Sierra Radio Systems

CommServer

Embedded Linux Platform

Reference Manual

Version 1.5

CommServer Introduction

The Sierra Radio Systems CommServer is an embedded linux host computer platform that can be used for many applications. Sierra Radio will develop applications on this platform and we encourage our users and 3rd parties to build on top of the platform as well. Where possible, all of our software applications will be open source allowing maximum creativity and range of development options.

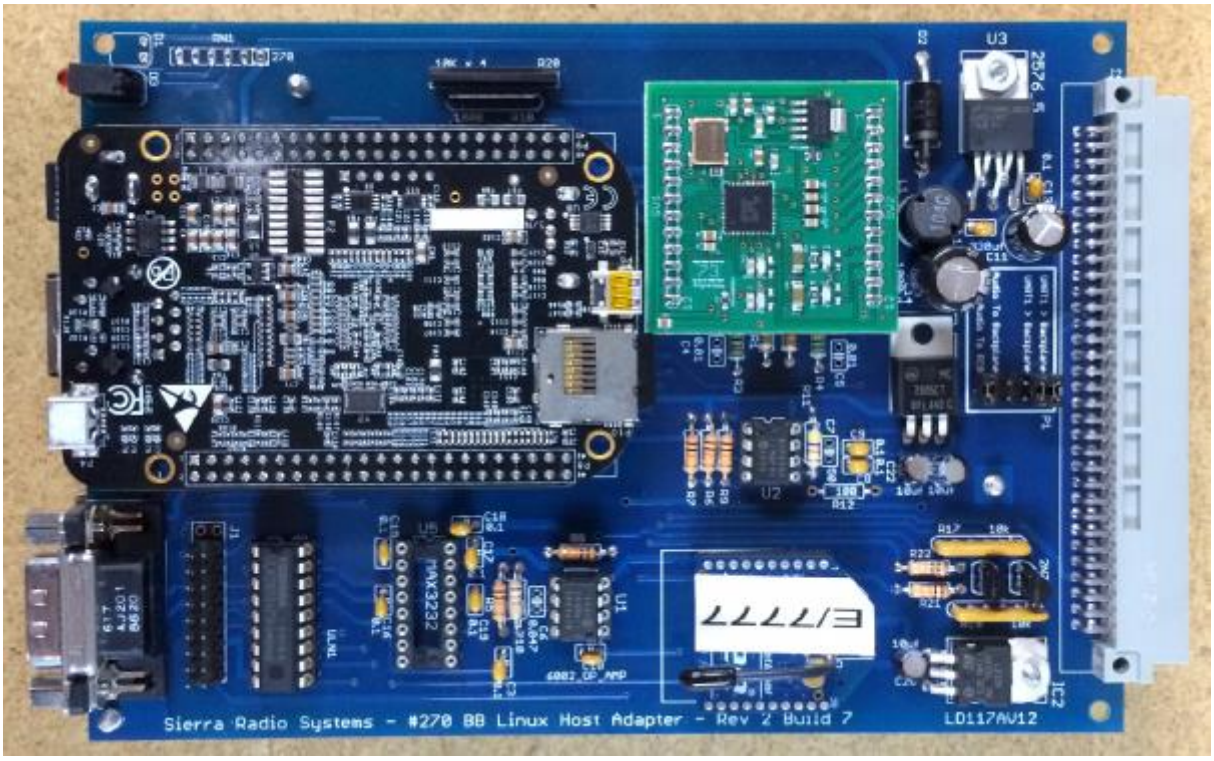
Lets start with the hardware. The key features are…

- BeagleBone Black single board linux computer
  - 1 GHz ARM Cortex A8 CPU
  - 512 MB RAM
  - 2 GB eMMC flash memory
  - MicroSD card slot
  - Ethernet
  - UARTs, I2C, SPI, A/D, etc.
- Power supplies to deliver 5.0v digital, 3.3v digital and 5.0v analog voltage rails
- Optional audio CODEC providing analog audio input and output
- Xbee data radio socket to allow communications with external devices
- RS232 level shifter and DB9 to bring out the linux board's UART 2 port

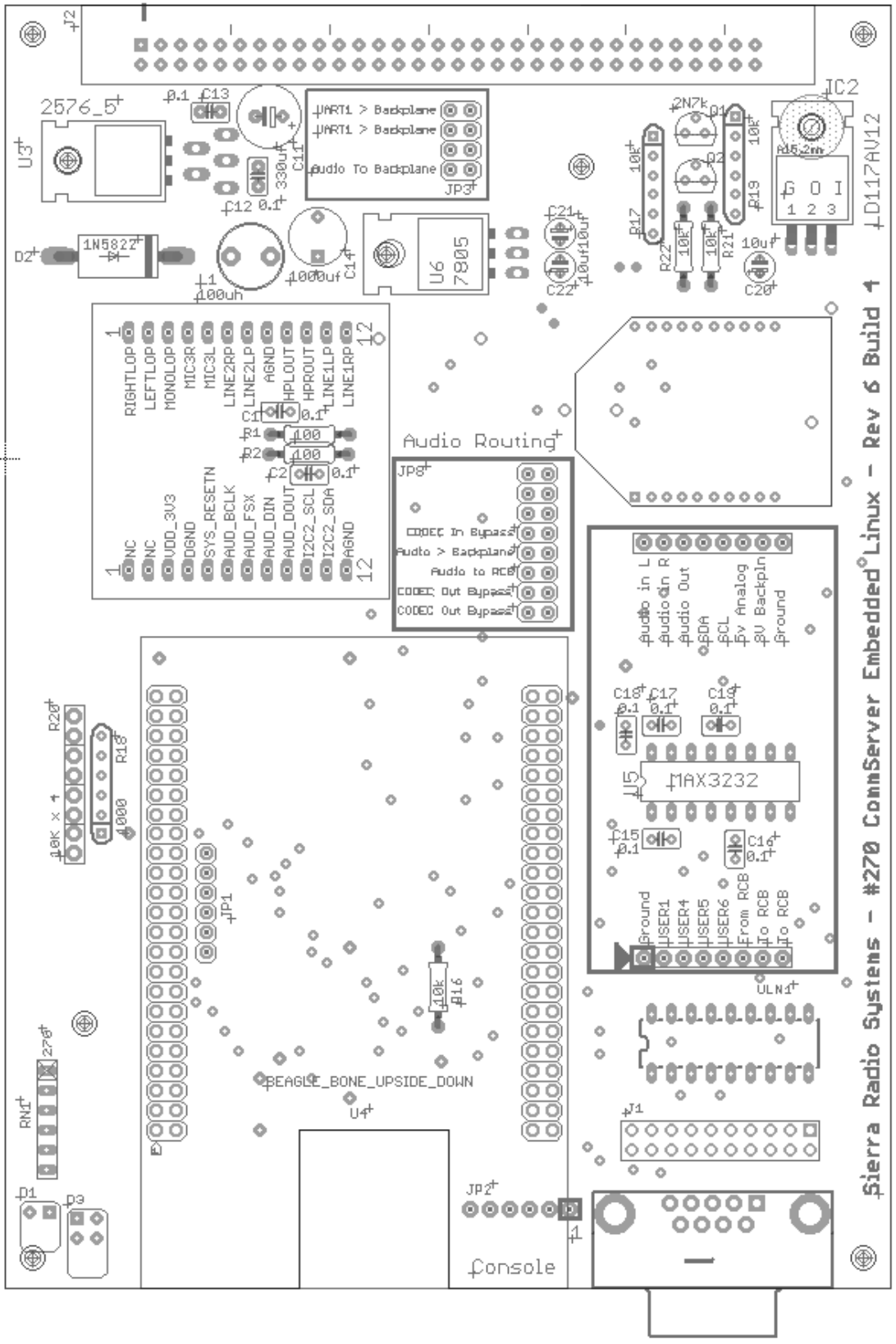
Key software features

- Ubuntu linux distribution
- Built in SRS apps including voice telemetry, scheduled events and command decoding from the main repeater controller CPU

CommServer Introduction

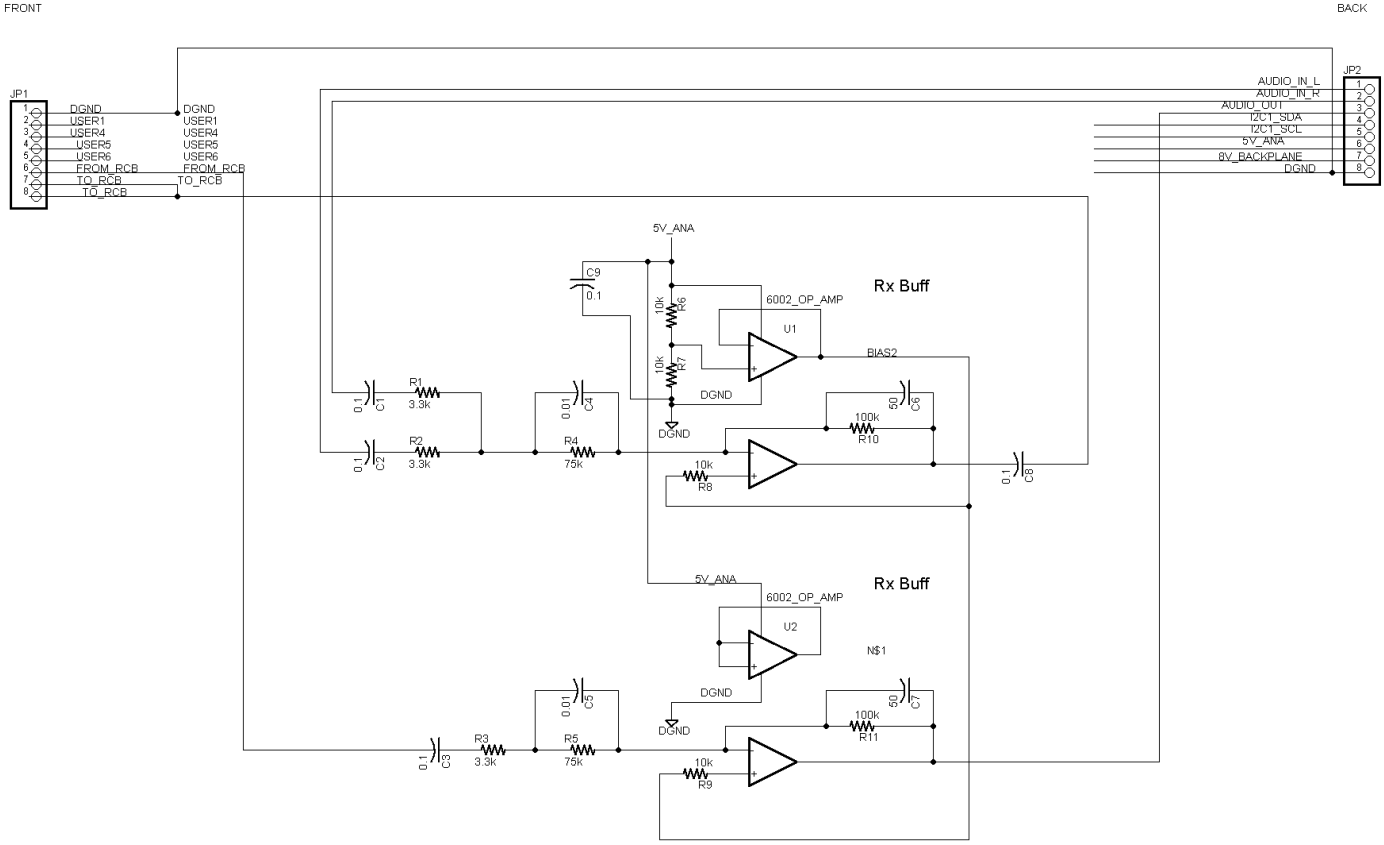


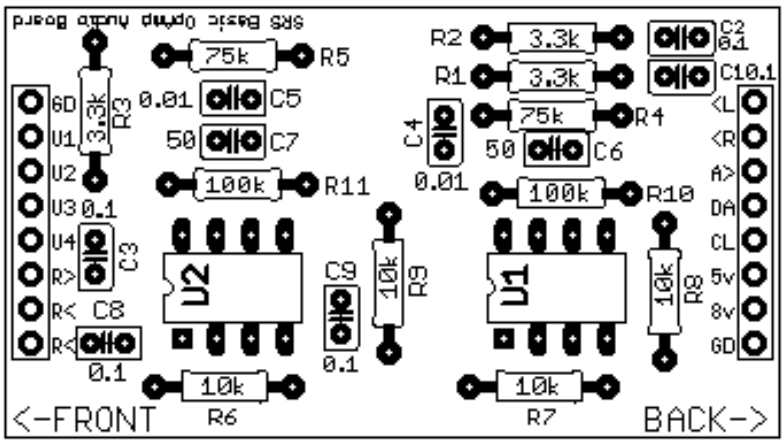
Parts Placement Diagram



Audio Processing Board - Buffer

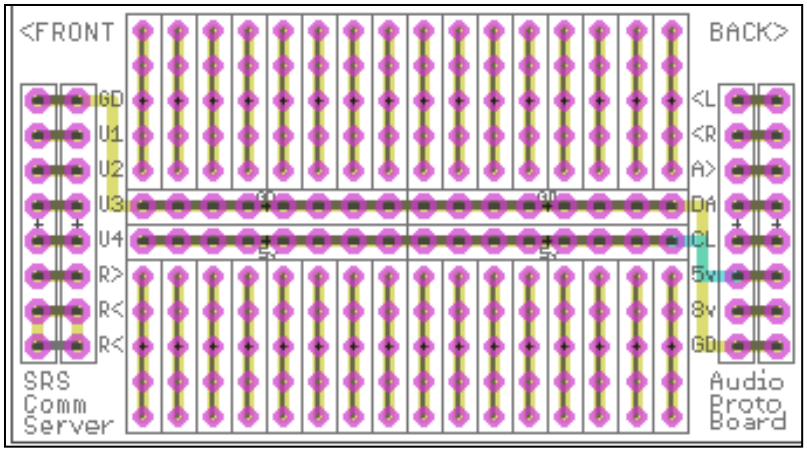
The audio buffer board provides a way to tailor the audio frequency response in and out of the CommServer's audio CODEC. The user can change resistor or capacitor values to add pre-emphasis or de-emphasis, change gain, etc.





Audio Processing Board – Proto Board

The audio proto board allows the user to build any kind of audio processing interface between the CommServer audio CODEC and the RCB.



CPU I/O Connections

BeagleBone CPU Pins	**Usage**
Analog I/O	
A/D 0	+12v backplane voltage monitor
A/D 1	+8v backplane voltage monitor
A/D 2	+5v local voltage monitor
A/D 3	Unused

All A/D inputs are a 10k series resistor and a 1k resistor to ground to form a voltage divider.

Digital I/O	
GPIO2_1	Logic output to RCB COR logic input. Used to key up the repeater control system.
GPIO0_26	Logic output to RCB PL decoder logic input.
GPIO1_14	Logic input from RCB PTT output. Pulled high. Used to detect activity on the repeater system.
GPIO1_12	Upper user programmable status LED
GPIO1_29	Lower user programmable status LED
GPIO1_15	Net reset. Controls reset to the repeater control system net_reset signal on the backplane
GPIO0_27	RF reset. Controls reset to the Xbee data radio
Serial UARTS	
UART 0	Console. 3.3v TTL logic brought to the 6 pin SIP header connector along the board edge. This connection is designed to work with a FTDI USB to TTL converter cable.
UART 1	Connected to the repeater control system master CPU UART2.
UART 2	Connected to the DB9 RS232 connector on the front edge of the board.
UART 3	Not available.
UART 4	Xbee data radio socket
UART 5	Not available.

Software Environment

The CommServer is built on Ubuntu, a very popular linux distribution. The CommServer software includes a variety of pre-installed programs, libraries and data files to make the system work properly.

Default root system login
Login: root
Password: sierra

File Structure

Just about all of the SRS developed software can be found in the directory /home/srs
Within that directory, you will find the following sub-directories…

Apps	Contains sub-directories with various installed applications
Audio	Audio data files
Install	Programs ready to be installed
Log	Log files
Python	Python development
Startup	Startup scripts
Scripts	Various scripts in development
Special	Not used
Test	Temporary working directory
Web	Web site data files

/home/srs/apps/tel
This folder contains the voice telemetry and command processing programs and scripts.
To launch the voice telemetry system: python /home/srs/apps/tel/command

/home/srs/apps/dcn
This folder contains the device control network (DCN) python and script programs.

Most of the CommServer apps are written in Python or BASH scripts.

Voice Telemetry System App

The voice telemetry system provides a mechanism to trigger the playback of pre-recorded voice phrases based on command decoding in the master CPU. The control program called “command” can also decode functions and trigger the execution of a python program or BASH script file.

How it works…

When a command is decoded in the master CPU of the repeater controller, let’s say *123, if that command is a user written macro, it will cause one or more built in commands to be executed. For example *123 can cause link 1 to be turned on by executing a built in C3311. In the configuration program (“config”) you would define the macro *123 to cause the following to be executed: C3311

You can add an instruction to send a serial command to the CommServer with the S377 command. Using the syntax S337[xxxxx] where the controller will send “xxxxx” to the CommServer. In our example your macro would be written as…

C3311 S377[L1ON]

This macro will turn on link 1 because of the C3311 built in command AND then send the string “L1ON” to the CommServer board. This is an arbitrary string you can make up as long as the command program on the CommServer is looking for the same command string.

On the CommServer, there is a program called “command” which must be running to intercept the command string from the repeater controller. In our example it is looking for the string “L1ON” for “Link one on”. The command program is written in Python and you can modify the source code if you like. Basically the command program will evaluate the new string, “L1ON”, and look to see if it is in it’s look up table of commands.

If the command “L1ON” is found, the command program will run the code associated with it.
For example “os.system(say + “link 1 on”) Which will speak “Link one on”
Or “os.system(python /home/srs/test/blork”) Which will run a python program called blork in /home/srs/test

There is a convenient utility that we include called “speak”. This is a python program is called like this…

python speak hello test 1 2 3

“python” will invoke the python program run time.
“speak” is the python program we wrote that does the speaking.
“1 2 3” are the list of arguments being sent to the speak program.

The way speak works, it will key up the repeater system, then cycle through each one of the arguments, 1 2 3, and attempt to play the associated audio files. Specifically /home/srs/audio/1.wav then /home/srs/audio/2.wav etc.
Then when it’s all done, it will unkey the repeater system.

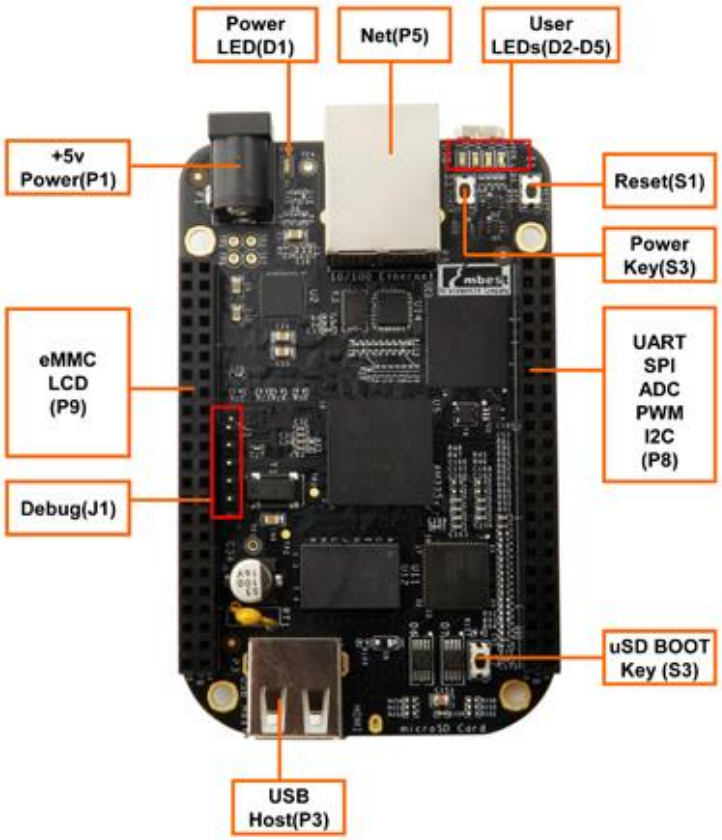
You can use the built in wav files or record your own and just replace them in the /home/sys/audio directory. You can also add words to the vocabulary by just dropping new .wav files into the same folder. If you ask speak to say a word that is not in the audio directory, it will just skip the word. A word can be a full phrase as long as you like. For example a wave file called srs.wav could speak “welcome to sierra radio systems” if that is the audio content of the srs.wav file.

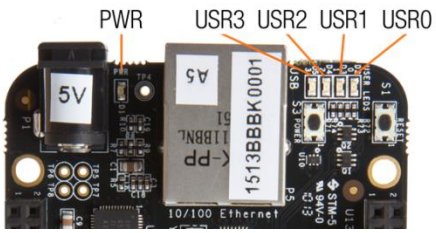
Linux System Software

INTRODUCTION

The CommServer runs on a standard distribution of Ubuntu Linux ported to the BeagleBone CPU board. This section describes the step by step installation and configuration of a new OS image and related system level application software.

SECTION 1 - PHYSICAL REFERENCE



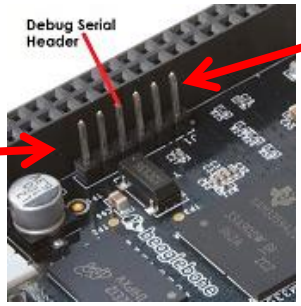


LED's

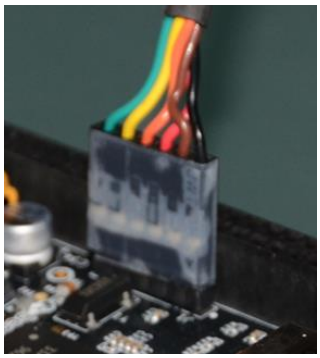
SECTION 2 - PHYSICAL CONNECTIONS

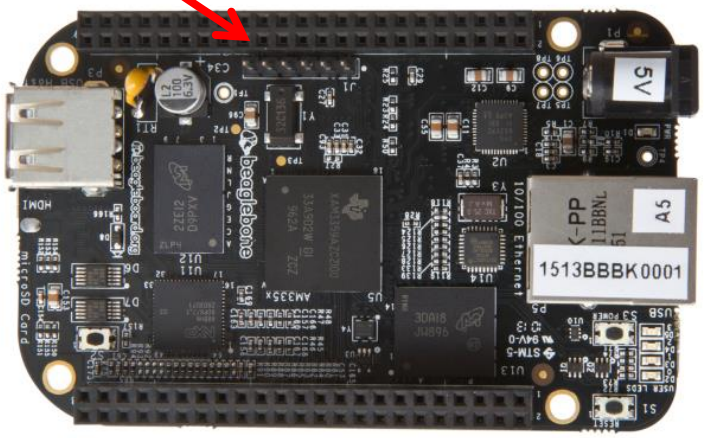
The easiest way to configure a new Beaglebone CPU board is to set it up before you plug it on to the SRS CommServer host adapter board. Take the Beaglebone CPU and connect 5V DC, ethernet and the console serial / USB cable.

Green



Black





← 5 VDC power input

← Ethernet

The USB/Serial console cable is made by FTDI and is a part number FTDI TTL-232R-3V3
These cables are available from Digikey as part number 768-1015-ND and sell for about $20.

SECTION 3 – PREPARE THE SD CARD AND INSTALL THE UBUNTU OS

❑ Get a micro SD card, at least 4 GB in size

❑ Download and install the following Windows PC programs

- ❑ Download and install the Win32 Disk Imager program from Sourceforge
  http://sourceforge.net/projects/win32diskimager/files/latest/download
- ❑ Download and install Winscp – Windows secure copy program using the ssh port
  http://winscp.net/download/winscp561.zip
- ❑ Download and install TeraTerm – Dumb terminal for ssh and serial port
  http://sourceforge.net/projects/win32diskimager/files/latest/download

❑ Download the Ubuntu "flasher" image
https://rcn-ee.net/deb/flasher/trusty/BBB-eMMC-flasher-ubuntu-14.04-console-armhf-2014-07-06-2gb.img.xz

❑ Image the micro SD card
Run Win32 SD Disk Imager to copy the Ubuntu Disk Image to the 4 GB Micro SD card.
Rename **`/boot/uboot/flash-eMMC.txt`** to **`flash-eMMC.txt__orig`**

SECTION 4 – CONFIGURE THE UBUNTU OS

❑ Login through the console port
Login `ubuntu`
Password `temppwd`

❑ Set the root password to 'sierra' or whatever you want to make it
```
sudo su root
sudo passwd
sierra
```

❑ Configure the network – Assign a static LAN IP address
`nano /etc/ssh/sshd_conf`
Change the port to `7171`

❑ Configure the network – Configure the interfaces file
`nano /etc/network/interfaces`

```
iface eth0 inet static
address 192.168.1.71
netmask 255.255.255.0
network 192.168.1.1
broadcast 192.168.1.255
gateway 192.168.1.1
```

❑ Configure the network – Check the resolv.conf file

```
nano /etc/resolv.conf

domain localdomain
search localdomain
nameserver 192.168.1.1
```

❑ Reboot the Beaglebone

❑ Log into the Beaglebone through the console

Login `root`
Password `sierra`

❑ Check the host name file

```
/etc/hostname
```

The file should look like this…

```
127.0,0,1 localhost
127.0.1.1 srs
```

Your Beaglebone should be connected to the internet

❑ Check to make sure you have a connection by pinging a remote computer

```
ping www.yahoo.com
```

You should see a return from the remote host. If not, you are not connected to the internet.

If you have a good internet connection, it's time to start installing software packages.

SECTION 5 – INSTALLING SOFTWARE PACKAGES

- ❑ Install the following applications. When prompted to continue, say "Y" for yes.
  - ❑ `apt-get update`
  - ❑ `apt-get install minicom`
  - ❑ `apt-get install alsa-utils`
  - ❑ `apt-get install python-serial`
  - ❑ `apt-get install mpg321`
  - ❑ `apt-get install mpg123`
  - ❑ `apt-get install mysql-server`

  When installing mysql-server, assign a password.
  Typically the password will be: sierra

- ❑ Install the Adafruit python BBIO library.

This library provides a very convenient set of IO functions for use in python programs.

- ❑ `sudo ntpdate pool.ntp.org` Set system date / time
- ❑ `sudo apt-get install build-essential`
- ❑ `sudo apt-get install python-dev`
- ❑ `sudo apt-get install python-setuptools`
- ❑ `sudo apt-get install python-pip`
- ❑ `sudo apt-get install python-smbus`
- ❑ `easy_install -U distribute`
- ❑ `sudo pip install Adafruit_BBIO`

- ❑ Test the Adafruit_BBIO install

```
sudo python -c "import Adafruit_BBIO.GPIO as GPIO; print GPIO"
```

Should return this or similar:

```
<module 'Adafruit_BBIO.GPIO' from '/usr/local/lib/python2.7/dist-packages/Adafruit_BBIO/GPIO.so'>
```

- ❑ Optional packages that can be installed
  - ❑ `apt-get install sox`
  - ❑ `apt-get install festival`
  - ❑ `apt-get install flite`
  - ❑ `apt-get install php5 libapache2-mod-php5`

SECTION 6 – FINAL SYSTEM SOFTWARE INSTALLAION STEPS

Edit the uEnv file to enable key peripherals on bootup.

- ❑ Edit `/boot/uboot/uEnv.txt` file

```
nano /boot/uboot/uEnv.txt
```

Change the configuration lines to the following…

```
optargs=capemgr.disable_partno=BB-BONEELT-HDMI,BB-BONELT-HDMIN
optargs=capemgr.enable_partno=BB-BONE-AUDI-01,BB-UART1,BB-UART2,BB-UART4,BB-UART5
```

- ❑ Create the SRS app folders

```
cd /home
mkdir srs
```

Now using scp, copy the SRS install file structure from your hard drive to the /home/srs folder on the Beablegone.

Linux Application Software

INTRODUCTION

The CommServer stores most, if not all, programs and data in the /home/srs folder. The sub folders typically used include…

/home/srs/audio

Audio .wav and .mp3 files, and other audio related software and scripts.

/home/srs/scripts

Various scripts to do various tasks.

/home/srs/apps

This folder contains more folders, one for each app installed on this Beagle.

/home/srs/apps/tel

This is the voice telemetry system folder.

/home/srs/apps/dcn

This is the device control network program folder.

/home/srs/test

Programs used to test various functions on the CommServer board.

Device Control Network

INTRODUCTION

The Sierra Radio Systems Device Control Network (DCN) provides a way to allow multiple devices communicate on a wired or wireless network. The primary application for the DCN is to allow a master computer or device, to control and monitor a network of real-time control devices. The DCN specification defines four components, the physical and RF connections and electrical signals, data link layer, the application layer and network topology. The purpose of the physical layer is to define the standard mechanical connectors, pin assignments, signaling voltages, and RF frequencies . The next layer up, the data link layer, defines the format of data packets sent on the network. The third layer defines the format of the payload being sent from point A to point B and finally recommendations for the network topology.

SECTION 1 - PHYSICAL CONNECTIONS AND ELECTRICAL SIGNALING

The DCN is a “dual-band” system meaning that data may be transmitted over wired or RF communications paths or both.

Physical Connections

The control protocol can be transmitted over any type of communications medium. The most common connections are wired through an RS232 or RS485 connection, a wireless connection, typically using an RF mesh data network or over ethernet.

Wired RS-485

The RS485 wired implementation uses commonly available Ethernet CAT5 cable and RJ-45 connectors. While the DCN protocol has nothing at all to do with Ethernet except that we take advantage of the wide availability of premade cables. The CAT5 cable provides 8 wires which carry the network traffic and power. Many devices have two RJ-45 connectors wired in parallel. This allows for easy daisy-chaining of multiple devices using CAT5 cable. This makes it easy to add more devices to the network without the need for any kind of hub or switch.

RJ-45 Cable Assignments

1 – Network data signal A	5 - Ground
2 – Network data signal B	6 - Reserved
3 – Reserved	7 - +12 VDC
4 – Ground	8 - +12 VDC

Wired RS-485 Electrical Signaling - continued

The electrical signaling used is based on RS-485. This signaling technique is a half-duplex, differential pair that allows multiple devices to be connected to a single pair of wires. RS-485 also has the advantage of allowing devices to be spread over 1000's of feet of cable without the need for signal conditioning or repeaters.

Power can be supplied by any device and delivered to all devices on the network. If a device can supply power to the network, there must be a way to disconnect the power, usually through a jumper block. Only 1 device is allowed to supply power to the network at a time. Network voltage should be between 12-14 VDC. This provides enough of headroom to power any DCN compatible device.

Wireless RF Data Network

The DCN can also use RF modules that operate on 2.4 GHz or 900 MHz. If mesh network radio modules are used, if you add new nodes to the RF network, the automatically become part of the network. This is particularly convenient when extending the range between devices. Each node can be thought of as a serial port that taps into an invisible network of other devices. When data is sent into the serial port of the data radio, the packet will be delivered to every data radio in the network and the packet will be transmitted out of the RF module's serial port into the local device's CPU.

A network can consist of a mixture of wired RS-485 and RF data network enabled nodes.

In the 2.4 GHz version of the RF network, there are 16 possible RF carrier channels. Our default pre-configured data radios are assigned channel "E". Note that these frequencies are in the same band of frequencies as WiFi, Bluetooth, ZigBee and other data services. While interference is rare, you can always change channels using the Digi International configuration program called X-CTU.

Channel*	SC	Frequency (MHz)
B	1	2.405
C	2	2.410
D	4	2.415
E	8	2.420
F	10	2.425
10	20	2.430
11	40	2.435
12	80	2.440
13	100	2.445
14	200	2.450
15	400	2.455
16	800	2.460
17	1000	2.465
18	2000	2.470
19	4000	2.475
1A	8000	2.480

Xbee Firmware Configuration Guide

Introduction

We use the Digi International Xbee data radios running the Digi-Mesh protocol stack for our station monitoring and control applications. To install firmware and set configuration parameters in each radio module, you will use a program called X-CTU available free from Digi-International.

Firmware Stack

The reason we use the Digi-Mesh stack is the ease of addressing nodes on a network and the ability for the network to grow by simply dropping new nodes in the network. Any data packet that enters the network will hop from node to node until the message goes to all nodes. Each device controller will inspect the packet to determine if the message is for that particular node or not.
The firmware comes pre-installed and the devices are pre-configured for RF channel E and a network ID of 7777. This is similar to an SSID in a WiFi network. If you need to re-flash your firmware or change the configuration of your data radios, read the following instructions…

❑ Step 1 – Install X-CTU PC software

The X-CTU software can be downloaded from the Digi-International web site.

❑ Step 2 - Install the Digi-Mesh firmware

For the 2.4 GHz xBee (1mw) module, install modem firmware type: **XB24-DM**
For the 2.4 GHz xBee PRO module, install modem firmware type: **XBP24-DM**
For the 900 MHz xBee PRO module, install modem firmware type: **XPB09-DM**
The field “Function Set” should be set to XBEE PRO DIGIMESH 2.4

There are only a few configuration parameters required to get your data radio module up and running on the network.

❑ Step 3 - Set network ID

This is the “name” of the RF network that all units will join.
This is an arbitrary 4 digit number. You can pick any number you want. You can run multiple independent networks on the same RF channel by setting some modules to one address and other modules in another network to another address and they will ignore each other.

Sierra Radio default recommendation:

Set Networking ID – Modem VID to 7777

xBee Firmware Configuration Guide

❑ Step 4 – Set operating channel

This is one of the 12 available RF carrier channels that the network will operate on. Channels are assigned values from 0C to 17 (C, D, E, F, 10, 11, 12,13,14,15, 16, 17, 18) represented as a hex value. The RF carrier channels overlap with other devices in the 2.4 GHz band including WiFi and Bluetooth networks. Generally speaking you can pick any channel and it will work fine even with these other networks in operation. If you are in an RF dense environment and want to take all steps possible to avoid interference, pick a channel that does not overlap with your local WiFi networks. See the frequency table in the appendix. We recommend using channel E.

Set Network CH – Operating Channel to E

❑ Step 5 – Set device address

Setting the high order destination address to 0 and the low order destination address to FFFF will put the radio in broadcast mode. All data packets received will be sent to the serial port. In a SRS environment running the DCN protocol on the controllers CPU, the device address decoding is done by the local CPU. This makes the network operate as a simple mesh of all devices where any data going into one data radio's serial port will appear at the output of all data radios. The CPU will then process the payload of the packets.

The SRS default recommendation:

Set Addressing DL – Destination Address Low to FFFF
Set Addressing DH – Destination Address High to 0

❑ Step 6 – Set serial port configuration

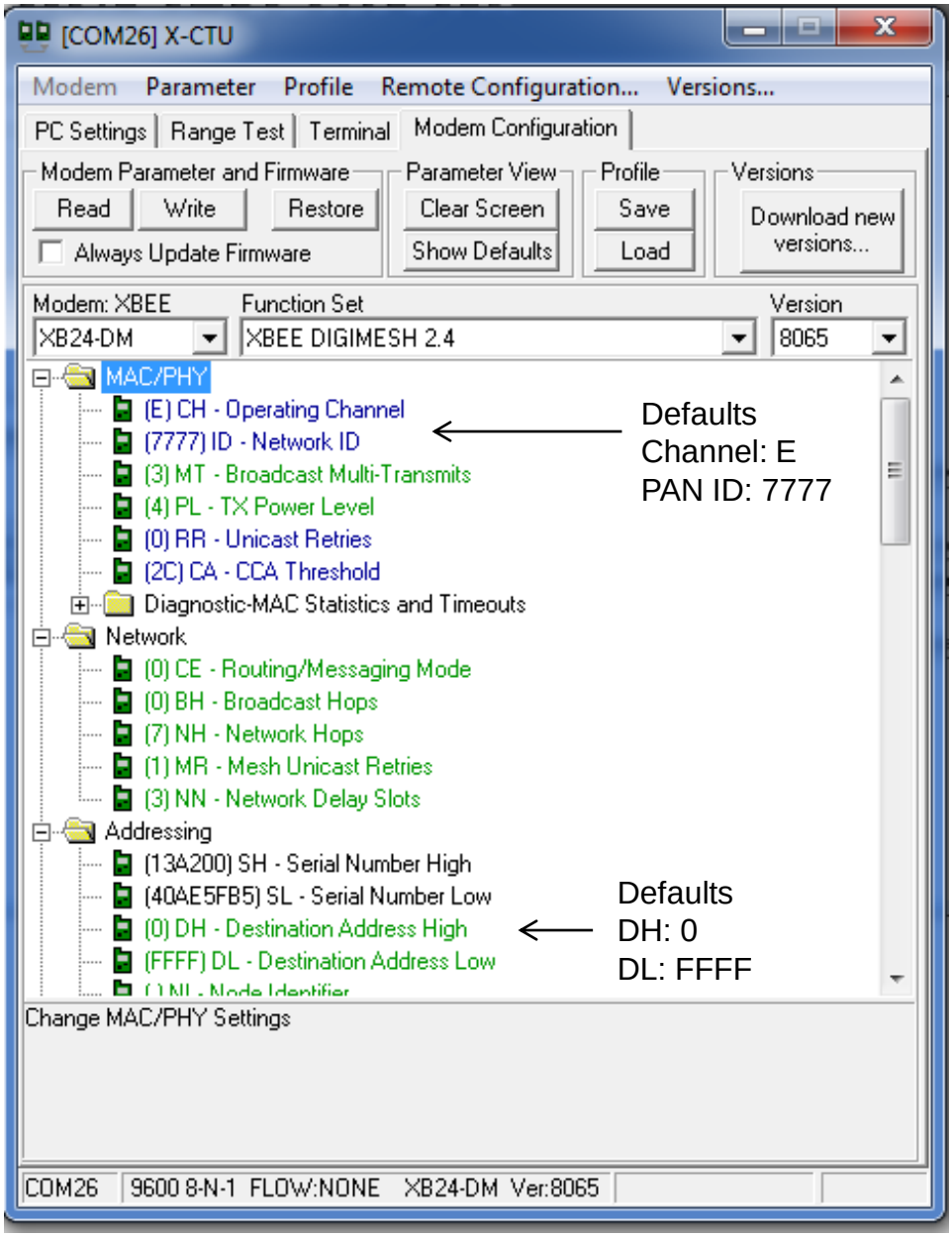
The data radio module's serial port can be configured to one of eight standard baud rates including 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, and 230400 baud.

The SRS default recommendation:

Set Serial Interfacing BD – Baud Rate to 9600

Digi International
X-CTU firmware configuration software

Firmware Type: XB24-DM



Defaults
Channel: E
PAN ID: 7777

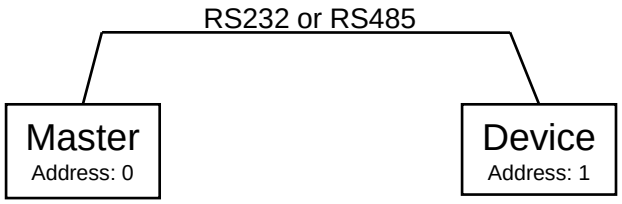
Defaults
DH: 0
DL: FFFF

Network Topology

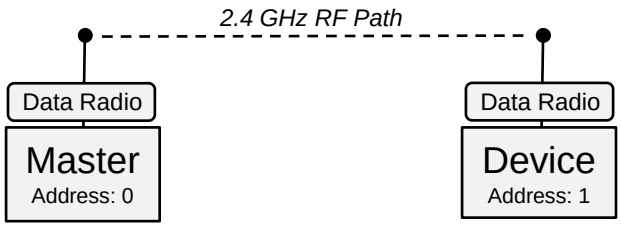
Overview

The Sierra Radio Device Control Network can be build with wired or wireless devices or a combination of both. The network protocol is completely independent from the physical communications channels. In the DCN, all devices are listening all the time to the network. One device, typically a computer, is the master and all other devices are slaves. All slaves remain quiet until the master communicates with them. Every device, including the master has unique address. Typically the master is address 0 (zero) and devices are numbered 1, 2, 3 and so on. Think of the network as one big “party line” where everyone is connected all the time. When anyone transmits, all devices can hear it. There are three commonly used physical connections used in a DCN. They are RS232 for point to point applications with one master and one slave device. RS485 which is a serial interface but very different from RS232. RS485 uses differential signaling on a pair of wires (called the A and B wires) and operate half duplex. The advantage to RS485 is the ability to hang multiple devices on the A/B pair and the long physical distances that can be used. The third common connection type is an RF data channel. The DCN data radios take serial data into their UART and package the data up in a packet and transmits them to the other data radios in the network where the same data comes out the UART on the other end and into the local devices microcontroller.

Lets look at some typical network configurations.



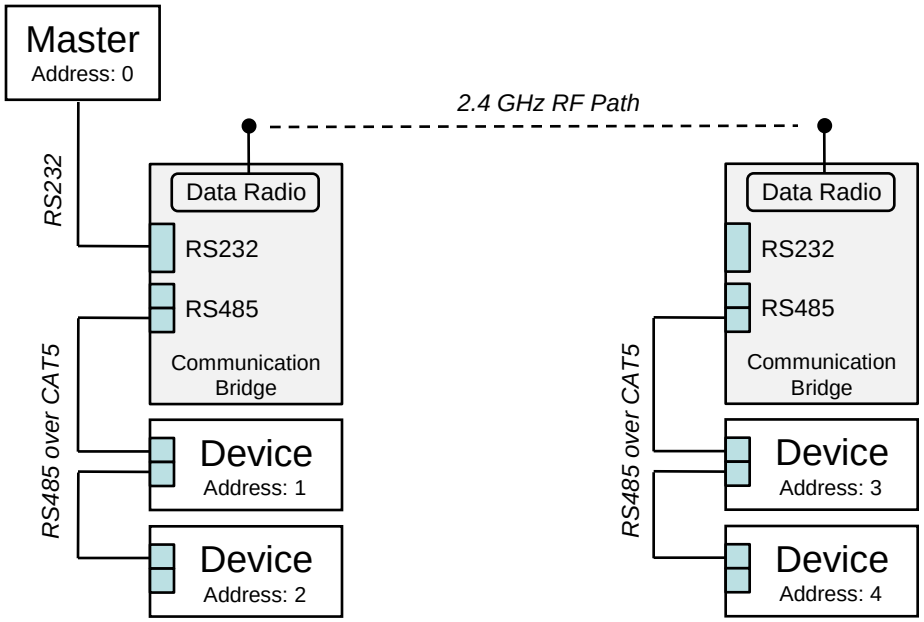
In the most simple example, the master can talk to a slave over a wired connection. Some Sierra Radio products have RS232 ports but all devices have the RS485 DCN connector which is a RJ45 (ethernet) modular connector.



When using a RF data radio module instead of a wired connection, the logical behavior is exactly the same. Data that comes out of the master’s UART will arrive at the input to the UART on the slave.

Network Topology

Wired and wireless connections can be combined. One example is where the local devices are connected with cable and the remote devices connected with a wireless data connection. For example…



In this example, all devices are listing at the same time to the DCN regardless of the medium of communications.

SECTION 2 - DATA RATES AND PACKET FORMAT

The DCN data protocol sends ASCII data at 9600 baud, non-inverted, 8 bits, no parity.

The protocol defines the format of packets of data transmitted on the network. The simple way to think of a packet is a string of ASCII characters that contains the payload to be transmitted from point A to point B and the additional characters necessary to provide synchronization, packet type identification, addressing, and error checking.

A typical packet looks like this...

/A01:RY1,1:4D <13>

Start of Packet	Packet Type	From Address	To Address	:	Payload	:	LRC Value	End of Packet

Start of Packet

A forward slash character / is reserved for the start of packet indication. When a slave device sees the slash, it knows there is a new packet.

Packet Type

The packet type character defines the format of the packet and instructions for how the packet is to be interpreted.

Packet types include direct, addressed and addressed with no error checking.

They start each packet with the characters //, /A, and /0 (zero) respectively.

Direct Packet type // (Example: //reset)

This is a very simple format that is intended only for use in a system with a single node. The format for a direct packet is simply the header // and the payload.

As you can see, there is no address or error checking. If multiple nodes are on the network and a direct command is issued, all nodes will decode and execute the command.

This can be very convenient to send master commands to all nodes but there is no error checking.

Addressed Packet type /A (Example: /A01:reset:39)

These packets start with /A and contain the source and destination address, payload and error check data.

Addressed Packet, no error checking type /0 (Example: /001:reset:34)

This is an Addressed packet that ignores the error checking field. This is used for manual entry of network addressed packets without the need to calculate the error checking value.

Device Addresses

You can assign any node an address using any printable character. However, for maximum functionality, we recommend using numbers as the addresses.

.

Pre-assigned default device addresses

0	System master. .
1-9	Devices 1-9
1	Site Controller
2	Remote RF coax relay
3	GPIO board
4	Not assigned
5	Watt meter
6	Not assigned
7	RadioRouter audio mixing and switching device
8	Not assigned
9	Not assigned
A-Z	Not assigned
*	Broadcast to all devices.

Any other characters are reserved and should not be used.

If you add a second device and the default device address is already in use, just pick another.

Payload

The payload is application dependent. See standard command summary.

Error Check Value

The error check value is an 8 bit LRC.. The LRC is applied to all characters in the packet except the initial start of packet character /. Of course the LRC is not applied to the LRC characters either.

Example: with the packet /A01:reset:39 the LRC is applied to A01:reset:

End of Packet

The end of packet character is a carriage return, ASCII byte value 013 (decimal). When an end of packet character is encountered, the input buffer is evaluated.

The evaluation process identifies a packet by finding the start of packet synchronizing character / and extracts the buffer contents up to the end of packet character.

The command parser then extracts the packet type, addresses, error check value and payload.

The error check value is calculated and compared to the packets error check value. If the values do not match, the buffer is flushed.

If the packet is good, then the to address is examined. If the to address is the same value as the devices address, the packet analysis will continue, if not, the packet is ignored and flushed from the buffer.

STANDARD COMMANDS – ALL DEVICES

Administrative Commands

Command SETADDR,<string>
Example //SETADDR,5
Definition Sets the network address for the RadioRouter to <string>, where <string> is a single, printable ASCII character.

The default address for the RadioRouter is 9

While the address character can be any single character we recommend the following guidelines for address programming.

0	Reserved for the controlling PC
1	Reserved for a hardware control head
2…9	Recommended for all user devices
A…Z	Also available for user devices
*	Reserved for broadcast
All others	Punctuation is reserved for system use. Avoid lower case letters to prevent confusion and accidental upper case translation.

Command REBOOT
Example //REBOOT
Definition This is a soft reboot command will re-start the RadioRouter.

Command PING
Example //PING
Definition This will return the name, address and type of connected device.

Command ROLLCALL
Example //ROLLCALL
Definition This will return the name, address and type of connected device from all devices on the network. Each device will wait for a short period of timebefore sending its response. The delay time is calculated based on the device address. For example an address of 5 will wait 5 * 100 ms = 500 ms. This will minimize or avoid collisions.

Command STATUS
Example //STATUS
Definition This will return useful status information about the connected device.

STANDARD COMMANDS – ALL DEVICES

Command	STATE
Example	//STATE
Definition	Returns the state of the device. This typically includes the state of relays, digital inputs, analog inputs, etc. The specific format varies depending on the device type. The return format will be UPDATE,From, To, Arg1,Arg2,Arg3, etc

Hardware Configuration

Key

Insert jumper

Do NOT insert jumper

Configuration – Connection to RCB