Glow for NXP MCUs

DL Compiler Study
2021-07-15

Overview

NXP’s Edge ML Framework - eIQ

Glow for eIQ

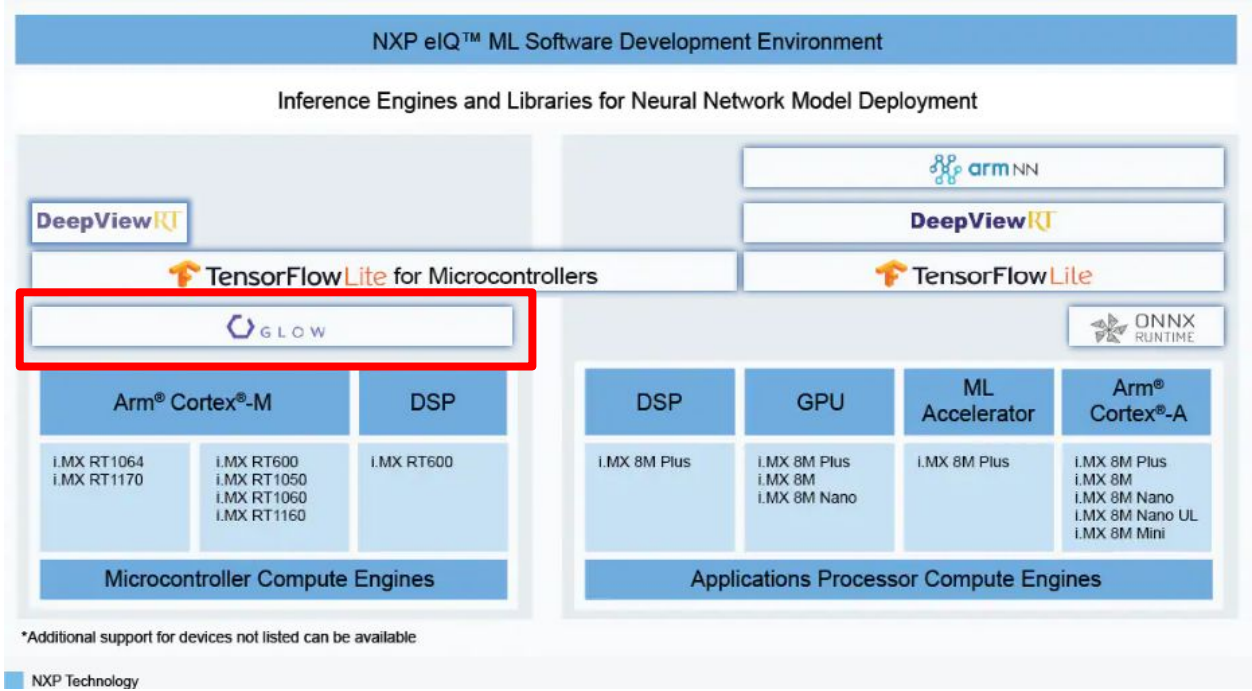
Tutorial

Future Works (Ideas?)

Objectives:

- To provide a jump-start tutorial for further “hacks” (later dive deep into Glow src).
- To share a viewpoint from the application point of view (embedded systems).

NXP eIQ™ Line-Up (as of July 2021)



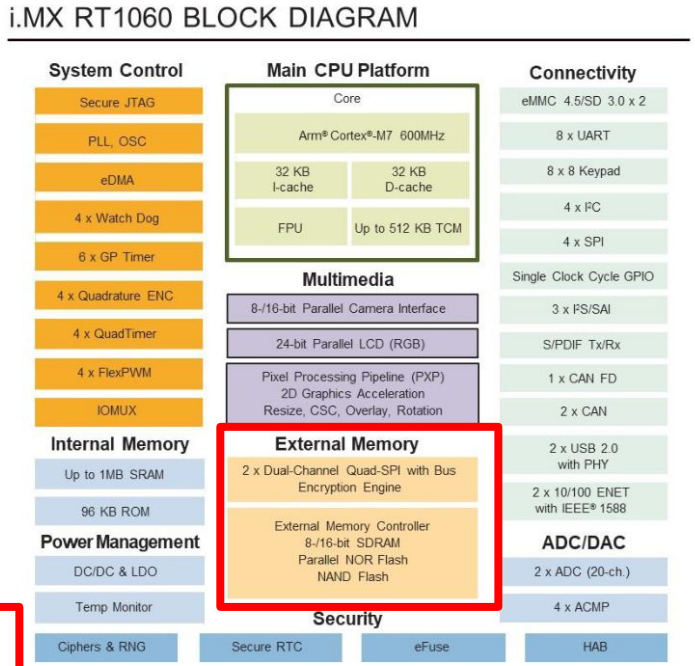
NXP offers the most mature ML framework for edge devices

Which side can offer the best ML Compiler experience -- Software company or Hardware manufacturer?

Hardware Overview

I.MX RT1060 MCUS

- Up to 600 MHz Arm Cortex-M7 core
- Up to 1 MB of SRAM
- HS GPIO
- CAN FD
- Synchronous parallel NAND/NOR/PSRAM controller
- 8-/16-bit Parallel camera interface



+ Flagship Processor for eIQ
+ 1 Ghz (!!) MCU also available (RT1170)
- No Accelerator

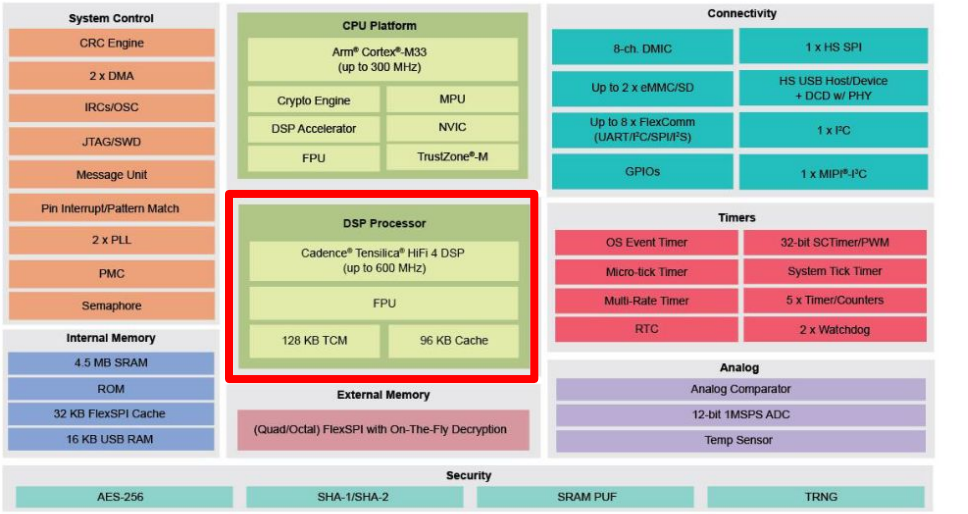
Hardware Overview

I.MX RT600 MCUS

- Up to 300 MHz Arm Cortex-M33 core
- Up to 600 MHz Cadence® Tensilica® HiFi 4 DSP
- 4.5 MB of SRAM
- Audio and sensor processing

NXP
i.MX RT685

i.MX RT600 CROSSOVER MCU FAMILY BLOCK DIAGRAM



+ **Fastest Cortex-M33 Processor (ARMv8-M) in Market (300 MHz) !!**
- **FOWLP**
- **Only Commercial Temperature Range (-20 to 70)**

eIQ™ ML Software Development Environment (WIP)

NEW

Machine Learning Workflow Tools

eIQ Toolkit NEW

- Enables graph-level profiling capability with runtime insights to help optimize neural network architectures
- Eases ML development with eIQ Portal and command line host tools

eIQ Portal NEW

- Intuitive graphical user interface (GUI) that simplifies ML development
- Creates, optimizes, debugs, converts and exports ML models

Supported ML Inference Engines

DeepViewRT™ NEW

- Proprietary inference engine for i.MX RT crossover MCUs and i.MX 8 series applications processors

TensorFlow™ Lite Micro

- Faster and smaller inference engine than the traditional TensorFlow Lite
- Optimized for running machine learning models i.MX RT crossover MCUs

TensorFlow™ Lite

- Flexible inference engine for embedded applications
- Targeting Arm® Cortex®-A, and Cortex-M processor cores and Verisilicon GPUs and NPUs

Arm® NN

- Inference engine framework for i.MX and Layerscape® processors
- Leveraging the Arm Compute Library for optimized NN layer support

CMSIS-NN

- Kernels to create a static, performance- and memory-optimized inference engine for Cortex-M processor cores

Glow

- Neural Network compiler targeted at i.MX RT Crossover MCUs
- NXP optimized library integration to deliver high-performance, memory-efficient inferencing

Why Glow (for eIQ)?

WHAT IS THE GLOW NN COMPILER?

Glow

An open source, community project released by Facebook in 2018 (Apache License 2.0)

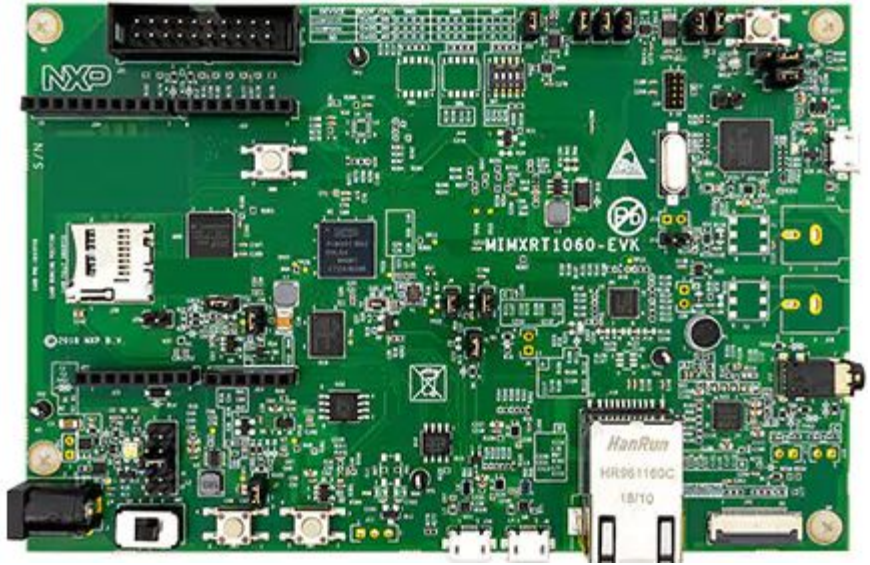
Meeting the needs of both ends of the computing spectrum
- Applicable for low-cost MCU environments using Ahead-of-Time compilation model (AOT)
- Applicable for running huge NN models in cloud-based environment using Glow runtime

Generates machine code by compiling NN layers optimized for a specific MCU target
- Parses graph at compile-time and the compiled library contains only the required optimized code

Why NXP chose Glow
- There are other choices including TVM, MLIR, ELL
- Glow was (and is) the most mature

Glow for eIQ: Hands-on Tutorial

Hardware

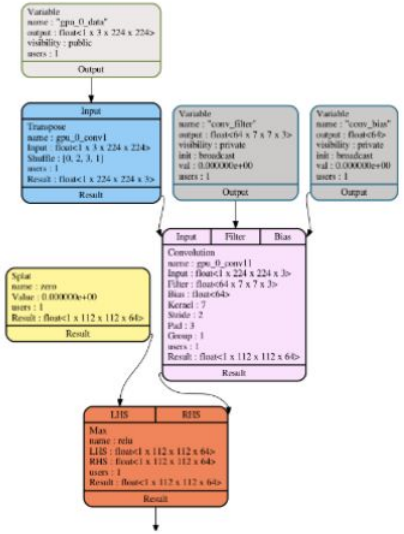


i.MX RT1060 EVK

Arm® Cortex®-M7 core (600 MHz)

GRAPH LOWERING TO MACHINE CODE WITH THE GLOW COMPILER

Glow




```
declare {
 %input = weight float<8 x 28 x 28 x 1>, broadcast, 0.0
 %filter = weight float<16 x 5 x 5 x 1>, xavier, 25.0
 %filter0 = weight float<16>, broadcast, 0.100
 %weights = weight float<10 x 144>, xavier, 144.0
 %bias = weight float<10>, broadcast, 0.100
 %selected = weight index<8 x 1>
 ...
 %result = weight float<8 x 10>
}

program {
 %allo = alloc float<8 x 28 x 28 x 16>
 %conv = convolution [5 1 2 16] @out %allo, @in %input,
     @in %filter3, @in %bias0
 %allo0 = alloc float<8 x 28 x 28 x 16>
 %relu = max0 @out %allo0, @in %allo
 %allo1 = alloc index<8 x 9 x 9 x 16 x 2>
 %allo2 = alloc float<8 x 9 x 9 x 16>
 %pool = pool max [3 3 0] @out %allo2, @in %allo0,
     @inout %allo1
 ...
 %deal6 = dealloc @out %allo6
 %deal7 = dealloc @out %allo7
 %deal8 = dealloc @out %allo8
 %deal9 = dealloc @out %allo9
}
```

```
LBB14_1:
  vmovaps 3211264(%rcx,%rax,4), %ymm1
  vmovaps 3211296(%rcx,%rax,4), %ymm2
  vmovaps 3211328(%rcx,%rax,4), %ymm3
  vaddps 6422528(%rcx,%rax,4), %ymm1, %ymm1
  vaddps 6422560(%rcx,%rax,4), %ymm2, %ymm2
  vmovaps 3211360(%rcx,%rax,4), %ymm4
  vaddps 6422592(%rcx,%rax,4), %ymm3, %ymm3
  vaddps 6422624(%rcx,%rax,4), %ymm4, %ymm4
  vmaxps %ymm0, %ymm1, %ymm1
  vmaxps %ymm0, %ymm2, %ymm2
  vmaxps %ymm0, %ymm3, %ymm3
  vmovaps %ymm1, 6422528(%rcx,%rax,4)
  vmovaps %ymm2, 6422560(%rcx,%rax,4)
  vmaxps %ymm0, %ymm4, %ymm1
  vmovaps %ymm3, 6422592(%rcx,%rax,4)
  vmovaps %ymm1, 6422624(%rcx,%rax,4)
  addq $32, %rax
```

HIGH-LEVEL GRAPH

MobileNet, CIFAR-10, Resnet, DeepSpeech

LOW-LEVEL IR

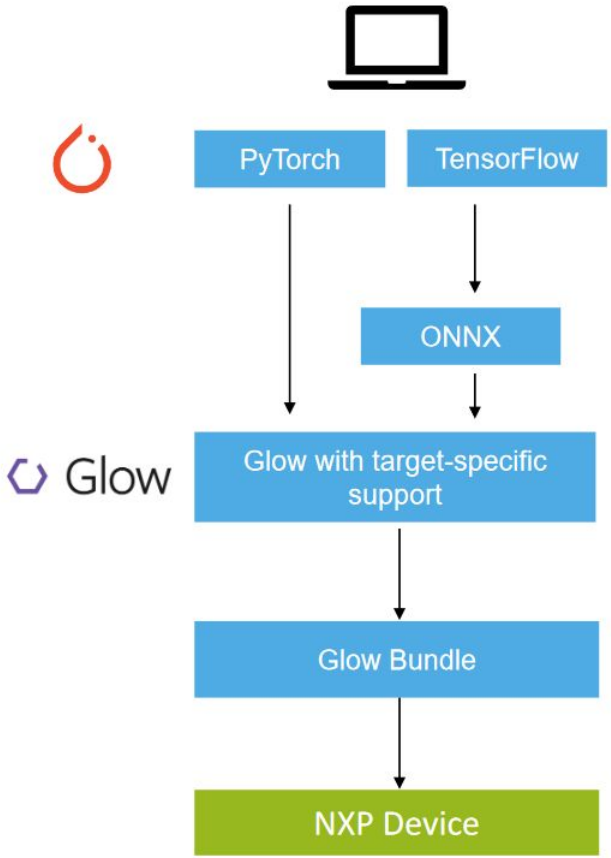
Standard compiler techniques such as kernel fusion and lowering of complex operations to simple kernels, and others

MACHINE CODE

Out-of-the-box, Glow generates target-agnostic, native machine code
Can apply target-specific optimizations and generate object file for integration into application project

NXP

METHODS FOR DEPLOYING GLOW



PyTorch is an open source ML framework that accelerates the path from research prototyping to production deployment

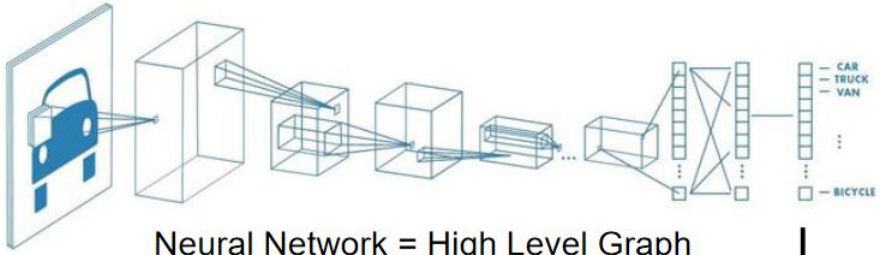
- PyTorch development framework can directly access Glow
- Operates on ONNX model (open neural network exchange)
- NXP upstreamed ability to bring in TensorFlow Lite models
  - Glow will directly load/import standard TensorFlowLite flatbuffer model format
- NXP integrated optimized back-end support for Arm® Cortex®-M (CMSIS-NN) cores and Cadence® Tensilica® HiFi 4 DSP
  - CMSIS-NN implements common model layers such as convolution, fully-connected, pooling, activation, etc, efficiently at a low level

Glow bundle is a self-contained compiled network model used to execute the model in a standalone mode

Delivery in MCUXpresso SDK resolves platform dependencies

- Bundle represented as object file is directly compiled by toolchain along with other application object files

TRADITIONAL VS. GLOW NN MODEL PROCESSING



Neural Network = High Level Graph

Offline (on host machine)

1. Glow compiler optimizes graph into intermediate representation
2. Glow compiler links in NXP target-specific backends
   a) Arm CMSIS-NN for Cortex M
   b) NN Library for Tensilica HiFi 4 DSP

On Target

1. TensorFlow Lite runtime performs dynamic interpretation of NN
2. CPU core performs runtime
3. CPU core performs NN operator computation

NXP i.MX RT1060

Performance and memory overhead

1. Executed as a sequence of computations - like other application code

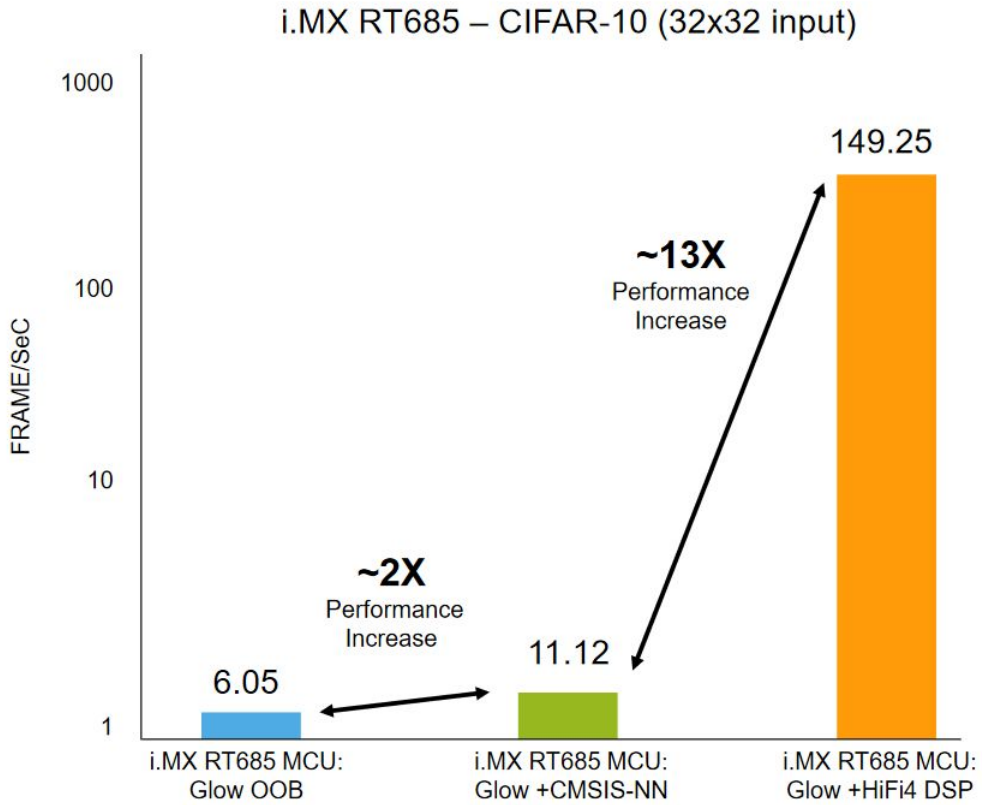
NXP i.MX RT1060

No runtime interpretation required

WHY NXP DID A CUSTOMIZED IMPLEMENTATION OF GLOW

Glow out of the box (OOB) is good for general deployment and testing but doesn't tap into acceleration libraries

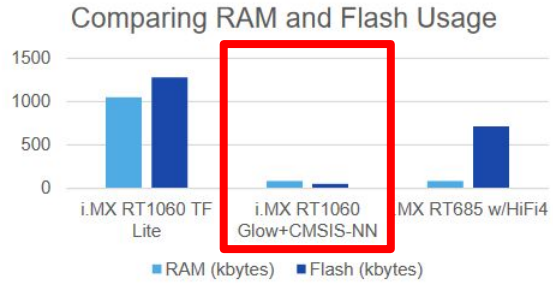
NXP integrated target-specific libraries for Cortex M and HiFi4 DSP providing the industry's first MCU-based implementation of Glow neural network compiler for machine learning at the edge

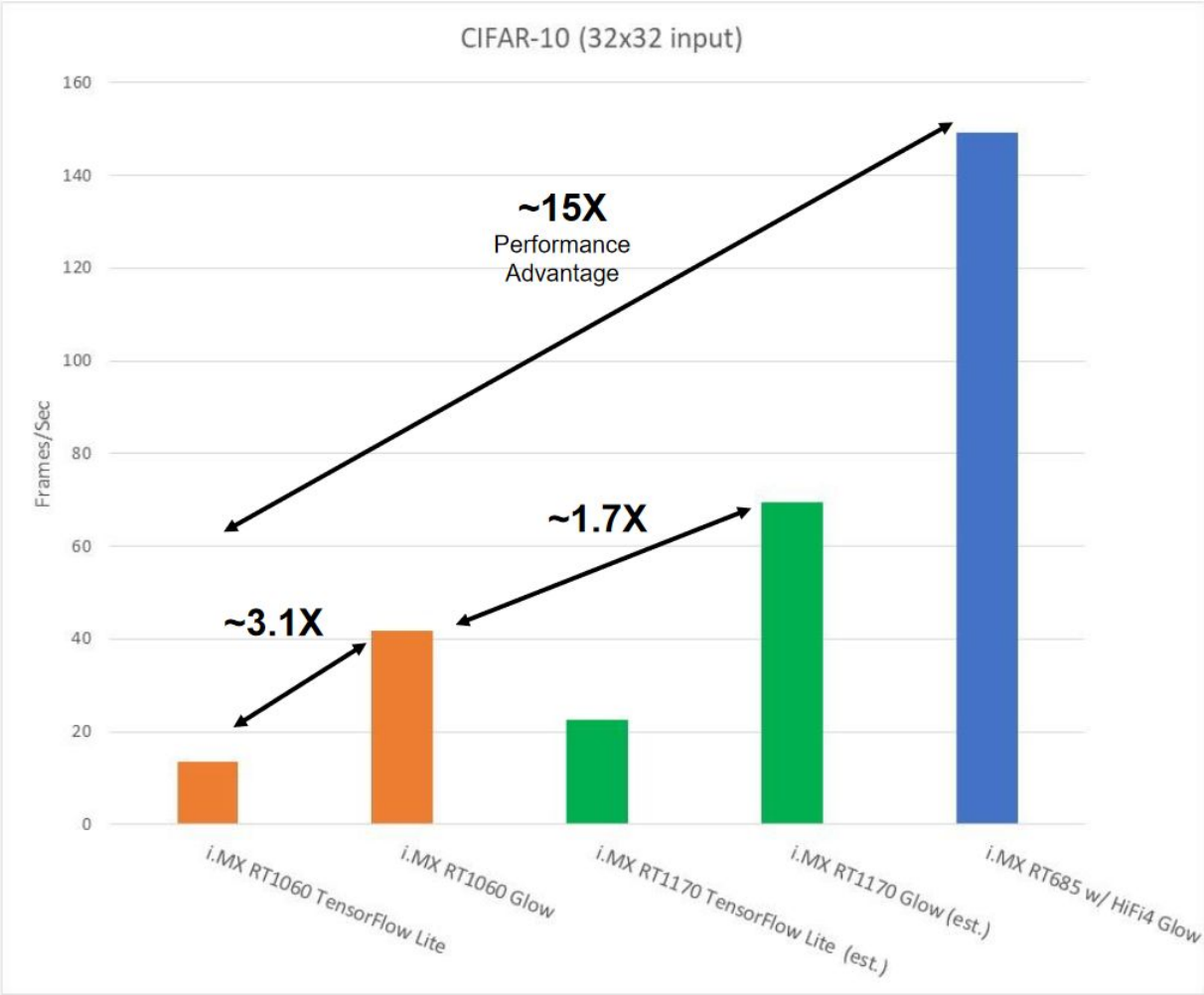


Source NXP Systems Engineering Testing
Performance differences are neural network model dependent

BENEFITS OF ADDING eIQ SUPPORT FOR GLOW

- NXP tested devices:
  - i.MX RT1060 @600MHz Arm Cortex M7
  - i.MX RT1170 @1000MHz Arm Cortex M7
  - i.MX RT685 @300 Arm Cortex M33 + 600MHz Tensilica HiFi4

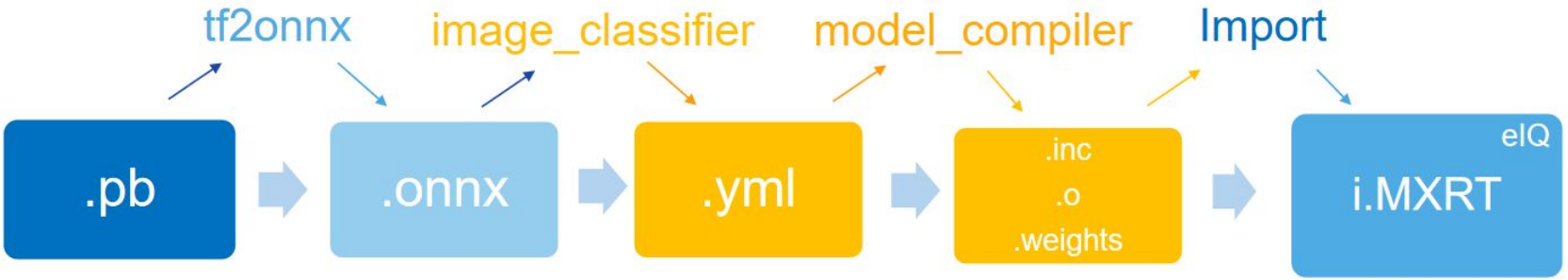




- Source NXP Systems Engineering Testing
- Glow support include CMSIS-NN and HiFi4 NN libraries
- Performance differences are neural network model dependent

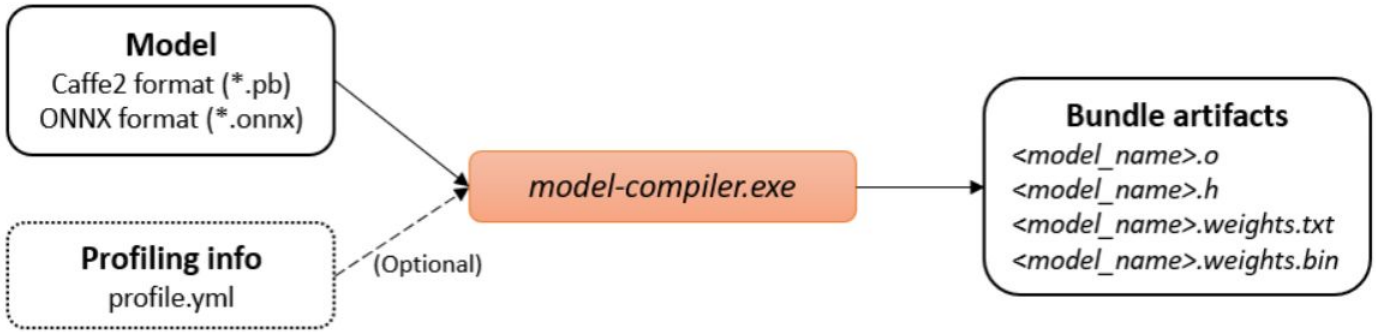
GLOW DEPLOYMENT FLOW

1. Transform model to the universal ONNX format.
2. Optimize model with profiler to create profile.yml file
3. Compile with Glow model_compiler to generate compiled files and weights.
4. Copy binary files into eIQ Glow SDK example.



GLOW COMPILER

- Glow model-compiler tool generates the compiled code that will be executed
- **-use-hifi** compile option will dispatch supported operations to use HiFi DSP core (RT685 only)
- **-use-cmsis** compile option will dispatch supported operations to use CMSIS-NN library.
  - The model must also be quantized as part of compilation in order to make use of the CMSIS-NN optimizations



Generates 4 files:

- <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.

KEY GLOW CODE

- Set input data buffer

```
// Load input data
memcpy(bundleInpAddr, ((char *)INPUT_DATA_START) + idx * INPUT_IMAGE_SIZE, INPUT_IMAGE_SIZE);
```

- Run model

```
78   // Perform inference and compute inference time.
79   start_time = get_time_in_us();
80   mnist(constantWeight, mutableWeight, activations);
81   stop_time = get_time_in_us();
82   duration_ms = (stop_time - start_time) / 1000;
```

- Get result from output buffer

```
// Get classification top1 result and confidence
float *out_data  = (float *)(bundleOutAddr);
float max_val    = 0.0;
uint32_t max_idx = 0;
for (int i = 0; i < OUTPUT_NUM_CLASS; i++)
{
    if (out_data[i] > max_val)
    {
        max_val = out_data[i];
        max_idx = i;
    }
}
```

GLOW MEMORY USAGE

- Glow does not use dynamically allocated memory (heap)

!! Hard Real-Time Applications!!

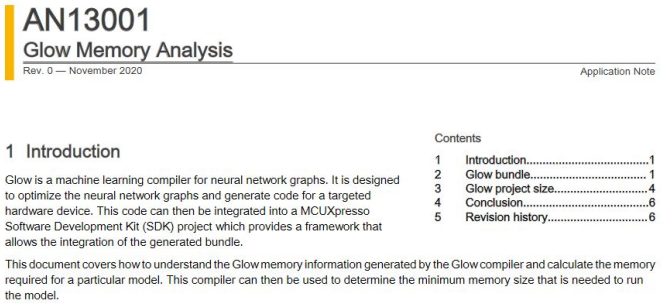
- All memory requirements of a compiled model are in the auto-generated header file

```
// Memory sizes (bytes).
#define CIFAR10_CONSTANT_MEM_SIZE    34176    // Stores model weights. Can be stored in Flash or RAM.
#define CIFAR10_MUTABLE_MEM_SIZE     12352    // Stores model inputs/outputs. Must be in RAM.
#define CIFAR10_ACTIVATIONS_MEM_SIZE 71680    // Store model scratch memory required for intermediate computations. Must be in RAM.
```

- Inference usually faster if weights are in RAM.

- Memory usage application note being developed

AN13001: Glow Memory Analysis
https://www.nxp.com/docs/en/application-note/AN13001.pdf



AN13001
Glow Memory Analysis
Rev. 0 — November 2020 Application Note

Contents

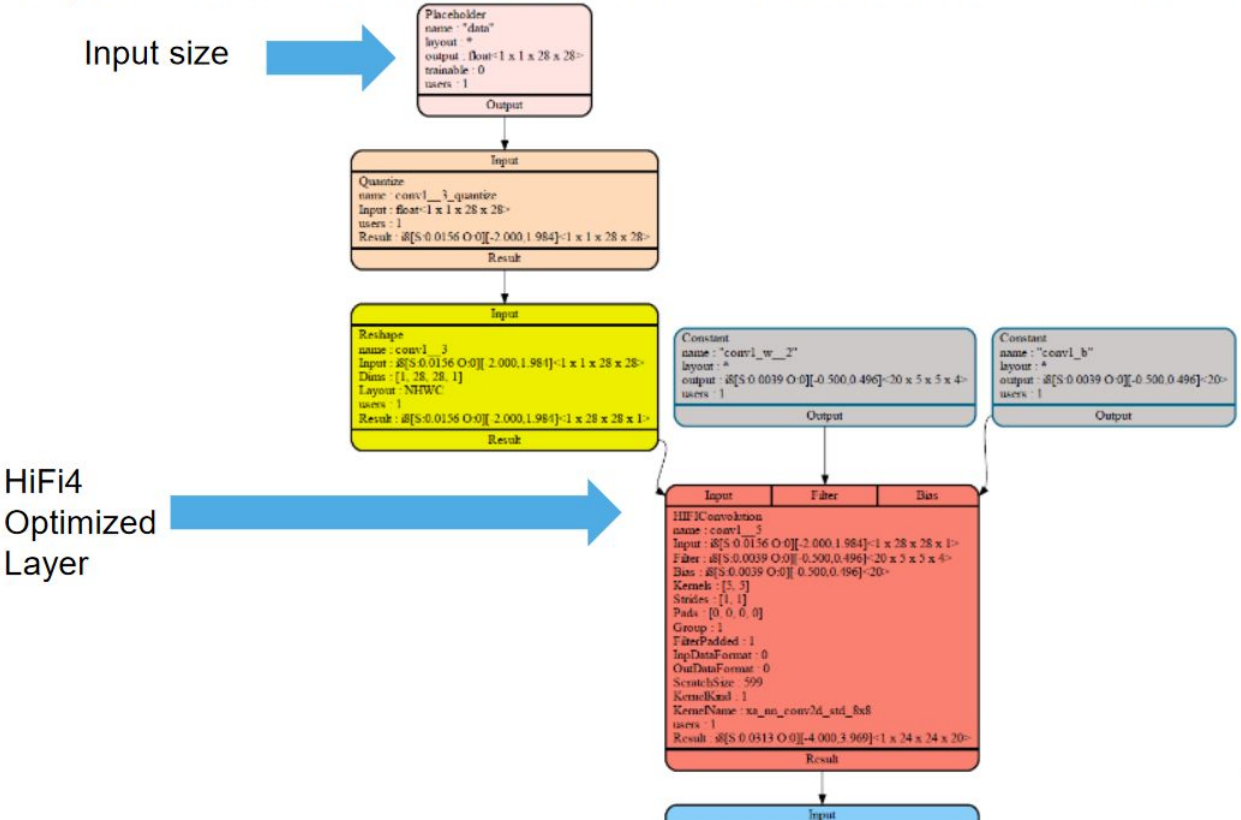
1 Introduction

Glow is a machine learning compiler for neural network graphs. It is designed to optimize the neural network graphs and generate code for a targeted hardware device. This code can then be integrated into a MCUXpresso Software Development Kit (SDK) project which provides a framework that allows the integration of the generated bundle.

This document covers how to understand the Glow memory information generated by the Glow compiler and calculate the memory required for a particular model. This compiler can then be used to determine the minimum memory size that is needed to run the model.

GLOW OUTPUT GRAPH

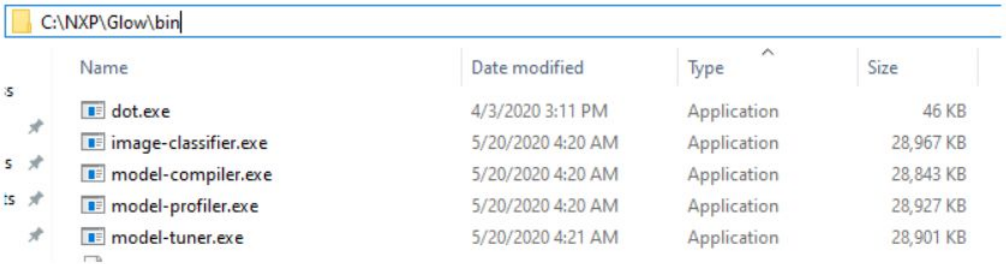
- Can use ***-dump-graph-DAG*** Glow compiler option to create graph of model
- Shows which layers/nodes use CMSIS-NN or HiFi4 optimizations, as well as input information



GLOW RELEASE

- Glow SDK projects released as part of MCUXpresso SDK

- Glow quantization and compiler tools released as separate installer on NXP.com
  - **image-classifier.exe**, **model-profiler.exe**, and **model-tuner.exe** used for quantization
  - **model-compiler.exe** is Glow compiler
  - **dot.exe** is used to convert graph file in .dot format to PDF
  - Glow installer default location is **C:\NXP\Glow**
    - \bin directory will be added to Windows executable path
    - \doc directory contains Getting Started guide and User Guide



Step-by-step instructions

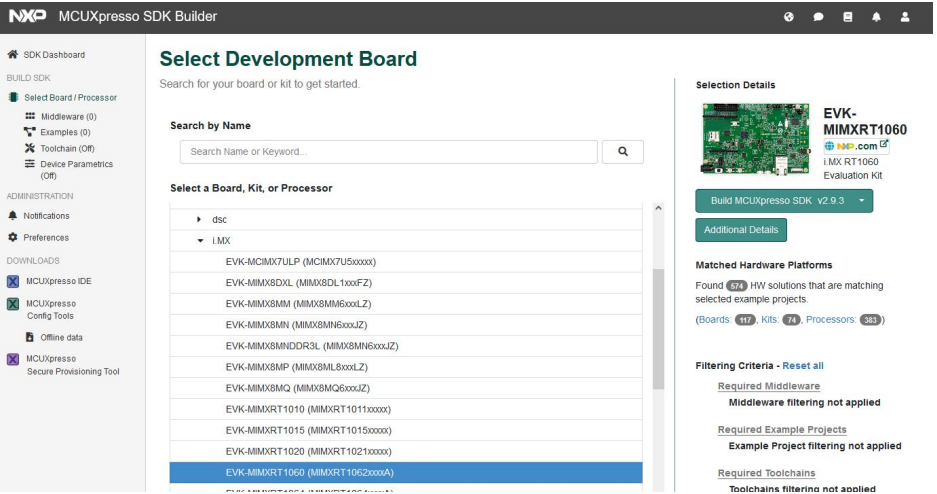
Versions Used

	Demo	Now
Python	3.7.4	3.9.5
Tensorflow	2.4	2.5.0
IDE	11.1.1_3241	11.3.1_5262
SDK	2.8.0	2.9.3

SDK Builder

https://mcuxpresso.nxp.com/

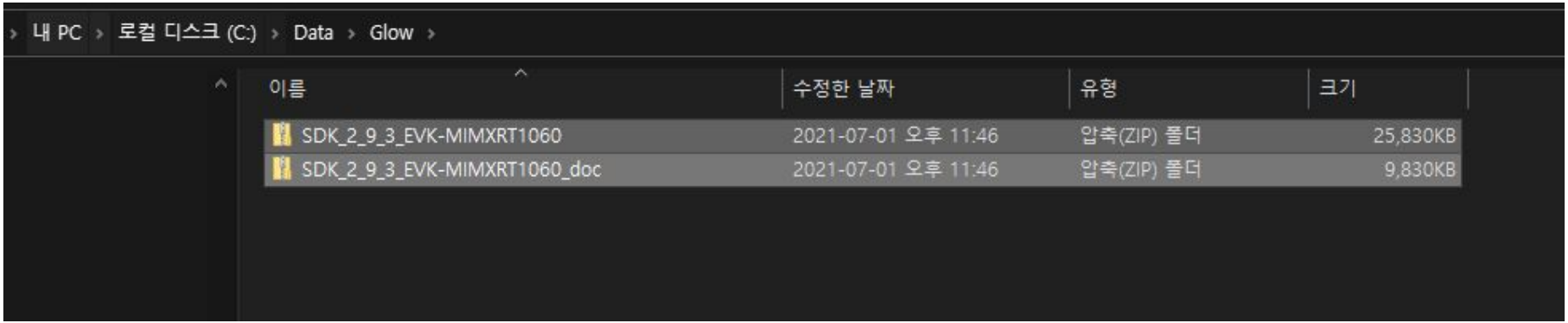
- Registration required
- Off-line config is also available



	Name	Category	Description	Dependencies
☑	SDMMC Stack	Middleware	Stack supporting SD, MMC, SDIO	
☐	CANopen	Middleware	MicroCANopen Stack from Embedded Solutions Academy	
☑	CMSIS DSP Library	CMSIS DSP Lib	CMSIS DSP Software Library	
☑	eIQ	Middleware	eIQ machine learning SDK containing: - ARM CMSIS-NN library ... (more)	CMSIS DSP Library
☐	Embedded Wizard GUI	Middleware	Embedded Wizard GUI from TARA Systems	
☐	Azure RTOS		Azure RTOS	

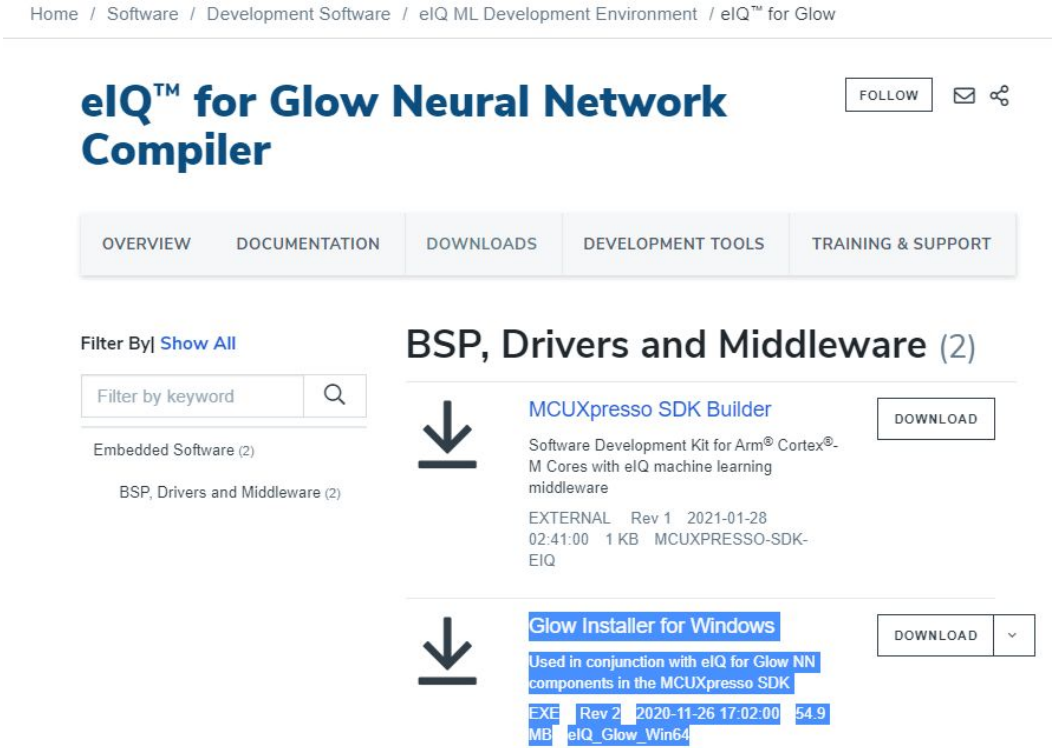
SDK Builder

Unzip SDK to `C:\Data\Glow`



Download & Install Glow Compiler

https://www.nxp.com/design/software/development-software/eiq-ml-development-environment/eiq-for-glow-neural-network-compiler:eIQ-Glow?tab=Design_Tools_Tab

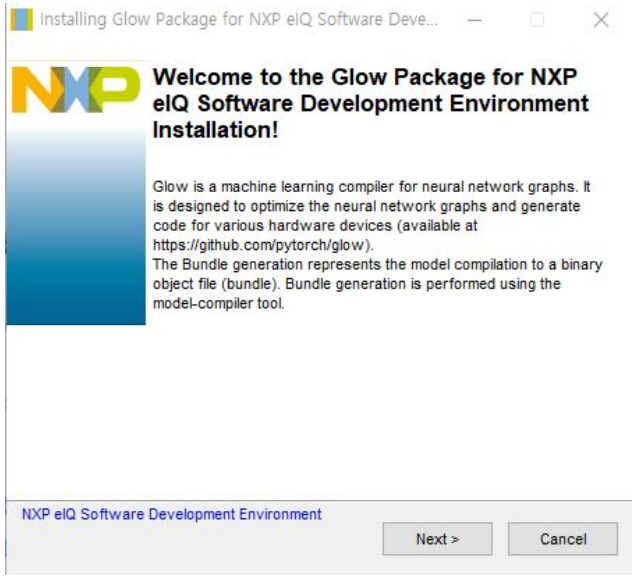


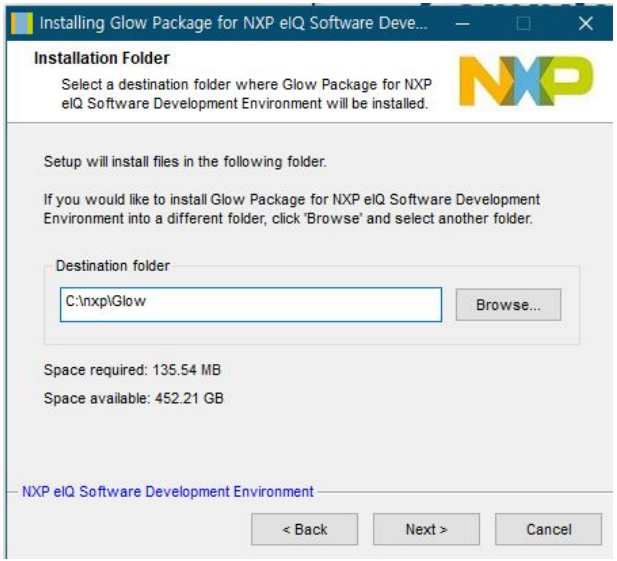
Download & Install Glow Compiler

Add path C:\nxp\Glow to executable

Only Available for Windows PC

Seems like still in the early stage of development





ML Installation

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 3.8. ****The 64-bit edition is required and Python 3.9 is not currently supported by TensorFlow, so for this lab it is highly recommended to use 64-bit 3.8.x****:
   https://www.python.org/downloads/
2. Open a Windows command prompt and verify that the python command corresponds to Python 3.x. You may need to use "python3" for all the commands instead of "python" or add Python to your Windows path:
   python -V
3. Still in the Windows command prompt, 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 tensorflow==2.4
   python -m pip install onnxmltools keras mmdnn tensorflow-datasets opencv-python
   python -m pip install numpy scipy matplotlib ipython jupyter pandas sympy nose imageio
   python -m pip install netron PILLOW

Tensorflow 2.5 and

Python 3.9.5

work just fine

LeNet MNIST Example

4 LeNet MNIST Example

The following steps can be used to run the LeNet MNIST example which does hand-written digit recognition. The steps below are based on the eIQ Glow User Guide and more details can be found in that document.

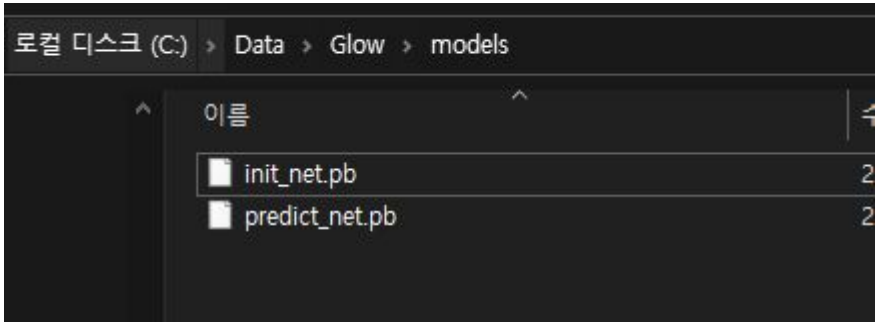
1. Create a new empty directory on your filesystem. Make sure there are no spaces in the path.
2. Download the Caffe2 MNIST models and put into a subdirectory named "**models**"
   - http://fb-glow-assets.s3.amazonaws.com/models/lenet_mnist/predict_net.pb
   - http://fb-glow-assets.s3.amazonaws.com/models/lenet_mnist/init_net.pb
3. Download some example MNIST images and put them in a subdirectory named "**images**"
   - https://github.com/pytorch/glow/tree/master/tests/images/mnist
4. 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.

LeNet MNIST Example

Caffe2 model: models/*.pb

Under C:\Data\Glow

Create mnist\models, put init_net.pb, predict_net.pb

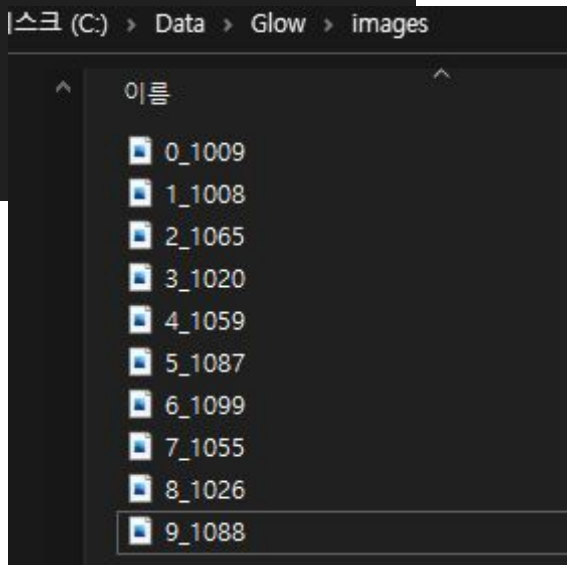


Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	12	FC	13	12	07	63	6F	6E	76	31	5F	77	1A	00	22	0F	.ü...conv1_w..".
00000010	47	69	76	65	6E	54	65	6E	73	6F	72	46	69	6C	6C	2A	GivenTensorFill*
00000020	0F	0A	05	73	68	61	70	65	30	14	30	01	30	05	30	05	...shape0.0.0.0.
00000030	2A	CC	13	0A	06	76	61	6C	75	65	73	2D	70	B8	97	BE	*Ì...values-p¸—¾
00000040	2D	EE	C5	BF	3E	2D	4E	BF	9A	BE	2D	2F	29	BB	3C	2D	-îÅ¿>-N¿š¾-/)»<-
00000050	2A	83	87	BE	2D	57	53	87	3E	2D	98	E7	64	3E	2D	27	*ƒ‡¾-WS‡>-˜çd>-'
00000060	3E	5A	3E	2D	75	E4	FF	3D	2D	E7	30	28	3E	2D	60	80	>Z>-uäÿ=-ç0(>-`€
00000070	B8	3C	2D	9E	F7	D1	3E	2D	7C	9B	2A	3E	2D	DD	F2	1B	¸<-ž÷Ñ>-|›*>-Ýò.
00000080	3E	2D	A3	84	8A	BD	2D	91	14	C2	3E	2D	4B	B5	80	3D	>-£„Š½-'.Â>-Kµ€=
00000090	2D	D1	FF	D1	3E	2D	54	ED	49	3D	2D	D2	CB	AA	3E	2D	-ÑÿÑ>-TíI=-ÒË*>-
000000A0	47	68	A3	BE	2D	F6	37	8E	3E	2D	73	BC	87	BE	2D	E0	Gh£¾-ö7Ž>-s¼‡¾-à
000000B0	DF	8D	BE	2D	42	19	03	3E	2D	21	F0	A4	BE	2D	36	B4	ß.¾-B..>-!ð¤¾-6´
000000C0	95	3D	2D	1C	8A	86	3E	2D	EE	8C	3E	BE	2D	68	77	32	•=-.Š†>-îŒ>¾-hw2
000000D0	BD	2D	6B	DA	A3	BD	2D	EA	D1	0A	3E	2D	51	34	12	BE	½-kÚ£½-êÑ.>-Q4.¾
000000E0	2D	D2	90	75	3C	2D	C3	81	3B	BE	2D	2D	03	7C	BC	2D	-Ò.u<-Ã.;¾--.|¼-
000000F0	24	66	17	BE	2D	3E	99	67	BE	2D	66	FE	C3	BD	2D	4A	$f.¾->™g¾-fþÃ½-J
00000100	AD	94	BE	2D	92	BA	FC	3D	2D	03	9F	05	BE	2D	51	9C	.”¾-’ºü=-.Ÿ.¾-Qœ
00000110	6F	3D	2D	94	58	B6	3D	2D	DE	7B	B9	3E	2D	8B	63	A2	o=-”X¶=-Þ{¹>-‹c¢
00000120	3E	2D	D2	1C	96	BE	2D	E5	49	81	3E	2D	AA	97	CC	3E	>-Ò.–¾-åI.>-ª—Ì>
00000130	2D	E2	2C	0D	BD	2D	24	A4	F2	BD	2D	EB	1C	A8	3E	2D	-â,.½-$¤ò½-ë.¨>-
00000140	40	C6	D4	3E	2D	CE	8A	EB	3E	2D	AD	C8	D5	3E	2D	F7	@ÆÔ>-ÎŠë>-.ÈÕ>-÷
00000150	C9	31	BE	2D	09	73	A2	3E	2D	6E	B3	D5	3E	2D	FC	53	É1¾-.s¢>-n³Õ>-üS
00000160	53	3D	2D	A0	BF	32	3E	2D	FF	53	D4	3D	2D	76	95	90	S=- ¿2>-ÿSÔ=-v•.
00000170	3E	2D	95	5C	AB	BD	2D	36	B2	02	BE	2D	54	55	8D	BE	>-•\«½-6².¾-TU.¾

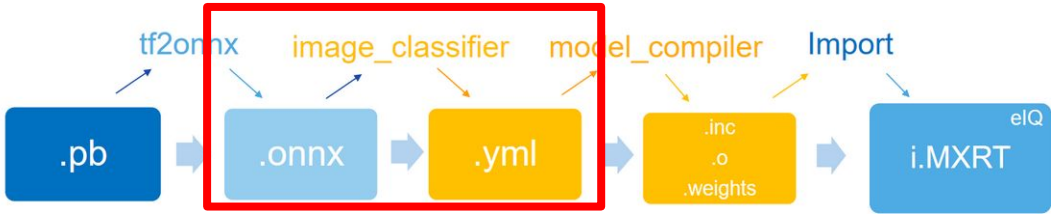
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	0A	0C	6D	6E	69	73	74	5F	64	65	70	6C	6F	79	12	42	..mnist_deploy.B
00000010	0A	04	64	61	74	61	0A	07	63	6F	6E	76	31	5F	77	0A	..data..convl_w.
00000020	07	63	6F	6E	76	31	5F	62	12	05	63	6F	6E	76	31	1A	.convl_b..convl.
00000030	00	22	04	43	6F	6E	76	2A	0A	0A	06	6B	65	72	6E	65	.".Conv*...kerne
00000040	6C	18	05	2A	0D	0A	05	6F	72	64	65	72	22	04	4E	43	l..*...order".NC
00000050	48	57	12	40	0A	05	63	6F	6E	76	31	12	05	70	6F	6F	HW.@..convl..poo
00000060	6C	31	1A	00	22	07	4D	61	78	50	6F	6F	6C	2A	0A	0A	l1..".MaxPool*..
00000070	06	6B	65	72	6E	65	6C	18	02	2A	0D	0A	09	75	73	65	.kernel..*...use
00000080	5F	63	75	64	6E	6E	18	00	2A	0A	0A	06	73	74	72	69	_cudnn..*...stri
00000090	64	65	18	02	12	43	0A	05	70	6F	6F	6C	31	0A	07	63	de...C..pool1..c
000000A0	6F	6E	76	32	5F	77	0A	07	63	6F	6E	76	32	5F	62	12	onv2_w..conv2_b.
000000B0	05	63	6F	6E	76	32	1A	00	22	04	43	6F	6E	76	2A	0A	.conv2..".Conv*.
000000C0	0A	06	6B	65	72	6E	65	6C	18	05	2A	0D	0A	05	6F	72	..kernel..*...or
000000D0	64	65	72	22	04	4E	43	48	57	12	40	0A	05	63	6F	6E	der".NCHW.@..con
000000E0	76	32	12	05	70	6F	6F	6C	32	1A	00	22	07	4D	61	78	v2..pool2..".Max
000000F0	50	6F	6F	6C	2A	0A	0A	06	6B	65	72	6E	65	6C	18	02	Pool*...kernel..
00000100	2A	0D	0A	09	75	73	65	5F	63	75	64	6E	6E	18	00	2A	*...use_cudnn..*
00000110	0A	0A	06	73	74	72	69	64	65	18	02	12	5B	0A	05	70	...stride...[..p
00000120	6F	6F	6C	32	0A	05	66	63	33	5F	77	0A	05	66	63	33	ool2..fc3_w..fc3
00000130	5F	62	12	03	66	63	33	1A	00	22	02	46	43	2A	0D	0A	_b..fc3..".FC*..
00000140	09	75	73	65	5F	63	75	64	6E	6E	18	00	2A	0D	0A	05	.use_cudnn..*...
00000150	6F	72	64	65	72	22	04	4E	43	48	57	2A	1B	0A	17	63	order".NCHW*...c
00000160	75	64	6E	6E	5F	65	78	68	61	75	73	74	69	76	65	5F	udnn_exhaustive_
00000170	73	65	61	72	63	68	18	00	12	23	0A	03	66	63	33	12	search...#..fc3.

LeNet MNIST Example

Create mnist\images, put images



LeNet MNIST Example



Run `image-classifier`

```
PS C:\Data\Glow\mnist> 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"
Model: models
Running 1 thread(s).
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0702 00:25:25.555121 30912 Partitioner.cpp:683] Profiling a model to be partitioned cross different backends. Each sub-
network will be optimized and run on cpu backend.
I0702 00:25:25.557121 30912 Provisioner.cpp:77] Checking for active networks when adding: models
I0702 00:25:25.557121 30912 Provisioner.cpp:93] Adding partition name: models_part1 to activeFunctions_
I0702 00:25:26.311772 30912 Provisioner.cpp:717] Removing partition name: models_part1 from activeFunctions_
I0702 00:25:26.313805 30912 HostManager.cpp:586] Successfully compiled and provisioned models
I0702 00:25:26.318773 30912 ImageClassifier.cpp:238] Graph profiling is ON. Processing of output is disabled.
I0702 00:25:26.321805 30912 ImageClassifier.cpp:238] Graph profiling is ON. Processing of output is disabled.
I0702 00:25:26.325803 30912 ImageClassifier.cpp:238] Graph profiling is ON. Processing of output is disabled.
I0702 00:25:26.333804 30912 ImageClassifier.cpp:238] Graph profiling is ON. Processing of output is disabled.
I0702 00:25:26.341811 30912 ImageClassifier.cpp:238] Graph profiling is ON. Processing of output is disabled.
I0702 00:25:26.349786 30912 ImageClassifier.cpp:238] Graph profiling is ON. Processing of output is disabled.
I0702 00:25:26.356813 30912 ImageClassifier.cpp:238] Graph profiling is ON. Processing of output is disabled.
I0702 00:25:26.364770 30912 ImageClassifier.cpp:238] Graph profiling is ON. Processing of output is disabled.
I0702 00:25:26.368760 30912 ImageClassifier.cpp:238] Graph profiling is ON. Processing of output is disabled.
I0702 00:25:26.371821 30912 ImageClassifier.cpp:238] Graph profiling is ON. Processing of output is disabled.
I0702 00:25:26.384928 30912 HostManager.cpp:228] Destroying host manager...
PS C:\Data\Glow\mnist>
```

LeNet MNIST Example

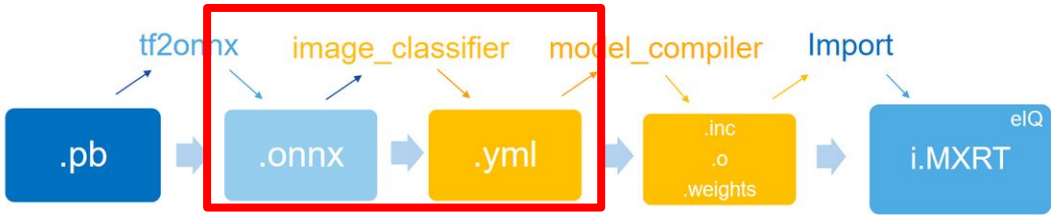


Image-classifier (해석)

-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.

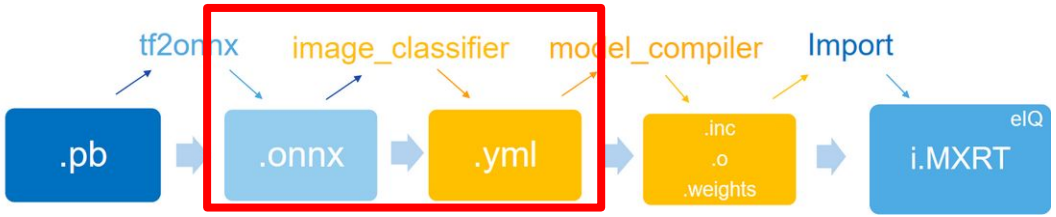
-quantization-schema=symmetric_with_power2_scale

Quantization schema. Symmetric with Power 2 scale is needed for CMSIS-NN optimizations.

-dump-profile=profile.yml

Filename to store the profiling results

LeNet MNIST Example

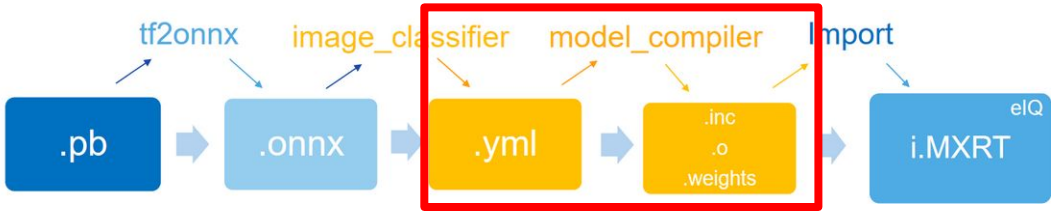


What's in `profile.yml`

C: > Data > Glow > mnist > ! profile.yml

```
1   ---
2   GlowToolsVersion: 2020-11-26
3   ...
4   ---
5   GraphPreLowerHash: 0x1582F452AC164D20
6   ...
7   ---
8   - NodeOutputName:  'conv1__1:0'
9     Min:             -2.4901134967803955078125
10    Max:             3.034297943115234375
11    Histogram:       [ 52, 739, 4996, 25755, 67239, 12193, 2952, 980, 241,
12                       53 ]
13  - NodeOutputName:  'conv1__3:0'
14    Min:             0.0
15    Max:             1.0
16    Histogram:       [ 6555, 98, 91, 68, 78, 88, 101, 69, 103, 589 ]
17  - NodeOutputName:  'conv1_b:0'
18    Min:             -0.44909933209419250488281250
19    Max:             0.03374387323856353759765625
20    Histogram:       [ 10, 0, 10, 40, 20, 0, 10, 30, 30, 50 ]
21  - NodeOutputName:  'conv1_w_1:0'
```

LeNet MNIST Example

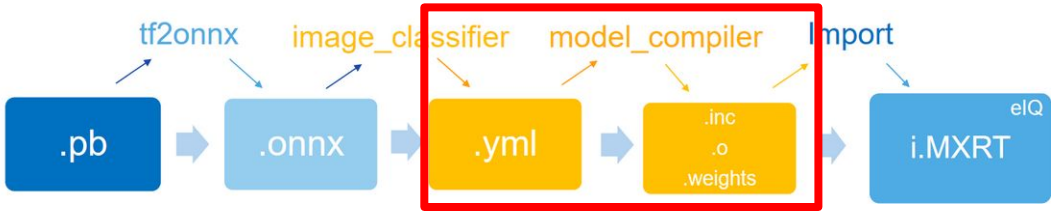


Run `model-compiler`

```
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
```

```
PS C:\Data\Glow\mnist> model-compiler -model=models -model-input="data,float,[1,1,28,28]" -emit-bundle=source -dump-grap
h-DAG="model_graph.dot" -backend=CPU -target=arm -mcpu=cortex-m7 -float-abi=hard -load-profile="profile.yml" -quantizati
on-schema=symmetric_with_power2_scale -quantization-precision-bias=Int8 -use-cmsis -network-name=mnist
Writing dotty graph for Function to: model_graph.dot
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0702 00:33:08.435181 11760 HostManager.cpp:228] Destroying host manager...
PS C:\Data\Glow\mnist>
```

LeNet MNIST Example



Run `model-compiler` (해설)

-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.**

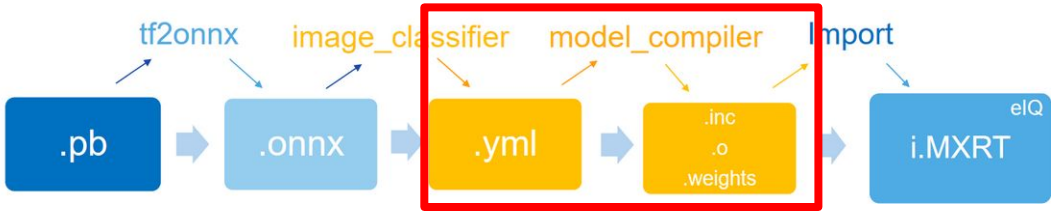
-model-input="data,float,[1,1,28,28]"

- **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 argument is not needed as this information can be read directly from the model.**

-emit-bundle=source

Directory to output the generated files

LeNet MNIST Example



Run `model-compiler` (해설)

-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

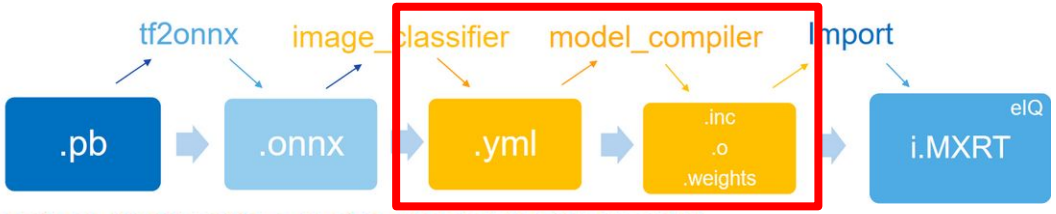
Specific CPU to compile for.

- **“cortex-m7” for M7 core (RT1050/RT1060/RT1170).**
- **“cortex-m33” for M33 core (RT685)**
- **“cortex-m4” for M4 core.**

-float-abi=hard

Compile to use floating point hardware on target

LeNet MNIST Example



-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-precisison=Int8

Use Int8 bias quantization. Needed for CMSIS-NN performance optimizations.

-quantization-schema=symmetric_with_power2_scale

Quantization schema. Symmetric with Power 2 scale is needed for CMSIS-NN performance optimizations.

-use-cmsis

Use CMSIS-NN library for supported quantized operations to speed up execution. This option only has an effect if the model is quantized with Int8 Symmetric Power 2 scale schema.

-use-hifi

Use HiFi4 DSP on RT685 for supported operations to speed up execution. Not used for RT1050/RT1060.

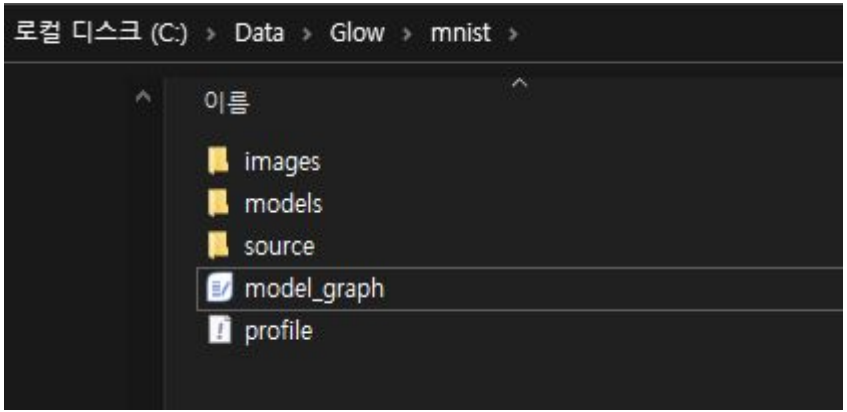
-network-name=mnist

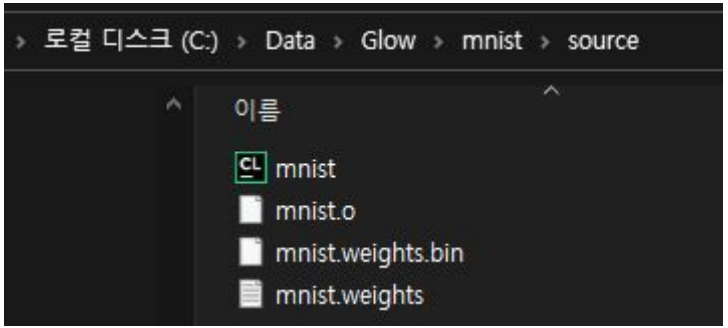
Use “mnist” as the name for the generated files. It is recommended to use a short and descriptive name as it will be used as a prefix for all macros and functions.

LeNet MNIST Example

Run `model-compiler`, generated files

- Minist.o : compiled binary code for Cortex-M7 core
- mnist.h : Header file, memory usage.





LeNet MNIST Example

Run `model-compiler`, generated files

- Minist.o : compiled binary code for Cortex-M7 core
- mnist.h : Header file, memory usage.

mnist.o

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	7F	45	4C	46	01	01	01	00	00	00	00	00	00	00	00	00	.ELF....
00000010	01	00	28	00	01	00	00	00	00	00	00	00	00	00	00	00	..(.....
00000020	60	59	00	00	00	00	00	05	34	00	00	00	00	00	28	00	`Y.....4.....(.
00000030	0A	00	01	00	F0	B5	03	AF	2D	E9	00	0F	91	B0	04	46	....ðµ.¯-é..‘°.F
00000040	10	46	91	46	01	91	00	F0	67	FB	09	F5	41	5A	04	F5	.F‘F.‘.ðgû.õAZ.õ
00000050	30	7B	09	F5	50	7C	4F	F0	00	08	00	21	04	94	CD	F8	0{.õP|Oð...!.”Íø
00000060	0C	90	CD	F8	14	C0	CD	F8	38	A0	CD	F8	18	B0	08	F1	..Íø.ÀÍø8 Íø.°.ñ
00000070	05	00	00	24	CD	F8	1C	80	0C	90	60	1D	42	46	43	46	...$Íø.€..`.BFCF
00000080	0D	94	C2	EB	C2	02	4F	EA	82	0E	22	46	9E	B2	00	25	.”ÂëÂ.Oê‚."Fž².%
00000090	1B	2E	9F	BF	A6	B2	1B	2E	72	44	19	F9	02	50	01	34	..Ÿ¿¦²..rD.ù.P.4
000000A0	2A	F8	11	50	01	31	22	B2	90	42	EF	DC	0C	9E	0D	9C	*ø.P.1"².BïÜ.ž.œ
000000B0	01	33	1A	B2	96	42	E4	DC	32	29	40	F0	94	80	04	9B	.3.²–BäÜ2)@ð”€.›
000000C0	0C	F1	14	01	0A	22	CD	F8	3C	B0	CD	E9	0A	21	4F	F4	.ñ..."Íø<°Íé.!Oô
000000D0	80	70	CD	F8	20	C0	06	24	4F	F0	00	0E	09	93	93	F9	€pÍø À.$Oð...““ù
000000E0	01	10	00	EB	81	16	93	F9	00	10	B3	46	00	EB	81	1A	...ë..“ù..³F.ë..
000000F0	D4	46	0E	98	10	96	01	3C	00	EB	4E	05	50	F8	1E	60	ÔF.˜.–.<.ëN.Pø.`
00000100	0F	98	D5	F8	32	80	50	F8	0E	30	00	EB	0E	02	0E	F1	.˜Õø2€Pø.0.ë...ñ
00000110	04	0E	4F	EA	33	29	2F	FA	89	F9	2F	FA	83	F3	D2	F8	..Oê3)/ú‰ù/úƒóÒø
00000120	19	20	C3	EA	09	40	C9	EA	23	43	4F	EA	32	21	2F	FA	. Ãê.@Éê#COê2!/ú
00000130	81	F1	2F	FA	82	F2	20	FB	06	CC	20	FB	08	AA	C2	EA	.ñ/ú‚ò û.Ì û.ªÂê
00000140	01	40	20	FB	06	BB	10	9E	20	FB	08	68	68	68	D5	F8	.@ û.».ž û.hhhÕø
00000150	36	50	23	FB	00	CC	23	FB	05	AA	C1	EA	22	41	21	FB	6P#û.Ì#û.ªÁê"A!û
00000160	00	BB	21	FB	05	86	20	04	C3	D1	0F	9C	0E	9B	33	F9	.»!û.† .ÃÑ.œ.›3ù
00000170	1E	10	14	F9	0E	00	03	EB	4E	03	11	FB	00	C2	52	12	...ù...ëN..û.ÂR.

mnist.h

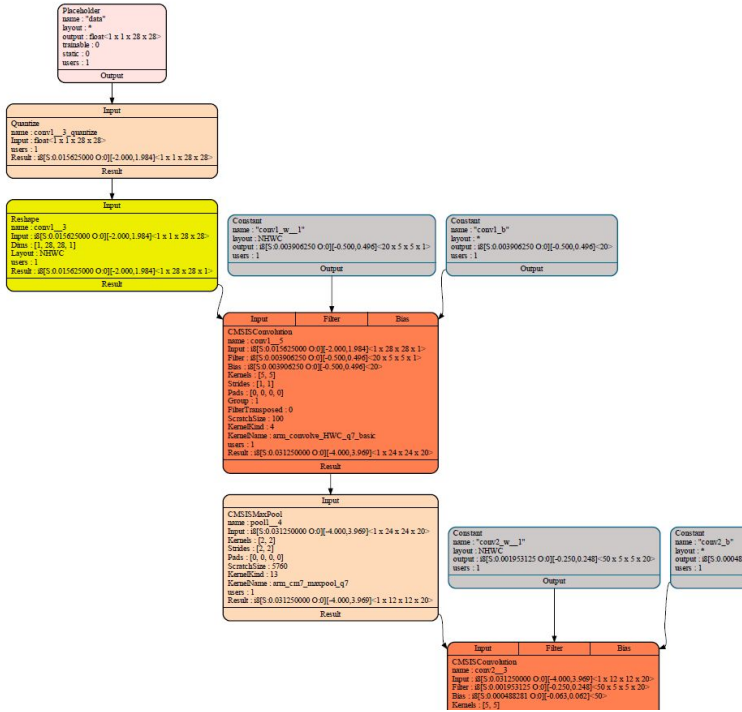
C: > Data > Glow > mnist > source > C mnist.h

```
1  // Bundle API auto-generated header file. Do not edit!
2  // Glow Tools version: 2020-11-26
3
4  #ifndef _GLOW_BUNDLE_MNIST_H
5  #define _GLOW_BUNDLE_MNIST_H
6
7  #include <stdint.h>
8
9  // ----------------------------------------------------------------
10 //                       Common definitions
11 // ----------------------------------------------------------------
12 #ifndef _GLOW_BUNDLE_COMMON_DEFS
13 #define _GLOW_BUNDLE_COMMON_DEFS
14
15 // Glow bundle error code for correct execution.
16 #define GLOW_SUCCESS 0
17
18 // Memory alignment definition with given alignment size
19 // for static allocation of memory.
20 #define GLOW_MEM_ALIGN(size)  __attribute__((aligned(size)))
```

LeNet MNIST Example

Dot file example

```

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

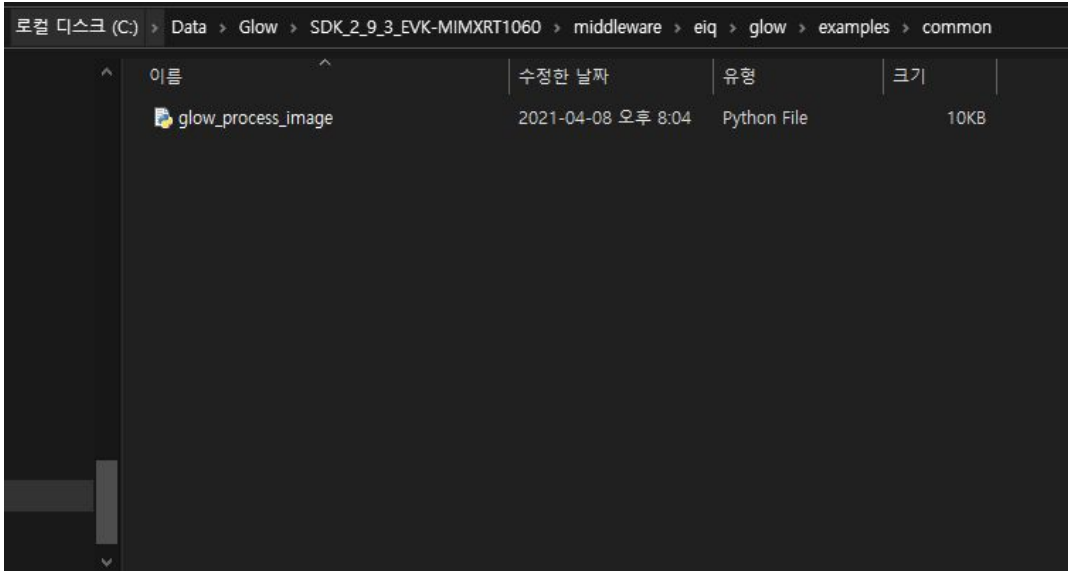


LeNet MNIST Example

Test the accuracy of the model

<SDK_dir>\middleware\eiq\glow\examples\common

Data\Glow\glow_process_image



glow_process_image.py를 C:\Data\Glow\mnist에다 갖다 놓는다.

```
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
```

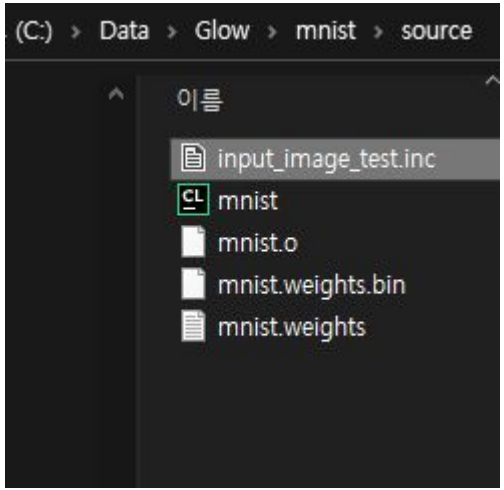
```
PS C:\Data\Glow\mnist> python glow_process_image.py -image-path="images\9_1088.png" -output-path="source\input_image_tes
t.inc" -image-mode=0to1 -image-layout=NCHW -image-channel-order=BGR -image-type=float32
Image processed to file "source\input_image_test.inc" ...
Image size = 3136 (bytes)
PS C:\Data\Glow\mnist>
```

LeNet MNIST Example

Source 폴더에 가면, input_image_test.inc 가 생긴것을 볼수 있다.

input_image_test.inc

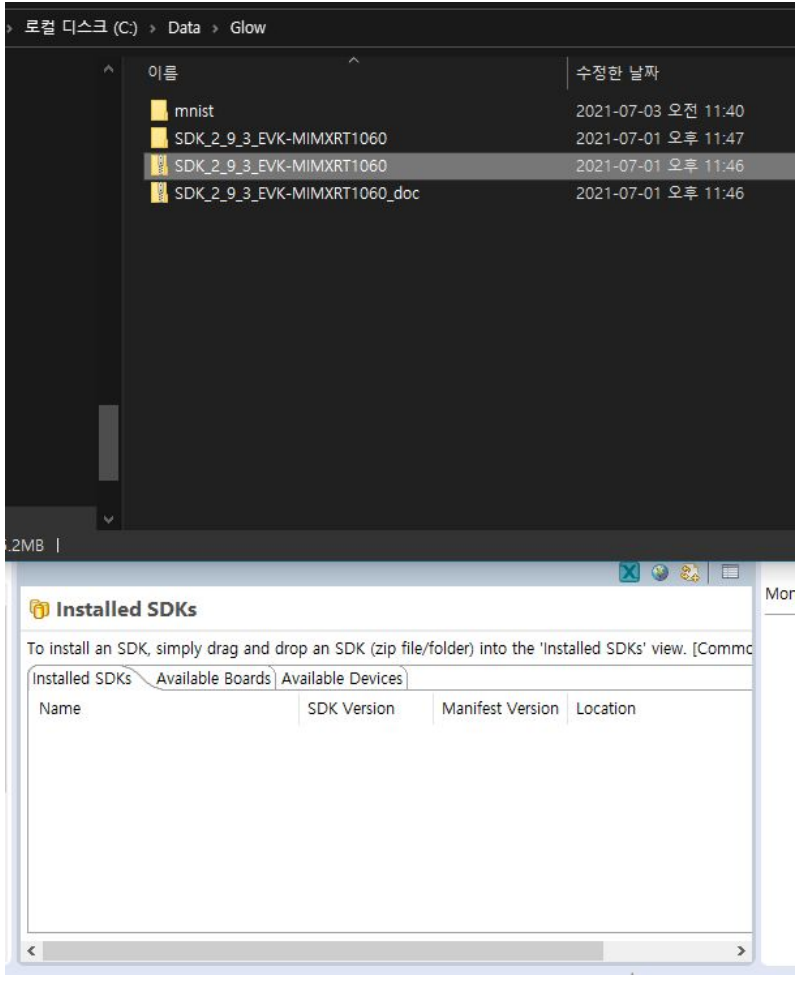
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00001830	44	2C	20	30	58	33	46	2C	20	30	58	46	46	2C	20	30	D, 0X3F, 0XFF, 0
00001840	58	46	44	2C	20	30	58	37	44	2C	20	30	58	33	46	2C	XFD, 0X7D, 0X3F,
00001850	20	30	58	46	46	2C	20	30	58	46	44	2C	20	30	58	37	0XFF, 0XFD, 0X7
00001860	44	2C	20	30	58	33	46	2C	20	0D	0A	30	58	46	34	2C	D, 0X3F, ..0XF4,
00001870	20	30	58	46	32	2C	20	30	58	37	32	2C	20	30	58	33	0XF2, 0X72, 0X3
00001880	46	2C	20	30	58	42	44	2C	20	30	58	42	43	2C	20	30	F, 0XBD, 0XBC, 0
00001890	58	33	43	2C	20	30	58	33	45	2C	20	30	58	30	30	2C	X3C, 0X3E, 0X00,
000018A0	20	30	58	30	30	2C	20	30	58	30	30	2C	20	30	58	30	0X00, 0X00, 0X0
000018B0	30	2C	20	30	58	30	30	2C	20	30	58	30	30	2C	20	30	0, 0X00, 0X00, 0
000018C0	58	30	30	2C	20	30	58	30	30	2C	20	30	58	30	30	2C	X00, 0X00, 0X00,
000018D0	20	30	58	30	30	2C	20	30	58	30	30	2C	20	30	58	30	0X00, 0X00, 0X0
000018E0	30	2C	20	0D	0A	30	58	30	30	2C	20	30	58	30	30	2C	0, ..0X00, 0X00,
000018F0	20	30	58	30	30	2C	20	30	58	30	30	2C	20	30	58	30	0X00, 0X00, 0X0
00001900	30	2C	20	30	58	30	30	2C	20	30	58	30	30	2C	20	30	0, 0X00, 0X00, 0
00001910	58	30	30	2C	20	30	58	30	30	2C	20	30	58	30	30	2C	X00, 0X00, 0X00,
00001920	20	30	58	30	30	2C	20	30	58	30	30	2C	20	30	58	30	0X00, 0X00, 0X0
00001930	30	2C	20	30	58	30	30	2C	20	30	58	30	30	2C	20	30	0, 0X00, 0X00, 0
00001940	58	30	30	2C	20	30	58	30	30	2C	20	30	58	30	30	2C	X00, 0X00, 0X00,
00001950	20	30	58	30	30	2C	20	30	58	30	30	2C	20	0D	0A	30	0X00, 0X00, ..0
00001960	58	30	30	2C	20	30	58	30	30	2C	20	30	58	30	30	2C	X00, 0X00, 0X00,
00001970	20	30	58	30	30	2C	20	30	58	30	30	2C	20	30	58	30	0X00, 0X00, 0X0
00001980	30	2C	20	30	58	30	30	2C	20	30	58	30	30	2C	20	30	0, 0X00, 0X00, 0
00001990	58	30	30	2C	20	30	58	30	30	2C	20	30	58	30	30	2C	X00, 0X00, 0X00,
000019A0	20	30	58	30	30	2C	20	30	58	30	30	2C	20	30	58	30	0X00, 0X00, 0X0



LeNet MNIST Example

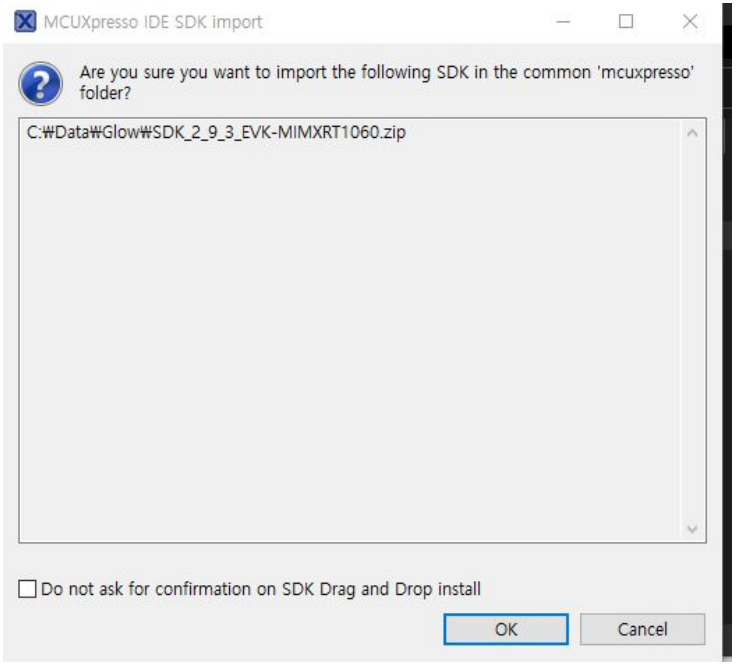
Run on the target

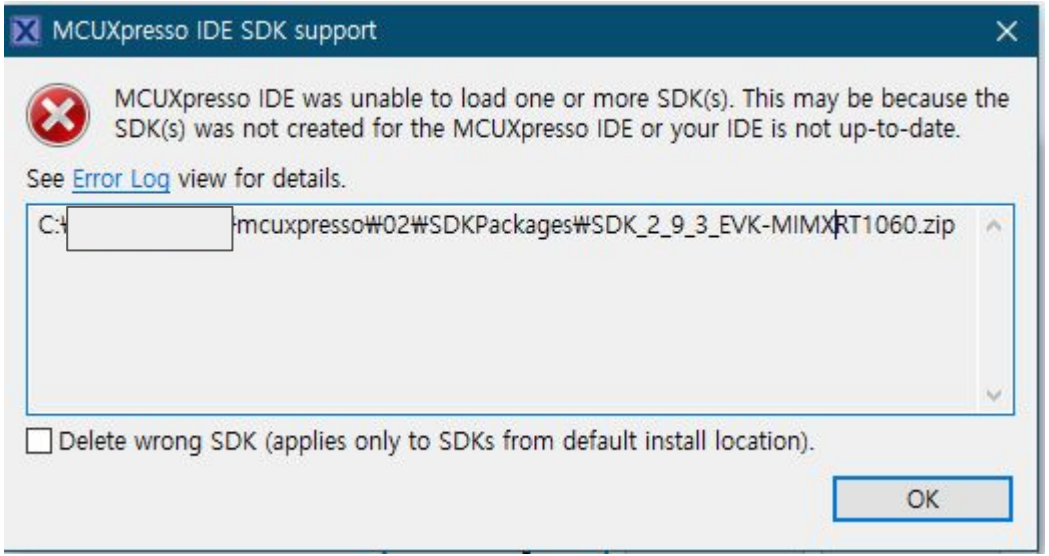
- Run MCUXpresso IDE
- Workspace directory (Check if empty)
- Drag “SDK_2_9_3_EVK-MIMXRT1060.zip”
- Drop to “Installed SDKs”



LeNet MNIST Example

Watchout for the version





LeNet MNIST Example

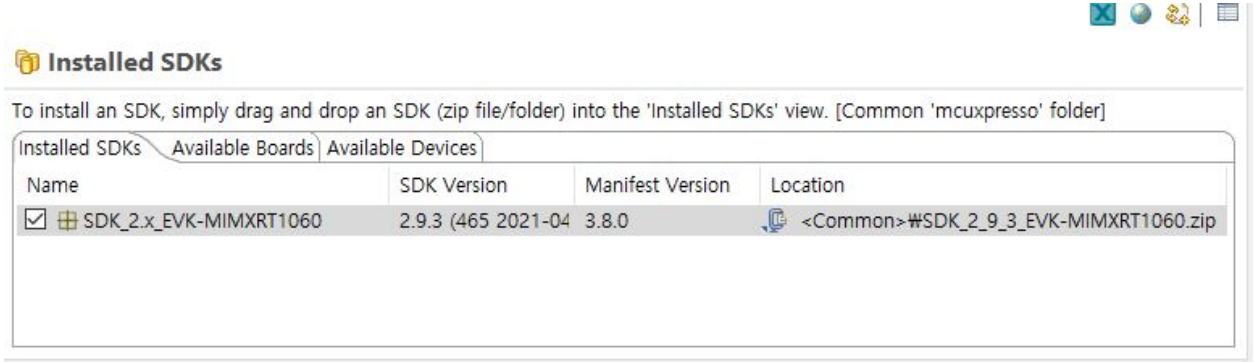
Check SDK version before doing anything!!
Source: NXP SDK site (https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-software-development-kit-sdk:MCUXpresso-SDK)

System Requirements

- SDK v2.9 requires IDE v11.3.x or later
- SDK v2.8 requires IDE v11.2.x or later
- SDK v2.7 requires IDE v11.1.x or later
- SDK v2.6 requires IDE v11.0.x or later
- SDK v2.5 requires IDE v10.3.x or later
- SDK v2.4 requires IDE v10.2.x or later
- SDK v2.3 requires IDE v10.1.x or later
- SDK v2.2 requires IDE v10.0.x or later

LeNet MNIST Example

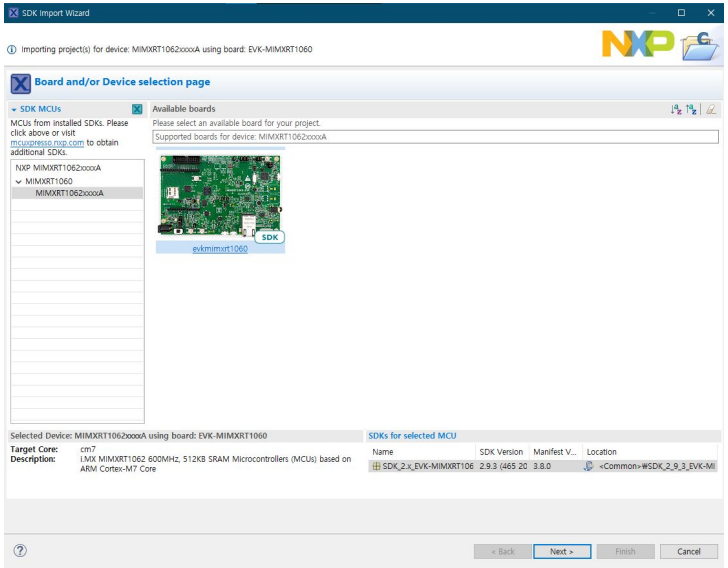
Successful installation

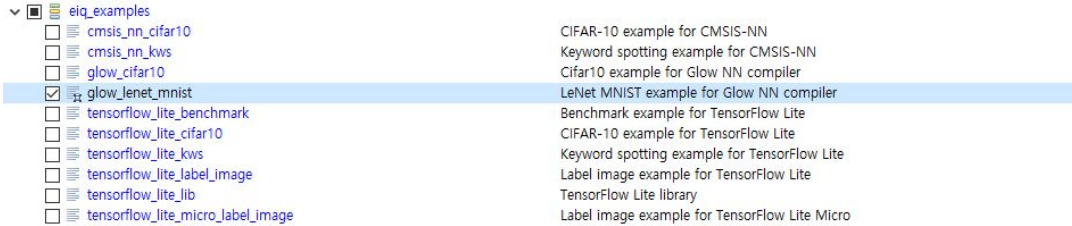


LeNet MNIST Example

Import SDK Examples

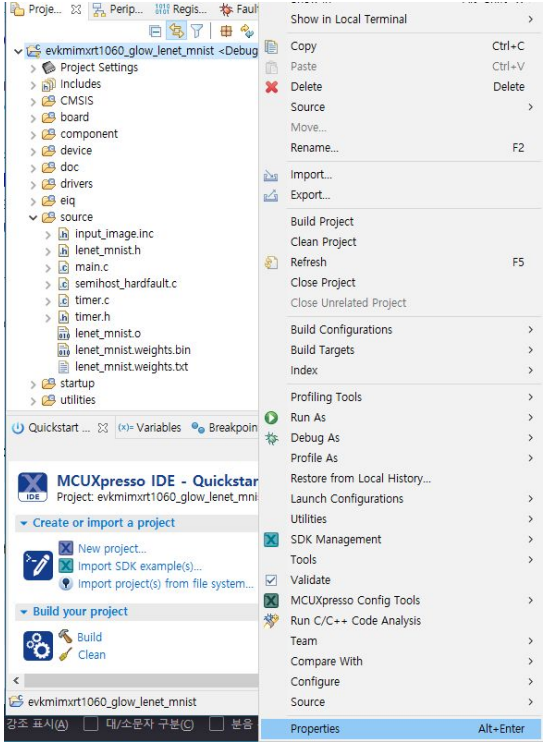
Select `eiq_examples` -> `glow_lenet_mnist`

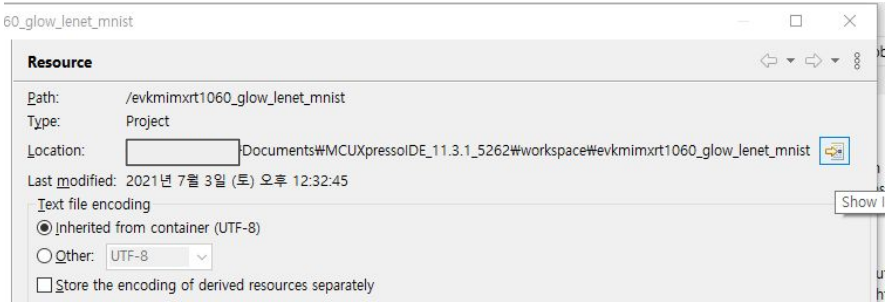


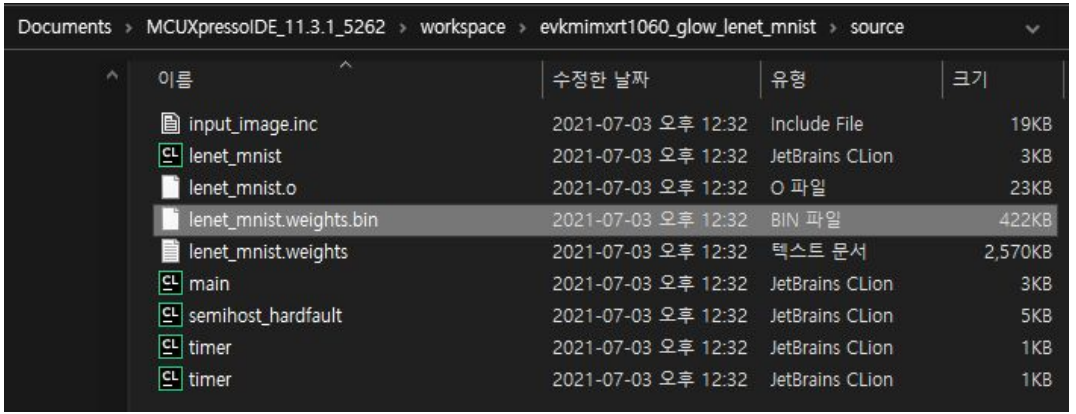


LeNet MNIST Example

Add Sources from Glow compiler output

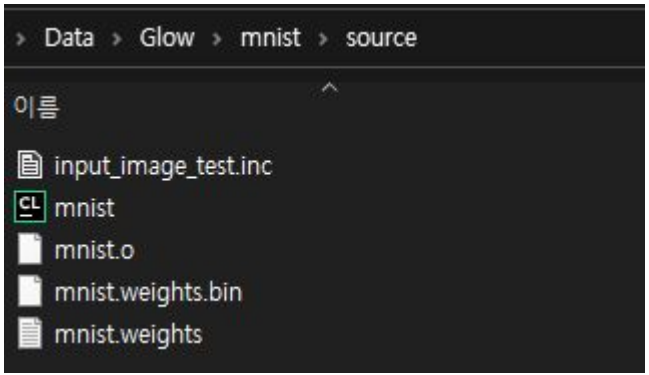


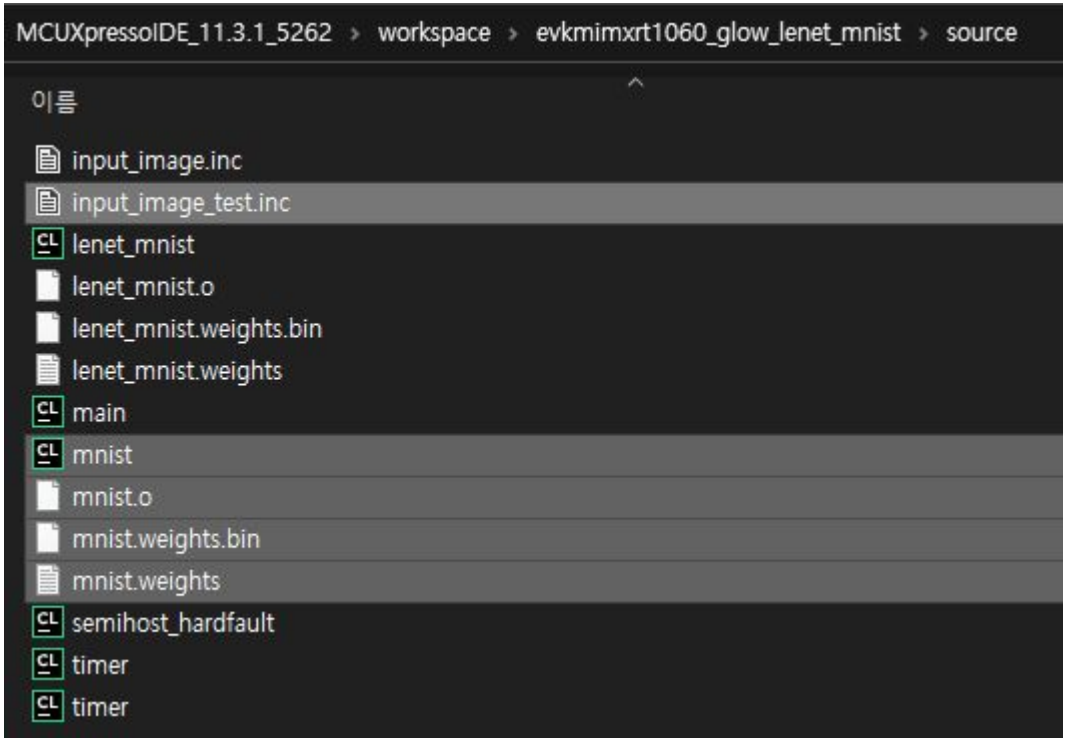




LeNet MNIST Example

Add Sources from Glow compiler output





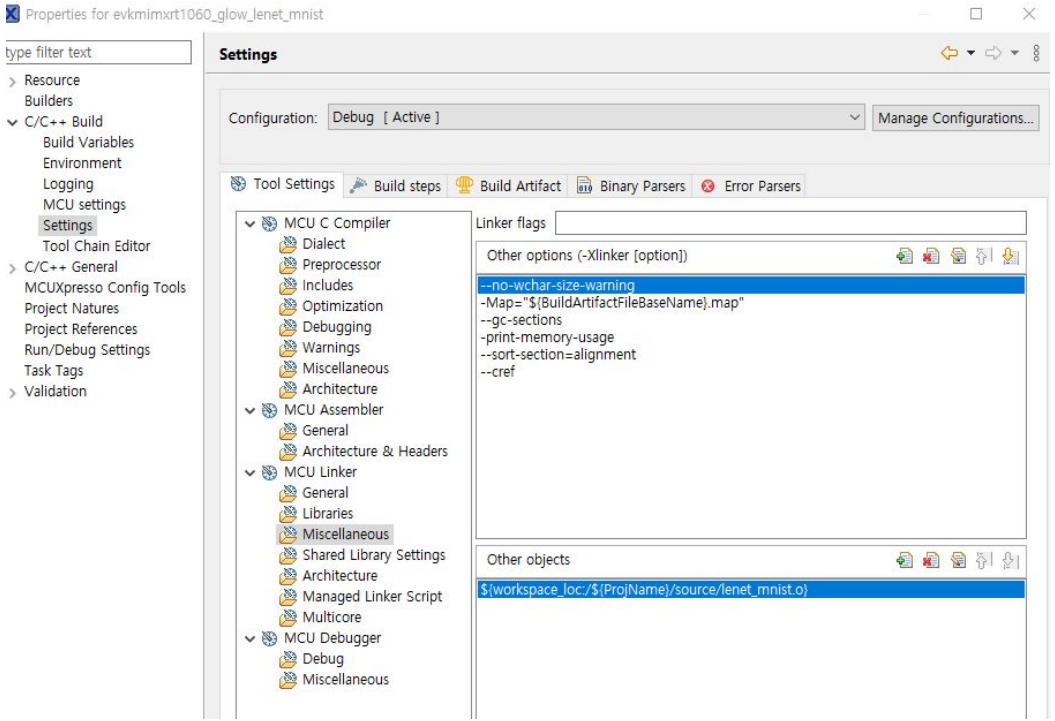
LeNet MNIST Example

Add Sources from Glow compiler output

Properties -> C/C++ build -> Settings

Other objects,

Change `lenet_mnist.o` -> `mnist.o`



LeNet MNIST Example

Source code modification: `main.c`

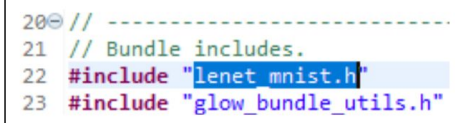
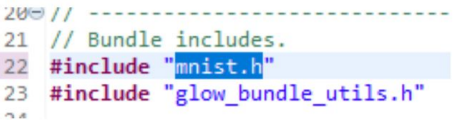
Reference “eIQ Glow Lab for RT1060.pdf” 13page

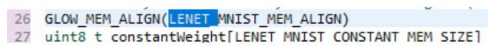
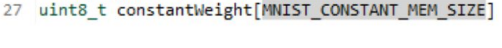
`lenet_mnist` -> `mnist`

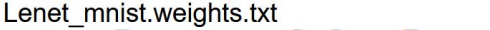
30. Everywhere in the **main.c** file that lenet_mnist is used, it should be changed to just “mnist” and use the new variable names created by the generated files. This includes:
    - Line 22 to include the generated header file “mnist.h”
    - Lines 26-27 for the new Glow variable names
    - Line 28 to include the generated weights file: “mnist.weights.txt”
    - Lines 32-37 to use the new Glow variable names
    - Line 40 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 43 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 47 should be set to the input size of the model.
    - Line 50 should be set to the number of classes of the model.
    - Line 55 to include the generated test image “input_image_test.inc”
    - Line 80 which is what starts the inference by calling “mnist(constantWeight, mutableWeight, activations)”
    - Line 88 should match the variable name used from line 50 for the number of classes in the model.

LeNet MNIST Example

Source code modification: `main.c`

Line	Before	After
22		

Line	Before	After
26-27	LENET_MNIST_MEM_ALIGN LENET_MNIST_CONSTANT_MEM_SIZE 	MNIST_MEM_ALIGN MNIST_CONSTANT_MEM_SIZE 

Line	Before	After
28	Lenet_mnist.weights.txt 	Mnist.weights.txt 

LeNet MNIST Example

Source code modification: `main.c`

Line	Before	After
32-37	LENET_MNIST_MEM_ALIGN LENET_MNIST_MUTABLE_MEM_SIZE LENET_MNIST_MEM_ALIGN LENET_MNIST_ACTIVATIONS_MEM_SIZE	MNIST_MEM_ALIGN MNIST_MUTABLE_MEM_SIZE MNIST_MEM_ALIGN MNIST_ACTIVATIONS_MEM_SIZE

```
31 // Statically allocate memory for mutable weights (moc
32 GLOW_MEM_ALIGN(LENET_MNIST_MEM_ALIGN)
33 uint8_t mutableWeight[LENET_MNIST_MUTABLE_MEM_SIZE];
34
35 // Statically allocate memory for activations (model i
36 GLOW_MEM_ALIGN(LENET_MNIST_MEM_ALIGN)
37 uint8_t activations[LENET_MNIST_ACTIVATIONS_MEM_SIZE];
```

```
31 // Statically allocate memory for mutable weights
32 GLOW_MEM_ALIGN(MNIST_MEM_ALIGN)
33 uint8_t mutableWeight[MNIST_MUTABLE_MEM_SIZE];
34
35 // Statically allocate memory for activations (mc
36 GLOW_MEM_ALIGN(MNIST_MEM_ALIGN)
37 uint8_t activations[MNIST_ACTIVATIONS_MEM_SIZE];
```

Line	Before	After
40	LENET_MNIST_data	MNIST_data

```
40 uint8_t *inputAddr = GLOW_GET_ADDR(mutableWeight, LENET_MNIST_data);
```

```
40 uint8_t *inputAddr = GLOW_GET_ADDR(mutableWeight, MNIST_data);
```

LeNet MNIST Example

Source code modification: `main.c`

Line	Before	After
43	LENET_MNIST_softmax 43 uint8_t *outputAddr = GLOW_GET_ADDR(mutableWeight, LENET_MNIST_softmax);	MNIST_softmax 43 uint8_t *outputAddr = GLOW_GET_ADDR(mutableWeight, MNIST_softmax);

Line	Before	After
47	LENET_MNIST_INPUT_SIZE 45 // ---------- Application ---------- 46 // Lenet Mnist model input data size (bytes). 47 #define LENET_MNIST_INPUT_SIZE 28*28*sizeof(float)	MNIST_INPUT_SIZE 45 // ---------- Application --- 46 // Lenet Mnist model input data size (bytes). 47 #define MNIST_INPUT_SIZE 28*28*sizeof(float)

Line	Before	After
50	LENET_MNIST_OUTPUT_CLASS 49 // Lenet Mnist model number of output classes. 50 #define LENET_MNIST_OUTPUT_CLASS 10	MNIST_OUTPUT_CLASS 49 // Lenet Mnist model number of output classes. 50 #define MNIST_OUTPUT_CLASS 10

LeNet MNIST Example

Source code modification: `main.c`

Line	Before	After
55	input_image.inc 52 // Allocate buffer for input data. 53 // pre-processed and serialized as 54 uint8_t imageData[MNIST_INPUT_SIZE] 55 #include "input_image.inc" 56 }; 57	input_image_test.inc 52 // Allocate buffer for input data. This 53 // pre-processed and serialized as text 54 uint8_t imageData[MNIST_INPUT_SIZE] = { 55 #include "input_image_test.inc" 56 }; 57

Line	Before	After
80	lenet_mnist 78 // Perform inference and compute inference time. 79 start_time = get_time_in_us(); 80 lenet_mnist(constantWeight, mutableWeight, activations);	Mnist 78 // Perform inference and compute inference time. 79 start_time = get_time_in_us(); 80 mnist(constantWeight, mutableWeight, activations);

Line	Before	After
88	LENET_MNIST_OUTPUT_CLASS 84 // Get classification top1 result and confidence. 85 float *out_data = (float*)(outputAddr); 86 float max_val = 0.0; 87 uint32_t max_idx = 0; 88 for(int i = 0; i < LENET_MNIST_OUTPUT_CLASS; i++) {	MNIST_OUTPUT_CLASS 84 // Get classification top1 result and confidence. 85 float *out_data = (float*)(outputAddr); 86 float max_val = 0.0; 87 uint32_t max_idx = 0; 88 for(int i = 0; i < MNIST_OUTPUT_CLASS; i++) {

LeNet MNIST Example

Review key code flow in `main.c`

- Set input data buffer

```
// Load input data
memcpy(bundleInpAddr, ((char *)INPUT_DATA_START) + idx * INPUT_IMAGE_SIZE, INPUT_IMAGE_SIZE);
```

- Run model

```
78  // Perform inference and compute inference time.
79  start_time = get_time_in_us();
80  mnist(constantWeight, mutableWeight, activations);
81  stop_time = get_time_in_us();
82  duration_ms = (stop_time - start_time) / 1000;
```

- Get result from output buffer

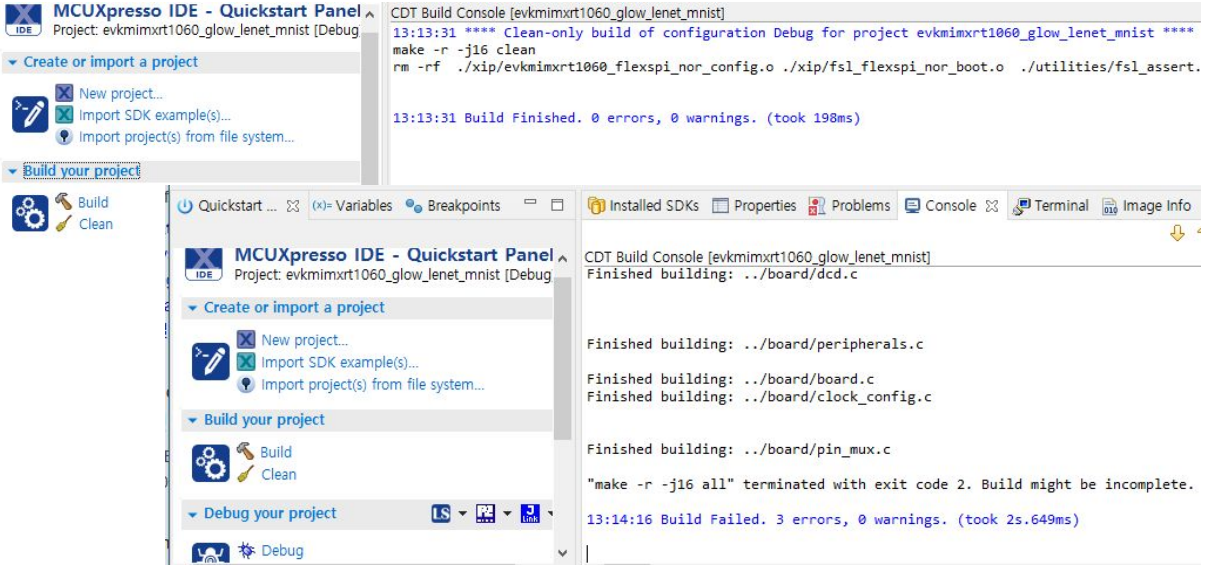
```
// Get classification top1 result and confidence
float *out_data  = (float *)(bundleOutAddr);
float max_val    = 0.0;
uint32_t max_idx = 0;
for (int i = 0; i < OUTPUT_NUM_CLASS; i++)
{
    if (out_data[i] > max_val)
    {
        max_val = out_data[i];
        max_idx = i;
    }
}
```

```
// Produce input data for bundle.
// Copy the pre-processed image data into the bundle input buffer.
memcpy(inputAddr, imageData, sizeof(imageData));

// Perform inference and compute inference time.
start_time = get_time_in_us();
mnist(constantWeight, mutableWeight, activations);
stop_time = get_time_in_us();
duration_ms = (stop_time - start_time) / 1000;

// Get classification top1 result and confidence.
float *out_data = (float*)(outputAddr);
float max_val = 0.0;
uint32_t max_idx = 0;
for(int i = 0; i < MNIST_OUTPUT_CLASS; i++) {
  if (out_data[i] > max_val) {
    max_val = out_data[i];
    max_idx = i;
  }
}

// Print classification results.
PRINTF("Top1 class = %lu\r\n", max_idx);
PRINTF("Confidence = 0.%03u\r\n",(int)(max_val*1000));
PRINTF("Inference time = %lu (ms)\r\n", duration_ms);
```

LeNet MNIST Example

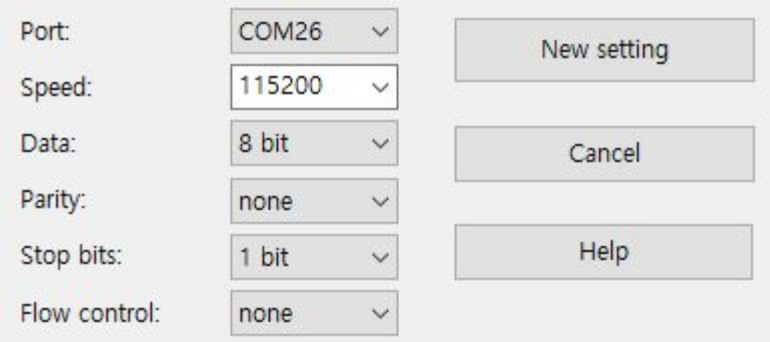
Build the project -- Whenever `model-compiler` runs!

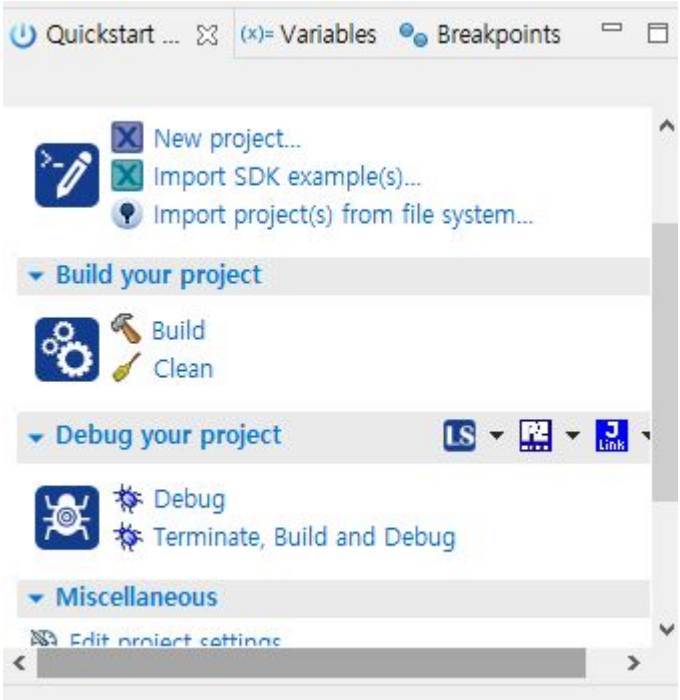
However because new Glow files were copied into the project, **you must do a clean first**. **Failing to do a clean could cause the newly imported weight data to become misaligned in memory**, and cause accuracy errors during the inferencing. This is only required when new Glow files are copied into the project.



LeNet MNIST Example

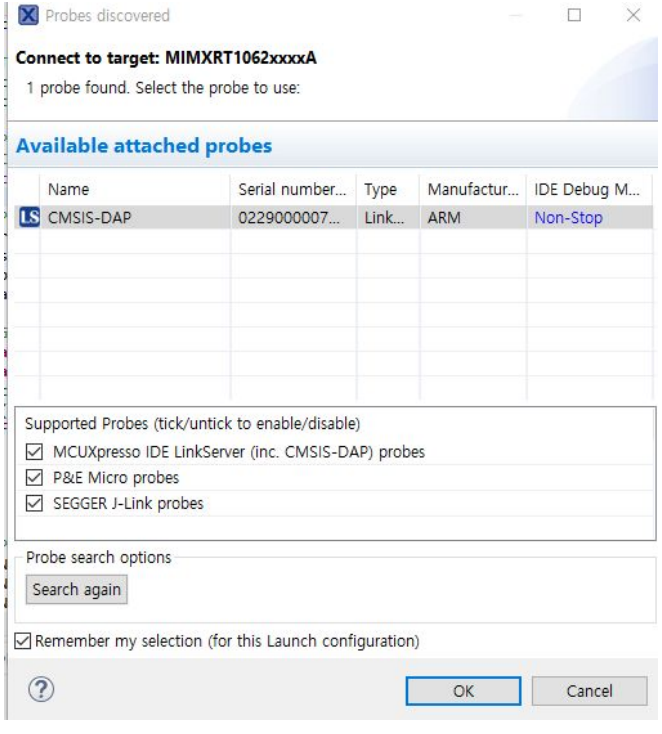
Serial Console Setup and Run Debug



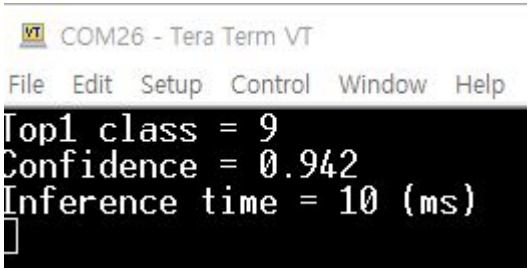


LeNet MNIST Example

Select Target then Hit Resume (F8)



Results


COM26 - Tera Term VT
File Edit Setup Control Window Help
Top1 class = 9
Confidence = 0.942
Inference time = 10 (ms)

Future Works (Ideas?)

Explore more examples with ONNX (Current format: Caffe2 (.pb))

Try ONNX model with larger model with compression:

- Inception V2 (onnx/models) 44MB
- https://github.com/onnx/models

Head to Head with microTVM

VTA on Zynq-7000 vs Glow on i.MX RT 1060

Applications

- Keyword Searching
- Time-series anomaly detection
- Real-time Object Tracking

Dive Deeper and Tinkering with Glow AOT Source code (from scratch??)

References

Develop ML Applications with the Glow Neural Network Compiler and TensorFlow Lite for i.MX RT Crossover MCUs

- Presentation: https://www.nxp.com/design/training/develop-ml-applications-with-the-glow-neural-network-compiler-and-tensorflow-lite-for-i-mx-rt-crossover-mcus:TIP-DEVELOP-ML-APPLICATIONS-WITH-THE-GLOW-NEURAL
- Slides: https://www.nxp.com/webapp/Download?colCode=TP-DEVELOP-ML-APPLICATIONS-WITH-THE-GLOW-NEURAL

Glow Documentation

- eIQ Glow Documentation
  - Glow Getting Started Guide
  - Glow User Guide
  - Glow MNIST Handwritten Digit Recognition Lab: https://community.nxp.com/t5/eIQ-Machine-Learning-Software/eIQ-Glow-Lab-for-i-MX-RT/ta-p/1123119
    - eIQ Glow for RT1060 Lab.pdf: https://community.nxp.com/pwmxy87654/attachments/pwmxy87654/eiq%40tkb/60/19/eIQ%20Glow%20for%20RT1060%20Lab.pdf

- Glow Documentation
  - Glow AOT (Ahead of Time) compiler overview https://github.com/pytorch/glow/blob/master/docs/AOT.md
  - Glow quantization https://github.com/pytorch/glow/blob/master/docs/Quantization.md
  - Glow documentation https://github.com/pytorch/glow/tree/master/docs

References

딥러닝 컴파일러 성능비교, 2019년 한국컴퓨터종합학술대회

- https://leejaymin.github.io/papers/dc18.pdf

(Blog) AI 컴파일러 NXP eIQ (Glow 컴파일러 파생)

- https://goodtogreate.tistory.com/entry/AI-%EC%BB%B4%ED%8C%8C%EC%9D%BC%EB%9F%AC-NXP-eIQ-Glow-%EC%BB%B4%ED%8C%8C%EC%9D%BC%EB%9F%AC-%ED%8C%8C%EC%83%9D

Image Classification on NXP i.MX RT1060 using Ultra-thin MobileNet DNN, 2020 10th Annual Computing and Communication Workshop and Conference (CCWC)

- https://ieeexplore.ieee.org/abstract/document/9031165?casa_token=5JrLwPygFn0AAAAA:rG4ig57Yw4hAKYJI1TxW0CoUHWfuUMAVZzWTKVf4qu549R5_GEM4onCtL7KjBEM_rPtyRCM6TxE

eIQ Glow Lab for i.MX RT

- https://community.nxp.com/t5/eIQ-Machine-Learning-Software/eIQ-Glow-Lab-for-i-MX-RT/ta-p/1123119