Object Counting Using Convolutional Neural Network Accelerator IP

Reference Design

FPGA-RD-02036 Version 1.1

September 2018

Contents

Figures

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
AXI	Advanced Extensible Interface
CNN	Convolutional Neural Network
DRAM	Dynamic Random Access Memory
FPGA	Field-Programmable Gate Array
DVI	Digital Visual Interface
GUI	Graphic User Interface
PIP	Picture In Picture
SPI	Serial Peripheral Interface
SD Card	Secure Digital (Memory) Card

1. Introduction

This document describes the Object (fruit) Counting Machine Learning Neural Network reference design. This reference design can be implemented on Lattice's Embedded Vision Development Kit, featuring the Lattice CrossLink™ and ECP5™ FPGA devices.

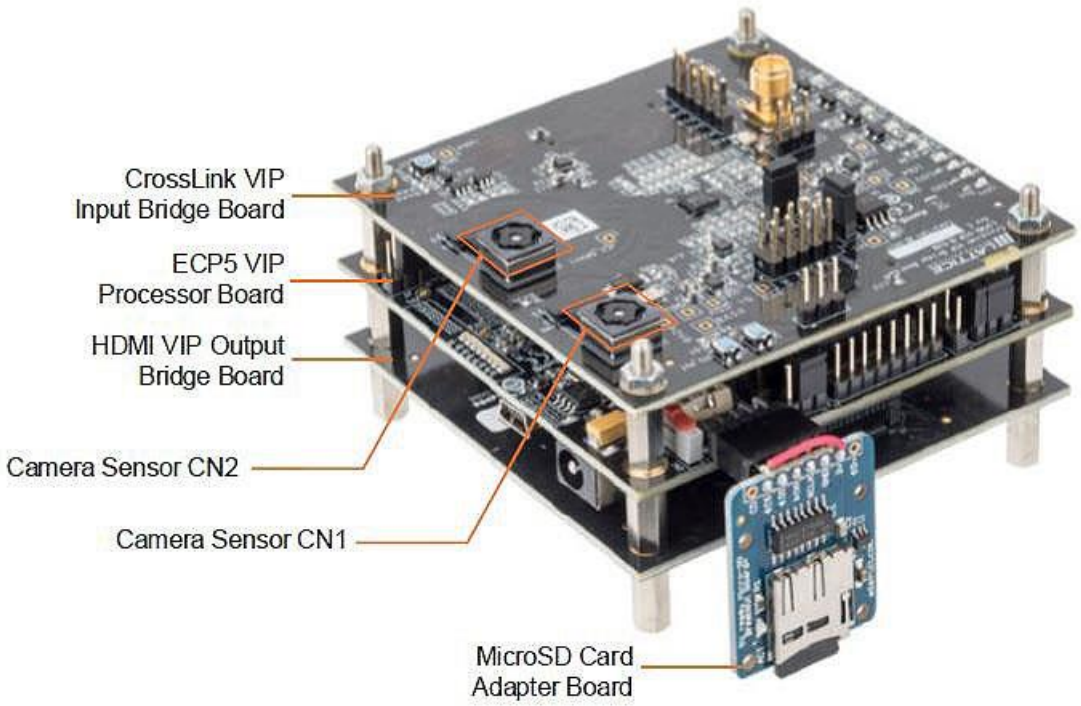

Figure 1.1. Lattice Embedded Vision Development Kit with MicroSD Card Adapter Board

2. Related Documentation

2.1. Soft IP Document

CNN Accelerator IP Core User Guide (FPGA-IPUG-02037)

2.2. Diamond Document

For more information on Lattice Diamond Software, visit the Lattice website at:
www.latticesemi.com/Products/DesignSoftwareAndIP

3. Software Requirements

- Lattice Diamond® Software Version 3.10
- Synplify Pro® Synthesis Tool

4. Hardware Requirements

- Lattice Embedded Vision Development Kit (LF-EVDK1-EVN)
- Mini-USB Cable (included in the Lattice Embedded Vision Development Kit)
- 12 V Power Supply (included with the Kit)
- HDMI Cable
- HDMI Monitor (1080p60)
- Micro-SD Card Adapter (MICROSD-ADP-ENV)
- Micro-SD Card. Standard Micro-SD card only.

5. Reference Design Overview

5.1. Block Diagram

Figure 5.1 shows the block diagram of the Object Counting reference design.

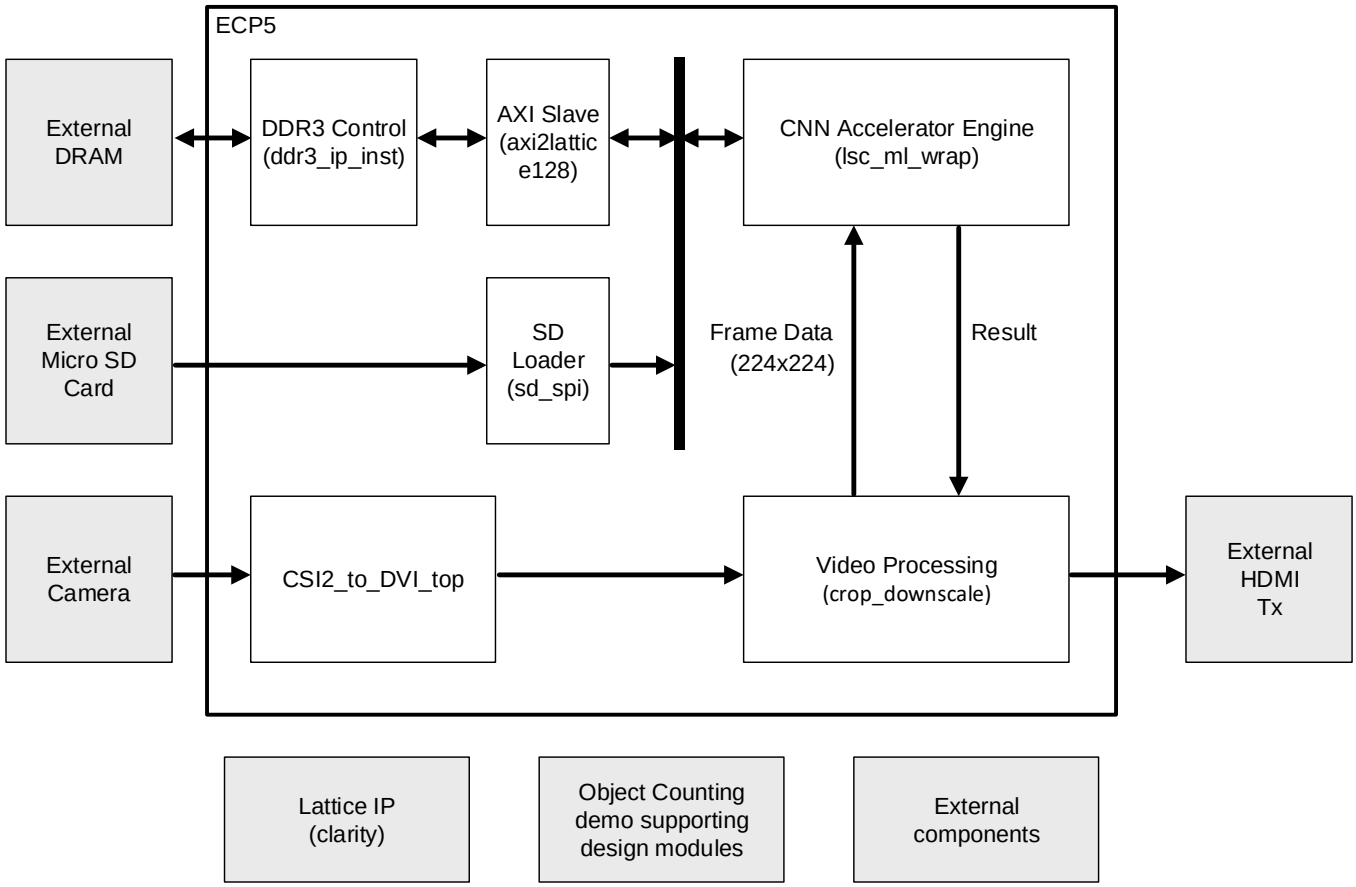

Figure 5.1. Object Counting Reference Design Block Diagram

5.2. Top Level Blocks

This Reference Design uses ECP5-85 FPGA containing the following major blocks:

- CNN Accelerator Engine
- SD card to SPI interface
- AXI Slave interface
- DDR3 memory interface
- CSI2 to DVI interface
- Video processing module

6. CNN Accelerator Engine

The Lattice Semiconductor CNN Accelerator IP Core is a calculation engine for Deep Neural Network with fixed point weight or binarized weight. It calculates full layers of Neural Network including convolution layer, pooling layer, batch normalization layer, and full connect layer by executing sequence code with weight value which is generated by the Lattice Neural Network Compiler tool. Engine is optimized for convolutional neural network, hence it can be used for vision-based applications such as classification or object detection and tracking. The IP Core does not require an extra processor; it can perform all required calculations by itself.

The design is implemented in Verilog HDL. It can be targeted to ECP5U, ECP5UM, and ECP5UM5G FPGA devices and implemented using the Lattice Diamond Software Place and Route tool integrated with the Synplify Pro® synthesis tool.

The key features of the CNN Accelerator IP Core include:

- Supports convolution layer, max pooling layer, batch normalization layer and full connect layer
- Configurable bit width of weight (16-bit, 1-bit)
- Configurable bit width of activation (16/8-bit, 1-bit)
- Dynamically support 16-bit and 8-bit width of activation
- Configurable number of memory blocks for tradeoff between resource and performance
- Configurable number of convolution engines for tradeoff between resource and performance
- Optimized for 3x3 2D convolution calculation
- Dynamically support various 1D convolution from 1 to 72 taps
- Support max pooling with overlap (For example, kernel 3 and stride 2)

Engine configuration parameters can be set using the Clarity Designer's IP Core Configuration GUI as shown in Figure 6.1.

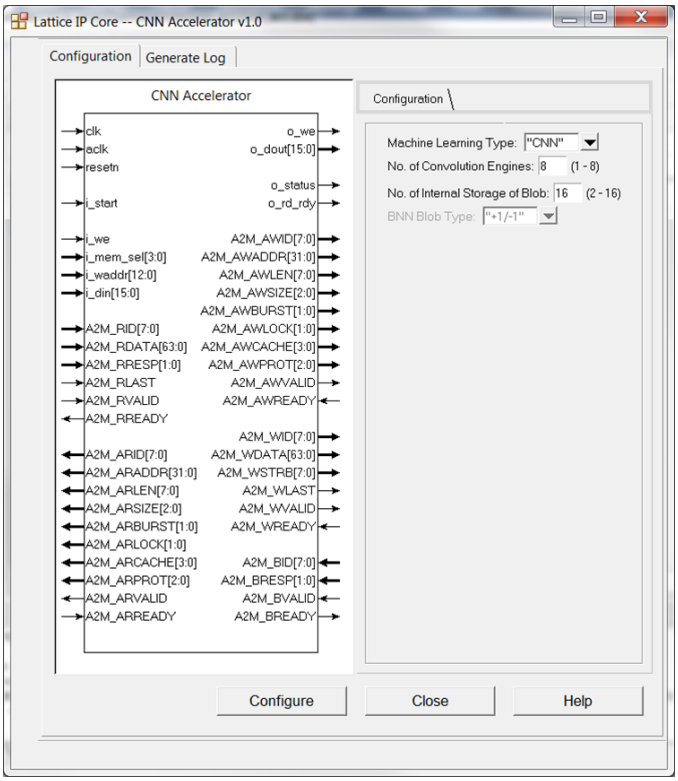

Figure 6.1. CNN Accelerator IP Core Generation GUI

Figure 6.2 shows the inputs and outputs of this IP module.

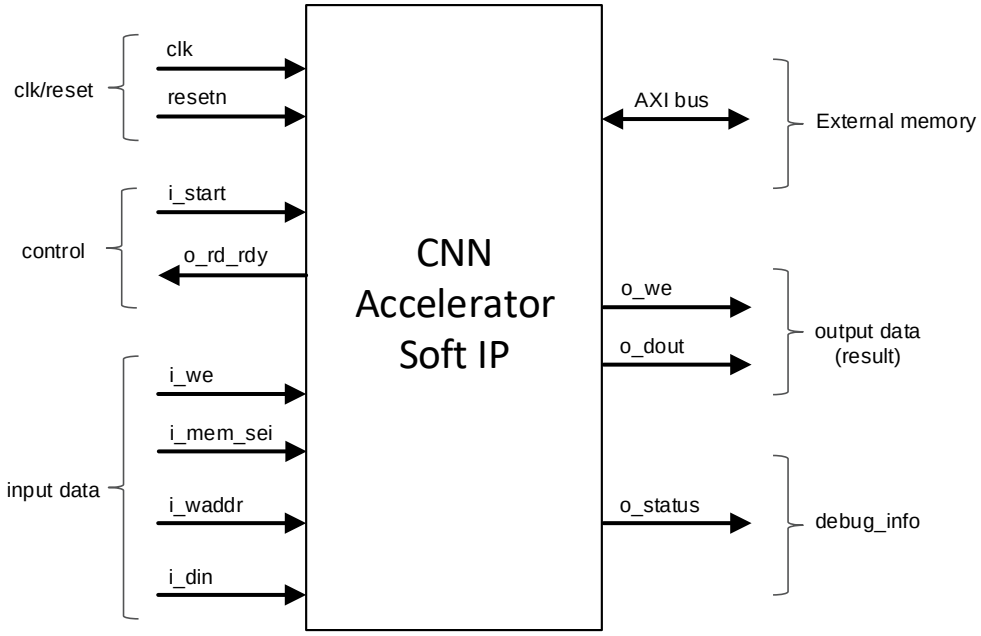

Figure 6.2. CNN Accelerator Soft IP Inputs and Outputs

Input Data Format:

Input data is a sequence of 8-bit or 16-bit data. Memory index and address are decided by Neural Network. Therefore, external block should process input raw data and write input data to Lattice CNN Accelerator IP Core through input data write interface. Since CNN Accelerator IP Core has only 16-bit width interface, external block should pack two of 8-bit data if 8-bit width is used for input data layer.

Because memory assignment is defined by Neural Network, external block should handle input raw data and write it to proper position of internal memory of the CNN Accelerator IP Core.

For fruit detection, an external memory is used as buffer for input data due to its size. Data will be fetched and placed into the internal memories as needed. The IP core expects data in little-endian order.

Result:

Result, that is, by final blob data of neural network can be written to DRAM per command code. In this case, external logic should read result data from DRAM. However, command code also can simply feed result data to external logic through this interface. Interface consists of o_we as valid indicator and o_dout as 16-bit data. Usually, it is a single burst series of 16-bit data and it's also fully programmable by command code.

Output Data Format:

Output data is a sequence of 16-bit data which is controlled by commands. Amount of data is also decided by Neural Network, that is, by output blobs. External block should interpret output sequence and generate usable information. For fruit detection example, the output stream consists of three parts. The first part is fruit detection confidence probability, the second part is confidence score for two classes (apple, orange), and the last part is bounding box coordinates. The dimensions are [6, 1, 7, 7], [6, 2, 7, 7], and [6, 4, 7, 7], respectively. The first index in the dimension represents the number of anchors per grid. The second index is part dependent. To represent the confidence probability, we only need only one element. For the confidence score, we need two elements to represent the score for each class, and the bounding box coordinate requires four elements to represent X center coordinance, Y center coordinance, width and height of a bounding box. The third and fourth indices are the number of grids in Y and X axis. The IP core outputs data in little-endian order.

Command Format:

Command is a sequence of 32-bit data with or without additional parameters or weights as shown in Figure 6.3. It should be loaded at DRAM address 0x0000 before execution. Command is generated by the Lattice Neural Network Compiler tool. For more information, refer to Lattice Neural Network Compiler Software User Guide (FPGA-UG-02052).

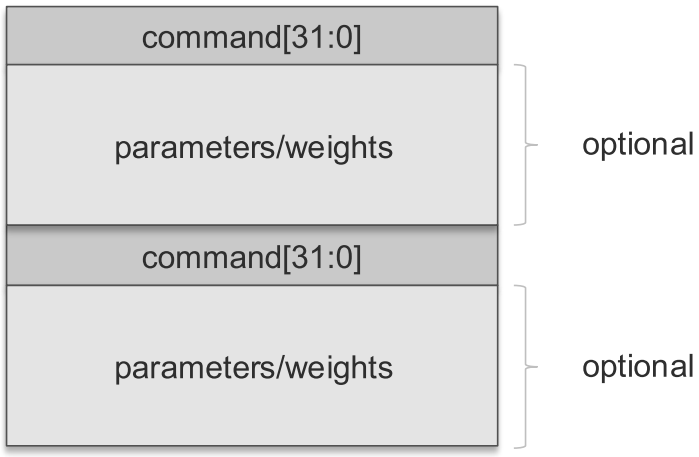

Figure 6.3. Command Format

7. SD Card Loader

SD card interface in this design is used to get the input data into the CNN accelerator IP. SD card contains a binary file that is generated by Lattice Neural Network Compiler Tool.

Lattice Neural Network Compiler tool allows analyzing and compiling a trained neural network (such as what is generated by Caffe or TensorFlow tools) for use with select Lattice Semiconductor FPGA products.

Lattice Neural Network Compiler tool outputs three files:

- A hardware configuration file (*.yml) that contains info on fixed point converted network and memory allocation.
- A firmware file (*.lscml) that contains weights coming from a trained model file.
- A binary file (*.bin) generated from firmware file (*.Lscml) for programming into SD card.

Figure 7.1 shows Neural Network Compiler tool's file generation flow.

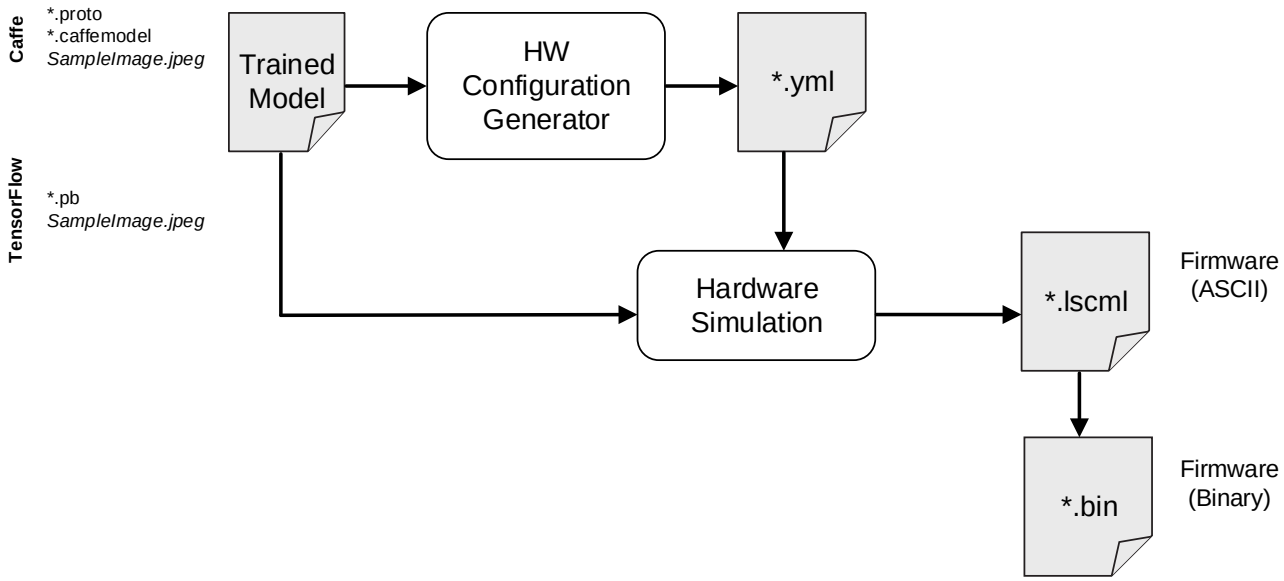

Figure 7.1. Neural Network Compiler Output File Generation and Flow

8. AXI Slave and DDR3 Memory Interface

AXI interface allows command code to be written in DRAM before execution of CNN Accelerator IP Core. Input data may also be written in DRAM. CNN Accelerator IP Core reads command code from DRAM, and performs calculations using internal sub execution engines. Intermediate data may also be transferred from/to DRAM per command code.

9. CSI2 to DVI Interface

This module implements a bridge function that converts the camera input's MIPI CSI data to DVI output using Lattice CrossLink and SiI1136 HDMI transmitter.

10. Video Processing Module

The crop_downscale module provides all the necessary functions needed to manage the process of inputting data, receiving output data, and generating a composite image for output to the HDMI interface. In this design crop_downscale_keyF.v – crops input to 224x224.

Key functions of the code include:

- Capturing a downscaled image from the camera input module and saving it to a frame buffer.
- Writing the frame buffer data into CNN accelerator engine during the blanking period.
- Buffering the output after completion of the image data processing.
- Creating a PIP (Picture-in-Picture) bounding box with green borders and outputting the composite image.

Output from CSI2_to_DVI_top module is a stream of data that reflects the camera image. Input image is then downscaled to 224x224 pixels, stored in a frame buffer and passed to output.

Image data is written from the frame buffer into the CNN acceleration engine prior to the start of the processing. Data is then formatted for compatibility with the trained network.

The *.yml file provides majority of the information needed for understanding how the input data should be prepared. A snippet of the code in *.yml file for Object (fruit) Counting design is shown in Figure 10.1.

```
cfrac: 1024
Mean: 128
Scale: 255
Input Size: [1, 3, 224, 224]
num_ebr: 16
ebr_blk_size: 16384
num_conv_eng: 8
binary_mode: false
blob:
  Input1:
    frac: 64
    memblks: 12
    extmem: 0x000000
    height_per_mem: 56
```

Figure 10.1. Object Counting Design *.yml file code snippet

- Input Size: [1, 3, 224, 224] -- Indicates one input array consisting of 3 layers of dimensions 224x224.
- memblks: 3 – Total number of memory blocks needed.
- depth_per_mem: 1 – Number of memory blocks allocated to each memory layer.
- frac: 8 – Number of bits that is allocated to the fractional component. It is equal to the minimum number of bits to represent this number minus 1. In this case, 3 bits to represent 8-1=7.
- num_ebr: 16 – Number of memory blocks.

  Note: Despite the variable name, this does not tie directly to the number of Embedded Block Ram (EBR) used in the design.
- ebr_blk_size: 16384 – This defines the size of the memory blocks in Bytes. Note the blocks have a width of 16 bits and the depth is variable.

CNN accelerator engine's port results, o_we and o_dout[15:0], can be used to output any number of results.

Designer can add a read command to allow reading any data based on the neural network design.

In fruit counting design, we use 5 results to be transferred from CNN accelerator engine to crop_downscale_keyF module. These are used to create red and green dots indicating the position of apples and oranges on the output image stream:

- X pixels (position of the fruit on x axis)
- Y pixels (position of the fruit on y axis)
- Fruit detection probability
- Class probability of apple
- Class probability of orange

11. Generating the Firmware File

To generate the Object Counting Firmware file:

1. Using the files located in **Software/Data/Tensorflow/VIP_ObjectCountDemo/VIP_ObjectCountDemo.ldnn**, create a new project in SensAI and apply the following settings as shown in Figure 11.1.
   - **Framework — TensorFlow**
   - **Device — ECP5**
   - **Class — CNN**
   - **Network File — Software/ML_Training_Results/Inference/Tensorflow/ObjectCount.pb**
   - **Image/Video/Audio Data — Software/ML_Training_Results/cap_11.png**
     (Change file type when searching for this file.)
2. Click **Next**.

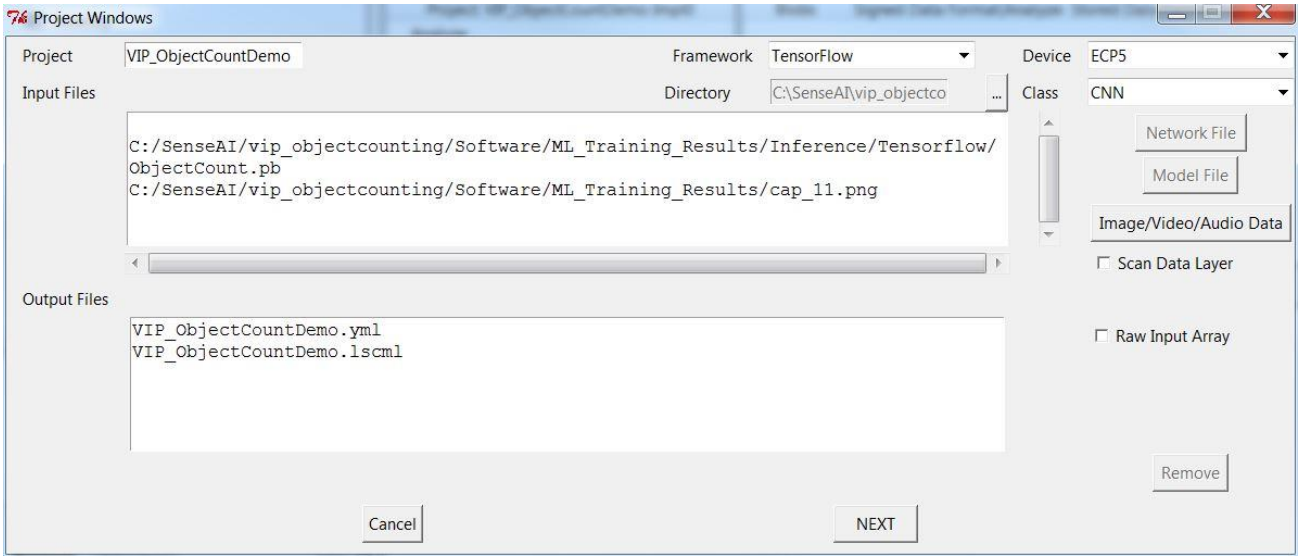

Figure 11.1. SensAI v1.1 Project Settings Part 1

3. In the second section, apply the Neural Network engine settings as shown in Figure 11.2:
   - **Mean Value for Data Pre-Processing — 128**
   - **Scale Value for Data Pre-Processing — 1.0**
   - **Enable Borrow Feature — Check**
   - **Borrow Factor (2/4/8) — 4**
4. Click **OK**.

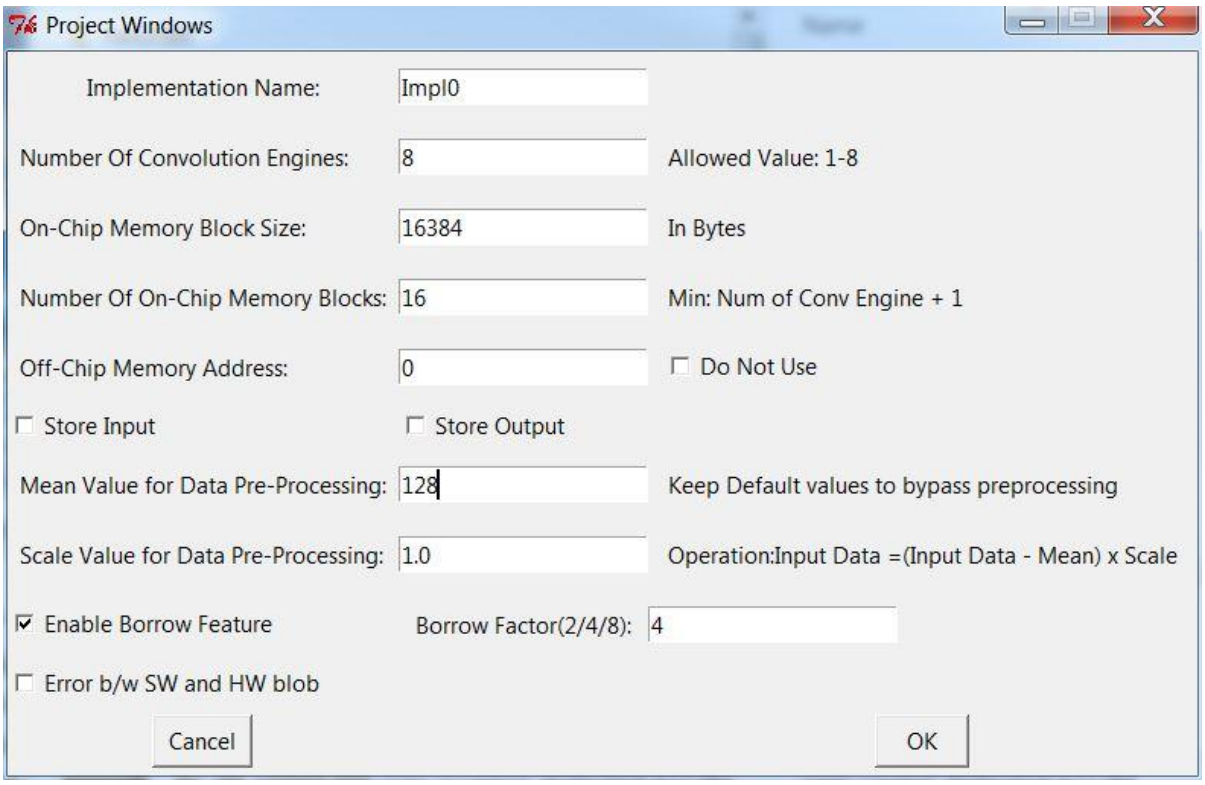

Figure 11.2. SensAI v1.1 Project Settings Part 2

5. Save the project file and analyze the network. Figure 11.3 shows the analysis results.

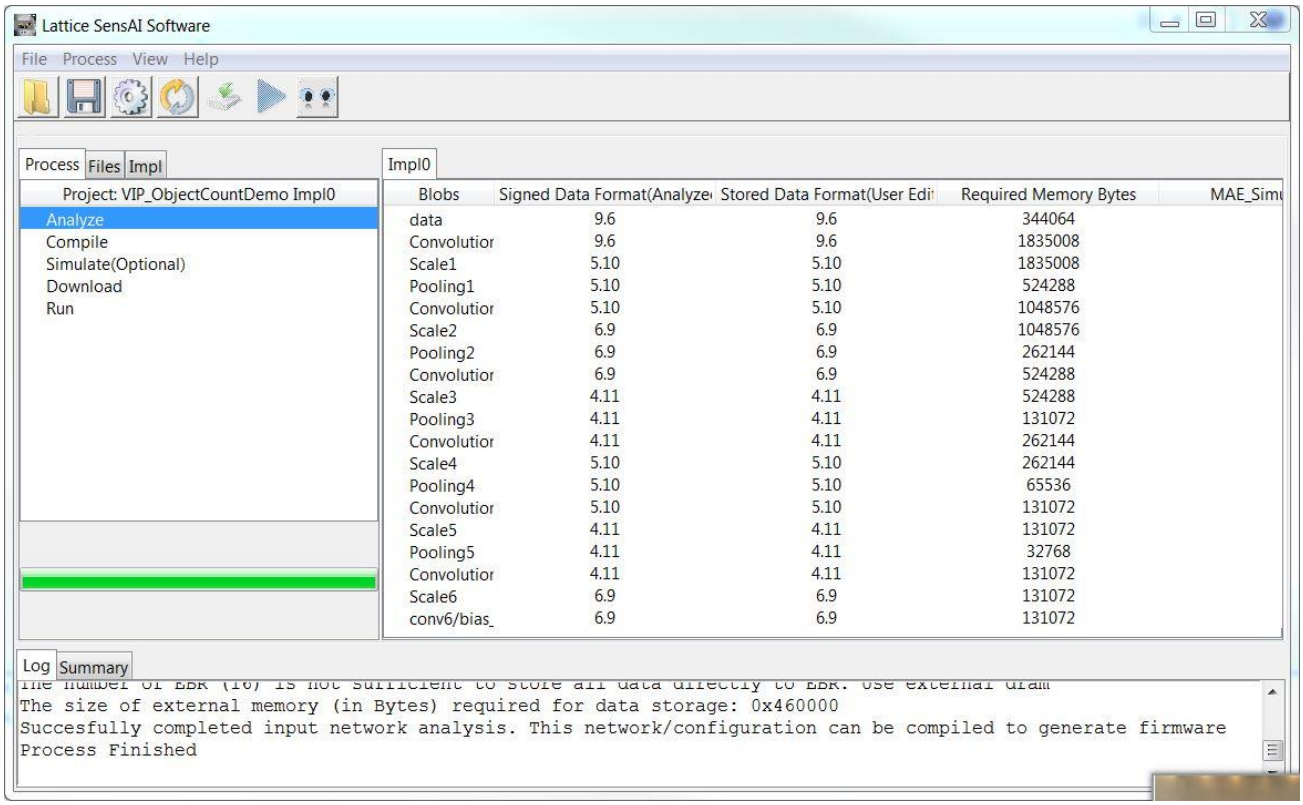

Figure 11.3. SensAI v1.1 Network Analysis Results

6. This reference design requires fractional bit changes to fine-tune its accuracy.

   Fine-tuning and optimization can be dramatic, hence, a golden **.yml** file is provided to replace the **VIP_ObjectCountDemo.yml** file that is generated when analyzing the network. The golden **.yml** file is located at **Software/Data/Tensorflow/VIP_ObjectCountDemo/Impl0/VIP_ObjectCountDemo_BorrowFinal_Tensorflow.yml**.

   Note: Every time you analyze the network, the **VIP_ObjectCountDemo.yml** is regenerated. As such, you must replace it with the golden **.yml** file before compiling.

7. After replacing the file, click **Compile** to generate the Firmware file.

After compiling, you can find the generated firmware file at: **Software/Data/Tensorflow/VIP_ ObjectCountDemo /Impl0/VIP_ ObjectCountDemo.bin**.

References

For more information on the FPGA device, visit: http://www.latticesemi.com/Products/FPGAandCPLD/ECP5.

For complete information on Lattice Diamond Project-Based Environment, Design Flow, Implementation Flow and Tasks, as well as on the Simulation Flow, see the Lattice Diamond User Guide.

Technical Support Assistance

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

Revision History

Revision 1.1, September 2018

Section	Change Summary
Generating the Firmware File	Added this section.

Revision 1.0, May 2018

Section	Change Summary
All	Initial release.

7th Floor, 111 SW 5th Avenue
Portland, OR 97204, USA
T 503.268.8000
www.latticesemi.com