|  |
| --- |
| HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY  **SCHOOL OF ELECTRICAL**  **AND ELECTRONIC ENGINEERING**  logo_128  PROJECT  **DIGITAL DESIGN II**  **TOPIC:**  **NEURON GRID BASED ON**  **SPIKING NEURAL NETWORK**  **Instructor**: Assoc. Prof. Dr. NGUYEN DUC MINH  **Students**: NGUYEN LE TRUNG 20186076  VU HOANG LONG 20182926  PHAM HUY HOANG 20182544  Ha Noi, 02-2022 |

**PREFACE**

This project report is submitted in fulfillment of the requirements for the course Digital Design II of School of Electrical Engineering. We are required to make a project on “Neuron grid based on spiking neural network”. The basic objective behind carrying out this project requirement is to understand and apply SNN theory to practical design.

In this report, we included various concepts, measurement step and simulation results on Modelsim and Vivado.

Through this progress of doing this project, we come to enhance the teamwork and problem-solving skill.

We are grateful to outstanding course instructor Nguyen Duc Minh. He was instrumental in preparing our class essential guidance and references to be capable of completing this assignment and has been helpful throughout. It's an honor to have completed this course under his direction, which aided in bringing out the best of our efforts in order to achieve our objective.

**TABLE OF CONTENTS**

[ABBREVIATIONS 1](#_Toc95862529)

[FIGURES AND DIAGRAMS 2](#_Toc95862530)

[TABLES 3](#_Toc95862531)

[ABSTRACT 4](#_Toc95862532)

[CHAPTER 1. INTRODUCTION 5](#_Toc95862533)

[1.1 Overview of Spiking Neuron Network 5](#_Toc95862534)

[1.2 Overview of Reconfigurable Architecture for Neuromorphic Computing 6](#_Toc95862535)

[CHAPTER 2. SPECIFICATIONS 8](#_Toc95862536)

[2.1 TOP module 10](#_Toc95862537)

[2.2 Interface Signals 10](#_Toc95862538)

[2.3 Function descriptions 11](#_Toc95862539)

[2.4 Algorithm and ASMD 12](#_Toc95862540)

[CHAPTER 3. IMPLEMENTATION 16](#_Toc95862541)

[3.1 Software 16](#_Toc95862542)

[3.2 Hardware 16](#_Toc95862543)

[3.2.1 ASMD code 17](#_Toc95862544)

[3.2.2 Testbench 20](#_Toc95862545)

[CHAPTER 4. RESULT 23](#_Toc95862546)

[CONCLUSION 24](#_Toc95862547)

[REFERENCES 25](#_Toc95862548)

# ABBREVIATIONS

|  |  |
| --- | --- |
| **SNN**  **RANC**  **FSM**  **ASM**  **ASMD** | Spiking neural network  Reconfigurable Architecture for Neuromorphic  Computing  Finite state machine  Algorithmic state machine  Algorithmic state machine with datapath |

# FIGURES AND DIAGRAMS

[*Figure 1 A natural Neuron* 5](#_Toc95862549)

[*Figure 2 Potential of a Neuron through time* 6](#_Toc95862550)

[*Figure 3 High level architectural overview of RANC components* 6](#_Toc95862551)

[*Figure 4 Our Neuron Grid* 8](#_Toc95862552)

[*Figure 5 Calculation process in Neuron Grid* 8](#_Toc95862553)

[*Figure 6 TOP module of Neuron Grid* 10](#_Toc95862554)

[*Figure 7 Timing diagram of Neuron Grid* 11](#_Toc95862555)

[*Figure 8 Controlling and feedback signals* 12](#_Toc95862556)

[*Figure 9 Flow chart of Neuron Grid (part 1)* 12](#_Toc95862557)

[*Figure 10 Flow chart of Neuron Grid (part 2)* 13](#_Toc95862558)

[*Figure 11 ASM diagram of Neuron Grid* 14](#_Toc95862559)

[*Figure 12 TOP module of Neuron Block* 15](#_Toc95862560)

[*Figure 13 Architecture of Neuron Block* 15](#_Toc95862561)

# TABLES

[*Table 1 Synthesis results table* 23](#_Toc95862562)

# ABSTRACT

In this project, a Neuron Grid is proposed using ASMD. The Neuron Grid will be designed and simulated on the ModelSim and synthesized on Vivado. This project introduces the involved theory and demonstrates the process of designing a Neuron Grid from making the flowchart, ASM, implementing on C++, Verilog and verifying the design. By which is divided in to 6 chapters explain detailly the design process :

**CHAPTER 1. INTRODUCTION**

*An Introduction SNN and RANC*

**CHAPTER 2. SPECIFICATIONS**

*Neuron Grid information through specifications*

**CHAPTER 3. IMPLEMENTATION**

*Neuron Grid implementation on C++ and Verilog*

**CHAPTER 4. RESULTS**

*Results after synthesizing*

# INTRODUCTION

## Overview of Spiking Neuron Network

Spiking neural network (SNN) is an artificial neural network that closely mimic natural neural network in the human brain which includes many Neurons connected to each other.

Diagram

Description automatically generated

*Figure 1 A natural Neuron*

Some important parts of a natural Neuron:

* Dendrite: a branch of a Neuron.
* Axon: a cable where electrical impulses from a Neuron travel to other neurons.
* Synapse: a point of contact between an Axon and a Dendrite.

Neurons in SNN transmit information only when a membrane potential – an intrinsic quality of the neuron related to its membrane electrical charge – reaches a specific value, called the threshold. When the membrane potential reaches the threshold, the neuron fires, and generates a signal (called Spike) that travels to other Neurons which, in turn, increase or decrease their potentials in response to this signal:

A picture containing chart

Description automatically generated

*Figure 2 Potential of a Neuron through time*

## Overview of Reconfigurable Architecture for Neuromorphic Computing

Reconfigurable Architecture for Neuromorphic Computing (RANC) is an open source software ad hardware ecosystem that seeks to make neuromorphic architectures widely accessible to researchers and application developers through a cohesive programming and testing environment.

RANC uses a multicore architecture with each Core mimicking a part of natural neural network based on SNN. Spikes generated from Neurons in each Core are transmitted to other Core:

Diagram, schematic

Description automatically generated

*Figure 3 High level architectural overview of RANC components*

* Five main parts of a Core in RANC:
* Neuron Block: primary computational component for the architecture.
* Core controller: a finite state machine (FSM) operating with a datapath (neuron block).
* Core SRAM: main memory of the architecture.
* Packet Router: component that is responsible for inter-core communication.
* Scheduler: final stop for a spike packet that is traversing the cores through the routing network.

# SPECIFICATIONS

Based on RANC, we redesign its Neuron Block and Core Controller and combine them into an architecture called Neuron Grid by using Algorithmic State Machine (ASMD).

A picture containing graphical user interface

Description automatically generated

*Figure 4 Our Neuron Grid*

We represent an architecture from a matrix N(a) x N(n) (with N(a) = 256 Axons and N(n) = 256 Neurons) in *Figure 4*.

Chart, diagram, box and whisker chart

Description automatically generated

*Figure 5 Calculation process in Neuron Grid*

*Figure 5* shows that output is generated every time a Neuron is processed with all 256 Axons. Each Neuron is processed successively. Potential is cumulated when there is a Spike and a Synaptic connection.

Equation of integrated potential in a Neuron:

*where: is the set of all axons that connect to the current neuron*

*indicates whether or not there is an incoming spike at j-th axon*

*is the integrated potential of the current neuron*

*i (t) is the potential of neuron i at time t*

*is the neuron instruction corresponding to j-th axon*

*is the -th kind of weight of axon j*

*li is the leak potential of i-th neuron*

*si indicates whether or not there is an outgoing spike at i-th neuron*

*is the positive threshold*

*is the negative threshold*

*is the positive reset potential*

*is the negative reset potential*

## TOP module

Diagram

Description automatically generated with medium confidence

*Figure 6 TOP module of Neuron Grid*

## Interface Signals

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal name** | **Width** | **I/O** | **Description** |
| clk | 1 | Input | DTI Clock Signal |
| reset\_n | 1 | Input | DTI Asynchronous Reset, active LOW |
| tick | 1 | Input | start working, active HIGH |
| neuron\_instruction | 2 | Input | refers to the weight types of each neuron |
| neuron\_parameter | 368 | Input | parameters of a neuron stored in RAM |
| axon\_spikes | 256 | Input | spikes information of the entire network. Bit “1” indicates that there is a spike. |
| axon\_num | 8 | Output | address where to take neuron\_instruction from storage |
| neuron\_num | 8 | Output | address where to take neuron\_parameter from RAM |
| potential\_out | 9 | Output | neuron potential after processed |
| spike\_out | 1 | Output | indicate the moment that there is a spike |
| update\_potential | 1 | Output | indicate the RAM to update potential\_out into neuron\_parameter |
| done | 1 | Output | indicate that the process finishes |
| sheduler\_set | 1 | Output | request axon\_spikes from the storage |
| scheduler\_clr | 1 | Output | clear the information after taken |
| error | 1 | Output | indicate there is error |

## Function descriptions

368-bit **neuron\_parameters** includes 256 bits synapse connections, 9 bits current potential, 9 bits reset potential, 36 bits weight (9 x 4), 9 bits leak value, 9 bits positive threshold, 9 bits negative threshold, 1 bit reset mode, 9 bits dx value, 9 bits dy value, 8 bits axon destination, 4 bits deliver tick offset. Not all the parameters are used in our Neuron Grid because they are based on RANC so some are for other blocks.

Asynchronous reset, active Low.

When **tick** is High, the network starts working. At the first positive edge of clock after **tick** is High, **scheduler\_set** is High to request **axon\_spike**s from the storage.

At the next negative edge of clock, **neuron\_num** and **axon\_num** is initialized and both are triggered by negative edge of clock. **axon\_num** increases after 1 cycle of clock and **neuron\_num** increases when **axon\_num** is 255.

At the first positive edge of clock when **axon\_num** is 255, **spike\_out** is triggered. **Potential\_out** is updated at the positive edge of clock when **axon\_num** changes. **Update\_potential** is High at positive edge of clock when **axon\_num** reaches 255.

**Scheduler\_clr** and **done** is High at the first positive edge of clock after **scheduler\_clr** is High. **Error** is High at the positive edge of clock when **tick** is High during the proccess.

Diagram

Description automatically generated with low confidence

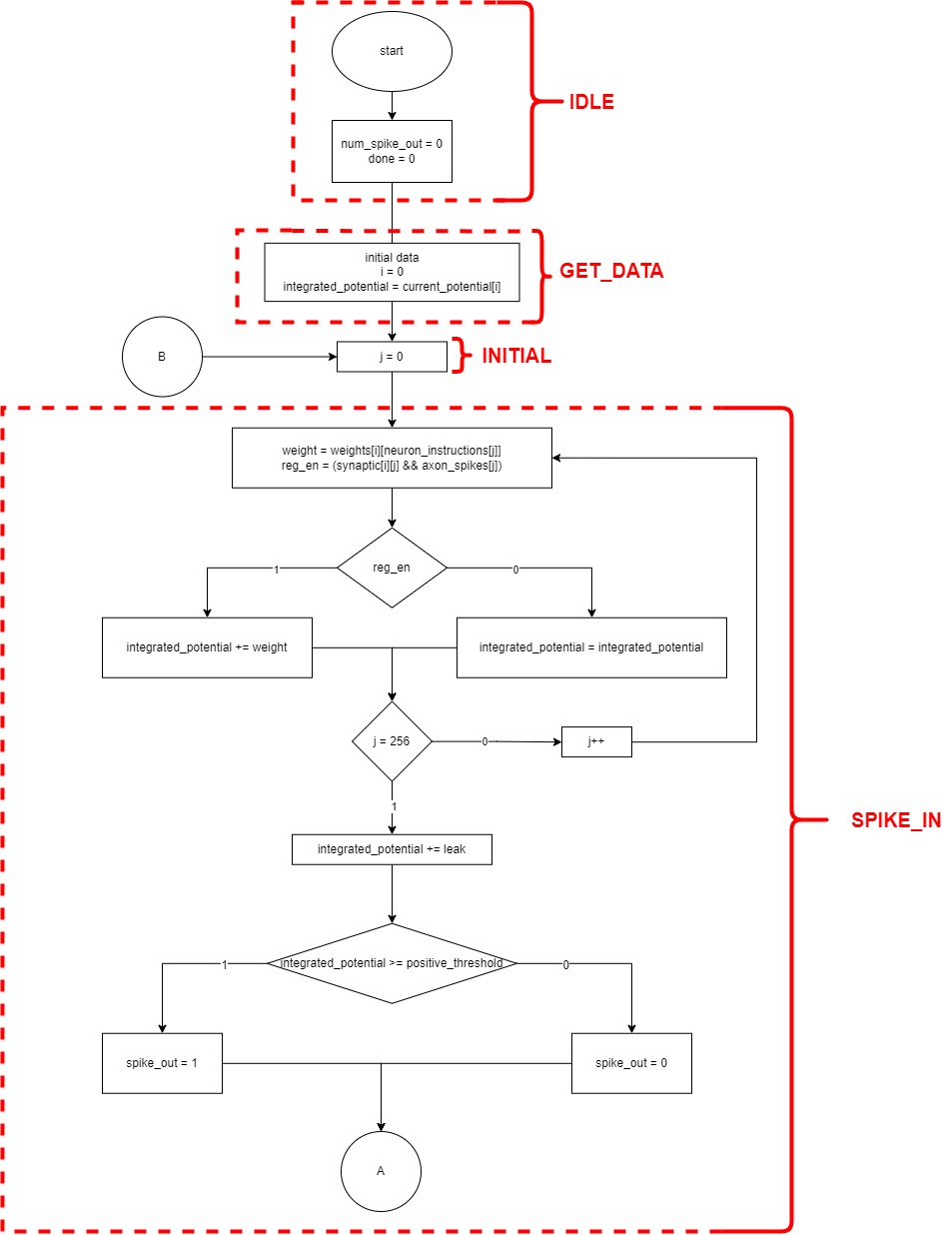
*Figure 7 Timing diagram of Neuron Grid*

## Algorithm and ASMD

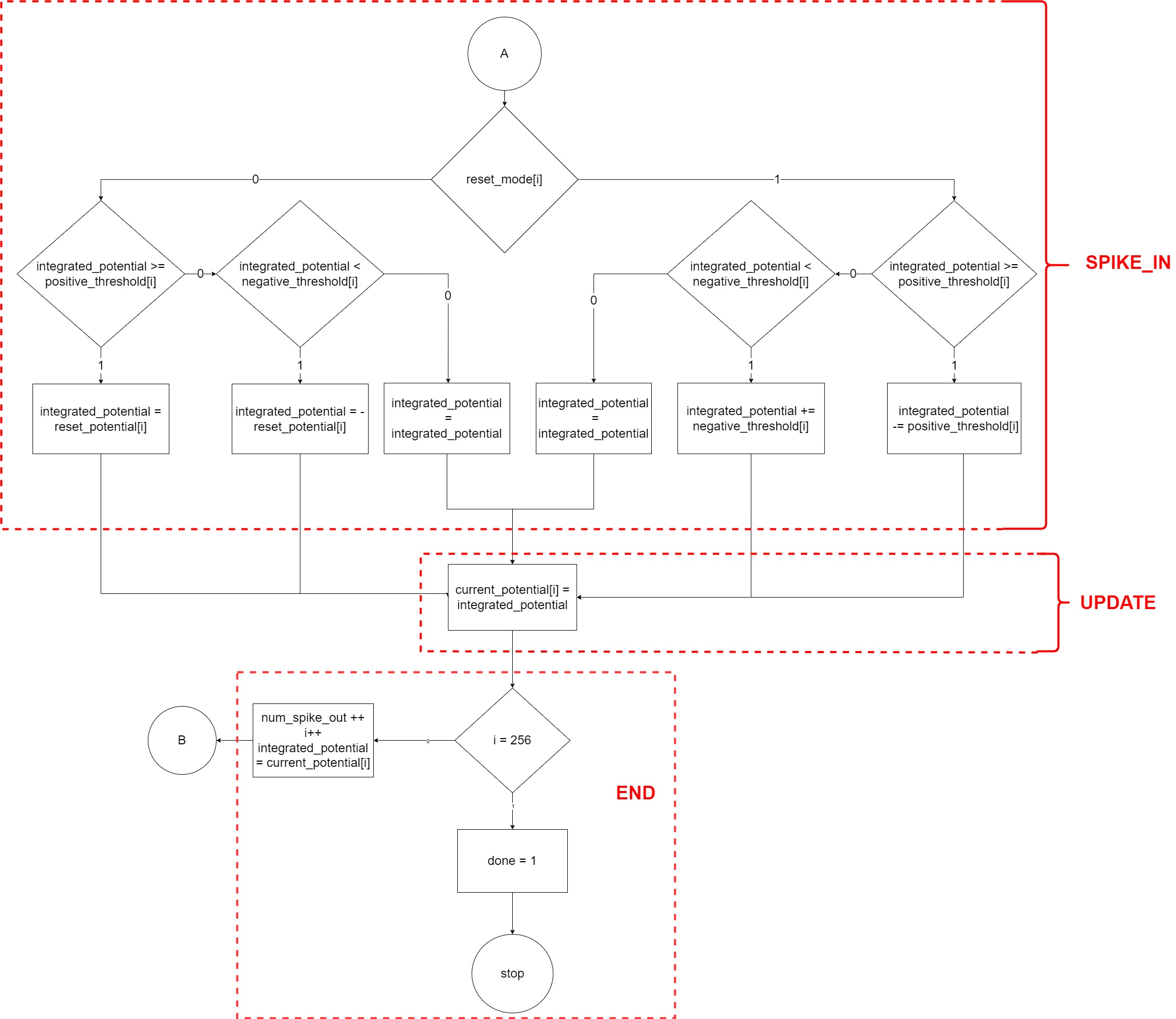
Diagram

Description automatically generated

*Figure 8 Controlling and feedback signals*



*Figure 9* *Flow chart of Neuron Grid (part 1)*



*Figure 10 Flow chart of Neuron Grid (part 2)*

The flowchart is split into states to create ASM

IDLE: Initialize counting variable (**num\_spike\_out**) and finishing variable (**done**).

GET\_DATA: Initialize index variable of neuron (i = 0) and assign **integrated\_potential** = **curent\_potential**

INITIAL: Initialize index variable of axon (j = 0). It is not suitable to combine this state and GET\_DATA because with each neuron, all 256 axons are processed (j from 0 to 255).

SPIKE\_IN: Perform the calculation for a neuron. The calculation is repeated for 256 neurons and the condition of **spike\_out** and **reset\_mode** is considered, so this process is in a state. The condition of **spike\_out** and **reset\_mode** can be considered in another state but it is not necessary because the calculation is not complex and does not affect the clock period significantly.

UPDATE: finish the process of saving new potential to RAM before the program is done.

END: finish the program, **done** =1 announces that the tick is done.

Diagram

Description automatically generated

*Figure 11 ASM diagram of Neuron Grid*

In datapath of Neuron Grid, we have a sub module called Neuron Block to calculate potential and generate spikes with TOP module in *Figure 12* and architecture in *Figure 13*.

Diagram

Description automatically generated

*Figure 12 TOP module of Neuron Block*

Diagram, schematic

Description automatically generated

*Figure 13 Architecture of Neuron Block*

# IMPLEMENTATION

## Software

To implement Neuron Grid, we initially apply the algorithm on C++. This software program can generate **neuron\_parameters** and **axon\_spikes** randomly and store in text files. However **neuron\_instruction** used is from existing files from RANC. Details are mentioned in *Guide to run experiment* file.

To verify our C++ software, we compare its output with RANC hardware. We combine Core Controller and Neuron Block of RANC and write a test bench to compare its results with our Neuron Grid. The test bench read output of software program stored in text files.

The C++ software can create 3 types of file: text file containing axon spikes information in 1 tick, text file containing axon spikes information in many ticks and text file containing all the neuron parameters. They can be created by running 3 generating functions in main().

|  |
| --- |
| int main() {  ///////////Generate////////////  generateCSRAM();  generateManyTickAxonSpikes(16); //16 tick  generateAxonSpikes(); //1 tick  //////////Simulation//////////  NeuronGrid(); //1 tick  ManyTick(16); //16 tick  return 0;  } |

There are also 2 functions to simulate in 1 tick and many ticks in main().

We have functions to receive data from input text files and use the flowchart in *Figure 9* to implement the calculation functions.

## Hardware

To implement Neuron Grid on Verilog, we use ASMD method. To verify Neuron Grid, we compare the results with results generated by software in text files.

### ASMD code

We map 1:1 ASMD to Verilog code.

In controller, conditions to change the states are in ASM’s condition block. Outputs of controller are shown in ASM’s process blocks. Always and sensitivity list are also used. These rules are described in the following Verilog code in controller:

|  |
| --- |
| always @(current\_state, tick, done\_neuron, spike\_in, done\_axon) begin  initial\_axon\_num = 0;  initial\_neuron\_num = 0;  scheduler\_set = 0;  scheduler\_clr = 0;  new\_neuron = 0;  process\_spike = 0;  inc\_axon\_num = 0;  update\_potential = 0;  done = 0;  inc\_neuron\_num = 0;  spike\_out = 0;  //////  case(current\_state)  IDLE: begin  next\_state = tick ? GET\_DATA : IDLE;  end  GET\_DATA: begin  initial\_neuron\_num = 1;  scheduler\_set = 1;  next\_state = INITIAL;  new\_neuron = 1;  end  INITIAL: begin  initial\_axon\_num = 1;  process\_spike = 1;  next\_state = SPIKE\_IN;  end  SPIKE\_IN: begin  process\_spike = 1;  if(done\_axon) next\_state = UPDATE;  else begin  inc\_axon\_num = 1;  next\_state = SPIKE\_IN;  end  end  UPDATE: begin  update\_potential = 1;  spike\_out = spike\_in;  next\_state = END;  end  END: begin  if(done\_neuron) begin  scheduler\_clr = 1;  done = 1;  next\_state = IDLE;  end  else begin  new\_neuron = 1;  inc\_neuron\_num = 1;  next\_state = INITIAL;  end  end  default: next\_state = IDLE;  endcase  end |

In controller, error happens when tick is High in all states except for IDLE which is shown in the following code:

|  |
| --- |
| always @(posedge clk, negedge reset\_n) begin  if(~reset\_n) begin  current\_state <= IDLE;  error <= 0;  end  else begin  current\_state <= next\_state;  if(~error && current\_state != IDLE && tick) error <= ~error;  else error <= error;  end  end |

Inputs of datapath are shown in ASM’s process blocks. Sub-block neuron block is used when there are both synaptic connection and spike shown inthe following code:

|  |
| --- |
| assign done\_axon = (axon\_num == 255);  always @(negedge clk, negedge reset\_n) begin  if(~reset\_n) axon\_num <= 8'd0;  else if(initial\_axon\_num) axon\_num <= 8'd0;  else if(inc\_axon\_num) axon\_num <= axon\_num + 1'b1;  else axon\_num <= axon\_num;  end  always @(negedge clk, negedge reset\_n) begin  if(~reset\_n) neuron\_num <= 8'd0;  else if(initial\_neuron\_num) neuron\_num <= 8'd0;  else if(inc\_neuron\_num) neuron\_num <= neuron\_num + 1'b1;  else neuron\_num <= neuron\_num;  end  always @(posedge clk, negedge reset\_n) begin  if(~reset\_n) done\_neuron <= 0;  else if(neuron\_num == 255) done\_neuron <= 1;  else done\_neuron <= 0;  end  wire reg\_en;  assign reg\_en = (neuron\_parameter[112 + axon\_num] & axon\_spikes[axon\_num]);  neuron\_block neuron\_block(); |

### Testbench

Our testbench read inputs from text files, give them to the module and update data from text file to module.

|  |
| --- |
| //Đọc file mem  reg [367:0] neuron\_parameters [0:255];  initial $readmemb(link\_to\_CSRAM\_data, neuron\_parameters);  reg [1:0] neuron\_instructions [0:255];  initial $readmemb(link\_to\_neuron\_instruction, neuron\_instructions);  //Tự động đưa data từ file mem vào module  always @(neuron\_num) begin  neuron\_parameter = neuron\_parameters[neuron\_num];  end  always @(axon\_num) begin  neuron\_instruction = neuron\_instructions[axon\_num];  end  //Tự động update lại data từ file mem sau mỗi lần tính toán xong cho 1 neuron trong 1 tick  always @(posedge clk) begin  if(update\_potential) neuron\_parameters[neuron\_num][111:103] <= potential\_out;  else neuron\_parameters[neuron\_num][111:103] <= neuron\_parameters[neuron\_num][111:103];  end  //Đọc axon\_spikes từ file  reg [255:0] mem [0:MAX\_NUM\_TICK - 1];  initial begin  if(MAX\_NUM\_TICK == 1) $readmemb(link\_to\_axon\_spike, mem);  else $readmemb(link\_to\_many\_ticks\_axon\_spike, mem);  end  integer row = 0;  always @(scheduler\_set) begin  if(scheduler\_set) begin  axon\_spikes = mem[row];  row = row + 1;  // $display("%b", axon\_spikes);  end  end |

Our testbench assigns tick = 1 after finish 1 tick and stops after all the ticks are processed automatically.

|  |
| --- |
| ////////AUTO//////////////////////////////////////////////////////////  integer num\_tick = 0;  reg finish;  initial finish = 0;  always @(tick) begin  num\_tick = tick ? num\_tick + 1 : num\_tick;  end  //tự động gán tick = 1 sau khi tính xong 1 tick/////////////////////////  always @(done) begin  if(done) begin  repeat(3) @(negedge clk);  tick = 1; @(negedge clk);  tick = 0;  end  else tick = tick;  end  //tự động dừng sau khi đạt đủ số tick/////////////////////////////  always @(num\_tick, MAX\_NUM\_TICK, done) begin  if((num\_tick == MAX\_NUM\_TICK) & done) begin  repeat(2) @(negedge clk);  finish = 1;  #2; $stop;  end  end |

The testbench also writes outputs to a text file and compares to output from software.

|  |
| --- |
| ///////compare with output from software////////////////////////////  reg [255:0] output\_soft [0:MAX\_NUM\_TICK - 1];  reg wrong;  initial wrong = 0;  initial $readmemb(link\_to\_output\_soft, output\_soft);  integer i, j;  always @(finish) begin  if(finish) begin  for(i = 0; i < MAX\_NUM\_TICK; i = i + 1) begin  for(j = 0; j < 256; j = j + 1) begin  if(output\_hard[i][j] != output\_soft[i][j]) begin  $display("Error at neuron %d, tick %d", j, i);  wrong = 1;  end  end  end  end  end  always @(finish) begin  if(finish) begin  #1; if(~wrong) $display("Test pass without error");  end    end |

# RESULT

*Table 1 Synthesis results table*

|  |  |  |
| --- | --- | --- |
| Section | Combination of RANC Token Controller and Neuron Block | Neuron Grid |
| Target Chip Technology | Zynq UltraScale+ MPSoC ZCU106 Evaluation | Zynq UltraScale+ MPSoC ZCU106 Evaluation |
| Maximum Frequency | 278.86 MHz | 331.90 MHz |
| Hardware  Usage | 235 LUTs, 37 FFs | 277 LUTs, 30 FFs |
| Detecting Throughput | 676.6 Mbps | 805.3 Mbps |
| Latency | 66050 clocks | 66048 clocks |

*Table 1* shows the results of our Neuron Grid comparing to Combination of RANC Token Controller and Neuron Block after synthesizing on the same target chip technology Zynq UltraScale+ MPSoC ZCU106 Evaluation.

Our design has higher maximum frequency and detecting throughput (331.90 MHz and 805.3 Mbps comparing to 278.86 MHz and 676.6 Mbps). Our Neuron Grid’s latency also has 2 clocks fewer than RANC’s. Our design uses fewer FFs but more LUTS than RANC’s.

# CONCLUSION

Eventually, the Neuron Grid is successfully designed and meet the desired specification. Although we have to deal with some problems in progress, consequently we have found the way to handle them. For further improvement, we will try to propose a more advanced method to replace our ideal components to achieve a better performance of simulated results. Moreover, throughout this assignment, we have gained a deeper knowledge about those concept. Anyway, nothing is perfect, perhaps our performance is not the most optimized way against this topic. Nevertheless, we hope our project contributes a significant part to the Digital Design, also helps other students to shorten their approach with designing a Neuron Grid. The process of learning and learning how to build a system, a term of simulation that seems easy but very complex and a feat to achieve professionalism. However, under the guidance of Prof. Nguyen Duc Minh, our team has learned the steps to create an economical project with practical application in the most effective way. Along with that, teamwork also helps us to perfect more skills such as arranging time, assigning work to each member, using software for teamwork, ... Our team sincerely thanks for the direction and very practical and straightforward instructions and suggestions from teachers during class presentations, through which we have a more accurate and clear orientation on the way to build products that meet market needs and have practical application. Due to the limited time, the simulation we build is still flawed, we look forward to having your feedback on the product so that we can try harder in the next projects. Once again, my group would like to thank you very much.

Once again, thank you very much for helping us.

REFERENCES

|  |  |
| --- | --- |
| [1] | "Spiking neural network," [Online]. Available: https://en.wikipedia.org/wiki/Spiking\_neural\_network. |
| [2] | Valancius, et al., "FPGA Based Emulation Environment for Neuromorphic Architectures," *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), New Orleans, LA, USA,* pp. 90-97, 2020. |
| [3] | Valancius, et al., "RANC: Reconfigurable Architecture for Neuromorphic Computing," 2020. |