Explore more resources
Altera® Design Hub

altera
An Intel Company

AN 1011: TinyML Applications in Altera FPGAs Using LiteRT for Microcontrollers

Online Version
Send Feedback

848984
2025.04.07

Contents

1. Overview

This application note is a fundamental guide for developing LiteRT for Microcontrollers software in a Nios® V processor system.

Altera recommends that you have the following skills or experience before starting this project:

- Quartus® Prime software—Platform Designer, and Board Support Package Editor
- Ashling* RiscFree* IDE for Altera® FPGAs
- Fundamental in C/C++ - Running a simple Hello World application in Nios V processor

Related Information

AN 985: Nios® V Processor Tutorial
For more information about for building a Nios® V processor system and running a simple Hello World program.

1.1. Requirements

Hardware

- Any Altera FPGA Development Kit
- Power Adapter
- Intel FPGA Download Cable II

Software

- Quartus Prime
- Ashling RiscFree IDE for Altera FPGAs

Note: You need to acquire the license for the Nios V processor to compile the design in Quartus Prime software.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

2. Preparing LiteRT Inference Model

The LiteRT development workflow involves identifying a Machine Learning (ML) problem, choosing a model that solves that problem, and implementing the model on embedded devices. LiteRT is designed to run machine learning models on embedded devices with only a few kilobytes of memory. It doesn't require operating system support, any standard C or C++ libraries, or dynamic memory allocation.

The following example illustrates how to prepare a LiteRT model for digit classification. It outlines the steps needed to prepare the model in a TensorFlow Python environment before converting it into a LiteRT model.

Import the following Python libraries at the start of the Python script:

```
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import random
```

2.1. Defining the Problem

Before developing any ML problem, determine the project's objective. Explore the project's key characteristics and identify the type of problem as regression or classification.

- Regression - Establish a relationship between input variables and output variables.
- Classification - Assign input data to specific predefined categories.

This example consists of the following aspects:

- The goal is to classify a single digit from 0 to 9, which presents a classification problem.
- Based on the MNIST (Modified National Institute of Standards and Technology) database that contains a large collection of handwritten digits.
- Implements the LeNet-5 Convolutional Neural Network (CNN) model architecture.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

2.2. Gathering and Preparing Sample Data

2.2.1. Preparing Dataset

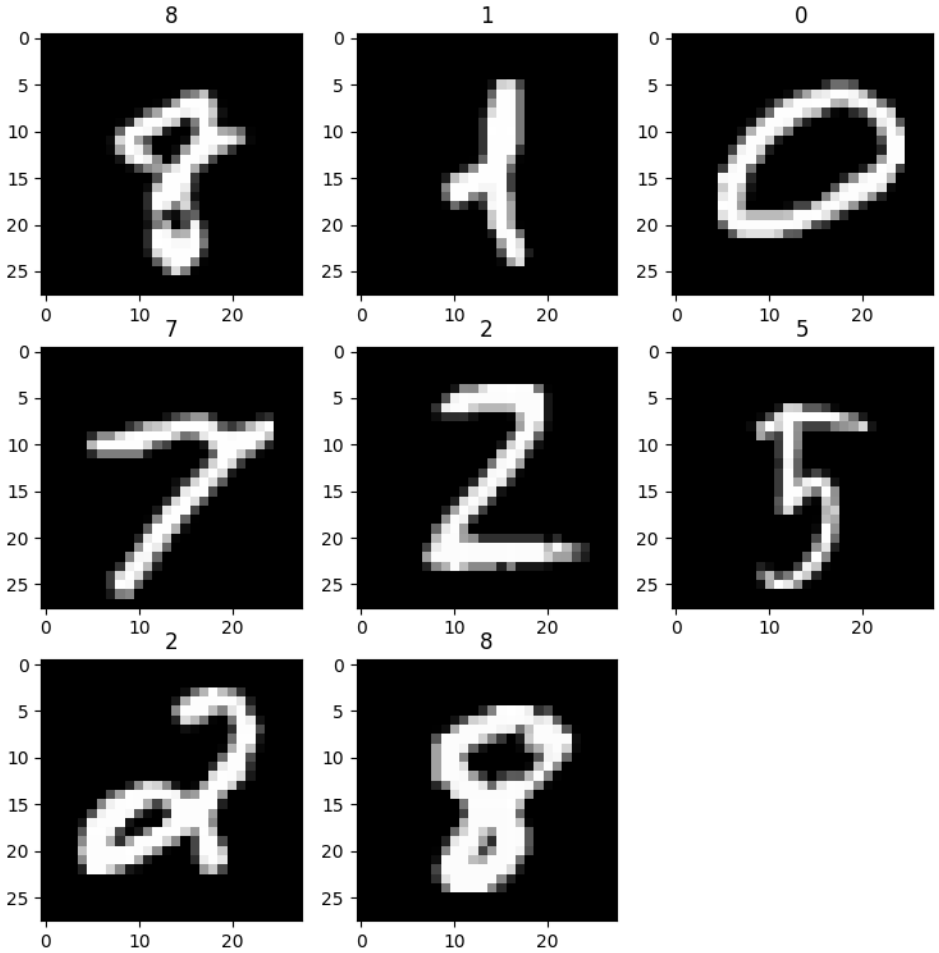
Apply the following Python commands to load the MNIST dataset. The Matplotlib library allows you to display random MNIST samples.

```
# Load the MNIST Train and Test Dataset
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

rows, cols = 28, 28

# Display Random Samples
fig = plt.figure(figsize=(9,9))
for i in range(8):
  ind = random.randint(0, len(x_train))
  plt.subplot(3,3,i+1)
  plt.imshow(x_train[ind], cmap="gray", interpolation=None)
  plt.title(y_train[ind])
```

Figure 1. Example of MNIST Samples



2.2.2. Preprocessing Dataset

Perform data preprocessing by reshaping, normalizing, and encoding the dataset.

```
# Reshape the data into a 4D Array
x_train = x_train.reshape(x_train.shape[0], rows, cols, 1)
x_test = x_test.reshape(x_test.shape[0], rows, cols, 1)

input_shape = (rows,cols,1)

# Set type as float32 and normalize the values to [0,1]
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train = x_train / 255.0
x_test = x_test / 255.0

# Transform labels to one hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

2.3. Building TensorFlow Model

2.3.1. Constructing Model Architecture

Construct a LeNet-5 model (or any other CNN model).

```
# Construct a Sequential Model
model = tf.keras.Sequential()

# Input Layer
model.add(tf.keras.layers.Input(shape=input_shape))
# C1 Convolution Layer
model.add(tf.keras.layers.Conv2D(filters=6, strides=(1,1), kernel_size=(5,5),
activation='relu'))
# S2 SubSampling Layer
model.add(tf.keras.layers.AveragePooling2D(pool_size=(2,2), strides=(2,2)))
# C3 Convolution Layer
model.add(tf.keras.layers.Conv2D(filters=6, strides=(1,1), kernel_size=(5,5),
activation='relu'))
# S4 SubSampling Layer
model.add(tf.keras.layers.AveragePooling2D(pool_size=(2,2), strides=(2,2)))
# C5 Fully Connected Layer
model.add(tf.keras.layers.Dense(units=120, activation='relu'))
# Flatten Layer
model.add(tf.keras.layers.Flatten())
# FC6 Fully Connected Layer
model.add(tf.keras.layers.Dense(units=84, activation='relu'))
# Output Layer
model.add(tf.keras.layers.Dense(units=10, activation='softmax'))

# Display a Summary of the Model
model.summary()
```

Figure 2. Summary of the Model

Model: "sequential_7"

Layer (type)	Output Shape	Param #
conv2d_14 (Conv2D)	(None, 24, 24, 6)	156
average_pooling2d_14 (AveragePooling2D)	(None, 12, 12, 6)	0
conv2d_15 (Conv2D)	(None, 8, 8, 6)	906
average_pooling2d_15 (AveragePooling2D)	(None, 4, 4, 6)	0
dense_21 (Dense)	(None, 4, 4, 120)	840
flatten_7 (Flatten)	(None, 1920)	0
dense_22 (Dense)	(None, 84)	161,364
dense_23 (Dense)	(None, 10)	850

Total params: 492,350 (1.88 MB)
Trainable params: 164,116 (641.08 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 328,234 (1.25 MB)

2.3.2. Configuring Model

Specify the loss function, optimizer, and interested metrics for the model.

```
# Compile the Model
model.compile(loss = tf.keras.metrics.categorical_crossentropy, optimizer =
"adam", metrics = ['accuracy'])
```

2.3.3. Training Model

Feed the training data into the model in batches and validate the model on the validation set after each epoch to monitor performance. Using the Matplotlib library, you can display the training and validation accuracy in a line graph.

```
# Train and Validate the Model
epochs = 10
history = model.fit(x_train, y_train, epochs=epochs, batch_size=128,
validation_data = (x_test, y_test), verbose=1)

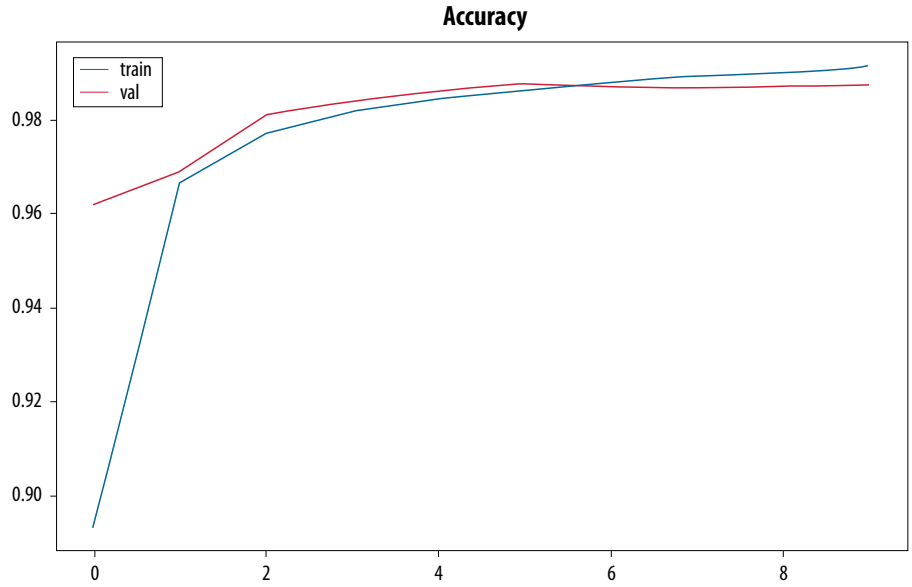
# Display the Training Progress
def summary_history(history):
  plt.figure(figsize = (10,6))
  plt.plot(history.history['accuracy'], color = 'blue', label = 'train')
  plt.plot(history.history['val_accuracy'], color = 'red', label = 'val')
  plt.legend()
  plt.title('Accuracy')
  plt.show()

summary_history(history)
```

Figure 3. Training and Validation Logs

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 28s 56ms/step - accuracy: 0.7908 - loss: 0.7733 - val_accuracy: 0.9618 - val_loss: 0.1300
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 41s 56ms/step - accuracy: 0.9632 - loss: 0.1238 - val_accuracy: 0.9688 - val_loss: 0.0957
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 41s 56ms/step - accuracy: 0.9756 - loss: 0.0793 - val_accuracy: 0.9809 - val_loss: 0.0584
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 24s 51ms/step - accuracy: 0.9805 - loss: 0.0650 - val_accuracy: 0.9838 - val_loss: 0.0503
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 41s 51ms/step - accuracy: 0.9845 - loss: 0.0503 - val_accuracy: 0.9860 - val_loss: 0.0410
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 41s 51ms/step - accuracy: 0.9859 - loss: 0.0448 - val_accuracy: 0.9874 - val_loss: 0.0394
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 26s 56ms/step - accuracy: 0.9885 - loss: 0.0374 - val_accuracy: 0.9870 - val_loss: 0.0405
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 26s 56ms/step - accuracy: 0.9897 - loss: 0.0324 - val_accuracy: 0.9866 - val_loss: 0.0387
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 40s 54ms/step - accuracy: 0.9905 - loss: 0.0299 - val_accuracy: 0.9871 - val_loss: 0.0411
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 25s 52ms/step - accuracy: 0.9918 - loss: 0.0263 - val_accuracy: 0.9873 - val_loss: 0.0410
```

Figure 4. Training and Validation Accuracy per Epoches



2.3.4. Evaluating the Model

Assess the model's performance on the test set to estimate how well it generalizes to unseen data.

```
# Evaluate Accuracy of the Model
loss ,acc= model.evaluate(x_test, y_test)
print('Accuracy : ', acc)
```

Figure 5. Classification Accuracy of TensorFlow Model on Test Dataset

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9844 - loss: 0.0503
Accuracy :  0.9872999787330627
```

2.3.5. Saving and Loading Model

Once you train a highly accurate model, Altera recommends that you save the model data for future use. Training a model consumes a lot of time, and the final accuracy can vary with each session. By saving the trained model, you can efficiently load it whenever you need it.

```
# Save Model
model.save('lenet.keras')

# Load Model
model = tf.keras.models.load_model('lenet.keras')
```

2.4. Preparing the LiteRT Model

2.4.1. Converting into LiteRT Model

A good starting point is converting a TensorFlow model to a LiteRT model without quantization, which generates a 32-bit floating-point LiteRT model.

```
# Convert the model from Tensorflow to LiteRT model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
```

Alternatively, you can use full integer-only quantization to reduce the model size and increase processing speed. However, this may impact the model's accuracy.

2.4.2. Saving the LiteRT Model

Save the LiteRT Model for reuse purposes.

```
# Save the LiteRT model
open("lenet.tflite", "wb").write(tflite_model)
```

2.4.3. Analyzing the LiteRT Model

Identify the type of op resolver needed to run the model using LiteRT for Microcontrollers libraries. Refer to *Appendix* for more information about op resolver registration in the main() function. For this LeNet-5 example, the op resolvers are:

- CONV_2D
- AVERAGE_POOL_2D
- FULLY_CONNECTED
- RESHAPE
- SOFTMAX

```
# Analyse the LiteRT Model
tf.lite.experimental.Analyzer.analyze(model_path="lenet.tflite")
```

Figure 6. Results from Analyzer

```
Subgraph#0 main(T#0) -> [T#20]
  Op#0 CONV_2D(T#0, T#10, T#11) -> [T#12]
  Op#1 AVERAGE_POOL_2D(T#12) -> [T#13]
  Op#2 CONV_2D(T#13, T#3, T#1) -> [T#14]
  Op#3 AVERAGE_POOL_2D(T#14) -> [T#15]
  Op#4 FULLY_CONNECTED(T#15, T#2, T#8) -> [T#16]
  Op#5 RESHAPE(T#16, T#9[-1, 1920]) -> [T#17]
  Op#6 FULLY_CONNECTED(T#17, T#5, T#7) -> [T#18]
  Op#7 FULLY_CONNECTED(T#18, T#4, T#6) -> [T#19]
  Op#8 SOFTMAX(T#19) -> [T#20]
```

Related Information

Appendix on page 39

2.4.4. Loading a LiteRT Interpreter

Setup the LiteRT Interpreter to test the newly converted LiteRT model.

```
# Load the LiteRT model in TFLite Interpreter
interpreter = tf.lite.Interpreter(model_path="lenet.tflite")

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Adjust the model interpreter to take 10,000 inputs at once instead of just 1
interpreter.resize_tensor_input(input_details[0]["index"], (x_test.shape[0],
rows, cols, 1))
interpreter.resize_tensor_input(output_details[0]["index"], (y_test.shape[0],
10))
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

2.4.5. Evaluating the LiteRT Model

Since the LiteRT model is generated without quantization, the final accuracy is expected to be preserved, which is the same as 0.9872 from the TensorFlow model.

It's important to check how much accuracy is lost after using post-training quantization. If the loss is significant, consider using quantization-aware training.

```
# Set the test input and run
interpreter.set_tensor(input_details[0]["index"], x_test)
interpreter.invoke()

# Get the result and check its accuracy
output_data = interpreter.get_tensor(output_details[0]["index"])

a = [np.argmax(y, axis=None, out=None) for y in output_data]
b = [np.argmax(y, axis=None, out=None) for y in y_test]

accuracy = (np.array(a) == np.array(b)).mean()
print("TFLite Accuracy:", accuracy)
```

Figure 7. Classification Accuracy of LiteRT Model on Test Dataset

```
TFLite Accuracy: 0.9873
```

2.5. Preparing Binaries for C/C++

Ensure you have acquired the following binaries before proceeding to the next chapters.

Table 1. LiteRT for Microcontrollers Binaries

Binaries	Description
`model_data.cc`	Model params stored within a C array.
`model_data.h`	Header file of `model_data.h`.
`model_settings.h`	List of classes to decode classification results in `main.cc`.
`main.cc`	Main LiteRT application to setup LiteRT model, uploading images, classifying images and profiling.
`Multiple figure.h`	Sample of MNIST images stored within a C array.

2.5.1. Converting LiteRT to a C Array

In a microcontroller environment, the LiteRT model is included as a C array and compiled into the C application. Use the following Python script to convert the LiteRT model into a C array:

```
! xxd -i lenet.tflite > model_data.cc
```

Example 1. Content of `model_data.cc`

```
unsigned char lenet_tflite[] = {…};
unsigned int lenet_tflite_len = 660296;
```

Apply the follow changes to improve the processor execution:

```
#include "model_data.h"

alignas(8) const unsigned char lenet_tflite[] = {…};
const unsigned int lenet_tflite_len = 660296;
```

2.5.2. Preparing a Header File for LiteRT C Array

Example 2. Content of `model_data.h`

```
#ifndef MODEL_DATA_H
#define MODEL_DATA_H

extern const unsigned char lenet_tflite[];
extern const unsigned int lenet_tflite_len;

#endif  // MODEL_DATA_H
```

2.5.3. Preparing Supporting Header File for LiteRT C Array

The example `main()` function implements the supporting `model_settings.h` header file to decode the output of the LiteRT model.

Example 3. Content of `model_settings.h`

```
#ifndef IMAGE_CLASSIFICATION_MODEL_SETTINGS_H_
#define IMAGE_CLASSIFICATION_MODEL_SETTINGS_H_

// Keeping these as constant expressions allow us to allocate fixed-sized arrays
// on the stack for our working memory.

// All of these values are derived from the values used during model training,
// if you change your model you'll need to update these constants.
constexpr int kNumCols = 28;
constexpr int kNumRows = 28;
constexpr int kNumChannels = 1;

constexpr int kMaxImageSize = kNumCols * kNumRows * kNumChannels;

constexpr int kCategoryCount = 10;
constexpr int k0Index    = 0;
constexpr int k1Index    = 1;
constexpr int k2Index    = 2;
constexpr int k3Index    = 3;
constexpr int k4Index    = 4;
constexpr int k5Index    = 5;
constexpr int k6Index    = 6;
constexpr int k7Index    = 7;
constexpr int k8Index    = 8;
constexpr int k9Index    = 9;

constexpr const char* kCategoryLabels[kCategoryCount] = {
  "number 0",
  "number 1",
  "number 2",
  "number 3",
  "number 4",
  "number 5",
  "number 6",
  "number 7",
  "number 8",
  "number 9"
};

#endif  // IMAGE_CLASSIFICATION_MODEL_SETTINGS_H_
```

2.5.4. Preparing Main Function to Run TinyML

Altera recommends reading and understanding the LiteRT documentation before writing the `main()` function in the Appendix. Refer to the following links for more information.

Related Information

- LiteRT – Hello World Example
- LiteRT - Get started with microcontrollers
- Appendix on page 39

2.5.5. Converting MNIST Sample into C Array

You can prepare a collection of C arrays representing MNIST samples. These arrays serve as static inputs to test the LiteRT C array model in the Nios V processor before deployment.

Repeat this Python script for a few rounds until every class (0 to 9) is collected. The script replaces duplicated class, thus resulting in a single sample for each class.

```
# Convert MNIST samples into a C array
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import random
import sys

# Load the MNIST Train and Test Dataset
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

rows, cols = 28, 28

# Reshape the data into a 4D Array
x_test = x_test.reshape(x_test.shape[0], rows, cols, 1)

input_shape = (rows,cols,1)

# Set type as float32 and normalize the values to [0,1]
x_test = x_test.astype('float32')
x_test = x_test / 255.0

# Transform labels to one hot encoding
y_test = tf.keras.utils.to_categorical(y_test, 10)

img = x_test*255.0
img = img.astype(np.uint8)
img_label=np.argmax(y_test, axis=1)

# Repeat for a few rounds to get all numbers (0-9)
fig = plt.figure(figsize=(9,9))
for i in range(9):
  ind = random.randint(0, len(img))
  np.set_printoptions(threshold=sys.maxsize)
  string1 = np.array2string(img[ind], separator=', ')
  c_array = string1.replace('[', '{').replace(']', '}').replace('.', '')
  c_array_label = img_label[ind]

  plt.subplot(3,3,i+1)
  plt.imshow(img[ind], cmap="gray", interpolation=None)
  plt.title(c_array_label)

  base_path = "figure-"
  label_name = str(c_array_label)
  file_type = ".h"
  file_name = base_path + label_name + file_type
  with open(file_name, "w") as f:
    f.write("#include <stdint.h>\n\n")
    f.write("#define IMAGE_WIDTH 28\n")
    f.write("#define IMAGE_HEIGHT 28\n")
    f.write("#define NUM_CHANNELS 1\n")
    f.write("#define IMAGE_SIZE (IMAGE_WIDTH * IMAGE_HEIGHT * NUM_CHANNELS)\n\n")
    f.write("uint8_t test_image[IMAGE_HEIGHT][IMAGE_WIDTH][NUM_CHANNELS] = ")
    f.write(c_array)
    f.write(";")
```

Figure 8. Randomly Selected MNIST Samples

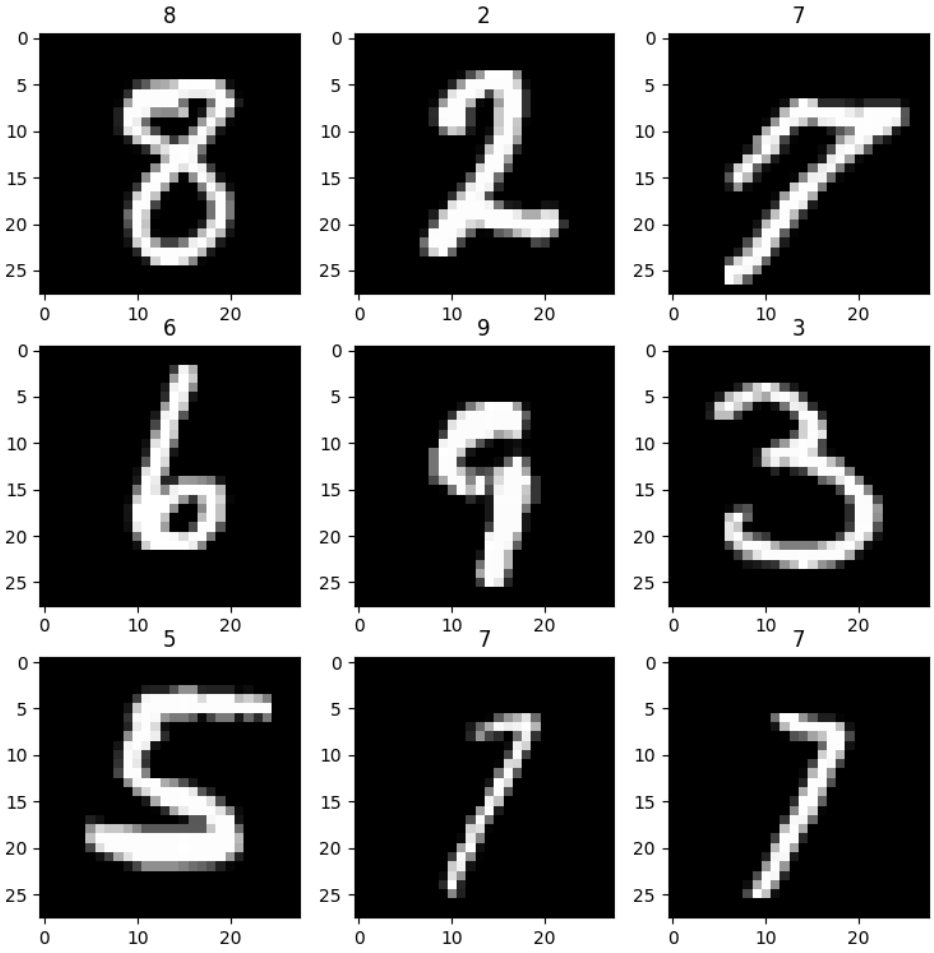
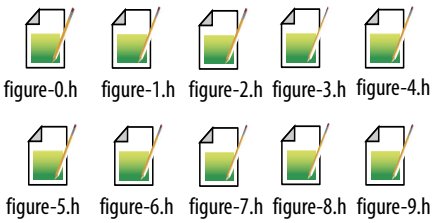


Figure 9. Complete Set of MNIST Samples



3. Generating Nios V Processor System

The general implementation of the Nios V processor system has two parts: hardware design and software design. The hardware design, which comprises the Nios V processor and peripherals, is developed using the Quartus Prime software.

Developing a Nios V processor application requires a software design that complements the processor hardware design. You can develop a Nios V processor software design using Ashling RiscFree IDE for Altera FPGAs.

3.1. Building Hardware Design in Platform Designer

This document shows how to use a simple Nios V processor hardware design for image classification with LiteRT for Microcontrollers libraries.

Table 2. Component Description

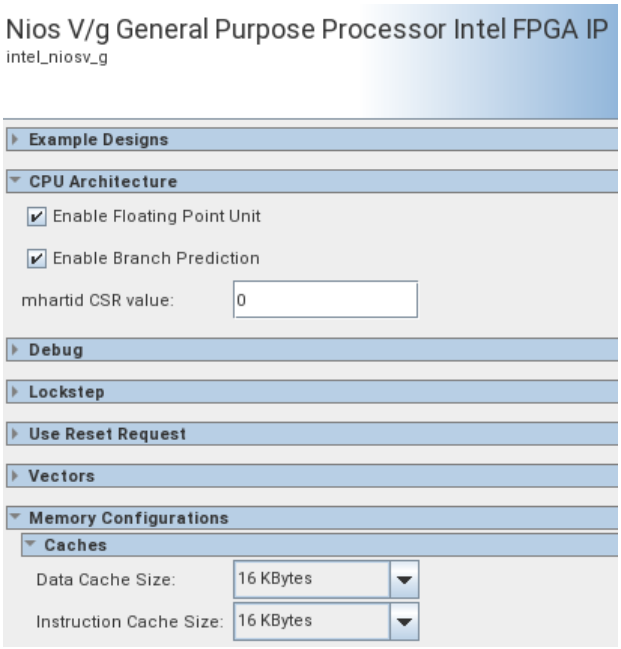
Components	Description
Nios V/m Processor Intel® FPGA IP	Runs application by executing instructions.
JTAG UART Intel FPGA IP	Enables serial character communication between Nios V/m processor and host computer
On-Chip Memory II Intel FPGA IP	Stores data and instructions.
Reset Release Intel FPGA IP	Recommended reset output in SDM-based devices.

Optionally, you can implement a Nios V/g processor with branch prediction, caches, and floating-point units for better performance. Altera recommends starting the project with large On-Chip RAM due to the size of the LiteRT C array model.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Figure 10. Enabling Cache and Floating-Point Unit



Related Information

AN 985: Nios® V Processor Tutorial
For more information about building the hardware design.

3.2. Building Software Design with Ashling RiscFree IDE for Altera FPGAs

Note: Ensure you complete the steps in Preparing LiteRT Inference Model before continuing this chapter.

Once the processor system is ready, start building the software design using Ashling RiscFree IDE for Altera FPGAs. Follow these steps:

1. Create a Board Support Package (BSP) project.
2. Build LiteRT for Microcontrollers static library.
3. Create a Nios V application project with Hello World TinyML example source code.
4. Import both projects into RiscFree IDE's workspace.
5. Build the Hello World application.

Altera recommends you create a similar directory tree in your design project to ensure a streamlined build flow. The following software design flow is based on this directory tree.

To create the software project directory tree:

- In your design project folder, create a folder called `software`.
- In the `software` folder, create two folders called `app` and `bsp`.

Figure 11. Software Project Directory Tree

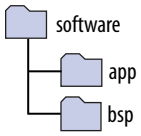


Table 3. LiteRT for Microcontrollers Binaries at Prerequisite

Binaries	Description
`model_data.cc`	Model params stored within a C array.
`model_data.h`	Header file of `model_data.h`.
`model_settings.h`	List of classes to decode classification results in `main.cc`.
`main.cc`	Main LiteRT application to setup LiteRT model, uploading images, classifying images and profiling.
`Multiple figure.h`	Sample of MNIST images stored within a C array.

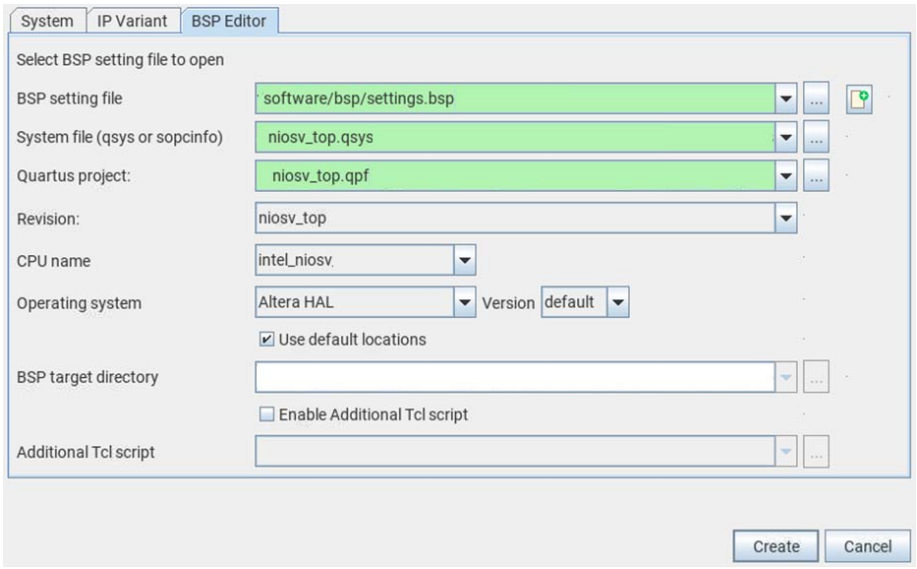
3.2.1. Creating a Board Support Package

Board Support Package (BSP) provides a software runtime environment for embedded systems, such as Nios V/m processor systems. Platform Designer includes the BSP Editor tool to generate and configure BSP content.

Follow these steps to create a BSP:

1. In the Quartus Prime software, go to **Tools ➤ Platform Designer**.
2. In the Platform Designer window, go to **File ➤ New BSP**.
3. The **Create New BSP** window appears.
4. For **BSP setting file**, create a BSP file (`settings.bsp`) in `<Working directory>/software/bsp/settings.bsp`.
5. For **System file (qsys or sopcinfo)**, select the Nios V/m processor Platform Designer system (`niosv_top.qsys`).
6. For **Quartus project**, select the example design Quartus Prime Project File (`niosv_top.qpf`).
7. For **Revision**, select `niosv_top`.
8. For **CPU name**, select `intel_niosv_m_0`.
9. Select the **Operating system** as **Altera HAL**.
10. Click **Create** to create the BSP file.

Figure 12. Create New BSP Window



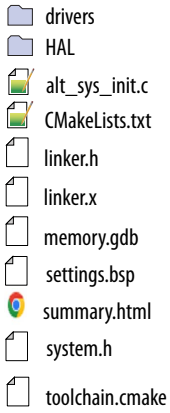
11. The **BSP Editor** tab appears.
12. Modify the GNU compiler flags as follows:

Table 4. GNU Compiler Flags Modification

BSP Settings	Values
`cflags_user_flags`	`-ffunction-sections -fdata-sections -fno-rtti -fno-exceptions`
`cflags_defined_symbols`	`-DTF_LITE_STATIC_MEMORY`
`cxx_flags`	`-std=c++11`
`link_flags`	`-Wl,--gc-sections`
`cflags_optimization`	`-O3`

13. Click **Generate BSP** to generate the BSP file.
14. The BSP Editor generates the BSP files in `<Working directory>/software/bsp` folder.

Figure 13. Generated BSP Files



3.2.2. Building LiteRT for Microcontroller Static Library

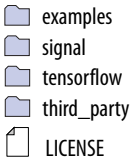
You must build LiteRT for Microcontroller libraries in Linux environments. If you are using Windows OS, you can invoke Windows* Subsystem for Linux* (WSL) within Nios V Command Shell.

Run the following commands:

```
$ sudo apt update
$ sudo apt install make git python3 unzip python3-pip
$ pip3 install numpy Image
$ git clone --depth 1 https://github.com/tensorflow/tflite-micro.git
$ cd tflite-micro
$ python3 tensorflow/lite/micro/tools/project_generation/create_tflm_tree.py\
 -e hello_world tflite_app
```

Note: Change Directory command (`cd`) to tflite-micro is mandatory. The execution of `create_tflm_tree.py` Python script fails when it is executed from different directory.

Figure 14. Content of the tflite_app Folder



Altera recommends you complete the following tasks:

- Review the Hello World TinyML example in the `examples` folder. It is a prediction model on the sine function. For more information, refer to LiteRT – Hello World Example and LiteRT - Get started with microcontrollers.
- Modify the `micro_time.cc` to be compatible with the Nios V processor. You can find the source code `micro_time.cc` in `tensorflow\lite\micro` folder. For more information, refer to LiteRT - Porting to a new platform.

Figure 15. Modification on micro_time.cc

```
#if !defined(TF_LITE_USE_CTIME)

// Reference implementation of the ticks_per_second() function that's required
// for a platform to support Tensorflow Lite for Microcontrollers profiling.
// This returns 0 by default because timing is an optional feature that builds
// without errors on platforms that do not need it.
uint32_t ticks_per_second() { return ALT_CPU_TICKS_PER_SEC; }

// Reference implementation of the GetCurrentTimeTicks() function that's
// required for a platform to support Tensorflow Lite for Microcontrollers
// profiling. This returns 0 by default because timing is an optional feature
// that builds without errors on platforms that do not need it.
uint32_t GetCurrentTimeTicks() { return clock(); }

#else  // defined(TF_LITE_USE_CTIME)
```

Related Information

- LiteRT – Hello World Example
- LiteRT - Get started with microcontrollers
- LiteRT - Porting to a New Platform

3.2.3. Creating an Application Project File

Application Project (APP) stores the software TinyML application for Nios V/g processor system.

Follow these steps to create an application project file:

1. In <Working directory>/software/app folder, copy the following LiteRT for Microcontrollers libraries from tflite_app.
   a. signal
   b. tensorflow
   c. third_party
2. Create a new folder in app called image_classification (or any preferred name).
3. Navigate into the image_classification folder, create two folders named image and model, and upload the main.cc binary (Refer to the examples in Appendix).
4. In image folder, store the MNIST C array samples.
5. In model folder, store the model_data.cc, model_data.h and model_settings.h source codes.
6. Launch the Nios V Command Shell.

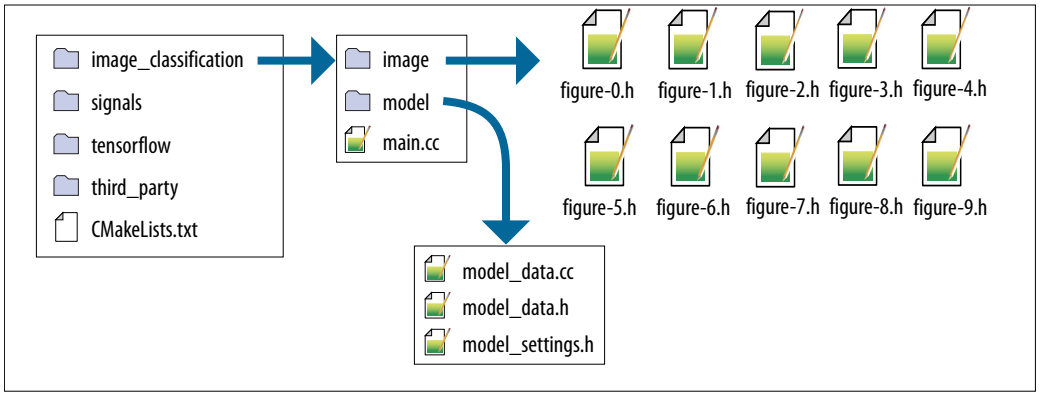
```
$ niosv-shell
```

7. Execute the command below to generate an application CMakeLists.txt.

```
$ niosv-app --bsp-dir=software/bsp --app-dir=software/app \
--srcs-recursive=software/app/image_classification,\
software/app/signal,\
software/app/tensorflow \
--incs=software/app,\
software/app/image_classification/model,\
```

```
software/app/image_classification/image,\
software/app/tensorflow,\
software/app/third_party/flatbuffers/include,\
software/app/third_party/gemmlowp,\
software/app/third_party/kissfft,\
software/app/third_party/ruy
```

Figure 16. Content of app Folder with Generated CMakeLists File



3.2.4. Building the LiteRT TinyML Application

Altera recommends importing the BSP and APP project into Ashling RiscFree IDE for Altera FPGAs for better project management and user experience.

Alternatively, run the cmake and make command to build the application.

- For debug build, select Debug configuration

```
$ niosv-shell
$ cmake -S software/app -B software/app/build/Debug -G "Unix Makefiles" \
-DCMAKE_BUILD_TYPE=Debug
$ make -C sw/app/build/Debug
```

- For release build, select Release configuration

```
$ niosv-shell
$ cmake -S software/app -B software/app/build/Release -G "Unix Makefiles" \
-DCMAKE_BUILD_TYPE=Release
$ make -C sw/app/build/Release
```

Related Information

Ashling* RiscFree* Integrated Development Environment (IDE) for Intel® FPGAs User Guide: Importing Nios® V Processor Project

4. Generating Arm Processor System

You can implement Arm HPS in two stages: first, the hardware design, and then the software design:

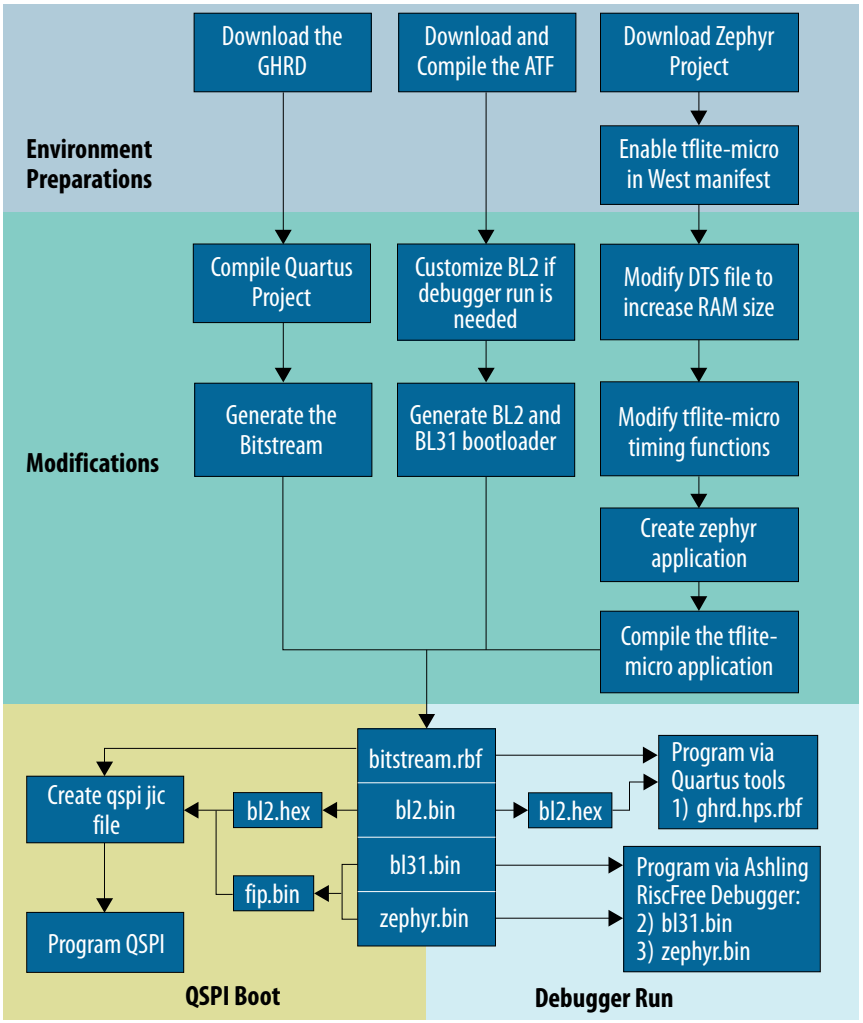
- You can use the Altera Golden Hardware Reference Design (GHRD) to implement the hardware design
- The software design comprises two primary components: the Arm-Trusted-Firmware (ATF) bootloader and the related application development.

The following figures summarizes the overall flow.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Figure 17. Development Flow Summary on ARM Processor System



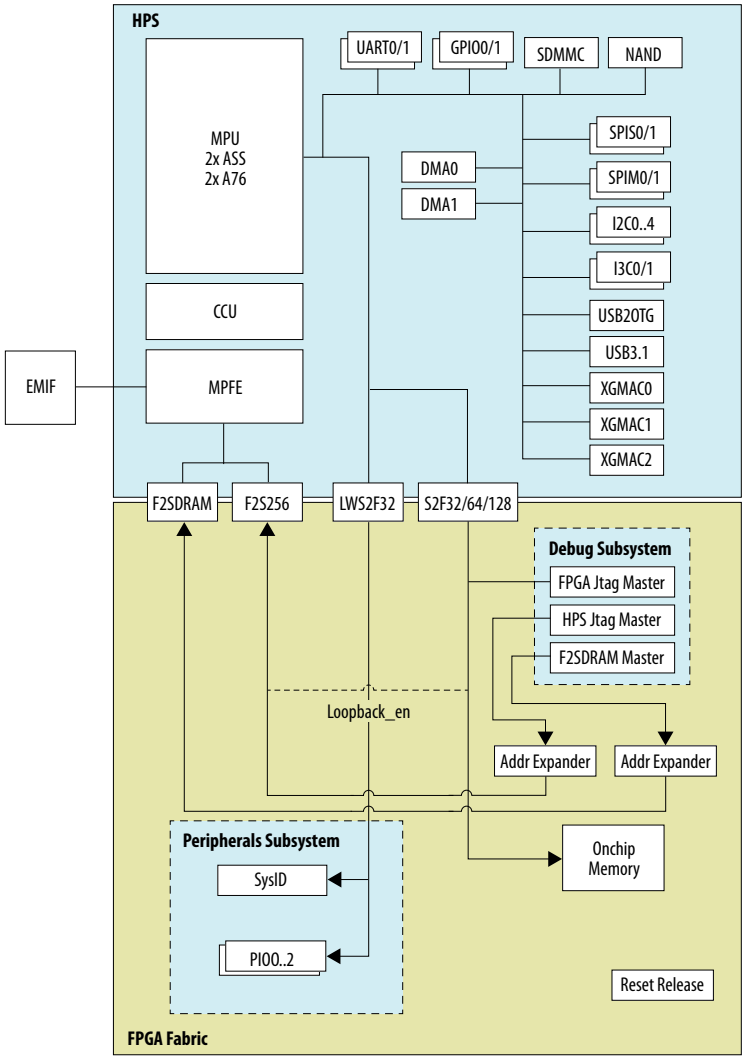
4.1. GHRD Hardware Design

Altera provides several GHRDs based on the FPGA SoC devices. This includes Stratix® 10, Agilex™ 7, and Agilex 5. The GHRD contains the following items:

- Arm* Cortex* processors
- Secure Digital/Embedded Multimedia Card (SD/eMMC) host controller
- Ethernet Media Access Controller (EMAC)
- USB UART
- HPS External Memory Interface (EMIF)
- FPGA peripherals, which can be customized based on the design needs.

A generic system-level design of Altera GHRD can be seen in the following figure. For more information, refer to the *Hard Processor System Technical Reference Manual* for specific Altera FPGA SoC devices.

Figure 18. Altera GHRD System Level Design



Refer to the following steps to prepare the GHRD:

1. Identify the target Altera FPGA SoC devices. This example uses the Agilex 5 FPGA E-Series 065B Premium Development Kit.
2. Install the appropriate Quartus Prime Pro Edition software version. This example uses Quartus Prime Pro Edition software version 24.3.
3. Start with the toolchain setup:

```
$ wget https://developer.arm.com/-/media/Files/downloads/gnu/11.2-2022.02/
binrel

    /gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz
$ tar xf gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz
$ rm -f gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu.tar.xz
$ export PATH=`pwd`/gcc-arm-11.2-2022.02-x86_64-aarch64-none-linux-gnu/
bin:$PATH
$ export ARCH=arm64
$ export CROSS_COMPILE=aarch64-none-linux-gnu-
```

```
$ export QUARTUS_ROOTDIR=~/intelFPGA_pro/24.3/quartus/
$ export PATH=$QUARTUS_ROOTDIR/bin:$QUARTUS_ROOTDIR/
linux64:$QUARTUS_ROOTDIR/../qsys/bin:$PATH
```

4. Follow the steps below to build the GHRD for Agilex 5 SoC:

```
$ mkdir tinyml_dev && cd tinyml_dev
$ export TOP_FOLDER=$(pwd)
$ git clone -b QPDS24.3_REL_GSRD_PR https://github.com/altera-opensource/
ghrd-socfpga
$ mv ghrd-socfpga/agilex5_soc_devkit_ghrd .
$ rm -rf ghrd-socfpga
$ cd agilex5_soc_devkit_ghrd
$ make config
$ make DEVICE=A5ED065BB32AE6SR0 HPS_EMIF_MEM_CLK_FREQ_MHZ=800
HPS_EMIF_REF_CLK_FREQ_MHZ=100 generate_from_tcl
$ make all
```

These steps generate the bitstream file that programs the board. You can start programming the board once the ATF bootloader is generated.

Related Information

- Stratix® 10 Hard Processor System Technical Reference Manual
- Hard Processor System Technical Reference Manual: Agilex™ 5 SoCs

4.2. Building the Software Design

The software development has two major parts: the bootloader and the application. Altera uses ATF as the first stage bootloader (FSBL) and second stage bootloader (SSBL). You can build the application stage through the Zephyr operating system. Therefore, you must set up and prepare the Zephyr environment before starting the application development stage.

Note: You must complete Preparing LiteRT Inference Model before continuing this chapter. The following binaries are required.

Table 5. LiteRT for Microcontrollers Binaries

Binaries	Description
`model_data.cc`	Model params stored within a C array.
`model_data.h`	Header file of `model_data.cc`.
`model_settings.h`	List of classes to decode classification results in `main.cc`.
`main.cc`	Main LiteRT application to setup LiteRT model, uploading images, classifying images and profiling.
`Multiple figure.h`	Sample of MNIST images stored within a C array.

4.2.1. Arm-Trusted-Firmware

ATF is the reference implementation of secure world running on Arm processors. It provides several services and drivers to initialize and configure the system components, making the Arm* processor system ready to run applications. You need two major software components from ATF to run Arm applications – BL2 and BL31.

BL2 runs at S-EL1 exception and performs all the needed initializations for system to boot the second stage boot. BL2 loads the following images:

1. BL31: EL3 runtime image (SSBL).
2. BL32: Secure partition manager.
3. Non-trusted firmware: BL33, U-boot

You can compile the ATF source code to generate the bootloader files. Use the bootloader to start your application from QSPI, or start via a debugger. The following sections explain each method.

4.2.1.1. Booting from QSPI Flash

Follow the steps below to generate the ATF bootloader files:

```
$ cd $TOP_FOLDER
$ git clone -b socfpga_v2.11.0 https://github.com/altera-opensource/arm-trusted-
firmware atf_tinyml
$ cd atf_tinyml
$ ARCH=arm64 CROSS_COMPILE=aarch64-none-linux-gnu- make PLAT=agilex5
SOCFPGA_BOOT_SOURCE_QSPI=1 bl2 bl31 PRELOADED_BL33_BASE=0x80100000 -j$(nproc)
```

You can find the generate bootloader files `bl2.bin` and `bl31.bin` in the following location: `$TOP_FOLDER/atf_tinyml/build/agilex5/release/`

4.2.1.2. Running through Ashling RiscFree IDE for Altera FPGAs Debugger

You can customize the ATF BL2 source code to generate a custom bl2.bin file that allows you to download the SSBL (bl31.bin) and the application manually via the Ashling RiscFree debugger tool

Follow these steps:

1. Download the ATF source code.

   ```
   $ cd $TOP_FOLDER
   $ git clone -b socfpga_v2.11.0 https://github.com/altera-opensource/arm-
   trusted-firmware atf_tinyml
   ```

2. Customize the ATF source code.

   BL2 must be customized in a way that allows the developer to debug the application or use the debugger tools to manually download the SSBL and the application binary files. Without this customization, you can generate the default BL2 version which expects the BL31 and the application to stored in QSPI flash (as shown in Booting from QSPI Flash).
   To generate a BL2 version that can be debugged, follow the steps below:

   a. Navigate to bl2_plat_setup.c file

      ```
      $ cd $TOP_FOLDER/atf_tinyml/plat/intel/soc/agilex5/
      ```

   b. Before the end of ***switch (boot_source)*** function, add the following codes:

      ```
      …
          NOTICE("%s, %s: BL2: Dummy BP for loading next images \n", __DATE__,
      __TIME__);
          mmio_write_32(0x10D12224, BIT(1));
          while (mmio_read_32(0x10D12224) != 0);
      …
      ```

c. Save the `bl2_plat_setup.c` file.

d. Compile the ATF to generate Bl2 and BL31 binary files using the command below.

```
$ cd $TOP_FOLDER/atf_tinyml
$ ARCH=arm64 CROSS_COMPILE=aarch64-none-linux-gnu- make -j$(nproc)
PLAT=agilex5 bl2 bl31 SOCFPGA_BOOT_SOURCE_QSPI=1 DEBUG=1 LOG_LEVEL=50
PRELOADED_BL33_BASE=0x80100000
```

Where:

DEBUG=1 isto enable the debug build.

LOG_LEVEL=50 is to print some useful debug messages.

After that, you can navigate the build directory to find the required files, as below:

```
$ cd $TOP_FOLDER/atf_tinyml/build/agilex5/debug/
```

The above flow generates the required BL2 and BL31 images. These files are used for debug purposes; the BL2 is loaded once the HPS .rbf file is loaded into the board, and BL31 is loaded into the board via Ashling RiscFree debugger tool.

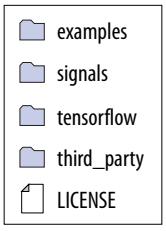
4.2.2. Building LiteRT for Microcontrollers Static Library

You must build LiteRT for Microcontroller libraries in Linux environments. If you are using Windows OS, you can invoke Windows* Subsystem for Linux* (WSL). Run the following commands:

```
$ sudo apt update
$ sudo apt install make git python3 unzip python3-pip
$ pip3 install numpy Image
$ cd $TOP_FOLDER
$ git clone --depth 1 https://github.com/tensorflow/tflite-micro.git
$ cd tflite-micro
$ python3 tensorflow/lite/micro/tools/project_generation/create_tflm_tree.py\
 -e hello_world tflite_app
```

Note: It is mandatory to change Directory command (`cd`) to tflite-micro. The execution of `create_tflm_tree.py` Python script fails when it is executed from different directory.

Figure 19. Content of the `tflite_app` Folder



Altera recommends to review the Hello World TinyML example in the `examples` folder. It is a prediction model on the sine function. For more information, please refer to LiteRT – Hello World Example and LiteRT - Get started with microcontrollers.

4.2.3. Zephyr OS Environment Setup

The example in this document uses Zephyr OS to run the LiteRT for Microcontrollers application. However, the application and operating system requirements are not dependent on each other.

- This example selects Zephyr OS for better user experience with the software runtime environment setup, Zephyr OS configuration system, and built-in LiteRT for Microcontrollers static libraries.
- You can refer to the Nios V processor build flow to build the LiteRT for Microcontrollers application and static libraries in a bare-metal environment.

To prepare the Zephyr OS SDK, begin by installing all the necessary tools and dependencies.

Related Information

Developing with Zephyr - Getting Started Guide

4.2.4. Building the LiteRT TinyML Application

To build a Zephyr application, you first need a Zephyr project. Once you've installed it, you'll need to make several modifications to enable the built-in LiteRT for Microcontrollers libraries in the Zephyr OS environment.

Follow these steps to integrate LiteRT for Microcontrollers platform using Zephyr and Agilex 5 HPS:

1. Device tree modifications: By default, the Agilex 5 HPS device tree RAM size is 8 MBytes, which is insufficient for a common tinyML application. Increase the RAM size to 1024 MBytes.
   a. Navigate to the device tree file `intel_socfpga_agilex5.dtsi` located in the following path.

   ```
   <install_Dir>/zephyrproject/zephyr/dts/arm64/intel/
   ```

   b. Modify the RAM size

   ```
       mem0: memory@80100000 {
           device_type = "memory";
           reg = <0x80100000 DT_SIZE_M(1024)>;
   ```

2. Peripheral integration: The device tree file selects the UART controller as the standard UART device by default. Keep the default UART device setup.
3. Enable LiteRT for Microcontrollers static library: To enable the static library within the Zephyr project, apply the following commands:

   ```
   $ west config manifest.project-filter -- +tflite-micro
   $ west config manifest.group-filter -- +optional
   $ west update
   ```

4. Use the generated LiteRT for Microcontrollers static library: Replace the library files downloaded by Zephyr OS environment with the ones created in Building LiteRT for Microcontrollers Static Library.
   a. Navigate to the location of the downloaded LiteRT library files by Zephyr OS environment.

   ```
   <Install_Dir>/zephyrproject/optional/modules/lib/tflite-micro
   ```

b. Replace the following directories with its files:
   - signal
   - tensorflow
   - third_party

5. Customize timing functions: The Agilex 5 HPS features a 64-bit Arm Cortex-A architecture with 64-bit timers. In contrast, the LiteRT for Microcontroller library is based on a 32-bit platform with a 32-bit timer. To prevent integer overflow, enhance the LiteRT for Microcontroller timing functions to read a 64-bit input from Arm's 64-bit timer. Navigate to `<zephyrproject_dir>/optional/modules/lib/tflite-micro/tensorflow/lite/micro`. Replace the following timing function:

   a. In `micro_time.cc`:

```
#define CLOCK_TICKS_PER_SEC 400000000
…
uint32_t ticks_per_second() { return (uint32_t)(CLOCK_TICKS_PER_SEC); }
…
uint64_t GetCurrentTimeTicks() { return (uint64_t)
(arch_k_cycle_get_64());}
```

   b. In `micro_time.h`:

```
uint64_t GetCurrentTimeTicks();
…
inline uint64_t TicksToMs(int64_t ticks)
{
  return static_cast<uint64_t>(1000.0f *static_cast<float>(ticks) /
static_cast<float>(ticks_per_second()));
}
```

   c. In `micro_profiler.h`, you must change all the 32-bit tick variables to 64-bit size.

```
uint64_t start_ticks_[kMaxEvents];
uint64_t end_ticks_[kMaxEvents];
…
struct TicksPerTag {
    const char* tag;
    uint64_t ticks;
  };
```

   In `micro_profiler.cc`, any variable that applies the above 32-bit ticks variables must be changed to a 64-bit size. With that, a newly modified TFLite library and Zephyr OS are compatible with the Agilex 5 HPS processor.

6. Create the Application Project: Inside Zephyr directory, navigate to:

```
<Install_Dir>/zephyrproject/zephyr/samples/modules/tflite-micro
```

   Create a new directory `tinyml_mnist`. Then, create the following files inside the `tinyml_mnist` main directory:

   a. `CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.20.0)

find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(tensorflow_hello_world)

set(NO_THREADSAFE_STATICS $<TARGET_PROPERTY:compiler-
cpp,no_threadsafe_statics>)
zephyr_compile_options($<$<COMPILE_LANGUAGE:CXX>:$
```

```
{NO_THREADSAFE_STATICS}>)

target_sources(app PRIVATE src/main.cc src/model/model_data.cc)
```

b. `sample.yaml`

```
sample:
  description: LiteRT app sample
  name: litert hps
common:
  tags: tensorflow
  modules:
    - tflite-micro
```

c. `prj.conf`

```
CONFIG_CPP=y
CONFIG_STD_CPP17=y
CONFIG_REQUIRES_FULL_LIBC=y
CONFIG_POSIX_API=y
CONFIG_TENSORFLOW_LITE_MICRO=y
CONFIG_STACK_USAGE=y
CONFIG_DEBUG=y
CONFIG_MAIN_STACK_SIZE=262144
CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE=262144
CONFIG_HEAP_MEM_POOL_SIZE=262144
CONFIG_THREAD_MONITOR=y
CONFIG_THREAD_STACK_INFO=y
CONFIG_THREAD_NAME=y
CONFIG_TIMER_READS_ITS_FREQUENCY_AT_RUNTIME=y
CONFIG_SYS_CLOCK_HW_CYCLES_PER_SEC=400000000
CONFIG_SYS_CLOCK_TICKS_PER_SEC=400000000
CONFIG_ARM_ARCH_TIMER=y
CONFIG_TIMING_FUNCTIONS=y
```

Note that the stack and heap size are being increased to adapt the requirements to run LiteRT application using `CONFIG_MAIN_STACK_SIZE`, `CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE`, and `CONFIG_HEAP_MEM_POOL_SIZE`.

7. Toolchain setup, compiler flags, and linker setup: This build uses the default settings of a Zephyr project. The generated application has -O0 settings to enable debugging. Compiling a release version can be enabled using CONFIG_SPEED_OPTIMIZATIONS for the -O2 compiler setting or CONFIG_SIZE_OPTIMIZATIONS for -Os in prj.conf file.
8. Build the application: Inside the `tinyml_mnist` directory, create two directories named `image` and `model`, and upload the `main.cc` (Example in Appendix). In `image` folder, store the MNIST C array samples, and in `model` folder, store the `model_data.cc`, `model_data.h` and `model_settings.h` source codes generated in chapter 2.

   You can start building the application using the following command:

   - ```
     $ west build -b intel_socfpga_agilex5_socdk samples/modules/tflite-
     micro/tinyml_mnist/ -d agilex5_mnist -p
     ```

   Where `-b` refers to the board used for this build, and `-d` is used to create the build directory with all the generated files. Inside that directory, there is a sub-directory named `zephyr`, where the `zephyr.bin` and `zephyr.elf` files reside.
     ```

5. Programming and Running

5.1. Nios V Processor System

Use the Quartus Prime Programmer tool and the Ashling RiscFree IDE for Altera FPGAs to program the Nios V processor-based system (hardware and software system respectively) into the FPGA and to run your application.

Once you successfully program both the hardware SOF and software ELF files, the application begins executing the TinyML application. Open the JTAG UART terminal to display the print log through the JTAG UART interface.

```
$ juart-terminal
```

Figure 20. Example Print Logs Part 1 - Setting up TinyML

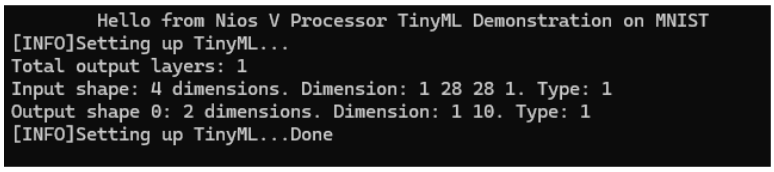


Figure 21. Example Print Logs Part 2 - Uploading Image

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Figure 22. Example Print Logs Part 3 - Classifying Image

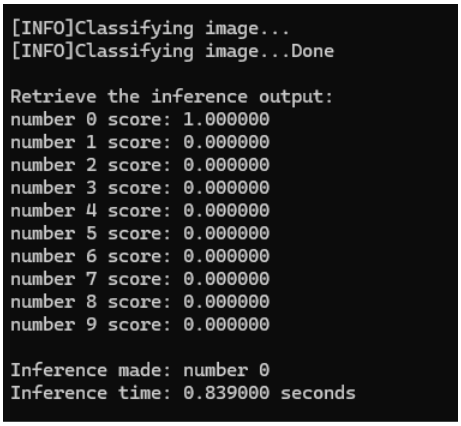
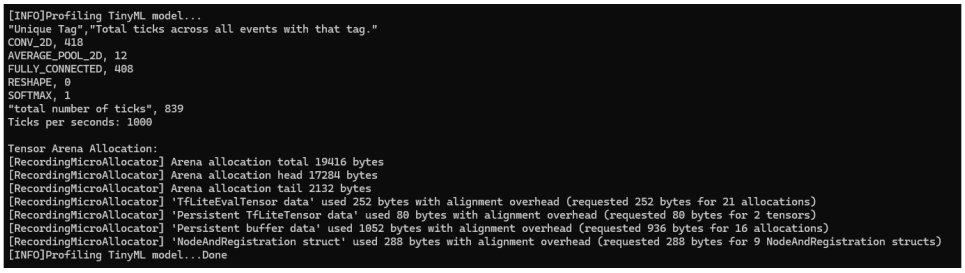


Figure 23. Example Print Logs Part 4 - Profiling TinyML Model



5.2. Arm HPS Processor System

You can run the application on HPS either through QSPI flash boot or using the Ashling RiscFree IDE for Altera FPGAs debugger tool. Before starting any process, ensure you have the necessary binary files ready to program the QSPI or run via the debugger. Refer to the following steps:

1. Create a new directory. Name it `tinyml_bins`.

```
$ cd $TOP_FOLDER
$ mkdir tinyml_bins
```

2. Generate the fiptool from ATF repository downloaded in section Arm-Trusted-Firmware. Use this tool to create the binary file that combines the SSBL (`bl31.bin`) and Zephyr application (`zephyr.bin`).

```
$ cd $TOP_FOLDER/atf_tinyml
$ ARCH=arm64 CROSS_COMPILE=aarch64-none-linux-gnu- make fiptool
$ cp tools/fiptool/fiptool $TOP_FOLDER/tinyml_bins
```

3. Copy the bootloader files to the `tinyml_bins`.

```
$ cd $TOP_FOLDER/atf_tinyml/build/agilex5/<release_or_debug>
$ cp bl2.bin bl31 $TOP_FOLDER/tinyml_bins
```

4. Copy the generated `.sof` file from the GHRD according to the steps in the topic GHRD Hardware Design.

```
$ cd $TOP_FOLDER/agilex5_soc_devkit/
$ cp ghrd_a5ed065bb32ae6sr0.sof $TOP_FOLDER/tinyml_bins
```

5.2.1. Booting from QSPI

Using this process, you can generate a QSPI image file that includes the configuration bitstream along with an embedded FSBL, including the firmware file that combines SSBL (bl31.bin) and the zephyr application.

1. Create the `fip.bin` image that contains both the SSBL and the Zephyr application.

```
$ cd $TOP_FOLDER/tinyml_bins
$ ./fiptool create --soc-fw bl31.bin --nt-fw zephyr.bin fip.bin
```

2. Download the QSPI `.pfg` file to create the `.jic` file.

```
$ wget https://releases.rocketboards.org/2024.11/zephyr/agilex5/hps_zephyr/
hello_world/
qspi_boot/qspi_flash_image_agilex5_boot.pfg
```

3. Create the `.jic` file.

```
quartus_pfg -c qspi_flash_image_agilex5_boot.pfg
```

4. Set MSEL to JTAG (OFF-OFF-OFF-OFF).
5. Turn on the Device.
6. Program the QSPI flash.

```
quartus_pgm -c 1 -m jtag -o "pvi;qspi_image.jic"
```

7. Set MSEL to QSPI (OFF-ON-ON-OFF).
8. Power-on the Device.
9. Inspect the HPS terminal.

5.2.2. Booting via the Debugger

Use the Quartus Prime Programmer tool and the Ashling RiscFree IDE for Altera FPGAs to program the Arm HPS processor-based system (hardware and software system respectively) into the FPGA and to run your application.

1. Make sure you set MSEL dipswitch SW27 to JTAG: (OFF-OFF-OFF-OFF). Then connect the HPS UART to a terminal on your machine, like Putty or MobaXterm on Windows, or Minicom on Linux.
2. Add the BL2 FSBL to the generated GHRD `.sof` file. First, convert `bl2.bin` file into a hex format, then use quartus_pfg to embed the hex format of BL2 with `ghrd .sof` file, and create the `.rbf` format file per the following steps:

```
$ cd $TOP_FOLDER/tinyml_bins
$ aarch64-none-linux-gnu-objcopy -v -I binary -O ihex --change-addresses 0x0
bl2.bin  bl2.hex
$ quartus_pfg -c ghrd_a5ed065bb32ae6sr0.sof ghrd.rbf -o hps=1 -o
hps_path=bl2.hex
```

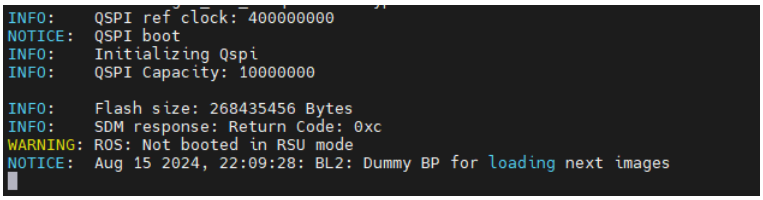
The `quartus_pfg` command converts the `.sof` file into two `.rbf` files: the core and the hps files. Use the core file to configure the FPGA fabric, while the hps file to configure the HPS IO.

3. Use the `-o` hps_path argument to guide the tool to the location of the FSBL and embed it with the `ghrd.hps.rbf` file. Ashling RiscFree IDE for Altera FPGAs debugger tool loads the `bl31.bin` file to the board. To run the application on board, program the device with the following generated `.rbf` bitstream:

```
quartus_pgm -c 1 -m jtag -o "p;ghrd.hps.rbf"
```

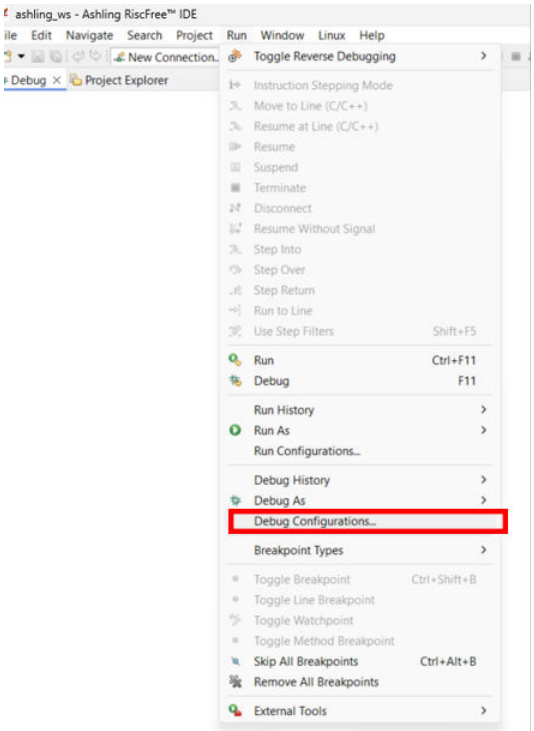
Once the process is complete, you can see the FSBL boot on the HPS terminal. The last message indicates that FSBL was booted successfully, and it is waiting for SSBL to load using the debugger tool.

Figure 24. BL2 Boot on HPS Terminal (Debugger Flow)



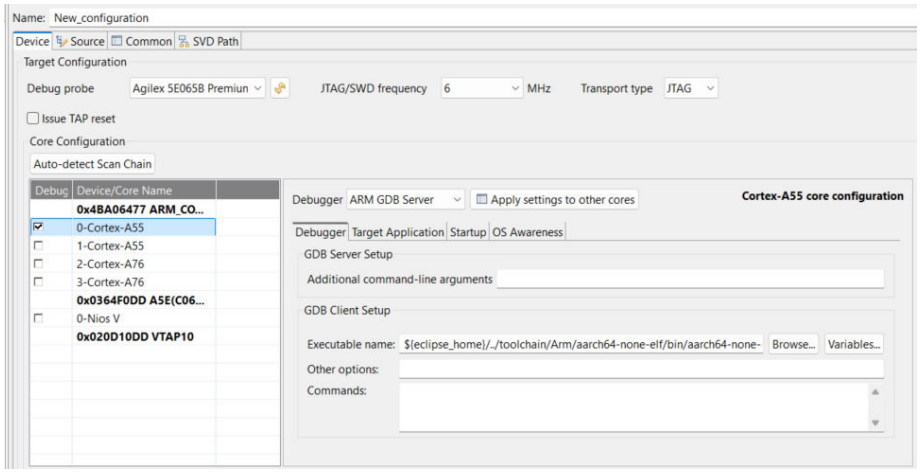
4. Next, open Ashling RiscFree IDE for Altera FPGAs tool, set your workspace directory, and copy the required binary files generated earlier: `bl31.bin`, `zephyr.bin`, and `zephyr.elf` (if debug is needed). From **Run** menu, go to **Debug Configuration**.

Figure 25. Ashling RiscFree IDE for Altera FPGAs Menus



5. Select **Ashling Heterogeneous Multicore Hardware Debugging**, then configure the ARM Coresight SOC core 0.

Figure 26. Heterogeneous Multicore Hardware Debugging Configuration 1



6. Go to **Target Application** tab:
   - Add the `.elf` file if you need the debug information. Otherwise, you can run the application only.
   - Keep **Load image** unchecked (because you need to load it later manually).
   - Check **Load symbols** (if the .elf file is loaded for debugging).
7. In **Startup** tab, keep everything to default. In **OS Awareness** tab, turn on **Enable OS Aware Debugging** and select Zephyr OS if you want to debug your application. You can ignore this step if you want to run the application only without debugging. Once done, click **Apply ➤ Debug**. Your system should be ready to receive the binary files via Debugger Console.

Figure 27. Heterogeneous Multicore Hardware Debugging Configuration 3

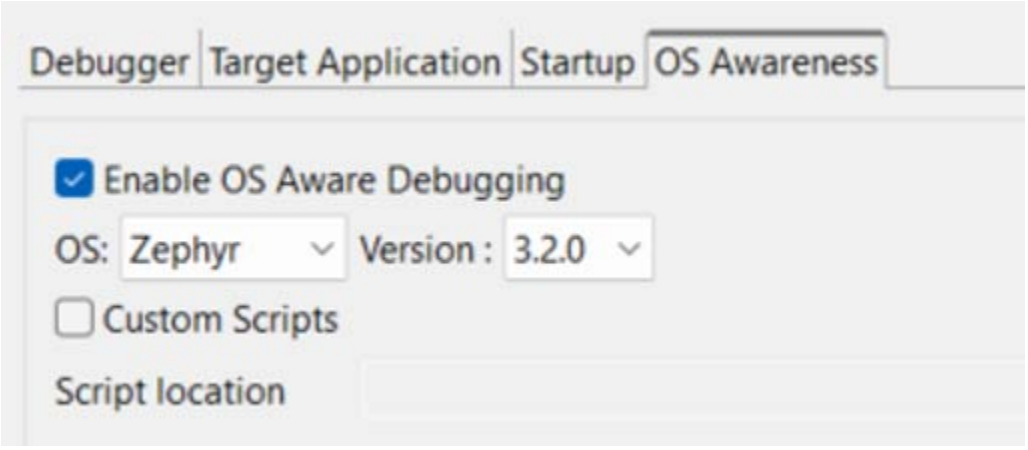
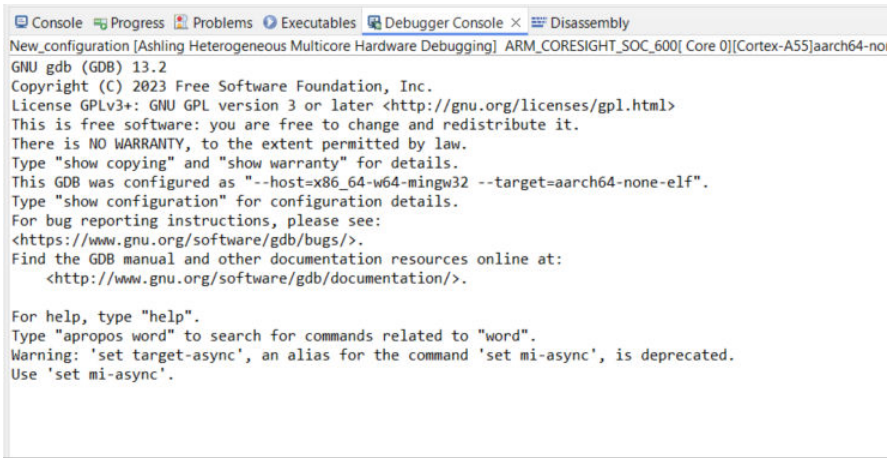


Figure 28. Ashling RiscFree Debugger Console



The commands required in Debugger Console:

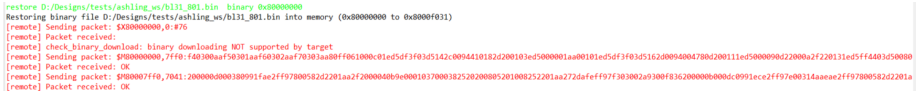
```
The needed commands in Debugger Console can be seen below:

set debug remote 1
restore <Absoulte_Path_to_your_Workspace>/bl31_801.bin  binary 0x80000000
restore <Absoulte_Path_to_your_Workspace>/zephyr.bin  binary 0x80100000
set $x1=0

The set debug remote 1 command is used to enable message transaction
acknowledgement
from the board to indicate the download completion of a binary file.
A sample screenshot can be seen below.
```

Use the **set debug remote 1** command to enable message transaction acknowledgement from the board to indicate the download completion of a binary file. Refer to the following example.

Figure 29. Ashling RiscFree Debugger Console Output



8. Use the restore command to load the binary file to its destination memory. In this example, the SSBL and Zephyr applications are loaded into HPS DDR memory. Make sure to use the same addresses in the commands above. Refer to the following sample screenshots of the application execution from the HPS UART terminal.

Figure 30. Example Print Logs Part 1 - Setting up Zephyr and TinyML

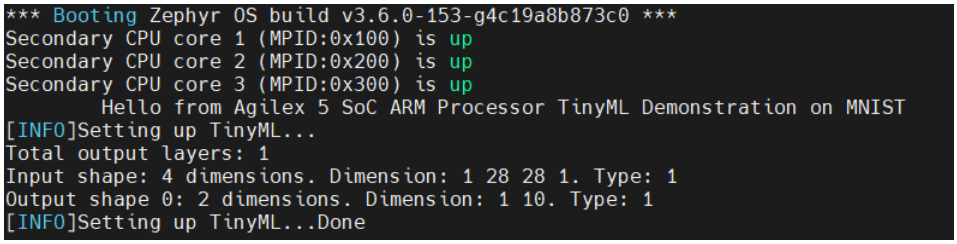


Figure 31. Example Print Logs Part 1 - Setting up Zephyr and TinyML

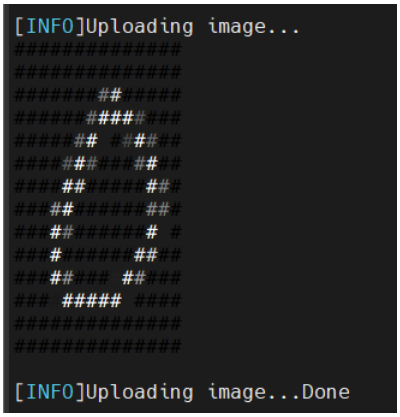


Figure 32. Example Print Logs Part 3 - Classifying Image

```
*** Booting Zephyr OS build v3.6.0-153-g4c19a8b873c0 ***
Secondary CPU core 1 (MPID:0x100) is up
Secondary CPU core 2 (MPID:0x200) is up
Secondary CPU core 3 (MPID:0x300) is up
        Hello from Agilex 5 SoC ARM Processor TinyML Demonstration on MNIST
[INFO]Setting up TinyML...
Total output layers: 1
Input shape: 4 dimensions. Dimension: 1 28 28 1. Type: 1
Output shape 0: 2 dimensions. Dimension: 1 10. Type: 1
[INFO]Setting up TinyML...Done
```

Figure 33. Example Print Logs Part 4 - Profiling TinyML model

```
[INFO]Profiling TinyML model...
"Unique Tag","Total ticks across all events with that tag."
CONV_2D, 6294619
AVERAGE_POOL_2D, 141921
FULLY_CONNECTED, 1255982
RESHAPE, 14279
SOFTMAX, 4578
"total number of ticks", 7711379
Detailed Profiling

0 - CONV_2D took 4116753 ticks (53.00%) --- (10.00 ms).
1 - AVERAGE_POOL_2D took 127688 ticks (1.00%) --- (0.00 ms).
2 - CONV_2D took 2177866 ticks (28.00%) --- (5.00 ms).
3 - AVERAGE_POOL_2D took 14233 ticks (0.00%) --- (0.00 ms).
4 - FULLY_CONNECTED took 104922 ticks (1.00%) --- (0.00 ms).
5 - RESHAPE took 14279 ticks (0.00%) --- (0.00 ms).
6 - FULLY_CONNECTED took 1143211 ticks (14.00%) --- (2.00 ms).
7 - FULLY_CONNECTED took 7849 ticks (0.00%) --- (0.00 ms).
8 - SOFTMAX took 4578 ticks (0.00%) --- (0.00 ms).
"total number of ticks", 7711379
Ticks per seconds: 400000000

Inference Time: 0.019286 sec

Tensor Arena Allocation:
[RecordingMicroAllocator] Arena allocation total 20752 bytes
[RecordingMicroAllocator] Arena allocation head 17288 bytes
[RecordingMicroAllocator] Arena allocation tail 3464 bytes
[RecordingMicroAllocator] 'TfLiteEvalTensor data' used 504 bytes with alignment overhead (requested 504 bytes for 21 allocations)
[RecordingMicroAllocator] 'Persistent TfLiteTensor data' used 128 bytes with alignment overhead (requested 128 bytes for 2 tensors)
[RecordingMicroAllocator] 'Persistent buffer data' used 1576 bytes with alignment overhead (requested 1484 bytes for 16 allocations)
[RecordingMicroAllocator] 'NodeAndRegistration struct' used 576 bytes with alignment overhead (requested 576 bytes for 9 NodeAndRegistration structs)
[INFO]Profiling TinyML model...Done
```

6. Appendix

6.1. main.cc for Nios V Processor

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <math.h>
#include "system.h"
#include <time.h>
#include <unistd.h>

//Import TensorFlow lite libraries
#include "tensorflow/lite/core/c/common.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/micro_log.h"
#include "tensorflow/lite/micro/micro_time.h"
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/micro/micro_profiler.h"
#include "tensorflow/lite/micro/recording_micro_interpreter.h"
#include "tensorflow/lite/micro/system_setup.h"
#include "tensorflow/lite/schema/schema_generated.h"

//Model data
#include "model/model_data.h"
#include "model/model_settings.h"

////Sample image
#include "image/figure-0.h"
#include "image/figure-1.h"
#include "image/figure-2.h"
#include "image/figure-3.h"
#include "image/figure-4.h"
#include "image/figure-5.h"
#include "image/figure-6.h"
#include "image/figure-7.h"
#include "image/figure-8.h"
#include "image/figure-9.h"

#define IMAGE_WIDTH 28
#define IMAGE_HEIGHT 28
#define NUM_CHANNELS 1
#define IMAGE_SIZE (IMAGE_WIDTH * IMAGE_HEIGHT * NUM_CHANNELS)

//ANSI Escape code
#define CRESET "\033[m"

int count;
uint8_t test_image[IMAGE_HEIGHT][IMAGE_WIDTH][NUM_CHANNELS];


namespace {
    const tflite::Model* model = nullptr;

    using OpResolver = tflite::MicroMutableOpResolver<5>;
    TfLiteStatus RegisterOps(OpResolver& op_resolver) {
        TF_LITE_ENSURE_STATUS(op_resolver.AddConv2D());
        TF_LITE_ENSURE_STATUS(op_resolver.AddAveragePool2D());
        TF_LITE_ENSURE_STATUS(op_resolver.AddReshape());
```

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

```
        TF_LITE_ENSURE_STATUS(op_resolver.AddFullyConnected());
        TF_LITE_ENSURE_STATUS(op_resolver.AddSoftmax());
        return kTfLiteOk;
    }
}  // namespace

TfLiteStatus LoadModelandInference() {

    printf("[INFO]Setting up TinyML...\n\r");
    tflite::MicroProfiler profiler;
    OpResolver op_resolver;
    TF_LITE_ENSURE_STATUS(RegisterOps(op_resolver));
    model = tflite::GetModel(lenet_tflite);
    TFLITE_CHECK_EQ(model->version(), TFLITE_SCHEMA_VERSION);

    /* Arena size is a round number, which is determined
     * using the RecordingMicroInterpreter.
     */
    constexpr int kTensorArenaSize = 20000;
    uint8_t tensor_arena[kTensorArenaSize];
    constexpr int kNumResourceVariables = 24;

    tflite::RecordingMicroAllocator* allocator(
      tflite::RecordingMicroAllocator::Create(tensor_arena, kTensorArenaSize));
    tflite::RecordingMicroInterpreter Recordinginterpreter(
      model, op_resolver, allocator,
      tflite::MicroResourceVariables::Create(allocator, kNumResourceVariables),
      &profiler);

    TF_LITE_ENSURE_STATUS(Recordinginterpreter.AllocateTensors());
    TFLITE_CHECK_EQ(Recordinginterpreter.inputs_size(), 1);

    //Print loaded model input and output shape
    printf("Total output layers: %d\n\r",
Recordinginterpreter.outputs_size());    printf("Input shape: %d dimensions.
Dimension: %d %d %d %d. Type: %d\n\r",
      Recordinginterpreter.input(0)->dims->size,
      Recordinginterpreter.input(0)->dims->data[0],
      Recordinginterpreter.input(0)->dims->data[1],
      Recordinginterpreter.input(0)->dims->data[2],
      Recordinginterpreter.input(0)->dims->data[3],
      Recordinginterpreter.input(0)->type
    );

    for (int i = 0; i < (int)(Recordinginterpreter.outputs_size()); ++i)
      printf( "Output shape %d: %d dimensions. Dimension: %d %d. Type: %d\n\r",
        i,
        Recordinginterpreter.output(i)->dims->size,
        Recordinginterpreter.output(i)->dims->data[0],
        Recordinginterpreter.output(i)->dims->data[1],
        Recordinginterpreter.output(i)->type
      );

    printf("[INFO]Setting up TinyML...Done\n\n\r");

    printf("[INFO]Uploading image...\n\r");

    //Visualize sample image in terminal
    int HashTag = 35;
    for (int i = 0; i < IMAGE_HEIGHT; i=i+2){
        for (int j = 0; j < IMAGE_WIDTH; j=j+2){
            printf("\033[38;2;%d;%d;%dm%c"
                    CRESET,
                    test_image[i][j][0],
                    test_image[i][j][0],
                    test_image[i][j][0],
                    HashTag);
        }
        printf("\n");
    }
    printf("\n");
```

```
    //Input sample image to tflite model input.
    int len = 0;
    for (int i = 0; i < IMAGE_HEIGHT; i++){
        for (int j = 0; j < IMAGE_WIDTH; j++){
            for (int k = 0; k < NUM_CHANNELS; k++){
                Recordinginterpreter.input(0)->data.f[len] =
                float(test_image[i][j][k])/255.0;
                len++;
            }
        }
    }
    printf("[INFO]Uploading image...Done\n\n\r");



    printf("[INFO]Classifying image...\n\r");
    clock_t startTime = clock();
    TF_LITE_ENSURE_STATUS(Recordinginterpreter.Invoke());
    clock_t endTime = clock();
    printf("[INFO]Classifying image...Done\n\n\r");

    //Retrieve inference output
    int answer = 0;
    printf("Retrieve the inference output:\n");
    for (int i = 0; i < kCategoryCount; ++i){
      printf("%s score: %f\n\r",
            kCategoryLabels[i],
            Recordinginterpreter.output(0)->data.f[i]);
      if (Recordinginterpreter.output(0)->data.f[i] >
            Recordinginterpreter.output(0)->data.f[answer]) answer = i;
      }
    printf("\nInference made: %s\n\r", kCategoryLabels[answer]);
    int time_spent = int(endTime - startTime) / ALT_CPU_TICKS_PER_SEC;
    printf("Inference time: %d seconds\n\n\r", time_spent);


    printf("[INFO]Profiling TinyML model...\n\r");
    profiler.LogTicksPerTagCsv();
    printf("Ticks per seconds: %d\n\n", ALT_CPU_TICKS_PER_SEC);

    printf("Tensor Arena Allocation:\n");
    Recordinginterpreter.GetMicroAllocator().PrintAllocations();
    printf("[INFO]Profiling TinyML model...Done\n\n\r");

  return kTfLiteOk;
}

int SelectImage(uint8_t (*image)[IMAGE_WIDTH][NUM_CHANNELS]){
    for (int i = 0; i < IMAGE_HEIGHT; i++){
           for (int j = 0; j < IMAGE_WIDTH; j++){
               for (int k = 0; k < NUM_CHANNELS; k++){
                   test_image[i][j][k] = image[i][j][k];
               }
           }
    }
    TF_LITE_ENSURE_STATUS(LoadModelandInference());

    return 0;
}


int main() {

   printf("\tHello from Nios V Processor TinyML Demonstration on MNIST\n\r");
   /************************TFLITE-MICRO TINYML*****************/

   SelectImage(test_image0);
   SelectImage(test_image1);
   SelectImage(test_image2);
   SelectImage(test_image3);
```

```
   SelectImage(test_image4);
   SelectImage(test_image5);
   SelectImage(test_image6);
   SelectImage(test_image7);
   SelectImage(test_image8);
   SelectImage(test_image9);

   return 0;
}
```

6.2. main.cc for Arm HPS Processor

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>

#include <zephyr/kernel.h>

//Import TensorFlow lite libraries
#include "tensorflow/lite/core/c/common.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/micro_log.h"
#include "tensorflow/lite/micro/micro_time.h"
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/micro/micro_profiler.h"
#include "tensorflow/lite/micro/recording_micro_interpreter.h"
#include "tensorflow/lite/micro/system_setup.h"
#include "tensorflow/lite/schema/schema_generated.h"

//Model data
#include " model/model_data.h"
#include " model/model_settings.h"

////Sample image
#include "image/figure-0.h"
#include "image/figure-1.h"
#include "image/figure-2.h"
#include "image/figure-3.h"
#include "image/figure-4.h"
#include "image/figure-5.h"
#include "image/figure-6.h"
#include "image/figure-7.h"
#include "image/figure-8.h"
#include "image/figure-9.h"

#define IMAGE_WIDTH 28
#define IMAGE_HEIGHT 28
#define NUM_CHANNELS 1
#define IMAGE_SIZE (IMAGE_WIDTH * IMAGE_HEIGHT * NUM_CHANNELS)

//ANSI Escape code
#define CRESET "\033[m"

#define SEC_TO_MSEC               1000ul
#define CLOCK_TICKS_PER_SEC       400000000

int count;
uint8_t test_image[IMAGE_HEIGHT][IMAGE_WIDTH][NUM_CHANNELS];

namespace {
    const tflite::Model* model = nullptr;

    using OpResolver = tflite::MicroMutableOpResolver<5>;
    TfLiteStatus RegisterOps(OpResolver& op_resolver) {
        TF_LITE_ENSURE_STATUS(op_resolver.AddConv2D());
```

```
        TF_LITE_ENSURE_STATUS(op_resolver.AddAveragePool2D());
        TF_LITE_ENSURE_STATUS(op_resolver.AddReshape());
        TF_LITE_ENSURE_STATUS(op_resolver.AddFullyConnected());
        TF_LITE_ENSURE_STATUS(op_resolver.AddSoftmax());
        return kTfLiteOk;
    }
}  // namespace

TfLiteStatus LoadModelandInference() {

    printf("[INFO]Setting up TinyML...\n\r");
    tflite::MicroProfiler profiler;
    OpResolver op_resolver;
    TF_LITE_ENSURE_STATUS(RegisterOps(op_resolver));
    model = tflite::GetModel(lenet_tflite);
    TFLITE_CHECK_EQ(model->version(), TFLITE_SCHEMA_VERSION);

    /* Arena size is a round number, which is determined
     * using RecordingMicroInterpreter.
     */
    constexpr int kTensorArenaSize = 30000;
    uint8_t tensor_arena[kTensorArenaSize];
    constexpr int kNumResourceVariables = 24;

    tflite::RecordingMicroAllocator* allocator(
      tflite::RecordingMicroAllocator::Create(tensor_arena, kTensorArenaSize));
    tflite::RecordingMicroInterpreter Recordinginterpreter(
      model, op_resolver, allocator,
      tflite::MicroResourceVariables::Create(allocator, kNumResourceVariables),
      &profiler);

    TF_LITE_ENSURE_STATUS(Recordinginterpreter.AllocateTensors());
    TFLITE_CHECK_EQ(Recordinginterpreter.inputs_size(), 1);

    //Print loaded model input and output shape
    printf("Total output layers: %d\n\r", Recordinginterpreter.outputs_size());
    printf("Input shape: %d dimensions. Dimension: %d %d %d %d. Type: %d\n\r",
      Recordinginterpreter.input(0)->dims->size,
      Recordinginterpreter.input(0)->dims->data[0],
      Recordinginterpreter.input(0)->dims->data[1],
      Recordinginterpreter.input(0)->dims->data[2],
      Recordinginterpreter.input(0)->dims->data[3],
      Recordinginterpreter.input(0)->type
    );

    for (int i = 0; i < (int)(Recordinginterpreter.outputs_size()); ++i)
      printf("Output shape %d: %d dimensions. Dimension: %d %d. Type: %d\n\r",
       i,
      Recordinginterpreter.output(i)->dims->size,
      Recordinginterpreter.output(i)->dims->data[0],
      Recordinginterpreter.output(i)->dims->data[1],
      Recordinginterpreter.output(i)->type
      );

    printf("[INFO]Setting up TinyML...Done\n\n\r");

    printf("[INFO]Uploading image...\n\r");

    //Visualize sample image in terminal
    int HashTag = 35;
    for (int i = 0; i < IMAGE_HEIGHT; i=i+2){
        for (int j = 0; j < IMAGE_WIDTH; j=j+2){
            printf("\033[38;2;%d;%d;%dm%c"
                    CRESET,
                    test_image[i][j][0],
                    test_image[i][j][0],
                    test_image[i][j][0],
                    HashTag);
        }
        printf("\n");
    }
```

```
    printf("\n");

    //Input sample image to tflite model input.
    int len = 0;
    for (int i = 0; i < IMAGE_HEIGHT; i++){
        for (int j = 0; j < IMAGE_WIDTH; j++){
            for (int k = 0; k < NUM_CHANNELS; k++){
                Recordinginterpreter.input(0)->data.f[len] =
                float(test_image[i][j][k])/255.0;
                len++;
            }
        }
    }
    printf("[INFO]Uploading image...Done\n\n\r");
    printf("[INFO]Classifying image...\n\r");
    int64_t  startTime, endTime =0;
    int64_t  tdelta = 0;
    startTime = sys_clock_cycle_get_64();
    TF_LITE_ENSURE_STATUS(Recordinginterpreter.Invoke());
    endTime = sys_clock_cycle_get_64();
    tdelta = endTime - startTime;
    printf("[INFO]Classifying image...Done\n\n\r");

    //Retrieve inference output
    int answer = 0;
    printf("Retrieve the inference output:\n");
    for (int i = 0; i < kCategoryCount; ++i){
      printf("%s score: %f\n\r",
          kCategoryLabels[i],
          Recordinginterpreter.output(0)->data.f[i]);
       if (Recordinginterpreter.output(0)->data.f[i]>
    Recordinginterpreter.output(0)->data.f[answer]) answer = i;
    }
    printf("\nInference made: %s\n\r", kCategoryLabels[answer]);
    printf("Ticks per seconds: %d\n\n", CLOCK_TICKS_PER_SEC);
    printf("Inference Time: %lf sec\n\n\r",
    ((double)(tdelta)) / CLOCK_TICKS_PER_SEC);
    printf("[INFO]Profiling TinyML model...\n\r");
    profiler.LogTicksPerTagCsv();
    printf("Detailed Profiling\n\n");
    profiler.Log();
    printf("Ticks per seconds: %d\n\n", CLOCK_TICKS_PER_SEC);
    printf("Inference Time: %lf sec\n\n\r",
    ((double)(tdelta)) / CLOCK_TICKS_PER_SEC);

    printf("Tensor Arena Allocation:\n");
    Recordinginterpreter.GetMicroAllocator().PrintAllocations();
    printf("[INFO]Profiling TinyML model...Done\n\n\r");

  return kTfLiteOk;
}

int SelectImage(uint8_t (*image)[IMAGE_WIDTH][NUM_CHANNELS]){
    for (int i = 0; i < IMAGE_HEIGHT; i++){
           for (int j = 0; j < IMAGE_WIDTH; j++){
               for (int k = 0; k < NUM_CHANNELS; k++){
                   test_image[i][j][k] = image[i][j][k];
               }
           }
       }
    TF_LITE_ENSURE_STATUS(LoadModelandInference());
    return 0;
}


int main() {

   printf("\tHello from Agilex 5 SoC ARM Processor TinyML Demonstration on MNIST
\n\r");
   /******************TFLITE-MICRO TINYML**********************/
```

```
    SelectImage(test_image0);
    SelectImage(test_image1);
    SelectImage(test_image2);
    SelectImage(test_image3);
    SelectImage(test_image4);
    SelectImage(test_image5);
    SelectImage(test_image6);
    SelectImage(test_image7);
    SelectImage(test_image8);
    SelectImage(test_image9);

    return 0;
}
```

7. Document Revision History for the AN 1011: TinyML Applications in Altera FPGAs Using LiteRT for Microcontrollers

Document Version	Changes
2025.04.07	Initial release.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered