Degree Course Systems Engineering
Option Infotronics

Bachelor Thesis 2014

Nathan Quinteiro

Wirelessly accessing instruments with standard GPIB/WLAN interfaces

Professor: Pierre-André Mudry
Expert: Lixin Ran

Hangzhou, China - September 4th, 2014
Title / Titel

Wirelessly accessing instruments with standard GPIB/WLAN interfaces

Description / Beschreibung

The GPIB (General Purpose Interface Bus) protocol is a short-range digital communication bus specification created in 1965. It was developed for use with automated test equipment and is still in use for that purpose nowadays. Usually this standard requires the use of a computer and large wires to connect on the instrument and exchange data or commands. The aim of this project is to develop a new way to connect on GPIB instruments, by providing a wireless interface between Smartphones and GPIB bus.

This work involves the development of a wireless communication interface between a smartphone and a GPIB bus. A hardware circuit will be connected on the GPIB bus and will provide the physical interface. A smartphone application must be develop to allow communication with the hardware device. The system provides the user the possibility to read or write on the GPIB bus with a smartphone.

Objectives:
- Design, manufacture and debug a small hardware to provide the physical interface between a Smartphone and a GPIB bus.
- Develop a software for the MCU able to provide the software interface between a Smartphone and a GPIB bus. Develop a smartphone application able to connect on the hardware and write/read on the GPIB bus using the interface provided by the circuit.

1 Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme. Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.
Objectives

This project involves the design of a communication interface between a smartphone and a GPIB bus. The aim is to develop a system easy to use allowing the user to simply write and read on a GPIB bus with his smartphone.

Methods | Experiences | Results

A Wi-Fi module (RAK421) is used for the communication between an Android smartphone and a MCU (ATmega164PA). The MCU can access to a GPIB bus via transceivers. The whole system allow the user to write or read on a GPIB bus with his smartphone.

A printed circuit board with Wi-Fi module, microcontroller, GPIB drivers and GPIB connector is designed and manufactured. This board is plugged on the GPIB bus and can read and write on the bus.

An Android application is developed and installed on a smartphone. This APP can connect to the Wi-Fi module allowing data transmission with the MCU. The application is then able to write or read data/command on the GPIB bus.

The application allows the user to create, save and use libraries containing different GPIB commands specific for each instruments.
## CONTENTS

1 Introduction 10

2 Objectives 11

3 Research and specifications 12
   3.1 GPIB standard ................................................................. 12
      3.1.1 GPIB in brief .......................................................... 12
      3.1.2 Connector ................................................................. 13
      3.1.3 Pins ............................................................................. 13
      3.1.4 Specifications ............................................................... 14
      3.1.5 GPIB handshake sequence ............................................. 14
   3.2 Wi-Fi chipset .................................................................. 16
      3.2.1 Technologies .................................................................. 16
      3.2.2 RS-WC-201 .................................................................. 16
      3.2.3 Problem with RS-WC-201 .............................................. 16
      3.2.4 RAK411 ....................................................................... 17
   3.3 MCU ................................................................................. 17
      3.3.1 Choice of MCU ............................................................. 17
   3.4 GPIB drivers ...................................................................... 18
   3.5 Power supply ..................................................................... 19
      3.5.1 Power source ................................................................. 19
      3.5.2 Supply voltages ............................................................. 19
      3.5.3 DC Regulator ................................................................. 20

4 Circuit development and Schematics 21
   4.1 Wi-Fi module - RS-WC-201 ................................................ 22
      4.1.1 Mode ........................................................................... 22
      4.1.2 Firmware upgrade ......................................................... 22
      4.1.3 Display LEDs ............................................................... 22
   4.2 Wi-Fi module - RAK411 ...................................................... 23
      4.2.1 Integration on the existing PCB ..................................... 23
      4.2.2 Firmware upgrade ......................................................... 24
      4.2.3 Display LED ................................................................. 24
   4.3 MCU ................................................................................. 25
      4.3.1 Programming ................................................................. 25
      4.3.2 Clock ........................................................................... 26
      4.3.3 Display LEDs ............................................................... 26
      4.3.4 Button ........................................................................ 26
   4.4 GPIB bus .......................................................................... 27
5 PCB
  5.1 Dimensions ....................................................... 28
  5.2 Components placement ............................................ 28
    5.2.1 Connectors .................................................. 28
    5.2.2 Jumpers ..................................................... 28
  5.3 Wi-Fi module ..................................................... 28
  5.4 Copper ground ................................................... 28
  5.5 Heatsink ......................................................... 28

6 Interface specifications ............................................. 29
  6.1 Library .......................................................... 29
  6.2 Wireless transmission protocol .................................. 30
    6.2.1 Write on GPIB bus ........................................... 30
    6.2.2 Read on GPIB bus ............................................ 30
    6.2.3 Set LEDs state ............................................... 31
    6.2.4 Paquets codes and status .................................... 31

7 MCU Software ......................................................... 32
  7.1 Startup .......................................................... 32
    7.1.1 Hardware initialization ...................................... 32
    7.1.2 XF initialization ............................................. 34
  7.2 XF ............................................................... 34
    7.2.1 Program tasks ................................................. 35
  7.3 RAK411 commands ................................................ 35
    7.3.1 Commands ..................................................... 36
    7.3.2 Command and response structures ......................... 37
    7.3.3 Setting structures ........................................... 37
    7.3.4 Sending header ............................................... 37
    7.3.5 Sending parameters ......................................... 38
    7.3.6 Send command states machine ................................ 38
    7.3.7 Receive command states machine ................................ 39
  7.4 Wi-Fi module states machine .................................... 40
    7.4.1 Init module states machine .................................. 41
    7.4.2 Working state ................................................ 42
  7.5 Communication with GPIB bus .................................... 43
    7.5.1 GPIB transmission ........................................... 43
    7.5.2 GPIB reception ............................................... 44
  7.6 GPIB states machine ............................................. 45
  7.7 I/O Management .................................................. 46
8 Android APP

8.1 Wireless communication

8.1.1 Connection

8.1.2 Data transmission

8.1.3 Data reception

8.2 Instruments library

8.2.1 Library structure

8.2.2 Library Manager

8.2.3 Load library

8.2.4 Create new library

8.2.5 Retrieving library commands

8.3 Main activity

8.3.1 Retrieving library

8.3.2 Application usage

9 Test

9.1 Hardware

9.1.1 MCU

9.1.2 RS-WC-201

9.1.3 RAK411

9.2 SPI transmission

9.3 Configuration of RAK411

9.4 Smartphone and Wi-Fi module communication

9.4.1 Connection on AP

9.4.2 Connection on the socket

9.4.3 Data transmission

9.5 Communication on GPIB bus

9.5.1 Handshake source test

9.5.2 Handshake acceptor test

10 Conclusion

10.1 Hardware

10.2 Software

10.3 Android application

10.4 Improvements

10.4.1 PCB

10.4.2 MCU Software

10.4.3 Android APP

References
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Explaining diagram of the project</td>
</tr>
<tr>
<td>2</td>
<td>GPIB female connector</td>
</tr>
<tr>
<td>3</td>
<td>GPIB - Handshake timing sequence</td>
</tr>
<tr>
<td>4</td>
<td>DS75160A functional truth table</td>
</tr>
<tr>
<td>5</td>
<td>DS75161 functional truth table</td>
</tr>
<tr>
<td>6</td>
<td>Hardware circuit diagram</td>
</tr>
<tr>
<td>7</td>
<td>Mode selection schematics</td>
</tr>
<tr>
<td>8</td>
<td>Extern module with MAX3232 and DB9 for firmware upgrade</td>
</tr>
<tr>
<td>9</td>
<td>RAK411</td>
</tr>
<tr>
<td>10</td>
<td>RAK411 - Pins definition</td>
</tr>
<tr>
<td>11</td>
<td>RAK411 - Evaluation board</td>
</tr>
<tr>
<td>12</td>
<td>MCU schematics with header for programming</td>
</tr>
<tr>
<td>13</td>
<td>Maximum frequency vs Vcc</td>
</tr>
<tr>
<td>14</td>
<td>GPIB schematics</td>
</tr>
<tr>
<td>15</td>
<td>Interface provided</td>
</tr>
<tr>
<td>16</td>
<td>Instrument configuration</td>
</tr>
<tr>
<td>17</td>
<td>Write command packet</td>
</tr>
<tr>
<td>18</td>
<td>Write response</td>
</tr>
<tr>
<td>19</td>
<td>Read command</td>
</tr>
<tr>
<td>20</td>
<td>Write response</td>
</tr>
<tr>
<td>21</td>
<td>Set led command</td>
</tr>
<tr>
<td>22</td>
<td>Codes and status values</td>
</tr>
<tr>
<td>23</td>
<td>Set led command</td>
</tr>
<tr>
<td>24</td>
<td>Class diagram : XF</td>
</tr>
<tr>
<td>25</td>
<td>Main states machine</td>
</tr>
<tr>
<td>26</td>
<td>Sending command to RAK</td>
</tr>
<tr>
<td>27</td>
<td>Receiving response values from RAK</td>
</tr>
<tr>
<td>28</td>
<td>Structure commands</td>
</tr>
<tr>
<td>29</td>
<td>Structure commands</td>
</tr>
<tr>
<td>30</td>
<td>Prototype of the different methods to set structure commands parameters</td>
</tr>
<tr>
<td>31</td>
<td>Structure command on SRAM</td>
</tr>
<tr>
<td>32</td>
<td>Send command states machine</td>
</tr>
<tr>
<td>33</td>
<td>Receive command states machine</td>
</tr>
<tr>
<td>34</td>
<td>RAK states machine</td>
</tr>
<tr>
<td>35</td>
<td>RAK initialization states machine</td>
</tr>
<tr>
<td>36</td>
<td>GPIB send states machine</td>
</tr>
<tr>
<td>37</td>
<td>GPIB receive states machine</td>
</tr>
<tr>
<td>38</td>
<td>GPIB states machine</td>
</tr>
<tr>
<td>39</td>
<td>Android mascot Bugdroid</td>
</tr>
<tr>
<td>40</td>
<td>Connection to Wi-Fi module’s AP</td>
</tr>
<tr>
<td>Page</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>41</td>
<td>Connection sequence diagram</td>
</tr>
<tr>
<td>42</td>
<td>Data transmission sequence diagram</td>
</tr>
<tr>
<td>43</td>
<td>Data reception sequence diagram</td>
</tr>
<tr>
<td>44</td>
<td>Library line structure</td>
</tr>
<tr>
<td>45</td>
<td>GPIB Command class</td>
</tr>
<tr>
<td>46</td>
<td>Library Manager</td>
</tr>
<tr>
<td>47</td>
<td>Create library</td>
</tr>
<tr>
<td>48</td>
<td>Load library</td>
</tr>
<tr>
<td>49</td>
<td>Choose library Activity</td>
</tr>
<tr>
<td>50</td>
<td>Library creator</td>
</tr>
<tr>
<td>51</td>
<td>Main activity options</td>
</tr>
<tr>
<td>52</td>
<td>Main activity</td>
</tr>
<tr>
<td>53</td>
<td>Circuit with USB/SPI connection to the PC</td>
</tr>
<tr>
<td>54</td>
<td>Circuit with USB/SPI connection to the PC</td>
</tr>
<tr>
<td>55</td>
<td>Evaluation board connection</td>
</tr>
<tr>
<td>56</td>
<td>Hardware modification to connect to EVB</td>
</tr>
<tr>
<td>57</td>
<td>Init response - Start information</td>
</tr>
<tr>
<td>58</td>
<td>AP visible by smartphone</td>
</tr>
<tr>
<td>59</td>
<td>Connection of smartphone on AP</td>
</tr>
<tr>
<td>60</td>
<td>Connection to listening port</td>
</tr>
<tr>
<td>61</td>
<td>Android test application</td>
</tr>
<tr>
<td>62</td>
<td>Data start</td>
</tr>
<tr>
<td>63</td>
<td>Data end - LED command</td>
</tr>
<tr>
<td>64</td>
<td>Data start</td>
</tr>
<tr>
<td>65</td>
<td>Data end - LED command</td>
</tr>
<tr>
<td>66</td>
<td>LEDs state displayed in the APP</td>
</tr>
<tr>
<td>67</td>
<td>Handshake sequence</td>
</tr>
<tr>
<td>68</td>
<td>Source test paquet</td>
</tr>
<tr>
<td>69</td>
<td>Writing successful</td>
</tr>
<tr>
<td>70</td>
<td>Timeout error</td>
</tr>
<tr>
<td>71</td>
<td>Acceptor test paquet</td>
</tr>
<tr>
<td>72</td>
<td>Reading successful</td>
</tr>
<tr>
<td>73</td>
<td>Timeout error</td>
</tr>
</tbody>
</table>
Acronyms

These different acronyms are used in this document for simplification.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADT</td>
<td>Android Developer Tools</td>
</tr>
<tr>
<td>AP</td>
<td>Access Point</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>APP</td>
<td>Application</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>DIO</td>
<td>Data Input/Output</td>
</tr>
<tr>
<td>GO</td>
<td>Group Owner - <em>The master of Wi-Fi direct connexion</em></td>
</tr>
<tr>
<td>GPIB</td>
<td>General Purpose Interface Bus - <em>also called IEE-488</em></td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Input/Output</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>JTAG</td>
<td>Joint Test Action Group</td>
</tr>
<tr>
<td>LDO</td>
<td>Low-Dropout (regulator)</td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine to Machine</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static Random Access Memory</td>
</tr>
<tr>
<td>SSID</td>
<td>Service Set Identifier</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>Wi-Fi P2P</td>
<td>Wi-Fi Peer-to-Peer - <em>the term Wi-Fi Direct is used in this document</em></td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>XF</td>
<td>eXecution Framework</td>
</tr>
</tbody>
</table>
1 Introduction

GPIB (General Purpose Interface Bus), or IEEE-488 is a short-range digital communications bus specification. It was created in 1965 by HP and standardized in 1975, developed for use with automated test equipment and is still in use for that purpose nowadays.

Different measure instruments in the ARE (Applied Research on Electromagnetics) lab in the Zhejiang University are equipped with a GPIB connector. Usually this standard require the use of large and heavy cables to connect the instruments together or to a computer.

The aim of this project is to develop a wireless communication interface allowing a smartphone to connect on a GPIB bus. By developing a small hardware with a wi-fi module and GPIB transceivers, a smartphone will be able to communicate on the GPIB bus. The figure 1 above explain it.

In this project, only an Android application was developed for the communication with the hardware. But the hardware could be use with every smartphone equipped with the Wi-Fi technology (iPhone, BlackBerry, Windows phone, etc...).

The small hardware, combined with an Android APP will provide a wireless interface between the Smartphone and a GPIB bus. It avoid the use of cable and offer a new way of controlling a GPIB bus.

The APP must offer a user friendly interface to connect to the hardware via WLAN standard, write and read on the GPIB bus.
2 OBJECTIVES

The final objective of the project is to provide a Smartphone/GPIB interface.

A small hardware and an Android application will allow the user to connect easily on a GPIB bus to send and receive data and commands. The device developed must offer a new and easier way to connect and get data from the instruments. The circuit must provide a transparent connection between the smartphone and the GPIB bus.

The project can be separated in three different main objectives:

♦ Development of a small hardware circuit able to allow a connection to a smartphone and to a GPIB bus.
♦ Development of the software for the MCU. The software must be able to communicate and exchange data with the smartphone via a Wi-Fi module and be able to read and write on the GPIB bus.
♦ Development of an Android application, able to communicate wirelessly with the MCU through the Wi-Fi module.

In order to accomplish these different parts of the project, the following steps must be followed.

♦ Make research about GPIB communication.
♦ Find a Wi-Fi chipset able to allow a connection with a smartphone and a MCU.
♦ Find a MCU able to handle the both communications, with the smartphone and with the GPIB bus.
♦ Develop the schematics of a small hardware to plug on GPIB connector.
♦ Design the circuit board and manufacture it.
♦ Develop the MCU software able to handle communication with the Wi-Fi module and the GPIB bus.
♦ Develop and test an Android application able to connect to the Wi-Fi chipset, and exchange data and commands with the GPIB bus through the MCU.
3 Research and Specifications

For this project, research must be done in order to find the most suitable components. The main components for this application are the wireless chipset that will allow communication with the smartphone and the MCU that will allow communication with the GPIB bus.

3.1 GPIB standard

To allow communication between the hardware board and the instruments, the circuit must be designed according to the GPIB standard.

Research on the GPIB standard is necessary in order to find the most suitable MCU for the communication with the GPIB bus.

3.1.1 GPIB in brief

GPIB is a 8-bit, electrically parallel bus with a maximum data rate of 1MB/s. The slowest participating device determines the data rate of the bus.

The protocol allows 1 controller and 1 to 14 other devices to share a single physical bus of up to 20 meters total cable length. [1]

An effective communication on the bus requires three basic elements to organize and manage the flow of information exchanged among devices:

♦ A controller
♦ A talker (Also called source)
♦ One or more listener (Also called acceptor)

The controller can send different commands to the other devices. A talker (not necessarily the controller) can write data on the bus that will be read by the listener(s).
3.1.2 Connector

The figure above describe the pins alignment in the GPIB connector.

- Pin 1: DIO1 Data input/output bit.
- Pin 2: DIO2 Data input/output bit.
- Pin 3: DIO3 Data input/output bit.
- Pin 4: DIO4 Data input/output bit.
- Pin 5: EOI End-or-identify.
- Pin 6: DAV Data valid.
- Pin 7: NRFD Not ready for data.
- Pin 8: NDAC Not data accepted.
- Pin 9: IFC Interface clear.
- Pin 10: SRQ Service request.
- Pin 11: ATN Attention.
- Pin 12: SHIELD
- Pin 13: DIO5 Data input/output bit.
- Pin 14: DIO6 Data input/output bit.
- Pin 15: DIO7 Data input/output bit.
- Pin 16: DIO8 Data input/output bit.
- Pin 17: REN Remote enable.
- Pin 18: GND (wire twisted with DAV)
- Pin 19: GND (wire twisted with NRFD)
- Pin 20: GND (wire twisted with NDAC)
- Pin 21: GND (wire twisted with IFC)
- Pin 22: GND (wire twisted with SRQ)
- Pin 23: GND (wire twisted with ATN)
- Pin 24: Logic ground

3.1.3 Pins

This chapter is presents the GPIB as explained one the document CEC-488 Programming and Reference [2, chapt. 6-7].

The connector uses 24 wires, separated in 4 groups.

8 bidirectional and asynchronous data lines:

- **DIO 1-8 (Data Input/Output)** - This lines read or written by the different devices connected to the bus in order to exchange data or commands.

3 lines used for the handshake (necessary for each byte transmission):

- **DAV (Data Valid)** - Indicates availability and validity of information on the DIO lines.
- **NDAC (No Data Accepted)** - Indicates the acceptance of data by all devices.
- **NRFD (Not Ready For Data)** - Indicates that all devices are not ready to accept data.

5 lines used to manage the bus:

- **ATN (Attention)** - Indicates whether the current byte is to be interpreted as data or a command. When asserted with EOI it indicates that a parallel poll is in process.
EOI (End Or Identify) - Indicates the termination of a data transfer. When asserted with ATN, it indicates that a parallel poll is in process.

IFC (Interface Clear) - Asserted only by the system controller to take unconditional control of the bus. The bus is cleared to a quiescent state and all talkers and listeners are placed in an idle state.

REN (Remote enable) - Allows instruments on the bus to be programmed by the active controller (as opposed to being programmed only through the instrument controls). System controller drives the REN line to place devices in remote or local program mode.

SRQ (Service Request) - Used by a device to asynchronously request service from the active controller.

And finally, 8 lines connected to the ground (GND and Shield). *note that the shield is usually connected to digital ground through a jumper. This jumper may be removed to connect the shield to chassis ground for example [2, chapt. 6-6]*

### 3.1.4 Specifications

The GPIB uses negative logic with standard TTL levels. For example, when DAV is true, it is a TTL low level, when false, it is a high level.

The GPIB electrical characteristics are [1, p. 62]:

<table>
<thead>
<tr>
<th>Single Type</th>
<th>Voltage Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Voltage High</td>
<td>$V_{IH} = 3.4V$ typical, 2.0V minimum</td>
</tr>
<tr>
<td>Input Voltage Low</td>
<td>$V_{IL} = 0.22V$ typical, 0.8V minimum</td>
</tr>
<tr>
<td>Output Voltage High</td>
<td>$V_{OH} = 3.4V$ typical, 2.4V maximum</td>
</tr>
<tr>
<td>Output Voltage Low</td>
<td>$V_{OL} = 0.22V$ typical, 0.5V maximum</td>
</tr>
<tr>
<td>Maximum Voltage</td>
<td>$V_{Max} = 5.25V$</td>
</tr>
</tbody>
</table>

### 3.1.5 GPIB handshake sequence

With the GPIB standard, every transmitted byte (data or command) undergoes a handshake sequence. This sequence is used to ensure that all listeners are ready to receive data from the talker, and that the data is transmitted properly.

The figure 3 shows the handshake timing sequence. [2, chapt.6-4]. A complete logical flow diagram is provided by the IEEE488 standard. [1, p. 87]

![Figure 3: GPIB - Handshake timing sequence](image)

The figure above indicates the state of the source (on the top), and the state of the acceptor (on the bottom) during the handshake sequence.
The sequence take place like this:

1. The source is initially in the Source Generate State (SGNS). In this state, the source does not assert the data lines or Data Valid (DAV). The acceptors are in Acceptor Not Ready State (ANRS), the Not Ready For Data (NRFD) and Not Data Accepted (NDAC) are asserted.

2. When the source wants to write a byte, it enters in Source Delay State (SDYS) and assert data lines. Note: if this is the last byte to send, source may also assert End Or Identify (EOI)

3. When they see that data have been asserted by the source, all the acceptors release their Not Ready For Data (NRFD) lines, and enter in Acceptor Ready State (ACRS). Note: Any acceptor can delay the handshake sequence, by not releasing its NRFD line, this is how the GPIB standard adapt to the slowest device.

4. When all the acceptors have released NRFD and the data asserted are correct, the source enters in Source Transfer State (STRS) and asserts Data Valid (DAV).

5. Acceptors enter in Accept Data State (ACDS) and assert NRFD since they are busy with the current data byte.

6. When acceptors accept the data, they release NDAC and move to Acceptor Wait for New cycle State (AWNS).

7. When all the acceptors have accepted the data, the source enters in Source Wait for New cycle State (SWNS) and release the Data Valid (DAV).

8. Source and acceptors return in their initial states.

The hardware developed must be able to handle this handshake sequence as a source or an acceptor in order to write and read on the GPIB bus.
3.2 Wi-Fi chipset

The Wi-Fi chipset allows the communication between the circuit and the Android smartphone. This is therefore one of the most essential components of the project. Research is needed in order to find the most suitable module for this application.

3.2.1 Technologies

There is different wireless technologies usable for a communication with a smartphone: Bluetooth, WLAN, Wi-fi direct, etc...

These different technologies have their own advantages and disadvantages.

♦ Bluetooth: It is easy to use and develop for the Android APP and the hardware board, but transmission speed and range are limited (up to 3Mbit/s and a few meters) [3, chapt. 1.2].

♦ ZigBee: ZigBee node are cheap, even cheaper than Bluetooth, but the speed of transmission is really limited (up to 250kbit/s)[3, chapt. 1.2].

♦ Wi-Fi direct (also called WLAN direct or Wi-Fi P2P): Fast transmission and easy to use (can be compared to Bluetooth), but chipsets are expensive.

♦ Wi-Fi via Access point: Fast transmission, chipsets are expensive.

The project specifies the use of WLAN standard, therefore the Bluetooth can not be used. Plus, the GPIB transmission rate can be up to 8Mbit/s, even if the transmission rate on GPIB bus is determined by the slowest participating device, ZigBee seems too limited for this project.

The Wi-Fi direct is the best technology to use for this project. It allows a simple connection between the smartphone and the hardware device. The communication can be fast and does not need an external AP. A standard Wi-Fi connection is also possible, but require an AP.

3.2.2 RS-WC-201

There is a lot of different chipsets available and suitable for this project. The first chosen module is the Redpine Signals’ RS-WC-201. This 95 pins module belongs to the WiSeConnect family of advanced Wi-Fi modules. [4]

It is a fully integrated 802.11 b/g/n module with advanced features for M2M, industrial, medical, enterprise and IoT applications. It can directly communicate with smartphones and tablet via Wi-Fi Direct. It integrates a MAC, Baseband Processor, RF Transceiver with power amplifier, a frequency reference and an antenna. This module comes with a comprehensive API set to make software integration quick and seamless, this will save time for this project. It can communicate with a Host Processor via UART or SPI.

3.2.3 Problem with RS-WC-201

As explained on chapter 9.1.2, problems have been encountered with the RS-WC-201 Wi-Fi module. Once the module soldered on the PCB, no communication have been able to be established with the MCU via SPI or UART interface. ¹

¹The hardware circuit has been designed for the RS-WC-201, all calculations for power, current, etc... use the RS-WC-201 characteristics.
3.2.4 RAK411

Another module is used in order to replace the RS-WC-201 in order to allow communication between the circuit and the smartphone.

The RAK411, from Chinese manufacturer RAK Wireless technology ltd has been chosen. It is a Wi-Fi module, fully compliant with IEEE 802.11b/g/n wireless standards, with internally integrated TCP/IP protocol stack, supporting protocols such as IP, TCP, UDP, DHCP CLIENT, DHCP SERVER, DNS. It supports AP mode, Station mode and Ad-hoc and mode. [5]

This module is not equipped with the Wi-Fi Direct technology, but as it supports AP mode and offer the possibility to open TCP socket as server, the communication with a smartphone is possible.

This module has been chosen because of the similarity with the RS-WC-201 in the communication interface, which simplify the integration in the developed hardware. It uses the SPI to communicate with a host MCU.

3.3 MCU

The microcontroller is the centrepiece of the circuit, it handles the communication with the measure instruments via the GPIB bus and with the Android smartphone through the Wi-Fi module.

3.3.1 Choice of MCU

The microcontroller must be able to:

♦ initialize the Wi-Fi module and configure it.
♦ Send and receive data to the smartphone through the Wi-Fi module.
♦ Reset the Wi-Fi module.
♦ Read and write on all GPIB pins.

In order to do that, the module needs a SPI interface to communicate with the Wi-Fi module. The Wi-Fi module's reset input must be mapped to a MCU controllable line, so that the system can recover through a hard reset.

The MCU needs GPIO pins to access to GPIB pins, 8 for data, 5 for interface gesture and 3 for handshake (The 8 GND lines do not need to be accessed). The pins must be accessed in read and write mode. Therefore, the MCU needs a certain amount of GPIO.

In the ARE lab, Atmel MCU have already been used and software development environment is available. For this reason, and according to the number of I/O required, the ATmega164PA has been chosen. This component is equipped with [6]:

♦ 32 Programmable I/O lines, enough to access to the GPIB, the Wi-Fi module and other use (indications LEDs, etc...). All I/O lines have interrupt on pin change, useful for the GPIB handshake and interface gesture.
♦ A master/slave SPI and two programmable USART, to access to the Wi-Fi module
♦ A JTAG interface, for programmation and debug.
♦ 1 KB of internal SRAM.

The frequency of the MCU can be up to 10MHz with 3.3V power supply, enough for the gesture of the GPIB bus.
3.4 GPIB drivers

Since the chosen MCU’s I/O pins are not open drain, they can’t be used directly to communicate with the GPIB bus. Transceivers must be used in order to provide a GPIB interface to the MCU.

The DS75160 and DS75161 (or SN75160/SN75161) are specific GPIB Transceivers. This two devices form a complete 16-lines interface between the GPIB bus and the MCU [7]. The 75160 IC is used for the transmission and reception of the 8 DIO of the GPIB bus, and the 75161 for the interface gesture.

The TE pin (Talk Enable) of the both chip allow to configure the MCU as a talker or a listener on the GPIB bus. The level of TE will configure the DIO and the handshake lines as reception or transmission. see chapter 3.1.5 for handshake sequence.

The PE pin (Pull-up Enable) configure the output as totem-pole or open collector (open drain), as explained on figure 4.

![Control Input Level - Data Transceivers](image)

Figure 4: DS75160A functional truth table

DC (Direction Control) is used to configure the MCU as active controller of GPIB bus. It chose the interface gesture signal direction, as explained on the figure 5.

![Control Input Level - Transceiver Signal Direction](image)

Figure 5: DS75161 functional truth table
3.5  Power supply

The GPIB bus does not provide a power supply, all GPIB instruments must be self powered. Therefore, a power source is needed for the circuit.

3.5.1  Power source

The power of the hardware circuit can be supplied in several ways :

♦ Power cable (e.g, micro-USB cable).
♦ Battery.
♦ Accumulator.
♦ Using power from other instrument's connector.

In order to determine the values for the battery/accumulator to use, an estimation of the power consumption of the device must be done.

The current consumption of the RS-WC-201, the Wi-Fi module, is as follow : [8, p. 23]

<table>
<thead>
<tr>
<th>Power Save State</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep Sleep</td>
<td>2.3 mA</td>
<td>The module is put in sleep mode by the MCU and can be woken up at any time</td>
</tr>
<tr>
<td>Continuous transmission</td>
<td>370 mA</td>
<td>Module transmitting data continuously at 54Mbps</td>
</tr>
<tr>
<td>Continuous reception</td>
<td>226 mA</td>
<td>Module receiving data continuously</td>
</tr>
<tr>
<td>Shut down</td>
<td>110 µA</td>
<td>Complete shut down, when woken up by the MCU, initiates a boot-up sequence</td>
</tr>
</tbody>
</table>

In comparison with this module, the power consumptions of other parts of the circuit (MCU, GPIB drivers, etc...) are almost negligible for the estimation.

As the current consumption is really high for an embedded system, the battery or accumulator would need a very high capacity to last at least a few hours. The use of an accumulator would imply the development of a charger on the circuit and would take longer to develop.

The use of a micro-USB connector for the power supply has been chosen because it gives the possibility to use a standard smartphone charger or a USB port from the measure instrument and provides a reliable 5V voltage.

3.5.2  Supply voltages

The USB standards specify that the voltage level is from 4.75V to 5.25V [9, p.199], which means it can be used directly by the GPIB drivers, without level regulation [7, p.3].

The Wi-Fi module requires a 3.3V power supply, the same is used for the MCU.
3.5.3 DC Regulator

In order to get the 3.3V power supply from the 5V provided by the micro-USB connector, a voltage regulator is used.

The regulator must be able to provide the current needed by the Wi-Fi module and the MCU (Max. 400 mA). The LM1117-3.3V is suitable for this application. It can provide a current up to 800 mA.

**Thermal relief**

The LM1117, with the package SOT-223 and without heat sink has the following thermal characteristics:

\[ \Theta_{JA} = 136^\circ C/W \]

The maximum power to be dissipated by the LDO is calculated as follow:

\[ P_{D_{max}} = (V_{in_{max}} - V_{out}) \times I_{L_{max}} = (5.2 - 3.3) \times 0.4 = 760mW \]

Where \( V_{in_{max}} \) is the maximal input voltage of the USB power supply, and \( I_{L_{max}} \) is the estimated maximum value of the current consumption during full time transmission on the Wi-Fi module and all LEDs on.

The maximum allowable temperature rise in the LDO, \( TR_{max} \) is the difference between the maximum allowable junction temperature, 125°C and the ambient temperature, 25°C, so 100°C.

The maximum allowable value for the junction-to-ambient thermal resistance (\( \Theta_{JA} \)) is calculated as follow:

\[ \Theta_{JA} = TR_{max} / P_{D_{max}} = 100 / 0.760 = 131^\circ C/W \]

It means that a copper area must be used as heat sink for the LDO in order to lower the junction-to-ambient thermal resistance.

---

2. This regulator has been chosen for availability in the lab.
4 CIRCUIT DEVELOPMENT AND SCHEMATICS

The following figure represents the diagram of the circuit developed. It shows the main components allowing to create a hardware circuit able to provide a communication interface between a smartphone and a GPIB bus.

The schematics is developed with OrCAD Capture.

The complete schematics of the circuit can be found in appendix [11].
4.1 Wi-Fi module - RS-WC-201

The RS-WC-201 Module Integration Guide shows a reference schematic [12, p. 6]. It has been used as a base for the hardware development of Wi-Fi module part. The module is supplied by the 3.3V provided by the LDO and has its UART and SPI interfaces connected to the MCU.

4.1.1 Mode

Different modes are available for the communication with the module. These modes can be selected with the three MODE-SEL pins. A DIP-Switch is used to allow the user to chose between the UART, SPI and USB interface. Only the UART and the SPI are implemented for the communication with the MCU. The mode must be chosen before the reset of the module, and can not be changed during its working.

![Figure 7: Mode selection schematics](image)

<table>
<thead>
<tr>
<th>MODE</th>
<th>MODE-SEL-2</th>
<th>MODE-SEL-1</th>
<th>MODE-SEL-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SPI</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>USB</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

4.1.2 Firmware upgrade

The RS-WC-201 module offers the possibility to be upgraded. The Module Integration Guide propose to use a MAX3232 and a DB9 connector to connect the module to the PC via UART interface and upgrade the firmware with Redpine Signal's software. [12, p. 6]

In order to save place on the circuit board, rather than directly use the DB9 connector and the transceiver, a header connector allows to connect the RX, TX, 3V3 and GND of the circuit to a module integrating DB9 connector and MAX232, shown on the figure 8.

![Figure 8: Extern module with MAX3232 and DB9 for firmware upgrade](image)

4.1.3 Display LEDs

Two LEDs are controlled by the Wi-Fi module, with the pins CARD-READY and WIFI-ACTIVITY, respectively to indicate that the boot-up of the module has been successful and that the module is wirelessly transferring data.
4.2 Wi-Fi module - RAK411

The module RAK411 has been used after the problem encountered with the RS-WC-201.  

4.2.1 Integration on the existing PCB

The following figures show the package of the RAK411 and its pins definition.

![Figure 9: RAK411](image)

<table>
<thead>
<tr>
<th>Pin Serial No.</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
<td>Ground</td>
<td>connected to ground pad or the copper</td>
</tr>
<tr>
<td>2</td>
<td>VCC3V3</td>
<td>Power</td>
<td>3.3V power supply</td>
</tr>
<tr>
<td>18</td>
<td>LINK</td>
<td>O, PU</td>
<td>&quot;0&quot; - STA connected in AP mode, Connected to router in STA mode &quot;1&quot; - disconnected Remain disconnected when no use</td>
</tr>
<tr>
<td>19</td>
<td>RESET</td>
<td>I, PU</td>
<td>Module reset pin, low effective</td>
</tr>
<tr>
<td>27</td>
<td>SPI_INT</td>
<td>O</td>
<td>SPI mode interrupt pin &quot;0&quot;—idle level “1”—has data sent to host</td>
</tr>
<tr>
<td>31</td>
<td>SPI_MISO</td>
<td>O</td>
<td>SPI slave: data of SPI Master Input, Slave Output</td>
</tr>
<tr>
<td>32</td>
<td>SPI_MOSI</td>
<td>I</td>
<td>SPI slave: data of SPI Master Output, Slave Input</td>
</tr>
<tr>
<td>33</td>
<td>SPI_CLK</td>
<td>O</td>
<td>SPI slave: SPI clock input</td>
</tr>
<tr>
<td>34</td>
<td>SPI_CS</td>
<td>I</td>
<td>SPI slave: SPI chip select input</td>
</tr>
<tr>
<td>Others</td>
<td>NC</td>
<td>NC</td>
<td>Remain disconnected when no use</td>
</tr>
</tbody>
</table>

Figure 10: RAK411 - Pins definition

---

3See on chapter 9.1.2
A evaluation board of the RAK411 is available.

![Image: RAK411 Evaluation Board]

Although the PCB designed during the project is made to be used with the RS-WC-201, it is modified with wires in order to provide a header with all the necessary connection for the Wi-Fi module (VCC, GND, SPI, RESET, etc.).

The RAK411 can be connected to this header thanks to the evaluation board and be used for the prototype.

A new PCB must be designed in the future to integrate the RAK411 module only.

**4.2.2 Firmware upgrade**

The firmware of the RAK411 can be upgraded wirelessly.

**4.2.3 Display LED**

The LINK (Pin 10) of RAK411 module indicates, in AP mode if a station is connected, or in station mode, if the module is connected to router. One of the LED expected to be used with the RS-WC-201 can be used to display the state of LINK.
4.3 MCU

4.3.1 Programming

In order to program the MCU, serial downloading is used. This can be done through the SPI pins, SCK, MISO and MOSI, while RESET is pulled to GND.

A 6 pins header is used to allow connection of the programmer on SPI (VCC, GND and the MOSI, MISO and RESET pin of the MCU).

Figure 12: MCU schematics with header for programming
4.3.2 Clock

The speed grade of the ATmega164PA depends on the power supply voltage, as shown on the figure below.

[6, p. 336]

![Figure 13: Maximum frequency vs Vcc](image)

The minimum output voltage of the LDO is 3.235V [10, p. 4].

As specified by the manufacturer, the curve is linear between 2.7 and 4.5V, which means that, at 3.235V, the maximum frequency for the clock is 12.97MHz. It can be considered as safe to use a clock of 12MHz or lower for the MCU.

This MCU has a **calibrated internal RC oscillator**, with a frequency range of 7.3 to 8.1 MHz [6, p. 36]. This clock can be used and is high enough for the GPIB communication.

Although, the schematic and the PCB are designed to let the possibility to use an external oscillator, to have a higher range and a more accurate frequency.

4.3.3 Display LEDs

In order to test the device and to provide information to the user, two LEDs can be controlled by the MCU. The ATmega164PA pin driver is strong enough to drive LED directly [6, p. 72]

4.3.4 Button

A button is connected to the MCU to let the possibility to implement different command from the user to the MCU (Turn ON, Turn OFF, Standby mode, reset, etc...).

Internal pull-up are available inside of the MCU, so there is no need to use an external pull-up for the button, the button is directly connected between a MCU input and the ground.

The button is connected to an external interruption pin (PB2, INT0) of the MCU to be easily treated by the software.
4.4 GPIB bus

The DS75160A and DS75161 are 20 pins IC. 8 pins of each can be connected directly on the terminal (the MCU) and 8 on the bus (GPIB). [7]

On the terminal side (MCU), the 8 DIO are connected on the 8 pin of PORTA, and the 8 interface gesture are connected on PORTC.

On the bus side, the pins are directly connected to the corresponding pins on the GPIB standard connector.

The following figure show the schematics of the GPIB, the connection between the drivers and the MCU and the connector.

The **TE** pin (Talk Enable) of both chip are connected to the same MCU output, in order to configure the MCU as a talker or a listener on the GPIB bus. The **PE** pin (Pull-up Enable) of the DS75160A and the **DC** pin (Direction Control) are both connected on a MCU output.
5 PCB

The PCB is designed with OrCAD Layout Plus. The netlist is directly imported from the OrCAD capture schematics.

The circuit design developed can be found in appendix [13].

5.1 Dimensions

The device will be used without housing, therefore, there is no constraints in the dimensions for the board development. The prototype’s PCB is developed on one side only with different test points to facilitate the measures.

Three mounting holes are used to attach the board during the test.

5.2 Components placement

5.2.1 Connectors

The connectors and the button must be accessible for the user.

The micro-USB and GPIB connectors are both placed on the edge of the board.

5.2.2 Jumpers

The jumpers allow the user to change the configuration on the device (Disable the Wi-Fi module, connect UART to the PC, change Wi-Fi module’s programming mode, etc...). They need to be accessible for the user.

5.3 Wi-Fi module

The RS-WC-201 Module Integration Guide gives some guidelines for the circuit and layout design [12, p. 13-14]. It has been used as a base for the PCB development.

5.4 Copper ground

In order to avoid the noise from the digital circuit of the Wi-Fi module, the copper ground of the MCU is separated from the ground of the Wi-Fi module.

5.5 Heatsink

As explained in the chapter 3.5.3, a heat sink must be drawn on the PCB for the thermal relief of the LDO, in order to obtain a $\Theta_{JA}$ under 131°C/W.

With a 0.2 in$^2$ (1.29 cm$^2$) copper area on the bottom, the $\Theta_{JA}$ becomes 115°C/W [10, p. 12].

The copper area has been extended on all the space available, and is 2 cm$^2$ on the bottom and 0.5 cm$^2$ on the top.
6 INTERFACE SPECIFICATIONS

The aim of this project is to provide the user a working Smartphone/GPIB interface. The interface is composed of the hardware circuit plugged on an the GPIB bus and an Android APP. The figure above represent the interface provided to the user.

If GPIB instruments used the same commands, the command would be stored in the MCU or inside the application. But since instruments from different manufacturers use different GPIB commands, it is not possible to develop an interface directly usable with every different instrument.

6.1 Library

In order to provide an interface that does not require to change the code of the hardware board or the smartphone application to access a specific instruments, the system must allow the user to easily configure the APP to access his instrument.

The configuration contains the different GPIB commands of the instrument. The configuration is saved in the form of library. Library can be created and load directly on the APP with a Library Manager.
6.2 Wireless transmission protocol

A protocol is used for the transmission between the smartphone application and the hardware circuit. This protocol specifies the different commands that can be sent from the smartphone and the corresponding responses from the MCU.

6.2.1 Write on GPIB bus

The write command is used when the user wants to send data or command to the instrument on the GPIB bus.

```
WRITE  CMD/DATA  LENGTH  DATA 0  ...  DATA n
```

Figure 17: Write command packet

The packet start with one byte code WRITE to indicate to the MCU that user want to write on the bus. The second byte indicates if the bytes to write are a command or a data for the instruments. One byte LENGTH indicate the length of byte to write on the bus.

The packet then contains all the bytes to write on the GPIB bus.

Once the data or command written, the MCU acknowledges the write packet by sending a response packet.

```
RESPONSE WRITE STATUS
```

Figure 18: Write response

All responses start with a RESPONSE byte indicating to the smartphone application that the MCU is responding to a command. In the case of a write response, the second byte contains the WRITE code. A status byte indicates the status of the circuit, to let the smartphone know if there is a problem writing on GPIB, or if the packet was not properly received.

6.2.2 Read on GPIB bus

The read command is used when the user wants to read bytes from the instrument on the GPIB bus.

```
READ  LENGTH
```

Figure 19: Read command

The packet start with on byte code READ to indicate to the MCU that user want to read on the bus. The second byte LENGTH indicate the number of bytes to read.

```
RESPONSE READ STATUS DATA 0 ... DATA n
```

Figure 20: Write response
In the case of a read response, the second byte contains the READ code. A status byte indicates the status of the circuit, to let the smartphone application know if there is a problem reading on GPIB, or if the packet was not properly received.

The packet contains then all the byte read from the GPIB bus.

### 6.2.3 Set LEDs state

A command can be use to turn on or turn off the LEDs of the circuit. It is really convenient for the tests.

The packet is composed of the LED byte code, then the states of the LEDs to set, ON or OFF (respectively 0x01 and 0x00).

The MCU acknowledges the command by sending a response containing the LED code and the **real states** of the LEDs after the command.

### 6.2.4 Packets codes and status

The different codes and status available in this protocol are presented on the following figure.
7 MCU SOFTWARE

The processing unit of the circuit is the ATmega164PA [6].

As central control unit of the board, the MCU has to communicate with the Wi-Fi module through SPI interface in order to communicate with the Android smartphone. It must also communicate with the GPIB bus via the transceivers.

The MCU software is in charge of the following tasks:

- The startup, configuration of registers, GPIO, SPI and GPIB communication.
- Initialization of the Wi-Fi module to enable a connexion with the smartphone.
- Handle the press on the button by the user (enter/exit of sleep mode, standby, etc...).
- Handle the LEDs to give information to the user.
- Read and write on the GPIB bus.
- Communication with smartphone data via SPI interface.

The MCU code is written and compiled with AtmelStudio 6.2. The HEX file generated is then downloaded on the MCU via SPI interface, with a chinese software, Xuan Wei programmer control platform. The code can be found in appendix [14] [15] [16].

7.1 Startup

The first function launched when the MCU is turn on is the main(). The init() method is called in order to set all the registers and I/O to their initial values.

7.1.1 Hardware initialization

Clock:

In order to use the calibrated internal RC oscillator as clock for the MCU, the CKSEL3:0 fuses are configure to 0010. The frequency of the clock goes from 7.3 to 8.1 MHz. The frequency is calibrated by configuring the OSCCAL register to get an accurate 8MHz clock, in order to have a reliable time base.

I/O:

All the pins are either configured as inputs or outputs. The LEDs are configured as outputs and the button as an input. The control signals of the GPIB transceivers (TE, PE and DC) are also configured as output.

Since the MCU must be able to act as a source or acceptor on the GPIB bus, the configuration of DIO and management signals of the GPIB bus will change and is not made during the startup.
Interruptions :

The EICRA register allow to enable the external interrupts. INT0 is on pin PD2 and correspond to the button. INT2 is on pin PB2 and correspond to the WIFI-INTR signal from the module, to indicate to the MCU that there is data to be read on the Wi-Fi module.

The PCICR register allow to enable the on-pin-change interrupts. This interrupts will indicates to the MCU when GPIB management signals change.

Timers :

The ATmega164PA provides two 8 bits timers (TIMER0 and TIMER2) and two 16 bits timer (TIMER1 and TIMER3).

One timer is used to provide the time base to the XF. This is the only hardware timer used, the XF will allow the use of different software timers.

The time base is chosen to allow some precision to the software timers, but since the MCU clock is not so high, the time base must be slow enough to allow the MCU to execute enough instructions between two timer interruptions. A 10 milliseconds time base is a good compromise.

The timers can be configured in CTC mode (Clear Timer on Compare Match), in order to define a specific time. The counter is cleared and triggers an interruption when it reaches the specified value. The period is defined with the following formula :

\[ T_{\text{timer}} = \frac{N \times (1+\text{OCR}nx)}{f_{\text{clk}}} \]

Where \( N \) represents the prescale factor (1, 8, 64, 256 or 1024). and \( \text{OCR}nx \) the value to reach with the counter.

In order to get a 10ms timer, the best solution is to use the Timer1, because the 16 bits counter allow a more accurate value.

The OCR1 value is calculated as follow, with a prescaler of 8 :

\[ \text{OCR1}x = \frac{T_{\text{timer}} \times f_{\text{clk}}}{N} - 1 = \frac{10 \times 10^{-3} \times 8 \times 10^6}{8} - 1 = 9999 = 0x270F \]

SPI :

The SPI is used as communication interface with the Wi-Fi module and is initialized in order to be the faster possible and comply with the specifications of the Wi-Fi module :

- Operation type : The SPI is configure in Master mode, the Wi-Fi module is the slave.
- Transmission speed : \( f_{\text{osc}}/2 = 4\text{MHz} \) (the maximum SPI clock for the Wi-Fi module is 16MHz).
- Clock polarity : CPOL = 0, SCk is idle on low level
- Clock phase : CPHA = 0, Data is latched on clock rising edge an transmitted on clock falling edge.
- Data order : MSB first

---

\(^4\)See chapter 7.2
7.1.2 XF initialization

The XF, see section below, is initialized with the method \texttt{XF\_init}. It empties the events queue and the timer lists.

7.2 XF

An execution framework (XF) is a very simple OS. It simplifies embedded systems design by offering the possibility to implement states machines on very small MCU. The software is more reliable and easier to improve and work on it.

The main reason why a XF is used in this software is to avoid blocking methods. For example, sometimes the Wi-Fi module does not acknowledge the commands immediately. The manufacturer recommends to wait 50ms before sending it again. If the process was completely stopped during this time, this would decrease the performances of the system.

Thanks to the use of different states machines for each tasks (I/O, GPIB and Wi-Fi module management), none of them will block the program when waiting.

The XF used was developed by prof. Medard Rieder, at the HES-SO Valais. It has been adapted for the ATmega164PA and offer the possibility to use states machines, schedule and unschedule timers.

The following UML class diagram shows the functions available with the XF.

![Figure 24: Class diagram : XF](image)

The XF contains a queue of events. The function \texttt{XF\_popEvent} allow the state machine to get the event that happened and treat it. Events are put in the queue using \texttt{XF\_pushEvent}. The logic behind these two functions ensure a reliable manipulation of the events, by disabling interruptions if needed.

The tempo is given by a hardware timer. Each 5ms, \texttt{XF\_ISR} is called and the software timers, scheduled with \texttt{XF\_scheduleTimer} are decremented. Timer can also be unscheduled with \texttt{XF\_unscheduleTimer}.

Once a software timer’s time has elapsed, the event associated is put in the events queue and will be treat by the state machine.
7.2.1 Program tasks

The program is divided in three main tasks:

♦ I/O Management: This task handles the different inputs and output on the MCU, it can turn on and off the LEDs and manage the button when pressed.

♦ RAK Management: This task handles the communication between the MCU and the Wi-Fi module with the SPI. It allows to send commands to the module and receive responses. Via this module, it can communicate with the Smartphone. This task implements the transmission protocol described in chapter 6.2 in order to communicate with the smartphone.

♦ GPIB Management: This task handles the communication with the GPIB bus. It allows to write or read data and commands on the bus.

As shown on the figure 25, after the initialization, the program enters in an infinite loop. The events are taken from the XF events queue and dispatch in every task. Each task is another states machine. The tasks only perform a few instructions, depending of their states, then another tasks is launched. This way, the tasks do not block the others.

The tasks run independently, and can interacts by pushing events in the XF. For example, when the GPIB Manager finished to read data from the GPIB bus, it push an event to indicate to the RAK Manager that there is data to be sent to the smartphone.

7.3 RAK411 commands

The SPI is used for the communication with the Wi-Fi module. The module must be initialized and configured with different commands.

The manufacturer provides an API for the developer, containing all the functions and structures to communicate with the Wi-Fi module. Unfortunately, this API is made to be used by more powerful MCU, the ATmega164 does not have enough program memory and RAM to use it. Besides, the functions provided by the manufacturer are blocking functions and are not suitable for the software structure with the XF.

this is why the different functions and structures have been implemented differently in order to optimize the resource management and get a suitable software for the MCU.

All the different commands available for the module have been implemented in the software.

The files RAK411.h and RAK411.c [14] define all the different commands to be sent to the module, and offer the different functions to easily create the structure, put the desired parameters and send the commands/structures via SPI.

[5] All the commands for the module can be found in the RAK411 programmation manual [17]
7.3.1 Commands

Each RAK411 command has a corresponding 1 byte code. (e.g. init command code is 0xA0).

The following figure explains how the commands are sent to the RAK411 module.

![Figure 26: Sending command to RAK](image)

Some commands does not need parameters to be sent, so there is no data to send, just the 4 bytes header, composed of the command code, the command fixed value (0x97) and two bytes for the length (0x0000 in the case of no parameters).

Some commands require parameters to be sent. The header is sent, and when the module acknowledges the command, parameters are sent.

Some commands ask for a return value. The following figure shows how the MCU can read the values from the module.

![Figure 27: Receiving response values from RAK](image)

0xA2 is the command code to read on the module. When the module acknowledges the reading, it will send the length of the response on 2 bytes, then the data of the responses.
7.3.2 Command and response structures

In order to facilitate the software development, structures are used to declare all the commands with their parameters. The RAK411.h [14] file contains all these structures, presented on the following form.

```c
typedef struct{
  uint32_t cmd;
  type1 param1;
  type2 param2;
  ...
} cmd_t;
```

Figure 28: Structure commands

As SRAM is limited on the MCU, a union with all the commands existing is used. This way, the size used for the command is only equal to the size of the biggest command.

Structures are also used in order to save the responses from the module.

```c
typedef struct{
  uint8_t rspCode[2];
  type1 param1;
  type2 param2;
  ...
} cmdResponse;
```

Figure 29: Structure commands

A union with all the possible responses is also used in order to save memory.

7.3.3 Setting structures

In order to put the desired parameters in the structure of a command, different methods have been implemented with the following prototypes:

```c
void set_structurename_cmd(structure *struct, type1 param1, type2 param2, ...);
```

Figure 30: Prototype of the different methods to set structure commands parameters

These methods allow to easily put the parameters in the structures. Only the integer (uint8_t, uint16_t and uint32_t) can be put in the structure with these methods. The arrays must be set directly by accessing to the array pointer.

7.3.4 Sending header

The method RAK_sendHeader offers the possibility to simply send a command header, by specifying the command code and the length of the parameters.

It sends the 4 bytes header, one byte by one with the SPI. It also read the value send by the RAK411 module, if the module sends a ACK (acknowledge of the command) it returns the value of the ACK, in order to indicate that the module has received the command and is ready to receive the parameters.
7.3.5 Sending parameters

The **RAK\_sendParameters** method allows to send the command's structure, by just giving a pointer on the structure to send and the size of the structure.

In order to simplify this method, all the element of the structure are stored in the memory in the order to be send to the module. The following figure show the structure stored in the memory.

![Figure 31: Structure command on SRAM](image)

With this, the method can easily send all the bytes of the command parameters, one by one, on the SPI.

7.3.6 Send command states machine

As explained on chapter 7.3.1, when sending a command to the RAK411 module, the header must be sent first. When the module acknowledges the command, the data can be sent.

Depending of the previous commands sent, the module can take several milliseconds (between 50 and 200 ms) to acknowledge the command. In order to avoid blocking the whole process while waiting for the acknowledge, a states machine is used to send the command.

![Figure 32: Send command states machine](image)

This states machine is initialized and call as a function in the code.
It starts by sending the header of the command. If the module acknowledge it, it sends the data and go in Idle state and return true, to indicate that the data has been sent successfully.

If the module does not acknowledge, it goes in wait state. In the wait state, the state machine return false to indicate that he command has not be sent yet and the process can continue while this states machine is waiting. When 50 ms have passed, the machine try to send the header again.

### 7.3.7 Receive command states machine

The same principle is used in order to receive data from the Wi-Fi module. A state machine sends the read command header and read the data once the module acknowledges it.

![Diagram of Receive command states machine](image)

**Figure 33:** Receive command states machine

The states machine return true if the data have been read successfully, false if the modules has not sent the data yet.
7.4 Wi-Fi module states machine

The RAK411 states machine handles the communication between the MCU and the Wi-Fi module, using the states machines described in chapter 7.3.7 and 7.3.6 to send or receive the commands.

This states machine is represented on the following figure.

At the beginning of the program, or after a reset of the board, the states machine machine is in waiting for a reset module event. The state **Init** is another states machine. *See chapter 7.4.1.*

After the initialization the states machine enter in working mode, where it just wait to receive data from the module or the GPIB.

When the module has data to send, it means that the smartphone sent data or just connected/disconnected to the Wi-Fi module. The machine enters in **read** state, reads the data from the module and treats them according to the wireless transmission protocol defined in chapter 6.2.

If a response to the packet sent by the smartphone is required, the machine enters in **send** state, send the response and return in working mode.

If the GPIB states machine has finished to read or write on the bus, this machine enters in **send** state and send the GPIB information according to the wireless transmission protocol defined in chapter 6.2.
7.4.1 Init module states machine

This states machine is used to initialize and configure the RAK411 module.

![StateMachine](image)

Figure 35: RAK initialization states machine

init, get_version and u_scan commands are needed in order to initialize and read the boot and software information.

After the initialization, a the Wi-Fi AP is configured with set_psk and connect. The password is set to "1234567890", the SSID to "GPIB connector". The connect command will configure the module as an access point. With this, the smartphone is able to connect to the Wi-Fi module with the SSID and the password.

The TCP_server command configure the module as a TCP server and open a listening port. The port number chosen is 7 and the IP address of the module is configured to "192.164.1.10" with the command set_ip_static. This configuration will let the smartphone application connects easily on the TCP socket in order to communicates with the module.
7.4.2 Working state

Once the initialization and configuration of the module is done, the Wi-Fi module states machines enters in working mode. In this mode, the machine wait to receive data from the Wi-Fi module or from the GPIB bus.

Data from Wi-Fi :

When the MCU receives data from the Wi-Fi module, it means that the smartphone has performed a new action to the Wi-Fi module.

The MCU enters in the read state and treat the data received.

In the case where the smartphone performed a connection or a disconnection to the Wi-Fi AP or the TCP server listening port, the MCU will just blink the LED to indicate it to the user.

In the case smartphone sent a command to the MCU, the command will be treat following the wireless transmission protocol, defined on chapter 6.2.

Data from GPIB :

When the smartphone sends a write or read command to the MCU, the MCU activate the GPIB states machines in order to execute the command.

When the data have been written or read from the GPIB bus an event is pushed in the XF to inform the Wi-Fi states machine. When this arrive, the response paquet is sent to the smartphone.
7.5 Communication with GPIB bus

The communication between the MCU and the GPIB bus is made through the DS75160A and DS75161 drivers.

As described in the chapter 3.1.5, every byte transmitted on the bus undergoes a handshake sequence.

In order to develop a reliable method to communicate with the GPIB bus, states machine are used to handle the handshake sequence, as a source or an acceptor.

7.5.1 GPIB transmission

When the MCU sends data on the bus, it will act as the GPIB source. As shown on chapter 3.1.5, there is 4 different states for the GPIB source :

- SGNS : Source Generate State, the passive state, before and after a data transmission.
- SDYS : Source Delay State, when the source has asserted data on the lines and wait for acceptors to acknowledge.
- STRS : Source Transfer State, when the acceptors are ready to receive data, the source asserts that data are valid.
- SWNS : Source Wait for New cycle State, when data are accepted by all acceptors. the source releases the data valid line and the cycle resumes.

The diagram representing the states machine developed to handle GPIB transmission is presented on the following figure.

![GPIB send states machine](image)

The software states machine is developed by following this diagram.
7.5.2 GPIB reception

When the MCU receives data on the bus, it acts as a GPIB acceptor. As shown on chapter 3.1.5, there is 4 different states for the GPIB acceptor:

♦ ANRS : Acceptor Not Ready State, the acceptor is not ready to receive data.
♦ ACRS : Acceptor Ready State, when the source is asserting data on the line and the acceptor is ready to accept data.
♦ ACDS : Accept Data State, when the source as indicated that the data on the line are valid, the acceptor start to read them.
♦ AWNS : Acceptor Wait for New cycle State, when data have been saved

The diagram representing the states machine developed to handle GPIB reception is presented on the following figure.

![GPIB receive states machine](image)

Figure 37: GPIB receive states machine

The software states machine is developed by following this diagram.
7.6 GPIB states machine

The GPIB states machine handles the communication between the MCU and the GPIB bus, using the states machines described in chapter 7.5.1 and 7.5.2 to write or read on the bus.

This states machine is represented on the following figure.

![GPIB states machine diagram](image)

Figure 38: GPIB states machine

The state **None** indicates that no GPIB transmission or reception is required by the user. The states machine is inactive.

When the smartphone sends a command indicating that he wants to read or write on the bus, the states machine enters in the corresponding state. Once the data or command transmission/reception is finished, the states machine returns in None state and push an event in the XF to inform the **RAK Manager**.

The two methods **GPIB_send_bytes()** and **GPIB_receive_bytes()** allow the **RAK Manager** to configure the bytes to write on the bus or the number of bytes to read.
7.7 I/O Management

The method `IO_management()` handles the gesture of input and output of the MCU.

It receives an Event as parameter and performed the required actions.

**LEDs set** :

Different events can be pushed in the XF by the other tasks in order to set the LEDs states. This events will be treated by the I/O manager. *For example, when the event `evLedAOn` is treated by the I/O manager, it will turn the LED A on.*

**LED Blinking** :

The method `LED_program()` let the possibility for other tasks to program a LED blinking sequence, by specifying the number of blinks and the period.

**Wi-Fi module reset** :

The RAK state machines can push events in order to reset the RAK module. This events will be treated by the I/O manager, that will performs a hardware reset of the module.

**Long click** :

When the button is pressed, the I/O manager schedule a timer of 2 seconds. If the user releases the button before, the timer is unschedule, if not, a long click event is pushed in the XF.

**Software Reset** :

In order to provide the user the possibility to perform a software reset, the button is used. When the user performs a long click on the button, all the states machine a put in their initial states, and the Wi-Fi module is reset.
8 Android APP

Android is an open source mobile OS based on the linux kernel and developed by Google. It represents, in 2013, 78.4% of the worldwide smartphone sales to end users [18].

In order to develop Android applications, a full package is available in a single download [19]. This package contains everything required for APP development:

♦ Eclipse + ADT plugin
♦ Android SDK Tools
♦ Android Platform-tools
♦ The latest Android platform
♦ The latest Android system image for the emulator

This environment has been used during this project to develop the APP. The application has been developed and tested with a Samsung GT-S7562C with Android 4.1.2.

The application must offer the user a friendly interface, allowing to connect easily to the hardware board and send and receive data.

*The Android application code can be found in appendix [20].*

8.1 Wireless communication

The communication with the hardware circuit is made through the access point provided by the Wi-Fi module, as explained in chapter 7.4.

A Java class NetworkTask is used to handle the communication. This class extends AsyncTask, which allows to easily create a new Thread, required for the wireless communication and communicate with the UI Thread. This class provides the necessary functions to connect, send and receive data with the Wi-Fi module.
8.1.1 Connection

Before starting data exchange between the smartphone and the GPIB connector, the APP must open a connection between them.

The user can connect to the circuit board AP provided by the Wi-Fi module simply with the WLAN settings. The SSID and the password of the AP are configured by the MCU software See chapter 7.4.1 for the AP configuration.

Connection to socket:

As explained in chapter 7.4, the Wi-Fi module acts as a TCP server, and open a socket on a specific port number with a static IP address. With this information the android application is able to connect on the socket. When the user press on the connection button, the connection sequence starts.

The connection sequence is explained on the following figure.

![Connection sequence diagram](image)

After this sequence, the smartphone and the hardware circuit are connected and ready to communicate through a TCP socket.
8.1.2 Data transmission

A simple function allows to send data from the smartphone to the Wi-Fi module. The main thread simply call the function `sendDataToNetwork`, that will write the data on the socket if the socket is connected.

![Data transmission sequence diagram](image)

Figure 42: Data transmission sequence diagram

8.1.3 Data reception

Once the connection is made, the NetworkTask thread waits to receive data. When data are received on the socket, data are read and transmitted to the UI thread with the `onProgressUpdate` method, which will notify the main activity.

![Data reception sequence diagram](image)

Figure 43: Data reception sequence diagram
8.2 Instruments library

Since each electronic instruments manufacturer uses different GPIB commands, it is not possible to develop one simple application able to communicate with every instruments.

The idea behind this application is to provide the user the possibility to create, save and use different library containing the specific commands of an instruments.

When starting the APP, the user can choose start the **LibraryManager** activity in order create or load a library containing the commands for its instruments.

8.2.1 Library structure

The libraries are stored as `.glb` files. They contain one line for each different command. Each lines is composed as follow:

![Library line structure](image)

This simple structure allow to easily write or read the library and extract the values.

A class `GPIBcommand` is used to facilitate the saving and the extraction of command from the library.

![GPIBcommand class](image)

The `toString()` method returns a String that can be display for the user to show the name and the value contained in the function.

The `toFile()` method returns a String that respect the library structure presented above, in order to write easily the library `.glb` file.
8.2.2 Library Manager

The library manager allows the user to load or create new GPIB commands library for a specific instrument or manufacturer. Once the library is created or chosen from the phone storage, a preview of the available commands is displayed on the screen.

The following figures show the three Android activity that allow the user to manage the GPIB commands library.

![Figure 46: Library Manager](image1)
![Figure 47: Create library](image2)
![Figure 48: Load library](image3)

8.2.3 Load Library

The load library activity is a file browser. It will let the user browse through the internal SD card of the smartphone and choose a library to load.

The following figure shows the logic behind the file browser.

![Figure 49: Chose library Activity](image4)

This will ensure that the user chose a correct library file from the smartphone internal SD card, and return the path of the library.
8.2.4 Create new library

The create library activity allows the user to easily create a new .glb file containing the GPIB commands for the instrument he wants to access.

The directory where to save the library can be chosen with a file browser, the same as presented on the chapter 8.2.3.

The user interface offers the possibility to add a command in the library by specifying the name of the command and its value. All the commands are saved in a GPIBcommand list, and displayed on the listView to let the user visualize the commands already saved.

Once all the commands have been written and saved, the user can chose a name for the library and save the library in the chosen directory. All the commands are written in the .glb file from the GPIBcommands list.

8.2.5 Retrieving library commands

The libraries are saved as a .glb files with the structure explained in chapter 8.2.1. In order to use the commands of the library, the files must be read and the commands retrieved.

Each line of the library contains a commands. The name and the value of the command are extracted in a String array with the function split:

```java
1 line.split(",");
```

The GPIBcommand is recreated with this two attributes and put in a GPIBcommand list. When all lines have been read from the .glb file, the GPIBcommand list contains all the commands and can be use in the activity.

---

6See chapter 8.2.1 for GPIBcommand class
8.3 Main activity

The Main activity is the first launched by the Android APP.

By clicking on the option button, it gives the user the possibility to activate the Wi-Fi of the smartphone in order to connect on the Wi-Fi module or to launch the Library Manager described in chapter 8.2.2.

8.3.1 Retrieving library

When the user chose the library with the Library Manager and comes back on the Main activity, the path of the library is given.

The GPIB command list containing all the commands of the library created as explained on chapter 8.2.5. All the commands name are put in the commands spinner in order to let the user choose which commands he wants to write on the GPIB bus.

8.3.2 Application usage

The connect button will launch the connection to the socket provided by the Wi-Fi module, in order to communicate with the MCU.

The LED B and LED A buttons are used to send LED command to the MCU, as specified in the transmission protocol in chapter 6.2.3. By pressing the button, the desired LED will change its state.

When the MCU sends the LED response, the two images on the right will refresh to indicate the real states of the LEDs.

All the GPIB commands of the used library can be chosen from a spinner. The Send command button will send it to the MCU with a write command.

The user can type the data to write on the GPIB bus in the edit text. The button Send data will send them with a write command to the MCU.

The user can read some data on the GPIB bus. He can chose the number of bytes to read with the spinner. The button Receive data will send a read command with the specified number of bytes to read.
9 Test

In this section the tests realized to qualify the system are described.

9.1 Hardware

Once the PCB has been mounted, the power supply has been measured. With the microUSB connector plugged, the LDO output level is 3.3V, as expected. All the components are powered with the expected voltage.

9.1.1 MCU

The circuit is tested to ensure that the MCU can be programmed. The USB/SPI is connected on the PC and on the header of the circuit to allow the Xuan Wei programmer control platform to access the MCU.

![Figure 53: Circuit with USB/SPI connection to the PC](image1)

The platform indicates that it could successfully download the code in the MCU which means the MCU hardware is working.

![Figure 54: Circuit with USB/SPI connection to the PC](image2)

The platform indicates that it could successfully download the code in the MCU which means the MCU hardware is working.
9.1.2 RS-WC-201

As explained on the RS-WC-201 programmation manuel [21, p. 14], when the Wi-Fi module is configured in UART mode, on the power-up, it must send a welcome text on the UART port “Welcome to WiSeConnect”.

Measures on the UART pins of the Wi-Fi module on power-up show that the message is not sent by the RS-WC-201.

In the SPI mode, the module still not respond to the commands. These tests confirms that the module is not working, it is damaged or not properly soldered.

This is why another Wi-Fi module has been used for the rest of the project, the RAK411.

9.1.3 RAK411

The RAK411 evaluation board is used for the rest of the project. The header of the board allow to connect the SPI pins on the developed hardware through wires.

The hardware is modified to allow the connection properly with the EVB.

Figure 55: Evaluation board connection
The wires allows the power supply of the EVB, with 3.3V and GND. The MOSI, MISO, CS and SCK pins are connected with the corresponding on the MCU for the SPI communication. The SPI interruption, RESET and LINK are also connected.

### 9.2 SPI transmission

The SPI transmission is tested in order to know if the MCU can send commands to the RAK411 Wi-Fi module in order to configure it.

After the module boot-up and when the MCU has sent the init command, the module must return start information, in ASCII : "Welcome to RAK411".

The following figure shows a measure on the oscilloscope of MISO pin with the response of the module to the init command.

![Figure 57: Init response - Start information](image)
9.3 Configuration of RAK411

The different commands to configure the RAK module as an AP with a specified password and a static IP address and commands to open a TCP server are sent. The commands and the responses are measured on the oscilloscope and controlled. The responses show that the commands are successfully accepted by the module and the configuration is correct.

9.4 Smartphone and Wi-Fi module communication

9.4.1 Connection on AP

After the configuration of the module, the AP provided is visible by the smartphone.

The connection is established by entering the password, specified previously in the command from the MCU to the RAK411 module.

Connection is successful. Once established, the RAK411 module INT pin is active, which means it has data to send to the MCU.

Data are read and measured, as shown on the figure 59.

The 0xBC code indicates that the network is connected. In AP mode, this means that a station just connected to the access point. This prove that the connexion between the smartphone and the Wi-Fi module is made.
9.4.2 Connection on the socket

RAK411 has been configured as a TCP server and opened a listening port. The Android application creates a socket and connect to the port of the Wi-Fi module.

Connection is successful and the RAK411 module INT pin is active, which means it has data to send to the MCU.

The **0xC9** code indicates that socket is connected on the Wi-Fi module. This prove that the android application has opened a socket and is connected to the listening port of the Wi-Fi module, which allows to communicate with it.
9.4.3 Data transmission

A test application is developed to test the data transmission between the Android smartphone and the Wi-Fi module.

The two LED buttons allow to send LED commands to the Wi-Fi module, as specified on chapter 6.2.3.

When the user press on a button, the LED command is sent to the Wi-Fi module.

The data are received on the Wi-Fi module and transmitted to the MCU. The following figures show the data transmitted to the MCU.

The **0xC8** code indicates that data are received from smartphone. The three last byte correspond to the data transmitted from the smartphone. the **0x73** indicates that this is a LED command, as specified in chapter 6.2.3. The two last bytes of the command order to the MCU to turn off LEDA and turn on LEDB.

After receiving setting the LED with the specified states, the MCU acknowledges the LED command by sending a LED command response.
The **0xB6** code indicates that the MCU is sending data on the smartphone.

At the end of the command, the 4 last bytes are the data sent to the smartphone. **0x72** indicates that this is a response. **0x73** represents a LED command, which means that this is a response to a LED commands. The two last bytes indicates the actual states of the LEDs.

This response is received by the smartphone, and the actual states of the LEDs are displayed on the phone.

This test prove that the bidirectional communication between the smartphone and the MCU, via the RAK411 Wi-Fi module if fully functional. Data can be sent from both device.
9.5 Communication on GPIB bus

The entire GPIB protocol has not been implemented in the software yet, only the handshake sequence is handled. Therefore, the access to an instrument is not possible.

The handshake sequence can be tested by simulating the acceptor or the source.

9.5.1 Handshake source test

To test if the hardware can handle the handshake sequence as a source, the acceptor is simulated by controlling the NRFD and NDAC level. The DIO are measured to see if the byte is written properly by the MCU.

The following command is sent by the smartphone:

```
| WRITE | DATA | 2  | 0xA5 | 0xF0 |
```

This paquet indicates the MCU that it needs to write two bytes of data on the GPIB bus, 0xA5 and 0xF0.

The NRFD and NDAC pins are controlled to simulate the acceptor. The measure of the DIO prove that the two bytes are written successfully one after the other.

Once the writing is complete, the MCU sends a response to the smartphone, indicating that writing is successful. If the data are not written before the time limit, the MCU sends a response indicating that there is a GPIB error.
This test proves that the handshake sequence is handled as a source.

### 9.5.2 Handshake acceptor test

To test if the hardware can handle the handshake sequence as an acceptor, the source is simulated by controlling the DAV and DIO level. The value $0x80$ is put on the DIO.

The following command is sent by the smartphone:

```
READ 1
```

This paquet indicates the MCU that it needs to read one byte of data on the GPIB bus and return its value to the smartphone.

The DAV and DIO pins are controlled to simulate the source.

Once the reading is complete, the MCU sends a response to the smartphone, indicating that writing is successful and returning the value read. If the data are not read before the time limit, the MCU sends a response indicating that there is a GPIB error.
The value \(-128\) corresponds to the \(0x80\) manually put on the DIO.

This test proves that the handshake sequence is handled as an acceptor.
10 Conclusion

According to the chapter 2 Objectives, a conclusion describes the status of the hardware and the software developed during this project and the improvements for future works.

10.1 Hardware

A functional hardware prototype has been developed, providing a physical interface between a Smartphone and a GPIB bus.

The hardware allows communication with a smartphone via a Wi-Fi module and can access to a GPIB bus via GPIB transceivers.

The Wi-Fi module used for the design of the circuit board, the RS-WC-201, has been replaced by the RAK411, using its evaluation board. In order to develop a final product, the design must integrates the new module properly.

10.2 Software

The MCU software developed use an execution framework that allows usage of states machines and events, providing a reliable code, easy to improve.

The software is capable to initialize the Wi-Fi module, configure an access point and handle the communication with the smartphone application. A transmission protocol is implemented by the software and the smartphone application, allowing to send and receive different commands and responses.

The GPIB protocol is not fully implemented yet, and the communication has not been tested to access any instruments. But the software can handle the handshake sequence, as a source or an acceptor, allowing to read or write on the GPIB bus, via the transceivers.

10.3 Android application

The android application developed is able to establish a connection to the Wi-Fi module configured as TCP server. It can creates a socket and connects to the listening port of the module.

With this, the application is able to send data to the MCU via the Wi-Fi module, and can also receive and treat the data received from the MCU.

A wireless transmission protocol is implemented on the application and the MCU, allowing the smartphone application to access the GPIB bus via different commands (write data, write command, read).

A library manager is developed and available in the APP. It offers the user the possibility to create, save and use GPIB libraries, containing different GPIB commands, corresponding to a specific instrument. The usage of library makes the use of the application and the access to the instruments easier.

The user can browse to the phone to find existing library and use it, or can create it with an editor.
10.4 Improvements

10.4.1 PCB

Size:

The prototype was designed with different measure point and jumper and on one side only, in order to make the tests easier. The final product can be designed with components on both sides, and SMD GPIB transceivers, this will save a lot of place and allow to significantly reduce the size of the board. We can imagine to develop a product that can directly be plugged on the instruments.

Wi-Fi module:

Since the RS-WC-201 Wi-Fi module has been replaced by the RAK411 after the development of the PCB, a new circuit must be designed to integrate the RAK411.

10.4.2 MCU Software

The MCU software is does not totally implement the GPIB protocol. It can handles the handshake sequence allowing to read or write on the GPIB bus, but does not handle all the GPIB sequence.

Development must be pursue in order to properly access all kind of GPIB instruments.

10.4.3 Android APP

The Android application can be improved by making the user interface more friendly.

Different data displays can be added to the application, in order to visualize the data received from the GPIB bus. For example, if the data to read is the waveform of an oscilloscope, the result must be displayed on a window, not as a text or a simple value.
REFERENCES

[16] Nathan Quinteiro. MCU Software code - GPIB. Appendix M.
[17] Beijing Rakwireless Technology Co. Ltd. RAK411 - SPI Software PRM. Appendix H.
The timing diagram relates the electrical signals on the bus to the states of the source and acceptor handshakes. By looking at both, it may be easier to relate the hardware handshake to IEEE 488 state diagrams.

1. Initially, the source goes to the source generate state (SGNS). In SGNS the source is not asserting the data lines or data valid (DAV). In the passive state the data lines rise to a high level. The acceptors are in the acceptor not ready state (ANRS), with both not ready for data (NRFD) and not data accepted (NDAC) asserted.

2. The source asserts the data lines and enters the source delay state (SDYS). If this is the last data byte in a message, the source may also assert end or identify (EOI). The source waits for the data to settle on the data lines and for all acceptors to reach the acceptor ready state (ACRS).

3. Each acceptor releases its not ready for data (NRFD) line and moves to the acceptor ready state (ACRS). Any acceptor can delay the handshake by not releasing NRFD.

4. When the source sees NRFD high, it enters the source transfer state (STRS) by asserting data valid (DAV).

5. When the receiver(s) see that DAV is asserted, they enter the accept data state (ACDS). Each device then asserts NRFD since it is busy with the current data byte.
6. As the devices accept data they release NDAC to move from the ACDS to the acceptor wait for new cycle state (AWNS). All receivers must release the NDAC line before the source can move to the next state (SWNS).

7. When NDAC is high, the source wait for new cycle state (SWNS) is entered. In SWNS, the source releases DAV. The acceptors then enter their initial state (acceptor not ready state ANRS).

8. The source returns to its initial state (source generate SGNS) and the cycle resumes.
The bus consists of 24 wires, 16 of which are information transmission lines.

- Data bus - eight bidirectional data lines.
- Transfer bus - three data transfer control lines.
- Management bus - five interface management lines.

The eight remaining lines are ground and one of these may be designated as the cable shield ground.

The cable shield ground on your board is connected to digital ground through a jumper near the connector. This jumper may be removed to allow the shield ground to be connected to chassis ground at the rear panel connecting screw. The jumper can be also be removed to allow an instrument to supply the shield ground. In most applications the cable must be shielded to comply with FCC regulations for computing equipment. The cable connectors are designed to be stacked and cables come in various lengths (.5, 1, 2, and 4 meter lengths) to accommodate most system configurations.
Bus Line Definitions

Data Bus

DIO1 through DIO8 - bidirectional asynchronous data lines.

Management Bus

ATN (Attention) - a bus management line that indicates whether the current data is to be interpreted as data or a command. When asserted with EOI it indicates that a parallel poll is in process.

EOI (End or identify) - a bus management line that indicates the termination of a data transfer. When asserted with ATN, it indicates that a parallel poll is in process.

IFC (Interface Clear) - a bus management line asserted only by the system controller to take unconditional control of the bus. The bus is cleared to a quiescent state and all talkers and listeners are placed in an idle state.

REN (Remote enable) - a bus management line that allows instruments on the bus to be programmed by the active controller (as opposed to being programmed only through the instrument controls).

SRQ (Service request) - a bus management line used by a device to request service from the active controller.

Transfer Bus

DAV (Data Valid) - one of three handshake lines used to indicate availability and validity of information on the DIO lines.

NDAC (Not Data Accepted) - a handshake line used to indicate the acceptance of data by all devices.

NRFD (Not Ready For Data) - a handshake line used to indicate that all devices are not ready to accept data.
The **star** cabling topology minimizes worst-case transmission path lengths but concentrates the system capacitance at a single node.

The **linear** cabling topology produces longer path lengths but distributes the capacitive load. Combinations of star and linear cabling configurations are also acceptable.
DS75160A, DS75161A

DS75160A/DS75161A IEEE-488 GPIB Transceivers

Literature Number: SNOSBL5A
DS75160A/DS75161A
IEEE-488 GPIB Transceivers

General Description
This family of high-speed-Schottky 8-channel bi-directional transceivers is designed to interface TTL/MOS logic to the IEEE Standard 488-1978 General Purpose Interface Bus (GPIB). PNP inputs are used at all driver inputs for minimum loading, and hysteresis is provided at all receiver inputs for added noise margin. The IEEE-488 required bus termination is provided internally with an active turn-off feature which disconnects the termination from the bus when $V_{CC}$ is removed.

The General Purpose Interface Bus is comprised of 16 signal lines — 8 for data and 8 for interface management. The data lines are always implemented with DS75160A, and the management lines are either implemented with DS75160A in a single-controller system.

Features
- 8-channel bi-directional non-inverting transceivers
- Bi-directional control implemented with TRI-STATE® output design
- Meets IEEE Standard 488-1978
- High-speed Schottky design
- Low power consumption
- High impedance PNP inputs (drivers)
- 500 mV (typ) input hysteresis (receivers)
- On-chip bus terminators
- No bus loading when $V_{CC}$ is removed
- Pin selectable open collector mode on DS75160A driver outputs
- Accommodates multi-controller systems

Connection Diagrams

Order Number DS75161AN or DS75161AWM
See NS Package Number M20B or N20B

TRI-STATE® is a registered trademark of National Semiconductor Corporation.

© 2001 National Semiconductor Corporation
Absolute Maximum Ratings (Note 2)
If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

- Supply Voltage, VCC: 7.0V
- Input Voltage: 5.5V
- Storage Temperature Range: −65°C to +150°C
- Lead Temperature (Soldering, 4 sec.): 260°C
- Maximum Power Dissipation (Note 1) at 25°C: 1897 mW

Electrical Characteristics (Notes 3, 4)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIH</td>
<td>High-Level Input Voltage</td>
<td>VCC, Supply Voltage</td>
<td>4.75</td>
<td>5.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIL</td>
<td>Low-Level Input Voltage</td>
<td>TA, Ambient Temperature</td>
<td>0</td>
<td>70</td>
<td>ºC</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input Clamp Voltage</td>
<td>IOL, Output Low Current Bus</td>
<td>48</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VHY</td>
<td>Input Hysteresis</td>
<td>Terminal</td>
<td>0.3</td>
<td>0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>High-Level Terminal</td>
<td>IOL = 16 mA</td>
<td>4.8</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Low-Level Terminal</td>
<td>IOL = 16 mA</td>
<td>4.8</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIL</td>
<td>Low-Level SC Inputs TE, PE, DC,</td>
<td>IOL = 16 mA</td>
<td>4.8</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBS</td>
<td>Terminator Bias Voltage at Bus Port</td>
<td>I(bus)</td>
<td>2.5</td>
<td>3.0</td>
<td>3.7</td>
<td>V</td>
</tr>
<tr>
<td>ILOAD</td>
<td>Terminator Bus Loading Current</td>
<td>VCC = 0V, V(bus) = 0V to 2.5V</td>
<td>40</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOS</td>
<td>Short-Circuit Output Current</td>
<td>VCC = 0V, V(b) = 0V to 2.5V</td>
<td>40</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>Supply Current</td>
<td>VCC = 0V, V(b) = 0V to 2.5V</td>
<td>40</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIN</td>
<td>Bus-Port Capacitance</td>
<td>VCC = 0V, V(b) = 0V to 2.5V, f = 1 MHz</td>
<td>20</td>
<td>30</td>
<td>pF</td>
<td></td>
</tr>
</tbody>
</table>

Note 2: “Absolute Maximum Ratings” are those values beyond which the safety of the device cannot be guaranteed. They are not meant to imply that the device should be operated at these limits. The table of “Electrical Characteristics” provides conditions for actual device operation.

Note 3: Unless otherwise specified, min/max limits apply across the 0°C to +70°C temperature range and the 4.75V to 5.25V power supply range. All typical values are for TA = 25°C and VCC = 5.0V.

Note 4: All currents into device pins are shown as positive; all currents out of device pins are shown as negative; all voltages are referenced to ground, unless otherwise specified. All values shown as max or min are so classified on absolute value basis.

Note 5: Only one output at a time should be shorted.

Note 6: This characteristic does not apply to outputs on DS75161A that are open collector.
### Switching Characteristics (Note 7)

$V_{CC} = 5.0V \pm 5\%$, $T_A = 0°C$ to $70°C$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>From</th>
<th>To</th>
<th>Conditions</th>
<th>DS75160A</th>
<th>DS75161A</th>
<th>DS75162A</th>
<th>Units</th>
</tr>
</thead>
</table>
| $t_{PLH}$ | Propagation Delay Time, Low to High Level Output | Terminal | Bus | $V_L = 2.3V$  
$R_L = 38.3\Omega$ | 10 | 20 | 10 | 20 | 10 | 20 | ns |
| $t_{PHL}$ | Propagation Delay Time, High to Low Level Output | Bus | Terminal | $V_L = 5.0V$  
$R_L = 240\Omega$  
$C_L = 30 pF$  
Figure 1 | 14 | 20 | 14 | 20 | 14 | 20 | ns |
| $t_{PLH}$ | Propagation Delay Time, Low to High Level Output | Bus | Terminal | $V_L = 5.0V$  
$R_L = 240\Omega$  
$C_L = 30 pF$  
Figure 1 | 14 | 20 | 14 | 20 | 14 | 20 | ns |
| $t_{PZH}$ | Output Enable Time to High Level | TE, DC, or SC | (Note 8) | $V_I = 3.0V$  
$V_L = 0V$  
$C_L = 15 pF$  
Figure 1 | 19 | 32 | 23 | 40 | 23 | 40 | ns |
| $t_{PHZ}$ | Output Disable Time From High Level | TE, DC, or SC | (Note 9) | $R_L = 480\Omega$  
$C_L = 15 pF$  
Figure 1 | 15 | 22 | 15 | 25 | 15 | 25 | ns |
| $t_{PZL}$ | Output Enable Time to Low Level | Bus | Terminal | $V_L = 0V$  
$V_L = 2.3V$  
$R_L = 38.3\Omega$  
$C_L = 15 pF$  
Figure 1 | 24 | 35 | 28 | 48 | 28 | 48 | ns |
| $t_{PZL}$ | Output Disable Time From Low Level | (Note 8) | (Note 9) | $R_L = 280\Omega$  
$C_L = 15 pF$  
Figure 1 | 15 | 27 | 20 | 35 | 20 | 35 | ns |
| $t_{PZH}$ | Output Enable Time to High Level | TE, DC, or SC | Terminal | $V_I = 3.0V$  
$V_L = 0V$  
$R_L = 3 k\Omega$  
$C_L = 15 pF$  
Figure 1 | 17 | 33 | 18 | 40 | 18 | 40 | ns |
| $t_{PHZ}$ | Output Disable Time From High Level | TE, DC, or SC | (Note 8) | (Note 9) | $V_I = 0V$  
$V_L = 5V$  
$R_L = 480\Omega$  
$C_L = 15 pF$  
Figure 1 | 10 | 17 | NA | NA | NA | ns |

**Note 7:** Typical values are for $V_{CC} = 5.0V$ and $T_A = 25°C$ and are meant for reference only.

**Note 8:** Refer to Functional Truth Tables for control input definition.

**Note 9:** Test configuration should be connected to only one transceiver at a time due to the high current stress caused by the $V_I$ voltage source when the output connected to that input becomes active.
Switching Load Configurations

\[ V_C \text{ logic high} = 3.0V \]
\[ V_C \text{ logic low} = 0V \]
\[ ^*C_L \text{ includes jig and probe capacitance} \]

**FIGURE 1.**

Connection Diagrams

**FIGURE 2.**

Dual-In-Line Package

**Top View**

Functional Description

**DS75160A**

This device is an 8-channel bi-directional transceiver with one common direction control input, denoted TE. When used to implement the IEEE-488 bus, this device is connected to the eight data bus lines, designated DIO1–DIO8. The port connections to the bus lines have internal terminators, in accordance with the IEEE-488 Standard, that are deactivated when the device is powered down. This feature guarantees no bus loading when \( V_{CC} = 0V \). The bus port outputs also have a control mode that either enables or disables the active upper stage of the totem-pole configuration. When this control input, denoted PE, is in the high state, the bus outputs operate in the high-speed totem-pole mode. When PE is in the low state, the bus outputs operate as open collector outputs which are necessary for parallel polling.

**DS75161A**

This device is also an 8-channel bi-directional transceiver which is specifically configured to implement the eight management signal lines of the IEEE-488 bus. This device, paired with the DS75160A, forms the complete 16-line interface between the IEEE-488 bus and a single controller instrumentation system. In compliance with the system organization of the management signal lines, the SRQ, NDAC, and NRFD bus port outputs are open collector. In contrast to the DS75160A, these open collector outputs are a fixed configuration. The direction control is divided into three groups. The DAV, NDAC, and NRFD transceiver directions are controlled by the TE input. The ATN, SRQ, REN, and IFC transceiver directions are controlled by the DC input. The EOI transceiver direction is a function of both the TE and DC inputs, as well as the logic level present on the ATN channel. The port connections to the bus lines have internal terminators identical to the DS75160A.
## Functional Description (Continued)

### Table of Signal Line Abbreviations

<table>
<thead>
<tr>
<th>Signal Line Classification</th>
<th>Mnemonic</th>
<th>Definition</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Signals</td>
<td>DC</td>
<td>Direction Control</td>
<td>DS75161A</td>
</tr>
<tr>
<td></td>
<td>PE</td>
<td>Pull-Up Enable</td>
<td>DS75160A</td>
</tr>
<tr>
<td></td>
<td>TE</td>
<td>Talk Enable</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td>SC</td>
<td>System Controller</td>
<td></td>
</tr>
<tr>
<td>Data I/O Ports</td>
<td>B1–B8</td>
<td>Bus Side of Device</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D1–D8</td>
<td>Terminal Side of Device</td>
<td>DS75160A</td>
</tr>
<tr>
<td>Management Signals</td>
<td>ATN</td>
<td>Attention</td>
<td>DS75161A</td>
</tr>
<tr>
<td></td>
<td>DAV</td>
<td>Data Valid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EOI</td>
<td>End or Identify</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IFC</td>
<td>Interface Clear</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NDAC</td>
<td>Not Data Accepted</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NRFD</td>
<td>Not Ready for Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REN</td>
<td>Remote Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SRQ</td>
<td>Service Request</td>
<td></td>
</tr>
</tbody>
</table>

### Logic Diagrams

![Logic Diagram](DS75160A)

**Note 1:** Denotes driver

**Note 2:** Denotes receiver

**Note 3:** Driver and receiver outputs are totem-pole configurations

**Note 4:** The driver outputs of DS75160A can have their active pull-up disabled by switching the PE input (pin 11) to the logic low state. This mode configures the outputs as open collector.

---

[www.national.com](www.national.com)
### Functional Truth Tables

#### DS75160A

<table>
<thead>
<tr>
<th>Control Input Level</th>
<th>Data Transceivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>TE</td>
<td>PE</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td>X</td>
</tr>
</tbody>
</table>

#### DS75161A

<table>
<thead>
<tr>
<th>Control Input Level</th>
<th>Transceiver Signal Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>TE</td>
<td>DC</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
</tr>
<tr>
<td>L</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>H</td>
</tr>
<tr>
<td>X</td>
<td>L</td>
</tr>
</tbody>
</table>

H = High level input  
L = Low level input  
X = Don't care  
T = Transmit, i.e., signal outputted to bus  
R = Receive, i.e., signal outputted to terminal  
*The ATN signal level is sensed for internal multiplex control of EOI transmission direction logic.
LM1117-N/LM1117I 800mA Low-Dropout Linear Regulator
Check for Samples: LM1117-N, LM1117I

FEATURES
- Available in 1.8V, 2.5V, 2.85V, 3.3V, 5V, and Adjustable Versions
- Space Saving SOT-223 and WSON Packages
- Current Limiting and Thermal Protection
- Output Current 800mA
- Line Regulation 0.2% (Max)
- Load Regulation 0.4% (Max)
- Temperature Range
  - LM1117-N: 0°C to 125°C
  - LM1117I: −40°C to 125°C

APPLICATIONS
- 2.85V Model for SCSI-2 Active Termination
- Post Regulator for Switching DC/DC Converter
- High Efficiency Linear Regulators
- Battery Charger
- Battery Powered Instrumentation

DESCRIPTION
The LM1117-N is a series of low dropout voltage regulators with a dropout of 1.2V at 800mA of load current. It has the same pin-out as Texas Instruments’ industry standard LM317.

The LM1117-N is available in an adjustable version, which can set the output voltage from 1.25V to 13.8V with only two external resistors. In addition, it is also available in five fixed voltages, 1.8V, 2.5V, 2.85V, 3.3V, and 5V.

The LM1117-N offers current limiting and thermal shutdown. Its circuit includes a zener trimmed bandgap reference to assure output voltage accuracy to within ±1%.

The LM1117-N series is available in WSON, PFM, SOT-223, TO-220, and TO-263 DPAK packages. A minimum of 10 F tantalum capacitor is required at the output to improve the transient response and stability.

TYPICAL APPLICATION
Active Terminator for SCSI-2 Bus

Figure 1. Fixed Output Regulator

* Required if the regulator is located far from the power supply filter.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet. All trademarks are the property of their respective owners.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of the Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Copyright © 2000–2013, Texas Instruments Incorporated
These devices have limited built-in ESD protection. The leads should be shorted together or the device placed in conductive foam during storage or handling to prevent electrostatic damage to the MOS gates.

### ABSOLUTE MAXIMUM RATINGS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Input Voltage (VIN to GND)</td>
<td>20V</td>
</tr>
<tr>
<td>Power Dissipation</td>
<td>Internally Limited</td>
</tr>
<tr>
<td>Junction Temperature (T_J)</td>
<td>150°C</td>
</tr>
<tr>
<td>Storage Temperature Range</td>
<td>-65°C to 150°C</td>
</tr>
<tr>
<td>Lead Temperature TO-220 (T) Package</td>
<td>260°C, 10 sec</td>
</tr>
<tr>
<td>Lead Temperature SOT-223 (IMP) Package</td>
<td>260°C, 4 sec</td>
</tr>
<tr>
<td>ESD Tolerance</td>
<td>2000V</td>
</tr>
</tbody>
</table>

1. Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. Operating Ratings indicate conditions for which the device is intended to be functional, but specific performance is not ensured. For ensured specifications and the test conditions, see the Electrical Characteristics.
2. If Military/Aerospace specified devices are required, please contact the Texas Instruments Sales Office/Distributors for availability and specifications.
3. The maximum power dissipation is a function of T_J(max) - θ_JA, and θ_A. The maximum allowable power dissipation at any ambient temperature is P_D = (T_J(max) - T_A)/θ_JA. All numbers apply for packages soldered directly into a PC board.
4. For testing purposes, ESD was applied using human body model, 1.5kΩ in series with 100pF.

### OPERATING RATINGS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Voltage (VIN to GND)</td>
<td>15V</td>
</tr>
<tr>
<td>Junction Temperature Range (T_J) (2)</td>
<td></td>
</tr>
<tr>
<td>LM1117-N</td>
<td>0°C to 125°C</td>
</tr>
<tr>
<td>LM1117I</td>
<td>-40°C to 125°C</td>
</tr>
</tbody>
</table>

1. Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. Operating Ratings indicate conditions for which the device is intended to be functional, but specific performance is not ensured. For ensured specifications and the test conditions, see the Electrical Characteristics.
2. The maximum power dissipation is a function of T_J(max) - θ_JA, and θ_A. The maximum allowable power dissipation at any ambient temperature is P_D = (T_J(max) - T_A)/θ_JA. All numbers apply for packages soldered directly into a PC board.
LM1117-N ELECTRICAL CHARACTERISTICS

Typicals and limits appearing in normal type apply for $T_J = 25^\circ$C. Limits appearing in **Boldface** type apply over the entire junction temperature range for operation, 0°C to 125°C.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>Min(1)</th>
<th>Typ(2)</th>
<th>Max(1)</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{REF}$</td>
<td>Reference Voltage</td>
<td>LM1117-N-ADJ $I_{OUT} = 10mA$, $V_{IN-VOUT} = 2V$, $T_J = 25^\circ$C, $10mA \leq I_{OUT} \leq 800mA$, $1.4V \leq V_{IN-VOUT} \leq 10V$</td>
<td>1.238</td>
<td>1.250</td>
<td>1.262</td>
<td>V</td>
</tr>
<tr>
<td>$VOUT$</td>
<td>Output Voltage</td>
<td>LM1117-N-1.8 $I_{OUT} = 10mA$, $V_{IN} = 3.8V$, $T_J = 25^\circ$C, $0 \leq I_{OUT} \leq 800mA$, $3.2V \leq V_{IN} \leq 10V$</td>
<td>1.782</td>
<td>1.800</td>
<td>1.818</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10mA</td>
<td>$\leq$</td>
<td>$\leq$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-2.5 $I_{OUT} = 10mA$, $V_{IN} = 4.5V$, $T_J = 25^\circ$C, $0 \leq I_{OUT} \leq 800mA$, $3.9V \leq V_{IN} \leq 10V$</td>
<td>2.475</td>
<td>2.500</td>
<td>2.525</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10mA</td>
<td>$\leq$</td>
<td>$\leq$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-2.85 $I_{OUT} = 10mA$, $V_{IN} = 4.85V$, $T_J = 25^\circ$C, $0 \leq I_{OUT} \leq 800mA$, $4.2V \leq V_{IN} \leq 10V$</td>
<td>2.920</td>
<td>2.850</td>
<td>2.880</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10mA</td>
<td>$\leq$</td>
<td>$\leq$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-3.3 $I_{OUT} = 10mA$, $V_{IN} = 5V$, $T_J = 25^\circ$C, $0 \leq I_{OUT} \leq 800mA$, $4.75V \leq V_{IN} \leq 10V$</td>
<td>3.267</td>
<td>3.300</td>
<td>3.333</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10mA</td>
<td>$\leq$</td>
<td>$\leq$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-5.0 $I_{OUT} = 10mA$, $V_{IN} = 7V$, $T_J = 25^\circ$C, $0 \leq I_{OUT} \leq 800mA$, $6.5V \leq V_{IN} \leq 12V$</td>
<td>4.950</td>
<td>5.000</td>
<td>5.050</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10mA</td>
<td>$\leq$</td>
<td>$\leq$</td>
<td></td>
</tr>
<tr>
<td>$\Delta VOUT$</td>
<td>Line Regulation(3)</td>
<td>LM1117-N-ADJ $I_{OUT} = 10mA$, $1.5V \leq V_{IN-VOUT} \leq 13.75V$</td>
<td>0.035</td>
<td>0.2</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-1.8 $I_{OUT} = 0mA$, $3.2V \leq V_{IN} \leq 10V$</td>
<td>1</td>
<td>6</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-2.5 $I_{OUT} = 0mA$, $3.9V \leq V_{IN} \leq 10V$</td>
<td>1</td>
<td>6</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-2.85 $I_{OUT} = 0mA$, $4.2V \leq V_{IN} \leq 10V$</td>
<td>1</td>
<td>6</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-3.3 $I_{OUT} = 0mA$, $4.7V \leq V_{IN} \leq 15V$</td>
<td>1</td>
<td>6</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-5.0 $I_{OUT} = 0mA$, $6.5V \leq V_{IN} \leq 15V$</td>
<td>1</td>
<td>10</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td>$\Delta VOUT$</td>
<td>Load Regulation(5)</td>
<td>LM1117-N-ADJ $V_{IN-VOUT} = 3V$, $10 \leq I_{OUT} \leq 800mA$</td>
<td>0.2</td>
<td>0.4</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-1.8 $V_{IN} = 3.2V$, $0 \leq I_{OUT} \leq 800mA$</td>
<td>1</td>
<td>10</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-2.5 $V_{IN} = 3.9V$, $0 \leq I_{OUT} \leq 800mA$</td>
<td>1</td>
<td>10</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-2.85 $V_{IN} = 4.2V$, $0 \leq I_{OUT} \leq 800mA$</td>
<td>1</td>
<td>10</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-3.3 $V_{IN} = 4.7V$, $0 \leq I_{OUT} \leq 800mA$</td>
<td>1</td>
<td>10</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-5.0 $V_{IN} = 6.5V$, $0 \leq I_{OUT} \leq 800mA$</td>
<td>1</td>
<td>15</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td>$V_{IN-VOUT}$</td>
<td>Dropout Voltage(4)</td>
<td>$I_{OUT} = 100mA$</td>
<td>1.10</td>
<td>1.20</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$I_{OUT} = 500mA$</td>
<td>1.15</td>
<td>1.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$I_{OUT} = 800mA$</td>
<td>1.20</td>
<td>1.30</td>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

(1) All limits are ensured by testing or statistical analysis.
(2) Typical Values represent the most likely parametric normal.
(3) Load and line regulation are measured at constant junction room temperature.
(4) The dropout voltage is the input/output differential at which the circuit ceases to regulate against further reduction in input voltage. It is measured when the output voltage has dropped 100mV from the nominal value obtained at $V_{IN} = V_{OUT} + 1.5V$.
### LM1117-N ELECTRICAL CHARACTERISTICS (continued)

Typicals and limits appearing in normal type apply for $T_J = 25^\circ$C. Limits appearing in **Boldface** type apply over the entire junction temperature range for operation, $0^\circ$C to $125^\circ$C.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>Min$(1)$</th>
<th>Typ$(2)$</th>
<th>Max$(1)$</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{\text{LIMIT}}$</td>
<td>Current Limit</td>
<td>$V_{IN}-V_{OUT} = 5\text{V}$, $T_J = 25^\circ$C</td>
<td>800</td>
<td>1200</td>
<td>1500</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td>Minimum Load Current$(5)$</td>
<td>LM1117-N-ADJ, $V_{IN} = 15\text{V}$</td>
<td>1.7</td>
<td>5</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quiescent Current</td>
<td>LM1117-N-1.8, $V_{IN} \leq 15\text{V}$</td>
<td>5</td>
<td>10</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-2.5, $V_{IN} \leq 15\text{V}$</td>
<td>5</td>
<td>10</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-2.85, $V_{IN} \leq 10\text{V}$</td>
<td>5</td>
<td>10</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-3.3, $V_{IN} \leq 15\text{V}$</td>
<td>5</td>
<td>10</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117-N-5.0, $V_{IN} \leq 15\text{V}$</td>
<td>5</td>
<td>10</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thermal Regulation</td>
<td>$T_A = 25^\circ$C, 30ms Pulse</td>
<td>0.01</td>
<td>0.1</td>
<td>%/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ripple Regulation</td>
<td>$f_{\text{RIPPLE}} = 1\text{Hz}$, $V_{IN}-V_{OUT} = 3\text{V}$</td>
<td>60</td>
<td>75</td>
<td>dB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Adjust Pin Current</td>
<td>$I_{\text{OUT}} = 10\text{mA}$</td>
<td>60</td>
<td>120</td>
<td>μA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Adjust Pin Current Change</td>
<td>$10 \leq I_{\text{OUT}} \leq 800\text{mA}$, $1.4V \leq V_{IN}-V_{OUT} \leq 10\text{V}$</td>
<td>0.2</td>
<td>5</td>
<td>μA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Temperature Stability</td>
<td>$T_A = 125^\circ$C, 1000Hrs</td>
<td>0.5</td>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Long Term Stability</td>
<td>$T_A = 125^\circ$C, 1000Hrs</td>
<td>0.3</td>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RMS Output Noise</td>
<td>(% of $V_{OUT}$), 10Hz $\leq f \leq 10\text{kHz}$</td>
<td>0.003</td>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thermal Resistance Junction-to-Case</td>
<td>3-Lead SOT-223</td>
<td>15.0</td>
<td>°C/W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3-Lead TO-220</td>
<td>3.0</td>
<td>°C/W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3-Lead TO-263</td>
<td>10</td>
<td>°C/W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thermal Resistance Junction-to-Ambient (No air flow)</td>
<td>3-Lead SOT-223 (No heat sink)</td>
<td>136</td>
<td>°C/W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3-Lead TO-220 (No heat sink)</td>
<td>79</td>
<td>°C/W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3-Lead TO-263$(6)$ (No heat sink)</td>
<td>92</td>
<td>°C/W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3-Lead PFM</td>
<td>55</td>
<td>°C/W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8-Lead WSON$(7)$</td>
<td>40</td>
<td>°C/W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(5) The minimum output current required to maintain regulation.
(6) Minimum pad size of 0.038in$^2$.
(7) Thermal Performance for the WSON was obtained using JESD51-7 board with six vias and an ambient temperature of 22°C. For information about improved thermal performance and power dissipation for the WSON, refer to Application Note AN-1187 (SNOA401).

### LM1117I ELECTRICAL CHARACTERISTICS

Typicals and limits appearing in normal type apply for $T_J = 25^\circ$C. Limits appearing in **Boldface** type apply over the entire junction temperature range for operation, $-40^\circ$C to $125^\circ$C.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>Min$(1)$</th>
<th>Typ$(2)$</th>
<th>Max$(1)$</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{\text{REF}}$</td>
<td>Reference Voltage</td>
<td>LM1117I-ADJ, $I_{\text{OUT}} = 10\text{mA}$, $V_{\text{REF}}-V_{OUT} = 2\text{V}$, $T_J = 25^\circ$C</td>
<td>1.238</td>
<td>1.250</td>
<td>1.262</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$10\text{mA} \leq I_{\text{OUT}} \leq 800\text{mA}$, $1.4V \leq V_{\text{REF}}-V_{OUT} \leq 10\text{V}$</td>
<td>1.200</td>
<td>1.250</td>
<td>1.290</td>
<td>V</td>
</tr>
<tr>
<td>$V_{\text{OUT}}$</td>
<td>Output Voltage</td>
<td>LM1117I-3.3, $I_{\text{OUT}} = 10\text{mA}$, $V_{\text{IN}} = 5\text{V}$, $T_J = 25^\circ$C</td>
<td>3.267</td>
<td>3.300</td>
<td>3.333</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$0 \leq I_{\text{OUT}} \leq 800\text{mA}$, $4.75V \leq V_{\text{IN}} \leq 10\text{V}$</td>
<td>3.168</td>
<td>3.300</td>
<td>3.432</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LM1117I-5.0, $I_{\text{OUT}} = 10\text{mA}$, $V_{\text{IN}} = 7\text{V}$, $T_J = 25^\circ$C</td>
<td>4.950</td>
<td>5.000</td>
<td>5.050</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$0 \leq I_{\text{OUT}} \leq 800\text{mA}$, $6.5\text{V} \leq V_{\text{IN}} \leq 12\text{V}$</td>
<td>4.800</td>
<td>5.000</td>
<td>5.200</td>
<td>V</td>
</tr>
</tbody>
</table>

(1) All limits are ensured by testing or statistical analysis.
(2) Typical Values represent the most likely parametric normal.
The RS-WC-201 module is a complete IEEE 802.11b/g/n wireless device server that provides a wireless interface to any equipment with a UART, SPI or USB interface. The module integrates a MAC, baseband processor, RF transceiver with power amplifier; and all WLAN protocol and networking stack functionality in embedded firmware to make a fully self-contained 802.11n WLAN solution for a variety of applications. The module integrates an antenna and a U.FL connector for external antenna with an option to select either one of them.

Applications:
- Seamless Wi-Fi connectivity for Application Processors
- Industrial M2M communications
- Enterprise applications – Wireless Printers, Point of Sale Terminals
- Security Cameras and Surveillance Equipment
- Medical applications – Wireless Medical Instrumentation, Secure Patient Data Logging and Transfer
- Consumer applications – Wi-Fi connectivity to smart-phones and Tablet PCs using Wi-Fi Direct™ in devices such as Refrigerators, Washing Machines, Audio/Video Equipment, Digital Picture Frames etc.

Device Features:
- 802.11b/g and single stream 802.11n module
- Wi-Fi Direct™
- Access Point with WPA2-PSK security
- Client Mode with with WPA/WPA2-PSK and Enterprise Security
  - WPA/WPA2-Enterprise (EAP-TLS, EAP-FAST, EAP-TTLS, PEAP-MS-CHAP-V2)
- Integrated TCP, UDP, DNS client, ICMP, IGMP and DHCP Server
- HTTP Server and Client
- DNS Client
- 802.11d
- WPS (Wi-Fi Protected Setup) for ease of provisioning
- UART, SPI and USB interfaces
- Upgrade firmware wirelessly
- Integrated antenna, frequency reference and low-frequency clock
- Ultra low-power operation with power save modes
- Single supply – 3.1 to 3.6V operation
- Dimensions – 35mm X 22mm X 2.75mm
1 Detailed Feature List

1.1 Host Interfaces

- UART
  - AT Command Interface supported for configuration and module operation
  - Baud-rate of 115200 bps supported currently. Support for higher baud rates are planned in future firmware releases

- SPI
  - Standard 4-wire SPI
  - Operation up to a maximum clock speed of 12.5MHz

- USB 2.0
  - The USB interface in the module corresponds to the CDC-ACM class and presents itself as a USB Device to the Host USB.

1.2 WLAN

- MAC
  - IEEE 802.11b/g/n standard
  - Dynamic selection of data rate depending on the channel conditions
  - Hardware accelerated implementation of AES
  - Wi-Fi Direct, Access Point and Client modes

- Baseband Processing
  - Supports DSSS (1, 2 Mbps) and CCK (5.5, 11 Mbps) modes
  - Supports all OFDM data rates (6 Mbps to 54 Mbps in 802.11g and MCS0 to MCS7 in 802.11n mode)
  - Supports long, short, and HT preamble modes
  - High-performance multipath compensation in OFDM, DSSS, and CCK modes

- RF
  - 2.4 GHz transceiver and Power Amplifier with direct conversion architecture
  - Integrated frequency reference and antenna

1.3 Security

- WPA/WPA2-Enterprise (supported when the module is configured in Client mode).

1 This frequency depends on the external delays also.
1.4 Wi-Fi Direct™

The module supports Wi-Fi Direct™. Wi-Fi Direct enables two Wi-Fi devices to communicate directly, without an access point in between. RS-WC-201 can act as a Wi-Fi Direct Group Owner, so that Wi-Fi Direct Peer-to-Peer nodes can join and exchange data. The module can also connect as a Peer-to-Peer client to another Wi-Fi Direct Group Owner (GO) node. Legacy Wi-Fi (non Wi-Fi Direct) nodes can also connect to the module.

1.5 Access Point

The module has an embedded Access Point. It supports four clients when in Access Point mode, and also acts as a DHCP server. WPA2-PSK security is supported. All standard Wi-Fi devices like Laptops, Smart-phones and Tablet PCs can connect to the Access Point. The connected clients can go to power save during this mode.

1.6 HTTP Server

The module hosts an HTTP server. It provides memory space where HTML pages can be loaded. The module implements the server back end and responds to HTTP Get or HTTP Post requests from the remote terminal. The remote terminal can access the web pages through a standard web browser.

1.7 HTTP Client

The module can act as a HTTP client and can be configured to send HTTP GET and HTTP POST requests to a remote HTTP server, and also receive the responses and forward to the Host.

1.8 Software and Documentation

The following are provided with the Wi-Fi modules

- Programming Reference Manual that contains descriptions of all commands to control and operate the module
- Reference schematics for module’s integration with application board.
- Sample Host driver source code for SPI interface
- Sample applications and Evaluation Board User Guide for evaluating functionality of the module
2.2 Module Dimensions

Figure 2: Module Dimensions

Package Type: LGA
Module height: 2.75 +/- 0.05 mm

2.3 Pin Description

<table>
<thead>
<tr>
<th>Pin No</th>
<th>Pin Name</th>
<th>Direction</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
<td>Ground</td>
<td>Ground</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>GND</td>
<td>Ground</td>
<td>Ground</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NC</td>
<td>-</td>
<td>No connect</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>USB_VREGIN</td>
<td>Power</td>
<td>5V power supply for USB transceiver. No connect when USB not used.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>GND</td>
<td>-</td>
<td>Ground</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>VCC3.3</td>
<td>Power</td>
<td>3.3V Power Supply</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>VCC3.3</td>
<td>Power</td>
<td>3.3V Power Supply</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>VCC3.3</td>
<td>Power</td>
<td>3.3V Power Supply</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>JP0</td>
<td>-</td>
<td>Connect pull down of 1 kOhms</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>JNC</td>
<td>-</td>
<td>No connect</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>JP02</td>
<td>-</td>
<td>Connect pull down of 1 kOhms</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>JP01</td>
<td>-</td>
<td>Connect pull down of 1 kOhms</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>WF_HNDSHKE1</td>
<td>Input</td>
<td>3.3V I/O, 9mA Handshake signal for wireless firmware upgrade. Should be connected to a GPIO pin of the Host MCU</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>WF_HNDSHKE2</td>
<td>Output</td>
<td>3.3V I/O, 9mA Handshake signal for wireless firmware upgrade. Should be connected to a GPIO pin of the Host MCU</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>NC</td>
<td>-</td>
<td>No connect</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>NC</td>
<td>-</td>
<td>No connect</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>NC</td>
<td>-</td>
<td>No connect</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>GND</td>
<td>-</td>
<td>Ground</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>WLAN_ACTIVE</td>
<td>Output</td>
<td>LVCMOS, 2mA Used for BT Coexistence. It indicates with logic high that WLAN activity is in progress. When bw, BT device has the opportunity to transmit. Not used in current firmware, should be left open</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>BT_PRIORITY</td>
<td>Input</td>
<td>LVCMOS, 2mA Used to indicate through logic high that BT is transmitting high priority traffic. Not used in current firmware, should be grounded</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>NC</td>
<td>-</td>
<td>No connect</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>GND</td>
<td>-</td>
<td>Ground</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>USB_DP</td>
<td>Input</td>
<td>3.3V I/O, USB differential pin (Positive). Not used in current firmware, should be left open</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>USB_DM</td>
<td>Input</td>
<td>3.3V I/O, USB differential pin</td>
<td></td>
</tr>
</tbody>
</table>
### RS-WC-201 Datasheet

**Version 2.02**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>I/O Voltage</th>
<th>Current</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>PT_GPIO1</td>
<td>3.3V</td>
<td>2mA</td>
<td>Output, Pass through output pin controllable by Host software.</td>
</tr>
<tr>
<td>26</td>
<td>PT_GPIO2</td>
<td>3.3V</td>
<td>2mA</td>
<td>Output, Pass through output pin controllable by Host software.</td>
</tr>
<tr>
<td>27</td>
<td>TP1</td>
<td>-</td>
<td>-</td>
<td>Test point</td>
</tr>
<tr>
<td>28</td>
<td>TP2</td>
<td>-</td>
<td>-</td>
<td>Test point</td>
</tr>
<tr>
<td>29</td>
<td>TP3</td>
<td>-</td>
<td>-</td>
<td>Test point</td>
</tr>
<tr>
<td>30</td>
<td>TP4</td>
<td>-</td>
<td>-</td>
<td>Test point</td>
</tr>
<tr>
<td>31</td>
<td>TP5</td>
<td>-</td>
<td>-</td>
<td>Test point</td>
</tr>
<tr>
<td>32</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>33</td>
<td>GND</td>
<td>-</td>
<td>-</td>
<td>Ground</td>
</tr>
<tr>
<td>34</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No Connect</td>
</tr>
<tr>
<td>35</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>36</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>37</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>38</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>39</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>40</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>41</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>42</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>43</td>
<td>RESET_N</td>
<td>3.3V</td>
<td></td>
<td>Active low reset for the module</td>
</tr>
<tr>
<td>44</td>
<td>GND</td>
<td>-</td>
<td>-</td>
<td>Ground</td>
</tr>
<tr>
<td>45</td>
<td>MODE_SEL_2</td>
<td>3.3V</td>
<td></td>
<td>Interface selection Bit[2]</td>
</tr>
<tr>
<td>46</td>
<td>MODE_SEL_1</td>
<td>3.3V</td>
<td></td>
<td>Interface selection Bit[1]</td>
</tr>
<tr>
<td>47</td>
<td>MODE_SEL_0</td>
<td>3.3V</td>
<td></td>
<td>Interface selection Bit[0]</td>
</tr>
<tr>
<td>48</td>
<td>WIFI_ACTIVITY</td>
<td>3.3V</td>
<td></td>
<td>Wi-Fi Activity indicator. This pin is '0' when the module is transferring data</td>
</tr>
<tr>
<td>49</td>
<td>CARD_READY</td>
<td>3.3V</td>
<td></td>
<td>Card ready indicator. Logic '0' indicates successful boot-up of the module</td>
</tr>
<tr>
<td>50</td>
<td>GND</td>
<td>-</td>
<td>-</td>
<td>Ground</td>
</tr>
<tr>
<td>51</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>52</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>53</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>54</td>
<td>NC</td>
<td>-</td>
<td>-</td>
<td>No connect</td>
</tr>
<tr>
<td>55</td>
<td>ADC2</td>
<td>3.3V</td>
<td></td>
<td>Input, Analog input to internal ADC. Not used in current firmware, should be left open</td>
</tr>
<tr>
<td>56</td>
<td>GND</td>
<td>Ground</td>
<td>-</td>
<td>Ground</td>
</tr>
<tr>
<td>57</td>
<td>ADC1</td>
<td>3.3V</td>
<td></td>
<td>Input, Analog input to internal ADC. Not used in current firmware, should be left open</td>
</tr>
<tr>
<td>58</td>
<td>WAKEUP</td>
<td>3.3V</td>
<td></td>
<td>Input, The module wakes up from sleep if logic high is driven into this pin. Used only in SPI mode, should be left open in UART mode.</td>
</tr>
<tr>
<td>59</td>
<td>SPI_READY</td>
<td>3.3V</td>
<td></td>
<td>Output, Handshake signal used in SPI mode and connected to a GPIO pin of the Host MCU. In other modes, this pin can be left open.</td>
</tr>
<tr>
<td>60</td>
<td>UART_RTS</td>
<td>3.3V</td>
<td></td>
<td>Output, UART Request to Send. Not used in current firmware, should be left open</td>
</tr>
<tr>
<td>61</td>
<td>UART_CTS</td>
<td>3.3V</td>
<td></td>
<td>Input, UART Clear to Send. Not used in current firmware, should be left open</td>
</tr>
<tr>
<td>62</td>
<td>UART_TX</td>
<td>3.3V</td>
<td></td>
<td>Output, UART Transmit</td>
</tr>
<tr>
<td>63</td>
<td>UART_RX</td>
<td>3.3V</td>
<td></td>
<td>Input, UART Receive</td>
</tr>
<tr>
<td>64</td>
<td>INTR</td>
<td>3.3V</td>
<td></td>
<td>Output, Active high, level triggered interrupt. Used in SPI mode. The interrupt is raised by the module to indicate there is data to be read by the Host, or to indicate the module has woken up from sleep. In UART mode, it can be left open</td>
</tr>
<tr>
<td>65</td>
<td>SPI_CS</td>
<td>3.3V</td>
<td></td>
<td>Input, SPI slave select</td>
</tr>
<tr>
<td>66</td>
<td>SPI_CLK</td>
<td>3.3V</td>
<td></td>
<td>Input, SPI clock input</td>
</tr>
<tr>
<td>67</td>
<td>SPI_MISO</td>
<td>3.3V</td>
<td></td>
<td>Output, SPI data output</td>
</tr>
<tr>
<td>68</td>
<td>SPI_MOSI</td>
<td>3.3V</td>
<td></td>
<td>Input, SPI data input</td>
</tr>
<tr>
<td>69</td>
<td>GND</td>
<td>-</td>
<td>-</td>
<td>Ground</td>
</tr>
<tr>
<td>70</td>
<td>PA.EN</td>
<td>LVCMOS, 4mA</td>
<td></td>
<td>External PA enable. If an external Power Amplifier is used, this should be connected to the enable of the external power amplifier.</td>
</tr>
</tbody>
</table>
5.2.8 WPA Supplicant
The WPA supplicant is used to initiate the 802.1x/EAP authentication. It also plays a major part in performing the 4-way handshake to derive the PTK in WPA/WPA2-PSK modes.

5.3 Power Save
The RS-WC-201 module is an ultra low power Wi-Fi module. The Host can select either an always-on mode or power save mode for the module. In power save mode, the module powers off the Baseband, RF and also the Core Control Block during the sleep intervals. There are three power save modes supported in the module:

Power Mode 1: This mode is based on a configurable internal timer. The module can be made to wake-up at periodic intervals, based on the timer configured by the Host.

Power Mode 2: In this mode, the module can be woken up from sleep by the Host at any time.

Power Mode 3: In this mode, the module can be set to shut down mode by the Host using a software command. When woken up by the Host, it initiates a boot-up sequence as if a fresh power up has happened. Power consumption is lowest in this mode. More detailed description of the power save modes is available in the Programming Reference Manual.

Some representative numbers for power consumption in specific scenarios are described below:

<table>
<thead>
<tr>
<th>Power Save State</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep Sleep</td>
<td>2.3 mA</td>
<td>This is the state of the module when it is in sleep state in Power Mode 1 and 2.</td>
</tr>
<tr>
<td>Continuous Tx</td>
<td>370 mA</td>
<td>Module transmitting data continuously at 54 Mbps physical data rate and 17dBm RF power</td>
</tr>
<tr>
<td>Continuous Rx</td>
<td>226 mA</td>
<td>Module receiving data continuously</td>
</tr>
<tr>
<td>Shut down</td>
<td>110 µA</td>
<td>This is the state of the module when it is set to shut down mode in Power Mode 3</td>
</tr>
</tbody>
</table>

### Table 1: Pin Description

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>71 NC</td>
<td>PA. Not used in current firmware, should be left open</td>
</tr>
<tr>
<td>72 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>73 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>74 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>75 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>76 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>77 USB_ID</td>
<td>Input 3.3V I/O Device/Host mode sense (OTG Mode). No connect if USB OTG not used</td>
</tr>
<tr>
<td>78 GND</td>
<td>Ground</td>
</tr>
<tr>
<td>79 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>80 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>81 USB_LDO_FLAG</td>
<td>Input 3.3V I/O, 2mA Power distribution IC Health Monitor (OTG Mode). Not used in current firmware, should be left open</td>
</tr>
<tr>
<td>82 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>83 PD</td>
<td>Connect pull down of 4.7K Ohms</td>
</tr>
<tr>
<td>84 GND</td>
<td>Ground</td>
</tr>
<tr>
<td>85 USB_LDO_EN</td>
<td>Output 3.3V I/O, 2mA Power Distribution IC Enable (OTG Mode). Not used in current firmware, should be left open</td>
</tr>
<tr>
<td>86 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>87 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>88 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>89 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>90 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>91 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>92 NC</td>
<td>No connect</td>
</tr>
<tr>
<td>93 GND</td>
<td>Ground</td>
</tr>
<tr>
<td>94 GND</td>
<td>Ground</td>
</tr>
<tr>
<td>95 GND PAD</td>
<td>Thermal Ground Pad</td>
</tr>
</tbody>
</table>

### Table 9: Power Consumption
Appendix E - RS-WC-201 Module integration guide

RS-WC-201

Module Integration Guide

Version 2.9

May 2014

Redpine Signals, Inc.
2107 N. First Street, #680
San Jose, CA 95131.
Tel: (408) 748-3385
Fax: (408) 705-2019
Email: info@redpinesignals.com
Website: www.redpinesignals.com
## Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>Notes on USB Connections included. Wireless firmware upgrade feature available from firmware version 2.1.0.1.2.5 onwards.</td>
</tr>
<tr>
<td>2.8</td>
<td>PT_GPIO1, PT_GPIO2, WIFI_ACTIVITY, and CARDReady pins are available from 2.8V onwards, note added on Inrush current.</td>
</tr>
<tr>
<td>2.9</td>
<td>Changed note on BT_PRIORITY and WLAN_ACTIVE.</td>
</tr>
</tbody>
</table>
Table of Contents

RS-WC-201 ........................................................................................................ 1
1  RS-WC-201 Reference Schematics ................................................................. 6
   1.1 RS-WC-201 Bill Of Materials ................................................................. 7
   1.2 Reset Timing .......................................................................................... 10
2  RS-WC-201 PCB Landing Pattern ................................................................. 12
3  Circuit and Layout Guidelines ...................................................................... 13
4  u.FL Connector Recommendations .............................................................. 14
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Reference Schematics</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Reset Timing</td>
<td>11</td>
</tr>
<tr>
<td>Figure 3</td>
<td>PCB Landing Pattern for RS-WC-201</td>
<td>12</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Copper Etching Guidelines</td>
<td>13</td>
</tr>
<tr>
<td>Figure 5</td>
<td>uFL Connector and External Antenna</td>
<td>14</td>
</tr>
</tbody>
</table>
1 RS-WC-201 Reference Schematics

Figure 1: Reference Schematics
Notes on Pins:
The pins of the module are shown as having valid terminations in the above schematics to provide a reference for the designer. However, some pins are not used in the current firmware and can be left open/grounded as the case may be. Such pins are described below. A notification will be put in the document when a firmware using these pins is available.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLAN_ACTIVE</td>
<td>Available from firmware version 2.6.1.1.7.7. If not used this pin should be left open.</td>
</tr>
<tr>
<td>BT_PRIORITY</td>
<td>Available from firmware version 2.6.1.1.7.7. If not used this pin should be grounded through 1k ohm resistor.</td>
</tr>
<tr>
<td>ADC1, ADC2</td>
<td>Not used in current firmware, should be left open.</td>
</tr>
</tbody>
</table>

Based on the Host SPI configuration, during the boot-up of the Host MCU, the SPI Master interface in the Host could be coming up as GPIO pins. As a result, it may be needed to add a pull up on the SPI_CS and a pull down (SPI CPOL=0) on the SPI_CLK. The value of pull up/ pull down resistor should follow the recommendations as given on the HOST side.

1.1 RS-WC-201 Bill Of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Qty</th>
<th>Reference</th>
<th>Part Value</th>
<th>Description</th>
<th>Jedec</th>
<th>Mfg</th>
<th>Part No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td></td>
<td>CAPACITORS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>C2</td>
<td>4.7nF 250V</td>
<td>CER CHIP C 4.7nF 10% B(JB1) 0805 6.3V</td>
<td>0805</td>
<td>Murata</td>
<td>GRM219R72A472KA01B</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>C34</td>
<td>100pF</td>
<td>CER CHIP C 1nF +/-10% X7R 0402 50V</td>
<td>0402</td>
<td>Murata</td>
<td>GRM155R71H102KA01D</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>C36,C61</td>
<td>0.1uF</td>
<td>CER CHIP C 0.1u 10% X5R 0402 6.3V</td>
<td>0402</td>
<td>Murata</td>
<td>GRM155R61A104KA01D</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>C35</td>
<td>10uF</td>
<td>CER CHIP C 10U 20% X5R 0805 10V</td>
<td>0805</td>
<td>Murata</td>
<td>GRM218R61A106KE10L</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>C60</td>
<td>100uF</td>
<td>CAP TANT 100uF 6.3V 20% 1210</td>
<td>Case B</td>
<td>AVX</td>
<td>YCB107M006R0070</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>R13,R40,R41,R42,R47,R10, R107,R109,R112,R114</td>
<td>0Ω</td>
<td>CHIP RES 0.0R 5% 200PPM 0402 1/10W</td>
<td>0402</td>
<td>Panasonic</td>
<td>ERI-2GE0R00X</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>R26,R27,R78**</td>
<td>33Ω</td>
<td>CHIP RES 33R 5% 200PPM 0402 1/10W</td>
<td>0402</td>
<td>Panasonic</td>
<td>ERI-2GE330X</td>
</tr>
<tr>
<td>No.</td>
<td>Component</td>
<td>Value</td>
<td>Description</td>
<td>Manufacturer</td>
<td>Part Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>------------</td>
<td>--------</td>
<td>---------------------</td>
<td>--------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>R6, R7</td>
<td>820E</td>
<td>CHIP RES 820 5% 200PPM 0402 1/10W</td>
<td>Panasonic</td>
<td>ERJ-2GE821X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>R44</td>
<td>1K</td>
<td>RES 1K OHM 1/16W 5% 0402 SMD</td>
<td>Yageo</td>
<td>RCO402JR-071KL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>R43</td>
<td>2K</td>
<td>CHIP RES 2K 1% 100PPM 0402 1/10W</td>
<td>Panasonic</td>
<td>ERJ-2KFX2001X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>R76</td>
<td>4.7K</td>
<td>CHIP RES 4.7K 5% 200PPM 0402 1/10W</td>
<td>Panasonic</td>
<td>ERJ-2GE472X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>R108, R115, R116</td>
<td>10K</td>
<td>CHIP RES 10K 5% 200PPM 0402 1/10W</td>
<td>Panasonic</td>
<td>ERJ-2GE103X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>R14</td>
<td>20K</td>
<td>CHIP RES 20K 1% 100PPM 0402 1/16W</td>
<td>Yageo</td>
<td>RCO402FR-0720KL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>R12</td>
<td>100K</td>
<td>CHIP RES 100K 5% 200PPM 0402 1/10W</td>
<td>Panasonic</td>
<td>ERJ-2GE104X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>R25</td>
<td>1M</td>
<td>CHIP RES 1M 5% 200PPM 0402 1/10W</td>
<td>Panasonic</td>
<td>ERJ-2GE105X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>D5, D6</td>
<td>LED</td>
<td>Green LED</td>
<td>Lite-On Inc</td>
<td>LTST-C190KYT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>U4</td>
<td>FDC6329L</td>
<td>DC-DC Switch</td>
<td>Fairchild Semi.</td>
<td>FDC6329L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>U1</td>
<td>RS-WC-201</td>
<td></td>
<td>Redpine</td>
<td>RS-WC-201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>PB1</td>
<td>2 Pin headers</td>
<td>Burg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>SW5</td>
<td>SW DIP-4 Positions</td>
<td>E-Switch</td>
<td>KA041LGGT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>J5</td>
<td>Micro USB Connector</td>
<td>Molex</td>
<td>S-4819-0572</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>C71, C72, C73, C74, C75, C76</td>
<td>0.1uF</td>
<td>CER CHIP C 0.1uF 10% X5R 0402 6.3V</td>
<td>Murata</td>
<td>GRM15S861A04KAC01D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>R77, R103, R110</td>
<td>0E</td>
<td>CHIP RES 0.0R 5% 200PPM 0402 1/10W</td>
<td>Panasonic</td>
<td>ERJ-2GE0R00X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>R52</td>
<td>1.5K</td>
<td>RES 1.5K OHM 1/10W 5% 0402 SMD</td>
<td>Panasonic</td>
<td>ERJ-2GE152X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>R11, R102</td>
<td>10K</td>
<td>CHIP RES 10K 5% 200PPM 0402 1/10W</td>
<td>Panasonic</td>
<td>ERJ-2GE103X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>R35, R47</td>
<td>15K</td>
<td>CHIP RES 15K 5% 200PPM 1/10W 0402</td>
<td>Panasonic</td>
<td>ERJ-2GE153X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>FBS, FB7</td>
<td>BEAD</td>
<td>Ferrite Beads</td>
<td>Murata</td>
<td>BLM12PG221SN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>U18</td>
<td>UART Level Shifter</td>
<td>Texas Instr.</td>
<td>MAX3232CDBR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>U30</td>
<td>ADM708SAR</td>
<td>SOIC Analog device</td>
<td>ADM708SAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>P1</td>
<td>CONNECTOR DB9_FEMALE PCB Mount</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>SW1</td>
<td>DPDT</td>
<td>CTS Corp.</td>
<td>20A-221ST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>SW6</td>
<td>SW PUSHBUTTON</td>
<td>OMRON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. If the user wants to use only the UART interface, the SPI pins of the module should be treated accordingly:
   a. SPI_MOSI, SPI_CS and INTR should be pulled down with a 1k ohm resistor.
   b. SPI_MISO should be left open

2. Irrespective of whether the user uses the UART or SPI interface between the Host and the module, the UART interface should be brought out, preferably as a DB9 connector. This is required to do firmware upgrades on the module through a PC. Firmware upgrade using the SPI interface is not supported. Mechanism to upgrade firmware using the wireless interface is available from 2.1.0.1.2.5 onwards. The DB9 connector shown in the schematic above is provided for PC connectivity. GUI based application is provided along with the software package to upgrade the firmware. The DPDT switch shown can be used to select either the PC or the Host MCU’s UART interface to drive the UART interface of the module.

3. The Power Supply design should provide for a peak current of 400 mA. However, during certain phases of activity such as initialization after reset, a portion of the module get powered on internally. At that time, approximately 90μF of capacitance gets added to the 3.3V supply, causing a current inrush. To handle this, we have indicated the addition of C60 capacitance of value 220μF on the 3.3V supply line to the module.

4. Pins TP1, TP2, TP3, TP4, TP5 should be left unconnected. These are for internal debug. It is recommended that the developer brings these out as probe points on the PCB. For any custom debug, Redpine Signals may need to probe the pins.

5. For descriptions on individual pins, refer to the datasheet.

6. The mode select pins should be rigged up before the hard reset of the module, it cannot be changed at any time during the working of the module.

Note: Signal Integrity Guidelines for SPI interface: Glitches in the SPI clock may take the SPI interface out of synchronization. The quality and integrity of the clock line should be maintained. The following steps are recommended. This is not an exhaustive list of guidelines and depending on individual cases additional steps may be needed.

1. Avoid using cables to connect the Host platform with the module’s SPI interface. It is recommended to use a companion card with a rigid connector to the host, or directly solder the module on the Host MCU board. If a cable is used, minimize its length to as small as possible, preferably to within two inches.

2. If a cable is used, increase the number of ground connections between the Wi-Fi PCB and the MCU PCB

3. Add a series resistor into the clock line. Choice of value is mentioned in the Module Integration Guide.
4. If the SPI clock line is mapped to a programmable I/O on the MCU, configure that I/O to an output with as high a 
drive as is available.
5. Ensure that the Wi-Fi board’s reset input is mapped to a MCU controllable line, so that the system can recover 
through a hard reset.

Notes on USB:
1. USB_VREGIN Pin’s tolerance is min 2.7V and max 5.5V but typical value is 5V.
2. The USB ID pins are used in case of OTG mode only, it should be left open for Device mode. In the Current 
firmware we are ignoring the ID pin even though it is connected to the USB connector.
3. USB_LDO_FLAG is an input pin for Power Distribution IC Health Monitor (OTG Mode) and USB_LDO_EN is an 
output pin for Power Distribution IC Enable (OTG Mode). Since the module is used as USB device these pins can be 
left open.

1.2 Reset Timing

Following diagram shows the timing requirement for Reset input in two scenarios.
A. Power up
B. Giving reset during module operation

![Diagram showing reset timing](image-url)

**Figure 2: Reset Timing**
2 RS-WC-201 PCB Landing Pattern

Figure 3: PCB Landing Pattern for RS-WC-201
3 Circuit and Layout Guidelines

The following are guidelines for integrating the RS-WC-201 module into a wireless LAN solution.

1. The module has a ground pad of size 2.7mm x 2.7mm. An application’s layout must have a provision to include this.
   a. Provide a 2.7 mm X 2.7 mm Copper pad on the Topside of the application board. Please open the solder mask in this area so that the Cu is exposed.
   b. Provide a 2.7 mm X 2.7 mm or higher Copper pad on the bottom side of the application board. Please open the solder mask in this area so that the Cu is exposed.
   The dimension of the ground pad is shown as 2.5mm x 2.5mm in the datasheet. However, the landing is shown as expanded to 2.7mm X 2.7mm in the module integration guide for it to act as a better thermal ground pad.
   c. Provide at least 12 via’s to connect these pads to the Ground plane. It is recommended that the via’s should be at least 10 mil x 18 mil.

2. The copper on the application board should be etched out on all the layers on the designated area shown in figure for the Landing Pattern. This is required for the Antenna Keep out area. An example is shown below.

![Figure 4: Copper Etching Guidelines](image)

Redpine Signals, Inc. Proprietary and Confidential
4 u.FL Connector Recommendations

The RS-WC-201 module comes with an integrated chip antenna, and also a u.FL connector where an external antenna can be connected. A choice between the chip antenna and the external antenna can be made through a software command. The figures show the u.FL connector integrated on the module. The connector on the external antenna should be pushed down to fit into the u.FL connector connected to the module. Some reference part numbers for external antenna are given below. This is for general guidance only, as the choice of the antenna will depend on the application.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Part No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laird Technologies</td>
<td>0600-00040</td>
</tr>
<tr>
<td>Digi International</td>
<td>A24-HABUF-P5I</td>
</tr>
<tr>
<td>Connectblue</td>
<td>cB-ACC-27</td>
</tr>
<tr>
<td>Connectblue</td>
<td>cB-ACC-29</td>
</tr>
</tbody>
</table>

*Figure 5: uFL Connector and External Antenna*

The RS-WC-201 module comes with an integrated chip antenna, and also a u.FL connector where an external antenna can be connected. A choice between the chip antenna and the external antenna can be made through a software command. The figures show the u.FL connector integrated on the module. The connector on the external antenna should be pushed down to fit into the u.FL connector connected to the module. Some reference part numbers for external antenna are given below. This is for general guidance only, as the choice of the antenna will depend on the application.
RS-WC-201/301

Software Programming Reference Manual

Version 2.01

August 2012
3 RS-WC-201/301 in UART Mode

The following figure illustrates a general flow for operating a UART module.

![Flowchart](image)

**Figure 2: Firmware Upgrade and General Operation in UART modules**

RS-WC-201 and RS-WC-301 modules use the following UART interface configuration for communication:

- **Baud Rate:** 115200
- **Data bits:** 8
- **Parity:** None
- **Stop bits:** 2
- **Flow control:** None

### 3.1 Messages on Power-up

When the module is powered up, the following sequence is executed:

1. The module sends four 0xFC bytes and one 0xFF byte out in the UART interface
2. The module sends the message “Welcome to WiSeConnect” to the Host and then starts boot-up:
3. After boot-up is complete, the module issues a message
   READY\r\n
   0x52 0x45 0x44 0x41 0x44 0x49 0x40D 0x40A

4. The module is now ready to accept commands from the Host.

3.2 UART Commands

The Wi-Fi AT command set represents the frames that are sent from the Host to operate the RS-WC-201/301 module. The command set resembles the standard AT command interface used for modems.

AT commands start with “AT” and are terminated with a carriage return and a new line character. The AT command set for the RS-WC-201/301 module starts with "at+rsi_" followed by the name of the command and any relevant parameters. In some commands, a '?' character is used after the command to query certain values inside the module.

**APPENDIX A: Sample Flow of Commands in UART** captures sample flow of commands to configure the module in various functional modes.

---

**NOTE:** All commands are issued from Host to module as a sequence of ASCII characters. All return messages from module to Host consist of OK or ERROR strings, along with some return parameters. The return parameters may be ASCII or Hex on a case by case basis. ERROR is accompanied by <Error code>, returned in two’s complement, hex format.

---

**3.2.1 Set Operating Mode**

**Description**

This is the first command that should be sent from the Host. This command configures the module in different operating modes.

**Command**

`at+rsi_opermode`

**Usage**

`at+rsi_opermode=mode_val\r\n`
RAK411 SPI-WIFI Module

Specification V1.1

Beijing Rakwireless Technology Co.,Ltd

info@rakwireless.com

QQ:1395415717  QQ:1930154262

www.rakwireless.com
Content

RAK411 SPI-WIFI Module ........................................................................................................ 1

1 Overview .................................................................................................................................. 1
  1.1 Module Overview .................................................................................................................. 1
  1.2 Key Applications ................................................................................................................... 1
  1.3 Device Features ................................................................................................................... 1
  1.4 RAK411 System Diagram ...................................................................................................... 3

2 Functional Description ............................................................................................................. 4
  2.1 HW Interface ......................................................................................................................... 4
  2.2 Wireless Driver ..................................................................................................................... 4
  2.3 TCP/IP .................................................................................................................................... 4
  2.4 Power Consumption .............................................................................................................. 5

3 Hardware Introduction .............................................................................................................. 6
  3.1 Top and Bottom View ............................................................................................................ 6
  3.2 Pin Definition ........................................................................................................................ 6
  3.3 Design Reference .................................................................................................................. 7
  3.4 RAK411 PCB Mechanical Size ............................................................................................. 8
  3.5 Reflow Soldering Temperature Graph .................................................................................... 9
  3.6 Baking Instructions ............................................................................................................... 9

4 Electrical Characteristics .......................................................................................................... 10
  4.1 Absolute Maximum ............................................................................................................. 10
  4.2 Recommended Operating Parameters ................................................................................... 10
  4.3 RF Electrical Characteristics ............................................................................................... 11
  4.4 MCU Reset ........................................................................................................................... 12

5 Order Information ..................................................................................................................... 13
  5.1 Products ................................................................................................................................ 13
  5.2 Size ....................................................................................................................................... 13

6 Sales and Service ...................................................................................................................... 14

7 Revision History ........................................................................................................................ 15
1 Overview

1.1 Module Overview

RAK411 module is a Wi-Fi module that fully compliant with IEEE 802.11b/g/n wireless standards, with internally integrated TCP/IP protocol stack, supporting numerous protocols such as ARP, IP, ICMP, TCP, UDP, DHCP CLIENT, DHCP SERVER, DNS and other etc. It supports AP mode, Station mode and Ad-hoc and mode. Users can easily and quickly use it to networking and data transmission. Through SPI interface, the module’s maximum transmission rate is up to 2Mbps.

RAK411 supports storing parameters, and by the customer commands it determines whether to enable automatic networking to realize easy networking and reduce time for system to networking. The module has built-in WEB server, supporting wireless network parameters configuration, supporting wireless firmware upgrade. It also supports WPS and EasyConfig one-key networking, significantly reducing software development effort.

RAK411 has four power management modes, among which the minimum standby power consumption is 2uA, fully meet customer’s requirement for low power design.

1.2 Key Applications

- Portable products
- Home appliances and electrical appliances
- Industrial sensors
- Sales terminals
- Buildings automation
- Logistics and freight management
- Home security and automation
- Medical applications, such as patient monitoring, medical diagnostics
- Metering (stop timing, measuring instruments, meters, etc.)

1.3 Device Features

- Support IEEE 802.11b/g/n wireless standards
• Support four-wire SPI interface
• Support SPI Clock up to Maximum 16Mhz
• Minimalist hardware peripheral circuit design
• Support Station, Ad-hoc and AP modes
• Support DHCP SERVER / DHCPCLIENT
• Support OPEN, WEP, WPA-PSK, WPA2-PSK and WPS encryptions
• Support TCP, UDP protocols, with maximum 8 UDP/TCP connections
• Support webpage-based parameter configuration
• Support WPS and EasyConfig one-key to network connection
• Support parameter storage, customer orders loading after boot
• Support parameters store in Deep Sleep State, with connection time as fastest as 300ms
• Support wireless upgrade firmware
• On-board ceramic antenna or U.FL antenna connector
• Operating voltage: 3.3V
• 4 kinds power working modes, with minimum power consumption as1-2uA
• Small package size: 28.75mmX23.14mmX3.40mm
• FCC, RoHS and CE compliant
1.4 RAK411 System Diagram

Figure 1-1 RAK411 System Diagram
2 Functional Description

2.1 HW Interface

- Support clock 16MHz Maximum
- Interface actual throughput up to 2Mbps
- Four-wire SPI interface, support SPI data interrupt pin

2.2 Wireless Driver

- Compliant with IEEE 802.11b/g/n standards
- Support AP、STA 、AD-HOC Mode
- Support WEP, WPA/WPA2-PSK encryptions
- Fast networking, allowing module to be added to network within 1 sec after power up
- Support WPS and EasyConfig one-key to network connection
- Support wireless configuration and firmware upgrade

2.3 TCP/IP

- DHCP Client and Server features
- DNS Client and Server functions
- TCP Client, TCP Server, UDP Client, UDP Server and Multicast functions
- 8-way socket applications
2.4 Power Consumption

The module supports four power consumption modes:

- Fullspeed working mode, with approx 80ma average power consumption, peak current less than 200ma
- Power-saving mode, with approx 10ma average power consumption, peak current <200ma, DTIM = 100ms
- Deep sleep mode, with approx 5ma average power consumption, peak current <200ma, DTIM = 100ms
- Standby mode, with power consumption <2uA
3 Hardware Introduction

3.1 Top and Bottom View

![RAK411 Top View](image1)
![RAK411 Bottom View](image2)

3.2 Pin Definition

<table>
<thead>
<tr>
<th>Pin Serial No.</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
<td>Ground</td>
<td>connected to ground pad or the copper</td>
</tr>
<tr>
<td>2</td>
<td>VCC3V3</td>
<td>Power</td>
<td>3.3V power supply</td>
</tr>
<tr>
<td>18</td>
<td>LINK</td>
<td>O, PU</td>
<td>&quot;0&quot; - STA connected in AP mode, Connected to router in STA mode &quot;1&quot; - disconnected</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Remain disconnected when no use</td>
</tr>
<tr>
<td>19</td>
<td>RESET</td>
<td>I, PU</td>
<td>Module reset pin, low effective</td>
</tr>
<tr>
<td>27</td>
<td>SPI_INT</td>
<td>O</td>
<td>SPI mode interrupt pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“0” — idle level</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“1” — has data sent to host</td>
</tr>
<tr>
<td>31</td>
<td>SPI_MISO</td>
<td>O</td>
<td>SPI slave: data of SPI Master Input, Slave Output</td>
</tr>
<tr>
<td>32</td>
<td>SPI_MOSI</td>
<td>I</td>
<td>SPI slave: data of SPI Master Output, Slave Input</td>
</tr>
<tr>
<td></td>
<td>SPI_CLK</td>
<td>O</td>
<td>SPI slave: SPI clock input</td>
</tr>
<tr>
<td>---</td>
<td>---------</td>
<td>---</td>
<td>---------------------------</td>
</tr>
<tr>
<td>33</td>
<td>SPI_CLK</td>
<td>O</td>
<td>SPI slave: SPI clock input</td>
</tr>
<tr>
<td>34</td>
<td>SPI_CS</td>
<td>I</td>
<td>SPI slave: SPI chip select input</td>
</tr>
<tr>
<td>3,4,6,7,8,9,10,11,14,15,16,17,20,21,22,23,24,26,28,29,30</td>
<td>NC</td>
<td>NC</td>
<td>Remain disconnected when no use</td>
</tr>
</tbody>
</table>

Note:
1. I – input  O - output  PU – pulling up  PD - pulling down  NC - not connected
2. Pin in NC, remain disconnected

3.3 Design Reference

Figure 3-3 Module Typical Design Reference

Note: Upon SPI interface, the value of R3 depends on output resistance of host and PCB trace resistance, the default value is 33 Euro.
3.4 RAK411 PCB Mechanical Size

PCB TOP VIEW

Figure 3-4: PCB Mechanical Size (mm)
3.5 Reflow Soldering Temperature Graph

![Temperature Graph](image)

**Figure 3-5: Temperature Graph**

**Note:**

As shown in Figure 3-5, it is based on the SAC305 lead-free tin paste (3% silver, 0.5% copper). Alpha OM-338 lead-free cleaning-free flux is recommended. The Figure 6 is mainly used for guidance. The entire process time is subject to thermal pad number of assembly board and device Intensity.

3.6 Baking Instructions

The RAK411 module is very sensitive to water. Be cautious to baking the device. At ambient conditions, it is required that within 168 hours removed from the vacuum packaging, the module should be processed with the circuit board assembly by reflow soldering; Or stored in the environment with a relative humidity below 10%. If the condition is not satisfied, the RAK411 must be processed with a 9-hour baking in the environment of 125 °C before the reflow soldering.
4 Electrical Characteristics

4.1 Absolute Maximum

The following table shows the absolute maximum. Note that the module device may be damaged when exceeds the maximum. To avoid damages to the module and the device, please operate under specified conditions.

Table 4-1: Parameters and Value Range

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Symbols</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>External supply voltage</td>
<td>VCC3V3</td>
<td>-0.3~4.0</td>
<td>V</td>
</tr>
<tr>
<td>Maximum RF Input (Reference: 50Ω )</td>
<td>RF_in</td>
<td>+10</td>
<td>dBm</td>
</tr>
<tr>
<td>When voltage is 3.3V, IO Max voltage</td>
<td>3V3VinIOMax</td>
<td>VCC+0.3</td>
<td>V</td>
</tr>
<tr>
<td>When voltage is 3.3V, IO Min voltage</td>
<td>3V3VinIOMin</td>
<td>-0.3</td>
<td>V</td>
</tr>
<tr>
<td>Storage ambient temperature</td>
<td>T_store</td>
<td>-65~+135</td>
<td>°C</td>
</tr>
<tr>
<td>ESD resistance</td>
<td>ESD_HBM</td>
<td>2000</td>
<td>V</td>
</tr>
</tbody>
</table>

4.2 Recommended Operating Parameters

Table 4-2: Recommended Operating Parameter Range

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Symbols</th>
<th>Min Value</th>
<th>Typical Value</th>
<th>Max Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>External voltage</td>
<td>Vcc</td>
<td>3.14</td>
<td>3.3</td>
<td>3.46</td>
<td>V</td>
</tr>
<tr>
<td>Ambient temperature</td>
<td>T_ambient</td>
<td>-40</td>
<td>--</td>
<td>+85</td>
<td>°C</td>
</tr>
</tbody>
</table>
4.3 RF Electrical Characteristics

- RF Transmit Specifications

Table 4-3: Partial RF Transmit Specifications

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Conditions</th>
<th>Typical Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ftx</td>
<td>Frequency range</td>
<td>--</td>
<td>2.4</td>
<td>GHz</td>
</tr>
<tr>
<td>Pout</td>
<td>Output power</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>802.11b</td>
<td>1Mbps</td>
<td>17</td>
<td>dBm</td>
</tr>
<tr>
<td></td>
<td>802.11g</td>
<td>6Mbps</td>
<td>17</td>
<td>dBm</td>
</tr>
<tr>
<td></td>
<td>802.11n,HT20</td>
<td>MCS0</td>
<td>17</td>
<td>dBm</td>
</tr>
<tr>
<td></td>
<td>802.11g,EVM</td>
<td>54Mbps</td>
<td>14</td>
<td>dBm</td>
</tr>
<tr>
<td></td>
<td>802.11n,HT20EVM</td>
<td>MCS7</td>
<td>10</td>
<td>dBm</td>
</tr>
</tbody>
</table>

- RF Receiver Specifications

Table 4-4: Partial Receiver Specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test conditions</th>
<th>Typical Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>11b,1Mbps</td>
<td>-97</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td>11b,2Mbps</td>
<td>-92</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td>11b,5.5Mbps</td>
<td>-90</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td>11b,11Mbps</td>
<td>-88</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td>11g,9Mbps</td>
<td>-91</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td>11g,18Mbps</td>
<td>-87</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td>11g,36Mbps</td>
<td>-81</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td>11g,54Mbps</td>
<td>-75</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td>11n,MCS1,13Mbps</td>
<td>-89</td>
<td>dBm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11n,MCS3,26Mbps</td>
<td>-82</td>
<td>dBm</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>11n,MCS5,52Mbps</td>
<td>-75</td>
<td>dBm</td>
</tr>
<tr>
<td></td>
<td>11n,MCS7,65Mbps</td>
<td>-72</td>
<td>dBm</td>
</tr>
<tr>
<td>Maximum input signal</td>
<td>CH7</td>
<td>11g,54Mbps</td>
<td>10</td>
</tr>
<tr>
<td>Adjacent channel</td>
<td>6Mbps</td>
<td>37</td>
<td>dBc</td>
</tr>
<tr>
<td></td>
<td>54Mbps</td>
<td>21</td>
<td>dBc</td>
</tr>
<tr>
<td></td>
<td>MCS0</td>
<td>38</td>
<td>dBc</td>
</tr>
<tr>
<td></td>
<td>MCS7</td>
<td>20</td>
<td>dBc</td>
</tr>
</tbody>
</table>

4.4 MCU Reset

Figure 4-1 shows the MCU reset timing diagram and reset pulse length. When power on the module or an exception occurs, the module needs to be reset. RESET pin is internally pulled up, low input is effective.

3.3V IO Supply

MCU_RESET

![Diagram of MCU Reset Timing]

Figure 4-1: MCU Reset Timing

Table 4-5 shows the description of MCU reset parameters.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>typical (mS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RESET</td>
<td>MCU reset pulse length</td>
<td>&gt;10</td>
</tr>
</tbody>
</table>
5 Order Information

5.1 Products

Table 5-1: Product Models

<table>
<thead>
<tr>
<th>Product</th>
<th>Description</th>
<th>Packaging</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAK411A</td>
<td>SPI interface module, with on-board antenna</td>
<td>27pcs/tray</td>
</tr>
<tr>
<td>RAK411B</td>
<td>SPI interface module, with external antenna</td>
<td>27pcs/tray</td>
</tr>
</tbody>
</table>

5.2 Size

Packaging: Hard plastic pallets

Weight: <=3.00g/pcs

Table 5-2: Thickness (Height)

<table>
<thead>
<tr>
<th>RAK411</th>
<th>Thickness (Height)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without shield holder</td>
<td>2.85 ± 0.05mm</td>
</tr>
<tr>
<td>With shield holder</td>
<td>2.95 ± 0.05mm</td>
</tr>
<tr>
<td>With shield</td>
<td>3.30 ± 0.15mm</td>
</tr>
</tbody>
</table>

Note: In considering height design of the product, please consider your motherboard thickness error and product fit gap (recommended 0.10-0.15mm).
6 Sales and Service

Beijing

FAE mailbox: allan.jin@rakwireless.com

Tel: 010-82671601

Fax: 010-82678368

Address: Room 1108, Jin Yanlong Building, Xisanqi Desheng Gate

Haidian District, Beijing

Shanghai

FAE mailbox: steven.tang@rakwireless.com

Tel: 021-51113558

Fax: 021-51113220

Address: Room 306, Ran East Business Center 150 Building 1, No. 2161

Wanyuan Road, Minhang District, Shanghai

Shenzhen

FAE mailbox: vincent.wu@rakwireless.com

Tel: 0755-82271586

Fax: 0755-85236551

Address: Room 406, Tsinghua information comprehensive building,

Nanshan Science Park North District, Shenzhen
## 7 Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Modifications</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.0</td>
<td>Initial Draft</td>
<td>2014-02-08</td>
</tr>
<tr>
<td>V1.1</td>
<td>Optimized power consumption to 1-2uA</td>
<td>2014-02-17</td>
</tr>
</tbody>
</table>
1 Overview

RAK411 module is a Wi-Fi module that fully compliant with IEEE 802.11b/g/n wireless standards, with internally integrated TCP/IP protocol stack, supporting numerous protocols such as ARP, IP, ICMP, TCP, UDP, DHCP CLIENT, DHCP SERVER, DNS, and other ones. It supports AP mode, Station mode, and Ad-hoc mode. Users can easily and quickly use it to networking and data transmission. Through SPI interface, the module’s maximum transmission rate is up to 2Mbps.

RAK411 supports storing parameters, and by the customer commands it determines whether to enable automatic networking to realize easy networking and reduce time for system to networking. The module has built-in WEB server, supporting wireless network parameters configuration, supporting wireless firmware upgrade. It also supports WPS and EasyConfigure key networking, significantly reducing software development effort.

RAK411 has four power management modes, among which the minimum standby power consumption is 2uA, fully meet customer’s requirement for low-power design.

1.2 Device Features

- Support IEEE 802.11b/g/n wireless standards
- Support four-wire SPI Interface
- Support SPI Clock up to Maximum 16MHz
- Minimalist hardware peripheral circuit design
- Support statistic, Ad-hoc and AP modes
- Support DHCP SERVER / DHCP CLIENT
- Support OPEN, WEB, WPA-PSK, WPA2-PSK and WPS encryptions
- Support TCP, UDP protocols, with maximum 8 UDP/TCP connections
- Support webpage-based parameter configuration
- Support WPS and EasyConfigure key to network connection
- Support parameter storage, customer orders loading after boot
- Support parameters store in Deep Sleep State, with connection time as fast as 300ms
- Support wireless upgrade firmware
- On-board ceramic antenna or U.FL antenna connector
- Operating voltage: 3.3V
- 4 kinds power working modes, with minimum power consumption as 1-2uA
1.3 Key Applications

- Portable products
- Home appliances and electrical appliances
- Industrial sensors
- Sales terminals
- Buildings automation
- Logistics and freight management
- Home security and automation
- Medical applications, such as patient monitoring, medical diagnostics
- Metering (stop timing, measuring instruments, meters, etc.)

2 Functional Description

2.1 HW Interface

- Support clock 16MHz Maximum
- Interface actual throughput up to 2Mbps
- Four-wire SPI interface, support SPI data interrupt pin

2.2 Wireless Driver

- Compliant with IEEE 802.11b/g/n standards
- Support AP and STA Mode
- Support WEP, WPA/WPA2-PSK encryptions
- Fast networking, allowing module to be added to network within 1 sec after power up
- Support WPS and EasyConfig one-key to network connection
- Support wireless configuration and firmware upgrade

2.3 TCP/IP

- DHCP Client and Server features
- DNS Client and Server functions
- TCP Client, TCP Server, UDP Client, UDP Server and Multicast functions
- 8-way socket applications

2.4 Power Consumption

The module supports four power consumption modes:

- Fullspeed working mode, with approx 80mA average power consumption, peak current less than 200mA
- Power-saving mode, with approx 10mA average power consumption, peak current <200mA, DTIM = 100ms
- Deep sleep mode, with approx 5mA average power consumption, peak current <200mA, DTIM = 100ms
- Standby mode, with power consumption<2uA

3 SPI Interface

RAK411 communicates with the host through a standard 4-wire SPI interface. SPI clock supports maximum 16MHz and optional SPI-INT pin. The SPI interface configuration diagram is as follows:

3.1 Hardware Connection

3.2 SPI Timing Diagram

- CPOL = 0--------------------------------------------------------SOC is idle in low level voltage
- CPHA = 0--------------------------------------------------------Data is latched on clock rising edge, while transmitted on clock falling edge
- MSB_FIRST-------------------------------------------------------MSB is first sent 8 BIT
3.3 Interrupt Pin

RAK411 provides an optional INT pin, so that the host can quickly respond to module data requests via the INT pin. The normal INT pin is low level voltage, when there is abnormal data to be sent, the module pulls INT to high level. After the host receives the rising edge, the module can read the data by sending read frames directly. After reading a package, INT pin goes low. If there is data in the module needs to be sent to the host, the module will once again pullup INT pin.

3.4 SPI Frame Format

The SPI basic operations are divided into three categories: read status, write data, and read data.

3.4.1 Read Status

![SPI Frame Format](image)

**Description:**
Host sends read status command and wait for the module’s reply of ACK. If ACK is received the host continues to receive status bytes. Byte0: command OxAL, Byte1: fixed value Ox07, Byte2: invalid data, must be 0; Byte3: invalid data, must be 0.

3.4.2 Write Data

![Write Data](image)

**Description:**
Host sends read data command and wait for the module’s reply of ACK. If ACK is received, the module continues to receive LEN bytes, and then interpret the rest of DATA after LEN. Byte0: command OxAd, Byte2: fixed value Ox07; Byte3: invalid data, must be 0; Byte4: invalid data, must be 0.

3.4.3 Read Data

![Read Data](image)

**Description:**
Host sends read data command and wait for the module’s reply of ACK. If ACK is received, the module continues to receive LEN bytes, and then interpret the rest of DATA after LEN. Byte0: command OxAd, Byte2: fixed value Ox07; Byte3: invalid data, must be 0; Byte4: invalid data, must be 0.

3.4.4 Status Register

![Status Register](image)

**Description:**
Here is the byte of status read, the last two bits represent the data type of module feedback.

3.5 Boot

RAK411’s boot time is about 210ms. After a normal start, the host sends the initialization command, and the module will return start information:

**ASCII** —— Welcome to RAK411

**HEX** —— 57 65 6C 6F 6D 65 70 6F 62 41 44 31 31

3.6 Power Mode

RAK411 supports four power modes, shown in the following table:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Control Part</th>
<th>Wireless Part</th>
<th>Wake-up Type</th>
<th>Min Power Consumption (µA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal</td>
<td>Normal</td>
<td>No need</td>
<td>80mA</td>
</tr>
<tr>
<td>1</td>
<td>Sleep_Mode</td>
<td>Power_Save</td>
<td>No need</td>
<td>10mA</td>
</tr>
</tbody>
</table>
The basic operation of the process is as follows:

4. Command Encyclopedia

The SPI commands are divided into four parts: module management commands, network operation commands, socket operation commands, and parameters storing commands, shown as the following:

4.1.2 Version Checking

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rak_get_version</strong></td>
<td>Initialize module, read host information</td>
</tr>
<tr>
<td><strong>rak_setpwrmode</strong></td>
<td>Set module power mode</td>
</tr>
<tr>
<td><strong>rak_readstatus</strong></td>
<td>Read module status information</td>
</tr>
<tr>
<td><strong>rak_stop</strong></td>
<td>Reset module</td>
</tr>
</tbody>
</table>

4.1.3 Setting PowerMode

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rak_setpwrmode</strong></td>
<td>Set module power mode</td>
</tr>
<tr>
<td><strong>rak_stop</strong></td>
<td>Reset module</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4</td>
<td>0x01</td>
<td>Command code</td>
</tr>
</tbody>
</table>

Parameter:

It is used to set module power mode.

Return Value:

N/A
4.1.4 Reading Module Status

Command: rak_read_status
Syntax:
uint32_t cmd;

Parameter:
- **Cmd**: It is used to read the data of module status register.

Return Value:
- **<STATUS>**: 1 byte
  - 0x01: To read data
  - 0x02: Data cache is full

4.1.5 Reset

Command: rak_reset
Syntax:
uint32_t cmd;

Parameter:
- **Cmd**: It is used to reset the entire module via command.

Return Value: N/A

4.2 Network Operation Commands

4.2.1 Scanning Wireless Network

Command: rak_scan
Syntax:
typedef struct {
  uint32_t cmd;
  uint32_t channel;
  struct rak_scan_t;
} rak_scan_t;

Parameter:
- **cmd**: Command Code
- **channel**: Channel
- **rak_scan_t**: Scan data

Return Value:
- **<CODE>**: 2 bytes
  - 0xA3, 0x97, 0x00, 0x00: Command Code
  - 0x08, 0x00: 8 WLANs scanned
- **<STATUS>**: 2 bytes
  - 0x00, 0x00: Command successful
  - 0x00, 0x02: Command failed

4.2.2 Getting Scanned Information

Command: rak_getscan
Syntax:
typedef struct {
  char ssid[32];
  uint32_t scan_num;
  rak_getscan_t;
} rak_getscan_t;

Parameter:
- **char ssid[32]**: Ssid Name
- **uint32_t scan_num**: SpecifiedSSID, scan all channels if value is 0

Return Value:
- **<CODE>**: 2 bytes
  - 0xA3, 0x97, 0x00, 0x00: Response Code
- **<AP_NUM>**
- **<SSID>**
- **<SEC_MODE>**
- **<RSSI>**
- **<CHANNEL>**
- **<BSSID>**
- **<SSID_LEN>**
- **<CODE>**: 2 bytes
  - 0xA4, 0x98: Response Code
- **<STATUS>**: 2 bytes
  - 0x00, 0x00: Command successful
  - 0x00, 0x02: Command failed

4.2.3 Setting Password

Command: rak_set_psk
Syntax:
typedef struct {
  uint32_t cmd;
} rak_set_psk_t;

Parameter:
- **cmd**: Command Code

Return Value:
- **<CODE>**: 2 bytes
  - 0xA4, 0x97, 0x00, 0x00: Response Code
- **<STATUS>**: 2 bytes
  - 0x00, 0x00: Command successful
  - 0x00, 0x02: Command failed

4.2.4 Setting Channel

Command: rak_set_channel
Syntax:
typedef struct {
  uint32_t cmd;
  uint32_t channel;
  rak_channel_t;
} rak_channel_t;

Parameter:
- **cmd**: Command Code
- **channel**: Channel

Return Value:
- **<CODE>**: 2 bytes
  - 0xA3, 0x97, 0x00, 0x00: Response Code
- **<STATUS>**: 2 bytes
  - 0x00, 0x00: Command successful
  - 0x00, 0x02: Command failed
4.2.5 Connecting Wireless Network

Command: 
rak_connect
Syntax:
typedef struct {
    uint32_t cmd;
    uint32_t mode;
} rak_connect_t;

Parameter:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CODE&gt;</td>
<td>2</td>
<td>0xA6,0x00</td>
<td>Response Code</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1</td>
<td>0</td>
<td>Command successful</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td></td>
<td>Command failed</td>
</tr>
</tbody>
</table>

4.2.6 Configuring Static IP Address

Command:  
rak_set_ipstatic
Syntax:
typedef struct {
    uint32_t cmd;
    uint32_t ssid;
    uint32_t mask;
} rak_ipstatic_t;

Parameter:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CODE&gt;</td>
<td>4</td>
<td>0xA6,0x97,0x00,0x00</td>
<td>Command Code</td>
</tr>
<tr>
<td>ssid</td>
<td>32</td>
<td>ASCII</td>
<td>SSID</td>
</tr>
<tr>
<td>mask</td>
<td>32</td>
<td></td>
<td>mask</td>
</tr>
</tbody>
</table>

Parameter:
This command is used to assign static IP address for module.  
Return Value:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CODE&gt;</td>
<td>2</td>
<td>0xA6,0x00</td>
<td>Response Code</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1</td>
<td>0</td>
<td>Command successful</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td></td>
<td>Command failed</td>
</tr>
</tbody>
</table>

4.2.7 Setting DHCP Mode

Command:  
rak_setconfig_dhcp
Syntax:
typedef struct {
    uint32_t cmd;
    uint32_t mode;
} rak_config_dhcp_t;

Parameter:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>4</td>
<td>0xA6,0x97,0x00,0x00</td>
<td>Command Code</td>
</tr>
<tr>
<td>mode</td>
<td>4</td>
<td>0=client, 1=dhcp server</td>
<td>DHCP CLIENT or DHCP SERVER</td>
</tr>
<tr>
<td>dnssvr1</td>
<td>4</td>
<td>IP address</td>
<td>DNS server 1</td>
</tr>
<tr>
<td>dnssvr2</td>
<td>4</td>
<td>IP address</td>
<td>DNS server 2</td>
</tr>
</tbody>
</table>

Parameter:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CODE&gt;</td>
<td>4</td>
<td>0xA6,0x97,0x00,0x00</td>
<td>Command Code</td>
</tr>
</tbody>
</table>

Parameter:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>4</td>
<td>0xA6,0x97,0x00,0x00</td>
<td>Command Code</td>
</tr>
</tbody>
</table>

4.2.8 Connecting Network by Easyconfig

Command:  
rak_easy_config
Syntax:  
uint32_t cmd;

Parameter:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>4</td>
<td>0xC0,0xA8,0x07,0x01</td>
<td>Address</td>
</tr>
<tr>
<td>mask</td>
<td>4</td>
<td>IP address</td>
<td>Gateway</td>
</tr>
<tr>
<td>gw</td>
<td>4</td>
<td>IP address</td>
<td>DNS server 1</td>
</tr>
<tr>
<td>dns1</td>
<td>4</td>
<td>IP address</td>
<td>DNS server 2</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1</td>
<td>0</td>
<td>Command successful</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td></td>
<td>Command failed</td>
</tr>
</tbody>
</table>

4.2.9 Connecting Network by WPS

Command:  
rak_wps
Syntax:  
uint32_t cmd;

Parameter:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>4</td>
<td>0xC0,0xA8,0x07,0x01</td>
<td>Address</td>
</tr>
<tr>
<td>mask</td>
<td>4</td>
<td>IP address</td>
<td>Gateway</td>
</tr>
<tr>
<td>gw</td>
<td>4</td>
<td>IP address</td>
<td>DNS server 1</td>
</tr>
<tr>
<td>dns1</td>
<td>4</td>
<td>IP address</td>
<td>DNS server 2</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1</td>
<td>0</td>
<td>Command successful</td>
</tr>
<tr>
<td></td>
<td>-2</td>
<td></td>
<td>Command failed</td>
</tr>
</tbody>
</table>

This command initiates module EasyConfig function, along with APP software in the phone, enabling module automatically to be added in the specified network. After networking successfully, module automatically saves the parameters via rak_storeconfig command.
### 4.2.11 Querying module IP information

**Command:** `rak_ipconfig_query`

**Syntax:** `uint32_t cmd;`

**Parameter:**
- **Cmd**: 4 bytes, Value: 0xA0,0x00,0x00,0x00
  - Description: Command Code

**Parameter:**
- **Value**: It is used to get module IP information, including MAC address, IP address, subnet mask, gateway, and DNS server.

**Return Value:**
- **Bytes**
- **Value**
- **Description**
  - `<CODE>`: 2 bytes, Value: 0x01,0x00, Response Code
  - `<STATUS>`: 2 bytes, Value: -90, Current signal strength (dBm)

### 4.2.12 Getting Network Signal Strength

**Command:** `rak_get_rssi`

**Syntax:** `uint32_t cmd;`

**Parameter:**
- **Cmd**: 4 bytes, Value: 0xA0,0x00,0x00,0x00
  - Description: Command Code

### 4.2.14 PING

**Command:** `rak_ping`

**Syntax:**

```c
typedef struct {
    uint32_t cmd;
    uint32_t hostaddr;
    uint32_t count;
    uint16_t size;
} rak_ping_t;
```

**Parameter:**
- **Cmd**: 4 bytes, Value: 0xA0,0x00,0x00,0x00
  - Description: Command Code
- **hostaddr**: 4 bytes, Value: IP address of the host to ping
- **count**: 2 bytes, Value: Number of packets
- **size**: 2 bytes, Value: Packet size, maximum size is 1400 bytes

**Parameter:**
- **Value**: It is used to run the ping command.

**Return Value:**
- **Bytes**
- **Value**
- **Description**
  - `<CODE>`: 2 bytes, Value: 0x00,0x00, Response Code
  - `<STATUS>`: 1 byte, Value: -2, Cannot access host

### 4.2.15 AP Network Advanced Settings

**Command:** `rak_apconfig`  

**Syntax:**

```c
typedef struct {
    uint32_t cmd;
    uint8_t hidden;
    uint8_t countryCode[2];
} rak_apconfig_t;
```

**Parameter:**
- **Cmd**: 4 bytes, Value: 0xA0,0x00,0x00,0x00
  - Description: Command Code
- **hidden**: 1 byte, Value: Hidden network setting
- **countryCode**: 2 bytes, Value: Country code

**Parameter:**
- **Value**: It is used to set the AP network advanced settings, including hidden network, country code, etc.
### 4.2.16 Setting Network Listening Intervals

**Command:** rak_set_listen  
**Syntax:**  
ytypedefstruct  
{  
uint32_t cmd;  
uint16_t time;  
uint16_t & local_port;  
}rak_client_t;  
**Parameter:**  
- cmd:  
  - parameter type:  
  - value: 0xA8,0x00,0x00  
  - description: Command Code  
- time:  
  - parameter type:  
  - value: 20–1000  
  - description: Need to refer to the wireless router settings for specific parameters

**Return Value:**  
- Parameter Bytes Value Description  
  - <CODE>  
    - parameter type:  
    - value: 0xA8,0x00  
    - description: Response Code  
  - <STATUS>  
    - parameter type:  
    - value: 1  
    - description: Command successful  
    - value: -2  
    - description: Command failed

**Module:**  
- It is used to set module beacon interval in Station mode.  
  - Note:  
    - In power saving mode, reducing power consumption can be realized by increasing parameter values, but by this way it may cause delay in receiving wireless data.

**Return Value:**  
- Parameter Bytes Value Description  
  - <CODE>  
    - parameter type:  
    - value: 0xB3,0x00  
    - description: Response Code  
  - <STATUS>  
    - parameter type:  
    - value: 1  
    - description: Command successful  
    - value: -2  
    - description: Command failed

### 4.2.17 Disconnecting Current Wireless Network

**Command:** rak_disconnect  
**Syntax:**  
uint32_t cmd;  
**Parameter:**  
- cmd:  
  - parameter type:  
  - value: 0xB7,0x00,0x00,0x00  
  - description: Command Code

**Return Value:**  
- Parameter Bytes Value Description  
  - <CODE>  
    - parameter type:  
    - value: 0xB4,0x00  
    - description: Response Code  
  - <STATUS>  
    - parameter type:  
    - value: 1  
    - description: Command successful  
    - value: -2  
    - description: Current network is disconnected

### 4.3 Socket Operation Commands

#### 4.3.1 TCP Server

**Command:** rak_tcpp_server  
**Syntax:**  
typedefstruct  
{  
uint32_t cmd;  
uint16_t dest_port;  
uint16_t local_port;  
uint16_t & local_port;  
}rak_client_t;  
**Parameter:**  
- cmd:  
  - parameter type:  
  - value: 0xB3,0x00,0x00,0x00  
  - description: Command Code  
- dest_port:  
  - parameter type:  
  - value: 1-65535  
  - description: Target port number  
- local_port:  
  - parameter type:  
  - value: 1-65535  
  - description: Local port number

**Return Value:**  
- Parameter Bytes Value Description  
  - <CODE>  
    - parameter type:  
    - value: 0xB3,0x00  
    - description: Response Code  
  - <SOCKET_FLAG>  
    - parameter type:  
    - value: 2  
    - description: Socket error
  - <STATUS>  
    - parameter type:  
    - value: 0  
    - description: Command successful  
    - value: -2  
    - description: Failed to create  
    - value: -3  
    - description: Failed to bind  
    - value: -4  
    - description: Target port connection error

#### 4.3.2 TCP Client

**Command:** rak_tcp_client  
**Syntax:**  
typedefstruct  
{  
uint32_t cmd;  
uint16_t dest_addr;  
uint16_t dest_port;  
uint16_t local_port;  
uint16_t & local_port;  
}rak_client_t;  
**Parameter:**  
- cmd:  
  - parameter type:  
  - value: 0xB3,0x00,0x00,0x00  
  - description: Command Code

**Return Value:**  
- Parameter Bytes Value Description  
  - <CODE>  
    - parameter type:  
    - value: 0xB1,0x00  
    - description: Response Code  
  - <SOCKET_FLAG>  
    - parameter type:  
    - value: 2  
    - description: Socket error
  - <STATUS>  
    - parameter type:  
    - value: 0  
    - description: Command successful  
    - value: -2  
    - description: Command failed

#### 4.3.3 UDP Client

**Command:** rak_udp_client  
**Syntax:**  
typedefstruct  
{  
uint32_t cmd;  
uint16_t dest_addr;  
uint16_t dest_port;  
uint16_t local_port;  
uint16_t & local_port;  
}rak_client_t;  
**Parameter:**  
- cmd:  
  - parameter type:  
  - value: 0xB3,0x00,0x00,0x00  
  - description: Command Code

**Return Value:**  
- Parameter Bytes Value Description  
  - <CODE>  
    - parameter type:  
    - value: 0xB4,0x00  
    - description: Response Code  
  - <SOCKET_FLAG>  
    - parameter type:  
    - value: 2  
    - description: Socket error
  - <STATUS>  
    - parameter type:  
    - value: 1  
    - description: Command successful  
    - value: -2  
    - description: Command failed  
    - value: -3  
    - description: Failed to create  
    - value: -4  
    - description: Target port connection error
### 4.3.4 UDP Server

**Command:**
rak_udp_server

**Syntax:**
typedef struct {
    uint32_t cmd;
    uint16_t dest_port;
    uint16_t local_port;
} rak_client_t;

datatype:
- **cmd**: uint32_t
- **dest_port**: uint16_t
- **local_port**: uint16_t

**Parameter:**
This command is used to create a port in local and wait for data remote port, if the remote port needs to establish a connection with this port, the remote port sends data to the port, and the module will remain the last connectionthrough which sending data to the local port, and other connections are invalid. If the local port is created successfully, the module will return a hexadecimal identifier that is used to manage the connection. This command can create up to eight connections. Port numbers are in sorted in ascending order.

**Return Value:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CODE&gt;</td>
<td>2</td>
<td>0x82,0x00</td>
<td>Response Code</td>
</tr>
<tr>
<td>&lt;SOCKET_FLAG&gt;</td>
<td>2</td>
<td>0–7</td>
<td>Socket identifier</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1</td>
<td>0</td>
<td>Command successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2</td>
<td>Failed to create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-3</td>
<td>Failed to bind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td>Target port connection error</td>
</tr>
</tbody>
</table>

4.3.5 Creating Multicast Communication

**Command:**
rak_multicast

**Syntax:**
typedef struct {
    uint32_t cmd;
    uint32_t dest_addr;
    uint16_t dest_port;
    uint16_t local_port;
} rak_client_t;

**Parameter:**
This command is used to create a UDP multicast socket, a specified multicast IP can be added to router, allowing data communication within the group. Port numbers are in sorted in ascending order.

**Return Value:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4</td>
<td>0xC4,0x97,0x00,0x00</td>
<td>Command Code</td>
</tr>
<tr>
<td>dest_addr</td>
<td>4</td>
<td>224.0.0.0,229.255.255.255</td>
<td>multicast IP address</td>
</tr>
<tr>
<td>dest_port</td>
<td>2</td>
<td>1-65535</td>
<td>Target port number</td>
</tr>
<tr>
<td>local_port</td>
<td>2</td>
<td>1-65535</td>
<td>Local port number</td>
</tr>
</tbody>
</table>

4.3.6 Closing Socket

**Command:**
rak_socket_close

**Syntax:**
typedef struct {
    uint32_t cmd;
    uint16_t dest_port;
    uint16_t local_port;
} rak_server_t;

datatype:
- **cmd**: uint32_t
- **dest_port**: uint16_t
- **local_port**: uint16_t

**Parameter:**
It is used to close the already opened socket identifier.

**Return Value:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4</td>
<td>0xB6,0x97,0x00,0x00</td>
<td>Command Code</td>
</tr>
<tr>
<td>dest_addr</td>
<td>4</td>
<td>124.0.0.0,229.255.255.255</td>
<td>Target IP address</td>
</tr>
<tr>
<td>dest_port</td>
<td>2</td>
<td>1-65535</td>
<td>Target port</td>
</tr>
<tr>
<td>flag</td>
<td>2</td>
<td>0-7</td>
<td>Socket identifier</td>
</tr>
<tr>
<td>len</td>
<td>2</td>
<td>1-1400</td>
<td>Data length</td>
</tr>
<tr>
<td>buffer</td>
<td>1-1400</td>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>

4.3.7 Sending Data

**Command:**
rak_send_data

**Syntax:**
typedef struct {
    uint32_t cmd;
    uint16_t dest_addr;
    uint16_t dest_port;
    uint16_t socket_flag;
    uint16_t len;
    char buffer[1400];
} rak_send_t;

**Parameter:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4</td>
<td>0x85,0x07,0x00,0x00</td>
<td>Command Code</td>
</tr>
<tr>
<td>dest_addr</td>
<td>4</td>
<td>224.0.0.0,229.255.255.255</td>
<td>Target IP address</td>
</tr>
<tr>
<td>dest_port</td>
<td>2</td>
<td>1-65535</td>
<td>Target port</td>
</tr>
<tr>
<td>flag</td>
<td>2</td>
<td>0-7</td>
<td>Socket identifier</td>
</tr>
<tr>
<td>len</td>
<td>2</td>
<td>1-1400</td>
<td>Data length</td>
</tr>
<tr>
<td>buffer</td>
<td>1-1400</td>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>

**Parameter:**
This command is used to send data to the target port identifier, the maximum data length is 1400, buffer can be data in any format, module will send data without any treatment. If the connection is a TCP connection, then the destination IP and destination port can be omitted, returned with value 0. When the connection is UDP, if not specified, the validation is 0. If it needs to send data to specified target as UDP, fill in the target IP, and target port number. Port numbers are in sorted in ascending order.

**Return Value:**
- no

4.3.8 Receiving Data

**Command:**
rak_recv_data

**Syntax:**
uint32_t cmd;

**Parameter:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4</td>
<td>0x85,0x07,0x00,0x00</td>
<td>Command Code</td>
</tr>
<tr>
<td>dest_addr</td>
<td>4</td>
<td>224.0.0.0,229.255.255.255</td>
<td>Target IP address</td>
</tr>
<tr>
<td>dest_port</td>
<td>2</td>
<td>1-65535</td>
<td>Target port</td>
</tr>
<tr>
<td>flag</td>
<td>2</td>
<td>0-7</td>
<td>Socket identifier</td>
</tr>
</tbody>
</table>
Parameter Bytes Value Description
Cmd 4 0x86,0x97,0x00,0x00 Command Code

Parameter:
This command is to read data command of module. The results can be command results, or the network data and connection information. The data type can be viewed by CODE.

Return Value:
 CODE> 0xC8 Receiving data

Receiving data from network

Parameter Bytes Value Description
<CODE> 2 0x00,0x00 Response Code
<SOCKET_FLAG> 2 0-7 Port descriptor
<DataLen> 2 Data length
<IP_PORT> 2 Port number
<IP_ADDR> 2 IP address
<DATA> 1-1400 Data
<STATUS> 1 0 Successful

<CODE> 0xC9 socket connected

Parameter Bytes Value Description
<CODE> 2 0xC0,0x00 Response Code
<SOCKET_FLAG> 2 0-7 Port descriptor
<dummy> 2 Invalid data
<IP_ADDR> 2 IP address
<STATUS> 1 0 Successful

<CODE> 0xCA socket disconnected

Parameter Bytes Value Description
<CODE> 2 0x0A,0x00 Response Code
<SOCKET_FLAG> 2 0-7 Port descriptor
<dummy> 2 Invalid data
<IP_ADDR> 2 IP address
<STATUS> 1 0 Successful

4.4 Save Parameters Commands

4.4.1 StoringNetwork Configuration Parameters

Command:
 rak_storeconfig_data

Syntax:
typedef struct {
uint32_t cmd;
uint32_t feature_bitmap;
uint1_t net_type;
uint8_t sec_mode;
ethtubcap[64];
ip_param ip_param;
ap_config ap_config;
} config_t

Parameter:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Value Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4 0xC6,0x97,0x00,0x00 Command Code</td>
</tr>
<tr>
<td>&lt;CODE&gt;</td>
<td>2 0xC8,0x00 Response Code</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1 0 Successful</td>
</tr>
</tbody>
</table>

network connected

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Value Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4 0xCC,0x00 Command Code</td>
</tr>
<tr>
<td>&lt;CODE&gt;</td>
<td>2 0xC6,0x97,0x00,0x00 Command Code</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1 0 Successful</td>
</tr>
</tbody>
</table>

network disconnected

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Value Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4 0xC6,0x90 Command Code</td>
</tr>
<tr>
<td>&lt;CODE&gt;</td>
<td>2 0xC6,0x00 Response Code</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1 0 Successful</td>
</tr>
</tbody>
</table>

4.4.2 StoringCurrent Network Parameters

Command:
rak_storeconfig

Syntax:
typedef struct {
uint32_t cmd;
params params;
user_name user_name;
charuser_psk[17];
} config_t

Parameter:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Value Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4 0xC6,0x97,0x00,0x00 Command Code</td>
</tr>
<tr>
<td>&lt;CODE&gt;</td>
<td>2 0xC6,0x97,0x00,0x00 Command Code</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1 0 Successful</td>
</tr>
</tbody>
</table>

4.4.3 Modifying NetworkParameters

Command:
rak_web_store

Syntax:
typedef struct {
uint32_t cmd;
config config;
user_name user_name;
charuser_psk[17];
} config_t

Parameter:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Value Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4 0xC6,0x97,0x00,0x00 Command Code</td>
</tr>
<tr>
<td>&lt;CODE&gt;</td>
<td>2 0xC6,0x97,0x00,0x00 Command Code</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1 0 Successful</td>
</tr>
</tbody>
</table>

4.4.4 EnablingAutomaticConnection

Command:
rak_auto_connect

Syntax:
uint32_t cmd;

Parameter:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Value Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4 0xC6,0x97,0x00,0x00 Command Code</td>
</tr>
</tbody>
</table>

Network Parameters

- dhcp_mode:
  0 = STA
  1 = DHCP client
- Ssid:
  ASCII Network identifier
- psk:
  ASCII Network key
- dummy:
  Null data
- ip_param:
  IP parameters
- ap_config:
  AP advanced parameters

Parameter:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Value Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4 0xC6,0x97,0x00,0x00 Command Code</td>
</tr>
<tr>
<td>&lt;CODE&gt;</td>
<td>2 0xC6,0x97,0x00,0x00 Command Code</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1 0 Successful</td>
</tr>
</tbody>
</table>

Network Parameters

- 0 = Command successful
- -2 = Command failed
Parameter:
Use the saved network parameters to enable automatic networking. Automatically run internal scan, join and IP setting, and then return IP allocation results.

Return Value:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CODE&gt;</td>
<td>2</td>
<td>0xC6,0x00</td>
<td>Response Code</td>
</tr>
<tr>
<td>&lt;MAC&gt;</td>
<td>6</td>
<td>IP address</td>
<td>MAC address</td>
</tr>
<tr>
<td>&lt;ADDR&gt;</td>
<td>4</td>
<td>Subnet mask</td>
<td>IP address</td>
</tr>
<tr>
<td>&lt;GW&gt;</td>
<td>4</td>
<td>Gateway</td>
<td>Gateway</td>
</tr>
<tr>
<td>&lt;DNS1&gt;</td>
<td>4</td>
<td>DNS Server 1</td>
<td>DNS Server 1</td>
</tr>
<tr>
<td>&lt;DNS2&gt;</td>
<td>4</td>
<td>DNS Server 2</td>
<td>DNS Server 2</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1</td>
<td>0</td>
<td>Command successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2</td>
<td>Failed to find specified SSID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-3</td>
<td>Failed to join router</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td>Failed to allocate IP address</td>
</tr>
</tbody>
</table>

4.4.5 Starting Web Server

Command: rak_start_web
Syntax: uint32_t cmd;

Parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4</td>
<td>IP address</td>
<td>Command Code</td>
</tr>
</tbody>
</table>

It is used to start the embedded WEB service. Module will start the WEB with default parameters, typically in AP mode. When user is added, user can use the browser to configure the module parameters for wireless modules or wireless firmware upgrade.

Return Value:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ap_config&gt;</td>
<td>3</td>
<td>AP parameters</td>
<td>AP parameters</td>
</tr>
<tr>
<td>&lt;SIAFUs&gt;</td>
<td>1</td>
<td>0</td>
<td>Command successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2</td>
<td>Command failed, or timeout</td>
</tr>
</tbody>
</table>

4.4.7 Getting Web Server Built-in Parameters

Command: Rak_get_webconfig
Syntax: uint32_t cmd;

Parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4</td>
<td>IP address</td>
<td>Command Code</td>
</tr>
</tbody>
</table>

This command is available to save network parameters.

Return Value:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CODE&gt;</td>
<td>2</td>
<td>0xC6,0x00</td>
<td>Response Code</td>
</tr>
<tr>
<td>params</td>
<td>128</td>
<td>Network parameters</td>
<td>Network parameters</td>
</tr>
<tr>
<td>User_name</td>
<td>17</td>
<td>User name</td>
<td>User name</td>
</tr>
<tr>
<td>User_psk</td>
<td>17</td>
<td>Network encryption</td>
<td>Password</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1</td>
<td>0</td>
<td>Command successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2</td>
<td>Command failed, or timeout</td>
</tr>
</tbody>
</table>

4.4.6 Getting Saved Network Parameters

Command: rak_get_storeconfig
Syntax: uint32_t cmd;

Parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>4</td>
<td>0x89,0x00</td>
<td>Command Code</td>
</tr>
</tbody>
</table>

This command is used to save network parameters.

Return Value:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CODE&gt;</td>
<td>2</td>
<td>0xC6,0x00</td>
<td>Response Code</td>
</tr>
<tr>
<td>&lt;STATUS&gt;</td>
<td>1</td>
<td>0</td>
<td>Configuration is successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2</td>
<td>Configuration timeout</td>
</tr>
</tbody>
</table>

5 Sales and Service

Beijing
FAE mailbox: allan.jin@rakwireless.com
Tel: 010-82671601
Fax: 010-82678368
Address: Room 1108, Jin Yanlong Building, Xuanqi Desheng Gate Haidian District, Beijing

Shanghai
FAE mailbox: steven.tang@rakwireless.com
Tel: 021-51113558
Fax: 021-51113220
Address: Room 306, Ran East Business Center 150 Building 1, No. 2161 Wanyuan Road, Minhang District, Shanghai

Shenzhen
FAE mailbox: vincent.wu@rakwireless.com
Tel: 0755-82271586
Fax: 0755-85236551
Address: Room 406, Tsinghua information comprehensive building, Nanshan Science Park North District, Shenzhen
## 6 Revision History

Version History and modify the contents:

<table>
<thead>
<tr>
<th>Version</th>
<th>Modifications</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.0</td>
<td>Initial Draft</td>
<td>2014-03-11</td>
</tr>
<tr>
<td>V1.1</td>
<td>Modify draft and release</td>
<td>2014-03-28</td>
</tr>
<tr>
<td>V1.2</td>
<td>Modify command returns status, data structure</td>
<td>2014-06-09</td>
</tr>
</tbody>
</table>
SPI CHIP SELECT

UART

SPI

LEDs

VCC_MODULE

3V3

GND

TP1

TP2

TP3

TP4

TP5

R23 4.7k

R5 47k

D3

LED

D4

LED

J1

JUMPER

C1 47uF

R8 10k

R4 820

R3 820

R24 1k

C1 47uF

The

Wirelessly accessing instruments via GPIB/WLAN

Date: Thursday, July 10, 2014

Rev: V1.1

Sheet 5 of 6
Appendix J - PCB Design

TOP VIEW

BOTTOM VIEW

/* FILENAME : defines.h       */
/* AUTHOR : Nathan Quinteiro   <nathan.quinteiro@gmail.com> */
/*------------------------------------------------------------------------------*/
/* FUNCTION : Contains general definition used in project */
/*------------------------------------------------------------------------------*/
/* REVISION : 1.0 (version for ATmega164PA) */
/****************************************************************+++++++++++++++*/

#ifndef DEFINE_H_
#define DEFINE_H_
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>

typedef unsigned char BOOL;
#ifndef false
#define false 0
#endif
#ifndef true
#define true 1
#endif

#define ON 1
#define OFF 0
#define NULL 0x00;

typedef int State;
#endif
```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include "XF/xf.h"
#include "Hardware/io.h"
#include "SM_main.h"
#include "GPIB/GPIB.h"

void init(void)
{
    SREG = 0x00;       // Reset status register
    SMCR = 0x04;       // Select "Power down" mode without
                       // enabling to enter in sleep mode
    OSCCAL = 0x4C;     // Oscillator Calibration for 8MHz
    DDRA = 0xFF;      // I/O initialisation
    PORTA = 0x00;
    DDRB = (1<<PE)| (1<<WIFI_RESET)| (1<<WIFI_CS)| (1<<MOSI)| (1<<SCK);
    PORTB = (1<<MISO)| (1<<WIFI_CS)| (1<<WIFI_RESET);
    DDRC = (1<<DAV) ;
    PORTC = 0xFF;
    DDRD = (1<<LEDA)| (1<<LEDB)| (1<<TE)| (1<<DC);
}
```
PORTD = (1<<BT1);

/*--------------------------------------------------------------------------*/
/*     Timer1 initialisation: 5ms */
/*--------------------------------------------------------------------------*/
TCCR1A = 0x00; // Set CTC operation mode
TCCR1B = 0x0A; // Set prescaler 8
OCR1AH = 0x27; // A output compare value:9999 (10 ms)
OCR1AL = 0x0F;
OCR1AL = 0x63;
TIMSK1 = 0x02;
TIFR1 = 0x02; // Enable A output match interrupt

/*--------------------------------------------------------------------------*/
/*   Interrupt init: */
/*--------------------------------------------------------------------------*/
EICRA = (1<<ISC20)|(1<<ISC00); //any edge of INT0 (Button) and INT2 (WIFI_INTR) generate interrupts
EIMSK = 0x05; //INT0 (Button) and INT2 (WIFI INTR) are enabled
PCICR = 0x05; //Enable interrupt on pin change for PORTC (GPIB interface gesture)
             //and on pin change for PORTA (GPIB DIOs)
PCMSK2 = 0xFF; //Enable interrupt on all pins of PORTC
PCMSK0 = 0xFF; //Enable interrupt on all pins of PORTA

/*--------------------------------------------------------------------------*/
/*  SPI initialisation: Master */
/*--------------------------------------------------------------------------*/
SPCR0 = (0<<SPIE0)|(1<<SPE0)|(1<<MSTR0)|(1<<SPR10); // Set SPI enable and as master, spi interrupt enable
SPSR0 = (1<<SPIF0)|(1<<SPI2X0); // Enable double SPI speed, clk/2

/*--------------------------------------------------------------------------*/
/*   Enable global interrupts */
/*--------------------------------------------------------------------------*/
sei();

int main(void)
{
    init(); //Hardware initialization
    GPIB_set_source(); //Software initialization
    XF_init(); //Start the main states machine
    start_program();
}
#ifndef STATEMACHINE_H_
#define STATEMACHINE_H_

#include "Hardware/io.h"
#include "XF/xf.h"
#include "Utility/defines.h"
#include "SPI/spi.h"
#include "RAK/RAK411.h"
#include "RAK/SM_RAK_init.h"
#include "Utility/utility.h"
#include "GPIB/GPIB.h"
#include "GPIB/SM_GPIB_send.h"
#include "GPIB/SM_GPIB_receive.h"

void start_program();

extern void Interrupt_getPORTC();
extern void Interrupt_getPORTA();

//The three task running on the XF
void IO_management(Event ev);
void RAK_management(Event ev);
void GPIB_management(Event ev);

//Init of States machines
void SM_RAK_init();
void SM_GPIB_init();
void SM_IO_init();

//GPIB management states function
extern void SendByte(Event ev);
extern void ReceiveByte(Event ev);

//Function to program a LED blinking session
void LED_program(int period, int numberofblinks);

#endif /* INCFILE1_H_ */
}
#include "SM_main.h"

TimerID longclickTimer;

//variables used for the LEDs blink
int blink = 0;
int blinkperiod = 0;

void Reset()
{
    SM_RAK_init();
    SM_GPIB_init();
    XF_pushEvent(evInitLed, false);
    XF_scheduleTimer(100, evRAKreset, false);
}

//Beginning of the program, after the initialization of the XF
void start_program()
{
    //Initialize the states machines
    Reset();
    //Infinite loop, pop event from the Event list of the XF and dispatch
    //them in the three tasks
    while(1)
    {
        Event ev = XF_popEvent(false);
        IO_management(ev);
        RAK_management(ev);
        GPIB_management(ev);
    }
}

/**************************************************************************/
/* I/O management task                                                   */
/**************************************************************************/

//Handle the LEDs display and the reset of Wi-Fi module
//with the events pushed by the two other tasks
void IO_management(Event ev)
{
    switch (ev)
    {
        case evBDown:   longclickTimer = XF_scheduleTimer(longClickTime, evLongClick, false);
                        break;
        case evBUp:     XF_unscheduleTimer(longclickTimer, false);
                        break;
        case evLongClick:    Reset();
                        break;
    }
case evLedBOn: setLED(LEDB, ON);  
break;  
case evLedBOff: setLED(LEDB, OFF);  
break;  
case evLedAOn: setLED(LEDA, ON);  
break;  
case evLedAOff: setLED(LEDA, OFF);  
break;  
case evBlinkOn: setLED(LEDA, ON);  
setLED(LEDB, ON);  
XF_scheduleTimer(blinkperiod, evBlinkOff, false);  
break;  
case evBlinkOff:  
  
  blink--;  
setLED(LEDA, OFF);  
setLED(LEDB, OFF);  
if (blink>0)  
  XF_scheduleTimer(blinkperiod, evBlinkOn, false);  
break;  
case evInitLed: LED_program(400, 2);  
break;  
case evRAKreset: PORTB &= (0xFF^(1<<WIFI_RESET));  
  //Reset Wifi Module  
break;  
case evRAKset: PORTB |= (1<<WIFI_RESET);  
  //Release the reset and start the module  
break;  
}  
//Function to program a LED blinking session  
void LED_program(int period, int numberofblinks)  
{  
  XF_scheduleTimer(period, evBlinkOn, false);  
  blink = numberofblinks;  
  blinkperiod = period;  
}
#ifndef XFDEF
#define XFDEF

//@ Events

#define evInitLed 10
#define evBUp 17
#define evBDown 18
#define evLedAOn 19
#define evLedAOff 20
#define evLedBOn 21
#define evLedBOff 22
#define evBlinkOn 23
#define evBlinkOff 24
#define evLongClick 25

#define evSRQ_released 50
#define evSRQ_asserted 51
#define evDAV_released 52
#define evDAV_asserted 53
#define evEOI_released 54
#define evEOI_asserted 55
#define evATN_released 56
#define evATN_asserted 57
#define evNRFD_released 58
#define evNRFD_asserted 59
#define evIFC_released 60
#define evIFC_asserted 61
#define evNDAC_released 62
#define evNDAC_asserted 63
#define evREN_released 64
#define evREN_asserted 65
#define evDIOs_asserted 66
#define evDIOs_released 67
#define evGPIBsendCMDdone 68
#define evGPIBsendDATAdone 69
#define evGPIB_timeout 70
#define evGPIBreadDATAdone 71

#define evRAKreset 100
#define evRAKset 101
#define evRAKinitwaitend 102
#define evRAKinitwaitend 103
```c
#define evWIFIintrHigh 104
#define evWIFIintrLow 105
#define evRAKsendwaitend 106
#define evRAKreceivewaitend 107

// TIME BASE FOR XF
#define TICKINTERVAL 10
#ifdef __cplusplus
extern "C"
{
#endif
#include "../Utility/defines.h"

typedef unsigned char Event;
typedef unsigned int Time;
typedef unsigned char TimerID;

typedef enum IRFLAG
{
    IFTMR1,
    IFBUP,
    IFBDOWN
} IRFLAG;

typedef struct Timer
{
    Time tm;
    Event ev;
    TimerID id;
} Timer;

#define MAXTIMER 32
#define MAXEVENT 32
#define NULLEVENT 0
#define NULLTIMER 9999
#define NULLID 0
#define longClickTime 2000

typedef struct XF
{
    // this will be the attributes of the femto-xf
    Timer timerList[MAXTIMER];
    Event eventQueue[MAXEVENT];
} XF;

// prototypes of xf - interface
void XF_init();
Event XF_popEvent(BOOL inISR);
void XF_pushEvent(Event ev, BOOL inISR);
TimerID XF_scheduleTimer(Time tm, Event ev, BOOL inISR);
void XF_unscheduleTimer(TimerID id, BOOL inISR);
void XF_ISR(IRFLAG fl);
#ifdef __cplusplus
}
#endif
#endif
```
#include <avr/interrupt.h>
#include "xf.h"

XF theXF;

inline void ENTERCRITICAL(BOOL inISR)
{
    if (inISR == false)
    {
        cli(); //disable global interrupts
    }
    else
    {
        //do nothing
    }
}

inline void LEAVECRITICAL(BOOL inISR)
{
    if (inISR == false)
    {
        sei(); //enable global interrupts
    }
    else
    {
        //do nothing
    }
}

void XF_init()
{
    int i;
    for (i=0; i<MAXEVENT; i++)
    {
        theXF.eventQueue[i] = NULLEVENT;
    }

    for (i=0; i<MAXTIMER; i++)
    {
        theXF.timerList[i].tm = NULLTIMER;
        theXF.timerList[i].ev = NULLEVENT;
        theXF.timerList[i].id = NULLID;
    }
}
Event XF_popEvent(BOOL inISR)
{
    Event ev;
    int i;
    ev = NULLEVENT;
    ENTERCRITICAL(inISR);
    if (theXF.eventQueue[0] != NULLEVENT)
    {
        ev = theXF.eventQueue[0];
    }
    for (i=0; i<MAXEVENT-1 && theXF.eventQueue[i] != NULLEVENT; i++)
    {
        theXF.eventQueue[i] = theXF.eventQueue[i+1];
    }
    theXF.eventQueue[i-1] = NULLEVENT;
    LEAVECRITICAL(inISR);
    return ev;
}

void XF_pushEvent(Event ev, BOOL inISR)
{
    int i;
    ENTERCRITICAL(inISR);
    for (i=0; i<MAXEVENT; i++)
    {
        if (theXF.eventQueue[i] == NULLEVENT)
        {
            theXF.eventQueue[i] = ev;
            break;
        }
    }
    // here you could use done to react
    // if eventqueue is full (done == 0)
    LEAVECRITICAL(inISR);
}

TimerID XF_scheduleTimer(Time tm, Event ev, BOOL inISR)
{
    static unsigned char TID = 0;
    int done;
    int i;
    done = 0;
    ENTERCRITICAL(inISR);
    for (i=0; i<MAXTIMER; i++)
    {
        if (theXF.timerList[i].id == NULLID)
        {
            theXF.timerList[i].tm = tm;
            theXF.timerList[i].ev = ev;
            TID++;
            if (TID >= 255)
            {
                TID = 1;
            }
            theXF.timerList[i].id = TID;
            done = TID;
        }
    }
}
void XF_unscheduleTimer(TimerID id, BOOL inISR)
{
    int i;

    ENTERCRITICAL(inISR);
    for (i=0; i<MAXTIMER; i++)
    {
        if (theXF.timerList[i].id == id)
        {
            theXF.timerList[i].tm = NULLTIMER;
            theXF.timerList[i].ev = NULLEVENT;
            theXF.timerList[i].id = NULLID;
            break;
        }
    }
    LEAVECRITICAL(inISR);
}

void XF_decrementAndQueueTimers()
{
    int i;
    for (i=0; i<MAXTIMER; i++)
    {
        if (theXF.timerList[i].id != NULLID)
        {
            theXF.timerList[i].tm = TICKINTERVAL;
            if (theXF.timerList[i].tm ==0)
            {
                XF_pushEvent(theXF.timerList[i].ev, true);
                XF_unscheduleTimer(theXF.timerList[i].id, true);
            }
        }
    }
}

void XF_ISR(IRFLAG fl)
{
    switch (fl)
    {
    case IFTMR1:
        XF_decrementAndQueueTimers();
        break;
    case IFBUP:
        XF_pushEvent(evBUp, true);
        break;
    case IFBDOWN:
        XF_pushEvent(evBDown, true);
        break;
    }
}
FILENAME : interrupt.c       
AUTHOR : Nathan Quinteiro <nathan.quinteiro@gmail.com>  
FUNCTION : Execute all interrupt handling routine.  
REVISION : 1.0 (version for ATmega164PA)     

#include <avr/io.h>  
#include <avr/interrupt.h>  
#include <avr/sleep.h>  
#include "../Utility/defines.h"  
#include "io.h"  
#include "../XF/xf.h"

variables  
uint8_t oldPORTC;  
uint8_t oldPORTA;  

void Interrupt_getPORTC()  
{  
oldPORTC = PINC;  //save the current value of PORTC to compare on next interruption   
    //in order to know which pin has moved  
}  

void Interrupt_getPORTA()  
{  
oldPORTA = PINA;  //save the current value of PORTA to compare on next interruption   
    //in order to know which pin has moved  
}  

SIGNAL(TIMER1_COMPA_vect)  
{  
    XF_ISR(IFTMR1);  
}

SIGNAL(INT0_vect)  
{  
    if((PIND & (1<<BT1)))  
    {  
        XF_ISR(IFBUP);  
    }  
    else  
    {  
        XF_ISR(IFBDOWN);  
    }  
}
SIGNAL (INT2_vect)
{
  if((PINB & (1<<WIFIINTR)))
  {
    XF_pushEvent(evWIFIintrHigh, true);
  }
  else
  {
    XF_pushEvent(evWIFIintrLow, true);
  }
}

SIGNAL (PCINT0_vect)
{
  if(oldPORTA == 0xFF)
  {
    XF_pushEvent(evDIOs_asserted, true);
  }
  else
  {
    XF_pushEvent(evDIOs_released, true);
  }
  Interrupt_getPORTA();
}

SIGNAL (PCINT2_vect)
{
  if((oldPORTC ^ PINC) == (1<<SRQ))
  {
    if ((PINC & (1<<SRQ)) == 0)
    {
      XF_pushEvent(evSRQ_asserted, true);
    }
    else
    {
      XF_pushEvent(evSRQReleased, true);
    }
  }
  if((oldPORTC ^ PINC) == (1<<DAV))
  {
    if ((PINC & (1<<DAV)) == 0)
    {
      XF_pushEvent(evDAV_asserted, true);
    }
    else
    {
      XF_pushEvent(evDAV_released, true);
    }
  }
if((oldPORTC ^ PINC) == (1<<EOI))
{
    if ((PINC & (1<<EOI)) == 0)
        XF_pushEvent(evEOI_asserted, true);
    else
        XF_pushEvent(evEOI_released, true);
}
if((oldPORTC ^ PINC) == (1<<ATN))
{
    if ((PINC & (1<<ATN)) == 0)
        XF_pushEvent(evATN_asserted, true);
    else
        XF_pushEvent(evATN_released, true);
}
if((oldPORTC ^ PINC) == (1<<NRFD))
{
    if ((PINC & (1<<NRFD)) == 0)
        XF_pushEvent(evNRFD_asserted, true);
    else
        XF_pushEvent(evNRFD_released, true);
}
if((oldPORTC ^ PINC) == (1<<IFC))
{
    if ((PINC & (1<<IFC)) == 0)
        XF_pushEvent(evIFC_asserted, true);
    else
        XF_pushEvent(evIFC_released, true);
}
if((oldPORTC ^ PINC) == (1<<NDAC))
{
    if ((PINC & (1<<NDAC)) == 0)
        XF_pushEvent(evNDAC_asserted, true);
    else
        XF_pushEvent(evNDAC_released, true);
}
if((oldPORTC ^ PINC) == (1<<REN))
{
    if ((PINC & (1<<REN)) == 0)
        XF_pushEvent(evREN_asserted, true);
    else
        XF_pushEvent(evREN_released, true);
}

Interrupt_getPORTC();

 //////////////////////////////////////////////////////////////////////////
/* FUNCTION : SPI_STC_vect */
/* INPUT : - */
/* OUTPUT : - */
/* COMMENTS : ISR for SPI transfer complete */
 //////////////////////////////////////////////////////////////////////////
SIGNAL(SPI_STC_vect)
{
    XF_pushEvent(evSPIWriteEnd, false);
}
#ifndef IO_H_
#define IO_H_

#include "../Utility/defines.h"

void setLED(int LED, int state);
BOOL getLED(int LED);
int getBT();
void setWifi();
void resetWifi();
int getWifiINTR();

//SPI pins definition
#define MOSI PORTB5
#define MISO PORTB6
#define SCK PORTB7
#define WIFI_CS PORTB4

//GPIB pins
#define DIO PORTA
#define GPIB PORTC
#define REN PORTC0
#define IFC PORTC1
#define NDAC PORTC2
#define NRFD PORTC3
#define DAV PORTC4
#define E0I PORTC5
#define ATN PORTC6
#define SRQ PORTC7

//WIFI module pins
#define WIFI_RESET PORTB3
#define WIFI_INTR PORTB2
#define SPI_READY PORTB1

//GPIB drivers pins
#define PE PORTB0
#define TE PORTD5
#define DC PORTD6

//LEDs and button
#define LEDA PORTD4
#define LEDB PORTD3
#define BT1 PORTD2

#define IO_H_ */
/*************/ 
/* FILENAME : io.c */  
/* AUTHOR : Nathan Quinteiro  <nathan.quinteiro@gmail.com> */  
/*----------------------------------------------------------------------------*/  
/* FUNCTION : Handle the gesture of IO of the MCU */  
/*----------------------------------------------------------------------------*/  
/* REVISION : 1.0 (version for ATmega164PA) */  
/*----------------------------------------------------------------------------*/  
/#include  
uint8_t ledaState;  
uint8_t ledbState;  
//Set state of the specified LED  
void setLED(int LED, int state)  
{  
    if(LED == LEDA)  
    {  
        PORTD = (PORTD & 0xEF) | (state << LEDA);  
        ledaState = state;  
    }  
    if(LED == LEDB)  
    {  
        PORTD = (PORTD & 0xF7) | (state << LEDB);  
        ledbState = state;  
    }  
}  
//Get the actual state of the specified LED  
BOOL getLED(int LED)  
{  
    if(LED == LEDA)  
    {  
        return ledaState;  
    }  
    if(LED == LEDB)  
    {  
        return ledbState;  
    }  
    return OFF;  
}  
//Get the state of button  
int getBT()  
{  
    if ((PIND & (1<<BT1)) > 0) return OFF; //Return OFF if button is released, else ON  
    else return ON;  
}  
void setWifi()  
{  
    PORTB |= (1<<WIFI_RESET);  
}  
void resetWifi()  
{  
    PORTB &= (0xFF^(1<<WIFI_RESET));  
}  
int getWifiINTR()  
{  
    if ((PIND & (1<<WIFI_INTR)) > 0) return ON; //Return ON if wifi module INTR is high, else OFF  
    else return OFF;  
}
#ifndef SPI_H
#define SPI_H

#include <avr/io.h>

uint8_t SPI_MasterTransmit(char cData);

#endif /* SPI_H */

#include "spi.h"
#include "../Hardware/io.h"

/* Flags */
extern int fSPIWR;

inline uint8_t SPI_MasterTransmit(char cData)
{
    PORTB &= ~(0xFF^(1<<WIFI_CS));
    //Start transmission
    SPDR0 = cData;
    //Wait for transmission complete
    while(!(SPSR0 & (1<<SPIF0)));
    PORTB |= (1<<WIFI_CS);
    return SPDR0;
}
#ifndef RAK_H_
#define RAK_H_

#include "../Utility/defines.h"
#include "../Utility/utility.h"
#include "../SPI/spi.h"

uint8_t RAK_sendHeader(uint8_t command, uint16_t len);
void RAK_sendParameters(char * data, uint16_t size);
uint16_t RAK_get_resp_len();
void RAK_receiveResp(char * re_data, int len);
uint8_t RAK_readData();
void SM_RAK_init();

//STATES of the RAK states machine
#define stRAKinit 10
#define stRAKconfigure 11
#define stRAKworking 12
#define stRAKread 13
#define stRAKsend 14
#define stRAKwait 15

//STATES of the RAK send command states machine
#define stSendHead 20
#define stSendwait 21
#define stSendData 22
#define stSendIdle 23

//STATES of the RAK receive command response states machine
#define stReceiveHead 30
#define stReceivewait 31
#define stReceiveData 32
#define stReceiveIdle 33

//STATES of the RAK init states machine
#define stRAKsendInit 100
#define stRAKreceiveInit 101
#define stRAKsendGet_version 102
#define stRAKreceiveGet_version 103
#define stRAKsendU_scan 104
#define stRAKreceiveU_scan 105
#define stRAKsendGet_scan 106
#define stRAKreceiveGet_scan 107
#define stRAKsendSet_psk 108
#define stRAKreceiveSet_psk 109
#define stRAKsendConnect 110
#define stRAKreceiveConnect 111
#define stRAKsendGet_net_status 112
#define stRAKreceiveGet_net_status 113
#define stRAKsendSet_ip_static 114
#define stRAKreceiveSet_ip_static 115
#define stRAKsendQuery_ip 116
#define stRAKreceiveQuery_ip 117
#define stRAKsendTCP_server 118
#define stRAKreceiveTCP_server 119
#define stRAKinitwait 120
#define stRAKinitDone 121

/******************************************************************************
// Defines
/******************************************************************************
#define RAK_RSPCODE_LEN 2
#define HEAD_LENGTH 4
#define RAK_WELCOME_LEN 17
#define SPI_VALID 0x80
#define SPI_CMD_ACK 0x85 //Sent by RAK to acknowledge the command received from MCU
#define HOST_RESERVE 0x97
#define RAK_SSID_LEN 32
#define RAK_PSK_LEN 64
#define RAK_BSSID_LEN 6
#define RAK_AP_SCANNED_MAX 10
#define RAK_MAX_DATA_SIZE 256
#define RAK_USER_NAME_LEN 17
#define RAK_USER_PSK_LEN 17

/*cmd */
#define SYS_INIT_CMD 0xA0
#define READ_STATUS_CMD 0xA1
#define READ_DATA_CMD 0xA2
#define SCAN_CMD 0xA3
#define GET_SCAN_CMD 0xA4
#define SET_PSK_CMD 0xA5
#define CONNECT_CMD 0xA6
#define GET_CONN_STATUS_CMD 0xA7
#define SET_LISTEN_CMD 0xA8
#define GET_RSSI_CMD 0xA9
#define SET_APCONFIG_CMD 0xAA
#define SET_CHANNEL_CMD 0xAB
#define SET_DHCP_MODE_CMD 0xAC
#define SET_IPSTATIC_CMD 0xAD
#define IPCONFIG_QUERY_CMD 0xAE
#define DNS_QUERY_CMD 0xAF
#define PING_CMD 0xB0
#define CREATE_UDP_CLIENT_CMD 0xB1
#define CREATE_UDP_SERVER_CMD 0xB2
#define CREATE_TCP_CLIENT_CMD 0xB3
#define CREATE_TCP_SERVER_CMD 0xB4
#define CLOSEPORT_CMD 0xB5
#define SEND_DATA_CMD 0xB6
#define DISCONNECT_CMD 0xB7
#define SET_PWR_CMD 0xB8
#define GET_STORECONFIG_CMD 0xB9
#define GET_STOREWEB_CMD 0xBA
#define SET_BOOT_CMD 0xBC
#define DEL_DATA_CMD 0xBD
#define GET_VERSION_CMD 0xBE
#define SET_RESET_CMD 0xBF
#define STORE_DATA_CMD 0xC0
#define STORE_CONFIG_CMD 0xC1
#define EASY_CONFIG_CMD 0xC2
#define WPS_CMD 0xC3
#define CREATE_UDP_MULTICAST_CMD 0xC4
```c
#define STORE_WEB_CMD 0xC5
#define AOTU_CONNECT_CMD 0xC6
#define START_WEB_CMD 0xC7
#define RECEIVE_DATA 0xC8
#define SOCKET_CONN 0xC9
#define SOCKET_CLOSE 0xCA
#define NET_CONN 0xCB
#define NET_CLOSE 0xCC
#define STATION_MODE 0
#define AP_MODE 1
#define AD_HOC_MODE 2
#define SPI_WAIT_TIME 50 // If a command is not acknowledged, wait 50ms before sending it again
#define stdIPAdress 0xC0A8010A // IP address
#define stdMask 0xFFFFFFF00 // subnet mask
#define stdGw 0xC0A8010A // Gateway
#define stdDnssvr1 0xC0A8010A // DNS server 1
#define stdDnssvr2 0x00000000 // DNS server 2

#define RAK_IPSTATIC_IP_ADDRESS "192.168.11.122"
#define RAK_IPSTATIC_NETMASK "255.255.255.0"
#define RAK_IPSTATIC_GATEWAY "192.168.11.1"
#define RAK_IPSTATIC_DNS1 "192.168.11.1"
#define RAK_IPSTATIC_DNS2 "0"
#define send_data_header_len 16 // Size of data packet header (in bytes)
#define RAK_SCAN_CHANNEL 0
#define RAK SCAN SSID "TP-LINK_RAK411"
#define RAK_SET_PSK "1234567890"
#define RAK_GETSCAN_NUM 1
```

```c
/********************************************************************************
// Module Management Commands
********************************************************************************/
/*cmd struct*/
typedef struct {
    uint32_t     cmd;
} rak_common;

#define rak_sys_init 0xA0970000 // It is used to initialize module, and read boot information.
    //Return value : CODE - 2 bytes
    // DATA - 17 bytes
    // STATUS - 1 byte (0 successful, -2 failed)
#define rak_get_version 0xBE970000 // It is used to check module versions,
    //including versions of host and WLAN.
    //Return value : CODE - 2 bytes
    // HOST_FW - 8 bytes
    // Version delimiter - 1 byte
    // WLAN_FW - 6 bytes
    // STATUS - 1 byte (0 successful, -2 failed)
#define rak_setpwrmode 0xB8970000
```
typedef struct{
    uint32_t cmd;  // Command code
    uint32_t powermode;  // First byte indicate the mode : 0-3
} rak_pwr_mode_t;

#define rak_read_status 0xA1970000  //It is used to read the data of module status register.
    //Return value : STATUS - 1 byte (0x81 To read data, 0x82 Data cache is full)

#define rak_reset 0xBF970000  //It is used to resets the entire module via command.

/********************************************************************************
// Network Operation Commands
********************************************************************************/

#define rak_scan 0xA3970000  //It is used to scan wireless networks via command.
typedef struct{
    uint32_t cmd;  // Command code
    uint32_t channel;  // Scanning specified channel(s), scan all channels
    char ssid[32];  // Specified SSID, scan all channels if value is null
} rak_scan_t;  //Return value : CODE - 2 bytes 0xA3097
    // AP_NUM - 2 bytes
    // STATUS - 1 byte (0 successful, -2 failed)

#define rak_getscan 0xA4970000  //It is used to get the scanned network information.
typedef struct{
    uint32_t cmd;  // Command code
    uint32_t scan_num;  // Get the number of network information
} rak_getscan_t;  //Return value : CODE - 2 bytes : 0xA4
    // CHANNEL - 1 byte : 1 to 14, channel
    // RSSI - 1 byte : -99 to 0, channel intensity
    // SEC_MODE - 1 byte
    // SSID_LEN - 2 bytes, length of SSID
    // SSID - 32 bytes
    // BSSID - 6 bytes
    // STATUS - 1 byte (0 successful, -2 failed)

#define rak_set_psk 0xA5970000  //It is used to set network password.
typedef struct{
    uint32_t cmd;  // Command code
    char psk[64];  // Network password
} rak_psk_t;  //Return value : CODE - 2 bytes 0xA500
    // STATUS - 1 byte (0 successful, -2 failed)

#define rak_set_channel 0xAB970000  //It is used to set network channel.
typedef struct{
    uint32_t cmd;  // Command code
    uint32_t channel;  // 1 to 13, SettingAP / Ad-Hoc channel
} rak_channel_t;  //Return value : CODE - 2 bytes 0xA800
    // STATUS - 1 byte (0 successful, -2 failed)

#define rak_connect 0xA6970000  //It is used to connect an AP/ ADHOC network or establish a
    // specified AP Network.
typedef struct{
    uint32_t cmd;  // Command code
uint32_t mode;     // Select network mode : 0 = station, 1 = ap, 2 = ad-hoc
char ssid[32];     // SSID
}

rak_conn_t:        // Return value : CODE - 2 bytes 0xA600
// STATUS - 1 byte (0 successful, -2 cannot find SSID, -3 command failed)

#define rak_set_ip_static 0xAD970000
typedef struct{
    uint32_t cmd;      // Command code
    uint32_t addr;     // IP address
    uint32_t mask;     // subnet mask
    uint32_t gw;       // Gateway
    uint32_t dnssvr1;  // DNS server 1
    uint32_t dnssvr2;  // DNS server 2
} rak_ipstatic_t;    // Return value : CODE - 2 bytes 0xAD00
// STATUS - 1 byte (0 successful, -2 failed)

#define rak_ipconfig_dhcp 0xAC970000
typedef struct{
    uint32_t cmd;      // Command code
    uint32_t mode;     // Select DHCP mode : 0 = client, 1 = server
} rak_ipdhcp_t;      // Return value : DHCP SERVER - CODE - 2 bytes 0xAC00
// STATUS - 1 byte (0 successful, -2 failed)
// DHCP CLIENT - CODE - 2 bytes 0xAC00
// MAC - 6 bytes
// ADDR - 4 bytes
// MASK - 4 bytes
// GW - 4 bytes
// DNS1 - 4 bytes
// DNS2 - 4 bytes
// STATUS - 1 byte (0 successful, -2 failed)

#define rak_easy_config 0xC2970000
// This command initiates module EasyConfig function, along with APP software
// in the phone, enabling module automatically to be added in the specified network.
// After networking successfully, module automatically saves the parameters via
// rak_storeconfig command.
// Return value : CODE - 2 bytes 0xC200
// SSID - 32 bytes
// SEC MODE - 1 byte
// PSK - 64 bytes Network password
// STATUS - 1 byte (0 successful, -2 cannot find AP, -3 failed to be added in router, -4 ip failed to get, -5 Easy config failed)

#define rak_wps 0xC3970000
// This command initiates module WPS function. When successful, module
// automatically saves the parameters via rak_storeconfig command.
// Return value : CODE - 2 bytes 0xC300
// SSID - 32 bytes
// SEC MODE - 1 byte
// PSK - 64 bytes Network password
// STATUS - 1 byte (0 successful, -2 cannot find AP, -3 join the router failure, -4 ip failed to get, -5 WPS failed)

#define rak_get_constatus 0xA7970000
// It is used to get module network status.
// If the module is working in Station mode, this command is used to get wireless
// network connection status. 
// If the module is working in AP mode, this command is used to 
determine the 
// device’s connection status. 
// Return value: 
// CODE - 2 bytes 0xA700 
// STATUS - 1 byte (0 successful, -2 failed) 

#define rak_ipconfig_query 0xAE970000 
// It is used to get module IP information, including MAC address, IP address, 
// subnet mask, gateway, and DNS server. 
// Return value: 
// CODE - 2 bytes 0xAA00 
// MAC - 6 bytes 
// ADDR - 4 bytes 
// MASK - 4 bytes 
// GW - 4 bytes 
// DNS1 - 4 bytes 
// DNS2 - 4 bytes 
// STATUS - 1 byte (0 successful, -2 failed) 

#define rak_get_rssi 0xA9970000 
// It is used to get the current signal strength of the network. 
// Return value: 
// CODE - 2 bytes 0xA900 
// RSSI - 2 bytes, -99 to 0, current signal strength 
// STATUS - 1 byte (0 successful, -4 failed to obtain IP information) 

#define rak_dns 0xAF970000 
// It is used to convert domain to the corresponding IP address, the domain must 
// be configured available DNS server. 
// Return value: 
// CODE - 2 bytes 0xAF00 
// ADDR - 4 bytes, IP address 
// STATUS - 1 byte (0 successful, -2 DNS resolution failed) 

#define rak_ping 0xB0970000 
// It is used to run the ping command. 
// Return value: 
// CODE - 2 bytes 0xB000 
// STATUS - 1 byte (0 successful, -2 Cannot access host) 

#define rak_apconfig 0xAA970000 
// It is used to set up parameters for a wireless access point, such as the country 
// code, whether network name is hidden or not. 
// Return value: 
// CODE - 2 bytes 0xAA00 
// STATUS - 1 byte (0 successful, -2 failed) 

#define rak_set_listen 0xA8970000 
// It is used to set module beacon interval in Station mode. 
// In power saving mode, reducing power consumption can be realized
// via increasing parameter values, but by this way it may cause
delay in receiving wireless data.

    uint32_t cmd;  // Command code
    uint32_t time; // Need to refer to the wireless router settings
for specific parameters, 20-1000
} rak Beacon_t:
    //Return value : CODE - 2 bytes 0xA800
    // STATUS - 1 byte (0 successful, -2 failed)
#define rak_disconnect 0xB7970000 // It is used to disconnect the current wireless network.
    //Return value : CODE - 2 bytes 0xB700
    // STATUS - 1 byte (0 successful, -2 failed)

/********************************************************************************
// Socket Operation Commands
********************************************************************************/
#define rak_tcp_server 0xB4970000 //Module is used as a TCP server and creates a listening port, if the
operation is
used to manage
    uint32_t cmd;  // Command code
    uint16_t dummy; // Invalid data
    uint16_t port; // Local port number, 1 to 65535
} rak_server_t:
    //Return value : CODE - 2 bytes 0xB400
    // SOCKET_FLAG - 2 bytes 0 to 7, socket identifier
    // STATUS - 1 byte (0 successful, -2 failed to create, -3 failed to bind, -4 target port connection error)
#define rak_tcp_client 0xB3970000 //This command is to create a TCP CLIENT and connect with the
remote TCP
    uint32_t cmd;  // Command code
    uint32_t dest_addr; // Target IP address
    uint16_t dest_port; // Target port number, 1 to 65535
    uint16_t local_port; // Local port number, 1 to 65535
} rak_client_t:
    //Return value : CODE - 2 bytes 0xB400
    // SOCKET_FLAG - 2 bytes 0 to 7, socket identifier
    // STATUS - 1 byte (0 successful, -2 failed to create, -3 failed to bind, -4 target port connection error)
#define rak_udp_client 0xB1970000 //This command is to create a UDP port on the module and set remote IP
address
    //and port number, if you create successful, the module will return a hexadecimal
    //identifier that is used to manage the connection. This command can create up to
    //eight connections. Port numbers are in sorted in ascending order.
    //Command code
    // Target IP address
    // Target port number, 1 to 65535
    // Local port number, 1 to 65535
} rak_udp_t:
    //Return value : CODE - 2 bytes 0xB100
    // SOCKET_FLAG - 2 bytes 0 to 7, socket identifier
    // STATUS - 1 byte (0 successful, -2 failed to create, -3 failed to bind, -4 target port connection error)
```c
#define rak_udp_server 0xB2970000  // This command is used to create a port in local and wait for data from remote port.
//the remote port needs to establish a connection with this port, the remote port
//sends data to the port, and the module will remain the last connection through
//which sending data to the local port, and other connections are invalid. If the local
//port is created successfully, the module will return a hexadecimal identifier that is
//used to manage the connection. This command can create up to eight connections.
//Port numbers are in sorted in ascending order.
//   Command code
//   Invalid data
//   Local port number, 1 to 65535
//Return value : CODE - 2 bytes 0xB400
//   SOCKET_FLAG - 2 bytes 0 to 7,
socket identifier
// STATUS - 1 byte (0 successful, -2 failed to create, -3 failed to bind, -4 target port connection error)
#endif

#define rak_multicast 0xC4970000  // This command is used to create a UDP multicast socket, a specified multicast IP
//can be added to router, allowing data communication within the group. Port numbers are in sorted in ascending order.
//   Command code
//   Target multicast IP address
//   Target port number, 1 to 65535
//   Local port number, 1 to 65535
//Return value : CODE - 2 bytes 0xB100
//   SOCKET_FLAG - 2 bytes 0 to 7, socket identifier
//   STATUS - 1 byte (0 successful, -2 failed to create, -3 failed to bind, -4 target port connection error)
#endif

#define rak_socket_close 0xB5970000  // It is used to close the already opened socket identifier.
#define rak_send_data 0xB6970000  // This command is used to send data to the target (port identifier), the
//maximum data length is 1400, buffer can be data in any format, module will send
//data without any treatment. If the connection is a TCP connection, then the
//destination IP and destination port can be omitted, entered with value 0. When the
//connection is UDP, if not specified, the value can be 0. If it needs to send data to
//specified target as LUDP, fill in the target IP, and target port number. Port numbers
//are in sorted in ascending order.
```

#define rak_recv_data 0xB6970000  //This command is to read data commands of module. The results can be
command results, or the network data and connection information.
The data type
//can be viewed by CODE.
//For the Return value, see software programming manual of RAK411

/********************************************************************************
// Save Parameters Commands
/********************************************************************************

typedef struct{
    //It is used to save user parameters, including password, SSID, IP address, and
    //scan information.
    uint32_t cmd;  //Command code
    uint32_t feature_bitmap; //Switching features
    uint8_t net_type; //0 = station, 1 = ap, 2 = ad-hoc
    uint8_t sec_mode; //0 = network not encrypted, 1 = network encrypted
    uint8_t dhcp_mode; //0 = STA:DHCP client, 1 = STA:ip static
    char ssid[32]; //Network identifier
    char psk[64]; //Network key
    char ip_param[20]; //IP parameters
    char ap_config[3]; //AP advanced parameters
}config_t;

#define rak_storeconfig_data 0xC0970000
//It is used to save user parameters. Parameters can be successfully saved only
//after correctly performing the commands scan, connect and get IP.
//Return value : CODE - 2 bytes 0xC000
//   STATUS - 1 byte (0 successful, -2 failed)

#define rak_storeconfig 0xC1970000
//It is used to save user parameters. Parameters can be successfully saved only
//after correctly performing the commands scan, connect and get IP.
//Return value : CODE - 2 bytes 0xC100
//   STATUS - 1 byte (0 successful, -2 failed)

#define rak_web_store 0xC5970000
//It is used to save the network parameters used to initiatenetwork.
//Return value : CODE - 2 bytes 0xC100
//   STATUS - 1 byte (0 successful, -2 failed)

typedef struct{
    //It is used to save the network parameters used to initiatenetwork.
    uint32_t cmd;  //Command code
    config_t params; //Network parameters
    char user_name[17]; //Web authentication user name
    char user_psk[17]; //Web authentication password
}web_t;

#define rak_auto_connect 0xC6970000
//Use the saved network parameters to enable automatic networking.
//Automatically run internal scan, join and IP setting, and then return IP allocation
//Return value : CODE - 2 bytes 0xC600
//   MAC - 6 bytes
//   ADDR - 4 bytes
//   MASK- 4 bytes
//   GW - 4 bytes
//   DNS1 - 4 bytes
//   DNS2 - 4 bytes
//   STATUS - 1 byte (0 successful, -2 SSID not found, -3 faild to join router, -4 failed to allocate IP address)

#define rak_start_web 0xC7970000
//It is used to start the embedded WEB service. Module will start the WEB with
//default parameters, typically in AP mode. When user is added, user can use the
//browser to configure the module parameters for wireless modules or wireless
#define rak_get_storeconfig 0xB9970000
// This command is used to save network parameters.
// Return value : CODE - 2 bytes 0xB900
// FEATURE_BITMAP - 4 bytes
// NET_TYPE - 1 byte: 0 = station, 1 = AP, 2 = ad-hoc
// SEC_MODE - 1 byte: 0 = network not encrypted, 1 = network encrypted
// DHCP_MODE - 1 byte: 0 = STA:DHCP client, 1 = STA:ip static
// SSID - 33 bytes : Network identifier
// PSK - 65 Bytes : Network key
// DUMMY - 2 bytes : Null data
// IP_PARAM - 20 bytes : IP parameters
// AP_CONFIG - 30 bytes : AP parameters
// STATUS - 1 byte (0 successful, -2 failed)

#define rak_get_webconfig 0xBA970000
// This command is available to save network parameters.
// Return value : CODE - 2 bytes 0xBA00
// PARAMS - 126 bytes : Network parameters
// USER_NAME - 17 bytes
// USER_PSK - 17 bytes
// STATUS - 1 byte (0 successful, -2 failed)

/****************************************************************************/
// parameters structures
/****************************************************************************/
/*listen struct*/
typedef struct {
    uint32_t cmd;
    uint32_t time;
} rak_listen_t;

/*ip struct*/
typedef struct {
    uint32_t addr;
    uint32_t mask;
    uint32_t gw;
    uint32_t dns1;
    uint32_t dns2;
} rak_ip_param;

typedef struct {
    uint8_t hidden;
    uint8_t countryCode[3];
} rak_ap_param;

typedef struct {
    uint32_t feature_bitmap;
    uint8_t net_type;
    uint8_t channel;
    uint8_t sec_mode;
    uint8_t dhcp_mode;
    char ssid[33];
    char psk[65];
    uint8_t dummy[2];
    rak_ip_param ip_param;
}
typedef struct {
    uint32_t cmd;
    param_t rak_param;
} rak_param_t;

/* web struct*/
typedef struct {
    uint32_t cmd;
    param_t net_params;
    char user_name[RAK_USER_NAME_LEN];
    char user_psk[RAK_USER_PSK_LEN];
} rak_web_t;

/********************************************************************************
// Structure containing all commands
********************************************************************************/

/*api struct*/
typedef union {
    uint8_t band;
    uint8_t powerMode;
    uint8_t macAddress[6];
    rak_channel_t uChannel;
    rak_scan_t uScanFrame;
    rak_getscan_t uGetscan;
    rak_conn_t uConnFrame;
    rak_ipdhcp_t uIpdhcpFrame;
    rak_pwr_mode_t uPwrModeFrame;
    rak_psk_t uPskFrame;
    rak_param_t uParamFrame;
    rak_web_t uWebFrame;
    rak_ping_t uPingFrame;
    rak_dns_t uDnsFrame;
    rak_server_t uServerFrame;
    rak_ipstatic_t uIpstaticFrame;
    rak_apconfig_t uApconfigFrame;
    rak_listen_t uListenFrame;
    rak_send_t uSendFrame;
} rak_api;

/********************************************************************************
// Commands responses
********************************************************************************/

/*scan rsp struct*/
typedef struct {
    uint8_t rspCode[RAK_RSPCODE_LEN];
    uint16_t ap_num;
    uint8_t status;
} rak_scanResponse;

/*scan info struct*/
typedef struct {
    uint8_t rfChannel;
    uint8_t rssiVal;
    uint16_t securityMode;
    uint16_t ssid_len;
}
typedef struct {
    uint8_t ssid[RACK_SSID_LEN];
    uint8_t bssid[RACK_BSSID_LEN];
} rak_scanInfo;

//getscan rsp struct
typedef struct {
    uint8_t rspCode[RACK_RSPCODE_LEN];
    rak_scanInfo strScanInfo[RACK_AP_SCANNED_MAX];
    uint8_t status;
} rak_getscanResponse;

/*getrssi rsp struct*/
typedef struct {
    uint8_t rspCode[RACK_RSPCODE_LEN];
    uint16_t rssi;
    uint8_t status;
} rak_getrssiResponse;

/*socketEst rsp struct*/
typedef struct {
    uint8_t rspCode[RACK_RSPCODE_LEN];
    uint16_t socket_flag;
    uint16_t dummy;
    uint16_t ip_port;
    uint8_t ip_addr[4];
} rak_recvsocketEst;

/*version rsp struct*/
typedef struct {
    uint8_t rspCode[RACK_RSPCODE_LEN];
    uint8_t host_fw[8];
    uint8_t dummy;
    uint8_t wla_fw[6];
    uint8_t status;
} rak_versionFrame;

/*ip rsp struct*/
typedef struct {
    uint8_t rspCode[RACK_RSPCODE_LEN];
    uint8_t macAddr[6];
    uint8_t ipaddr[6];
    uint8_t netmask[4];
    uint8_t gateway[4];
    uint8_t dns1[4];
    uint8_t dns2[4];
    uint8_t status;
} rak_ipparamFrameRcv;

/*socket rsp struct*/
typedef struct {
    uint8_t rspCode[RACK_RSPCODE_LEN];
    uint16_t socket_flag;
    uint8_t status;
} rak_socketFrameRcv;

/*init rsp struct*/
typedef struct {
    uint8_t rspCode[RACK_RSPCODE_LEN];
    uint8_t strdata[RACK_WELCOME_LEN];
    uint8_t status;
}
typedef struct {
    uint8_t            rspCode[RAK_RSPCODE_LEN];
    uint16_t           socket_flag;
    uint16_t           data_len;
    uint16_t           ip_port;
    uint8_t           ip_addr[4];
    uint8_t           recvDataBuf[RAK_MAX_DATA_SIZE];
} rak_recvFrame;

typedef struct {
    uint8_t            rspCode[RAK_RSPCODE_LEN];
    uint8_t            status;
} rak_mgmtResponse;

typedef struct {
    uint8_t            rspCode[RAK_RSPCODE_LEN];
    uint8_t           addr[4];
    uint8_t            status;
} rak_dnsResponse;

typedef struct {
    uint8_t            rspCode[RAK_RSPCODE_LEN];
    uint8_t           ssid[RAK_SSID_LEN];
    uint8_t            sec_mode;
    uint8_t           psk[RAK_PSK_LEN];
    uint8_t            status;
} rak_easynetResponse;

typedef struct {
    uint8_t            rspCode[RAK_RSPCODE_LEN];
    param_t            rak_param;
    uint8_t            status;
} rak_paramResponse;

typedef struct {
    uint8_t            rspCode[RAK_RSPCODE_LEN];
    param_t            net_params;
    char               user_name[RAK_USER_NAME_LEN];
    char               user_psk[RAK_USER_PSK_LEN];
    uint8_t            status;
} rak_webResponse;

typedef union {
    uint8_t            rspCode[RAK_RSPCODE_LEN];
    rak_initResponse   initResponse;
    rak_scanResponse   scanResponse;
    rak_getscanResponse getscanResponse;
    rak_getrssiResponse getrssiResponse;
    rak_ipparamFrameRcv ipparamFrameRcv;
    rak_socketFrameRcv  socketFrameRcv;
    rak_recvFrame      recvFrame;
    rak_recvsocketEst  recvsocketEst;
    rak_versionFrame   versionFrame;
    rak_easynetResponse easynetFrame;
} rak_initResponse;
rak_mgmtResponse mgmtResponse;
rak_paramResponse paramFrame;
rak_webResponse webFrame;
uint8_t...
uint8_t dhcp_mode);
void set_web_cmd(web_t * web, uint32_t cmd);

#endif /* RAK_H_ */
#include "RAK411.h"

int i;
int j;

/**
 * @fn   uint8_t RAK_sendCommand(uint8_t command, uint16_t len)
 * @brief  Send a command header
 * @param[in]  uint8_t command, the command code, uint6_t len, the length of the parameters to send
 * @param[out]  none
 * @return  uint8_t, the value returned by the module, 0 or Acknowledge
 */
uint8_t RAK_sendHeader(uint8_t command, uint16_t len)
{
    uint8_t receivedData = 0x00;
    uint8_t cmd[4];
    cmd[0] = command;
    cmd[1] = HOST_RESERVE;
    cmd[2] = ((len&0xFF));
    cmd[3] = ((len>>8)&0xFF);
    for(i = 0; i <= 3; i++)
    {
        receivedData = SPI_MasterTransmit(cmd[i]);
        if (receivedData == SPI_CMD_ACK)
        {
            return receivedData;
        }
    }
    return receivedData;
}

/**
 * @fn   void RAK_sendStructCmd(char * data, uint16_t size)
 * @brief  Send a the command parameter
 * @param[in]  char * data, pointer on the structure containing the parameters to send, uint16_t size, length of packet to send
 * @param[out]  none
 * @return  none
 */
void RAK_sendParameters(char * data, uint16_t size)
{
    for(i = 0; i < size; i++)
    {
        SPI_MasterTransmit(*data);
        data++;
    }
}

/**
 * @fn   uint8_t RAK_readData()
 * @brief  Read a byte with the SPI
 * @param[in]  none
 * @param[out]  none
 * @return  uint8_t, the value of the byte read on the SPI
 */
uint8_t RAK_readData()
{
    return SPI_MasterTransmit(0x00);
}

/**
 * @fn   uint16_t RAK_get_resp_len()
 * @brief  Get the length of packet to be received from the Wi-Fi module
 * @param[in]  none
 * @param[out]  none
 * @return  uint16_t, size of packet to be received
 */
uint16_t RAK_get_resp_len()
{
    uint16_t len = SPI_MasterTransmit(0x00) << 8;
    len |= SPI_MasterTransmit(0x00);
    return len;
}

/**
 * @fn   void RAK_receiveResp(char *re_data, int len)
 * @brief  Receive response from the Wi-Fi module and save it in the response structure
 * @param[in]  int len, size of packet to receive
 * @param[out]  char * re_data, pointer of the structure where the response is saved
 * @return  none
 */
void RAK_receiveResp(char *re_data, int len)
{
    int i = 0;
    for (i=0; i < len; i++) {
        re_data[i] = SPI_MasterTransmit(0x00);
    }
}

void set_pwr_mode_cmd(rak_pwr_mode_t * pwrmode, uint32_t cmd, uint32_t powermode)
{
    pwrmode->cmd = (cmd);
    pwrmode->powermode = (powermode);
}

void set_scan_cmd(rak_scan_t * scan, uint32_t cmd, uint32_t channel)
{
    scan->cmd = (cmd);
    scan->channel = (channel);
}

void set_getscan_cmd(rak_getscan_t * getscan, uint32_t cmd, uint32_t scan_num)
{
    getscan->cmd = (cmd);
    getscan->scan_num = (scan_num);
}

void set_psk_cmd(rak_psk_t * psk_cmd, uint32_t cmd)
{
    psk_cmd->cmd = (cmd);
}
void set_channel_cmd(rak_channel_t * channel_cmd, uint32_t cmd, uint32_t channel) {
    channel_cmd->cmd = cmd;
    channel_cmd->channel = channel;
}

void set_conn_cmd(rak_conn_t * conn, uint32_t cmd, uint32_t mode) {
    conn->cmd = cmd;
    conn->mode = mode;
}

void set_ipstatic_cmd(rak_ipstatic_t * ipstatic, uint32_t cmd, uint32_t addr, uint32_t mask, uint32_t gw, uint32_t dnsrv1, uint32_t dnsrv2) {
    ipstatic->cmd = cmd;
    ipstatic->addr = addr;
    ipstatic->mask = mask;
    ipstatic->gw = gw;
    ipstatic->dnsrv1 = dnsrv1;
    ipstatic->dnsrv2 = dnsrv2;
}

void set_ipdhcp_cmd(rak_ipdhcp_t * ipdhcp, uint32_t cmd, uint32_t mode) {
    ipdhcp->cmd = cmd;
    ipdhcp->mode = mode;
}

void set_dns_cmd(rak_dns_t * dns, uint32_t cmd) {
    dns->cmd = cmd;
}

void set_ping_cmd(rak_ping_t * ping, uint32_t cmd, uint32_t hostaddr, uint32_t count, uint32_t size) {
    ping->cmd = cmd;
    ping->hostaddr = hostaddr;
    ping->count = count;
    ping->size = size;
}

void set_apconfig_cmd(rak_apconfig_t * apconfig, uint32_t cmd, uint8_t hidden, uint16_t countryCode) {
    apconfig->cmd = cmd;
    apconfig->hidden = hidden;
    apconfig->countryCode = countryCode;
}

void set_beacon_cmd(rak_beacon_t * beacon, uint32_t cmd, uint32_t time) {
    beacon->cmd = cmd;
    beacon->time = time;
}

void set_server_cmd(rak_server_t * server, uint32_t cmd, uint16_t dummy, uint16_t port) {
    server->cmd = cmd;
    server->dummy = dummy;
    server->port = port;
}
void set_client_cmd(rak_client_t * client, uint32_t cmd, uint32_t dest_addr, uint16_t dest_port, uint16_t local_port)
{
    client->cmd = bigEndian_32(cmd);
    client->dest_addr = dest_addr;
    client->dest_port = dest_port;
    client->local_port = local_port;
}

void set_close_cmd(rak_close_t * close, uint32_t cmd, uint16_t dummy, uint16_t flag)
{
    close->cmd = cmd;
    close->dummy = dummy;
    close->flag = flag;
}

void set_send_cmd(rak_send_t * send, uint32_t cmd, uint32_t dest_addr, uint16_t dest_port, uint16_t socket_flag, uint16_t len)
{
    send->cmd = cmd;
    send->dest_addr = dest_addr;
    send->dest_port = dest_port;
    send->socket_flag = socket_flag;
    send->len = len;
}

void set_config_cmd(config_t * config, uint32_t cmd, uint32_t feature_bitmap, uint8_t net_type, uint8_t sec_mode, uint8_t dhcp_mode)
{
    config->cmd = cmd;
    config->feature_bitmap = feature_bitmap;
    config->net_type = net_type;
    config->sec_mode = sec_mode;
    config->dhcp_mode = dhcp_mode;
}

void set_web_cmd(web_t * web, uint32_t cmd)
{
    web->cmd = cmd;
}
/**
 * FILENAME : SM_RAK.h
 * AUTHOR : Nathan Quinteiro <nathan.quinteiro@gmail.com>
 * FUNCTION : State machine handling communication with the RAK411 module
 * REVISION : 1.0
 */

#include "../Hardware/io.h"
#include "../XF/xf.h"
#include "../Utility/defines.h"
#include "../Utility/utility.h"
#include "RAK411.h"
#include "SM_RAK_init.h"
#include "../SM_main.h"
#include <string.h>
#include "Smartphone.h"
#include "../GPIB/SM_GPIB.h"

#ifndef SM_RAK_H_
#define SM_RAK_H_
#endif

void SmartphoneHandling();

#endif /* SM_RAK_H_*/

/*********************************************************************************/

// RAK management states machine

void RAK_management(Event ev)
{
    //On start or after a problem with the Wi-Fi module, the module is reset
    //and the initialization sequence is launched
    if (ev == evRAKreset)
    {
        SM_RAK_init();
        XF_scheduleTimer(20, evRAKset, false);
    }
}

int f = 0;
uint32_t clientIPAddress = 0x00;

//States of RAK states machine
State RAK_state;
extern State RAK_init_state;
extern State RAK_configure_state;

//Commands used
rak_api RAK_command;

//RAK response
rak_CmdRsp RAK_response;

//Smartphone packet
smartphone_packet * packet;
RESPONSE_packet * resp_packet;

void RAK_management(Event ev)
{
RAK_state = stRAKwait;
}
if (ev == evRAKset)
{
  RAK_state = stRAKinit;
}

switch (RAK_state)
{
    case stRAKwait:
        break;
    case stRAKinit:
        // Initialize Wi-Fi module, configure an access point and open a TCP server socket
        if (RAK_sys_init(ev, &RAK_response) == true)
        {
            RAK_state = stRAKworking;
            LED_program(400, 3);
        }
        break;
    case stRAKworking:
        // Once module is initialized wait to receive commands data from Wi-Fi module
        if (ev == evWIFIintrHigh || PINB & (1<<WIFI_INTR))
        {
            SM_RAK_initReceive(&RAK_response);
            RAK_state = stRAKread;
        }
        break;
    case stRAKread:
        // When GPIB finished the transmission of data or command, send response to smartphone
        if (ev == evGPIBsendCMDdone || ev == evGPIBsendDATAdone)
        {
            set_send_cmd(&RAK_command.uSendFrame, SEND_DATA_CMD, 0, 0,
                         RAK_response.recvFrame.socket_flag, 3);
            resp_packet = (RESPONSE_packet*) &RAK_command.uSendFrame.buffer;
            resp_packet->command = RESPONSE_CMD;
            resp_packet->responseTo = WRITE_CMD;
            resp_packet->status = STATUS_OK;
            SM_RAK_initSend(SEND_DATA_CMD, (char*) &RAK_command,
                            send_data_header_len + 3);
            RAK_state = stRAKsend;
        }
        break;
    case stRAKsend:
        // When GPIB had timeout, send response with error status to smartphone
        if (ev == evGPIB_timeout)
        {
            set_send_cmd(&RAK_command.uSendFrame, SEND_DATA_CMD, 0, 0,
                         RAK_response.recvFrame.socket_flag, 3);
            resp_packet = (RESPONSE_packet*) &RAK_command.uSendFrame.buffer;
            resp_packet->command = RESPONSE_CMD;
            resp_packet->responseTo = WRITE_CMD;
            resp_packet->status = GPIB_ER;
            SM_RAK_initSend(SEND_DATA_CMD, (char*) &RAK_command,
                            send_data_header_len + 3);
            RAK_state = stRAKsend;
        }
        break;
    case stRAKread:
        // When GPIB finished the reception of data, send response to smartphone
        if (ev == evGPIBreadDATAdone)
        {
            set_send_cmd(&RAK_command.uSendFrame, SEND_DATA_CMD, 0, 0,
                         RAK_response.recvFrame.socket_flag, 4);
            resp_packet = (RESPONSE_packet*) &RAK_command.uSendFrame.buffer;
            resp_packet->command = RESPONSE_CMD;
            resp_packet->responseTo = READ_CMD;
            resp_packet->status = STATUS_OK;
            SM_RAK_initSend(SEND_DATA_CMD, (char*) &RAK_command,
                            send_data_header_len + 4);
            RAK_state = stRAKsend;
        }
    break;
}
else
    RAK_state = stRAKwait;
resp_packet->data[0] = 0x80;
SM_RAK_initSend(SEND_DATA_CMD, (char *)&RAK_command,
send_data_header_len + 4);

RAK_state = stRAKsend;
}
break;
case stRAKread: //Module has data to send
if(RAK_receiveCMD(ev))
{
    RAK_state = stRAKworking;
    switch(RAK_response.CmdRspBuf[0])
    {
        //If the data comes from the smartphone
        case RECEIVE_DATA: SmartphoneHandling();
            break;
        //When client connects to the TCP socket
        case SOCKET_CONN: LED_program(120, 8);
            break;
        //When client disconnects to the TCP socket
        case SOCKET_CLOSE: LED_program(200, 4);
            break;
        //When client connects to access point
        case NET_CONN: LED_program(120, 16);
            break;
        //When client disconnects to the point
        case NET_CLOSE: LED_program(200, 8);
    }
    break;
}

break;
case stRAKsend: //Sends command to the module
if(RAK_sendCMD(ev))
{
    RAK_state = stRAKworking;
}
break;
}

// SM Init
void SM_RAK_init()
{
    RAK_state = stRAKwait;
    RAK_init_state = stRAKsendInit;
}

// Handle communication with smartphone, reception and transmission of packet
void SmartphoneHandling()
{
    packet = (smartphone_packet *) &RAK_response.recvFrame.recvDataBuf;
    resp_packet = (RESPONSE_packet *) &RAK_command.uSendFrame.buffer;
    uint8_t SMARTPHONE_COMMAND = RAK_response.recvFrame.recvDataBuf[0];
    switch(SMARTPHONE_COMMAND)
    {
        case LED_CMD: //Set LEDs states with the states specified in the command
            if(packet->led.ledastate == ON) setLED(LEDA, ON);
else setLED(LEDA, OFF);
if(packet->led.ledbstate == ON) setLED(LEDB, ON);
else setLED(LEDB, OFF);
//Send LED response with the actual state on the LEDs
set_send_cmd(&RAK_command.uSendFrame, SEND_DATA_CMD, 0, 0,
RAK_response.recvFrame.socket_flag, 5);
resp_packet->command = RESPONSE_CMD;
resp_packet->responseTo = LED_CMD;
resp_packet->status = STATUS_OK;
resp_packet->data[0] = getLED(LEDA);
resp_packet->data[1] = getLED(LEDB);
SM_RAK_initSend(SEND_DATA_CMD, (char *) &RAK_command, send_data_header_len + 5);
break;

case WRITE_CMD: //If user wants to send a command to GPIB bus
if(packet->send.type == COMMAND)
{
    GPIB_send_bytes((char *) packet->send.data, packet->send.length, true);
    LED_program(100, packet->send.length);
}
//If user wants to send data to GPIB bus
if(packet->send.type == DATA)
{
    GPIB_send_bytes((char *) packet->send.data, packet->send.length, false);
    LED_program(100, packet->send.length);
}
break;

case READ_CMD: GPIB_receive_bytes(packet->send.length);
    LED_program(250, packet->send.type);
break;
}
FILENAME: SM_RAK_init.h
AUTHOR: Nathan Quinteiro <nathan.quinteiro@gmail.com>

FUNCTION: State machine of the initialization of RAK module

REVISION: 1.0

FILENAME: SM_RAK_init.c
AUTHOR: Nathan Quinteiro <nathan.quinteiro@gmail.com>

FUNCTION: State machine of the initialization of RAK

REVISION: 1.0

FILENAME: SM_RAK_init.h

#ifndef SM_RAK_INIT_H_
#define SM_RAK_INIT_H_

BOOL RAK_sys_init(Event ev, rak_CmdRsp * rsp);

#include "../Utility/defines.h"
#include "../Utility/utility.h"
#include "RAK411.h"
#include "../XF/xf.h"
#include "../Hardware/io.h"
#include "SM_RAK_sendCMD.h"
#include "SM_RAK_receiveCMD.h"
#include "RAKtest.h"
#include <string.h>

#endif /* SM_RAK_INIT_H_ */

FILENAME: SM_RAK_init.c

#include "SM_RAK_init.h"

State RAK_init_state:

rak_common cmd;
extern rak_api RAK_command;

RAK initialization states machine

BOOL RAK_sys_init(Event ev, rak_CmdRsp * rsp)

brief Start with the init command, then read registers and configure
the module in order to configure it as a TCP server to enable
communication with the smartphone

@in Event ev, last event that happen. rak_CmdRsp * rsp, pointer on the response structure
@out none
@return BOOL finished, true if the initialization is finished and successful

*/
BOOL RAK_sys_init(Event ev, rak_CmdRsp * rsp)
{
    switch(RAK_init_state)
    {
    case stRAKsendInit: if (RAK_sendHeader(SYS_INIT_CMD, 0) == SPI_CMD_ACK)
    {
        RAK_init_state = stRAKreceiveInit;
        return false;
    }
    else
    {
        XF_scheduleTimer(SPI_WAIT_TIME, evRAKinitwaitend, false);
        RAK_init_state = stRAKinitwait;
        return false;
    }
    break;
    case stRAKreceiveInit: RAK_receiveResp((char*) rsp, RAK_get_resp_len());
    initGetVersion();
    return false;
    break;
    case stRAKinitwait: if(ev == evRAKinitwaitend)
    {
        RAK_init_state = stRAKsendInit;
    }
    break;
    case stRAKsendGet_version: if(RAK_sendCMD(ev))
    {
        if(ev == evWIFIintrHigh)
        {
            SM_RAK_initReceive(rsp);
            RAK_init_state = stRAKreceiveGet_version;
        }
    }
    return false;
    break;
    case stRAKreceiveGet_version: if(RAK_receiveCMD(ev))
    {
    initUScan();
    }
    return false;
    break;
    case stRAKsendU_scan: if(RAK_sendCMD(ev))
    {
        if(ev == evWIFIintrHigh)
        {
            SM_RAK_initReceive(rsp);
            RAK_init_state = stRAKreceiveU_scan;
        }
    }
    return false;
    break;
    case stRAKreceiveU_scan: if(RAK_receiveCMD(ev))
    {
        initSetPsk();
    }
    return false;
    break;
    case stRAKsendSet_psk: if(RAK_sendCMD(ev))
    {
        if(ev == evWIFIintrHigh || PINB & (1<<WIFI_INTR))
SM_RAK_initReceive(rsp);
RAK_init_state = stRAKreceiveSet_psk;
}
}
return false;

break;
case stRAKreceiveSet_psk:
if(RAK_receiveCMD(ev))
{
    initConnect();
}
return false;
break;
case stRAKsendConnect:
if(RAK_sendCMD(ev))
{
    if(ev == evWIFIintrHigh || PINB & (1<<WIFI_INTR))
    {
        SM_RAK_initReceive(rsp);
        RAK_init_state = stRAKreceiveConnect;
    }
}
return false;
break;
case stRAKreceiveConnect:
if(RAK_receiveCMD(ev))
{
    initSetIPStatic();
}
return false;
break;
case stRAKsend_set_ip_static:
if(RAK_sendCMD(ev))
{
    if(ev == evWIFIintrHigh || PINB & (1<<WIFI_INTR))
    {
        SM_RAK_initReceive(rsp);
        RAK_init_state = stRAKreceive_set_ip_static;
    }
}
return false;
break;
case stRAKreceive_set_ip_static:
if(RAK_receiveCMD(ev))
{
    initTCPserver();
}
return false;
break;
case stRAKsendTCP_server:
if(RAK_sendCMD(ev))
{
    if(ev == evWIFIintrHigh || PINB & (1<<WIFI_INTR))
    {
        SM_RAK_initReceive(rsp);
        RAK_init_state = stRAKreceiveTCP_server;
    }
}
return false;
break;
case stRAKreceiveTCP_server:
if(RAK_receiveCMD(ev))
{
    RAK_init_state = stRAKinitDone;
}
return false;
break;
case stRAKinitDone:
return true;
break;
}
return false;
}

/********************************************************************************
// Initialization of the commands, the desired value are put on the structure
// in order to be sent to the Wi-Fi module
 ********************************************************************************/
void initGetVersion()
{
    cmd.cmd = GET_VERSION_CMD;
    SM_RAK_initSend(GET_VERSION_CMD, (char *) &cmd, sizeof(rak_common));
    RAK_init_state = stRAKsendGet_version;
}

void initScan()
{
    set_scan_cmd(&RAK_command.uScanFrame, SCAN_CMD, RAK_SCAN_CHANNEL);
    SM_RAK_initSend(SCAN_CMD, (char *) &RAK_command, sizeof(rak_scan_t));
    RAK_init_state = stRAKsendU_scan;
}

void initGetScan()
{
    set_getscan_cmd(&RAK_command.uGetscan, GET_SCAN_CMD, RAK_GETSCAN_NUM);
    SM_RAK_initSend(GET_SCAN_CMD, (char *) &RAK_command, sizeof(rak_getscan_t));
    RAK_init_state = stRAKsendGet_scan;
}

void initSetPsk()
{
    set_psk_cmd(&RAK_command.uPskFrame, SET_PSK_CMD);
    strcpy((char *) &RAK_command.uPskFrame.psk, RAK_SET_PSK);
    SM_RAK_initSend(SET_PSK_CMD, (char *) &RAK_command, sizeof(rak_psk_t));
    RAK_init_state = stRAKsendSet_psk;
}

void initConnect()
{
    set_conn_cmd(&RAK_command.uConnFrame, CONNECT_CMD, AP_MODE);
    strcpy((char *) &RAK_command.uConnFrame.ssid, "GPIB connector");
    SM_RAK_initSend(CONNECT_CMD, (char *) &RAK_command, sizeof(rak_conn_t));
    RAK_init_state = stRAKsendConnect;
}

void initGetNetStatus()
{
    RAK_init_state = stRAKsendGet_net_status;
    cmd.cmd = GET_CONN_STATUS_CMD;
    SM_RAK_initSend(GET_CONN_STATUS_CMD, (char *) &cmd, sizeof(rak_common));
}

void initSetIPStatic()
{
    set_ipstatic_cmd(&RAK_command.ipstaticFrame, SET_IPSTATIC_CMD, stdIPAdress, stdMask, stdGw, stdDnssvr1, stdDnssvr2);
    SM_RAK_initSend(SET_IPSTATIC_CMD, (char *) &RAK_command, sizeof(rak_ipstatic_t));
    RAK_init_state = stRAKsendSet_ip_static;
}

void initIPQuery()
{
    RAK_init_state = stRAKsendQuery_ip;
    cmd.cmd = IPCONFIG_QUERY_CMD;
    SM_RAK_initSend(IPCONFIG_QUERY_CMD, (char *) &cmd, sizeof(rak_common));
}
void initTCPServer()
{
    set_server_cmd(&RAK_command.uServerFrame, CREATE_TCP_SERVER_CMD, 0, 7);
    SM_RAK_InitSend(CREATE_TCP_SERVER_CMD, (char *)&RAK_command, sizeof(rak_server_t));
    RAK_init_state = stRAKsendTCP_server;
}

#include "/Utility/defines.h"
#include "RAK411.h"
#include "/../XF/xf.h"
#include "/../Hardware/io.h"

#ifndef SM_RAK_RECEIVECMD_H_
#define SM_RAK_RECEIVECMD_H_

BOOL RAK_receiveCMD(Event ev);
void SM_RAK_initReceive(rak_CmdRsp *rsp);
#endif /* SM_RAK_RECEIVECMD_H_ */

#include "SM_RAK_receiveCMD.h"

BOOL RAK_receiveCMD(Event ev);
void SM_RAK_initReceive(rak_CmdRsp *rsp);

State RAK_receive_state;
ra任何一个你的休眠
int tryreceive = 0;

switch(RAK_receive_state)
{
    case stReceiveHead:  // If module acknowledge the read data command
        if(RAK_sendHeader(READ_DATA_CMD, 0) == SPI_CMD_ACK)
        {
            // Go in read data state
            RAK_receive_state = stReceiveData;
```c
tryreceive = 0;
return false;
}

// If module does not acknowledge, wait 50ms before sending the command again
else
{
    // After 20 unsuccessful tries, reset the module
    tryreceive++;
    if (tryreceive > 20)
    {
        XF_pushEvent(evRAKreset, false);
    }
    else
    {
        XF_scheduleTimer(SPI_WAIT_TIME, evRAKreceivewaitend, false);
    }
    RAK_receive_state = stReceivewait;
    return false;
}
break;

case stReceiveData:
    // Get length of packet to read from the module, read it and save it in the
    response structure
    RAK_receiveResp((char*) rak_rsp, RAK_get_resp_len());
    RAK_receive_state = stReceiveIdle;
    break;

case stReceiveIdle:
    return true;
    break;

case stReceivewait:
    // Read command not acknowledge, wait 50 ms
    if (ev == evRAKreceivewaitend)
    {
        RAK_receive_state = stReceiveHead;
    }
    return false;
    break;

return false;
}

/********************************************************************************
// SM Init
********************************************************************************/

void SM_RAK_initReceive(rak_CmdRsp * rsp) {
    RAK_receive_state = stReceiveHead;
    rak_rsp = rsp;
}
```
#include "../Utility/defines.h"
#include "RAK411.h"
#include "../XF/xf.h"
#include "../Hardware/io.h"

#ifndef SM_RAK_SENDCMD_H_
#define SM_RAK_SENDCMD_H_

BOOL RAK_sendCMD(Event ev);
void SM_RAK_initSend(uint8_t cmd, char * dt, uint16_t len);

#endif /* SM_RAK_SENDCMD_H_ */

BOOL RAK_sendCMD(Event ev);
void SM_RAK_initSend(uint8_t cmd, char * dt, uint16_t len);

State RAK_send_state;
int trysend = 0;
int cmdLength;
uint8_t command;
char * data;

BOOL RAK_sendCMD(Event ev)
{
    switch(RAK_send_state)
    {
    case stSendHead:
        if(RAK_sendHeader(command, cmdLength) == SPI_CMD_ACK)
        {
            RAK_send_state = stSendData;
            trysend = 0;
            return false;
        }
    }
}
If module does not acknowledge command, wait 50ms before sending the command again

else {
    //After 20 unsuccessful tries, reset the Wi-Fi module
    trysend++;
    if (trysend > 20) {
        XF_pushEvent(evRAKreset, false);
    } else {
        XF_scheduleTimer(SPI_WAIT_TIME, evRAKsendwaitend, false);
    }
    RAK_send_state = stSendwait;
    return false;
} break;

case stSendData: //Send the parameters
    RAK_sendParameters(data, cmdLength);
    RAK_send_state = stSendIdle;
    break;

case stSendIdle: return true;
    break;

case stSendwait: //Command not acknowledge, wait 50 ms
    if (ev == evRAKsendwaitend) {
        RAK_send_state = stSendHead;
    }
    return false;
} break;

return false;

/********************************************************************************
// SM Init
********************************************************************************/

void SM_RAK_initSend(uint8_t cmd, char * dt, uint16_t len) {
    RAK_send_state = stSendHead;
    command = cmd;
    data = dt;
    cmdLength = len;
}
#ifndef SMARTPHONE_H_
define SMARTPHONE_H_

//Commands
#define WRITE_CMD 0x70
#define READ_CMD 0x71
#define RESPONSE_CMD 0x72
#define LED_CMD 0x73
#define COMMAND 0x74
#define DATA 0x75

//Status
#define STATUS_OK 0x00
#define GPIB_ER 0x01
#define PACK_ER 0x02

//Structure packet
typedef struct{
    uint8_t command;
    uint8_t type;
    uint8_t length;
    uint8_t data[];
}SEND_packet;

typedef struct{
    uint8_t command;
    uint8_t length;
}READ_packet;

typedef struct{
    uint8_t command;
    uint8_t responseTo;
    uint8_t status;
    uint8_t data[];
}RESPONSE_packet;

typedef struct{
    uint8_t command;
    uint8_t ledastate;
    uint8_t ledbstate;
}LED_packet;

typedef union{
    SEND_packet send;
    READ_packet read;
    LED_packet led;
}smartphone_packet;

#endif /* SMARTPHONE_H_ */
/* FILENAME : GPIB.h       */
/* AUTHOR : Nathan Quinteiro <nathan.quinteiro@gmail.com> */
/*------------------------------------------------------------------------------*/
/* FUNCTION : Handle the communication with the GPIB bus */
/*------------------------------------------------------------------------------*/
/* REVISION : 1.0 */
/*------------------------------------------------------------------------------*/

#include "../Utility/defines.h"
#include "../Hardware/io.h"

#ifndef GPIB_H_
#define GPIB_H_

//States of the GPIB states machine
#define stGPIBinit 50
#define stGPIBnone 51
#define stGPIBsendCMD 52
#define stGPIBsendDATA 53
#define stGPIBreceiveDATA 54

//STATES of the GPIB send data states machine
#define SGNS 520
#define SDYS 521
#define STRS 522
#define SWNS 523

#define GPIB_send_done 524

//STATES of the GPIB receive data states machine
#define ANRS 540
#define ACRS 541
#define ACDS 542
#define AWNS 543
#define GPIB_receive_done 544

//GPIB timeout
#define Timeout_GPIB 20000

//This method is used to send a byte on the GPIB bus, it handles the Handshake sequence
void GPIB_send_byte(char byte);

//This method set the MCU as a GPIB source (talker)
void GPIB_set_source();

//This method set the MCU as a GPIB acceptor (listener)
void GPIB_set_acceptor();

//This method asserts the DIO line with the desired value
void GPIB_assert_data(uint8_t data);

#endif /* GPIB_H_ */
#include "GPIB.h"

void GPIB_set_source()
{
    PORTD |= (1 << TE); // Set TE and PE to configure open collector output
    PORTB &= (0xFF ^ (1 << PE)); // and set the MCU as talker
    DDRA = 0xFF; // Set PORTA (DIO0-7) as output
    DDRC = (1 << DAV); // Set DAV as output, NDAC and NRFD are input
    DIO = 0xFF;
}

void GPIB_set_acceptor()
{
    PORTD &= (0xFF ^ (1 << TE)); // Set TE to 0 to configure input
    DDRA = 0x00; // Set PORTA (DIO0-7) as input
    PORTA = 0xFF;
    DDRC = ((1 << NDAC) | (1 << NRFD)); // Set DAV as input, NDAC and NRFD are output
}

void GPIB_assert_data(uint8_t data)
{
    DIO = data;
}
#include "../Hardware/io.h"
#include "../XF/xf.h"
#include "../Utility/defines.h"
#include "GPIB.h"
#include "SM_GPIB_receive.h"
#include "SM_GPIB_send.h"

#ifdef SM_GPIB_H_
#define SM_GPIB_H_

void GPIB_send_bytes(char *data, uint8_t nb, BOOL command);
void GPIB_receive_bytes(uint8_t nb);

#endif /* SM_GPIB_H_ */
#include "SM_GPIB.h"

//States of GPIB states machine
State GPIB_state;
TimerID GPIB_timeout;
//Bytes to send or received from instruments through GPIB bus
uint8_t GPIB_bytes[10];
uint8_t numberOfByteToSend;
uint8_t numberOfByteToReceive;
int i;

// GPIB management
void GPIB_management(Event ev)
{
    switch (GPIB_state)
    {
    case stGPIBnone:
        break;
    case stGPIBsendCMD:
        if (GPIB_SendByte(ev))
        {
            i++;
            //End of transmission?
            if (i == numberOfByteToSend)
            {
                GPIB_state = stGPIBnone;
                XF_pushEvent(evGPIBsendCMDdone, false);
                GPIB |= (1<<ATN);  //Drive ATN false (high level) to let talker send data
                XF_unscheduleTimer(GPIB_timeout, false);
            }
            //Transmit next byte
            else
            {
                SM_init_GPIB_send(GPIB_bytes[i]);
            }
        }
        break;
    case stGPIBsendDATA:
        if (GPIB_SendByte(ev))
        {
            i++;
            //End of transmission?
            if (i == numberOfByteToSend)
            {
                GPIB_state = stGPIBnone;
                XF_pushEvent(evGPIBsendDATAdone, false);
                XF_unscheduleTimer(GPIB_timeout, false);
            }
            //Transmit next byte
            else
            {
SM_init_GPIB_send(GPIB_bytes[i]);
}
break;

case stGPIB_receive_DATA:
    if (GPIB_ReceiveByte(ev))
    {
        i++;
        // End of reception?
        if (i == numberOfByteToReceive)
        {
            GPIB_state = stGPIBnone;
            XF_pushEvent(evGPIB_read_DATA_done, false);
            XF_unscheduleTimer(GPIB_timeout, false);
        }
        // Read next byte
        else
        {
            SM_init_GPIB_receive(&GPIB_bytes[i]);
        }
    }
    break;
}

/******************************************************************************
// Initialize a transmission of data or command on the GPIB bus
******************************************************************************/
void GPIB_send_bytes(char* data, uint8_t nb, BOOL command)
{
    if (command)
    {
        GPIB &= (0xFF^(1<<ATN));    // Drive ATN true (low level) to send command
        GPIB_state = stGPIBsendCMD;
    }
    else
    {
        GPIB_state = stGPIBsendDATA;
    }
    GPIB_set_source();
    for (i = 0; i < nb; i++)
    {
        GPIB_bytes[i] = data[i];
    }
    numberOfByteToSend = nb;
    i = 0;
    SM_init_GPIB_send(GPIB_bytes[i]);
    GPIB_timeout = XF_scheduleTimer(Timeout_GPIB, evGPIB_timeout, false);
}

/******************************************************************************
// Initialize a reception of data on the GPIB bus
******************************************************************************/
void GPIB_receive_bytes(uint8_t nb)
{
    GPIB_state = stGPIB_receive_DATA;
    GPIB_set_acceptor();
    numberOfByteToReceive = nb;
    i = 0;
    SM_init_GPIB_receive(&GPIB_bytes[i]);
    GPIB_timeout = XF_scheduleTimer(Timeout_GPIB, evGPIB_timeout, false);
}
// SM Init

void SM_GPIB_init()
{
    GPIB_state = stGPIBnone;
}

#ifndef GPIB_RECEIVE_SM_H_
#define GPIB_RECEIVE_SM_H_

#include "../Utility/defines.h"
#include "../Hardware/io.h"
#include "../XF/xf.h"
#include "GPIB.h"

// States machine
BOOL GPIB_ReceiveByte(Event ev);

// Init
void SM_init_GPIB_receive(uint8_t * byte);
void anrs(Event ev);
void acrs(Event ev);
void acds(Event ev);
void awns(Event ev);

#endif /* GPIB_RECEIVE_SM_H_ */

#include "SM_GPIB_send.h"

State GPIB_receive_state;
uint8_t * ReceiveByte;

// State Machine for the handshake sequence when MCU want to receive a byte on
// GPIB bus

BOOL GPIB_ReceiveByte(Event ev)
{
    switch (GPIB_receive_state)
    {
        case ANRS: anrs(ev); break;
        case ACRS: acrs(ev); break;
        case ACDS: acds(ev); break;
        case AWNS: awns(ev); break;
        case GPIB_receive_done: return true;
    }
    return false;
}

//Acceptor Not Ready State
void anrs(Event ev)
{
    if(ev == evDIOs_asserted)
    {
        PORTC |= (1<<NRFD); //release NRFD
        GPIB_receive_state = ACRS;
    }
}

//Acceptor Ready State
void acrs(Event ev)
{
    if(ev == evDAV_asserted)
    {
        GPIB_receive_state = ACDS;
        PORTC &= (0xFF^(1<<NRFD)); //assert NRFD
        *ReceiveByte = PINA; //read byte and save it
        PORTC |= (1<<NDAC); //release NDAC
    }
}

//Accept Data State
void acds(Event ev)
{
    GPIB_receive_state = AWNS;
}

//Acceptor waiting for new cycle state
void awns(Event ev)
{
    if(ev == evDAV_released)
    {
        GPIB_receive_state = ANRS;
        PORTC &= (0xFF^(1<<NDAC)); //assert NDAC
    }
}

//Init
void SM_init_GPIB_receive(uint8_t * byte)
{
    GPIB_receive_state = ANRS;
}
ReceiveByte = byte;

/***********************
/* FILENAME : SM_GPIB_send.h */
/* AUTHOR : Nathan Quinteiro <nathan.quinteiro@gmail.com> */
/* Revision : 1.0 */
/***********************
#ifndef GPIB_SEND_SM_H
#define GPIB_SEND_SM_H
#include "../Utility/defines.h"
#include "../Hardware/io.h"
#include "../XF/xf.h"
#include "GPIB.h"
#include "../SM_main.h"
//States machine
BOOL GPIB_SendByte(Event ev);
//Init
void SM_init_GPIB_send(char byte);
//GPIB send byte states function
void sgns(Event ev);
void sdys(Event ev);
void strs(Event ev);
void swns(Event ev);
#endif /* GPIB_SEND_SM_H */

/***********************
/* FILENAME : SM_GPIB_send.c */
/* AUTHOR : Nathan Quinteiro <nathan.quinteiro@gmail.com> */
/* Revision : 1.0 */
/***********************
#include "SM_GPIB_send.h"
State GPIB_send_state;
/***********************
//State Machine for the handshake sequence when MCU want to send a byte on GPIB
//bus
***********************
BOOL GPIB_SendByte(Event ev)
{
    switch (GPIB_send_state)
    {
        case SGNS: sgns(ev);
                    break;
        case SDYS: sdys(ev);
                    break;
        case STRS: strs(ev);
                    break;
        case SWNS: swns(ev);
                    break;
        case GPIB_send_done: return true;
                    break;
    }
    return false;
}
void sgns(Event ev)
{
    if (ev == evNRFD_released)
    {
        PORTC &= (0xFF ^ (1 << DAV)); //assert DAV
        GPIB_send_state = SDYS;
        LED_program(200, 1);
    }
}

void sdys(Event ev)
{
    if (ev == evNRFD_asserted)
    {
        GPIB_send_state = STRS;
        LED_program(200, 2);
    }
}

void strs(Event ev)
{
    if (ev == evNDAC_released)
    {
        PORTC |= (1 << DAV); //release DAV
        GPIB_send_state = SWNS;
        LED_program(200, 3);
    }
}

void swns(Event ev)
{
    if (ev == evNDAC_asserted)
    {
        GPIB_send_state = GPIB_send_done;
        DIO = 0xFF;
        LED_program(200, 4);
    }
}

void SM_init_GPIB_send(char byte)
{
    GPIB_send_state = SGNS;
    DIO = byte;
}
package nathan.quinteiro.gpibconnector;

import java.io.BufferedReader;

public class MainActivity extends ActionBarActivity {

    // Commands from Wi-Fi module header
    public final byte WRITE_CMD = 0x70;
    public final byte READ_CMD = 0x71;
    public final byte RESPONSE_CMD = 0x72;
    public final byte LED_CMD = 0x73;
    public final byte COMMAND = 0x74;
    public final byte DATA = 0x75;

    // Status
    public final byte STATUS_OK = 0x00;
    public final byte GPIB_ER = 0x01;
    public final byte PACK_ER = 0x02;

    private String librarypath;
    List<GPIBcommand> GPIBlib = null;

    MainActivity mActivity;
    LibraryActivity libraryActivity;
    TextView tv;
    TextView tvresponse;
    Button ba;
    Button bb;
    Button bsenddata;
    Button bsendcmd;
    Button breceivedata;
    Button bstart;
    ImageView ima;
    ImageView imb;
    EditText etData;
    Spinner spCMD;
    Spinner spREC;
    int a = 0;
    int b = 0;

    NetworkTask nTask;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        tv = (TextView) findViewById(R.id.tvText);
        tvresponse = (TextView) findViewById(R.id.tvresponse);
        etData = (EditText) findViewById(R.id.data);
        ba = (Button) findViewById(R.id.btA);
        bb = (Button) findViewById(R.id.btB);
        bsenddata = (Button) findViewById(R.id.bsenddata);
        bsendcmd = (Button) findViewById(R.id.bsendcommand);
        breceivedata = (Button) findViewById(R.id.breceivedata);
        bstart = (Button) findViewById(R.id.bstart);
        ima = (ImageView) findViewById(R.id.imageView1);
        imb = (ImageView) findViewById(R.id.ImageView2);
        ba.setOnClickListener(bledlistener);
    }
}
MainActivity.java

bb.setOnClickListener(bledlistener);
bsenddata.setOnClickListener(bsendlistener);
bsendcmd.setOnClickListener(bsendlistener);
breceivedata.setOnClickListener(bsendlistener);
bstart.setOnClickListener(bstartlistener);

spCMD = findViewById(R.id.spcommand);
spREC = findViewById(R.id.spreceivedata);
String array_spinner[] = new String[256];
for(int i = 0; i < 256; i++)
{
    array_spinner[i] = i+1 + " bytes";
}
ArrayAdapter<String> adapter = new ArrayAdapter<>((this,
android.R.layout.simple_spinner_item, array_spinner);
spREC.setAdapter(adapter);

mActivity = this;
libraryActivity = new LibraryActivity();

nTask = new NetworkTask();
nTask.setActivity(mActivity);

//Start the network task, connect to socket
private OnClickListener bstartlistener = new OnClickListener()
{
@Override
    public void onClick(View v) {
        nTask.execute();
tv.setText("Connected to GPIB connector");
bstart.setVisibility(View.INVISIBLE);
    }
};

//Send data to wifi module through the network task
private OnClickListener bsendlistener = new OnClickListener()
{
@Override
    public void onClick(View v) {
        //Send data to GPIB bus
        if(v == bsenddata)
        {
            byte data[] = StringCommandToBytes(etData.getText().toString());
            byte cmd[] = new byte[3+data.length];
cmd[0] = WRITE_CMD;
cmd[1] = DATA;
cmd[2] = (byte) data.length;
for(int i = 0; i < data.length; i++)
{
    cmd[i+3] = invertbits(data[i]);
}
nTask.SendDataToNetwork(cmd);
}
if(v == bsendcmd)
{
    if(GPIBlib != null)
    {
        //Get the value of the command selected by the user
MainActivity.java

String command = GPIBlib.get(spCMD.getSelectedItemPosition()).getValue();
//Convert it to a byte array
byte data[] = StringCommandToBytes(command);
byte cmd[] = new byte[3+data.length];
cmd[0] = WRITE_CMD;
cmd[1] = COMMAND;
cmd[2] = (byte) data.length;
for(int i = 0; i < data.length; i++)
{
    cmd[i+3] = invertbits(data[i]);
}
nTask.SendDataToNetwork(cmd);

else
{
    showToast("No library selected!");
}

if(v == breceivedata)
{
    byte cmd[] = new byte[2];
    cmd[0] = READ_CMD;
    cmd[1] = (byte) (spREC.getSelectedItemPosition()+1);
nTask.SendDataToNetwork(cmd);
}

//Inverts all the bits (0 = 7, 1 = 6, 2 = 5, etc...) to suit the hardware
private byte invertbits(byte b) {
    byte invertedByte = 0x00;
    for(int j = 0; j <= 7; j++)
    {
        invertedByte |= (((b&(1<<j))>>j)<<((7-j)));
    }
    return invertedByte;
}

//Convert the string value of a command to a byte array with the proper value
private byte[] StringCommandToBytes(String command) {
    byte[] data = new byte[command.length()/2];
    for(int i = 0; i < data.length; i++)
    {
        byte byte03 = 0;
        byte byte47 = 0;
        if((byte) command.charAt(2*i) >= 48 & (byte) command.charAt(2*i) <= 57)
            byte47 = (byte) (command.charAt(2*i) - 48);
        if((byte) command.charAt(2*i) >= 65 & (byte) command.charAt(2*i) <= 70)
            byte47 = (byte) (command.charAt(2*i) - 55);
        if((byte) command.charAt(2*i+1) >= 48 & (byte) command.charAt(2*i+1) <= 57)
            byte03 = (byte) (command.charAt(2*i+1) - 48);
        if((byte) command.charAt(2*i+1) >= 65 & (byte) command.charAt(2*i+1) <= 70)
            byte03 = (byte) (command.charAt(2*i+1) - 55);
    }
    data[i] = (byte) (byte03 + 16*byte47);
MainActivity.java

```java
return data;
```

```java
private OnClickListener bledlistener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        if (v == ba) {
            if (a == 1) {
                a = 0;
            } else {
                a = 1;
            }
        } else {
            // send LED command to wifi module
            sendLedCmd();
        }
    }
}
```

```java
// Send a LED command to wi-fi module to change the LEDs states
protected void sendLedCmd() {
    byte cmd[];
    cmd = new byte[3];
    cmd[0] = LED_CMD;
    cmd[1] = (byte) a;
    cmd[2] = (byte) b;
    nTask.SendDataToNetwork(cmd);
}
```

```java
// Data received from the Wi-Fi module
public void receivedData(byte[] data) {
    if (data[0] == RESPONSE_CMD) {
        switch(data[1]) {
            // Case it is a response from a LED command, refresh LED indicator with the real state
            // and display a text informing of the state of the LEDs
            case LED_CMD: String txt = "LED A : ";
                if(data[3] > 0) {
                    txt+= "ON";
                    ima.setImageResource(android.R.drawable.presence_online);
                } else {
                    //
```
MainActivity.java
{
    txt+= "OFF";
    ima.setImageResource
(android.R.drawable.presence_offline);
}

if(data[4] > 0)
{
    txt+= "ON";
    imb.setImageResource
(android.R.drawable.presence_online);
}
else
{
    txt+= "OFF";
    imb.setImageResource
(android.R.drawable.presence_offline);
}
showToast(txt);

//Case it is a response of a send command

 case WRITE_CMD: if(data[2] == STATUS_OK)
 {
     showToast("Bytes successfully written on GPIB");
 }
 else
 {
     showToast("GPIB timeout error!");
 }

//Case it is a response of a read command

 case READ_CMD: if(data[2] == STATUS_OK)
 {
     showToast("Received : " + data[3]);
 }
 else
 {
     showToast("GPIB timeout error!");
 }

else
{
    tv.setText("Connection problem");
}

//Used to display toast with the UI thread
public void showToast(final String message)
{
    this.runOnUiThread(new Runnable() {
        public void run()
        {
            Toast.makeText(mActivity.getApplicationContext(), message,
Toast.LENGTH_LONG).show();
        }
    });
}
MainActivity.java

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.action_items, menu);
    return true;
}

/*
 * (non-Javadoc)
 * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    //Open the Wifi settings to let the user connect to Wi-Fi module AP
    case R.id.wifi_enable:
        startActivity(new Intent(Settings.ACTION_WIFI_SETTINGS));
        return true;
    //Open the library manager
    case R.id.library:
        Intent library = new Intent(MainActivity.this, LibraryActivity.class);
        startActivityForResult(library, LibraryActivity.CHOOSE_FILE_RESULT_CODE);
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}

//On activity results. If user has chosen a GPIB library
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode == RESULT_OK) {
        if (requestCode == LibraryActivity.CHOOSE_FILE_RESULT_CODE) {
            librarypath = data.getStringExtra(CreateLibrary.PATH);
            if (librarypath != null) {
                GPIBlib = new ArrayList<GPIBcommand>();
                try {
                    BufferedReader br = new BufferedReader(new FileReader(new File(librarypath)));
                    String line = "";
                    while ((line = br.readLine()) != null) {
                        String arrayString[] = line.split(",\");
                        if (arrayString.length == 2) {
                            GPIBlib.add(new GPIBcommand(arrayString[0], arrayString[1]));
                        } else {
                            
Page 6
MainActivity.java

    showToast("Library corrupted!");
    return;
  }
  }
  }
  
  } catch (IOException e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
  }

  // Display the chosen library on the listview
  List<String> commandDisplay = new ArrayList<String>();
  for(int i = 0; i < GPIBlib.size(); i++)
  {
      commandDisplay.add(GPIBlib.get(i).getName());
  }
  ArrayAdapter<String> cmdList = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, commandDisplay);
  spCMD.setAdapter(cmdList);
  }
  }
  
  }
  
  }
package nathan.quinteiro.gpibconnector;
import java.io.IOException;
public class NetworkTask extends AsyncTask<Void, byte[], Boolean> {
    private MainActivity mActivity;
    private final String RAKIPaddress = "192.168.1.10";
    private final int RAKsocket = 7;
    Socket nSocket = null; //Network Socket
    InputStream nIS; //Network Input Stream
    OutputStream nOS; //Network Output Stream

    public void setActivity(MainActivity act) {
        mActivity = act;
    }

    @Override
    protected void onPreExecute() {
        Log.i("AsyncTask", "onPreExecute");
    }

    //Different thread that handle connexion and reception of message from the socket
    @Override
    protected Boolean doInBackground(Void... params) {
        boolean result = false;
        try {
            Log.i("AsyncTask", "doInBackground: Creating socket");
            SocketAddress sockaddr = new InetSocketAddress(RAKIPaddress, RAKsocket);
            nSocket = new Socket();
            nSocket.connect(sockaddr, 5000); //5 seconds connection timeout
            if (nSocket.isConnected()) {
                nIS = nSocket.getInputStream();
                nOS = nSocket.getOutputStream();
                Log.i("AsyncTask", "doInBackground: Socket created, streams assigned");
                //Display that connection to socket is OK
                mActivity.showToast(mActivity.getString(R.string.connec_ok));

                //Wait to receive data
                byte[] buffer = new byte[4096];
                int read = nIS.read(buffer, 0, 4096);
                //When data received, publish data and send it to main activity
                while (read != -1) {
                    byte[] tempdata = new byte[read];
                    System.arraycopy(buffer, 0, tempdata, 0, read);
                    publishProgress(tempdata);
                    Log.i("AsyncTask", "doInBackground: Got some data");
                    read = nIS.read(buffer, 0, 4096); //This is blocking
                    Log.i("Data = ", buffer.toString());
                }
            } catch (IOException e) {
                e.printStackTrace();
                Log.i("AsyncTask", "doInBackground: IOException");
                result = true;
            } catch (Exception e) {
                e.printStackTrace();
                result = true;
            }
        }
        return result;
    }

    //Different thread that handle connexion and reception of message from the socket
    @Override
    protected void onPostExecute(Boolean result) {
        if (result) {
            //Process the received data
        } else {
            //Handle the IOException and publishProgress
        }
    }
}

//NetworkTask.java
NetworkTask.java

Log.i("AsyncTask", "doInBackground: Exception");
result = true;
} finally {
    try {
        nIS.close();
        nOS.close();
        nSocket.close();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
    Log.i("AsyncTask", "doInBackground: Finished");
} 
return result;

//Send the data to the GPIB Wi-Fi module
public void SendDataToNetwork(byte[] cmd) {
    if(nSocket != null) {
        if(nSocket.isConnected()) {
            try {
                if (nSocket.isConnected()) {
                    Log.i("AsyncTask", "SendDataToNetwork: Writing received message to socket");
                    nOS.write(cmd);
                } else {
                    Log.i("AsyncTask", "SendDataToNetwork: Cannot send message. Socket is closed");
                }
            } catch (Exception e) {
                Log.i("AsyncTask", "SendDataToNetwork: Message send failed. Caught an exception");
            }
        } else {
            mActivity.showToast("Connection not established with GPIB connector");
        }
    } else {
        mActivity.showToast("Connection not established with GPIB connector");
    }
}

//When data are received from Wi-Fi module
@Override
protected void onProgressUpdate(byte[]... values) {
    if (values.length == 1) {
        Log.i("AsyncTask", "onProgressUpdate: " + values[0].length + " bytes received.");
        //Send string value to main activity
        mActivity.receivedData(values[0]);
    }
@Override
protected void onCancelled() {
    Log.i("AsyncTask", "Cancelled.");
}

@Override
protected void onPostExecute(Boolean result) {
    if (result) {
        Log.i("AsyncTask", "onPostExecute: Completed with an Error.");
    } else {
        Log.i("AsyncTask", "onPostExecute: Completed.");
    }
}
LibraryActivity.java

package nathan.quinteiro.gpibconnector;

import java.io.BufferedReader;

public class LibraryActivity extends ListActivity {

    Button bNew;
    Button bChose;
    Button bValidate;
    TextView tvPath;
    String path;
    List<GPIBcommand> GPIBlib = null;

    protected static final int CHOOSE_FILE_RESULT_CODE = 20;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_library);

        bNew = (Button) findViewById(R.id.bNewLibrary);
        bNew.setOnClickListener(bnewlistener);
        bChose = (Button) findViewById(R.id.bChoseLibrary);
        bChose.setOnClickListener(bchoselistener);
        bValidate = (Button) findViewById(R.id.bValidate);
        bValidate.setOnClickListener(bvalidatelistener);

        tvPath = (TextView) findViewById(R.id.tvLibrarymanager);
    }

    private OnClickListener bnewlistener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent newlibrary = new Intent(LibraryActivity.this, CreateLibrary.class);
            startActivity(newlibrary);
        }
    };

    private OnClickListener bchoselistener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(LibraryActivity.this, ChoseLibrary.class);
            startActivityForResult(intent, CHOOSE_FILE_RESULT_CODE);
        }
    };

    private OnClickListener bvalidatelistener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent result = new Intent();
            result.putExtra(CreateLibrary.PATH, path);
            setResult(RESULT_OK, result);
            finish();
        }
    };
}

Page 1
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode == RESULT_OK) {
        if (requestCode == CHOOSE_FILE_RESULT_CODE) {
            path = data.getStringExtra(CreateLibrary.PATH);
            tvPath.setText(path);
            if (path != null) {
                GPIBlib = new ArrayList<GPIBcommand>();
                try {
                    BufferedReader br = new BufferedReader(new FileReader(new File(path)));
                    String line = "";
                    while ((line = br.readLine()) != null) {
                        String arrayString[] = line.split(",");
                        if (arrayString.length == 2) {
                            GPIBlib.add(new GPIBcommand(arrayString[0], arrayString[1]));
                        } else {
                            showToast("Library corrupted!");
                            List<String> commandDisplay = new ArrayList<String>();
                            ArrayAdapter<String> cmdList = new ArrayAdapter<String>(getBaseContext(), R.layout.rowcommand, commandDisplay);
                            setListAdapter(cmdList);
                            return;
                        }
                    }
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        } else {
            showToast("Library corrupted!");
            List<String> commandDisplay = new ArrayList<String>();
            for (int i = 0; i < GPIBlib.size(); i++) {
                commandDisplay.add(GPIBlib.get(i).toString());
            }
            ArrayAdapter<String> cmdList = new ArrayAdapter<String>(getBaseContext(), R.layout.rowcommand, commandDisplay);
            setListAdapter(cmdList);
        }
    }
}

//Used to display toast with the UI thread

public void showToast(final String message)
{
    Toast.makeText(this.getApplicationContext(), message, Toast.LENGTH_LONG).show();
}
}
package nathan.quinteiro.gpibconnector;

import java.io.File;

public class CreateLibrary extends ListActivity {

    public static final int CHOOSE_FILE_RESULT_CODE = 20;
    public static final String PATH = "nathan.quinteiro.GPIB.path";

    private List<GPIBcommand> commands = new ArrayList<GPIBcommand>();

    private String path = "";

    Button bsave;
    Button bchose;
    Button bsavem; 
    TextView tvPath;
    TextView tvResp;
    EditText etLibName;
    EditText etCmdName;
    EditText etCmd;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_library);

        bsave = (Button) findViewById(R.id.bSave);
        bsave.setOnClickListener(bsavelistener);
        bchose = (Button) findViewById(R.id.bChose);
        bchose.setOnClickListener(bchoselistener);
        bsavem = (Button) findViewById(R.id.bsavem);
        bsavem.setOnClickListener(bsavemlistener);

        tvPath = (TextView) findViewById(R.id.pathtosave);
        etLibName = (EditText) findViewById(R.id.libraryname);
        etCmdName = (EditText) findViewById(R.id.edCmdName);
        etCmd = (EditText) findViewById(R.id.edCmd);
    }

    private OnClickListener bchoselistener = new OnClickListener() {
        //Start activity to choose directory to save library
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(CreateLibrary.this, ChoseFolder.class);
            startActivityForResult(intent, CHOOSE_FILE_RESULT_CODE);
        }
    }

    private OnClickListener bsavelistener = new OnClickListener() {
        //Save the library in the chosen directory
        if(path!=null && !etLibName.getText().toString().equals("") )
        {
            try {

            }
        }
    }
}

CreateLibrary.java

File dir = new File(path);
File myFile = new File(dir, etLibName.getText().toString() + ".glb");
tvPath.setText(myFile.toURI().toString());
if(!myFile.createNewFile())
{
    myFile.delete();
    if(!myFile.createNewFile())
    {
        showToast("Error system - library already exists and can't be overwrite");
        return;
    }
    else
    {
        showToast("Library modified");
    }
}
FileOutputStream fOut = new FileOutputStream(myFile);
OutputStreamWriter myOutWriter = new OutputStreamWriter(fOut);
for(int i = 0; i < commands.size(); i++)
{
    myOutWriter.append(commands.get(i).toFile());
}
myOutWriter.close();
fOut.close();
showToast("Library saved!");
}
else
{
    showToast("Choose correct path");
}

private OnClickListener bsavecmdlistener = new OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        //if user chosed a command name and put values
        if(!etCmdName.getText().toString().equals("") && !etCmd.getText().toString().equals(""))
        {
            for(int i = 0; i < commands.size(); i++)
            {
                if(commands.get(i).name.equals(etCmdName.getText().toString()))
                {
                    showToast("Commands name already used");
                    return;
                }
            }
            //Save command
            commands.add(new GPIBcommand(etCmdName.getText().toString(),
etCmd.getText().toString()));
            //Display the current library on the listview
        }
    }
}
CreateLibrary.java

List<String> commandDisplay = new ArrayList<String>();
for(int i = 0; i < commands.size(); i++)
{
    commandDisplay.add(commands.get(i).toString());
}
ArrayAdapter<String> cmdList = new ArrayAdapter<String>(getBaseContext(), R.layout.rowcommand, commandDisplay);
setListAdapter(cmdList);

//No command name or no value
else
{
    showToast("Please chose a name and a value for the command");
}
};

@Override
protected void onListItemClick(ListView l, View v, int position, long id)
{
    //Remove commands when user click on it
    commands.remove(position);
    //Display the current library on the listview
    List<String> commandDisplay = new ArrayList<String>();
    for(int i = 0; i < commands.size(); i++)
    {
        commandDisplay.add(commands.get(i).toString());
    }
    ArrayAdapter<String> cmdList = new ArrayAdapter<String>(getBaseContext(), R.layout.rowcommand, commandDisplay);
    setListAdapter(cmdList);
}

//Used to display toast with the UI thread
public void showToast(final String message)
{
    Toast.makeText(this.getApplicationContext(), message, Toast.LENGTH_LONG).show();
}

//On activity results. If user has chosen a folder where to save the library
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if(resultCode == RESULT_OK)
    {
        if(requestCode == CHOOSE_FILE_RESULT_CODE)
        {
            path = data.getStringExtra(PATH);
            tvPath.setText(path);
        }
    }
}
ChoseLibrary.java

package nathan.quinteiro.gpibconnector;

import java.io.File;

public class ChoseLibrary extends ListActivity {

    private List<String> item = null;
    private List<String> path = null;
    private String root;
    private TextView myPath;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_chose_folder);
        myPath = (TextView)findViewById(R.id.path);
        root = Environment.getExternalStorageDirectory().getPath();
        getDir(root);
    }

    private void getDir(String dirPath) {
        myPath.setText("Location: " + dirPath);
        item = new ArrayList<String>();
        path = new ArrayList<String>();
        File f = new File(dirPath);
        File[] files = f.listFiles();
        if(!dirPath.equals(root)) {
            item.add(root);
            path.add(root);
            item.add("../");
            path.add(f.getParent());
        }
        for(int i=0; i < files.length; i++) {
            File file = files[i];
            if(!file.isHidden() && file.canRead()){
                path.add(file.getPath());
                if(file.isDirectory()){
                    item.add(file.getName() + "/");
                }else{
                    item.add(file.getName());
                }
            }
        }

        ArrayAdapter<String> fileList =
            new ArrayAdapter<String>(this, R.layout.row, item);
        setListAdapter(fileList); 
    }
}
ChoseLibrary.java

@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
// TODO Auto-generated method stub
    File file = new File(path.get(position));

    if (file.isDirectory())
        {
            if(file.canRead()){
                getDir(path.get(position));
            }else
            {
                new AlertDialog.Builder(this)
                    .setIcon(R.drawable.ic_launcher)
                    .setTitle("[" + file.getName() + "] folder can't be read!")
                    .setPositiveButton("OK", null).show();
            }
        }else
        {
            if(file.getName().endsWith(".glb"))
            {
                Intent result = new Intent();
                result.putExtra(CreateLibrary.PATH, path.get(position));
                setResult(RESULT_OK, result);
                finish();
            }else
            {
                new AlertDialog.Builder(this)
                    .setIcon(R.drawable.ic_launcher)
                    .setTitle("Incompatible file! Chose a correct GPIB library (.glb)"")
                    .setPositiveButton("OK", null).show();
            }
        }
}
package nathan.quinteiro.gpibconnector;
import java.io.File;
public class ChoseFolder extends ListActivity {

private List<String> item = null;
private List<String> path = null;
private String root;
private TextView myPath;
private Button bChose;
private String chosedPath;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_chose_folder);
    myPath = (TextView) findViewById(R.id.path);
    root = Environment.getExternalStorageDirectory().getPath();
    getDir(root);
    bChose = (Button) findViewById(R.id.bChosed);
    bChose.setOnClickListener(bchosedlistener);
}

private void getDir(String dirPath) {
    myPath.setText("Location: " + dirPath);
    item = new ArrayList<String>();
    path = new ArrayList<String>();
    File f = new File(dirPath);
    File[] files = f.listFiles();
    if(!dirPath.equals(root))
    {
        item.add(root);
        path.add(root);
        item.add("../");
        path.add(f.getParent());
    }
    for(int i=0; i < files.length; i++)
    {
        File file = files[i];
        if(!file.isHidden() && file.canRead()){
            path.add(file.getPath());
            if(file.isDirectory()){
                item.add(file.getName() + "/");
            }else{
                item.add(file.getName());
            }
        }
    }
    ArrayAdapter<String> fileList =
ChoseFolder.java

    new ArrayAdapter<String>(this, R.layout.row, item);
    setListAdapter(fileList);
}

@Override
    protected void onListItemClick(ListView l, View v, int position, long id) {
        // TODO Auto-generated method stub
        File file = new File(path.get(position));
        if (file.isDirectory())
            if (file.canRead()){
                chosedPath = path.get(position);
                getDir(path.get(position));
            }
        else
            {  
                new AlertDialog.Builder(this)
                    .setIcon(R.drawable.ic_launcher)
                    .setTitle("[" + file.getName() + "] folder can't be read!")
                    .setPositiveButton("OK", null).show();
            }
        else
            {  
                new AlertDialog.Builder(this)
                    .setIcon(R.drawable.ic_launcher)
                    .setTitle("[") + file.getName() + "]")
                    .setPositiveButton("OK", null).show();
            }
    }

private OnClickListener bchosedlistener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent result = new Intent();
        result.putExtra(CreateLibrary.PATH, chosedPath);
        setResult(RESULT_OK, result);
        finish();
    }
};
package nathan.quinteiro.gpibconnector;
import java.util.Locale;

public class GPIBcommand {
    public String name;
    public String value;

    public GPIBcommand(String n, String v)
    {
        name = n.toUpperCase(Locale.ENGLISH);
        value = v.toUpperCase(Locale.ENGLISH);
    }

    public String getName()
    {
        return name;
    }

    public String getValue()
    {
        return value;
    }

    public String toString()
    {
        return ("cmd : " + name + ", value : 0x" + value);
    }

    public String toFile()
    {
        return (name + "," + value + "\n");
    }
}