Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

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

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2013/14**	No TD / *Nr. DA* **it/2014/70**
Mandant / *Auftraggeber* ☐ HES—SO Valais ☐ Industrie ☒ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Nathan Quinteiro** Professeur / *Dozent* **Pierre-André Mudry**	Lieu d'exécution / *Ausführungsort* ☐ HES—SO Valais ☐ Industrie ☒ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte* (données complètes) **Prof. Lixin Ran, Zhejiang University**	

Titre / *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.

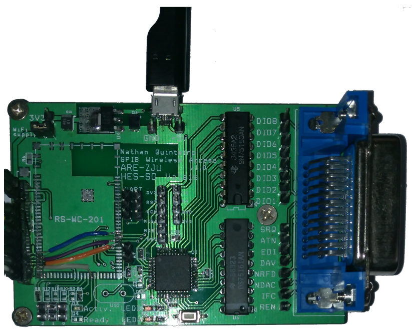
This work involves the development of a wireless communication interface between a smartphone and a GPIB bus. A hardware circuit will be connected on the GPIB bus and will provide the physical interface. A smartphone application must be develop to allow communication with the hardware device. The system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

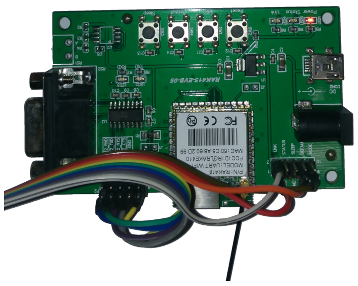
Objectives:
- Design, manufacture and debug a small hardware to provide the physical interface between a Smartphone and a GPIB bus.
- Develop a software for the MCU able to provide the software interface between a Smartphone and a GPIB bus.
- Develop a smartphone application able to connect on the hardware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

Signature ou visa / *Unterschrift oder Visum* Responsable de l'orientation *Leiter der Vertiefungsrichtung:*... [1] Etudiant / Student : [illegible]	Délais / *Termine* Attribution du thème / *Ausgabe des Auftrags:* 15.05.2014 Remise du rapport / *Abgabe des Schlussberichts:* 05.09.2014 Défense orale / *Mündliche Verfechtung:* Semaine | Woche 37

[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.

Rapport reçu le / *Schlussbericht erhalten am* …….….……. Visa du secrétariat / *Visum des Sekretariats* …………..





Bachelor's Thesis

| 2014 |

Degree course

Systems Engineering

Field of application

Infotronic

Supervising professor

Dr. Mudry Pierre-André

pandre.mudry@hevs.ch

Partner

Zhejiang University

Wirelessly accessing instruments with GPIB/WLAN

Graduate Quinteiro Nathan

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.

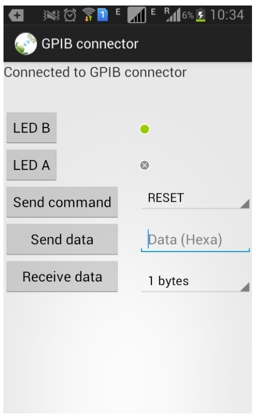
Methods | Experiences | Results

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.

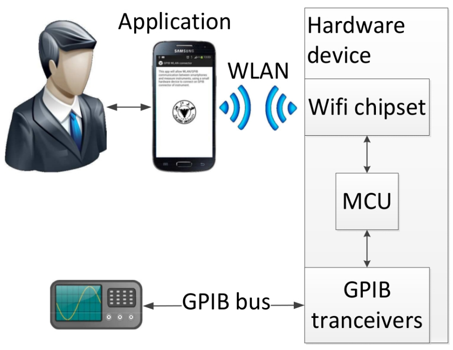


10:34

GPIB connector

Connected to GPIB connector

The Android application allowing the user to connect on the hardware and send command to the hardware in order to read or write on the GPIB bus.



Bloc diagram of the system explaining the interface. The smartphone application allow the user to write or read on the GPIB bus through the developed hardware circuit.

Contents

List of Figures

Acronyms

These different acronyms are used in this document for simplification.

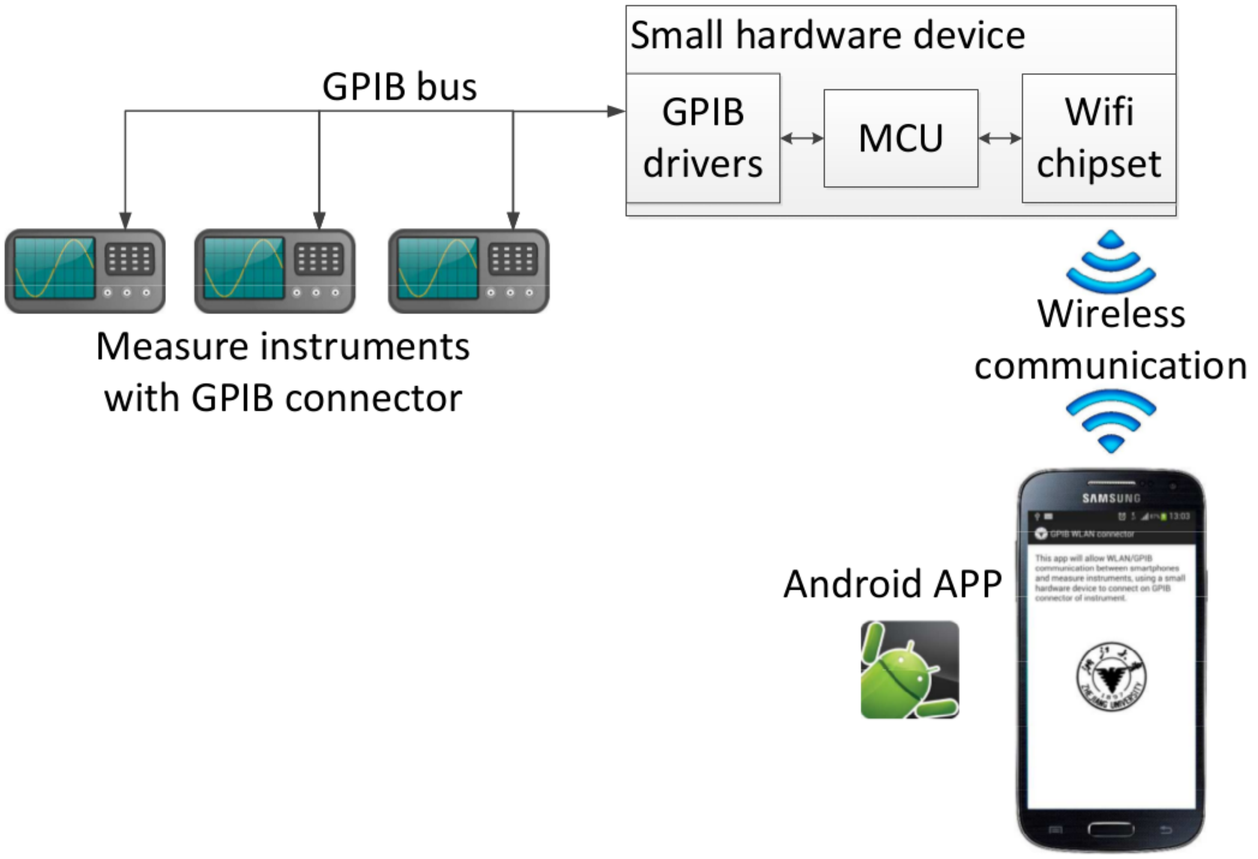
ADT	Android Developer Tools
AP	Access Point
API	Application Programming Interface
APP	Application
DHCP	Dynamic Host Configuration Protocol
DIO	Data Input/Output
GO	Group Owner - *The master of Wi-Fi direct connexion*
GPIB	General Purpose Interface Bus - *also called IEE-488*
GPIO	General Purpose Input/Output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

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.



This app will allow WLAN/GPIB communication between smartphones and measure instruments, using a small hardware device to connect on GPIB connector of instrument.

Figure 1: Explaining diagram of the project

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 align-ment 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

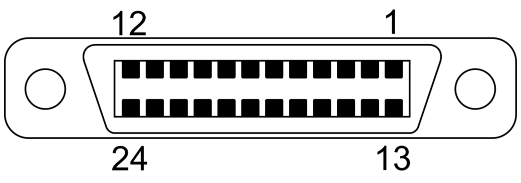


Figure 2: GPIB female connector

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] :

Single Type	**Voltage Value**
Input Voltage High	V$_{IH}$ = 3.4V typical, 2.0V minimum
Input Voltage Low	V$_{IL}$ = 0.22V typical, 0.8V maximum
Output Voltage High	V$_{OH}$ = 3.4V typical, 2.4V minimum
Output Voltage Low	V$_{OL}$ = 0.22V typical, 0.5V maximum
Maximum Voltage	V$_{Max}$ = 5.25V

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 listener 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]

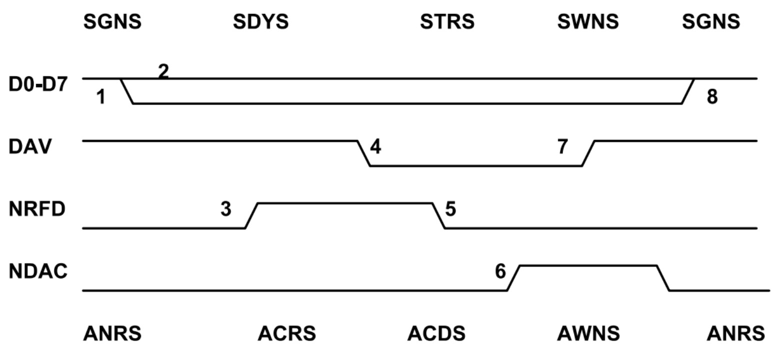


Figure 3: GPIB - Handshake timing sequence

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]

[1] The 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**	
TE	**PE**	**Direction**	**Bus Port Configuration**
H	H	T	Totem-Pole Output
H	L	T	Open Collector Output
L	X	R	Input

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**								
TE	**DC**	**ATN***		**EOI**	**REN**	**IFC**	**SRQ**	**NRFD**	**NDAC**	**DAV**
H	H		R		R	R	T	R	R	T
H	L		T		T	T	R	R	R	T
L	H		R		R	R	T	T	T	R
L	L		T		T	T	R	T	T	R
H	X	H		T						
L	X	H		R						
X	H	L		R						
X	L	L		T						

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]

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

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], wich 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 [2]. It can provide a current up to 800 mA [10].

Thermal relief :

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

$\Theta_{JA} = 136°C/W$

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

$P_{Dmax} = (V_{inmax} - V_{out}) * I_{Lmax} = (5.2 - 3.3) * 0.4 = 760mW$

Where V$_{inmax}$ is the maximal input voltage of the USB power supply, and I$_{Lmax}$ 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 (Θ_{JA}) is calculated as follow :

$\Theta_{JA} = TR_{max}/P_{Dmax} = 100/0.760 = 131°C/W$

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 is 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.

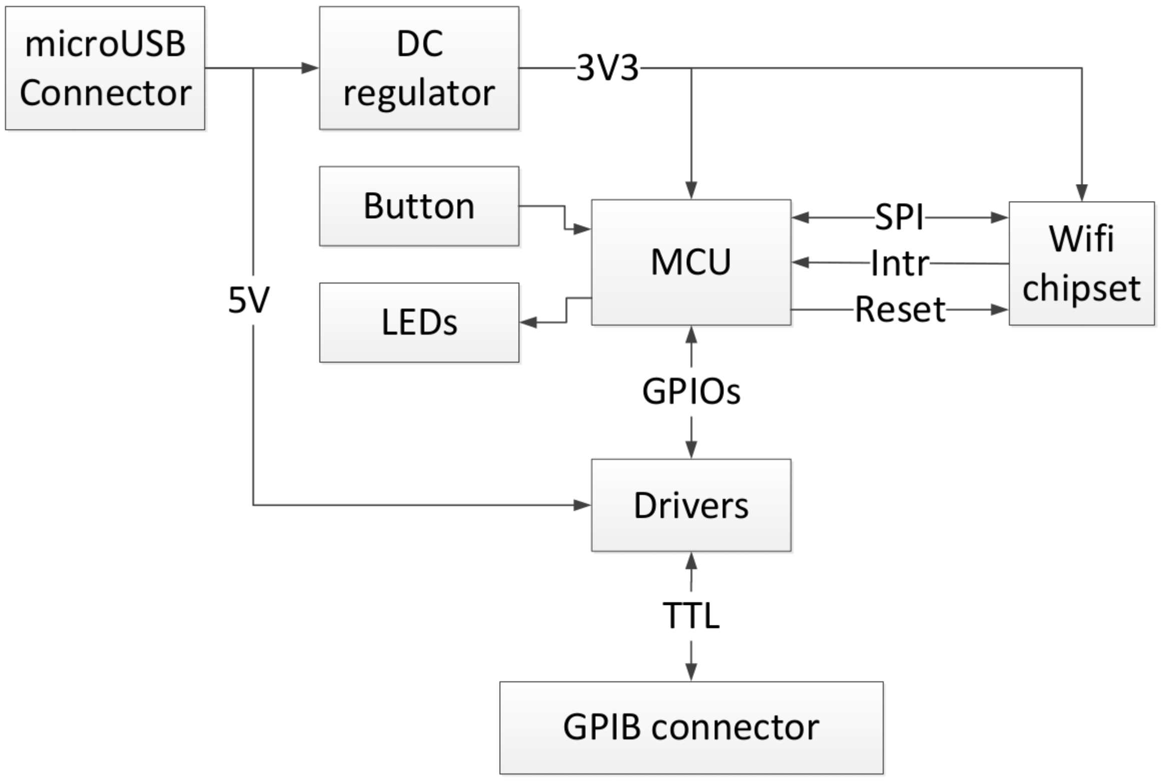


Figure 6: Hardware circuit diagram

The schematics is developed with **OrCAD Capture**.

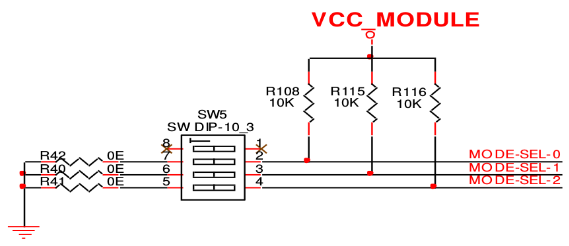
The complete schematics of the circuit can be found in appendix [11].

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.



MODE	MODE-SEL-2	MODE-SEL-1	MODE-SEL-0
UART	0	0	0
SPI	0	0	1
USB	0	1	0

Figure 7: Mode selection schematics

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.

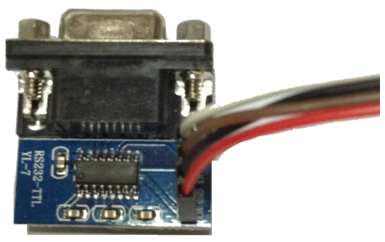


Figure 8: Extern module with MAX3232 and DB9 for firmware upgrade

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. [3]

4.2.1 Integration on the existing PCB

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

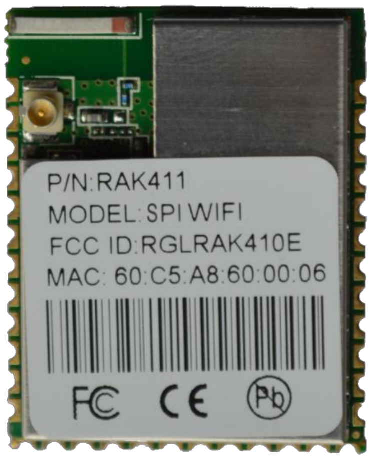
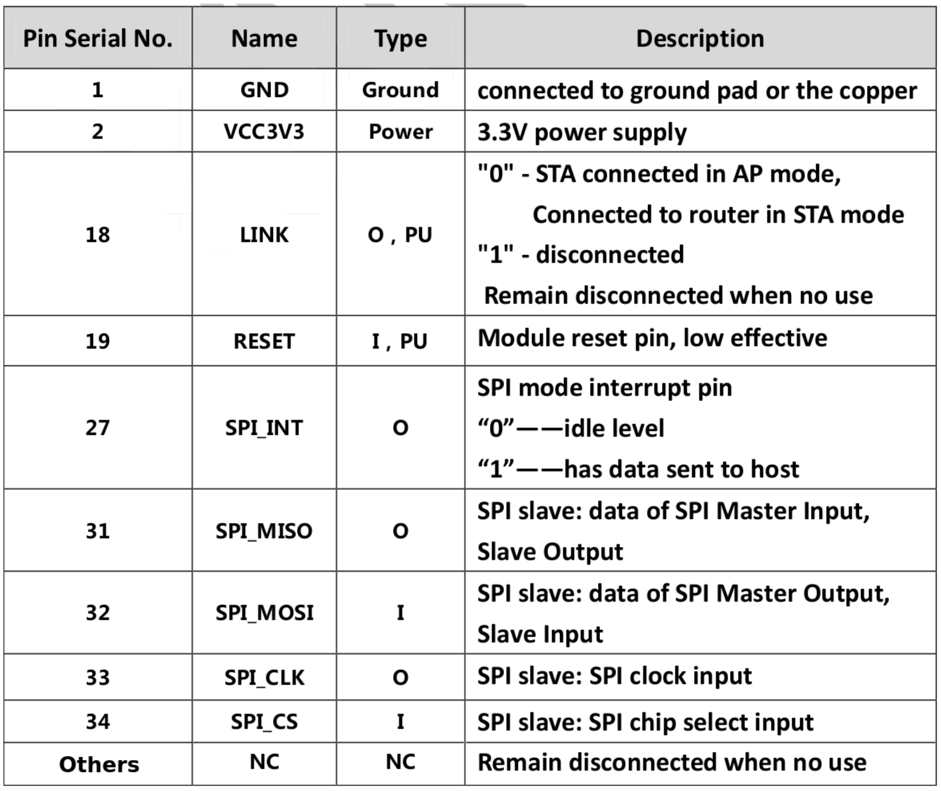


Figure 9: RAK411



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

Figure 10: RAK411 - Pins definition

[3] See on chapter 9.1.2

A evaluation board of the RAK411 is available.

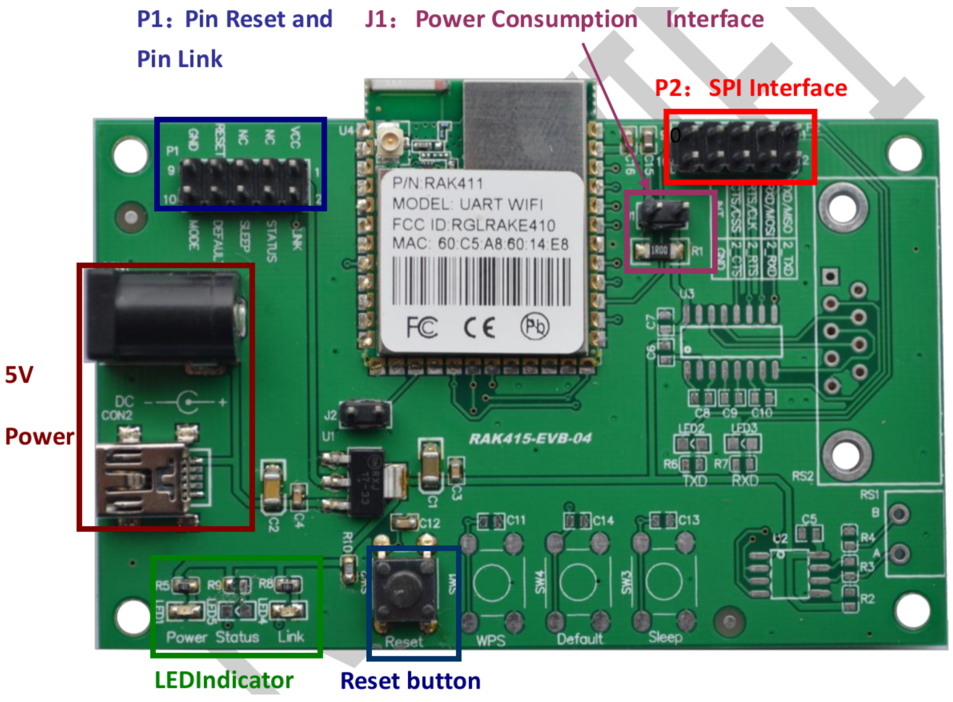


Figure 11: RAK411 - Evaluation board

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 $\overline{\text{RESET}}$ 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).

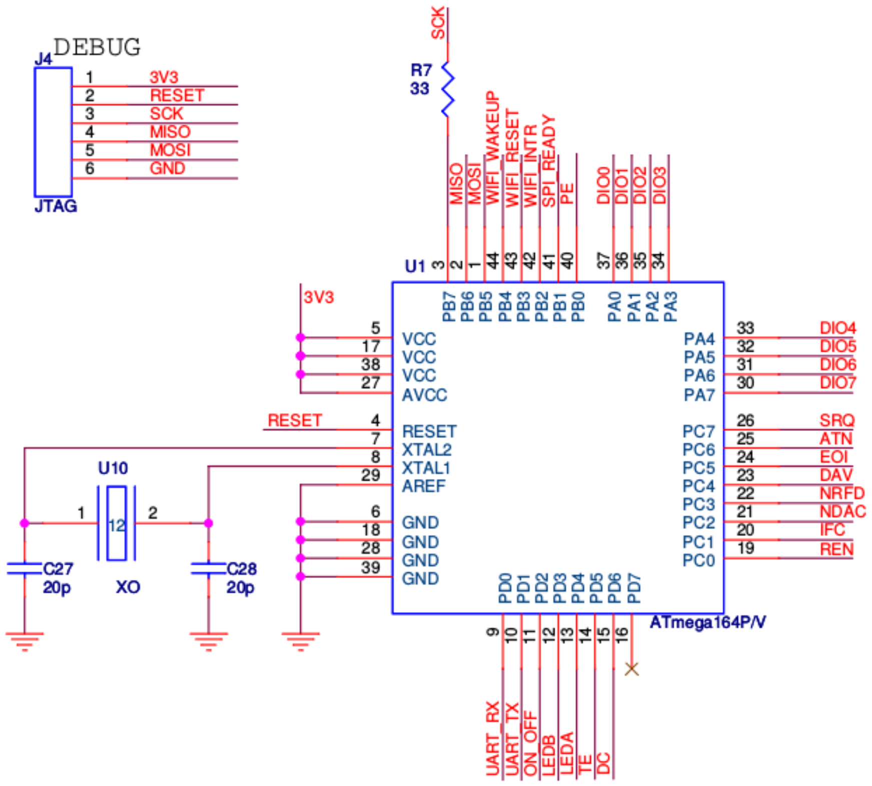


Figure 12: MCU schematics with header for programming

4.3.2 Clock

The speed grade of the ATmega164PA depends of the power supply voltage, as shown on the figure bellow. [6, p. 336]

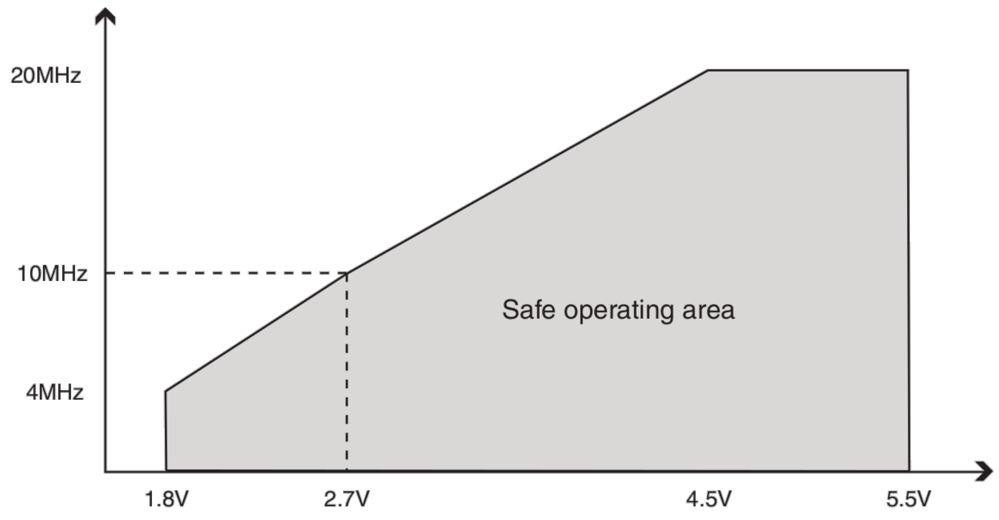


Figure 13: Maximum frequency vs Vcc

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.

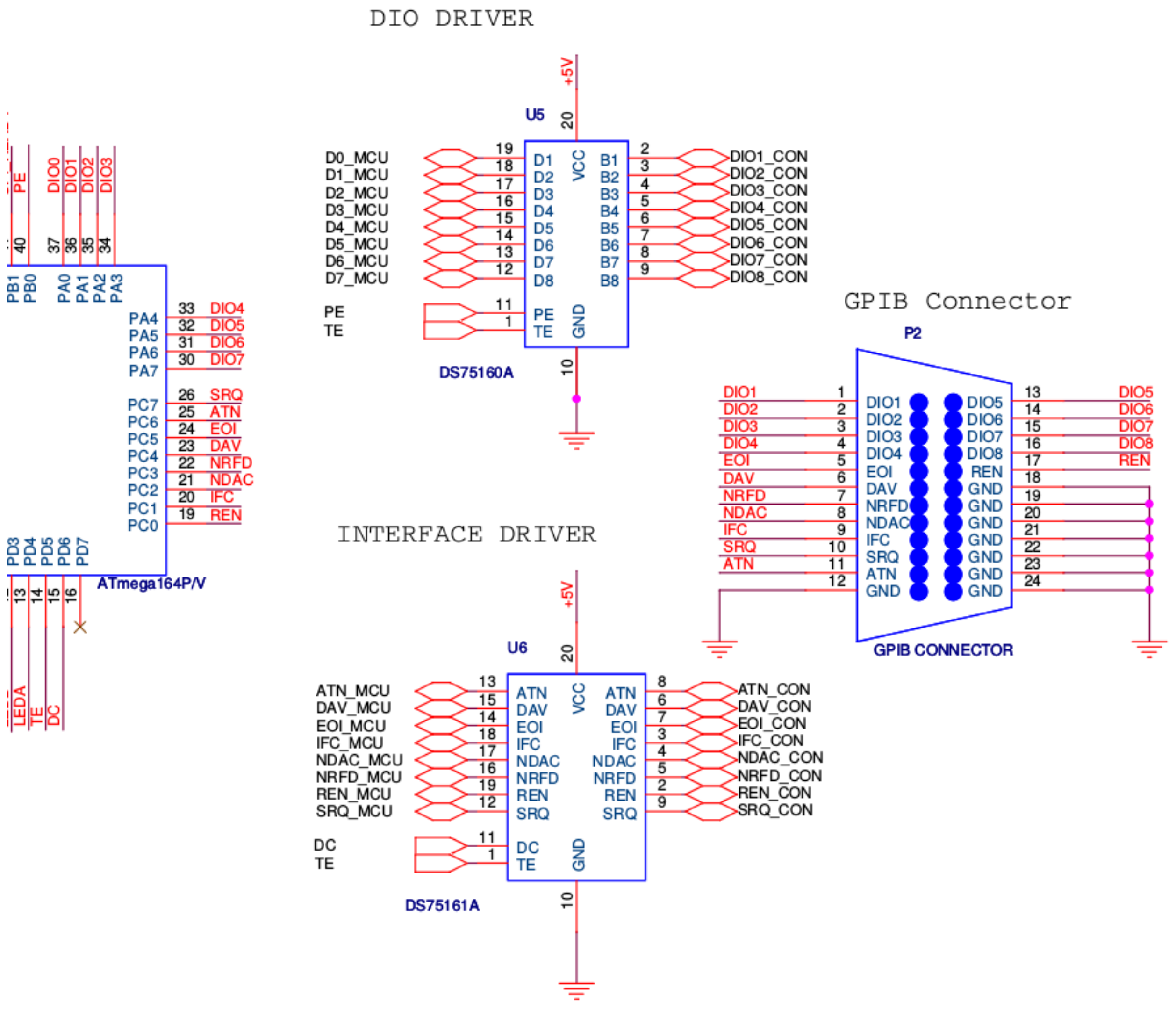


Figure 14: GPIB schematics

The **TE** pin (Talk Enable) of the 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 Θ_{JA} under 131°C/W.

With a 0.2 in^2 (1.29 cm^2) copper area on the bottom, the Θ_{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.

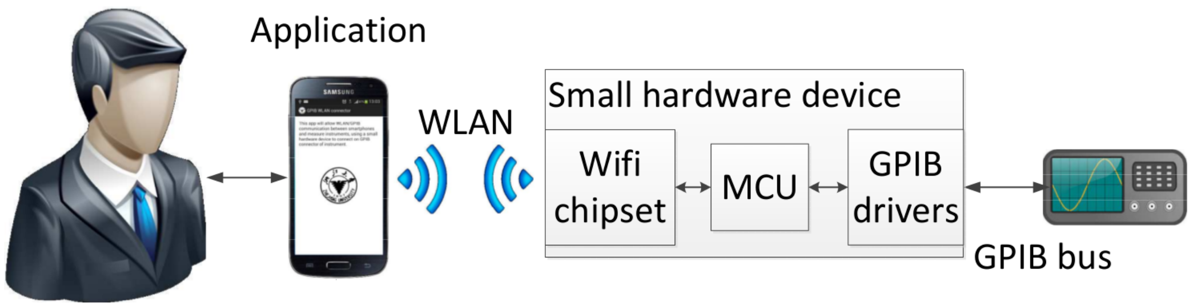


Figure 15: Interface provided

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 instruments.

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.

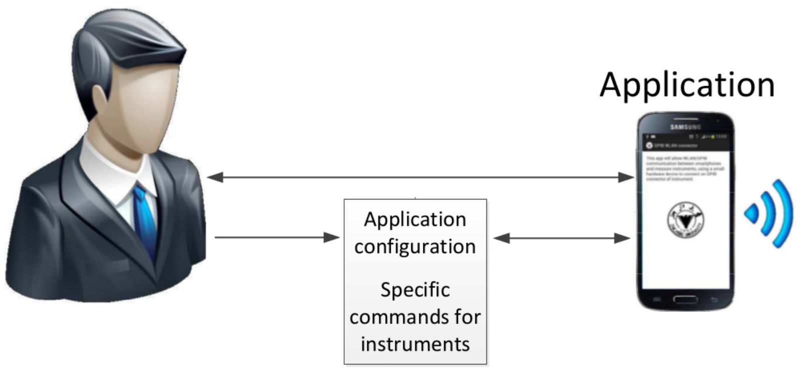


Figure 16: Instrument configuration

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 tranmission 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.

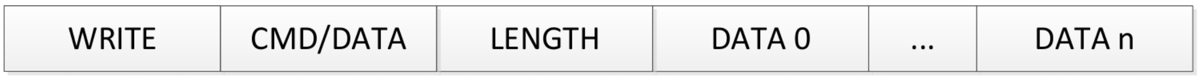


Figure 17: Write command packet

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.

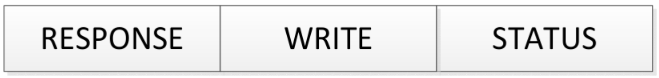


Figure 18: Write response

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.

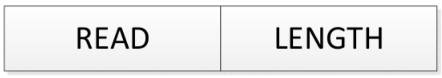


Figure 19: Read command

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.

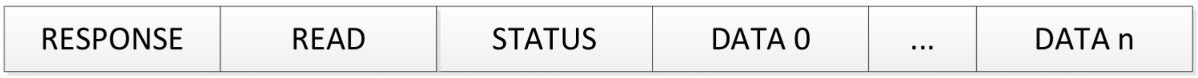


Figure 20: Write response

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.

LED	LEDA	LEDB

Figure 21: Set led command

The paquet 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.

RESPONSE	LED	STATUS	LEDA	LEDB

Figure 22: Codes and status values

6.2.4 Paquets codes and status

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

WRITE	0x70
READ	0x71
RESPONSE	0x72
LED	0x73
CMD	0x74
DATA	0x75
STATUS	0x00 = OK, 0x01 = GPIB err, 0x02 = packet err

Figure 23: Set led command

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** [4]. 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 :
$T_{timer} = \frac{N \times (1+OCRnx)}{f_{clk}}$
Where N represents the prescale factor (1, 8, 64, 256 or 1024). and OCRnx 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 :
$OCR1x = \frac{T_{timer} \times f_{clk}}{N} - 1 = \frac{10 \times 10^{-3} \times 8 \times 10^{6}}{8} - 1 = 9999 = 0x270F$

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_{osc}/2 = **4MHz** (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 **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.

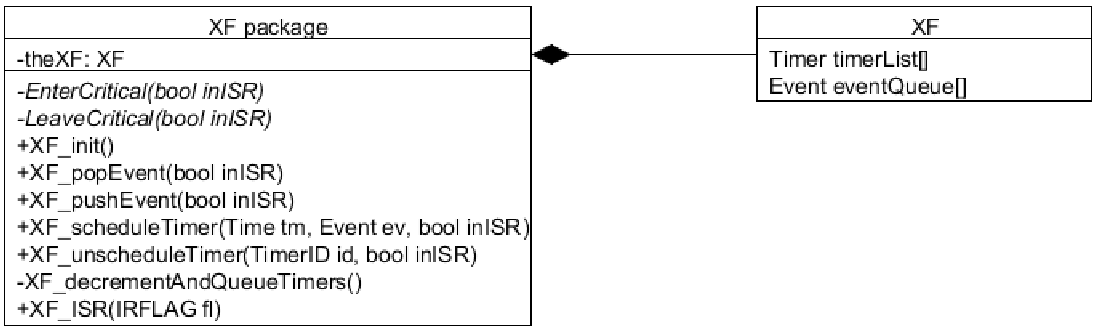

XF package
-theXF: XF
-EnterCritical(bool inISR)
-LeaveCritical(bool inISR)
+XF_init()
+XF_popEvent(bool inISR)
+XF_pushEvent(bool inISR)
+XF_scheduleTimer(Time tm, Event ev, bool inISR)
+XF_unscheduleTimer(TimerID id, bool inISR)
-XF_decrementAndQueueTimers()
+XF_ISR(IRFLAG fl)
XF
Timer timerList[]
Event eventQueue[]

Figure 24: Class diagram : XF

The XF contains a queue of events. The function **XF_ popEvent** allow the state machine to get the event that happened and treat it. Events are put in the queue using **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, **XF_ ISR** is called and the software timers, scheduled with **XF_ scheduleTimer** are decremented. Timer can also be unscheduled with **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 descibed 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.

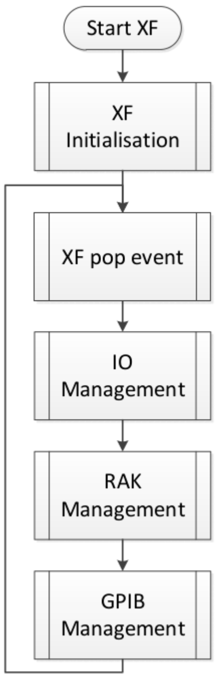


Figure 25: Main states machine

As shown on the figure 25, after the initialization, the program enters in a 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. [5]

The manufacturer provides an API for the developper, 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.

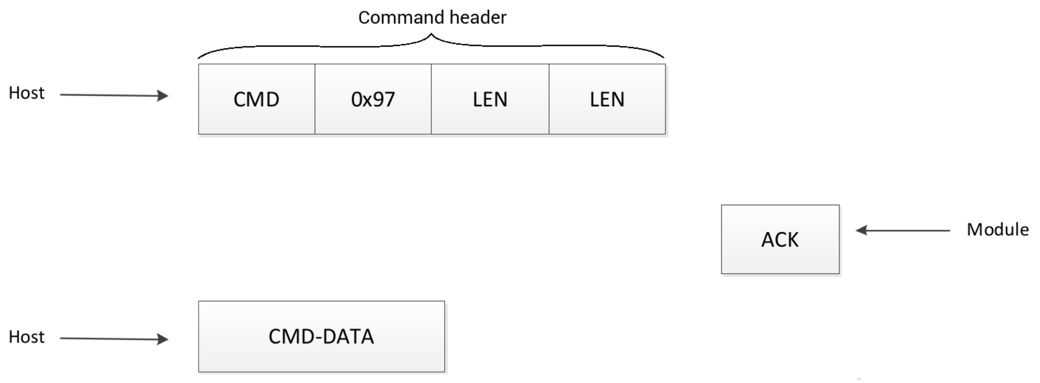


Figure 26: Sending command to RAK

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.

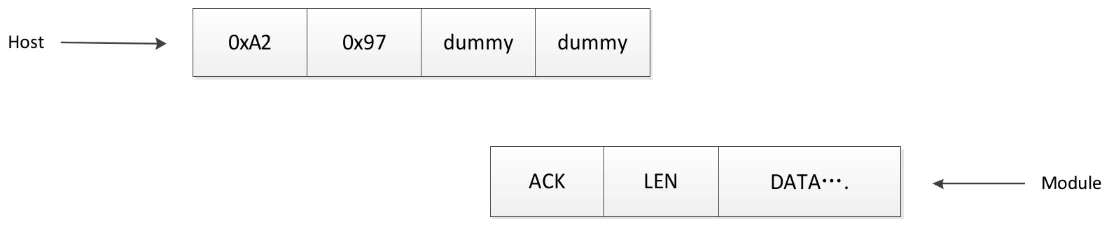


Figure 27: Receiving response values from RAK

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.

```
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.

```
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 :

```
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.

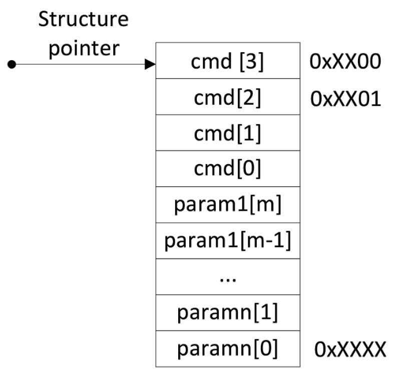


Figure 31: Structure command on SRAM

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.

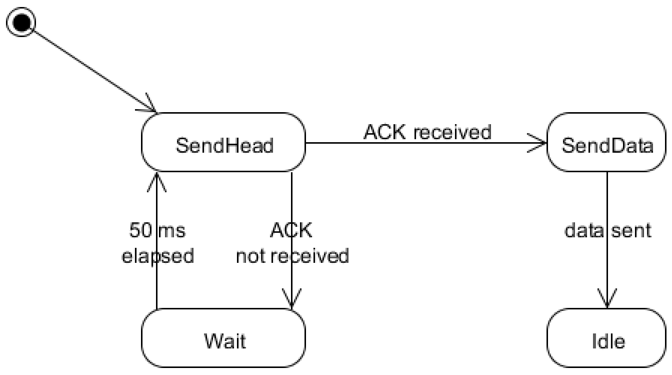


Figure 32: Send command states machine

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.

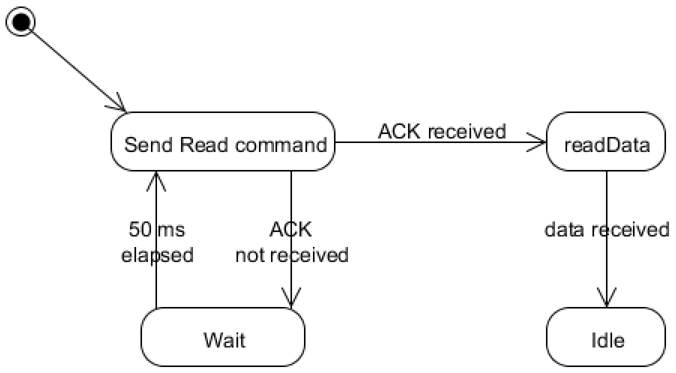


Figure 33: Receive command states machine

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.

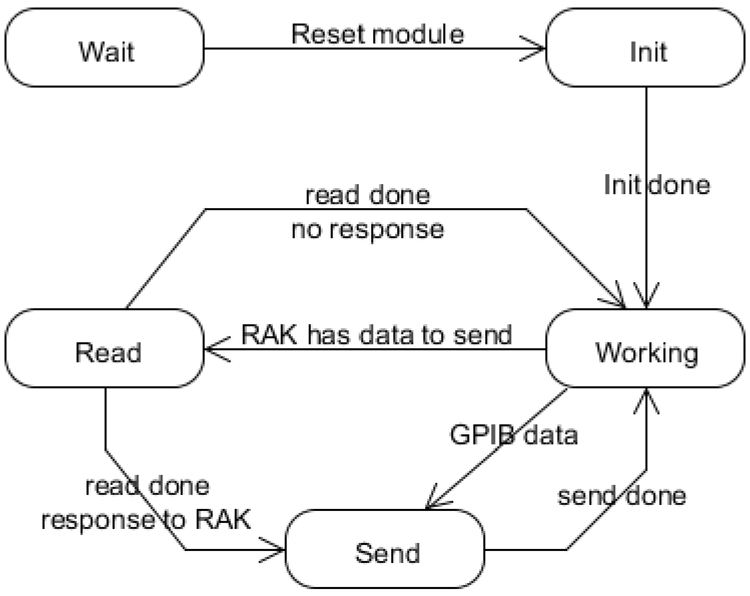


Figure 34: RAK states machine

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.

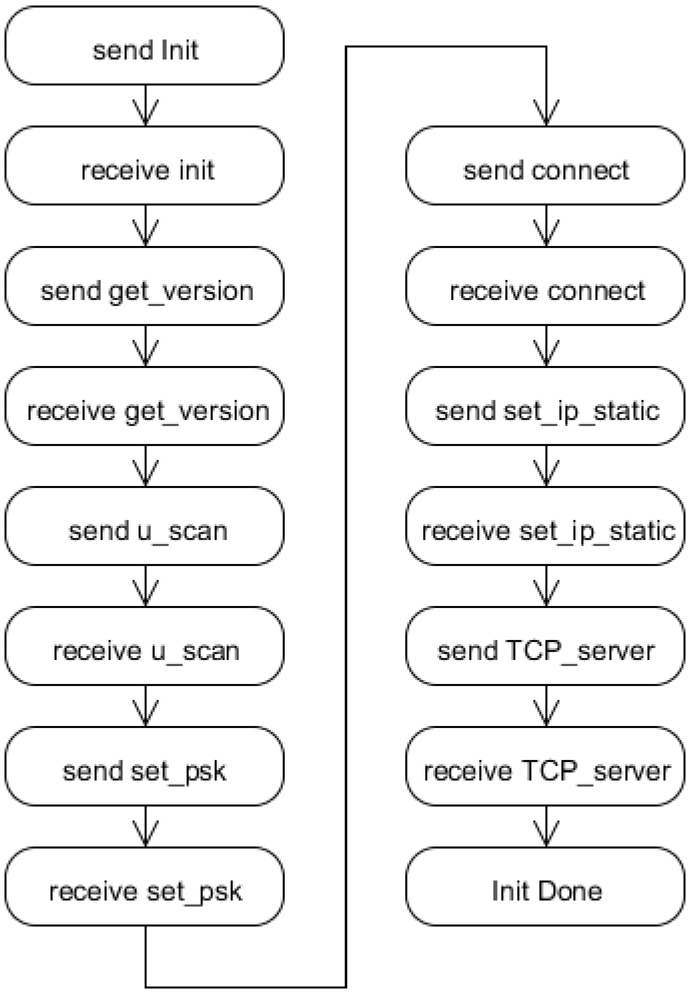


Figure 35: RAK initialization states machine

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.

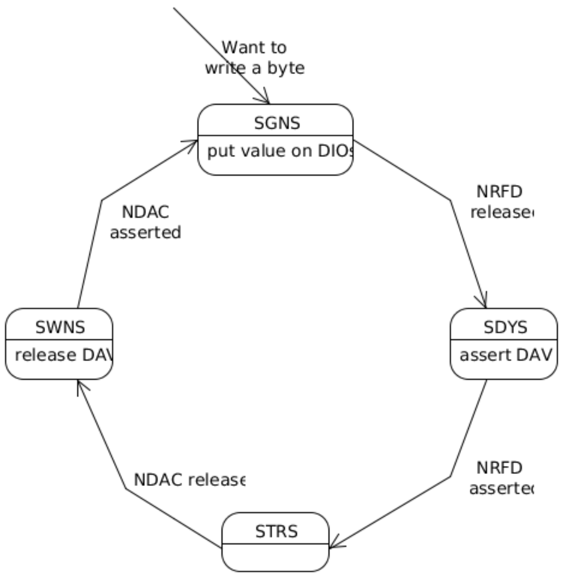


Figure 36: GPIB send states machine

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.

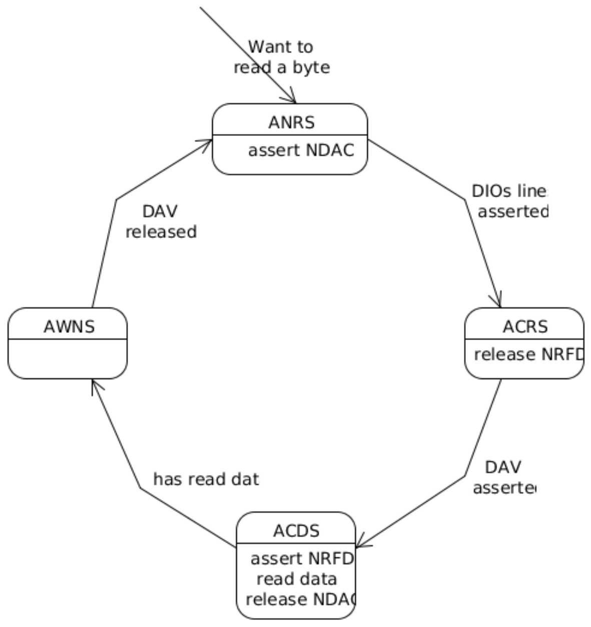


Figure 37: GPIB receive states machine

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.

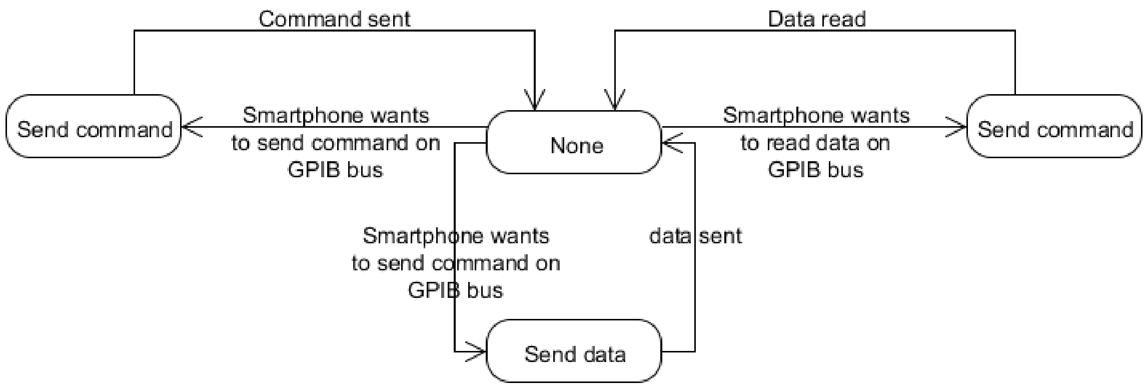

Command sent
Data read
Send command
Smartphone wants to send command on GPIB bus
None
Smartphone wants to read data on GPIB bus
Send command
Smartphone wants to send command on GPIB bus
data sent
Send data

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 ouput 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

Figure 39: Android mascot Bugdroid

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*.

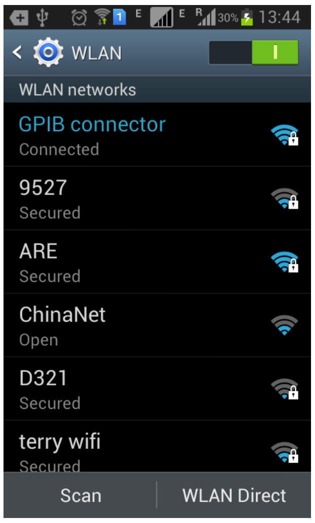


Figure 40: Connection to Wi-Fi module's AP

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.

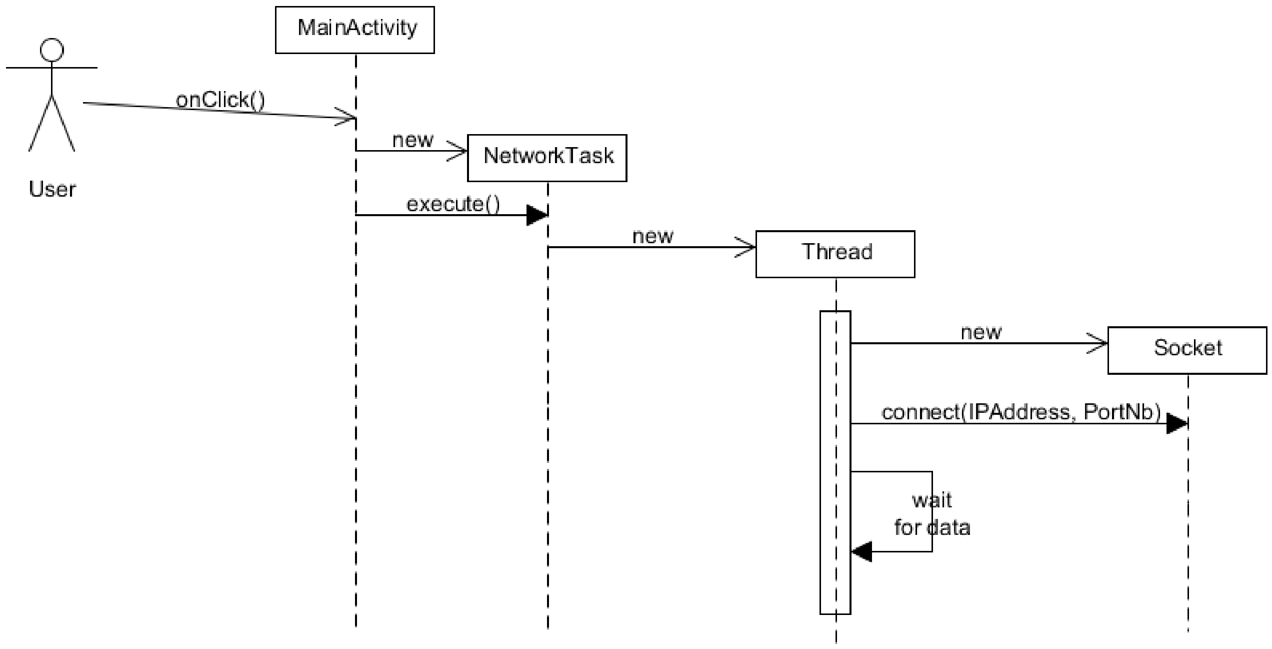


Figure 41: Connection sequence diagram

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.

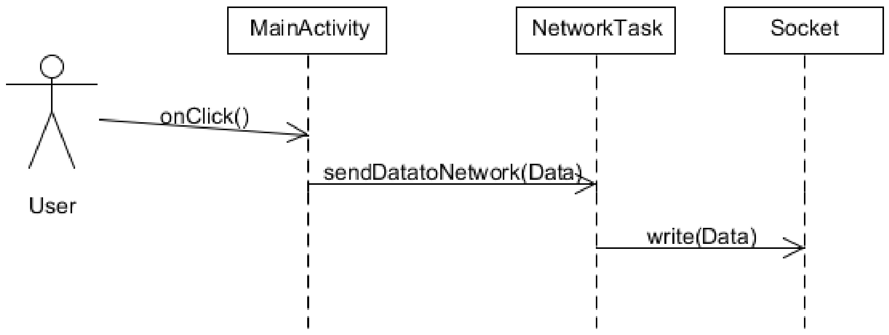


Figure 42: Data transmission sequence diagram

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.

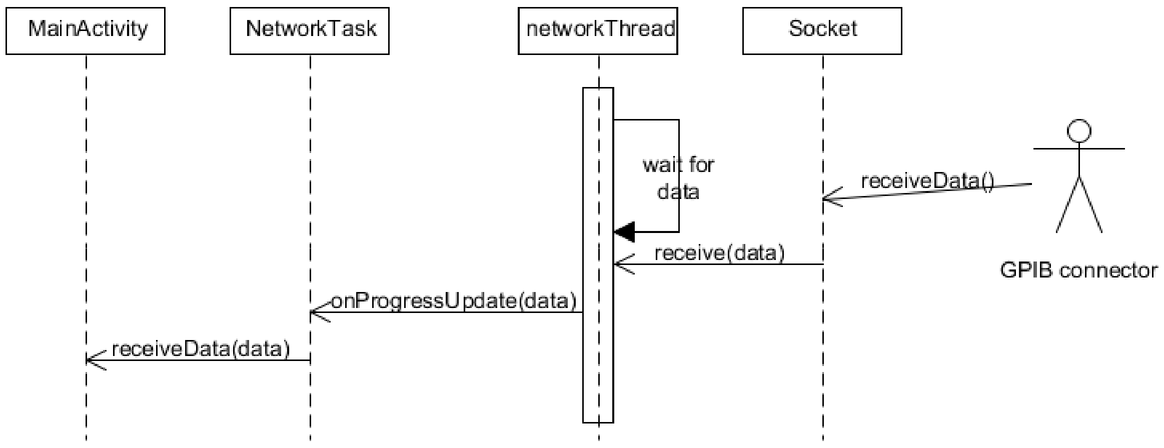


Figure 43: Data reception sequence diagram

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

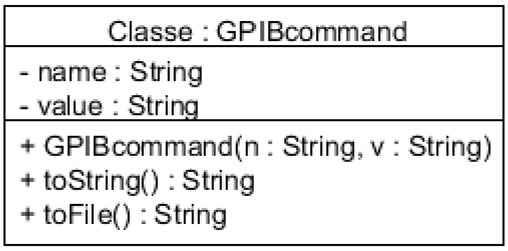
The libraries are stored as **.glb** files. They contain one line for each different command. Each lines is composed as follow :

Command name	,	Value (HEX)	\n

Figure 44: Library line structure

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.



```
Classe : GPIBcommand
- name : String
- value : String
+ GPIBcommand(n : String, v : String)
+ toString() : String
+ toFile() : String
```

Figure 45: GPIBcommand class

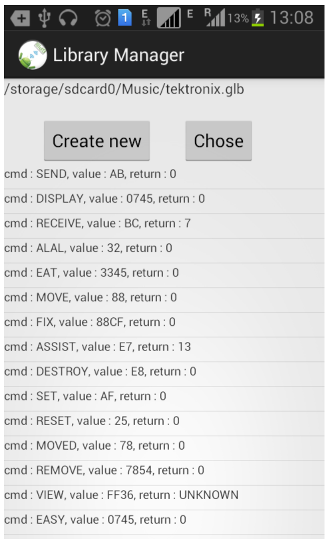
The **toString()** method returns a String that can be display for the user to show the name and the value contained in the function.

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 instruments or manufacturer. Once the library is created or chosen from the phone storage, a preview of the available commands is display on the screen.

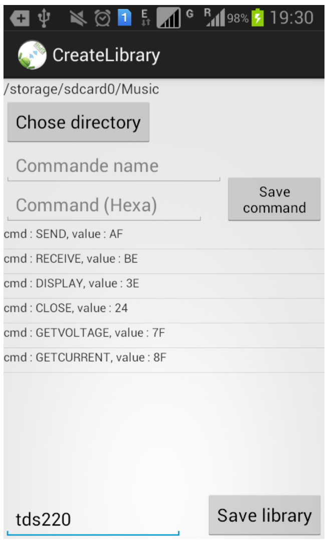
The following figures show the three Android activity that allow the user to manage the GPIB commands library.



/storage/sdcard0/Music/tektronix.glb

```
cmd : SEND, value : AB, return : 0
cmd : DISPLAY, value : 0745, return : 0
cmd : RECEIVE, value : BC, return : 7
cmd : ALAL, value : 32, return : 0
cmd : EAT, value : 3345, return : 0
cmd : MOVE, value : 88, return : 0
cmd : FIX, value : 88CF, return : 0
cmd : ASSIST, value : E7, return : 13
cmd : DESTROY, value : E8, return : 0
cmd : SET, value : AF, return : 0
cmd : RESET, value : 25, return : 0
cmd : MOVED, value : 78, return : 0
cmd : REMOVE, value : 7854, return : 0
cmd : VIEW, value : FF36, return : UNKNOWN
cmd : EASY, value : 0745, return : 0
```

Figure 46: Library Manager

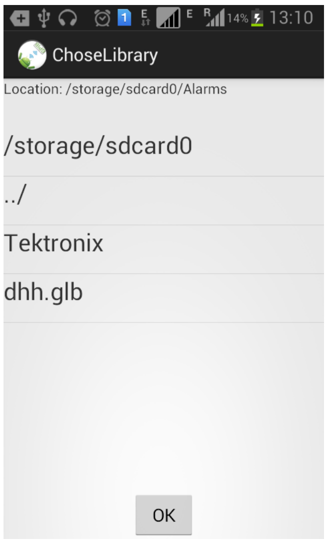


/storage/sdcard0/Music

```
cmd : SEND, value : AF
cmd : RECEIVE, value : BE
cmd : DISPLAY, value : 3E
cmd : CLOSE, value : 24
cmd : GETVOLTAGE, value : 7F
cmd : GETCURRENT, value : 8F
```

tds220

Figure 47: Create library



Location: /storage/sdcard0/Alarms

/storage/sdcard0
../
Tektronix
dhh.glb

Figure 48: Load library

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 smart-phone and chose a library to load.

The following figure shows the logic behind the file browser.

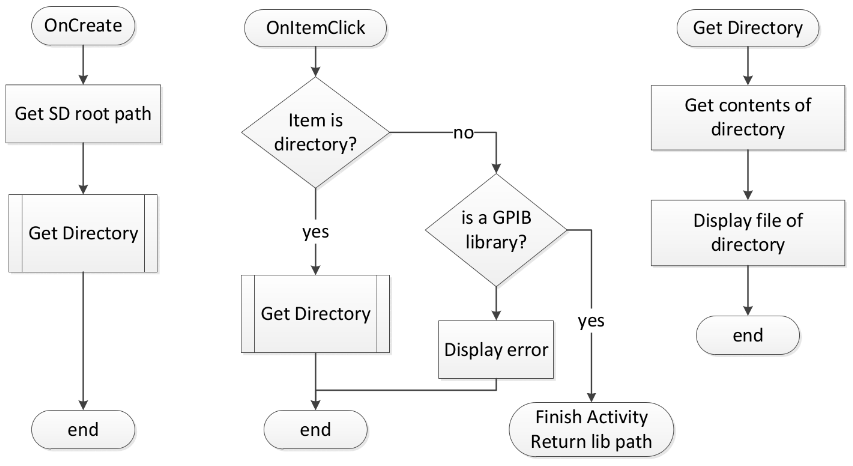


Figure 49: Chose library Activity

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

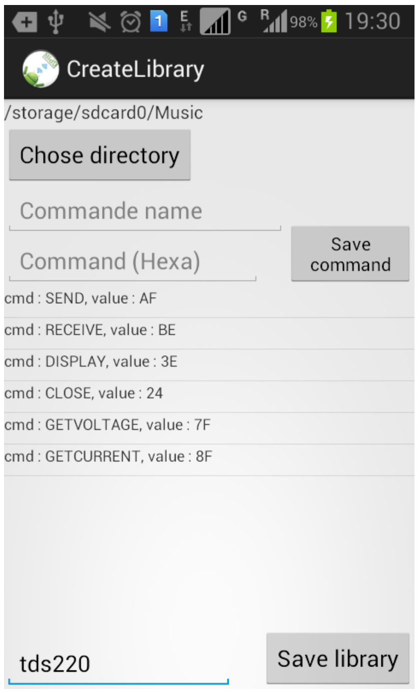
8.2.4 Create new library

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** [6] 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.



cmd : SEND, value : AF
cmd : RECEIVE, value : BE
cmd : DISPLAY, value : 3E
cmd : CLOSE, value : 24
cmd : GETVOLTAGE, value : 7F
cmd : GETCURRENT, value : 8F

Figure 50: Library creator

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:

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

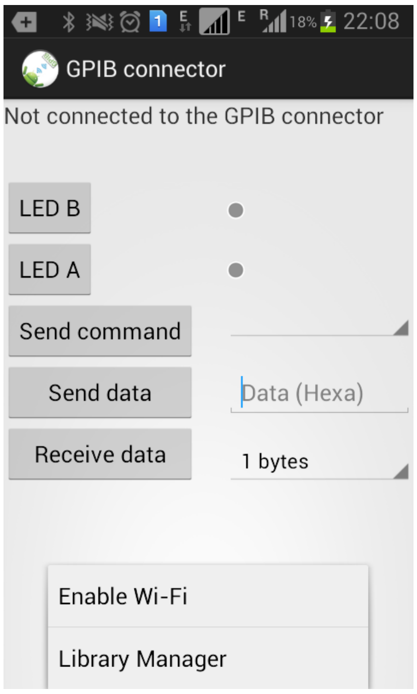
The GPIBcommand is recreated with this two attributes and put in a GPIBcommand list. When all lines have been read from the .glb file, the GPIBcommand list contains all the commands and can be use in the activity.

[6]See 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.



Not connected to the GPIB connector

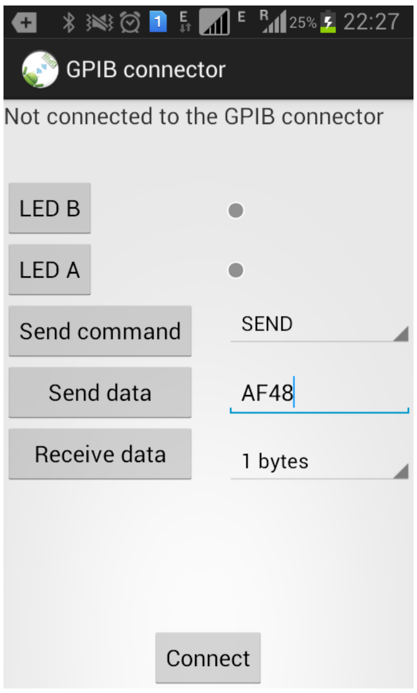
Figure 51: Main activity options

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 GPIBcommand 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



Not connected to the GPIB connector

Figure 52: Main activity

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 transmisson 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.

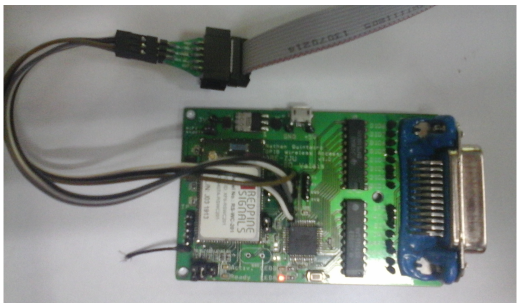
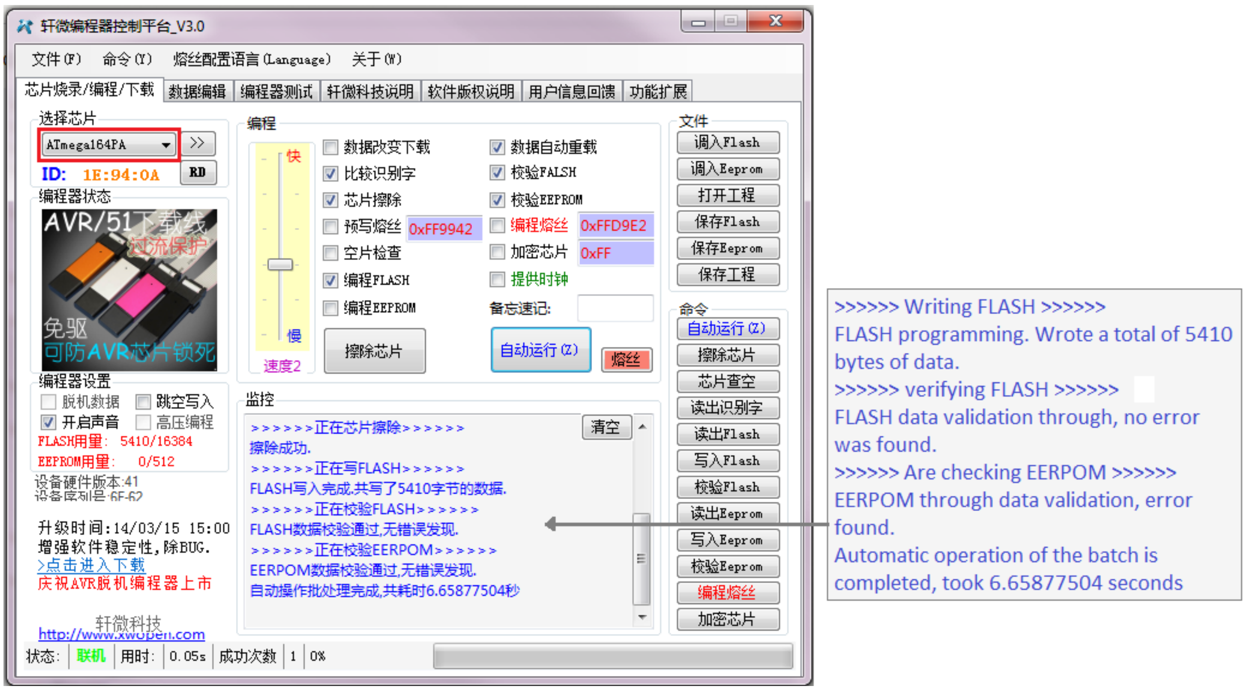


Figure 53: Circuit with USB/SPI connection to the PC

The programmer platform is started on the PC and a test code is downloaded in the MCU.



ATmega164PA
ID: 1E:94:0A

☐ 数据改变下载 ☑ 数据自动重载
☑ 比较识别字 ☑ 校验FALSH
☑ 芯片擦除 ☑ 校验EEPROM
☐ 预写熔丝 0xFF9942 ☐ 编程熔丝 0xFFD9E2
☐ 空片检查 ☐ 加密芯片 0xFF
☑ 编程FLASH ☐ 提供时钟
☐ 编程EEPROM 备忘速记:

☐ 脱机数据 ☐ 跳空写入
☑ 开启声音 ☐ 高压编程
FLASH用量: 5410/16384
EEPROM用量: 0/512
设备硬件版本:41
设备序列号:6F-62

升级时间:14/03/15 15:00
增强软件稳定性,除BUG.
庆祝AVR脱机编程器上市

```
>>>>>>>正在芯片擦除>>>>>>>
擦除成功.
>>>>>>>正在写FLASH>>>>>>>
FLASH写入完成.共写了5410字节的数据.
>>>>>>>正在校验FLASH>>>>>>>
FLASH数据校验通过,无错误发现.
>>>>>>>正在校验EERPOM>>>>>>>
EERPOM数据校验通过,无错误发现.
自动操作批处理完成,共耗时6.65877504秒
```

状态: 联机 | 用时: 0.05s | 成功次数 | 1 | 0%

```
>>>>>> Writing FLASH >>>>>>
FLASH programming. Wrote a total of 5410
bytes of data.
>>>>>> verifying FLASH >>>>>>
FLASH data validation through, no error
was found.
>>>>>> Are checking EERPOM >>>>>>
EERPOM through data validation, error
found.
Automatic operation of the batch is
completed, took 6.65877504 seconds
```

Figure 54: Circuit with USB/SPI connection to the PC

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.

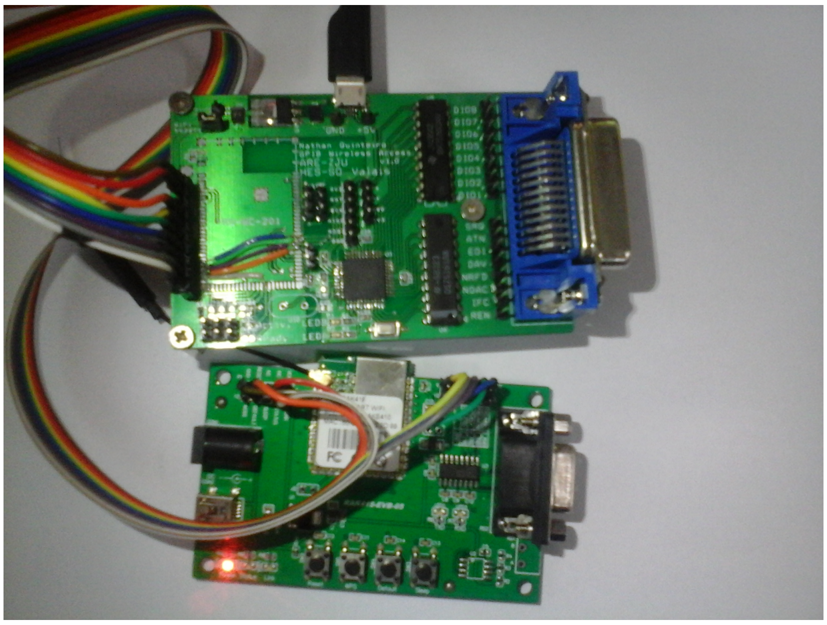


Figure 55: Evaluation board connection

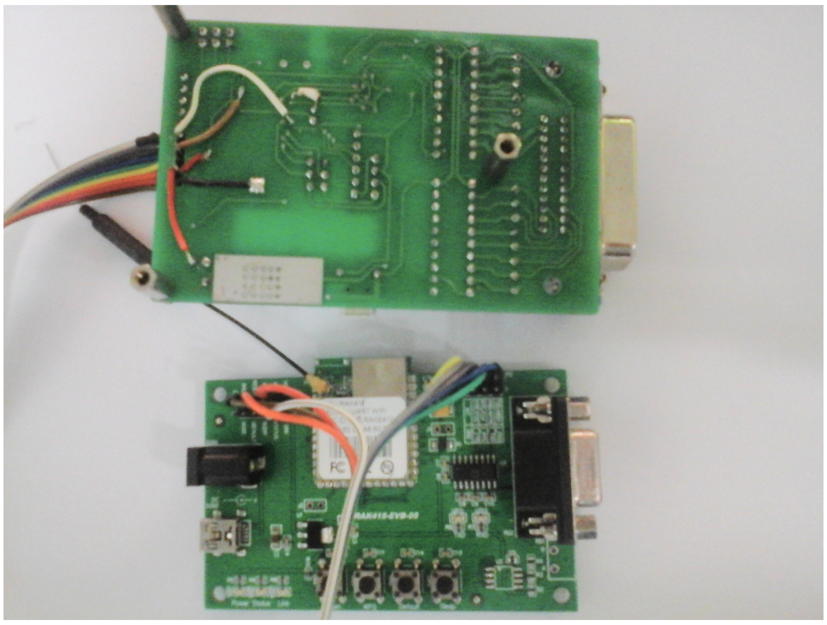


Figure 56: Hardware modification to connect to EVB

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.

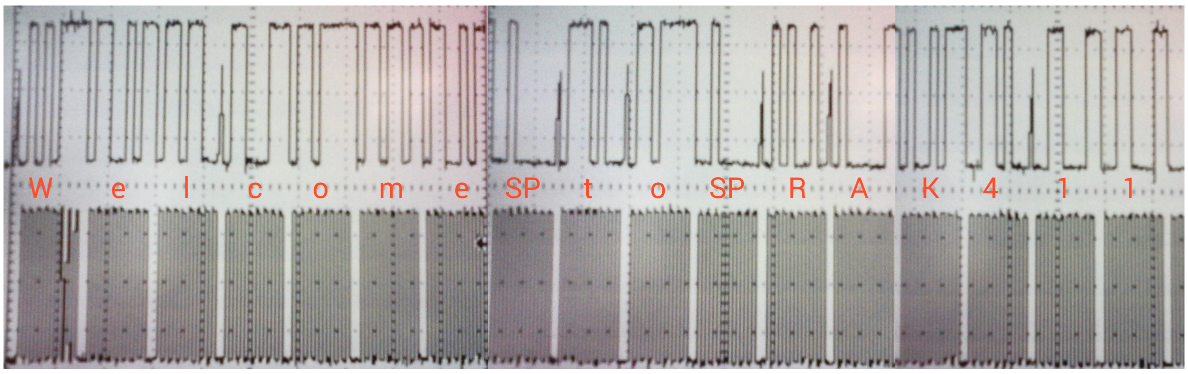


Figure 57: Init response - Start information

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

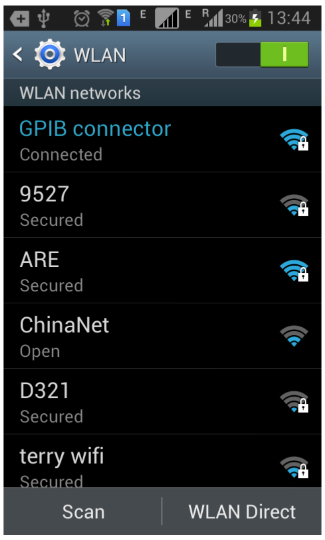
9.4.1 Connection on AP

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.



GPIB connector
Connected
9527
Secured
ARE
Secured
ChinaNet
Open
D321
Secured
terry wifi
Secured

Figure 58: AP visible by smartphone

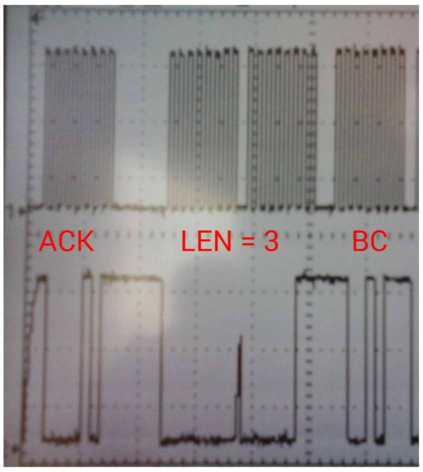


Figure 59: Connection of smartphone on AP

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 lis-
tening 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.

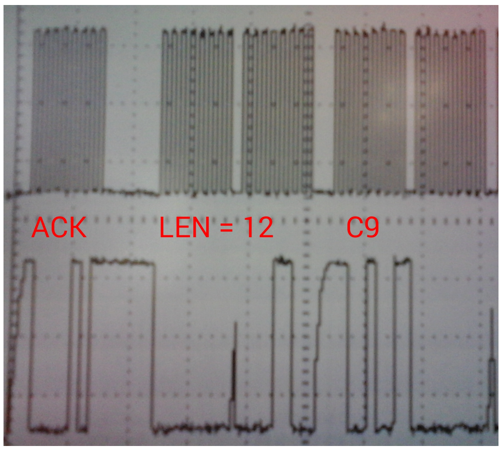


Figure 60: Connection to listening port

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.

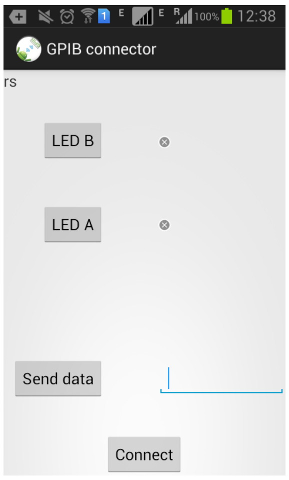


Figure 61: Android test application

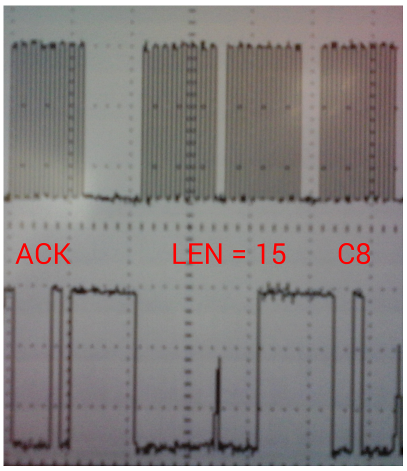


Figure 62: Data start

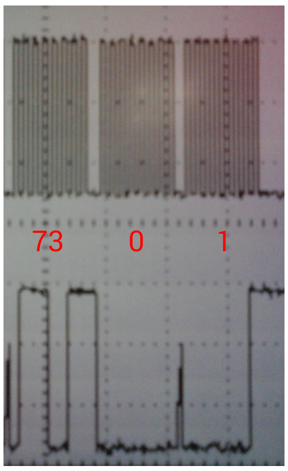


Figure 63: Data end - LED command

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.

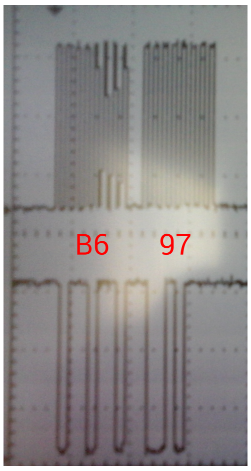


Figure 64: Data start

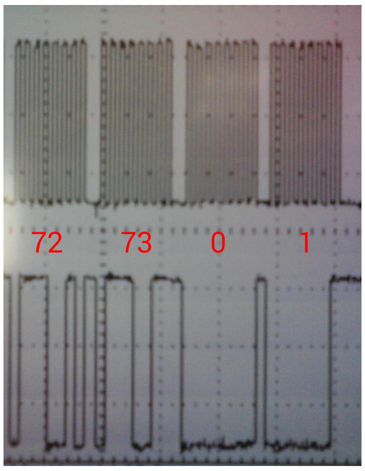
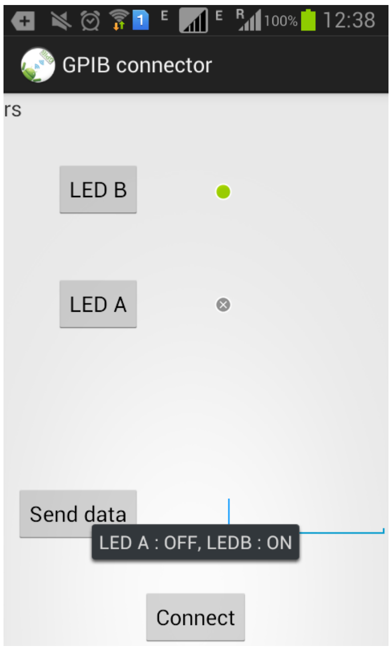


Figure 65: Data end - LED command

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.



LED A : OFF, LEDB : ON

Figure 66: LEDs state displayed in the APP

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.

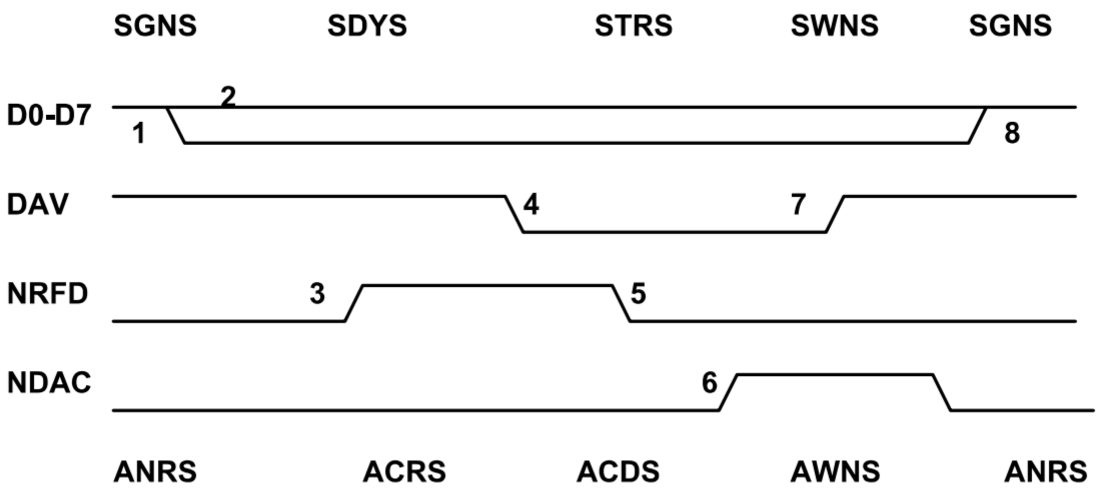


Figure 67: Handshake sequence

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 :

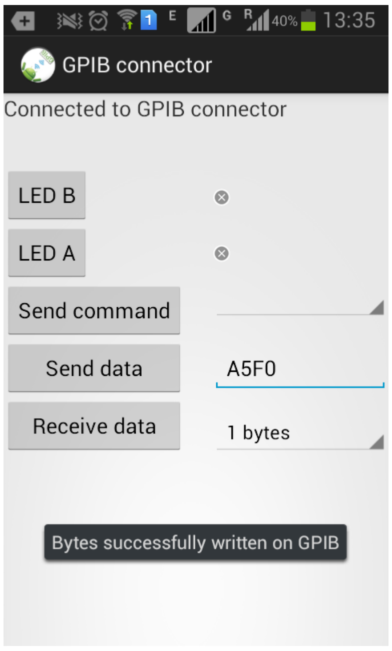
WRITE	DATA	2	0xA5	0xF0

Figure 68: Source test paquet

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.

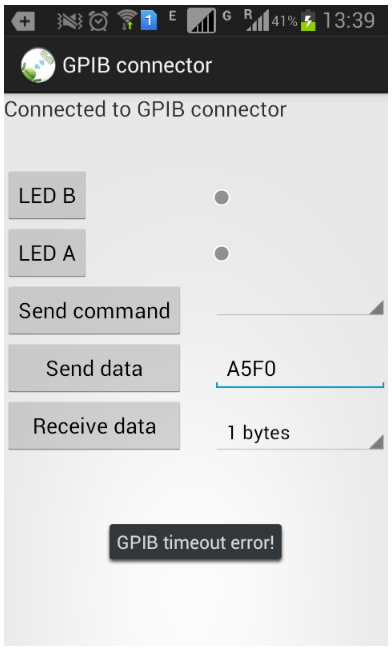
Once the writing is complete, the MCU sends a response to the smartphone, indicating that writing is successful. If the data are not written before the time limit, the MCU sends a response indicating that there is a GPIB error.


GPIB connector

Connected to GPIB connector

Bytes successfully written on GPIB

Figure 69: Writing successful


GPIB connector

Connected to GPIB connector

GPIB timeout error!

Figure 70: Timeout 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 :

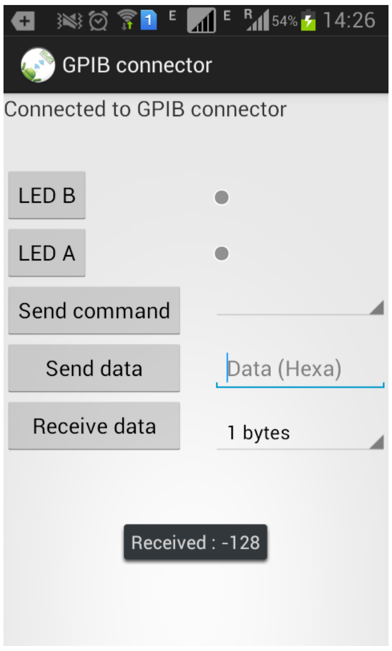
READ	1

Figure 71: Acceptor test paquet

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.

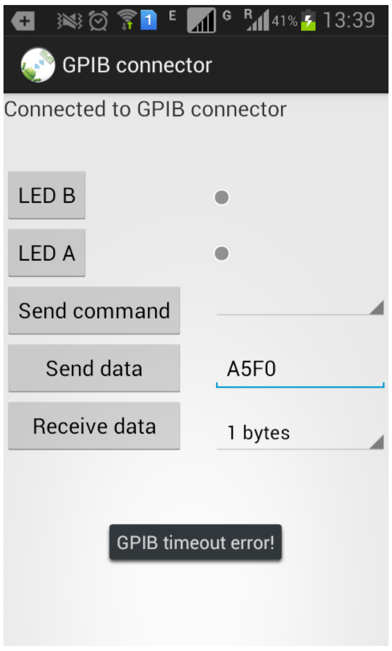
Once the reading is complete, the MCU sends a response to the smartphone, indicating that writing is successful and returning the value read. If the data are not read before the time limit, the MCU sends a response indicating that there is a GPIB error.


GPIB connector

Connected to GPIB connector

Received : -128

Figure 72: Reading successful


GPIB connector

Connected to GPIB connector

A5F0

GPIB timeout error!

Figure 73: Timeout 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 the 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

[1] The Institute of Electrical and Electronics Engineers Inc. IEEE Standard Digital Interface for Programmable Instrumentation, 1988.

[2] Capital Equipment Corporation. CEC-488 Programming and References. **Appendix A**.

[3] Shahin Farahani. *ZigBee Wireless Networks and Transceivers*. Newnes, 2011.

[4] Redpine Signals ®. RS-WC-201 Description. `www.redpinesignals.com/Modules/Internet_of_Things/Wiseconnect_Family/RS-WC-201.php`.

[5] Beijing Rakwireless Technology Co. Ltd. RAK411 - Datasheet. **Appendix G**.

[6] Atmel ®. ATmega164PA Datasheet. `www.atmel.com/devices/ATMEGA164PA.aspx`.

[7] Texas Instruments ®. DS75160A/DS75161A Datasheet. **Appendix B**.

[8] Redpine Signals ®. RS-WC-201 Datasheet. **Appendix D**.

[9] USB Community. Universal Serial Bus Power Delivery Specification. `www.usb.org/developers/docs/`.

[10] Texas Instruments ®. LM1117 Datasheet. **Appendix C**.

[11] Nathan Quinteiro. Schematic. **Appendix I**.

[12] Redpine Signals ®. RS-WC-201 Module Integration Guide. **Appendix E**.

[13] Nathan Quinteiro. Pcb design. **Appendix J**.

[14] Nathan Quinteiro. MCU Software code - General. **Appendix K**.

[15] Nathan Quinteiro. MCU Software code - RAK. **Appendix L**.

[16] Nathan Quinteiro. MCU Software code - GPIB. **Appendix M**.

[17] Beijing Rakwireless Technology Co. Ltd. RAK411 - SPI Software PRM. **Appendix H**.

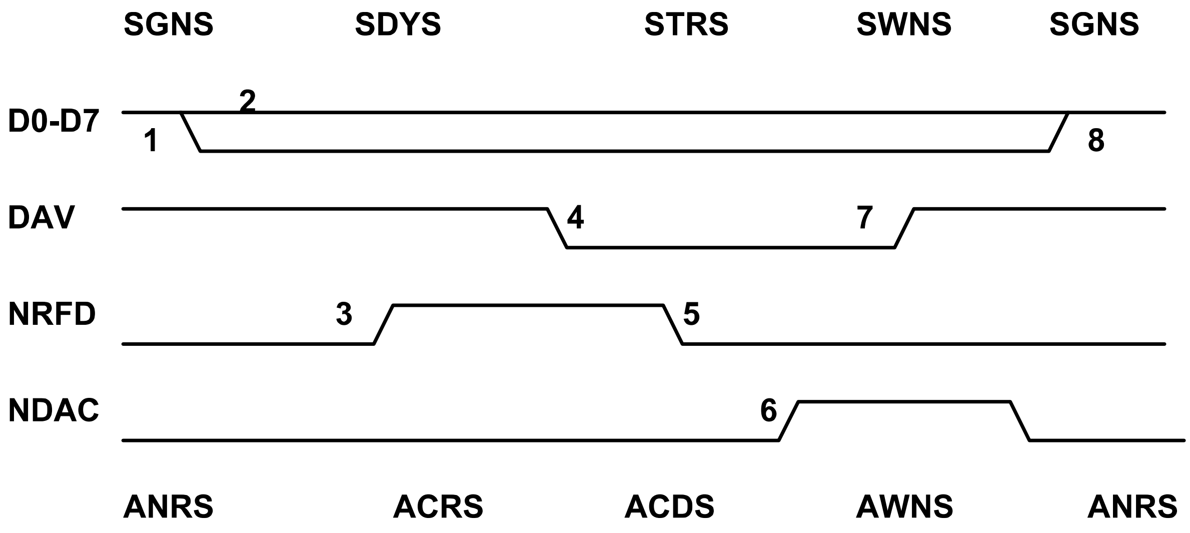
[18] Gartner Inc. Gartner's statistics on mobile OS. `http://www.gartner.com/newsroom/id/2573415`.

[19] Android Open Source Project. ADT Bundle download. `http://developer.android.com/sdk/index.html`.

[20] Nathan Quinteiro. Android Java Code. **Appendix N**.

[21] Redpine Signals ®. RS-WC-201 Programming Reference Manual. **Appendix F**.

Handshake Timing Sequence



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 availablility 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.

Appendix B - DS75160A/DS75161 datasheet

DS75160A,DS75161A

DS75160A/DS75161A IEEE-488 GPIB Transceivers

Literature Number: SNOSBL5A

May 1999

National Semiconductor

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.

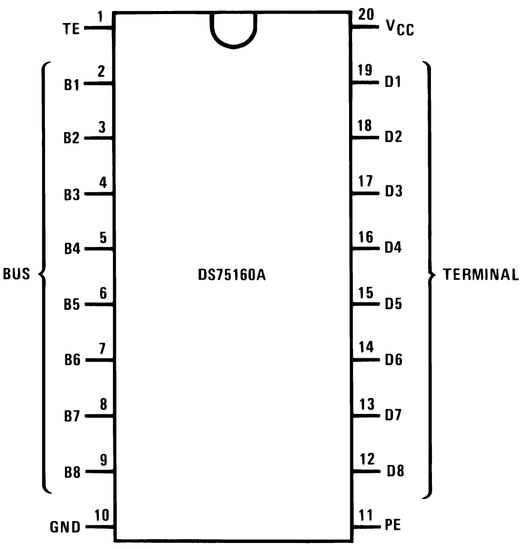
The General Purpose Interface Bus is comprised of 16 signal lines — 8 for data and 8 for interface management. The data lines are always implemented with DS75160A, and the management lines are either implemented with DS75161A in a single-controller system.

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

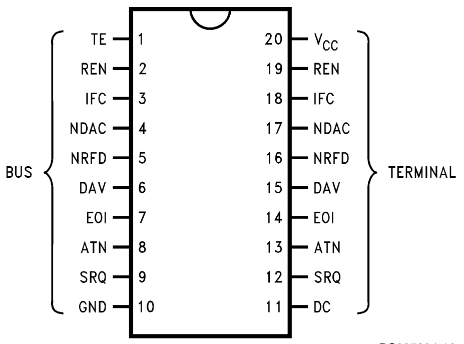
Dual-In-Line Package



DS005804-1

Top View
Order Number DS75160AN or DS75160AWM
See NS Package Number M20B or N20A

Dual-In-Line Package



DS005804-16

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

TRI-STATE® is a registered trademark of 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, V_{CC}	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	
Molded Package	1897 mW

	Min	Max	Units
V_{CC}, Supply Voltage	4.75	5.25	V
T_A, Ambient Temperature	0	70	˚C
I_{OL}, Output Low Current			
Bus		48	mA
Terminal		16	mA

Note 1: Derate molded package 15.2 mW/˚C above 25˚C.

Electrical Characteristics (Notes 3, 4)

Symbol	Parameter		Conditions		Min	Typ	Max	Units
V_{IH}	High-Level Input Voltage				2			V
V_{IL}	Low-Level Input Voltage						0.8	V
V_{IK}	Input Clamp Voltage		I_I = −18 mA			−0.8	−1.5	V
V_{HYS}	Input Hysteresis	Bus			400	500		mV
V_{OH}	High-Level Output Voltage	Terminal	I_{OH} = −800 A		2.7	3.5		V
		Bus (Note 4)	I_{OH} = −5.2 mA		2.5	3.4		
V_{OL}	Low-Level Output Voltage	Terminal	I_{OL} = 16 mA			0.3	0.5	V
		Bus	I_{OH} = 48 mA			0.4	0.5	
I_{IH}	High-Level Input Current	Terminal and TE, PE, DC, SC Inputs	V_I = 5.5V			0.2	100	A
			V_I = 2.7V			0.1	20	
I_{IL}	Low-Level Input Current		V_I = 0.5V			−10	−100	A
V_{BIAS}	Terminator Bias Voltage at Bus Port	Bus	Driver Disabled	$I_{I(bus)}$ = 0 (No Load)	2.5	3.0	3.7	V
I_{LOAD}	Terminator Bus Loading Current		Driver Disabled	$V_{I(bus)}$ = −1.5V to 0.4V	−1.3			mA
				$V_{I(bus)}$ = 0.4V to 2.5V	0		−3.2	
				$V_{I(bus)}$ = 2.5V to 3.7V			2.5 −3.2	
				$V_{I(bus)}$ = 3.7V to 5V	0		2.5	
				$V_{I(bus)}$ = 5V to 5.5V	0.7		2.5	
			V_{CC} = 0V, $V_{I(bus)}$ = 0V to 2.5V				40	A
I_{OS}	Short-Circuit Output Current	Terminal	V_I = 2V, V_O = 0V (Note 5)		−15	−35	−75	mA
		Bus (Note 6)			−35	−75	−150	
I_{CC}	Supply Current	DS75160A	Transmit, TE = 2V, PE = 2V, V_I = 0.8V			85	125	mA
			Receive, TE = 0.8V, PE = 2V, V_I = 0.8V			70	100	
		DS75161A	TE = 0.8V, DC = 0.8V, V_I = 0.8V			84	125	
C_{IN}	Bus-Port Capacitance	Bus	V_{CC} = 5V or 0V, V_I = 0V to 2V, f = 1 MHz			20	30	pF

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 T_A = 25˚C and V_{CC} = 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 ±5%, T_A = 0˚C to 70˚C

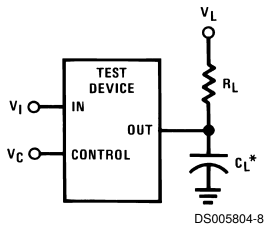
Symbol	Parameter	From	To	Conditions	DS75160A			DS75161A			DS75162A			Units
					Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
t_{PLH}	Propagation Delay Time, Low to High Level Output	Terminal	Bus	V_L = 2.3V R_L = 38.3Ω		10	20		10	20		10	20	ns
t_{PHL}	Propagation Delay Time, High to Low Level Output			C_L = 30 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 = 240Ω		14	20		14	20		14	20	ns
t_{PHL}	Propagation Delay Time, High to Low Level Output			C_L = 30 pF *Figure 2*		10	20		10	20		10	20	ns
t_{PZH}	Output Enable Time to High Level	TE, DC, or SC (Note 8) (Note 9)	Bus	V_I = 3.0V V_L = 0V		19	32		23	40		23	40	ns
t_{PHZ}	Output Disable Time From High Level			R_L = 480Ω C_L = 15 pF *Figure 1*		15	22		15	25		15	25	ns
t_{PZL}	Output Enable Time to Low Level			V_I = 0V V_L = 2.3V		24	35		28	48		28	48	ns
t_{PLZ}	Output Disable Time From Low Level			R_L = 38.3Ω C_L = 15 pF *Figure 1*		17	25		17	27		17	27	ns
t_{PZH}	Output Enable Time to High Level	TE, DC, or SC (Note 8) (Note 9)	Terminal	V_I = 3.0V V_L = 0V		17	33		18	40		18	40	ns
t_{PHZ}	Output Disable Time From High Level			R_L = 3 kΩ C_L = 15 pF *Figure 1*		15	25		22	33		22	33	ns
t_{PZL}	Output Enable Time to Low Level			V_I = 0V V_L = 5V		25	39		28	52		28	52	ns
t_{PLZ}	Output Disable Time From Low Level			R_L = 280Ω C_L = 15 pF *Figure 1*		15	27		20	35		20	35	ns
t_{PZH}	Output Pull-Up Enable Time (DS75160A Only)	PE (Note 8)	Bus	V_I = 3V V_L = 0V		10	17		NA			NA		ns
t_{PHZ}	Output Pull-Up Disable Time (DS75160A Only)			R_L = 480Ω C_L = 15 pF *Figure 1*		10	15		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.

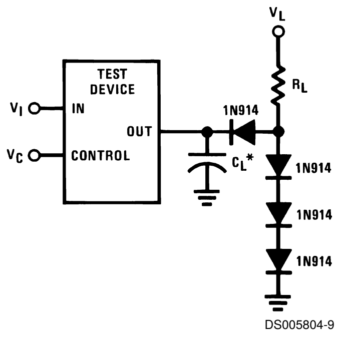
Note 9: Test configuration should be connected to only one transceiver at a time due to the high current stress caused by the V_I voltage source when the output connected to that input becomes active.

Switching Load Configurations



V_C logic high = 3.0V
V_C logic low = 0V
*C_L includes jig and probe capacitance

FIGURE 1.

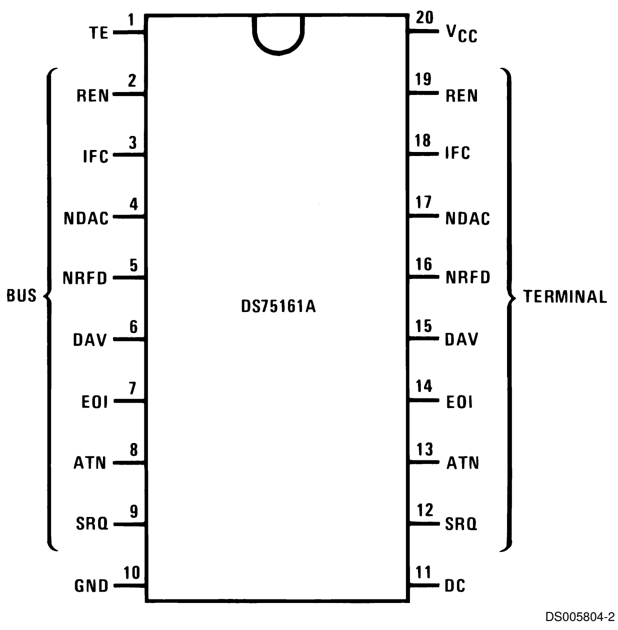


V_C logic high = 3.0V
V_C logic low = 0V
*C_L includes jig and probe capacitance

FIGURE 2.

Connection Diagrams

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 DIO_1–DIO_8. 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.

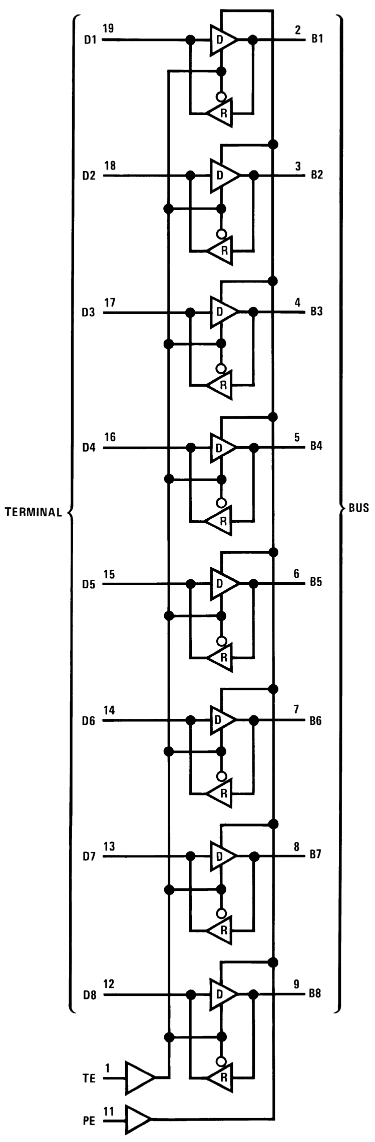
Functional Description (Continued)

Table of Signal Line Abbreviations

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

Logic Diagrams

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-ups disabled by switching the PE input (pin 11) to the logic low state. This mode configures the outputs as open collector.

DS005804-4

Functional Truth Tables

DS75160A

Control Input Level		Data Transceivers	
TE	PE	Direction	Bus Port Configuration
H	H	T	Totem-Pole Output
H	L	T	Open Collector Output
L	X	R	Input

DS75161A

Control Input Level			Transceiver Signal Direction							
TE	DC	ATN*		EOI	REN	IFC	SRQ	NRFD	NDAC	DAV
H	H		R		R	R	T	R	R	T
H	L		T		T	T	R	R	R	T
L	H		R		R	R	T	T	T	R
L	L		T		T	T	R	T	T	R
H	X	H		T						
L	X	H		R						
X	H	L		R						
X	L	L		T						

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.

Appendix C - LM1117 datasheet

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.

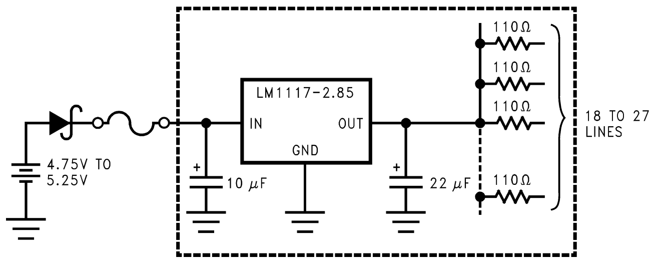
The LM1117-N is available in an adjustable version, which can set the output voltage from 1.25V to 13.8V with only two external resistors. In addition, it is also available in five fixed voltages, 1.8V, 2.5V, 2.85V, 3.3V, and 5V.

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 DDPAK 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



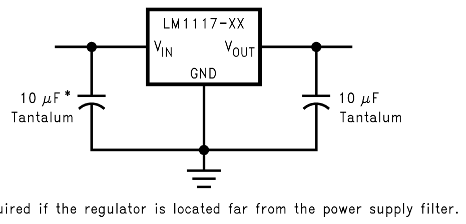
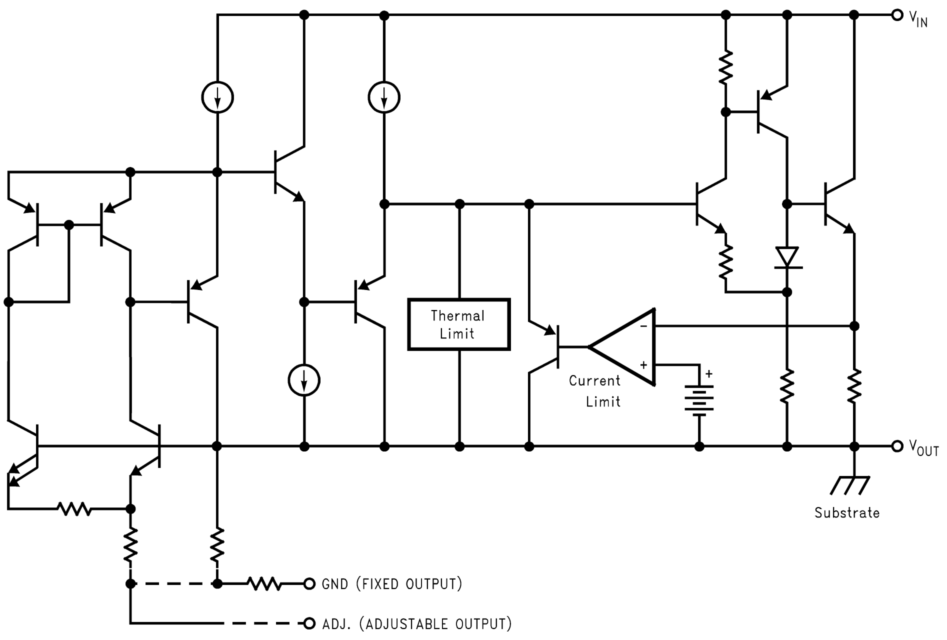

LM1117-XX
V_{IN}
V_{OUT}
GND
10 μF *
Tantalum
10 μF
Tantalum
* Required if the regulator is located far from the power supply filter.

Figure 1. Fixed Output Regulator

Block Diagram



Connection Diagrams

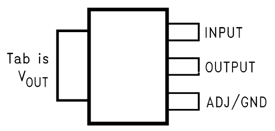


Figure 2. SOT-223 Top View

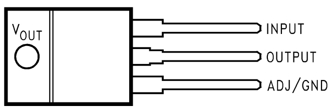


Figure 3. TO-220 Top View

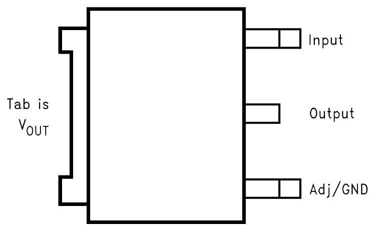


Figure 4. PFM Top View

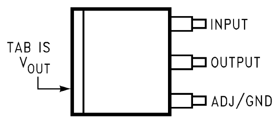


Figure 5. DDPAK/TO-263 Top View

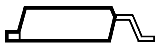
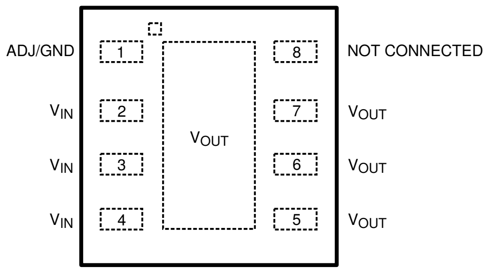


Figure 6. DDPAK/TO-263 Side View



When using the WSON package
Pins 2, 3 & 4 must be connected together and
Pins 5, 6 & 7 must be connected together

Figure 7. WSON Top View

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[1][2]

Maximum Input Voltage (V_{IN} to GND)		20V
Power Dissipation[3]		Internally Limited
Junction Temperature (T_J)[3]		150°C
Storage Temperature Range		-65°C to 150°C
Lead Temperature	TO-220 (T) Package	260°C, 10 sec
	SOT-223 (IMP) Package	260°C, 4 sec
ESD Tolerance[4]		2000V

(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 T_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[1]

Input Voltage (V_{IN} to GND)		15V
Junction Temperature Range (T_J)[2]	LM1117-N	0°C to 125°C
	LM1117I	−40°C to 125°C

(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 T_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°C. Limits appearing in **Boldface** type apply over the entire junction temperature range for operation, 0°C to 125°C.

Symbol	Parameter	Conditions	Min(1)	Typ(2)	Max(1)	Units
V_{REF}	Reference Voltage	LM1117-N-ADJ I_{OUT} = 10mA, V_{IN}-V_{OUT} = 2V, T_J = 25°C 10mA ≤ I_{OUT} ≤ 800mA, 1.4V ≤ V_{IN}-V_{OUT} ≤ 10V	1.238 **1.225**	1.250 1.250	1.262 **1.270**	V V
V_{OUT}	Output Voltage	LM1117-N-1.8 I_{OUT} = 10mA, V_{IN} = 3.8V, T_J = 25°C 0 ≤ I_{OUT} ≤ 800mA, 3.2V ≤ V_{IN} ≤ 10V	1.782 **1.746**	1.800 1.800	1.818 **1.854**	V V
		LM1117-N-2.5 I_{OUT} = 10mA, V_{IN} = 4.5V, T_J = 25°C 0 ≤ I_{OUT} ≤ 800mA, 3.9V ≤ V_{IN} ≤ 10V	2.475 **2.450**	2.500 2.500	2.525 **2.550**	V V
		LM1117-N-2.85 I_{OUT} = 10mA, V_{IN} = 4.85V, T_J = 25°C 0 ≤ I_{OUT} ≤ 800mA, 4.25V ≤ V_{IN} ≤ 10V 0 ≤ I_{OUT} ≤ 500mA, V_{IN} = 4.10V	2.820 **2.790** **2.790**	2.850 2.850 2.850	2.880 **2.910** **2.910**	V V V
		LM1117-N-3.3 I_{OUT} = 10mA, V_{IN} = 5V T_J = 25°C 0 ≤ I_{OUT} ≤ 800mA, 4.75V≤ V_{IN} ≤ 10V	3.267 **3.235**	3.300 3.300	3.333 **3.365**	V V
		LM1117-N-5.0 I_{OUT} = 10mA, V_{IN} = 7V, T_J = 25°C 0 ≤ I_{OUT} ≤ 800mA, 6.5V ≤ V_{IN} ≤ 12V	4.950 **4.900**	5.000 5.000	5.050 **5.100**	V V
ΔV_{OUT}	Line Regulation(3)	LM1117-N-ADJ I_{OUT} = 10mA, 1.5V ≤ V_{IN}-V_{OUT} ≤ 13.75V		0.035	**0.2**	%
		LM1117-N-1.8 I_{OUT} = 0mA, 3.2V ≤ V_{IN} ≤ 10V		1	**6**	mV
		LM1117-N-2.5 I_{OUT} = 0mA, 3.9V ≤ V_{IN} ≤ 10V		1	**6**	mV
		LM1117-N-2.85 I_{OUT} = 0mA, 4.25V ≤ V_{IN} ≤ 10V		1	**6**	mV
		LM1117-N-3.3 I_{OUT} = 0mA, 4.75V ≤ V_{IN} ≤ 15V		1	**6**	mV
		LM1117-N-5.0 I_{OUT} = 0mA, 6.5V ≤ V_{IN} ≤ 15V		1	**10**	mV
ΔV_{OUT}	Load Regulation(3)	LM1117-N-ADJ V_{IN}-V_{OUT} = 3V, 10 ≤ I_{OUT} ≤ 800mA		0.2	**0.4**	%
		LM1117-N-1.8 V_{IN} = 3.2V, 0 ≤ I_{OUT} ≤ 800mA		1	**10**	mV
		LM1117-N-2.5 V_{IN} = 3.9V, 0 ≤ I_{OUT} ≤ 800mA		1	**10**	mV
		LM1117-N-2.85 V_{IN} = 4.25V, 0 ≤ I_{OUT} ≤ 800mA		1	**10**	mV
		LM1117-N-3.3 V_{IN} = 4.75V, 0 ≤ I_{OUT} ≤ 800mA		1	**10**	mV
		LM1117-N-5.0 V_{IN} = 6.5V, 0 ≤ I_{OUT} ≤ 800mA		1	**15**	mV
V_{IN}-V_{OUT}	Dropout Voltage(4)	I_{OUT} = 100mA		1.10	**1.20**	V
		I_{OUT} = 500mA		1.15	**1.25**	V
		I_{OUT} = 800mA		1.20	**1.30**	V

(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°C. Limits appearing in **Boldface** type apply over the entire junction temperature range for operation, 0°C to 125°C.

Symbol	Parameter	Conditions	Min[1]	Typ[2]	Max[1]	Units
I_{LIMIT}	Current Limit	V_{IN}-V_{OUT} = 5V, T_J = 25°C	800	1200	1500	mA
	Minimum Load Current[5]	LM1117-N-ADJ V_{IN} = 15V		1.7	**5**	mA
	Quiescent Current	LM1117-N-1.8 V_{IN} ≤ 15V		5	**10**	mA
		LM1117-N-2.5 V_{IN} ≤ 15V		5	**10**	mA
		LM1117-N-2.85 V_{IN} ≤ 10V		5	**10**	mA
		LM1117-N-3.3 V_{IN} ≤ 15V		5	**10**	mA
		LM1117-N-5.0 V_{IN} ≤ 15V		5	**10**	mA
	Thermal Regulation	T_A = 25°C, 30ms Pulse		0.01	0.1	%/W
	Ripple Regulation	f_{RIPPLE} =1 20Hz, V_{IN}-V_{OUT} = 3V V_{RIPPLE} = 1V_{PP}	**60**	75		dB
	Adjust Pin Current			60	**120**	μA
	Adjust Pin Current Change	10 ≤ I_{OUT}≤ 800mA, 1.4V ≤ V_{IN}-V_{OUT} ≤ 10V		0.2	**5**	μA
	Temperature Stability			0.5		%
	Long Term Stability	T_A = 125°C, 1000Hrs		0.3		%
	RMS Output Noise	(% of V_{OUT}), 10Hz ≤ f ≤10kHz		0.003		%
	Thermal Resistance Junction-to-Case	3-Lead SOT-223		15.0		°C/W
		3-Lead TO-220		3.0		°C/W
		3-Lead TO-263		10		°C/W
	Thermal Resistance Junction-to-Ambient (No air flow)	3-Lead SOT-223 (No heat sink)		136		°C/W
		3-Lead TO-220 (No heat sink)		79		°C/W
		3-Lead TO-263[6] (No heat sink)		92		°C/W
		3-Lead PFM		55		°C/W
		8-Lead WSON[7]		40		°C/W

(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°C. Limits appearing in **Boldface** type apply over the entire junction temperature range for operation, −40°C to 125°C.

Symbol	Parameter	Conditions	Min[1]	Typ[2]	Max[1]	Units
V_{REF}	Reference Voltage	LM1117I-ADJ I_{OUT} = 10mA, V_{IN}-V_{OUT} = 2V, T_J = 25°C 10mA ≤ I_{OUT} ≤ 800mA, 1.4V ≤ V_{IN}-V_{OUT} ≤ 10V	1.238 **1.200**	1.250 1.250	1.262 **1.290**	V V
V_{OUT}	Output Voltage	LM1117I-3.3 I_{OUT} = 10mA, V_{IN} = 5V, T_J = 25°C 0 ≤ I_{OUT} ≤ 800mA, 4.75V ≤ V_{IN} ≤ 10V	3.267 **3.168**	3.300 3.300	3.333 **3.432**	V V
		LM1117I-5.0 I_{OUT} = 10mA, V_{IN} = 7V, T_J = 25°C 0 ≤ I_{OUT} ≤ 800mA, 6.5V ≤ V_{IN} ≤ 12V	4.950 **4.800**	5.000 5.000	5.050 **5.200**	V V

(1) All limits are ensured by testing or statistical analysis.
(2) Typical Values represent the most likely parametric normal.

RS-WC-201

Datasheet

Version 2.02

June 2013

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

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.

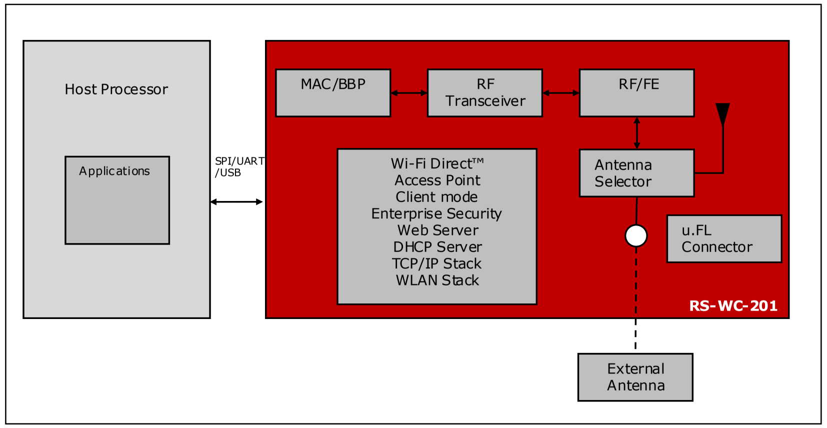
Applications:

- 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

RS-WC-201 System Architecture



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[1]
- 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.

- EAP-TLS
- EAP-TTLS
- EAP-FAST
- PEAP-MS-CHAP-V2

- WPA/WPA2-PSK
- WPS-Push Button Support

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 Package Description

2.1 Top View

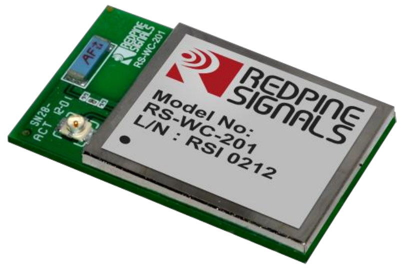


Figure 1: RS-WC-201 Module

2.2 Module Dimensions

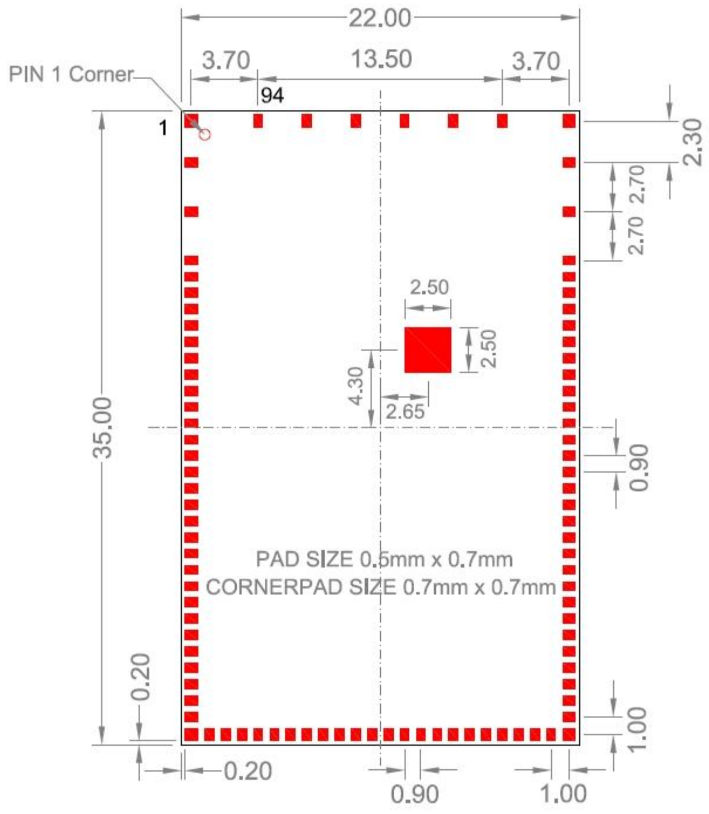


Figure 2: Module Dimensions

Package Type: LGA

Module height: 2.75 +/- 0.05 mm

2.3 Pin Description

Pin No	Pin Name	Direction	Type	Description
1	GND	Ground	-	Ground
2	GND	Ground	-	Ground
3	NC	-	-	No connect
4	USB_VREGIN	Power	-	5V power supply for USB transceiver. No connect when USB not used
5	GND	-	-	Ground
6	VCC3.3	Power	-	3.3V Power Supply
7	VCC3.3	Power	-	3.3V Power Supply
8	VCC3.3	Power	-	3.3V Power Supply
9	JPD0	-	-	Connect pull down of 1 kOhms
10	JNC	-	-	No connect
11	JPD2	-	-	Connect pull down of 1 kOhms
12	JPD1	-	-	Connect pull down of 1 kOhms
13	WF_HNDSHKE1	Input	3.3V I/O, 9mA	Handshake signal for wireless firmware upgrade. Should be connected to a GPIO pin of the Host MCU
14	WF_HNDSHKE2	Output	3.3V I/O, 9mA	Handshake signal for wireless firmware upgrade. Should be connected to a GPIO pin of the Host MCU
15	NC	-	-	No connect
16	NC	-	-	No connect
17	NC	-	-	No connect
18	GND	-	-	Ground
19	WLAN_ACTIVE	Output	LVCMOS, 2mA	Used for BT Coexistence. It indicates with logic high that WLAN activity is in progress. When low, BT device has the opportunity to transmit. Not used in current firmware, should be left open
20	BT_PRIORITY	Input	LVCMOS, 2mA	Used to indicate through logic high that BT is transmitting high priority traffic. Not used in current firmware, should be grounded
21	NC	-	-	No connect
22	GND	-	-	Ground
23	USB_DP	Input	3.3V I/O	USB differential pin (Positive). Not used in current firmware, should be left open
24	USB_DM	Input	3.3V I/O	USB differential pin

				(Negative). No connect if USB is not used
25	PT_GPIO1	Output	3.3V I/O, 2mA	Pass through output pin controllable by Host software.
26	PT_GPIO2	Output	3.3V I/O, 2mA	Pass through output pin controllable by Host software.
27	TP1	-	-	Test point
28	TP2	-	-	Test point
29	TP3	-	-	Test point
30	TP4	-	-	Test point
31	TP5	-	-	Test point
32	NC	-	-	No connect
33	GND	-	-	Ground
34	NC	-	-	No Connect.
35	NC	-	-	No connect
36	NC	-	-	No connect
37	NC	-	-	No connect
38	NC	-	-	No connect
39	NC	-	-	No connect
40	NC	-	-	No connect
41	NC	-	-	No connect
42	NC	-	-	No connect.
43	RESET_N	Input	3.3V I/O	Active low reset for the module
44	GND	-	-	Ground
45	MODE_SEL_2	Input	3.3V I/O	Interface selection Bit[2]
46	MODE_SEL_1	Input	3.3V I/O	Interface selection Bit[1]
47	MODE_SEL_0	Input	3.3V I/O	Interface selection Bit[0] Module interface Selection: Bit[2],Bit[1],Bit[0] = '000' for selecting UART interface Bit[2],Bit[1],Bit[0] = '001' for selecting SPI interface Bit[2],Bit[1],Bit[0] = '010' for selecting USB interface
48	WIFI_ACTIVITY	Output	3.3V I/O	Wi-Fi Activity indicator. This pin is '0' when the module is transferring data.
49	CARD_READY	Output	3.3V I/O	Card ready indicator. Logic '0' indicates successful boot-up of the module
50	GND	Ground	-	Ground

51	NC	-	-	No connect
52	NC	-	-	No connect
53	NC	-	-	No connect
54	NC	-	-	No connect
55	ADC2	Input	3.3V I/O	Analog input to internal ADC. Not used in current firmware, should be left open
56	GND	Ground	-	Ground
57	ADC1	Input	3.3V I/O	Analog input to internal ADC. Not used in current firmware, should be left open
58	WAKEUP	Input	3.3V I/O	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.
59	SPI_READY	Output	3.3V I/O, 2mA	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
60	UART_RTS	Output	3.3V I/O, 2mA	UART Request to Send. Not used in current firmware, should be left open
61	UART_CTS	Input	3.3V I/O	UART Clear to Send. Not used in current firmware, should be left open
62	UART_TX	Output	3.3V I/O, 2mA	UART- Transmit
63	UART_RX	Input	3.3V I/O	UART – Receive
64	INTR	Output	3.3V I/O, 2mA	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
65	SPI_CS	Input	3.3V I/O	SPI slave select
66	SPI_CLK	Input	3.3V I/O	SPI clock input
67	SPI_MISO	Output	3.3V I/O, 2mA	SPI data output
68	SPI_MOSI	Input	3.3V I/O	SPI data input
69	GND	-	-	Ground
70	PA_EN	Output	LVCMOS, 4mA	External PA enable. If an external Power Amplifier is used, this should be connected to the enable of the external

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

Table 1: Pin Description

Notes: The reference schematics showing pin connections for the module are present in the Module Integration Guide document. The document can be requested from Redpine Signals. Some pins are not used in the default configuration or mode of operation. These may be used in custom applications with appropriate firmware.

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.

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

Table 9: Power Consumption

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

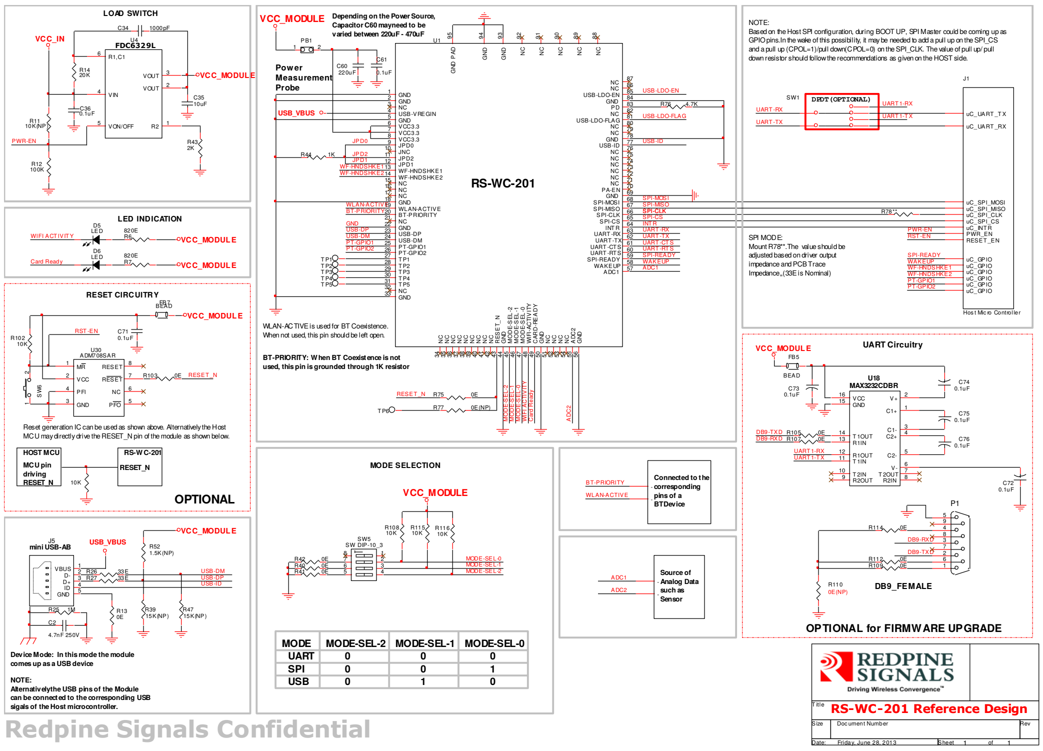
Document Version	**Changes**
2.7	Notes on USB Connections included. Wireless firmware upgrade feature available from firmware version 2.1.0.1.2.5 onwards.
2.8	PT_GPIO1, PT_GPIO2, WIFI_ACTIVITY, and CARD_READY pins are available from 2.8V onwards, note added on Inrush current.
2.9	Changed note on BT_PRIORITY and WLAN_ACTIVE.

Table of Contents

List of Figures

<This page is intentionally left blank>

1 RS-WC-201 Reference Schematics



VCC_MODULE Depending on the Power Source, Capacitor C60 may need to be varied between 220uF - 470uF

WLAN-ACTIVE is used for BT Coexistence. When not used, this pin should be left open.

BT-PRIORITY: When BT Coexistence is not used, this pin is grounded through 1K resistor

Reset generation IC can be used as shown above. Alternatively the Host MCU may directly drive the RESET_N pin of the module as shown below.

Device Mode: In this mode the module comes up as a USB device

NOTE:
Alternatively the USB pins of the Module can be connected to the corresponding USB sigals of the Host microcontroller.

NOTE:
Based on the Host SPI configuration, during BOOT UP, SPI Master could be coming up as GPIO pins. In the wake of this possibility, it may be needed to add a pull up on the SPI_CS and a pull up (CPOL=1)/pull down(CPOL=0) on the SPI_CLK. The value of pull up/ pull down resistor should follow the recommendations as given on the HOST side.

SPI MODE:
Mount R78**. The value should be adjusted based on driver output impedance and PCB Trace Impedance,,(33E is Nominal)

MODE	MODE-SEL-2	MODE-SEL-1	MODE-SEL-0
UART	0	0	0
SPI	0	0	1
USB	0	1	0

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.

Pin Name	Comments
WLAN_ACTIVE	Available from firmware version 2.6.1.1.7.7, If not used this pin should be left open.
BT_PRIORITY	Available from firmware version 2.6.1.1.7.7, If not used this pin should be grounded through 1k ohm resistor.
ADC1, ADC2	Not used in current firmware, should be left open.

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

Item	Qty	Reference	Part Value	Description	Jedec	Mfg	Part No
CAPACITORS							
1	1	C2	4.7nF 250V	CER CHIP C 4.7nF 10% B(JB) 0805 6.3V	0805	Murata	GRM219R72A472KA01B
2	1	C34	1000pF	CER CHIP C 1nF +/-10% X7R 0402 50V	0402	Murata	GRM155R71H102KA01D
3	2	C36,C61	0.1uF	CER CHIP C 0.1U 10% X5R 0402 6.3V	0402	Murata	GRM155R61A104KA01D
4	1	C35	10uF	CER CHIP C 10U 20% X5R 0805 10V	0805	Murata	GRM21BR61A106KE19L
5	1	C60	100uF	CAP TANT 100UF 6.3V 20% 1210	Case B	AVX	TCJB107M006R0070
RESISTORS							
6	10	R13,R40,R41,R42,R47,R105, R107,R109,R112,R114	0E	CHIP RES 0.0R 5% 200PPM 0402 1/10W	0402	Panasonic	ERJ-2GE0R00X
7	3	R26,R27,R78**	33E	CHIP RES 33R 5% 200PPM 0402 1/10W	0402	Panasonic	ERJ-2GEJ330X

8	2	R6,R7	820E	CHIP RES 820 5% 200PPM 0402 1/10W	0402	Panasonic	ERJ-2GEJ821X
9	1	R44	1K	RES 1.0K OHM 1/16W 5% 0402 SMD	0402	Yageo	RC0402JR-071KL
10	1	R43	2K	CHIP RES 2K 1% 100PPM 0402 1/10W	0402	Panasonic	ERJ-2RKF2001X
11	1	R76	4.7K	CHIP RES 4K7 5% 200PPM 0402 1/10W	0402	Panasonic	ERJ-2GEJ472X
12	3	R108,R115,R116	10K	CHIP RES 10K 5% 200PPM 0402 1/10W	0402	Panasonic	ERJ-2GEJ103X
13	1	R14	20K	CHIP RES 20K 1% 100PPM 0402 1/16W	0402	Yageo	RC0402FR-0720KL
14	1	R12	100K	CHIP RES 100K 5% 200PPM 0402 1/10W	0402	Panasonic	ERJ-2GEJ104X
15	1	R25	1M	CHIP RES 1M 5% 200PPM 0402 1/10W	0402	Panasonic	ERJ-2GEJ105X
DIODES							
16	2	D5,D6	LED	Green LED	0603	Lite-On Inc	LTST-C190YKT
IC'S							
17	1	U4	FDC6329L	DC-DC Switch	SSOT-6	Fairchild Semi.	FDC6329L
18	1	U1	RS-WC-201			Redpine	RS-WC-201
Miscellaneous							
19	1	PB1		2 Pin headers		Burg	
20	1	SW5		SW DIP-4 Positions	SMD	E-Switch	KAJ04LGGT
21	1	J5		Micro USB Connector	SMD	Molex	54819-0572
Do Not Populate							
22	6	C71,C72,C73,C74,C75,C76	0.1uF	CER CHIP C 0.1U 10% X5R 0402 6.3V	0402	Murata	GRM155R61A104KA01D
23	3	R77,R103,R110	0E	CHIP RES 0.0R 5% 200PPM 0402 1/10W	0402	Panasonic	ERJ-2GE0R00X
24	1	R52	1.5K	RES 1.5K OHM 1/10W 5% 0402 SMD	0402	Panasonic	ERJ-2GEJ152X
25	2	R11,R102	10K	CHIP RES 10K 5% 200PPM 0402 1/10W	0402	Panasonic	ERJ-2GEJ103X
26	2	R39,R47	15K	CHP RES 15K 5% 200PPM 1/10W 0402	0402	Panasonic	ERJ-1GEJ153C
27	2	FB5,FB7	BEAD	Ferrite Beads	0805	Murata	BLM21PG221SN
28	1	U18		UART Level Shifter	SSOP	Texas Instr.	MAX3232CDBR
29	1	U30		ADM708SAR	SOIC	Analog device	ADM708SAR
30	1	P1		CONNECTOR DB9_FEMALE PCB Mount			
31	1	SW1		DPDT		CTS Corp.	204-221ST
32	1	SW6		SW PUSHBUTTON	Dip	OMRON	

Notes:

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 90uF 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 220uF 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

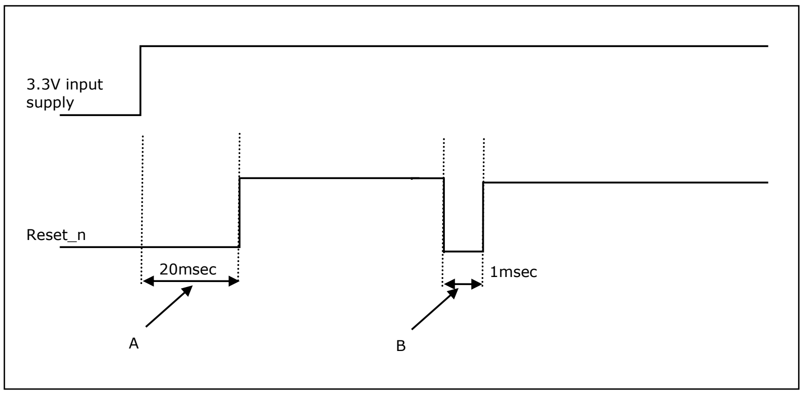


Figure 2: Reset Timing

2 RS-WC-201 PCB Landing Pattern

TOP VIEW

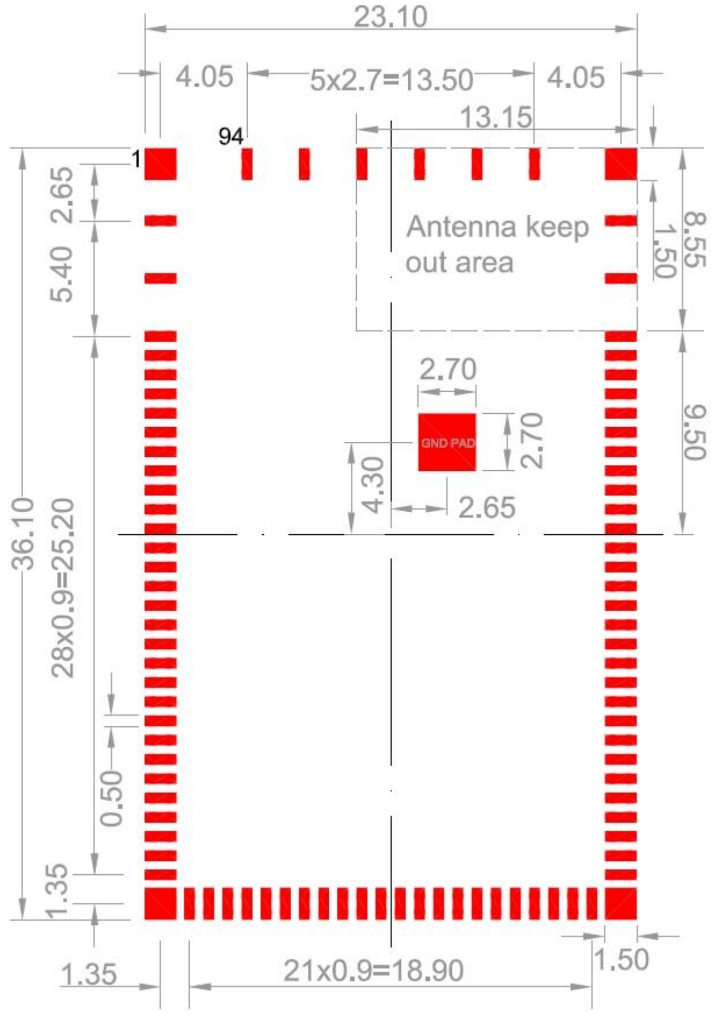


Figure 3: PCB Landing Pattern for RS-WC-201

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.

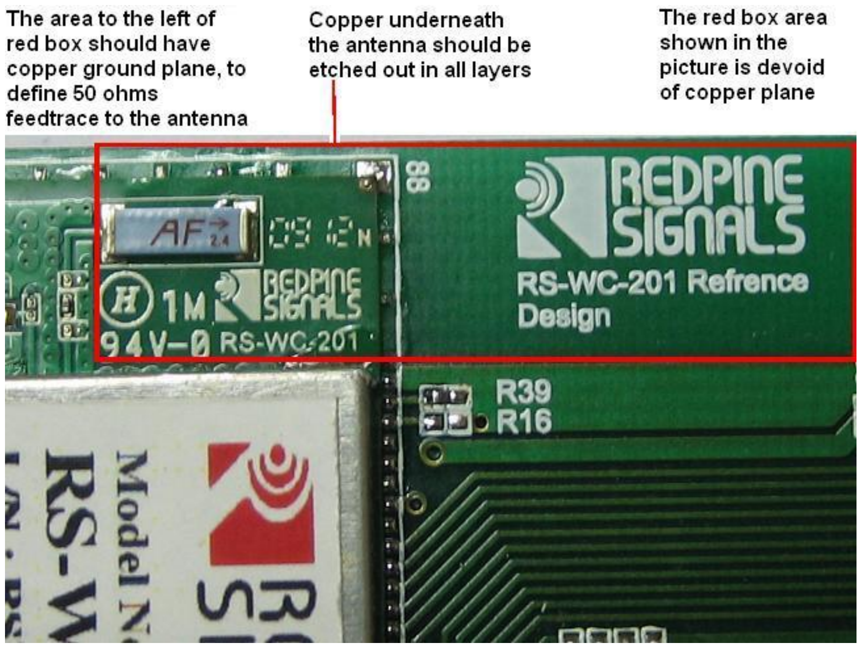
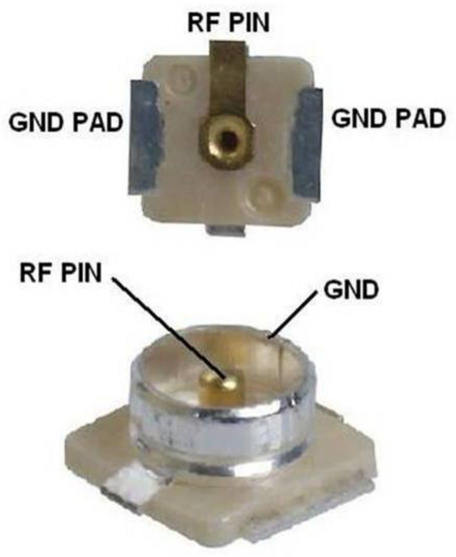
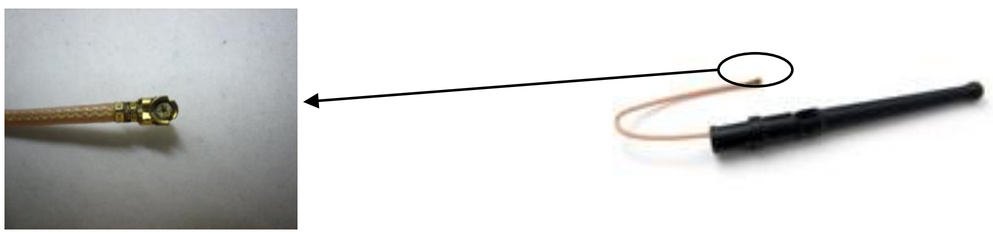

The area to the left of red box should have copper ground plane, to define 50 ohms feedtrace to the antenna
Copper underneath the antenna should be etched out in all layers
The red box area shown in the picture is devoid of copper plane
REDPINE SIGNALS
RS-WC-201 Refrence Design
R39
R16

Figure 4: Copper Etching Guidelines

4 u.FL Connector Recommendations



uFL Connector (Available in the module)



u. FL Mating Connector

External Antenna

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.

Manufacturer	**Part No**
Laird Technologies	0600-00040
Digi International	A24-HABUF-P5I
Connectblue	cB-ACC-27
Connectblue	cB-ACC-29

RS-WC-201/301

Software Programming Reference Manual

Version 2.01

August 2012

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

3 RS-WC-201/301 in UART Mode

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

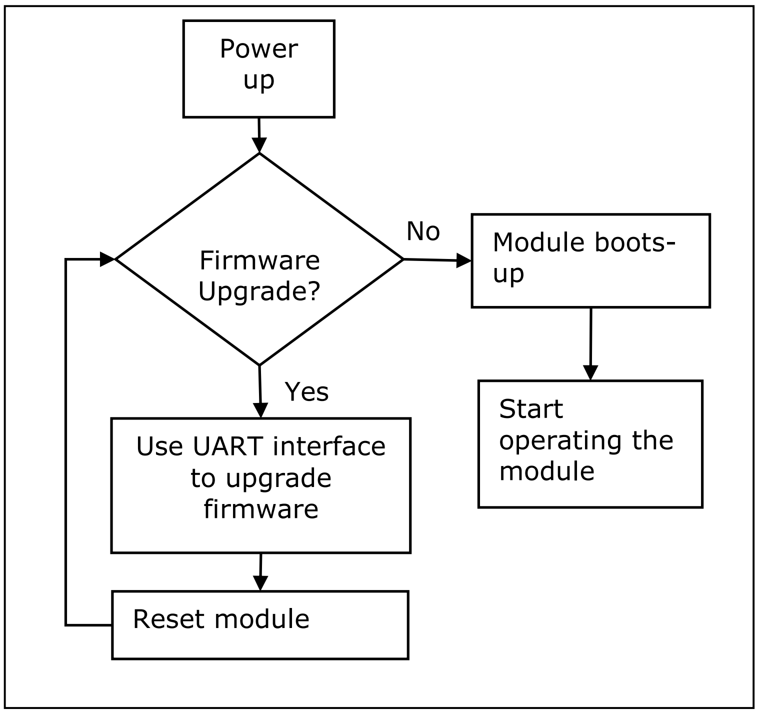

Power up
Firmware Upgrade?
No
Module boots-up
Start operating the module
Yes
Use UART interface to upgrade firmware
Reset module

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:

[0xFC 0xFC 0xFC 0xFC 0xFF Welcome to WiSeConnect\r\n]
………..
0xFC 0xFC 0xFC 0xFC 0xFF 0x57 0x65 0x6C 0x63 0x6F 0x6D 0x65 0x20 0x74 0x6F 0x20 0x57 0x69 0x53 0x65 0x43 0x6F 0x6E 0x6E 0x65 0x63 0x74 0x0D 0x0A

3. After boot-up is complete, the module issues a message

   READY\r\n>

   ………
   0x52 0x45 0x41 0x44 0x59 0x0D 0x0A

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

1Overview

1.1 Module Overview

RAK411 module is aWi-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 supportsstoring parameters, and by the customer commands it determines whether to enable automaticnetworking to realize easy networking and reduce time for system to networking. The module has built-in WEB server, supporting wireless network parametersconfiguration, supporting wireless firmwareupgrade. 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 lowpower design.

1.2 Key Applications

- Portable products
- Home appliances and electricalappliances
- 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
- SupportStation, 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

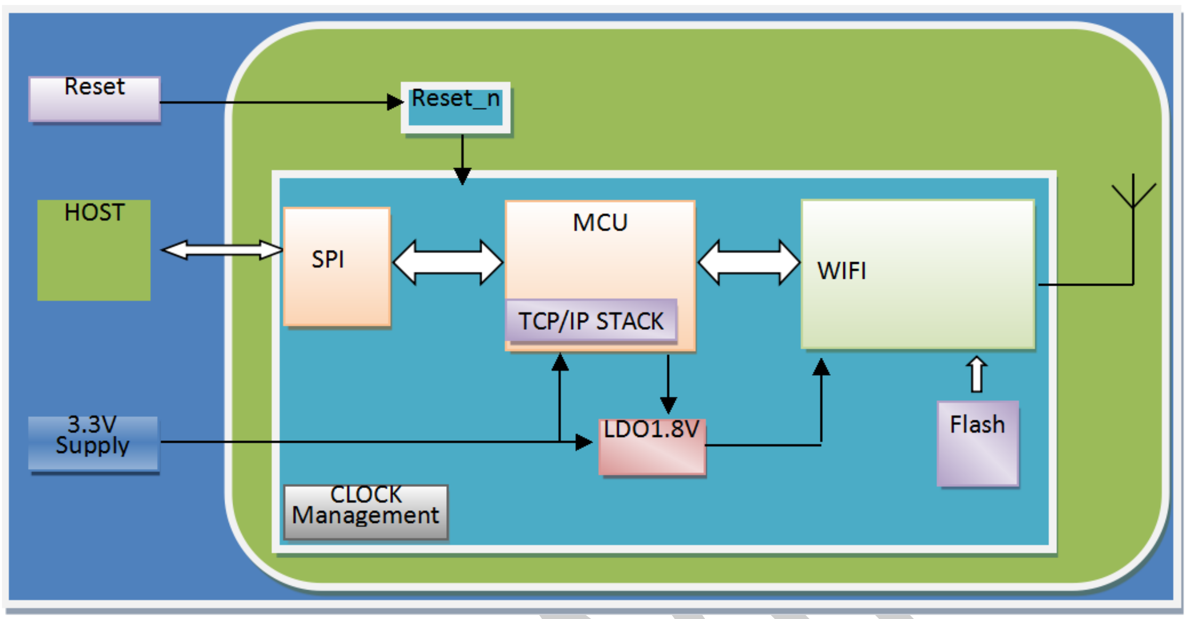


Figure 1-1 RAK411 System Diagram

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, withapprox 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

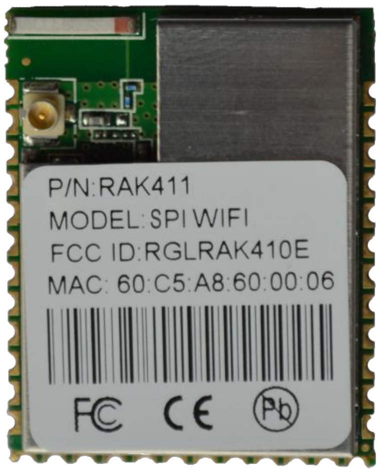


Figure 3-1 RAK411 Top View

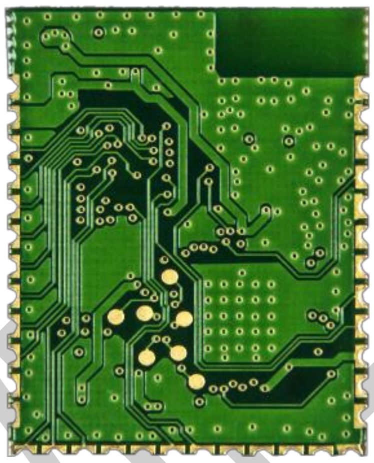


Figure 3-2 RAK4115 Bottom View

3.2 Pin Definition

Table 3-1: Pin Definition

Pin Serial No.	Name	Type	Description
1	GND	Ground	connected to ground pad or the copper
2	VCC3V3	Power	3.3V power supply
18	LINK	O , PU	"0" - STA connected in AP mode, Connected to router in STA mode "1" - disconnected Remain disconnected when no use
19	RESET	I , PU	Module reset pin, low effective
27	SPI_INT	O	SPI mode interrupt pin “0”——idle level “1”——has data sent to host
31	SPI_MISO	O	SPI slave: data of SPI Master Input, Slave Output
32	SPI_MOSI	I	SPI slave: data of SPI Master Output, Slave Input

33	SPI_CLK	O	SPI slave: SPI clock input
34	SPI_CS	I	SPI slave: SPI chip select input
3,4,6,7,8,9,10,11,14,15,16,17,20,21,22,23,24,26,28,29,30)	NC	NC	Remain disconnected when no use

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

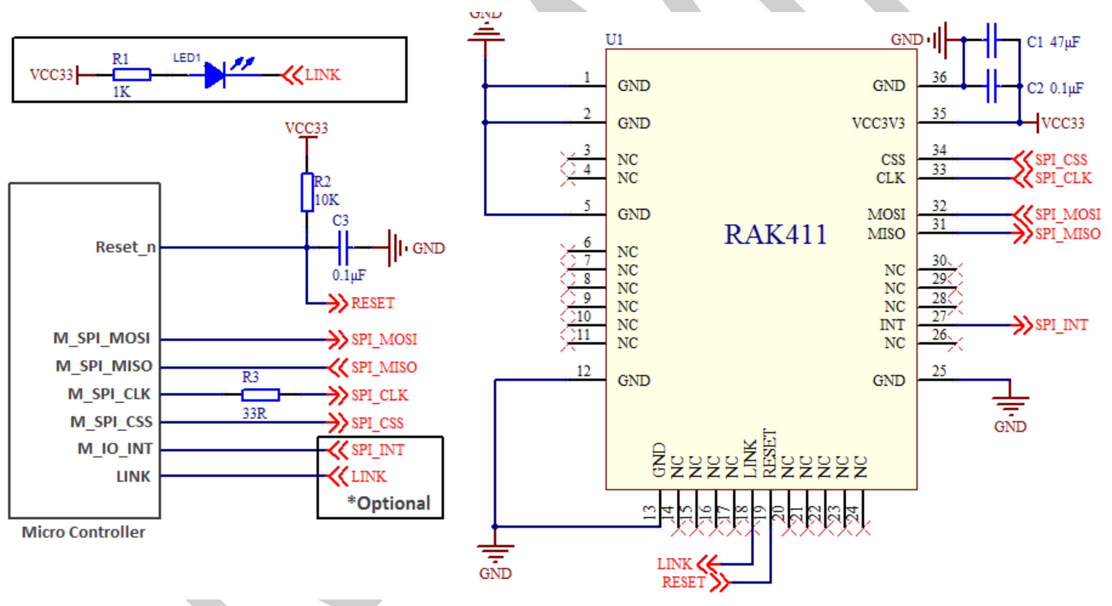


Figure 3-3 Module Typical Design Reference

Note: Upon SPI interface, the value of R3 depends onoutput resistance of host and PCB trace resistance, the default value is 33 Euro.

3.4 RAK411 PCBMechanical Size

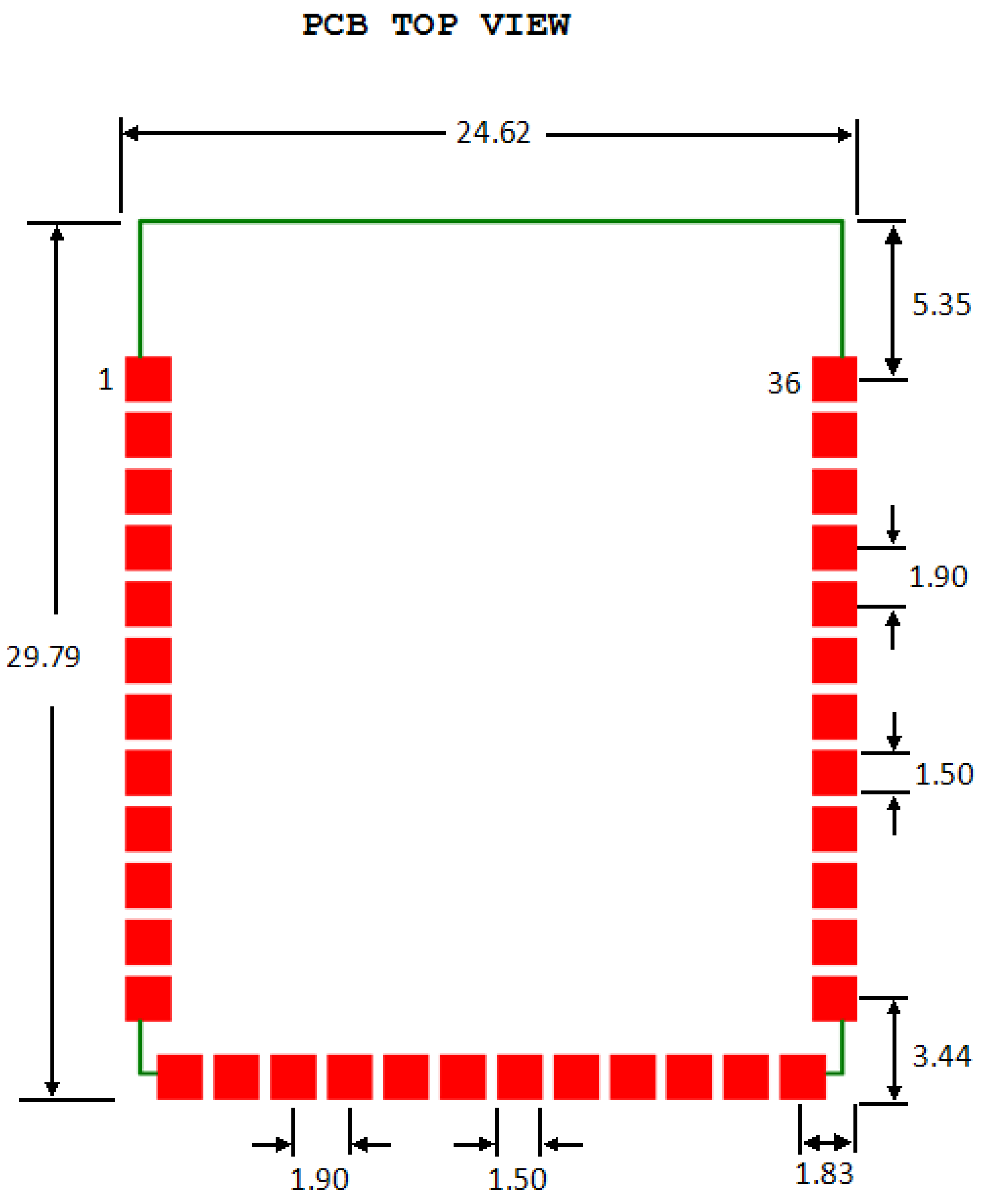


Figure 3-4: PCB Mechanical Size (mm)

3.5 Reflow Soldering Temperature Graph

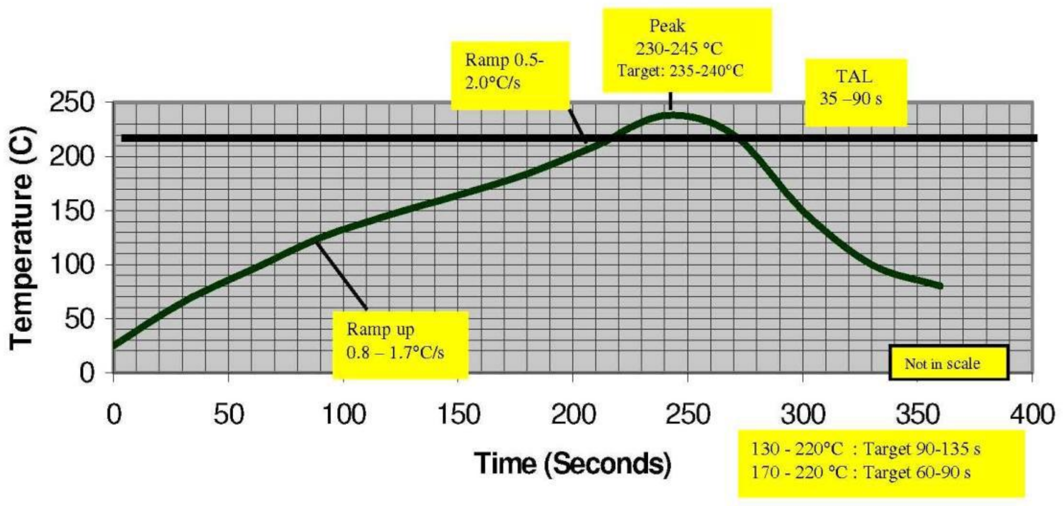


Figure 3-5: Temperature Graph

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 ℃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 avoiddamages to the module and the device,please operate under specified conditions.

Table 4-1: Parameters and Value Range

Parameters	Symbols	Value	Unit
External supply voltage	VCC3V3	-0.3~4.0	V
Maximum RF Input (Reference: 50Ω)	RF_{in}	+10	dBm
When voltage is 3.3V, IO Max voltage	$3V3V_{in}IOMax$	VCC+0.3	V
When voltage is 3.3V, IO Min voltage	$3V3V_{in}IOMin$	-0.3	V
Storage ambient temperature	T_{store}	-65~+135	℃
ESD resistance	ESD_{HBM}	2000	V

4.2 Recommended Operating Parameters

Table 4-2: Recommended Operating Parameter Range

Parameters	Symbols	Min Value	Typical Value	Max Value	Unit
External voltage	V_{cc}	3.14	3.3	3.46	V
Ambient temperature	$T_{ambient}$	-40	--	+85	℃

4.3 RF Electrical Characteristics

- **RF Transmit Specifications**

Table 4-3: Partial RF Transmit Specifications

Symbol	Parameter	Conditions	Typical Value	Unit
Ftx	Frequency range	--	2.4	GHz
Pout	Output power	--	--	--
	802.11b	1Mbps	17	dBm
	802.11g	6Mbps	17	dBm
	802.11n,HT20	MCS0	17	dBm
	802.11g,EVM	54Mbps	14	dBm
	802.11n,HT20EVM	MCS7	10	dBm

- **RF Receiver Specifications**

Table 4-4: Partial Receiver Specifications

Parameter	Parameter	Test conditions	Typical Value	Unit
Receiver sensitivity	11b,1Mbps		-97	dBm
	11b,2Mbps		-92	dBm
	11b,5.5Mbps		-90	dBm
	11b,11Mbps		-88	dBm
	11g,9Mbps		-91	dBm
	11g,18Mbps		-87	dBm
	11g,36Mbps		-81	dBm
	11g,54Mbps		-75	dBm
	11n,MCS1,13Mbps		-89	dBm

	11n,MCS3,26Mbps		-82	dBm
	11n,MCS5,52Mbps		-75	dBm
	11n,MCS7,65Mbps		-72	dBm
Maximum input signal	CH7	11g,54Mbps	10	dBm
Adjacent channel	6Mbps		37	dBc
	54Mbps		21	dBc
	MCS0		38	dBc
	MCS7		20	dBc

4.4 MCU Reset

Figure 4-1 shows theMCU 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.

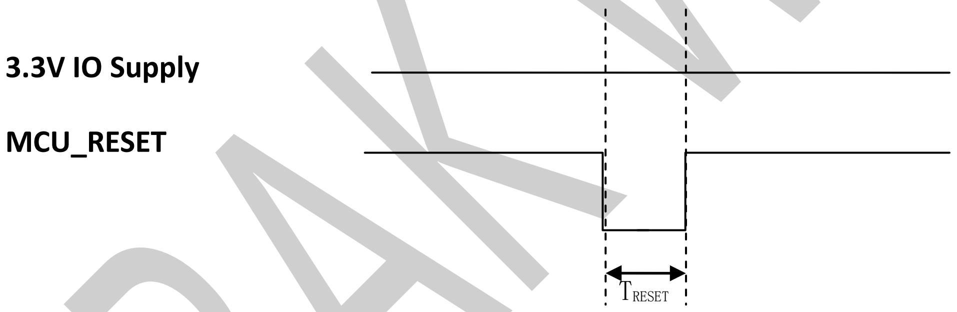


Figure 4-1: MCU Reset Timing

Table 4-5 shows thedescription of MCU reset parameters.

Table 4-5: MCU Reset Parameter

Symbol	Description	typical（mS）
T_{RESET}	MCU reset pulse length	>10

5 Order Information

5.1 Products

Table 5-1: Product Models

Product	Description	Packaging
RAK411A	SPI interface module, with on-board antenna	27pcs/tray
RAK411B	SPI interface module, with external antenna	27pcs/tray

5.2 Size

Packaging: Hard plastic pallets

Weight: <=3.00g/pcs

Table 5-2: Thickness (Height)

RAK411	Thickness (Height)
Without shield holder	2.85±0.05mm
With shield holder	2.95±0.05mm
With shield	3.30±0.15mm

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,Shengzhen

7 Revision History

Version	Modifications	Date
V1.0	Initial Draft	2014-02-08
V1.1	Optimized power consumption to 1-2uA	2014-02-17

RAK411 SPI-WIFI Module Programming Manual V1.2

Beijing Rakwireless Technology Co.,Ltd

info@rakwireless.com

QQ:1395415717 QQ:1930154262

www.rakwireless.com

Content

1 Overview

1.1 Module Introduction

RAK411 module is aWi-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 supportsstoring parameters, and by the customer commands it determines whether to enable automaticnetworking to realize easy networking and reduce time for system to networking. The module has built-in WEB server, supporting wireless network parametersconfiguration, supporting wireless firmwareupgrade. It also supports WPS and EasyConfigone-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 lowpower 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
- SupportStation, 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 EasyConfigone-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 300m
- 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

- Small package size: 28.75mmX23.14mmX3.40mm
- FCC, RoHS and CE compliant

1.3 Key Applications

- Portable products
- Home appliances and electricalappliances
- 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 EasyConfigone-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, withapprox 80ma average power consumption, peak current less than 200ma

➢ Power-saving mode, with approx 10ma average power consumption, peak current <200ma, DTIM = 100ms

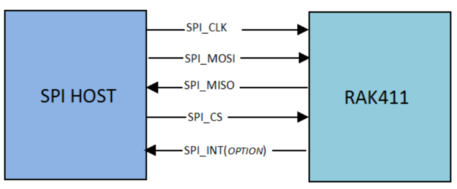
➢ Deep sleep mode, with approx 5ma average power consumption, peak current <200ma, DTIM = 100ms

➢ Standby mode, with power consumption<2uA

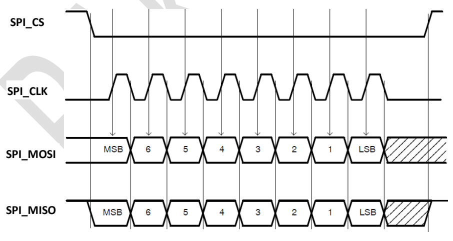
3 SPI Interface

RAK411 communicates with the hostthrough 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-------------------------------------SCK 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 sent8 BIT

MODE--Data length is 8

bitsCS---Slave selective signal is effective low

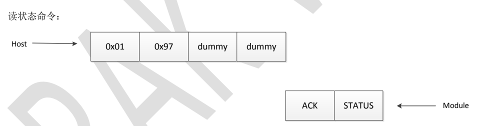
3.3 Interrupt Pin

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.

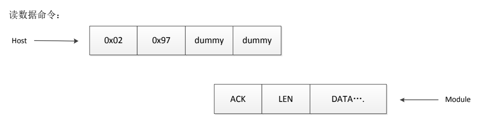
3.4.1Read Status



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 0xA1, Byte1: fixed value 0x97; Byte2: invalid data, must be 0; Byte3: invalid data, must be 0.

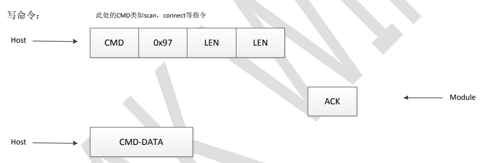
3.4.2 Read Data



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 0xA2, Byte1: fixed value0x97; Byte2: invalid data, must be 0; Byte3: invalid data, must be 0.

3.4.3 Write Data



Description:

Host sends commands to drive module for various operations. Host sends command and wait for the module's reply of ACK. If ACK is received,the module continues to send the detailed CMD+DATA, where the CMD are the commands such as scan, connect, etc. Host's command Byte0: command CMD, Byte1: fixed value0x97; Byte2: invalid data, must be 0; Byte3: invalid data, must be 0.

3.4.4 Status Register

STATUS

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Reserve	Reserve	Reserve	Reserve	Reserve	Reserve	Recv_full	Data_flag

Description:

Here is the byte of status read, the last two bits representative the data type of module feedback.

Bit 7: SPI_STATUS
Bit 6: reserve
Bit 5: reserve
Bit 4: reserve
Bit 3: reserve
Bit 2 reserve
Bit 1: Buffer_full=1，The internal data cache area of module is full, valid when SPI_STATUS is 1
Bit 0: Data_flag=1，Module has data to inform host (response of command or received data), valid when SPI_STATUS is 1

3.4.5 Error code

When the command frame error, will prompt the error code returned,detailed as follows:

Code	description
-1	parameter input error (parameter cannot identify / missing parameter / order is too long or other illegal parameter)
-11	system error (restart module)
-12	fatal error (contact factory)
Other	details of specific commands

3.5 Boot

RAK411's reboot 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 63 6F 6D 65 20 74 6F 20 52 41 4B 34 31 31

3.6 Power Mode

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

Mode	Control Part	Wireless Part	Wake-up Type	Min Power Consumption (AP)
0	Normal	Normal	No need	80mA
1	Sleep_Mode	Power_Save	No need	10mA
2	Deep_Sleep	Power_Save	SPI wake up	5mA
3	Deep_Sleep	Shut_down	SPI wake up, Reset	2uA

pwrmode=0-------------Mode 0

Module works under the maximum performance, control part and wireless part are fully opened.

pwrmode=1-------------Mode 1

The control part enters into shallow sleep, the wireless part maintains the current connection status, and enters a low-power mode. The communication is normal, but this will reduce performance of module, the speed of sending and receivingis reduced.

pwrmode=2-------------Mode 2

The control part enters into a deep sleep, the wireless part maintains the current wireless connection status, and enters a low-power mode.The remote data or host initiates communications to wakeup control part, thenentersinto mode 1. If no sending and receiving data, it automatically enters into Mode 2.

pwrmode=3-----------Mode 3

When enters into this mode, module firstly saves current connective status to RAM, and shuts down the power of wireless part, then control part enters into deep state. In this state, module cannot respond to any command or wireless data, lowering consumption to minimum.User can initiate communication or reset module. It enters into mode 0 by default after start.

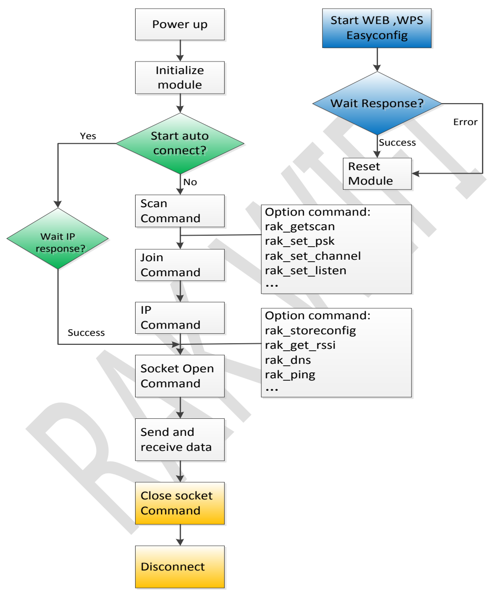
3.7 Operational Process

RAK411 SPI command operation feature completes a few basic steps of WIFI communication, including network scanning, joining network and obtaining an IP address, and eventually establishing Socket communication. RAK411 provides a variety of convenient operation to implement networking, so that customers can easily complete the network configuration, and concentrate on the managementof socketand their own data protocols.

To realize automatic networking management, customers can take advantage of WEB, WPS and EasyConfig configuration features. The module will automatically store paramenters after a successful configuration, and these automatic networking

commands can be used any time, letting the module automatically complete networking operation, and returning the results.

The basic operation of the process is as follows:



4. Command Encyclopedia

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

Command	Description
Module Management Commands	
rak_sys_init	Initialize module, read boot information
rak_get_version	Check software version
rak_setpwrmode	Set module power mode
rak_read_status	Read module status information
rak_reset	Reset module
Network Operation Commands	
rak_scan	Scanwireless networks
rak_getscan	Reads a specified number of scan results
rak_set_psk	Set network password
rak_set_channel	Set network channel
rak_connect	Connect wireless network
rak_set_ipstatic	Configure Static IP Address
rak_ipconfig_dhcp	Setting DHCP Mode
rak_easy_config	Connecting network by Easyconfig
rak_wps	Connecting Network byWPS
rak_get_con_status	Get network connection status
rak_ipconfig_query	Query module IP information
rak_get_rssi	Get network signal strength of module
rak_dns	DNS
rak_ping	Ping hosts in the network
rak_apconfig	AP Network Advanced Settings
rak_set_listen	Set network listening intervals
rak_disconnect	Disconnect the current wireless network
Socket Operation Commands	
rak_udp_client	Establish UDP client
rak_udp_server	EstablishUDP Server
rak_tcp_client	Establish TCP client
rak_tcp_server	Establish TCP Server
rak_multicast	Create UDP multicast communication
rak_socket_close	Close an opened socket handle
rak_send_data	Send data to an opened socket handle
rak_read_data	Read command returns / network data / network information
Save parameters commands	
rak_storeconfig_data	Storenetwork configuration parameters
rak_storeconfig	Store the current network parameters
rak_web_store	Storeweb serverbuilt-in parameters
rak_auto_connect	Enable automatically connecting
rak_start_web	Start web server and configure module
rak_get_storeconfig	Get saved network parameters
rak_get_webconfig	Get web server built-in parameters

4.1 Module Management Commands

4.1.1 Initializing Module

Command:
rak_sys_init
Syntax:
uint32_t cmd;
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xA0,0x97,0x00,0x00	Command Code

Description:
It is used to initialize module, and read boot information.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xA0,0x00	Response Code
<DATA>	17	Welcome to RAK411	Welcome string
<STATUS>	1	0	Command successful
		-2	Command faile

4.1.2 Checking Software Version

Command:
rak_get_version
Syntax:

uint32_tcmd;
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xBE,0x97,0x00,0x00	Command Code

Parameter:
It is used to check module versions, including versions of host and WLAN.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xBE,0x00	Response Code
<HOST_FW>	8	ASCII	Host version
<->	1	-	Version delimiter
<WLAN_FW>	6	ASCII	Wlan version
<STATUS>	1	0	Command successful
		-2	Command failed

4.1.3 Setting PowerMode

Command:
rak_setpwrmode
Syntax:
```
typedefstruct {
uint32_t cmd;
uint32_t powermode;
}rak_pwr_mode_t;
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xB8,0x97,0x00,0x00	Command Code
powermode	4	0--3,0x00,0x00,0x00	Power mode

Parameter:
It is used to set module power mode.
Return Value:
N/A

4.1.4 ReadingModule Status

Command:
  rak_read_status
Syntax:
  uint32_t cmd;
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xA1,0x97,0x00,0x00	Command Code

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

Return Value:

Parameter	Bytes	Value	Description
<STATUS>	1	0x81	To read data
		0x82	Data cache is full

4.1.5 Reset

Command:
  rak_reset
Syntax:
  uint32_t cmd;
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xBF,0x97,0x00,0x00	Command Code

Parameter:
  It is used to resets the entire module via command.
Return Value:
  N/A

4.2 Network Operation Commands

4.2.1 Scanning Wireless Network

Command:
  rak_scan
Syntax:
```
typedefstruct {
uint32_tcmd;
uint32_t    channel;
charssid[32];
} rak_scan_t;
```
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xA3,0x97,0x00,0x00	Command Code
channel	4	0—13,0x00,0x00,0x00	Scanning specified channel(s), scan all channels if value is 0
Ssid	32	Wireless Network Name	Specified SSID, scan all channels if value is null

Parameter:
  It is used to scan wireless networks via command.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xA3,0x97,0x00,0x00	Response Code
<AP_NUM>	2	0x08,0x00	Scanned 8 WLANs
<STATUS>	1	0	Command successful
		-2	Command failed

4.2.2 Getting Scanned Information

Command:
  rak_getscan
Syntax:

```
typedefstruct {
```

```
uint32_tcmd;
uint32_t    scan_num;
}rak_getscan_t;
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xA4,0x97,0x00,0x00	Command Code
Scan_num	4	1—10,0x00,0x00,0x00	Get the number of network information, if scan_numis greater than the actual number of scanned network information, then returns the actual number

Parameter:
  It is used to get the scanned network information.

Return Value:

Parameter	Bytes	Value	Description							
<CODE>	2	0xA4	Response Code							
<CHANNEL>	1	1--14	Channel							
<RSSI>	1	-99--0	Channel intensity (negative value)							
<SEC_MODE>	1		Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
			Wpa2	Wpa	Wep	802.1x	Psk	Wep	Tkip	Ccmp
<SSID_LEN>	2		Length of SSID							
<SSID>	32		SSID							
<BSSID>	6		BSSID							
<STATUS>	1	0	Command successful							
		-2	Command failed							

4.2.3 Setting Password

Command:
  rak_set_psk
Syntax:
```
typedefstruct {
uint32_tcmd;
```

```
charpsk[64];
}rak_psk_t;
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xA5,0x97,0x00,0x00	Command Code
Psk	64		Network password

Parameter:
  It is used to set network password.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xA5,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Command failed

4.2.4 Setting Channel

Command:
  rak_set_channel
Syntax:
```
typedefstruct {
uint32_tcmd;
uint32_t channel;
}rak_channel_t;
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xAB,0x97,0x00,0x00	Command Code
channel	4	1--13	SettingAP / Ad-Hoc channel

Parameter:
  It is used to set network channel.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xAB,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Command failed

4.2.5 Connecting Wireless Network

Command:
rak_connect
Syntax:

```
typedefstruct {
uint32_tcmd;
uint32_t    mode;
charssid[32];
}rak_conn_t;
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xA6,0x97,0x00,0x00	Command Code
mode	4	0=station	Select network mode
		1=ap	
		2=ad-hoc	
Ssid	32	ASCII	SSID

Parameter:
It is used to connect an AP/ADHOC network or establish a specified AP Network.
Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xA6,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Cannot find the SSID
		-3	Command failed

4.2.6 Configuring Static IP Address

Command:
rak_set_ipstatic

Syntax:

```
typedefstruct {
uint32_tcmd;
uint32_taddr;
uint32_t mask;
uint32_t gw;
uint32_t dnssvr1;
uint32_t dnssvr2;
}rak_ipstatic_t;
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xAD,0x97,0x00,0x00	Command Code
addr	4	0xC0,0xA8,0x07,0x01	IP address
mask	4	0xFF,0xFF,0xFF,0x00	subnet mask
gw	4	0xC0,0xA8,0x07,0x01	gateway
dnssvr1	4	0xC0,0xA8,0x07,0x01	DNS server 1
dnssvr2	4	0x00,0x00,0x00,0x00	DNS server 2

Parameter:
This command is used to assign static IP address for module.
Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xAD,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Command failed

4.2.7 Setting DHCP Mode

Command:
rak_ipconfig_dhcp
Syntax:

```
typedefstruct {
uint32_tcmd;
uint32_t    mode;
```

```
} rak_ipdhcp_t ;
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xAC,0x97,0x00,0x00	Command Code
mode	4	0= DHCP CLENT	Select DHCP Mode
		1= DHCP SEVER	

Parameter:
This command is used to set DHCP working mode.
Return Value:

DHCP SERVER

Parameter	Bytes	Value	Description
<CODE>	2	0xAC,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Command failed

DHCP CLENT:

Parameter	Bytes	Value	Description
<CODE>	2	0xAC,0x00	Response Code
<MAC>	6		MAC address
<ADDR>	4		IP address
<MASK>	4		Subnet mask
<GW>	4		Gateway
<DNS1>	4		DNS server 1
<DNS2>	4		DNS server 2
<STATUS>	1	0	Command successful
		-2	Command failed

4.2.8 Connecting Network by Easyconfig

Command:
rak_easy_config
Syntax:
uint32_tcmd;

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xC2,0x97,0x00,0x00	Command Code

Parameter:
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:

Parameter	Bytes	Value	Description							
<CODE>	2	0xC2	Response Code							
<SSID>	32	ASCII	SSID							
<SEC_MODE>	1		Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
			Wpa2	Wpa	Wep	802.1x	Psk	Wep	Tkip	Ccmp
PSK	64	ASCII	Network password							
<STATUS>	1	0	Command successful							
		-2	cannot find the ap							
		-3	Failed to be added in router							
		-4	Failed to get dynamic IP							
		-6	Easy config failed							

4.2.9 Connecting Network by WPS

Command:
rak_wps
Syntax:
uint32_tcmd;
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xC3,0x97,0x00,0x00	Command Code

Parameter:
This command initiates module WPS function. When successful, module automatically saves the parameters via rak_storeconfigcommand.

Return Value:

Parameter	Bytes	Value	Description							
<CODE>	2	0xC3,0x00	Response Code							
<SSID>	32	ASCII	SSID							
SEC_MODE	1		Bit7	Bit 6	Bit5	Bit4	Bit 3	Bit 2	Bit1	Bit0
			Wpa 2	W pa	We p	802. 1x	Ps k	W ep	Tkip	Ccm p
Psk	64	ASCII	Network password							
<STATUS>	1	0	Command successful							
		-2	Cannot find AP							
		-3	Join the router failure							
		-4	Ip failed to get							
		-5	Wps failed							

4.2.10 Getting Network Connection Status

Command:

rak_get_constatus

Syntax:

uint32_tcmd;

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xA7,0x97,0x00,0x00	Command Code

Parameter:

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:

Parameter	Bytes	Value	Description
<CODE>	2	0xA7,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Command failed

4.2.11 Querying module IP information

Command:

rak_ipconfig_query

Syntax:

uint32_tcmd;

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xAE,0x97,0x00,0x00	Command Code

Parameter:

It is used to get module IP information, including MAC address, IP address, subnet mask, gateway, and DNS server.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xAE,0x00	Response Code
<MAC>	6		MAC address
<ADDR>	4		IP address
<MASK>	4		Subnet mask
<GW>	4		Gateway
<DNS1>	4		DNS Server 1
<DNS2>	4		DNS Server 2
<STATUS>	1	0	Command successful
		-2	To obtain IP information failed

4.2.12 Getting Network Signal Strength

Command:

rak_get_rssi

Syntax:

uint32_tcmd;

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xA9,0x97,0x00,0x00	Command Code

Parameter:

It is used to get the current signal strength of the network.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xA9,0x00	Response Code
<RSSI>	2	-99--0	current signal strength
<STATUS>	1	0	Command successful
		-4	To obtain IP information failed

4.2.13 DNS

Command:

rak_dns

Syntax:

```
typedefstruct {
uint32_t cmd;
uint32_t name[42];
}rak_dns_t;
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xAF,0x97,0x00,0x00	Command Code
name	<42	ASCII	Domain

Parameter:

It is used to convert domain to the corresponding IP address, the domain must be configured available DNS server.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xAF,0x00	Response Code
<ADDR>	4		Ip address
<STATUS>	1	0	Command successful
		-2	DNS resolution failure

4.2.14 PING

Command:

rak_ping

Syntax:

```
typedefstruct {
uint32_t cmd;
uint32_t hostaddr;
uint32_t count;
uint32_tsize;
}rak_ping_t;
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xB0,0x97,0x00,0x00	Command Code
hostaddr	4	String	Specified host
count	2		Number of packets
size	2	1--1400	Packet size, maximum 1400byte

Parameter:

It is used torun the ping command.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xB0,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Cannot access host

4.2.15 AP Network Advanced Settings

Command:

rak_apconfig

Syntax:

```
typedefstruct {
uint32_tcmd;
uint8_t    hidden;
uint8_t    countryCode[2];
}rak_apconfig_t;
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xAA,0x97,0x00,0x00	Command Code
hidden	1	0	Network name is visible
		1	Network name is hidden
countryCode	2	Country code	Country code, e.g. CN

Parameter:
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:

Parameter	Bytes	Value	Description
<CODE>	2	0xAA,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Command failed

4.2.16 Setting Network Listening Intervals

Command:
rak_set_listen
Syntax:
```
typedefstruct {
uint32_tcmd;
uint32_t time;
}rak_beacon_t;
```
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xA8,0x97,0x00,0x00	Command Code
time	4	20--1000	Need to refer to the wireless router settings for specific parameters

Parameter:
It is used to set modulebeacon interval in Station mode.
Note:
In power saving mode, reducing power consumption can be realized

viaincreasing parameter values, but by this way it may cause delay in receiving wireless data.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xA8,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Command failed

4.2.17 DisconnectingCurrent Wireless Network

Command:
rak_disconnect
Syntax:
```
uint32_tcmd;
```
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xB7,0x97,0x00,0x00	Command Code

Parameter:
It is used to disconnect the current wireless network.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xB7,0x00	Response Code
<STATUS	1	0	Command successful
		-2	Current network is disconnected

4.3 Socket Operation Commands

4.3.1 TCP Server

Command:
rak_tcp_server
Syntax:

```
typedefstruct {
uint32_tcmd;
uint16_t dummy;
uint16_t port;
}rak_server_t;
```
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xB4,0x97,0x00,0x00	Command Code
dummy	2		Invalid data
port	2	1-65535	Local port number

Parameter:
Module is used as a TCP server and creates a listening port, if the operation is successful, the module will return a hexadecimal identifier that is used to manage the connection. This command can create up to four connections.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xB4,0x00	Response Code
<SOCKET_FLAG>	2	0--7	Socket identifier
<STATUS>	1	0	Command successful
		-2	Failed to create
		-3	Failed to bind
		-4	Target port connection error

4.3.2 TCP Client

Command:
rak_tcp_client
Syntax:
```
typedefstruct {
uint32_tcmd;
uint32_t dest_addr;
uint16_t dest_port;
uint16_t local_port;
}rak_client_t;
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xB3,0x97,0x00,0x00	Command Code
dest_addr	4		Target IP address
dest_port	2	1-65535	Target port number
local_port	2	1-65535	Local port number

Parameter:
This command is to create a TCP CLIENT and connect with the remote TCP SERVER, if the operation is 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.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xB3,0x00	Response Code
<SOCKET_FLAG>	2	0--7	Socket identifier
<STATUS>	1	0	Command successful
		-2	Failed to create
		-3	Failed to bind
		-4	Target port connection error

4.3.3 UDP Client

Command:
rak_udp_client
Syntax:
```
typedefstruct {
uint32_tcmd;
uint32_t dest_addr;
uint16_t dest_port;
uint16_t local_port;
}rak_client_t;
```
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xB1,0x97,0x00,0x00	Command Code
dest_addr	4		Target IP address

dest_port	2	1-65535	Target port number
local_port	2	1-65535	Local port number

Parameter:

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.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xB1,0x00	Response Code
<SOCKET_FLAG>	2	0--7	Socket identifier
<STATUS>	1	0	Command successful
		-2	Failed to create
		-3	Failed to bind
		-4	Target port connection error

4.3.4 UDP Server

Command:
rak_udp_server
Syntax:
```
typedefstruct {
uint32_tcmd;
uint16_t dummy;
uint16_tport;
}rak_server_t;
```
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xB2,0x97,0x00,0x00	Command Code
dummy	2		Invalid data
port	2	1-65535	Local port number

Parameter:

This command is used to create a port in local and wait for dataremote 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 connectionsare 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:

Parameter	Bytes	Value	Description
<CODE>	2	0xB2,0x00	Response Code
<SOCKET_FLAG>	2	0--7	Socket identifier
<STATUS>	1	0	Command successful
		-2	Failed to create
		-3	Failed to bind
		-4	Target port connection error

4.3.5 CreatingMulticast Communication

Command:
rak_multicast
Syntax:
```
typedefstruct {
uint32_tcmd;
uint32_t dest_addr;
uint16_t dest_port;
uint16_t local_port;
}rak_client_t;
```
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xC4,0x97,0x00,0x00	Command Code
dest_addr	4	224.0.0.1-239.255.255.255	Target multicast IP address
dest_port	2	1-65535	Target port number
local_port	2	1-65535	Local port number

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:

Parameter	Bytes	Value	Description
<CODE>	2	0xC4,0x00	Response Code
<SOCKET_FLAG>	2	0--7	Socket identifier
<STATUS>	1	0	Command successful
		-2	Failed to create
		-3	Failed to bind
		-4	Target port connection error

4.3.6ClosingSocket

Command:
rak_socket_close
Syntax:
```
typedefstruct {
uint32_tcmd;
uint16_t dummy;
uint16_t flag;
}rak_close_t;
```
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xB5,0x97,0x00,0x00	Command Code
dummy	2		Invalid data
flag	2	0-7	Socket identifier

Parameter:

It is used to close the already opened socket identifier.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xB5,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Specified port does not exist
		-3	Failed to close

4.3.7 Sending Data

Command:
rak_send_data
Syntax:
```
typedefstruct {
uint32_tcmd;
uint32_t dest_addr;
 uint16_tdest_port;
 uint16_t         socket_flag;
uint16_tlen;
char buffer[1400];
}rak_send_t;
```
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xB6,0x97,0x00,0x00	Command Code
dest_addr	4		Target IP address
dest_port	2	1-65535	Target port
flag	2	0-7	Socket identifier
len	2	1--1400	Data length
buffer	1--1400		Data

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 datawithout 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 valuecan 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.

Return Value:
no

4.3.8 Receiving Data

Command:
rak_recv_data
Syntax:
```
uint32_tcmd;
```
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xB6,0x97,0x00,0x00	Command Code

Parameter:

This command is to read data commandsof 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	0xC8,0x00	Response Code
<SOCKET_FLAG>	2	0-7	Port descriptor
<DATA_LEN>	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	0xC9,0x00	Response Code
<SOCKET_FLAG>	2	0-7	Port descriptor
<dummy>	2		Invalid data
<IP_PORT>	2		IP address
<IP_ADDR>	2		Data
<STATUS>	1	0	Successful

<CODE> =0xCA socketdisconnected

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

<CODE> =0xCB network connected

Parameter	Bytes	Value	Description
<CODE>	2	0xCB,0x00	Response Code
<STATUS>	1	0	Successful

<CODE> =0xCC network disconnected

Parameter	Bytes	Value	Description
<CODE>	2	0xCC,0x00	Response Code
<STATUS>	1	0	Successful

4.4 Save Parameters Commands

4.4.1 StoringNetwork Configuration Parameters

Command:

rak_storeconfig_data

Syntax:

```
typedefstruct {
uint32_t    cmd
uint32_tfeature_bitmap;
uint8_t net_type;
uint8_t channel;
uint8_t sec_mode;
uint8_t dhcp_mode;
charssid[32];
charpsk[64];
ip_param_tip_param;
ap_config_tap_config;
}config_t
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xC0,0x97,0x00,0x00	Command Code
feature_bitmap	4	0x00,0x00,0x00,0x00	Switching features
net_type	1	0	Station
		1	Ap
		2	Ad-hoc
sec_mode	1	0	Network is not encrypted

		1	Network is encrypted
dhcp_mode	1	0	STA:Dhcp client
		1	STA:ip static
Ssid	32	ASCII	Network identifier
psk	64	ASCII	Network key
dummy	2		Null data
ip_param	20		IP parameters
ap_config	3		AP advanced parameters

Parameter:

It is used to save user parameters, including password, SSID, IP address, and scan information.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xC0,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Command failed

4.4.2StoringCurrent Network Parameters

Command:

rak_storeconfig

Syntax:

uint32_tcmd;

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xC1,0x97,0x00,0x00	Command Code

Parameter:

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:

Parameter	Bytes	Value	Description
<CODE>	2	0xC1,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Command failed

4.4.3 Modifying NetworkParameters

Command:

rak_web_store

Syntax:

```
typedefstruct {
uint32_t    cmd;
config_tparams;
charuser_name[17];
charuser_psk[17];
}web_t
```

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xC5,0x97,0x00,0x00	Command Code
params	126		Network parameters
User_name	17	ASCII	Web authentication user name
User_psk	17	ASCII	Web authentication password

Parameter:

It is used to save thenetwork parameters used to initiatenetwork.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xC5,0x00	Response Code
<STATUS>	1	0	Command successful
		-2	Command failed

4.4.4EnableingAutomaticConnection

Command:

rak_auto_connect

Syntax:

uint32_tcmd;

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xC6,0x97,0x00,0x00	Command Code

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:

Parameter	Bytes	Value	Description
<CODE>	2	0xC6,0x00	Response Code
<MAC>	6		MAC address
<ADDR>	4		IP address
<MASK>	4		Subnet mask
<GW>	4		Gateway
<DNS1>	4		DNS Server 1
<DNS2>	4		DNS Server 2
<STATUS>	1	0	Command successful
		-2	Specified SSID is not found
		-3	Failed to join router
		-4	Failed to allocate IP address

4.4.5StartingWeb Server

Command:
rak_start_web
Syntax:
uint32_tcmd;
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0xC7,0x97,0x00,0x00	Command Code

Parameter:

It is used to start the embedded WEB service. Module will start the WEBwith 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:

Parameter	Bytes	Value	Description
<CODE>	2	0xC7,0x00	Response Code
<STATUS>	1	0	Configuration is successful or upgrade is successful
		-2	Configuration timeout

4.4.6Getting Saved Network Parameters

Command:
rak_get_storeconfig
Syntax:
uint32_tcmd;

Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0Xb9,0x97,0x00,0x00	Command Code

Parameter:

This command is used to save network parameters.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xb9,0x00	Response Code
<feature_bitmap>	4		Featured parameters
<net_type>	1	0	Station
		1	Ap
		2	Ad-hoc
<sec_mode>	1	0	Network is not encrypted
		1	Network is encrypted
<dhcp_mode>	1	0	STA:DHCP client
		1	STA:ip static
<Ssid>	33		Network identifier
<psk>	65		Network key
<DUMMY>	2		Null data
<ip_param>	20		IP parameters

<ap_config>	3		AP parameters
<STATUS>	1	0	Command successful
		-2	Command failed, or timeout

4.4.7 GettingWeb Server Built-in Parameters

Command:
Rak_get_webconfig
Syntax:
uint32_tcmd;
Parameter:

Parameter	Bytes	Value	Description
Cmd	4	0Xba,0x97,0x00,0x00	Command Code

Parameter:

This command is available to save network parameters.

Return Value:

Parameter	Bytes	Value	Description
<CODE>	2	0xba,0x00	Response Code
params	126		Network parameters
User_name	17		User name
User_psk	17	Network encryption	Password
<STATUS>	1	0	Command successful
		-2	Command failed, or timeout

5 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

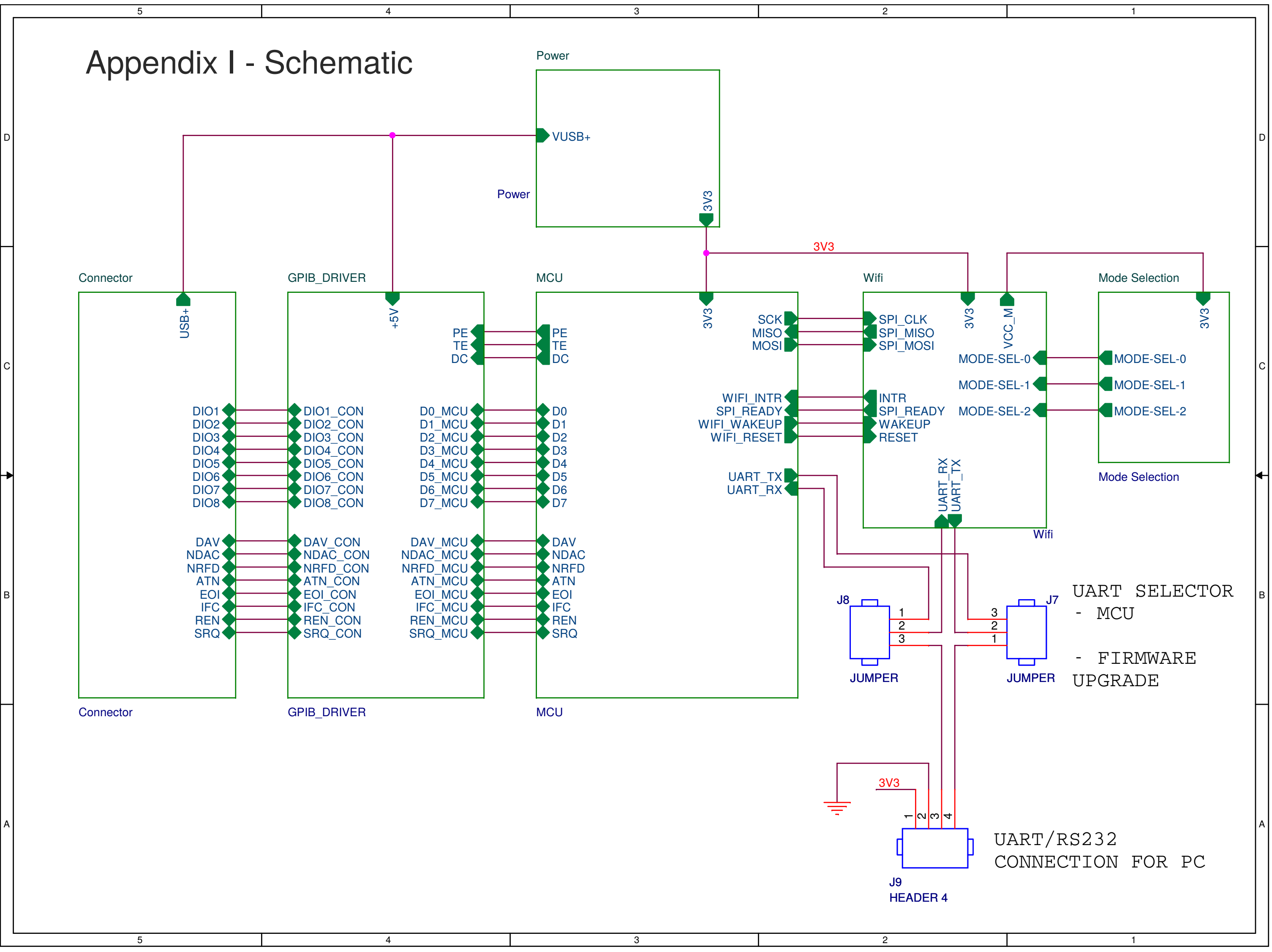
Address: Room 406, Tsinghua information comprehensive building, Nanshan Science Park North District,Shengzhen

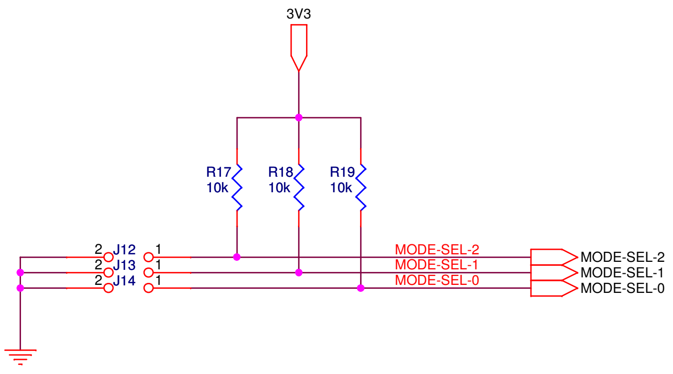
6 Revision History

Version History and modify the contents:

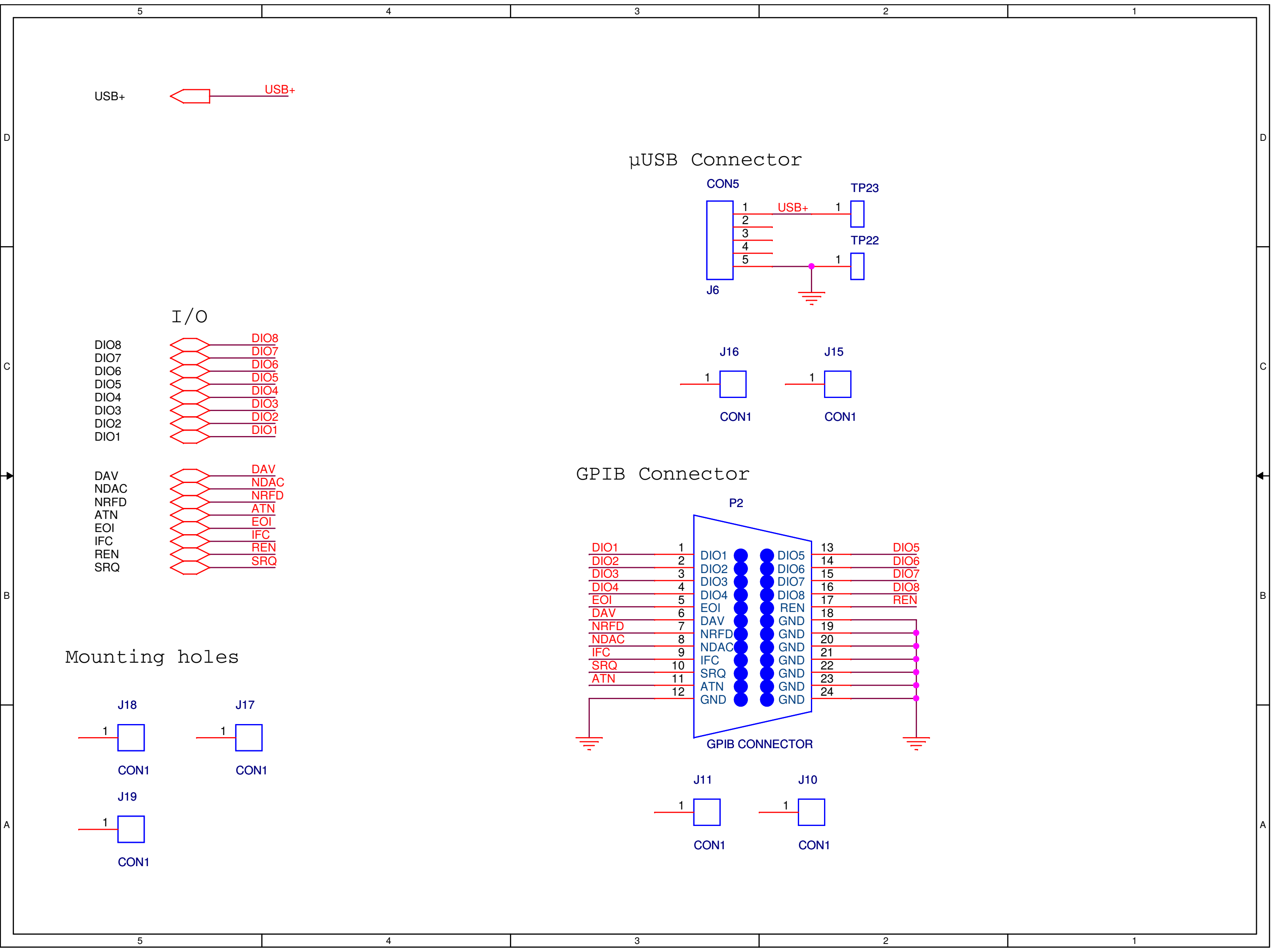
Version	Modifications	Date
V1.0	Initial Draft	2014-03-11
V1.1	Modify draft and release	2014-03-28
V1.2	Modify command returns status, data structure	2014-06-09

Appendix I - Schematic

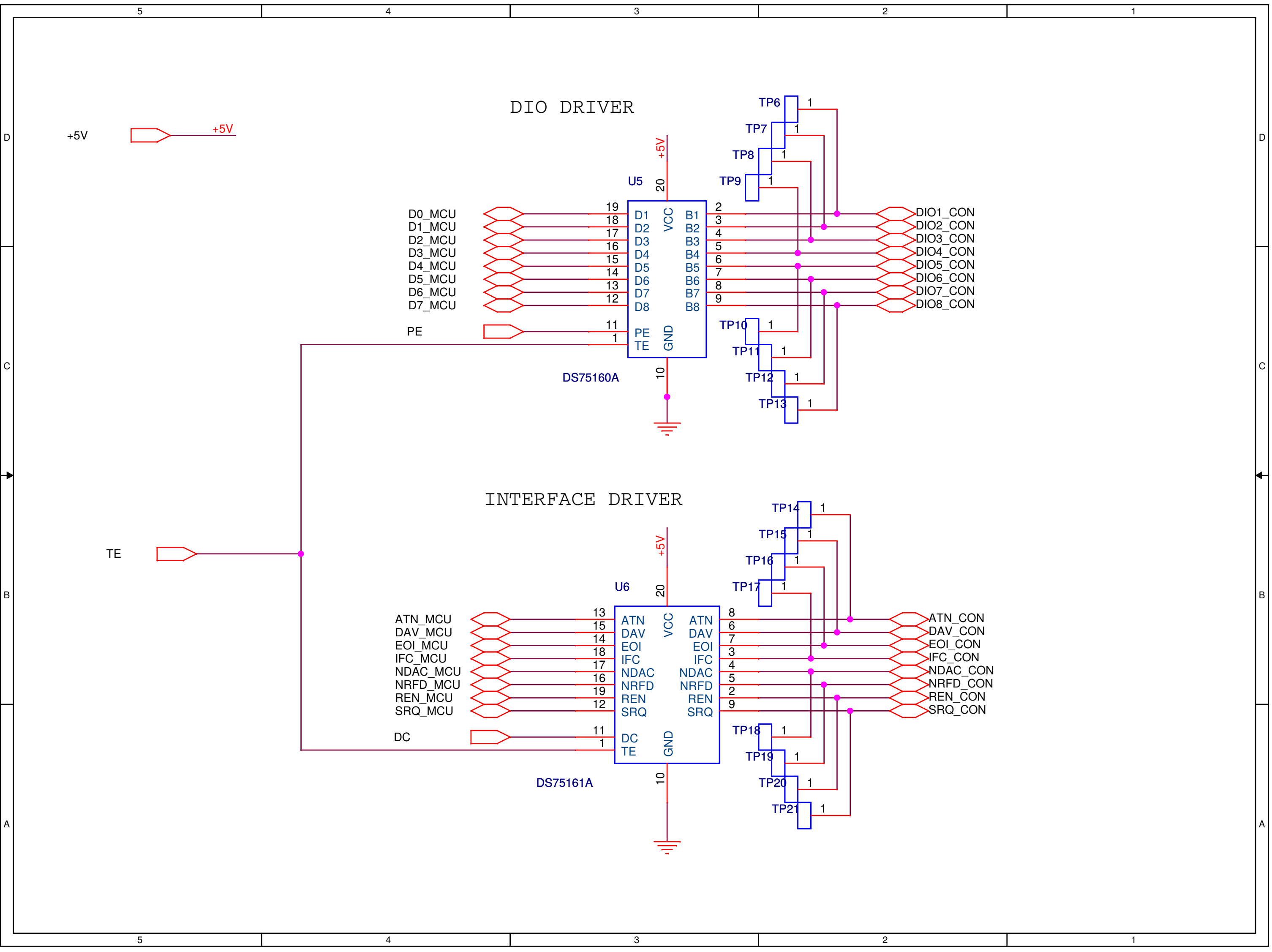




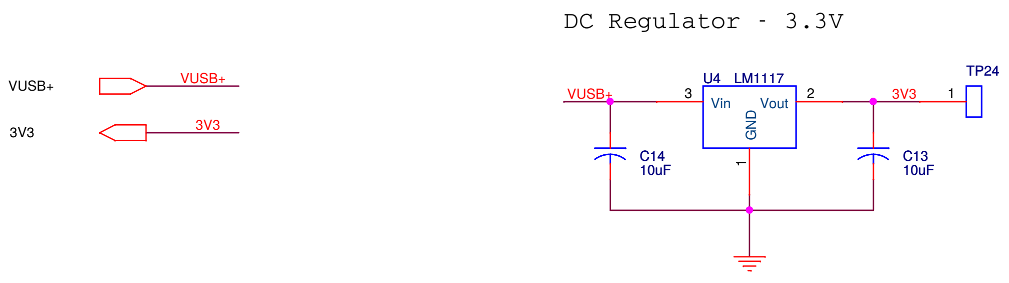
3V3
R17
10k
R18
10k
R19
10k
2
J12
1
2
J13
1
2
J14
1
MODE-SEL-2
MODE-SEL-1
MODE-SEL-0
MODE-SEL-2
MODE-SEL-1
MODE-SEL-0



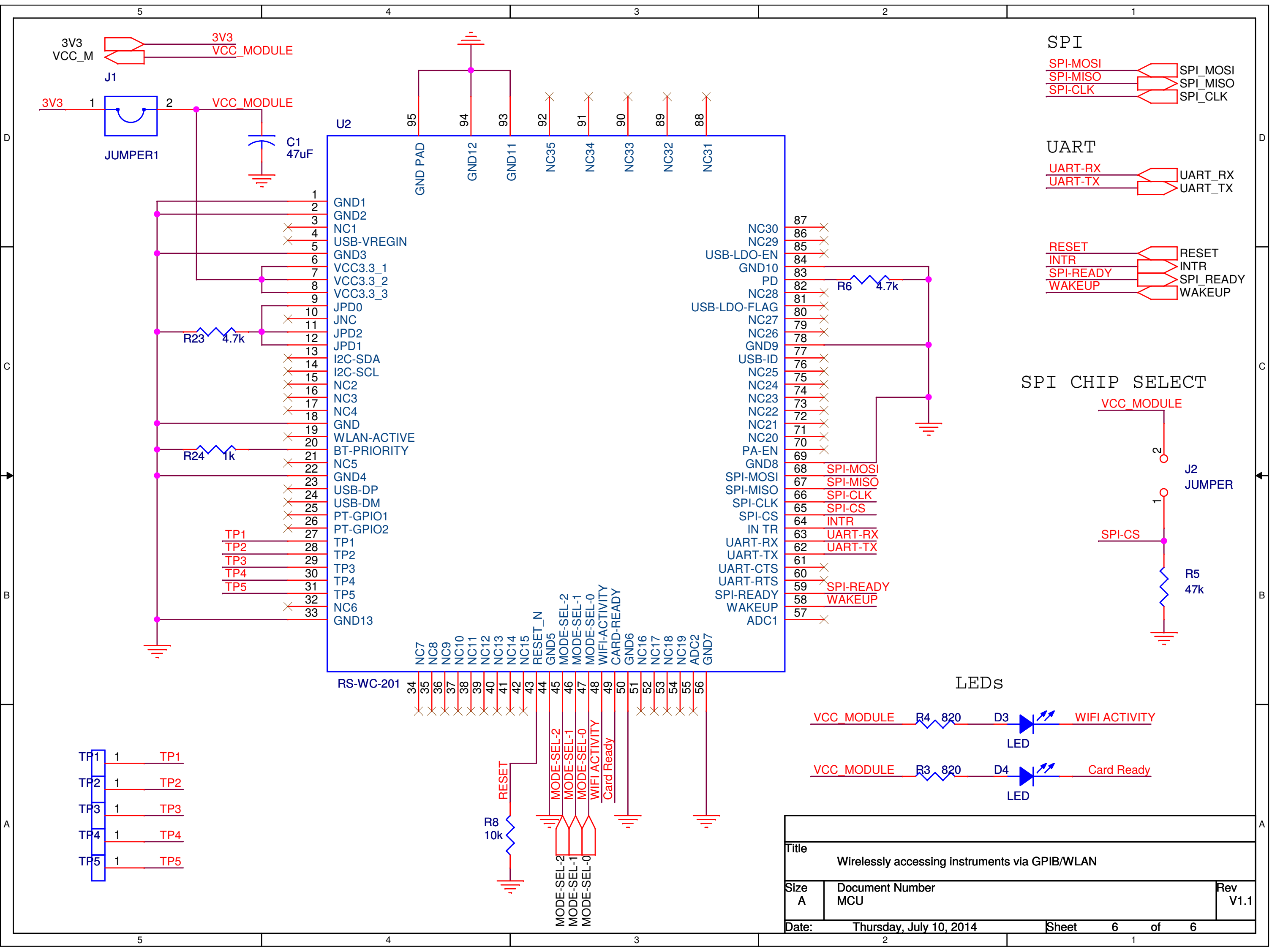
USB+
µUSB Connector
CON5
J6
TP23
TP22
I/O
DIO8
DIO7
DIO6
DIO5
DIO4
DIO3
DIO2
DIO1
DAV
NDAC
NRFD
ATN
EOI
IFC
REN
SRQ
J16
J15
CON1
GPIB Connector
P2
GPIB CONNECTOR
Mounting holes
J18
J17
J19
J11
J10



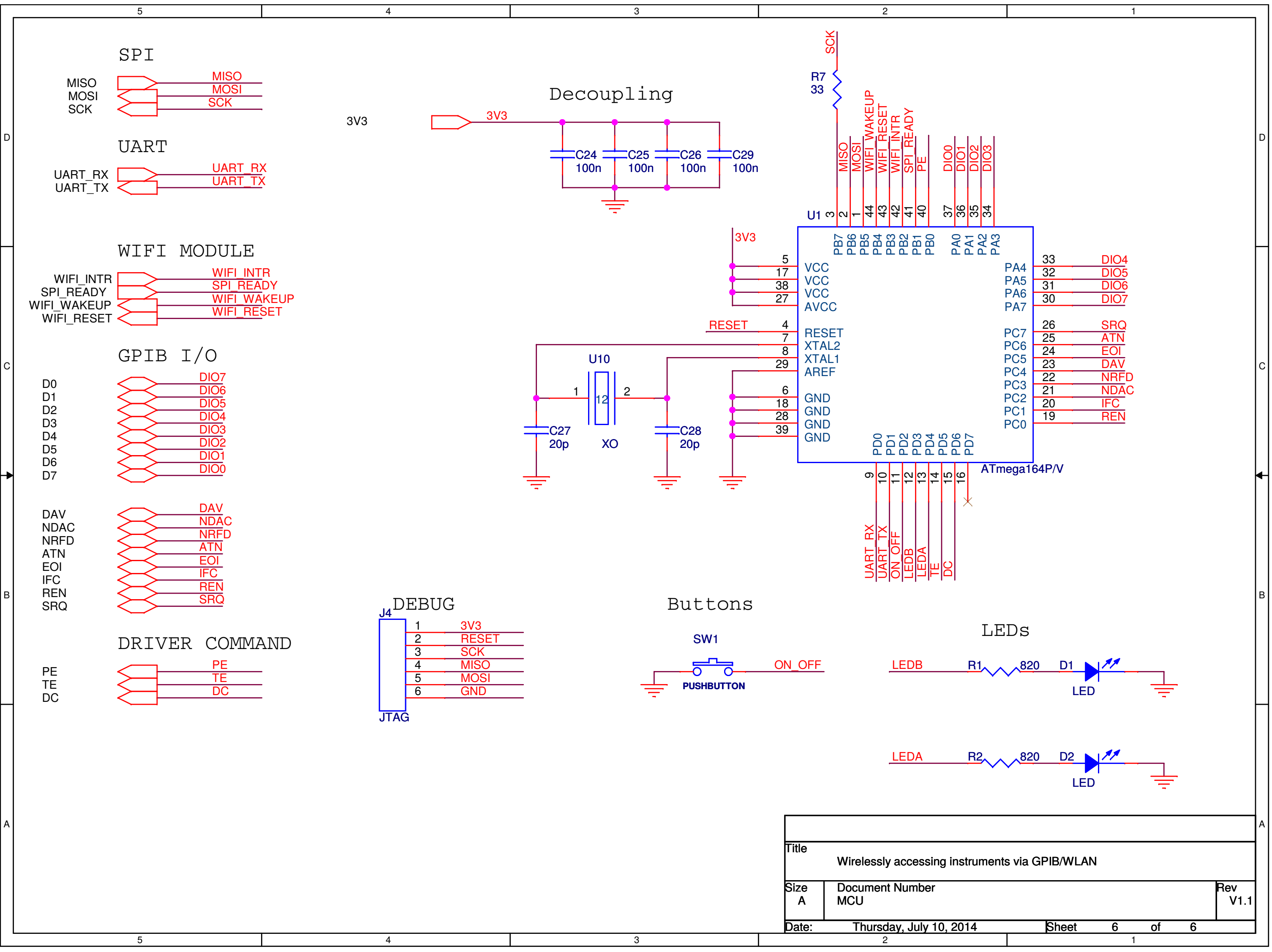
DIO DRIVER
INTERFACE DRIVER
+5V
TE
PE
DC
U5
DS75160A
U6
DS75161A
D0_MCU
D1_MCU
D2_MCU
D3_MCU
D4_MCU
D5_MCU
D6_MCU
D7_MCU
DIO1_CON
DIO2_CON
DIO3_CON
DIO4_CON
DIO5_CON
DIO6_CON
DIO7_CON
DIO8_CON
ATN_MCU
DAV_MCU
EOI_MCU
IFC_MCU
NDAC_MCU
NRFD_MCU
REN_MCU
SRQ_MCU
ATN_CON
DAV_CON
EOI_CON
IFC_CON
NDAC_CON
NRFD_CON
REN_CON
SRQ_CON
TP6
TP7
TP8
TP9
TP10
TP11
TP12
TP13
TP14
TP15
TP16
TP17
TP18
TP19
TP20
TP21



Title	Wirelessly accessing instruments via GPIB/WLAN		
Size A	Document Number Power management		Rev V1.1
Date:	Thursday, July 10, 2014	Sheet 3	of 6

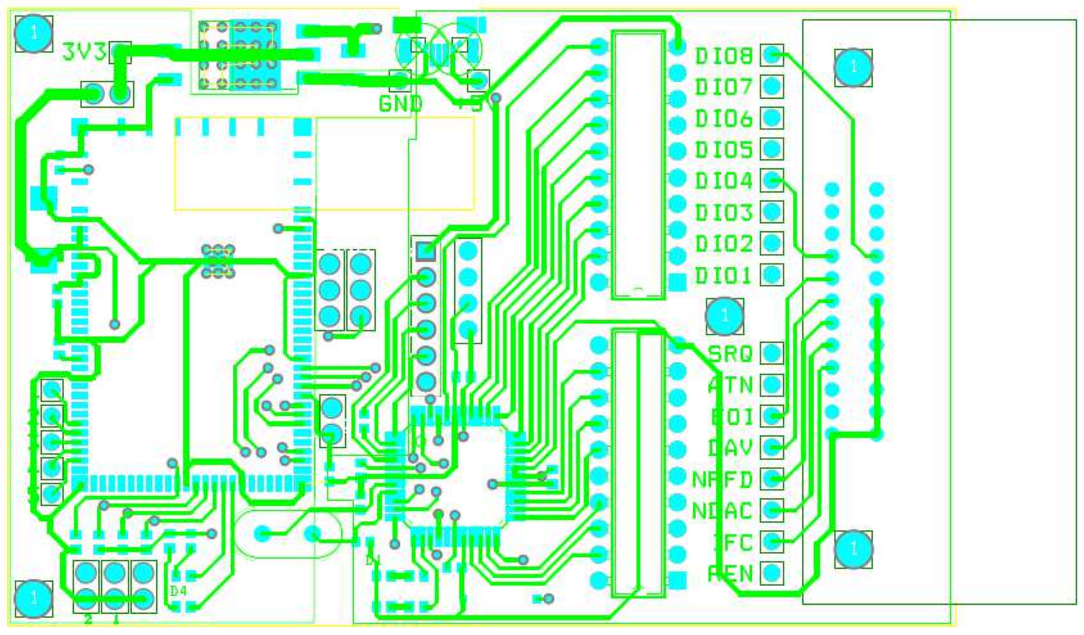


Title	Wirelessly accessing instruments via GPIB/WLAN	
Size A	Document Number MCU	Rev V1.1
Date: Thursday, July 10, 2014	Sheet 6 of 6	

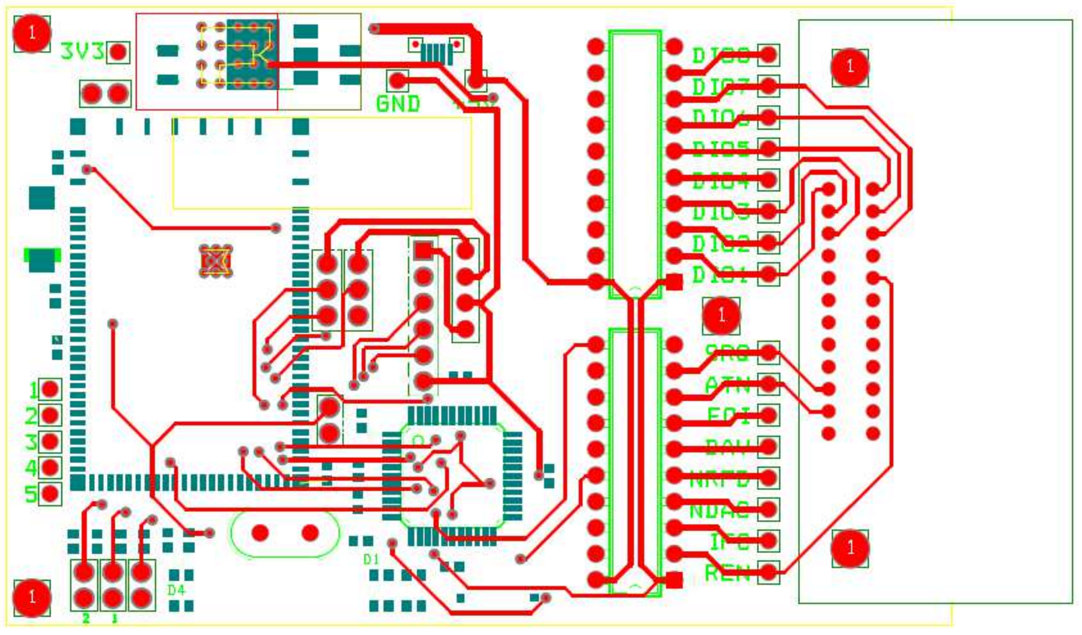

SPI
MISO
MOSI
SCK
UART
UART_RX
UART_TX
WIFI MODULE
WIFI_INTR
SPI_READY
WIFI_WAKEUP
WIFI_RESET
GPIB I/O
D0 DIO7
D1 DIO6
D2 DIO5
D3 DIO4
D4 DIO3
D5 DIO2
D6 DIO1
D7 DIO0
DAV
NDAC
NRFD
ATN
EOI
IFC
REN
SRQ
DRIVER COMMAND
PE
TE
DC
Decoupling
3V3
C24 100n
C25 100n
C26 100n
C29 100n
SCK
R7 33
U1
ATmega164P/V
VCC
VCC
VCC
AVCC
RESET
XTAL2
XTAL1
AREF
GND
GND
GND
GND
PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0
PA0 PA1 PA2 PA3
PA4 PA5 PA6 PA7
PC7 PC6 PC5 PC4 PC3 PC2 PC1 PC0
PD0 PD1 PD2 PD3 PD4 PD5 PD6 PD7
U10
XO
C27 20p
C28 20p
DEBUG
J4
JTAG
1 3V3
2 RESET
3 SCK
4 MISO
5 MOSI
6 GND
Buttons
SW1
PUSHBUTTON
ON_OFF
LEDs
LEDB R1 820 D1 LED
LEDA R2 820 D2 LED
Title
Wirelessly accessing instruments via GPIB/WLAN
Size A
Document Number MCU
Rev V1.1
Date: Thursday, July 10, 2014
Sheet 6 of 6

Appendix J - PCB Design

TOP VIEW



BOTTOM VIEW



Appendix K - MCU Software general

```
/*******************************************************************************/
/* 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
```

```
/*****************************************************************************/
/* FILENAME    :       main.c                                              */
/* AUTHOR      :       Nathan Quinteiro            <nathan.quinteiro@gmail.com> */
/*-------------------------------------------------------------------------------*/
/* FUNCTION    :       Realizing the bidirectional tranceiver between Android    */
/*                             smartphone and GPIB bus.                          */
/*-------------------------------------------------------------------------------*/
/* REVISION    :       1.0 (version for ATmega164PA)
       */
/*****************************************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>
#include "XF/xf.h"
#include "Hardware/io.h"
#include "SM_main.h"
#include "GPIB/GPIB.h"

/*****************************************************************************/
/* FUNCTION     : init()                                                     */
/* INPUT        : -                                                          */
/* OUTPUT       : -                                                          */
/* COMMENTS     : Peripherals initialization                                 */
/*****************************************************************************/
void init(void)
{
       /*-------------------------------------------------------------------------*/
       /*            Status register initialisation                               */
       /*-------------------------------------------------------------------------*/
       SREG   = 0x00;                             // Reset status register

       /*-------------------------------------------------------------------------*/
       /*            Power management initialisation     :      Power down     */
       /*-------------------------------------------------------------------------*/
       SMCR   = 0x04;                             // Select "Power down" mode without
                                                  // enabling to enter in sleep mode

       /*-------------------------------------------------------------------------*/
       /*            Oscillator Calibration for 8MHz                              */
       /*-------------------------------------------------------------------------*/
       OSCCAL = 0x4C;

       /*-------------------------------------------------------------------------*/
       /*            I/O initialisation                                           */
       /*-------------------------------------------------------------------------*/
       DDRA = 0xFF;
       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);
```

```c
	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();
	XF_init();                   //Software initialization
	start_program();             //Start the main states machine
}
```

```c
/*******************************************************************************/
/* FILENAME    :      SM_main.h                                                  */
/* AUTHOR      :      Nathan Quinteiro              <nathan.quinteiro@gmail.com>  */
/*-----------------------------------------------------------------------------*/
/* FUNCTION    :      State machine of the program                               */
/*-----------------------------------------------------------------------------*/
/* REVISION    :      1.0 (version for ATmega164PA)                              */
/*******************************************************************************/


#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_ */
```

```c
/********************************************************************************/
/* FILENAME     :       SM_main.c                                              */
/* AUTHOR       :       Nathan Quinteiro             <nathan.quinteiro@gmail.com>  */
/*------------------------------------------------------------------------------*/
/* FUNCTION     :       Main state machine of the program                      */
/*------------------------------------------------------------------------------*/
/*      The program is divided in three different tasks, communication with RAK    */
/*      Wi-Fi module, communication with GPIB bus and I/O management.          */
/*      Each task is independant and does not block the program                */
/*------------------------------------------------------------------------------*/
/* REVISION     :       1.0 (version for ATmega164PA)                          */
/********************************************************************************/

#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;
}
```

```
/*******************************************************************************/
/*  xf.h                                                                       */
/*  FEMTO-XF                                                                   */
/*                                                                             */
/*  Created by Medard Rieder on 26.08.11.                                      */
/*  Copyright 2011 JFAM. All rights reserved.                                  */
/*                                                                             */
/*      Edited by Nathan Quinteiro on 17.07.14                                 */
/*      To suit to ATmega164PA device                                          */
/*                                                                             */
/*******************************************************************************/

#ifndef XFDEF
#define XFDEF

/*******************************************************************************/
//XF Events
/*******************************************************************************/
//I/O 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

//GPIB Events
#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_asserted66
#define evDIOs_released67
#define evGPIBsendCMDdone     68
#define evGPIBsendDATAdone    69
#define evGPIB_timeout        70
#define evGPIBreadDATAdone    71
//RAK Events
#define evRAKreset                    100
#define evRAKset                      101
#define evSPIWriteEnd         102
#define evRAKinitwaitend      103
```

```
#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
```

```c
/******************************************************************************/
/*  xf.c                                                                      */
/*  FEMTO-XF                                                                  */
/*                                                                            */
/*  Created by Medard Rieder on 26.08.11.                                     */
/*  Copyright 2011 JFAM. All rights reserved.                                 */
/*                                                                            */
/*      Edited by Nathan Quinteiro on 17.07.14                                */
/*      To suit to ATmega164PA device                                         */
/*                                                                            */
/******************************************************************************/

#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;
    }
}
```

```c
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;
            break;
        }
    }
    LEAVECRITICAL(inISR);
    return done;
}

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;
    }

}
```

```c
/*****************************************************************************/
/* 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
}
/*****************************************************************************/
/* FUNCTION     : TIMER1_COMPA_vect                                          */
/* INPUT        : -                                                          */
/* OUTPUT       : -                                                          */
/* COMMENTS     : ISR for Timer 1 - (10ms that gives time base for XF)       */
/*****************************************************************************/
SIGNAL(TIMER1_COMPA_vect)
{
       XF_ISR(IFTMR1);
}
/*****************************************************************************/
/* FUNCTION     : INT0_vect                                                  */
/* INPUT        : -                                                          */
/* OUTPUT       : -                                                          */
/* COMMENTS     : ISR for BT1(Button) both edges                             */
/*****************************************************************************/
SIGNAL(INT0_vect)
{
       if((PIND & (1<<BT1)))
       {
              XF_ISR(IFBUP);
       }
       else
       {
              XF_ISR(IFBDOWN);
       }
}
```

```c
/*****************************************************************************/
/* FUNCTION     : INT2_vect                                                  */
/* INPUT        : -                                                          */
/* OUTPUT       : -                                                          */
/* COMMENTS     : ISR for WIFI INTR rising edge                              */
/*****************************************************************************/
SIGNAL(INT2_vect)
{
        if((PINB & (1<<WIFI_INTR)))
        {
                XF_pushEvent(evWIFIintrHigh, true);
        }
        else
        {
                XF_pushEvent(evWIFIintrLow, true);
        }
}
/*****************************************************************************/
/* FUNCTION     : PCINT0_vect                                                */
/* INPUT        : -                                                          */
/* OUTPUT       : -                                                          */
/* COMMENTS     :       ISR for PORTA pin level change (GPIB DIOs)           */
/*                      Compare the value of PORTA with the previous and push event */
/*                      in the XF when the DIOs are asserted or released     */
/*****************************************************************************/
SIGNAL(PCINT0_vect)
{
        if(oldPORTA == 0xFF)
                XF_pushEvent(evDIOs_asserted, true);
        else if(PINA == 0xFF)
                XF_pushEvent(evDIOs_released, true);

        Interrupt_getPORTA();
}
/*****************************************************************************/
/* FUNCTION     : PCINT2_vect                                                */
/* INPUT        : -                                                          */
/* OUTPUT       : -                                                          */
/* COMMENTS     :       ISR for PORTC pin level change (GPIB interface gesture)     */
/*                      Compare the value of PORTC with the previous and push event */
/*                      in the XF for each change                            */
/*****************************************************************************/
SIGNAL(PCINT2_vect)
{
        if((oldPORTC ^ PINC) == (1<<SRQ))
        {
                if ((PINC & (1<<SRQ)) == 0)
                        XF_pushEvent(evSRQ_asserted, true);
                else
                        XF_pushEvent(evSRQ_released, 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);
}
```

```
/*******************************************************************************/
/* FILENAME    :       io.h                                                     */
/* AUTHOR      :       Nathan Quinteiro              <nathan.quinteiro@gmail.com>  */
/*-----------------------------------------------------------------------------*/
/* FUNCTION    :       Handle the gesture of IO of the MCU                      */
/*-----------------------------------------------------------------------------*/
/* REVISION    :       1.0 (version for ATmega164PA)                            */
/*******************************************************************************/

#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_CSPORTB4

//GPIB pins
#define DIO      PORTA
#define GPIB PORTC
#define REN  PORTC0
#define IFC  PORTC1
#define NDAC PORTC2
#define NRFD  PORTC3
#define DAV  PORTC4
#define EOI  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

#endif /* IO_H_ */
```

```c
/******************************************************************************/
/* 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 "io.h"
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;
}
```

```c
/*******************************************************************************/
/* FILENAME    :       spi.h                                                     */
/* AUTHOR      :       Nathan Quinteiro              <nathan.quinteiro@gmail.com>  */
/*-----------------------------------------------------------------------------*/
/* FUNCTION    :       Handle the gesture of SPI, transmission and reception       */
/*-----------------------------------------------------------------------------*/
/* REVISION    :       1.0 (version for ATmega164PA)                             */
/*******************************************************************************/

#ifndef SPI_H_
#define SPI_H_

#include <avr/io.h>

uint8_t SPI_MasterTransmit(char cData);

#endif /* SPI_H_ */

/*******************************************************************************/
/* FILENAME    :       spi.c                                                     */
/* AUTHOR      :       Nathan Quinteiro              <nathan.quinteiro@gmail.com>  */
/*-----------------------------------------------------------------------------*/
/* FUNCTION    :       Handle the gesture of SPI, transmission and reception       */
/*-----------------------------------------------------------------------------*/
/* REVISION    :       1.0 (version for ATmega164PA)                             */
/*******************************************************************************/
#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;
}
```

Appendix L - MCU Software RAK

```
/*****************************************************************************/
/* FILENAME     :       RAK421.h                                             */
/* AUTHOR       :       Nathan Quinteiro        nathan.quinteiro@gmail.com      */
/*---------------------------------------------------------------------------*/
/* FUNCTION     :       Contains all the commands SPI for the RAK421         */
/*---------------------------------------------------------------------------*/
/* REVISION     :       1.0                                                  */
/*****************************************************************************/


#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
```

```
#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                 0xFFFFFF00      //      subnet mask
#define stdGw                   0xC0A8010A      //      Gateway
#define stdDnssvr1              0xC0A8010A      //      DNS server 1
#define stdDnssvr2              0x00000000      //      DNS server 2

/*IP*/
#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


/*******************************************************************************/
// 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{                                 //It is used to set module power mode.
        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
typedef struct{                                 //It is used to scan wireless networks via command.
        uint32_t        cmd;                    //                      Command code
        uint32_t        channel;                //                      Scanning specified channel(s), scan all channels
if value is 0
        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
typedef struct{                                 //It is used to get the scanned network information.
        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
typedef struct{                                 //It is used to set network password.
        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
typedef struct{                                 //It is used to set network channel.
        uint32_t        cmd;                    //                      Command code
        uint32_t        channel;                //                      1 to 13, SettingAP / Ad-Hoc channel
} rak_channel_t;                                //Return value :        CODE - 2 bytes 0xAB00
                                                //                      STATUS - 1 byte (0 successful, -2 failed)

#define         rak_connect 0xA6970000
typedef struct{                                 //It  is  used  to  connect  an  AP/ ADHOC  network  or  establish  a
specified  AP Network.
        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{                                 //This command is used to assign static IP address for module.
        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{                                 //This command is used to set DHCP working mode.
        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_storeconfigcommand.
                                                //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 0xAE00
                                              //                     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
typedef struct{                               //It is used to convert domain  to the corresponding IP address, the
domain  must
                                              //be configured available DNS server .
        uint32_t        cmd;                  //                     Command code
        uint32_t        name[42];             //                     Select DHCP mode : Domain
} rak_dns_t;                                  //Return value :       CODE - 2 bytes 0xAF00
                                              //                     ADDR - 4 bytes, IP address
                                              //                     STATUS - 1 byte (0 successful, -2  DNS
resolution failed)
#define         rak_ping  0xB0970000
typedef struct{                               //It is used torun the ping command.
        uint32_t        cmd;                  //                     Command code
        uint32_t        hostaddr;             //                     Specified host
        uint32_t        count;                //                     number of packet
        uint32_t        size;                 //                     size of packet, 1 to 1400
} rak_ping_t;                                 //Return value :       CODE - 2 bytes 0xB000
                                              //                     STATUS - 1 byte (0 successful, -2  Cannot access
host)

#define         rak_apconfig  0xAA970000
typedef struct{                               //It is used to set up parameters for  a wireless access point, such as
the country
                                              //code, whether network name is hidden or not.
        uint32_t        cmd;                  //                     Command code
        uint32_t        hidden;               //                     0 - Network name is visible, 1 - Network name is
hidden
        uint16_t        countryCode;          //                     Country code, e.g. CN
} rak_apconfig_t;                             //Return value :       CODE - 2 bytes 0xAA00
                                              //                     STATUS - 1 byte (0 successful, -2  failed)


#define         rak_set_listen  0xA8970000
typedef struct{                               //It is used to set modulebeacon 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
typedef struct{                                 //Module is used as a TCP server and creates  a listening port, if the
operation is
                                                //successful, the module will return a hexadecimal identifier that is
used to manage
                                                //the connection. This command can create up to four connections.
        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

typedef struct{                                 //This  command  is  to  create  a  TCP  CLIENT  and  connect  with  the
remote  TCP
                        //SERVER,  if  the  operation  is  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 .
        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
                                                //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)
```

```
#define         rak_udp_server 0xB2970000       //This command  is used to create a port in local and  wait for
dataremote 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 connectionsare  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)

#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)




#define         rak_socket_close 0xB5970000
typedef struct{                                 //It is used to close the already opened socket identifier .
        uint32_t        cmd;                    //                      Command code
        uint16_t        dummy;                  //                      Invalid data
        uint16_t        flag;                   //                      0-7 Socket identifier
}rak_close_t;                                   //Return value :        CODE - 2 bytes 0xB100
                                                //                      STATUS - 1 byte (0 successful, -2  specified
port does not exist, -3 failed to close)

#define         rak_send_data 0xB6970000
typedef struct{                                 //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
                //datawithout  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  valuecan  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.
        uint32_t        cmd;                    //                      Command code
        uint32_t        dest_addr;              //                      Target IP address
        uint32_t        dest_port;              //                      Target port, 1 to 65535
        uint16_t        socket_flag;            //                      0-7 Socket identifier
        uint16_t        len;                    //                      1 to 1400, data length
        char            buffer[RAK_MAX_DATA_SIZE];//                    data
} rak_send_t;
```

```
#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
/*********************************************************************************/

#define         rak_storeconfig_data 0xC0970000
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;                                      //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
typedef struct{                                 //It is used to save thenetwork 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;                                         //Return value :        CODE - 2 bytes 0xC100
                                                //                      STATUS - 1 byte (0 successful, -2  failed)

#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
WEBwith
                                                //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 :        CODE - 2 bytes 0xC700
                                                //                      STATUS - 1 byte (0 successful, -2  failed)


#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;
        rak_ap_param                                ap_param;
}param_t;

/* param struct*/
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;
        uint8_t                                         ssid[RAK_SSID_LEN];
        uint8_t                                         bssid[RAK_BSSID_LEN];
} rak_scanInfo;

//getscan rsp struct
typedef struct {
        uint8_t                         rspCode[RAK_RSPCODE_LEN];
        rak_scanInfo                strScanInfo[RAK_AP_SCANNED_MAX];
        uint8_t                     status;
}rak_getscanResponse;


/*getrssi rsp struct*/
typedef struct {
        uint8_t                         rspCode[RAK_RSPCODE_LEN];
        uint16_t                    rssi;
        uint8_t                     status;
}rak_getrssiResponse;

/*socketEst rsp struct*/
typedef struct {
        uint8_t                         rspCode[RAK_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[RAK_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[RAK_RSPCODE_LEN];
        uint8_t                                 macAddr[6];
        uint8_t                                 ipaddr[4];
        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[RAK_RSPCODE_LEN];
        uint16_t                    socket_flag;
        uint8_t                                     status;
} rak_socketFrameRcv;

/*init rsp struct*/
typedef struct {
        uint8_t                         rspCode[RAK_RSPCODE_LEN];
        uint8_t                     strdata[RAK_WELCOME_LEN];
        uint8_t                     status;
```

```
} rak_initResponse;

/*recv rsp struct*/
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;

/*mgmt rsp struct*/
typedef struct {
        uint8_t                                 rspCode[RAK_RSPCODE_LEN];
        uint8_t                                 status;
} rak_mgmtResponse;
/*dns rsp struct*/
typedef struct {
        uint8_t                                 rspCode[RAK_RSPCODE_LEN];
        uint8_t                                                 addr[4];
        uint8_t                                 status;
} rak_dnsResponse;
/*easy and wps rsp struct*/
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;

/*param rsp struct*/
typedef struct {
        uint8_t                                 rspCode[RAK_RSPCODE_LEN];
        param_t                                                 rak_param;
        uint8_t                                 status;
}rak_paramResponse;

/*web rsp struct*/
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;

/*cmd rsp struct*/
typedef union {
        uint8_t                         rspCode[RAK_RSPCODE_LEN];                               // command code response
        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_mgmtResponse                        mgmtResponse;
        rak_paramResponse                               paramFrame;
        rak_webResponse                                 webFrame;
        uint8_t                                         CmdRspBuf[RAK_MAX_DATA_SIZE+52];
} rak_CmdRsp;

/*********************************************************************************/
// Functions used to configure structures to be sent on RAK421 module
//-------------------------------------------------------------------------------
// Only integer values are put in the structures, the arrays must be filled
// manually in the structures.
/*********************************************************************************/
void set_pwr_mode_cmd(rak_pwr_mode_t * pwrmode, uint32_t cmd, uint32_t powermode);
void set_scan_cmd(rak_scan_t * scan, uint32_t cmd, uint32_t channel);
void set_getscan_cmd(rak_getscan_t * getscan, uint32_t cmd, uint32_t scan_num);
void set_psk_cmd(rak_psk_t * psk_cmd, uint32_t cmd);
void set_channel_cmd(rak_channel_t * channel_cmd, uint32_t cmd, uint32_t channel);
void set_conn_cmd(rak_conn_t * conn, uint32_t cmd, uint32_t mode);
void set_ipstatic_cmd(rak_ipstatic_t * ipstatic, uint32_t cmd, uint32_t addr, uint32_t mask, uint32_t gw, uint32_t
dnssrv1, uint32_t dnssrv2);
void set_ipdhcp_cmd(rak_ipdhcp_t * ipdhcp, uint32_t cmd, uint32_t mode);
void set_dns_cmd(rak_dns_t * dns, uint32_t cmd);
void set_ping_cmd(rak_ping_t * ping, uint32_t cmd, uint32_t hostaddr, uint32_t count, uint32_t size);
void set_apconfig_cmd(rak_apconfig_t * apconfig, uint32_t cmd, uint8_t hidden, uint16_t countrycode);
void set_beacon_cmd(rak_beacon_t * beacon, uint32_t cmd, uint32_t time);
void set_server_cmd(rak_server_t * server, uint32_t cmd, uint16_t dummy, uint16_t port);
void set_client_cmd(rak_client_t * client, uint32_t cmd, uint32_t dest_addr, uint16_t dest_port, uint16_t local_port);
void set_close_cmd(rak_close_t * close, uint32_t cmd, uint16_t dummy, uint16_t 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);
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);
void set_web_cmd(web_t * web, uint32_t cmd);


#endif /* RAK_H_ */
```

```
/********************************************************************************/
/* FILENAME      :       RAK.c                                                  */
/* AUTHOR        :       Nathan Quinteiro        <nathan.quinteiro@gmail.com>   */
/*------------------------------------------------------------------------------*/
/* FUNCTION      :       Contains all the functions SPI for the RAK421          */
/*------------------------------------------------------------------------------*/
/* REVISION      :       1.0                                                    */
/********************************************************************************/
#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
 */
```

```c
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);
    }
}
/*******************************************************************************/
/*      Functions to fill the commands structure
        */
/*******************************************************************************/
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);
}
```

```c
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
dnssrv1, uint32_t dnssrv2)
{
ipstatic->cmd = (cmd);
ipstatic->addr = (addr);
ipstatic->mask = (mask);
ipstatic->gw = (gw);
ipstatic->dnssvr1 = (dnssrv1);
ipstatic->dnssvr2 = (dnssrv2);
}

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);
}
```

```c
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 commmunication 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_

void SmartphoneHandling();


#endif /* SM_RAK_H_ */
/*******************************************************************************/
/* FILENAME     :       SM_RAK.c                                               */
/* AUTHOR       :       Nathan Quinteiro        <nathan.quinteiro@gmail.com>    */
/*-----------------------------------------------------------------------------*/
/* FUNCTION     :       State machine handling commmunication with the RAK411 module    */
/*-----------------------------------------------------------------------------*/
/* REVISION     :       1.0                                                    */
/*******************************************************************************/
#include "SM_RAK.h"

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;
/*******************************************************************************/
//      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);
                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;
                                        }
                                //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;
                                        }
                                        //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;
                                        }
                                        //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;
                                                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;

                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 tranmission 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);
                                RAK_state = stRAKsend;
                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;
        }
}
```

```c
/*******************************************************************************/
/* FILENAME     :       SM_RAK_init.h                                          */
/* AUTHOR       :       Nathan Quinteiro        <nathan.quinteiro@gmail.com>    */
/*-----------------------------------------------------------------------------*/
/* FUNCTION     :       State machine of the initialization of RAK module       */
/*-----------------------------------------------------------------------------*/
/* REVISION     :       1.0                                                    */
/*******************************************************************************/
#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>

#ifndef SM_RAK_INIT_H_
#define SM_RAK_INIT_H_

BOOL RAK_sys_init(Event ev, rak_CmdRsp * rsp);

void initGetVersion();
void initUScan();
void initGetScan();
void initSetPsk();
void initConnect();
void initGetNetStatus();
void initSetIPStatic();
void initIPQuery();
void initTCPServer();

#endif /* SM_RAK_INIT_H_ */
/*******************************************************************************/
/* FILENAME     :       SM_RAK_init.c                                          */
/* AUTHOR       :       Nathan Quinteiro        <nathan.quinteiro@gmail.com>    */
/*-----------------------------------------------------------------------------*/
/* FUNCTION     :       State machine of the initialization of RAK              */
/*-----------------------------------------------------------------------------*/
/* REVISION     :       1.0                                                    */
/*******************************************************************************/
#include "SM_RAK_init.h"

State RAK_init_state;

rak_common              cmd;
extern rak_api          RAK_command;
/*******************************************************************************/
//      RAK initialization states machine
/*******************************************************************************/
/**
 * @fn                  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
 * @param[in]   Event ev, last event that happen. rak_CmdRsp * rsp, pointer on the response structure
 * @param[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 stRAKsendSet_ip_static:    if(RAK_sendCMD(ev))
                                        {
                                                if(ev == evWIFIintrHigh || PINB & (1<<WIFI_INTR))
                                                {
                                                SM_RAK_initReceive(rsp);
                                                RAK_init_state = stRAKreceiveSet_ip_static;
                                                }
                                        }
                                        return false;
        break;
        case stRAKreceiveSet_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 initUScan()
{
        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.uIpstaticFrame, 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;
}


/********************************************************************************/
/* FILENAME     :       SM_RAK_receiveCMD.h                                     */
/* AUTHOR       :       Nathan Quinteiro        <nathan.quinteiro@gmail.com>    */
/*------------------------------------------------------------------------------*/
/* FUNCTION     :       State machine used to receive response from RAK module  */
/*------------------------------------------------------------------------------*/
/* REVISION     :       1.0                                                     */
/********************************************************************************/
#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_ */
/********************************************************************************/
/* FILENAME     :       SM_RAK_receiveCMD.c                                     */
/* AUTHOR       :       Nathan Quinteiro        <nathan.quinteiro@gmail.com>    */
/*------------------------------------------------------------------------------*/
/* FUNCTION     :       State machine used to receive response from RAK module  */
/*------------------------------------------------------------------------------*/
/* REVISION     :       1.0                                                     */
/********************************************************************************/
#include "SM_RAK_receiveCMD.h"

State RAK_receive_state;
rak_CmdRsp * rak_rsp;
int tryreceive = 0;

/********************************************************************************/
//      RAK receive states machine
/********************************************************************************/
/**
 * @fn                  BOOL RAK_receiveCMD(Event ev)
 * @brief               Send data read command to module. Once the module acknowledge, read the response
 * @param[in]   Event ev, last event that happen
 * @param[out]  rak_rsp, pointer on the response structure filled by this states machine
 * @return              BOOL finished,  true if the response has finished successfuly be read
 */
BOOL RAK_receiveCMD(Event ev)
{
        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;
                                                tryreceive = 0;
                                                return false;
                                        }
                                        //If module does not aknowledge, 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;
}
```

```
/*********************************************************************************/
/* FILENAME     :       SM_RAK_sendCMD.h                                         */
/* AUTHOR       :       Nathan Quinteiro        <nathan.quinteiro@gmail.com     */
/*-------------------------------------------------------------------------------*/
/* FUNCTION     :       State machine used to send a command to RAK module      */
/*-------------------------------------------------------------------------------*/
/* REVISION     :       1.0                                                      */
/*********************************************************************************/
#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_ */

/*********************************************************************************/
/* FILENAME     :       SM_RAK_sendCMD.c                                         */
/* AUTHOR       :       Nathan Quinteiro        <nathan.quinteiro@gmail.com>    */
/*-------------------------------------------------------------------------------*/
/* FUNCTION     :       State machine used to send a command to RAK module      */
/*-------------------------------------------------------------------------------*/
/* REVISION     :       1.0                                                      */
/*********************************************************************************/
#include "SM_RAK_sendCMD.h"

State RAK_send_state;
int trysend = 0;
int cmdLength;
uint8_t command;
char * data;

/*********************************************************************************/
//      RAK send states machine
/*********************************************************************************/
/**
 * @fn                  BOOL RAK_sendCMD(Event ev)
 * @brief               The desired command is sent to the module until the module acknowledge it. Once the module
acknowledge, the parameters are sent
 * @param[in]   Event ev, last event that happen
 * @param[out]  none
 * @return              BOOL finished,  true if the command has finished successfuly be sent
 */
BOOL RAK_sendCMD(Event ev)
{
        switch(RAK_send_state)
        {
                case stSendHead:        //If module acknowledge command
                                        if(RAK_sendHeader(command, cmdLength) == SPI_CMD_ACK)
                                        {
                                                //Send the parameters
                                                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;
}
```

```c
/********************************************************************************/
/* FILENAME     :       Smartphone.h                                            */
/* AUTHOR       :       Nathan Quinteiro        <nathan.quinteiro@gmail.com>    */
/*------------------------------------------------------------------------------*/
/* FUNCTION     :       Define the structure of packet for communication with smartphone        */
/*------------------------------------------------------------------------------*/
/* REVISION     :       1.0                                                     */
/********************************************************************************/


#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_ */
```

Appendix M - MCU Sofware GPIB

```
/*******************************************************************************/
/* 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_ */
```

```
/********************************************************************************/
/* FILENAME     :       GPIB.c                                                  */
/* AUTHOR       :       Nathan Quinteiro        <nathan.quinteiro@gmail.com>    */
/*------------------------------------------------------------------------------*/
/* FUNCTION     :       Handle the communication with the GPIB bus              */
/*------------------------------------------------------------------------------*/
/* REVISION     :       1.0                                                     */
/********************************************************************************/

#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;
}
```

```
/*********************************************************************************/
/* FILENAME     :       SM_GPIB.h
                        */
/* AUTHOR       :       Nathan Quinteiro                <nathan.quinteiro@gmail.com>                    */
/*------------------------------------------------------------------------------*/
/* FUNCTION     :       State machine handling commmunication with the GPIB bus                 */
/*------------------------------------------------------------------------------*/
/* REVISION     :       1.0
                                */
/*********************************************************************************/
#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"

#ifndef 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_ */
```

```
/*********************************************************************************/
/* FILENAME     :       SM_GPIB.c                                                */
/* AUTHOR       :       Nathan Quinteiro        <nathan.quinteiro@gmail.com>     */
/*-------------------------------------------------------------------------------*/
/* FUNCTION     :       State machine of the GPIB management                     */
/*-------------------------------------------------------------------------------*/
/* REVISION     :       1.0                                                      */
/*********************************************************************************/
#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 stGPIBreceiveDATA: if(GPIB_ReceiveByte(ev))
                                        {
                                                i++;
                                                //End of reception?
                                                if(i == numberOfByteToReceive)
                                                {
                                                        GPIB_state = stGPIBnone;
                                                        XF_pushEvent(evGPIBreadDATAdone, 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 = stGPIBreceiveDATA;
        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;
}

/*******************************************************************************/
/* FILENAME     :       SM_GPIB_send.h
                */
/* AUTHOR       :       Nathan Quinteiro                <nathan.quinteiro@gmail.com>                    */
/*-----------------------------------------------------------------------------*/
/* FUNCTION     :       State machine of the GPIB byte reception                                        */
/*-----------------------------------------------------------------------------*/
/* REVISION     :       1.0
                                */
/*******************************************************************************/


#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);
//GPIB receive byte states function
void anrs(Event ev);
void acrs(Event ev);
void acds(Event ev);
void awns(Event ev);

#endif /* GPIB_RECEIVE_SM_H_ */


/*******************************************************************************/
/* FILENAME     :       SM_GPIB_send.c
                */
/* AUTHOR       :       Nathan Quinteiro                <nathan.quinteiro@gmail.com>                    */
/*-----------------------------------------------------------------------------*/
/* FUNCTION     :       State machine of the GPIB byte reception                                        */
/*-----------------------------------------------------------------------------*/
/* REVISION     :       1.0
                                */
/*******************************************************************************/

#include "SM_GPIB_receive.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;
                break;
        }
        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>    */
/*------------------------------------------------------------------------------*/
/* FUNCTION     :       State machine of the GPIB byte transmission             */
/*------------------------------------------------------------------------------*/
/* 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>    */
/*------------------------------------------------------------------------------*/
/* FUNCTION     :       State machine of the GPIB byte transmission             */
/*------------------------------------------------------------------------------*/
/* 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;
}
```

```
//Source generate state
void sgns(Event ev)
{
        if(ev == evNRFD_released)
        {
                PORTC &= (0xFF^(1<<DAV));       //assert DAV
                GPIB_send_state = SDYS;
                LED_program(200, 1);
        }
}

//Source delay state
void sdys(Event ev)
{
        if(ev == evNRFD_asserted)
        {
                GPIB_send_state = STRS;
                LED_program(200, 2);
        }
}

//Source transfer state
void strs(Event ev)
{
        if(ev == evNDAC_released)
        {
                PORTC |= (1<<DAV);              //release DAV
                GPIB_send_state = SWNS;
                LED_program(200, 3);
        }
}

//Source waiting for new cycle state
void swns(Event ev)
{
        if(ev == evNDAC_asserted)
        {
                GPIB_send_state = GPIB_send_done;
                DIO = 0xFF;
                LED_program(200, 4);
        }
}

//Init
void SM_init_GPIB_send(char byte)
{
        GPIB_send_state = SGNS;
        DIO = byte;
}
```

Appendix N - Android java code

MainActivity.java

```
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);
        bb.setOnClickListener(bledlistener);
        bsenddata.setOnClickListener(bsendlistener);
        bsendcmd.setOnClickListener(bsendlistener);
        breceivedata.setOnClickListener(bsendlistener);
        bstart.setOnClickListener(bstartlistener);

        spCMD = (Spinner) findViewById(R.id.spcommand);
        spREC = (Spinner) 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
                    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);
            }
            return data;
        }
    };

        //
        private OnClickListener bledlistener = new OnClickListener() {
            @Override
            public void onClick(View v) {
                if(v == ba)
                {
                    if(a == 1)
                        a = 0;
                    else
                        a = 1;
                }
                if(v == bb)
                {
                    if(b == 1)
                        b = 0;
                    else
                        b = 1;
                }
                //send LED command to wifi module
                sendLedCmd();
            }

        };

    //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);
    }


    //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
                            {
                                txt+= "OFF";
                                ima.setImageResource
(android.R.drawable.presence_offline);
                            }
                            txt+=", LEDB : ";
                            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);
                break;
            //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!");
                            }

                break;
            //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!");
                            }
                break;
            }
        }
        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();
            }
        });
    }
```

```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
                            {
```

```
                            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");
                Log.i("AsyncTask", "doInBackground: Waiting for inital data...");
                //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();
            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]);
        }
```

```java
    }
    @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.");
        }
    }
}
```

```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();
        }
    };
    //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 == 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();
                    }

                    //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).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
();
    }

}
```

```java
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 bsavecmd;
    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);
        bsavecmd = (Button) findViewById(R.id.bsavecmd);
        bsavecmd.setOnClickListener(bsavecmdlistener);

        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() {
        @Override
        public void onClick(View v)
        {
            //Save the library in the chosen directory
            if(path!=null && !etLibName.getText().toString().equals(""))
            {
                try {
                    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!");
                } catch (Exception e) {
                    showToast("Error while saving library!");
                }
            }
            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
                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);
            }
        }
    }
}
```

```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);
        }
```

```
    @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 =
           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();
        }
    };


}
```

```java
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");
    }
}
```