Release Notes for CrossCore Serial Flash Programmer 1.0.0

http://www.analog.com
processor.tools.support@analog.com

Contents

1 Introduction 3
2 Technical support 4
3 File formats 5
4 Supported parts 6
4.1 ADSP-CM40x 6
4.1.1 Example 6
4.1.2 Security 7
4.1.3 Recovering a locked part 8
5 Command line invocation 9

1 Introduction

The CrossCore Serial Flash Programmer (CCSFP) is a Windows utility for programming the flash memory of supported Analog Devices processors via a UART serial port.

The installer allows the user to choose the install location. By default, CCSFP is installed at *C:\Analog Devices\CrossCore Serial Flash Programmer 1.0.0*. A start menu entry is created under *Analog Devices\CrossCore Tools*. Settings are stored in *Analog Devices\Cross Core Serial Flash Programmer.ini* within the current user's local settings directory.

2 Technical support

You can reach Analog Devices software and tools technical support in the following ways:

- Post your questions in the software and development tools support community at EngineerZone®.
- E-mail your questions to processor.tools.support@analog.com.
- E-mail your questions about processors and processor applications to processor.support@analog.com.
- Submit your questions to technical support directly via http://www.analog.com/support.
- Contact your Analog Devices sales office or authorized distributor.

3 File formats

CCSFP supports Intel Hex (*.hex*) and plain binary (*.bin*) as flash image file formats. In *IAR Embedded Workbench* projects, generation of an Intel Hex file or a binary file can be enabled on the *Output Converter* pane of the project options.

For Intel Hex files, CCSFP checks addresses covered in the file against the flash address range of the target processor. Any other files are treated as binary files assumed to start at the flash start address, while their size is checked against the target flash size. The use of Intel Hex files is recommended for the additional address checking.

CCSFP does not support ELF executables. If given one as an input file, it will treat it as a binary file and write its entire content including bookkeeping information such as section headers and symbol table to flash, so the code in the ELF will not run as expected.

4 Supported parts

CCSFP 1.0.0 supports ADSP-CM40x parts from revision G onwards.

4.1 ADSP-CM40x

Flash programming is not directly supported by the ADSP-CM40x boot ROM. Hence, CCSFP works by first downloading a second stage kernel to RAM, which then receives and programs the actual flash image.

This requires the UART slave boot mode to be active, which is boot mode number 3. The boot mode is selected by the two BMODE pins, which are connected to a rotary switch on ADSP-CM40x EZ-KIT boards. The boot mode becomes active on reset.

In UART slave boot mode, the boot ROM receives a boot loader stream via UART0, which is connected to the RS232 port on EZ-KIT boards. The *ADSP-CM40x-FlashProgrammer.ldr* file in the CCSFP install is the loader stream containing the second stage kernel. Its source code can be found in the *src\ADSP-CM40x* directory.

Booting an application that has been programmed into flash requires boot mode 1 to be selected.

4.1.1 Example

The *examples* directory of the CCSFP install contains example flash application images for ADSP-CM403F and ADSP-CM408F EZ-KIT boards: *ADSP-CM403F-Button.hex* and *ADSP-CM408F-Button.hex*. These are builds of the Button_LED_GPIO example from the *ADSP-CM40x Enablement Software Package* (ESP) version 2.1.0.

The following instructions assume that the flash is blank or that it contains a valid image without security features enabled.

1. Connect the host to the EZ-KIT's RS232 port, either directly from a serial port or via a USB-to-Serial adapter.
2. Select boot mode 3 on the board.
3. Reset the board.
4. Open CrossCore Serial Flash Programmer.
5. Select *ADSP-CM40x* as the target and the appropriate serial port.
6. Select the *Program* action.
7. Click the *Browse* button for the *File to download*, and select the appropriate *.hex* file for the connected EZ-KIT from the *examples* directory of the CCSFP install.

8. Click *Start*.
    a. If the autobaud fails when trying to send the second stage kernel, retry steps 1 to 3.
    b. If the autobaud fails when trying to program the flash, the part is probably locked. See the Recovering a locked part section below.
9. Once the operation completes, select boot mode 1 on the board and reset.
10. The example application should now be running. Pressing the *PB1* or *PB2* buttons should toggle the LEDs adjacent to them.

4.1.2 Security

ADSP-CM40x parts implement a security scheme intended to prevent unauthorized reading of the flash content. This uses a 128-bit key that is part of a security header at the start of flash at address 0x1800_0000.

In the ESP example projects, the key is defined in header *inc\adi_ecc.h*. That header and the ESP release notes have further details on this. Initially, the header contains the so-called default debug key, which enables debugging and disables security:

```
#define ADI_SECURITY_USER_KEY0      0xa4b8e4a5
#define ADI_SECURITY_USER_KEY1      0xd2041dd7
#define ADI_SECURITY_USER_KEY2      0x18839df8
#define ADI_SECURITY_USER_KEY3      0x8392c1fe
```

However, if an application image with a valid secure header containing a key other than the default debug key has been written to flash, then that key needs to be provided to allow programming through CCSFP. (Note that changing the key in *inc\adi_ecc.h* header requires a utility called *EccGen.exe* to be run on it to update the checksum fields in the header. Otherwise the security header becomes invalid. Further details are in the header.)

The key has to be entered as a 32-digit hexadecimal number into CCSFP's *Key* field, in the same order as the definitions in the header, and without a *0x* or other prefix. For example, a key defined as follows would need to be entered as *00112233445566778899aabbccddeeff*:

```
#define ADI_SECURITY_USER_KEY0      0x00112233
#define ADI_SECURITY_USER_KEY1      0x44556677
#define ADI_SECURITY_USER_KEY2      0x8899aabb
#define ADI_SECURITY_USER_KEY3      0xccddeeff
```

(Due to little-endian byte order within words, in flash that key would be stored in order *33 22 11 00 77 66 55 44 bb aa 99 88 ff ee dd cc*.)

4.1.3 Recovering a locked part

If a part is locked due to an invalid security header or a valid security header with a non-default key, it can be recovered by using the *Erase locked flash* action after resetting in boot mode 3. This sends a command for erasing the entire flash. The command is ignored if the flash is not locked.

The erase operation may take up to 4 minutes, although on EZ-KITs it typically takes less than 30 seconds. Completion is not reported back to the host, but is indicated by the SYS_FAULT pin connected to a red LED on EZ-KITs. Following the erase, the part has to be reset again to enable programming.

5 Command line invocation

CCSFP can also be invoked from a command line, as follows:

```
Usage: ccsfp [options] [file]

Options:
-a|-auto
        Enable unattended mode.
-b|-baud <number>
        Select baud rate.
-k|-key <key>
        Provide 32-digit hexadecimal unlock key.
-p|-port <name>
        Select serial (e.g. COM1).
-t|-target <name>
        Select target (as defined in ADIChip.ini).
-x|-action program|erase|load
        Select action. Defaults to 'program'.
-v|-version
        Print version information.
-h|-help
        Print this help message.
```

In unattended mode, the file argument has to be provided. The download starts automatically and the application exits as soon as the download finishes or fails. Settings not provided on the command line are read from the settings file. The exit code is 0 in case of a successful download, 1 in case of failure, and 2 if invalid command line arguments are supplied.

For example:

```
> ccsfp -a -p COM0 -b 115200 ADSP-CM403F-Button.hex
```

Note that since ccsfp.exe is a Windows subsystem program rather than a Console one, the Windows Command Prompt (cmd.exe) will automatically launch it as a background process, which means that the process cannot print to the console and that cmd.exe does not wait for it to finish. This can be worked around by piping output to the *more* command:

```
> ccsfp -a -p COM0 -b 115200 ADSP-CM403F-Button.hex | more
```

No such workaround is needed when running from a Cygwin terminal.