Group 9

Rayyan Hussain
Vishal Kumar
Sibani Sasmala

End-Term Evaluation Report

Introduction

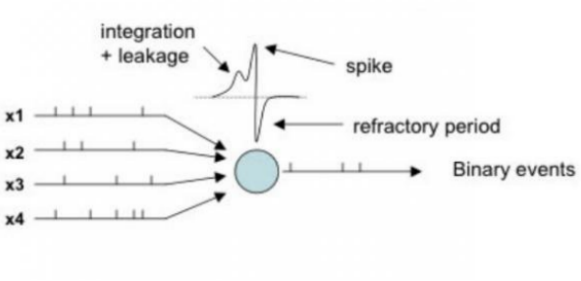
This report examines the implementation of RVNE Instruction Set Architecture (ISA) in neuromorphic processors, which is specified in the paper "Back to Homogeneous Computing: A Tightly-Coupled Neuromorphic Processor With Neuromorphic ISA" (IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 34, NO. 11)

Spiking Neural Networks (SNN)

The features of RVNE make them highly suitable for real-time data processing and complex decision-making applications. SNNs, which rely on discrete spikes for communication instead of continuous signals, are naturally aligned with this computing paradigm.

It incorporates specialized instructions to manage synaptic weights, spikes, neuron states, and both neuron-wise and synaptic-wise current computations. By addressing these needs, the ISA facilitates efficient processing and supports the inherent parallelism of neuromorphic systems.

The following diagram illustrates all the essential actions required for a SNN.



RVNE ISA

The following instructions were implemented:

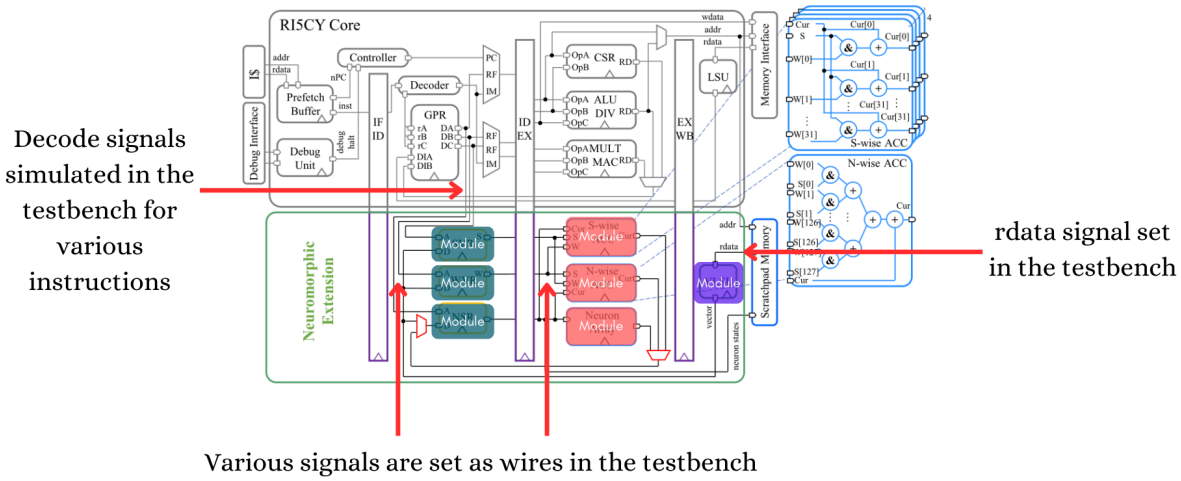
THE FORMATS OF ALL NEUROMORPHIC INSTRUCTIONS IN RVNE

synaptic weight loading instructions					
imm[11:0]	**rs1[4:0]**	**funct3[2:0]**	**rd[4:0]**		**opcode[6:0]**
offset	base	lw.wv	WVR_dst		CUSTOM
imm[11:0]	**rs1[4:0]**	**funct3[2:0]**	**hint[0]**	**rd[3:0]**	**opcode[6:0]**
offset	base	lh.wv	hint	WVR_dst	CUSTOM
offset	base	la.wv	hint	WVR_dst	CUSTOM
spike vector loading instructions					
imm[11:0]	**rs1[4:0]**	**funct3[2:0]**	**hint[0]**	**rd[3:0]**	**opcode[6:0]**
offset	base	lw.sv	SVR_dst		CUSTOM
imm[11:0]	**rs1[4:0]**	**funct3[2:0]**	**hint[0]**	**rd[3:0]**	**opcode[6:0]**
offset	base	lh.sv	hint	SVR_dst	CUSTOM
offset	base	la.sv	hint	SVR_dst	CUSTOM
neuron states and parameters loading instructions					
funct7[6:0]	**rs2[4:0]**	**rs1[4:0]**	**funct3[2:0]**	**rd[4:0]**	**opcode[6:0]**
000 0000	offset	base	lw.rp	00000	CUSTOM
000 0001	offset	base	lw.vt	00000	CUSTOM
000 0010	offset	base	lw.nt	dst	CUSTOM
neuron current computing instructions					
funct7[6:0]	**rs2[4:0]**	**rs1[4:0]**	**funct3[2:0]**	**rd[4:0]**	**opcode[6:0]**
111 0000	SVR_src	WVR_src	convh	NSR_dst	CUSTOM
111 0001	SVR_src	WVR_src	conva	NSR_dst	CUSTOM
111 0010	SVR_src	WVR_src	convmh	NSR_dst	CUSTOM
111 0011	SVR_src	WVR_src	convma	NSR_dst	CUSTOM
111 0100	SVR_src	WVR_src	doth	NSR_dst	CUSTOM
111 0101	SVR_src	WVR_src	dota	NSR_dst	CUSTOM
neuron states updating instructions					
funct7[6:0]	——	——	**funct3[2:0]**	**rd[4:0]**	**opcode[6:0]**
1110100	00000	00000	upds	NSR_src/dst	CUSTOM
1110101	00000	00000	updg	NSR_src/dst	CUSTOM
1110110	00000	00000	upda	NSR_src/dst	CUSTOM

Approach

The Verilog code and testbench replicate the workings of the neuromorphic extension part of the proposed NeuroRV Core.

Every Unit (SVR, NVR, WVR, S-Wise ACC) is a module in the Verilog code with appropriate input and output signals provided in the testbench to emulate various instructions accordingly.



The testbench emulates the different signals sent to the modules for execution. For example, for “lw.wv”, the DC signal would be triggered, sending funct3 + rd of 0000 + rd to take the first 32 bits of the bus to store at the appropriate rd register.

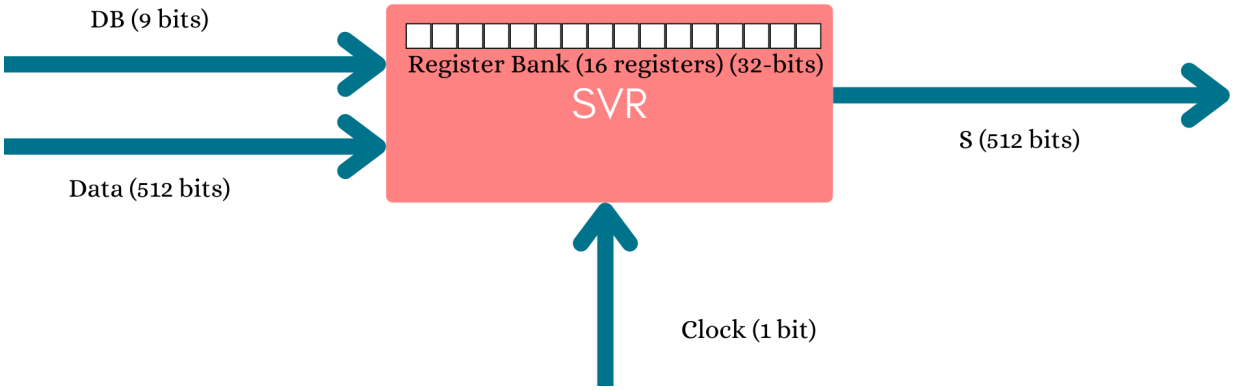
Assumptions

- DA, DB, and DC Signals are 9-bits for uniformity.
- Data, S, W, and Cur bus are taken as 512 bits for uniformity.
- One spike is 1-bit.
- One weight is 4-bits.
- One current is 4-bits.
- Funct3 has been modified to include 4 bits.
- All registers are 32 bits.
- NTR is 4 bits (1111 to represent a spike and 0000 for inhibiting). This decision was made to match the width of the current of a neuron (4 bits) to make neuron state updating easy (Refer to the explanation under NSR Module for more details).
- The formula used for calculating membrane potential:

$$V = I \cdot \text{Resistance}(0.04) + \text{Rest Potential}(0.01)$$

Modules

SVR



Data Signal

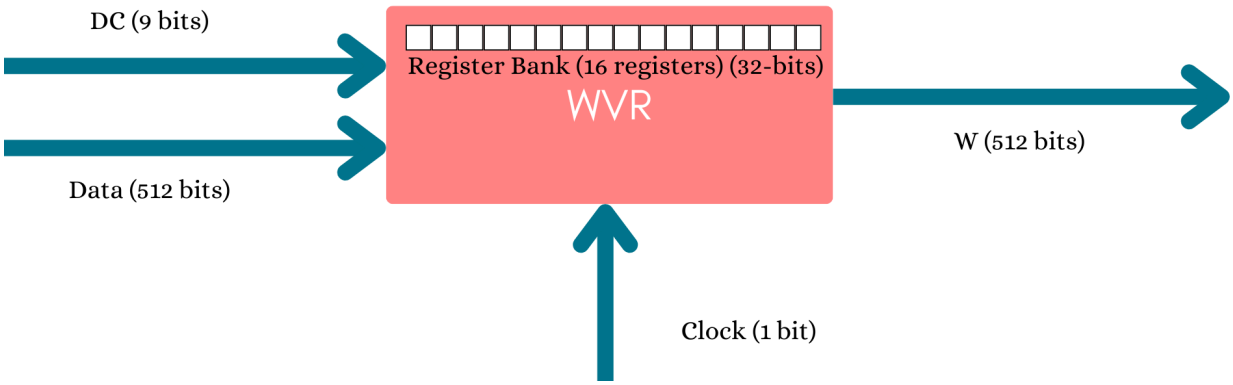
As per our assumptions, 1 bit = 1 spike. Since neuron computation instruction uses at-most 128 neurons, only 128 bits of this bus will be used in the maximum case.

DB Signal

Consists of funct (4-bits) + rd (5-bits). The following funct numbers are used for various cases

1. **0000**: Takes first 32 bits of Data bus and stores it in rd (32 neurons' spike data)
2. **0001**: Takes first 128 bits of Data bus and stores it in four consecutive registers starting from rd (128 neurons' spike data)
3. **0010**: Takes 512 bits of the data bus and stores it in all 16 registers.
4. **0011**: Loads 1 register (32 neurons' spike data) onto first 32 bits of S bus.
5. **0100**: Loads 4 registers (128 neuron's spike data) onto first 128 bits of S bus.

WVR



Data Signal

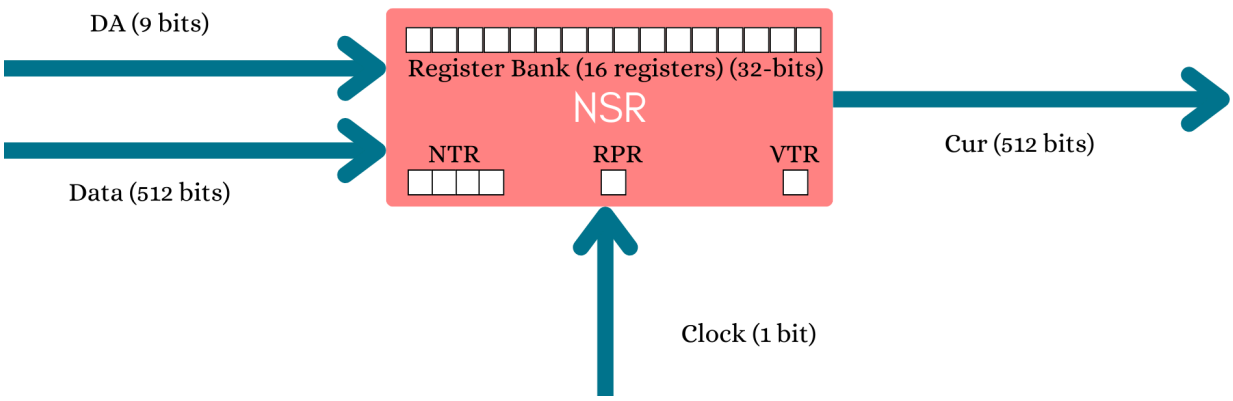
As per our assumptions, 4-bits = 1 weight. Since neuron computation instruction uses at most 128 neurons, all 512 bits of this bus will be used in the maximum case.

DC Signal

Consists of funct (4-bits) + rd (5-bits). The following funct numbers are used for various cases

1. **0000**: Takes the first 32 bits of Data bus and stores it in rd (8 neurons weight data).
2. **0001**: Takes the first 128 bits of Data bus and stores it in four consecutive registers starting from rd (32 neurons' weight data).
3. **0010**: Takes 512 bits of the data bus and stores it in all 16 registers (128 neurons' spike data).
4. **0011**: Loads four registers (32 neurons' spike data) onto the first 128 bits of S bus.
5. **0100**: Loads 16 registers (128 neuron's spike data) onto 512 bits of S bus.

NSR



Data Signal

As per our assumptions, 4-bits = 1 current. Since neuron computation instruction uses at-most 128 neurons, all 512 bits of this bus will be used in the maximum case.

DC Signal

Consists of funct (4-bits) + rd (5-bits). The following funct numbers are used for various cases

1. **0000**: Takes the first 32 bits of the data bus and stores them in RPR.
2. **0001**: Takes the first 32 bits of Data bus and stores it in VTR.
3. **0010**: Takes the first 128 bits of data bus and stores them in all four NPR registers (NPR data of 32 neurons).
4. **0011**: Stores current of 32 Neurons onto 4 consecutive registers indexed by rd. This funct3 is used for convh/doth
5. **0100**: Stores current of 128 Neurons onto all 16 registers. This funct3 is used for conva/dota
6. **0101**: Takes the first 4 bits of data bus and stores it in the first 4 bits of the register indexed by rd (Updating one neuron).
7. **0110**: Takes the first 16 bits of data bus and stores it in the first 16 bits of the register indexed by rd (Updating 4 neurons' current). Used in convmh.
8. **0111**: Takes the first 64 bits of data bus and stores it in two consecutive registers indexed by rd (Updating 16 neurons' current). Used in convma.
9. **1000**: Update states of one NTR register.
10. **1001**: Update states of all 4 NTR registers.

State Updating

The algorithm for state updating is as follows:

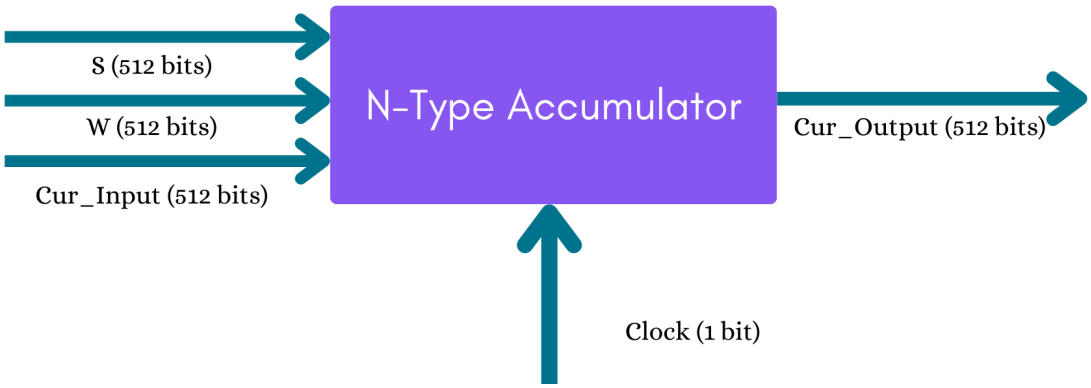
1. If RPR == 0.
    a. For every four bits from 0 - 32 [i * 4 +: 4]:
        i. Fetch current from current_register indexed by rd for the same bits [i * 4 +: 4].

ii. Calculate potential.

iii. If potential has crossed VTR:

1. If it does, set the NTR of those four bits to 1111.

NAcc



S Signal

Input 32 bits (32 Neurons' spike data) or 128 bits (128 Neurons' spike data) sent by SVR.

W Signal

Input 128 bits (32 Neurons' spike data) or 512 bits (128 Neurons' spike data) sent by WVR.

Cur_Input Signal

Initially 0, Cur_Input signal is updated with one of Cur_Output from N-Type, S-Type, and Neuron Array after computation.

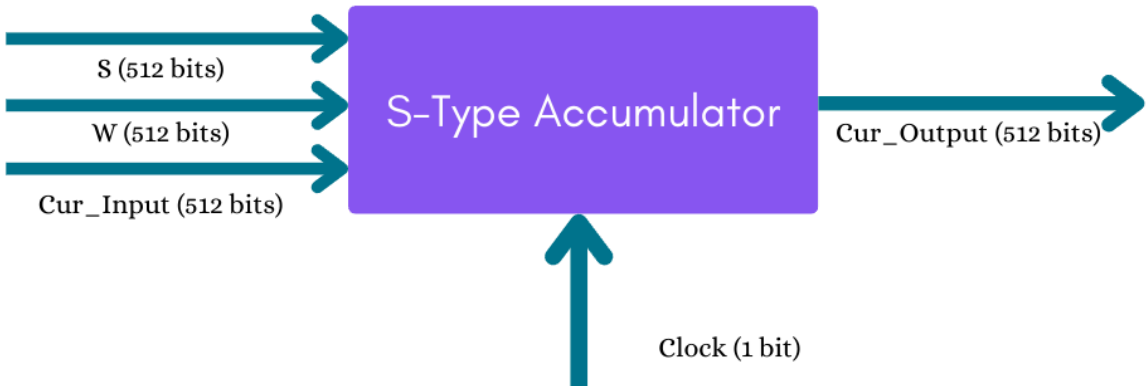
Accumulation Logic

Computation is done for all 128 neurons across the bus regardless of whether the input is 32 or 128 from S/W. NeuroRV Core doesn't specify any signal going into the Accumulator. Hence, for instructions like "doth/convh" if only 32 neurons are loaded, other bits of the S signal is 0, and other bits of Cur_Output are computed as 0. Moreover, NSR will only store one (4/16 in convmh/convma - refer to Neuron Array Module) neuron, taking the first 4 bits of the Cur_Output bus.

The following formulae are used to compute N-Type Accumulation (one neuron's output current):

$$\sum_{i=0}^{128} S[i] \cdot W[j] \quad \text{where } j = i \cdot 4 + 4 \quad + \text{Cur\_input}[3:0]$$

SAcc



S Signal

Input 32 bits (32 Neurons' spike data) or 128 bits (128 Neurons' spike data) sent by SVR.

W Signal

Input 128 bits (32 Neurons' spike data) or 512 bits (128 Neurons' spike data) sent by WVR.

Cur_Input Signal

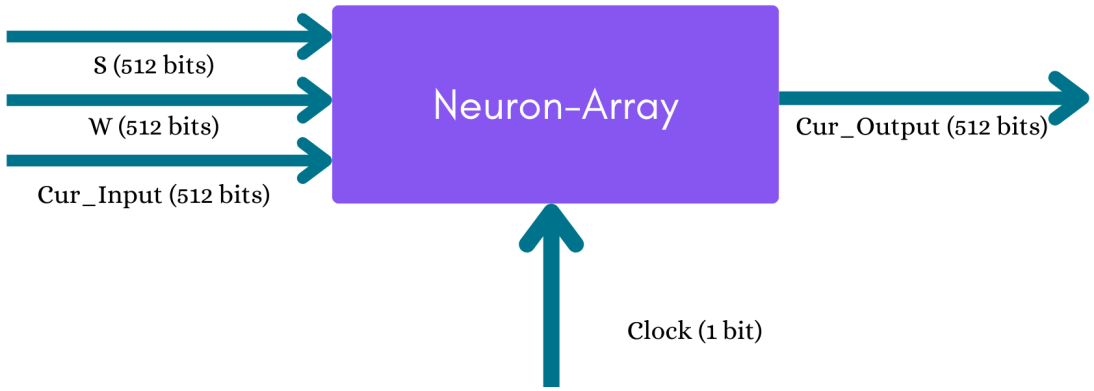
Initially 0, Cur_Input signal is updated with one of Cur_Output from N-Type, S-Type, and Neuron Array after computation.

Accumulation Logic

Similar to N-Type, computation is done for all 128 neurons of the S/W/Cur bus regardless of the input/output. The following equation is used to compute the S-type.

$$\text{Cur\_Output}[i \cdot 4+ : 4] = (\text{S}[0] \cdot \text{W}[i \cdot 4+ : 4]) + \text{Cur\_Input}[i \cdot 4+ : 4]$$

NeuronArray



S Signal

Input 128 bits (128 Neurons' spike data) sent by SVR.

W Signal

Input 512 bits (128 Neurons' spike data) sent by WVR.

Cur_Input Signal

Initially 0, the Cur_Input signal is updated with one of Cur_Output from N-Type, S-Type, and Neuron Array after computation.

Accumulation Logic

Since convmh/convma takes in 128 neurons as input from SVR, WVR, and NSR, we utilize 128 bits of S signal and 512 bits of W signal. Since Computation is done for 4/16 neurons, S-type computation is done for all 16 neurons. The module sends a 16-bit output signal in Cur_Output (works for 4 neurons case as bits of other 12 neurons will be 0, and NSR would be taking the first 16 bits / 4 neurons of the bus to store in one register as appropriate DA signal will be sent).

Simulating Instructions (Testbench)

Loading in Weight Vectors

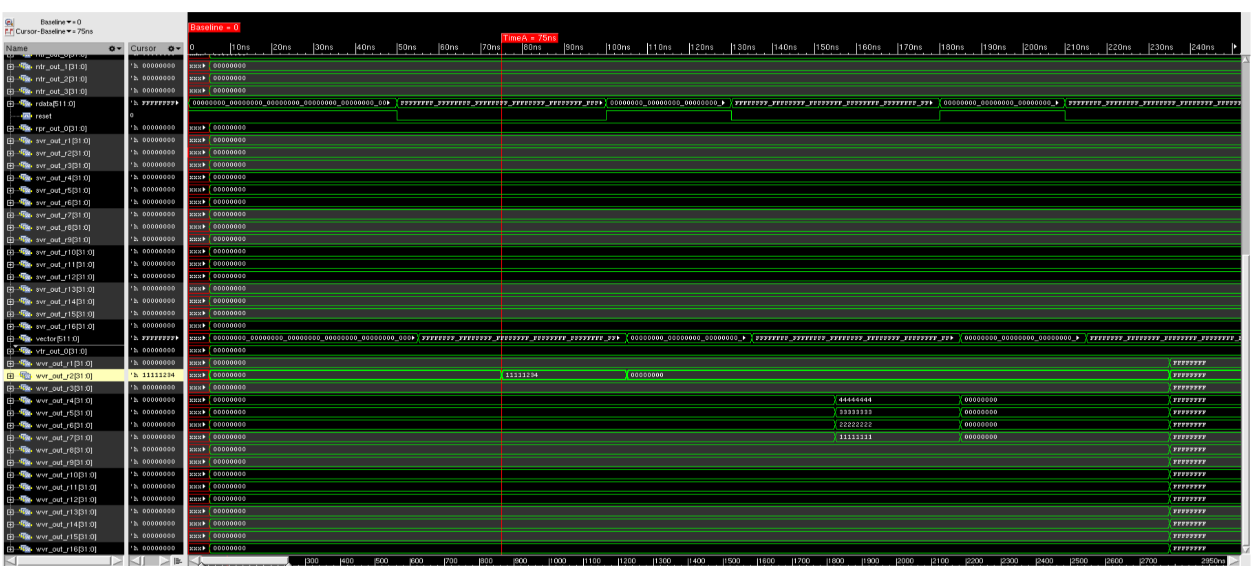
```
//Resetting
clk = 0;
reset = 1; #50; reset = 0;

//WVR Storing Instructions--------------------------------------------------
//lw.wv
DC = 9'b000000001; //A = funct (4 bits) + rd (5 bits)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF11111234; // Test data
#50;reset = 1; #30;reset = 0;
//lh.wv
DC = 9'b000100011; //A = funct (4 bits) + rd (5 bits)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF11111111222222223333333344444444; // Test data
#50;reset = 1; #30;reset = 0;
//la.wv
DC = 9'b001000001;
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test data
#50;reset = 1; #30;reset = 0;
```

Initially, the clock signal is reset to clear all register banks. After 50 ns, the reset signal is pulled high (disabled), and the WVR instruction is executed. The DC signal, which includes the function bits and the destination address, loads different weights into the WVR bank.

There are three types of WVR load instructions.

Case 1 (lw.wv): The function bits 0000 (4 bits) corresponds to the lw.wv instruction, which loads a word (8 weights, 8 bytes) of data. The data is sourced from a 512-bit rdata and is directed to the WVR bank using a 5-bit address. As seen in the waveform, an address value of 0x1 loads the rdata 0x11111234 into the second register of the WVR bank.

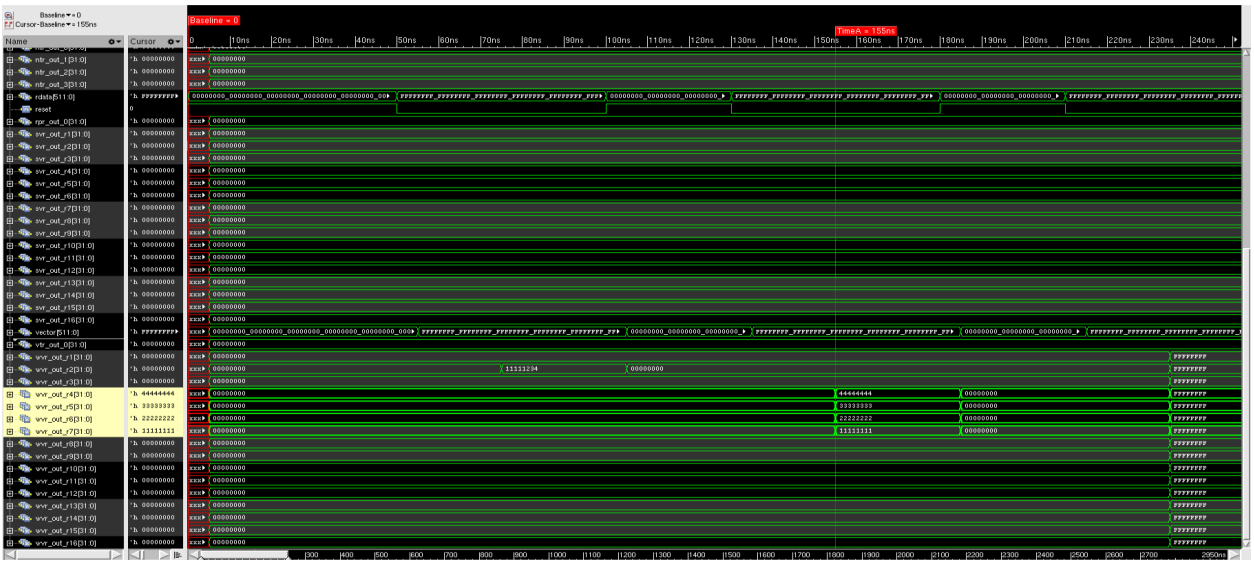
```
DC = 9'b000000001
```



Case 2 (lh.wv): The function bits 0001 (4 bits) correspond to the lh.wv instruction, which loads half weight (32 weights, 32 bytes) of data. This data is sourced from a 512-bit rdata and written to the WVR bank using a 5-bit address. As observed in the waveform, an address value of 0x3 loads the

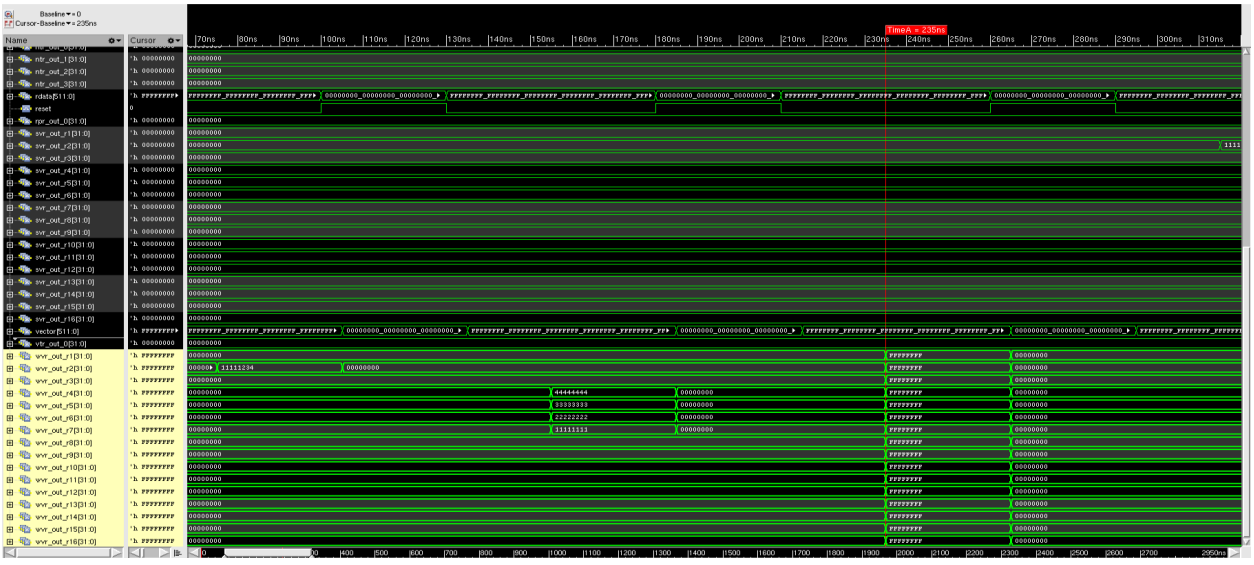
`rdata = 0x11111111222222223333333344444444`, starting from the 4th register of the WVR bank in a contiguous manner.

```
DC = 9'b000100011
```



Case 3 (la.wv): The function bits 0010 (4 bits) correspond to the la.wv instruction, which loads all (128 weights, 128 bytes) of data. This data is sourced from a 512-bit Rdata and written to the WVR bank using a 5-bit address. As observed in the waveform, the rdata is loaded from the first register into all 16 register banks of the WVR bank in a contiguous manner.

```
DC = 9'b001000001
```



Loading in Spike Vectors

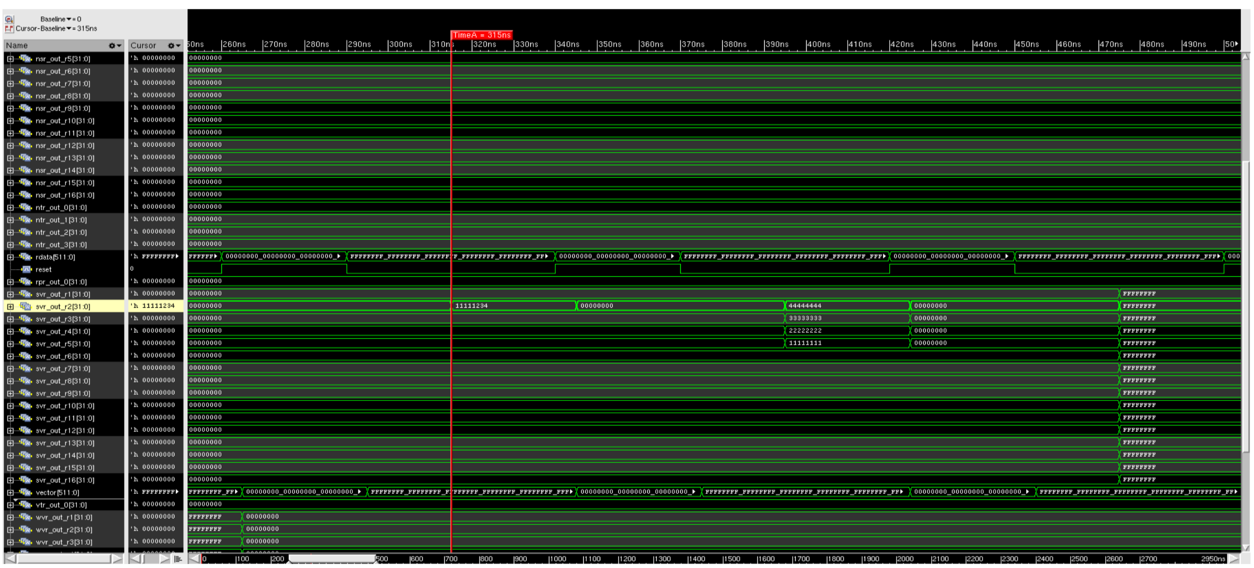
```
//SVR Storing Instructions-----------------------------------------------
//lw.sv
DB = 9'b000000001; //A = funct (4 bits) + rd (5 bits)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF11111234; // Test data
#50;reset = 1; #30;reset = 0;
//lh.sv
DB = 9'b000100001; //A = funct (4 bits) + rd (5 bits)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF11111111222222223333333344444444; // Test data
#50;reset = 1; #30;reset = 0;
//la.sv
DB = 9'b001000001;
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test data
#50;reset = 1; #30;reset = 0;
```

The DB signal, which includes the function bits and the destination address, is used to load different spikes into the SVR bank.

There are three types of SVR load instructions.

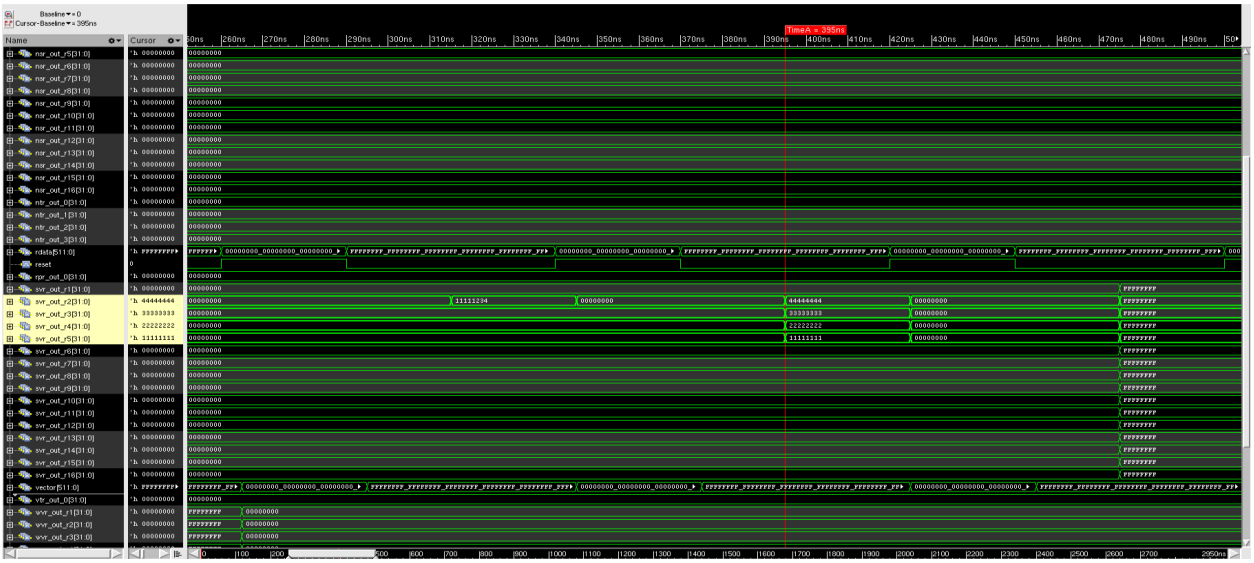
Case 1 (lw.sv): The function bits 0000 (4 bits) corresponds to the lw.sv instruction, which loads a word (32 spikes, 32 bits) of data. The data is sourced from a 512-bit rdata and is directed to the SVR bank using a 5-bit address. As seen in the waveform an address value of 0x1 loads the data 0x11111234 into the second register of the SVR bank.

```
DB = 9'b000000001
```



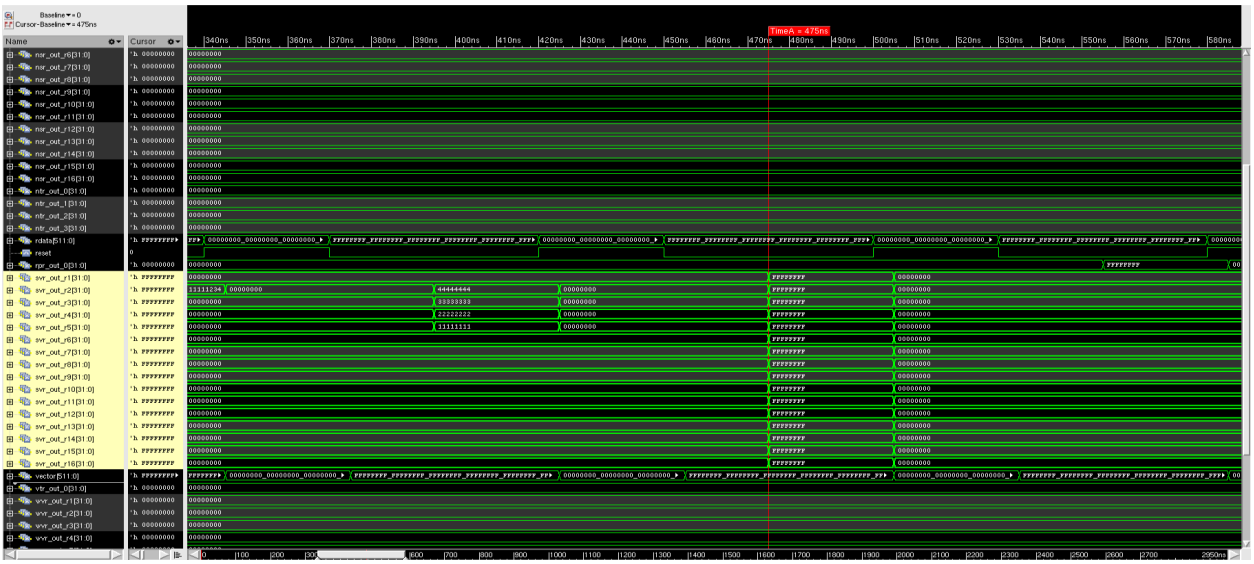
Case 2 (lh.sv): The function bits 0001 (4 bits) corresponds to the lh.sv instruction, which loads half spikes (128 spikes, 128 bits) of data. This data is sourced from a 512-bit rdata and written to the WVR bank using a 5-bit address. As observed in the waveform, an address value of 0x1 loads the data 0x11111111222222223333333344444444, starting from the 2nd register of the SVR bank in a contiguous manner.

```
DB = 9'b000100001
```



Case 3 (la.sv): The function bits 0010 (4 bits) corresponds to the la.sv instruction, which loads all (512 spikes, 512 bits) of data. This data is sourced from a 512-bit rdata and written to the SVR bank using a 5-bit address. As observed in the waveform, the data is loaded from the first register into all 16 register banks of the SVR bank in a contiguous manner.

```
DB = 9'b001000001
```



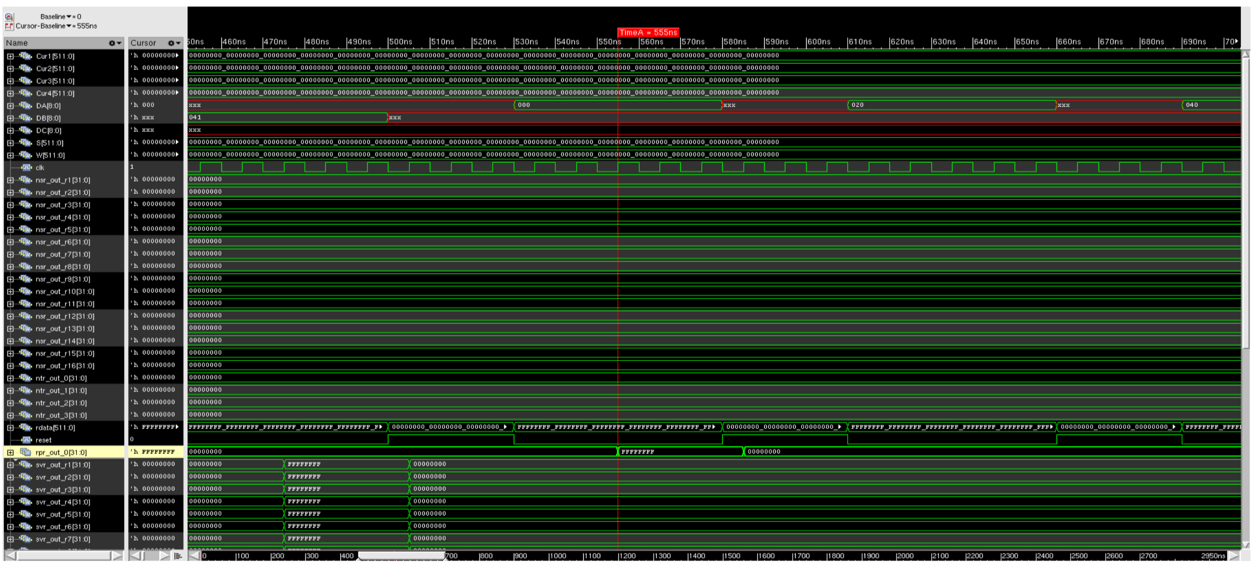
Loading Neuron Paramters

```
//NSR Storing Instructions-----------------------------------------------
//lw.rp
DA = 9'b000000000; //A = funct (4 bits) + rd (5 bits)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test data
#50;reset = 1; #30;reset = 0;
//lw.vt
DA = 9'b000100000; //A = funct (4 bits) + rd (5 bits)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test data
#50;reset = 1; #30;reset = 0;
//lw.nt
DA = 9'b001000000;
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test data
#50;reset = 1; #30;reset = 0;
```

There are three types of Neuron parameter load instructions.

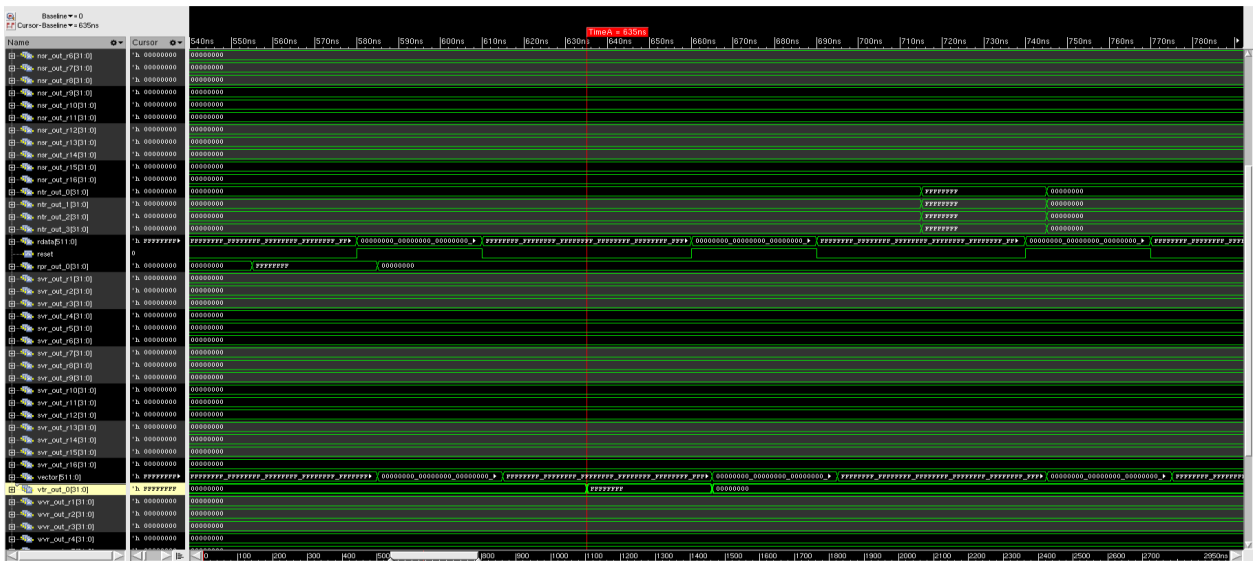
Case 1 (lw.rp): The function bits 0000 (4 bits) corresponds to the lw.rp instruction, which loads a refractory period (32 bits) of data. This data is sourced from a 512-bit rdata and directed to the RPR register in the NSR bank using a fixed 5-bit address. As seen in the waveform an fixed address value loads the data 0xFFFFFFFF into the pre-defined register of the NSR bank.

```
DA = 9'b000000000
```



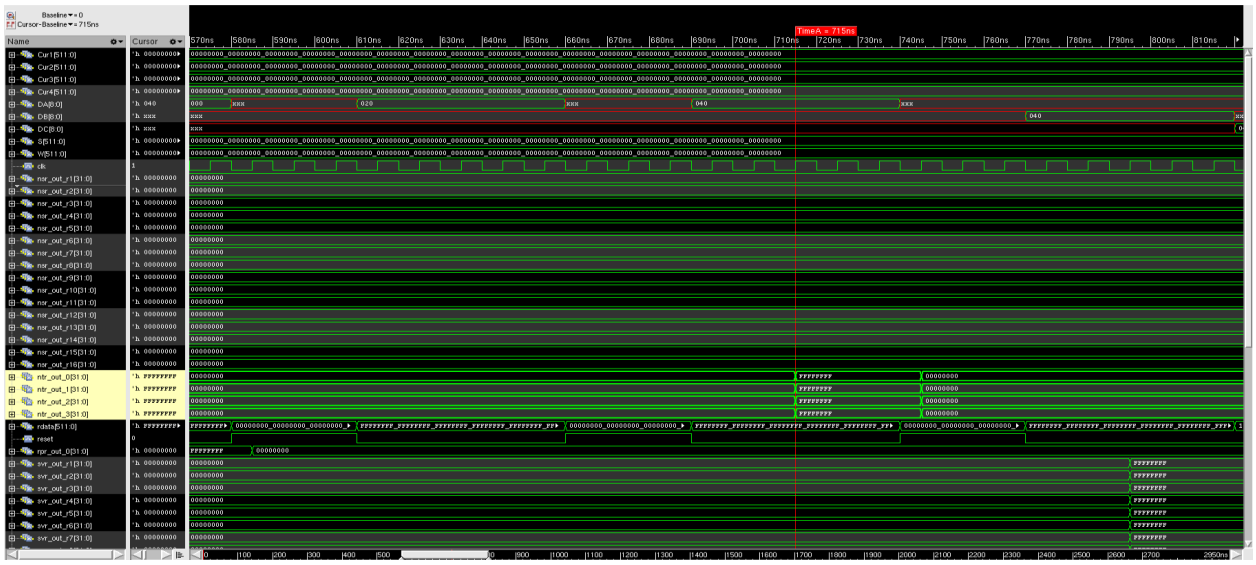
Case 2 (lw.vt): The function bits 0001 (4 bits) corresponds to the lw.vt instruction, which loads a threshold voltage (32 bits) of data. This data is sourced from a 512-bit rdata and directed to the VTR register in the NSR bank using a fixed 5-bit address. As seen in the waveform an fixed address value loads the data 0xFFFFFFFF into the pre-defined register of the NSR bank.

```
DA = 9'b000100000
```



Case 3 (lw.nt): The function bits 0010 (4 bits) corresponds to the lw.nt instruction, which loads neuron-type data. This data is extracted from a 512-bit rdata and stored in the NTR register within the NSR bank, distributed across four 32-bit NTR registers. As observed in the waveform, it loads data for 32 neurons, with each bit indicating the neuron type, into the designated registers of the NSR bank..

```
DA = 9'b001000000
```



Neuron Computation

convh

The instruction read 32 weights from the Weight Vector Register (WVR) and 32 1-bit spikes from the Spike Vector Register (SVR), updating the current of a specified neuron through neuron-wise synaptic operations with a parallelism of 32.

```
//convh
//1. Storing data into WVR and SVR from Memory (la.wv)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test dat
DB = 9'b001000000;
#50;DB = 9'bxxxxxxxxx;
//(la.sv)
rdata = 512'h11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111;
DC = 9'b001000000;
#50;
//2. Sending approriate DB DC Signals to WVR and SVR respectively to load the data from registers
DB = 9'b001100000; //load spike vectors of 32 neurons (1 register)
DC = 9'b001100000; //load weight vectors of 32 neurons (4 registers)
#100; //time for computation
DA = 9'b010100010; //Storing the result in NSR register 4
rdata = Cur3; //selecting Output from N-Type accumulator
#50;
reset = 1; #50; reset = 0;
```

- As explained earlier `DB= 9'b001000000` and `DC=9'b001000000` are used to load all 16 SVR and 16 WVR for the required computation of the convh.
- A `DB = 9'b001100000` signal is then sent to load spikes of 32 Neurons (1 Register) onto the first 32 lines of the output bus.
- `DC = 9'b001100000` is sent to load weights of 32 Neurons (4 Register) onto the first 128 lines of the output bus.
- Finally, `DA=9'b010100010` signal is sent along with the computed rdata (current_ouput in the respective module is sent to rdata, which then sends it as Data signal to NSR) from N-type accumulator to update the current of 1 neuron (4 bits each) in NSR's register bank indexed by (00010).

conva

These instructions read 128 weights from the Weight Vector Register (WVR) and 128 1-bit spikes from the Spike Vector Register (SVR), updating the current of a specified neuron through neuron-wise synaptic operations with a parallelism of 128.

```
//conva
//1. Storing data into WVR and SVR from Memory (la.wv)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test dat
DB = 9'b001000000;
#50;DB = 9'bxxxxxxxxx;
//(la.sv)
rdata = 512'h11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111;
DC = 9'b001000000;
#50;
//2. Sending approriate DB DC Signals to WVR and SVR respectively to load the data from registers
DB = 9'b010000000; //load spike vectors of 128 neurons (4 registers)
DC = 9'b010000000; //load weight vectors of 128 neurons (16 registers)
#100; //time for computation
DA = 9'b010100010; //NSR signal to storeType 128 neurons
rdata = Cur3; //selecting Output from N- accumulator
#50;
reset = 1; #50; reset = 0;
```

- `DB= 9'b001000000` and `DC=9'b001000000` user for loading all 16 SVR and 16 WVR for the required computation of the conva.

- Then `DB = 9'b010000000`, which loads in spikes of 128 Neurons (4 Register) onto the first 128 lines of the output bus.

- `DC = 9'b010000000`; Load weights of 128 Neurons (16 Register) onto the 512 lines of the output bus.

- `DA=9'b010100010` signal along with the computed rdata (current_ouput in the respective module is sent to rdata, which then sends it as Data signal to NSR) from the N-Type accumulator is used to update the current of 1 neuron (4 bits each) in NSR's register bank indexed by 00010.

convmh

The instructions read 128 weights from the WVR and 128 1-bit spikes from the SVR, performing multiple neuron-wise synaptic operations to update the currents of 4 specified neurons with a parallelism of 128, facilitating the development of neural networks with smaller weight kernels.

```
//convmh - stores 4 neurons
//1. Storing data into WVR and SVR from Memory (la.wv)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test dat
DB = 9'b001000000;
#50;DB = 9'bxxxxxxxxx;
//(la.sv)
rdata = 512'h11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111;
DC = 9'b001000000;
#50;
//2. Sending approriate DB DC Signals to WVR and SVR respectively to load the data from registers
DB = 9'b001100000; //load spike vectors of 128 neurons (4 registers)
DC = 9'b001100000; //load weight vectors of 128 neurons (16 registers)
#100; //time for computation
DA = 9'b011000010; //Storing the result 4 neurons in NSR register 4
rdata = Cur4; //selecting Output from N-Type accumulator
#50;
reset = 1; #50; reset = 0;
```

- `DB= 9'b001000000` and `DC=9'b001000000` used for loading all 16 SVR and 16 WVR for the required computation of the convmh.
- `DB = 9'b010000000`: Loads spikes of 128 Neurons (4 Register) onto the first 128 lines of the output bus.
- `DC = 9'b010000000`: Loads weights of 128 Neurons (16 Register) onto the 512 lines of the output bus.
- `DA=9'b010100010` signal along with the computed rdata (current_ouput in the respective module is sent to rdata, which then sends it as Data signal to NSR) from Neuron Array Accumulator is used for updating the current of 4 neurons (4 bits each) in NSR's register bank indexed by 00010.

convma

The instructions read 128 weights from the WVR and 128 1-bit spikes from the SVR, performing multiple neuron-wise synaptic operations to update the currents of 16 specified neurons with a parallelism of 128, facilitating the development of neural networks with smaller weight kernels.

```
//convma - stores 16 neurons
//1. Storing data into WVR and SVR from Memory (la.wv)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test dat
DB = 9'b001000000;
#50;DB = 9'bxxxxxxxxx;
//(la.sv)
rdata = 512'h11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111;
DC = 9'b001000000;
#50;
//2. Sending approriate DB DC Signals to WVR and SVR respectively to load the data from registers
DB = 9'b010000000; //load spike vectors of 128 neurons (4 registers)
DC = 9'b010000000; //load weight vectors of 128 neurons (16 registers)
#100; //time for computation
DA = 9'b011100010; //Storing result of 16 neurons in register 4 and 5
rdata = Cur3; //selecting Output from N- accumulator
#50;
//Not resetting so as to use the result for state updating (next instruction)
reset = 1; #50; reset = 0;
```

- `DB = 9'b001000000` and `DC = 9'b001000000` were used to load all 16 SVR and 16 WVR for the required computation of the convma.
- `DB = 9'b010000000`; Load spike of 128 Neurons (4 Register) onto the first 128 lines of the output bus.
- `DC = 9'b010000000`; Load weights of 128 Neurons (16 Register) onto the 512 lines of the output bus.
- `DA=9'b011100010` signal along with the computed rdata (current_ouput in the respective module is sent to rdata, which then sends it as Data signal to NSR) from Neuron Array Accumulator is used for updating the current of 16 neurons (4 bits each) in NSR's register bank indexed by 00010 and 00010.

dota

These instructions read 128 weights from the WVR, a 1-bit spike from the SVR, and a set of neuron currents, performing synapse-wise operations to update the currents of 128 specified neurons.

```
//Neuron Current Computing Instructions--------------------------------------------------
//dota
//1. Storing data into WVR and SVR from Memory (la.wv)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test dat
DB = 9'b001000000;
#50;DB = 9'bxxxxxxxxx;
//(la.sv)
rdata = 512'h11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111;
DC = 9'b001000000;
#50;|
//2. Sending approriate DB DC Signals to WVR and SVR respectively to load the data from registers
DB = 9'b010000000; //load spike vectors of 128 neurons (4 registers)
DC = 9'b010000000; //load weight vectors of 128 neurons (16 registers)
#100; //time for computation
DA = 9'b010000000; //NSR signal to store 128 neurons
rdata = Cur2; //selecting Output from S-Type accumulator
#50;
reset = 1; #50; reset = 0;
```

- `DB= 9'b001000000` and `DC=9'b001000000` user for loading all 16 SVR and 16 WVR for the required computation of the dota.

- `DB = 9'b010000000`; Load spike of 128 Neurons (4 Register) onto the first 128 lines of the output bus.

- `DC = 9'b010000000`; Load weights of 128 Neurons (16 Register) onto the 512 lines of the output bus.
- `DA=9'b010000000` signal is sent along with the computed rdata (current_ouput in the respective module is sent to rdata, which then sends it as Data signal to NSR) from S-Type Accumulator to update the current of 128 neurons (4 bits each) in all 16 NSR's register bank.

doth

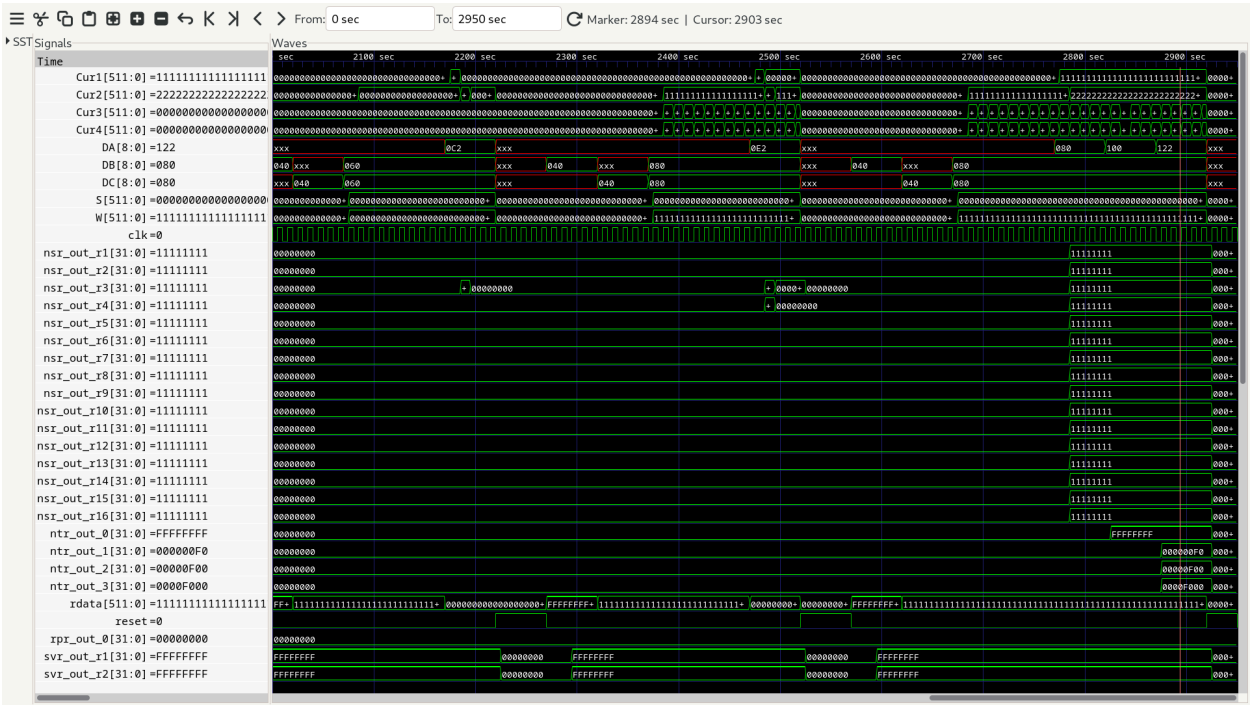
These instructions read 32 weights from the WVR, a 1-bit spike from the SVR, and a set of neuron currents, performing synapse-wise operations to update the currents of 32 specified neurons.

```
//doth
//1. Storing data into WVR and SVR from Memory (la.wv)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test dat
DB = 9'b001000000;
#50;DB = 9'bxxxxxxxxx;
//(la.sv)
rdata = 512'h11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111;
DC = 9'b001000000;
#50;
//2. Sending approriate DB DC Signals to WVR and SVR respectively to load the data from registers
DB = 9'b001100000; //load spike vectors of 32 neurons (1 register)
DC = 9'b001100000; //load weight vectors of 32 neurons (4 registers)
#100; //time for computation
DA = 9'b001100000; //NSR signal to store 128 neurons
rdata = Cur2; //selecting Output from S-Type accumulator
#50;
reset = 1; #50; reset = 0;
```

- `DB= 9'b001000000` and `DC=9'b001000000` user for loading all 16 SVR and 16 WVR for the required computation of the doth.
- `DB = 9'b001100000`; Load spikes of 32 Neurons (1 Register) onto the first 128 lines of the output bus.
- `DC = 9'b001100000`; Load weights of 32 Neurons (4 Registers) onto the 512 lines of the output bus.
- `DA=9'b001100000` signal is sent along with the computed rdata (current_ouput in the respective module is sent to rdata, which then sends it as Data signal to NSR) from the S-Type Accumulator to update the currents of 32 neurons (4 bits each) in 4 of NSR's register bank indexed by 00000.

Neuron States Updating

```
//Neuron State Updating Instructions-----------------------------------------------
//Re running dota
//1. Storing data into WVR and SVR from Memory (la.wv)
rdata = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Test dat
DB = 9'b001000000;
#50;DB = 9'bxxxxxxxxx;
//(la.sv)
rdata = 512'h11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111;
DC = 9'b001000000;
#50;
//2. Sending approriate DB DC Signals to WVR and SVR respectively to load the data from registers
DB = 9'b010000000; //load spike vectors of 128 neurons (4 registers)
DC = 9'b010000000; //load weight vectors of 128 neurons (16 registers)
#100; //time for computation
DA = 9'b010000000; //NSR signal to store 128 neurons
rdata = Cur2; //selecting Output from S-Type accumulator
#50;

//upds
DA = 9'b100000000; //Updating states of one register (8 neurons)
#50
//updg
DA = 9'b100100010; //Updating states of all regiseters (32 neurons)
#50
reset = 1; #30; reset = 0;
```



upds

```
DA = 9'b100000000;
```

As explained previously in NSR Module's Approach, 1000 as funct3 bits of DA signal updates states of neurons in one NTR register (indexed by 00000). For more information, refer to "States Update" section under NSR Module's approach

updg

```
DA = 9'b100100000;
```

As explained previously in NSR Module's Approach, 1001 as funct3 bits of DA signal updates states of neurons in all 4 NTR registers. For more information, refer to "States Update" section under NSR Module's approach