Key Phrase Detection Using Compact CNN Accelerator IP

Reference Design

FPGA-RD-02066-1.0

October 2019

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
CKPT	Checkpoint
CNN	Convolutional Neural Network
EVDK	Embedded Vision Development Kit
FPGA	Field-Programmable Gate Array
LED	Light-emitting diode
MLE	Machine Learning Engine
NN	Neural Network
NNC	Neural Network Compiler
SD	Secure Digital
SDHC	Secure Digital High Capacity
SDXC	Secure Digital extended Capacity
SPI	Serial Peripheral Interface
VIP	Video Interface Platform
USB	Universal Serial Bus

1. Introduction

This document describes the key phrase detection design process using an iCE40 UltraPlus™ FPGA platform (HiMax HM01B0 UPduino Shield).

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 2.1 program
3. FPGA design
    - Creating the FPGA Bitstream file
4. FPGA Bitstream and Quantized Weights and Instructions
    - Flashing the binary and bitstream files to iCE40 UPduino hardware

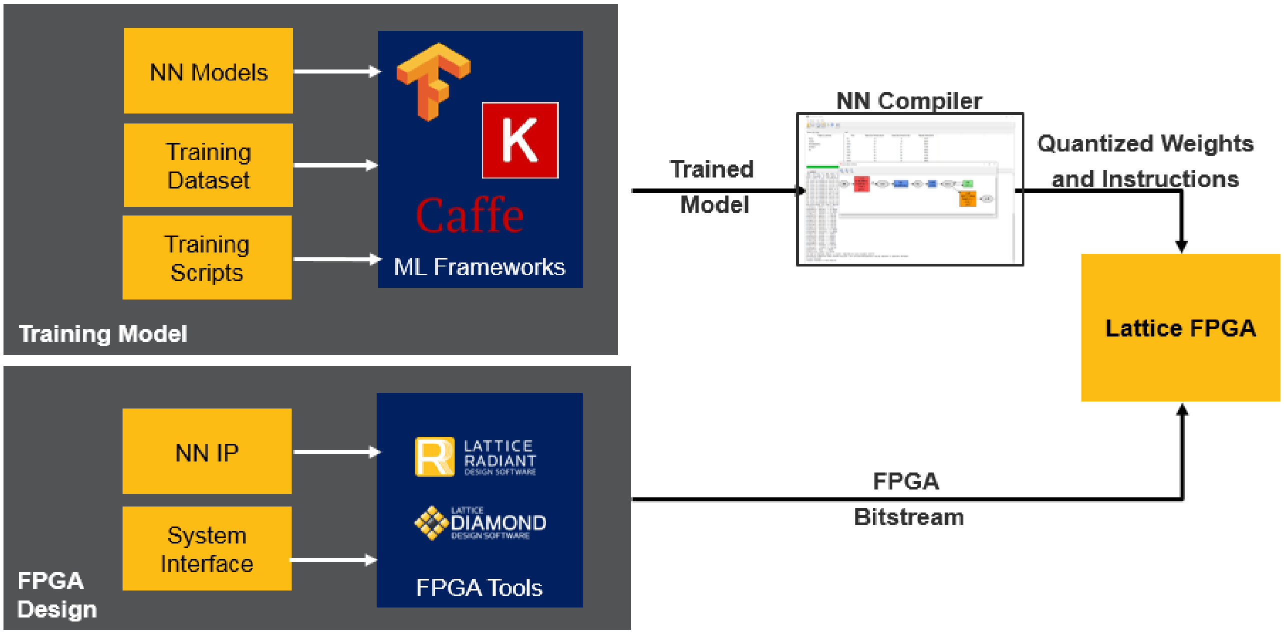


Figure 1.1. Lattice Machine Learning Design Flow

2. Setting Up the Basic Environment

2.1. Tools and Hardware Requirements

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

2.1.1. Lattice Tools

- Lattice Radiant Tool v1.1 – Refer to http://www.latticesemi.com/latticeradiant
- Lattice Radiant Programmer v1.1 – Refer to http://www.latticesemi.com/latticeradiant
- Lattice SensAI Compiler v2.1 – 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/en/Products/DevelopmenBoardsAndKits/HimaxHM01B0.

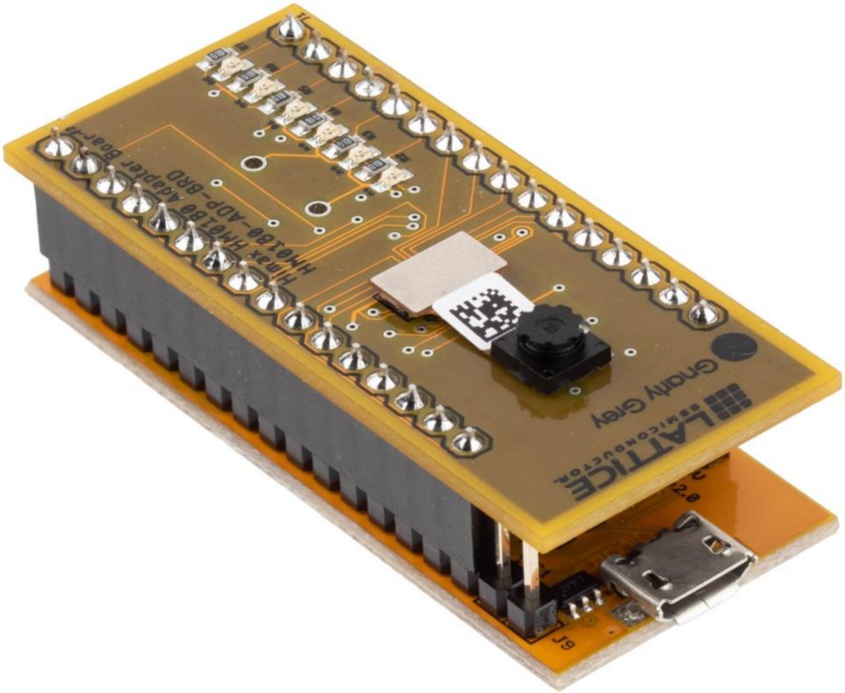


Figure 2.1. Lattice Himax HM01B0 Upduino Shield Board

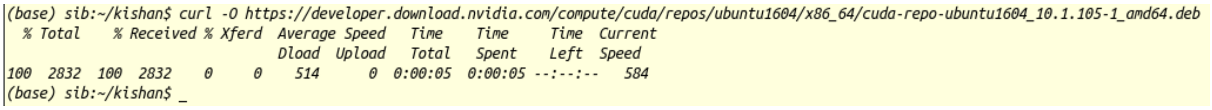
2.2. Setting Up the Linux Environment for Machine Training

2.2.1. Installing the CUDA Toolkit

To install the NVIDIA CUDA toolkit, run the commands below:

1. Download the NVIDIA CUDA toolkit.

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

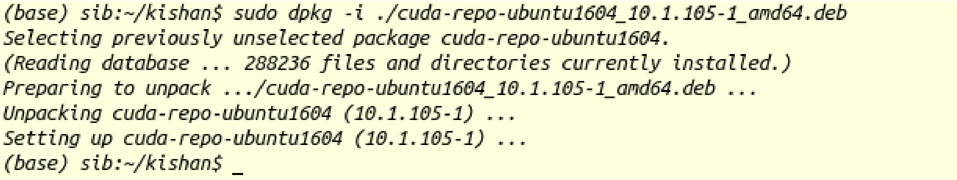


```
(base) sib:~/kishan$ curl -O https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2832  100  2832    0     0    514      0  0:00:05  0:00:05 --:--:--   584
(base) sib:~/kishan$ _
```

Figure 2.2. CUDA Repo Download

2. Install the deb package.

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

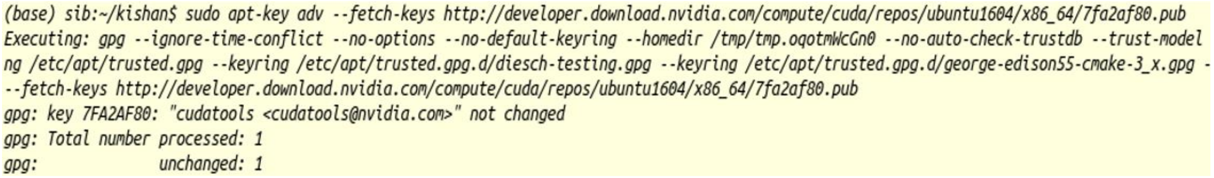


```
(base) sib:~/kishan$ sudo dpkg -i ./cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
Selecting previously unselected package cuda-repo-ubuntu1604.
(Reading database ... 288236 files and directories currently installed.)
Preparing to unpack .../cuda-repo-ubuntu1604_10.1.105-1_amd64.deb ...
Unpacking cuda-repo-ubuntu1604 (10.1.105-1) ...
Setting up cuda-repo-ubuntu1604 (10.1.105-1) ...
(base) sib:~/kishan$ _
```

Figure 2.3. CUDA Repo Installation

3. Proceed with the installation.

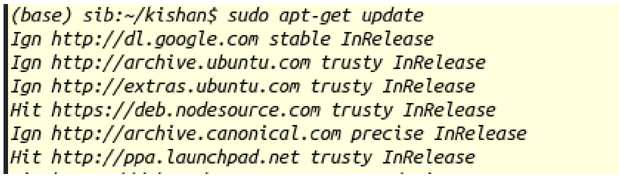
```
$ sudo apt-key adv --fetch-keys
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa
2af80.pub
```



```
(base) sib:~/kishan$ sudo apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --homedir /tmp/tmp.oqotmWcGn0 --no-auto-check-trustdb --trust-model
ng /etc/apt/trusted.gpg --keyring /etc/apt/trusted.gpg.d/diesch-testing.gpg --keyring /etc/apt/trusted.gpg.d/george-edison55-cmake-3_x.gpg -
--fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
gpg: key 7FA2AF80: "cudatools <cudatools@nvidia.com>" not changed
gpg: Total number processed: 1
gpg:               unchanged: 1
```

Figure 2.4. Fetch Keys

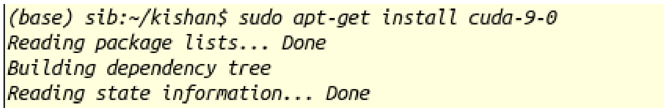
```
$ sudo apt-get update
```



```
(base) sib:~/kishan$ sudo apt-get update
Ign http://dl.google.com stable InRelease
Ign http://archive.ubuntu.com trusty InRelease
Ign http://extras.ubuntu.com trusty InRelease
Hit https://deb.nodesource.com trusty InRelease
Ign http://archive.canonical.com precise InRelease
Hit http://ppa.launchpad.net trusty InRelease
```

Figure 2.5. Updated Ubuntu Package Repositories

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



```
(base) sib:~/kishan$ sudo apt-get install cuda-9-0
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

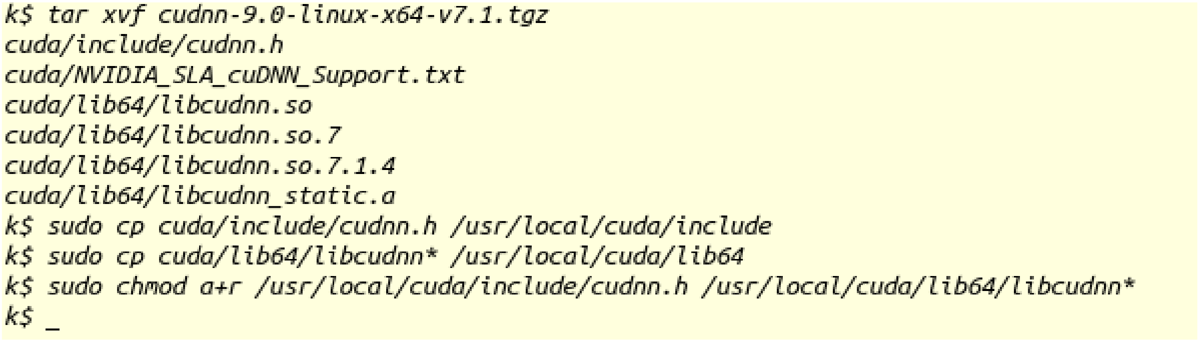
Figure 2.6. CUDA Installation Completed

2.2.2. cuDNN Installation

To install the cuDNN:

1. Create Nvidia developer account in https://developer.nvidia.com.
2. Download cuDNN library 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*
```



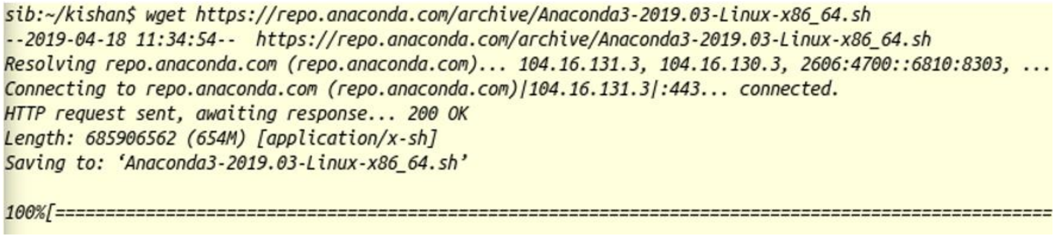
```
k$ tar xvf cudnn-9.0-linux-x64-v7.1.tgz
cuda/include/cudnn.h
cuda/NVIDIA_SLA_cuDNN_Support.txt
cuda/lib64/libcudnn.so
cuda/lib64/libcudnn.so.7
cuda/lib64/libcudnn.so.7.1.4
cuda/lib64/libcudnn_static.a
k$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
k$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
k$ sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
k$ _
```

Figure 2.7. cuDNN Library Installation

2.2.3. Installing the Anaconda and Python3

To install the Anaconda and Python 3:

1. Go to https://www.anaconda.com/distribution/#download.
2. Download the Python 3 version of Anaconda for Linux.



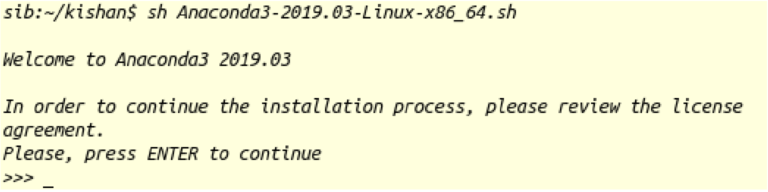
```
sib:~/kishan$ wget https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86_64.sh
--2019-04-18 11:34:54--  https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.131.3, 104.16.130.3, 2606:4700::6810:8303, ...
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.131.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 685906562 (654M) [application/x-sh]
Saving to: 'Anaconda3-2019.03-Linux-x86_64.sh'

100%[==========================================================================================
```

Figure 2.8. Anaconda Package Download

3. Run the command below to install the Anaconda environment.

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

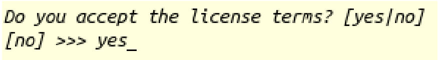
Note: Anaconda3-<version>-Linux-x86_64.sh version may vary based on the release.



```
sib:~/kishan$ sh Anaconda3-2019.03-Linux-x86_64.sh

Welcome to Anaconda3 2019.03

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

Figure 2.9. Anaconda Installation

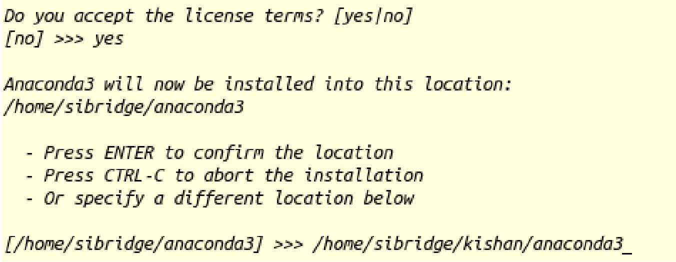
4. Accept the license.



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

Figure 2.10. License Terms Prompt

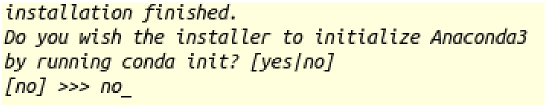
5. Confirm the installation path. Follow the instructions onscreen to change the default path.



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

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

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

[/home/sibridge/anaconda3] >>> /home/sibridge/kishan/anaconda3_
```

Figure 2.11. Installation Path Confirmation

6. After installation, enter **No** as shown in Figure 2.12.



```
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>> no_
```

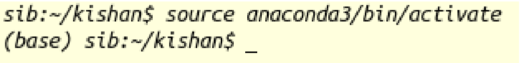
Figure 2.12. Launch/Initialize Anaconda Environment Installation Complete

2.2.4. Installing TensorFlow v1.12

To install the TensorFlow v1.12:

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

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

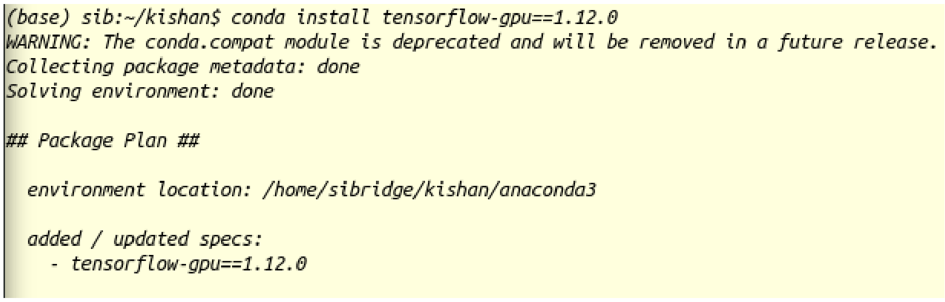


```
sib:~/kishan$ source anaconda3/bin/activate
(base) sib:~/kishan$ _
```

Figure 2.13. Anaconda Environment Activation

2. Install the TensorFlow by running the command below:

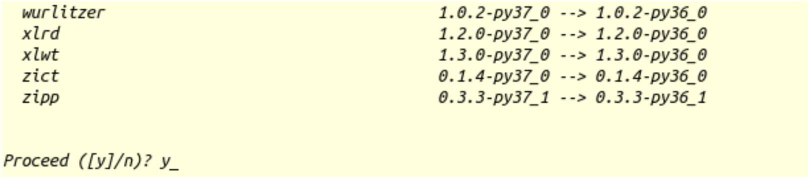
```
$ conda install tensorflow-gpu==1.12.0
```



```
(base) sib:~/kishan$ conda install tensorflow-gpu==1.12.0
WARNING: The conda.compat module is deprecated and will be removed in a future release.
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - tensorflow-gpu==1.12.0
```

Figure 2.14. TensorFlow Installation

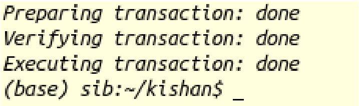
3. After installation, enter **Y** as shown in Figure 2.15.



```
  wurlitzer                                  1.0.2-py37_0 --> 1.0.2-py36_0
  xlrd                                       1.2.0-py37_0 --> 1.2.0-py36_0
  xlwt                                       1.3.0-py37_0 --> 1.3.0-py36_0
  zict                                       0.1.4-py37_0 --> 0.1.4-py36_0
  zipp                                       0.3.3-py37_1 --> 0.3.3-py36_1


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

Figure 2.15. TensorFlow Installation Confirmation

Figure 2.16 shows TensorFlow installation is completed.



```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(base) sib:~/kishan$ _
```

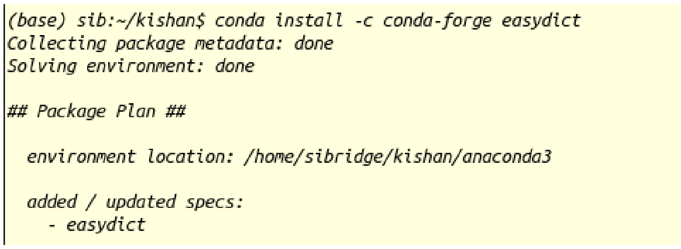
Figure 2.16. TensorFlow Installation Complete

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
```

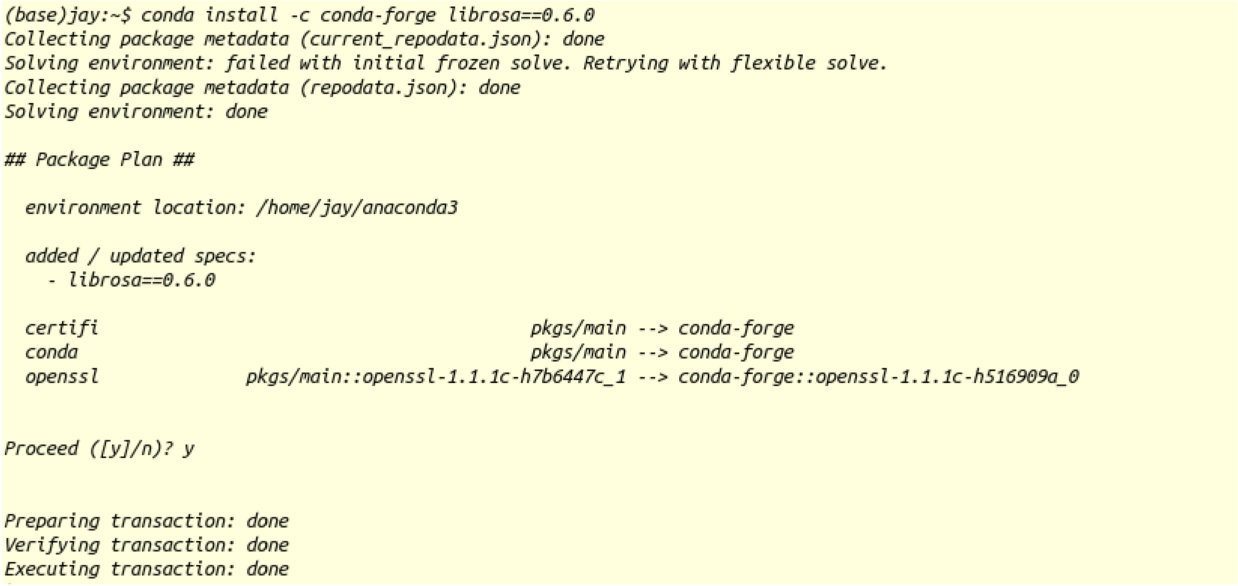


```
(base) sib:~/kishan$ conda install -c conda-forge easydict
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - easydict
```

Figure 2.17. Easydict Installation

2. Install Easydict by running the command below:

```
$ conda install –c conda-forge librosa==0.6.0
```



```
(base)jay:~$ conda install -c conda-forge librosa==0.6.0
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/jay/anaconda3

  added / updated specs:
    - librosa==0.6.0

  certifi                                         pkgs/main --> conda-forge
  conda                                           pkgs/main --> conda-forge
  openssl              pkgs/main::openssl-1.1.1c-h7b6447c_1 --> conda-forge::openssl-1.1.1c-h516909a_0


Proceed ([y]/n)? y


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

Figure 2.18. Anaconda Env – librosa Installation

Figure 2.18 shows the successful installation.

3. Preparing the Dataset

This chapter describes how to create a dataset using examples from Google Speech Command.

3.1. Selecting the Dataset

Below are two options of dataset for Key Phrase Detection models which can be used for training:

- Google Speech Command Dataset v0.01
- Google Speech Command Dataset v0.02

3.2. Downloading the Dataset

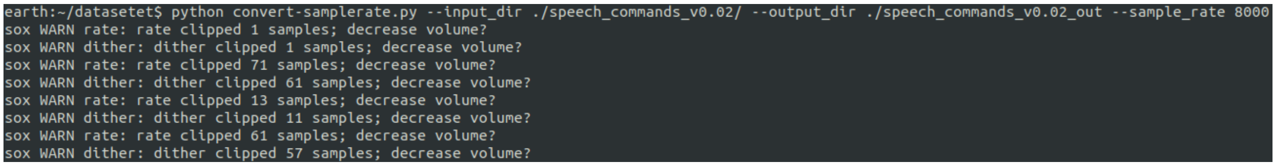
To download the dataset:

1. Go to the links below to download the Google Speech Command Dataset:
   - Google speech command dataset v0.01:
     https://storage.cloud.google.com/download.tensorflow.org/data/speech_commands_v0.01.tar.gz
   - Google speech command dataset v0.02:
     https://storage.cloud.google.com/download.tensorflow.org/data/speech_commands_v0.02.tar.gz
2. Convert the sample rate of 16 KHz .wav files to 8 KHz by running the commands below:
   a. Install the necessary dependency:

```
$ sudo apt-get install sox
```

   b. Convert the dataset:

```
$python convert-samplerate.py
```



```
earth:~/datasetet$ python convert-samplerate.py --input_dir ./speech_commands_v0.02/ --output_dir ./speech_commands_v0.02_out --sample_rate 8000
sox WARN rate: rate clipped 1 samples; decrease volume?
sox WARN dither: dither clipped 1 samples; decrease volume?
sox WARN rate: rate clipped 71 samples; decrease volume?
sox WARN dither: dither clipped 61 samples; decrease volume?
sox WARN rate: rate clipped 13 samples; decrease volume?
sox WARN dither: dither clipped 11 samples; decrease volume?
sox WARN rate: rate clipped 61 samples; decrease volume?
sox WARN dither: dither clipped 57 samples; decrease volume?
```

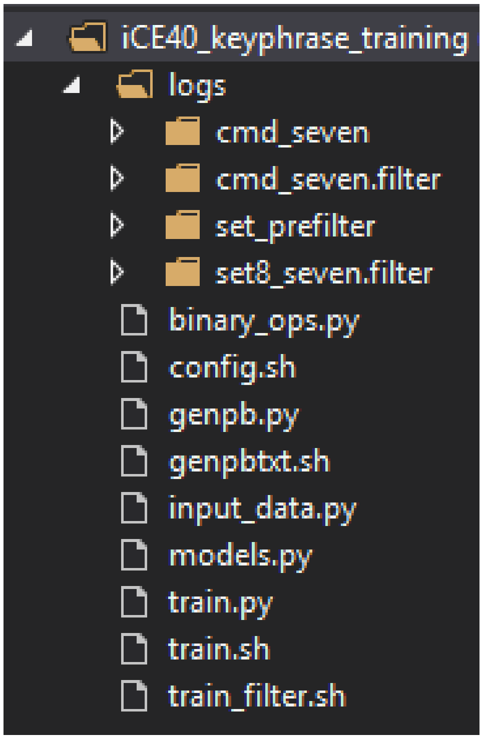
Figure 3.1. Convert Sample Rate Script Execution

3.3. Data Augmentation

Dataset Augmentation is done by adding background noise in training-code itself. Background noises are used from the dataset's *_background_noise_* directory. Refer to README.md in the background noise directory for more information on how to add noise files.

4. Training the Machine

4.1. Training Code Structure



```
◢ iCE40_keyphrase_training
    ◢ logs
        ▷ cmd_seven
        ▷ cmd_seven.filter
        ▷ set_prefilter
        ▷ set8_seven.filter
      binary_ops.py
      config.sh
      genpb.py
      genpbtxt.sh
      input_data.py
      models.py
      train.py
      train.sh
      train_filter.sh
```

Figure 4.1. Training Code Directory Structure

4.2. Neural Network Architecture

4.2.1. Neural Network Architecture

This section provides information on the Convolution Neural Network Configuration of the Key Phrase Detection design.

Table 4.1. Key Phrase Detection Training Network Topology

Audio Input (8320)		
Freqconv	Conv1d-(64, 64)	Conv1d basically performs convolution on 1d data and gives an output of 2D data. In this case,. the output is 64 x 64. Conv3 - # where: • Conv3 – 3 x 3 Convolution filter Kernel size • # - The number of filters For example, Conv3 - 8 = 8 3 x 3 convolution filter Batchnorm : Batch Normalization FC - # where: • FC – Fully connected layer • # - The number of outputs
	Relu	
Reshape	Reshape- (64, 1, 64)	
Transpose	Transpose-(64, 64, 1)	
Fire 1	Conv3 – 16	
	Batchnorm	
	Relu	
	Maxpool	
Fire 2	Conv3 – 16	
	Batchnorm	
	Relu	
Fire 3	Conv3 – 20	
	Batchnorm	
	Relu	
	Maxpool	
Fire 4	Conv3 – 20	
	Batchnorm	
	Relu	
Fire 5	Conv3 – 20	
	Batchnorm	
	Relu	
	Maxpool	
Fire6	Conv3 – 24	
	Batchnorm	
	Relu	
	Maxpool	
Dropout	Dropout - 0.50	
fc4	FC – (2 + Num-keywords)	

- In Table 4.1, Layer contains Convolution (conv), batch normalization (bn), Relu and dropout layers.
- Output of layer fc4 is number of classes in dataset along with silence and unknown considered as 2 keywords, so total number of outputs of fc4 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 input layer/image and generates activation map (I.e. feature map). This filter is an array of numbers (the numbers are called weights or parameters). Each of these filters can be thought of as feature identifiers, like straight edges, simple colors, and curves & 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 (for example, curve) it is looking for is in the input volume.

- Relu (Activation layer)
  After each conv layer, it is convention to apply a nonlinear layer (or activation layer) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations).In the past, nonlinear functions like 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 to the 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 just 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, programmers 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 2x2) and a stride of the same length. It then applies it 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 we know 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. Second is that it controls over fitting. This term refers to 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.

- Batchnorm Layer
  Batch normalization layer reduces the internal covariance shift. In order to train a neural network, we do some preprocessing to the input data. For example, we could normalize all data so that it resembles a normal distribution (that means, zero mean and a unitary variance). Reason being preventing the early saturation of non-linear activation functions like the sigmoid function, assuring that all input data is in the same range of values, etc.

  But the problem 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 themselves to a new distribution in every training step. This problem 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 below process during training time:

  - Calculate the mean and variance of the layers input.
    - Normalize the layer inputs using the previously calculated batch statistics.
    - Scales and shifts in order 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.

- Drop-out 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 do 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 isn't 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 or ReLU or pool 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.

4.2.2. Key Phrase Detection Network Output

Key phrase detection network gives N + 2 values from last output node, where N is number of keywords trained. Two additional values represent *Silence* and *Unknown* as 2 keywords.

4.2.3. Training Code Overview

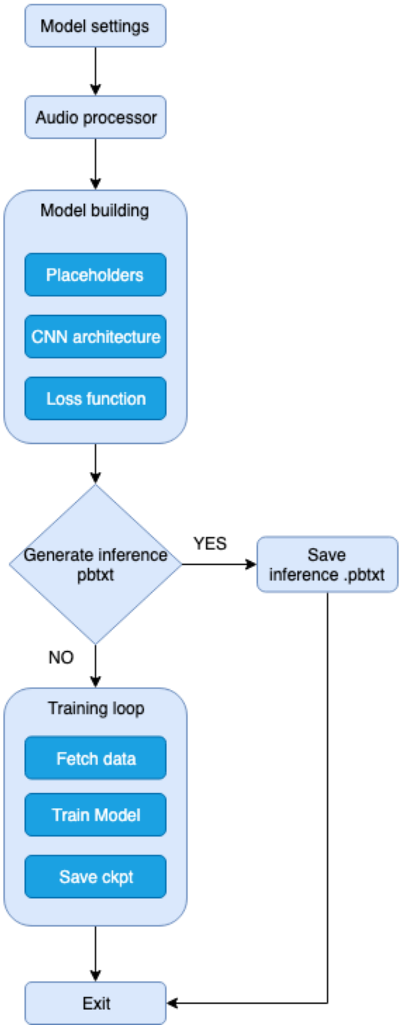
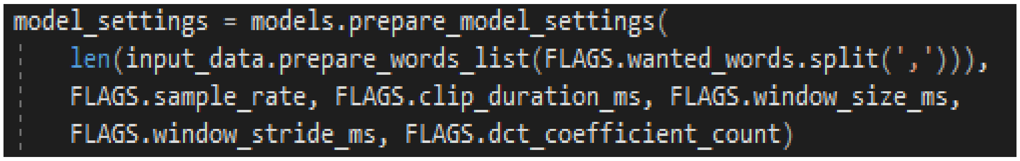


Figure 4.2. Training Code Flow Diagram

4.2.3.1. Model Settings

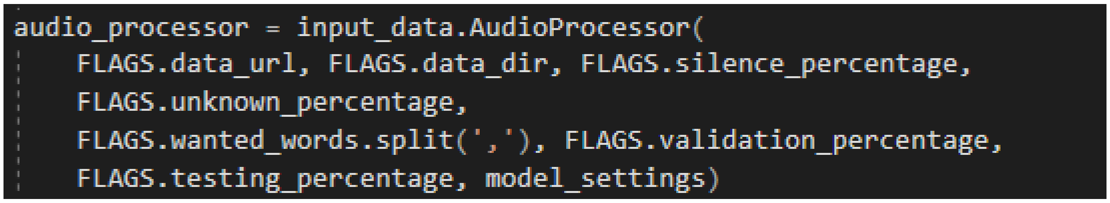


```
model_settings = models.prepare_model_settings(
    len(input_data.prepare_words_list(FLAGS.wanted_words.split(','))),
    FLAGS.sample_rate, FLAGS.clip_duration_ms, FLAGS.window_size_ms,
    FLAGS.window_stride_ms, FLAGS.dct_coefficient_count)
```

Figure 4.3. Code Snippet – Model Settings

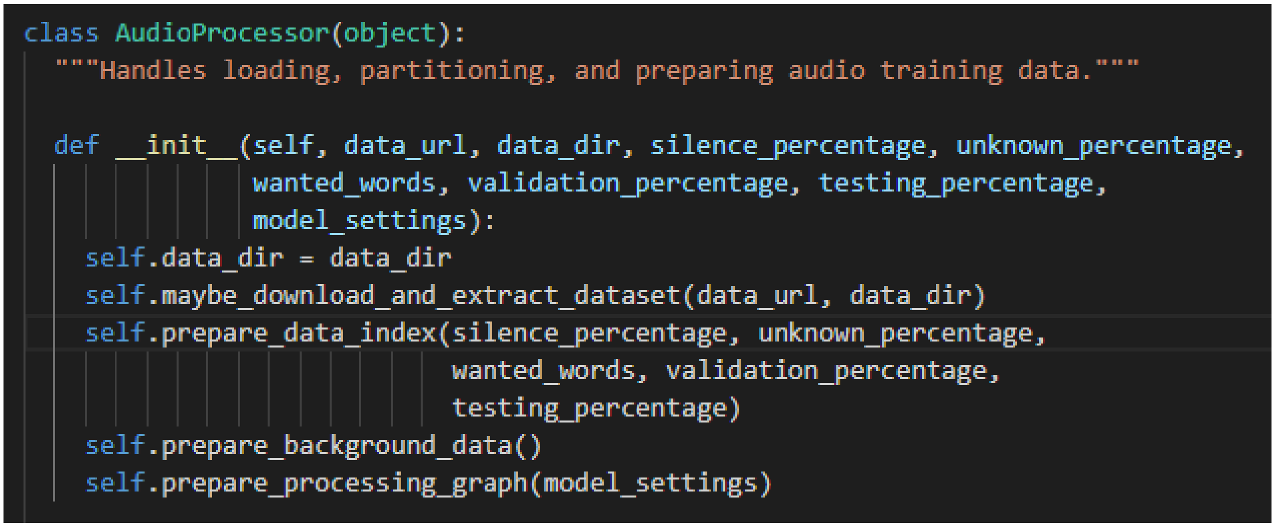
- *prepare_model_settings* function calculates and returns the dictionary containing all common settings for training from given parameters like *label_count* (# of keywords), *sample_rate*, *clip_duration_ms*, *window_size_ms*, *window_stride_ms* and *dct_coefficient_count*. More details about parameters are as follows:
  - label_count – Number of classes.
  - sample_rate – Number of audio samples per second.
  - clip_duration_ms – Length of each audio clip to be analyzed.
  - window_size_ms – Duration of frequency analysis window.
  - window_stride_ms – Length of move in time between frequency windows.
  - dct_coefficient_count – Number of frequency bins to use for analysis.

4.2.3.2. Audio Processor



```
audio_processor = input_data.AudioProcessor(
    FLAGS.data_url, FLAGS.data_dir, FLAGS.silence_percentage,
    FLAGS.unknown_percentage,
    FLAGS.wanted_words.split(','), FLAGS.validation_percentage,
    FLAGS.testing_percentage, model_settings)
```

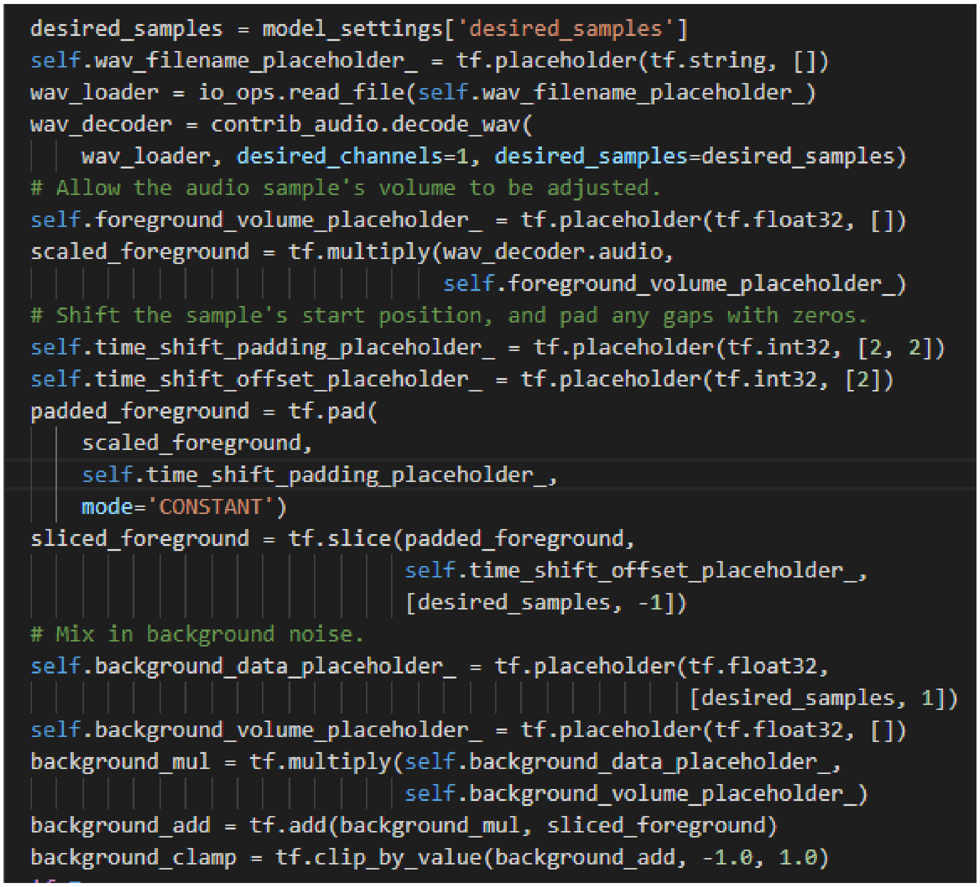
Figure 4.4. Code Snippet – Audio Processor



```
class AudioProcessor(object):
  """Handles loading, partitioning, and preparing audio training data."""

  def __init__(self, data_url, data_dir, silence_percentage, unknown_percentage,
               wanted_words, validation_percentage, testing_percentage,
               model_settings):
    self.data_dir = data_dir
    self.maybe_download_and_extract_dataset(data_url, data_dir)
    self.prepare_data_index(silence_percentage, unknown_percentage,
                            wanted_words, validation_percentage,
                            testing_percentage)
    self.prepare_background_data()
    self.prepare_processing_graph(model_settings)
```

Figure 4.5. Audio Processor Class Object

AudioProcessor function handles the loading, partitioning, and preparing of audio training data. It also downloads the dataset from given *data_url* in argument if dataset is not present.

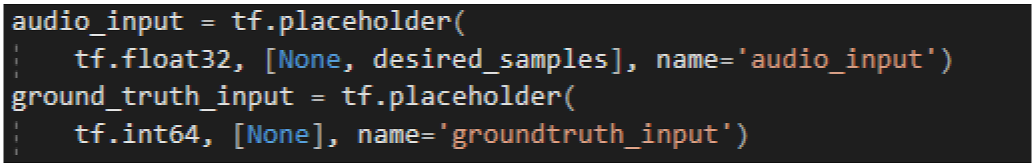


```
desired_samples = model_settings['desired_samples']
self.wav_filename_placeholder_ = tf.placeholder(tf.string, [])
wav_loader = io_ops.read_file(self.wav_filename_placeholder_)
wav_decoder = contrib_audio.decode_wav(
    wav_loader, desired_channels=1, desired_samples=desired_samples)
# Allow the audio sample's volume to be adjusted.
self.foreground_volume_placeholder_ = tf.placeholder(tf.float32, [])
scaled_foreground = tf.multiply(wav_decoder.audio,
                                self.foreground_volume_placeholder_)
# Shift the sample's start position, and pad any gaps with zeros.
self.time_shift_padding_placeholder_ = tf.placeholder(tf.int32, [2, 2])
self.time_shift_offset_placeholder_ = tf.placeholder(tf.int32, [2])
padded_foreground = tf.pad(
    scaled_foreground,
    self.time_shift_padding_placeholder_,
    mode='CONSTANT')
sliced_foreground = tf.slice(padded_foreground,
                             self.time_shift_offset_placeholder_,
                             [desired_samples, -1])
# Mix in background noise.
self.background_data_placeholder_ = tf.placeholder(tf.float32,
                                                   [desired_samples, 1])
self.background_volume_placeholder_ = tf.placeholder(tf.float32, [])
background_mul = tf.multiply(self.background_data_placeholder_,
                             self.background_volume_placeholder_)
background_add = tf.add(background_mul, sliced_foreground)
background_clamp = tf.clip_by_value(background_add, -1.0, 1.0)
```

Figure 4.6. Code Snippet – Decoding the Audio, Scaling and Adding of Noise

4.2.3.3. Model Building

Placeholders

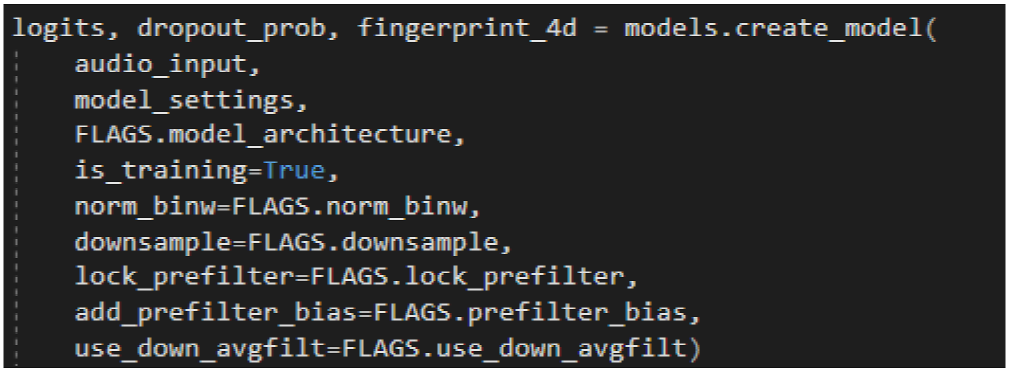


```
audio_input = tf.placeholder(
    tf.float32, [None, desired_samples], name='audio_input')
ground_truth_input = tf.placeholder(
    tf.int64, [None], name='groundtruth_input')
```

Figure 4.7. Code Snippet – Placeholders

- *audio_input* is placeholder to feed preprocessed audio input to network.
- *ground_truth_input* is placeholder to feed targeted labels of given batch. It is later used for evaluating training and optimization purpose.

CNN Architecture Generation

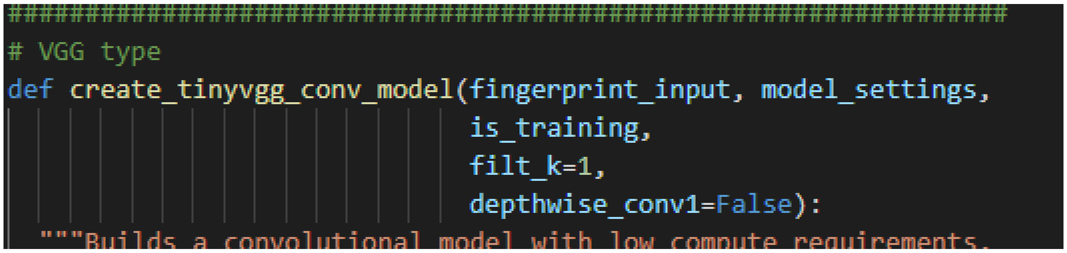


```
logits, dropout_prob, fingerprint_4d = models.create_model(
    audio_input,
    model_settings,
    FLAGS.model_architecture,
    is_training=True,
    norm_binw=FLAGS.norm_binw,
    downsample=FLAGS.downsample,
    lock_prefilter=FLAGS.lock_prefilter,
    add_prefilter_bias=FLAGS.prefilter_bias,
    use_down_avgfilt=FLAGS.use_down_avgfilt)
```

Figure 4.8. Code Snippet – Create Model

create_model creates training graph or training model using given configuration. More details about flags are as follows:

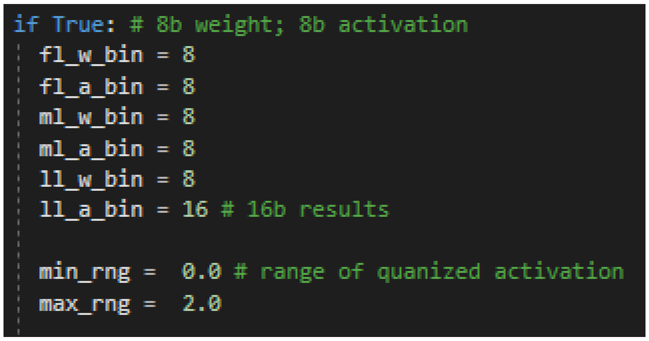
- audio_input – TensorFlow node that gives audio feature map as output.
- model_settings – Dictionary of information about the model.
- model_architecture – The default architecture is *tinyvgg_conv* defined in *config.sh*.
- is_training – Whether the model is going to be used for training.



```
#################################################################
# VGG type
def create_tinyvgg_conv_model(fingerprint_input, model_settings,
                              is_training,
                              filt_k=1,
                              depthwise_conv1=False):
  """Builds a convolutional model with low compute requirements.
```

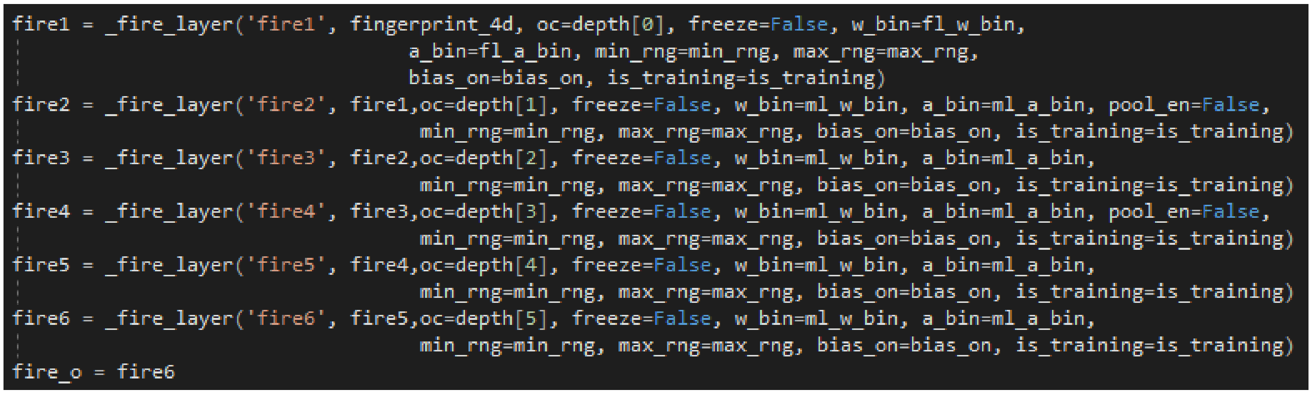
Figure 4.9. Code Snippet – Create_tinyvgg_conv_model

Quantization and Network Model Configuration



```
if True: # 8b weight; 8b activation
  fl_w_bin = 8
  fl_a_bin = 8
  ml_w_bin = 8
  ml_a_bin = 8
  ll_w_bin = 8
  ll_a_bin = 16 # 16b results

  min_rng =  0.0 # range of quanized activation
  max_rng =  2.0
```

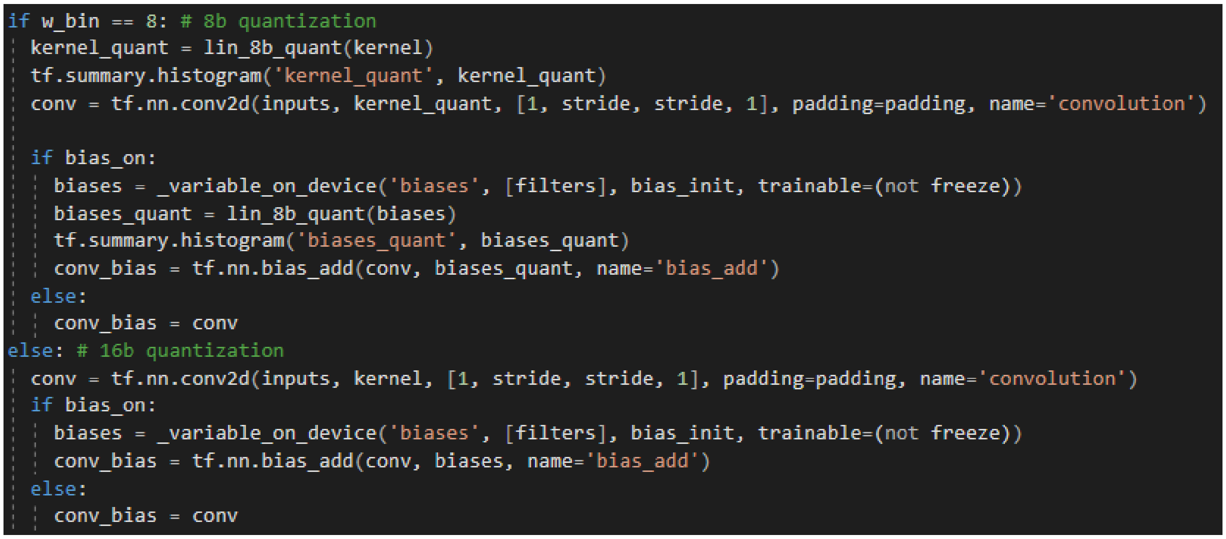
Figure 4.5. Code Snippet – Quantization Parameters Setting



```
fire1 = _fire_layer('fire1', fingerprint_4d, oc=depth[0], freeze=False, w_bin=fl_w_bin,
                                  a_bin=fl_a_bin, min_rng=min_rng, max_rng=max_rng,
                                  bias_on=bias_on, is_training=is_training)
fire2 = _fire_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, is_training=is_training)
fire3 = _fire_layer('fire3', fire2,oc=depth[2], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
                                   min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, is_training=is_training)
fire4 = _fire_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, is_training=is_training)
fire5 = _fire_layer('fire5', fire4,oc=depth[4], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
                                   min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, is_training=is_training)
fire6 = _fire_layer('fire6', fire5,oc=depth[5], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
                                   min_rng=min_rng, max_rng=max_rng, bias_on=bias_on, is_training=is_training)
fire_o = fire6
```

Figure 4.6. Code Snippet – Forward Graph Fire Layers

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 is to be done or not.



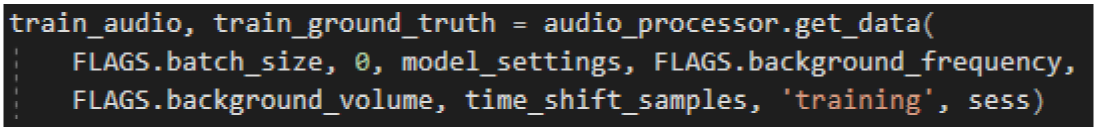
```
if w_bin == 8: # 8b quantization
    kernel_quant = lin_8b_quant(kernel)
    tf.summary.histogram('kernel_quant', kernel_quant)
    conv = tf.nn.conv2d(inputs, kernel_quant, [1, stride, stride, 1], padding=padding, name='convolution')

    if bias_on:
        biases = _variable_on_device('biases', [filters], bias_init, trainable=(not freeze))
        biases_quant = 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 = _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 4.7. Code Snippet – Convolution Quantization

4.2.3.4. Training Loop

Training Loop is divided in three segments:

- Fetching Data

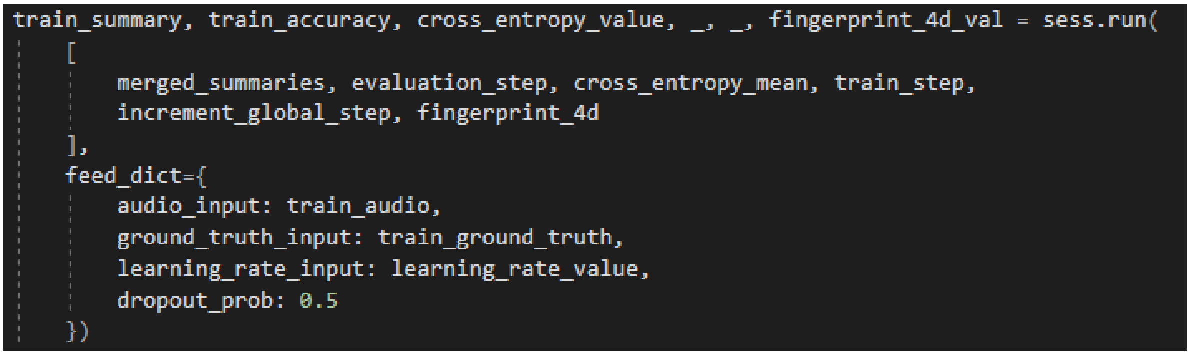


```
train_audio, train_ground_truth = audio_processor.get_data(
    FLAGS.batch_size, 0, model_settings, FLAGS.background_frequency,
    FLAGS.background_volume, time_shift_samples, 'training', sess)
```

Figure 4.8. Code Snippet – Training Loop

Using *audio_processor* object, the code shown in Figure 4.8 fetches the next batch for training.

- Training Model

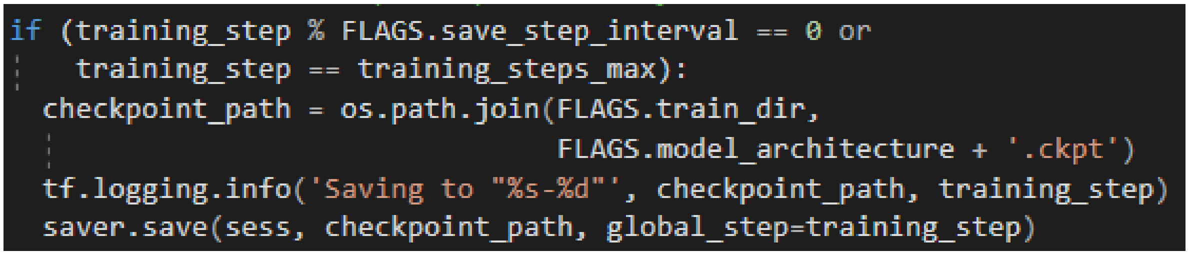


```
train_summary, train_accuracy, cross_entropy_value, _, _, fingerprint_4d_val = sess.run(
    [
        merged_summaries, evaluation_step, cross_entropy_mean, train_step,
        increment_global_step, fingerprint_4d
    ],
    feed_dict={
        audio_input: train_audio,
        ground_truth_input: train_ground_truth,
        learning_rate_input: learning_rate_value,
        dropout_prob: 0.5
    })
```

Figure 4.9. Code Snippet – Train Model

Using *session.run* code feeds next batch to network placeholders and evaluates desired operations mentioned in argument.

- Saving Checkpoints



```
if (training_step % FLAGS.save_step_interval == 0 or
    training_step == training_steps_max):
  checkpoint_path = os.path.join(FLAGS.train_dir,
                                 FLAGS.model_architecture + '.ckpt')
  tf.logging.info('Saving to "%s-%d"', checkpoint_path, training_step)
  saver.save(sess, checkpoint_path, global_step=training_step)
```

Figure 4.10. Code Snippet – Save Checkpoints

Trained Checkpoints are periodically saved based on flag specified *save_step_interval* at *checkpoint_path*.

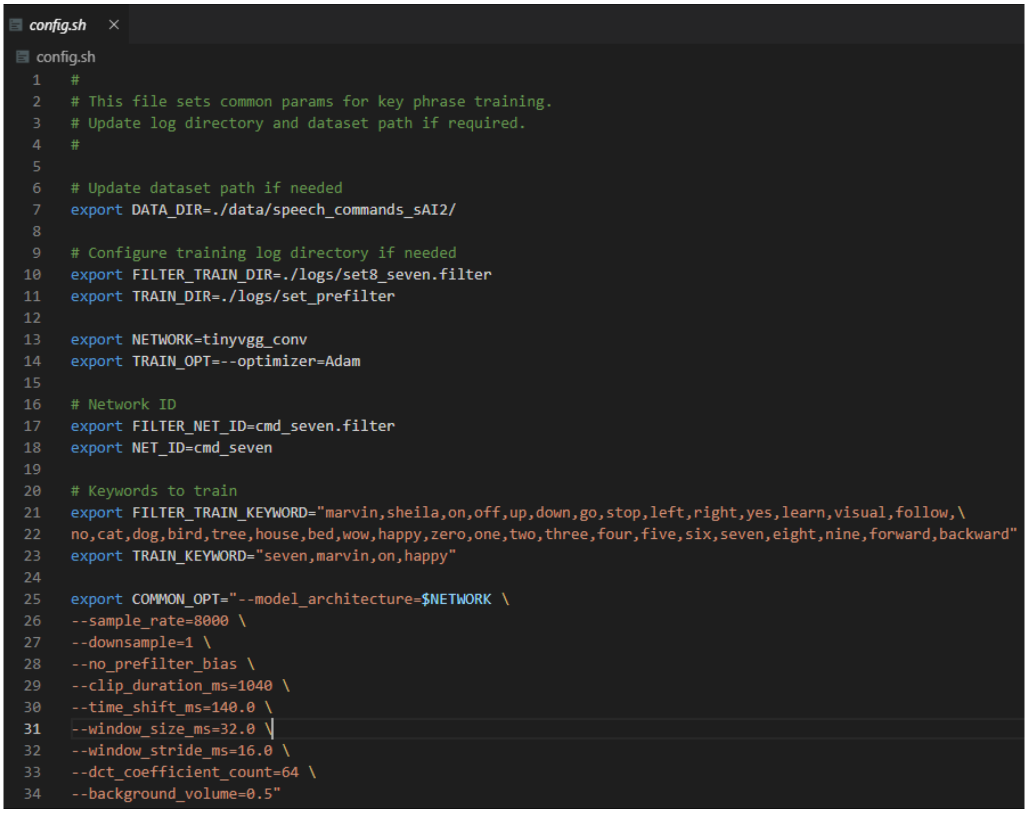
4.2.4. Training from Scratch and/or Transfer Learning

Training of Key Phrase Detection model has two phases:

- Phase 1 – Training with all keywords available in dataset (Filter training).
- Phase 2 – Training for wanted keywords using the checkpoint of phase 1 model (Keyword training).

4.2.4.1. Modifying Training Keywords and Other Configurations

- To configure keywords or other parameters, modify *config.sh* under training directory. Default parameter values for Key Phrase Detection training are as shown in Figure 4.11.



config.sh

```
#
# This file sets common params for key phrase training.
# Update log directory and dataset path if required.
#

# Update dataset path if needed
export DATA_DIR=./data/speech_commands_sAI2/

# Configure training log directory if needed
export FILTER_TRAIN_DIR=./logs/set8_seven.filter
export TRAIN_DIR=./logs/set_prefilter

export NETWORK=tinyvgg_conv
export TRAIN_OPT=--optimizer=Adam

# Network ID
export FILTER_NET_ID=cmd_seven.filter
export NET_ID=cmd_seven

# Keywords to train
export FILTER_TRAIN_KEYWORD="marvin,sheila,on,off,up,down,go,stop,left,right,yes,learn,visual,follow,\
no,cat,dog,bird,tree,house,bed,wow,happy,zero,one,two,three,four,five,six,seven,eight,nine,forward,backward"
export TRAIN_KEYWORD="seven,marvin,on,happy"

export COMMON_OPT="--model_architecture=$NETWORK \
--sample_rate=8000 \
--downsample=1 \
--no_prefilter_bias \
--clip_duration_ms=1040 \
--time_shift_ms=140.0 \
--window_size_ms=32.0 \
--window_stride_ms=16.0 \
--dct_coefficient_count=64 \
--background_volume=0.5"
```

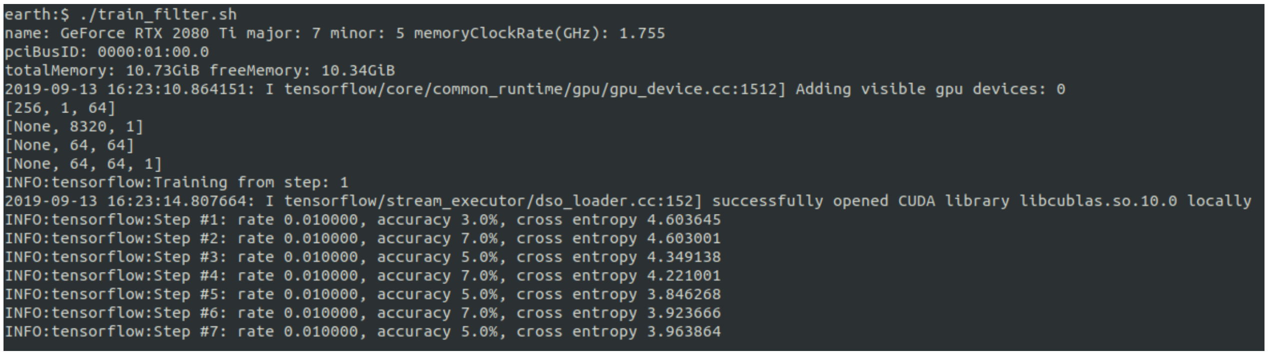
Figure 4.11. Code Snippet – Config File config.sh

- Update *DATA_DIR* with the path to root directory of speech commands dataset.
- Update FILTER_TRAIN_KEYWORD with all available keywords in dataset.
- Update TRAIN_KEYWORD with all keywords that need to be trained by model.
- You can also configure additional parameters in *train_filter.sh* and *train.sh* if needed.

4.2.4.2. Filter training

After configuring (only if required) parameters mentioned in Modifying Training Keywords and Other Configurations section, run the script below to start filter training from scratch.

```
./train_filter.sh
```



```
earth:$ ./train_filter.sh
name: GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.755
pciBusID: 0000:01:00.0
totalMemory: 10.73GiB freeMemory: 10.34GiB
2019-09-13 16:23:10.864151: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1512] Adding visible gpu devices: 0
[256, 1, 64]
[None, 8320, 1]
[None, 64, 64]
[None, 64, 64, 1]
INFO:tensorflow:Training from step: 1
2019-09-13 16:23:14.807664: I tensorflow/stream_executor/dso_loader.cc:152] successfully opened CUDA library libcublas.so.10.0 locally
INFO:tensorflow:Step #1: rate 0.010000, accuracy 3.0%, cross entropy 4.603645
INFO:tensorflow:Step #2: rate 0.010000, accuracy 7.0%, cross entropy 4.603001
INFO:tensorflow:Step #3: rate 0.010000, accuracy 5.0%, cross entropy 4.349138
INFO:tensorflow:Step #4: rate 0.010000, accuracy 7.0%, cross entropy 4.221001
INFO:tensorflow:Step #5: rate 0.010000, accuracy 5.0%, cross entropy 3.846268
INFO:tensorflow:Step #6: rate 0.010000, accuracy 7.0%, cross entropy 3.923666
INFO:tensorflow:Step #7: rate 0.010000, accuracy 5.0%, cross entropy 3.963864
```

Figure 4.12. Key Phrase Detection – Trigger Training with Default Options (Phase 1)

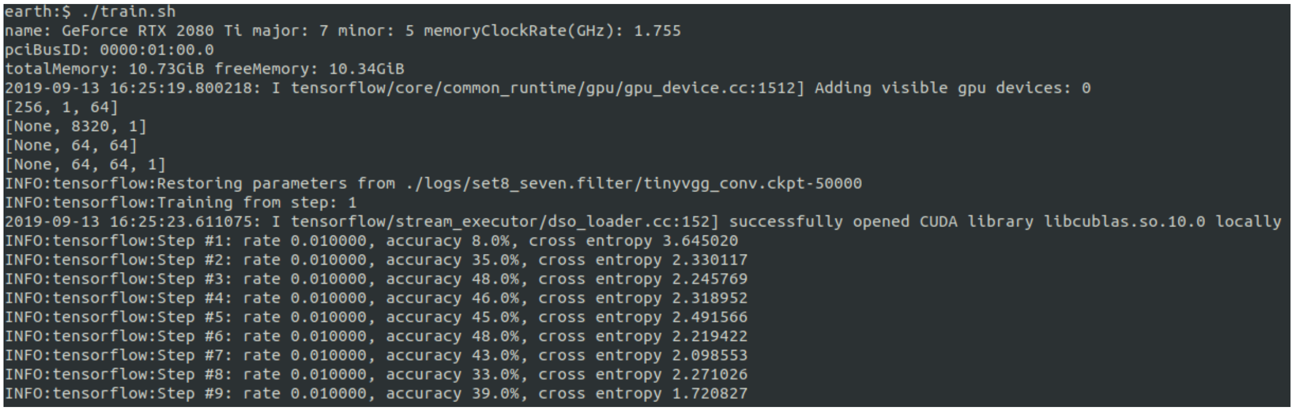
4.2.4.3. Keyword Training

- For phase 2 training, update the path of phase 1 checkpoint in *train.sh* by running the command below:
  - Make sure that below line refers to checkpoints generated by Phase 1 training.

```
TRAIN_OPT = "$TRAIN_OPT –set_prefilter=<path_to_traindir/tinyvgg_conv.ckpt-
50000> --lock_prefilter"
```

- After configuring (only if required) parameters mentioned in Modifying Training Keywords and Other Configurations section, run the script below to start training from scratch.

```
./train.sh
```



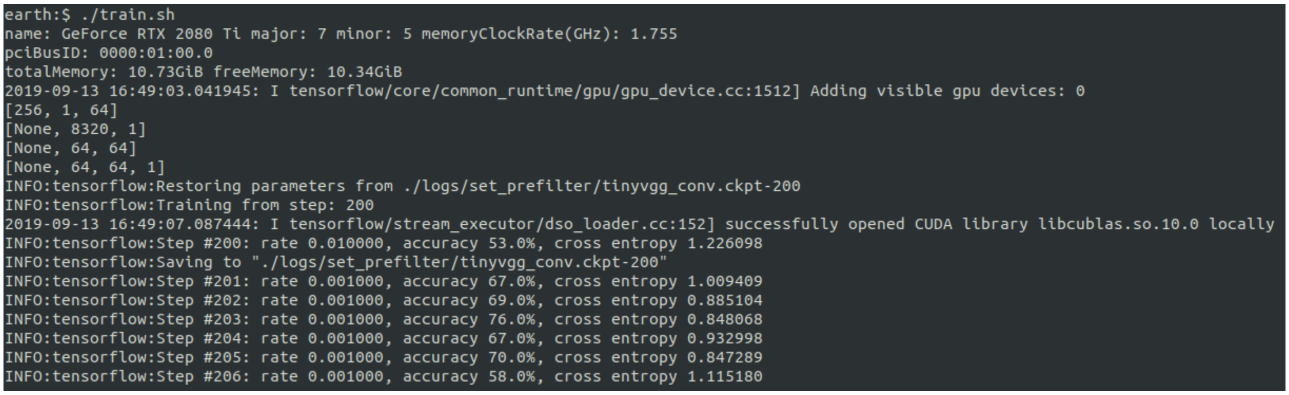
```
earth:$ ./train.sh
name: GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.755
pciBusID: 0000:01:00.0
totalMemory: 10.73GiB freeMemory: 10.34GiB
2019-09-13 16:25:19.800218: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1512] Adding visible gpu devices: 0
[256, 1, 64]
[None, 8320, 1]
[None, 64, 64]
[None, 64, 64, 1]
INFO:tensorflow:Restoring parameters from ./logs/set8_seven.filter/tinyvgg_conv.ckpt-50000
INFO:tensorflow:Training from step: 1
2019-09-13 16:25:23.611075: I tensorflow/stream_executor/dso_loader.cc:152] successfully opened CUDA library libcublas.so.10.0 locally
INFO:tensorflow:Step #1: rate 0.010000, accuracy 8.0%, cross entropy 3.645020
INFO:tensorflow:Step #2: rate 0.010000, accuracy 35.0%, cross entropy 2.330117
INFO:tensorflow:Step #3: rate 0.010000, accuracy 48.0%, cross entropy 2.245769
INFO:tensorflow:Step #4: rate 0.010000, accuracy 46.0%, cross entropy 2.318952
INFO:tensorflow:Step #5: rate 0.010000, accuracy 45.0%, cross entropy 2.491566
INFO:tensorflow:Step #6: rate 0.010000, accuracy 48.0%, cross entropy 2.219422
INFO:tensorflow:Step #7: rate 0.010000, accuracy 43.0%, cross entropy 2.098553
INFO:tensorflow:Step #8: rate 0.010000, accuracy 33.0%, cross entropy 2.271026
INFO:tensorflow:Step #9: rate 0.010000, accuracy 39.0%, cross entropy 1.720827
```

Figure 4.13. Key Phrase Detection – Training from Scratch (Phase 2)

4.2.4.4. Transfer Learning

- For transfer learning, $FILTER_TRAIN_DIR and/or $TRAIN_DIR should point to last iteration's log directory in *config.sh* and the latest checkpoint name should be updated in training script. New checkpoints are stored at the path given in *$TRAIN_DIR* and/or *$FILTER_TRAIN_DIR*.
- Modify below line in *train_filter.sh* and *train.sh* to specify checkpoints from where training should resume:

```
TRAIN_OPT="$TRAIN_OPT --start_checkpoint=$TRAIN_DIR/$NETWORK.ckpt-50000"
```

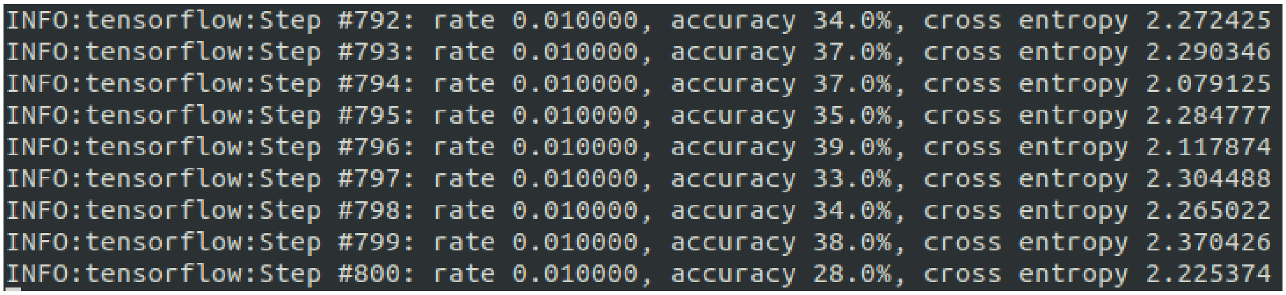


```
earth:$ ./train.sh
name: GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.755
pciBusID: 0000:01:00.0
totalMemory: 10.73GiB freeMemory: 10.34GiB
2019-09-13 16:49:03.041945: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1512] Adding visible gpu devices: 0
[256, 1, 64]
[None, 8320, 1]
[None, 64, 64]
[None, 64, 64, 1]
INFO:tensorflow:Restoring parameters from ./logs/set_prefilter/tinyvgg_conv.ckpt-200
INFO:tensorflow:Training from step: 200
2019-09-13 16:49:07.087444: I tensorflow/stream_executor/dso_loader.cc:152] successfully opened CUDA library libcublas.so.10.0 locally
INFO:tensorflow:Step #200: rate 0.010000, accuracy 53.0%, cross entropy 1.226098
INFO:tensorflow:Saving to "./logs/set_prefilter/tinyvgg_conv.ckpt-200"
INFO:tensorflow:Step #201: rate 0.001000, accuracy 67.0%, cross entropy 1.009409
INFO:tensorflow:Step #202: rate 0.001000, accuracy 69.0%, cross entropy 0.885104
INFO:tensorflow:Step #203: rate 0.001000, accuracy 76.0%, cross entropy 0.848068
INFO:tensorflow:Step #204: rate 0.001000, accuracy 67.0%, cross entropy 0.932998
INFO:tensorflow:Step #205: rate 0.001000, accuracy 70.0%, cross entropy 0.847289
INFO:tensorflow:Step #206: rate 0.001000, accuracy 58.0%, cross entropy 1.115180
```

Figure 4.14. Key Phrase Detection – Trigger Training with Transfer Learning

4.2.4.5. Training Status

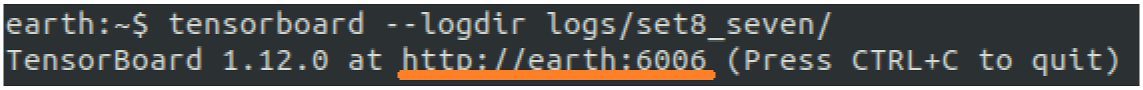
- Training status can be checked in logs by observing different terminologies like cross entropy loss, learning rate, and training accuracy.



```
INFO:tensorflow:Step #792: rate 0.010000, accuracy 34.0%, cross entropy 2.272425
INFO:tensorflow:Step #793: rate 0.010000, accuracy 37.0%, cross entropy 2.290346
INFO:tensorflow:Step #794: rate 0.010000, accuracy 37.0%, cross entropy 2.079125
INFO:tensorflow:Step #795: rate 0.010000, accuracy 35.0%, cross entropy 2.284777
INFO:tensorflow:Step #796: rate 0.010000, accuracy 39.0%, cross entropy 2.117874
INFO:tensorflow:Step #797: rate 0.010000, accuracy 33.0%, cross entropy 2.304488
INFO:tensorflow:Step #798: rate 0.010000, accuracy 34.0%, cross entropy 2.265022
INFO:tensorflow:Step #799: rate 0.010000, accuracy 38.0%, cross entropy 2.370426
INFO:tensorflow:Step #800: rate 0.010000, accuracy 28.0%, cross entropy 2.225374
```

Figure 4.15. Key Phrase Detection – Training Logs

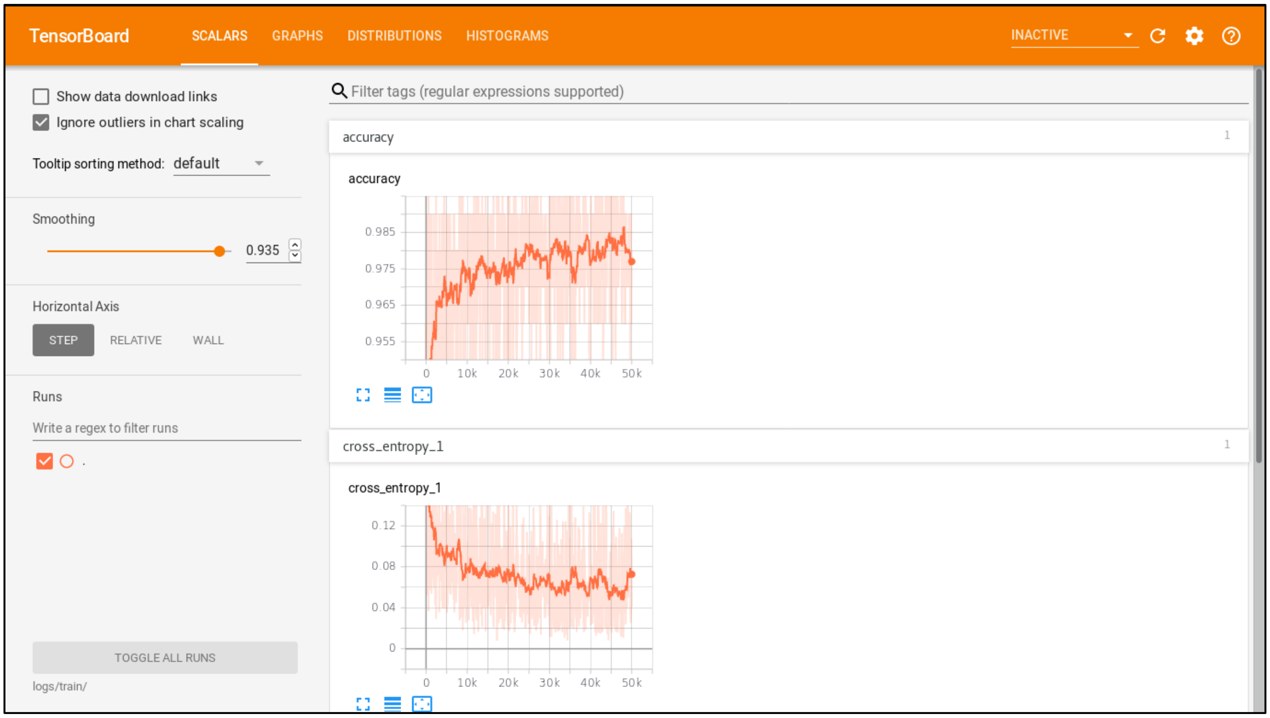
- You can use the TensorBoard utility for checking training status.
  - Run the command below to start TensorBoard:
    `$ tensorboard –logdir=<log directory of training>`



```
earth:~$ tensorboard --logdir logs/set8_seven/
TensorBoard 1.12.0 at http://earth:6006 (Press CTRL+C to quit)
```

Figure 4.16. TensorBoard – Launch

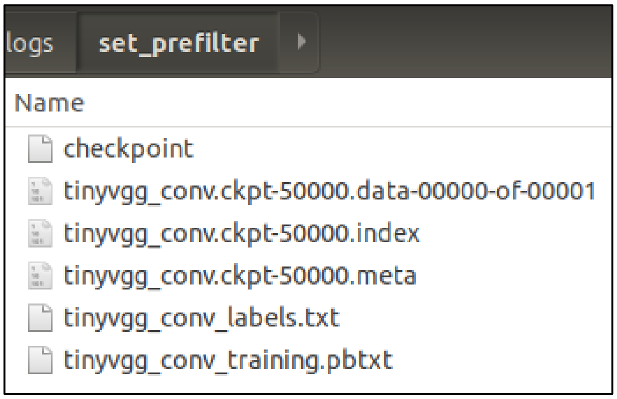
  - This command provides the link *http://<name>:6006* which needs to be copied and open in any browser like Chrome, Firefox, and so on.



☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.935
Horizontal Axis: STEP RELATIVE WALL
Runs
Write a regex to filter runs
☑ ○ .
logs/train/

Figure 4.17. TensorBoard – Link Default Output in Browser

- Similarly, other graphs can be investigated from the available list.
- Check if the *checkpoint*, *data*, *meta* and *index* files are created at the log directory. These files are used for creating the frozen file (*.pb).



logs set_prefilter

Name

checkpoint

tinyvgg_conv.ckpt-50000.data-00000-of-00001

tinyvgg_conv.ckpt-50000.index

tinyvgg_conv.ckpt-50000.meta

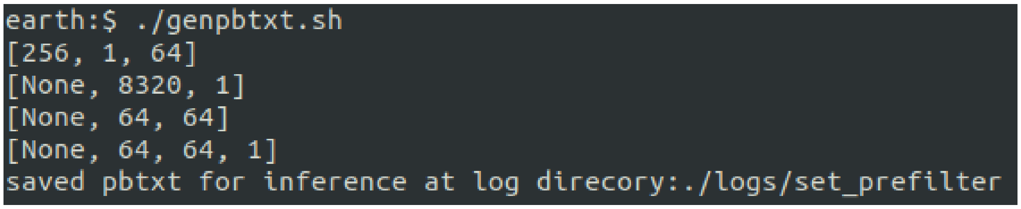
tinyvgg_conv_labels.txt

tinyvgg_conv_training.pbtxt

Figure 4.18. Checkpoint Storage Directory Structure

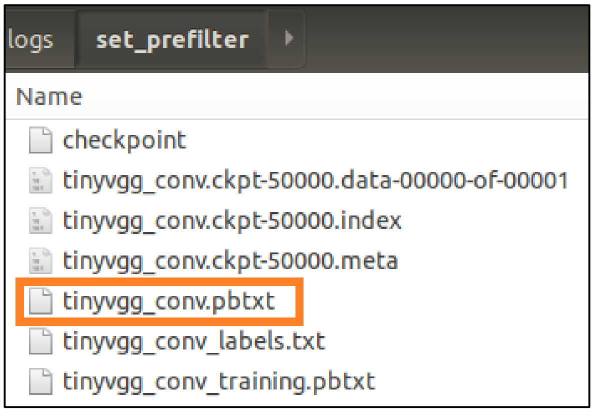
5. Creating Frozen File

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:



```
earth:$ ./genpbtxt.sh
[256, 1, 64]
[None, 8320, 1]
[None, 64, 64]
[None, 64, 64, 1]
saved pbtxt for inference at log direcory:./logs/set_prefilter
```

Figure 5.1. Generating .pbtxt For Inference



```
checkpoint
tinyvgg_conv.ckpt-50000.data-00000-of-00001
tinyvgg_conv.ckpt-50000.index
tinyvgg_conv.ckpt-50000.meta
tinyvgg_conv.pbtxt
tinyvgg_conv_labels.txt
tinyvgg_conv_training.pbtxt
```

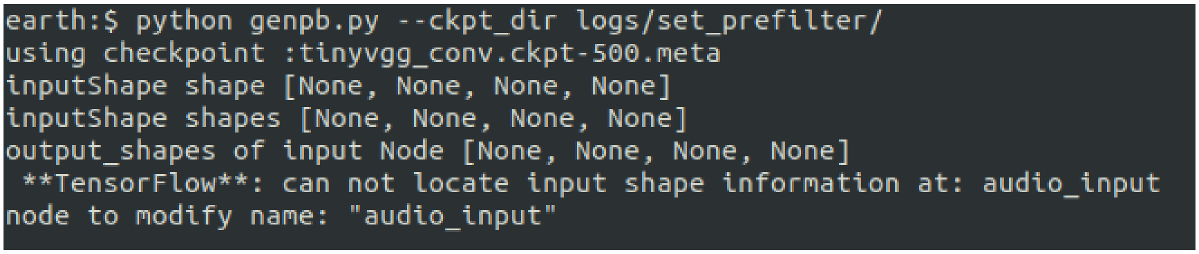
Figure 5.2. Generated .pbtxt for Inference

It generates the .pbtxt for inference under train log directory.

5.1. Generating the Frozen (.pb) File

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

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

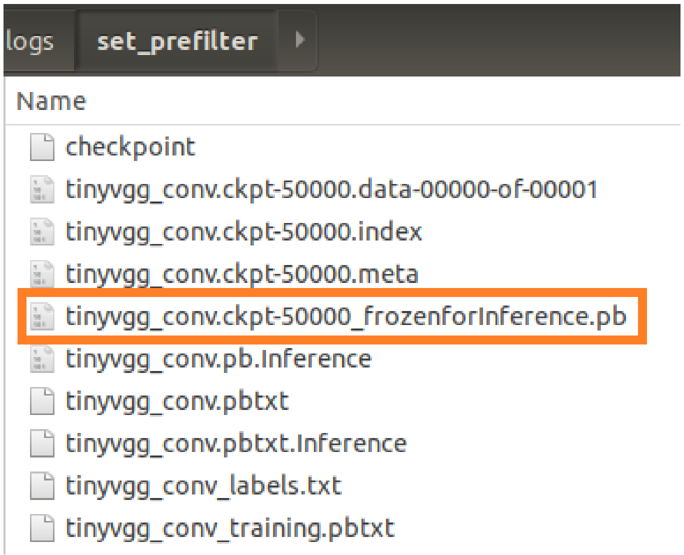


```
earth:$ python genpb.py --ckpt_dir logs/set_prefilter/
using checkpoint :tinyvgg_conv.ckpt-500.meta
inputShape shape [None, None, None, None]
inputShape shapes [None, None, None, None]
output_shapes of input Node [None, None, None, None]
 **TensorFlow**: can not locate input shape information at: audio_input
node to modify name: "audio_input"
```

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

genpb.py uses the generated *.pbtxt* and latest checkpoint in train directory to generate frozen .pb file.

Once the genpb.py is executed successfully, the log directory now contains the *<ckpt-prefix>_frozenforinference.pb* file as shown in Figure 5.4Figure 5.4.



logs set_prefilter

Name

checkpoint
tinyvgg_conv.ckpt-50000.data-00000-of-00001
tinyvgg_conv.ckpt-50000.index
tinyvgg_conv.ckpt-50000.meta
tinyvgg_conv.ckpt-50000_frozenforInference.pb
tinyvgg_conv.pb.Inference
tinyvgg_conv.pbtxt
tinyvgg_conv.pbtxt.Inference
tinyvgg_conv_labels.txt
tinyvgg_conv_training.pbtxt

Figure 5.4. Frozen Inference .pb Output

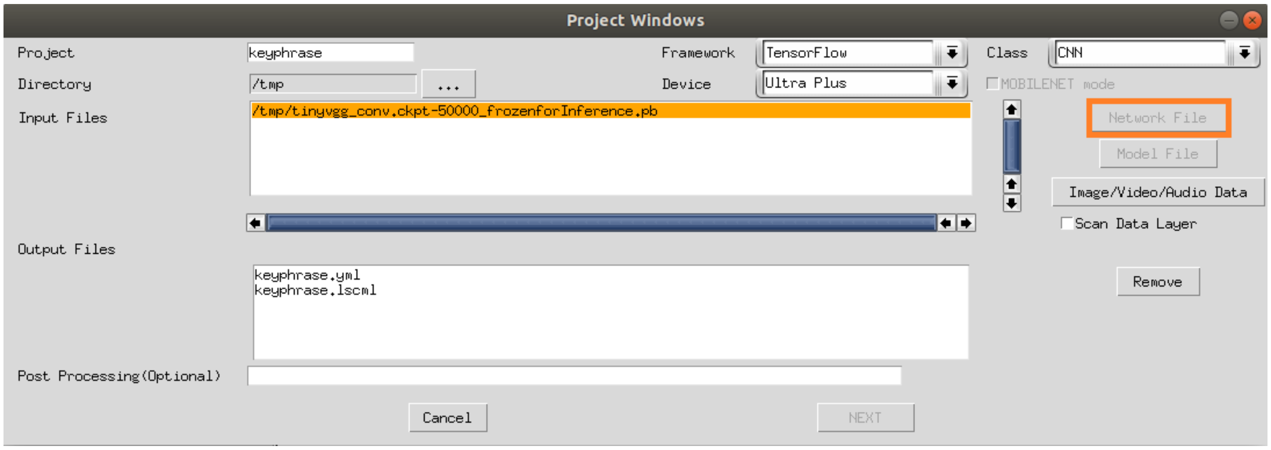
6. Creating Binary File with SensAI

This chapter describes how to generate binary file using the Lattice SensAI version 2.1 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 keyphrase

Directory /tmp

Framework TensorFlow

Device Ultra Plus

Class CNN

□ MOBILENET mode

Input Files

/tmp/tinyvgg_conv.ckpt-50000_frozenforInference.pb

□ Scan Data Layer

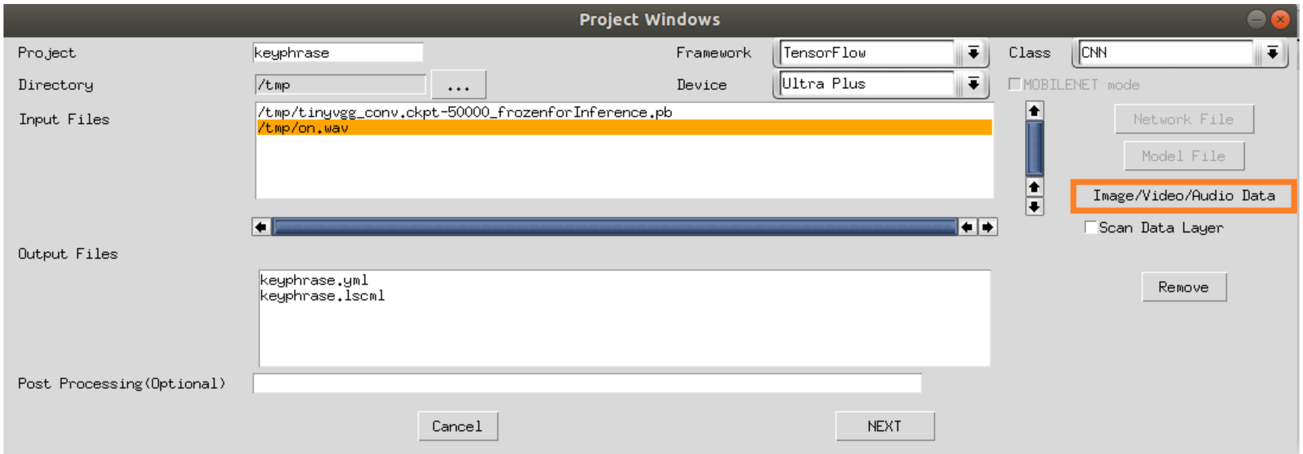
Output Files

keyphrase.yml
keyphrase.lscml

Post Processing(Optional)

Figure 6.2. SensAI –Network File Selection

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



Project keyphrase

Directory /tmp

Framework TensorFlow

Device Ultra Plus

Class CNN

□ MOBILENET mode

Input Files

/tmp/tinyvgg_conv.ckpt-50000_frozenforInference.pb
/tmp/on.wav

□ Scan Data Layer

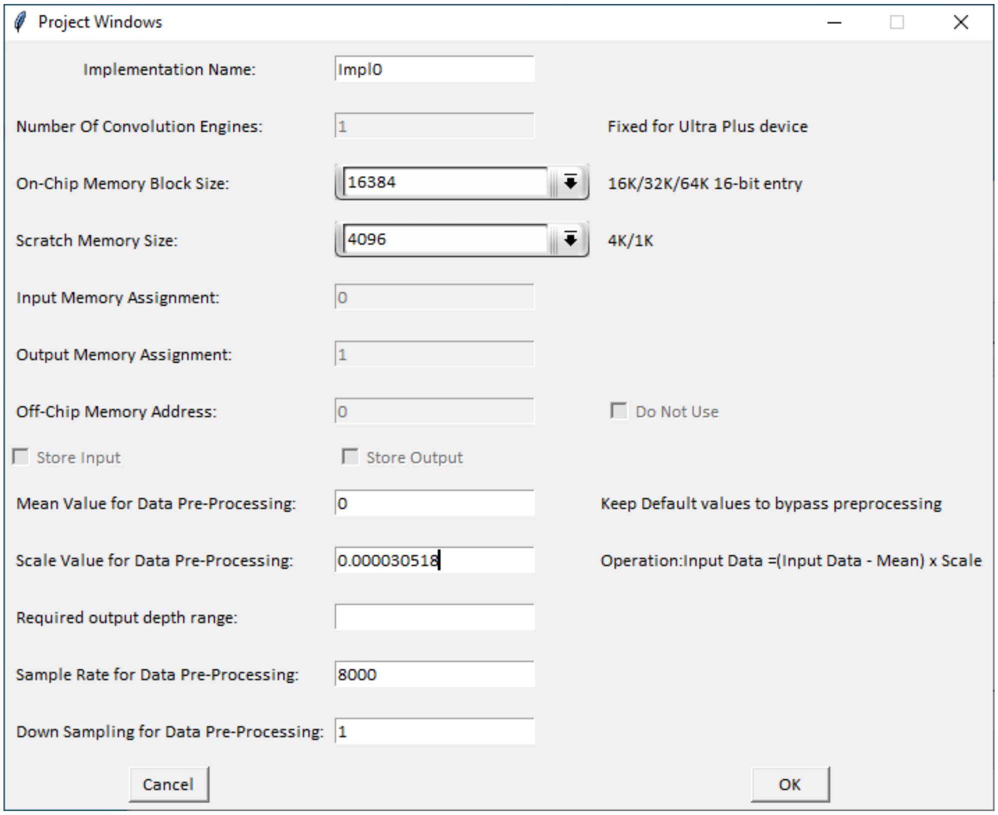
Output Files

keyphrase.yml
keyphrase.lscml

Post Processing(Optional)

Figure 6.3. SensAI –Image Data File Selection

5. Click **Next**.
6. Configure your project settings as shown in Figure 6.4.



Project Windows

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

☐ Store Input ☐ Store Output

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

Scale Value for Data Pre-Processing: 0.000030518 — Operation:Input Data =(Input Data - Mean) x Scale

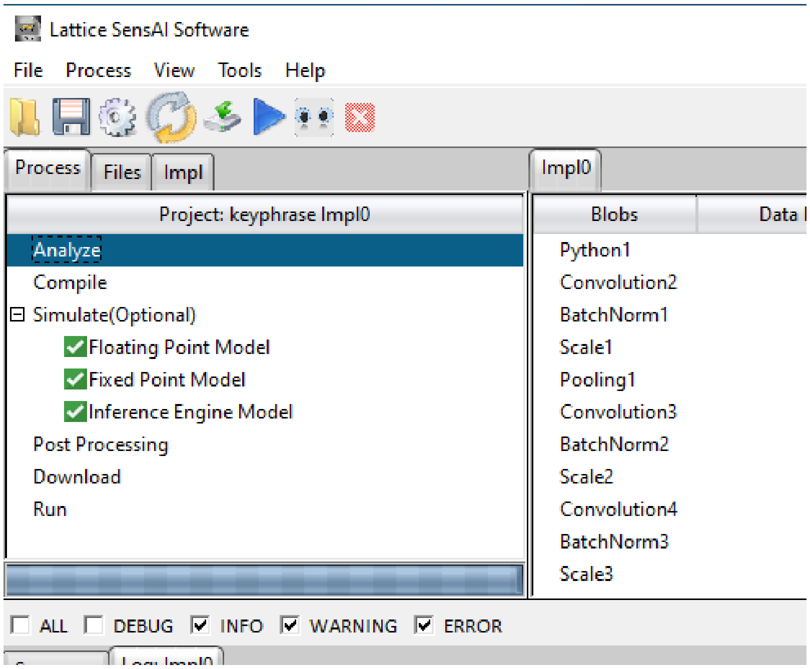
Required output depth range:

Sample Rate for Data Pre-Processing: 8000

Down Sampling for Data Pre-Processing: 1

Figure 6.4. SensAI – Project Settings

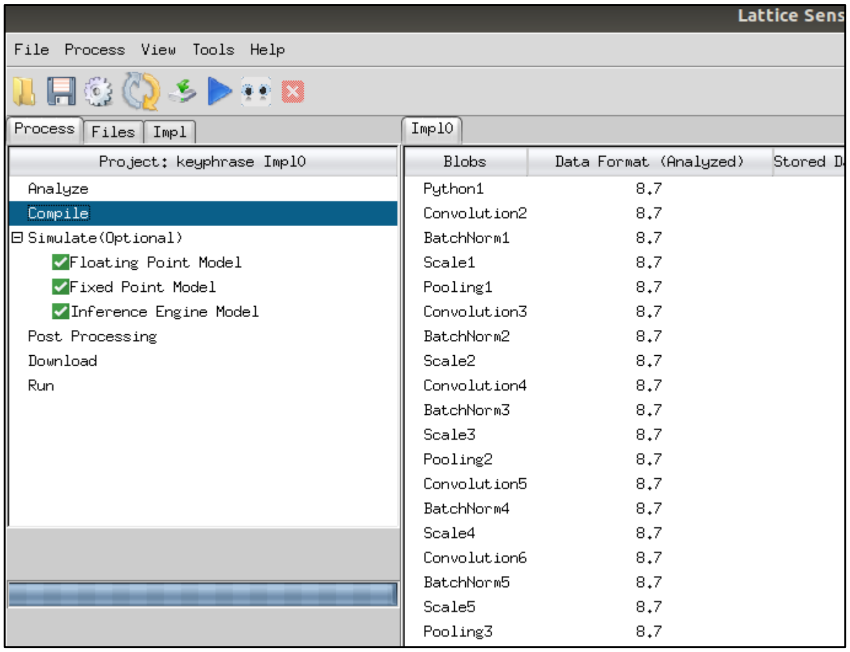
7. Click **OK** to create project.
8. Double-click **Analyze**.



☐ ALL ☐ DEBUG ☑ INFO ☑ WARNING ☑ ERROR

Figure 6.5. SensAI – Analyze Project

9. Double-click **Compile** to generate the Firmware file.



Project: keyphrase Impl0

- Analyze
- Compile
- ⊟ Simulate(Optional)
  - ✓ Floating Point Model
  - ✓ Fixed Point Model
  - ✓ Inference Engine Model
- Post Processing
- Download
- Run

Blobs	Data Format (Analyzed)	Stored D
Python1	8.7	
Convolution2	8.7	
BatchNorm1	8.7	
Scale1	8.7	
Pooling1	8.7	
Convolution3	8.7	
BatchNorm2	8.7	
Scale2	8.7	
Convolution4	8.7	
BatchNorm3	8.7	
Scale3	8.7	
Pooling2	8.7	
Convolution5	8.7	
BatchNorm4	8.7	
Scale4	8.7	
Convolution6	8.7	
BatchNorm5	8.7	
Scale5	8.7	
Pooling3	8.7	

Figure 6.6. Compile Project

Firmware bin file location is displayed in the compilation log. Use the generated firmware and filter bin on the hardware for testing which gives you two output bin files:

- **<Project name>_filter.bin** – contains weights to generate spectrogram from audio input.
- **<Project name>.bin** – contains remaining neural network and weights.

7. Hardware Implementation

7.1. Top Level Information

7.1.1. Block Diagram

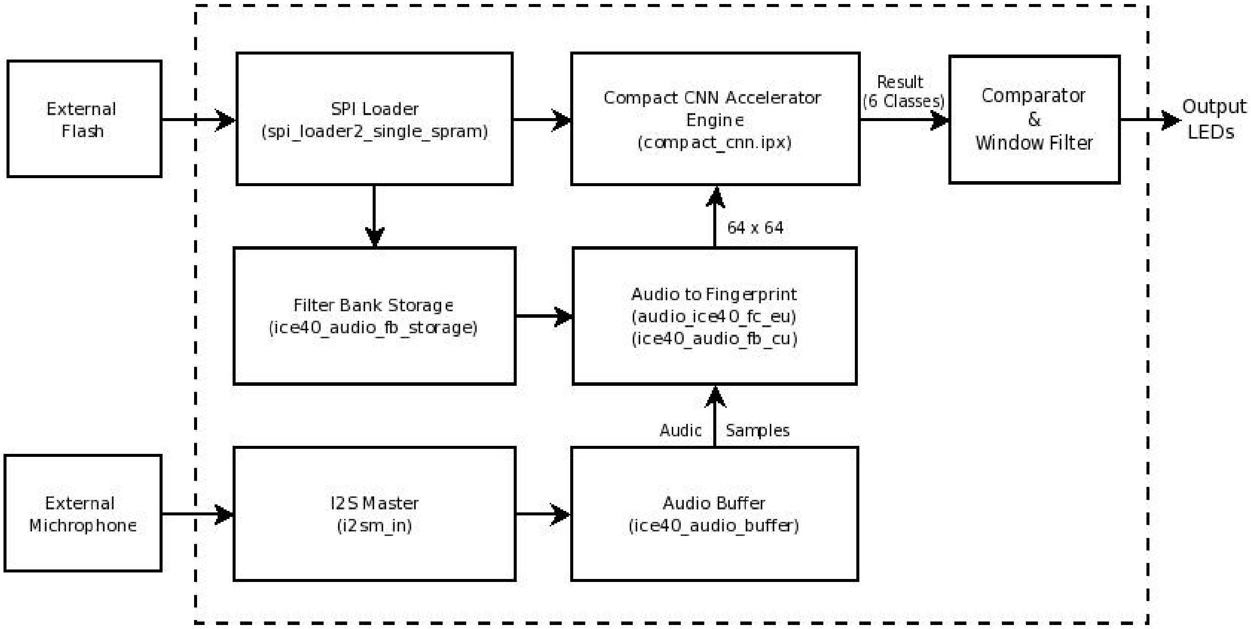

External Flash
SPI Loader (spi_loader2_single_spram)
Compact CNN Accelerator Engine (compact_cnn.ipx)
Result (6 Classes)
Comparator & Window Filter
Output LEDs
64 x 64
Filter Bank Storage (ice40_audio_fb_storage)
Audio to Fingerprint (audio_ice40_fc_eu) (ice40_audio_fb_cu)
Audic Samples
External Microphone
I2S Master (i2sm_in)
Audio Buffer (ice40_audio_buffer)

Figure 7.1. Top Level Block Diagram Key Phrase Detection iCE40

7.1.2. Overall Operational Flow

This section provides a brief idea about the data flow across ice40 Upduino2 Board.

- The I2S master module communicates with I2S microphone on Upduino2 board. It receives the serial audio data from the External Microphone and transports the data to the audio buffer module for storing purpose.
- The SPI loader module reads the external flash for two different files: the command sequence file and the filter bank storage file.
- The command sequence file is sent to the compact CNN IP core while the filter bank storage file is sent to the filter bank module. A filter bank storage module is used while converting audio data into a spectrogram-like picture for CNN input.
- The audio fingerprint module takes the stored audio data from the audio buffer along with the filter bank values from filter bank storage and creates an image of 64 x 64 which is then passed to CNN IP.
- As per command sequence received from SPI Loader, CNN IP performs operations on 64 x 64 input audio image and provides six values as output for six classes in following order:

Table 7.1. CNN Output Format

Index	0	1	2	3	4	5
Class	Silence	Unknown (No Keyword)	Seven (Keyword)	Marvin (Keyword)	On (Keyword)	Happy (Keyword)

- CNN outputs are passed to the audio post processing unit to compare all the six output values and generate difference value between 1st maximum value and 2nd maximum value (No Keyword). The index of 1st maximum value is verified to check whether it belongs to a valid keyword or not (valid Keyword Index values = 2, 3, 4, 5).
- The calculated difference value is compared with two Thresholds defined in Top module which is then used to drive the LEDs. The Detection LED logic uses another Threshold for Keyword Length which relates to the duration of the Keyword spoken. Currently the threshold values are as following:
  - Lower Threshold value = 1536 (Decimal)
  - Higher Threshold Value = 4096 (Decimal)
  - Keyword Length Threshold = 8 (Decimal)
- There are six LEDs with potential to turn on. According to the parameter EN_DEBUG_LED configuration, the demo has following two output representations.
  - Configuration 1 (EN_DEBUG_LED = 0)
    - LED D1 – It is ON if active audio is detected (including noise), and OFF when silence is detected.
    - LED D2 – OFF
    - LED D3 – OFF
    - LED D4 – OFF
    - LED D5 – OFF
    - LED D6 (Detection LED) – It is ON if any of the Keywords (Sheila, Marvin, On, Off) is detected and Keyword Length Threshold is crossed.
  - Configuration 2 (EN_DEBUG_LED = 1)

    LED D1 and D6 are driven same as configuration 1 above. LED D2 to D5 is used for debugging information.
    - LED D1 – It is ON if active audio is detected (including noise), and OFF when silence is detected.
    - LED D2: – t has similar behavior as LED D6.
    - LED D3 – It is ON if any of the Keywords is detected with the calculated difference value greater than Higher Threshold.
    - LED D4 – It is ON if any of the Keywords is detected with the calculated difference value greater than Lower Threshold.
    - LED D5 – It is ON if any of the Keywords is detected with the calculated difference value lesser than both the Thresholds.
    - LED D6 (Detection LED) – It is ON if any of the Keywords (Sheila, Marvin, On, Off) is detected and Keyword Length Threshold is crossed.

7.1.3. Core Customization

Table 7.2. Core Parameters

Parameter	Default (Decimal)	Description
Configurable Parameters		
EN_DEBUD_LED	0	Enable additional LED detection for threshold value debugging. 1: D2 - D5 LEDs represents debug information for keyword detection with difference value in between certain threshold range. 0: D2 - D5 LEDs are always Off.

8. RTL Bitstream Generation

This section provides the procedure for creating your FPGA bitstream file using Lattice Radiant Software.

Note: This reference design includes a Compact CNN IP that requires a license to be able to generate a bitstream. Lattice provides a 30-day evaluation license for this IP for those who want to evaluate the IP and reference design. You can obtain an evaluation license from the Lattice website Software Licensing page.

Lattice Radiant software version 1.1 is required to generate a bitstream along with a software license patch. You can obtain the software patch file from the Lattice website through Lattice Radiant 1.1 Software Patch.

To create the FPGA bitstream file:

1. Open the Lattice Radiant software.

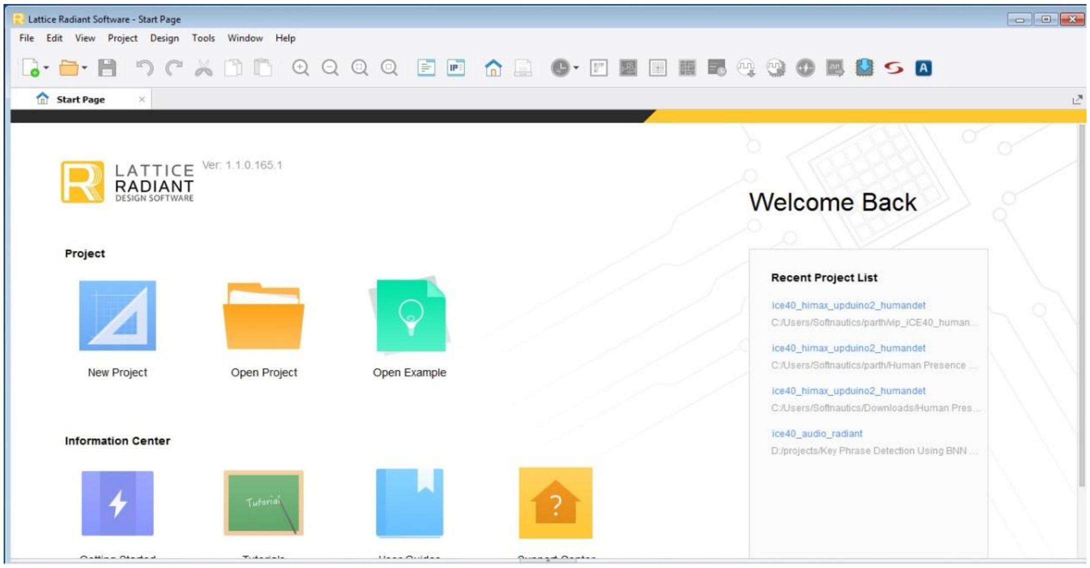
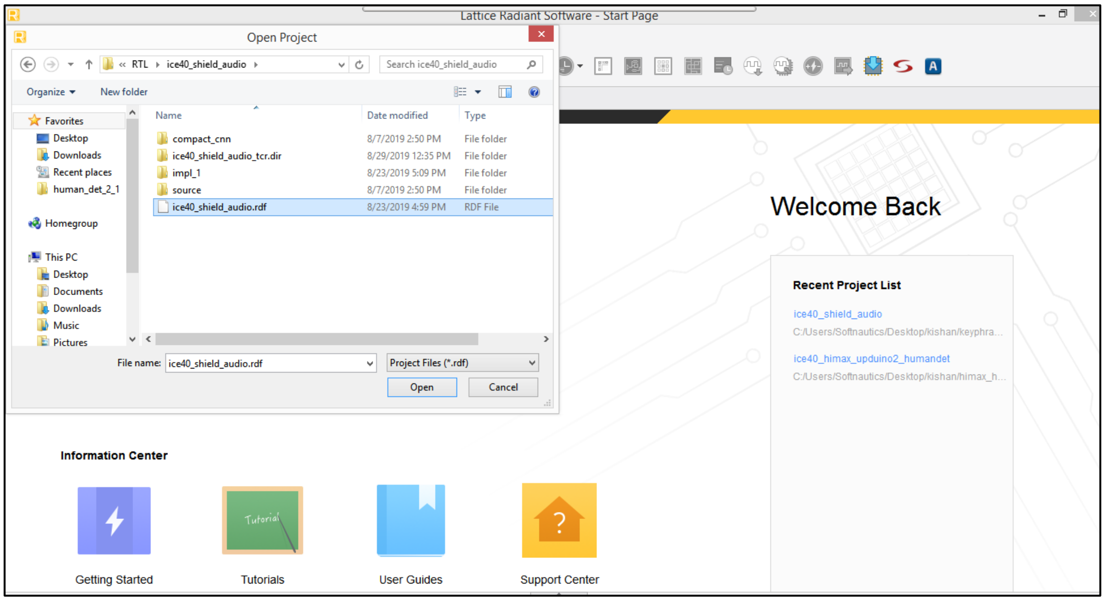

Lattice Radiant Software - Start Page
File Edit View Project Design Tools Window Help
Start Page
LATTICE RADIANT DESIGN SOFTWARE
Ver: 1.1.0.165.1
Welcome Back
Project
New Project
Open Project
Open Example
Information Center
Recent Project List
ice40_himax_upduino2_humandet
C:/Users/Softnautics/parth/vip_iCE40_human...
ice40_himax_upduino2_humandet
C:/Users/Softnautics/parth/Human Presence ...
ice40_himax_upduino2_humandet
C:/Users/Softnautics/Downloads/Human Pres...
ice40_audio_radiant
D:/projects/Key Phrase Detection Using BNN ...

Figure 8.1. Lattice Radiant Software

2. Click **File > Open > Project.** Browse and open Radiant project file for iCE40 key phrase detection RTL.

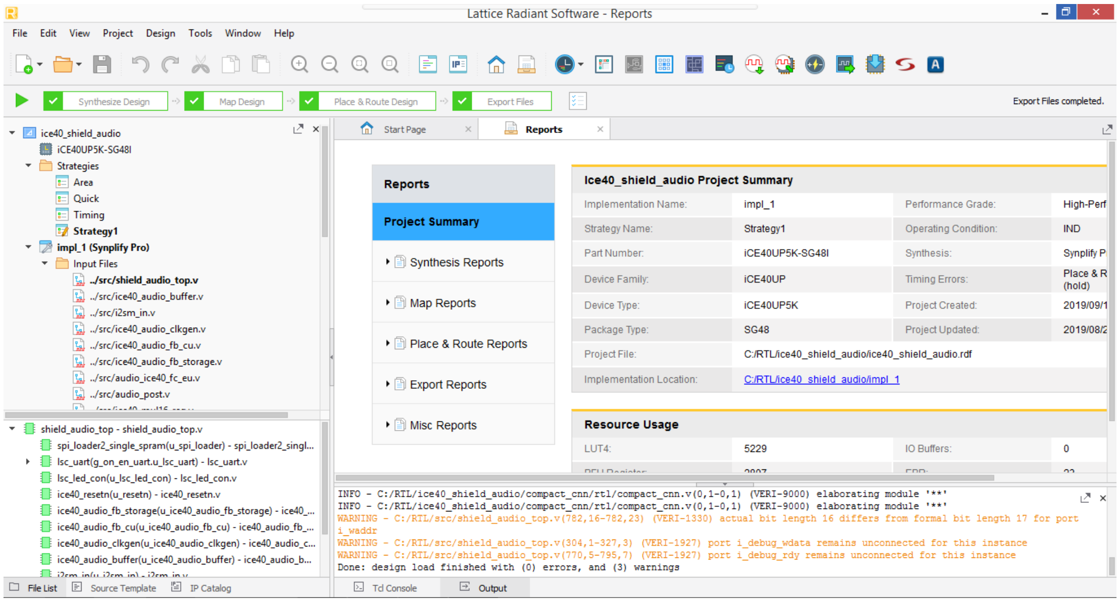


Name	Date modified	Type
compact_cnn	8/7/2019 2:50 PM	File folder
ice40_shield_audio_tcr.dir	8/29/2019 12:35 PM	File folder
impl_1	8/23/2019 5:09 PM	File folder
source	8/7/2019 2:50 PM	File folder
ice40_shield_audio.rdf	8/23/2019 4:59 PM	RDF File

File name: ice40_shield_audio.rdf
Project Files (*.rdf)

Figure 8.2. Lattice Radiant Software – Open Project

3. Click **Export Files** to generate the bit file.


Lattice Radiant Software - Reports
File Edit View Project Design Tools Window Help
Synthesize Design
Map Design
Place & Route Design
Export Files
Export Files completed.
ice40_shield_audio
iCE40UP5K-SG48I
Strategies
Area
Quick
Timing
Strategy1
impl_1 (Synplify Pro)
Input Files
../src/shield_audio_top.v
../src/ice40_audio_buffer.v
../src/i2sm_in.v
../src/ice40_audio_clkgen.v
../src/ice40_audio_fb_cu.v
../src/ice40_audio_fb_storage.v
../src/audio_ice40_fc_eu.v
../src/audio_post.v
shield_audio_top - shield_audio_top.v
spi_loader2_single_spram(u_spi_loader) - spi_loader2_singl...
lsc_uart(g_on_en_uart.u_lsc_uart) - lsc_uart.v
lsc_led_con(u_lsc_led_con) - lsc_led_con.v
ice40_resetn(u_resetn) - ice40_resetn.v
ice40_audio_fb_storage(u_ice40_audio_fb_storage) - ice40_...
ice40_audio_fb_cu(u_ice40_audio_fb_cu) - ice40_audio_fb_...
ice40_audio_clkgen(u_ice40_audio_clkgen) - ice40_audio_c...
ice40_audio_buffer(u_ice40_audio_buffer) - ice40_audio_b...
File List
Source Template
IP Catalog
Start Page
Reports
Project Summary
Synthesis Reports
Map Reports
Place & Route Reports
Export Reports
Misc Reports
Tcl Console
Output

Ice40_shield_audio Project Summary

Implementation Name:	impl_1	Performance Grade:	High-Perf
Strategy Name:	Strategy1	Operating Condition:	IND
Part Number:	iCE40UP5K-SG48I	Synthesis:	Synplify P
Device Family:	iCE40UP	Timing Errors:	Place & R (hold)
Device Type:	iCE40UP5K	Project Created:	2019/09/1
Package Type:	SG48	Project Updated:	2019/08/2
Project File:	C:/RTL/ice40_shield_audio/ice40_shield_audio.rdf		
Implementation Location:	C:/RTL/ice40_shield_audio/impl_1		

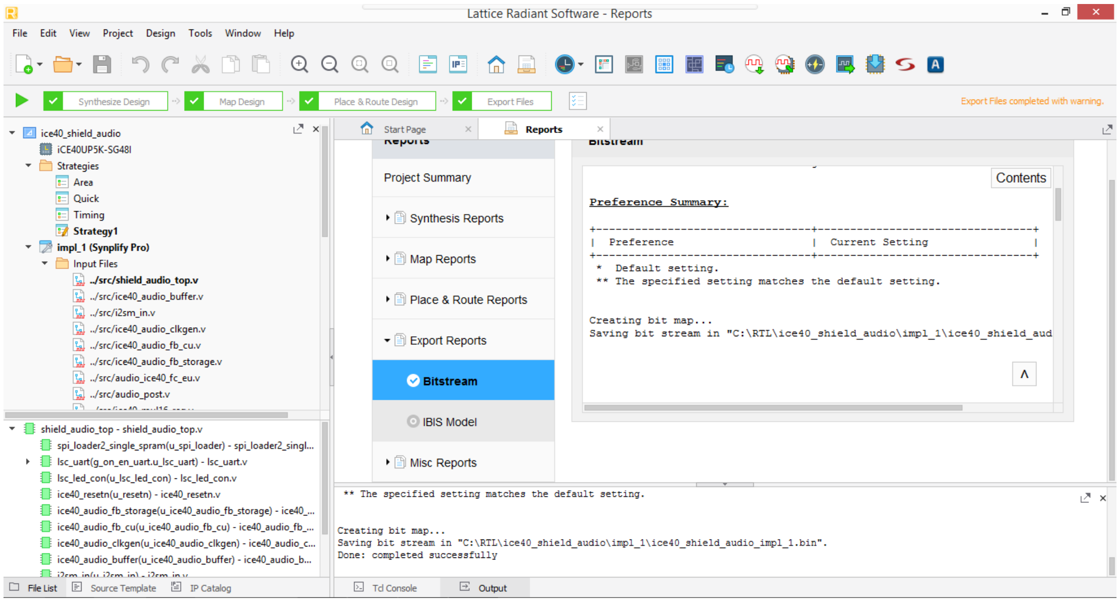
Resource Usage

LUT4:	5229	IO Buffers:	0

```
INFO - C:/RTL/ice40_shield_audio/compact_cnn/rtl/compact_cnn.v(0,1-0,1) (VERI-9000) elaborating module '**'
INFO - C:/RTL/ice40_shield_audio/compact_cnn/rtl/compact_cnn.v(0,1-0,1) (VERI-9000) elaborating module '**'
WARNING - C:/RTL/src/shield_audio_top.v(782,16-782,23) (VERI-1330) actual bit length 16 differs from formal bit length 17 for port
i_waddr
WARNING - C:/RTL/src/shield_audio_top.v(304,1-327,3) (VERI-1927) port i_debug_wdata remains unconnected for this instance
WARNING - C:/RTL/src/shield_audio_top.v(770,5-795,7) (VERI-1927) port i_debug_rdy remains unconnected for this instance
Done: design load finished with (0) errors, and (3) warnings
```

Figure 8.3. Lattice Radiant Software – Bitstream Generation

4. The **Export Reports** displays the generated bitstream as shown in Figure 8.4.


Lattice Radiant Software - Reports
File Edit View Project Design Tools Window Help
Synthesize Design
Map Design
Place & Route Design
Export Files
Export Files completed with warning.
ice40_shield_audio
iCE40UP5K-SG48I
Strategies
Area
Quick
Timing
Strategy1
impl_1 (Synplify Pro)
Input Files
../src/shield_audio_top.v
../src/ice40_audio_buffer.v
../src/i2sm_in.v
../src/ice40_audio_clkgen.v
../src/ice40_audio_fb_cu.v
../src/ice40_audio_fb_storage.v
../src/audio_ice40_fc_eu.v
../src/audio_post.v
shield_audio_top - shield_audio_top.v
spi_loader2_single_spram(u_spi_loader) - spi_loader2_singl...
lsc_uart(g_on_en_uart.u_lsc_uart) - lsc_uart.v
lsc_led_con(u_lsc_led_con) - lsc_led_con.v
ice40_resetn(u_resetn) - ice40_resetn.v
ice40_audio_fb_storage(u_ice40_audio_fb_storage) - ice40_...
ice40_audio_fb_cu(u_ice40_audio_fb_cu) - ice40_audio_fb_...
ice40_audio_clkgen(u_ice40_audio_clkgen) - ice40_audio_c...
ice40_audio_buffer(u_ice40_audio_buffer) - ice40_audio_b...
File List
Source Template
IP Catalog
Start Page
Reports
Project Summary
Synthesis Reports
Map Reports
Place & Route Reports
Export Reports
Bitstream
IBIS Model
Misc Reports
Contents
Tcl Console
Output

```
Preference Summary:

+---------------------------------+---------------------------------+
|  Preference                     |  Current Setting                |
+---------------------------------+---------------------------------+
 *  Default setting.
 ** The specified setting matches the default setting.


Creating bit map...
Saving bit stream in "C:\RTL\ice40_shield_audio\impl_1\ice40_shield_aud
```

```
** The specified setting matches the default setting.


Creating bit map...
Saving bit stream in "C:\RTL\ice40_shield_audio\impl_1\ice40_shield_audio_impl_1.bin".
Done: completed successfully
```

Figure 8.4. Lattice Radiant Software – Bitstream Generation Export Reports

9. Programming the Key Phrase Detection Demo

9.1. Functional Description

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

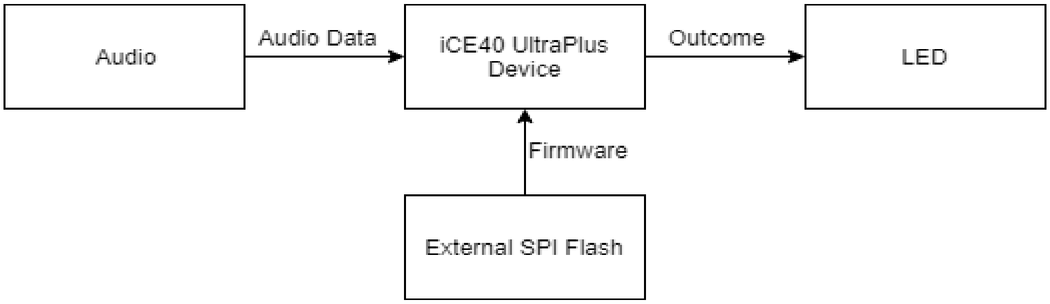


Figure 9.1. iCE40 Key Phrase Demo Diagram

9.2. Programming Key Phrase Detection Demo on iCE40 SPI Flash

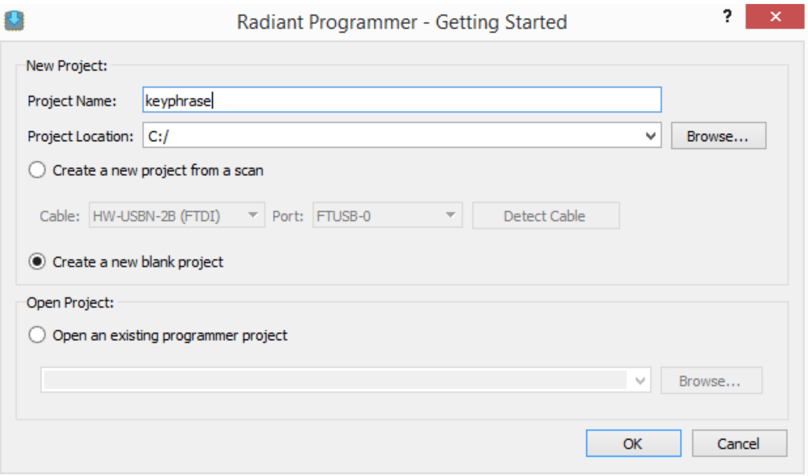
This section provides the procedure for programming the SPI Flash on the Himax HM01B0 UPduino Shield Board.

Two different files 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. Note that the USB connector on board is delicate, so handle it with care.
2. Start Radiant Programmer.
3. In the Radiant Programmer **Getting Started** dialog box, select **Create a new blank project.**
4. Click **OK.**


Radiant Programmer - Getting Started

New Project:

Project Name: keyphrase

Project Location: C:/

○ Create a new project from a scan

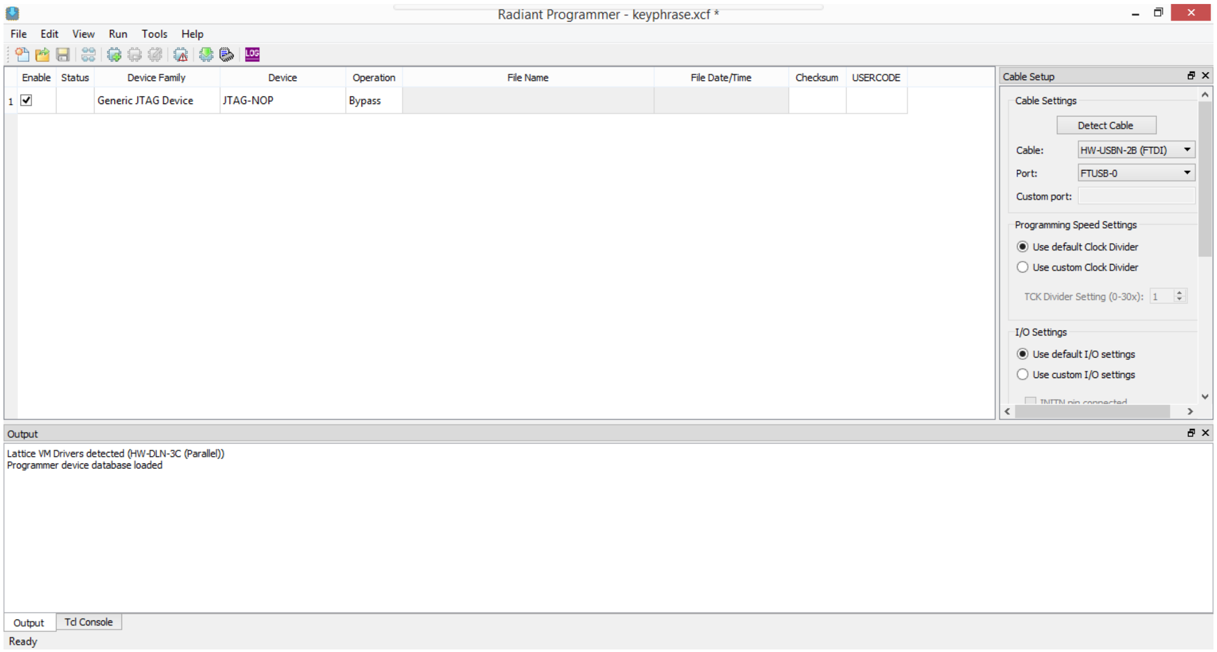
Cable: HW-USBN-2B (FTDI) Port: FTUSB-0

● Create a new blank project

Open Project:

○ Open an existing programmer project

Figure 9.2. Radiant Programmer – Creating New Project



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

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

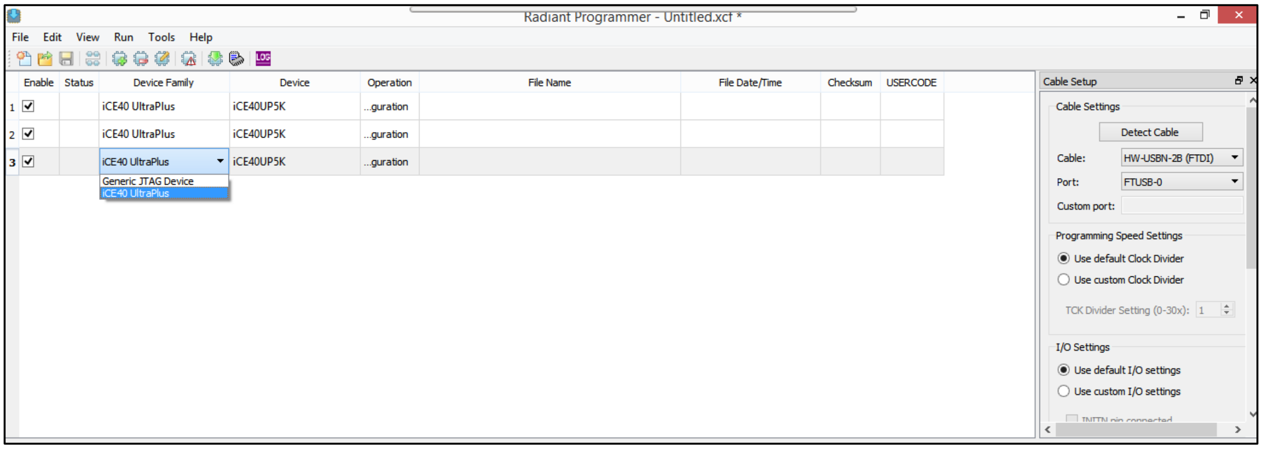
Programming Speed Settings
● Use default Clock Divider
○ Use custom Clock Divider
TCK Divider Setting (0-30x): 1

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
```

Figure 9.3. Radiant Programmer – Initial Project Window

5. Initially, the *.xcf* have the option to add only one bin file. But since you need to program three bin files in case of key phrase demo, add two more devices by clicking the button from the toolbar.
6. Set Device Family to **iCE40 UltraPlus** and Device to **iCE40UP5K** for all three cases.



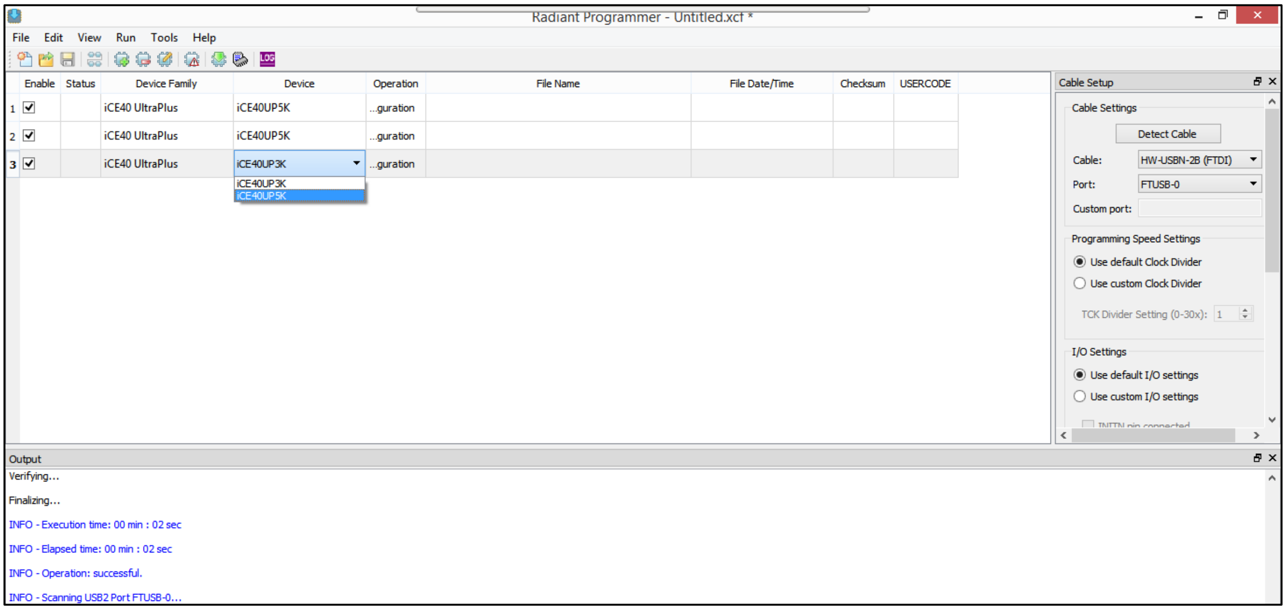
	Enable	Status	Device Family	Device	Operation	File Name	File Date/Time	Checksum	USERCODE
1	☑		iCE40 UltraPlus	iCE40UP5K	...guration				
2	☑		iCE40 UltraPlus	iCE40UP5K	...guration				
3	☑		iCE40 UltraPlus	iCE40UP5K	...guration				

Cable Settings
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

Figure 9.4. Radiant Programmer – iCE40 UltraPlus Device Family Selection



	Enable	Status	Device Family	Device	Operation	File Name	File Date/Time	Checksum	USERCODE
1	☑		iCE40 UltraPlus	iCE40UP5K	...guration				
2	☑		iCE40 UltraPlus	iCE40UP5K	...guration				
3	☑		iCE40 UltraPlus	iCE40UP3K iCE40UP3K iCE40UP5K	...guration				

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): 1

I/O Settings

● Use default I/O settings

○ Use custom I/O settings

□ INITN pin connected

Output

```
Verifying...

Finalizing...

INFO - Execution time: 00 min : 02 sec

INFO - Elapsed time: 00 min : 02 sec

INFO - Operation: successful.

INFO - Scanning USB2 Port FTUSB-0...
```

Figure 9.5. Radiant Programmer – iCE40 UltraPlus Device Selection

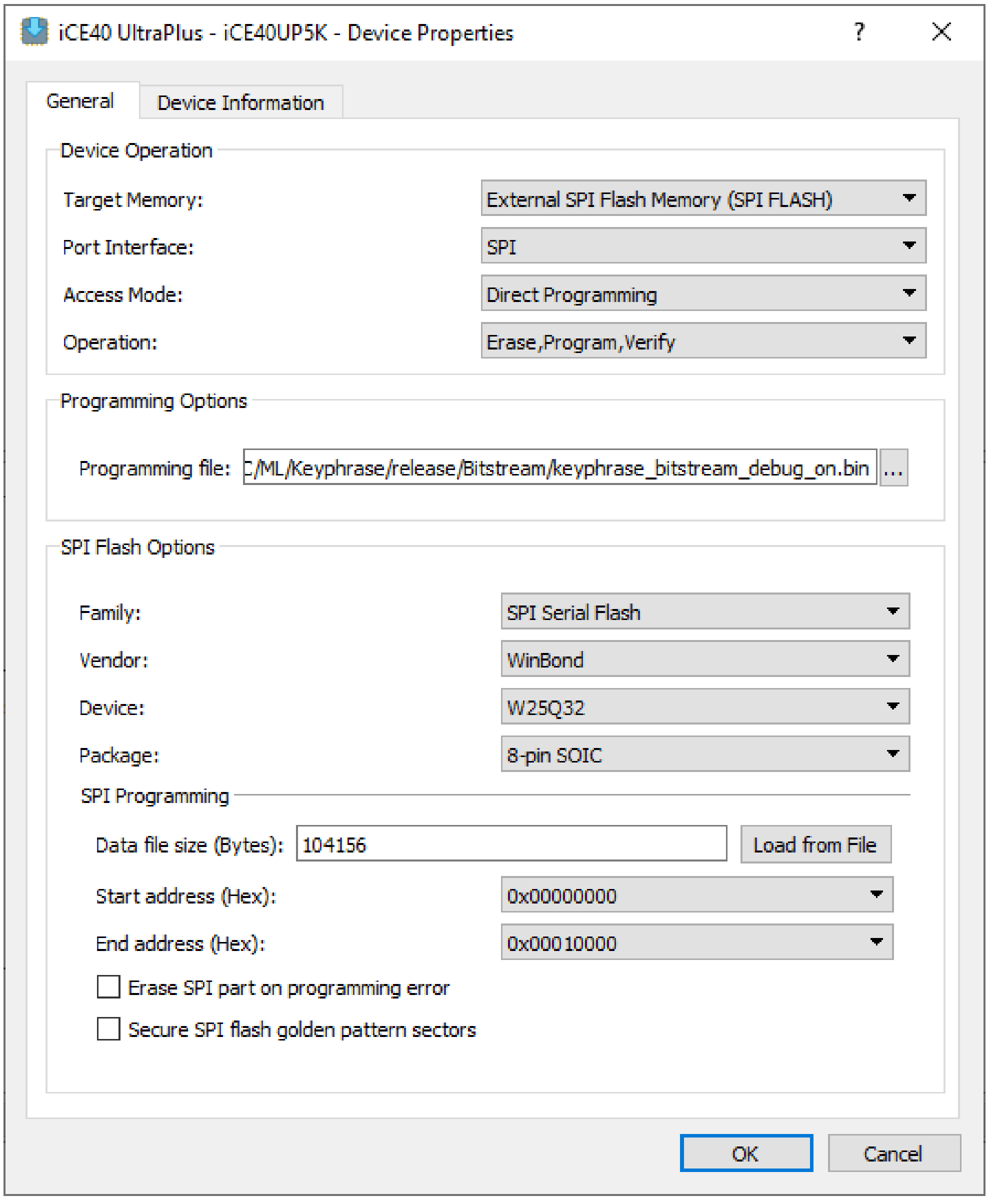
7. Click the iCE40 UltraPlus row and select **Edit > Device Properties**.
8. In the **Device Properties** dialog box, apply the settings below that are common to the two files to program.

   Under Device Operation, select the options below:

   - **Target Memory – External SPI Flash Memory**
   - **Port Interface – SPI**
   - **Access Mode – Direct Programming**
   - **Operation – Erase, Program, Verify**

   Under SPI Flash Options, select the options below:

   - **Family – SPI Serial Flash**
   - **Vendor – Winbond**
   - **Device – W25Q32**
   - **Package – 8-pin SOIC**
9. To program the bitstream file, select the options below as shown inFigure 9.6.
   - Under Programming Options, select the key phrase RTL bitstream file in Programming file.
   - Click Load from File to update the Data file size (Bytes) value.
   - Ensure that the following addresses are correct:
     - Start Address (Hex) – **0x00000000**
     - End Address (Hex) – **0x00010000**
10. 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/ML/Keyphrase/release/Bitstream/keyphrase_bitstream_debug_on.bin

SPI Flash Options

Family: SPI Serial Flash

Vendor: WinBond

Device: W25Q32

Package: 8-pin SOIC

SPI Programming

Data file size (Bytes): 104156

Start address (Hex): 0x00000000

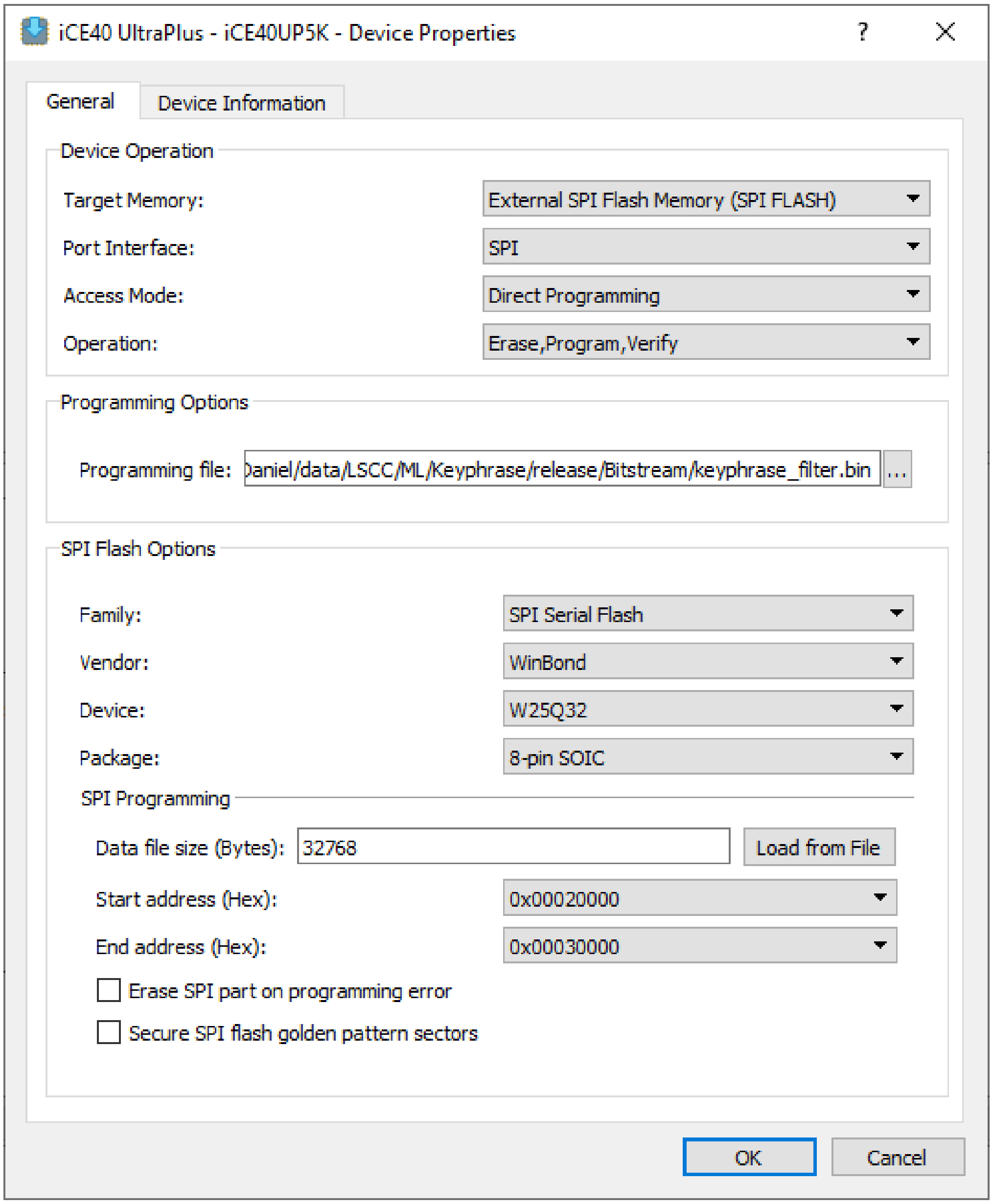
End address (Hex): 0x00010000

□ Erase SPI part on programming error

□ Secure SPI flash golden pattern sectors

Figure 9.6. Radiant Programmer – Bitstream Flashing Settings

11. To program the filter binary firmware for generating the spectrogram, select the options below as shown in Figure 9.7.
    - Under Programming Options, select the key phrase filter bin firmware generated by SensAI tool.
    - Click Load from File to update the Data file size (Bytes) value. Change **Data file size (Bytes)** value to **32768**.
    - Ensure that the following addresses are correct:
        - Start Address (Hex) – **0x00020000**
        - End Address (Hex) – **0x00030000**
12. 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:)aniel/data/LSCC/ML/Keyphrase/release/Bitstream/keyphrase_filter.bin

SPI Flash Options

Family: SPI Serial Flash

Vendor: WinBond

Device: W25Q32

Package: 8-pin SOIC

SPI Programming

Data file size (Bytes): 32768

Start address (Hex): 0x00020000

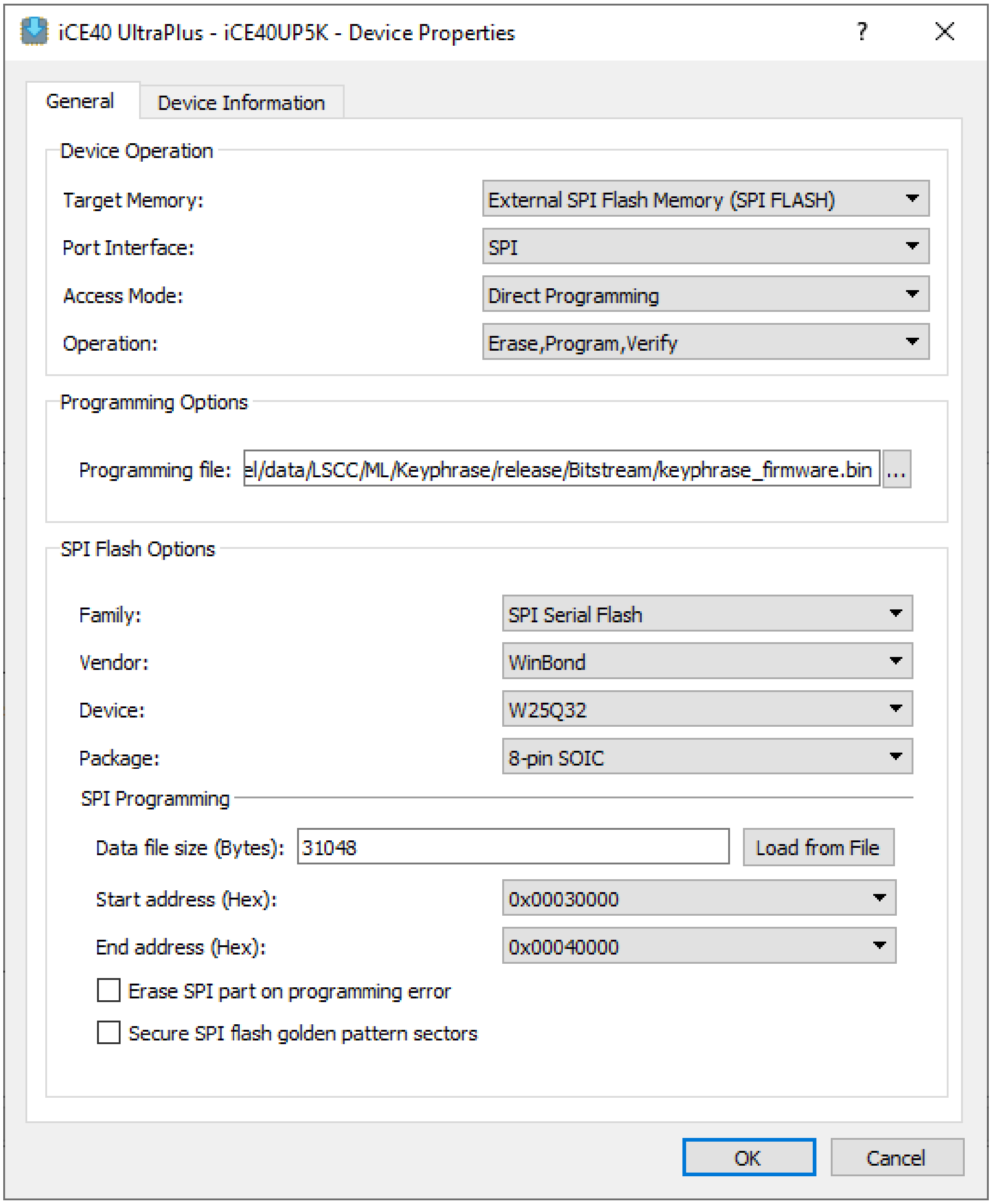
End address (Hex): 0x00030000

□ Erase SPI part on programming error

□ Secure SPI flash golden pattern sectors

Figure 9.7. Radiant Programmer – Filter-Firmware Bin File Flashing Setting

13. To program firmware bin which contains model architecture, select the options below as shown in Figure 9.8.
    - Under Programming Options, select the key phrase firmware binary file generated by SensAI tool.
    - Click Load from File to update the Data file size (Bytes) value. Change **Data file size (Bytes)** value to **31048** (In case model is changed, write the actual size of the file).
    - Ensure that the following addresses are correct:
        - Start Address (Hex) – **0x00030000**
        - End Address (Hex) – **0x00040000**
14. Click **OK**.



iCE40 UltraPlus - iCE40UP5K - Device Properties

General | Device Information

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: el/data/LSCC/ML/Keyphrase/release/Bitstream/keyphrase_firmware.bin

SPI Flash Options

Family: SPI Serial Flash

Vendor: WinBond

Device: W25Q32

Package: 8-pin SOIC

SPI Programming

Data file size (Bytes): 31048

Start address (Hex): 0x00030000

End address (Hex): 0x00040000

☐ Erase SPI part on programming error

☐ Secure SPI flash golden pattern sectors

Figure 9.8. Radiant Programmer – Firmware Bin File Flashing Setting

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

9.3. Run iCE40 Key Phrase Detection Demo on Hardware

To run the demo and observe results on the board:

1. Power ON the Himax HM01B0 UPduino Shield Board.
2. Speak the keyword in front of the board.
3. An LED light turns on if the keyword is detected. Refer to Figure 9.9 for the LED information.

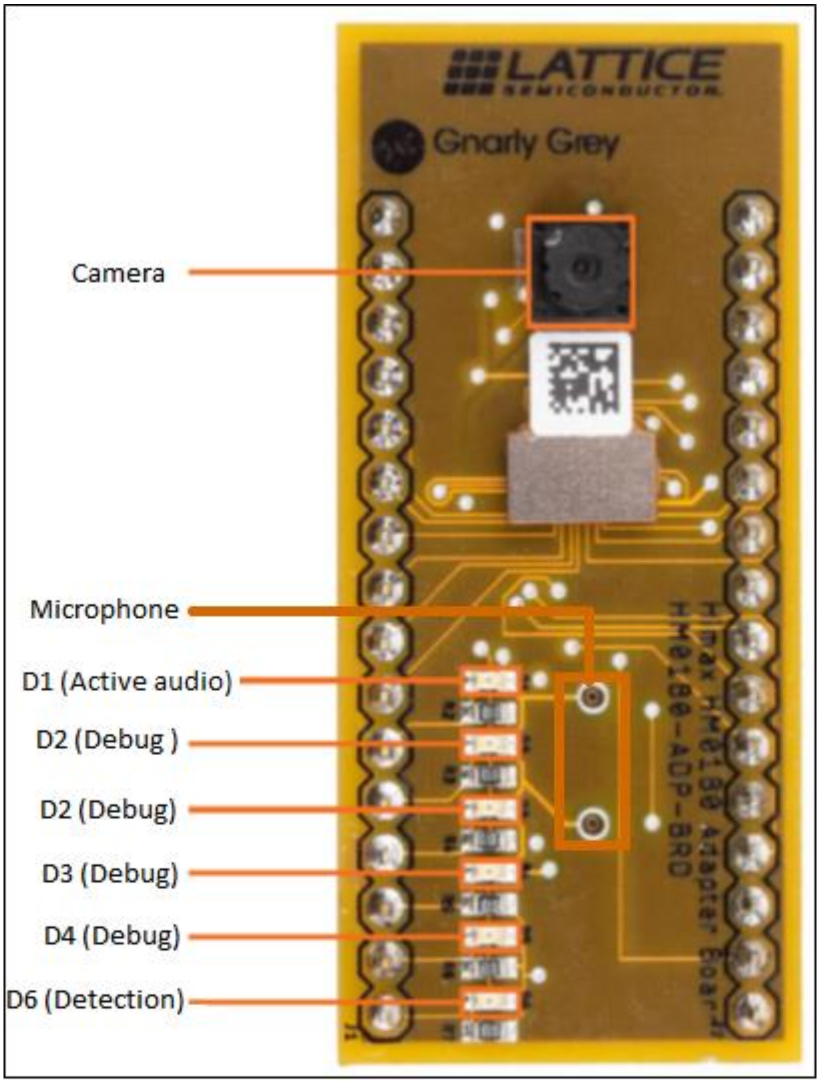


Figure 9.9. Camera and LED Location

- **D1** – It is ON if active audio is detected (including noise), and OFF when silence is detected.
- **D2-D4** – These are Debug LEDs. For more information, refer to Overall Operational Flow section.
- **D6** – This LED is turned on when the keyword is detected.

Technical Support Assistance

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

Revision History

Revision 1.0, October 2019

Section	Change Summary
All	Initial release.

LATTICE SEMICONDUCTOR®

www.latticesemi.com