CC23xx ROM BOOTLOADER

(DRAFT)

Basic usage of the ROM Bootloader is nearly identical to that of the previous CC26x2. For a background in how those bootloaders functioned, review chapter 10 (Bootloader) of the CC26x2 Technical Reference Manual

Configuration

Configuration of the ROM Bootloader is done between the FCFG and the CCFG. There are defaults that are set in the FCFG that will take effect if a valid CCFG is not present on startup. If the user wants to alter the defaults set by the FCFG they can update their CCFG to provide the behavior they desire. Both the FCFG and CCFG have the following struct inside providing ROM Bootloader Configuration

bootCfg

BootCfg

```
    // Bootloader/application configuration
    struct {    // [0]: length 16B
        // Pointer to user bootloader vector table
        void* pBldrVtor;
            #define CCFG_BC_PBLDR_USE_FCFG ((void*)0xFFFFFFF0)
            #define XCFG_BC_PBLDR_FORBID   ((void*)0xFFFFFFFC)
            #define XCFG_BC_PBLDR_UNDEF    ((void*)0xFFFFFFFF)
            #define CCFG_BC_PBLDR_VALID(x) ((x) < CCFG_BC_PBLDR_USE_FCFG)
        // Parameter passed to bootloader
        union {
            uint32_t val32;
                #define CCFG_BC_BLDRCFG_UNDEF   0xFFFFFFFF
            // Serial ROM bootloader parameters (also used in FCFG.h)
            struct serialRomBldrParam_struct {
                uint32_t bldrEnabled        : 1;
                    #define XCFG_BC_BLDR_DIS  0
                    #define XCFG_BC_BLDR_EN   1
                uint32_t pinTriggerEnabled  : 1;
                    #define XCFG_BC_PINTRIG_DIS  0
                    #define XCFG_BC_PINTRIG_EN   1
                uint32_t pinTriggerLevel    : 1;
                    #define XCFG_BC_PINTRIG_LEVEL_LO  0
                    #define XCFG_BC_PINTRIG_LEVEL_HI  1
                uint32_t res0 : 13;
                uint32_t pinTriggerDio : 6;
                uint32_t res1 : 2;
                uint32_t serialIoCfgIndex : 3;
                    #define XCFG_BC_IOCFGIND_DEFAULT  0
```

```
            uint32_t res2 : 5;
        };
    } bldrParam;
    // Pointer to application VTOR table
    void* pAppVtor;
        #define CCFG_BC_PAPP_NONE  ((void*)0xFFFFFFFF)
    uint32_t crc32;
} bootCfg;
```

pBldrVtor

This pointer is used to specify if/which bootloader should be used.

- CCFG_BC_PBLDR_USE_FCFG - Indicates to use the bootloader that is specified as a default in the FCFG (ROM Bootloader)
- XCFG_BC_PBLDR_FORBID - Indicates that the user doesn't want a bootloader to be used
- XCFG_BC_PBLDR_UNDEF - Indicates that the user hasn't specified a bootloader
- 0xXXXXXXXX - Can indicate a completely variable pointer to a custom flash bootloader

bldrParam.serialRomBldrParam_struct

This struct contains the majority of the bootloader functional configuration.

- **bldrEnabled** - Indicates if the bootloader should be enabled. If disabled, it's possible that the bootloader still be entered/triggered, though the ROM Bootloader will only respond to BLDR_CMD_GET_STATUS commands. In other words, if disabled, firmware updates are not possible. This may be desired for security reasons.
- **pinTriggerEnabled** - Indicates if the bootloader can be triggered via a GPIO pin. Normally, the bootloader will only be triggered if there is no application. However if you'd like to trigger the ROM Bootloader with a external pin, that behavior can be configured here.
- **pinTriggerLevel** - Only has effect if pinTriggerEnabled is enabled. This value indicates if the bootloader is to be triggered by either a logic LOW or logic HIGH value on the GPIO pin. If the configured pin has the configured value on bootup, the bootloader will be triggered and instead of bootup into the application, the bootcode will remain in the bootloader.
- **pinTriggerDio** - only has effect if pinTriggerEnabled is enabled. This value indicates which DIO should be used for reading the pinTriggerLevel.
- **serialIoCfgIndex** -
    - Previous devices had no restrictions on which DIO's could be muxed to peripherals. For instance, any and all DIOs could be used for UART/SPI peripherals. This is not the case for the CC23xx devices, there are a very limited subset of DIOs that can be muxed to the UART/SPI peripherals and in addition, there are specific muxing settings for each pin that need to be known ahead of time. For these reasons, the bootloader makes use of a table of possible serial IO configurations. Where each index into the table has its own set of pre-configured DIOS.

```
const bldrIoCfgTable_t BldrIoCfgTable[] = {
    {/* 0: FCFG default for all but QFN40 devices*/
        .rxDio   = 20, .rxMux   = 3,
        .txDio   =  6, .txMux   = 5,
        .misoDio = 12, .misoMux = 1,
        .mosiDio = 13, .mosiMux = 2,
        .sclkDio =  8, .sclkMux = 1,
        .csnDio  = 11, .csnMux  = 1
    },
    {/* 1:*/
        .rxDio   = 12, .rxMux   = 3,
        .txDio   = 13, .txMux   = 3,
        .misoDio = 21, .misoMux = 4,
        .mosiDio = 13, .mosiMux = 2,
        .sclkDio = 24, .sclkMux = 1,
        .csnDio  = 11, .csnMux  = 1
    },
    {/* 2: FCFG default for QFN40 devices (LaunchPad compatible) */
        .rxDio   = 22, .rxMux   = 2,
        .txDio   = 20, .txMux   = 2,
        .misoDio = 12, .misoMux = 1,
        .mosiDio = 13, .mosiMux = 2,
        .sclkDio = 24, .sclkMux = 1,
        .csnDio  = 11, .csnMux  = 1
    }
};
```

- The value of serialIoCfgIndex indicates which index of the BldrIoCfgTable to be used. For example, if serialIoCfgIndex has a value of 0x00, then the 0th index of the table will be used and the DIOS specified in that index are to be used for any/all bootloader communication.

Note: By default, the FCFG sets pin 21 as backdoor pin. FCFG default shouldn't all that important in most cases because if there is no CCFG, then the bootloader will run regardless of trigger pin.

pAppVtor

This pointer is used to specify if/where the application vector table resides. If this value points to valid flash memory, then the bootcode/bootloader will attempt to pass execution on to the application through this vector table. Of course, if the bootcode/bootloader is unable to pass execution to the application for any reason, then the bootloader will still be triggered as a fall back.

Bootloader Entry Conditions

Invocation conditions

Using the FCFG and CCFG struct as described above the bootloader will be entered based on the following conditions.

- Bootloader is configured with CCFG settings if all of the following conditions are true:
  - The device is not exiting a shutdown
  - FCFG is valid
  - FCFG.bootCfg.pBldrVtor != XCFG_BC_PBLDR_FORBID
  - CCFG is valid
  - XCFG_BC_PBLDR_VALID(CCFG.bootCfg.pBldrVtor)

- Bootloader is configured with FCFG settings if all of the following conditions are true:
  - The device is not exiting a shutdown
  - The conditions for the Bootloader to be configured with the CCFG were not met
  - FCFG is valid
  - XCFG_BC_PBLDR_VALID(FCFG.bootCfg.pBldrVtor)
  - CCFG is invalid OR CCFG.bootCfg.pBldrVtor is not XCFG_BC_PBLDR_FORBID

If the conditions are met for the CCFG settings, the bootloader will be invoked using the settings of the CCFG. Conversely, if the conditions are met for the FCFG settings, the bootloader will be invoked using the settings of the FCFG.

Bootloader Definition

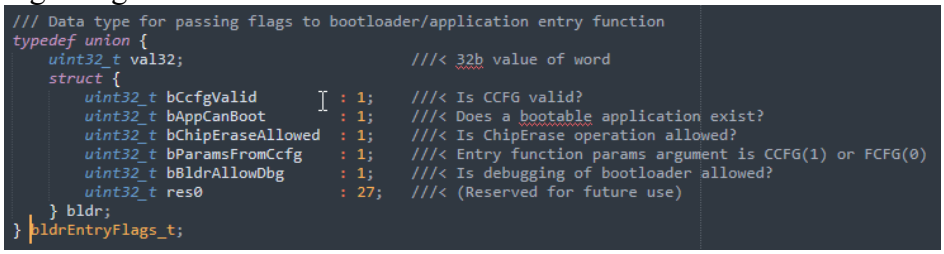
Here is the function prototype of the ROM Bootloader.

```
__noreturn static void  EnterBootloader(serialRomBldrParam_t bldrParam,
bldrEntryFlags_t bldrFlags);
```

Just as the ROM Bootloader has been defined, it's possible for a customer to create their own bootloader in flash. In such a situation, it is important the flash bootloader use the same function prototype and use the bldrParam and bldrFlags as they've been provided from the bootcode.

The ROM Bootloader sets `CFGAP.DEVICESTATUS.BOOTSTA = 0xBA (BLDR_STARTED)` immediately upon invocation.

As seen in the function prototype, there are two input params to any bootloader.

- serialRomBldrParam_t bldrParam
  - We've already seen the definition for this in the CCFG/FCFG
  - This will hold the CCFG or FCFG settings depending on the method of invocation
- bldrEntryFlags_t bldrFlags
  - This is a struct that can be found in platform.h. It provides helpful information regarding the state of the device and the invocation of the bootloader.
  - 

```
/// Data type for passing flags to bootloader/application entry function
typedef union {
    uint32_t val32;                         ///< 32b value of word
    struct {
        uint32_t bCcfgValid       : 1;      ///< Is CCFG valid?
        uint32_t bAppCanBoot      : 1;      ///< Does a bootable application exist?
        uint32_t bChipEraseAllowed  : 1;    ///< Is ChipErase operation allowed?
        uint32_t bParamsFromCcfg    : 1;    ///< Entry function params argument is CCFG(1) or FCFG(0)
        uint32_t bBldrAllowDbg      : 1;    ///< Is debugging of bootloader allowed?
        uint32_t res0               : 27;   ///< (Reserved for future use)
    } bldr;
} bldrEntryFlags_t;
```

The ROM Bootloader uses these two parameters to determine what should be done next. The two possible outcomes are

1. Stay in the ROM Bootloader
2. Pass through to the application in Flash

Stay in the ROM Bootloader

Although the bootloader has been invoked, we still need to determine based on the bldrParams if we should stay in the bootloader. There are two reasons for this.

1. bldrFlags.bldr.bAppCanBoot is False
    1. If there is no application in flash, then the bootloader cannot continue execution and is forced to stay
    2. If there is an application in flash, but for whatever reason it is not able to bootup, the bootloader is forced to stay.
2. The bootloader trigger pin is enabled and active

    If the following conditions are met, then the trigger pin has been activated and the bootloader will stay.

    bldrParams.pinTriggerEnabled is True AND the DIO specified by bldrParams.pinTriggerDio has the value (high/low) specified by the bldrParams.pinTriggerLevel

If it is determined to stay in the bootloader, then the bootloader will begin to select the serial interface being used.

Pass through to the application in flash

Although the bootloader has been invoked, it's still possible that the bootloader decides to continue the boot sequence and pass through to the application in flash.

If the conditions for staying in the bootloader are not met, the bootloader will pass on execution and the application application vector will soon be invoked.

The bootloader sets CFGAP.DEVICESTATUS.BOOTSTA = 0xC0 (BLDR_COMPLETE)

Serial Interface Selection

Just as it was with previous devices, the ROM Bootloader supports SPI and UART. In order to determine which interface is being used in any given situation, it will listen on the UART RX and SPI lines for incoming traffic.

- UART

The bootloader waits for what is called the auto baud sequence. This is simply a two byte alternating bit pattern (0x55). If this sequence is received on the UART RX DIO, the bootloader will recognize that UART is the desired interface and calculate the baud rate it was sent with. After configuring the UART to match the incoming baud rate, a customer is free to send/receive normal bootloader commands.

- SPI

The bootloader waits for any traffic at all over SPI. If the SPI hardware detects properly formatted SPI transactions, the bootloader will recognize that SPI is the desired interface and finish up the SPI configuration. From this point on a customer is free to send/receive normal bootloader commands.

After the serial interface has been selected, the normal command processing loop begins

Packet Protocol

Again, the packet protocol is identical to the ROM Bootloader of past devices. The protocol is described in more detail in the CC26x2 TRM. Here is an image of that section for quick review.

10.2.1 Packet Handling

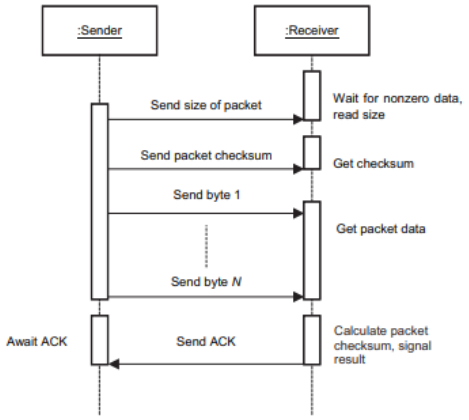
The bootloader uses well-defined packets to ensure reliable communications with the external communicating program. All communications (with the exception of the UART automatic baud [see Section 10.2.2.1]) use these well-defined packets. The packets are always acknowledged or not acknowledged by the communicating devices with defined ACK or NACK bytes.

The packets use the same format for receiving and sending packets. This format includes the method to acknowledge successful or unsuccessful reception of a packet.

While the actual signaling on the serial ports is different, the packet format remains the same for supported UART and SSI interfaces.

Packet send and packet receive must adhere to the simple protocol shown in Figure 10-1.

Figure 10-1. Sequence Diagram for Send and Receive Protocol



Commands

Command IDs

CMD ID	Value
BLDR_CMD_PING	0x20
BLDR_CMD_GET_STATUS	0x21
BLDR_CMD_GET_PART_ID	0x22
BLDR_CMD_RESET	0x23
BLDR_CMD_CHIP_ERASE	0x24
BLDR_CMD_CRC32	0x25
BLDR_CMD_DOWNLOAD	0x26
BLDR_CMD_DOWNLOAD_CRC	0x27
BLDR_CMD_SEND_DATA	0x28

Command List

BLDR_CMD_PING

This command is used to receive an acknowledge from the bootloader proving that communication has been established. This command is a single byte.

Byte	Field	Value	Description
0	cmdId	BLDR_CMD_PING	Ping Command ID

BLDR_CMD_GET_STATUS

This command returns the status of the last command that was issued. Typically this command should be received after every command is sent to ensure that the previous command was successful or, if unsuccessful, to properly respond to a failure. The command requires one byte in the data of the packet and the bootloader should respond by sending a packet with one byte of data that contains the current status code.

Byte	Field	Value	Description
0	cmdId	BLDR_CMD_GET_STATUS	Get StatusCommand ID

Possible returned status codes are as follows.

Status Codes

Return Name	Value	Description

BLDR_CMD_RET_SUCCESS	0x40	This is returned in response to a **BLDR_CMD_GET_STATUS** command and indicates that the previous command completed successful.
BLDR_CMD_RET_UNKNOWN_CMD	0x41	This is returned in response to a **BLDR_CMD_GET_STATUS** command and indicates that the command sent was an unknown command.
BLDR_CMD_RET_INVALID_CMD	0x42	This is returned in response to a **BLDR_CMD_GET_STATUS** command and indicates that the previous command was formatted incorrectly.
BLDR_CMD_RET_INVALID_ADR	0x43	This is returned in response to a **BLDR_CMD_GET_STATUS** command and indicates that the previous download command contained an invalid address value.
BLDR_CMD_RET_FLASH_FAIL	0x44	This is returned in response to a **BLDR_CMD_GET_STATUS** command and indicates that an attempt to program or erase the flash has failed.
BLDR_CMD_RET_CRC_FAIL	0x45	This is returned in response to a **BLDR_CMD_GET_STATUS** command and indicates that the previous CRC32 command match failed.
BLDR_CMD_RET_NEEDS_CHIP_ERASE	0x46	This is returned in response to a **BLDR_CMD_GET_STATUS** command and indicates that the previous Download command failed because a **BLDR_CMD_CHIP_ERASE** command must be run first.

BLDR_CMD_GET_PART_ID

This command is sent to the bootloader to get the Part ID of the device.

Byte	Field	Description
0	cmdId	BLDR_CMD_GET_PART_ID command ID

BLDR_CMD_RESET

This command is used to tell the bootloader to reset. This can be is used after downloading a new image to the device to cause the new application to start from a reset. The normal boot sequence occurs and the image runs as if from a hardware reset. It can also be used to reset the bootloader if a critical error occurs and the host device wants to restart communication with the boot loader.

The bootloader responds with an ACK signal to the host device before actually executing the system reset of the device running the bootloader. This informs the updater host device that the command was received successfully and the part will be reset.

Byte	Field	Description
0	cmdId	BLDR_CMD_RESETcommand ID

BLDR_CMD_CHIP_ERASE

This command is used to perform a chip erase of the device. All main flash bank sectors not protected by FCFG1 and CCFG protect bits will be erased. The CCFG will be erased once the bank erase has completed.

This command will first invalidate the CCFG and then begin erasing all unprotected sectors in the main flash bank. Once the flash sectors have been erased, it will finally erase the contents of the CCFG.

If the CCFG permissions disallow a chip erase, the command will respond with CMD_INVALID_CMD.

Byte	Field	Description
0	cmdId	BLDR_CMD_CHIP_ERASE command ID

BLDR_CMD_CRC32

This command is sent to the bootloader to calculate a CRC32 for a specified memory area. The command consists of three 32-bit values that are each transferred MSB first.

The Memory address must be sector aligned. Only memory addresses within the main flash region or the CCFG_BASE address itself are valid.

The Size must be sector aligned or (sector - 4 bytes) aligned.

The combination of memory address and size cannot go outside of either the main flash region or the CCFG region. If the parameters are valid, the command will only report if the expected CRC matches the calculated CRC. One should follow up this command with the **BLDR_CMD_GET_STATUS** to read the result of the CRC comparison.

Byte	Field	Description
0	cmdId	BLDR_CMD_CRC32 command ID
1	Memory Address [31:24]	Memory Address to start the CRC calculation
2	Memory Address [23:16]	
3	Memory Address [15:8]	
4	Memory Address [7:0]	
5	Memory Area Size [31:24]	Number of bytes to run the CRC calculation over
6	Memory Area Size [23:16]	
7	Memory Area Size [15:8]	
8	Memory Area Size [7:0]	
9	Expected CRC [31:24]	The CRC value the host is expecting the CRC calculation will result in
10	Expected CRC [23:16]	
11	Expected CRC [15:8]	
12	Expected CRC [7:0]	

BLDR_CMD_DOWNLOAD

This command is sent to the bootloader to indicate where to store data and how many bytes will be sent by the **BLDR_CMD_SEND_DATA** commands that follow. The command consists of two 32-bit values that are both transferred MSB first.

The first 32-bit value is the address to start programming data into, while the second is the 32-bit size of the data that will be sent.

This command should be followed by **BLDR_CMD_GET_STATUS** to ensure that the program address and program size were valid for the device running the bootloader.

Byte	Field	Description
0	cmdId	BLDR_CMD_DOWNLOAD command ID
1	Program Address [31:24]	Start address of the download
2	Program Address [23:16]	
3	Program Address [15:8]	
4	Program Address [7:0]	
5	Program Size [31:24]	Number of bytes (length of the download)
6	Program Size [23:16]	
7	Program Size [15:8]	
8	Program Size [7:0]	

BLDR_CMD_DOWNLOAD_CRC

This command is sent to the bootloader to indicate where to store data, how many bytes will be sent by the **BLDR_CMD_SEND_DATA** commands that follow and the crc32 value covering all the bytes.

The command consists of three 32-bit values that are all transferred MSB first. The first 32-bit value is the address to start programming data into, the second is the 32-bit size of the data that will be sent and the third is the 32-bit crc expected value.

This command should be followed by a **BLDR_CMD_GET_STATUS** to ensure that the program address and program size were valid for the device running the bootloader.

Byte	Field	Description
0	cmdId	BLDR_CMD_DOWNLOAD_CRC command ID

1	Program Address [31:24]	Start address of the download
2	Program Address [23:16]	
3	Program Address [15:8]	
4	Program Address [7:0]	
5	Program Size [31:24]	Number of bytes (length of the download)
6	Program Size [23:16]	
7	Program Size [15:8]	
8	Program Size [7:0]	
9	Expected CRC [31:24]	The expected CRC calculation over the completed download image. If the CRC calculation fails, the download will be considered a failure and immediately erase the content that was downloaded.
10	Expected CRC [23:16]	
11	Expected CRC [15:8]	
12	Expected CRC [7:0]	

BLDR_CMD_SEND_DATA

This command should only follow a **BLDR_CMD_DOWNLOAD** command or another **BLDR_CMD_SEND_DATA** command, if more data is needed.

Consecutive send data commands automatically increment the address and continue programming from the previous location. The caller should allow the device to finish the flash programming before issuing another command in order to avoid overflowing the input buffers of the serial interface. The command terminates programming once the number of bytes indicated by the **BLDR_CMD_DOWNLOAD** command has been received. Each time this function is called, it should be followed by a **BLDR_CMD_GET_STATUS** command to ensure that the data was successfully programmed into the flash.

If the bootloader responds with a **BLDR_CMD_NAK** to this command, the bootloader will not increment the current address to allow re-transmission of the previous data.

A maximum of 253 bytes of data can be sent per **BLDR_CMD_SEND_DATA** command.

Byte	Field	Description
0	cmdId	BLDR_CMD_SEND_DATA command ID
1	download_image[0]	Consecutive bytes of the image to be downloaded
....	....	
X <= 253	download_image[X <= 253]	

FW Image Update Example

The following steps can be followed to perform a FW image update to a device enabled to run the ROM Bootloader.

For this example, let's assume we have an updated application which begins at address 0x00000000, has a length of 0x28000 and a CRC over all of the bytes has a value of 0xFACEFACE.

1. The device needs to bootup into the ROM Bootloader. This can be done either by setting the pAppVtor to an invalid value OR by setting the pinTriggerDio to the specified pinTriggerLevel.
2. The bootloader needs to know which serial interface is being used.
    1. UART - send the AutoBaud sequence as described in the CC26x2 Technical Reference Manual
    2. SPI - send any ROM Bootloader cmd (pin command is a good suggestion)
3. Now that the bootloader is triggered and it you're communicating with it correctly, prepare the device for the FW image update
    1. Send the BLDR_CMD_CHIP_ERASE command
    2. Wait for the ACK/NACK and then send the BLDR_CMD_GET_STATUS
    3. Wait for the ACK/NACK and ensure the previous command was completed successfully with the BLDR_CMD_RET_SUCCESS value
4. Start the Application Download
    1. Send the BLDR_CMD_DOWNLOAD_CRC command, passing as input the startAddress=(0x00000000), length=(0x28000), CRC=(0xFACEFACE)
    2. wait for the ACK/NACK and then send the BLDR_CMD_GET_STATUS
    3. wait for the ACK/NACK and ensure the previous command was completed successfully with the BLDR_CMD_RET_SUCCESS value

5. Loop over the bytes of the image and send the data to the ROM Bootloader
    1. Send the BLDR_CMD_SEND_DATA command, passing as input the next 253 bytes of the application image.
    2. wait for the ACK/NACK and then send the BLDR_CMD_GET_STATUS
    3. wait for the ACK/NACK and ensure the previous command was completed successfully with the BLDR_CMD_RET_SUCCESS value
    4. Repeat steps 5a-5c until all 0x28000 bytes have been transferred
6. Start the CCFG Download
    1. Send the BLDR_CMD_DOWNLOAD_CRC command passing as input the startAddress=(0x4E020000), length=(2048), CRC=(calc_crc(ccfg_contents))
    2. wait for the ACK/NACK and then send the BLDR_CMD_GET_STATUS
    3. wait for the ACK/NACK and ensure the previous command was completed successfully with the BLDR_CMD_RET_SUCCESS value
7. Loop over the bytes of the CCFG contents and send the data to the ROM Bootloader
    1. Send the BLDR_CMD_SEND_DATA command, passing as input the next 253 bytes of the CCFG content.
    2. wait for the ACK/NACK and then send the BLDR_CMD_GET_STATUS
    3. wait for the ACK/NACK and ensure the previous command was completed successfully with the BLDR_CMD_RET_SUCCESS value
    4. Repeat steps 5a-5c until all 2048 bytes have been transferred
8. Reset the device either by pulling the RST pin externally or by sending the BLDR_CMD_RESET command.
    1. Keeping in mind that the triggerPin should now be inverted so that the bootcode/bootlaoder can freely pass execution onto the application this time around
9. DONE! The device will now bootup into the new flash content that has been programmed to it.