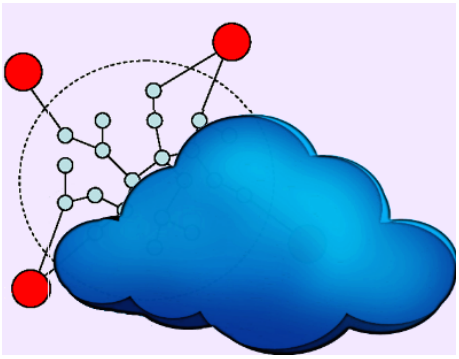


Android Based Application for Low Energy WSN

Course Instructor:
Dr. Z. Zilic

By:

MohammadHossein Askari-Hemmat

MohammadAli Askari-Hemmat

A final Report
Submitted in partial fulfillment of the requirements of
ECSE 649
Mcgill University

Winter 2014

1 Project Description :

A Wireless Sensor Network (WSN) is a wireless network consisting of sensors which can cooperate with each other to sense an environment. Each sensor is consist of a processor, a wireless transceiver, a battery and a single or multiple type of sensor (such as temperature, humidity, etc.). The task of a sensor is to collect the required data from the environment and then send this data to a Base Station(BS). Usually the sensors are limited to a small amount of power and memory resources. For this reason, the design of the sensors should be as efficient as possible to meet the requirements. Wireless Sensor Networks are used in different types of application such as military, health care, agriculture etc. Each application has its own requirement. For example, in agriculture applications, the size of the sensors has the least priority in the design phase whereas in health care applications, specially for implantable sensors, the size of the sensor is one the most important limitation.
Usually the collected data from the sensors by the BS is sent to a back-end processing machine. Figure 1 illustrates a very simple WSN architecture. In this application, senors send the collected data to the BS. The application running on the user's device will collect the data from BS and will sync it to the a cloud service. On the back-end part, the computer will analysis the collected data and will provide the user with required information.

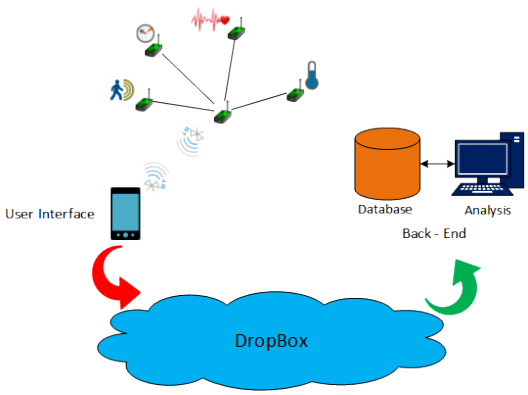


Figure 1: A simple WSN system architecture

In this project,we are going to use above architecture. Beside the hardware architecture, the network should run a protocol to be able to collect the data from the sensor correctly. For this reason a good network protocol should be implemented. In this project we used a very simple FDM (Frequency Division Multiplexing) protocol. Each sensor is associated with a unique frequency. In order to receive data from the node, BS should select the frequency that is associated to the corresponding node. As it is illustrated in figure 1, we used STAR topology for the network so there will only one BS. The network topology and architecture plus the sensor design will be explained more in details later in this report.
The rest of the report is organized as follow:
In the first section, the HW(Hardware) and SW(Software) of both node and BS will be explained. In the second section, the bluetooth communication between BS and an Android application will be explained.In the third section we will talk about the developed Android application. Finally we will conclude our project in the last section.

2 Wireless Node

2.1 Hardware

In this project we used FRDM-KL25Z from freescale[1]. The FRDM-KL25Z is an ultra-low-cost development platform for Kinetis L Series KL1x (KL14/15) and KL2x (KL24/25) MCUs built on ARM Cortex-M0+ processor. Features include easy access to MCU I/O, battery-ready, low-power operation, a

standard-based form factor with expansion board options and a built-in debug interface for flash programming and run-control. The FRDM-KL25Z is supported by a range of Freescale and third-party development software. Here is some of the fetures of this platform:

- High performance ARM Cortex-M0+ Core
- 48MHz, 16KB RAM, 128KB FLASH
- 2xSPI, 2xI2C, 3xUART, 6xPWM, 6xADC, Touch Sensor, GPIO
- MKL25Z128VLK4 MCU 48 MHz, 128 KB flash, 16 KB SRAM, USB OTG (FS), 80LQFP
- P&E Multilink interface provides run-control debugging and compatibility with IDE tools

A brief user guide document for FRDM-KL25Z Evaluation board can be found in here[2]. For this platform we designed a PCB that we can later solder all the necessary modules ont it. Using a PCB instead of wire connection will increase the reliability of the connections. The whole PCB process, from designing the board to getting the first reliable response, took around 10 days. You can find BOM and PCB schematic in Appendix A. Figure 2 and figure 3 illustrate the top and bottom view of the designed PCB which were designed in AltiumDesigner[3].

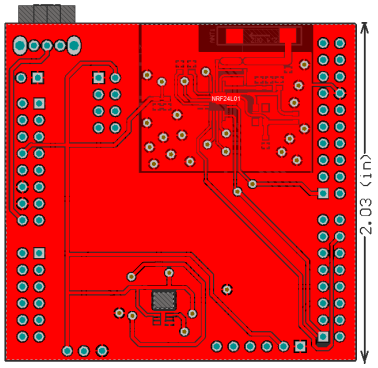


Figure 2: Top view of the PCB

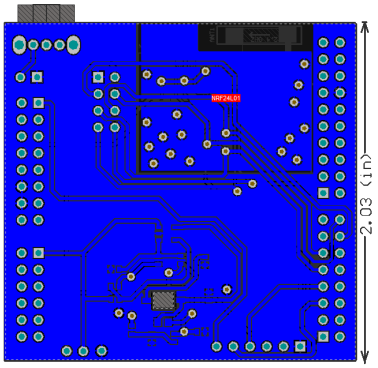


Figure 3: Bottom view of the PCB

The RF response of the designed board is still under review, how ever, after spending some time on the transceiver datasheet, we were able to send and receive data from a distance of more than 7 meters which is more than what we needed in this project. Figure 4 illustrates the final product. In this project we used 4 of these nodes. Each of them can be configured both as a node or BS. We used STAR topology[4] in our project so we need one BS and multiple number of nodes.

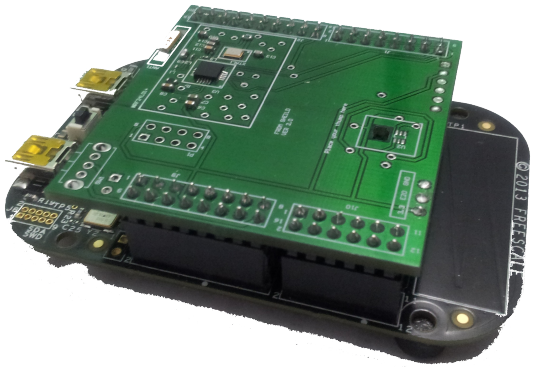


Figure 4: Designed board

In this board we used NRF24L01+[5] as the transceiver module. The Nordic nRF24L01+ is a highly integrated, ultra low power (ULP) 2Mbps RF transceiver IC for the 2.4GHz ISM (Industrial, Scientific and Medical) band. With peak RX/TX currents lower than 14mA, a sub A power down mode, advanced power management, and a 1.9 to 3.6V supply range, the nRF24L01+ provides ULP solution enabling months to years of battery life from coin cell or AA/AAA batteries. The Enhanced ShockBurst hardware protocol accelerator offloads time critical protocol functions from the application microcontroller enabling the implementation of advanced and robust wireless connectivity with low cost 3rd-party microcontrollers. The low power consumption of this module was the most interesting feature for us. Beside the NRF transceiver, we used HC-05 to make a bluetooth connection between the BS and the Android application. Here are some main features of the RF transceivers (Bluetooth and NRF module):

- NRF24L01+(Both on nodes and BS):
  - On-board 3.3V LDO Regulator (3.3 to 7V supply allowed)
  - Reverse Polarized SMA Connector for external 2.4 GHz Antenna
  - 100m Range at 250kbps
  - 250kbps to 2Mbit Data Rate
  - Multiceiver - 6 Data Pipes
  - Software selectable channel from 2400MHz to 2525MHz (125 Selectable channels)
- HC-05(Only on BS):
  - Chipset CSR BC417143
  - Bluetooth version V2.0+EDR
  - Output power Class II
  - Flash 8Mbit
  - Power Supply 3.3V

Beside the NRF module and the bluetooth module, all the boards can connect to an external sensor. Also, we have added a heart beat sensor on the board that can be soldered later on the board. The power consumption of the node was our initial concern, the final node is powered only by a single CR2032. The BS however, needs a lithium pack battery. We used a 3.6v 800mAH lipo battery.

2.2 Software

2.2.1 Protocol

TDMA(Time division multiple access) and FDM(Frequency-division multiplexing) scheduling are two main scheduling technique in WSN. TDMA is a channel access method for shared medium networks. It allows several users to share the same frequency channel by dividing the signal into different time slots. The users transmit in rapid succession, one after the other, each using its own time slot. This allows multiple stations to share the same transmission medium (e.g. radio frequency channel) while using only a part of its channel capacity[6]. This method (TDMA) need the nodes and BS to be synchronized. This task could be very challenging. Also it will need to transmit several control package in synchronization phase.
Frequency-division multiplexing (FDM) is a technique by which the total bandwidth available in a communication medium is divided into a series of non-overlapping frequency sub-bands, each of which is used to carry a separate signal. These sub-bands can be used independently with completely different information streams, or used dependently in the case of information sent in a parallel stream. This allows a single transmission medium such as the radio spectrum, a cable or optical fiber to be shared by multiple separate signals[7]. As mentioned earlier, we are using NRF24L01+ transceiver. In this module, we have software channel selection, ranging from 2400MHz to 2525MHz (125 Selectable channels). In FDM based WSNs, BS should have software channel selection. So we used this technique (FDM scheduling) for our project. Figure 5 shows how does BS works.

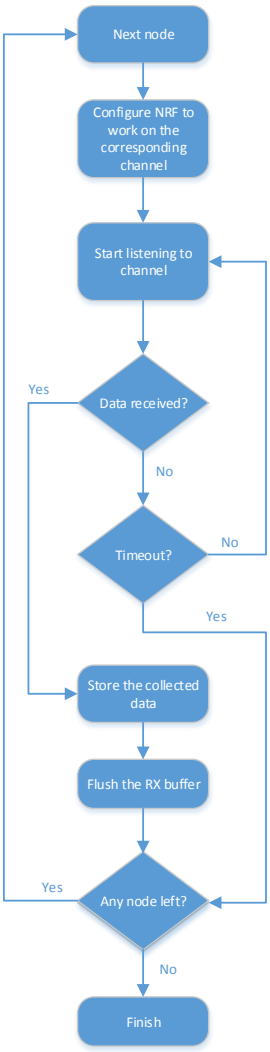


Figure 5: Base station channel search algorithm

In the node side, each node will wake up at the certain time and will transmit the sampled data to the base station. Each node has its own unique ID and data channel so no data conflict will happened.

2.2.2 RTX RTOS

As mentioned earlier, we are using an FDM based scheduling technique to send and receive data. Decreasing the power consumption was our first goal. To do so, node and BS should be kept in sleep mode as much as possible. Data transmission errors, data sampling error or any other undefined error should not affect the functionality of the network. The best way to achieve all the above goals is to use an RTOS. Each task should be implemented as a thread. All the device drivers must be compatible with RTOS. In our project we used Keil RTX RTOS. The Keil RTX is a royalty-free, deterministic Real-Time Operating System designed for ARM and Cortex-M devices. It allows us to create programs that simultaneously perform multiple functions and helps to create applications which are better structured and more easily maintained[8]. Figure 6 illustrates the threads and interrupts subroutines that we used in our design.

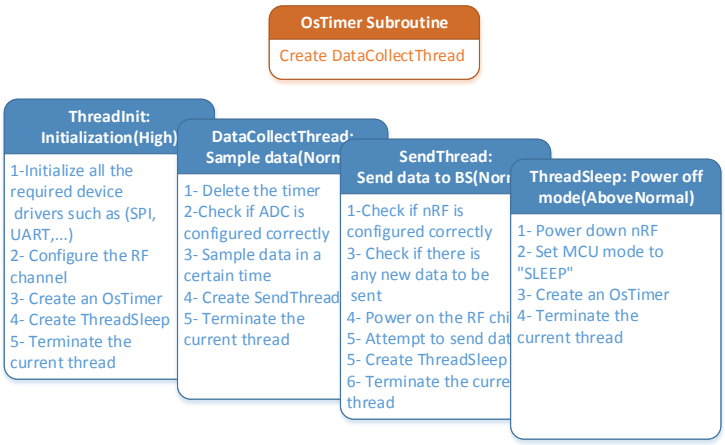


Figure 6: Threads and subroutines that are defined in RTX

We wrote most of the device drivers in C and some other in C++. Some of the drivers had to be modified in order to function correctly in RTX. Here are some of the drivers that we used in this project:

- UART0: Debug and report (BS and nodes)
- UART1: To communicate with Bluetooth module (Only BS)
- SPI0: To communicate with NRF24L01+ (BS and nodes)
- ADC0: To sample data. Configuration: 16bit data, 2MHZ sample rate, hardware averaging was selected. (BS and nodes)
- MCG: MCU PLL(Phase Lock Loop) that was configured to work at 48MHZ(BS and nodes)
- NRF24L01+: This driver was built on SPI0 driver. The device is configured to work as TX/RX and the selected data rate is 250Kbps(BS and nodes)

We programmed BS to generate a log file. At the beginning, BS should know what would be the maximum number of node in the network. In this demo, we set this parameter to 5. Here is an output result of this example:

```
=====================================================
ThreadInit:
=====================================================
Device Configuration.............................passed
nRF24L01+ Configuration ...................in Progress
nRF24L01+ Configuration.........................passed
Initial RF Channel..............................2402Mhz
Number of Nodes in Network.....................5
Node[1] RF Channel..............................2410Mhz
Node[2] RF Channel..............................2420Mhz
Node[3] RF Channel..............................2430Mhz
Node[4] RF Channel..............................2440Mhz
Node[5] RF Channel..............................2450Mhz
Timer CallBack!
Starting ThreadSleep...
Starting ReCThread ...
Starting BLThread...
Waiting to receive data from Node[1] on Channel[10]
Data from Node[1] did not received.
Waiting to receive data from Node[2] on Channel[20]
Data from Node[2] did not received.
Waiting to receive data from Node[3] on Channel[30]
```

```
Data from Node[3] did not received.
Waiting to receive data from Node[4] on Channel[40]
RF_CH[40]-Data Received from Node[4]: 50241
Waiting to receive data from Node[5] on Channel[50]
Data from Node[5] did not received.
ReCThread Finished...
```

In BS, we have all the same threads that we had in the node implementation except instead of DataCollectThread, we have BLThread. This thread is responsible to send the collected data to the Android application. As you can see in the log above, only one available node was found. BS will listen to each channel for at least 100ms. This interval can be easily modified. The only limitation is the startup time for NRF24L01+ to convert to a new channel. In this example, BS will wait to receive a data for 1500ms. Any number of node between 1 to 5 can attend or leave the network at any time.

3 Android Bluetooth Service

In order to collect the incoming data from the Remote sensor, I wrote a serial service for bluetooth module. This service was originally taken from the "BlueTerm" project. You can find the link to the Google Store in [9]. All the bluetooth configuration are well handled in this project. Since we need to process the incoming data and there is no need for many of the functions that are implemented in the "BlueTerm" project, I just copied the "DeviceListActivity.java" file and I made some major modification to "BluetoothSerialService.java". I will explain the modifications later in this report.
To start up the bluetooth connection, we need to check wether the device has a Bluetooth module or not. After checking the Bluetooth module availability, a dialog will appear and will ask the user if the application is allowed to turn the Bluetooth on or not. Since the application is highly dependent on the bluetooth connectivity, If the user decided not to turn the bluetooth on, the application will deny the access of the user to the tool by calling the android *finish()* function.

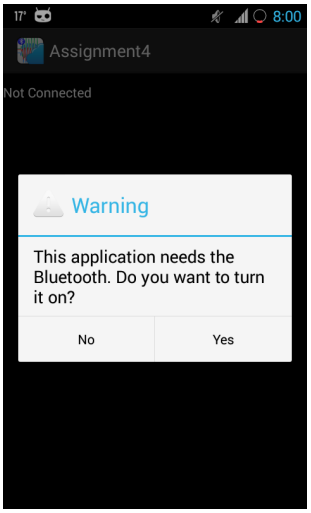


Figure 7: Application Modules

In the case that the user agreed to turn the bluetooth on, an Intent for *BluetoothAdapter.ACTION_REQUEST_ENABLE* will be requested. By start this activity for result, eventually, the bluetooth module will be turned on. Since starting discovery to find any nearby bluetooth module is very power consuming, the application will not search for any remote device unless the user ask for it.
Now the bluetooth is turned on, the user can choose to connect to a remote sensory device. To do this, the user should click on the Connect device option in the settings. By selecting the *Connect device* button, a menu will appear that shows the paired devices. The name and the MAC id of all the paired device will be listed in the menu.

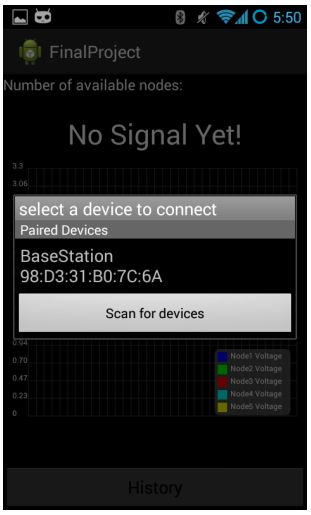


Figure 8: Application Modules

To search for the new nearby bluetooth devices, the user should click on the *Scan for devices* option. The application will call *mBtAdapter.startDiscovery()* to start the discovery. In this example, *mBtAdapter* is an instance for the BluetoothAdapter class. A very practical way of getting the name of the founded device is to use a *BroadcastReceiver*. I registered two actions for the receiver. One when a device is discovered, and the other one is when discovery has finished. On the receiver side, the application will check whether the founded device is already paired or not. to check whether its been already paired or not we can ask the bluetooth device object to return it's BondeState. If its true, it means that the founded device is already paired to the phone. Note that this does not mean that we are connected to the remote device. The phone can have a multiple paired devices but it can be connected to only one of them. If its false it means that its the first attempt of the founded device to connect to the phone. After finding the paired and unpaired devices, the application will list all the founded devices. It also will terminate the discovery for the new devices. As I mentioned, discovery for the new bluetooth modules is a very power consuming task so it is very beneficial to terminate the discovery when ever it is not needed. The application will list the paired and unpaired devices in to to sets: *"Paired Devices"* and *"Other Available Devices"* . If there are no new devices, the application will write "No " otherwise it will list the device.

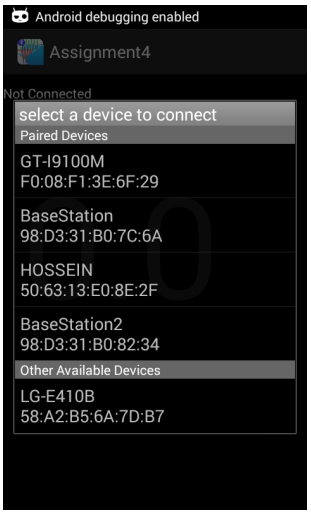


Figure 9: Application Modules

To connect to the remote device the user can simply click on any of names that appears either in the *"Paired Devices"* menu or *"Other Available Devices"* menu. By clicking on any of the names, the ap-

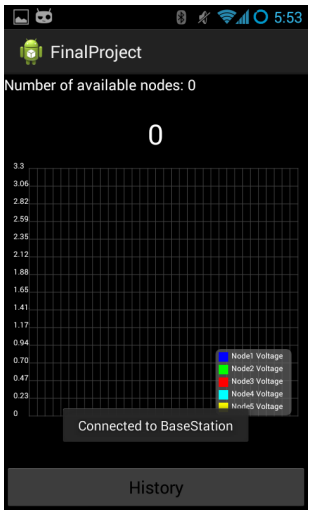


Figure 10: Application Modules

plication will start for the connection activity. The application will create a *ConnectThread*. This thread will create and open a *"RFcommSocket"*for bluetooth by passing the appropriate UUID(Universally unique identifier). Based on the application the UUID may differ. I used the following UUID:
00001101-0000-1000-8000-00805F9B34FB
After the Bluetooth socket is opened, the application will attempt to connect to the remote device by calling *mmSocket.connect()*. Where mmSocket is an instance for the *BluetoothSocket* class. If the connection was successful, a toast will appear on the screen confirming the successful connection.

Otherwise it will toast *Unable to connect the device*. Usually it takes two to five second for the bluetooth module to discover wether the connection was successful or not.
After a successful connection to the bluetooth module, the application will create a new thread called *ConnectedThread*. This thread will deal with the incoming data stream. The connected thread will keep listening to the InputStream while there is a valid connection. I used a data buffer to collect the input stream. I am using */n* character to terminate the input stream for collecting data. This is an agreement between the both sides(the remote sensor and the phone). When ever the input stream is fulled or a */n* character has been received, the *ConnectedThread* will convert the stream to string using *US-ASCII* encoding and then it will parse it to an integer value. After that, it will send the translated data to the MainActivity using *Message*. I am using the following key and pair for handling messages between the Bluetooth service and the MainActivity:

- MESSAGE_STATE_CHANGE
- MESSAGE_READ
- MESSAGE_WRITE
- MESSAGE_DEVICE_NAME
- MESSAGE_TOAST
- CONNECTION_LOST
- UNABLE_TO_CONNECT

To send any received data to the MainActivity, I need to use *MESSAGE_READ* key.

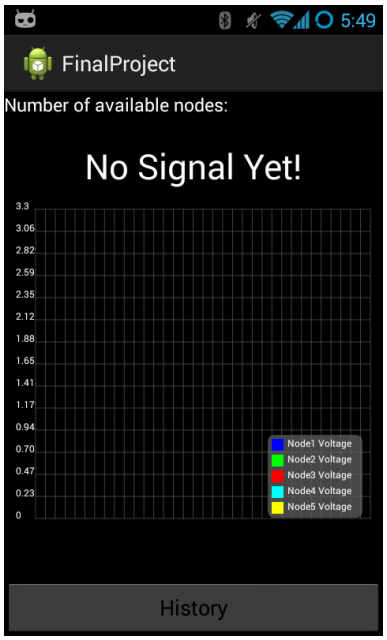


Figure 11: Application main page

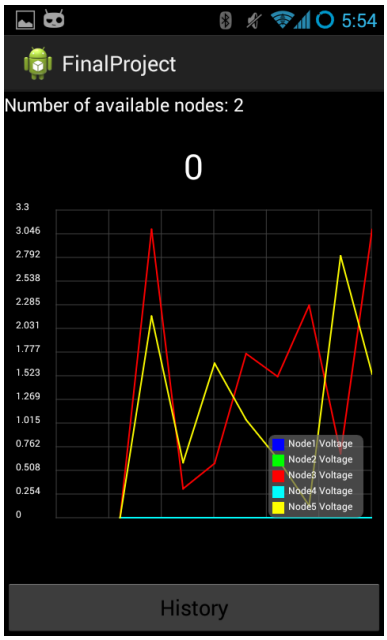


Figure 12: Live view graph

4 Android application

Based on the WSN protocol and the bluetooth serial service module, we developed an android application that could monitor the data transactions in the network. In this application we used DropBox DataStore as our data base. At the main page of the developed application, user can view a live graph of the data that are sent by each node. Figure 11 and figure 12 show the application main page and the live view graph.

Beside live view, the total number of available nodes is also provided to user. If the bluetooth connection to BS is available and the application is successfully linked to DropBox, the incoming from the BS is also available. The application will save the incoming data as records. Node_ID and Node_Data are two pairs of each record. Based on this information a table of records will be generated automatically. Each table is stored in a session. Creating a new session is automatically. We plan to generate a new session for each day. All the sessions are accessible to user in the history view. Figure 13 illustrates the history view of the application. When user selects the history view, a listview will be displayed containing all the available sessions. Each session is selectable and when it is selected, a graph view of the current session will be displayed. You can see this feature in figure 14.

4.1 DropBox DataStore:

In order to save the node ID and Data, and adding the ability to easily search for the previously saved data by date, ID number, etc, we tried to use a database structure. DropBox DataStore API lets developers to easily manage and store data. All classes and methods in the DataStore API are thread-safe, which give us the power to write and read from a table/datastore at the same time. Another advantage of using the DataStore API is the ability to store data locally without access to the internet. The next time the user has access to the internet the tables and datastore are automatically synced. There are some storage size limitations which are above our need for this application. *DbxAccountManager* class is used to initialize the Sync API and link the application to the user account at DropBox. In the *onCreate* method the static method *getInstance()* is called to get an instance of the *DbxAccountManager*. By calling the *startLink()* method the application links the application to the DropBox account.

DbxDatastore class lets developers to communicate with tables and records in that DataStore. *DbxDatastore* contains all data for the application.

DbxTable contains a collection of records. Records could be queried or inserted into this class. Each table has a specific string ID which is used to distinguish different tables. Calling the *getTable(String id)*

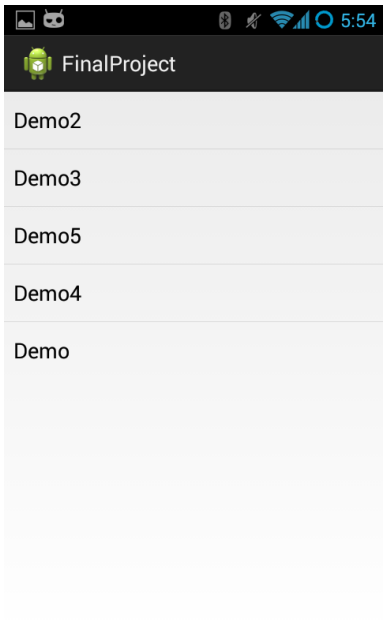


Figure 13: Search available sessions in History view

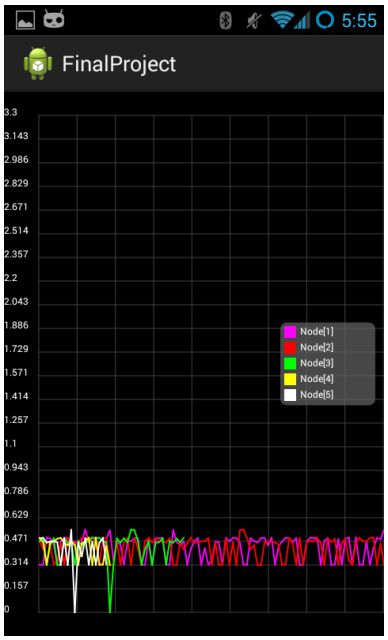


Figure 14: Graph view of the selected session

method returns a reference to the table with the given ID in the current datastore. If the ID is not in the store it will create the table with that ID and return a reference.

DbxRecord class let user to create a new record and insert that record into different tables. Each record is assigned a unique ID in the table, and each record can have several different fields.

There are currently no methods to delete a table. If a table is empty, i.e. no record is inserted yet or all records are deleted, the table is not created. So one easy way to delete a table is to delete all available record in that table. When the user tries to use the application for the first time, he/she needs to authorizes the application to sync to his/her dropbox account. To do this, application has to send an authentication request to dropbox. When user selects *Log to DropBox Account* button, the application will send an authentication request to DropBox. Figure 15 shows this procedure. If any error occurs during this procedure, the application will toast *Not Linked to DropBox* text. If the user is already linked to DropBox, application will toast *The app is already synced to DropBox*.

5 Conclusion:

In this project, we developed an android based application to monitor data transaction in a WSN. All the transactions are synced and stored using DropBox DataStore API. We used Keil RTX RTOS for developing an FDM protocol inside the FRDM platform. We also fabricated a PCB as a FRDM shield to increase the reliability of the electrical connections. As a future work, we are planing to participate resmiq innovation day and we are going to develop both software and hardware in such a way that would be much more useful for medical applications.

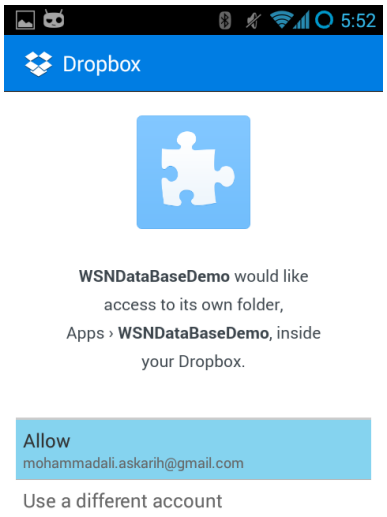


Figure 15: Authentication window in DropBox

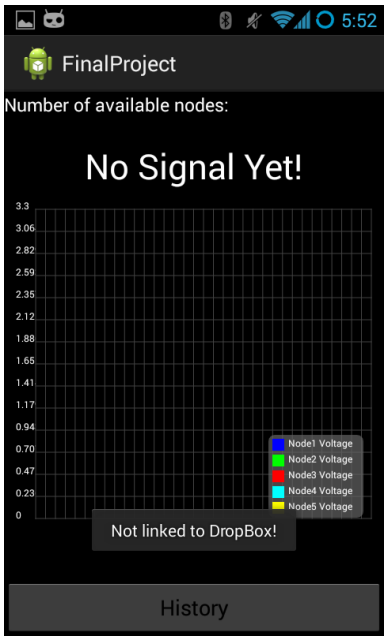


Figure 16: Unsuccessfull authentication

A AppendixA BOM and Schematics

Bill of Materials

VER.2.0

Source Data From: FRDM_Shield.PrjPcb
Project: FRDM_Shield.PrjPcb
Variant: None

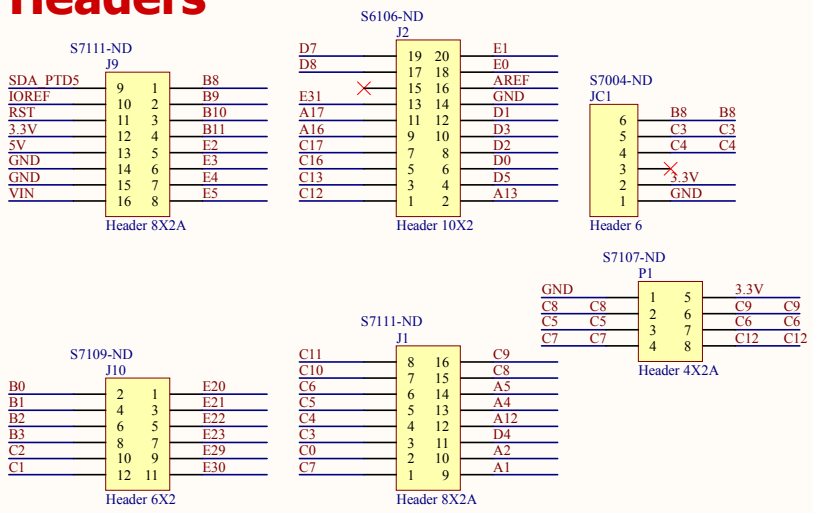
Creation Date: 4/2/2014 4:32:11 AM
Print Date: 22-Apr-14 5:30:52 PM

Production Quantity: 6
Currency CAD

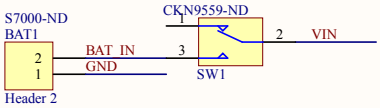
#Column Name Erro	Comment	LibRef	Designator	Description	Quantity	Supplier 1	Supplier Part Number 1	Supplier Unit Price 1	Supplier Order Qty 1	Supplier Subtotal 1	#Column Name Error:M	#Column Name Error:Manufacturer Part Number 1
	ANT-2.45-CHP-T	ANT-2.45-CHP-T	ANT1	ANT 2.4GHZ 802.11 BLUETOOTH SMD	1	Digi-Key	ANT-2.45-CHPCT-ND	2.87	6	$ 17.22		
	Header 2	Header 2	BAT1	Header, 2-Pin	1	Digi-Key	S7000-ND	0.49	6	$ 2.94		
	Coin Battery Hold	Coin Battery Hold	Battery Holder		1	Digi-Key	BK-883CT-ND	0.49	6	$ 2.94		
	4.7uF	Cap-Bipolar	C1, C2, C5	Bipolar Capacitor	3	Digi-Key	587-1441-1-ND	0.157	18	$ 2.83		
	2.2uF	Cap-Bipolar	C3, C4	Bipolar Capacitor	2	Digi-Key	587-1253-1-ND	0.09	12	$ 1.08		
	10nF	Cap-Bipolar	C6	Bipolar Capacitor	1	Digi-Key	399-1091-1-ND	0.02	10	$ 0.20		
	1nF	Cap-Bipolar	C7	Bipolar Capacitor	1	Digi-Key	587-1070-1-ND	0.34	6	$ 2.04		
	33nF	Cap-Bipolar	C8	Bipolar Capacitor	1	Digi-Key	1276-2042-1-ND	0.025	10	$ 0.25		
	1.5pF	Cap-Bipolar	C9	Bipolar Capacitor	1	Digi-Key	712-1327-1-ND	0.29	6	$ 1.74		
	1pF	Cap-Bipolar	C10	Bipolar Capacitor	1	Digi-Key	478-6732-1-ND	0.9	6	$ 5.40		
	2.2nF	Cap-Bipolar	C11	Bipolar Capacitor	1	Digi-Key	445-9139-1-ND	0.24	6	$ 1.44		
	4.7pF	Cap-Bipolar	C12	Bipolar Capacitor	1	Digi-Key	712-1172-1-ND	0.31	6	$ 1.86		
	22pF	Cap-Bipolar	C13, C14	Bipolar Capacitor	2	Digi-Key	490-6089-1-ND	0.924	12	$ 11.09		
	Coin Cell Battery	Coin Cell Battery	Cell		1	Digi-Key	P189-ND	0.28	6	$ 1.68		
	CD0603-S01575	CD0603-S01575	D1	DIODE SMALL SIG 75V 0.15A 0603	1	Digi-Key	CD0603-S01575CT-ND	0.4	6	$ 2.40		
	AM2520ZGC09	AM2520ZGC09	D2	LED 2.5X2 525NM GRN WTR CLR SMD	1	Digi-Key	754-1423-1-ND	0.79	2	$ 1.58		
	Header 8X2A	Header 8X2A	J1, J9	Header, 8-Pin, Dual row	2	Digi-Key	S7111-ND	1.064	12	$ 12.77		
	Header 10X2	Header 10X2	J2	Header, 10-Pin, Dual row	1	Digi-Key	S6106-ND	1.51	6	$ 9.06		
	Header 6X2	Header 6X2	J10	Header, 6-Pin, Dual row	1	Digi-Key	S7109-ND	1.02	6	$ 6.12		
	Header 6	Header 6	JC1	Header, 6-Pin	1	Digi-Key	S7004-ND	0.68	2	$ 1.36		
	L1	Inductor	L1	Inductor	1	Digi-Key	712-1492-1-ND	0.036	10	$ 0.36		
	L2	Inductor	L2	Inductor	1	Digi-Key	712-1502-1-ND	0.048	10	$ 0.48		
	L3	Inductor	L3	Inductor	1	Digi-Key	712-1490-1-ND	0.036	10	$ 0.36		
	Male Header 2*6	Header	Male Header		1	Digi-Key	952-2126-ND	0.41	6	$ 2.46		
	Male Header 2*8	Header	Male Header		1	Digi-Key	952-2128-ND	0.51	6	$ 3.06		
	Male Header 2*8	Header	Male Header		1	Digi-Key	952-2128-ND	0.51	6	$ 3.06		
	Male Header2*10	Header	Male		1	Digi-Key	952-2129-ND	0.64	6	$ 3.84		
	Header 4X2A	Header 4X2A	P1	Header, 4-Pin, Dual row	1	Digi-Key	S7107-ND	0.86	4	$ 3.44		
	10K	Resistor	R1	Resistor	1	Digi-Key	541-10.0KHDKR-ND	0.083	10	$ 0.81		
	3.3M	Resistor	R2	Resistor	1	Digi-Key	541-3.3MGDKR-ND	0.074	10	$ 0.74		
	100K	Resistor	R3, R5, R6	Resistor	3	Digi-Key	541-100KHDKR-ND, 541	0.081	18	$ 1.46		
	470	Resistor	R4	Resistor	1	Digi-Key	541-470HDKR-ND	0.081	10	$ 0.81		
	22K	Resistor	R7	Resistor	1	Digi-Key	311-22.0KHRCT-ND	0.015	10	$ 0.15		
	1M	Resistor	R8	Resistor	1	Digi-Key	541-1.00MHDKR-ND	0.081	10	$ 0.81		
	Switch SPDT_Su	Switch SPDT_Su	SW1	Subminiator SPDT Switch	1	Digi-Key	CKN9559-ND	0.35	6	$ 2.10		
	MCP6001RT-I/O	MCP6001RT-I/O	U1	IC OPAMP GP 1MHZ RRO SOT23-5	1	Digi-Key	MCP6001RT-I/OTCT-ND	0.29	6	$ 1.74		
	APDS-9008-020	APDS-9008-020	U2	IC PHOTOSENSOR AMBIENT 6CHIPLED	1	Digi-Key	516-2662-1-ND	1.23	2	$ 2.46		
	NRF24L01P-T	NRF24L01P-T	U3	IC RF TRANSCEIVER 2.4GHZ 20QFN	1	Digi-Key	1490-1019-ND	3.36	6	$ 20.16		
	403C11E16M000	403C11E16M000	XTAL1	CRYSTAL 16MHZ 20PF SMD	1	Digi-Key	CTX732CT-ND	0.79	6	$ 4.74		
					46					$ 159.88		

Approved **Notes**

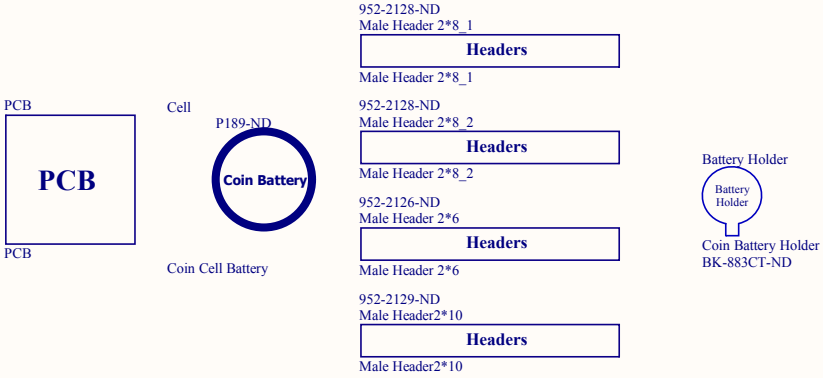
Headers



Power

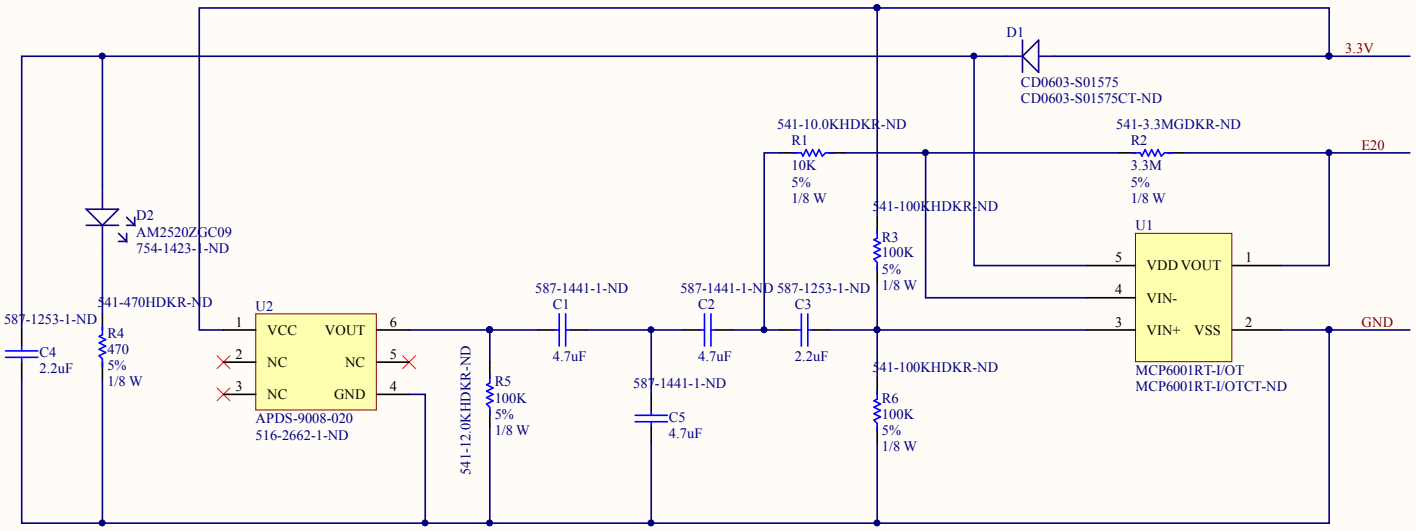


Mechanical

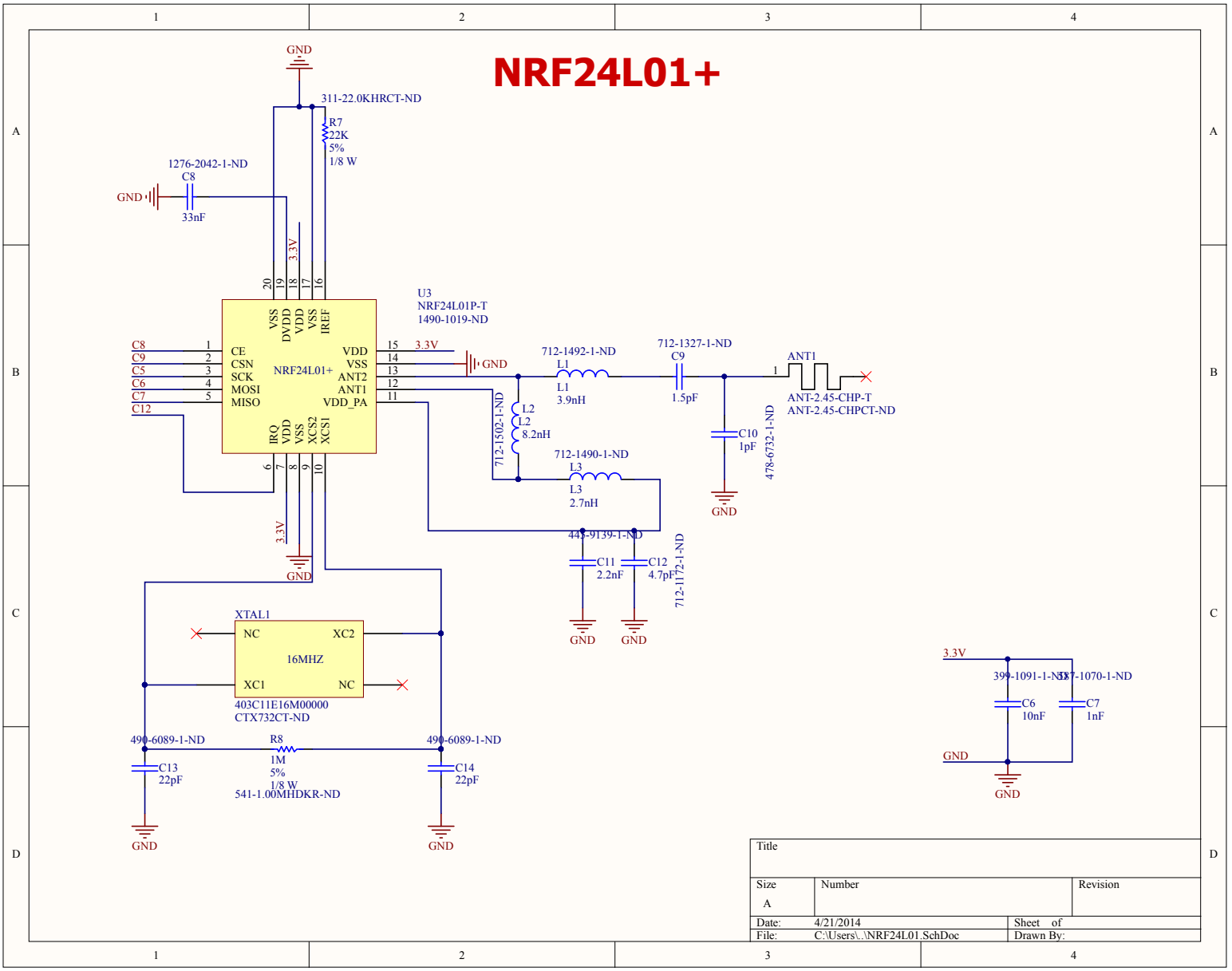


Title		
Size: A	Number	Revision
Date: 4/21/2014		Sheet of
File: C:\Users\..\FRDM_Shield.SchDoc		Drawn By:

HeartBeat Sensor



Title		
Size: A	Number	Revision
Date: 4/21/2014		Sheet of
File: C:\Users\..\HeartBeat.SchDoc		Drawn By:


NRF24L01+
GND
311-22.0KHRCT-ND
R7
22K
5%
1/8 W
1276-2042-1-ND
C8
33nF
3.3V
U3
NRF24L01P-T
1490-1019-ND
VSS
DVDD
VDD
VSS
IREF
CE
CSN
SCK
MOSI
MISO
NRF24L01+
VDD
VSS
ANT2
ANT1
VDD_PA
IRQ
VDD
VSS
XCS2
XCS1
712-1492-1-ND
L1
3.9nH
712-1327-1-ND
C9
1.5pF
ANT1
ANT-2.45-CHP-T
ANT-2.45-CHPCT-ND
712-1502-1-ND
L2
8.2nH
712-1490-1-ND
L3
2.7nH
C10
1pF
478-6732-1-ND
445-9139-1-ND
C11
2.2nF
C12
4.7pF
712-1172-1-ND
XTAL1
NC
XC2
16MHZ
XC1
NC
403C11E16M00000
CTX732CT-ND
490-6089-1-ND
R8
1M
5%
1/8 W
541-1.00MHDKR-ND
C13
22pF
C14
22pF
399-1091-1-ND
587-1070-1-ND
C6
10nF
C7
1nF
Title
Size
A
Number
Revision
Date: 4/21/2014
Sheet of
File: C:\Users\..\NRF24L01.SchDoc
Drawn By:

References

[1] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=FRDMKL25Z

[2] https://www.dropbox.com/s/kq5f95bdvgcklsf/Keil.pdf

[3] http://www.altium.com/

[4] http://en.wikipedia.org/wiki/Star_network

[5] https://www.nordicsemi.com/kor/node_176/2.4GHz-RF/nRF24L01P

[6] http://en.wikipedia.org/wiki/Time_division_multiple_access

[7] http://en.wikipedia.org/wiki/Frequency-division_multiplexing

[8] http://www.keil.com/rl-arm/kernel.asp

[9] https://play.google.com/store/apps/details?id=es.pymasde.blueterm