Apollo3-Blue
Getting Started Guide

Revision 2.1
July 2018

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

Contents

1. Introduction

Vanilla Apollo3-Blue 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.

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	

3. Setting up the environment

The untouched Apollo3-Blue part has no INFO0 (uninitialized) and no main image. So, SBL will wait for wired update for ~5 sec before getting in to a spin loop (when debugger can connect). That means when you first boot up the part, you literally cannot do anything for ~10 sec. After that time, you can then connect the JLink SWO viewer. Once connected you can see the prints for subsequent boots.

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

In addition, the windows PC must be connected via a USB-UART adapter to pins:

- Apollo3 UART-RX pin 49
- Apollo3 UART-TX pin 48

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

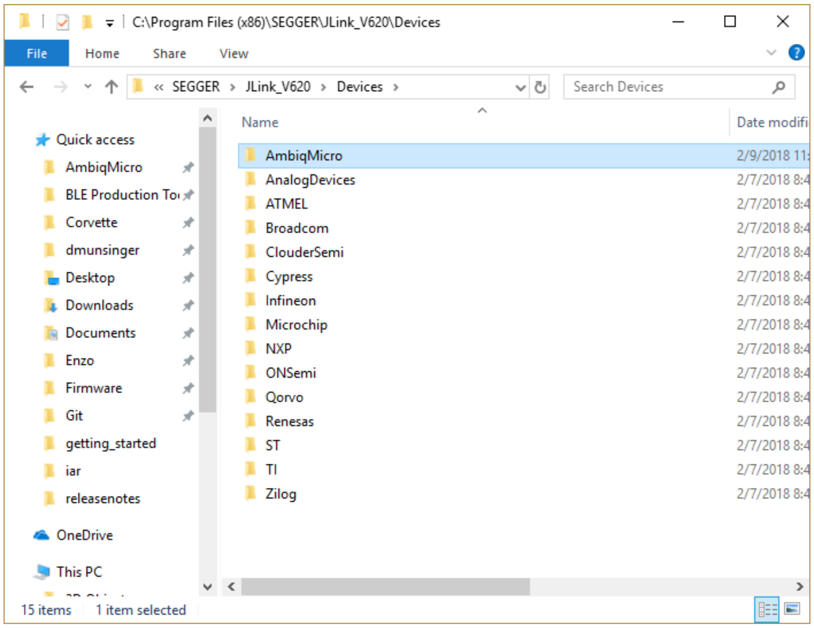
Apollo3-Blue is the first Ambiq MCU with a Secure Bootloader. As such, the JLink tools need to be customized to interact with the initialization of the Apollo3-Blue boot process.

The Apollo3-Blue MCU is natively supported by the SEGGER JLink tools as of V6.34 by simply selecting AMA3BKK-KBR as the target device. The following steps must not be applied when using JLink V6.34 or later. Ambiq recommends using the latest SEGGER JLink tools and uninstalling previous versions prior to upgrading.

The remainder of this section describes how to install a modified device specific JLink script for the Apollo3-Blue MCU for SEGGER tools prior to JLink V6.33 (Beta). The target device in this case should be AMAPH1KK-KBR.

The JLink script can be found in the **\tools\apollo3_scripts\** directory. It is called AMAPH1KK-KBR.JLinkScript.

The first step is to create an AmbiqMicro directory in your JLink tools local install. For example, in C:\Program Files (x86)\SEGGER\JLink_V620\Devices create an AmbiqMicro folder as follows:



Name	Date modifi
AmbiqMicro	2/9/2018 11:
AnalogDevices	2/7/2018 8:4
ATMEL	2/7/2018 8:4
Broadcom	2/7/2018 8:4
ClouderSemi	2/7/2018 8:4
Cypress	2/7/2018 8:4
Infineon	2/7/2018 8:4
Microchip	2/7/2018 8:4
NXP	2/7/2018 8:4
ONSemi	2/7/2018 8:4
Qorvo	2/7/2018 8:4
Renesas	2/7/2018 8:4
ST	2/7/2018 8:4
TI	2/7/2018 8:4
Zilog	2/7/2018 8:4

Copy the AMAPHKK-KBR.JLinkScript to this directory.

In order for the JLink tools to see and use the script, we need to update the JLinkDevices.xml file in the primary (JLink_V620) directory with the following:

```
  <Device>
    <ChipInfo Vendor="AmbiqMicro" Name="AMAPH1KK-KBR" Core="JLINK_CORE_CORTEX_M4" 
JLinkScriptFile="Devices/AmbiqMicro/AMAPH1KK-KBR.JLinkScript" />
  </Device>
```

This will direct the JLink tools to use the modified script.

4. Programming Customer InfoSpace (INFO0)

4.1 Generate INFO0

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

- Create INFO0 image with GPIO Override is set to pin 47 (0x2f) active low. Baudrate for INFO0 UART is set to 115200 (0x1C200). Main image is expected at 0xC000. Apollo3 is configured for UART-RX pin 49 (0x31) & UART-TX pin 48 (0x30).

```
./create_info0.py --valid 1 info0 --pl 1 --u0 0x1C200c0 --u1 0xFFFF3031 --u2 0x2
--u3 0x0 --u4 0x0 --u5 0x0 --main 0xC000 --gpio 0x2f --version 0 --wTO 5000
```

4.2 Program INFO0

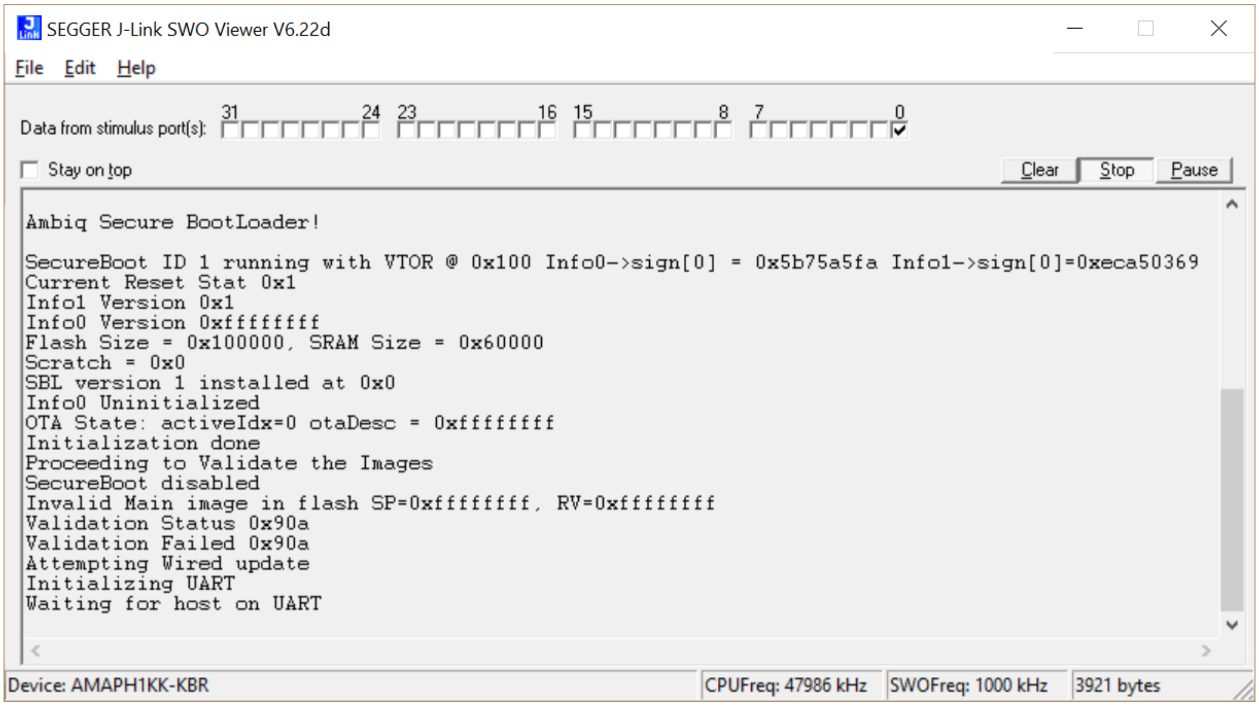
INFO0 can be programmed either using a JLink script, or through the SBL assisted Wired Update.

4.2.1 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_eb as you should see:



```
Ambiq Secure BootLoader!

SecureBoot ID 1 running with VTOR @ 0x100 Info0->sign[0] = 0x5b75a5fa Info1->sign[0]=0xeca50369
Current Reset Stat 0x1
Info1 Version 0x1
Info0 Version 0xffffffff
Flash Size = 0x100000, SRAM Size = 0x60000
Scratch = 0x0
SBL version 1 installed at 0x0
Info0 Uninitialized
OTA State: activeIdx=0 otaDesc = 0xffffffff
Initialization done
Proceeding to Validate the Images
SecureBoot disabled
Invalid Main image in flash SP=0xffffffff, RV=0xffffffff
Validation Status 0x90a
Validation Failed 0x90a
Attempting Wired update
Initializing UART
Waiting for host on UART
```

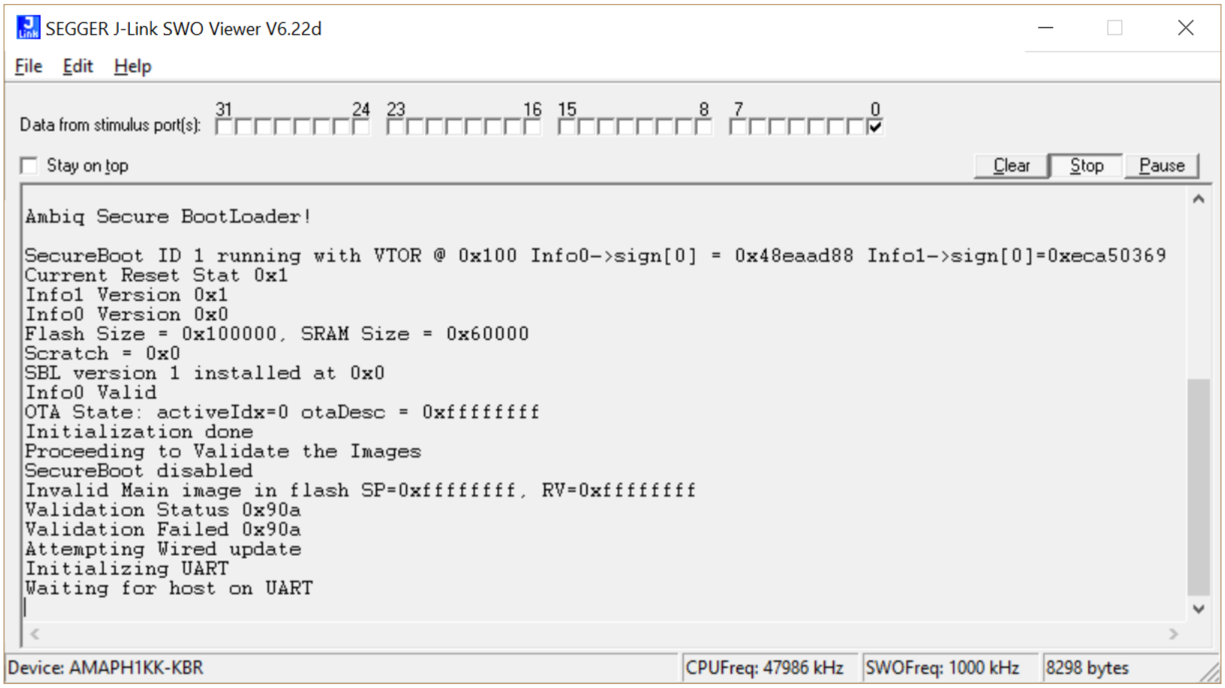
Device: AMAPH1KK-KBR | CPUFreq: 47986 kHz | SWOFreq: 1000 kHz | 3921 bytes

Note the “Info0 Uninitialized”.

3. 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_eb. After which the display on the JLink SWO viewer should be:



```
Ambiq Secure BootLoader!

SecureBoot ID 1 running with VTOR @ 0x100 Info0->sign[0] = 0x48eaad88 Info1->sign[0]=0xeca50369
Current Reset Stat 0x1
Info1 Version 0x1
Info0 Version 0x0
Flash Size = 0x100000, SRAM Size = 0x60000
Scratch = 0x0
SBL version 1 installed at 0x0
Info0 Valid
OTA State: activeIdx=0 otaDesc = 0xffffffff
Initialization done
Proceeding to Validate the Images
SecureBoot disabled
Invalid Main image in flash SP=0xffffffff, RV=0xffffffff
Validation Status 0x90a
Validation Failed 0x90a
Attempting Wired update
Initializing UART
Waiting for host on UART
```

Note the “Info0 Valid”.

4.2.2 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:

```
    ./program_info0.bat
```

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.

4.2.2.1 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:

```
   ./verify_info0.bat
```

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:

```
Thu Feb 15, 2018 14:29:33: IAR Embedded Workbench 8.20.2 (C:\Program Files (x86)\IAR
Systems\Embedded Workbench 8.0\arm\bin\armproc.dll)
Thu Feb 15, 2018 14:29:33: Loaded macro file: C:\Program Files (x86)\IAR
Systems\Embedded Workbench 8.0\arm\config\debugger\AmbiqMicro\apollo2.dmac
Thu Feb 15, 2018 14:29:39: Device "AMAPH1KK-KBR" selected.
Thu Feb 15, 2018 14:29:39: JLINK command: ProjectFile = C:\Git\a2sd-
581\boards\apollo3_eb\examples\hello_world\iar\settings\hello_world_Debug.jlink,
return = 0
Thu Feb 15, 2018 14:29:39: Device "AMAPH1KK-KBR" selected.
Thu Feb 15, 2018 14:29:39: Selecting SWD as current target interface.
Thu Feb 15, 2018 14:29:39: JTAG speed is initially set to: 1000 kHz
Thu Feb 15, 2018 14:29:39: Found SW-DP with ID 0x2BA01477
Thu Feb 15, 2018 14:29:39: Scanning AP map to find all available APs
Thu Feb 15, 2018 14:29:39: AP[1]: Stopped AP scan as end of AP map has been reached
Thu Feb 15, 2018 14:29:39: AP[0]: AHB-AP (IDR: 0x24770011)
Thu Feb 15, 2018 14:29:39: Iterating through AP map to find AHB-AP to use
Thu Feb 15, 2018 14:29:39: AP[0]: Core found
Thu Feb 15, 2018 14:29:39: AP[0]: AHB-AP ROM base: 0xE00FF000
Thu Feb 15, 2018 14:29:39: CPUID register: 0x410FC241. Implementer code: 0x41 (ARM)
Thu Feb 15, 2018 14:29:39: Found Cortex-M4 r0p1, Little endian.
Thu Feb 15, 2018 14:29:39: FPUnit: 6 code (BP) slots and 2 literal slots
Thu Feb 15, 2018 14:29:39: CoreSight components:
Thu Feb 15, 2018 14:29:39: ROMTbl[0] @ E00FF000
Thu Feb 15, 2018 14:29:39: ROMTbl[0][0]: E000E000, CID: B105E00D, PID: 000BB00C SCS
Thu Feb 15, 2018 14:29:39: ROMTbl[0][1]: E0001000, CID: B105E00D, PID: 003BB002 DWT
Thu Feb 15, 2018 14:29:39: ROMTbl[0][2]: E0002000, CID: B105E00D, PID: 002BB003 FPB
Thu Feb 15, 2018 14:29:39: ROMTbl[0][3]: E0000000, CID: B105E00D, PID: 003BB001 ITM
Thu Feb 15, 2018 14:29:39: ROMTbl[0][4]: E0040000, CID: B105900D, PID: 000BB9A1 TPIU
Thu Feb 15, 2018 14:29:39: JDEC PID 0x000000CF
Thu Feb 15, 2018 14:29:39: Ambiq Apollo3 ResetTarget
Thu Feb 15, 2018 14:29:39: Bootldr = 0x04000000
Thu Feb 15, 2018 14:29:39: Secure Part.
Thu Feb 15, 2018 14:29:39: Secure Chip. Bootloader needs to run which will then halt
when finish.
Thu Feb 15, 2018 14:29:39: CPU halted after reset. Num Tries = 0x00000000
Thu Feb 15, 2018 14:29:40: Hardware reset with strategy 0 was performed
Thu Feb 15, 2018 14:29:40: Initial reset was performed
Thu Feb 15, 2018 14:29:40: 512 bytes downloaded (6.41 Kbytes/sec)
Thu Feb 15, 2018 14:29:40: Loaded debugee: C:\Program Files (x86)\IAR Systems\Embedded
Workbench 8.0\arm\config\flashloader\AmbiqMicro\FlashApollo2_RAM256K.out
Thu Feb 15, 2018 14:29:40: Target reset
Thu Feb 15, 2018 14:29:41: Downloaded C:\Git\a2sd-
581\boards\apollo3_eb\examples\hello_world\iar\bin\hello_world.out to flash memory.
Thu Feb 15, 2018 14:29:41: JDEC PID 0x000000CF
Thu Feb 15, 2018 14:29:41: Ambiq Apollo3 ResetTarget
Thu Feb 15, 2018 14:29:41: Bootldr = 0x04000000
Thu Feb 15, 2018 14:29:41: Secure Part.
Thu Feb 15, 2018 14:29:41: Secure Chip. Bootloader needs to run which will then halt
when finish.
Thu Feb 15, 2018 14:29:41: CPU halted after reset. Num Tries = 0x00000000
Thu Feb 15, 2018 14:29:41: Hardware reset with strategy 0 was performed
Thu Feb 15, 2018 14:29:41: 6454 bytes downloaded into FLASH (13.91 Kbytes/sec)
Thu Feb 15, 2018 14:29:41: Loaded debugee: C:\Git\a2sd-
581\boards\apollo3_eb\examples\hello_world\iar\bin\hello_world.out
Thu Feb 15, 2018 14:29:41: JDEC PID 0x000000CF
Thu Feb 15, 2018 14:29:41: Ambiq Apollo3 ResetTarget
Thu Feb 15, 2018 14:29:41: Bootldr = 0x04000000
Thu Feb 15, 2018 14:29:41: Secure Part.
Thu Feb 15, 2018 14:29:41: Secure Chip. Bootloader needs to run which will then halt
when finish.
Thu Feb 15, 2018 14:29:41: CPU halted after reset. Num Tries = 0x00000000
Thu Feb 15, 2018 14:29:41: Hardware reset with strategy 0 was performed
Thu Feb 15, 2018 14:29:41: 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

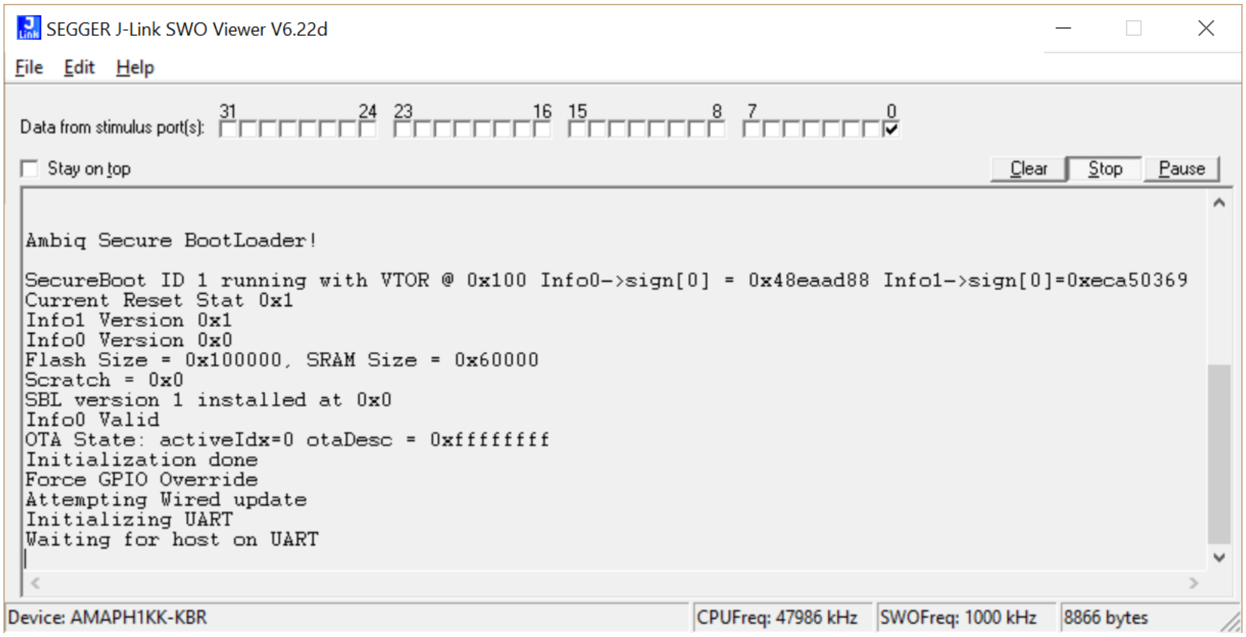
This example demonstrates how to create and load a customer main image using the Secure Bootloader Wired Update protocol. In this example, the GPIO Override is set to pin 47 active low. Baudrate for INFO0 UART is set to 115200.

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. Attach pin 47 (Override) to GND.

3. Reset the apollo3_eb as you should see:



```
Ambiq Secure BootLoader!

SecureBoot ID 1 running with VTOR @ 0x100 Info0->sign[0] = 0x48eaad88 Info1->sign[0]=0xeca50369
Current Reset Stat 0x1
Info1 Version 0x1
Info0 Version 0x0
Flash Size = 0x100000, SRAM Size = 0x60000
Scratch = 0x0
SBL version 1 installed at 0x0
Info0 Valid
OTA State: activeIdx=0 otaDesc = 0xffffffff
Initialization done
Force GPIO Override
Attempting Wired update
Initializing UART
Waiting for host on UART
```

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

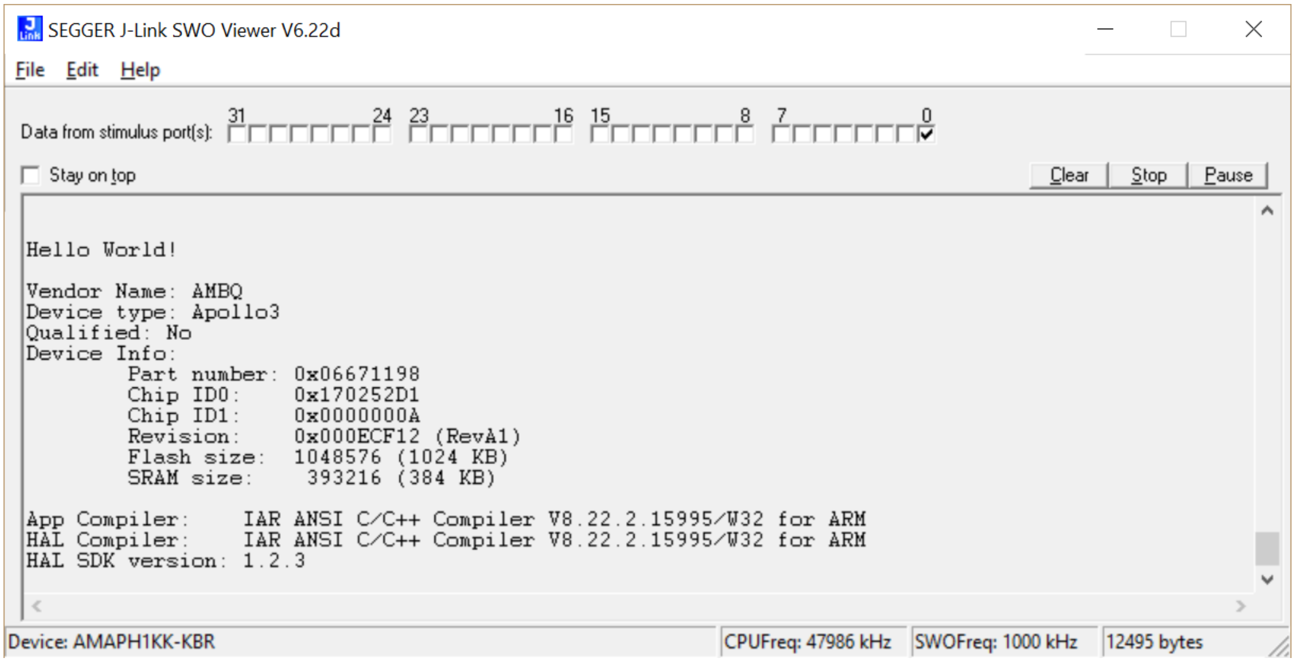
```
./uart_wired_update.py -b 115200 COM<X> -r 1 -f main_nonsecure_wire.bin -i 6
```

   where COM<X> is the PC COM port connected to the apollo3_eb.

5. Disconnect pin 47 from GND.

6. Reset the board and you should see:

[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.



```
Hello World!

Vendor Name: AMBQ
Device type: Apollo3
Qualified: No
Device Info:
        Part number: 0x06671198
        Chip ID0:    0x170252D1
        Chip ID1:    0x0000000A
        Revision:    0x000ECF12 (RevA1)
        Flash size:  1048576 (1024 KB)
        SRAM size:    393216 (384 KB)

App Compiler:    IAR ANSI C/C++ Compiler V8.22.2.15995/W32 for ARM
HAL Compiler:    IAR ANSI C/C++ Compiler V8.22.2.15995/W32 for ARM
HAL SDK version: 1.2.3
```

Device: AMAPH1KK-KBR | CPUFreq: 47986 kHz | SWOFreq: 1000 kHz | 12495 bytes

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.

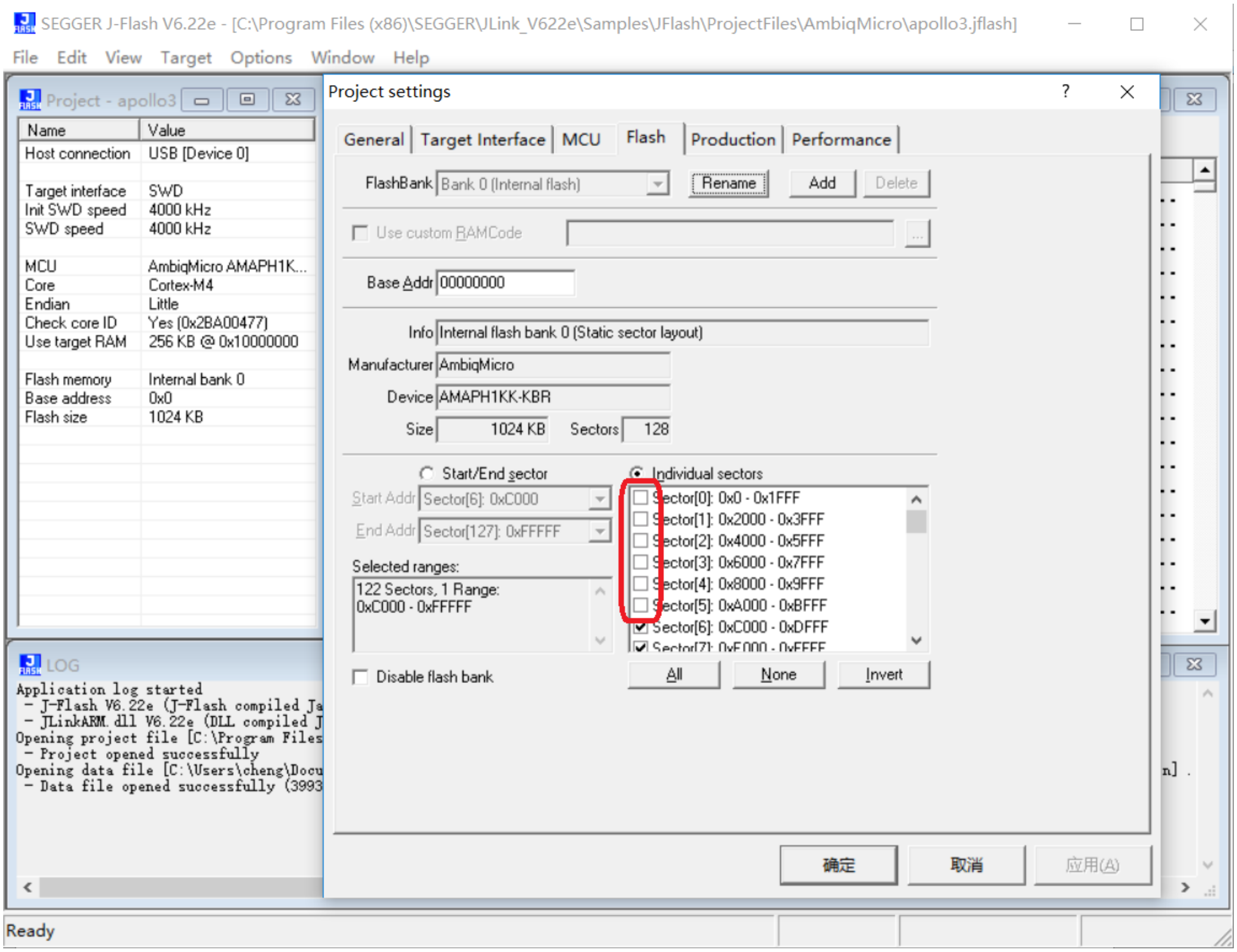
"Erase Chip" operation will not work from JFlashLite, as the pre-installed Secure Bootloader is protected, and cannot be erased. Equivalent effect could be achieved by programming all 1's to the desired sectors of the flash.

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.

Since first few sectors of the flash are reserved for SBL, deselect those sectors in the Project Settings.

An example screenshot is shown below.


SEGGER J-Flash V6.22e - [C:\Program Files (x86)\SEGGER\JLink_V622e\Samples\JFlash\ProjectFiles\AmbiqMicro\apollo3.jflash]
File Edit View Target Options Window Help
Project - apollo3
Project settings
General
Target Interface
MCU
Flash
Production
Performance
Rename
Add
Delete
...
All
None
Invert
确定
取消
应用(A)
LOG
Ready

Name	Value
Host connection	USB [Device 0]
Target interface	SWD
Init SWD speed	4000 kHz
SWD speed	4000 kHz
MCU	AmbiqMicro AMAPH1K...
Core	Cortex-M4
Endian	Little
Check core ID	Yes (0x2BA00477)
Use target RAM	256 KB @ 0x10000000
Flash memory	Internal bank 0
Base address	0x0
Flash size	1024 KB

```
Application log started
 - J-Flash V6.22e (J-Flash compiled Ja
 - JLinkARM.dll V6.22e (DLL compiled J
Opening project file [C:\Program Files
 - Project opened successfully
Opening data file [C:\Users\cheng\Docu
 - Data file opened successfully (3993
```

FlashBank: Bank 0 (Internal flash)

□ Use custom RAMCode

Base Addr: 00000000

Info: Internal flash bank 0 (Static sector layout)

Manufacturer: AmbiqMicro

Device: AMAPH1KK-KBR

Size: 1024 KB Sectors: 128

○ Start/End sector ● Individual sectors

Start Addr: Sector[6]: 0xC000

End Addr: Sector[127]: 0xFFFFF

Selected ranges:
122 Sectors, 1 Range:
0xC000 - 0xFFFFF

□ Sector[0]: 0x0 - 0x1FFF
□ Sector[1]: 0x2000 - 0x3FFF
□ Sector[2]: 0x4000 - 0x5FFF
□ Sector[3]: 0x6000 - 0x7FFF
□ Sector[4]: 0x8000 - 0x9FFF
□ Sector[5]: 0xA000 - 0xBFFF
☑ Sector[6]: 0xC000 - 0xDFFF
☑ Sector[7]: 0xE000 - 0xFFFF

□ Disable flash bank

5.4.1.1 Attach Debugger to a Running Image

At this point, you can return to the hello_world IAR project and attempt to attach the debugger (Debug without Downloading on IAR) to a running example just previously loaded using the UART wired update.

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 47 (0x2f) active low. Baudrate for INFO0 UART is set to 115200 (0x1C200). Main image is expected at 0xC000. Apollo3 is configured for UART-RX pin 49 (0x31) & UART-TX pin 48 (0x30). Secure Boot is enabled (note the –s option).

```
./create_info0.py --valid 1 info0 --pl 1 --u0 0x1C200c0 --u1 0xFFFF3031 --u2 0x2
--u3 0x0 --u4 0x0 --u5 0x0 --main 0xC000 --gpio 0x2f --version 0 --wTO 5000 –s 1
```

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):

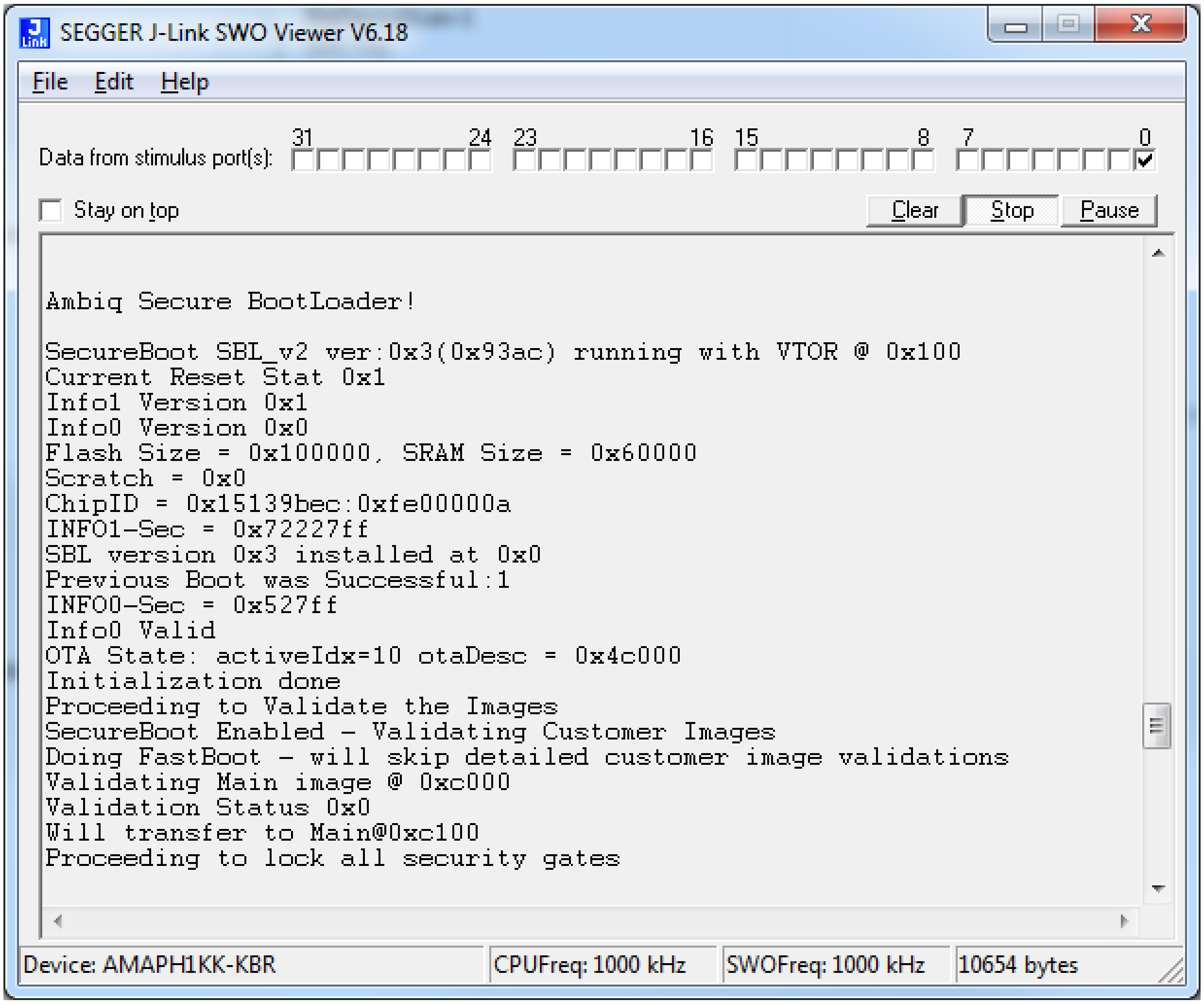
```
./create_cust_wireupdate_blob.py --load-address 0xF0000 --bin ./sbl_updates/<SBL
Binary file provided by Ambiq> -i 0 -o sbl_wire --options 0x1
```

7.2 Program SBL Upgrade Firmware

Use the GPIO override feature configurable in INFO0 (set to pin 47 – Active Low in examples above) to interrupt the boot & use the UART Wired Update script to upgrade SBL Firmware using the SBL wire update blob (generated as in section 7.1):

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

After the completion of the wired update the SWO output should show the updated SBL:



```
Ambiq Secure BootLoader!

SecureBoot SBL_v2 ver:0x3(0x93ac) running with VTOR @ 0x100
Current Reset Stat 0x1
Info1 Version 0x1
Info0 Version 0x0
Flash Size = 0x100000, SRAM Size = 0x60000
Scratch = 0x0
ChipID = 0x15139bec:0xfe00000a
INFO1-Sec = 0x72227ff
SBL version 0x3 installed at 0x0
Previous Boot was Successful:1
INFO0-Sec = 0x527ff
Info0 Valid
OTA State: activeIdx=10 otaDesc = 0x4c000
Initialization done
Proceeding to Validate the Images
SecureBoot Enabled - Validating Customer Images
Doing FastBoot - will skip detailed customer image validations
Validating Main image @ 0xc000
Validation Status 0x0
Will transfer to Main@0xc100
Proceeding to lock all security gates
```

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.