**ECE 298A Deliverable**

**Vivi Huang, Evan Zhang**

**Design Specifications**

Our design specifications are shown below.

Design Objective:

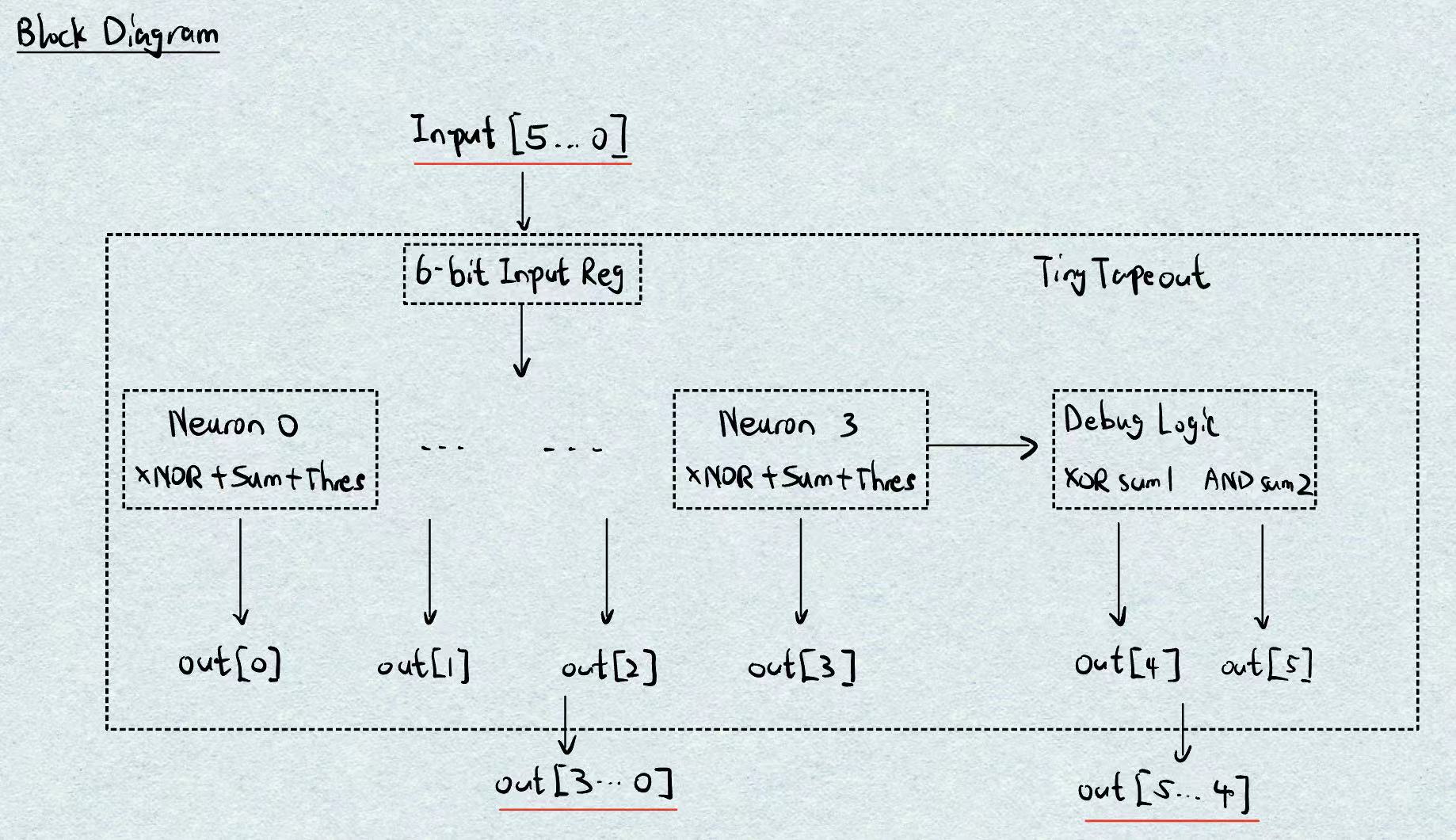
Implement a 4-neuron Binary Neural Network (BNN), on a Tiny TapeOut ASIC to classify a 6-bit binary input (line inputs) to perform classification into 4 different classes: turn left, turn right, stop, and U-turn.

Input/Output:

Use 6-bit parallel input and 4 binary outputs + 2 debug outputs, out of the 8 total I/Os. 6-bit parallel inputs are assumed to be “sensor-gathered data”. 4-bit output provides 4 different classifications for the input. 2-bit debug bits can be used to observe intermediate arithmetic processes for neuron 0, both for our building purposes as well as learning purposes.

4 of the 8 bidirectional I/O pins will be used to implement dynamic weight loading. This is loaded during runtime, and weights are shifted serially based on the cycle number. In two clock cycles, the 4 bidirectional I/O pins will finish writing into one neuron, and the complete process can be completed within 8 cycles. We use an additional I/O pin to use as the *load\_enable* signal to enable loading.

A block diagram is shown below:



Hardware Specs:

Each neuron will contain 6 weights and registers accordingly to perform **XOR + popcount + threshold** operations to produce a single output bit. By estimation and per neuron, we will need 11 flip-flops per neuron as well as 10 wires per neuron. FFs are used for weight storage, threshold register plus sum register. Wires are used in these 3 processes as well. In the total design, this should accumulate to **53 FFs and 46 wires**. This means approximately 800 gates total, which is well within the limit of Tiny Tapeout Specs.

Testing:

Testing will be done just using CocoTB. We’ll be able to use the 8-bit I/O, clock, reset, power and ground like written above. The weight-loading can also be tested using 8 bi-directional I/Os.

Note that weight-training will be done outside of the Tiny-Tapeout chip, and will be performed separately using the Larq framework in Python to train the weights for BNN. A library of training data will be given to the BNN; this is what underlies the fundamental weights for our neural network.