# MakeRom

## Aim

It generates the ROM content files for production, with the checksum information required by ***TestRom*** procedure of LibSys (read ***SamVS-C System Library.pdf*** documentation) and ***_NFM_TestNAND*** procedure of LibNFM.

## Input

It takes as input a description text file, which sets the file(s) to use, what destination address...
Limitation: each file size must not be greater than 4GB-1. Cut it in pieces if greater.
The description text file has the following content:

### *Directives*

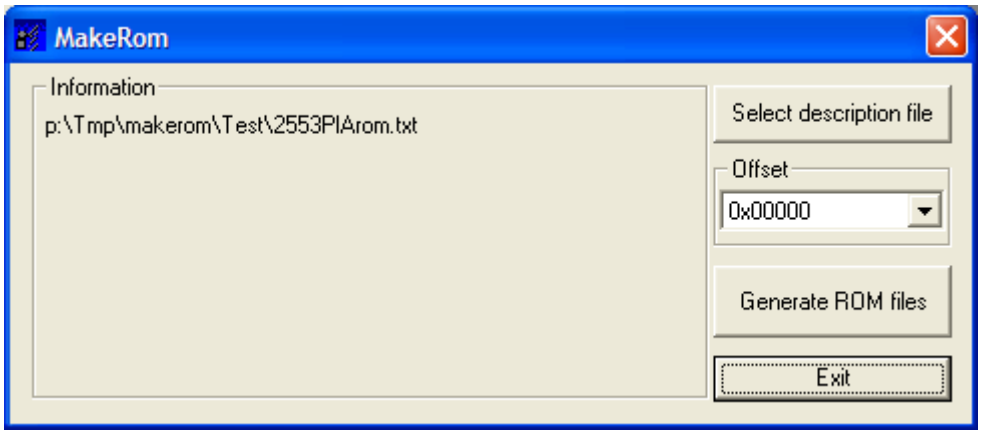
| | |
|---|---|
| **$output** *NameFile* | It generates several files, named *NameFile*00.bin *NameFile01*.bin…<br>In case of $**octal** used, ***NameFile00L.bin***, ***NameFile00H.bin***, …<br>***Default value*** *ROMfile*. |
| **$devicesize** *0x400000* | Set the ROM device size in words: 0x200000 for a 32Mbits. ***Default value 0x200000*** |
| **$programableamount** *80* | **%** amount of device to receive data. ***Default value 100; 80 in case of NandFlash.*** |
| **$pattern** *0x600D* | Set the word value to use for gap/padding area(s). ***Default value*** *0xFFFF.* |
| **$target** *2553* | Specify the target name: 3516 (SAM3516), 3716 (SAM3716), 3816 (SAM3816), 3703 (SAM3703), 2533 (SAM2533), 2553 (SAM2553), 2634 (SAM2634), 2635 (SAM2635), 2653 (SAM2653), 2655 (SAM2655), 3103 (SAM3103), 3303 (SAM3303), 3108 (SAM3108), 3308 (SAM3308), 5704 (SAM5704), 5716 (SAM5716), 5808 (SAM5808), 5916 (SAM5916), 5504 (SAM5504), 5708 (SAM5708)…<br>***Default value*** *5916* for SAM5916 |
| **$directory** *Nameofdirectory* | Specify a directory to apply to the following common entries. Use as many time as needed. |
| **$octal** | Memory devices are used by pair: two 8bits values read from two memories generate one 16 bits value. |
| **$merge** | Join together each element, adding padding between if needed (value set by $pattern directive): it then does not care neither of device size nor start offset nor target nor checksum nor octal mode. |
| **$nandflash *spareinclude*** | To generate file(s) with DREAM ECC for NandFlash device external programmer. By default, it generates a file (*.img) for *User data* area and a file (*.spr) for *Spare data* area. Option '**spareinclude'** generates one file with *User data* area and *Spare data* area interlaced. |
| **$nandflashpageusersize** *0x400* | To define in word the size of the *User data* area of the NandFlash page. ***Default value*** *0x400* (1k words). |
| **$nandflashpagesparesize** *0x70* | To define in word the size of the *Spare data* area of the NandFlash page. ***Default value*** *0x70* (112 words). |
| **$nandflashpage4block** *0x40* | To define the number of page inside a block of the NandFlash. ***Default value*** *0x40* (64 pages per block). |

### *Common entry*

*Namefile* *AbsoluteAddress* *OffsetInFile* *Length*

| | | |
|---|---|---|
| *OffsetInFile* | *(optional)* | the word offset from the file *NameFile*. ***Default value 0***, |
| *Length* | *(optional)* | the word length from the file *NameFile*.By default the size of the file or ('size of file' –*offsetInFile*) if *OffsetInFile* defined. |

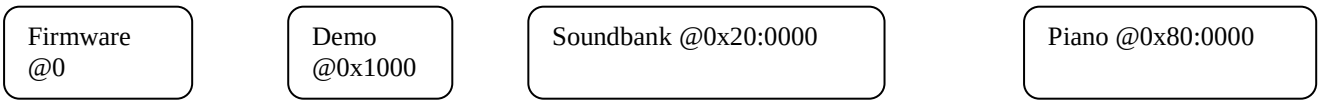
It takes, at word offset *OffsetInFile*, *Length* words data from the file *NameFile*, and it maps them at the destination address *AbsoluteAddress*.
*Namefile* can be a file name with extension or a full path name file.



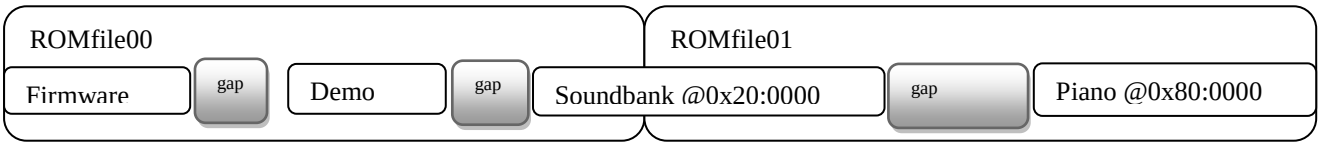

## Output

Button ‘Generate ROM files’ creates the .bin file(s) and a .log file (in case of errors, remarks).

From the following input description,




it generates:




### *‘Offset’ parameter*

By default, it generates the file from address 0x0000. The generated files are to be used with external memory programming device, that writes entire memory chip.
With the parameter ‘*Offset*’, the first generated file (named ROMfile00**z**.bin) will start at specified address for Offset: If need to directly program the memory board, with **ProgSam**.

*Remark:*

- **MakeRom** does not support overlapping files: each entry definition must **NOT** overlap other entry definition. It reports this situation as error.

- a firmware checksum is computed, at word location ‘end of firmware file -1’ (the last word).
- In case of target 3416, 3516, 3716 and 3816, the low part (the first 0x4000 words) of the firmware (i.e. for the file set at address 0x0000) is not copied in the output file (mandatory for a secured firmware): it is replaced by a gap area (using the value defined by $pattern).
- In case of target 2533, 2553, 2635, 2653 and using a specific firmware, that requires the use of the tool *94split4*, **MakeRom** is doing the same job as *94split4* tool: Message “*Detected 'FIRM5533' inside firmware. Act as 94Split4.*”

## *'octal'* mode

By default, files-devices are 16 bits values data. Organized as follow:

| | | |
|---|---|---|
| File#0 (16 bits value)<br>@0 | File#1 (16 bits value)<br>@DeviceSize | File#2 (16 bits value)<br>@2*DeviceSize |

In case of **$octal** used, files-devices are organized to read one 8 bits value from each two memory devices to generate one 16 bits value data. Organized as follow:

| | |
|---|---|
| File#0-H (8 bits value)<br>@0 | File#1-H (8 bits value)<br>@2*DeviceSize |
| File#0-L (8 bits value)<br>@0 | File#1-L (8 bits value)<br>@2*DeviceSize |

***Warning***: File#-H must be written inside the first memory device connected to SAMx. File#-L must be written inside the second memory device connected to SAMx

## *'merge'* mode

Join together each element, according to their destination address, add padding (padding value sets with **$pattern**). Directives Target, device size, start offset, checksum and octal mode are ignored.

## *'nandflash'* mode

NandFlash structure is defined by its U*ser data* area size (directive **$nandflashpageusersize**) and its *Spare data* area (directive **$nandflashpagesparesize**).
**$devicesize** is the total, in word, of all the *User data* area pages for the NandFlash device.
$**programableamount** is the amount (in percent) of blocks to receive data, on one NandFlash device. Default value is 80; the remaining 20% are for bad blocks, system reserved blocks and pool of free blocks for wear-leveling mechanism done by DREAM firmware. High limit of this parameter is 95%. See below

Due to the logical address detection done by DREAM firmware, by reading protected data with a valid DREAM ECC inside block, NandFlash blocks that do not received data, must be left empty. Therefore:

- **$pattern** is used only for gap between entries files,
- It does not generate result file(s) with a size of **$devicesize** words.
  The size is up to **$devicesize** * **$programamount**/100, which is the limit for each NandFlash result file.
  If there are less data as entry, the result file is the size of the entry files and their gaps.

Requirements
*NandFlash structure*:
DREAM ECC requires *User data* area page to be 1k words and *Spare data* area page at least 32 words ***or*** 2k words for *User data* area and at least 64 words for *Spare data* area.
Programmable amount of data (directive **$programableamount)** is, at maximum 95%: to have a minimum of pool of free blocks for wear levelling and the system reserved blocks.

In this mode, **$octal** and **$merge** directives are ignored.

Remark:
External programming softwares accept entry files as:

- Two files: One file (*.img) for *User data* part and one file (*.spr) for *Spare data* part.
- Or only one file, with *User data* area and *Spare data* area interlaced: file *.bin

The directive **$nandflash** will create separated files *.img & *.spr
The directive **$nandflash spareinclude** will create one interlaced file *.bin.

## Important:

- The programming mode '***skip bad block***' must be chosen in the external programming software.
- In case of **4kbyte** page NandFlash, after programming data with the external programming software, a 'Verify' operation can NOT be directly used. Also the 'detect bad block' operation will report many physical block as bad. Due to the Sam5x NandFlash controller added ECC area management, in case of 4k byte page, the 'bad block marker' area of the NandFlash (located at the beginning of the *Spare data* area) is overlapped by the written data: external programming software often checks the 'bad block' state of a block before reading it; Therefore, it will report some written data blocks as 'bad block' but those blocks are good and written with the expected data.
For example, from an empty 4k NandFlash, with NO factory bad block, after writing the data, 'Verify' operation can be used only if 'detect bad block' is disabled in the external programming software.
In the case of an empty 4k NandFlash, with factory bad block, after writing the data, 'Verify' operation can NOT be used: it will detect block as bad but they have data inside. With 'detect bad block' disabled, it will read back bad block as usefull data to compare with original data.
SAM5x firmware correctly detects the 'bad block' state for all physical blocks.
- in case of **2kbyte** page NandFlash, after programming data with external programming software, 'Verify' operation and 'detect bad block' operation are usable and report accurate information.

## How to have maximum data amount of 95%

A simple solution is to use as highest logical address value for data (where no more data after this address), to 95% of the total NandFlash size.
For example, a solution with two NandFlash of 8 Gbit each, with a block size of 256 kbyte, the highest logical address value is 0x3C00:0000 in word (95% of 2*0x2000:0000, truncated to the near low multiple of block size 0x2:0000).

## Information about GAP / padding area

Those areas have got the data defined by the directive **$pattern**.
When the generated MakeROM file is written to NandFlash, these areas are assumed to be data and 'consume' NandFlash blocks. Be sure to reduce the size of these GAP/padding area, as possible.