Apollo3-Blue MCU Family Getting Started Guide

Revision 3.0
Jan 2020

Revision History

Date	Revision	History	Reviser
Feb 12, 2018	0.1	Initial Version	D. Munsinger
Feb 21, 2018	1.0	Minor type edit in create_info0 cmd	D. Munsinger
Mar 19, 2018	1.1	Update create_info0 cmd Added section for OTA	J. Shah
Mar 22, 2018	1.2	Repurposed the document as “Getting Started Guide”. Created separate Guide for Scripts Added JLink Script for INFO0 programming	J. Shah
	2.0	Updates for SBLv1	J. Shah
July 05, 2018	2.1	Updates for SBLv2	J. Shah
Jan 29, 2019	2.2	Updates for SBLv3	D. Munsinger
Sep 24, 2019	2.3	Updates for Apollo3-Blue-Plus	D. Munsinger
Jan 20, 2020	3.0	Updates for SDK Release 2.4.1	J. Shah

Contents

1. Introduction

Vanilla Apollo3-Blue & Apollo3-Blue-Plus parts from Ambiq factory are pre-programmed with a Secure Boot Loader, and an uninitialized INFO0.

In general, initial provisioning of the part would include programming a valid INFO0, and programming the main firmware image in the flash.

Ambiq Apollo3-Blue SDK contains a number of python scripts to demonstrate generation of Customer InfoSpace (INFO0) settings, Customer Main images, and creation of images for the Wired Update protocol over UART.

This document will explain their usage. Unless specifically noted, the content applies to all products in Apollo3-Blue family of MCUs, even though it may refer to Apollo3-Blue only.

Part of this demonstration is to upgrade the JLink environment to ensure the debugging tools continue to work with the Apollo3-Blue.

Disclaimer: This document shows the detailed Debug JLink SWO output from the Secure Bootloader. This output will disappear in a later release and is informational only.

2. References

REF	Title	File
REF1	Apollo3-Blue Secure Update Flow	Apollo3-Blue_Secure_Update_Flow.pdf
REF2	AMOTA Example User's Guide	AMOTA_example_user's_guide.pdf
REF3	Apollo3-Blue Security Whitepaper	
REF4	Apollo3-Blue Secure Bootloader Scripts User's Guide	Apollo3-Blue_SBL_Scripts.pdf

3. Setting up the environment

The Apollo3 EVB comes with INFO0 initialized and the default binary_counter example programmed into main (at 0xC000). The python scripts and supporting binary images for these examples can be found in **/tools/apollo3_scripts/**

3.1 Preparation of the Python Environment

This document assumes that the user has a python3 environment available. The SBL scripts require the addition of the python crypto modules. Those can be obtained as follows:

```
pip install pycryptodome
pip install pyserial
```

3.2 Setting up Host connection

3.2.1 USB-UART Host Connection

The windows PC will be connected via a USB-UART adapter to EVB.

Apollo3Blue EVB & Apollo3BluePlus EVB pins:

- UART-RX pin 23
- UART-TX pin 22

3.2.2 IOS-SPI Host Connection

The following table shows the required Host to Slave connections for IOS-SPI operation. This is only supported on SBL-v3 onwards for Apollo3. All Apollo3P SBL versions support IOS.

HOST (Apollo3* EVB)	SLAVE (Apollo3* EVB + SBL)	Signal
GPIO(2)	GPIO(4)	Slave to Host Interrupt
GPIO(4)	GPIO(16)	Override – Force SBL to scan for updates on wired interface(s)
GPIO(5)	GPIO(0)	SCLK
GPIO(6)	GPIO(2)	MISO
GPIO(7)	GPIO(1)	MOSI
GPIO(11)	GPIO(3)	CS
GPIO(17)	nRST	Reset
GND	GND	Ground

Note: In this configuration the HOST Apollo3Blue EVB and Apollo3BluePlus may be programmed with the “uart_boot_host” example and the USB-UART connection can be used with the provide python script “uart_wired_update.py” in the same way as the UART examples below. The uart_boot_host example should be configured as follows:

```
57  //
58  // Configure to output as SPI or I2C.
59  //
60  #define USE_SPI                         1   // 0 = I2C, 1 = SPI
61
```

3.2.3 IOS-I2C Host Connection

The following table shows the required Host to Slave connections for IOS-I2C operation.

HOST (Apollo3 EVB)	SLAVE (Apollo3 EVB + SBL)	Signal
GPIO(2)	GPIO(4)	Slave to Host Interrupt
GPIO(4)	GPIO(16)	Override – Force SBL to scan for updates on wired interface(s)
GPIO(5)	GPIO(0)	I2C SCL
GPIO(6)	GPIO(1)	I2C SDA
GPIO(17)	nRST	Reset
GND	GND	Ground

Note: In this configuration the HOST Apollo3 EVB may be programmed with the “uart_boot_host” example and the USB-UART connection can be used with the provide python script “uart_wired_update.py” in the same way as the UART examples below. The uart_boot_host example should be configured as follows:

```
57  //
58  // Configure to output as SPI or I2C.
59  //
60  #define USE_SPI                         0   // 0 = I2C, 1 = SPI
61
```

3.3 Keys

File keys_info.py needs to be created containing customers’ secret keys.

For quick start, a template for this file is included in SDK.

Copy keys_info0.py to keys_info.py

```
cp keys_info0.py keys_info.py
```

3.4 Installing the JLink scripts for Apollo3-Blue

The Apollo3-Blue MCU is natively supported by the SEGGER JLink tools as of V6.34 by simply selecting AMA3B1KK-KBR as the target device. The Apollo3-Blue-Plus MCU is natively supported by the SEGGER JLink tools as of V6.62 by simply selecting AMA3B2KK-KBR as the target device. Ambiq recommends using the latest SEGGER JLink tools and uninstalling previous versions prior to upgrading. Support for earlier versions of the SEGGER tools is no longer supported in the SDK.

4. Programming Customer InfoSpace (INFO0)

Ambiq recommends programming for the single desired interface (UART, SPI, or I2C) as shown the following script examples. It is also possible to enable UART+SPI or UART+I2C, but having both SPI and I2C enabled at the same time will not work.

4.1 Generate INFO0 for UART Operation

Initially it is best to flash a valid INFO0 with GPIO override provision (before you play with main image).

- For Apollo3Blue, create INFO0 image with GPIO Override is set to pin 16 (0x10) active low. Baudrate for INFO0 UART is set to 115200 (0x1C200). Main image is expected at 0xC000. Apollo3Blue is configured for UART-RX pin 23 (0x17) & UART-TX pin 22 (0x16). The chip type is specified as apollo3.

```
./create_info0.py --valid 1 info0 --pl 1 --u0 0x1C200c0 --u1 0xFFFF1617
--u2 0x2 --u3 0x0 --u4 0x0 --u5 0x0 --main 0xC000 --gpio 0x10 --version
0 --wTO 5000 --chipType apollo3
```

- For Apollo3BluePlus, create INFO0 image with GPIO Override is set to pin 16 (0x10) active low. Baudrate for INFO0 UART is set to 115200 (0x1C200). Main image is expected at 0xC000. Apollo3 is configured for UART-RX pin 23 (0x17) & UART-TX pin 22 (0x16). The chip type is specified as apollo3p

```
./create_info0.py --valid 1 info0 --pl 1 --u0 0x1C200c0 --u1 0xFFFF1617
--u2 0x2 --u3 0x0 --u4 0x0 --u5 0x0 --main 0xC000 --gpio 0x10 --version
0 --wTO 5000 --chipType apollo3p
```

4.2 Generate INFO0 for IOS-SPI Operation

- For Apollo3Blue, create INFO0 image with GPIO Override is set to pin 16 (0x10) active low. Main image is expected at 0xC000. Only the SPI interface is enabled (0x2). The Slave interrupt is set to Pin 4 (0x4).

```
./create_info0.py --valid 1 info0 --pl 1 --main 0xC000 --gpio 0x10 --
version 0 --wmask 0x2 --wSlInt 0x4 --chipType apollo3
```

- For Apollo3BluePlus, create INFO0 image with GPIO Override is set to pin 16 (0x10) active low. Main image is expected at 0xC000. Only the SPI interface is enabled (0x2). The Slave interrupt is set to Pin 4 (0x4).

```
./create_info0.py --valid 1 info0 --pl 1 --main 0xC000 --gpio 0x10 --
version 0 --wmask 0x2 --wSlInt 0x4 --chipType apollo3p
```

4.3 Generate INFO0 for IOS-I2C Operation

- For Apollo3Blue, create INFO0 image with GPIO Override is set to pin 16 (0x10) active low. Main image is expected at 0xC000. Only the I2C interface is enabled (0x4). The Slave interrupt is set to Pin 4 (0x4).

```
./create_info0.py --valid 1 info0 --pl 1 --main 0xC000 --gpio 0x10 --
version 0 --wmask 0x4 --wSlInt 0x4 --chipType apollo3
```

- For Apollo3BluePlus, create INFO0 image with GPIO Override is set to pin 16 (0x10) active low. Main image is expected at 0xC000. Only the I2C interface is enabled (0x4). The Slave interrupt is set to Pin 4 (0x4).

```
./create_info0.py --valid 1 info0 --pl 1 --main 0xC000 --gpio 0x10 --
version 0 --wmask 0x4 --wSlInt 0x4 --chipType apollo3p
```

4.4 Program INFO0

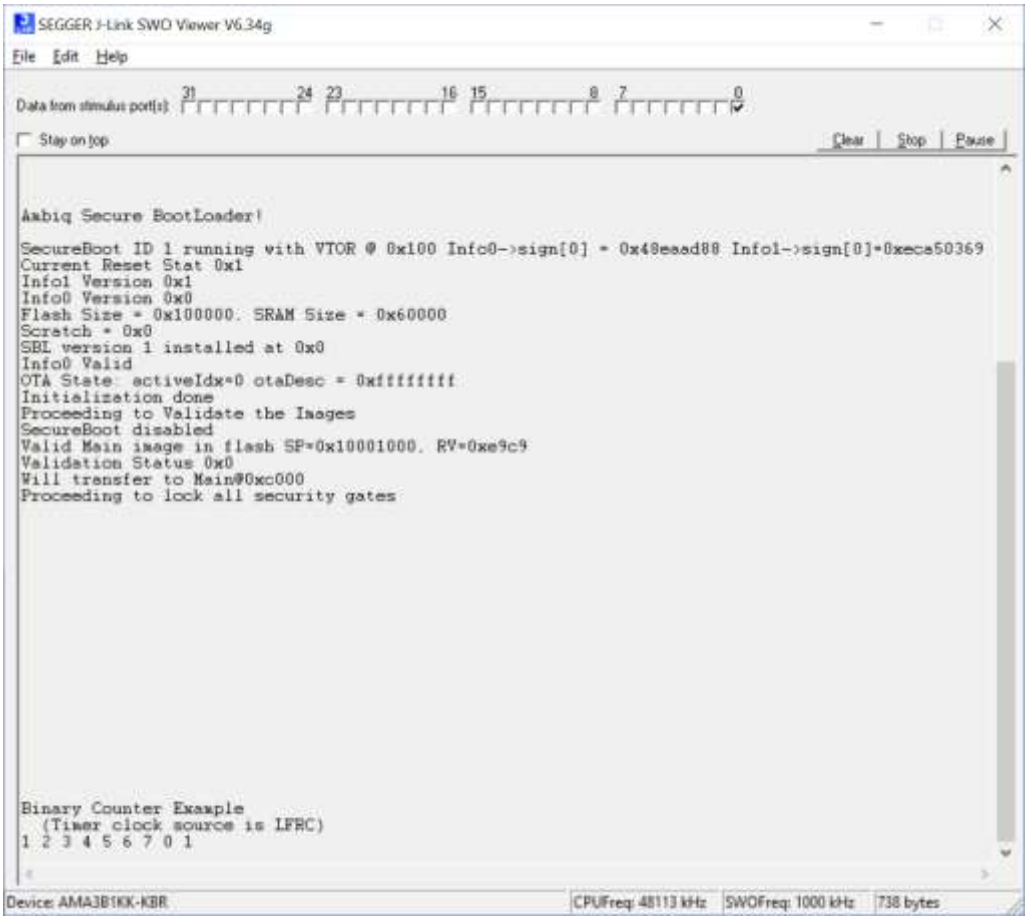
INFO0 can be programmed either using a JLink script, or through the SBL assisted Wired Update.

4.4.1 Program INFO0 through Wired Update

1. Create INFO0-NOOTA Wired Update Image blob from the INFO0 image in the previous step:

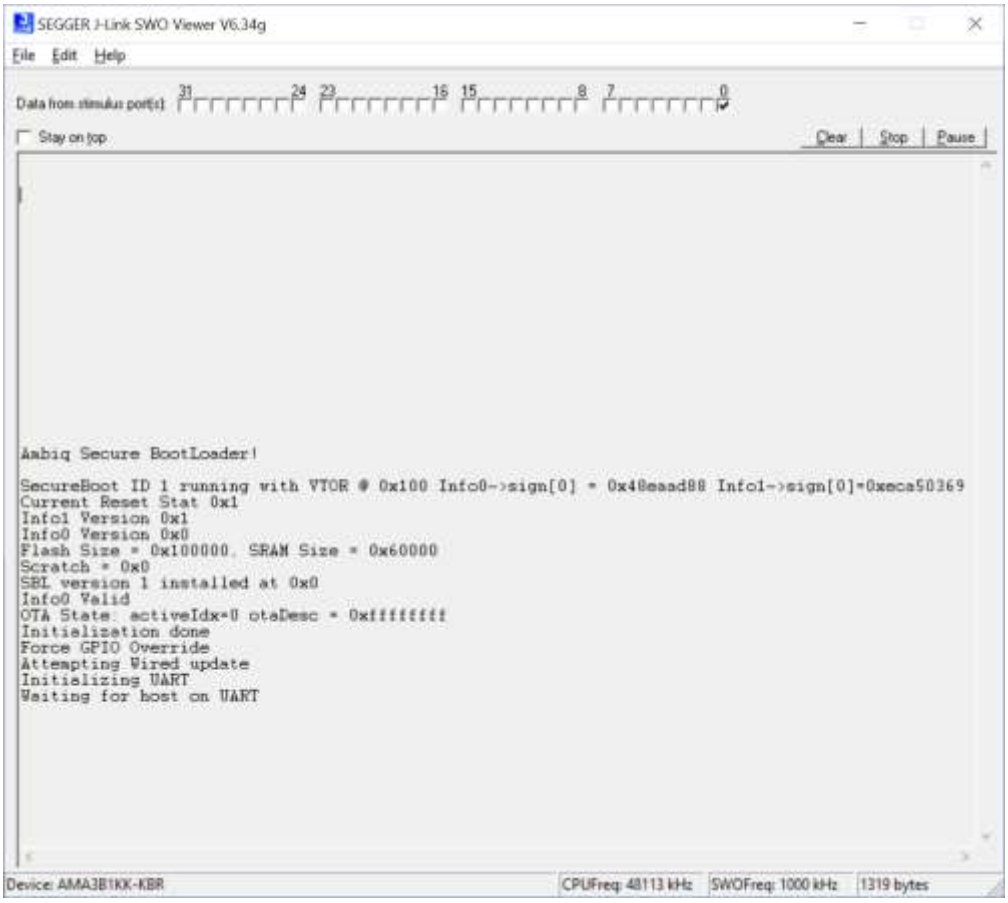
```
./create_cust_wireupdate_blob.py --bin info0.bin -o info0_wire -i 32 --load-address 0
```

2. Reset the Apollo3 EVB by pressing the SYSTEM RESET button as you should see:



Note the “Info0 Valid”. Our Apollo3 EVBs are shipped with valid INFO0 and the binary_counter example pre-programmed.

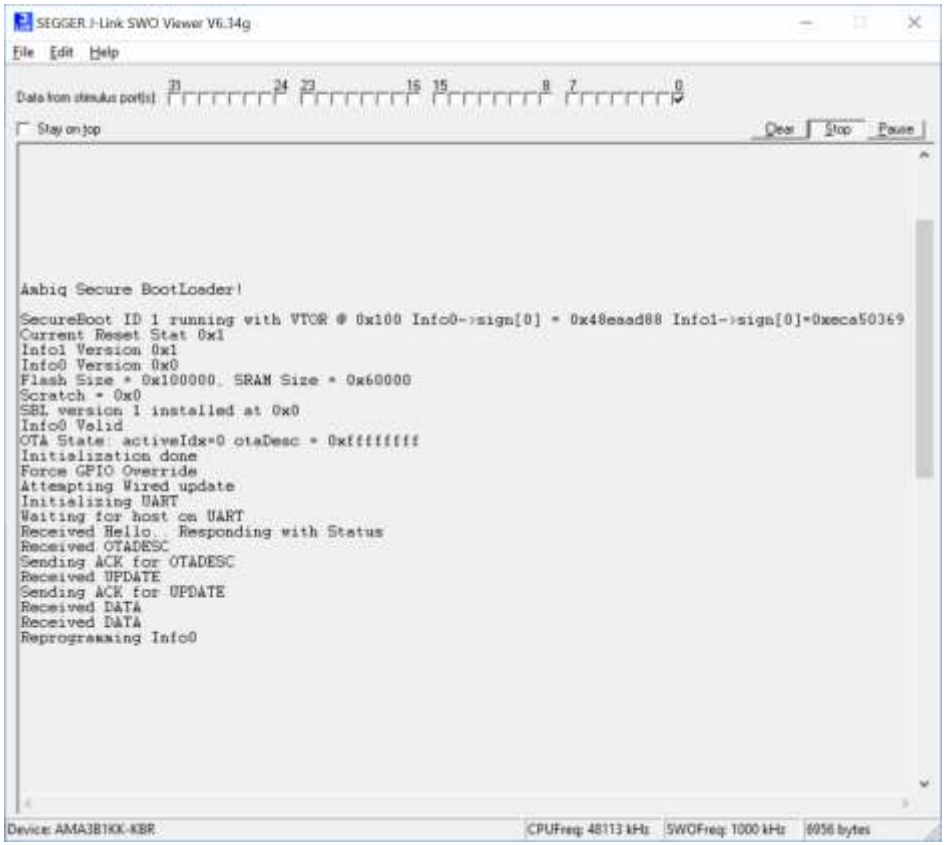
3. Holding BTN2 while pressing the SYSTEM RESET button you should see:



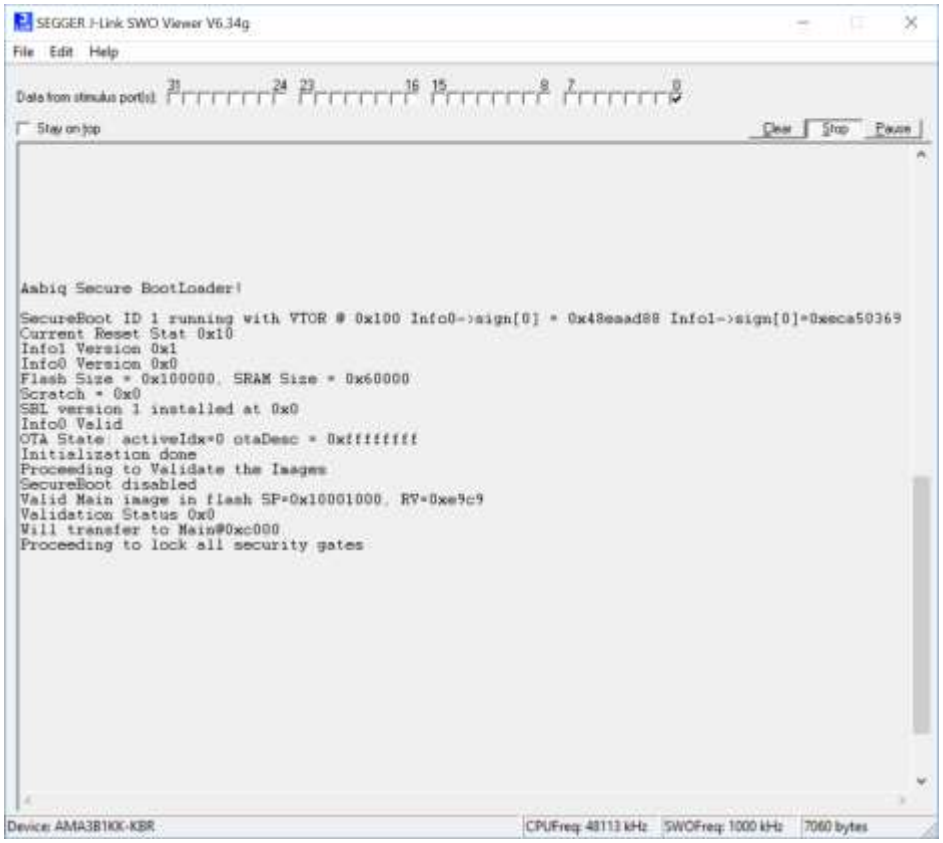
4. Within 5 seconds, use the UART Wired Update script to transfer the INFO0-NOOTA blob to the Secure Bootloader:

```
./uart_wired_update.py -b 115200 COM<X> -r 0 -f info0_wire.bin -i 32
```

where COM<X> is the PC COM port connected to the Apollo3 EVB. After which the display on the JLink SWO viewer should be:



5. Reset the board (without holding BTN2) and you should see:



4.4.2 Program INFO0 through JLink Commander

Ambiq SDK provides a windows batch file “program_info0.bat” which use the JLink Commander scripting language to program INFO0. The script needs to edit the file for the location of info0.bin.

After that, run this from windows command line as follows.

For Apollo3Blue:

```
./program_info0.bat AMA3B1KK-KBR
```

or

```
./program_info0.bat AMA3B1KK-KCR
```

For Apollo3BluePlus:

```
./program_info0.bat AMA3B2KK-KBR
```

or

```
./program_info0.bat AMA3B2KK-KCR
```

It is important to note that when using this method - there is no built in error checking. Users need to independently verify that programming was successful (e.g. by reading the infospace back and then comparing with expected values).

Top half of INFO0 (0x1000-0x1FFF) is read protected. Reading back complete INFO0 requires special handling to unlock top half meant for security information like keys etc, or any other sensitive information customer may want to keep. Access to this area is restricted.

If one does not care about this region, just reading back the first 0x1000 bytes and comparing is enough.
If verifying complete INFO0 is needed, special procedure is defined for accessing the protected half of INFO0.

4.4.2.1 Special Handling required for reading back INFO0

Top half of INFO0 (offset 0x1000 onwards) is meant for security information like keys etc, or any other sensitive information customer may want to keep.

Access to this area is restricted.

To unlock reading of this region, unique 128b "customer key" needs to be written to the lock registers.
Customer Key is whatever value customer programmed in INFO0 as part of initial programming (INFO0_CUSTOMER_KEY0 ADDRESS: 0x50021A00 to INFO0_CUSTOMER_KEY3 ADDRESS: 0x50021A0C).
This way it is known only to the customer.

The following needs to be done for unlocking this region of INFO0:

- Write 0x1 to REG_SECURITY_LOCKCTRL ADDRESS: 0x40030078
- Write the key value to REG_SECURITY_KEY0 ADDRESS: 0x40030080 to REG_SECURITY_KEY3 ADDRESS: 0x4003008C

One can confirm that the region is unlocked by checking REG_SECURITY_LOCKSTAT ADDRESS: 0x4003007C (should be 0x1)

Once the need for access is done, the same procedure can be repeated, but this time with some wrong value to the key registers to lock the access.

Ambiq SDK also provides a windows batch file “verify_info0.bat” which use the JLink Commander scripting language to extract the current INFO0 contents to a file called info0_dump.bin, compare it to info0.bin and state the result of the comparison. Note that the customer keys are set to the default in this script. Edit the customer keys in order to unlock INFO0 access.

To run this from windows command line as follows:

For Apollo3Blue:

```
./verify_info0.bat AMA3B1KK-KBR
```

or

```
./verify_info0.bat AMA3B1KK-KCR
```

For Apollo3BluePlus:

```
./verify_info0.bat AMA3B2KK-KBR
```

or

```
./verify_info0.bat AMA3B2KK-KCR
```

5. Firmware image for non-secure Boot

An IDE like IAR could be used to both generate and flash the images to Apollo3-Blue for debugging.

Alternatively, the images can be generated using an IDE, and then flashed using other means (e.g. JFlashLite, or through SBL wired update). The IDE can then be used to attach to a running target for debugging.

5.1 Using the IAR IDE with Secure Bootloader

At this point (and hereafter) it is possible to use the IAR IDE to load and program. The Apollo3-SDK release includes modifications to the build system to instantiate the JLink script changes for the IAR and Keil IDEs. In addition, all of the examples have been updated to relocate the Flash base address to 0xC000.

Build the hello_world example and try loading it through the IAR debugger. The Debug Log should look like this:

```
Tue Jan 29, 2019 12:09:12: IAR Embedded Workbench 8.32.2 (C:\Program Files (x86)\IAR
Systems\Embedded Workbench 8.2\arm\bin\armproc.dll)
Tue Jan 29, 2019 12:09:12: Loaded macro file: C:\Program Files (x86)\IAR
Systems\Embedded Workbench 8.2\arm\config\debugger\AmbiqMicro\apollo3.dmac
Tue Jan 29, 2019 12:09:12: Device "AMA3B1KK-KBR" selected.
Tue Jan 29, 2019 12:09:12: JLINK command: ProjectFile = C:\AmbiqMicro\AmbiqSuite-
Rel2.0.0\boards\apollo3_evb\examples\hello_world\iar\settings\hello_world_Debug.jli
nk, return = 0
Tue Jan 29, 2019 12:09:12: JLINK command: scriptfile = C:\Program Files (x86)\IAR
Systems\Embedded Workbench 8.2\arm\config\debugger\AmbiqMicro\AMA3B1KK-
KBR.JLinkScript, return = 0
Tue Jan 29, 2019 12:09:12: Device "AMA3B1KK-KBR" selected.
Tue Jan 29, 2019 12:09:12: DLL version: V6.40 , compiled Oct 26 2018 15:06:02
Tue Jan 29, 2019 12:09:12: Firmware: J-Link OB-SAM3U128 V3 compiled Jul 12 2018
12:17:50
Tue Jan 29, 2019 12:09:12: Selecting SWD as current target interface.
Tue Jan 29, 2019 12:09:12: JTAG speed is initially set to: 1000 kHz
Tue Jan 29, 2019 12:09:12: Found SW-DP with ID 0x2BA01477
Tue Jan 29, 2019 12:09:12: Scanning AP map to find all available APs
Tue Jan 29, 2019 12:09:12: AP[1]: Stopped AP scan as end of AP map has been reached
Tue Jan 29, 2019 12:09:12: AP[0]: AHB-AP (IDR: 0x24770011)
Tue Jan 29, 2019 12:09:12: Iterating through AP map to find AHB-AP to use
Tue Jan 29, 2019 12:09:12: AP[0]: Core found
Tue Jan 29, 2019 12:09:12: AP[0]: AHB-AP ROM base: 0xE00FF000
Tue Jan 29, 2019 12:09:12: CPUID register: 0x410FC241. Implementer code: 0x41 (ARM)
Tue Jan 29, 2019 12:09:12: Found Cortex-M4 r0p1, Little endian.
Tue Jan 29, 2019 12:09:12: FPUnit: 6 code (BP) slots and 2 literal slots
Tue Jan 29, 2019 12:09:12: CoreSight components:
```

```
Tue Jan 29, 2019 12:09:12: ROMTbl[0] @ E00FF000
Tue Jan 29, 2019 12:09:12: ROMTbl[0][0]: E000E000, CID: B105E00D, PID: 000BB00C SCS-
M7
Tue Jan 29, 2019 12:09:12: ROMTbl[0][1]: E0001000, CID: B105E00D, PID: 003BB002 DWT
Tue Jan 29, 2019 12:09:12: ROMTbl[0][2]: E0002000, CID: B105E00D, PID: 002BB003 FPB
Tue Jan 29, 2019 12:09:12: ROMTbl[0][3]: E0000000, CID: B105E00D, PID: 003BB001 ITM
Tue Jan 29, 2019 12:09:12: ROMTbl[0][4]: E0040000, CID: B105900D, PID: 000BB9A1 TPIU
Tue Jan 29, 2019 12:09:12: Executing J-Link script file function: ResetTarget()
Tue Jan 29, 2019 12:09:12: JDEC PID 0x000000CF
Tue Jan 29, 2019 12:09:12: Ambiq Apollo3 ResetTarget
Tue Jan 29, 2019 12:09:12: Bootldr = 0x04000000
Tue Jan 29, 2019 12:09:12: Secure Part.
Tue Jan 29, 2019 12:09:12: Secure Chip. Bootloader needs to run which will then halt
when finish.
Tue Jan 29, 2019 12:09:12: CPU halted after reset. Num Tries = 0x00000000
Tue Jan 29, 2019 12:09:12: Hardware reset with strategy 0 was performed
Tue Jan 29, 2019 12:09:12: Initial reset was performed
Tue Jan 29, 2019 12:09:12: 512 bytes downloaded (6.41 Kbytes/sec)
Tue Jan 29, 2019 12:09:12: Loaded debugee: C:\Program Files (x86)\IAR Systems\Embedded
Workbench 8.2\arm\config\flashloader\AmbiqMicro\FlashApollo3_RAM256K.out
Tue Jan 29, 2019 12:09:12: Target reset
Tue Jan 29, 2019 12:09:13: Downloaded C:\AmbiqMicro\AmbiqSuite-
Rel2.0.0\boards\apollo3_evb\examples\hello_world\iar\bin\hello_world.out to flash
memory.
Tue Jan 29, 2019 12:09:13: 8598 bytes downloaded into FLASH (8.26 Kbytes/sec)
Tue Jan 29, 2019 12:09:13: Executing J-Link script file function: ResetTarget()
Tue Jan 29, 2019 12:09:13: JDEC PID 0x000000CF
Tue Jan 29, 2019 12:09:13: Ambiq Apollo3 ResetTarget
Tue Jan 29, 2019 12:09:13: Bootldr = 0x04000000
Tue Jan 29, 2019 12:09:13: Secure Part.
Tue Jan 29, 2019 12:09:13: Secure Chip. Bootloader needs to run which will then halt
when finish.
Tue Jan 29, 2019 12:09:13: CPU halted after reset. Num Tries = 0x00000000
Tue Jan 29, 2019 12:09:13: Hardware reset with strategy 0 was performed
Tue Jan 29, 2019 12:09:13: 8598 bytes downloaded into FLASH (16.79 Kbytes/sec)
Tue Jan 29, 2019 12:09:13: Loaded debugee: C:\AmbiqMicro\AmbiqSuite-
Rel2.0.0\boards\apollo3_evb\examples\hello_world\iar\bin\hello_world.out
Tue Jan 29, 2019 12:09:13: Executing J-Link script file function: ResetTarget()
Tue Jan 29, 2019 12:09:13: JDEC PID 0x000000CF
Tue Jan 29, 2019 12:09:13: Ambiq Apollo3 ResetTarget
Tue Jan 29, 2019 12:09:13: Bootldr = 0x04000000
Tue Jan 29, 2019 12:09:13: Secure Part.
Tue Jan 29, 2019 12:09:13: Secure Chip. Bootloader needs to run which will then halt
when finish.
Tue Jan 29, 2019 12:09:13: CPU halted after reset. Num Tries = 0x00000000
Tue Jan 29, 2019 12:09:13: Hardware reset with strategy 0 was performed
Tue Jan 29, 2019 12:09:13: Target reset
```

5.2 Programming the device using SBL assisted Wired update

This option requires reformatting the image generated by an IDE to an update format as understood by the SBL, and then using the Wired Update method to let SBL update the image.

5.2.1 Generating Main Customer Image Upgrade Blob

This example demonstrates how to create a customer main image Upgrade blob, which can then be used for upgrading the main image in flash, either using the OTA or wired update process.

Create a non-secure customer image from a built binary with Flash base address of 0xC000. This is the Customer Main Non-Secure format from the Apollo3 Security Whitepaper.

```
./create_cust_image_blob.py --bin hello_world.bin --load-address 0xC000 --magic-
num 0xCB -o main_nonsecure_ota --version 0x0
```

The generated image blob (main_nonsecure_ota.bin) can then be used to upgrade the program in the flash using the wired update (section 5.2.2).

5.2.2 Wired Update Example – Main Customer Image

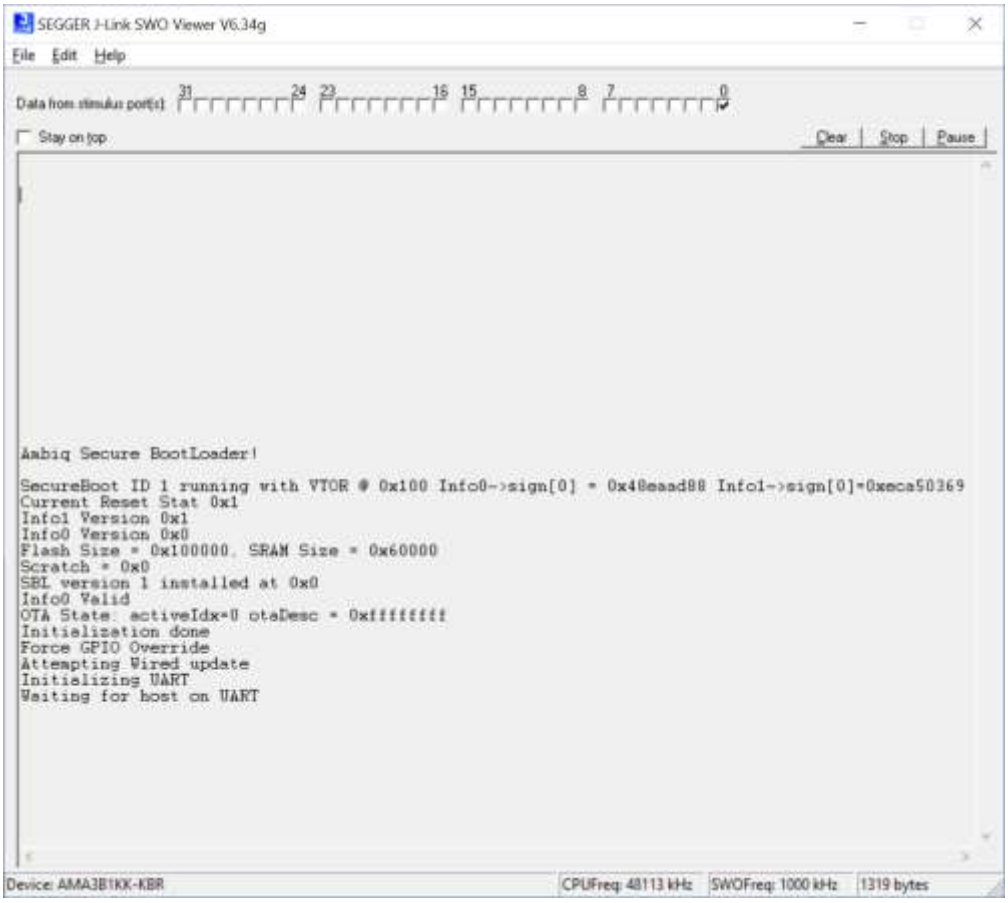
5.2.2.1 UART Wired Update

This example demonstrates how to create and load a customer main image using the Secure Bootloader Wired Update protocol. In this example, INFO0 has been configured as outlined in section 4.1 above.

6. Create Non-Secure Wired Update Image blob corresponding to the Upgrade image, as shown in the Apollo3-Blue Secure Update Flow document:

```
./create_cust_wireupdate_blob.py --load-address 0x20000 --bin
main_nonsecure_ota.bin -i 6 -o main_nonsecure_wire --options 0x1
```

7. Hold BTN2 on the Apollo3 EVB while pressing SYSTEM RESET, you should see:



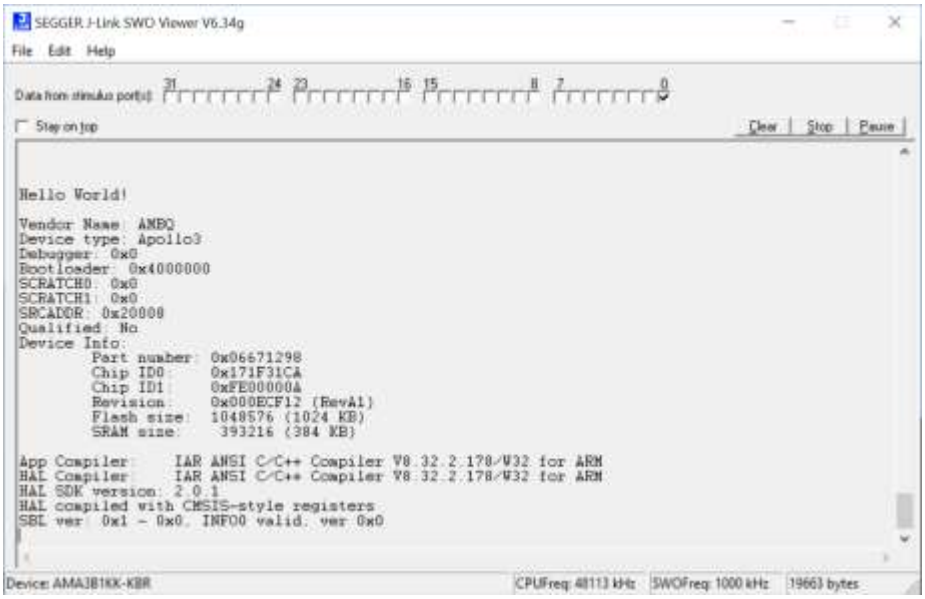
8. Within 5 seconds, use the UART Wired Update script to transfer the Non-Secure Wired Update image blob to the Secure Bootloader[1]:

```
./uart_wired_update.py -b 115200 COM<X> -r 1 -f main_nonsecure_wire.bin -i 6
```

   where COM<X> is the PC COM port connected to the Apollo3 EVB.

[1] The default command assumes last page of available flash to construct the OTA descriptor page, as required by the Upgrade process, as described in [REF1]. For non-default allocation of the OTA descriptor page, it can be specified using –o parameter.

9. Reset the board (without holding BTN2) and you should see:



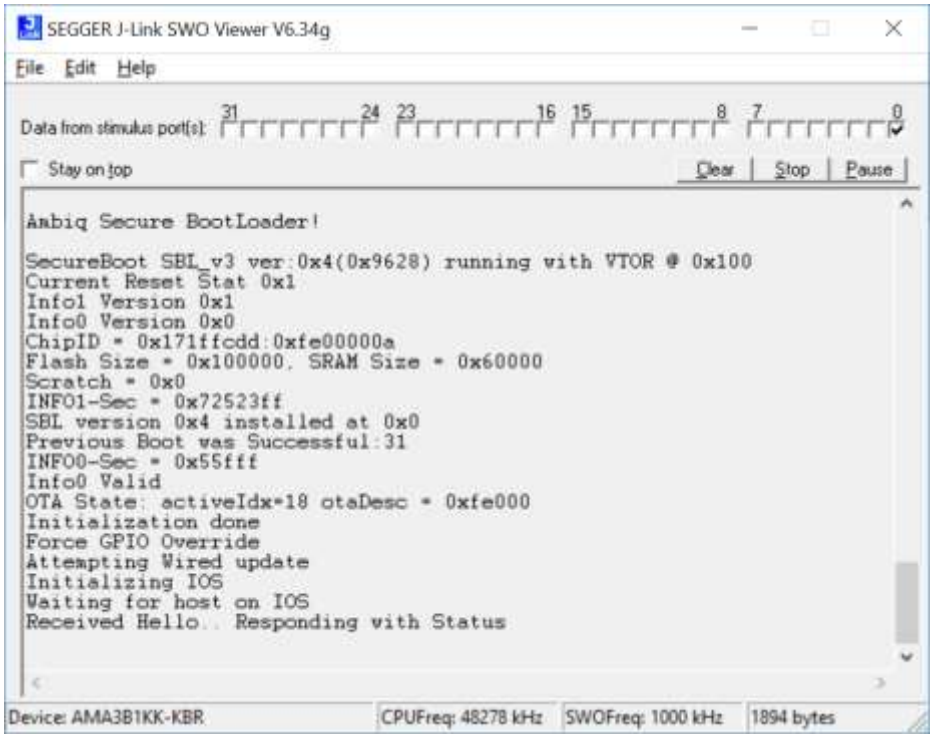
5.2.2.2 IOS (SPI) Wired Update

This example demonstrates how to create and load a customer main image using the Secure Bootloader Wired Update protocol. In this example, INFO0 has been configured as outlined in section □ or □ above. In addition, this section assumes that the Apollo3 EVB has been upgraded to SBL-v3 as outlined in section 7 below. Finally, this example assumes that a Host Apollo3 EVB is programmed with the uart_boot_host example and connected as shown in section 3.2.2 above.

1. Create Non-Secure Wired Update Image blob corresponding to the Upgrade image, as shown in the Apollo3-Blue Secure Update Flow document:

```
./create_cust_wireupdate_blob.py --load-address 0x20000 --bin
main_nonsecure_ota.bin -i 6 -o main_nonsecure_wire --options 0x1
```

2. Press the SYSTEM RESET button on the Host Apollo3 EVB, you should see:



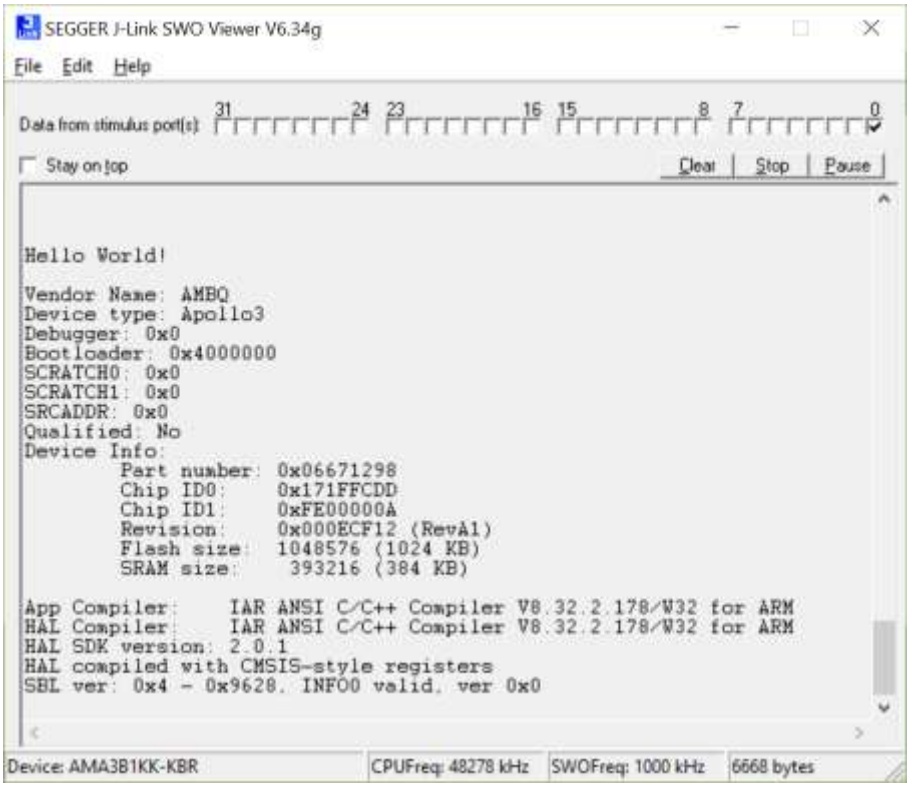
Notice that the uart_boot_host program immediately and autonomously sends a HELLO message and receives a STATUS message in order to capture the interface. It is now ready to accept UART-to-SPI traffic via the python scripts.

3. Use the UART Wired Update script to transfer the Non-Secure Wired Update image blob to the Secure Bootloader:

```
./uart_wired_update.py -b 115200 COM<X> -r 1 -f main_nonsecure_wire.bin -i 6
```

where COM<X> is the PC COM port connected to the Host Apollo3 EVB. You should see:

4. Connect pin 16 on the Slave to VDD and press the SYSTEM RESET button, you should see:



5.3 Programming Images with JFlashLite

Programming images with JFlashLite is similar to the behavior without the Secure Bootloader, but it is important to note the differences. Valid images should be located and downloaded at 0xC000. If there is no valid image, when JFlashLite attaches to the Apollo3 it will be at a stage after the SBL has run and it waiting in an infinite loop. Once the flashing operation is complete, JFlashLite with resume the program, but it will still be inside the infinite loop. Pressing a reset will bring the Apollo3 through a normal boot operation which will begin execution of the image at 0xC000.

“Erase Chip” operation will not work from JFlashLite, as the pre-installed Secure Bootloader is protected, and cannot be erased. Equivalent effect could be achieved by programming all 1’s to the desired sectors of the flash.

5.4 Programming Images with JFlash

Programming images with JFlash is similar to the behavior without the Secure Bootloader, but it is important to note the differences. Valid images should be located and downloaded at 0xC000. If there is no valid image, when JFlash attaches to the Apollo3 it will be at a stage after the SBL has run and it waiting in an infinite loop. Once the flashing operation is complete, JFlash with resume the program, but it will still be inside the infinite loop. Pressing a reset will bring the Apollo3 through a normal boot operation which will begin execution of the image at 0xC000. SEGGER Tools V6.34 and beyond are aware of the SBL Flash configuration for Apollo3-Blue, so the tool can be used as normal.

6. Secure Boot

For Apollo3-Blue Secure SKU's customers can optionally enable secure boot. When enabled for Secure Boot, SBL transfers control to the main image only after it passes required validation checks.

Secure Boot is enabled by programming INFO0 SECURITY settings as per customers' requirements, giving flexibility on the level of enforcement of security policies. Please refer to [REF3] for more details.

This section briefly lists essential steps for getting started with secure boot, and highlights differences with non-secure parts. Working with secure images is inherently more involved and not as debug friendly. Hence, it is expected that most of the development work is done using non-secure settings, and once the program has been validated, security is enabled as one of final steps.

6.1 Enabling Secure Boot in INFO0

As mentioned above, secure boot is enabled by setting the INFO0 security settings. Procedure for generating and programming the INFO0 is same as listed in section 4, except for a small difference in INFO0 generation to enable secure boot.

- Create INFO0 image with GPIO Override is set to pin 16 (0x10) active low. Baudrate for INFO0 UART is set to 115200 (0x1C200). Main image is expected at 0xC000. Apollo3 is configured for UART-RX pin 23 (0x17) & UART-TX pin 22 (0x16). Secure Boot is enabled (note the –s option).

```
./create_info0.py --valid 1 info0 --pl 1 --u0 0x1C200c0 --u1 0xFFFF1617 --u2 0x2
--u3 0x0 --u4 0x0 --u5 0x0 --main 0xC000 --gpio 0x10 --version 0 --wTO 5000 –s 1
--chipType apollo3
```

- Create INFO0 image with GPIO Override is set to pin 16 (0x10) active low. Baudrate for INFO0 UART is set to 115200 (0x1C200). Main image is expected at 0xC000. Apollo3 is configured for UART-RX pin 23 (0x17) & UART-TX pin 22 (0x16). Secure Boot is enabled (note the –s option).

```
./create_info0.py --valid 1 info0 --pl 1 --u0 0x1C200c0 --u1 0xFFFF1617 --u2 0x2
--u3 0x0 --u4 0x0 --u5 0x0 --main 0xC000 --gpio 0x10 --version 0 --wTO 5000 –s 1
--chipType apollo3p
```

6.2 Firmware image for Secure Boot

For secure boot, the images are reformatted with additional security header information for SBL to use for verification.

Provisioning secure parts is a multi-step process to program permanent images when using secure boot.

- The images can first be generated using an IDE
    - The link address needs to be set to a value 0x100 more than the mainPtr configured in INFO0, to allow for required security headers (e.g. when using default 0xC000 as the main image location, one should link the program at 0xC100)
- Images are reformatted for SBL to understand
    - This step creates the required security headers
- Reformatted image is then flashed using other means (e.g. JFlashLite, or through SBL wired update) at flash address specified by mainPtr in INFO0 (e.g. 0xC000).
- The IDE can then be used to attach to a running target for debugging.

Note that IDEs can still be used as a one-step shop for generating, programming and debugging the programs (as in section 5.1). However, on reset the Secure Boot will fail due to lack of compatible security information for SBL.

7. Secure Bootloader Update

From time to time, Ambiq will provide updates to the preinstalled Secure Bootloader (SBL). These updates are provided as binary files.

Customers have a choice to perform the SBL upgrade either using their OTA protocol, or use the SBL provided wired update protocol.

When using either method for SBL upgrade, there are certain restrictions to keep in mind:

1. It needs to be ensured that SBL OTA is provided exclusively (SBL upgrade cannot be bundled with other images during OTA process).
2. SBL can be upgraded only if the device is booting successfully otherwise. A successful boot means SBL booting to a valid main firmware image (secure or non-secure).
    - This means that SBL cannot be upgraded on a vanilla factory part with no main image. A test main image needs to be installed first, before SBL can be upgraded.

When using wired update method, following steps can be followed:

7.1 Create Secure Bootloader (SBL) Wired Update Image blob

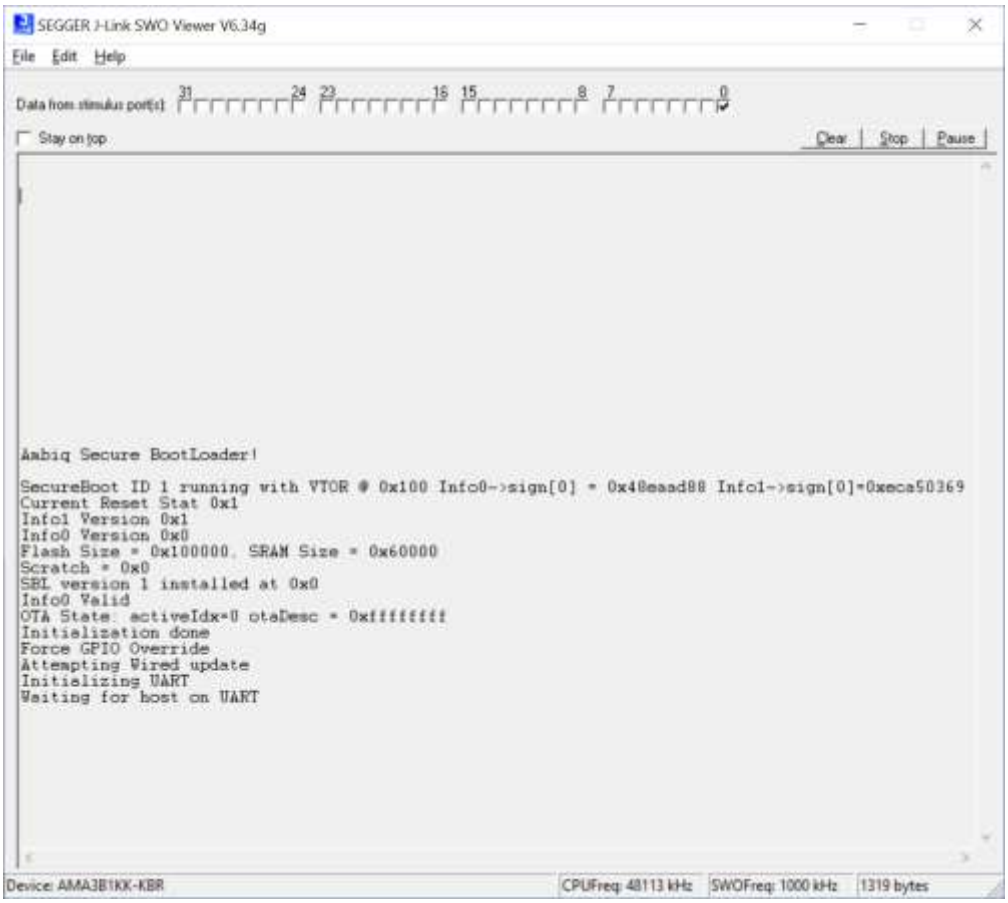
Create SBL Wired Update Image blob corresponding to the Upgrade image (sbl.bin provided by Ambiq):

```
./create_cust_wireupdate_blob.py --load-address 0xF0000 --bin ./sbl_updates/<SBL
Binary file provided by Ambiq> -i 0 -o sbl_wire --options 0x1
```

7.2 Program SBL Upgrade Firmware

To use the UART Wired Update script to upgrade SBL Firmware using the SBL wire update blob (generated as in section 7.1):

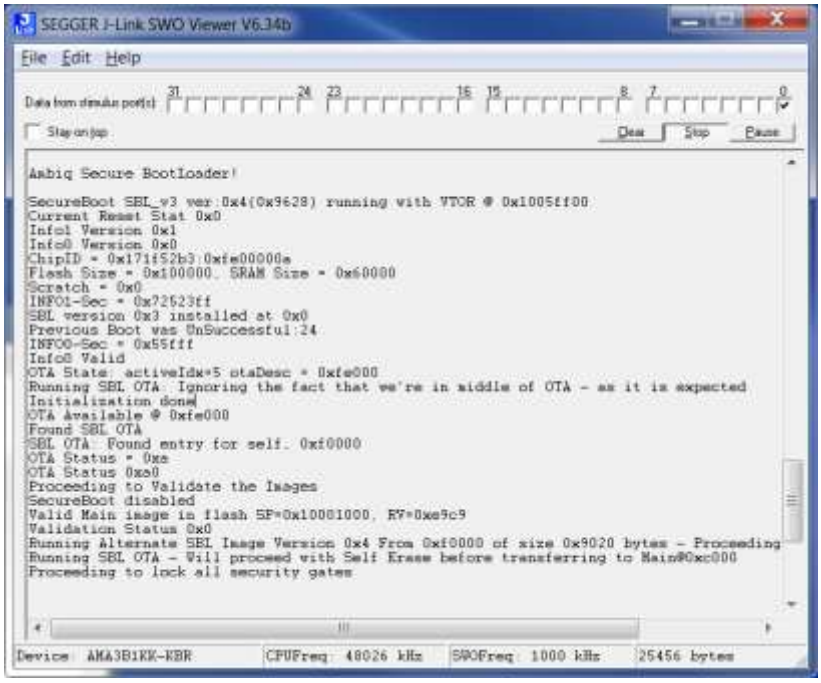
1. Hold BTN2 on the Apollo3 EVB while pressing SYSTEM RESET, you should see:



2. Within 5 seconds, use the UART Wired Update script to transfer the Non-Secure Wired Update image blob to the Secure Bootloader:

```
./uart_wired_update.py -b 115200 COM<X> -r 1 -f sbl_wire.bin -i 0
```

3. Reset the board (without holding BTN2) and you should see:



7.3 Program SBL Upgrade Firmware through JLink Commander

The SBL may also be programmed via the following script:

```
/tools/apollo3_scripts/jlink-prog-sbl.txt
```

The script may be modified based on the flash addressing of Apollo3Blue vs. Apollo3BluePlus and the customer's requirements. The default loads the SBL image to 0x80000 and the OTA Descriptor to 0xF0000 which will work for both Apollo3Blue EVB and the Apollo3BluePlus EVB. The device must be specified on the command line as follows:

For Apollo3Blue:

```
JLink.exe -device AMA3B1KK-KBR -CommanderScript jlink-prog-sbl.txt
```

For Apollo3BluePlus:

```
JLink.exe -device AMA3B2KK-KBR -CommanderScript jlink-prog-sbl.txt
```

Contact Information

Address Ambiq Micro, Inc.
6500 River Place Blvd.
Building 7, Suite 200
Austin, TX 78730

Phone +1 (512) 879-2850

Website http://www.ambiqmicro.com
General Information info@ambiqmicro.com
Sales sales@ambiqmicro.com
Technical Support support@ambiqmicro.com

Legal Information and Disclaimers

AMBIQ MICRO INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. AMBIQ MICRO MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. AMBIQ MICRO AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". AMBIQ MICRO AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

AMBIQ MICRO DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF AMBIQ MICRO COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM AMBIQ MICRO UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF AMBIQ MICRO.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE AMBIQ MICRO PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. AMBIQ MICRO RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. AMBIQ MICRO MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES AMBIQ MICRO ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. “TYPICAL” PARAMETERS WHICH MAY BE PROVIDED IN AMBIQ MICRO DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING “TYPICALS” MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER’S TECHNICAL EXPERTS. AMBIQ MICRO DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. AMBIQ MICRO PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE AMBIQ MICRO PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE AMBIQ MICRO PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD AMBIQ MICRO AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT AMBIQ MICRO WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.