button is selected its color changes indicating the operation, and the connection label indicates if the peripheral has successfully connected or not. The temperature, and position labels read and display the information sent by the peripheral corresponding to the variables of interest. The move right, move left, move down, and move up buttons activate specific functions performed by the external device. The disconnect button ends the communication between devices; additionally, the connection label changes again to the status disconnected. Finally, a progress bar displays the percentage of work completed, creating a friendly interaction between the user and the application.

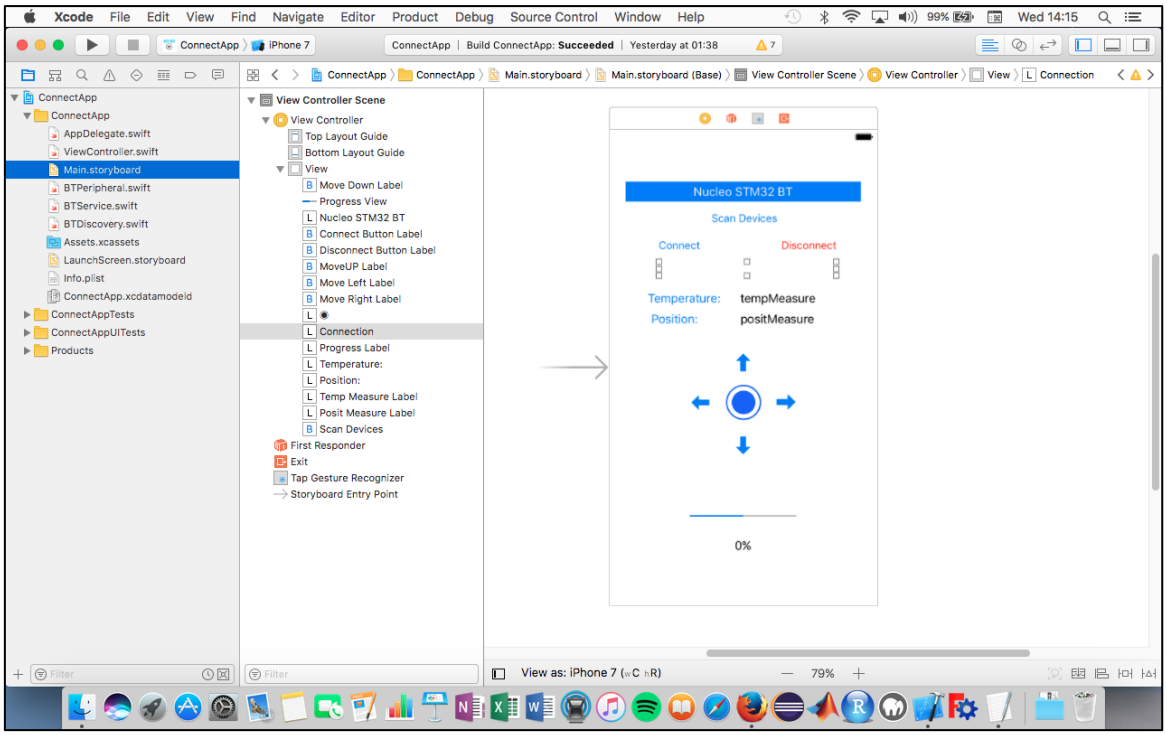

Figure 15. Application view controller scene.

Results

The peripheral system, illustrated in figure 16, provides the following operations:

1. Start button.
2. Read, and display temperature.
3. Read, and display potentiometers (range 0 – 100%).
4. Display terminal (observer) connection status.
5. Display BT communication status.
6. RGB status indication (Wrong authentication, authorized, move up, etc.)
7. Mini-USB serial communication with terminal (observer), see figure 17.
8. BT communication App, see figure 18.

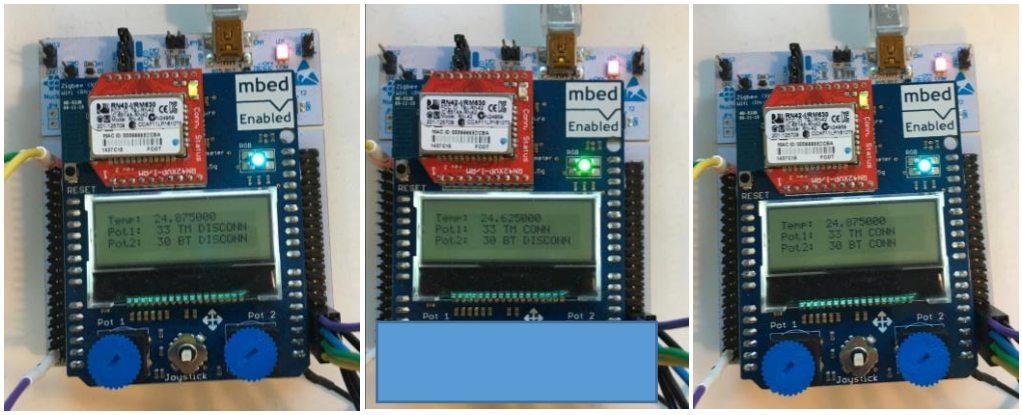

Figure 16. Peripheral system.

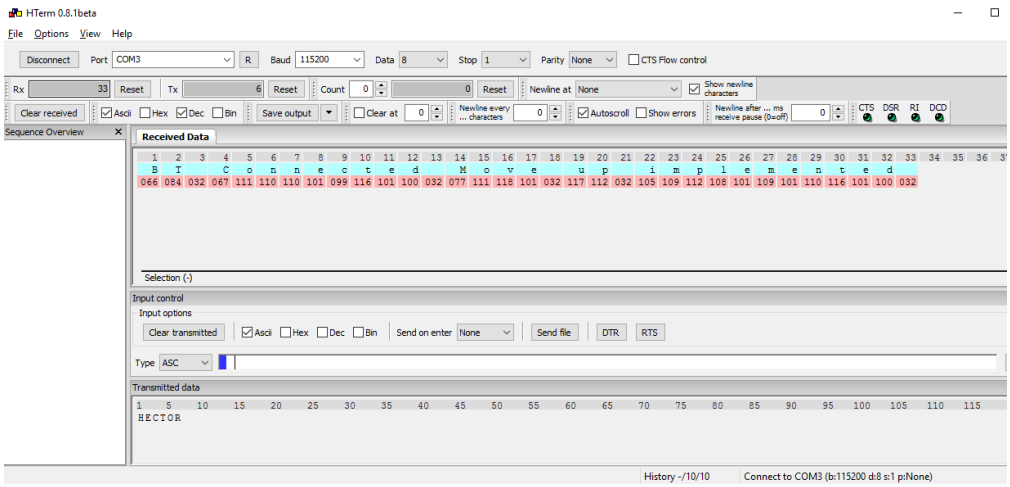

Figure 17. Observer terminal.

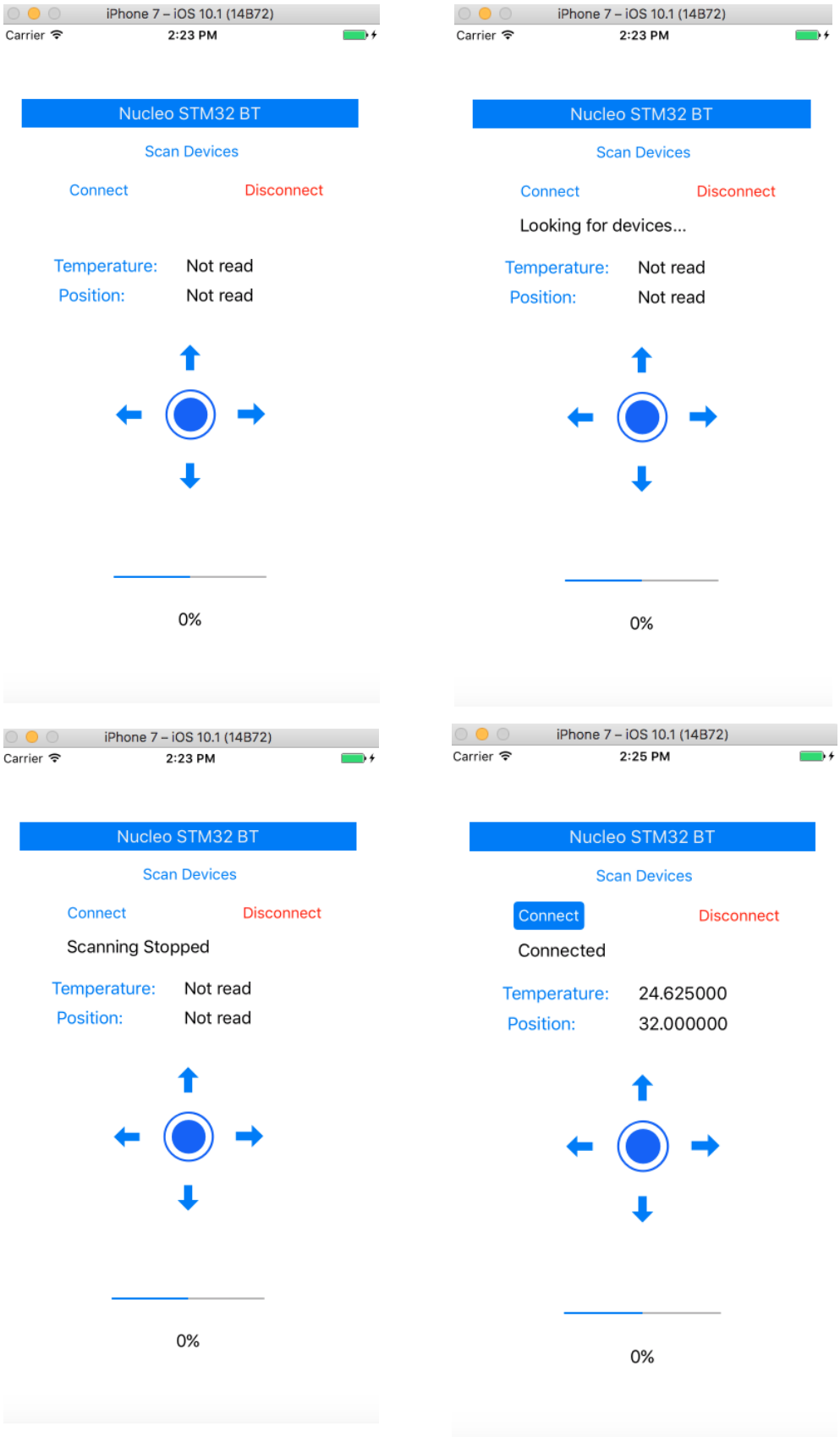

Figure 18. Mobile Application performance

Conclusions

The results have proven the functionality of mobile applications, and the ease of integration developing new wireless technologies. The code development must include the functions which comply with the communication protocols. The Graphical User Interface (GUI) keil µvision, should be configurated before code development including the features and components to be used, such as enable USART, SPI, I2C, etcetera; moreover, the code must be robust, and include interruption, and error handlers, to avoid any overflow leading to device malfunction.

Serial communication could be handled in three different ways. The first option is to call the function directly, i.e. USART_SendData(), which is a simple method, and requires no other features to be integrated. The second option is using interruption handlers, i.e. USART_IRQHandler. This method is very effective for data reception, due to the interruption when data is acquired, the software must attend immediately the request. The last option is to use a retarget method, for example, redefinition of printf (); therefore, the function can be called in any class avoiding code complexity.

Further work increasing data transfer security should be done. A combination of Amplitude Key Shifting (ASK) and Frequency Key Shifting (FSK) increases encoding complexity, consequently, the security of the system is enhanced. However, ASK is susceptible to data corruption which could lead to loss of information. The inclusion of idle mode improves low power consumption; moreover, data transmission is enabled after a secure connection request from the external controller is received, yielding enhanced data protection. Though Half duplex mode has the advantage to improve noise and error detection

executing unidirectional communication, speed rate is considerably affected. Differential signaling provides a better signal to noise ratio (SNR) sending paired signals rather than single signaling. Due to SNR enhancement transmission is faster; thus, low power consumption is achieved. Parity check is a simple solution to control data corruption; however, if the interference produces parity then hazard recognition fails. For that reason, Cyclic Redundancy Check (CRC), is the best solution for data protection detecting raw data changes due to unknown sources using polynomial division. Nevertheless, a cryptographic authentication mechanism must be implemented before data transmission starts. In conclusion, CRC in combination with message authentication code (MAC), differential signaling, and idle mode could be a good solution for data protection.

References

- RM0008 Reference Manual, STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM®-based 32-bit MCUs, ST
- Bluetooth Data Module Command, Reference & Advanced Information User's Guide, RN-BT-DATA-UG Version 1.0r 3/26/13, Rovin Networks
- Covered Core Package version: 4.2; Publication date: Dec 02 2014; Bluetooth SIG Proprietary
- The Swift Programming Language, Swith 3.0.1 Edition, 2016, Apple Inc
- Core Bluetooth Programming Guide, Bluetooth for Apple Developers, 2013, Apple Inc
- MBED application shield documentation, https://developer.mbed.org/cookbook/mbed-application-shield