Object Counting Using CNN Accelerator IP

Reference Design

FPGA-RD-02058-1.1

December 2020

Disclaimers

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

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
CKPT	Checkpoint
CNN	Convolutional Neural Network
EVDK	Embedded Vision Development Kit
FPGA	Field-Programmable Gate Array
ML	Machine Learning
MLE	Machine Learning Engine
SPI	Serial Peripheral Interface
VIP	Video Interface Platform

1. Introduction

This document describes the Human Counting Design process using the ECP5™ EVDK FPGA platform. Human Counting is a subset of the generic Object Counting base design.

1.1. Design Process Overview

The design process involves the following steps:

1.1.1. Training Model

- Setting up the basic environment
- Preparing the dataset
  - Preparing 224 x 224 Image
  - Labeling dataset of human bounding box
- Training the machine
  - Training the machine and creating the checkpoint data
- Creating Frozen file (*.pb)

1.1.2. Neural Network Compiler

- Creating Binary file with Lattice sensAI™ 2.0 program

1.1.3. FPGA Design

- Creating FPGA Bitstream file

1.1.4. FPGA Bitstream and Quantized Weights and Instructions

- Flashing Binary and Bitstream files
  - Binary File to MicroSD
  - Bitstream to Flash Memory on VIP Board

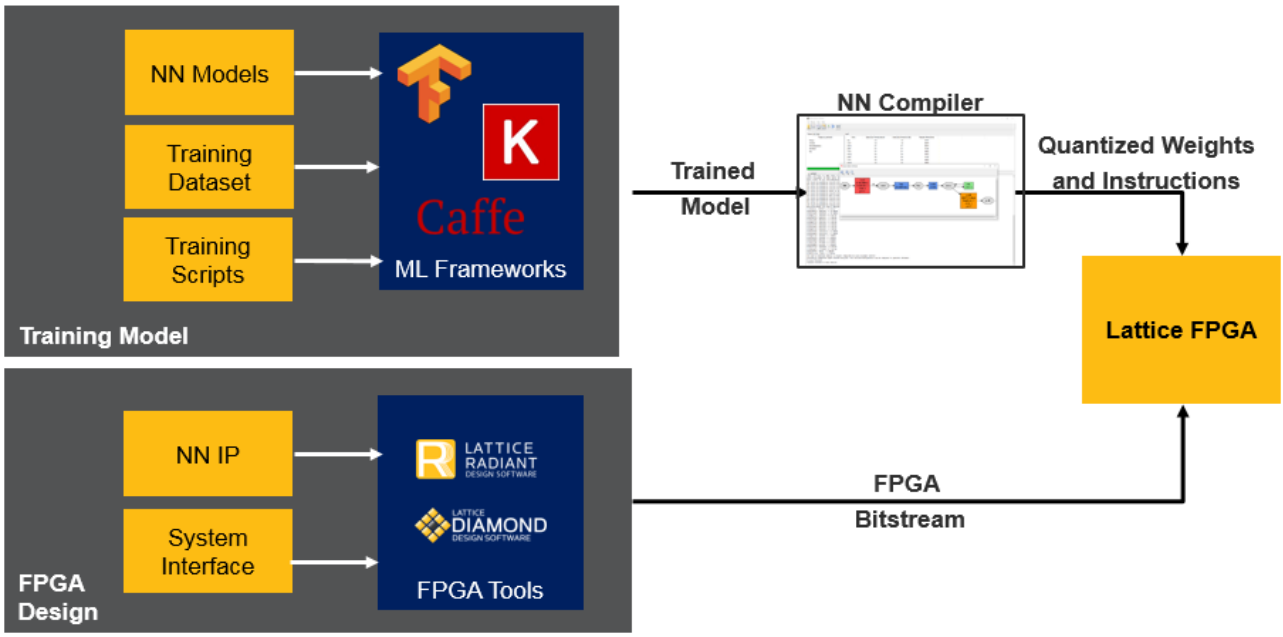

Figure 1.1 Lattice Machine Learning design Flow

2. Setting Up the Basic Environment

2.1. Tools and Hardware Requirements

This section describes the required tools and environment setup for FPGA Bitstream and Flashing.

2.1.1. Lattice Tools

- Lattice Diamond® Tool – Refer to http://www.latticesemi.com/latticediamond.
- Lattice Diamond Programmer – Refer to http://www.latticesemi.com/programmer.
- Lattice sensAI Compiler v2.0 – Refer to https://www.latticesemi.com/Products/DesignSoftwareAndIP/AIML/NeuralNetworkCompiler.

2.1.2. Win32 MicroSD Disk Imager

Refer to https://sourceforge.net/projects/win32diskimager/

2.1.3. Hardware

- ECP5 FPGA VIP Board – Refer to http://www.latticesemi.com/Solutions/Solutions/SolutionsDetails02/VIP.

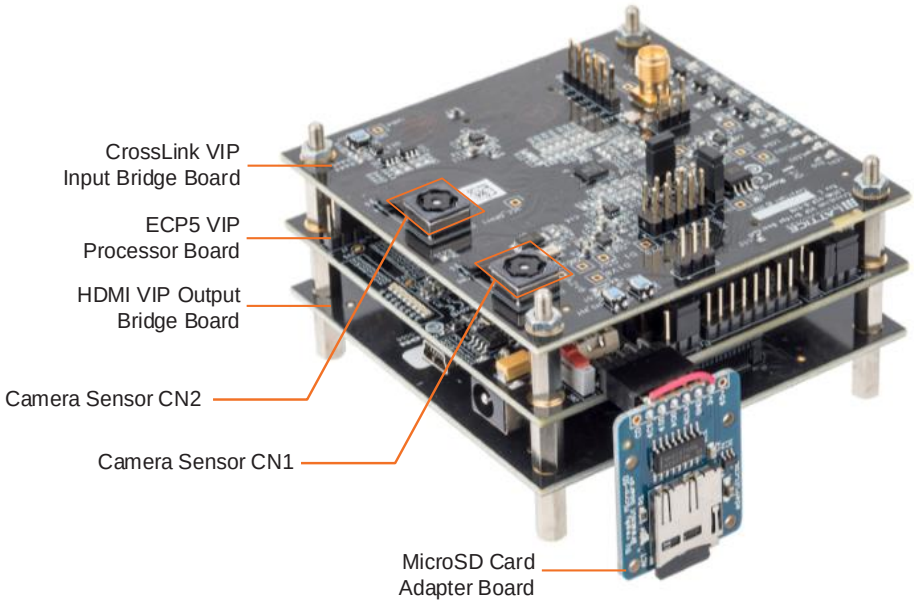

Figure 2.1. Lattice EVDK with MicroSD Card Adapter Board

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. NVIDIA library and TensorFlow version is dependent on PC and 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 CUDA Toolkit

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

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

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

Figure 2.2. Download CUDA Repo

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

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

Figure 2.3. Install CUDA Repo

```
$ sudo apt-key adv --fetch-keys
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.
pub
```

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

Figure 2.4. Fetch Keys

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

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

Figure 2.5. Update Ubuntu Packages Repositories

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

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

Figure 2.6. CUDA Installation

2.2.1.2. Installing the cuDNN

To install the cuDNN:

1. Create Nvidia developer account:https://developer.nvidia.com.
2. Download cuDNN lib: 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 below commands to install cuDNN

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

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

Figure 2.7 cuDNN Library Installation

2.2.2. 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 and Python 3:

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

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

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

Figure 2.8 Anaconda Package Download

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

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

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

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

Welcome to Anaconda3 2019.03

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

Figure 2.9 Anaconda Installation

4. Accept the license.

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

Figure 2.10 Accept License Terms

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

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

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

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

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

Figure 2.11 Confirm/Edit Installation Location

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

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

Figure 2.12 Launch/Initialize Anaconda Environment on Installation Completion

2.2.3. Installing the TensorFlow v1.12

To install the TensorFlow v1.12:

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

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

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

Figure 2.13 Anaconda Environment Activation

2. Install the TensorFlow by running the command below:

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

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

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

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

Figure 2.14 TensorFlow Installation

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

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


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

Figure 2.15 TensorFlow Installation Confirmation

Figure 2.16 shows TensorFlow installation is complete.

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

Figure 2.16 TensorFlow Installation Completion

2.2.4. Installing the Python Package

To install the Python package:

1. Install Easydict by running the command below:

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

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

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - easydict
```

Figure 2.17 Easydict Installation

2. Install Joblib by running the command below:

```
$ conda install joblib
```

```
(base) sib:~/kishan$ conda install joblib
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - joblib
```

Figure 2.18 Joblib Installation

3. Install Keras by running the command below:

```
$ conda install keras
```

```
(base) sib:~/kishan$ conda install keras
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - keras
```

Figure 2.19 Keras Installation

4. Install OpenCV by running the command below:

```
$ conda install opencv
```

```
(base) sib:~/kishan$ conda install opencv
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - opencv
```

Figure 2.20 OpenCV Installation

5. Install Pillow by running the command below:

```
$ conda install pillow
```

```
(base) sib:~/kishan$ conda install pillow
Collecting package metadata: done
Solving environment: done

# All requested packages already installed.

(base) sib:~/kishan$ _
```

Figure 2.21 Pillow Installation

3. Preparing the Dataset

This chapter describes how to create a dataset using Google Open Image Dataset as an example.

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, run the commands below:

1. Clone the OIDv4_Toolkit repository:

```
$ 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 code 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:

```
$ 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 the Person class images:

```
$ 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. However, the downloaded dataset is not in exact 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.

Code converts Xmin, Ymin, Xmax, Ymax into x, y, w, h while training as *bounding box rectangle co-ordinates*.

3.2. Visualizing and Tuning/Cleaning Up the Dataset

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

1. Visualize the labelled images.

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

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 *annotate* to KITTI.

```
$ 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

For information on other labelling tools, see Appendix A. Other Labelling Tools.

4. Training the Machine

4.1. Training Code Structure

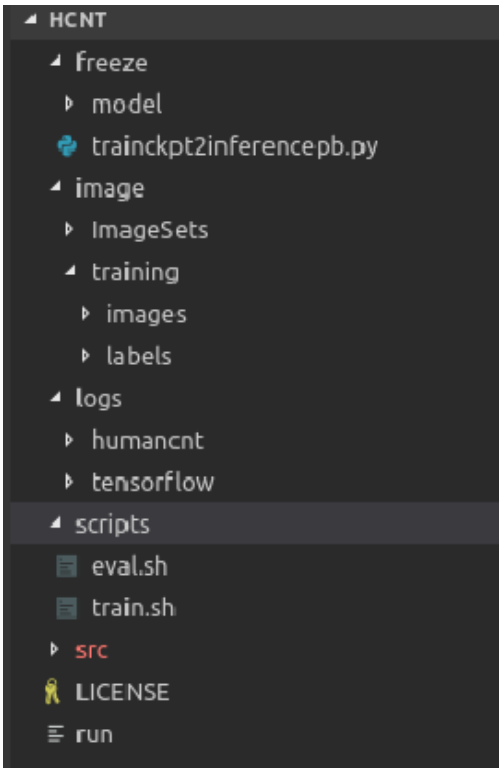

Figure 4.1. Training Code Directory Structure

4.2. 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 224 x 224 input resolution for training.

   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 & <data_path>/training/labels*.

```
class kitti(imdb):
  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
    self._class_to_idx = dict(zip(self.classes, xrange(self.num_classes)))
```

Figure 4.2. Training Code Snippet for Dataset Path

2. Create a train.txt.

```
$ cd image
$ python dataset_create.py
```

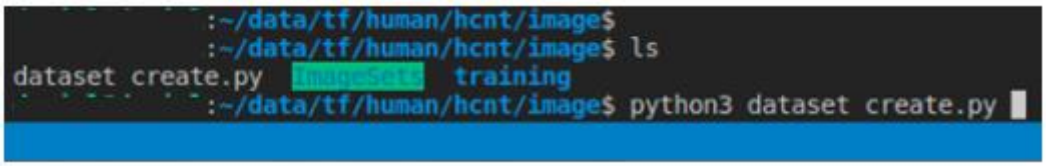

Figure 4.3. Create File for Dataset train.txt

Notes:

- train.txt – file name of dataset images.
- image_set – train (ImageSets/train.txt)
- data_path – $ROOT/image/.
  - Images – $ROOT/image/training/images
  - Annotations – $ROOT/image/training/labels

Figure 4.4 shows the input parameters.

```
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 4.4. Training Input Parameter

- $TRAIN_DATA_DIR – dataset directory path. */data/humancnt* 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 *command* script.

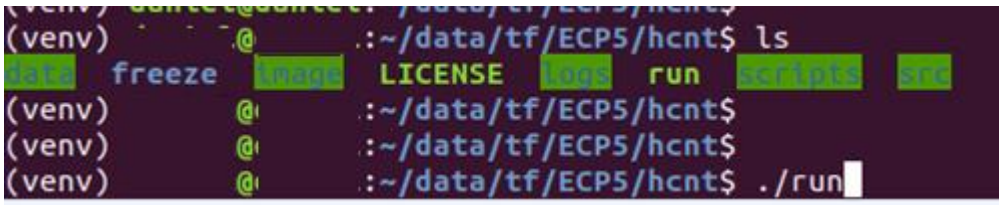

Figure 4.5. Execute Command Script

4. Start TensorBoard by running the command below. For example, *tensorboard –logdir='./logs/'*.

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

5. Open the local host port on your web browser.

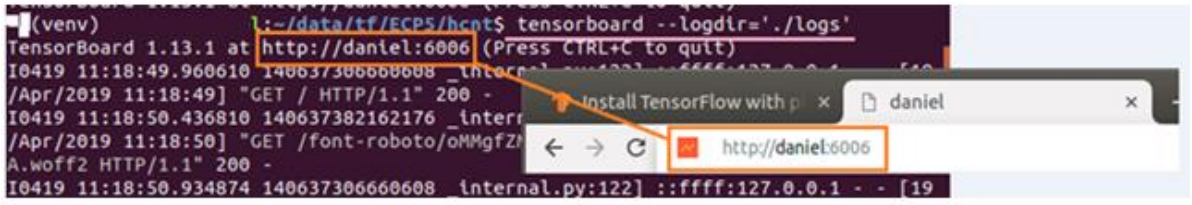

Figure 4.6. TensorBoard – Generated Link

6. Check the training status on Tensorboard.

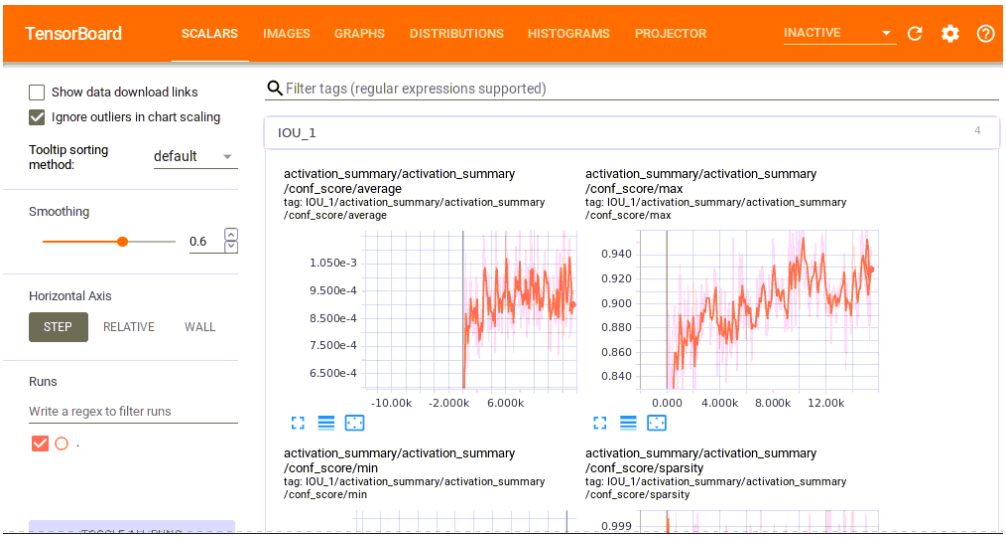

Figure 4.7. TensorBoard

Figure 4.8 shows the image menu of TensorBoard.

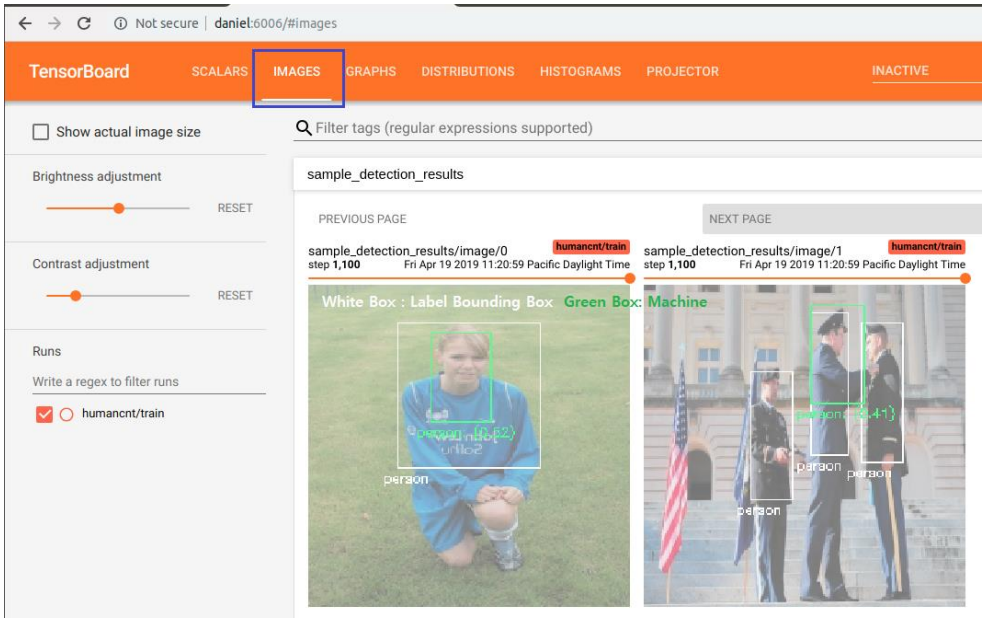

Figure 4.8. Image Menu of TensorBoard

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).

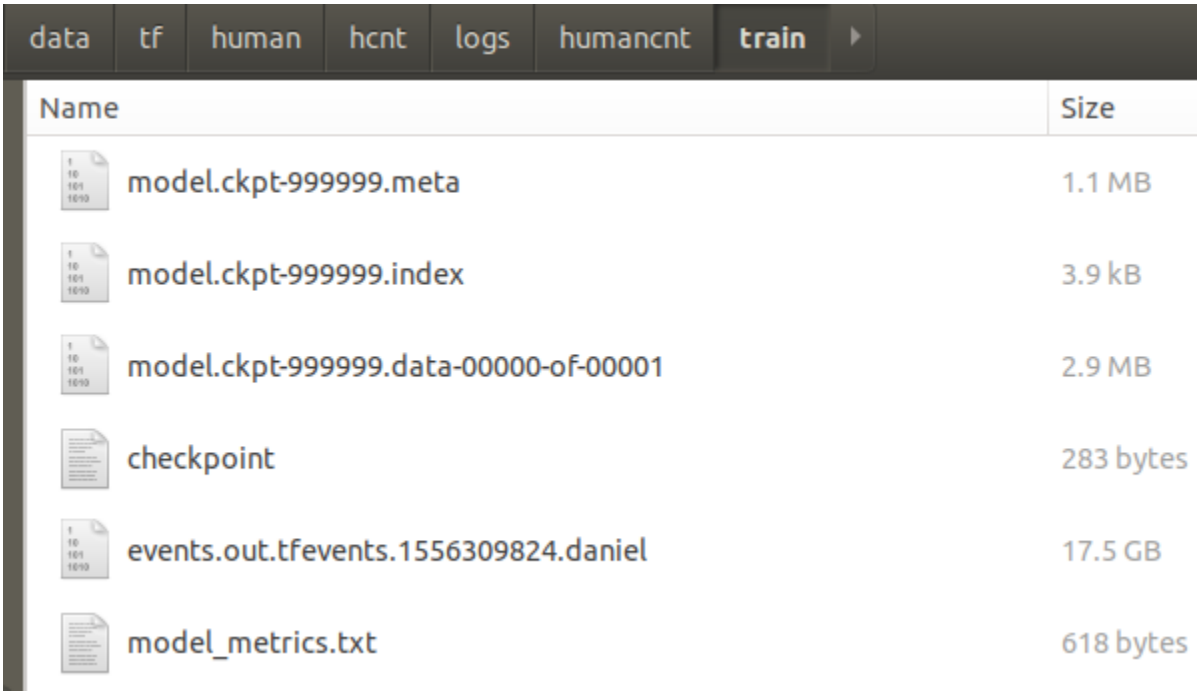

Figure 4.9. Example of Checkpoint Data Files at Log Folder

4.3. Neural Network Architecture

4.3.1. Neural Network Architecture

This section describes the Convolution Network Configuration of Object Counting design

The Neural Network model of Human Counting design used VGG Neural Network base model and the detection layer of SqueezeDet model.

Table 4.1. Convolution Network Configuration of Human Counting Design

Input Image (224 x 224)		
Layer 1	Conv3 – 32	Conv3 - # where: • *Conv3* – 3 x 3 Convolution filter Kernel size • *#* - The number of filter For example, Conv3-32 = 32 3 x 3 convolution filter BN – Batch Normalization
	BN	
	Maxpool	
Layer 2	Conv3 – 32	
	BN	
Layer 3	Conv3 – 32	
	BN	
	Maxpool	
Layer 4	Conv3 – 64	
	BN	
Layer 5	Conv3 – 64	
	BN	
	Maxpool	
Layer 6	Conv3 – 128	
	BN	
Layer 7	Conv3 – 128	
	BN	
	Maxpool	
Convolution	Conv3 - 42	

4.3.2. Training Code of Neural Network Configuration

- File Name – squeezeDet.py
- Neural Network Architecture function – *'_add_forwrad_graph'*

The Neural Network model can be configured by the python training code.

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

fire1 = self._fire_layer('fire1', self.image_input, oc=depth[0], freeze=False, w_bin=fl_w_bin, a_bin=fl_a_bin,               min_rng=min_rng, max_rng=max_rng)
fire2 = self._fire_layer('fire2', fire1,            oc=depth[1], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=False, min_rng=min_rng, max_rng=max_rng)
fire3 = self._fire_layer('fire3', fire2,            oc=depth[2], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,               min_rng=min_rng, max_rng=max_rng)
fire4 = self._fire_layer('fire4', fire3,            oc=depth[3], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=False, min_rng=min_rng, max_rng=max_rng)
fire5 = self._fire_layer('fire5', fire4,            oc=depth[4], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,               min_rng=min_rng, max_rng=max_rng)
fire6 = self._fire_layer('fire6', fire5,            oc=depth[5], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=False, min_rng=min_rng, max_rng=max_rng)
fire7 = self._fire_layer('fire7', fire6,            oc=depth[6], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,               min_rng=min_rng, max_rng=max_rng)
fire_o = fire7

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

num_output = mc.ANCHOR_PER_GRID * (mc.CLASSES + 1 + 4)
self.preds = self._conv_layer('conv12', fire_o, filters=num_output, size=3, stride=1,
    padding='SAME', xavier=False, relu=False, stddev=0.0001, w_bin=sl_w_bin)
print('self.preds:', self.preds)
```

Figure 4.10. Neural Network Configuration Training Code

```
def _fire_layer(self, layer_name, inputs, oc, stddev=0.01, freeze=False, w_bin=16, a_bin=16, pool_en=True, min_rng=-0.5, max_rng=0.5):
  with tf.variable_scope(layer_name):
    ex3x3 = self._conv_layer('conv3x3', inputs, filters=oc, size=3, stride=1,padding='SAME', stddev=stddev, freeze=freeze, relu=False, w_bin=w_bin)
    tf.summary.histogram('before_bn', ex3x3)
    ex3x3 = self._batch_norm('bn', ex3x3)
    tf.summary.histogram('before_relu', ex3x3)
    ex3x3 = self.binary_wrapper(ex3x3, a_bin=a_bin, min_rng=min_rng, max_rng=max_rng)
    tf.summary.histogram('after_relu', ex3x3)

    if pool_en:
        pool = self._pooling_layer('pool', ex3x3, size=2, stride=2, padding='SAME')
    else:
        pool = ex3x3
    tf.summary.histogram('pool', pool)

    return pool
```

Figure 4.11. '_fire_layer' Function

```
def _add_forward_graph(self):
    """NN architecture."""

    mc = self.mc
    bin_k = 1 # K for BNN

    if mc.LOAD_PRETRAINED_MODEL:
      assert tf.gfile.Exists(mc.PRETRAINED_MODEL_PATH), \
          'Cannot find pretrained model at the given path:' \
          '  {}'.format(mc.PRETRAINED_MODEL_PATH)
      self.caffemodel_weight = joblib.load(mc.PRETRAINED_MODEL_PATH)

    depth = [32, 32, 32, 64, 64, 128, 128]
```

Figure 4.12. 'depth' – Convolution Filter Number Definition

5. Creating Frozen File

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

5.1. Generating the pbtxt File

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

```
$ python src/genpb.py –ckpt_dir="<log directory>"
```

For example, *python src/genpb.py –ckpt_dir='./logs/humancnt/train/'*.

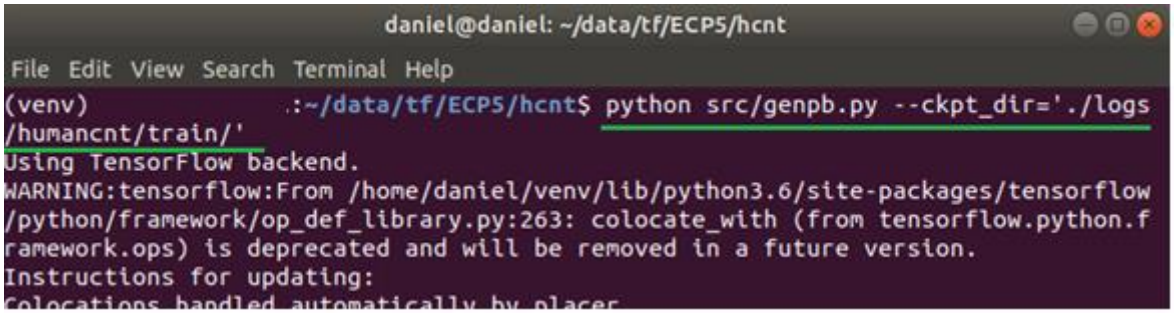

Figure 5.1. pbtxt File Generation from Checkpoint

Figure 5.2 shows the generated pbtxt file.

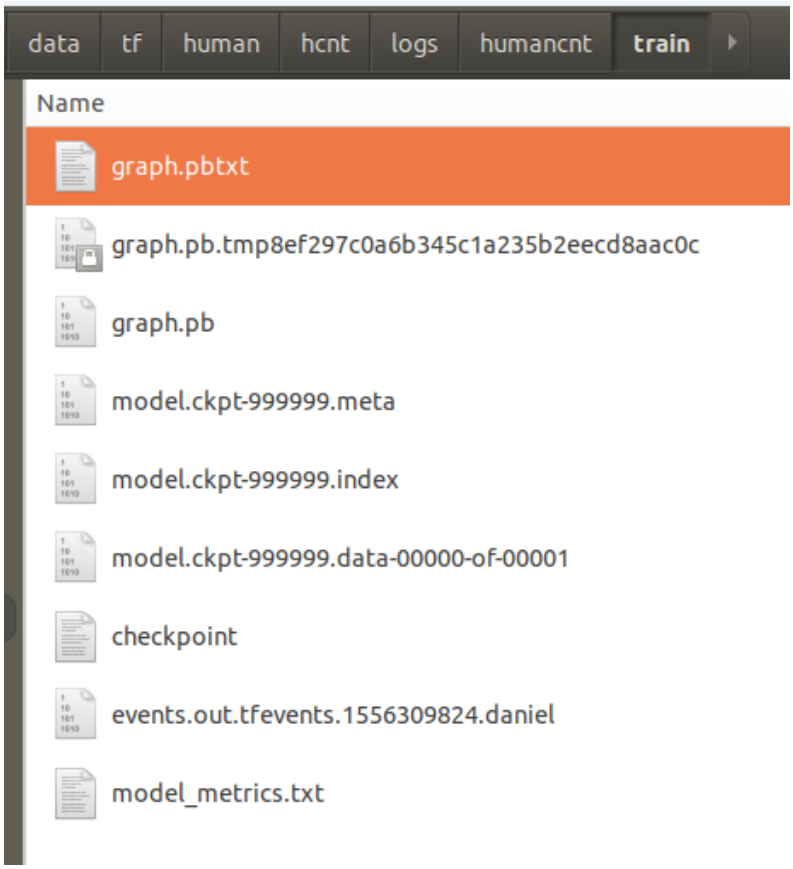

Figure 5.2. pbtxt File Generation from Checkpoint

5.2. Generating the frozen (.pb) File

To generate the frozen file:

1. Go to the root directory where trainckpt2inferencepb.py is located.
2. Rename *.pbtxt to model.pbtxt.
3. Copy all checkpoint data to */freeze/model/*.

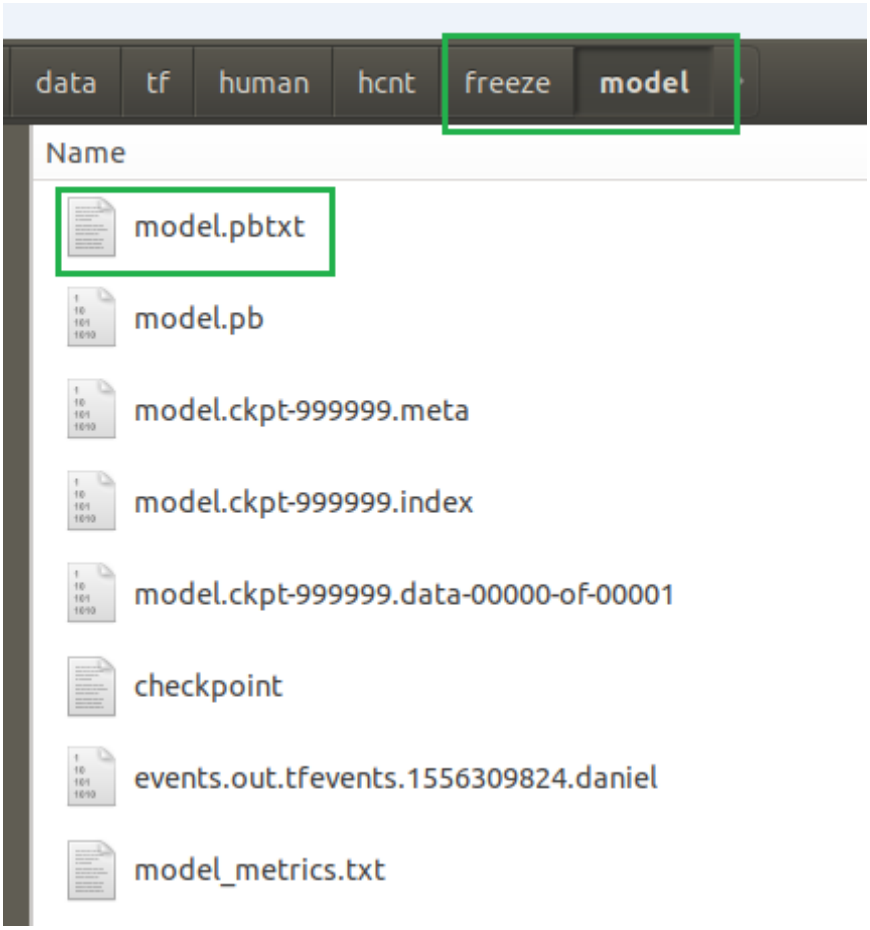

Figure 5.3. Checkpoint and pbtxt Example

4. Check the *indexOfModel* variable value based on the model's index in *dataParas []* in *trainckpt2inferencepb.py* for the demo entry *model*. See the *indexOfModel=3* and fourth entry *model.pbtxt* as reference.

```
indexOfModel=3
# 0: Humandet_ECP5
# 1: HumanGesture
# 2: Humandet_UPlus
# 3: HumanCount_ECP5

dataParas=[
            ['Humandet_ECP5.pbtxt','image_input', 'image_input', 'conv12/bias_add' ,True,[1,224,224,3]],
            ['HumanGesture.pbtxt',      'fifo_queue',  'fifo_queue',  'logit/add' ,    True,[1,32,32,1]] ,
            ['model.pbtxt',      'image_input',  'image_input',  'conv12/convolution' ,    True,[1,64,64,1]],
            ['model.pbtxt','image_input', 'image_input', 'conv12/bias_add' ,False],
           ]
```

Figure 5.4. IndexofModel Setting on trainckpt2inferencepb.py

5. Open a command line console and run the python script file *$ python trainckpt2inferencepb.py*.

```
(venv)                :~/data/tf/ECP5/hcnt/freeze$ ls
model  trainckpt2inferencepb.py
(venv)                :~/data/tf/ECP5/hcnt/freeze$ python trainckpt2inferencepb.py
```

Figure 5.5. Model Freezing Script – Output

The frozen file is created. Figure 5.6 shows an example *model_frozenforInference.pb* file.

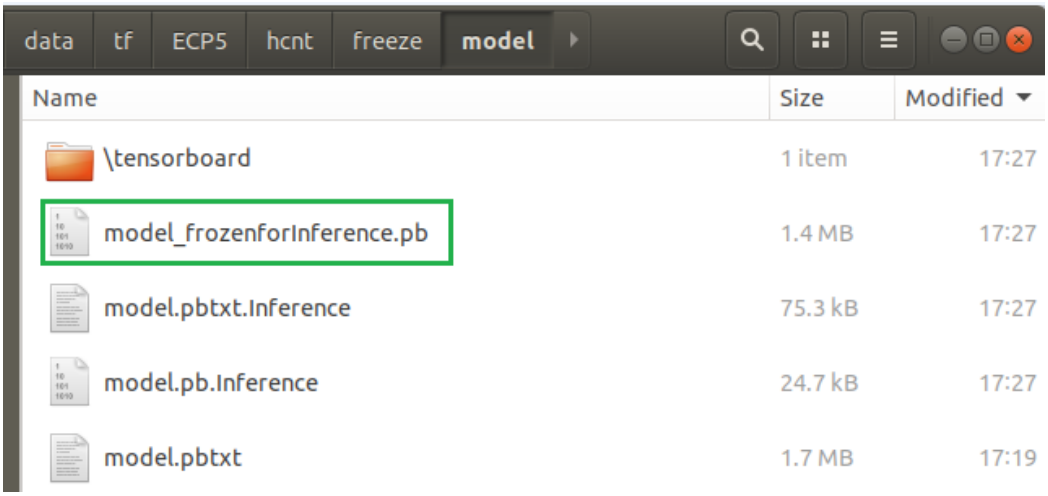

Figure 5.6. Frozen PB File

6. Creating Binary File with Lattice sensAI

This chapter describes how to generate binary file using the Lattice sensAI version 2.0 program.

Figure 6.1. Lattice sensAI Home Screen

To create the project in Lattice sensAI tool:

1. Click **File > New**.
2. Enter the following settings:
   - **Project name**
   - **Framework – TensorFlow**
   - **Class – CNN**
   - **Device – ECP5**
3. Click **Network File** and select the network (PB) file.

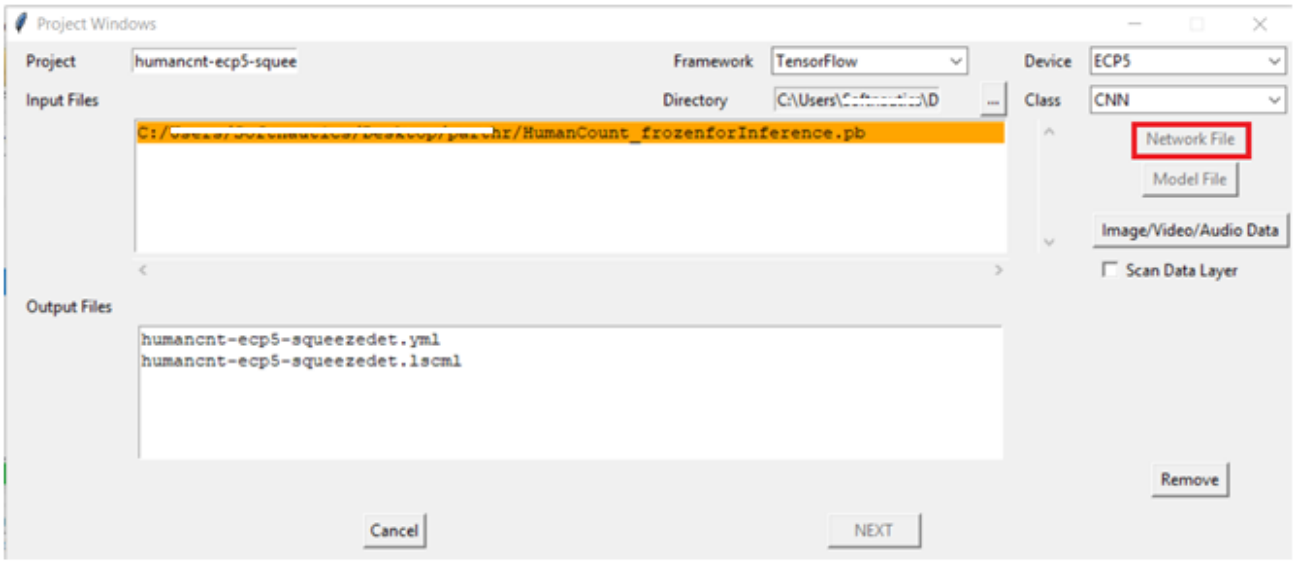

Figure 6.2. Lattice sensAI –Network File Selection

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

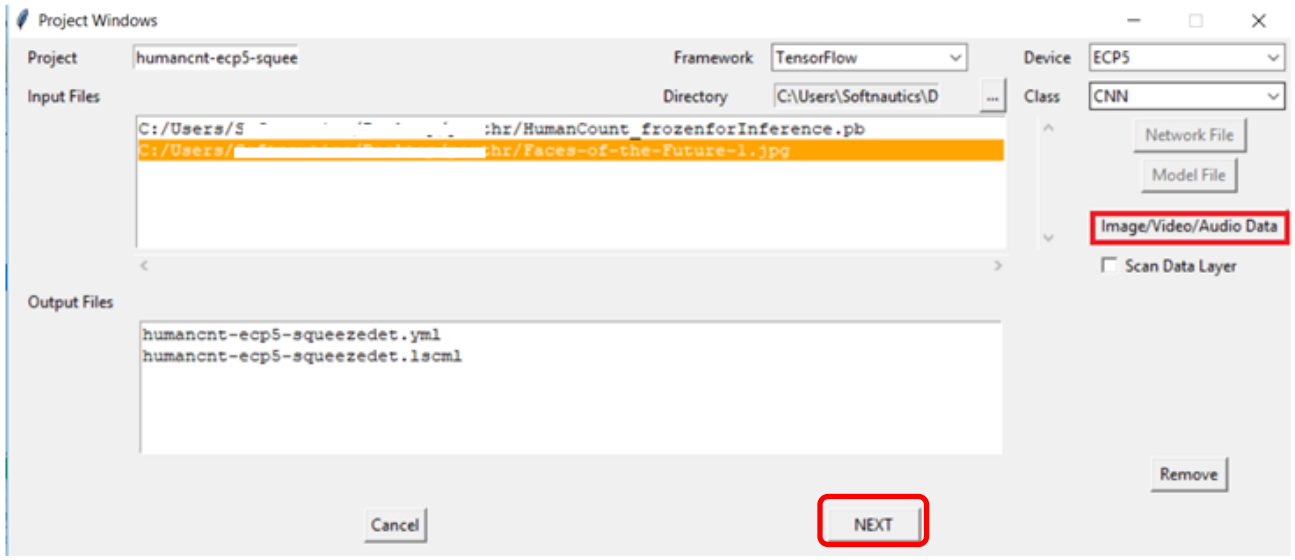

Figure 6.3. Lattice sensAI –Image Data File Selection

5. Click **NEXT**.
6. Configure your project settings.

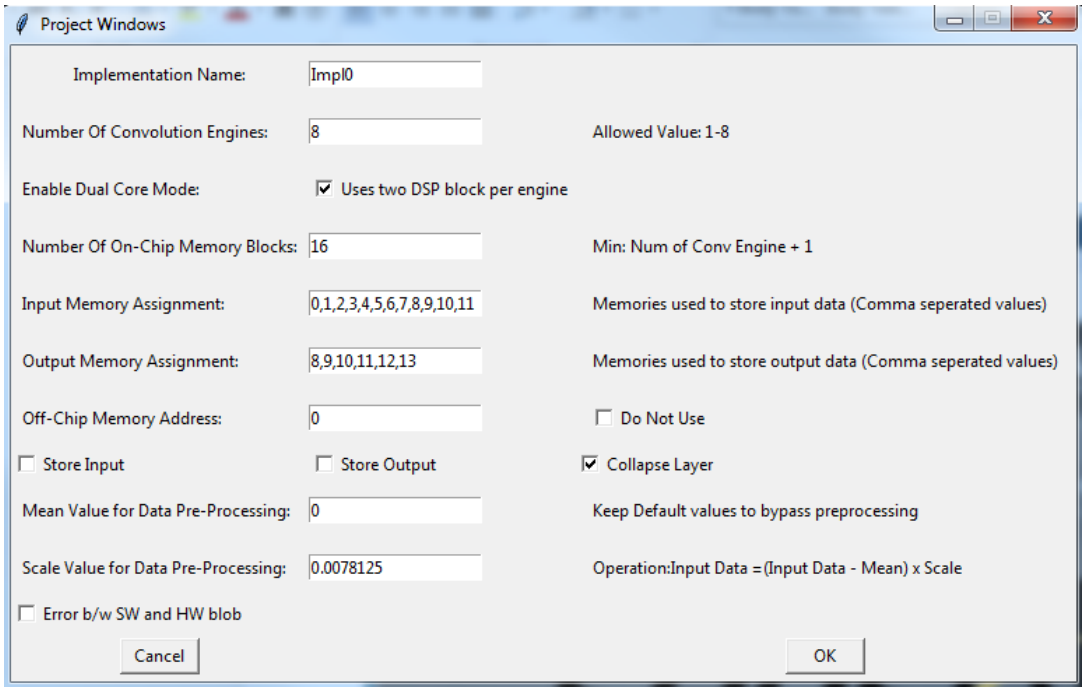

Figure 6.4. Lattice sensAI – Project Settings

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

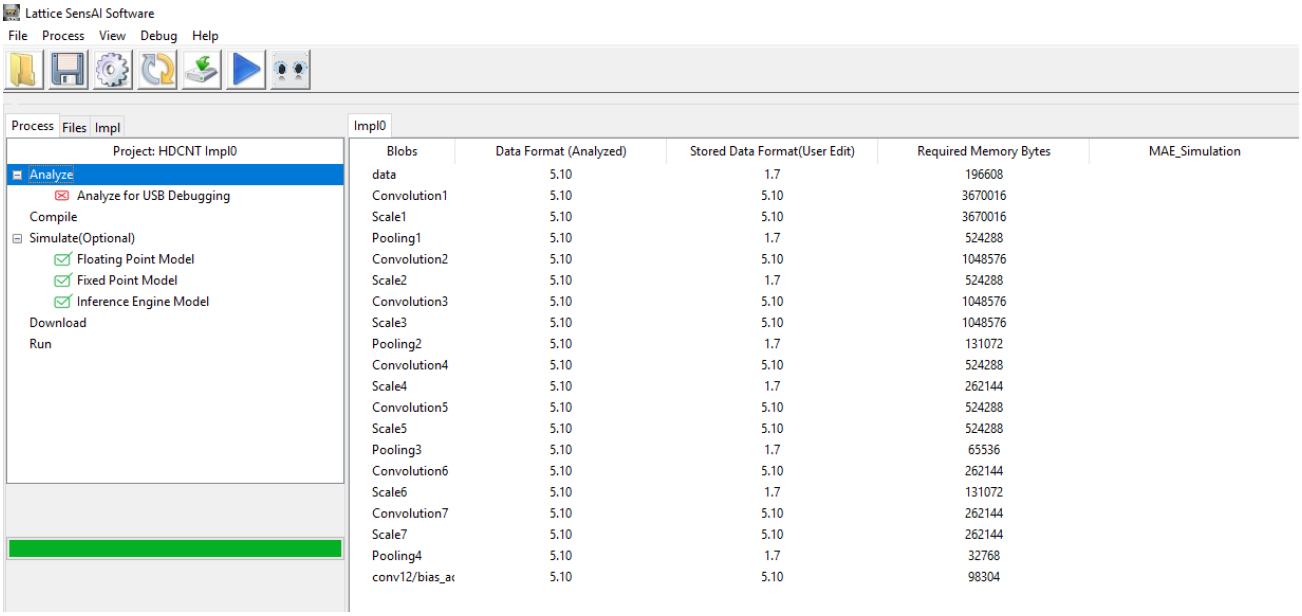

Figure 6.5. Lattice sensAI – Analyze Project

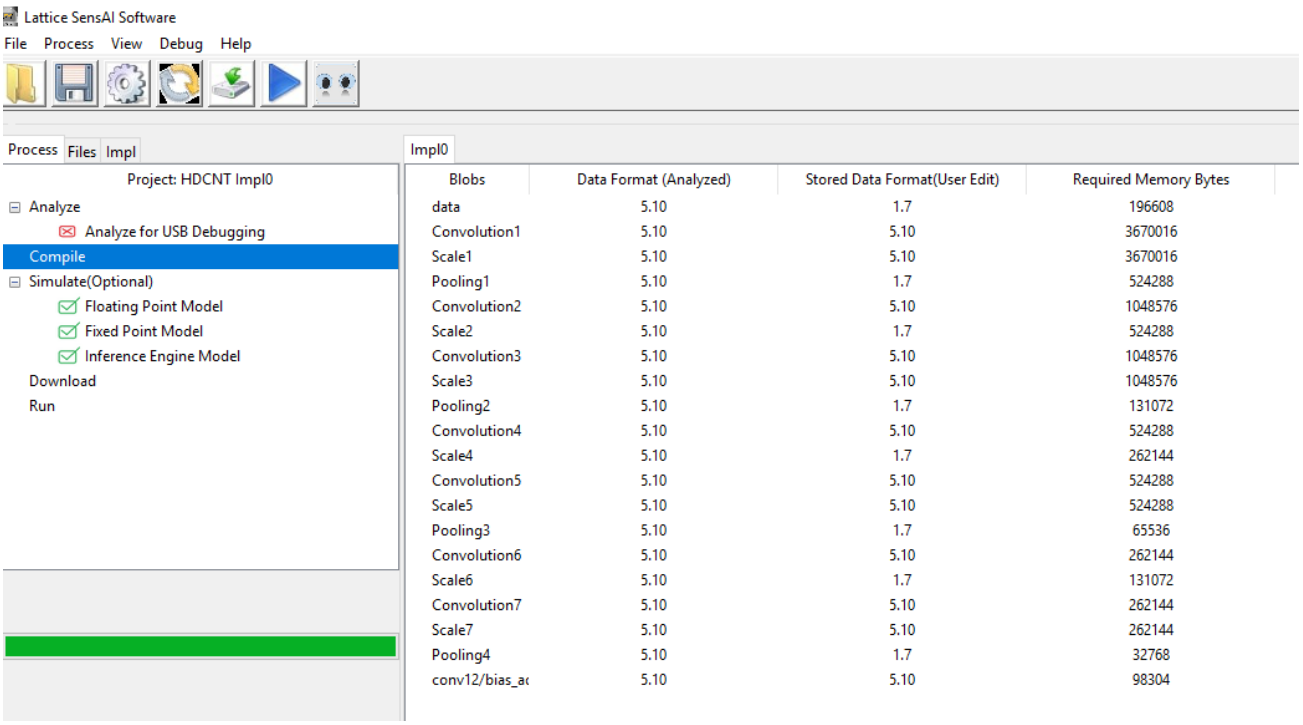

Figure 6.6. Lattice sensAI – Compile Project

7. Creating FPGA Bitstream File

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

To create the FPGA bitstream file:

1. Open Lattice Diamond Software.

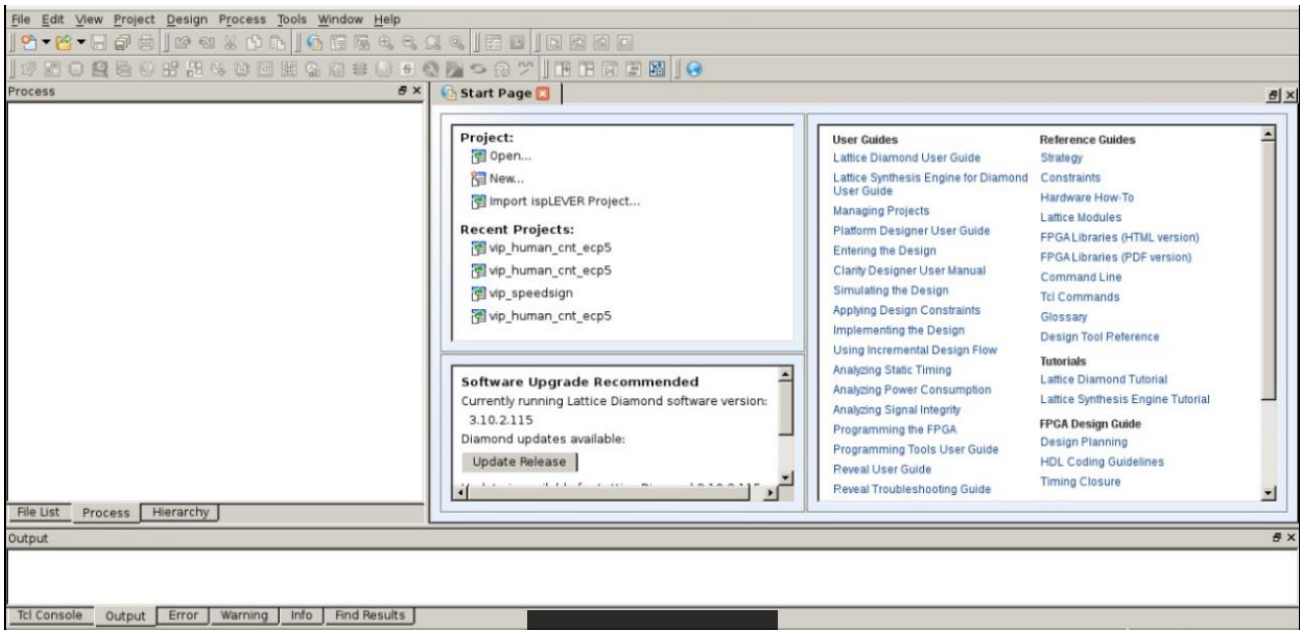

Figure 7.1. Diamond Software

2. Click **File > Open Project**.
3. Open the Diamond project file for ECP5 human count demo RTL.

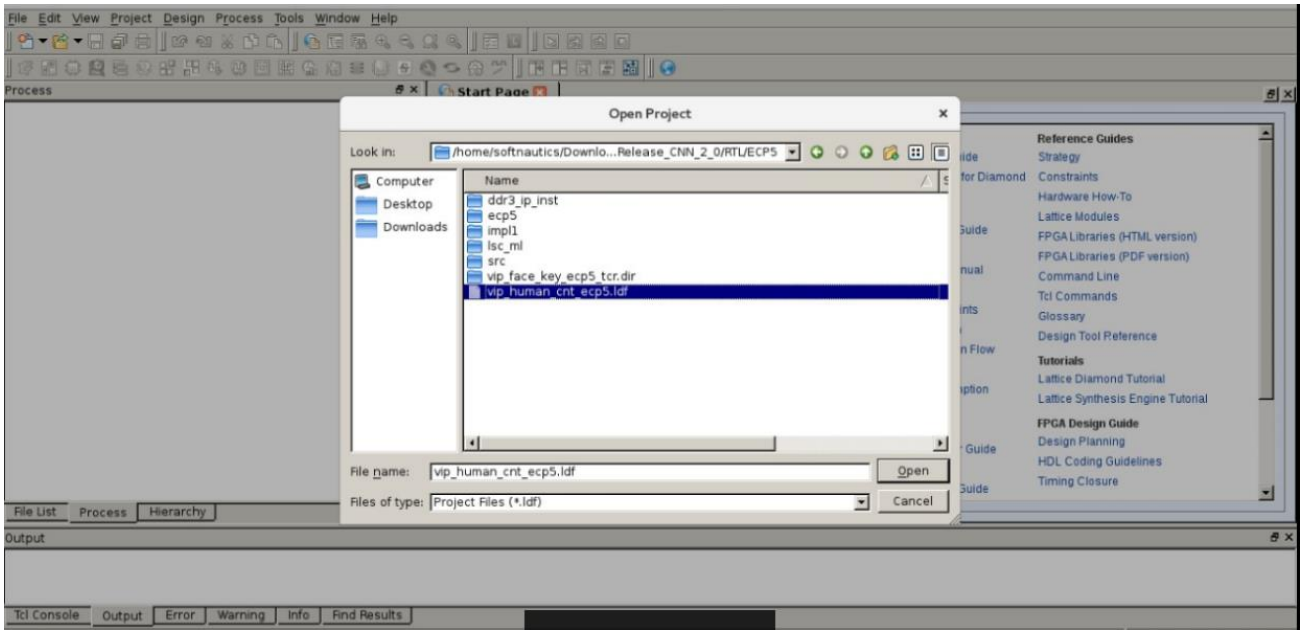

Figure 7.2. Diamond Software – Open Project

4. Click **Bitsream File** to generate the bit file.

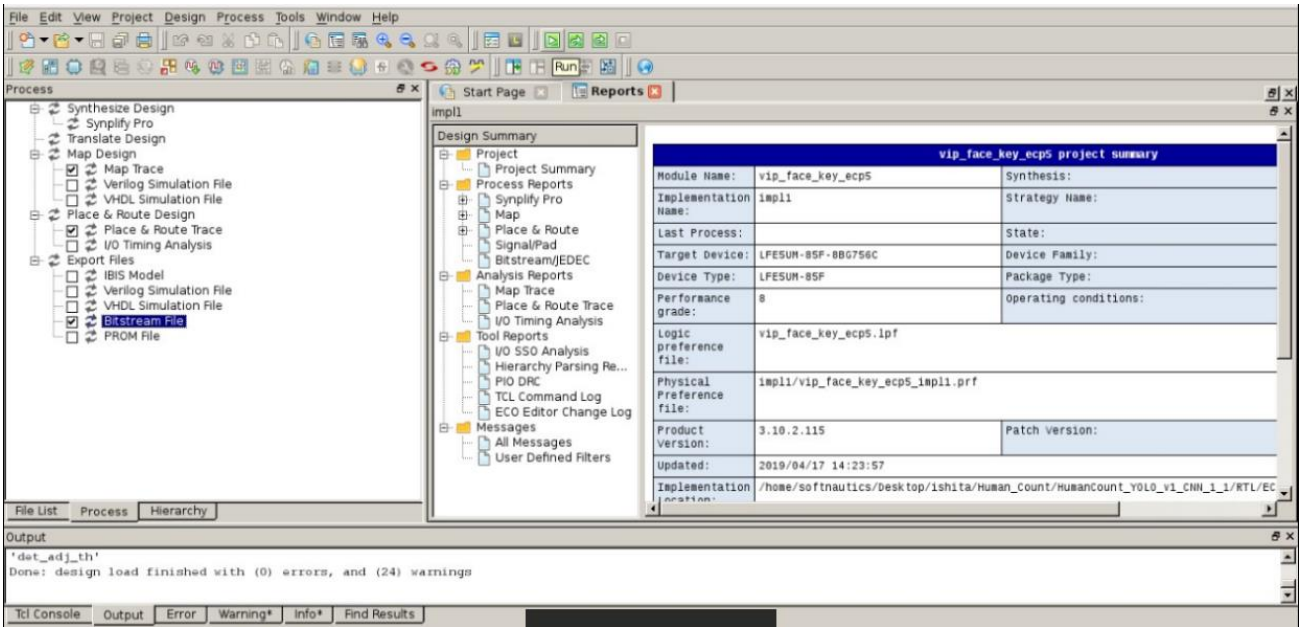

Figure 7.3. Diamond Software – Bitstream Generation

5. View the log message in **Export Reports** that indicates the generated bitstream path:

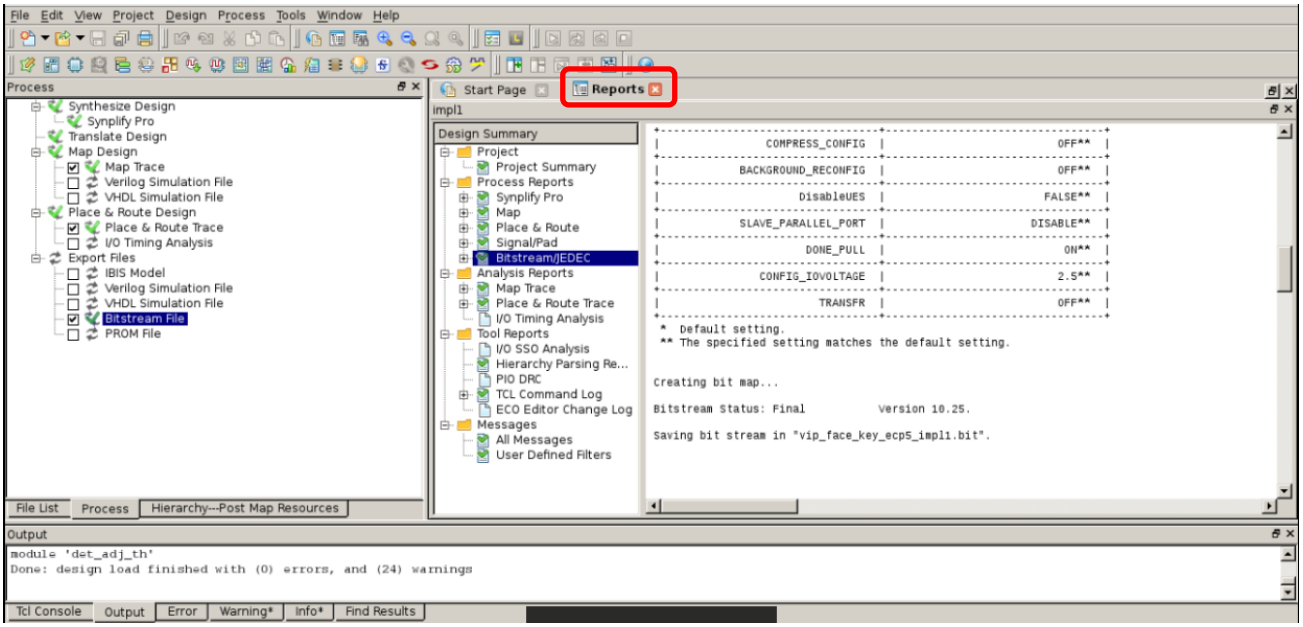

Figure 7.4. Diamond Software – Bitstream Generation Export Report

8. Programming the Binary and Bitstream Files

Both the CrossLink™ VIP Input Bridge Board and the ECP5 VIP Processor Board must be configured and programmed. Also, the demo design firmware must be programmed onto the MicroSD card which is plugged into the MicroSD Card Adaptor Board.

8.1. Programming the CrossLink SPI Flash

8.1.1. Erasing the CrossLink SRAM Prior to Reprogramming

If the CrossLink is already programmed (either directly or loaded from SPI Flash), erase the CrossLink SRAM before reprogramming the CrossLink SPI Flash. Keep the board powered on to prevent reloading on reboot.

To erase CrossLink:

1. Launch Diamond Programmer with **Create a new blank project**.
2. Select **LIFMD** for **Device Family** and **LIF-MD6000** for **Device**.

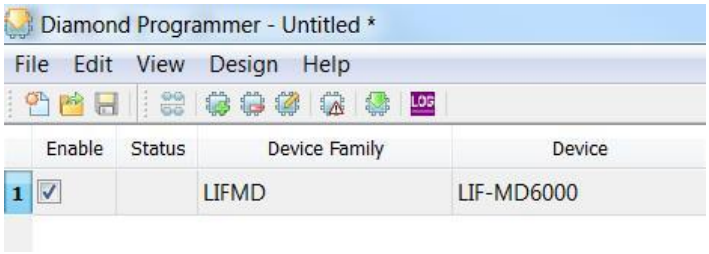

Figure 8.1. Select Device

3. Right-click and select **Device Properties.**
4. Select SSPI SRAM Programming for Access Mode and Erase Only for Operation.

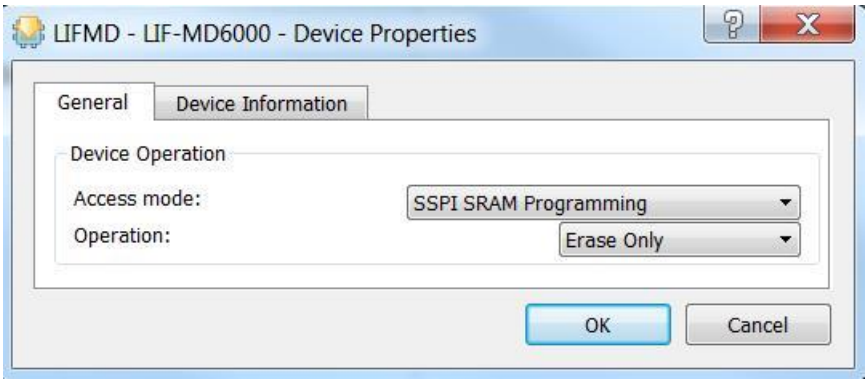

Figure 8.2. Device Operation

5. Click **OK** to close the Device Properties window.
6. Click the **Program** button  in Diamond Programmer to start the Erase sequence.

8.1.2. Programming the SPI on the CrossLink VIP Input Bridge Board

To program the SPI on the CrossLink VIP Inout Bridge Board:

1. Ensure the CrossLink device is erased by performing Steps 1-6.
2. Right-click and select **Device Properties**.
3. Select **SPI Flash Programming** for **Access mode** and make the following selections:
   a. For **Programming File**, browse and select the **CrossLink bitfile** (*.bit).
   b. For **SPI Flash Options**, refer to Table 8.1.

Table 8.1. SPI Flash Options Selection Guide

Item	Rev A/B	Rev C – Option 1	Rev C – Option 2	Rev C – Option 3
Family	SPI Serial Flash	SPI Serial Flash	SPI Serial Flash	SPI Serial Flash
Vendor	Micron / ST Micro	Micron	Micron / ST Micro	Numonyx / Micron
Device	SPI-M25PX16	MT25QL128	SPI-M25PX16	M25P128
Comment	Marked with ST-Micro Logo	Marked with Micron Logo	Marked with ST-Micro Logo	Marked with Lot Code Only – No Vendor or Part Number Marking

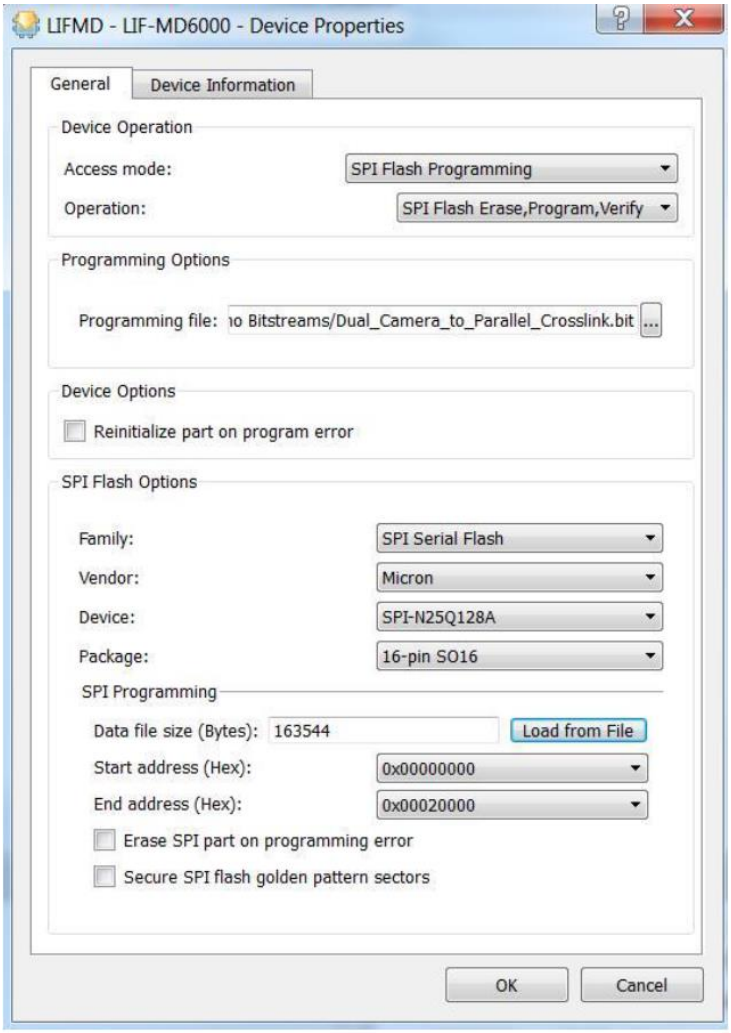

Figure 8.3. Device Properties

4. Click **OK** to close the **Device Properties** window.
5. Click the **Program** button in Diamond Programmer to start the programming sequence.
6. After successful programming, the **Output** console displays the results as shown in Figure 8.4.

Figure 8.4. Output Console

8.2. Programming the ECP5 VIP Processor Board

8.2.1. Erasing the ECP5 Prior to Reprogramming

If the ECP5 and CrossLink VIP Processor Boards are already configured and programmed, erase first the ECP5 SRAM memory, then program the ECP5's SPI Flash in the next section.The demo design firmware must also be programmed onto the MicroSD card which is plugged into the MicroSD Card Adaptor Board.

Keep the board powered when re-programming the SPI Flash in the next section.

To erase the ECP5:

1. Launch Diamond Programmer with **Create a new blank project.**
2. Select **ECP5UM** for **Device Family** and **LFE5UM-85F** for **Device**.

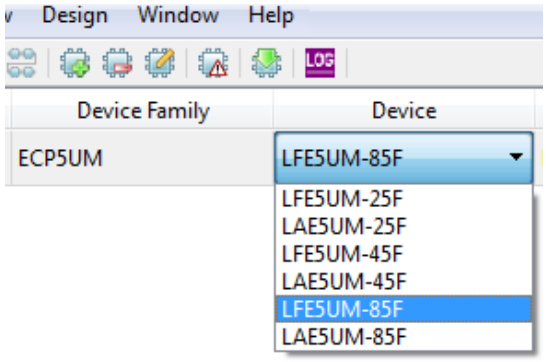

Figure 8.5. Selecting Device

3. Right-click and select **Device Properties**.
4. Select **JTAG 1532 Mode** for **Access Mode** and **Erase Only** for **Operation**.

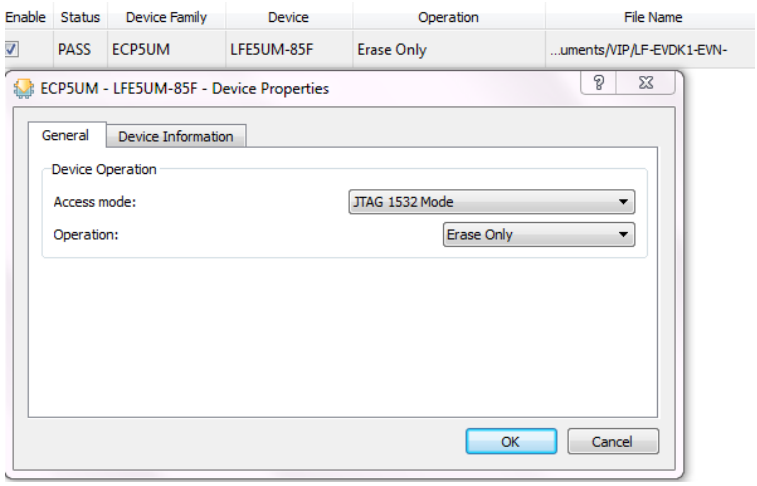

Figure 8.6. Device Operation

5. Click **OK** to close the Device Properties window.
6. Click the **Program** button in Diamond Programmer to start the Erase sequence.

8.2.2. Programming the SPI on the ECP5 VIP Processor Board

To program the SPI:

1. Ensure the ECP5 device is erased by performing Steps 1-6.
2. Right-click and select **Device Properties**.
3. Select **SPI Flash Background Programming** for **Access mode** and make the following selections:
   a. For **Programming File**, browse and select the Human Count Demo bitfile (*.bit).
   b. For **SPI Flash Options**, refer Table 8.2.

Table 8.2. SPI Flash Options Selection Guide

Item	Rev B	Rev C
Family	SPI Serial Flash	SPI Serial Flash
Vendor	Micron	Macronix
Device	SPI-N25Q128A	MX25L12835F
Package	8-pin SO8	8-Land WSON

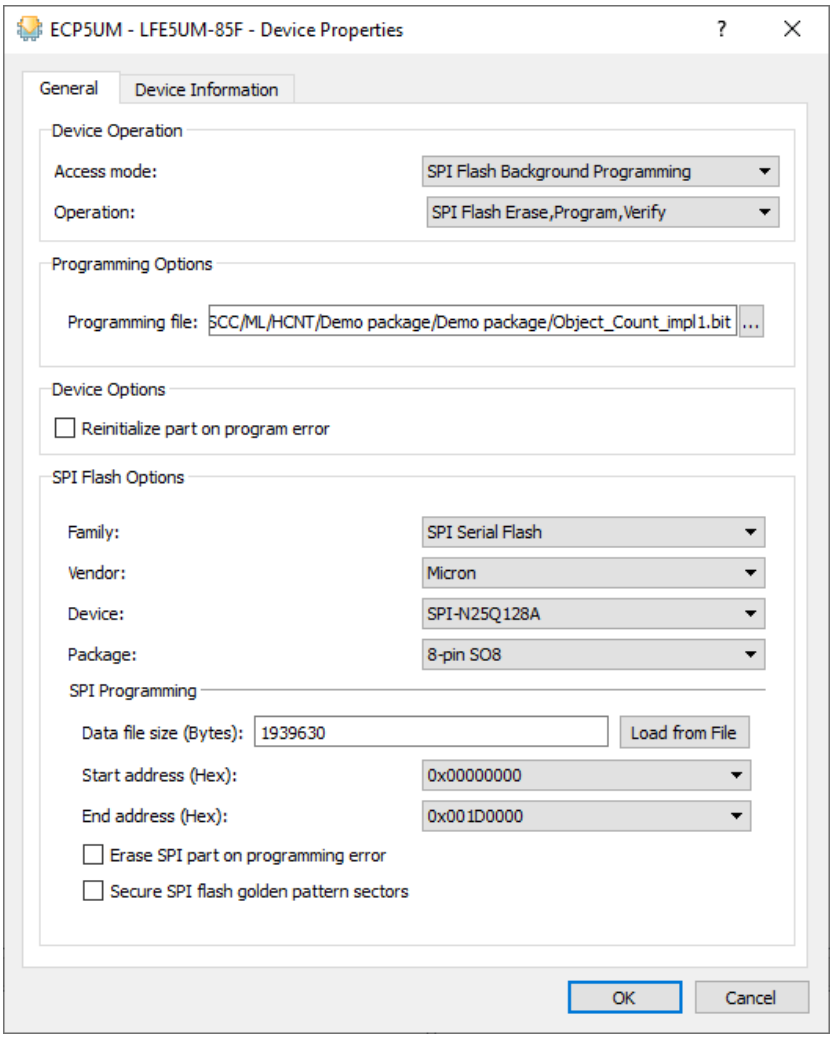

Figure 8.7 Device Properties

4. Click **OK** to close the **Device Properties** window.
5. Click the **Program** button  in Diamond Programmer to start the programming sequence.
6. After successful programming, the **Output** console displays the results as shown in Figure 8.8.

Figure 8.8 Output Console

8.2.3. Programming the MicroSD Card Firmware

To write the image to the MicroSD Card:

1. Download and install the Win32diskimager Image Writer software from the following link: https://sourceforge.net/projects/win32diskimager/.
2. Use Win32diskimager to write the appropriate Flash image file to the SD memory card. Depending on your PC, you may need a separate adapter (not described in this document) to physically connect to the card.
3. In Win32 Disk Imager, select the **Image File** and **Card Reader** as shown in Figure 8.9

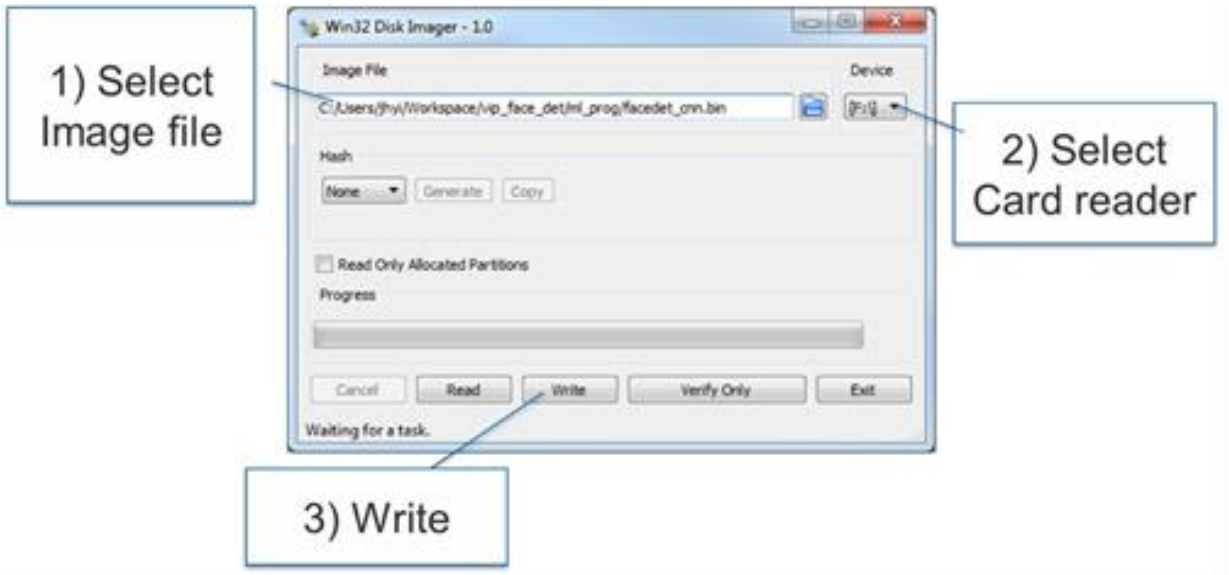

Figure 8.9 Win32 Disk Imager

4. Click **Write**.

Appendix A. Other Labelling Tools

Table A.1 provides information on other labelling tools.

Table A.1. Other Labelling Tools

Software	Platform	License	Reference	Converts To	Notes
annotate-to-KITTI	Ubuntu/Windows (Python based utility)	No License (Open source GitHub project)	https://github.com/SaiPrajwal95/annotate-to-KITTI	KITTI	Python based CLI utility. Just clone it and launch. Simple and Powerful.
LabelBox	JavaScript, HTML, CSS, Python	Cloud or On-premise, some interfaces are Apache-2.0	https://www.labelbox.com/	json, csv, coco, voc	Web application
LabelMe	Perl, JavaScript, HTML, CSS, On Web	MIT License	http://labelme.csail.mit.edu/Release3.0/	xml	Converts only jpeg images.
Dataturks	On web	Apache License 2.0	https://dataturks.com/	json	Converts to json format but creates single json file for all annotated images.
LabelImg	ubuntu	OSI Approved:: MIT License	https://mlnotesblog.wordpress.com/2017/12/16/how-to-install-labelimg-in-ubuntu-16-04/	xml	Need to install dependencies given in reference.
Dataset_annotator	Ubuntu	2018 George Mason University Permission is hereby granted, Free of charge	https://github.com/omenyayl/dataset-annotator	json	Need to install app_image and run it by changing permissions.

References

- Google Tensorflow Object Detection Github
- Pretrained TensorFlow Model for Object Detection
- Python Sample Code for Custom Object Detection
- Train Model Using TensorFlow

Technical Support Assistance

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

Revision History

Revision 1.1, December 2020

Section	Change Summary
Programming the Binary and Bitstream Files	Updated Table 8.1. SPI Flash Options Selection Guide.

Revision 1.0, May 2019

Section	Change Summary
All	Initial release

www.latticesemi.com