NAND Flash Memories

–

Using Multiply Partition (with Skip IB)

on ELNEC Device Programmers

–

(Quick Guide)

Application Note

November 2018
an_elnec_nand_partitions, version 1.11

Increasing demands on embedded devices involve also increased complexity of programming tasks. Embedded memory of today does not store just simple piece of firmware. It carries various types of data used by modern operating system driven appliances. Typically, there are some kind of boot-loader, operating system executables and user data stored in single-chip memory unit. Within NAND flash memory devices, management of these mixed-type data is considerably impacted by invalid blocks presence. Correct partitioning of NAND flash memory therefore becomes crucial element of many embedded devices designs.

ELNEC programmers are able to cope with NAND flash partitioning tasks. This application note is aimed to help you to understand how to use their features to fulfil your needs.

Actually, NAND flash partitioning is supported on BeeProg2, BeeHive204, BeeHive204AP, BeeHive208S, BeeProg3 and BeeHive304 programmers.

This application note applies to area under dynamical development. Please, check our application notes section (see http://www.elnec.com/support/application-notes/) regularly and always use the newest version of the document.

If you recognize the apparatus described in this application note not suitable for your demand, you can consider also using our Multi-Project feature described in separate application note (see http://www.elnec.com/sw/an_elnec_multi_prj.pdf).

OVERVIEW

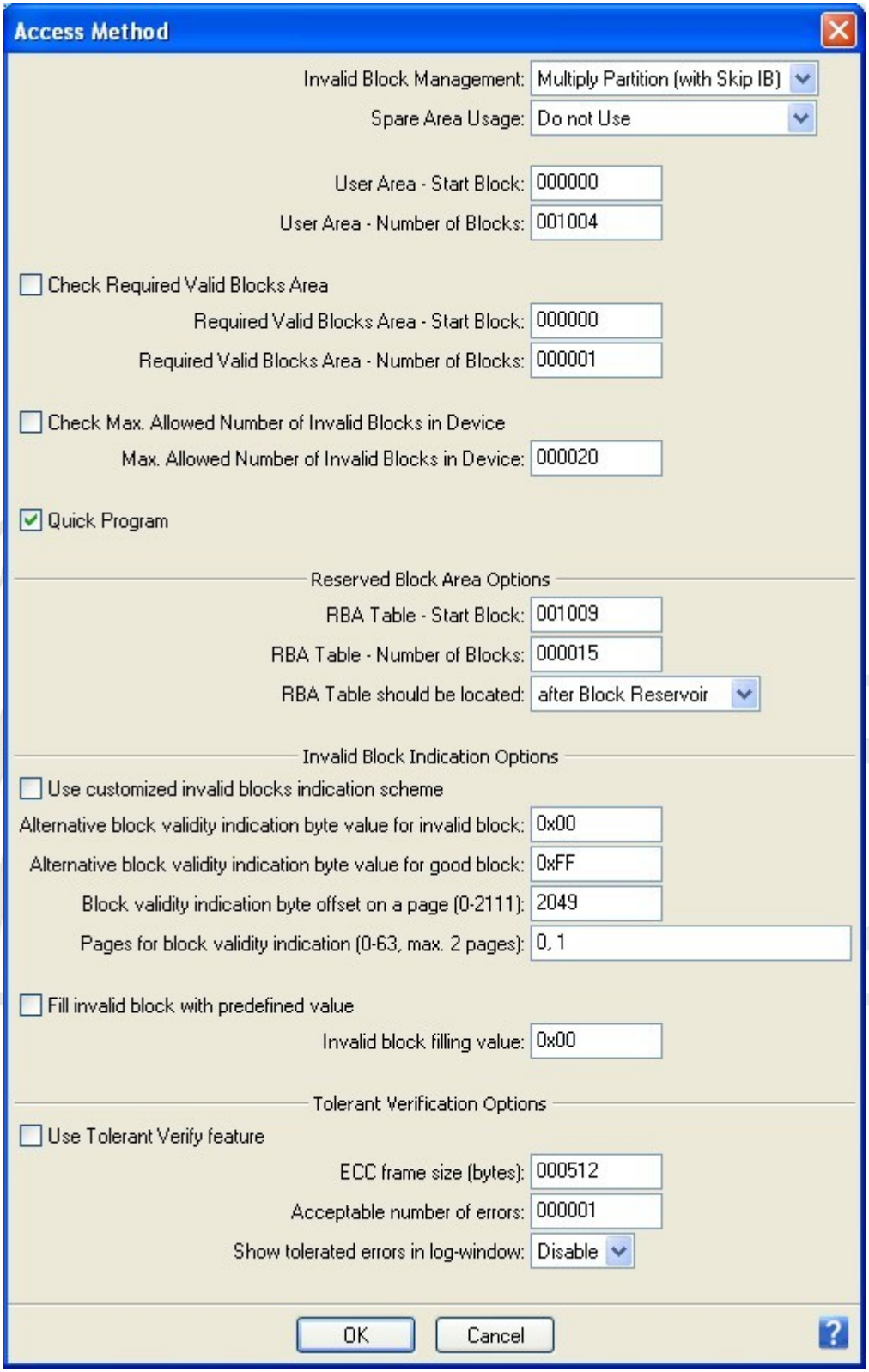

Figure 1. Access Method dialog window (<Alt+S>) with Multiply Partition set

To enable Multi-Partition feature, open **Access Method** dialog window (easy accessible through clicking the link in **Device** panel in bottom right of **pg4uw** software window, or shortcut **<Alt+S>**) and set **Invalid Block Management** option to **Multiply Partition (with Skip IB)**.

ACCESS METHOD OPTIONS VALIDITY

“With Skip IB” means that invalid blocks are skipped and next valid blocks are used instead. This is the only invalid blocks management method available in partition and is used automatically.

Setting of **Spare Area Usage** option is ignored. Spare area is always used and input data file(s) must contain correct data for spare area. Spare area is programmed using method corresponding to User Data setting. If your input data file doesn't contain spare area data (e.g. your system doesn't use spare area), you are still required to add empty (blank) spare area data. You can do it on your own, or you can use **Add blank spare area** feature available in **Load file** dialog window.

Setting of **Required Good Blocks Area** is accepted and is applied globally (i. e. over whole device, not over individual partitions).

Setting of **Max. Allowed Number of Invalid Blocks in Device** is accepted and is applied globally.

Reserved Block Area Options are irrelevant when skipping invalid blocks and respective setting is not taken into account.

Setting of **Invalid Block Indication Options** is accepted and is applied globally.

Setting of **Tolerant Verification Options** is accepted and is applied globally.

OPERATIONS EXECUTION

Blank check

The device is always blank checked entirely, from the first up to the last address location, including spare area.

If **Blank check before programming** is enabled (and is not skipped due to enabling both, Blank check and Erase before programming), it will be performed in the same way, not taking partitioning into account.

Read

Read operation is performed on per partition basis, i. e. individual partitions are processed step by step, from the first partition up to the last one.

Note: Partition unused (padding) area is not read. Buffer data at corresponding locations are not altered.

Verify

Verify operation is performed on per partition basis, i. e. individual partitions are processed step by step, from the first partition up to the last one.

If **Verify after reading** or **Verify after programming** is used, the partition will be read or programmed firstly, and after then it will be verified, see 2.

Note: Partition unused (padding) area is not verified. Buffer data at corresponding locations are ignored.

Program

Program operation is performed on per partition basis, i. e. individual partitions are processed step by step, from the first partition up to the last one. See 2 for programming round example.

Note: Partition unused (padding) area is not programmed. Buffer data at corresponding locations are ignored.

Erase

The device is always erased entirely, from the first up to the last block

If **Erase before programming** is enabled, it will be performed in the same way, not taking partitioning into account.

```
L0278: >> 17.12.2009, 15:34:52
L0279: Checking the Partition Table...
L0280: Checking the Partition Table - O.K.
L0281:
L0282: ---------------------------
L0283: ====  Partition Table  ====
L0284: ---------------------------
L0285:
L0286:  START   | END     | USED
L0287: ---------+---------+---------
L0288:  0x0000 | 0x0064 | 0x0014
L0289:  0x0065 | 0x00C8 | 0x0032
L0290:  0x00C9 | 0x012C | 0x0000
L0291:  0x012D | 0x0190 | 0x0032
L0292:  0x0191 | 0x01F4 | 0x0000
L0293:  0x01F5 | 0x03E8 | 0x0032
L0294:
L0295:
L0296: >> 17.12.2009, 15:34:56
L0297: Programming device: Samsung K9F1G08U0M [TSOP48].
L0298: SN#1: 1105-00001 (878/15000).
L0299: Converter test - O.K.
L0300: Device insertion test ...
L0301: Checking device ID ...
L0302: Building IB info table...
L0303: 0 invalid block(s) found.
L0304: Erasing device ...
L0305: Partition 0:
L0306: Programming device ...
L0307: Verifying device with buffer ...
L0308: Partition 1:
L0309: Programming device ...
L0310: Verifying device with buffer ...
L0311: Partition 2: Not in use
L0312: Partition 3:
L0313: Programming device ...
L0314: Verifying device with buffer ...
L0315: Partition 4: Not in use
L0316: Partition 5:
L0317: Programming device ...
L0318: Verifying device with buffer ...
L0319: All partitions processed...
L0320: Programming device - O.K.
L0321: Elapsed time: 0:00:27.3
L0322: Statistics info: Success:3  Failure:0  Total:3
```

Figure 2. Program operation round example

GENERAL PROCEDURE DESCRIPTION

Setting-up the Multi-Partition mode consists of three basic steps:

Selection of Multiply Partition

Open **Access Method** dialog window and set **Invalid Block Management** option to **Multiply Partition (with Skip IB)**, see Figure 1.

Set other options, if necessary.

After clicking OK, internal software environment will be prepared for Multi-Partition mode. Also, special buffer for Partition Table will be created, see Figure 3. The buffer is not editable, the only way how to define partition table is loading relevant definition file.

After Multiply Partition method is deselected, internal software environment is switched back to standard mode and special buffer is destroyed. In consequence, partition table data are lost and must be load again after re-enabling Multi-Partition mode again.

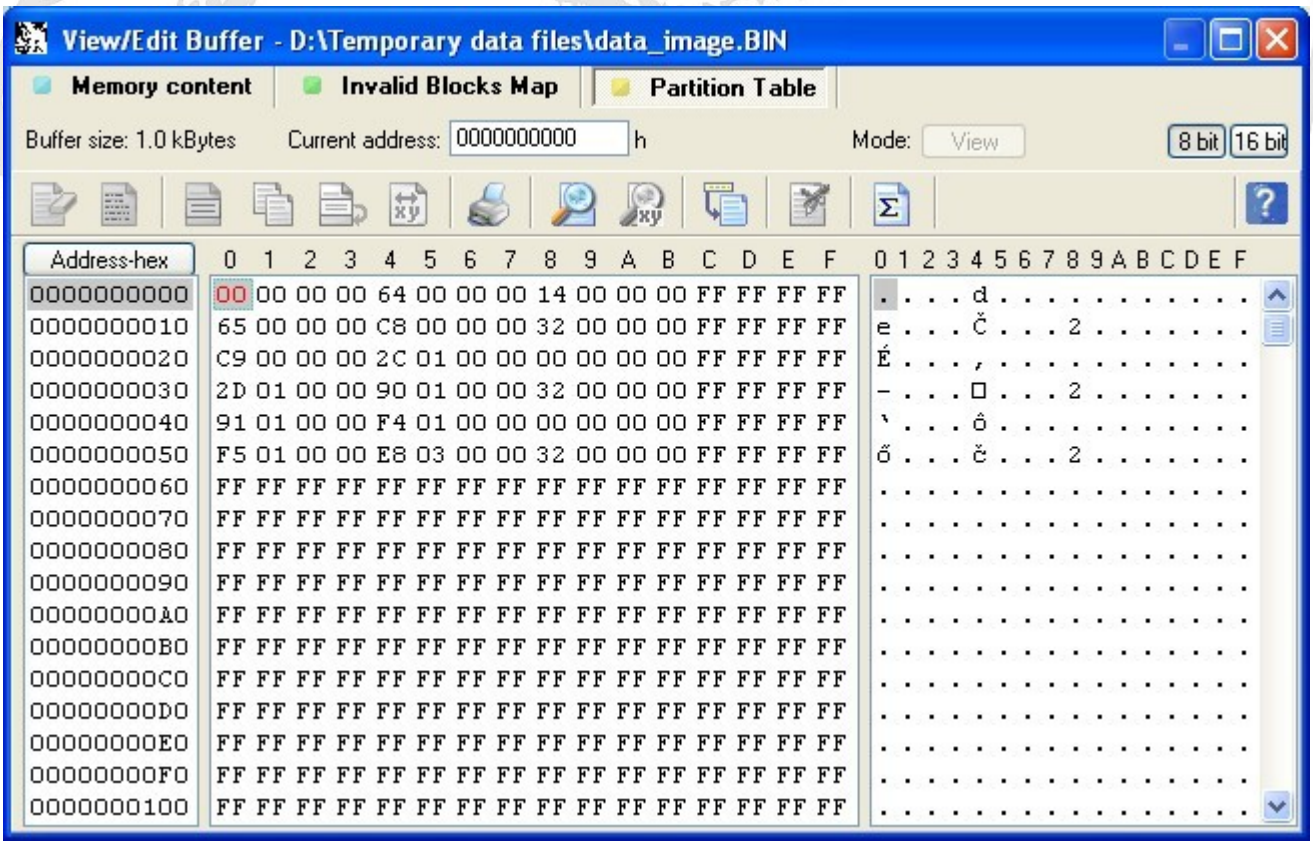

Figure 3. Partition Table buffer example

Loading partition table definition file

After enabling Multi-Partition mode, the partition table must be defined. There is only one way how to define the partition table - using menu **File >> Load Partition table...**, see Figures 4, 5 and 6. However, there are several partition definition file formats supported, see later chapters for more details.

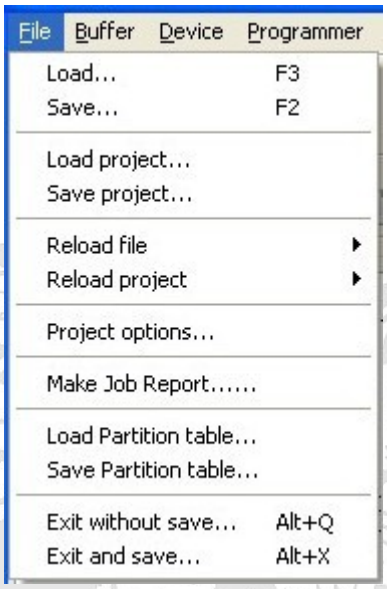

Figure 4. Load Partition table menu

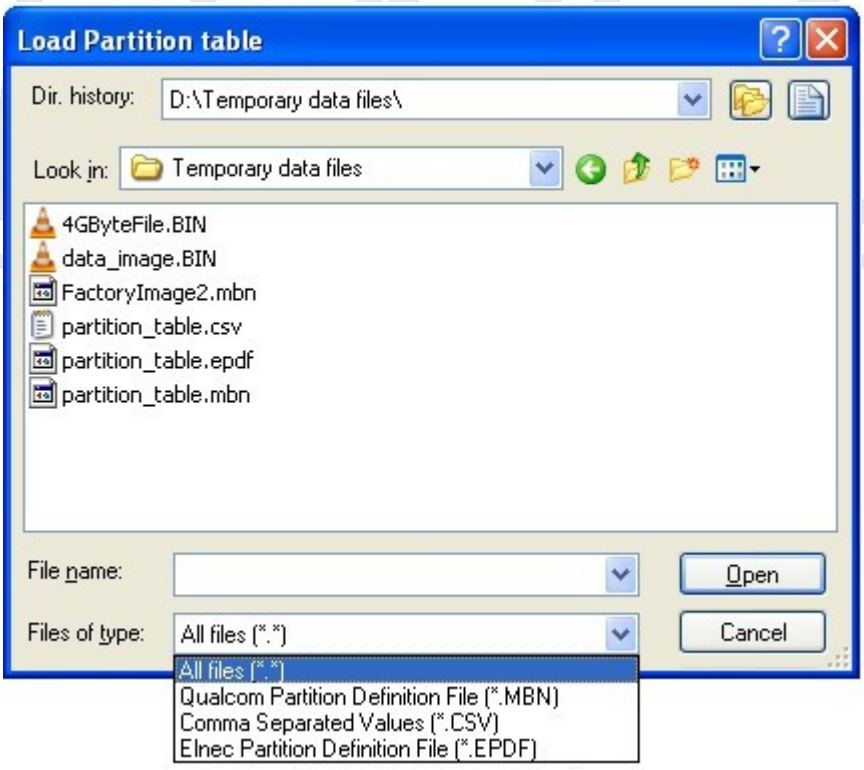

Figure 5. Load Partition table dialog

```
L0250: >> 17.12.2009, 12:46:07
L0251: Loading file: D:\Temporary data files\partition_table.csv
L0252: File format: Comma Separated Values
L0253: Loading Partition Table...
L0254: Partition 0
L0255: Partition start (in blocks): 000000 (0x0000)
L0256: Partition end (in blocks):   000100 (0x0064)
L0257: Partition size (in blocks):  000020 (0x0014)
L0258: Comment:                     boot
L0259: Partition 1
L0260: Partition start (in blocks): 000101 (0x0065)
L0261: Partition end (in blocks):   000200 (0x00C8)
L0262: Partition size (in blocks):  000050 (0x0032)
L0263: Comment:                     exec
L0264: Partition 2
L0265: Partition start (in blocks): 000201 (0x00C9)
L0266: Partition end (in blocks):   000300 (0x012C)
L0267: Partition size (in blocks):  000000 (0x0000)
L0268: Comment:                     resl
L0269: Partition 3
L0270: Partition start (in blocks): 000301 (0x012D)
L0271: Partition end (in blocks):   000400 (0x0190)
L0272: Partition size (in blocks):  000050 (0x0032)
L0273: Comment:                     fsys
L0274: Partition 4
L0275: Partition start (in blocks): 000401 (0x0191)
L0276: Partition end (in blocks):   000500 (0x01F4)
L0277: Partition size (in blocks):  000000 (0x0000)
L0278: Comment:                     res2
L0279: Partition 5
L0280: Partition start (in blocks): 000501 (0x01F5)
L0281: Partition end (in blocks):   001000 (0x03E8)
L0282: Partition size (in blocks):  000050 (0x0032)
L0283: Comment:                     data
L0284: File loading successful.
```

Figure 6. Partition table definition file load example

Loading data image file

If relevant chapter intended to individual partition definition file formats doesn't specify otherwise, data image file should correspond with a copy of NAND flash device without any invalid blocks. It must contain data for all partitions placed at correct locations. Also, it must contain spare area data (or you can add spare area data automatically on file load using **Add blank spare area** feature).

To load input data image file, use standard Load command (menu **File >> Load**, shortcut **<F3>** or **Load** command from **Main toolbar**).

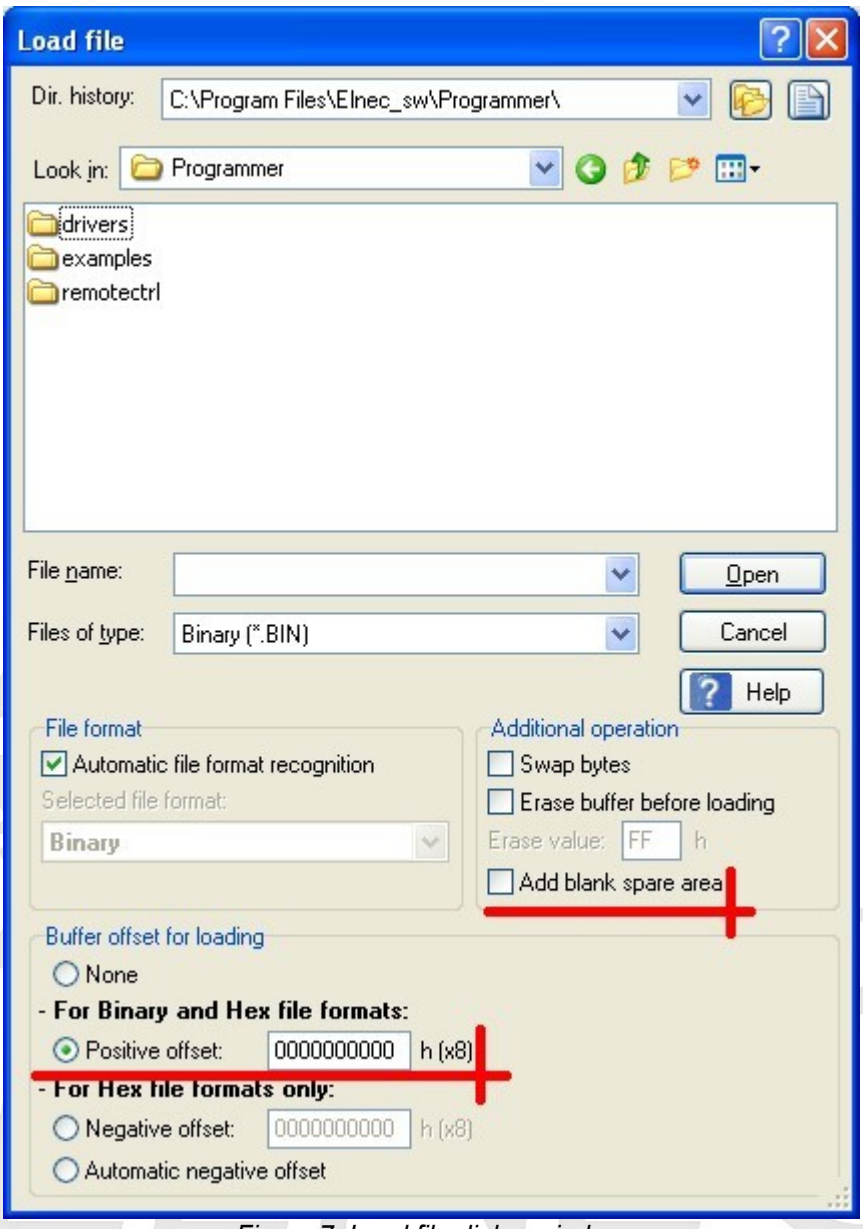

Figure 7: Load file dialog window

If you haven't a single input data image file but you have several input data files for individual partitions instead, you will need to use **Positive offset** setting in File load dialog. You can compute the offset using the following formula:

positive_offset = partition_start_block_number x number_of_pages_in_block x page_size_including_spare

In this way, you can load multiple files into buffer, each one at respective offset corresponding to partition start block address offset.

Example: Partition should start from block 30, devices has 64 pages in block and one page contains 2048 bytes of data area and 64 bytes of spare area.

positive_offset = 30 x 64 x (2048+64) = 4055040_{dec} = 3DE000h

USING QUALCOMM MULTIPLY PARTITION FORMAT (*.MBN)

Generally, there are two versions of Qualcomm Multiply Partition format used. They can be simply distinguished by the number of programming files:

Procedure for two input files (Qualcomm 80-VE594-1 B)

If you have two input files available, they are generally named FactoryImage.bin and PartitionTable.mbn.

PartitionTable.mbn is rather small (typically 256 bytes) and contains partition table definition. Load this file using menu **File >> Load Partition table...**, see chapter Loading partition table definition file. Actually, the maximum count of supported partitions is 64 (MBN file size of 1040 bytes).

FactoryImage.bin may be rather huge and contains binary data image. Load this file using standard Load procedure, see chapter Loading data image file.

It is possible to save your data using this format. To save buffer content in binary format, use standard Save procedure (menu **File >> Save**, shortcut **<F2>** or **Save** command from **Main toolbar**). To save partition table in Qualcomm Multiply Partition compatible form, use menu **File >> Save Partition table...,** see Figure 4.

Procedure for single input file (Qualcomm 80-VF498-1 A)

If you have single input file available, it is generally named FactoryImage2.mbn. The file is rather huge and contains both, partition table definition and binary data image, plus header. The file can be simply identified using hex-viewer: you must identify text “Image file with header” at file start.

The header specifies also block validity indication byte position. This parameter is also accepted and used for proper reading and/or verifying the device. The value overwrites manual setting in **Invalid Block Indication Options** section of **Access Method** dialog.

Load this file using standard Load procedure, see chapter Loading data image file.

It is not possible to save your data using this form.

Note: Qualcomm Multiply Partition format was invented by Qualcomm Incorporated (USA), not by ELNEC. The owner of all potential legal rights is Qualcomm Incorporated.

USING COMMA SEPARATED VALUES FORMAT (*.CSV)

This mode uses two input files. You can simply prepare both of them on your own:

Partition table definition file

Partition table definition file uses widely used comma separated values file format.

The file should contain the number of rows corresponding to the number of partitions. Each row specifies one partition.

Values in row should be separated by separator - comma (,) or semicolon (;) can be used. Space characters (ASCII code 0x20) are ignored and should not be used in place of values separator.

Each row should contain several values (both, decimal and/or hexadecimal values can be used):

- **partition start** (mandatory) - specifies the block in device, where partition should start. Enter the block number here.
- **partition end** (mandatory) - specifies the block in device, where partition should end. Enter the block number here.
- **used partition size** (mandatory) - specifies the number of blocks really occupied by partition data. Typically, there are some reserve blocks added for invalid blocks replacement, therefore partition end - partition start > used partition size. Enter number of blocks here.
- **special options/reserved** (optional/mandatory) - this value enables to specify some special options. If you use it just due to **comment** option, enter the value of 0xFFFFFFFF (4 bytes size) here to ensure future compatibility.

  Special options format specification:

  ```
  MSB(bit 31)                    LSB(bit 0)
  xxxx.xxxx.xxxx.xxxx.xxxx.xxxx.xxxx.xxxx
  ```

  bits 11:0 - Maximum allowed number of invalid blocks in partition:

  0xFFF = option disabled

  any other value specifies the number of invalid blocks that can be accepted in partition

  bits 15:12 - Invalid blocks management method:

  0x0 = Treat All Blocks

  0x1 or 0xFF = Skip IB (default)

  0x2 = Skip IB with Excess Abandon

  0x3 = Check IB without Access

  0x4 = Discard Invalid Block(s) data

  Note: Check IB with Skip IB can be specified using Skip IB (0x1 or 0xF) method and non 0xFFF value for Max. allowed number of invalid blocks in partition.

bits 22:16 - reserved for future use, consider 0x7F value for future compatibility

bit 24 - First block in partition must be good:

0b = if first block in any partition is invalid, device is considered bad and operation is aborted

1b = feature disabled

bits 27:24 - File system preparation:

0xFF = option not used

0x00 = JFFS2 Clean Markers using MSB byte ordering (big endian)

0x01 = JFFS2 Clean Markers using LSB byte ordering (little endian)

bits 31:28 - reserved for future use, consider 0xF value for future compatibility

Using values not specified here will cause partition table load error.

– **comment** (optional) - you can enter any text here. Primarily, this item is intended for your notes that will help you to orientate in the file. It may contain e. g. partition name. If you use comments, **reserved** item must be also used.

Partition table definition file listening example:

```
0;100;20;0xffffffff;boot
101;200;50;0xffffffff;exec
201;300;0;0xffffffff;res1
301;400;50;0xffffffff;fsys
401;500;0;0xffffffff;res2
501;1000;50;0xffffffff;data
```

Load this partition table definition file using menu **File >> Load Partition table...**, see chapter Loading partition table definition file.

It is possible to save your partition table definition using this format. To save the partition table data, use menu **File >> Save Partition table...,** see 4. The table is saved using all items in raw. A partition number is used for comment.

Saved partition table definition file listening, using the above example:

```
0;100;20;0xFFFFFFFF;Partition 0
101;200;50;0xFFFFFFFF;Partition 1
201;300;0;0xFFFFFFFF;Partition 2
301;400;50;0xFFFFFFFF;Partition 3
401;500;0;0xFFFFFFFF;Partition 4
501;1000;50;0xFFFFFFFF;Partition 5
```

Data image file

Input data image file should be a binary file (recommended) that meets all requirements specified in chapter Loading data image file. Use standard Load procedure to load this file (see chapter Loading data image file).

USING GROUP DEFINE FILE FORMAT (*.DEF)

Important note!

The support of this format is implemented based only on fragment of specification available from customer. Therefore it cannot be considered full and reliable. We do not recommend to use it, until you exactly know what you do. If you observe any problems, please, contact our technical support with full Group Define file format specification.

Partition table definition file

This file consists of file header and group records. Each group record specifies one partition.

Load this partition table definition file using menu **File >> Load Partition table...**, see chapter Loading partition table definition file.

It is possible to save your partition table definition using this format. To save the partition table data, use menu **File >> Save Partition table...,** see 4.

Data image file

Input data image file should be a binary file (recommended) that meets all requirements specified in chapter Loading data image file. Use standard Load procedure to load this file (see chapter Loading data image file).

ERROR CODES

During input file(s) loading, several errors can occur. Errors are always displayed in **pg4uw** log-window. Error message consists of error code and error description in the following form:

File loading problem!

Error code: #xxyy - error description

where **xx** stands for file format:

00 - binary file

01 - *.mbn file containing just partition table

02 - *.mbn file containing both, partition table as well as partitions data

03 - *.csv file containing partition table

04 - reserved for future use

05 - *.def file containing partition table

and **yy** stands for error type:

10 - disk i/o error (disk i/o error, file access error, ...)

11 - maximum buffer limit exceeded (file size grater than max. supported buffer size

12 - unable to re-allocate buffer (file size is grater than buffer size and there is some problem when reallocating the buffer (e. g. not enough disk space))

13 - unknown separator (for *.csv files, nor comma (,) nor semicolon (;) were detected as separator)

14 - file does not specify any partition (none partition definition read)

15 - too many partitions specified in file (max. 16 partitions are supported for Qualcomm Multiply Partition, max. 64 partitions generally)

16 - incorrect numeric values format (typo in text-oriented files (*.csv), e. g. @34 instead of 234)

17 - version not supported (not supported version of algorithm specification detected)

18 - invalid file header (damaged header, some mandatory item missing, ...)

USAGE TIPS

“Must-be-good” blocks screening

Sometimes, nand flash memories are programmed after the device is completely assembled, e. g. during the first boot process. In such case, it may be necessary to guarantee that start blocks for all (or some of) the partitions will be good, or some other device quality related parameters.

Problem example:

There are three partitions in device (say with 2048 blocks total), starting from blocks 0, 100 and 1000. First block in each partition must be good. Moreover, maximum 2 invalid blocks are allowed in first partition. There are two blocks reserved for BBT at device end, they both must be good. And finally, the total count of invalid blocks in device cannot exceed 20.

Solution:

It is necessary to employ following features:

- **First block in partition must be good**, available through partition definition file in CSV format
- **Maximum allowed number of invalid blocks in partition**, available through partition definition file in CSV format
- **Max. allowed number of invalid blocks in device**, available through Access Method dialog <Alt+S>.

Firstly, suitable partition definition file must be prepared. For screening purpose, it is necessary to consider reserved BBT blocks being other, fourth, partition. Used partition size will be equal to 0 blocks in all partitions, since we don't wont to program any of allocated blocks.

Using available partitioning mode features, the file will look like this:

```
0; 99; 0; 0xFF7FF002; partition 1 – 1-st block must be good, max. 2 invalid blocks are allowed
100; 999; 0; 0xFFF7FFFF; partition 2 – 1-st block must be good
1000; 2045; 0; 0xFFF7FFFF; partition 3 – 1-st block must be good
2046; 2047; 0; 0xFFFFF000; reserved for BBT, all blocks are expected good (max. 0 invalid blocks are allowed)
```

This will screen-out first blocks in all three expected partitions, check invalid blocks count in first partition with error generation if there are more than 2 invalid blocks there, and check reserved blocks for all being good.

The last requirement must be ensured using <Alt+S> dialog. It is necessary to set option **Max. allowed number of invalid blocks in device** to 20, and to enable **Check max. allowed number of invalid blocks in device** check-box.

Now just remains to disable **Erase before programming** in <Alt+O> dialog and to run **Program** operation...

HISTORY

Version 1.00 - December 2009

– initial release

Version 1.01 - January 2010

– block validity indication byte position acceptance for Qualcomm Multiply Partition added

– Error codes section added

Version 1.02 - February 2010

– Add blank spare area feature incorporated

Version 1.03 - June 2010

– Special options added for CSV partition definition file (max. allowed number of invalid blocks in partition, invalid blocks management method, file system preparation)

Version 1.04 - August 2010

– Qualcomm Multiply Partition format specification refined

Version 1.05 - September 2010

– added passage on loading multiple input data files

Version 1.06 - February 2011

– some minor changes in values alignment for CSV format description

Version 1.07 - May 2011

– Group Define file format added

Version 1.08 - April 2012

– “First block in partition must be good” feature added for *.CSV partition definition file format

Version 1.09 - October 2012

– “Usage tips” section added

– “Must-be-good” blocks screening” chapter added

Version 1.10 - October 2017

– MBN file size updated

Version 1.11 – November 2018

– *.CSV specification updated

– availability information updated