How to Write your own FLASHFILE Algorithm

How to Write your own FLASHFILE Algorithm

Version 18-Nov-2020

FLASHFILE Programming

Target Controlled Flashfile Programming

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

When do I have to write my own target-based FLASHFILE algorithm?

- If the flash controller you want to program is not listed on **www.lauterbach.com/ylistnand.html**, then you will need to develop your own flashfile algorithm.

What do I need for my own target-based FLASHFILE algorithm?

- You need to write a programming routine that runs on the target processor.
- RAM for data buffer, argument buffer, and stack is required on the target.

What do I have to do for the target-based FLASHFILE algorithm?

1. Develop the “GETID/SAVE/LOAD/ERASE” functions which will run on the target processor.
2. Convert the function source code to a binary file.
3. Connect the above functions to the debugger by using the commands:

   FLASHFILE.CONFIG … and

   FLASHFILE.TARGET …
4. Use the commands **FLASHFILE.GETID**, **FLASHFILE.DUMP**, **FLASHFILE.LOAD** …, **FLASHFILE.Erase** …

   These commands will call the flash algorithm file routine with the appropriate parameters.

Declaration

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

FLASHFILE.CONFIG: The defined parameter values `<param_1>` to `[<param_4>]` are passed to the flash algorithm file to access the flash controller. For more details about the parameter block table, see **"Contents of the Parameter Block"**, page 5.

```
;                <param_1>   <param_2>   <param_3>    [<param_4>]
FLASHFILE.CONFIG 0x6E0000AC 0x6E0000B0 0x6E0000B4    0x0
```

FLASHFILE.TARGET: This command informs the debugger about the start address range (0x3000000++0x1FFF) of the flash algorithm file (./own_flash.bin), about the data range (0x30002000++0x1FFF) for the parameter block (size: 64bytes) plus the data buffer (size: 0x2000 - parameter block).

```
                 <code_range>        <data_range>     <own_algorithm_file>
FLASHFILE.TARGET 0x30000000++0x1FFF 0x30002000++0x1FFF ./own_flash.bin \
/KEEP
```

The data range must be bigger than the flash size of "64byte (param.) + 1page + 256byte (stack)". The data buffer size can be selected by the user. It determines the maximum number of bytes which are handled on one programming routine call. If the command is executed, then the debugger loads the "own_algorithm_file" to the start address and internally sets the software breakpoint code at the end of the code.

What does the Debugger do?

GETID:

If the command **FLASHFILE.GETID** is entered, then the debugger loads the parameter data (parameter block), sets the program counter to the start address of the routine and starts the program execution.

If the execution has reached the end of the code which is the software breakpoint code, and if the return information is OK, then the host TRACE32 takes the flash ID / type / page size / block size information, i.e. the values from the algorithm file.

READ (SAVE, DUMP):

If the command **FLASHFILE.SAVE** is entered, then the debugger loads the parameter data (parameter block), sets the program counter to the start address of the routine and starts the program execution.

If the execution has reached the end of the code which is the software breakpoint code, then the host TRACE32 checks the return information (see parameter block), and if the return information is OK, then the host TRACE32 takes data buffer contents as flash data.

LOAD:

If the command **FLASHFILE.LOAD** is entered, the debugger 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 the program execution.

If the execution has reached the end of the code, then the host TRACE32 checks the return information (see parameter block) and if the return information is OK, then the host TRACE32 repeats the same load procedure for the remaining data.

ERASE:

If the command **FLASHFILE.Erase** is entered, the host TRACE32 loads the parameter block with the defined erasing address range and executes the erase function in the flash algorithm file. Then the host TRACE32 checks the return information.

The parameter block consists of 8 unsigned long long (8x64Bit; target byte order) and is used to pass information to/from the target programming routine. The debugger is informed about the start address by these commands:

FLASHFILE.LOAD	Load files to NAND FLASH
FLASHFILE.Erase	Erase NAND FLASH
FLASHFILE.DUMP	Dump NAND FLASH
FLASHFILE.SAVE	Read data from the NAND FLASH main area and write the data to <file>.

When the above commands are being executed, then this parameter block is set on the target controller:

idx	Type	Description
0	flash module start address	FLASHFILE.LOAD aaa.bin <start_address> FLASHFILE.ERASE <start_address>++<size>
1	parameter_1	User-defined hex value for the FLASHFILE.CONFIG command
2	parameter_2	User-defined hex value for the FLASHFILE.CONFIG command
3	parameter_3	User-defined hex value for the FLASHFILE.CONFIG command
4	reserv	-
5	size of data	FLASHFILE.ERASE <start_address>++<size>
6	parameter_4	User-defined hex value for the FLASHFILE.CONFIG command
7	command	When executed, the command (refer to the table **below**) will set the value from the column to the right.

```
FLASHFILE.CONFIG <parameter_1> <parameter_2> <parameter_3> <parameter_4>
```

Command	Value
FLASHFILE.SAVE(FLASHFILE.DUMP)	0x11
FLASHFILE.LOAD /WriteBadBlocks	0x12
FLASHFILE.ERASE	0x13
FLASHFILE.UNLOCK	0x14

Command	Value
FLASHFILE.LOAD	0x18
FLASHFILE.ERASE /EraseBadBlocks	0x19
FLASHFILE.GETID	0x20

After the algorithm file has finished, the parameter block below returns this information to the debugger:

idx	Type	Description
0	-	not valid
1	-	reserved
2	-	not valid
3	-	reserved
4	-	returnValue, how many bytes were skipped as bad blocks (bytes)
5	-	not valid
6	-	reserved
7	status	status=0: no error. status>10: error: • 100=program error • 101=sector erase error • 102=bulk erase error • 103=lock error • 104=unlock error • 141=not implemented

Situation:

We have a NAND flash (page size 0x800, block size 0x20000) on the target processor. It is controlled only by the Nand Flash Controller Registers:

- Command Register: 0x61400000
- Address Register: 0x61200000
- Data Register: 0x61000000

We also have an internal/external SRAM 16 KByte located at 0x2000_0000H to 0x2000_3FFFH.

We have prepared the flash programming routine named “flashprog.bin” (size: 0x1450H bytes; format binary). So we can use half of the memory for the code (0x2000_0000++0x1FFF), and the other half for the data (0x2000_2000++0x1FFF).

This function expects the parameter block in SRAM at address 0x2000_2000H followed by the data buffer for programming and reading data.

We want to load our application program “application.bin” (size 1.0 MByte; format binary; located 0x1000- ...) into the flash.

Required actions:

It is recommended to do the following steps in a PRACTICE script file (*.cmm) to automate these settings:

1. Make the required settings. Especially configure the flash controller and S(D)RAM.
2. Inform the debugger about the flash controller register, and location of the algorithm file routine, parameter block and data buffer:

```
FLASHFILE.RESet
//               <cmd_reg>  <addr_reg>  <data_reg>
FLASHFILE.CONFIG 0x61400000 0x61200000  0x61000000

//               <code_range>       <data_range> <algorithm_file>
FLASHFILE.TARGET 0x20000000++0x1FFF 0x20002000++0x1FFF   \
                                                 flashprog.bin /KEEP
```

3. Get the ID values, page size, block size, and the NAND Flash ID from the flash device:

```
AREA.view
FLASHFILE.GETID
```

4. Read/save the NAND flash data:

```
FLASHFILE.DUMP  0x1000 ;or FLASHFILE.SAVE aaa.bin  0x1000++0xFFFFF
```

5. Erase the flash.

```
FLASHFILE.Erase 0x1000++0xFFFFF /EraseBadBlocks
```

6. Program the application program.

```
FLASHFILE.LOAD application.bin  0x1000  /WriteBadBlocks
```

Memory Map in SRAM:

```
0x2000_0000--0x2000_1FFF   flashprog.bin + software breakpoint
0x2000_2000--0x2000_203F   parameter block
0x2000_2040--0x2000_383F   data buffer, flash page size aligned
                           (example: 0x800 )
0x2000_3840--0x2000_393F   stack for the flashprog.bin
```

FLASHFILE.GETID

When **FLASHFILE.GETID** is executed, the programming routine will be started without valid content in the data buffer and with the following data in the parameter block:

```
0x2000_2000
0 -
1 0x61400000 H
2 0x61200000 H
3 0x61000000 H
4 -
5 0x40   (size of data to read)
6 0
7 0x20 command: FLASHFILE.GETID
```

The programming routine returns the control to the debugger and sets the following parameter in the parameter block and data buffer for use by the host TRACE32:

```
0x2000_2000 H
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 <status> status: 0: no error; >10: error
```

Provided the above return information is OK (no error), the host TRACE32 takes the flash ID / type / page size / block size information:

```
0x2000_2040H    0x00D300EC ( Device_ID<<16 | Manufacturer_ID )
0x2000_2044H    -
0x2000_2048H    -
0x2000_204CH    -
0x2000_2050H    0x800 (NAND page size)
0x2000_2054H    0x40 (NAND spare size)
0x2000_2058H    0x20000 (NAND block size)
0x2000_205CH    0x1 (Flash type)

Flash type      index
NAND            0x01
SPI             0x02
MMC / EMMC      0x03
ONENAND         0x04
I2C             0x05
SD              0x06
HYPER           0x07
```

FLASHFILE.SAVE

When **FLASHFILE.SAVE** aaa.bin 0x1000++0xFFFF is executed, the programming routine will be started without valid content in the data buffer and with the following data in the parameter block:

```
0 0x1000  (start addr)
1 0x61400000 H
2 0x61200000 H
3 0x61000000 H
4 -
5 0x1800   (size of data to read, page size aligned)
6 0
7 0x11 command: FLASHFILE.SAVE
```

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

```
0x2000_0000 H
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 <status> status: 0: no error; >10: error

data buffer
0x2000_0040--0x2000_17FF    the flash memory contents
```

Then the flash algorithm file repeats these steps until all data is read (example: 0x1000++0xFFFF) or an error occurred. And if the status is OK, the host TRACE32 finally generates the file aaa.bin with data from the read buffer.

FLASHFILE.Erase

When **FLASHFILE.Erase** 0x1000++0xFFFFF is executed, the programming routine will be started without valid content in the data buffer and with the following data in the parameter block:

```
0 0x1000
1 0x61400000 H
2 0x61200000 H
3 0x61000000 H
4 -
5 0x100000   ( size of data to read )
6 0
7 0x19 command: FLASHFILE.Erase  /EraseBadBlocks
```

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

```
0x2000_0000 H
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 <status> status: 0: no error; >10: error
```

When **FLASHFILE.LOAD** application.bin 0x1000++0xFFFFF /WriteBadBlocks is executed, the programming routine will be started with the application.bin contents in the data buffer and with the following data in the parameter block:

```
0 0x1000  (start addr)
1 0x61400000
2 0x61200000
3 0x61000000
4 -
5 0x1800   ( size of data to program, page size aligned )
6 0
7 0x18 command: FLASHFILE.LOAD /WriteBadBlocks

data buffer
0x2000_0040--0x2000_17FF    (the application.bin contents)
```

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

```
0x2000_0000
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 <status> status: 0: no error; >10: error
```

Then the flash algorithm file repeats these steps until all data is programmed (example: 0x1000++0xFFFFF or 0x1000--0x100FFF) or an error occurred.