Design of a Stochastic Re-Configurable Artificial Neural Networks Using FPGA

Dr. Mutaz S. Abdul-Wahab* Dr. Hannan A. R. Akkar* Dr. Manal H. Jassim*

Received on: 3/10/2007
Accepted on: 10/6/2009

Abstract

This paper uses the theory of stochastic arithmetic as a solution for the FPGA implementation of a complex feed forward, multi layered neural network. Compared with the traditional digital implementations, the stochastic approach simplifies the computation involved and saves digital resources. The architecture combines stochastic computation techniques with a novel Look Up-Table-based that fully exploits the Look Up-Table structure of many FPGAs. Basic operations of simple ANN are mapped into a modular design. The system control module, random pulse generating module , bit stream generating module , LFSR_32(Liner Feedback Shift Register) sub module, modulator sub module, neuron module and bit stream converter module , are described in hardware using a schematic editor of the Foundation 4.1i, which is a software tool from Xilinx. Thus the modules can be parameterized, providing easy scalability of the system to the different applications constraints and requirements. The feasibility of the proposed ANN is demonstrated by testing it using two case studies. The objective of the first test is the to decomposition of Boolean Function sets (AND, OR, EXOR) the simulation results show that the design is able to find the obtainable values for the functions, while, the objective of the second test is to find the frequency recognition for square wave with different frequencies, the simulation results show that the design is suitable for using in this field.

الخلاصة

يتضمن العمل تصميم و تنفيذ نظام متكامل للشبكات العصبية باستخدام (Field Programmable Gate Array) بمساعدة النظام الأساسي 4.1i .حيث يقدم تصميم متكامل لشبكه عصبية متعددة الاستخدامات(باعتماد تقنية حسابيه نوع Stochastic Computing) يتم وصفها باستخدام أحد أدوات وصف المكونات ألصلبه للدوائر المتكاملة الفائقة السرعة Schematic Editor والتي تعد أحد أدوات وصف المكونات ألصلبه HDLs . حيث يتم تنفيذ النظام الكلي على أحدى رقائق مصفوفة البوابات المبرمجة باستخدام البرنامج الأساسي 4.1i وهو أداة برمجيه توفرها Xilinx.يتم تحويل العمليات الأساسية المكونة لهذه الشبكة الذكية إلى تصميم مكون من وحدات أساسية. حيث يتم وصف الوحدات الأساسية ألمكونه للشبكة الذكية والتي تشمل وحدة السيطرة System Control Module , وحدة توليد النبضات العشوائية Random Pulse Generation Module , وحدة توليد سلسلة البت Bit Stream Generating Module , وحدة النيرون Neuron Module و وحدة تحويل البت Bit Stream Converter Module بواسطة Schematic Editor وهذا يوفر أمكانية بناء تلك الوحدات والتحكم بها من قبل المصمم وعوامل أخرى خاصة بالمكونات ألصلبهHardware Parameters وبالتالي يوفر سهولة موائمة التصميم لمتطلبات التطبيقات المختلفة .وللتأكد من تصميم الشبكة المقترح تم اختبارها باستخدام حالتين تطبيقيتين,ألحاله التطبيقية الأولى هي تطبيق الدوائر ألرقميه Boolean Gates وذلك من خلال تحويل التصميم إلى شبكه ذكية لتطبيق العمليات ألرقميه Boolean Neural Network , أما الاختبار الثاني فهو تطبيق تميز الترددات ألمتغيره Frequency Recognition , وقد أثبت التطبيقان نجاح التصميم .

*Department of Electrical & Electronic Engineering University Of TechnologyBaghdad-Iraq,

1. Introduction

An artificial neural network (ANN) is a parallel and distributed network of simple non-linear processing units interconnected in a layered arrangement. Parallelism, modularity and dynamic adaptation are three computational characteristics typically associated with ANNs. FPGA-based reconfigurable computing architectures are well suited to implement ANNs as one can exploit concurrency and rapidly reconfigure to adapt the weights and topologies of an ANN.

1.1Review

All FPGA implementation of ANNs attempt to exploit the reconfigurability of FPGA hardware in one way or another. Identifying the purpose of reconfiguration sheds light on the motivation behind different implementation approaches [1].

1-[1992][10]Max van Daalen, Pete Jeavons, John Shawe-Taylor, and Dave Cohen proposed a novel technique for the generation of high speed stochastic bit streams in which the '1' density is proportional to a given value. Bit streams of this type are particularly useful in bit serial stochastic computing systems, such as digital stochastic neural networks. This proposed circuitry is highly suitable for VLSI fabrication.

2-Then in [1994][3] they present the hardware design of an extremely compact and novel digital stochastic neuron, that has the ability to generate the derivative of its output with respect to an arbitrary input. These derivatives may be used to form the basis of an on chip gradient descent learning algorithm. Then they present an expandable digital architecture that provides an efficient real time implementation platform for large neural networks [1994] [4]. The architecture makes heavy use of the techniques of bit serial stochastic computing to carry out the large number of required parallel synaptic calculations. In this design all real valued quantities are encoded on to stochastic bit streams in which the '1' density is proportional to the given quantity. The actual digital circuitry is simple and highly regular thus allowing very efficient space usage of fine graind FPGAs. Another feature of the design is that the large number of weights required by a neural network are generated by circuitry tailored to each of their specific values, thus saving valuable cells. Whenever one of these values is required to change, the appropriate circuitry must be dynamically reconfigured. This may always be achieved in a fixed and minimum number of cells for a given bit stream resolution.

3-[1994] [5] Michael Gschwind, Valentina Salapura, Oliver Maischberger show how FPGA can be used to efficiently implement neural nets. By implementing the training phase in software and the actual application in hardware, conflicting demands can be met: training benefits from a fast edit-debug cycle, and once the design has stabilized, a hardware implementation results in

higher performance. They present a bit-serial encoding scheme and computation model, which allows space-efficient computation of the sum of weighted inputs, thereby facilitating the implementation of complex neural networks. Then they present an expandable digital architecture which allows fast and space-efficient computation of the sum of weighted inputs, providing an efficient implementation base for large neural networks[1994][6].

4-[1994] [6, 8, 9,10] James G. Eldredge and Brad L.Hutchings proved that the Run-Time Reconfiguration is a way of more fully exploiting the flexibility of reconfigurable FPGAs. The RRANN uses run-time reconfiguration to increase the hardware density of FPGAs. This is done by dividing the backpropagation algorithm into three sequentially executed stages and configuring the FPGAa to execute only one stage at a time. The FPGAs are reconfigured as part of normal execution in order to change stages. Using reconfigurability in this way increases the number of hardware neurons a single FPGA can implement by 500%.

5-[1994] [11] Stephen L. Bade and Brad L. Hutchings present an architecture that makes it feasible to implement large ANNs with FPGAs. The architecture combines stochastic computation techniques with a novel lookup-table-based architecture that fully exploits the lookup-table-structure of many FPGAs.

6-[1995] [12] M. van Daalen, T. Kosel, P. Jeavons, and J. Shawe-Taylor present the results of experimental work that demonstrates the generation of linear and sigmoid activation functions in a digital stochastic bit-stream neuron.These activation function are generated by a stochastic process and require no additional hardware, allowing the design as an individual neuron to be extremely compact.

7-[1996] [13] Michael Gschwind, Valentina Salapura, Oliver Maischberger present an extendable digital architecture for the implementation of a Hopfield NN using FPGAs. They exploit the reprogrammability of these devices to support on-chip learning.

8-[1997] [14] M. Rossmann, A. Buhlmeier, and G. Manteuffe, K. Goser present the implementation of the Hebbian learning rule in a hardware-friendly architecture based on a stochastic pulse representation of the signals. They compare implementation costs and speed of this approach with those of a parallel and a bit-serial implementation.

9-[1998] [15] Jean-Luc Beuchat, Jacques-Olivier Haenni and Eduardo Sanchez present the concept of reconfigurable systems using the example of a digital hardware implementation of NN, as well as RENCO, a platform very well-suited for the prototyping of such systems.

10-[2000] [16] Kathernie Compton and Scott Hauck give an overview of the hardware architectures of reconfigurable computing machines, and the software that targets these machines, such as compilation

tools. And they consider the issues involved in run-time reconfigurable system, which re-use the configurable hardware during program execution.

11-[2000] [17] Hector Fabio Restrepo, Ralph Hoffmann and Andres Perez-Uribe describe a networked FPGA-based implementation of the FAST (Flexible Adaptable-Size Topology) architecture. They used a network of Labomat 3 boards (a reconfigurable platform developed in this work laboratory).

12-[2003] [18] Kristian Robert Nichols created in his Master thesis a new FPGA-based ANN architecture, called RTR-MANN to demonstrate the performance enhancements gained from using current-generation tools and methologies. RTR-MANN was shown to have an order of magnitude with more scalability and functional density compared to order-generation FPGA-based ANN architectures.

13-[2004] [19] Dennis Duncan Earl, in his PhD. Thesis, genetic algorithms to evolve complex architecture ANN designs in FPGAs using reconfigurable hardware. His system presented as a powerful new tool for researchers working to develop both artificially intelligent systems and complex evolvable hardware.

14-[2005] [20]David Verstracten, Benjamin Schrauwen and Dirk Stroobant used analogue neurons to build an RC-system(Reservoir Computing) on FPGA, using stochastic neurons that communicates using stochastic bit streams instead of fixed-point values. This drastically simplifies the hardware implementation of arithmetic operations such as addition, the nonlinearity and multiplication.

15-[2006] [21] Roman Kohut, Bernd Steinbach and Domink Frohlich suggest a new approach for the modelling of Boolean NN on FPGAs using UML. In this work they decreased of the required number of configurable logic blocks(CLB) for the realization of Boolean neuron that is mapped directly to Look Up-Table(LUT) and configurable logic block(CLB) of FPGAs.

16-[2006] [22] Saumil G. Merchant, Gregory D. Peterson and Seong G. Kong designed an intrinsic embedded online evolution system using Block-based neural networks and implemented on Xilinx Virtex II Pro FPGAs. The designed network can dynamically adapt its structure and parameters to input data pattern variations without any FPGA reconfiguration overheads, overcoming a major bottleneck for online evolution systems.

2. Stochastic Arithmetic

The hardware implementations of many Artificial Neural Networks applications are normally space consuming due to the huge size of digital multipliers, adders, etc. Stochastic arithmetic provides a way to carry out complex computations with very simple hardware and very flexible design of the system. The stochastic implementation is also compatible with modern VLSI design and manufacturing technology. The fundamental

principles of the stochastic arithmetic are summarized as follows [11]:

- All inputs are transformed into the binary stochastic pulse streams. This is called randomization. In this step, the real number is coded into a sequence of binary bits where the information is contained in the probability of any given bit in the stream being logic '1'.
- The stochastic arithmetic that consists of digital gate circuits takes place of the normal arithmetic. After the first step, the real number becomes the one bus random stream and so the math operations including multiplication, addition, division, integration and many others can be implemented with simple digital circuits.
- The stochastic pulse streams are converted back to the normal numerical values. After these corresponding mathematical operations, the result is still the stochastic binary streams. This process is to extract the probability information from the streams and get the real number with the proper representation type. This step is called de randomization.

2-1 Stochastic Multiplication (Bit-Serial Multipliers)

Multiplication is the basic arithmetic operation and typically presents the advantage of the stochastic arithmetic. Figure-1a shows the traditional digital logic unsigned 4 x 4 multiplier which composes tens of logic gates, however, the stochastic unsigned multiplier shown in Figure-1b greatly reduces the number of the logic elements used for the calculation. Figure-1c presents an example of unsigned stochastic multiplication. The stochastic unsigned multiplier uses an AND gate as the arithmetic operator. With the one-bit stream represented inputs, the unsigned multiplication is as simple as just one AND gate. It is obvious that **(** P_x AND P_y **)** = P_x . P_y = X . Y. The digital implementation of signed multiplication is derived based on the following equations [8]:

$X=2P_x–1$ (2.1)

$Y=2P_y–1$ (2.2)

$P_{XNOR}= P_x . P_y + (1–P_x). (1– P_y) = 2 . P_xP_y – P_x – P_y + 1$ (2.3)

$X.Y = (2Px – 1). (2Py – 1) = 4. P_xP_y – 2P_x – 2Py + 1 = 2.(2 . P_xP_y – P_x – P_y + 1) – 1 = 2P_{XNOR} – 1$ (2.4)

(Eq.2.3) gives the signed representation of X, Y and X.Y while the output of a XNOR gate has the probability relation of (Eq.2.4) that derives the expression of the signed multiplication and so the signed multiplication could be performed using one XNOR gate.

2.2 Stochastic Addition (The Bit-Serial Adder)

The stochastic unsigned addition presents in Figure-2. It is based on a two-port selector and the selection signal is decided by a stream of random bits which has the equal probability to be '1' and '0' . The two inputs are all transformed into stochastic representations. The selector randomly chooses the output from the two input streams. Since the possibility of both ports is the same, so the result contains the probability information from both

two input ports which gives the addition performance. From Figure-2, if the input X has the probability of P_x and the probability of P_y, the output stream has the output probability of $(P_x+P_y)/2$ which is automatically normalized [11].

2.3 Stochastic Square.

The stochastic square operation presents in Figure-3. It is composes of a D Flip-Flop and an AND gate. The AND gate performs the stochastic unsigned multiplication and the two inputs are the input X and its sequence after the D Flip-Flop. Based on the random theory, the output of the D Flip-Flop has the same probability as the input sequence. However, it is independent of its input sequence, which is an important condition for the stochastic unsigned multiplication [11].

2.4 Stochastic Signed Subtraction (Bit-Serial Subtracted)

The stochastic signed subtraction presents in Figure-4. It is similar to the unsigned addition where a **NOT** gate is used to change the sign [11].

3. Design Re-Configurable ANN Circuitry

The design of re-configurable ANN circuit consists of the following five top-level modules:

- System Controller Module
- Random Pulse Generator Module
- Bit-Stream Generator Module
- Bit-Stream Converter Module
- Neuron Module

The overall layout and interconnection of the five top-level modules is shown in Figure-5.

3.1 System Controller Module

The System Controller Module is responsible for resetting and initializing all of the modules during power-up or reset and for controlling all timing signals. There is only one system controller module for any given design. The schematic of module is shown in Figure-6. 4 MHz (time period of clock signal = 0.25μs) is the system clock that serves as the global clock for the circuit. Upon power up, the system clock feeds an 8-bit counter whose output feeds three comparators used for system initialization. After 100 pulses of the system clock, one comparator sends an enable signal to the Random Pulse Generator module. After 200 pulses, one comparator sends an enable signal to the Bit-Stream Generator modules. After 250 pulses, one comparator sends an enable signal to the Bit-Stream Converter modules. This last comparator's signal is routed back to the 8-bit counter's enable pin allowing the comparator signals to remain static (until the reset pin to the module is triggered), the system-controller operation was tested using simulation tools as shown in Figure-6a .

3.2 Random Pulse Generator Module

RPG module is responsible for generating a random 4-bit address value to be used by the Neuron modules. This level of randomness is more than suitable for ANN designs. An overview of Random Pulse Generator module is shown in Figure-7. The clock input triggers an

LFSR_32 sub module and four bs_generator12 sub modules (because most multifunction data acquisition boards are of a 12 digital I/O pins).The LFSR_32 sub module is a 32-bit Maximal Length Linear Feedback Shift Register (LFSR) that generates a pseudo-random stream of pulses . The LFSR_32 sub-module (consisting mainly of D-flip-flops) is shown in Figure.8: .the probability of receiving a high output from the LFSR_32 is 0.5 (i.e. for a given clock cycle, there is an equal chance of receiving a high output versus a low output). In Figure-7, it is seen that the random output from LFSR is passed through to an output pin on the Random Pulse Generator module and it is also used internally to produce four unique random pulses via the bs_generator12 sub-module. The bs_generator12 sub module allows the 'firing' probability of the original LFSR output to be weighted. The details of the bs_generator12 sub module are shown in Figure-9. The waveform output from bs_generrator12 sub module (which is presented in RPG module) are shown in Figure-7a:.The bs_generator12 sub-module modifies the input firing probability using the stochastic arithmetic techniques .The circuitry at the heart of the sub-module is the modulator, shown in Figure-10.

The four bs_generator12 sub-modules, contained within the Random Pulse Generator module, allow the LFSR output to be modified such that it creates four outputs having a firing probability of 0.666178 (0.101010101000), 0.799511 (0.110011001010), 0.947008 (0.11110010011) and 0.994383 (0.111111101000).These outputs are assigned the labels Address-Bit-Stream[0], Address-Bit-Stream[1], Address-Bit-Stream[2] and Address-Bit-Stream[3] .The result is a randomly generated 4-bit Address value that can take on any value between 0 and 15.

3.3 Bits-Stream Generator Module

The Bit-Stream Generator module is responsible for taking 12-bit input values and converting them into stochastic bit-streams. The number of Bit-Stream Generator modules required in an ANN design depends on the number of inputs .The circuitry for the Bit-Stream Generator is very similar to the bs_generator12 sub-modules shown previously in Figure.9: .The only difference is that the input to the Bit-Stream Generator module is an external 12-bit pin on the FPGA chip, and these inputs are AND-ed with an ' Enable ' pin.

3.4 Neuron Module

The Neuron module is responsible for multiplying chosen inputs by static weight values, summing the results, and then producing a non-linear (pseudo-sigmoid) output. These are the fundamental operations of the neuron model. Because these operations are performed through stochastic arithmetic techniques, the circuitry for the Neuron module is actually quite small. 3-input Neuron shown in Figure-11: consists of three Look Up-Tables (labelled LUT0), some AND and OR gates, and two D-Flip-Flop. Each Look Up-Table has a 4-bit Address line (which is fed by the Random Pulse Generator module discussed previously), a 16-

bit weight value input, and a clock input. The 16-bit weight value fills the Look Up-Table with data to be accessed by the address lines resulting in a final output, q, which is a stochastic representation of a 16-bit weight value. Based on the weight value for the ANN design, these weight values are multiplied (stochastically) with the inputs (input as bit-streams from the Bit-Stream Generator modules) using AND gates. The result of each multiplication is summed and thresholdned using OR gates. The output is a bit-stream of date representing the neuron's final output. This output can then be fed to other neurons based on the connectivity in the ANN design.

3.5 Bits-Stream Converter Module

The Bit-Stream Converter module is responsible for converting bit-stream signals into 12-bit output signals. To do this, it is necessary to define a fixed bit-stream pulse length over which to calculate the average probability of firing. The circuitry of the Bit-Stream Converter module is shown in Figure-12. For an ANN design with N number of primary outputs, N numbers of Bit-Stream Converter modules are required.

4. Experimental Result and Implementation

4.1 Case Study 1 : AND- OR-EXOR Decomposition of the BNN.

The trained Boolean Neural Network represents an output signal $y_0, y_1, \ldots, y_8$. Each of which is defined on three variables x_1, x_2 and x_3 (Table 1 :). Let the transfer functions of hidden neurons be Boolean dependences, which are expressed as:

- Hidden neuron

$$Out^{[z]} = f^{[z]} (Inp_i \ . \ w_i^{[z]}) \quad (4.1)$$

$Out^{[z]}$ - output signal of the neuron with number z.

$f^{[z]}$ - transfer function of the neuron with number z.

[z] - index $z = 1,\ldots, z_N$.

z_N - number of neurons on the hidden layer,

$f^{[i]} \neq f^{[j]} : i \neq j \quad i, j \in [1, z_N]$

- Output neuron

$$Out^{[j]} = f_{i=1}^{z_n} (Inp_i \wedge w_{i[j]}) \quad (4.2)$$

$f \in \{ AND, OR, EXOR \}$

[j] - number of neuron on the output layer of the BNN

Table 1: The Input of the BNN

X_1	X_2	X_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

The weight coefficients obtained by training the net in MATLAB program are shown in Table 2: and Table 3:

Table 2: Weight Coefficients for Input, Hidden neurons

	W_1	W_2	W_3	W_4	W_5	W_6
AND	1	1	0	1	1	1
	0	1	1	1	1	1
	1	1	1	1	1	1
OR	1	1	1	1	1	1
	1	1	1	0	1	1
	0	1	1	1	1	1
EXOR	1	1	1	1	1	1
	1	1	1	1	0	1
	1	1	1	1	1	1

Table 3: Weight coefficients for Hidden, Output neurons

	w_1	w_2	w_3	w_4	w_5	w_6
AND	0	1	0	0	0	1
OR	0	1	1	0	0	1
EXOR	0	0	1	1	1	0

By training of BNN, the number of inputs for neurons was restricted to the number of inputs of the LUT of the FPGA .The structure of the BNN for a set of Boolean function (AND, OR, EXOR) in to common NN. The output waveform shown in Figure-13

4.2 Case Study 2: Frequency Recognition

The proposed network is tested with another experiment which is a frequency recognition problem for a square waveform of varying frequency (ranging from 1MHz, 0.5MHz, 0.25MHz, 0.1MHz, and 0.125MHz). Figure-14: shows the output waveform from the Frequency Recognition NN with three inputs and three Hidden Neurons and one output neuron with weight equal to [1 0 1] .

4.3 Reports

The reports can be summarized as shown in Table 4: as follows:

1- The hardware cost for the Neural network design is shown in Table 4:. It is clear that the hardware utilization for the proposed BNN is 100% of FPGA platform but for Frequency Recognition NN is 34%.
2- All modules which belong to the neural network design are fitted on the same FPGA platform Spartan-xl. The hardware utilization for BNN was 100% of the platform capacity. i.e. a larger capacity version is needed for this applications on the proposed Neural Network design such as Virtex to be fitted on the same FPGA platform (Virtex xcv400 hase a Number of Slices=4800, IOB=404, Block RAM=20).
   The software (Xilinx Foundation v4.1i) which was used, was not produced
3- No error was declared in all the reports with the analysis components belonging to Virtex but from the results of the reports it is noted that the proposed BNN design needs a larger number of units from Spartan-xl) ..
4- The worst case connection delay for the BNN network is 10.950ns, for Frequency Recognition NN is 1.303ns.

5. Conclusion

In This paper it is demonstrated that it is possible to construct a stochastic Re-configurable Artificial Neural Network based Look Up-Table that maps efficiently to FPGAs with minimal digital circuitry. Each of the FPGA Neural Network modules is verified to be functionally correct through numerous simulations. After the correct functionality of each module has been verified, the modules are connected to each other and the system is synthesized and implemented successfully using Xilinx Foundation 4.1i synthesis and implementation tools. It is shown that such a neuron of the proposed network produces an emergent activation function without any

additional hardware, of the predicted mathematical form. In most application of these neurons, it is envisaged that a sigmoid activation function would be the most useful, as this corresponds most closely to the standard neural network model. It exploits all of the advantages inherent to FPGAs: low cost, availability, re-configurability, the ability to support arbitrary network architectures, lower development costs, all while providing a level of density that makes it feasible to implement large ANNs with FPGAs.

References

[1] Jihan Zhu and Peter Sutton, " FPGA Implementations of Neural Networks-a Survey of a Decade of Progress, " In proceeding of 13th International Conference on Field Programmable Logic and Applications (FPL 2003), Lisbon, Sep 2003.

[2] Max van Daalen, Peter Jeavons, John Shawe-Taylor and Dave Cohen, " A Device for Generating Binary Sequences for Stochastic Computing, " Electronics Letters, Vol. 29, No. 1, pp. 80-81, Jan 1993.

[3] Max van Daalen, J. Zhao and John Shawe-Taylor, " Real Time Output Derivative for on Chip Learning Using Digital Stochastic Bit Stream Neurons, " Electronics Letters, Vol. 30, No. 21, pp. 1775-1777 , 1994.

[4] Max van Daalen, Peter Jeavons and John Shawe-Taylor, "A Stochastic Neural Architecture that Exploits Dynamically Reconfigurable FPGAs, " In Proceedings. IEEE workshop on FPGAs for custom computing machines, pp. 202-211, 1993.

[5] Michael Gachwind, Valentina Salapura and oliver Maischberger, " Space Efficient Neural Net Implementation, " Proc. Of the Second International ACM/SIGDA workshop on Field-Programmable Gate Arrays. Berkeley, CA, ACM, February 1994.

[6] Valentina Salapura, Michael Gachwind and oliver Maischberger, " A Fast Implementation of a General Purpose Neuron, " Lecture Notes in Computer Science 849, Springer Verlag, Berlin, 1994.

[7] J. G. Eldredge and B. L. Hutchings, "Density Enhancement of a Neural Network Using FPGAs and Run-Time Reconfiguration, "In Proceedings. IEEE workshop on FPGAs for custom computing machines, pp. 180-188, 1994.

[8] James G. Eldredge and Brad L. Hutchings, " RRANN: The Run-Time Reconfiguration Artificial Neural Network, " presented at IEEE Custom Integrated Circuits Conference, San Diego, CA, pp. 77-80, May 1-4, 1994.

[9] James G. Eldredge, " FPGA Density Enhancement of a Neural Network through Run-Time Reconfiguration, " MS.c. Thesis, Brigham Young University, May 1994.

[10] J. D. Hadley and B. L. Hutchings, " Designing a Partially Reconfigured System, " in Proceedings. IEEE workshop on FPGAs for custom computing machines, 1995.

[11] Stephen L. Bade and Brad L. Hutchings, " FPGA-Based Stochastic Neural Networks-Implementation, " in Proceedings. IEEE workshop on FPGAs for Custom Computing machines, pp. 189-198, 1994.

[12] Max van Daalen, Peter Jeavons, T. Kosel and John Shawe-Taylor, " Emergent Activation Functions from a Stochastic Bit-Stream Neuron, " Electronics Letters, Vol. 30, No. 4, pp. 331-333 , Feb 1994.

[13] Michael Gachwind, Valentina Salapura and oliver Maischberger, " A Generic Block for Hopfield Neural Networks with On-Chip Learning, " IEEE International Symposium on Circuits and Systems, Atlanta, GA, May 1996.

[14] M. Rossmann, A. Buhimeier, G. Manteuffe and K. Goser, " Short- and Long-Term Dynamics in a Stochastic Pulse Stream Neuron Implemented in FPGA, " In Artificial Neural Networks: 6th international conference; proceedings (ICANN 96), Germany, 1997.

[15] Jean-Luc Beuchat, Jacques-Oliver Haenni and Eduardo Sanchez, " Hardware Reconfigurable Neural Networks, " IPPS/SPDP workshops, pp 91-98, 1998.

[16] Katherine Compton and Scott Hauck, " An Introduction to Reconfigurable Computing, " IEEE Computer, April, 2000.

[17] Hector Fabio Restrepo, Ralph Hoffmann, Andres Perez-Uribe, Christof Teuscher and Eduard Sanchez, " A Networked FPGA-Based Hardware Implementation of a Neural Network Application, " Logic System Laboratory, Swiss Federal Institute of Technology, Lausanne, Switzerland, 2000.

[18] Kristian Robert Nichols, " A Reconfigurable Computing Architecture for Implementing Artificial Neural Networks on FPGA, " MS.c. Thesis, University of Guelph, 2004.

[19]Dennis Duncan Earl, " Development of an FPGA-Based hardware evaluation system for use with GA-Designed Artificial Neural Networks, " Ph.D. Thesis, University of Tennessee, Knoxville, May 2004.

[20] David Verstraeten, Benjamin Schrauwen and Dirk Stroobandt, " Reservoir Computing with Stochastic Bit stream Neurons, " Available at http://citeseer.com,2005.

[21] Roman Kohut, Bernd Steinbach and Dominik Frohlich, " FPGA Implementation of Boolean Neural Networks Using UML, " Proceeding of the 5th International workshops on Boolean Problems, BP 2006-FPGA-BNN, 2006.

[22] Saumil G. Merchant, Gregory D. Peterson and Seong G. Kong, " Intrinsic Embedded Hardware Evolution of Block-Based Neural Networks, " In ECE Dept, MS, Knoxville: University of Tennessee, 2006.

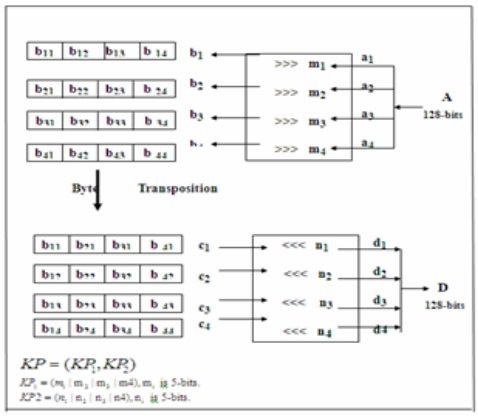


Fig. (4) P-function. $D = P(A, KP_1 | KP_2)$ for the Improved Algorithm

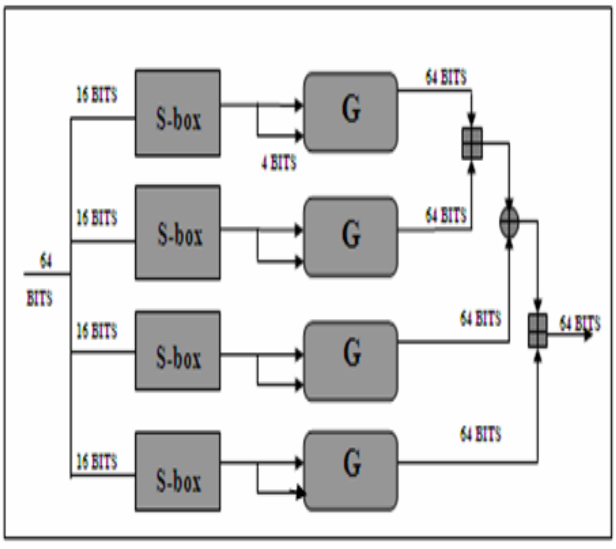


Fig.(5) Function F of the proposal Algorithm

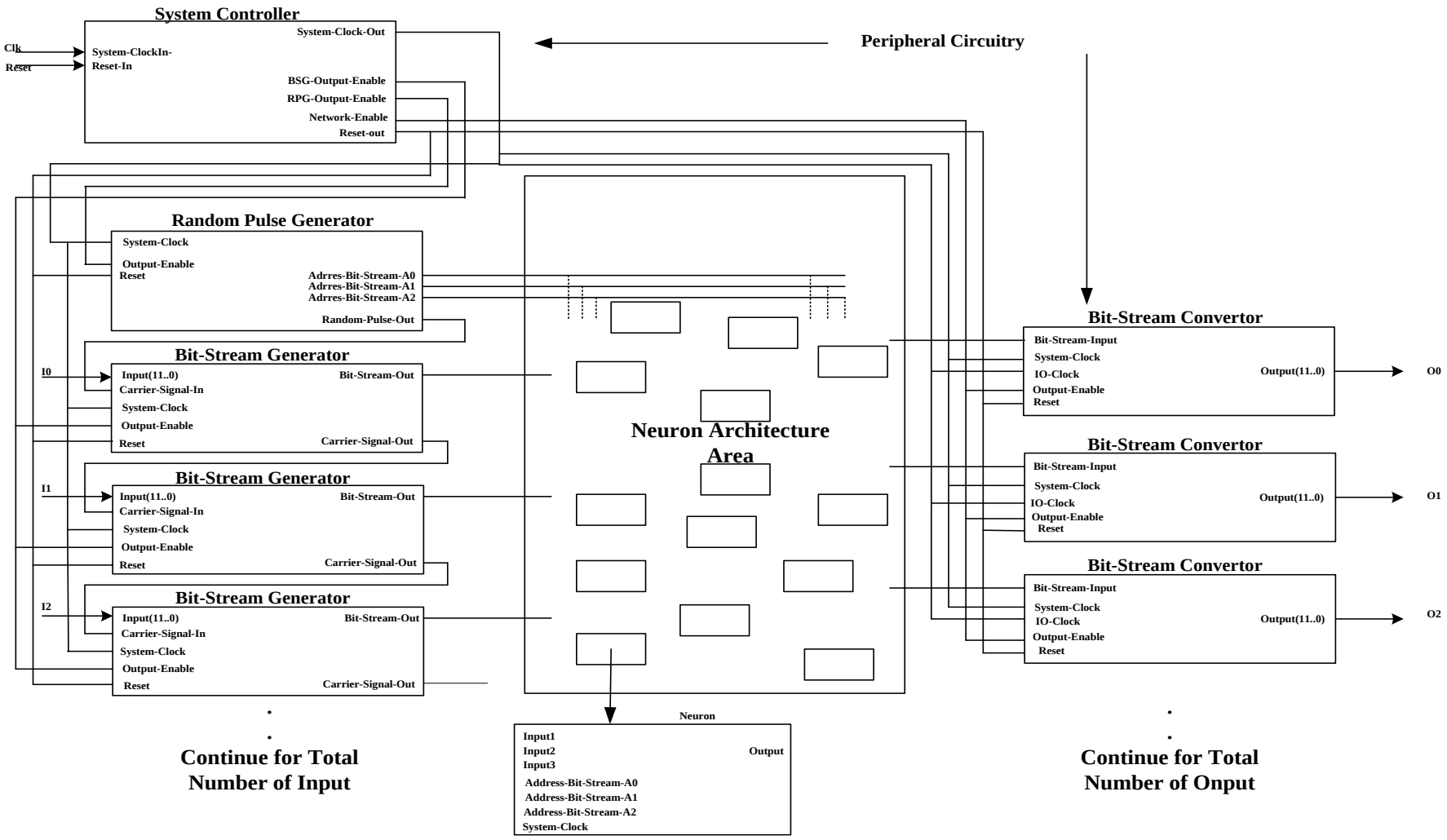
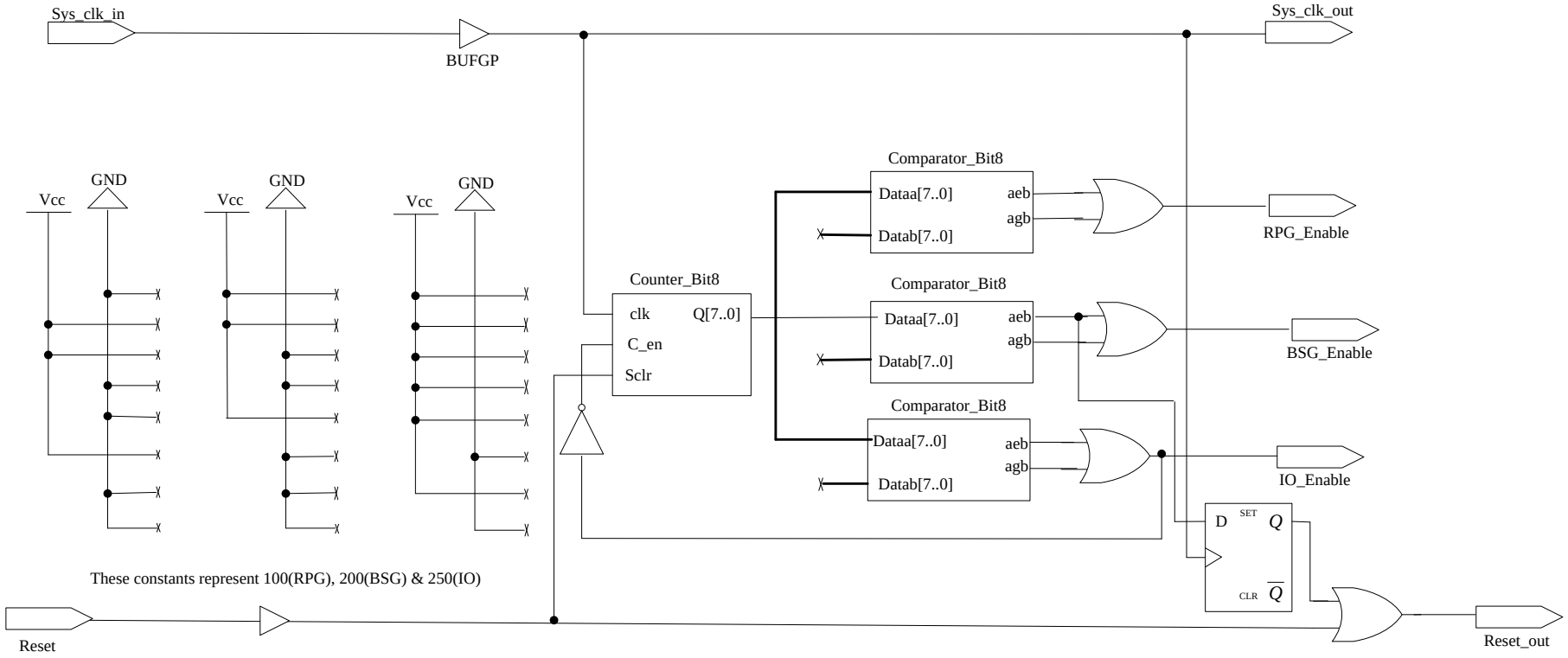


Fig-5 A Complet View of the Network



.

Fig-6 Schematic of System Controller Module

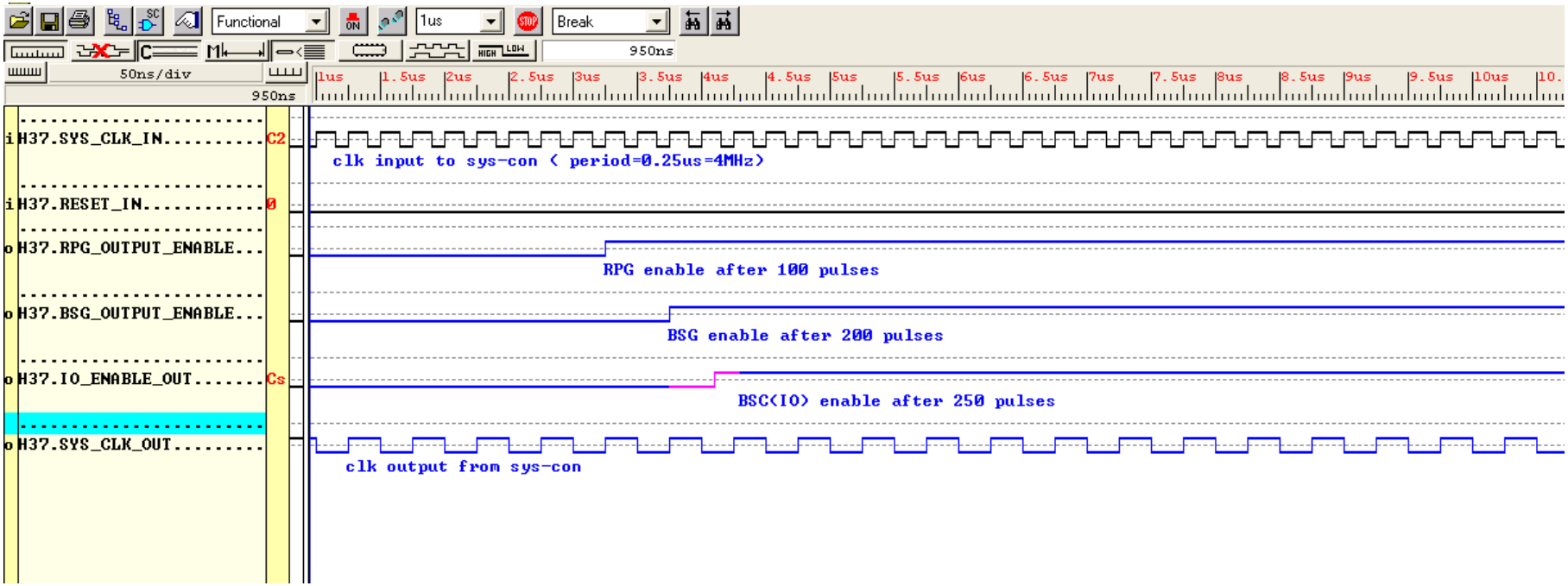
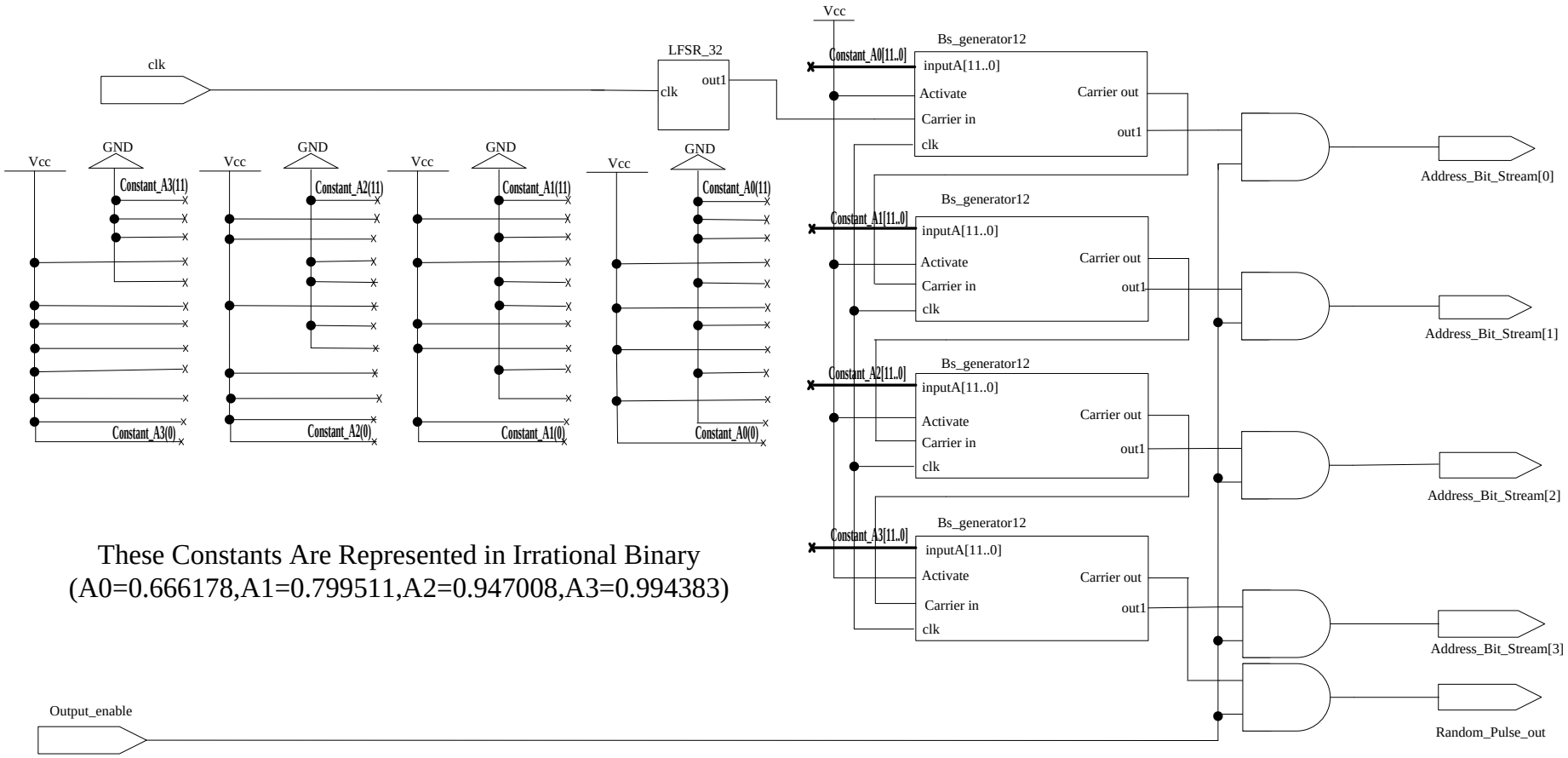


Fig-6a the waveform output from system controller Module



These Constants Are Represented in Irrational Binary
(A0=0.666178,A1=0.799511,A2=0.947008,A3=0.994383)

Fig-7 Schematic of Random Pulse Generator Module

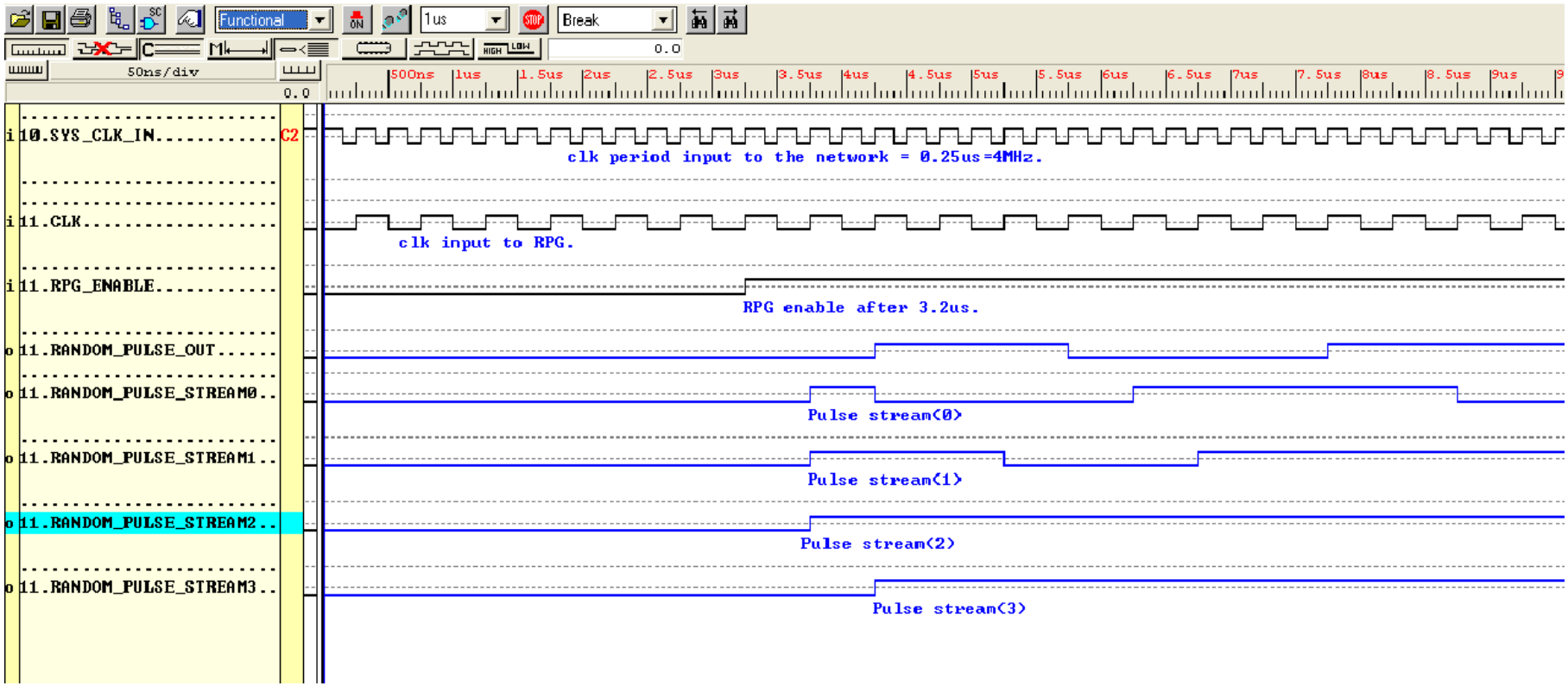


Fig-7a the waveform output from RPG Module (bs-generator sub module)

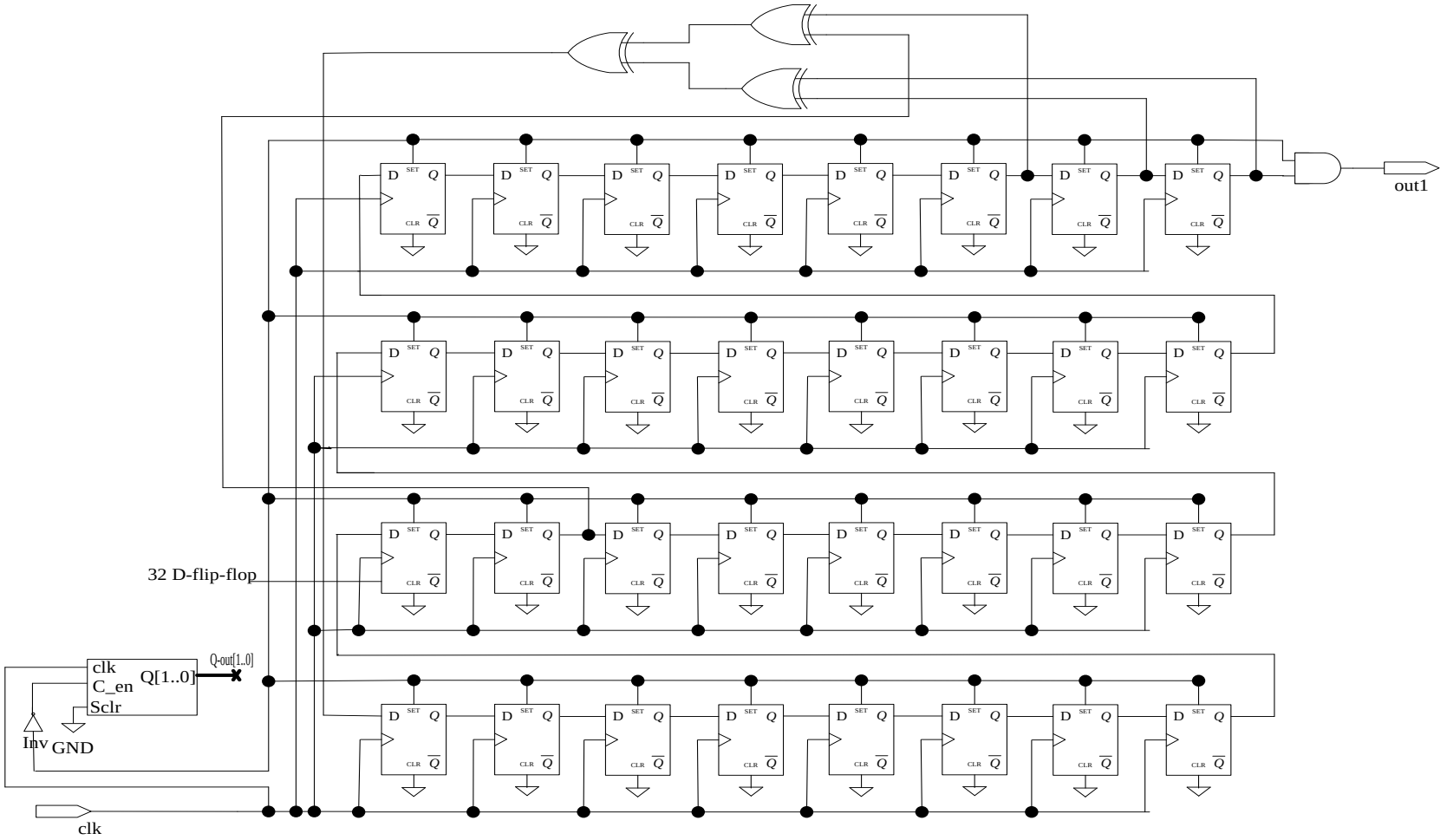


Fig-8 Schematic of LFSR_32 Sub-Module

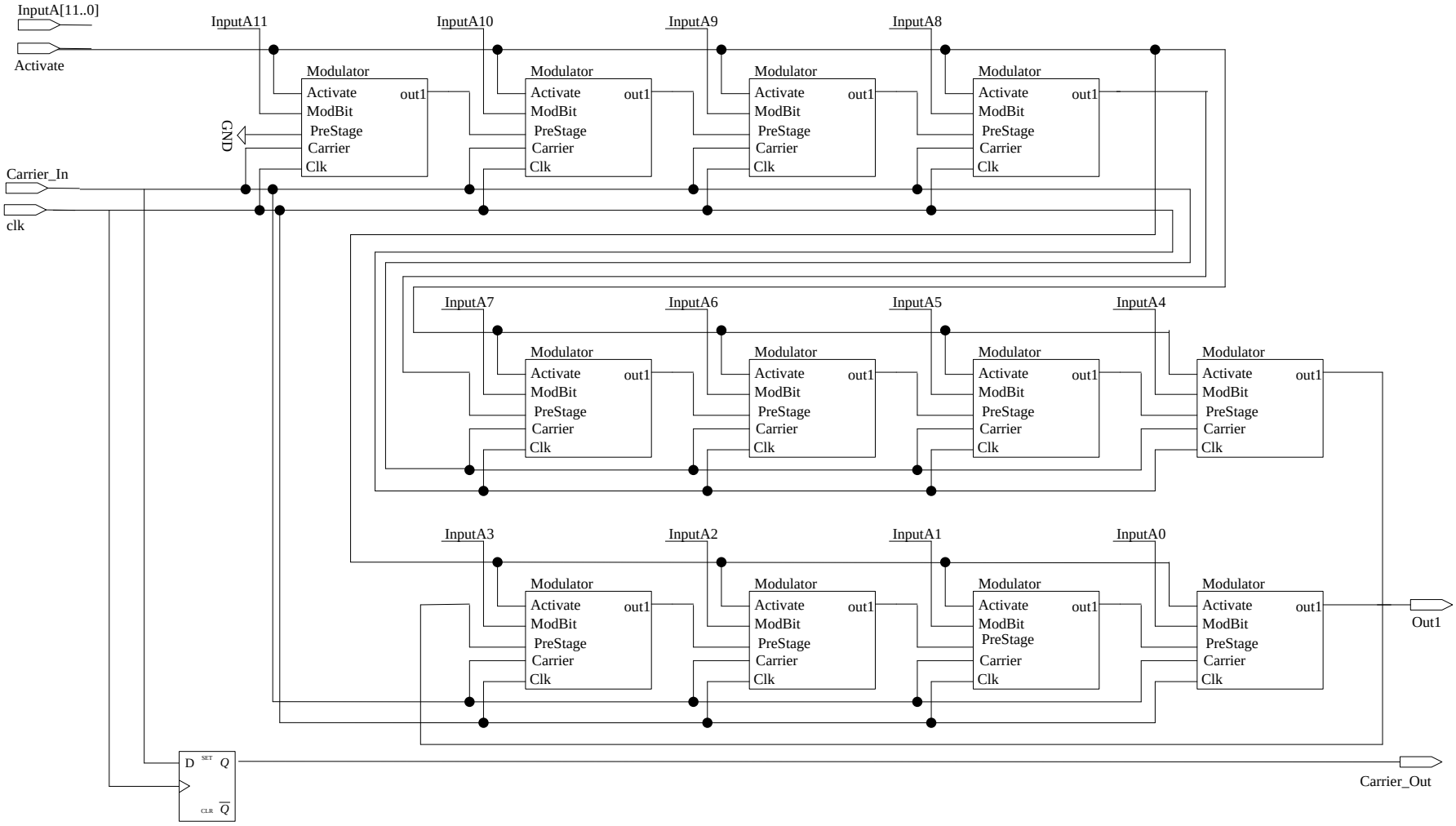


Fig-9 Schematic of Bit Stream-Generator Sub-Module

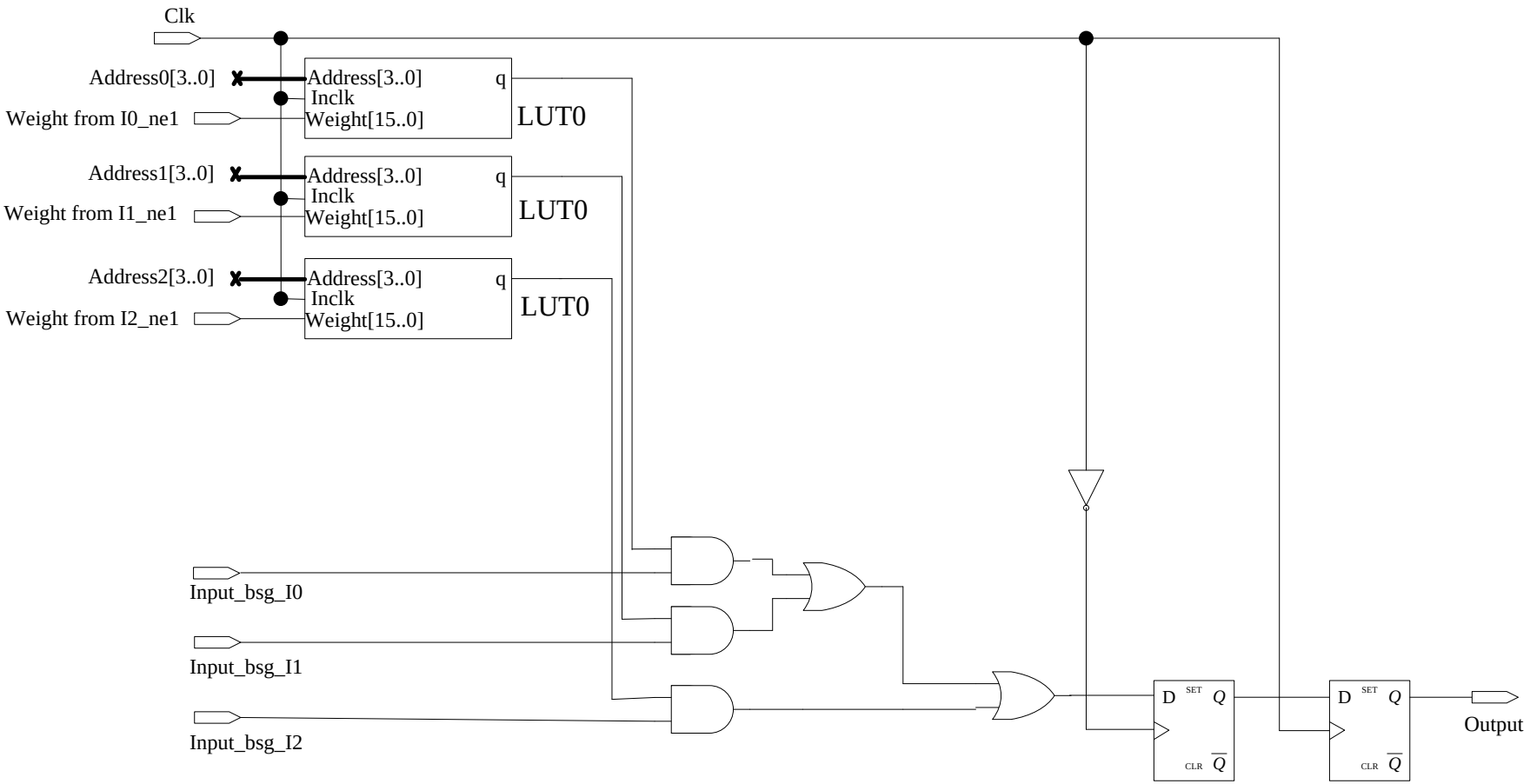


Fig-11 Schematic of Neuron Module (with 3 input)

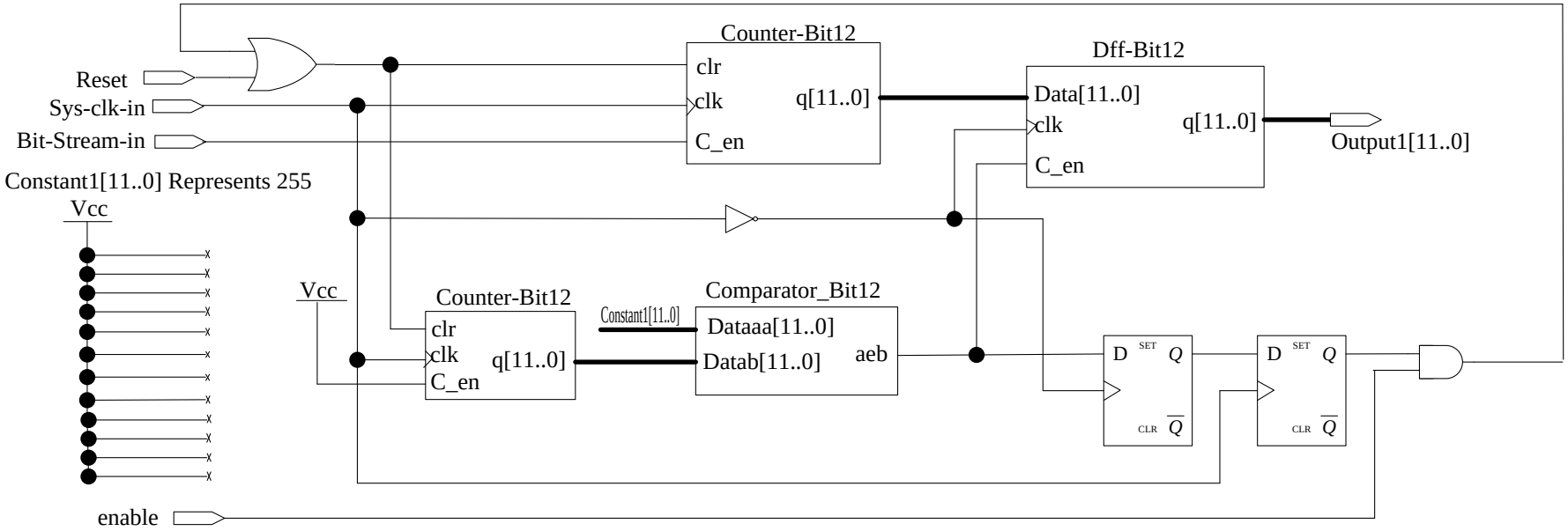


Fig-12 Schematic of Bit Stream-Converter Module

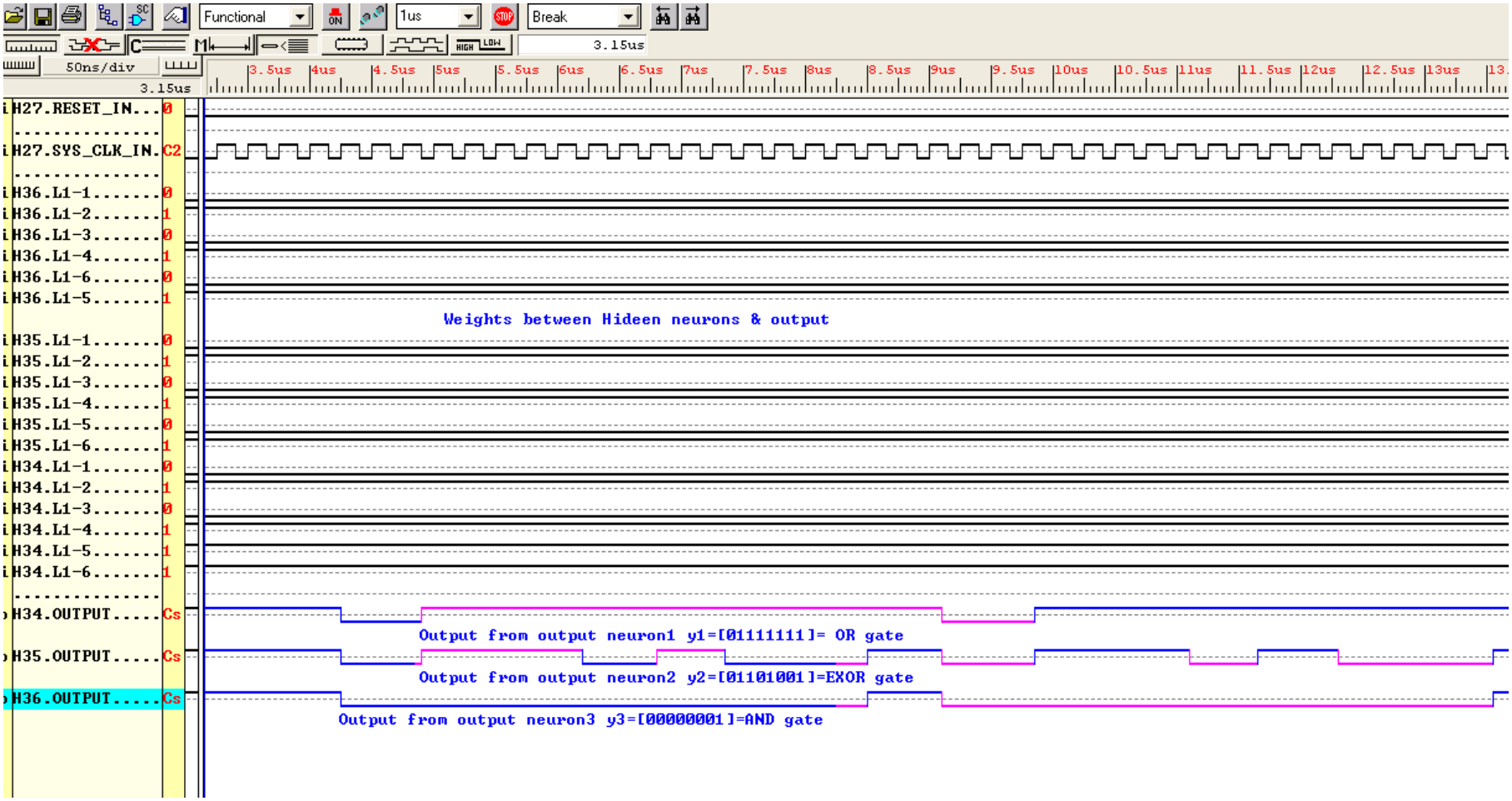


Fig-13 the waveform of BNN for a set of Boolean Function

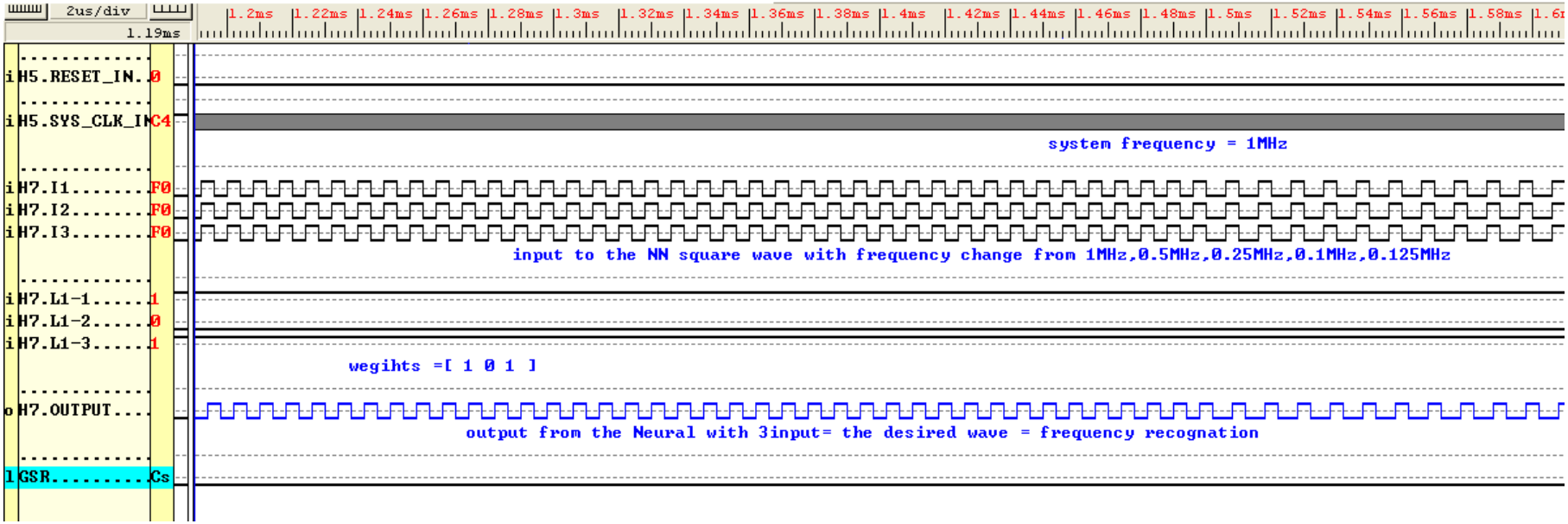


Fig-14 the waveform out from Frequency Recognition NN

Table 4: Hardware Utilization of the Boolean NN, Frequency Recognition NN

	CLB(100)	CLB F.F(200)	4INPUT LUT(200)	3INPUT LUT(100)	16X1 RAM	I/O BLOCKS(61)	GATES COUNT	ADDITIONAL GATES
BNN	100(100%)	147(73%)	175(88%)	11(11%)	1	30	2610	1488
F. Rec.	34(34%)	38(19%)	40(20%)	6(6%)	0	16	776	768