



eIQ™ Inference with Glow for i.MX RT1170

Lab Hand Out - Revision 8

August 2023

Contents

1 Lab Overview	3
2 Software and Hardware Installation	3
2.1 Glow Installation.....	3
2.2 ML Installation.....	3
3 Convert models	4
4 LeNet MNIST Example.....	4
5 Run Glow on the RT1170	10
6 Conclusion.....	16

1 Lab Overview

This lab will cover how to compile a model using Glow and run that model on a RT1170-EVK board.

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.

This lab can also be used with the following evaluation boards that support Glow in the SDK by downloading their respective SDK packages. A [camera+LCD](#) is not required but is suggested.

- i.MX RT1050
- i.MX RT1060
- i.MX RT1064
- i.MX RT1160
- i.MX RT1170

For i.MX RT685 there is a separate lab document.

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 RT1170](#). Make sure to include the “eIQ” middleware option.
5. Unzip the RT1170 SDK zip file into a directory path that does not contain any spaces.

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 they can be very 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](#).

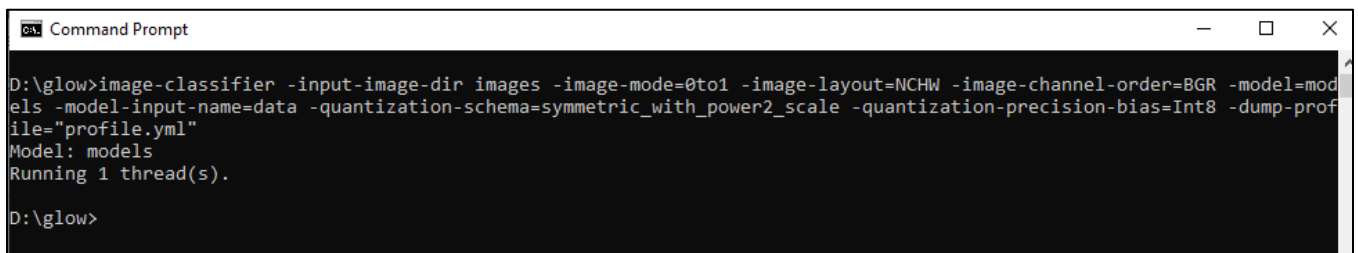
4 LeNet MNIST Example

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=mod
els -model-input-name=data -quantization-schema=symmetric_with_power2_scale -quantization-precision-bias=Int8 -dump-prof
ile="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 TFLite and 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

- **-load-profile="profile.yml"**

Load the profile file generated from the previous step

- **-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 1 channel 28x28 image.
- For ONNX models, this argument is not needed as this information can be read directly from the model.

- **-dump-tuned-profile="profile_tuned.yml"**

Filename to store the highly tuned profile results

- **-backend=CPU**

Use CPU as the backend

- **-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.

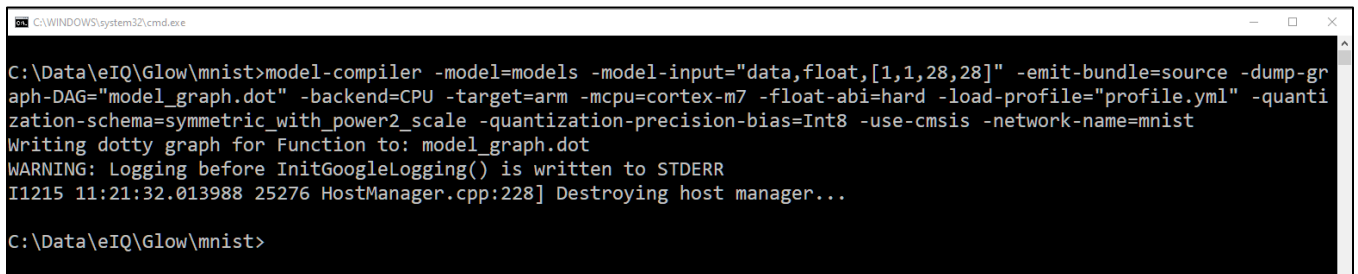
- **-target-accuracy="0.95"**

Stop tuning when accuracy has reached 95%. This saves significant amounts of time.

- The profile file created from the **image-classifier** tool 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-m7
-float-abi=hard -load-profile="profile.yml"
-quantization-schema=symmetric_with_power2_scale -quantization-precision-bias=Int8
-use-cmsis -network-name=mnist
```



```
C:\Data\elQ\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-m7 -float-abi=hard -load-profile="profile.yml" -quanti
zation-schema=symmetric_with_power2_scale -quantization-precision-bias=Int8 -use-cmsis -network-name=mnist
Writing doty graph for Function to: model_graph.dot
WARNING: Logging before InitGoogleLogging() is written to STDERR
I1215 11:21:32.013988 25276 HostManager.cpp:228] Destroying host manager...

C:\Data\elQ\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.

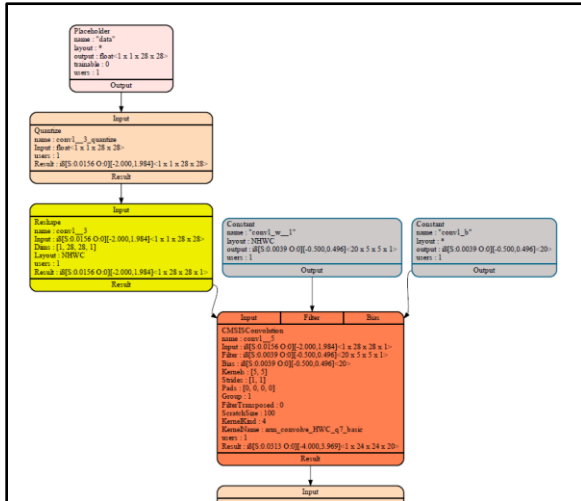
- Here's an explanation for the arguments in the command you just ran. You can also use "model-compiler -help":
 - model=models**
 - For Caffe2 models, directory name that contains the Caffe2 model files named `init_net.pb` and `predict_net.pb` that will be compiled by Glow
 - For TFLite and ONNX models, it should be set to the 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 TFLite and ONNX models, this argument 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-m7**
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. Note that if the optional model-tuner profile was generated, would use the 'profile_tuned.yml' file instead.
 - **-quantization-precisison=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/RT1170.
 - **-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.
7. You can see the generated files inside the **source** folder specified 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 CMSIS-NN 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>
```



- If a camera is not available for your board, a static image can be used to test the accuracy of the model when running on an embedded system. To do this, we will convert an image into a C array that will be loaded into Flash and used as input for the model inference on the board. This step is not required if using a board with a camera+LCD.

The `glow_process_image.py` python script can be found in the RT1170 MCUXpresso SDK at `<SDK_dir>\middleware\eiq\glow\examples\common`. This script will generate a `input_image_test.inc` file that will be used in the next section. Notice the generated file is $28 \times 28 \times 4 = 3136$ bytes in size because of the 28×28 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.inc" -image-mode=0to1 -image-layout=NCHW
-image-channel-order=BGR -image-type=float32
```

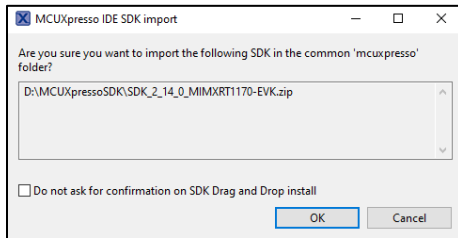
```
Command Prompt
D:\work\eiQ\Glow\mnist>python glow_process_image.py -image-path="images\9_1088.png" -output-path="source\input_image_test.inc" -image-mode=0to1 -image-layout=NCHW -image-channel-order=BGR -image-type=float32
Image processed to file "source\input_image_test.inc" ...
Image size = 3136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

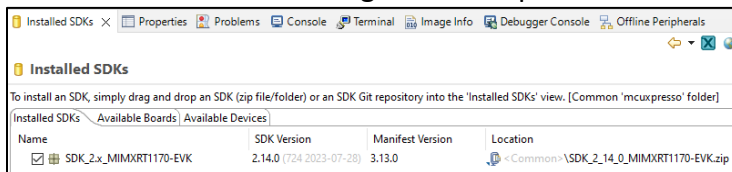
5 Run Glow on the RT1170

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 RT1170 board.

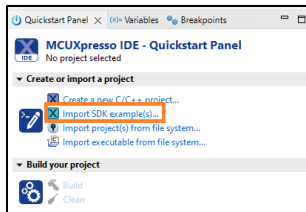
10. Open up MCUXpresso IDE and select a new workspace
11. Install the RT1170 SDK into the “Installed SDKs” tab by dragging-and-dropping the **RT1170 SDK .zip** file downloaded earlier into the Installed SDK window. This dialog box will come up, and click OK to continue the import:



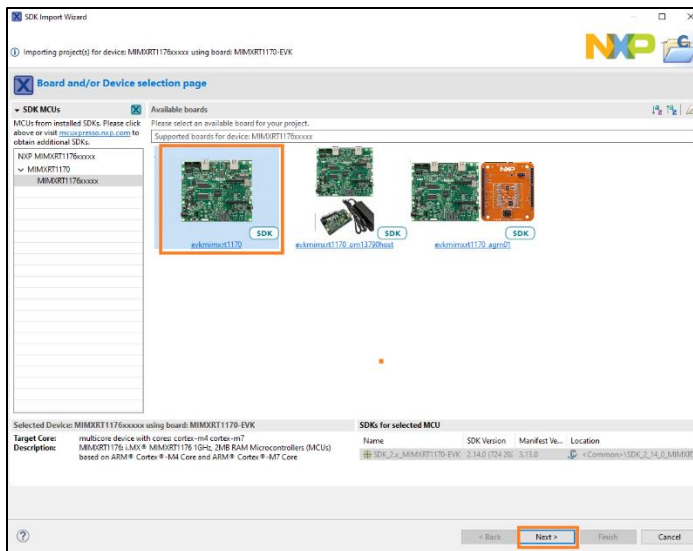
12. It will look like the following when complete:



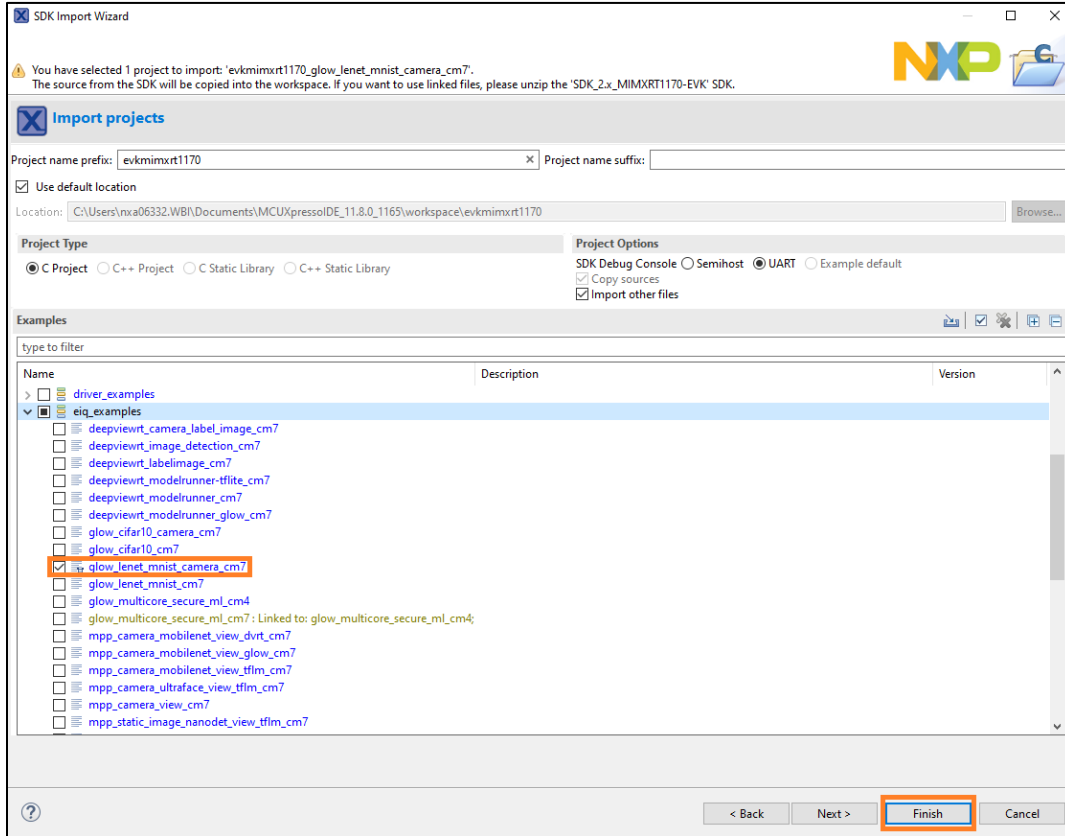
13. In the Quickstart Panel in the lower left corner, click on Import SDK examples(s)...



14. Select the RT1170 board and click on Next



15. Expand the **eiq_examples** category and select the **glow_lenet_mnist_camera_cm7** example. Click on Finish.



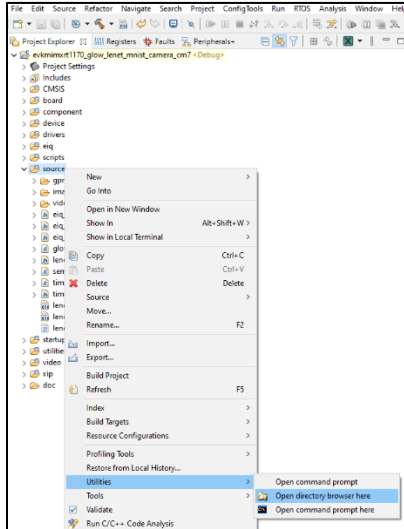
Note: If you do not have a camera+LCD for your board, use the **glow_lenet_mnist_cm7** example instead.

16. 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.inc comes from the python script to generate the data to do the inferencing on.

- mnist.h
- mnist.o
- mnist.weights.txt
- input_image_test.inc

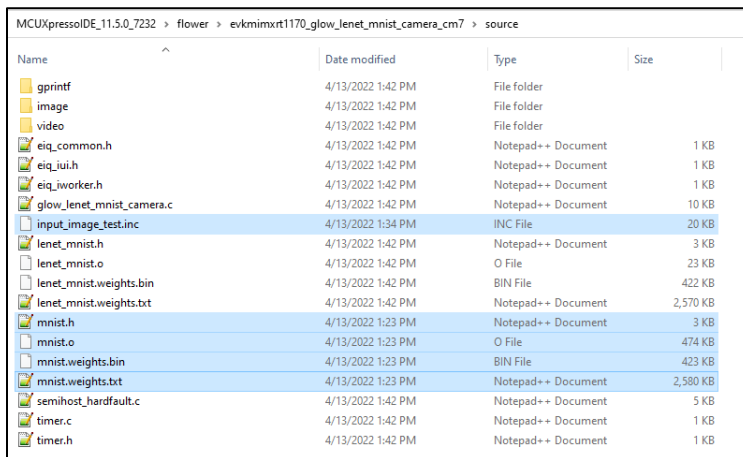
- Open the directory location to place these files by right clicking on the **source** folder in the Project Explorer and selecting **Utilities->Open directory browser here**



- It should open a directory at something like something like: **C:\Users\user_name\Documents\MCUXpressoIDE_11.8.0_1165\workspace\evkmimxrt1170_glow_lenet_mnist_camera_cm7\source**
- Find the **mnist.h**, **mnist.o**, **mnist.weights.bin**, and **mnist.weights.txt** files from the Glow source directory created from the previous section, and copy those files into this directory.

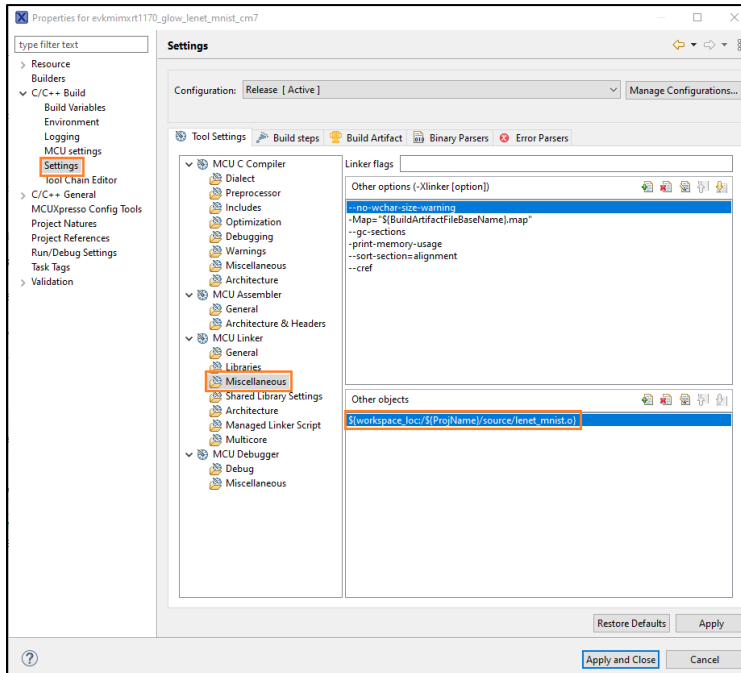
Then find the **input_image_test.inc** file created in the previous section and also copy that file to that same directory.

It will look like the following when done:

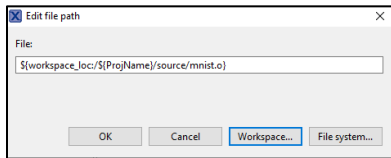


- Now go back to MCUXpresso IDE
- Open up the Project Properties by right clicking on the project name and select Properties.

22. Now go to the C/C++ Build->Settings->MCU Linker->Miscellaneous screen and double click on the item in “Other Objects” to change the object file to the one that was just created:



23. Click on Workspace, navigate to the source folder, and select “mnist.o”. Then hit OK.



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

25. Now there are modifications that need to be made to **glow_lenet_minst_camera.c** to edit the file to use the new file names. If we had chosen to use `-network-name="lenet_minst"` then these changes wouldn't be necessary since the original name and the old name would match up in the source code. But by using a new network name we can walk through the structure of the code and point out changes needed if running a custom model.

Also note that the line numbers will differ based on the SDK package being used and if the camera supported version is being used or not. However the changes themselves will be the same for all projects.

Everywhere in the `glow_lenet_mnist_camera.c` file that `lenet_mnist` is used, it should be changed to just “mnist” and use the new variable names created by the generated files. This includes:

- Line 61 to include the generated header file “mnist.h”
- Lines 65-66 for the new Glow variable names
- Line 67 to include the generated weights file: “mnist.weights.txt”
- Lines 71-76 to use the new Glow variable names
- Line 79 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 `mnist.h` file.
- Line 82 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 `mnist.h` file.
- Line 86-88 should be set to the input image size
- Line 91 should be set to the number of classes of the model.
- Lines 105-114 do not need to be modified but is where the labels can be set
- Line 164 which is what starts the inference by calling “`mnist(constantWeight, mutableWeight, activations)`”

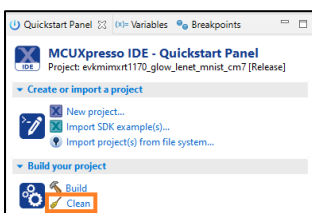
```

59// ----- Bundle API -----
60 // Bundle includes.
61 #include "mnist.h"
62 #include "glow_bundle_utils.h"
63
64 // Statically allocate memory for constant weights (model weights) and initialize.
65 #define MNIST_CONSTANT_MEM_SIZE 1000000
66 uint8_t constantWeight[MNIST_CONSTANT_MEM_SIZE] = {
67 #include "mnist.weights.txt"
68 };
69
70 // Statically allocate memory for mutable weights (model input/output data).
71 #define MNIST_MUTABLE_MEM_SIZE 1000000
72 uint8_t mutableWeight[MNIST_MUTABLE_MEM_SIZE];
73
74 // Statically allocate memory for activations (model intermediate results).
75 #define MNIST_ACTIVATIONS_MEM_SIZE 1000000
76 uint8_t activations[MNIST_ACTIVATIONS_MEM_SIZE];
77
78 // Bundle input data absolute address.
79 uint8_t *inputAddr = GLOW_GET_ADDR(mutableWeight, MNIST_data);
80
81 // Bundle output data absolute address.
82 uint8_t *outputAddr = GLOW_GET_ADDR(mutableWeight, MNIST_softmax);
83
84 // ----- Application -----
85
86 #define IMAGE_WIDTH 28
87 #define IMAGE_HEIGHT 28
88 #define IMAGE_CHANNELS 3
89
90 // Number of output classes for the model.
91 #define MODEL_NUM_OUTPUT_CLASSES 10
  
```

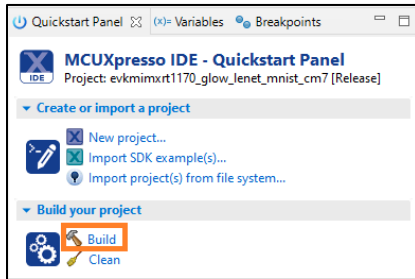
```

163 start = get_time_in_us();
164 mnist(constantWeight, mutableWeight, activations);
165 end = get_time_in_us();
  
```

26. 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 inferencing. This is only required when new Glow files are copied into the project. Click on “Clean” in the Quickstart Panel first:



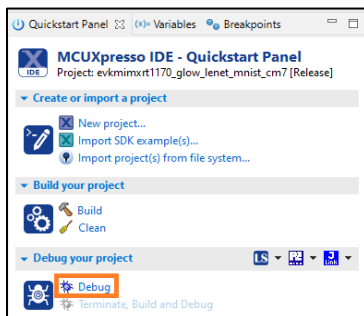
27. Build the project by clicking on “Build” in the Quickstart Panel.



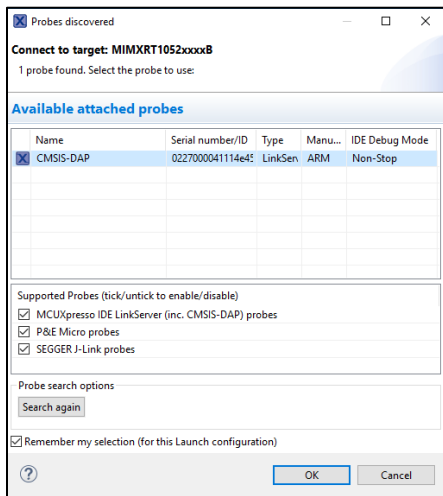
28. Plug the micro-B USB cable into the board at J11 and [connect the camera+LCD](#) if available.

29. 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.

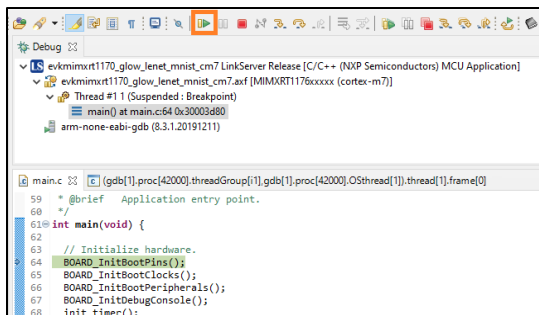
30. Debug the project by clicking on “Debug” in the Quickstart Panel.



31. It will ask what interface to use. Select CMSIS-DAP.



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



33. Draw a number on a piece of white paper and you should see the console output identify the number when you point the camera at it as well as display on the LCD screen. You can also use the PDF attached to this NXP Community post for example numbers.

```
Termin - [disconnected]VT
File Edit Setup Control Window Help
data for inference are ready
last class = 6 (6)
Selfidence = 0.999
Inference time = ? (ms)
data for inference are ready
last class = 6 (6)
Selfidence = 0.999
Inference time = ? (ms)
data for inference are ready
last class = 6 (6)
Selfidence = 0.999
Inference time = ? (ms)
data for inference are ready
last class = 6 (6)
Selfidence = 0.999
Inference time = ? (ms)
data for inference are ready
last class = 6 (6)
Selfidence = 0.999
Inference time = ? (ms)
```

6 Conclusion

This lab demonstrates how to use the Glow tools to create executable models that will run on the RT1170 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 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\evkmimxrt1170\eiq_examples\glow_cifar10\doc` for more details.

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](#).