DESIGN OF A TWO-BIT POSITIVE CLOCK-EDGE TRIGGERED COUNTER UTILIZING THRESHOLD LOGIC UNIT BASED ON PERCEPTRON LEARNING ALGORITHM

Ratnadip Dey[1], Debdut Biswas[2], Pijush Dutta[3]

[1,2]Department of Electronics and Communication Engineering
Institute of Engineering & Management, Kolkata
{ratnadip87, debdut123}@gmail.com
[3]Department of Mechatronics Engineering
National Institute of Technical Teachers' Training & Research, Kolkata
[3]pijushdutta009@gmail.com

ABSTRACT

A Threshold Logic Unit (TLU) is a mathematical function conceived as a crude model, or abstraction of biological neurons. Threshold logic units are the constitutive units in an artificial neural network. In this paper a positive clock-edge triggered T flip-flop is designed using Perceptron Learning Algorithm, which is a basic design algorithm of threshold logic units. Then this T flip-flop is used to design a two-bit up-counter that goes through the states 0, 1, 2, 3, 0, 1… Ultimately, the goal is to show how to design simple logic units based on threshold logic based perceptron concepts.

KEYWORDS

Threshold Logic, Perceptron, VHDL, Flip-flop, Counter

1. INTRODUCTION

In 1943, Warren McCulloch and Walter Pitts introduced one of the first artificial neurons or so called threshold logic units. The main feature of their neuron model is that a weighted sum of input signals is compared to a threshold to determine the neuron output. When the sum is greater than the threshold, the output is 1. When the sum is less than or equal to the threshold, the output is 0. They went on to show that networks of these neurons could, in principle, compute any arithmetic or logic function. [1] In the late 1950s, Frank Rosenblatt and several other researchers developed a class of neural networks called perceptrons. The neurons in these networks were similar to those of McCulloch and Pitts. Rosenblatt's key contribution was the introduction of a learning rule for training perceptron networks to solve pattern recognition problems. [2] He proved that his learning rule will always converge to the correct network weights, if weights exist that solve the problem. Learning was simple and automatic.

Ultimately, *learning rule* means a procedure for modifying the weights and biases of a network. (This procedure may also be referred to as a training algorithm.) The purpose of the learning rule is to train the network to perform some task. There are many types of neural network learning rules. In this paper we take the advantage of the *Perceptron Learning Algorithm* as was developed by Frank Rosenblatt.

Rupak Bhattacharyya et al. (Eds) : ACER 2013,
pp. 357–367, 2013. © CS & IT-CSCP 2013 DOI : 10.5121/csit.2013.3233

There is a reason for choosing to implement a flip-flop using the threshold logic units or perceptrons. Let us envision a flip flop from the biological point of view. The brain is a large, web-like structure in which neurons gather information from other neurons, make a decision to discharge or not, and then pass this information onto other cells. Neuronal activity can rapidly flip-flop between stable states. The role of neuronal flip-flops has already been an area of interest for neuroscientists for some recent years now. [3] These neuronal flip-flops have been theorized in the same way as flip-flops in computers. In computers, these devices are commonly used for storage of a bit of information. The nervous system can also perform a similar role. A particular neuron cell is considered to exist in several states called *multistable states*. These states have the capability of bringing cells or entire networks 'on-line' in a behaviorally appropriate manner. The combination of synaptic, dendritic, neuronal and network flip-flops could provide a powerful range of states to guide and influence neuronal processing. Complicated calculations such as integration, gain modulation can be easily performed by the multiple stable states in the neurons and networks in a robust and stable manner. Rapid modifications in excitability and activity can be useful for keeping track of information such as position of eyes or head, making rapidly changing sensory world to the appropriate motor responses. However, this proposition does not hold well when a simple piece of information is to be stored in an entire cell or local network of cells.

Although the perceptron initially seemed promising, it was eventually proved that perceptrons could not be trained to recognize many classes of patterns. This led to the field of neural network research stagnating for many years, before it was recognized that a feedforward neural network with two or more layers (also called a multilayer perceptron) had far greater processing power than perceptrons with one layer (also called a single layer perceptron). [4] More recently, interest in the perceptron learning algorithm increased again after Freund and Schapire (1998) [5] presented a voted formulation of the original algorithm (attaining a large margin) to which the *kernel trick* can be applied.

2. PROPOSED MODEL

A linear threshold function f is a multi-input function in which each input, $x_i \in \{0,1\}$, $i \in \{0,1,\dots,l\}$, is assigned a weight $w_i$such that f assumes the value 1 when the weighted sum of its inputs exceeds the value of the function's threshold, T. That is, [6]

$$f(x_1, x_2, \dots, x_n) = \begin{cases} 1 & if \sum_{i=1}^{l} w_i x_i > T + \delta_{on} \\ 0 & if \sum_{i=1}^{l} w_i x_i \leq T - \delta_{off} \end{cases}$$

Parameters δ_{on} and δ_{off}represent defect tolerances that must be considered since variations (due to manufacturing defects, temperature changes, etc.) in the weights can lead to network malfunction.

In the context of artificial neural networks, a perceptron is similar to a linear neuron. However, where a perceptron tries to learn weights that are always getting closer to a better set of weights, a linear neuron learns a set of weights where the outputs are always getting closer to the target outputs. Put another way, a perceptron is more interested in learning the hyperplane that correctly separates two classes of training input, whereas a linear neuron is more interested in learning a set of weights which reduce a real-valued prediction error. The perceptron algorithm is also termed the single-layer perceptron, to distinguish it from the case of a multilayer perceptron, which is a misnomer for a more complicated neural network. As a linear classifier, the (single-layer)

perceptron is the simplest kind of feedforward neural network. [7]

We first define some variables:

- $y = f(\bar{z})$, where y denotes the output from the perceptron for an input vector $\bar{z}$.
- b is the bias term which is considered to be 0 in this paper.
- $D = \{(\overline{x_1}, d_1), \dots, (\overline{x_s}, d_s)\}$ is the training set of s examples. Here $\overline{x_j}$ is the n-dimensional input vector and d_j is the desired output value of the perceptron for that input.
- $x_{j,i}$ is the value of the ith node of the jth training input vector.
- $x_{j,0} = 1$
- w_i is the ith value in the weight vector, to be multiplied by the value of the ith input node.
- $w_i(t)$, is the weight i at time t.
- r is the learning rate, where $0 < r < 1$. If the learning rate is too high then the perceptron periodically oscillates around the solution unless *additional steps* are taken.

Then the steps of the learning algorithm as proposed by Rosenblatt,

1. Initialize weights and threshold. In this paper $w_i(0) = 0$ and threshold $\gamma = 0.5$ in most cases.
2. Calculate the actual output-
   $$y_j(t) = f\left[\overline{w}(t) \cdot \overline{x_j}\right] = f\left[w_o(t) + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \cdots + w_n(t)x_{j,n}\right]$$
   and $\quad w_i(t+1) = w_i(t) + r\left(d_j - y_j(t)\right)x_{j,i}$
3. Step 2 is repeated until iteration error e
   $$e = \frac{1}{s}\sum_{j=1}^{s}\left(d_j - y_j(t)\right)$$
   is less than the threshold γ.

Let us design a two-input AND gate using the perceptron algorithm. The following table illustrates the Perceptron Algorithm and shows its implementation on AND logic. [8]

Table 1. Design of AND gate using the *Perceptron Learning Algorithm*

Input			**Initial Weights**		**Output**				**Error**	**Correction**	**Final Weights**	
Sensor Values		**Desired Output**			**Per Sensor**		**Sum**	**Network**				
a	b	y	w_0	w_1	c_0	c_1	s	n	e	d	w_0	w_1
					$a * w_0$	$b * w_1$	$c_0 + c_1$	if $s > t$ then 1, else 0	$y - n$	$r * e$	$\Delta(a * d)$	$\Delta(b * d)$
1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	1	0.1	0.1	0.1
1	0	0	0.1	0.1	0.1	0	0.1	0	0	0	0.1	0.1
1	1	1	0.1	0.1	0.1	0.1	0.2	0	1	0.1	0.2	0.2
1	0	0	0.2	0.2	0.2	0	0.2	0	0	0	0.2	0.2
1	1	1	0.2	0.2	0.2	0.2	0.4	0	1	0.1	0.3	0.3
1	0	0	0.3	0.3	0.3	0	0.3	0	0	0	0.3	0.3
1	1	1	0.3	0.3	0.3	0.3	0.6	1	0	0	0.3	0.3

Inputs: a, b with input a held constant at 1.
Threshold (γ): 0.5
Learning Rate (r): 0.1

In the following, the final weights of one iteration become the initial weights of the next. Each cycle over all the samples in the training set is demarcated with heavy lines. Thus in doing so, the AND gate can be represented with weights 0.3 and 0.3 on its input lines with a threshold value of 0.5

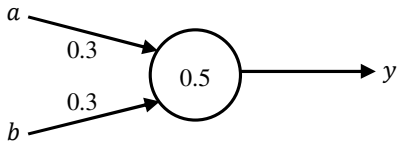

Figure 1. An AND gate in threshold logic

In order to design a counter, a JK flip-flop with both inputs connected to logic high level, or a T flip-flop with its input connected to logic high level is needed. We prefer T flip-flop to JK flip-flop, because the perceptron iteration will shorter. But a T flip-flop can be easily designed with the help of a D flip-flop. We only require an additional XOR gate to create the T flip-flop from its D counterpart. The following table (Table 2) shows the use of the perceptron algorithm to develop the D flip-flop. Here, $t = 0.5$ and $r = 0.1$ and one of the inputs D held constant at 1.

Table 2. Design of D flip-flop using Perceptron Algorithm with $D = 1$

Input				**Initial Weights**			**Output**					**Error**	**Final Weights**		
Sensor Values			**Desired Output**				**Per Sensor**			**Sum**	**Network**				
D	L	Q_n	Q_{n+1}	w_0	w_1	w_2	c_0	c_1	c_2	s	k	e	w_0	w_1	w_2
							$D * w_0$	$L * w_1$	$Q_n * w_2$	$c_0 + c_1 + c_2$	if $s > t$ then 1, else 0	$Q_{n+1} - k$	$\Delta(D * d)$	$\Delta(L * d)$	$\Delta(Q_n * d)$
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	1	0.1	0	0.1
1	1	0	1	0.1	0	0.1	0.1	0	0	0.1	0	1	0.2	0.1	0.1
1	1	1	1	0.2	0.1	0.1	0.2	0.1	0.1	0.4	0	1	0.3	0.2	0.2
1	0	0	0	0.3	0.2	0.2	0.3	0	0	0.3	0	0	0.3	0.2	0.2
1	0	1	1	0.3	0.2	0.2	0.3	0	0.2	0.5	0	1	0.4	0.2	0.3
1	1	0	1	0.4	0.2	0.3	0.4	0.2	0	0.6	1	0	0.4	0.2	0.3
1	1	1	1	0.4	0.2	0.3	0.4	0.2	0.3	0.9	1	0	0.4	0.2	0.3
1	0	0	0	0.4	0.2	0.3	0.4	0	0	0.4	0	0	0.4	0.2	0.3
1	0	1	1	0.4	0.2	0.3	0.4	0	0.3	0.7	1	0	0.4	0.2	0.3
1	1	0	1	0.4	0.2	0.3	0.4	0.2	0	0.6	1	0	0.4	0.2	0.3
1	1	1	1	0.4	0.2	0.3	0.4	0.2	0.3	0.9	1	0	0.4	0.2	0.3

Thus the D flip-flop with $D = 1$ can be visualized as neural node having three input lines having weights 0.4, 0.2 and 0.3, and with the threshold $\gamma = 0.5$. Here the input L represents the clock signal.

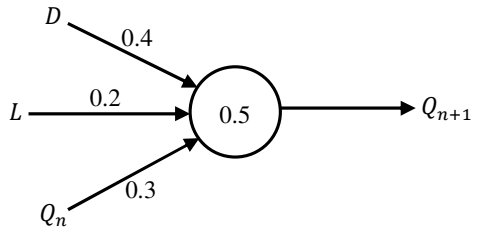

Figure 2. A D flip-flop in threshold logic with $D = 1$

The VHDL code for the above threshold logic neural node can be written as follows.

VHDL program used	Xilinx – ISE
Program version	$Xilinx\ 9.2i$
Targeted Family	Spartan3
Target Device	XC3S200
Package	PQ208
Speed	-4

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity d_ff_d_1 is
        Port ( D : in STD_LOGIC;
                L :in STD_LOGIC;
                Qn :in STD_LOGIC;
                Q :out STD_LOGIC);
end d_ff_d_1;
architecture Behavioral of d_ff_d_1 is
begin
        process(L)
                variable y, z : real;
        begin
                if rising_edge(L) or falling_edge(L) then
                        if rising_edge(L) then
                                y := 1.0;
                        else
                                y := 0.0;
                        end if;
                        z := real(conv_integer(D))*0.4 + y*0.2 +  real(conv_integer(Qn))*0.3;
                        if(z > 0.5) then
                                Q <= '1';
                        else
                                Q <= '0';
                        end if;
                end if;
        end process;
end Behavioral;
```

Let us build the remaining part of the flip-flop with one input D held constantly at $D = 0$.

Table 3. Design of D flip-flop using Perceptron Algorithm with $D = 0$

Input				**Initial Weights**			**Output**					**Error**	**Final Weights**		
Sensor Values			**Desired Output**				**Per Sensor**			**Sum**	**Network**				
D	L	Q_n	Q_{n+1}	w_0	w_1	w_2	c_0	c_1	c_2	s	k	e	w_0	w_1	w_2
							$D * w_0$	$L * w_1$	$Q_n * w_2$	$c_0 + c_1 + c_2$	if $s > t$ then 1, else 0	$Q_{n+1} - k$	$\Delta(D * d)$	$\Delta(L * d)$	$\Delta(Q_n * d)$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0.1
0	1	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0.1
0	1	1	0	0	0	0.1	0	0	0.1	0.1	0	0	0	0	0.1
0	0	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0.1
0	0	1	1	0	0	0.1	0	0	0.1	0.1	0	1	0	0	0.2
0	1	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0.2
0	1	1	0	0	0	0.2	0	0	0.2	0.2	0	0	0	0	0.2
0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0.2
0	0	1	1	0	0	0.2	0	0	0.2	0.2	0	1	0	0	0.3
0	1	0	0	0	0	0.3	0	0	0	0	0	0	0	0	0.3
0	1	1	0	0	0	0.3	0	0	0.3	0.3	0	0	0	0	0.3
0	0	0	0	0	0	0.3	0	0	0	0	0	0	0	0	0.3

0	0	1	1	0	0	0.3	0	0	0.3	0.3	0	1	0	0	0.4
0	1	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0.4
0	1	1	0	0	0	0.4	0	0	0.4	0.4	0	0	0	0	0.4
0	0	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0.4
0	0	1	1	0	0	0.4	0	0	0.4	0.4	0	1	0	0	0.5
0	1	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0.5
0	1	1	0	0	0	0.5	0	0	0.5	0.5	0	0	0	0	0.5
0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0.5
0	0	1	1	0	0	0.5	0	0	0.5	0.5	0	1	0	0	0.6
0	1	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0.6
0	1	1	0	0	0	0.6	0	0	0.6	0.6	1	-1	0	-0.1	0.5
0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0.5
0	0	1	1	0	0	0.5	0	0	0.5	0.5	0	1	0	0	0.6
0	1	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0.6
0	1	1	0	0	0	0.6	0	0	0.6	0.6	1	-1	0	-0.1	0.5
0	0	0	0	0	-0.1	0.5	0	0	0	0	0	0	0	-0.1	0.5
0	0	1	1	0	-0.1	0.5	0	0	0.5	0.5	0	1	0	-0.1	0.6
0	1	0	0	0	-0.1	0.6	0	-0.1	0	-0.1	0	0	0	-0.1	0.6
0	1	1	0	0	-0.1	0.6	0	-0.1	0.6	0.5	0	0	0	-0.1	0.6
0	0	0	0	0	-0.1	0.6	0	0	0	0	0	0	0	-0.1	0.6
0	0	1	1	0	-0.1	0.6	0	0	0.6	0.6	1	0	0	-0.1	0.6
0	1	0	0	0	-0.1	0.6	0	-0.1	0	-0.1	0	0	0	-0.1	0.6
0	1	1	0	0	-0.1	0.6	0	-0.1	0.6	0.5	0	0	0	-0.1	0.6

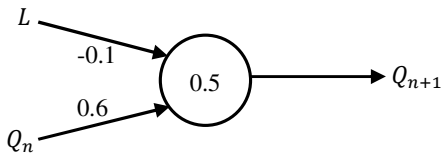

Figure 3. A D flip-flop in threshold logic with $D = 0$

The overall D flip-flop can be represented as a neural network as follows-

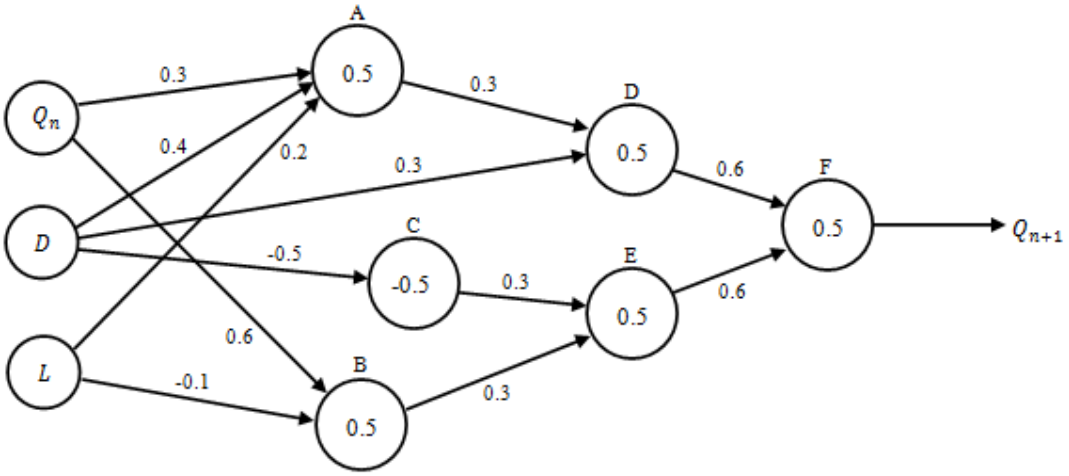

Figure 4. The D flip-flop neural network with inputs D, L and Q_n and output Q_{n+1}

Figure 4 represents the D flip-flop with two individual components - one with $D = 1$ that is shown as the node A and the other with input $D = 0$ that is shown as node B. The nodes D and E represent AND gates and the node F, an OR gate. Lastly, the node C is a NOT gate with threshold -0.5 and weight on the input line also -0.5. Thus the D flip-flop with inputs D_{in}, CLK and output Q_{out} can be described as follows –

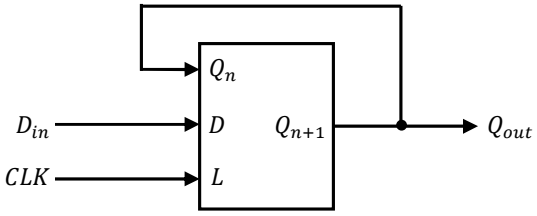

Figure 5. The complete real D flip-flop neural network

Now, in order to convert it to a T flip-flop, we just need to add an XOR gate to its input D_{in}.

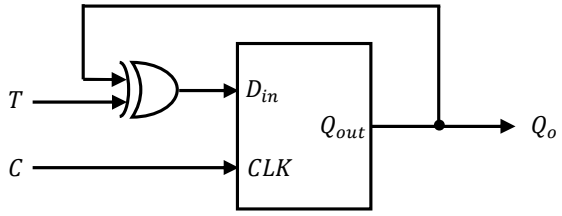

Figure 6. The T flip-flop using D flip-flop

The most classic example of linearly inseparable pattern is a logical XOR function. Our initial approach to solving linearly inseparable patterns of XOR function is to have multiple stages of perceptron networks. Each stage would set up one decision surface or a line that separate patterns. Based on the classification determined by the previous stage, the current stage can form sub-classifications. Thus, the XOR gate can be represented in threshold logic [9] by-

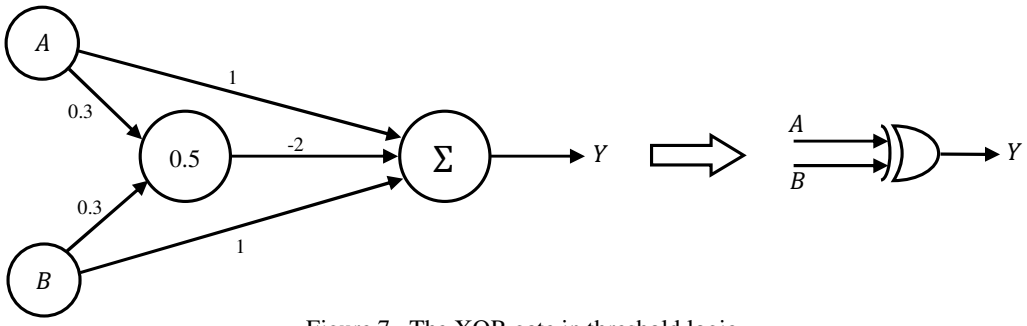

Figure 7. The XOR gate in threshold logic

Finally, we come to the two-bit up-counter. Figure 8 shows its implementation using two T flip-flops.

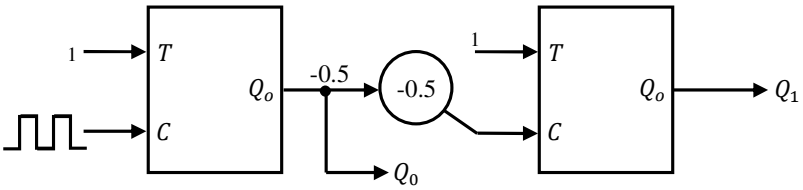

Figure 8. The two-bit up-counter

3. SIMULATION RESULTS

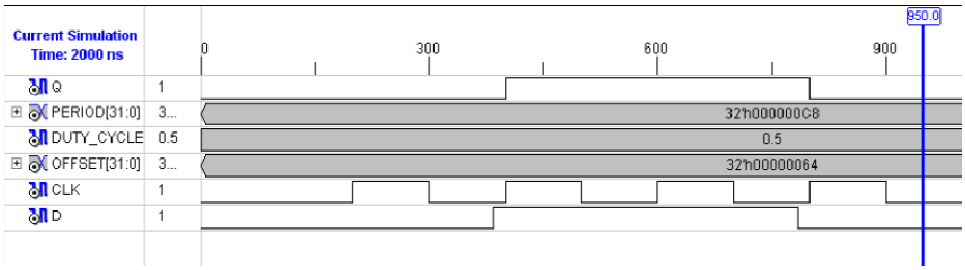

Figure 9. The D flip-flop output

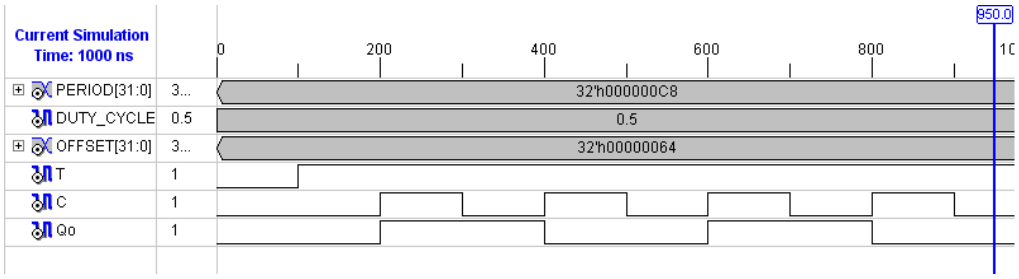

Figure 10. The T flip-flop output

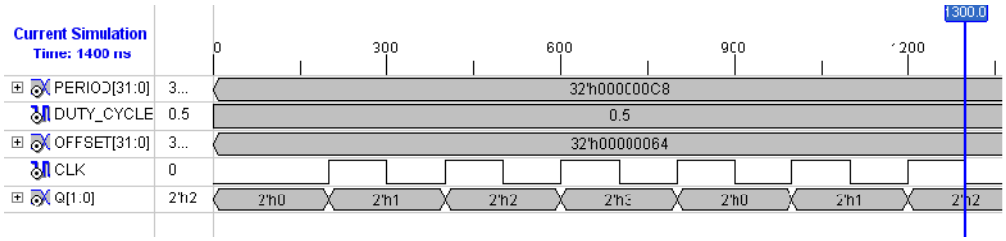

Figure 11. The two-bit up-counter output

4. CONCLUSION

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions. [10, 11]

The perceptron learning rule is very simple, but it is also quite powerful. The rule will always converge to a correct solution, if such a solution exists. The weakness of the perceptron network lies not with the learning rule, but with the structure of the network. The standard perceptron is only able to classify vectors that are linearly separable.

Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. One of the preferred techniques for gesture recognition.
3. Multi-layer Perceptrons/Neural networks do not make any assumption regarding the underlying probability density functions or other probabilistic information about the pattern classes under consideration in comparison to other probability based models.
4. They yield the required decision function directly via training.

REFERENCES

[1] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", Bulletin of Mathematical Biophysics, Vol. 5, pp. 115-133, 1943.

[2] Rosenblatt, Frank (1957), The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

[3] David A. McCormick, "Neuronal Networks: Flip-Flops in the Brain", Current Biology, Volume 15, Issue 8, 26 April 2005, Pages R294-R296.

[4] Grossberg, "Contour enhancement, Short-term Memory, and Constancies in Reverberating Neural Networks". Studies in Applied Mathematics, 52 (1973), pp. 213-257.

[5] Freund, Y. and Schapire, R. E. 1998, "Large margin classification using the perceptron algorithm", in Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT' 98), ACM Press.

[6] Rui Zhang, Pallav Gupta, Lin Zhong, and Niraj K. Jha, Department of Electrical Engineering, Princeton University, Princeton, NJ 08544, "Synthesis and Optimization of Threshold Logic Networks with Application to Nanotechnologies".

[7] Chapter 3 Weighted networks - the perceptron and chapter 4 Perceptron learning of Neural Networks - A Systematic Introduction by Raúl Rojas (ISBN 978-3-540-60505-8).

[8] W. Krauth and M. Mezard, "Learning algorithms with optimal stability in neural networks". J. of Physics A: Math. Gen. 20: L745-L752 (1987).

[9] Marifi Güler, The Neuron as a Computational Unit, Hints from Life to AI, edited by Ugur HALICI, METU, 1994 ã.

[10] Michael Black, "Applying Perceptrons to Speculation in Computer Architecture- Neural Networks in Future Microprocessor"s, VDM Verlag Dr.Mueller e.K (August 14, 2007).

[11] M. C. Su, W. F. Jean, and H. T. Chang, 1996, "A Static Hand Gesture Recognition System Using a Composite Neural Network," in Fifth IEEE Int. Conf. on Fuzzy Systems, pp. 786-792, New Orleans, U.S.A. (NSC85-2213-E-032-009).

AUTHORS

Ratnadip Dey Iwas born in Raniganj (W.B.), India. I received my B. Tech. degree in Electronics and Communication Engineering from Asansol Engineering College, Asansol, India in 2009. Currently, I am pursuing M. Tech. in Microelectronics & VLSI from Institute of Engineering & Managem ent, (WBUT), Kolkata, India. My research interest is Digital Integrated Circuit design, Advanced Communication-based System design, VLSI Low Power System design.

Debdut Biswas was born in Kolkata (W.B.), India. I received my B. Tech. degree in Electronics and Communication Engineering from RCC Institute of Information Technology, Kolkata, Indi a in 2010. Currently, I am pursuing M. Tech. in Microelectronics & VLSI from Institute of Engineering & Management, (WBUT), Kolkata, India. My research interest is Microprocessor & Microcontroller-based System design, Optics-based Sensors, Mixed Signal System design.

Pijush Dutta received the B. Tech. degree in Electronics & Communication Engineering from B.P. Poddar Institute of Management & Technology, Kolkata, India in 2007 and M. Tech. degree in Mechatronics Engineering at N.I.T.T.T.R., Kolkata in 2012. I have recently joined as an assistant professor in Global Institute of Management & Technology (2012 – till date).