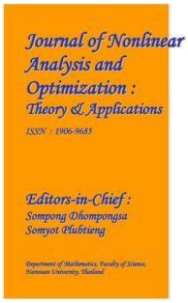
Journal of **N**onlinear **A**nalysis and **O**ptimization
Vol. 15, Issue. 1 : 2024
ISSN : **1906-9685**



VERILOG BASED BRAIN - INSPIRED SYSTEM

Swathi Kambhampati, K.Varsha, A.Chaitanya, B.Madhavi, G.Lakshmi Yoga Vardhini,
Department of Electronics & Communication Engineering, NRI Institute of Technology,
Pothavarappadu (V), Agiripalli (M), Eluru (Dt)-521212

Abstract
Computers, as man-made machines, operate based on predefined sets of inputs and algorithms to produce outputs, capable of executing intricate tasks. However, they lack the innate imaginative capacity and comprehensive understanding inherent in the human brain. The human brain, a remarkable biological machine, learns and comprehends through a complex interplay of cognition, experience, and sensory input. Achieving computer cognition akin to human thought necessitates modeling machines on the workings of the human brain. At the core of this endeavor lies the emulation of neurons, the fundamental units of the brain responsible for cognitive processes and intelligent behavior. With billions of neurons interconnected through intricate neural networks, the brain communicates via electrical impulses, facilitating cognition and consciousness. Replicating this neural architecture within computers enables the emulation of human-like cognitive processes, potentially unlocking greater understanding, creativity, and problem-solving capabilities in artificial intelligence systems. However, while significant strides have been made in neural network-based AI models, fully mirroring the complexity and adaptability of the human brain remains an ongoing challenge in the pursuit of true artificial intelligence.
Keywords: Neuron, layer, neural column, activation function

1.Introduction

The Blue Brain Project, initiated in May 2005 by the Brain and Mind Institute of the École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland, endeavors to fabricate a synthetic brain through the reverse-engineering of mammalian brains. Utilizing the Blue Gene supercomputer and Michael Hine's NEURON software, the project aims to explore the structural and functional principles of the brain, employing biologically realistic neuron models rather than merely analyzing artificial neural networks. Its ultimate ambition is to digitally reconstruct and simulate the human brain. The project's innovative research approach capitalizes on interdependencies within experimental data to generate detailed brain maps across various organizational levels (molecules, cells, micro-circuits, brain regions, and the entire brain) without requiring exhaustive measurements. Meanwhile, the Fast Analog Computing with Emergent Transient States (FACETS) project, established in September 2005 and funded by the European Union, involves approximately 80 scientists from Austria, France, Germany, Hungary, Sweden, Switzerland, and the United Kingdom. This initiative aims to investigate the computational properties of the human brain. Additionally, it seeks to develop microchip hardware capable of emulating around 200,000 neurons with 50 million synapses on a single silicon wafer. Current prototypes exhibit computational speeds approximately 100,000 times faster than their biological counterparts, positioning them as the swiftest analog computing devices ever devised for

neuronal computations.Neurogrid, a supercomputer, employs analog computation to mimic ionchannel activity and utilizes digital communication to establish synaptic connections. With the capacity to simulate one million neurons, each with two subcellular compartments, Neurogrid's design is informed by neurophysiological studies. It successfully replicates nonlinear interactions between projections terminating in distinct cortical layers using a pyramidal-cell model with just two compartments. Notably, Neurogrid's chief advantage lies in its real-time simulation of a million neurons connected by billions of synapses, rivaling supercomputers while consuming 100,000 times less energy. Additionally, its integration of analog and digital systems enhances computational capabilities.Furthermore, this paper introduces an evolvable hardware system housed within an FPGA, capable of autonomously generating digital processing circuits on an array of processing elements (PEs). Employing an embedded evolutionary algorithm, candidate circuits are dynamically reconfigured and evaluated in the final hardware, leveraging a systolic approach for optimal performance. The system showcases smaller reconfiguration times compared to software-based approaches, even when considering hardware evaluation in the target device. Focusing on digital image filtering and edge detection, the system produces evolved filters that outperform classic linear and nonlinear filters, exhibiting superior adaptability to various noise types and intensities without compromising filtering quality, even at high noise levels (40 percent). This advancement represents a significant step towards fully autonomous, adaptive systems.In summary, this collection of selected contributions fills a void in FPGA-based neurocomputing, serving as both an introductory guide and a comprehensive reference. While the book may have minor graphical and typographical issues, along with occasional instances of poor English and lack of an index, it offers valuable insights into the field, benefiting researchers, engineers, and students alike. Nonetheless, the Evolvable Hardware research area, while showing promising progress, still has room for improvement in efficiently tackling more complex problem instances, as discussed in this paper [1-7].

2. Proposed Method

The model described in this paper uses three types of neurons

Input Neuron

In the initial layer of the column, the neuron acts as a foundational unit designed to replicate specific characteristics of biological neurons. This neuron possesses the ability to process incoming signals in pulse form. It receives two pulse inputs, which are then tallied and transmitted after a designated delay. This delay is essential for mimicking the temporal dynamics inherent in neural processing and is achieved through a latch mechanism. Furthermore, the neuron integrates two distinct types of thresholds to regulate its function. The first threshold, referred to as the synaptic gap threshold, is established using logical AND operations on the input pulses. This threshold mechanism ensures that the neuron responds only when a certain level of input activity is attained, mirroring the synaptic strength observed in biological synapses.

The neuron also uses a threshold for the activation function, which is modelled as a comparator. This threshold establishes if the total input activity exceeds a predetermined threshold value, activating the neuron. The neuron efficiently analyses incoming information and adds to the overall functionality of the brain-inspired system by integrating these threshold mechanisms.

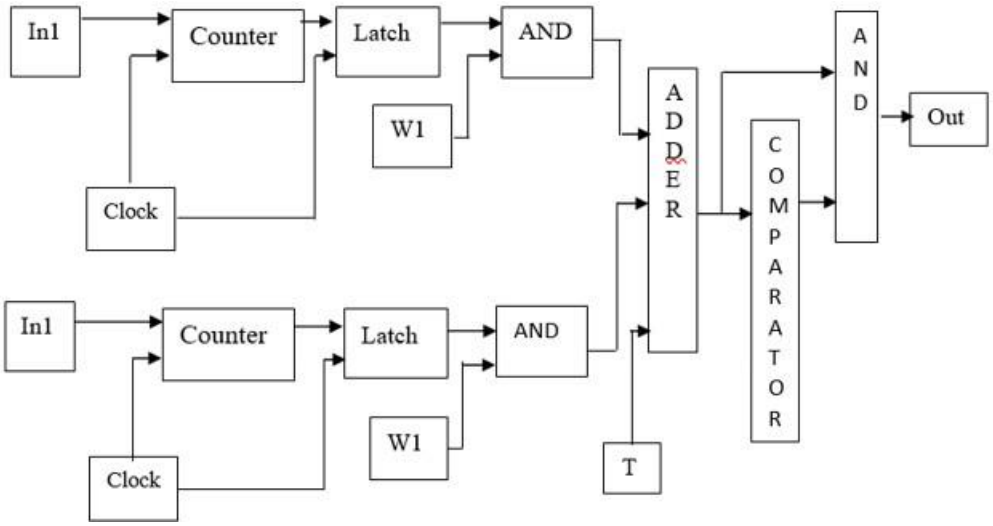


Figure.1. Input Neuron Model

Middle Neuron

In layers 2 to 5 of the column, neurons primarily act as conduits for transferring information from lower layers to upper layers, facilitating hierarchical processing within the brain-inspired system. Unlike the neurons in the input layer, these neurons do not engage in counting or utilize latch mechanisms, as their role is solely focused on information transmission. However, similar to the neurons in the input layer, they still rely on threshold mechanisms to regulate their activation. These thresholds are implemented in two forms, resembling the synaptic gap threshold and the activation function threshold observed in the input layer. The synaptic gap threshold, achieved through logical AND operations on input signals, ensures that the neuron responds only when a sufficient level of input activity is detected.

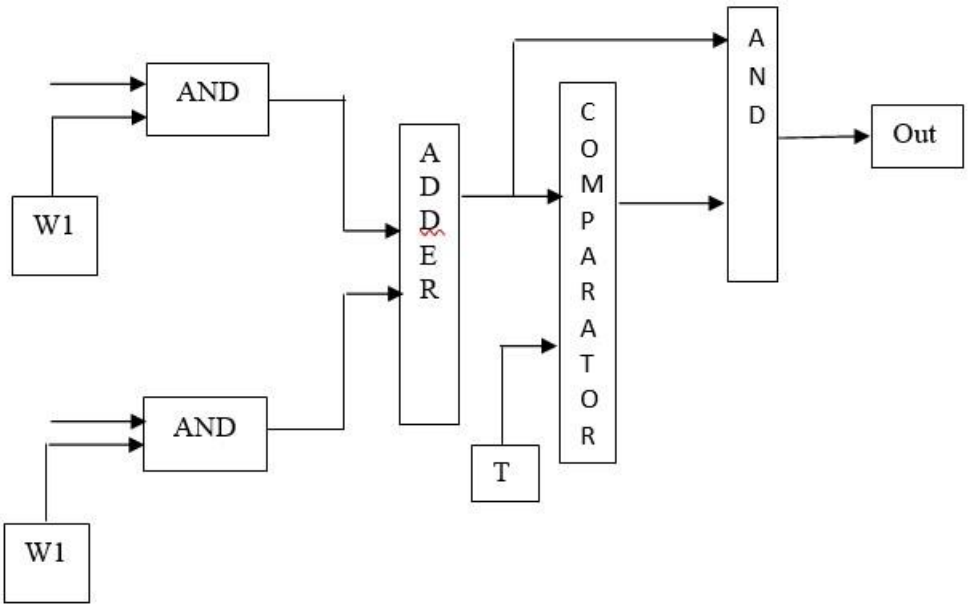


Figure.2. Middle Neuron Model

This mechanism mirrors the selective connectivity observed in biological synapses, where signal transmission depends on the strength of synaptic connections. Furthermore, these neurons include an activation function threshold, typically represented by a comparator, to assess whether the accumulated input activity exceeds a predetermined threshold level, thus initiating the neuron's activation. Through the integration of these threshold mechanisms, neurons in layers 2 to 5 efficiently facilitate information flow across hierarchical layers, enhancing the overall functionality and adaptability of the braininspired system.

Pulsar Neuron:

In layer 6 of the column, neurons operate on a unique principle of negative potential, where the nerve potential is elevated in the absence of information. This unconventional approach entails that neurons are primed to fire when there is minimal input activity, contrasting with the typical response pattern

observed in neural networks. Essentially, the high potential state signifies a readiness to transmit signals, with activation occurring in response to deviations from this baseline. This distinctive characteristic enables neurons in layer 6 to efficiently detect and amplify signals representing novel or unexpected information, thereby contributing to the system's ability to discern salient stimuli and adapt to changing environmental conditions.

NEURAL COLUMN MODEL

The single-column structure comprises six layers: input neurons at the bottom, middle neurons spanning layers 2 to 5, and pulsars at the topmost layer. The distinctive aspect of this architecture is the interconnection between layer 4 neurons, where outputs from one layer 4 neuron act as inputs to the next layer 4 neuron, facilitating connectivity across various columns.

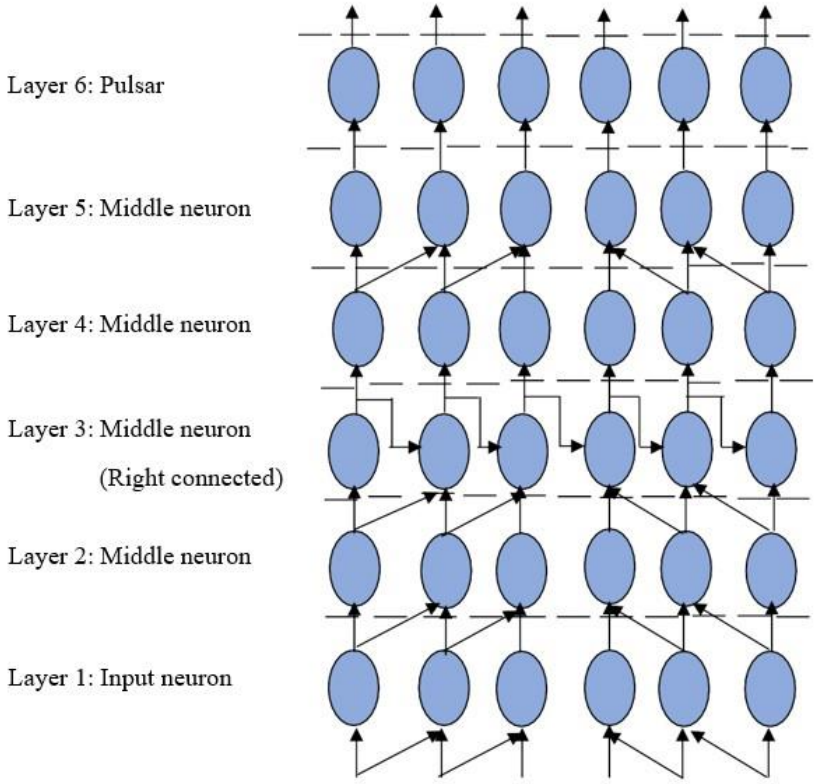


Figure.3. Pipelined Single Neural Column Model

Pipelining

Pipelining stands as a fundamental technique in computer architecture and digital circuit design, aimed at boosting system performance by harnessing parallelism. By segmenting the processing of a task into sequential stages, pipelining enables simultaneous operation of different stages on various parts of the task. This approach mirrors an assembly line in a factory, where each stage in the pipeline performs a specific operation on the data it receives before passing it to the next stage. This concurrent execution of tasks enhances throughput and reduces execution time by allowing multiple operations to progress simultaneously through the pipeline, rather than waiting for each operation to complete sequentially. Essentially, a pipeline embodies a structured flow of data through a series of stages, with each stage dedicated to a specific processing aspect. This division of labor streamlines task execution by eliminating idle time and maximizing resource utilization. Through seamless operation overlap, pipelining optimizes system efficiency and responsiveness, empowering modern computing systems to handle increasingly complex tasks with greater speed and agility. The concept of pipelining finds extensive use in various computing architectures, including CPUs (Central Processing Units), GPUs (Graphics Processing Units), and digital signal processors, among others, particularly effective in scenarios where tasks can be divided into smaller, independent steps executed sequentially.

Various pipelining techniques are employed across computing systems, each tailored to optimize specific aspects of task execution. Instruction pipelining, prevalent in CPUs, segments the instruction

execution process into discrete stages such as fetching, decoding, and executing. By distributing these tasks across different stages, instruction pipelining enables simultaneous processing of multiple instructions, thereby enhancing throughput and overall system performance. Similarly, data pipelining, commonly found in digital circuits, divides data processing into stages like arithmetic operations, memory access, and output generation, facilitating parallel execution of data-related tasks, further boosting system efficiency and responsiveness.

Despite the manifold benefits of pipelining, its implementation introduces challenges that must be addressed to ensure correct operation. One such challenge is pipeline hazards, which can disrupt the orderly flow of instructions or operations through the pipeline. Data hazards occur when subsequent instructions rely on the results of preceding instructions that have not yet completed, while control hazards arise from branching instructions that alter the flow of execution. Additionally, structural hazards may occur due to resource conflicts when multiple pipeline stages compete for the same hardware resources. Effectively managing these hazards is crucial to mitigating their impact on system performance and ensuring the accurate execution of instructions or operations in pipelined architecture. The following block diagram represents the combined as well as the sub-operations performed in each segment of the pipeline.

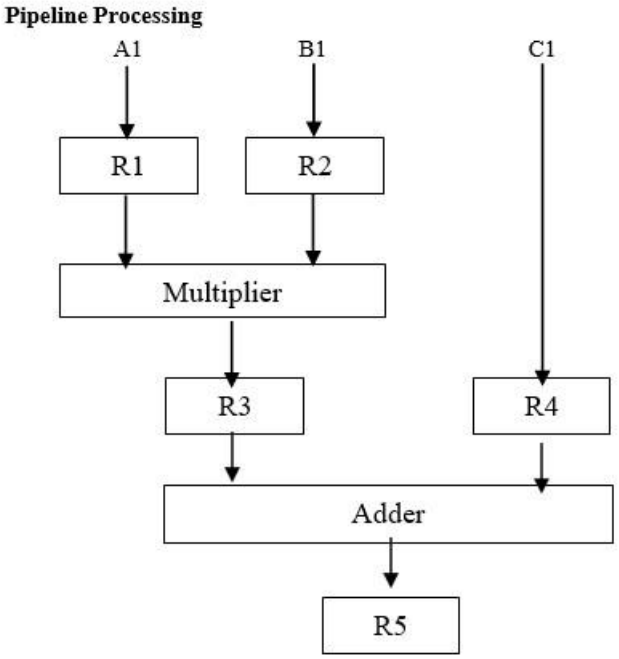


Figure.4. Sub-operations performed in each segment of the pipeline.

Registers R1, R2, R3, and R4 serve as temporary data storage, while combinational circuits within a specific segment process this data without internal state or memory. These circuits conduct operations solely based on their input values, rendering them suitable for tasks such as arithmetic calculations or logical operations within the designated segment. The output generated by the combinational circuit in a given segment is applied as an input register for the next segment. For example, from the block diagram, we observe that register R3 is utilized as one of the input registers for the combinational adder circuit. Pipeline processing in the instruction stream involves overlapping the execution of multiple instructions, enabling enhanced throughput and efficiency by fetching, decoding, and executing instructions concurrently, akin to how data pipeline processing enhances data throughput sequentially. Most digital computers with complex instructions necessitate instruction pipelines to carry out operations like instruction fetch, decode, and execute.

In general, the computer needs to process each instruction with the following sequence of steps.

1. Fetch instruction from memory.
2. Decode the instruction.
3. Calculate the effective address.
4. Fetch the operands from memory.
5. Execute the instruction.

6. Store the result in the proper place.

The concept of an instruction pipeline plays a pivotal role in maximizing the efficiency of instruction execution. Each step of instruction processing, including fetching, decoding, executing, and storing results, is typically segmented within the pipeline. However, variations in task complexity or external factors like memory access latency may cause these segments to operate at different speeds. When segments process incoming information at different rates, pipeline inefficiencies may arise. For instance, if one segment completes its task faster than another, it may encounter idle time while waiting for the slower segment to finish. Additionally, situations may occur where multiple segments simultaneously require memory access, leading to contention and potential delays as one segment waits for another to finish accessing memory before proceeding. To mitigate these issues and optimize pipeline performance, organizing the pipeline into segments of equal duration is desirable. This ensures that each segment completes its operation within a consistent timeframe, reducing bottlenecks and idle periods. One commonly used approach is the Four-segment instruction pipeline, dividing the instruction cycle into four equal-duration segments: instruction fetch, instruction decode, execute, and write-back. By dividing the instruction cycle into equally sized segments, the Four-segment pipeline promotes a balanced distribution of workload across pipeline stages. This organization helps maintain a steady flow of instructions through the pipeline, minimizing idle time and maximizing throughput. Moreover, it simplifies coordination of memory access and other resource dependencies among pipeline segments, further enhancing overall efficiency. Segmenting the instruction cycle into equally sized segments, as demonstrated by the Four-segment pipeline, is a fundamental strategy for optimizing pipeline efficiency in computer architectures. This approach fosters balanced workload distribution, reduces contention for shared resources, and ultimately enhances instruction processing performance within the pipeline.

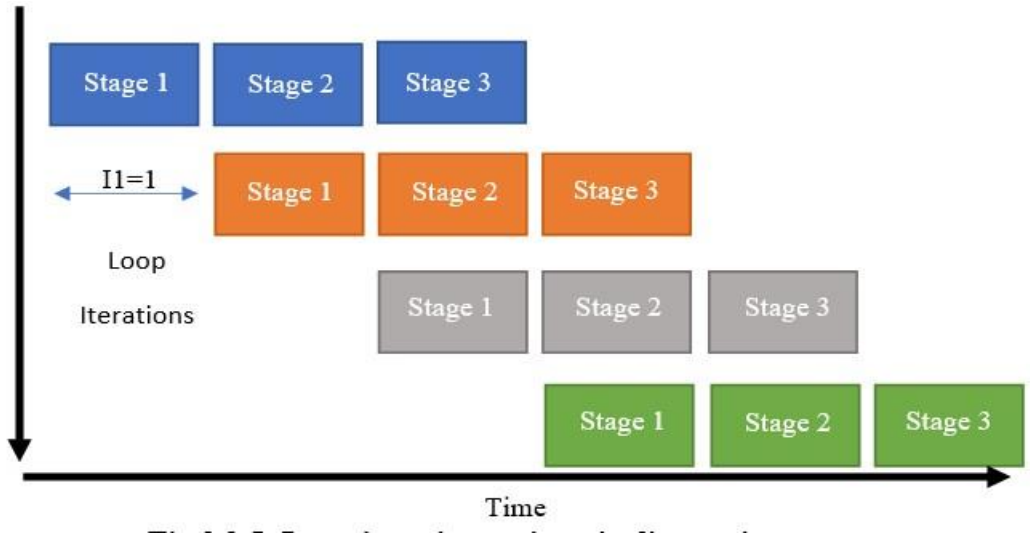


Figure.5. Loop iterations using pipeline registers

In a four-segment instruction pipeline, the goal is to streamline instruction processing stages to enhance efficiency and throughput by consolidating two or more distinct segments into a single one. This consolidation reduces the overall number of pipeline stages and potentially minimizes the overhead associated with instruction processing.

For example, in traditional instruction pipelines, the decoding of an instruction and the calculation of its effective address are typically separate stages. The decoding stage interprets the opcode and determines the operation to be performed, while the effective address calculation stage computes the memory address or operand location based on addressing modes specified in the instruction.

By merging these two stages into a single segment, the pipeline can potentially decrease the number of clock cycles required for instruction execution. This consolidation enables the processor to initiate the effective address calculation as soon as the instruction opcode is decoded, rather than waiting for a separate stage to perform the calculation. Consequently, the overall latency of instruction execution may be reduced, resulting in improved performance.

However, consolidating segments also presents trade-offs. Combining decoding and effective address calculation may increase the complexity of the combined segment, potentially making it more

challenging to design and optimize. Moreover, if the tasks within the combined segment have significantly different processing requirements or resource dependencies, achieving optimal performance without introducing bottlenecks may be challenging.
Overall, while consolidating segments in a four-segment instruction pipeline can streamline instruction processing and potentially enhance efficiency, careful consideration of the trade-offs involved is necessary to ensure that performance gains are achieved without introducing undue complexity or compromising resource utilization.

3. Results and Discussion RTL
schematic
The single neuron column model, an artificial human brain model, employs a blend of full adders, counters, latches, and comparators to facilitate information transmission between neurons. This model has been meticulously designed, implemented, and subjected to simulation and synthesis using the Xilinx ISE tool.

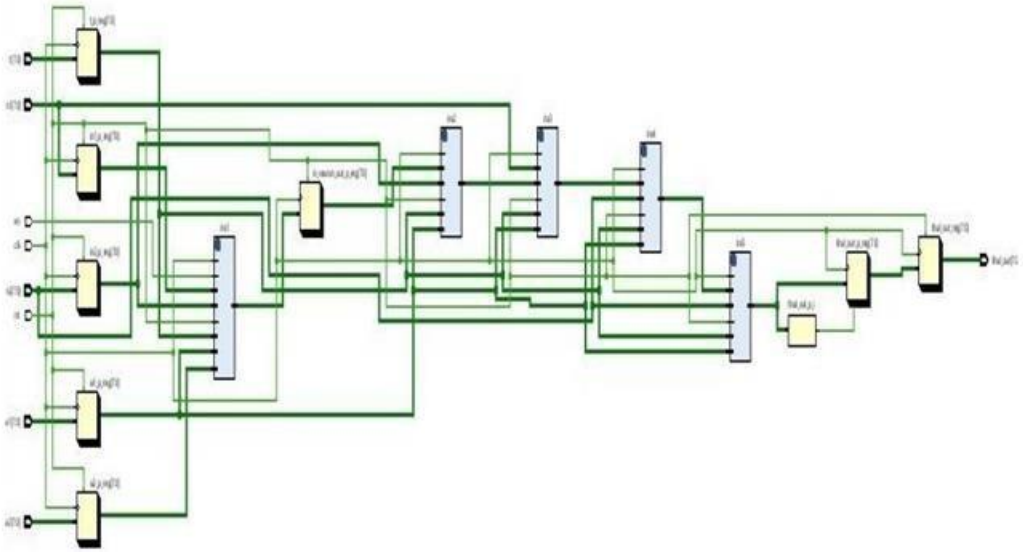


Figure.6. RTL Schematic of single neuron column model
Figure 6 depicts the RTL Schematic of a single neuron column model. This figure describes the schematic of a single neuron column model in the proposed system. It illustrates how information is passed from input neurons to the final output neuron, i.e., the pulsar layer.
Simulation results:
Taking some values for the simulation as below, Unsigned decimal inputs:
int1=18, int2=116, w1=133, w2=99, T=0,255 Unsigned decimal outputs:
For T=0:
final out=0, middle neuron1=0, middle neuron2=0, middle neuron3=0, middle neuron4=0.
For T=255: final out=136, middle neuron1=136, middle neuron2=128, middle neuron3=132, middle neuron4=136.

Figure.7. Simulation result of single neuron column model

The behavioural simulation of the single neuron column model is shown in figure 7 above. We can see that the single neuron column model's enable pins in this simulation are clk, rst, and en. The value of T, which acts as the threshold, determines the outputs. The output changes depending on this threshold. Depending on the threshold value, information is transferred from the input neuron to the final output neuron.

Technology schematic:

Figure 8 illustrates the technological schematic of the single neuron column model, showcasing a sophisticated framework crafted to mimic the functionality of the human brain. This depiction integrates components such as full adders, counters, latches, and comparators to enable seamless information transmission between neurons, closely resembling biological neural networks.

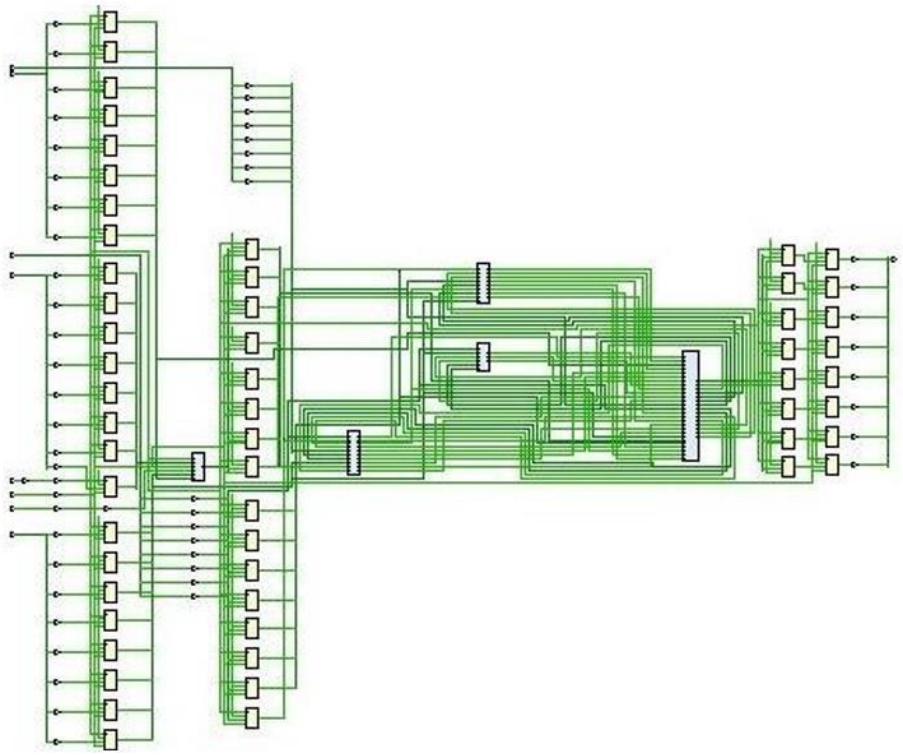


Figure.8. Technological schematic of single neuron column model Area:

Name	Slice LUTs (134600)	Slice Registers (269200)	Bonded IOB (400)	BUFGCTRL (32)
single_neuron_column_model	127	72	51	2
ins1 (input_neuron)	31	16	0	0
ins2 (middle_neuron)	80	0	0	0
ins3 (middle_neuron_0)	6	0	0	0
ins4 (middle_neuron_1)	2	0	0	0
ins5 (middle_neuron_2)	8	0	0	0

Figure.9. Synthesis table for single neuron column model

The synthesis table presented in Figure 9 provides an overview of the area required to construct the single neuron column model. This figure offers insights into the number of Look Up Tables (LUTs) and registers necessary for the model.

Delay:

Figure 10 showcases the synthesis report presenting the delay, indicating that the time delay of the single neuron column model in simulation is 15.988ns.

```
Max Delay Paths
--------------------------------------------------------------------------------
Slack:                  inf
  Source:               in2_p_reg[1]/C
                          (rising edge-triggered cell FDRE)
  Destination:          final_out_p_reg[0]/S
  Path Group:           (none)
  Path Type:            Max at Slow Process Corner
  Data Path Delay:      15.988ns  (logic 6.079ns (38.022%)  route 9.909ns (61.978%))
  Logic Levels:         32  (CARRY4=8 FDRE=1 LUT4=8 LUT5=5 LUT6=10)
```

Figure.10. Synthesis report displaying delay

Power:

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**2.812 W**
Design Power Budget:	**Not Specified**
Process:	typical

Figure.11. Power estimation of single neuron column model

The single neuron column model's power estimation is shown in figure 11, above. This figure helps to comprehend the model's overall efficiency and possible energy-saving optimisations by showing the electrical requirements for operating the model. For the single neuron column model, the total on-chip power is 2.812 W. **Comparison table**

Table 1: Comparison table

	Area(LUT'S)	**Delay(ns)**	**Power (watts)**
Proposed method	127	15.988	2.812
Existing method	101	17.347	3.762

A comparison of the current approach and the suggested approach—the single neuron column model—is shown in table 1 above. This table shows that when comparing the suggested way to the current method, the area increased by 26 LUTs. On the other hand, the suggested approach demonstrated a 0.95 W reduction in power usage and a 1.359 ns delay reduction.

3.1 ADVANTAGES AND APPLICATIONS

3.1.1 Advantages

- Parallel Processing: Different phases of brain activities can be carried out in parallel thanks to pipelines. Replicating the brain's parallel processing architecture can improve compute speed and efficiency.
- Performance Enhancement: Compared to conventional sequential approaches, the pipelined model may be able to mimic brain processes more quickly. This might make it easier to analyse and interpret computations that resemble the brain more quickly.
- Resource Utilisation: By enabling several components of the brain model to function concurrently, pipelines maximise the use of hardware resources. This economical use of resources could result in the implementation of hardware that is more effective.
- Scalability: Pipelining frequently makes scaling simpler. Research and application development may be made more flexible if a model of this kind could be appropriately scaled down for certain uses or scaled up to represent more complicated brain activities.

3.1.2 Applications

- Neuromorphic Computing: The development of neuromorphic computing systems may benefit from the use of a pipelined electronic brain model. Real-time cognitive processing, pattern recognition, and effective learning are made possible by these systems, which imitate the structure and operations of the brain.
- Brain-Computer Interfaces (BCIs): A model like this could help create more sophisticated BCIs that enable real-time communication between the brain and electronics. This has the potential to revolutionise direct brain-to-machine communication and assistive technology for people with disabilities.
- Medical Applications: Research and treatment in the field of medicine may benefit from real-time simulations. For example, predicting brain reactions to stimuli or designing treatments for neurological illnesses could be aided by modelling neural responses in certain brain regions.
- Cognitive Robotics: By using pipelined brain models, robotics researchers may be able to build more intelligent and adaptable robots. These machines could mimic human thought processes by processing sensory data, learning from mistakes, and making judgements instantly.
- Machine learning and artificial intelligence: Using models resembling brains could improve algorithms for machine learning and AI. AI systems' capacity for pattern recognition, decision-making, and adaptive learning may be improved by real-time simulations.
- Cutting-Edge Research Instruments: These models may be used as cutting-edge neuroscience research instruments. Studying neural networks, comprehending brain processes, and investigating emergent characteristics of intricate brain functions may all be facilitated by real-time simulations.

3.1.3 Dis-Advantages

- Limited Complexity: Humans' capacity to perform complicated tasks efficiently is hampered by this model's inability to capture the entire complexity of human cognition.
- Scalability Challenge: It is challenging to scale up this model for larger datasets or more complex jobs, which could result in computing bottlenecks.
- Dependency on Training Data: The amount and quality of training data have a significant impact on performance, which may introduce biases and restrict generalisation. ss

4. Conclusion

Generally speaking, research has demonstrated that basic pipelining-based artificial brain models can replicate human cognitive functions by utilising sequential stages and parallel processing. Similar to human cognition, these models demonstrate potential in tasks like pattern recognition and picture processing. They efficiently analyse information by mimicking the neuronal architecture of the brain,

which provides inspiration for the development of increasingly sophisticated and flexible AI systems. This breakthrough paves the way for artificial systems to accomplish a wide range of activities and get close to the complexity of the human brain. The use of pipelining techniques to create artificial brain models has enormous potential. Further development of their design and algorithms could greatly improve performance and adaptability, and their incorporation into practical uses like robotics and healthcare could completely transform a number of industries. New avenues may be opened by interdisciplinary approaches that combine cognitive psychology, computer science, and neuroscience. Building trust will need addressing social and ethical issues like privacy and openness. In the end, these models have enormous potential to spur innovation in a variety of fields and produce intelligent systems that can solve difficult problems and improve human experiences.

Reference

1. H. Markram, “The blue brain project” Nature Reviews Neuroscience, 7(2), (pp.153- 160), 2006.
2. J.R. Cary, J. Candy, J. Cobb, R.H. Cohen, T. Epperly, D.J. Estep, S. Krasheninnikov, A.D. Malony, D.C. McCune, L. McInnes and A. Pankin, “Concurrent, parallel, multiphysics coupling in the FACETS project”. InJournal of Physics: Conference Series IOP Publishing (Vol. 180, No. 1, p. 012056), 2009. 3. T. Theocharides, G. Link, N. Vijaykrishnan, M. Irwin and V. Srikantam, “A generic reconfigurable neural network architecture implemented as a network on chip” In Proceedings of IEEE International SOC Conference (pp. 191-194) Sept., 2004.
4. D.B. Carr and S.R. Sesack, “ A Bag containing neurons in the rat ventral tegmental area project to the prefrontal cortex”. Synapse,38(2)(pp.114-123), 2000
5. R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo and L. Sekanina, “Self- reconfigurable evolvable hardware system for adaptive image processing”. Computers, IEEE Transactions on, 62(8), (pp.1481-1493), 2013.
6. C. Teuscher, "FPGA Implementations of Neural Networks” (Ormondi. A.R. and Rajapakse, J.C., Eds.; 2016)," in IEEE Transactions on Neural Networks, vol. 18, no. 5, pp. 1550-1550, Sept. 2017, doi: 10.1109/TNN.2007.906886.
7. F. Cancare, S. Bhandari, D.B. Bartolini, M. Carminati and M.D. Santambrogio, “A bird's eye view of FPGA-based Evolvable Hardware” Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on (pp. 169-175) June 2011.