MDK Flash Download

AN334, Autumn 2020, V 1.0

keil-feedback@arm.com

Abstract

An integral part of debugging is the download of the application into the target's Flash memory. This application explains how Flash algorithms are used in MDK to erase, download, and verify the application in the Flash memory.

Contents

Copyright © 2020 Arm Ltd. All rights reserved

Introduction

Flash algorithms are position-independent software to erase, download, and verify applications in Flash devices of an MCU. After download into the target's RAM, they are executed by a CPU in the target system.

Download and execution are controlled by the debugger which takes over control of the CPU and manipulates the PC, SP, LR, and **general-purpose** registers to select the appropriate algorithm function and to pass arguments to it. LR is normally programmed with the address of a BKPT instruction to ensure that the CPU stops after finishing a function. Results can be read back via R0 as per the Arm Procedure Call Standard (APCS).

While it would be a viable approach to directly program Flash memory with debugger accesses, the current Flash algorithms come with the following benefits:

- Better scalability by following a fixed scheme to program devices of various brands and technologies.
- Higher programming speed: particularly low-cost debug probes have bandwidth and clock limitations on the debug channel. Executing the algorithms in the target system reduces the number of necessary debugger accesses.
- Complex Flash controllers can be handled much easier in plain C-code than by sending a series of debugger commands.
- Using C-code allows better error handling and decision making during the Flash download operations. Debuggers often lack the capabilities of complex error handling/decision making.

Terminology

The following terms are used in this document:

- **Flash Page**: Smallest Flash memory range being programmed with a single flash controller operation.
- **Flash Sector**: Smallest Flash memory range that can be erased with a single flash controller operation. Normally, a sector consists of multiple pages.
- **Flash Device**: A Flash device with its own properties and Flash algorithm. A Flash device normally consists of multiple sectors. Sectors can have a varying size.
- **Fill Value**: The fill value normally corresponds to the bit pattern that resides in blank Flash memory. Hence it can also be referred to as the "blank value". An erase operation should make sure that memory is set to this value after it is erased (if readable at all in erased state).

Flash Download Steps

The following high-level overview shows the steps that are gone through in µVision to download code into the target's Flash memory, using the target's RAM as temporary storage:

Each of these Flash download steps is completed before entering the next. That means for example that all relevant sectors are erased first before programming the first Flash page. Flash algorithms required in multiple steps are downloaded again for each following step.

- **Connect Debugger**: Establish debugger connection.
- **Load AXF files**: Load and cache code from AXF files in debug driver (on the host PC side). If required, multiple AXF files can be loaded and merged before Flash download.
- **Erase**: Erase full chip or sectors that are going to be newly programmed.
- **Program**: Program pages covered by the loaded AXF files and the Flash Download Setup dialog.
- **Verify**: Verify that the programmed areas hold the expected code.
- **Shutdown Flash Download**: Finalize the Flash download and disable core debug.
- **Reset & Run Application**: Execute a HW reset, i.e. a reset via the device's external reset pin. The type of this final reset is not configurable.

Copyright © 2020 Arm Ltd. All rights reserved

Connect Debugger

Please refer to Algorithm Functions of the CMSIS documentation for an example sequence of events on a debugger connection.

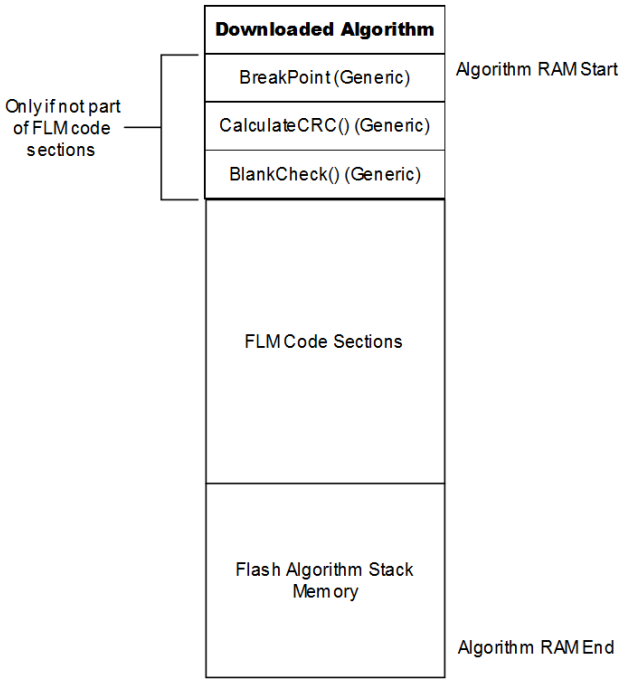
Load AXF File(s)

µVision loads and caches code in the active debug driver from AXF and HEX (Intel 386 format) files that are selected for Flash download:

- The build output of the current build target is always automatically loaded.
- If an INI script is added to the "Utilities" tab of the "Options for Target" dialog, and if this INI script contains LOAD commands, then additional AXF or HEX (Intel 386 format) files can be specified for Flash download.
- Code regions from multiple files must not overlap to avoid unpredictable side effects.
- You can program code regions from multiple files that do not overlap but share the same Flash sector/page. However, these must be programmed with a single Flash download. µVision does not have a read-modify-write functionality for Flash sectors.
- Pages that are not fully filled with code are padded with the "Fill Value" that is specified in the Flash algorithm. Otherwise it is possible that specific Flash memories will not function correctly.

Loading Flash Algorithm

Before executing Flash algorithm functions, they are downloaded to the target system's RAM. This is complemented by other supporting functions, code snippets, and information. Information and functions downloaded into target RAM have the following format.



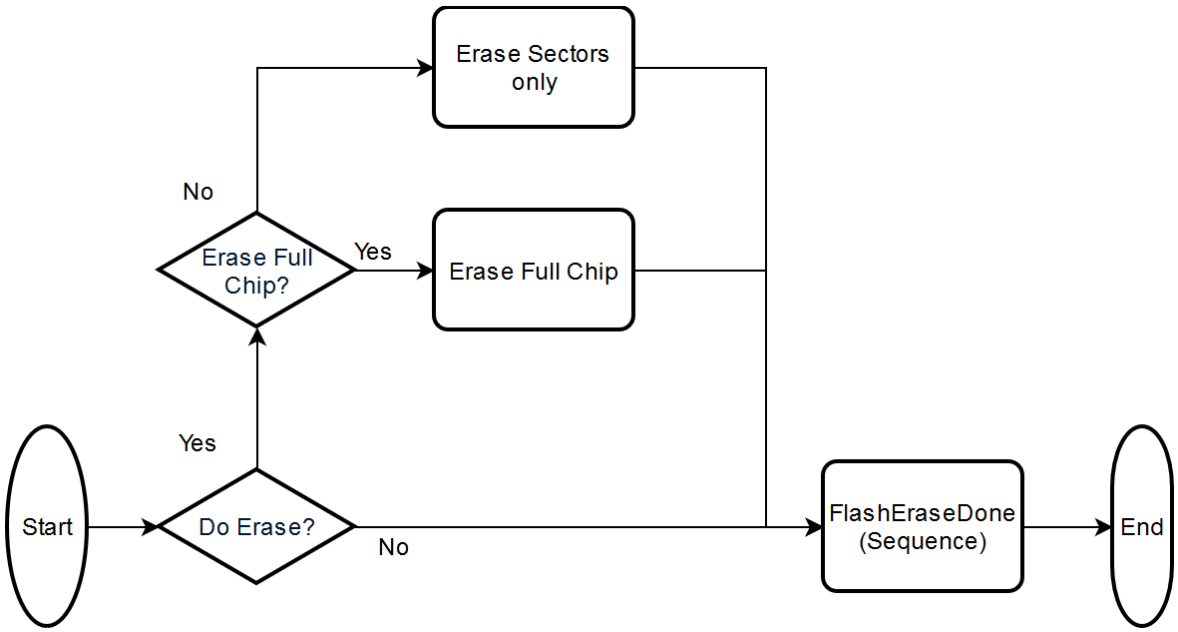
Notes:

- µVision checks if the first instruction of each algorithm function has the least significant bit set in order to detect any non-Thumb code for Cortex-M targets.
- More information on Keil Flash algorithms can be found under Algorithm Functions.

Copyright © 2020 Arm Ltd. All rights reserved

Erase

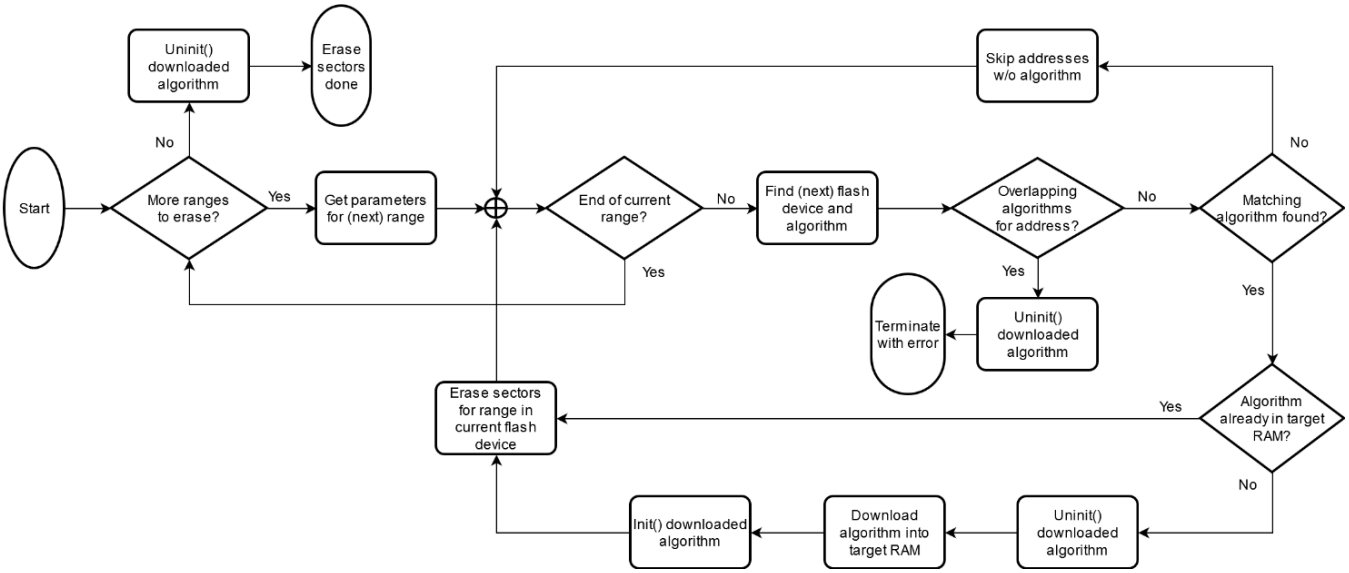
The erase operation can be configured to erase the full chip, erase only sectors that are to be programmed, and to entirely skip the erase step.
Additionally, the FlashEraseDone debug sequence can be provided by a Device Family Pack to execute device specific functionality at the end of the erase step. Currently, it is executed for all three possible paths.



Erase Sectors Only

When erasing sectors only, the debug driver identifies which Flash sectors need to be erased. This is based on the address ranges to program as per the AXF file contents.

Note: It can be necessary to load multiple Flash algorithms for one Flash download procedure. This happens sequentially in µVision, i.e. one Flash algorithm is downloaded and executed to the target RAM at a time.

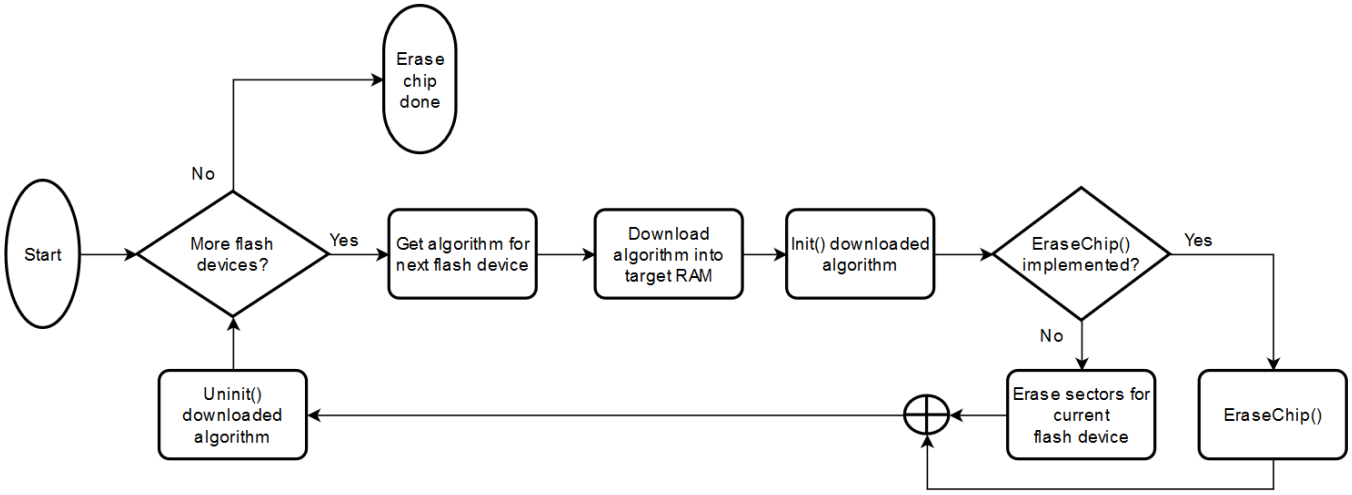


Copyright © 2020 Arm Ltd. All rights reserved

Erase Full Chip

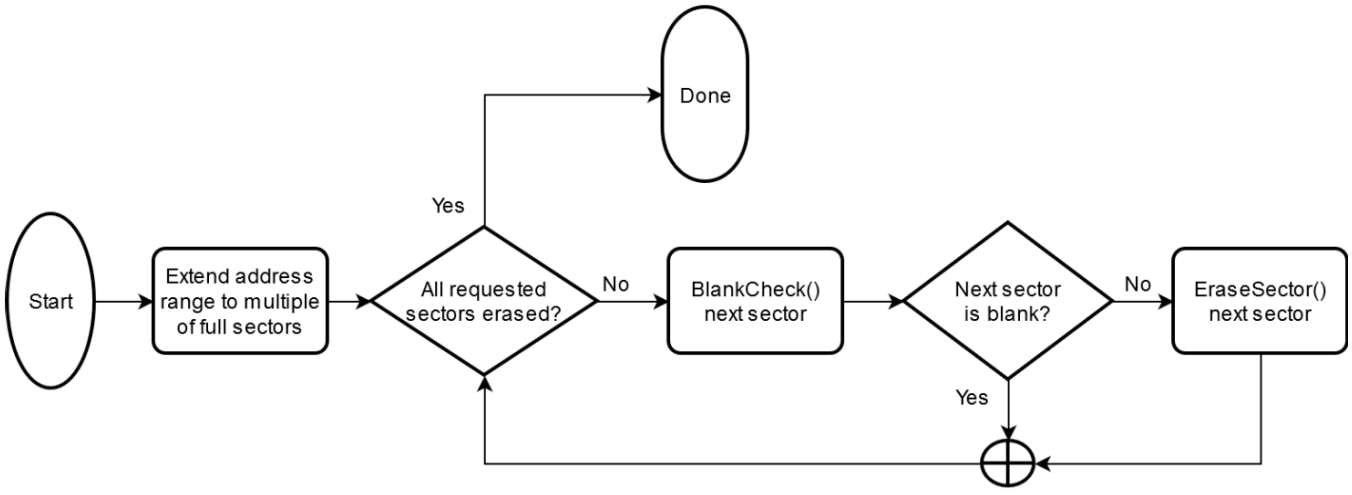
Erasing the full chip can happen via one of the following two methods:

- If the Flash algorithm provides a dedicated EraseChip() function, then this gets called. See the flow chart below.
- If the Flash algorithm does not provide an EraseChip() function, then the debugger erases the entire chip by erasing all sectors of each Flash device with the use of the EraseSector() function.



Erasing Sectors by addresses

The following flow chart shows how sectors are erased. Input parameter is an address range (start address and number of bytes). The chart assumes that the correct Flash algorithm for the requested address range has been found, downloaded into target RAM, and initialized by executing the Init() function of the algorithm. Uninit() is called after completion as outlined in **Erase Sectors Only** and **Erase Full Chip**.

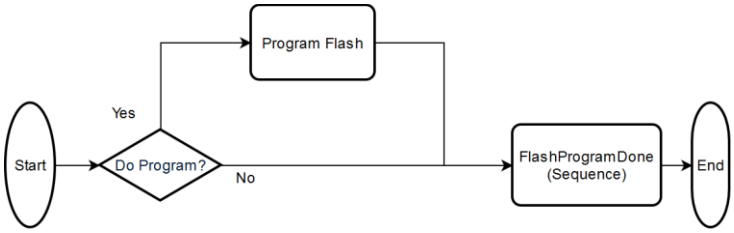


Copyright © 2020 Arm Ltd. All rights reserved

Program

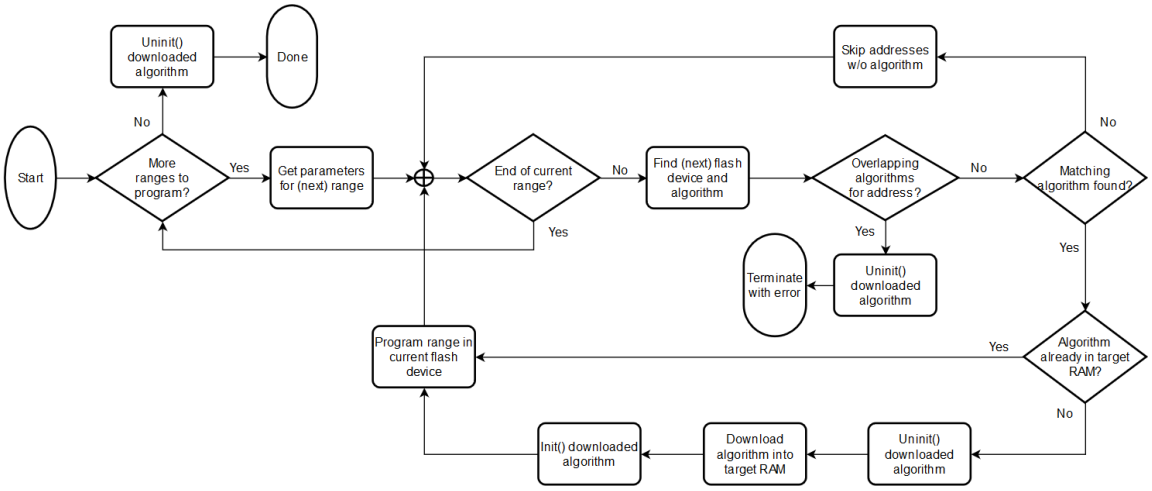
This step programs the AXF contents into the target Flash. Address ranges in the AXF not covered by the Flash Download Setup dialog in µVision are assumed to be ranges in RAM. They are skipped during Flash download.

The FlashProgramDone debug sequence can be provided by a Device Family Pack to execute device specific functionality at the end of the programming step. Currently, it is executed regardless of executing any Flash programming.



Program Flash

The following diagram shows the various steps and decisions a typical debug driver in µVision takes.
Like the erase step, Flash memory is programmed sequentially. That implicitly means that it is programmed Flash device by Flash device.



The actual programming ("Program range in current Flash device") happens page by page:

- Page buffer gets prepared
  - If start address of target range is not aligned with page start, then the beginning of the page is padded with the Flash algorithm's Fill Value.
  - If the end of target range is not aligned with the end of a page, then the remainder of the page is padded with the Flash algorithm's Fill Value.
  - In the rare event of other kinds of gaps these are of course padded with the Fill Value, too.
- Page buffer is downloaded behind the algorithm functions and helpers in the target RAM that is reserved for Flash algorithms by the user setup.
- The ProgramPage() algorithm function is called with the corresponding parameters.

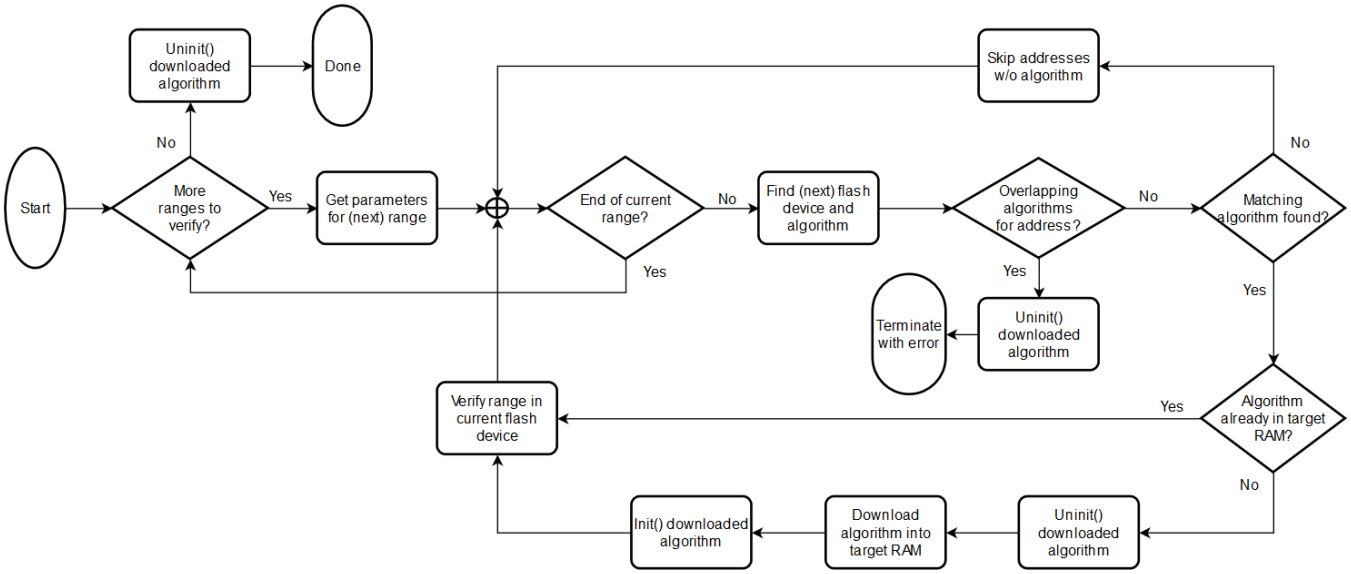
Notes:

- µVision currently supports a maximum page buffer size of 64 KBytes.
- µVision does not read-modify-write existing Flash sectors.
- µVision does not check if a configured range needs update, i.e. it does not check if the contents to write patches the current Flash contents.
- µVision does not care about pages within a sector that have no code to program. The already existing content remains. If the content is blank or if it holds previously downloaded code depends on whether it is part of a sector that was erased during the erase step.

Copyright © 2020 Arm Ltd. All rights reserved

Verify

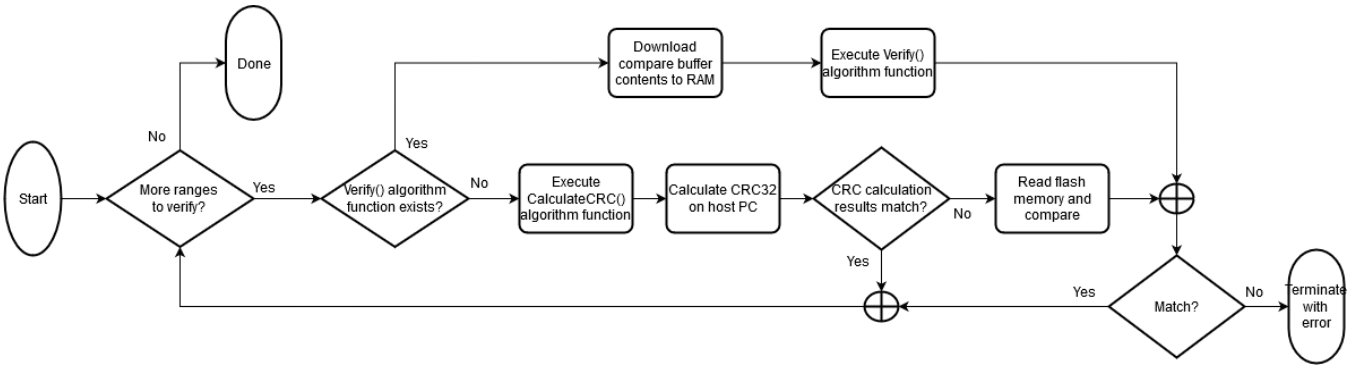
The verify step shall ensure that the Flash memory has been correctly programmed.



Verify address range

The actual verification of the contents downloaded to a Flash device depends on the provided by the Flash algorithm:

- If a Verify() function is implemented, the debug driver downloads the expected contents to the algorithm's RAM buffer and executes the function on the target CPU.
- If no Verify() function is implemented then the debug IDE
  - executes the CalculateCRC() function (internal to µVision) over the specified range on the target system.
  - executes the same algorithm on the expected Flash contents in the debug driver.
  - compares the results.
- If there is no Verify() function and the CRC comparison fails, a last attempt is done by reading the Flash contents via debug accesses and comparing it against the expected contents.



Notes:

- The CalculateCRC() function executed in target is not part of the actual Flash algorithm. It is a standard implementation within µVision and downloaded with each new Flash algorithm.
- "Read Flash memory and compare" does not really terminate the operation on the first mismatch. Standard AGDI implementations detect and print up to 100 byte mismatches before aborting the verify.

Copyright © 2020 Arm Ltd. All rights reserved

Shutdown Flash Download

The following things happen during shutdown of a successful Flash download:

- If a BKPT instruction was written to RAM while downloading the last Flash algorithm it gets replaced by a dead loop. Deactivating core debug will set the CPU running. If a CPU executes a BKPT instruction without halt debug or a debug monitor enabled, then it will cause a HardFault which can further escalate to a CPU lockup. After a normal order of events, the PC will be at this point halted at a BKPT instruction. This is the result of the last successful execution of a Flash algorithm function.
- Core debug is stopped. The functionality basically corresponds to the default implementation of the CMSIS debug sequence DebugCoreStop.

Reset & Run Application

Set the option "Reset & Run" to execute a final reset and hence run the application after finishing the Flash download. The executed reset is always a HW Reset (device's external reset pin) regardless of the selected reset type in the debugger settings.

Notes:

- Some target systems have configurable reset controllers that determine the functionality of this reset. Also, there are specific SoC and board implementations which have the reset line to the debugger disabled/disconnected. In such cases you need to make sure that the ResetHardware debug sequence is overwritten in the Device Family Pack to generate a comparable reset.
- On some boards, a HW Reset is not sufficient to reset all required components like for example Flash caches. In that case, overwrite the ResetHardware debug sequence in the Device Family Pack. If this is not possible, then you have to manually power-cycle (pull the power cable and plug it back in) the board/device.

Using Alternative Flash Programming Algorithms

Some devices cannot be programmed with a standard Flash algorithm executed from the target CPU. System security features might be impacting accessibility/usability of system components like RAM or Flash controllers.

In this case, use CMSIS debug sequences to replace the execution of RAM based Flash algorithms. See Default debug access sequences for more details. Involved sequences are FlashInit, FlashUninit, FlashEraseSector, FlashEraseChip, and FlashProgramPage.
The difference to above flow charts is that the execution of RAM based functions is replaced by the according sequence. Also, this approach always needs to read target Flash memory with debugger accesses to compare it against the expected contents.

Copyright © 2020 Arm Ltd. All rights reserved