DediWare
Software User Manual
For NAND Flash

Please read the instructions carefully before use.

Table of Contents

I. Introduction

DediWare provides a professional user-friendly interface for NAND Flash. It provides corresponding bad block management, ECC condition settings, and different partition tables for loading the project files. Furthermore, it can easily switch to engineering mode for verification and change to production mode for mass production.

Engineering mode provides basic ReadID, Read, Blank Check, and Program, verify as well as Batch…etc. You can save a project file (*.dprj) in a SD card through the USB, and then switch to production mode.

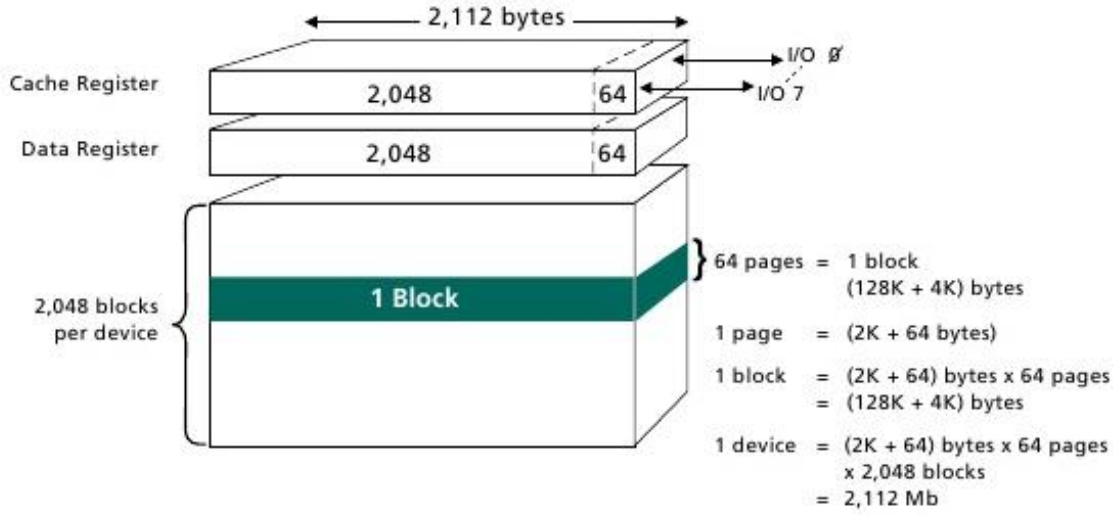
II. The Concepts of NAND in DediWare

2.1 Basic Concept of NAND Flash

With the advantages of low cost, large volume with fast saving speed, NAND Flash IC has been gradually replacing another kind of large volume IC, NOR Flash. However, NAND Flash contains Bit Error and bad block issues, so the software should provide more flexible options and supports. Therefore, DediWare provides bad block management and supports ECC conditions, which can be used accordingly. This manual will guide you to set up the settings and accomplish effective NAND flash programming.

2.2 The Structure of NAND Flash (Greater than 1Gb)

Use Micron MT29F2G08AxB (the below image) as an example; the entire IC is formed of 2048 blocks; each block contains 64 pages; each page is formed of 2048Bytes (Data)+64Bytes (Spare).



2.3 Spare

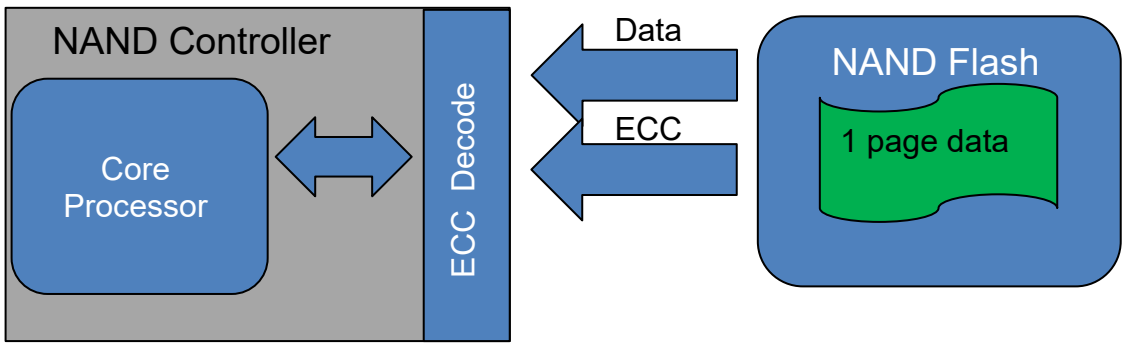
Spare is also called OOB (Out of Band). The structure of NAND shows every page has an extra Spare. A Spare is for bad block markings and extra applications, such as building BBT or ECC calculations…etc.

2.4 ECC (Error-correcting code) and Bit Error

During the process of reading NAND Flash, Bit Error might occur due to its structure. Once the NAND Controller receives the ECC Codes through different calculation methods, it will analyze the data accuracy and solve the Bit Error. In addition, DediWare provides a hardware inspection for Bit Error. Set up the values and execute Verify, then it will filter the reference number to check if it complies with the inspection requirement
Please refer to Section 4.1.2 **BBM Setting** for detail information.

Some NAND Flash support Internal ECC functions. For example: SPI NAND from GigaDevice. When internal ECC is enabled, the NAND flash will calculate the ECC and write it into the address where SpareArea has assigned. In addition, during the reading process, NAND flash will compare and correct the ECC to improve data accuracy.

Furthermore, DediWare also supports ECC Code generation computing. The ECC Code will be written into the assigned area after calculations. Provide the required calculation methods and management methods to check if it can accomplish in the programming software or not.
If you need this function, please contact DediProg.



2.5 Bad Block

In general, the bad block definition of NAND flash is when the first Byte of the Spare in the first Page of each block is non-0xFF, also known as Bad Block indication (BI). This definition also applies to the DediWare. However, the bad block marking characteristics might be different from each manufacturer; some markings will remain after Block Erase, some will be removed, and some will not be erased at first, but will be erased after several times of Block Erase. Therefore, the characteristics should be determined by the actual condition.

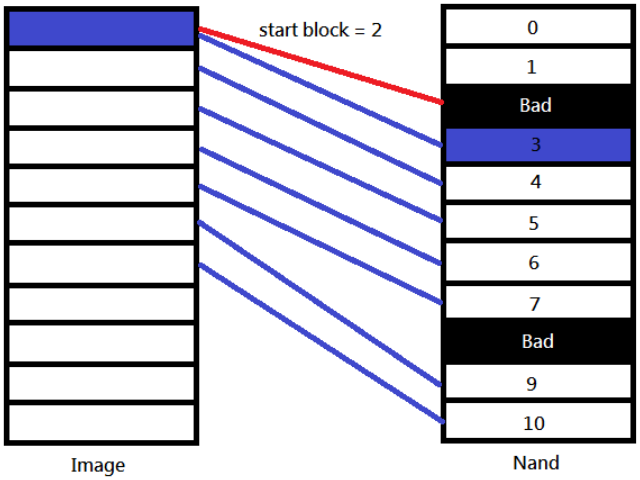
Every manufacture might define the Bad Block differently.
For example: a NAND from MXIC.
If the first Page of the Block or the first Byte of the Spare in the second Page is “non-0xFF”, then it will be considered as Bad Block. Our software will consider it as a Bad Block as well.

There are two kinds of bad blocks: The factory bad block and the application bad block.
Factory bad block: The IC manufacturer will filter the blocks in the progress of FT (Final Testing); the IC that does not comply with the requirements will be marked as bad block.

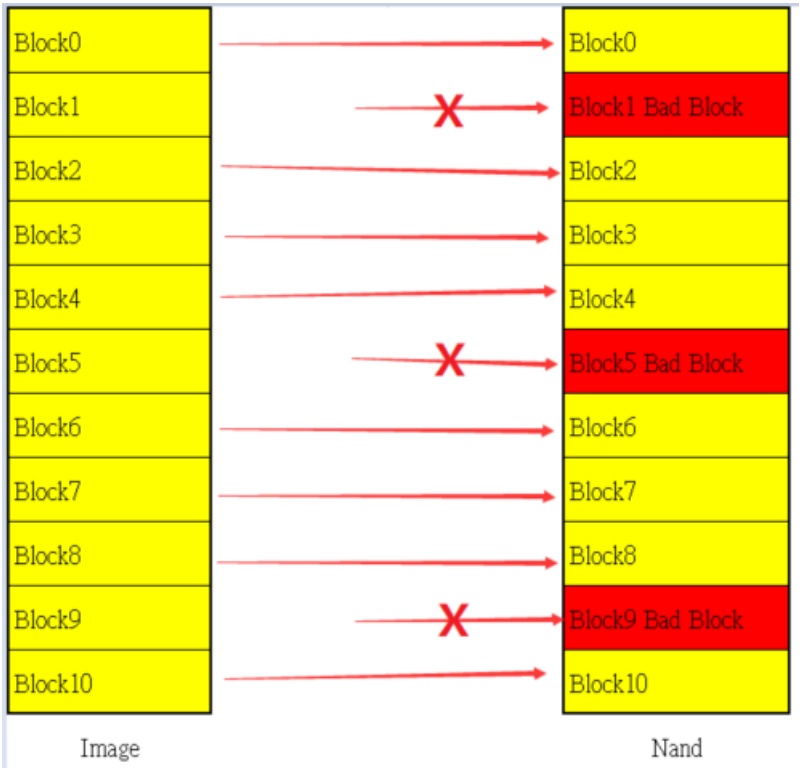
Application bad block: Resulted from too many times of erasing and writing. The board controller will mark and manage it.

2.6 Bad Block Management

- Hard Copy: Force write the data into every Block; it will cause high failure rate.
- Skip Bad Block: The most common BBM is Skip, which means when it detects a Block; it will shift the data to next Block.
  Use the below image as an example, if you tend to start writing the data from Block 2, but it is bad block, then the data will Skip it and start from Block 3, and so on.



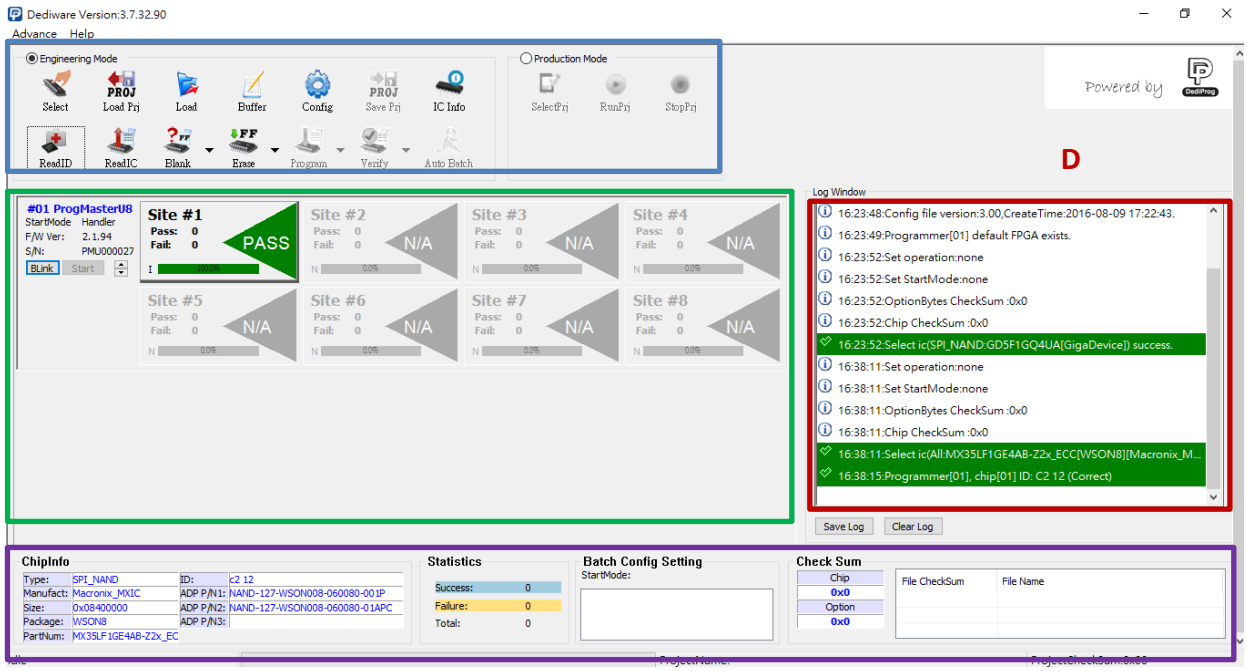
- No management: When a Bad Block is encountered, the data of the corresponding block in the programming file will be discarded and not written.



On the other hands, there are times when a specific block cannot have bad blocks or cannot exceed a certain number, our software can inspect the assigned area to see if it has exceeded the numbers or not, by using the Guarded Area function.

III. DediWare Introduction

DediWare Screen:



A. Main Menu and Functions

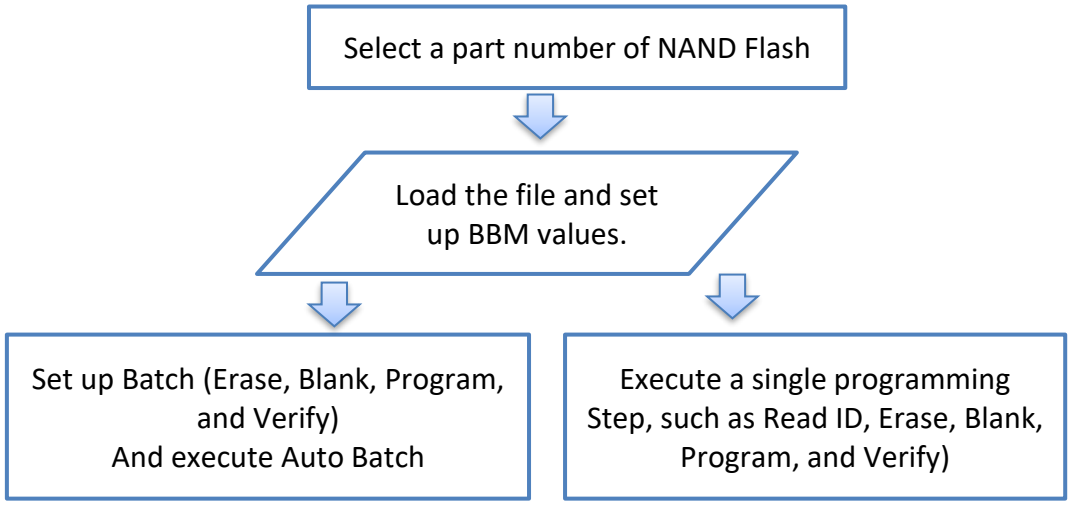
B. Programmer Information

C. IC information, Programming Setting, and Data Info.

D. Programming Log Window

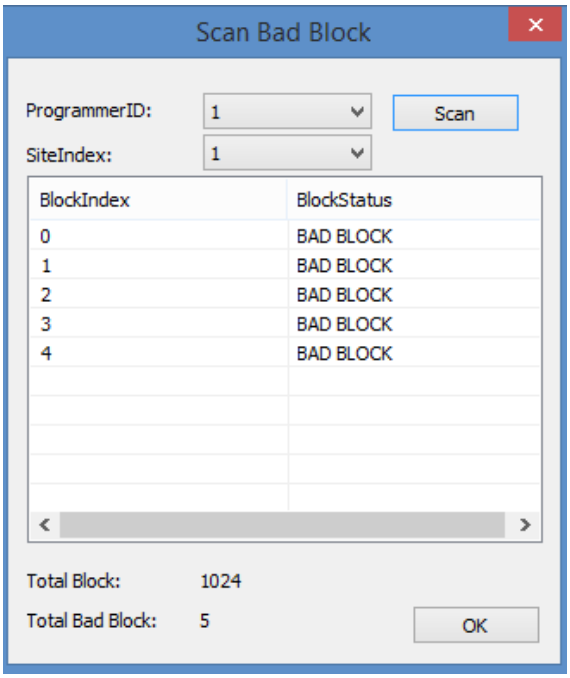
※ For more detailed DediWare instruction, please refer to **DediWare User Manual**.

3.1 The Basic Procedure of Programming the NAND Flash in DediWare



3.2 Special Function Only For NAND Flash: Addon

Locates at the Main Menu > Advance > Addon, which observes the Bad Block status of the NAND Flash. Bad block judging mechanism, please refer to the block management method of the datasheet.



Descriptions:

- **ProgrammerID**: Select a programmer.
- **SiteIndex:** Select a site.
- **Scan:** Scan the Bad Block.
- **OK:** Close the window.

Note:

※ Only allow to select one programming site for one programmer at a time.

※ This function can only inspect 255 blocks at most. It will only show the first 255 blocks if the numbers exceeded, then it will show a warning message "Too Many".

3.3 The Actual Procedure of The Software

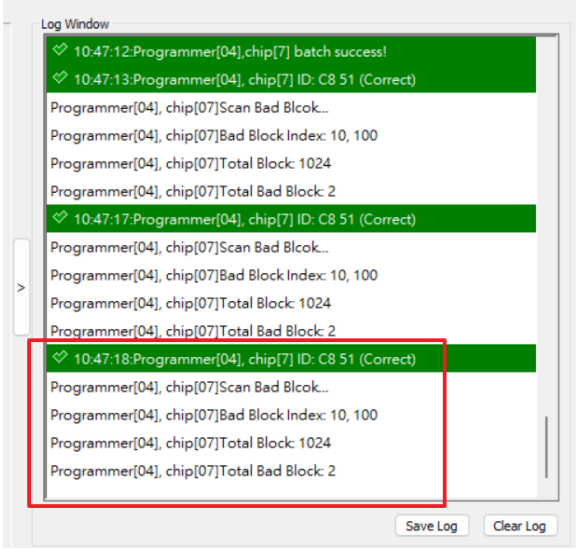
The software will establish a BBT (Bad Block Table) according to the Site that has IC. There are three kinds of situations:

Situation 1: Execute a single programming function:

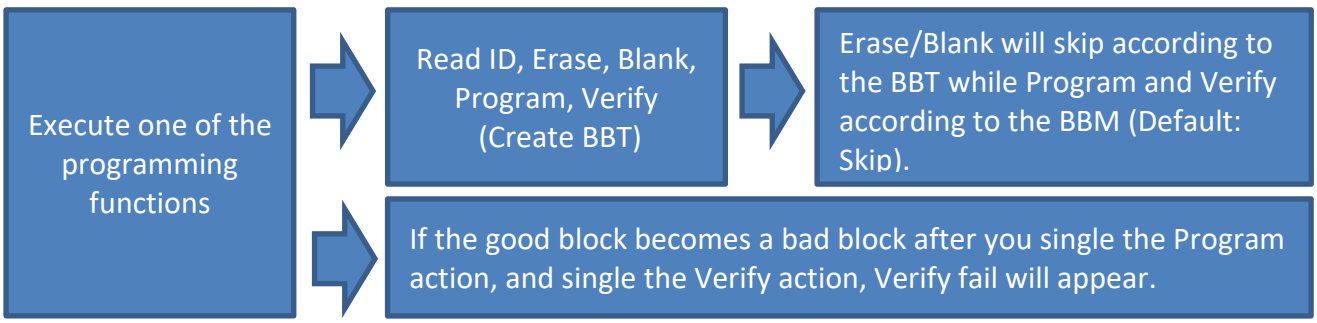
Select an IC part number and load the file (including BBM method and ECC setting), and then execute one of the programming functions.

The software will establish a BBT in the process of Read ID, Erase, Blank, Program, Verify, and it will be the reference values for BBM.

- **Read ID:** After verifying whether the ID is correct, the software will scan for bad blocks and create a BBT, and the result of scanning for bad blocks will be displayed in the log windows as shown in the figure below.



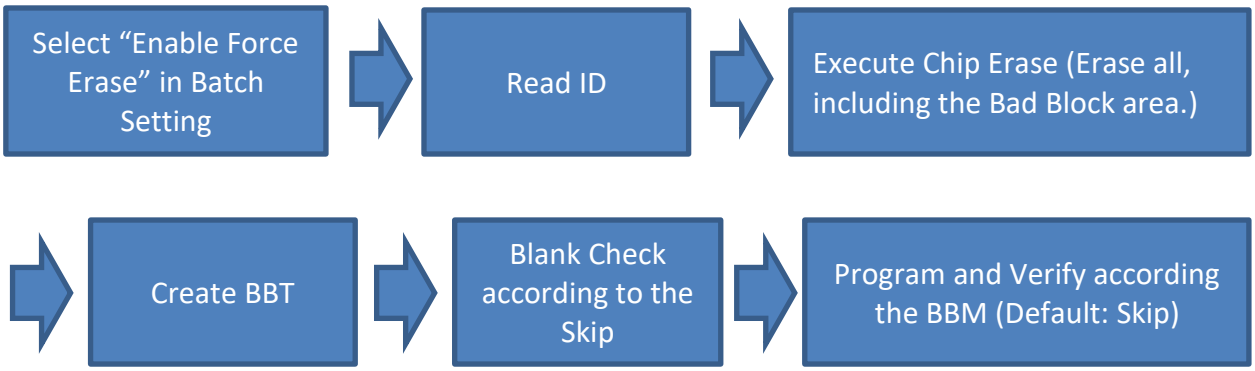
- **Erase:** The software will perform the action of establishing BBT, and then erases the data according to the BBT. If there is Enable Force Erase, the software will re-establish BBT after Erase.
- **Blank:** The software will perform the action of establishing BBT, and blank checks the data according to the BBT.
- **Program:** The software will perform the action of establishing BBT, and programming according to the BBM Setting.
- **Verify:** The software will perform the action of establishing BBT, and verifies according to the BBM Setting.



Situation 2: The Enable Force Erase in the Batch Setting:

Select an IC part number and load the file (Including BBM and ECC), and then check the Enable Force Erase option of the Erase Function in the Batch Setting.

Before proceed to the next procedure (Blank/Program/Verify), the software will create a BBT after Erase, which will erase all the data (Including bad blocks). This function is to re-work the NAND Flash that has been considered as Bad Block due to the data format distributions.

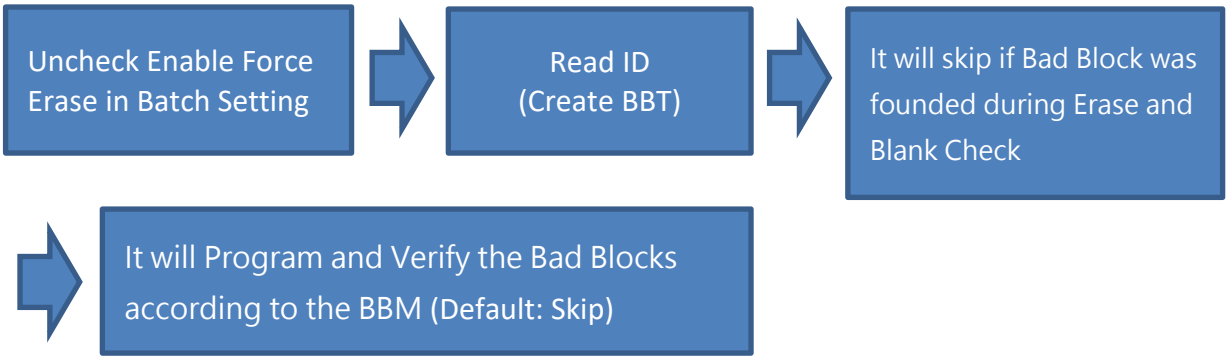


Situation 3: Disable Force Erase in Batch Setting

Select an IC part number and load the file (Including BBM and ECC).

If you did not select the Enable Force Erase option in the Batch Setting, then the software will establish a BBT in the beginning process of Read ID.

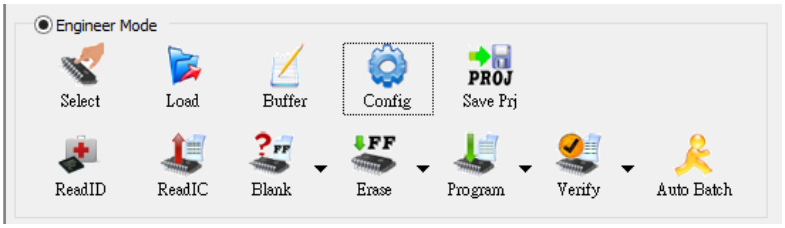
When erasing and blank checking, the software will skip the bad blocks according to the BBT while programming and verify are according to the BBM (Default: Skip).



IV. Engineering Mode and Production Mode

4.1 Engineering Mode

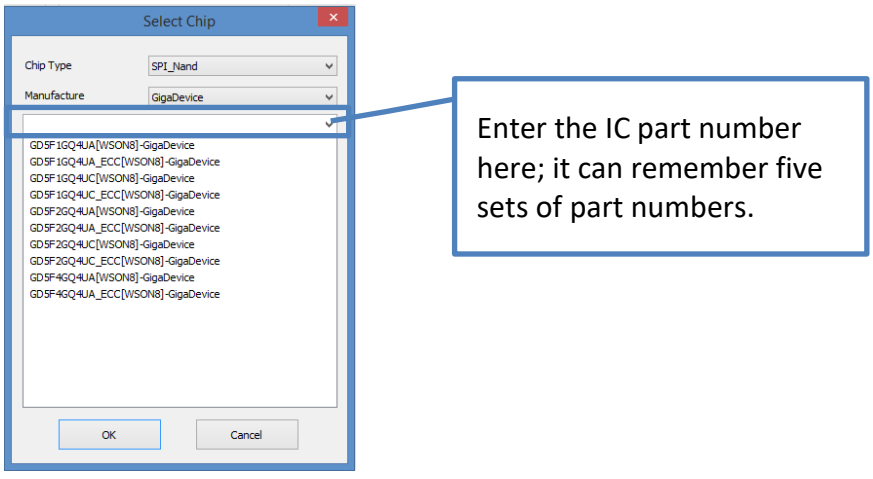
Open DediWare and follow the below steps to proceed engineering programming and verification. After verification, create a project file and switch to Production Mode for mass production.



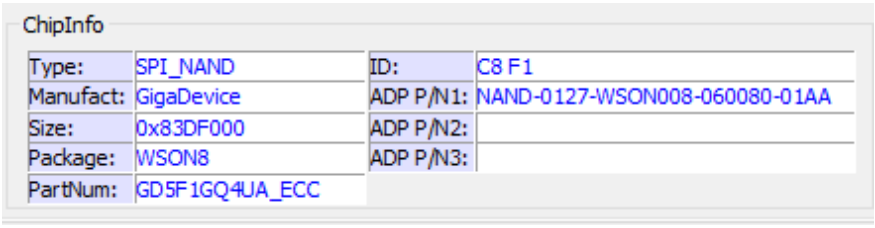
4.1.1 Select IC Brand / Part Number / Packaging (Select)

- The current NAND Types that DediWare support are Parallel NAND and SPI NAND.
- If the number is shown as XXXX_ECC, then it means the software will turn on the Internal ECC during the entire process, otherwise, it will be off.

Take the below image as an example.
There are two kinds of part numbers: GD5FGQ4UA and GD5FGQ4UA_ECC
GD5FGQ4UA means the Internal ECC will be off during the entire process.
On the other hand, GD5FGQ4UA_ECC means the Internal ECC will be on. Therefore, please ensure to evaluate whether you need Internal ECC or not before choosing the part number.



After selecting the part number, the Chip Info will show the IC information.



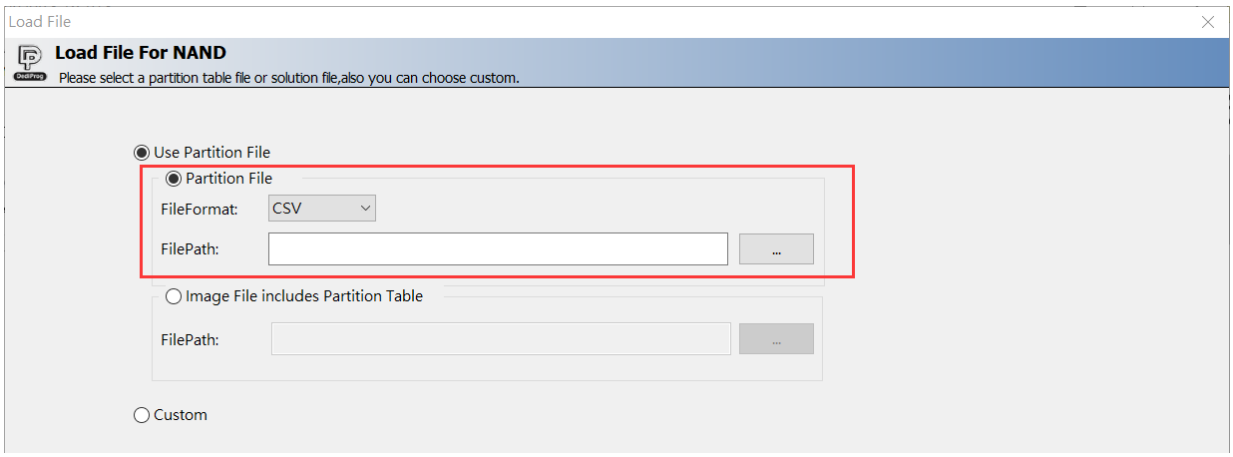
4.1.2 Load the Programming Code (Load) 

There are two ways of loading the programming code: Combine the partition files or set up manually.

4.1.2.1. Use Partition File:
If your Partition file is in CSV (Comma Separated Values Format), DEF (Group Define File Format), or MBN (Qualcomm Multiply Partition Format), then you can select this option for loading the Image File and Partition File.

For example, load a Partition table that is in MBN format (Please go to chapter VI for detailed definition).

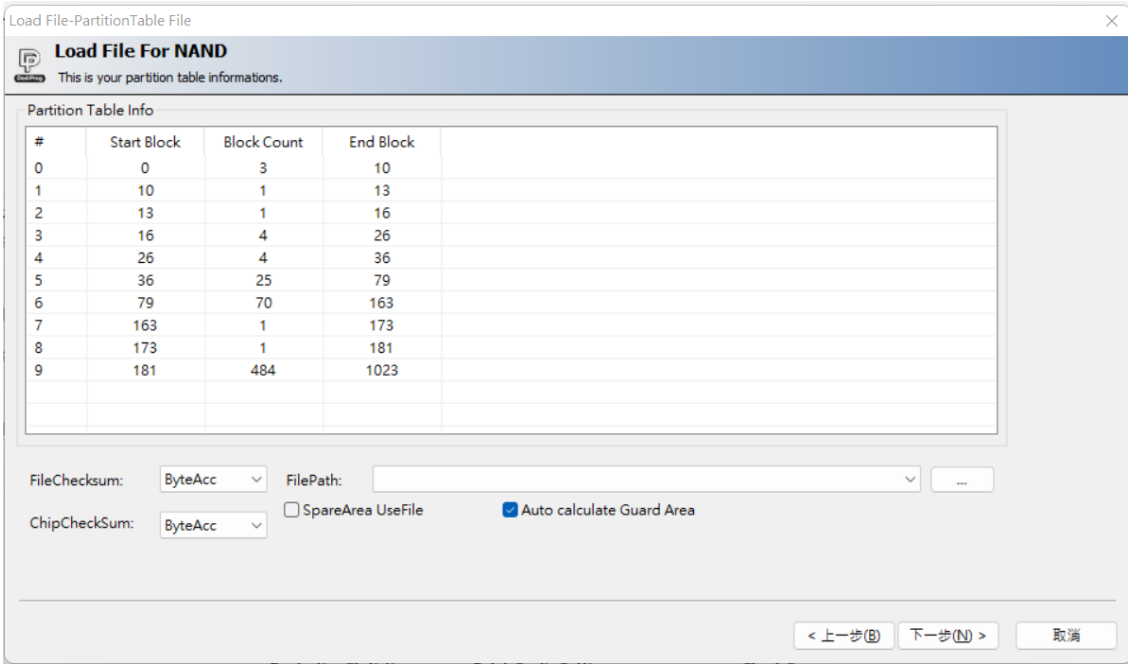
Step 1. Load Partition table file.



Step 2. After the file is loaded, it will show the definition of each Partition. For FilePath, choose the image file that you are going to program; as for “FileCheckSum” and “ChipCheckSum”, choose a method for Checksum calculation.

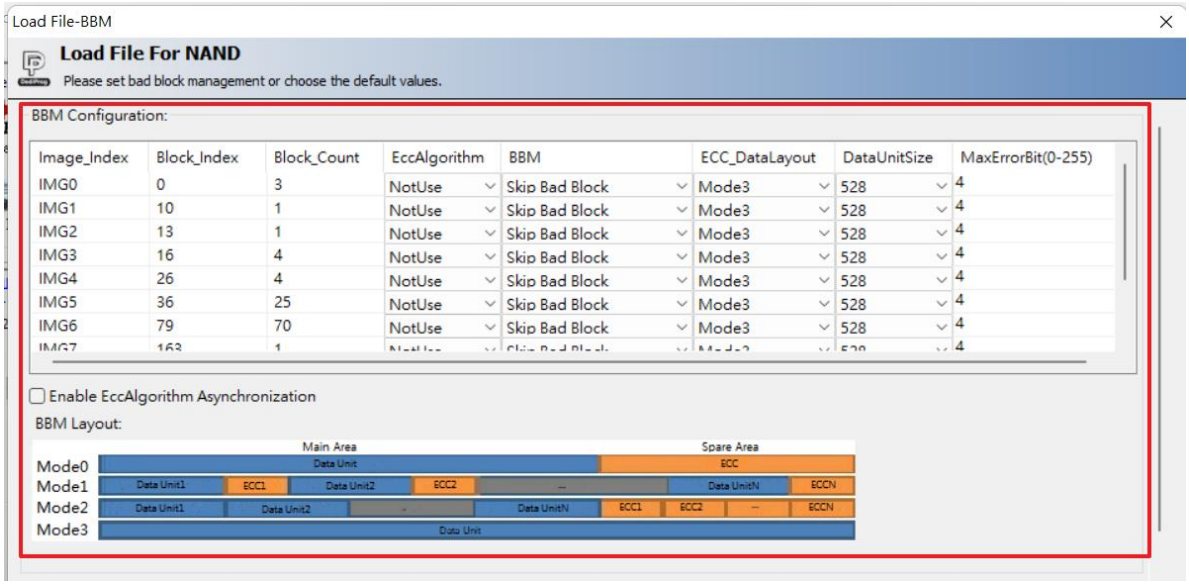
SpareArea UseFile: It decides whether the file will include SpareArea or not; check the box if SpareArea is required.

Auto calculate Guard Area (detailed function description will be supplemented in Chapter VI): This function automatically calculates how many bad blocks are allowed in each partition in the partition table, and automatically sets Guard Area on the "Load File-BBM" page.



Step 3. Set up BBM and Guarded Area Conditions

A

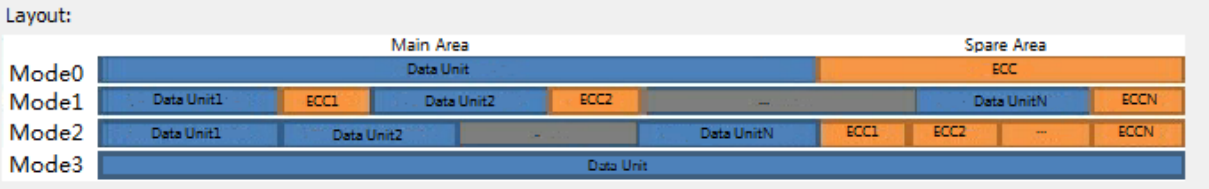


B

A. BBM Configuration:

Set up BBM according to the actual amounts of the loaded file. As the above image, there are multiple images files need EccAlgorithm, BBM, EccDataLayout, DataUnitSize, and MaxErrorBit.

- **EccAlgorithm:**
  - If "SpareArea UseFile" is checked on the "Load File-IMAGE" page, it will display:
    - Use File: indicating that the Image File has data containing SpareArea.
  - If “SpareArea UseFile” is not checked on the “Load File-IMAGE” page, it will display:
    - NotUse: Indicates that the Image File does not contain SpareArea data, and the ECC algorithm is not used.
    - BCH8 MDM: This ECC algorithm is only provided to Flex customers. Due to the confidentiality agreement, we cannot provide the detailed file of this algorithm.
    -
- **BBM:** A choice of bad block management. Currently only supports Hard Copy, Skip Bad Block and No management. If you need other kinds of BBM, please feel free to contact us.
- **BBM Layout:** The BBM Layout feature only functions when Verify is executed on DediWare. During Verify execution, DediWare not only verifies data accuracy but also conducts a Bit Error check to ensure that the IC can boot up normally after being mounted on the board.

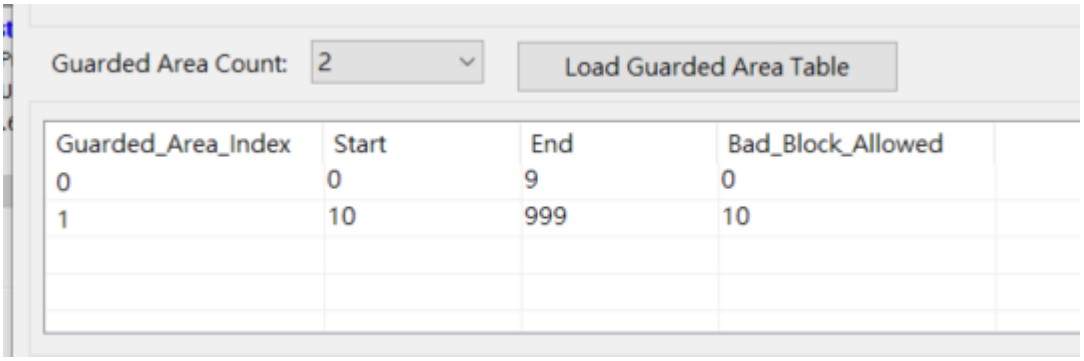


BBM Layout will check Error Bits based on the following settings.

- Ecc_DataLayout：Four data layout modes (Mode 0, 1, 2, 3) are provided. Please select the corresponding mode based on the arrangement of data in your programming file. The default mode is Mode 3.
  ※ Note: The ECC area (orange blocks) in Mode 0, 1, and 2 does not allow any Bit Errors. Therefore, if Bit Errors occur in the ECC area (orange blocks) of Mode 0, 1, or 2 during verify, the verify will fail.
- DataUnitSize：The size of the Data Unit under data layout modes (Mode 0, 1, 2, 3).
- MaxErrorBit：The maximum allowable number of Bit Errors for each Data Unit. Please set the maximum allowable number of Bit Errors for each DataUnit based on the ECC correction capability.

B. Guarded Area Count:

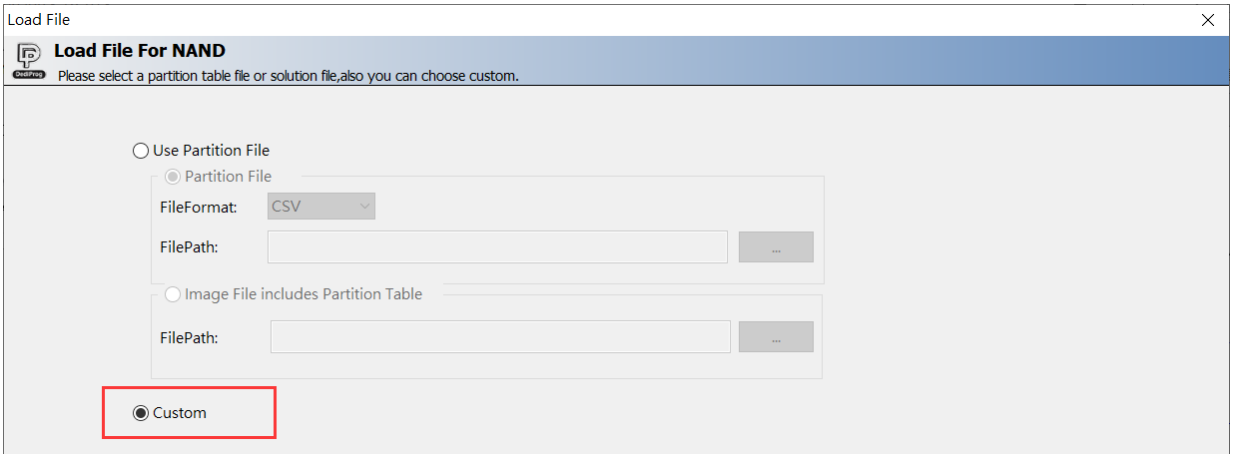
This function can set up a range for block and detect the maximum acceptable value for bad blocks. In this chart, there two sets of ranges; the first condition does not allow any bad block from Block0 to Block9 while the second condition only allows ten bad blocks from Block 10 to Block 999. Then, during the process of creating BBT, the software will determine accordingly. If the above conditions are true, then the IC will be listed as NG parts.



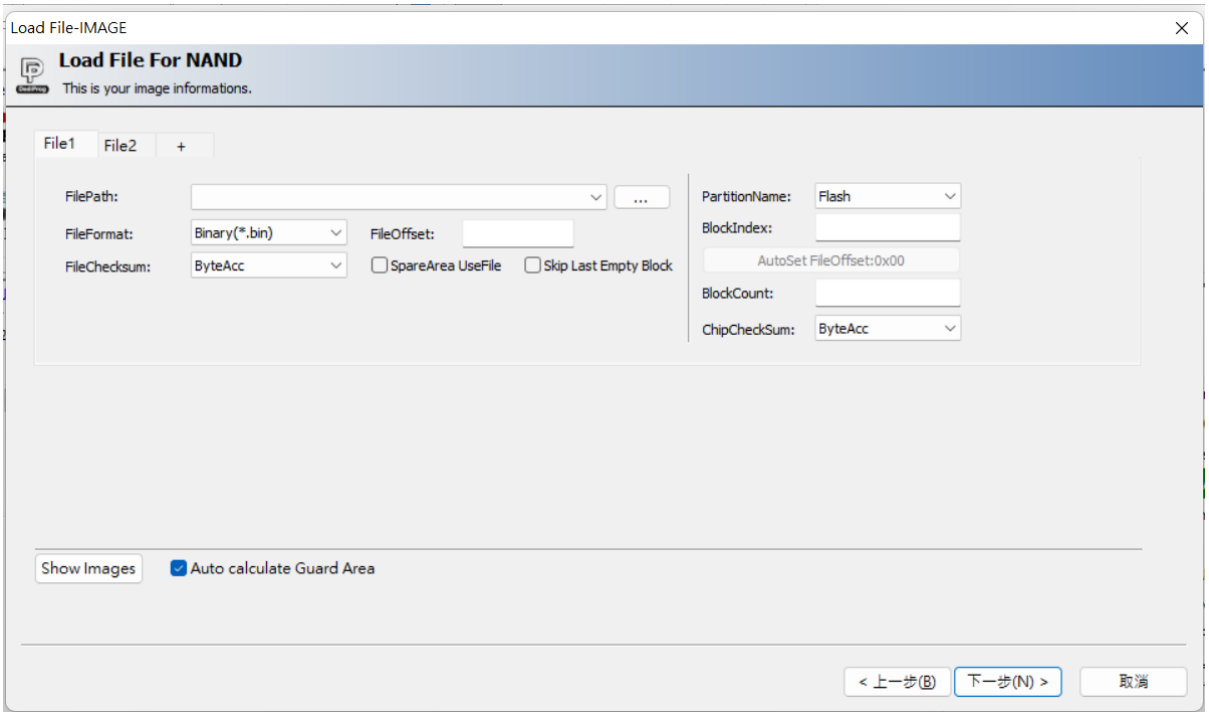
- **Load Guarded Area Table**: Directly load Guarded Area Table in MBN (Qualcomm Multiply Partition Format) format.

4.1.2.2. Custom: Set up Manually

a) Select Custom



b) Function Descriptions (From Left to Right):



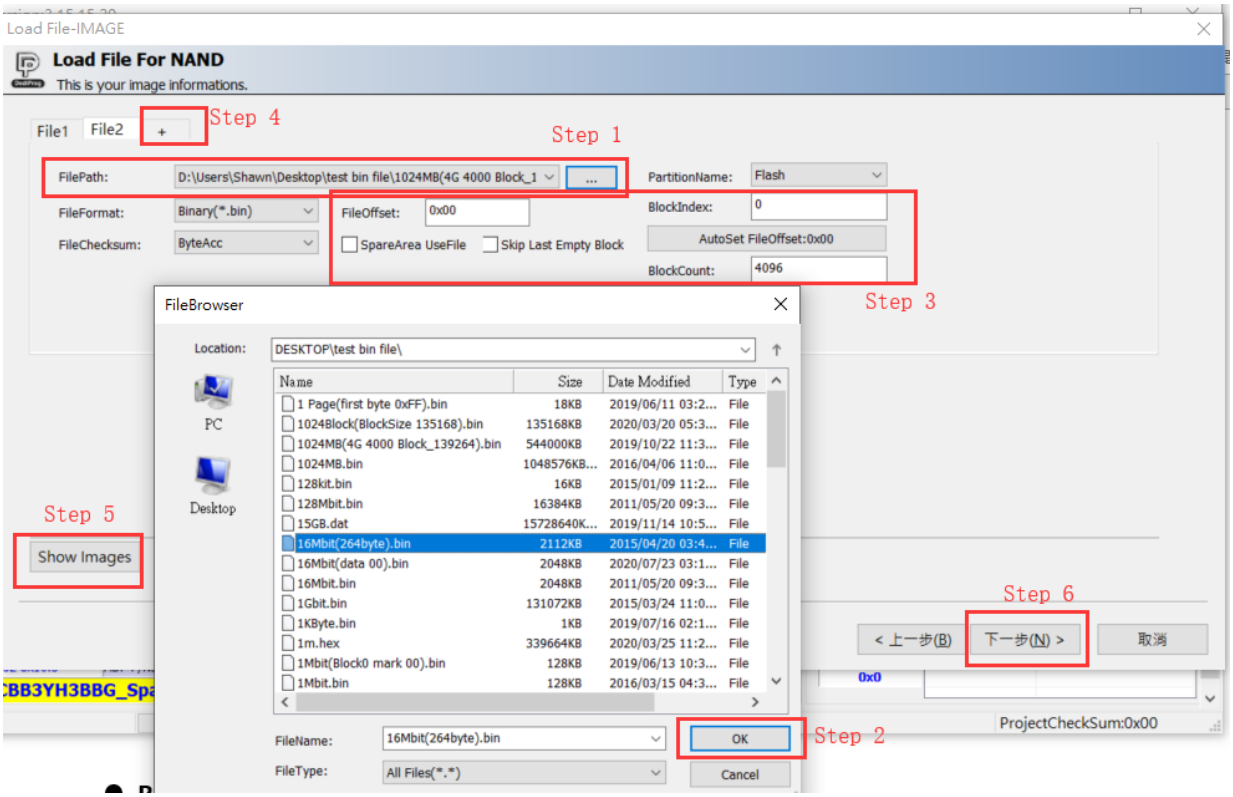
- **FilePath:** The path of the programming file (it can remember the File Path)
- **File Format:** The file format of the image file (Binary Only).
- **File Checksum:** The calculation method for the file Checksum
- **File Offset:** Assign a file offset address for loading the Buffer.
- **SpareArea UseFile:** It decides whether the file will include SpareArea or not; check the box if SpareArea is required.
- **Skip Last Empty Block:** If the file has continuous block data whose values are all FFh, check this option to skip these blocks from being loaded.
- **For Example:** The chip has 1024 blocks (0~1023) in total, and the loaded programming file Block Count is 1024.

1. The values of Block1000~1023 of the programming file are all FFh. Check "Skip Last Empty Block" box will not load the blocks of Block1000~1023.
2. The values of Block1000~1022 of the programming file are all FFh, but Block1023 has valid data non-FFh value, because the last Block1023 has non-FFh value, so the value of Block1000~1023 is not continuous FFh, so check "Skip Last Empty Block" box will not Skip Block 1000~1022.

- **Partition Name:** Currently, NAND only has Flash option.
- **Block Index:** The beginning of the Block
- **AutoSet FileOffset:** When the address of the Image file and the NAND Flash are parallel, you can set up the offset value for Block index. Enter the value, click the AutoSet FileOffset, and then the File offset value will change accordingly.
- **Block Count:** The total Block number for programming.

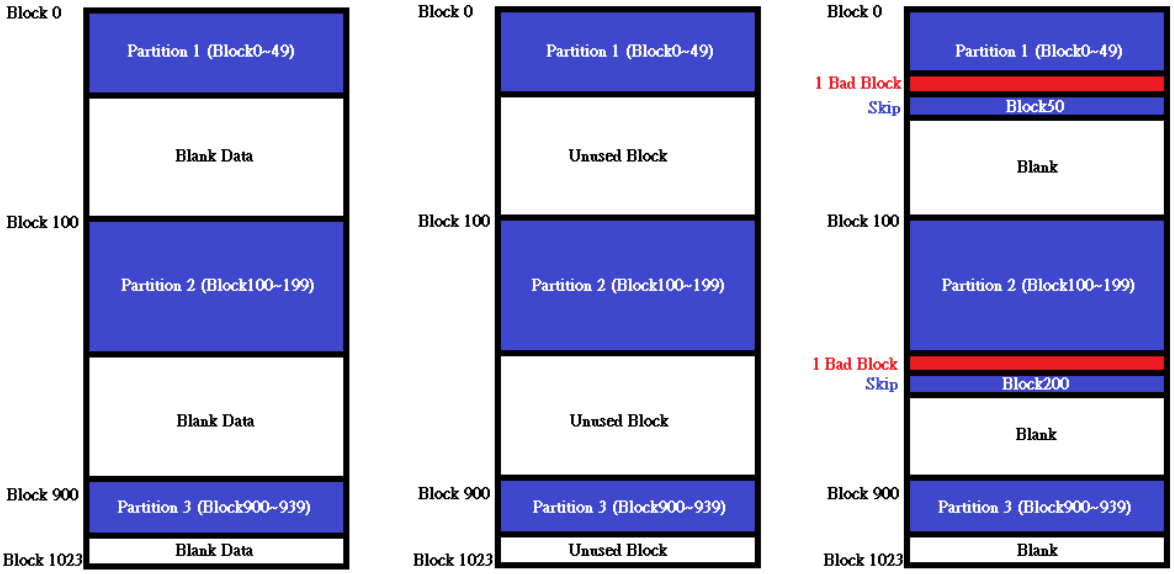
- **+ :** Add a programming file.
- **Auto calculate Guard Area** (detailed function description will be supplemented in Chapter VI): This function automatically calculates how many bad blocks are allowed in each partition in the partition table, and automatically sets Guard Area on the "Load File-BBM" page.

c) Loading Steps:



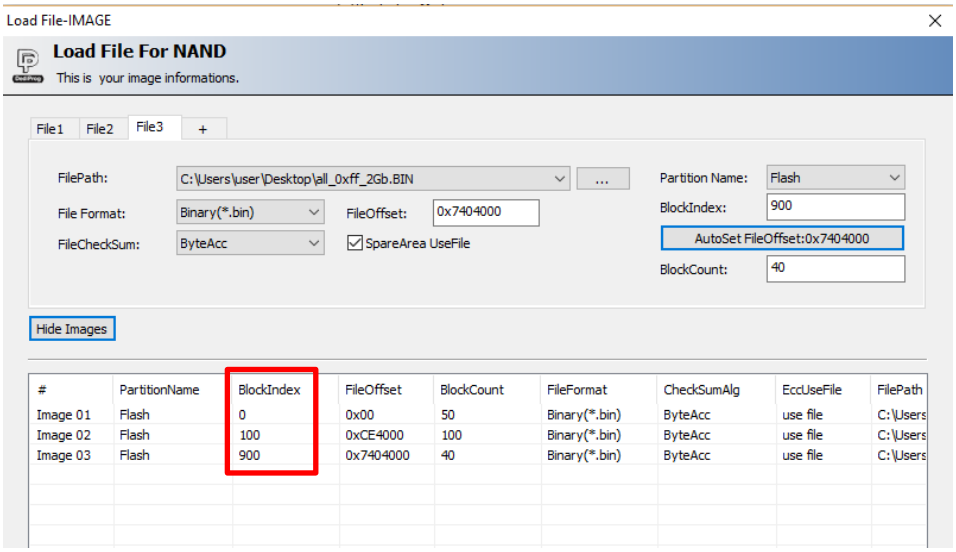
Step1: Click the (….) button and it will bring up the FileBrowser.
Step2: Find the programming file that you need, click OK, and then set up the settings.
Step3: Confirm the values and the settings.
Step4: Click + to add a file. Please repeat step 1 to 4 for multiple files.
Step5: The file information will show in the window.
Step6: Finish setting and click next.

When you assign BlockIndex and need the File offset to move with it; there is a function called AutoSet FileOffset in DediWare for combining multiple Partition Images into one Image File, so you will only need to load one file Image and set up the BlockIndex separately (Shown as the below image).

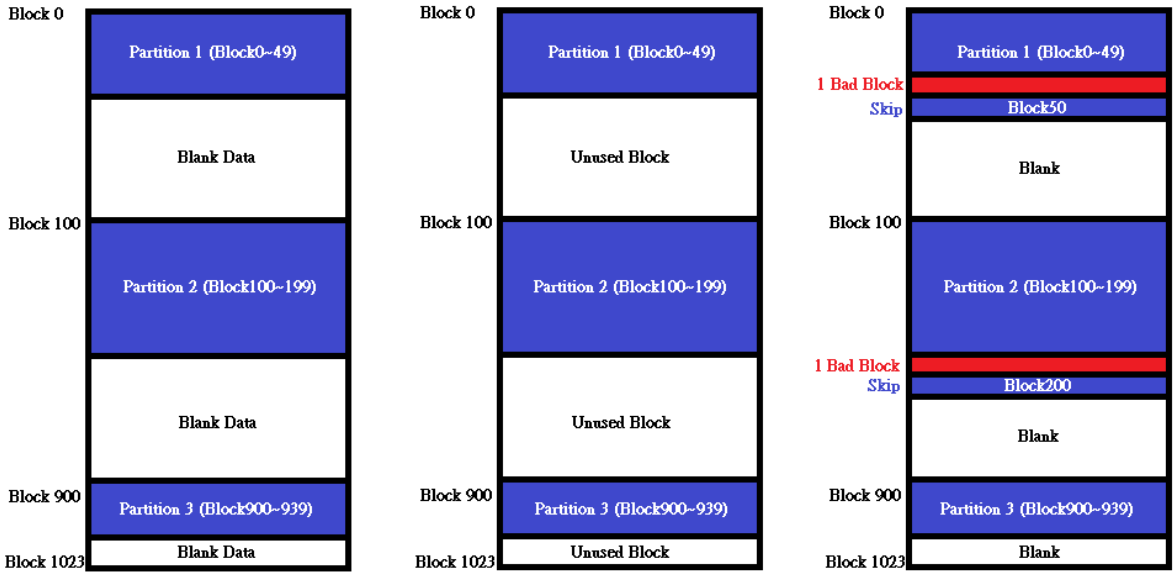


The Image File on the left (Image File x1) is a combination of three Partitions; the software will set up the block division when the file is loaded, which will load the same Image File and able to set up three different settings. For Example:
Set up Partition 1, BlockIndex=0, BlockCount=50, FileOffset=0.
Set up Partition 2, BlockIndex=100, BlockCount=100, FileOffset=0xCE4000.
Set up Partition 3, BlockIndex=900, BlockCount=40, FileOffset=0x7404000.

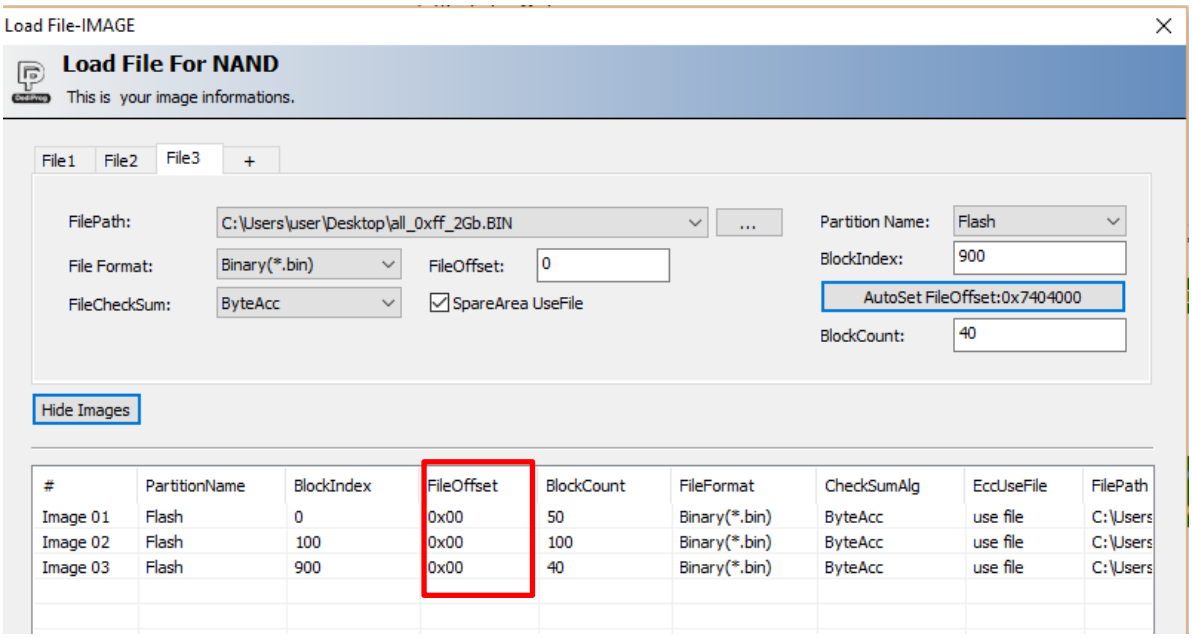
FileOffset will calculate the corresponding address (Byte wide) automatically while setting up the BlockIndex. As for Partition2, when BlockIndex is 100 while 1 Block of the IC = 64pages and 1Page =2112Bytes, then 1 Block is 64*2112=135168Bytes (0x21000). For Block 100, the starting address will be 0x21000 * 0x64=0xCE4000; for Partition 3, it will be 0x21000 * 0x384=0x7404000 (Shown as the below image).



The middle image (Image Filex3) shows Partition 1~3 will be divided into three images while loading, but remember to set the FileOffset to zero for BlockIndex in order to avoid data offset.



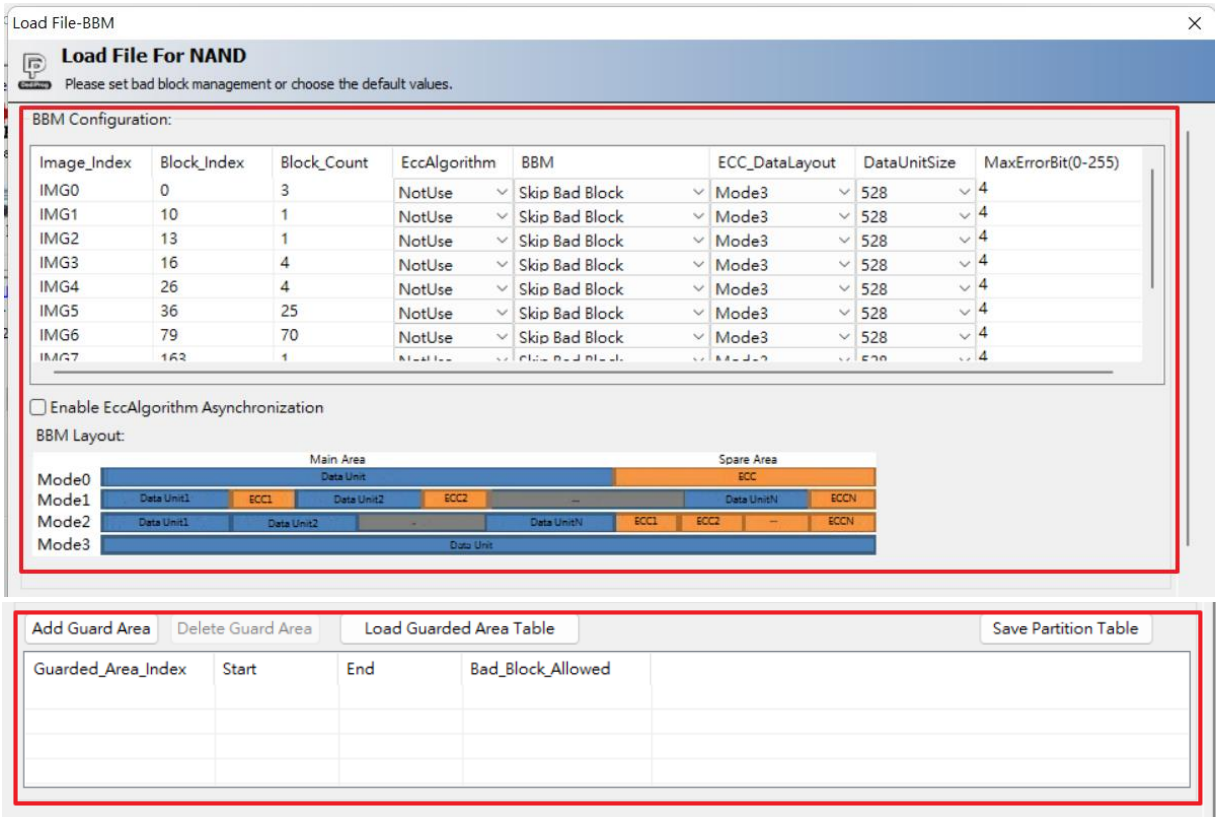
Set up Partition 1, BlockIndex=0, FileOffset=0, BlockCount will be 50 automatically.
Set up Partition 2, BlockIndex=100, FileOffset=0, BlockCount will be 100 automatically.
Set up Partition 3, BlockIndex=900, FileOffset=0, BlockCount will be 40 automatically.



The image on the right is the listed order after the image has been programmed.

If Partition 1 and Partition 2 have detected bad block separately, then this will be the list order after the Skip process.

Step7: Open BBM Setting

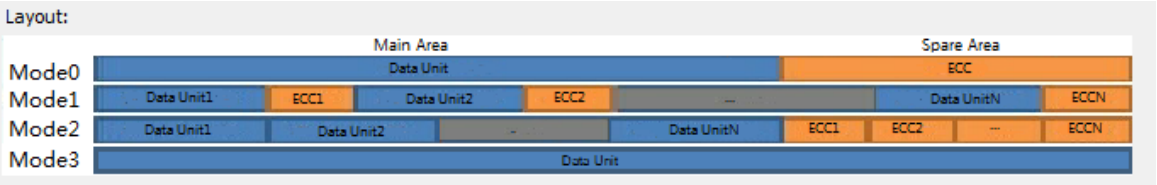


A. BBM Configuration:

Set up the BBM according to the actual amounts of the loaded file. As the above image, there are three image files need EccAlgorithm, BBM, EccDataLayout, DataUnitSize, and MaxErrorBit.

- **EccAlgorithm:**
  - If "SpareArea UseFile" is checked on the "Load File-IMAGE" page, it will display:
    - Use File: indicating that the Image File has data containing SpareArea.
  - If “SpareArea UseFile” is not checked on the “Load File-IMAGE” page, it will display:
    - NotUse: Indicates that the Image File does not contain SpareArea data, and the ECC algorithm is not used.
    - BCH8 MDM: This ECC algorithm is only provided to Flex customers. Due to the confidentiality agreement, we cannot provide the detailed file of this algorithm.
- **BBM:** A choice of bad block management. Currently only supports Hard Copy, Skip Bad Block and No management. If you need other kinds of BBM, please feel free to contact us.

- **BBM Layout:** The BBM Layout feature only functions when Verify is executed on DediWare. During Verify execution, DediWare not only verifies data accuracy but also conducts a Bit Error check to ensure that the IC can boot up normally after being mounted on the board.

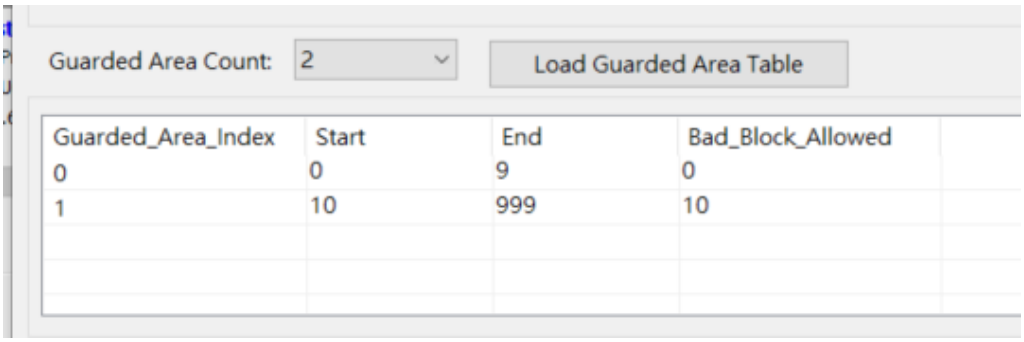


BBM Layout will check Error Bits based on the following settings.

- Ecc_DataLayout：Four data layout modes (Mode 0, 1, 2, 3) are provided. Please select the corresponding mode based on the arrangement of data in your programming file. The default mode is Mode 3.
  ※ Note: The ECC area (orange blocks) in Mode 0, 1, and 2 does not allow any Bit Errors. Therefore, if Bit Errors occur in the ECC area (orange blocks) of Mode 0, 1, or 2 during verify, the verify will fail.
- DataUnitSize：The size of the Data Unit under data layout modes (Mode 0, 1, 2, 3).
- MaxErrorBit：The maximum allowable number of Bit Errors for each Data Unit. Please set the maximum allowable number of Bit Errors for each DataUnit based on the ECC correction capability.

B. Guarded Area Count:

This function can set up a range for block and detect the maximum acceptable value for bad blocks. In this chart, there two sets of ranges; the first condition does not allow any bad block from Block0 to Block9 while the second condition only allows ten bad blocks from Block 10 to Block 999. Then, during the process of creating BBT, the software will determine accordingly. If the above conditions are true, then the IC will be listed as NG parts.

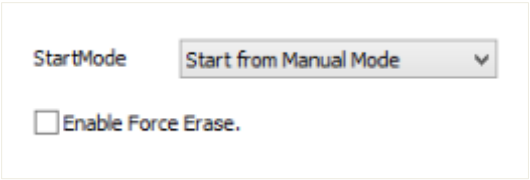


- **Load Guarded Area Table**: Directly load Guarded Area Table in MBN (Qualcomm Multiply Partition Format) format.

Note:
Since NAND Flash programming requires configuration of BBM and ECC, so if you cannot find any suitable BBM and ECC, please contact DediProg for further evaluation.

4.1.3 Config Setting 

Enable Force Erase for NAND Flash



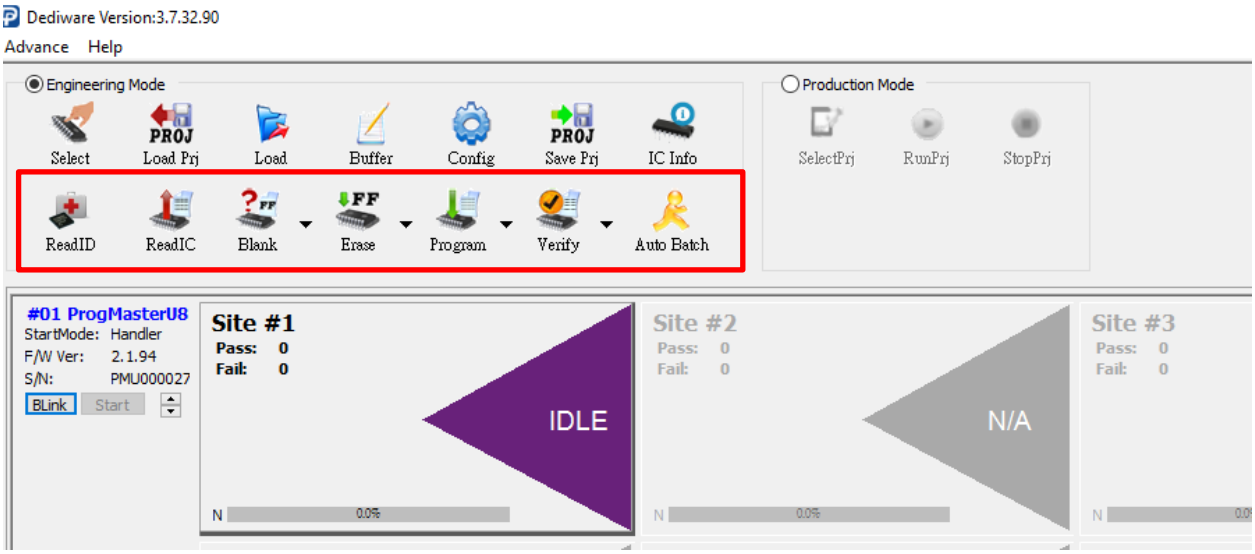
DediWare provides Enable Force Erase function for solving the Bad Block issues. If the Batch Setting in the Config window is Erase Blank +Program + Verify, then the actual action of the software will be like the followings:

A. **When Enable Force Erase in not selected:** It will read ID while creating BBT (Bad Block Table) → When execute the Erase and the Blank, it will skip the data according to the BBT. → It will Program and Verify the data according to the BBM setting (Default is Skip as well).
B. **When Enable Force Erase is selected:** Read ID →Erase all (including Bad Block) → the software will create a BBT (Bad block Table) → When execute the Blank, it will skip the data according to the BBT→ It will Program and Verify the data according to the BBM setting (Default: Skip).

4.1.4 Execute Programming Functions

In the Engineering mode, no matter how many programmers are there, it can only program one programming site at a time.

Therefore, please ensure to select the correct programming site before executing. Now, Site 1 is selected for programming in the below image.
The basic programming functions are circled in red.



4.1.4.1 Functions for Programming Single Site

- **Read ID:** Create a BBT. Read the NAND's ID and scan bad block shows the information in the Log window.
- **Read IC:** Create a BBT. Read and display the data of the NAND, and also can compare the files.
- **Erase:** Create a BBT. Erase and skip the bad block of the NAND Flash according to the BBT.
- **Blank:** Create a BBT. After skipping bad block according to the BBT, it will check if the IC is empty or not
- **Program:** Create a BBT. program the imported file to the NAND Flash according to the BBM Setting.
- **Verify:** Create a BBT. Verify the NAND Flash according to the BBM setting.
- **Auto Batch:** Read id (create BBT) → Execute the programming functions that you set up for the Batch Setting that is in the Config window.

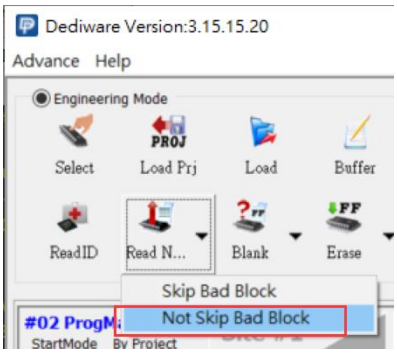
4.1.4.2 When to Execute These Functions

Actions / Functions	IC not selected	IC is selected	IC is selected and the file has imported	IC is selected, the file has imported and has set up the Config/Batch
Read ID	×	✓	✓	✓
Read IC	×	✓	✓	✓
Erase	×	✓	✓	✓
Blank	×	✓	✓	✓
Program	×		✓	✓
Verify	×		✓	✓
Auto Batch	×			✓

※ **The functions are not allowed to perform separately during the process of executing the project programming.**

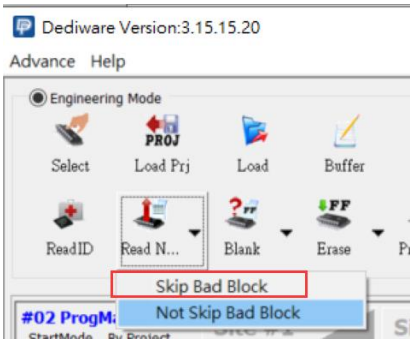
4.1.4.3 Read IC

- **Read IC with Not Skip Bad Block**: It will read the entire NAND Flash, including the Spare and the Bad Block.

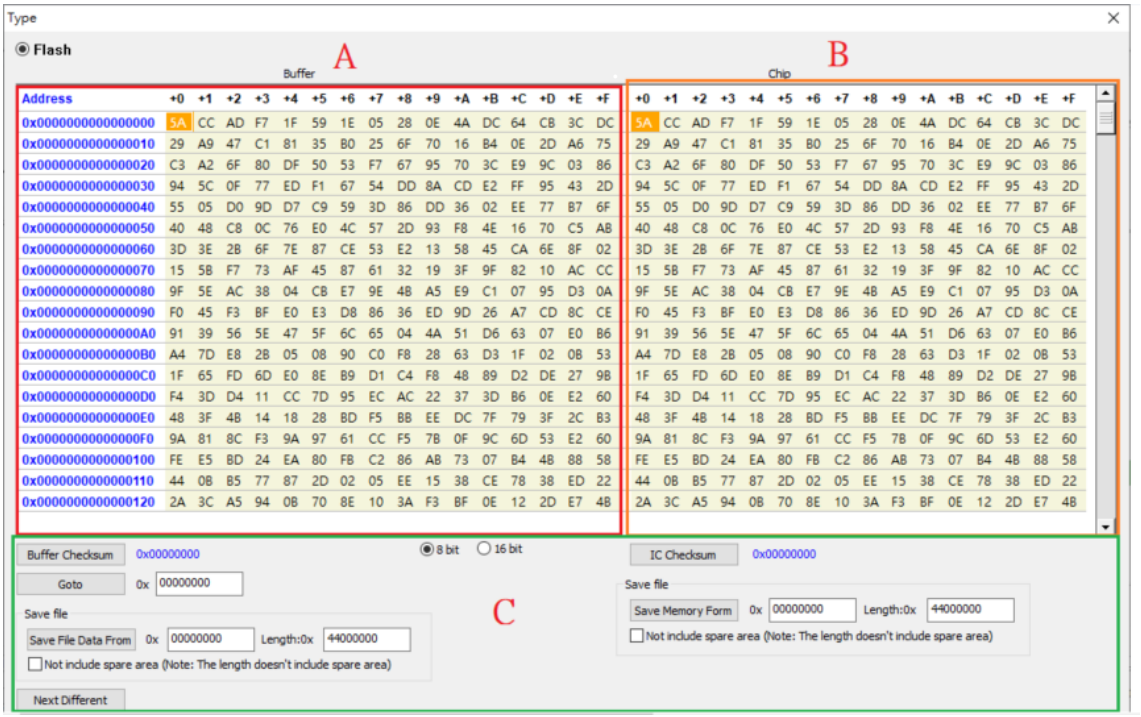


- **Read IC with Skip Bad Block:** It will read the entire NAND Flash, including the Spare and the Good Block. When encountering a bad block, it will skip and not read

※ **For problems that may be encountered when using Read IC with skip bad block to read Chip data, please refer to "VI. Troubleshooting " Q16.**



The window will be shown as below.



A. Buffer Area

The data will show in this area, once the file is imported.

B. Chip Area

These are the data from the IC. After reading the memory, it will compare with the data in the buffer area. The abnormal part will be high-lighted in red for better analyzing.

C. Other Functions

➢ **Buffer Checksum 及 Chip Checksum**

Provide the Buffer Checksum and Chip Checksum functions of the specified Partition to facilitate verification.

➢ **Goto**

It can quickly shift to the assigned address.

➢ **Next Different**

It will search the next address that has different data.

➢ **Save Memory**

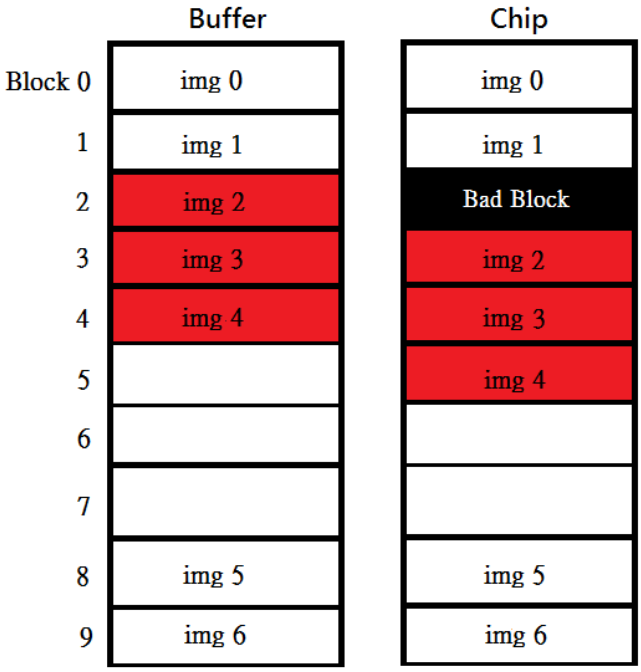
It saves the data of the entire NAND. However, since it has Bit Error issue, so the data might be a little different after each read.

➢ **Not include spare area (Note: The length doesn't include spare area)**

This function decides whether the content of the saved file should include the data of the Spare area.

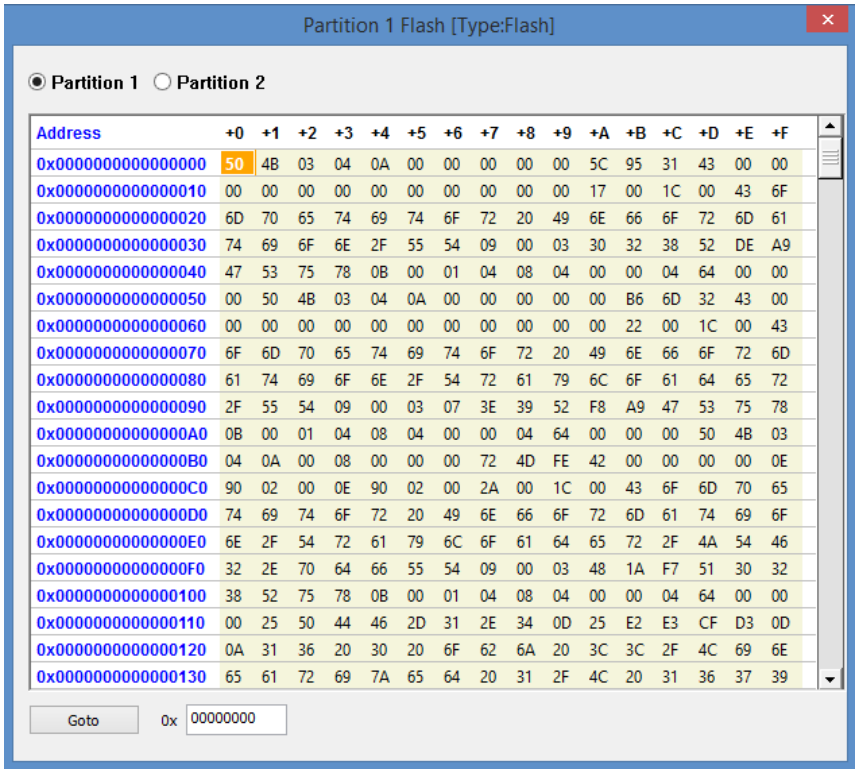
Again, after the process of Read IC with Not Skip Bad Block, the Compare function will compare the entire NAND Flash and the imported Image (without the skip process), so if Bad Block happens in the assigned image area, then the entire data might shift to the next block.

As the below image, each block is corresponded to an image. However, if Block 2 of the IC is a bad block, then the entire data will move to the next one. Therefore, when comparing the data from Block 2, the offset data will be in red. In addition, the second source starts from Block 8, and it will be a new beginning. Since NAND has Bit Error issues and the programmer will read the data from the original source without any ECC verification, so it is normal to have a little offset.



4.1.4.4 Buffer 

After import the file, the data will show in the Buffer area. Please check if the data has been placed at the correct address or not.



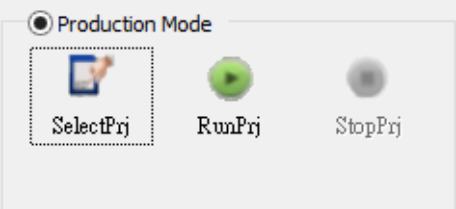
Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0x0000000000000000	50	4B	03	04	0A	00	00	00	00	00	5C	95	31	43	00	00
0x0000000000000010	00	00	00	00	00	00	00	00	00	00	17	00	1C	00	43	6F
0x0000000000000020	6D	70	65	74	69	74	6F	72	20	49	6E	66	6F	72	6D	61
0x0000000000000030	74	69	6F	6E	2F	55	54	09	00	03	30	32	38	52	DE	A9
0x0000000000000040	47	53	75	78	0B	00	01	04	08	04	00	00	04	64	00	00
0x0000000000000050	00	50	4B	03	04	0A	00	00	00	00	00	B6	6D	32	43	00
0x0000000000000060	00	00	00	00	00	00	00	00	00	00	00	22	00	1C	00	43
0x0000000000000070	6F	6D	70	65	74	69	74	6F	72	20	49	6E	66	6F	72	6D
0x0000000000000080	61	74	69	6F	6E	2F	54	72	61	79	6C	6F	61	64	65	72
0x0000000000000090	2F	55	54	09	00	03	07	3E	39	52	F8	A9	47	53	75	78
0x00000000000000A0	0B	00	01	04	08	04	00	00	04	64	00	00	00	50	4B	03
0x00000000000000B0	04	0A	00	08	00	00	00	72	4D	FE	42	00	00	00	00	0E
0x00000000000000C0	90	02	00	0E	90	02	00	2A	00	1C	00	43	6F	6D	70	65
0x00000000000000D0	74	69	74	6F	72	20	49	6E	66	6F	72	6D	61	74	69	6F
0x00000000000000E0	6E	2F	54	72	61	79	6C	6F	61	64	65	72	2F	4A	54	46
0x00000000000000F0	32	2E	70	64	66	55	54	09	00	03	48	1A	F7	51	30	32
0x0000000000000100	38	52	75	78	0B	00	01	04	08	04	00	00	04	64	00	00
0x0000000000000110	00	25	50	44	46	2D	31	2E	34	0D	25	E2	E3	CF	D3	0D
0x0000000000000120	0A	31	36	20	30	20	6F	62	6A	20	3C	3C	2F	4C	69	6E
0x0000000000000130	65	61	72	69	7A	65	64	20	31	2F	4C	20	31	36	37	39

4.1.4.5 Create a Project File 

After confirm the imported file, the Config setting, and the engineering verification, and then save it as a project file for production use. Click SavePrj, assign a file path and press OK to finish.

4.2 Production Mode

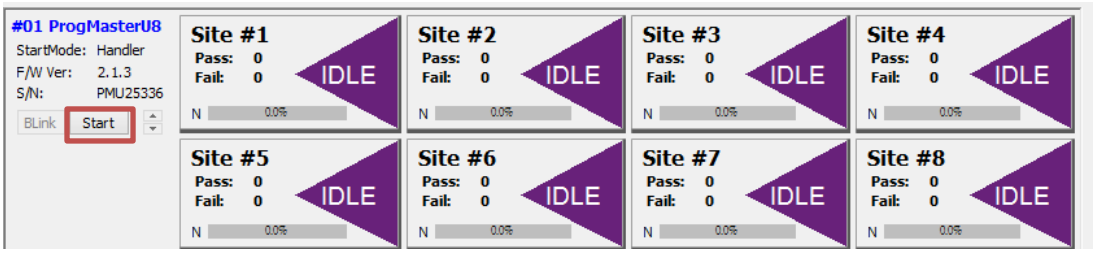
After saving a project file in Engineering Mode, then switch to Production Mode.



- **SelectPrj:** Select a PC and the project file from the SD card.
- **RunPrj:** Execute the project file.
- **StopPrj:** Stop the project file.
  Steps: Choose a project file > Execute > Stop.

NAND Flash provides two kinds of Start Mode:

A. Start from Manual Mode:
After executing the RunPrj, then press the start button on the screen (circled in red) or the one on the programmer to start programming.

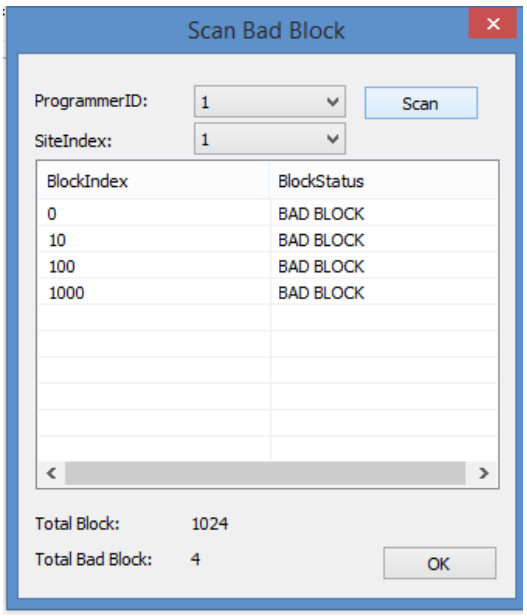


B. Start from Handler: Only for DediProg Automated Handlers.

V. Application Examples

5.1 How to Check the Bad Block Amounts Through the Guarded Area Count?

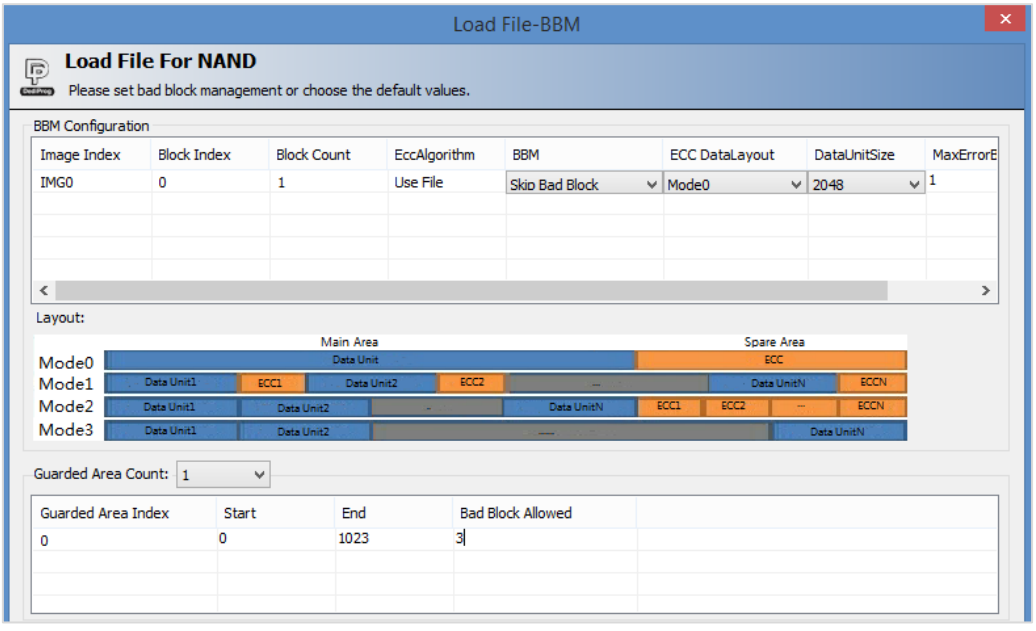
As the below image, there are bad blocks scattered in Block 0, Block 10, Block 100, and Block 1000 in a 1 Gb NAND Flash.



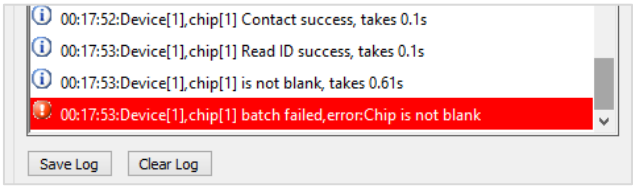
There are two ways to inspect the Bad Block distributions of each NAND Flash before production programming.

Method A: Inspect through Blank

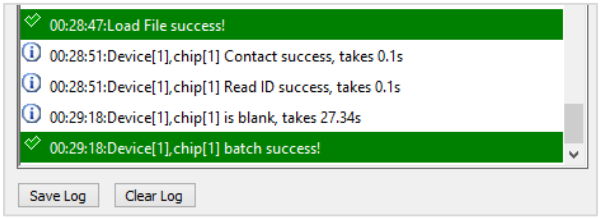
Step 1. Select an IC part number and execute Load. The software needs a file in order to go to the next page, which will be the Guarded Area Count. Therefore, select any file and click next to go to the BBM configuration and Guarded Area Count Setting.



Step 2. Open Config; only need to set up the Blank setting.
Step 3. Execute Auto Batch to start filtering.
A NAND Flash has a total of 1024 blocks, so the Guarded Area setting is from Block 0~1023. When Bad Block Allowed is 3 means only three Bad Blocks are allowed in the entire area. If there are four Bad Blocks, the software will use skip to analyze the Guarded Area conditions before executing the Blank. Since it has exceeded the limits, the software will show “Chip is not blank” message.



When Bad Block Allowed is 4 and four bad blocks appear, then the message will show “Chip is Blank” and “Batch Success”.



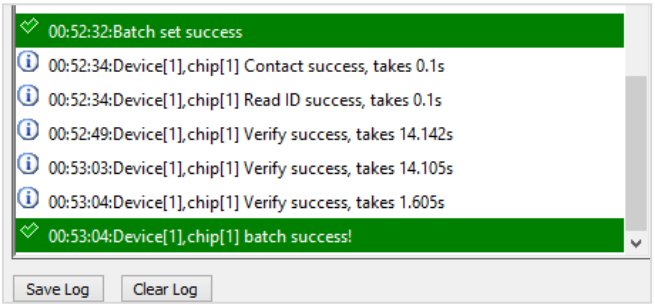
If there are no errors, then save it as a project file (*dprj) for large amount inspection.

Method B: Inspect through Verify

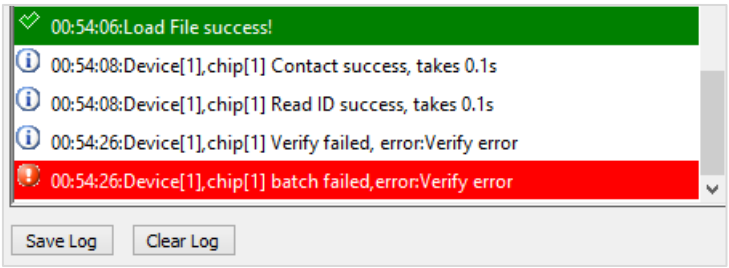
Step 1. Import all files that are 0xFF; set up the block size according the Bad Block Allowed in the Guarded Area, the equation is as below:

Total Block Number = Block count when loading the file + Quantity that Bad block Allowed
If total block number of a 1Gb Block is 1024, and the Bad Block Allowed is 4, then the Block Count should be 1020 when the file is loading. If you did not set up as the above equation, then the size of programming file will be bigger than the available blocks of the actual NAND flash when verifying, which might cause occurs.

Step 2. Execute Auto Batch (Only set up Verify).
When Bad Block Allowed is 4, it will take about 30 seconds to detect, but it will only take about 27 seconds through Blank.



If Bad Block Allowed is 3 and number has exceeded and appear error while verifying, it will take about 18 seconds, however, with Blank, it will only take about 1~2 seconds.



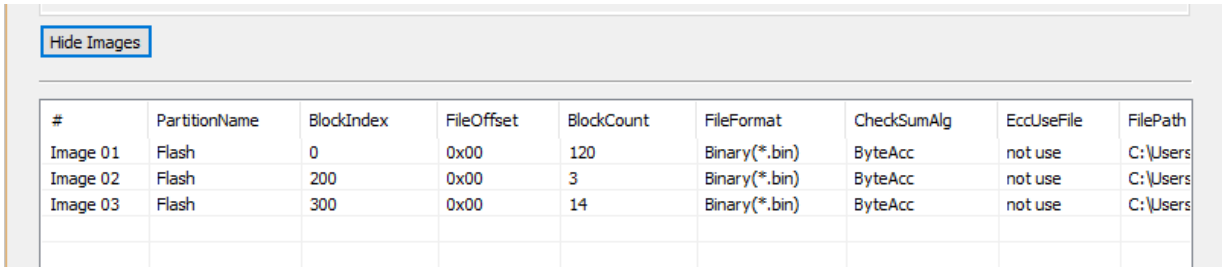
If there are no errors, then save it as a project file (*dprj) for large amount inspection.
As a result, method A is recommended for inspection, which is time saving and easier to operate.

5.2 If There Are Three Image Files Need to Be Written to Different Block Indexes While Using Skip Bad Block For BBM.

If there are three image files (not including the Spare) that need to be written to different block indexes while choosing Skip Bad Block for BBM; Error bit/Page is 3bit, and Block 0 and Block 1 cannot have bad blocks, then set up as the following:

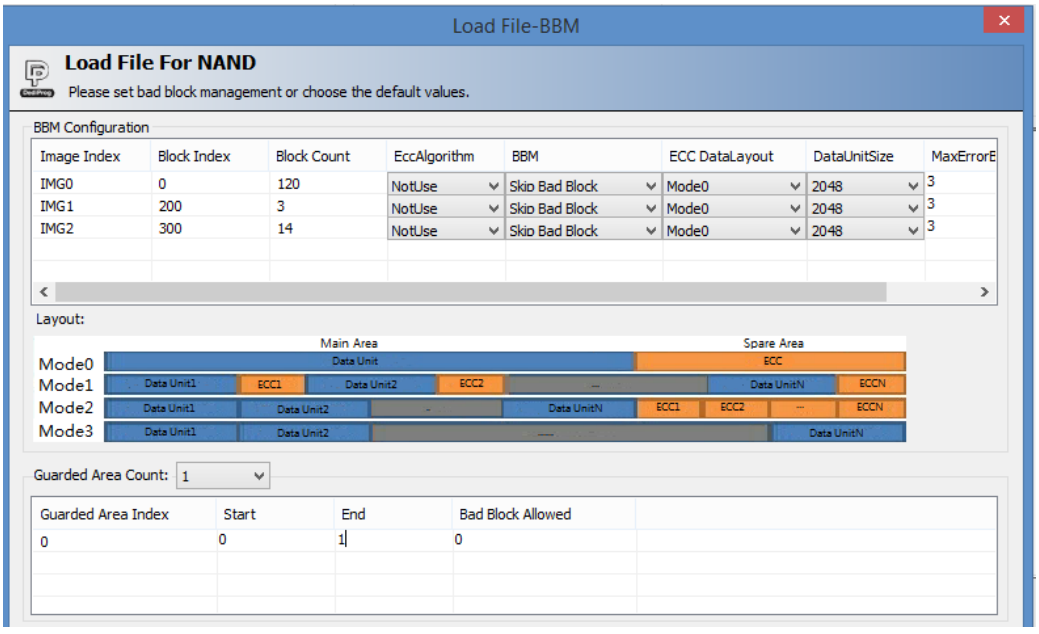
File	Block Start	Block End
A	0	119
B	200	202
C	300	313

Step 1. Load the file and set up BBM.

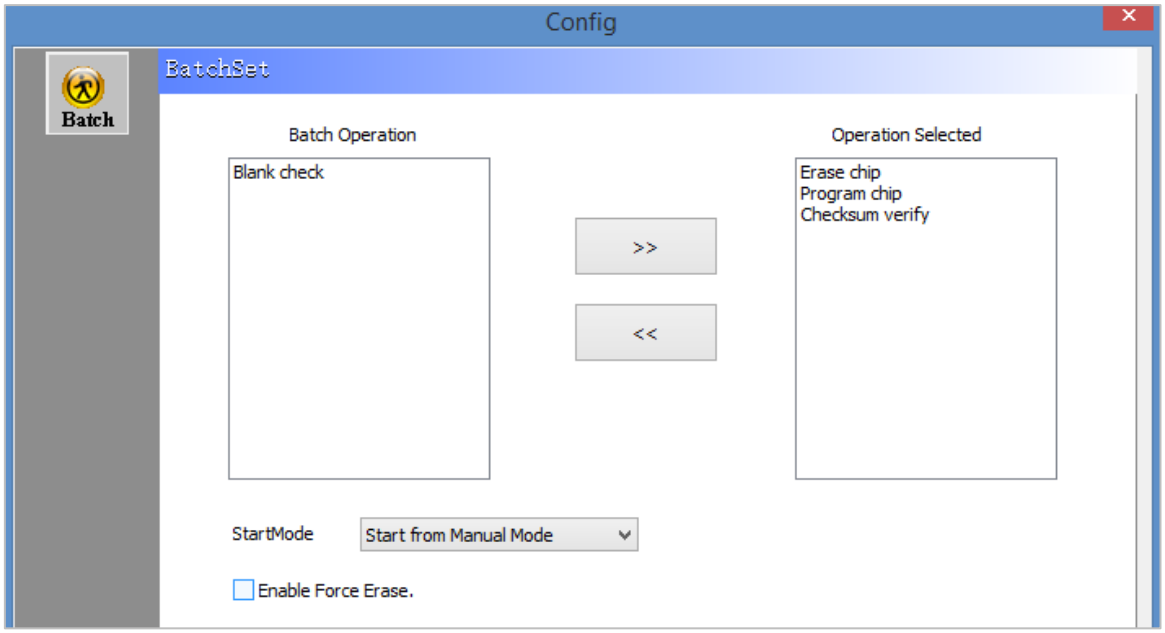


Step 2. Set up BBM Configuration and Guarded Area Count:

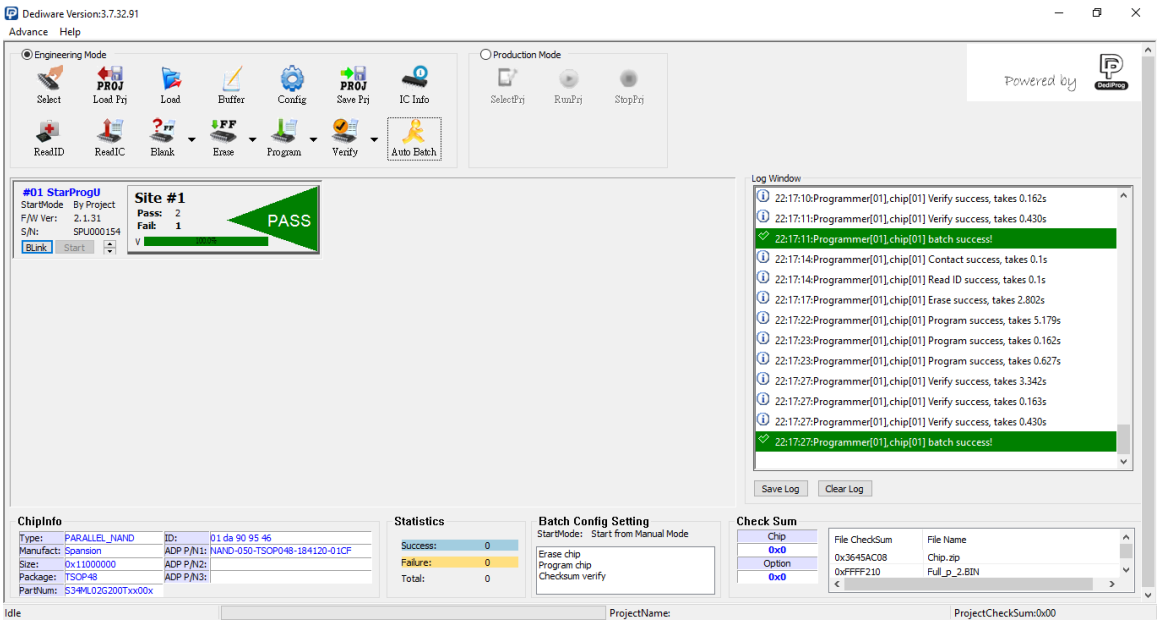
- Set MaxErrorBit as 3: it means Error bit/Page is 3bit.
- Choose Skip Bad Block in BBM: Skip when it detects bad blocks.
- Set Guarded Area Count as 1, Start as 0, End as 1, Bad Block Allowed as 0: Not allowing Bad Blocks in Block0/1.
- Click next and finish.



Step 3. Set up Config as Erase > Program > Verify.



Step 4. Execute Auto Batch.

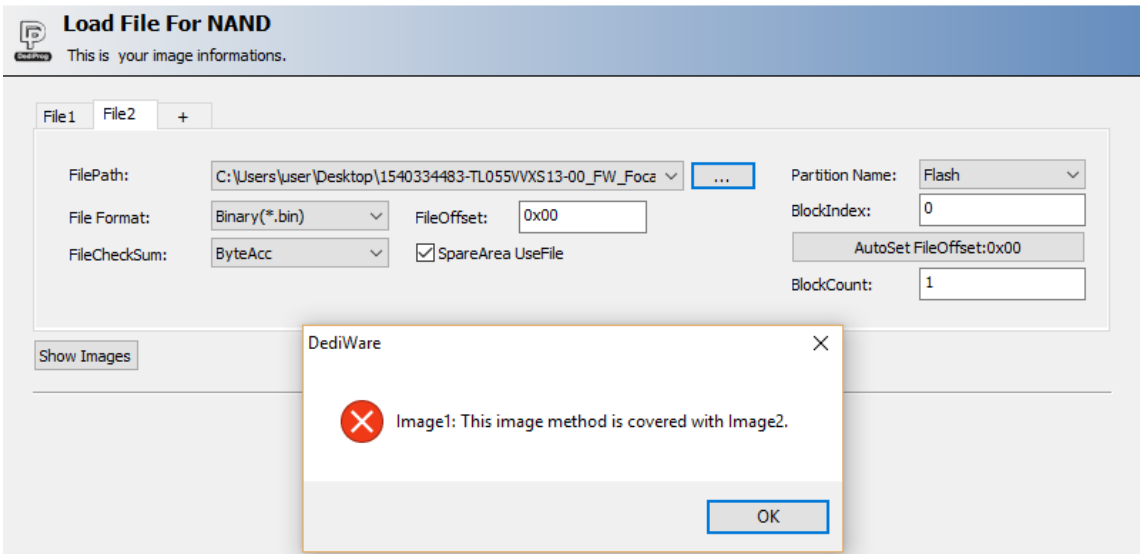


Step 5. After there are no more errors, save it as a project file for mass production.

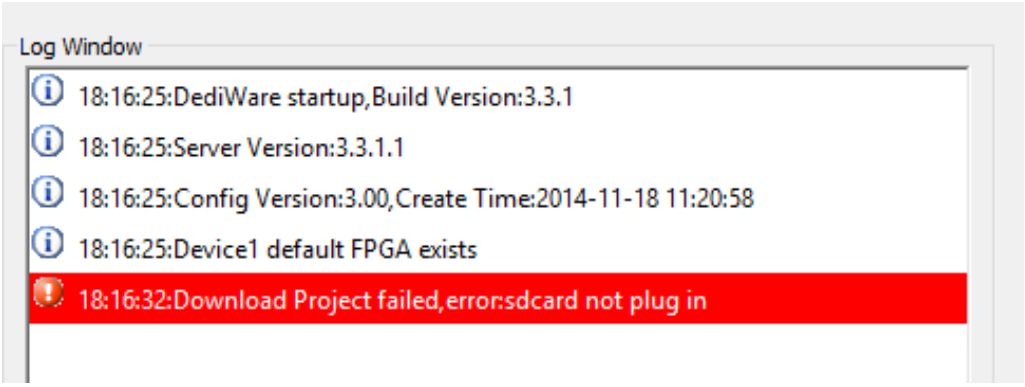
VI. Troubleshooting

Q1. What to do when message occurs during importing the file?

If it shows “This image method is covered with Image 2”, then it means you have imported multiple files and the data will overwrite on the previous file. Please ensure the data length and the starting address (BlockIndex) of each file will not be covered by other file.



Q2. What to do when it shows “Download Project failed, error:sdcard not plug in” in the production Mode?



1. Ensure the SD card is inserted to the programmer.
2. Format the SD card and retry.
3. Check if the SD card is damaged or not.

※ **It is recommended to use the industrial SD card that DediProg provided, which is included in the package, because it will be more stable and consistent.**

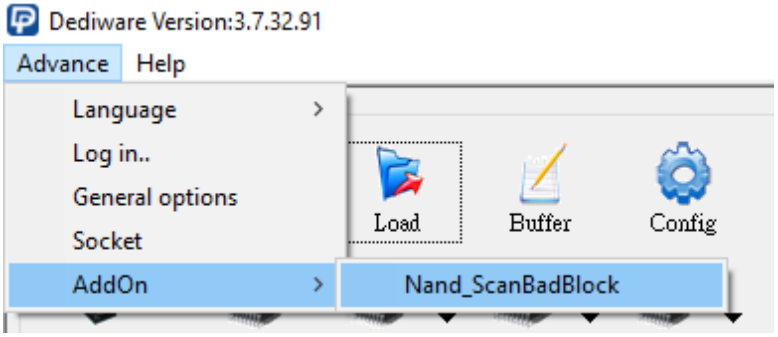
Q3. What if Log shows Contact Fail while programming?

Check the IC brand and the part number.
Check the socket adapter.
Check if the socket adaptor has bad connection.
Update the software and retry.

Q4. What if Log shows Erase Fail while erasing?

Go to Advance > AddOn > Nand_ScanBadBlock. If it shows “Too Many”, then it means there are too many bad blocks to erase. Please go to Config > Batch > Erase, and then select the Enable Force Erase option. In addition, execute Batch to erase all, or execute Erase immediately right after reselecting the part number.

※ **Please know that Enable Force Erase might erase all the factory bad blocks, please confirm before erasing.**



Q5. What if IC does not work on board even when the NAND has been programmed successfully?

Other than the welding and the circuit issues, there are some programming settings that you can check:

1. Check the programming file, and if it contains SpareArea.
2. Check BBM and ECC settings and see if the offset address of the imported file is correct.
3. If there are multiple Partitions in an IC, check if any partition still needs programming.
4. Is there any IC requires turning on the Internal ECC function, but it is not on.
5. Do you use master chip production? If yes, please refer to Q14
6. Are all IC blocks used in the programming file? If yes, please refer to Q14
7. Whether there is a specific block cannot be a bad block? If so, please use the Guarded Area function of the software
8. If the data between the partition and the partition is continuous, use the Guarded Area function of the software to detect the partition with continuous data to ensure that there are no bad blocks

Q6. Is it possible to read from a master NAND and duplicate to other NAND?

Since NAND Flash has Bit Error and Bad Block issues, so it is recommended to use the original file for production, and not duplication. However, if you use SPI NAND with Internal ECC on, then it can read and duplicate, but will need to process the bad blocks (if there is any bad blocks).

Q7. Why does it have different data after Read IC?

Since NAND Flash has Bit Error issue and has not done ECC verification, so it is normal to have a little offset.

Q8. What if verification keeps failing?

Please check the followings:

1. Is the condition for EccDataLayout, DataUnitSize, and MaxErrorBit of the BBM configuration complies with the Bit Error range for the IC?
2. If Bit Error occurs within the ECC area of the EccDtaLayout, then it will fail to verify.

Q9. What are format definitions of MBN, DEF, and CSV?

MBN Definition:

Byte0~3: Start Block (in red)

Byte4~7: End Block (in blue)

Byte8~11: Block Counts (in yellow)

Byte12~15: Remark (in green)

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	00	00	00	00	00	00	00	00	01	00	00	00	FF	FF	FF	FF
00000010	01	00	00	00	04	00	00	00	02	00	00	00	FF	FF	FF	FF
00000020	05	00	00	00	08	00	00	00	02	00	00	00	FF	FF	FF	FF
00000030	10	00	00	00	2F	00	00	00	11	00	00	00	FF	FF	FF	FF
00000040	30	00	00	00	CF	00	00	00	61	00	00	00	FF	FF	FF	FF
00000050	D0	00	00	00	4F	01	00	00	45	00	00	00	FF	FF	FF	FF
00000060	50	01	00	00	6F	01	00	00	11	00	00	00	FF	FF	FF	FF
00000070	70	01	00	00	0F	02	00	00	61	00	00	00	FF	FF	FF	FF
00000080	10	02	00	00	8F	02	00	00	45	00	00	00	FF	FF	FF	FF
00000090	90	02	00	00	9F	02	00	00	01	00	00	00	FF	FF	FF	FF
000000A0	A0	02	00	00	EF	02	00	00	02	00	00	00	FF	FF	FF	FF
000000B0	F0	02	00	00	0F	03	00	00	01	00	00	00	FF	FF	FF	FF
000000C0	10	03	00	00	7F	03	00	00	0A	00	00	00	FF	FF	FF	FF
000000D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Configuration after loading:

Partition Table Info

#	Start Block	Blcok Count	End Block
0	0	1	0
1	1	2	4
2	5	2	8
3	16	17	47
4	48	97	207
5	208	69	335
6	336	17	367
7	368	97	527
8	528	69	655
9	656	1	671
10	672	2	751
11	752	1	783

DEF Definitions:

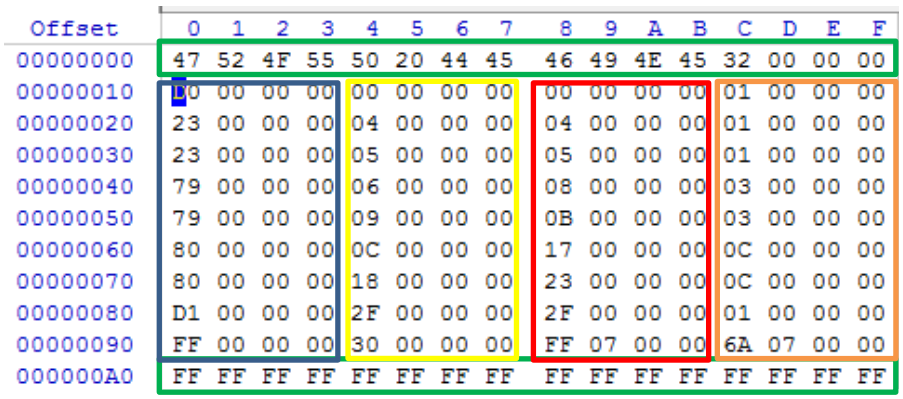
File head and file tail (in green, as the below image)

Type: Byte0~3 (in blue)

Start Block: Byte4~7 (in yellow)

End Block: Byte8~11 (in red)

Block Count: Byte12~15 (in orange)



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	47	52	4F	55	50	20	44	45	46	49	4E	45	32	00	00	00
00000010	00	00	00	00	00	00	00	00	00	00	00	00	01	00	00	00
00000020	23	00	00	00	04	00	00	00	04	00	00	00	01	00	00	00
00000030	23	00	00	00	05	00	00	00	05	00	00	00	01	00	00	00
00000040	79	00	00	00	06	00	00	00	08	00	00	00	03	00	00	00
00000050	79	00	00	00	09	00	00	00	0B	00	00	00	03	00	00	00
00000060	80	00	00	00	0C	00	00	00	17	00	00	00	0C	00	00	00
00000070	80	00	00	00	18	00	00	00	23	00	00	00	0C	00	00	00
00000080	D1	00	00	00	2F	00	00	00	2F	00	00	00	01	00	00	00
00000090	FF	00	00	00	30	00	00	00	FF	07	00	00	6A	07	00	00
000000A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Configuration after loading:

Partition Table Info

#	Start Block	Blcok Count	End Block
0	0	1	0
1	4	1	4
2	5	1	5
3	6	3	8
4	9	3	11
5	12	12	23
6	24	12	35
7	47	1	47
8	48	1898	2047

CSV uses semicolon ";" for separation. Definitions:
Start Block; End Block; Block count; special option; comment

```
0;5;4;0xffffffff;burn_mtd0
6;12;5;0xffffffff;burn_mtd1
13;68;52;0xffffffff;burn_mtd2
69;220;148;0xffffffff;burn_mtd3
221;540;296;0xffffffff;burn_mtd4
541;692;136;0xffffffff;burn_mtd5
693;954;254;0xffffffff;burn_mtd6
955;957;2;0xffffffff;burn_mtd7
958;961;3;0xffffffff;burn_mtd8
962;964;2;0xffffffff;burn_mtd9
965;980;13;0xffffffff;burn_mtd10
981;983;2;0xffffffff;burn_mtd11
984;986;2;0xffffffff;burn_mtd12
987;989;2;0xffffffff;burn_mtd13
```

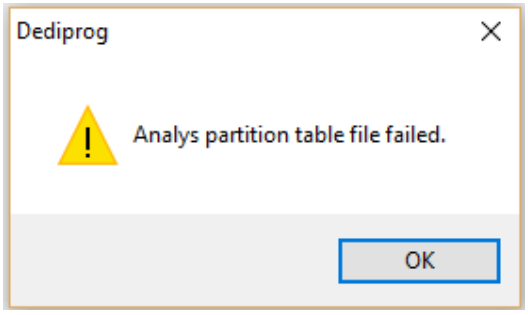
Configuration after loading:

Partition Table Info

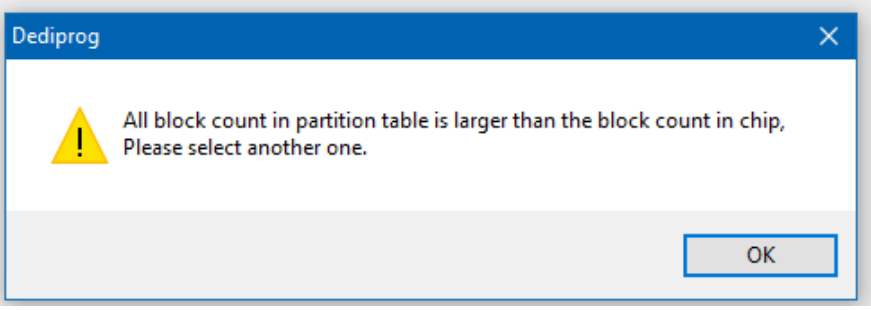
#	Start Block	Blcok Count	End Block
0	0	4	5
1	6	5	12
2	13	52	68
3	69	148	220
4	221	296	540
5	541	136	692
6	693	254	954
7	955	2	957
8	958	3	961
9	962	2	964
10	965	13	980
11	981	2	983

Q10. What if error appears when loading the Partition Table?

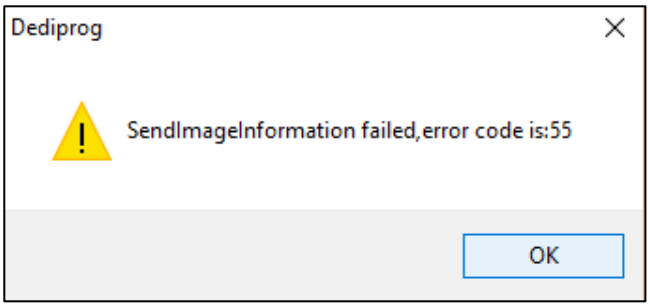
It means it cannot analyze the file format correctly.



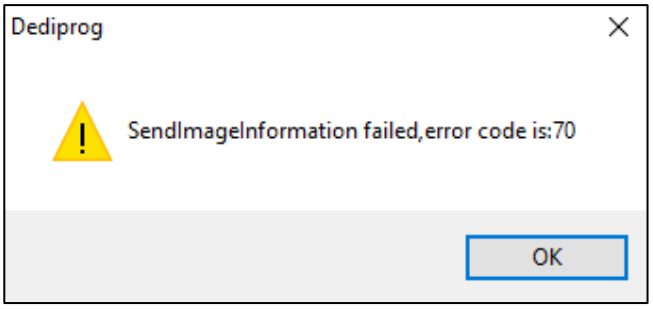
If block number of the table has exceeded the number of NAND, then please check the part number and the partition table.



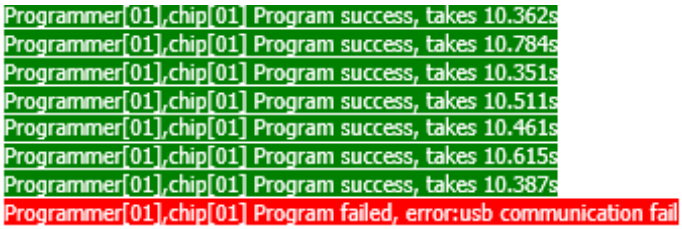
The image file is too small to find the partition table.



The image file is too big that it exceeds the NAND size.



Q11. What if it shows USB communication fail during programming?



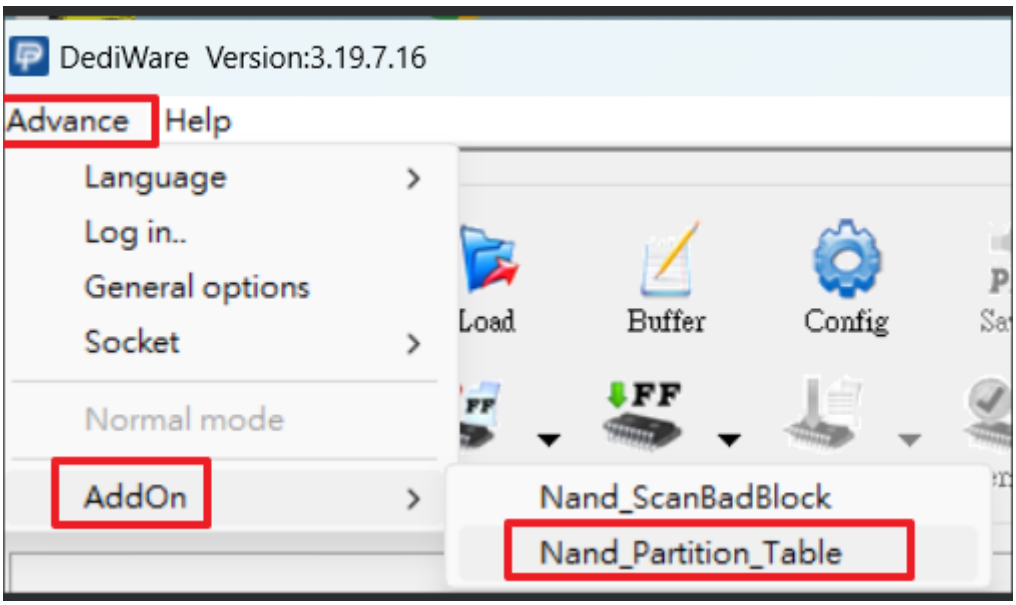
First, please check if the NAND and the image file size are the same. If it has a Bad Block, then the last block of the image file will not be able to write into the NAND, then the error message will appear. Usually, the image file size will be smaller than NAND, and should have preserved space for skipping or other appropriate actions when bad block appears during programming. Second, if factory bad blocks markings are missing during programming, it might cause failure.

Q12. What could be the reasons that cause programming failure during the re-program process?

- If the image only has Main area (Not including SpareArea) and the SpareArea useFile option is selected when loading the file, then it will fail to erase during programming. The reason is that the BI of every block in the IC has been written incorrectly, which will cause too many bad blocks.
- If there are too many restrictions for EccDataLayout, DataUnitSize, and MaxErrorBit.
- If NAND is erase-able and the Force Erase is selected or Read ID has not been done while the first time programming and erasing, then it will erase all factory bad blocks and will timeout or fail during the process of programming or verification.

Q13. How to use DediWare software to generate Partition Table?

Go to Advance > AddOn > NAND_Partition_Table

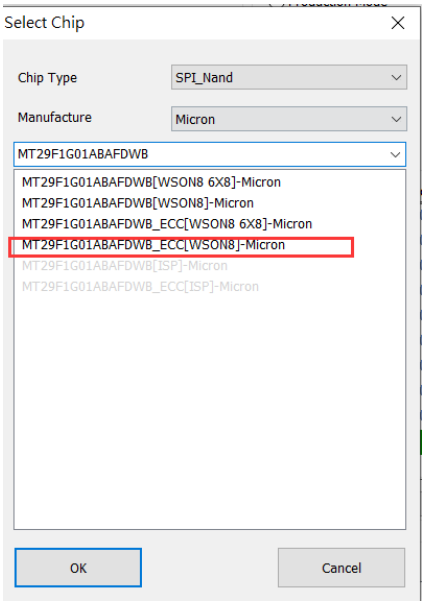


Q14. How to solve the problem of not being able to boot up after using master chip programming?

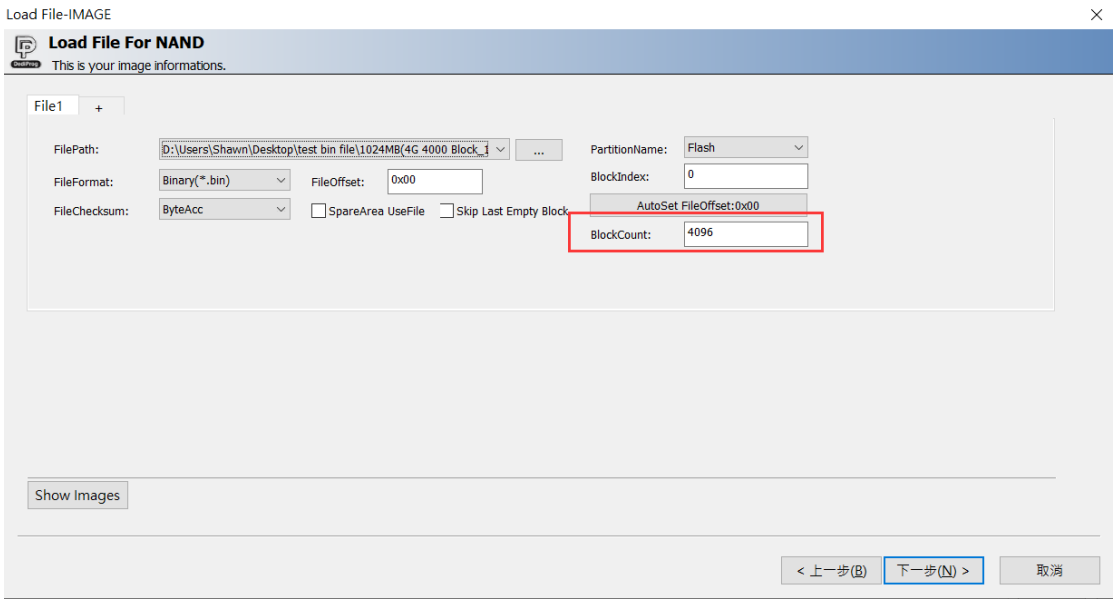
Before solving the problem, please study the user manual and confirm the following points

1. Confirm whether the programming file is correct and whether the contains SpareArea (OOB)? If it contains SpareArea (OOB), check "Spare area use file" box when loading the file.

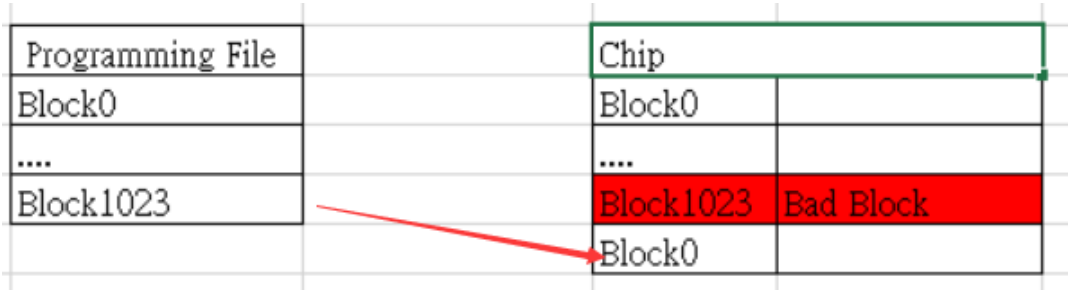
2. Do you use Internal ECC? If Chip provides Internal ECC and you want to use this function, select PartName that contains the _ECC string when selecting PartName.



3. How to check how many blocks are used in the programming file.
   Example:
   The total number of blocks in W29N04GVBIAF is 4096. When the DediWare software loads the programming file, if the BlockCount is equal to the total number of blocks in the Chip, it can be known that it is produced by copying from the master.



4. What problems will I encounter when the programming file uses all the blocks of the IC?
   Example:
   The chip capacity of W25N01GVZEIG has a total of 1024 (Block0~1023) blocks. When programming a file with a size of 1024 blocks, if the programmed Chip Block1023 is a bad block, the data of the programming file Block1023 will be programmed to Block0, causing Block0 to be overwritten. Block0 data can be wrong.



5. Suggested ways to improve

A. Use Partition Table to load programming files
   Please refer to Q9 for the format of Partition Table.
   Please use binary editing software to generate MBN and DEF formats. When saving files, please type .mbn or .def with the extension. Please use excel or a text editor to generate the CSV format. When saving the file, please type .csv as the extension.

B. Manually set the programming file to load & generate Partition Table files.
   Please refer to Q13.

C. If you use Partition Table to produce, how to deal with Chip still not working?
   Please refer to Q5.

Q15. What does the error message in production mode mean?

- **Not Enough Valid Block (0x1):** If the Guarded Area function is set, it means that the detected bad block is greater than the set allowable value of bad blocks within the set range.
- **MORE THAN 255 BAD BLOCKS (0x2) :** IC has more than 255 bad blocks。
- **WRITE ENABLE FAIL (0x3):** After SPI NAND sends the WRITE ENABLE command, it will check whether the Write Enable Latch (WEL) bit of the Status Register is set to 1, if not, it will report this error.
- **P_FAIL BIT = 1 (0x4) :** The Program fail (P_Fail) bit of the Status Register is set to 1. Please refer to the datasheet and ask the original equipment manufacturer for details on why this error is caused.
- **SCAN BAD BLOCK FAIL (0x5) :** This error will be reported when a bit error occurs when scanning for bad blocks.

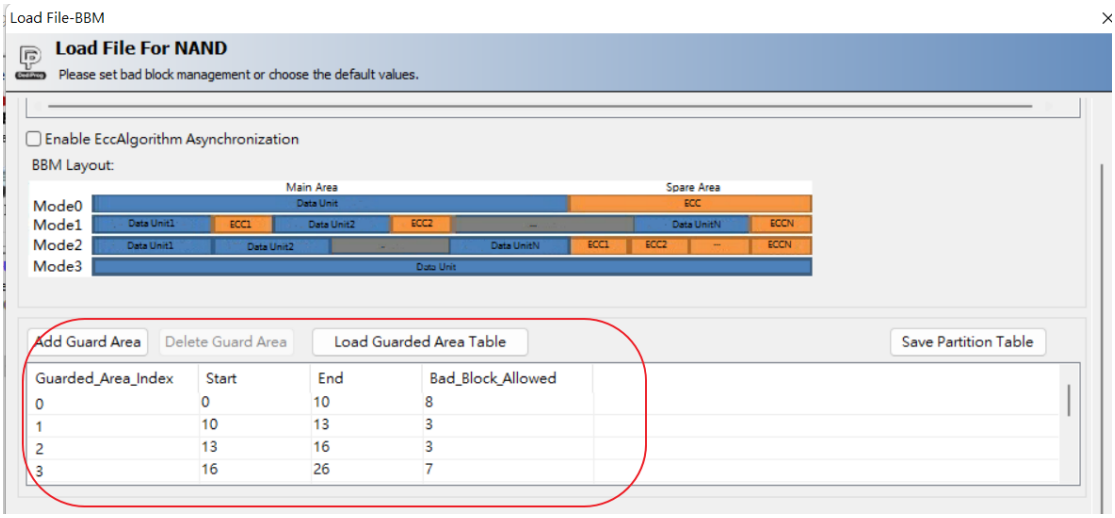
Q16. Problems that may be encountered when using Read IC with skip bad block to read Chip data

Please refer to this document:

https://eip.dediprog.com/ftp/download.php?fileNo=MTA3MzI=

Q17. What is the function description of Auto calculate Guard Area?

Check "Auto calculate Guard Area", the software will calculate how many bad blocks are allowed in each partition in the partition table, and automatically set the Guard Area on the "Load File-BBM" page



For Example:

The partition table is as follows (assuming that the total number of NAND blocks is 1024, block0~1023)

Partition index	Start Block	Block Count	End Block
0	0	3	9
1	10	1	12
2	13	1	1023

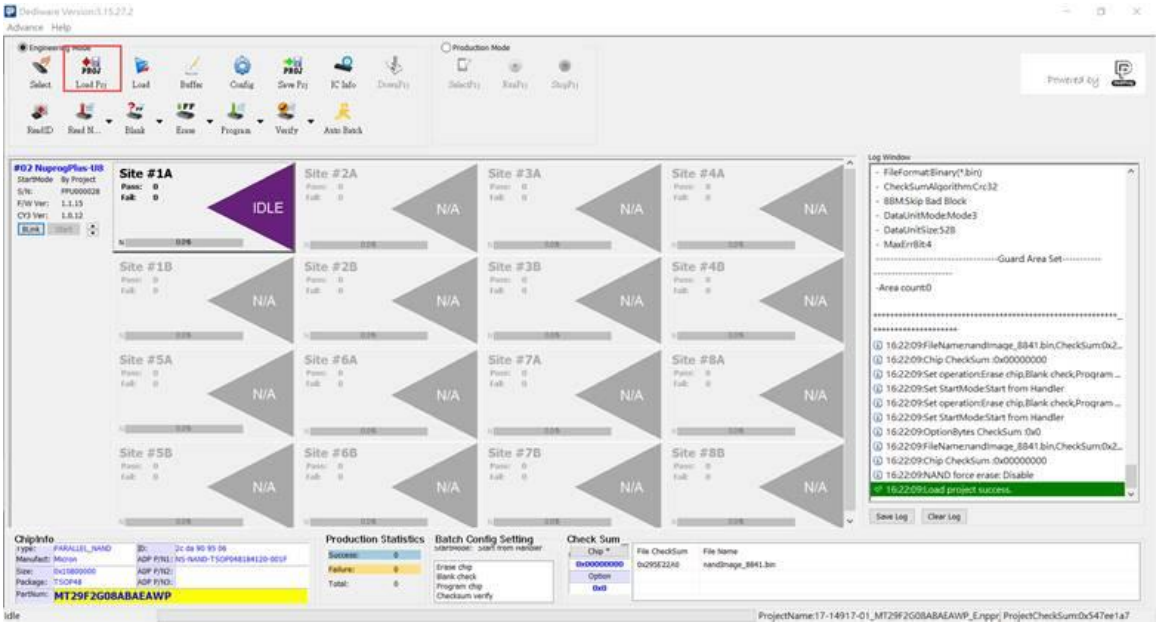
Use partition 0 to illustrate how each partition calculates how many bad blocks are allowed:

1. First calculate how many blocks are available in this partition: 10(End Block of Partition index 0) - 0(Start Block of Partition index 0) = 10(Block0~9)
2. Then calculate how many bad blocks are allowed in this partition: 10 - 3(Block Count of Partition index 0) = 7

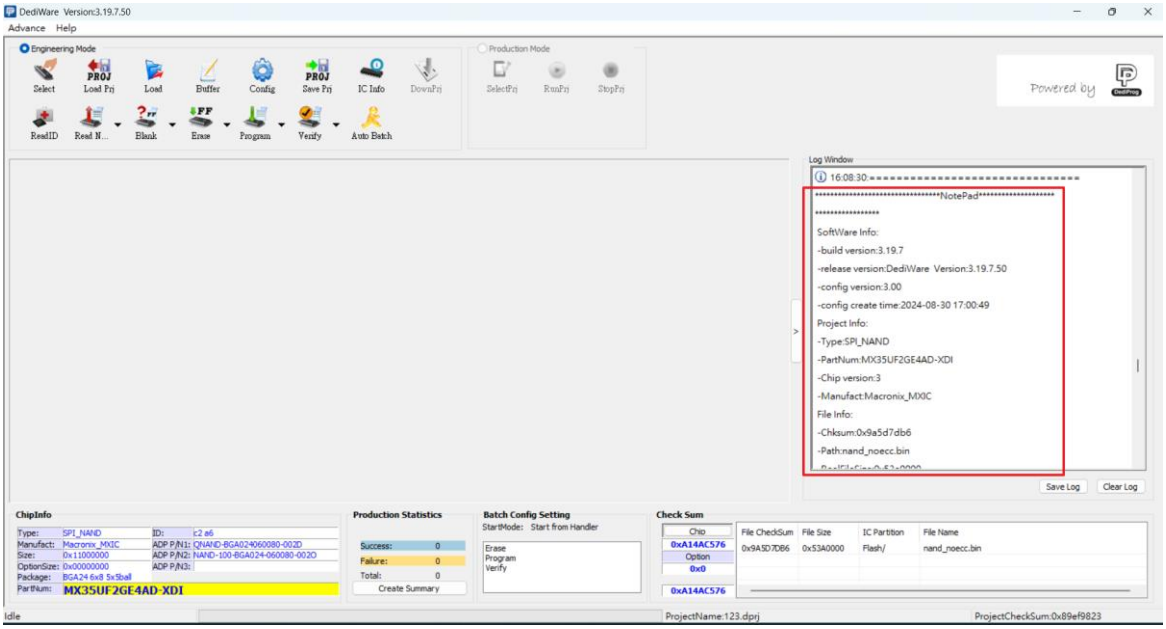
If the IC to be programmed has 8 bad blocks, which are block 3, 4, 5, 6, 7, 8, 9, 10, because the "bad block allowance" setting value of partition 0 is 7, so the software checks the Guard Area Will judge this IC as NG

Q18. How to view file information, BBM configuration and Guarded Area settings in Project?

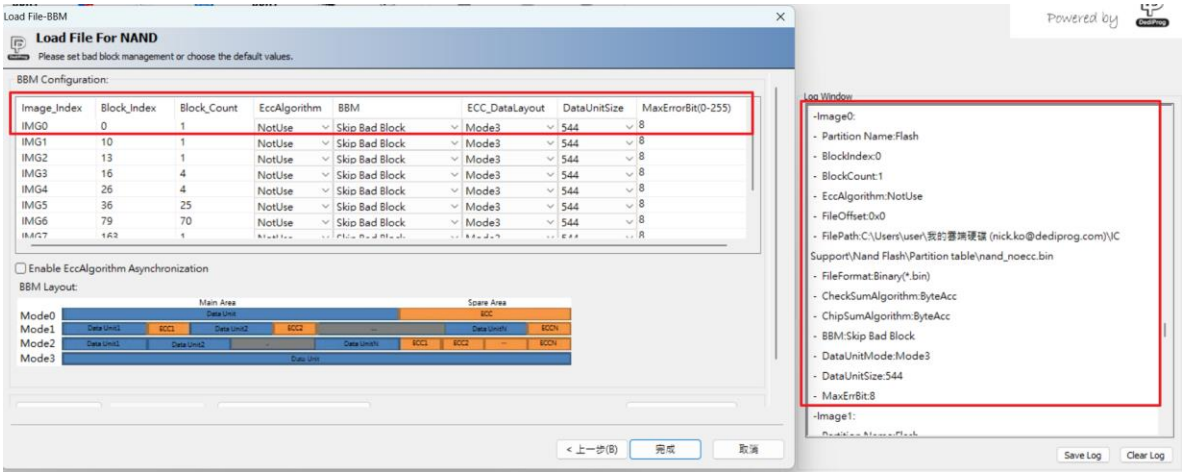
1. Open Dediware software and load Prj in Engineering mode.



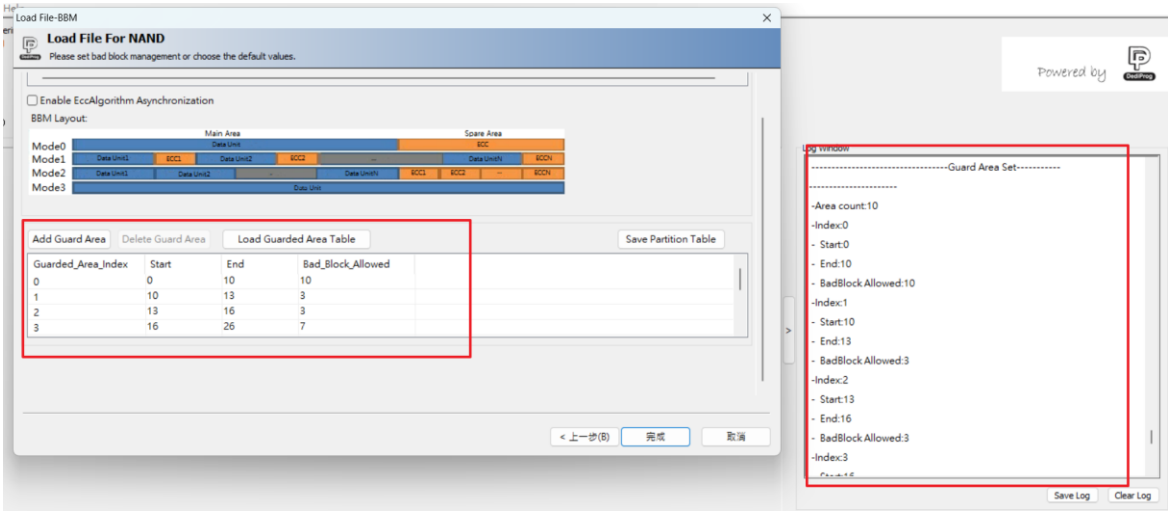
2. Slide up the Log Window on the right side of the software to see “NotePad”.



3. Log Window can see the BBM settings of each Image



You can also see the Guard Area settings in the Log Window.



Q19. How is Chip Checksum calculated when using partition tables?

The purpose of the partition table is to divide the data in the programming file you provide into several different partitions and load them into the programmer's Buffer.

For example: The partition table used by IC W29N02GZBJBF is as follows.

#	Start Block	Block Count	End Block
0	0	3	7
1	8	2	15
2	24	11	47
3	72	1	75
4	88	1	91
5	96	1	99
6	108	5	115
7	128	240	495
8	1008	3	1011

The capacity of IC W29N02GZBJBF has a total of 2048 blocks (Block0~2047). Assuming that the programming file size is also 2048 blocks, the data of the programming file will be loaded into the Buffer through the partition table as shown below. For data that has not been loaded into the Buffer by the partition table, the default value will be filled with the 0xFF value.

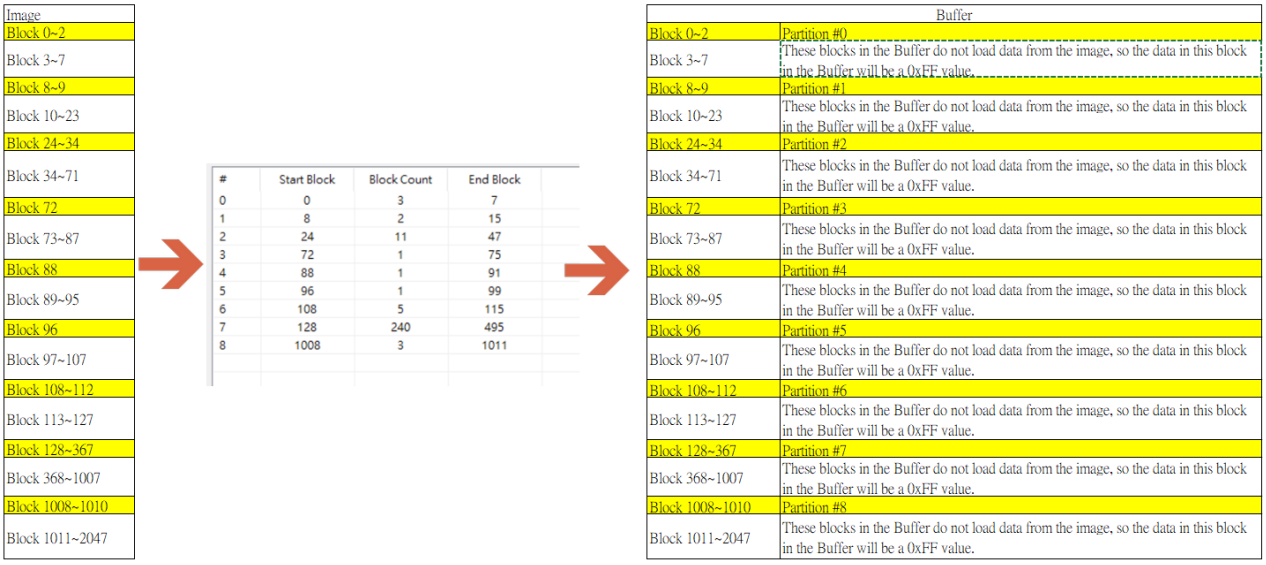
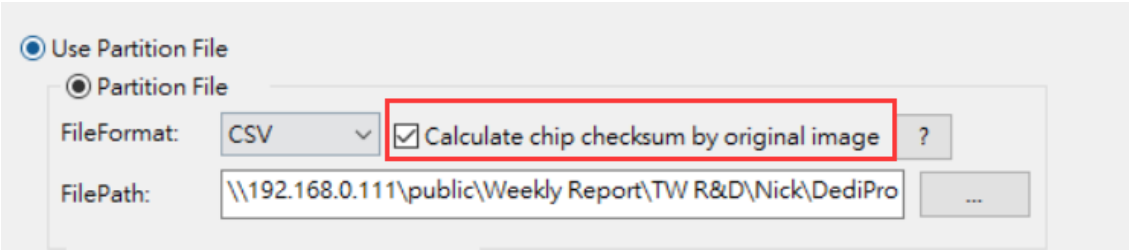


Image
Block 0~2
Block 3~7
Block 8~9
Block 10~23
Block 24~34
Block 34~71
Block 72
Block 73~87
Block 88
Block 89~95
Block 96
Block 97~107
Block 108~112
Block 113~127
Block 128~367
Block 368~1007
Block 1008~1010
Block 1011~2047

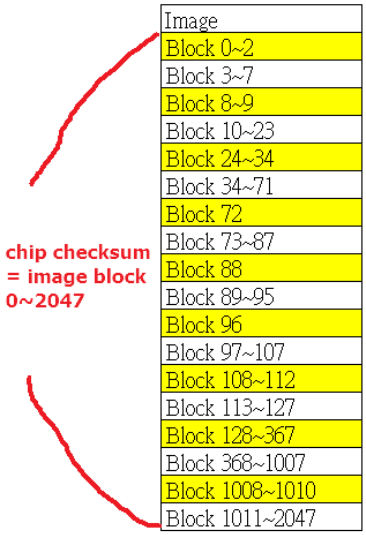
#	Start Block	Block Count	End Block
0	0	3	7
1	8	2	15
2	24	11	47
3	72	1	75
4	88	1	91
5	96	1	99
6	108	5	115
7	128	240	495
8	1008	3	1011

Buffer	
Block 0~2	Partition #0
Block 3~7	These blocks in the Buffer do not load data from the image, so the data in this block in the Buffer will be a 0xFF value.
Block 8~9	Partition #1
Block 10~23	These blocks in the Buffer do not load data from the image, so the data in this block in the Buffer will be a 0xFF value.
Block 24~34	Partition #2
Block 34~71	These blocks in the Buffer do not load data from the image, so the data in this block in the Buffer will be a 0xFF value.
Block 72	Partition #3
Block 73~87	These blocks in the Buffer do not load data from the image, so the data in this block in the Buffer will be a 0xFF value.
Block 88	Partition #4
Block 89~95	These blocks in the Buffer do not load data from the image, so the data in this block in the Buffer will be a 0xFF value.
Block 96	Partition #5
Block 97~107	These blocks in the Buffer do not load data from the image, so the data in this block in the Buffer will be a 0xFF value.
Block 108~112	Partition #6
Block 113~127	These blocks in the Buffer do not load data from the image, so the data in this block in the Buffer will be a 0xFF value.
Block 128~367	Partition #7
Block 368~1007	These blocks in the Buffer do not load data from the image, so the data in this block in the Buffer will be a 0xFF value.
Block 1008~1010	Partition #8
Block 1011~2047	These blocks in the Buffer do not load data from the image, so the data in this block in the Buffer will be a 0xFF value.

When loading the partition table, check "Calculate chip checksum by original image", which literally means using the original Image data to calculate the chip checksum, that is, using image block 0~2047 data to calculate the chip checksum.

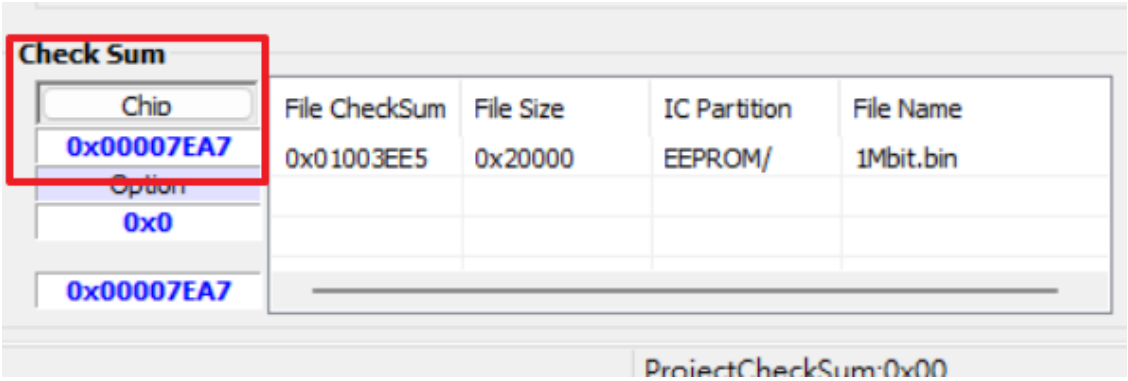


chip checksum = image block 0~2047.

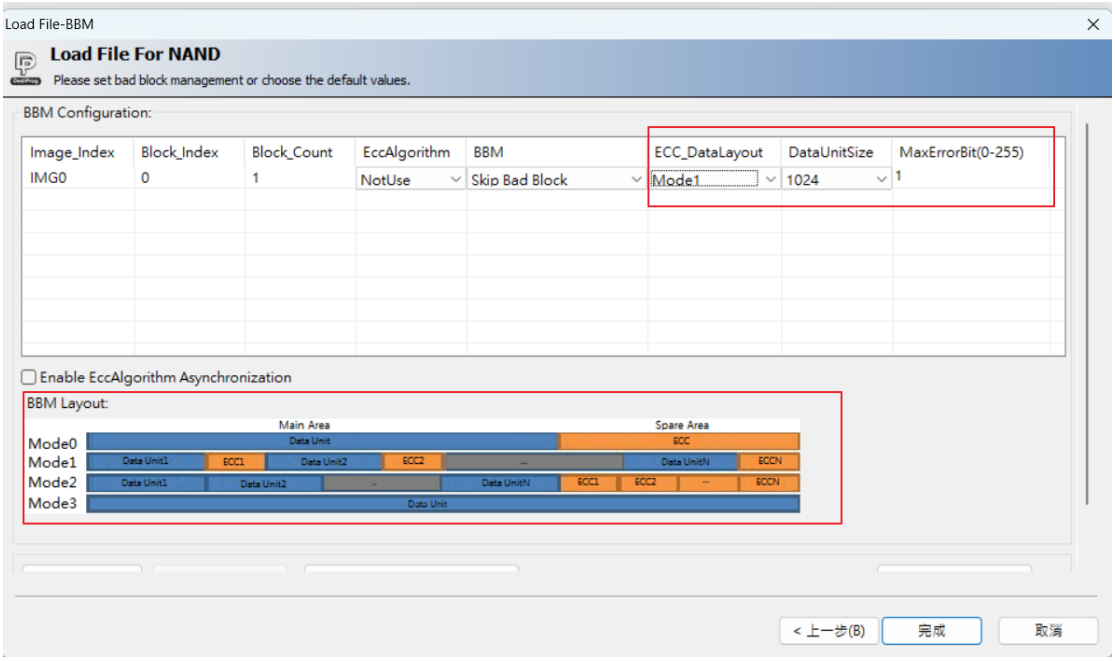


Uncheck "Calculate chip checksum by original image", that is, calculate the chip checksum using the data in the Buffer, as shown in the figure below.

It should be noted that whether "Calculate chip checksum by original image" is checked or not will only affect the display of the chip checksum at the bottom right of the software. The data of the Buffer is based on the partition table, cutting the data of the Image and loading it into the Buffer. When "Calculate chip checksum by original image" is checked, the Buffer Checksum in the Buffer will be different from the Chip Checksum.

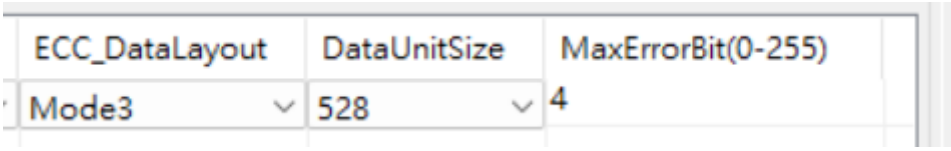


VII. Supplementary explanation of BBM Layout function



The BBM Layout feature only functions when Verify is executed on DediWare. During Verify execution, DediWare not only verifies data accuracy but also conducts a Bit Error check to ensure that the IC can boot up normally after being mounted on the board.

BBM Layout will check Error Bits based on the following settings.

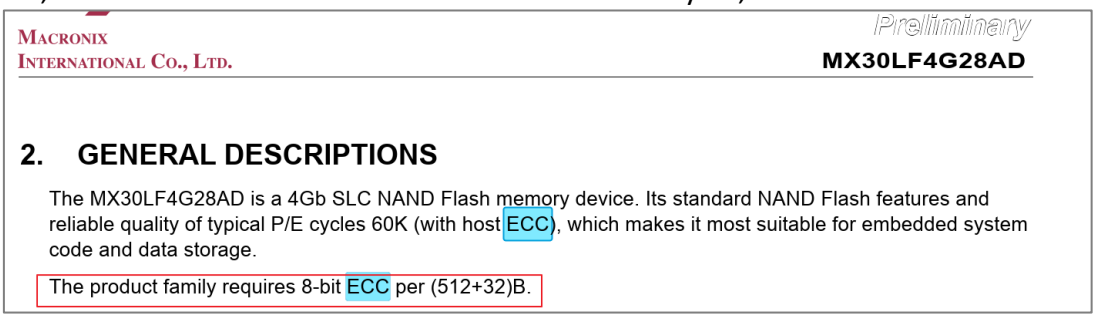


- **Ecc_DataLayout**：Four data layout modes (Mode 0, 1, 2, 3) are provided. Please select the corresponding mode based on the arrangement of data in your programming file. The default mode is Mode 3.
  ※ Note: The ECC area (orange blocks) in Mode 0, 1, and 2 does not allow any Bit Errors. Therefore, if Bit Errors occur in the ECC area (orange blocks) of Mode 0, 1, or 2 during verify, the verify will fail.

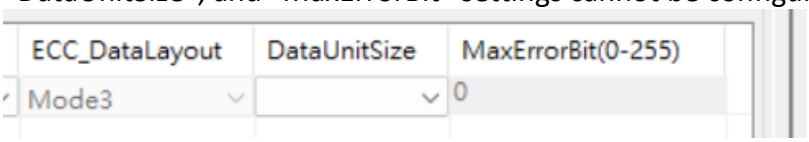
- **DataUnitSize**：The size of the Data Unit under data layout modes (Mode 0, 1, 2, 3).
- **MaxErrorBit**：The maximum allowable number of Bit Errors for each Data Unit. Please set the maximum allowable number of Bit Errors for each DataUnit based on the ECC correction capability.

Note:

1. The default values for DataUnitSize and MaxErrorBit are defined according to the Datasheet. For example, in the case of MXIC's Parallel NAND MX30LF4G28AD, it requires 8-bit ECC per (512+32)B. Therefore, the default value for DataUnitSize is set to 544 bytes, and MaxErrorBit is set to 8 bits.



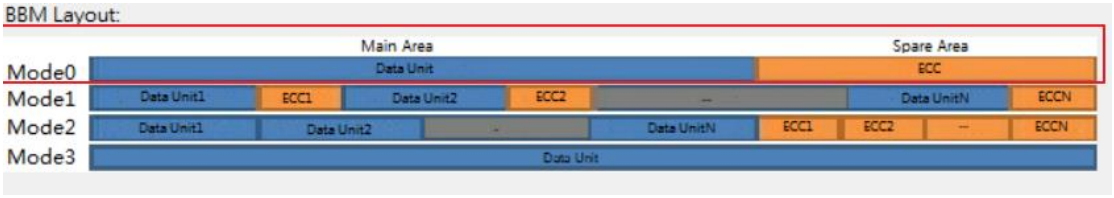
2. If the part number includes "XXXX_ECC" (with "_ECC" appended to the IC part number), the “Ecc_DataLayout”, “DataUnitSize”, and “MaxErrorBit“ settings cannot be configured.



Parts with "_ECC" in the part number will enable Internal ECC. During Verify in Dediware, it defaults to checking for zero Bit Errors. If any Bit Errors are detected during verify, the verify will fail. Because parts with "_ECC" in the part number enable Internal ECC, theoretically, any Bit Errors detected during verification will be automatically corrected by the IC's internal ECC engine. If Bit Errors are detected during verification, it indicates that the number of Bit Errors exceeds the capability of the IC's internal ECC engine to correct.

Using a NAND with a Page size structure of 2048+64 bytes as an example, let's illustrate the settings for Ecc_DataLayout, DataUnitSize, and MaxErrorBit：

A. Ecc_DataLayout：Mode 0



DataUnitSize: 2048 ，MaxErrorBit: 1.

In the programming file, the data arrangement for each page is 2048+64, as shown in the table below:

Data Unit	ECC(spare area)
2048byte	64 byte

Data Unit: During verify, this area allows a maximum of 1 Bit Error. If there are 2 or more Bit Errors detected during the check, it will be considered as verify fail.

ECC: The orange area does not allow any Bit Errors. If any Bit Error is detected during verify in this area, it will result in verify fail.

B. Ecc_DataLayout：Mode 1

1. DataUnitSize：1024 ，MaxErrorBit：1，In the programming file, each page's data is divided into two equal parts, with the arrangement as 1024+32+1024+32, as shown in the table below:

Data Unit1	ECC1	Data Unit2	ECC2
1024byte	32byte	1024byte	32byte

- Data Unit1~2: During verify, each Data Unit area allows a maximum of 1 Bit Error. If there are 2 or more Bit Errors detected during the check, it will be considered as verify fail.
- ECC1~2: During verify, each ECC region does not allow any Bit Errors. If any Bit Errors are detected during the check, it will be considered as verify fail.

2. DataUnitSize：512 ，MaxErrorBit：1，In the programming file, each page's data is divided into four equal parts, with the arrangement as 512+16+512+16+512+16+512+16, as shown in the table below:

Data Unit1	ECC1	Data Unit2	ECC2	Data Unit3	ECC3	Data Unit4	ECC4
512byte	16 byte	512byte	16 byte	512byte	16 byte	512byte	16 byte

- Data Unit1~4: During verify, each Data Unit area allows a maximum of 1 Bit Error. If there are 2 or more Bit Errors detected during the check, it will be considered as verify fail.
- ECC1~4: During verify, each ECC region does not allow any Bit Errors. If any Bit Errors are detected during the check, it will be considered as verify fail.

3. DataUnitSize：256 ，MaxErrorBit：1，In the programming file, each page's data is divided into eight equal parts, with the arrangement as 256+8+256+8+256+8+256+8+256+8+256+8+256+8+256+8, as shown in the table below:

Data Unit1	ECC1	Data Unit2	ECC2	Data Unit3	ECC3	Data Unit4	ECC4
256byte	8 byte	256byte	8 byte	256byte	8 byte	256byte	8 byte
Data Unit5	ECC5	Data Unit6	ECC6	Data Unit7	ECC7	Data Unit8	ECC8
256byte	8 byte	256byte	8 byte	256byte	8 byte	256byte	8 byte

- Data Unit1~8: During verify, each Data Unit area allows a maximum of 1 Bit Error. If there are 2 or more Bit Errors detected during the check, it will be considered as verify fail.
- ECC1~8: During verify, each ECC region does not allow any Bit Errors. If any Bit Errors are detected during the check, it will be considered as verify fail.

C. Ecc_DataLayout：Mode 2

1. DataUnitSize：1024 ，MaxErrorBit：1，In the programming file, each page's data is divided into two equal parts, with the arrangement as 1024+1024+32+32, as shown in the table below:

Data Unit1	Data Unit2	ECC1	ECC2
1024byte	1024 byte	32byte	32 byte

- Data Unit1~2: During verify, each Data Unit area allows a maximum of 1 Bit Error. If there are 2 or more Bit Errors detected during the check, it will be considered as verify fail.
- ECC1~2: During verify, each ECC region does not allow any Bit Errors. If any Bit Errors are detected during the check, it will be considered as verify fail.

2. DataUnitSize：512 ，MaxErrorBit：1，In the programming file, each page's data is divided into four equal parts, with the arrangement as 512+512+512+512+16+16+16+16, as shown in the table below:

Data Unit1	Data Unit2	Data Unit3	Data Unit4	ECC1	ECC2	ECC3	ECC4
512byte	512 byte	512byte	512byte	16 byte	16 byte	16 byte	16 byte

- Data Unit1~4: During verify, each Data Unit area allows a maximum of 1 Bit Error. If there are 2 or more Bit Errors detected during the check, it will be considered as verify fail.
- ECC1~4: During verify, each ECC region does not allow any Bit Errors. If any Bit Errors are detected during the check, it will be considered as verify fail.

3. DataUnitSize：256 ，MaxErrorBit：1，In the programming file, each page's data is divided into eight equal parts, with the arrangement as 256+256+256+256+256+256+256+256+8+8+8+8+8+8+8+8, as shown in the table below:

Data Unit1	Data Unit2	Data Unit3	Data Unit4	Data Unit5	Data Unit6	Data Unit7	Data Unit8
256byte	256byte	256byte	256byte	256byte	256byte	256byte	256byte
ECC1	ECC2	ECC3	ECC4	ECC5	ECC6	ECC7	ECC8
8byte	8byte	8byte	8byte	8byte	8byte	8byte	8byte

- Data Unit1~8: During verify, each Data Unit area allows a maximum of 1 Bit Error. If there are 2 or more Bit Errors detected during the check, it will be considered as verify fail.
- ECC1~8: During verify, each ECC region does not allow any Bit Errors. If any Bit Errors are detected during the check, it will be considered as verify fail.

D. Ecc_DataLayout：Mode 3

1. DataUnitSize：2112 ，MaxErrorBit：1，In the programming file, each page's data is arranged as 2112 bytes, as shown in the table below:

Data Unit1
2112byte

- Data Unit1: During verify, each Data Unit area allows a maximum of 1 Bit Error. If there are 2 or more Bit Errors detected during the check, it will be considered as verify fail.

2. DataUnitSize：1056 ，MaxErrorBit：1，In the programming file, each page's data is divided into two equal parts, with the arrangement as 1056+1056, as shown in the table below:

Data Unit1	Data Unit2
1056byte	1056byte

- Data Unit1~2: During verify, each Data Unit area allows a maximum of 1 Bit Error. If there are 2 or more Bit Errors detected during the check, it will be considered as verify fail.

3. DataUnitSize：528 ，MaxErrorBit：1，In the programming file, each page's data is divided into four equal parts, with the arrangement as 528+528+528+528, as shown in the table below:

Data Unit1	Data Unit2	Data Unit3	Data Unit4
528byte	528byte	528byte	528byte

- Data Unit1~4: During verify, each Data Unit area allows a maximum of 1 Bit Error. If there are 2 or more Bit Errors detected during the check, it will be considered as verify fail.

4. DataUnitSize：256 ，MaxErrorBit：1，In the programming file, each page's data is divided into eight equal parts, with the arrangement as 256+256+256+256+256+256+256+256, as shown in the table below:

Data Unit1	Data Unit2	Data Unit3	Data Unit4	Data Unit5	Data Unit6	Data Unit7	Data Unit8
256byte	256byte	256byte	256byte	256byte	256byte	256byte	256byte

- Data Unit1~8: During verify, each Data Unit area allows a maximum of 1 Bit Error. If there are 2 or more Bit Errors detected during the check, it will be considered as verify fail.

Consider the following points to choose the appropriate "Ecc_DataLayout" data layout mode (Mode 0, 1, 2, 3):

1 **The arrangement of data in the programming file**: Choose the appropriate data layout mode based on the arrangement of data in the programming file.

2 Does the programming file contain ECC data?

2.1 If the programming file does not contain ECC data, please select Mode3.

2.2 If the programming file contains ECC data, then choose the data layout mode according to the following conditions:

2.2.1 If the ECC data itself allows for Bit Errors, then please select Mode 3.

2.2.2 If the ECC data itself does not allow for Bit Errors, then please select Mode 0, 1, or 2 based on the arrangement of ECC data in the programming file.

Assuming the following conditions:

1. The ECC correction capability of the CPU is to correct 4 bit errors per 528 bytes.
2. The programming file contains ECC data.
3. The ECC data itself allows Bit Errors.

Based on the above conditions, Ecc_DataLayout is set to Mode 3, DataUnitSize is set to 528 bytes, and MaxErrorBit is set to 4 bits. If a stricter verification is desired during burning, MaxErrorBit can be set to 3 bits or less to filter out more stringent NAND Flash.

VIII. Revision History

Date	Version	Changes
2013/08/28	1.0	Initial Release
2014/04/30	1.1	Modify the layout
2014/07/18	2.0	Software descriptions
2016/10/07	2.1	Adding images and information for file loading.
2019/07/24	2.2	Content modification: 1. ***3.2 Special function only for NAND Flash: Addon*** 2. ***3.3 The actual procedure of the software*** 3. ***4.1.4.1 Functions for Programming Single Site*** The above functions are implemented in the version after DediWare3.12.83.15
2019/08/26	2.3	1. ***"4.1.2.1. Use Partition File"*** added Load Guarded Area Table description 2. ***"4.1.2.2. Custom"*** added Load Guarded Area Table / Skip Last Empty Block description in 3. ***"4.1.4.3 Read IC"*** added Read IC with Not Skip Bad Block / Read IC with Skip Bad Block description 4. Added ***Q12***~Q15 to the troubleshooting chapter
2021/04/20	2.4	1. ***"2.6 Bad Block Management"*** Added Hard Copy, Skip Bad Block and No management instructions
2022/02/25	2.5	1. Supplement 4.1.2 Description of EccAlgorithm 2. Added Q16 to the troubleshooting chapter
2023/02/05	2.6	1. Added the function description of "Auto calculate Guard Area and "Enable EccAlgorithm Asynchronization"
2024/09/09	2.7	Added "Supplementary Description of BBM Layout Function" chapter

DediProg Technology Co., Ltd. (Headquarter)
No. 142, Ankang Rd., Neihu Dist., Taipei City, Taiwan, R.O.C. 114044
TEL: 886-2-2790-7932 FAX: 886-2-2790-7916

Technical Support: **support@dediprog.com** Sales Support: **sales@dediprog.com**