# Integrated Key Word Spotting and Image Classification Demo

Introduction

This is a brief set of instructions to help set up the integrated Key Word Spotting and Image Classification demo on dual Cortex-M55 cores with Ethos-U55 NPU. A more detailed set of app notes is being written but this quick start guide is available for customers in the interim.

The demo consists of two applications: a Key Word Spotting (KWS) application runs on the Cortex-M55 High-Efficiency core (H55-HE / M55_1) and at the same time an Image Classification application run on the Cortex-M55 High-Performance core (M55-HP / M55_0).

When the demo is started, both cores are booted by the Secure Enclave. The KWS application on M55-HE uses continuously listens to the audio input from the built-in microphones on the Alif base board. When the word "Go" is recognized, a command is sent to the M55 HP core to initiate the image classification application. The image classification application uses the camera to detect objects and displays the image output of the camera to a display along with the results of the image classification in real time. The M55 HE continues to listen for keywords and upon detecting the word "Stop", a command is sent to the M55 HP core to stop the image classification.
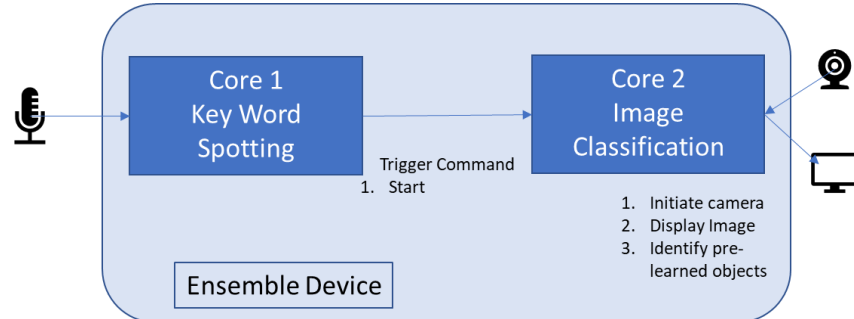
Required Hardware and Setup

- Alif CPU board
- Alif Base board
- Camera module
- Display panel
- USB-to-UART (two)
- Path 1: ULINKpro or ULINKpro D
- Path 2: Segger J-Link

For the Segger J-Link, you will want the non-BASE version for Ozone support, see Software Features link.

Connect Camera and Display

Make sure the development kit is not connected to a power source before proceeding.

Start with the camera module. Find the camera connector on the base board beside the coin cell battery holder and microSD slot. Lightly press the camera module's ribbon cable connector into the base board connector. Not too much pressure is needed before you should feel a snap.

The display panel connects to J415. Start with lifting the black plastic latch to open the connector.
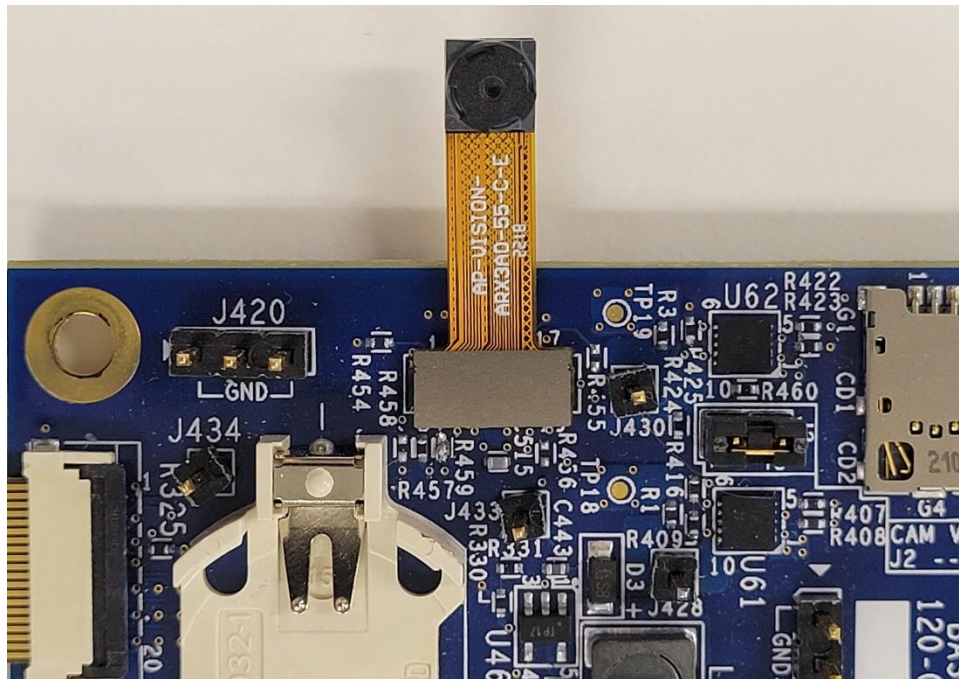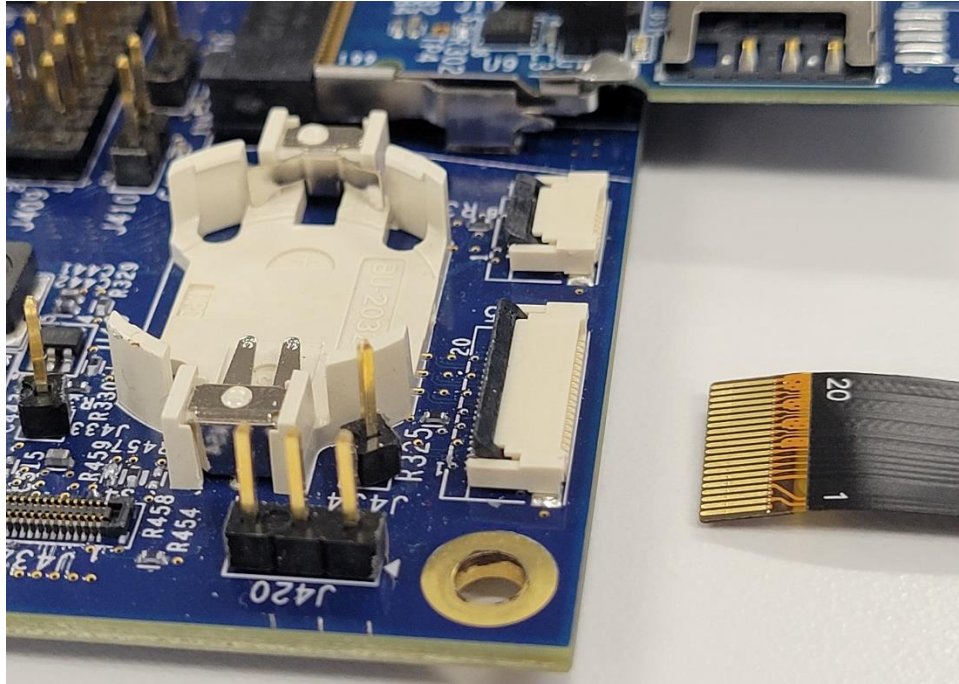
With the latch lifted place the display panel's ribbon cable into the connector. Then press the latch down to lock the ribbon cable in place.
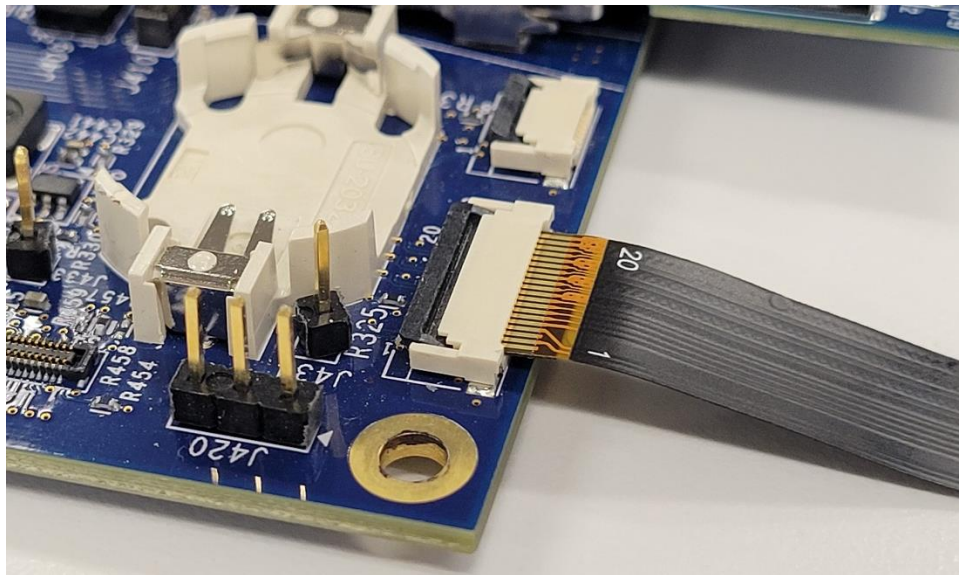

Connect USB-to-UART Adapters

In the Getting Started with Bare Metal User Guide it shows how to connect your USB-to-UART adapter to SEUART as part of the board setup.

For this example, you will need two adapters. The first will be connected to UART2 and the second will be connected to UART4 for console output. Each UART is used by one of the M55 cores. Be sure to follow the precautions in the Getting Started Guide to make sure the 1.8V jumper is selected on the adapters.

Connect the first UART cable between your PC and UART2 on the base board:

- J413 pin 13 (UART2_RX): Connects to TXDATA (blue wire or as noted above) on the cable
- J413 pin 14 (UART2_TX): Connects to RXDATA (black wire or as noted above) on the cable
- J418 (any pin) (GND): Connects to GND (green wire or as noted above) on the cable

Connect the second UART cable between your host PC and UART4 on the base board:

- J412 pin 11 (UART4_RX): Connects to TXDATA (blue wire or as noted above) on the cable
- J412 pin 12 (UART4_TX): Connects to RXDATA (black wire or as noted above) on the cable
- J417 (any pin) (GND): Connects to GND (green wire or as noted above) on the cable
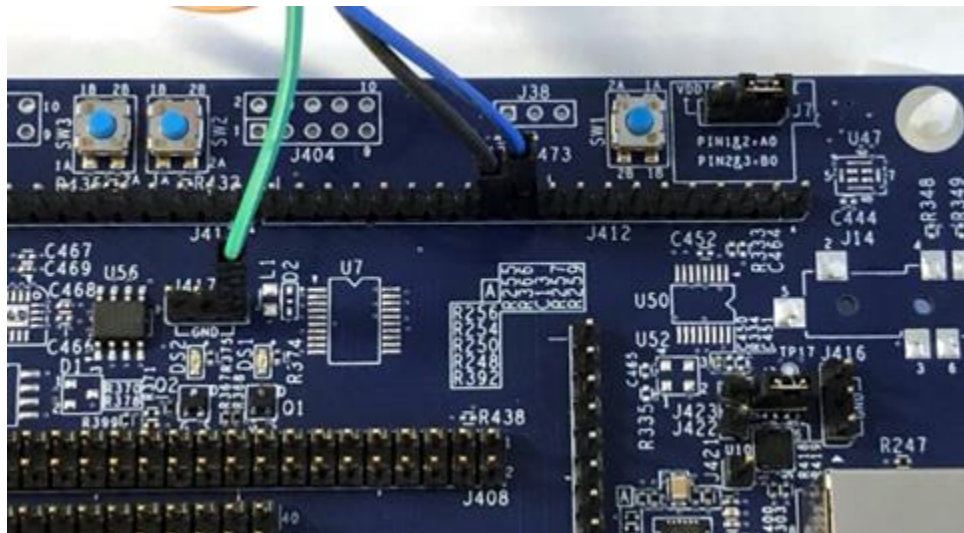
Debug and Power

At this point you may make the final connections for debug and power.

- Connect the ULINKpro to the host PC and then the CPU board
- Connect the DevKit to a power source via microUSB cable

Required Software and Setup

- Ubuntu 20.04 LTS or Ubuntu MATE 20.04 LTS (link)
- Oracle VirtualBox (link)

Path 1

- Arm Development Studio (link – choose Linux 64-bit version)
- Arm Compiler for Embedded (link – choose x86_64 Linux version, not the AArch64)

Path 2

- Segger Ozone (3.26f or later, link)
- Segger J-Link (7.66a or later, link)
- Arm GNU Compiler for Embedded (link – choose x86_64 Linux version)

Virtual Machine Setup

Follow the guide at this page to create an Ubuntu virtual machine using Virtual Box (link). You should proceed far enough to install the VirtualBox Guide Additions. This enables USB support and allows for using Arm DS to debug from within Ubuntu later in the guide. In the machine settings, under the USB category, add the ULINKpro or J-Link debugger to your USB Device filters. After adding the USB device to the filter, unplug the device and plug it back in again. The USB devices that are pulled into the Virtual Machine are not available in Windows while the box is checked. To restore the functionality to Windows, uncheck the box, unplug the device, and plug it back in.
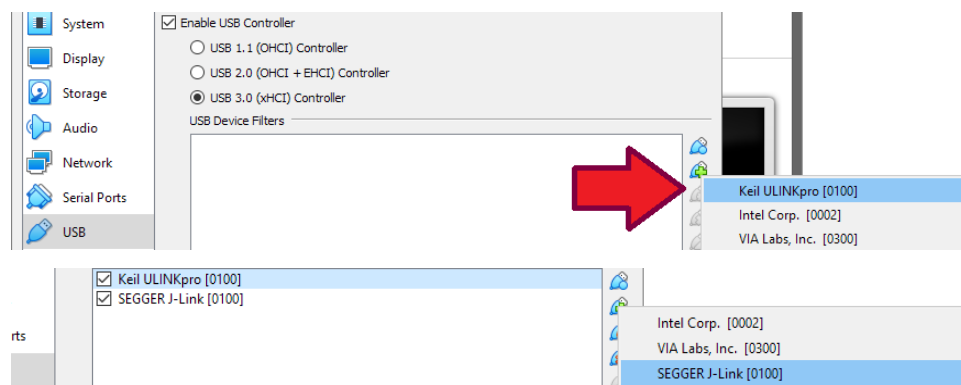
Ubuntu Setup

With Ubuntu running, run the below commands to get the environment ready for development.
sudo apt update
sudo apt upgrade -y

# required dependencies for Arm Development Studio and Arm compiler
sudo apt install libncurses5 libc6-i386 lib32gcc1 lib32stdc++6 lib32z1

# required dependencies for ML Embedded Toolkit
sudo apt install curl dos2unix python-is-python3
snap install cmake –classic

At the time of writing, the following software versions were used:
- Python 3.8.10

- Pip 22.1.2
- Arm Compiler 6.18
- cmake 3.23.2

Arm Software Setup – Path 1

Start with downloading the tgz file for Development Studio and then extract it.

Within the extracted folder is a shell script. Open a terminal window, navigate to the extracted folder, and run the shell script.

Next, download the tar.gz file for Arm Clang Compiler and use sudo to extract it to /usr/local/bin/

sudo tar xf ARMCompiler6.18_standalone_linux-x86_64.tar.gz -C /usr/local/bin

Add the ARM license server to your environment (if applicable), example shown:
sudo sh -c "echo export ARMLMD_LICENSE_FILE=PORT@IPADDRESS > /etc/profile.d/arm-license.sh"

After extracting Arm Clang compiler we will need to add it to the path, example shown:
sudo sh -c "echo export PATH=/usr/local/bin/ArmCompiler6.18/bin:$PATH > /etc/profile.d/arm-compiler.sh"

It is needed to log out and then log in for the above environment changes to take effect.

Arm Software Setup – Path 2

You will only need to download the GNU Arm Embedded Toolchain from the provided link. You may ignore the note on Arm's webpage that the compiler is deprecated. Using the latest release may work but has not been tested.

Next, extract the downloaded tar.bz2 file and use sudo to extract it to /usr/local/bin

sudo tar xf gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2 -C /usr/local/bin

After extracting Arm Clang compiler we will need to add it to the path, example shown:
sudo sh -c "echo export PATH=/usr/local/bin/gcc-arm-none-eabi-10.3-2021.10/bin:$PATH > /etc/profile.d/arm-compiler.sh"

It is needed to log out and then log in for the above environment changes to take effect.

Following Path 2, the GNU Arm Embedded Toolchain is used to compile our applications and Ozone is used to debug. Download the two .deb files from the links provided above. Then install both using dpkg.
sudo dpkg -i JLink_Linux_V768b_x86_64.deb
sudo dpkg -i Ozone_Linux_V326h_x86_64.deb

Ozone should now appear in your list of installed applications.

Building the Applications

The two applications are built separately. They can also be executed one at a time to demonstrate the individual application's functionality. Later they are combined in a flash download for the Alif device so that it can be run on the device standalone, without requiring the debug environment.

Building the Key Word Spotting for M55-HE
Private repository on Alif Semi GitHub account:

https://github.com/alifsemi/ml-kws-ic-example-ensemble-share

git clone https://github.com/alifsemi/ml-kws-ic-example-ensemble-share.git
cd ml-kws-ic-example-ensemble-share

git submodule init
git submodule update
python set_up_default_resources.py --additional-ethos-u-config-name ethos-u55-128

The above python command will take some time, around a few minutes. If you get errors right at the beginning, try doing the following and run the command again:

sudo apt-get install python3-pip
sudo apt install python3-venv
Delete the folder resource-downloaded
Rerun the commands

After the python command completes, let's continue the build process.

mkdir build
cd build

```
cmake -DTARGET_PLATFORM=ensemble \
    -DTARGET_SUBSYSTEM=RTSS-HE \
    -DCMAKE_TOOLCHAIN_FILE=scripts/cmake/toolchains/bare-metal-armclang.cmake \
    -DUSE_CASE_BUILD=kws \
    -Dkws_MODEL_TFLITE_PATH=resources_downloaded/kws/kws_micronet_m_vela_H128.tflite \
    -Dkws_FILE_PATH=resources/kws/samples/down.wav \
    -DCMAKE_BUILD_TYPE=Debug -DLOG_LEVEL=LOG_LEVEL_DEBUG ..
make ethos-u-kws -j
```

The output should be in ml-kws-ic-example-ensemble-share/build/bin/ethos-u-kws.axf

Change **bare-metal-armclang** to **bare-metal-gcc** for the GNU Toolchain.

Building the Image Classification for M55-HP

Public repository https://git.qhub.com/alifsemi/ensembleML/tree/image_demo

```
git clone https://github.com/alifsemi/ensembleML.git
cd ensembleML
```

```
git checkout image_demo
python download_dependencies.py
```

If you get errors during download_dependencies.py, delete the dependencies directory and try again. , try doing the following and run the command again:

```
mkdir build
cd build
```

```
cmake -DTARGET_PLATFORM=ensemble \
    -DTARGET_SUBSYSTEM=RTSS-HP \
    -DCMAKE_TOOLCHAIN_FILE=scripts/cmake/toolchains/bare-metal-armclang.cmake \
    -DUSE_CASE_BUILD=img_class \
    -
Dimg_class_MODEL_TFLITE_PATH=resources/img_class/model/mobilenet_softmax_v2_1.0_224_uint8_vela_H256.tflite \
    -Dimg_class_FILE_PATH=resources/img_class/samples/cat.bmp \
    -DCMAKE_BUILD_TYPE=Debug -DLOG_LEVEL=LOG_LEVEL_DEBUG ..
make -j
```

Change **bare-metal-armclang** to **bare-metal-gcc** for the GNU Toolchain.

The output should be in ensembleML/build/bin/ethos-u-img_class.axf

Running the Applications using Arm DS

How to run the applications using Arm Development Studio and the ULINKpro debug tool.

Open Arm Development Studio. Follow the steps from the Getting Started with Bare Metal Guide to "create a debug connection…" to the Alif Development Kit.
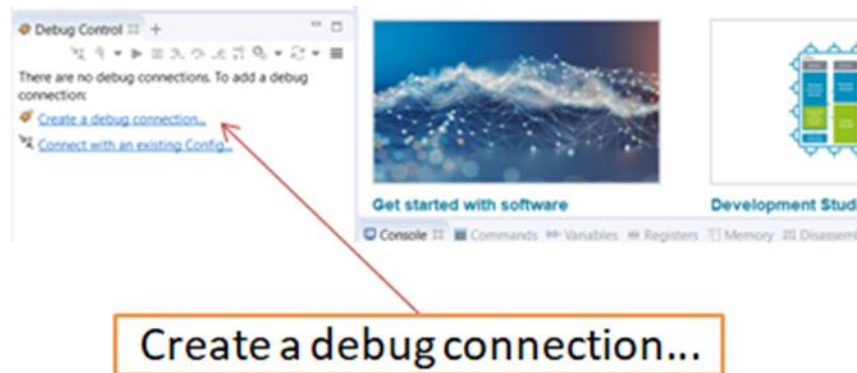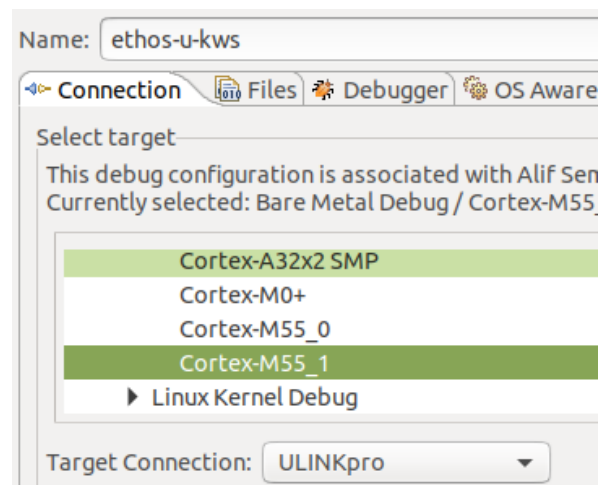

Follow the Getting Started Guide until you have a ".launch" file capable of connecting any core on the development kit.
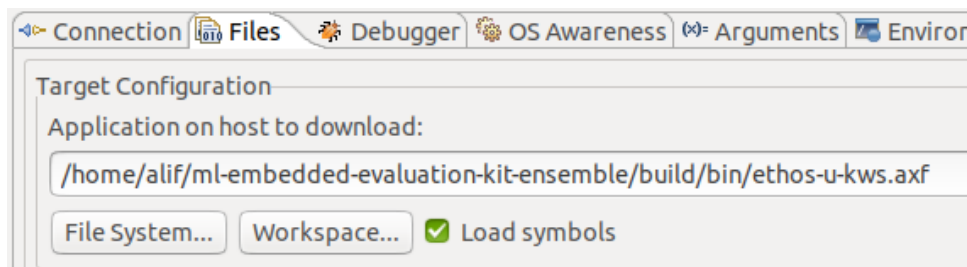
Running the M55-HE KWS Application

Create a copy of the ".launch" config called "ethos-u-kws.launch".

Open the launch config and choose the Cortex-M55_1 in the "Connection" tab.

Then use the ethos-u-kws.axf file built from the steps before in the "Files" tab.

Finally, select "Debug from symbol: main" and close.

You should be able to run the application and see the below in the very first lines of output from UART2

```
INFO - Processor internal clock: 160000000Hz
INFO - platform_init: complete
DEBUG - EthosU IRQ#: 55, Handler: 0x6200890d
INFO - Ethos-U device initialised
INFO - Ethos-U version info:
INFO -    Arch:        v1.0.0
INFO -    Driver:      v0.16.0
INFO -    MACs/cc:     128
INFO -    Cmd stream: v0
INFO - Target system design: Ensemble
i2s_pinmux_config(1) = 0
i2s_pinmux_config(2) = 0
i2s_pinmux_config(3) = 0
I2S API version = 258
INFO - ARM ML Embedded Evaluation Kit
INFO - Version 22.05.0 Build date: Jun 16 2022 @ 09:48:04
INFO - Copyright (C) ARM Ltd 2021-2022. All rights reserved.
```

The application will run in a loop listening to the microphones. It will report if the keywords from the following list are spotted: up, down, left, right, yes, no, go, stop, on, off

Correct behavior:

```
INFO - NPU TOTAL: 412213 cycles
Original sample stats: min = -1816, max = 0, average = -1758
Normalized sample stats: min = -232, max = 212, average = -2
DEBUG - Input tensor populated
```

Detection of left and right key words

```
l sample stats: min = -2257, max = 0, average = -1758
zed sample stats: min = -1996, max = 2392, average = 0
 Input tensor populated
 Input tensor populated
Final results:
Total number of inferences: 2
For timestamp: 0.000000 (inference #: 0); label: left, score: 0.789587; thr
For timestamp: 0.000000 (inference #: 0); label: (none), threshold: 0.70000
Profile for Inference:
NPU IDLE: 6007 cycles
NPU AXI0_RD_DATA_BEAT_RECEIVED: 132130 beats
NPU AXI0_WR_DATA_BEAT_WRITTEN: 48252 beats
NPU AXI1_RD_DATA_BEAT_RECEIVED: 17545 beats
NPU ACTIVE: 406142 cycles
NPU TOTAL: 412148 cycles
l sample stats: min = -2121, max = 0, average = -1757
zed sample stats: min = -1456, max = 2072, average = 0
 Input tensor populated
 Input tensor populated
Final results:
Total number of inferences: 2
For timestamp: 0.000000 (inference #: 0); label: (none), threshold: 0.70000
For timestamp: 0.000000 (inference #: 0); label: right, score: 0.994280; th
Profile for Inference:
NPU IDLE: 6139 cycles
NPU AXI0_RD_DATA_BEAT_RECEIVED: 132130 beats
NPU AXI0_WR_DATA_BEAT_WRITTEN: 48252 beats
NPU AXI1_RD_DATA_BEAT_RECEIVED: 17545 beats
NPU ACTIVE: 406189 cycles
NPU TOTAL: 412328 cycles
```

Incorrect behavior (mics not working):

```
INFO - NPU TOTAL: 412202 cycles
Original sample stats: min = 0, max = 0, average = 0
Normalized sample stats: min = 0, max = 0, average = 0
DEBUG - Input tensor populated
```
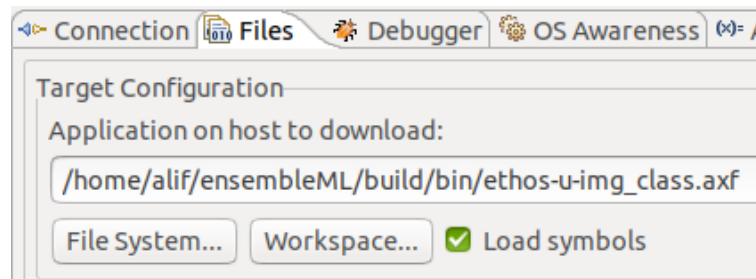
Running the M55-HP Image Classification Application

Create a copy of the ".launch" config called "ethos-u-img-class.launch".

Open the launch config and choose the Cortex-M55_0 in the "Connection" tab.
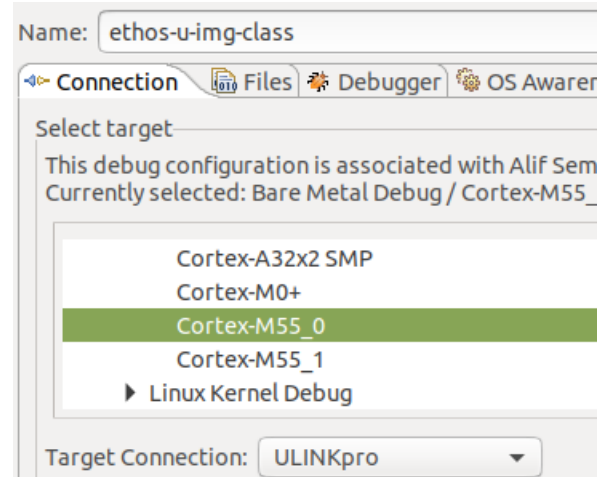
Then use the ethos-u-img-class.axf file built from the steps before in the "Files" tab.
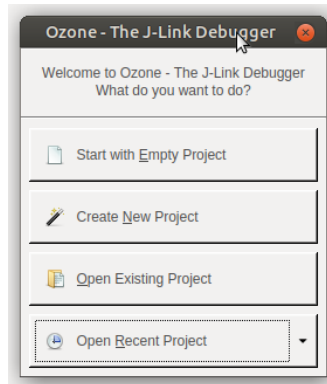

Finally, select "Debug from symbol: main" and close.

You should be able to run the application and see the below in the very first lines of output from UART2

Running the Applications using Ozone

Launch Ozone from the applications menu. It should either start with the New Project Wizard or launch a menu with the option to Create New Project.

In the New Project Wizard press the "…" button to bring up device selection. Type "AE7" to help filter the options. To debug the Key Word Spotting for M55-HE, select the **_HE** core. And to debug the Image Classification for M55-HP, select the **_HP** core.

In the next window, you will set up the J-Link settings. Keep the target interface speed at default or at least no higher than 8MHz. If your debugger does not appear here, then re-check the USB settings for your virtual machine.

In the next window, simply browse to the path of the axf file. Then in the next window click Finish.

Running the Applications Standalone without Debuggers

These steps require that you have the Arm Clang Toolchain.

After building the applications above, these steps will prepare them to be stored in the Alif device and booted by the Secure Enclave automatically.

cd ensembleML/build/bin
fromelf --bin --output ethos-u-img_class.bin ethos-u-img_class.axf

cd ml-kws-ic-example-ensemble-share/build/bin
fromelf --bin --output ethos-u-kws.bin ethos-u-kws.axf

Copy the binaries to the following directory: app-release/build/images

Note, if you see a warning, when writing the binaries to MRAM, about the binaries not being 16-byte aligned then use **one** of the following commands to pad out the binaries.

truncate --size /16 build/images/image_name.bin

python3 pad_16bytes.py -b build/images/image_name.bin

The next step is to generate the binary ATOC image for the two programs. Create a new JSON file called <app-release>/build/config/kws_img_class_demo.json with the following content.

```
{
  "HP_Image": {
     "binary": "ethos-u-img_class.bin",
     "version" : "1.0.0",
     "mramAddress": "0x80001000",
     "cpu_id": "M55_HP",
     "flags": ["boot"],
     "signed": false
  },
  "HE_Voice": {
     "binary": "ethos-u-kws.bin",
     "version" : "1.0.0",
     "loadAddress": "0x62000000",
     "cpu_id": "M55_HE",
     "flags": ["boot", "load"],
     "signed": false
  }
}
```

Next, run app-gen-toc.py to generate the package image, which will be written to the file AppTocPackage.bin in the build directory. We will use the "-f" option to specify the input filename (kws_img_class_demo.json) for the configuration file we just created. Execute this command:
python app-gen-toc.py -f build/config/kws_img_class_demo.json

Finally, write the applications using the SETOOLS command
python app-write-mram.py

Applications output to UART2 (kws) and UART4 (img_class).