Engineer-to-Engineer Note EE-381

Technical notes on using Analog Devices products and development tools
Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

Using the ADuCM302x Processor Boot Kernel

Contributed by Nabeel Shah

Rev 1 – February 19, 2016

Introduction

The ADuCM302x processor features integrated flash memory that contains the user application code (*user space*) and a dedicated 2 KB bank of memory called *info space*, arranged as shown in Table 1.

<table>
<tr><td rowspan="3">Info Space
(2KB)</td><td>0x4 07FF</td><td>Part Information</td></tr>
<tr><td></td><td rowspan="2">Boot Kernel</td></tr>
<tr><td>0x4 0000</td></tr>
<tr><td rowspan="3">User Space
(up to 256 KB)</td><td>0x3 FFFF</td><td rowspan="3">User Application Code</td></tr>
<tr><td></td></tr>
<tr><td>0x0 0000</td></tr>
</table>

Table 1. Flash Info Memory Space

Some parts feature 128 KB of user space. Please refer to the *ADuCM3027/ADuCM3029 Ultra Low-Power ARM® Cortex®-M3 MCU with Integrated Power Management Datasheet*[1] for details.

As can be seen, the info space block is further broken down into the *boot kernel*, residing in the upper 2 KB of flash memory, and the product *part information*. The boot kernel is responsible for implementing a secure environment, where user application code can optionally be read- and/or write-protected, and is used to execute the application from flash memory upon reset. It also provides a mechanism to upgrade the firmware through a UART downloader.

This EE-note details the info space region of the on-chip flash memory and describes both the boot process and the means of utilizing the UART downloader to perform field upgrades to the processor firmware.

Copyright 2016, Analog Devices, Inc. All rights reserved. Analog Devices assumes no responsibility for customer product design or the use or application of customers' products or for any infringements of patents or rights of others which may result from Analog Devices assistance. All trademarks and logos are property of their respective holders. Information furnished by Analog Devices Applications and Development Tools Engineers is believed to be accurate and reliable, however no responsibility is assumed by Analog Devices regarding the technical accuracy and topicality of the content provided in all Analog Devices' Engineer-to-Engineer Notes.

Boot Kernel Overview

The boot kernel jumps to the user application after doing certain checks, including the CRC integrity of the user application, or enters the UART downloader mode to upgrade the user application in flash memory (depending on the `SYS_BMODE0` boot mode pin state at reset).

The boot kernel supports in-field updates to the user application through the UART port. For security reasons, the boot kernel itself does not provide the flash programming feature; however, it allows the firmware update code, which has flash driver code for updating the user flash, to be downloaded to the device over the UART port. This code is referred to as a *Second Stage Loader* (SSL) and is run out of SRAM. The SSL needs to be authenticated before it can be provided run access. The security scheme implemented is discussed in the following sections, describing the critical part of the kernel to provide the secure environment in which the user code can be read/write protected, allowing for IP security.

The serial download capability allows developers to reprogram the part while it is soldered directly onto the target system, avoiding the need for an external device programmer and removing the need to swap the device out of the system. The serial download feature also enables system upgrades to be performed in the field, provided the hardware infrastructure involving the `SYS_BMODE0` pin and the UART port are implemented on the target board.

Configuring Security Options

The boot kernel provides the flexibility to configure the security options of the device by allowing the user to program certain keys and parameters in predefined locations in page 0 of the user flash memory. The kernel provides the user code security and integrity, which depends on the number of user-defined parameters in the first page of the user flash memory. Table 2 summarizes the list of keys/parameters and their locations in the user flash memory.

Address Range	Size	Description
0x0000_0180 - 0x0000_018F	128 bits (16 bytes)	Read protection key hash
0x0000_0190 - 0x0000_0193	32 bits (4 bytes)	CRC of read protection key hash
0x0000_0194 - 0x0000_0197	32 bits (4 bytes)	Length of user boot loader or entire user code (used for CRC verification before boot)
0x0000_0198 - 0x0000_019B	32-bit word	In-circuit write protect if set to `NOWR`
0x0000_019C- 0x0000_019F	32-bit word	CPU write protection of individual flash blocks

Table 2: List of Keys and Parameters

Read Protection Key Hash

The 128-bit read protection key hash should be programmed by the user at address 0x00000180 in the first page of user flash memory. The value depends on what kind of security is desired in the system, as it defines the read accessibility to the device. The key hash is used for defining the state of the serial wire debugger (SWD), as well as the access permission of the SSL downloaded for upgrades via the UART:

- The reset state of the flash memory of all logic high memory cells (along with a valid key hash CRC) indicates that the user does not desire read protection. In this case, the SWD interface is automatically enabled during booting.
- Any non-reset value results in the SWD getting locked; therefore, there will be no SWD access to the device.
- The key hash is the 128-bit truncated SHA-256 hash of the user key (which is 128 bits in length), which can be sent along with the SSL during the UART download phase. If the user key is valid and the hash of the received key matches the key hash stored, then the SSL runs with all the permissions. If it fails the key hash check, the SSL only has write permission to the user flash.

Key Hash CRC

The key hash has a 32-bit CRC checksum which is stored at address 0x00000190. The key hash is valid only if its associated 4-byte checksum is valid. The key hash has a separate key hash to protect it against flash tempering attacks. For all practical purposes, the user must ensure that a valid CRC for the key hash is stored along with the key hash itself.

In-Circuit Write Protect Key

The 32-bit in-circuit write protect key at address 0x00000198 of the user flash memory is used to prevent in-circuit programming of the device. If the user intends to disable in-circuit reprogramming, the hexadecimal value of the ASCII string “NoWr” (without the terminating null character) should be programmed to this address. In this case, SWD access to the device is locked, and the only way to update the device code is via the UART downloader.

In-circuit write protection should be used along with read protection (providing both read and write protection for the user). In-circuit write protection alone does not have any significance.

Write Protection

To prevent code from accidentally erasing and reprogramming critical flash memory blocks (such as the user code boot loader), pages can be locked. There is a hardware register in the flash controller used to disable programming of pages grouped into blocks. This register is not automatically loaded via hardware; rather, it is written via the kernel. The kernel reads the write protection word from the user flash address 0x0000019C and writes it to the write protection register in the flash controller. The user can write the appropriate word to this location, depending on the pages that are intended to be protected against accidental writes. The pages are protected in groups of four, with each bit in the 32-bit word corresponding to four continuous flash pages. Refer to the *ADuCM302x Mixed-Signal Control Processor with ARM Cortex-M3 and Low-Power Management Hardware Reference*[2] for more details.

User Code Length

There is a 32-bit value stored at flash memory address 0x00000194 which defines the CRC-protected user code length. The value programmed in this field defines the page number of the user flash memory up to which CRC protection is desired by the user. The value of N means CRC protection is desired from page 0 to page N of the flash memory, protecting total of N+1 pages.

Valid values for this field are 0 to 127 for the 256 KB ADuCM3029 processor and 0 to 63 for the 128 KB ADuCM3027 processor. Any value outside this range is treated as invalid and results in a CRC check failure.

User Code CRC

The user code CRC is stored at the end of page N. The CRC32 (MSB first) with a polynomial of 0x4C11DB7 is expected by the kernel. If the page number is N, the CRC is expected to reside at flash memory address (N*0x800) + 0x7FC. For example, if N= 5, a total of six pages are CRC-protected, and the CRC is stored at address 0x2FFC. There is an option to disable the CRC check by programming 0xFFFFFFFF to the expected CRC location. Once the kernel sees this value in the CRC location, it skips the CRC check.

Boot Code Flow

Based on the user-programmable parameters described above, we can now describe how the kernel operates. Figure 1 shows a flowchart of the boot code.

After reset, the boot kernel inspects all the parameters stored in page 0 of the user flash memory. If the user read protection key hash is not programmed (all FFs) and the key hash CRC is valid, it implies that the user has not requested the read protection. As such, the SWD is enabled; however, flash access may be protected, depending on the state of the user code CRC.

- If the CRC is valid, access to user flash memory is unrestricted.
- If the CRC is disabled by the user by programming 0xFFFFFFFF to the CRC location, access to user flash memory is also unrestricted.
- If the CRC is invalid, the user flash memory is protected with no read/write access allowed. Only the flash mass erase command is allowed. In this case, user code execution is not allowed.

Care should be taken to program a valid CRC or 0xFFFFFFFF in the defined CRC location, otherwise the user flash memory is read-protected by the kernel. In this case, flash-based applications will fail to load unless the user flash memory is mass erased.

If the user read protection key hash is programmed with a non-reset value (which means the user has turned on the read protection), or if the key hash CRC is invalid, the SWD is disabled by the kernel and SWD access to the device is not possible. This should be done only after product development is completed and SWD access is not intended in the field. However, in this case where read protection is enabled, the SWD is opened up only if CRC protection is enabled and the CRC has been corrupted, which allows for device recovery when the CRC is accidentally corrupted. In this case, the SWD is opened up, but the user flash memory is protected with no read/write or page erase accessibility (to maintain the user code confidentiality while allowing for device recovery). However, mass erase is still possible, which results in the user flash memory being open again (with read/write access).

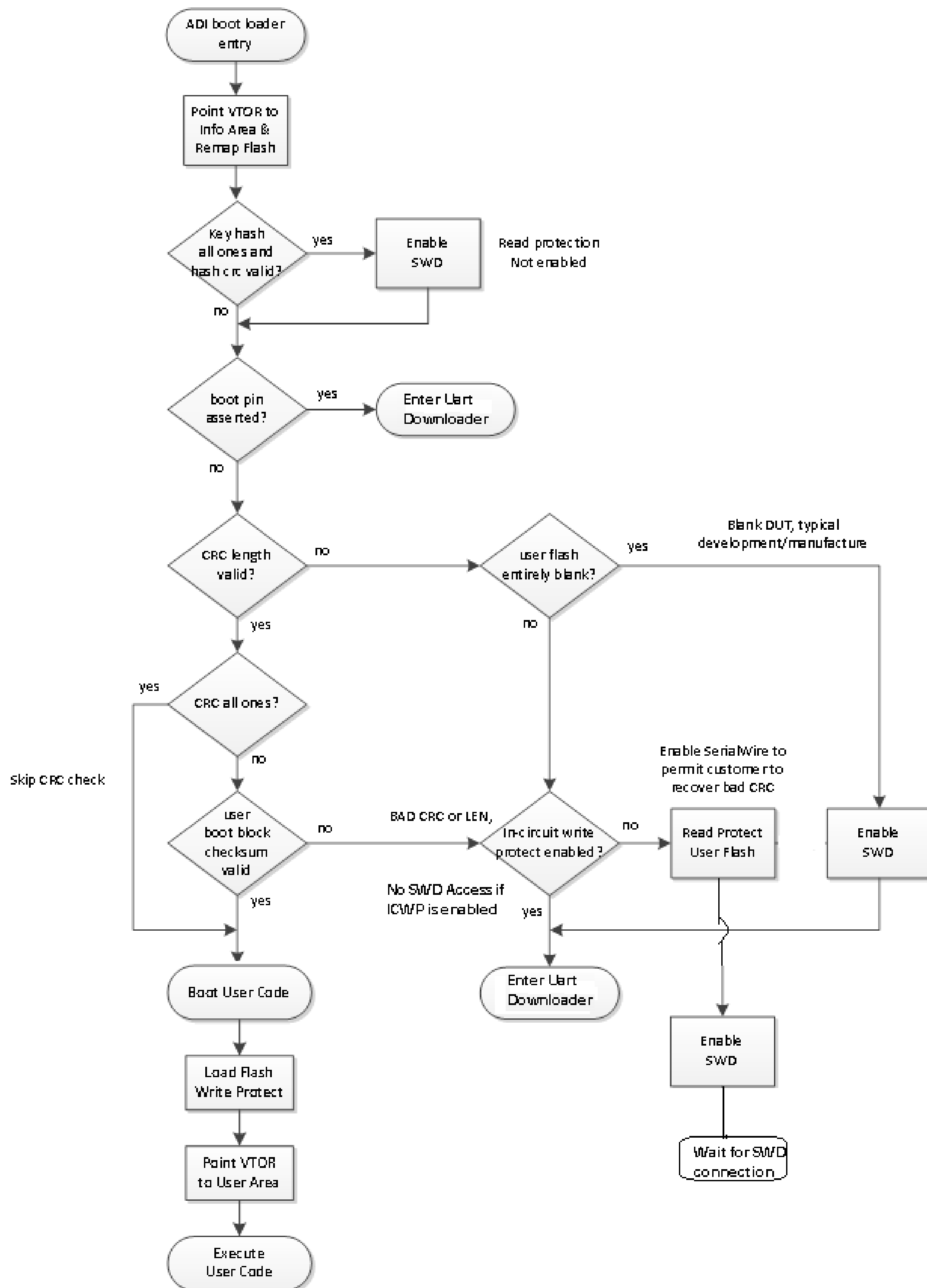

Figure 1: Boot Kernel Flowchart

The user flash memory (user space) is completely blank when shipped; therefore, none of the security keys/parameters are programmed. As such, most of the parameters like the key hash CRC and the user code length will have invalid values (all set to 0xFF). To handle this condition, the kernel performs a check of the user flash memory to identify if it is blank (completely not programmed). If the user flash is blank, the kernel skips all the checks and opens up the SWD. In addition to opening up the SWD, allowing users to connect to the part through SWD for their development, it also enters the UART downloader mode and awaits reception of the SSL.

Once the part is programmed via the SWD, the user flash memory is no longer blank, and the kernel relies on the state of the `SYS_BMODE0` pin to decide if the user code should be executed (after doing all the checks explained above) or if it should enter the UART downloader mode:

- If the `SYS_BMODE0` pin is asserted (low), the kernel enters the UART downloader and waits for the SSL to be downloaded.
- If the `SYS_BMODE0` pin is de-asserted (high), the kernel jumps to the user reset vector in the user flash memory after performing all the security checks.

The only case where the kernel enters the UART download mode without sampling the `SYS_BMODE0` pin is when the user flash is blank.

SSL code that is downloaded over the UART must be mapped to the SRAM. In UART downloader mode, the SSL is loaded to the SRAM and can have flash programming capabilities. The kernel authenticates the SSL and allows execution only if authentication is successful. This code will be responsible for downloading and upgrading the actual firmware (i.e., the user application) in the user flash memory. The kernel does not support direct updates to the user flash memory, so the SSL is required to perform such actions.

The kernel follows a specific protocol to download the SSL to the processor, which must be adhered to by the transmitting host. If the SSL follows the same packet protocol as the kernel, the host interface is simplified (i.e., communication with the kernel and the SSL is uniform). The details of the protocol are discussed in following sections.

UART Downloader

The ADuCM302x processor enters UART downloader mode if the `SYS_BMODE0` pin (`GPIO17`) is pulled low. If this condition is detected by the part at power-on or hard reset, the part enters serial download mode. In this mode, an on-chip loader routine in the kernel is initiated, which configures the device's UART port and, via a specific serial download protocol, communicates with a host to manage the firmware upgrade process. Figure 2 shows the UART downloader flow.

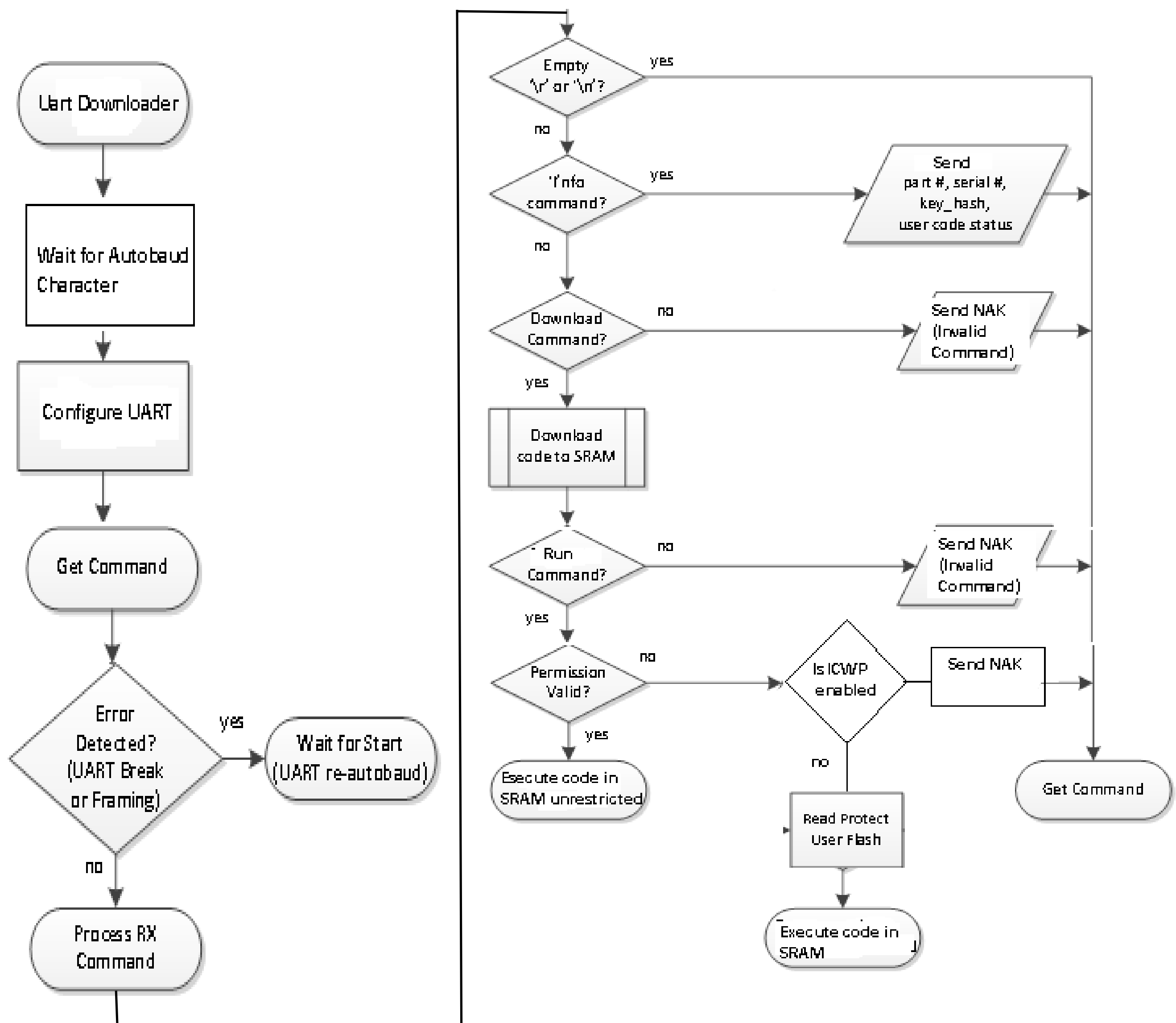

Figure 2: UART Downloader Flowchart

Protocol

Once the serial downloader is triggered by asserting the `SYS_BMODE0` pin, the kernel waits for the host to send a carriage return character (ASCII 0x0D, as shown in Figure 3) to initiate the UART autobaud process. The kernel makes use of the UART autobaud feature to detect the baud rate of the host and subsequently configure the UART port to transmit/receive at the host's baud rate with eight data bits and no parity. Due to the reset peripheral clock (PCLK) of 6.5 MHz, the UART can be configured by the kernel to support baud rates up to 230,400 bps. Baud rates above this have more error and may result in unreliable data transfer. However, after loading the SSL, higher baud rates are possible if the SSL increases the PCLK (up to 26 MHz) and performs a second autobaud detection via the UART.

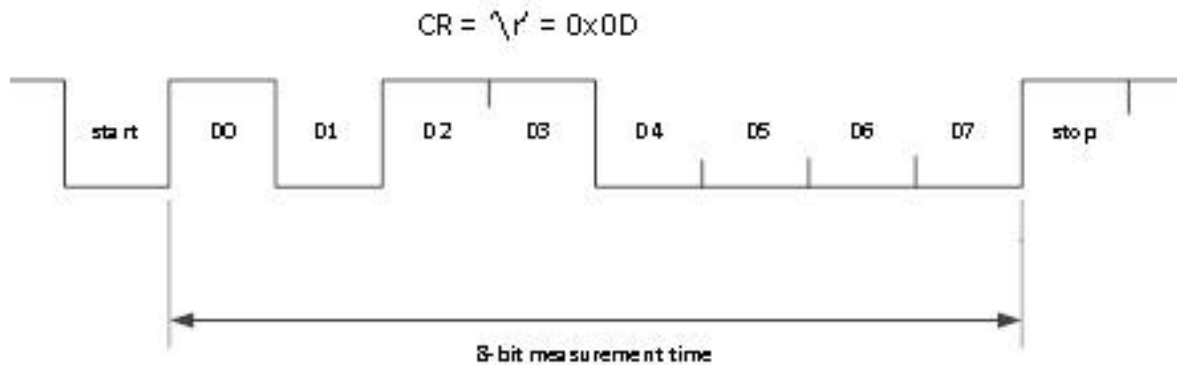

Figure 3. Autobaud Character

Upon receiving the autobaud character, the kernel calculates the required clock divisor values and configures the UART, at which point the kernel sends the part information as part of a 57-byte ID data packet, as shown in Table 3, to acknowledge that the autobaud detection process was successful.

Bytes	Content
1-15	Product identifier: "ADuCM302x" and six spaces (x = 7 or 9)
16-18	Hardware and firmware version numbers
19	User code blank ('X' = code to execute, '-' = blank)
20	User code checksum ('P' = checksum passed, 'F' = checksum failed)
21	Write protection enabled ('W' = disabled, '-' = enabled)
22	Read protection enabled ('R' = disabled, '-' = enabled)
23	Space
24-55	128-bit serial number, as a 32-digit upper-case hex number (e.g., 0123456789ABCDEF0123456789ABCDEF)
56	Line Feed
57	Carriage Return

Table 3. Autobaud Response

Packet Structure

In addition to indicating to the host that the processor is now ready to communicate, the autobaud acknowledgement also contains information about the part, the state of user flash memory, and security restrictions. After the autobaud acknowledgement, the data transfer itself can begin, as governed by the communications data transport packet format shown in Figure 4.

ID0	ID1	Number of Data Bytes	CMD	Value	Data	Checksum
0x07	0x0E	5 to 255	'W', 'R', or 'I'	h,u,m,l	xx	CS

Figure 4. UART Packet Structure

Packet Start ID Field (ID0/ID1)

The first field is the two-byte packet start ID field (ID0 and ID1), which is comprised of two start characters (0x07 and 0x0E, respectively). These bytes are constant and are used by the loader to detect the beginning of a valid data packet.

Number of Data Bytes Field

The next field is the total number of data bytes, which includes the 1-byte command (CMD), the 4-byte address (Value), and the remaining payload (Data). The minimum number of data bytes is five, which corresponds to a command and address only. The maximum number of data bytes is 255, supporting a command, address, and up to 250 bytes of data.

Command Function Field (CMD) – Data Byte 1

The command function field describes the function of the data packet. Three commands are supported by the kernel, represented in ASCII format:

- 'W' (0x57) - write command
- 'R' (0x52) - run command
- 'I' (0x45) - info command

Write Command

The write command packet shown in Figure 5 includes the number of data bytes (5 + n, where n is the payload size in bytes), the write command ('W'), the 32-bit start address to write to, and the n data bytes in the payload.

ID0	ID1	Number of Data Bytes	CMD	Value	Data	CS
0x07	0x0E	5 + n	'W' (0x57)	Start Address	n bytes	CS

Figure 5. Write Command Packet

When a write command packet is received by the kernel, the payload bytes are placed sequentially in the SRAM as they arrive, beginning at the start address. The kernel sends a NAK if the checksum is incorrect or if the address received is out of range. If the host receives a NAK from the loader, the download process should be aborted and restarted.

ANALOG DEVICES

Run Command

Once the host has transmitted all the data packets to the kernel, it can send a final packet instructing the kernel to start executing code. This is achieved by sending the run command packet, which is comprised of the run command ('R') and the 32-bit address to begin running from, as shown in Figure 6.

ID0	ID1	Number of Data Bytes	CMD	Value	CS
0x07	0x0E	5	'R' (0x52)	Start Address	CS

Figure 6. Run Command Packet

When the kernel receives a run command packet, it jumps to the address supplied in the packet only after the permission checks have passed.

Info Command

The info command packet shown in Figure 7 can be sent by the host at any time, and it is comprised of the command ('I') and a 32-bit address. Though the Value field is required for the packet to be properly received by the kernel, the content is irrelevant.

ID0	ID1	Number of Data Bytes	CMD	Value	CS
0x07	0x0E	5	'I' (0x52)	0xXXXXXXXX	CS

Figure 7. Info Command Packet

When the kernel receives the info command packet, it responds with the 57-byte ID packet from Table 3.

Value Field (Data Bytes 2-5)

The value field contains a 32-bit address (h, u, m, l) with the MSB in the h (Data Byte 2) location and the LSB in the l location (Data Byte 5). As described in the previous section:

- In a write command packet, value indicates the start address in memory to which the data payload will be written.
- In a run command packet, value indicates the address in SRAM where the SSL code begins.
- In an info command packet, value has no meaning.

Data Field (Data Bytes 6-255)

User code is downloaded a byte at a time, and the data field can contain a maximum of 250 bytes. The data is normally stripped out of the Intel Hex extended 16-byte record format, reassembled by the host, and then sent in packetized form using a series of write command packets to the ADuCM302x processor.

Checksum Field (CS)

The data packet checksum is written into the checksum field. This twos-complement checksum is calculated from the summation of the hexadecimal values spanning the number of bytes field to the end of the Data field. Thus, the 8-bit LSB of the sum of all the bytes in the packet from the number of data bytes field up to and including the checksum field should be 0.

Acknowledge of Command

The loader routine issues a NAK (0x07) as a negative response or an ACK (0x06) as a positive response to each data packet received.

A NAK is transmitted by the loader if it hits any of following conditions:

1. Receives an incorrect checksum.
2. UART Framing/Break error (it may not reach the host if the UART link is bad).
3. SRAM code verification fails.

If any of these conditions is met, it’s required to reset the target and restart the firmware upgrade process.

If none of these conditions is met, an ACK is transmitted.

Read Protection Key and Hashing

The read protection key can be used to allow access to the device during failure analysis. If the customer has read-protected the device and failure analysis of the current flash memory content is necessary, the SWD interface can be enabled by sending the key corresponding to the hash stored in the user flash memory. It is advised that the key be unique to the device and be based on the device’s unique identifier (serial number stored in info space).

A hash is stored in the user flash memory after the interrupt vectors. This is the hash of a secret customer key. It is highly advised that this key be unique to the device for security reasons, and the unlock key would only be valid for that one specific device. To maintain a unique key per device, there needs to be a device identifier to associate which key belongs to a particular device. To make key management simpler for the customer, it is advised to make the key a hash of a master secret and the device identifier. For example:

Read protection key = hash(master_secret || unique_device_identifier);
Key hash = hash(Read protection key)

When the kernel is in UART loader mode, it can accept the read protection key. The boot-loader will perform a hash of the read protection key and compare it to the stored key hash. Upon a successful match, the boot-loader will permit the downloaded SSL code in SRAM to be executed with all the permissions enabled. If the key hash check fails, then the kernel checks the ICWP key in the user flash memory. If ICWP is turned off by the user by programming any value to address 0x00000198 other than the hex equivalent of the ASCII ‘NoWr’, then the SSL is allowed to run after protecting the flash against read/write accesses. In this case, the SSL should first issue a mass erase of the user flash memory before attempting to perform any access to the user flash memory space. If ICWP is also enabled by the user, then the SSL is not granted permission to run unless the key-hash authentication passes.

The 128-bit read protection key is passed as a part of SRAM code. It should be stored in big-endian format in the SRAM as a data payload starting at address 0x20000180 and must be oriented is a specific fashion in memory for the kernel to parse it correctly. Specifically, if the read protection key is represented as ABCDEFGHIJKLMNOP, where each letter represents one byte (with A being the first byte and P being the last byte), the required arrangement of the bytes in memory is shown in Table 4.

Address	Byte0	Byte1	Byte2	Byte3
0x20000180	D	C	B	A
0x20000184	H	G	F	E
0x20000188	L	K	J	I
0x2000018C	P	O	N	M

Table 4. Read Protection in SRAM

For example, if the read protection key is 0x000102030405060708090A0B0C0D0E0F, then Table 5 shows how the memory must be written.

Address	Byte0	Byte1	Byte2	Byte3
0x20000180	0x03	0x02	0x01	0x00
0x20000184	0x07	0x06	0x05	0x04
0x20000188	0x0B	0x0A	0x09	0x08
0x2000018C	0x0F	0x0E	0x0D	0x0C

Table 5. Example Read Protection Key in SRAM

The kernel computes the SHA-256 hash of this key, truncates it to a 128-bit hash, and then compares it to the hash stored in page 0 of the user flash memory at address 0x00000180. The user must store the 128-bit truncated hash of the key to the flash memory using a similar pattern. The SHA-256 hash for the example key above is 0xBE45CB2605BF36BEBDE684841A28F0FD43C69850A3DCE5FEDBA69928EE3A8991, which means the 128-bit truncated hash that needs to be stored properly to the user flash memory space is 0x43C69850A3DCE5FEDBA69928EE3A8991, arranged as shown in Table 6.

Address	Byte0	Byte1	Byte2	Byte3
0x00000180	0x50	0x98	0xC6	0x43
0x00000184	0xFE	0xE5	0xDC	0xA3
0x00000188	0x28	0x99	0xA6	0xDB
0x0000018C	0x91	0x89	0x3A	0xEE

Table 6. Example Read Protection Key Hash in Flash Memory

The CRC32 of the key hash is calculated with a polynomial of 0x04C11DB7 and a seed value of 0xFFFFFFFF, and it is stored in LSB format in the flash memory space at address 0x00000190.

Memory Configuration

Table 7 summarizes the different keys and parameters stored in page 0 of the user flash memory and their addresses, along with the values that are programmed to page 0 when creating a project with the default startup file.

Content	Address Range		Size (bytes)	Section Name	Default Content
	Start Address	End Address			
Vector Table	0x0000_0000	0x0000_017F	384	.intvec	Vector table
Read protection key hash	0x0000_0180	0x0000_018F	16	ReadProtection KeyHash	0xFFFFFFFF 0xFFFFFFFF 0xFFFFFFFF 0xFFFFFFFF
CRC of read protection key	0x0000_0190	0x0000_0193	4	CRC_ReadProtection KeyHash	0xA79C3203
Number of pages CRC to be computed	0x0000_0194	0x0000_0197	4	NumCRCPages	0
Checksum	0x0000_07FC	0x0000_07FF	4	Checksum	Checksum of 0-0x7FB (if enabled in the tools by the user)
Page-0 user memory	0x0000_01A0	0x0000_07FC	1628	Page0_region	User application

Table 7. Page 0 Memory Configuration

Handling CRC in Tools

The CRC should be calculated from part of the application image which will be loaded into the first several pages of the flash memory. The page number of the last page involved in the CRC calculation should be stored at address 0x194 as a 32-bit integer. For example, if only page 0 is involved in the CRC

calculation, the value of 0x00 should be stored at address 0x194. If the CRC is calculated for the first 3 pages, the value must be 0x02.

When the CRC is calculated, the last four bytes of the last page included in the CRC calculation are excluded. These four bytes are used for storing the CRC value itself. For example, if the last page number is 0, the CRC is calculated from address 0x000 up to and including address 0x7FB. The tool stores the calculated CRC value at address 0x7FC as a 32-bit integer.

The standard of the CRC calculation is CRC32 with a polynomial of 0x04C11DB7, stored in MSB-first format, with an initial value of 0xFFFFFFFF. The unit size is 32-bit, which means the tool needs to read 32 bits at the time from the image when calculating the CRC.

Linker Option→Checksum

There is a "Checksum" tab under the "Linker" settings in the IAR tools, which can be used to generate the CRC of the user application code. In order to store the correct CRC, the following settings must be used:

- Check "Fill unused code memory"
- Set the "Fill pattern→End address" to 0x7FB (changes depending on page number)
- Check "Generate checksum"
- Select "4 bytes" from the "Checksum size" pull-down
- Set "Alignment" to 4 (bytes)
- Select "CRC32" from the "Algorithm" pull-down
- Set the "Initial value" to 0xFFFFFFFF and uncheck "Use as input"
- Select "32-bit" from the "Checksum unit size" pull-down

Figure 8 is a screen capture of the Linker settings Checksum tab, showing the above settings.

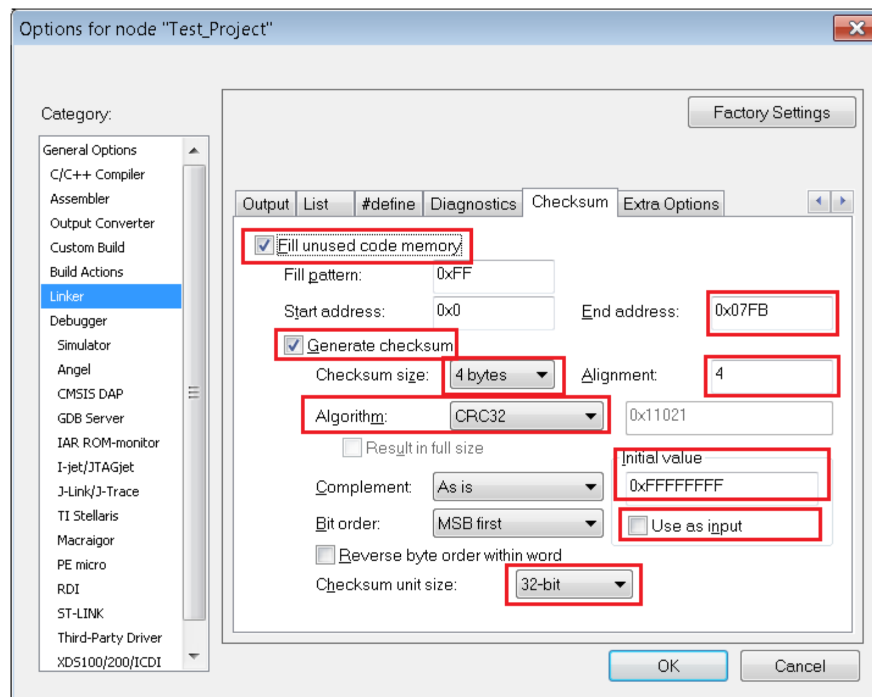

Figure 8. Checksum Settings

CrossCore® Serial Flash Programmer

The CrossCore Serial Flash Programmer (CCSFP) is a PC-based host utility provided by Analog Devices, which can be used to upgrade the user code over the UART port. It provides a graphical user interface to provide the following options for the UART upgrade:

1. Target processor
2. UART PC port number
3. Baud rate
4. SSL hex file to be used for the upgrade
5. User application hex file to be upgraded
6. Key to authenticate the SSL

Figure 9 shows the GUI for the CCSFP. The user has to provide the SSL in the "Second stage kernel" dialog box, which is first downloaded into the SRAM of the processor and then is executed before the user application in the "File to download" dialog box is sent to flash, based on the authentication. The 128-bit key for the authentication can be entered in the "Key" dialog box.

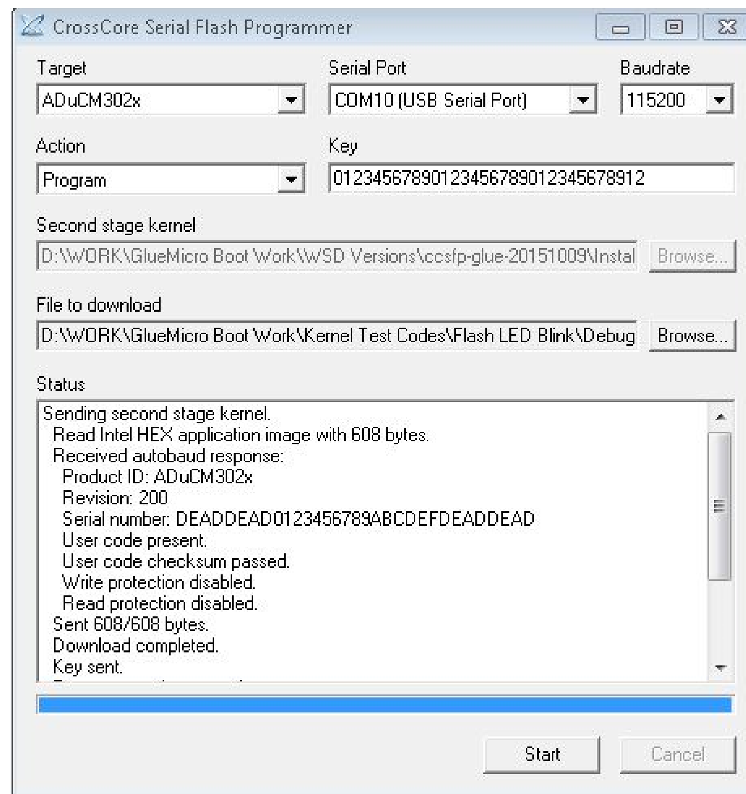

Figure 9. CrossCore Serial Flash Programmer GUI

The "Status" window shows the state of the UART download process and also displays the part-related information and status of the commands as returned by the kernel. As can be seen, the status window shows the part information sent by the kernel, showing the product ID, serial number, and user code status.

Once the SSL is downloaded, as indicated by the "Download completed" message displayed in the "Status" window, the SSL is then authenticated by the kernel and the actual user application is sent.

Figure 10 shows the SSL executing on the part, receiving the user application, and writing it to the user flash memory space.

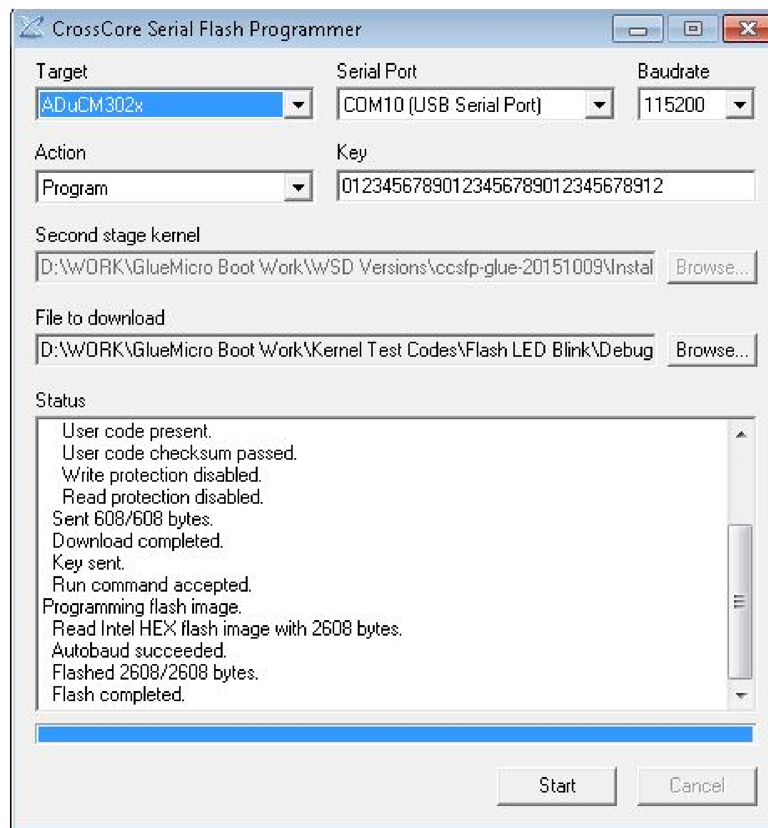

Figure 10. User Application Code Being Written by the SLL to User Flash Memory via the CCSFP

References

[1] *ADuCM3027/ADuCM3029 Ultra Low-Power ARM Cortex-M3 MCU with Integrated Power Management Datasheet*. Rev PrF, February 2016. Analog Devices, Inc.

[2] *ADuCM302x Mixed-Signal Control Processor with ARM® Cortex®-M3 and Low-Power Management Hardware Reference Revision 0.1, November 2014 Analog Devices, Inc.*

Document History

Revision	Description
Rev 1 – Feb 19, 2016 by Nabeel Shah	Initial Release.