Indian Institute of Technology Tirupati

Mini Project

Course No: EE522L

Academic Year/Semester: 2023

LeNet5

Name-Vijay Raj

Roll Number-EE22M215

Name-Gaurav kumar

Roll Number-EE22M206

Name-Pawan Kumar

Roll Number-EE22M205

Name-Praveen Kumar Yadav

Roll Number-EE22M308

Instructor:

Dr. Vikramkumar Pudi

Teaching Assistant:

Pulli Keerthija

Vijaya Lakshmi

Thu 11th May, 2023

Contents

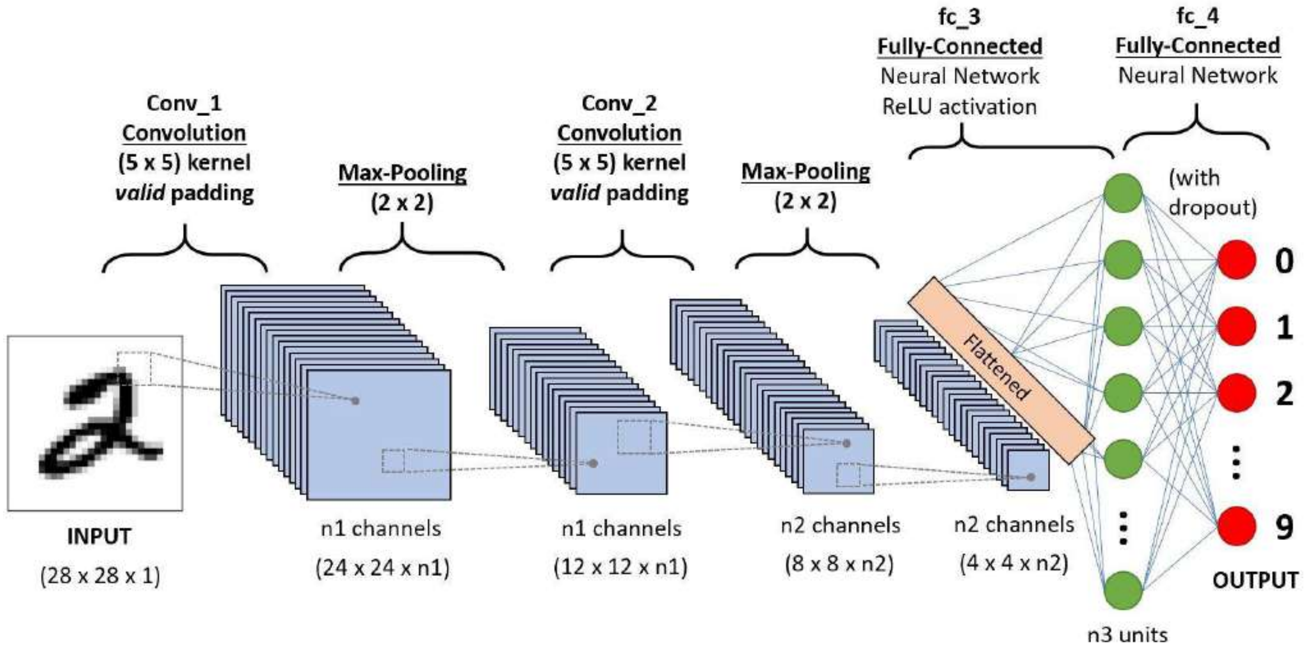
1 Objective

The objective of this project to implement the LeNet Architecture using verilog.

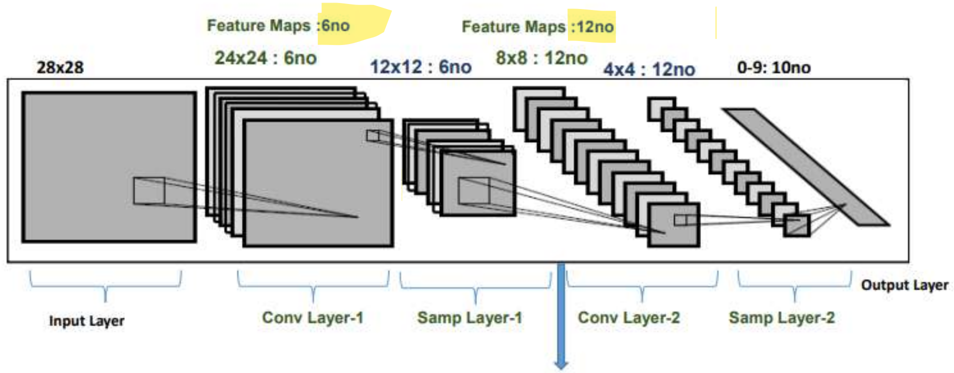
2 Introduction

2.1 LeNet

LeNet is a big breakthrough in the world of image recognition. It is one of the oldest convolution neural networks that was introduced by Yann LeCunn back in 1995 in his research paper. During those days he came up with this LeNet Model to find the handwritten digits representing the Zip codes of the US postal service.



Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with
dropout)
Flattened
0
1
2
9
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
OUTPUT



In General :

Input Neurons	: 12x12x6 = 864
Output Neurons	: 8x8x12 = 768
No of weights	: 864x768 =663552
No of Biases	: 12x8x8 = 768

In CNN :

Input Neurons	: 12x12x6 = 864
Output Neurons	: 8x8x12 = 768
No of weights	: 12x6x5x5 = 1800
No of Biases	: 12

6kernels of 5x5 size for each of 12output maps

Total No of kernels: 12x6=72 each 5x5 size

2.1.1 Components of LeNet Structure:

- Input image of a 28x28 size.
- Convolutional layer used with a kernel size(5x5) and padding.
- Filters used in Convolutional layers as per requirement.

- Relu layer

- Average Max pooling is used with specific size, strides, and padding.

2.1.2 Convolution layer

It’s constructed using multiple convolutions and average pooling layers. We take an input greyscale image of size 28x28 consisting of digits as images. We introduce a kernel of size 3x3 with padding as 0 and convolve it with the input image. We use 1 filters or kernels to generate the convolutional layer of 26x26x1. The image stride is taken as 1.

Input image = 28x28 Kernel size = 3x3

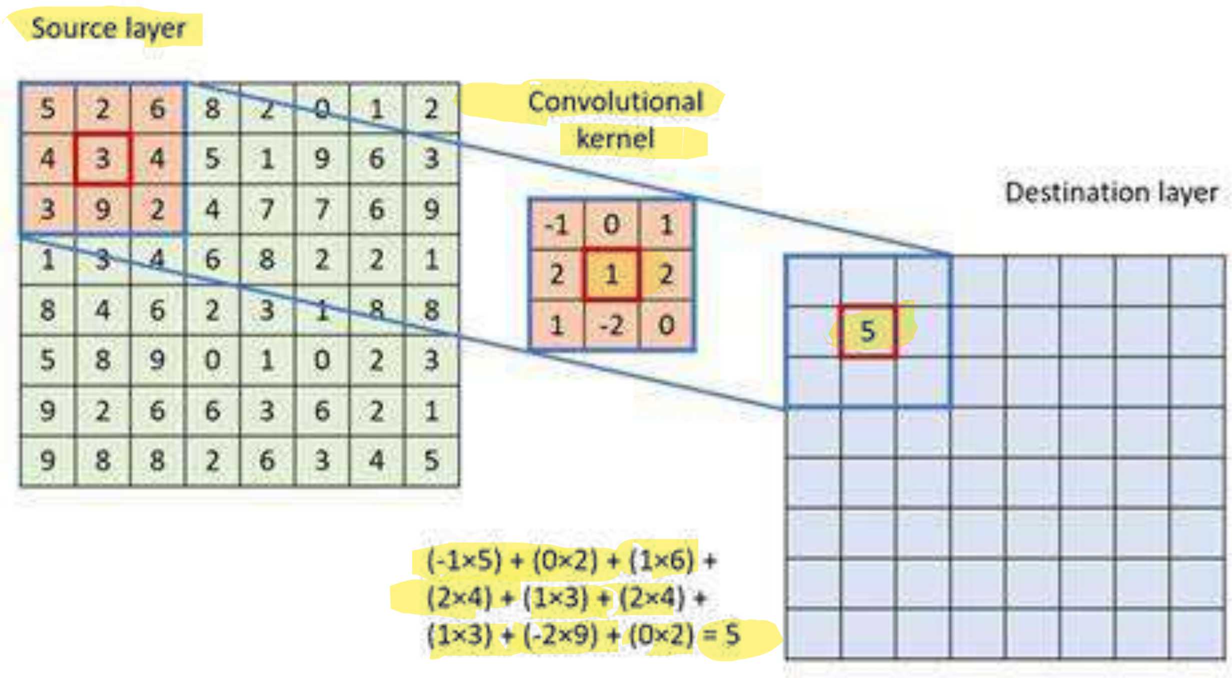
No of kernels = 1

Padding = 0

Stride = 1

So size of conv1 = [n+2p-f+1] / s = [28+0–3+1/1] = 26

Hence the conv1 = 26x26x1

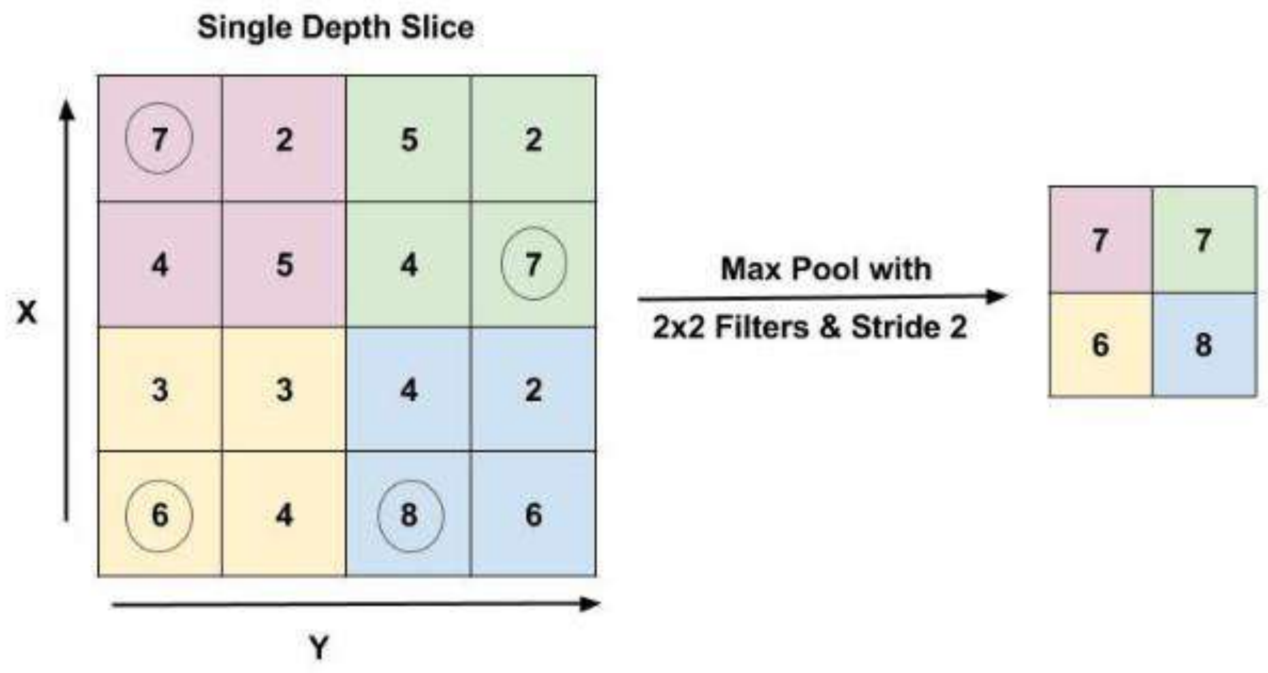


2.1.3 Pooling Layer

Pooling is a feature commonly imbibed into Convolutional Neural Network (CNN) architectures. The main idea behind a pooling layer is to "accumulate" features from maps generated by convolving a filter over an image. Formally, its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. The most common form of pooling is max pooling.

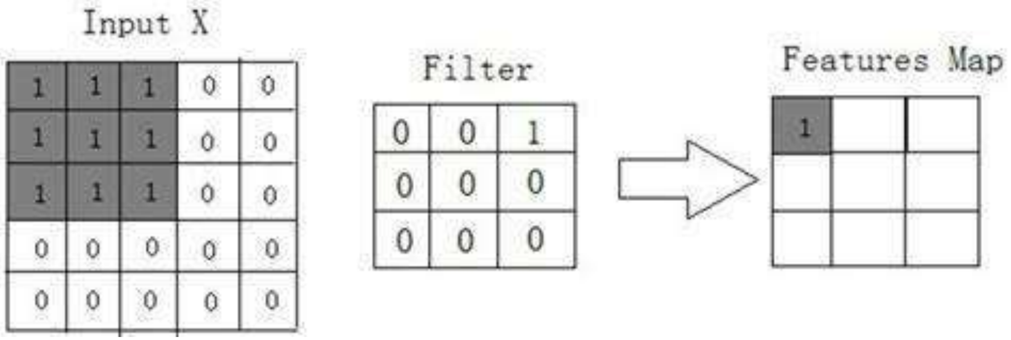
2.1.4 Max Pooling Layer

Max pooling picks the highest value from 2x2 matrix, and replaces the 2x2 with that highest value. here stride is equal to 2. Max pooling is done to in part to help over-fitting by providing an abstracted form of the representation. As well, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation invariance to the internal representation. Max pooling is done by applying a max filter to (usually) non-overlapping subregions of the initial representation. The other forms of pooling are: average, general.



2.1.5 ReLu Layer

We are using most widely used activation function called ReLU (Rectified Linear Unit) and it is preferred as the default choice for Neural Networks. It removes all the negative values from the filtered images and replace them with zeros.



3 Design Approach

To implement the convolution layer using following approach has been taken:

- First We have designed Lenet Architecture in Python.
- We have used Tensor flow to generate weights and update the weights
- We store the value of weights in ROM
- We have create one module that will take 25 weights(kernel) and 25 input(Image) and output as one value
- For multiplying(dot product) the 5x5 matrix of the input 28x28 and the kernal 5x5 matrix, a Verilog code has been written.
- A verlog code is written where the multiplication code is instantiated for calculating the values .

- Similarly We have designed The Max Pooling layer .For this ,A Verilog code is written in which module name max poll is taking four input and one output(comparing all the value which one is maximum it is giving output.
-

3.1 Implementation of Lenet Architecture in Python Using Tensorflow.


```
import tensorflow as tf
import keras as keras
```

```
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
```

```
x_train.shape
```

```
(60000, 28, 28)
```

sample

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=6, kernel_size=(5, 5), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(filters=16, kernel_size=(5, 5), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=120, activation='relu'),
    tf.keras.layers.Dense(units=84, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')
])
```

sequential structure

→ give to ANN

→ deep N/W, so gradient can be diminished

$\max(f)$

→ multiple class, 10

$f = \max, \; x > 0$

$0, \; x < 0$

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 24, 24, 6)         156

 max_pooling2d (MaxPooling2D  (None, 12, 12, 6)        0
 )

 conv2d_1 (Conv2D)           (None, 8, 8, 16)          2416

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 16)         0
 2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 120)               30840

 dense_1 (Dense)             (None, 84)                10164

 dense_2 (Dense)             (None, 10)                850

=================================================================
Total params: 44,426
Trainable params: 44,426
Non-trainable params: 0
_________________________________________________________________
```

→ store in ROM
filter's weig
take from pyth

```
c1=model.layers[0].get_weights()[0]
```

```
import numpy as np
c1=np.array(c1)
```

```
np.min(c1)
```

```
-0.18441719
```

```
1/np.sqrt(175)
```

```
0.07559289460184544
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

→ better in accuracy

```
model.layers[0].get_weights()[0]
```

```
array([[[[ 0.14488555, -0.06074834, -0.17811978, -0.15754612,
          -0.15390977,  0.08955278]],

        [[-0.15237969,  0.06597529,  0.16895168, -0.05155353,
          -0.1719914 ,  0.16773714]],

        [[ 0.07100953, -0.00092161, -0.11273857, -0.02113454,
```

```
   -0.01282665,  0.1501381 ]],

  [[-0.09702267, -0.11181794, -0.09235991,  0.05235502,
   -0.06786727, -0.05358776]],

  [[ 0.02299272, -0.11935521, -0.13209635, -0.07491823,
    0.1144488 ,  0.14425798]]],


 [[[-0.06491189, -0.04609518, -0.11396814,  0.14550658,
    0.05697429, -0.17216489]],

  [[ 0.02366771,  0.09389015, -0.05640599, -0.15808657,
    0.14515312,  0.0660343 ]],

  [[ 0.01223508,  0.02025108,  0.06177267, -0.16304466,
   -0.1535467 ,  0.04261854]],

  [[ 0.15677787, -0.15513241, -0.14413184,  0.12458573,
   -0.06259234, -0.18441719]],

  [[ 0.15664695,  0.07016928, -0.10614802,  0.0576153 ,
   -0.15393052,  0.09363653]]],


 [[[ 0.02710019, -0.09788428,  0.15383784, -0.09753101,
    0.09660788,  0.06520735]],

  [[ 0.15677144, -0.09341638,  0.17318614, -0.05552898,
   -0.03538018,  0.05597807]],

  [[-0.01025334,  0.18190317, -0.0713975 , -0.04127873,
   -0.05274104,  0.08341943]],

  [[-0.17334254, -0.14851351, -0.15497375, -0.13077852,
   -0.17453769, -0.12262174]],

  [[ 0.03056455, -0.12915958, -0.16689897, -0.13764353,
    0.10691006,  0.17367844]]],


 [[[-0.14007835, -0.06999721, -0.0436046 , -0.03706248,
   -0.12109427,  0.10004871]],

  [[-0.1091763 , -0.009113  , -0.03284974, -0.129911  ,
   -0.1503637 , -0.05377239]],

  [[ 0.05879812,  0.04231544,  0.03911838,  0.13170283,
   -0.15990981,  0.0898429 ]],
```

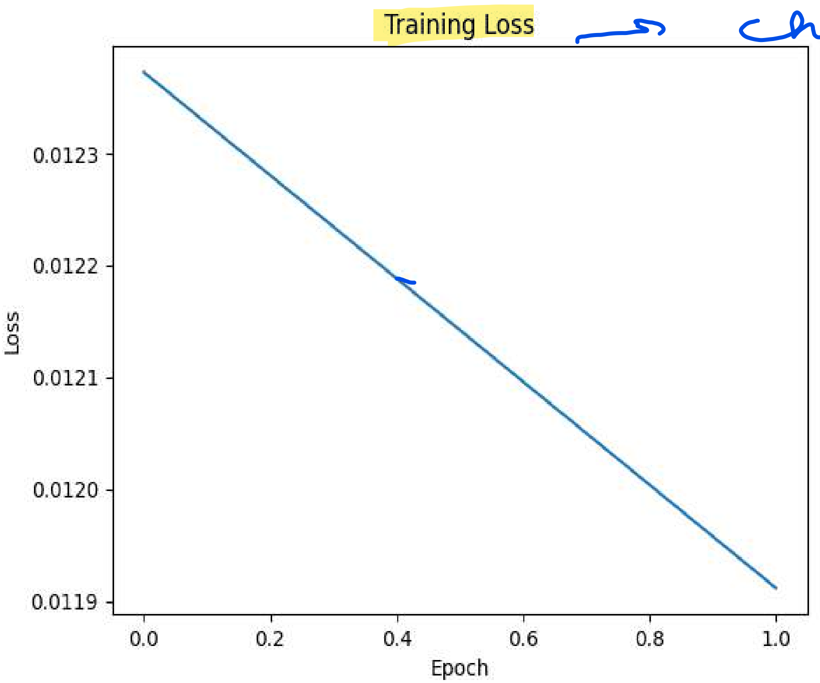
```
num=[]
for i in range(c2.shape[0]):
  num.append(dcb1(c2[i]))

for i in range(150):
    print("rom[",i,"]=16'b",num[i],";", sep="")
```

Rom

```
rom[0]=16'b0000011110100010;
rom[1]=16'b0000001011001111;
rom[2]=16'b1000001101101001;
rom[3]=16'b1000010011011001;
rom[4]=16'b1000110010011111;
rom[5]=16'b0000000100110011;
rom[6]=16'b1000111000111010;
rom[7]=16'b1000000001110110;
rom[8]=16'b0000011100010110;
rom[9]=16'b0000111001001110;
rom[10]=16'b1000001100111101;
rom[11]=16'b0000110110100110;
rom[12]=16'b0000000100100110;
rom[13]=16'b1000010111111100;
rom[14]=16'b1001100001000000;
rom[15]=16'b0001011111110110;
rom[16]=16'b0000110001101011;
rom[17]=16'b0000101110100011;
rom[18]=16'b1000011001110110;
rom[19]=16'b1000011000110001;
rom[20]=16'b1001001111001000;
rom[21]=16'b0001010101010000;
rom[22]=16'b1000000100001110;
rom[23]=16'b1000011100001111;
rom[24]=16'b0000011110110010;
rom[25]=16'b1000011010101010;
rom[26]=16'b1000110110101010;
rom[27]=16'b0000101000101100;
rom[28]=16'b1000011000100011;
rom[29]=16'b0000101000011110;
rom[30]=16'b1000101001001101;
rom[31]=16'b0000010000001000;
rom[32]=16'b1000001000101111;
rom[33]=16'b0000010111001011;
rom[34]=16'b1001101101110110;
rom[35]=16'b1000111000011000;
rom[36]=16'b1000010000010101;
rom[37]=16'b1000101010111111;
rom[38]=16'b1000000111100101;
rom[39]=16'b1000000110000010;
rom[40]=16'b1001111110111111;
rom[41]=16'b0000011110000100;
rom[42]=16'b1000000100101111;
rom[43]=16'b1000101111001111;
rom[44]=16'b0000000000010011;
rom[45]=16'b0000000111111101;
rom[46]=16'b1011011101110001;
rom[47]=16'b0000101010110001;
rom[48]=16'b0000110110110011;
rom[49]=16'b1000000101110110;
rom[50]=16'b1000110101110011;
rom[51]=16'b0001010000001001;
rom[52]=16'b1001011110110100;
rom[53]=16'b1000111001111111;
rom[54]=16'b0000101101110010;
rom[55]=16'b0001001101110110;
rom[56]=16'b1001011111101010;
rom[57]=16'b0001011000111101;
```

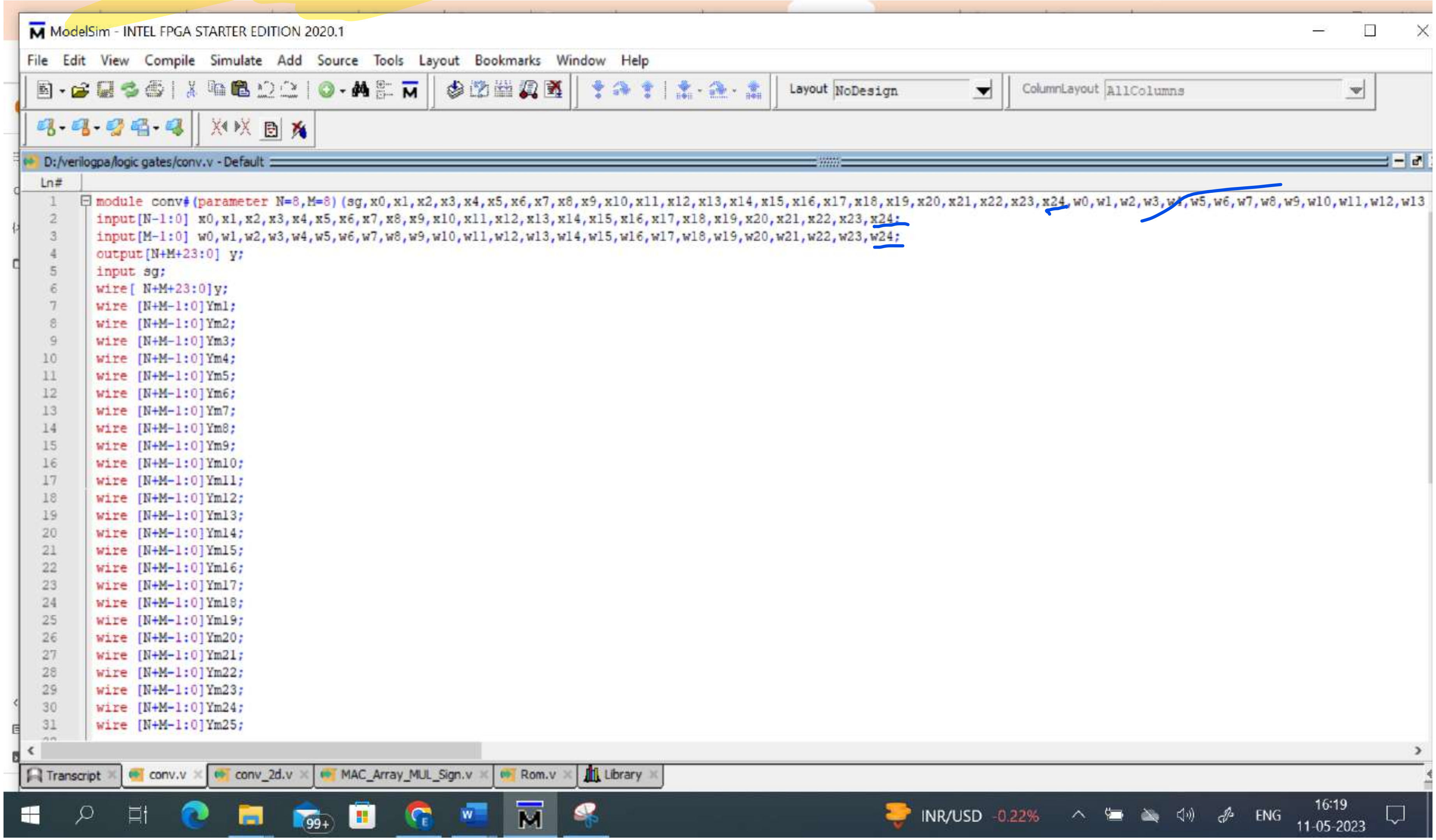
```python
import matplotlib.pyplot as plt

# Train the model and collect the history
history = model.fit(x_train, y_train, epochs=2, validation_data=(x_test, y_test))

# Plot the training loss
plt.plot(history.history['loss'])
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```

```
Epoch 1/2
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0124 - accuracy: 0.9959 - val_loss: 0.0446 - val_accuracy: 0.9895
Epoch 2/2
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0119 - accuracy: 0.9960 - val_loss: 0.0417 - val_accuracy: 0.9901
```

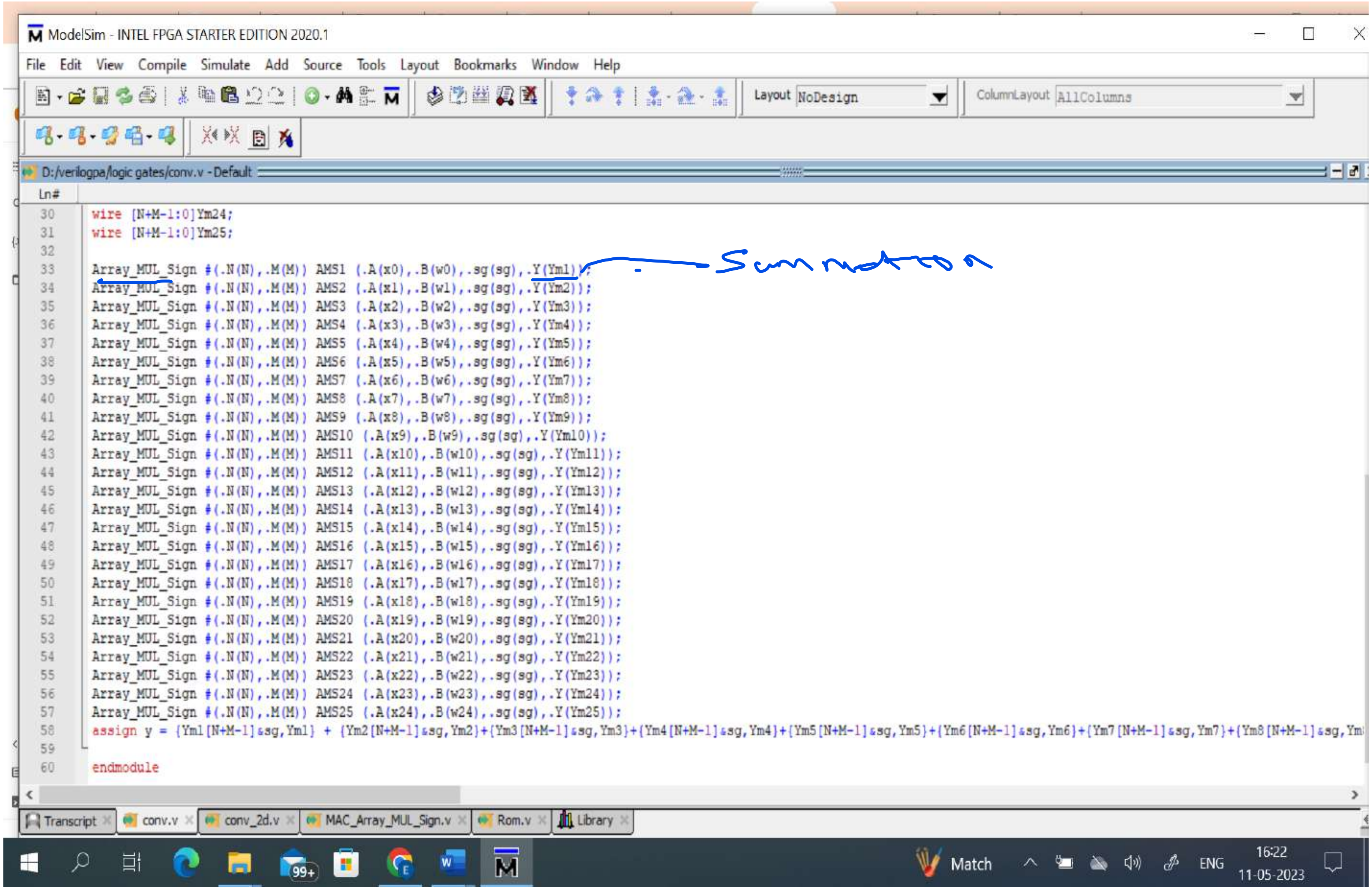


→ check whether weights are good or not

3.2 Implementation Of Convolution module

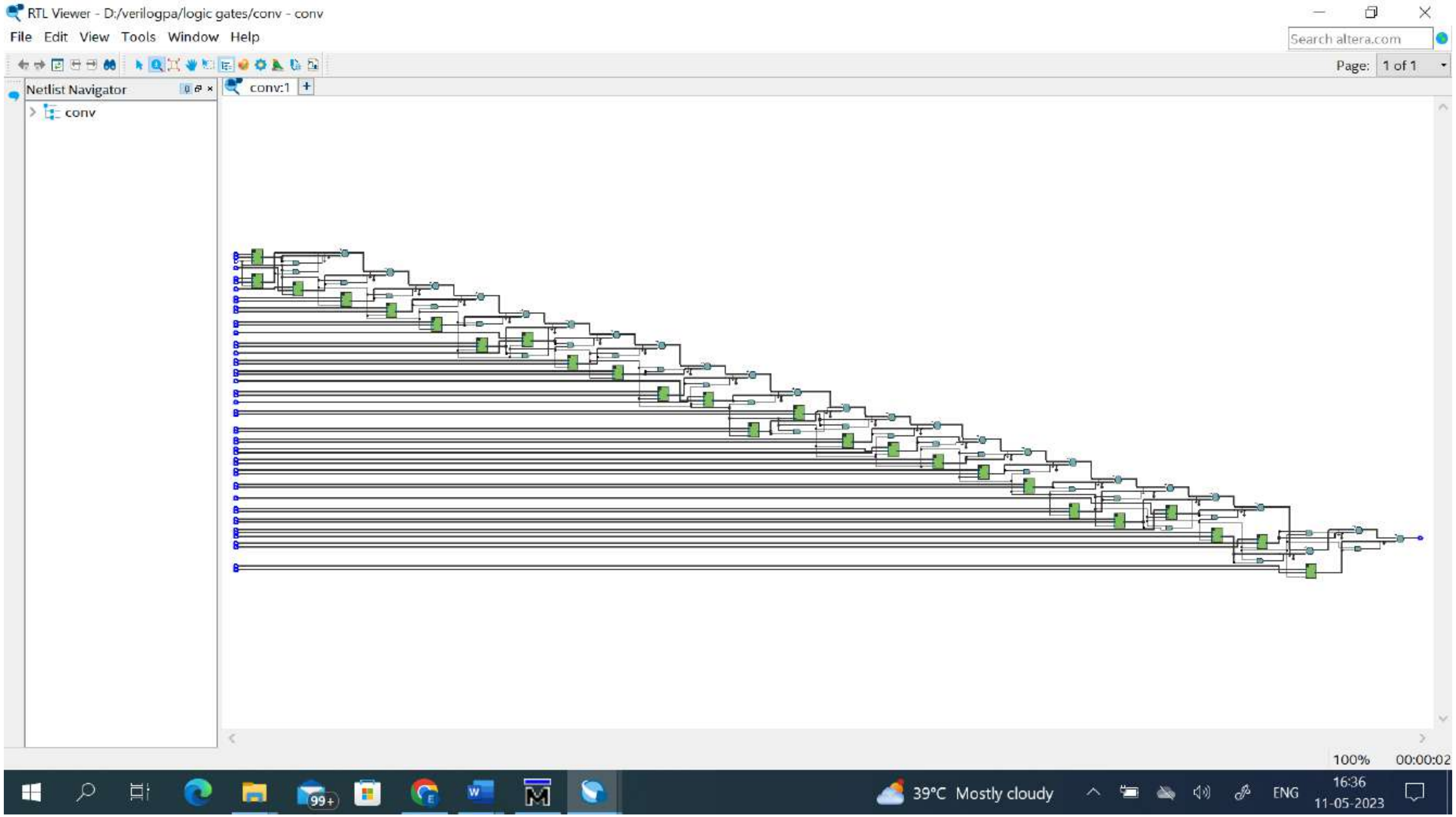


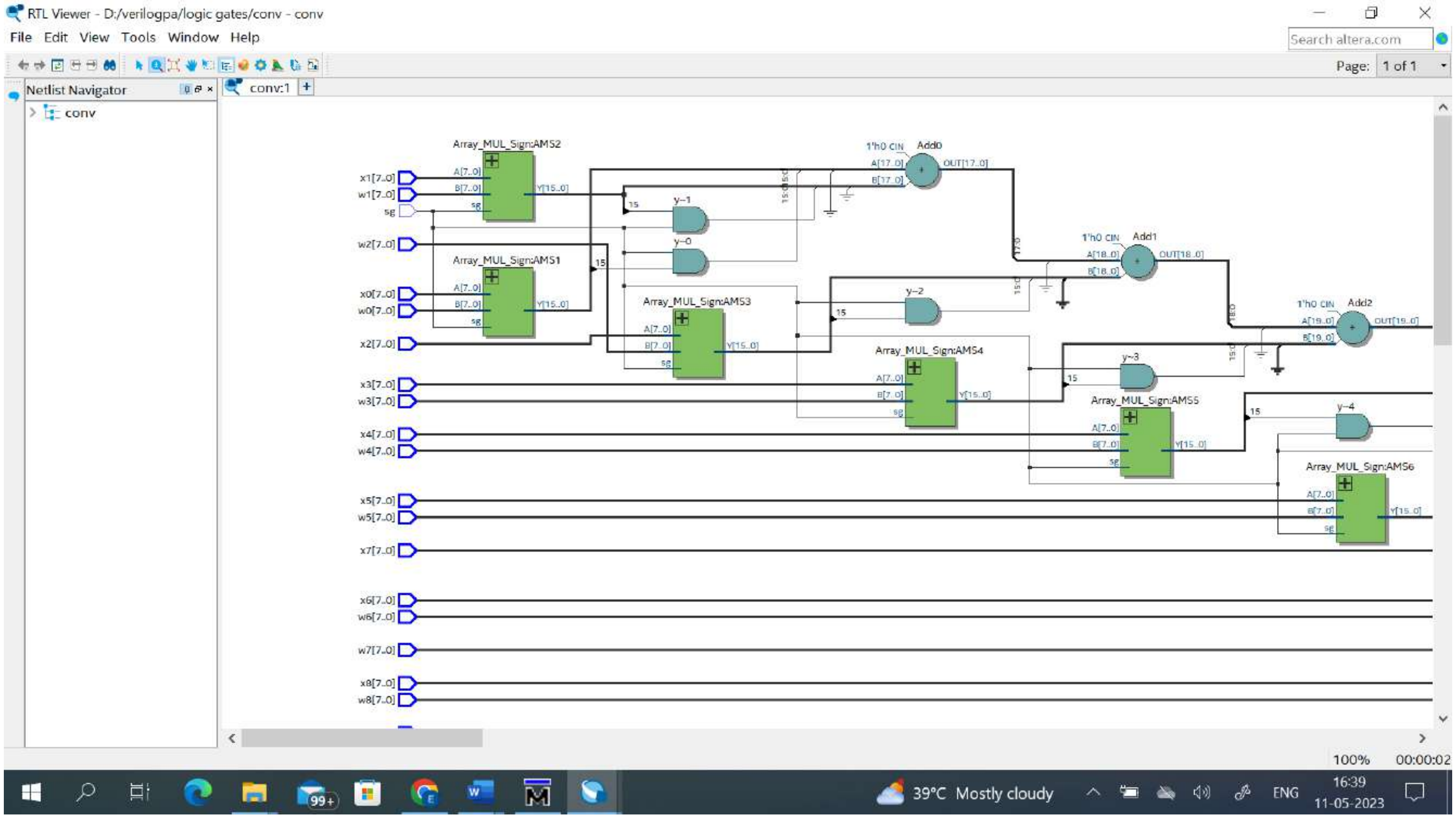
```
module conv#(parameter N=8,M=8)(sg,x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x21,x22,x23,x24,w0,w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13
input[N-1:0] x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x21,x22,x23,x24;
input[M-1:0] w0,w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14,w15,w16,w17,w18,w19,w20,w21,w22,w23,w24;
output[N+M+23:0] y;
input sg;
wire[ N+M+23:0]y;
wire [N+M-1:0]Ym1;
wire [N+M-1:0]Ym2;
wire [N+M-1:0]Ym3;
wire [N+M-1:0]Ym4;
wire [N+M-1:0]Ym5;
wire [N+M-1:0]Ym6;
wire [N+M-1:0]Ym7;
wire [N+M-1:0]Ym8;
wire [N+M-1:0]Ym9;
wire [N+M-1:0]Ym10;
wire [N+M-1:0]Ym11;
wire [N+M-1:0]Ym12;
wire [N+M-1:0]Ym13;
wire [N+M-1:0]Ym14;
wire [N+M-1:0]Ym15;
wire [N+M-1:0]Ym16;
wire [N+M-1:0]Ym17;
wire [N+M-1:0]Ym18;
wire [N+M-1:0]Ym19;
wire [N+M-1:0]Ym20;
wire [N+M-1:0]Ym21;
wire [N+M-1:0]Ym22;
wire [N+M-1:0]Ym23;
wire [N+M-1:0]Ym24;
wire [N+M-1:0]Ym25;
```



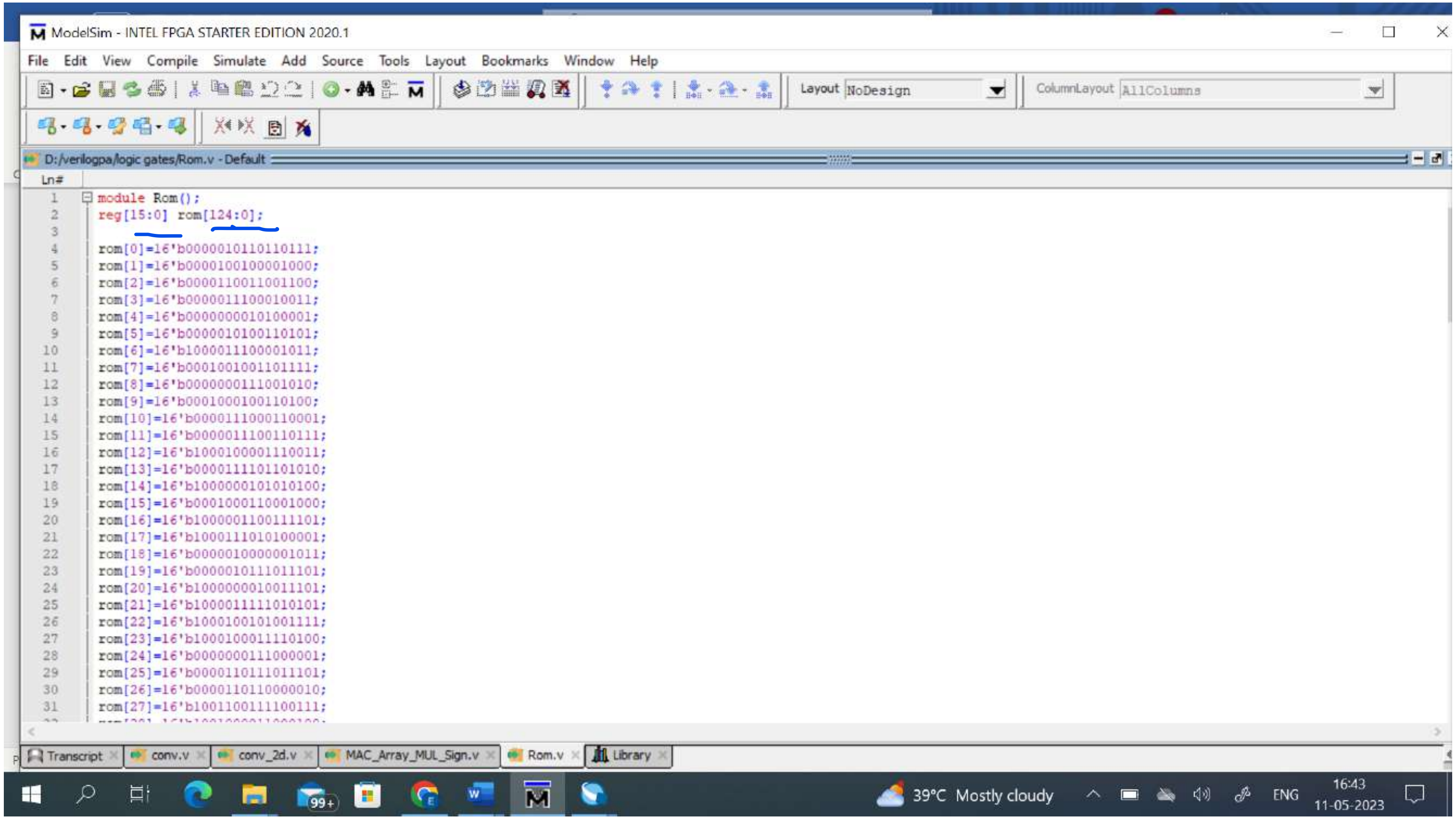

```
wire [N+M-1:0]Ym24;
wire [N+M-1:0]Ym25;

Array_MUL_Sign #(.N(N),.M(M)) AMS1 (.A(x0),.B(w0),.sg(sg),.Y(Ym1));
Array_MUL_Sign #(.N(N),.M(M)) AMS2 (.A(x1),.B(w1),.sg(sg),.Y(Ym2));
Array_MUL_Sign #(.N(N),.M(M)) AMS3 (.A(x2),.B(w2),.sg(sg),.Y(Ym3));
Array_MUL_Sign #(.N(N),.M(M)) AMS4 (.A(x3),.B(w3),.sg(sg),.Y(Ym4));
Array_MUL_Sign #(.N(N),.M(M)) AMS5 (.A(x4),.B(w4),.sg(sg),.Y(Ym5));
Array_MUL_Sign #(.N(N),.M(M)) AMS6 (.A(x5),.B(w5),.sg(sg),.Y(Ym6));
Array_MUL_Sign #(.N(N),.M(M)) AMS7 (.A(x6),.B(w6),.sg(sg),.Y(Ym7));
Array_MUL_Sign #(.N(N),.M(M)) AMS8 (.A(x7),.B(w7),.sg(sg),.Y(Ym8));
Array_MUL_Sign #(.N(N),.M(M)) AMS9 (.A(x8),.B(w8),.sg(sg),.Y(Ym9));
Array_MUL_Sign #(.N(N),.M(M)) AMS10 (.A(x9),.B(w9),.sg(sg),.Y(Ym10));
Array_MUL_Sign #(.N(N),.M(M)) AMS11 (.A(x10),.B(w10),.sg(sg),.Y(Ym11));
Array_MUL_Sign #(.N(N),.M(M)) AMS12 (.A(x11),.B(w11),.sg(sg),.Y(Ym12));
Array_MUL_Sign #(.N(N),.M(M)) AMS13 (.A(x12),.B(w12),.sg(sg),.Y(Ym13));
Array_MUL_Sign #(.N(N),.M(M)) AMS14 (.A(x13),.B(w13),.sg(sg),.Y(Ym14));
Array_MUL_Sign #(.N(N),.M(M)) AMS15 (.A(x14),.B(w14),.sg(sg),.Y(Ym15));
Array_MUL_Sign #(.N(N),.M(M)) AMS16 (.A(x15),.B(w15),.sg(sg),.Y(Ym16));
Array_MUL_Sign #(.N(N),.M(M)) AMS17 (.A(x16),.B(w16),.sg(sg),.Y(Ym17));
Array_MUL_Sign #(.N(N),.M(M)) AMS18 (.A(x17),.B(w17),.sg(sg),.Y(Ym18));
Array_MUL_Sign #(.N(N),.M(M)) AMS19 (.A(x18),.B(w18),.sg(sg),.Y(Ym19));
Array_MUL_Sign #(.N(N),.M(M)) AMS20 (.A(x19),.B(w19),.sg(sg),.Y(Ym20));
Array_MUL_Sign #(.N(N),.M(M)) AMS21 (.A(x20),.B(w20),.sg(sg),.Y(Ym21));
Array_MUL_Sign #(.N(N),.M(M)) AMS22 (.A(x21),.B(w21),.sg(sg),.Y(Ym22));
Array_MUL_Sign #(.N(N),.M(M)) AMS23 (.A(x22),.B(w22),.sg(sg),.Y(Ym23));
Array_MUL_Sign #(.N(N),.M(M)) AMS24 (.A(x23),.B(w23),.sg(sg),.Y(Ym24));
Array_MUL_Sign #(.N(N),.M(M)) AMS25 (.A(x24),.B(w24),.sg(sg),.Y(Ym25));
assign y = {Ym1[N+M-1]&sg,Ym1} + {Ym2[N+M-1]&sg,Ym2}+{Ym3[N+M-1]&sg,Ym3}+{Ym4[N+M-1]&sg,Ym4}+{Ym5[N+M-1]&sg,Ym5}+{Ym6[N+M-1]&sg,Ym6}+{Ym7[N+M-1]&sg,Ym7}+{Ym8[N+M-1]&sg,Ym

endmodule
```

RTL View of Convolution

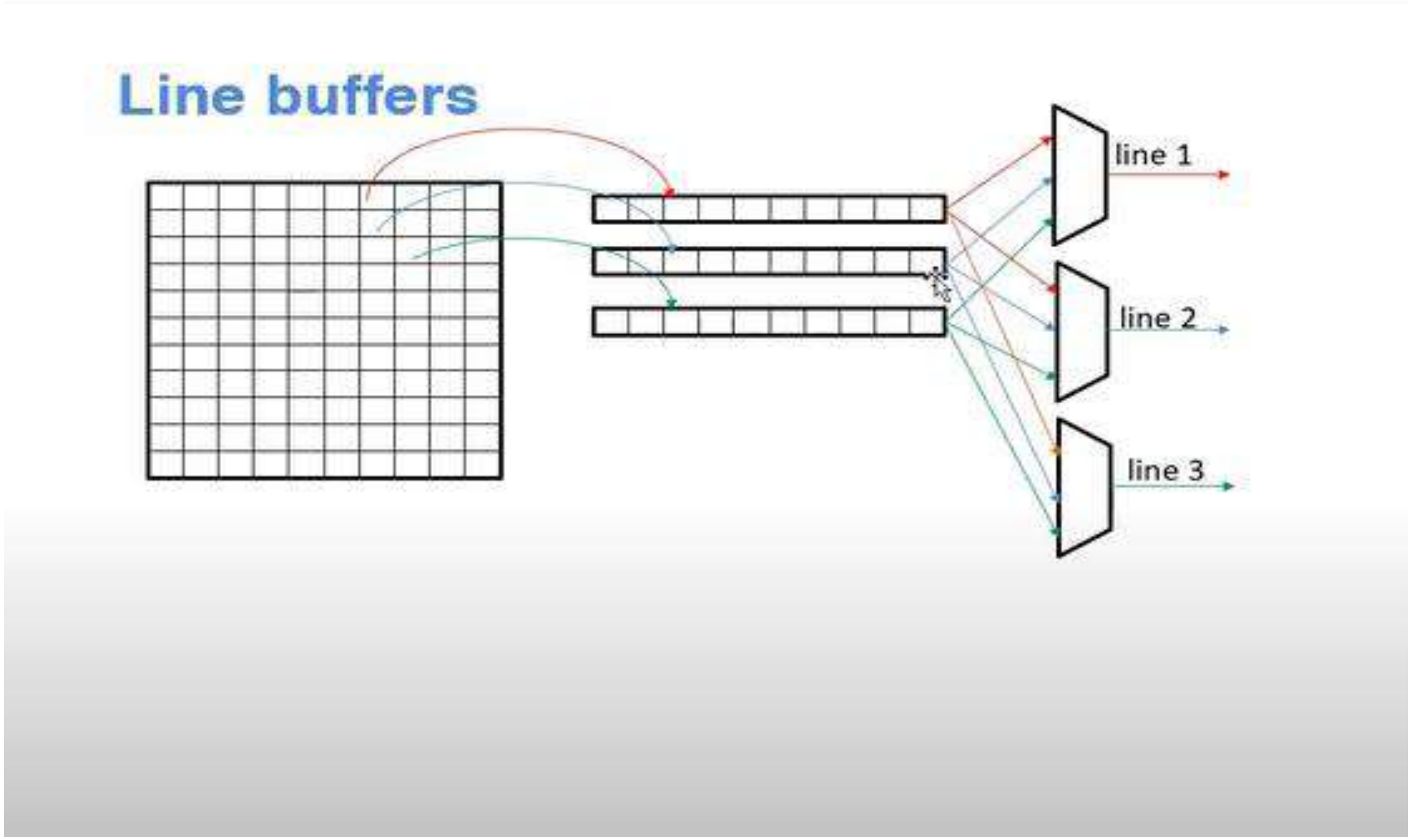



RTL Viewer - D:/verilogpa/logic gates/conv - conv
File Edit View Tools Window Help
Page: 1 of 1
Netlist Navigator
conv
conv:1
Array_MUL_Sign:AMS2
Array_MUL_Sign:AMS1
Array_MUL_Sign:AMS3
Array_MUL_Sign:AMS4
Array_MUL_Sign:AMS5
Array_MUL_Sign:AMS6
Add0
Add1
Add2
x1[7..0]
w1[7..0]
w2[7..0]
x0[7..0]
w0[7..0]
x2[7..0]
x3[7..0]
w3[7..0]
x4[7..0]
w4[7..0]
x5[7..0]
w5[7..0]
x7[7..0]
x6[7..0]
w6[7..0]
w7[7..0]
x8[7..0]
w8[7..0]
100%
00:00:02
39°C Mostly cloudy
ENG
16:39
11-05-2023

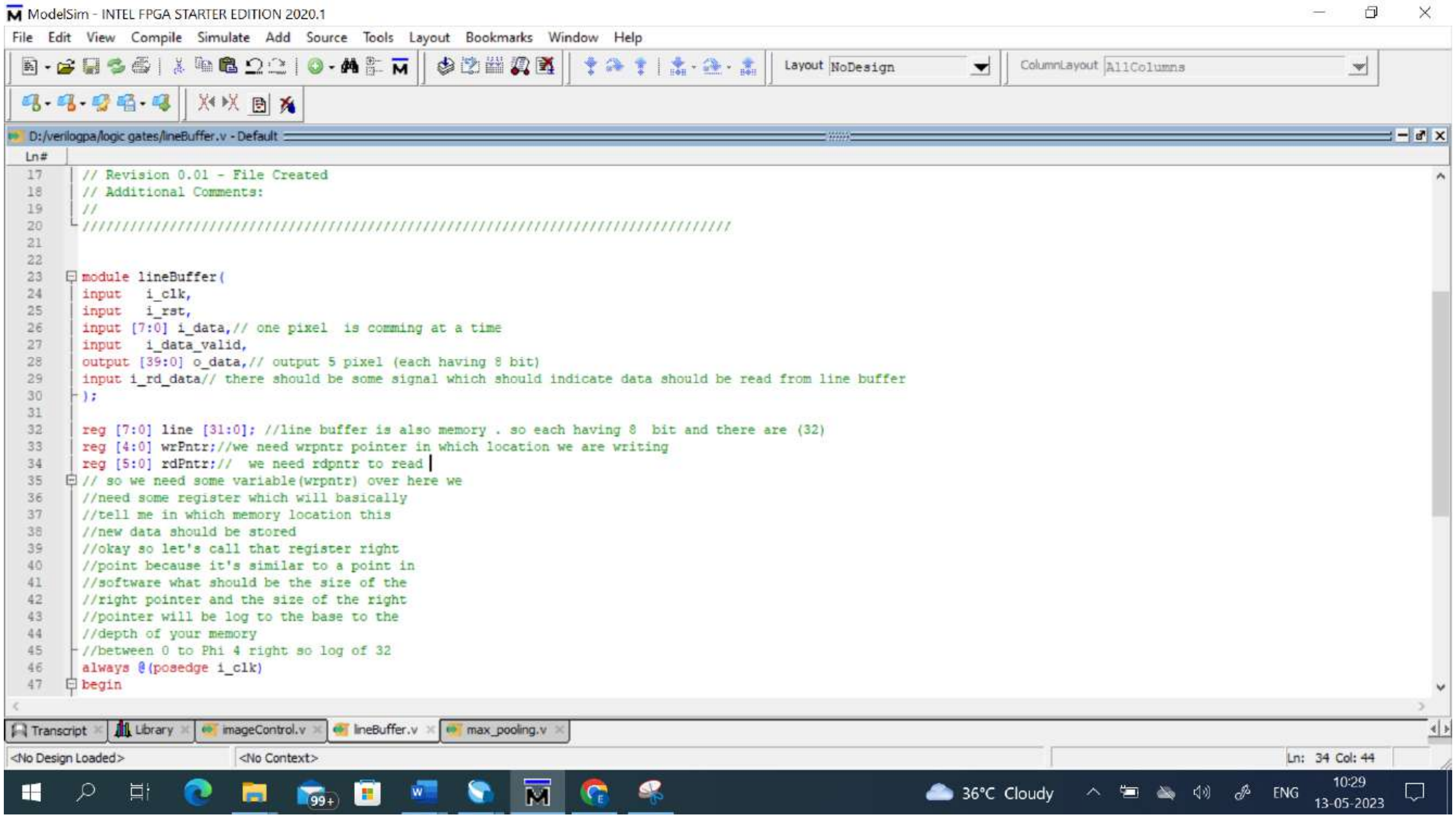
3.2 Store Weight in Rom



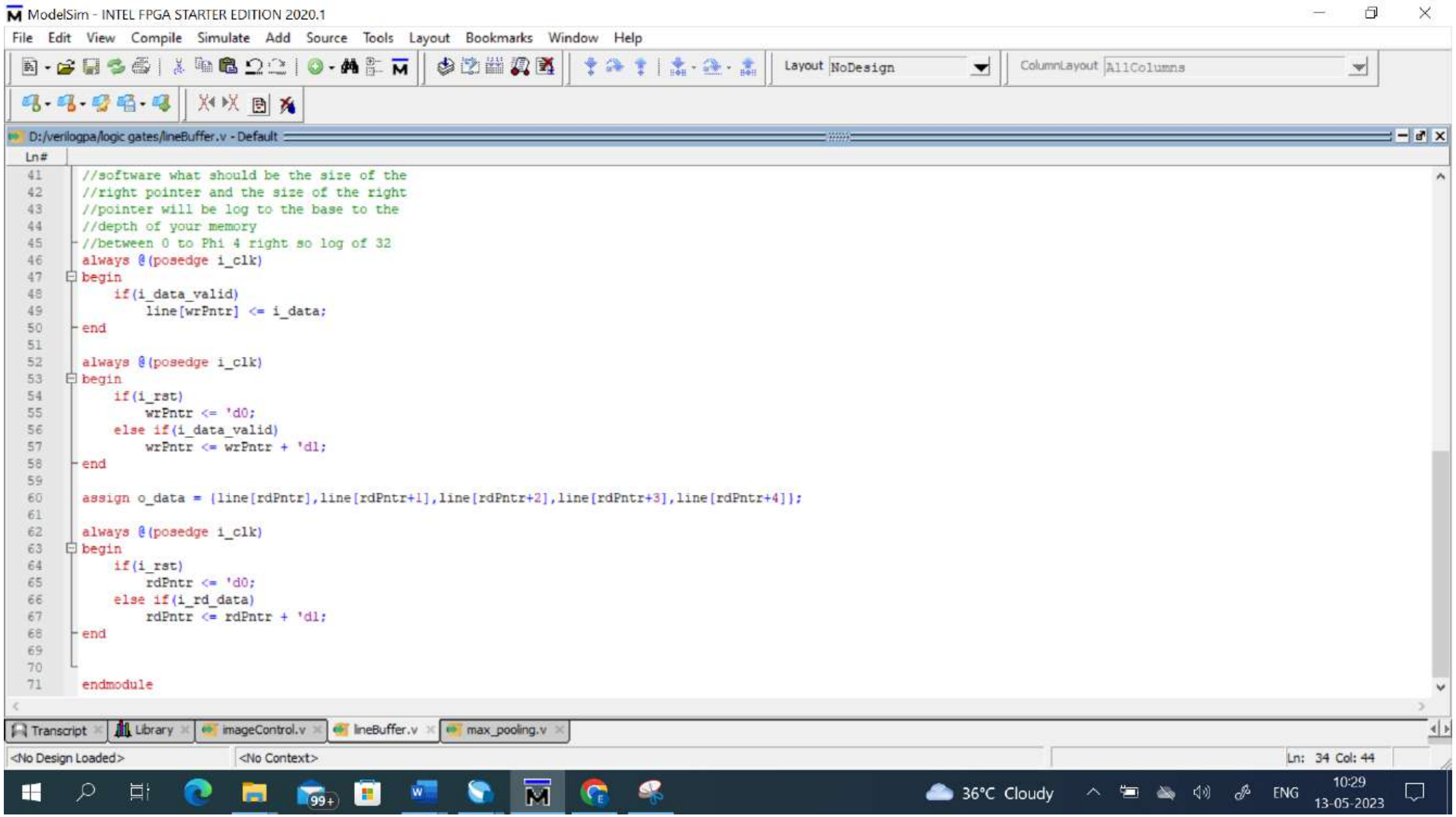
3.3 Design of Line Buffer



Verilog code for Line Buffer:



```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module lineBuffer(
input   i_clk,
input   i_rst,
input [7:0] i_data,// one pixel  is comming at a time
input   i_data_valid,
output [39:0] o_data,// output 5 pixel (each having 8 bit)
input i_rd_data// there should be some signal which should indicate data should be read from line buffer
);

reg [7:0] line [31:0]; //line buffer is also memory . so each having 8  bit and there are (32)
reg [4:0] wrPntr;//we need wrpntr pointer in which location we are writing
reg [5:0] rdPntr;//  we need rdpntr to read
// so we need some variable(wrpntr) over here we
//need some register which will basically
//tell me in which memory location this
//new data should be stored
//okay so let's call that register right
//point because it's similar to a point in
//software what should be the size of the
//right pointer and the size of the right
//pointer will be log to the base to the
//depth of your memory
//between 0 to Phi 4 right so log of 32
always @(posedge i_clk)
begin
```


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
D:/verilogpa/logic gates/lineBuffer.v - Default
Ln#
//software what should be the size of the
//right pointer and the size of the right
//pointer will be log to the base to the
//depth of your memory
//between 0 to Phi 4 right so log of 32
always @(posedge i_clk)
begin
if(i_data_valid)
line[wrPntr] <= i_data;
end
always @(posedge i_clk)
begin
if(i_rst)
wrPntr <= 'd0;
else if(i_data_valid)
wrPntr <= wrPntr + 'd1;
end
assign o_data = {line[rdPntr],line[rdPntr+1],line[rdPntr+2],line[rdPntr+3],line[rdPntr+4]};
always @(posedge i_clk)
begin
if(i_rst)
rdPntr <= 'd0;
else if(i_rd_data)
rdPntr <= rdPntr + 'd1;
end
endmodule
Transcript
Library
imageControl.v
lineBuffer.v
max_pooling.v
<No Design Loaded>
<No Context>
Ln: 34 Col: 44
36°C Cloudy
ENG
10:29
13-05-2023

Application of LeNet

LeNet, also known as LeNet-5, is a convolutional neural network (CNN) architecture. It was originally designed for **handwritten digit recognition** but has since found applications in various areas of **computer vision** and **pattern recognition.** Here are some common applications of LeNet:

1. Handwritten Digit Recognition: LeNet was initially developed for recognizing handwritten digits in postal addresses and zip codes. Its architecture, consisting of convolutional layers, pooling layers, and fully connected layers, makes it well-suited for this task.

2. Optical Character Recognition (OCR): LeNet's ability to recognize individual characters makes it suitable for OCR applications. It can be used to extract text from scanned documents, license plates, or any other image-based text.

3. Facial Recognition: LeNet can be used for facial recognition tasks, such as identifying individuals in images or videos. By training the network on a dataset of labeled faces, it can learn to recognize facial features and classify images accordingly.

4. Object Recognition: LeNet can be applied to general object recognition tasks, where the goal is to classify objects into predefined categories. It can be used for tasks such as identifying vehicles, animals, or everyday objects in images.

5. Traffic Sign Recognition: LeNet's architecture, with its ability to capture local image features, can be utilized for recognizing and classifying traffic signs. This is important for autonomous driving systems and traffic management applications.

6. Medical Image Analysis: LeNet can be employed in medical image analysis tasks, such as identifying abnormalities in X-rays, MRIs. It can help automate the process of diagnosing diseases or detecting specific features in medical images.

7. Document Analysis: LeNet can be used for analyzing structured or unstructured documents, such as extracting information from forms, recognizing handwriting, or classifying document types.

These are just a few examples of the applications of LeNet. Its architecture and convolutional operations make **it particularly effective** for tasks involving image analysis, recognition, and classification. However, it's worth noting that since LeNet's inception, many more advanced CNN architectures, such as VGG, ResNet, and Inception, have been developed and are commonly used in modern computer vision applications.