Quick Start Guide

MRAM recovery using the pre-generated assets

The Ambiqsuite SDK's MRAM recovery tools come preconfigured to provision the EVB board to do recovery from the eMMC device or MSPI NOR Flash device depending on the specific EVB board. The \tools\apollo510_scripts\mram_recovery and its subdirectories have prebuilt binaries, scripts and configuration files for loading the recovery images into the specific NV FLASH devices. Also included is an example OEM recovery application that gets "recovered" to enable fully demonstrating the MRAM recovery process. Listed here are the subdirectories in \tools\apollo510_scripts\mram_recovery and what they contain:

\ambiq_recovery_images	-	Ambiq's released MRAM recovery images
\example_info0_configs	-	Example INFO0 configurations for MRAM recovery
\oem_recovery_bins	-	Example OEM Recovery App [1]
\recovery_loader	-	Example script (and app) for loading eMMC and MPI on EVBs [1]

[1] These directories contain files or sub directories specific to the EVB board being used.

Along with scripts to automate the loading of the recovery images into the Flash devices available on the EVB Boards, and generation of the INFO0 configurations for the Apollo510 or Apollo510B devices, the SDK also includes various example configurations for the MSPI and eMMC flash devices included on the EVB boards. To get started using the pre-generated assets there are 5 steps to enable non-magnetic testing of MRAM recovery, and 3 additional steps to enable complete magnetic induced MRAM recovery testing.

For more detailed information about the MRAM recovery process and the tools see ***Apollo510 MRAM Recovery Guide A-SOCAP5-UGGA01EN***

Note: This quick start guide provides the steps to evaluate MRAM recovery, which involves programming INFO0-OTP (in step 6) in preparation for causing MRAM corruption induced by a magnet. This guide provides the steps to test in DM-LCS without enabling secureboot. In a typical OEM development flow, when Secureboot and/or Secure-LCS is to be used in the final product, the OEM would coordinate and test with those additional settings for Info0 before finalizing Info0-OTP programming. This guide and the pre-generated MRAM recovery assets provided in the SDK only provide the necessary configuration(s) and steps to verify MRAM recovery operation.

This guide can be used with the Apollo510_EVB or Apollo510B_EVB hardware boards, along with Python version 3.8+, pylink-square and Segger Jlink tools. It is assumed these SW packages are installed and that 'python3' is the path to the python executable.

Note: Because there are BSP differences between the EVBs boards, there are different binaries for the **nvloader_app.bin** and the **oem_rcv_app.bin**. The turn-key configurations for each board are included, and where different, the correct command for each will be specified in each step.

Note: Ambiq's MRAM Recovery Info0 example configurations enable the Watchdog timer (**WD_EN**) option in the MRAM Recovery control word (**MRAM_RCVY_CTRL**) and any application that is loaded will need to turn off or service the watchdog timer.   Ambiq's supplied OEM Recovery application, used in this Quickstart guide, knows about the WD_EN being enabled, and handles it. The other example applications in the SDK do not service or disable the Watchdog timer, and will experience repeated resets when loaded with Info0 still having a MRAM Recovery configuration loaded with the WD_EN option enabled. When through working with MRAM recovery, the INFO0 configuration should be removed from Info0-MRAM and the active configuration set back to Info0-MRAM (if set to OTP).

Step 1: Power on the EVB and Start the SWO Viewer (speed 1000)

Step 2: Load the Recovery Images into a EVB Flash Device.

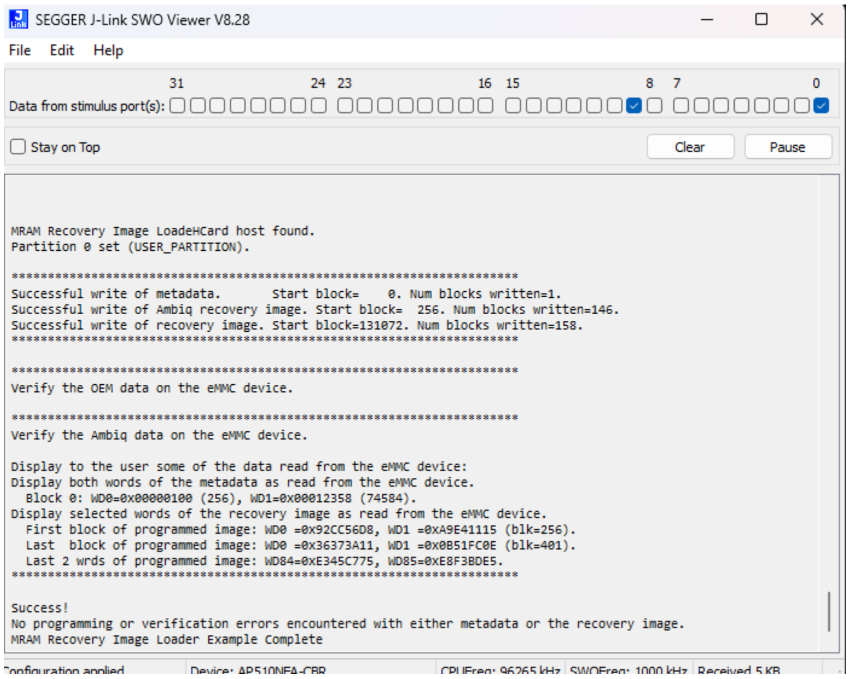
This demo will load and utilize the EMMC, and will also generate and load Info0-MRAM.

- Open a command window (in the ambiqsuite install directory):

  Apollo510_EVB:
  ```
  ➢ python3 recovery_nvloader.py --ldrCfg ..\recovery_nvloader_510.ini
  ```

  Apollo510B_EVB:
  ```
  ➢ python3 recovery_nvloader.py --ldrCfg ..\recovery_nvloader_510B.ini
  ```

  At this point the required recovery images have been loaded into the on-board eMMC Flash device and INFO0-MRAM has been configured for recovery from the eMMC device on SDIO0 and the output on the SWO view will look like:



```
MRAM Recovery Image LoadeHCard host found.
Partition 0 set (USER_PARTITION).

**********************************************************************
Successful write of metadata.       Start block=    0. Num blocks written=1.
Successful write of Ambiq recovery image. Start block=  256. Num blocks written=146.
Successful write of recovery image. Start block=131072. Num blocks written=158.
**********************************************************************

**********************************************************************
Verify the OEM data on the eMMC device.

**********************************************************************
Verify the Ambiq data on the eMMC device.

Display to the user some of the data read from the eMMC device:
Display both words of the metadata as read from the eMMC device.
  Block 0: WD0=0x00000100 (256), WD1=0x00012358 (74584).
Display selected words of the recovery image as read from the eMMC device.
  First block of programmed image: WD0 =0x92CC56D8, WD1 =0xA9E41115 (blk=256).
  Last  block of programmed image: WD0 =0x36373A11, WD1 =0x0B51FC0E (blk=401).
  Last 2 wrds of programmed image: WD84=0xE345C775, WD85=0xE8F3BDE5.
**********************************************************************

Success!
No programming or verification errors encountered with either metadata or the recovery image.
MRAM Recovery Image Loader Example Complete
```

Step 3: Load the OEM recovery application (example)

- Using the previous command window and moving up one directory (\tools\apollo510_scripts\mram_recovery):

  ```
  ➢ cd ..
  ```

- Start JLink Commander, connect to the device, and then):

  Apollo510_EVB:
  ```
  ➢ Loadbin   .\oem_recovery_bins\AP510_assets\oem_rcv_app.bin  0x410000
  ```

  Apollo510B_EVB:
  ```
  ➢ Loadbin   .\oem_recovery_bins\AP510B_assets\oem_rcv_app.bin  0x410000
  ```

- `Do a button reset (or a Jlink ‘r’, and ‘g’ command)`

  The application will output a banner "**Hello World! { MRAM Recovery Example }**" on the SWO follow by the output from the hello_world example. Additionally at the end there will be an MRAM recovery count output.

  ```
  MRAM RECOVERY INFO
  ==================
  Successful Ambiq MRAM Recovery Count = 1
  Successful OEM MRAM Recovery Count = 1
  ```

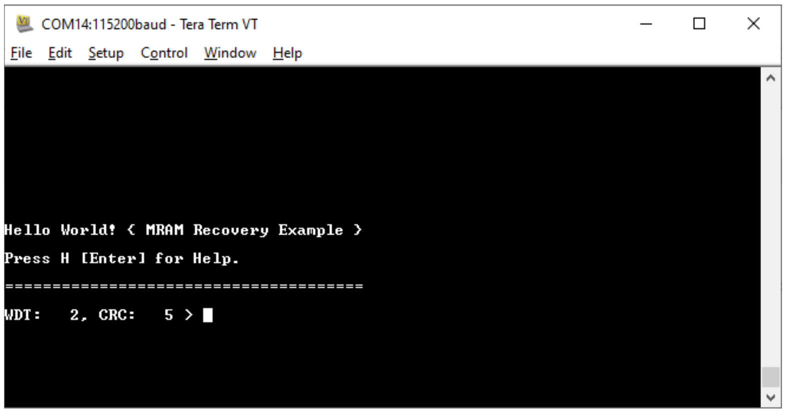
This is the same application that is in the oem_rcv_app_OTA.bin that was loaded into the eMMC flash device and represents a customer's application that is to be recovered.

Step 4: Open a terminal for UART communications to the Apollo5

- Start a Terminal Window (e.g. Puddy/Tera Term) and configure for the EVB's virtual com port from EVB. (Apollo5's UART0). (This will be named 'Jlink CDC UART Port (COMxx)')

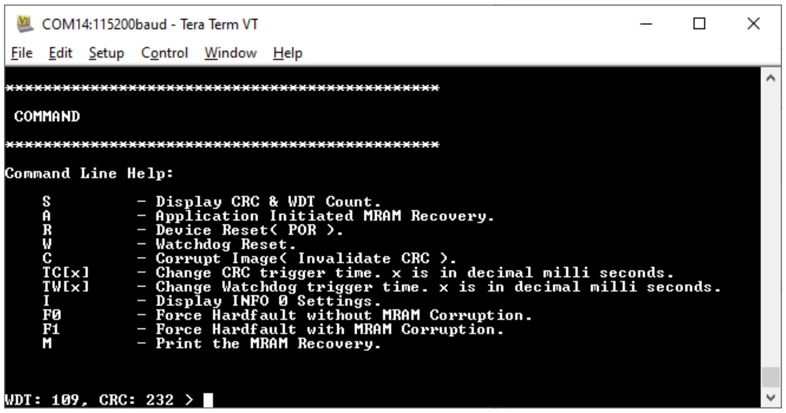
  Configure the serial port xx for 115200 baud, 8, 1, 1, N

  The oem_rcv_app example has a small command line processor via the UART to initiate various test conditions for MRAM recovery, the initial display should look like (A button reset might be required to see the complete output):


COM14:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
Hello World! { MRAM Recovery Example }
Press H [Enter] for Help.
=====================================
WDT:   2, CRC:   5 >
```

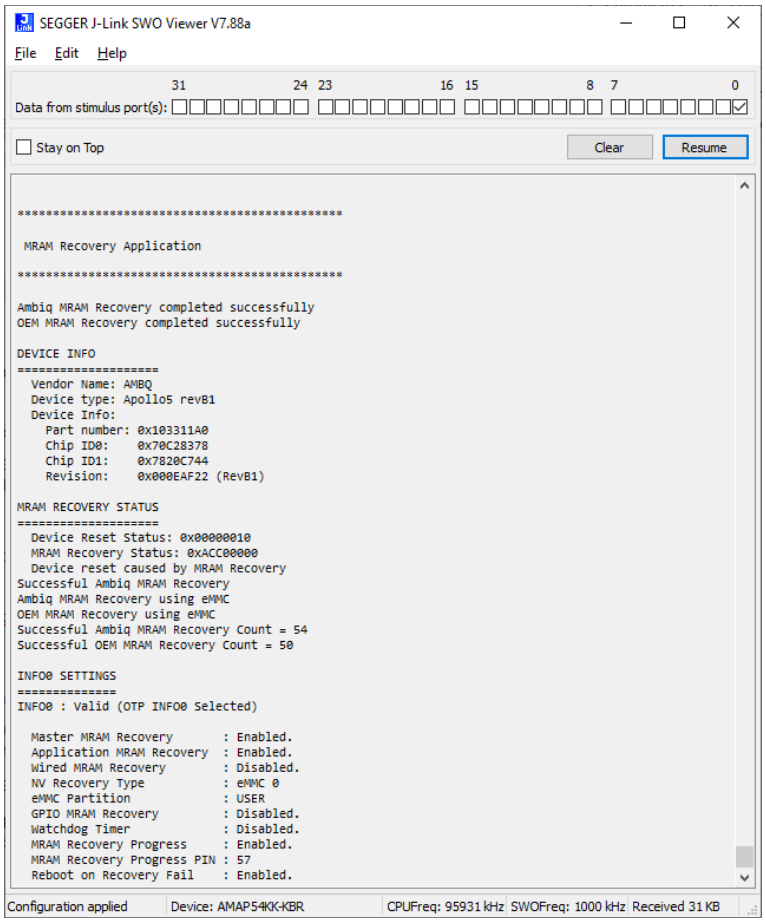
The command line has a help menu ("H" command) that has the following menu items:


COM14:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
***********************************************

 COMMAND

***********************************************

Command Line Help:

    S          - Display CRC & WDT Count.
    A          - Application Initiated MRAM Recovery.
    R          - Device Reset( POR ).
    W          - Watchdog Reset.
    C          - Corrupt Image( Invalidate CRC ).
    TC[x]      - Change CRC trigger time. x is in decimal milli seconds.
    TW[x]      - Change Watchdog trigger time. x is in decimal milli seconds.
    I          - Display INFO 0 Settings.
    F0         - Force Hardfault without MRAM Corruption.
    F1         - Force Hardfault with MRAM Corruption.
    M          - Print the MRAM Recovery.


WDT: 109, CRC: 232 >
```

Step 5: Initiating an application recovery:

At the command prompt on the terminal window press the "A" command (followed by the enter key) to start an "Application Initiated MRAM recovery". When the application recovery is complete a special MRAM recovery screen is shown for 10 seconds before it resets and the recovered "hello_world" screen appears again.



```
***********************************************

 MRAM Recovery Application

***********************************************

Ambiq MRAM Recovery completed successfully
OEM MRAM Recovery completed successfully

DEVICE INFO
====================
  Vendor Name: AMBQ
  Device type: Apollo5 revB1
  Device Info:
    Part number: 0x103311A0
    Chip ID0:    0x70C28378
    Chip ID1:    0x7820C744
    Revision:    0x000EAF22 (RevB1)

MRAM RECOVERY STATUS
====================
  Device Reset Status: 0x00000010
  MRAM Recovery Status: 0xACC00000
  Device reset caused by MRAM Recovery
Successful Ambiq MRAM Recovery
Ambiq MRAM Recovery using eMMC
OEM MRAM Recovery using eMMC
Successful Ambiq MRAM Recovery Count = 54
Successful OEM MRAM Recovery Count = 50

INFO0 SETTINGS
==============
INFO0 : Valid (OTP INFO0 Selected)

  Master MRAM Recovery        : Enabled.
  Application MRAM Recovery   : Enabled.
  Wired MRAM Recovery         : Disabled.
  NV Recovery Type            : eMMC 0
  eMMC Partition              : USER
  GPIO MRAM Recovery          : Disabled.
  Watchdog Timer              : Disabled.
  MRAM Recovery Progress      : Enabled.
  MRAM Recovery Progress PIN  : 57
  Reboot on Recovery Fail     : Enabled.
```

Note: The MRAM recovery screen is part of the oem_rcvy_app itself, but is only shown when the `rstgen->stat` register indicates MRAM recovery has occurred.

Preparing for Magnetic Testing

CAUTION: The previous steps must be completed successfully before programming INFO0-OTP (in step 6). Extra caution should be taken to verify that data to be programmed into INFO0-OTP matches the configuration that was verified to work when programmed into INFO0-MRAM. (and is consistent with the expected normal non-MRAM recovery operation) as once programmed OTP bits set to a '1' cannot be set back to '0'.

To prepare the device for magnetic testing, there are three additional steps that need to be performed.

- Program Info0-OTP
- Switch the INFO0 Configuration to OTP
- Perform a POI reset or Power Cycle the device.

Step 6: Program Info0-OTP

The same info0.bin file generated in step 1 is used to program INFO0-OTP.

- Open a command window (in the ambiqsuite install directory):

  ➢ `cd tools\apollo510_scripts`

  ➢ `jlink -CommanderScript jlink-prog-info0_otp.txt`

  Note: The programming for info0 MRAM and OTP is bit-for-bit identical (for the defined locations), even though Info0-MRAM is larger (2K) vs the Info0-OTP (256K bytes), the Ambiq supplied script create_info0.py outputs a info0.bin file 256 bytes in size (the size of INFO0-OTP) and that file is used to program both INFO0-MRAM and INFO0-OTP. The additional unused space in left for the OEM's use.

Step 7: Switch Info0 to use OTP

Connect with Jlink and execute to change INFO0 to be sourced from OTP:

➢ `w4 400C23FC 1`

Read back and verify that the register has an odd number of bits set (odd = OTP, even = MRAM)

➢ `mem32 400C23FC 1`

Step 8: Restart with a POI (or power cycle)

To have the new configuration take effect, reset the device with the Jlink command (that initiates a POI reset)

➢ `w4 40000004 1B`

At this point the device can be tested with magnetic interference to induce MRAM corruption/recovery

Creating a custom INFO0 Configuration

The OEM will need to make changes that are specific to their design for different devices, pin configurations and recovery options that are desired. The OEM would typically start with one of the info0.ini example files located in:

```
ambiqsuite\tools\apollo510_scripts\mram_recovery\example_info0_configs
```

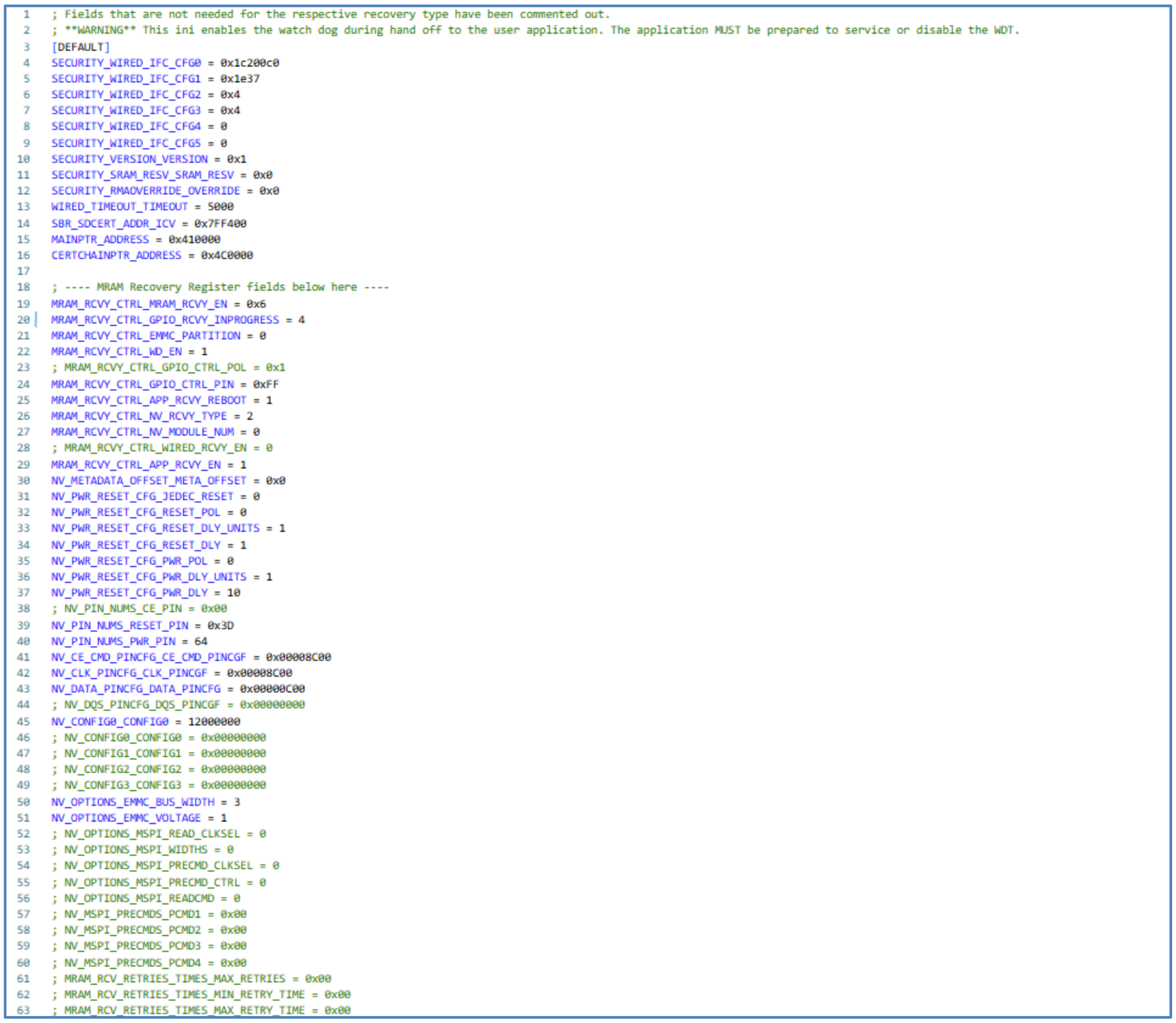
After creating a info0.ini file that matches their design requirements (e.g. oem_info0_options.ini) they will use this to create a Info0.bin file:

-
  ```
  cd  \tools\apollo510_scripts
  ```

-
  ```
  python3 create_info0.py --info0Cfg
    .\mram_recovery\example_info0_configs\oem_info0_options.ini  info0
  ```

This will create an info0.bin in `\tools\apollo510_scripts`

Below is a screenshot of the defaults in `info0_Recovery_EMMC.ini` for creating a info0.bin image with MRAM recovery fields configured for EMMC (SDIO0, User partition).



```
; Fields that are not needed for the respective recovery type have been commented out.
; **WARNING** This ini enables the watch dog during hand off to the user application. The application MUST be prepared to service or disable the WDT.
[DEFAULT]
SECURITY_WIRED_IFC_CFG0 = 0x1c200c0
SECURITY_WIRED_IFC_CFG1 = 0x1e37
SECURITY_WIRED_IFC_CFG2 = 0x4
SECURITY_WIRED_IFC_CFG3 = 0x4
SECURITY_WIRED_IFC_CFG4 = 0
SECURITY_WIRED_IFC_CFG5 = 0
SECURITY_VERSION_VERSION = 0x1
SECURITY_SRAM_RESV_SRAM_RESV = 0x0
SECURITY_RMAOVERRIDE_OVERRIDE = 0x0
WIRED_TIMEOUT_TIMEOUT = 5000
SBR_SDCERT_ADDR_ICV = 0x7FF400
MAINPTR_ADDRESS = 0x410000
CERTCHAINPTR_ADDRESS = 0x4C0000

; ---- MRAM Recovery Register fields below here ----
MRAM_RCVY_CTRL_MRAM_RCVY_EN = 0x6
MRAM_RCVY_CTRL_GPIO_RCVY_INPROGRESS = 4
MRAM_RCVY_CTRL_EMMC_PARTITION = 0
MRAM_RCVY_CTRL_WD_EN = 1
; MRAM_RCVY_CTRL_GPIO_CTRL_POL = 0x1
MRAM_RCVY_CTRL_GPIO_CTRL_PIN = 0xFF
MRAM_RCVY_CTRL_APP_RCVY_REBOOT = 1
MRAM_RCVY_CTRL_NV_RCVY_TYPE = 2
MRAM_RCVY_CTRL_NV_MODULE_NUM = 0
; MRAM_RCVY_CTRL_WIRED_RCVY_EN = 0
MRAM_RCVY_CTRL_APP_RCVY_EN = 1
NV_METADATA_OFFSET_META_OFFSET = 0x0
NV_PWR_RESET_CFG_JEDEC_RESET = 0
NV_PWR_RESET_CFG_RESET_POL = 0
NV_PWR_RESET_CFG_RESET_DLY_UNITS = 1
NV_PWR_RESET_CFG_RESET_DLY = 1
NV_PWR_RESET_CFG_PWR_POL = 0
NV_PWR_RESET_CFG_PWR_DLY_UNITS = 1
NV_PWR_RESET_CFG_PWR_DLY = 10
; NV_PIN_NUMS_CE_PIN = 0x00
NV_PIN_NUMS_RESET_PIN = 0x3D
NV_PIN_NUMS_PWR_PIN = 64
NV_CE_CMD_PINCFG_CE_CMD_PINCGF = 0x00008C00
NV_CLK_PINCFG_CLK_PINCGF = 0x00008C00
NV_DATA_PINCFG_DATA_PINCFG = 0x00000C00
; NV_DQS_PINCFG_DQS_PINCGF = 0x00000000
NV_CONFIG0_CONFIG0 = 12000000
; NV_CONFIG0_CONFIG0 = 0x00000000
; NV_CONFIG1_CONFIG1 = 0x00000000
; NV_CONFIG2_CONFIG2 = 0x00000000
; NV_CONFIG3_CONFIG3 = 0x00000000
NV_OPTIONS_EMMC_BUS_WIDTH = 3
NV_OPTIONS_EMMC_VOLTAGE = 1
; NV_OPTIONS_MSPI_READ_CLKSEL = 0
; NV_OPTIONS_MSPI_WIDTHS = 0
; NV_OPTIONS_MSPI_PRECMD_CLKSEL = 0
; NV_OPTIONS_MSPI_PRECMD_CTRL = 0
; NV_OPTIONS_MSPI_READCMD = 0
; NV_MSPI_PRECMDS_PCMD1 = 0x00
; NV_MSPI_PRECMDS_PCMD2 = 0x00
; NV_MSPI_PRECMDS_PCMD3 = 0x00
; NV_MSPI_PRECMDS_PCMD4 = 0x00
; MRAM_RCV_RETRIES_TIMES_MAX_RETRIES = 0x00
; MRAM_RCV_RETRIES_TIMES_MIN_RETRY_TIME = 0x00
; MRAM_RCV_RETRIES_TIMES_MAX_RETRY_TIME = 0x00
```

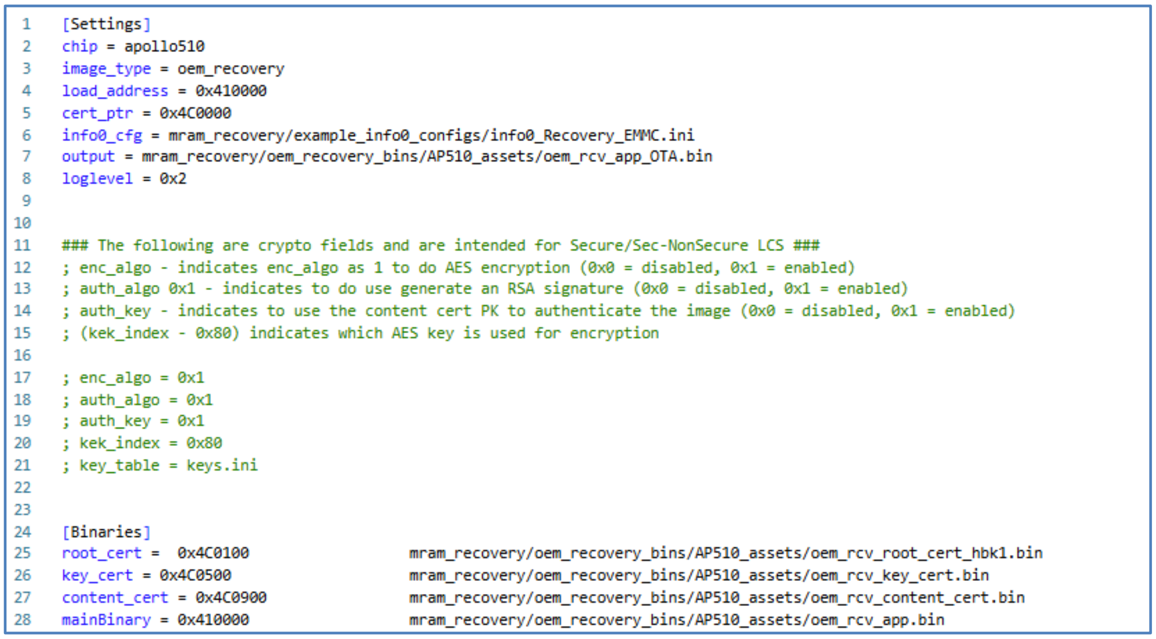
Creating an OEM Recovery OTA image

The OEM will create their own recovery application, and when the executable (.bin) file is ready, it is then built, along with the OEM certs into a secure image that is typically decrypted and authenticated with the OEMs provisioned keys during the recovery loading process. To build the OEM recovery image, a `oem_recovery.ini` located in the `\tools\apollo510_scripts\mram_recovery` is created to provide the file paths for the OEM's executable binary and security certificates, plus settings for the chosen authentication and encryption options.

- ➢ `cd  \tools\apollo510_scripts`
- ➢ `python3 create_cust_image_blob.py -c .\mram_recovery\oem_recovery.ini`

This creates an image at the location and name specified in the .ini file. (e.g. oem_rcv_app_OTA.bin in the example below)

Below is a screenshot of the current defaults in the oem_recovery.ini for creating a **Non-Secure** recovery OTA for an eMMC device.



```
[Settings]
chip = apollo510
image_type = oem_recovery
load_address = 0x410000
cert_ptr = 0x4C0000
info0_cfg = mram_recovery/example_info0_configs/info0_Recovery_EMMC.ini
output = mram_recovery/oem_recovery_bins/AP510_assets/oem_rcv_app_OTA.bin
loglevel = 0x2


### The following are crypto fields and are intended for Secure/Sec-NonSecure LCS ###
; enc_algo - indicates enc_algo as 1 to do AES encryption (0x0 = disabled, 0x1 = enabled)
; auth_algo 0x1 - indicates to do use generate an RSA signature (0x0 = disabled, 0x1 = enabled)
; auth_key - indicates to use the content cert PK to authenticate the image (0x0 = disabled, 0x1 = enabled)
; (kek_index - 0x80) indicates which AES key is used for encryption

; enc_algo = 0x1
; auth_algo = 0x1
; auth_key = 0x1
; kek_index = 0x80
; key_table = keys.ini


[Binaries]
root_cert =  0x4C0100            mram_recovery/oem_recovery_bins/AP510_assets/oem_rcv_root_cert_hbk1.bin
key_cert = 0x4C0500              mram_recovery/oem_recovery_bins/AP510_assets/oem_rcv_key_cert.bin
content_cert = 0x4C0900          mram_recovery/oem_recovery_bins/AP510_assets/oem_rcv_content_cert.bin
mainBinary = 0x410000            mram_recovery/oem_recovery_bins/AP510_assets/oem_rcv_app.bin
```

Note: MRAM recovery can be tested and verified in DM-LCS without the need to generate or include the OEM certificates. OEM certs are needed whenever the part is to be transition to Secureboot. The topic of Certificate generation and the OEM key provisioning and their use in secure loadable images is beyond the scope of this guide and is covered the detailed security documents.

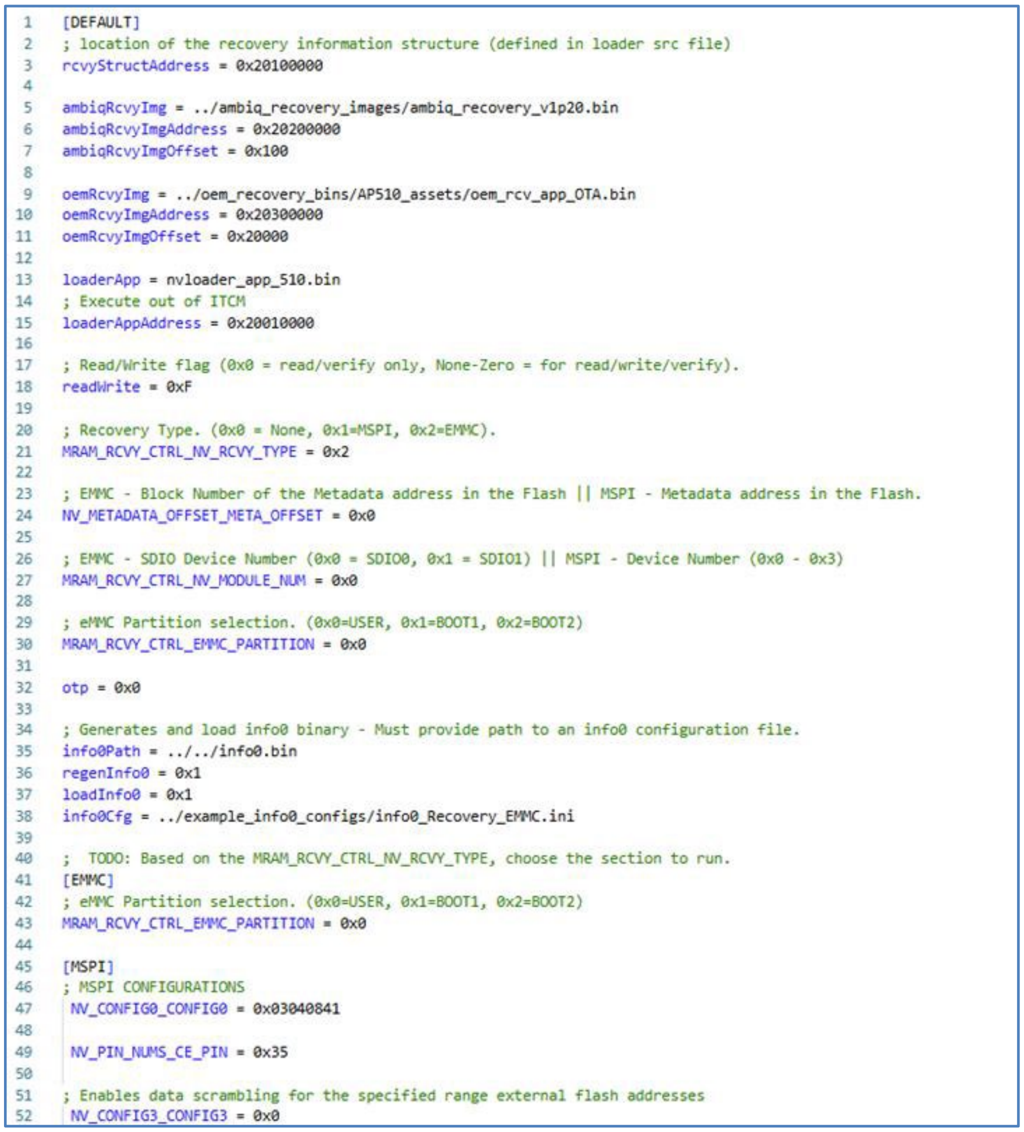
Changing the flash device and load addresses

The loading of the recovery images and metadata into the EVB's flash device(s) is done by the **recovery_nvloader.py** script which reads the filenames and paths for Ambiq's and the OEM's recovery images, along with location information for where in the Flash device that the images are to be located.

The **recovery_nvloader.py** script takes the .ini information and does error and consistency checking with the Info0 configuration (if provided) and then loads and passes the information to an Apollo5 application, nvloader_app.bin, which has device drivers for the various flash devices on the supported boards, which then loads the recovery images and reads back and verifies that they were programmed correctly.

The OEM will need to add specific read/write functions for their chosen flash device, but other aspects of the **recovery_nvloader.py** script the **nvloader_app_510.bin** should be easily adopted for their specific design with little or no change.

Below is a screenshot of the of the settings used for loading the recovery images into the eMMC user partition on SDIO0 the EVB.



```
[DEFAULT]
; location of the recovery information structure (defined in loader src file)
rcvyStructAddress = 0x20100000

ambiqRcvyImg = ../ambiq_recovery_images/ambiq_recovery_v1p20.bin
ambiqRcvyImgAddress = 0x20200000
ambiqRcvyImgOffset = 0x100

oemRcvyImg = ../oem_recovery_bins/AP510_assets/oem_rcv_app_OTA.bin
oemRcvyImgAddress = 0x20300000
oemRcvyImgOffset = 0x20000

loaderApp = nvloader_app_510.bin
; Execute out of ITCM
loaderAppAddress = 0x20010000

; Read/Write flag (0x0 = read/verify only, None-Zero = for read/write/verify).
readWrite = 0xF

; Recovery Type. (0x0 = None, 0x1=MSPI, 0x2=EMMC).
MRAM_RCVY_CTRL_NV_RCVY_TYPE = 0x2

; EMMC - Block Number of the Metadata address in the Flash || MSPI - Metadata address in the Flash.
NV_METADATA_OFFSET_META_OFFSET = 0x0

; EMMC - SDIO Device Number (0x0 = SDIO0, 0x1 = SDIO1) || MSPI - Device Number (0x0 - 0x3)
MRAM_RCVY_CTRL_NV_MODULE_NUM = 0x0

; eMMC Partition selection. (0x0=USER, 0x1=BOOT1, 0x2=BOOT2)
MRAM_RCVY_CTRL_EMMC_PARTITION = 0x0

otp = 0x0

; Generates and load info0 binary - Must provide path to an info0 configuration file.
info0Path = ../../info0.bin
regenInfo0 = 0x1
loadInfo0 = 0x1
info0Cfg = ../example_info0_configs/info0_Recovery_EMMC.ini

;  TODO: Based on the MRAM_RCVY_CTRL_NV_RCVY_TYPE, choose the section to run.
[EMMC]
; eMMC Partition selection. (0x0=USER, 0x1=BOOT1, 0x2=BOOT2)
MRAM_RCVY_CTRL_EMMC_PARTITION = 0x0

[MSPI]
; MSPI CONFIGURATIONS
 NV_CONFIG0_CONFIG0 = 0x03040841

 NV_PIN_NUMS_CE_PIN = 0x35

; Enables data scrambling for the specified range external flash addresses
 NV_CONFIG3_CONFIG3 = 0x0
```

The `recovery_nvloader.py` script and the nvloader_app_510.bin (or nvloader_app_510B.bin) are not part of the recovery process itself, and are only used to prepare the external Flash devices with the recovery images.

Notes: The `info0_cfg` field is used for validation purposes and is a relative path to the Info0 configuration file that will be in use during the recovery process to load the images. When provided, the `info0_cfg` field enables cross-referencing the `load_address, cert_ptr` and other fields that must be consistent with the Info0 configuration file when the creation of the OEM recovery OTA image is done to ensure correct operation during the recovery operation.

The `regenInfo0` field, when enabled, will run the create_info0.py script using the specified info0.ini in the `info0_cfg field`, thus ensuring that the Info0.bin that is loaded matches the .ini file that was used for the consistency checks.

The l`oadInfo0`, field, when enabled, will cause the nvloader to load the generated info0.bin onto the device at the same along with the recovery images into the flash device.

The source for the nvloader is provided in the examples directory of the SDK:

Apollo510_EVB:
```
  \boards\apollo510_evb\examples\mram_recovery\nvloader_app
```

Apollo510B_EVB:
```
  \boards\apollo510b_evb\examples\mram_recovery\nvloader_app
```

Notes about the example OEM Recovery App

The example OEM recovery application is a modification of the “hello_world” application example with special features added to detect and handle MRAM corruption, and facilitate the testing of the recovery configuration before committing to program Info0-OTP. The source for the OEM_recovery app is provided in:

```
\boards\apollo510_evb\examples\mram_recovery\mram_rcv_app
```

Special features that have been added to the standard hello_world application are:

- Active MRAM corruption detection via the WDT and CRC checking over MRAM memory range
- Relocated the vector table into RAM with the Fault, WDT ISRs and CRC checking functions run out of RAM
- A CRC over the binary is automatically added at build time, that the application uses to check itself at runtime.
- A UART command line to facilitate test conditions to help verify correct configuration and operation before real magnetic testing. Commands provide to cause simulated failure conditions for:
  - Watch dog timer time out (with no corruption)
  - Hardfaults with and without corruption
  - MRAM Corruption
  - Changing the time intervals between WDT and CRC checks.
- A “MRAM Recovery screen”, which is displayed when recovery has occurred with a timeout before resetting and running the Hello_world recovered app again.
  - The MRAM Recovery Screen can be displayed via a command line option.