Lazy Neural Processing Unit

Contact Info:
Varadraj Sinai Kakodkar
varadrajsinaikakodkar@gmail.com

Disclaimer: *The information provided here is very abstract and best way to interpret it is to send me an email or discuss it over a call. Please reach out to me for source code or git access.

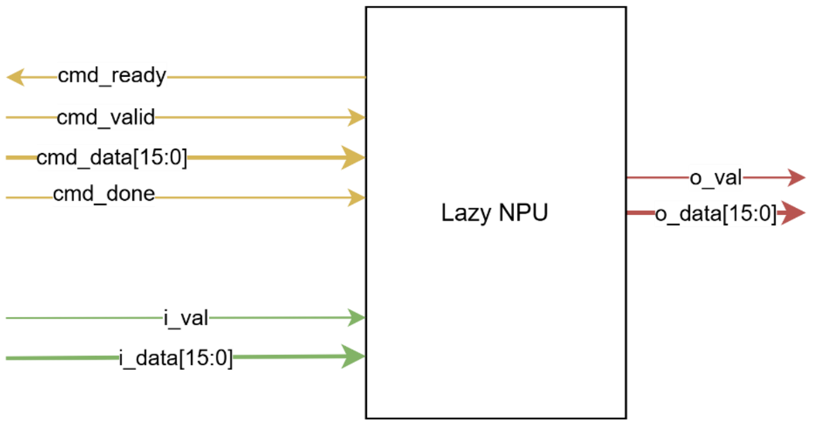
Introduction

This document describes the architecture of the hardware implementation of a AI Neural Processing Unit-D which is capable of acting as an accelerator for inferencing any generic Deep Neural Network.

The design requires that the module be configured and loaded with pre-trained weights and biases by the user via software which will be described in the following sections. All the computations are assumed to be in 16bit fixed point representation.

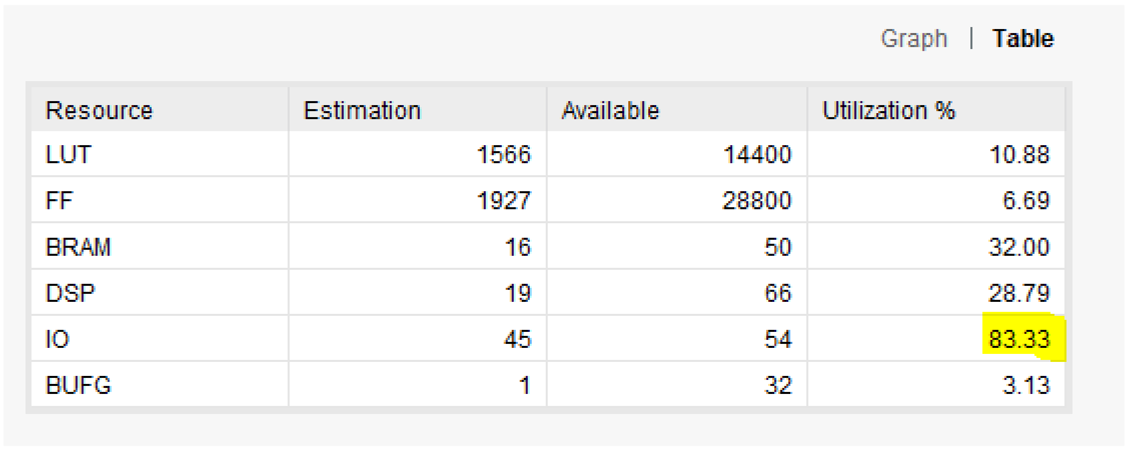
The main goal is to package this IP as an AXI slave acting as an accelerator/co-processor for any CPU.

This module when seen from over 60,000ft looks something like below.



The idea behind this module is to exploit the instruction level parallelism encountered in Deep Neural Network inference models which typically have thousands of Multiply-Accumulate(MAC) operations required for every test vector.

This design is targeted for lost cost inference edge devices. The project is targeted for the Avnet Minized (xc7z007sclg225-1) board and following is the resource utilization for the IP (without any interfacing logic with either a Zync/Microblaze system)

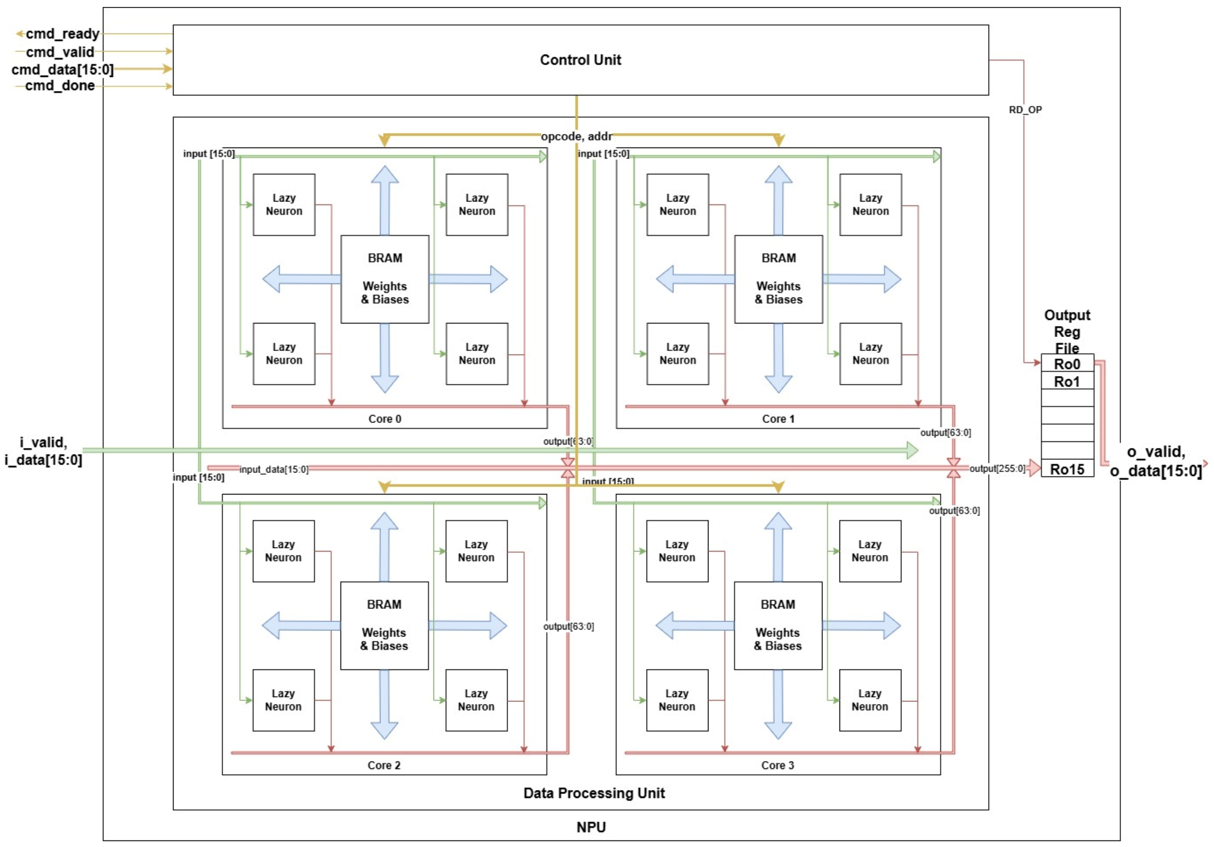


Resource	Estimation	Available	Utilization %
LUT	1566	14400	10.88
FF	1927	28800	6.69
BRAM	16	50	32.00
DSP	19	66	28.79
IO	45	54	83.33
BUFG	1	32	3.13

*Please ignore the IO utilization as this table is derived with the synthesis report of the IP top and does not mean that the IP Top with will be mapped to the FPGA pins.

Microarchitecture

This module when seen from over 20,000ft looks something like below.



The design consists 3 primary blocks which are as Control Unit, Data Processing Unit, Output Register File.

The **Data Processing Unit** consists of 4 cores. Each core consists of 4 Neurons (Perceptrons) and a dedicate weights and biases memory. Depending on the command and configuration, either 1, 2 or all 4 of the cores can be active at any time which essentially means that at most we can have 16 active neurons available to the application.

The **Control Unit** handles the command interface which supports the following set of commands.

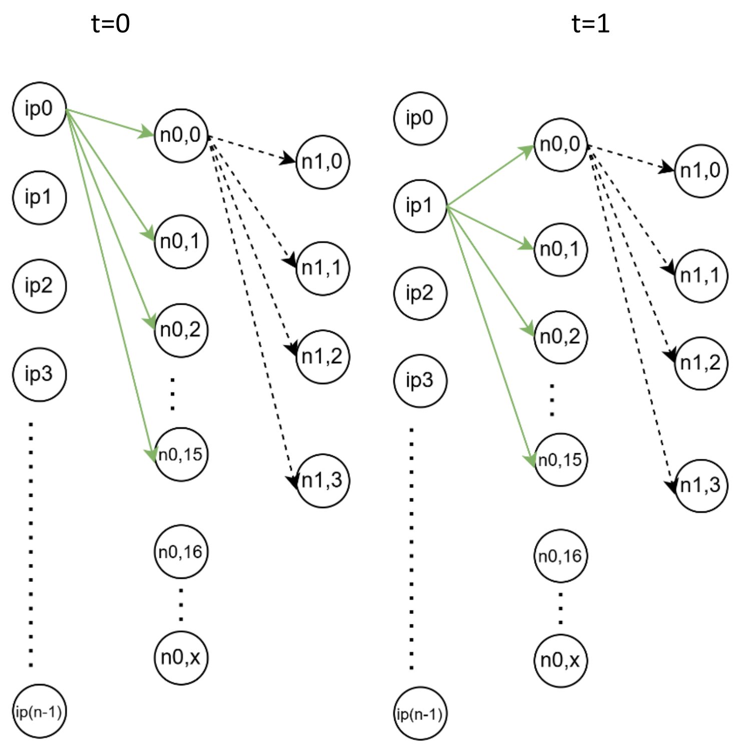
- **IDLE:** Since the name of the module goes as Lazy NPU, this command serves as the most important command and does absolutely nothing!
- **Configure:** This command is used to configure the NPU to use either 1, 2 or 4 cores.
- **Execute:** This command instructs the NPU start data processing.
- **Read_Op:** This command is use to read the values computed by the NPU.

The **Register File** contains 16 registers which hold the output of the data processing operation from 16 neurons and are read during the Read_Op stage.

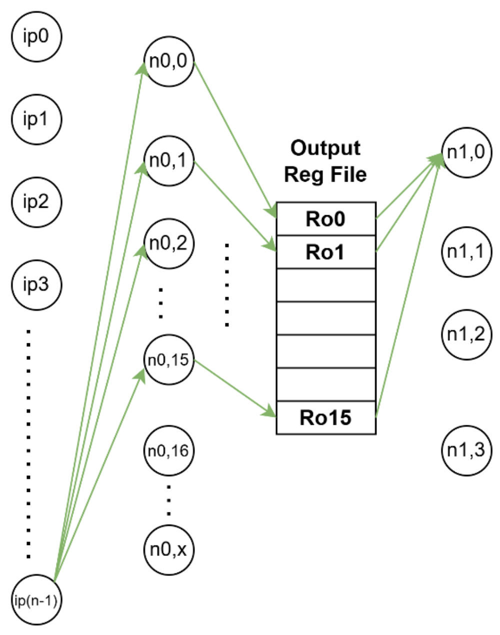
Usage

The Lazy NPU is capable of emulating any deep neural network not necessarily anything less than 16 neurons. The way we do it is as follows.

Since we have 16 neurons at most, we stream the input stream for these 16 neurons such that after 16 clock cycles we will have the output from the n0,0 to n0,15 in the reg file.



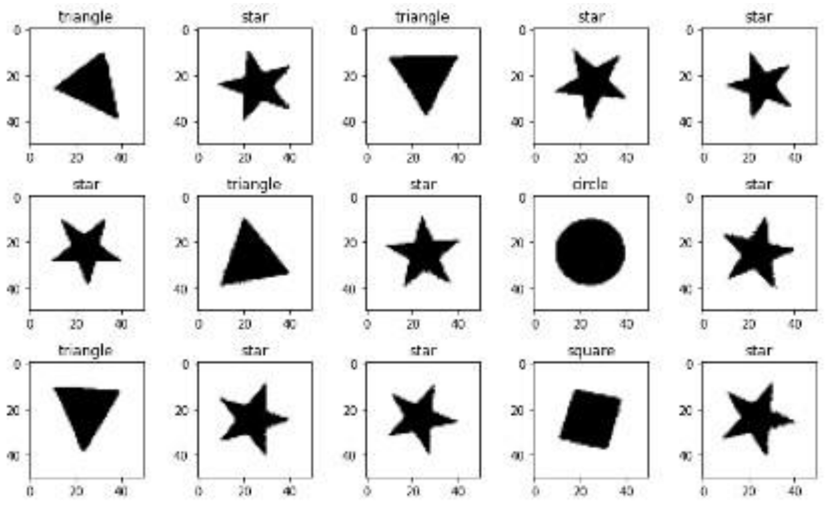
After all the inputs have been streamed for first 16 neurons, it is the responsibility of the application software to readback the register file and feed the values as input to the neural net again.



Example Application

Four Shapes Detection

This application consists of a set of 16,000 images consisting of 4 types of images {Circle, Triangle, Square, Start}



The google collab notebook explains the keras implementation of the fully connected deep neural network. The neural net consists of 2 layers with the first hidden layer of 16 neurons and output layer with 4 neurons.

The test set is divided into following categories for train, validate and testing

```
Number of training images:  10479
Number of validation images:  3992
Number of testing images:  499
```

The collab notebook contains the explanation for each of the sections and is used to derive the weights and biases and also process the test images for generating the test inputs.

The keras model has a prediction accuracy of over 100% for the test dataset of 499 images! This serves as a baseline for the RTL model accuracy.

Testing

Since this neural net has a shape of (256,16,4), the testing needs the Lazy NPU resources to be time shared and the following sequence to be implemented for each test input vector.

First iteration of the calculation will need 16 neurons to be active i.e all 4 cores to be active. We then read back the register file and configure the Lazy NPU to have only 1 active core. We stream the data back into the Lazy NPU to check the output of the first core to find the neuron with the maximum output.

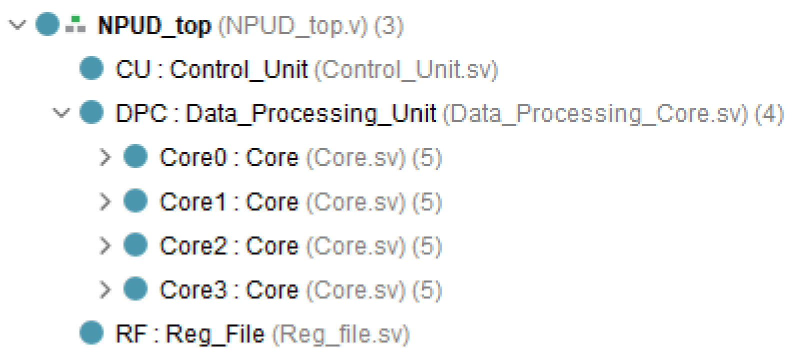
The following are systemverilog tasks provided along with the RTL which do the low-level transaction modelling for testing the input.

1. Init()
2. ConfigCores(4)
3. Start_Img()

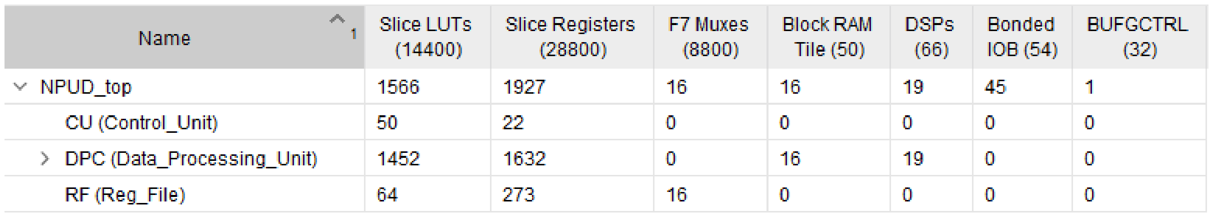
4. Loop: Send_Pixel()
5. Finish_Img()
6. ConfigCore(1)
7. Load_Back()
8. Check_Output

Results

Lazy NPU Structure & Resource Utilization



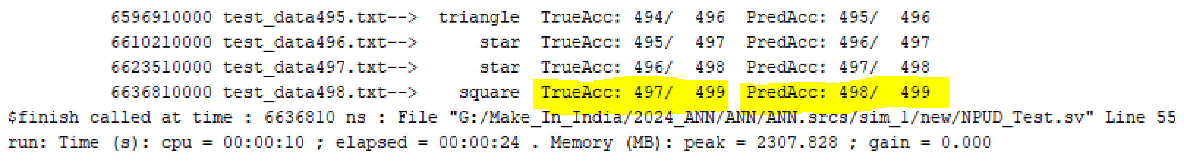
- **NPUD_top** (NPUD_top.v) (3)
  - CU : Control_Unit (Control_Unit.sv)
  - DPC : Data_Processing_Unit (Data_Processing_Core.sv) (4)
    - Core0 : Core (Core.sv) (5)
    - Core1 : Core (Core.sv) (5)
    - Core2 : Core (Core.sv) (5)
    - Core3 : Core (Core.sv) (5)
  - RF : Reg_File (Reg_file.sv)



Name	Slice LUTs (14400)	Slice Registers (28800)	F7 Muxes (8800)	Block RAM Tile (50)	DSPs (66)	Bonded IOB (54)	BUFGCTRL (32)
NPUD_top	1566	1927	16	16	19	45	1
CU (Control_Unit)	50	22	0	0	0	0	0
DPC (Data_Processing_Unit)	1452	1632	0	16	19	0	0
RF (Reg_File)	64	273	16	0	0	0	0

The Lazy NPU RTL model was fed the same dataset of 499 test images. The Lazy NPU predicted 497/499 images correctly. The prediction accuracy for the test dataset was over 99%.

There was 1 mis prediction between the Lazy NPU and the keras model which is attributed to the quantization of weights and biases into 16 bit floating point arithmetic.



```
          6596910000 test_data495.txt-->  triangle  TrueAcc: 494/   496  PredAcc: 495/   496
          6610210000 test_data496.txt-->      star  TrueAcc: 495/   497  PredAcc: 496/   497
          6623510000 test_data497.txt-->      star  TrueAcc: 496/   498  PredAcc: 497/   498
          6636810000 test_data498.txt-->    square  TrueAcc: 497/   499  PredAcc: 498/   499
$finish called at time : 6636810 ns : File "G:/Make_In_India/2024_ANN/ANN/ANN.srcs/sim_1/new/NPUD_Test.sv" Line 55
run: Time (s): cpu = 00:00:10 ; elapsed = 00:00:24 . Memory (MB): peak = 2307.828 ; gain = 0.000
```