ES204 Digital Systems Project Report 1

Project Title: Implementation of Fully Connected Neural Networks on FPGA

Team Members:

Name	Roll number
Anmol Bhargava	22110027
Anura Mantri	22110144
Mohit Maurya	22110145
Neerja Kasture	22110165

Problem Objective:

Implement a fully connected neural network on a Field-Programmable Gate Array for efficient and real time image classification task. The project aims to leverage the parallel processing capabilities of FPGA architecture in order to explore and utilise the potential of hardware acceleration for neural networks.
We plan to fulfil the following objectives:

- Implement the FCNN model using a Nexys 4 board on the MNIST Digits Dataset.
- Explore optimization techniques such as quantization to minimise resource utilisation and power consumption while maintaining accuracy.
- Implement hardware acceleration modules for critical operations like matrix multiplications and activation functions to enhance performance.
- Implement parameterized code for scalability of neural network.
- Incorporate UART communication for real time prediction of test data
- Evaluate the performance of the fully connected neural network on the FPGA in terms of execution time, resource utilisation, power consumption and model metrics like accuracy, precision, recall etc.

Description of modules:

1. Neuron Module: Represents a single neuron in the neural network. This module computes the weighted sum of inputs, applies the activation function, and adds a bias term.
2. Matrix Multiplication Module: Performs matrix multiplication operation between input data and neuron weights to compute the weighted sum of inputs for each neuron in a layer
3. Activation Function (RELU) module:Implements the Rectified Linear Unit (ReLU) activation function, which introduces non-linearity by setting negative values to zero
4. Activation Function (log softmax) module:Implements the log softmax activation function, which converts raw output values into a probability distribution over multiple classes
5. Fully Connected Layer module: Represents a layer of neurons with dense connectivity to the previous layer. It aggregates multiple neuron instances and manages their connections, weights, and biases
6. Forward pass module: It passes input data through each layer, computes neuron outputs, applies activation functions, and generates the final output prediction.
7. Prediction module: Generates the final output prediction by selecting maximum index from the output layer typically used during inference.
8. Top Module: Integrates the neural network modules with the BRAM and UART communication to produce required output.

Project Timeline:

Phase 1 (4th March - 13th March)

- Successfully establish UART communication between the Nexys4 board and the laptop.
- Understanding BRAM to store the weights determining the parameters for configuring it
- Testing BRAM for storing data
- Define a clear objective to be fulfilled by the neural network e.g classification of digits in MNIST dataset
- Implement a fully connected neural network for the above objective in Python in order to obtain a clear understanding of its functioning.
- Decide the specifications of neural network to be implemented on FPGA

Phase 2 (13th March - 20th March)

- Implement the neural network in Verilog. Steps:

- Make neuron, hidden layer
- Activation functions
- Create testbenches for each layer to test individual functionality and demonstrate adequate working through simulations.

Phase 3 (20th March - April first week)

- Implement the neural network in Verilog. Steps:
- Output Layer
- Select output
- Integrate the neural network implementation with the UART communication setup
- Conduct comprehensive testing of the integrated system and fine tune parameters
- Evaluate the performance of the fully connected neural network on the FPGA in terms of execution time, resource utilisation, power consumption and accuracy metrics etc.
- Prepare a report and presentation summarising the project and its outcomes

Week 1 Update:

1. Implementation of the given FCNN model in Python. Model specifications:
    a. fc1 512 neurons, relu activation function
    b. fc2 10 neurons, log softmax activation function

Python Code:

Training the model using PyTorch:

```
import torch
from torchvision import datasets, transforms
import numpy as np
import matplotlib.pyplot as plt

transform = transforms.Compose([
    transforms.ToTensor(),  # Convert images to tensors
    transforms.Normalize((0.5,), (0.5,))  # Normalize pixel values to the
range [-1, 1]
])

trainset = datasets.MNIST('~/.pytorch/MNIST_data/', download=True,
train=True, transform=transform)
testset = datasets.MNIST('~/.pytorch/MNIST_data/', download=True,
train=False, transform=transform)
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms

# Define a neural network model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(784, 512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x): # uses relu activation function
        x = torch.relu(self.fc1(x))
        x = torch.log_softmax(self.fc2(x),dim=1)
        return x

model = NeuralNetwork()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
# Training loop without batches
epochs = 5
for epoch in range(epochs):
    running_loss = 0.0
    for i in range(len(trainset)):
        image = trainset[i][0]  # Get image
        label = trainset[i][1]  # Get label
        image = image.view(1, -1)  # Flatten the image
        optimizer.zero_grad()

        outputs = model(image)
        loss = criterion(outputs, torch.tensor([label]))
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss/len(trainset)}")
```

Obtaining the weights and biases of model

```
weights = []
biases = []

for name, param in model.named_parameters():
    if 'weight' in name:
        weights.append(param.data.numpy())
    elif 'bias' in name:
        biases.append(param.data.numpy())
```

Testing Neural Network using Python

```
from sklearn.metrics import accuracy_score, precision_score, recall_score

def forward_pass(testset,weights,biases):
    true_labels = []
    predicted_labels = []
    for i in range(len(testset)):
        test_data,test_label=testset[i]
        img = test_data.view(1,-1)
        a=np.matmul(weights[0],img.T)+biases[0].reshape(-1,1)
        a=torch.relu(a)
        b=np.matmul(weights[1],a)+biases[1].reshape(-1,1)
        y=torch.log_softmax(b,dim=0)
        y=torch.argmax(y).item()
        true_labels.append(test_label)
        predicted_labels.append(y)

    true_labels = np.array(true_labels)
    predicted_labels = np.array(predicted_labels)
    return true_labels,predicted_labels

true_labels,predicted_labels=forward_pass(testset,weights,biases)

accuracy = accuracy_score(true_labels, predicted_labels)
precision = precision_score(true_labels, predicted_labels,average='macro')
recall = recall_score(true_labels, predicted_labels, average='macro')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

Obtaining new weights with precision INT8 in first layer and INT16 in second layer

```
new_weight=[np.int8(weights[0]*100),np.int16(weights[1]*100)]
new_biases=[np.int8(biases[0]*100),np.int16(biases[1]*100)]
true_labels,predicted_labels=forward_pass(testset,new_weight,new_biases)

accuracy = accuracy_score(true_labels, predicted_labels)
precision = precision_score(true_labels, predicted_labels,
average='macro')
recall = recall_score(true_labels, predicted_labels, average='macro')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

Implemented UART and used it for storing data in bram:

Verilog code:

Transmitter:

```
`timescale 1ns / 1ps

//parameter bit_size= 8;   // same time change reg_index range so it can
accommodate the bit size enter over here
module Transmitter (
    input clk,              // System clock
    input reset,          // Reset signal (active low)
    input [7:0] data_in,   // Data to be transmitted
    input send_data,
    output reg tx_out,
    output reg is_transmitted   // UART transmit output
);

parameter max_baud_count=10417;
parameter IDEAL =2'b00;
parameter start_bit = 2'b01;
parameter DATA =2'b10;
parameter stop_bit = 2'b11;

reg [1:0]STATE;
reg [14:0]Baud_counter=0;
reg [2:0] reg_index=0;  // change here after changing bit_size if needed

always@(posedge clk ,posedge reset)
begin

if (reset)
begin
tx_out <= 1'b1 ;
STATE<=IDEAL;
Baud_counter<=0;
reg_index<=0;
```

```
is_transmitted<=0;
end
else
begin
case(STATE)
IDEAL :
   begin
   tx_out<=1;
   reg_index<=0;
   is_transmitted<=0;
   if (send_data & !is_transmitted)
      begin
         if (Baud_counter < (max_baud_count)/2)
               begin
               Baud_counter <= Baud_counter+1;
               STATE <= IDEAL;
               end
         else
            begin
            Baud_counter<=0;
            STATE <= start_bit;

               end
      end
   else
      begin
      STATE<=IDEAL;
      Baud_counter<=0;
      end

   end
start_bit :
   begin
   if (Baud_counter < ( max_baud_count))
      begin
      is_transmitted <=0;
      Baud_counter <=Baud_counter +1;
```

```
        tx_out <=0;
        reg_index<=0;
        STATE <= start_bit;
        end
    else
        begin
        Baud_counter<=0;
        STATE<=DATA;
        end
    end

DATA :
    begin
    if (Baud_counter < ( max_baud_count))
        begin
        Baud_counter<= Baud_counter+1;
        tx_out <= data_in[reg_index];
        STATE <=DATA;
        end
    else
        begin
        Baud_counter <=0;
        if ( reg_index<7)
            begin
            reg_index <= reg_index+1;
            STATE <=DATA;
            end
        else
            begin
            reg_index <=0;
            STATE <=stop_bit;
            end
         end
    end
stop_bit :
    begin
    if (Baud_counter < ( max_baud_count))
```

```
        begin
        tx_out <= 1;
        Baud_counter <= Baud_counter+1;
        STATE <= stop_bit;
        end

    else
        begin
        is_transmitted <=1;
        Baud_counter <=0;
        STATE <= IDEAL;
        end

    end
default :
begin
    tx_out<=1;
    is_transmitted<=0;
    STATE <= IDEAL;
    Baud_counter <=0;
    reg_index <=0;

end

endcase
end
end
endmodule
```

Receiving:

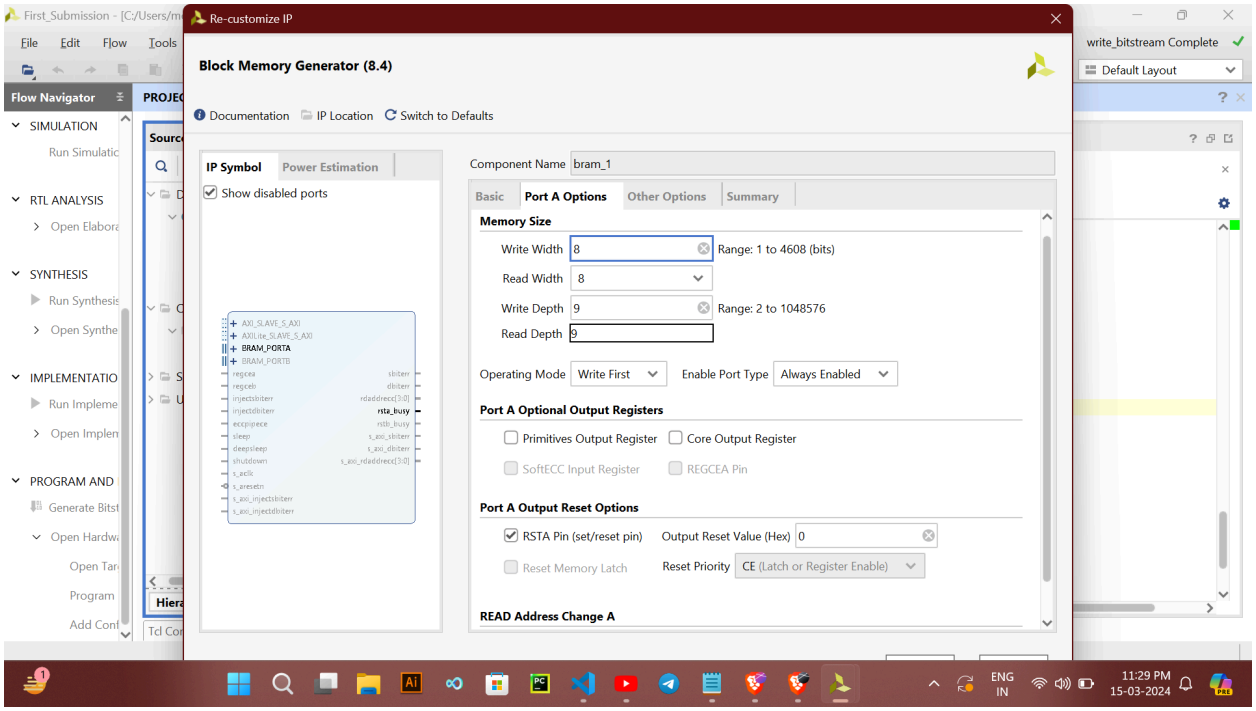
```
`timescale 1ns / 1ps
//parameter bit_size= 8;   // same time change reg_index range so it can accommodate the bit size enter over here

module Receiving (
input clk,
input Rx_data,reset,
output reg [7:0] Data,
output reg is_received
);
parameter max_baud_count=10417;
parameter IDEAL =2'b00;
parameter start_bit = 2'b01;
parameter stop_bit = 2'b10;

reg [1:0]STATE;
reg [14:0]Baud_counter=0;
reg [31:0] reg_index;     // change here after changing bit_size if needed


always@(posedge clk,posedge reset)
begin


   if (reset)
   begin
   Data<=0;
   Baud_counter<=0;
   reg_index<=0;
   is_received<=0;
   end

   else
   begin
   case(STATE)
   IDEAL :
   begin
   is_received <=0;
   reg_index<=0;
   if (Rx_data==0)
      begin
      if (Baud_counter<(max_baud_count)/2)
```

```
        begin
        Baud_counter<=Baud_counter+1;
        is_received <=0;
        STATE <=IDEAL;
        end
    else
        begin
        STATE<=start_bit;
        Baud_counter<=0;
        end
    end
end

start_bit :
  begin
  if (Baud_counter <(max_baud_count))
      begin
      Baud_counter <=Baud_counter+1;
      STATE<=start_bit;
      end
  else
      begin
        Baud_counter <= 0;
        Data[reg_index] <= Rx_data;
      if (reg_index<7)
          begin
          reg_index <= reg_index+1;
          STATE <= start_bit;
          end
      else
          begin
          is_received  <= 0;
          reg_index <= 0;
          Baud_counter <= 0;
          STATE <= stop_bit;
          end
       end
  end

stop_bit :
    begin
    if (Baud_counter <(max_baud_count))
        begin
        Baud_counter <= Baud_counter+1;
```

```
            STATE <= stop_bit;
            end
        else
            begin
            STATE <= IDEAL;
            is_received  <=1;
            end
        end
    default :
    begin
    STATE <= IDEAL;
    Baud_counter <=0;
    reg_index <=0;
    is_received  <=0;
    end
    endcase
    end

 end
// end
endmodule
```

Bram:

We have made a single port Bram of width 8 and depth 9 which is always enabled and has a reset pin.



Main module: This module does the following:

1. takes values from using uart
2. Stores the values in bram
3. Performs summation of values in bram
4. Transmits the sum using uart again to the PC.

Code:

```
`timescale 1ns / 1ps


module UART_BRAM(
input clk,          //input clock
input Rx_data,reset,  //Receving data bit and univeral reset
output tx_out,        //Transmiting  data bit
output reg [7:0]data_out,   //transmiting data
output reg do_transmit,     // flag to transmit data
output reg [3:0]data_count ,  // address to store data
output reg RECEIVED          // flag to confirm  wheather full data is received
sucessfully

);

wire is_received_A;    // flag to confirm whether 8 bits are received
wire [7:0] data_A;     // to store data received fron receiving module
wire [7:0] bram_data;   //to store output data from BRAM


//Instantiation of Receiver Module
Receiving R( .clk(clk), .Rx_data(Rx_data), .reset(reset) , .Data(data_A) ,
.is_received(is_received_A) );



// Instantiation of BRAM
bram_1 BRAM1 (
.clka(clk),                // input wire clka
```

```
.rsta(reset),              // input wire rsta
.wea(RECEIVED),              // input wire [0 : 0] wea
.addra(data_count),         // input wire [3 : 0] addra
.dina(data_A),              // input wire [7 : 0] dina
.douta(bram_data),          // output wire [7 : 0] douta
.rsta_busy()               // output wire rsta_busy
);


// FSM to store recieved data and to proecess the output
always@(posedge clk)
begin



if (reset)
begin
data_count <=0;
RECEIVED<=1;
do_transmit<=0;
data_out<=0;
// check_store<=0;
// check_sum<=0;
end

else if (data_count == 9 & RECEIVED)
begin
RECEIVED<=0;
data_count<=0;
//check_store<=1;

end
else if (data_count == 9 & !RECEIVED)
begin
do_transmit<=1;
RECEIVED<=1;
data_count<=0;
```

```
//check_sum<=1;
end


else if (is_received_A & RECEIVED)
begin
      if (data_count<9)
         begin
         data_count = data_count+1;
         end

    else
        begin

        end

    end

else if(!RECEIVED)
begin
data_out <= data_out+bram_data;
data_count = data_count+1;

end

else if (is_transmitted)
begin
do_transmit<=0;
end

else
begin
end
end


// Instantiation of Transmitter module
```

```
Transmitter T1(.clk(clk) , .reset(reset) , .data_in(data_out),
.send_data(do_transmit) , .tx_out(tx_out) , .is_transmitted(is_transmitted));
endmodule
```

Constraints file:

```
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports Rx_data]
set_property IOSTANDARD LVCMOS33 [get_ports tx_out]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports RECEIVED]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[0]}]
set_property PACKAGE_PIN V11 [get_ports {data_out[7]}]
set_property PACKAGE_PIN V12 [get_ports {data_out[6]}]
set_property PACKAGE_PIN V14 [get_ports {data_out[5]}]
set_property PACKAGE_PIN V15 [get_ports {data_out[4]}]
set_property PACKAGE_PIN T16 [get_ports {data_out[3]}]
set_property PACKAGE_PIN U14 [get_ports {data_out[2]}]
set_property PACKAGE_PIN T15 [get_ports {data_out[1]}]
set_property PACKAGE_PIN V16 [get_ports {data_out[0]}]

set_property PACKAGE_PIN V17 [get_ports check_store]
set_property PACKAGE_PIN R18 [get_ports check_sum]
set_property PACKAGE_PIN N14 [get_ports do_transmit]
set_property PACKAGE_PIN E3 [get_ports clk]
set_property PACKAGE_PIN H17 [get_ports RECEIVED]
set_property PACKAGE_PIN V10 [get_ports reset]
set_property PACKAGE_PIN C4 [get_ports Rx_data]
set_property PACKAGE_PIN D4 [get_ports tx_out]
set_property IOSTANDARD LVCMOS33 [get_ports check_store]
set_property IOSTANDARD LVCMOS33 [get_ports do_transmit]
set_property IOSTANDARD LVCMOS33 [get_ports check_sum]

set_property IOSTANDARD LVCMOS33 [get_ports {data_count[1]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {data_count[0]}]
set_property PACKAGE_PIN U16 [get_ports data_count]

set_property PACKAGE_PIN V17 [get_ports {data_count[1]}]
set_property PACKAGE_PIN R18 [get_ports {data_count[0]}]

set_property PACKAGE_PIN U16 [get_ports {data_count[3]}]
set_property PACKAGE_PIN U17 [get_ports {data_count[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_count[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_count[2]}]
```

Python code:

```
import serial
import time
ser = serial.Serial('COM15', baudrate=9600)  # Replace 'COMx' with your
actual serial port
print("done")
l=[1,2,3,4,5,6,7,10,11]

for i in l:
    bin_data=i.to_bytes(1, byteorder='big')
    ser.write(bin_data)
    time.sleep(1)
# time.sleep(2)
a=0
try:
    while True:
        if (a>0):
            break
        # Read 1 byte (8 bits) of binary data from the serial port
        binary_byte = ser.read(1)
        # Convert binary data to an integer
        decimal_value = int.from_bytes(binary_byte, byteorder='big')
        print("Decimal equivalent:", decimal_value)
        a=a+1


except Exception as e:
    print("An error occurred:", e)
ser.close()  # Close the serial port in case of any exception
```

The full notebook with outputs can be found on Github.