TLSR9x Axon Software Development Kit User Guide

06/23/2022
Revision 1.0

Atlazo Inc
4250 Executive Square, Suite 675
La Jolla, CA 92037

TABLE OF CONTENTS

Revision History

Version	Date	Description
1.0	*06/23/2022*	Initial document

PREFACE

This document provides information on installing and using the TLSR9X Axon Software Development Kit.

REFERENCES

- *Axon Driver User Guide*
- *Axon TensorFlow User Guide*

ACRONYMS AND DEFINITIONS

BLE – BlueTooth Low Energy

IDE – Integrated development environment.

ML – Machine Learning

1 SDK DESCRIPTION

The TLSR9x Axon SDK is a group of projects that is imported into the AndeSight IDE (the RISC-V CPU core is provided by Andes Technology). The focus of this SDK is use of the Axon neural processing engine.

1.1 IDE TERMINOLOGY

The TLSR9X AXON SDK is delivered as a series of projects that are imported into the AndeSight IDE. The IDE is based on Eclipse, so some Eclipse terminology is used.

A *project* is a top-level node in the project explorer. It includes one or more of the following: source code files, header files, pre-compiled libraries. A project can be compiled into a static library or an elf (executable file).

A *project* can include header files from another project.

A *build configuration* describes how a single build artifact (static library or elf) is built within a project.

Build configurations can be used to customize build artifact type, tool settings (debug, release, optimizations, etc), include paths, linked libraries, and excluded source directories.

The TLSR9X AXON SDK makes extensive use of projects and build configurations to provide over 30 different demo applications.

TLSR9X AXON SDK project names follow a basic naming condition:

- Projects whose names begin with *core_* produce executable elf files (and the corresponding binary files that are written to device flash).
- Projects whose names end in *_lib* produce static libraries that are linked in by some of the build configurations of the *core_* projects.

1.2 TOP LEVEL PROJECTS

Top level projects produce an executable image binary that can be burned into device flash memory. These projects will

There are three top-level projects in the SDK that produce elf files, *core_drivers*, *core_drivers_ble*, and *ml_ble_platform*.

These projects each include the core start-up code, peripheral drivers, and interrupt vectors.

These projects have build configurations that produce different elf files for demonstrating usage of different features in the SoC.

Project *core_drivers*:

- focuses on internal peripheral usage (there are over 20 separate build configurations)

- Supports logging over a serial port.
- does not support Blue Tooth Low Energy.

Project *core_drivers_ble*:

- includes all the peripheral drivers as *core_drivers*, but without the explicit examples provided by *core_drivers*.
- includes Blue Tooth Low Energy stack.
- Provides BLE use case examples.
- Does NOT support logging over a serial port.
- Includes a static library build config that is used by ml_ble_platform.

Project *ml_ble_platform*:

- Receives its peripheral drivers, start-up code, and ble stack in static library from *core_drivers_ble*.
- Includes platform management code for system time, BLE communications with a host app, sensor and audio interfaces.
- Implements high level use-cases with power management for sensors (IMU, PPG) and audio key word spotting.
- ***NOTE: the standard evaluation kit does not include IMU or PPG sensors, these must be attached separately.***

1.3 AXON PROJECTS

Axon projects focus on the Axon machine-learning accelerator, which is one of the major differentiating features of TLSR9X. These projects compile into static libraries and are linked into a core project to form an Axon use case.

Axon Project Name/ Build Configuration	**Description**	**Dependencies**	**linked in Elfs**
axon_audio_fc4_lib/ fc4_32_ms_windows	Implements FC4 key word spotting model trained for 32ms audio window slices. Generated from axon model precompiler	axon_driver_lib, axon_audio_ml_lib	core_drivers: •axon_app_kws_fc4_g12_btn_trigger ml_ble_platform: •audio_kws_fc4_btn_trigger
axon_audio_features_lib/ debug	Implements audio feature extraction (MFCCs).	axon_driver_lib	core_drivers: •axon_app_kws_fc4_g12_btn_trigger •axon_app_kws_grnn_g12_btn_trigger ml_ble_platform: •audio_kws_fc4_btn_trigger •audio_kws_grnn_btn_trigger

axon_audio_framework_lib/ button_press_trigger	Implements user button and audio codec control functions, and top-level logic for dispatching audio windows.	axon_audio_ml_lib	core_drivers: •axon_app_kws_fc4_g12_btn_trigger •axon_app_kws_grnn_g12_btn_trigger ml_ble_platform: •none
axon_audio_framework_lib/ ble_sdk_debug			core_drivers: •none ml_ble_platform: •audio_kws_fc4_btn_trigger •audio_kws_grnn_btn_trigger
axon_audio_grnn_lib/ grnn_g12_debug	Implements grnn key word spotting model trained for 32ms audio window slices.	axon_driver_lib, axon_audio_ml_lib	core_drivers: •axon_app_kws_grnn_g12_btn_trigger ml_ble_platform: •audio_kws_grnn_btn_trigger
axon_audio_lstm_ib/ debug	Implements lstm key word spotting model trained for 32ms audio window slices. Generated from axon model precompiler		core_drivers: •axon_app_kws_lstm_audio_g12_btn_trigger
axon_audio_ml_lib/ kws_fc4_g12_debug	Dispatches audio samples to audio feature and inference model as necessary.	axon_audio_features_lib, axon_audio_grnn_lib, axon_audio_fc4_lib	core_drivers: •axon_app_kws_fc4_g12_btn_trigger ml_ble_platform: •audio_kws_fc4_btn_trigger
axon_audio_ml_lib/ kws_grnn_g12_debug			core_drivers: •axon_app_kws_grnn_g12_btn_trigger ml_ble_platform: •audio_kws_grnn_btn_trigger
axon_demo_lib/debug	Basic axon usage demo.	axon_driver_lib	core_drivers: •axon_app_base_demo ml_ble_platform: •none
axon_driver_lib/Release	Low-level axon driver, delivered in binary form.	None	All axon-related demos link to this library.
axon_utils/debug	Basic logging functions	None	All

core_drivers_ble/ ble_driver_lib	Peripheral drivers, startup code, and BLE stack compiled into a library.	None	core_drivers: •none ml_ble_platform: •all

1.4 AXON CORE_DRIVERS APPLICATIONS

Axon core_drivers applications are created by defining a build configuration in core_drivers that excludes all source folders directories except axon_app and common. The axon_app source files provide basic start-up code, ISRs, and a simple API to invoke an application use case.

The primary application logic is implemented in separate libraries. The axon application build configuration determines which application it runs by specifying which libraries to link and where to find them.

1.4.1 axon_app_base_demo

Its application logic is implemented in project axon_demo_lib. It also links in axon_driver_lib.

This demo shows basic examples of using each of the axon APIs, and in different modes of operation.

See the document *Axon Driver User's Guide* for further information on axon APIs and operation modes.

Results are printed out the debug serial port.

1.4.1.1 Usage

Application delays 10s at start up then prints out log messages indicating progress.

1.4.2 axon_app_kws_grnn_g12_btn_trigger

(Note, this was renamed from axon_app_audio_ml_demo in SDK version 1.1)

Its application logic is implemented axon_audio_framework_lib.

This application demonstrates background/asynchronous execution of audio processing and algorithm execution for keyword spotting, using a fast GRNN type model. The basic call flow is as follows:

- Button press event handled in user/thread context, initiates audio recording and a timer for monitoring audio (in lieu of audio DMA interrupt). CPU enters WFI.
- Timer interrupt expires every 16ms to submit a slice of audio for feature extraction. This code executes in the timer interrupt context.
- Axon interrupt fires whenever it needs servicing. More work is submitted to Axon in the axon interrupt context until the audio feature extraction has completed.
- When enough audio slices have been processed, the inference process is started (again, in the interrupt context).
- Each interrupt (audio or axon) wakes the CPU from WFI and the code checks for any state changes (like recording complete, inference complete, etc), then re-enters WFI.

This demo uses the axon accelerator to extract audio features (spectrogram) as will as perform keyword spotting inference (using a FastGRNN model).

Results are printed out the debug serial port.

1.4.2.1 Usage

Audio is recorded using the left analog microphone.

Pressing then releasing the button on SW2 starts recording.

The user has two seconds to speak a keyword (one of on, off, yes, no, stop, go, up, down, left, right). The keyword needs to be spoken separately as a command. It cannot be part of a sentence nor combined with other words .

Recording is active while the blue LED is lit.

If a valid window of audio is detected, the green LED will light while classification occurs. The classification results are then printed to the debug console.

Note: the recorded audio can be played out of Line Out 1 (J24) by pressing the button on SW5. The playback buffer is only 1.5s so the playback quality is best if the audio is recorded early in the 2s window. Pressing SW5 will also cause the raw audio samples (last 1.5s) to be dumped out the debug console as comma separated int16's.

1.4.3 axon_app_kws_fc4_g12_btn_trigger

This application is identical to axon_app_kws_grnn_g12_btn_trigger except it uses a full-connected (fc4)model instead of the fastGRNN model.

1.4.4 axon_app_kws_lstm_audio_g12_btn_trigger

This application is identical to axon_app_kws_grnn_g12_btn_trigger except it uses a lstm model instead of the fastGRNN model.

1.5 AXON ML_BLE_PLATFORM APPLICATIONS

All ml_ble_platform applications include the BLE stack, and support proprietary BLE services for sending sensor data, data stored in FLASH, and a user console. A separate Windows host application is used to communicate with the device.

These applications utilize TLSR9X's ultra-low power mode "deep sleep", which saves power but also only retains 64KB (out of 256KB) while in deep sleep.

The BLE stack manages the entry into deep sleep; the interval is determined by the BLE connection parameter "max latency" (set to 1 second).

Power management code provides a voting mechanism to enable/disable deep sleep depending on what functions are active.

1.5.1 audio_kws_fc4_btn_trigger

This application enables the same keyword spotting model as the similarly named application under core_drivers. Usage is the same, except the inference results are displayed over the BLE console, not a serial console.

1.5.2 audio_kws_grnn_btn_trigger

This application enables the same keyword spotting model as the similarly named application under core_drivers. Usage is the same, except the inference results are displayed over the BLE console, not a serial console.

1.5.3 sensor_base

This application provides a platform for support external sensors (MAX141 PPG, BMI160 IMU) and logging raw sample values over BLE. Note the base evaluation kit does include these sensors; these must be obtained and connected separately.

2 INSTALLATION

The TLSR9X AXON SDK is a series of projects imported into the AndeSight IDE. The instructions below are for the following setup:

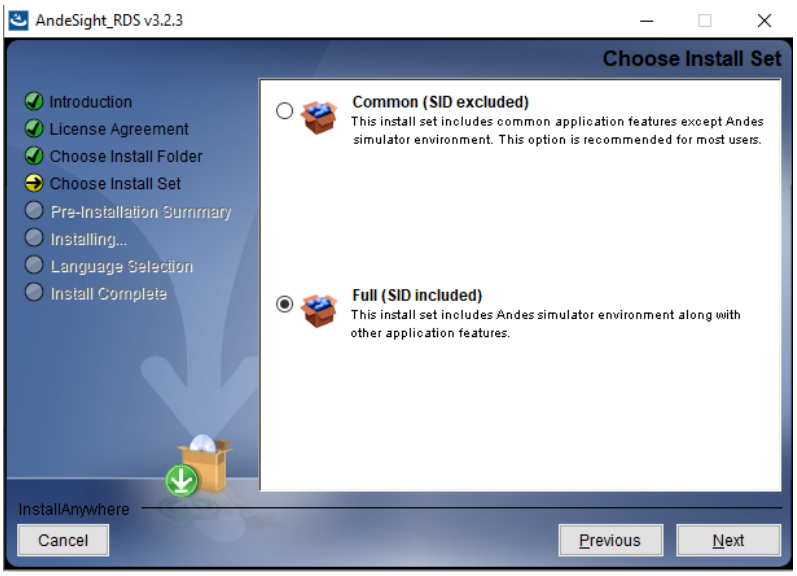
- Source and project files are in the folder C:\Atlazo\TLSR9X-AXON-SDK
- The AndeSight workspace is in the folder C:\Atlazo\ TLSR9X-AXON-SDK-workspace
- The projects are not copied to the workspace.

This is not the only way to set up the workspace, but this is one way to keep workspace files from polluting a source repository.

2.1 ATLAZO/ANDESSIGHT IDE

After installing AndeSight, it will appear as “Atlazo_RDS” in the Windows start menu.

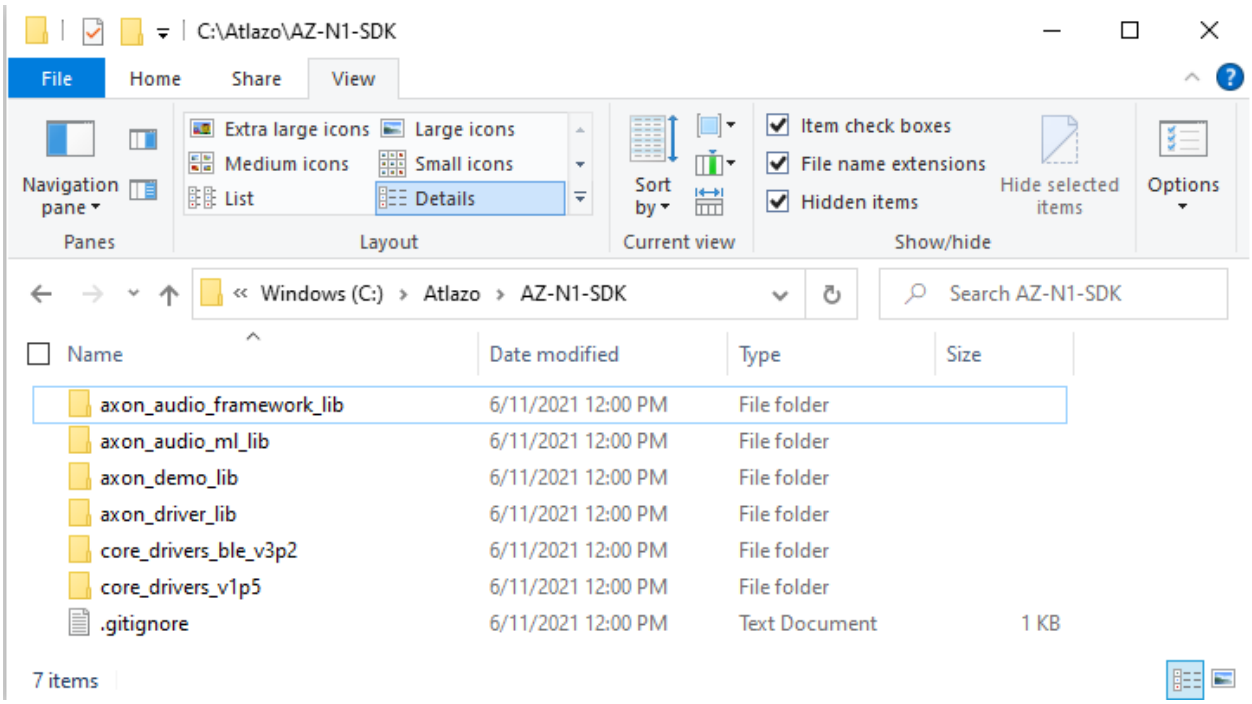
1. Download and unzip the Atlazo/AndeSight IDE.
2. Launch Setup.exe under <unzipped_root>windows/disk1
    a. The simulator is not required but can be installed:



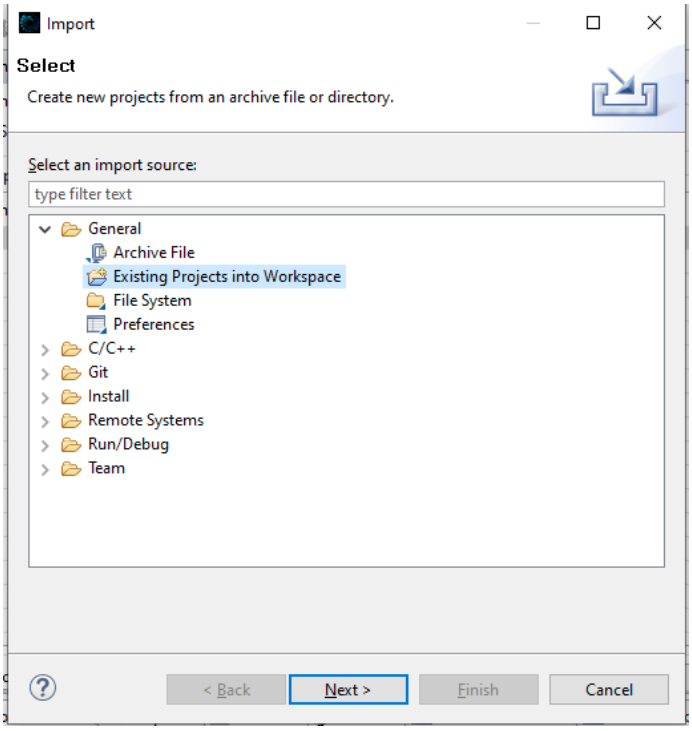
3. Accept all the installation defaults.
4. Install and AndeSight IDE license; this file is provided separately from the IDE.

2.2 SDK

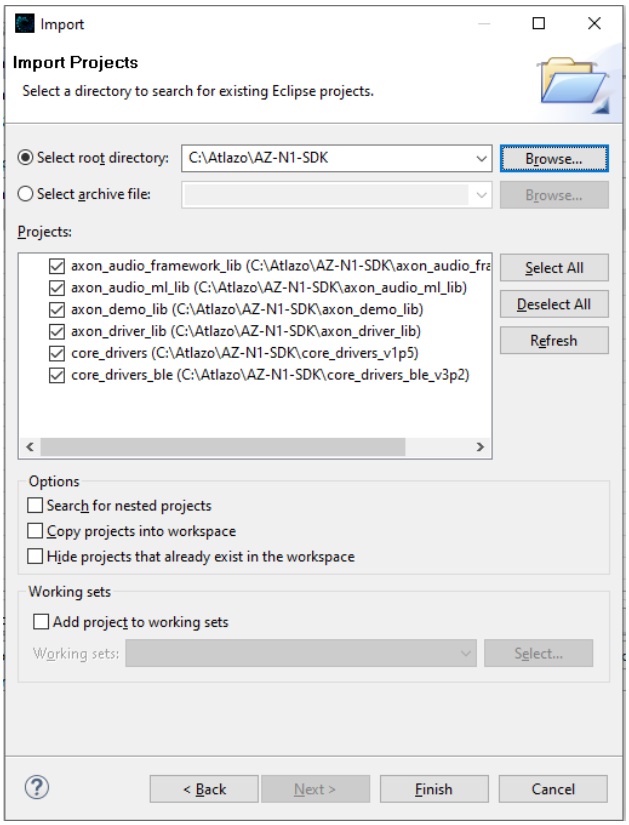
1. Download and unzip the SDK into a working directory. Here, TLSR9X-SDK is the working directory:



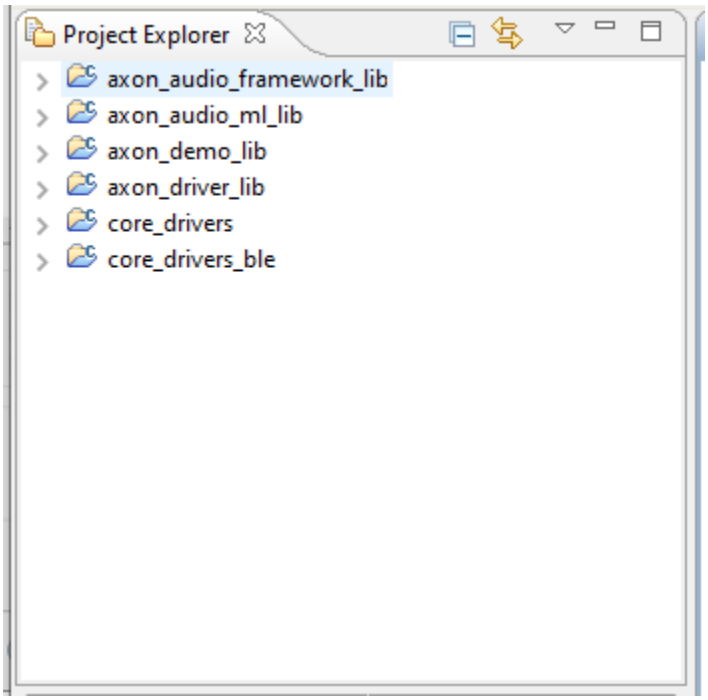
2. Launch AndeSight, select *File|Import*, expand the *General* folder, then select *Existing Projects into Workspace*, then *Next.*



3. *Browse* to the root of the SDK, select all the projects, then select *Finish*.



4. The SDK projects will appear in the project explorer window.



3 USAGE

3.1 EVALUATION BOARD (EVB) CONNECTIONS

The EVB is connected to the debug probe and the USB/Serial Bridge.

- Debug probe cable connects to J56 on the EVB; see picture for orientation.
- USB/Serial bridge RXD (RX into bridge) connects to J34-10, and GND connects to any ground pin (J50 on the back of the board has many).

3.2 BUILDING AXON_APP_BASE_DEMO APPLICATION

Axon_app_base demo is a build configuration in the core_drivers project. It has a dependency on the axon_demo_lib project.

1. Right click on axon_demo_lib in the project explorer and select *Build Project*.
2. Right click on core_drivers in project explorer, select Build Configurations > Set Active, and make sure axon_app_base_demo is checked.
3. Right click on core_drivers in the project explorer and select *Build Project*.

It is assumed that the user has a fundamental understanding of embedded software design and working knowledge of the Eclipse IDE.

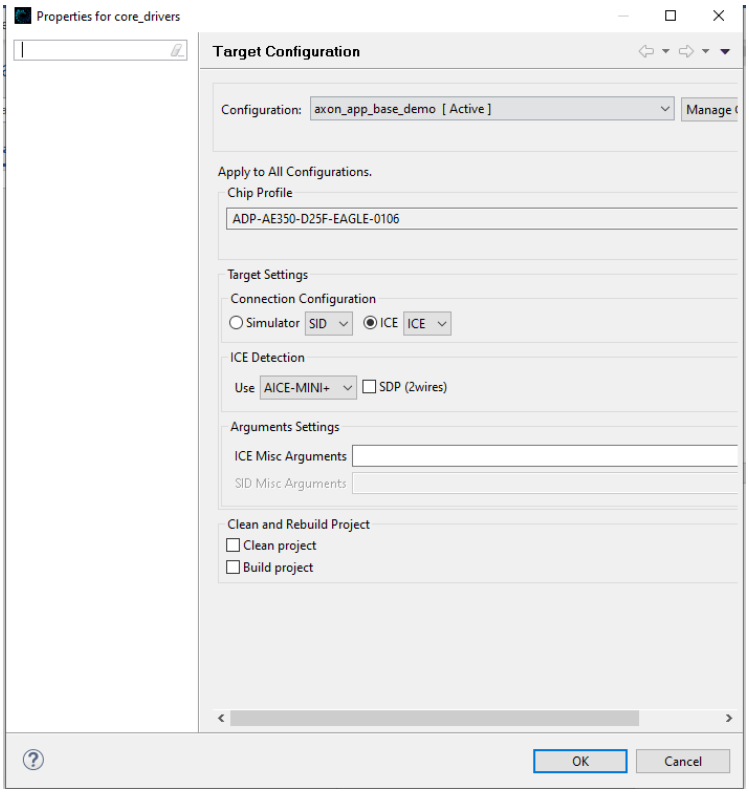
Successfully building an application requires that the desired build configuration has been selected, and all the dependent libraries have been built (dependencies can be seen in the Linker tab of the build settings).

The builds are configured to automatically generate the flash image binary file. This file can be found in the output folder of the build configuration folder, with a .bin extension. For example:

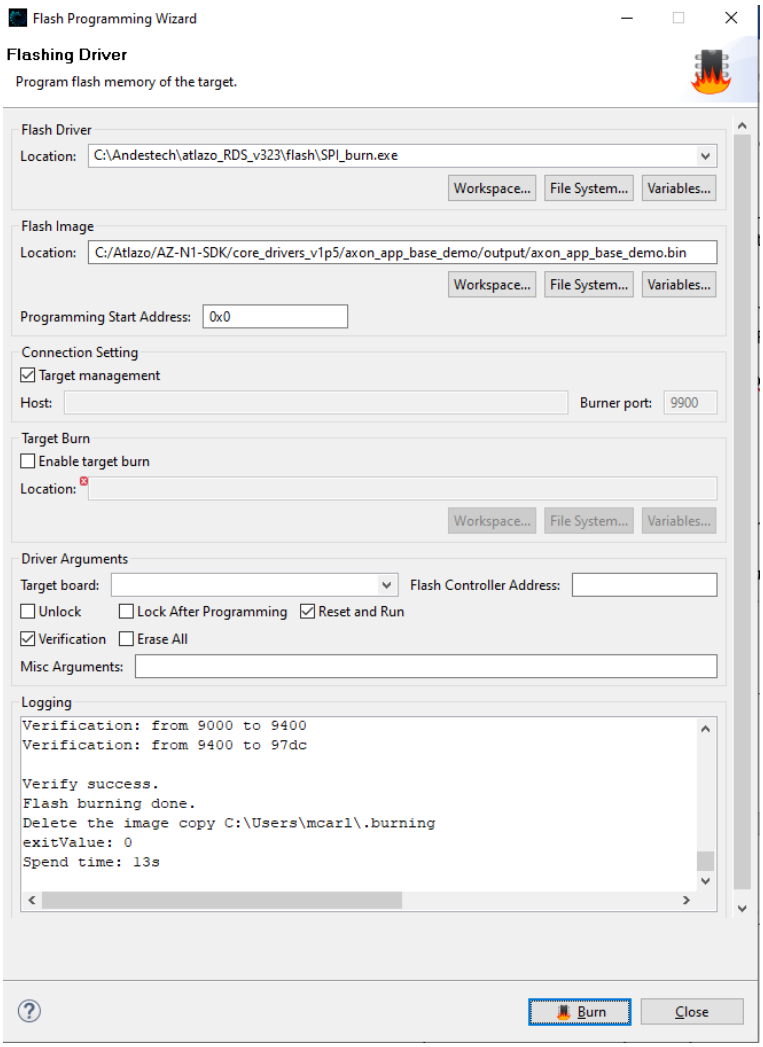
<workspace_root>\core_drivers\axon_app_base_demo\output\ axon_app_base_demo.bin

3.3 FLASHING AXON_APP_BASE_DEMO APPLICATION TO THE EVB

1. Make sure the EVB is connected to the debug probe and the debug probe is connected via USB. The EVB will get power from the debug probe.
2. Right click on core_drivers in the project explorer and select *Target Configuration.* Make sure ICE is the connection configuration, ICE Detection is AICE-MINI+, and SDP is not checked, then click OK.



3. Right click on core_drivers in the project explorer and select *Flash Burner*.
4. Make sure the Flash Driver location is correct (flash\SPI_Burn.exe under the IDE installation folder).
5. Flash image is the binary file to burn. It can be found under <workspace>core_drivers\axon_app_base_demo\output\ axon_app_base_demo.bin
6. *Target Management* should be checked, and *Enable Target Burn* should be unchecked.



7. Click Burn.

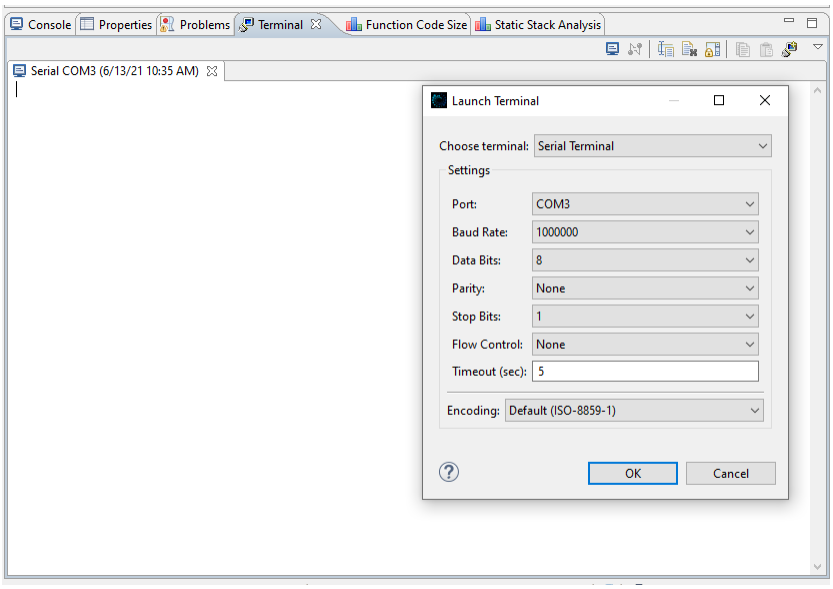
3.4 RUNNING AXON_APP_BASE_DEMO APPLICATION

The application will run immediately after a reset or flash burn.

There is a 10 second delay at start-up to give the user time to make a debug connection before starting the demo.

The application will print result out the serial debug port. To view the results, make sure the serial/USB bridge is connected, and set up a serial terminal connect with properties 1 start bit, 8 data bits, no parity, 1 stop bit, no flow control, 1000000 BAUD.

The IDE has a built-in terminal that can be used.



Sample demo output: