CC2650 Java SBL Imp – CC2650.

Author: David Agassi

Date: 12/04/17

Abstract

This project implements a simple Serial BootLoader for TI cc2650 using the SBL backdoor on the chip. It uses the RXTX java (serial and parallel communication interface) to connect to the device through USB on a COM port. The program erases the device, programs it with an input program, verifies the program and resets it.

Index

Background:

CC2650 :

CC2650 LunchPad is a TI (Texas Industries) programmable wireless (Bluetooth) MCU that was used to as the subject of this project. cc2650.pdf

SBL :

Serial BootLoader, an API that allows programming the device through a serial connection. In order to enable the API the device's SBL Backdoor should be enabled. The API is documented in this link: bootloader.pdf

RXTX:

RXTX is a java library that is used to connect serially (or in parallel) to external devices through the computers physical ports (mainly USB Connections). It could be found easily on the internet and is documented and distributed wildly.

Motivation:

We would like to have the ability to program the device directly with a standard interface that does not involve the debugger or the CCS. By enabling this we will be able to convey software updates easily to an already installed chip, using only standard equipment.

Challenges:

- Getting a first response from the SBL API on the chip:
  - The java program was sending establish messages but no response came.
  - Snuffing the port showed the bytes going out, but none returning.
  - The main issue was getting the device into the Bootloader "mode". Eventually with some help, the device was configured to enter that "mode" on reset when the button was clicked.
  - Once that happened the device responded.
- Formatting the CCS output to a programmable bin file:
  - The bits sent from the java program were exactly the file given as input to the device. Reading the device using the Flash Programmer showed the file was identical to the input.
  - Yet the program didn't run.
  - Programming using the same input and the Flash programmer resulted in a correct manner and the device started running.
  - Eventually, the problem was that the input file was a .hex file, which is not the memory bank the chip should hold. There was some parsing that needed to be done on the input.
  - Instead I decided to use .bin files which are the actual memory map that should be copied on to the device. Instruction Included. "Getting a bin file:"

Tools:

CCS:

The IDE provided By TI that is used to develop and test TI's Products.

WireShark + USBCap:

Used to record (USBCap) and display (WireShark) pcap files over the computer network interfaces. For example we use USBCap to record the communication on "COM9" and then analyze it on using WireShark.

Java:

A strong, cross-platform, and popular programming language. The project was written in this language ☺. I used the IntelliJ IDE during the development.

Flash Programmer 2:

A gui program that allows easy access to the device and programming without CCS.

Project's code description:

This section describes the main code entities of the project.

CC2650Bootloader:

The main project class, called by main, it is responsible for all the flow of the program. Only the non-trivial entities are described here.

Instances:

- `private Network network;`
  - An Instance of the Network class responsible to sending and receiving data.
- `private NetworkProxy proxy;`
  - A bridge between the Network and the Bootloader class. Is called to get the device's response.

Functions:

- `public void burn(String com, String path)`
  - Receives a com port and a path to the .bin to program.
  - Connects, erases , programs, verifies, and resets the controller
- `private int sendCmd(ECc2650Commands cmd, byte[] data)`
  - Receives command and appending data and transmits it to the device using the Network instance. Implements the package type descripted in the SBL API document.
  - Returns ACK/NACK/no response.
- `private boolean DeviceStatus()`
  - Reads device Status, and sends computer's ACK in return.
    - After the Read Status, the device expects an Ack from the computer.

  - Returns false if not Ok.

- `private long CalcCRC()`
  - Uses the built in CRC32 to calculate the program's CRC so to compare it to the CRC calculated on the chip.

Network:

An external RXTX example adapted by me to use as the reader and writer of the project. It uses an interface to communicate once input from a device is detected.

Functions:

```
public Network(Network_iface contact)
```
Constructor: receives the Network proxy.

```
public boolean connect(String portName)
```
Connects to given Com port.

```
public boolean writeSerial(int numBytes, byte message[])
```
Writes serially.

```
private class SerialReader implements Runnable
```
A reader that reads continually and writs to the proxy.

Network_iface:

Part of the external RXTX example.

Functions:

```
public void parseInput(int id, int numBytes, byte[] message)
```
Called once the input from the device was received and stored in the buffer.

NetworkProxy:

Implements Network_iface.

Functions:

```
public void parseInput(int id, int numBytes, byte[] message)
```
Copies the message from the buffer to the proxy's buffer. And marks available data to be true.

```
public byte[] awaitResponse()
```
Puts the thread to sleep until response is ready. Once ready, returns the response buffer and sets available message to false.

```
public static ECommandReturn getCmdRetValue
```
Returns the flash command response. If no response returns NULL response. See enum ECommandReturn.

```
public static int AckNack(byte[] bytes)
```
Searches for and ACK in the response and returns it's offset index in the response.

CONSTS Files:

```
class Consts
```

Holds all the Constancies used in the project

```
public enum ECc2650Commands
```

Holds all the commands in the SBL API

```
public enum ECommandReturn
```

Holds all the responses available and a null response.

Comments:

In the code some sections where left in comments in case of debugging or updating the project.

- Printing byte messages: in the write/ read function there are prints that parse the data and writes them to the console.
- In the delete function you could alternate to page deletion instead of deleting the whole device.

User Guide:

Getting a bin file:

In the post build stages in CCS add a step with this parameter:

"${CCS_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin" "${BuildArtifactFileName}" "${BuildArtifactFileBaseName}.bin" "${CG_TOOL_ROOT}/bin/armofd" "${CG_TOOL_ROOT}/bin/armhex" "${CCS_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin"

This will create a .bin file with the projects name in the Debug file of the project.

Finding your COM port:

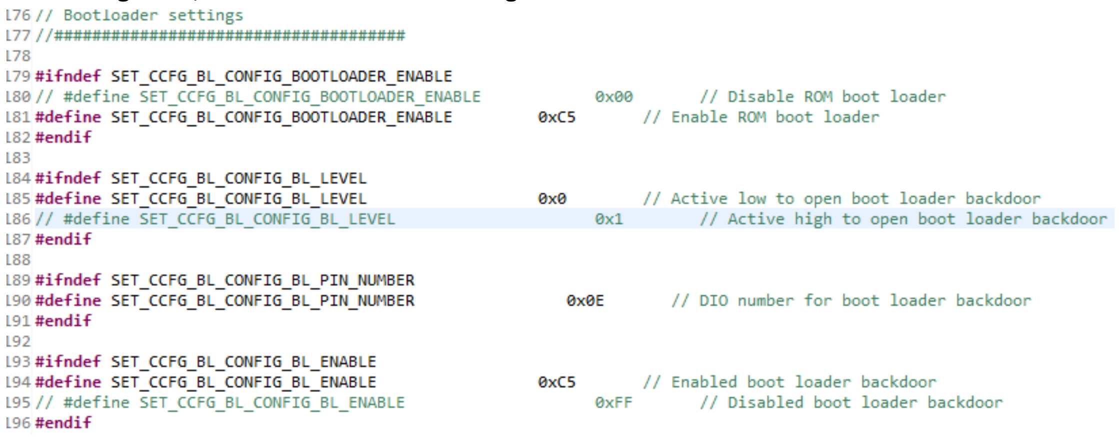
There are many ways of finding the com port to connect to. I used the USBCap menu that iterates through all the com ports and displays them. The Java Network class also has such an option.

Enabling SBL backdoor:

Make sure the program already on the chip is either invalid (no program is running for example) or that the SBL Backdoor was already enabled.

To enable the backdoor you must reprogram your device with a CCFG section enabling it.

In the ccfg.c file, set the Bootloader setting to "enabled".



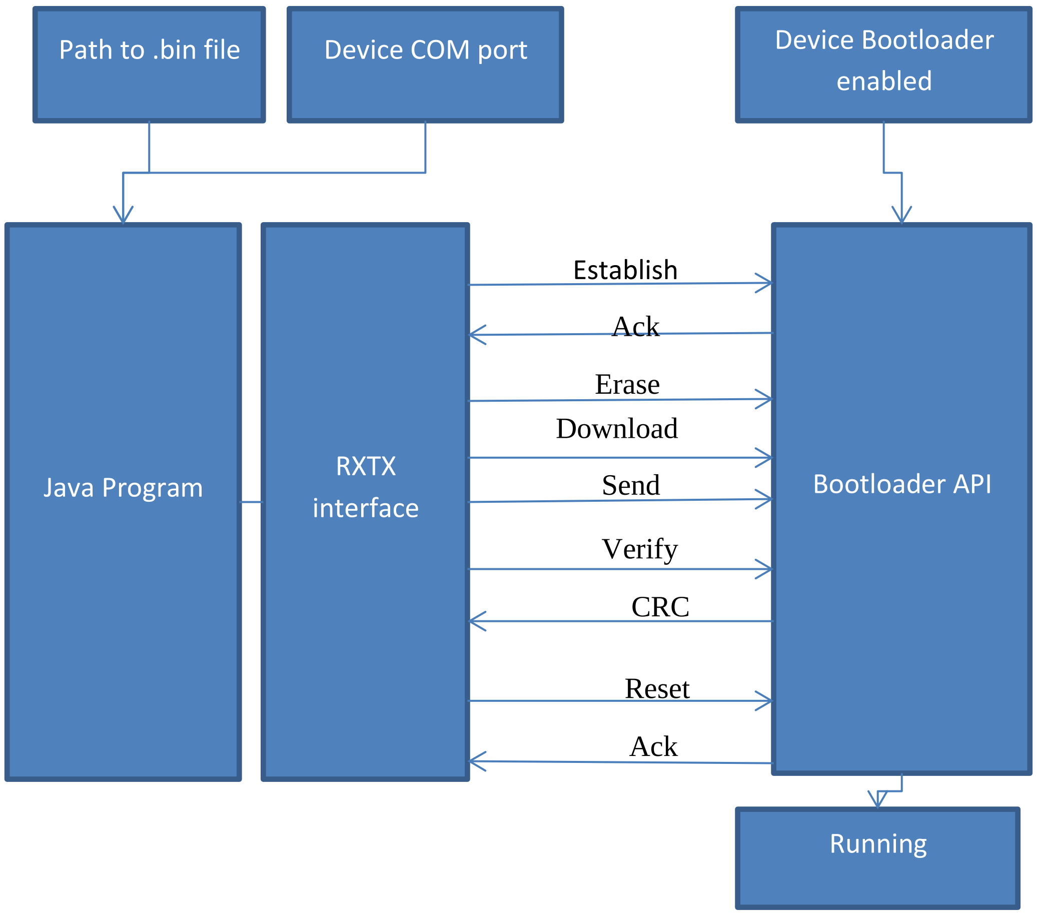
```
// Bootloader settings
//#####################################

#ifndef SET_CCFG_BL_CONFIG_BOOTLOADER_ENABLE
// #define SET_CCFG_BL_CONFIG_BOOTLOADER_ENABLE          0x00       // Disable ROM boot loader
#define SET_CCFG_BL_CONFIG_BOOTLOADER_ENABLE          0xC5       // Enable ROM boot loader
#endif

#ifndef SET_CCFG_BL_CONFIG_BL_LEVEL
#define SET_CCFG_BL_CONFIG_BL_LEVEL                   0x0        // Active low to open boot loader backdoor
// #define SET_CCFG_BL_CONFIG_BL_LEVEL                   0x1        // Active high to open boot loader backdoor
#endif

#ifndef SET_CCFG_BL_CONFIG_BL_PIN_NUMBER
#define SET_CCFG_BL_CONFIG_BL_PIN_NUMBER                0x0E       // DIO number for boot loader backdoor
#endif

#ifndef SET_CCFG_BL_CONFIG_BL_ENABLE
#define SET_CCFG_BL_CONFIG_BL_ENABLE                  0xC5       // Enabled boot loader backdoor
// #define SET_CCFG_BL_CONFIG_BL_ENABLE                  0xFF       // Disabled boot loader backdoor
#endif
```

In this example the Bootloader is to be entered when the left button (which is 0x0E pin) is clicked.

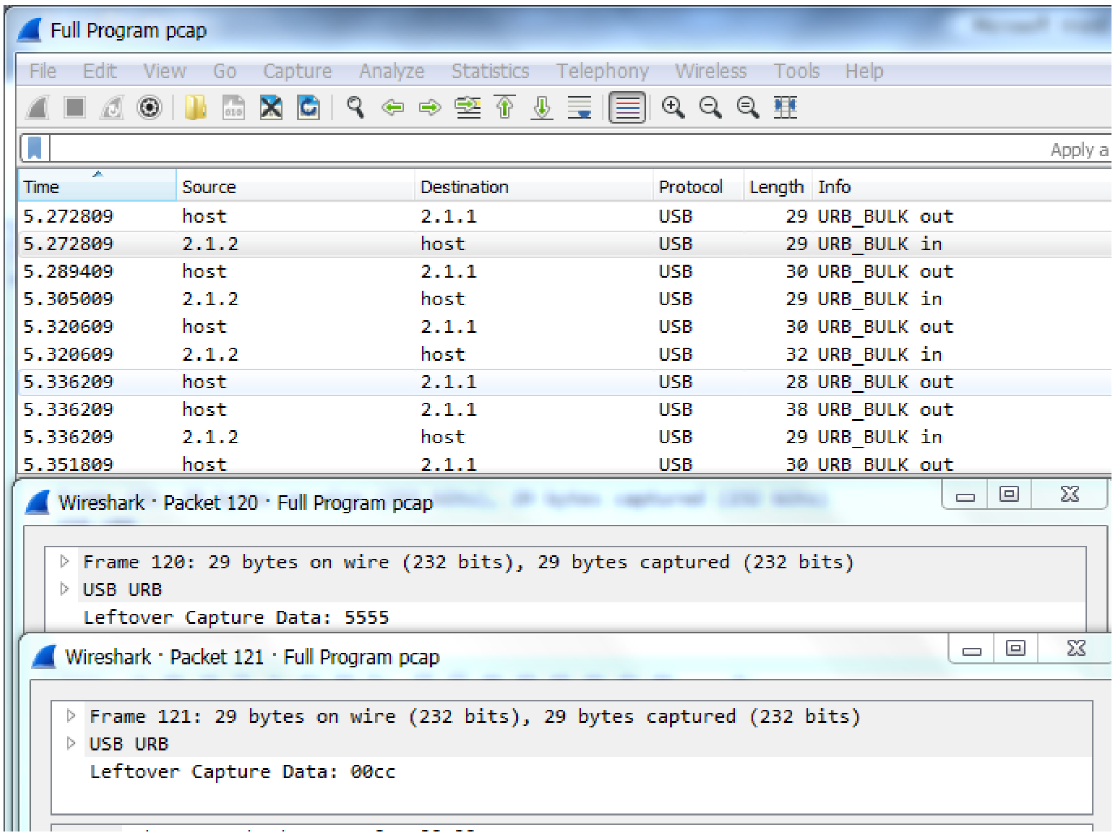
Now once the device is restarts it tests the pin, if it is high, you have entered the Bootloader.

Results:

A program that receives a .bin file and a port and programs a TI CC2650 MCU (!!!)



Here is a full pcap of the communication done by the program. Full Program pcap.


Full Program pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a

```
Time      Source  Destination  Protocol  Length  Info
5.272809  host    2.1.1        USB       29      URB_BULK out
5.272809  2.1.2   host         USB       29      URB_BULK in
5.289409  host    2.1.1        USB       30      URB_BULK out
5.305009  2.1.2   host         USB       29      URB_BULK in
5.320609  host    2.1.1        USB       30      URB_BULK out
5.320609  2.1.2   host         USB       32      URB_BULK in
5.336209  host    2.1.1        USB       28      URB_BULK out
5.336209  host    2.1.1        USB       38      URB_BULK out
5.336209  2.1.2   host         USB       29      URB_BULK in
5.351809  host    2.1.1        USB       30      URB_BULK out
```

Wireshark · Packet 120 · Full Program pcap

```
▷ Frame 120: 29 bytes on wire (232 bits), 29 bytes captured (232 bits)
▷ USB URB
  Leftover Capture Data: 5555
```

Wireshark · Packet 121 · Full Program pcap

```
▷ Frame 121: 29 bytes on wire (232 bits), 29 bytes captured (232 bits)
▷ USB URB
  Leftover Capture Data: 00cc
```

Connection being established, send 0x55,0x55. Received 0xcc.

Source code:

CC2650-SBL-java.rar

Future advancement:

Supporting Hex files: this project accepts full memory maps and downloads it completely to the chip. Instead by supporting .hex files it will allow configuring the device instead of fully deleting it. To do so, first one has to parse the hex file and find the pages to write to. Once that is done, simply change the delete function and write only the pages needed. Make sure to verify every page for itself instead of the whole flash range.