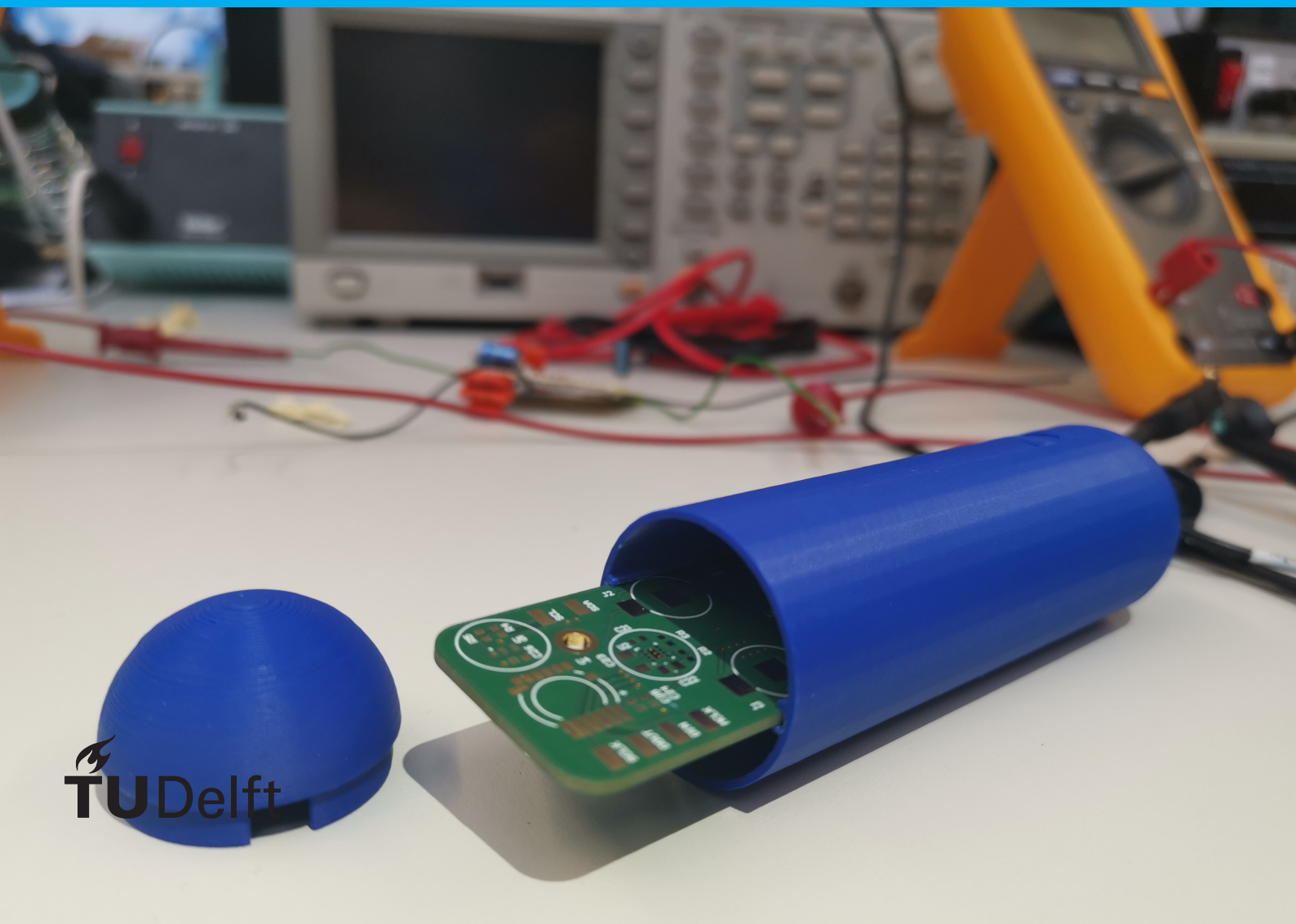
BAP Group E

Data management & control

A. Gardouh (4579941)
A. Aouchi (4521692)

Endoscopic pill project



TU Delft

BAP Group E

Data management & control

by

A. Gardouh (4579941)
A. Aouchi (4521692)

TU Delft

Preface and Acknowledgement

Before you is the thesis of the data management & control group of group E which consists of two members, Aschraf Gardouh & Ahmed Aouchi. This project is made as a graduation project for Bsc. Electrical Engineering at the technical university of Delft. Writing this thesis and finishing took us a period of 9 weeks with extensive research and hard work in the period of april 23th 2019 to june 21st 2019.

The project was undertaken at the request of dr. Virgilio Valente as one of the proposers of the Bachelors graduation project and as supervisor of our group altogether. The project was very ambitious in the amount of time allocated for the project so we have worked our hardest to try and deliver the best product we could about which more is talked in the rest of the thesis.

We would to thank our parents and friends for their support during the project. We would like to thank especially our supervisor dr. Virgilio Valente for his excellent guidance during the period of the project. We also would like to thank Martin Schumacher for his support in the Teleggen hal.

To all the group members of group E we would like to thank you all for your wonderful cooperation where occasionally good and bad ideas were bounced of each other. We had a wonderful time working with you all and we appreciate all the ups and downs we went trough during the project.

We Hope you will enjoy your reading,

A. Gardouh (4579941)
A. Aouchi (4521692)
Delft, June 2019

Abstract

Endoscopic capsules have been a very interesting idea in which a lot of progress has been developed lately. These are capsules with electronics inside of them instead of the regular pill. The capsule can be used for a lot of different applications. This thesis documents the design process of data management and control module of a endoscopic capsule with multiple sensing units. In this process subjects such as component selection for the control unit, the transmission unit and the reason behind these selections. In the end the data is being wireless transmitted and managed by a CC2650 MCU of TI. Also the data from the sensor are being managed correctly and then sent to the transmitter. The transmitter then should be able to send up to 5m from inside the human body and the average current of the wireless transmission is a average of 23.36 μA and a transmission time per package of 2.736ms. The transmitted data is then received on a development board of the CC2650 and shown real time in a matlab graph.

Contents

1 Introduction

Endoscopy is a way of looking into the body and is an important part of medicine nowadays. Endoscopy is still used mostly in a traditional way with endoscopes which are small tubes that are used to investigate the Gastrointestinal(GI) tract. Due to technological advancements instead of using small tubes endoscopy can be performed by using a endoscope in capsule form which is called capsule endoscopy(CE). Capsule endoscopy is a important tool to investigate the small bowel since this can not be done completely with traditional endoscopes[1]. Investigating the small bowel is important because there are some symptoms of diseases that can be spotted in the small bowel. Thus these symptoms can not be spotted using traditional endoscopy [2]. Using capsule endoscopy is a safe and non-invasive way of doing that[3].

The goal of this project is to design such a capsule that does not just have a camera like the commercially available endoscopic capsules[4] but add sensing units to it that can measure useful parameters in the gastrointestinal(GI) tract in real time. The system consists of a controller unit, multiple sensing units, power management and data transmission. For designing the system we have split the workload into to three subgroups: the sensor group, power management group and the data transmission & control group. This thesis is about the product delivered by the data transmission & control group. The part that is worked on by the data transmission & control group is highlighted in figure 1.1.

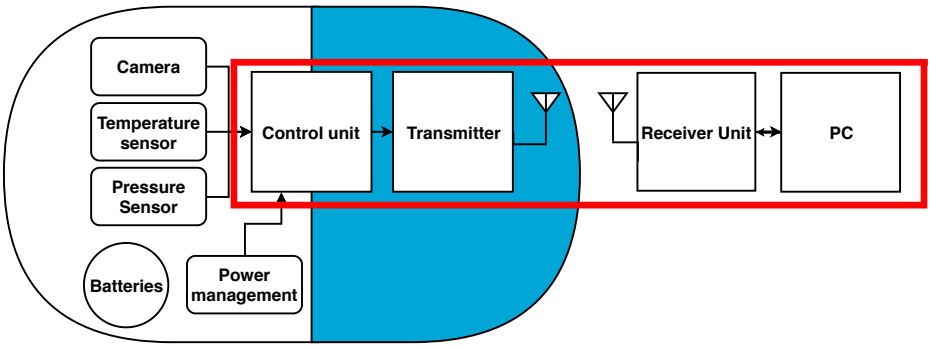


Figure 1.1: Endoscopic capsule on system level

1.1. System design

1.1.1. Top level design

In order to design the endoscopic capsule, first the capsule needs to be considered as a system that consists of several modules that are interconnected to deliver the desired behavior.

The system or the capsule is considered as a black box which uses input values and creates the expected output. This way the input and the output can be determined and the functionality of the system at a top

level can be described. The input in this case are physical signals picked from the body and the output would be the illustration of this signals as values in a graphical user interface. See figure 1.2.

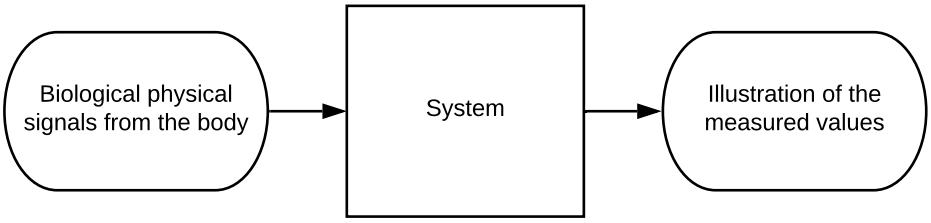


Figure 1.2: Input/output system overview

After determining the input, the output and the functionality of the system, the black box is divided in several components that each has its own functionality and interconnected with each other. By designing each component, together the parts should create the desired output using the input. The design steps followed to determine the components of the system and their functionality are:

- Determining which input and output interfaces are necessary for the system.
- Determining a main part which connects the input interface with the output interface.
- Further dividing the input interface, output interface and the main part into smaller modules that can deliver the overall functionality.
- Determining a block to deliver the energy required for the system to be able to function.

By following the design steps described above, the system can be described in several components. See figure 1.3 for the top level overview. The components of the system are as follows:

- Input interface:
  - Temperature sensor
  - Pressure sensor
- Main block:
  - Digital IC
- Output interface:
  - RF (transmitter and receiver)
  - Graphical User Interface
- Energy:

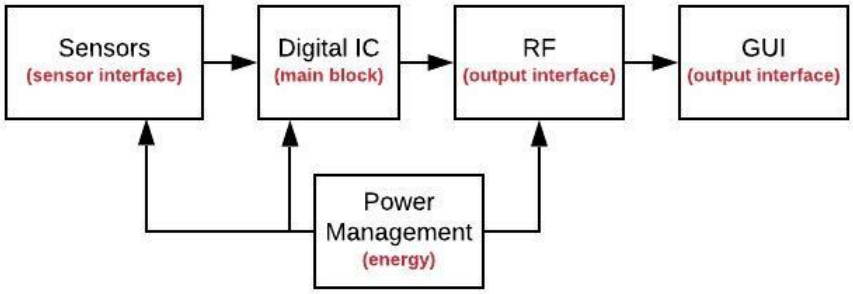


Figure 1.3: System modules

- Power management
- Batteries

The last design step is to design each component of the system. In the next chapters the design of the RF, the Digital IC and GUI modules will be discussed. These three components together are responsible for data management and control.

1.1.2. System components

In this paragraph the functionality of each component of the system will be briefly discussed so the overall functionality of digital IC and RF in the system will be clear. In the next sections the design of the digital IC and the RF blocks will be explained in detail.

Sensors
The physical signals from the body are physical signals which need to be translated to analog electrical signals and eventually to digital electrical signals. The digital values that describe the information of the desired quantities of the human body can be further processed a digital IC to eventually be illustrated in a user interface. By using the physical signals as input from the human body the output can be generated as digital translated signals which serves as input for the digital IC block.

Digital IC
This part of the system is responsible for data management by capturing data from the sensors and deliver it to the transmitter. Digital IC, as the brain of the system, controls and communicates with other modules and processes data in an efficient manner in terms of power and performance. The output serves as input of the RF module which transmits and receives the processed data.

RF
The RF module consists of a transmitter and a receiver which both of the components uses an antenna. This part of the system is responsible to deliver the captured and processed data from the capsule in the body to a workstation outside the body. From the workstation the values can be illustrated in a user interface.

GUI
The final stage of the system is the graphical user interface. GUI illustrates the measured values in the first stage of the process in an understandable manner for the users.

Power management
The endoscopic capsule uses power to be able to function and that applies for each part described above. Power management is a crucial part of the system since it is important that the system should be able to function optimally in the period of time that it is required. Also this part delivers power using batteries and converters.

1.1.3. Data management and control

This part of the system contains the digital IC module and the RF module. See the previous paragraph for a brief description of the modules. In this section the components of this part of the system and their interactions with each other and with the system will be discussed.

Figure 1.4 shows the interactions and the connections of the components of the data management and control module. The digital IC should contain the control unit, timer and the data processing blocks and the RF module should contain an antenna and an antenna interface in order to function as described in the previous paragraph.

The control unit is responsible to determine the timing of the behavior of the different blocks. It uses a timer to synchronize the reading of the sensor and setups the communication with the sensor and the antenna interface. Each a request is done by the control unit, the sensor interface gives data to the data processing block where data are processed. Afterwards the data is sent to the antenna via the antenna interface.

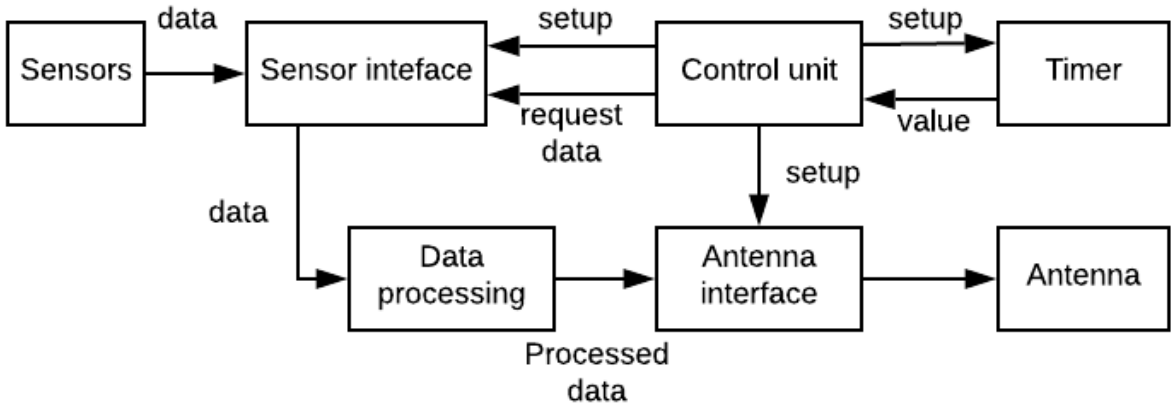


Figure 1.4: Data management and transmission architecture

In the next chapters the implementation of the system design will be discussed in detail. The abstract system design was necessary to design the individual modules of the system. In this theses the design of the digital IC and the RF blocks will be explained. Both of the modules describe the behavior of the data management and control of the system.

1.2. State of art analysis

Capsule endoscopy has come a long way already. There are endoscopic capsules that are already commercially available such as the pillcam of Medtronic. These are endoscopic capsule that just have a camera. The next step that has to be made for capsule endoscopy is to make the capsules have multiple sensing units added to the camera so that there is extra information linked to the images.

There is also research being done into wireless powering such an endoscopic capsule [5]. This would enable more features for a endoscopic capsule since if there is wireless powering there is less worries about power since there will be a constant supply of energy. This would be a huge advantage compared to the system which at the moment are powered by batteries. Now there is also research being done on if UWB-transmission is possible trough the body [6]. When UWB-transmission will be available for endoscopic capsules the frame rates and resolution of the cameras used can be upped quite a bit since the data rate will be very high when using UWB-transmission.

1.3. Document Structure

In this section the document structure will be given. In chapter 2 the requirements of the system and the requirements of the data management and control which follows from the system requirements are given. Then in chapter 3 there will be a break down of the complete system for the endoscopic capsule. After that in chapter 4 and 5 the design of the Digital IC and Data transmission will be explained. All the decisions made with all the reasons and component selection. After that in chapter 6 there will be further elaborated on the graphical user interface (GUI) and how it got implemented. In chapter 7 the performance of the designed system will be tested and how the implementation went. with in chapter 8 and 9 the discussion conclusion recommendation and future work.

2

Requirements

As first fase of the project, the requirements of the product was gathered and grouped together under functional and non-functional requirements.

2.1. Functional Requirements

The functional requirements of the system are requirements about what the system should do. In functional requirements now mandatory requirement to which the system must comply with and trade-off requirements which would be preferred to comply with but is not necessary. The functional requirements are the following:

- The user must be able to see the result of the measurements in a graphical user interface. See figure 2.1 which illustrates the use case diagram.
- The capsule should be able to measure the temperature and the pressure
- The capsule should be water proof
- The capsule could consist of a image sensor to detect images
- The measurements must be real-time
- The measurements must be transmitted wireless
- The capsules must have at least one sensing unit.
- The capsule should last for 24 hours

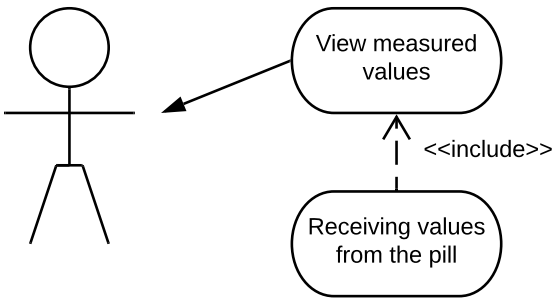


Figure 2.1: Use case diagram

2.2. Non-Functional Requirements

The non-functional requirements are the requirements of how the system should work. There are between the non functional requirements also mandatory and trade off requirements as described in the functional requirements. The non functional requirements are the following:

- The capsule could be able to be reused
- The capsule could be safe for a human body to swallow
- The capsule should have a diameter of 11 cm and a length of 23cm
- The capsule could be swallowed by a human.
- The data from the capsule should be able to be received in a range of 5 m from inside a human body
- The data rate must be at least 16 bits per second

3

Digital IC

In this chapter the design of the digital IC module will be discussed after explaining the functionalities of the components in the previous chapter.

There are two choices for the implementation of the digital IC, either a micro-controller or logic based circuit. [7] [8].
Logic based digital IC would be a better design choice than a micro-controller in terms of performance and efficiency while the micro-controller is more flexible, simple and contains more features. In this project the choice went to the micro-controller CC2650 for its simplicity to program and the flexibility to change or add more features.[8]

3.1. Modules

The digital IC component consists of three main modules: sensor interface, actuator interface and a control unit. The sensor interface is an important part for the sensors to communicate with the control unit. See figure 3.1 for the model scheme.

3.1.1. Sensor interfaces

The sensor interface which is called a sensor controller integrated in the micro-controller opens the communication with the sensors using the I2C protocol and read the values from the sensors. In order to save power, capturing the sensor values is synchronized with a frequency of 1Hz since the temperature and the pressure of a human body does not change so frequently. Each time the values is read from the sensors the sensor controller generates an interrupt in the main code on the CPU. The value of the sensors is passed by the sensor controller and captured by the interrupt function in the CPU.

3.1.2. Transmitter interfaces

The antenna interface or the RF core integrated in the micro-controller is responsible to deliver the processed data by the CPU to the antenna. After the communication is opened between the CPU and the RF core, each time when processing data is done the CPU sends a signal to the RF core to deliver the packet. After the transmission is successful the function that is used to send the packet returns a value to the CPU while the CPU continues with the main code since the RF core contains it own processing unit. [9]

3.1.3. CPU

This unit is the main control unit of the Digital IC. CPU acts as a control unit of the entire system connecting different component with each other. This control unit interconnects the sensor interface, transmitter interface and the power management in order to synchronize and monitor the components of the system.

3.2. Design

The design of the digital IC is done by software and that is why the design explained in this paragraph is mostly about software design and the design choices.

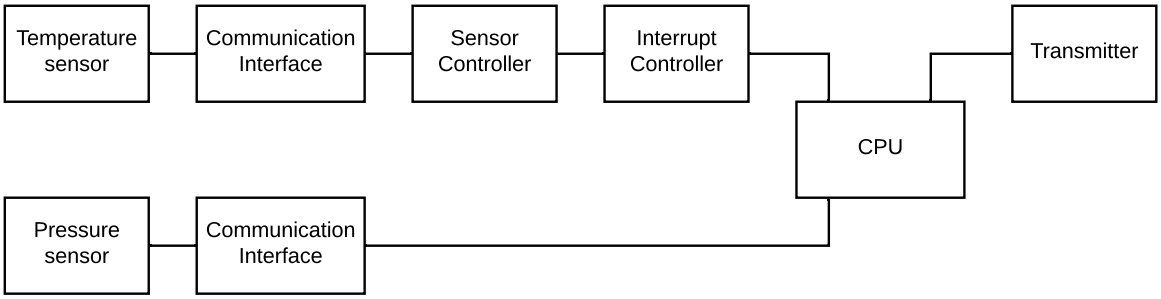


Figure 3.1: Digital IC modules

3.2.1. Real-time architecture

In order to create a real-time system, the communication between the sensor interface and the transmitter interface and processing data should be real-time. By using a real-time software architecture the control unit can respond directly if needed. By using a RTOS (real-time software architecture) the power of the sensor controller, RF core and the CPU can be managed such it consumes as low as possible power.

There are one task, one semaphore and one interrupts used in the CPU and one interrupts and task in the RF core. When the interrupt from the sensor controller is generated in the CPU, the semaphore that holds the task that captures and processes the data will be freed which afterwards the task can be run. This way processing and sending data to the RF core will not be done in an interrupt function since an interrupt function cannot be interrupted in RTOS and also the task will be run by CPU only if necessary which is done by the semaphore.

At the start of the program the CPU configures the sensors and the transmitter and opens the communications with the transmitter and the sensors. Afterwards the CPU goes to the idle mode to save power. Each time a semaphore is freed by the interrupts the task runs once and the transmitter sends the data to the antenna. And again the system goes idle. The model of the behavior of the control unit is described in the state diagram shown in figure 3.2. See appendix C for the main code of the CPU. See appendix C for the complete main code of the CPU.

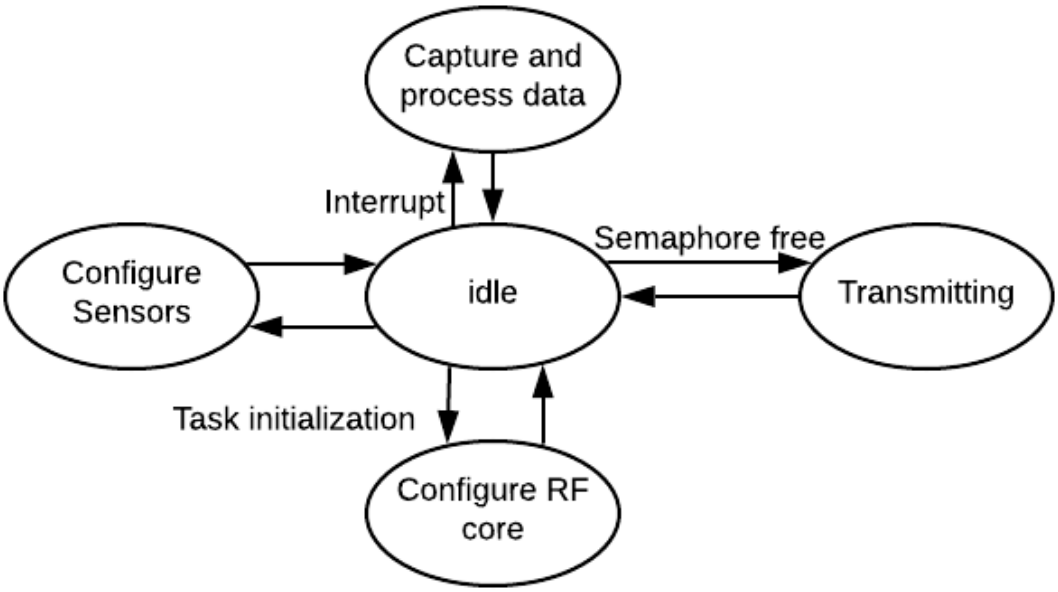


Figure 3.2: Digital IC state diagram

4

Data transmission

In this chapter the design of the wireless data transmission(RF part in chapter 3) will be discussed. Data transmission part consists of the transmission and the receiver part shown in figure 1.1. The design is based on the requirements in chapter 2. The choices made during the design and the thought process behind them will be explained in this chapter.

4.1. Selection protocol

The first design step that had to be made was to choose a transmission protocol. This had to be done since there is limited amount of time and the design of the capsule had to be done with off the shelf components. The choice between the following wireless transmission protocols has been made: Zigbee, Bluetooth, TI proprietary mode and WiFi. The TI proprietary mode is not a conventional protocol. It is a protocol supported by a lot of TI MCU's and is a highly t protocol where transmission power, Symbol Rate, Modulation types can all be modified. This is in between here since most of the low power MCUs are from TI and support this. There are of course more protocols such as Zwave, UWB or 6LowPAN. But these are not looked into since these protocols are not integrated into commercially available micro controller units or transceivers/transmitters.

Table 4.1: Comparison of different protocols

	WiFi [10]	Bluetooth [11][12]	Zigbee [11]	TI-Proprietary
Max Data Rate	1.2Gbps	2500Kbps	250Kbps	1000Kbps
Low-Power		✓	✓	✓
Carrier Frequency	2.4/5GHz	2.4GHz	2.4GHz; 915/868 MHz	433/868/915 MHz; 2.4GHz

4.1.1. Data Rate

In chapter 2 the requirements for the data rate can be found. Since the sampling rate of the sensors will be very low, about 1 Hz and a maximum of 3 Hz. All the protocols will suffice as can be seen in table 4.1.

4.1.2. Power consumption

WiFi has a high data rate but the problem is that there are no low power alternative of WiFi. This means that the requirement for lifetime of the pill would never be achieved so this choice falls of. Now three alternatives remain with low power modes. In [13] the power consumption of multiple low power wireless communications protocols is tested in which Bluetooth comes out on top of Zigbee in terms of power consumption. Now the hard part is comparing these results to the ti proprietary mode. There is no literature available about this so even tough it is known that is low power its performance versus Bluetooth an Zigbee is unknown but would probably not be a lot worse then Bluetooth or Zigbee.

4.1.3. carrier frequency

Al these different protocols work in the unlicensed wireless frequencies, which can be seen in table 4.2. From table 4.2 the frequency that are allowed here in Europe can be found which is useful to know before choos-

ing components. As can be seen in table 4.1 most of the protocols do use the 2.4 GHz frequency band. This has a problem which is that Electromagnetic waves in this frequency band get absorbed a lot by water [14]. Since the human body consists mostly of water there is interference of the human body outside of the frequency range of 1 MHz - 1GHz [15] [16]. So the optimal Protocols would be Zigbee or TI proprietary mode in spite of this there has been looked into transmission power needed for Zigbee transmission for capsule endoscopy [14]. Using this information 2.4GHz transmission can still be used the problem know is that the power consumption will be higher.

Table 4.2: Relevant unlicensed wireless frequencies

Wireless Frequency	Frequency	Bandwidth	Country/Region
High Freq. ISM	2400-2483 MHz	20 MHz	Worldwide
	5725-5875 MHz	40 MHz	
	433-434 MHz	KHz Range	Worldwide
	315 MHz		
Mid ISM	865-868 MHz	200-500 KHz	Europe
	902-928 MHz	MHz	U.S., Canada, Australia
UWB	3.1-10.6 GHz	>500MHz	International

4.1.4. Chosen protocol

The final chosen protocol is the TI Proprietary mode since it is possible to have wireless data transmission in the sub GHz range and have a high enough data rate. The power consumption is unknown with respect to Bluetooth an Zigbee. This is not a big problem since it is still low power and has configurable transmission power to low values much lower then Zigbee and Bluetooth.

4.2. MCU or transceiver/transmitter unit

The first choice that had to be made was to decide between a stand alone transmitter/transceiver or a MCU with a transmission part integrated since the controller unit will be a MCU. Since most MCUs without a transmission part had the same size as MCUs with transmission part it would safe space to have a MCu with a transmission part because then a transceiver/transmitter is not needed.

4.3. Chosen Component

The final chosen component is The TI CC2650 MCU as mentioned before in chapter 3. This component supports TI Proprietary mode and Bluetooth transmission. This is chosen because in case the project would extend to adding more sensing units or even a camera it would be able to still be able to implement this using the higher data rates. So the main reason was the flexibility of the project and the prototype. In the end only the TI proprietary mode is used about which more will be told in the following sections. Even tough it works in the 2.4 GHz range on the cc2650 this will still be good enough to make the pill last long enough and have a theoretical acceptable range from inside the human body.

4.4. TI Proprietary mode

In this section the TI proprietary mode will be explained, SmartRF studio and its settings will be discussed and a little of what happens inside the MCU when it should transmit something.

4.4.1. SmartRF Studio

SmartRF studio is a TI supported tool that can be used to generate configuration files for the data transmission for its MCUs. These files are then used in Code Composer Studio and included in the projects to configure the wireless data transmission. in figures 4.1 and 4.2 all the configurable parameters of the transmitted package and the received package can be seen. Also on the top of those figures the modulation type

can be seen which is 2-GFSK which stands for Gaussian frequency shift keying. In figure 4.1 and 4.2 also parameters such as the symbol rate transmission power, receiver bandwidth, carrier frequency and deviation can be seen. In this figure they are already set to their final value. The reason that this symbol rate at the end is not important is because sending a package is done manually in the code so if in the code no packet is send it will not unnecessarily send packages. So this symbol rate is not a set symbol rate but just a maximum. The symbol rate in this case thus put in there to make sure that the deviation is is at the minimum half the maximum symbol rate and the receiver bandwidth is at least twice the maximum symbol rate.

In figure 4.1 the configurations of the transmitted package can be seen which is a package with a preamble and a sync word which also can be seen in figure 4.1. Since the packet length will be fixed this is not included in the packet but in the configuration file.

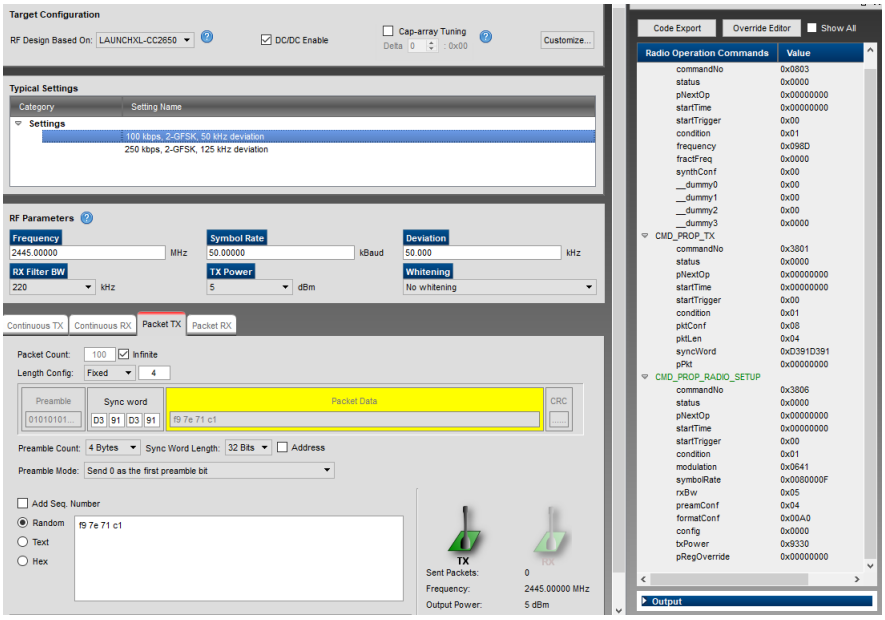


Figure 4.1: SmartRF studio Transmit package settings

In figure 4.2 also the configurations of the received package can be seen. This had to be consistent with the transmitted package because the receiver unit should expect the a package with the same overhead as the package that is being transmitted.

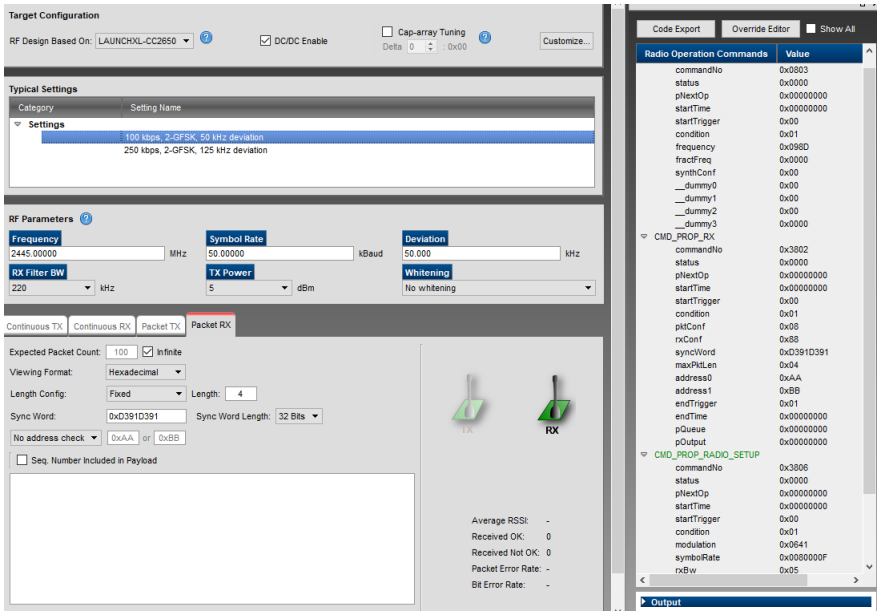


Figure 4.2: SmartRF studio Receive package settings

4.4.2. Gaussian frequency shift keying

To know why certain configurations is used a little about the modulation type used has to be explained. Gaussian frequency shift keying is a modulation technique where the digital filter first goes trough a Gaussian filter and then gets modulated as a frequency shift keying to have a better base-band bandwidth performance [17] [18]. In the smartRF settings the symbol rate is set to 50 kBaud the deviation should be at least half that to make sure there are no overlapping signals on certain frequencies where in this case is thus chosen for a deviation of 50 kHz. The occupied bandwidth of a FSK modulated signal is equal to the symbol rate + twice the deviation. Thus in this case the occupied bandwidth is equal to 50 + 100 = 150kHz. In the smart RF settings the RX bandwidth is set to 220 kHz since this was the lowest after 150 kHz.

4.5. Antenna

For selecting the antenna there were three different choices that could be made. To choice was between a chip antenna, a PCB antenna or a wire antenna[19]. Since chip antennas are the smallest possible antennas that can be found that is the antenna that is used. This is because the end goal is to work towards an endoscopic pill and is one of the trade off requirements. So thinking of size while making choices is a important part. Since in the data sheet [8] the antenna matching circuit is given for a antenna with a 50 ohm input impedance so a corresponding antenna is chosen with a 50 ohm input impedance. the antenna chosen is the Wurth Elektronik SMT Antenna 7488930245. And its matching circuit can be found in figure 4.3

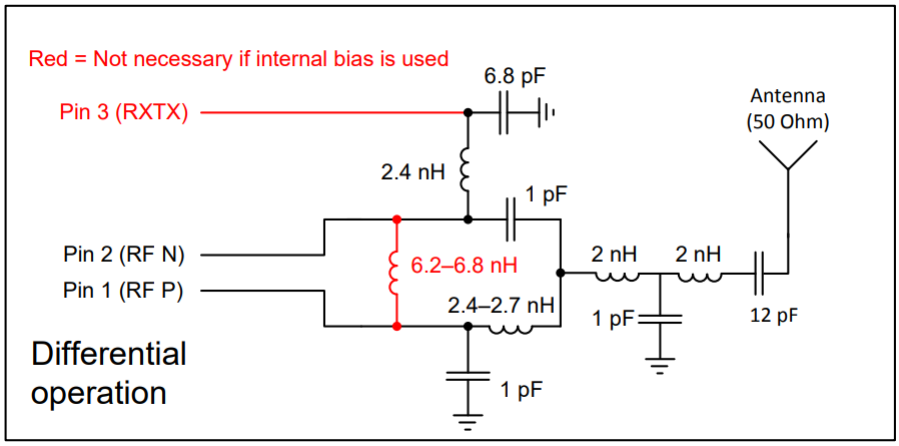


Figure 4.3: Antenna matching circuit

4.6. Implementation

Everything mentioned before in this chapter is now used when making the system work. The Transmitter was in this case just the cc2650 programmed correctly to transmit the temperature and pressure every second. The receiver unit is now a development board, the launchpad of the cc2650. The launchpad receives the transmitted package and the send it to the PC trough the serial com port using uart. The code for the cc2650 is written in code composer studio with help of the user guide [7] and coding examples in code composer studio.

4.7. Transmission power

The transmission power set is a important parameter to set the rang of the wireless data transmission. The maximum range can be estimated using friis equation which can be found in formula 4.1.

$$\frac{P_R}{P_T} = \frac{G_T G_R c^2}{(4\pi R f)^2} \tag{4.1}$$

In [14] the transmission power that is needed for Zigbee trough an animal is tested. in here the minimum needed transmission power is equal to 533 μW for a transmission range of 2 m. The assumption now is that this also works for the TI proprietary mode since the carrier frequency is almost the same and that the interference of the human body will be the same as the pig use in [14]. Rewriting equation 4.1 a bit will turn it into 4.2.

$$\frac{P_R(4\pi f)^2}{c^2 G_T G_R} = \frac{P_T}{R_{max}^2} \tag{4.2}$$

Equation 4.2 now can be used to estimate the needed transmission power for a certain range since all the parameters on the left will not change so with the information from [14] since there the ratio is known for a signal travelling trough an animals body. Where this is equal to $133.25 \cdot 10^{-6}$ in the requirements it says that the range should be 5 m so that means that the transmission power should be around 3.3mW which is around 5dBm which is set into the configuration files as can be seen in figures 4.1 and 4.2.

4.8. Transmission Flow

The sensors that are going to be used are a pressure sensor and a temperature sensor. The sensors used are TI TMP 112B temperature sensor and the TE MS5534C barometer module. The sensors each has a 2 byte output. These bytes are then send to the cc2650 and then being processed as can be seen in figure 4.4.

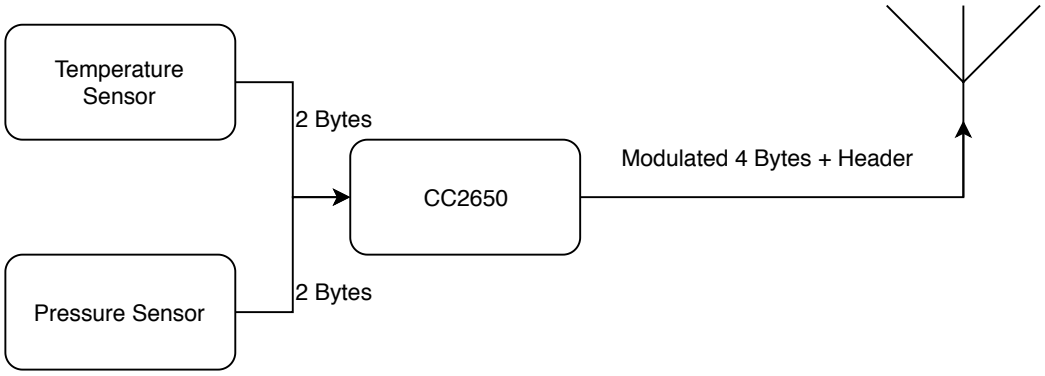


Figure 4.4: High level description of data flow

In figure 4.5 a overview of the RF-core in the cc2650 can be seen with its external dependencies. Inside the CC2650 when a transmission happens the main CPU the Cortex-M3 will send a command to the RF-core CPU a Cortex-M0. This command is then scheduled and performed by the Cortex-M0. The modern, Frequency synthesizer and RF interfaces is where the modulation happens of the digital data or the demodulation of the received signal.

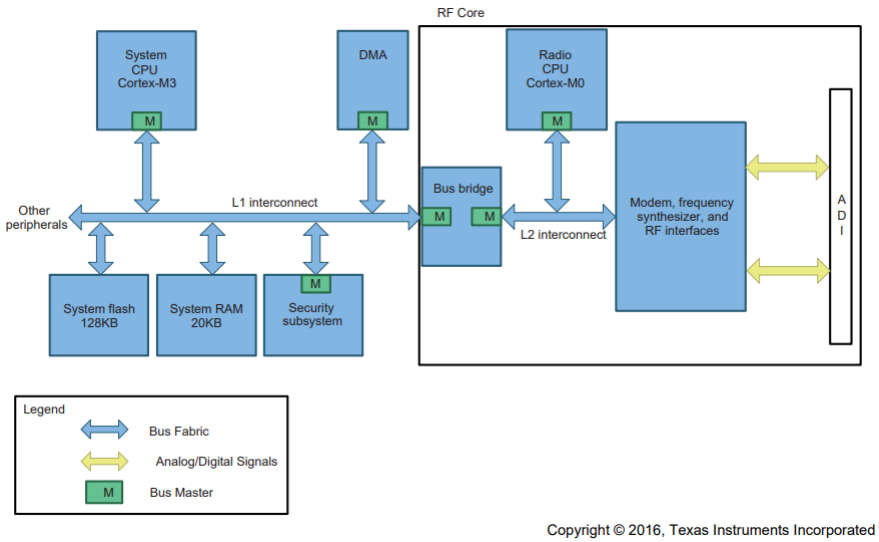


Figure 4.5: High level description RF functionality CC2650 [8]

4.9. Receiver

For the receiver a CC2650 launchpad will be used. This is the development board for the c2650 which is made by TI. The launchpad has an integrated antenna so this was very easy to implement. The flow chart of the reception can be seen in figure 4.6. This launchpad then transmits this data using UART to the PC. In the PC then this data is being processed and shown on the screen.

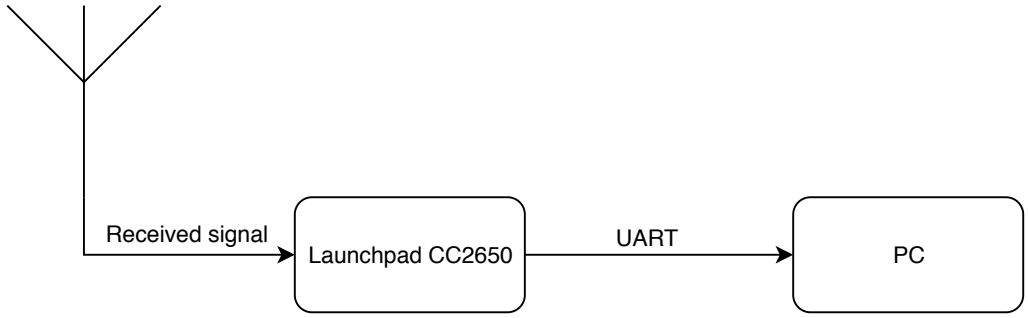


Figure 4.6: Data flow at the Receiver end

5

GUI

GUI as the last stage in the process which delivers the required output for the system will be discussed in this chapter. The Graphical User Interface illustrates the measured values by the sensors in a real-time manner. The temperature and the pressure values are given in a graph that changes with time. See figure 5.1.

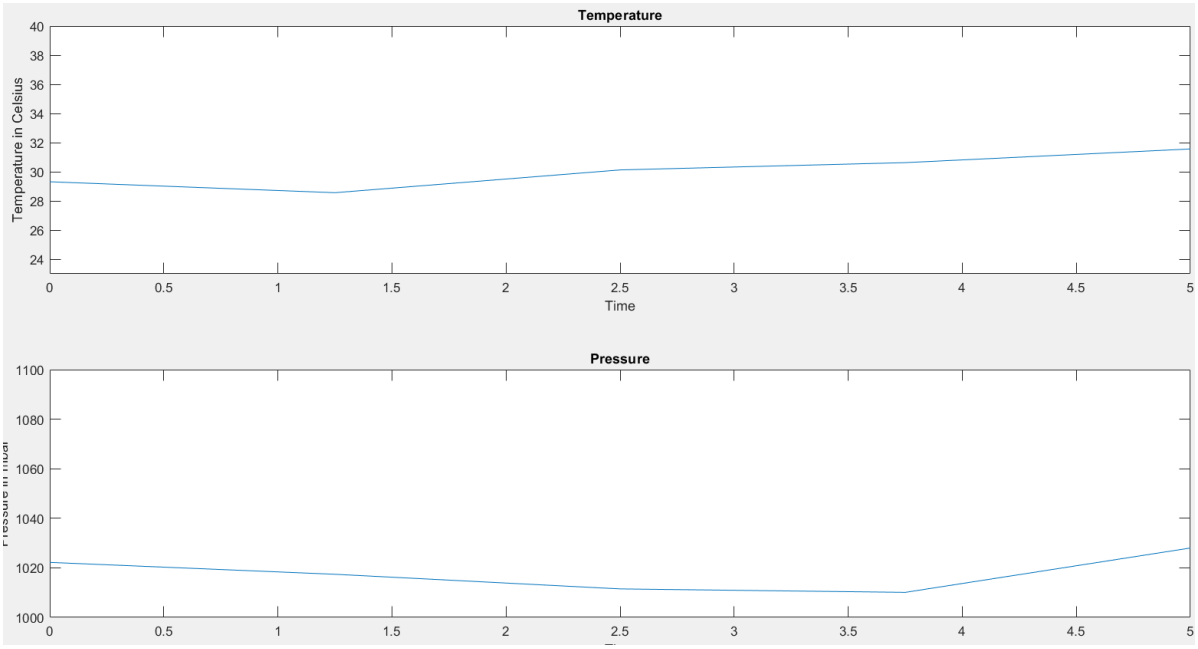


Figure 5.1: Sensor values plots

5.1. Receiver

After receiving the values from the transmitter it is sent to the the workstation where the values are illustrated. The receiver waits forever for packets to be received. Each time a packet is received an interrupt function is called where the information is extracted from the packet and sent to the GUI. The packets are sent using UART protocol to the graphical user interface where the receiver is connected to a com port using USB. See appendix D for the C code of the receiver.

5.2. Design

The GUI is written in MatLab and uses an interrupt function every time a value is received and sent to the workstation. See appendix A for the main Matlab code and appendix B for the interrupt function code.

The main code initializes and allocates the communication with the receiver and stays in an infinite loop

waiting for a packet to be received from the receiver.
The interrupt function collects four bytes data received from the UART and applies some additional calculation to combine some packets in order extract the required information from the packet. By shifting the plot or in other words deleting the first received data and adding the last received data to the plot, the plot changes every time it receives a packet from the receiver. This way the plot is considered a real-time plot of the temperature and the pressure values. A plot that changes each time a packet is received as shown in figure 5.1.

6

Testing and Prototype

In this chapter all the results will be discussed of the implementation. Also the prototype and its relevant

6.1. Data Transmission

6.1.1. Testing of the transmission

The first test was to just send one byte wireless from one launchpad to another launchpad. So for this test an additional launchpad was needed. After receiving the results are then read the results that are received er then being seen from puTTY or code composer studio on the PC. After this worked no matter what package would be send would be no problem. So now knowing that the data transmission worked multiple extra test of the characteristics of this transmission can be done.

6.1.2. Average current Consumption and Transmission time

There have been test done to see the power consumption of the MCU when it is transmitting a package. The measurement setup is a 10 ohm resistor in series with the MCU with a oscilloscope measuring the voltage with a x10 probe. It is a 10 ohm resistor because that was the lowest resistance that was available and a 10x probe so that the voltage would be equal to the current so that the current would be immediately measured. This setup can be seen in figure 6.1.

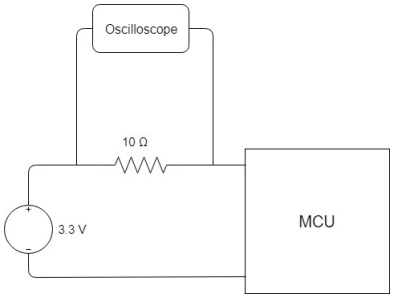


Figure 6.1: Setup measurement power consumption transmission

Using this setup the MCU was just sending data without doing anything else. There where two measurements done one while sending 1 Byte and 1 sending 100 Bytes with a transmission power of 5 dBm. These results can be found in figures 6.2 and 6.3.

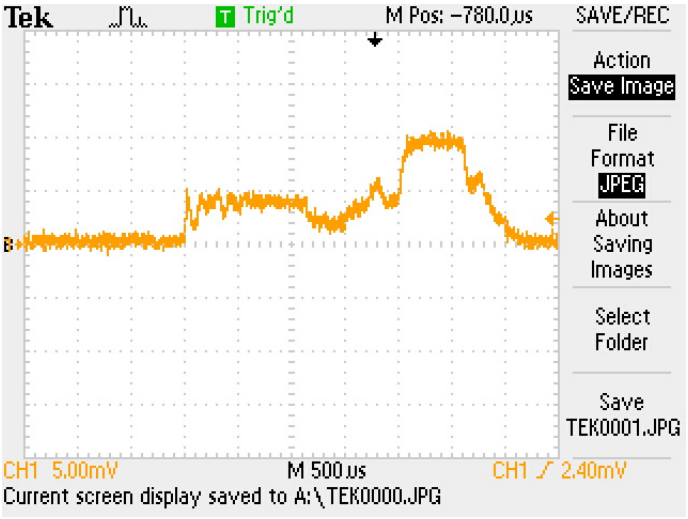


Figure 6.2: Current measurement sending 1 Byte

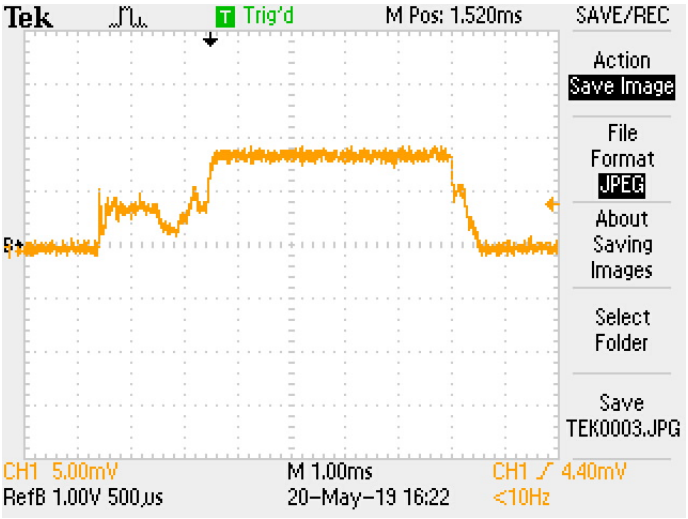


Figure 6.3: Current measurement sending 100 Byte

Using figures 6.2 and 6.3 an estimation can be made for the average current. In this estimation figure 6.2 is the transmission of just the package header. In both figures the pre-transmission period can be seen. This is the period where the preparation for the transmission happens such as the modulation. This period takes 2ms and has an average current of about 4mA. After this pre-transmission period it will start transmitting data with a estimated current of 10mA. From figure 6.2 it is possible to see that the MCU transmits for about 0.6ms. So as a rough estimate the transmission of the header and the CRC checksum, 0.55 ms will suffice as transmission time of the header since it has a payload of 1 byte. To have a good estimation for the average current per package sent per second and the transmission time for every payload length. In figure 6.3 it can be seen that when sending 100 bytes plus the header that it takes about 5.2 ms so when removing the header length to that 4.65 ms is left for 100 bytes so it takes an extra 0.0465ms for every byte send. with this information the formula for the transmission time(6.1) and the average current(6.2) can be constructed. Where the n is the amount of bits per payload and k the amount of packages per second. So for a payload of 4 bytes and 1 package per second a average current of 23.36 μA and a transmission time of 2.736ms.

$$T_{TX} = 2.55 + 0.046 \cdot n \tag{6.1}$$

$$I_{TX}^{AVG} = ((2ms \cdot 4mA) + ((0.55ms + 0.046ms \cdot n) \cdot 10mA)) \cdot k \tag{6.2}$$

6.2. Digital IC

In this chapter the test of the digital IC will be discussed. The test of this component plays a main role since it controls other parts of the system.
First the values received from the sensor interface and the output to the transmitter interface needs to be tested. This intermediate steps will show if the module generates the desired output and if it captures the input.
Then the software architecture is tested to show if it delivers real-time behavior and switches to low power modes.

From the plot shown in figure 5.1 the temperature and the pressure values detected by the sensors can be observed. From this plot it can be concluded that the output values of the system are as required. By changing the temperature and the pressure the graph responses with the correct values using the setup as shown in figure 6.4.

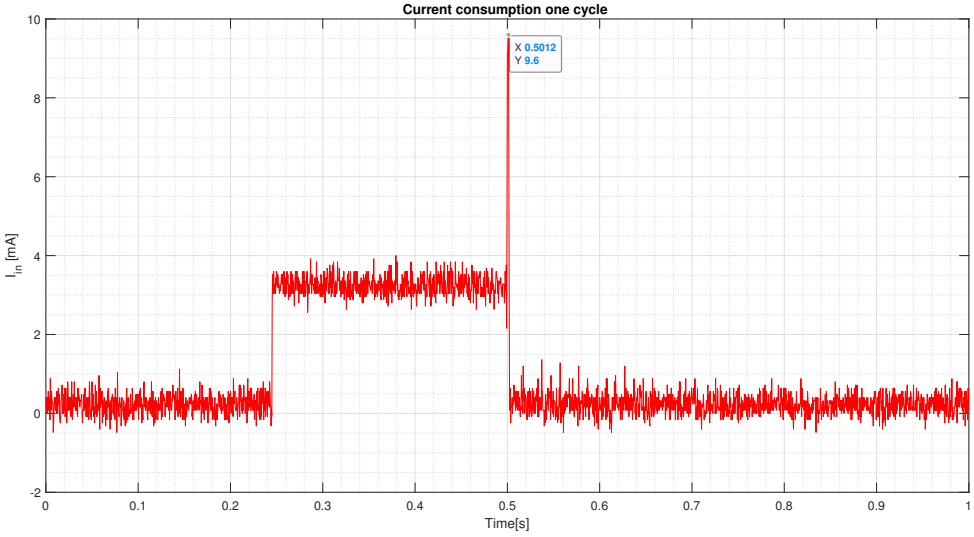


Figure 6.4: Current consumption of the endoscopic capsule during the process

The graph in figure 6.4 shows the current during a period of time by using a 10 ohm resistance to measure the current. When the CPU is idle the current is approximately 0A and that is required for the system in order to save power. At a certain moment the current jumps from 0A to 0,4mA when the digital IC is capturing data, processing it and sent to the transmitter. The peak shown in the plot before the system goes idle is when the system is transmitting data and the current that is used at that moment is approximately 1,6 mA. From this it can be concluded that the digital IC is working as required in terms of functionality and power efficiency.

As tools the debugger and a test board is used to test the digital IC during the design. These tools is used as intermediate test setups in order to verify the design choices. All in all the endoscopic capsule works as desired and the next chapters will continue to discuss the test results.

6.3. Prototype

6.3.1. Breadboard prototype

The breadboard prototype consists out of the cc2650 launchpad development board as substitute of the stand alone cc2650, the antenna and its matching circuit. All the sensors are connected to the launchpad which then sends the data received to another launchpad which is connected to the PC that then shows the data on the Matlab GUI. To make the whole system complete everything now is also powered by batteries and the converters so that this is a stand alone system. This can be seen to be working again when looking at the figure 5.1. The power consumption of the whole system can be seen in figure 6.5.

Using figure 6.5 to get an average current consumption of the prototype is hard because of all the noise due to the converter. So to get an average current multi-meters were used which measured a average current of 0.83mA. Due to the batteries having a capacity of 30 mAh the system will work for a little longer then 60 hours because of the efficiency of the converters(85.1%) and the 2 batteries (30mAh with 3.3V each) in series that will supply 198mWh. This means the requirement for the lifetime of the system mentioned in chapter 2 is achieved.

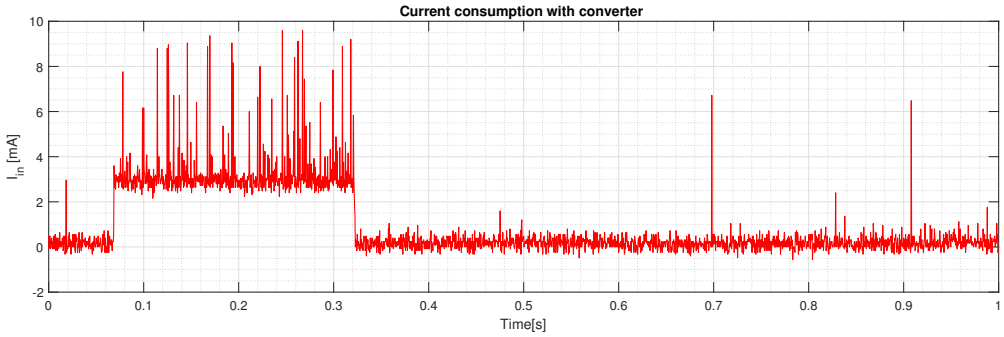


Figure 6.5: Current consumption breadboard prototype

6.3.2. PCB Prototype

The was also an attempt to creating a PCB Prototype, the PCB design and the 3D-design of a capsule are already finished. In figure 6.6 the designed pcb can be seen with the highlighted part is the data and management groups part on the pcb. On the PCB the biggest part of the data management and control part is the matching circuit for the antenna. The PCB has already arrived and the 3D-Design is already printed but the assembly has to be done still. So also no test have been done for this prototype. See figure 6.7 for the currently product.

Figure 6.6: Designed PCB prototype

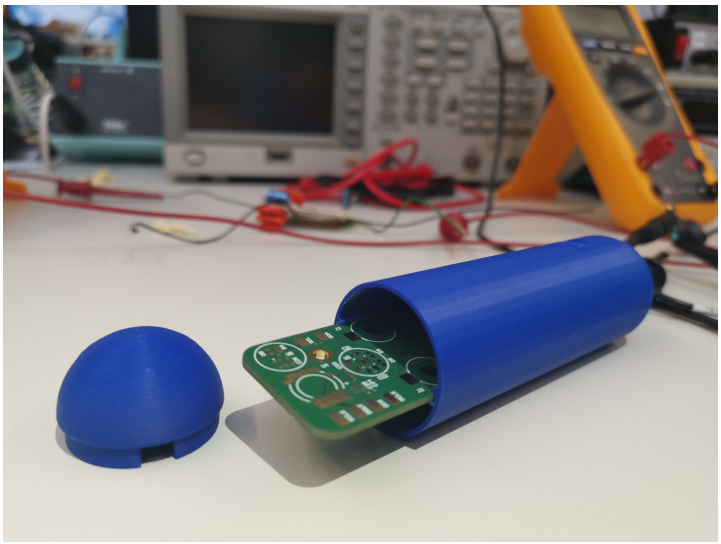


Figure 6.7: The endoscopic capsule

7 Discussion

In the beginning of the project after the components were selected a bunch of programming had to be done. All the libraries needed had to be found on known how to be used. Everything is explained in the user guide of the CC2650 MCU but it took a long time to complete. In the end the programming worked out in time with the help of some by ti provided examples that made understanding the user manual a lot easier.

In the end the range of the transmission from inside the human body has not been tested even tough there has been thought about it even in the 2.4GHz frequency range where there is a lot of absorption of water. Also there was not a way to isolate the MCU and test it separately. So instead having a clear separate test for the MCU the implementation was the test for the MCU. In the end the capsule also was not in a size required for a pill which was unfortunate due to time restrictions. There were also multiple thing that would be interesting to test such as the latency from acquiring the data to showing it in the GUI.

The prototype itself in the end was working but unfortunately not to many test have been since there has been a lot of work done in improving the system and in the end the test were a little bit neglected and also imitating a human body is hard to do. So even tough the system works and a lot of the requirements have been met, the performance of the system can not be specified.

7.1. Teamwork

There were three teams working on this project together on completing the complete system mentioned in the introduction. To make sure some of the requirements were met a lot of communication had to be done in between sub groups. Especially for the data management and control part since here everything is combined and then send to the PC.

To make sure the power management would be done correctly also a lot of communication between the power management group had to be done with the other subgroups. Predicted current consumption of their respective part was given so that the power management group a estimation of the consumed power by the system. So in the case of the data management and control group for example the power consumption test in chapter 6.

From the sensor group the data management and control group needed to know how the raw data would look like. So that the raw data that would be received could be converted into understandable data.

8

Conclusion & Future work

8.1. conclusion

In this thesis the first steps towards the data management and control for a complete endoscopic capsule have been made. Where the digital IC of the system is implemented using a MCU of TI. Transmitting is also done using the same MCU where the modulation type is set to Gaussian frequency shift keying with a deviation of 50KHz and a receiver bandwidth of 220 KHz. The MCU makes the pressure and the temperature sensor measure their value and sends them once a second to the receiver which then sends the data to the serial port and matlab makes a real time plot of this data. For the receiver unit The development board of TI, the launchpad for the cc2650 is used. When looking at the requirements in chapter 2 all the mandatory requirements have been accomplished. All the mandatory requirements were that the data must be transmitted wirelessly. Another mandatory requirement was that the data must be able to be seen in real time. the final mandatory requirement was for the system to have at least on sensing unit. The data is being transmitted wireless using TI Proprietary mode, the data can be followed real time which happens in the matlab GUI and that there are now two sensing units instead of the mandatory one sensing unit. Of all the trade off requirements the capsule has managed to comply to a few but not all. One of the trade-off requirements that has been accomplished is that the capsule now has two sensing units the pressure sensor and temperature sensor. And another is that the system can run for 24 hours. All the other trade off requirements unfortunately could not be accomplished in the limited time period.

8.2. Future Work

In this section the ways to further improve the work that has been discussed in this thesis and maybe future developments of the endoscopic pill.

8.2.1. Data transmission

For the Data transmission the first thing that has to be done is change the carrier frequency so that less of the signals strength will be absorbed by the water in the human body. This is a quick but effective strategy to lessen the power used in the data transmission.

Another Way to reduce the size is instead of using a MCU for the digital IC and transmission to have self designed ASIC and transmitter in the pill to save space.
As mentioned in the introduction UWB can be very uesfull for very high resolution images so looking into how to improve the designs that are being used know by following the tips in [6].

8.2.2. Digital IC

In the current design a micro-controller is used as the digital integrated circuit of the capsule which interfaces with the sensors and the actuators and synchronizes and controls the system. The real-time software architecture is used for the design of the control unit to guarantee real-time response and multi-tasking if needed. The down-side of a CPU is that it does not actually delivers a real-time behavior but by using the architecture the system is real-time enough for the design purposes.

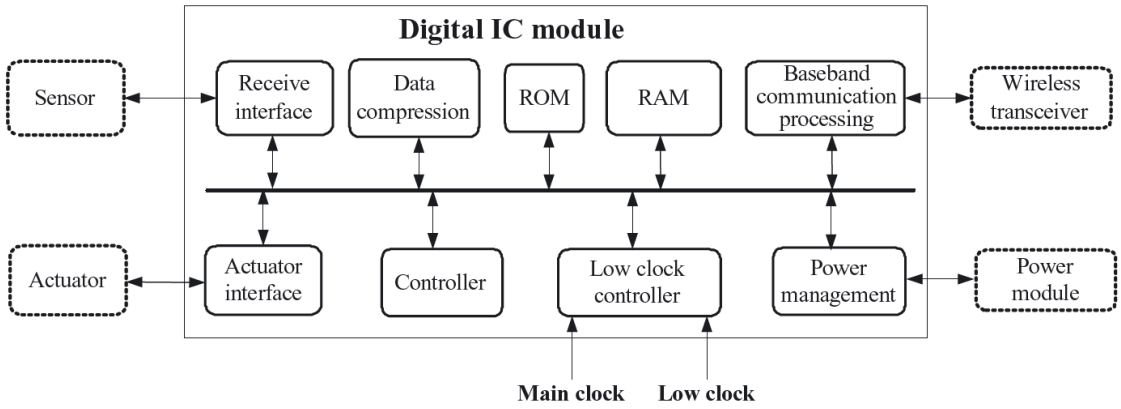


Figure 8.1: Digital IC modules [20]

A CPU uses assembly instructions that are executed one by one where on the other a logic based integrated circuit would execute more instructions at the same time. CPU uses also more power than a logic based circuit since it is a general purpose control unit and a logic based circuit would use only power needed to function. Because a micro-controller is a general purpose unit it is also larger than a logic based circuit in size. That is why a logic based digital IC would be a better design if there was more time and resources for the project.

The state diagram or the behavior of the control unit would not be different than one used for the micro-controller. See figure 8.2 for the state diagram which can be used for the logic based circuit. The power saving would be different in a logic based design. A RTOS uses software to jump to different modes and is micro-controller dependent how the low power mode functions. A logic based circuit saves power by using different clock signals for different modules and states. See figure 8.3.

The digital IC consists of several modules as described in the previous chapters and the design of those modules is different to realise in a micro-controller since the general purpose processor is pre designed and it is not open for modifications. On the other in a logic based design the modules can be designed independently and efficient as possible. The circuit would interconnect the modules and using a controller to send control signals to each module at a efficient way. See figure 8.1.

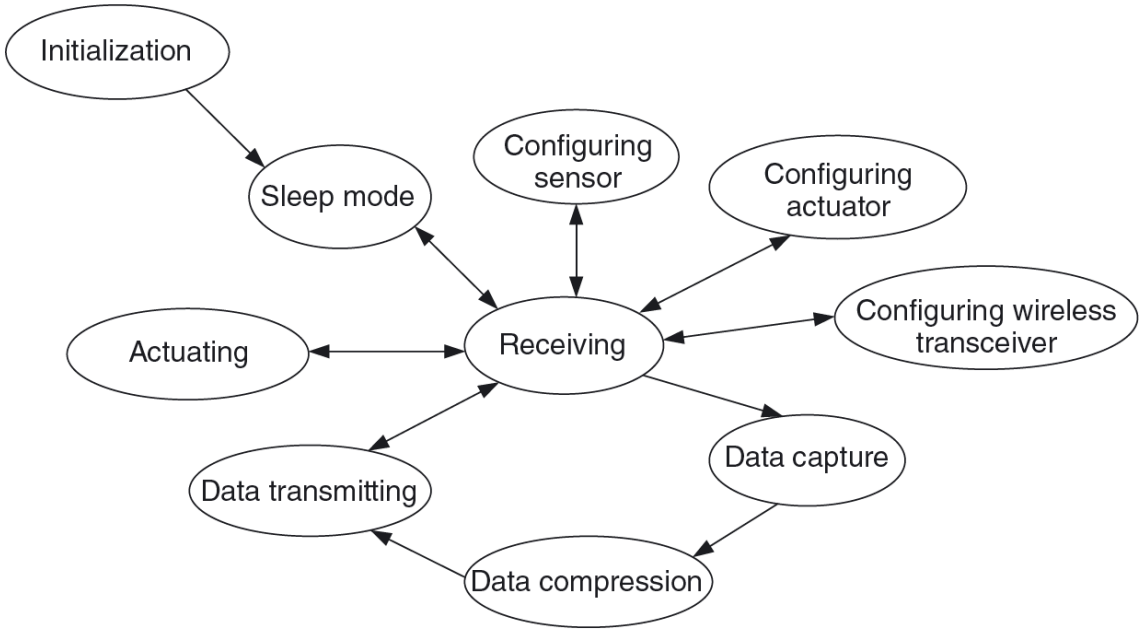


Figure 8.2: State diagram of the controller [20]

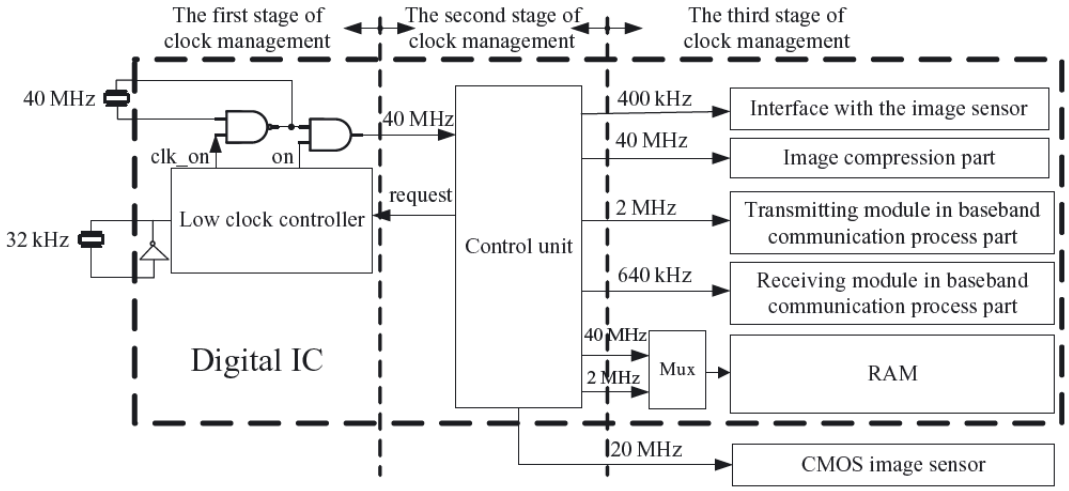


Figure 5.14. The clock management architecture in the wireless endoscopy capsule.

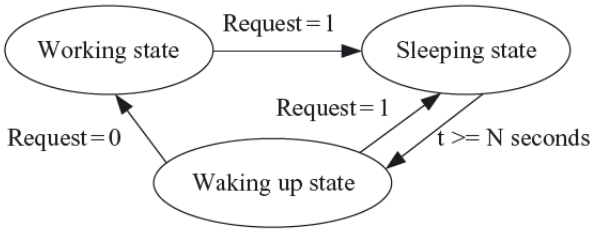


Figure 5.15. The state machine in a low clock controller.

Figure 8.3: Clock management of the digital IC module [20]

Bibliography

[1] ASGE, *Understanding capsule endoscopy*, https://www.asge.org/home/for-patients/patient-information/understanding-capsule-endoscopy.

[2] L. E. Moore, "The advantages and disadvantages of endoscopy", *Elsevier*, vol. 18, pp. 250–253, 2003. DOI: 10.1016/S1096-2867(03)00071-9.

[3] M. Muñoz-Navas, "Capsule endoscopy", *World J Gastroenterol*, vol. 15(13), pp. 1584–1586, 2009.

[4] M. R. Yuce and T. Dissanayake, "Easy-to-swallow wireless telemetry", *IEEE Microwave Magazine*, vol. 13, no. 6, pp. 90–101, 2012, ISSN: 1527-3342. DOI: 10.1109/MMM.2012.2205833.

[5] D. C. Zoya Popovic Erez Falkenstein and R. Zane, "Low-power far-field wireless powering for wireless sensors", *Proceedings of the IEEE*, vol. 101, pp. 1397–1409, 2013. DOI: 10.1109/JPROC.2013.2244053.

[6] R. Chávez-Santiago, J. Wang, and I. Balasingham, "The ultra wideband capsule endoscope", in *2013 IEEE International Conference on Ultra-Wideband (ICUWB)*, 2013, pp. 72–78. DOI: 10.1109/ICUWB.2013.6663825.

[7] TI, *Cc13x0, cc26x0 simplelink™ wireless mcu technical reference manual (rev. h)*, English, Texas Instruments, 1741 pp., february, 2015.

[8] ——, *Cc2650 simplelink™ multistandard wireless mcu datasheet (rev. b)*, English, Texas Instruments,

[9] W. Elektronik, *7488930245 specification*, English, 3 pp.

[10] W. Alliance, "Wi-fi certified™ n: Longer-range, faster-throughput, multimedia-grade wi-fi® networks", 2009.

[11] J. Lee, Y. Su, and C. Shen, "A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi", in *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*, 2007, pp. 46–51. DOI: 10.1109/IECON.2007.4460126.

[12] K. Ren, *Exploring bluetooth 5 – how fast can it be?*, May 2019. [Online]. Available: https://www.bluetooth.com/blog/exploring-bluetooth-5-how-fast-can-it-be/.

[13] A. Dementyev, S. Hodges, S. Taylor, and J. Smith, "Power consumption analysis of bluetooth low energy, zigbee and ant sensor nodes in a cyclic sleep scenario", Apr. 2013, pp. 1–4. DOI: 10.1109/IEEE-IWS.2013.6616827.

[14] P. D. Pietro Valdastri Arianna Menciassi, "Transmission power requirements for novel zigbee implants in the gastrointestinal tract", *IEEE Transactions on Biomedical Engineering*, pp. 1705–1710, 2008. DOI: 10.1109/TBME.2008.919117.

[15] *Water and microwaves*, http://www1.lsbu.ac.uk/water/microwave_water.html, 2001.

[16] P. Valdastri, A. Menciassi, A. Arena, C. Caccamo, and P. Dario, "An implantable telemetry platform system for in vivo monitoring of physiological parameters", *IEEE Transactions on Information Technology in Biomedicine*, vol. 8, no. 3, pp. 271–278, Sep. 2004, ISSN: 1089-7771. DOI: 10.1109/TITB.2004.834389.

[17] S. H. Gerez, *Implementation of digital signal processing: Some background on gfsk modulation*, https://wwwhome.ewi.utwente.nl/~gerezsh/sendfile/sendfile.php/gfsk-intro.pdf?sendfile=gfsk-intro.pdf, Online PDF file.

[18] S. T, M. Herceg, and T. Matić, "A simple signal shaper for gmsk/gfsk and msk modulator based on sigma-delta look-up table", *Radioengineering*, Jun. 2009.

[19] T. Pattnayak and G. Thanikachalam, "Antenna design and rf layout guidelines", Cypress Semiconductors, Tech. Rep., 2014.

[20] K. Iniewski, *LaTeX: CMOS Biomicrosystems*, 1st ed. John Wiley and Sons Ltd, 1994.

A

Main GUI code

```
close all
clear
clc

% Global variables
timeInterval = 5;

% Open port
delete(instrfindall); %Solved unavailable port error
MyPort = serial('COM11');
MyPort.BytesAvailableFcnCount = 1;
MyPort.BytesAvailableFcnMode = 'byte';
MyPort.BytesAvailableFcn = {@mycallback, timeInterval};
fopen(MyPort);
disp(MyPort)
disp('Start reading')

global temp;
global count;
global press;
press = zeros(timeInterval, 1);
temp = zeros(timeInterval, 1);
count = 1;

% Start/Stop button
 c = uicontrol('Style','pushbutton','String','Start/Stop','Callback', {@pushbutton_callback, MyPort});

% Infinite loop
while(true)
end

% Deallocating memories
fclose(MyPort); %Disconnect port
delete(MyPort); %Remove port from memory
clear MyPort;   %Clean port object from workspace
```

B

Interrupt GUI code

```
function mycallback(obj,~, length)
    global temp;
    global count;
    global press;
    x = linspace(0,length,length);

    tmp = fread(obj,1);

    if(tmp>0)
        Data = fread(obj,3);
        temperature = tmp*(2^8)+Data(1);
%         temperature = temperature/10; % temperature pressure sensor
        temperature = temperature * 0.0625;
        pressure = Data(2)*(2^8)+Data(3);
        pressure = pressure / 10;
        fprintf('Temperature: %.2f Celsius\n Pressure: %.2f mbar\nCount: %d\n\n', temperature, pressure,
     count);

        if(temp(length)==0)
            temp(count) = temperature;
            press(count) = pressure;
        else
            i=1;
            while(i<length)
                temp(i) = temp(i+1);
                press(i) = press(i+1);
                i = i+1;
            end
            temp(length) = temperature;
            press(length) = pressure;
        end

        subplot(2,1,1);
        plot(x,temp)
        ylim([23 40])
        xlim([0 length])
        xlabel('Time')
        ylabel('Temperature in Celsius')
        title('Temperature');
        subplot(2,1,2);
        plot(x,press)
        ylim([1000 1100])
        xlim([0 length])
        xlabel('Time')
        ylabel('Pressure in mbar')
        title('Pressure');
        drawnow;
        count = count + 1;

    end
```

```
50
51 end
```

C

Transmitter code

```
1
2  /*
3   * Titel : Transmitter
4   * Module: Digital IC of the pill
5   * Author: Group E
6   */
7
8  /* XDCtools Header files */
9  #include <xdc/std.h>
10 #include <xdc/runtime/System.h>
11
12 /* BIOS Header files */
13 #include <ti/sysbios/BIOS.h>
14 #include <ti/sysbios/knl/Clock.h>
15 #include <ti/sysbios/knl/Task.h>
16
17 /* TI-RTOS Header files */
18 // #include <ti/drivers/I2C.h>
19 #include <ti/drivers/PIN.h>
20 // #include <ti/drivers/SPI.h>
21 #include <ti/drivers/UART.h>
22 // #include <ti/drivers/Watchdog.h>
23
24 /* Board Header files */
25 #include "Board.h"
26 #include <scif.h>
27 #include <driverlib/rf_prop_mailbox.h>
28
29 /* SmartRF settings */
30 #include "smartrf_settings/smartrf_settings.h"
31 #include <driverlib/aon_ioc.h>
32
33
34 #define TASKSTACKSIZE       1024
35 #define TX_TASK_PRIORITY    3
36 #define TMP_TASK_PRIORITY   2
37 #define PR_TASK_PRIORITY    1
38 #define TMP_ARGUMENT        100000 / Clock_tickPeriod
39 #define PR_ARGUMENT         100000 / Clock_tickPeriod
40 #define TX_ARGUMENT         100000 / Clock_tickPeriod
41 #define PAYLOAD_LENGTH      30
42 #define N 100
43 #define wait_time 10000000
44 #define PRESSURE_SAMPLES    4
45
46 /* Task variables */
47 Char txTaskStack[TASKSTACKSIZE];
48 Task_Struct txTask;
49 Task_Params taskParams;
50
```

```
51 /* Semaphore variables */
52 Semaphore_Struct semStruct1;
53 Semaphore_Handle semTX;
54 Semaphore_Params semParams;
55
56 /* RF variables */
57 RF_Params rfParams;
58 RF_Object rfObject;
59 RF_Handle rfHandle;
60
61 /* UART variables */
62 UART_Handle uart;
63 UART_Params uartParams;
64
65 /* Temporary packet variables */
66 uint32_t time;
67 uint8_t packet[PAYLOAD_LENGTH];
68
69 /* Pin driver handle */
70 PIN_Handle ledPinHandle;
71 PIN_State ledPinState;
72 PIN_Config ledPinTable[] = {
73                                 Board_LED0 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX
      , // Red led
74                                 Board_LED1 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX
      , // Green led
75                                 IOID_0 | PIN_INPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,         //
     DOUT
76                                 IOID_12 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
        // SCLK
77                                 IOID_15 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
        // DIN
78                                 IOID_21 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
        // MCLK
79                                 PIN_TERMINATE
80 };
81
82 int temperature;
83 int pressure;
84
85
86 /* reading pressure sensor: */
87
88 unsigned int coefficients_[6];
89
90 // send command MS bit first
91 void SendCommand(unsigned long cmd, size_t nbits)
92 {
93     while(nbits--)
94     {
95         if(cmd & (unsigned long)(1 << nbits))
96             PIN_setOutputValue(ledPinHandle, IOID_15, 1);
97         else
98             PIN_setOutputValue(ledPinHandle, IOID_15, 0);
99
100         PIN_setOutputValue(ledPinHandle, IOID_12, 1);
101         PIN_setOutputValue(ledPinHandle, IOID_12, 0);
102     }
103 }
104
105 /* Reset the sensor */
106 void ResetSensor()
107 {
108     SendCommand(0x155540, 21); // 1010101010101010 + 00000
109 }
110
111 /* Read one word from the sensor */
112 unsigned int ReadWord(void)
113 {
114     unsigned int w;
115     unsigned int clk = 16;
```

```
116     w = 0;
117     while(clk--)
118     {
119         PIN_setOutputValue(ledPinHandle, IOID_12, 1);
120         PIN_setOutputValue(ledPinHandle, IOID_12, 0);
121         w |=  (PIN_getInputValue(IOID_0) << clk);
122     }
123     PIN_setOutputValue(ledPinHandle, IOID_12, 1);
124     PIN_setOutputValue(ledPinHandle, IOID_12, 0);
125 
126     return w;
127 }
128 
129 /* Read the coefficient from the sensor */
130 size_t ReadCoefficient(unsigned char addr)
131 {
132     // 111 + 6bit coeff addr + 000 + 1clk(send0)
133     unsigned long cmd = (unsigned long)0x1C00 | (((unsigned long)addr) << 4);
134     SendCommand(cmd,13);
135     return ReadWord();
136 }
137 
138 /* Read the coefficients from the sensor */
139 void ReadCoefficients(void)
140 {
141     unsigned int wb = ReadCoefficient(0x16);
142     unsigned int wa = ReadCoefficient(0x15);
143 
144     coefficients_[0] = (unsigned int)((wa >> 1) & (unsigned int)0x7FFF);
145     coefficients_[4] = (unsigned int)(((wa & 0x1) << 10) | ((wb >> 6) & (unsigned int)0x3FF));
146     coefficients_[5] = (unsigned int)(wb & 0x3F);
147 
148     wb = ReadCoefficient(0x1A);
149     wa = ReadCoefficient(0x19);
150 
151     coefficients_[3] = (unsigned int)((wa >> 6) & 0x3FF);
152     coefficients_[1] = (unsigned int)(((wa & 0x3F) << 6) | (wb & 0x3F));
153     coefficients_[2] = (unsigned int)((wb >> 6) & 0x3FF);
154 
155 #ifdef DEBUG
156     //    for(size_t i=0; i<6; ++i)
157     //        {
158     //            Serial.print(Coefficient  );
159     //            Serial.print(i + 1, DEC);
160     //            Serial.print(  :  );
161     //            Serial.println(coefficients_[i], DEC);
162     //    }
163 #endif
164 }
165 
166 /* Calculate the pressure value */
167 long ConvertPressureTemperature()
168 {
169     const long UT1  = (coefficients_[4] << 3) + 20224;
170     const long dT   = (long)temperature  - UT1;
171     const long TEMP = 200 + ((dT * (coefficients_[5] + 50)) >> 10);
172     const long OFF  = (coefficients_[1] <<2) + (((coefficients_[3] -512) * dT) >> 12);
173     const long SENS = coefficients_[0]  + ((coefficients_[2] * dT) >> 10)  + 24576;
174     const long X    = ((SENS* ((long)pressure  - 7168)) >> 14) - OFF;
175     pressure    = ((X * 10) >> 5) + 2500;
176     temperature = TEMP;
177 
178     long T2 = 0, P2 = 0;
179     if(TEMP < 200)
180     {
181         T2 = (11 * (coefficients_[5] + 24) * (200 - TEMP) * (200 - TEMP)) >> 20;
182         P2 = (3 * T2 * (pressure - 3500)) >> 14;
183         pressure    = pressure - P2;
184         temperature = temperature - T2;
185     }
186 
```

```
187     return pressure;
188 }
189
190 /* Read one sample from the sensor */
191 void TriggerTemperatureSample(void)
192 {
193     // 111 + 1001 + 000 + 2clks(send 0)
194     ResetSensor();
195     SendCommand(0xF20, 12);
196 }
197
198 void TriggerPressureSample(void)
199 {
200     // 111 + 1010 + 000 + 2clks(send 0)
201     ResetSensor();
202     SendCommand(0xF40, 12);
203 }
204
205 /* Read the average value by reading multiple samples */
206 void AcquireAveragedSampleCm(const size_t nSamples)
207 {
208     long pressAccum = 0;
209     int n;
210     for(n = nSamples; n; n--)
211     {
212         TriggerTemperatureSample();
213         while(PIN_getInputValue(IOID_0))
214             ;
215         temperature = ReadWord();
216         TriggerPressureSample();
217         while(PIN_getInputValue(IOID_0))
218             ;
219         pressure = ReadWord(); // read pressure
220         pressAccum += ConvertPressureTemperature();
221     }
222     long pressAvg = pressAccum / nSamples;
223     pressure = pressAvg;
224 }
225
226 /* acquire the pressure value from the sensor */
227 void calc_pressure(){
228     AcquireAveragedSampleCm(PRESSURE_SAMPLES);
229 }
230
231 /* reading temperature  sensor: */
232
233 // SCIF driver callback: Sensor Controller task code has generated an alert interrupt
234 void scTaskAlertCallback(void) {
235
236     scifClearAlertIntSource();
237     calc_pressure(); //reading pressure sensor
238     temperature = scifTaskData.tmp112.output.value; //reading temperature sensor (from Sensor Controller)
239     Semaphore_post(semTX); //run the transmitter task
240     scifAckAlertEvents();
241 }
242
243 /* Task: */
244
245 /* Transmitter task */
246 void TaskFunction(UArg arg0, UArg arg1){
247
248     //initialisation of the rf parameters
249     RF_Params_init(&rfParams);
250     RF_cmdPropTx.pktLen = PAYLOAD_LENGTH;
251     RF_cmdPropTx.pPkt = packet;
252     RF_cmdPropTx.startTrigger.triggerType = TRIG_ABSTIME;
253     RF_cmdPropTx.startTrigger.pastTrig = 1;
254     RF_cmdPropTx.startTime = 0;
255     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup, &rfParams);
256     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
257
```

```
258     while(1){
259         Semaphore_pend(semTX, BIOS_WAIT_FOREVER); //blocking the task
260 
261         packet[0] = (uint8_t)(temperature >> 8);
262         packet[1] = (uint8_t)(temperature);
263         packet[2] = (uint8_t)(pressure >> 8);
264         packet[3] = (uint8_t)(pressure);
265 
266         /* sending data to the antenna */
267         RF_CmdHandle cmdHandle = RF_postCmd(rfHandle, (RF_Op*)&RF_cmdPropTx, RF_PriorityNormal, NULL, 0);
268         if (cmdHandle<0)
269         {
270             while(1);
271         }
272         RF_EventMask result2 = RF_pendCmd (rfHandle, cmdHandle, 0);
273         RF_yield(rfHandle);
274     }
275 }
276 
277 /* Initialisations: */
278 
279 /* Tasks initialisation */
280 void tasksInit(){
281 
282     Task_Params_init(&taskParams);
283     taskParams.stackSize = TASKSTACKSIZE;
284     taskParams.priority = TX_TASK_PRIORITY;
285     taskParams.stack = &txTaskStack;
286     taskParams.arg0 = TX_ARGUMENT;
287     Task_construct(&txTask, TaskFunction, &taskParams, NULL);
288 }
289 
290 /* Semaphore initialisation */
291 void semaphoresInit(){
292 
293     Semaphore_Params_init(&semParams);
294     Semaphore_construct(&semStruct1, 1, &semParams);
295     semTX = Semaphore_handle(&semStruct1);
296 }
297 
298 /*initialisation of the pressure sensor*/
299 void pressure_init(){
300 
301     ResetSensor();
302     ReadCoefficients();
303     IOCPortConfigureSet(IOID_9, IOC_PORT_AON_CLK32K, IOC_STD_OUTPUT);
304     AONIOC32kHzOutputEnable();
305 }
306 
307 /*initialisation of the Sensor Controller and its interrupt*/
308 void temperature_init(){
309 
310     // Initialize the SCIF operating system abstraction layer
311     scifOsalInit();
312 //    scifOsalRegisterCtrlReadyCallback(scCtrlReadyCallback);
313     scifOsalRegisterTaskAlertCallback(scTaskAlertCallback);
314 
315     // Initialize the SCIF driver
316     scifInit(&scifDriverSetup);
317 
318     // Enable RTC ticks, with N Hz tick interval
319     scifStartRtcTicksNow(0x00010000 / N);
320 
321     // Start the "TMP112" Sensor Controller task
322     scifStartTasksNbl(1 << SCIF_TMP112_TASK_ID);
323 }
324 
325 /*initialisation of the pins*/
326 void pins_init(){
327     ledPinHandle = PIN_open(&ledPinState, ledPinTable);
328 }
```

```
329
330 /*
331  *  ======== main ========
332  */
333 int main(void){
334
335     Board_initGeneral();
336     tasksInit();
337     semaphoresInit();
338     pins_init();
339     pressure_init();
340     temperature_init();
341     BIOS_start(); //start RTOS kernel
342
343     return (0);
344 }
```

D

Receiver code

```
1
2  /*
3   * Titel : Receiver
4   * Module: GUI
5   * Author: Group E
6   */
7
8  /***** Includes *****/
9  #include <stdlib.h>
10 #include <xdc/std.h>
11 #include <stdint.h>
12 #include <xdc/cfg/global.h>
13 #include <xdc/runtime/System.h>
14 #include <ti/sysbios/BIOS.h>
15 #include <ti/sysbios/knl/Task.h>
16
17 /* Drivers */
18 #include <ti/drivers/rf/RF.h>
19 #include <ti/drivers/PIN.h>
20 #include <driverlib/rf_prop_mailbox.h>
21 #include <ti/drivers/UART.h>
22
23 /* Header files */
24 #include "Board.h"
25 #include "RFQueue.h"
26 #include "smartrf_settings/smartrf_settings.h"
27
28 /***** Defines *****/
29 #define RX_TASK_STACK_SIZE 1024
30 #define RX_TASK_PRIORITY   2
31
32 /* Packet RX Configuration */
33 #define DATA_ENTRY_HEADER_SIZE 8  /* Constant header size of a Generic Data Entry */
34 #define MAX_LENGTH             30 /* Max length byte the radio will accept */
35 #define NUM_DATA_ENTRIES       2  /* NOTE: Only two data entries supported at the moment */
36 #define NUM_APPENDED_BYTES     2  /* The Data Entries data field will contain:
37                                    * 1 Header byte (RF_cmdPropRx.rxConf.bIncludeHdr = 0x1)
38                                    * Max 30 payload bytes
39                                    * 1 status byte (RF_cmdPropRx.rxConf.bAppendStatus = 0x1) */
40 #define TASKSTACKSIZE      1024
41
42 /* Pin driver handle */
43 PIN_Handle ledPinHandle;
44 PIN_State ledPinState;
45 PIN_Config ledPinTable[] = {
46     Board_LED1 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
47     Board_LED2 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL | PIN_DRVSTR_MAX,
48     PIN_TERMINATE
49 };
50
```

```
51 /* RF variables */
52 Task_Params rxTaskParams;
53 Task_Struct rxTask;    /* not static so you can see in ROV */
54 uint8_t rxTaskStack[RX_TASK_STACK_SIZE];
55 RF_Object rfObject;
56 RF_Handle rfHandle;
57
58 /* Receive dataQueue for RF Core to fill in data */
59 dataQueue_t dataQueue;
60 rfc_dataEntryGeneral_t* currentDataEntry;
61 uint8_t packetLength;
62 uint8_t* packetDataPointer;
63 uint8_t packet[MAX_LENGTH + NUM_APPENDED_BYTES - 1];
64
65 /* UART variables */
66 UART_Handle uart;
67 UART_Params uartParams;
68
69 /* Buffer which contains all Data Entries for receiving data.
70  * Pragmas are needed to make sure this buffer is 4 byte aligned (requirement from the RF Core) */
71 #if defined(__TI_COMPILER_VERSION__)
72     #pragma DATA_ALIGN (rxDataEntryBuffer, 4);
73         static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
74                                                                          MAX_LENGTH,
75                                                                          NUM_APPENDED_BYTES)];
76 #elif defined(__IAR_SYSTEMS_ICC__)
77     #pragma data_alignment = 4
78         static uint8_t rxDataEntryBuffer[RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
79                                                                          MAX_LENGTH,
80                                                                          NUM_APPENDED_BYTES)];
81 #elif defined(__GNUC__)
82         static uint8_t rxDataEntryBuffer [RF_QUEUE_DATA_ENTRY_BUFFER_SIZE(NUM_DATA_ENTRIES,
83             MAX_LENGTH, NUM_APPENDED_BYTES)] __attribute__ ((aligned (4)));
84 #else
85     #error This compiler is not supported.
86 #endif
87
88 /* Rx callback function */
89 void callback(RF_Handle h, RF_CmdHandle ch, RF_EventMask e){
90
91         if (e & RF_EventRxEntryDone){
92
93             /* Extract data */
94             currentDataEntry = RFQueue_getDataEntry();
95             packetLength      = *(uint8_t*)(&currentDataEntry->data);
96             packetDataPointer = (uint8_t*)(&currentDataEntry->data + 1);
97             memcpy(packet, packetDataPointer, (packetLength + 1));
98
99             /* send data to PC */
100            UART_write(uart, packet, sizeof(packet));
101
102            PIN_setOutputValue(ledPinHandle, Board_LED0,!PIN_getOutputValue(Board_LED0));
103
104           RFQueue_nextEntry();
105         }
106 }
107
108
109 void rf_init(){
110
111     RF_Params rfParams;
112     RF_Params_init(&rfParams);
113
114     if( RFQueue_defineQueue(&dataQueue,
115                             rxDataEntryBuffer,
116                             sizeof(rxDataEntryBuffer),
117                             NUM_DATA_ENTRIES,
118                             MAX_LENGTH + NUM_APPENDED_BYTES))
119     {
120         /* Failed to allocate space for all data entries */
121         while(1);
```

```
122     }
123
124     /* Modify CMD_PROP_RX command for application needs */
125     RF_cmdPropRx.pQueue = &dataQueue;           /* Set the Data Entity queue for received data */
126     RF_cmdPropRx.rxConf.bAutoFlushIgnored = 1;  /* Discard ignored packets from Rx queue */
127     RF_cmdPropRx.rxConf.bAutoFlushCrcErr = 1;   /* Discard packets with CRC error from Rx queue */
128     RF_cmdPropRx.maxPktLen = MAX_LENGTH;        /* Implement packet length filtering to avoid
        PROP_ERROR_RXBUF */
129     RF_cmdPropRx.pktConf.bRepeatOk = 1;
130     RF_cmdPropRx.pktConf.bRepeatNok = 1;
131
132     /* Request access to the radio */
133     rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup, &rfParams);
134     RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);
135     RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropRx, RF_PriorityNormal, &callback, IRQ_RX_ENTRY_DONE);
136 }
137
138 /* Initialisation UART interface */
139 void uart_init(){
140
141     UART_Params_init(&uartParams);
142     uartParams.writeDataMode = UART_DATA_BINARY;
143     uartParams.readDataMode = UART_DATA_BINARY;
144     uartParams.readReturnMode = UART_RETURN_FULL;
145     uartParams.readEcho = UART_ECHO_OFF;
146     uartParams.baudRate = 9600;
147     uart = UART_open(Board_UART0, &uartParams);
148 }
149
150 /* Rx task function */
151 void rxTaskFunction(UArg arg0, UArg arg1){
152     rf_init();
153
154     /* forever waiting for data */
155     while(1);
156 }
157
158 void rxTask_init() {
159
160     Task_Params_init(&rxTaskParams);
161     rxTaskParams.stackSize = RX_TASK_STACK_SIZE;
162     rxTaskParams.priority = RX_TASK_PRIORITY;
163     rxTaskParams.stack = &rxTaskStack;
164     rxTaskParams.arg0 = (UInt)1000000;
165     Task_construct(&rxTask, rxTaskFunction, &rxTaskParams, NULL);
166 }
167
168 /*initialisation of the pins*/
169 void pin_init(){
170     ledPinHandle = PIN_open(&ledPinState,  ledPinTable);
171 }
172
173 /*
174  * ==========main==============
175  */
176 int main(void){
177
178     Board_initGeneral();
179     pin_init();
180     rxTask_init();
181     uart_init();
182     BIOS_start();
183
184     return (0);
185 }
```