Domain-Specific Accelerator

Bi-Fan Liu

Outline

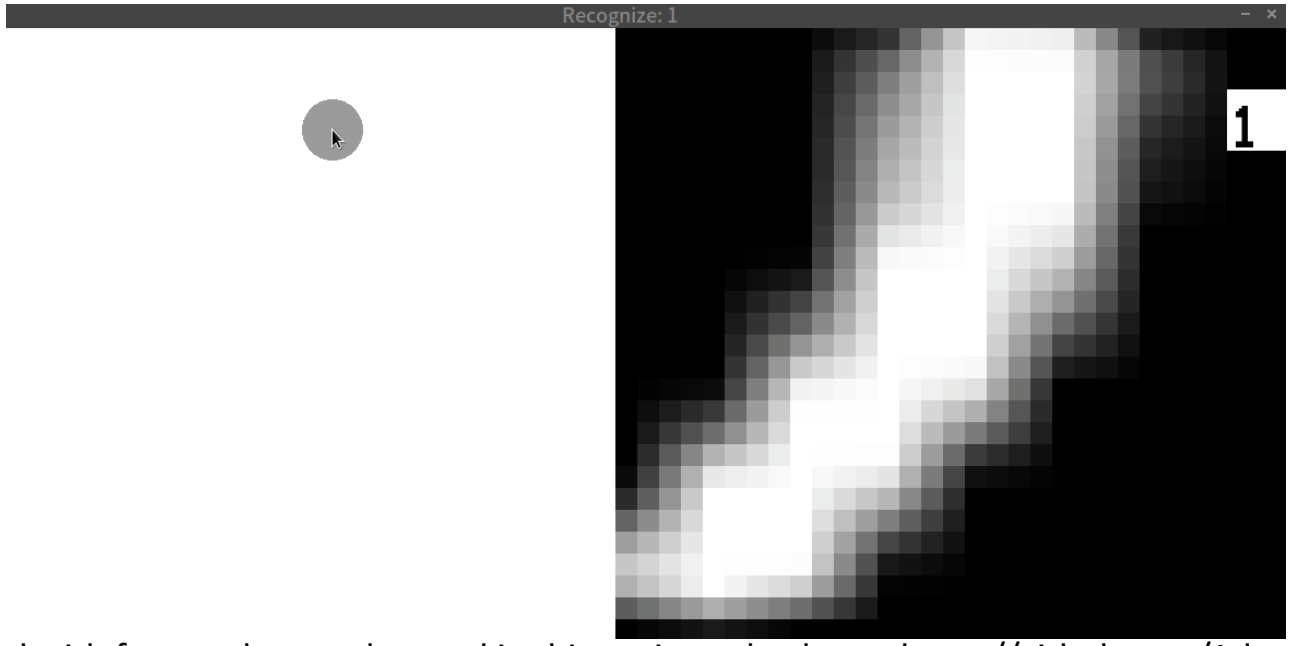
- Personal Project
  - Transitioning neural network computations from software to hardware
    - Introduction
    - Goal
    - System Architecture (32-bits RISC-V Architecture)
    - Dcache Architecture
    - Original Neural Network Evaluate C code
    - Neural Network Computation Implemented in Hardware
    - Result

Outline

- Personal Project
  - Transitioning neural network computations from software to hardware
    - Introduction
    - Goal
    - System Architecture (32-bits RISC-V Architecture)
    - Dcache Architecture
    - Original Neural Network Evaluate C code
    - Neural Network Computation Implemented in Hardware
    - Result

Hand-Written Character Recognition

- We will use a multi-layer perceptrons (MLP) for hand-written character recognition
  - The MLP has 3 layers with 784, 48, 10 neurons in each layer
- The neural network weights (i.e. the program) is trained by the MNIST dataset:
  - Each image has 28×28 pixels



To train a model with format that can be used in this project, check out: https://github.com/AdamYuan/SimpleNN

MLP Layer Design for MNIST Data

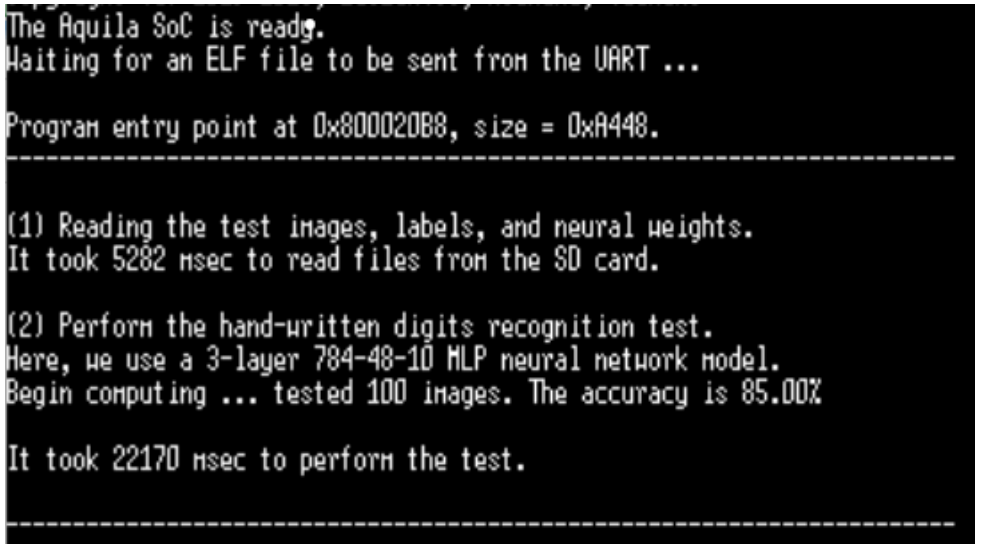
- Since the MLP has 1D input layer, we must convert 2D image input to 1D input:
  - Using the scanline order to do the conversion: $R^{28\times28} \rightarrow R^{784\times1}$
  - Therefore, we need 784 input neurons
- The output layer shows the "likelihood" of each digits
  - A reasonable choice is to use 10 output neurons
  - The maximal neuron gives us the most likely digit in the image
- The # of hidden layer neurons is a tough choice
  - A tradeoff between accuracy and complexity
  - Can be chosen by trial-and-err†

Outline

- Personal Project
  - Transitioning neural network computations from software to hardware
    - Introduction
    - Goal
    - System Architecture (32-bits RISC-V Architecture)
    - Dcache Architecture
    - Original Neural Network Evaluate C code
    - Neural Network Computation Implemented in Hardware
    - Result

Goal

- In this Project, integrate a domain-specific accelerator (DSA) to Aquila (a RISC-V Architecture Core) to improve the speed of an MLP neural network
- Add a vector floating-point HW IP by Xilinx into the Aquila SoC
- Learn how to communicate with Dcache
- Use FPGA for circuit development and ILA for circuit debugging

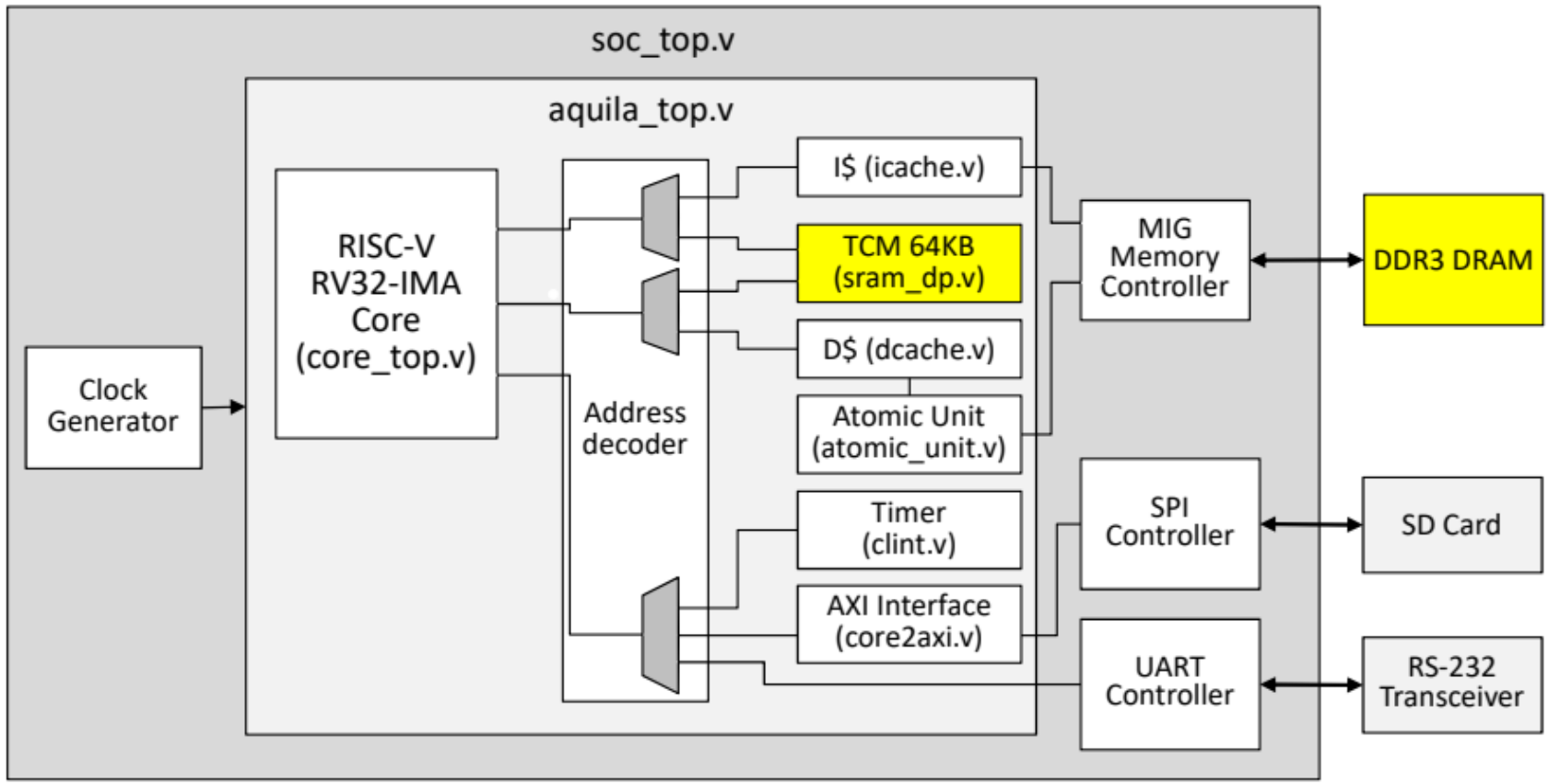


Use Software Evaluate Result

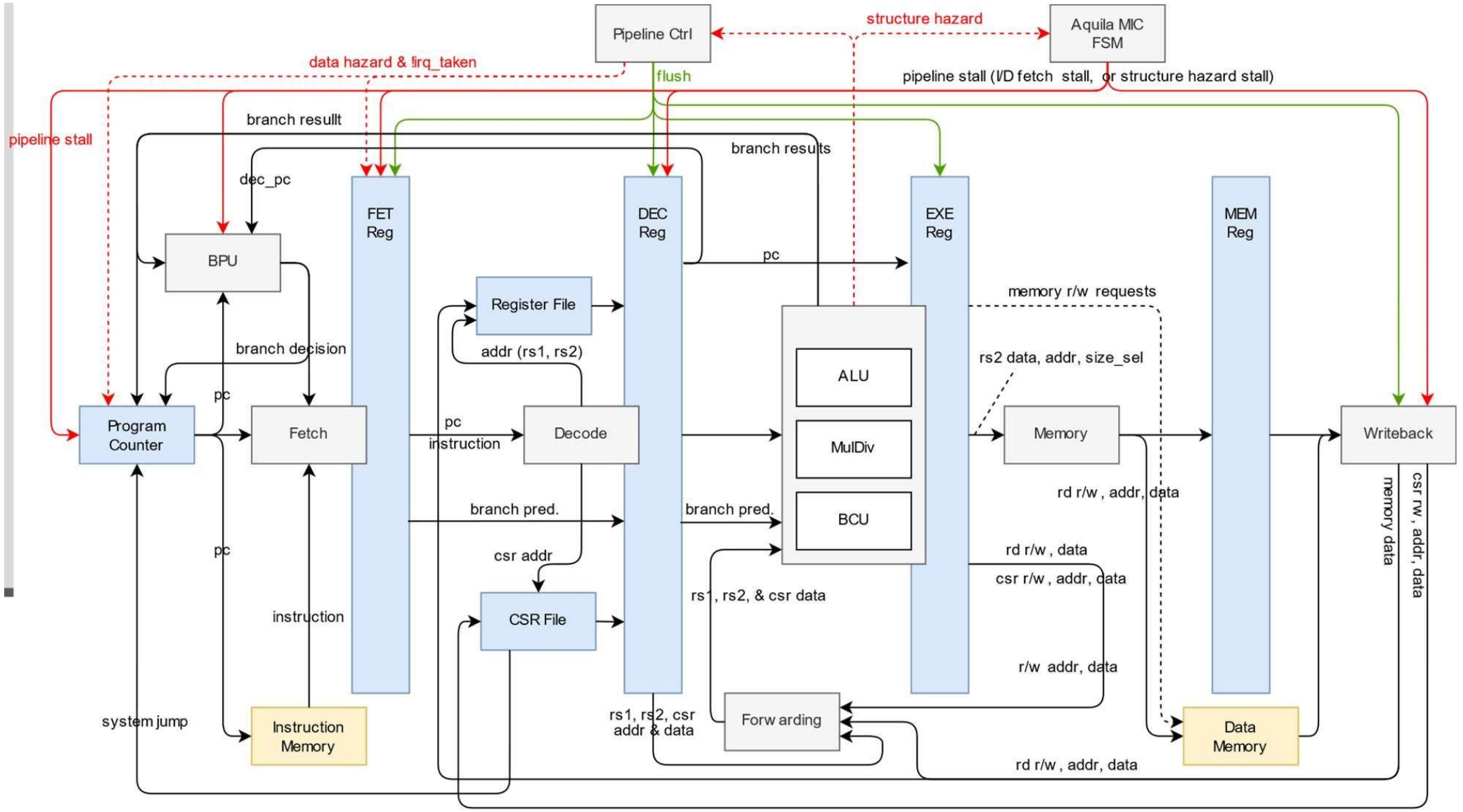
Outline

- Personal Project
  - Transitioning neural network computations from software to hardware
    - Introduction
    - Goal
    - System Architecture (32-bits RISC-V Architecture)
    - Dcache Architecture
    - Original Neural Network Evaluate C code
    - Neural Network Computation Implemented in Hardware
    - Result

System Architecture



Aquila Core


Pipeline Ctrl
structure hazard
Aquila MIC FSM
data hazard & !irq_taken
flush
pipeline stall (I/D fetch stall, or structure hazard stall)
pipeline stall
branch resullt
branch results
dec_pc
FET Reg
DEC Reg
EXE Reg
MEM Reg
BPU
pc
Register File
memory r/w requests
branch decision
addr (rs1, rs2)
ALU
rs2 data, addr, size_sel
pc
Program Counter
Fetch
pc
instruction
Decode
MulDiv
Memory
Writeback
branch pred.
branch pred.
BCU
rd r/w , addr, data
memory data
csr rw , addr, data
pc
csr addr
rd r/w , data
csr r/w , addr, data
rs1, rs2, & csr data
CSR File
instruction
r/w addr, data
system jump
Instruction Memory
rs1, rs2, csr addr & data
Forw arding
Data Memory
rd r/w , addr, data

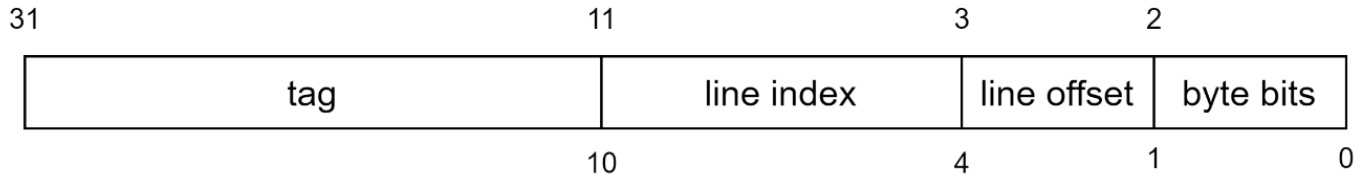
Outline

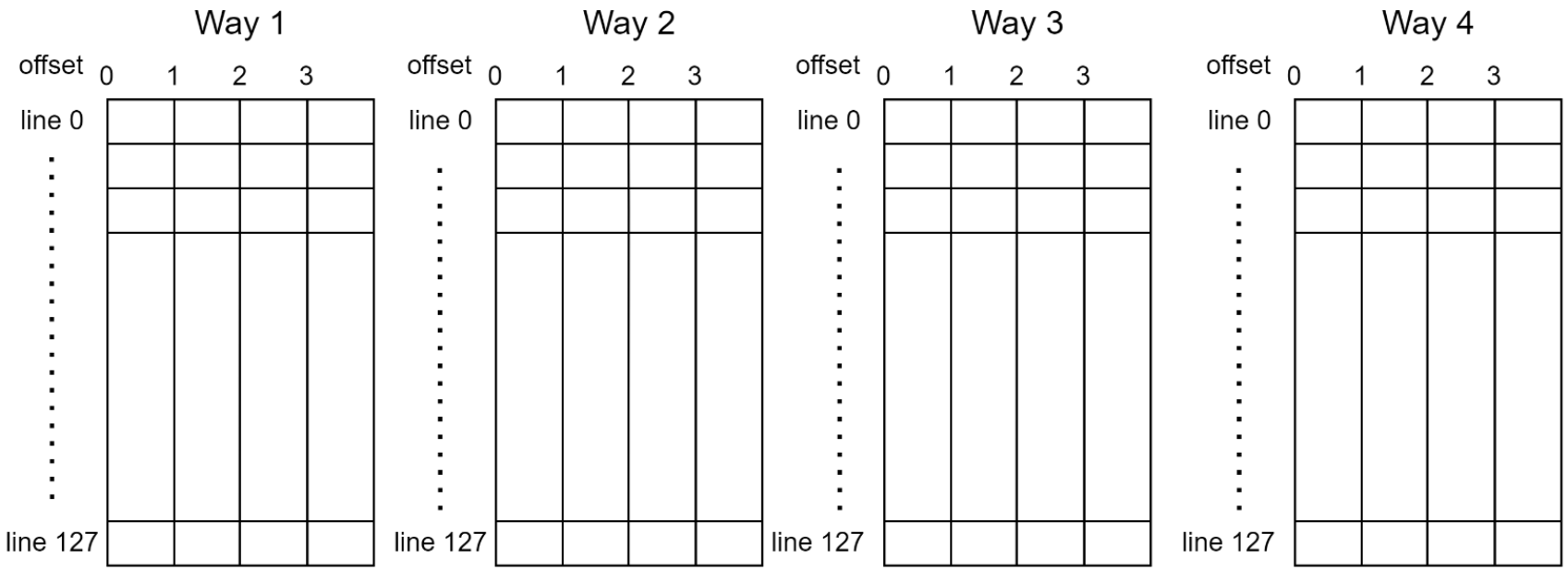
- Personal Project
  - Transitioning neural network computations from software to hardware
    - Introduction
    - Goal
    - System Architecture (32-bits RISC-V Architecture)
    - D Cache Architecture
    - Original Neural Network Evaluate C code
    - Neural Network Computation Implemented in Hardware
    - Result

Cache Types

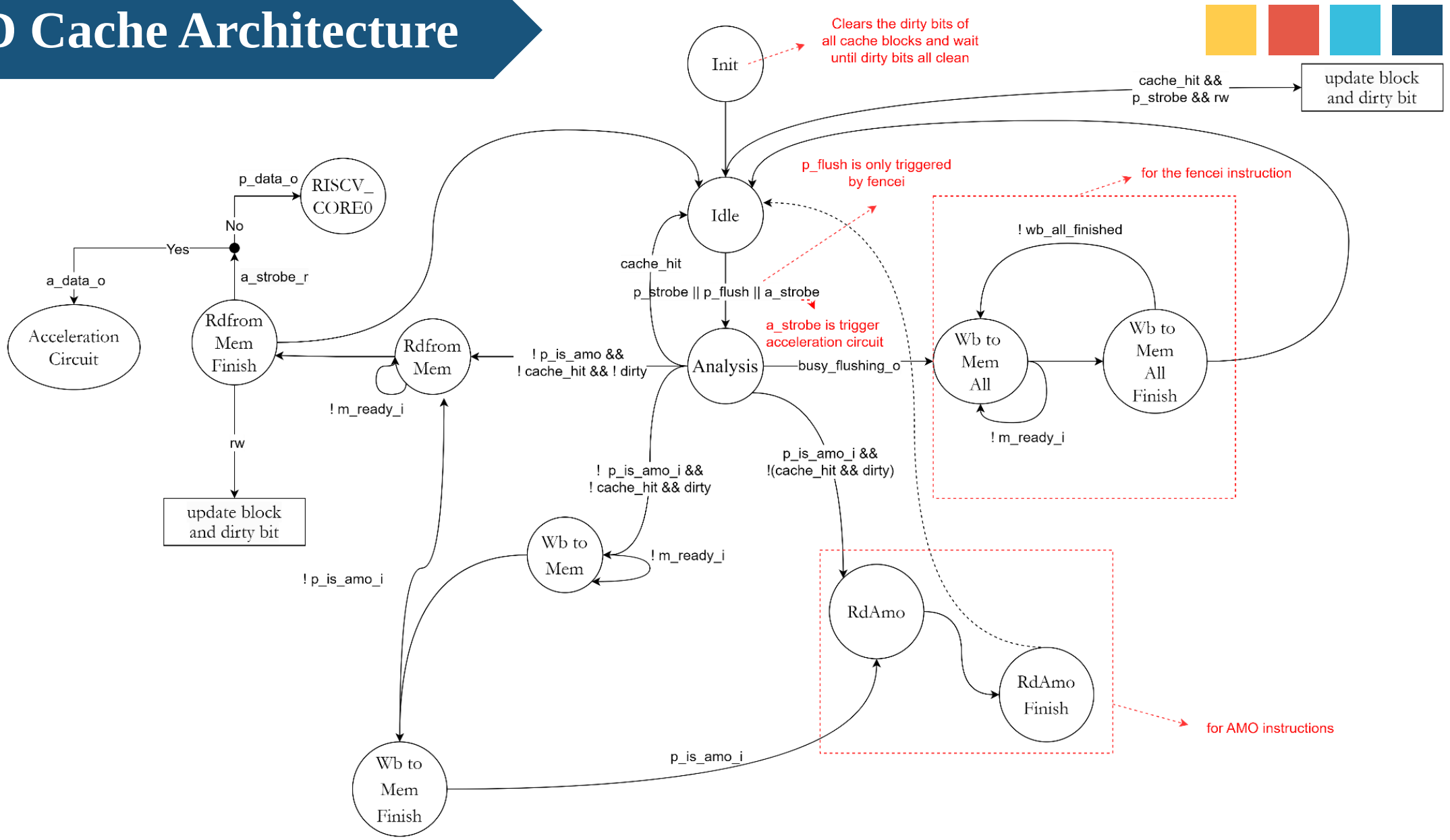
32-bits RISC-V Core
4-way associative cache
Cache Size = 8KB
line size = 128bits
# of line is 128

Format





D Cache Architecture



Outline

- Personal Project
  - Transitioning neural network computations from software to hardware
    - Introduction
    - Goal
    - System Architecture (32-bits RISC-V Architecture)
    - D Cache Architecture
    - Original Neural Network Evaluate C code
    - Neural Network Computation Implemented in Hardware
    - Result

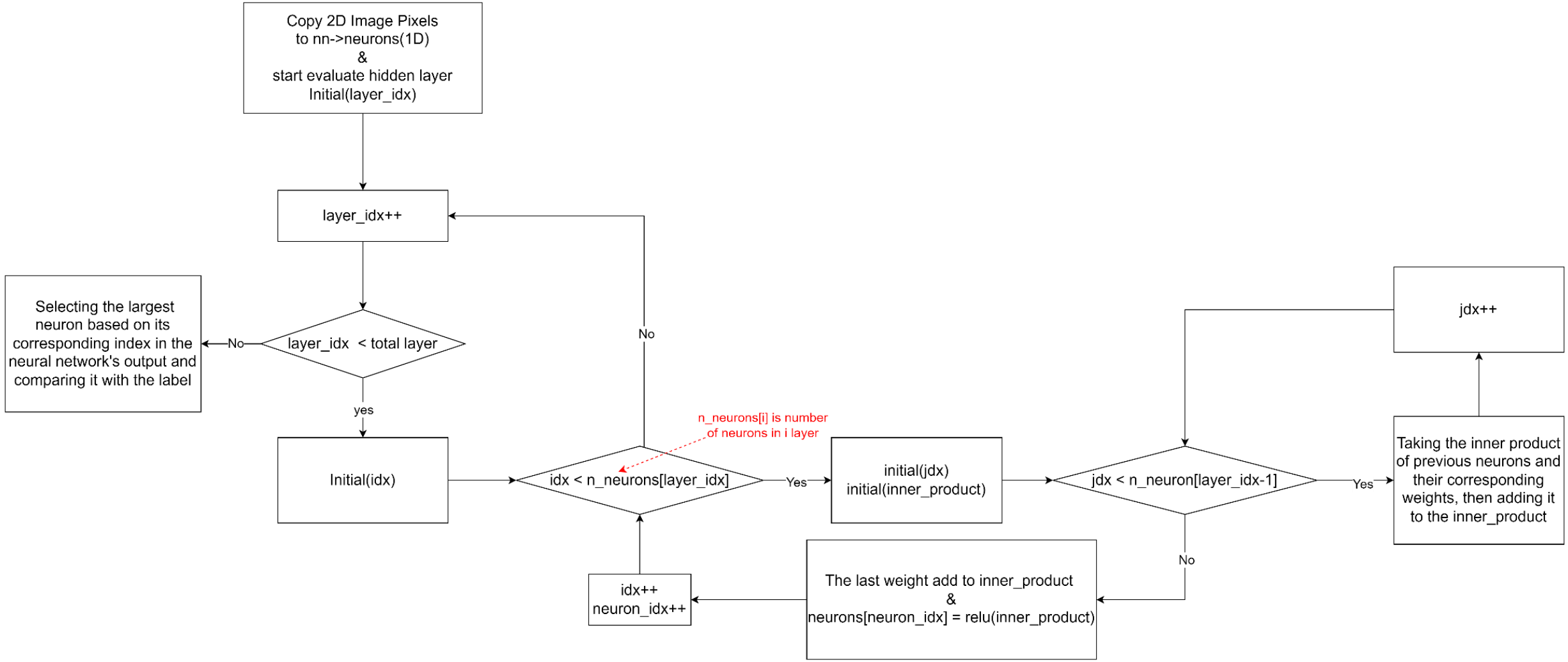
Neural Network Struct

```c
typedef struct __NeuroNet
{
    float *neurons;             // Array that stores all the neuron values.
    float *weights;             // Array that store all the weights & biases.

    float **previous_neurons;   // Pointers to the previous-layer neurons.
    float **forward_weights;    // Pointers to the weights & bias.

    int n_neurons[MAX_LAYERS];  // The # of neurons in each layer.
    int total_layers;           // The total # of layers.
    int total_neurons;          // The total # of neurons.
    int total_weights;          // The total # of weights.
    float *output;              // Pointer to the neurons of the output layer.
} NeuroNet;
```

Neural Network Eval C Code

```c
int neuronet_eval(NeuroNet *nn, float *images)
{
    float inner_product, max;
    float *p_neuron, *p_weight;
    int idx, layer_idx, neuron_idx, max_idx;

    // Forward computations
    neuron_idx = nn->n_neurons[0];
    for (layer_idx = 1; layer_idx < nn->total_layers; layer_idx++)
    {
        for (idx = 0; idx < nn->n_neurons[layer_idx]; idx++, neuron_idx++)
        {
            // 'p_weight' points to the first forward weight of a layer.
            p_weight = nn->forward_weights[neuron_idx];
            inner_product = 0.0;

            // Loop over all forward-connected neural links.
            p_neuron = nn->previous_neurons[neuron_idx];
            for (int jdx = 0; jdx < nn->n_neurons[layer_idx-1]; jdx++)
            {
                inner_product += (*p_neuron++) * (*p_weight++);
            }

            inner_product += *(p_weight); // The last weight of a neuron is the bias.
            nn->neurons[neuron_idx] = relu(inner_product); /* relu(x) = x < 0.0 ? 0.0 : x; */
        }
    }

    // Return the index to the maximal neuron value of the output layer.
    max = -1.0, max_idx = 0;
    for (idx = 0; idx < nn->n_neurons[nn->total_layers-1]; idx++)
    {
        if (max < nn->output[idx])
        {
            max_idx = idx;
            max = nn->output[idx];
        }
    }

    return max_idx;
}
```

Neural Network Eval UML



Outline

- Personal Project
  - Transitioning neural network computations from software to hardware
    - Introduction
    - Goal
    - System Architecture (32-bits RISC-V Architecture)
    - D Cache Architecture
    - Original Neural Network Evaluate C code
    - Neural Network Computation Implemented in Hardware
    - Result

Set Trigger Function to Linker script

- Set code ram origin address and length
- Set data ram origin address and length
- Place the trigger function at the beginning of code ram
- Set Entry point
  - Entry point will execute main function

```
__stack_size = 0x800;
__heap_size  = 0x200000;

MEMORY
{
    code_ram   (rx!rw) : ORIGIN = 0x80001000, LENGTH = 0x10000
    data_ram   (rw!x)  : ORIGIN = 0x80011000, LENGTH = 0x310000
}

ENTRY(crt0)


SECTIONS
{
    .my_section :
    {
        *(.acceleration_circuit)
    } > code_ram

    .text :
    {
        *(.text*)
    } > code_ram

    .data :
    {
        *(.data)
        *(.bss)
        *(.rodata*)
    } > data_ram

    .heap : ALIGN(0x10)
    {
        __heap_start = .;
        . += __heap_size;
    } > data_ram

    .stack : ALIGN(0x10)
    {
        . += __stack_size;
        __stack_top = .;
        __freertos_irq_stack_top = .;
    } > data_ram
}
```

Trigger Function Address

Set Trigger Function in Software

- Set trigger function in software

```
// trigger function
volatile int __attribute__
((section (".acceleration_circuit")))
neuronet_eval_hardware(NeuroNet *nn, float *images);
```

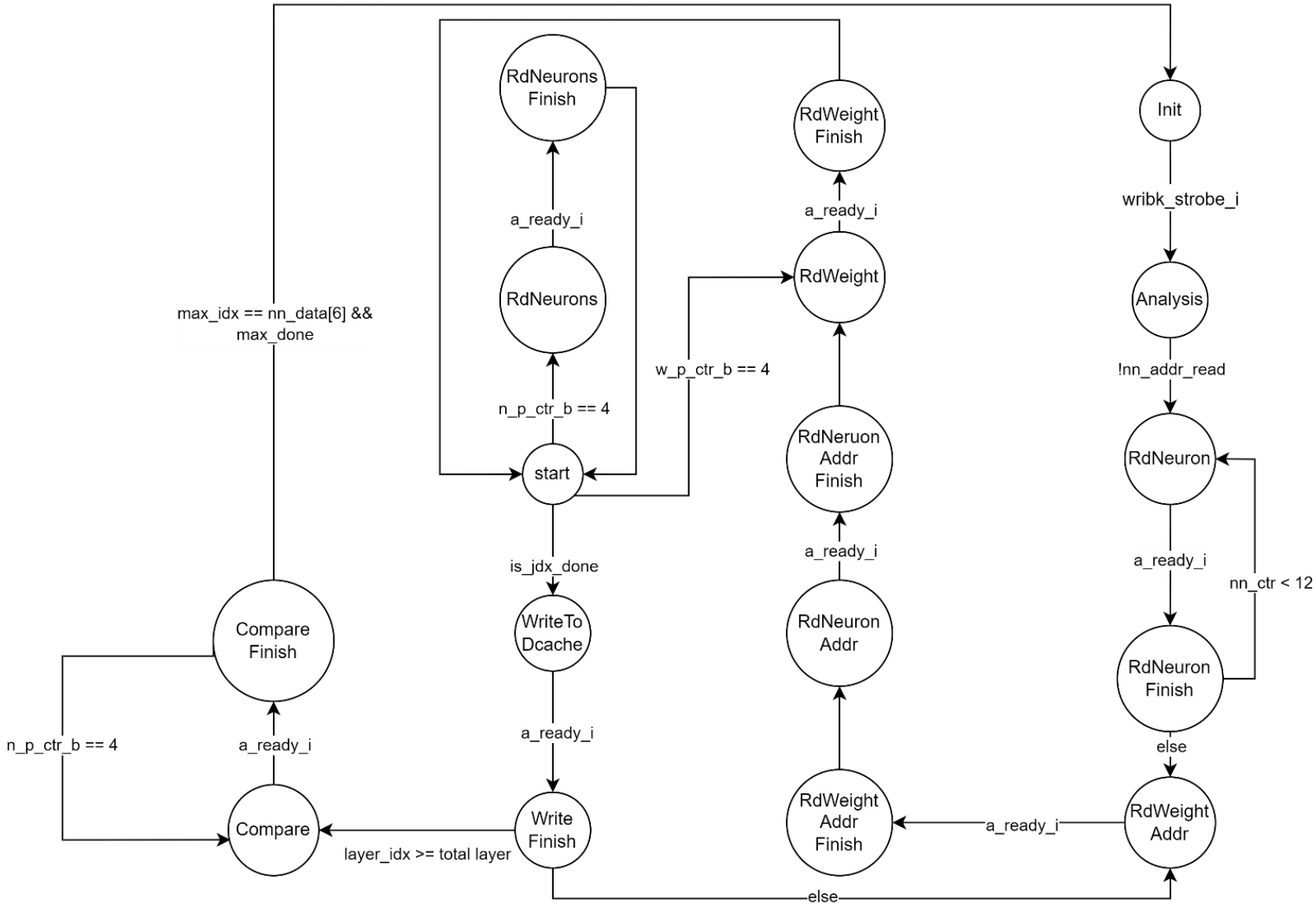
- Use assembly code to set end point in the register file
- Write assembly code to initialize the address of the struct NeuronNet and save it to a register file
- Busy Waiting until acceleration circuit done

```
int neuronet_eval_hardware(NeuroNet *nn, float *images){
    /*
    s0 register file addr  is 8 use
    s1 register file addr  is 9
    a0 register file addr  is 10 use
    a1 register file addr  is 11
    a2 register file addr  is 12 use
    a3 register file addr  is 13
    */
    int max_idx;

    /* Let a2(12) equal zero */
    asm volatile ("addi a2, x0, 0");
    /* put nn address in a1(11) */
    asm volatile ("mv a1, %0": : "r" (nn)); /* address for nn */

    /* if over then is_over = true(1) also means a2(12) is 1*/
    do
        asm volatile ("mv %0, a2": "=r" (max_idx) );
    while(!max_idx);

    return max_idx-1;
}
```

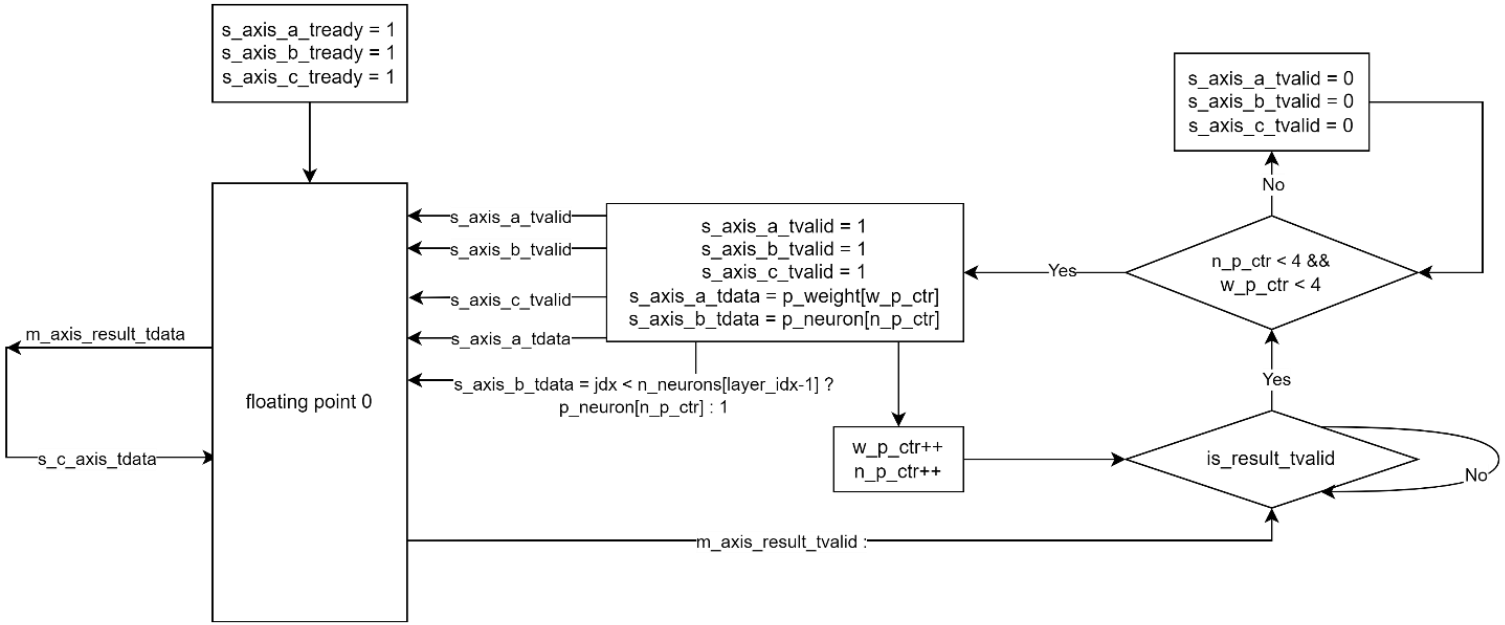
End point a2

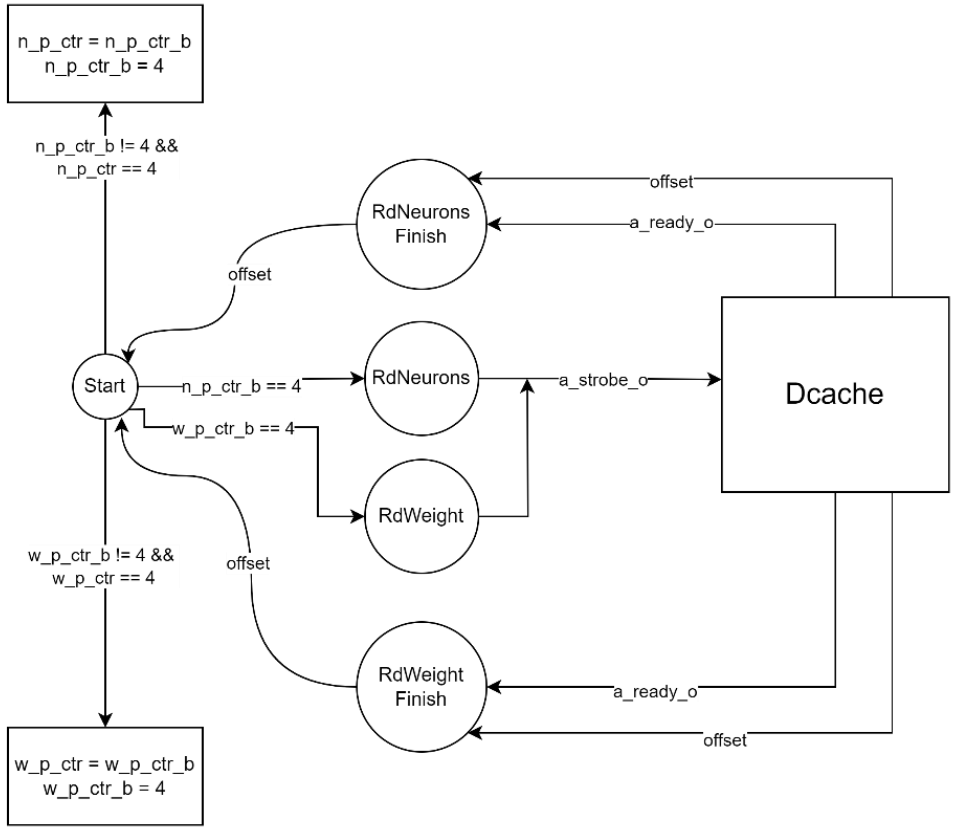
NeruonNet address save to a1

Acceleration Circuit Architecture

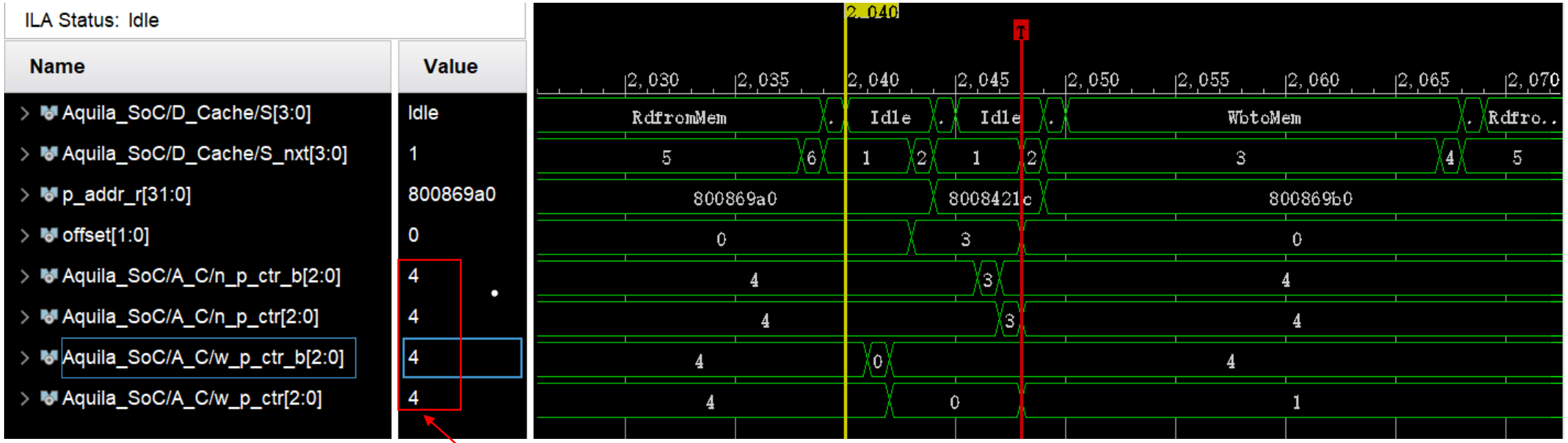


Synchronize Between Dcache and evaluate





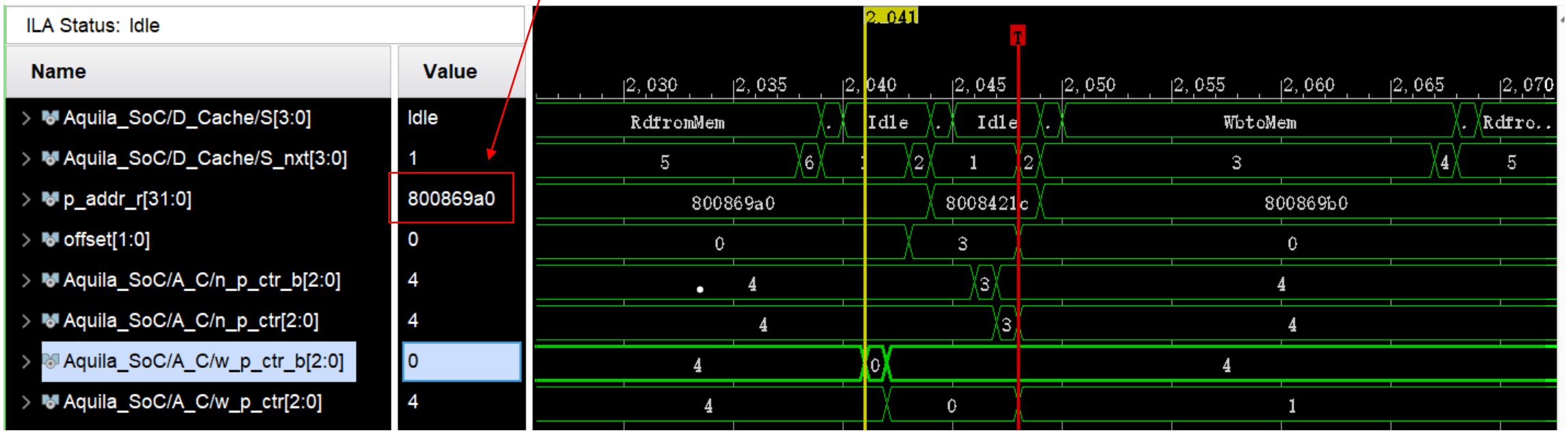
Waveform



Initial counter is all equal four

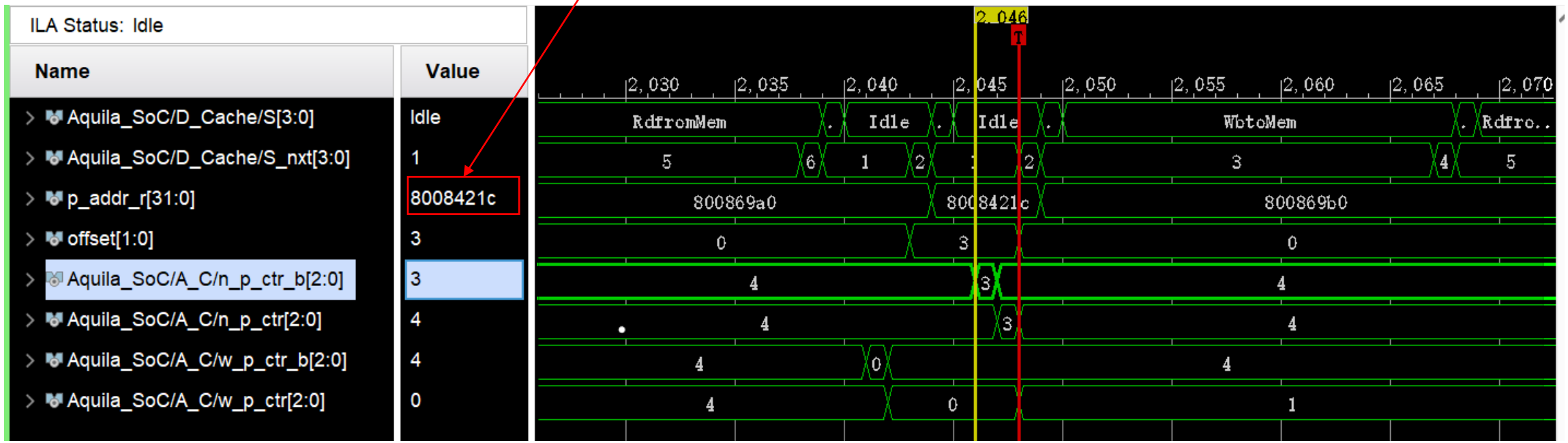
Waveform

Weight Initial address last value is 0 so offset is 0

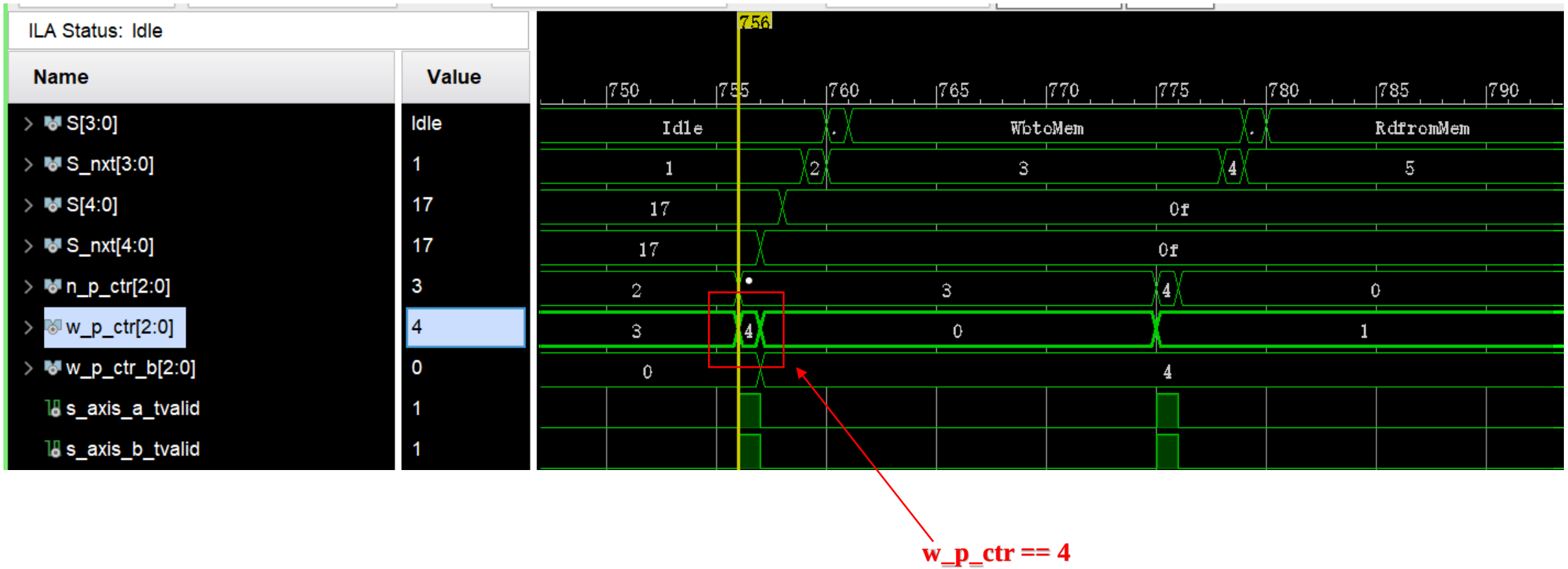


Waveform

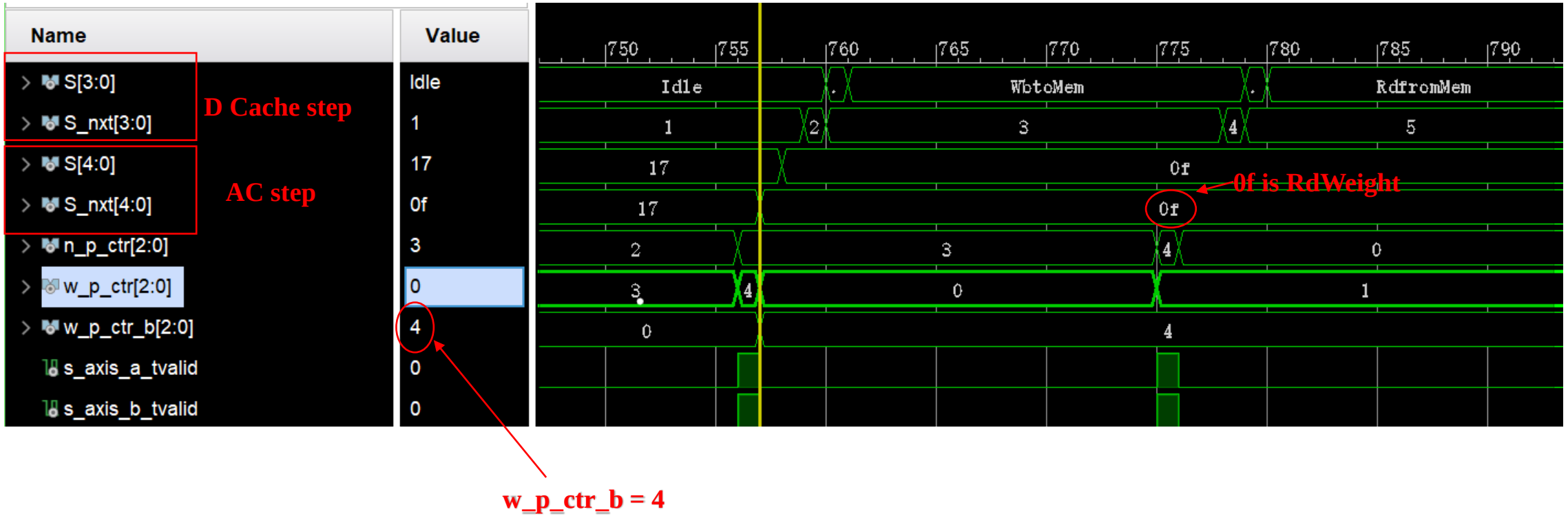
Neurons Initial address last value is c so offset is 3



Waveform

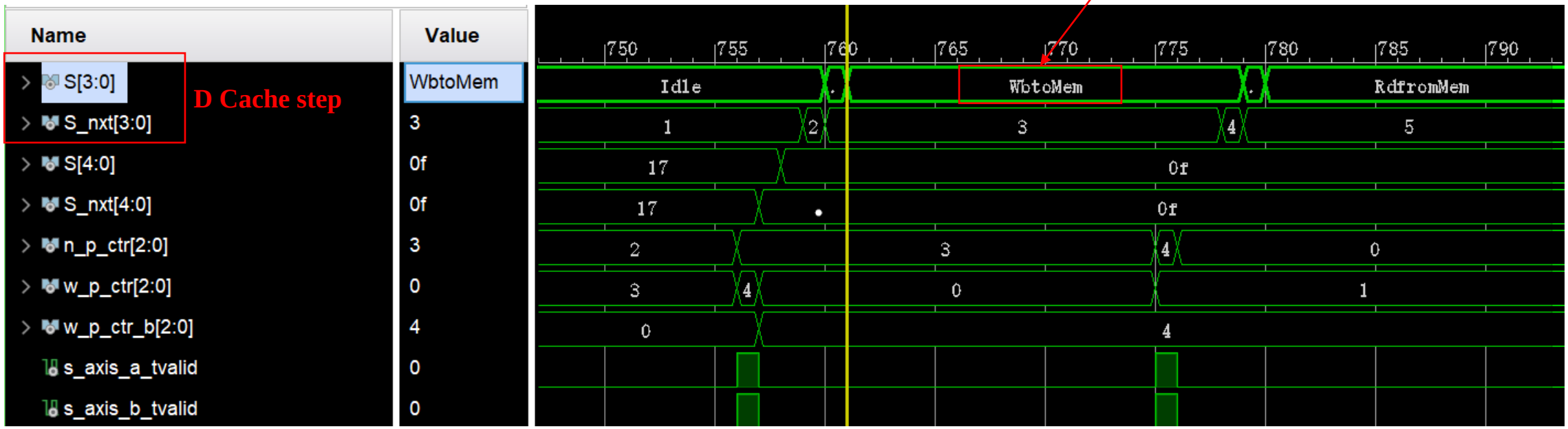

ILA Status: Idle
Name
Value
S[3:0]
Idle
S_nxt[3:0]
1
S[4:0]
17
S_nxt[4:0]
17
n_p_ctr[2:0]
3
w_p_ctr[2:0]
4
w_p_ctr_b[2:0]
0
s_axis_a_tvalid
1
s_axis_b_tvalid
1
756
750
755
760
765
770
775
780
785
790
Idle
WbtoMem
RdfromMem
w_p_ctr == 4

Waveform


Name
Value
S[3:0]
S_nxt[3:0]
S[4:0]
S_nxt[4:0]
n_p_ctr[2:0]
w_p_ctr[2:0]
w_p_ctr_b[2:0]
s_axis_a_tvalid
s_axis_b_tvalid
Idle
1
17
0f
3
0
4
0
0
D Cache step
AC step
750
755
760
765
770
775
780
785
790
Idle
WbtoMem
RdfromMem
0f is RdWeight
w_p_ctr_b = 4

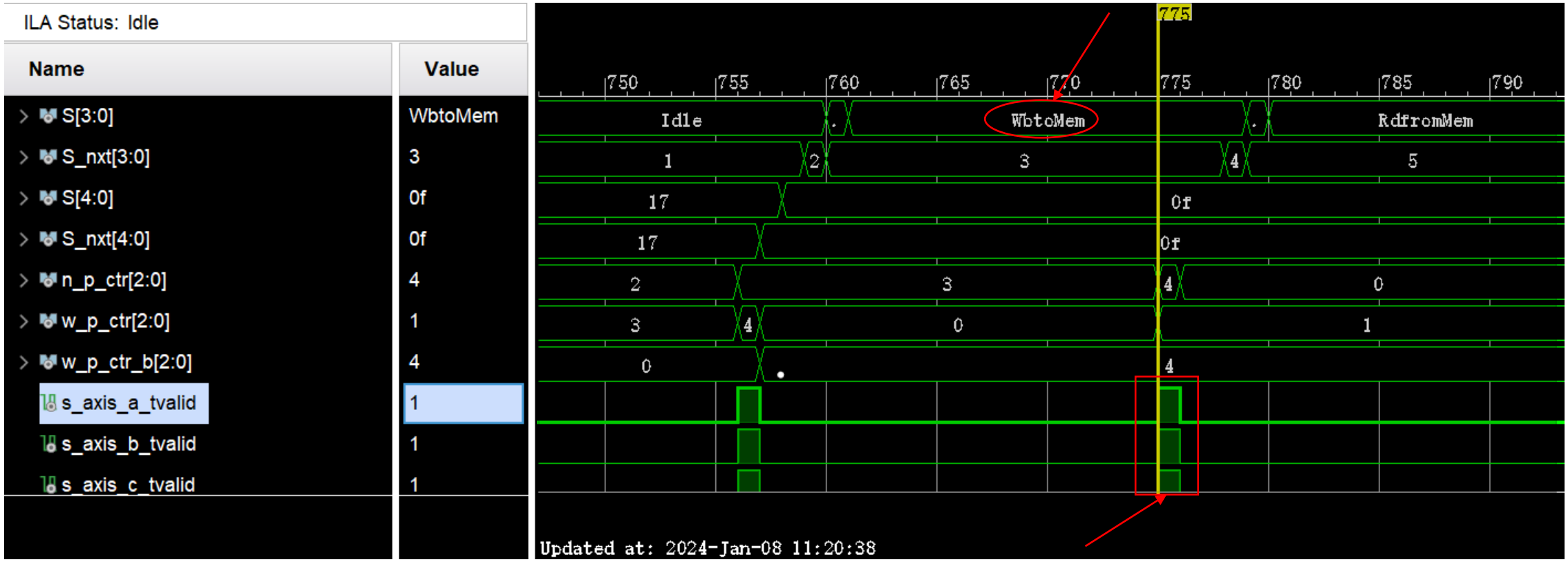
Waveform

D Cache is starting read weight data



Waveform

Dcache is reading



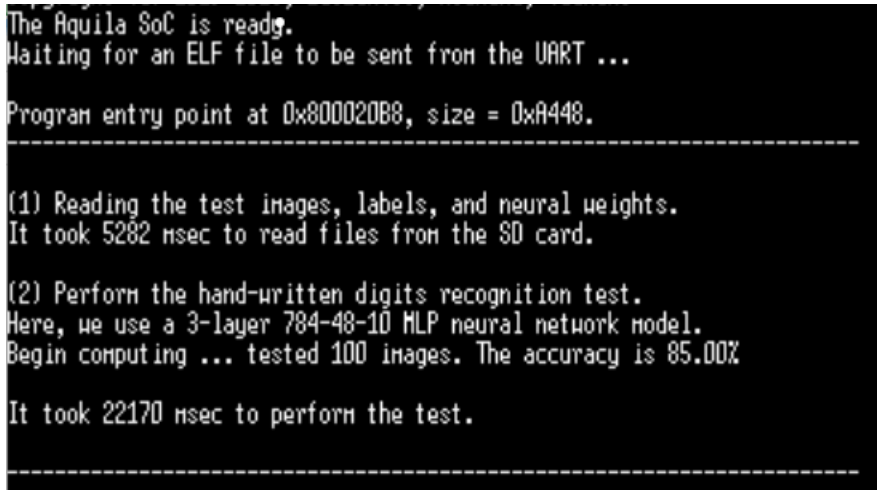
Because w_p_ctr != 0 && n_p_ctr != 0
so the evaluate can be made

Outline

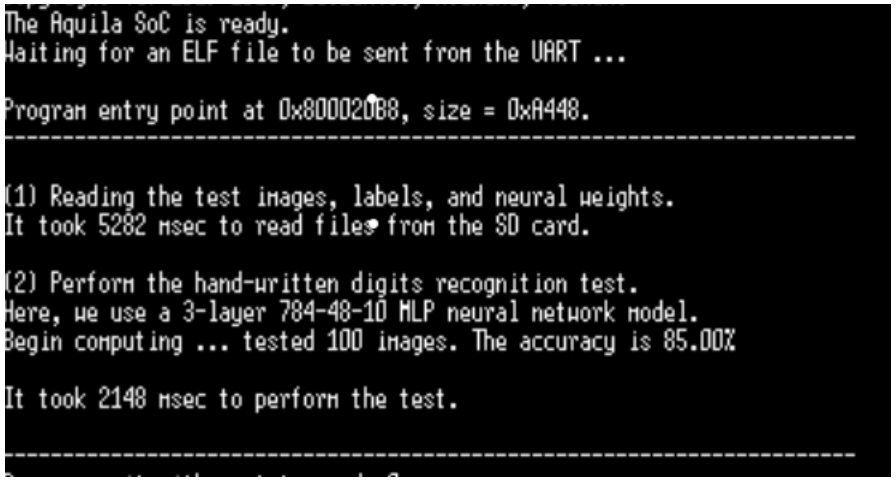
- Personal Project
  - Transitioning neural network computations from software to hardware
    - Introduction
    - Goal
    - System Architecture (32-bits RISC-V Architecture)
    - D Cache Architecture
    - Original Neural Network Evaluate C code
    - Neural Network Computation Implemented in Hardware
    - Result

Result

Use software



Use hardware



- Ten times difference

Thank you for your time

Bi-Fan Liu