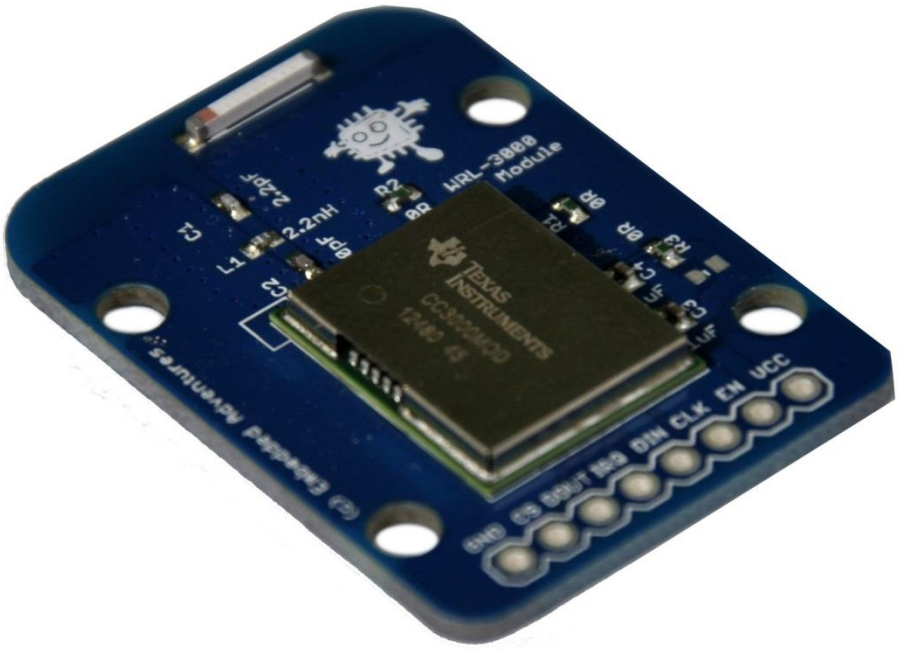
Device: WRL-3000v2

This document Version: 1

Matches module hardware: [27 Jun 2013 v2]

Date: 31 July 2013

Description: CC3000 WiFi module



Contents

Introduction

The WRL-3000 is a WiFi module for embedded systems, based on the CC3000 chip from TI. It contains the entire WiFi and TCP/IP stacks and requires relatively straightforward SPI commands to control it. No stack is required on the side of the microcontroller.

Features

It's got 802.11B/G goodness, runs down to 2.7V, can transmit at up to 7Mbps and can operate down to -20°C. Always handy if you're putting your project in the freezer.

But best of all, it has a neat technology that TI have called Smart Config. With a little iPhone / Andriod app, or a java applet on a webpage, you can configure the WRL-3000 and get it connected to a network without having to tell it directly about the network password etc.

The design utilises the reference antenna and as such is similarly covered by the FCC module approval process.

Connections

The WRL-3000 module has one connection port.

VCC	Power connection. The WRL-3000 requires up to 300mA. Voltage should be between 2.7 and 4.8V.
EN	Pull high to enable module Pull low to disable module and enter sleep mode
CLK	Data Clock
DIN	Data IN to the module
IRQ	Interrupt request line
DOUT	Data OUT of the module
CS	Chip select (active low)
GND	Ground connection

General notes

This document goes into detail on the commands and events that we have tried and tested and know work successfully. We provide sample code that uses these functions.

There is more that the CC3000 module itself can do – not least of which, includes UDP sockets, select calls on sockets and so on.

As we confirm in our own code that these work we will be updating this document with our findings. Or if you have suggestions, please email us!

To explore these further commands on your own, download the SDK from TI at http://processors.wiki.ti.com/index.php/CC3000_Wi-Fi_Downloads

Communicating with the WRL-3000

TI's website is a good place to start in learning about how to communicate with the module:

http://processors.wiki.ti.com/index.php/CC3000_Host_Driver_Porting_Guide

However, TI's documentation is focussed on how to get their API code working on your microcontroller. We didn't like their code so we wrote ours from scratch. We hope that this document forms the basis of your learning about this module. If you get it working on a new platform, we would love to hear from you so we can share this work with others.

SPI protocol

At its base, the SPI protocol is the way in which you get information into and out of the module.

Data is sampled at the falling edge of the clock cycle (CPHA=1)

Clock is idle when it is low (CPOL=0)

The entire packet must be an even number of bytes. If the total number of bytes is not even, a padding byte is added to the end. The Payload Length *includes* the padding byte.

SPI write operation

The SPI write is used when the microcontroller wants to send something to the WRL-3000. It is initiated by the microcontroller when it pulls the CS line low.

For an SPI write operation:

- Pull CS line low
- Wait for IRQ line to go low (indicates WRL-3000 is ready)
- Send SPI packet header

- Send HCI packet
- Send padding (if uneven number of bytes)

An SPI Write packet is made up of the following with one byte per cell:

SPI Header (WRITE)					Payload	
SPI Opcode	Payload Length MSB	Payload Length LSB	Busy	Busy	HCI packet	Optional Padding

The SPI Opcode byte is SPI_OPERATION_WRITE (0x01)

The Busy byte is ignored but should be set to zero.

The Payload contains an HCI packet.

Note that the very first SPI transaction needs to be in a particular magic format and timing (see startup section below).

SPI read operation

The SPI read is used when the WRL-3000 wants to send something to the microcontroller. It is initiated by the WRL-3000 when it pulls the IRQ line low.

To handle an SPI read operation:

- IRQ line is pulled low by WRL-3000 (indicating it has something to be read)
- Pull CS line low
- Send SPI packet header
- Send HCI packet
- Send padding (if uneven number of bytes)

An SPI Read packet is made up of the following with one byte per cell:

SPI Header (READ)					Payload	
SPI Opcode	Busy	Busy	Payload Length MSB	Payload Length LSB	HCI packet	Optional Padding

The SPI opcode is SPI_OPERATION_READ (0x03)

The Busy byte is ignored but should be set to zero.

The Payload contains an HCI packet.

The provided software implementation for the WRL-3000 tried to deliver the simplest implementation possible. However if you are looking at a DMA implementation, it's

clear from this packet structure that transferring 5 bytes (for the SPI header) gives enough information to then do a second DMA transfer of [Payload Length] bytes. This payload can then be handed to the HCI layer for processing.

In our simple implementation, the microcontroller is busy waiting for all packets and then handles processing. It's not as efficient, but it makes the protocol clearer.

HCI protocol

Right. So you've got yourself an SPI packet. What now?

The next step is to understand the HCI packet that is contained within it. HCI stands for Host Communication Interface and represents the guts of the communication between the WRL-3000 and your microcontroller.

HCI Command

An HCI command is something that the microcontroller wants to tell the WRL-3000 to do. It is usually followed by an HCI Event that at least acknowledges the command but may also pass back data to the microcontroller.

HCI Header (Command)				Payload
HCI Message Type	Opcode (LSB)	Opcode (MSB)	Args Length	Arguments

The HCI message type is HCI_TYPE_COMMAND (0x01)

The Opcode is a 16 bit number split over two bytes.

The Args Length is up to 255 bytes.

The Arguments themselves are Args Length bytes long, unsurprisingly. In the following section, you can see a list of the commands and the Opcodes associated with them. Or you can just grab them from our sample source code.

HCI Event

An HCI Event occurs when the WRL-3000 wants to indicate something to the microcontroller. It may be a response to a command (using effectively the same HCI Command opcode), an acknowledgement, in other words. It may also be an "asynchronous" or, as the CC3000 documentation puts it, an "unsolicited" event, where the WRL-3000 is communicating that some sort of event has happened that was not immediately triggered by a microcontroller required.

HCI Header (Event)					Payload
HCI Message Type	Opcode (LSB)	Opcode (MSB)	Args Length (LSB)	Args Length (MSB)	Arguments

The HCI message type is HCI_TYPE_EVENT (0x04). Note that in this case the arguments length is 16 bits long.

HCI Data

HCI Data is used to transport the data received from or sent to a socket.

HCI Header (Event)					Payload	
HCI Message Type	Opcode	Args Length	Payload Length (LSB)	Payload Length (MSB)	Arguments	Data

The HCI message type is 0x02

The Payload Length is the total of the Arguments Length + Data Length. To calculate the actual data length, subtract the Args Length from the Payload Length.

Note that the Opcode and Args Length are both 8 bit numbers.

Startup

Startup requires some particular timing.

- Disable the module by setting EN low
- Wait for IRQ line to go high
- Enable the module by setting EN high
- Wait for IRQ line to go low
- Set CS line high
- Wait 50 µs
- Send via SPI: SPI_OPERATION_WRITE
- Send via SPI: 0x00 (length MSB)
- Send via SPI: 0x05 (length LSB)
- Send via SPI: 0x00 (busy 0)
- Wait 50 µs
- Send via SPI: 0x00 (busy 1)
- Send via SPI: HCI_TYPE_COMMAND
- Send via SPI: HCI_COMMAND_SIMPLE_LINK_START (LSB)
- Send via SPI: HCI_COMMAND_SIMPLE_LINK_START (MSB)

- Send via SPI: 0x01 (1 byte payload)
- Send via SPI: PatchesRequest (0x00 = Don't load patches 0x01 = Load patches)
- Set CS line low
- Wait for IRQ line to go low
- < Receive HCI>

Note that the normal state is to run with patches loaded. There are certain circumstances during flash updating where it is desirable to startup without patches.

Following this special timing for the startup command, normal commands can be issued. The first command that should be issued is:

HCI_COMMAND_READ_BUFFER_SIZE

which returns:

HCI_EVENT_READ_BUFFER_SIZE

Once this is received, the WRL-3000 is good to go.

HCI Commands / Events - General

HCI_COMMAND_READ_BUFFER_SIZE

Find out how many buffers the WRL-3000 has available. Typically this command is used at startup.

Opcode: 0x400B

Args: None

Total HCI payload length = 0

Returns via HCI_EVENT_READ_BUFFER_SIZE message:

uns8	Free buffers
uns16	Buffer length

HCI Commands / Events - WLAN

HCI_COMMAND_WLAN_CONNECT

Initiate the connection of the WRL-3000 to a given WiFi router.

Opcode: 0x0001

Args:

uns32	Magic	0x0000001c
uns32	SSID length	<length of SSID string>
uns32	Security Type	0=Unsecured 1=WEP 2=WPA 3=WPA2
uns32	Magic	0x10 + <length of SSID string>
uns32	Key (password) length	<length of key>
uns32	Magic	0x00000000
6 x uns8	BSSID	0x00 0x00 0x00 0x00 0x00 0x00
L x uns8	SSID	<SSID string> L = length of SSID string

Total HCI payload length = 28 + <length of SSID string> + <length of key string>

Returns: HCI_EVENT_WLAN_CONNECT, followed by asynchronous connection messages

HCI_COMMAND_WLAN_ DISCONNECT

Disconnect from the current wifi router

Opcode: 0x0002

Args: None

Returns: HCI_EVENT_WLAN_DISCONNECT, followed by asynchronous connection messages

HCI_COMMAND_WLAN_ SET_SCANPARAM

Set the parameters for scanning for WiFi access points.

Opcode: 0x0003

Args:

uns32	Magic	36 (0x24)
uns32	Scan Frequency (ms)	1 = default scan frequency of 10 minutes 1000+ = scan frequency in ms (minimum is 1 second) Must pull EN low and restart module for this to have an affect.
uns32	Min channel dwell time	20 = default value 100 = recommended value
uns32	Max channel dwell time	30 = default value 100 = recommended value
uns32	Max probe requests	2 = default value

	per channel	5 = recommended value
uns32	Channel Mask	Bitwise mask, up to 13 channels (0x1fff) 0x07ff = default
uns32	RSSI threshold	-80 = default
uns32	SNR threshold	0 = default
uns32	TX power for probe	2000 = default
16 x uns32	Per channel timeout between periodic connection scan (ms)	205 = default Not used in 1.11 firmware
uns32	Minimum dwell time	20 = default value 100 = recommended value

Total HCI payload length = 100

Returns: HCI_EVENT_WLAN_SET_SCANPARAM

HCI_COMMAND_WLAN_ SET_CONNECTION_POLICY

Set startup connection policy, whether to fast connect to the last connection, try connecting to stored profiles, and/or connect to open hotspots.

Opcode: 0x0004

Args:

uns32	Fast Connect	0 = Fast connect disabled 1 = Fast connect enabled
uns32	Open Ap Connect	0 = Open ap auto connect disabled 1 = Open ap auto connect enabled
uns32	Profile Options	0 = Use profile options disabled 1 = Use profile options enabled

Total HCI payload length = 12

Returns: HCI_EVENT_WLAN_SET_CONNECTION_POLICY

HCI_COMMAND_WLAN_ GET_SCAN_RESULTS

Opcode: 0x0007

Request a single scan result. Each subsequent call decrements the "number of networks left to send" until it is 0 (the entry for which will have the Result is not valid bit cleared).

Args:

uns32	Magic	0x0000

Total HCI payload length = 4

Response: HCI_EVENT_WLAN_GET_SCAN_RESULTS

uns32	Networks	Number of networks left to send
uns32	Scan status	0 = aged results 1 = results valid 2 = no results
uns8	Scan result	bit 7 = 1 Result is valid bit 7 = 0 Result is not valid bit 6 - bit 0 = RSSI value
uns8	Scan result	bit 7 – bit 6 = 0b00 open security bit 7 – bit 6 = 0b01 WEP bit 7 – bit 6 = 0b10 WPA bit 7 – bit 6 = 0b11 WPA2 bit 5 – bit 0 = SSID name length
2x uns8	Scan result	Time hotspot was found
32x uns8	Scan result	SSID Name

HCI_COMMAND_WLAN_SET_EVENT_MASK

Mask out asynchronous events

Opcode: 0x0008

Args:

uns32	Event Mask	Asynchronous events Bit set to 1 = don't send event

Total HCI payload length = 4

HCI_COMMAND_WLAN_GET_STATUS

Mask out asynchronous events

Opcode: 0x0009

Args:

uns32	Event Mask	Asynchronous events Bit set to 1 = don't send event

Total HCI payload length = 4

HCI_COMMAND_WLAN_ SMART_CONFIG_START

Start listening for the Smart Config messages

Opcode: 0x000a

Args:

uns32	Encrypted	0 = Smart config not encrypted 1 = Smart config is encrypted

Total HCI payload length = 4

Response: HCI_EVENT_WLAN_SMART_CONFIG_START

HCI_COMMAND_WLAN_ SMART_CONFIG_STOP

Stop listening for the Smart Config messages

Opcode: 0x000b

Args:

uns32	Encrypted	0 = Smart config not encrypted 1 = Smart config is envrypted

Total HCI payload length = 4

Response: HCI_EVENT_WLAN_SMART_CONFIG_STOP

HCI Commands / Events – Socket

HCI_COMMAND_SOCKET

Create a socket

Opcode: 0x1001

Args:

uns32	Domain	AF_INET is the only supported option
uns32	Socket type	One of SOCK_STREAM, SOCK_DGRAM, or SOCK_RAW
uns8	Protocol	One of IPPROTO_TCP, IPPROTO_UDP or IPPROTO_RAW

Total HCI payload length = 12

Returns: HCI_EVENT_SOCKET, HCI_STATUS is set to -1 on error or otherwise the socket descriptor used for further socket commands.

HCI_COMMAND_CONNECT

Connect an existing socket to an end point

Opcode: 0x1007

Args:

uns32	sd	Socket descriptor from previous create socket
uns32	Magic	0x00000008
uns32	Address Length	Always 0x00000008

uns16	Family	Always AF_INET
uns32	Address	Quad IP address

Total HCI payload length = 20

Returns: HCI_EVENT_CONNECT. HCI_STATUS = -1 on error, 0 on success

HCI_COMMAND_SEND

Send data to the other end of the socket.

Note that this is a data transaction, not a command.

Opcode: 0x0081

Args:

uns32	sd	Socket descriptor from previous create socket
uns32	Magic	12
uns32	Data length	Length of data portion
uns32	Flags	Not currently used

Total HCI payload length = 16

Data:

Data to be sent to the other end of the socket.

Returns: Nothing

HCI_COMMAND_RECV

Receive data from a socket. This is a combined command + data transaction.

Opcode 0x1004

Args:

uns32	sd	Socket descriptor from previous create socket
uns32	Data length	Amount of data that is requested from the socket (maximum)
uns32	Flags	Not currently used

Response: HCI_EVENT_RECV

uns32	sd	Socket descriptor
uns32	Number of bytes	Amount of data that is actually going to be returned
uns32	Flags	Not currently used

If the number of bytes returned > 0 then the data will be returned in a new transaction – a data transaction of type HCI_DATA_RECV (opcode 0x85)

HCI_COMMAND_CLOSE_SOCKET

Close socket.

Opcode 0x100B

Args:

uns32	sd	Socket descriptor from previous create socket

Response: HCI_EVENT_CLOSE_SOCKET

HCI_COMMAND_MDNS_ADVERTISE

Advertise the MDNS capabilities. Typically this is used to signal back to the iPhone app that Smart Config has completed successfully.

Opcode 0x1011

Args:

uns32	MDNS enabled	0 = MDNS disabled 1 = MDNS enabled
uns32	Service name length	
L x uns8	Service name	Where L is the service name length

Response: HCI_EVENT_MDNS_ADVERTISE

HCI Commands / Events – NVMEM

HCI_COMMAND_READ_SP_VERSION

Read the current version of the software.

Opcode 0x0207

Args:None

Returns: HCI_EVENT_READ_SP_VERSION

uns8	Ignore	
uns8	Ignore	
uns8	SP_MAJ	Major version of service pack

uns8	SP_MIN	Minor version of service pack

HCI Commands / Events - Network

HCI_COMMAND_GETHOSTNAME

Convert hostname to IP address

Opcode: 0x1010

Args:

uns32	Magic	0x00008
uns32	Name length	Length of the host name
L x uns8	Host name	Host name, L = length of host name

Total HCI payload length = 8 + length of host name

Response: HCI_EVENT_GETHOSTBYNAME

int32	Return value	-1 = error, 0 = success
4 x uns8	IP Address	In reverse order, 4 quad IP address

On Porting

You can use our source code as a base. It has been written in SourceBoost C for Pic Microcontrollers, but it should be pretty portable.

Using defines for int8, uns8, int16, uns16, int32 and uns32 will make your life easier (see pic_utils.h).

Substitute your own microcontroller commands for setting up the pins to communicate with the WRL-3000 in `cc3000_setup_io()` in cc3000.c/h.

Swap out the SPI routines in cc3000_spi.c/h as appropriate for your microcontroller.

The demo program uses well-worn library files from the Pic Pack library for communicating with humans via a serial interface; of course you will need to replace these with your own method of calling the library routines.

See also the debug.h file to debug routines – you can choose to simply use the #defines here to silence the debugging information, or turn it on using your own serial routines to see what is happening under the covers. Sometimes it's not pretty looking under the covers, but you need to know what's going on.

In the example source, the first step is to press <s> and <enter> to start the WRL-3000 module.

Then, if you have compiled it with the correct network, network type and password in the main program file, you can use <c> and <enter> to connect to the network.

Alternatively, if you have the iPhone CC3000 SmartConfig app (search the apple store), you can use it to configure the module automatically. Enter your network password in the app, and use <!> and <enter> to call the smart config routines.

Note that once you have completed smart config, the profile will be stored in the profile settings for the WRL-3000, and the startup options changed. That means that the next time the WRL-3000 starts up (<s> command above) it will automatically reconnect to the network you set up.

If you don't want that, you can use the <n> and <enter> to disable the connection policy options.

Versions

Doc Version	HW Version	Date	Comments
1	2	31 July 2013	Initial Version for first run boards