Training Artificial Neural Networks by PSO to Perform Digital Circuits Using Xilinx FPGA

Dr. Hanan A. R. Akkar & Firas R. Mahdi*

Received on: 30/6/2010
Accepted on: 7/4/2011

Abstract

One of the major constraints on hardware implementations of Artificial Neural Networks (ANNs) is the amount of circuitry required to perform the multiplication process of each input by its corresponding weight and there subsequent addition. Field Programmable Gate Array (FPGA) is a suitable hardware IC for Neural Network (NN) implementation as it preserves the parallel architecture of the neurons in a layer and offers flexibility in reconfiguration and cost issues. In this paper the adaption of the ANN weights is proposed using Particle Swarm Optimization (PSO) as a mechanism to improve the performance of ANN and also for the reduction in the ANN hardware. For this purpose we modified the MATLAB PSO toolbox to be suitable for the taken application. In the proposed design training is done off chip then the fully trained design is download into the chip, in this way less circuitry is required. This paper executes four bit Arithmetic Logic Unit (ALU) implemented using Xilinx schematic design entry tools as an example for the implementation of digital circuits using ANN trained by PSO algorithm.

Keywords: Particle Swarm Optimization (PSO), Artificial Neural Network (ANN), Field Programmable Gate Array (FPGA).

تدريب الشبكات العصبية الاصطناعية بواسطة أفضلية الحشد الجزيئي لتنفيذ الدوائر الرقمية باستخدام مصفوفة البوابات القابلة للبرمجة

الخلاصة

تعتبر كمية الدوائر اللازمة لتنفيذ عملية ضرب كل إدخال بالوزن المرافق له وعملية الجمع اللاحقة لها إحدى المعوقات الرئيسية لتنفيذ دوائر الشبكات العصبية الاصطناعية (ANN). مصفوفة البوابات القابلة للبرمجة (FPGA) هي دائرة متكاملة مناسبة لبناء الشبكات العصبية الاصطناعية لأنها تحافظ على البناء المتوازي للخلايا العصبية الاصطناعية في الطبقة الواحدة وتعرض مرونة في إطار إعادة البرمجة بالإضافة إلى قضايا الكلفة. تم في هذا البحث اقتراح تبني أوزان الشبكات العصبية الاصطناعية باستخدام أفضلية الحشد الجزيئي (PSO) كإلية لتحسين أداء الشبكات العصبية الاصطناعية بالإضافة إلى تقليل الدوائر اللازمة لبناء الشبكات. لهذا الغرض قمنا بتعديل صندوق أدوات أفضلية الحشد الجزيئي في بيئة الماتلاب ليكون مناسب للتطبيق المأخوذ. في التصميم المقترح تمت عملية التدريب خارج الرقاقة ثم حمل التصميم كامل التدريب إلى الرقاقة, بهذه الطريقة سنحتاج اقل عدد من الدوائر. ينفذ هذا البحث دائرة حساب ومنطق (ALU) ذات أربع بت باستخدام أدوات الرسم التخطيطي لزايلنكس كمثال لتطبيق الدوائر الرقمية باستخدام الشبكات العصبية الاصطناعية المرنة بواسطة خوارزمية أفضلية الحشد الجزيئي.

* Electrical and Electronic Engineering Department, University of Technology/ Baghdad

https://doi.org/10.30684/ etj.29.7.8
University of Technology-Iraq, Baghdad, Iraq/2412-0758

Introduction

Swarm Intelligence

Swarm Intelligence (SI) is a modern artificial intelligence discipline that is concerned with the design of multi agent systems. The design paradigm for these systems is fundamentally different from many traditional approaches. Instead of the sophisticated controller that governs the global behavior of the system, the SI principle is based on many unsophisticated entities that cooperate in order to exhibit a desired behavior. Inspiration for the design is taken from the collective behavior of social insects such as ants, termites, bees and wasps, as well as from the behavior of other animal societies such as flocks of birds or schools of fish. Even though the single members of these societies are unsophisticated individuals, they are able to achieve complex task in cooperation [1].

Particle Swarm Optimization

PSO is an emerging population based optimization method, a parallel evolutionary computation technique originally designed by Kennedy and Eberhart in 1995 [2]. The basic concept of PSO basically comes from a large number of birds flying randomly and looking for food together. Each bird is an individual called a particle. As the bird looking for food, the particles fly in a multidimensional search space looking for the optimal solution. All the particles are composed of family rather than the isolated individual for each other. They can remember their own flying experience. According to the cognitive memory, all the particles can adjust their position moving toward.

Their global best position or their neighbor's local best position. PSO has a few parameters to adjust, so that its convenient to make the parameters reach to the optimum values, a large amount of calculation work and much time can be saved, on the other hand, PSO can find the optimum solutions or near the optimal solutions with a fast convergent speed, because it has only two computation formulas for iteration [3].

The most popular formulation of how particle adjusts its velocity and position are shown in equations (1) and (2).

$$VI_{(t+1)}(d) = Wv_{it}(d) + C_1r_1\,(Pb_i(d)-x_{it}(d)) + C_2r_2\,(Gb_i(d)-x_{it}(d)) \quad \dots (1)$$

$$Xi_{(t+1)}\,(d) = x_{it}(d)+v_{i(t+1)}(d) \quad \dots (2)$$

Where, d is the index of dimension in the search space, W represents the inertia weight, C_1 and C_2 are regarded as cognitive and social parameters for algorithm respectively, r_1 and r_2 are two random numbers. Pb_i is the personal best position recorded by particle i, while Gb_i is the global best position obtained by any particle in the population [4].

Artificial Neural Network

The discipline of NNs originates from the understanding of the human brain. The average human brain consists of $3*10^{10}$ neurons of various types, with each neuron connecting to up to 10^4 synapses processing information separately and simultaneously [5]. ANN consists of a number of very simple and highly interconnected processors, which are analogous to the biological neuron. Each neuron receives a number of input signals through its connections, producing a single output signal. The output signal is transmitted through the neuron outgoing connection. Neurons are connected by links; each link has a numerical weight associated with it. Weights are the basic means of long term memory in ANNs. ANN learns through repeated adjustments of these weights [6].

Each neuron computes the weighted sum of the input signal applied some activation function to the net sum then firing an output according to the used activation function, thus the output of the neuron could be depicted as in equation (3).

$$Y_i=f_i(\sum_{j=1}^{n} wij * xj - \theta i) \quad ...(3)$$

Where, Y_i is the output of neuron i, x_j is the j^{th} input to the neuron, w_{ij} is the connection weight between neuron i and j input, θ_i neuron i bias and f_i is the activation function corresponding to neuron i [7].

Field Programmable Gate Array

FPGA is a prefabricated silicon device that can be electrically programmed to become almost any kind of digital circuit or system [8]. FPGA architecture is dominated by its programmable interconnects, and configurable logic blocks which are relatively simple. However, these devices are more flexible than other devices like Complex Programmable Logic Device (CPLD) especially in terms of the range of designs [9].

FPGA based reconfigurable computing architecture are well suited to implement ANNs as one can develop concurrency and rapidly reconfigure to adapt the weights and topologies of an ANN [10]. FPGA realization of ANN with large number of neurons is still a not easy task because ANN algorithm is wealthy with multiplication process and it's relatively expensive to realize. Various works reported in this area includes new multiplication algorithm for ANN, NNs with some constraints to achieve higher speed of process at lower price and multichip realization [11- 13].

Theory Description

ANNs have been successfully used in a wide range of scientific and engineering applications. ANN is capable of exhibiting intelligent behavior and modeling complex non linear functions which makes it proper to variable conditions. Training is the process of gradually adjusting the weight of connections. BP algorithm is mainly used to train ANNs for many applications. Since this algorithm is based on gradient descent which demands derivatives, it's complex and prone to get trapped in local optima. It also has a slow convergence rate and a high resources requirement on hardware. PSO is one of the evolutionary computation techniques based on SI; PSO algorithm has been employed for training feed forward NNs.

In NN training, the main goal is to obtain a set of weights that minimizes error. In order to address the problem of NN training to PSO; we represent each set of weights and biases of a network by a single particle. Thus, each particle is a string of continuous valued numbers encoding a candidate solution for weights and biases of all neurons in the network the length of the particle depends on the network intended to train. A pool of particles is considered as a swarm (population) for PSO. By repetitively updating particles of the swarm, the most suited network weights are gradually determined. Different stopping criteria are used. One criterion is to update the particles until the error between actual and target output is lower than a given threshold. Stopping the training processes after a given period without any improvement in training is the next criterion. Another criterion is training the network for a given iteration [14].

PSO has its own advantages and drawbacks over other computational algorithms, advantages like its probabilistic mechanism and multi starting points, hence PSO can avoid getting into the local optimal solution [15], but the most utilized PSO property is its free derivative activation function, which means that we will train feed forward NNs using PSO as the learning algorithm with only hard limit activation function for all network layers. According to the hard limit activation function properties the output will be either one or zero, this property will be very helpful simplifying the network multiplication process.

For the purpose of NN implementation of digital logic circuits we have modified the MATLAB PSO toolbox to be more suitable with our application. The modification is in the search space environment, instead of searching all real search space values the searching will be restricted to the integer search space only which means more save in term of time and efforts. PSO tools will give us the exactly weights needed for the network training, these weights will be only integers numbers. These integers will be helpful in execution the multiplication process using only AND gates. The goal of training ANN using modified MATLAB toolbox is to get zero error value as depicted in equation (4).

$$E=\frac{\sum_{i=1}^{m}\sum_{k=1}^{n}(t_i^k-y_i^k)^\wedge 2}{mn} \quad \dots (4)$$

Where, t_i^k and y_i^k represent the actual and the predicted function values respectively, m is the number of training samples, and n is the number of output nodes.

The Proposed Design of PSO Neuron

Hardware realization of ANN depends on the efficient execution of single neuron; one of the major constraints on hardware implementations of NNs is the amount of circuitry required to perform the multiplication of each input by its corresponding weight and their subsequent addition. This problem is especially acute in digital designs, where parallel multipliers and adders are extremely expensive in terms of circuitry.

FPGAs provide different design choices to be evaluated in a short time and keep system cost at a minimum. For the proposed design, training of ANN using modified PSO toolbox will be done off chip, and a fully trained configuration is downloaded into hardware. In this way less hardware is required. As training occurs only once in the lifetime of an application training off line method does not reduce the network's functionality.

Constructing digital circuits using ANNs means that the input will be restricted between two values one and zero, when the NN is trained with only integer weights values, the multiplication process will not need more than AND gates. Suppose a neuron with a single input and weights are blocked in the integer range of [-3, 3], two bits will represent the weight and a single bit represents the weight's sign. Therefore three AND gates are sufficient to represent the multiplication process with sign, where 0 stands for the positive sign and 1 stands for the negative sign. Figure (1) shows single input two bit weight multiplication process. This structure will be repeated for each input. Product produced by the multiplication process will be added or subtracted according to the weights signs using special designed Adder/Subtracter with sign digital circuit. Figure (2) shows a two bit Adder/Subtracter with sign logic circuit. Adder/Subtracter with sign digital circuit will consist of: maximum numbers block, magnitude comparator, full adders, 4*1 multiplexer, AND gates and XOR gates. The function of the maximum number block is to put the maximum number of two input numbers on its X output and the smallest number on its Y output. Figure (3) shows two bit maximum number block logic circuit. Since hard limit activation function has been applied for all network layers, therefore the output of the neurons will be 1 if the net (final neuron input weights product summation) greater or equal zero, and 0 if the net is less than zero. Figure (4) shows two inputs, two bits weight neuron logic circuit. Output of the neuron will be the same as the inverted sign of the final neuron net. Figure (5) shows the inverted neuron output.

Arithmetic Logic Unit

ALU performs all the necessary arithmetic and logic operations. It requires one or two operands upon which it operates and produces result. It's basically a multifunction combination logic circuit. It provides select inputs to select the particular operation. The popular ALU IC, IC74LS181, is a four bit high speed parallel ALU, controlled by the four select input (S_0- S_3) and the Mode control (M). It can perform all the 16 possible logic function operations or 16 different arithmetic operation on active HIGH or active LOW operand. Table (1) shows the function table of the ALU operations [16]. Figure (6) shows the logical diagram of the DM 74LS181 4-bit ALU.

Since, we have a large data set 16,384 due to 14 inputs we will divide the network into 6 parts with maximum 8 inputs to simplify the design. Figures (7, 8, 9, 10, 11, 12) show part (1, 2, 3, 4, 5, 6) ALU logic diagrams respectively, figures (13, 14, 15, 16, 17, 18) show part (1, 2, 3, 4, 5, 6) error against iteration respectively. All parts will be designed on Xilinx XC 3000 chip.

Part One ALU

Part 1 will be repeated four times in the overall ALU design using only 6 neurons in the input layer and 2 neurons in the output layer. The training parameters are given by: W=0.6, C_1= C_2=1.7 these parameters will be repeated for all designs. No. of particles 1000, weights are arranged in the integer range between [-7, 7] therefore 4 bits will be sufficient to represent this range, three bits for the weight and a single bit for weight's sign. Weights obtained by the PSO modified tools are: W{1, 1}=[-4 -3 0 0 -1 -2; -5 -7 2 -2 1 0] and B{1}= [7; 5]. Error goal reached successfully termination after 19 iterations. Figure (19), shows the hardware design of part 1 ANN based on FPGA.

Part Two ALU

Part 2 will be represented by 4 neurons in the input layer, 3 neurons in the hidden layer and single neuron in the output layer. Part 2 training parameters are: No. of particles 10000, minimum range for weights required for the network training arranged in the integer range of [-3, 3] therefore, three bits are sufficient to represent the weight and its sign. Error goal reached successful termination after 31 iterations. Weights obtained by modified PSO tools are: W{1, 1}=[2 2 1 -1;2 1 -1 0;-1 1 2 -1], W{2,1}=[2 -2 -2], B{1}=[-3;-1;-2] and B{2}= 1. Figure (20), shows part 2 ALU ANN hardware design.

Part Three ALU

Part 3 will be represented by 6 neurons in the input layer, 3 neurons in the hidden layer and 1 neuron in the output layer. Part 3 training parameters are: No. of particles 10000, minimum weights range required for the training process will be blocked in the integer range between [-7, 7]. Error goal reached successful termination after 55 iterations. Weights obtained by PSO modified tools are: W{1,1}=[4 0 7 -4 1 -4; -6 -1 -3 -3 -1 7; 2 1 -5 5 1 -3], B{1}=[-2;4;-2] , B{2} = 7 and W{2, 1}=[-5 -5 -3]. Figure (21) shows part 3 ALU hardware ANN design.

Part Four ALU

Part 4 will be represented by 8 neurons in the input layer, 3 neurons in the hidden layer and single neuron in the output layer. Part 4 ALU training parameters are: No. of particles 1000, minimum weights range required for the training process will be blocked in the integer range between [-31, 31]. Error goal reached successful termination after 124 iterations. Weights obtained by the PSO modified tools are: W{1, 1}=[-2 -1 -5 -3 -8 28 -1 13; -4 -1 -9 -6 -30 30 -2 15; 2 1 12 3 25 -31 1 -19], B{1}=[18; 8; 26], W{2, 1}=[-23 23 14] and B{2}= -9. Figure (22) shows part 4 ALU ANN design.

Part Five ALU

Part 5 will be implemented using 8 neurons in the input layer and single output neuron. Part 5 training parameters are: No. of particles 1000, minimum weights required to reach the intended error goal are arranged in the integer range between [-31, 31]. Error goal reached successful termination after 27 iterations, weights obtained by PSO modified tools are: W{1, 1}=[-2 -1 -5 -3 -14 -9 -1 26] and B{1}=13. Figure (23) shows part 5 ALU hardware design.

Part Six ALU

Part 6 will be implemented by 3 neurons in the input layer, three neurons in the hidden layer and single output neuron. Part 6 training parameters are: No. of particles 100, minimum weights required to reach the intended error goal are arranged in the integer range between [-3, 3]. Error goal reached successful termination after 62 iterations, weights obtained by PSO modified tools are: W{1, 1}=[- 2 1 -2; 0 2 1; -2 -2 -2] and B{1}={1; -2; 3}, W{2, 1}=[-1 1 1], B{2}= -1. Figure (24) shows part 6 ALU ANN hardware design. Figure (25) shows the overall 4 bit ALU design and figures (26) (a, b, c) represent random output readings from the overall ALU using Xilinx foundation series logic simulator.

Conclusions

This paper proposes a hardware design of an ANN using FPGA. FPGA is chosen mainly because of the lower price as compared to other technologies, parallel implementation of ANN using software as well as with hardware approaches.

The objective of this paper was to reduce the number of neurons needed for the training process, as well as reducing the single neuron complexity by the abstraction of the multiplication process needed to multiply each input by the corresponding weight. PSO optimization algorithm is the most suitable training algorithm required to our application implementing of digital circuits using ANN. NNs are trained by minimizing the error function in search space based on weights. PSO generates possible solutions and measure their quality by using a forward propagation through the NN to obtain the value of error function (minimized error to zero). This error value is used as the particle's fitness function to direct it towards more promising solution. The global best particle corresponded to the desired trained after adequate iterations. The design of NN on to an FPGA is a relatively simple process. Once the training is completed and the correct network weights are determined, these weights will be hard coded on the FPGA, and since we have taken these weights as integers, these integers will be represented as a binary digital bits one and zero entered to the designed ANN hardware bus as a VCC and ground, where the VCC stands for binary 1 and the ground stands for binary 0. The accuracy in which these weights are coded depends upon the number of the available bits. The MATLAB modified PSO toolbox is used in the training of the ANNs minimum weights required to get the highest accuracy (100%) which was calculated using trail and error method.

Refrences

[1] C. Blum, D. Merkle, "Swarm Intelligence: Introduction and Application", Springer, 2008.

[2] J. Kennedy and R. Eberhart, " Particle Swarm Optimization", IEEE Int. Conf. on Neural Networks, Australia, PP. 1942-1948, 1995.

[3] L. Wang, X. Wang, J. Fu and L. Zhen, "A Novel Probability Binary Partical Swarm Optimization Algorithm and its Application", Academy publisher, Journal of software, China, Vol. 3, No. 9, December, 2008.

[4] T. Gong and A. L. Tuson, "Particle Swarm Optimization for Quadratic Assignment Problem A Forma Analysis Approach", International Journal of Computational Intelligence Research, Vol. 4, No.2, PP. 177-185, 2008.

[5] K. Du and M. Swamy, "Neural Network in a Soft Computing Framework", Springer, 2006.

[6] M. Negnevitsky, "Artificial Intelligence: A Guide to Intelligent Systems. (2nd edition)", Addison Wesley, 2005.

[7] X. Yao, "Evolving Artificial Neural Network", IEEE, Vol. 87, No. 9, September, 1999.

[8] I. Kuon, R. Tessier and J. Rose, "FPGA Architecture: Survey and Challenges", Now Publisher, 2008.

[9] A. K. Mani, "Digital Electronics: Principles, Devices and Applications", Wiley, 2007.

[10] A. Muthuramalingam, S. Himavathi and E. Srinivasan,

"Neural Network Implementation Using FPGA: Issues and Application", International Journal of Information Technology, India, Vol. 4, No. 2, PP. 86-92, April, 2007.

[11] R. H. Turner, R. F. Woods, "Highly Efficient Limited Range Multipliers for LUT- Based FPGA Architecture", IEEE transactions on vary large scale integration system, Vol. 15, No.10, PP. 1113-1117, 2004.

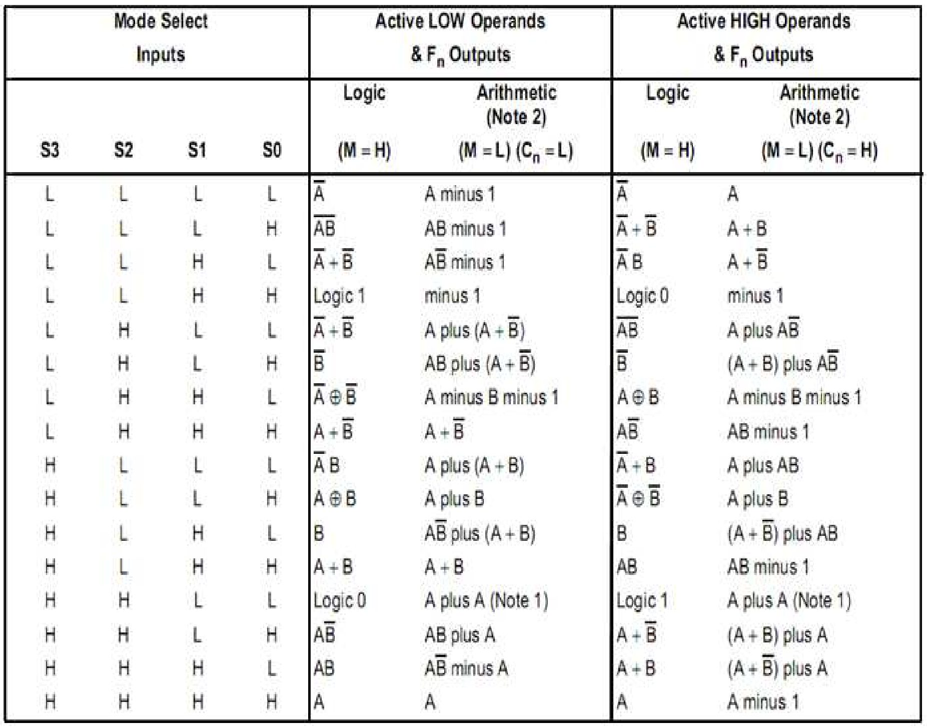
[12] M. Marchesi, G. Orlandi, F. piazza and A. Uncini, "Fast Neural Network without Multipliers", IEEE transactions on NN , Vol.4, No.1,1993. [13] B. Noory and V. Grozo, "A Reconfigurable Approach to Hardware Implementation of Neural Network", IEEE Canadian conference on Electrical and Computer Engineering, PP.1861-1863, 2003.

[14] A. Farahani, S. Fakhraie, S. Safari, "Scalable Architecture for On-Chip Neural Network Training Using Swarm Intelligence", EDAA, 2008.

[15] R. Mendes, P. Cortez, M. Rocha and J. Nevers, "Particle Swarm for Feed forward Neural Network Training", IEEE Transactions, PP. 1895-1899, June, 2002.

[16]A. P. Godse, D. A. Godse, "Digital Electronics", Technical Publications Pune, 2008.

Table 1 ALU operation



Mode Select Inputs				Active LOW Operands & F_n Outputs		Active HIGH Operands & F_n Outputs	
				Logic	Arithmetic (Note 2)	Logic	Arithmetic (Note 2)
S3	S2	S1	S0	(M = H)	(M = L) (C_n = L)	(M = H)	(M = L) (C_n = H)
L	L	L	L	$\overline{A}$	A minus 1	$\overline{A}$	A
L	L	L	H	$\overline{AB}$	AB minus 1	$\overline{A} + \overline{B}$	A + B
L	L	H	L	$\overline{A} + \overline{B}$	$A\overline{B}$ minus 1	$\overline{A}$ B	$A + \overline{B}$
L	L	H	H	Logic 1	minus 1	Logic 0	minus 1
L	H	L	L	$\overline{A} + \overline{B}$	A plus $(A + \overline{B})$	$\overline{AB}$	A plus $A\overline{B}$
L	H	L	H	$\overline{B}$	AB plus $(A + \overline{B})$	$\overline{B}$	(A + B) plus $A\overline{B}$
L	H	H	L	$\overline{A} \oplus \overline{B}$	A minus B minus 1	$A \oplus B$	A minus B minus 1
L	H	H	H	$A + \overline{B}$	$A + \overline{B}$	$A\overline{B}$	AB minus 1
H	L	L	L	$\overline{A}$ B	A plus (A + B)	$\overline{A} + B$	A plus AB
H	L	L	H	$A \oplus B$	A plus B	$\overline{A} \oplus \overline{B}$	A plus B
H	L	H	L	B	$A\overline{B}$ plus (A + B)	B	$(A + \overline{B})$ plus AB
H	L	H	H	A + B	A + B	AB	AB minus 1
H	H	L	L	Logic 0	A plus A (Note 1)	Logic 1	A plus A (Note 1)
H	H	L	H	$A\overline{B}$	AB plus A	$A + \overline{B}$	(A + B) plus A
H	H	H	L	AB	$A\overline{B}$ minus A	A + B	$(A + \overline{B})$ plus A
H	H	H	H	A	A	A	A minus 1

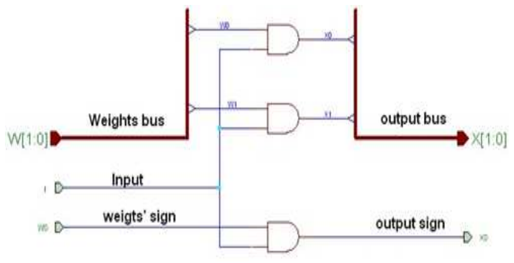


Figure.1 Two bits with sign multiplication digital circuit.

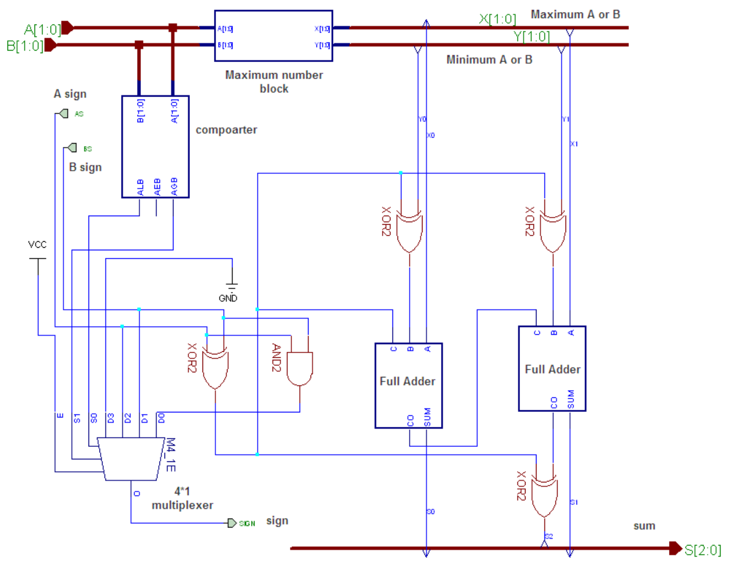


Figure.2 Two bits Adder/Subtracter circuit.

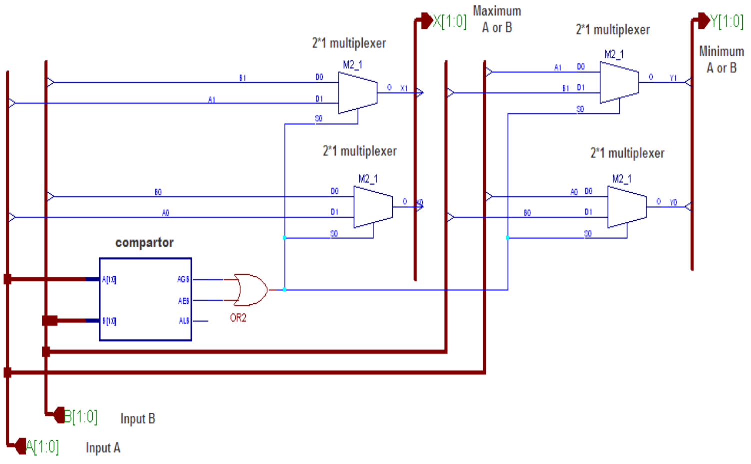


Figure.3 Two bits maximum number logic diagram.

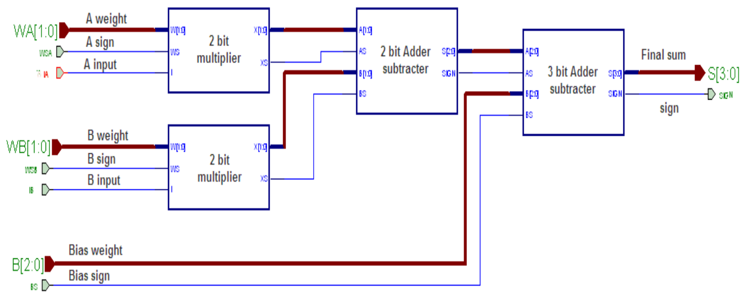


Figure.4 Two inputs neuron with two bits weight circuit.

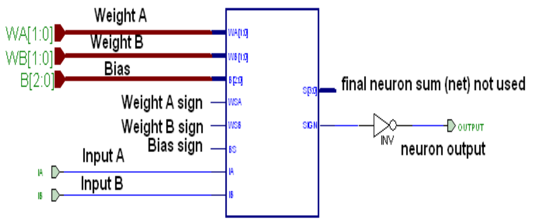


Figure.5 Two inputs neuron with two bits weight circuit.

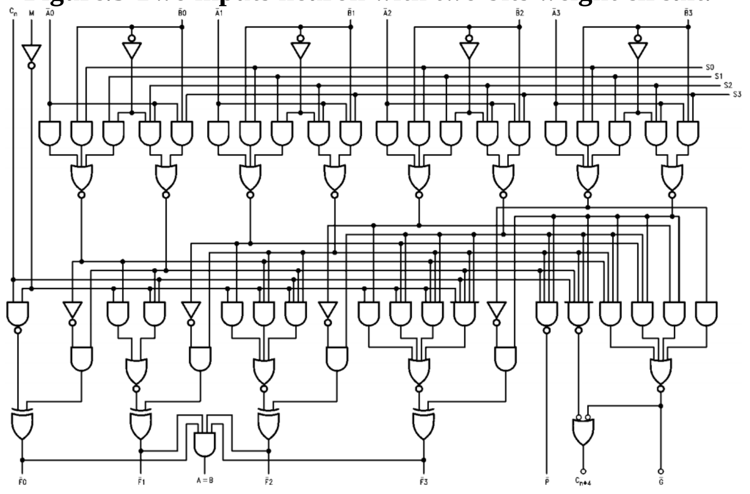
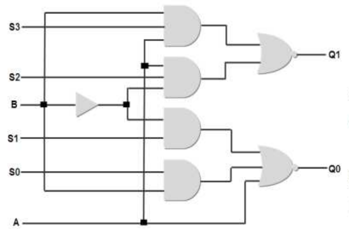
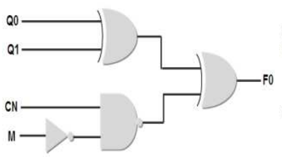


Figure.6 DM 74LS181 4-bit ALU logic diagram.





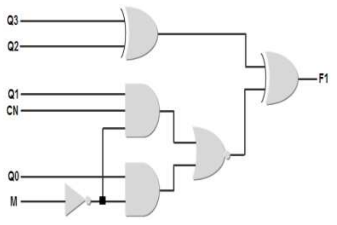
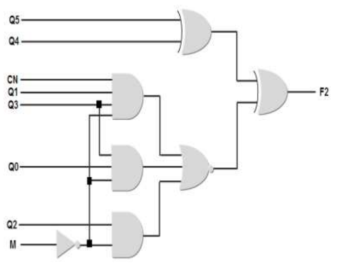
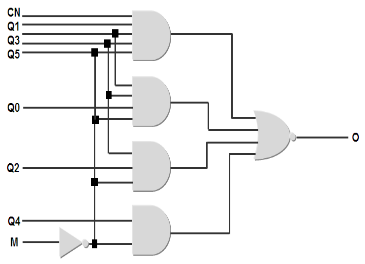


Figure.7 Part 1 logic diagram. Fig.8 Part 2 logic diagram. Fig.9 Part 3 logic diagram.





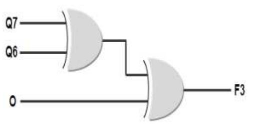
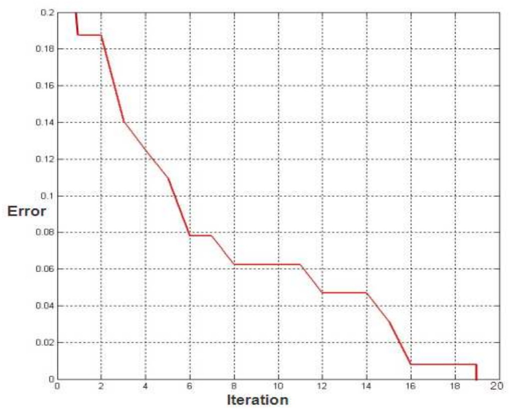


Figure.10 Part 4 logic diagram. Fig.11 Part 5 logic diagrams. Fig.12 Part 6 logic diagrams.



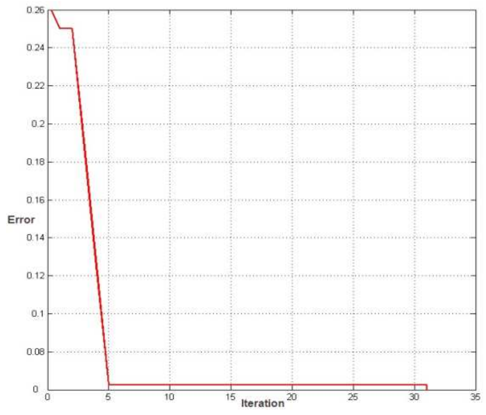
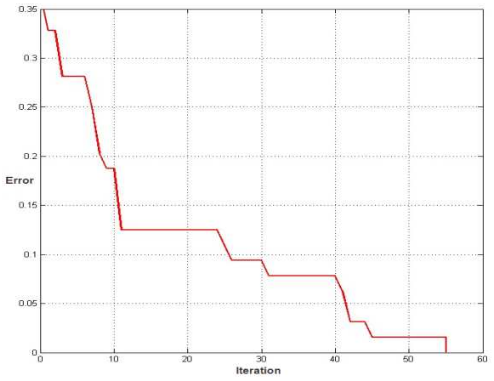


Figure.13 Error against iteration part 1 ALU. Figure.14 Error against iteration part 2 ALU.



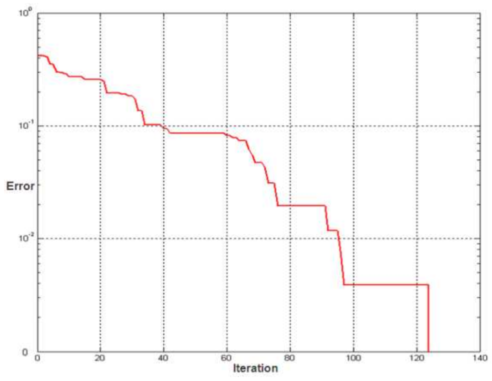
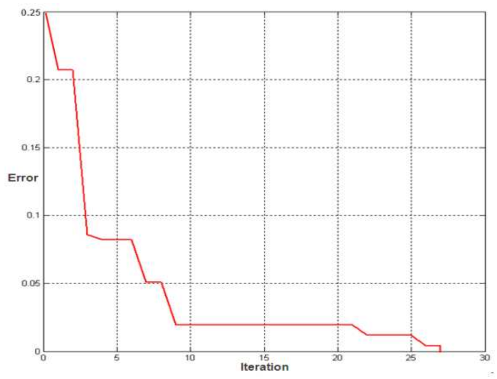


Figure.15 Error against iteration part 3 ALU. Figure16 Error against iteration part 4 ALU.



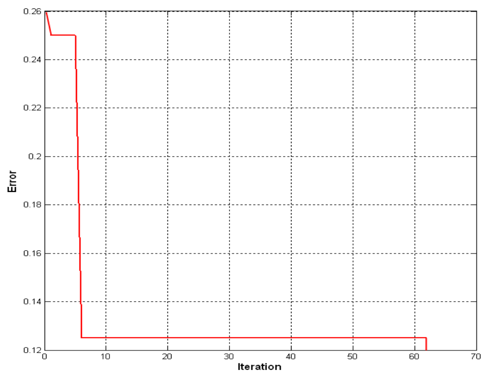


Figure.17 Error against iteration part 5 ALU. Figure.18 Error against iteration part 6 ALU.

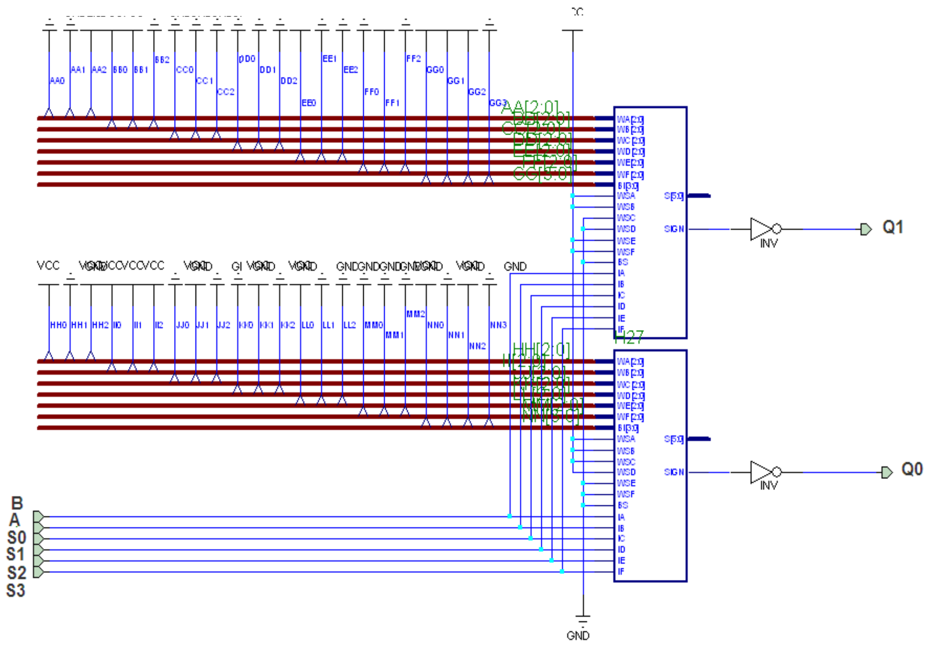


Figure.19 Part1ANNimplementation.

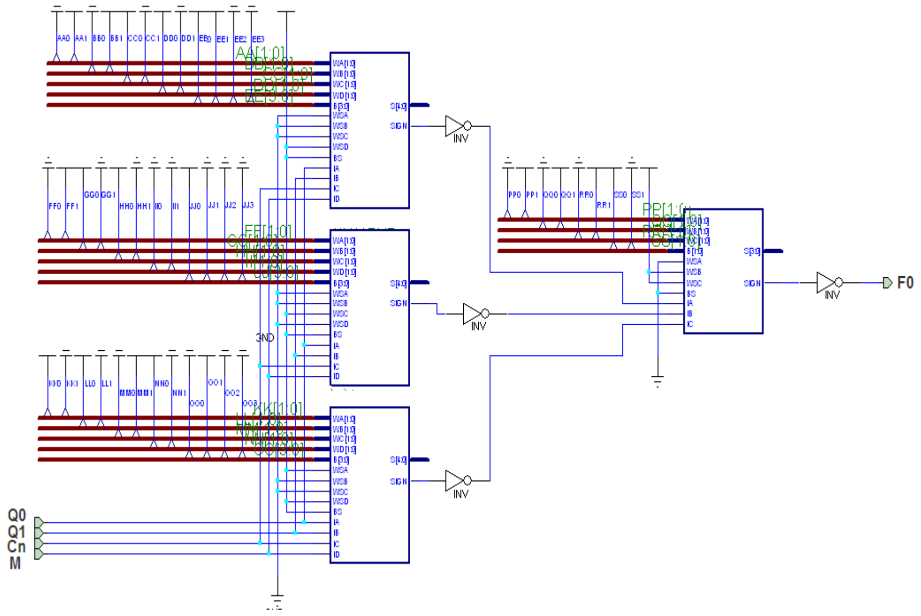


Figure.20 Part2 ANN implementation.

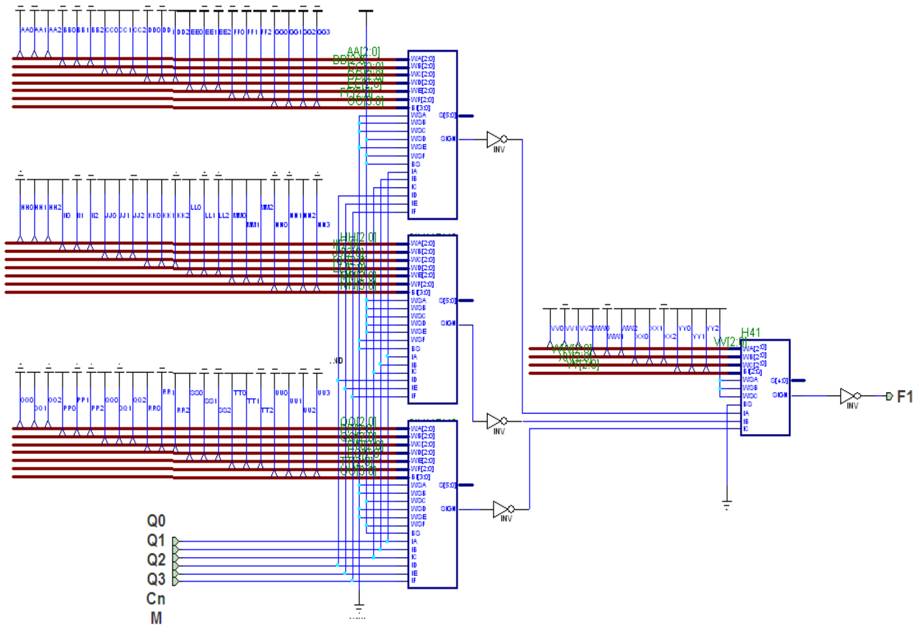


Figure.21 Part3 ANN implementation.

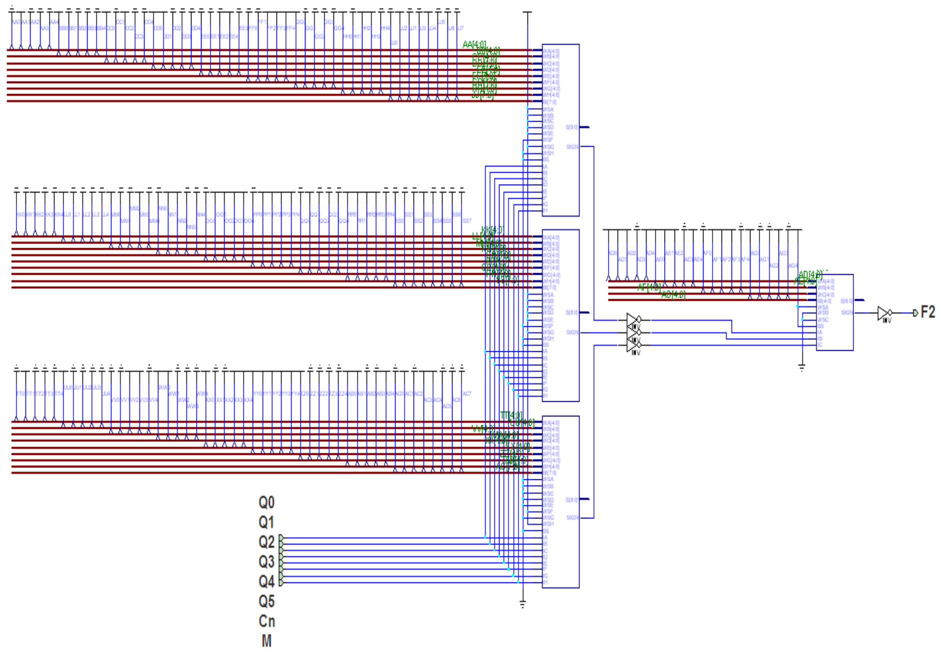


Figure.22 Part4 ANN implementation.

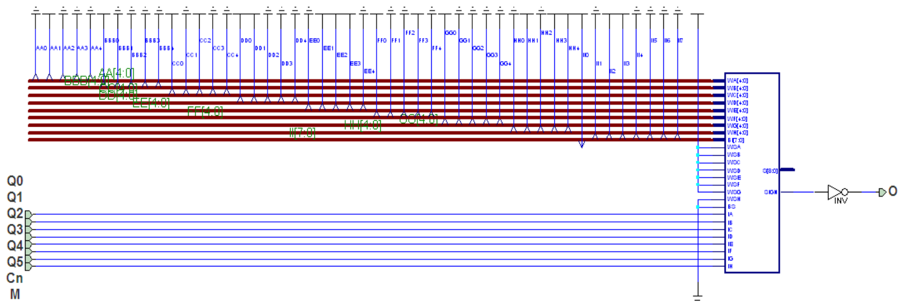


Figure.23 Part5 ANN implementation.

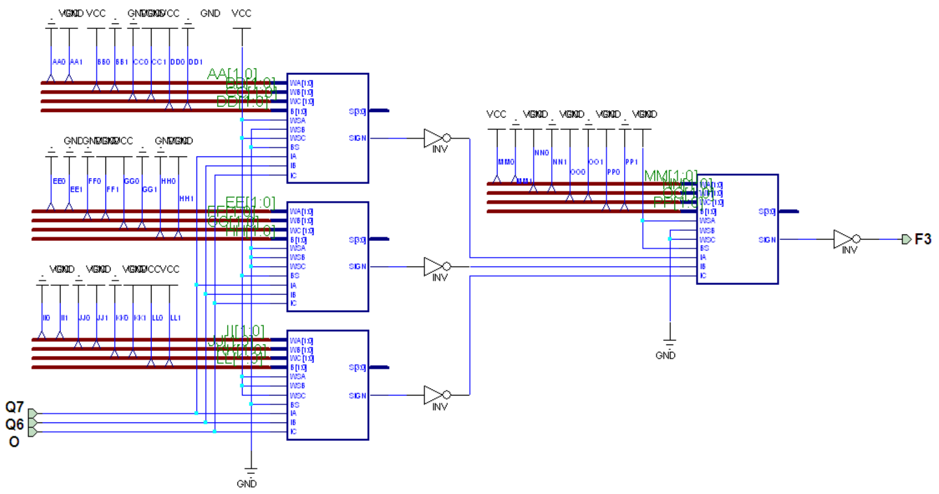


Figure.24 Part6 ANN implementation.

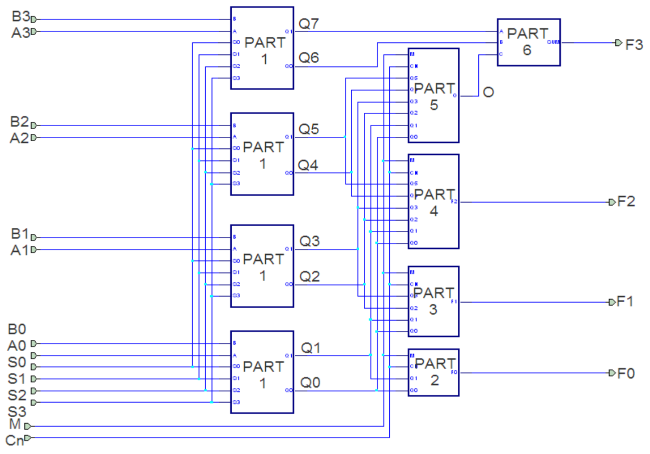
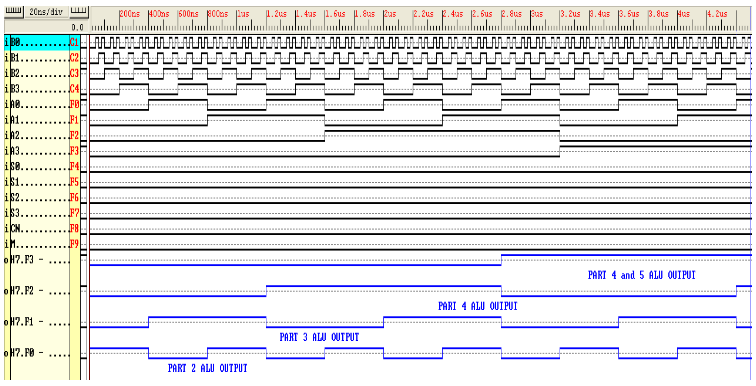
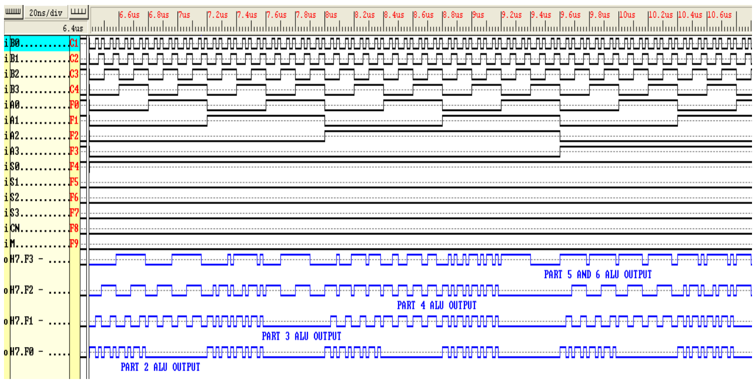


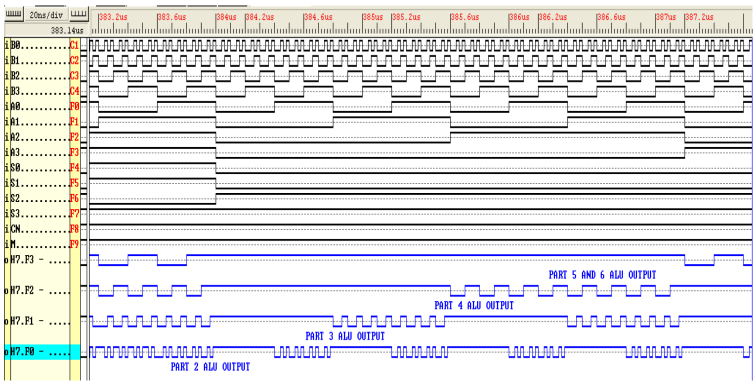
Figure.25 Overall 4-bit ALU block diagram.



(a)



(b)



(c)

Figure. (26) 4bit ALU random simulation reading.