iCE40 UltraPlus Hand Gesture Detection

Reference Design

FPGA-RD-02206-1.0

December 2020

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS and with all faults, and all risk associated with such information is entirely with Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
CNN	Convolutional Neural Network
FPGA	Field-Programmable Gate Array
LED	Light-emitting diode
LSB	Least Significant Bit
NN	Neural Network
ML	Machine Learning
MSB	Most Significant Bit
SPI	Serial Peripheral Interface
USB	Universal Serial Bus

1. Introduction

This document describes the hand gesture detection design process using an iCE40 UltraPlus™ FPGA platform (HiMax HM01B0 UPduino Shield v2.1).

1.1. Design Process Overview

The design process involves the following steps:

1. Training the model
    - Setting up the basic environment
    - Preparing the dataset
    - Training the machine
        - Training the machine and creating the checkpoint data
    - Creating the frozen file (*.pb)
2. Compiling Neural Network
    - Creating the filter and firmware binary files with Lattice SensAI 3.1 program
3. FPGA Design
    - Creating the FPGA bitstream file
4. FPGA Bitstream and Quantized Weights and Instructions
    - Flashing the binary and bit stream files to iCE40 UltraPlus UPduino hardware

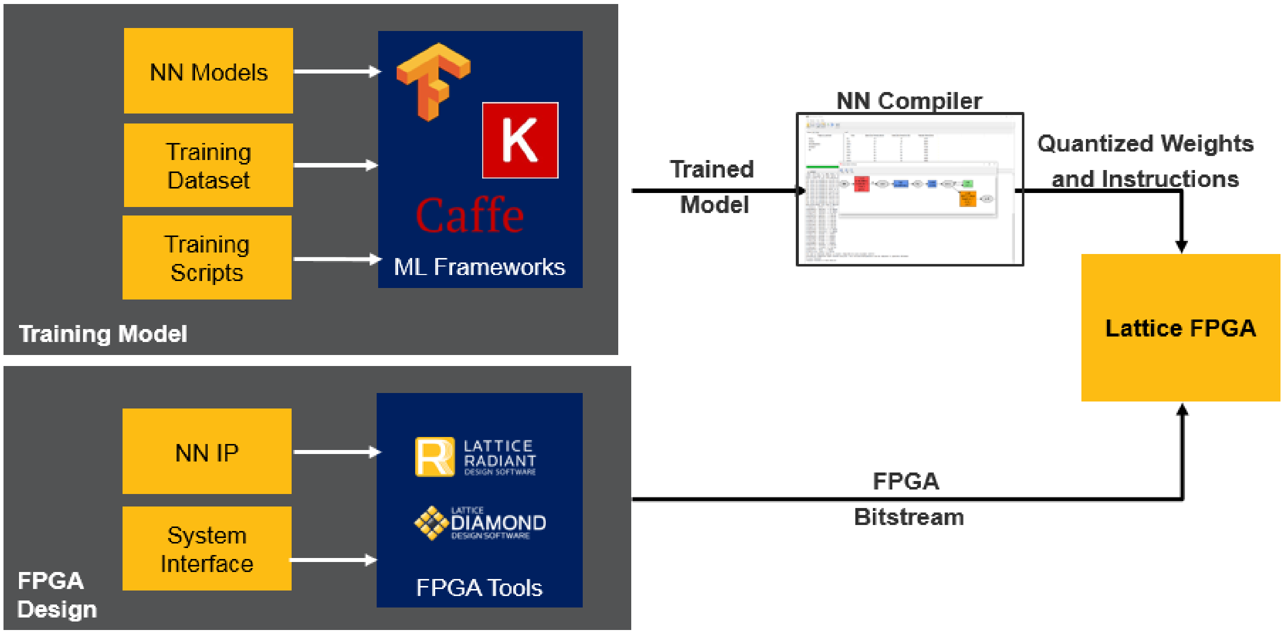


Figure 1.1. Lattice Machine Learning Design Flow

2. Setting Up the Basic Environment

2.1. Software and Hardware Requirements

This section describes the required tools and environment setup for training and model freezing.

2.1.1. Lattice Software

- Lattice Radiant™ Tool v2.2 – Refer to http://www.latticesemi.com/latticeradiant
- Lattice Radiant Programmer v2.2 – Refer to http://www.latticesemi.com/latticeradiant
- Lattice SensAI Compiler v3.1 – Refer to https://www.latticesemi.com/Products/DesignSoftwareAndIP/AIML/NeuralNetworkCompiler

2.1.2. Hardware

This design uses the HiMax HM01B0 UPduino Shield as shown in Figure 2.1. Refer to http://www.latticesemi.com/products/developmentboardsandkits/himaxhm01b0.

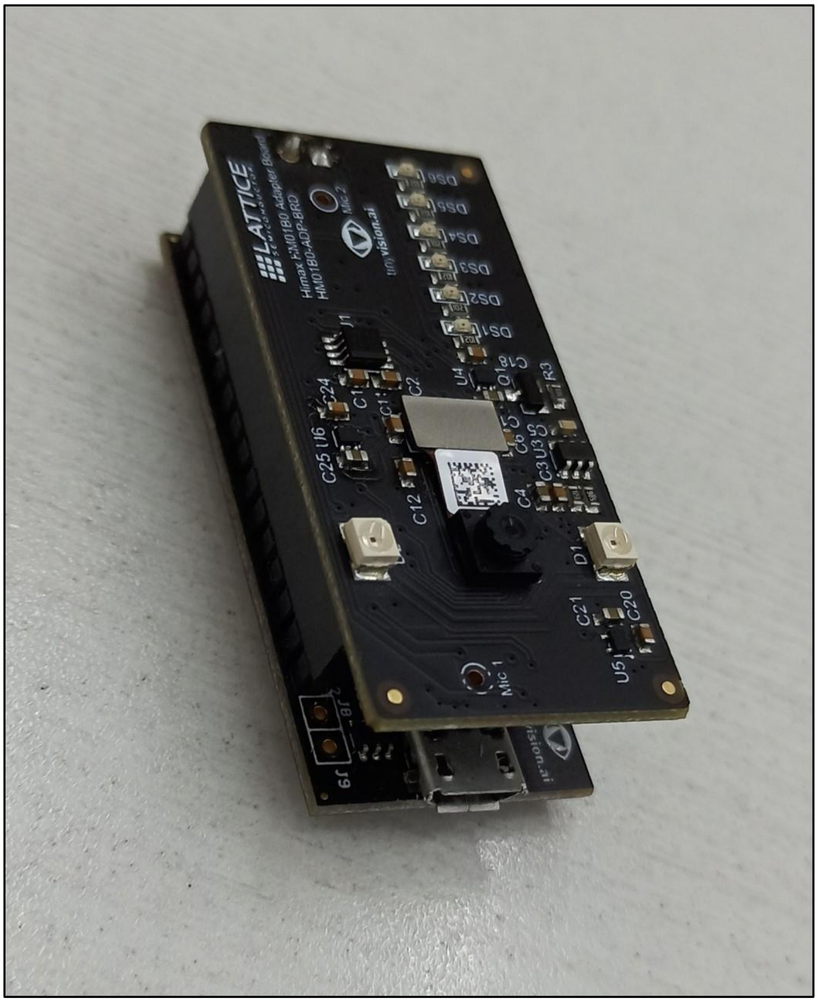


Figure 2.1. Lattice Himax HM01B0 UPduino Shield Board

Note: HiMax HM01B0 board with IR sensor is needed in order to run Hand Gesture demo.

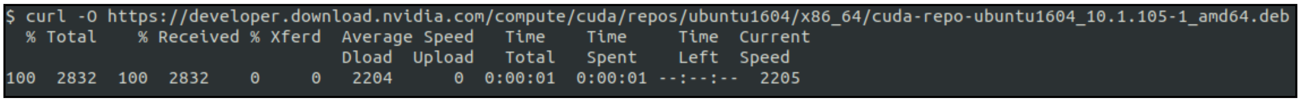
2.2. Setting Up the Linux Environment for Machine Training

This section describes the steps for NVIDIA GPU drivers and/or libraries for 64-bit Ubuntu 16.04 OS. The NVIDIA library and TensorFlow version is dependent on the PC and Ubuntu/Windows version.

2.2.1. Installing the CUDA Toolkit

To install the CUDA toolkit, run the following commands in the order specified below:

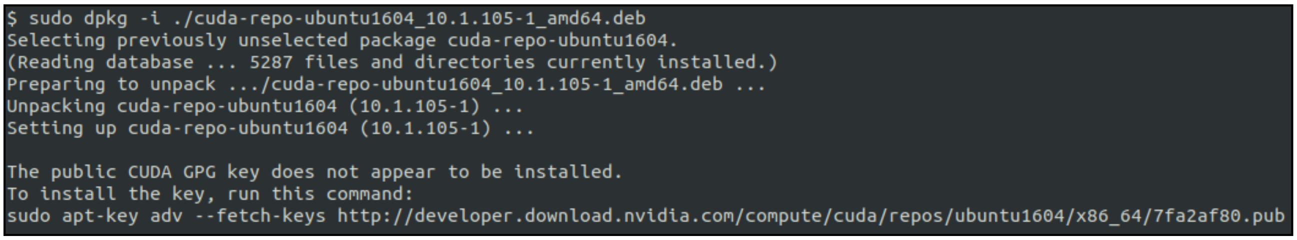
```
$ curl -O
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-
repo-ubuntu1604_10.1.105-1_amd64.deb
```



```
$ curl -O https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2832  100  2832    0     0   2204      0  0:00:01  0:00:01 --:--:--  2205
```

Figure 2.2. Download CUDA Repo

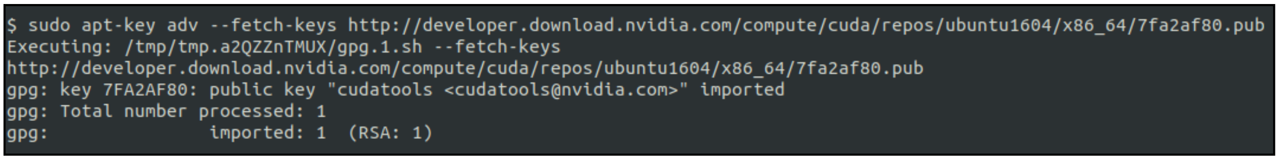
```
$ sudo dpkg –I ./cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
```



```
$ sudo dpkg -i ./cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
Selecting previously unselected package cuda-repo-ubuntu1604.
(Reading database ... 5287 files and directories currently installed.)
Preparing to unpack .../cuda-repo-ubuntu1604_10.1.105-1_amd64.deb ...
Unpacking cuda-repo-ubuntu1604 (10.1.105-1) ...
Setting up cuda-repo-ubuntu1604 (10.1.105-1) ...

The public CUDA GPG key does not appear to be installed.
To install the key, run this command:
sudo apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
```

Figure 2.3. Install CUDA Repo

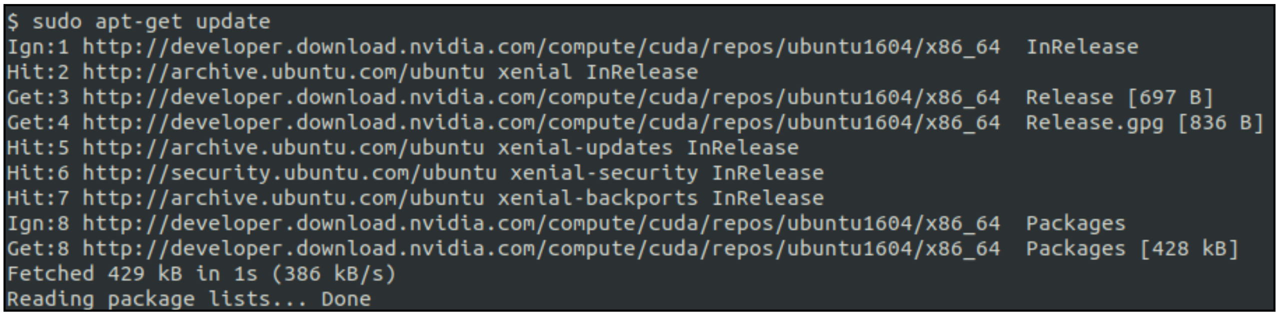
```
$ sudo apt-key adv --fetch-keys
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af
80.pub
```



```
$ sudo apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
Executing: /tmp/tmp.a2QZZnTMUX/gpg.1.sh --fetch-keys
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
gpg: key 7FA2AF80: public key "cudatools <cudatools@nvidia.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

Figure 2.4. Fetch Keys

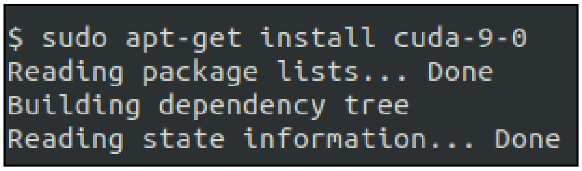
```
$sudo apt-get update
```



```
$ sudo apt-get update
Ign:1 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  InRelease
Hit:2 http://archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Release [697 B]
Get:4 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Release.gpg [836 B]
Hit:5 http://archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:6 http://security.ubuntu.com/ubuntu xenial-security InRelease
Hit:7 http://archive.ubuntu.com/ubuntu xenial-backports InRelease
Ign:8 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Packages
Get:8 http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64  Packages [428 kB]
Fetched 429 kB in 1s (386 kB/s)
Reading package lists... Done
```

Figure 2.5. Update Ubuntu Packages Repositories

```
$ sudo apt-get install cuda-9-0
```



```
$ sudo apt-get install cuda-9-0
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

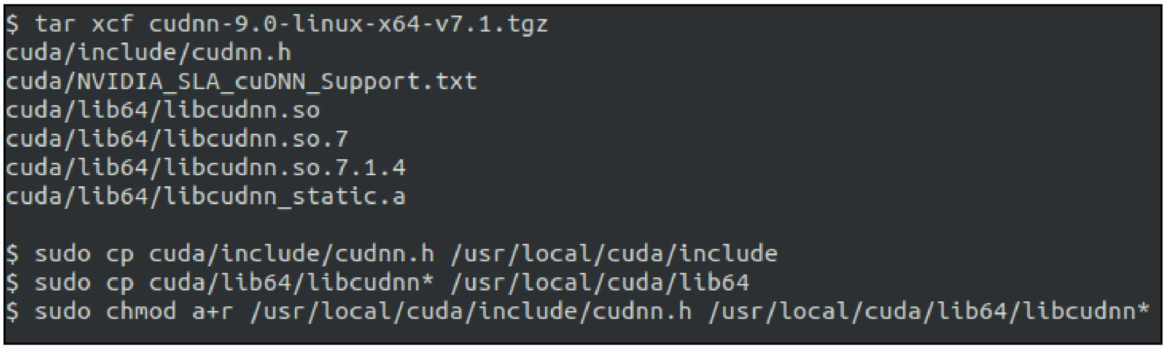
Figure 2.6. CUDA Installation

2.2.2. Installing the cuDNN

To install the cuDNN:

1. Create NVIDIA developer account in https://developer.nvidia.com.
2. Download cuDNN lib in https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v7.1.4/prod/9.0_20180516/cudnn-9.0-linux-x64-v7.1.
3. Execute the commands below to install cuDNN:

```
$ tar xvf cudnn-9.0-linux-x64-v7.1.tgz
$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
$ sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
```



```
$ tar xcf cudnn-9.0-linux-x64-v7.1.tgz
cuda/include/cudnn.h
cuda/NVIDIA_SLA_cuDNN_Support.txt
cuda/lib64/libcudnn.so
cuda/lib64/libcudnn.so.7
cuda/lib64/libcudnn.so.7.1.4
cuda/lib64/libcudnn_static.a

$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
$ sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
```

Figure 2.7. cuDNN Library Installation

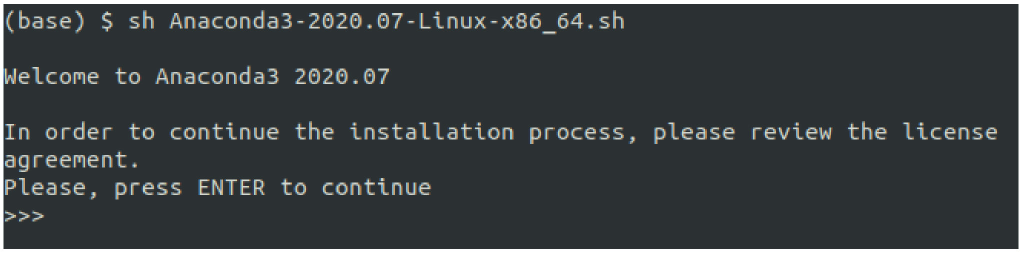
2.2.3. Installing the Anaconda and Python 3

To install the Anaconda and Python 3:

1. Go to https://www.anaconda.com/distribution/#download-section.
2. Download Python 3 version of Anaconda for Linux.
3. Install the Anaconda environment by running the command below:

```
$ sh Anaconda3-2019.03-Linux-x86_64.sh
```

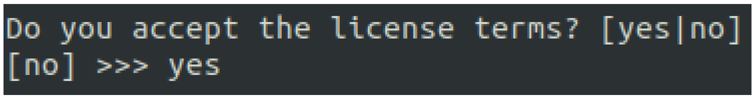
Note: Anaconda3-**<version>**-Linux-x86_64.sh, version may vary based on the release



```
(base) $ sh Anaconda3-2020.07-Linux-x86_64.sh

Welcome to Anaconda3 2020.07

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
```

Figure 2.8. Anaconda Installation

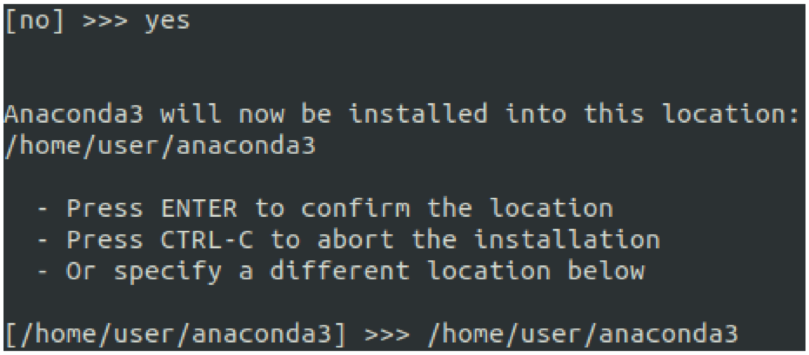
4. Accept the license.



```
Do you accept the license terms? [yes|no]
[no] >>> yes
```

Figure 2.9. Accept License Terms

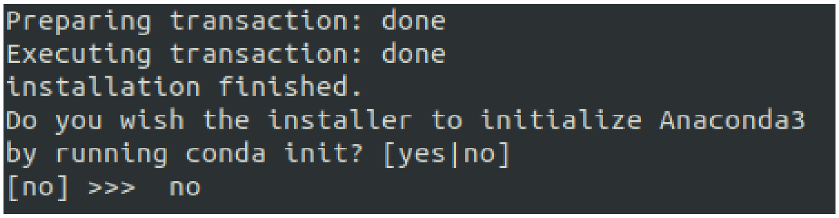
5. Confirm the installation path. Follow the instruction on screen if you want to change the default path.



```
[no] >>> yes


Anaconda3 will now be installed into this location:
/home/user/anaconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/user/anaconda3] >>> /home/user/anaconda3
```

Figure 2.10. Confirm/Edit Installation Location

6. After installation, enter **No** as shown in Figure 2.11.



```
Preparing transaction: done
Executing transaction: done
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>>  no
```

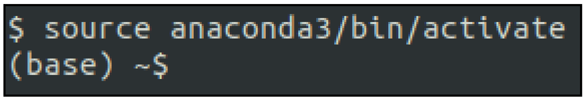
Figure 2.11. Launch/Initialize Anaconda Environment on Installation Completion

2.2.4. Installing the TensorFlow v1.14

To install the TensorFlow v1.14:

1. Activate the conda environment by running the command below:

```
$ source <conda directory>/bin/activate
```

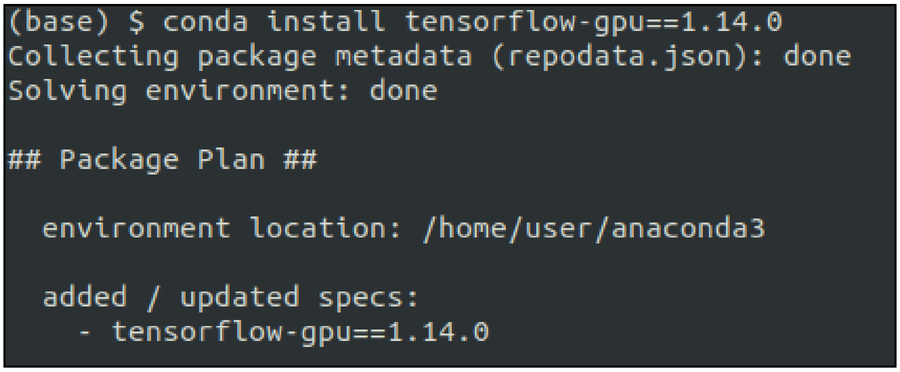


```
$ source anaconda3/bin/activate
(base) ~$
```

Figure 2.12. Anaconda Environment Activation

2. Install the TensorFlow by running the command example below:

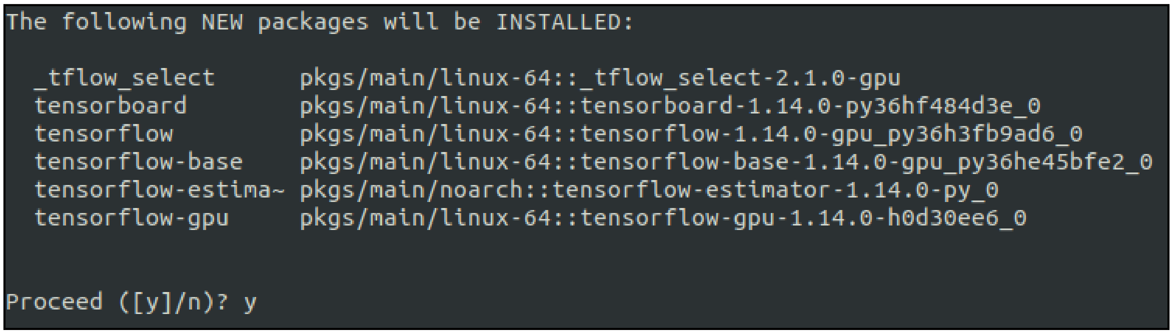
```
$ conda install tensorflow-gpu==1.14.0
```



```
(base) $ conda install tensorflow-gpu==1.14.0
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - tensorflow-gpu==1.14.0
```

Figure 2.13. TensorFlow Installation

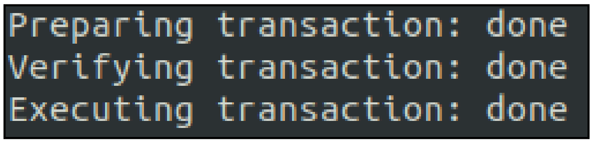
3. After installation, enter **Y** as shown in Figure 2.14.



```
The following NEW packages will be INSTALLED:

  _tflow_select      pkgs/main/linux-64::_tflow_select-2.1.0-gpu
  tensorboard        pkgs/main/linux-64::tensorboard-1.14.0-py36hf484d3e_0
  tensorflow         pkgs/main/linux-64::tensorflow-1.14.0-gpu_py36h3fb9ad6_0
  tensorflow-base    pkgs/main/linux-64::tensorflow-base-1.14.0-gpu_py36he45bfe2_0
  tensorflow-estima~ pkgs/main/noarch::tensorflow-estimator-1.14.0-py_0
  tensorflow-gpu     pkgs/main/linux-64::tensorflow-gpu-1.14.0-h0d30ee6_0


Proceed ([y]/n)? y
```

Figure 2.14. TensorFlow Installation Confirmation

Figure 2.15 shows that the TensorFlow installation is complete.



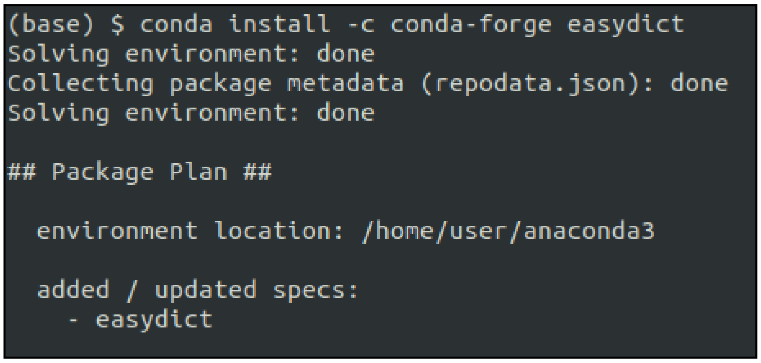
```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Figure 2.15. TensorFlow Installation Completion

2.2.5. Installing the Python Package

To install the Python package:

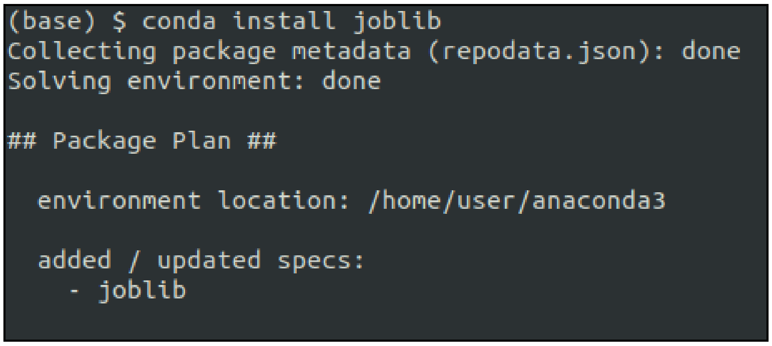
1. Install Easydict by running the command below:
   ```
   $ conda install –c conda-forge easydict
   ```



```
(base) $ conda install -c conda-forge easydict
Solving environment: done
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - easydict
```

Figure 2.16. Easydict Installation

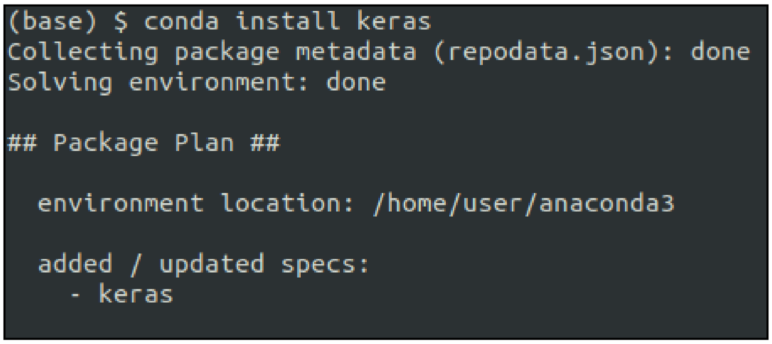
2. Install Joblib by running the command below:
   ```
   $ conda install joblib
   ```



```
(base) $ conda install joblib
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - joblib
```

Figure 2.17. Joblib Installation

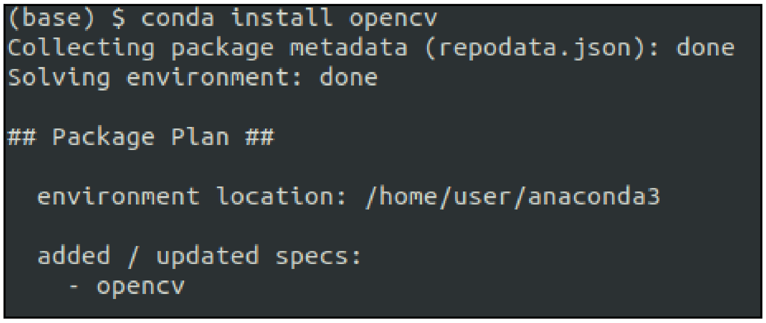
3. Install Keras by running the command below:
   ```
   $ conda install keras
   ```



```
(base) $ conda install keras
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - keras
```

Figure 2.18. Keras Installation

4. Install OpenCV by running the command below:
   ```
   $ conda install opencv
   ```

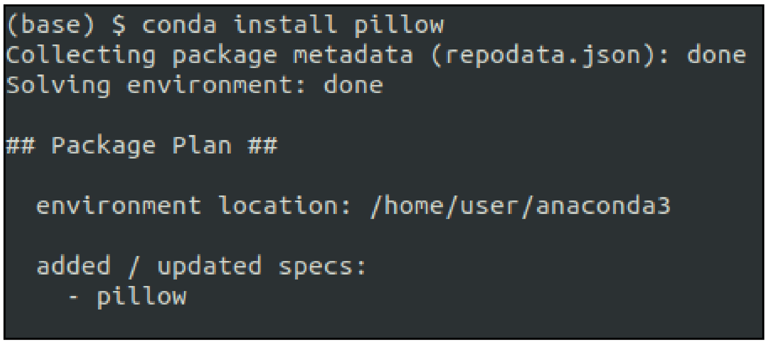


```
(base) $ conda install opencv
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - opencv
```

Figure 2.19. OpenCV Installation

5. Install Pillow by running the command below:

```
$ conda install pillow
```



```
(base) $ conda install pillow
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - pillow
```

Figure 2.20. Pillow Installation

3. Preparing the Dataset

This section provides the guidelines for preparing dataset to train the hand gesture model for the iCE40 UltraPlus device. Note that these examples are for reference only. Lattice is not recommending any of these dataset(s). It is recommended that you gather and prepare your own datasets for your end applications.

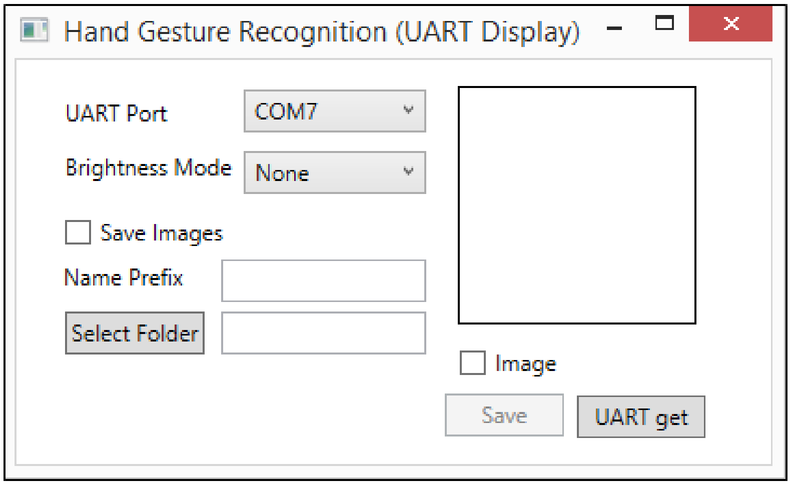
3.1. Creating the Dataset

The dataset is an important part of machine learning and model training. Feeding an accurate and diverse dataset is vital for training a high accuracy neural network.

Before creating the dataset, make sure you have the Hand Gesture Recognition UART Display software and iCE40 UltraPlus device flashed with prebuilt binaries of hand gesture demo.

Note: To flash prebuilt binaries, follow the steps in the Programming the Demo section**.**

Once the iCE40 UltraPlus device is inserted in your Windows system, the application automatically detects the port, which appears in the *UART Port* as shown in Figure 3.1.



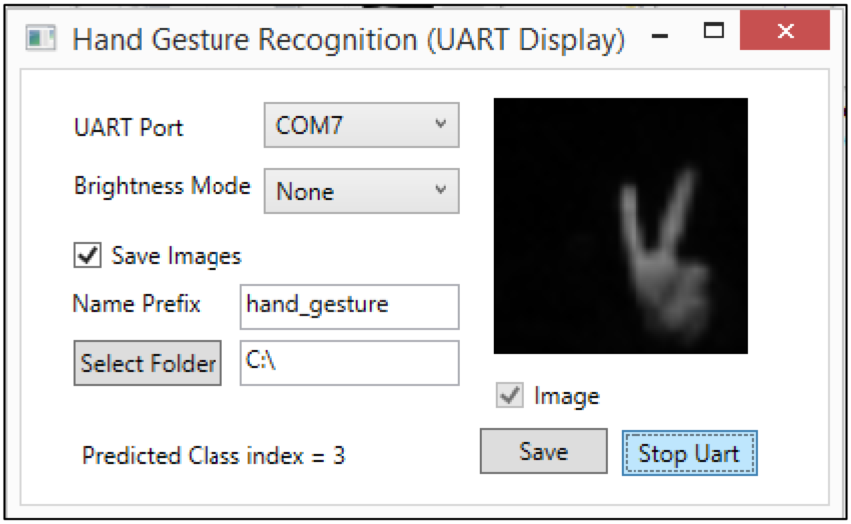
UART Port: COM7

Brightness Mode: None

Figure 3.1. UART Display Windows Application

To create the dataset:

1. Select the **Image** and **Save Images** checkboxes.
2. Enter the **Name Prefix** and select the destination folder for the gesture image.
3. Click **UART get** to display the image as shown in Figure 3.2.
4. Click **Save**.
   Note: Take at least 1000 images per gesture class. Try to capture the image from all aspects to increase the model accuracy. Also, ensure that value for **Brightness Mode** is **None**.



Hand Gesture Recognition (UART Display)

UART Port: COM7

Brightness Mode: None

☑ Save Images

Name Prefix: hand_gesture

Select Folder: C:\

☑ Image

Predicted Class index = 3

Figure 3.2. Capture Dataset Image

5. Store one gesture class image in a single folder.
6. Name the gesture class from 1 to N, where N is <= 14. As shown in Figure 3.3, a total of 10 gesture datasets are displayed along with eleventh class as background. Arrange your dataset accordingly.

   Note: (N+1) Th class is always the background, which contains plain black images.

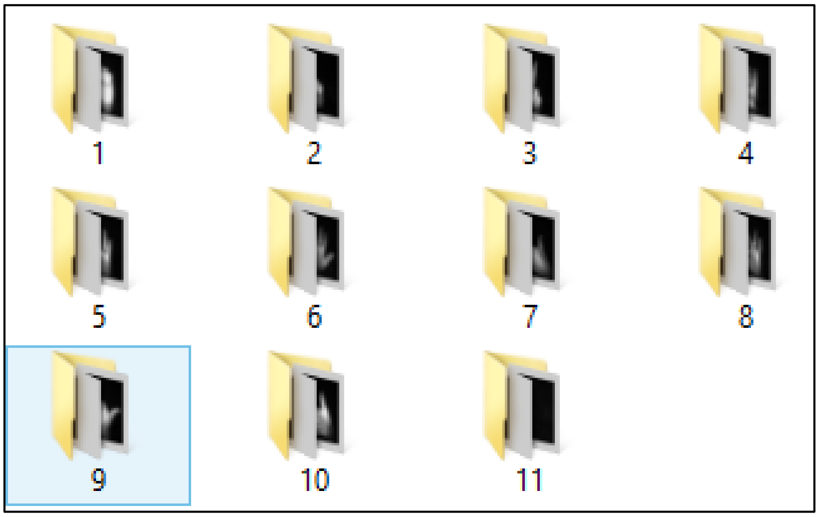
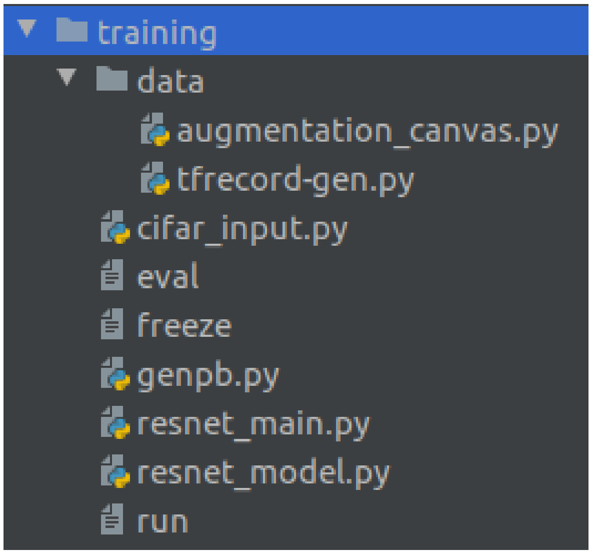


Figure 3.3. Dataset Format Sample

4. Training the Machine

4.1. Training Code Structure



```
▼ training
    ▼ data
        augmentation_canvas.py
        tfrecord-gen.py
    cifar_input.py
    eval
    freeze
    genpb.py
    resnet_main.py
    resnet_model.py
    run
```

Figure 4.1. Training Code Directory Structure

4.2. Dataset Augmentation

To augment the data, use the canvas approach. Place one image in a different part of the black canvas (see Figure 4.2).

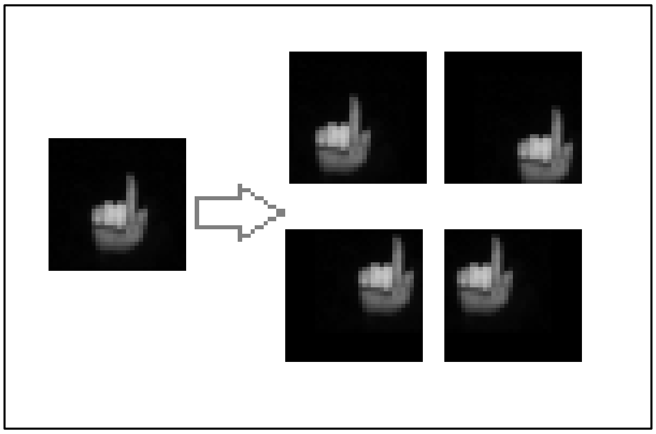


Figure 4.2. Augmentation Operations Sample

To run augmentation on the dataset, go to the *data* directory under training code and run the command below:

```
$ python augmentation_canvas.py -i <Input_dataset_root_path> -o
<Output_augmented_dataet_path>
```

4.3. Generating tfrecords from Augmented Dataset

Since this reference design only takes tfrecords of a specific format for input, generate the tfrecords file first.

To generate tfrecords from the augmented dataset you generated in Dataset Augmentation, run the command below:

```
$ python tfrecord-gen.py -i <Input_augmented_dataset_root> -o
<Output_tfrecord_path>
```

The input directory should have the directory structure shown in Figure 3.3.

Note: The output of the augmentation script (see Figure 4.2) is in the expected format of *'tfrecord-gen.py'*.

4.4. Neural Network Architecture

4.4.1. Neural Network Architecture

This section provides information on the Convolution Neural Network Configuration of the Hand Gesture Recognition design.

Table 4.1. Hand Gesture Recognition Training Network Topology

Grayscale Image Input (32 x 32 x 1)		
Fire 1	Conv3x3–16	Conv3x3 - # where: • *Conv3x3* = 3 x 3 Convolution filter Kernel size • *#* = The number of filters For example, Conv3x3 - 8 = 8 3 x 3 convolution filters Batch Norm – Batch Normalization FC - # where: • FC – Fully-connected layer • # – The number of outputs
	Batch Norm	
	ReLU	
	MaxPool	
Fire 2	Conv3x3 – 16	
	Batch Norm	
	ReLU	
Fire 3	Conv3x3 – 16	
	Batch Norm	
	ReLU	
	MaxPool	
Fire 4	Conv3x3 – 32	
	Batch Norm	
	ReLU	
Fire 5	Conv3x3 – 32	
	Batch Norm	
	ReLU	
	MaxPool	
Fire 6	Conv3x3 – 44	
	Batch Norm	
	ReLU	
Fire 7	Conv3x3 – 48	
	Batch Norm	
	ReLU	
	MaxPool	
Dropout	Dropout - 0.80	
logit	FC – (2 + Num-Gestures)	

- In Table 4.1, the layer contains Convolution (conv), Batch Normalization (bn), ReLU, MaxPool, and Dropout layers.
- The output of layer logit is the number of classes in the dataset, along with background and unknown considered as two gesture classes. The total number of outputs of the ogit layer is # of classes + 2.
- Layer information:
  - Convolutional Layer

    In general, the first layer in a CNN is always a convolutional layer**.** Each layer consists of number of filters (sometimes referred as kernels), which convolves with the input layer/image and generates an activation map (that is. feature map). This filter is an array of numbers (called weights or parameters). Each of these filters can be thought of as feature identifiers, such as straight edges, simple colors, curves, and other high-level features. For example, the filters on the first layer convolve around the input image and *activate* (or compute high values) when the specific feature it is looking for (such as curve, for example) is in the input volume.

  - ReLU (Activation Layer)

    It is the convention to apply a nonlinear layer (or activation layer) immediately after each conv layer. The purpose of this layer is to introduce nonlinearity to a system that is basically computing linear operations during the conv layers (element wise multiplications and summations). In the past, nonlinear functions such as tanh and sigmoid were used, but researchers found out that ReLU layers work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference in accuracy. The ReLU layer applies the function f(x) = max (0, x) to all of the values in the input volume. In basic terms, this layer changes all the negative activations to 0. This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the conv layer.

  - Pooling Layer

    After some ReLU layers, you may choose to apply a pooling layer. It is also referred to as a down sampling layer. In this category, there are also several layer options, with Maxpooling being the most popular. This basically takes a filter (normally of size 2 x 2) and a stride of the same length. It then applies a filter to the input volume and outputs the maximum number in every sub region that the filter convolves around.

    The intuitive reasoning behind this layer is that once it is known that a specific feature is in the original input volume (there is a high activation value), its exact location is not as important as its relative location to the other features. As you can imagine, this layer drastically reduces the spatial dimension (the length and the width change but not the depth) of the input volume. This serves two main purposes. The first is that the number of parameters or weights is reduced by 75%, thus lessening the computation cost. The second is that it controls over fitting. This term is used when a model is so tuned to the training examples that it is not able to generalize well for the validation and test sets. A symptom of over fitting is having a model that gets 100% or 99% on the training set, but only 50% on the test data.

  - Batch Normalization Layer

    Batch Normalization layer reduces the internal covariance shift. To train a neural network, some preprocessing to the input data are performed. For example, you can normalize all data so that it resembles a normal distribution (that means, zero mean and a unitary variance). This prevents the early saturation of non-linear activation functions such as sigmoid and assures that all input data are in the same range of values, and others.

    An issue, however, appears in the intermediate layers because the distribution of the activations is constantly changing during training. This slows down the training process because each layer must learn to adapt them to a new distribution in every training step. This is known as internal covariate shift.

    Batch normalization layer forces the input of every layer to have approximately the same distribution in every training step by following the process below during training time:

    a. Calculate the mean and variance of the layers input.

    b. Normalize the layer inputs using the previously calculated batch statistics.

    c. Scale and shift to obtain the output of the layer.

    This makes the learning of layers in the network more independent of each other and allows you to be carefree about weight initialization, works as regularization in place of dropout, and other regularization techniques.

- Dropout layer
  Dropout layers have a very specific function in neural networks. After training, the weights of the network are so tuned to the training examples they are given that the network does not perform well when given new examples. The idea of dropout is simplistic in nature. This layer *drops out* a random set of activations in that layer by setting them to zero. It forces the network to be redundant. That means the network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out. It makes sure that the network is not getting too *fitted* to the training data and thus helps alleviate the over fitting problem. An important note is that this layer is only used during training, and not during test time.
- Fully-connected layer
  This layer basically takes an input volume (whatever the output is of the Conv, ReLU, or MaxPool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program must choose from.
- Quantization
  Quantization is a method to bring the neural network to a reasonable size, while also achieving high performance accuracy. This is especially important for on-device applications, where the memory size and number of computations are necessarily limited. Quantization for deep learning is the process of approximating a neural network that uses floating-point numbers by a neural network of low bit width numbers. This dramatically reduces both the memory requirement and computational cost of using neural networks.

The architecture above provides nonlinearities and preservation of dimension that help to improve the robustness of the network and control over fitting.

4.4.2. Hand Gesture Recognition Network Output

The Hand Gesture recognition network gives N + 2 values from the last output node, where N is number of gesture trained. Two additional values represent background and unknown as two gesture classes.

4.4.3. Training Code Overview

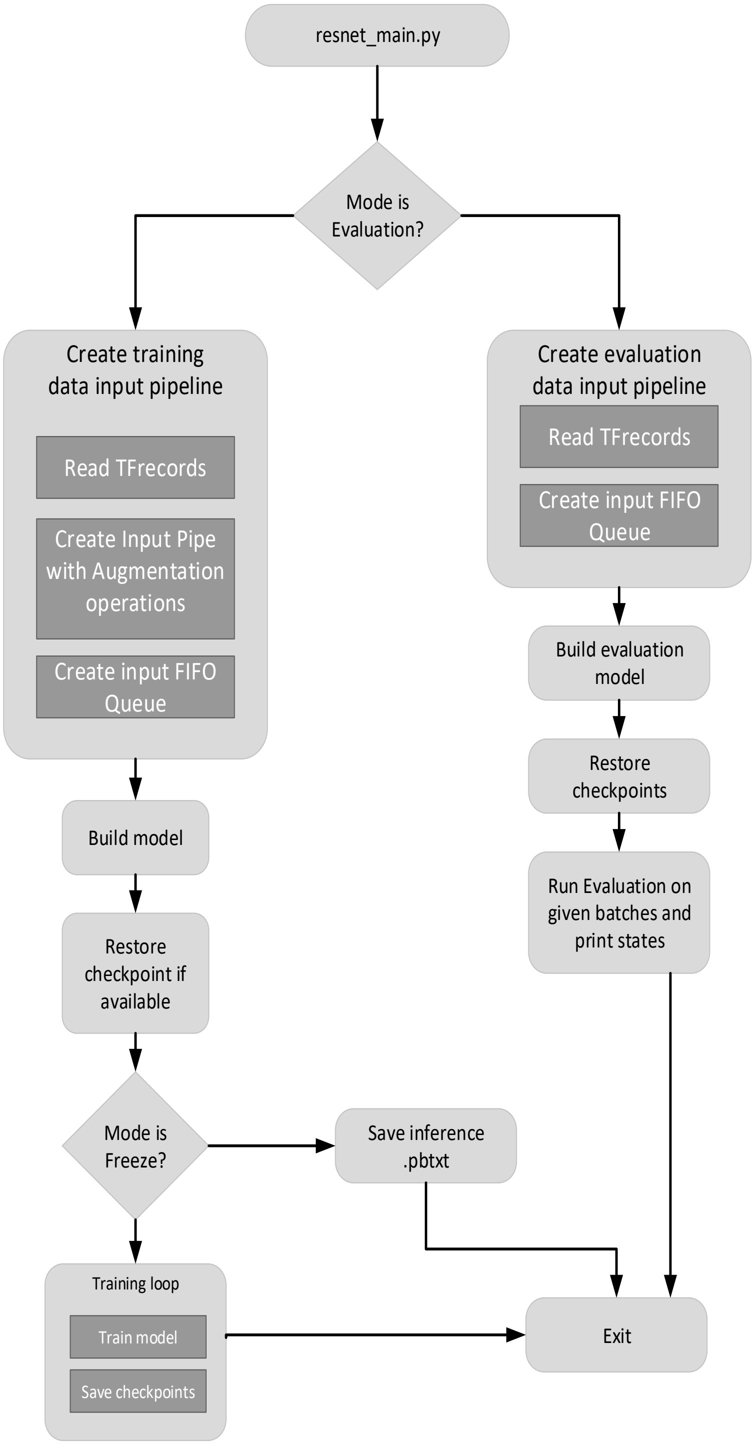
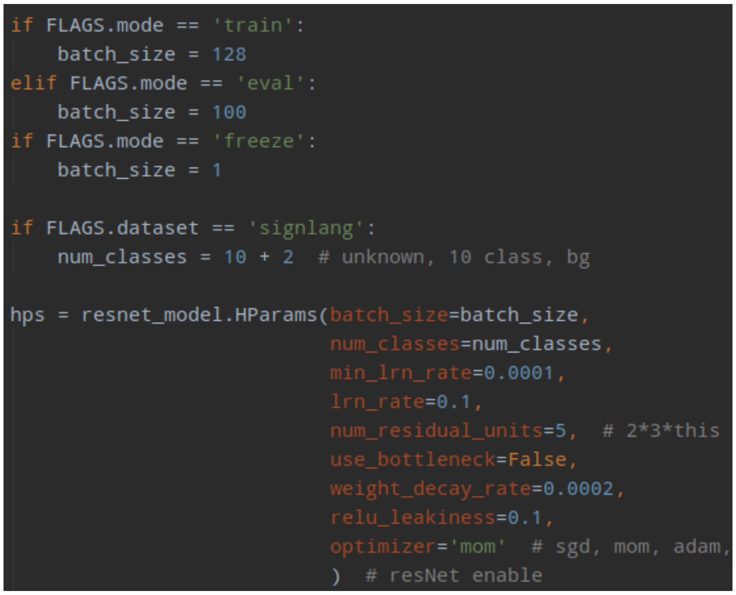


Figure 4.3. Training Code Flow Diagram

4.4.3.1. Configuring Hyper Parameters



```
if FLAGS.mode == 'train':
    batch_size = 128
elif FLAGS.mode == 'eval':
    batch_size = 100
if FLAGS.mode == 'freeze':
    batch_size = 1

if FLAGS.dataset == 'signlang':
    num_classes = 10 + 2  # unknown, 10 class, bg

hps = resnet_model.HParams(batch_size=batch_size,
                           num_classes=num_classes,
                           min_lrn_rate=0.0001,
                           lrn_rate=0.1,
                           num_residual_units=5,  # 2*3*this
                           use_bottleneck=False,
                           weight_decay_rate=0.0002,
                           relu_leakiness=0.1,
                           optimizer='mom'  # sgd, mom, adam,
                           )  # resNet enable
```

Figure 4.4. Code Snippet – Hyper Parameters

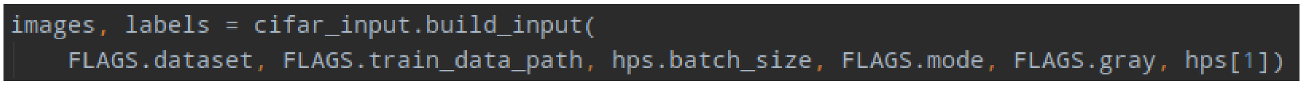
To configure hyper parameters:

1. Set the number of gestures in *num_classes* (default = 10+2).
2. Change the batch size for specific mode, if required.

   Note: The *hps* contains the list of hyper parameters for custom resnet backbone and optimizer.

4.4.3.2. Creating Training Data Input Pipeline

- *build_input* () from *cifer_input.py* reads Tfrecords and creates some augmentation operations before pushing the input data to FIFO queue.
  - *FLAGS.dataset* – dataset type (signlang)
  - *FLAGS.train_data_path* – input path to tfrecords
  - *FLAGS.batch_size* – training batch size
  - *FLAGS.mode* – train or eval
  - *FLAGS.gray* – True if model is of 1 channel otherwise False
  - *hps[1]* – num_classes configured in model hyper parameters

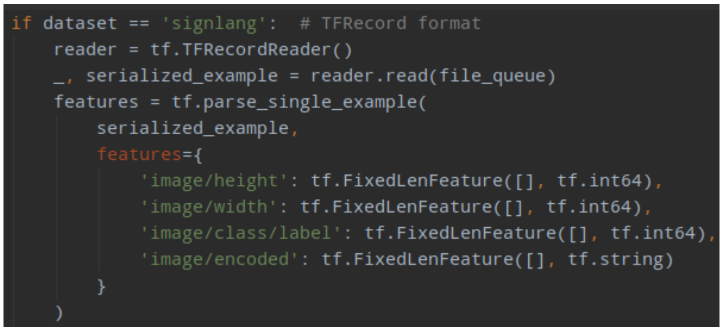


```
images, labels = cifar_input.build_input(
    FLAGS.dataset, FLAGS.train_data_path, hps.batch_size, FLAGS.mode, FLAGS.gray, hps[1])
```

Figure 4.5. Code Snippet – Build Input

Reading tfrecords

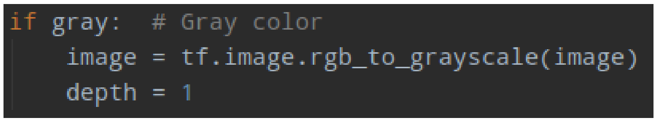
Figure 4.6 shows the reading and parsing of tfrecord files and features such as *height*, *label*, and *image*.



```
if dataset == 'signlang':  # TFRecord format
    reader = tf.TFRecordReader()
    _, serialized_example = reader.read(file_queue)
    features = tf.parse_single_example(
        serialized_example,
        features={
            'image/height': tf.FixedLenFeature([], tf.int64),
            'image/width': tf.FixedLenFeature([], tf.int64),
            'image/class/label': tf.FixedLenFeature([], tf.int64),
            'image/encoded': tf.FixedLenFeature([], tf.string)
        }
    )
```

Figure 4.6. Code Snippet – Parse tfrecords

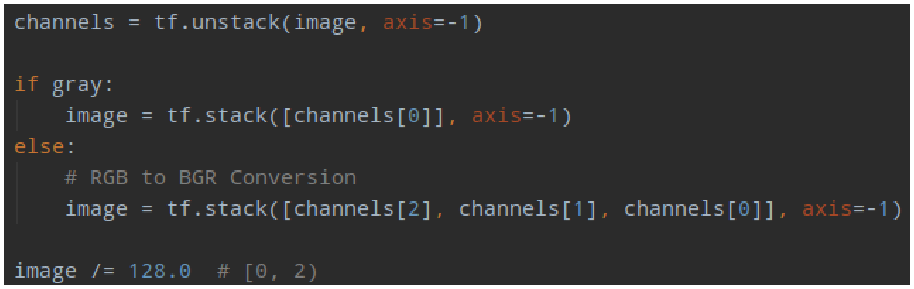
Converting and Scaling Image to Grayscale



```
if gray:  # Gray color
    image = tf.image.rgb_to_grayscale(image)
    depth = 1
```

Figure 4.7. Code Snippet – Convert Image to Grayscale

- Convert the RGB image to grayscale if gray flag is true.

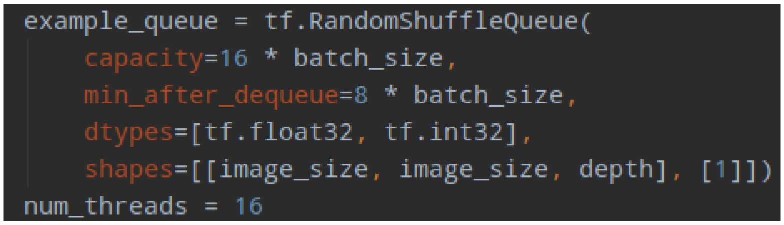


```
channels = tf.unstack(image, axis=-1)

if gray:
    image = tf.stack([channels[0]], axis=-1)
else:
    # RGB to BGR Conversion
    image = tf.stack([channels[2], channels[1], channels[0]], axis=-1)

image /= 128.0  # [0, 2)
```

Figure 4.8. Code Snippet – Convert and Scale Image to BGR

- Unstack the channel layers and convert to BGR format if the image mode is not gray. The RGB is converted to BGR because the iCE40 UltraPlus device works on BGR image.
- Divide every element on image with 128 so that the values can be scaled to 0–2 range.

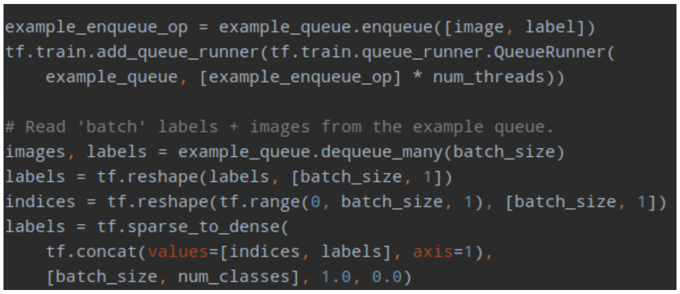
Creating Input Queue



```
example_queue = tf.RandomShuffleQueue(
    capacity=16 * batch_size,
    min_after_dequeue=8 * batch_size,
    dtypes=[tf.float32, tf.int32],
    shapes=[[image_size, image_size, depth], [1]])
num_threads = 16
```

Figure 4.9. Code Snippet – Create Queue

- *tf.RandomShuffleQueue* is a queue implementation that dequeues elements in random order.



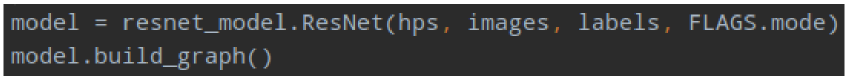
```
example_enqueue_op = example_queue.enqueue([image, label])
tf.train.add_queue_runner(tf.train.queue_runner.QueueRunner(
    example_queue, [example_enqueue_op] * num_threads))

# Read 'batch' labels + images from the example queue.
images, labels = example_queue.dequeue_many(batch_size)
labels = tf.reshape(labels, [batch_size, 1])
indices = tf.reshape(tf.range(0, batch_size, 1), [batch_size, 1])
labels = tf.sparse_to_dense(
    tf.concat(values=[indices, labels], axis=1),
    [batch_size, num_classes], 1.0, 0.0)
```

Figure 4.10. Code Snippet – Add Queue Runners

- Figure 4.9 and Figure 4.10 show the enqueuing of images and labels to *RandomShuffleQueue* and adding queue runners. This directly feeds data to the network.

4.4.3.3. Model Building

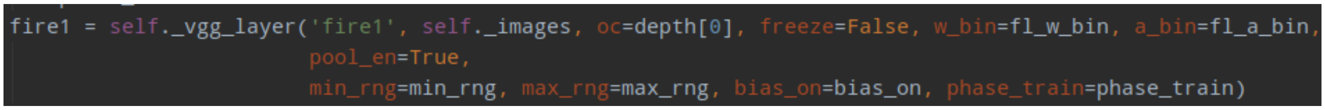
CNN Architecture



```
model = resnet_model.ResNet(hps, images, labels, FLAGS.mode)
model.build_graph()
```

Figure 4.11. Code Snippet – Create Model

- The *Build_graph ()* method creates a training graph or training model using the configuration shown in Figure 4.11.
- The *Build_graph* creates a model with seven fire layers, followed by the dropout layer and fully-connected layer. Each fire layer contains the Convolution, ReLU as activation, Batch Normalization, and MaxPool (in Fire 1, Fire 3, Fire 5, and Fire 7 only). The fully-connected layer gives the final output.

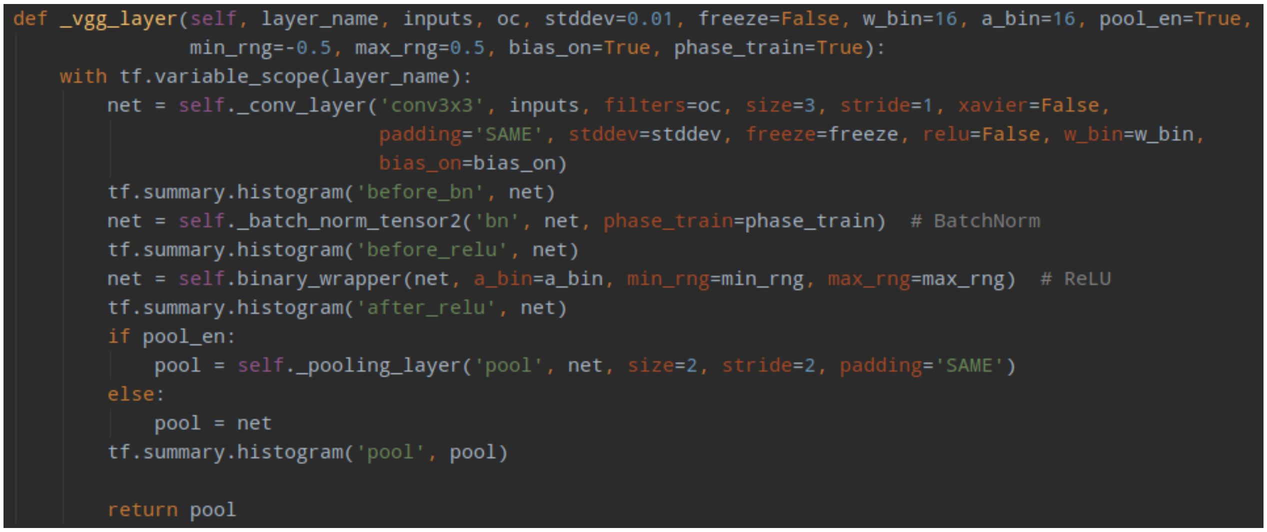


```
fire1 = self._vgg_layer('fire1', self._images, oc=depth[0], freeze=False, w_bin=fl_w_bin, a_bin=fl_a_bin,
                             pool_en=True,
                             min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, phase_train=phase_train)
```

Figure 4.12. Code Snippet – Fire Layer

- The arguments of *_vgg_layer* are specified below:
  - The first argument is the name of the block.
  - The second argument is the input node to the new fire block.
    - *oc* – Output channels are the number of filters of the convolution.
    - *freeze* – Setting weights are trainable or not.
    - *w_bin* – Quantization parameter for convolution.
    - *a_bin* – Quantization parameter for activation binarization (ReLU).

- *pool_en* – Flag to include MaxPool in the fire layer.
- *min_rng, max_rng* – Setting maximum and minimum values of quantized activation. Default values for *min_rng = 0.0* and *max_rng = 2.0*.
- *bias_on* – Sets bias add operation in graph if true.
- *phase_train* – Argument to generate graph for inference and training.



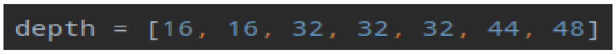
```
def _vgg_layer(self, layer_name, inputs, oc, stddev=0.01, freeze=False, w_bin=16, a_bin=16, pool_en=True,
               min_rng=-0.5, max_rng=0.5, bias_on=True, phase_train=True):
    with tf.variable_scope(layer_name):
        net = self._conv_layer('conv3x3', inputs, filters=oc, size=3, stride=1, xavier=False,
                               padding='SAME', stddev=stddev, freeze=freeze, relu=False, w_bin=w_bin,
                               bias_on=bias_on)
        tf.summary.histogram('before_bn', net)
        net = self._batch_norm_tensor2('bn', net, phase_train=phase_train)  # BatchNorm
        tf.summary.histogram('before_relu', net)
        net = self.binary_wrapper(net, a_bin=a_bin, min_rng=min_rng, max_rng=max_rng)  # ReLU
        tf.summary.histogram('after_relu', net)
        if pool_en:
            pool = self._pooling_layer('pool', net, size=2, stride=2, padding='SAME')
        else:
            pool = net
        tf.summary.histogram('pool', pool)

        return pool
```

Figure 4.13. Code Snippet – Convolution Block

- In the *resnet_model.py* file, the basic network construction blocks are implemented in the specific functions below:
  - Convolution – _conv_layer
  - Batch Normalization – _batch_norm_tensor2
  - ReLU – binary_wrapper
  - MaxPool – _pooling_layer
- _conv_layer
  - The *_conv_layer* contains the code to create the convolution block. This code contains the kernel variable, variable initializer, quantization code, convolution operation, and ReLU if argument *relu* is True.
- _batch_norm_tensor2
  - The *_batch_norm_tensor2* contains the code to create the batch normalization operation for both training and inference phase.
- binary_wrapper
  - The *binary_wrapper* is used for quantized activation with ReLU.
- _pooling_layer
  - The *_pooling_layer* adds MaxPooling with the given kernel size and stride size to training and inference graph.

Fire Layer Feature Depth

The *depth* list contains the feature depth for seven fire layers in network.

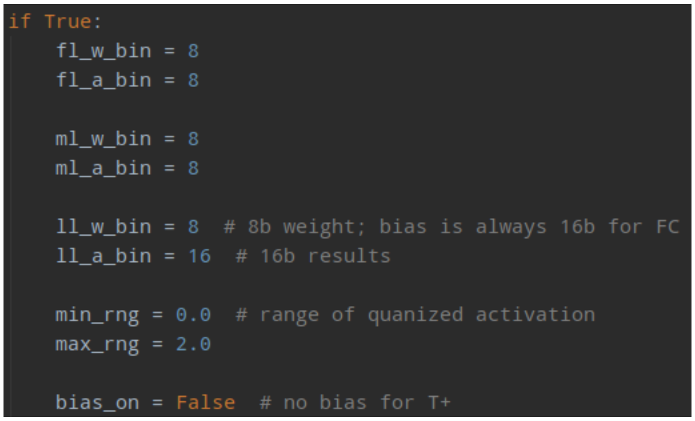


```
depth = [16, 16, 32, 32, 32, 44, 48]
```

Figure 4.14. Code Snippet: Feature depth array for fire layers

Quantization

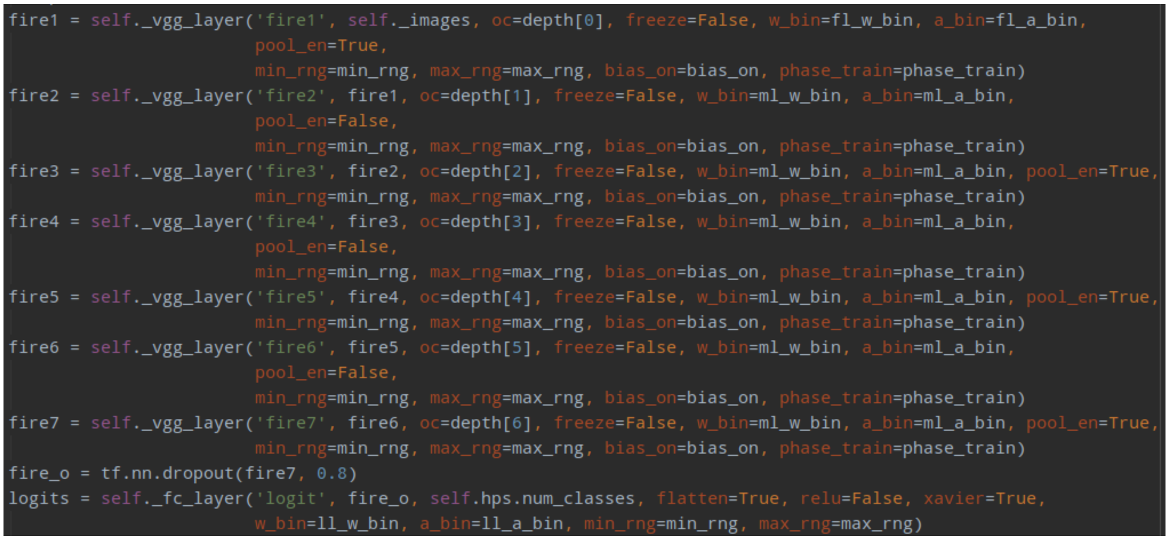
- For more information about quantization, refer to the Quantization section in Neural Network Architecture.



```
if True:
    fl_w_bin = 8
    fl_a_bin = 8

    ml_w_bin = 8
    ml_a_bin = 8

    ll_w_bin = 8  # 8b weight; bias is always 16b for FC
    ll_a_bin = 16  # 16b results

    min_rng = 0.0  # range of quanized activation
    max_rng = 2.0

    bias_on = False  # no bias for T+
```

Figure 4.15. Code Snippet – Quantization Parameters

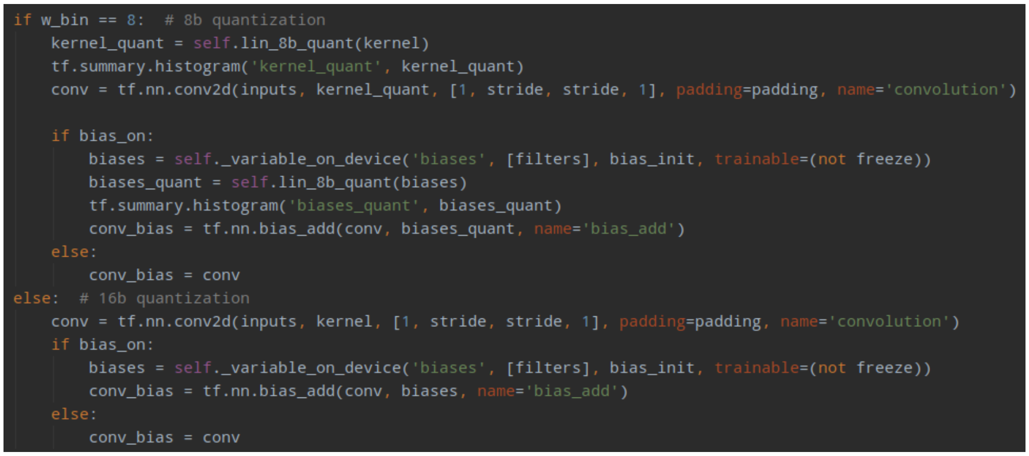
- **fl_w_bin** – Quantization parameter for first convolution layer
- **fl_a_bin** – Quantization parameter for activation of first layer
- **ml_w_bin** – Quantization parameter for rest of convolution layers
- **ml_a_bin** – Quantization parameter for rest of activation layers
- **l1_w_bin** – Quantization parameter for multiplication in fully-connected layer
- **l1_a_bin** – Quantization parameter for bias add operation in fully-connected layer
- **min_rng** – Minimum value for quantization output
- **max_rng** – Maximum value for quantization output



```
fire1 = self._vgg_layer('fire1', self._images, oc=depth[0], freeze=False, w_bin=fl_w_bin, a_bin=fl_a_bin,
                        pool_en=True,
                        min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, phase_train=phase_train)
fire2 = self._vgg_layer('fire2', fire1, oc=depth[1], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
                        pool_en=False,
                        min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, phase_train=phase_train)
fire3 = self._vgg_layer('fire3', fire2, oc=depth[2], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=True,
                        min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, phase_train=phase_train)
fire4 = self._vgg_layer('fire4', fire3, oc=depth[3], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
                        pool_en=False,
                        min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, phase_train=phase_train)
fire5 = self._vgg_layer('fire5', fire4, oc=depth[4], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=True,
                        min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, phase_train=phase_train)
fire6 = self._vgg_layer('fire6', fire5, oc=depth[5], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
                        pool_en=False,
                        min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, phase_train=phase_train)
fire7 = self._vgg_layer('fire7', fire6, oc=depth[6], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=True,
                        min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, phase_train=phase_train)
fire_o = tf.nn.dropout(fire7, 0.8)
logits = self._fc_layer('logit', fire_o, self.hps.num_classes, flatten=True, relu=False, xavier=True,
                        w_bin=ll_w_bin, a_bin=ll_a_bin, min_rng=min_rng, max_rng=max_rng)
```

Figure 4.16. Code Snippet – Forward Graph Fire Layers

- The 8-bit quantization is done on weights and activations in this model. Based on value of *w_bin* and *a_bin*, it is decided if quantization should be performed or not.

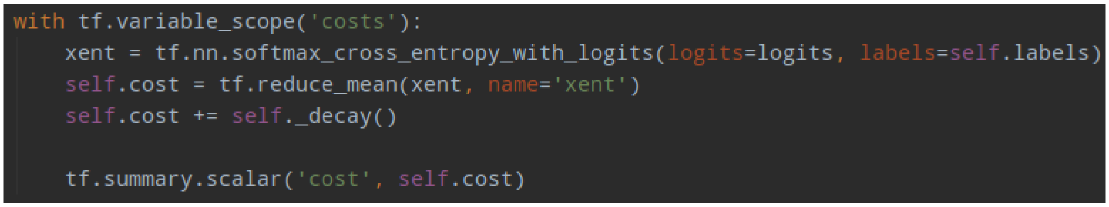


```
if w_bin == 8:  # 8b quantization
    kernel_quant = self.lin_8b_quant(kernel)
    tf.summary.histogram('kernel_quant', kernel_quant)
    conv = tf.nn.conv2d(inputs, kernel_quant, [1, stride, stride, 1], padding=padding, name='convolution')

    if bias_on:
        biases = self._variable_on_device('biases', [filters], bias_init, trainable=(not freeze))
        biases_quant = self.lin_8b_quant(biases)
        tf.summary.histogram('biases_quant', biases_quant)
        conv_bias = tf.nn.bias_add(conv, biases_quant, name='bias_add')
    else:
        conv_bias = conv
else:  # 16b quantization
    conv = tf.nn.conv2d(inputs, kernel, [1, stride, stride, 1], padding=padding, name='convolution')
    if bias_on:
        biases = self._variable_on_device('biases', [filters], bias_init, trainable=(not freeze))
        conv_bias = tf.nn.bias_add(conv, biases, name='bias_add')
    else:
        conv_bias = conv
```

Figure 4.17. Code Snippet – Convolution Quantization

Loss Function and Optimizers

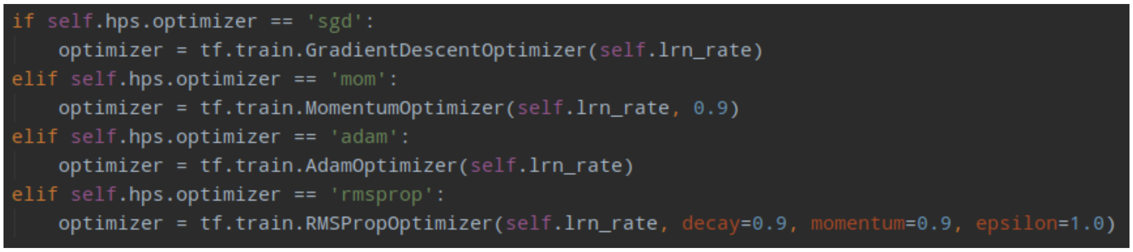
- The Loss Function model uses *softmax_cross_entropy_with_logitds* because the labels are in the form of class index.



```
with tf.variable_scope('costs'):
    xent = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=self.labels)
    self.cost = tf.reduce_mean(xent, name='xent')
    self.cost += self._decay()

    tf.summary.scalar('cost', self.cost)
```

Figure 4.18. Code Snippet – Loss Function

- Figure 4.19 shows the four options for selecting optimizers. In this model, you are using *mom* optimizer as default.

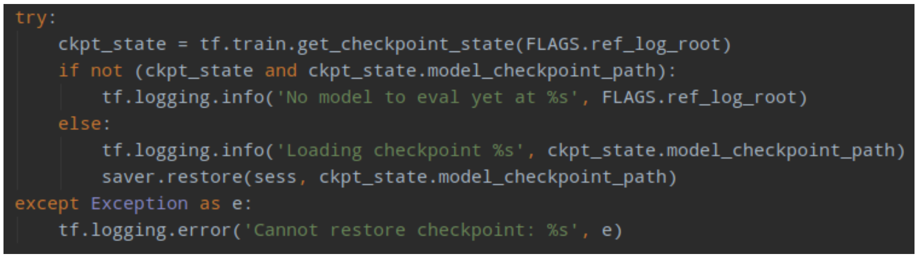


```
if self.hps.optimizer == 'sgd':
    optimizer = tf.train.GradientDescentOptimizer(self.lrn_rate)
elif self.hps.optimizer == 'mom':
    optimizer = tf.train.MomentumOptimizer(self.lrn_rate, 0.9)
elif self.hps.optimizer == 'adam':
    optimizer = tf.train.AdamOptimizer(self.lrn_rate)
elif self.hps.optimizer == 'rmsprop':
    optimizer = tf.train.RMSPropOptimizer(self.lrn_rate, decay=0.9, momentum=0.9, epsilon=1.0)
```

Figure 4.19. Code Snippet – Optimizers

Restoring Checkpoints

Checkpoints are restored from the log directory and starts training from that checkpoint, if the checkpoints exist in the log directory.

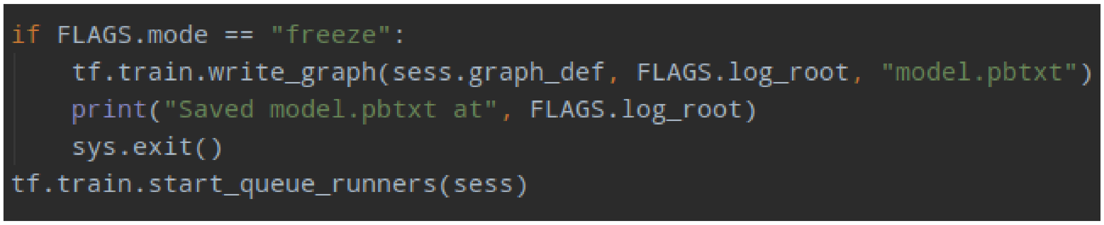


```
try:
    ckpt_state = tf.train.get_checkpoint_state(FLAGS.ref_log_root)
    if not (ckpt_state and ckpt_state.model_checkpoint_path):
        tf.logging.info('No model to eval yet at %s', FLAGS.ref_log_root)
    else:
        tf.logging.info('Loading checkpoint %s', ckpt_state.model_checkpoint_path)
        saver.restore(sess, ckpt_state.model_checkpoint_path)
except Exception as e:
    tf.logging.error('Cannot restore checkpoint: %s', e)
```

Figure 4.20. Code Snippet – Restore Checkpoints

Saving .pbtxt

If mode is *freeze*, it saves the inference graph (model) as *.pbtxt* file. The *.pbtxt* file is later used for freezing purposes.

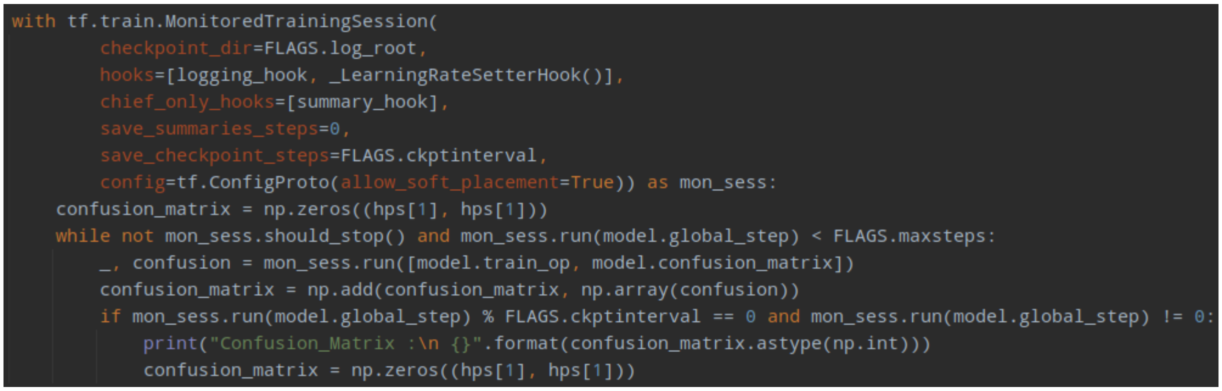


```
if FLAGS.mode == "freeze":
    tf.train.write_graph(sess.graph_def, FLAGS.log_root, "model.pbtxt")
    print("Saved model.pbtxt at", FLAGS.log_root)
    sys.exit()
tf.train.start_queue_runners(sess)
```

Figure 4.21. Code Snippet – Save .pbtxt

Training Loop

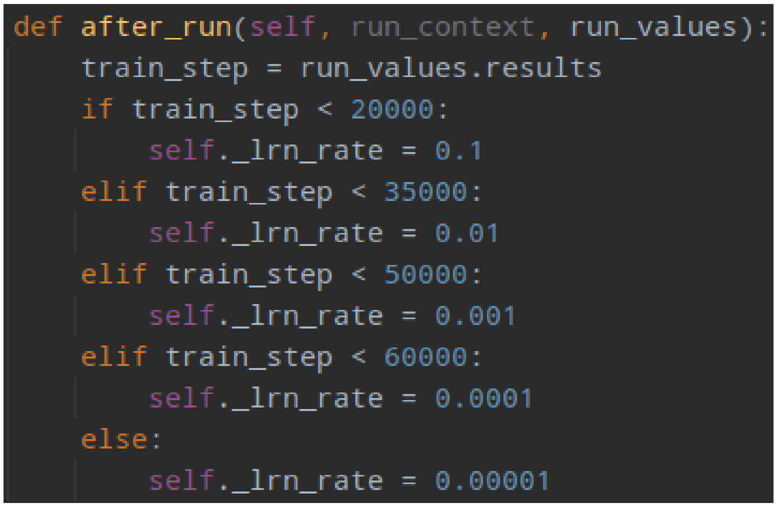
- The *MonitoredTrainingSession* utility sets the proper session initializer/restorer. It also creates hooks related to checkpoint and summary saving. For workers, this utility sets proper session creator, which waits for the chief to initialize/restore. For more information, go to tf.compat.v1.train.MonitoredSession.



```
with tf.train.MonitoredTrainingSession(
        checkpoint_dir=FLAGS.log_root,
        hooks=[logging_hook, _LearningRateSetterHook()],
        chief_only_hooks=[summary_hook],
        save_summaries_steps=0,
        save_checkpoint_steps=FLAGS.ckptinterval,
        config=tf.ConfigProto(allow_soft_placement=True)) as mon_sess:
    confusion_matrix = np.zeros((hps[1], hps[1]))
    while not mon_sess.should_stop() and mon_sess.run(model.global_step) < FLAGS.maxsteps:
        _, confusion = mon_sess.run([model.train_op, model.confusion_matrix])
        confusion_matrix = np.add(confusion_matrix, np.array(confusion))
        if mon_sess.run(model.global_step) % FLAGS.ckptinterval == 0 and mon_sess.run(model.global_step) != 0:
            print("Confusion_Matrix :\n {}".format(confusion_matrix.astype(np.int)))
            confusion_matrix = np.zeros((hps[1], hps[1]))
```

Figure 4.22. Code Snippet – Training Loop

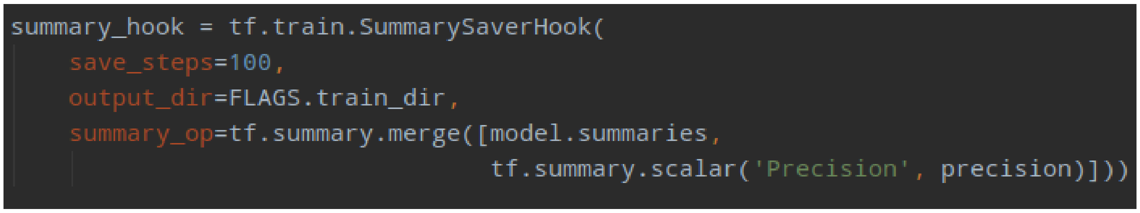
- _LearningRateSetterHook
    - The *_LearningRateSetterHook* sets the learning rate based on the training steps performed.



```
def after_run(self, run_context, run_values):
    train_step = run_values.results
    if train_step < 20000:
        self._lrn_rate = 0.1
    elif train_step < 35000:
        self._lrn_rate = 0.01
    elif train_step < 50000:
        self._lrn_rate = 0.001
    elif train_step < 60000:
        self._lrn_rate = 0.0001
    else:
        self._lrn_rate = 0.00001
```

Figure 4.23. Code Snippet – _LearningRateSetterHook

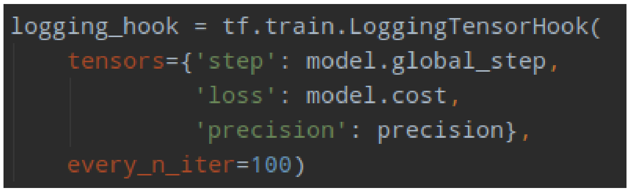
- Summary_hook
  - The *Summary_hook* saves the TensorBoard summary for every 100 steps.



```
summary_hook = tf.train.SummarySaverHook(
    save_steps=100,
    output_dir=FLAGS.train_dir,
    summary_op=tf.summary.merge([model.summaries,
                                 tf.summary.scalar('Precision', precision)]))
```

Figure 4.24. Code Snippet – Save Summary for TensorBoard

- Logging_hook
  - The *Logging_hook* prints the logs after every 100 iterations.



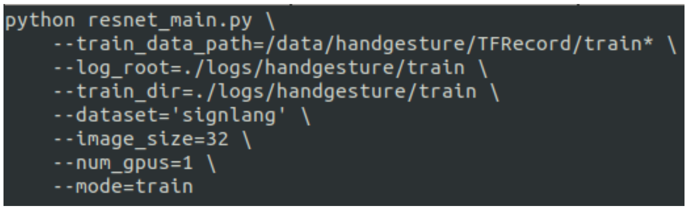
```
logging_hook = tf.train.LoggingTensorHook(
    tensors={'step': model.global_step,
             'loss': model.cost,
             'precision': precision},
    every_n_iter=100)
```

Figure 4.25. Code Snippet – Logging Hook

4.5. Training from Scratch and/or Transfer Learning

To train the machine:

1. Open **run** script and modify the parameters as required.

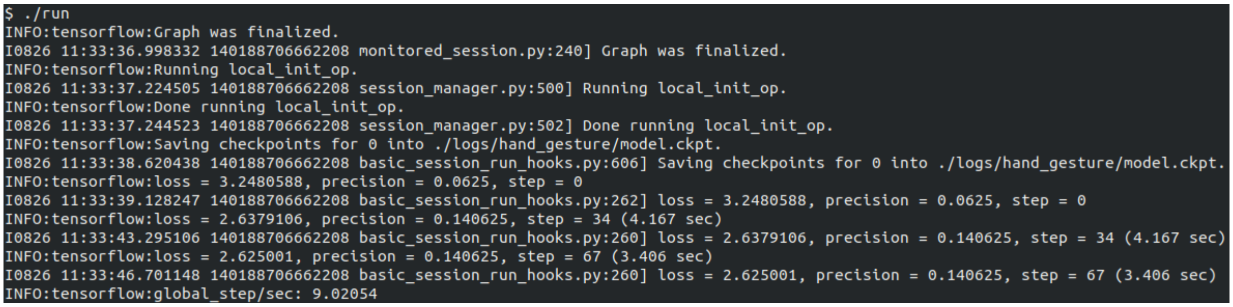


```
python resnet_main.py \
    --train_data_path=/data/handgesture/TFRecord/train* \
    --log_root=./logs/handgesture/train \
    --train_dir=./logs/handgesture/train \
    --dataset='signlang' \
    --image_size=32 \
    --num_gpus=1 \
    --mode=train
```

Figure 4.26. Hand Gesture Recognition – Run Script

To start training, run the command below.

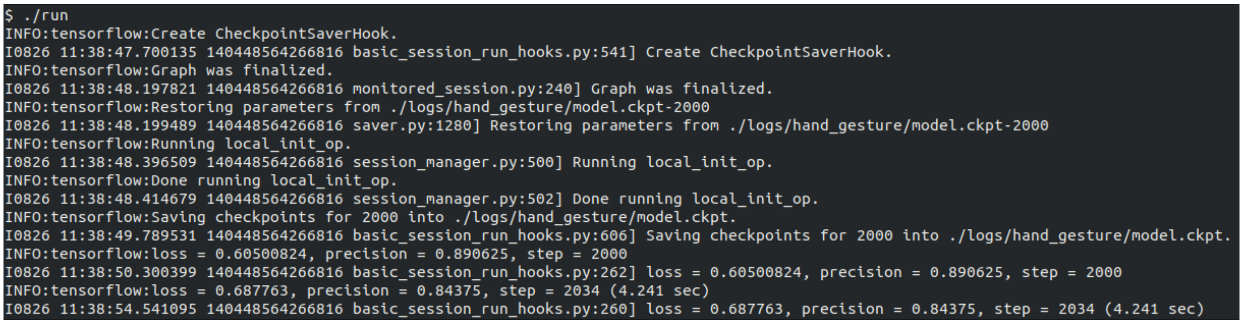
```
$ ./run
```



```
$ ./run
INFO:tensorflow:Graph was finalized.
I0826 11:33:36.998332 140188706662208 monitored_session.py:240] Graph was finalized.
INFO:tensorflow:Running local_init_op.
I0826 11:33:37.224505 140188706662208 session_manager.py:500] Running local_init_op.
INFO:tensorflow:Done running local_init_op.
I0826 11:33:37.244523 140188706662208 session_manager.py:502] Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 0 into ./logs/hand_gesture/model.ckpt.
I0826 11:33:38.620438 140188706662208 basic_session_run_hooks.py:606] Saving checkpoints for 0 into ./logs/hand_gesture/model.ckpt.
INFO:tensorflow:loss = 3.2480588, precision = 0.0625, step = 0
I0826 11:33:39.128247 140188706662208 basic_session_run_hooks.py:262] loss = 3.2480588, precision = 0.0625, step = 0
INFO:tensorflow:loss = 2.6379106, precision = 0.140625, step = 34 (4.167 sec)
I0826 11:33:43.295106 140188706662208 basic_session_run_hooks.py:260] loss = 2.6379106, precision = 0.140625, step = 34 (4.167 sec)
INFO:tensorflow:loss = 2.625001, precision = 0.140625, step = 67 (3.406 sec)
I0826 11:33:46.701148 140188706662208 basic_session_run_hooks.py:260] loss = 2.625001, precision = 0.140625, step = 67 (3.406 sec)
INFO:tensorflow:global_step/sec: 9.02054
```

Figure 4.27. Hand Gesture Recognition – Trigger Training

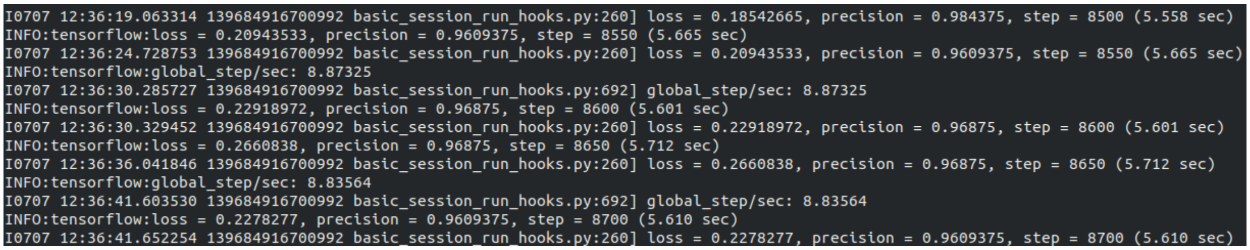
2. To restore the checkpoints, run the same command again with the same log directory. If checkpoints are present in the log path, it is restored and resumes training from that point.



```
$ ./run
INFO:tensorflow:Create CheckpointSaverHook.
I0826 11:38:47.700135 140448564266816 basic_session_run_hooks.py:541] Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
I0826 11:38:48.197821 140448564266816 monitored_session.py:240] Graph was finalized.
INFO:tensorflow:Restoring parameters from ./logs/hand_gesture/model.ckpt-2000
I0826 11:38:48.199489 140448564266816 saver.py:1280] Restoring parameters from ./logs/hand_gesture/model.ckpt-2000
INFO:tensorflow:Running local_init_op.
I0826 11:38:48.396509 140448564266816 session_manager.py:500] Running local_init_op.
INFO:tensorflow:Done running local_init_op.
I0826 11:38:48.414679 140448564266816 session_manager.py:502] Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 2000 into ./logs/hand_gesture/model.ckpt.
I0826 11:38:49.789531 140448564266816 basic_session_run_hooks.py:606] Saving checkpoints for 2000 into ./logs/hand_gesture/model.ckpt.
INFO:tensorflow:loss = 0.60500824, precision = 0.890625, step = 2000
I0826 11:38:50.300399 140448564266816 basic_session_run_hooks.py:262] loss = 0.60500824, precision = 0.890625, step = 2000
INFO:tensorflow:loss = 0.687763, precision = 0.84375, step = 2034 (4.241 sec)
I0826 11:38:54.541095 140448564266816 basic_session_run_hooks.py:260] loss = 0.687763, precision = 0.84375, step = 2034 (4.241 sec)
```

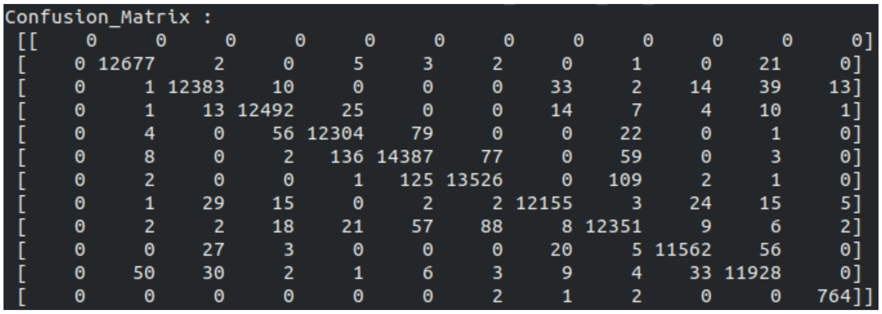
Figure 4.28. Hand Gesture Recognition – Trigger Training with Transfer Learning

Training status can be checked in the logs by observing different terminologies like loss, precision, and confusion matrix.



```
I0707 12:36:19.063314 139684916700992 basic_session_run_hooks.py:260] loss = 0.18542665, precision = 0.984375, step = 8500 (5.558 sec)
INFO:tensorflow:loss = 0.20943533, precision = 0.9609375, step = 8550 (5.665 sec)
I0707 12:36:24.728753 139684916700992 basic_session_run_hooks.py:260] loss = 0.20943533, precision = 0.9609375, step = 8550 (5.665 sec)
INFO:tensorflow:global_step/sec: 8.87325
I0707 12:36:30.285727 139684916700992 basic_session_run_hooks.py:692] global_step/sec: 8.87325
INFO:tensorflow:loss = 0.22918972, precision = 0.96875, step = 8600 (5.601 sec)
I0707 12:36:30.329452 139684916700992 basic_session_run_hooks.py:260] loss = 0.22918972, precision = 0.96875, step = 8600 (5.601 sec)
INFO:tensorflow:loss = 0.2660838, precision = 0.96875, step = 8650 (5.712 sec)
I0707 12:36:36.041846 139684916700992 basic_session_run_hooks.py:260] loss = 0.2660838, precision = 0.96875, step = 8650 (5.712 sec)
INFO:tensorflow:global_step/sec: 8.83564
I0707 12:36:41.603530 139684916700992 basic_session_run_hooks.py:692] global_step/sec: 8.83564
INFO:tensorflow:loss = 0.2278277, precision = 0.9609375, step = 8700 (5.610 sec)
I0707 12:36:41.652254 139684916700992 basic_session_run_hooks.py:260] loss = 0.2278277, precision = 0.9609375, step = 8700 (5.610 sec)
```

Figure 4.29. Hand Gesture Recognition – Training Logs

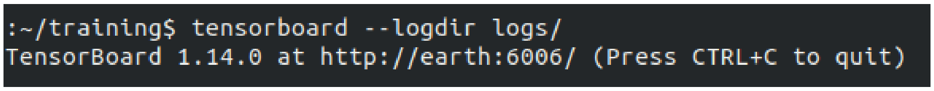


```
Confusion_Matrix :
 [[    0     0     0     0     0     0     0     0     0     0     0     0]
 [    0 12677     2     0     5     3     2     0     1     0    21     0]
 [    0     1 12383    10     0     0     0    33     2    14    39    13]
 [    0     1    13 12492    25     0     0    14     7     4    10     1]
 [    0     4     0    56 12304    79     0     0    22     0     1     0]
 [    0     8     0     2   136 14387    77     0    59     0     3     0]
 [    0     2     0     0     1   125 13526     0   109     2     1     0]
 [    0     1    29    15     0     2     2 12155     3    24    15     5]
 [    0     2     2    18    21    57    88     8 12351     9     6     2]
 [    0     0    27     3     0     0     0    20     5 11562    56     0]
 [    0    50    30     2     1     6     3     9     4    33 11928     0]
 [    0     0     0     0     0     0     2     1     2     0     0   764]]
```

Figure 4.30. Hand Gesture Recognition – Confusion Matrix

3. Start TensorBoard.

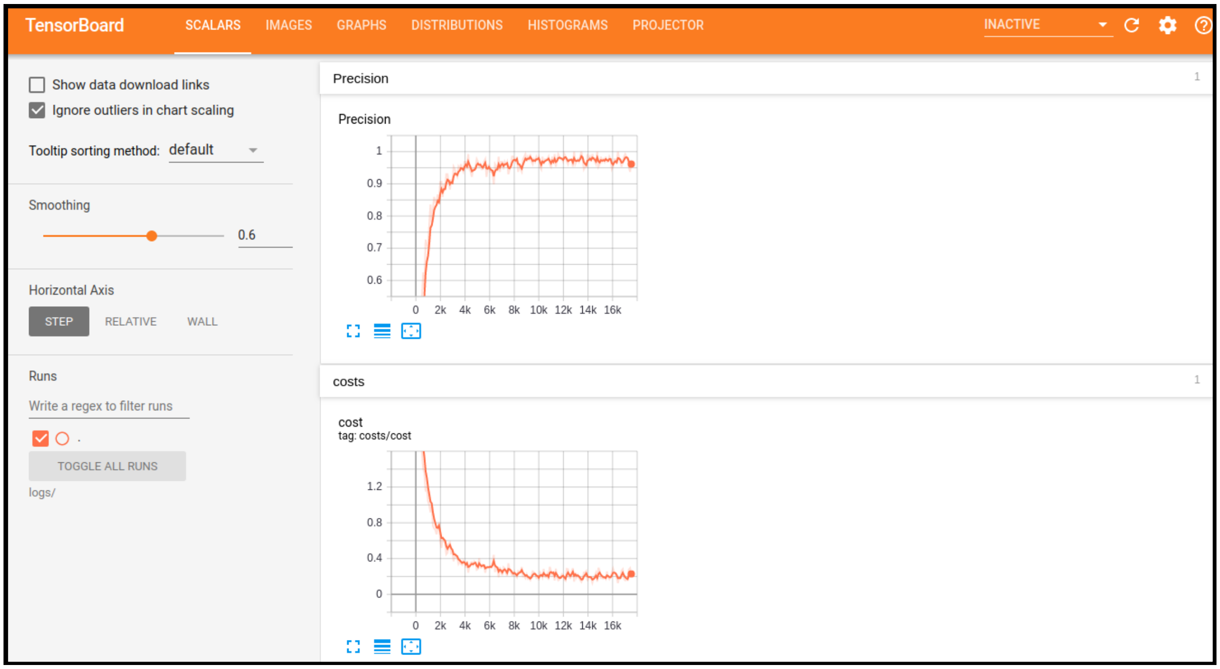
```
$ tensorboard – logdir=<log directory of training>
```



```
:~/training$ tensorboard --logdir logs/
TensorBoard 1.14.0 at http://earth:6006/ (Press CTRL+C to quit)
```

Figure 4.31. TensorBoard – Launch

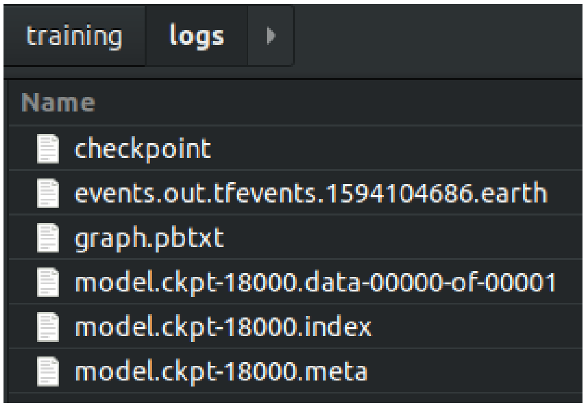
4. Check the training status on TensorBoard.



□ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP RELATIVE WALL
Write a regex to filter runs
☑ ○ .

Figure 4.32. TensorBoard Scalars

5. Check if the checkpoint, data, meta, index, and events (if using TensorBoard) files are created at the log directory. These files are used for creating the frozen file (*.pb).



training › logs ›

Name
checkpoint
events.out.tfevents.1594104686.earth
graph.pbtxt
model.ckpt-18000.data-00000-of-00001
model.ckpt-18000.index
model.ckpt-18000.meta

Figure 4.33. Checkpoint Storage Directory Structure

5. Creating Frozen File

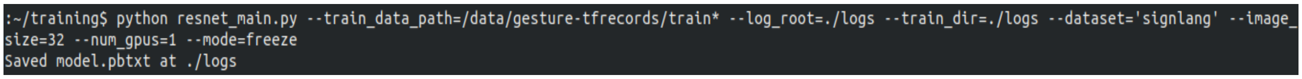
This section describes the procedure for freezing the model, which is aligned with the Lattice SensAI tool. Perform the steps below to generate the frozen protobuf file.

5.1. Generating the .pbtxt File Inference

Once training is completed, generate the inference .pbtxt file using the command below:

Note: Do not modify *config.sh* after training.

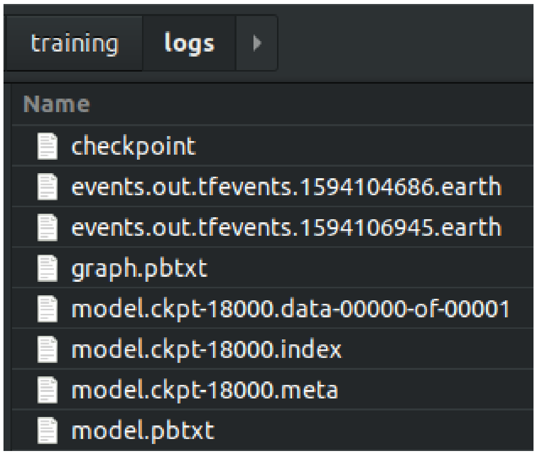
```
$ python resnet_main.py --train_data_path=<TFRecord_root_path>/train* --
log_root=<Logging_Checkpoint_Path> --train_dir=<tensorboard_summary_path> --
dataset='signlang' --image_size=32 --num_gpus=<num_GPUs> --mode=freeze
```



```
:~/training$ python resnet_main.py --train_data_path=/data/gesture-tfrecords/train* --log_root=./logs --train_dir=./logs --dataset='signlang' --image_
size=32 --num_gpus=1 --mode=freeze
Saved model.pbtxt at ./logs
```

Figure 5.1. .pbtxt File Generation for Inference

The output pbtxt file is generated for inference under the train log directory.



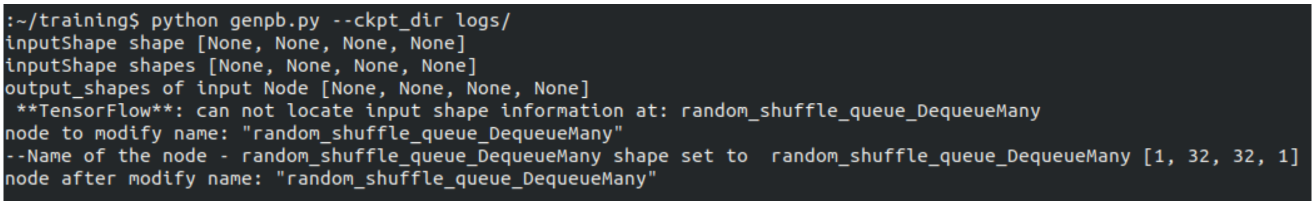
checkpoint
events.out.tfevents.1594104686.earth
events.out.tfevents.1594106945.earth
graph.pbtxt
model.ckpt-18000.data-00000-of-00001
model.ckpt-18000.index
model.ckpt-18000.meta
model.pbtxt

Figure 5.2. Generated .pbtxt for Inference

5.2. Generating the Frozen (.pb) File

Generate .pb file from latest checkpoint using the command below from the training code's root directory.

```
$ python genpb.py --ckpt_dir <COMPLETE_PATH_TO_LOG_DIRECTORY>
```

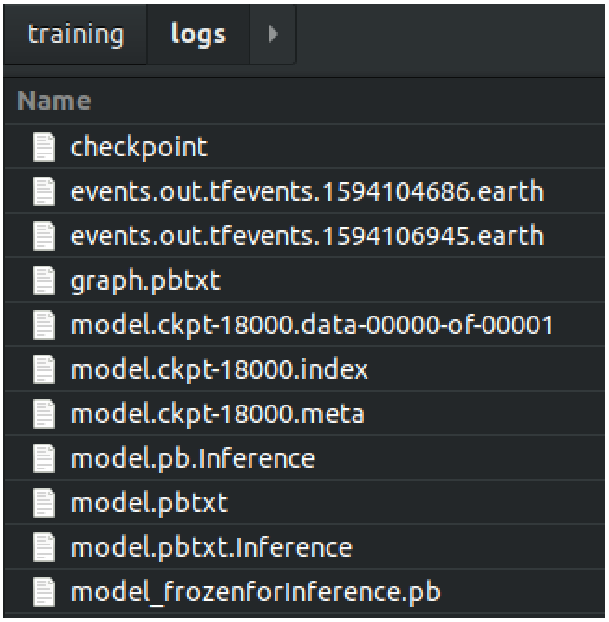


```
:~/training$ python genpb.py --ckpt_dir logs/
inputShape shape [None, None, None, None]
inputShape shapes [None, None, None, None]
output_shapes of input Node [None, None, None, None]
 **TensorFlow**: can not locate input shape information at: random_shuffle_queue_DequeueMany
node to modify name: "random_shuffle_queue_DequeueMany"
--Name of the node - random_shuffle_queue_DequeueMany shape set to  random_shuffle_queue_DequeueMany [1, 32, 32, 1]
node after modify name: "random_shuffle_queue_DequeueMany"
```

Figure 5.3. Run genpb.py to Generate Inference .pb

genpb.py uses the .pbtxt generated by the procedure performed in Generating the .pbtxt File Inference and the latest checkpoint in the train directory to generate the frozen .pb file.

Once the *genpb.py* is executed successfully, the log directory generates the *<ckpt-prefix>_frozenforinference.pb* file as shown in Figure 5.2**.**



```
training   logs
Name
checkpoint
events.out.tfevents.1594104686.earth
events.out.tfevents.1594106945.earth
graph.pbtxt
model.ckpt-18000.data-00000-of-00001
model.ckpt-18000.index
model.ckpt-18000.meta
model.pb.Inference
model.pbtxt
model.pbtxt.Inference
model_frozenforInference.pb
```

Figure 5.4. Frozen Inference .pb Output

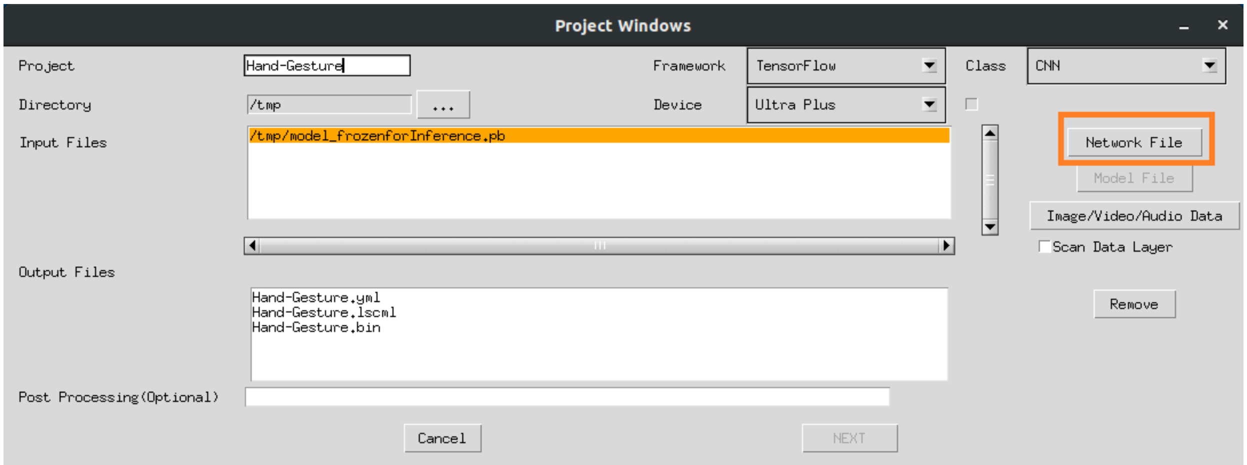
6. Creating Binary File with Lattice SensAI

This chapter describes how to generate the binary file using the Lattice SensAI version 3.1 program.

Figure 6.1. SensAI Home Screen

To create the project in SensAI tool:

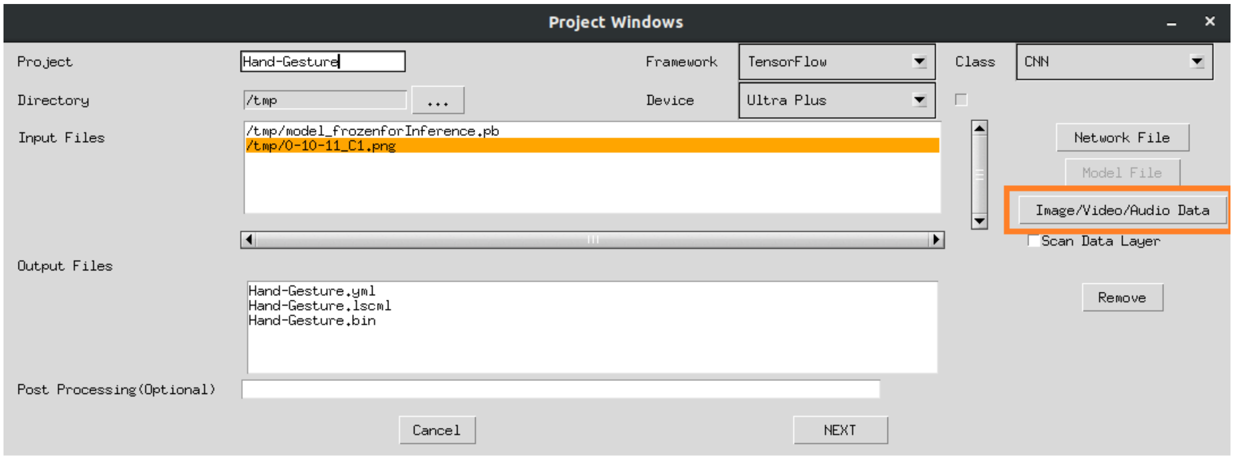
1. Click **File > New**.
2. Enter the following settings:
   - **Project Name**
   - **Framework – TensorFlow**
   - **Class – CNN**
   - **Device – UltraPlus**
3. Click **Network File** and select the network (.pb) file.



Project: Hand-Gesture
Directory: /tmp
Framework: TensorFlow
Device: Ultra Plus
Class: CNN
Input Files: /tmp/model_frozenforInference.pb
Output Files:
Hand-Gesture.yml
Hand-Gesture.lscml
Hand-Gesture.bin
Post Processing(Optional)

Figure 6.2. SensAI – Network File Selection

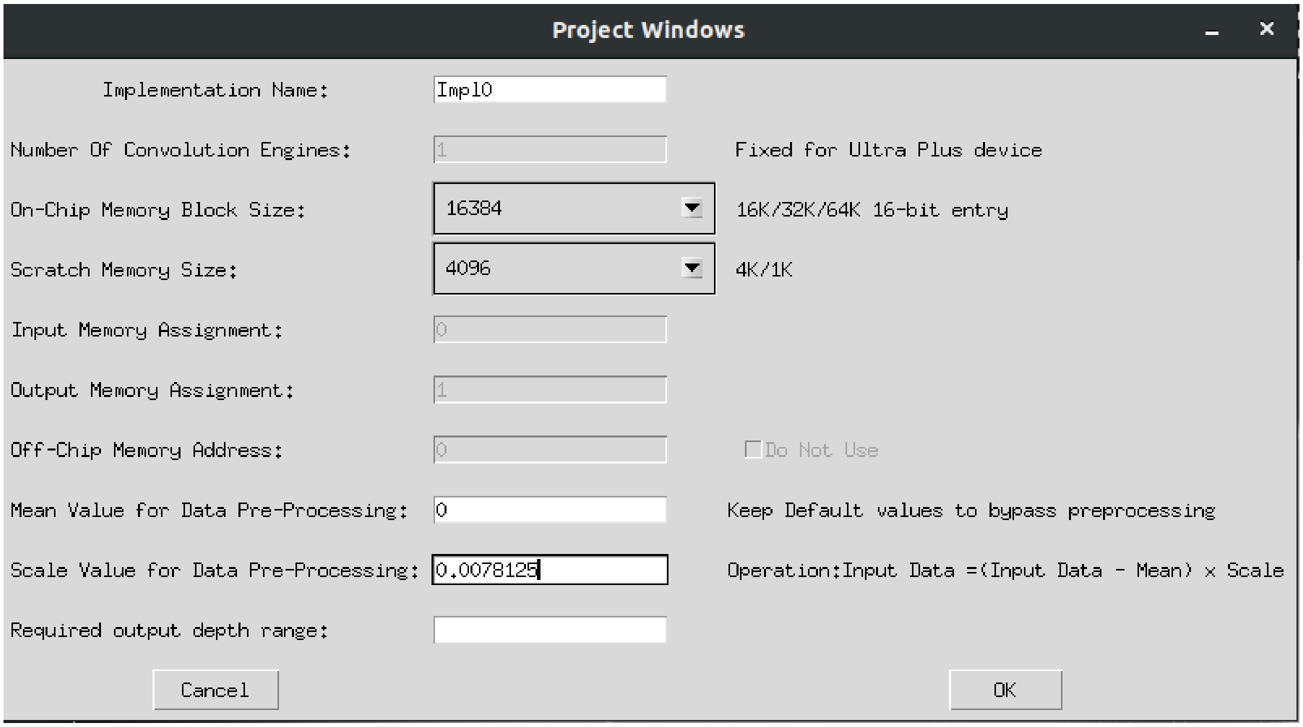
4. Click **Image/Video/Audio Data** and select the image input file.



Project: Hand-Gesture
Directory: /tmp
Framework: TensorFlow
Device: Ultra Plus
Class: CNN
Input Files:
/tmp/model_frozenforInference.pb
/tmp/0-10-11_C1.png
Output Files:
Hand-Gesture.yml
Hand-Gesture.lscml
Hand-Gesture.bin
Post Processing(Optional)

Figure 6.3. SensAI – Image Data File Selection

5. Click **NEXT**.
6. Configure your project settings.
    a. **Mean Value for Data Pre-Processing – 0**
    b. **Scale Value for Data Pre-Processing – 0.0078125**



Implementation Name: Impl0

Number Of Convolution Engines: 1 — Fixed for Ultra Plus device

On-Chip Memory Block Size: 16384 — 16K/32K/64K 16-bit entry

Scratch Memory Size: 4096 — 4K/1K

Input Memory Assignment: 0

Output Memory Assignment: 1

Off-Chip Memory Address: 0 — □ Do Not Use

Mean Value for Data Pre-Processing: 0 — Keep Default values to bypass preprocessing

Scale Value for Data Pre-Processing: 0.0078125 — Operation:Input Data =(Input Data - Mean) x Scale

Required output depth range:

Figure 6.4. SensAI – Project Settings

7. Click **OK** to create the project.
8. Double-click **Analyze**.

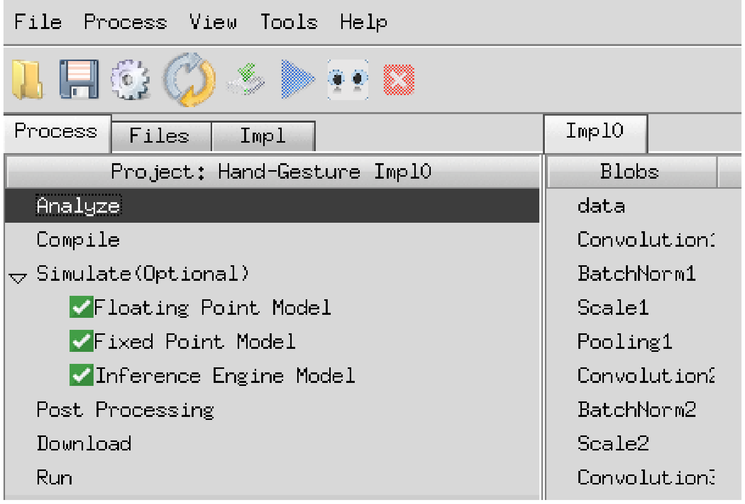
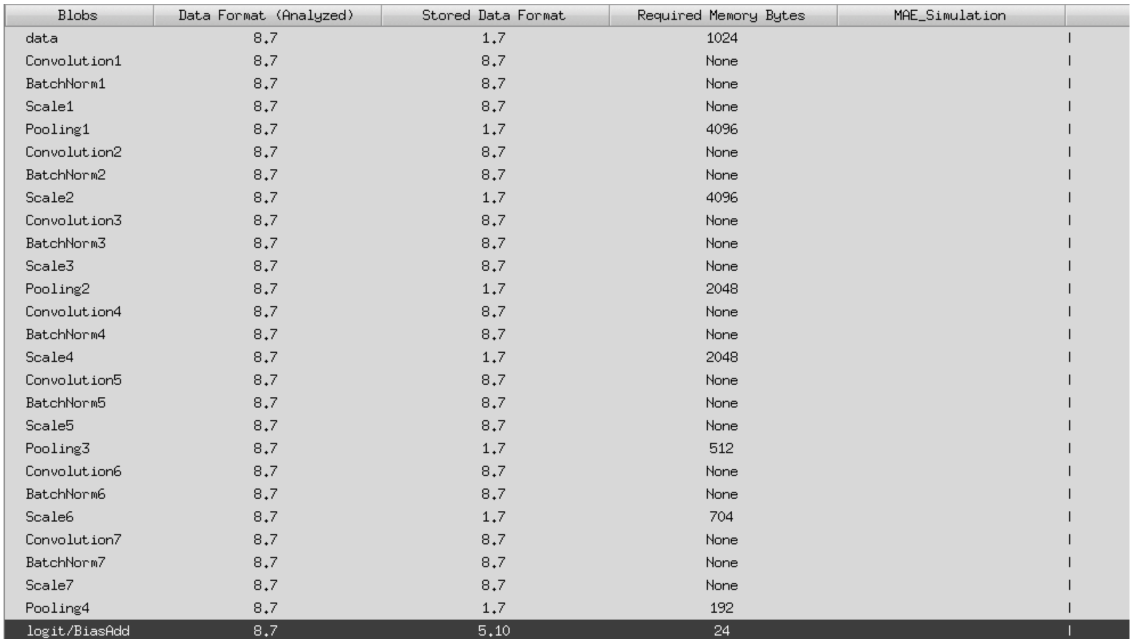


Figure 6.5. SensAI – Analyze Project

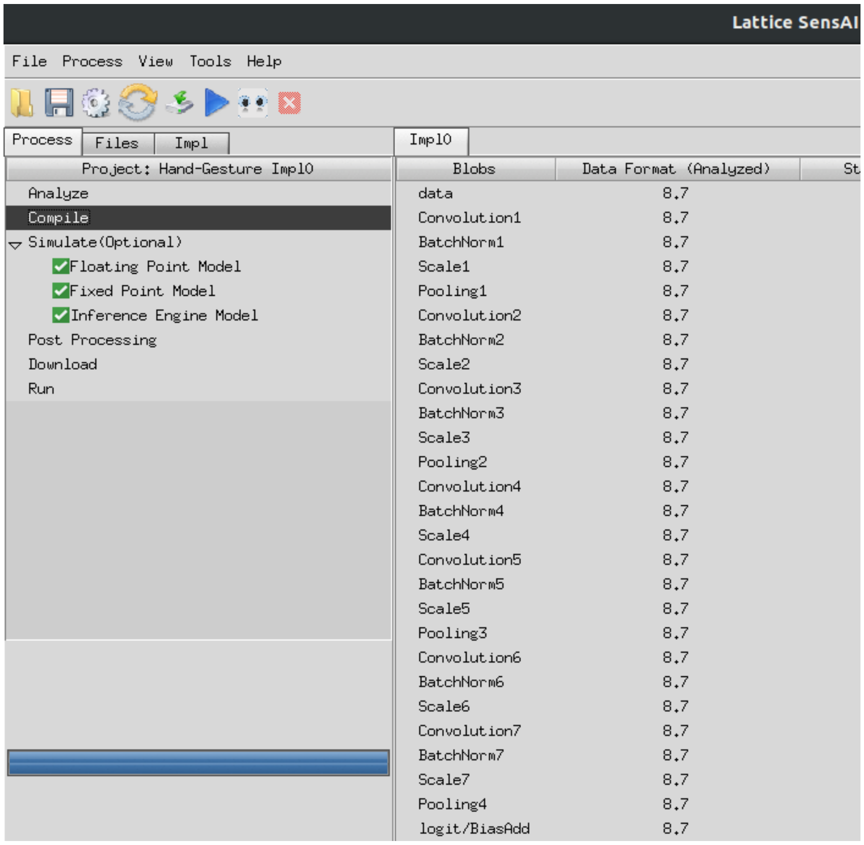
9. Confirm the Q format of each layer as shown in Figure 6.6.



Blobs	Data Format (Analyzed)	Stored Data Format	Required Memory Bytes	MAE_Simulation
data	8.7	1.7	1024	
Convolution1	8.7	8.7	None	
BatchNorm1	8.7	8.7	None	
Scale1	8.7	8.7	None	
Pooling1	8.7	1.7	4096	
Convolution2	8.7	8.7	None	
BatchNorm2	8.7	8.7	None	
Scale2	8.7	1.7	4096	
Convolution3	8.7	8.7	None	
BatchNorm3	8.7	8.7	None	
Scale3	8.7	8.7	None	
Pooling2	8.7	1.7	2048	
Convolution4	8.7	8.7	None	
BatchNorm4	8.7	8.7	None	
Scale4	8.7	1.7	2048	
Convolution5	8.7	8.7	None	
BatchNorm5	8.7	8.7	None	
Scale5	8.7	8.7	None	
Pooling3	8.7	1.7	512	
Convolution6	8.7	8.7	None	
BatchNorm6	8.7	8.7	None	
Scale6	8.7	1.7	704	
Convolution7	8.7	8.7	None	
BatchNorm7	8.7	8.7	None	
Scale7	8.7	8.7	None	
Pooling4	8.7	1.7	192	
logit/BiasAdd	8.7	5.10	24	

Figure 6.6. Q Format Settings for Each Layer

10. Double-click **Compile** to generate the firmware and filter binary file.



Project: Hand-Gesture Impl0
- Analyze
- Compile
- Simulate(Optional)
  - ☑ Floating Point Model
  - ☑ Fixed Point Model
  - ☑ Inference Engine Model
- Post Processing
- Download
- Run

Blobs	Data Format (Analyzed)	St
data	8.7	
Convolution1	8.7	
BatchNorm1	8.7	
Scale1	8.7	
Pooling1	8.7	
Convolution2	8.7	
BatchNorm2	8.7	
Scale2	8.7	
Convolution3	8.7	
BatchNorm3	8.7	
Scale3	8.7	
Pooling2	8.7	
Convolution4	8.7	
BatchNorm4	8.7	
Scale4	8.7	
Convolution5	8.7	
BatchNorm5	8.7	
Scale5	8.7	
Pooling3	8.7	
Convolution6	8.7	
BatchNorm6	8.7	
Scale6	8.7	
Convolution7	8.7	
BatchNorm7	8.7	
Scale7	8.7	
Pooling4	8.7	
logit/BiasAdd	8.7	

Figure 6.7. Compile Project

11. The Firmware bin file location is displayed in the compilation log. Use the generated firmware bin on hardware for testing.

7. Hardware Implementation

7.1. Top Level Information

7.1.1. Block Diagram

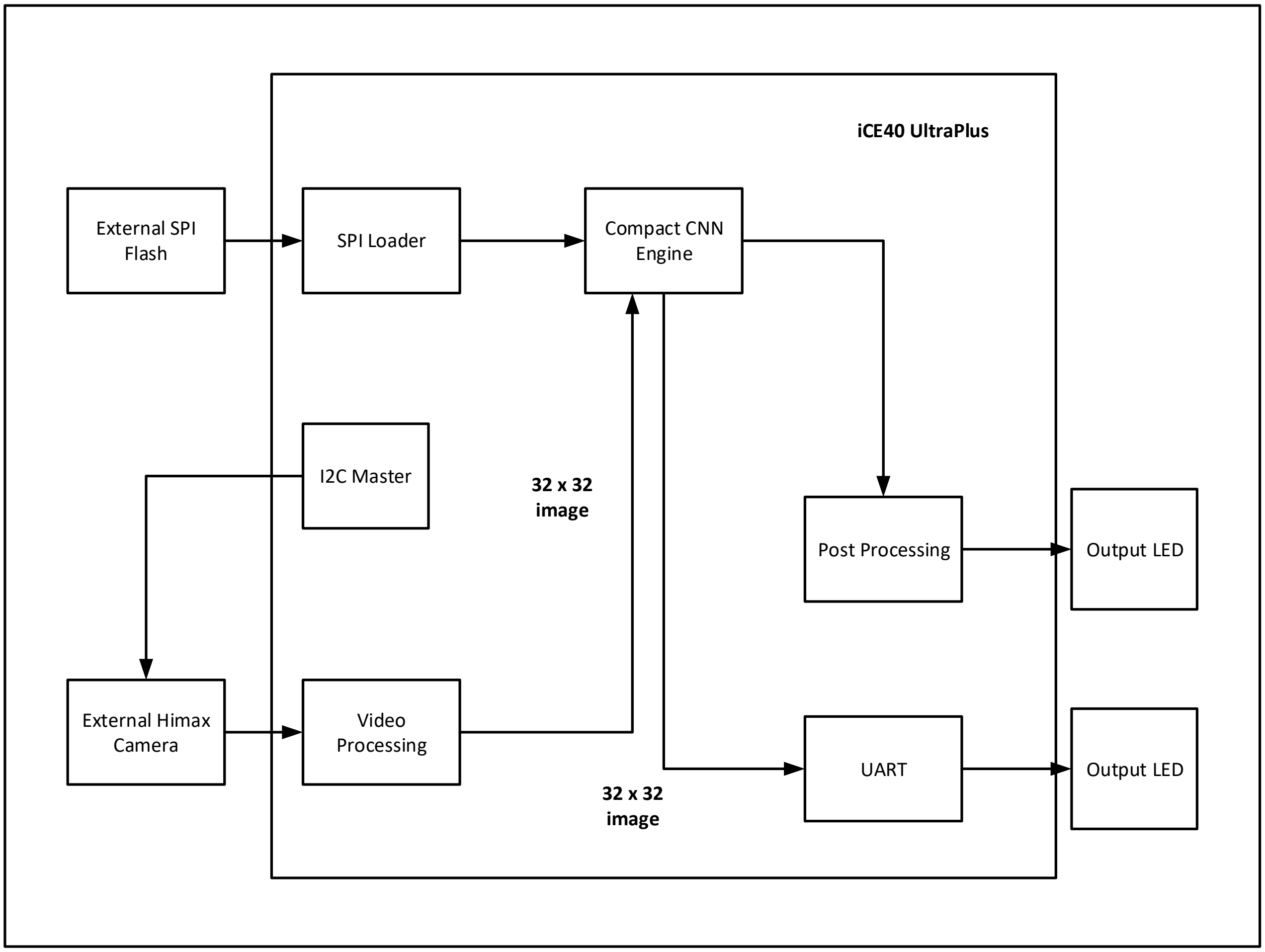


Figure 7.1. Himax Hand Gesture Detection Using iCE40 UltraPlus Block Diagram

7.1.2. Operational Flow

This section provides an overview of the data flow across the Himax UPduino board.

- The Compact CNN module is configured with the firmware (BIN) file from the external SPI Flash through the spi_loader_wrap module. The BIN file is a command sequence code, which is generated by the Lattice Machine Learning software tool.
- The external Himax HM01B0 imaging camera is configured with the *lsc_i2cm_himax* module. The grayscale captured by the camera image is sent to the iCE40 UltraPlus device.
- When the imaging camera is active, strobe control is used for IR LEDs through the *strobe_ctl* module.
- The image data is then downscaled to 32 x 32 image resolution by the *ice40_himax_video_process_128_32_wide_br* module to make it compatible for the Compact CNN engine input resolution. This data is written into the internal memory block of the Compact CNN Accelerator IP Core through the input data ports.

- By using the loaded firmware and the downscaled input image, the CNN engine performs inference and generates output.
- The Compact CNN result data (o_dout) is given to the post processing module *signdet_post*, which provides the valid index value for the detected hand gesture.
- The index value from the *signdet_post* module is then used in the top module to drive the output debug LEDs (DS1 to DS4) and RGB LED on board. Also, the LED DS5 represents that the CNN engine is enabled and LED DS6 represents that a hand gesture is detected.
- The 32 x 32 gray scale image used for display is obtained as valid debug data from the CNN engine.
- This grayscale image of hand gesture and its corresponding index value can be observed in the Lattice UART Display Software through the *lsc_uart* module.

7.1.3. Core Customization

Table 7.1. Core Parameter

Parameter	Default (Decimal)	Description
Configurable Parameter		
BYTE_MODE	UNSIGNED	Configured for CNN input data layer width The following are the possible configurations: UNSIGNED – The data is directly passed to CNN input for unsigned 8-bit input data layer. SIGNED – 128 is subtracted from the data for signed 8-bit input data layer of CNN. DISABLED – Disable byte mode
Non-Configurable Parameter		
USE_ML	1	Enable ML engine
EN_I2CS	0	Used to instantiate I^{2}C slave for control and debugging
EN_UART	1	Used to instantiate UART for video output EN_CLKMASK parameter of ice40_himax_signdet_clkgen module must be 0 in order to enable EN_UART.
EN_DUAL_UART	1	Used to get Wired AND connection for UART signal
EN_SEQ	0	Sequence mode for CNN input Data
EN_STROBE_CTL	1	Enable strobe control for power reduction
EN_FILTER	1	Enable to capture filtered maximum index value
EN_ONEHOT	0	If 1, drive one hot LED output. If 0, drive 4-bit BCD representation LED output.
CODE_MEM	TRI_SPRAM	Type of Memory utilized Other possible configurations are EBRAM, SINGLE_SPRAM, and DUAL_SPRAM.
LCD_TYPE	OLED	Used OLED as output type Other possible configuration is LCD.
ROTATE	0	No rotation used to readout post processing result Other possible values are 90, 180, and 270 degrees.

7.2. Architectural Details

7.2.1. Pre-Processing CNN

The captured grayscale camera image is sent to the *ice40_himax_video_process_128_32_wide_br* module from the top.

The *ice40_himax_video_process_128_32_wide_br module* processes that image data and generates input of 32 x 32 grayscale image data for the Compact CNN IP. The pre-processing flow is described below.

- Image data values are fed serially line by line for an image frame of 640 x 320 pixels.
- Mask parameters are set to mask out the boundary area of 640 x 320 pixels to 512 x 256 pixels as shown in Figure 7.2. This 512 x 256 image data is then downscaled into 32 x 32 image data (1024 pixels) for the CNN engine.
- As shown in Figure 7.3, every 16 horizontal pixels (512/32) and 8 vertical pixels (256/32), which make a pixel block of 16 x 8 are accumulated into a single pixel.
- The foreground detection process is used to extract the changes from moving images as shown in Figure 7.4.
- The downscaled 1024 image values of the current frame are stored and used as background image (*rd_pixel_bg*).
- When the new frame arrives, the newly accumulated 1024 values are also stored (*accu_mod*). Both old and new values are compared. If the new value is greater than the previously stored background value, the background value is subtracted from the new value and stored (*accu_mod_br*). After processing, all these 1024 values are sent to CNN for inference.

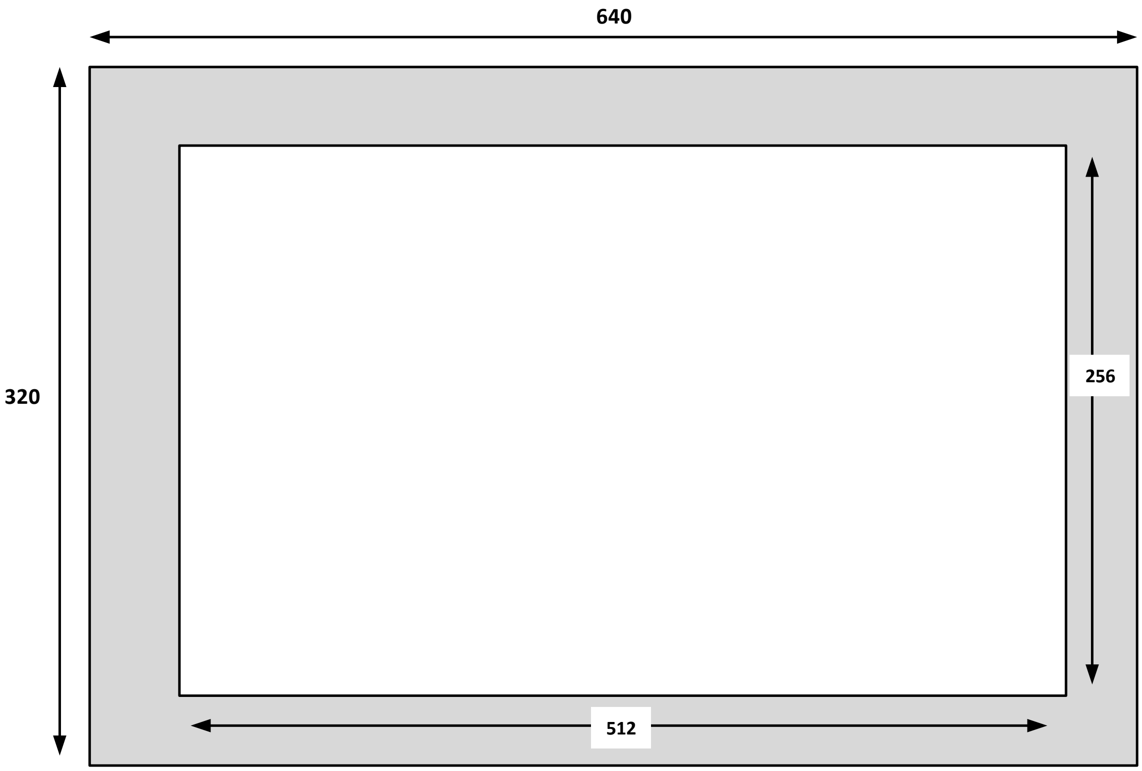


Figure 7.2. Masking and Cropping Image

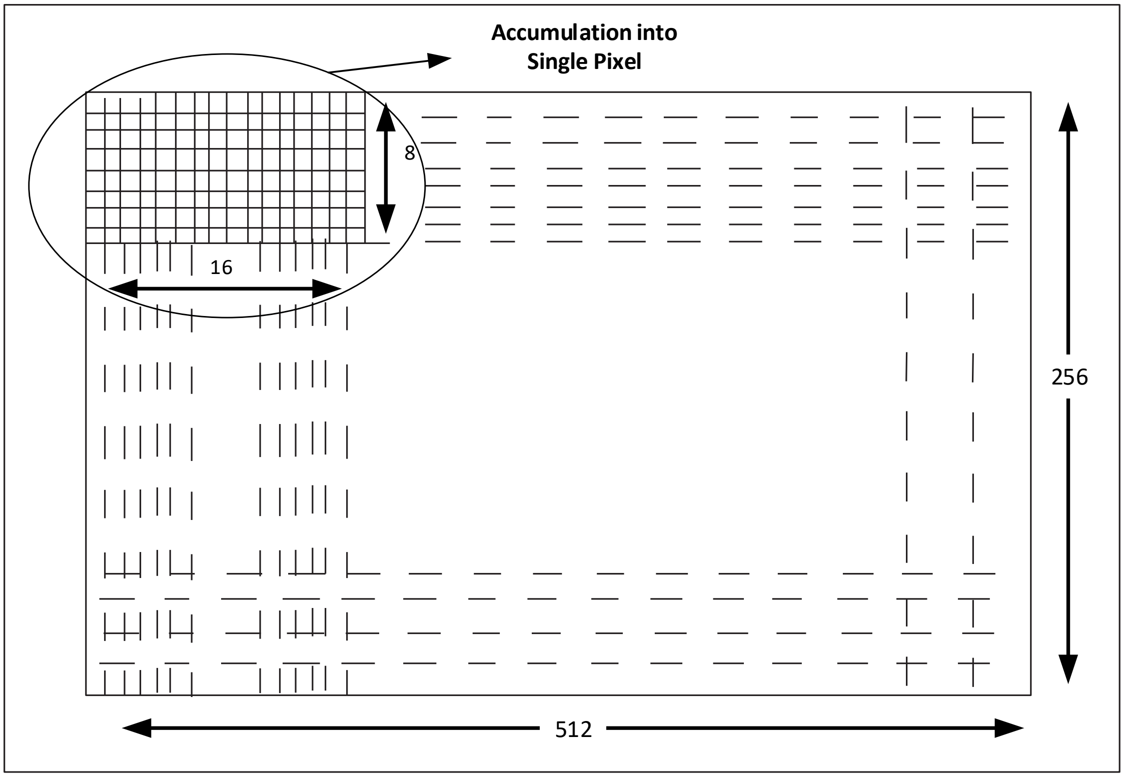
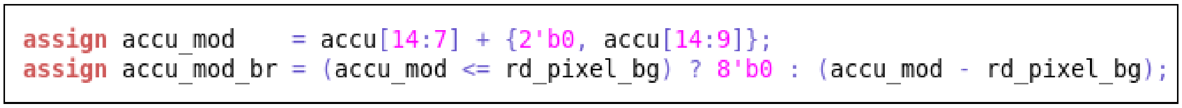


Figure 7.3. Downscaling Image



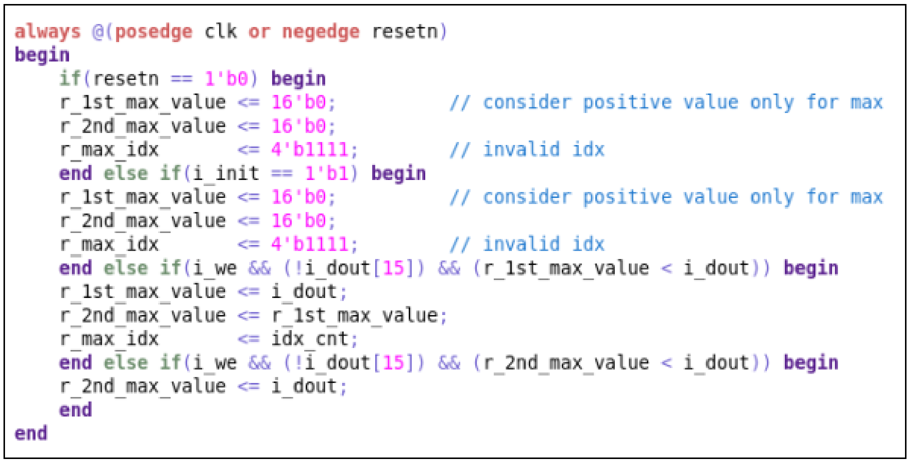
```
assign accu_mod    = accu[14:7] + {2'b0, accu[14:9]};
assign accu_mod_br = (accu_mod <= rd_pixel_bg) ? 8'b0 : (accu_mod - rd_pixel_bg);
```

Figure 7.4. Foreground Detection from Downscaled Image

7.2.2. Post-Processing CNN

Post-processing is discussed below:

- For this demo, the CNN engine gives probability values for detected hand gestures to the *signdet_post* module.
- The *signdet_post* module finds out the index value (r_max_idx) of the detected hand gesture, as shown in Figure 7.5 by implementing an up-counter for index count (idx_cnt) until the CNN result is available.
- The maximum value is considered valid only if the CNN output is a positive value, that is, SIGN bit (Highest bit) is 0.



```
always @(posedge clk or negedge resetn)
begin
    if(resetn == 1'b0) begin
    r_1st_max_value <= 16'b0;          // consider positive value only for max
    r_2nd_max_value <= 16'b0;
    r_max_idx       <= 4'b1111;        // invalid idx
    end else if(i_init == 1'b1) begin
    r_1st_max_value <= 16'b0;          // consider positive value only for max
    r_2nd_max_value <= 16'b0;
    r_max_idx       <= 4'b1111;        // invalid idx
    end else if(i_we && (!i_dout[15]) && (r_1st_max_value < i_dout)) begin
    r_1st_max_value <= i_dout;
    r_2nd_max_value <= r_1st_max_value;
    r_max_idx       <= idx_cnt;
    end else if(i_we && (!i_dout[15]) && (r_2nd_max_value < i_dout)) begin
    r_2nd_max_value <= i_dout;
    end
end
```

Figure 7.5. Logic to Obtain Maximum Index of Detected Gesture

- After all CNN outputs are processed, the valid maximum index value (o_max_idx) is obtained. This value is passed to the top module, which is mainly used for three operations:
  - To drive the output and debug the LEDs (DS1, DS2, DS3, and DS4) on the HM01B0 Adapter board in 4-bit BCD format.
  - To drive the RGB LED on the UPduino board through the *RGB_DRIVER* module.
  - To observe the index value of detected gesture in the Lattice UART display software through the *lsc_uart* module.

7.2.3. UART Operation for Display

This section explains the UART data transfer for Lattice UART Display Software.

- The UART communication is mentored by *lsc_uart* block configured at a Baud rate of 230400 in top module.
- The Lattice UART display software operates with four different modes. Each mode is identified by an ASCII character as shown Table 7.2.

Table 7.2. UART Display Modes and Description

Mode	Character	ASCII Value	Description
Image	h	0x68	Obtain the grayscale downscaled image in display
Data	i	0x69	Obtain the CNN output result after inference in display
Result	j	0x6A	Obtain the index value of detected hand gesture in display
Image +	k	0x6B	Obtain the grayscale downscaled image and its index value in display

- The *lsc_uart* block provides the required UART data for display by identifying the UART mode as shown in Figure 7.6.

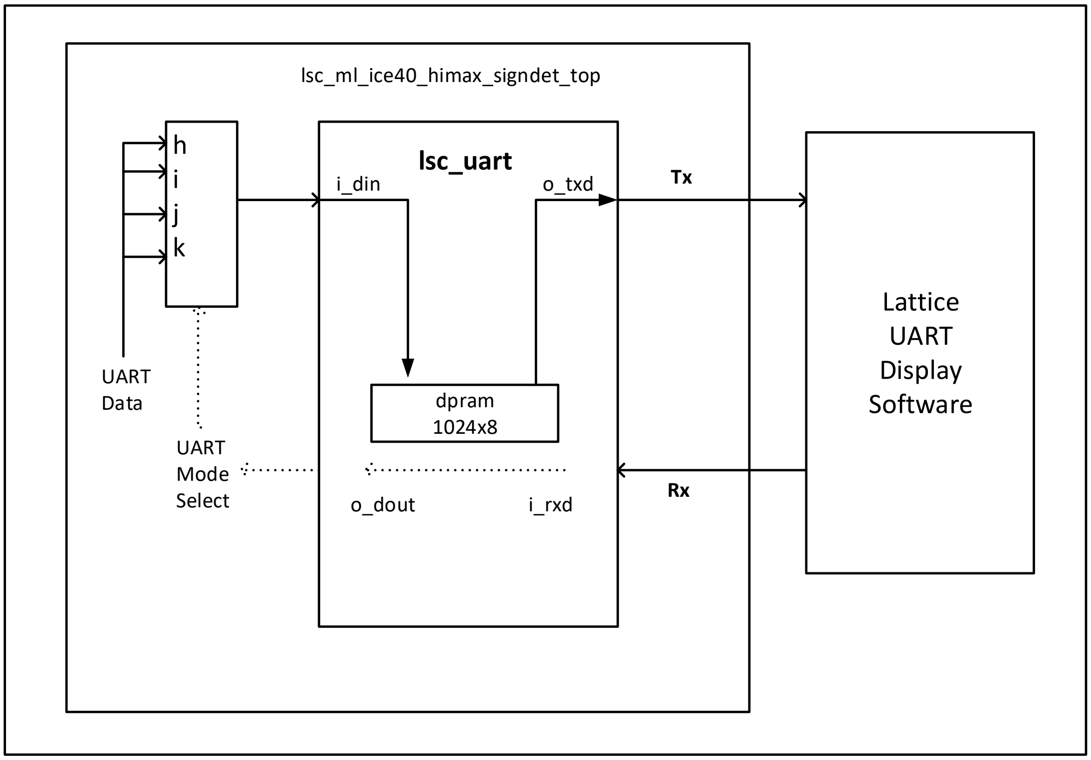
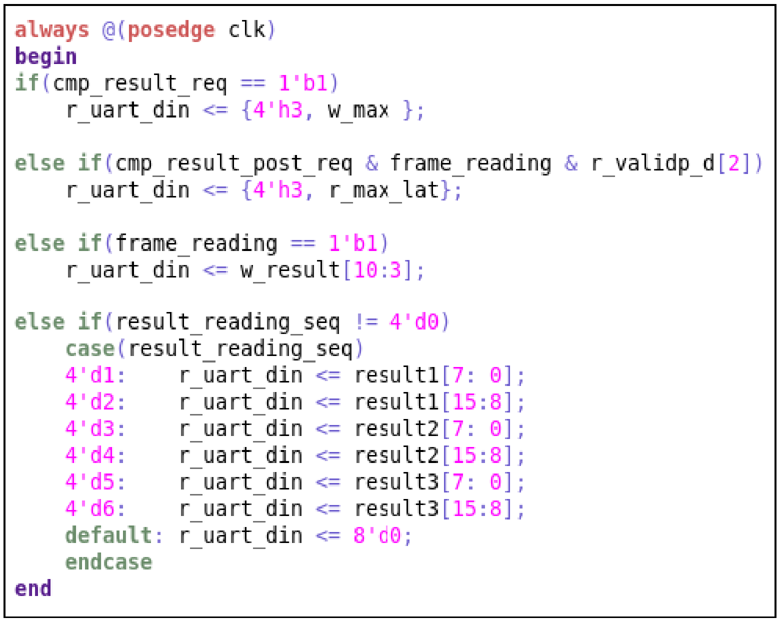


Figure 7.6. UART Operation for Lattice UART Display Software

- The *w_uart_dout* output processed from the *i_rxd* data is used to identify the modes in the top module as follows:
  - If w_uart_dout [2:0] is 0, it indicates Character h.So the UART mode is Image.
  - If w_uart_dout [2:0] is 1, it indicates Character i. So the UART mode is Data.
  - If w_uart_dout [2:0] is 2, it indicates Character j. So the UART mode is Result.
  - If w_uart_dout [2:0] is 3, it indicates Character k.So the UART mode is Image+.
- The logic to obtain the UART data for an identified mode is shown in Figure 7.7.



```
always @(posedge clk)
begin
if(cmp_result_req == 1'b1)
    r_uart_din <= {4'h3, w_max };

else if(cmp_result_post_req & frame_reading & r_validp_d[2])
    r_uart_din <= {4'h3, r_max_lat};

else if(frame_reading == 1'b1)
    r_uart_din <= w_result[10:3];

else if(result_reading_seq != 4'd0)
    case(result_reading_seq)
    4'd1:    r_uart_din <= result1[7: 0];
    4'd2:    r_uart_din <= result1[15:8];
    4'd3:    r_uart_din <= result2[7: 0];
    4'd4:    r_uart_din <= result2[15:8];
    4'd5:    r_uart_din <= result3[7: 0];
    4'd6:    r_uart_din <= result3[15:8];
    default: r_uart_din <= 8'd0;
    endcase
end
```

Figure 7.7. Logic for UART Display Data

- This 8-bit UART data *r_uart_din* latched for its respective mode is sent back to the *lsc_uart* module through the *i_din* port, which is then stored in a local DPRAM as shown in Figure 7.6.
- When the DPRAM is not empty, the data is sent out on the *o_txd* port of the *lsc_uart* module in the UART data packet format consisting of a Start Bit (0), followed by the 8-bit data read out from the DPRAM, and a Stop Bit (1).
- Finally, the output data packet coming from the *o_txd* is sent serially to the Lattice UART Display software through SPI Interface output (spi_mosi).
- The downscaled 32 x 32 image is obtained from the CNN as debug data on top when valid debug signal *w_debug_vld* is present. This image data is sent to Lattice UART display software when frame reading is present.
- The values for the data mode are the CNN output results after inference which is sent in two bytes, latched by result1, result2, and result3.

7.2.4. Strobe Control

This section provides an overview of the Strobe control for the IR LEDs on board.

- There are two IR LEDs placed on the left and right side of the image sensor on the board.
- The Strobe control for these IR LEDs is managed by the *strobe_ctl* block in the top module when EN_STROBE_CTL parameter is enabled.
- This module uses object detection *w_obj_det_trig* and *w_det_obj* signals from the *ice40_himax_video_process_128_32_wide_br* module to generate the *o_en_strobe* output.
- The Strobe signal for the LEDs is mainly received from this block if any of the below condition is valid:
  - Image sensor is detecting the object (uses skip counter).
  - Image sensor has captured the object (uses detection counter).
- Condition 1 is valid when no hand gesture is present. Hence, when trigger is present, the image sensor is detecting the object and during this time the IR LEDs can be seen blinking slowly at regular intervals onboard.
- Condition 2 is valid when hand gesture is detected after trigger. So the image sensor is not in search mode *o_search_mode* and the IR LEDs can be seen blinking very quickly on board.
- During both the above conditions, whether it is slow or fast, both LEDs blink simultaneously.
- Instead of keeping the IR LEDs active regularly, the power is saved due to the pulsed ON/OFF control. It helps increase the life span of the LEDs.

8. Creating FPGA Bitstream File

This section describes the steps to compile RTL bitstream using Lattice Radiant tool.

To create the FPGA bitstream file:

1. Open the Lattice Radiant software. Default screen is shown in Figure 8.1.

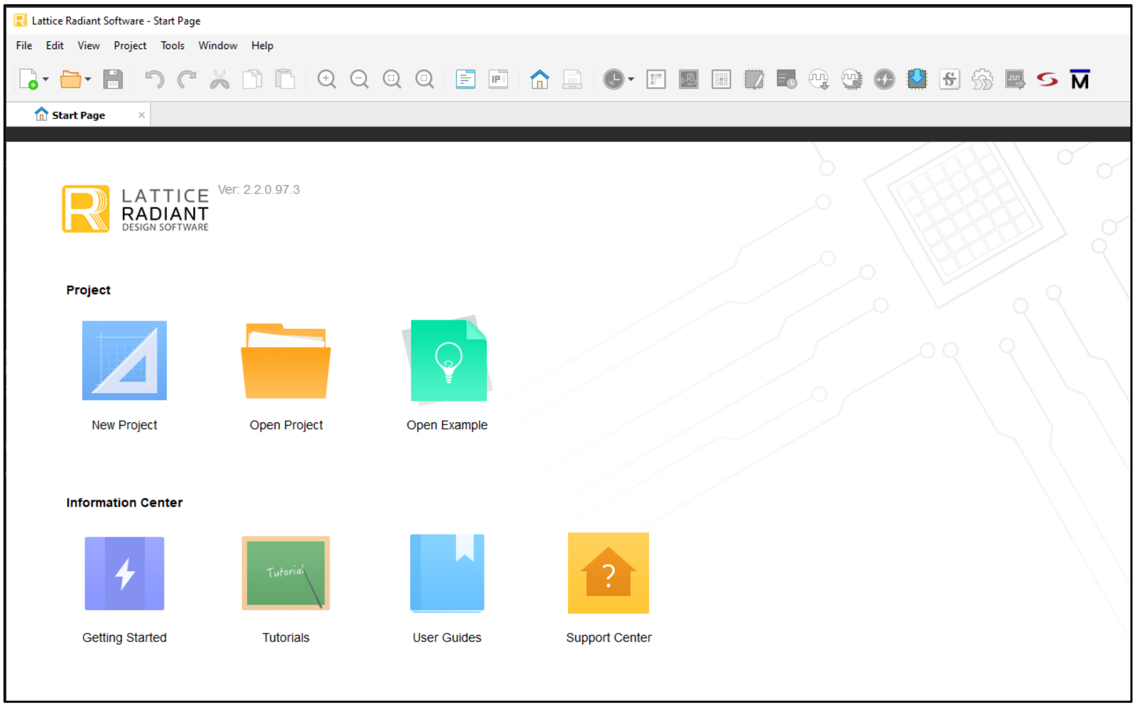
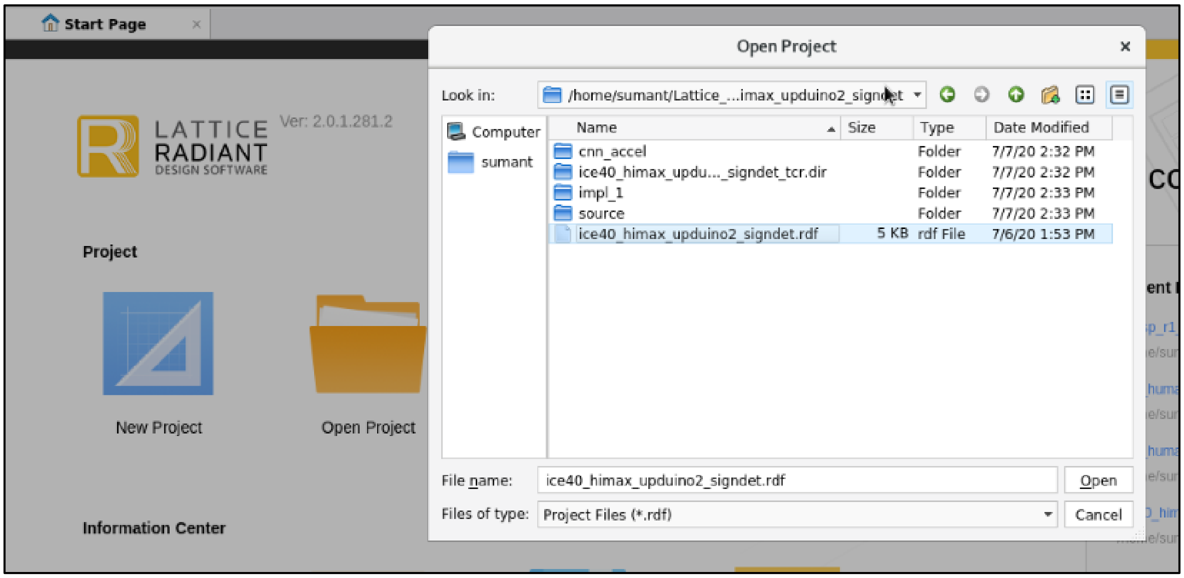


Figure 8.1. Lattice Radiant – Default Screen

2. Go to **File > Open > Project**.
3. Open the Lattice Radiant project file *ice40_himax_upduino2_signdet.rdf*. As shown in Figure 8.2, you can also open the project by selecting the yellow folder shown in the user interface.



Look in: /home/sumant/Lattice_...imax_upduino2_signdet

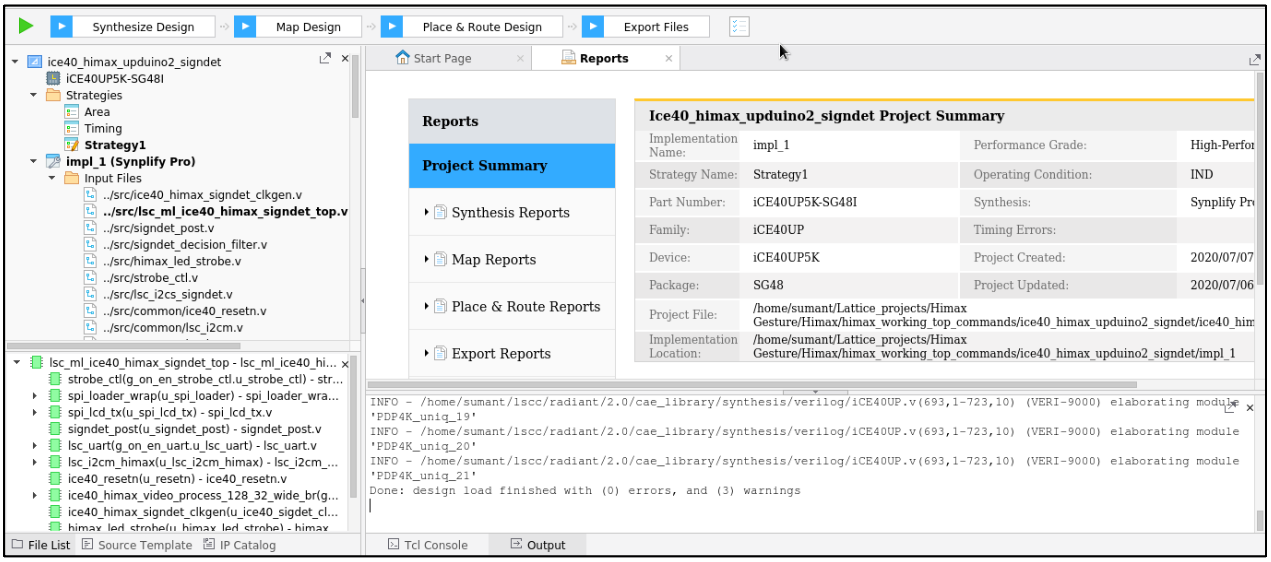
Name	Size	Type	Date Modified
cnn_accel		Folder	7/7/20 2:32 PM
ice40_himax_updu..._signdet_tcr.dir		Folder	7/7/20 2:32 PM
impl_1		Folder	7/7/20 2:33 PM
source		Folder	7/7/20 2:33 PM
ice40_himax_upduino2_signdet.rdf	5 KB	rdf File	7/6/20 1:53 PM

File name: ice40_himax_upduino2_signdet.rdf

Files of type: Project Files (*.rdf)

Figure 8.2. Lattice Radiant – Open iCE40 Himax Gesture Detection Project File

4. After opening the project file, check the following points shown in Figure 8.3.
    - The design loaded with zero error message shown in the *Output* window.
    - Check the following information in the *Project Summary* window.
        - **Part Number – iCE40UP5K-SG48I**
        - **Family – iCE40UP**
        - **Device – iCE40UP5K**
        - **Package – SG48**


Synthesize Design
Map Design
Place & Route Design
Export Files
ice40_himax_upduino2_signdet
iCE40UP5K-SG48I
Strategies
Area
Timing
Strategy1
impl_1 (Synplify Pro)
Input Files
../src/ice40_himax_signdet_clkgen.v
../src/lsc_ml_ice40_himax_signdet_top.v
../src/signdet_post.v
../src/signdet_decision_filter.v
../src/himax_led_strobe.v
../src/strobe_ctl.v
../src/lsc_i2cs_signdet.v
../src/common/ice40_resetn.v
../src/common/lsc_i2cm.v
lsc_ml_ice40_himax_signdet_top - lsc_ml_ice40_hi...
strobe_ctl(g_on_en_strobe_ctl.u_strobe_ctl) - str...
spi_loader_wrap(u_spi_loader) - spi_loader_wra...
spi_lcd_tx(u_spi_lcd_tx) - spi_lcd_tx.v
signdet_post(u_signdet_post) - signdet_post.v
lsc_uart(g_on_en_uart.u_lsc_uart) - lsc_uart.v
lsc_i2cm_himax(u_lsc_i2cm_himax) - lsc_i2cm_...
ice40_resetn(u_resetn) - ice40_resetn.v
ice40_himax_video_process_128_32_wide_br(g...
ice40_himax_signdet_clkgen(u_ice40_sigdet_cl...
himax_led_strobe(u_himax_led_strobe) - himax
File List
Source Template
IP Catalog
Start Page
Reports
Reports
Project Summary
Synthesis Reports
Map Reports
Place & Route Reports
Export Reports
Tcl Console
Output

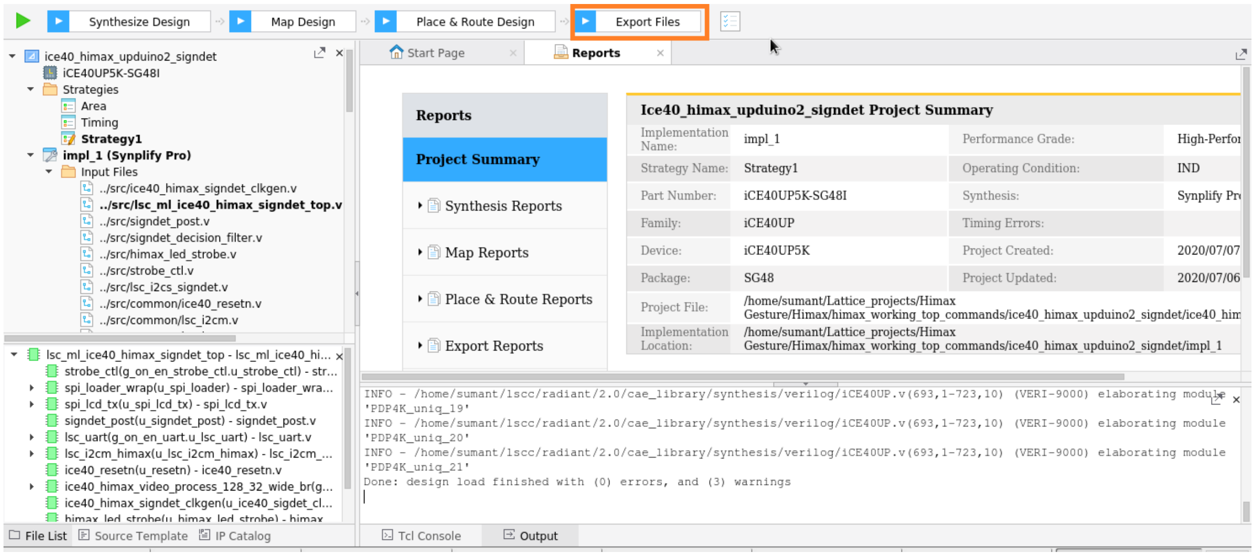
Ice40_himax_upduino2_signdet Project Summary

Implementation Name:	impl_1	Performance Grade:	High-Perfor
Strategy Name:	Strategy1	Operating Condition:	IND
Part Number:	iCE40UP5K-SG48I	Synthesis:	Synplify Pr
Family:	iCE40UP	Timing Errors:	
Device:	iCE40UP5K	Project Created:	2020/07/07
Package:	SG48	Project Updated:	2020/07/06
Project File:	/home/sumant/Lattice_projects/Himax Gesture/Himax/himax_working_top_commands/ice40_himax_upduino2_signdet/ice40_him		
Implementation Location:	/home/sumant/Lattice_projects/Himax Gesture/Himax/himax_working_top_commands/ice40_himax_upduino2_signdet/impl_1		

```
INFO - /home/sumant/lscc/radiant/2.0/cae_library/synthesis/verilog/iCE40UP.v(693,1-723,10) (VERI-9000) elaborating module
'PDP4K_uniq_19'
INFO - /home/sumant/lscc/radiant/2.0/cae_library/synthesis/verilog/iCE40UP.v(693,1-723,10) (VERI-9000) elaborating module
'PDP4K_uniq_20'
INFO - /home/sumant/lscc/radiant/2.0/cae_library/synthesis/verilog/iCE40UP.v(693,1-723,10) (VERI-9000) elaborating module
'PDP4K_uniq_21'
Done: design load finished with (0) errors, and (3) warnings
```

Figure 8.3. Lattice Radiant – Design Check After Loading the Project File

5. If the design is loaded without errors, click the **Export Files** button to trigger bitstream generation as shown in Figure 8.4.


Synthesize Design
Map Design
Place & Route Design
Export Files
ice40_himax_upduino2_signdet
iCE40UP5K-SG48I
Strategies
Area
Timing
Strategy1
impl_1 (Synplify Pro)
Input Files
../src/ice40_himax_signdet_clkgen.v
../src/lsc_ml_ice40_himax_signdet_top.v
../src/signdet_post.v
../src/signdet_decision_filter.v
../src/himax_led_strobe.v
../src/strobe_ctl.v
../src/lsc_i2cs_signdet.v
../src/common/ice40_resetn.v
../src/common/lsc_i2cm.v
lsc_ml_ice40_himax_signdet_top - lsc_ml_ice40_hi...
strobe_ctl(g_on_en_strobe_ctl.u_strobe_ctl) - str...
spi_loader_wrap(u_spi_loader) - spi_loader_wra...
spi_lcd_tx(u_spi_lcd_tx) - spi_lcd_tx.v
signdet_post(u_signdet_post) - signdet_post.v
lsc_uart(g_on_en_uart.u_lsc_uart) - lsc_uart.v
lsc_i2cm_himax(u_lsc_i2cm_himax) - lsc_i2cm_...
ice40_resetn(u_resetn) - ice40_resetn.v
ice40_himax_video_process_128_32_wide_br(g...
ice40_himax_signdet_clkgen(u_ice40_sigdet_cl...
himax_led_strobe(u_himax_led_strobe) - himax
File List
Source Template
IP Catalog
Start Page
Reports
Reports
Project Summary
Synthesis Reports
Map Reports
Place & Route Reports
Export Reports
Tcl Console
Output

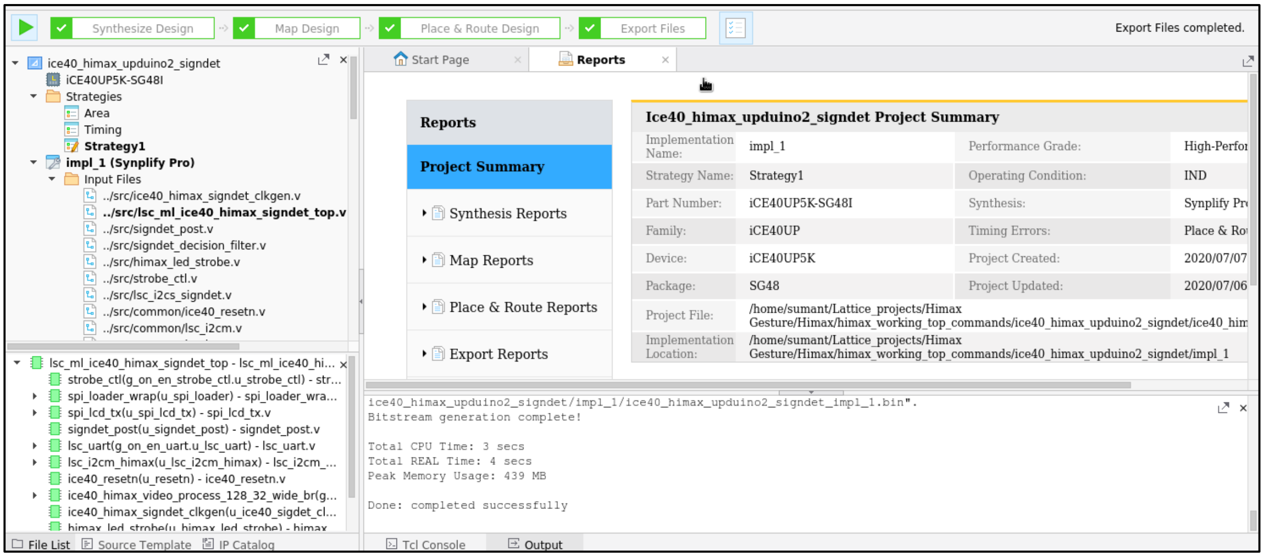
Ice40_himax_upduino2_signdet Project Summary

Implementation Name:	impl_1	Performance Grade:	High-Perfor
Strategy Name:	Strategy1	Operating Condition:	IND
Part Number:	iCE40UP5K-SG48I	Synthesis:	Synplify Pr
Family:	iCE40UP	Timing Errors:	
Device:	iCE40UP5K	Project Created:	2020/07/07
Package:	SG48	Project Updated:	2020/07/06
Project File:	/home/sumant/Lattice_projects/Himax Gesture/Himax/himax_working_top_commands/ice40_himax_upduino2_signdet/ice40_him		
Implementation Location:	/home/sumant/Lattice_projects/Himax Gesture/Himax/himax_working_top_commands/ice40_himax_upduino2_signdet/impl_1		

```
INFO - /home/sumant/lscc/radiant/2.0/cae_library/synthesis/verilog/iCE40UP.v(693,1-723,10) (VERI-9000) elaborating module
'PDP4K_uniq_19'
INFO - /home/sumant/lscc/radiant/2.0/cae_library/synthesis/verilog/iCE40UP.v(693,1-723,10) (VERI-9000) elaborating module
'PDP4K_uniq_20'
INFO - /home/sumant/lscc/radiant/2.0/cae_library/synthesis/verilog/iCE40UP.v(693,1-723,10) (VERI-9000) elaborating module
'PDP4K_uniq_21'
Done: design load finished with (0) errors, and (3) warnings
```

Figure 8.4. Lattice Radiant – Trigger Bitstream Generation

6. The Lattice Radiant tool displays *Saving bit stream in …* message in the **Reports** window. The bitstream is generated at *Implementation Location* shown in Figure 8.5.



```
ice40_himax_upduino2_signdet
    iCE40UP5K-SG48I
  Strategies
      Area
      Timing
      Strategy1
  impl_1 (Synplify Pro)
    Input Files
        ../src/ice40_himax_signdet_clkgen.v
        ../src/lsc_ml_ice40_himax_signdet_top.v
        ../src/signdet_post.v
        ../src/signdet_decision_filter.v
        ../src/himax_led_strobe.v
        ../src/strobe_ctl.v
        ../src/lsc_i2cs_signdet.v
        ../src/common/ice40_resetn.v
        ../src/common/lsc_i2cm.v
```

```
lsc_ml_ice40_himax_signdet_top - lsc_ml_ice40_hi...
    strobe_ctl(g_on_en_strobe_ctl.u_strobe_ctl) - str...
    spi_loader_wrap(u_spi_loader) - spi_loader_wra...
    spi_lcd_tx(u_spi_lcd_tx) - spi_lcd_tx.v
    signdet_post(u_signdet_post) - signdet_post.v
    lsc_uart(g_on_en_uart.u_lsc_uart) - lsc_uart.v
    lsc_i2cm_himax(u_lsc_i2cm_himax) - lsc_i2cm_...
    ice40_resetn(u_resetn) - ice40_resetn.v
    ice40_himax_video_process_128_32_wide_br(g...
    ice40_himax_signdet_clkgen(u_ice40_sigdet_cl...
    himax_led_strobe(u_himax_led_strobe) - himax...
```

Reports

Project Summary

- Synthesis Reports
- Map Reports
- Place & Route Reports
- Export Reports

Ice40_himax_upduino2_signdet Project Summary

Implementation Name:	impl_1	Performance Grade:	High-Perfor
Strategy Name:	Strategy1	Operating Condition:	IND
Part Number:	iCE40UP5K-SG48I	Synthesis:	Synplify Pr
Family:	iCE40UP	Timing Errors:	Place & Ro
Device:	iCE40UP5K	Project Created:	2020/07/07
Package:	SG48	Project Updated:	2020/07/06
Project File:	/home/sumant/Lattice_projects/Himax Gesture/Himax/himax_working_top_commands/ice40_himax_upduino2_signdet/ice40_him		
Implementation Location:	/home/sumant/Lattice_projects/Himax Gesture/Himax/himax_working_top_commands/ice40_himax_upduino2_signdet/impl_1		

```
ice40_himax_upduino2_signdet/impl_1/ice40_himax_upduino2_signdet_impl_1.bin".
Bitstream generation complete!

Total CPU Time: 3 secs
Total REAL Time: 4 secs
Peak Memory Usage: 439 MB

Done: completed successfully
```

Figure 8.5. Lattice Radiant – Bit Generation Report Window

9. Programming the Demo

9.1. Functional Description

Figure 9.1 shows the diagram of the hand gesture demo. The microphone captures audio and sends it to the iCE40 UltraPlus device. The iCE40 UltraPlus device then uses the audio data with the firmware file from the external SPI Flash to determine the outcome.

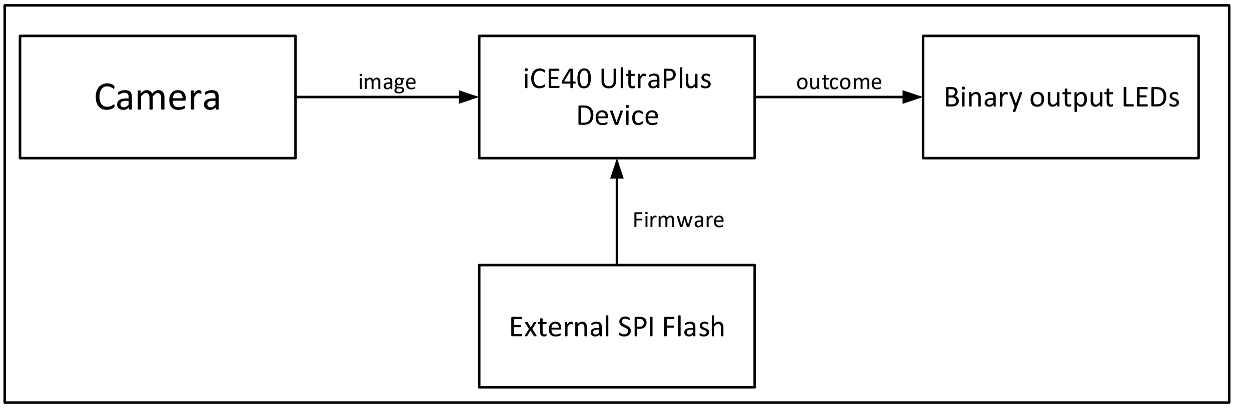


Figure 9.1. iCE40 Hand Gesture Demo Diagram

9.2. Programming the Hand Gesture Recognition on iCE40 UltraPlus SPI Flash

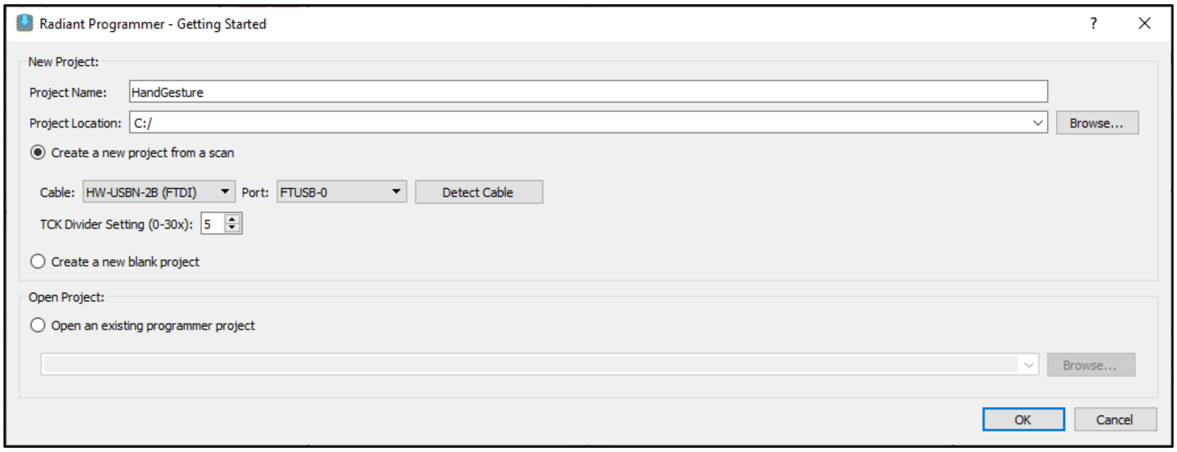
This section provides the procedure for programming the SPI Flash on the Himax HM01B0 UPduino Shield board.

There are two different files that should be programmed into the SPI Flash. These files are programmed to the same SPI Flash, but at different addresses:

- Bitstream
- Firmware

To program the SPI Flash in Radiant Programmer:

1. Connect the Himax HM01B0 UPduino Shield board to the PC using a micro USB cable. Please note that the USB connector onboard is delicate so handle it with care.
2. Start Radiant Programmer. In the **Getting Started** dialog box, select **Create a new blank project**.



Radiant Programmer - Getting Started

New Project:

Project Name: HandGesture

Project Location: C:/

● Create a new project from a scan

Cable: HW-USBN-2B (FTDI) Port: FTUSB-0

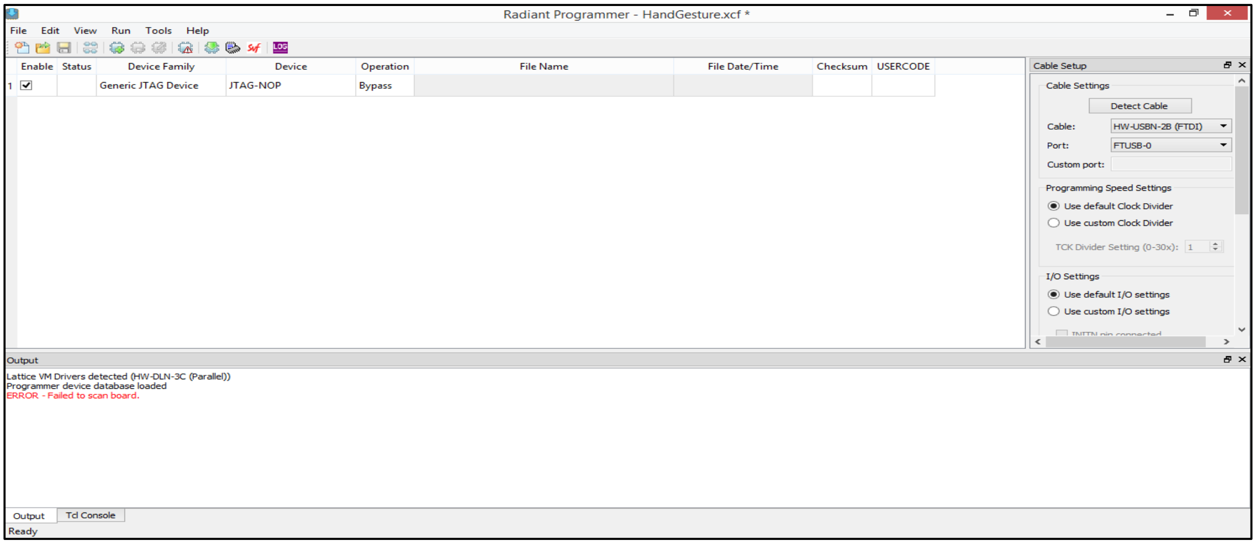
TCK Divider Setting (0-30x): 5

○ Create a new blank project

Open Project:

○ Open an existing programmer project

Figure 9.2. Radiant Programmer – Default Screen


Radiant Programmer - HandGesture.xcf *
File Edit View Run Tools Help
Cable Setup
Output
Tcl Console

	Enable	Status	Device Family	Device	Operation	File Name	File Date/Time	Checksum	USERCODE
1	☑		Generic JTAG Device	JTAG-NOP	Bypass				

Cable Settings

Detect Cable

Cable: HW-USBN-2B (FTDI)

Port: FTUSB-0

Custom port:

Programming Speed Settings

● Use default Clock Divider

○ Use custom Clock Divider

TCK Divider Setting (0-30x): 1

I/O Settings

● Use default I/O settings

○ Use custom I/O settings

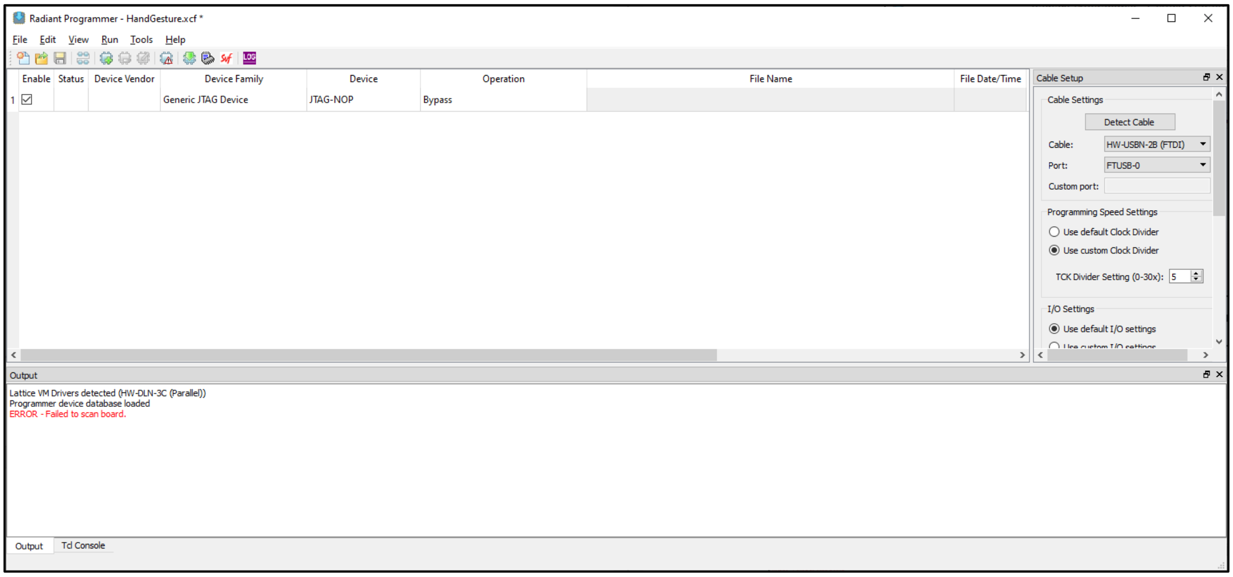
□ INITN pin connected

```
Lattice VM Drivers detected (HW-DLN-3C (Parallel))
Programmer device database loaded
ERROR - Failed to scan board.
```

Ready

Figure 9.3. Radiant Programmer – Initial Project Window

3. Click **OK**.
4. In the Radiant Programmer main interface, select **iCE40 UltraPlus** for **Device Family** and **iCE40UP5K** for **Device** as shown in Figure 9.4.


Radiant Programmer - HandGesture.xcf *
File Edit View Run Tools Help
Cable Setup
Output
Tcl Console

	Enable	Status	Device Vendor	Device Family	Device	Operation	File Name	File Date/Time
1	☑			Generic JTAG Device	JTAG-NOP	Bypass		

Cable Settings

Detect Cable

Cable: HW-USBN-2B (FTDI)

Port: FTUSB-0

Custom port:

Programming Speed Settings

○ Use default Clock Divider

● Use custom Clock Divider

TCK Divider Setting (0-30x): 5

I/O Settings

● Use default I/O settings

○ Use custom I/O settings

```
Lattice VM Drivers detected (HW-DLN-3C (Parallel))
Programmer device database loaded
ERROR - Failed to scan board.
```

Figure 9.4. Diamond Programmer – Device Selection

5. Right-click and select **Device Properties**.

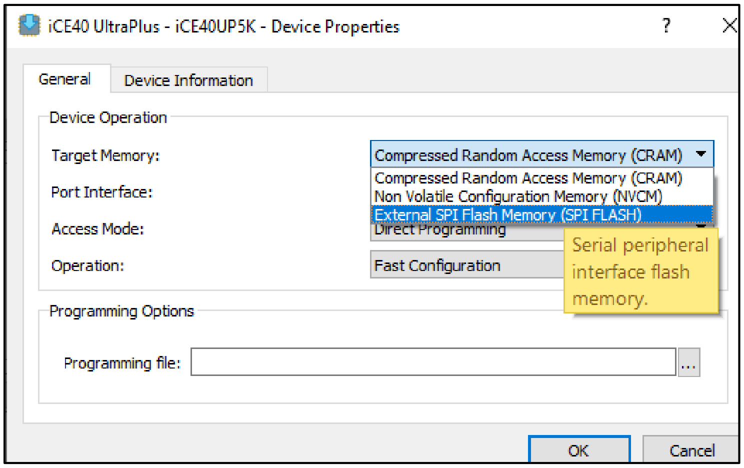

iCE40 UltraPlus - iCE40UP5K - Device Properties
?
General
Device Information
Device Operation
Target Memory:
Compressed Random Access Memory (CRAM)
Compressed Random Access Memory (CRAM)
Non Volatile Configuration Memory (NVCM)
External SPI Flash Memory (SPI FLASH)
Port Interface:
Access Mode:
Direct Programming
Operation:
Fast Configuration
Serial peripheral interface flash memory.
Programming Options
Programming file:
...
OK
Cancel

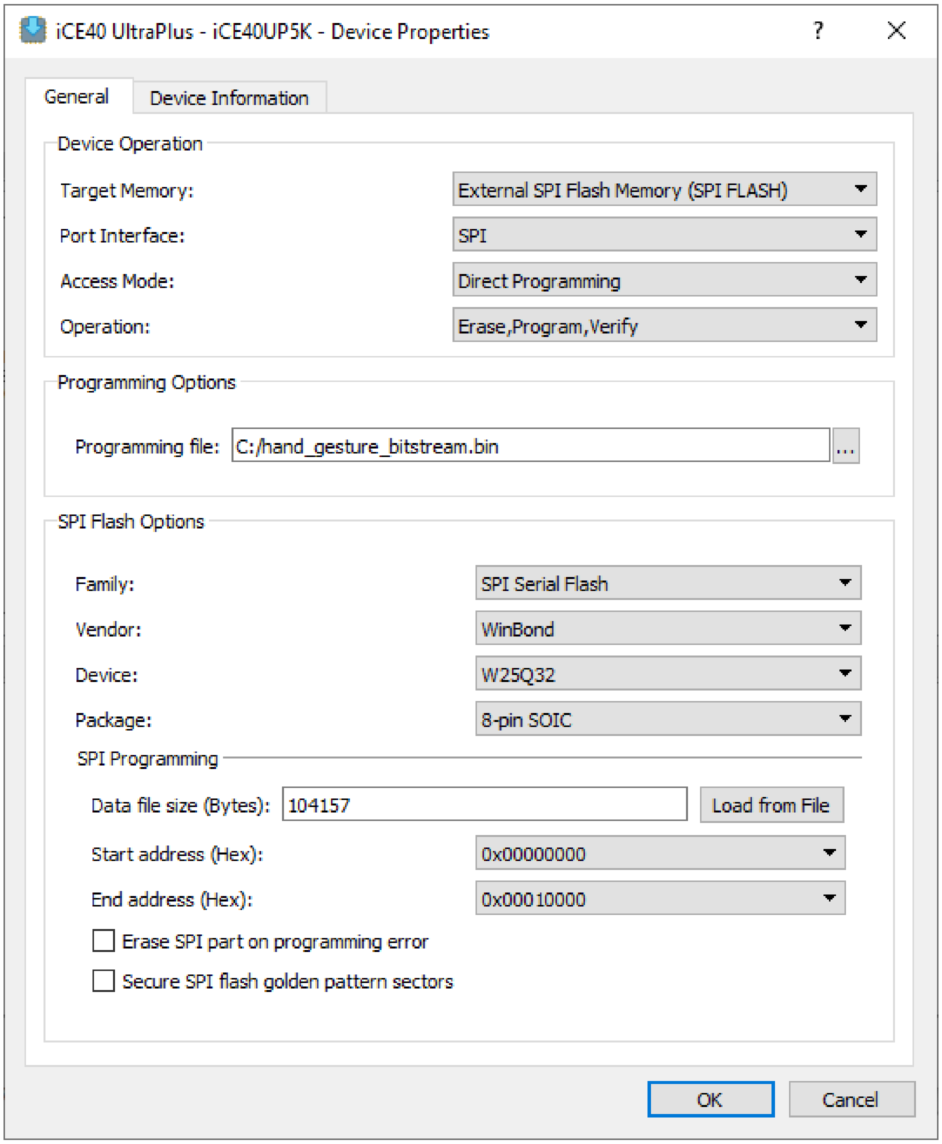
Figure 9.5. Diamond Programmer – Device Operation

6. Apply the settings below:
   a. Under Device Operation, select the options below:
      - **Target Memory – External SPI Flash Memory**
      - **Port Interface – SPI**
      - **Access Mode – Direct Programming**
      - **Operation – Erase, Program, Verify**

   b. Under SPI Flash Options, select the options below:
      - **Family – SPI Serial Flash**
      - **Vendor – Winbond**
      - **Device – W25Q32**
      - **Package – 8-pin SOIC**
7. To program the bitstream file, select the options as shown in Figure 9.6.
   a. Under **Programming Options**, select the *Hand Gesture RTL* bitstream file in Programming file.

   b. Click **Load from File** to update the **Data file size (Bytes)** value.

   c. Ensure that the following addresses are correct:
      - **Start Address (Hex) – 0x00000000**
      - **End Address (Hex) – 0x00010000**
8. Click **OK**.



iCE40 UltraPlus - iCE40UP5K - Device Properties

Device Operation

Target Memory: External SPI Flash Memory (SPI FLASH)

Port Interface: SPI

Access Mode: Direct Programming

Operation: Erase,Program,Verify

Programming Options

Programming file: C:/hand_gesture_bitstream.bin

SPI Flash Options

Family: SPI Serial Flash

Vendor: WinBond

Device: W25Q32

Package: 8-pin SOIC

SPI Programming

Data file size (Bytes): 104157

Start address (Hex): 0x00000000

End address (Hex): 0x00010000

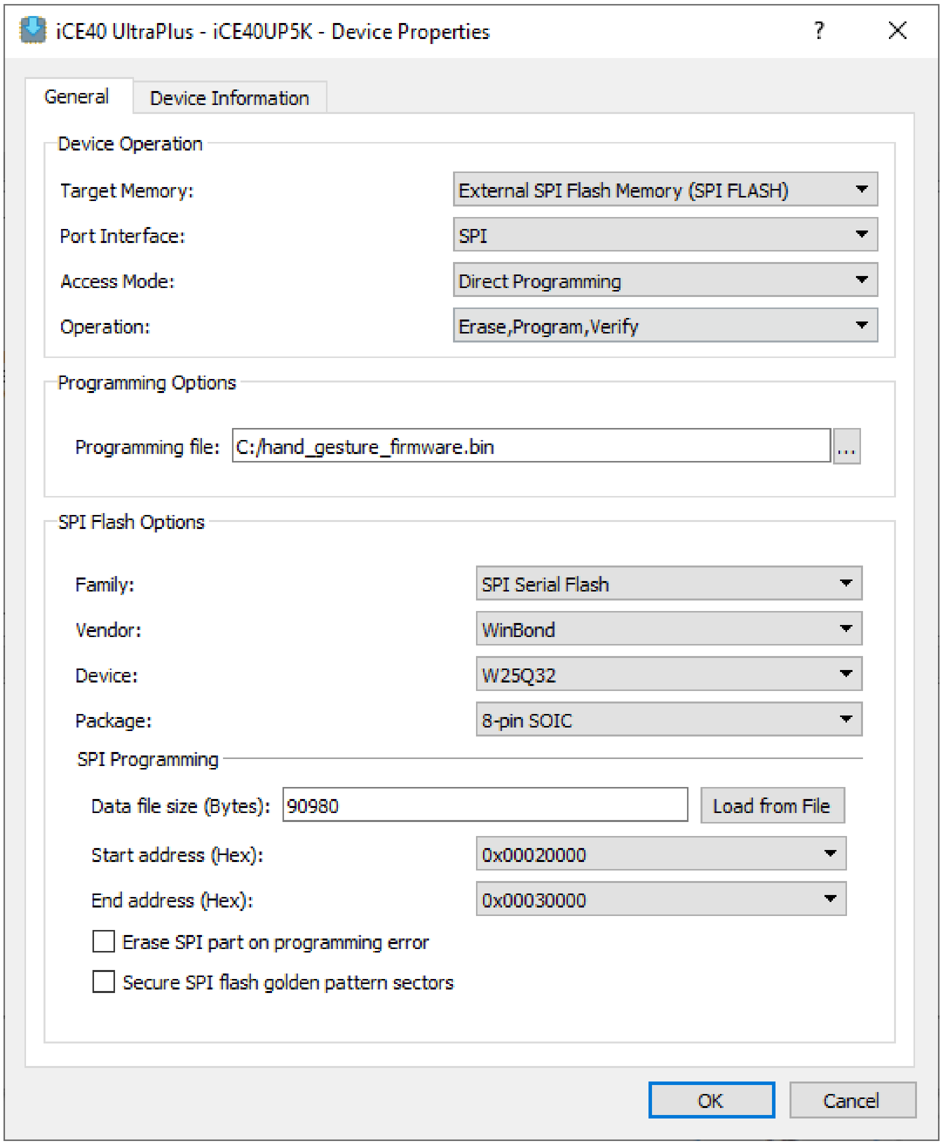
☐ Erase SPI part on programming error

☐ Secure SPI flash golden pattern sectors

Figure 9.6. Radiant Programmer – Bitstream Flashing Settings

9. Initially, the .xcf file only has one option to add bin file. You need to program two bin files in case of hand gesture demo, add one more device from in the toolbar. Set **Device Family** to **iCE40 UltraPlus** and Device to **iCE40UP5K**.

10. To program the firmware, select the options as shown in Figure 9.7.
    a. Under **Programming Options**, select the hand gesture firmware generated by the SensAI tool.
    b. Ensure that the following addresses are correct:
        - **Start Address (Hex) – 0x00020000**
        - **End Address (Hex) – 0x00030000**
11. Click **OK**.



Target Memory: External SPI Flash Memory (SPI FLASH)

Port Interface: SPI

Access Mode: Direct Programming

Operation: Erase,Program,Verify

Programming file: C:/hand_gesture_firmware.bin

Family: SPI Serial Flash

Vendor: WinBond

Device: W25Q32

Package: 8-pin SOIC

Data file size (Bytes): 90980

Start address (Hex): 0x00020000

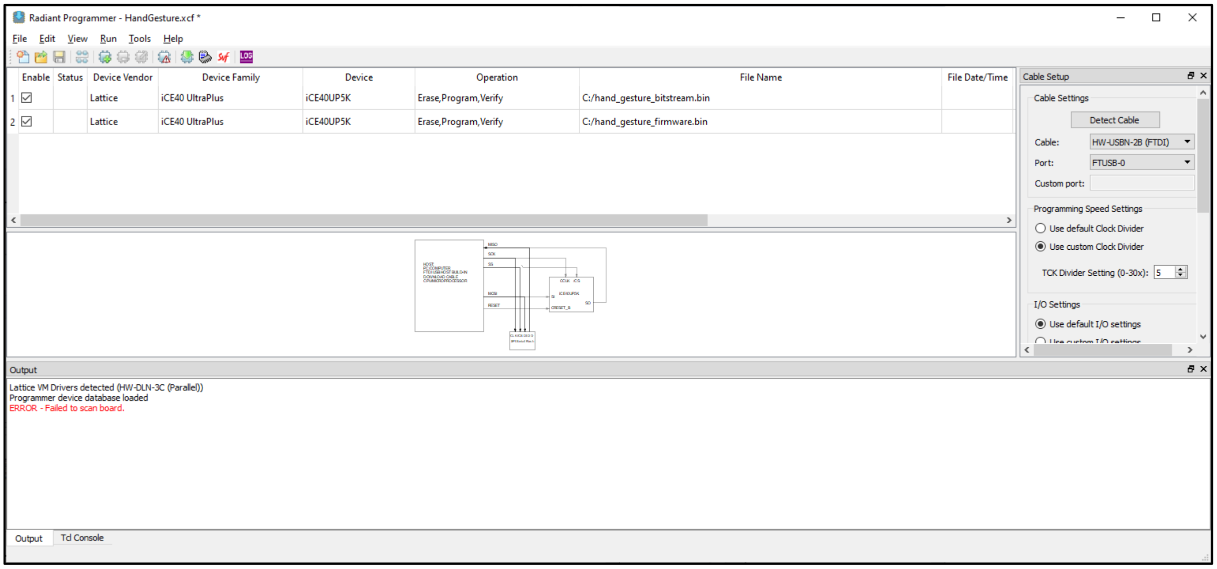
End address (Hex): 0x00030000

☐ Erase SPI part on programming error

☐ Secure SPI flash golden pattern sectors

Figure 9.7. Radiant Programmer – Firmware Bin File Flashing Setting

12. In the main interface, click **Program Device** to program the binary file.



Radiant Programmer - HandGesture.xcf *

File Edit View Run Tools Help

	Enable	Status	Device Vendor	Device Family	Device	Operation	File Name	File Date/Time
1	☑		Lattice	iCE40 UltraPlus	iCE40UP5K	Erase,Program,Verify	C:/hand_gesture_bitstream.bin	
2	☑		Lattice	iCE40 UltraPlus	iCE40UP5K	Erase,Program,Verify	C:/hand_gesture_firmware.bin	

Cable Setup

Cable Settings

Detect Cable

Cable: HW-USBN-2B (FTDI)

Port: FTUSB-0

Custom port:

Programming Speed Settings

○ Use default Clock Divider

● Use custom Clock Divider

TCK Divider Setting (0-30x): 5

I/O Settings

● Use default I/O settings

○ Use custom I/O settings

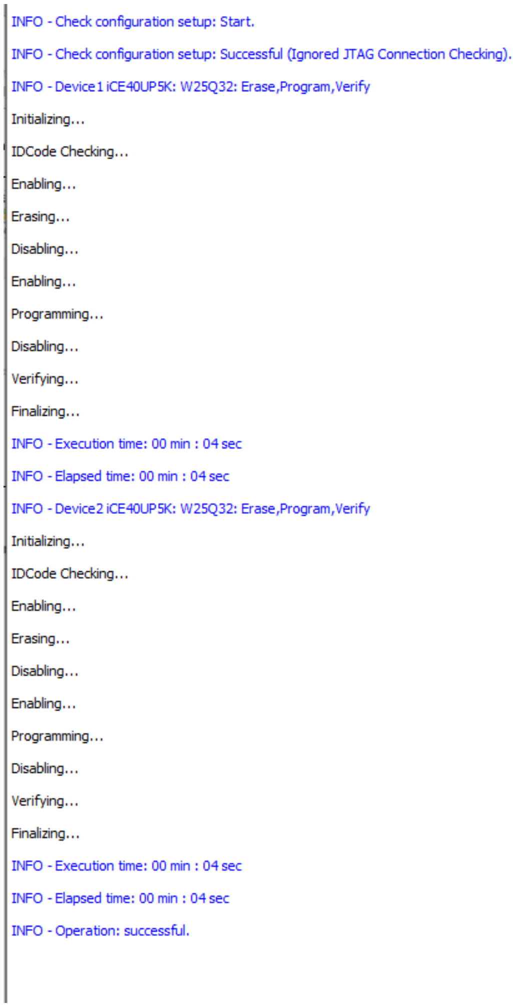
Output

```
Lattice VM Drivers detected (HW-DLN-3C (Parallel))
Programmer device database loaded
ERROR - Failed to scan board.
```

MISO
SCK
SS
MOSI
RESET
HOST PC/COMPUTER FTDI USB HOST BUILD-IN DOWNLOAD CABLE CPU/MICROPROCESSOR
CCLK
iCS
iCE40UP5K
SI
SO
CRESET_B
Output
Tcl Console

Figure 9.8. Radiant Programmer – Program Device

13. After successful programming, the **Output** console displays the result as shown in Figure 9.9.



```
INFO - Check configuration setup: Start.
INFO - Check configuration setup: Successful (Ignored JTAG Connection Checking).
INFO - Device1 iCE40UP5K: W25Q32: Erase,Program,Verify
Initializing...
IDCode Checking...
Enabling...
Erasing...
Disabling...
Enabling...
Programming...
Disabling...
Verifying...
Finalizing...
INFO - Execution time: 00 min : 04 sec
INFO - Elapsed time: 00 min : 04 sec
INFO - Device2 iCE40UP5K: W25Q32: Erase,Program,Verify
Initializing...
IDCode Checking...
Enabling...
Erasing...
Disabling...
Enabling...
Programming...
Disabling...
Verifying...
Finalizing...
INFO - Execution time: 00 min : 04 sec
INFO - Elapsed time: 00 min : 04 sec
INFO - Operation: successful.
```

Figure 9.9. Radiant Programmer – Output Console

10. Running the Demo

10.1. Running the Demo in LEDs

To run the demo and observe results on the board:

1. Power ON the Himax HM01B0 UPduino Shield board.
2. Show the gesture in front of the board, which results to the LEDs to turn on. Refer to Figure 10.1 for the LED information.

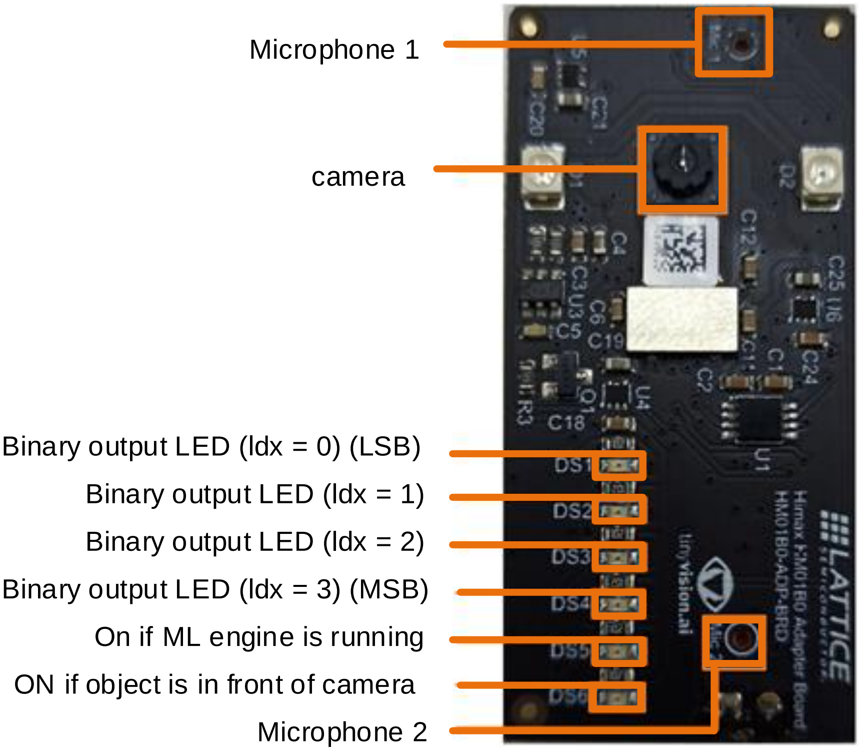
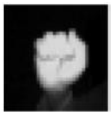
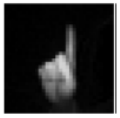
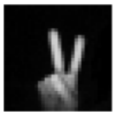
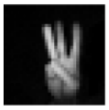
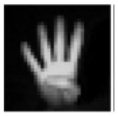
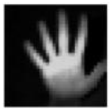
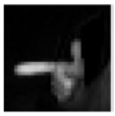
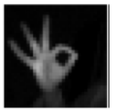
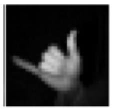
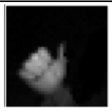

Microphone 1
camera
Binary output LED (Idx = 0) (LSB)
Binary output LED (Idx = 1)
Binary output LED (Idx = 2)
Binary output LED (Idx = 3) (MSB)
On if ML engine is running
ON if object is in front of camera
Microphone 2
DS1
DS2
DS3
DS4
DS5
DS6
Himax HM01B0 Adapter Board
HM01B0-ADP-BRD
LATTICE
tinyVision.ai

Figure 10.1. Camera and LED Location

- **DS1- DS4 –** Binary output LEDs, where DS1 is LSB (Least Significant Bit) and DS4 is MSB (Most Significant Bit).
- **DS5 –** If this LED is on, ML engine is running.
- **DS6 –** If this LED is on, an object is in front of camera.

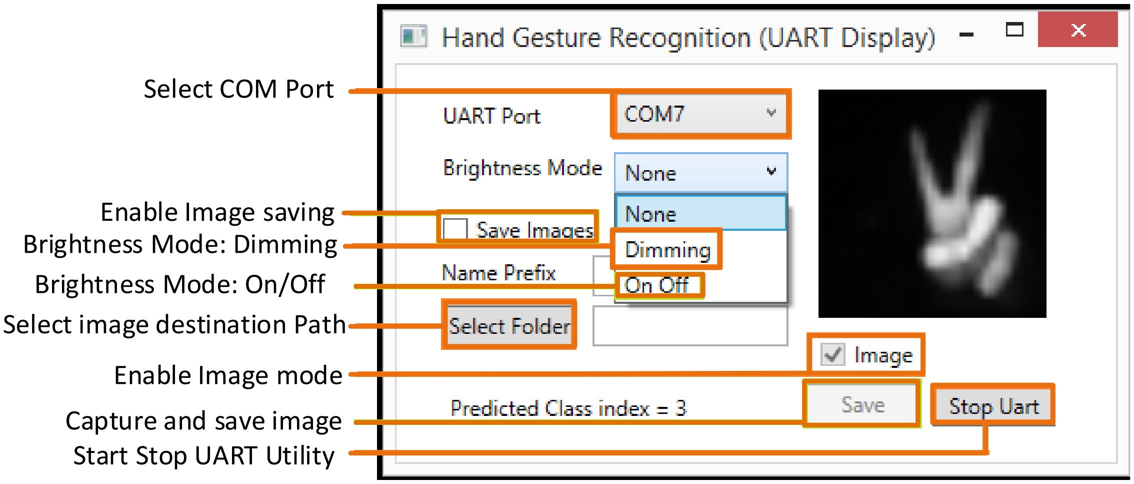
Table 10.1. Hand Gesture Recognition Classes

Gesture ID	Gesture Image	Detection LEDs state [DS4, DS3, DS2, DS1]
1		[OFF, OFF, OFF, ON]
2		[OFF, OFF, ON, OFF]
3		[OFF, OFF, ON, ON]
4		[OFF, ON, OFF, OFF]
5		[OFF, ON, OFF, ON]
6		[OFF, ON, ON, OFF]
7		[OFF, ON, ON, ON]
8		[ON, OFF, OFF, OFF]
9		[ON, OFF, OFF, ON]
10		[ON, OFF, ON, OFF]

10.2. Running the Demo in Windows UART Display Utility

To run the demo in Windows UART Display utility:

1. Start UART Windows utility.



Hand Gesture Recognition (UART Display)

UART Port: COM7

Brightness Mode: None

Predicted Class index = 3

Figure 10.2. UART Windows Utility

UART display utility can display camera output, predicted hand gesture class and change screen brightness based on brightness mode configured. This utility can also be used to capture camera output images as mentioned in the Creating the Dataset section.

2. Apply the settings below to run the demo:
   a. Select UART port.
   b. Enable **Image** to see the camera output image.
   c. Click on **Start UART**.
   d. Predicted class is displayed as shown in Figure 10.2**.**
3. Run the demo with different brightness modes:
   - There are two brightness modes: **Dimming** and **On/Off**.
   - In Dimming Mode, the system's brightness is adjusted by the detected class index.
   - For example, if number of classes are 10, the brightness level is 20 if class id detected is 2.
     Note: UART Widows utility has default number of classes as 10.
   - In On/Off mode, if detected class id is 1, then the display is turned off. For the rest of class indexes, display is on.

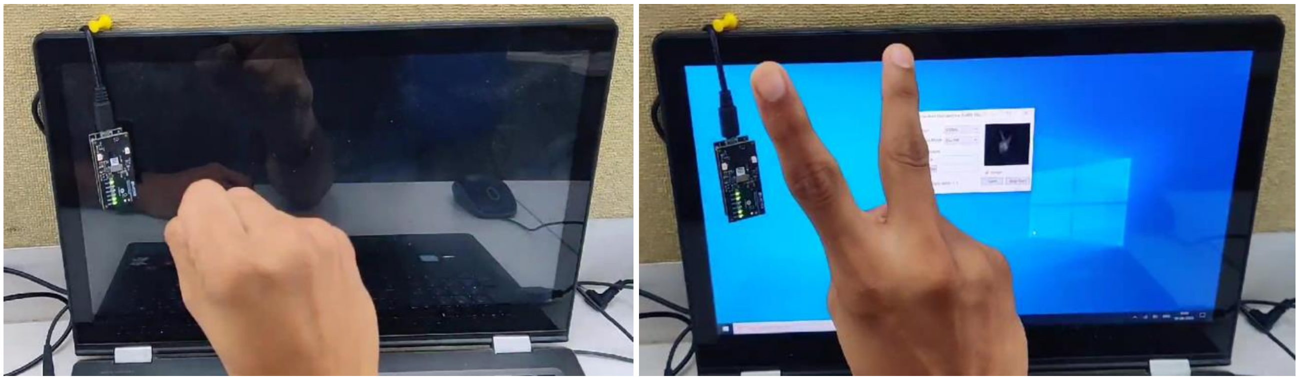


Figure 10.3. Brightness Mode On/Off

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Revision 1.0, December 2020

Section	Change Summary
All	Initial release.

LATTICE SEMICONDUCTOR®

www.latticesemi.com