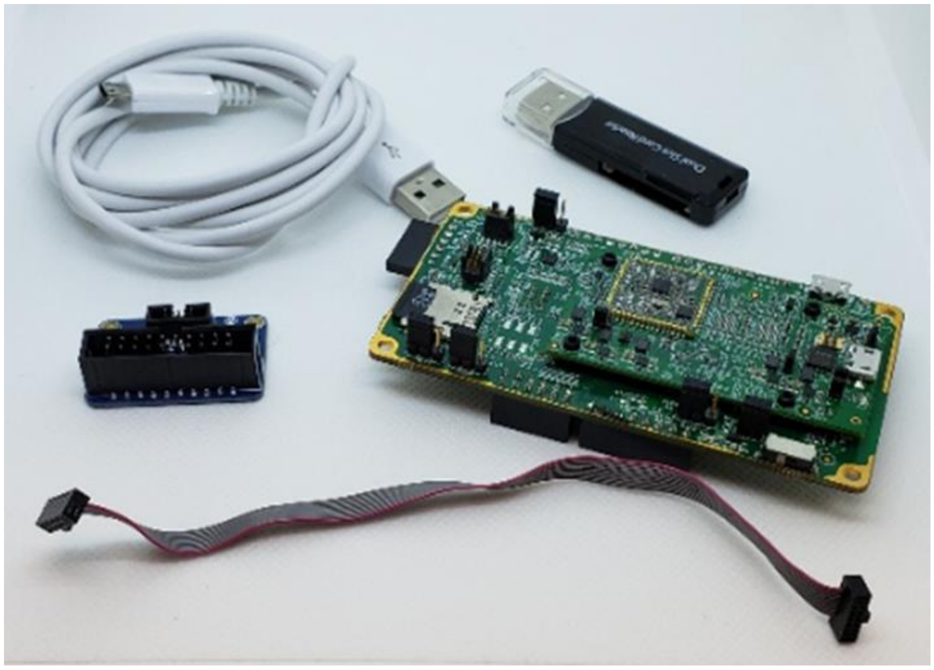
QuickAI: QuickAI SDK

January 22, 2019

Quicklogic EOS-S3-AI

Merced V1.1 Development Board



1 Background

This release of the QuickAI SDK (Software Development Kit) runs on the Merced V1.1 HDK (Hardware Development Kit) and is intended to provide a good starting point for developing software applications targeting the Quicklogic EOS-S3-AI silicon. Directory structure

The code in the QuickAI SDK is organized into several main directories:

```
+---App                       ← All applications are found here.
|   +---BL0                   ← First Boot Loader, in Binary form only
|   +---BL1                   ← Second Boot Loader
|   +---Pre-builts            ← prebuilt Libraries from QuickLogic
|   \---Recognition           ← The recognition example application
+---BSP
|   \---QuickAI               ← The Merced V1.1 board support package
+---Docs                      ← This document is found here.
+---FreeRTOSv10_1_1           ← RTOS used by this SDK.
+---HAL                       ← Hardware Abstraction Layer
+---Libraries
|   +---CMSIS                 ← CortexM Libs & Headers from ARM
|   +---FatFS                 ← Simple FAT file system code
|   +---FreeRTOS_FAT          ← FreeRTOS Fat File System.
|   +---IOP                   ← Used with Nordic BLE Chip
|   +---Power                 ← Power Management
|   +---QLSPI                 ← SPI interface
|   +---QLSRC
|   +---QLUART
|   +---SensorFramework       ← Sensor Framework
|   +---SysFlash              ← Flash
|   +---UARTFlashTool
|   \---Utils
\---Tasks                     ← Tasks that may be used within an app
    +---Audio
    +---BL1
    +---BLE
    +---SDCard
    \---TestUtils
```

2 EOS-S3-AI Boot Sequence for this SDK.

This section discusses from a Software Developers view point how the EOS-S3-AI chip boots. Note that other boot modes are possible (for details see the S3-AI-HDK, or the S3-AI-TRM).

2.1 SPI Flash Memory Map

The Memory map of the SPI flash is as follows:

```
Offset          -> Description

@ 0x0000.0000   -> BL0 Start Location
                -> BL0 Length = 64K
@ 0x0001.0000   -> BL1 Primary Start @ 64K
                -> BL1 Maximum Length = (512K-64K) = 448K
@ 0x0008.0000   -> BL1 Secondary (Backup) Start @ 512K
                -> BL1 Secondary Maximum Length = 448K
@ 0x000f.0000   -> 64K hole, unused.
                -> Length: 64K
@ 0x0010.0000   -> Start of Quasi-FAT File system @ +1Mbyte
                -> Runs until the end of the SPI Flash
```

2.2 Boot Sequence

For this specific SDK and the Merced V1.1 HDK, the QuickAI EOS-S3-AI boots as follows:

Step 0: Initial conditions

The Debug Board has two jumpers (P4 and P5) that control the boot process.

There are two jumper settings of interest, (A) Debug mode and (B) SPI Flash Boot mode.

In DEBUG mode, it allows a JLINK (SWD) type debugger to attach/ control the CPU for debug purposes.

In SPI_FLASH mode is described in Steps 1 to 4.

Step 1: At power-up reset…

At power-up the Boot Pins are read, for the QuickAI Merced board the jumpers cause the device to read 64K bytes from the SPI flash into RAM (More details in the Chip TRM) the image loads at address ZERO.

Important points to note:

- The first code that is loaded from the SPI flash is referred to as BL0 (Boot Loader Zero). The purpose of BL0 is to load either BL1 or enter download/recovery mode.
- If BL0 is corrupt, or not present a JTAG/SWD probe is required to reload the flash image (See the Flash Programing Documentation for details)

- The design intent of BL0 is that it should never be updated, changed or removed. BL0 is the "boot loader of last resort" when all things are corrupt it provides a means to download and write to flash a new flash image.

Step 2: BL0 attempts to find/load BL1

- There are TWO BL1 images the Primary and Secondary image. If something happens to BL0-Primary, BL1-Secondary can be used
- If BL1 is not found, BL0 goes into serial port download, or recovery mode.
- BL0 recovery mode can only download and write BL1 images to the SPI flash

Step 3: BL1 is loaded into RAM and execution of BL1 begins.

BL1 waits about 3 seconds for some serial port activity – If activity occurs, BL1 enters Download mode.

The Quicklogic Flash Tool can be used to download other files and images described below.

Step 4: Application boot begins (no activity for 3 seconds)

Starting at address 1MBYTE in the SPI flash, a quasi-MSDOS-FAT file system is present until the end of the SPI flash. Contained in this file system are the following:

a) A configuration file (ascii text)
b) A binary image of the FPGA configuration file
c) A binary image of the FFE control program image
d) The CortexM4 application image.

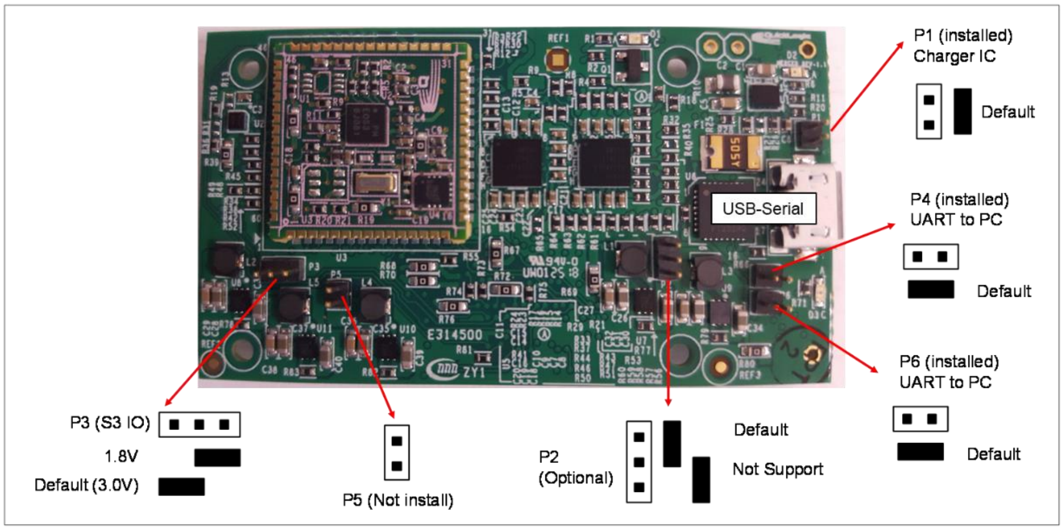
BL1 reads the configuration file, and if configured – BL1 will load the FPGA and/or the FFE image.

Step 5: The BL1 loads the APP

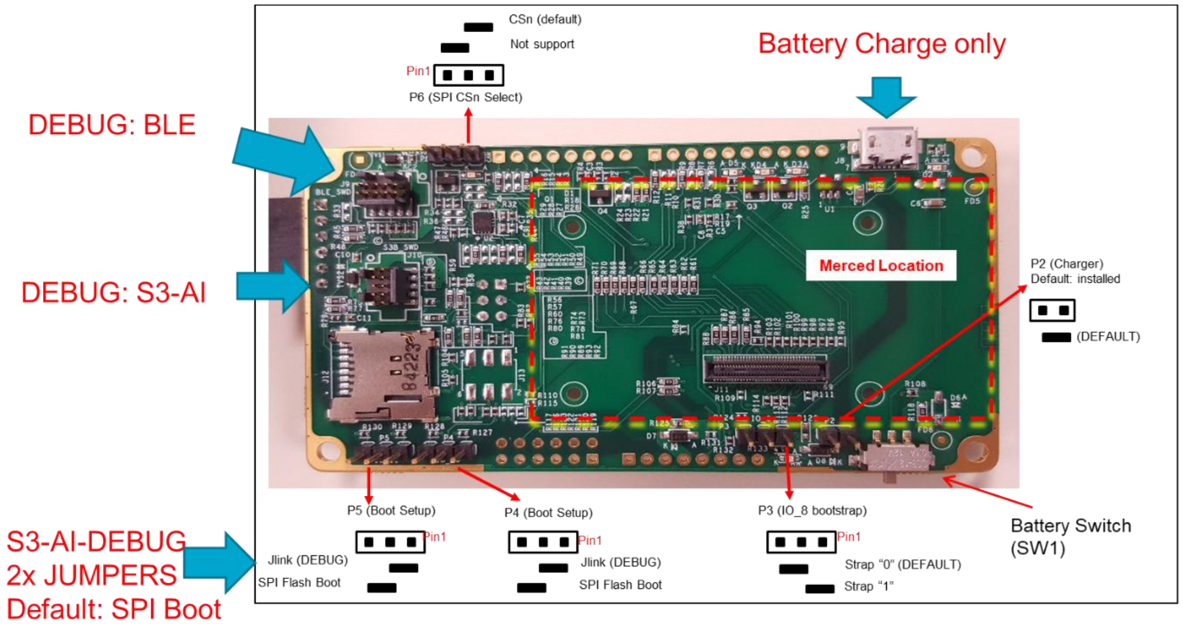
The BL1 image loads the application into RAM and jumps to APP entry point.

3 Merced Jumpers & Connections

3.1 Merced v1.1 Board Jumpers



3.2 Merced v1.1 Debug Board Jumpers



4 Getting Started – Creating and Modifying the Sample App

Using Windows Explorer, copy and paste the Recognition App

Step 1: Right click and copy ${SDK}/App/Recognition

Step 2: Paste, creating: ${SDK}/App/Recognition – Copy

The IAR Project files and Makefiles assume all application are contained here:

${SDK_ROOT/App/<appname_here>

Step 3: Rename "Recognition" to "my_experiment"

5 BL1 – using IAR

The IAR project files can be found here:

${SDK_ROOT}/Apps/BL1/IAR_Project/BL1.eww

6 APP – Using GNU Makefiles

The GNU Makefiles can be found here:

${SDK_ROOT}/Apps/Recognition/GCC_Project

7 APP – Using Eclipse

There are two methods:

${SDK_ROOT}/Apps/Recognition/Eclipse_Project

This uses the standard Eclipse CDT cross compilation techniques.

${SDK_ROOT}/Apps/Recognition/GCC_Project/Eclipste_Make_Project

This uses the standard Eclipse External Makefile complation techniques.

8 Notes on Eclipse on Windows Platforms

This note applies to Windows platforms only; MAC and Linux platforms do not exhibit this problem. In short: We strongly suggest placing your source code and Eclipse workspaces in a 'shallow' or 'short-path' location.

Safe example: `C:\work\ProjectFoo\<project-contents>`

Bad Example: `C:\some\place\with\a\very\deep\path\<project-content>`

Note that <project-content> can either be the project itself or the Eclipse Workspace.

Background: Microsoft Windows has two APIs for all file system operations. A simple ASCII interface, and a Unicode interface. The ASCII interface has a hard limited to file path names of 260 characters, where as the Unicode interface supports filename paths of 32K bytes.

It would be nice if every SW tool in the ecosystem exclusively use the windows Unicode API, but they do not; in fact, some tools are internally inconsistent and there is no simple fix.

The best solution is avoidance by avoiding deeply nested directory structures.

9 Limitations of this Release:

In this release:

BL1 only supports the IAR Toolchain

The Recognition App – supports only the Eclipse build process described above.

Future releases will support both IAR and Eclipse uniformly.

END