

# Fluke FBLE Radio Module

## Developers Guide

Fluke Mfg Co.

Oct 8, 2013

V1.0

# Table of Contents

Table of Contents .....	2
Introduction .....	3
Comprehensive Integration Instructions for Fluke FBLE radio module.....	3
Interface to external Microprocessor .....	3
Physical Interface .....	3
Software Interface .....	3
Network Command Communication .....	4
Device Command Communication .....	4
Device Notification Communication .....	5
SPI Data Packets.....	5
SPI Packet Formats.....	6
SPI Packet Field Definitions.....	7
SPI Packets are Big-Endian .....	8
Network Control Commands .....	8
Network Control Commands .....	10
Device Control Command Payloads.....	11
Device Control Command Responses .....	12
Interrupt Status Word.....	14
Network Control Commands and Responses are Big-Endian .....	15
BLE Custom UUID.....	15
1. Introduction .....	15
Custom UUID Ranges .....	15
Firmware Upgrade over BLE .....	22
Required Items.....	22
Upgrade Process .....	23
Reference material.....	24
External processor upgrade over BLE .....	24
Overview .....	24
Required Items.....	25
Upgrade Process .....	25
Reference material.....	26

## Introduction

This document is intended to allow developers to work with the CC2540 based BLE Slave Module.

## Comprehensive Integration Instructions for Fluke FBLE radio module

### LABELING REQUIREMENTS FOR END-PRODUCT

The Original Equipment Manufacturer (OEM) must ensure that FCC labeling requirements are met. The FBLE module is labeled with its own FCC ID and IC Certification Number. The FCC ID and IC certification numbers are not visible when the module is installed inside another device. The final end product must be labeled in a visible area with the following:

“Contains Transmitter Module FCC ID: T68-FBLE”

“Contains Transmitter Module IC: 6627A-FBLE”



The Fluke FBLE Module has been certified with its own on-board antenna and must not be used with other antenna(s).

## Interface to external Microprocessor

### Physical Interface

The CC254x based Slave module uses a modified SPI interface to communicate as a SPI Slave device. The Interface needs to be configured as described below.

- The **SPI** interface is implemented in the 4-wire configuration consisting of signals RFEN (i.e., chip select line), SIMO, SOMI and SCK.
- The BUSY line must go low before the master begins to clock data. The BUSY line must be checked between each byte transaction as well.
- A Interrupt line is provided to allow the radio to notify the SPI master when there is data pending. The line is level triggered. When the line is high, the Interrupt Status word should be read. When the line is low, no data is pending.
- **SPI** port speed is controlled by the master or slave application up to a maximum speed of 4 MHz.
- Positive clock polarity, non-delayed clock phase, MSB sent first.

### Software Interface

The BLE Radio uses a propriety packet format. This format allows the radio to be configured via network control commands, and allows data to be passed via device control commands. In order to

support a number of BLE based service, the host processor can also generate a number of event items in a response style pack as well.

### Network Command Communication

Unless otherwise noted in the detailed command description, all communications use the following process. The following abbreviations are used in the description:

- MA: Master Application
- MRA: Master Radio Application

Step #	Master Application	Master Radio
1	MA formats a command	
2	MA clocks the command over the SPI to the MRA	
3		MRA sets Busy high
4		MRA process the command
7		MRA formats a command response to be transmitted over the SPI to the MR
8		MRA sets Busy low
9	MA clocks the response over the SPI from the MRA	
10	MA processes the response	

### Device Command Communication

In the BLE radio setup, the device commands are used to get legacy data out of the host processor into the BLE radio for use by the TI BLE Stack.

Unless otherwise noted in the detailed command description, all communications use the following process. The following abbreviations are used in the description:

- MA: Master Application
- MRA: Master Radio Application

Master Application	Master Radio
	Command received from BLE, or timer expires. Command is generated to be sent to MA. Once command is ready, INT line is set high.

MA clocks out new command from MRA	
MA processes command from MRA	
MA Clocks data back to MRA once BUSY line is low.	
	Return data is processed. Data is either stored in BLE stack for BLE Master, or sent back to BLE master via Notification command.

### Device Notification Communication

The Notification style of data consists of a number of small data packets of variable length. This format allows common data that is grouped together to be sent over to the BLE radio as it is available.

Unless otherwise noted in the detailed command description, all communications use the following process. The following abbreviations are used in the description:

- MA: Master Application
- MRA: Master Radio Application

Master Application	Master Radio
Timer expires, or local data is updated, triggering a notification event. New Notification packet is created.	
MA clocks out new command from MRA	
MA processes command from MRA	
MA Clocks data back to MRA once BUSY line is low.	
	Return data is processed. Data is either stored in BLE stack for BLE Master, or sent back to BLE master via Notification command.
ACK sent back from MRA	

### SPI Data Packets

#### SPI Packet Types and Enumeration

Packet Type	Description	Enumeration	Payload
Command ACK (C_ACK)	Response from the radio to the application upon receipt of a Control Command.	0x01	None
Application ACK (A_ACK)	Response from a slave to the master upon receipt of a Device Control Command which requires no data returned.	0x02	None
Network Control	Network Control Command.	0x03	See NetworkControlCommands
Device Control	Device Control Command.	0x04	See DeviceControlCommands
Single Data Transfer	Data sent from a slave to a master.	0x06	
Command Data Transfer	Data returned after reading the Interrupt status word or a Network Control command data request.	0x07	

### SPI Packet Formats

C_ACK Packet					
Packet Field	Length	Fluke Preamble	Packet Type	C_ACK Error	CRC
Size in Bytes	1	4	1	1	2

A_ACK Packet					
Packet Field	Length	Packet Type	RF Signal Strength	A_ACK error	CRC
Size in Bytes	1	1	1	1	2

Network Control Packet					
Packet Field	Length	Packet Type	Command Enumeration	Payload	CRC
Size in Bytes	1	1	1	0-89	2

Device Control Packet					
-----------------------	--	--	--	--	--

Packet Field	Length	Packet Type	Slave Device Number	Time to Next Transmission	Device Command Enumeration	Payload	CR C
Size in Bytes	1	1	1	1	1	0-87	2


#### Single Data Transfer Packet

Packet Field	Length	Packet Type	RF Signal Strength	Payload	CR C
Size in Bytes	1	1	1	0-89	2

#### Command Data Transfer Packet

Packet Field	Length	Packet Type	Payload	CRC
Size in Bytes	1	1	0-90	2

### SPI Packet Field Definitions

- Length
  - Number of bytes following the length field.
- Packet Type
  -  See SPI Packet Types
- RF Signal Strength
  - A number indicating the RF signal strength measured at the radio. The BLE radio adds these values in, all the application has to do is leave a blank byte to be filled.
- Slave Device Number
  - The numerical identity of a slave as determined by the discovery process.
- Time to Next Transmission
  - The time until the next transmission from the master to the slave. This information is used by the slave radio module to schedule when its radio needs to be in receive mode. The minimum time allowed is 160ms, the maximum is 5.120 seconds and the resolution is 120ms. To calculate the value to be entered into this field, N, from a desired time to next transmission, T (in seconds), use one of the two formulas.
    - $N = \text{integer}(T / 160)$

For example, for a time to next transmission of 480ms,  $N = 3$ .

- Device Command Enumeration
  - A number identifying the Device Control Command being sent. See BLE Device Control Commands
- Payload
  - Any data being transmitted.

- C\_ACK or A\_ACK Error
  - 0 = No error, 1 = Invalid Command, 2 = Invalid Device Number, 3 = CRC error.
- CRC
  - a 16-bit CRC. The CRC is calculated using a initial value of 0xffff, and a poly value of 0x8408. This is based on the CRC-16-CCITT algorithm.

### SPI Packets are Big-Endian


Unless specified otherwise in this documentation, multi-byte numeric fields in BLE SPI packets are big-endian.

### Network Control Commands

Command	Description	Enumeration	Command Payload	Response to Command	Notes
Set Channel	Use specified channel for all following transmissions. Intended for use in certification.	0x01	Channel #, 1 byte unsigned value between 11 and 26	C_ACK	Used by FCC Testing and Developer packet sniffing Only.
Set Power	Use specified power for all following communication.	0x02	Power - see 📶 PowerLevels	C_ACK	Used by FCC Testing Only.
Turn On Transmitter	Transmit continuously. Used for certification of the radio module.	0x03	None	C_ACK	Used by FCC Testing Only.

Get Interrupt Status	Get the status word from the radio module which indicates the cause of the interrupt.	0x05	None	Command Data Packet with the Interrupt Status word	Used by low level interrupt service code.
Power On	Power up the radio in the specified mode.	0x06	Mode (0 = slave, 1 = master)	C_ACK	



	Also perform a checksum test.				
Power Off	Power off the radio.	0x07	None	C_ACK	
Get Data	Get a information from the radio based on the interrupt.	0x08	Interrupt Status received from a Get Interrupt Status command.	Depends on interrupt status  (See <a href="#">Network Control Command Interrupts</a> )	Used by low level interrupt service code.
Get Version	Get the software version from the radio module.	0x09	None	Radio software version in a Command Data Transfer Packet. The version number is encoded in a ASCII string, the format of the version number will be 'BLE xx.yy.zzbd' xx = BLE APP Major version, yy = BLE APP Minor Version zz = BLE APP sub-Minor Version b = 'b' for beta (e.g non release), ' ' for release, 'd' = Development version	
Flash Erase	Erase the flash.	0x0f	Slave device number, if 0, the local radio.	C_ACK	Firmware update only command
Flash Write	Write a new program fragment to flash.	0x10	Slave device number, if 0, the local radio.	C_ACK	Firmware update only command
Flash Verify	Verify the newly loaded code.	0x11	Slave device number, if 0, the local radio.	C_ACK	Firmware update only command
Flash Swap	Send C_ACK, swap to the newly loaded code and reset.	0x12	Slave device number, if 0, the local radio.	C_ACK	Firmware update only command

When the radio module needs to communicate with the application it requests service via the IRQ interrupt line. In response to this interrupt, the application will send the Get Interrupt Status command to the radio module. The radio will respond with a two byte interrupt status word detailed below. Depending on the interrupt cause, the application may then send the Get Data network control command, with the interrupt status word as a payload, to get further information from the radio. Only one bit in the payload should be set. If more than one bit is set, only the least significant set bit will be processed. The interrupt status word is cleared after the Get Interrupt Status command has been serviced.

### Network Control Commands

Command	Description	Enumeration	Command Payload	Response to Command
Query Device Info	Get device ID string.	0x01	None	C_ACK or A_ACK or Device ID string contained in a Single Transfer Data Packet
Query user String	Get user programmed string.	0x04	None	C_ACK or A_ACK or user string contained in a Single Transfer Data Packet
Set user String	Set device user string.	0x05	User String	C_ACK or A_ACK
Activate Locator	Activate the visual location indicator.	0x09	Locator	C_ACK or A_ACK
Query measurement	Get measurement data.	0x0a	None	C_ACK or A_ACK or Measurement value contained in a Single Transfer Data Packet
Clear Stored Data	Clear logged data.	0x0b	None	C_ACK or A_ACK
Get System Status	Get a data containing status for Logging, power and firmware	0x0c	None	C_ACK or A_ACK or System Status data in in a Single Transfer Data Packet
Set logging mode	Start or stop logging	0x0d	Logging Mode	C_ACK or A_ACK
Set Time	Set the real time clock.	0x10	Time	C_ACK or A_ACK

Query Time	Get the real time clock value.	0x11	None	C_ACK or A_ACK or Time value contained in a Single Transfer Data Packet
Erase Program Memory	Erase the memory used to store the new program.	0x13	None	C_ACK or A_ACK
Store Program Fragment	Store a fragment of the new program.	0x14	Code Fragment	C_ACK or A_ACK
Verify Program	Verify the stored new program, see system data to get status	0x15	None	C_ACK or A_ACK
Load Program	Load the new program and reset.	0x16	None	C_ACK or A_ACK
Execute Command	This is a pass thru function telling the remote application to execute the command in the payload.	0x17	Command to be executed.	C_ACK or A_ACK or Response in a Single Transfer Data Packet
Configure Logging	Set the logging interval and duration in seconds.	0x018	Logging Configuration data	C_ACK or A_ACK

## Device Control Command Payloads

- User String
  - A User defined string of up to 86 bytes in UTF8 format (86 - 21 characters depending on the characters).
  - The string should not be NULL-terminated. The payload length is used to determine the length of the string.
- Locator
  - 1 byte value, unsigned integer, 0: Deactivate Locator, 1: Start Locator . Note that this behavior is device family specific. A Locator beacon may turn off automatically after several seconds.
- Logging Mode

- 1 byte value, unsigned integer, 0: Stop Logging, 1: start logging.
- Logging Configuration data
  -

Interval	Duration
4 byte unsigned integer, log interval in seconds	4 byte unsigned integer, log duration in seconds

- Time
  - A time value used to set a real time clock, 64 bit signed int, POSIX format.
- Code Fragment
  - A fragment of the new code intended to replace the software in the remote device. A code fragment is defined to be one line of an Intel format hex file minus the beginning colon and the two byte checksum at the end of the line. The maximum fragment length is 40 bytes. All hex file lines are considered code fragments independent of the record type and should be sent to the remote device.

### Device Control Command Responses

- Device Info
  - A comma separated string containing the manufacture name and model, firmware rev, and serial number. For example: `FLUKE 289,V1.01,78080001` .
- User String
  - A User defined string of up to 86 bytes in UTF8 format (86 - 21 characters depending on the characters).
  - The string is not NULL-terminated. Use the payload length to determine the length of the string.
- Measurement
  - A multi-byte value containing the measured value and any annunciators required. The most significant byte defines the format of the measurement data and the following bytes contain the measurement data.

Format byte	Measurement Data Description						
0 = Meter style	Field	reading	unit multiplier	unit	ac or	bolt	inrush

measurement data. All data is ASCII characters.					d		
	<b>Size in bytes</b>	6	1	4	2	1	2
	<b>Description</b>	Allows for 4 digits, a decimal and a minus. Right justified .	n, u, k, m, M, or space for no multiplier .	V, A, OHMS, H, VHZ, F, DEGC, DEGF, R Left justified .	ac or dc	* if the bolt is on or space if not.	The word 'in' or two spaces .
1 = TBD							

- System Status data
  - A multi-byte value containing the Logging state and settings, the system power information, and the firmware information.

Format byte	System Status Data Description								
	0 = Argenta System data	<b>Field</b>	Power Battery level	Power State	Firmware State	Logging State	Log - Total Bytes	Log - Used Bytes	Log - Interval
All data is Binary	<b>Size in bytes</b>	1	1	1	1	4	4	4	4
	<b>Description</b>	Unsigned Int - Battery percent	Battery State Enum: 0:Good,1:Low,2:Locked down,3:No Battery ,4: External	Firmware Enum: 0:Idle, 1:Erasing Flash, 2:Progr	Logging state enum : 0:Idle ,	unsigned integer, total num	unsigned integer, used byte	unsigned integer, logging	unsigned integer, logging

		from 0 to 100	Power, 5:Charging	aming Flash, 3:Verif y in progres s, 4:Verif y passed, 5:Verif y failed	1:Lo gging , 2:Not Supp orted	ber of byte s for logg ing in byte s.	s for logg ing in byte s	inter val in seco nds	dura tion in seco nds
1 = TB D									

- Time
  - A time value from the real time clock, 64 bit signed int POSIX format.

#### Interrupt Status Word

Command Bit (0 = LSB)	Description	Response to Get Data command with this payload
0	Discovery complete	Command Data Packet containing the number of slaves discovered.
1	Packet Received	Single Transfer Data Packet, A_ACK or C_ACK packet received from slave
2	Loss of Communication (Used to tell a master it has lost a particular slave.)	Command Data Transfer packet containing the slave device number which has been lost.
3	Bound State Initiated (Used to tell a slave application it has become bound.)	None
4	Bound State Terminated (Used to tell a slave application it has become unbound.)	None
5	Checksum of radio software failed	None
6	Talk to Slave	Command Data Packet with the slave

		number who is ready to be talked with.
7-15	Reserved	

## Network Control Commands and Responses are Big-Endian

Unless specified otherwise in this documentation, multi-byte numeric fields in Network Control Commands, their payloads and their responses are

## BLE Custom UUID

### 1. Introduction

In order to create non-standard Bluetooth Low Energy field, we needed to create a custom set of UUID values. The 'base' 128-bit UUID that we are using for all of these services is B698XXXX-7562-11E2-B50D-00163E46F8FE, where the XXXX will be replaced with the number listed below.

For now, our 16-bit ranges are following the range rules laid out for the BLE standard range.

- 0x1800 - 0x26FF : Service UUID
- 0x2700 - 0x27FF : Units
- 0x2800 - 0x28FF : Attribute Types

These are pulled from "Bluetooth Low Energy The Developer's Handbook" by Robin Heydon, 1st publishing, page 191 (Section 10.2.3). You'll note that we didn't do a range for the Characteristic Descriptors or the Characteristic Types, that is because these do not need to be customized.

## Custom UUID Ranges

### Services

#### 2.2. 0x1800 : FWCS Display reading

- CC254x Module: Active as of 00.00.03
- FWCS Display reading service. Has a 16 Byte ASCII string representing the display value.

#### 2.3. 0x1801 : CNX Services

- CC254x Module: Active as of 00.00.05
- CNX Services (Module ID, Name settings?, Serial pass-thru?, etc..)

#### *2.4. 0x1802 : IG Log Data Services*

- CC254x Module: Active as of 00.00.06
- IG Log Data Services (Get Logging status, Setup Logging parameters)

#### *2.5. 0x1803 : IG Debug Services*

- CC254x Module: Active as of 00.00.07
- IG Debug Services (Send / RX data - additional debug items as needed -- MSP430 asserts?)

#### *2.6. 0x1804 : TI OAD Service*

- CC254x Module: Active as of 00.00.15
- TI OAD Service (Firmware Update for the radio), moved to new number so TI devices don't try to update our devices.

### *Characteristics*

#### *3.1. 0x2900 : SPI Transaction Count*

- CC254x Module: Deprecated in 00.00.03
- SPI Transaction Count. uint8, rolling counter of SPI transaction.

#### *3.2. 0x2901 : Module Display value*

- CC254x Module: Active as of 00.00.03
- Module Display value, 16-byte ASCII string. See FWCS documentation for decoding.

#### *3.3. 0x2902 : Model ID Number*

- CC254x Module: Active as of 00.00.03
- Model ID Number. uint8, read/write, displayed ID Number on slave unit UI.

#### *3.4. 0x2903 : Module User String*

- CC254x Module: Active as of 00.00.05
- Module User String, UTF-8 data, up to 98 bytes. This is the same as modifying the <<Device Name>> BLE Characteristic. However, this additional item is created in order to allow Apple products to modify this (they lockout modifying <<Device Name>> directly).

#### *3.5. 0x2904 : Force Drop*

- CC254x Module: Active as of 00.00.05



- Force Drop - Writing anything to this item causes the BLE module to drop. This is a work around for Apple related issues, per Brady H.

### 3.6. 0x2905 : Set Locator State

- CC254x Module: Active as of 00.00.05
- Set Locator State - write only uint8, 0 to turn off locator, 1 to turn on locator.

### 3.7. 0x2906 : IG Logging Status

- CC254x Module: Active as of 00.00.06
- IG Logging Status - read only uint8 array, 9 bytes.
  - 
  -

Logging Status Data Description			
<b>Field</b>	Logging State	Log - Total Bytes	Log - Used Bytes
<b>Size in bytes</b>	1	4	4
<b>Description</b>	Logging state enum: 0:Idle, 1:Logging, 2:Not Supported	unsigned integer, total number of bytes for logging in bytes.	unsigned integer, used bytes for logging in bytes

### 3.8. 0x2907 : IG Logging Settings

- CC254x Module: Active as of 00.00.06
- IG Logging Settings - read / write uint8 array, 9 bytes.
  - 
  -

Logging Settings Data Description		
<b>Field</b>	Log - Interval	Log - Duration
<b>Size in bytes</b>	4	4
<b>Description</b>	unsigned integer, logging interval in seconds	unsigned integer, logging duration in seconds

### 3.9. 0x2908 : IG Logging Control point

- CC254x Module: Active as of 00.00.07
- IG Logging Control point.

- Write a Enum to to this point ask the logging state machine to change. Changes can be seen via notification of the logging Status characteristic.
  - 
  -

Enum Value	Meaning
0x00	Stop logging
0x01	Start logging
0x02	Erase logged Data

### 3.10. 0x2909 : IG Debugging - Read

- CC254x Module: Active as of 00.00.07
- IG Debugging - Read point. ASCII data, up to 87 bytes (see FWCS DCC payload size -- this is matched to that.). No NULL needed.

### 3.11. 0x290A : IG Debugging - Write

- CC254x Module: Active as of 00.00.07
- IG Debugging - Write point. ASCII data, up to 89 bytes (see FWCS Single Data payload size -- this is matched to that.). No NULL included.

### 3.12. 0x290B : Sync / Pass-Thru calls

- CC254x Module: Active as of 00.00.07
- IG Debugging - Number of time a sync / pass thru was called. Temporary, used to determine how TI stack behaved with blog write.

### 3.13. 0x290C : BLE Disconnect Code

- CC254x Module: Active as of 00.00.07
- IG Debugging - Code from the last BLE disconnect. values are from ll.h, (The status values map directly to the HCI Error Codes. Per the Bluetooth Core Specification, V4.0.0, Vol. 2, Part D.)


### 3.14. 0x290D : Logging Memory Size

- CC254x Module: Active as of 00.00.12
- LOGGING\_MEMSIZE - 4 byte unsigned values representing the size in bytes of the logging memory.

### 3.15. 0x290E : POSIX Time

- CC254x Module: Active as of 00.00.12
- POSIX\_TIME - Read / Nofity back point. Used to set or get the device time in POSIX format. See FWCS documentation for more info on how we use this value, 8 byte unsigned integer value.

### 3.16. 0x290F : Binary Reading

- CC254x Module: Active as of 00.00.12
- READING - Binary format for passing a display reading, see  [wiki:Elektra:TechnicalInvestigation/BleServicesMapping/service\\_flukeReading](http://wiki.Elektra:TechnicalInvestigation/BleServicesMapping/service_flukeReading) for more information, and formatting details (Still under development at this point -- [ThomasAnderson](#) 2013-06-25)

### 3.17. 0x2910 : Reading Nomenclature

- CC254x Module: Active as of 00.00.12
- READING\_NOMENCLATURE - ?? Marked, but not yet implimented, need to talk to John L. -- [ThomasAnderson2013](#)-06-25

### 3.18. 0x2911 : CNX FW Update Control Point

- CC254x Module: Active as of 00.00.14
- FW\_UPDATE\_CP - CNX Style MSP430 Firmware Update Control point. Write in "Commands", and it sets the item back to a state. used to update firmware in MSP430.
  -

Value	State
0x00	CNX_FW_CP_STATE_IDLE
0x01	CNX_FW_CP_STATE_ERASING
0x02	CNX_FW_CP_STATE_ERASED
0x03	CNX_FW_CP_STATE_BUFFER_CLEARED
0x04	CNX_FW_CP_STATE_BUFFER_WRITING
0x05	CNX_FW_CP_STATE_VERIFYING
0x06	CNX_FW_CP_STATE_VERIFY_PASS
0x07	CNX_FW_CP_STATE_VERIFY_FAIL
0x08	CNX_FW_CP_STATE_SWAPPING

0x09	CNX_FW_CP_STATE_ERROR	
0x0a	CNX_FW_CP_STATE_BUFFER_FILLING	
0x0b	CNX_FW_CP_STATE_SWAP_FAIL	
0x0c	CNX_FW_CP_STATE_RADIO_INIT_QUERY	
0x0d	CNX_FW_CP_STATE_RADIO_INIT_OFF	
0x0e	CNX_FW_CP_STATE_RADIO_INIT_ON	
0x0f	CNX_FW_CP_STATE_RADIO_INIT_CHANGING	
Value	Command	Note
0x80	CNX_FW_CP_CMD_ERASE_MEM	Erase the Flash Memory on the MSP430
0x81	CNX_FW_CP_CMD_CLEAR_BUF	Clear the FW_UPDATE_BUF in the radio
0x82	CNX_FW_CP_CMD_WRITE_AND_CLEAR_BUF	Write the contents of the FW_UPDATE_BUF to the MSP430, then clear the FW_UPDATE_BUF
0x83	CNX_FW_CP_CMD_DO_VERIFY	Cause the MSP430 to verify the local copy of the new firmware
0x84	CNX_FW_CP_CMD_DO_SWAP	Load the new Firmware into the MSP430 (requires a verify to pass first)
0x85	CNX_FW_CP_CMD_RADIO_INIT_ON	Change device settings to turn on the radio at power up.
0x86	CNX_FW_CP_CMD_RADIO_INIT_OFF	Change device settings to keep the radio off at power up.

0x87	CNX_FW_CP_CMD_RADIO_INIT_CHECK	Query the initial radio state setting.
------	--------------------------------	--

### 3.19. 0x2912 : CNX FW Update Buffer

- CC254x Module: Active as of 00.00.14
- FW\_UPDATE\_BUF - CNX Style MSP430 Firmware Update Buffer. Used to allow the master to write up a buffer in chunks, and then write that using the Control point. This is done so we can use "write no response" in order to maximize the write speed. Clearing this buffer, and writing this buffer to the MSP430 is done via the FW\_UPDATE\_CP characteristic.

### 3.20. 0x2913 : TI OAD Image Identify

- CC254x Module: Active as of 00.00.15
- TI OAD, Image Identify Characteristics. Works the same as TI, only moved here so other devices don't try to update us.

### 3.21. 0x2914 : TI OAD Image Block

- CC254x Module: Active as of 00.00.15
- TI OAD, Image Block Characteristic. Works the same as TI, only moved here so other devices don't try to update us.

### 3.22. 0x2915 : User String Buffer

- F3000 BLE DMM: Active as of 00.00.08
- Module User String buffer, UTF-8 data, up to 98 bytes. This is the same as modifying the <<Device Name>> BLE Characteristic. However, this additional item is created in order to allow Apple products to modify this (they lockout modifying <<Device Name>> directly).

### 3.23. 0x2916 : User String Control Point

- F3000 BLE DMM: Active as of 00.00.08
- Module User String Control Point, allows for clearing, or writing data in the user string buffer. See 0x2915.

### 3.24. 0x2917 : Bulk Data Download Interface

- CC254x Module: Active as of 00.00.xx (expcted is 00.00.18 )
- Control point for doing Bulk Data download. Master writes in a block number, and get back a bunch of notify packets with data.
  - Write in :

Cmd	State
00	Return total Number of Bytes
01<state[1 byte unsigned int]>	Lock the unit for bulk download, or unlock it. <state> should be 0x00 for unlock, and 0x01 for lock
02<startByte[4 byte, unsigned int]><endByte[4 byte, unsigned int]>	System will notify back data in MTU packet sized from start byte to end byte.
03	Cancel download. If data is coming in and the master wants to stop the transfer, send this command

- Notify Back: Bytes, in 1 to 18 bytes notify Chunks.

## Firmware Upgrade over BLE

- The TI CC254x style upgrade is based on the idea of three sections in code. First off is a Boot Manager, which decided which image to run. This manager is never updated. The other two sections are labeled Image A, and Image B. On Power up the Boot manger decides which image to run. When it is time to update the device, the code in running image allows for the other image to be erased (in other words, if Image A is running, then you can only update Image B). Once Image B is updated and verified as good, Image A will be tweaked so it no longer passes the CRC check. Now the boot manger will run the newly updated Image B. On the next update, Image A will be erased, updated, and once verified.. run.

## Required Items

- A Over the Air Upgrade File.
  - A Image A and Image B File, can be compiled with scons via "scons rf\_module\_imgA" or "scons rf\_module\_imgB"
- System Hardware
  - A CNX Style slave unit, which can put the radio into a discoverable state.
  - A BLE Master device, capable to driving the process outlined below.
- Firmware
  - RF Module with boot loader code present.
  - RF Module BLE Services.
    - BLE Device Information Service ( UUID16 0x180A ).
      - Firmware Characteristic (Which Identifies the running RF Module Code).
    - TI OAD ( UUID128 f000180304514000b000000000000000 )
      - TI - OAD Img Id Characteristic (UUID128 f000ffc104514000b000000000000000 )

- TI - OAD Img Block Characteristic (UUID128 f000ffc204514000b000000000000000 )

## Upgrade Process

1. Establish Connection between Slave and master.
2. Verify and configure connection.
  1. Ensure connection rate is as fast as the master and slave will allow for.
  2. Ensure that the slave has all the required services and characteristics.
    1. Enable **Notify** on the <<TI - OAD Img Id>> characteristic.
    2. Enable Notification on the <<OAD Img Block>> characteristic.
  3. Check the running version in order to see which image to download.
    1. Write 0x00 to <<TI - OAD Img Id>>. If we get "0x00" back, then Write 0x01 to <<TI - OAD Img Id>>.
    2. <<TI - OAD Img Id>> Now has the block identification information in it.

<b>Size</b>	4 Bytes	2 Bytes	2 Bytes
<b>Name</b>	Image ID	Image Size	Version

- Image ID is 4 copies of either a ASCII 'A' (for Image A), or ASCII 'B' (for Image B). This is the currently running image on the RF Module.
  - Size is the size in bytes divided by 4 for the running image.
  - Version is the image version.
  - Example: "424242427C000001", Image is A (0x42,0x42,0x42,0x42), Size is 0x2700 \* 4 = 126976 Bytes, ID = 0x0001
- If Image A is running, then we can update Image B. If Image B is running, we can update Image A.
3. Start the download.
    1. Get the OAD Summary line (see the line read above from <<TI - OAD Img Id>>) for the new Image from the raw binary file, and send this to the <<TI - OAD Img Id>> characteristic.
      - Open the Raw Binary file, skip the first 4 bytes, and then read out the next 8 bytes. These 8 bytes are the Image Block Identification for this file.
      - Doing this write starts the upgrade process on the Radio, when the radio is read to get data it will **notify** to <<OAD Img Block>>
  4. Write each block to the slave.
    1. Take the entire Binary file (including the skipped first 4 bytes, and 8 bytes of Image Block Identification), and break it into 16 byte blocks.
    2. Number each block sequentially, starting at 0x0000.
    3. When the Slave notifies us with a 16-bit number, that is the ID of the next block that it wants. (so when we get 0x0000, we should send block 0x0000).

- Multiple blocks can be written at once using the BLE "Write, No Response" command. For TI's BLE Stack (V1.3.x), we can write up to 4 blocks at one time.
4. Continue this process until all blocks are written. If a block isn't delivered (due to being sent with write, no response), the Radio will re-request it when the next block is written.
  5. Loop though this process until all the blocks are written.
5. Wait for the disconnect.
    1. After the last block is written, wait for the BLE Slave device to disconnect, the radio is settings up to use the new firmware image.
  6. Re-connect, and verify that the new image loaded by writing a 0x00 or 0x01 to <<TI - OAD Img Id>> (Don't forget to enable Notifications before doing the write).

## Reference material

- Implementation notes at  [wiki:Self:TechnicalInvestigation/BleServicesMapping/service\\_fwDownload](https://wiki.ti.com/TechnicalInvestigation/BleServicesMapping/service_fwDownload)
- TI OAD Documentation, located at <http://processors.wiki.ti.com/index.php/OAD>
- pyFwUpdate.py script in `tools/pyBleDongle/pyFwUpdate.py` of the Elektra cc2540 GIT repository. (Uses TI CC2540 Development USB Dongle to do a FW upgrade).

## External processor upgrade over BLE

This page outlines the process for doing a Firmware update to a CNX style MSP430 based device over the air, using BLE.

Note that this process is heavily based on the CNX Firmware update process.

*Note:* For a example implementation, see the `pyFwUpdate.py` script in `tools/pyBleDongle/pyFwUpdate.py` of the Elektra cc2540 GIT repository.

## Overview

- The basic sequence of events is a duplicate of the CNX Firmware Update process. However, due to differences between FWCS and BLE, I had to make some minor changes. In the BLE style upgrade, the packet size is much smaller (18 bytes instead of 89 bytes). In order to get around this, the BLE radio provides a 'register', which can be filled with data. This allows us to use the 'write no response' BLE command to send multiple packets in a single connection event. The Radio then caches these items up, until they are the same size as the FWCS packet size. This is only needed for the command uses to write data. The commands to erase the flash, verify the flash, and do the swap remain as a single action.



In this model a 'swap' area is erased, filled with data, and then verified. If it passes the verify, the data is copied back into the proper program space of the MSP430, and the device re-boots.

## Required Items




- A Firmware File
  - a .HEX file, post-processed to ensure longer line length. (Shorter lines work too, but take much longer).
  - File must comply with CNX Verification needs (CRC at minimum, some unit type checking in some modules too.)
- System Hardware
  - A CNX style slave unit, with a BLE radio. Note that some units (Such as the modules) need to have their swapper built and included to support firmware update.
  - A BLE master device, capable of following the process outlined below.
- Firmware
  - a BLE radio which supports CNX style Firmware Updates.
    - BLE Service <<Fluke CNX Services>> (UUID b6981801756211e2b50d00163e46f8fe) containing:
      - <<Fluke - FW Update Control Point>> (UUID b6982911756211e2b50d00163e46f8fe)
      - <<Fluke - FW Update Buffer>> , (UUID b6982912756211e2b50d00163e46f8fe)
  - MSP430 code which supports Firmware Updates.
  - BLE Master capable of binding to the slave, and following the procedure listed below.


## Upgrade Process

1. Establish Connection between Slave and Master.
2. Verify and configure connection.
  1. Ensure connection rate is as fast as the master and slave will allow for.
  2. Ensure that the slave has all the required services and characteristics.
  3. Enable Notifications for <<Fluke - FW Update Control Point>>.
  4. Write "CNX\_FW\_CP\_CMD\_ERASE\_MEM" command to <<Fluke - FW Update Control Point>>
  5. Wait for the state to be notified as "CNX\_FW\_CP\_STATE\_ERASED".
  6. Write "CNX\_FW\_CP\_CMD\_CLEAR\_BUF" command to <<Fluke - FW Update Control Point>>
3. Download new Firmware.
  1. use BLE "Write No Response" to write 18 byte chunks of data to <<Fluke - FW Update Buffer>>, do this until all the data is written out. Note that you may need to limit the write no response to 4 per connection cycle in order to keep from overloading TI's BLE Stack.

- These are the lines from the .hex file, with the leading ':' removed, and the line checksum (last byte) removed, converted into binary.
- 2. Once a complete line is written send the "CNX\_FW\_CP\_CMD\_WRITE\_AND\_CLEAR\_BUF" command to <<Fluke - FW Update Control Point>>
- 3. Wait for the state to be notified as "CNX\_FW\_CP\_STATE\_BUFFER\_CLEARED".
- 4. Repeat these steps until the entire hex file has been written out.
- 4. Verify download / Swap in new Firmware.
  1. Send the "CNX\_FW\_CP\_CMD\_DO\_VERIFY" command to <<Fluke - FW Update Control Point>>.
  2. Wait for "CNX\_FW\_CP\_STATE\_VERIFY\_PASS" or "CNX\_FW\_CP\_STATE\_VERIFY\_FAIL" to notify back.
  3. if "CNX\_FW\_CP\_STATE\_VERIFY\_PASS"
- 1. Send "CNX\_FW\_CP\_CMD\_DO\_SWAP" command to <<Fluke - FW Update Control Point>>
- 2. Wait for master to drop connection.
  4. If "CNX\_FW\_CP\_STATE\_VERIFY\_FAIL"
- 0. Repeat process from the start, re-erasing the flash and sending the data down again.

## Reference material

- <<Fluke - FW Update Control Point>> States :
  - 0x00 : "CNX\_FW\_CP\_STATE\_IDLE"
  - 0x01 : "CNX\_FW\_CP\_STATE\_ERASING"
  - 0x02 : "CNX\_FW\_CP\_STATE\_ERASED"
  - 0x03 : "CNX\_FW\_CP\_STATE\_BUFFER\_CLEARED"
  - 0x04 : "CNX\_FW\_CP\_STATE\_BUFFER\_WRITING"
  - 0x05 : "CNX\_FW\_CP\_STATE\_VERIFYING"
  - 0x06 : "CNX\_FW\_CP\_STATE\_VERIFY\_PASS"
  - 0x07 : "CNX\_FW\_CP\_STATE\_VERIFY\_FAIL"
  - 0x08 : "CNX\_FW\_CP\_STATE\_SWAPPING"
- <<Fluke - FW Update Control Point>> Commands :
  - 0x80 : "CNX\_FW\_CP\_CMD\_ERASE\_MEM"
  - 0x81 : "CNX\_FW\_CP\_CMD\_CLEAR\_BUF"
  - 0x82 : "CNX\_FW\_CP\_CMD\_WRITE\_AND\_CLEAR\_BUF"
  - 0x83 : "CNX\_FW\_CP\_CMD\_DO\_VERIFY"
  - 0x84 : "CNX\_FW\_CP\_CMD\_DO\_SWAP"
- Implementation notes at  [wiki:Self:TechnicalInvestigation/BleServicesMapping/service\\_fwDownload](https://wiki.fluke.com/wiki:Self:TechnicalInvestigation/BleServicesMapping/service_fwDownload)
- Using the dongle to do a Firmware Update  [wiki:Argenta:SoftwareGuide/ScriptedFirmwareUpdate](https://wiki.fluke.com/wiki:Argenta:SoftwareGuide/ScriptedFirmwareUpdate)
- See commands on  [v1.00.00 USB Dongle Commands \(Interactive\)](https://wiki.fluke.com/wiki:Argenta:SoftwareGuide/v1.00.00_USB_Dongle_Commands_(Interactive))
  - Erase Firmware Swap Area
  - Store Firmware Fragment in Swap Area
  - Verify Firmware in Swap Area

- Load New Firmware from Swap Area
- See command on  [FWCS SPI Network Control Commands.](#)
  - Flash Erase
  - Flash Write
  - Flash Verify
  - Flash Swap