



SDK6 AN
Build Environment
(Ubuntu 14.04)

Draft Version 1.6

March 24, 2016



Confidentiality Notice:

Copyright © 2016 Ambarella, Inc.

The contents of this document are proprietary and confidential information of Ambarella, Inc.

The material in this document is for information only. Ambarella assumes no responsibility for errors or omissions and reserves the right to change, without notice, product specifications, operating characteristics, packaging, ordering, etc. Ambarella assumes no liability for damage resulting from the use of information contained in this document. All brands, product names, and company names are trademarks of their respective owners.

For full Ambarella contact information, please visit website: www.ambarella.com.

For Confidential
HAOTEX Only

I Contents

II Prefaceiii
1 Overview	1
1.1 Overview: Introduction	1
1.2 Overview: SDK6 Architecture	1
1.3 Overview: System Support Package (SSP)	2
1.4 Overview: Middleware Support Package (MSP)	3
1.5 Overview: Ambarella Reference Design (ARD)	4
1.6 Overview: Equipment Requirements	6
2 System Requirements	8
2.1 System Requirements: Overview	8
2.2 System Requirements: Windows PC	8
2.3 System Requirements: Linux Build Machine	8
2.4 System Requirements: Server Sharing with Development Team	9
3 Build Environment	10
3.1 Build Environment: Overview	10
3.2 Environment: Linux Ubuntu Setup	10
3.3 Environment: Windows Tools Setup	18
4 Purchase JTAG Debug Probe	39
4.1 Purchase JTAG Debug Probe: Overview	39
4.2 Purchase JTAG Debug Probe: Order JTAG Debug Probe	39
4.3 Purchase JTAG Debug Probe: Package of JTAG Debug Probe	40
5 Build SDK6	44
5.1 Build SDK6: Overview	44
5.2 Build SDK6: Build Ambalink SDK	44
5.3 Build SDK6: Build SSP	46
5.4 Build SDK6: Build MSP	47
5.5 Build SDK6: Build ARD	47
5.6 Build SDK6: Burn the Firmware	48

6 Customize the AmbaLink SDK	55
6.1 Customize the AmbaLink SDK: Overview	55
6.2 Customize the AmbaLink SDK: Introduction to Buildroot	55
6.3 Customize AmbaLink SDK: Modify Boot Script	55
6.4 Customize AmbaLink SDK: Remove Ambarella Add-Ons	56
6.5 Customize AmbaLink SDK: Incorporate New Add-Ons	58
6.6 Customize AmbaLink SDK: Wi-Fi Network Packages	60
6.7 Customize AmbaLink SDK: Rebuild Package	61
6.8 Customize AmbaLink SDK: Store Configurations.....	62
7 Build the Pure Linux SDK	64
7.1 Build the Pure Linux SDK: Overview	64
7.2 Build the Pure Linux SDK: Prepare the Linux Binary Image	64
7.3 Build the Pure Linux SDK: Build the Firmware.....	65
7.4 Build the Pure Linux SDK: Burn Firmware	66
8 SDK6 Toolchain and C/C++ Libraries	67
8.1 SDK6 Toolchain and C/C++ Libraries: Toolchain and C libraries.....	67
8.2 SDK6 Toolchain and C/C++ Libraries: Libraries from Toolchain Used by SDK67	
8.3 SDK6 Toolchain and C/C++ Libraries: POSIX System Call.....	67
8.4 SDK6 Toolchain and C/C++ Libraries: Newlib C library (libc.a).....	68
8.5 SDK6 Toolchain and C/C++ Libraries: Newlib C math library (libm.a).....	69
8.6 SDK6 Toolchain and C/C++ Libraries: SDK6 C++ Support and Limitation ..	69
9 Troubleshooting	72
9.1 Troubleshooting: Overview	72
9.2 Troubleshooting: Select ThreadX Toolchain	72
Appendix 1 Additional Resources	A1
Appendix 2 Important Notice	A2
Appendix 3 Revision History	A3

II Preface

This document provides technical details using a set of consistent typographical conventions to help the user differentiate key concepts at a glance.

Conventions include:

Example	Description
AmbaGuiGen, DirectUSB Save, File > Save Power, Reset, Home	Software names GUI commands and command sequences Computer / Hardware buttons
Flash_IO_control da, status, enable	Register names and register fields. For example, Flash_IO_control is the register for global control of Flash I/O, and bit 17 (da) is used for DMA acknowledgement.
GPIO81, CLK_AU	Hardware external pins
VIL, VIH, VOL, VOH	Hardware pin parameters
INT_O, RXDATA_I	Hardware pin signals
amb_performance_t amb_operating_mode_t amb_set_operating_mode()	API details (e.g., functions, structures, and type definitions)
/usr/local/bin success = amb_set_operating_mode (amb_XXX_base_address, & operating_mode)	User entries into software dialogues and GUI windows File names and paths Command line scripting and Code

Table II-1. Typographical Conventions for Technical Documents.

Additional Ambarella typographical conventions include:

- Acronyms are given in UPPER CASE using the default font (e.g., AHB, ARM11 and DDRIO).
- Names of Ambarella documents and publicly available standards, specifications, and databooks appear in *italic* type.

1 Overview

1.1 Overview: Introduction

The Ambarella Software Development Kit Version 6 (SDK6) enables the development of high-performance camera products in the sports, wearable (consumer as well as police/security) and automotive market segments. SDK6 provides the necessary software and hardware tools to enable designers to create customized, fully-featured camera products.

This document applies to multiple Ambarella product lines, including A9S, A12 and H1. Note that this list of supported chips is subject to change. Please contact an Ambarella representative with questions regarding chip compatibility.

The document is organized into the following sections.

- [\(Section 1.2\) Overview: SDK6 Architecture](#)
- [\(Section 1.3\) Overview: System Support Package \(SSP\)](#)
- [\(Section 1.4\) Overview: Middleware Support Package \(MSP\)](#)
- [\(Section 1.5\) Overview: Ambarella Reference Design \(ARD\)](#)
- [\(Section 1.6\) Overview: Equipment Requirements](#)

1.2 Overview: SDK6 Architecture

Figure 1-1 below provides a software block diagram of the SDK6.

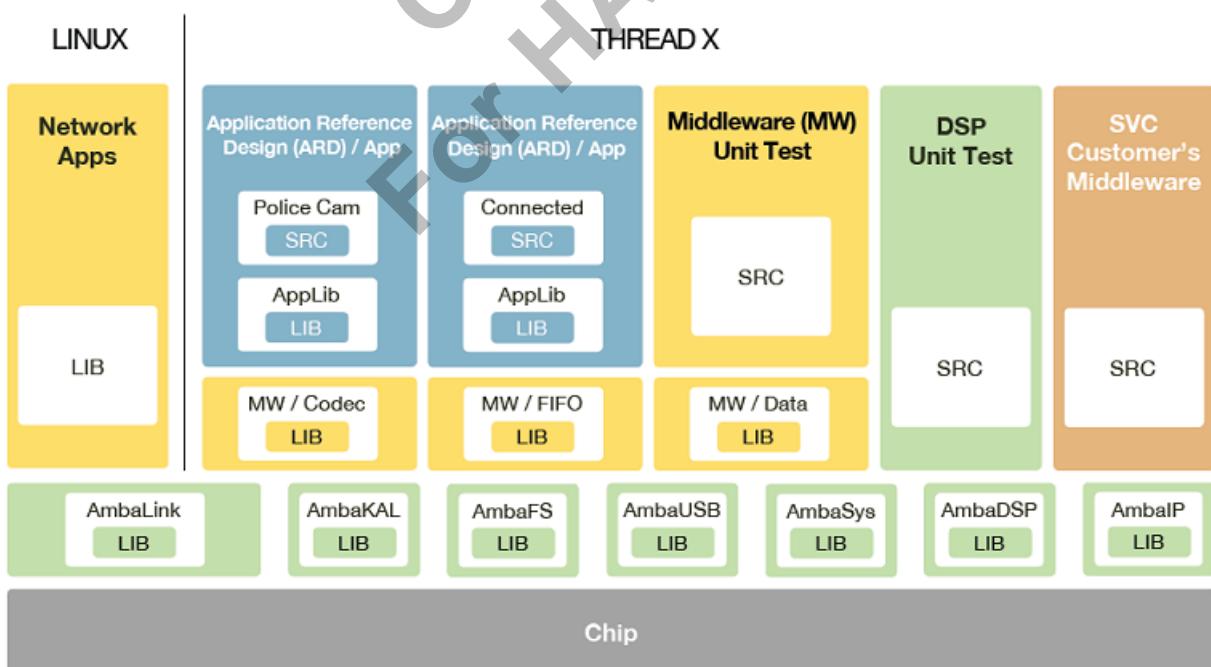


Figure 1-1. SDK6 Architecture.

The SDK6 consists of three distinct layers:

- [\(Section 1.3\) Overview: System Support Package \(SSP\)](#)
- [\(Section 1.4\) Overview: Middleware Support Package \(MSP\)](#)
- [\(Section 1.5\) Overview: Ambarella Reference Design \(ARD\)](#)

1.3 Overview: System Support Package (SSP)

The SDK6 System Support Package (SSP) provides the fundamental services used to run Ambarella chips. This package includes basic unit tests and system service code (sample muxer/demuxers, simple GUIs, file-naming rules, etc.).

Of the available SDK6 packages, the SSP enables the greatest degree of control and customization when developing new products. Customers who select the SSP package are typically competing in markets where innovation and diversification are key requirements for success. In addition, because the use of the SSP package often leads to a fuller understanding of our technologies, customers who select this package are typically engaged in longer-term, multi-product relationships with Ambarella.

It should be noted, however, that **leveraging the capabilities of the SSP requires the investment of considerable engineering resources on the part of the customer.**

For example, developing products with the SSP requires (a.) studying the source code for Unit Tests/System Service Code, (b.) understanding fundamental SSP behaviors and protocols, and (c.) developing customized software from the ground up, especially in the case of a first-time product (Ambarella typically maintains similar API interfaces across generations of SoCs; therefore, the development cycles for future products may be shortened).

The SSP layer consists of the components listed below:

- **System Libraries:**
 - AmbaKAL: RTOS Kernel Abstraction Layer
 - AmbaFS: File System
 - AmbaUSB: USB Stack
 - AmbaSys: System I/O drivers based on Ambarella chip design
- **AmbaDSP: DSP Support Package**
 - Used to control the DSP inside Ambarella chips
- **AmbaLink:** Provides RTOS and Linux communication and network support
- **AmbalP:** Ambarella reference AE/AWB/ADJ libraries
- **SSP Unit Test:** Unit tests for the SSP layer
- **SVC (System serVice Codes):** Sample application over SSP

In a typical SSP release, customers will receive the following:

1. **SSP Libraries**
2. **SSP Unit Test Source Code**
3. **SVC Source Code**
4. **Common Service Source Code:** Frequently-used small utilities
5. **Image Quality Utility Libraries:** Utilities for calibration, AE/AWB/ADJ scheduling, bitrate monitoring

1.4 Overview: Middleware Support Package (MSP)

The SDK6 Middleware Support Package (MSP) enables the full utilization of SSP capabilities via easy-to-control mechanisms, allowing customers to pursue a straightforward product development path.

Because a majority of SSP protocols are either managed or translated to simplified forms, the MSP does not require customers to understand low-level SSP protocols. Customers can create diverse features or refine existing features (demonstrated in the relevant application) from the middleware level.

It should be noted, however, that **leveraging the capabilities of the MSP requires the investment of engineering resources on the part of the customer**. Depending upon customer goals, this investment can be either:

1. Light: The customer studies the ARD/APP Applib source code to gain an understanding of how the Applib utilizes the middleware layer to implement a specific feature.
2. Heavy: The customer studies the Unit Test source code in order to learn how to use middleware APIs directly.

In either case, customers who select the MSP package will be limited by the middleware architecture and available feature set. For this reason, the MSP is typically selected by customers who are competing in markets where standardized products can be successful. If the customers are competing in markets that value differentiation and feature innovation, the SSP package may be preferable, assuming the customer is able to dedicate the necessary engineering resources.

The MSP layer consists of the components listed below:

- **MW/Codecs:** Flow controllers for video encoding/decoding/transcoding, still picture capturing/decoding, audio recording/decoding, external track (e.g., GPS information) recording/decoding, etc. This also includes arbitration mechanisms (pipelines) coordinating codes working together for multi-stream in/out and synchronization (e.g., A/V sync).
- **MW/FIFO:** Bitstream information dispatcher, which handles multiple bitstream client features.
- **MW/Data:** Data flows including muxer/demuxer/editor, cached file read/write scheduler, network transfer controller, DCF indexing (file naming) system, etc. This also includes arbitration mechanisms (pipelines) coordinating multiple muxer/demuxer instances running simultaneously.
- **MW Unit Test:** Unit tests for all middleware modules

In a typical MSP release, the customers will receive the following:

1. **SSP Libraries**
2. **MSP Libraries**
3. **Unit Test Source Code**
4. **ARD/APP Source Code** (Including its Applibs)
5. **Common Service Source Code**: Frequently-used small utilities
6. **Image Quality Utility Libraries**: Utilities for calibration, AE/AWB/ADJ scheduling, bitrate monitoring

Note that DCF (file naming rules) and GUI drawing utilities are included in the ARD/APP's Applib.

1.5 Overview: Ambarella Reference Design (ARD)

The SDK6 Ambarella Reference Design (ARD) system refers to the logically separated text section where the top-level functionality of the system exists. It is the main entrance point to the system and provides generic features for reference.

The ARD design is based on market segments, such as **Connected** (Section 1.5.1), **Police Camera** (Section 1.5.2), **CarDV** (Section 1.5.3), **Connected Multi-VIN APP** (Section 1.5.4), **MotorVu360 APP** (Section 1.5.5) and **e-Mirror** (Section 1.5.6). These ARDs/APPs share the same low-level protocols, including SSP, MSP and drivers; however, they diversify in terms of their visible feature sets. For example, the Car DV ARD includes event recording, while the Police Camera ARD does not.

Of the available SDK6 packages, the ARD system typically offers the shortest time to market. Because modularized and generic flows are both provided, only minimal engineering resources are required (e.g., to modify GUI or GUI flows) when using the ARD system, assuming the selected ARD matches the customers' product requirements exactly.

It should be noted that the **feature set of a given ARD is fixed**.

The ARD/APP layer consists of the components listed below:

- **ARD/APP Applib**: Reference code showing how to construct a specific function by using middleware APIs, such as the materials to configure a video recorder and how to control it.
 - Some middleware modules only provide frameworks, while the ARD/APP Applib includes implementations (e.g., DCF file naming rule, graphics engine).
 - By default, ARD/APP Applib is released as a library.
- **ARD/APP Source Code**: Control/UI flows for generic functions
 - While the ARD/APP Applib provides modules to configure and control middleware components, control/UI flows can be used to diversify products in terms of user experience.

In a typical ARD release, customers will receive the following:

1. **SSP Libraries**
2. **MSP Libraries**
3. **ARD/APP Source Code**
4. **ARD/APP's Applib Libraries**
5. **Common Service Source Code:** Frequently-used small utilities
6. **Image Quality Utility Libraries:** Utilities for calibration, AE/AWB/ADJ scheduling, bitrate monitoring.

1.5.1 Overview: Connected APP

Connected APP is designed to demonstrate generic functions of the Ambarella SDK, such as video encoding, still capture and playback. Connected APP provides a production-wise framework and the simplest flow required to construct a feature. Because Connected APP is not market-specific, all productions can begin from this starting point, making it easier for customers to differentiate their products in a crowded marketplace.

1.5.2 Overview: PoliceCam APP

The PoliceCam APP is designed for a specific hardware reference platform known as Libra, which includes dual switchable sensors (Main sensor: OV4689 ; Secondary attachable sensor: OV2710). The PoliceCam APP is used to demonstrate the functions of the Ambarella SDK, such as video encoding, still capture, playback, and sensor switching. The functions are implemented according to the production requirements of the police equipment segment, making it a reliable reference design for developing a production-ready police camera product in a minimal amount of time.

1.5.3 Overview: CarDV APP

The CarDV APP is designed for a specific hardware reference platform (known as Aries) and is used to demonstrate functions of the Ambarella SDK, such as video encoding, still capture and playback. Its functions were implemented according to the production requirements of the CarDV market, making it a reliable reference design for developing a production-ready automotive video product in a minimal amount of time.

1.5.4 Overview: Connected Multi-VIN APP

The Connected Multi-VIN APP is designed to demonstrate generic functions of AMBA SDK, such as video encoding, still capture and playback. Compared to Connected APP, Connected Multi-VIN APP provides the framework of multiple video inputs. It is easy for the customers to develop their product with the hardware platform have multiple video inputs.

1.5.5 Overview: MotorVu360 APP

The MotorVu360 APP is designed to demonstrate the functions of AMBA SDK, such as 360° video surround viewing, four-channel HD video encoding, and playback. The propose is to get the distortion-free views of the vehicle from every side with the hardware Motorvu360 platform. The customer can develop their product for the automotive market.

1.5.6 Overview: e-Mirror APP

The e-Mirror APP is designed to demonstrate the low latency and reliable video of AMBA SDK. The propose is to replace the vehicle's back mirrors with viewing camera. The customer can develop their product for the automotive market.

1.6 Overview: Equipment Requirements

It is recommended to use the instructions in this document when using the EVK package, PC system, and the **JTAG** debug probe emulator described in this section. For details, please refer to the following sections:

- (Section 1.6.1) Requirements: Hardware Package
- (Section 1.6.2) Requirements: PC
- (Section 1.6.3) Requirements: JTAG Debug Probe Emulator

1.6.1 Requirements: Hardware Package

The EVK package provides the following hardware and accessories.

- Main Board: A12 Dragonfly Board with an Ambarella A12_A0_RH System-on-Chip (depends on what the user actually uses)
- Sensor Board: Sensor board that connects to the Main Board for video input
- LCD Board: For video and OSD output
- Cables: Serial cable and USB cable

1.6.2 Requirements: PC

Two PCs are needed. One is typically used to communicate with the EVK hardware, download the SDK, upgrade the firmware and build the system firmware. The other PC is used for the AmbaLink SDK (please refer to Chapter 2).

1.6.3 Requirements: JTAG Debug Probe Emulator

The instructions provided in this document are for use with a **JTAG** debug probe emulator (also refer to as a debugging probe) for ARM processors.

For more details, please visit <http://www.iar.com/en/Products/Hardware-Debug-probes/>.

For Confidential
For HAOTEX Only

2 System Requirements

2.1 System Requirements: Overview

This chapter provides a summary of the hardware system requirements for the SDK6 build environment. The chapter is organized as follows:

- [\(Section 2.2\) System Requirements: Windows PC](#)
- [\(Section 2.3\) System Requirements: Linux Build Machine](#)
- [\(Section 2.4\) System Requirements: Server Sharing with Development Team](#)

2.2 System Requirements: Windows PC

A Windows development environment is typically used to communicate with the EVK hardware, download the SDK, and upgrade the firmware or the firmware for debugging.

- CPU: 4th Generation Intel Core i5-4570 Processor (3.2 GHz, 6 M, 84 W)
- Memory: 4 GB (1x4 GB) 1600 MHz DDR3 Non-ECC
- Hard Disk: 1TB 3.5inch Serial ATA (7.200 Rpm) Hard Drive
- Video Card: ATI Radeon HD 3450 or 3D hardware accelerator card required – 100% DirectX 9.0c compatible
- Monitor: 24-inch LCD or larger with 15-pin D-subminiature (D-sub) input
- Ethernet Card: 1000 Mbps (adaptive)
- I/O interfaces: RS-232 Standard
- Operating System: Windows 7 (32-bit or 64-bit)

2.3 System Requirements: Linux Build Machine

A Linux development environment is typically used in the AmbaLink SDK. Hardware specifications for the Linux build environment are included below.

- CPU: 64-bit capable, Intel® Xeon® E5-26xx-v2 15-MB cache x 2
- Memory: 32-GB Memory (8 x 4 GB), 1333 MHz, Dual-Rank RDIMM registered memory modules for two processors

- Hard Disk: 600 GB, 3.5-inch, 15K RPM, 6 Gbps, SAS Hot-Plug Hard Drive
- Ethernet Card: Dual-port Gigabit Ethernet
- Operating System: Ubuntu 14.04

When installing Linux, please ensure there is no data present on the machine, as all data will be erased during the installation process. Note that the use of a virtual machine is not recommended as it lengthens the SDK build time. When building the Ambalink SDK on a virtual machine, at least 1.5 GB of RAM and at least 15 GB of disk space is required.

2.4 System Requirements: Server Sharing with Development Team

If multiple developers are working on the Ambalink SDK, Ambarella suggests sharing the Ubuntu 14.04 LTS installation on a power workstation to the PCs belonging to the individual developer. Developer A, B and C can connect to the build server using secure shell (SSH) protocols with a LAN (Ethernet A).

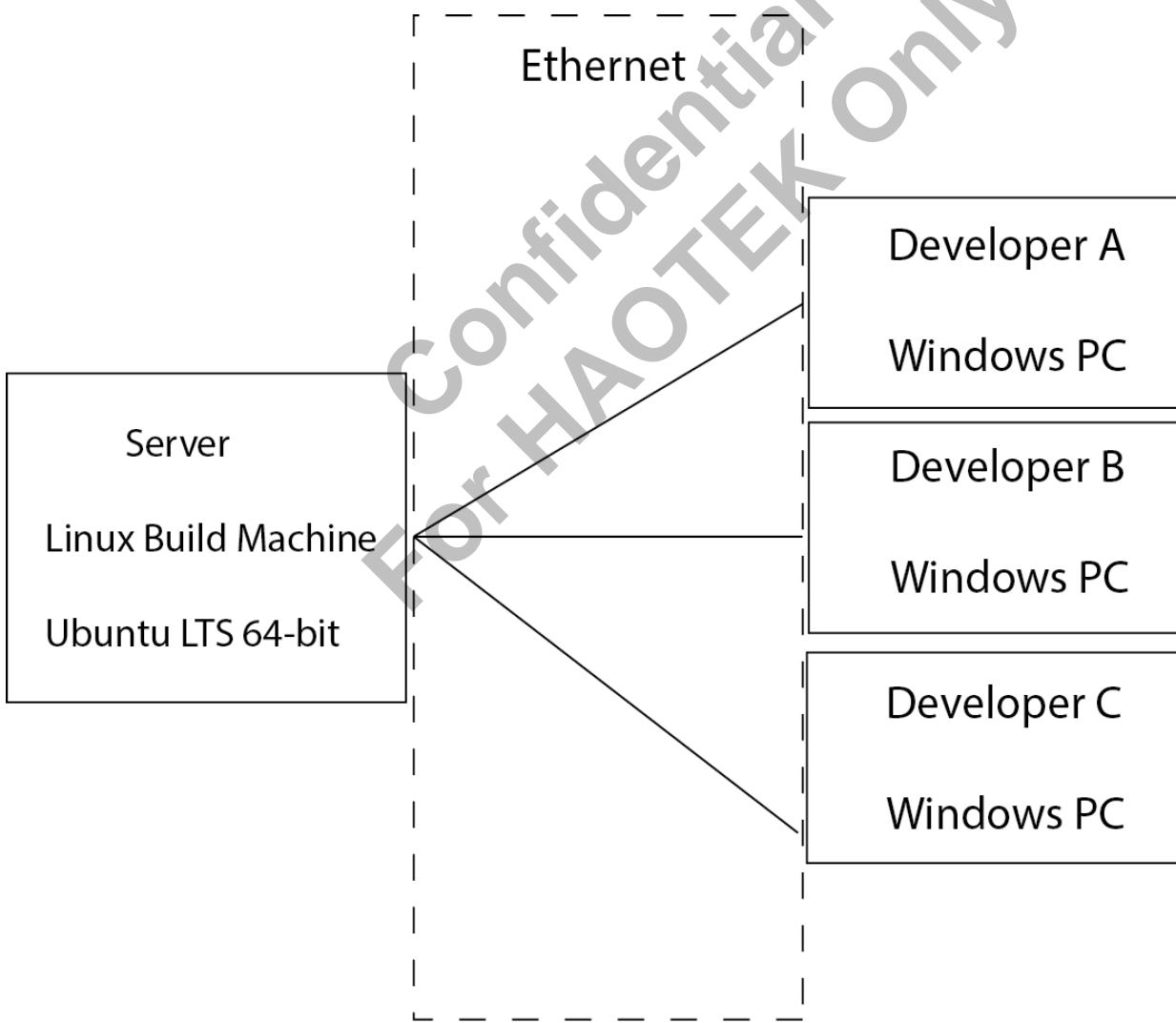


Figure 2-1. Topology for Development Team with Server Sharing.

3 Build Environment

3.1 Build Environment: Overview

This chapter provides setup instructions for the Windows and the Linux software build environments. Please refer to [Chapter 2 “System Requirements”](#).

The chapter is organized as follows:

- [\(Section 3.2\) Environment: Linux Ubuntu Setup](#)
- [\(Section 3.3\) Environment: Windows Tools Setup](#)

3.2 Environment: Linux Ubuntu Setup

This section provides the step-by-step installation instructions for the Linux Ubuntu 14.04 LTS release (64-bit version), including the following:

- [\(Section 3.2.1\) Linux Ubuntu: Download Ubuntu 14.04 LTS 64-bit](#)
- [\(Section 3.2.2\) Linux Ubuntu: Burn ISO File onto CD](#)
- [\(Section 3.2.3\) Linux Ubuntu: Install Ubuntu](#)
- [\(Section 3.2.4\) Environment: Configure Linux Server](#)
- [\(Section 3.2.5\) Environment: Samba Setup in Linux](#)
- [\(Section 3.2.6\) Environment: Linux Buildroot Toolchain](#)

3.2.1 Linux Ubuntu: Download Ubuntu 14.04 LTS 64-bit

ISO image files for the Linux Ubuntu 14.04 LTS release can be downloaded from the official Ubuntu website at <http://www.ubuntu.com/>. 64-bit support is recommended.

3.2.2 Linux Ubuntu: Burn ISO File onto CD

Create a CD to install Ubuntu after downloading the ISO file.

There are a number of CD-burning tools such as UltraISO, Nero Burning ROM and InfraRecorder. For tool recommendations:

1. Refer to the Ubuntu website: <http://www.ubuntu.com/>.
2. Scroll down to **2. Burn Your CD or Create a USB Drive** and click **Show Me How**.

2

Burn your CD or create a USB drive

Once you've finished
downloading your file, you'll
need to create a CD or USB
drive.

I would like to create a:

CD USB stick

To create it, I will be using:

Windows Mac Ubuntu

Show me how

Figure 3-1. Linux Burn Ubuntu ISO File.

3.2.3 Linux Ubuntu: Install Ubuntu

1. Prepare the computer for installation. Note the following:
 - A 64-bit system is recommended.
 - The installation process will erase all data. Please ensure that the system contains no operating system and no data.
 - If a virtual machine is used, at least 1.5 GB of RAM and at least 15 GB of disk space are required. The use of a virtual machine is not recommended as it lengthens the SDK build time.
2. Place the CD containing the Ubuntu ISO file in the drive and restart the machine. Verify that the BIOS configuration and the machine settings are consistent with a boot from the CD ROM.
3. From the initial installation dialog, select the preferred **Language**.
4. From the Ubuntu installation screen, select **Install Ubuntu Server**.



Figure 3-2. Linux Install Ubuntu Start Screen.

5. Follow the on-screen instructions. **Default** settings are typically appropriate.
6. At the **Partition Disks** prompt, **Remove existing logical volume data** (select **Yes**).

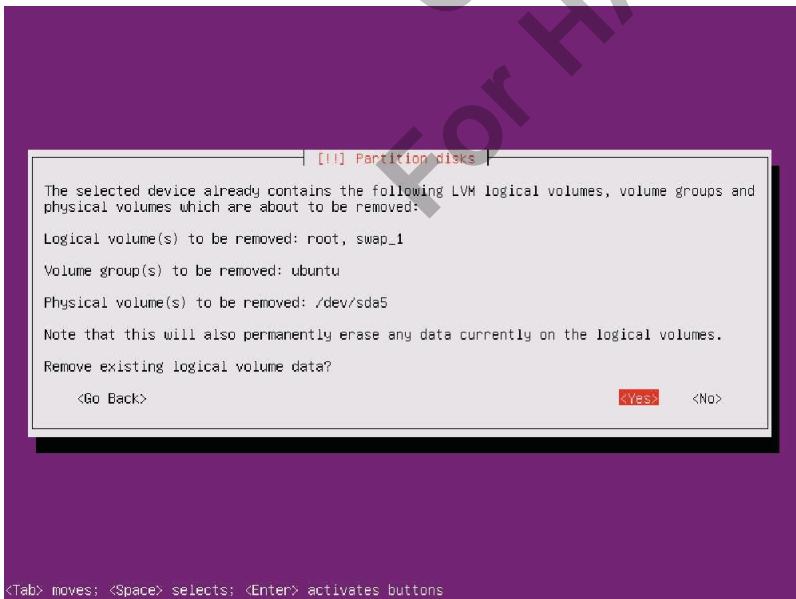


Figure 3-3. Linux Ubuntu Installation Partition Disks.

- At the **Write the Changes to Disks and Configure LVM** prompt, select **Yes**.

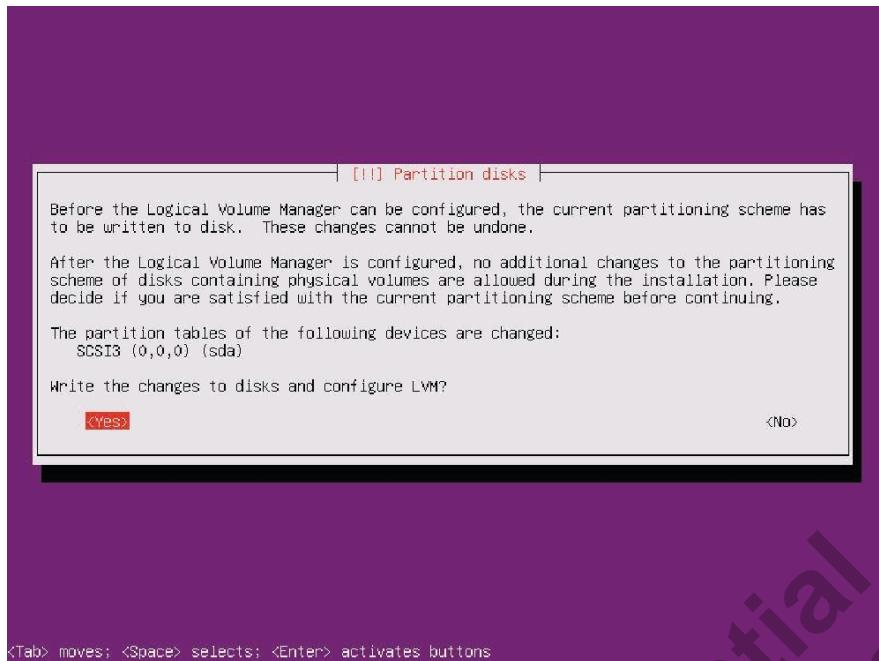


Figure 3-4. Linux Install Ubuntu Change Disks and Configure LVM.

- At the **Write the Changes to Disks** prompt, select **Yes**.

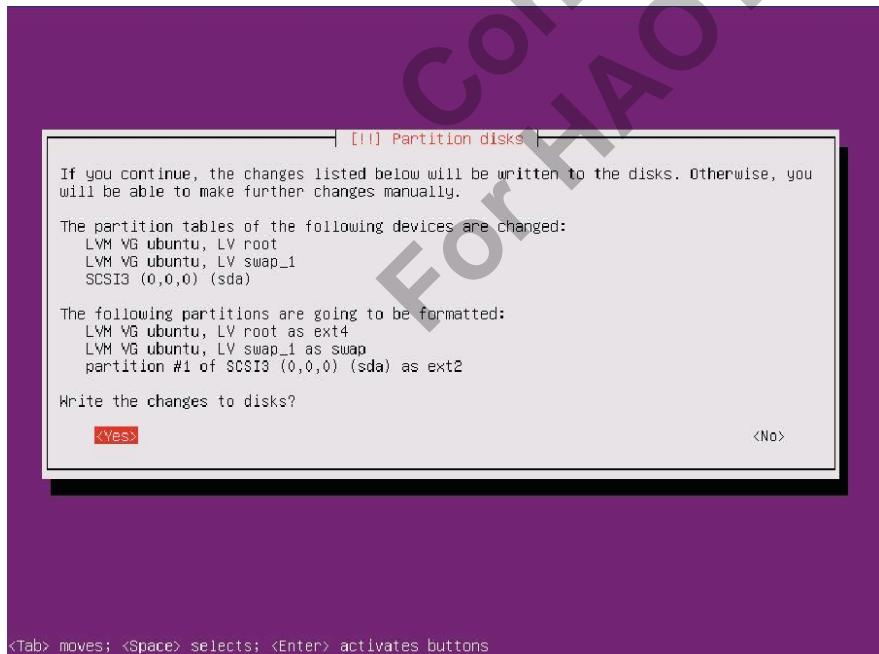


Figure 3-5. Linux Install Ubuntu Partition Disks Write Changes.

- Configure **Network Settings** as appropriate.

10. At the **Software Selection** prompt, select **Open SSH Server** and **Samba File Server** and continue.

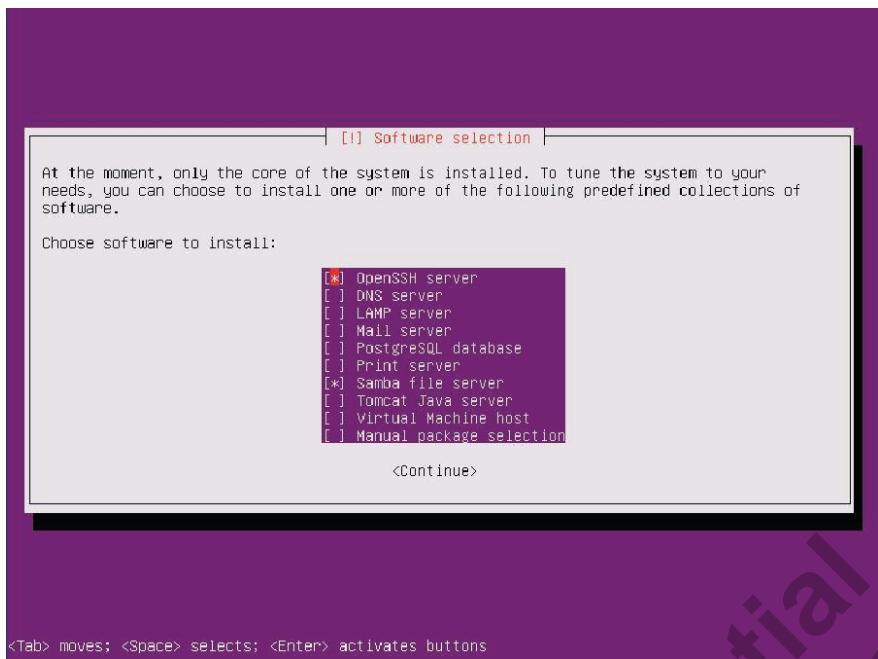


Figure 3-6. Linux Ubuntu Installation **Samba** File Server Select.

11. At the GRand Unified Bootloader (GRUB) or Configuring GRUB-PC prompt, select **Yes**.

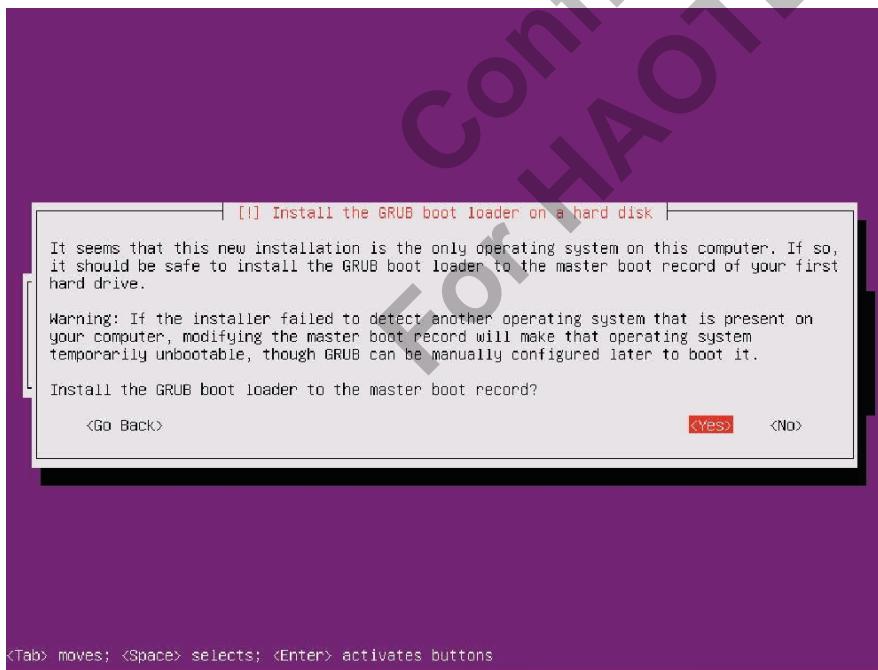


Figure 3-7. Linux Ubuntu Install or Configure GRUB-PC.

12. The following screen indicates that the installation is complete.

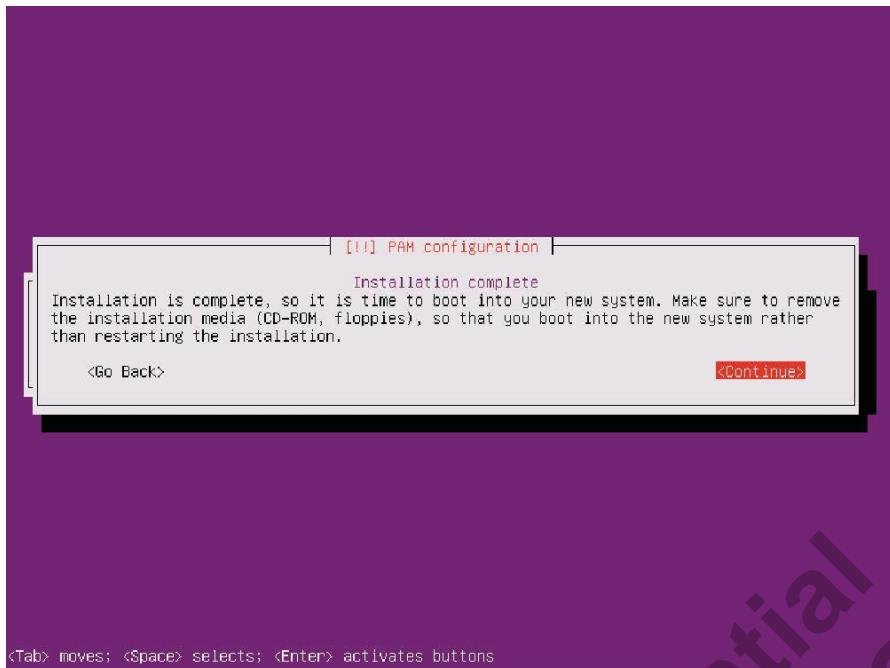


Figure 3-8. Linux Ubuntu Install Complete.

3.2.4 Environment: Configure Linux Server

The following sections describe how to configure and update the Linux server:

- ([Section 3.2.4.1](#)) Linux Server: Configure Server Network
- ([Section 3.2.4.2](#)) Linux Server: Update Server Packages
- ([Section 3.2.4.3](#)) Linux Server: Specify Bison (GNU Parser Generator) Revision

3.2.4.1 Linux Server: Configure Server Network

1. Use the following command to determine whether the network is functional or not.

```
$ ping www.google.com      (or other website, such as: www.ubuntu.com )
```

2. Configure the network as DHCP or Static IP.
3. Login to the sever and configure the network according to the machine environment.

IP Address:	<code>\$sudo vim /etc/network/interfaces</code>	(edit files ...)
DNS server:	<code>\$sudo vim /etc/resolv.conf</code>	(edit files ...)
Restart networking:	<code>\$sudo /etc/init.d/networking restart</code>	

4. Repeat Step 1 to verify the functionality of the network.

3.2.4.2 Linux Server: Update Server Packages

Use the following commands to update the server packages.

```
$ sudo apt-get install git-core gnupg flex bison gperf build-essential zip curl  
zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 lib32ncurses5-dev  
x11proto-core-dev libx11-dev gawk gettext texinfo subversion dos2unix tofrodos  
lib32z1 lib32bz2-1.0
```

3.2.4.3 Linux Server: Specify Bison (GNU Parser Generator) Revision

```
$ sudo apt-get remove bison  
$ curl -O http://ftp.gnu.org/gnu/bison/bison-2.5.1.tar.gz  
$ tar -xzvf bison-2.5.1.tar.gz  
$ cd bison-2.5.1  
$ sudo ./configure  
$ sudo make  
$ sudo make install  
$ sudo ln -s /usr/local/bin/bison /usr/bin/bison  
$ cd ..  
$ sudo rm -fr bison-2.5.1*
```

3.2.5 Environment: Samba Setup in Linux

Setup of the AmbaLink SDK requires **Samba** to be installed on the Linux machine. **Samba** is an open-source software suite that provides seamless file and print services to SMB/CIFS clients. **Samba** is available free, unlike other SMB/CIFS implementations, and allows for interoperability between Linux/Unix servers and Windows-based clients. The Samba configuration is specific to a user's Intranet and is beyond the scope of this document. The reader is referred to the Linux Ubuntu user forum for additional details.

3.2.6 Environment: Linux Buildroot Toolchain

This section explains the Linux Toolchain in the environment. Ambarella will provide the related toolchains.

3.2.6.1 Environment: Toolchain for Linux

The AmbaLink Linux SDK uses a Buildroot build system. The Buildroot system includes a set of makefiles that simplify and automate the Linux system build process through the use of cross-compilation toolchains. These toolchains are capable of creating a root file system, compiling a Linux kernel image, and generating a boot loader.

The AmbaLink Buildroot build system will generate all required cross-compilation toolchains for the AmbaLink SDK. Please refer to [Chapter 5 “Build SDK6”](#) for additional information on the AmbaLink Buildroot build system.

Linux DTB (device tree blob) can be built either by Buildroot while building the linux kernel or ambarella/bootloader/boards/xxxx, which is an independent build from the Buildroot System.

While using ambarella/bootloader/boards/xxx to build DTB, a bare-metal toolchain is needed. It uses arm-none-eabi-gcc for 32 bit linux or aarch64-elf-gcc for 64 bit linux. Please check “[\(Section 3.2.6.2\) Environment: Toolchain for ThreadX](#)” for the toolchain installation.

3.2.6.2 Environment: Toolchain for ThreadX

This section explains the following:

- (Section 3.2.6.2.1) Toolchain for ThreadX: Toolchain for 32 bit ThreadX
- (Section 3.2.6.2.2) Toolchain for ThreadX: Toolchain for building fwprog64

3.2.6.2.1 Toolchain for ThreadX: Toolchain for 32 bit ThreadX

32 bit ThreadX toolchain is chosen from the linaro Bare-metal toolchain, which can be downloaded from <https://launchpad.net/gcc-arm-embedded>.

Ambarella SDK could upgrade the support for newer toolchain.

User can download the toolchain from website or Ambarella could provide the current used toolchain revision for user to setup the environment.

Ambarella could provide the related toolchain file in a tar format file (please refer to the format file below).

Place the following tar file in /usr/local/ and unzip them.

gcc-arm-none-eabi-4_9-2015q1-20150306-linux.tar.bz2 [project dependent]
gcc-arm-none-eabi-4_9-2015q3-amba-20160309-20150921-linux.tar.bz2 [project dependent]

The following command may be used to unzip a tar file.

```
$ sudo tar -xvf <filename>.tar.gz or  
$ sudo tar -jxf <filename>.tar.bz2 (based on the actual file format)
```

For example:

```
$ sudo tar -jxf gcc-arm-none-eabi-4_9-2015q1-20150306-linux.tar.bz2  
(This file is provided by Ambarella or downloaded from https://launchpad.net/gcc-arm-embedded)  
$ sudo chmod -R 755 gcc-arm-none-eabi-4_9-2015q1
```

Or

```
$ sudo tar -jxf gcc-arm-none-eabi-4_9-2015q3-amba-20160309-20150921-linux.tar.bz2  
(This file is provided by Ambarella or downloaded from https://launchpad.net/gcc-arm-embedded)  
$ sudo chmod -R 755 gcc-arm-none-eabi-4_9-2015q3-amba-20160309
```

If there is another pre-installed toolchain, and the user wants to know how to select pre-installed toolchain in the user's build environment, please refer to (Section 9.2) Troubleshooting: Select ThreadX Toolchain for more details.

3.2.6.2.2 Toolchain for ThreadX: Toolchain for building fwprog64

64 bit ThreadX toolchain is chosen from the linaro Bare-metal toolchain, which can be downloaded from <http://www.linaro.org/downloads/>.

Ambarella SDK could upgrade the support for newer toolchain.

The user can download the toolchain from website or Ambarella could provide the current used toolchain revision for the users to setup the environment.

Ambarella could provide the related toolchain file in a tar format file (please refer to the format file below). Place the following tar file in /usr/local/ and unzip them.

```
gcc-linaro-4.9-2015.02-3-x86_64_aarch64-elf.tar.xz
```

The following command may be used to unzip a tar file.

```
$ sudo tar -xvf <filename>.tar.xz
```

For example:

```
$ sudo tar -xvf gcc-linaro-4.9-2015.02-3-x86_64_aarch64-elf.tar.xz  
(This file is provided by Ambarella or downloaded from http://www.linaro.org/downloads/)  
$ sudo chmod -R 755 gcc-linaro-4.9-2015.02-3-x86_64_aarch64-elf
```

3.2.7 Environment: Setup User Profiles

The Ambarella-provided file (user-profile.tgz) should be uncompressed. With a build machine, please ensure that the environment scripts resident in the supplied user-profile.tgz are executed in the personal build environment. Also (1) copy the unzip file .bash_profile to the user home directory, (2) log out and (3) log in again.

```
$ cp user-profile.tgz ~  
$ tar zvfx user-profile.tgz (This file is provided by Ambarella)  
$ rm user-profile.tgz
```

Next, please log out and log in again.

3.3 Environment: Windows Tools Setup

This section provides step-by-step setup instructions for Windows tools that include **PuTTY** and **Signum Systems Chameleon Debugger** and **CVD64 Debugger**. It includes the following sections:

- ([Section 3.3.1](#)) Windows: PuTTY Setup
- ([Section 3.3.2](#)) Windows: Chameleon Setup
- ([Section 3.3.3](#)) Windows: CVD64 Setup

3.3.1 Windows: PuTTY Setup

This section includes basic instructions for installing and configuring **PuTTY**, an open-source serial console and terminal emulation application. The free SSH and telnet client **PuTTY** (or a package with similar functionality) is required in order to enable telnet/SSH communications with the Main Board UART module.

For download information, please visit www.putty.org.

1. Run **putty.exe**.
2. To enable a serial connection, configure **PuTTY** as follows ([Figure 3-9](#) and [Figure 3-10](#)):

Speed (baud rate in bits per second): 115200
Data bits: 8

Stop bits: 1
Parity: None
Flow Control: None

3. Connect the RS-232 serial cable included with the EVK. If the cable is not available, a female-to-female RS-232 RX/TX crossover cable should be used.

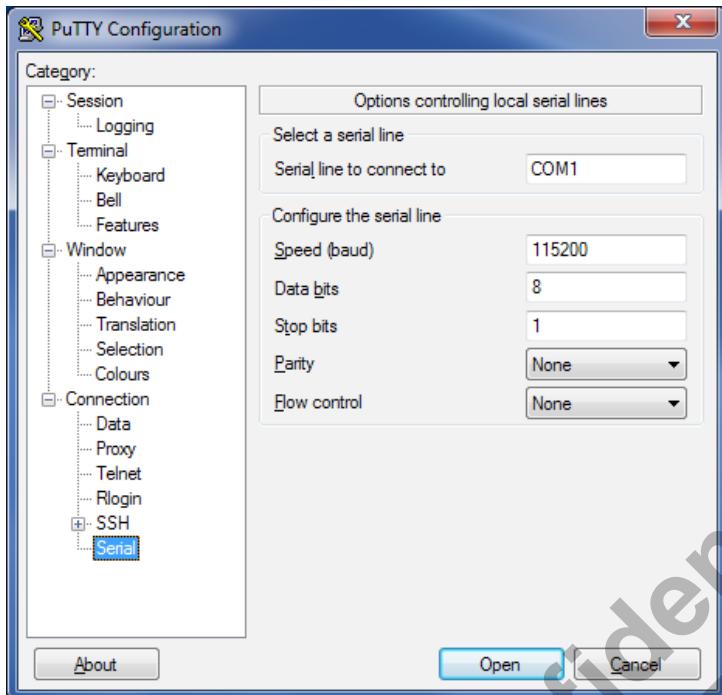


Figure 3-9. **PuTTY Configuration - Serial Control.**

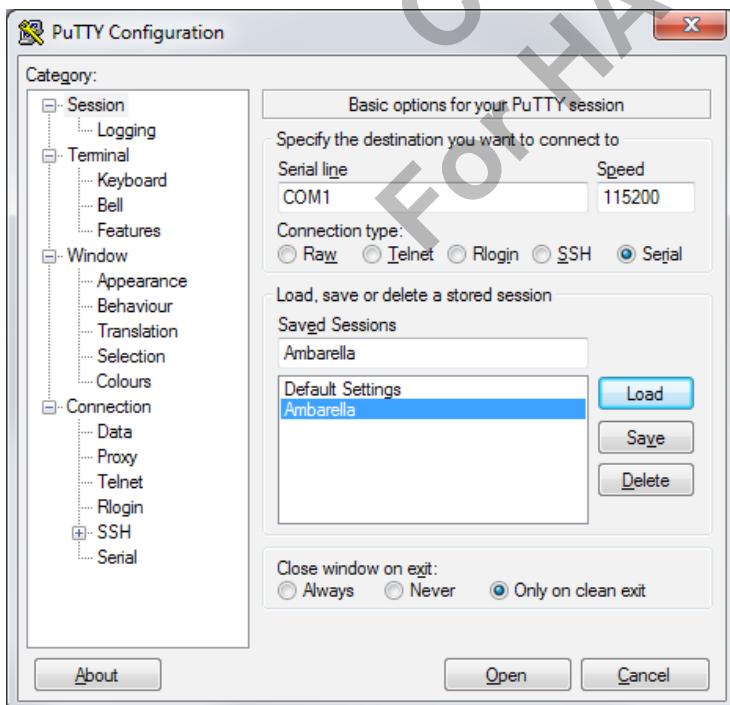


Figure 3-10. **PuTTY Configuration - Basic.**

3.3.2 Windows: Chameleon Setup

This section provides installation and configuration instructions for the Signum Systems **Chameleon** Debugger.

3.3.2.1 Chameleon: Installation

1. For I-jet purchased before June 2015, just double-click the Chameleon Debugger setup executable (setup_charm.exe) whose version is 2.99.37 SP5, and skip Step 2. For those shipped after June 2015, please use Chameleon version 2.99.41 and complete Step 2 before launching the installer.
2. Please contact with Ambarella and provide one's I-jet serial number which is listed in a sheet shipped together with I-jet. Ambarella will provide the following files and please put them on the right paths:
 - Chameleon.lic: Please put this file under C:\Signum\Chameleon. Overwrite if it already exists.
 - J<your serial number>.lic: For example, J12345.lic, and 12345 is one's serial number of I-jet. Please put it under C:\Signum\Licenses.
3. When the **Setup Wizard** opens, click **Next** to continue.

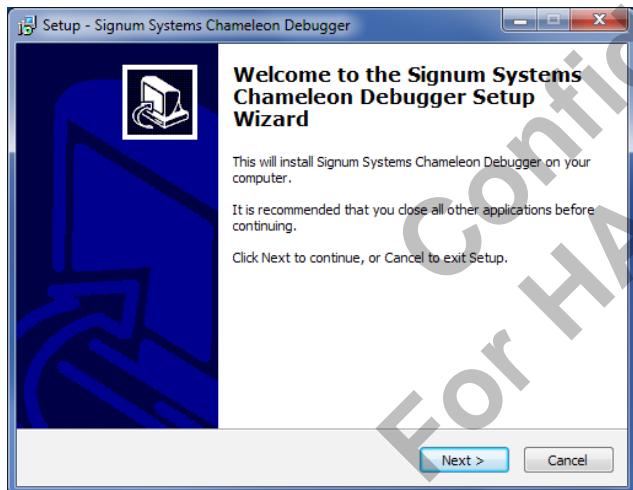


Figure 3-11. **Signum System Chameleon Debugger Setup Wizard.**

4. From the **License Agreement** screen, choose **I accept the agreement** (radio button) and click **Next** to continue.



Figure 3-12. **Chameleon Debugger Installation: License Agreement.**

5. Read the Information screen, and click **Next**.

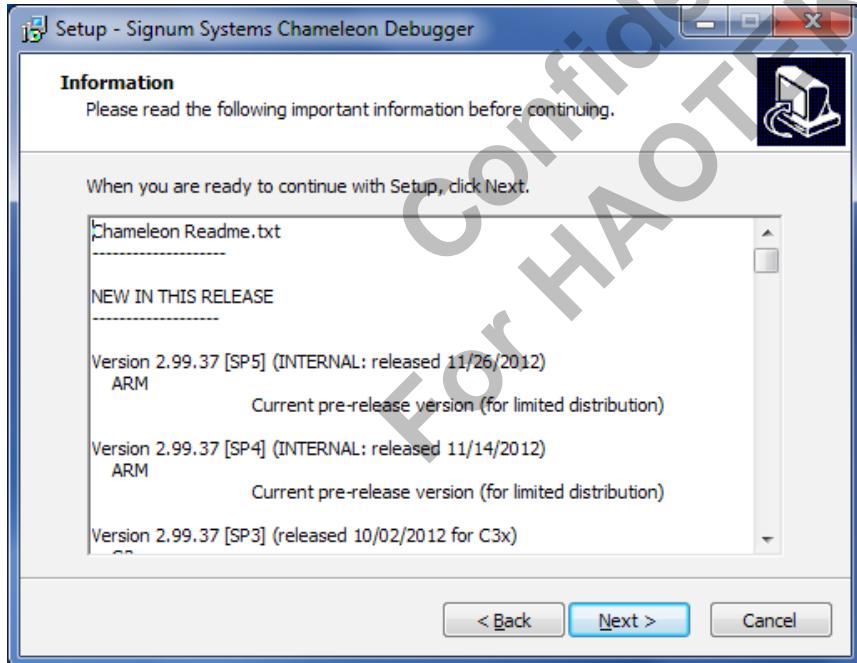


Figure 3-13. **Chameleon Debugger Installation: Information.**

6. From the **Select Destination Location** screen, either accept the default location or click **Browse** and select a different folder. Click **Next** to continue.

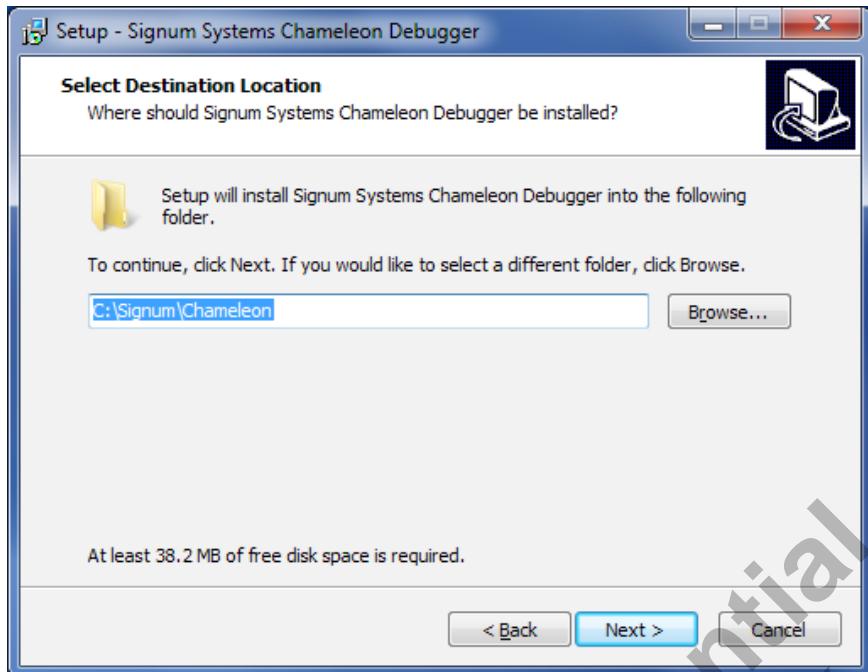


Figure 3-14. **Chameleon Debugger Installation: Select Destination Location.**

7. From the **Select Start Menu Folder** screen, either accept the default location or click **Browse** and select a different folder. Click **Next** to continue.

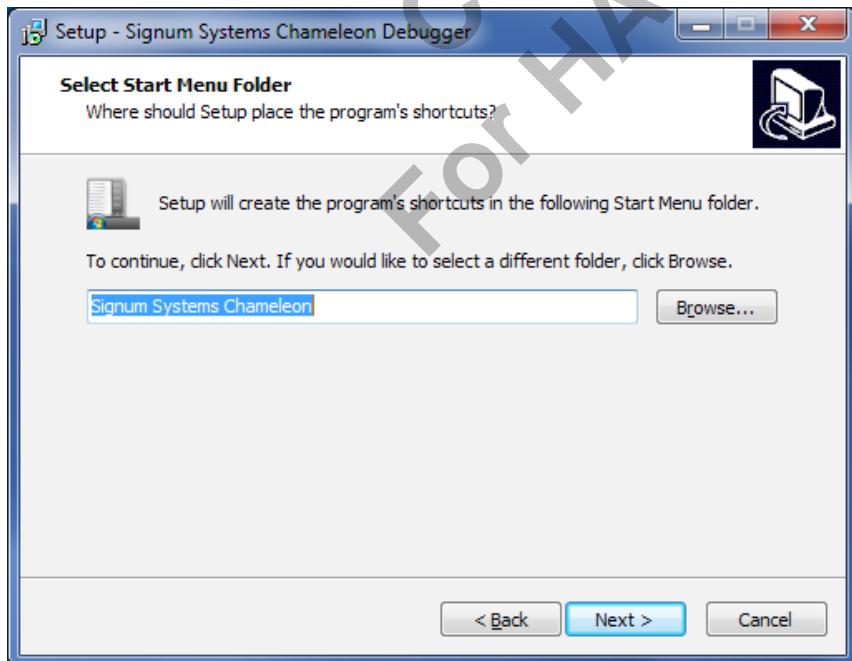


Figure 3-15. **Chameleon Debugger Installation: Select Start Menu Folder.**

8. From the **Select Additional Tasks** screen, choose whether to add a desktop icon or not and click **Next**.

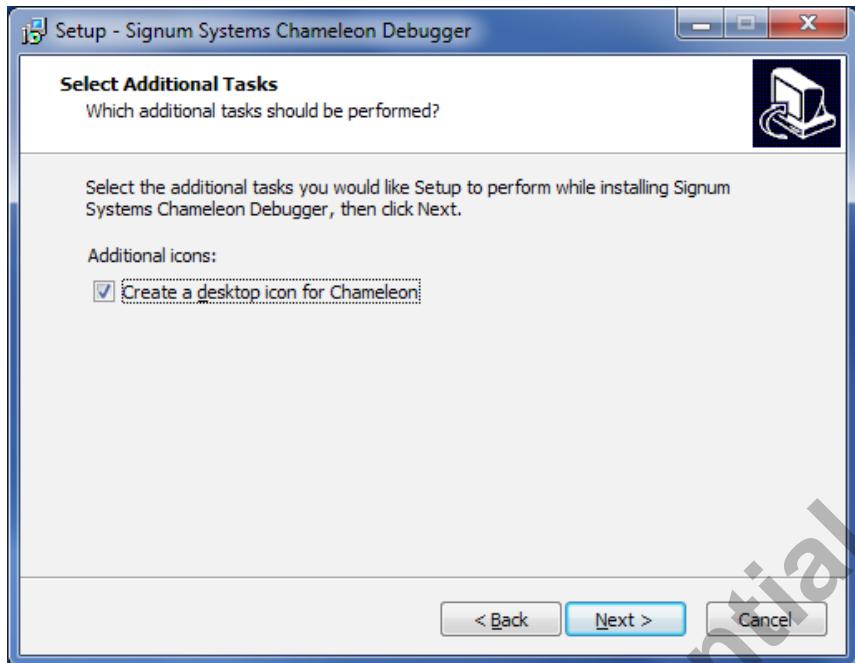


Figure 3-16. **Chameleon Debugger Installation:** **Select Additional Tasks.**

9. When the **Ready to Install** screen appears, click **Install** to begin.

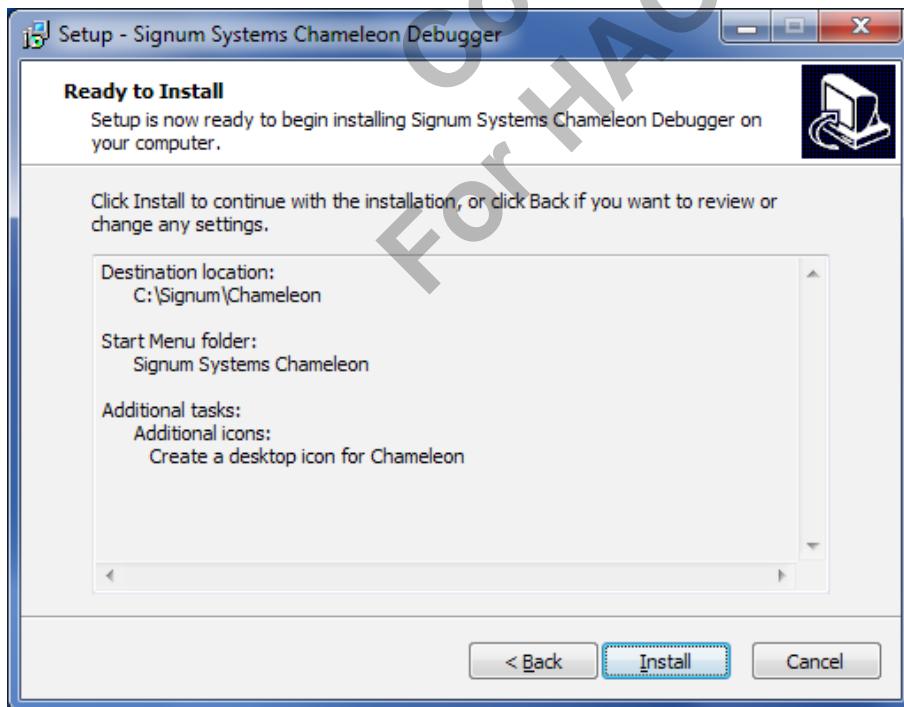


Figure 3-17. **Chameleon Debugger Installation:** **Ready to Install.**

10. The installation should complete successfully. Click **Finish** to exit the Setup Wizard.

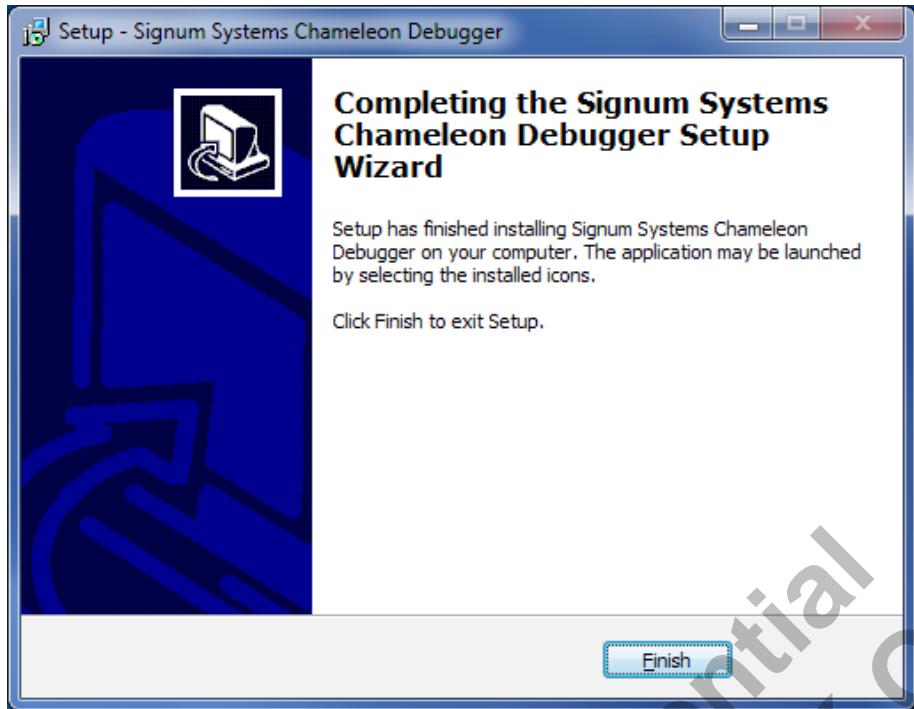


Figure 3-18. **Chameleon Debugger Installation:** *Finish*.

- If the screen below appears, this is an indication that the **Chameleon Debugger** has not been previously installed on the PC or laptop. If this occurs, click **OK** to continue.

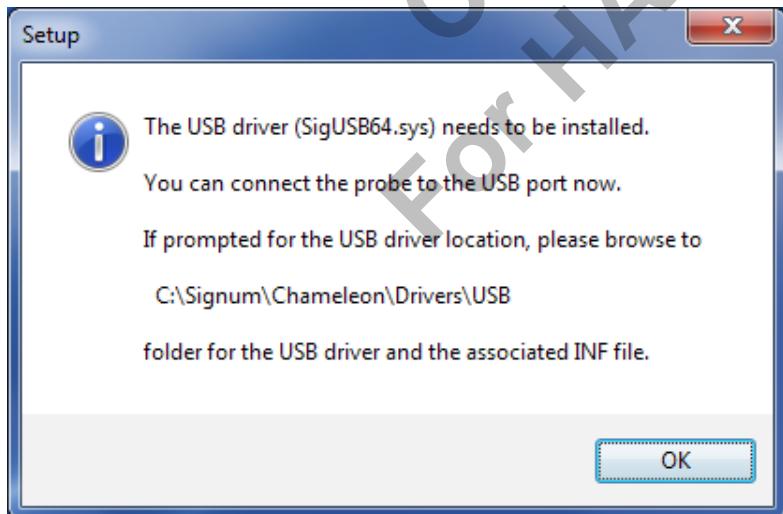


Figure 3-19. **Chameleon Debugger Installation:** *USB Driver Setup*.

3.3.2.2 Chameleon: Setup

1. Connect the **JTAG** debug probe emulator, EVK board, and PC as follows:
 - Connect the power cable and the **JTAG** debug probe emulator to the EVK board.
 - Connect the **JTAG** debug probe emulator to the PC via the USB cable.
 - Press the power key on the main board to power-on the board.
2. Copy the Ambarella files required for use with the **Chameleon Debugger**, please obtain the released SDK from Ambarella.
Navigate to the `rtos\tools\soc\Chameleon\Ambarella-A12\`, copy these files in the folder, and then paste them to `C:\Signum\Chameleon` (or the user-selected path) by replacing the original files/folder.
3. Click the **Chameleon** executable (red lizard desktop icon) and navigate to the Main Tool Bar, select the View menu and choose **System Configuration**. In the **System Configuration** dialog that appears, click **Add Target**.

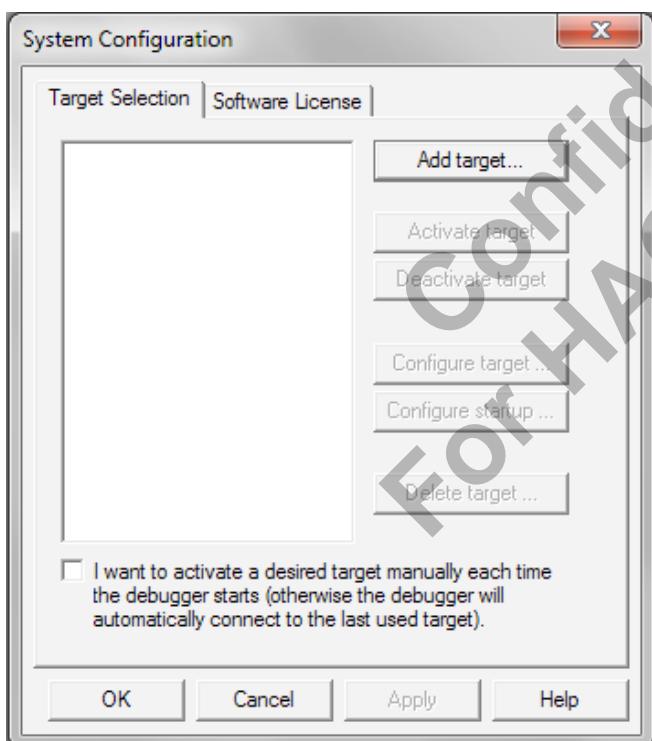


Figure 3-20. **Chameleon Setup: System Configuration.**

4. In the **Target Selection** dialog, for **Target Name**, key in A12 and click **OK**. (Here Ambarella uses A12SDK as an example)

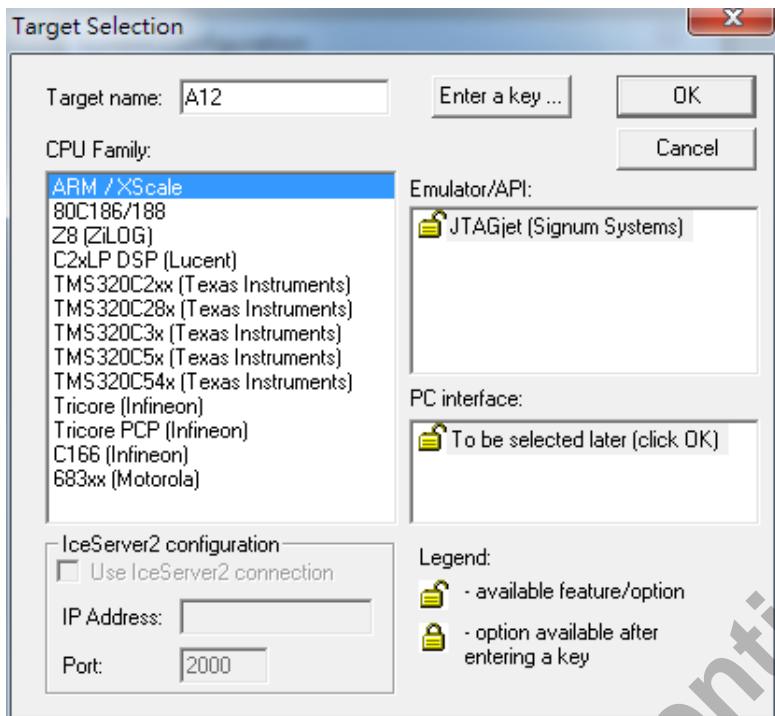


Figure 3-21. **Chameleon Setup: Target Selection.**

5. When the **About to Connect to an Emulator** dialog appears, select **Connect to an Emulator Automatically** and click **Next**.

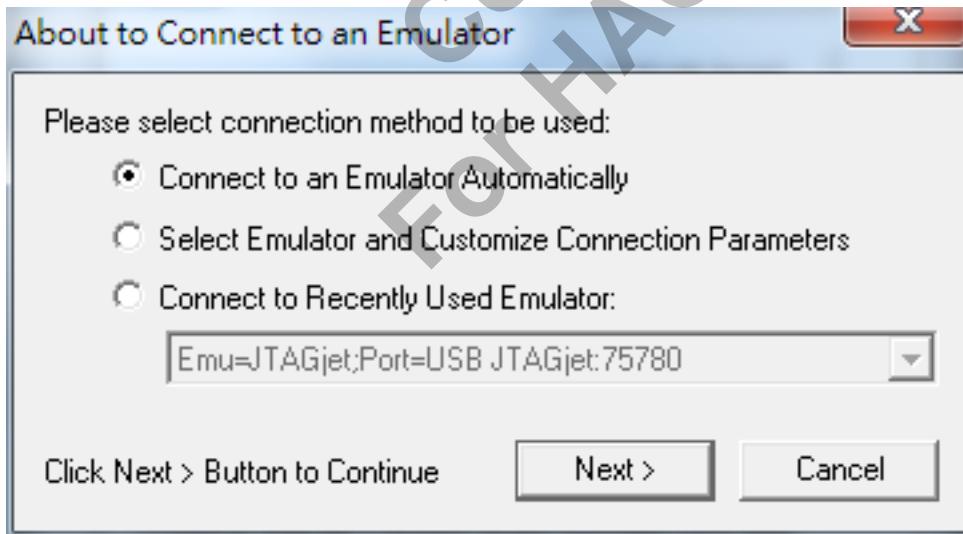


Figure 3-22. **Chameleon Setup: About to Connect to an Emulator.**

6. In the **Startup Configuration Selection** dialog that appears, choose **My board is not listed – I want to create my own board configuration** and click **OK**.

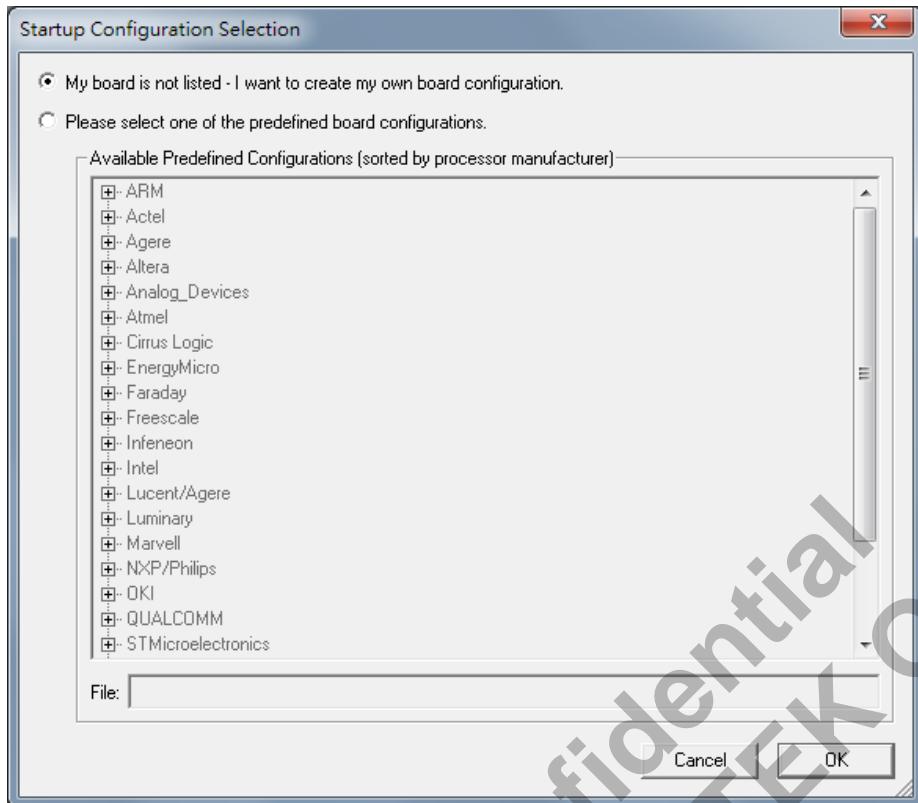


Figure 3-23. **Chameleon Setup: Startup Configuration Selection.**

7. A CPU/ARM core selection list will appear, choose the **ARM Cortex cores – Cortex-A9 core**, and click **OK**.

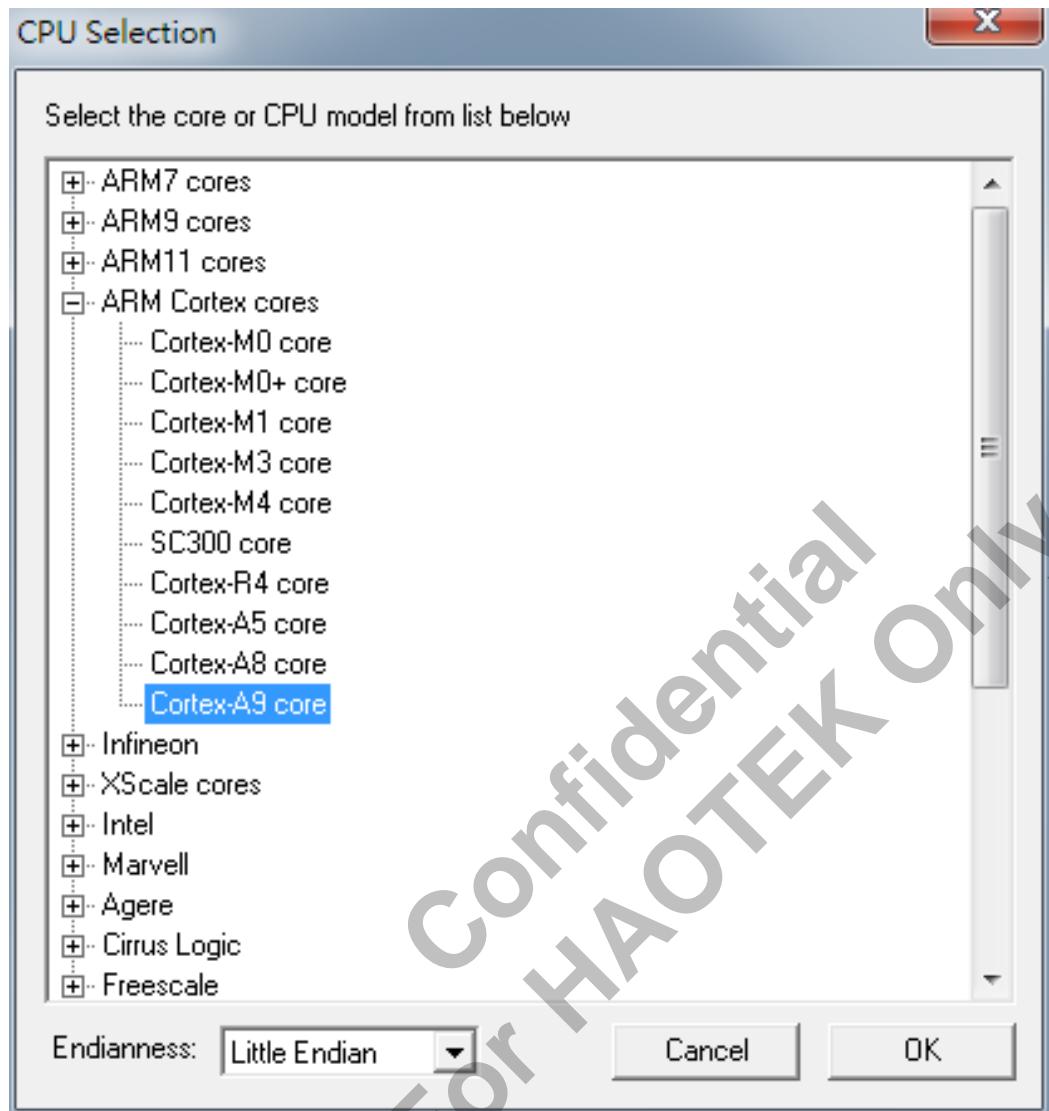


Figure 3-24. **Chameleon Setup: Stop the CPU.**

8. Choose “Use JTAG Chain Configuration File”, and click “Browse”
Select C:\Signum\Chameleon\AmbarellaA12.cfg, and click OK.

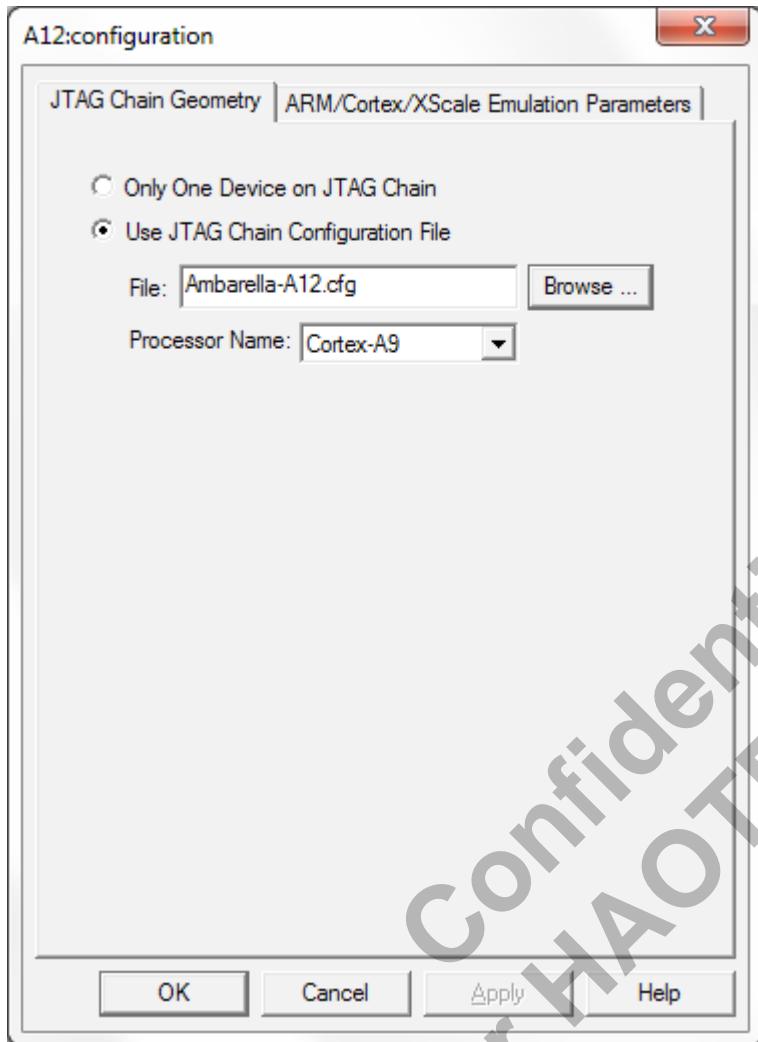


Figure 3-25. **Chameleon Setup:** Choose View > **System Configuration**.

9. Click on the right tab to select the **ARM/Cortex/XScale Emulation Parameters**. This tab allows the user to choose the JTAG clock speed.

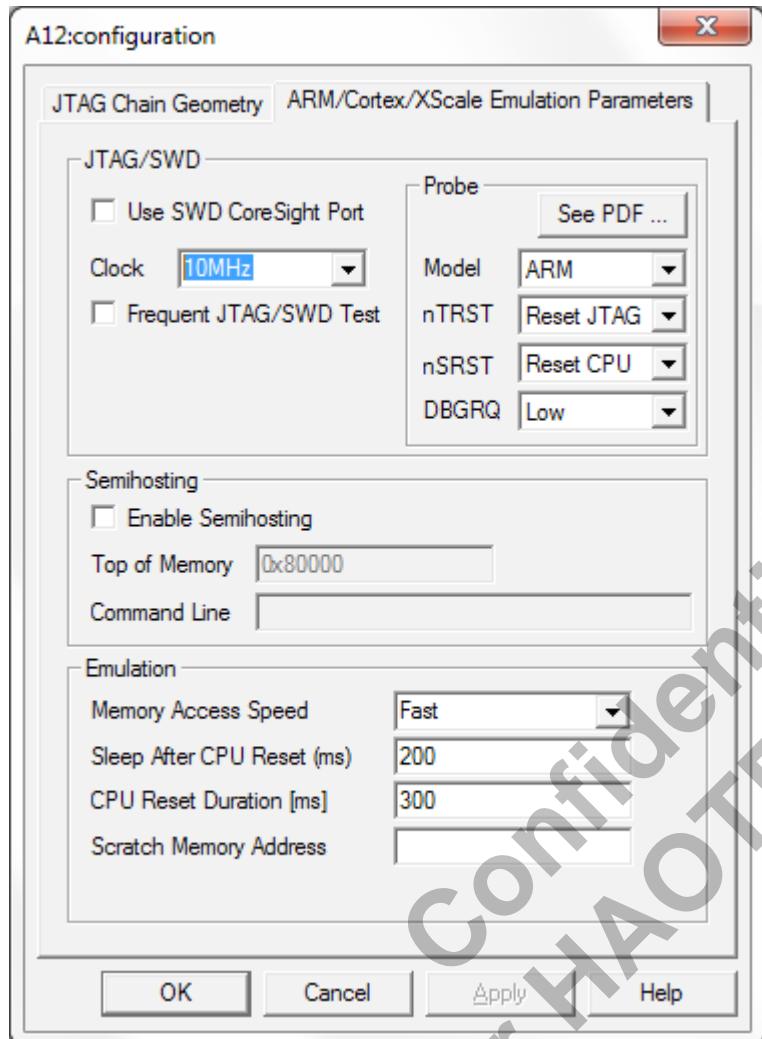


Figure 3-26. **Chameleon Setup: System Configuration > Add Target.**

10. The next step in the custom target configuration is to select a **Startup Macro**. When the dialog appears, do nothing and then click **Cancel**.

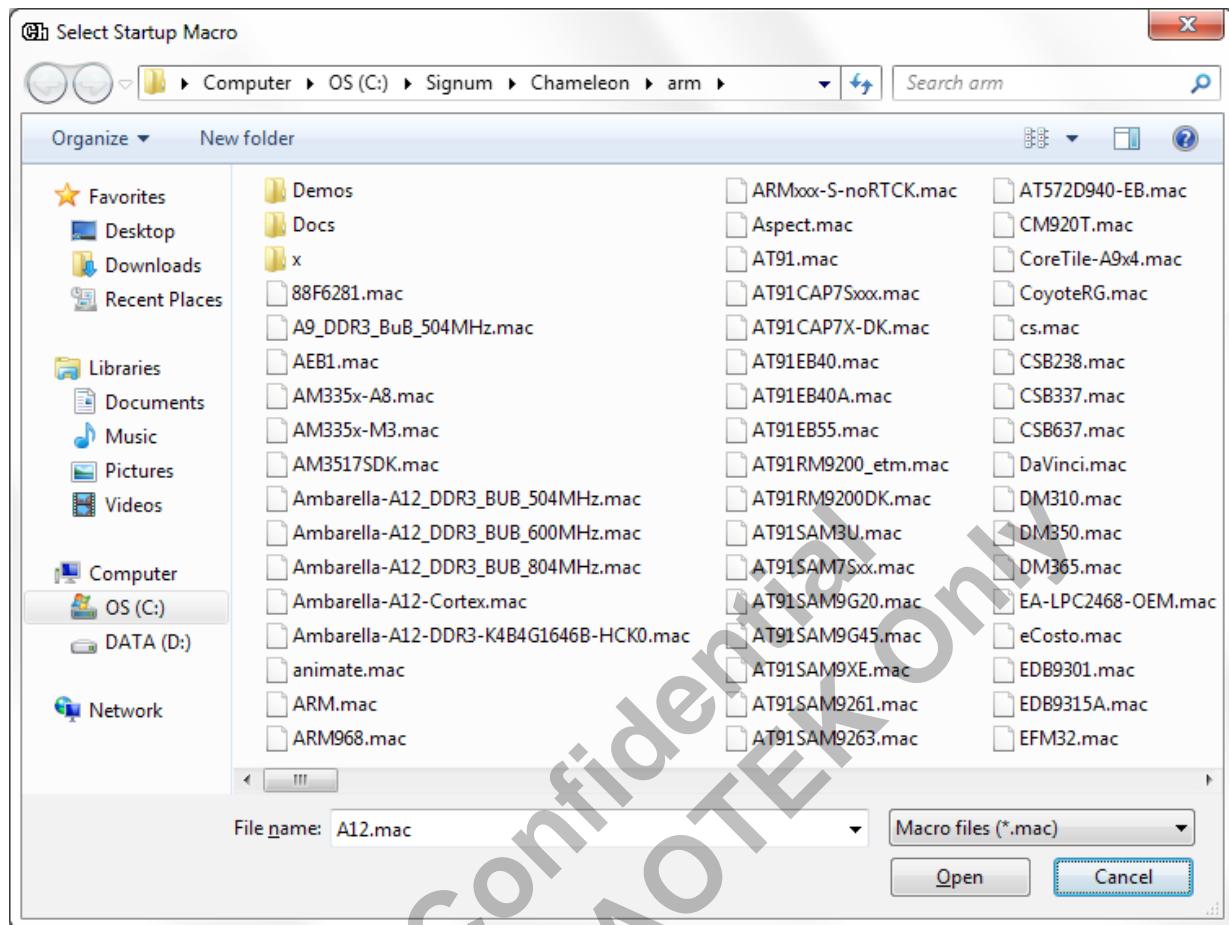


Figure 3-27. **Chameleon Setup:** Cancel the dialog.

11. The installation and the setup process should complete successfully.

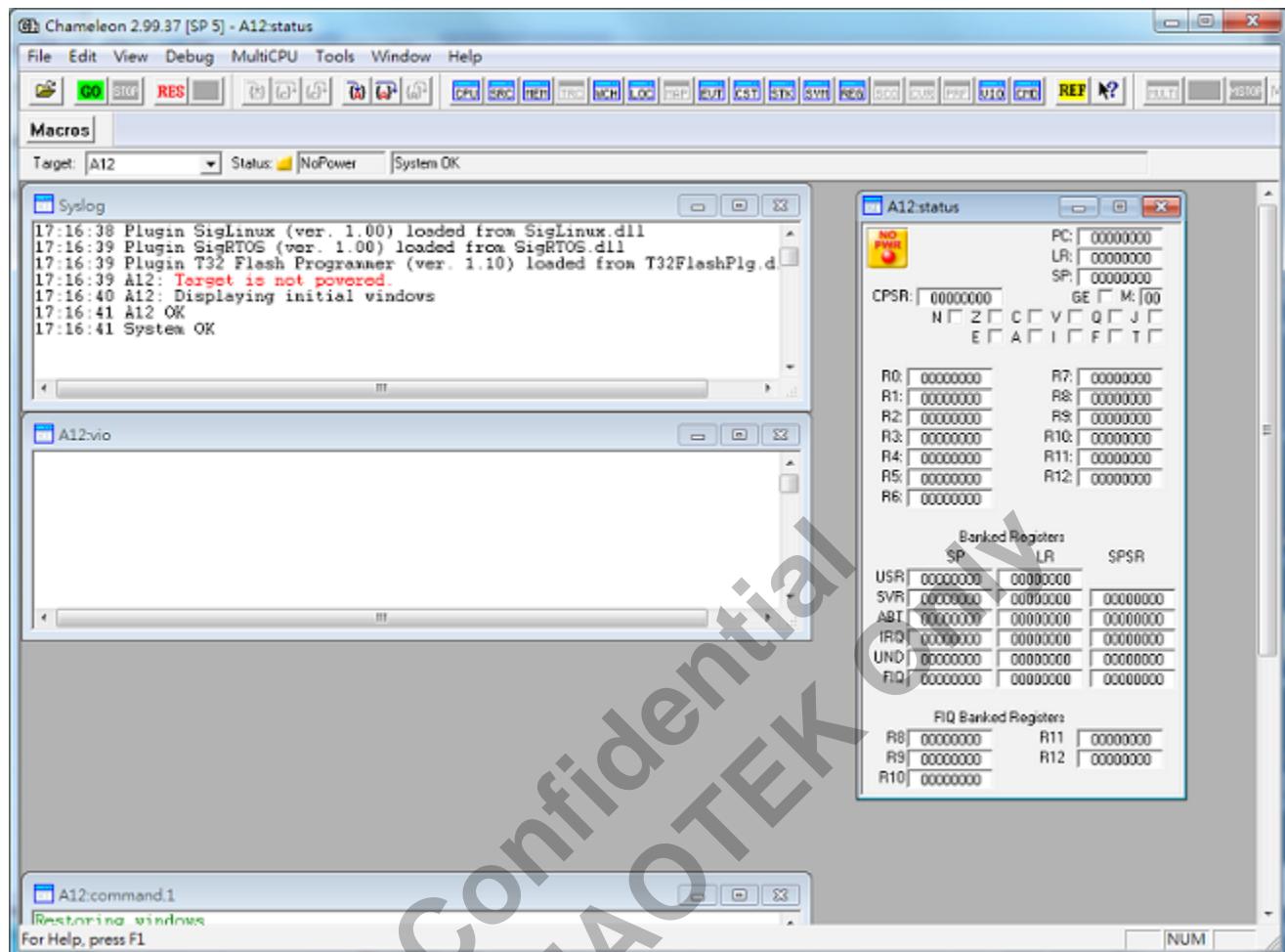


Figure 3-28. **Chameleon Setup: Startup Configuration Complete.**

3.3.3 Windows: CVD64 Setup

This section provides the installation and the driver update for the **CVD64** Debugger.

3.3.3.1 CVD64: Installation

1. Double-click the **CVD64** Debugger setup executable (CVD64_V1.20_2015_0731.exe).

- When the **Setup Wizard** opens, click **Next** to continue.

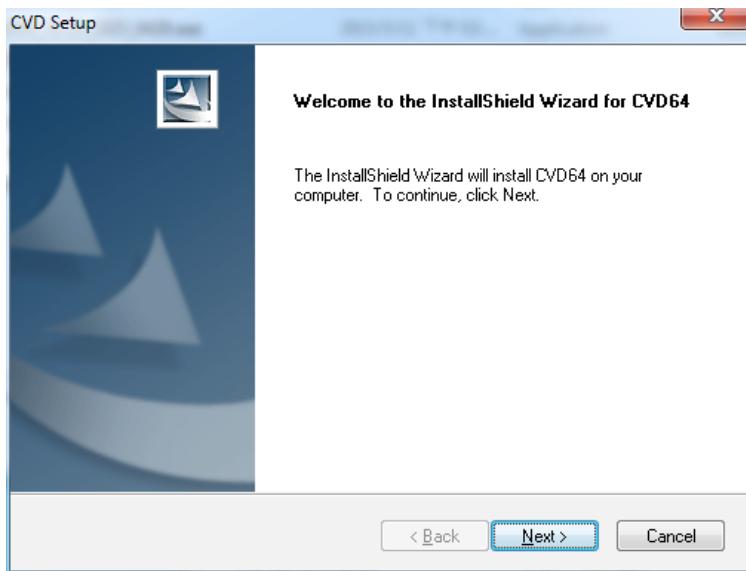


Figure 3-29. **CVD64 Debugger Installation: Setup Wizard.**

- On the **Choose Destination Location** screen, either accept the default location or click **Browse** to select a different folder. Click **Next** to continue.

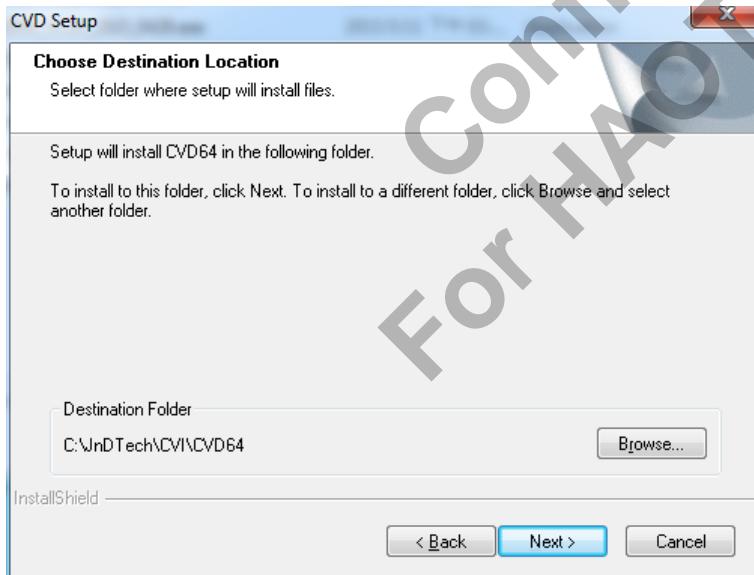


Figure 3-30. **CVD64 Debugger Installation: Choose Destination Location.**

4. On the **Setup Type** screen, accept the default type (Typical) to install. Click **Next** to continue.

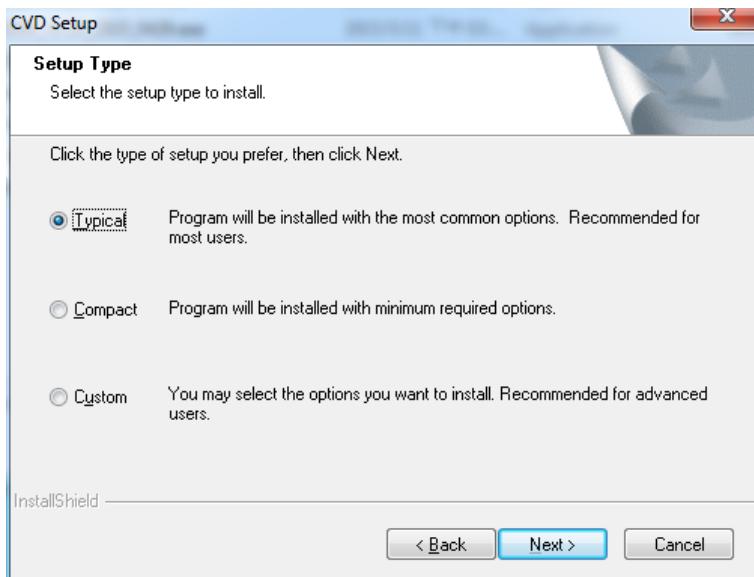


Figure 3-31. **CVD64 Debugger Installation:** *Select the Setup Type*.

5. On the **Select Program Folder** screen, accept the default folder (CVD64). Click **Next** to continue.

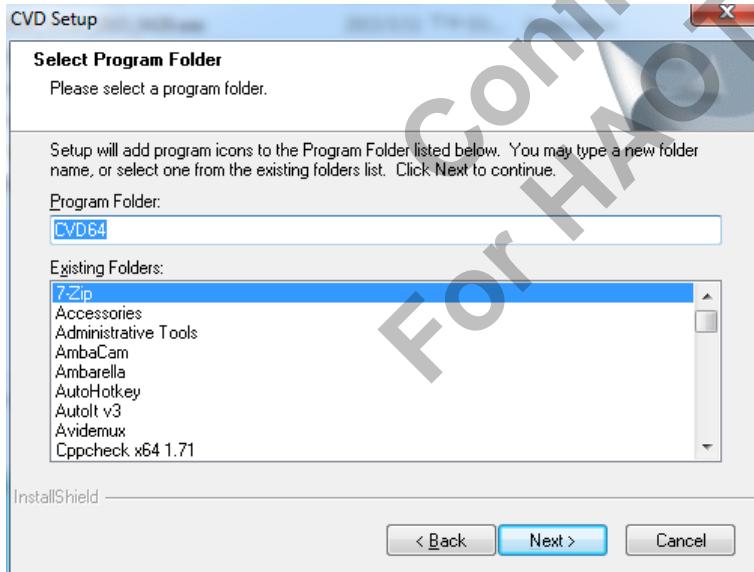


Figure 3-32. **CVD64 Debugger Installation:** *Select Program Folder*.

6. On the **Question** screen, click **No** and do not upgrade the firmware after installation completes.

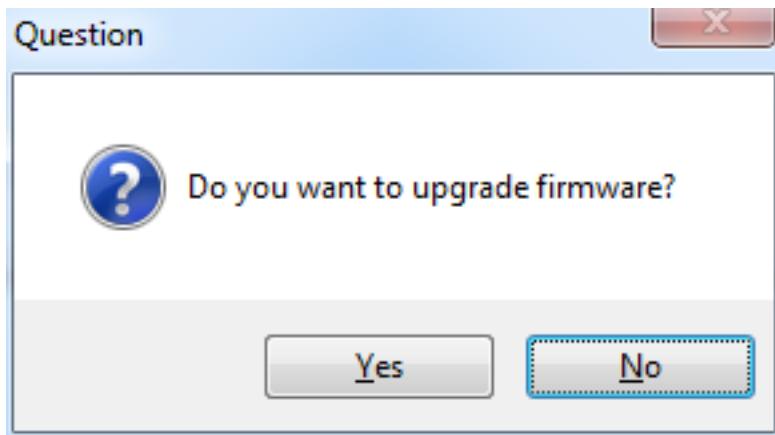


Figure 3-33. **CVD64 Debugger Installation: Question.**

7. On the **Setup complete CVD64** screen, choose **Yes, I want to restart my computer now.** Click **Finish** to continue.

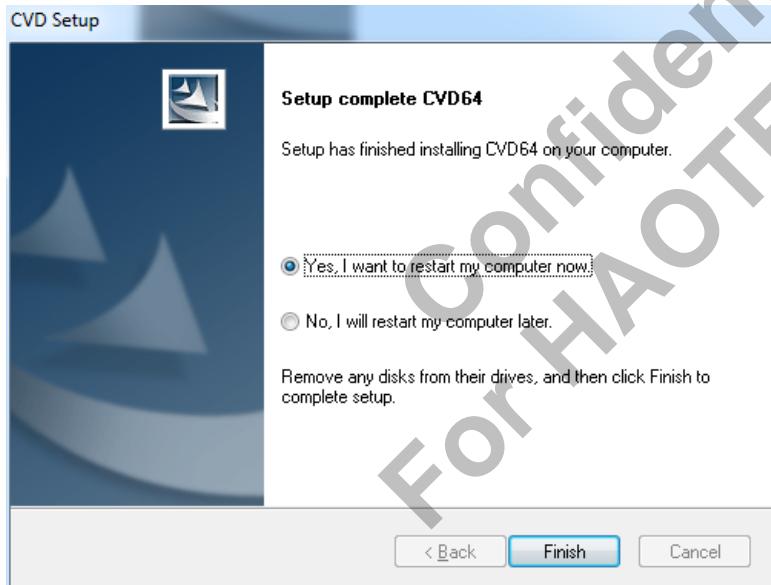


Figure 3-34. **CVD64 Debugger Installation: Setup Complete CVD64.**

8. After the computer restarts, the installation completes.

3.3.3.2 CVD64: Driver Update

1. Open the **Device Manager** on the computer management and right click the **CodeViser-ICE USB driver**. Click **Update Driver Software...** to continue.

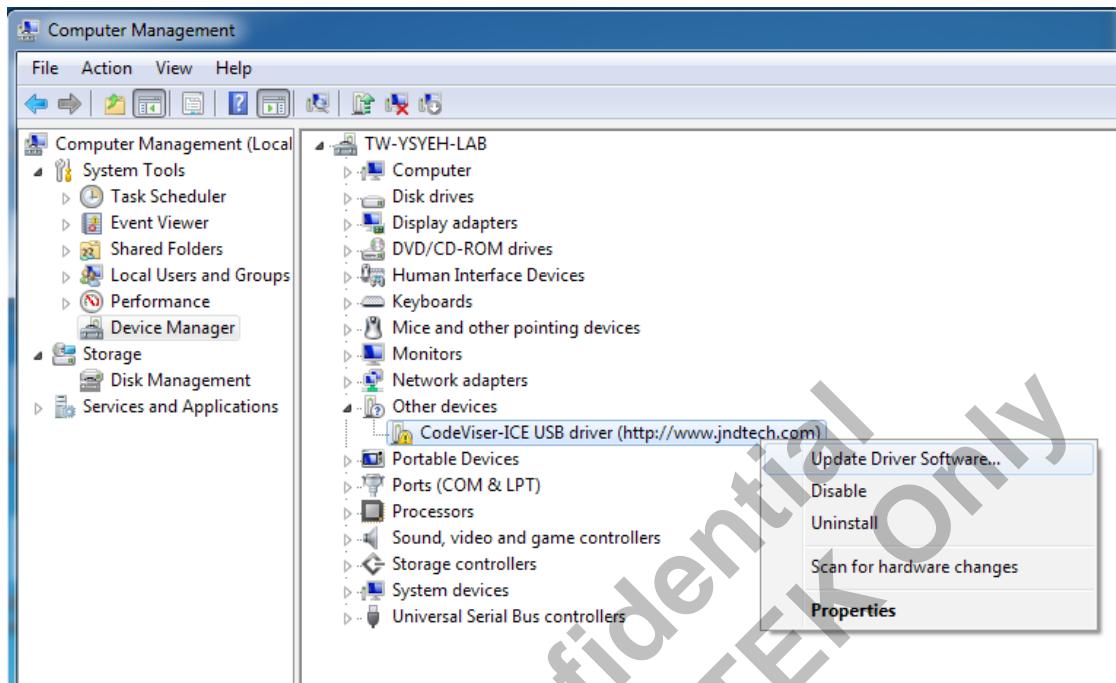


Figure 3-35. CVD64 Debugger driver update: *Device Manager*.

2. Click **Browse my computer for driver software** to search for the driver software.

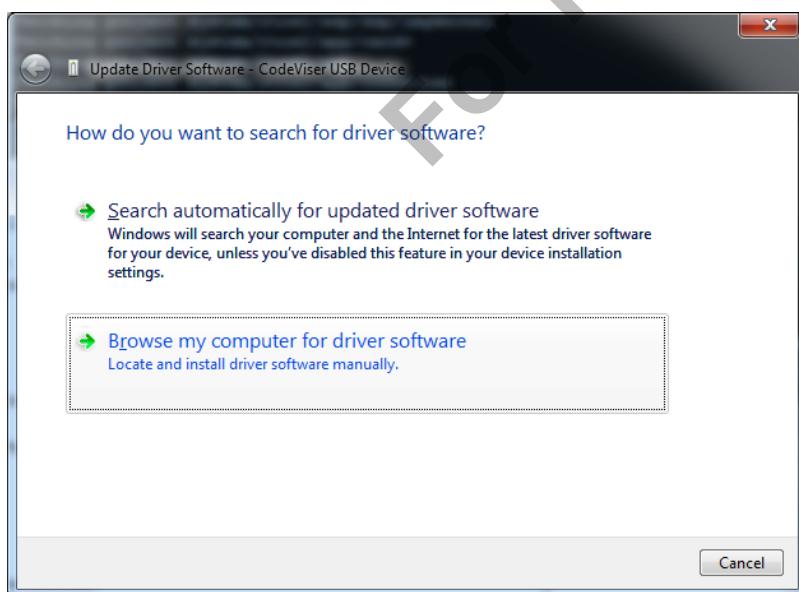


Figure 3-36. CVD64 Debugger Driver Update: *How to Search for Driver Software*.

3. Click **Browse** and search the driver software in the location: C:\JnDTech\CVI\CVD64\Drv. Click **Next** to continue.

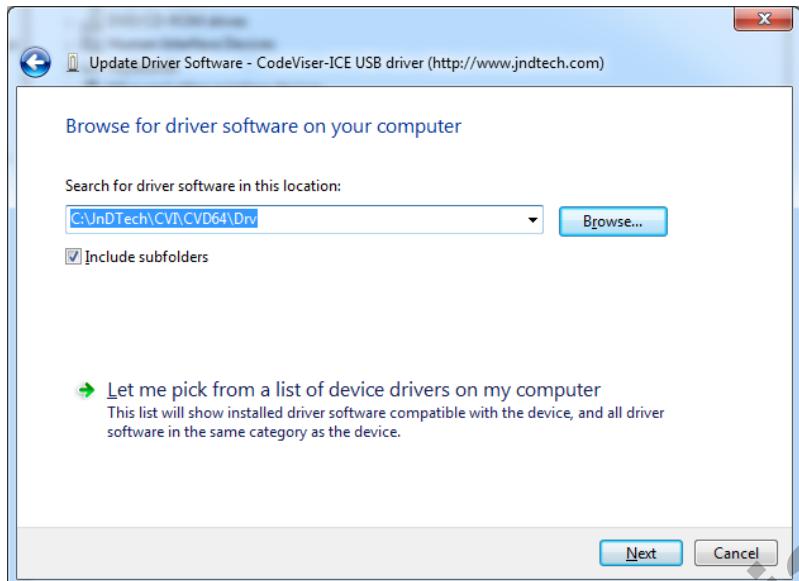


Figure 3-37. **CVD64 Debugger Driver Update: Browse for Driver Software on Your Computer.**

4. On the **Windows Security** screen, click **Install** to install the device software.

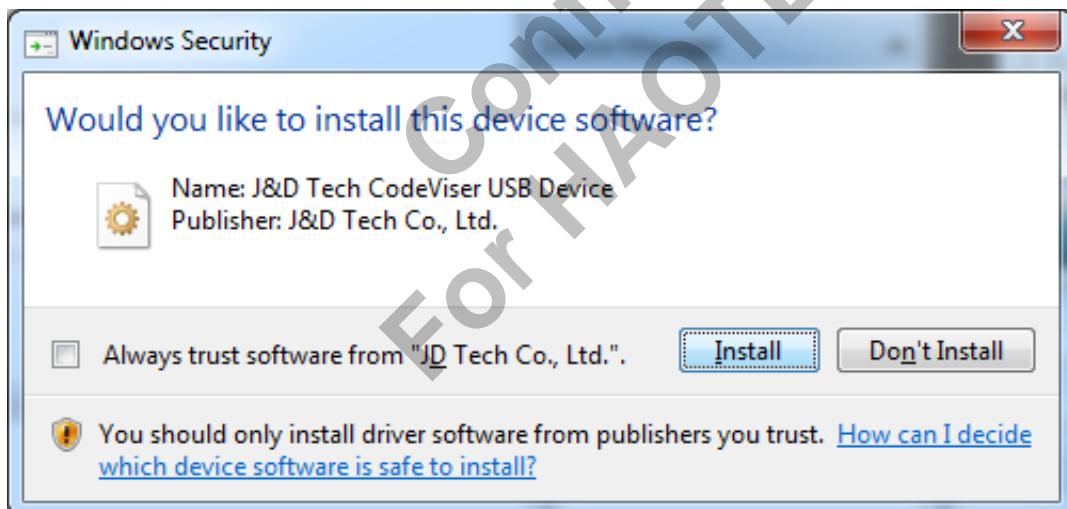


Figure 3-38. **CVD64 Debugger Driver update: Windows Security.**

5. After the installation finishes, the driver update has completed.

3.3.3.3 CVD64: Setup

1. Connect the **JTAG** debug probe emulator, EVK board, and PC as follows:
 - Connect the power cable and the **JTAG** debug probe emulator to the EVK board.
 - Connect the **JTAG** debug probe emulator to the PC via the USB cable.
 - Press the power key on the main board to power-on the board.
2. Copy the Ambarella files required for use with the CVD64 Debugger, and please obtain the released SDK from Ambarella.
Navigate to the `rtos\tools\soc\CodeViser\`, copy “Ambarella-H2” folder to `C:\JnDTech\CVI\CVD64\Script` (or the user-selected path).

For Confidential
For HAOTEK Only

4 Purchase JTAG Debug Probe

4.1 Purchase JTAG Debug Probe: Overview

This chapter explains how to purchase **JTAG** debug probe and this information is provided in the following section:

- ([Section 4.2](#)) Purchase JTAG Debug Probe: Order JTAG Debug Probe
- ([Section 4.3](#)) Purchase JTAG Debug Probe: Package of JTAG Debug Probe

4.2 Purchase JTAG Debug Probe: Order JTAG Debug Probe

The flow of getting **JTAG** debug probe from Ambarella is given below.

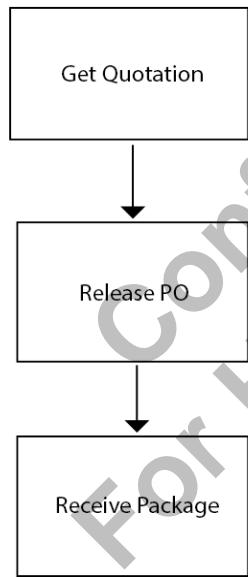


Figure 4-1. The Flow of Getting JTAG Debug Probe from Ambarella.

1. Customer has to get the official quotation.
 - a. If the customer is an Ambarella direct account, the quotation should be from the Ambarella Account manager (A/M) directly.
 - b. If the customer is from Ambarella distributor, the quotation should be from the distributor (disty).
 - c. P.N.
 - **JTAG** debug probe.
2. Release PO.
 - a. Customer PO has to note the correct P.N.

3. Receive Package of **JTAG** debug probe:
 - a. Within 7-10 working days, the customer can receive **JTAG** debug probe package from Ambarella.
 - b. Details of **JTAG** debug probe:

4.3 Purchase JTAG Debug Probe: Package of JTAG Debug Probe

4.3.1 Package of JTAG Debug Probe: IAR I-jet (ARM v7 32 bit architecture)



Figure 4-2. Package of **IAR I-jet** Debug Probe - 1.



Figure 4-3. Package of IAR I-jet Debug Probe - 2.



Figure 4-4. Package of IAR I-jet debug probe - 3.

4.3.2 Package of JTAG Debug Probe: CodeViser (ARM v8 64 Bit Architecture)



Figure 4-5. Package of **CodeViser** Debug Probe - 1.

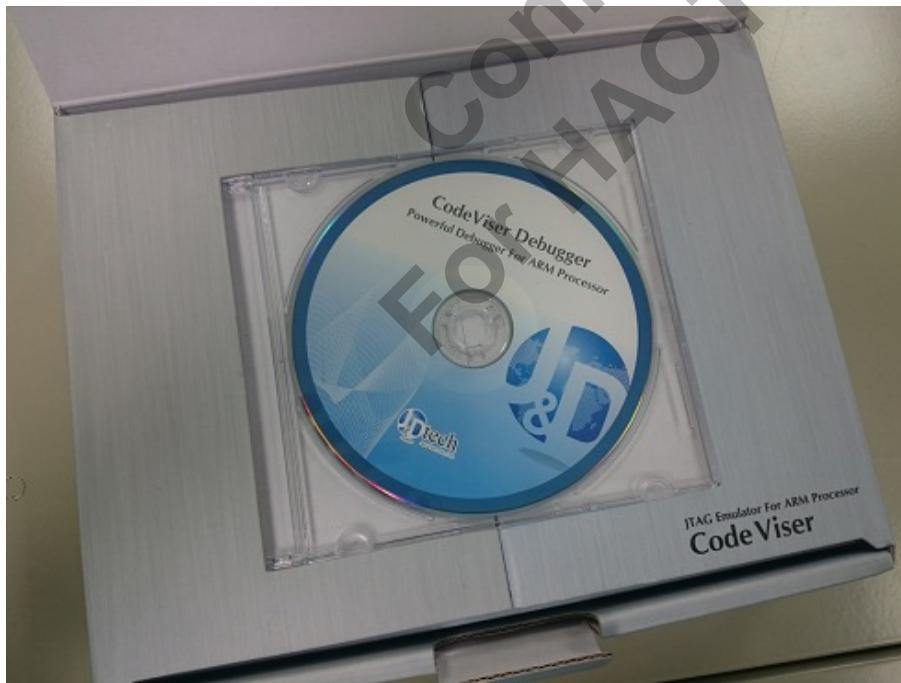


Figure 4-6. Package of **CodeViser** Debug Probe - 2.



Figure 4-7. Package of **CodeViser** debug probe - 3.



Figure 4-8. Package of **CodeViser** debug probe - 4.

5 Build SDK6

5.1 Build SDK6: Overview

This chapter provides the step-by-step instructions for building the SDK6 package. These instructions assume that the build environments have been setup according to instructions provided in [Chapter 3 “Build Environment”](#) of this document.

The user must get the Ambarella SDK release package first and then, follow the following steps to build. The package may include two files:

- `ambalink_sdk_3_10.xxx.tar.gz`: The AmbaLink SDK is used to enable the network support for the system-on-chip (SoC), and this is based on the one that the user actually uses.
- `ax_release.xxx.tar.gz`: The SSP, ARD/MW or ARD/APP consists of software components.

This chapter is organized as follows.

- ([Section 5.2](#)) Build SDK6: Build Ambalink SDK
- ([Section 5.3](#)) Build SDK6: Build SSP
- ([Section 5.4](#)) Build SDK6: Build MSP
- ([Section 5.5](#)) Build SDK6: Build ARD
- ([Section 5.6](#)) Build SDK6: Burn the Firmware

Note that the user is encouraged to store an unmodified version of the SDK to facilitate the update process.

5.2 Build SDK6: Build Ambalink SDK

The Ambalink SDK must be built on a Linux machine. Please follow the steps below to build the AmbaLink Linux binary image.

1. Untar the following SDK packages into a working directory: (Let's use the A12SDK as an example)

```
$ mkdir project_name; cd project_name  
$ tar zxf ambalink_sdk_3_10.20141201.tar.gz
```

The output for an `ls` command should read as follows:

```
$ ls  
ambalink_sdk_3_10 ambalink_sdk_3_10.20141201.tar.gz
```

Prepare suitable WIFI driver and untar it under `ambalink_sdk_3_10\external_sdk`

```
$ cd ambalink_sdk_3_10  
$ mkdir external_sdk
```

Please copy bcmdhd.tar.gz into external_sdk
\$ cd external_sdk
\$ tar zxf bcmdhd.tar.gz

[project dependent]

2. Build the Linux image:

The command “make prepare_oem” is used to copy Ambarella’s binaries from the output/ to the user’s build directory in output.oem/. Therefore, in the Ambalink SDK, the **AMBA_OUT_TARGET** and **TARGET**

variables need to be specified.

- **AMBA_OUT_TARGET** is the target directory name in the output/ used for releasing SDK. Typically, it is \${chipname}_ambalink, for example, a12_ambalink, a9_ambalink or a9s_ambalink.
- **TARGET** is the target directory name in output.oem/ used by the user.

[project dependent]: For which defconfig to be used in the release SDK, please check README of ambalink_sdk_3_10/ambarella/configs in the SDK or contact an Ambarella AE.

```
$ cd ambalink_sdk_3_10/ambarella  
$ make prepare_oem AMBA_OUT_TARGET=a12_ambalink TARGET=a12_ambalink  
[project dependent]  
$ make O=../output.oem/a12_ambalink a12_ambalink_defconfig [project dependent]  
$ cd ../output.oem/a12_ambalink/  
$ make
```

3. After the build process is complete, locate the Linux images at
ambalink_sdk_3_10/output.oem/a12_ambalink/images/

The output for an ls command should read as follows:

```
$ ls images  
Image linuxfs.bin.pref.ubifs rootfs.tar rootfs.tar.gz rootfs.ubi rootfs.ubifs
```

The images used for the Linux Kernel and the Root File System partition are Image and rootfs.ubi, respectively.

Build DTB while building Linux images:

Add the path of the device tree source file and enable the building of the device tree in the buildroot menuconfig. The user can check the value of BR2_LINUX_KERNEL_CUSTOM_DTS_PATH in buildroot config for the default device tree source file.

If the configs below are set in buildroot config, dtb can be built while building the linux kernel.

```
BR2_LINUX_KERNEL_USE_CUSTOM_DTS=y  
BR2_LINUX_KERNEL_CUSTOM_DTS_PATH="$(TOPDIR) /../ambarella/bootloader/boards/  
a12evk/bsp/a12evk_nand.dts"
```

The output DTB is located at

ambalink_sdk_3_10/output.oem/a12_ambalink/images/a12evk_nand.dtb.
[project dependent]

4. Build the device tree blob for Linux kernel 3.10:

```
$ cd ambalink_sdk_3_10/ambarella/bootloader/boards/a12evk      [project dependent]
$ source ../../amboot/config/build_amboot.env
$ make sync_build_mkcfg
$ make a12evk_amboot_only_config                                [Only once for the first time]
$ make dtb                                                       [project dependent]
```

5. After the build process completes, locate the device tree blob at

ambalink_sdk_3_10/ambarella/bootloader/out/a12evk/dtb_programmer/

6. The output for an ls command should read as follows:

```
$ ls dtb*
dtb_debug.bin  dtb_debug.o
```

The device tree blob binary used for the Linux Kernel is dtb_debug.bin.

And then, copy Image, rootfs.ubi and dtb_debug.bin to rtos\linux_image\
[project dependent]

5.3 Build SDK6: Build SSP

Please follow the steps below to build the SSP binary image.

1. Untar the following SDK packages into a working directory. (Let's use the A12SDK as an example)

```
$ mkdir project_name; cd project_name
$ tar zxf a12_release.20141201.tar.gz
```

The output for an ls command should read as follows:

```
$ ls
rtos  a12_release.20141201.tar.gz
```

2. Please copy Image, rootfs.ubi and dtb_debug.bin to rtos\linux_image\
[project dependent] if the user rebuilds Ambalink SDK.

3. Build the SSP firmware:

```
$ cd rtos
$ make distclean
$ make a12_ssp_unittest_defconfig                               [project dependent]
$ make
```

4. After the build process completes, obtain the final firmware at

rtos/output/out/fwprog/bst_bld_sys_dsp_rom_lnx_rfs.elf [project dependent]

Based on the project, after the build is done, it may have 64 bit firmware at
rtos/output/out/fwprog64/bst_bld_sys_dsp_rom_lnx_rfs.elf [project dependent]

5.4 Build SDK6: Build MSP

Please follow the steps below to build the MSP binary images.

1. Untar the following SDK packages into a working directory: (Let's use the A12SDK as an example)

```
$ mkdir project_name; cd project_name  
$ tar zxf a12_release.20141201.tar.gz
```

The output for an `ls` command should read as follows:

```
$ ls  
rtos a12_release.20141201.tar.gz
```

2. Please copy `Image`, `rootfs.ubi` and `dtb_debug.bin` to `rtos\linux_image\
[project dependent]` if user rebuilds Ambalink SDK.

3. Build the MSP firmware using the following commands:

```
$ cd rtos  
$ make distclean  
$ make a12_mw_unittest_defconfig  
$ make
```

[project dependent]

4. After the build process completes, obtain the final firmware at

```
rtos/out/fwprog/bst_bld_sys_dsp_rom_lnx_rfs.elf
```

[project dependent]

Based on the project, after the build is done, it may have 64 bit firmware at
`rtos/output/out/fwprog64/bst_bld_sys_dsp_rom_lnx_rfs.elf`

[project dependent]

5.5 Build SDK6: Build ARD

Please follow the steps below to build the ARD binary image.

1. Untar the following SDK packages into a working directory: (Let's use the A12SDK as an example)

```
$ mkdir project_name; cd project_name  
$ tar zxf a12_release.20141201.tar.gz
```

The output for an `ls` command should read as follows:

```
$ ls  
rtos a12_release.20141201.tar.gz
```

2. Please copy `Image`, `rootfs.ubi` and `dtb_debug.bin` to `rtos\linux_image\`.
[project dependent] if the user rebuilds Ambalink SDK.
3. Build the ARD firmware:


```
$ cd rtos
$ make distclean
$ make a12_app_connected_defconfig
$ make
```

[project dependent]
4. After the build process completes, obtain the final firmware at
`rtos/out/fwprog/bst_bld_sys_dsp_rom_lnx_rfs.elf`
[project dependent]

Based on the project, after the build is done, it may have 64 bit firmware at
`rtos/output/out/fwprog64/bst_bld_sys_dsp_rom_lnx_rfs.elf`
[project dependent]

5.6 Build SDK6: Burn the Firmware

- (Section 5.6.1) Burn the Firmware: By I-jet (ARM v7 32 Bit Architecture)
- (Section 5.6.2) Burn the Firmware: By CodeViser (ARM v8 64 Bit Architecture)

5.6.1 Burn the Firmware: By I-jet (ARM v7 32 Bit Architecture)

This section describes the steps required to burn the EVK firmware to the NAND flash using **Chameleon** (Chapter 3 “Build Environment”).

Burn the firmware to the NAND flash memory by using the following steps: (Let’s use the A12SDK as an example)

1. Power on the EVK Board, and enter the **AmbaTools** mode.
 - If the EVK firmware currently exists in the NAND flash memory, the serial console can be used to enter the **AmbaTools** mode. From the PC serial console, hold the **Enter** key down and simultaneously press the **Power** button on the EVK Board.
2. From the PC, open the **JTAG** debug probe **Chameleon** program:
 - Select **A12** (by default, only **A12** is running) and click the **STOP** icon.
3. Open and load the firmware to **A12**:
 - Click the **Load** button to open the **A12: Load** window.
 - Click **Browse** and select the firmware package.
 - The firmware is located at
`rtos/output/out/fwprog/bst_bld_sys_dsp_rom_lnx_rfs.elf` [project dependent]

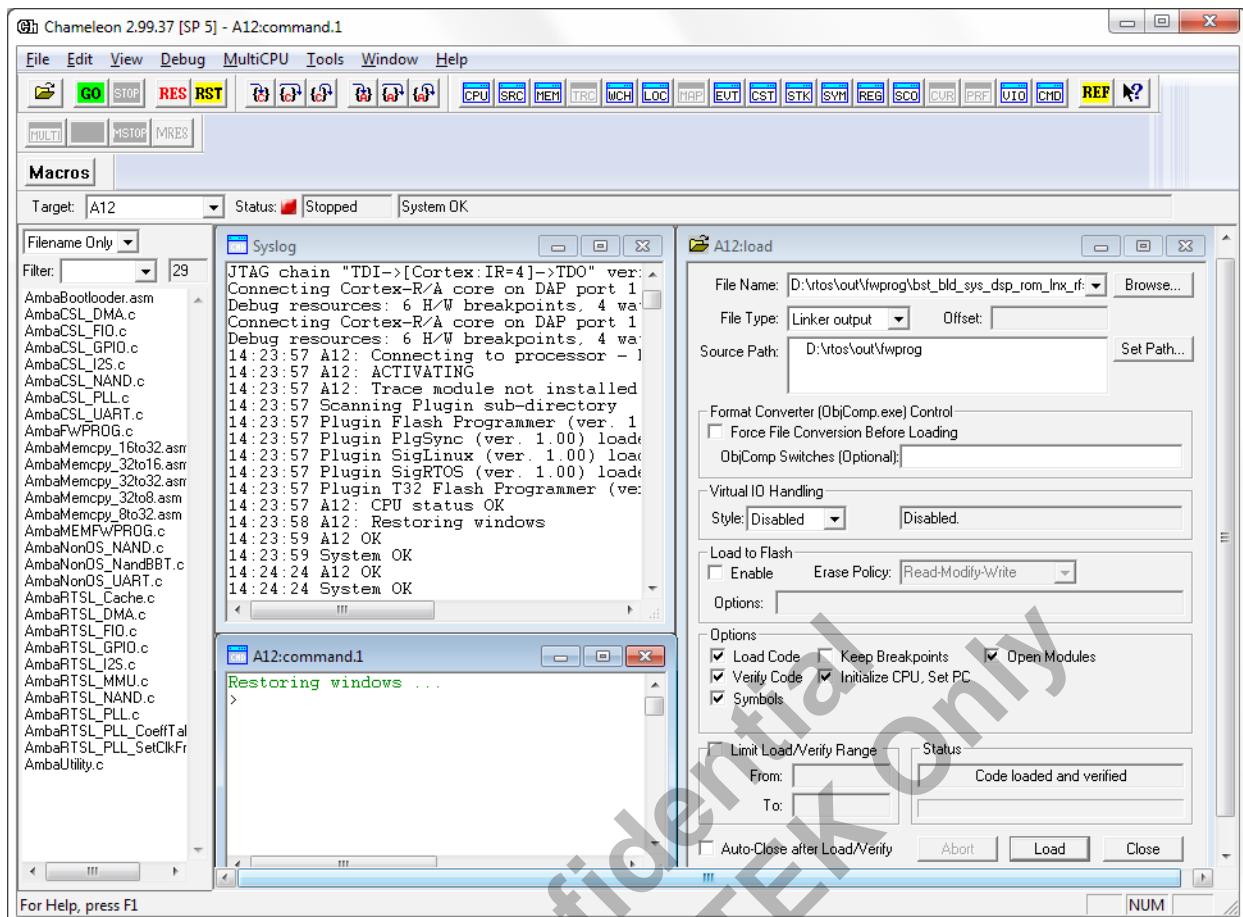


Figure 5-1. Chameleon Starts.

- Click the **Load** button. **Chameleon** will begin loading the firmware.

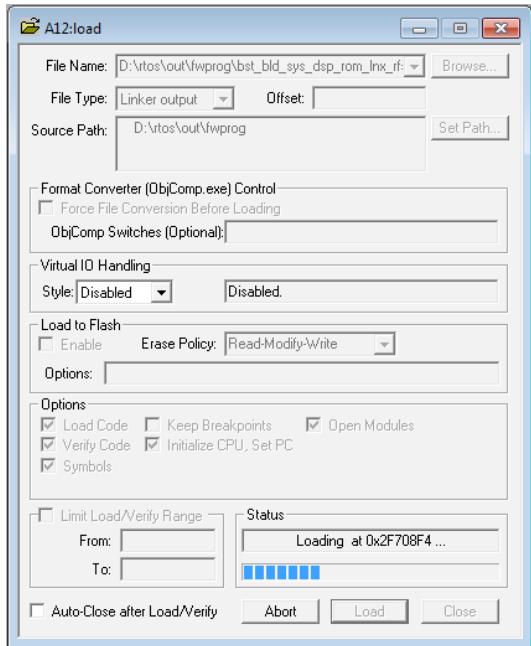


Figure 5-2. Firmware: Load the Firmware with **Chameleon**.

4. When the firmware has completed loading, click the **Chameleon** main menu icon **GO**. The terminal emulator readout should indicate that the NAND Flash program is running, as shown below.

```

COM1 - PuTTY

erase Linux Hibernation Image ...Done
program ok

Linux Root FS code found in firmware!
    Crc32 : 0x3a118c13
    Version: 1.0.0
    VerDate: 2014/12/4
    ImgLen : 30146560
    MemAddr: 0x0
    Flag : 0x0
    Magic : 0xa324eb90

Program Linux Root FS to NAND flash ...
verifying image crc ... Done
progress: 100
program ok

----- Report -----
Bootstrap : success
Bootloader : success
System Software : success
DSP uCode : success
System ROM Data : success
Linux Kernel : success
Linux Root FS : success

- Program Terminated -

```

Figure 5-3. Firmware: Burn the Firmware to NAND Flash.

5. Power-off the EVK Board.
6. Power-on the EVK Board, and the system will boot from the NAND Flash.

5.6.2 Burn the Firmware: By CodeViser (ARM v8 64 Bit Architecture)

This section describes the steps required to burn the EVK firmware to the NAND flash using **CVD64** program ([Chapter 3 “Build Environment”](#)).

Burn the firmware to the NAND flash memory by using the following steps: (use H2SDK as an example)

1. Switch No.6 of SW3 for 32/64 bit firmware burning. (See Figure 5-4 and Figure 5-5).

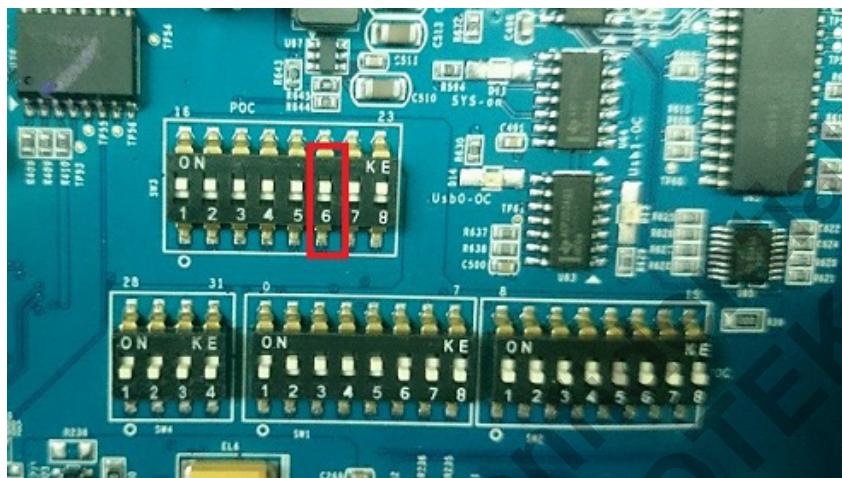


Figure 5-4. Switch No.6 of SW3 to ON for 32 Bit Firmware Burning.

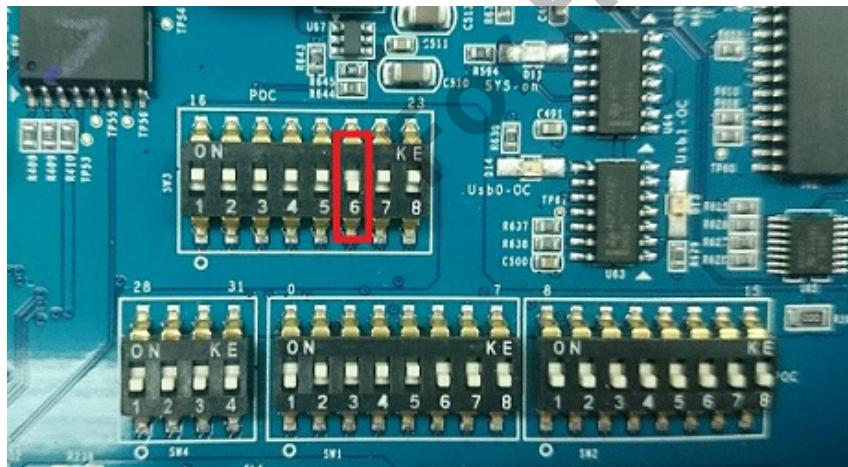


Figure 5-5. Switch No.6 to OFF of SW3 for 64 Bit Firmware Burning.

2. Turn on ByPass of EVK Board. (Switch No.1 & 3 of SW2 to ON).

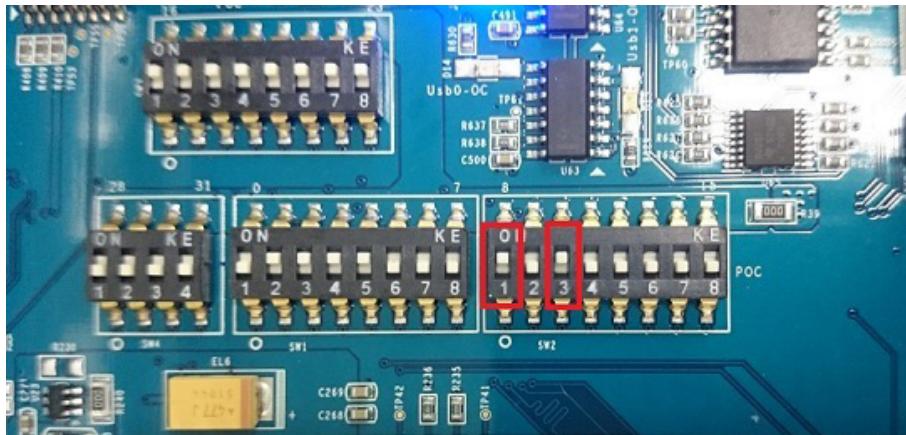


Figure 5-6. ByPass State.

3. Connecting **CodeViser** debug probe between EVK board and PC.
4. Power on the EVK board.
5. From the PC, open the **CodeViser** debug probe **CVD64** program.
6. Run the script file:

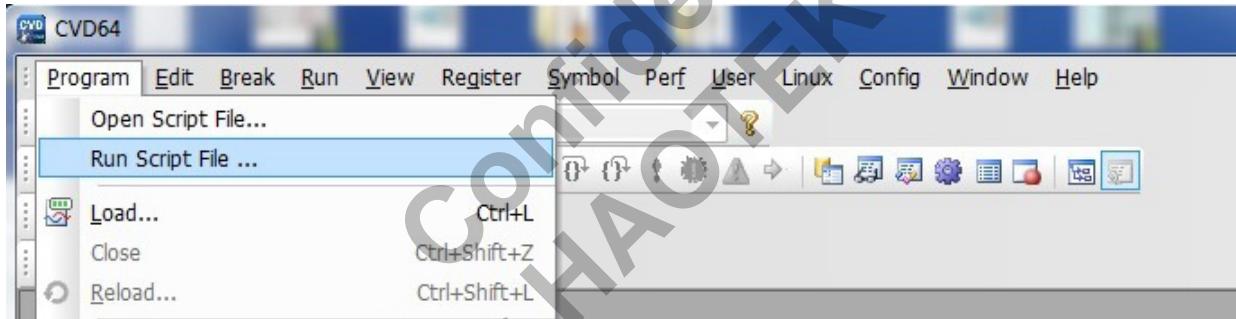


Figure 5-7. Run Script.

- First Run "C:\JnDTech\CVI\CVD64\Script\Ambarella-H2\Ambarella_H2_CVD64.csf" script file.
- Second Run "C:\JnDTech\CVI\CVD64\Script\Ambarella-H2\Bin\Ambarella-H2-LPDDR3_H9CCNNN8GTMLAR_336MHz.csf" script file.

7. After running the above two script files, press **Load** to load the firmware.

- The 32bit f/w is located in: `rtos\output\out\fwprog\bst_bld_sys_dsp_rom.elf` [project dependent]

or

- The 64bit f/w is located in: `rtos\output\out\fwprog64\bst_bld_sys_dsp_rom.elf` [project dependent]

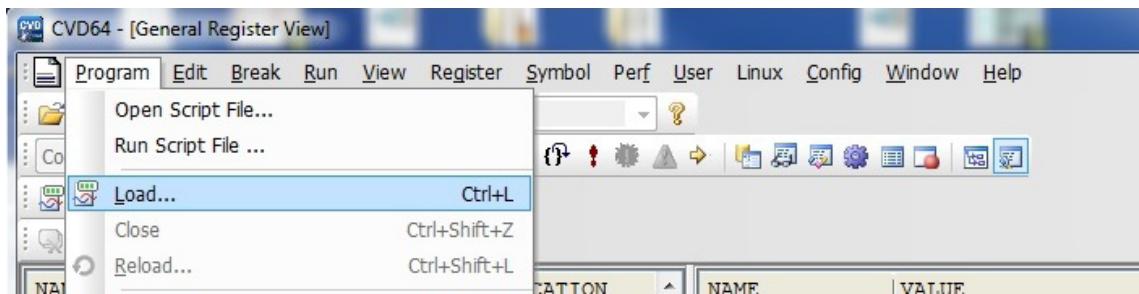


Figure 5-8. Press Load to Load the Firmware.

8. Wait for the download to complete.

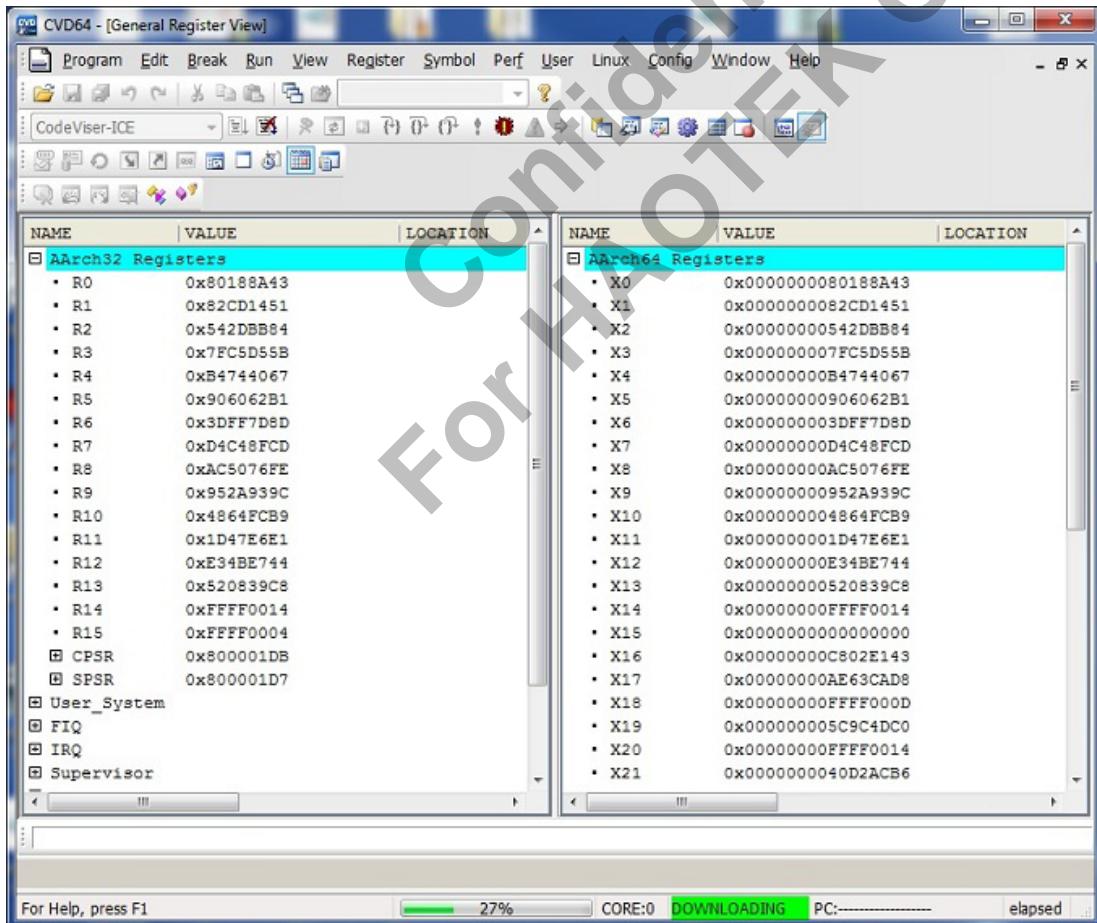


Figure 5-9. Downloading.

- After the downloading completes, press **Go** to burn the firmware to NAND flash.

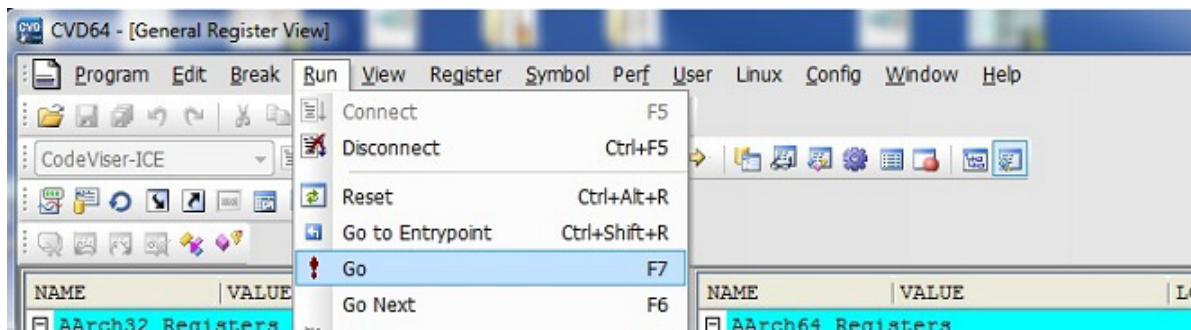


Figure 5-10. Press **Go** to Burn the Firmware to the NAND Flash.

- The terminal emulator readout should indicate that the NAND Flash program is running, as shown below.

```
SYS_DATA found in firmware!

Crc32 : 0xa2bdd622
Version: 256.0
VerDate: 2016/2/3
ImgLen : 16349184
MemAddr: 0x0
Flag : 0x0
Magic : 0xa324eb90
Verifying image crc ...PASS
                                         100%
Program image to NAND flash ...
SYS_DATA program ok

----- Report -----
BOOTSTRAP : SUCCESS
BOOTLOADER : SUCCESS
SYS_SW : SUCCESS
DSP_uCode : SUCCESS
SYS_DATA : SUCCESS
- Program Terminated: SUCCESS -
```

Figure 5-11. Burn the Firmware to the NAND Flash.

- Power off the EVK board.
- Turn off ByPass of EVK board. (Switch No.1 and 3 of SW2 to OFF)
- Switch No.6 of SW3 back to ON after burning 64 bit firmware.
- Power on the EVK board and the system will boot from the NAND flash.

6 Customize the AmbaLink SDK

6.1 Customize the AmbaLink SDK: Overview

This chapter provides instructions for customizing the AmbaLink SDK, including procedures for modifying the boot script, removing pre-existing Ambarella add-ons, and incorporating custom add-ons.

The chapter is organized as follows:

- [\(Section 6.2\) Customize the AmbaLink SDK: Introduction to Buildroot](#)
- [\(Section 6.3\) Customize AmbaLink SDK: Modify Boot Script](#)
- [\(Section 6.4\) Customize AmbaLink SDK: Remove Ambarella Add-Ons](#)
- [\(Section 6.5\) Customize AmbaLink SDK: Incorporate New Add-Ons](#)
- [\(Section 6.6\) Customize AmbaLink SDK: Wi-Fi Network Packages](#)
- [\(Section 6.7\) Customize AmbaLink SDK: Rebuild Package](#)
- [\(Section 6.8\) Customize AmbaLink SDK: Store Configurations](#)

6.2 Customize the AmbaLink SDK: Introduction to Buildroot

The Ambalink SDK package leverages the Buildroot system as a basic framework. The Buildroot system includes a set of makefiles that simplify and automate the Linux system build process through the use of cross-compilation toolchains. These toolchains are capable of creating a root file system, compiling a Linux kernel image, and generating a boot loader.

For more information about the Buildroot system, please visit the official website at <http://buildroot.uclibc.org/>.

For a list of relevant documents and manuals, please refer to <http://buildroot.uclibc.org/downloads/manual/manual.html>.

6.3 Customize AmbaLink SDK: Modify Boot Script

The Ambalink SDK enables the system boot file to be customized in order to add or remove operations that will be executed on boot-up (e.g., executing a new daemon).

Steps for customizing the system boot file are provided below:

1. Modify the boot script:

```
$ vi project_name/ambalink_sdk_3_10/ambarella/package/ambarella_customize/
source/etc/init.d/S50service
```

2. Build the Linux images:

```
$ cd project_name/ambalink_sdk_3_10/output.oem/a12_ambalink/; make
```

3. Rebuild the firmware:

Copy Image and rootfs.ubi to rtos\linux_image. [project dependent]
And then, rebuild the SDK firmware.

```
$ cd rtos  
$ make amba_fwprog_clean  
$ make amboot
```

The output firmware is at rtos/output/out/fwprog/bst_bld_sys_dsp_rom_lnx_rfs.elf [project dependent].

6.4 Customize AmbaLink SDK: Remove Ambarella Add-Ons

To remove pre-existing Ambarella add-on libraries or daemons, please complete the steps below.

1. Navigate to the Ambalink SDK directory: (using A12SDK as an example)

```
$ cd project_name/ambalink_sdk_3_10/
```

2. Clean up the existing target template:

```
$ cd output.oem/a12_ambalink  
$ make clean  
$ cd ../../ambarella  
$ make prepare_oem AMBA_OUT TARGET=a12_ambalink TARGET=a12_ambalink  
$ make O=../output.oem/a12_ambalink a12_ambalink_defconfig
```

3. Remove add-on packages as desired:

```
$ cd ../output.oem/a12_ambalink  
$ make menuconfig
```

From the ambalink_sdk_3_10/output.oem/a12_ambalink/.config menu, select **Local Packages** for the target and **Ambarella Packages** as shown below:

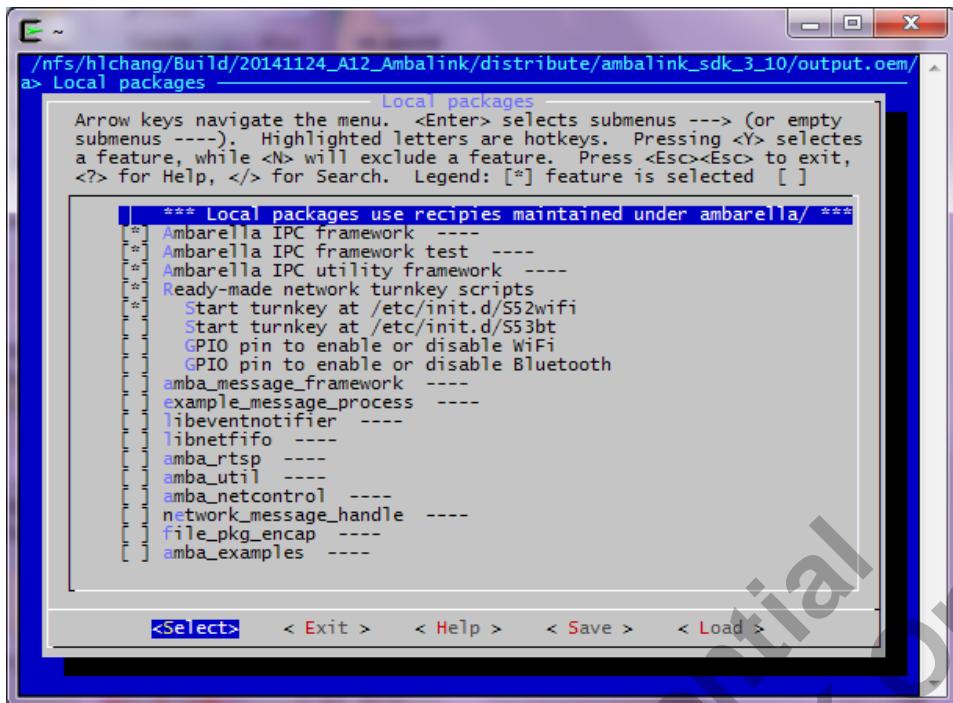


Figure 6-1. Remove Ambarella Add-Ons to Customize the SDK.

4. Build the Linux images:

```
$ make
```

5. Rebuild the firmware:

- Copy Image and rootfs.ubi to rtos\linux_image. [project dependent]
- And then, rebuild the SDK firmware.

```
$ cd rtos  
$ make amba_fwprog_clean  
$ make amboot
```

The output firmware is at rtos/output/out/fwprog/bst_bld_sys_dsp_rom_lnx_rfs.elf [project dependent].

6.5 Customize AmbaLink SDK: Incorporate New Add-Ons

To incorporate a new add-on package, modify the contents of the directories listed below.

- ambalink_sdk_3_10/ambarella/package/

This folder contains all package configurations. Modify the Config.in file to specify a package and add a new folder for custom configurations.

- ambalink_sdk_3_10/pkg/

This folder contains the package implementations. Add a new folder for custom implementations. For example, to add a new package lu_example_util, the following steps should be performed:

1. Create a folder for the new implementation in ambalink_sdk_3_10/pkg/:

```
$ cd ambalink_sdk_3_10/pkg/  
$ mkdir lu_example_util  
$ cd lu_example_util
```

2. Add an implementation and br.mk to the folder.

```
$ ls  
br.mk lu_example_util.c
```

3. Create a folder for the package configuration in ambalink_sdk_3_10/ambarella/package/:

```
$ cd ambalink_sdk_3_10/ambarella/package/  
$ mkdir lu_example_util  
$ cd lu_example_util
```

4. Add a configuration file to the folder:

```
$ ls  
Config.in lu_example_util.mk
```

5. Modify ambalink_sdk_3_10/ambarella/package/Config.in to make changes to the new package:

```
@@ -11,3 +11,8 @@ menu "Ambarella external packages"  
comment "External SDK from Other vendors like Atheros/Realtek/Broadcom/Marvell"  
source "ambarella/package/external_sdk/Config.in"  
endmenu  
+  
+menu "Customer packages"  
+comment "Customer Packages"  
+source "ambarella/package/lu_example_util/Config.in"  
+endmenu
```

6. Select the new package in menuconfig:

```
$ cd ambalink_sdk_3_10/output.oem/a12_ambalink  
$ make menuconfig
```

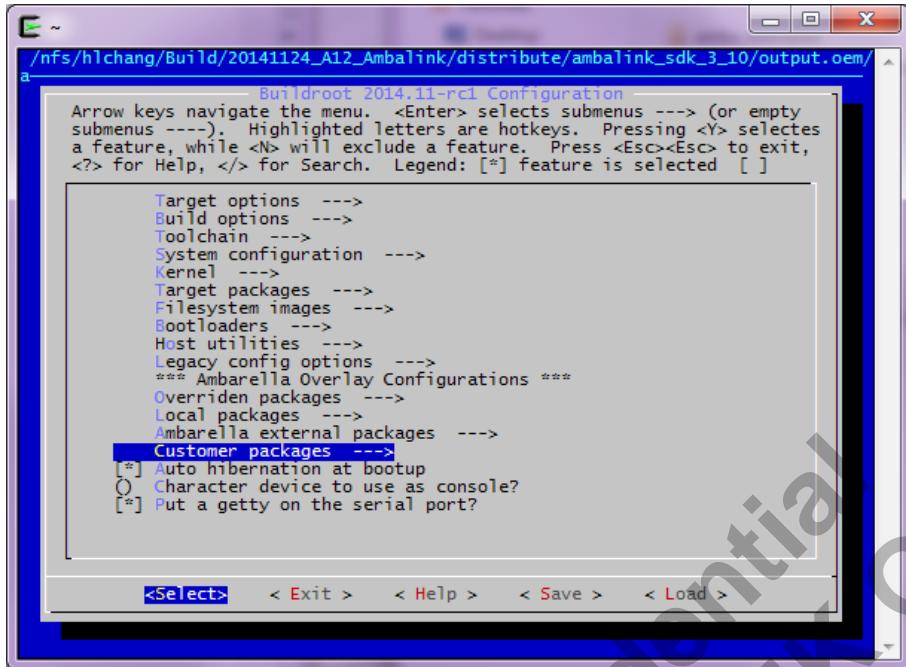


Figure 6-2. Select the New Package from **menuconfig > Customer Packages**.

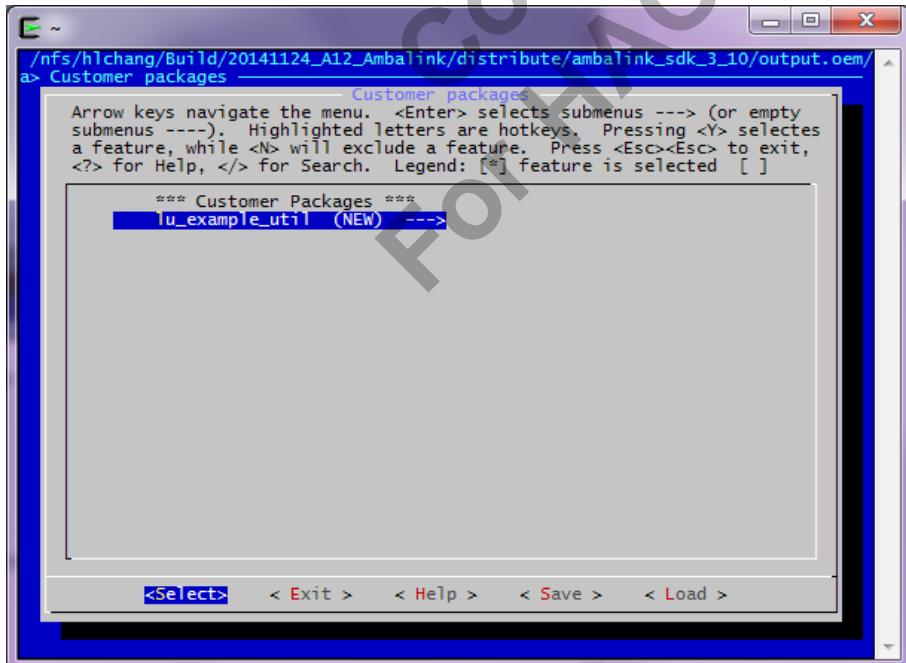


Figure 6-3. From **menuconfig > Customer Packages** Choose the New Package.

7. Re-build the target:

```
$ make
```

8. Rebuild the firmware:

Copy Image and rootfs.ubi to the rtos\linux_image.
And then, rebuild the SDK firmware.

[project dependent]

```
$ cd rtos  
$ make amba_fwprog_clean  
$ make amboot
```

The output firmware is at rtos/output/out/fwprog/bst_bld_sys_dsp_rom_lnx_rfs.elf
[project dependent].

6.6 Customize AmbaLink SDK: Wi-Fi Network Packages

This section is divided into the following sections:

- [\(Section 6.6.1\) Wi-Fi Packages: Add/Remove Wi-Fi Network Driver Packages](#)
- [\(Section 6.6.2\) Wi-Fi Packages: Enable/Disable Ambarella Network Turnkey Scripts](#)

6.6.1 Wi-Fi Packages: Add/Remove Wi-Fi Network Driver Packages

To add or remove a vendor-specific Wi-Fi network driver package, select from the options listed under the Ambarella external packages menu, as shown in [Figure 6-4](#). It is recommended that only one Wi-Fi driver package be chosen for a given build. In addition, after adding or removing Wi-Fi network package elements, a clean build is recommended.

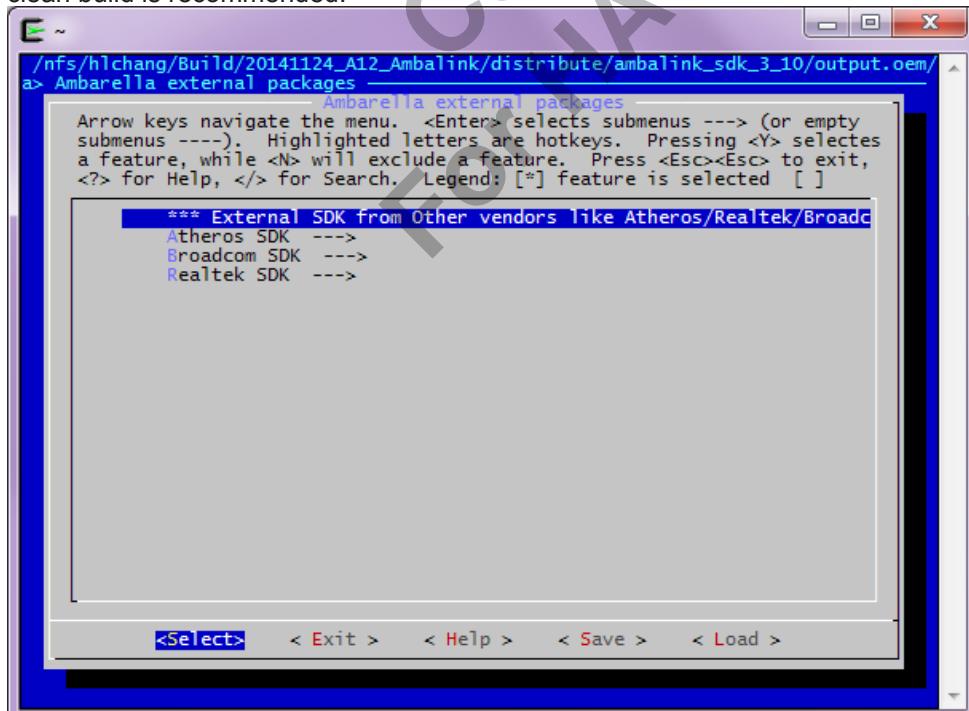


Figure 6-4. From **menuconfig > Ambarella external packages** Choose the Wi-Fi Package.

6.6.2 Wi-Fi Packages: Enable/Disable Ambarella Network Turnkey Scripts

Ambarella provides a number of automated turnkey scripts to manage the Wi-Fi network configuration. The source files for these script packages are located under `ambalink_sdk_3_10/pkg/network_turnkey/`. These packages can be enabled or disabled from menuconfig by selecting items under the Local packages menu, as shown in [Figure 6-5](#).

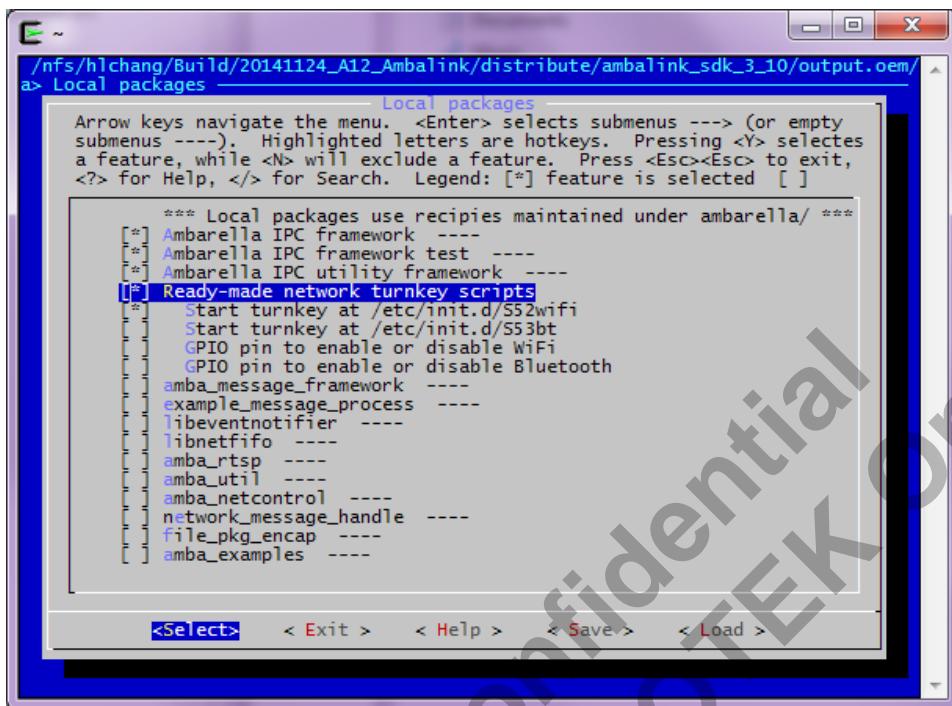


Figure 6-5. From `menuconfig > Local packages` Choose `Ready-make network turnkey scripts Package`.

6.7 Customize AmbaLink SDK: Rebuild Package

After a dedicated package is modified, the package must be rebuilt.

For example, after modifying the `lu_example_util` package, the following steps should be performed:

1. Change to the correct directory:

```
$ cd ambalink_sdk_3_10/output.oem/a12_ambalink
```

2. Re-build the package:

```
$ make lu_example_util-dirclean  
$ make lu_example_util
```

3. Re-build the Linux image:

```
$ make
```

4. Rebuild the firmware:

Copy Image and rootfs.ubi to the rtos\linux_image. [project dependent]
And then, rebuild the SDK firmware.

```
$ cd rtos  
$ make amba_fwprog_clean  
$ make amboot
```

The output firmware is at rtos/output/out/fwprog/bst_bld_sys_dsp_rom_lnx_rfs.elf [project dependent].

6.8 Customize AmbaLink SDK: Store Configurations

This section provides step-by-step information for saving customer-defined configuration settings.

- [\(Section 6.8.1\) Save Configuration Settings: Update Whole System Configuration](#)
- [\(Section 6.8.2\) Save Configuration Settings: Update Linux Configuration](#)

6.8.1 Save Configuration Settings: Update Whole System Configuration

1. Change to the directory shown below:

```
$ cd ambalink_sdk_3_10/output.oem/a12_ambalink
```

2. Apply the desired custom configuration settings. Save the result to defconfig as shown below.

```
$ make savedefconfig
```

3. Copy the file to the location shown below and rename the file customer_defconfig. This represents the save file used for all customer-defined system configuration settings.

```
$ cp defconfig ../../ambarella/configs/customer_defconfig
```

6.8.2 Save Configuration Settings: Update Linux Configuration

The AmbaLink SDK provides two methods to save configuration settings for Linux only. Both methods are outlined below.

A. Method A:

1. Change to the directory shown below:

```
$ cd ambalink_sdk_3_10/output.oem/a12_ambalink
```

2. Copy the ambalink_sdk_3_10/output.oem/a12_ambalink/build/linux-custom/.config file to the following location:

ambalink_sdk_3_10/linux/arch/arm/configs/ambarella_a12_ambalink_defconfig

```
$ make linux-update-config
```

B. Method B:

1. Change to the directory shown below:

```
$ cd ambalink_sdk_3_10/output.oem/a12_ambalink
```

2. Generate a defconfig file for the custom Linux kernel configuration. The file location will be:

ambalink_sdk_3_10/output.oem/a12_ambalink/build/linux-custom/defconfig

```
$ make linux-savedefconfig
```

3. Copy the ambalink_sdk_3_10/output.oem/a12_ambalink/build/linux-custom/defconfig file to:

ambalink_sdk_3_10/linux/arch/arm/configs/ambarella_a12_ambalink_defconfig

```
$ make linux-update-defconfig
```

or

```
$ cp build/linux-custom/defconfig ../../linux/arch/arm/configs/linux_customer_defconfig
```

7 Build the Pure Linux SDK

7.1 Build the Pure Linux SDK: Overview

This chapter includes step-by-step instructions for building the Pure Linux SDK package. These instructions assume that the Linux Build environments have been correctly prepared according to instructions provided in [Chapter 3 “Build Environment”](#) of this document.

The chapter is organized as follows:

- [\(Section 7.2\) Build the Pure Linux SDK: Prepare the Linux Binary Image](#)
- [\(Section 7.3\) Build the Pure Linux SDK: Build the Firmware](#)
- [\(Section 7.4\) Build the Pure Linux SDK: Burn Firmware](#)

7.2 Build the Pure Linux SDK: Prepare the Linux Binary Image

Follow the steps below to build the pure Linux binary image.

1. Untar the following SDK packages into a working directory: (Let’s use the A12SDK for an example)

```
$ mkdir project_name; cd project_name  
$ tar zxf ambalink_sdk_3_10.20141201.tar.gz
```

The output for an `ls` command should read as follows:

```
$ ls  
ambalink_sdk_3_10 ambalink_sdk_3_10.20141201.tar.gz
```

Prepare suitable WIFI driver and untar it under `ambalink_sdk_3_10\external_sdk`
\$ cd ambalink_sdk_3_10
\$ mkdir external_sdk

Please copy `ar6004.tar.gz` into `external_sdk`

[project dependent]

```
$ cd external_sdk  
$ tar zxf ar6004.tar.gz
```

2. Build the pure Linux image:

```
$ cd ambalink_sdk_3_10/ambarella  
$ make O=../output.oem/a12_purelinux a12_purelinux_defconfig  
$ cd ../output.oem/a12_purelinux/; make
```

After the build process completes, the output for an `ls` command should read as follows:

```
$ ls images  
image linuxfs.bin.pref.ubifs rootfs.tar rootfs.tar.gz rootfs.ubi  
rootfs.ubifs
```

The images used for the Linux Kernel and the Root File System partition are `Image` and `rootfs.ubi` respectively.

7.3 Build the Pure Linux SDK: Build the Firmware

Follow the steps below to build the firmware. The toolchain for this is bare-metal gcc using by ThreadX. Please check “[\(Section 3.2.6.2\) Environment: Toolchain for ThreadX](#)”.

1. Build the pure Linux firmware using **AmbaTools**:

```
$ cd ../../ambarella  
$ mkdir -p bootloader/out/a12evk/images/
```

Copy `Image` and `rootfs.ubi` to `ambarella/bootloader/out/a12evk/images/`
\$ cp ../output.oem/a12_purelinux/images/Image bootloader/out/a12evk/images/
\$ cp ../output.oem/a12_purelinux/images/rootfs.ubi bootloader/out/a12evk/images/

```
$ cd bootloader/boards/a12evk  
$ source ../../amboot/config/build_amboot.env
```

```
$ make sync_build_mkcfg (the first time)
```

Please modify `bsp/a12evk.dts` and change the `uart0` status to “ok”.

```
$ vi bsp/a12evk.dts
```

Please refer to [Figure 7-1](#).

```
$ make a12evk_amboot_only_config  
$ make
```

```
/*
 * Copyright (C) 2013 Ambarella, Inc. - http://www.ambarella.com/
 * Author: Cao Rongrong <rrcao@ambarella.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */
/dts-v1;
/include/ "ambarella-a12.dtsci"
{
    model = "Ambarella A12 Dragonfly Board";
    compatible = "ambarella,hawthorn", "ambarella,s21";
    chosen {
        linux,stdout-path = &uart1;
    };
    apb@e8000000 {
        /* if building a12 pure linux, please MUST change the status to ok
        . */
        uart0: uart@e8005000 {
            status = "ok";
        };
        i2c0: i2c@e8003000 {
            status = "disabled";
        };
        i2c1: i2c@e8001000 {
            status = "disabled";
        };
    };
}
"bsp/a12evk.dts" 146L, 2469C
1,1
Top
```

Figure 7-1. Change the `uart0` status.

2. After the build process completes, the firmware can be found at the following location:

```
..../out/a12evk/images
```

The output for an `ls` command should read as follows:

```
$ cd ..../out/a12evk/images
$ ls

bld_release.bin bld_release.elf bst_bld_secondary_lnx_release.bin
bst_bld_secondary_lnx_release.elf bst_release.bin bst_release.elf
dtb_debug.elf Image lnx_release.bin lnx_release.elf rootfs.ubi
secondary_release.bin secondary_release.elf
```

7.4 Build the Pure Linux SDK: Burn Firmware

This section describes the steps required to burn the pure Linux SDK firmware to the NAND flash memory, please refer to [\(Section 5.6\) Build SDK6: Burn the Firmware](#).

8 SDK6 Toolchain and C/C++ Libraries

GCC toolchain provides many features in C libraries. Ambarella SDK6 links some necessary libraries. However, not all functions provided by the toolchain C libraries works or be verified. This document describes the restrictions.

8.1 SDK6 Toolchain and C/C++ Libraries: Toolchain and C libraries

The GCC toolchain is from Linaro for ARM embedded. Please check <https://launchpad.net/gcc-arm-embedded> for more details.

The C libraries used by toolchain is newlib. Please check <https://sourceware.org/newlib/docs.html> for more details.

The newlib library is a collection of software from several sources. Please refer to `COPYING.NEWLIB` of newlib source for more details.

Because ThreadX is not a POSIX compatible operation system, here are some notes while using these C libraries on the Ambarella platform.

8.2 SDK6 Toolchain and C/C++ Libraries: Libraries from Toolchain Used by SDK

- `libnosys.a`: It is the stub of POSIX system call. Customers can also use their own implementation.
- `libc.a`: Some string and memory operation APIs, e.g. `vprintf()`, `memcpy()`
- `libm.a`: Some mathematical operation for image processing, e.g. `cos()`, `pow()`
- `libgcc.a`: Runtime ABI APIs, C++ exception handle
- `libstdc++`: C++ supported

It is recommended NOT to use system-call related APIs. But somehow customers have requirements to customize them. Ambarella collects some samples at “`vendors/foss/syscall/`”.

8.3 SDK6 Toolchain and C/C++ Libraries: POSIX System Call

There is no POSIX system call supported by ThreadX. Therefore, `libnosys.a` is used for the stub APIs.

Since these low-level APIs are stubs only, high-level APIs, e.g. `system()`, `exit()`, `fseek()`, `tmpfile()`, `tzset()`, etc. will not work well.

Customers can link their own implementation to replace the APIs.

Please check <https://sourceware.org/newlib/libc.html#Syscalls> for more details.

These APIs can be found at `newlib/libgloss/libnosys` in newlib source codes, they include:

Task Related									
<code>_env</code>	<code>environ</code>	<code>_execve()</code>	<code>_exit()</code>	<code>_fork()</code>	<code>_getpid()</code>	<code>_kill()</code>	<code>_wait()</code>		
I/O, Filesystem Related									
<code>_chown()</code>	<code>_close()</code>	<code>_fstat()</code>	<code>_isatty()</code>	<code>_link()</code>	<code>_lseek()</code>	<code>_open()</code>	<code>_read()</code>	<code>_readlink()</code>	<code>_stat()</code>
Memory-Management Related									
<code>_sbrk()</code> : The default stub API will work if the linker script provides the “end” symbol and a heap region.									
Time Related									
<code>_ gettimeofday()</code>	<code>_times()</code>								

Table 8-1. APIs in Newlib Source Codes.

8.4 SDK6 Toolchain and C/C++ Libraries: Newlib C library (libc.a)

This section explains the following aspects:

- (Section 8.4.1) Newlib C library (libc.a): Reentrant Issue
- (Section 8.4.2) Newlib C library (libc.a): Standard Utility Functions (stdlib.h)
- (Section 8.4.3) Newlib C library (libc.a): Input and Output (stdio.h)

8.4.1 Newlib C library (libc.a): Reentrant Issue

For reentrant issue, please refer to <http://www.billgatliff.com/newlib.html>.

Note that some older SDKs may not have this fixed, please contact Ambarella's representative for updates.

8.4.2 Newlib C library (libc.a): Standard Utility Functions (stdlib.h)

`malloc()` will work if a heap region is assigned for `sbrk()`. Usually, `sbrk` heap region is small and `malloc()` or its low level call `_malloc_r()` is used internally by newlib.

Therefore, customers MUST NOT use these standard APIs, e.g. `malloc()`, `free()`. Instead, customers MUST use APIs defined in `vendors/ambarella/inc/ssp/AmbaKAL.h`.

The heap used by `malloc()` is from “`_sbrk_start`” to “`_sbrk_end`” defined in `.lds.S` file. This section MUST be behind “`end`” which means “end of text section”. The `_sbrk()` in `libnosys.a` use this symbol as the start of heap allocation.

Note that, it is necessary to take care of the MMU setting for the heap buffer.

Customers can also overwrite `_sbrk()`, it is defined as the weak function in newlib. Then, the memory used by `malloc()` could be the same as ThreadX applications.

`__malloc_lock()` and `__malloc_unlock()` are used for the critical section guard when `malloc()` is called. However, the toolchain is built for single thread, so these two APIs are the stubs only.

Ambarella SDK wrap these functions during linking stage.

(Note that some older SDK may not have this fix, please contact Ambarella's representative for updates)

The commands to wrap symbol at linking stage can refer to `link-amba_XXX.sh` and check “`--wrap symbol`” option of the GNU ld manual.

8.4.3 Newlib C library (libc.a): Input and Output (stdio.h)

I/O, filesystem operations does not work in the newlib of Ambarella SDK.

Newlib does not support filesystem operation well. Customers MUST NOT use standard APIs, e.g. `open()`, `close()`, `read()`, `write()`. Instead, Customers MUST use APIs defined in `vendors/ambarella/inc/ssp/fs/AmbaFS.h`.

It is the same for the I/O operation, e.g. `printf()`, `scanf()`, will not work well. Customers MUST use `AmbaPrint()` instead.

8.5 SDK6 Toolchain and C/C++ Libraries: Newlib C math library (libm.a)

It supposes to work since there is no I/O, file system operations and system to call it in. However, Ambarella does not test all those functions.

8.6 SDK6 Toolchain and C/C++ Libraries: SDK6 C++ Support and Limitation

Ambarella RTOS SDK, which is built by bare-metal GCC supports C++ with some features unavailable. The description below shows how to enable C++ support and the limitation of C++ support.

1. Enable C++ support:

`CONFIG_CC_CXX_SUPPORT` is used to enable C++ support in RTOS SDK.

Add files using `Makefile` and for those files with “`.cpp`” extension, compile using `arm-none-eabi-c++`.

```
$ cd rtos
$ make XXX_defconfig (based on what the user actually uses)
$ make menuconfig
```

Go to GCC Option and enable Support C++ files. Then, save it.

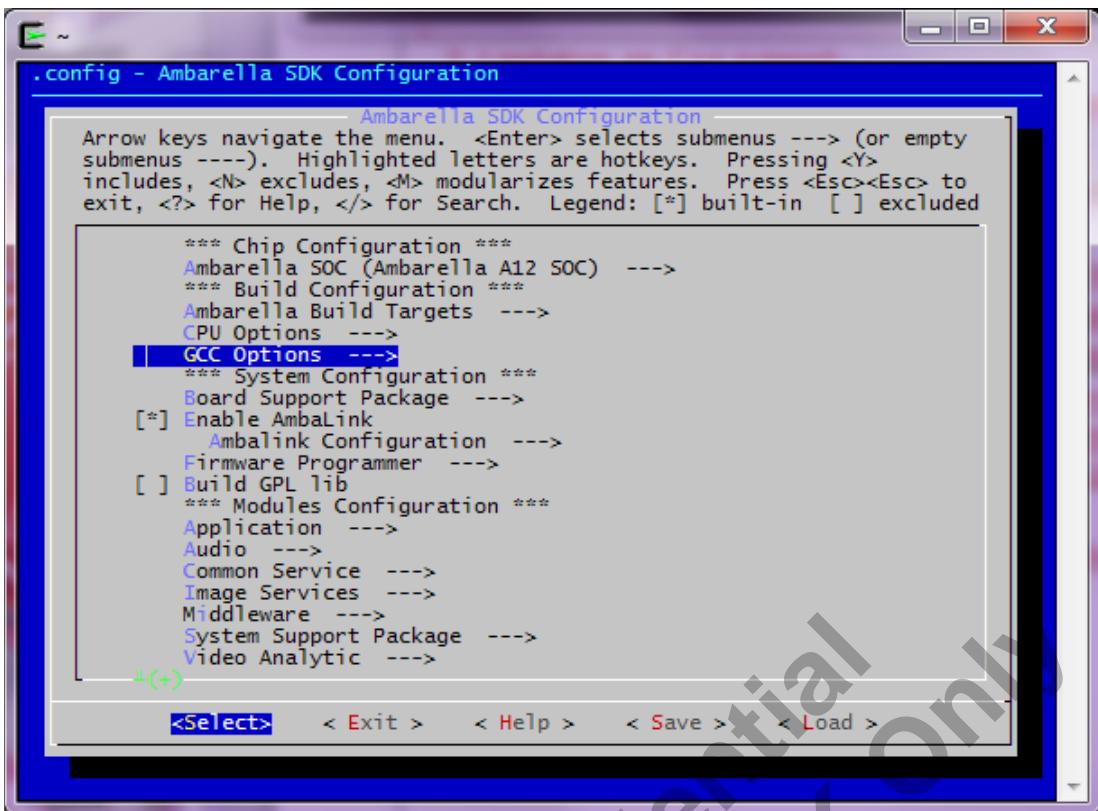


Figure 8-1. Enable C++ Support: GCC_Option.

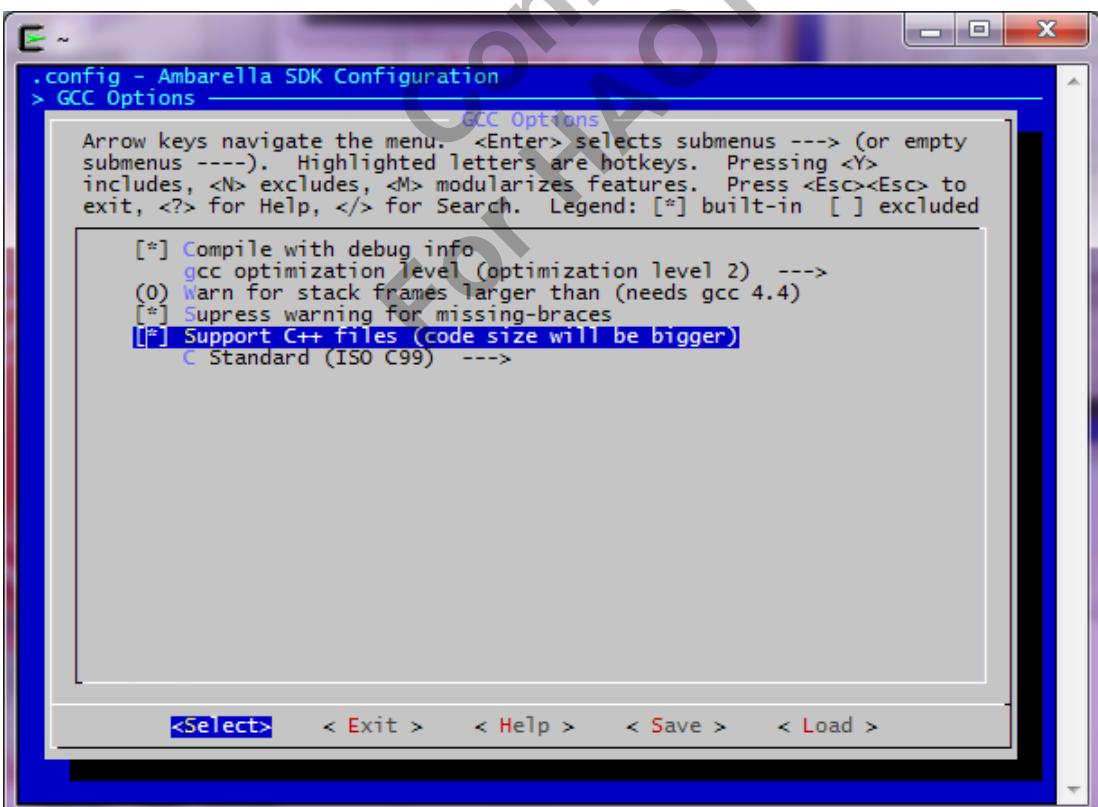


Figure 8-2. Enable C++ Support: Support_C++.

After using the settings in [Figure 8-1](#) and [Figure 8-2](#), please use the following command.

```
$ make
```

2. Limitation on C++ support:

There are some C++ runtime features unavailable when passing some flags to the C++ compiler.

- Exceptions
 - Ambarella does not port unwind library to support C++ exception.
 - Disable it by “`-fno-exceptions`” flag.
- RTTI (Run-Time Type Information)
 - Attempt to use either exceptions or RTTI in a C++ kernel needs the `libsupc++` library to be cross-compiled.
 - Ambarella uses pre-built Linaro bare-metal toolchain and disables it using the “`-fno-rtti`” flag.
- C++ destructors are registered using `__cxa_atexit()`
 - The C++ Standard requires that the destructors are called for global objects when a program exits in the opposite order of construction. RTOS is not like the Linux environment where destructors should be called when a shared library needs to be unloaded. Disable it using the “`-fno-use-cxa-atexit`” flag.
- STL (Standard Template Library)
 - STL needs porting to work on an OS. Ambarella does not port a STL implementation to RTOS. Thus STL functions or classes required OS porting do not work. However, STL functions or classes required OS porting could work, for example, `class std::vector`.

9 Troubleshooting

9.1 Troubleshooting: Overview

This chapter provides troubleshooting tips and additional information. Please see the following sections for more details.

- (Section 9.2) Troubleshooting: Select ThreadX Toolchain

9.2 Troubleshooting: Select ThreadX Toolchain

This section describes how to select pre-installed toolchain from the user's build environment, especially when more than one toolchain is installed in the user's environment.

ThreadX toolchain is chosen from the linaro Bare-metal toolchain, which can be downloaded from <https://launchpad.net/gcc-arm-embedded>.

Ambarella SDK could upgrade the support for the newer toolchains.

Users may need to know how to use the right toolchain version in the Linux build environment.

9.2.1 Select ThreadX ToolChain: Select Toolchain

The following are two methods to select the right toolchain once it is installed properly.

Assume it is installed to a path like /usr/local/gcc-arm-none-eabi-xxxx, where xxxx represents the toolchain version.

9.2.1.1 Select ToolChain: Add Toolchain Path in the Startup Script of the Login Shell

Export PATH shown as below in .bash_profile or .bashrc for bash

Example for bash:

```
export PATH=/usr/local/gcc-arm-none-eabi-xxxx/bin:$PATH
```

9.2.1.2 Select ToolChain: Using Script “gcc-arm-none-eabi-env” in SDK

Ambarella provides some simple examples as follows:

- Using latest version of GCC

```
$ source rtos/build/maintenance/gcc-arm-none-eabi-env
```

- Using specific GCC version

For example, using GCC4.7

```
$ source rtos/build/maintenance/gcc-arm-none-eabi-env -v 4.7.4-20130913
```

- Show help (There will be more support versions in the future)

```
$source rtos/build/maintenance/gcc-arm-none-eabi-env -h
```

Please source this script instead of executing it!

Usage:

-v version

version: 4.9.3-20150303 (default version)

version: 4.7.4-20130913

Example:

. gcc-arm-none-eabi-env (source default version)

. gcc-arm-none-eabi-env -v 4.9.3-20150303

. gcc-arm-none-eabi-env -v 4.7.4-20130913

Appendix 1 Additional Resources

Please contact an Ambarella representative for digital copies.

For Confidential
HAOTEK Only

Appendix 2 Important Notice

All Ambarella design specifications, datasheets, drawings, files, and other documents (together and separately, "materials") are provided on an "as is" basis, and Ambarella makes no warranties, expressed, implied, statutory, or otherwise with respect to the materials, and expressly disclaims all implied warranties of noninfringement, merchantability, and fitness for a particular purpose. The information contained herein is believed to be accurate and reliable. However, Ambarella assumes no responsibility for the consequences of use of such information.

Ambarella Incorporated reserves the right to correct, modify, enhance, improve, and otherwise change its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

All products are sold subject to Ambarella's terms and conditions of sale supplied at the time of order acknowledgement. Ambarella warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with its standard warranty. Testing and other quality control techniques are used to the extent Ambarella deems necessary to support this warranty.

Ambarella assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using Ambarella components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

Ambarella does not warrant or represent that any license, either expressed or implied, is granted under any Ambarella patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which Ambarella products or services are used. Information published by Ambarella regarding third-party products or services does not constitute a license from Ambarella to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from Ambarella under the patents or other intellectual property of Ambarella.

Reproduction of information from Ambarella documents is not permissible without prior approval from Ambarella.

Ambarella products are not authorized for use in safety-critical applications (such as life support) where a failure of the product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Customers acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of Ambarella products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by Ambarella. Further, Customers must fully indemnify Ambarella and its representatives against any damages arising out of the use of Ambarella products in such safety-critical applications.

Ambarella products are neither designed nor intended for use in automotive and military/aerospace applications or environments. Customers acknowledge and agree that any such use of Ambarella products is solely at the Customer's risk, and they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

Appendix 3 Revision History

NOTE: Page numbers for previous drafts may differ from page numbers in the current version.

Version	Date	Comments
0.1	9 June 2014	Formatting.
0.2	30 June 2014	Update in Chapter 3, Build environment, adding Chapter 4, Setup EWARM License and Chapter 7, Boot-Up using the JTAG debugger.
0.3	22 July 2014	Updated Chapter 1: Overview, Section 3.3.3.2: Chameleon: Setup, Chapter 5: Build A9 SSP_AmbaLink SDK, Section 6.4: Customize AmbaLink SDK: Remove Ambarella Add-Ons, Section 6.9.2: Save Configuration Settings: Linux-Only Configuration, and Chapter 7: Boot-Up Using the I-JET Debugger Removed Chapter on Build Pure Linux SDK
	30 July 2014	Rearranging figures.
0.4	19 August 2014	Update in Section 5.5. Add Chapter 8: Troubleshoot.
0.5	12 December 2014	Formatted to SDK6, Updated Chapter 1 Overview, Chapter 2 System Requirements, Chapter 3 Environment, Revise Chapter 4 Purchase I-JET, Revise Chapter 5 Build SDK6, Updated Chapter 6 Customize the AmbaLink SDK and Chapter 7 Build Pure Linux SDK.
0.6	30 April 2015	Update in Section 2.2, Section 2.3, Section 2.4, Section 3.2, Section 3.2.1, Section 5.1 and Section 5.2. Added Section 3.2.4.3 and Chapter 8 SDK6++ Support and Limitation.
0.7	8 May 2015	Updated Section 5.2 and 7.2.
0.8	1 July 2015	Update in Chapter 1 Overview, Section 1.5.2, 3.2.6.2 Environment: Tool-chain for ThreadX, Section 5.2, Section 6.4 and Section 7.2. Add Chapter 9 Troubleshooting.
0.9	13 July 2015	Update in Section 1.5. Add Section 1.5.4 Overview: MultiVIN APP.
1.0	30 July 2015	Update in Section 3.3.2.1 Chameleon: Installation.
1.1	5 August 2015	Update in Section 5.2 Build SDK6: AmbaLink SDK.
1.2	11 August 2015	Update in Section 3.2.4.2 Linux Server: Update Server Packages.
	13 August 2015	Update in Section 5.2 Build SDK6: Build AmbaLink SDK.
1.3	16 September 2015	Update in Section 1.5.4. Add Sections 1.5.5 and 1.5.6.
1.4	10 December 2015	Update in Section 1.1 Overview: Introduction. Delete Section 1.7 Overview: Scope of Document.
1.5	26 February 2016	Update in Sections 3.3, 4.1, 4.3, 3.2.6.1, 5.2, 5.3, 5.4, 5.5, 7.3 and 7.4. Add Sections 3.2.6.2.1 Toolchain for ThreadX: Toolchain for 32 bit ThreadX, 3.2.6.2.2 Toolchain for ThreadX: Toolchain for 64 bit ThreadX , 3.3.3 Windows: CVD64 Setup, 3.3.3.2 CVD64: Driver Setup, 3.3.3.3 CVD64: Setup, 5.6.1 Burn the Firmware with Chameleon: By I-Jet (ARM v7 32 Bit Architecture), 5.6.2 Burn the Firmware with Chameleon: By CodeViser (ARM v8 64 Bit Architecture) and Chapter 8 SDK6 Toolchain and C/C++ Libraries. TroubleShooting becomes Chapter 9.
1.6	24 March 2016	Update in Sections 3.2.6.2.1 and 5.2.

Table A3-1. Revision History.