LAUTERBACH
DEVELOPMENT TOOLS

How to Write your own FLASH Algorithm

Release 02.2025

MANUAL

How to Write your own FLASH Algorithm

How to Write your own FLASH Algorithm

Version 13-Feb-2025

FLASH Programming

Target Controlled Flash Programming

On target controlled programming these control accesses are done by a user designed program executed by the target processor. The mechanism of the debugger to access memory is used only to hand the data which should be programmed (via a data buffer) to the program routine.

When do I have to write my own target-based FLASH Algorithm?

- Every flash type can be supported.
- Flash can be supported even if it is not connected typically to the processor (e.g. exchanged data or address lines).

What do I need for my own target-based FLASH Algorithm?

- The programming routine running on the target processor must be written by the user.
- RAM for data buffer, argument buffer and stack is required on the target.

What must I do for target-based FLASH Algorithm?

1. Write the program/erase function running on the target processor.
2. Convert the function to a binary file.
3. Connect this routine to the debugger by using the commands FLASH.Create … and FLASH.TARGET …
4. Use the same commands FLASH.Program …, FLASH.Erase … These commands will call the programming routine with the appropriate parameters.

Declaration

There are two commands which inform the debugger about the flash configuration and the prepared programming routine:

```
FLASH.Create 1. 0x200000--0x3fffff 0x10000 TARGET WORD
```

This command specifies the unit number (1.), the address range (0x200000—0x3fffff) of the flash on the target, all uniform 64 KByte sectors (0x10000), informs that a prepared routine running on the target should be used (TARGET) and there is another parameter (WORD) which will be handed as a parameter to the target routine (see below). This parameter can be BYTE (1), WORD (2), LONG (4) or QUAD (8) the meaning should be the data bus size of the processor when accessing the flash(s). E.g. on two 16-bit wide flashes in parallel it should be: LONG.

```
FLASH.TARGET 0x100000 0x101000 0x1000 flashprog.bin
```

This command informs the debugger about the target routine start address (0x100000), about parameter block start address (0x101000; size: 20H bytes) always immediately followed by the data buffer (size: 1000H bytes). The parameter block size must be 20H plus up to 4 KByte data buffer for programming data. Data buffer size can be selected by the user. It determine the maximum number of bytes which are handled on one programming routine call.

What is the Debugger doing?

Programming:

If there is a data transfer entered (e.g. Data.Set … or Data.LOAD …) and if the programming is active (e.g. by Flash.Program ALL), the debugger checks if the address is within the flash address range (given by FLASH.Create <address_range> …). If so, it loads the data buffer with (part of) the programming data, initializes the parameter block, sets the program counter to the start address of the routine and starts program execution. When it gets back the control, it checks the return information (see parameter block) and if it is ok and if there are remaining data it does the same steps with the next data.

Erasure:

If the address matches part of the flash address range, the parameter block will be prepared for sector erasure and the target routine will be called for each sector inside the address trange. Then it checks the return information.

If the whole flash should be erased, the parameter block will be prepared for bulk erasure and the target routine will be called once. Then it checks the return information. If return code represents “not implemented” bulk erasure is performed by sector erase calls for each declared sector of this unit.

Contents of the Parameter Block

The parameter block consists of 8 long-words (8x32Bit; target byte order) and is used to exchange information to/from the target programming routine. The debugger is informed about the start address by the command FLASH.TARGET … (see above).

0	flash module start address	(FLASH.Create <address_range> …)
1	-	reserved
2	data bus width	(FLASH.Create …<bus_size>)
3	offset	(FLASH.OFFSET …)
4	address	start address in flash
5	size	size of data
6	timing	(FLASH.CLocK <frequency>) frequency in kHz
7	command	command (1=program, 2=block erase, 5=chip erase, 8=lock, 9=unlock)

The same parameter block returns information to the debugger after executing the programming routine:

0	-	not valid
1	-	reserved
2	-	not valid
3	-	reserved
4	address	address where error occurred
5	-	not valid
6	-	reserved
7	status	status (=0: no error, >10: error, 100=program error, 101=sector erase error, 102=bulk erase error, 103=lock error, 104=unlock error, 141=not implemented)

Example

Situation:

We have two flashs AM29F800 with 16-bit wide data bus parallel on the 32-bit wide data bus of the target processor. Each flash contains 1 MByte data. They will be accessed in the address range started at 200000H

We have also an external SRAM 8 KByte located from 0H to 1FFFH.

We have prepared the flash programming routine named “flashprog.bin” (size: 450H bytes; format binary). This function expect the parameter block in SRAM at address 101000H followed by the data buffer for programming data. There is enough memory available, so we want to use data buffer size of 4 KByte.

We want to load our application program “application.i3e” (size 1.4 MByte; format IEEE; located 200000- …) into the flash.

Required actions:

It is recommended to do the following steps in a script file (-> PRACTICE) to automate these settings:

1. Do the required settings, especially configure the chip selects that SRAM and flash can be really accessed in the address range described above.
2. Inform the debugger about the flash, location of target routine, parameter block and data buffer:

```
FLASH.Create 1. 0x200000--0x3fffff 0x20000 TARGET LONG
FLASH.TARGET 0x1200 0x0 0x1000 flashprog.bin
```

3. Erase the complete flash (assuming that there is no other flash in the flash list; otherwise the flash should be specified by the unit number oder address range e.g. “FLASH.Erase 1.”, “FLASH.Erase 200000—3fffff).

```
FLASH.Erase ALL
```

4. Program the application program

```
FLASH.Program ALL
Data.LOAD.Ieee application.i3e
FLASH.Program OFF
```

How it works:

Memory map in SRAM:

```
0000--001f    parameter block
0020--101f    data buffer
1020--111f    stack
1200--164f    programming routine
1650--165f    debug stub for breakpoint
```

On “FLASH.Erase ALL” the programming routine will be started with no valid content in the data buffer and with the following data in the parameter block:

```
0     00200000H     start address of flash module
1     -
2     00000004H     data bus width (1=8Bit, 2=16Bit, 4=32Bit, 8=64Bit)
3     -
4     00200000H     start address to begin erasure
5     00200000H     number of bytes to erase
6     -
7     00000005      command: 5=chip erasure
```

The programming routine returns the control to the debugger and sets the following parameter in the parameter block:

```
0     -
1     -
2     -
3     -
4     <address>     address where error occurred (error case only)
5     -
6     -
7     <status>      status: 0: no error; >10: error
```

In case of “FLASH.Erase 1.” and “FLASH.Erase 200000—3fffff” the parameter would be the same.

A block erasure can be realized by using a smaller address range for the erase command e.g. “FLASH.Erase 240000—25ffff”.

On “Data.LOAD.Ieee application.i3e” the programming routine will be started with the first 4 KByte data in the data buffer and with the following data in the parameter block:

```
0     00200000H     start address of flash module
1     -
2     00000004H     data bus width (1=8Bit, 2=16Bit, 4=32Bit, 8=64Bit)
3     -
4     00200000H     start address to begin programming
5     00001000H     number of bytes to program = valid bytes in buffer
6     -
7     00000001      command: 1=programming
```

The programming routine returns the control to the debugger and sets the following parameter in the parameter block:

```
0     -
1     -
2     -
3     -
4     <address>     address where error occurred (error case only)
5     -
6     -
7     <status>      status: 0: no error; >10: error
```

Then it repeats these steps until all data are programmed or an error occurred.

Further Hints for Designing the target routine

The debugger sets a breakpoint at the end of the code. This is required to give the control back to the debugger.

Take care that the stackpointer points to the same position when starting and ending the routine. Otherwise a stack overflow or underflow occurs, because normally this function will be called repeatedly.

Take care that all settings required for program execution are done (e.g. watchdog, …).

Available binary versions and PRACTICE script file examples are supplied in the directory of the selected architectures "~~/demo/<*processor*>/flash/…".

If the parameter block will be set manually, the target routine can be debugged with the debugger's help.