eIQ™ Inference with Tensorflow Lite for Microcontrollers on i.MX RT685 -Without Camera

Lab Hand Out - Revision 3
August 2023

Contents

1 Lab Overview 3

2 Prerequisites 3

2.1 RT685 Software Installation 3

2.2 Create a model 3

3 Create Updated HiFi4 Binary 3

4 Modify Source Code 4

5 Compile and Run 6

5.1 Compile HiFi4 Code 6

5.2 Compile MCUXpresso IDE Code 7

5.3 Run Example 9

6 Conclusion 10

1 Lab Overview

This lab is specific to the i.MX RT685 device which includes a HiFi4 DSP. This can significantly reduce the inference times when running a model. It does require compiling the TFLM SDK example with Xtensa Explorer IDE and then integrating the resulting binary into the MCUXpresso SDK project.

2 Prerequisites

2.1 RT685 Software Installation

1. Follow all the instructions on the MIMXRT685-EVK Getting Started website. Before continuing with this lab you should have:
    - Installed Xtensa Xplorer IDE
    - Updated LPC-Link2 firmware on the EVK to use the J-Link interface
    - Successfully debugged the **dsp_mu_polling_cm33** SDK example with both MCUXpresso IDE and Xtensa Xplorer IDE.

2.2 Create a model

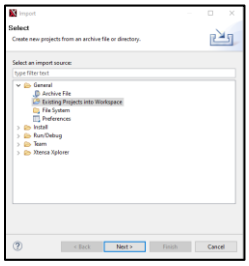
1. Follow all the instructions in the **eIQ TensorFlow Lite for Microcontrollers Lab for RT1170 - Without Camera.pdf** lab until Section 5. You should have:
    - Created a model
    - Converted the model into a **flower_model.h** file
    - Converted an example image into a **daisy.h** file
    - Created a **flower_labels.h** file

    Note: The default flower model created by eIQ Toolkit in the eIQ Toolkit lab is too large for the DSP RAM and will generate a compile error. A smaller model should be used. This lab will provide the instructions for using the HiFi4 DSP with a generic model.

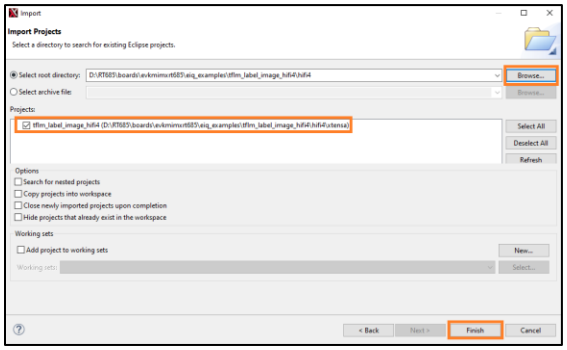
3 Create Updated HiFi4 Binary

Make sure the pre-requisites have been done in the previous section. The next step will be to take the files generated and the Xtensa Explorer project in the SDK and create an updated HiFi4 binary.

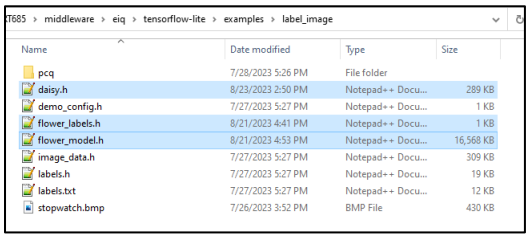
1. Unzip the i.MX RT685 SDK if not done already.
    - Make sure it is unzipped into a short filename path (ie C:\nxp\RT685), as an excessively long filename path can cause compile issues
    - Make sure the directory path contains no spaces
2. Open up Xtensa Xplorer.
3. Import the HiFi4 DSP for the TensorFlow for Microcontrollers (TFLM) project by going to File->Import
4. Expand the **General** category to select **Existing Projects into Workspace** and click on **Next**



5. Click on **Browse** next to **Select root directory** and navigate to where you unzipped the RT685 SDK. Select the **\boards\evkmimxrt685\eiq_examples\tflm_label_image_hifi4\hifi4** directory which contains the HiFi4 DSP project for the TFLM Label Image demo. The **tflm_label_image_hifi4** project should then already be selected. Leave all the other options unchecked as-is. Click on **Finish** to import this project.



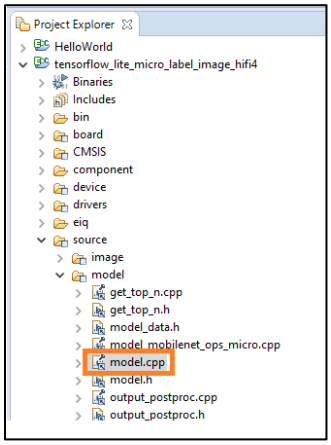
6. Now copy in the files generated from the previous section into **<SDK dir>\middleware\eiq\tensorflow-lite\examples\label_image**. It should look like the following when done:



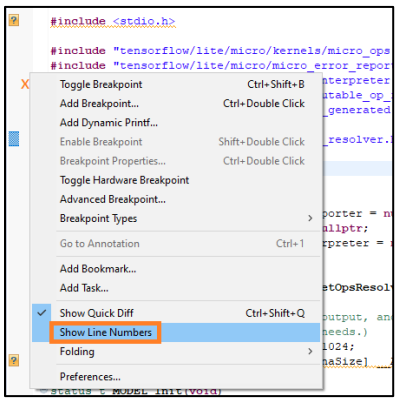
4 Modify Source Code

Now make the same edits as done in the i.MX RT1170 labs, except they'll be made in Xtensa Explorer instead of MCUXpresso IDE. These edits are:

1. Double click on the **model.cpp** file under the "source\model" folder in the Project View to open it.



2. Right click on the blank space next to the code and select Show Line Numbers to display the line numbers



3. On line 24 add the following #include for the ops resolver that supports all the operands used by this retrained model:
   #include "tensorflow/lite/micro/all_ops_resolver.h"

4. On line 28, comment out original #include for the original model defined in **model_data.h**. Then add a new #include to bring in the new model with **flower_model.h**. It should look like the following when finished:

```
23 #include "tensorflow/lite/schema/schema_generated.h"
24 #include "tensorflow/lite/micro/all_ops_resolver.h"
25
26 #include "fsl_debug_console.h"
27 #include "model.h"
28 //#include "model_data.h"
29 #include "flower_model.h"
```

5. On line 44, change the model name to the array name in **flower_model.h**:

```
40 status_t MODEL_Init(void)
41 {
42     // Map the model into a usable data structure. This doesn't involve any
43     // copying or parsing, it's a very lightweight operation.
44     s_model = tflite::GetModel(flower_model_tflite);
45     if (s_model->version() != TFLITE_SCHEMA_VERSION)
46     {
```

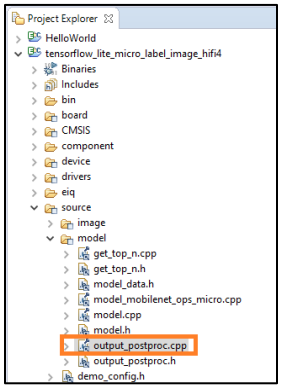
6. To reduce the size of the project, the Label Image example only supports the specific operands required by the default Mobilenet model. Our retrained model uses a few new operands. These specific operands can be determined by analyzing the model with an application called **netron** and then manually add the operands as described in Section 7.1 of the eIQ TensorFlow Lite Library User's Guide. Or alternatively all the TFLite operands can be supported in a project by using the built in **tflite::AllOpsResolver** method. For this lab we'll use the latter method in order to provide the greatest compatibility with other models. On line 61 in model.cpp, comment out the original resolver line. Then add a new line

   tflite::AllOpsResolver micro_op_resolver;

   It will look like the following when done:

```
61     //tflite::MicroOpResolver &micro_op_resolver = MODEL_GetOpsResolver();
62     tflite::AllOpsResolver micro_op_resolver;
```

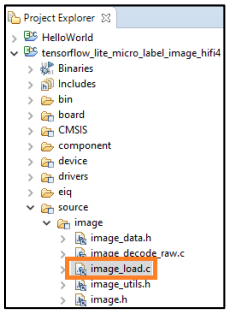
7. Next open the **output_postproc.cpp** file.



8. On line 12, comment out original #include for the original label file. Then add a new #include to bring in the new labels file. It should look like the following when finished:

```
12 //#include "labels.h"
13 #include "flower_labels.h"
```

9. Next open the **image_load.c** file under **source\image**



10. Make the following changes to bring in the daisy image for inferencing:
    a. Comment out the #include on line 12 in **image_load.c** and add **#include "daisy.h".**

```
12 //#include "image_data.h"
13 #include "daisy.h"
```

    b. Change line 25 for the IMAGE_Decode argument to the name of the daisy array, **daisy**

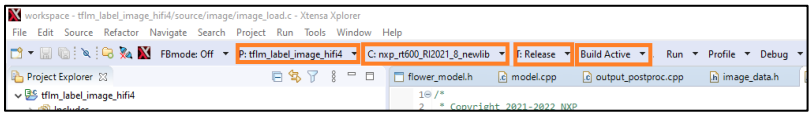
```
18 status_t IMAGE_GetImage(uint8_t* dstData, int32_t dstWidth, int32_t dstHeight, int32_t dstChannels)
19 {
20     s_staticCount++;
21     /* Single static sample only */
22     if (s_staticCount == 1)
23     {
24         PRINTF(EOL "Static data processing:" EOL);
25         return IMAGE_Decode(daisy, dstData, dstWidth, dstHeight, dstChannels);
26     }
27     else
```

5 Compile and Run

5.1 Compile HiFi4 Code

With all the necessary edits made, it's now time to generate the HiFi4 DSP binary.

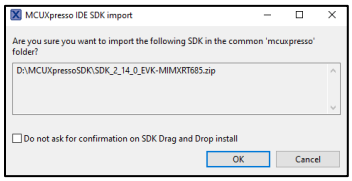
7. In the toolbar, select the **tflm_label_image_hifi4** project, select the **nxp_rt600_RI2021_8_newlib** library, select the **Release** target, and then click on **Build Active** to build the neural network DSP library. You will see the results in the Console tab at the bottom. If there are any errors, make sure the SDK directory path does not have any spaces.



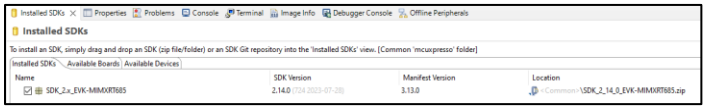
8. Remember that if using the default flower model that it may be too large and generate a compile error, so a smaller model may need to be used. You can also just use the default Mobilenet model.
9. When the compilation is complete, two .bin file will be generated in **<SDK_Path>\boards\evkmimxrt685\eiq_examples\tflm_label_image_hifi4\hifi4\binary**
10. These binary files will replace the default DSP library files in the MCUXpresso SDK project so make note of the location because it will be used in the next section.

5.2 Compile MCUXpresso IDE Code

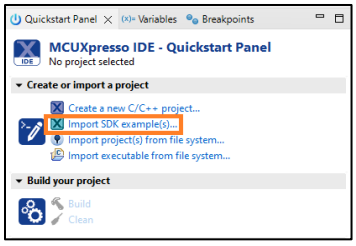
1. Open up MCUXpresso IDE and select a new workspace
2. Install the RT685 SDK into the "Installed SDKs" tab by dragging-and-dropping the **RT685 SDK .zip** downloaded earlier into the Installed SDK window located in the bottom center. This dialog box will come up, and click OK to continue the import:



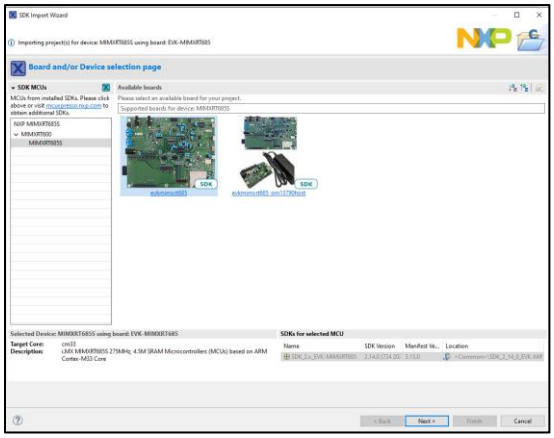
3. It will look like the following when complete:



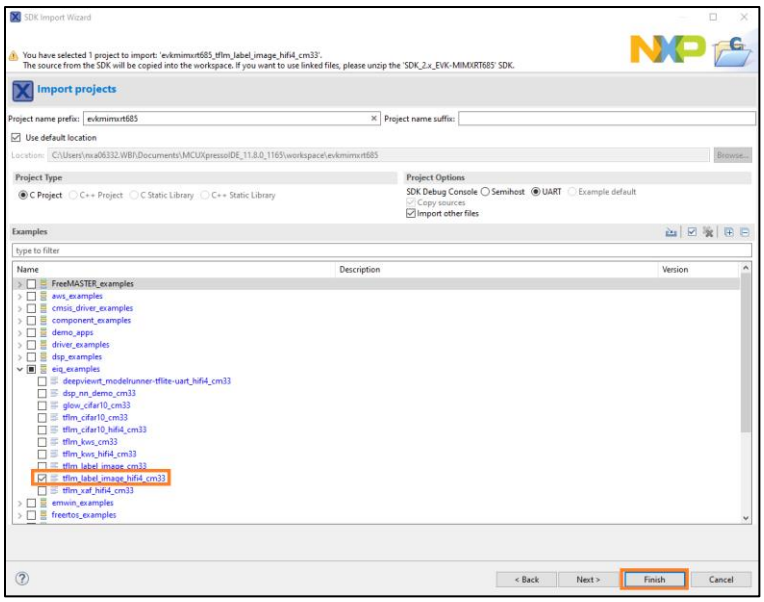
4. In the Quickstart Panel in the lower left corner, click on Import SDK examples(s)…



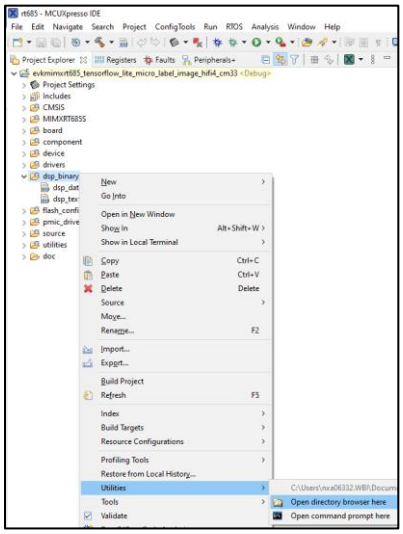
5. Select the **evkmimxrt685** board and click on Next



6. Expand the **eIQ_examples** category and select the **tflm_label_image_hifi4_cm33** example. Make sure it's the HiFi4 version. Click on **Finish**.

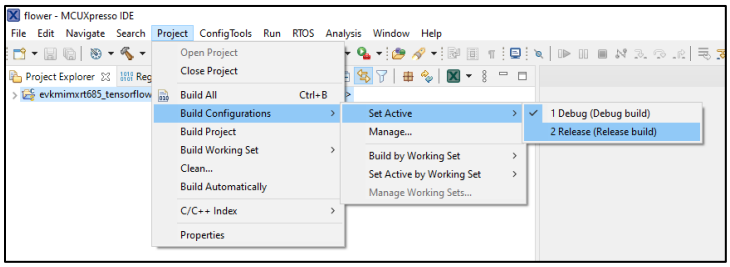


7. This will copy the TLFM demo project to the MCUXpresso IDE workspace.
8. In the Project Explorer tab, right click on the **dsp_binary** folder and go to **Utilities->Open directory browser here** to open up a Windows Explorer at the directory where the default DSP binaries are located.

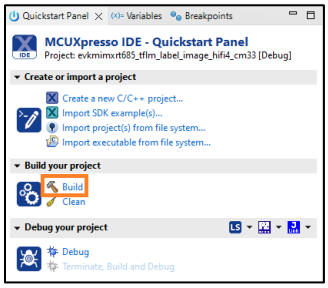


9. In that directory, copy and overwrite the default **dsp_data_release.bin** and **dsp_text_release.bin** files with the new DSP binary files that were generated in the previous section.

11. Change the build configuration to “Release” by clicking on the project and going to the menu bar and going to **Project->Build Configurations->Set Active->Release**. This will enables high compile optimizations.

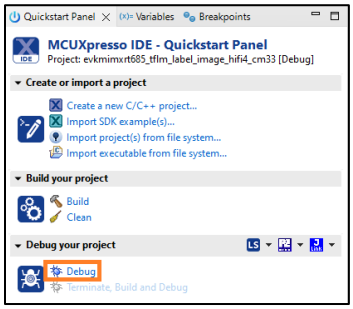


12. Build the project by clicking on “Build” in the Quickstart Panel and make sure there are no errors.

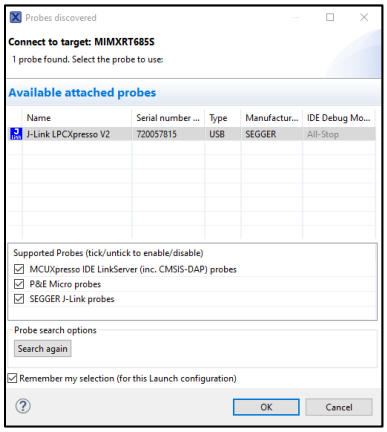


5.3 Run Example

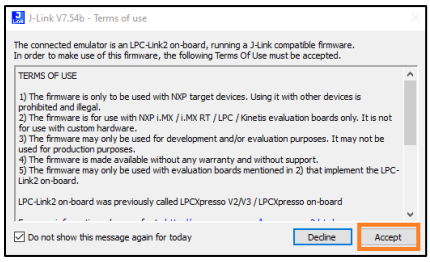
13. Plug the micro-B USB cable into the board at J5 on the i.MX RT685 board.
14. Open TeraTerm or other terminal program, and connect to the COM port that the board enumerated as. Use 115200 baud, 1 stop bit, no parity.
15. Debug the project by clicking on “Debug” in the Quickstart Panel.



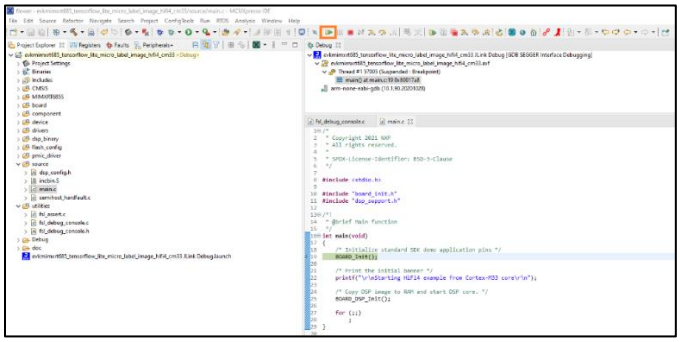
16. It will ask what interface to use. Select JLink.



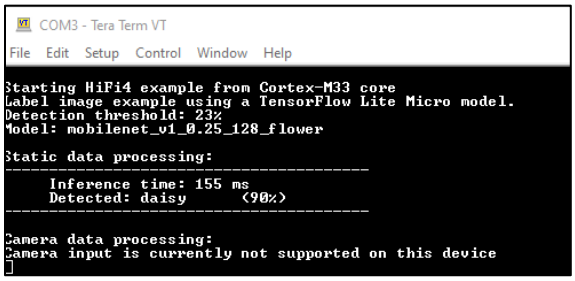
17. You may get the following message. Put a checkmark on “Do not show this message again today” and then click on **Accept**.



18. The debugger will download the firmware and open up the debug view. It may take some time to download the firmware. Click on the Resume button to start running.



19. Open up a terminal window, and you the result of the inference from the static image:



6 Conclusion

This lab demonstrated how to use TensorFlow to generate a retrained TensorFlow model in TFLite format that can be imported and ran on an embedded system using the eIQ software platform.

It’s also important to note that the HiFi4 implementation does not support float32 so the model should be quantized before running on the HiFi4.