Enabling Zigbee Communications in Android Devices

Siquan Hu, Yu Fu
School of Computer and Communication Engineering
University of Science and Technology Beijing
Beijing 100083, China
husiquan@ustb.edu.cn

Chundong She, Hui Yao
Ruijie Networks Co., Ltd.
Beijing 100036，China
yaohui@ruijie.com.cn

***Abstract*—Android is one of the most popular smart phone and tablet platforms; Zigbee is the standard for home automation and will play an important role on future WPAN. Enabling the Zigbee communication on Android can exploit the rich network and nice display power of the Android phone in Zigbee applications. In this paper we present our work on integrating CC2530 into an Android smart phone. Based on JNI library, the software is developed to enable Android system can control the CC2530 module insides. The firmware in CC2530 is also discussed. The test is carried on with the configuration that an Android smart phone communicates with a standalone CC2530 node via Zigbee radio. The test verified the design feasibility.**

Keywords-Android devices, zigbee networks; JNI library; wireless networks

I. INTRODUCTION

Zigbee technology is a low cost, low power consumption, and short distance wireless communication technology, which is developed for wireless personal area network (WPAN) in recent years [1] [2] [3]. Zigbee is a synonym of IEEE 802.15.4 protocol, which is a hot topic research in short-distance wireless communication technology. Its main advantages are low-power, short-distance, low-complexity, self organization, low-speed, low-cost, and so on. It is widely used in building automatic control, industry autoimmunization, monitoring and control of hospital and home and other fields.

Android is one of the most popular smart phone and tablet platforms [4]. Android is a Linux based open source operating system for smart mobile devices. It is based on a modified Linux kernel optimized for mobile devices. Due to these optimizations, it is not possible to run any Linux application on the device which makes the work with Android OS somewhat more technically challenging compared to working with a full featured Linux distribution. However, because of Android‘s open source and high flexibility, software developers can access to hardware and rich software easily. Since a portable device such as an Android tablet or mobile phone has nice display and rich network connections such as WIFI, 3G, Blue, many researchers are working on exploiting the ability of the Android platform in sensing applications or wireless sensor networks field [5] [6] [7] [8] [9]. Android smart phones were used in body sensor network and two different smart phone configurations are proposed in [5]. Android smart phones are used to track human activity in smart home in [6] and to weigh human body in [8]. A mobile TeleCare system are designed in [7] with a smart wrist-worn device sensing health parameters and an Android smart phone working on data processing and communication. Based on Near Field Communication (NFC) technology and Bluetooth, an Android smart phone collects sensor data from body sensor networks, displays sensor data on the screen and streams to the central server simultaneously in [9]. [10] reports its the implementation of an Android IPv6 smart phone that interworks with Web-enabled wireless sensors in the Internet of Things.

Since Zigbee is the standard for home automation and will play an important role on future WPAN, enabling the Zigbee communications into Android smart phones is more attractive than WiFi, Bluetooth or NFC. Some work has been done on discussing the framework. For instance, a plug-in SD module is used to gain Zigbee ability in [11]. A RF4CE module is mounted to the Android phone to gain RF4CE ability in [12]. Different from above work, in this paper we present our work on integrating CC2530 into an Android smart phone and software development to enabling its Zigbee communication with a standalone CC2530 node. The hardware connection between Android phone CPU and CC2530 are showed in part II; the work on software development in JNI library to control CC2530 from Android is presented in part III; Part IV is the evaluation result and Part V is the conclusion.

II. HARDWARE DESIGN

A typical Android phone has many kinds of radios, for example, 2G, 3G, WIFI, Bluetooth, FM, etc. To enable Zigbee on such platform, a Zigbee chip is needed. There are two options when integrating the Zigbee chip. One is using radio chip such as CC2420. In this case, the phone CPU will control the radio chip directly via high speed SPI connection. The radio state machine should be implemented on the ARM CPU such as registers read/write, TX/RX switch, buffer processing, etc. Furthermore, Zigbee protocol should also be implemented on the ARM CPU. Technically this could be done, but it could not reuse the effort that has already finished on the microcontroller of the Zigbee node. This makes the first option less preferable than the second one. , which uses a SOC solution such as CC2430 or CC2530. The SOC solution can leave the radio control logic and Zigbee protocol processing unchanged exactly as a standalone Zigbee node.

In our design, we use CC2530 working with the phone CPU. The smart phone uses Android 2.3 and its CPU is a MTK6573 chip which has an ARM11 core, which has four USART ports. We use UART1 to connect to the USART0 of CC2530. CC2530 is a SOC with an enhanced 8051 core inside. The 8051 part run program to send and receive Zigbee packets, to schedule the radio sleep and wakeup cycles, to start a Zigbee network or join a network, to route the packets to the desired destination, etc. In our design, the CC2530 also send the received packets to USART0 so that the Android system can handle them. When the Android wants to send a packet into the Zigbee network, it will send it via UART1, and then CC2530 will receive it and relay it into the air via Zigbee radio. In this way, CC2530 acts a serial device from the point of view of Android system, and the complexity is reduced significantly because the ARM CPU skips all the radio details.

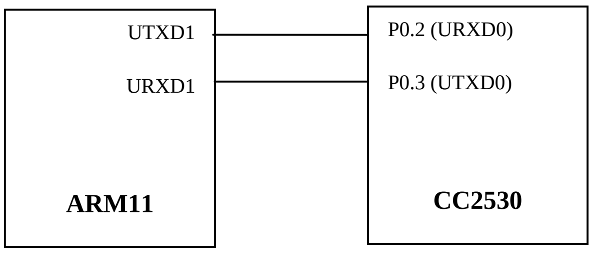


Figure 1. The connections bwtween CC2530 with the ARM CPU of an Android phone

III. CONTROL ZIGBEE DEVICE IN ANDROID

To control a device added into Android system, three tasks should be carried out. One is to provide the Linux device driver, which is same as ordinary situation in Linux because Android uses Linux kernel as a basis. The second task is to create the JNI (Java Native Interface) library, which is the share library to connect the Android application with the underlined Linux device driver. JNI library is created in Android NDK (Native Development Kit). The third task is to create an Android application which will control device function by accessing the Linux device driver via the JNI shared library. The Android NDK is a toolset that allows you to implement parts of your app using native-code languages such as C and C++. For certain types of apps, this can be helpful so you can reuse existing code libraries written in these languages.

In the hardware configuration in part II, an Android application must have the ability to operate the serial port /dev/ttyS1 so that it can send or receive Zigbee packets via CC2530. Since Android doesn't provide API for the serial port, a JNI library is needed. An Android application can manipulate the serial port with the help of the JNI library.

To create the serial port interface library, we create a C file JavaUART.c and compile it with NDK in Linux. The code skeleton of JavaUART.c is illustrated in Fig.2, all the serial port operating function such as open the serial port, close the serial port, read or write strings, reading or write binary packets via the serial port are provided. It is very important to follow the naming rules of JNI here for the sake of accessing the c function in the library from Java. For instance, "Java_com_JavaUART_JavaUART_open" means that if an Android application wants to call this function, it should have a class named as "JavaUART" in the package "com.JavaUART". In the java class JavaUART, this function is available as a Java function open().

Figure 2. Main function implemented in the serial JNI library

```
jint Java_com_JavaUART_JavaUART_open( JNIEnv*
env, jobject jobj, jstring strPort, jstring strBaut, jint
vtime, jint vmin);
jint Java_com_JavaUART_JavaUART_close( JNIEnv*
env, jobject jobj, jint nFd);
jint Java_com_JavaUART_JavaUART_read(JNIEnv*
env, jobject jobj, jint nFd, jint nMaxBytes);
jint Java_com_JavaUART_JavaUART_write(JNIEnv*
env, jobject jobj, jint nFd, jstring str1, jint nSendBytes);
jint Java_com_JavaUART_JavaUART_readb( JNIEnv*
env, jobject jobj, jint nFd, jcharArray rcvdata, jint
nMaxBytes);
jint Java_com_JavaUART_JavaUART_writeb(JNIEnv*
env, jobject jobj, jint nFd, jcharArray senddata,  jint
nSendBytes);
```

To compile the serial source file into JNI library, a Make file "Android.mk" should be prepared to specify the source files, the shared library compilation option and the library name. It is illustrated in Fig.3. The last line is setting the flag of shared library compilation. LOCAL_MODULE specify the name of shared library to be created. In NDK, the final library name will use "lib" as a prefix and ".so" as a extension. Then the library created from the Make file in Fig.3 will be "libJavaUART.so".

Figure 3. Android.mk for compiling the serial JNI library

```
LOCAL_PATH := $(CALL my-dir)

include $(CLEAR_VARS)
LOCAL_MODULE := JavaUART
LOCAL_SRC_FILES := JavaUART.c

include $(BUILD_SHARED_LIBRARY)
```

Once the serial port interface library is created, it can be used in the Android application development. It means to access the function in the library from Android source files, which are written in java. As illustrated in Fig.4, a "JavaUART" class in package "com.JavaUART" is created in Android application project to access the functions in the serial port JNI library. The library should be loaded before its functions can be called in Java. In the mean time, the interface functions implemented in the library should be declared in Java beforehand. The "native" keyword is used to indicate the function is from the JNI library, enabling Java language to access the function written in C or C++.

In Fig.4, all the functions in JNI library are declared and can be accessed from *JavaUART* class. To provide the serial port operation function for the components out of the *JavaUART* class, the serial interfaces are reassembled as java functions such as “openPort()”, “recv()” ,”send()”. They will call the functions implemented in the JNI library actually.

```
package com.JavaUART;
…..
public class JavaUART extends Thread {
  static {System.loadLibrary(“JavaUART”);}
  private native int open(…);
  private native int close(…);
  private native String read(…);
  private native int write(…);
  private native int readb(…);
  private native int writeb(…);
  …
  public JavaUART(Eventhandler eHandler, int
lenBuf){…}
  public int openPort(…) {…}
  public int closePort(){…}
  publc void run(){…}
  public String recv(){…}
  public int send(string  strBuf) {…}
  public int recvb(char buf[]){…}
  public int sendb(char buf[], int len) {…}
  }
```

Figure 4. The java class accessing the serial JNI library

To avoid missing coming packets, the Android application needs to keep on listening to the UART port. This is achieved by implementing JavaUART class as a Thread class. Since the serial port in Android is used both on sending and receiving packets, to avoid conflict, a critical section is needed. We implemented it as an event handler as Fig.5.

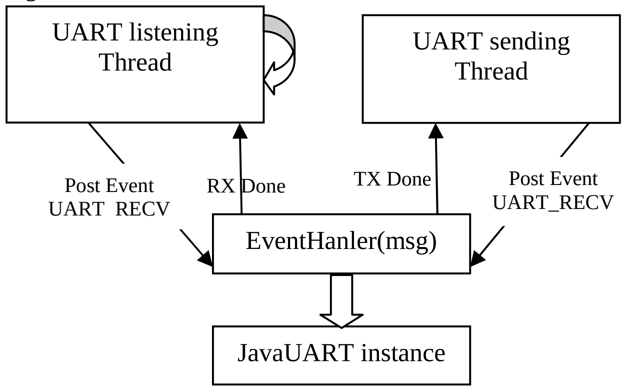


Figure 5. The eventhandler processing the sending and receiving

The event handler will accept two kings of event, UART_RECV and UART_SEND. Once an event triggered, the handler will call the *recv()* or *send()* function of the *JavaUART* class depending on the kind of event. Based on the analysis, the event handler is key point for both sending thread and receiving thread. However, the receiving thread is implemented in the *JavaUART* class rather the top level activity class of the Android application. So a proper design would be putting the event handler component in the main activity class, and transferring it as a argument to the creator function of class *JavaUART*.

IV. EVALUATION

To evaluate the Zigbee communication of the Android device, we implemented a *SerialBase* application on CC2530 to serve as the bridge of the CC2530 UART and radio. The firmware is based on TI Z-Stack. The main function codes are illustrated in Fig.6. When the Android sends a packet to CC2530 via UART, the firmware on CC2530 will relay it to radio. And when CC2530 receives a radio packet, the firmware will use *HALUARTwrite* to forward it to the UART and will be received by the Android receiving thread.

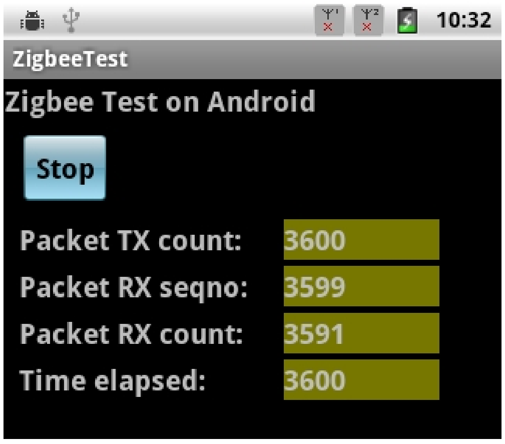
```
uint16 SerialBase_ProcessEvent( uint8 task_id, uint16
events ){
    ....
    // Send a message out
    if ( events & UART_RX_CB_EVT)  {
      SerialBase_Radio_SendData( rbuf, rxlen+1);
      return (events ^ UART_RX_CB_EVT);
    }
  ...
  }
  ...
  void SerialBase_MessageMSGCB
( afIncomingMSGPacket_t *pkt ){
    uint8 *pRecv;  //the pointer to the recv data
    switch ( pkt->clusterId ) {
      case SERIALBASE_FLASH_CLUSTERID:
          pRecv=&pkt->cmd.Data[1];
                //cmd.Data[0] is the size of recv data
          HalUARTWrite(0,pRecv,pkt->cmd.Data[0]);
      break;  } }
```

Figure 6. Main code of the firmware on CC2530

To verify our design, the custom Android phone was deployed in the lab to communicate with a standalone CC2530 node. The Android phone sent a packet every second; the packet has a plus 1 sequence number field. The standalone CC2530 node run the similar code except it sent packets by itself. A laptop is attached to the standalone CC2530’s serial port to check the receiving.

An Android test application *ZigbeeTest* is developed in Eclipse with Android add-ons and deployed on the test Android phone. The Android GUI is presented in Fig.7.

ZigbeeTest implements a *ZigbeeTestActivity* class which extends Android *Activity* class. Its GUI layout has a Start/Stop button to control the test. Once started, a event handler will be created and a *JavaUART* instance is created with the event handler as a argument. Then the serial port "/dev/ttyS1"is opened with a baud rate 115200 and the receiving thread is started. After that, a sending thread is created to send Zigbee packets once per second. The test runs for an hour and the results are collected, which shows that 99.75% sent packets are successfully received by the standalone CC2530, and 99.8% packets are successfully received by Android *ZigbeeTest* application.



Zigbee Test on Android

Packet TX count: 3600
Packet RX seqno: 3599
Packet RX count: 3591
Time elapsed: 3600

Figure 7. The execution screenshot of ZigbeeTest application

V. CONCLUSION

Enabling the Zigbee communication on an Android device can exploit the power of the Android phone in home automation or WPAN. In this paper we present our work on integrating CC2530 into an Android smart phone. Based on JNI library, the software is developed to enable Android system can control the CC2530 module insides. The firmware in CC2530 is also discussed. The test is carried on with the configuration that an Android smart phone communicates with a standalone CC2530 node via Zigbee radio. The test verified the design feasibility.

ACKNOWLEDGMENT

This project is sponsored by National Key Technology R&D Program Grant #2012BAH25B02, and the Guangdong Province University-Industry Cooperation Grant #2011A090200008, and the Scientific Research Foundation, Returned Overseas Chinese Scholars, State Education Ministry.

REFERENCES

[1] J. Li,X. Zhu, N. Tang and J. Sui, "Study on ZigBee network architecture and routing algorithm",2nd International Conference on Signal Processing Systems (ICSPS), vol.2, pp.389-393, 2010.

[2] S.S.Riaz Ahamed, "The role of zigbee technology in future data communication system", Journal of Theoretical and Applied Information Technology,Vol.5, No.2, pp.129-135. 2009.

[3] G.Hu,"Key technologies analysis of ZigBee network layer",2nd International Conference on Computer Engineering and Technology (ICCET), Vol.7, pp. 560-563, 2010.

[4] Android, http://www.android.com

[5] M. Wagner, B.Kuch, C. Cabrera, P.Enoksson and A. Sieber,"Android based Body Area Network for the evaluation of medical parameters," 2012 Proceedings of the Tenth Workshop on Intelligent Solutions in Embedded Systems (WISES), pp.33-38, 2012.

[6] M. Fahim, I. Fatima, S. Lee and Y.K. Lee, "Daily life activity tracking application for smart homes using android smartphone", 14th International Conference on Advanced Communication Technology (ICACT), pp. 241-245, 2012.

[7] O. Postolache, P.S. Girao, M. Ribeiro, M. Guerra, J. Pincho, et.al., "Enabling telecare assessment with pervasive sensing and Android OS smartphone", 2011 IEEE International Workshop on MeMeA, pp. 288-293, 2011.

[8] C. Park, B. Park, K. Park, H. Uhm and H. Choi,"LR-WPAN based weighing scales and smartphones",2011 IEEE International Conference on Consumer Electronics (ICCE), pp.461-462, 2011.

[9] W.J. Yi, W. Jia and J.Saniie, "Mobile sensor data collector using Android smartphone", IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 956-959, 2012.

[10] P. Schleiss, N. Torring, S.A. Mikkelsen and R. H. Jacobsen, "Interconnecting IPv6 wireless sensors with an Android smartphone in the Future Internet" ,2nd Baltic Congress on Future Internet Communications (BCFIC), pp. 14-18, 2012.

[11] N. He ,Z.H. Li,C.X. Jiang,C. Yang, S.Fan, et.al.,"A Sensing Platform to Support Smartphones Accessing into Wireless Sensor Networks", IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS), pp.173-175, 2011.

[12] B. Koo,T. Ahn, J. In, Y. Park and T. Shon,"R-URC: RF4CE-Based Universal Remote Control Framework Using Smartphone", 2010 International Conference on Computational Science and Its Applications (ICCSA), pp.311-314, 2010.