Verilog code for CNN-based digit classification

Dr. Edwin Dhas D

This Verilog code can be used to implement the digit classification system on hardware like FPGA. The digit classification architecture uses a CNN algorithm that classifies the digits 0,1,2,3,4,5,6,7,8, and 9. The model was initially trained using the digit dataset. The following MATLAB link helps to train the model that uses the digit dataset present in the MATLAB toolbox.

https://in.mathworks.com/help/deeplearning/ug/create-simple-deep-learning-network-for-classification.html

The structure of the CNN architecture used for digit classification is shown in Fig. 1

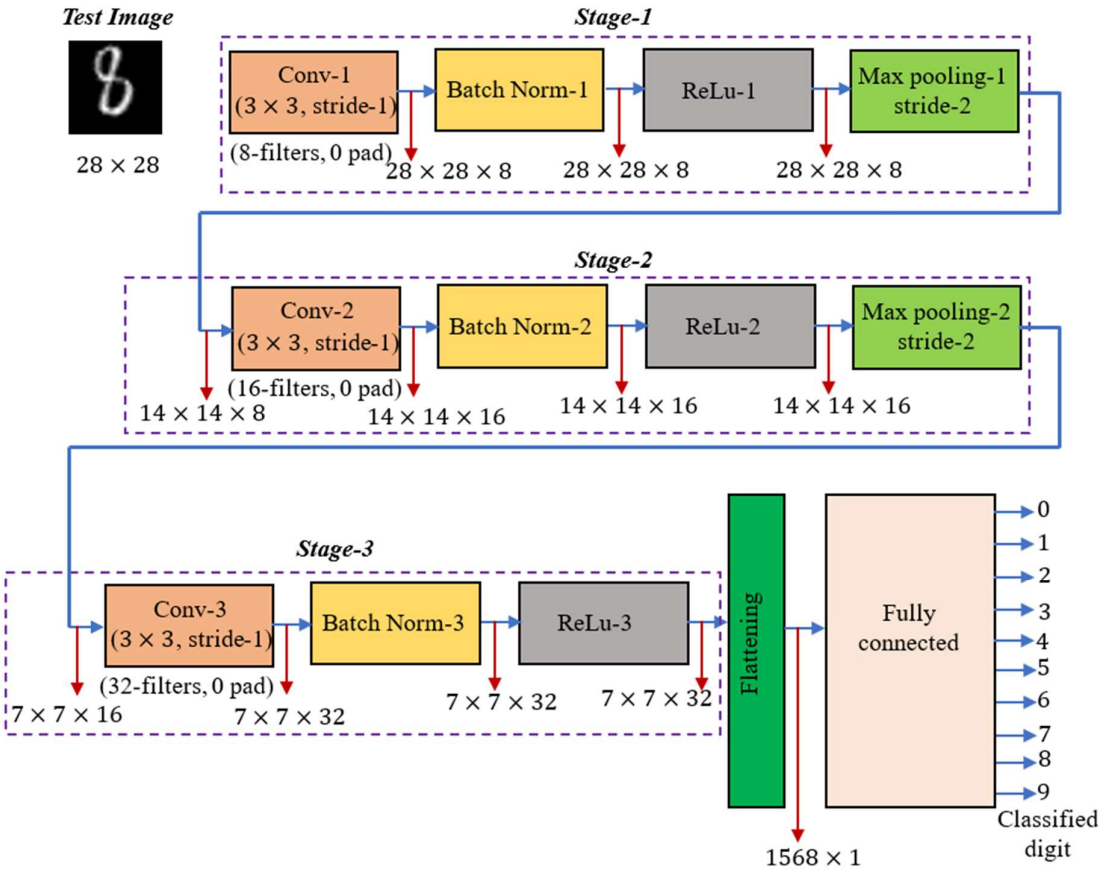

Fig. 1: CNN model in Digit classification

As the MATLAB code in the above link is executed it will generate a trained model. The trained model has the following essential components to develop the Verilog code to perform digit classification.

1. Convolutional-1 filter weights and biases (8 filters).
2. Parameters in Batch normalization-1 (Mean, Variance, Scale, Shift parameters)
3. Convolutional-2 filter weights and biases (16 filters).
4. Parameters in Batch normalization-2 (Mean, Variance, Scale, Shift parameters)
5. Convolutional-3 filter weights and biases (32 filters).
6. Parameters in Batch normalization-2 (Mean, Variance, Scale, Shift parameters)
7. Fully connected network weights and biases.

The dataset can be downloaded from the following Kaggle links for testing the Verilog codes

https://www.kaggle.com/datasets/jidhumohan/mnist-png

https://www.kaggle.com/datasets/alexanderyyy/mnist-png

The digit images from these datasets are grayscale images that have a size of 28x28.

Annexure A: Instruction for Verilog simulation of Digit classification using Vivado

1. Download the CNN_based_digit_classification folder from GitHub and save it to any drive on your computer.
2. Download the digit images from any one of the two Kaggle links and paste them into the DigitDataset folder (These images are used as test images).
3. *Conversion of Image to Text file:*

   (a) For MATLAB: Run 'write_text_data.m' and upload any one of the test images from the DigitDataset folder. This will convert the image file to text data ('data.txt') suitable for the circuit input.

   (b) For Python: Run 'write_text_data.py'. Kindly provide the path of the test image in the variable xb before executing 'write_text_data.py' as shown below (Fig. 2).

```
6    xb='./DigitDataset/6/image5025.png'
```

Fig. 2: Specifying the path of the input image

In this example, I have uploaded the digit 6 with the file named 'image 5025' as shown below (Fig. 3).

Fig. 3: Sample image

This will convert the image file to text data (‘data.txt’) suitable for circuit input. The ‘data.txt’ file will be created in the directory named ‘data’.

4. **Vivado 2018.2**: If you are using vivado 2018.2, you can directly download the project directory from the google drive https://drive.google.com/drive/folders/1xpHIwD0S2VJ7uspfz5nIIryuGqFVdC1K?usp=sharing.

   After downloading, unzip and open the Vivado project ‘Digit_classification_Vivado.xpr’ (from the Digit_classification_Vivado folder) using Vivado and follow the next step.

   Other Vivado versions: If you are using other versions of Vivado, follow the procedure provided in **Annexure B** and proceed to the next step.

5. Make ‘tb_main_program.v’ as the top level and provide an appropriate path of the data folder in the tb_main_program for variables file, file1, file2….. file56 as shown below (Fig. 4).

```
75  file= $fopen("H:/CNN_based_digit_classification/data/data.txt","r");
76  file1= $fopen("H:/CNN_based_digit_classification/data/conv1_1.txt","r+");
77  file2= $fopen("H:/CNN_based_digit_classification/data/conv1_2.txt","r+");
78  file3= $fopen("H:/CNN_based_digit_classification/data/conv1_3.txt","r+");
79  file4= $fopen("H:/CNN_based_digit_classification/data/conv1_4.txt","r+");
80  file5= $fopen("H:/CNN_based_digit_classification/data/conv1_5.txt","r+");
81  file6= $fopen("H:/CNN_based_digit_classification/data/conv1_6.txt","r+");
82  file7= $fopen("H:/CNN_based_digit_classification/data/conv1_7.txt","r+");
83  file8= $fopen("H:/CNN_based_digit_classification/data/conv1_8.txt","r+");
84  file9= $fopen("H:/CNN_based_digit_classification/data/conv2_1.txt","r+");
85  file10= $fopen("H:/CNN_based_digit_classification/data/conv2_2.txt","r+");
86  file11= $fopen("H:/CNN_based_digit_classification/data/conv2_3.txt","r+");
87  file12= $fopen("H:/CNN_based_digit_classification/data/conv2_4.txt","r+");
88  file13= $fopen("H:/CNN_based_digit_classification/data/conv2_5.txt","r+");
```

Fig. 4: Specifying the path to record the Convolutional filter outputs

6. Run the simulation

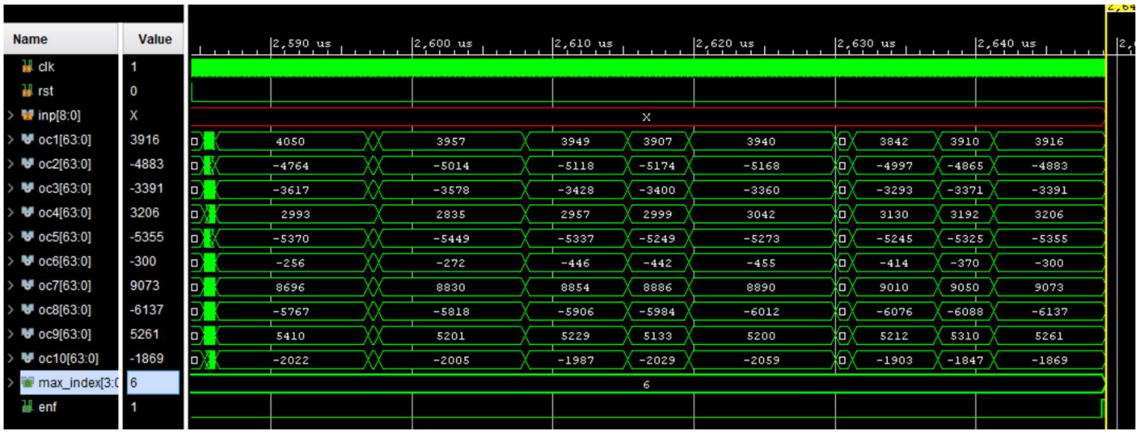

Fig. 5: Simulation result

The probabilities of the class digits 0, 1,2… and 9 are obtained in the variables oc1, oc2, oc3….and oc10 respectively. The Classified result is obtained in the variable max_index making the output enable signal 'enf' to logic 1 as shown in Fig. 5. In this example, the class is correctly recognized as digit 6. The classified result will also be displayed in the TCL console as follows

```
run all
THE CLASSIFIED RESULT IS DIGIT:  6
$finish called at time : 2648950 ns
```

Fig. 6: Classified result in TCL console window

7. The results at each convolutional stage (Convolutional layer-1, Convolutional layer-2, and Convolutional layer-3) which are generated during HDL simulation are recorded in the data directory. This can be viewed either using Python or MATLAB
   (a) **For MATLAB:** Run 'display_output.m'
   (b) **For Python:** Run 'display_output.py'

   The output of convolutional filter-1, Convolutional filter-2, and Convolutional filter-3 will be displayed as follows (Fig. 7 to Fig. 9).

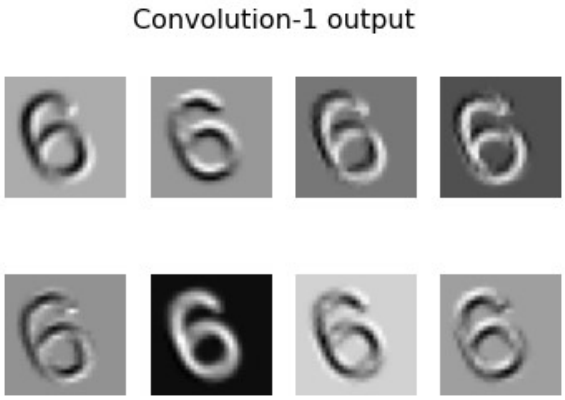

Fig. 7: HDL-generated Convolutional layer-1 output (8 filter outputs)

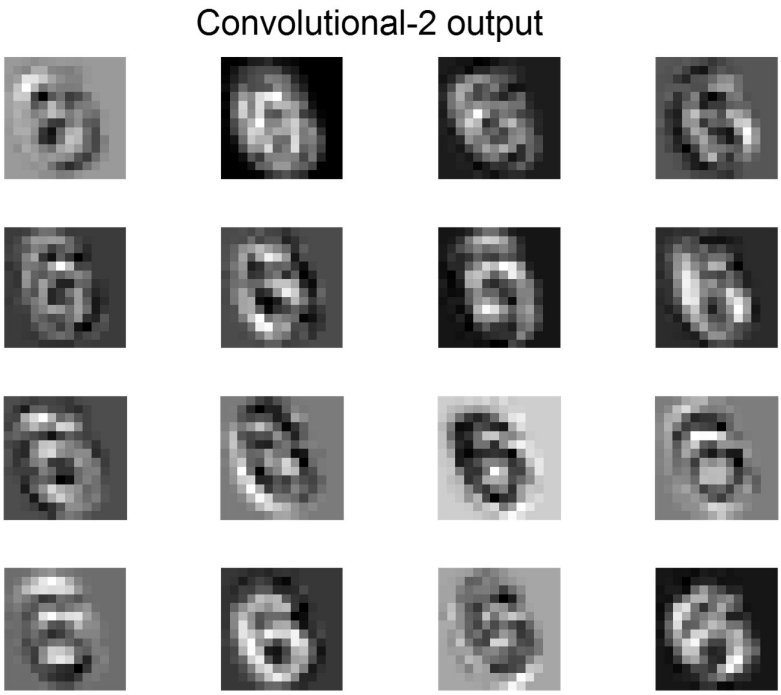

Fig. 8: HDL-generated Convolutional layer-2 output (16 filter outputs)

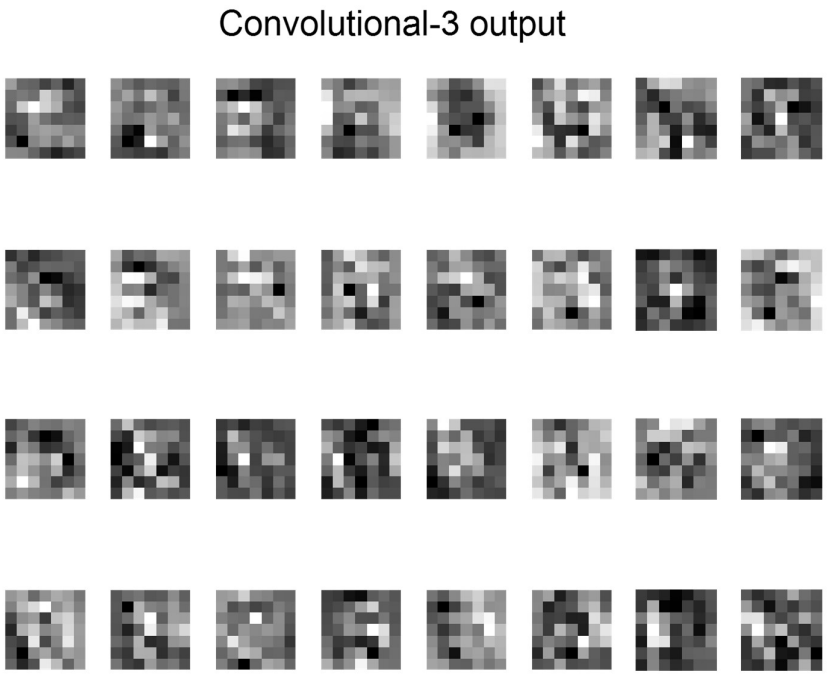

Fig. 9: HDL-generated Convolutional layer-3 output (32 filter outputs)

The schematic of the part of the circuit is illustrated in Fig. 10 to Fig. 12.

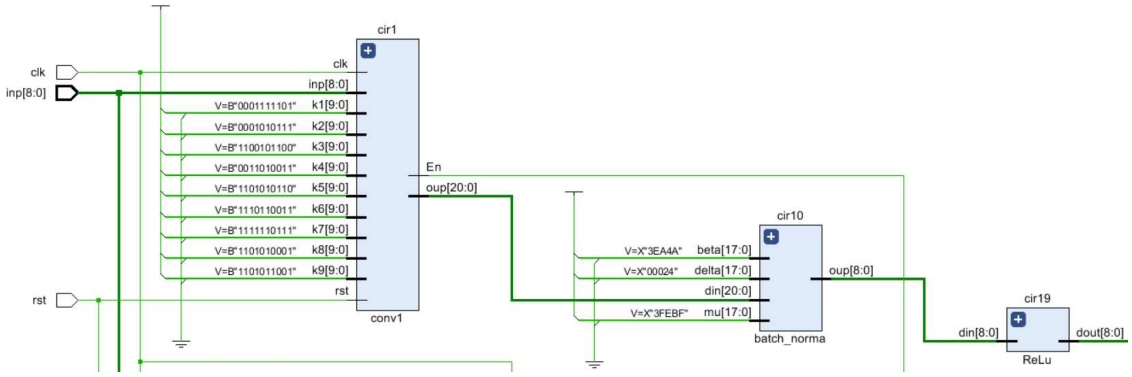

Fig. 10: Schematic section of inputs (clk, input pixel (inp), reset signal 'rst') with convolution, batch normalization, and ReLu

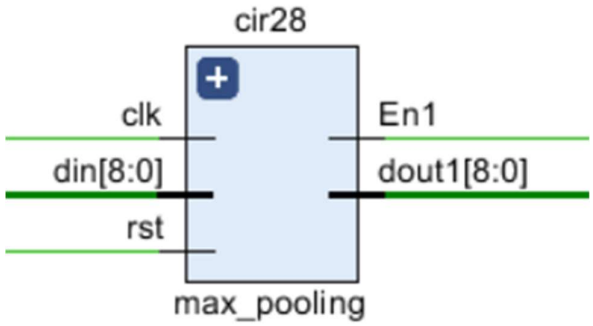

Fig. 11: Schematic section of max pooling

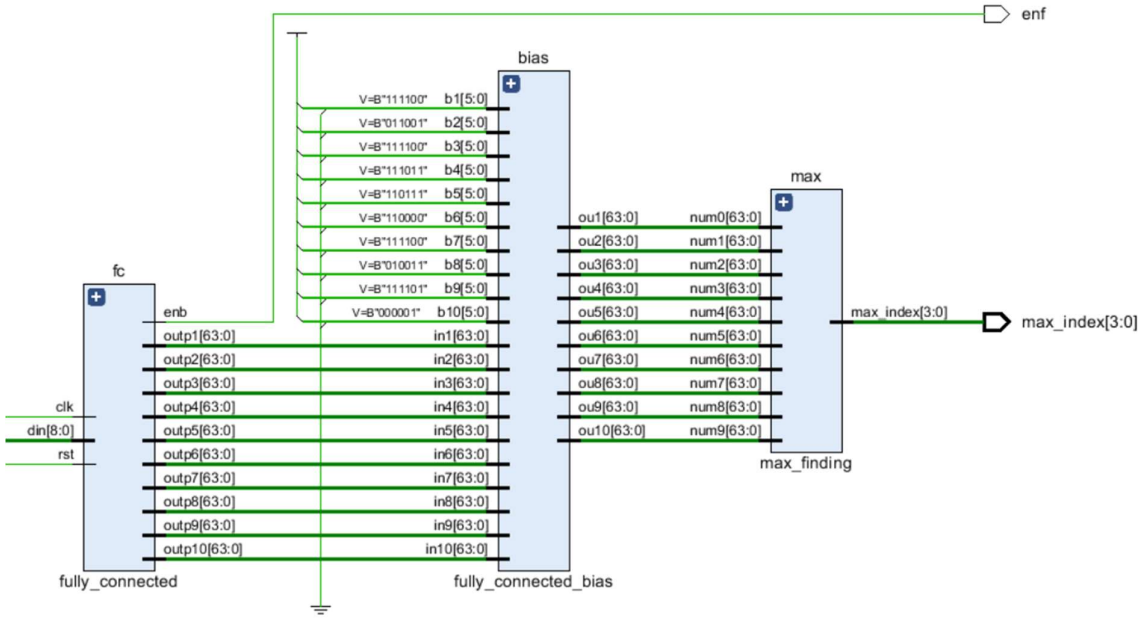

Fig. 12 Schematic section of the Fully connected layer with output ‘max_index’ and output enable signal ‘enf’

The output max_index generates the classified result as the enable signal enf is made high.

Annexure B: How to use with other versions of Vivado other than Vivado 2018.2

1. Upload the following files in your Vivado project (present in the verilog_files folder).
   tb_main_program.v
   main_program.v
   conv1.v
   batch_norma.v
   division.v
   ReLu.v
   max_pooling.v
   max_4.v
   conv2.v
   adding_8.v
   batch_norm2.v
   max_pooling1.v
   max_4a.v
   conv3.v
   adding_16.v
   batch_nom3.v
   division1.v
   flatten1.v
   fully_connected.v
   fully_connected_bias.v
   max_finding.v
2. Make the tb_main_program as top level.
3. Download the coefficient files (.coe files) that contain the weights of the fully connected layer.
4. Create 10 different Block memory generator IPs (Single port ROM), with width 10 and depth 1568, and load the 10 coe files, so that the read port latency must be 1 clock cycle as shown in Fig. 13 to Fig. 16.

Create the Block memory generator IP with the following names and upload its corresponding coe files.

weights_layer1<<< one.coe

weights_layer2<<< two.coe

weights_layer3<<< three.coe

weights_layer4<<< four.coe

weights_layer5<<< five.coe

weights_layer6<<< six.coe

weights_layer7<<< seven.coe

weights_layer8<<< eight.coe

weights_layer9<<< nine.coe

weights_layer10<<< ten.coe

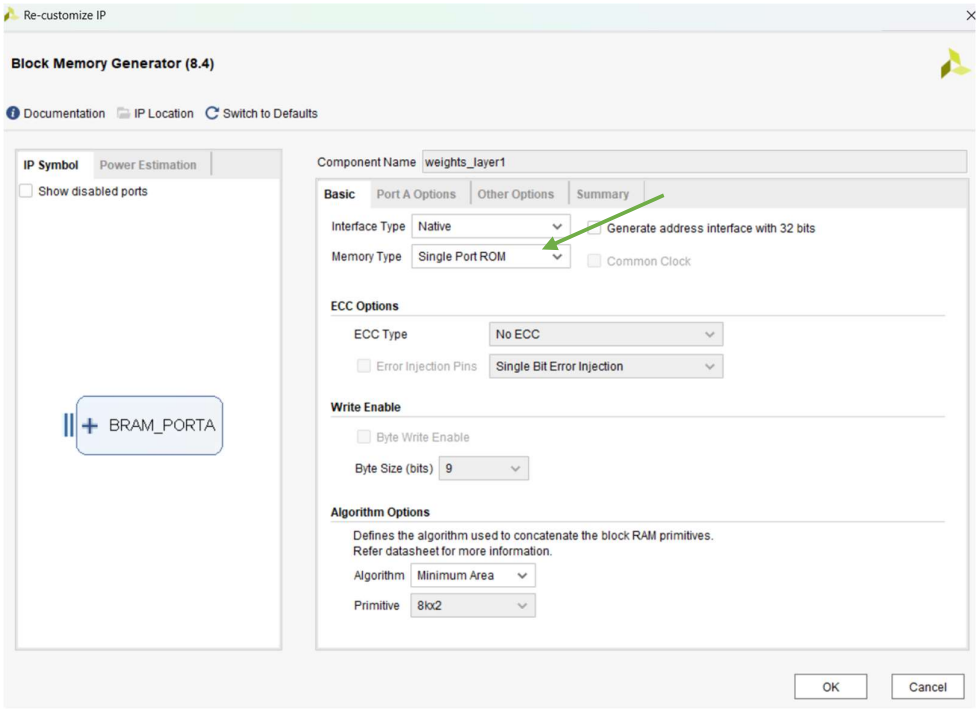

Fig. 13

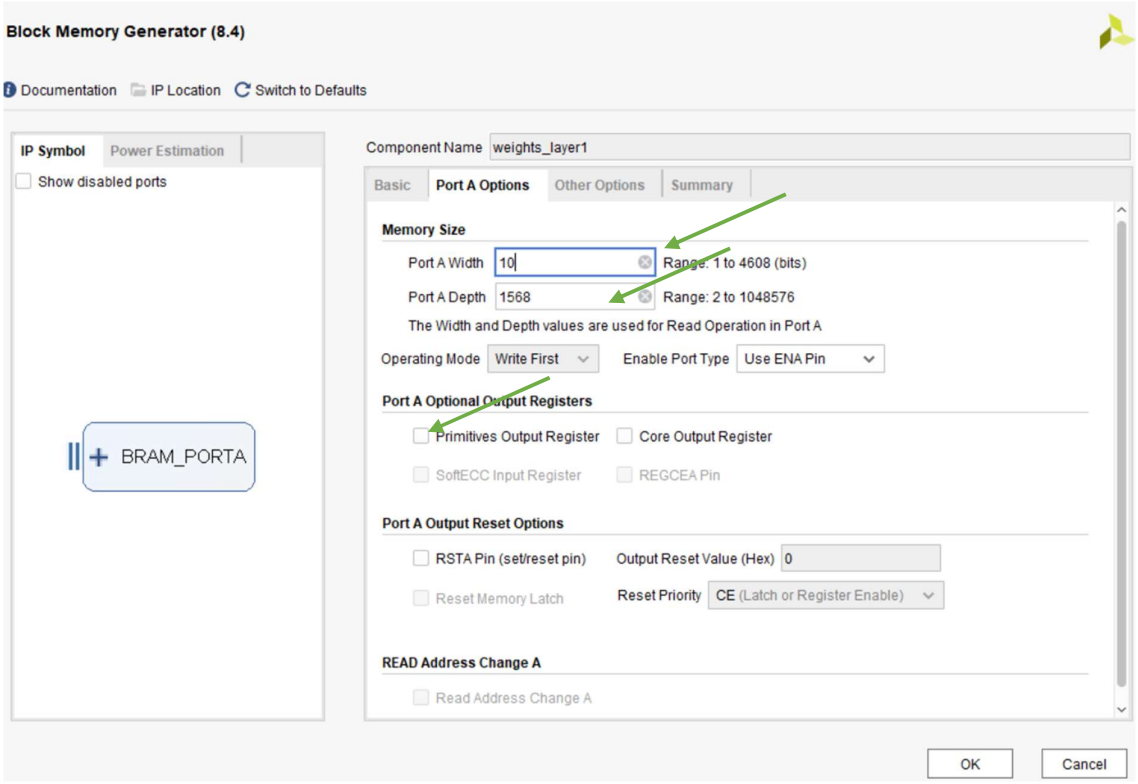
Block Memory Generator (8.4)
Documentation
IP Location
Switch to Defaults
IP Symbol
Power Estimation
Show disabled ports
BRAM_PORTA
Component Name
weights_layer1
Basic
Port A Options
Other Options
Summary
Memory Size
Port A Width
10
Range: 1 to 4608 (bits)
Port A Depth
1568
Range: 2 to 1048576
The Width and Depth values are used for Read Operation in Port A
Operating Mode
Write First
Enable Port Type
Use ENA Pin
Port A Optional Output Registers
Primitives Output Register
Core Output Register
SoftECC Input Register
REGCEA Pin
Port A Output Reset Options
RSTA Pin (set/reset pin)
Output Reset Value (Hex)
0
Reset Memory Latch
Reset Priority
CE (Latch or Register Enable)
READ Address Change A
Read Address Change A
OK
Cancel

Fig. 14

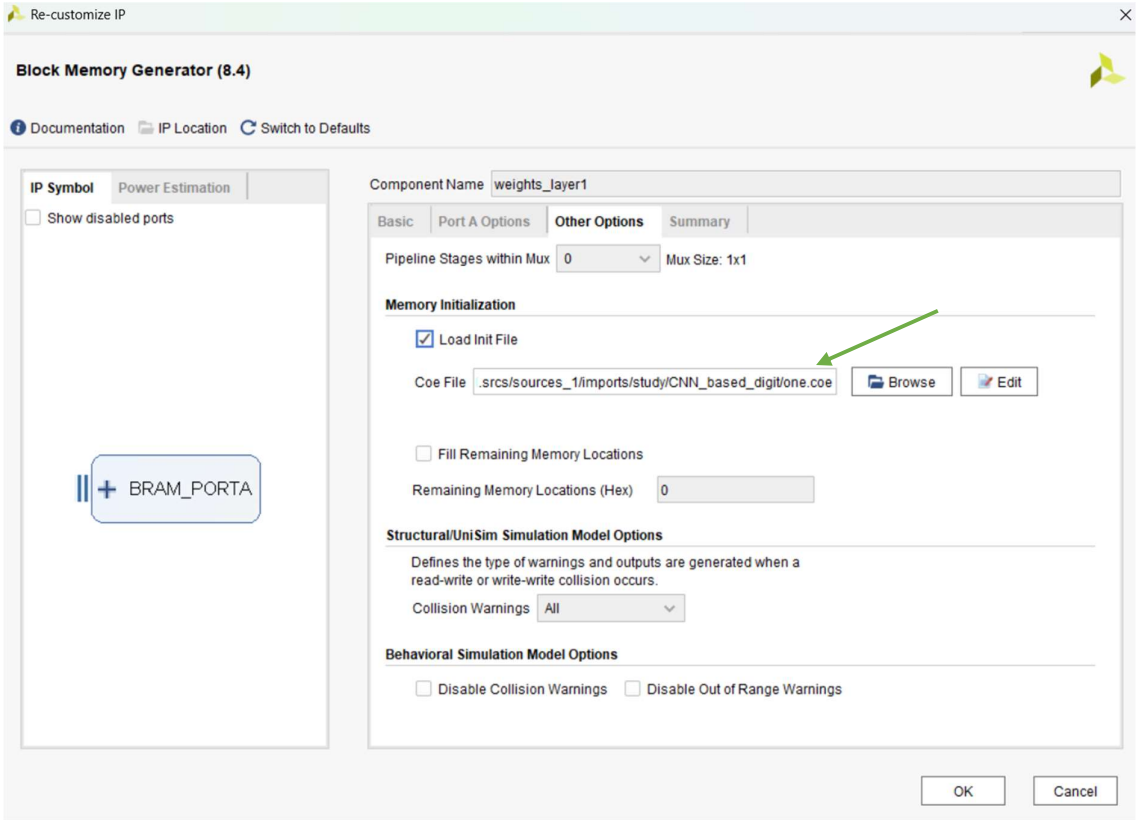
Re-customize IP
Block Memory Generator (8.4)
Documentation
IP Location
Switch to Defaults
IP Symbol
Power Estimation
Show disabled ports
BRAM_PORTA
Component Name
weights_layer1
Basic
Port A Options
Other Options
Summary
Pipeline Stages within Mux
0
Mux Size: 1x1
Memory Initialization
Load Init File
Coe File
.srcs/sources_1/imports/study/CNN_based_digit/one.coe
Browse
Edit
Fill Remaining Memory Locations
Remaining Memory Locations (Hex)
0
Structural/UniSim Simulation Model Options
Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.
Collision Warnings
All
Behavioral Simulation Model Options
Disable Collision Warnings
Disable Out of Range Warnings
OK
Cancel

Fig. 15

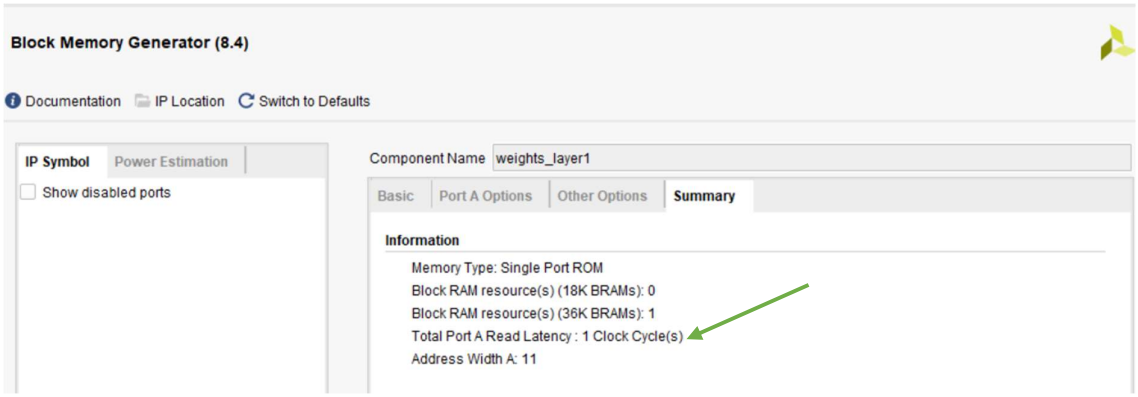

Fig. 16

5. Create a Block memory generator IP (Single port ROM) named Rom1, with width 11 and depth 1568, and load the flat_index.coe file as shown in Fig. 17.

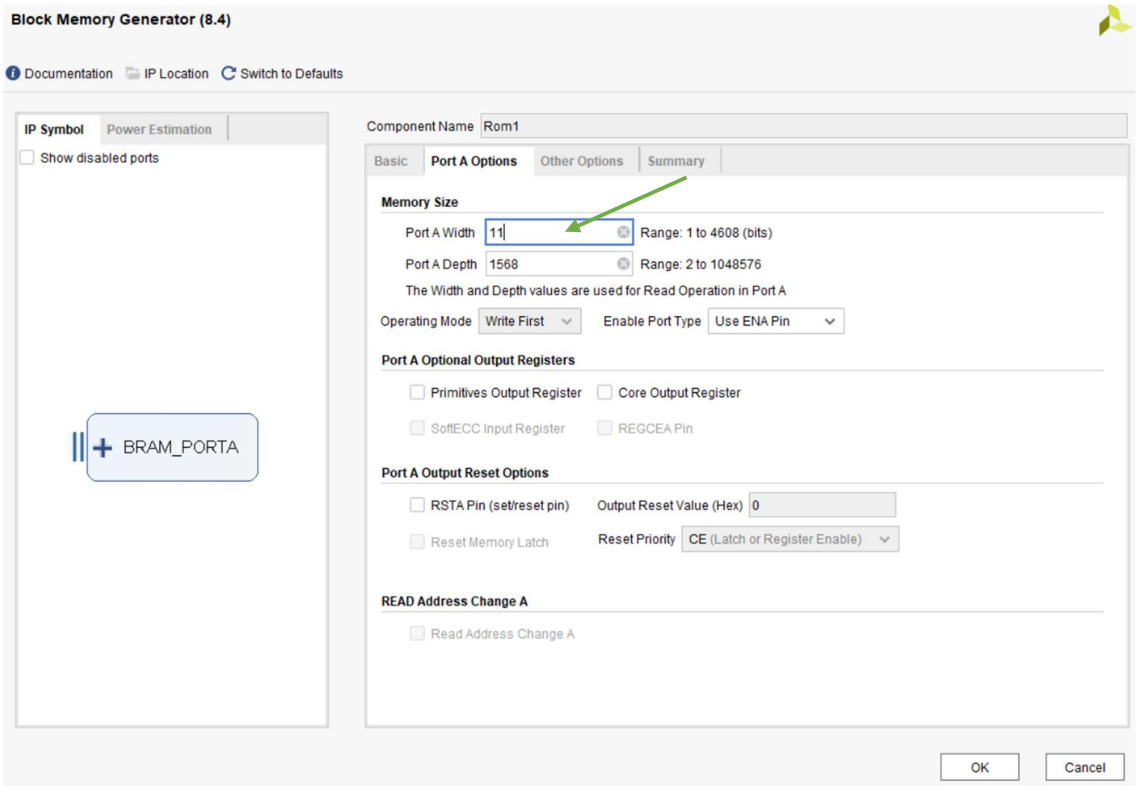

Fig. 17

Conclusion:

1. This Verilog code can be used to implement a Digit classification system in FPGAs
2. This design procedure can be used to implement various artificial intelligence algorithms (Deep learning architectures) in FPGA for different applications.