categories that the perceptron has separated into. In essence, a perceptron is a binary classifier that classifies on input data.

8.2 NEURAL NETWORK REPRESENTATION

The neural network representation of a perceptron is show in figure 8.2. A perceptron can be

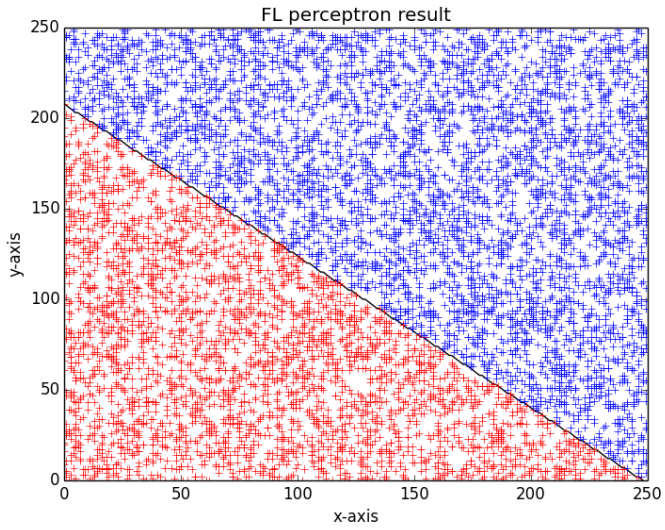


Figure 8.1 Graphical representation of perceptron.

Equation: $31x_0 + 37x_1 - 7700 = 0$

represented by one neuron. And figure 8.3 is the RTL CGRA representation of the perceptron neural network. In the RTL implementation, I implemented a two dimensional space perceptron for the ease of representing the result graphically. A two dimensional space perceptron takes in a

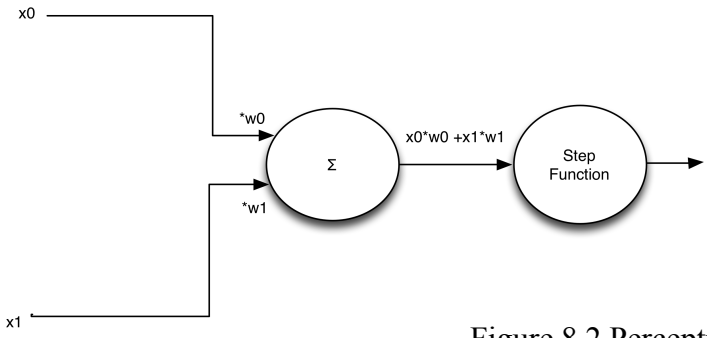


Figure 8.2 Perceptron neural network

two point data set x0, x1, and the perceptron contains two weights w0 and w1. The graphical representation of the data set is a point on a plane, and the perceptron classifier becomes a line on the plane.

For a three dimensional perceptron, an input data set would contain 3 points and appears as a point in three-dimensional space, and the classifier would be a plane in the space. Of course this

perceptron can be easily expanded to support many dimensional space. The RTL representation involves two PEs. PE0 acts as the neuron and has weights w0 and w1 stored in its internal registers. For perceptrons that implements learning algorithms, the value of w0 and w1 would be updated based on a set of data used to train the perceptron. The RTL implementation does not implement this learning algorithm, and its weights are manually set. As in figure 8.1, w0 is 31 and w1 is 37. PE0 performs multiply add operations on the data set with weights. The end result x0*w0+x1*w1 is then passed to PE1 which represent a step function that classifies the data set. The output is either 1 or 0, representing two different categories. PE1 contains an offset (offset = 7700 in figure 8.1) that is used to compare to the processed result received from PE0. The PE RTL implements the approximate multiplier, and in the next section, we will explore the

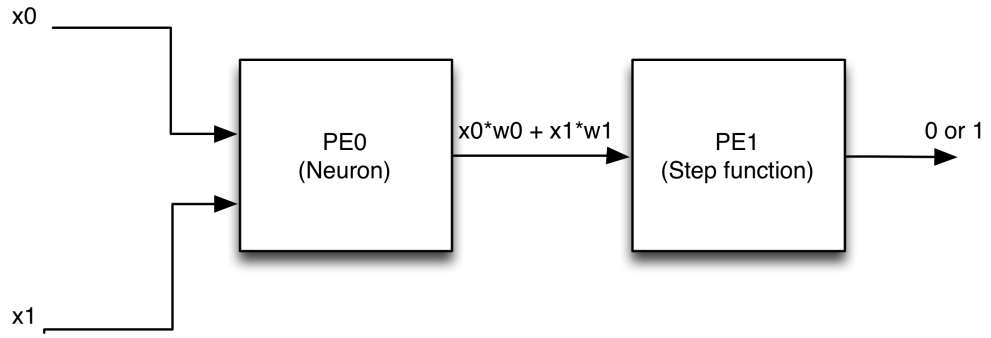


Figure 8.3 Perceptron RTL CGRA

approximate multiplier's impact on the perceptron calculation.

8.3 RESULT

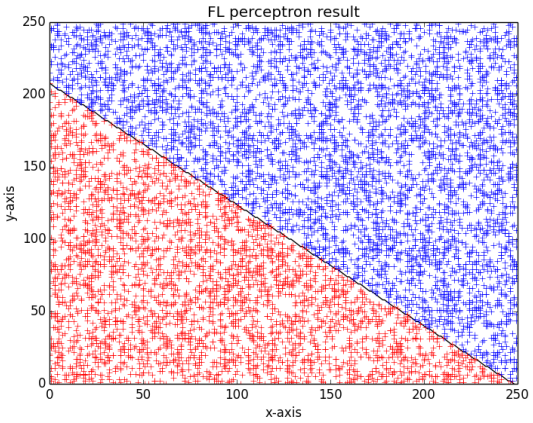
The experiments below are done on an 8 bit input 16 bit output multiplier output with various k values on the same 8000 random generated integer data points whose values range from 0 to 250. Table 8.1 represents the summarized result of perceptron with various parameters. According to this table, the nature of the number that the multiplier is multiplying determines its accuracy. For the left most column , the multiplier multiply the inputs with weights w0=31(0b11111) and w1=37 (0b100101). Based on the discussion approximate multiplier in section 6, the more 1 bits there are in the number, the more accurate the final result will be because the multiplier always sets the last bit before truncation to 1 to approximate the truncated values. So the more 1's an operand has the higher the chance that the truncation's approximation will be correct. Operand w0=31(0b11111) and w1=37 (0b100101) have a reasonable number of 1's, so at k=2 it yields 88.21% accuracy compared to the functional level and at k=4 it is 96.76% accuracy. However, for the right most column w0=16(0b10000), w1=33(0b100001), the operands contain mostly 0's. As a result, at k=2, it is only 69.03% accurate and at k=4 it is 90.58% accurate.

The result of the left most column and right most column are presented graphically in table 8.2 and 8.3 respectively. The approximate multiplier results in a zig-zag shape. The number of zig-zags increase as the number of k increases, and the approximation results becomes closer to the functional model. For w0=31 and w1=37, the zig-zag mostly is centered around the line that

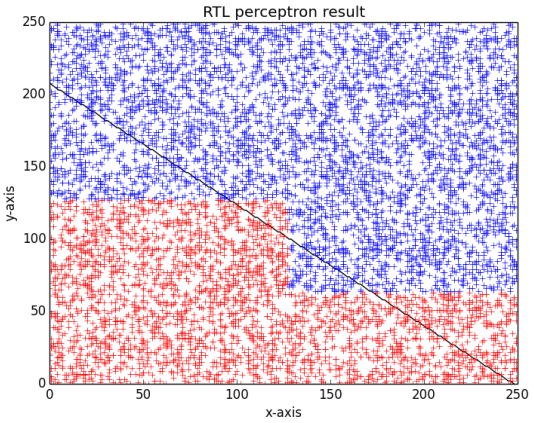
classifies the two sets of data points, giving more accurate result for most k values. However, for w0=16 and w1=33, the zig-zag is mostly below the dividing line, yielding less accurate result. However, as the value k increases, there tends to be little difference in accuracy across the various weight values. These results prove that for the application of neural network, the approximate multiplier would be a viable choice. It reduces the area, energy and latency, while compromising little on the results.

	w0 = 31, w1 = 37, offset = 7700	**w0 = 24, w1 = 37, offset = 7700**	**w0 = 16, w1 = 33, offset = 7700**
k=2	88.2125%	87.175%	69.025%
k=3	93.6375%	92.025%	85.1875%
k=4	96.7625%	95.9625%	90.5875%
k=5	98.1875%	97.025%	96.1375%
k=6	99.3125%	98.05%	98.425%

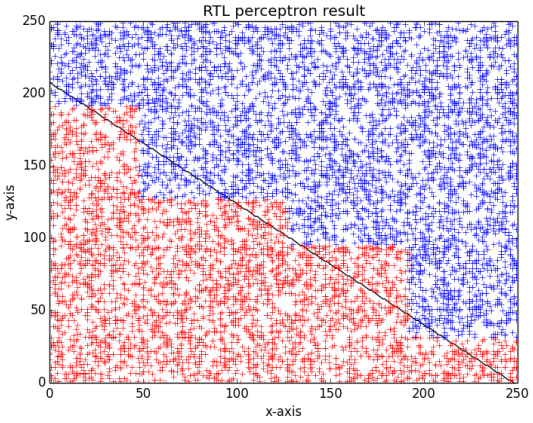
Table 8.1 Perceptron Accuracy of RTL Results Compared to FL Simulation



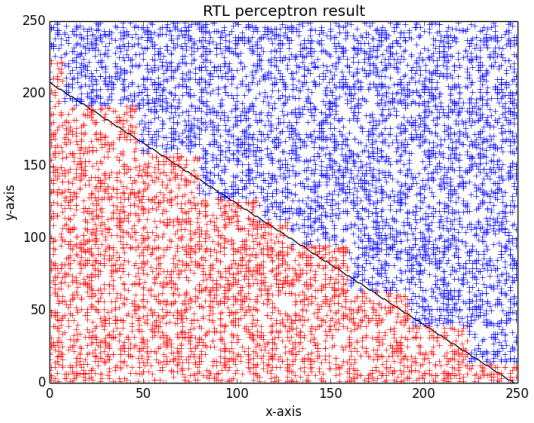
Functional level perceptron
w0 = 31, w1 = 37, offset = 7700, 8000 data points



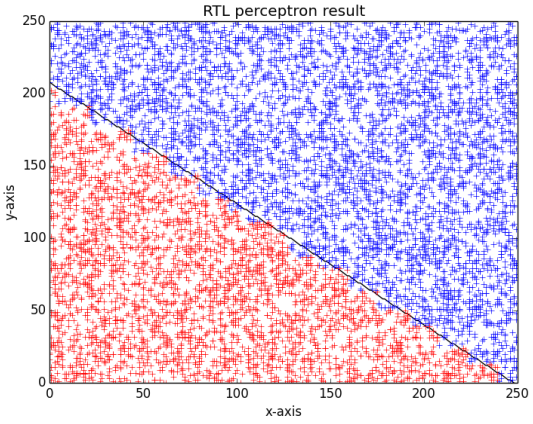
RTL perceptron
w0 = 31, w1 = 37, offset = 7700, 8000 data points k = 2
88.2125% accurate



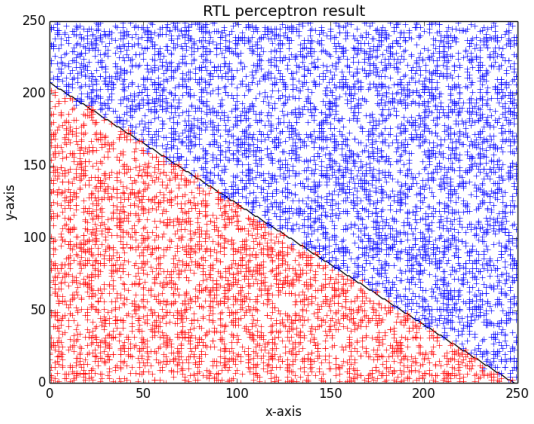
RTL perceptron
w0 = 31, w1 = 37, offset = 7700, 8000 data points k = 3
93.6375% accurate



RTL perceptron
w0 = 31, w1 = 37, offset = 7700, 8000 data points k = 4
96.7625% accurate

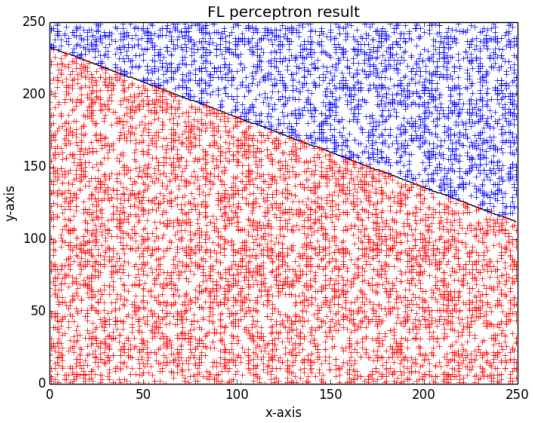


RTL perceptron
w0 = 31, w1 = 37, offset = 7700, 8000 data points k = 5
98.1875% accurate

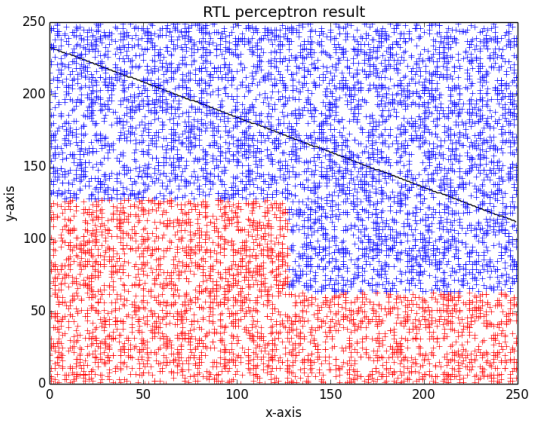


RTL perceptron
w0 = 31, w1 = 37, offset = 7700, 8000 data points k = 6
99.3125% accurate

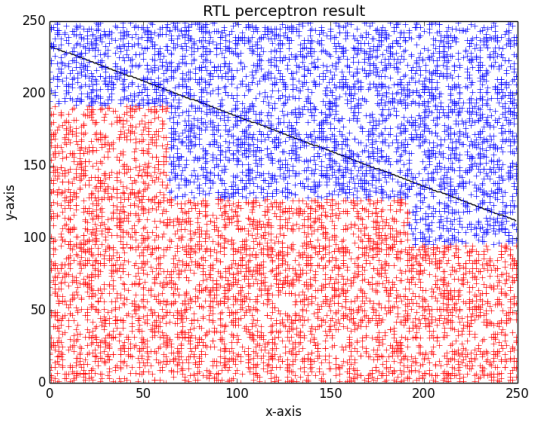
Table 8.1 Perceptron RTL w0 = 31, w1 = 37, offset = 7700



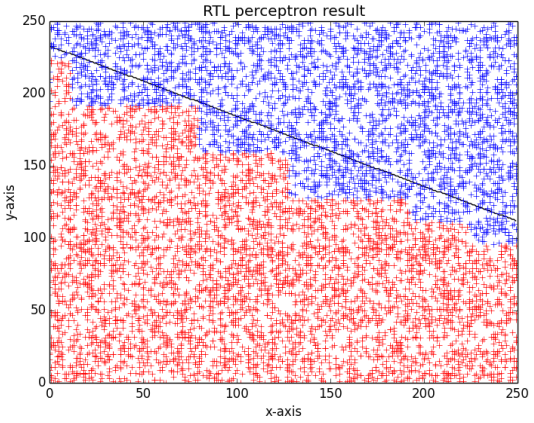
Functional level perceptron
w0 = 16, w1 = 33, offset = 7700, 8000 data points



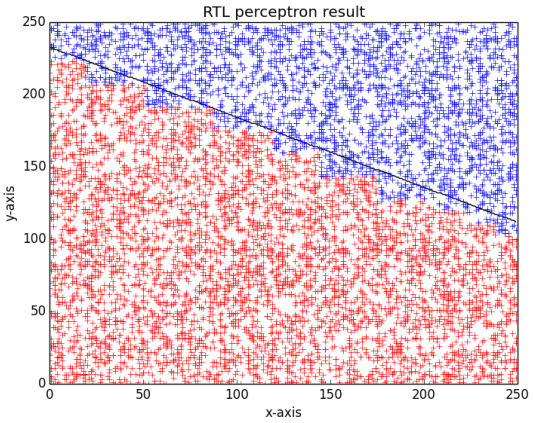
RTL perceptron
w0 = 16, w1 = 33, offset = 7700, 8000 data points k = 2
69.025% accurate



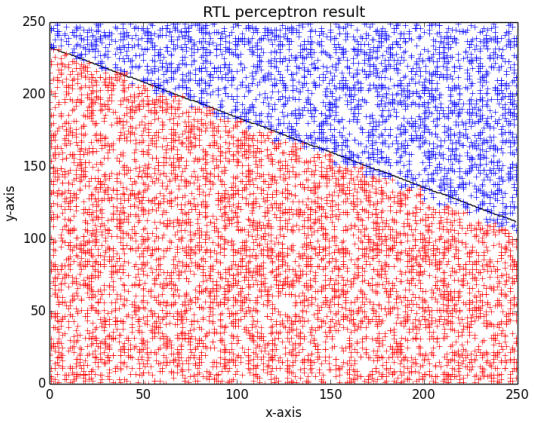
RTL perceptron
w0 = 16, w1 = 33, offset = 7700, 8000 data points k = 3
85.1875% accurate



RTL perceptron
w0 = 16, w1 = 33, offset = 7700, 8000 data points k = 4
90.5875% accurate



RTL perceptron
w0 = 16, w1 = 33, offset = 7700, 8000 data points k = 5
96.1375% accurate



RTL perceptron
w0 = 16, w1 = 33, offset = 7700, 8000 data points k = 6
98.425% accurate

Table 8.2 Perceptron RTL w0 = 16, w1 = 33, offset = 7700