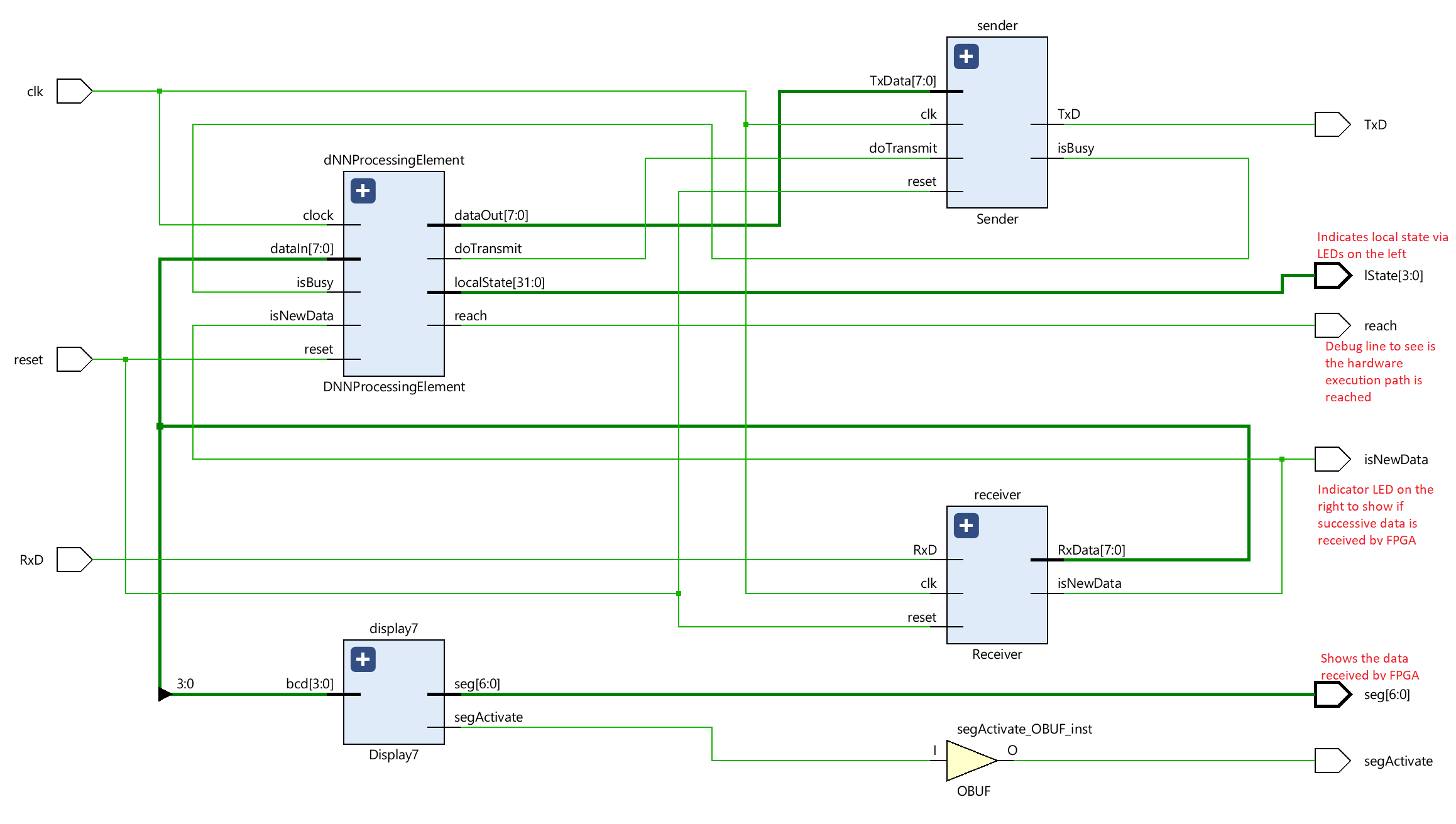
**Schematic Diagram of DNN Accelerator attached to PC**



**Signals:**

clk – as name implies

reset – as name implies

RxD – UART pin IN from computer

TxD – UART pin OUT to the computer

//The following output signals are purely for your understanding and debugging and are unnecessary in a real design

lState – connected to leds to show the internal states | shows the current local state of the state machine (first four leds from left)

reach – inside the code there is a statement ‘reach =1’, if the execution path reaches this point then reach output will be 1 (second led from right)

isNewData – toggles between 1 and 0 when getting successive data from PC.

Seg- shows the last data received from PC on all three 7 segment display – optimize if you care !!!

segActivate – disactivates one of the 7 segment.

Modules

module Receiver (

    clk,  //input clock

    reset,  //input reset

    RxD,  //input receiving data line

    RxData,  // output for 8 bits data

    isNewData  //change in value

);

module Sender (

    clk,

    reset,

    TxD, **//output to computer**

    doTransmit**, //input from DNN core telling sender to send data present in TxData**

    TxData,

    isBusy **//output from sender saying this module is busy sending data – wont show up for a few(I guess 1 or 2 cyles) cycles even after doTransmit goes up**

);

module DNNProcessingElement (

    //\*\*\*\*\*\*\*\*\*\*inputs\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    //clock signal

    reset,

    clock,

    //data from uart receiver to dnn accelerator

    dataIn,

    //to see a valid data is present at dataIn from UART receiver

    isNewData,

    isBusy,

    //\*\*\*\*\*\*\*\*\*\*\*\*\*outputs\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    // data from DNN accelerator to UART transmitter

    dataOut,

    // to see valid data is present at dataOut

    doTransmit,

    localState,  //debug tool to see the local state

    reach  //to see if a piece of code is getting reached.

);

**State Machine**

Has three state tracking variables

globalState, localState, sequenceCounter

global state set based on first input from UART – 1 is CNN – place holders made for other types of layer for you to implmenet

under CNN global state

local state 0: load the shape of the layer

local state 1: load inputs

local state 2: load weights

local state 3: set corresponding outputs to zero

local state 4: do layer operation – convolution

local state 5: do post processing – I haven’t added any operation here

local state 6: transmit the data out by checking the busy ness of UART.

Sequence counter – code is documentation.

How to run demo

1. Install python (if not installed)
2. Install pySerial (if not installed) – look up in the internet on how to install pySerial using pip
3. Generate Bitstream (on vivado) – of the project with main as top module
4. Program FPGA with bitstream (on vivado) – look at class reference on how to do this if you forgot
5. Right most switch is the reset – reset it once.
6. Run the Server.py python file in the project folder
   1. Your com port will different that the one in the code
      1. printPortInfo() – check the output of this function and set the port accordingly using the following function
      2. def openPort(port="COM6", baudRate=9600): [\\you](file:///\\you) can also call it with the right port \\
7. Observe the output on the terminal
8. Read the python code to get ideas on how you can modify the code – a lot of stuff have been demonstrated in the code
   1. Some ideas for you are – modify the code get the weights from CSV file
   2. Pass the inference input from an image
   3. Store the intermediate layer results in DS in python.

Training

Tips

\*Here is some c++ code on training modify it for 8 bit

<https://github.com/Dynmi/AlexNet>

\*I have written the UART in a simple manner so that you can extend the bitsize by replicating the states and ganging together multiples of 8 bits. – Reading and changing code is involved in this activity