Managing firmware integrity in XMC™

XMC1000 and XMC4000 family devices

About this document

Scope and purpose

This document introduces methods for managing flash integrity, and gives examples using a CRC-32 method for the XMC1000 and XMC4000 family devices.

Intended audience

The document is intended for users of XMC1000 and XMC4000 family devices.

Table of contents

About this document **1**
Table of contents **1**
1 Flash integrity and management overview **2**
1.1 Cyclic Redundancy Check (CRC) method 2
2 Conditioning the application code **5**
2.1 Methodology 5
2.1.2 Generating an SRecord 6
2.1.3 Formatting the SRecord 8
2.1.4 Using XMC™ flasher to apply the CRC 9
3 Example application **10**
3.1 XMC1000 family devices 10
3.2 XMC4000 family devices 14
Revision history **17**

1 Flash integrity and management overview

Managing the integrity of the microcontroller flash is critical for confirming safe programming of the flash; at time of manufacturing, with every firmware upload, or even with every startup. There are common methods in place to manage flash integrity, such as redundancy checks, checksums, fallback boot modes, and so on. This document is not intended to discuss all methods, instead we will focus on an example using a 32 bit CRC calculation with a common polynomial, in order to provide a reliable method to check that the device was flashed with the intended data. Furthermore, using DAVE™ v4, XMC™ flasher, and the open source tool SRecord, we can create a process from application build in the native DAVE™ v4 environment to creating a reliable firmware image (containing integrity checks) that is ready to download to the target. Finally, we will provide examples of target code that can be used to manage these integrity checks.

1.1 Cyclic Redundancy Check (CRC) method

Cyclic Redundancy Check (CRC) is a method commonly used in digital applications to detect accidental changes to data. CRCs are specifically designed to mitigate errors caused by noise in data transmission and flash corruption. The premise behind this checking procedure is that blocks of data undergo polynomial division of their contents, which generates a unique remainder. These polynomials are often pre-determined values that are common to different applications, and have been tested to do so. CRCs are commonplace in communication systems (CAN, Ethernet, RS485, mobile communication, to name a few). The flexible CRC engine available on XMC4000 devices has several options for CRC calculation. Using the XMC1000, we will apply a common CRC algorithm in the absence of a dedicated CRC engine. Figure 1 below shows a block diagram of the XMC4000 FCE system and components available within the FCE.

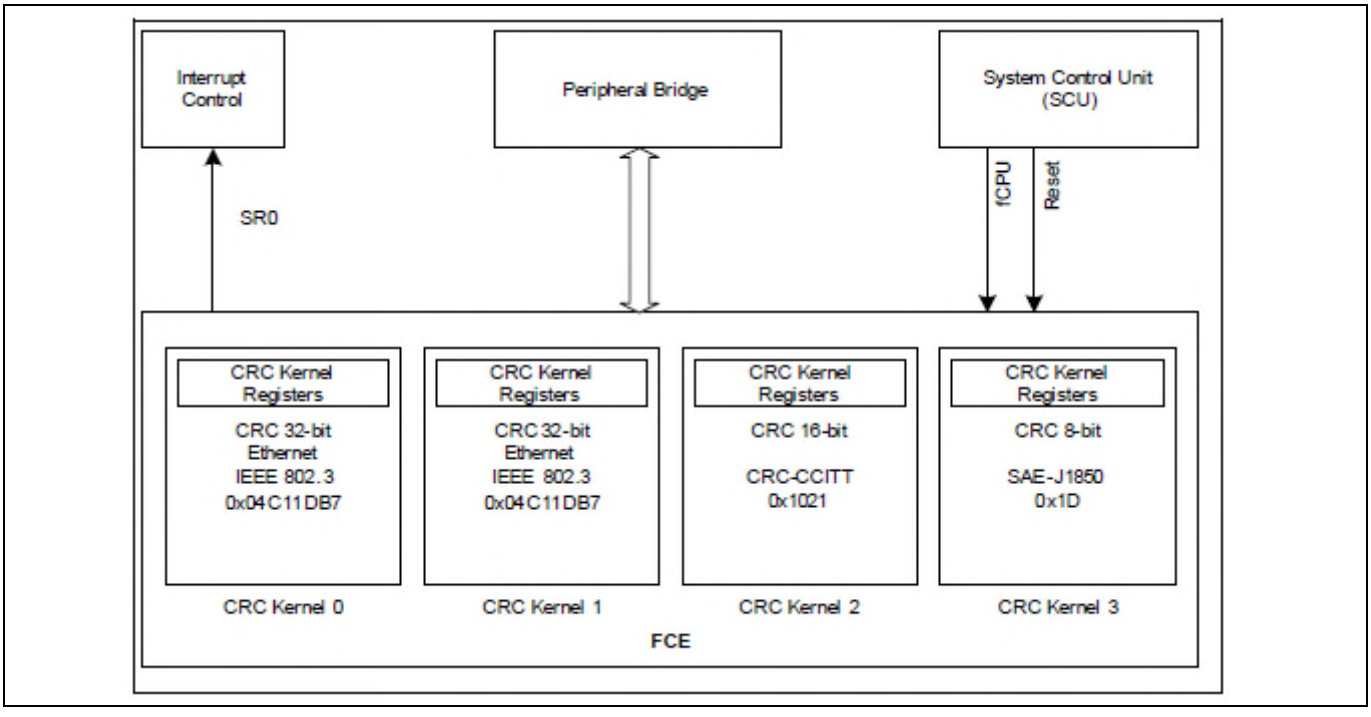


Figure 1 FCE block diagram

Given that the focus of this document is on testing integrity in large amounts of data, a higher bit CRC will calculate a more unique code. We can take a polynomial, such as the Ethernet polynomial 0x04C11DB7, which is available in the FCE. It translates to:

Eq. 1 $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$ **(with leading '1' at 32nd bit):**

or in binary:

100000100110000010001110110110111

As an example of polynomial division for the remainder, we take a 32 bit word, 0x01234567, and calculate the CRC below:

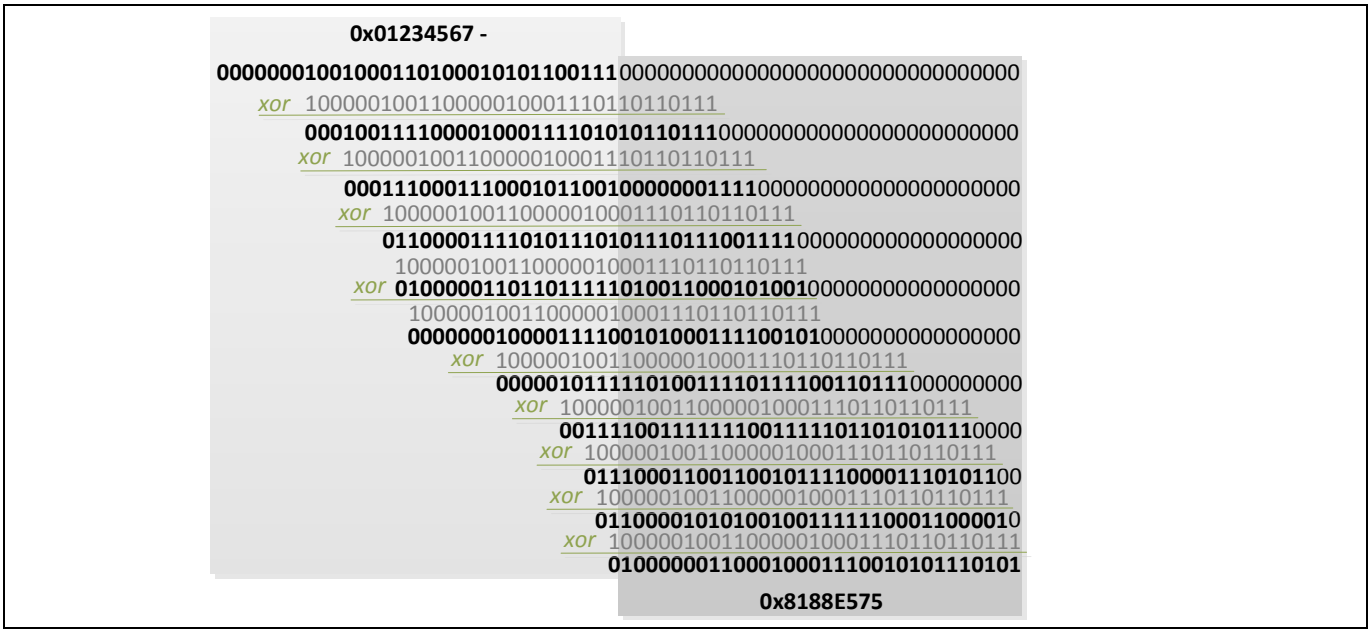


Figure 2 Example calculation of 32 bit CRC with FCE Kernel 0/1 polynomial

Above, we have the original data, given on the left. At the end of the data, we append a 32 bit seed value, N-1 the size of the polynomial (or 32 bits in this case). Here we use 0x00000000 as the appended seed value - it is also common to use 0xFFFFFFFF as a seed value. The operation of the CRC is to run sequential exclusive or (XOR) operations upon each bit position that will provide a leading '1'. After the XOR operations are completed across the data, the remainder is our calculated CRC value. We can confirm this behavior on the XMC4000 device. Figure 3 below shows the above example running on the hardware engine:

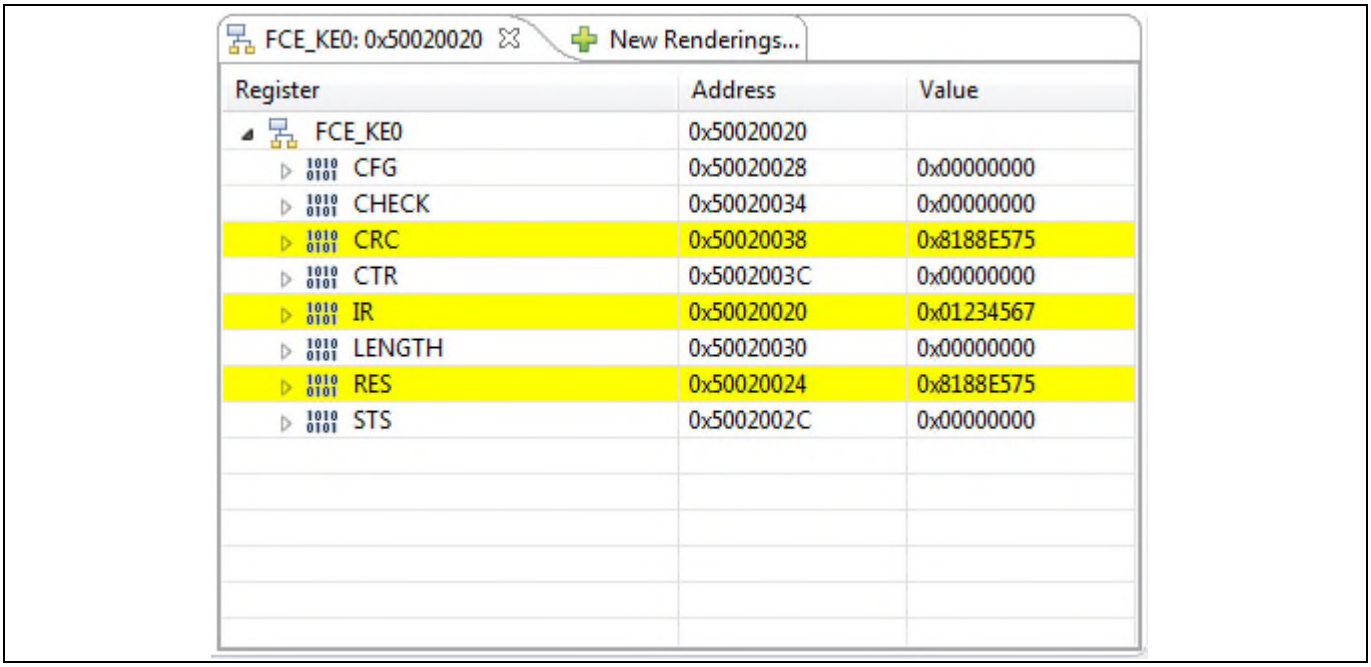


Figure 3 Example calculation of 32 bit CRC run on FCE hardware

The FCE Input Register (IR) receives the data in 32-bit width format. The hardware then runs the above operations and produces a result (RES). As we can see above, the data matches the calculation example in Figure 2.

2 Conditioning the application code

In order to run the integrity check and confirm its accuracy, we need to condition the application code to provide us with usable information. Mainly, we need to determine the size of a block of data, and store the known CRC value to the application code. This will give us a reliable method to compare a known good check against the check upon boot up or re-flashing.

2.1 Methodology

As briefly discussed, we need to control two items in order to accurately calculate the data integrity CRC. Firstly, we have to know the definititive application size. One method is to put a key at the end of the application. This unique identifier key will allow a routine to calculate the size by finding the key, thus determining the end of data. Another important step is to fill a given sector or multiple sectors of flash. The sector termination point usually coincides with flash partitioning for various functions (i.e. application code vs emulated EEPROM or other forms of calibration). Setting a limit at a given sector is a clean method for placement of the CRC. Next we need to consider installing the correct CRC in our application, in order to run a comparison. We have to compare this to a known good value, a part of the hex file (a value installed by PC-side tool), in order to confirm our calculation. In order to do this, we will need to apply the correct CRC at a known address. At the time of compiling, we cannot know the correct CRC value, so we have to allocate a location where we can later calculate the CRC and change the address location to the correct CRC value. Our application will appear as follows:

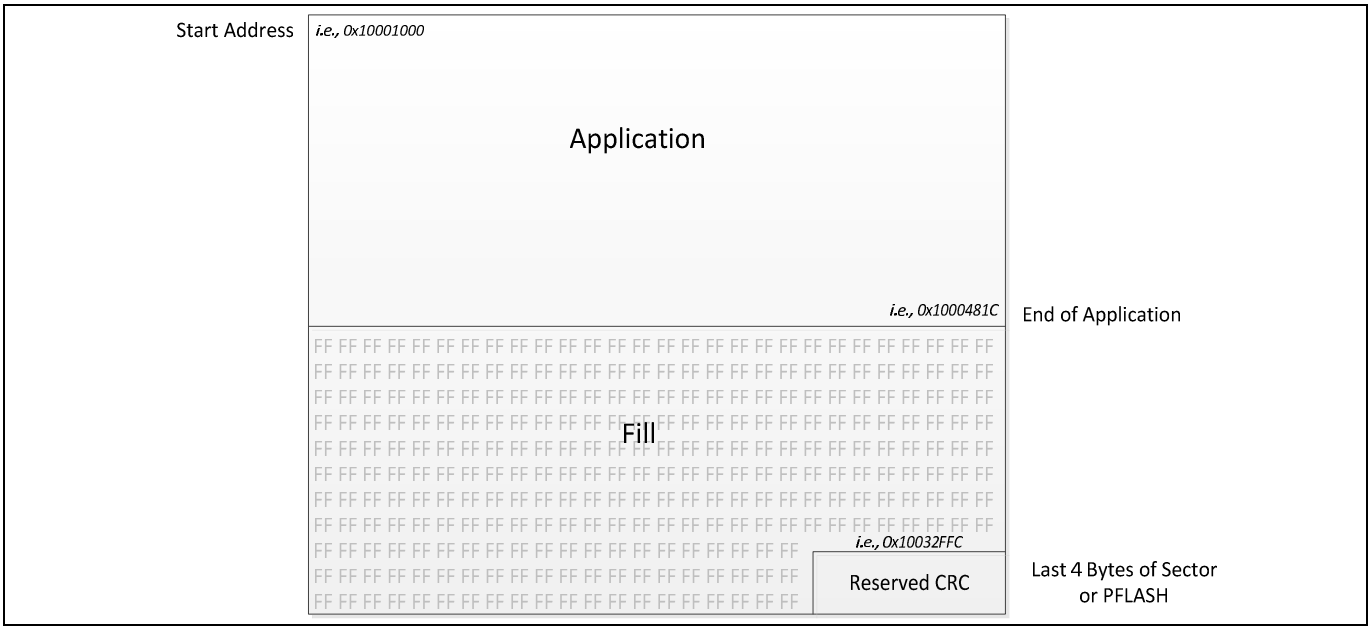


Figure 4 High level organization of application code

2.1.1.1 Using DAVE™, SRecord, and XMC™ flasher

As previously mentioned, we have to condition the raw application code to fill and generate the CRC. In order to precondition the code, we need a utility to fill and reserve a place for the CRC value. One way to do this is to use the widely available SRecord utility (http://srecord.sourceforge.net/). SRecord allows manipulation, by several methods, in order to append, crop, fill, etc. an SRecord format output. Once we have the SRecord in the corrected format, we can then manipulate the file to include the correct CRC value, which can be done with the XMC™ flasher.

2.1.2 Generating an SRecord

ARM-GCC - the compiler behind DAVE™ v4 - gives us the ability to output a SRecord. To do so, we have to do some basic manipulation of the project properties. The following setup already assumes you know how to set up a DAVE™ project (www.infineon.com/dave). In the project that you wish to output SRecord, you can right click the project and select “Properties”:

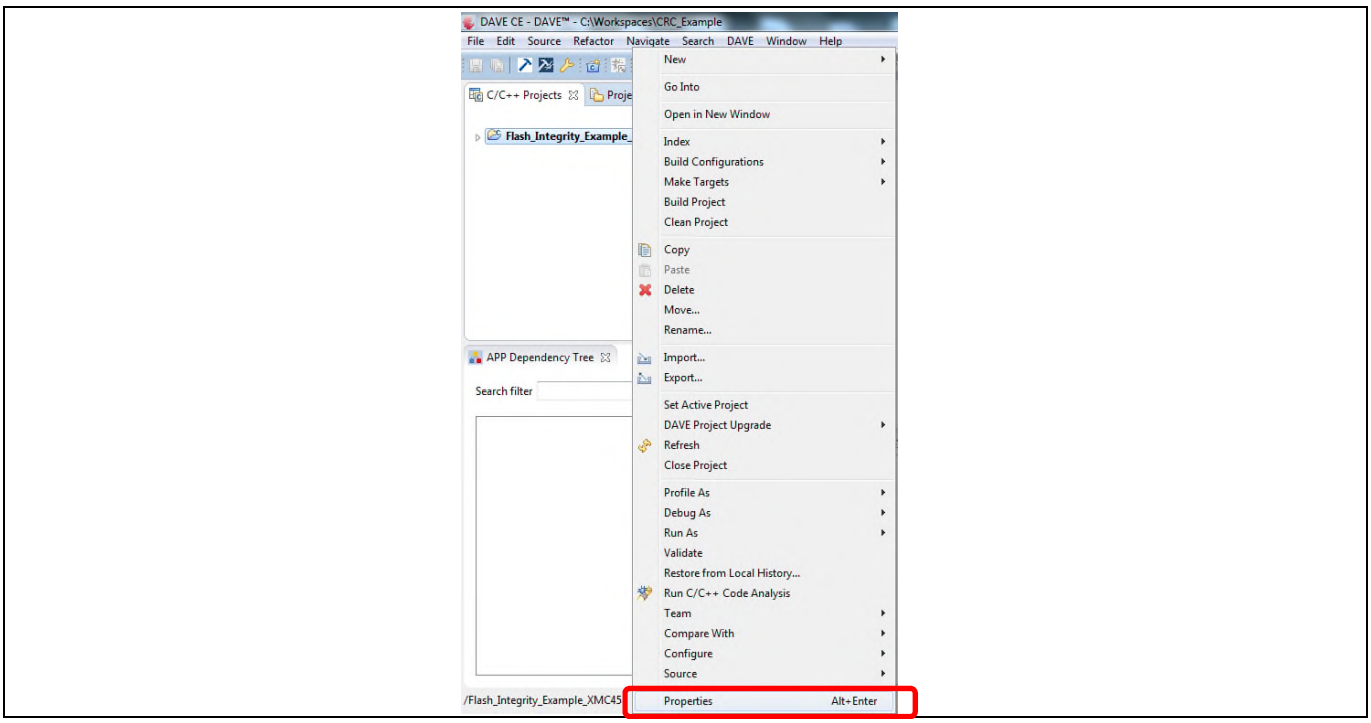


Figure 5 Properties tab

This will open up a window with properties for the project. Under “Settings”, there will be another set of selections for properties related to the GCC compiler, linker, and assembler. Under “ARM-GCC Create Flash Image” and the subheading “Output”, you will find an option to select the flash image output type. By default, it will be Intel Hex, or **‘ihex’**. You can change this choice to **‘srec’** as described below.

First, go to “Properties”, which you can right-click the project to open the properties of the project. Then go to “C/C++ Build” and “Settings”. Under “ARM-GCC Create Flash Image” you can choose “Output” and change to **‘srec’**.

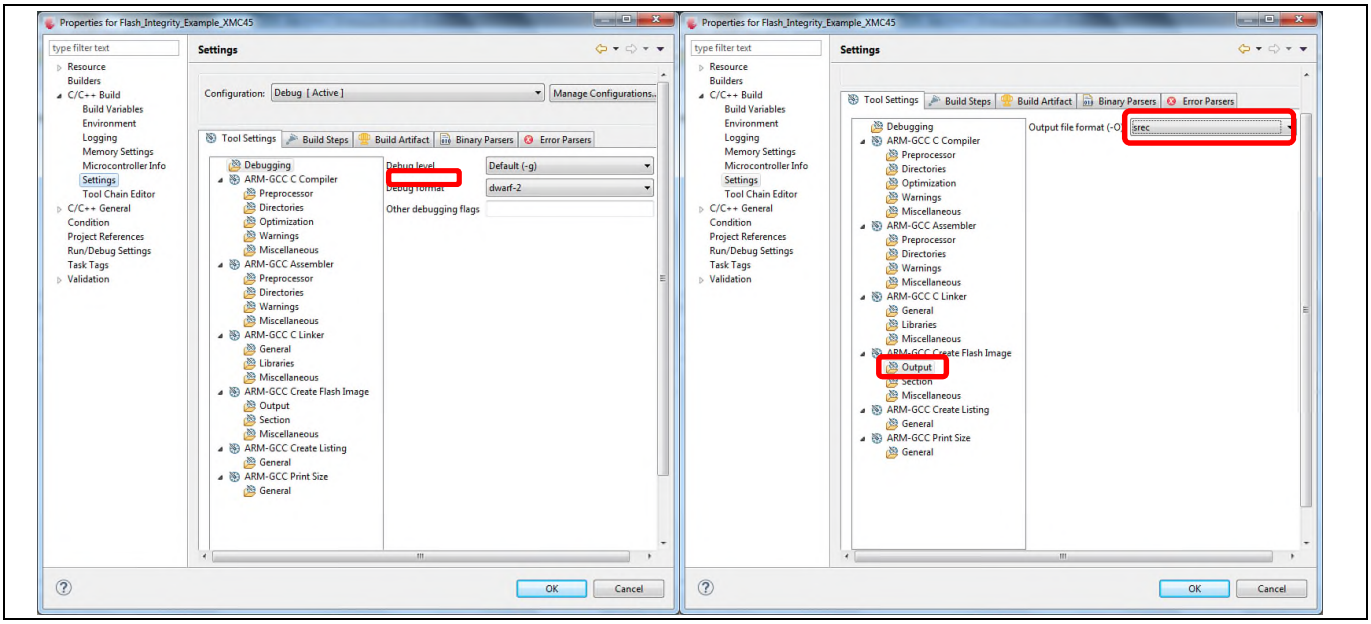


Figure 6 Properties setup of SRecord output

On the “ARM-GCC Create Flash Image” options page, you also need to inform the compiler of the preferred output file name, which requires some manipulation of the “Command line pattern”:

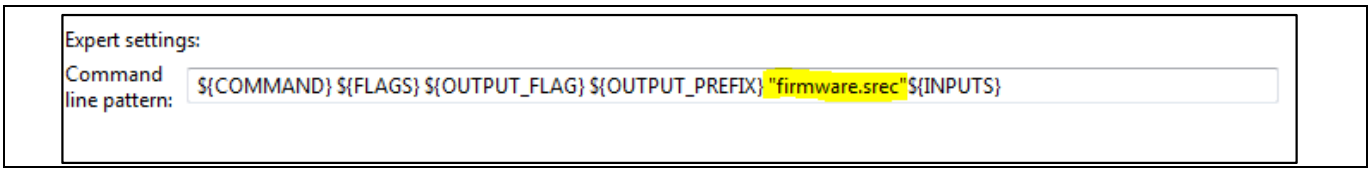


Figure 7 Output setup of SRecord

When you build the project , an SRecord file will appear in your “Debug” folder. The SRecord file is raw application code, with no fill or CRC value added.

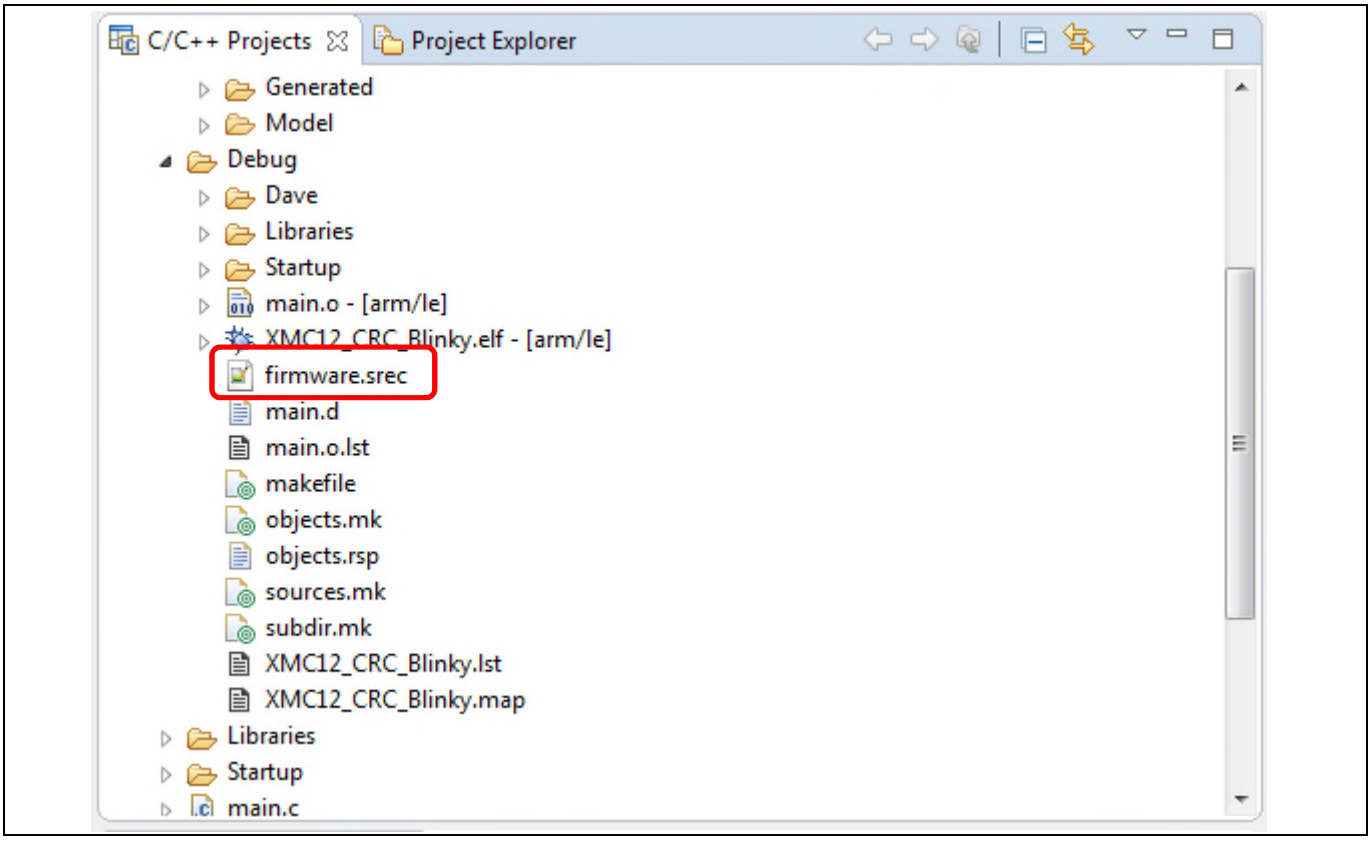


Figure 8 Debug output of SRecord

2.1.3 Formatting the SRecord

Now that we have the raw output of the SRecord, we can modify the file to meet our desired criteria. First we will fill the SRecord to a specific flash partitioning, such as a page or sector of flash. This is advantageous so we can check the flash integrity of a given boundary of flash. It is generally good practice to fill with non-zeros, although there are several opinions regarding regarding the best fill content. These examples fill the open flash space with '0xFF'. This guarantees that the integrity of the complete sector is checked (zero positions are passed over by CRC), but does not offer any special operations in application out-of-range conditions. In some cases, it may be reasonable to use no operations, or jump statements to push the program counter to a specific addresss. Finally, we need to pre-condition the flash word where we want to store the CRC. The tool we will use to create the recorded CRC will use a 'magic' number, a 32 bit value that is very unlikely to exist in real application code. XMC™ flasher, the tool we will use to do this, recognizes the 'magic' number 'FECABAFA'. The tool runs a CRC check on the SRecord, PC-side, and then replaces this 'magic' number with the real CRC calculation. This becomes our static comparison to in-application CRC checks.

In order to do this, we can use SRecord to fill and generate the magic key, with the command below:

```
srec_cat.exe application_firmware.srec      -crop 0x10001000 0x10032FFC
                                            -fill 0xFF 0x10001000 0x10032FFC
                                            -generate 0x10032FFC 0x10033000
                                            -constant-b-e 0xFECABAFA 4
                                            -o new_application_firmware.srec
```

We make several calls to commands that the SRecord tool manages for us:

- 'application_firmware.srec' is our input file.
- The "-crop" command crops the file to only the relevant address area we want to manipulate.
- The "-fill" command takes all unused address space and fills it with the value '0xFF'
- The "-generate" command will tell the tool to generate a specific value at an address range
- The "-constant-b-e" command tells "-generate" to apply the following constant (oxFECABAFA) over 4 bytes
- "-o new_application_firmware.srec" defines the output file to new_application_firmware.srec

2.1.4 Using XMC™ flasher to apply the CRC

Now we have a conditioned application file, without the calculated CRC32 value. In order to calculate the CRC, we can use XMCFlasher via the "-addchecksum" feature:

XMCFlasher.jar -addchecksum new_application_firmware.srec

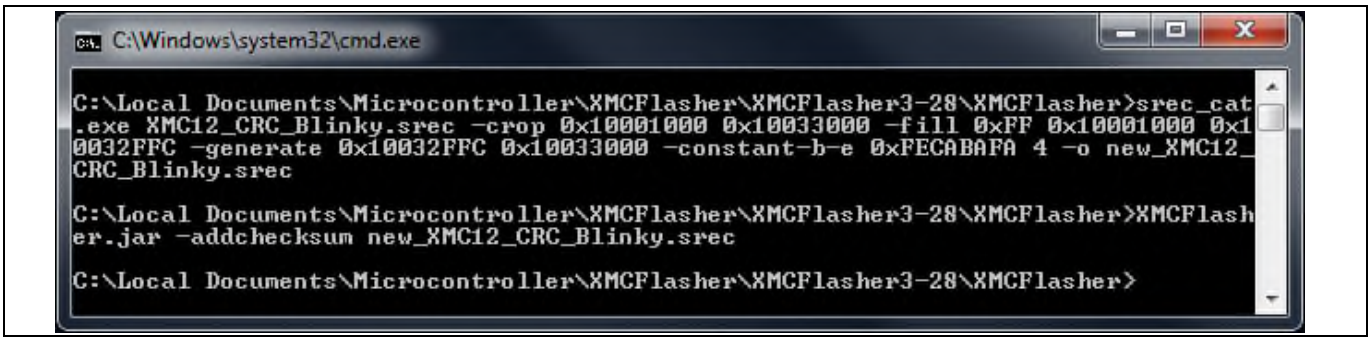


Figure 9 Command line example of SRecord conditioning

Here, we can see the image with fill and the Magic Key (excerpt from srecord file):

```
6399  S32510032FA0FFFF
```

3 Example application

3.1 XMC1000 family devices

The example application, “XMC12_CRC_Blinky” provides an example of how the explained image file conditioning works on a target device. The example first sets up a couple of indicators; P0.0 (which is connected to an LED circuit on the XMC1200 Boot Kit) is a 5Hz pulse which lets the user know that the application is running correctly. P0.2 is also an LED indicator that is used to show whether the CRC calculation is pass or fail. If the LED is OFF, the calculation has failed,and if the LED is ON, the calculation has passed. The passing condition is that the calculation matches the PC-placed CRC in a specific flash address.

The application also has a UART configured to a UART to USB connection through the debugger on the boot kit. We can accomplish this by configuring a UART DAVE™ App to the appropriate pins (P1.3 Rx, P1.2 Tx), which ties them to the debugger Virtual COM setup. So not only can we check against the LED, we can check the value via a terminal window as well.

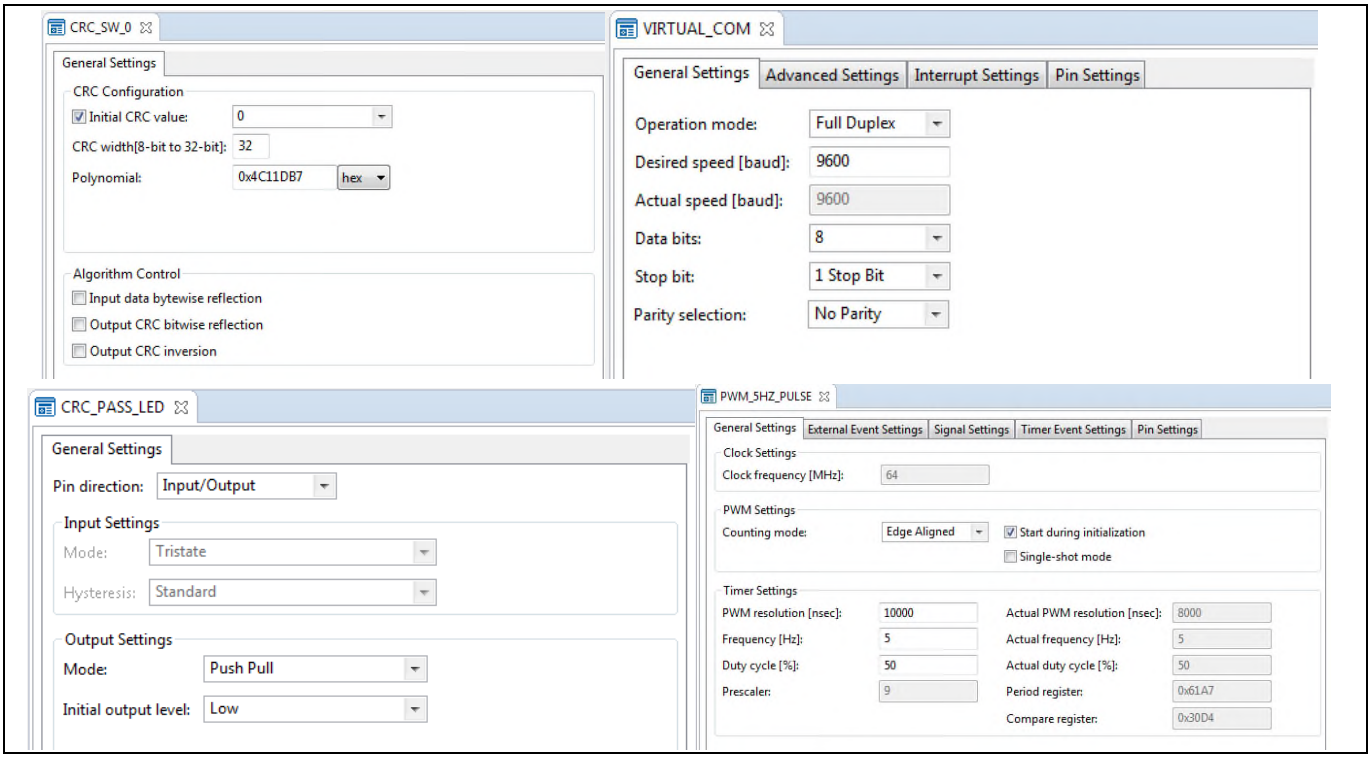


Figure 10 Configuration of related apps

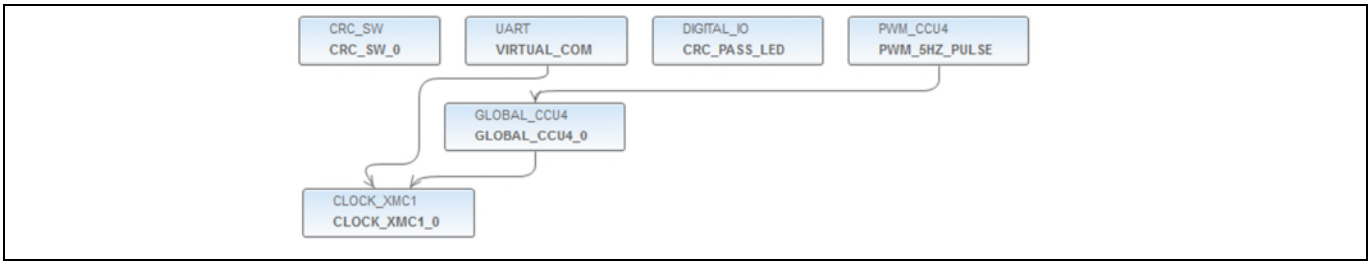


Figure 11 DAVE™ app setup of XMC12_CRC_Blinky

The project is supplied with the document. It is possible to test this by importing the project to DAVE™

(File->Import->Infineon->DAVE Project).

In order to compile and condition the file, the project is conditioned to output an SRecord via the methods described previously. There is also a batch process file included that will perform the conditioning of the file. This batch file utilizes both the “srec_cat.exe” and “XMCFlasher.jar” utilities, which must both be present in the directory. Copy the SRecord from DAVE™ v4 and run the batch file with the filenames adjusted and we will have a formatted SRecord output.

```
srec_cat.exe XMC12_CRC_Blinky.srec      -crop 0x10001000 0x10033000
                                        -fill 0xFF 0x10001000 0x10032FFC
                                        -generate 0x10032FFC 0x10033000
                                        -constant-b-e 0xFECABAFA 4
                                        -o new_XMC12_CRC_Blinky.srec

XMCFlasher.jar -addchecksum new_XMC12_CRC_Blinky.srec
```

Figure 12 XMC12_CRC_Blinky Batch File Contents

In order to run the application, we load the application firmware (srec file) via the XMC™ flasher. To do so, open the XMC™ flasher tool:

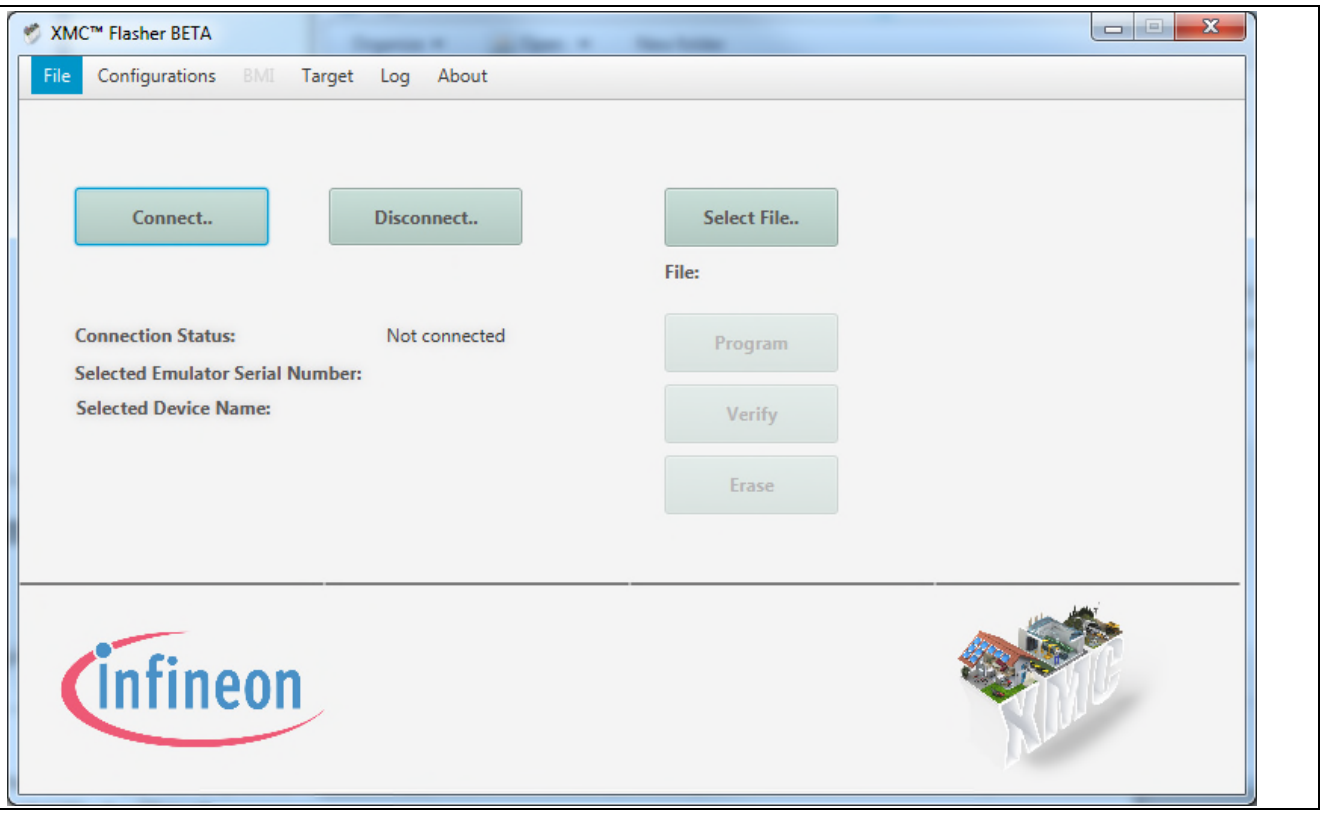


Figure 13 XMC™ flasher connection process

Choose “Connect”, and a device selection window will appear. Choose “XMC1200-200”.

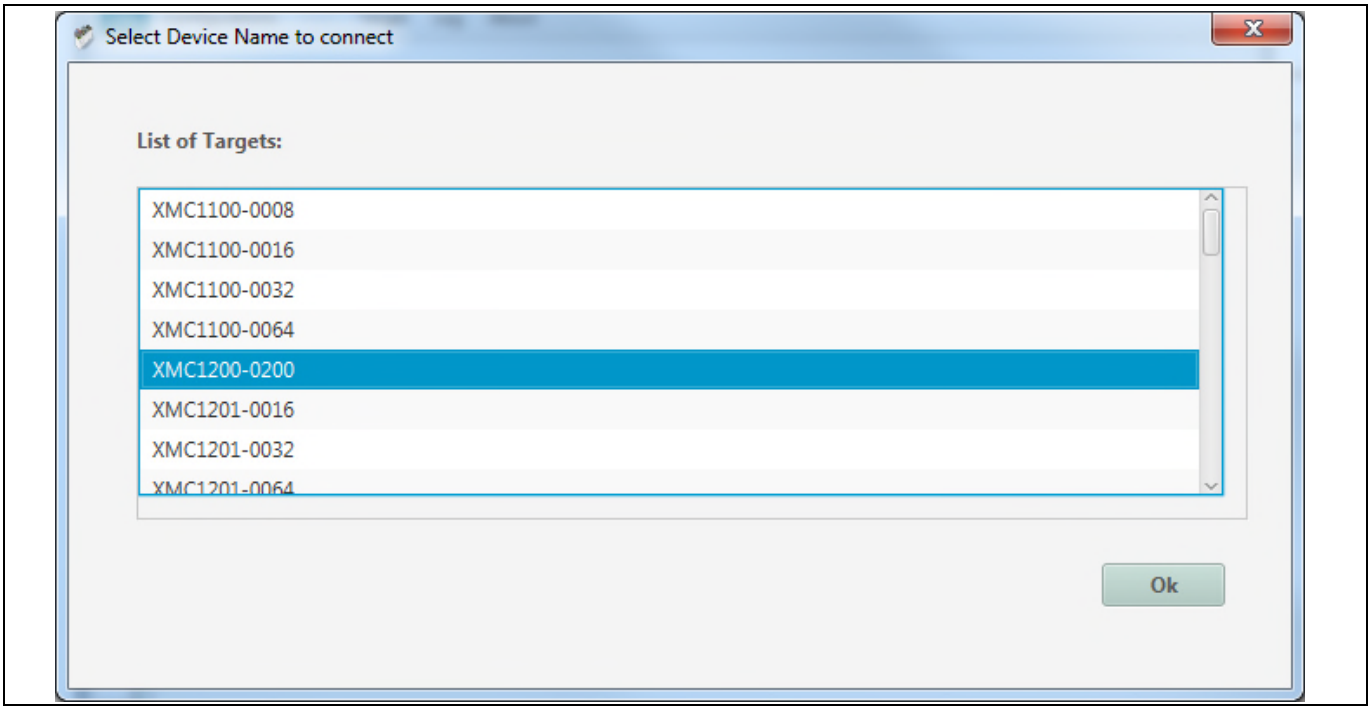


Figure 14 Device selection in XMC™ flasher

You are now connected to the target, now you choose “Select File” and in order to see the *.srec file, you will need to choose the “Motorola SREC-Files(*.srec, *.s19, *.mot, *.s)” format. The file “new_XMC12_CRC_Blinky.srec” is our formatted flash image.

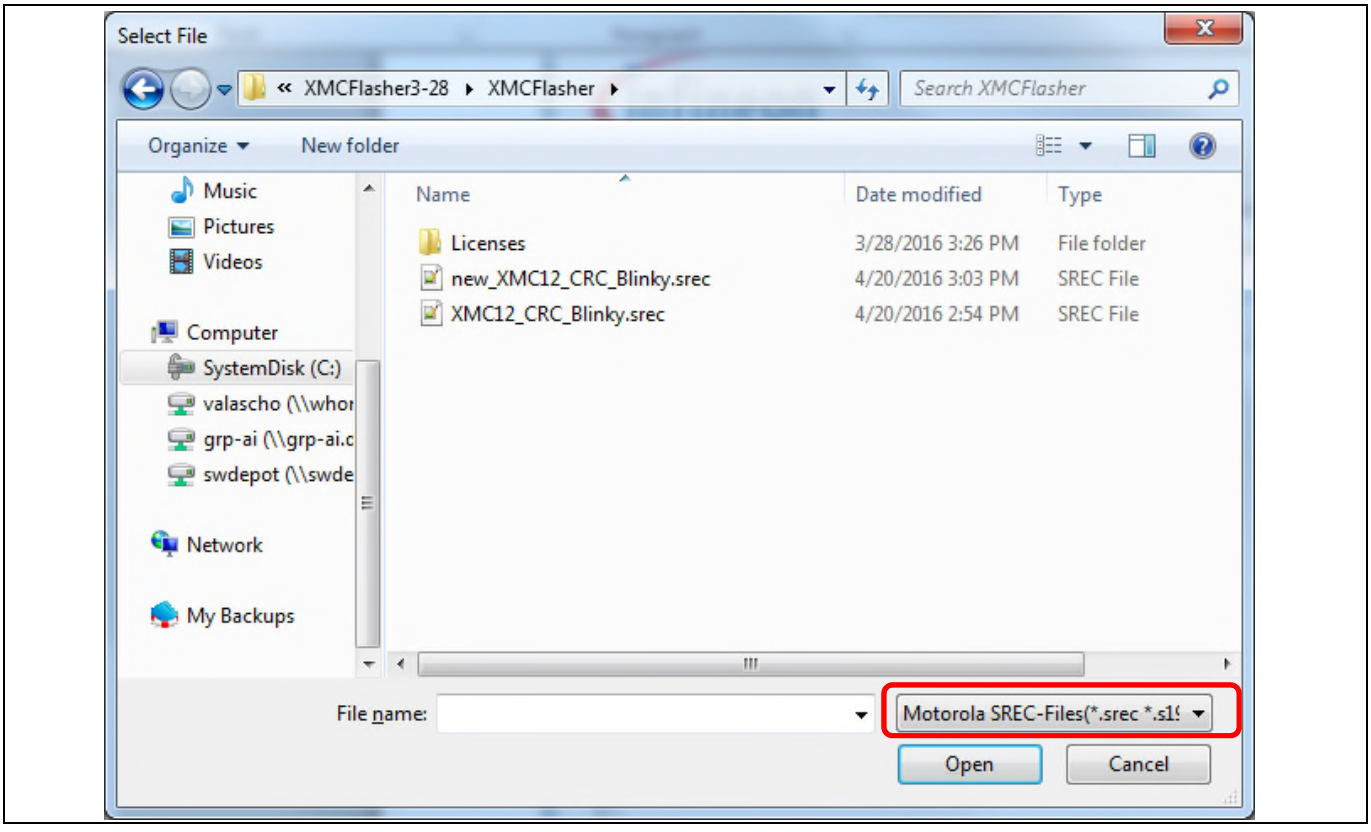


Figure 15 XMC™ flasher file selection

Open the file and select “Program”.

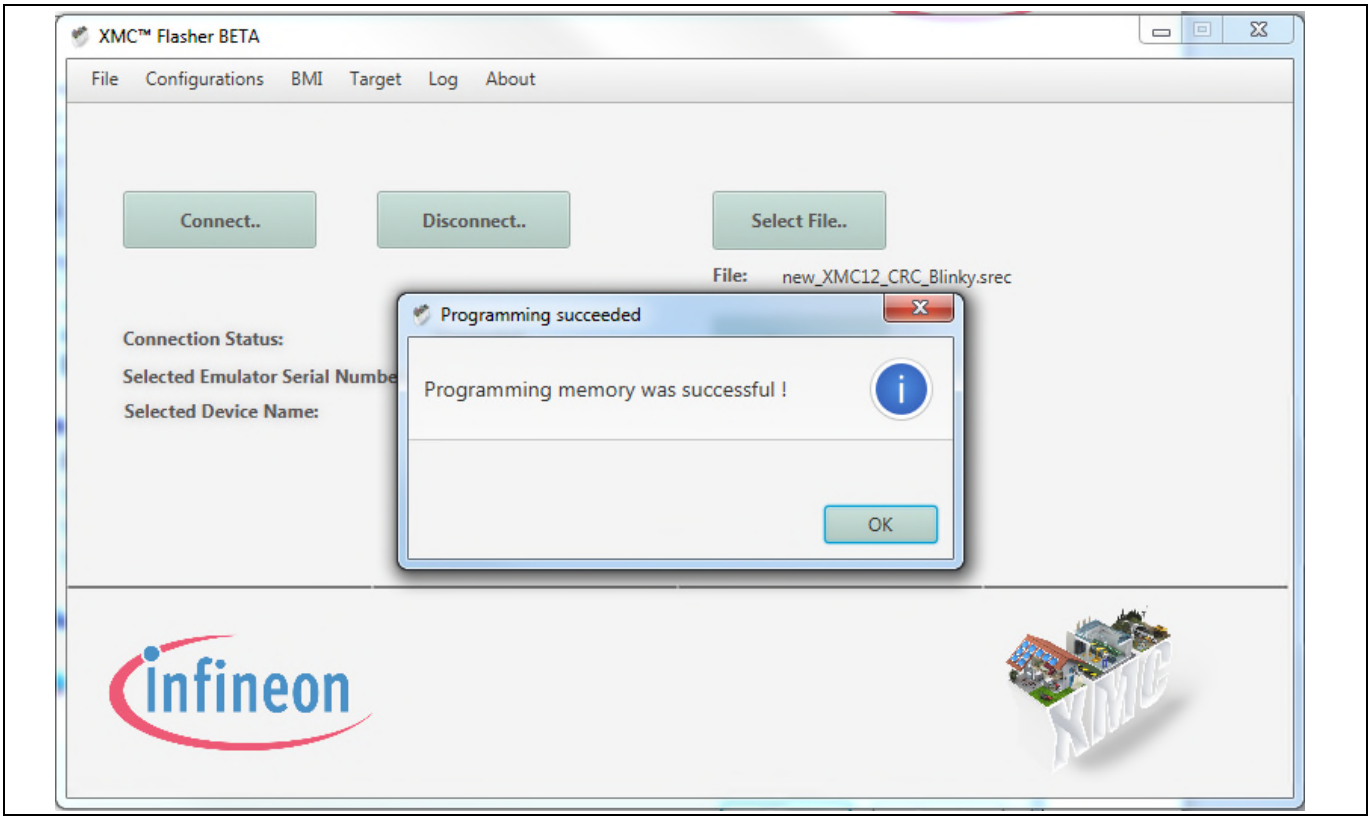


Figure 16 XMC™ flasher successful program

At this point, the application is running and you should be able to verify the process via either the LED indicators or through a terminal window.

View through terminal window.

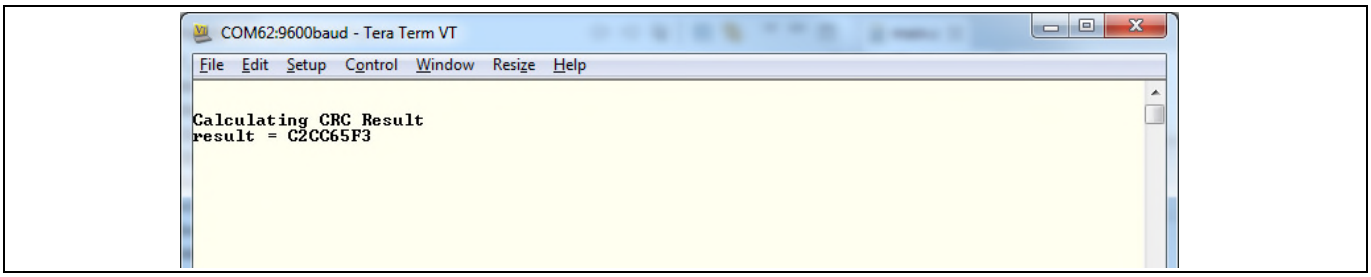


Figure 17 Terminal window output

The execution time for the example CRC program is 1.52 seconds on XMC1200 running at 32MHz clock.

3.2 XMC4000 family devices

As an example, “XMC45_CRC_Blinky” is provided. This example runs on the XMC4500 Relax or Relax Lite Kit. As this board does not have a virtual communications port through the debugger, we are using a USB VCOM setup directly from the target processor. The USB VCOM prints information to the terminal window at a 1s rate. The example looks very similar to the XMC1200 based design with exception of the inclusion of the flexible CRC engine.

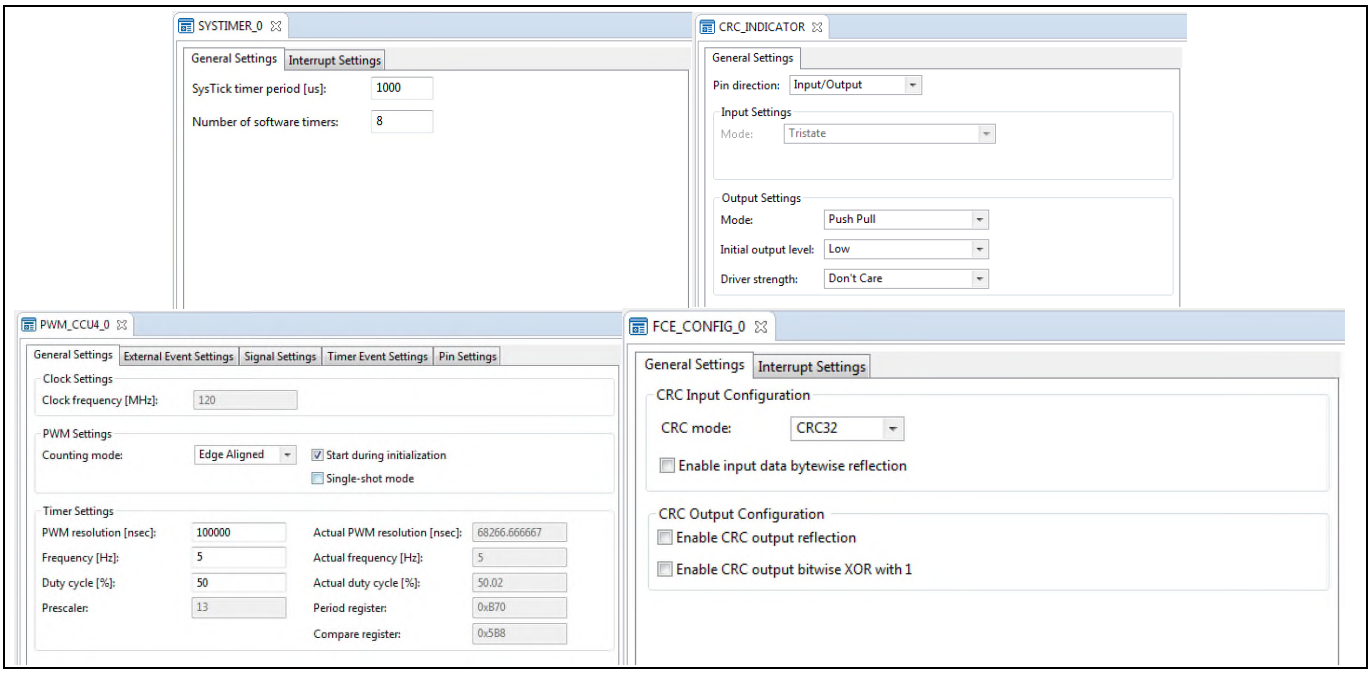


Figure 18 Configuration of related apps

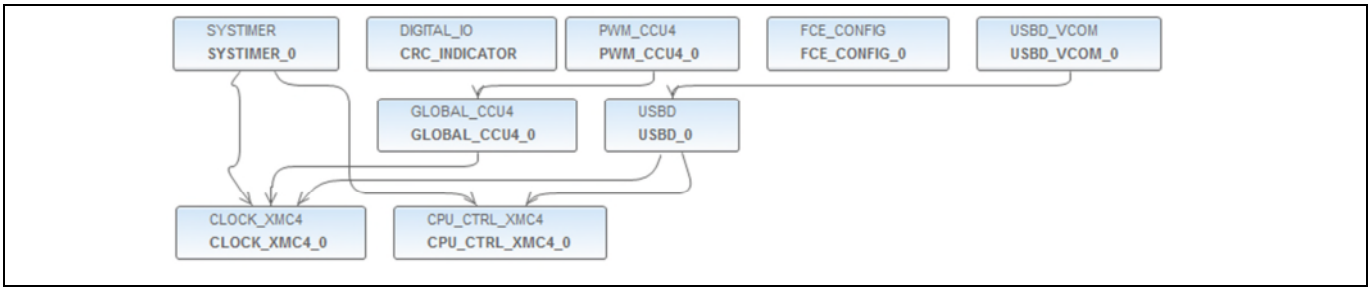


Figure 19 DAVE app setup of XMC45_CRC_Blinky

The XMC4000 family essentially follows the same process as the XMC1000 family, with just a few exceptions. Now we are using the HW accelerated flexible CRC engine, and as a result we will have to do a little code conditioning to make this functional. The XMC™ processors are Little Endian. As the CRC32 engine’s other primary usage is for managing communication, in particular Ethernet, the FCE is Big Endian. So, in order to manage the difference, we have to do some conversion. An algorithm has been placed in code to do so:

```
//Change the endianess before write to FCE.
for (uint32_t index = 0; index < MEMORY_SIZE; index=index + 4)
{
        value_for_FCE = 0;

    for (int k = 0; k < 4; k++)
    {
        value_for_FCE |= ((uint32_t) (*(data_for_FCE + (index + k)))) << (8 * (3 - k));
    }

    //Push 4 bytes to CRC engine
    FCE_KE0->IR = (value_for_FCE);
}
for(uint8_t i=0; i<20; i++);  //Small Delay to Insure Correct CRC Result
crc_result = FCE_KE0->RES;    //Read Result
```

Figure 20 Endianness adjustment algorithm for FCE

The same process is followed to generate an SRecord output. A batch process file has been created to convert the raw application image to the formatted version. When you load the application through the XMC™ flasher, you will see that the P1.0 LED blinks at a 5 Hz rate, signaling the target is running. P1.1 is an LED indication that the CRC has passed confirmation. Again, we can view through the terminal window (make sure to connect through USB port that is connected to target, and you have installed “driver.inf” on the PC-side..part of USB_VCOM DAVE generated software, ‘Dave\Generated\USBD_VCOM\inf’):

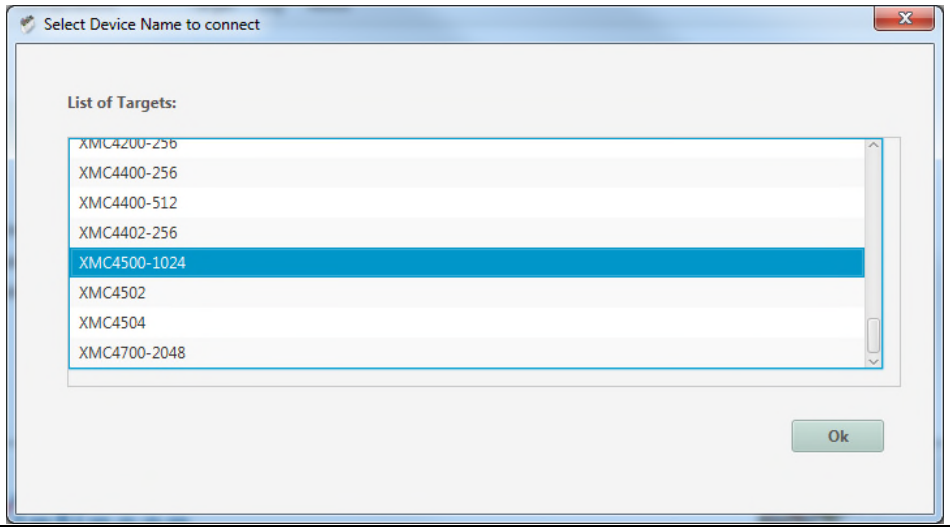


Figure 21 XMC™ flasher device selection

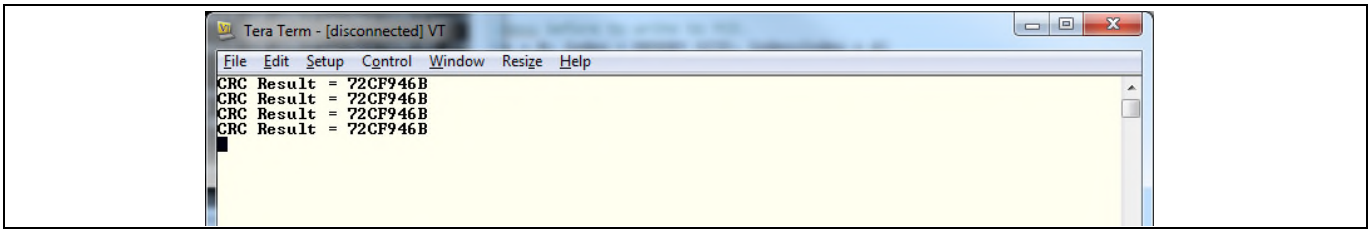


Figure 22 Terminal window result

The execution time for the example CRC program is 1.8 ms on an XMC4500 with a 120 MHz clock.

Revision history

Major changes since the last revision

Page or reference	Description of change
All	First release

Trademarks of Infineon Technologies AG

µHVIC™, µIPM™, µPFC™, AU-ConvertIR™, AURIX™, C166™, CanPAK™, CIPOS™, CIPURSE™, CoolDP™, CoolGaN™, COOLiR™, CoolMOS™, CoolSET™, CoolSiC™, DAVE™, DI-POL™, DirectFET™, DrBlade™, EasyPIM™, EconoBRIDGE™, EconoDUAL™, EconoPACK™, EconoPIM™, EiceDRIVER™, eupec™, FCOS™, GaNpowIR™, HEXFET™, HITFET™, HybridPACK™, iMOTION™, IRAM™, ISOFACE™, IsoPACK™, LEDrivIR™, LITIX™, MIPAQ™, ModSTACK™, my-d™, NovalithIC™, OPTIGA™, OptiMOS™, ORIGA™, PowIRaudio™, PowIRStage™, PrimePACK™, PrimeSTACK™, PROFET™, PRO-SIL™, RASIC™, REAL3™, SmartLEWIS™, SOLID FLASH™, SPOC™, StrongIRFET™, SupIRBuck™, TEMPFET™, TRENCHSTOP™, TriCore™, UHVIC™, XHP™, XMC™

Trademarks updated November 2015

Other Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2016-06-10

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2016 Infineon Technologies AG.
All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AP32339

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.