



# eIQ™ Inference with Glow for i.MX RT685

---

Lab Hand Out - Revision 8

August 2023

---

# Contents

<b>1 Lab Overview .....</b>	<b>3</b>
<b>2 Software and Hardware Installation .....</b>	<b>3</b>
2.1 Glow Installation.....	3
2.2 ML Installation.....	3
<b>3 Convert models .....</b>	<b>4</b>
<b>4 LeNet MNIST Example.....</b>	<b>4</b>
<b>5 Run Glow on the RT685.....</b>	<b>9</b>
<b>6 Compile Neural Network HiFi4 DSP.....</b>	<b>16</b>
6.1 RT685 Software Installation .....	16
6.2 Compile HiFi4 DSP Libraries.....	16
<b>7 Conclusion.....</b>	<b>17</b>

---

## 1 Lab Overview

---

This lab will cover how to compile a model using Glow and run it on an RT685 board. The compiled model will run on the Cortex-M33 core and can also make use of the HiFi4 DSP for supported operations.

The MNIST models used as the example for this lab are no longer available online and Glow is no longer a focus of development, so this lab will not be updated. It is recommended to use TensorFlow Lite for Microcontrollers for any new products due to better compatibility with newly developed models. This lab document will be kept online for any current users of Glow.

---

## 2 Software and Hardware Installation

---

This section will cover the steps needed to install the eIQ software and common machine learning applications on your computer.

### 2.1 Glow Installation

1. Install [MCUXpresso IDE](#)
2. Install a terminal program like [TeraTerm](#).
3. Install the latest [Glow package](#). This will put the Glow compiler and helper programs into `C:\NXP\Glow` and add it to your executable path.
4. Download [MCUXpresso SDK for RT685](#). Make sure to include the “eIQ” middleware option. Also make sure to download the MCUXpresso SDK docs zip as well.
5. Unzip the RT685 SDK zip file into a directory path that does not contain any spaces.
6. If not done already, follow the instructions on the [MIMXRT685-EVK Getting Started website](#) to update the LPC-Link2 firmware on the RT685-EVK to use the J-Link interface

### 2.2 ML Installation

The following instructions are for install basic ML libraries with Python. Not all of these are required when using Glow, but can be helpful for training models and using scripts.

1. Download and install Python 3.10. **\*\*The 64-bit edition is required and for compatibility it is highly recommended to use Python 64-bit 3.10.x\*\***: <https://www.python.org/downloads/>
2. Open a Windows command prompt and verify that the python command corresponds to Python 3.10.x.  
**python -V**
3. Update the python installer tools:  
**python -m pip install -U pip**  
**python -m pip install -U setuptools**
4. Install other useful python packages. Not all of these will be used for this lab but will be useful for other eIQ demos and scripts.  
**python -m pip install numpy scipy matplotlib ipython jupyter pandas sympy nose imageio**  
**python -m pip install netron seaborn west pyserial sklearn opencv-python pillow**

### 3 Convert models

The Glow tools can compile models that are in the [TensorFlow Lite](#), [ONNX](#), and [Caffe2](#) format. More conversion options can be found in Section 6.1 of the [eIQ Glow Ahead of Time User Guide](#).

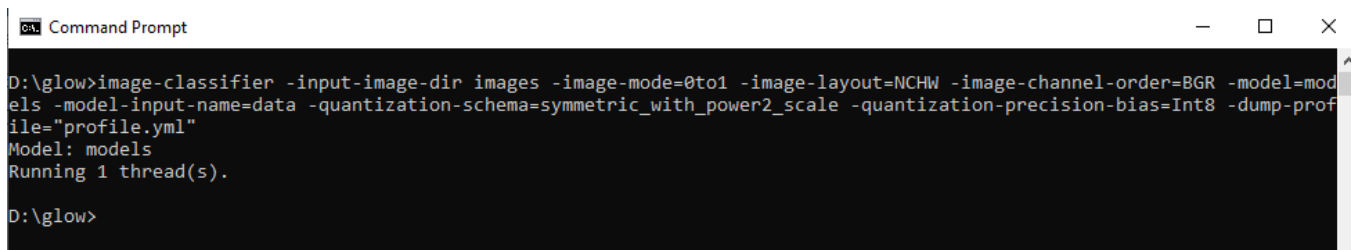
### 4 LeNet MNIST Example

The following steps can be used to run an LeNet MNIST example which does hand-written digit recognition. More details can be found in the [eIQ Glow User Guide](#).

1. The fastest inference performance is achieved by quantizing the model, and the best way to quantize the model without losing much accuracy is to create a quantization profile. Because different parts of the neural network contain floating point values in different ranges, Glow uses profile-guided quantization to estimate the possible numerical range for each stage of the network. The **image-classifier** tool generates a **profile.yml** file that can be used to optimize quantization when compiling the model. Generating this profile file requires a small subset of images to analyze, which were downloaded in the previous step. More details on how Glow utilizes quantization can be found on the [Glow website](#).

This should be one long continuous line:

```
image-classifier -input-image-dir images -image-mode=0to1 -image-layout=NCHW
-image-channel-order=BGR -model=models -model-input-name=data
-quantization-schema=symmetric_with_power2_scale -quantization-precision-bias=Int8
-dump-profile="profile.yml"
```



```
Command Prompt
D:\glow>image-classifier -input-image-dir images -image-mode=0to1 -image-layout=NCHW -image-channel-order=BGR -model=models
-quantization-schema=symmetric_with_power2_scale -quantization-precision-bias=Int8 -dump-profile="profile.yml"
Model: models
Running 1 thread(s).
D:\glow>
```

**Note:** If copying and pasting the text above into a Windows command line, there may be "?" instead of hyphens "-", but the command will still work.

2. Here's an explanation for the arguments in the command you just ran. You can also use "image-classifier -help":
  - **-input-image-dir images**  
The directory location of the PNG images to perform the profiling on.
  - **-image\_mode=0to1**  
Specifies range of values for input tensor. In this example it expects values between [0,1]. Other options are [-1,1], [-128,127], or [0,255].

- **-image\_layout=NCHW**  
Specifies image layout to use. It's important to preprocess the input images the same way they were processed when training the model.  
NCHW is Num x Channels x Height x Width.  
NHWC is Num x Height x Width x Channels.
  - **-image-channel-order=BGR**  
Specifies image channel order. Could be Blue-Green-Red or Red-Green-Blue.
  - **-model=models**
    - For Caffe2 models, directory containing the Caffe2 model files named `init_net.pb` and `predict_net.pb` that will be compiled by Glow
    - For ONNX models, it should be set to the ONNX model file name.
  - **-model-input-name=data**  
Name of the input layer of the model. For this MNIST model it is named "data"
  - **-quantization-precisison=Int8**  
Use Int8 bias quantization. Needed for CMSIS-NN optimizations.
  - **-quantization-schema=symmetric\_with\_power2\_scale**  
Quantization schema. Symmetric with Power 2 scale is needed for CMSIS-NN optimizations.
  - **-dump-profile=profile.yml**  
Filename to store the profiling results
3. **This step is optional and not required for this lab.** It is then possible to further tune this quantization profile to provide even better accuracy. This is not required for this lab as the default quantization is already accurate for this model, but this step is included here for completeness. This highly optimized quantization profile will be named **profile\_tuned.yml** to differentiate it from the basic profiling done in the previous step. Using the **model-tuner** tool causes the accuracy of the quantized Glow model to significantly increase. It uses a labeled dataset listed in a CSV file. The **eIQ Glow Ahead of Time User Guide** in the SDK docs zip file has more details on how this tuning works.
- **Note:** The more images used from across all categories will help improve accuracy. Example image files would need to be created and those are not included as part of this lab.
  - **Note:** This step can take a very long time without the **-target-accuracy** parameter, up to several hours for large datasets.
  - **Note:** If copying and pasting the text above into a Windows command line, there may be "?" instead of hyphens "-", but the command will still work.

This should be one long continuous line:

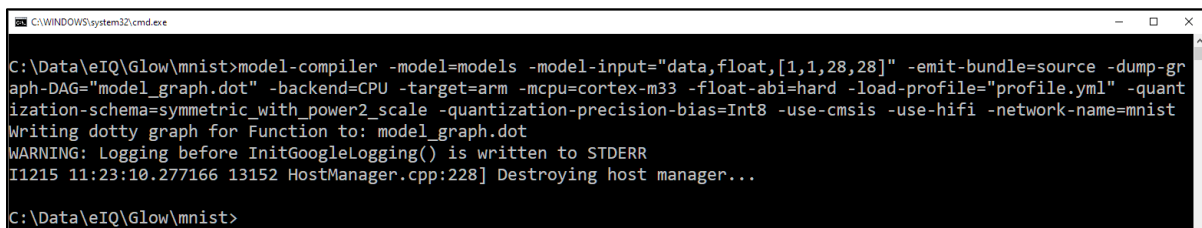
```
model-tuner -dataset-file="dataset-tuning\Labels.csv" -dataset-path=dataset-tuning
-image-mode=0to1 -image-layout=NCHW -image-channel-order=BGR -model=models
-model-input="data,float,[1,1,28,28]" -load-profile="profile.yml"
-dump-tuned-profile="profile_tuned.yml" -backend=CPU -quantization-precision=Int8
-quantization-schema=symmetric_with_power2_scale -target-accuracy="0.95"
```

4. Here's an explanation for the new arguments in the command you just ran. You can also use "model-tuner -help":
  - **-dataset-file="dataset-tuning\Labels.csv"**  
 CSV file containing image names and labels in *image\_file\_name,category\_label, format*. Ex:
 

```
4_1059.png,4,
0_1009.png,0,
8_1026.png,8,
```
  - **-dataset-path=dataset-tuning**  
 Directory path to the images specified in the CSV file
  - **-model-input="data,float,[1,1,28,28]"**
    - For Caffe2 models like this MNIST model, specify the input tensor (data), the input layer type (float), and the shape ([1,1,28,28]). This is a 28x28 image.
    - For ONNX models, this option is not needed as this information can be read directly from the model.
  - **-load-profile="profile.yml"**  
 Load the profile file generated from the previous step
  - **-dump-tuned-profile="profile\_tuned.yml"**  
 Filename to store the highly tuned profile results
  - **-backend=CPU**  
 Use CPU as the backend
  - **-quantization-precison=Int8**  
 Use Int8 bias quantization. Needed for CMSIS-NN optimizations.
  - **-quantization-schema=symmetric\_with\_power2\_scale**  
 Quantization schema. Symmetric with Power 2 scale is needed for CMSIS-NN optimizations.
  - **-target-accuracy="0.95"**  
 Stop tuning when accuracy has reached 95%. This saves significant amounts of time.
5. The profile file created in Step 4 will then be used to generate the compiled Glow executable by using the **model-compiler** tool. Generate the files with the following command:

This should be one long continuous line:

```
model-compiler -model=models -model-input="data,float,[1,1,28,28]" -emit-bundle=source
-dump-graph-DAG="model_graph.dot" -backend=CPU -target=arm -mcpu=cortex-m33
-float-abi=hard -load-profile="profile.yml"
-quantization-schema=symmetric_with_power2_scale -quantization-precision-bias=Int8
-use-cmsis -use-hifi -network-name=mnist
```



```
C:\Data\eiQ\Glow\mnist>model-compiler -model=models -model-input="data,float,[1,1,28,28]" -emit-bundle=source -dump-gr
aph-DAG="model_graph.dot" -backend=CPU -target=arm -mcpu=cortex-m33 -float-abi=hard -load-profile="profile.yml" -quant
ization-schema=symmetric_with_power2_scale -quantization-precision-bias=Int8 -use-cmsis -use-hifi -network-name=mnist
Writing dotty graph for Function to: model_graph.dot
WARNING: Logging before InitGoogleLogging() is written to STDERR
I1215 11:23:10.277166 13152 HostManager.cpp:228] Destroying host manager...
C:\Data\eiQ\Glow\mnist>
```

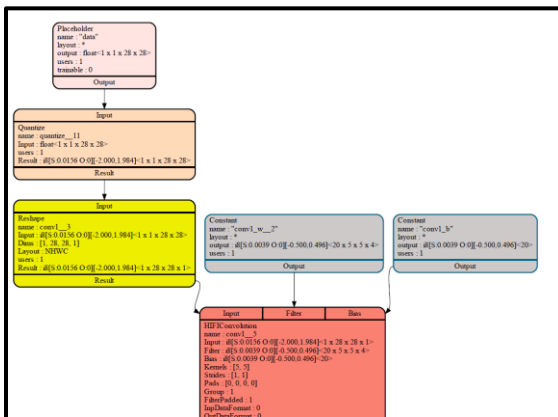
**Note:** If copying and pasting the text above into a Windows command line, there may be "?" instead of hyphens "-", but the command will still work.

6. Here's an explanation for the arguments in the command you just ran. You can also use "model-compiler -help":
- **-model=models**
    - For Caffe2 models, directory name that contains the Caffe2 model files named `init_net.pb` and `predict_net.pb` that will be compiled by Glow
    - For ONNX models, it should be set to the ONNX model file name.
  - **-model-input=data,float,[1,1,28,28]**
    - For Caffe2 models like this MNIST model, specify the input tensor (`data`), the input layer type (`float`), and the shape (`[1,1,28,28]`). This is a 28x28 image.
    - For ONNX models, this option is not needed as this information can be read directly from the model.
  - **-emit-bundle=source**  
Directory to output the generated files
  - **-dump-graph-DAG=model\_graph.dot**  
Generates a visual representation of the compiled model in dot format. See further below for how to convert the dot file to a PDF file.
  - **-backend=CPU**  
Use CPU as the backend
  - **-target=ARM**  
Target architecture to compile for
  - **-mcpu=cortex-m33**  
Specific CPU to compile for.
    - "cortex-m7" for M7 core (RT1050/RT1060/RT1160/RT1170).
    - "cortex-m33" for M33 core (RT685)
    - "cortex-m4" for M4 core.
  - **-float-abi=hard**  
Compile to use floating point hardware on target
  - **-load-profile=profile.yml**  
Load the profile file generated earlier. This option is also what tells the model-compiler to quantize the model. The model must be quantized to use the CMSIS-NN performance optimizations. Note that if the optional model-tuner profile was generated, would use the 'profile\_tuned.yml' file instead.
  - **-quantization-precision=Int8**  
Use Int8 bias quantization.
  - **-quantization-schema=symmetric\_with\_power2\_scale**  
Quantization schema.
  - **-use-cmsis**  
Use CMSIS-NN library for supported quantized operations to speed up execution.
  - **-use-hifi**  
Use HiFi4 DSP on RT685 for supported operations to speed up execution. Not used for RT1050/RT1060.
  - **-network-name=mnist**  
Use "mnist" as the name for the generated files. It is recommended to use a short and descriptive name as it will be used as a prefix for all macros and functions.

- You can see the generated files inside the **source** folder defined by the **emit-bundle** argument.
  - <network\_name>.o - the bundle object file (code).
  - <network\_name>.h - the bundle header file (API).
  - <network\_name>.weights.bin - the model weights in binary format.
  - <network\_name>.weights.txt - the model weights in text format as C text array.
- The model\_graph.dot file can be converted to a PDF file by using the Dot application included in the Glow bundle. You can see the calls to the HiFi4 DSP for the supported layers.
 

```
dot -Tpdf model_graph.dot -o model_graph.pdf -Nfontname="Times New Roman,"
```

```
C:\WINDOWS\system32\cmd.exe
C:\Data\eiQ\Glow\mnist>dot -Tpdf model_graph.dot -o model_graph.pdf -Nfontname="Times New Roman,"
C:\Data\eiQ\Glow\mnist>
```



- Finally to test the accuracy of the model when running on an embedded system, convert an image into a C array. This array will be loaded into Flash and used as input for the model inference on the board. The **glow\_process\_image.py** Python script can be found attached to the NXP Community post that this lab was in. This script will generate a **input\_image\_test.bin** file that will be used in the next section. Notice the generated file is 28\*28\*4=3136 bytes in size because of the 28x28 pixel monochromatic image which uses 4-byte floating point input that the model requires.

This should be one long continuous line:

```
python glow_process_image.py -image-path="images\9_1088.png"
-output-path="source\input_image_test.bin" -image-mode=0to1 -image-layout=NCHW
-image-channel-order=BGR -image-type=float32 -output-format=bin
```

```
Command Prompt
D:\work\eiQ\Glow\mnist>python glow_process_image.py -image-path="images\9_1088.png" -output-path="source\input_image_test.bin" -image-mode=0to1 -image-layout=NCHW -image-channel-order=BGR -image-type=float32 -output-format=bin
Image processed to file "source\input_image_test.bin" ...
Image size = 3136 (bytes)
D:\work\eiQ\Glow\mnist>
```



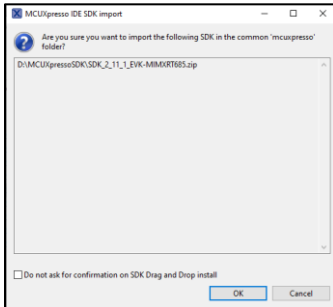
If you get an error about no module named numpy.typing make sure you have numpy version 1.20 by using `'python -m pip install numpy=1.20'`

## 5 Run Glow on the RT685

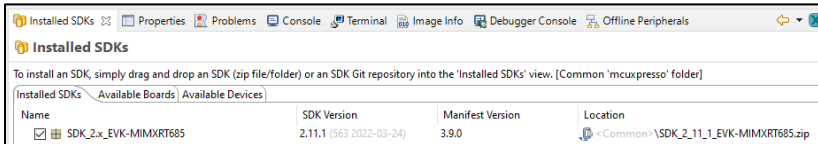
Now that the Glow files have been generated, the next step is to use them in the Glow MCUXpresso IDE project and run it on the RT685 board.

This lab will use the glow\_cifar10 demo found in the SDK and use it as a baseline to run the MNIST Glow model compiled in the previous section.

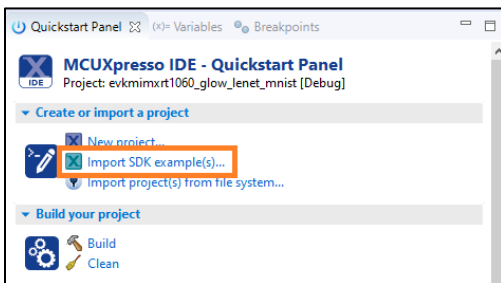
1. Open up MCUXpresso IDE and select a new workspace
2. Install the RT685 SDK into the "Installed SDKs" tab by dragging-and-dropping the **RT685 SDK .zip** downloaded earlier into the Installed SDK window. This dialog box will come up, and click OK to continue the import:



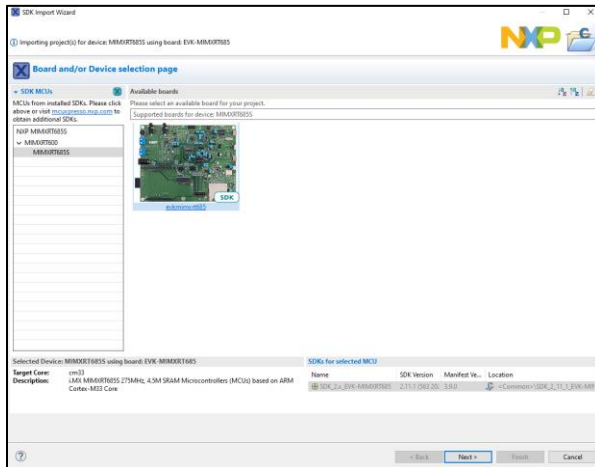
3. It will look like the following when complete:



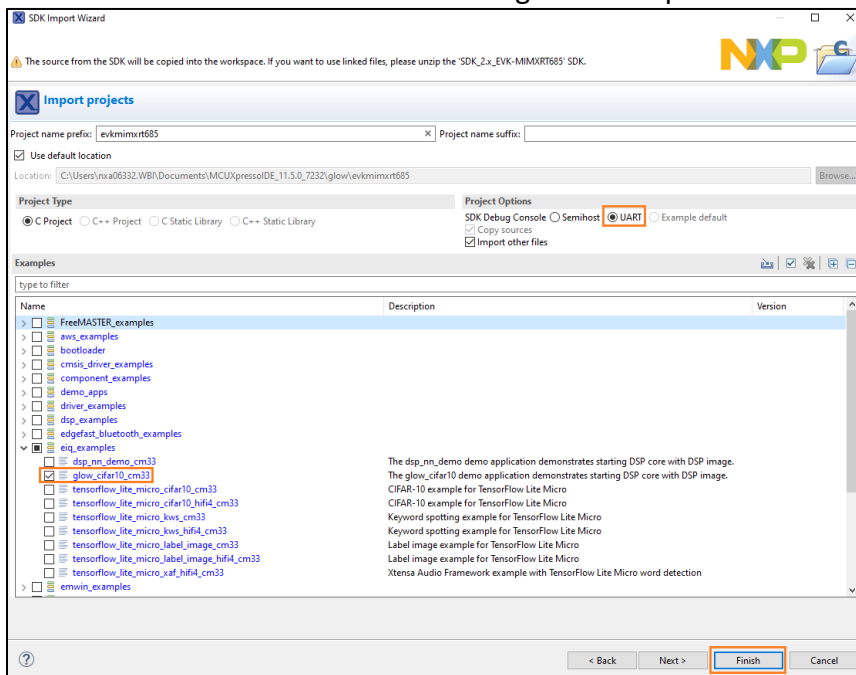
4. In the Quickstart Panel in the lower left corner, click on Import SDK examples(s)...



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



6. Expand the **elQ\_examples** category and select the **glow\_cifar10\_cm33** example. Then make sure **UART** is selected for the SDK Debug Console option. Click on **Finish**.

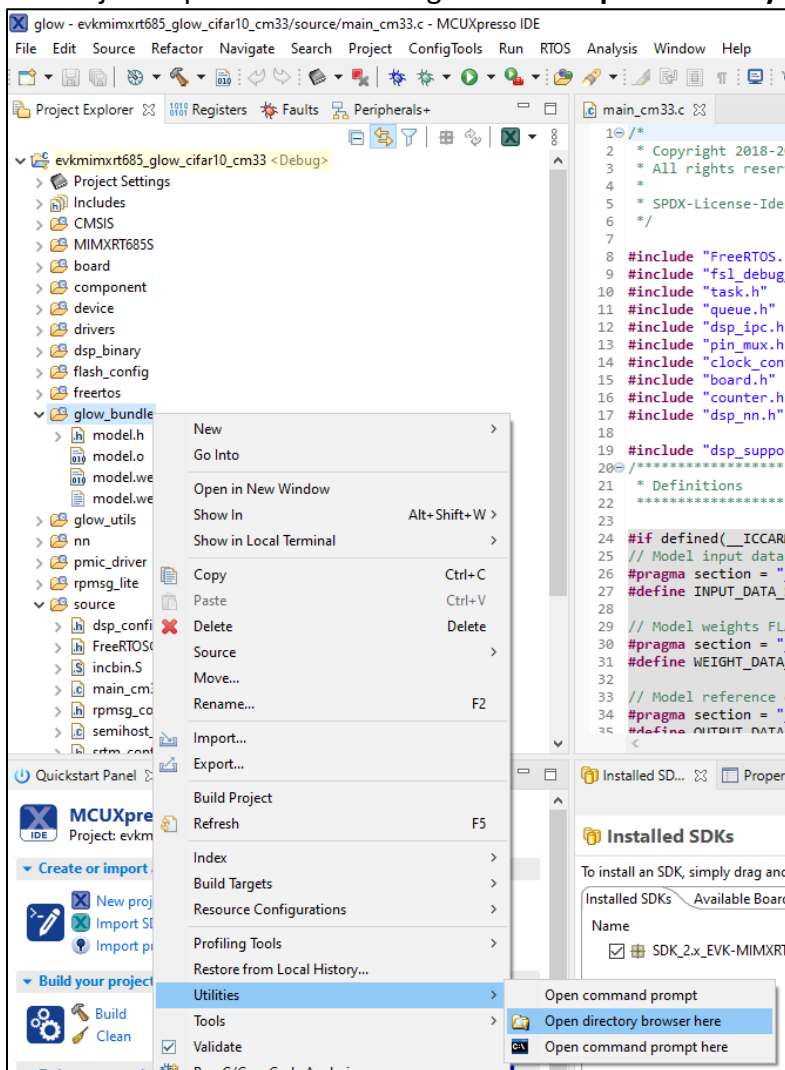


7. This will copy the CIFAR10 demo project to the MCUXpresso IDE workspace.
8. The project already includes the pre-compiled HiFi4 binaries for neural network acceleration. Section 6 will cover how to use Xtensa Xplorer to compile the DSP code, but for this section, that code will be automatically loaded by the project.

9. Now we need to add the files generated in the previous section into this project, which will be covered in the next few steps. There are 4 files that need to be included. Three of them are found in the “source” folder that was created by the compiler. Remember that the “mnist” name comes from the “-network-name” argument you gave when running **model-compiler**. **input\_image\_test.bin** comes from the python script to generate the data to do the inferencing on.

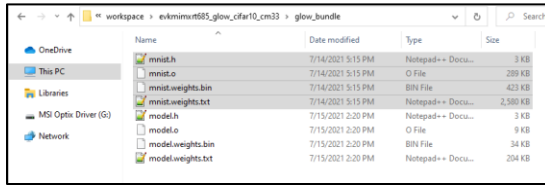
- mnist.h
- mnist.o
- mnist.weights.txt
- input\_image\_test.bin

10. Open the directory location to place these files by right clicking on the **glow\_bundle** folder in the Project Explorer and selecting **Utilities->Open directory browser here**

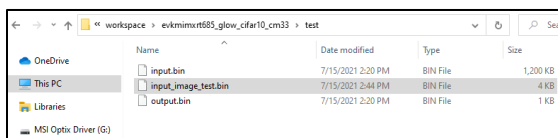


11. It should open a directory at something like something like: **C:\Users\user\_name\Documents \MCUXpressoIDE\_11.5.0\_7232\workspace\evkmimxrt685\_glow\_cifar10\_cm33\glow\_bundle**

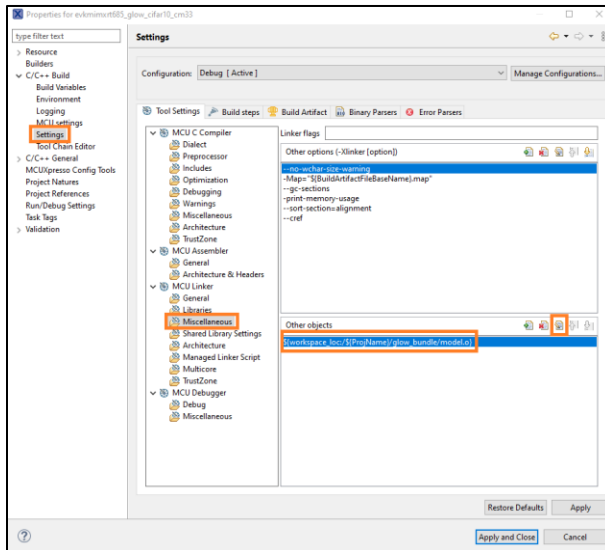
- Find the **mnist.h**, **mnist.o**, **mnist.weights.txt**, and **mnist.weights.bin** files from the Glow source directory created from the previous section, and copy those files into the **evkmimxrt685\_glow\_cifar10\_cm33\glow\_bundle** directory. It will look like the following when done:



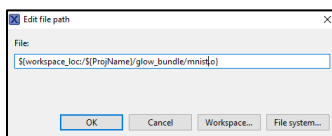
- Copy the **input\_image\_test.bin** file that was created from the previous section, and copy that file into the **evkmimxrt685\_glow\_cifar10\_cm33\test** directory. It should look like the following when done:



- Now go back to MCUXpresso IDE
- Next open up the Project Properties by clicking on the project name again and select Properties like done before.
- Now go to the **C/C++ Build->Settings->MCU Linker->Miscellaneous** screen and double click on the item (or hit the edit button) in “Other Objects” to modify the object file to be the one that was just created:

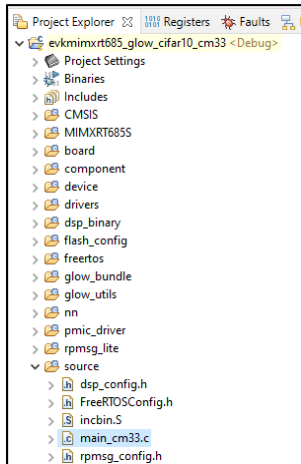


- Click on Workspace, navigate to the **glow\_bundle** folder, and select “mnist.o”. This will add the mnist model that was compiled earlier. Then hit OK.



- Then click on “Apply and Close” to close out of the Properties dialog box.

19. Now there are modifications that need to be made to **main\_cm33.c** to edit the file to use the new file names and models.



20. Everywhere in the **main\_cm33.c** file that **cifar10** is used, it should be changed to just “**mnist**” and use the new variable names created by the generated files. This includes:

- Line 52 to include the generated header file “**mnist.h**”
- Lines 56-65 for the new Glow variable names that use **MNIST** as the network name
- Line 68 should set the **inputAddr** pointer to the network name plus the name of the model’s input layer (**MNIST\_data** in this example). This name be found in the **minst.h** file.
- Line 69 should set the **outputAddr** pointer to the network name plus the name of the model’s output layer (**MNIST\_softmax** in this example). This name be found in the **minst.h** file.
- Line 72 should be set to the input size of the model (in this case a 28x28 monochrome image with floating point input).
- Line 75 should be set to the number of classes of the model
- Line 78 should be the number of images being inferred. In this case that is **1**

```

50
51 //----- Glow Bundle API -----
52 #include "mnist.h"
53 #include "glow_bundle_utils.h"
54
55 // Statically allocate memory for constant weights.
56 GLOW_MEM_ALIGN(MNIST_MEM_ALIGN)
57 uint8_t constantWeight[MNIST_CONSTANT_MEM_SIZE];
58
59 // Statically allocate memory for mutable weights (model input/output data).
60 GLOW_MEM_ALIGN(MNIST_MEM_ALIGN)
61 uint8_t mutableWeight[MNIST_MUTABLE_MEM_SIZE];
62
63 // Statically allocate memory for activations (model intermediate results).
64 GLOW_MEM_ALIGN(MNIST_MEM_ALIGN)
65 uint8_t activations[MNIST_ACTIVATIONS_MEM_SIZE];
66
67 // Bundle input/output data absolute addresses.
68 uint8_t *bundleInpAddr = GLOW_GET_ADDR(mutableWeight, MNIST_data);
69 uint8_t *bundleOutAddr = GLOW_GET_ADDR(mutableWeight, MNIST_softmax);
70
71 // Model input data size (bytes).
72 #define INPUT_IMAGE_SIZE 28 * 28 * 1 * sizeof(float)
73
74 // Model number of output classes.
75 #define OUTPUT_NUM_CLASS 10
76
77 // Number of images to run inference
78 #define NUM_IMAGES 1
79

```

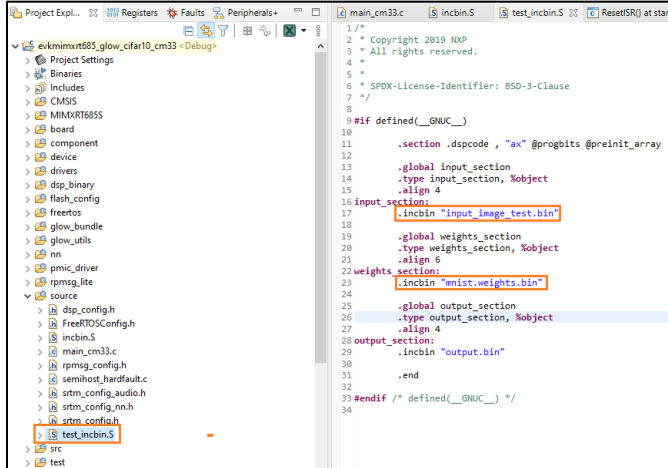
21. Then modify Line 122 which is what starts the inference by changing the function name: **“mnist(constantWeight, mutableWeight, activations)”**

```

121     tic = get_ccount();
122     mnist(constantWeight, mutableWeight, activations);
123     toc = get_ccount();

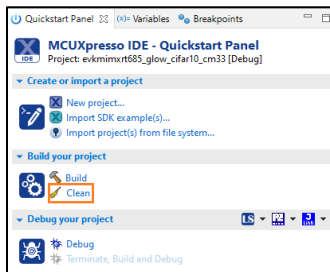
```

22. Then open up the **test\_incbin.S** file and change the input and weights sections to match the new input data and the new weights data:

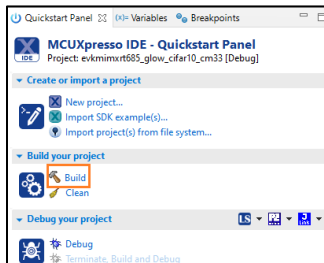


23. **Optional:** The **serialize\_data.py** script found at **\boards\evkmimxrt685\eiq\_examples\glow\_cifar10\scripts** could also be used to generate input data to read in for inferecing if multiple images need to be inferred. It is not used for this lab though.

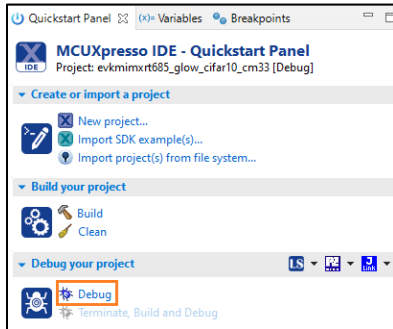
24. Now it’s time to build the project. However because new Glow files were copied into the project, you must do a clean first. Failing to do a clean could cause the newly imported weight data to become misaligned in memory, and cause accuracy errors during the inferecing. This is only required when new Glow files are copied into the project. Click on “Clean” in the Quickstart Panel first:



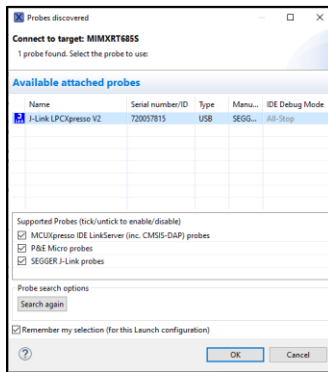
25. Then build the project by clicking on “Build” in the Quickstart Panel.



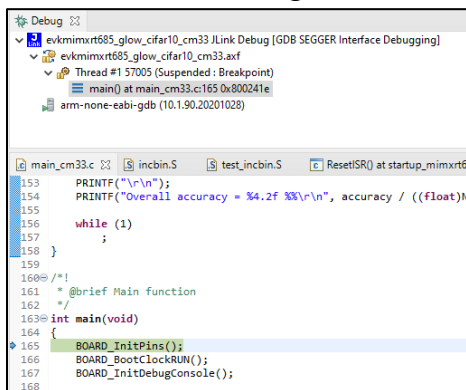
26. Plug the micro-B USB cable into the board at J5.
27. Open TeraTerm or other terminal program, and connect to the COM port that the board enumerated as. Use 115200 baud, 1 stop bit, no parity.
28. Debug the project by clicking on “Debug” in the Quickstart Panel.



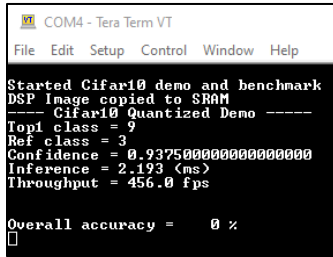
29. It will ask what interface to use. Select **J-Link LPCXpresso V2** as the LPC-Link2 debug circuit should have been updated to use the JLink interface as part of the RT685 Getting Started.



30. The debugger will download the firmware and open up the debug view. Click on the Resume button to start running.



- You should see the output on the console identifying the image as the number '9' with ~93% confidence and a 2.19ms inference time. Ignore the Ref class item and the Overall accuracy as it assumes the output.bin file has that correct information, but that is not being used in this lab:



```

COM4 - Tera Term VT
File Edit Setup Control Window Help
Started Cifar10 demo and benchmark
DSP Image copied to SRAM
---- Cifar10 Quantized Demo ----
Top1 class = 9
Ref class = 3
Confidence = 0.9375000000000000
Inference = 2.193 (ms)
Throughput = 456.0 fps

Overall accuracy = 0 %
  
```

## 6 Compile Neural Network HiFi4 DSP

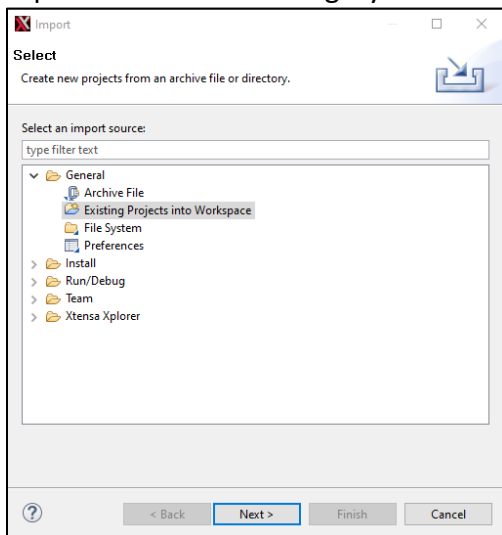
This section is not required but may be of interest to those who want to study the HiFi4 DSP code. However the HiFi4 code should not normally need to be modified. The project in the previous section already includes the pre-compiled DSP code. To explore the DSP code, Xtensa Xplorer can be used to compile the neural network HiFi4 DSP libraries.

### 6.1 RT685 Software Installation

- 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.

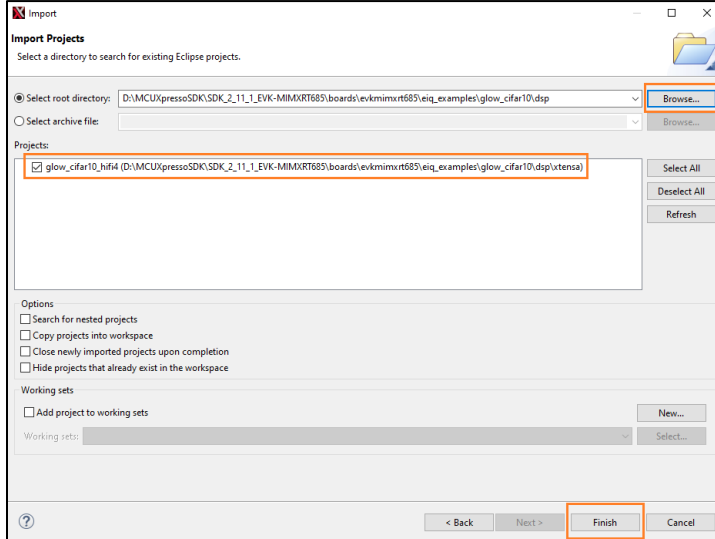
### 6.2 Compile HiFi4 DSP Libraries

- Open up Xtensa Xplorer.
- Import the DSP project by going to File->Import
- Expand the **General** category to select **Existing Projects into Workspace** and click on **Next**

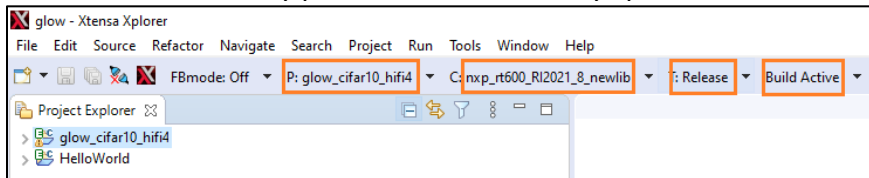




- Click on **Browse** next to **Select root directory** and navigate to where you unzipped the RT685 SDK. Select the `\boards\evkmimxrt685\eiq_examples\glow_cifar10\dsp` directory which contains the DSP project for the CIFAR-10 demo. The `glow_cifar10_hifi4` project should then already be selected. Leave all the other options unchecked as-is. Click on **Finish** to import this project.



- In the toolbar, select the `glow_cifar10_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.



- When the compilation is complete, two .bin file will be generated in `<SDK_Path>\boards\evkmimxrt685\eiq_examples\glow_cifar10\dsp\binary`
- These binary files can replace the default DSP library files in the project. Overwrite the default files by copying them to the `evkmimxrt685_glow_cifar10_cm33` directory in the MCUXpresso IDE workspace. It's the same directory location the Glow files were copied to in the previous section.

## 7 Conclusion

This lab demonstrates how to use the Glow tools to create executable models that will run on the RT685 device. For more information on the Glow tools, please read the [eiQ Glow Ahead of Time User Guide](#) in the SDK docs zip file, and also visit the [Glow AOT](#) (Glow Ahead of Time) website.

The default CIFAR-10 example can also be explored. Creating a CIFAR-10 model to convert requires installing Caffe2 and training a CIFAR-10 model which is beyond the scope of this lab document. See the `readme.txt` file at `<SDK_path>\boards\evkmimxrt685\eiq_examples\glow_cifar10\cm33` for more details.

The `serialize_data.py` script at `<SDK_path>\boards\evkmimxrt685\eiq_examples\glow_cifar10\scripts` can be used to rapidly test multiple images. It creates an `input.bin` (the input data) and `output.bin` (the correct result for each image) that can be placed into the `test` folder of the CIFAR-10 project to be compiled in.

Further performance optimizations making use of the internal memory found in the i.MX RT can be done by using the techniques described in [Section 5.3 of the Glow User Guide](#).

Finally, quantization profiles for models that do not use images can be created by using the `model-profiler` tool. It is included in this Glow release, and documentation can be found on the [Glow website](#).