Freescale Semiconductor
Application Note

Document Number: AN3860
Rev. 1.0, 02/2010

MC1322x Flash Loader Utility (Second Stage Loader)

1 Introduction

The Freescale MC13224 for the 2.4 GHz IEEE® 802.15.4 Standard is a 32-bit ARM7 core based device with an onboard 128 Kbyte serial FLASH memory. At system boot, the binary program code for the ARM7 CPU core gets transferred from the serial FLASH to the MCU SRAM (96Kbytes total available), and then application execution transfers to the SRAM and runs from there. The target application image that resides in the FLASH is typically loaded into the MC13224 before board assembly or is loaded as part of the manufacturing process.

Once in the field, it may become necessary to load a new image into the onboard FLASH. A new FLASH image can either be loaded via the standard ARM7 JTAG-based debug tools or via the MC13224 boot process. Use of the debug/development tools is addressed elsewhere.

This application note describes how an new executable binary image can be loaded into the onboard serial FLASH of the MC1322x using the boot process and provides tools to assist that process. The tools consist of two separate pieces.

Contents

- The primary function is a target application (Second Stage Loader) for the ARM7 that gets loaded at boot time. This utility application is then used to load a new target application binary image into the onboard serial FLASH using the UART1 serial port.
- The secondary function is an example PC application (Console Loader) that communicates to the MC1322x UART1 port using a PC COM port. This PC app supports the protocol to the target Second Stage Loader to load the binary image to the MC1322x which is then written to the serial FLASH.

NOTE

- Using the SSL is a two stage process, where the MC1322x onboard serial FLASH must first be erased before it can be reloaded. See Section 2.2, “Clearing (Erasing) FLASH.
- To use a PC to communicate with the MC1322x, auxiliary hardware must be provided connecting the 22x UART1 port to either a standard PC serial COM port using RS232 translators or to a PC USB port using a UART <> USB serial interface IC creating a virtual COM port (VCP). This is left to the user, however, Freescale provides examples of both hardware designs

The applications are provided as executables as well as source. They are useful in their executable form, but they can also be adapted to other scenarios or platforms, such as a Linux-based host, manufacturing test environment, or a primary MCU host.

This application note first describes the background requirements for loading a new FLASH image. Second, the applications are described to allow a “quick start” use of these tools. Finally, detailed information is given on the implementation of the tools such that the user can adapt them to the individual target application. This document is intended for users of MC1322x that need to update the executable stored in the internal FLASH from a separate host processor, during production or after the product has been deployed.

NOTE

- Users must be familiar with the MC1322x platform. See the *MC1322x Reference Manual* (MC1322xRM).
- The MC1322x can be used with or without (normal operation) operation of its optional onboard buck voltage regulator. See AN3962, *MC13244 Configuration and Operation with the Buck Regulator.* The Second Stage Loader application works with either configuration.
- Users must also be familiar with the BeeKit Wireless Connectivity ToolKit.
- While the sample applications have been tested and are believed to be correct, users are responsible for their applications and that they meet the reliability requirements of the specific project. Thoroughly testing the process before deployment is recommended.
- Source code and application code is available on the Freescale website as a link related to this application note.

2 MC1322x Boot Process

It is important to understand the MC1322x boot process as background to using the load utility. Figure 1 shows a very simplified view of the boot flow. The boot process must first be used to clear/erase the FLASH before the load utility can be used with the boot process to write a new image to the FLASH

NOTE

For additional detailed information on the boot process, refer to the *MC1322x Reference Manual* (MC1322xRM), Chapter 3 and Appendix C.

2.1 Bootstrap Flow Overview

Upon exiting reset, the MC1322x executes an application stored in the ROM – called the bootstrap – that:

1. Determines if FLASH is to be cleared (erased)
2. Determines if a valid FLASH image is present
3. If a valid FLASH image is not present, alternatively, determines a secondary boot source
4. Fetches the needed execution binary bytes from the determined boot source and loads them into the MCU RAM
5. Lastly, starts execution of the application from the bottom of RAM.

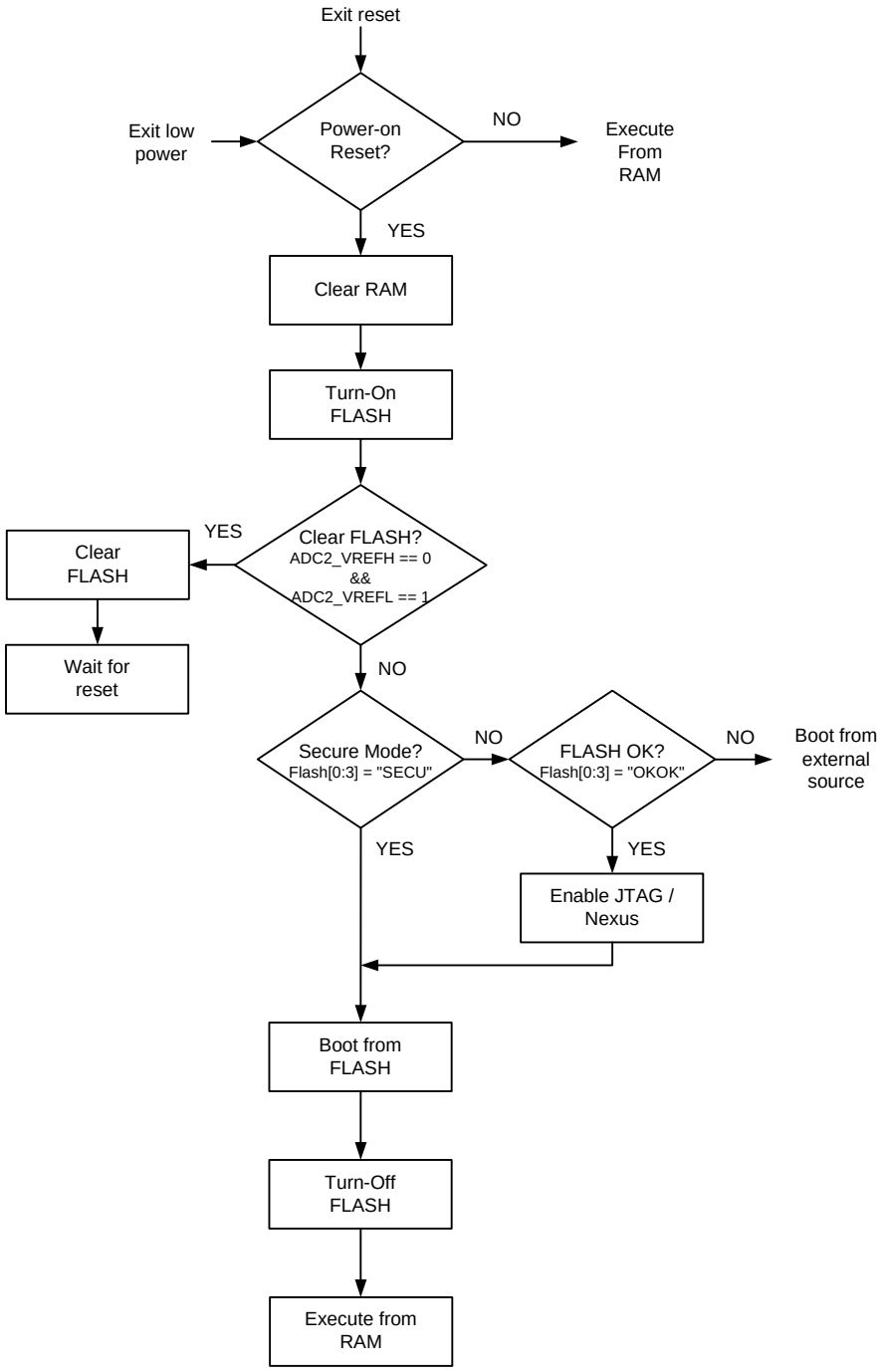

Figure 1. Simplified Bootstrap Flow

Observing Figure 1, the device initialization flow algorithm is:

1. After reset exit, go to execution from RAM immediately if the reset condition was from a low power state, else go to Step 2.

2. Clear RAM
3. Turn-on FLASH (power it up)
4. Clear FLASH?: if external signals ADC_VREFH = 0 AND ADC2_VREHL = 1, then clear (erase) the FLASH and wait, else go to Step 5

NOTE

- This is an important step - the boot routine cannot be used to load a new FLASH image if an existing valid FLASH image is present. More is said later on how this impacts use of the loader tool
- The FLASH is a 128Kbyte device. When it is cleared, the upper most 4kbyte segment is not touched as it is reserved, so only the lower 124kbytes are erased.

5. Is the FLASH a “secure” valid image?: if the first 4 bytes of FLASH content = “SECU” (ASCII), then go to Step 8, else go to Step 6.

NOTE

This test is done first, because if the image is secured, the boot process does not enable the debug ports so that a user cannot gain entry to the device and copy the image.

6. Is the FLASH an un-secured valid image?: if the first 4 bytes of FLASH content = “OKOK” (ASCII), then go to Step 7, else go to external boot source.

NOTE

Going to an external boot source enables use of the loader tool to supply the FLASH image. This part of the process is discussed later.

7. Enable debug ports
8. Boot from FLASH: transfer binary executable image from FLASH to RAM (starting at lowest RAM address 0x0040_0000)
9. Turn-off FLASH (power it down)
10. Start execution from bottom of RAM (CPU address 0x0040_0000)

2.2 Clearing (Erasing) FLASH

As previously stated, a new FLASH image cannot be loaded via the boot process if an existing valid FLASH image is present. As a result, the boot process needs to be run one time to clear the image before actually going through the load boot process using an external port to supply the boot image.

Freescale provides a number of MC1322x development boards and reference design circuits, and Figure 2 illustrates a typical FLASH erase hardware detail. In this example, shorting bars must be placed on two separate headers and must be present for a boot cycle. The boot flow will test these pins and clear the FLASH accordingly.

A user must provide a similar hardware capability on a target board if the application ever requires reloading of a secure FLASH image or reloading of a FLASH image via the boot flow versus a debug port.

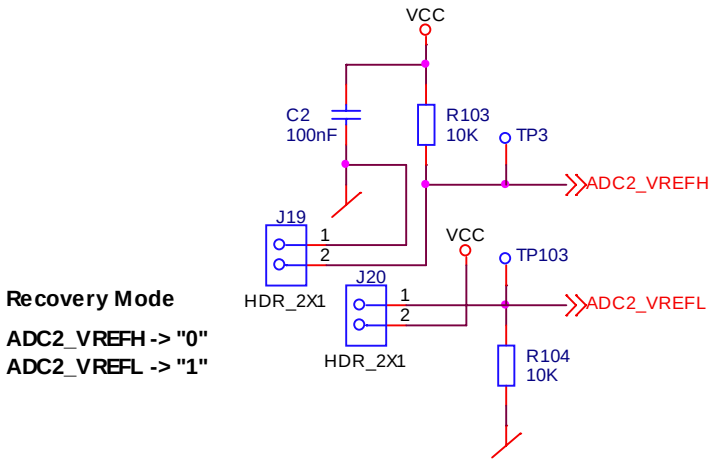

Figure 2. MC1322x FLASH Erase Circuit

NOTE

Figure 2 is an example of one possible circuit implementation. Refer to the appropriate hardware reference manual and the specific schematic for a given Freescale board or design.

2.3 Valid FLASH Boot Image

The boot image stored in FLASH follows the format of Table 1. The image includes the following contents:

1. Signature - initial 4-byte parameter is read by the boot program to determine a valid image. Contents must be either “SECU” or “OKOK” in ASCII.
2. Length - second 4-byte binary parameter that equals the number of valid bytes of executable code that follows in FLASH. Maximum number is $98{,}296_{dec}$ (limited by MC1322x RAM size)
3. Binary executable bytes - number of bytes matches the value of the Length variable

Table 1. Valid FLASH Boot Image

Byte Order	Byte Content	Value
0	Signature (ASCII)	“S”/”O”
1		“E”/”K”
2		“C”/”O”
3		“U”/”K”

Table 1. Valid FLASH Boot Image

Byte Order	Byte Content	Value
4	Length (binary)	LSB
5		
6		
7		MSB
8[1]	Executable (binary)	
9		
10		
11		
*		
*		
Length		

1 For any 32-bit code word the first byte resides at the lowest byte rail in memory (little endian)

2.4 Booting From FLASH (Normal Flow)

Booting from a valid FLASH image is the normal operational flow. Reviewing the algorithm in Section 2.1, “Bootstrap Flow Overview” for booting from FLASH:

1. The FLASH is not cleared
2. A valid signature is found (either “SECU” or “OKOK”) and the debug ports are enabled if the signature is “OKOK”
3. The length variable is read to know how many code bytes to transfer
4. The executable binary code bytes are serially accessed and transferred to RAM starting at lowest RAM address
5. The ROM boot flow jumps execution to the bottom of RAM (address 0x0040_0000)

The following items apply:

- Any specified check that fails results in the bootstrap moving to the next alternative boot source to be used – the UART1 port.
- If the FLASH has been erased, it aborts this process to the next alternative boot source
- Be aware that some MC1322x GPIO are affected by the boot process. See the *MC1322x Reference Manual* (MC1322xRM), Chapter 3, Section3.11.7.

2.5 Alternative External Boot Sources

If the normal flow fails, i.e., a valid FLASH image is not found, the boot flow seeks an alternate source of boot code from an external port, see Figure 3. The possible external boot sources include (listed in the order in which they are checked):

- UART1 serial port
- SPI port with MC1322x as slave, attached to a master device
- SPI port with MC1322x as master, attached to an external serial EEPROM or FLASH
- I2C port with MC1322x as master, attached to an external serial EEPROM or FLASH

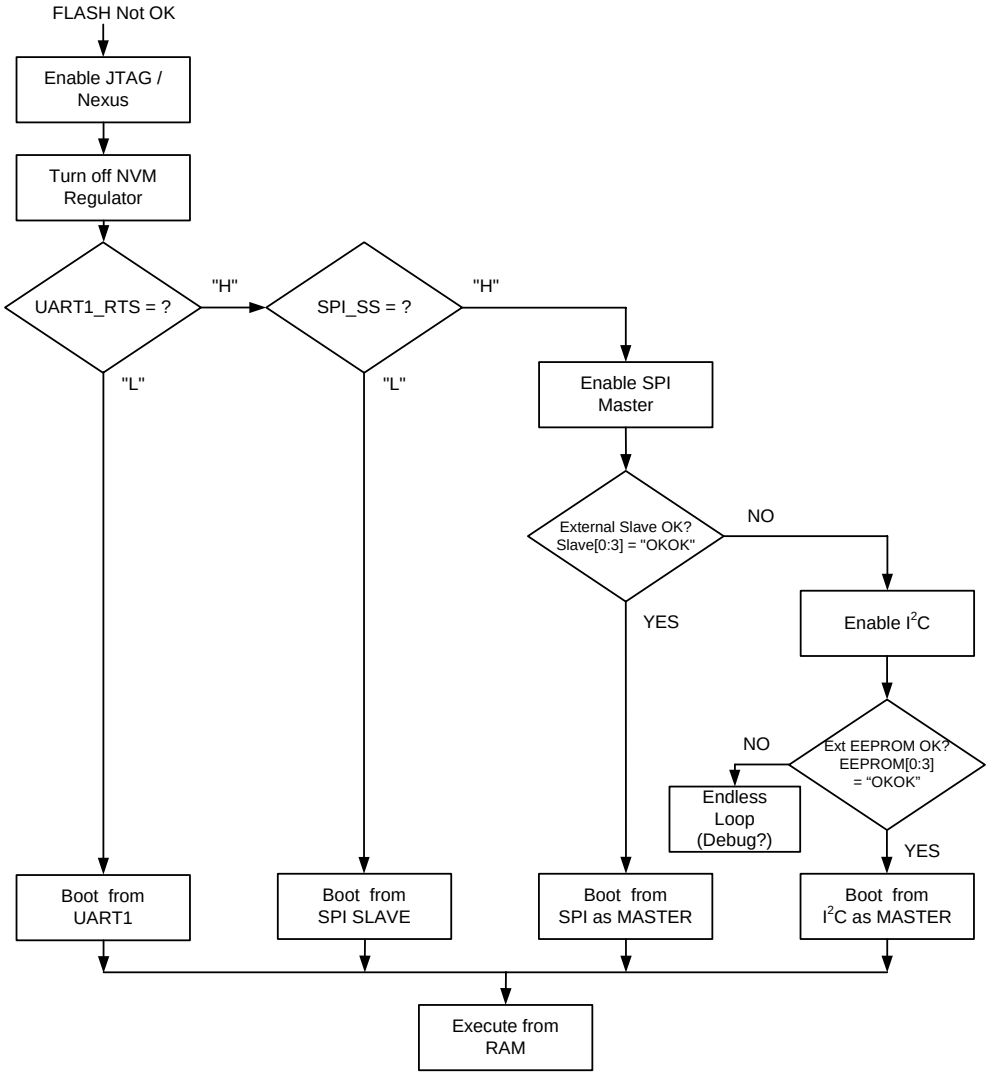

Figure 3. External Sources Boot Flow

The external source is expected to provide properly formatted data (Section 2.6, “Booting From UART1) to load into RAM. The boot process from an external source does not load an image directly into FLASH, rather, using the defined format, the binary executable data is loaded directly into RAM and then execution flow is transferred to RAM.

The boot flow first seeks a source from the UART1 port.

- The UART1 flow control input UART1_RTS must be driven low to enable the boot flow to seek code from the UART port.

- This mode tends to be most commonly used.
- The data flow to the UART1 port must meet timing and protocol requirements (baud rate is detected by the boot flow).

As seen in Figure 3, the other ports are checked relevant to the states of the UART1_RTS and SPI_SS signals. The user is directed to the *MC1322x Reference Manual* (MC1322xRM), Appendix C for a detailed discussion of these other boot ports.

NOTE

As supplied, the Second Stage Loader uses the UART1 port as the boot source. The tool can be modified by the user to support one of these other sources because source code is provided.

2.6 Booting From UART1

Boot from UART1 is described in more detail because the Second Stage Loader uses this port.

NOTE

- UART1_RTS control input must be held low by hardware before the boot flow is entered; the signal must be in the proper state before its mode is tested. There is no synchronization event/signal possible.
- The executable data format consists of the 4-byte Length parameter and the actual executable binary, similar to that described in Table 1 minus the 4-byte Signature.

The following steps are performed by the bootstrap while executing UART1 boot mode:

1. Test UART1_RTS - this pin must be driven low to indicate that a host is ready to provide the executable code to load into RAM. If UART1_RTS = 0, go to Step 2, else go to SPI_SS test.
2. Select UART baud rate (up to 2Mbaud) -the host sends the ASCII character number ‘\0’. After the baud rate is detected, go to Step 3.

NOTE

The host should repeatedly send character ‘\0’ until it receives the response of Step 3.

3. Send the string “CONNECT” (ASCII) - this is the response to the host using the baud rate determined at Step 2. The connection is now established and confirmed.
4. Wait for the Length parameter from the host - 4 bytes (32-bit binary number) are sent by the host that is the Length parameter for the following transfer. The first byte sent is the LSB.
5. Receive executable data bytes - image bytes are received as they are sent by the host and copied into RAM. The first byte received is placed into the bottom of RAM, and the RAM data are built from this point as data bytes are received. The process continues until the number of bytes equal to the Length parameter are received.

NOTE

No confirmation response is send to the host, nor is there any CRC check.

6. Jump to RAM address 0x0040_0000 - ROM-based boot flow ends; execution transfers to the RAM-based app.

Figure 4 shows the event flow for the UART1 boot.

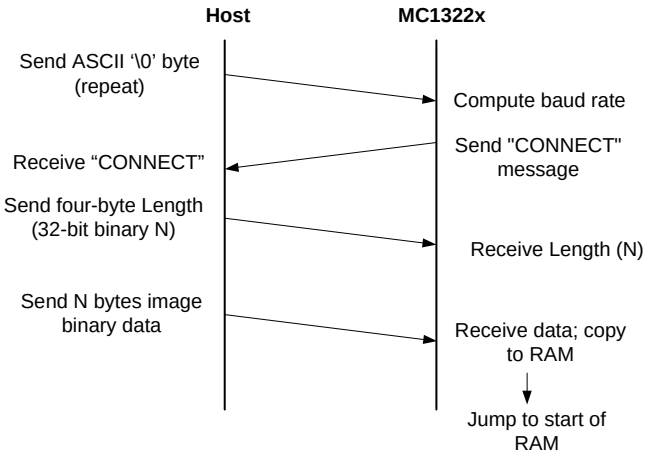

Figure 4. UART1 Boot Flow

Additional items:

- It is recommended that the host attempt multiple sequences of sending the ‘\0’ byte and waiting to receive the “CONNECT” message, until success.
- When booted through UART1 and starting execution from RAM, the bootstrap will leave the following signals configured as used during the boot process:
  - GPIO_38 (ADC2_VRefH) - GPIO mode, input, read from pad, pullup enabled
  - GPIO_39 (ADC2_VRefL) - GPIO mode, input, read from pad, pulldown enabled
  - GPIO_14 (UART1_TX) - Function1 mode
  - GPIO_15 (UART1_RX) - Function1 mode, pullup enabled
  - GPIO_16 (UART1_CTS) - Function1 mode
  - GPIO_17 (UART1_RTS) - Function1 mode, pullup enabled

3 Quick Start Guide for Using Second Stage Loader with PC Host App “Console Loader”

After review of the boot process, it becomes evident that one possible approach to update the FLASH image is to load an application in RAM and then to use the application to load FLASH. The sequence to update the FLASH image is:

- Erase the existing FLASH image
- Load a target application through UART1 that can receive a new FLASH image from the host
- Use the target application to:
  - Receive the new FLASH image from the host
  - Write the image to FLASH

— Write the required FLASH header (Signature and Length) so that subsequent resets cause the MC1322x to boot from FLASH

Freescale provides two software utilities to implement this flow:

- Second Stage Loader (SSL) - is an embedded MC1322x application that can write, read and commit images to FLASH. All of these operations are performed under the control of a host by executing commands received through UART1.
- Console Loader (CL) - is a PC application that acts as host to the MC13224 that can:
  1. Connect to UART1 during the boot flow
  2. Send the SSL application binary (during boot)
  3. Connect/communicate with the SSL (through the SSL command interface)
  4. Load the desired FLASH image
  5. Write the FLASH header

NOTE

- The Console Loader is provided as a PC application because the PC is the most convenient environment to demonstrate this flow. Freescale offers several MC1322x development boards that can be connected to a PC through a USB-based virtual COM port interface to UART1 of the MC1322x.
- SSL and CL are both available as executables and as source
- SSL is described in Section 4, “Second Stage Loader (SSL)
- CL is described in Section 5, “Console Loader (CL)

3.1 Preparing Software for Use

To execute a FLASH update sequence, three software files must be present on the PC:

- SSL executable - “*ssl.bin*”
- CL executable - “*ConsoleLoader.exe*”
- MC1322x FLASH image - “*myApp.bin*”

Files *ssl.bin* and *ConsoleLoader.exe* must reside in the same directory; for our example this is:

```
c:\CL.
```

The FLASH image binary can reside in a separate directory and can be called by the CL application command line.

3.2 Updating the FLASH Image in the MC1322x

The following section describes the step-by-step procedure to load FLASH using the SSL and CL tools. The target for this description can be either of the Freescale MC1322x Network Node or MC1322x Sensor Node development boards.

3.2.1 Erase the FLASH

Before trying to update the internal FLASH of the MC1322x, the user must ensure that the onboard FLASH is erased. As described in Section 2.2, “Clearing (Erasing) FLASH”, the bootstrap must be run with the hardware jumpers (shorting bars) in place to enable FLASH erase.

Two common development boards for the MC1322x are the Network Node (MC1322NN) and Sensor Node (MC1322xSN). Using these two boards as examples, the required locations for mounting jumpers are shown in Figure 5 and Figure 6.

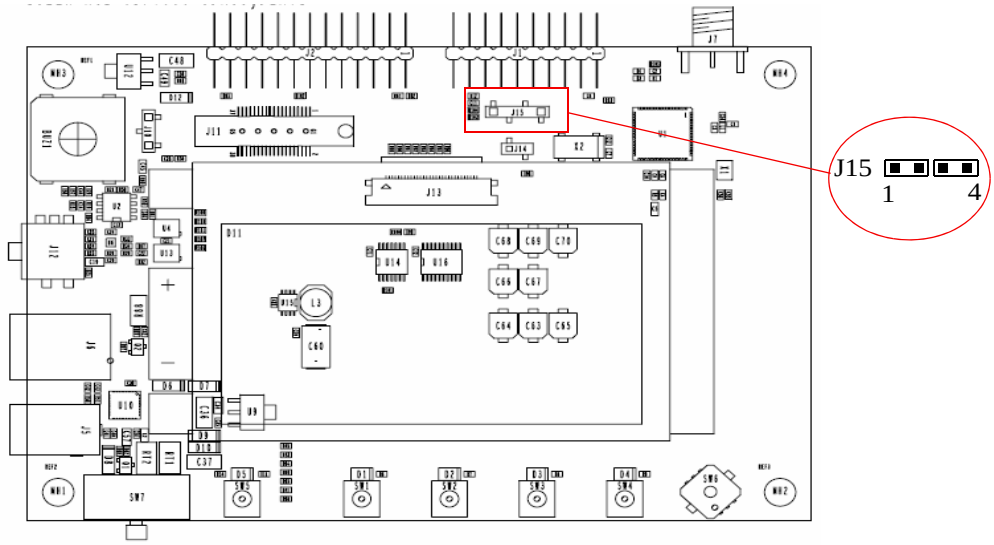

Figure 5. MC1322x Network Node FLASH Erase Jumper Location

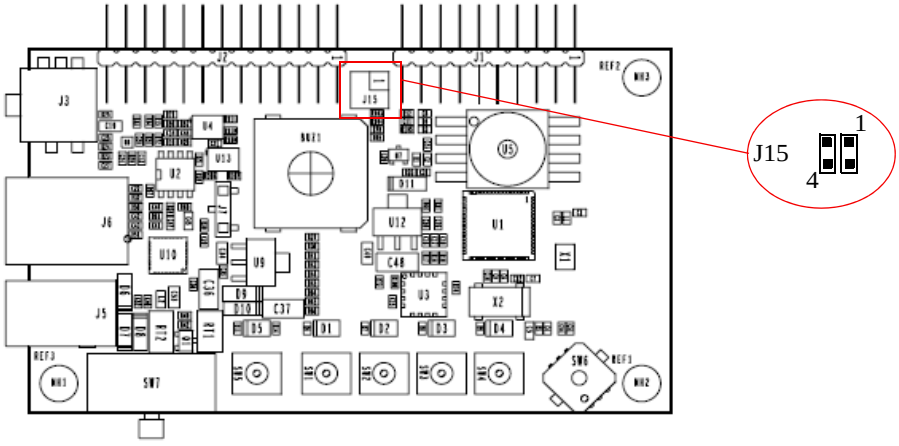

Figure 6. MC1322x Sensor Node FLASH Erase Jumper Location

NOTE

For a user’s target board, the required jumpers must similarly be in place.

To erase the FLASH, perform the following sequence:

1. Turn-off power to board.

2. Place required shorting bars.
3. Turn-on power and wait a short period.
4. Turn-off power to board.
5. Remove the two shorting bars
6. Turn-on power when ready to boot from an external source

3.2.2 Connect the Hardware

The target board must be connected to the PC, and communication uses a standard serial COM port or a USB based virtual COM port:

- For the example MC1322xNN and MC1322xSN modules - these boards provide USB<>UART1 connection through a USB-serial interface chip. The USB virtual COM drivers must be present.
- For target boards without either RS232 buffers or a USB<>UART interface device - a translator board will be required to connect the PC to the target module. This is left as a requirement to the user.

NOTE

The SSL is programmed for a 115200 baud rate. The CL PC application is also programmed for this rate.

3.2.3 Run the Tools (Load SSL and Update FLASH)

The hardware is now ready to support running the tools:

- Be sure the jumpers are removed from the module
- Power-up the target module.

The suggested procedure is described:

1. Determine the COM port associated with the target board - this may be the USB virtual COM connection between the PC and the development board or a true serial COM port. The Windows “Device Manager” allows users to see the connection. Figure 7 shows the “Ports (COM&LPT)” group and the “USB Serial Port (COMxxx)” name.

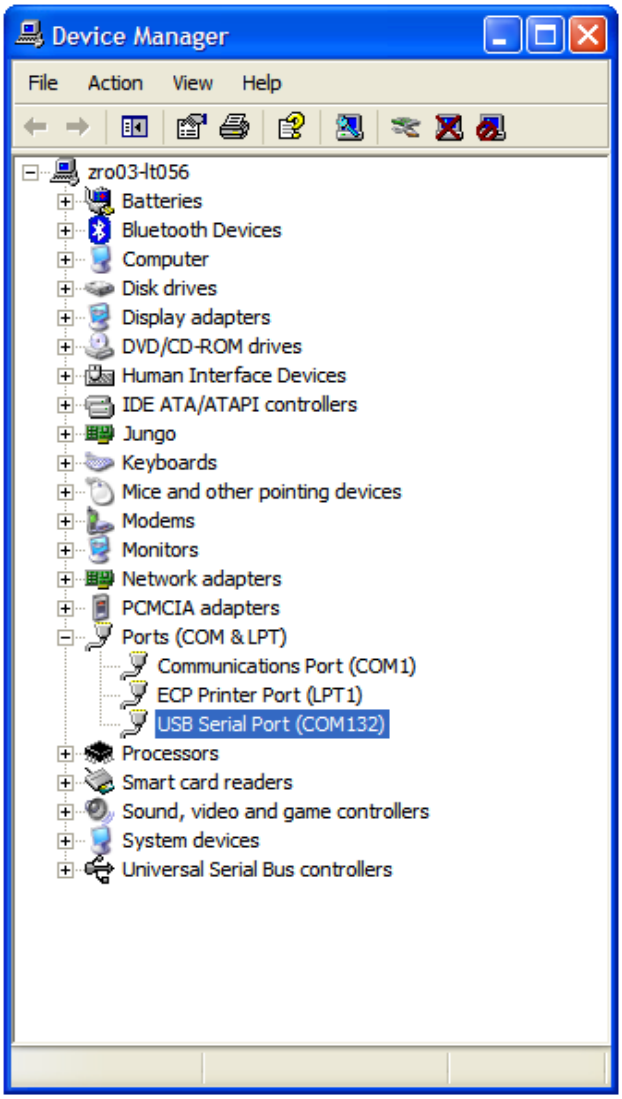

Figure 7. Device Manager Window COM Port

2. When powered the MC132xx module device will exit reset and execute a bootstrap sequence where it will be waiting for boot input from the UART1 port.

NOTE

As previously noted, the UART1 flow control input UART1_RTS must be driven low to before initiating the boot sequence.

3. The CL application is now launched - first open a Cmd Window. The CL application is called via a command line that needs three command line parameters:

```
ConsoleLoader.exe <COM #> <Security Option> <PATH\myApp.bin>
```

— `COM #` - the COM number is the port that communicates to the target board

— `Security Option` - this is the security option for the image to be written to FLASH:

– ‘u’ or ‘U’ for unsecured

– ‘s’ or ‘S’ for secured

— `PATH\myApp.bin` - this is the path to the binary that will be written to the FLASH

As a requirement, the binary of the SSL application must reside in the same directory as the CL executable.

As a command line example, a call to write the image `c:\myApp.bin` (secured) to the internal FLASH of the MC1322x board connected to COM132, appears as follows:

```
c:\CL\ConsoleLoader.exe 132 S c:\myApp.bin
```

Alternatively, a call to write the same image (unsecured) appears as follows:

```
c:\CL\ConsoleLoader.exe 132 U c:\myApp.bin
```

If all the initial application checks on the input parameters are passed (see Section 5.3, “CL Application Processing for details), users are asked to press the reset button on the MC1322x board. Pressing reset allows the CL to initiate communication with the MC1322x and download the SSL into the RAM, and subsequently the user specified binary in the internal FLASH.

Figure 8 shows a successful update session on the PC terminal.

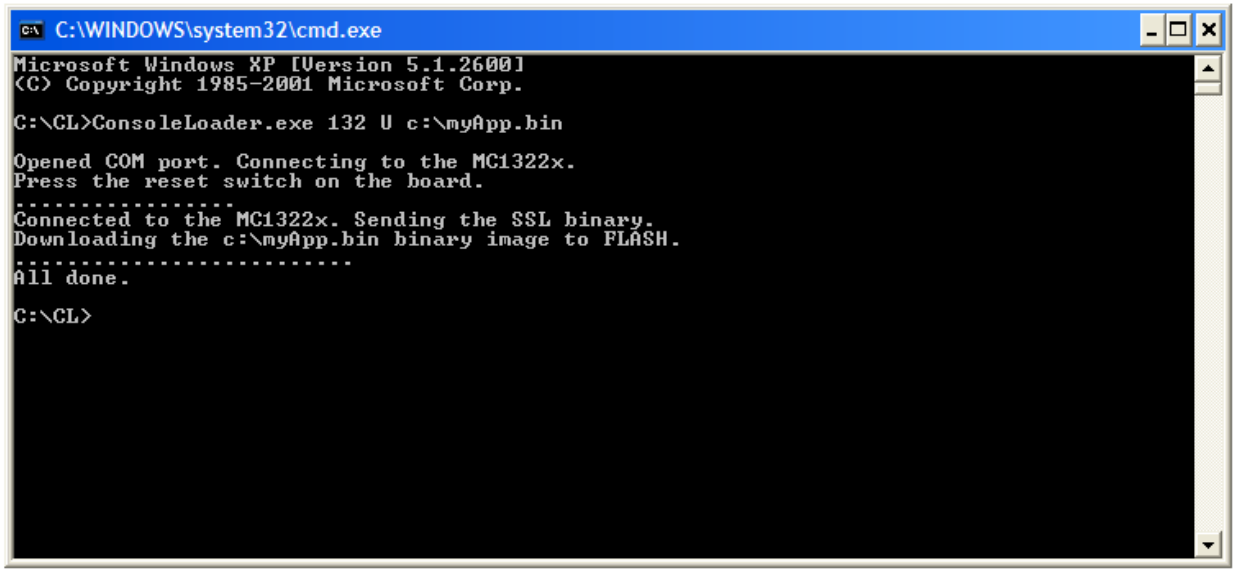

Figure 8. Running the Console Loader

NOTE

An unsuccessful session will report an appropriate error message.

4. The FLASH will now have a valid image for normal boot after a reset or power cycle.

4 Second Stage Loader (SSL)

The Second Stage Loader (SSL) utility is an MC1322x embedded application that can write, read and commit images to FLASH.

- When the MC1322x boots from an external source, any application can be loaded and then take control of the CPU. SSL gets loaded at boot through UART1 and then facilitates transferring a target FLASH image from the host to writing/committing it to FLASH
- Because SSL in loaded through UART1, all its operations are performed under the control of a host by executing commands received through the UART.
- Source is available for SSL, and as a result, the user can modify this application to make use of an alternate boot source for a similar function.

NOTE

The SSL is programmed for a 115200 baud rate by default. Any host application must also support this baud rate. Also when supplying the *ssl.bin* during the boot process, the same baud rate should be utilized.

This chapter details the capabilities and details of the SSL.

4.1 SSL UART Command Interface

The SSL executes a set of predefined commands, received through the UART1 interface. The commands are encapsulated in a generic UART frame of variable length, as presented in Table 2.

Table 2. SSL UART Command Format

Byte 0	Byte 1	Byte 2	Byte 3	Byte	Byte N+2	Byte N+3
SOF	Length		Command (N Bytes)			CRC

The frame is composed of the following elements:

Start Of Frame (SOF)	Always 0x55. Signals the beginning of a frame.
Length	2 bytes, little endian number. This is the length of the command contained within the frame, without the SOF, length and CRC. It is equal to N for the presented frame.
Command	The command contained by the frame.
CRC	One byte sum of the byte values of the N bytes of the command field

This frame format is used for commands received by the SSL as well as for confirms or responses sent by the SSL to the host.

4.2 SSL UART Commands

Table 3 presents a UART command summary

- All commands start with a field identifying the command followed by specific command fields.
- Multi-byte fields are always written in little endian format.
- The command structure, identification field values and status values are defined in the `Engine.h` file.

Table 3. SSL UART Command Summary

Command Name	Cmd ID	Description
Read Request	0x01	Host requests a read of FLASH contents; provides start address and length
Read Response	0x02	SSL response to Host “Read Request”; returns status, length and data
Write Request	0x03	Host requests a write of FLASH contents; provides start address, length, and data
Commit Request	0x04	Host requests a write of the FLASH Length and Signature fields; provides Length and Signature choice
Erase Request	0x05	Host requests an erase of FLASH contents; provides address
Confirm	0xF0	SSL response to Host command(s)

4.2.1 Read Request

The read request triggers a read of FLASH data starting at the “Start address” in the internal FLASH of “Length” bytes to be performed.

- If the command is received and interpreted correctly by the MC1322x, the host receives a read response that contains the data read during the operation.
- If the command was not received correctly, a confirm message is sent back to the host.

Table 4 shows the command format.

Table 4. Read Request Cmd Format

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
ID (0x01)	Start Address				Length (N)	

4.2.2 Read Response

The read response is sent by the MC1322x to the host as a response to a read request received and correctly interpreted.

- “Status” field (0x00 = success, 0x01= failure) - denotes the success status of the requested read operation.
- If the status is success, then the response holds “Length” bytes in the “Data” section, read from the start address specified in the read request.

Table 5 shows the command structure.

Table 5. Read Response Cmd Format

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte . . .	Byte N+3
ID (0x02)	Status	Length (N)		Data (N bytes)		

4.2.3 Write Request

The write request triggers a write to the FLASH, starting from the specified “Start address” of “Length” bytes from the “Data” buffer received in the command. A confirm message is sent by the MC1322x after the command is received and executed.

NOTE

Any byte written in the internal FLASH memory is verified prior to the confirm message for the command being sent to the host. Ensure that the FLASH area to be written is erased beforehand.

A confirm message is sent to the host by the MC1322x after the execution of the write request.

Table 6 shows the command structure.

Table 6. Write Request Cmd Format

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte . . .	Byte N+6
ID (0x03)	Start Address				Length (N)		Data (N Bytes)		

4.2.4 Commit Request

The commit request is the final step in writing an image to the MC1322x internal FLASH. It writes the “Length” and “Secure” option to the FLASH header, thus marking the internal FLASH as bootable. Checks will be made for the following:

- A valid Length for the image - “Length” needs to be smaller than the FLASH available size. The FLASH available size is the total FLASH size minus the last 4 kbyte sector (used for production data) and the internal FLASH header size.
- “Secure” field must contain one of two values (these values are defined in the `Engine.h` file.):
  - engSecured_c (0xC3) - causes “SECU” to be written to the Signature field
  - engUnsecured_c (0x3C) - causes “OKOK” to be written to the Signature field

A confirm message is sent to the host by the MC1322x after the execution of the commit request.

Table 7 shows the command structure.

Table 7. Commit Request Cmd Format

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
ID (0x04)	Length (N)				Secure

4.2.5 Erase Request

The erase request triggers a MC1322x internal FLASH erase (clear) function.

NOTE

- The MC1322x internal serial FLASH is a 128 kbyte device comprised of 32 uniform 4 kbyte sectors

- The last or top 4 kbyte sector of FLASH is reserved for production data and cannot be erased.

The command execution erases the 4Kbyte sector to which the “Address” belongs. Exceptions include:

- If “Address” is equal to 0xFFFFFFFF, the entire internal FLASH is erased (excluding the reserved top sector).
- If the address is invalid (not within the FLASH address range), an error is returned in the confirm message that is sent to the host by the MC1322x after the execution of the erase request.

A confirm message is sent to the host by the MC1322x after the execution of the erase request.

Table 8 shows the command structure.

Table 8. Erase Request Cmd Format

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
ID (0x05)	Address			

4.2.6 Confirm Message

The confirm message is sent by the MC1322x to the host after executing one of the above enumerated commands. It contains only the “Status” of the previously executed command and the “ID” denoting the message identity.

The possible values of the “Status” field (found in the ENGCmdStatus_t enumeration of the `Engine.h` file):

Table 9. Confirm Message Status Field Values

Name	Value	Description
gEngValidReq_c	0x00	Valid request command
gEngInvalidReq_c	0x01	Invalid request command
gEngSuccessOp_c	0x02	Request command success
gEngWriteError_c	0x03	Write request command error
gEngReadError_c	0x04	Read request command error
gEngCRCError_c	0x05	Request command CRC error
gEngCommError_c	0x06	Request command communication error
gEngExecError_c	0x07	Request command execution error

Table 10 shows the Confirm command structure.

Table 10. Confirm Message Cmd Format

Byte 0	Byte 1
ID (0xF0)	Status

4.3 SSL Application Modules

The SSL application contains three modules that implement the following functionality needed by the application:

- Communication with the host
- Processing commands
- FLASH read
- FLASH write
- FLASH erase

Each module is presented in the following sections.

4.3.1 UART Module

The UART module is implemented by the `UART.h` and `UART.c` files, and covers the communication needs of the application. See the `UART.h` file for the interface exposed to the rest of the application modules, along with detailed explanation for usage, and `UART.c` for implementation details.

Functions available to the application:

- *UART_Init()*: configures the GPIO for UART1 usage. Configures the UART1 for host communication at 115200 baud rate from the 24M Hz system clock, 8-N-1.
- *UART_ReceiveCmd()*: receives a command on the UART, formatted as described in Section 4.1 of this document. It waits for the SOF, receives the length and the command, computes the CRC and verifies it against the received CRC.
- *UART_SendCmd()*:Transmits a command on the UART, formatted as described in Section 4.1 of this document. The command is accompanied by the start of frame delimiter, the length of the command, the command itself and a very simple 8-bit CRC.
- *UART_SendRawData()*: transmits a raw data buffer over the UART.

4.3.2 NVM Module

The Non Volatile Memory (NVM) module is implemented by the `NVM.h` and `NVM.c` files. It allows the application to read, write and erase the FLASH memory present on the MC1322x. Macros for start and stop of the MC1322x onboard NVM voltage regulator are also present. See the `NVM.h` file for the interface exposed to the rest of the application modules, along with detailed explanation for usage, and `NVM.c` for implementation details.

Main functionality available to the application:

- *NVM_Read ()*: read a specified number of bytes, starting at a specified address, in a buffer provided by the application.
- *NVM_Write()*: write a specified number of bytes, starting from a specified address, from a buffer provided by the application. The data written are verified, and an error is returned at the first write error encountered.
- *NVM_EraseSector()*: erase a specified sector.

- *NVM_BlankCheck()*: verify that a specified number of bytes, starting at a specified address, holds the value 0xFF. This means that the respective locations can be written successfully.
- *NVM_StartNvmRegulator()*: starts the NVM regulator. The bootstrap leaves the NVM regulator off, in order to save power. The NVM regulator needs to be turned on before the internal FLASH memory can be accessed.
- *NVM_StopNvmRegulator()*: stops the NVM regulator. This will save power, if the application no longer needs to access the internal FLASH memory.

4.3.3 Engine module

The Engine module is implemented by the `Engine.h` and `Engine.c` files. It receives, interprets, executes and confirms commands received from the host over the UART1 interface. It uses the UART module to receive the commands and send back to the host the responses and confirm messages. For access to the internal FLASH of the MC1322x, the NVM module is used. Important details of the communication protocol between the MC1322x and the host, described in Section 4.2 of this document, like command ID's, status codes, command formats, are defined in the `Engine.h` file. Implementation details can be found in the `Engine.c` file.

There is only one function exposed to the application:

- *ENG_Process()*: wait for one command to be received from the host. Execute the command and send back a response to the host. The function should be called for as many messages as need to be handled by the MC1322x.

4.4 SSL application processing

As presented previously of this document, the SSL application will be loaded by the bootstrap starting at the beginning of the RAM, and then immediately given control of the MC1322x. The application performs the following actions in its main function, implemented in the `Main.c` file:

5. Start the NVM regulator - The bootstrap leaves the NVM off, as a measure to lower the power consumption. Since the application uses the NVM module to access the internal FLASH, the NVM regulator must be turned on.

NOTE

In order to support boards that use the buck regulator option, the buck must be bypassed prior to starting the NVM regulator. This is accomplished by setting the buckBypassEn bit in the CRM_VREG_CNTL Register to 1, then waiting at least 700us. The 1.8V NVM regulator can then be enabled.

6. Configure UART1 for usage - This is done through a call to the UART module initialization function.
7. Signal the host that the SSL application is ready to receive commands from the host - This is done by sending the "READY" string, unformatted. This is the single place where an unformatted message between the host and MC1322x is used through the application.
8. Enter an infinite loop where messages from the host are processed - This is done by calling the message processing function exposed by the Engine module in a *while(1)* infinite loop.

The application shall be always terminated by the host through a reset. If all the steps needed to load an executable image into the MC1322x internal FLASH have been executed, the reset should cause the bootstrap to load and start executing the new image from the internal FLASH.

5 Console Loader (CL)

Once the FLASH has been erased enabling boot from UART1, an entity is required that plays the role of the host. As described previously, the host must do the following:

- Create the needed conditions for the MC1322x to boot from UART1
- Load the SSL application binary through UART1
- Transmit the needed commands to the SSL for loading the desired FLASH image
- Correctly write the FLASH header so that a subsequent reset will have the MC1322x booting the image loaded into FLASH

The most convenient environment to demonstrate all these is the PC, as many MC1322x development boards offered by Freescale can be connected to a PC through a USB connection. The USB connection is mapped to a virtual COM port on the PC and to UART1 on the MC1322x development board.

The CL PC application is able to perform all these actions as a host to load a binary image that the user specifies to the MC1322x internal FLASH.

The CL has a command line interface that is detailed in Section 3.2.3, “Run the Tools (Load SSL and Update FLASH)”, list Item #3.

The following section details the capabilities and implementation details of the CL.

5.1 CL UART Interface and Commands

The UART interface and command formats used by the CL are identical to those used by the SSL as shown in Section 4.1, “SSL UART Command Interface and Section 4.2, “SSL UART Commands.

5.2 CL Modules

The CL has a number of modules that implement the needed application functionality:

- Communication with the MC1322x
- Commands formation and processing.

5.2.1 UART Module

The UART module is implemented by the `UART.h` and `UART.cpp` files, and covers the communication needs of the application. See the `UART.h` file for the interface exposed to the rest of the application modules, along with detailed explanation for usage, and `UART.cpp` for implementation details.

Functions available to the application:

- *UART_OpenCom()*: tries to open and configure a specified PC COM port, for a specified baudrate, in 8-N-1 mode. If successful, it will return a handle that can be used to read and write data to the COM port.
- *UART_SendCmd()*: transmits a command on the specified COM port, properly formatted as required by the host to MC1322x interface. The command is accompanied by the start of frame delimiter, the length of the command, the command itself and a very simple 8-bit CRC.
- *UART_ReceiveCmd()*: receives a command on the specified COM port, in a specified application buffer. It waits for the SOF, receives the length and the command, computes the CRC and verifies it against the received CRC.

5.2.2 Engine Module

The Engine module is implemented by the `Engine.h` and `Engine.cpp` files. It will create and send commands acting as the host, wait for the confirm messages of the execution on the MC1322x, and report the result to the caller. It uses the UART module to send the commands and receive the confirm messages from the SSL.

See the `Engine.h` file for the interface exposed to the application along with detailed explanation for usage, and `Engine.cpp` for implementation details.

NOTE

The command definitions and status codes from the Engine module of the CL application must be maintained in synch with the Engine module of the SSL application. Compiler peculiarities, related to structures packing and the enum base type in Visual C++, prevent the interface definition file from being shared between the CL and SSL applications.

Functions available to the application:

- *ENG_Erase()*: send the command to erase a sector or all sectors of the MC1322x. The last sector of the internal FLASH of the MC1322x holds production data and is never erased.
- *ENG_Write()*: send the command to write data from application-provided data into the internal FLASH of the MC1322x. All of the written bytes in FLASH are read back and verified by the SSL before returning the confirm message for the write operation.
- *ENG_Read()*: send the command to read data from the internal FLASH of the MC1322x in an application provided buffer.
- *ENG_Commit()*: commit the image written to the internal FLASH of the MC1322x by writing the FLASH header with the image descriptor. The length and security option is provided by the application.

5.3 CL Application Processing

The main function of the CL application is *_tmain()* and resides in the `ConsoleLoader.cpp` file. All the processing that will end up in an image being loaded into the MC1322x internal FLASH is executed in this function in the following order:

1. Parse the input arguments for the executable, and open all the needed handles in the process for the files accessed in the application and the COM port - This is accomplished with a call to the *ParseArguments()* function. If the function returns success, move to the next step.
2. Try to connect the MC1322x coming out of reset. - This is done by calling the *Connect()* function that sends 0x00 for baudrate detection and then waits for the "CONNECT" string from the MC1322x. The process is repeated a number of times, waiting for the user to reset the MC1322x. On success, it moves to the next step, as the MC1322x is ready to receive a binary image through UART1.
3. Downloads the SSL image to MC1322x through UART1, then waits the “READY” string from the SSL executing on the MC1322x. - This is done by the *DownloadSSL()* function.
4. Downloads the user specified binary image to the MC1322x internal FLASH by calling the *DownloadBin()* function - At this point, the SSL is running on the MC1322x waiting for commands from the host to execute on the MC1322x. The binary image will be partitioned into data packets of the maximum supported size by the SSL and sent to the SSL to be written into FLASH. After the file is transmitted, the binary is committed by writing the MC1322x FLASH header indicating the presence in the FLASH of a valid executable. This is the last step of the application.

As a general approach, if at any step an error occurs, the application prints a relevant error message to the user and the application help text, then it closes all opened handles and exits.

6 Compiling the Applications

To compile the SSL application, IAR EWARM v5.20 or above is required. The user needs to open the following project file:

`..\SSL\ssl.eww`

By default, the “Release” configuration is selected. This can be changed from the top of the “Workspace” window.

The SSL application can be built by pressing ‘F7” or by selecting from the menu “Project->Make”. Depending on the selected project configuration, the SSL binary is found in one of two locations:

- For the “Release” configuration, in `..\SSL\Release\Exe\ssl.bin`
- For the “Debug” configuration, in `..\SSL\Debug\Exe\ssl.bin`

To compile the ConsoleLoader application, Visual C++ 2003 or higher is required. The user needs to open the following project file:

`..\ConsoleLoader\ConsoleLoader.sln`

The “Release” configuration is selected by default. The “Debug” configuration can be selected using the “Configuration Manager”, opened through the menu entry “Build->Configuration Manager”.

Depending on the selected project configuration, the ConsoleLoader executable is found in one of two locations:

- For the “Release” configuration, in `..\ConsoleLoader\Release\ConsoleLoader.exe`
- For the “Debug” configuration, in `..\ConsoleLoader\Debug\ConsoleLoader.exe`

Before running the `ConsoleLoader.exe` application, make sure that the `ssl.bin` file is copied in the same directory as the `ConsoleLoader.exe`.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only*:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3860
Rev. 1.0
02/2010

freescale™
semiconductor