# RISC-V SoC-Based Human Presence Detection using CrossLink-NX VVML Board

## Reference Design

FPGA-RD-02230-1.0

June 2021

## Disclaimers

## Contents

## Figures

## Tables

# Acronyms in This Document

A list of acronyms used in this document.

| Acronym | Definition |
|---|---|
| AHB | Advanced High performance Bus |
| CKPT | Checkpoint |
| CNN | Convolutional Neural Network |
| CSI | Camera Serial Interface |
| cuDNN | CUDA® Deep Neural Network |
| EVDK | Embedded Vision Development Kit |
| FPGA | Field-Programmable Gate Array |
| GPIO | General Purpose Input output |
| GPU | Graphics Processing Unit |
| I2C | Inter-Integrated Circuit |
| IP | Intellectual Property |
| LED | Light-emitting diode |
| MIPI | Mobile Industry Processor Interface |
| ML | Machine Learning |
| MLE | Machine Learning Engine |
| NN | Neural Network |
| NNC | Neural Network Compiler |
| RAM | Random Access Memory |
| RD | Reference Design |
| RISC | Reduced Instruction Set Computers |
| RW | Read and Write |
| SD | Secure Digital |
| SDHC | Secure Digital High Capacity |
| SDXC | Secure Digital eXtended Capacity |
| SoC | System on Chip |
| SPI | Serial Peripheral Interface |
| UART | Universal Asynchronous Receiver/Transmitter |
| USB | Universal Serial Bus |
| VIP | Video Interface Platform |
| VVML | Voice and Vision Machine Learning Board |

# 1. Introduction

This document describes the Lattice Semiconductor RISC-V SoC-based human presence detection design and process using the CrossLink™-NX Voice and Vision Machine Learning (VVML) Board platform.

## 1.1. Design Process Overview

The design process involves the following steps:

1. Training the model
    - Setting up the basic environment
    - Preparing the dataset
    - Preparing the 64 x 64 image
    - Labeling dataset of human bounding box
    - Training the machine
    - Training the machine and creating the checkpoint data
    - Creating the frozen file (*.pb)
2. Compiling Neural Network to TensorFlow Lite and C array
3. Generating the .bin file using C/C++ Lattice Propel™ project
4. FPGA design
    - Creating the FPGA bitstream file

# 2. Setting up the Basic Environment

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

## 2.1. Software and Hardware Requirements

### 2.1.1. Software

- Lattice Propel version 2.0
  Refer to http://www.latticesemi.com/LatticePropel
- Lattice Radiant version 2.2
  Refer to http://www.latticesemi.com/latticeradiant

### 2.1.2. Hardware

CrossLink-NX Voice and Vision Machine Learning (VVML) Board

Refer to https://www.latticesemi.com/products/developmentboardsandkits/crosslink-nxvoiceandvisionmachinelearning.

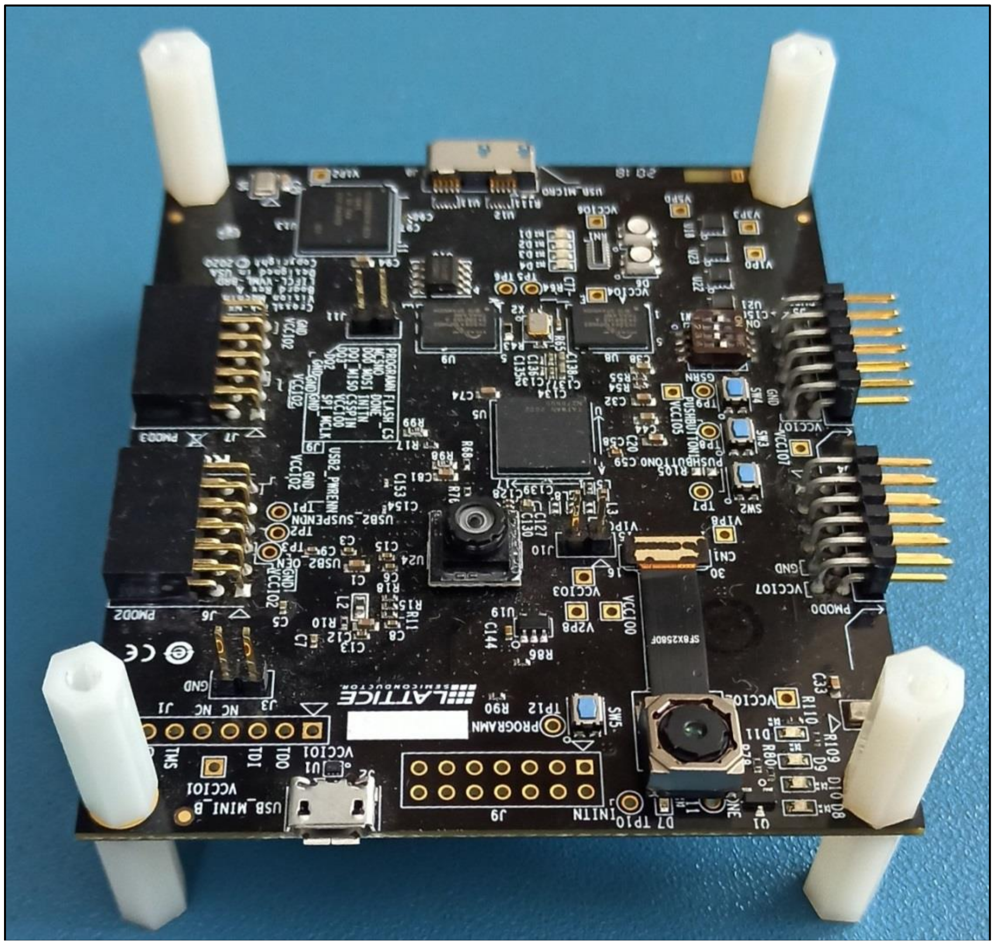


**Figure 2.1. CrossLink-NX Voice and Vision Machine Learning Board**

## 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.

**Note:** The NVIDIA library and the TensorFlow versions are dependent on the PC and the Ubuntu/Windows version.

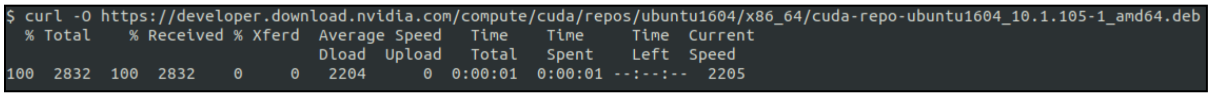
### 2.2.1. Installing the NVIDIA CUDA and cuDNN Library for Machine Learning Training on GPU

#### 2.2.1.1. Installing the NVIDIA 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
```

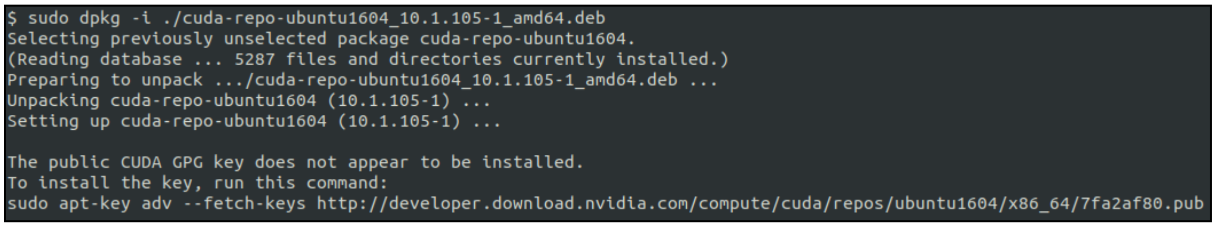


```
$ curl -O https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2832  100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

**Figure 2.2. CUDA Repo Download**

2. Install the deb package.

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

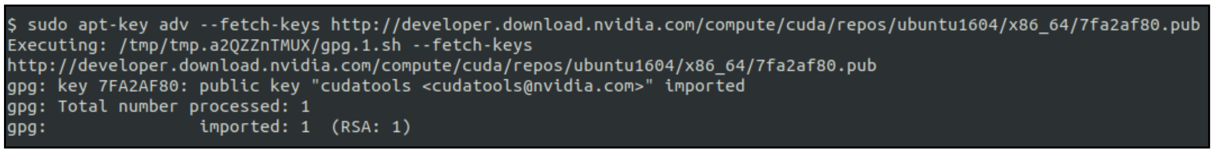


```
$ sudo dpkg -i ./cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
Selecting previously unselected package cuda-repo-ubuntu1604.
(Reading database ... 5287 files and directories currently installed.)
Preparing to unpack .../cuda-repo-ubuntu1604_10.1.105-1_amd64.deb ...
Unpacking cuda-repo-ubuntu1604 (10.1.105-1)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

3. Proceed with the installation.

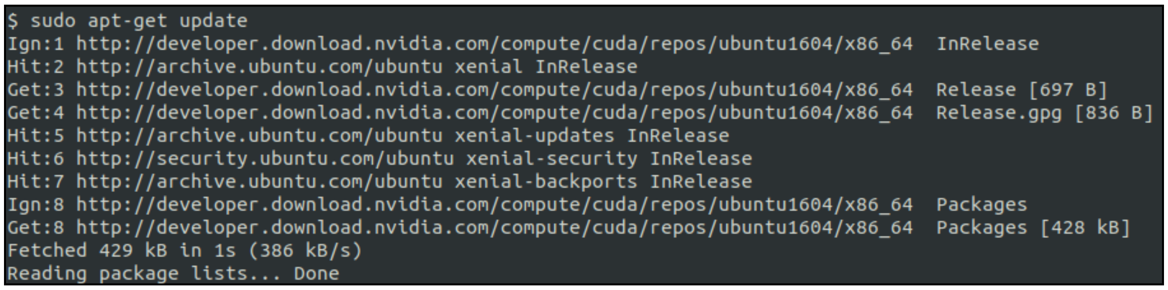
```
$ sudo apt-key adv --fetch-keys
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa
2af80.pub
```



```
$ 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**

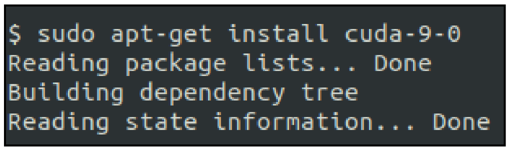
```
$sudo apt-get update
```



```
$ 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. Updated 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
```

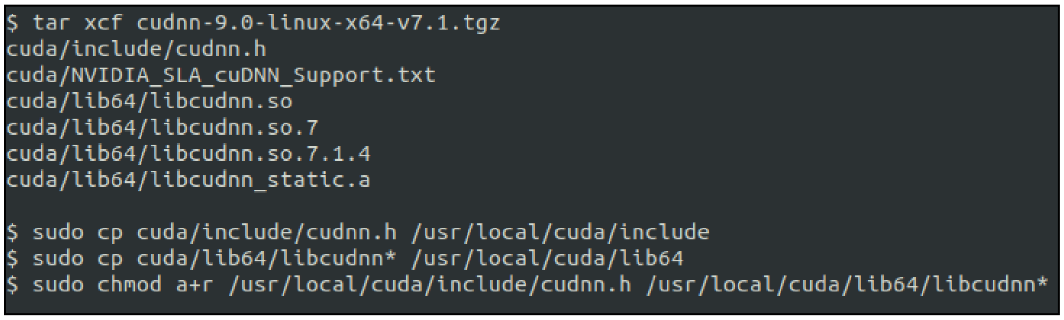
**Figure 2.6. CUDA Installation Completed**

#### 2.2.1.2. Installing the cuDNN

To install the cuDNN:

1. Create your Nvidia developer account in https://developer.nvidia.com.
2. Download cuDNN library from 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 xvfcudnn-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
$ sudochmoda+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**

### 2.2.2. Setting Up the Environment for Training and Model Freezing Scripts

This section describes the environment setup information for training and model freezing scripts for 64-bit Ubuntu 16.04. Anaconda provides one of the easiest ways to perform machine learning development and training on Linux.

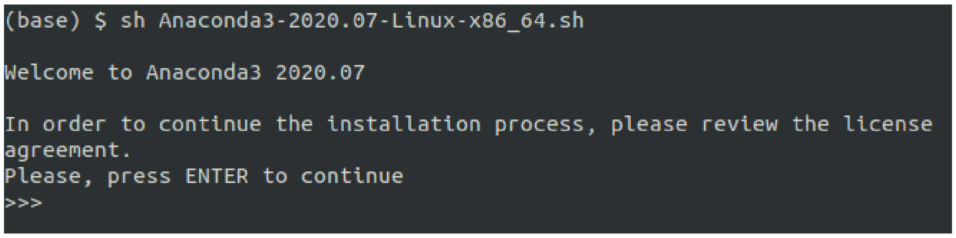
#### 2.2.2.1. Installing the Anaconda Python

To install the Anaconda Python 3:

1. Go to https://www.anaconda.com/products/individual#download.
2. Download the Python3 version of Anaconda for Linux.
3. Run the command below to install the Anaconda environment:

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

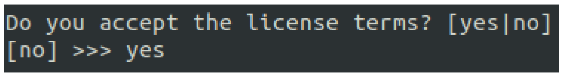
**Note:** Anaconda3-<version>-Linux-x86_64.sh, version may vary based on the release



```
(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**

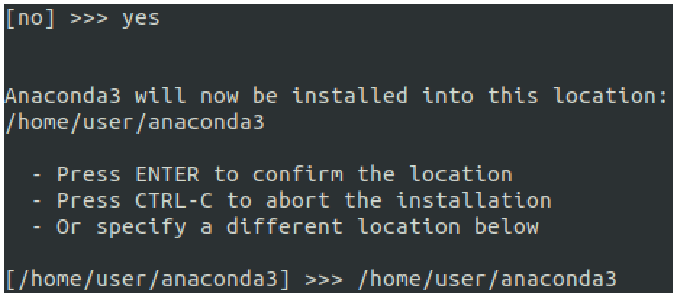
4. Accept the license.



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

**Figure 2.9. License Terms Acceptance**

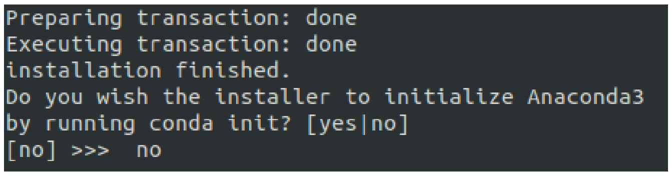
5. Confirm the installation path. Follow the instruction onscreen if you want to change the default path.



```
[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. 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
```

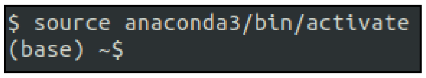
**Figure 2.11. Launch/Initialization of Anaconda Environment**

### 2.2.3. Installing TensorFlow version 1.14

To install TensorFlow version 1.14, run the commands below:

1. Activate the conda environment.

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

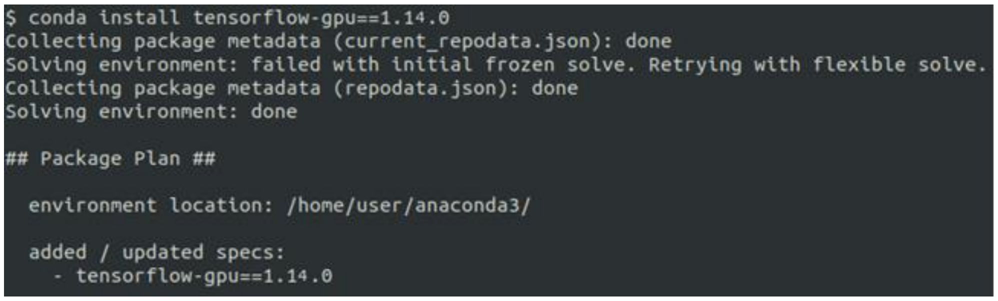


```
$ source anaconda3/bin/activate
(base) ~$
```

**Figure 2.12. Anaconda Environment Activation**

2. Install TensorFlow.

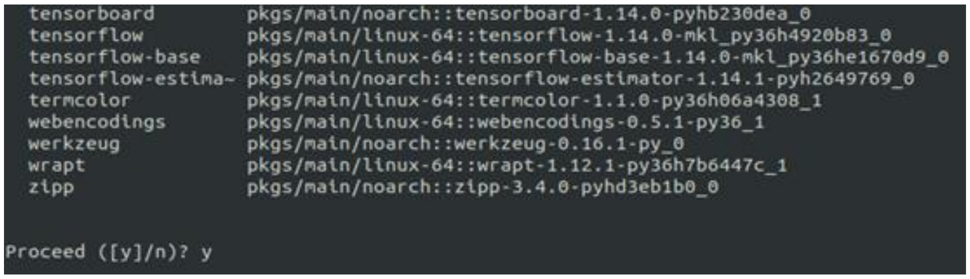
```
$ conda install tensorflow-gpu==1.14.0
```



```
$ conda install tensorflow-gpu==1.14.0
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/user/anaconda3/

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

**Figure 2.13. TensorFlow Installation**

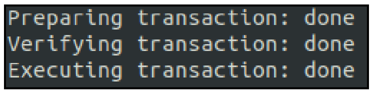
3. After installation, enter **Y** as shown in Figure 2.14.



```
  tensorboard          pkgs/main/noarch::tensorboard-1.14.0-pyhb230dea_0
  tensorflow           pkgs/main/linux-64::tensorflow-1.14.0-mkl_py36h4920b83_0
  tensorflow-base      pkgs/main/linux-64::tensorflow-base-1.14.0-mkl_py36he1670d9_0
  tensorflow-estima~   pkgs/main/noarch::tensorflow-estimator-1.14.1-pyh2649769_0
  termcolor            pkgs/main/linux-64::termcolor-1.1.0-py36h06a4308_1
  webencodings         pkgs/main/linux-64::webencodings-0.5.1-py36_1
  werkzeug             pkgs/main/noarch::werkzeug-0.16.1-py_0
  wrapt                pkgs/main/linux-64::wrapt-1.12.1-py36h7b6447c_1
  zipp                 pkgs/main/noarch::zipp-3.4.0-pyhd3eb1b0_0


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

**Figure 2.14. TensorFlow Installation Confirmation**

Figure 2.15 shows that the TensorFlow installation is completed.



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

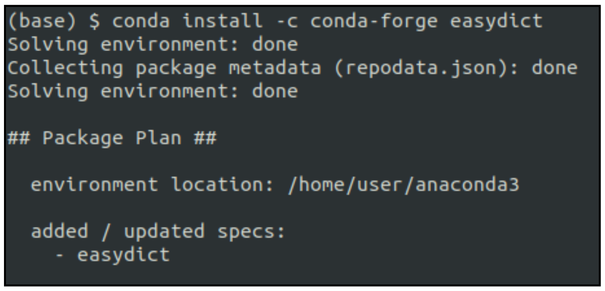
**Figure 2.15.TensorFlow Installation Completed**

### 2.2.4. Installing the Python Package

To install the Python package, run the commands below:

1. Install Easydict.

```
$ conda install –c conda-forge easydict
```

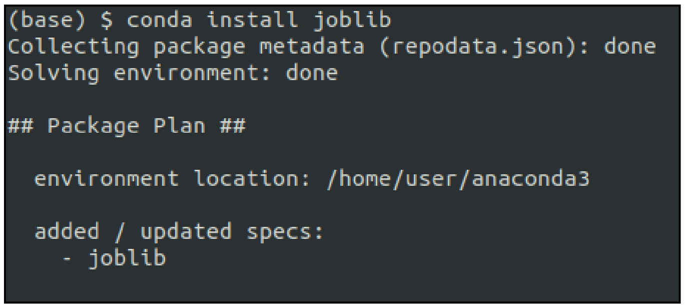


```
(base) $ conda install -c conda-forge easydict
Solving environment: done
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - easydict
```

**Figure 2.16. Easydict Installation**

2. Install Joblib.

```
$ conda install joblib
```

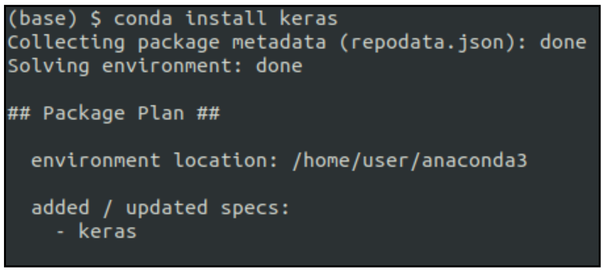


```
(base) $ conda install joblib
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - joblib
```

**Figure 2.17. Joblib Installation**

3. Install Keras.

```
$ conda install keras
```

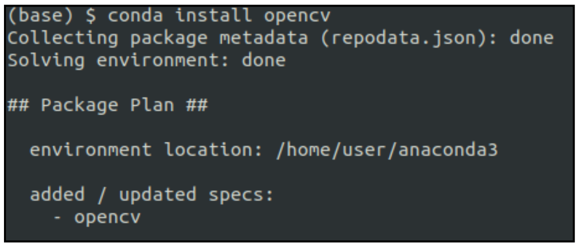


```
(base) $ conda install keras
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - keras
```

**Figure 2.18. Keras Installation**

4. Install OpenCV.

```
$ conda install opencv
```

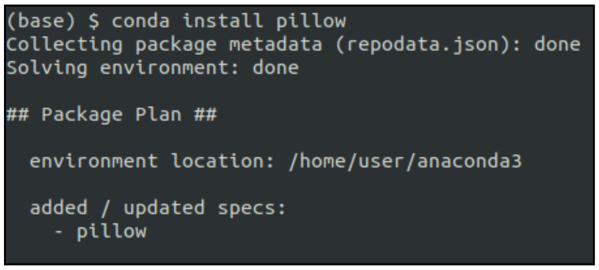


```
(base) $ conda install opencv
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/user/anaconda3

  added / updated specs:
    - opencv
```

**Figure 2.19. OpenCV Installation**

5. Install Pillow.

```
$ 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**

## 2.3. Creating the TensorFlow Lite Conversion Environment.

To create a new Anaconda environment and install TensorFlow version 2.2.0, run the commands below:

1. Create a new Anaconda environment.

```
$ conda create -n <New Environment Name> python=3.6
```

2. Activate the new Anaconda environment.

```
$ conda activate <New Environment Name>
```

3. Install TensorFlow version 2.2.0.

```
$ conda install tensorflow=2.2.0
```

**Note:** Output difference between TensorFlow (2.2.0) and TensorFlow-GPU (2.2.0) in terms of TensorFlow Lite size is observed. It is therefore recommended that TensorFlow (2.2.0) is used.

4. Install opencv.

```
$conda install opencv
```

# 3. Preparing the Dataset

This chapter describes how to create a dataset using examples from Google Open Image Dataset.

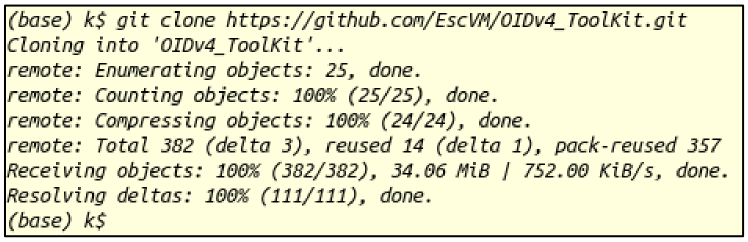
The Google Open Image Dataset version 4 (https://storage.googleapis.com/openimages/web/index.html) features more than 600 classes of images. The Person class of images include human annotated and machine annotated labels and bounding box. Annotations are licensed by Google Inc. under CC BY 4.0 and images are licensed under CC BY 2.0.

## 3.1. Downloading the Dataset

To download the dataset:

1. Clone the OIDv4_Toolkit repository by running the command below.

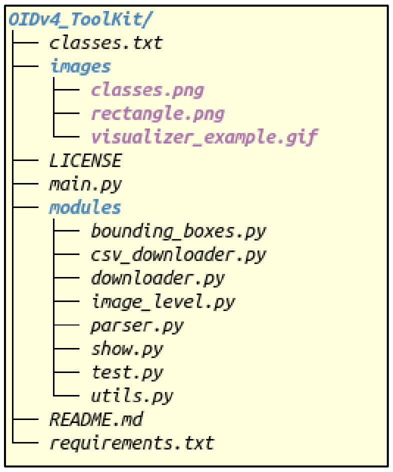
```
$ git clone https://github.com/EscVM/OIDv4_ToolKit.git
$ cd OIDv4_ToolKit
```



```
(base) k$ git clone https://github.com/EscVM/OIDv4_ToolKit.git
Cloning into 'OIDv4_ToolKit'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 382 (delta 3), reused 14 (delta 1), pack-reused 357
Receiving objects: 100% (382/382), 34.06 MiB | 752.00 KiB/s, done.
Resolving deltas: 100% (111/111), done.
(base) k$
```

**Figure 3.1. Open Source Dataset Repository Cloning**

Figure 3.2 shows the OIDv4 directory structure.



```
OIDv4_ToolKit/
├── classes.txt
├── images
│   ├── classes.png
│   ├── rectangle.png
│   └── visualizer_example.gif
├── LICENSE
├── main.py
├── modules
│   ├── bounding_boxes.py
│   ├── csv_downloader.py
│   ├── downloader.py
│   ├── image_level.py
│   ├── parser.py
│   ├── show.py
│   ├── test.py
│   └── utils.py
├── README.md
└── requirements.txt
```

**Figure 3.2. OIDv4_Toolkit Directory Structure**

View the OIDv4 Toolkit Help menu by running the command below.

```
$ python3 main.py -h
```

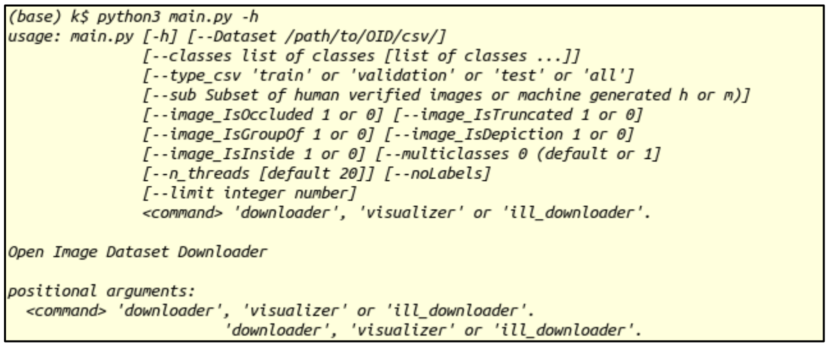

```
(base) k$ python3 main.py -h
usage: main.py [-h] [--Dataset /path/to/OID/csv/]
               [--classes list of classes [list of classes ...]]
               [--type_csv 'train' or 'validation' or 'test' or 'all']
               [--sub Subset of human verified images or machine generated h or m)]
               [--image_IsOccluded 1 or 0] [--image_IsTruncated 1 or 0]
               [--image_IsGroupOf 1 or 0] [--image_IsDepiction 1 or 0]
               [--image_IsInside 1 or 0] [--multiclasses 0 (default or 1]
               [--n_threads [default 20]] [--noLabels]
               [--limit integer number]
               <command> 'downloader', 'visualizer' or 'ill_downloader'.

Open Image Dataset Downloader

positional arguments:
  <command> 'downloader', 'visualizer' or 'ill_downloader'.
                        'downloader', 'visualizer' or 'ill_downloader'.
```

**Figure 3.3. Dataset Script Option/Help**

2. Use the OIDv4 Toolkit to download dataset. Download Person class images by running the command below.

```
$ python3 main.py downloader --classes Person --type_csv validation
```

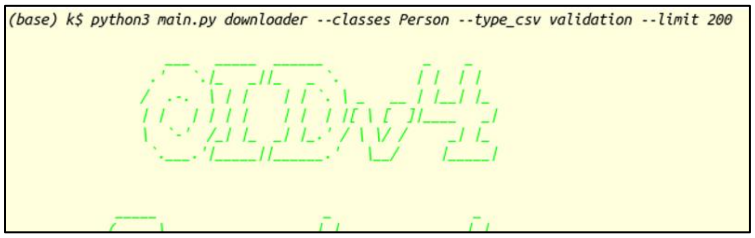

```
(base) k$ python3 main.py downloader --classes Person --type_csv validation --limit 200
```

**Figure 3.4. Dataset Downloading Logs**

Figure 3.5 shows the downloaded dataset directory structure.

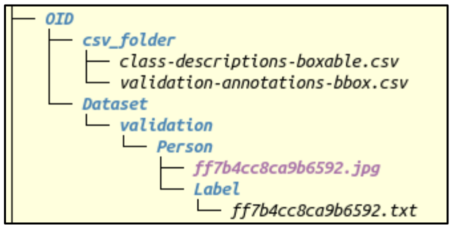

```
├── OID
    ├── csv_folder
    │   ├── class-descriptions-boxable.csv
    │   └── validation-annotations-bbox.csv
    └── Dataset
        └── validation
            └── Person
                ├── ff7b4cc8ca9b6592.jpg
                └── Label
                    └── ff7b4cc8ca9b6592.txt
```

**Figure 3.5. Downloaded Dataset Directory Structure**

3. Lattice training code uses KITTI (.txt) format. The downloaded dataset is not in the required KITTI format. Convert the annotation to KITTI format.

```
$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/validation/Person/Label/*
$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/train/Person/Label/*
$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/test/Person/Label/*
```

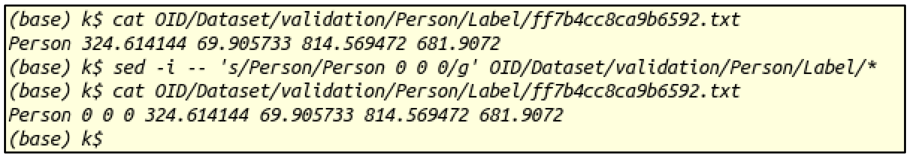

```
(base) k$ cat OID/Dataset/validation/Person/Label/ff7b4cc8ca9b6592.txt
Person 324.614144 69.905733 814.569472 681.9072
(base) k$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/validation/Person/Label/*
(base) k$ cat OID/Dataset/validation/Person/Label/ff7b4cc8ca9b6592.txt
Person 0 0 0 324.614144 69.905733 814.569472 681.9072
(base) k$
```

**Figure 3.6. OIDv4 Label to KITTI Format Conversion**

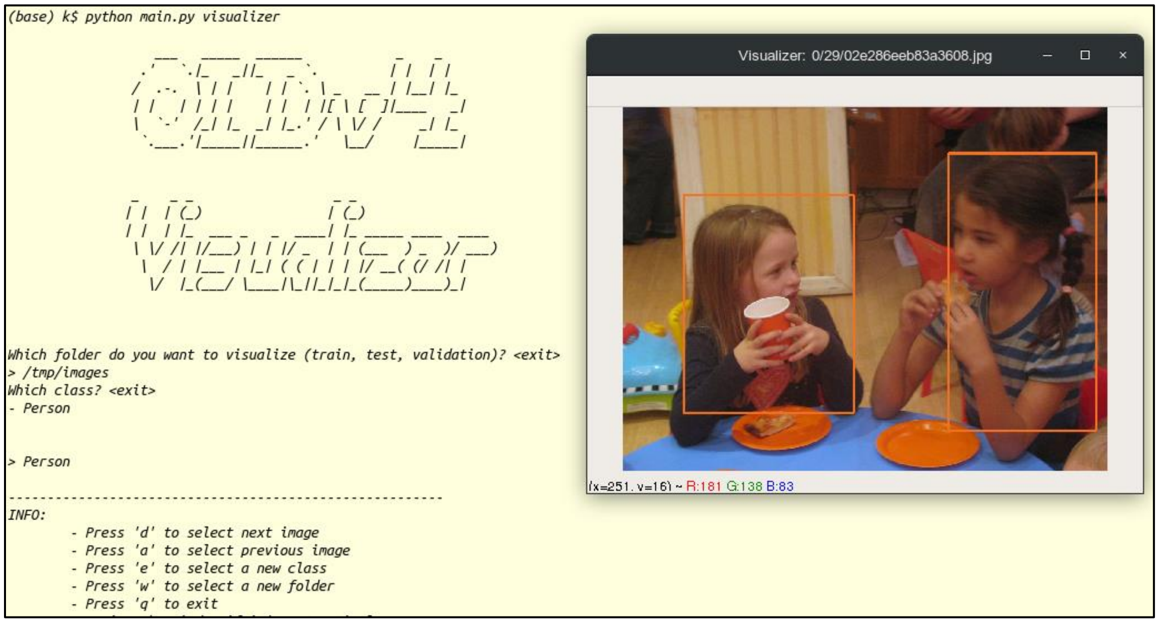
**Note:** KITTI Format: Person 0 0 0 324.61 69.90 814.56 681.90. It has class ID followed by truncated, occluded, alpha, Xmin, Ymin, Xmax, Ymax. The code converts Xmin, Ymin, Xmax, Ymax into x, y, w, h while training as *bounding box rectangle coordinates*.

## 3.2. Visualizing and Tuning/Cleaning Up the Dataset

To visualize and annotate the dataset, run the commands below:

1. Visualize the labelled images.

```
$ python3 main.py visualizer
```



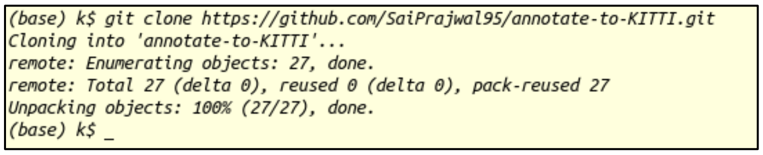
```
(base) k$ python main.py visualizer

              ___   _____  ______            _    _
            .'   `.|_   _||_   _ `.         | |  | |
           /  .-.  \ | |    | | `. \ _   __ | |__| |_
           | |   | | | |    | |  | |[ \ [  ]|____   _|
           \  `-'  /_| |_  _| |_.' / \ \/ /     _| |_
            `.___.'|_____||______.'   \__/     |_____|


           _    _ _               _ _
          | |  | (_)             | (_)
          | |  | |_ ___ _   _  __ _| |_ _______ _ __
           \ \/ /| / __| | | |/ _` | | |_  / _ \ '__|
            \  / | \__ \ |_| | (_| | | |/ /  __/ |
             \/  |_|___/\__,_|\__,_|_|_/___\___|_|


Which folder do you want to visualize (train, test, validation)? <exit>
> /tmp/images
Which class? <exit>
- Person


> Person

--------------------------------------------------------
INFO:
        - Press 'd' to select next image
        - Press 'a' to select previous image
        - Press 'e' to select a new class
        - Press 'w' to select a new folder
        - Press 'q' to exit
```

Visualizer: 0/29/02e286eeb83a3608.jpg
(x=251, y=16) ~ R:181 G:138 B:83

**Figure 3.7. Toolkit Visualizer**

2. Clone the manual annotation tool from the GitHub repository.

```
$ git clone https://github.com/SaiPrajwal95/annotate-to-KITTI.git
```

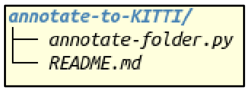


```
(base) k$ git clone https://github.com/SaiPrajwal95/annotate-to-KITTI.git
Cloning into 'annotate-to-KITTI'...
remote: Enumerating objects: 27, done.
remote: Total 27 (delta 0), reused 0 (delta 0), pack-reused 27
Unpacking objects: 100% (27/27), done.
(base) k$ _
```

**Figure 3.8. Manual Annotation Tool – Cloning**

3. Go to the *annotate-to-KITTI* directory.

```
$ cd annotate-to-KITTI
$ ls
```



```
annotate-to-KITTI/
├── annotate-folder.py
└── README.md
```

**Figure 3.9. Manual Annotation Tool – Directory Structure**

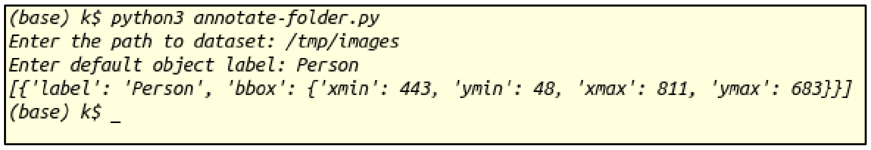
4. Install the dependencies (OpenCV 2.4).

```
$ sudo apt-get install python-opencv
```

5. Launch the utility.

```
$ python3 annotate-folder.py
```

6. Set the dataset path and default object label.



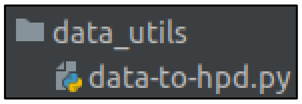
```
(base) k$ python3 annotate-folder.py
Enter the path to dataset: /tmp/images
Enter default object label: Person
[{'label': 'Person', 'bbox': {'xmin': 443, 'ymin': 48, 'xmax': 811, 'ymax': 683}}]
(base) k$ _
```

**Figure 3.10. Manual Annotation Tool – Launch**

7. For annotation, run the script provided in the website below.
   https://github.com/SaiPrajwal95/annotate-to-KITTI

## 3.3. Data Augmentation

Deep networks need a large amount of training data to achieve good performance. To train a neural network using minimal training data, image augmentation is usually required to boost the performance. Image augmentation creates training images through different processes such as random rotation, shifts, shears and flips, and others. Combinations of multiple processes may also be used.



```
data_utils
    data-to-hpd.py
```

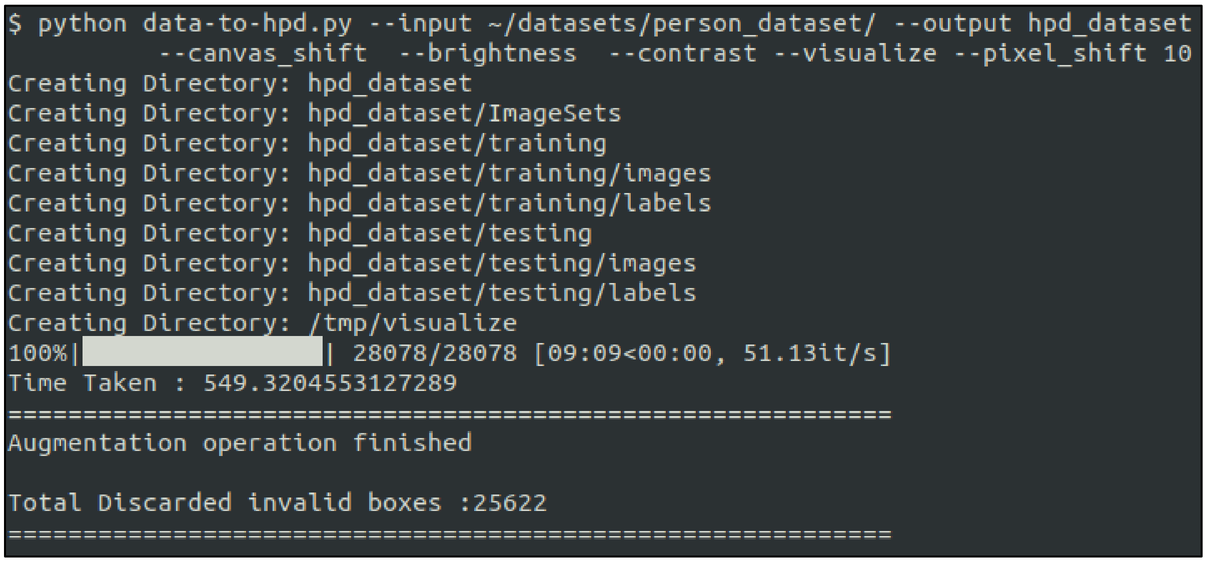
**Figure 3.11. Augmentation Directory Structure**

- Data_to_qvga.py: This file is the augmentation operations script.

### 3.3.1. Running the Augmentation

Run the augmentation using the following command:

```
$ python data_to_hpd.py  --input <input_dataset_path>  --output <output_dataset_path> --canvas_shift  --brightness  --contrast
```



```
$ python data-to-hpd.py --input ~/datasets/person_dataset/ --output hpd_dataset
          --canvas_shift  --brightness  --contrast --visualize --pixel_shift 10
Creating Directory: hpd_dataset
Creating Directory: hpd_dataset/ImageSets
Creating Directory: hpd_dataset/training
Creating Directory: hpd_dataset/training/images
Creating Directory: hpd_dataset/training/labels
Creating Directory: hpd_dataset/testing
Creating Directory: hpd_dataset/testing/images
Creating Directory: hpd_dataset/testing/labels
Creating Directory: /tmp/visualize
100%|████████████████| 28078/28078 [09:09<00:00, 51.13it/s]
Time Taken : 549.3204553127289
============================================================
Augmentation operation finished

Total Discarded invalid boxes :25622
============================================================
```

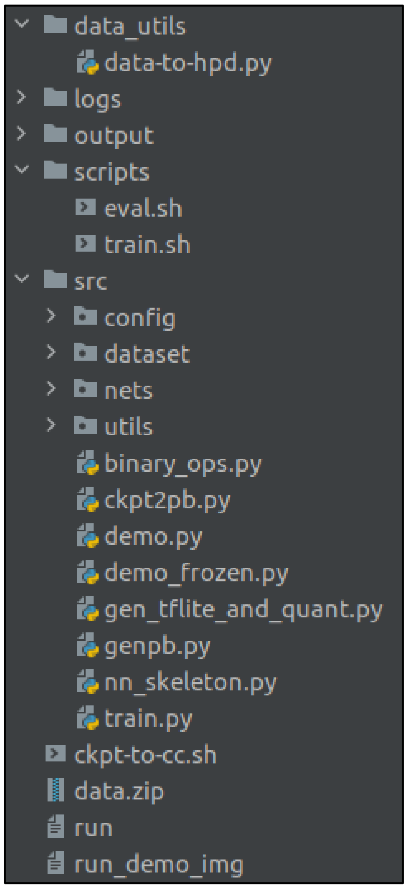
**Figure 3.12. Running the Augmentation**

The *data_to_hpd,py* file contains additional optional flags as described below.

- --input – Input Dataset Path
- --output – Output Dataset Path
- --canvas_shift – Flag to add Canvas shifting augmentation
- --brightness – Flag to add brightness augmentation
- --contrast – Flag to add contrast augmentation
- --pixel_shift – Number of pixel shift in canvas shift augmentation
- --type – (kitti/pascal) type of input dataset (default kitti)
- --output_dimension – Expected output dimension in form of x, y (Default 64, 64)
- --visualize – Flag that saves images in /tmp/visualize with drawn box (Optional)
- --canvas_shift_percentage – Percentage of dataset to apply canvas shift augmentation
- --brightness_percentage – Percentage of dataset to apply brightness augmentation
- --contrast_percentage – Percentage of dataset to apply contrast augmentation

# 4. Training the Machine

## 4.1. Training Code Directory Structure



```
data_utils
    data-to-hpd.py
logs
output
scripts
    eval.sh
    train.sh
src
    config
    dataset
    nets
    utils
    binary_ops.py
    ckpt2pb.py
    demo.py
    demo_frozen.py
    gen_tflite_and_quant.py
    genpb.py
    nn_skeleton.py
    train.py
ckpt-to-cc.sh
data.zip
run
run_demo_img
```

**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 Network Configuration of the Human Presence Detection design. The Neural Network model of the Human Presence Detection design uses VGG NN base model and the detection layer of SqueezeDet model.

**Table 4.1. Convolution Network Configuration of Human Presence Detection Design**

| Image Input (64 x 64 x 1) | | |
|---|---|---|
| Fire 1 | Conv3 - 16 | Conv3 - # where:<br>• *Conv3* – 3 × 3 Convolution filter Kernel size<br>• *#* - The number of filter<br>For example, Conv3 - 16 = 16 3 × 3 convolution filter<br><br>BN – Batch Normalization<br><br>FC - # where:<br>• *FC* – Fully connected layer<br>• *#* - The number of output |
| | BN | |
| | Relu | |
| | Maxpool | |
| Fire 2 | Conv3 - 16 | |
| | BN | |
| | Relu | |
| Fire 3 | Conv3 – 32 | |
| | BN | |
| | Relu | |
| | Maxpool | |
| Fire 4 | Conv3 – 32 | |
| | BN | |
| | Relu | |
| Fire 5 | Conv3 – 32 | |
| | BN | |
| | Relu | |
| | Maxpool | |
| Fire 6 | Conv3 – 44 | |
| | BN | |
| | Relu | |
| Fire 7 | Conv3 – 48 | |
| | BN | |
| | Relu | |
| | Maxpool | |
| Conv12 | Conv3 – 42 | |

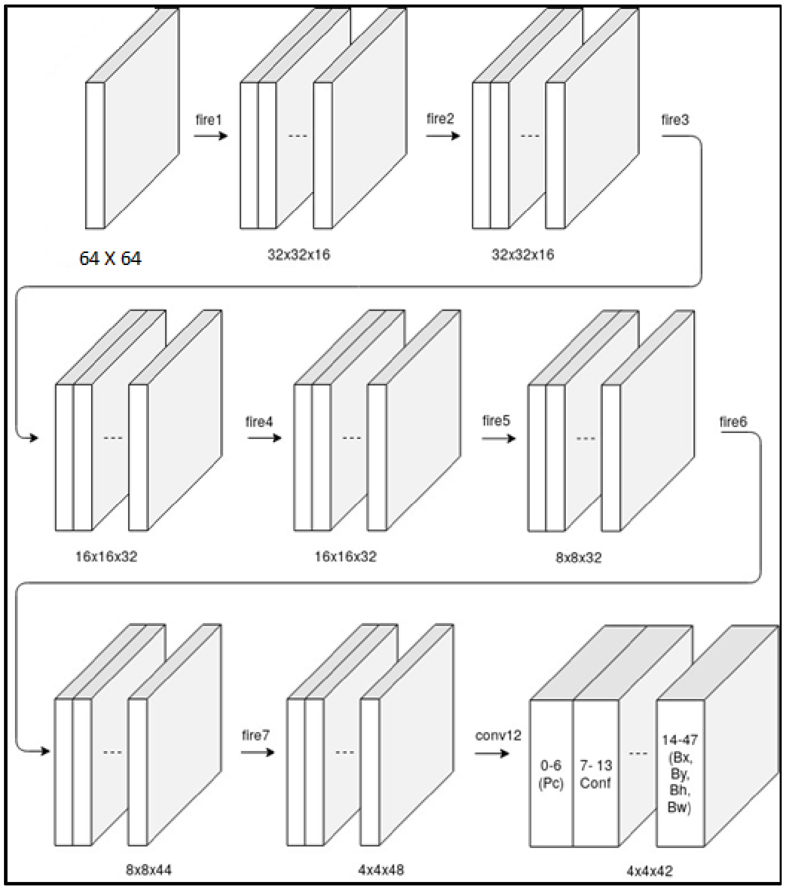


**Figure 4.2. Model Layer Dimensions**

- Human detection network structure consists of seven fire layers followed by one convolution layer. A fire layer contains convolution, batch normalization, and relu layers. Pooling layers are present only in fire 1, fire 3, fire 5, and fire 7. Layers fire 2, fire 4, and fire 6 do not contain pooling.
- Table 4.1 details the contents of the fire layers: convolution (conv), batch normalization (bn), and relu.
- Figure 4.2 shows the dimensions of each layer of the network.
- 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 (such as 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, 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 (such as 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 2 × 2) 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 will be 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 will control 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*
  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, and others.

  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.
  - Scale Layer 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 care free about weight initialization, works as regularization in place of dropout and other regularization techniques.

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

### 4.2.2. Human Presence Detection Network Output

From the input image model first extracts feature maps, overlays them with a W × H grid and at each cell computes K pre-computed bounding boxes called anchors. Each bounding box has the following:

- Four scalars (x, y, w, h)
  - A confidence score ( Pr(Object) ×IOU )
  - C conditional class probability
- The current model architecture has a fixed output of *W×H×K(4+1+C)*. Where,
  - W, H = Grid Size
  - K = Number of Anchor boxes
  - C = Number of classes for which we want detection
- The model has total 672 output values. It is derived from following:
  - 4 × 4 grid with 7 anchor boxes per grid an 6 values per anchor box. It consists of:
    - 4 bounding box coordinates (x, y, w, h)
    - 1 class probability
    - 1 confidence score

In total, 4 × 4 × 7 × 6 = 672 output values.

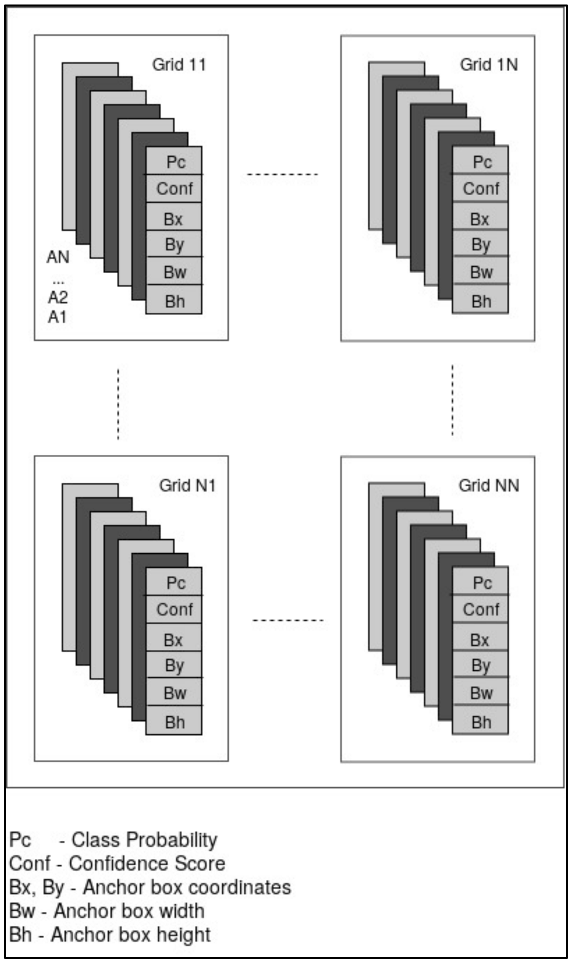


**Figure 4.3. Model Output Format**

#### 4.2.2.1. Model Output Format on Hardware

- On hardware, human presence detection demo works based on confidence score. So, other output values like class probability and bbox coordinates are byproduct which are not used at all.
- If the last layer in the network is Convolution, CNN IP supports partial output processing based on given filter range. The sensAI tool provides option to specify the required filter range from the convolution layer output. This will also result in hardware performance improvement.
- In Human Presence demo, last convolution layer has 42 filters as described in the Neural Network Architecture section. Out of 42, first seven filters provides class probability values, the next seven are for confidence score, and rest for bbox coordinates.
- By configuring *output depth range* as shown in Figure 4.4, CNN only provides 112 confidence values as output.

### 4.2.3. Training Code Overview

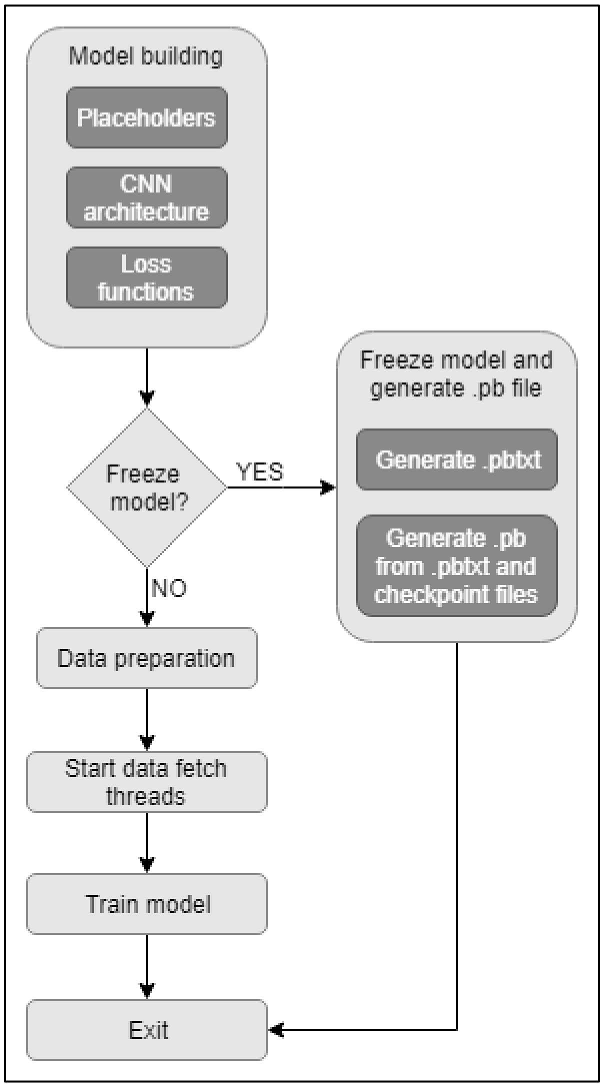


**Figure 4.4. Training Code Flow Diagram**

Training code can be divided into below parts:

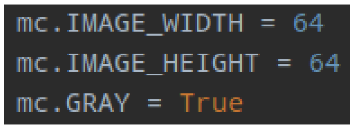
- Model config
- Model building
- Model freezing
- Data preparation
- Training for overall execution flow.

Details of each can be found in subsequent sections.

#### 4.2.3.1. Model Configuration

Demo uses Kitti dataset and SqueezeDet model. kitti_squeezeDet_config.py maintains all the configurable parameters for the model. Below is summary of configurable parameters:

- Image size
  - Change mc.IMAGE_WIDTH and mc.IMAGE_HEIGHT to configure Image size (width and height) in *src/config/kitti_squeezeDet_config.py*

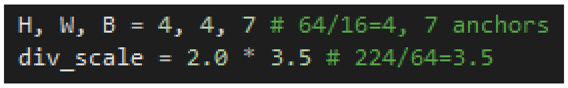


```
mc.IMAGE_WIDTH = 64
mc.IMAGE_HEIGHT = 64
mc.GRAY = True
```

**Figure 4.5. Code Snippet: Input Image Size Configuration**

  - Since there are four pooling layers, grid dimension would be H = mc.IMAGE_WIDTH/(2 ^ 4) and W = mc.IMAGE_HEIGHT/(2 ^ 4). Update grid size anchors per grid in set_anchors() in src/config/kitti_squeezeDet_config.py..

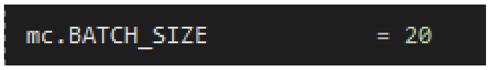
    For example,. if Image size is 128 x 128, H = 128 / 16 = 8 and W = H = 128 / 16 = 8



```
H, W, B = 4, 4, 7 # 64/16=4, 7 anchors
div_scale = 2.0 * 3.5 # 224/64=3.5
```

**Figure 4.6. Code Snippet: Input Image Size Configuration (Grid Sizes)**

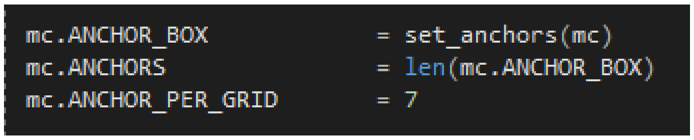
- Batch size
  - Change mc.BATCH_SIZE in src/config/kitti_squeezeDet_config.py to configure batch size.



```
mc.BATCH_SIZE              = 20
```

**Figure 4.7. Code Snippet: Batch Image Size Configuration**

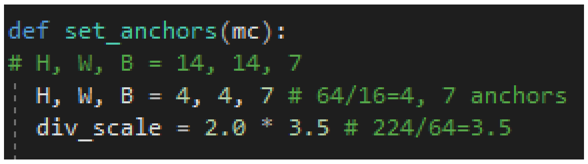
- Anchors per grid
  - Change mc.ANCHOR_PER_GRID in src/config/kitti_squeezeDet_config.py to configure anchors per grid.



```
mc.ANCHOR_BOX              = set_anchors(mc)
mc.ANCHORS                 = len(mc.ANCHOR_BOX)
mc.ANCHOR_PER_GRID         = 7
```

**Figure 4.8. Code Snippet: Anchors Per Grid Configuration #1**

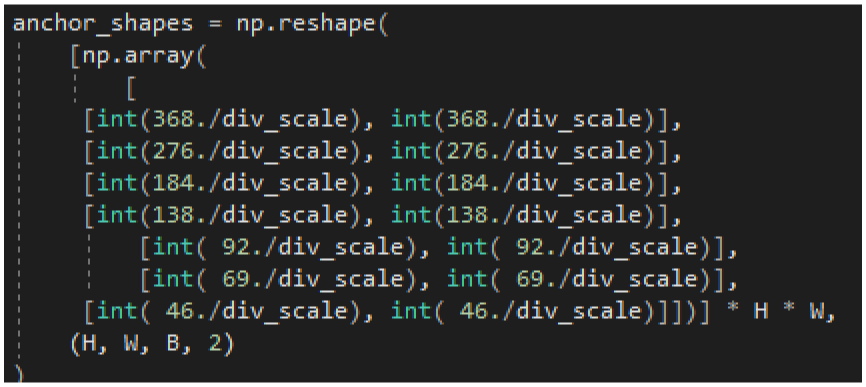
  - Change hard coded anchors per grid in set_anchors() in src/config/kitti_squeezeDet_config.py. Here B (value 7) indicates anchors per grid.



```
def set_anchors(mc):
# H, W, B = 14, 14, 7
  H, W, B = 4, 4, 7 # 64/16=4, 7 anchors
  div_scale = 2.0 * 3.5 # 224/64=3.5
```

**Figure 4.9. Code Snippet: Anchors Per Grid Configuration #2**

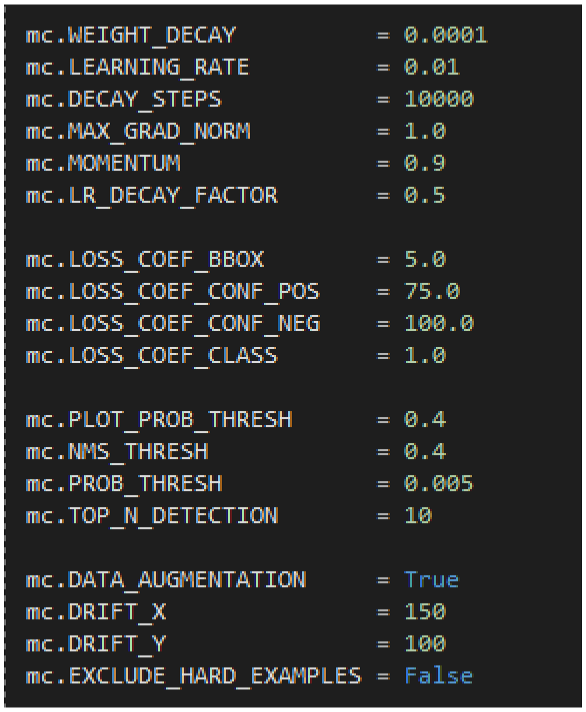
  - anchor_shapes variable of set_anchors() in *src/config/kitti_squeezeDet_config.py* indicates anchors width and heights. Update it based on anchors per grid size changes.



```
anchor_shapes = np.reshape(
    [np.array(
        [
     [int(368./div_scale), int(368./div_scale)],
     [int(276./div_scale), int(276./div_scale)],
     [int(184./div_scale), int(184./div_scale)],
     [int(138./div_scale), int(138./div_scale)],
         [int( 92./div_scale), int( 92./div_scale)],
         [int( 69./div_scale), int( 69./div_scale)],
     [int( 46./div_scale), int( 46./div_scale)]])] * H * W,
    (H, W, B, 2)
)
```

**Figure 4.10. Code Snippet: Anchors Per Grid Configuration #3**

- Training parameters
    - Other training related parameters like learning rate, loss parameters and different thresholds can be configured from src/config/kitti_squeezeDet_config.py.



```
mc.WEIGHT_DECAY          = 0.0001
mc.LEARNING_RATE         = 0.01
mc.DECAY_STEPS           = 10000
mc.MAX_GRAD_NORM         = 1.0
mc.MOMENTUM              = 0.9
mc.LR_DECAY_FACTOR       = 0.5

mc.LOSS_COEF_BBOX        = 5.0
mc.LOSS_COEF_CONF_POS    = 75.0
mc.LOSS_COEF_CONF_NEG    = 100.0
mc.LOSS_COEF_CLASS       = 1.0

mc.PLOT_PROB_THRESH      = 0.4
mc.NMS_THRESH            = 0.4
mc.PROB_THRESH           = 0.005
mc.TOP_N_DETECTION       = 10

mc.DATA_AUGMENTATION     = True
mc.DRIFT_X               = 150
mc.DRIFT_Y               = 100
mc.EXCLUDE_HARD_EXAMPLES = False
```

**Figure 4.11. Code Snippet: Training Parameters**

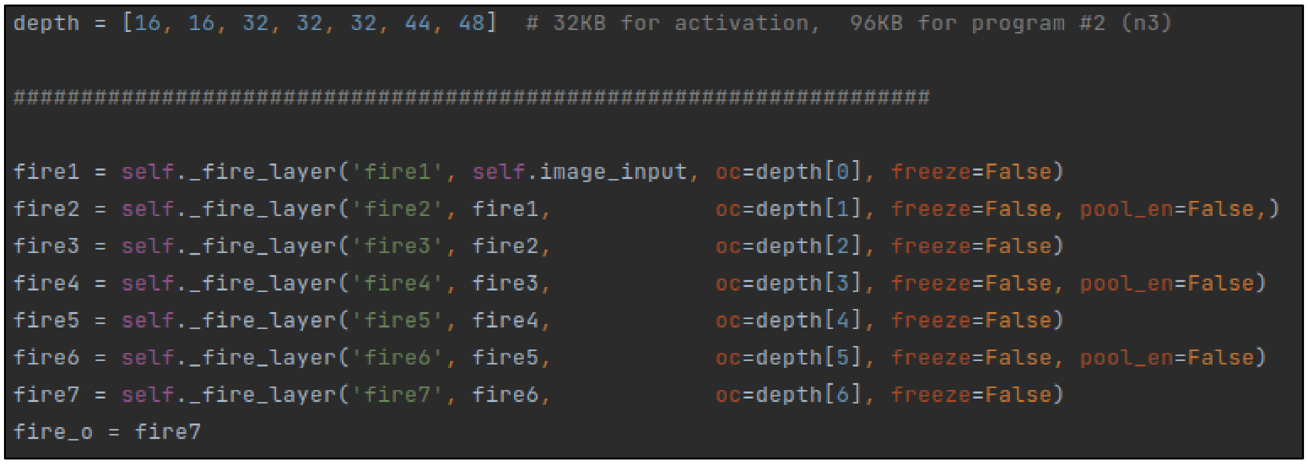
#### 4.2.3.2. Model Building

SqueezeDet class constructor builds model which can be divided in below sections:

- Forward graph
- Interpretation graph
- Loss graph
- Train graph
- Visualization graph

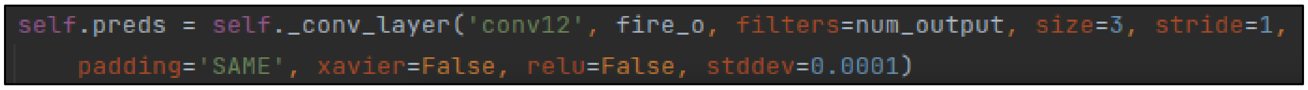
**Forward graph**

- Forward graph consists of seven fire layers.
    - Each fire layers contains a 3 × 3 convolution layer with padding='SAME' and stride=1, a batch normalization layer, ReLU layer and an optional max pool layer. Out of these 3 fire layers, fire 2, fire 4 and fire 6 layers do not use max pool.
- These seven fire layer is followed by a 3 × 3 convolution layer with padding='SAME' and stride=1.
- Filter sizes of each convolutional blocks is mentioned in Table 4.1, which can be configured by changing values of *depth* shown in Figure 4.12.



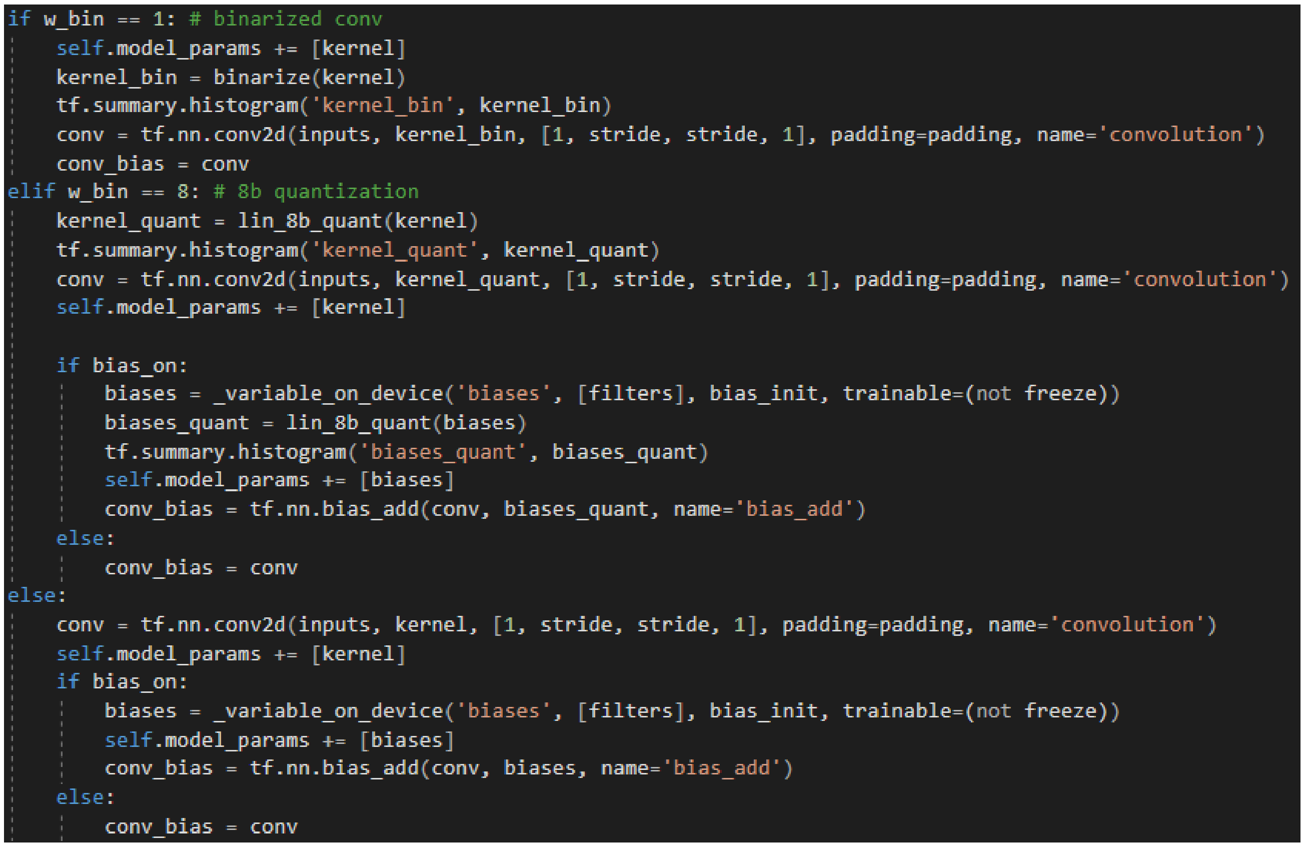
```
depth = [16, 16, 32, 32, 32, 44, 48]  # 32KB for activation,  96KB for program #2 (n3)

####################################################################

fire1 = self._fire_layer('fire1', self.image_input, oc=depth[0], freeze=False)
fire2 = self._fire_layer('fire2', fire1,            oc=depth[1], freeze=False, pool_en=False,)
fire3 = self._fire_layer('fire3', fire2,            oc=depth[2], freeze=False)
fire4 = self._fire_layer('fire4', fire3,            oc=depth[3], freeze=False, pool_en=False)
fire5 = self._fire_layer('fire5', fire4,            oc=depth[4], freeze=False)
fire6 = self._fire_layer('fire6', fire5,            oc=depth[5], freeze=False, pool_en=False)
fire7 = self._fire_layer('fire7', fire6,            oc=depth[6], freeze=False)
fire_o = fire7
```

**Figure 4.12. Code Snippet: Forward Graph Fire Layers**



```
self.preds = self._conv_layer('conv12', fire_o, filters=num_output, size=3, stride=1,
    padding='SAME', xavier=False, relu=False, stddev=0.0001)
```

**Figure 4.13. Code Snippet: Forward Graph Last Convolution Layer**



```
if w_bin == 1: # binarized conv
    self.model_params += [kernel]
    kernel_bin = binarize(kernel)
    tf.summary.histogram('kernel_bin', kernel_bin)
    conv = tf.nn.conv2d(inputs, kernel_bin, [1, stride, stride, 1], padding=padding, name='convolution')
    conv_bias = conv
elif 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')
    self.model_params += [kernel]

    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)
        self.model_params += [biases]
        conv_bias = tf.nn.bias_add(conv, biases_quant, name='bias_add')
    else:
        conv_bias = conv
else:
    conv = tf.nn.conv2d(inputs, kernel, [1, stride, stride, 1], padding=padding, name='convolution')
    self.model_params += [kernel]
    if bias_on:
        biases = _variable_on_device('biases', [filters], bias_init, trainable=(not freeze))
        self.model_params += [biases]
        conv_bias = tf.nn.bias_add(conv, biases, name='bias_add')
    else:
        conv_bias = conv
```

**Figure 4.14. Code Snippet: Quantization Layer (Disabled)**

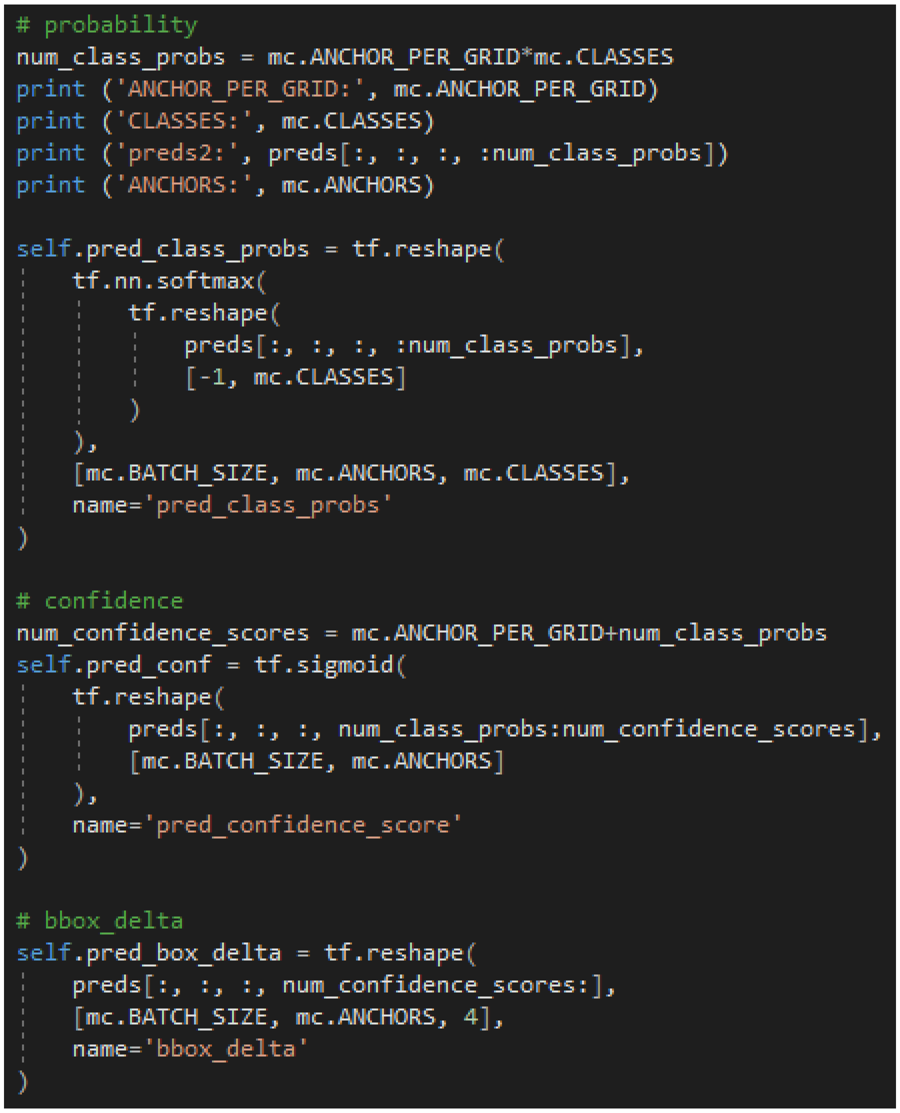
**Graph interpretation**

The Interpretation Graph consists of the following sub-blocks::

- interpret_output
  As mentioned in Figure 4.3, model output is 4 × 4 × 42. There are 42 channels in last layers which contain probability for the class, confidence score and bounding boxes values.

  This block interprets output from network and extracts predicted class probability, confidence score and bounding box values. From training code output value processing perspective, for each grid,
  - First N values (0:N-1) contains probabilities. Where N is number of anchor boxes. For N = 7, this ranges from 0 to 6 (including 6).
  - Next N values (N:2N – 1) contains confidence score. Where N is number of anchor boxes. For N = 7, this ranges from 7 to 13 (including 13).
  - Last 2N * 4 values contains bounding boxes information. Where N is number of anchor boxes. For N = 7, this ranges from 14 to 41 (including 41).

The code below shows how output from conv12 layer (4d array of batch size x 4 × 4 × 42) is sliced with proper indexes to get all values of probability, confidence and coordinates.



```
# probability
num_class_probs = mc.ANCHOR_PER_GRID*mc.CLASSES
print ('ANCHOR_PER_GRID:', mc.ANCHOR_PER_GRID)
print ('CLASSES:', mc.CLASSES)
print ('preds2:', preds[:, :, :, :num_class_probs])
print ('ANCHORS:', mc.ANCHORS)

self.pred_class_probs = tf.reshape(
    tf.nn.softmax(
        tf.reshape(
            preds[:, :, :, :num_class_probs],
            [-1, mc.CLASSES]
        )
    ),
    [mc.BATCH_SIZE, mc.ANCHORS, mc.CLASSES],
    name='pred_class_probs'
)

# confidence
num_confidence_scores = mc.ANCHOR_PER_GRID+num_class_probs
self.pred_conf = tf.sigmoid(
    tf.reshape(
        preds[:, :, :, num_class_probs:num_confidence_scores],
        [mc.BATCH_SIZE, mc.ANCHORS]
    ),
    name='pred_confidence_score'
)

# bbox_delta
self.pred_box_delta = tf.reshape(
    preds[:, :, :, num_confidence_scores:],
    [mc.BATCH_SIZE, mc.ANCHORS, 4],
    name='bbox_delta'
)
```

**Figure 4.15. Code Snippet: Interpret Output Graph**

For confidence score, value should be between 0 and 1, so sigmoid is used.

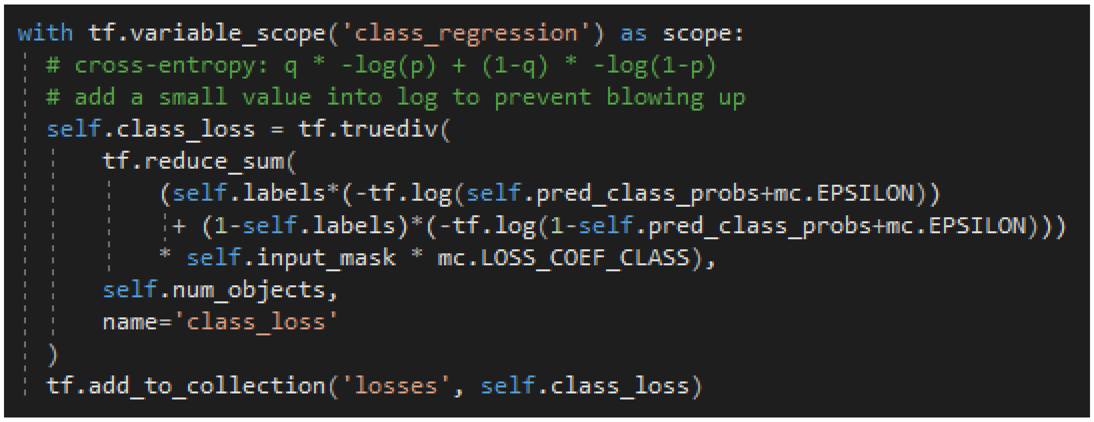
For predicting the class probabilities, there is a vector of NUM_CLASS values at each bounding box. Applying softmax makes it a better probability distribution.

- Bbox – This block calculates bounding boxes based on anchor box and predicated bounding boxes.
- IOU – This block calculates Intersection Over Union for detected bounding boxes and actual bounding boxes.
- Probability – This block calculates detection probability and object class.

## Loss graph

This block calculates different types of losses which need to be minimized. There are three types of losses which are considered for calculation:

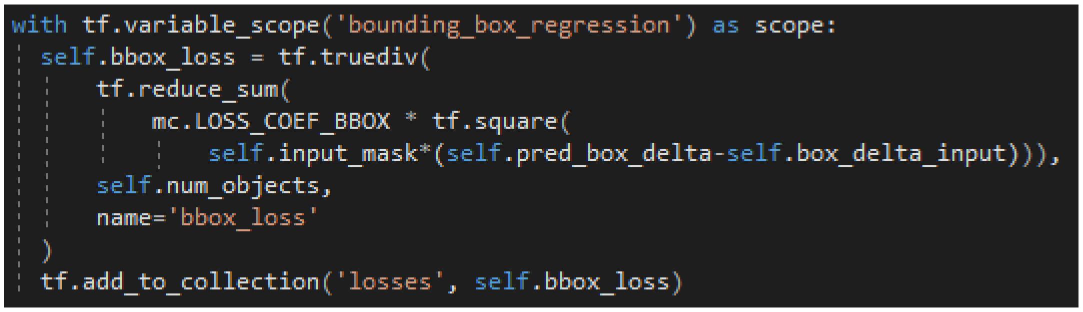
- Class probability
  The class loss function is just cross-entropy loss for classification for each box to do classification (predicted class vs. actual class), as we would for image classification.



```
with tf.variable_scope('class_regression') as scope:
  # cross-entropy: q * -log(p) + (1-q) * -log(1-p)
  # add a small value into log to prevent blowing up
  self.class_loss = tf.truediv(
      tf.reduce_sum(
          (self.labels*(-tf.log(self.pred_class_probs+mc.EPSILON))
          + (1-self.labels)*(-tf.log(1-self.pred_class_probs+mc.EPSILON)))
          * self.input_mask * mc.LOSS_COEF_CLASS),
      self.num_objects,
      name='class_loss'
  )
  tf.add_to_collection('losses', self.class_loss)
```

**Figure 4.16. Code Snippet: Class Loss**

- Bounding box
  This loss is regression of the scalars for the anchors



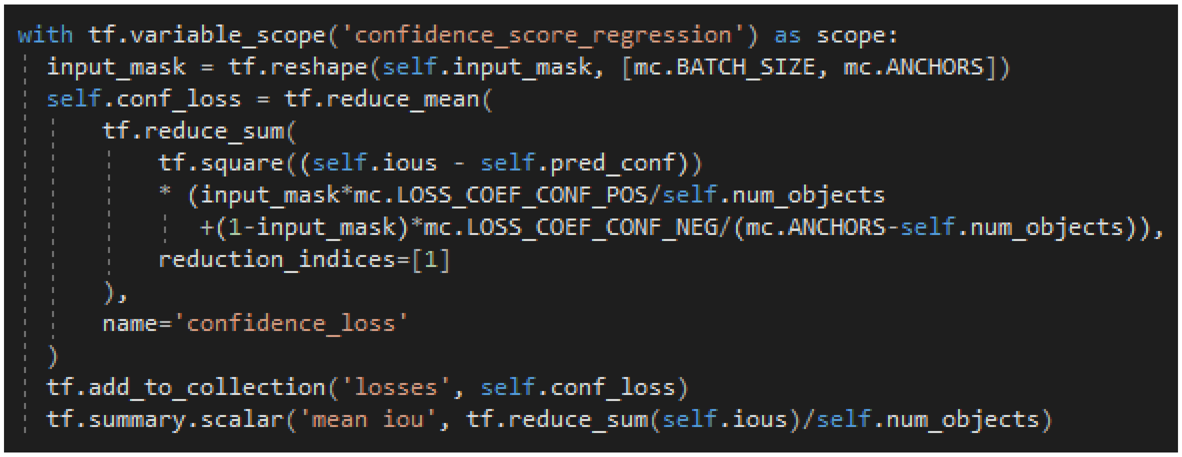
```
with tf.variable_scope('bounding_box_regression') as scope:
  self.bbox_loss = tf.truediv(
      tf.reduce_sum(
          mc.LOSS_COEF_BBOX * tf.square(
              self.input_mask*(self.pred_box_delta-self.box_delta_input))),
      self.num_objects,
      name='bbox_loss'
  )
  tf.add_to_collection('losses', self.bbox_loss)
```

**Figure 4.17. Code Snippet: Bbox Loss**

- Confidence score
  To obtain meaningful confidence score, each box's predicted value is regressed against the Intersection over Union of the real and the predicted box. During training, compare the ground truth bounding boxes with all anchors and assign them to the anchors that have the largest overlap (IOU) with each of them.

  The reason being is to select the *closest* anchor to match the ground truth box such that the transformation needed is reduced to minimum. Equation evaluates to 1 if the k-th anchor at position-(i, j) has the largest overlap with a ground truth box, and to 0 if no ground truth is assigned to it. This way, include only the loss generated by the *responsible* anchors.

  As there can be multiple objects per image, normalize the loss by dividing it by the number of objects (*self.num_objects*).



```
with tf.variable_scope('confidence_score_regression') as scope:
  input_mask = tf.reshape(self.input_mask, [mc.BATCH_SIZE, mc.ANCHORS])
  self.conf_loss = tf.reduce_mean(
      tf.reduce_sum(
          tf.square((self.ious - self.pred_conf))
          * (input_mask*mc.LOSS_COEF_CONF_POS/self.num_objects
             +(1-input_mask)*mc.LOSS_COEF_CONF_NEG/(mc.ANCHORS-self.num_objects)),
          reduction_indices=[1]
      ),
      name='confidence_loss'
  )
  tf.add_to_collection('losses', self.conf_loss)
  tf.summary.scalar('mean iou', tf.reduce_sum(self.ious)/self.num_objects)
```

**Figure 4.18. Code Snippet: Confidence loss**

**Train graph**

This block is responsible for training the model with Momentum optimizer to reduce all losses.

**Visualization graph**

This provides visualization of detected results.

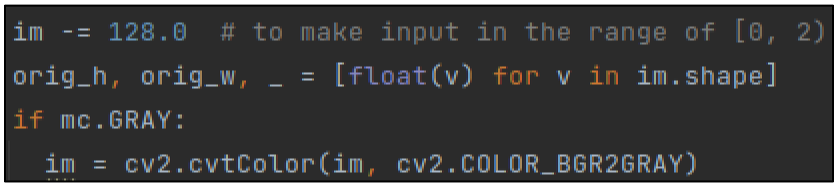
## 4.3. Training from Scratch and/or Transfer Learning

To train the machine:

1. Go to the top/root directory of the Lattice training code from command prompt.

   The Model works on 64 × 64 input resolution for training.

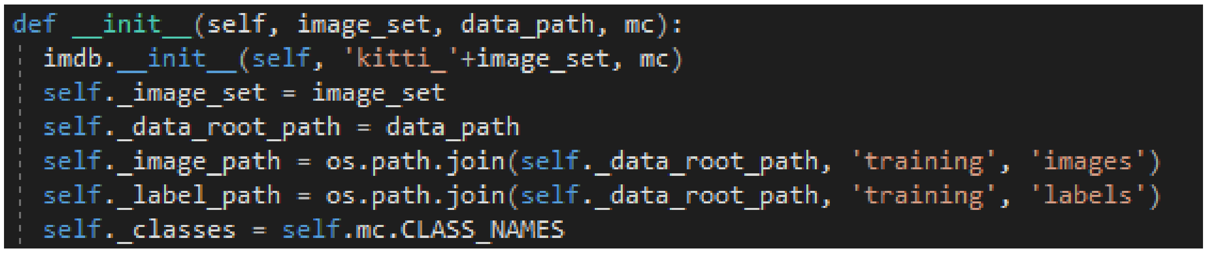
   Current human detection training code uses mean = -128 and scale = 1) in pre-processing step. Mean and scale can be changed in training code *@src/dataset/imdb.py* as shown in Figure 4.19.



```
im -= 128.0  # to make input in the range of [0, 2)
orig_h, orig_w, _ = [float(v) for v in im.shape]
if mc.GRAY:
  im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
```

**Figure 4.19. Training Code Snippet for Mean and Scale**

   The dataset path can be set in the training code *@src/dataset/kitti.py* and can be used in combination with the --*data_path* option while triggering training using *train.py* to get the desired path. For example, you can have *<data_path>/training/images* and *<data_path>/training/labels*.

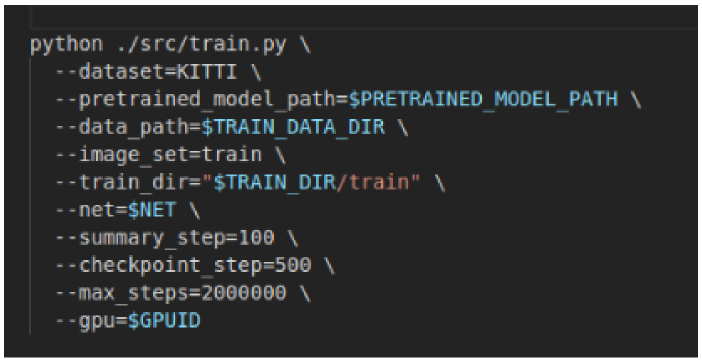


```
def __init__(self, image_set, data_path, mc):
  imdb.__init__(self, 'kitti_'+image_set, mc)
  self._image_set = image_set
  self._data_root_path = data_path
  self._image_path = os.path.join(self._data_root_path, 'training', 'images')
  self._label_path = os.path.join(self._data_root_path, 'training', 'labels')
  self._classes = self.mc.CLASS_NAMES
```

**Figure 4.20. Training Code Snippet for Dataset Path**

2. Modify the training script.

The training script at *@scripts/train.sh* is used to trigger training. Figure 4.21 shows the input parameters which can be configured.

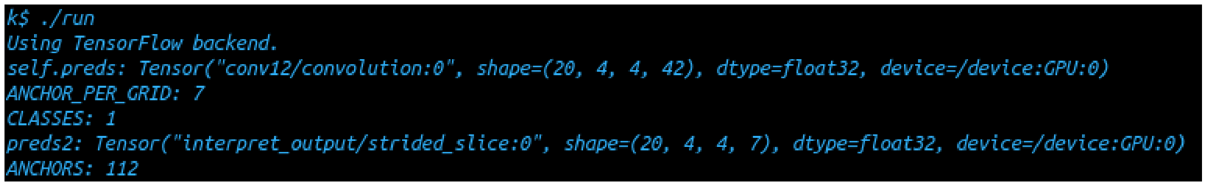


```
python ./src/train.py \
  --dataset=KITTI \
  --pretrained_model_path=$PRETRAINED_MODEL_PATH \
  --data_path=$TRAIN_DATA_DIR \
  --image_set=train \
  --train_dir="$TRAIN_DIR/train" \
  --net=$NET \
  --summary_step=100 \
  --checkpoint_step=500 \
  --max_steps=2000000 \
  --gpu=$GPUID
```

**Figure 4.21. Training Input Parameter**

- $TRAIN_DATA_DIR – dataset directory path. */data/humandet* is an example.
- $TRAIN_DIR – log directory where checkpoint files are generated while model is training.
- $GPUID – gpu id. If the system has more than one gpu, it indicates the one to use.
- --summary_step – indicates at which interval loss summary should be dumped.
- --checkpoint_step – indicates at which interval checkpoints will be created.
- --max_steps – indicates the maximum number of steps for which the model is trained.

3. Execute the *run* command script which starts training.



```
k$ ./run
Using TensorFlow backend.
self.preds: Tensor("conv12/convolution:0", shape=(20, 4, 4, 42), dtype=float32, device=/device:GPU:0)
ANCHOR_PER_GRID: 7
CLASSES: 1
preds2: Tensor("interpret_output/strided_slice:0", shape=(20, 4, 4, 7), dtype=float32, device=/device:GPU:0)
ANCHORS: 112
```

**Figure 4.22. Execute Run Script**

4. Start TensorBoard by running the command below.

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

   For example: tensorboard –logdir='./logs/'
5. Open the local host port on your web browser.
6. Check the training status on TensorBoard.

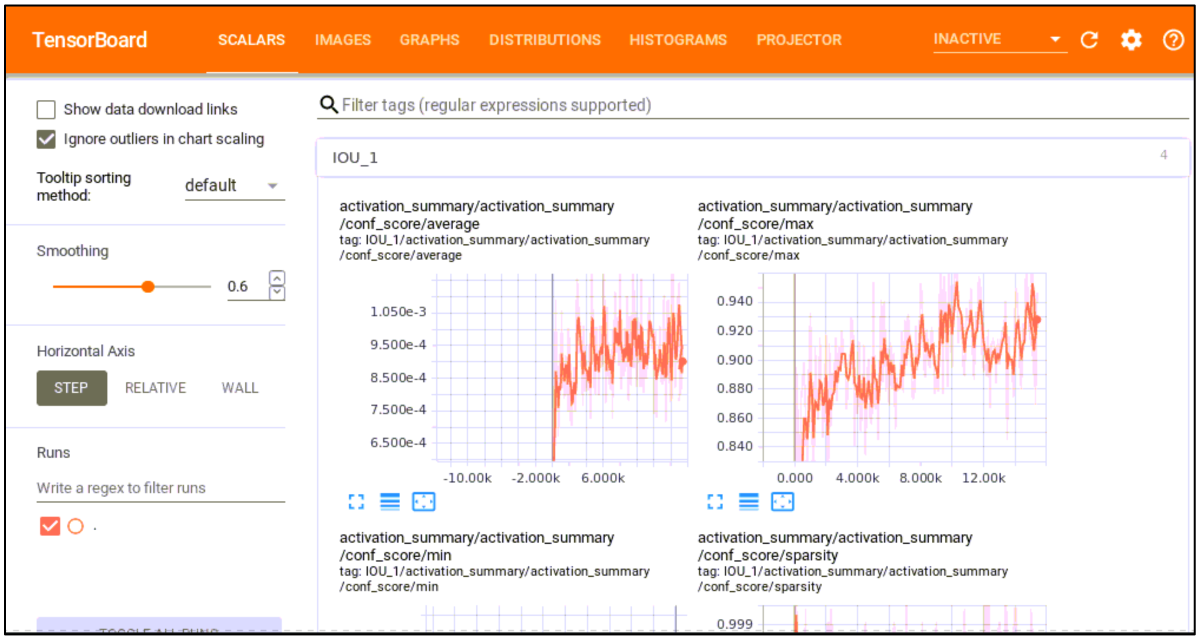
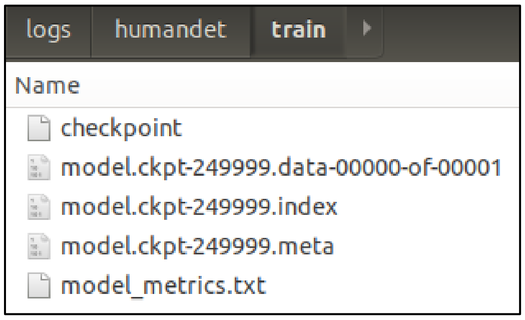

TensorBoard
SCALARS
IMAGES
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
PROJECTOR
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
Filter tags (regular expressions supported)
IOU_1
4
activation_summary/activation_summary/conf_score/average
tag: IOU_1/activation_summary/activation_summary/conf_score/average
activation_summary/activation_summary/conf_score/max
tag: IOU_1/activation_summary/activation_summary/conf_score/max
activation_summary/activation_summary/conf_score/min
tag: IOU_1/activation_summary/activation_summary/conf_score/min
activation_summary/activation_summary/conf_score/sparsity
tag: IOU_1/activation_summary/activation_summary/conf_score/sparsity

Figure 4.23. TensorBoard

7. 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).



logs humandet train

Name
checkpoint
model.ckpt-249999.data-00000-of-00001
model.ckpt-249999.index
model.ckpt-249999.meta
model_metrics.txt

Figure 4.24. Example of Checkpoint Data Files at Log Folder

# 5. Creating Frozen File and Generating C Array

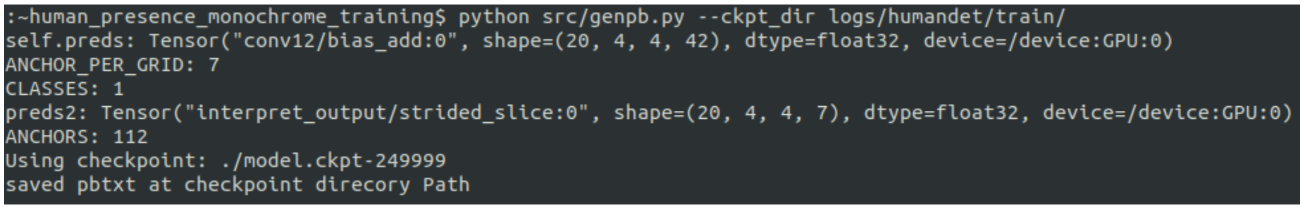
This section describes the procedure for freezing the model, which is aligned with the Lattice sensAI tool. Perform the steps below to generate the frozen protobuf file:

## 5.1. Generating the Frozen *.pbtxt* File

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

```
$ python src/genpb.py –ckpt_dir=”<log directory>” --freeze
```

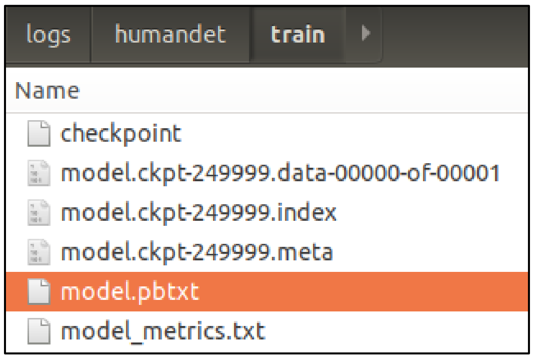
For example, *python src/genpb.py –ckpt_dir './logs/humandet/train' --freeze*



```
:~human_presence_monochrome_training$ python src/genpb.py --ckpt_dir logs/humandet/train/
self.preds: Tensor("conv12/bias_add:0", shape=(20, 4, 4, 42), dtype=float32, device=/device:GPU:0)
ANCHOR_PER_GRID: 7
CLASSES: 1
preds2: Tensor("interpret_output/strided_slice:0", shape=(20, 4, 4, 7), dtype=float32, device=/device:GPU:0)
ANCHORS: 112
Using checkpoint: ./model.ckpt-249999
saved pbtxt at checkpoint direcory Path
```

**Figure 5.1. pb File Generation from Checkpoint**

Figure 5.2 shows the generated .pbtxt file.



logs › humandet › train

Name

checkpoint
model.ckpt-249999.data-00000-of-00001
model.ckpt-249999.index
model.ckpt-249999.meta
model.pbtxt
model_metrics.txt

**Figure 5.2. Frozen .pbtxt File**

## 5.2. Generating the .pb, .tflite, and .cc Files from Checkpoints

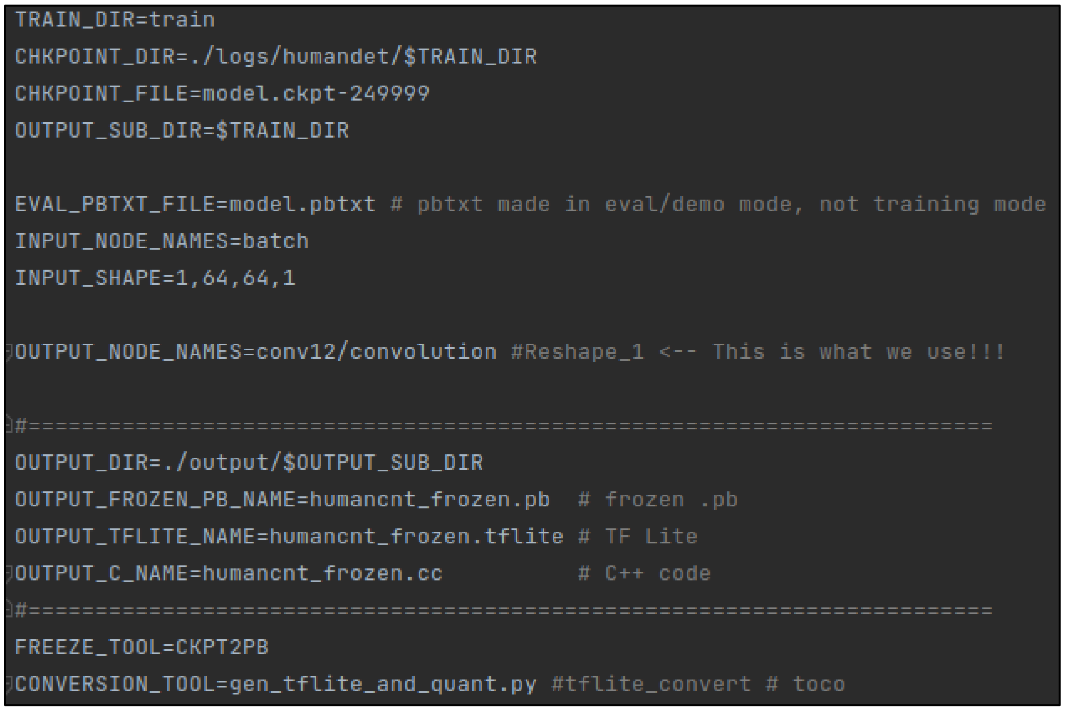
To generate pb, tflite, and carray files from checkpoints:

1. Open *@ckpt_to_cc.sh* and modify configurations if needed.

   You may change the following items:

   - TRAIN_DIR
   - CHKPOINT_DIR
   - CHKPOINT_FILE

   **Note:** If you do not change logdir and model architecture No need to change anything in below file.



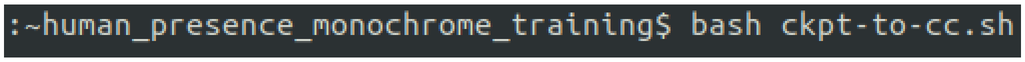
```
TRAIN_DIR=train
CHKPOINT_DIR=./logs/humandet/$TRAIN_DIR
CHKPOINT_FILE=model.ckpt-249999
OUTPUT_SUB_DIR=$TRAIN_DIR

EVAL_PBTXT_FILE=model.pbtxt # pbtxt made in eval/demo mode, not training mode
INPUT_NODE_NAMES=batch
INPUT_SHAPE=1,64,64,1

OUTPUT_NODE_NAMES=conv12/convolution #Reshape_1 <-- This is what we use!!!

#======================================================================
OUTPUT_DIR=./output/$OUTPUT_SUB_DIR
OUTPUT_FROZEN_PB_NAME=humancnt_frozen.pb  # frozen .pb
OUTPUT_TFLITE_NAME=humancnt_frozen.tflite # TF Lite
OUTPUT_C_NAME=humancnt_frozen.cc          # C++ code
#======================================================================
FREEZE_TOOL=CKPT2PB
CONVERSION_TOOL=gen_tflite_and_quant.py #tflite_convert # toco
```

**Figure 5.3. ckpt_to_cc.sh File**

2. Run ckpt_to_cc.sh file using the Bash command.

   **Important:** In this stage, use the environment with TensorFlow 2.2.0 described in the Creating the TensorFlow Lite Conversion Environment. section.

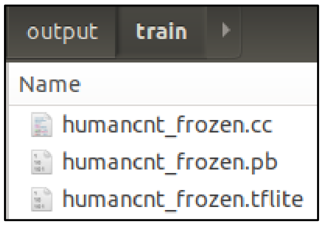
```
$ bash ckpt_to_cc.sh
```



```
:~human_presence_monochrome_training$ bash ckpt-to-cc.sh
```

**Figure 5.4. Running ckpt_to_cc.sh**

After a sucessful run, the.pb, .tflite, and .cc files are created as shown below.



```
humancnt_frozen.cc
humancnt_frozen.pb
humancnt_frozen.tflite
```

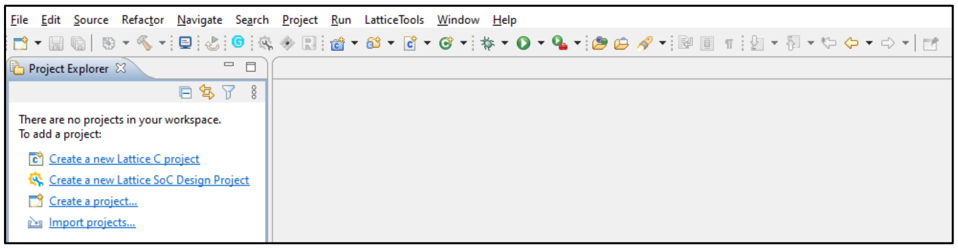
**Figure 5.5. Generated Files**

# 6. Generating the Firmware

This section describes the procedure for generating the firmware using Lattice Propel 2.0.

To generate the firmware:

1. Open Lattice Propel.


File Edit Source Refactor Navigate Search Project Run LatticeTools Window Help
Project Explorer

There are no projects in your workspace.
To add a project:

Create a new Lattice C project
Create a new Lattice SoC Design Project
Create a project...
Import projects...

Figure 6.1. Lattice Propel Main Window

2. Open a Lattice Propel C/C++ project by clicking **File > Import**.

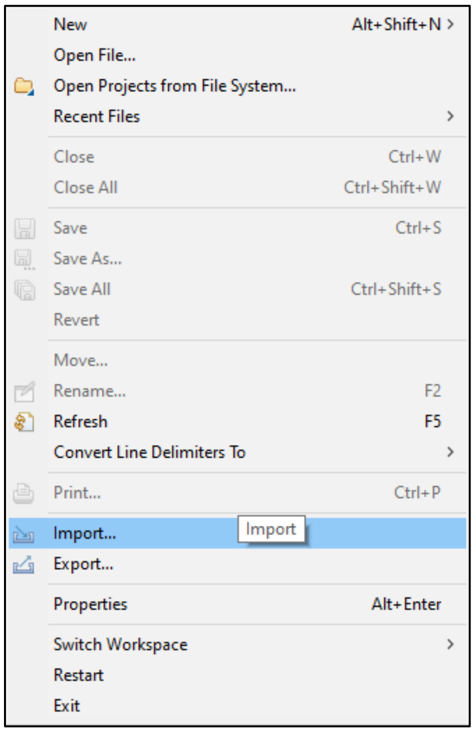
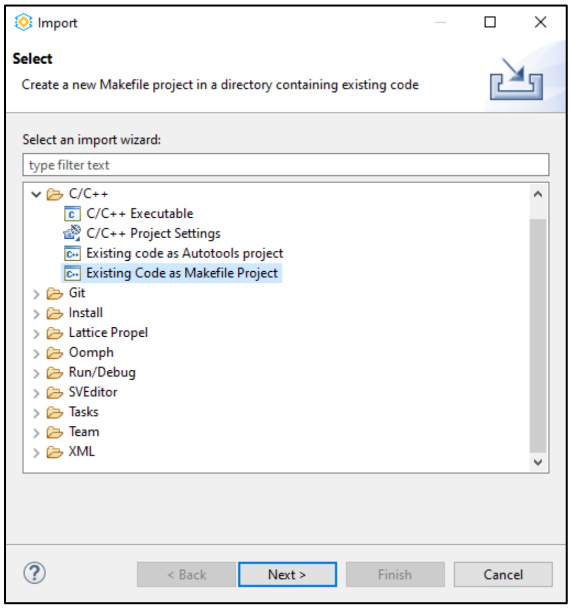


Figure 6.2. Propel Import Project

3. Select **Existing Code as Makefile Project** in the **Import** dialog box and click Next.


Import

**Select**

Create a new Makefile project in a directory containing existing code

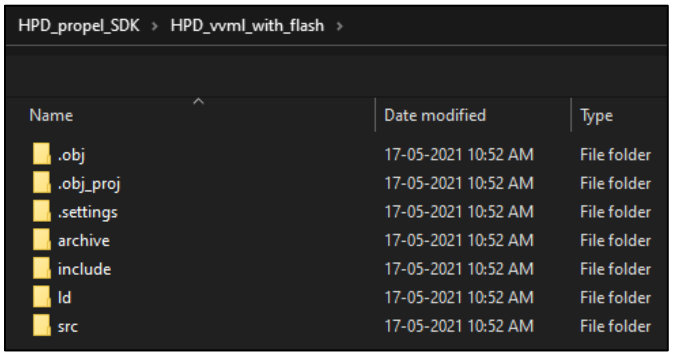
Select an import wizard:

type filter text

- C/C++
  - C/C++ Executable
  - C/C++ Project Settings
  - Existing code as Autotools project
  - Existing Code as Makefile Project
- Git
- Install
- Lattice Propel
- Oomph
- Run/Debug
- SVEditor
- Tasks
- Team
- XML

**Figure 6.3. Open Makefile Project**

4. Browse for the Propel project as shown in Figure 6.4.


HPD_propel_SDK › HPD_vvml_with_flash ›

| Name | Date modified | Type |
|---|---|---|
| .obj | 17-05-2021 10:52 AM | File folder |
| .obj_proj | 17-05-2021 10:52 AM | File folder |
| .settings | 17-05-2021 10:52 AM | File folder |
| archive | 17-05-2021 10:52 AM | File folder |
| include | 17-05-2021 10:52 AM | File folder |
| ld | 17-05-2021 10:52 AM | File folder |
| src | 17-05-2021 10:52 AM | File folder |

**Figure 6.4. Browse Project**

5. Right-click the project and select **Clean Project**.
   **Note:**
   If an error occurs related to make file, perform the following steps:
   a. Go to the project properties.
   b. Go to **C/C++ Build > Settings**.
   c. Click **Apply** and then close the dialog box.

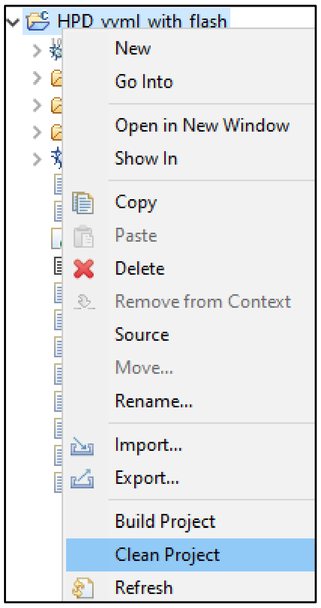
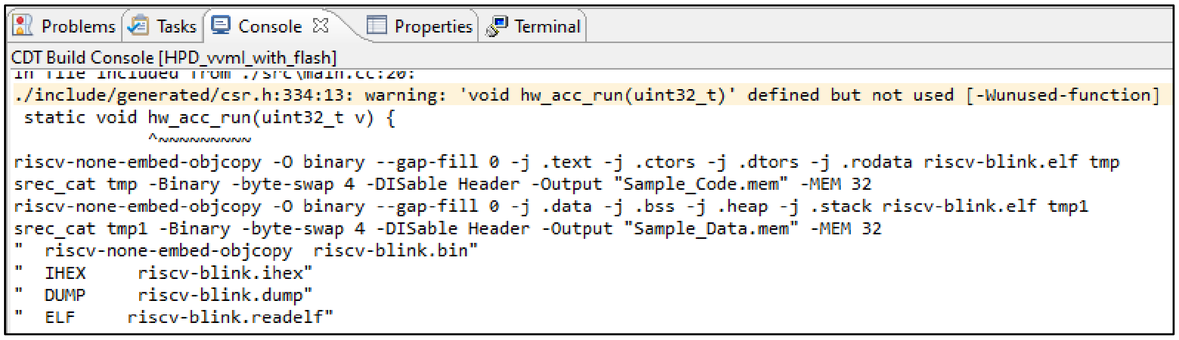


**Figure 6.5. Clean Project**

6. After related changes (Optional), build the project by right-clicking the project and selecting **Build Project**.



CDT Build Console [HPD_vvml_with_flash]

```
In file included from ./src\main.cc:20:
./include/generated/csr.h:334:13: warning: 'void hw_acc_run(uint32_t)' defined but not used [-Wunused-function]
 static void hw_acc_run(uint32_t v) {
             ^~~~~~~~~~
riscv-none-embed-objcopy -O binary --gap-fill 0 -j .text -j .ctors -j .dtors -j .rodata riscv-blink.elf tmp
srec_cat tmp -Binary -byte-swap 4 -DISable Header -Output "Sample_Code.mem" -MEM 32
riscv-none-embed-objcopy -O binary --gap-fill 0 -j .data -j .bss -j .heap -j .stack riscv-blink.elf tmp1
srec_cat tmp1 -Binary -byte-swap 4 -DISable Header -Output "Sample_Data.mem" -MEM 32
"  riscv-none-embed-objcopy  riscv-blink.bin"
"  IHEX     riscv-blink.ihex"
"  DUMP     riscv-blink.dump"
"  ELF     riscv-blink.readelf"
```

**Figure 6.6. Build Project**

The procedure produces intermediate files:

- Sample_Data.mem is used in bitstream generation for memory initialization.
- Risckv-blink.bin is later flashed on hardware along with the bitstream.

# 7. Hardware Implementation

## 7.1. Top Level Information

### 7.1.1. Block Diagram

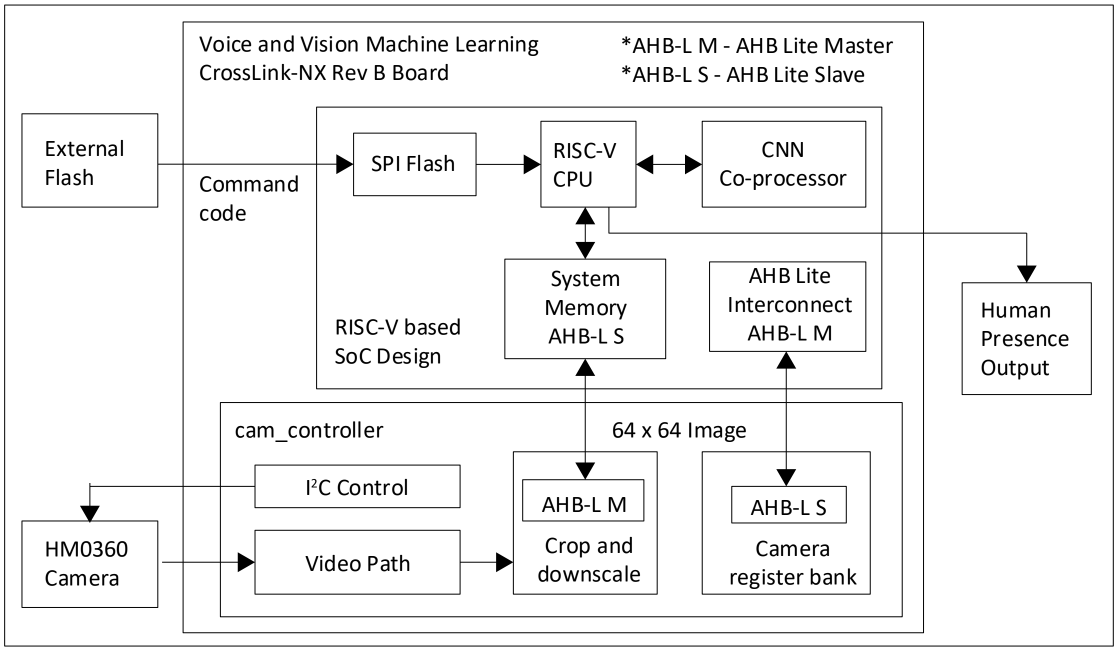


**Figure 7.1. Top Level Design Block Diagram**

### 7.1.2. Operational Flow

This section describes the flow of data across the CrossLink-NX Voice and Vision Machine Learning (Revision B) Board.

1. RISC-V based SoC Design is configured with the help of a binary (BIN) file. The BIN file is a command sequence code which is generated by the Lattice Propel SDK software.
2. Command code is written to SPI Flash controller in SoC Design top before the execution of RISC-V CPU starts.
3. The raw image data is captured using external HM0360 Camera which is configured by I$^2$C Control module and converted into RAW8 data type by Video path modules present in *cam_controller* block.
4. This RAW8 image data is further downscaled to 64x64 image resolution by Crop Downscale module. This data is written into System memory of SoC Design through an AHB Lite interface.
5. After command code and input image data both are available, they are received by RISC-V CPU.
6. Now the RISC-V and the CNN Co-processor collectively work to provide the inference results.
7. The RISC-V further performs post processing on these inference values and indicates human presence.
8. The handshake between external Camera HM0360 and RISC-V CPU present in SoC Design is managed by Camera controller register bank module with the help of an AHB Lite interface.
9. Moreover, the human presence can be observed by the printed values on the UART terminal using Lattice Propel SDK Software. The human presence output can also be observed by LED output on the board.

### 7.1.3. Core Customization

The AHB Lite MASTER & SLAVE interface parameters used in design top *hpd_tflite_top.v* module are shown below in Table 7.2. These are constant parameters and not to be modified.

**Table 7.2. Top Design Parameters**

| Constant | Default (Decimal) | Description |
|---|---|---|
| MASTER_HADDR_WIDTH | 32 | Indicates width of AHB Lite Master Address bus signal. |
| MASTER_HWDATA_WIDTH | 32 | Indicates width of AHB Lite Master Data Write bus signal. |
| MASTER_HRDATA_WIDTH | 32 | Indicates width of AHB Lite Master Data read bus signal. |
| MASTER_HBURST | 3 | Indicates width of AHB Lite Master Burst type signal. |
| MASTER_HTRANS | 2 | Indicates width of AHB Lite Master Transfer type signal. |
| MASTER_HSIZE | 3 | Indicates width of AHB Lite Master Transfer size signal. |
| SLAVE_DATA_WIDTH | 32 | Indicates width of AHB Lite Slave Data Bus. |
| SLAVE_ADDR_WIDTH | 32 | Indicates width of AHB Lite Slave Address Bus. |

### 7.1.4. Pre-processing CNN

The pre-processing of image captured by external camera is handled by cam_controller module. The Video path modules receive the real time camera input data through MIPI DPHY lines.

Output from Video path module is a stream of RAW8 data that reflects the camera image which is given to crop and downscale module.

The Crop downscale module processes that image data and generates downscaled input of 64x64 image data for CNN Co-processor. The Pre processing flow is explained below.

RAW8 data values for each pixel are fed serially line by line for an image frame. These values are considered as valid only when horizontal and vertical masks are inactive. Mask parameters set to mask out boundary area of resolution 640 × 480 are shown below in Figure 7.2.

Left masking = 64 and Right masking = 576 (Obtained as 64+512)

Top masking = 1 and Bottom masking = 480 (Obtained as 1 to 480 lines)

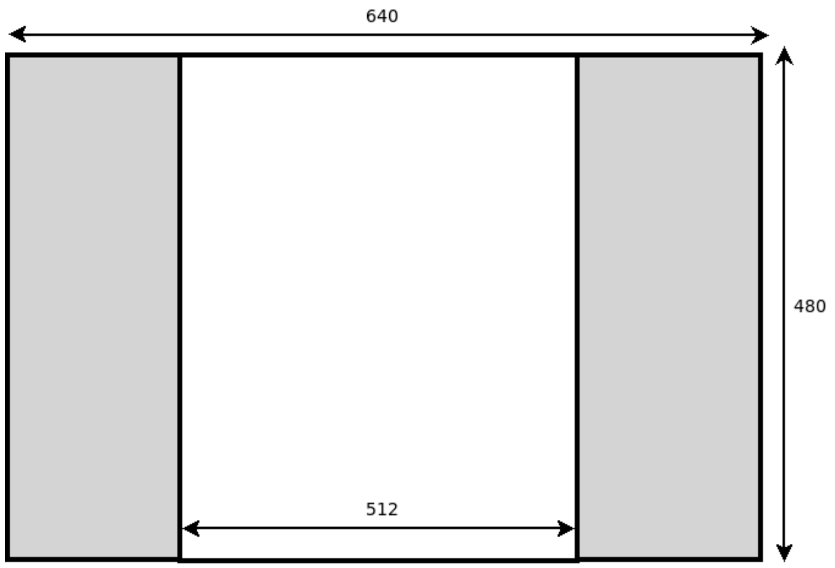


**Figure 7.2.Masking**

This 512 × 480 frame block is downscaled into 64 × 60 resolution image as shown in Figure 7.3 by accumulating 8 × 8 pixels into a single pixel. For example, 512/8 × 480/8 = 64 × 60).

The actual resolution of downscaled image required by CNN is 64x64. So in order to obtain extra 4 vertical lines required in current 64x60 image, total four more lines are padded to 64x60 image and made to 64x64. Hence upper two lines and lower two lines are added as all values 8'd0 (i.e; 64 horizontal pixels x extra 4 lines.)

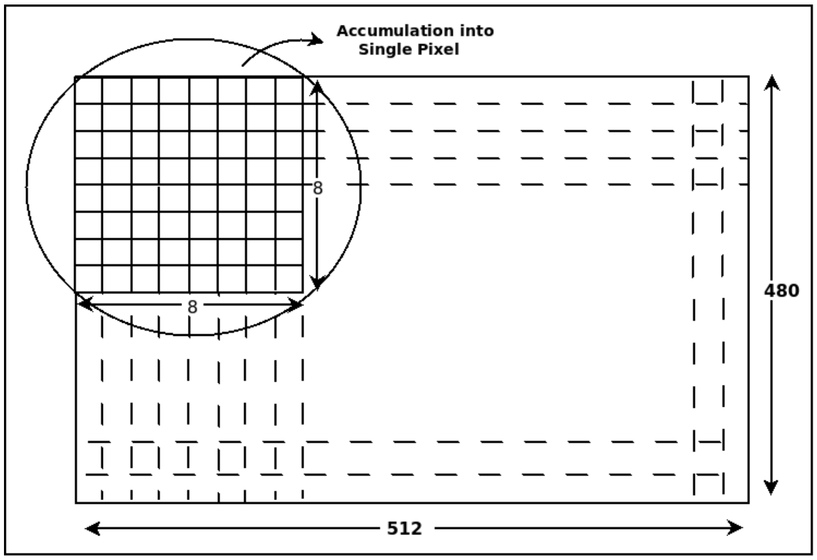


**Figure 7.3. Downscaling**

This accumulated value is written into Line Buffer. Line Buffer is a True Dual-Port RAM. Accumulated RAW8 pixel values for 8 × 8 grid are stored in the same memory location.

When Data is read from memory, the RAW8 value is divided by 64 (that is. the area of 8 × 8 grid) to take the average of 8 × 8 grid matrix.

For CNN 64x64 Data is then stored at dedicated Base Address location in System memory present in SoC Design through AHB Lite MASTER (*cam_controller*) to AHB Lite SLAVE (*sysmem0_inst*) interface.

### 7.1.5. Camera Controller Register Bank with AHB Lite Slave

This module basically looks after the handshake required to maintain a continuous flow of downscaled image data from Crop downscale to RISC-V CPU present in SoC Design top.

When 64 × 64 downscaled data is received by RISC-V CPU after reading it from System memory, the response from RISCV has to be informed to Crop downscale module so that it can start operating on image data of next frame and store it back again in the System memory. This operation will be going on continuously.

This response exchange happens through the AHB Lite MASTER in SoC Design connected with AHB Lite SLAVE interface present in this module via AHB Lite interconnect.

As shown in Table 7.3, mainly four registers are used for the handshake development in this module.

**Table 7.3. Camera Controller Registers**

| Register | AHB Address Offset | Access Type | Bit | Description |
|---|---|---|---|---|
| CAM_CFG_REG | 32'h00108000 | R/W | [0] – cam_sign_mode | 1'b0 – Unsigned, Input range is [0,255]<br>1'b1 – Signed, Input range is [-128,127] |
| CAM_CTRL_REG | 32'h00108004 | R/W | [0] – cnn_rdy | 1'b0 – RISC-V Processor is busy<br>1'b1 – RISC-V Processor is ready, Capture next frame |
| CAM_STATUS_REG | 32'h00108008 | R | [0] – cam_data_rdy | 1'b0 – New frame data absent from cam_controller.v<br>1'b1 – New frame data present in System memory from cam_controller.v |
| CAM_DATA_ADDR_REG | 32'h0010800C | R/W | [31:0] – cam_data_addr | Memory address for storing the 64x64 frame data. |

**Implemented Handshake Flow**

The following describes the handshake flow:

1. RISC-V configures cam_sign_mode and cam_data_addr.
2. RISC-V then asserts cnn_rdy when it is ready to process a new frame.
3. Camera controller captures a new frame and stores it in memory, whose address is specified by cam_data_addr.
4. Camera controller asserts cam_data_rdy.
5. RISC-V clears cnn_rdy bit.
6. Camera Controller clears cam_data_rdy bit
7. Repeat again from step 2.

## 7.2. SoC Design Information

The RISC-V based SoC Top level Design is created using software Lattice Propel Builder Version 2.0. The Device Family used to build the Design is LIFCL – 40 – 8MG289C. The top block diagram is shown below in Figure 7.4.

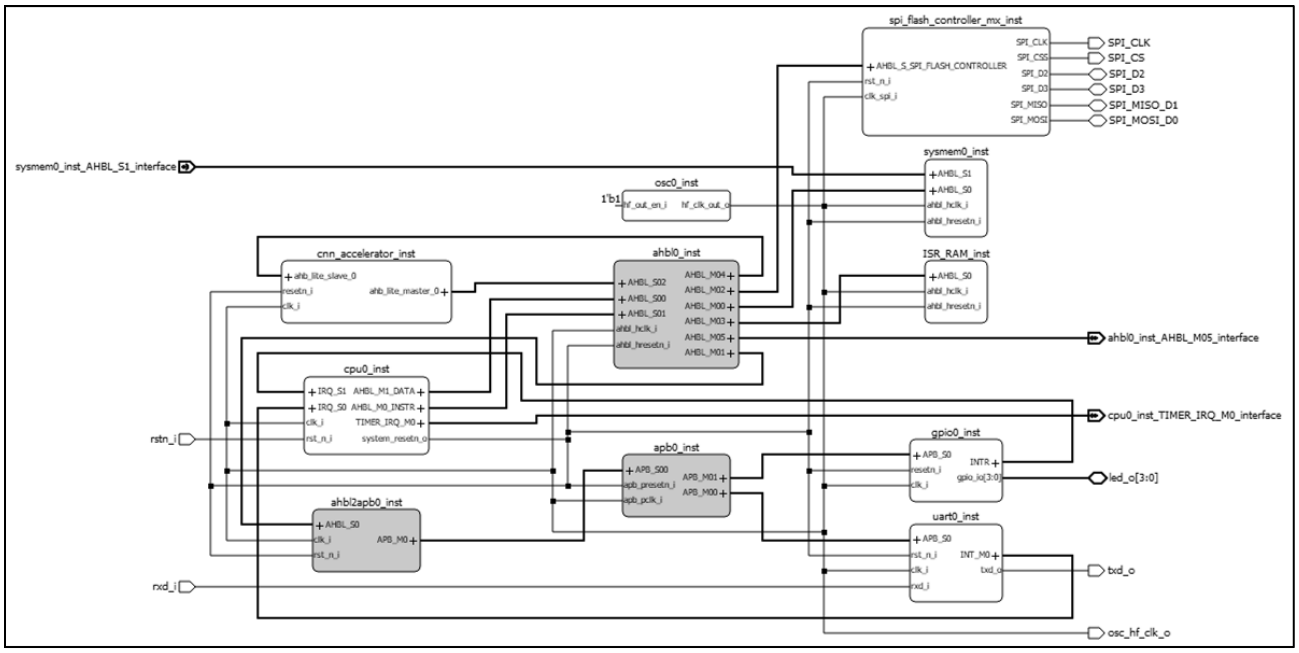


**Figure 7.4. SoC Design Top Block Diagram**

The modules and respective IPs used to build this SoC Design are shown below in Table 7.4.

**Table 7.4. IP used in Lattice Builder SoC Design**

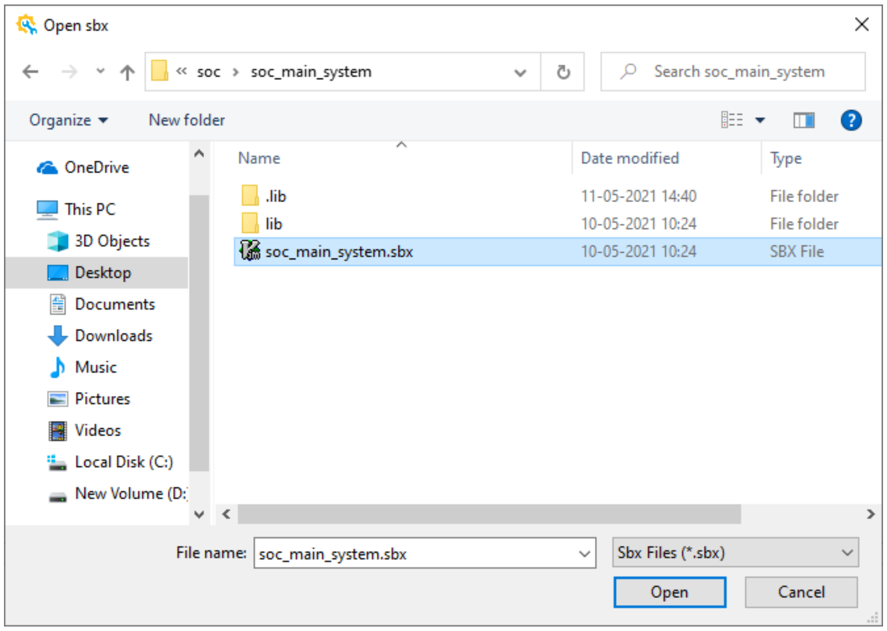
| Sr No | Design Blocks | IP | IP Version |
|---|---|---|---|
| 1 | cpu0_inst | RISC-V MC | v1.1.0 |
| 2 | cnn_coprocessor | CNN_Coprocessor_Unit | v1.0.1 |
| 3 | osc0_inst | OSC | v1.0.1 |
| 4 | spi_flash_controller_mx | SPI_FLASH_CONTROLLER_MX | v1.0.0 |
| 5 | sysmem0_inst | System_Memory | v1.0.2 |
| 6 | ISR_RAM_inst | System_Memory | v1.0.2 |
| 7 | gpio0_inst | GPIO | v1.3.0 |
| 8 | uart0_inst | UART | v1.1.0 |
| 9 | apb0_inst | APB Interconnect | v1.0.4 |
| 10 | ahbl0_inst | AHB Lite Interconnect | v1.1.2 |
| 11 | ahbl2apb0_inst | AHB Lite to APB Bridge | v1.0.4 |

The CNN_Coprocessor_Unit and SPI_FLASH_CONTROLLER_MX IPs are to be added using "Install a user IP" option in IP Catelog. Other mentioned IPs are available in the Lattice Propel Builder IP Catelog. GPIO, UART and RISC-V IPs have to be downloaded and installed from "IP on Server". Refer to the IP Installation in Lattice Radiant or Lattice Propel Software section for IP installations.

### 7.2.1. Opening the SoC Project

To open an SoC project:

1. In the Lattice Propel Builder main interface, choose **File** > **Open Design**.The **Open sbx** dialog box is diaplayed.
2. From the SoC project path, as shown in Figure 7.5, select the *.sbx* file *hpd_tflite_nx_vvml /soc / soc_main_system / soc_main_system.sbx*.



| Name | Date modified | Type |
|---|---|---|
| .lib | 11-05-2021 14:40 | File folder |
| lib | 10-05-2021 10:24 | File folder |
| soc_main_system.sbx | 10-05-2021 10:24 | SBX File |

File name: soc_main_system.sbx
Sbx Files (*.sbx)

**Figure 7.5. Open SoC Project**

### 7.2.2. Generating RTL File from SoC

This section provides the steps in creating the RTL file for the SoC design after opening the *.sbx* project and applying modifications, if any.
To generate the RTL file:

1. Right-click the Schematic window and choose **Relayout**.
2. Click the **Validate Design** button. This starts Design Rule Check (DRC) to see if there are unwanted connections or overlapping address space. The DRC results is displayed in the **Tcl Console** window.
3. Click the **Generate** button. This saves the Propel Builder design and runs the DRC.
4. After the design is generated and validated without any errors, click the **Save Design** button.
5. The SoC Design RTL file is created in hpd_tflite_nx_vvml /soc / soc_main_system / soc_main_system.v.

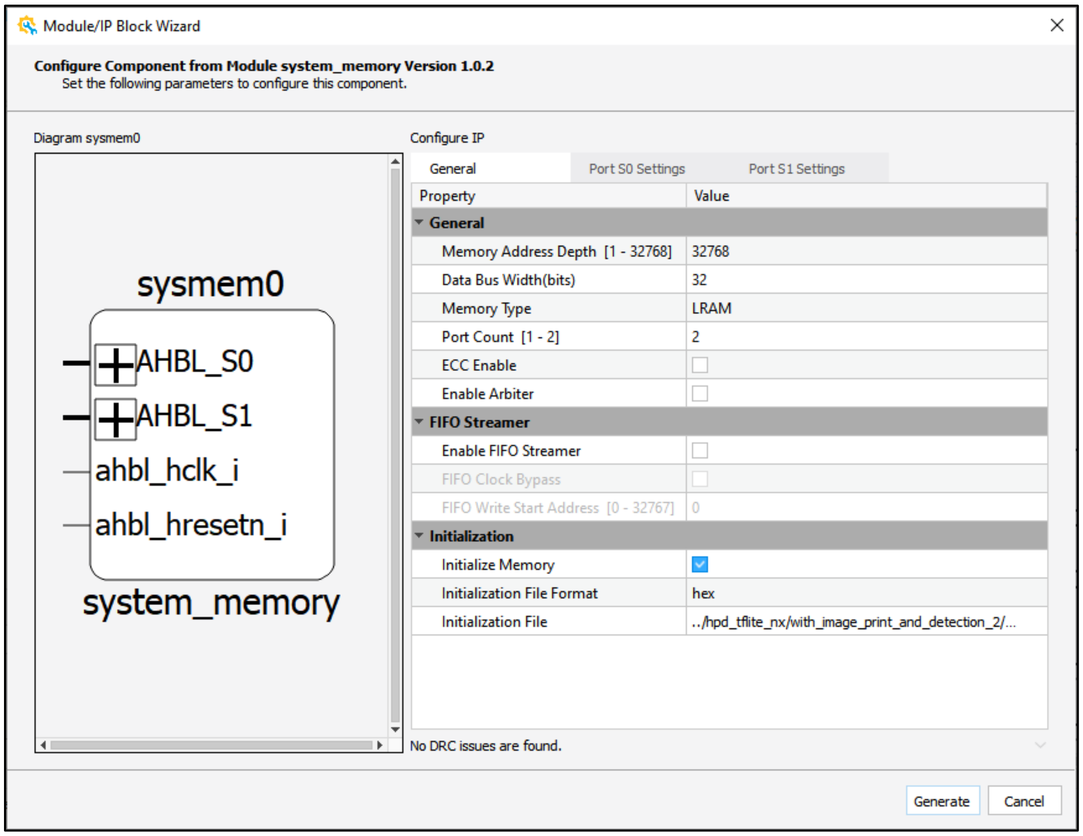
### 7.2.3. Initializing Memory in SoC Design

The memory file basically contains the data section addresses for the firmware (BIN file). This section provides the steps in initializing the system memory in the SoC Design after opening the *.sbx* project in Lattice Propel Builder. Perform this procedure before generating RTL from SoC as discussed in the Generating RTL File from SoC section.

To initialize system memory:

1. Find the memory initialization file in hpd_tflite_nx_vvml /sw/TF_lite_RiscV_Acc/sample_data.mem.
2. In the Schematic window of Lattice Propel Builder, double click the *sysmem0_inst* module.
3. The System Memory Module/IP Block wizard opens as shown in Figure 7.6.
4. Select the **Initialize memory** option.
5. In **Initialization File**, browse and select the *sample_data.mem* file and generate the IP Block again.

6. After the system memory IP Block is generated, follow the steps mentioned in the Generating RTL File from SoC section again, This generates the updated RTL file for the SoC Design.



**Configure Component from Module system_memory Version 1.0.2**
Set the following parameters to configure this component.

| Property | Value |
|---|---|
| **General** | |
| Memory Address Depth [1 - 32768] | 32768 |
| Data Bus Width(bits) | 32 |
| Memory Type | LRAM |
| Port Count [1 - 2] | 2 |
| ECC Enable | ☐ |
| Enable Arbiter | ☐ |
| **FIFO Streamer** | |
| Enable FIFO Streamer | ☐ |
| FIFO Clock Bypass | ☐ |
| FIFO Write Start Address [0 - 32767] | 0 |
| **Initialization** | |
| Initialize Memory | ✓ |
| Initialization File Format | hex |
| Initialization File | ../hpd_tflite_nx/with_image_print_and_detection_2/... |

No DRC issues are found.

**Figure 7.6. Initialize Memory File in SoC Design**

For detailed information on generating SoC Design from scratch by adding components and connections in Lattice Propel Builder, refer to the Lattice-Propel-Builder-2.0-User-Guide (FPGA-UG-02127).

# 8. Creating FPGA Bitstream file

## 8.1. Generating the Bitstream File

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

To generate the FPGA bitstream file:

1. Open Lattice Radiant Software version 2.2 as shown in Figure 8.1.

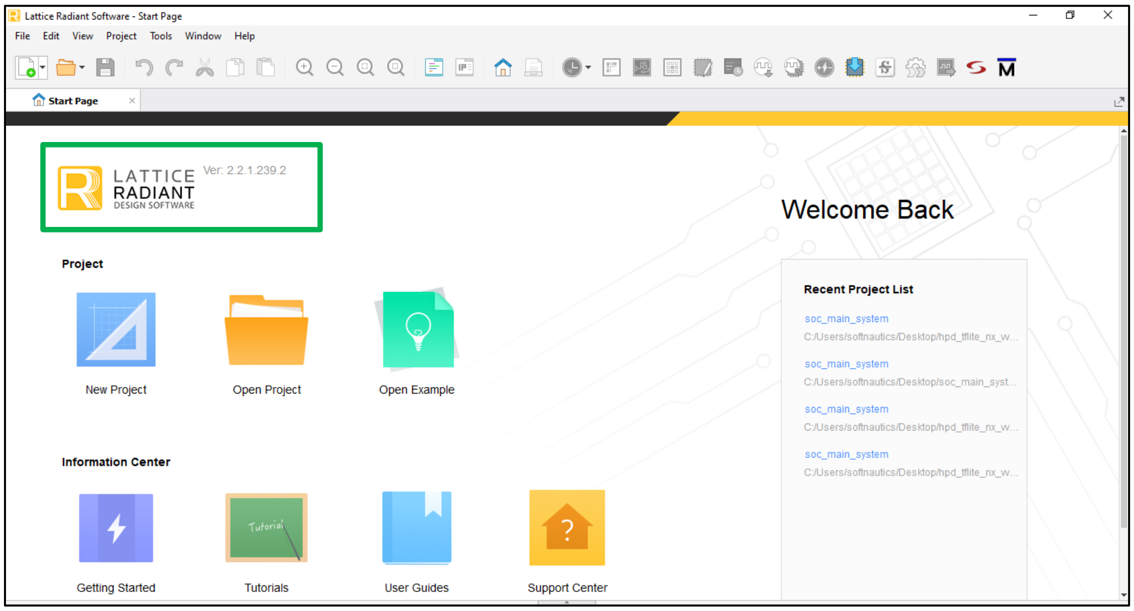
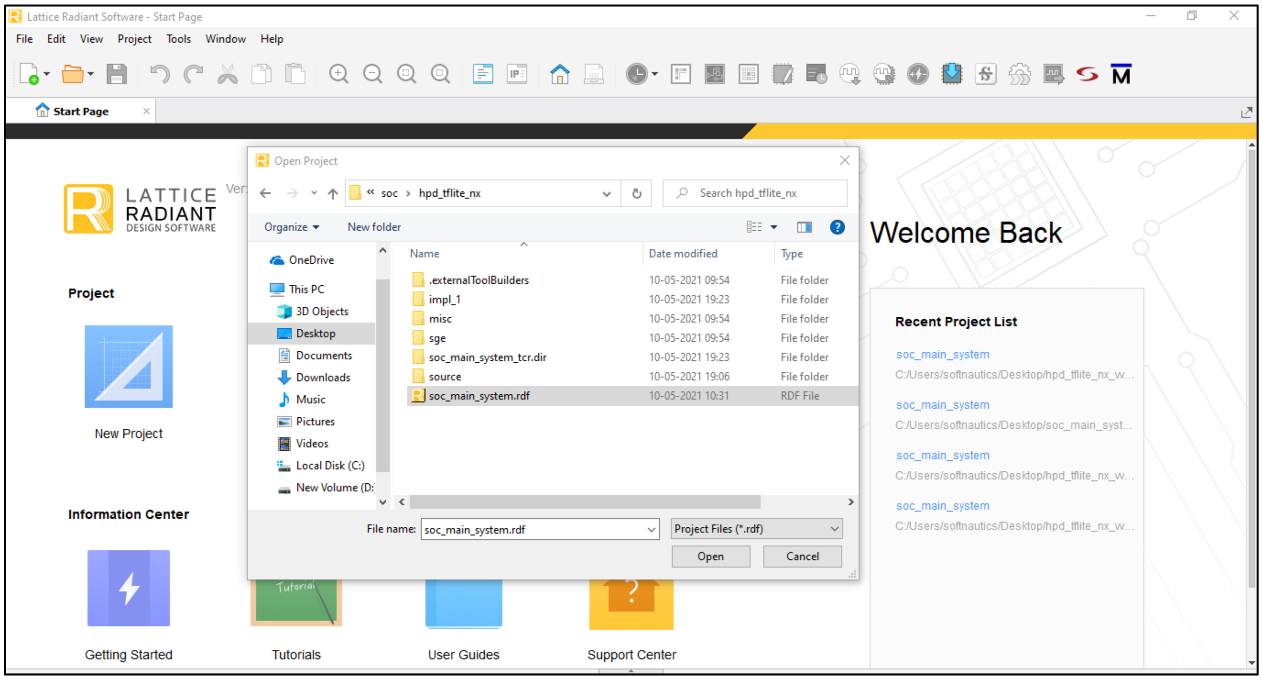

Lattice Radiant Software - Start Page
File Edit View Project Tools Window Help
Start Page
LATTICE RADIANT DESIGN SOFTWARE
Ver: 2.2.1.239.2
Welcome Back
Project
New Project
Open Project
Open Example
Information Center
Getting Started
Tutorials
User Guides
Support Center
Recent Project List
soc_main_system
C:/Users/softnautics/Desktop/hpd_tflite_nx_w...
soc_main_system
C:/Users/softnautics/Desktop/soc_main_syst...
soc_main_system
C:/Users/softnautics/Desktop/hpd_tflite_nx_w...
soc_main_system
C:/Users/softnautics/Desktop/hpd_tflite_nx_w...

**Figure 8.1. Lattice Radiant Start Page**

2. Click File > Open Project.
3. Open the Lattice Radiant project file *soc_main_system.rdf* from *hpf_tflite_nx* folder. As shown in Figure 8.2**,** you can also open project by clicking the folder shown in the **Open Project** dialog box.



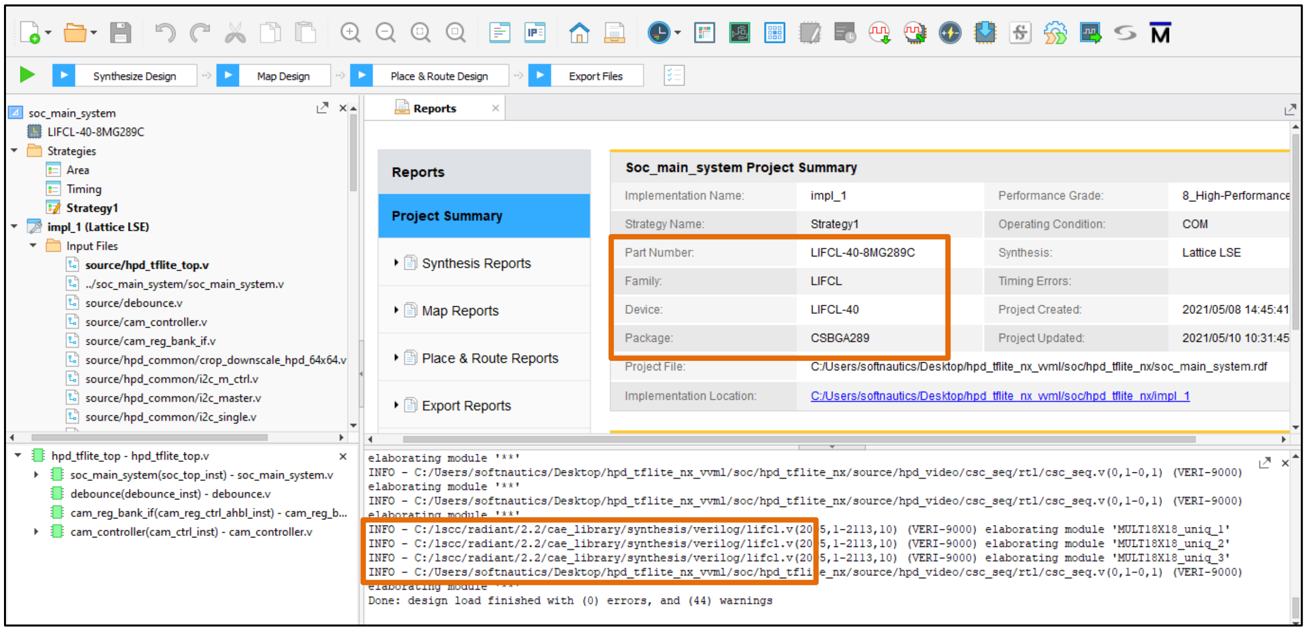
| Name | Date modified | Type |
|---|---|---|
| .externalToolBuilders | 10-05-2021 09:54 | File folder |
| impl_1 | 10-05-2021 19:23 | File folder |
| misc | 10-05-2021 09:54 | File folder |
| sge | 10-05-2021 09:54 | File folder |
| soc_main_system_tcr.dir | 10-05-2021 19:23 | File folder |
| source | 10-05-2021 19:06 | File folder |
| soc_main_system.rdf | 10-05-2021 10:31 | RDF File |

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

Recent Project List:
- soc_main_system — C:/Users/softnautics/Desktop/hpd_tflite_nx_w...
- soc_main_system — C:/Users/softnautics/Desktop/soc_main_syst...
- soc_main_system — C:/Users/softnautics/Desktop/hpd_tflite_nx_w...
- soc_main_system — C:/Users/softnautics/Desktop/hpd_tflite_nx_w...

**Figure 8.2. Lattice Radiant – Open Project**

4. After opening the project file, verify the items below as shown in Figure 8.3.
   - Design loaded with zero errors message shown in the Output pane.
   - Check for the information below in the project summary window.
     - Part Number : LIFCL-40-8MG289C
     - Family : LIFCL
     - Device : LIFCL-40
     - Package : CSBGA289


Synthesize Design
Map Design
Place & Route Design
Export Files
soc_main_system
LIFCL-40-8MG289C
Strategies
Area
Timing
Strategy1
impl_1 (Lattice LSE)
Input Files
source/hpd_tflite_top.v
../soc_main_system/soc_main_system.v
source/debounce.v
source/cam_controller.v
source/cam_reg_bank_if.v
source/hpd_common/crop_downscale_hpd_64x64.v
source/hpd_common/i2c_m_ctrl.v
source/hpd_common/i2c_master.v
source/hpd_common/i2c_single.v
hpd_tflite_top - hpd_tflite_top.v
soc_main_system(soc_top_inst) - soc_main_system.v
debounce(debounce_inst) - debounce.v
cam_reg_bank_if(cam_reg_ctrl_ahbl_inst) - cam_reg_b...
cam_controller(cam_ctrl_inst) - cam_controller.v
Reports
Reports
Project Summary
Synthesis Reports
Map Reports
Place & Route Reports
Export Reports

Soc_main_system Project Summary

| | | | |
|---|---|---|---|
| Implementation Name: | impl_1 | Performance Grade: | 8_High-Performance |
| Strategy Name: | Strategy1 | Operating Condition: | COM |
| Part Number: | LIFCL-40-8MG289C | Synthesis: | Lattice LSE |
| Family: | LIFCL | Timing Errors: | |
| Device: | LIFCL-40 | Project Created: | 2021/05/08 14:45:41 |
| Package: | CSBGA289 | Project Updated: | 2021/05/10 10:31:45 |
| Project File: | C:/Users/softnautics/Desktop/hpd_tflite_nx_wml/soc/hpd_tflite_nx/soc_main_system.rdf | | |
| Implementation Location: | C:/Users/softnautics/Desktop/hpd_tflite_nx_wml/soc/hpd_tflite_nx/impl_1 | | |

```
elaborating module '**'
INFO - C:/Users/softnautics/Desktop/hpd_tflite_nx_vvml/soc/hpd_tflite_nx/source/hpd_video/csc_seq/rtl/csc_seq.v(0,1-0,1) (VERI-9000)
elaborating module '**'
INFO - C:/Users/softnautics/Desktop/hpd_tflite_nx_vvml/soc/hpd_tflite_nx/source/hpd_video/csc_seq/rtl/csc_seq.v(0,1-0,1) (VERI-9000)
elaborating module '**'
INFO - C:/lscc/radiant/2.2/cae_library/synthesis/verilog/lifcl.v(205,1-2113,10) (VERI-9000) elaborating module 'MULT18X18_uniq_1'
INFO - C:/lscc/radiant/2.2/cae_library/synthesis/verilog/lifcl.v(205,1-2113,10) (VERI-9000) elaborating module 'MULT18X18_uniq_2'
INFO - C:/lscc/radiant/2.2/cae_library/synthesis/verilog/lifcl.v(205,1-2113,10) (VERI-9000) elaborating module 'MULT18X18_uniq_3'
INFO - C:/Users/softnautics/Desktop/hpd_tflite_nx_vvml/soc/hpd_tflite_nx/source/hpd_video/csc_seq/rtl/csc_seq.v(0,1-0,1) (VERI-9000)
elaborating module '**'
Done: design load finished with (0) errors, and (44) warnings
```

**Figure 8.3. Lattice Radiant – Design Load Check after Opening Project File**

5. If the design is loaded without errors, click the bitstream generation button as shown in Figure 8.4.

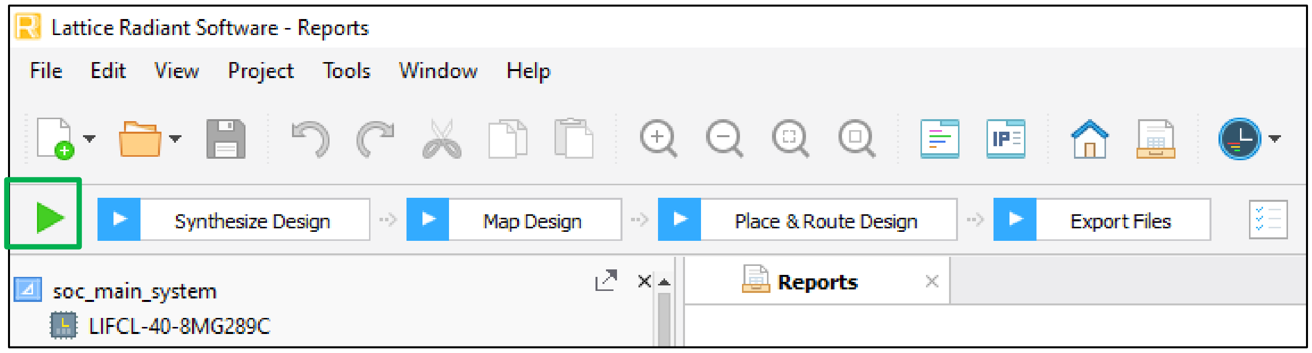
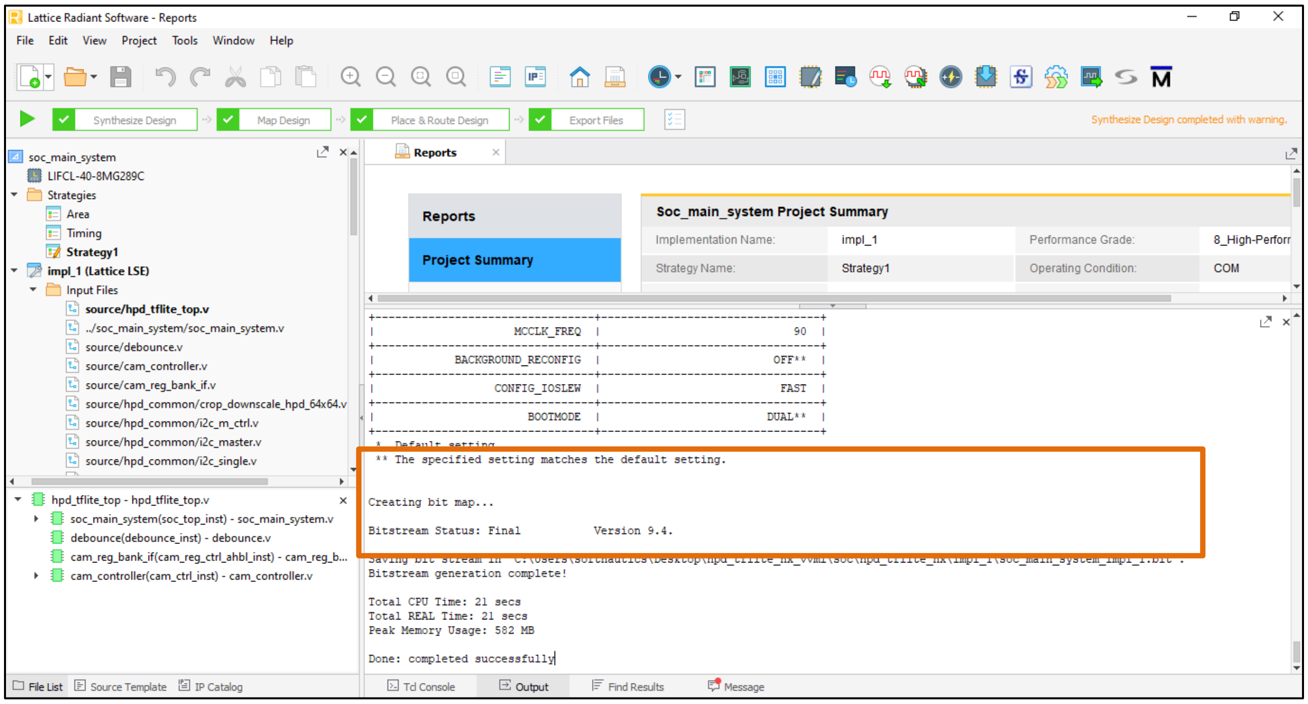


**Figure 8.4. Lattice Radiant –Bitstream Generation Button**

6. After generating the bitstream, the Lattice Radiant tool displays the *Saving bit stream in …* message in Reports window. Generated bitstream can be found in "*Implementation – impl1 folder*" shown in Figure 8.5.


Lattice Radiant Software - Reports
File Edit View Project Tools Window Help
Synthesize Design
Map Design
Place & Route Design
Export Files
Synthesize Design completed with warning.
soc_main_system
LIFCL-40-8MG289C
Strategies
Area
Timing
Strategy1
impl_1 (Lattice LSE)
Input Files
source/hpd_tflite_top.v
../soc_main_system/soc_main_system.v
source/debounce.v
source/cam_controller.v
source/cam_reg_bank_if.v
source/hpd_common/crop_downscale_hpd_64x64.v
source/hpd_common/i2c_m_ctrl.v
source/hpd_common/i2c_master.v
source/hpd_common/i2c_single.v
hpd_tflite_top - hpd_tflite_top.v
soc_main_system(soc_top_inst) - soc_main_system.v
debounce(debounce_inst) - debounce.v
cam_reg_bank_if(cam_reg_ctrl_ahbl_inst) - cam_reg_b...
cam_controller(cam_ctrl_inst) - cam_controller.v
File List
Source Template
IP Catalog
Reports
Reports
Project Summary
Tcl Console
Output
Find Results
Message

Soc_main_system Project Summary

| | | | |
|---|---|---|---|
| Implementation Name: | impl_1 | Performance Grade: | 8_High-Perforr |
| Strategy Name: | Strategy1 | Operating Condition: | COM |

```
+-----------------------------------+-----------------------------------+
|                       MCCLK_FREQ  |                               90  |
+-----------------------------------+-----------------------------------+
|              BACKGROUND_RECONFIG  |                             OFF**  |
+-----------------------------------+-----------------------------------+
|                    CONFIG_IOSLEW  |                             FAST  |
+-----------------------------------+-----------------------------------+
|                         BOOTMODE  |                            DUAL**  |
+-----------------------------------+-----------------------------------+
 *  Default setting.
 ** The specified setting matches the default setting.


Creating bit map...

Bitstream Status: Final           Version 9.4.

Saving bit stream in "C:\Users\softnautics\Desktop\hpd_tflite_nx_vvml\soc\hpd_tflite_nx\impl_1\soc_main_system_impl_1.bit".
Bitstream generation complete!

Total CPU Time: 21 secs
Total REAL Time: 21 secs
Peak Memory Usage: 582 MB

Done: completed successfully
```

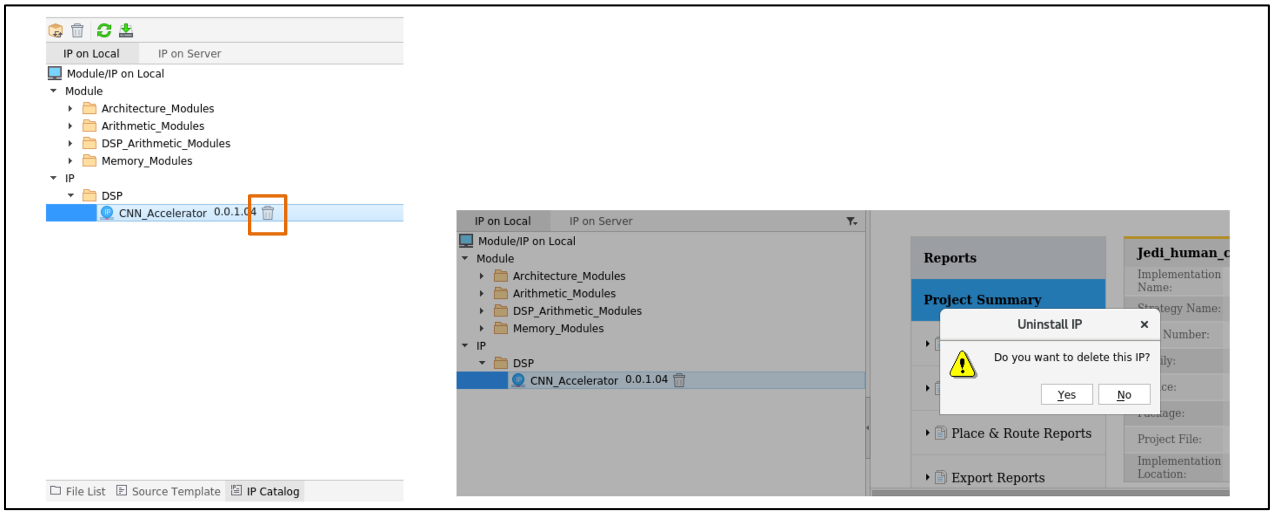
**Figure 8.5. Lattice Radiant – Bit File Generation Report Window**

## 8.2. IP Installation in Lattice Radiant or Lattice Propel Software

After loading the design without any errors, peform the steps below to uninstall an old IP or install the latest version of an IP.

To uninstall an existing IP:

1. Click **IP Catelog** and go to **IP > DSP** in the IP tree.
2. In the IP tree, go to **IP > DSP** and select the IP to be uninstalled.
3. Click the **Delete** button as highlighted in Figure 8.6.
4. In the **Uninstall IP** prompt, click **Yes**.

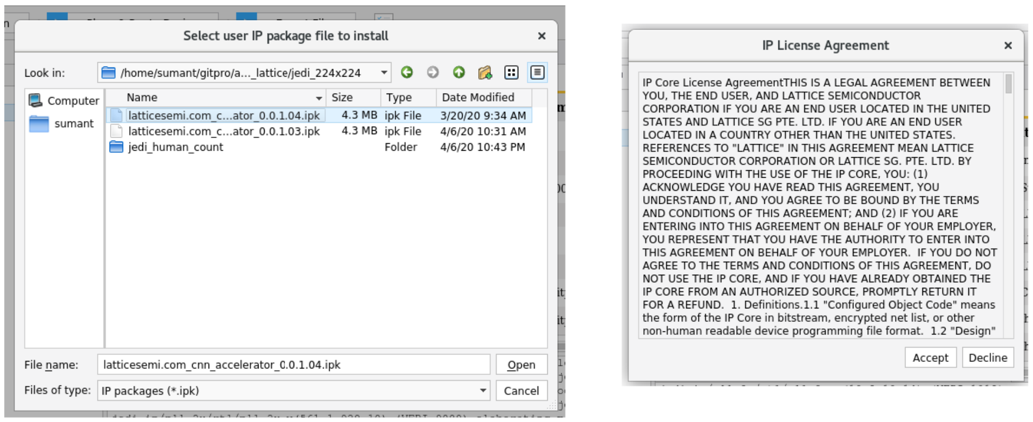


Do you want to delete this IP?

Figure 8.6. Lattice Radiant or Lattice Propel – Uninstall Existing IP

To install a new IP:

1. Select **Install a User IP** in **IP Catelog.**
2. In the **Select user IP package file to install** dialog box, select the IP package (.ipk) to be installed and click **Open**.
3. In the **IP License Agreement** window, select **Accept**.



Look in: /home/sumant/gitpro/a..._lattice/jedi_224x224

| Name | Size | Type | Date Modified |
|---|---|---|---|
| latticesemi.com_c...ator_0.0.1.04.ipk | 4.3 MB | ipk File | 3/20/20 9:34 AM |
| latticesemi.com_c...ator_0.0.1.03.ipk | 4.3 MB | ipk File | 4/6/20 10:31 AM |
| jedi_human_count | | Folder | 4/6/20 10:43 PM |

File name: latticesemi.com_cnn_accelerator_0.0.1.04.ipk

Files of type: IP packages (*.ipk)

IP Core License AgreementTHIS IS A LEGAL AGREEMENT BETWEEN YOU, THE END USER, AND LATTICE SEMICONDUCTOR CORPORATION IF YOU ARE AN END USER LOCATED IN THE UNITED STATES AND LATTICE SG PTE. LTD. IF YOU ARE AN END USER LOCATED IN A COUNTRY OTHER THAN THE UNITED STATES. REFERENCES TO "LATTICE" IN THIS AGREEMENT MEAN LATTICE SEMICONDUCTOR CORPORATION OR LATTICE SG. PTE. LTD. BY PROCEEDING WITH THE USE OF THE IP CORE, YOU: (1) ACKNOWLEDGE YOU HAVE READ THIS AGREEMENT, YOU UNDERSTAND IT, AND YOU AGREE TO BE BOUND BY THE TERMS AND CONDITIONS OF THIS AGREEMENT; AND (2) IF YOU ARE ENTERING INTO THIS AGREEMENT ON BEHALF OF YOUR EMPLOYER, YOU REPRESENT THAT YOU HAVE THE AUTHORITY TO ENTER INTO THIS AGREEMENT ON BEHALF OF YOUR EMPLOYER. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT, DO NOT USE THE IP CORE, AND IF YOU HAVE ALREADY OBTAINED THE IP CORE FROM AN AUTHORIZED SOURCE, PROMPTLY RETURN IT FOR A REFUND. 1. Definitions.1.1 "Configured Object Code" means the form of the IP Core in bitstream, encrypted net list, or other non-human readable device programming file format. 1.2 "Design"

Figure 8.7. Lattice Radiant or Lattice Propel – Installing New IP

You can also download and install required IPs from **IP Catelog > IP on server** option. Then click the **Refresh** button to see the installed versions in **IP on Local** space.

After the IP is installed, the bitstream can be generated again by clicking the Bitstream generation button.

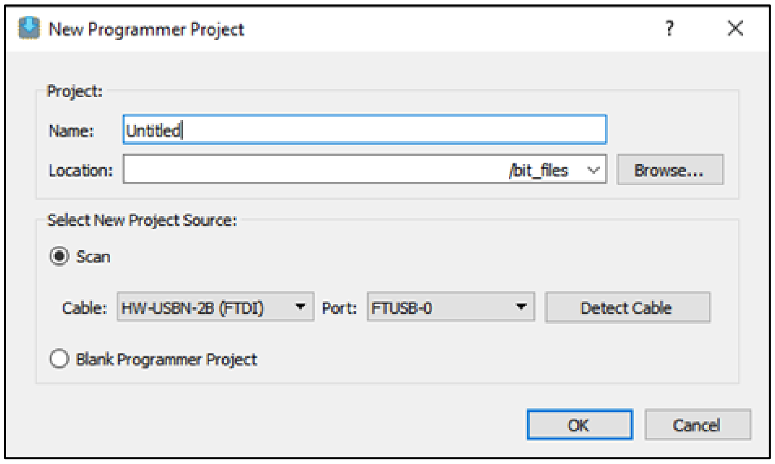
# 9. Programming the Demo

## 9.1. Erasing the CrossLink-NX VVML Contents Prior to Reprogramming

Follow this procedure to initially erase the old programmed data to CrossLink-NX Voice and Vision ML board before re-programming the new generated Bit file with required BIN to SPI Flash. Keep the board powered ON when re-programming the SPI Flash.

To erase the CrossLink-NX VVML Board contents.

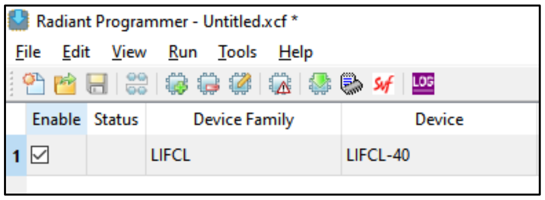
1. Launch the Lattice Radiant Programmer with **Create a new blank project**.



Name: Untitled
Location: /bit_files
Scan [✓]
Cable: HW-USBN-2B (FTDI)
Port: FTUSB-0
Blank Programmer Project

**Figure 9.1. Lattice Radiant Programmer – Default Screen**

2. Select the Device options as shown in Figure 9.2.

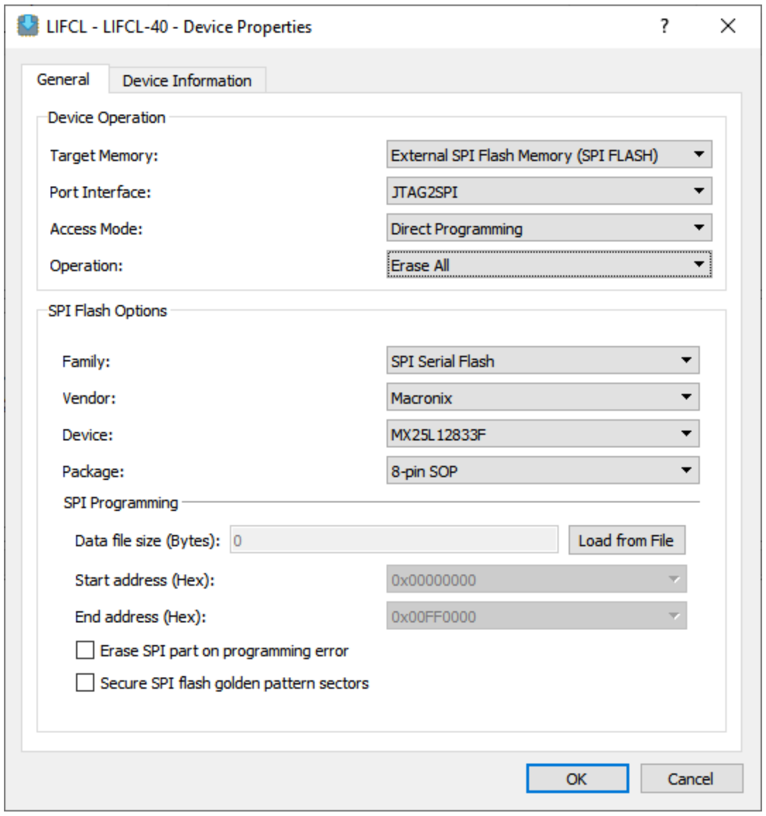


|  | Enable | Status | Device Family | Device |
|---|---|---|---|---|
| 1 | ✓ |  | LIFCL | LIFCL-40 |

**Figure 9.2. Radiant Programmer – Device Selection**

4. Right-click on **Operations** and select **Device Properties**.
5. Select the options shown in Figure 9.3 to initially erase the previously programmed contents.
6. Click **OK** to close the **Device Properties** dialog box.
7. Press and hold the SW5 button on the board and click the **Program** button to start the erase operation.
8. Release the button after the **Operation Successful** message is displayed in output console as shown in Figure 9.6.

The SW5 program button and the SW4 reset button on the CrossLink-NX VVML Board are shown in Figure 9.4.


LIFCL - LIFCL-40 - Device Properties

General | Device Information

Device Operation

Target Memory: External SPI Flash Memory (SPI FLASH)
Port Interface: JTAG2SPI
Access Mode: Direct Programming
Operation: Erase All

SPI Flash Options

Family: SPI Serial Flash
Vendor: Macronix
Device: MX25L12833F
Package: 8-pin SOP

SPI Programming

Data file size (Bytes): 0
Start address (Hex): 0x00000000
End address (Hex): 0x00FF0000
☐ Erase SPI part on programming error
☐ Secure SPI flash golden pattern sectors

**Figure 9.3. Lattice Radiant Programmer – Erase Previous Content**

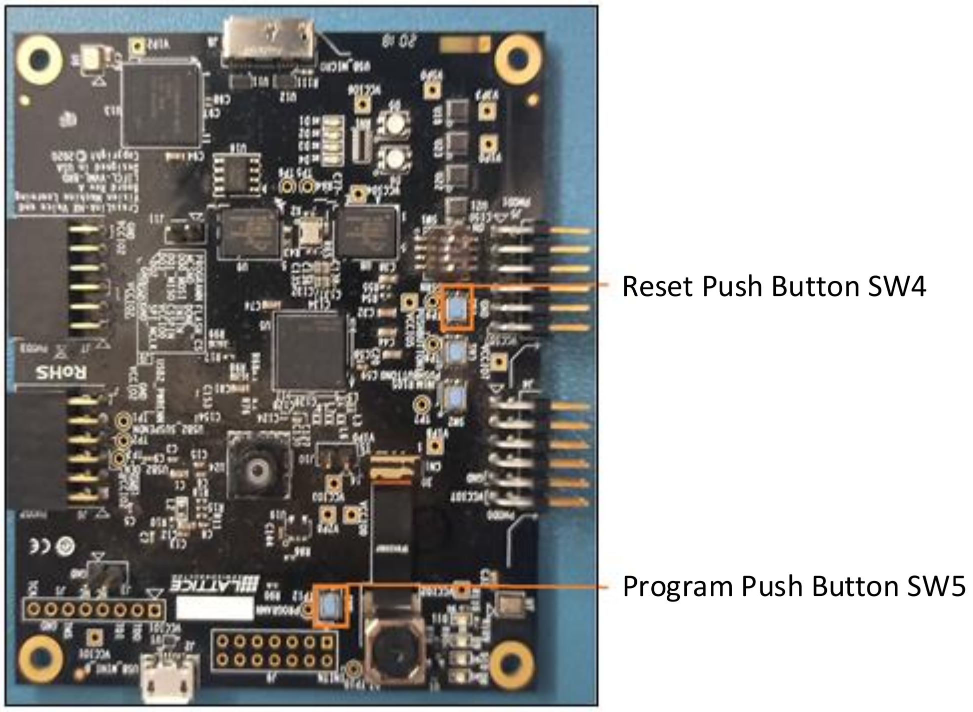


**Figure 9.4. CrossLink-NX VVML Board – SW5 and SW4 Buttons**

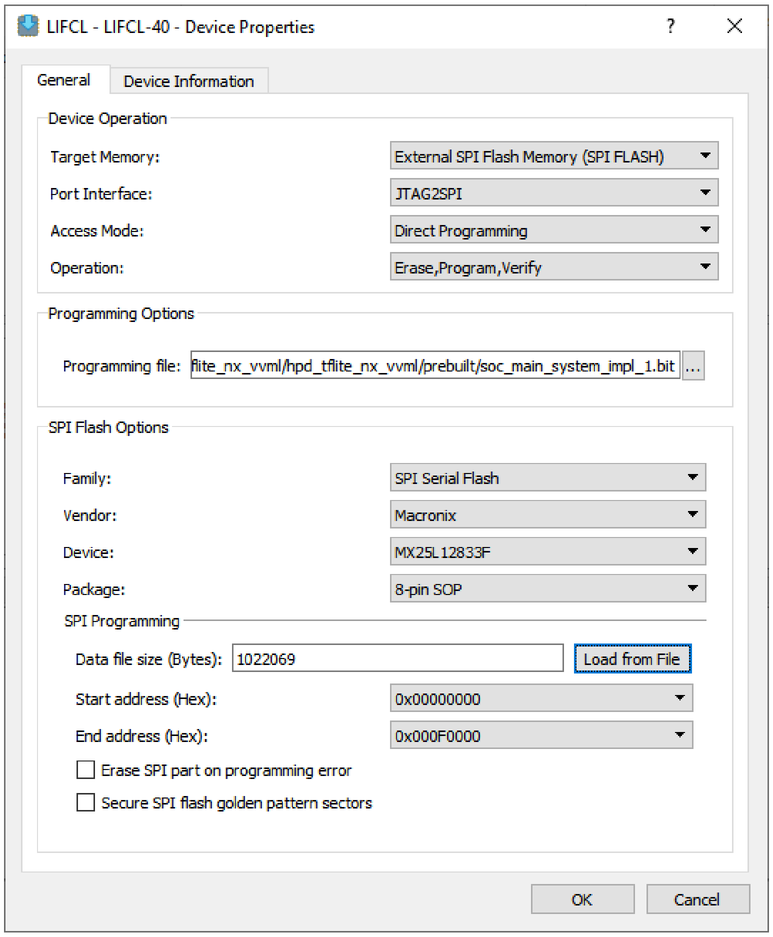
## 9.2. Programming the CrossLink-NX VVML Board

This sections provides the steps to program the generated Bitstream file and the firmware BIN file to VVML board. Ensure that the CrossLink-NX VVML Board is erased before performing the procedure.

### 9.2.1. Programming the Generated Bitstream File

To program the generated bitstream file to the CrossLink-NX VVML Board.

1. In the Lattice Radiant Programmer main interface, right click on Operation and select **Device Properties** to open the **Device Properties** dialog box.
2. Select the options as shown in Figure 9.5.
3. For Programming File, browse and select the generated Bitstream file **soc_main_system_impl_1.bit** from "impl1" folder.


LIFCL - LIFCL-40 - Device Properties

General | Device Information

Device Operation
Target Memory: External SPI Flash Memory (SPI FLASH)
Port Interface: JTAG2SPI
Access Mode: Direct Programming
Operation: Erase,Program,Verify

Programming Options
Programming file: flite_nx_vvml/hpd_tflite_nx_vvml/prebuilt/soc_main_system_impl_1.bit

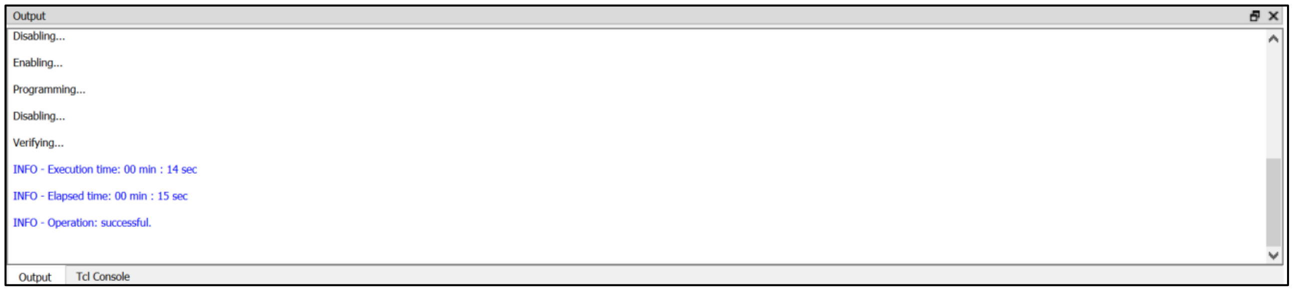
SPI Flash Options
Family: SPI Serial Flash
Vendor: Macronix
Device: MX25L12833F
Package: 8-pin SOP

SPI Programming
Data file size (Bytes): 1022069
Load from File
Start address (Hex): 0x00000000
End address (Hex): 0x000F0000
☐ Erase SPI part on programming error
☐ Secure SPI flash golden pattern sectors

OK Cancel

**Figure 9.5. Lattice Radiant Programmer – Device Properties to Flash Bit File**

4. Click **Load from File** to update the Data file size (Bytes) value.
5. Ensure that the following addresses are correct and Click **OK**.
   - Start Address (Hex): 0x00000000
   - End Address (Hex): 0x000F0000
6. Press the SW5 button on the board and click the Program button to start the Bit Flash operation .
7. Release the button after the **Operation Successful** message is displayed on the Lattice Radiant log window as shown in Figure 9.6.


Output

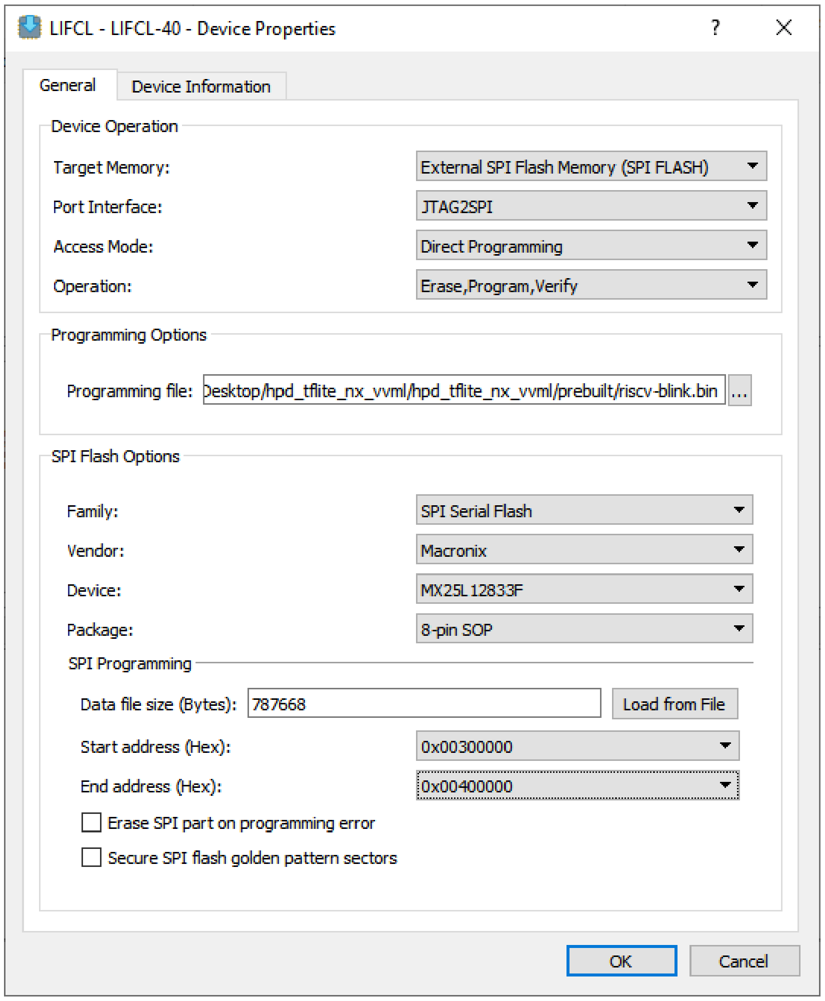
```
Disabling...
Enabling...
Programming...
Disabling...
Verifying...
INFO - Execution time: 00 min : 14 sec
INFO - Elapsed time: 00 min : 15 sec
INFO - Operation: successful.
```

**Figure 9.6. Lattice Radiant Programmer – Output Console**

### 9.2.2. Programming the sensAI Firmware BIN File

To program the sensAI firmware BIN file to the CrossLink-NX VVML board:

1. In the Lattice Radiant Programmer main interface, right click on **Operation** and select **Device Properties** to open the **Device Properties** dialog box.
2. Select the options as shown in Figure 9.7.
3. In **Programming File**, browse and select the firmware BIN file **riscv-blink.bin** from the *prebuilt* folder.


LIFCL - LIFCL-40 - Device Properties

Device Operation

Target Memory: External SPI Flash Memory (SPI FLASH)
Port Interface: JTAG2SPI
Access Mode: Direct Programming
Operation: Erase,Program,Verify

Programming Options

Programming file: Desktop/hpd_tflite_nx_vvml/hpd_tflite_nx_vvml/prebuilt/riscv-blink.bin

SPI Flash Options

Family: SPI Serial Flash
Vendor: Macronix
Device: MX25L12833F
Package: 8-pin SOP

SPI Programming

Data file size (Bytes): 787668
Start address (Hex): 0x00300000
End address (Hex): 0x00400000
☐ Erase SPI part on programming error
☐ Secure SPI flash golden pattern sectors

**Figure 9.7. Lattice Radiant Programmer – Device Properties to Flash Bin File**

**Note:** Do not click on Load from File option.

4. Ensure that the following addresses are correct and click **OK**.
    - Start Address (Hex): 0x00300000
    - End Address (Hex): 0x00400000
5. Press the SW5 push button on board and click the Program button to start the Bin Flash operation .
6. Release the button after the Operation Successful message is seen on the Lattice Radiant log window as shown in Figure 9.6.

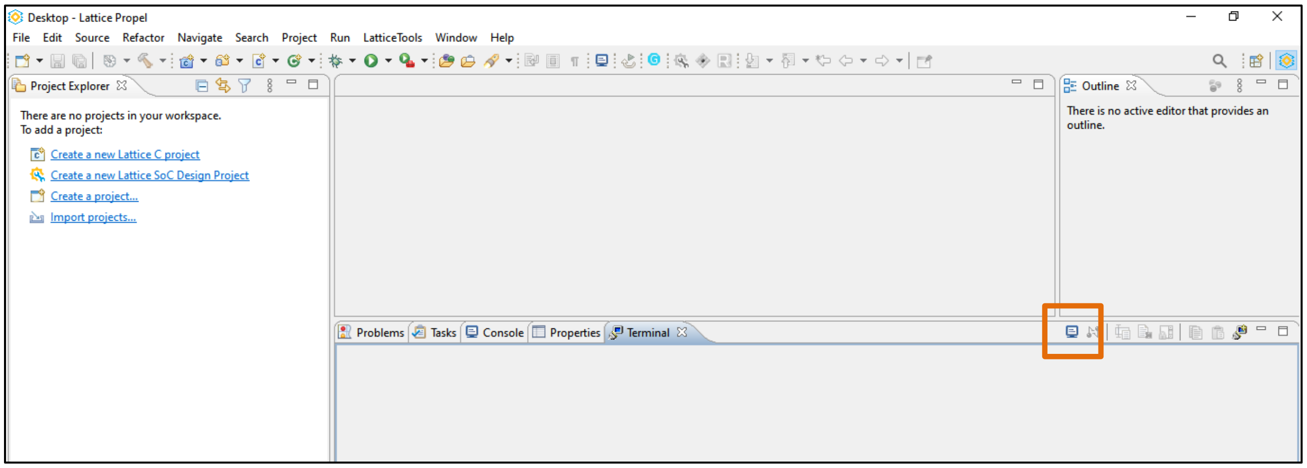
**Note**: After programming the Bit file and Bin file, refer to the Observing UART Output on Lattice Propel SDK section to observe the UART output. If nothing is printed in the UART terminal, erase the old contents loaded to the board as detailed in the Programming the Generated Bitstream File section. This time, select **Erase,Program,Verify Quad 1** under **Device Operation**. This option enables the SPI on the board and UART outputs become visible. This option should be used only once if the UART output is unavailable. Continue with the rest of the steps to load the Bit file.

# 10. Observing UART Output on Lattice Propel SDK

This section provides the steps to observe human presence output in the UART Terminal using the Lattice Propel SDK Software as shown in Figure 10.1.

To observe human presence output:

1. After the Bit and the Bin files are programmed to the CrossLink-NX VVML Board, power OFF the board.
2. Open Lattice Propel SDK version 2.0 Software tool.
3. The Lattice Propel Launcher window opens and prompts for the Workspace location. Launch the tool.
4. Click the **Terminal** tab as shown in Figure 10.1.
5. If the Terminal window is not visible after opening Lattice Propel SDK Tool, you can open the Terminal from the Toolbar by selecting **Window > Show view > other > Terminal (folder) > Terminal**.


Desktop - Lattice Propel

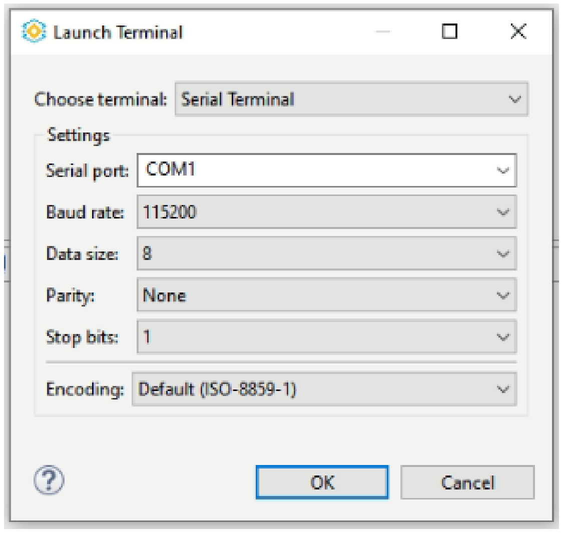
There are no projects in your workspace.
To add a project:

Create a new Lattice C project
Create a new Lattice SoC Design Project
Create a project...
Import projects...

There is no active editor that provides an outline.

Figure 10.1. Opening a Terminal in Propel SDK

6. Click the **Launch Terminal** button to open a Terminal.
7. The **Launch Terminal** dialog box is displayed.
8. Power ON the board by connecting to the system
9. Select the settings in the Terminal Dialog Box as shown in Figure 10.2. Always select the highest numbered COM Port available in Serial Port option. For example if COM1, COM6, COM7 are visible, select COM7.
10. Click **OK**.
11. Reset the Board using the SW4 button and observe the UART values printed in the COM7 space.



Choose terminal: Serial Terminal

Settings

Serial port: COM1

Baud rate: 115200

Data size: 8

Parity: None

Stop bits: 1

Encoding: Default (ISO-8859-1)

**Figure 10.2. Lattice Propel Launch UART Terminal**

## References

- Lattice-Propel-Builder-2.0-User-Guide
- Lattice-Propel-SDK-2.0-User-Guide
- For complete information on Lattice Radiant Project-Based Environment, Design Flow, Implementation Flow, Tasks, and Simulation Flow, see the Lattice Radiant Software 2.2 User Guide.
- For more information on the CrossLink NX Voice and Vision Machine Learning FPGA Board, visit CrossLink-NX Voice and Vision Machine Learning Board.

## Technical Support Assistance

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

## Revision History

### Revision 1.0, June 2021

| Section | Change Summary |
|---|---|
| All | Initial release. |