MRAM Recovery via UART Host

Ambiqsuite SDK's MRAM recovery tools come with preconfigured example .ini files and scripts that enable MRAM recovery over a UART (or SPI) from a host processor. The SDK provides a python script (**uart_recovery_host.py**), that performs the host function from a PC connected to the Apollo510 EVB board via the USB virtual COM port from the Apollo510 device's UART0.

The example INFO0 configuration file (**info0_Recovery_UART.ini**) configures the EVB's Button1 as the GPIO_CTRL_PIN used to initiate MRAM recovery when the device is reset. With Button1 pressed as the device is reset (e.g. a button reset), the MRAM recovery process is initiated with the host PC via the UART.

The steps to perform MRAM recovery via a wired UART are:

- Enable Wired operations and specify the UART module number in INFOC
- Generate the "_wired.bin" version of the Ambiq and OEM recovery images
- Generate and load the INFO0 used to configure MRAM Recovery to use the UART interface
- Load the OEM recovery application (example)
- Run the **uart_recovery_host.py** script
- Initiate the MRAM Recovery by pressing Button1 as the EVB is reset by the reset button (AP5_RSTn)

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 7) in preparation for causing MRAM corruption induced by a magnet. This guide provides the steps to test while in DM-LCS without enabling secureboot. In a typical OEM development flow, when Secureboot and/or Secure-LCS are to be used in the final product, the OEM would coordinate and test with those additional settings for INFO0 before finalizing and programming INFO0 -OTP. 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+, pyserial 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 EVB boards, there are different binaries for the **oem_rcv_app.bin** used. The turn-key configurations for each board are included, and where different, the command for each will be specified separately 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: Enable Wired operations and verify the UART module number in INFOC

- Start JLink Commander, then enable UART0 for wired updates in the **INFOC_WIRED_CONFIG** register:

  ➢ `W4 0x400C2254 0x00000001`

  This enables the UART function (bit[0]) on UART module 0 (bits[17:16], which is the default already for the Apollo510x_EVBs.

Step 2: Generate the "_wired.bin" version of the recovery images:

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

  ➢ `cd \tools\apollo510_scripts`

  Then generate the Wired version of the Ambiq recovery image:

  ➢ `python3 create_cust_image_blob.py -c .\mram_recovery\ambiq_recovery_wired.ini`

- Then generate the Wired version of the oem_rcv_app image:

  Apollo510_EVB:
  ➢ `python3 create_cust_image_blob.py -c .\mram_recovery\oem_recovery_wired_apollo510.ini`

  Apollo510B_EVB:
  ➢ `python3 create_cust_image_blob.py -c .\mram_recovery\oem_recovery_wired_apollo510B.ini`

Step 3: Generate and load the INFO0 that configures MRAM Recovery options

The example INFO0 for Wired MRAM recovery will enable GPIO initiated MRAM recovery using Button1 on the EVB (GP94, pulled high when not pressed). To initiate MRAM recovery, press Button1 while resetting the EVB using the RSTn button.

- In the same command window as the previous step (\tools\apollo510_scripts) create the **info0.bin**:

  ➢ `python3 create_info0.py --info0Cfg .\mram_recovery\example_info0_configs\info0_Recovery_UART.ini info0`

- In the same command window, run the Jlink script to load the INFO0 (MRAM):

  ➢ `jlink -CommanderScript jlink-prog-info0.txt`

Step 4: Load the OEM recovery application (example application)

- In the same command window as the previous step (\tools\apollo510_scripts):

  Apollo510_EVB:
  - `loadbin .\mram_recovery\oem_recovery_bins\AP510_assets\oem_rcv_app.bin 0x410000`

  Apollo510B_EVB:
  - `loadbin .\mram_recovery\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 }**" over the SWO followed 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 is used as the OEM recovery image (**oem_rcv_app_OTA.bin)** that will be "recovered" and loaded via the UART during MRAM recovery process, which represents an example customer application.

  Note: The oem_rcvy_app example has a small command line processor that outputs to the UART that allows the user to initiate various test conditions for MRAM recovery, but is not typically used when doing wired recovery since the UART is used for loading the recovery images. Additional details about it can be found in the MRAM recovery (for NV flash devices) Quickstart guide. When using the UART for MRAM recovery, the serial terminal window, if open, must be closed before starting the **uart_recovery_host.py** script or the script will not be able to open the serial port.

Step 5: Run the uart_recovery_host.py script:

- In the same command window ((\tools\apollo510_scripts):

  Apollo510_EVB:
  - `python3 uart_recovery_host.py`
    `-fa .\ambiq_recovery_images\ambiq_recovery_v1p20_a_wired.bin`
    `-fo .\oem_recovery_bins\`**`AP510_assets`**`\oem_rcv_app_wired.bin <COMx>`

  Apollo510B_EVB:
  - `python3 uart_recovery_host.py`
    `-fa .\ambiq_recovery_images\ambiq_recovery_v1p20_a_wired.bin`
    `-fo .\oem_recovery_bins\`**`AP510B_assets`**`\oem_rcv_app_wired.bin <COMx>`

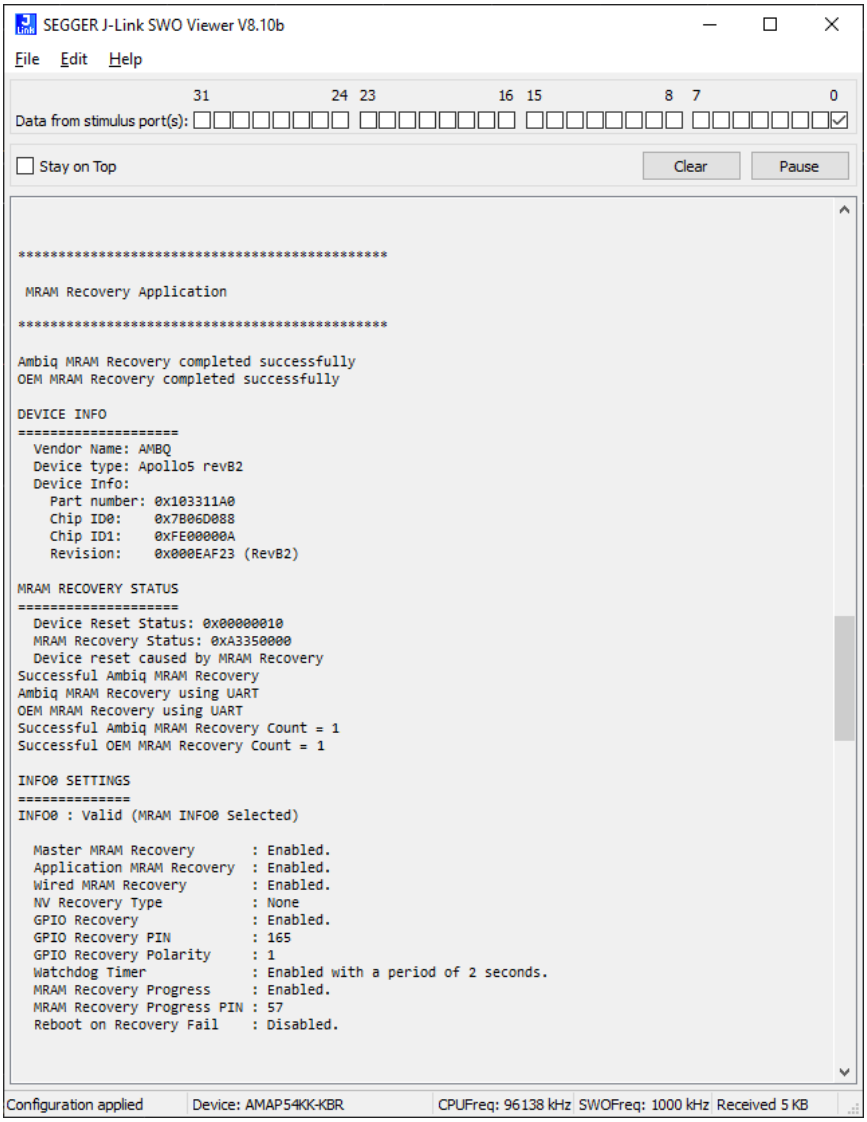
  The < COMx> specifies the EVB's virtual com port from the Apollo device (UART0). (it be named 'Jlink CDC UART Port (COMx)')

Step 6: Initiate the MRAM Recovery.

- Press and hold the Button1 (SW3) while pressing the RSTn (AP5_RSTn) on the EVB board

This will reset the device and initiate MRAM recovery which and the uart_recovery_host will detect it (by receiving a response to the hello message) and with then download the requested recovery image (Ambiq or OEM)

When both the Ambiq and OEM recovery is complete, the OEM's recovery application will run and a special MRAM recovery screen is displayed for 10 seconds before it resets the device and the recovered "hello_world" screen appears again.

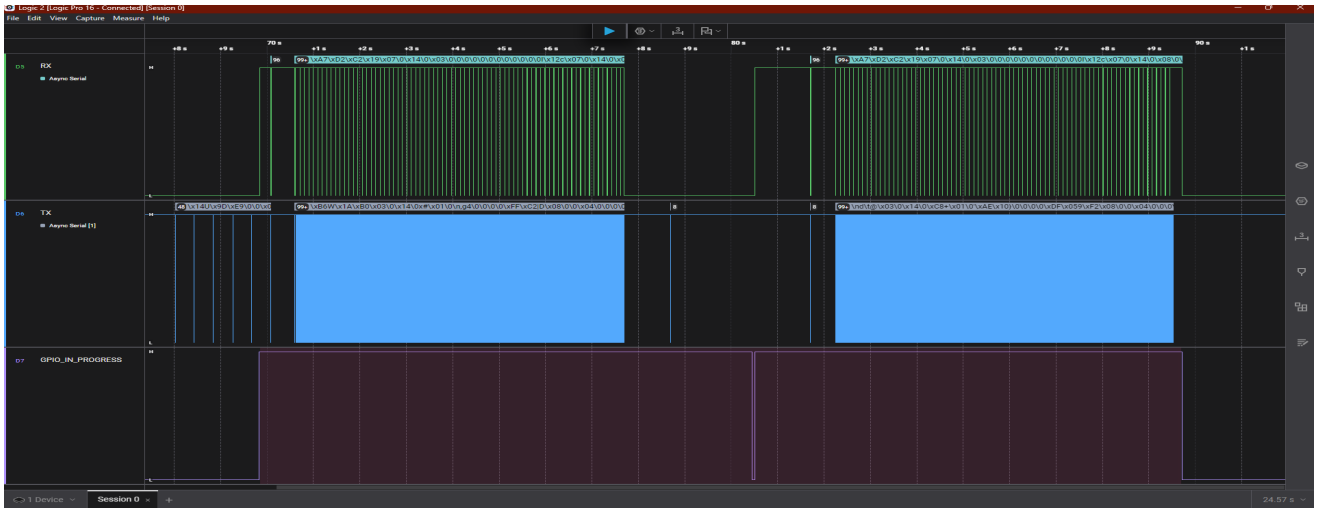


Both the SBL's and the example **oem_rcv_app's** SWO print out the count of successful recoveries, which will increase up to a maximum of 64. The OEM recovery count is part of the **oem_rcv_app** which defines its location in the customer area of INFO0-OTP as an example of how this could be done. The Ambiq recovery count is defined and maintained within INFO1-OTP by Ambiq's MRAM recovery process.

Note: The MRAM recovery screen is part of the **oem_rcvy_app** itself, and it's only shown at startup when the

rstgen->stat register indicates MRAM recovery has occurred.

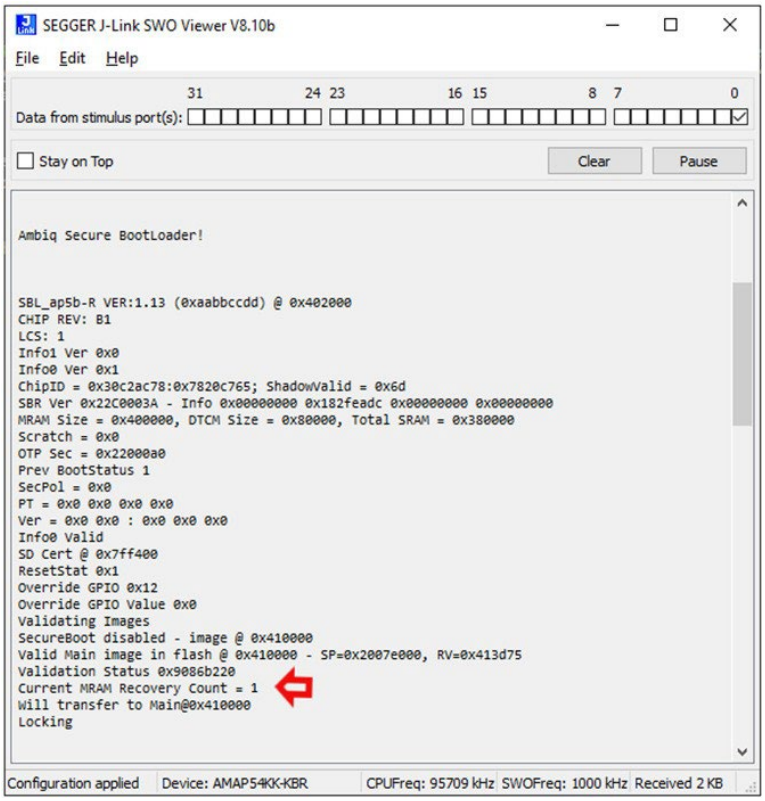
Note: When using the onboard Jlink, the SWO Viewer may lose connection during the MRAM recovery process so this recovery screen may not be displayed and restarting the SWO view may be required. The external Jlink probes do not have this issue and continue to work throughout the MRAM recovery process.



Above is a logic analyzer capture of the MRAM recovery process over UART, the first section is the loading of the Ambiq Recovery Image, and the second is the loading OEM Recovery Image. Each is initiated by the responding to the host's (uart_recovery_host.py) hello msg, that is being sent looking for a response msg indicating MRAM recovery is in progress.

Then verify that the MRAM recovery count has successfully incremented:

Note: EVBs do not allow for passing of the RTS/CTS over the USB's virtual serial port (used to signal MRAM Recovery is in progress) so the **user_recovry_host.py** polls by sending repeated HELLO messages until it receives the response indicating MRAM recovery is in progress. This is not how a customer would typically implement the host and does not allow for optimal timing. A more typical implementation is discussed in the section "Using UART1 for optimal host timing" latter in this document.



Preparing for Magnetic Testing

CAUTION: The previous steps must be completed successfully before programming INFO0-OTP (in step 7). 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 four additional steps to be performed.

- Program INFO0-OTP
- Switch the INFO0 Configuration to OTP
- Reset the Apollo510 with GPIO 165 high (indicating a MRAM recovery request).

Step 7: Program INFO0-OTP

The same info0.bin file generated in step 1 is used to program INFO0-OTP. In the \apollo510_scripts directory (`ambiqsuite\tools\apollo510_scripts`) execute the Jlink commander script:

- `cd ambiqsuite\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 8: Switch Info0 to use OTP

Connect with Jlink and execute the command (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`

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

- `w4 40000004 1B`

Step 9: Verify MRAM recovery again with INFO0-OTP (initiate MRAM recovery)

Start the uart_recovery_host (step 5) and initiate MRAM recovery by setting GPIO 165 high (step 6) and pressing the button reset on the EVB board (SW3), and verify that recovery completes successfully

At this point the device can be tested with magnetic interference to induce MRAM 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 the editing or creating an updated info0.ini file that matches their design requirements (e.g. oem_info0_options.ini) they will create a Info0.bin file:

- `cd \tools\apollo510_scripts`

- `python3 create_info0.py --info0Cfg .\mram_recovery\example_info0_configs\ info0_recovery_UART_SPI.ini`

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

Below is a screenshot of the defaults in `info0_recovery_UART_SPI.ini` for creating a info0.bin image with MRAM recovery fields configured for UART0 at 115200 baud, with the recovery in-progress indication in GPIO 04 and MRAM recovery initiated by GPIO 94 (button1 on the Apollo510 EVB) .

```
; 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 = 0x4
; MRAM_RCVY_CTRL_EMMC_PARTITION = 0
MRAM_RCVY_CTRL_WD_EN = 1
MRAM_RCVY_CTRL_GPIO_CTRL_POL = 0
MRAM_RCVY_CTRL_GPIO_CTRL_PIN = 94
; MRAM_RCVY_CTRL_APP_RCVY_REBOOT = 0
; MRAM_RCVY_CTRL_NV_RCVY_TYPE = 0
; MRAM_RCVY_CTRL_NV_MODULE_NUM = 0
MRAM_RCVY_CTRL_WIRED_RCVY_EN = 1
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 = 0
; NV_PWR_RESET_CFG_RESET_DLY = 0
; NV_PWR_RESET_CFG_PWR_POL = 0
; NV_PWR_RESET_CFG_PWR_DLY_UNITS = 0
; NV_PWR_RESET_CFG_PWR_DLY = 0
; NV_PIN_NUMS_CE_PIN = 0x00
; NV_PIN_NUMS_RESET_PIN = 0x00
; NV_PIN_NUMS_PWR_PIN = 0x00
; NV_CE_CMD_PINCFG_CE_CMD_PINCGF = 0x00000000
; NV_CLK_PINCFG_CLK_PINCGF = 0x00000000
; NV_DATA_PINCFG_DATA_PINCFG = 0x00000000
; NV_DQS_PINCFG_DQS_PINCGF = 0x00000000
; NV_CONFIG0_CONFIG0 = 0x00000000
; NV_CONFIG1_CONFIG1 = 0x00000000
; NV_CONFIG2_CONFIG2 = 0x00000000
; NV_CONFIG3_CONFIG3 = 0x00000000
; NV_OPTIONS_EMMC_BUS_WIDTH = 0
; NV_OPTIONS_EMMC_VOLTAGE = 0
; 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
```

Only the fields in blue are necessary for UART configuration, all other fields (in green) are unused for this configuration.

Creating a Wired OEM MRAM Recovery 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 OEM Recovery image that is typically decrypted and authenticated with the OEMs provisioned keys during the recovery loading process. Once the OEM Recovery image has been created, a second step adds a wired header to the recovery image allowing it to be used with the wired protocol. Both steps use the **create_cust_image_blob.py** script. All the inputs and output pathnames are specified in the .ini file and can be any reachable storage locations.

- **Step 1:** (create an OEM MRAM recovery OTA image)

To build the OEM recovery image the `oem_recovery.ini` located in the `\tools\apollo510_scripts\mram_recovery` is modified to provide the file pathnames for the OEM's executable binary and security certificates (if used), plus settings for the chosen authentication and encryption options.

➢ cd `\tools\apollo510_scripts`

Apollo510_EVB:

➢ `python3 create_cust_image_blob.py -c .\mram_recovery\oem_recovery_apollo510.ini`

Apollo510B_EVB:

➢ `python3 create_cust_image_blob.py -c .\mram_recovery\oem_recovery_apollo510B.ini`

This creates an OEM recover 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. When configuring for MRAM Recovery over UART or SPI, Line 7 should be changed to specify the **info0_Recovery_UART_SPI.ini** file.

```
1  [Settings]
2  chip = apollo510
3  image_type = oem_recovery
4  load_address = 0x410000
5  cert_ptr = 0x4C0000
6  info0_cfg = mram_recovery/example_info0_configs/info0_Recovery_EMMC.ini
7  output = mram_recovery/oem_recovery_bins/AP510_assets/oem_rcv_app_OTA.bin
8  loglevel = 0x2
9
10
11 ### The following are crypto fields and are intended for Secure/Sec-NonSecure LCS ###
12 ; enc_algo - indicates enc_algo as 1 to do AES encryption (0x0 = disabled, 0x1 = enabled)
13 ; auth_algo 0x1 - indicates to do use generate an RSA signature (0x0 = disabled, 0x1 = enabled)
14 ; auth_key - indicates to use the content cert PK to authenticate the image (0x0 = disabled, 0x1 = enabled)
15 ; (kek_index - 0x80) indicates which AES key is used for encryption
16
17 ; enc_algo = 0x1
18 ; auth_algo = 0x1
19 ; auth_key = 0x1
20 ; kek_index = 0x80
21 ; key_table = keys.ini
22
23
24 [Binaries]
25 root_cert =  0x4C0100              mram_recovery/oem_recovery_bins/AP510_assets/oem_rcv_root_cert_hbk1.bin
26 key_cert = 0x4C0500                mram_recovery/oem_recovery_bins/AP510_assets/oem_rcv_key_cert.bin
27 content_cert = 0x4C0900             mram_recovery/oem_recovery_bins/AP510_assets/oem_rcv_content_cert.bin
28 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 undergo transition to Secureboot or Secure-LCS. 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.

- **Step 2:** (add wired header to the OEM MRAM recovery image)

The second step adds the wired protocol header to the recovery image allowing it to be used with the UART or SPI wired interfaces for MRAM recovery. The **create_cust_image_blob.py** script is run from the same directory as before (\tools\apollo510_scripts):

Apollo510_EVB:

```
➢ python3 create_cust_image_blob.py -c .\mram_recovery\oem_recovery_wired_apollo510.ini
```

Apollo510B_EVB:

```
➢ python3 create_cust_image_blob.py -c .\mram_recovery\oem_recovery_wired_apollo510B.ini
```

Below is a screenshot of the defaults for creating a “wired” OEM recovery image. Note the only difference between the Apollo510 and Apollo510B versions is the input and output pathnames. (because the oem_rcv_app.bin’s are different because each there is a different BSP for each board).

```
#*******************************************************************************
#
# Configuration file for create_cust_image_blob.py
#
# Run "create_cust_image_blob.py --help" for more information about the options
# below.
#
# All numerical values below may be expressed in either decimal or hexadecimal
# "0x" notation.
#
# To re-generate this file using all default values, run
# "create_cust_image_blobl.py --create-config"
#
#*******************************************************************************
[Settings]
chip = apollo510
app_file = ./mram_recovery/oem_recovery_bins/AP510_assets/oem_rcv_app_OTA.bin
# Temp location in MRAM where the OTA blob should be loaded
load_address = 0x7d0000
kek_index = 0x80
image_type = wired
output = ./mram_recovery/oem_recovery_bins/AP510_assets/oem_rcv_app_wired.bin
key_table = keys.ini
loglevel = 0x2

wired_chunk_size = 0x40000
```

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 across full MRAM memory range
- Relocated the vector table into RAM with the Faults, WDT ISRs and CRC checking functions run out of RAM
- A CRC over the OEM recovery app 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 is done. Commands are available 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.

Using UART1 for optimal host Timing

By default the **uart_recovery_host.py** script, polls to determine when MRAM recovery is in progress by repeatedly sending the HELLO message and looking for a STATUS response message that indicates MRAM recovery is in progress (either Ambiq or OEM), and then sends the requested recovery image. This does not represent how a typical host would be designed. A typical embedded host would setup and monitor the MRAM Recovery "in_progress" GPIO pin and use that to trigger the sending the HELLO message. In many USB Virtual UARTs the "in_progress" GPIO can be sent to the PC via the UART's RTS->CTS signal. Unfortunately, the on-board Jlink devices used on the Ambiq EVBs do not support this type of RTS/CTS signaling, so the default operation of the **uart_recovery_host.py** script is to poll to determine when MRAM recovery has started.

If precise timing and operation of MRAM Recovery is desired then an alternative method is to use an external USB Virtual COM port connected to UART1, with its RTS input connected to the GPIO pin that's configured as the MRAM recovery "in_progress" indicator pin. All FTDI based USB Virtual COM ports support the necessary RTS/CTS independent operation. When selecting a USB Virtual COM adapter, it should also support 1.8v I/O operation as well, such as **the DSD TECH SH-U09C2 USB to TTL Adapter**.

The **uart_recovery_host.py** script supports the RTS/CTS signal being used to indicated MRAM Recovery "in_progress", when used with the **-inprogCTS** command line option. When "-inprogCTS" is specified it monitors the CTS signal (at the PC) to indicate when MRAM Recovery has started, instead of polling for it. The "in_progress" GPIO pin is connected to the RTS input of the USB's virtual com interface. (the RTS signal on the Apollo510 side is reflected in the CTS signal status at the COM port in the PC).

The example Info0 .ini file **info0_Recovery_UART1.ini**. provides an example configured to monitor the "in_progress" GPIO in the file: **info0_Recovery_UART1.ini**. This has the configuration used with an external USB->COM adapter wired as shown:

EVB Pins	Signal	USB->COM port adaptor
0	TX	TX (input)
2	RX	RX (ouput)
4	In_progress	RTS (input)
SYS_VDD	1.8V	VCC (Power for adaptor)
GND	GND	GND

Note: That the configured UART in the **UARTMODULE** field of **INFOC_WIRED_CONFIG** field also needs to be changed to '1' (from the default 0) to use the UART1 port 1, which is a permanent change, and will affect all wired updates and MRAM recovery operations from that that point on, and will utilize UART1. (the pins can be changed via the **SECURITY_WIRED_IFC_CFG1** fields in INFO0, but the UART module change is in made in INFOC which is OTP and cannot be changed back.

Below is a captured trace utilizing UART1 and using **-inprogCTS** option to use the CTS status to monitor the in_progress GPIO pin.