

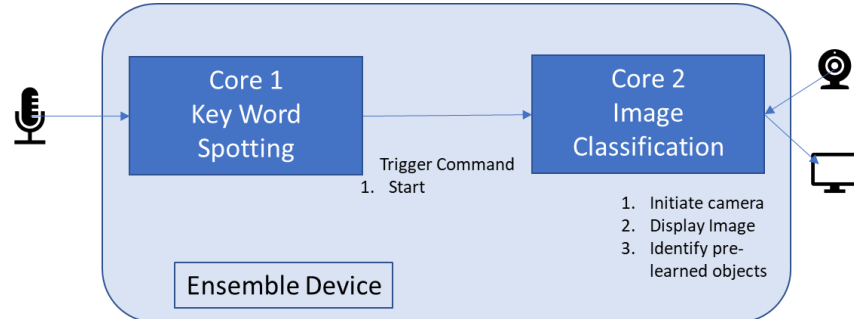
# Integrated Key Word Spotting and Image Classification Demo

## 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 NPUs. 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

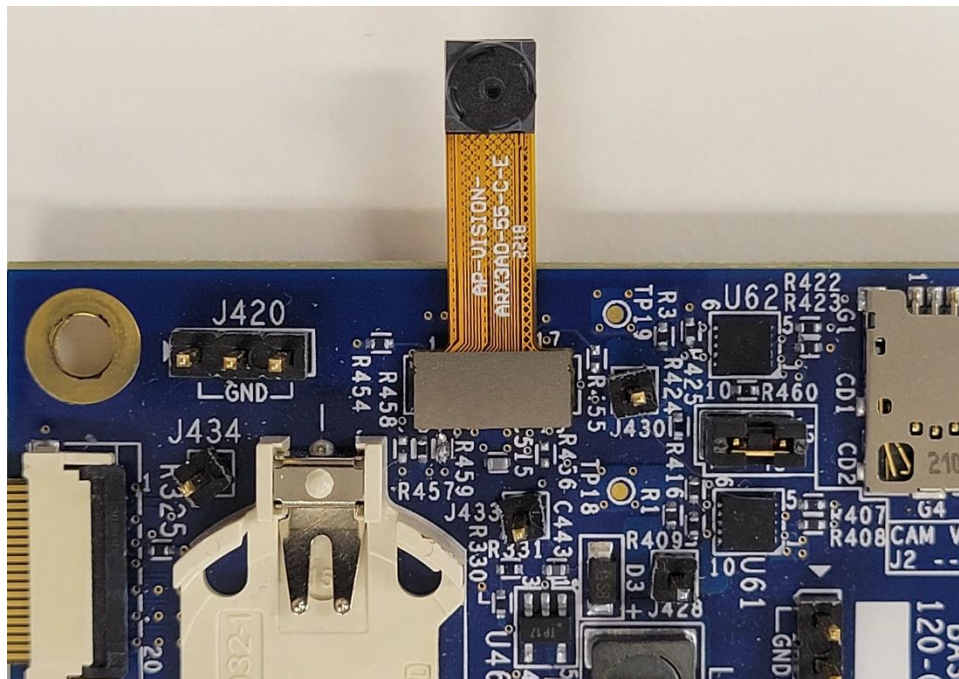
---

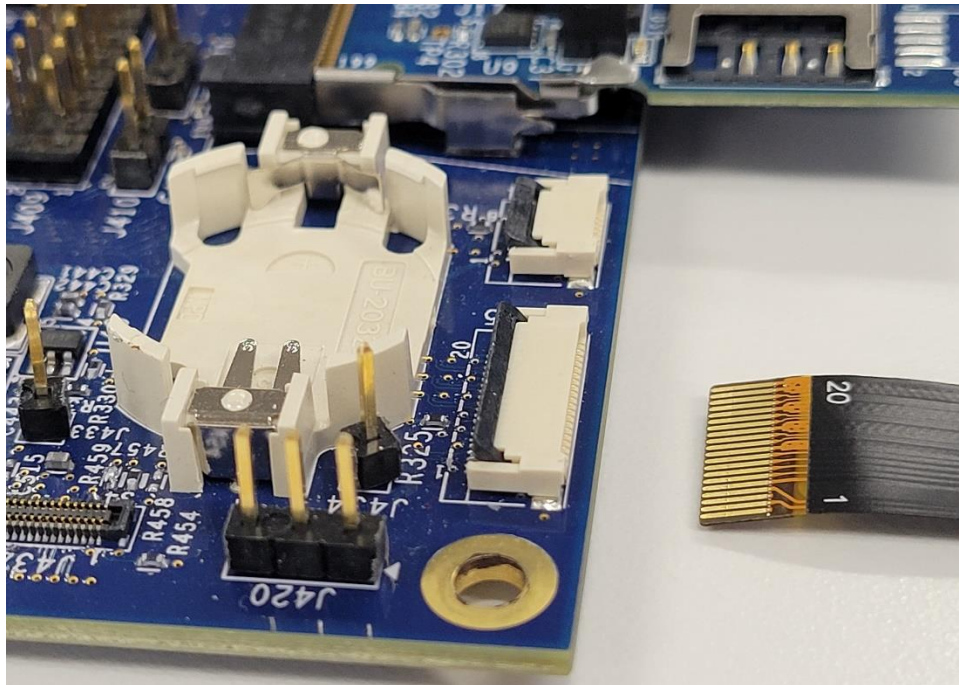
- Alif CPU board
- Alif Base board
- Camera module
- Display panel
- USB-to-UART (two)
- Path 1: ULINKpro or ULINKpro D
- Path 2: Segger J-Link

### Connect Camera and Display

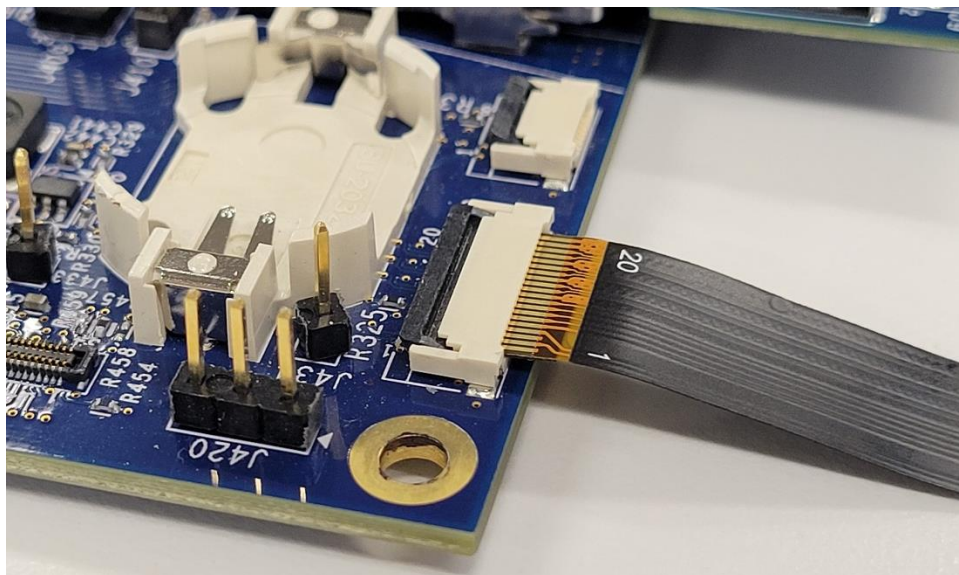
Make sure the development kit is not connected to a power source before proceeding.

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.





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

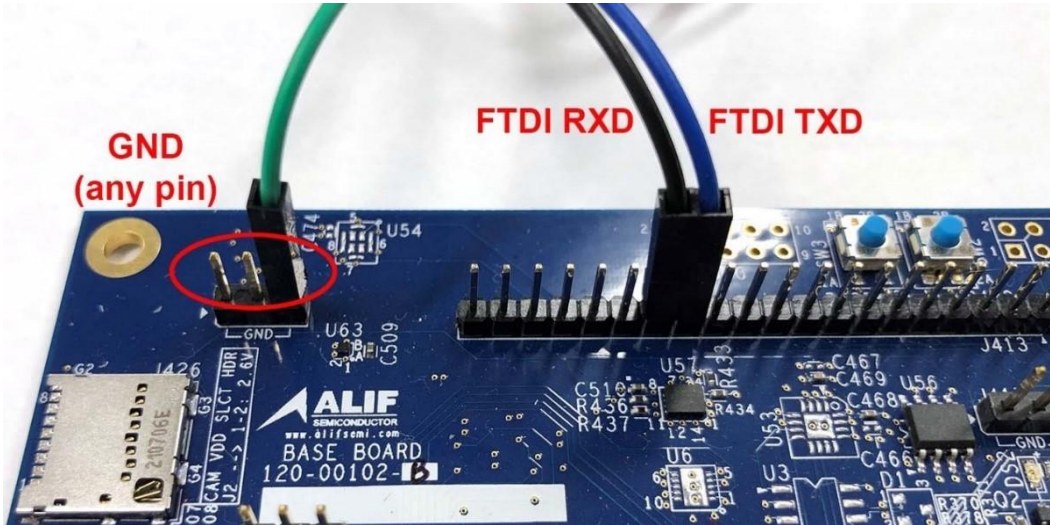
In the Getting Started with Bare Metal User Guide it shows how to connect your USB-to-UART adapter to SEUART as part of the board setup.

For this example, you will need one adapter although two adapters are preferred. 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.



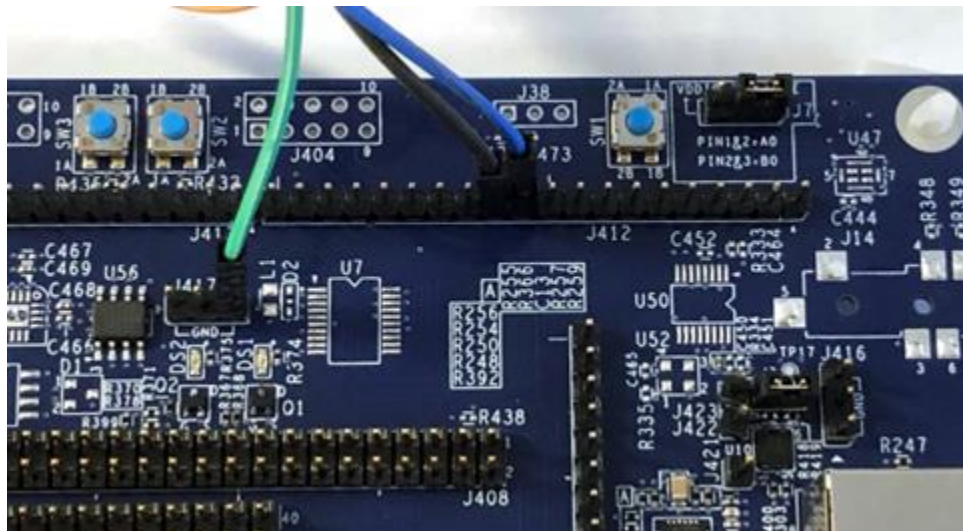
Connect the first UART cable between your PC and UART2 on the base board:

- 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 or J-Link 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

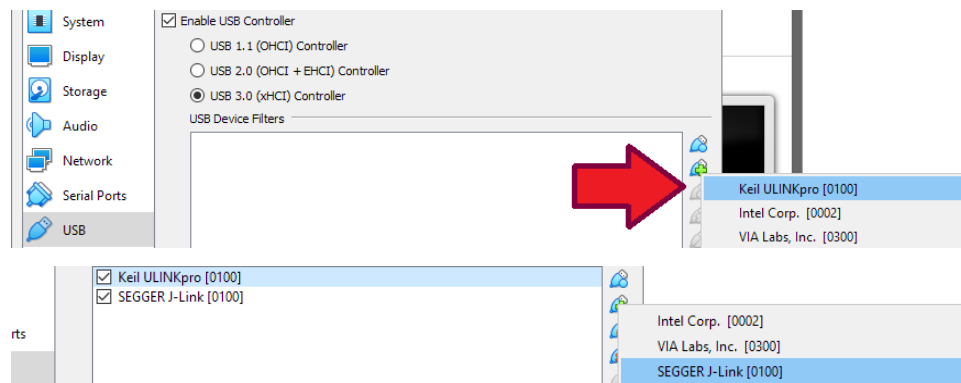
- Arm Development Studio ([link](#) – choose Linux 64-bit version)
- Arm Compiler for Embedded ([link](#) – choose x86\_64 Linux version, not the AArch64)

### 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 Guest 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
```

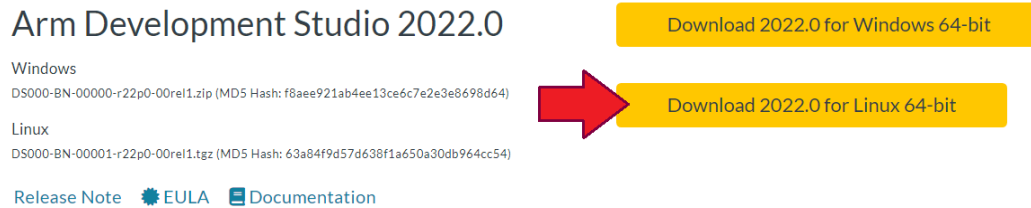
At the time of writing, the following software versions were used:

- Python 3.8.10

- Pip 22.1.2
- cmake 3.23.2

## Arm Software Setup – Path 1

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



**Arm Development Studio 2022.0**

Windows  
DS000-BN-00000-r22p0-00rel1.zip (MD5 Hash: f8aee921ab4ee13ce6c7e2e3e8698d64)

Linux  
DS000-BN-00001-r22p0-00rel1.tgz (MD5 Hash: 63a84f9d57d638f1a650a30db964cc54)

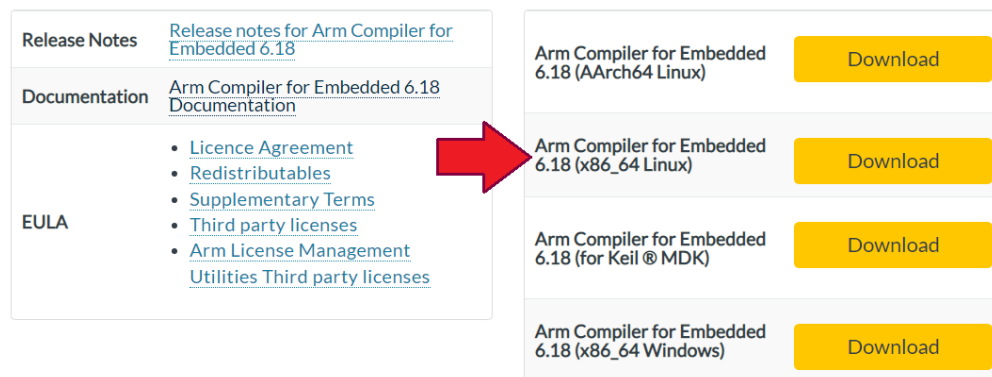
[Release Note](#) [EULA](#) [Documentation](#)

[Download 2022.0 for Windows 64-bit](#)

[Download 2022.0 for Linux 64-bit](#)

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



**Release Notes** [Release notes for Arm Compiler for Embedded 6.18](#)

**Documentation** [Arm Compiler for Embedded 6.18 Documentation](#)

**EULA**

- [Licence Agreement](#)
- [Redistributables](#)
- [Supplementary Terms](#)
- [Third party licenses](#)
- [Arm License Management Utilities Third party licenses](#)

**Arm Compiler for Embedded 6.18 (AArch64 Linux)** [Download](#)

**Arm Compiler for Embedded 6.18 (x86\_64 Linux)** [Download](#)

**Arm Compiler for Embedded 6.18 (for Keil ® MDK)** [Download](#)

**Arm Compiler for Embedded 6.18 (x86\_64 Windows)** [Download](#)

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"
```

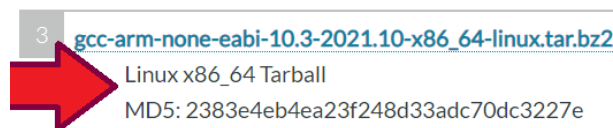
After extracting Arm Clang compiler we will need to add it to the path, example shown:

```
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.



[gcc-arm-none-eabi-10.3-2021.10-x86\\_64-linux.tar.bz2](#)

Linux x86\_64 Tarball

MD5: 2383e4eb4ea23f248d33adc70dc3227e

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 applications' 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

Public repository on Alif Semi GitHub account:

[https://github.com/alifsemi/alif\\_ml-embedded-evaluation-kit](https://github.com/alifsemi/alif_ml-embedded-evaluation-kit)

```
git clone https://github.com/alifsemi/alif_ml-embedded-evaluation-kit.git
cd alif_ml-embedded-evaluation-kit
```

```
git submodule init
git submodule update
python set_up_default_resources.py --additional-ethos-u-config-name ethos-u55-256
```

The above python command will take some time, around a few minutes. Python command fetches and optimizes the needed Ethos models for all the use cases in the kit. 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_he
cd build_he
```

```
cmake -DTARGET_PLATFORM=ensemble \  
-DTARGET_SUBSYSTEM=RTSS-HE \  
-DCMAKE_TOOLCHAIN_FILE=scripts/cmake/toolchains/bare-metal-armclang.cmake \  
-DGLCD_UI=NO -DLINKER_SCRIPT_NAME=ensemble-RTSS-HE-TCM \  
-DCMAKE_BUILD_TYPE=Release -DLOG_LEVEL=LOG_LEVEL_DEBUG ..  
make ethos-u-alif_kws -j
```

The output should be in alif\_ml-embedded-evaluation-kit/build\_he/bin/ethos-u-alif\_kws.axf

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

The options for LCD and linker ensure that the image does not clash with the image classification using the LCD and the main SRAM. If intending to build as a stand-alone application, those options could be omitted.

## Building the Image Classification for M55-HP

Public repository on Alif Semi GitHub account:

[https://github.com/alifsemi/alif\\_ml-embedded-evaluation-kit](https://github.com/alifsemi/alif_ml-embedded-evaluation-kit)

NOTE! This is the same repository as in Key word spotting. Skip the cloning and downloading of resources if already done with Key word spotting and go to creating build directory.

```
git clone https://github.com/alifsemi/alif_ml-embedded-evaluation-kit.git  
cd alif_ml-embedded-evaluation-kit
```

```
git submodule init  
git submodule update  
python set_up_default_resources.py --additional-ethos-u-config-name ethos-u55-256
```

The above python command will take some time, around a few minutes. Python command fetches and optimizes the needed Ethos models for all the use cases in the kit. 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_hp  
cd build_hp
```

```
cmake -DTARGET_PLATFORM=ensemble \  
-DTARGET_SUBSYSTEM=RTSS-HP \  
-DCMAKE_TOOLCHAIN_FILE=scripts/cmake/toolchains/bare-metal-armclang.cmake \  
-DCONSOLE_UART=4 \  
-DCMAKE_BUILD_TYPE=Release -DLOG_LEVEL=LOG_LEVEL_DEBUG ..  
make ethos-u-alif_img_class -j
```

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

These options permit the default use of LCD and SRAM, which is okay because the HE image has them disabled. The CONSOLE\_UART=4 option avoids the HE image's use of UART2, and could be omitted to run standalone HP applications.



The output should be in `alif_ml-embedded-evaluation-kit/build_hp/bin/ethos-u-alif_img_class.axf`

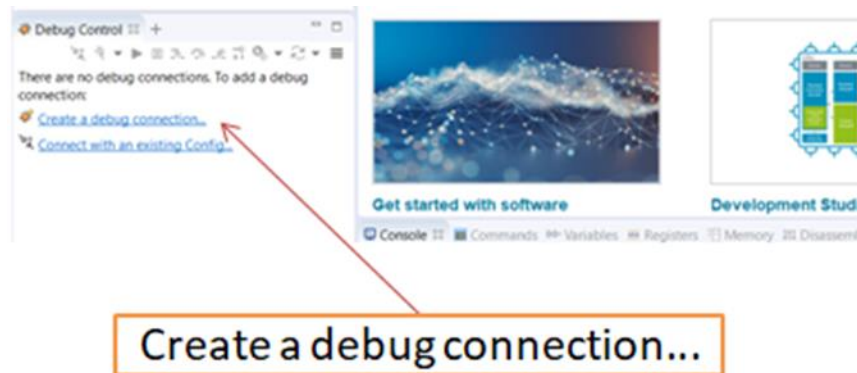
## Display Panels

Please check the model number of the display panel that is attached to the Ensemble DevKit and edit the file at `source/hal/source/platform/ensemble/cmsis-pack/config/M55_HP/RTE_Device.h` file to enable the correct model number, by selecting the applicable `RTE_ILI9806E_PANEL_<model>_EN` option. (If intending to use the display from the M55-HE, also edit the M55\_HE version of the file).

## 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.

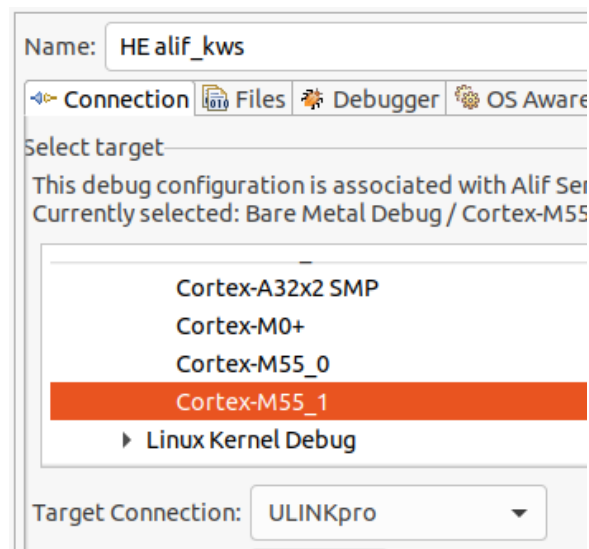


Follow the Getting Started Guide until you have a “.launch” file capable of connecting any core on the development kit.

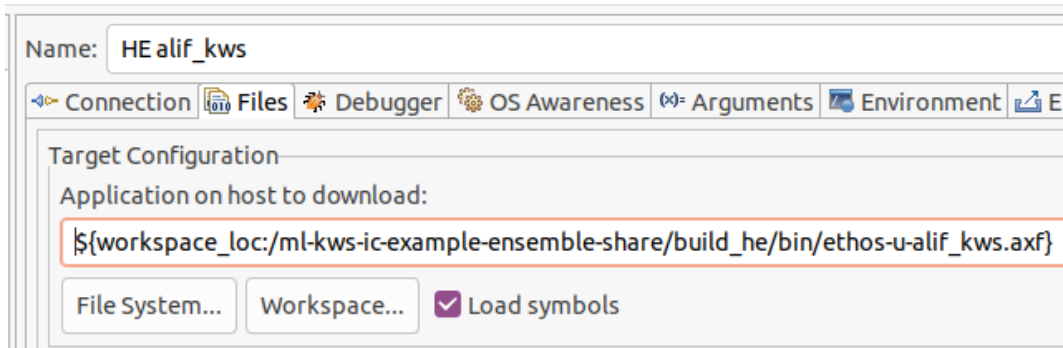
### Running the M55-HE KWS Application

Create a copy of the “.launch” config called “HE alif-kws.launch”.

Open the launch config and choose the Cortex-M55\_1 in the “Connection” tab.



Then use the ethos-u-alif\_kws.axf file built from the steps before in the “Files” tab.



Finally, select “Debug from symbol: main” and close.

## KWS Usage Notes

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: 0x0000a299
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
INFO - ARM ML Embedded Evaluation Kit
INFO - Version 22.11.0 Build date: Jan 24 2023 @ 13:25:49
INFO - Copyright 2021-2022 Arm Limited and/or its affiliates
```

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

It may be necessary to disconnect the JTAG connector for the microphones to work, as there is a pin conflict with the JTAG trace pins. This is a known issue when using ULINKpro, but ULINKpro D appears to avoid it.

Correct behavior:

```
Original sample stats: absmax = 132, mean = -9
Normalized sample stats: absmax = 28912, mean = -1973 (gain = 47 dB)
```

Detection of left and right key words

```
Original sample stats: absmax = 13, mean = -5
Normalized sample stats: absmax = 4872, mean = -2079 (gain = 52 dB)
DEBUG - Input tensor populated
Preprocessing time = 6.961 ms
Inference time = 2.707 ms
Postprocessing time = 0.024 ms
INFO - Final results:
INFO - Total number of inferences: 8
INFO - For timestamp: 23.000002 (inference #: 46); label: _unknown_, score: 0.71061
INFO - For timestamp: 23.500002 (inference #: 47); label: up, score: 0.868759; thr
INFO - For timestamp: 24.000000 (inference #: 48); label: up, score: 0.925107; thr
INFO - For timestamp: 24.500002 (inference #: 49); label: left, score: 0.994687; t
INFO - For timestamp: 25.000000 (inference #: 50); label: left, score: 0.983821; t
INFO - For timestamp: 25.500002 (inference #: 51); label: _unknown_, score: 0.70111
INFO - For timestamp: 26.000000 (inference #: 52); label: _unknown_, score: 0.68831
INFO - For timestamp: 26.500002 (inference #: 53); label: _unknown_, score: 0.68301
INFO - Profile for Inference:
```

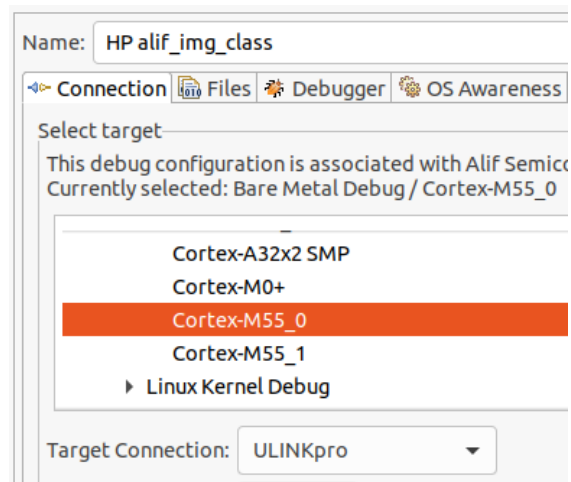
Incorrect behavior (mics not working):

```
Original sample stats: absmax = 0, mean = 0  
Normalized sample stats: absmax = 0, mean = 0 (gain = 80 dB)
```

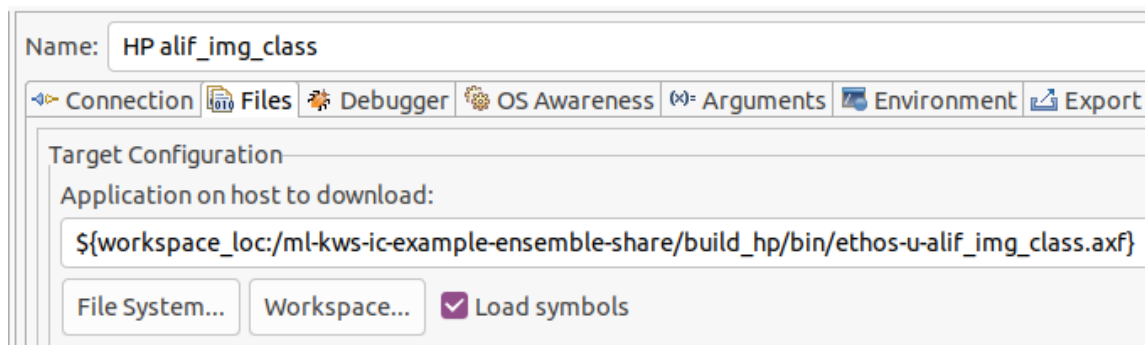
## Running the M55-HP Image Classification Application

Create a copy of the “.launch” config called “HP alif\_img\_class.launch”.

Open the launch config and choose the Cortex-M55\_0 in the “Connection” tab.



Then use the ethos-u-alif\_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 now.

### Image Classification Usage Notes

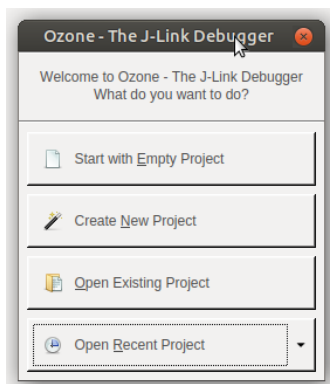
The display will turn on with a white background and at the top will be the camera viewfinder, a live feed of the image captured through the camera module, at the 224x224 resolution as it will be presented to the ML inference. Below the viewfinder are three text boxes showing the top 3 inferencing results.

To start inferencing, press SW2 for a single frame inference or press SW3 to enable continuous inferencing. To disable continuous inferencing press SW3 again.



## 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.

Target Device Settings			
Selected Device: AE722F80F55D5_HP			
Manufacturer /	Device	Core	NumCores
	ae7		Filter
AlifSemiconductor	AE722F80F55D5_HE	Cortex-M55	1
AlifSemiconductor	AE722F80F55D5_HP	Cortex-M55	1
AlifSemiconductor	AE722F80F55D5_A32_0	Cortex-A32	1
AlifSemiconductor	AE722F80F55D5_A32_1	Cortex-A32	1

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.

Target Interface	Target Interface Speed	
JTAG	4 MHz	
Host Interface	Serial No (optional)	
USB	51009989	
Emulators connected via USB		
Product	Nickname	Serial No
SEGGER J-Link ARM		51009989

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

Please refer to the above two Usage Notes sections on the previous pages.

## Running the Applications Standalone without Debuggers

---

After building the applications above, these steps will prepare them to be stored in the Alif device and booted by the Secure Enclave automatically. If you followed Path 1 to install Arm DS and Arm Compiler for Embedded, then use the “fromelf” command. Or if you followed Path 2 to install the Arm GNU Compiler for Embedded, then use the “arm-none-eabi-objcopy” command.

```
cd alif_ml-embedded-evaluation-kit/build_hp/bin
fromelf --bin --output ethos-u-alif_img_class.bin ethos-u-alif_img_class.axf
or
arm-none-eabi-objcopy -O binary ethos-u-alif_img_class.axf ethos-u-alif_img_class.bin
```

```
cd alif_ml-embedded-evaluation-kit/build_he/bin
fromelf --bin --output ethos-u-alif_kws.bin ethos-u-alif_kws.axf
or
arm-none-eabi-objcopy -O binary ethos-u-alif_kws.axf ethos-u-alif_kws.bin
```

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

Note, binaries written to MRAM must be 16-byte aligned or the write operation will fail. If you see a warning in the SE Tools about the binaries not being 16-byte aligned, then use the following command to pad out the binaries.

```
truncate --size %16 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-alif_img_class.bin",
    "version": "1.0.0",
    "mramAddress": "0x80001000",
    "cpu_id": "M55_HP",
    "flags": ["boot"],
    "signed": false
  },
  "HE_Voice": {
    "binary": "ethos-u-alif_kws.bin",
    "version": "1.0.0",
    "mramAddress": "0x80480000",
    "cpu_id": "M55_HE",
    "flags": ["boot"],
    "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 (RTSS-HE) and UART4 (RTSS-HP).

## Further information

---

Beyond the demo build described above, many other use cases and options of the upstream Arm ML Embedded Evaluation kit should work on Alif hardware. See the original documentation for full details.

One easy alternative build is to replace the image classification use case on the HP core with object (face) detection.

### Tested use cases

Of the original ARM use cases the following are known to work:

- kws
- img\_class
- object\_detection
- vww

The following are known to not work:

- asr (model too large)

New use cases have been added:

- alif\_kws (equivalent of kws with live microphone input)
- alif\_img\_class (equivalent of img\_class with live camera input)
- alif\_object\_detection (equivalent of object\_detection with live camera input)

### Alif-specific build options

There are several build options – these determine behaviour of the porting layer. Once these are set, you can build multiple use cases in one build directory using these options. See original ARM documentation for details of the upstream options. Alif has added extra options:

`-DROTATE_DISPLAY=<0|90|180|270>`

Rotates the display by the specified amount, and reorganizes the UI if necessary. 90 and 270 will be appreciably slower. (Default is 0)

`-DGLCD_UI=<ON|OFF>`

Enables or disables the basic “GLCD” UI emulating ARM’s MPS3 display. This is the display method of all the original ARM use cases and alif live audio demos.. (Default is ON)

If running on two cores, one of the applications must have this disabled.

The Alif camera use cases switch over from the GLCD display to a full LVGL UI, and the LVGL UI functions independently of this option – it can be turned off to save RAM.

`-DCONSOLE_UART=<2|4>`

Specifies which UART to use for console. (Default is 2)

#### **-DLINKER\_SCRIPT\_NAME**

Specifies a linker script/scatter file to use. The default is ensemble-RTSS-<HE|HP>, a layout which uses both TCM and SRAM0/SRAM1.

If running on two cores, the HE core must use the alternative ensemble-RTSS-HE-TCM layout which uses only TCM. This will only fit the smallest use-cases such as kws or alif\_kws, and GLCD\_UI must be disabled.

To fit in TCM the kws use case must have its activation area reduced using -  
Dkws\_ACTIVATION\_BUF\_SZ=0x20000. (This is already the default for alif\_kws).