ECE 551
Project Specification

Spring ’18

Simple Neural Network

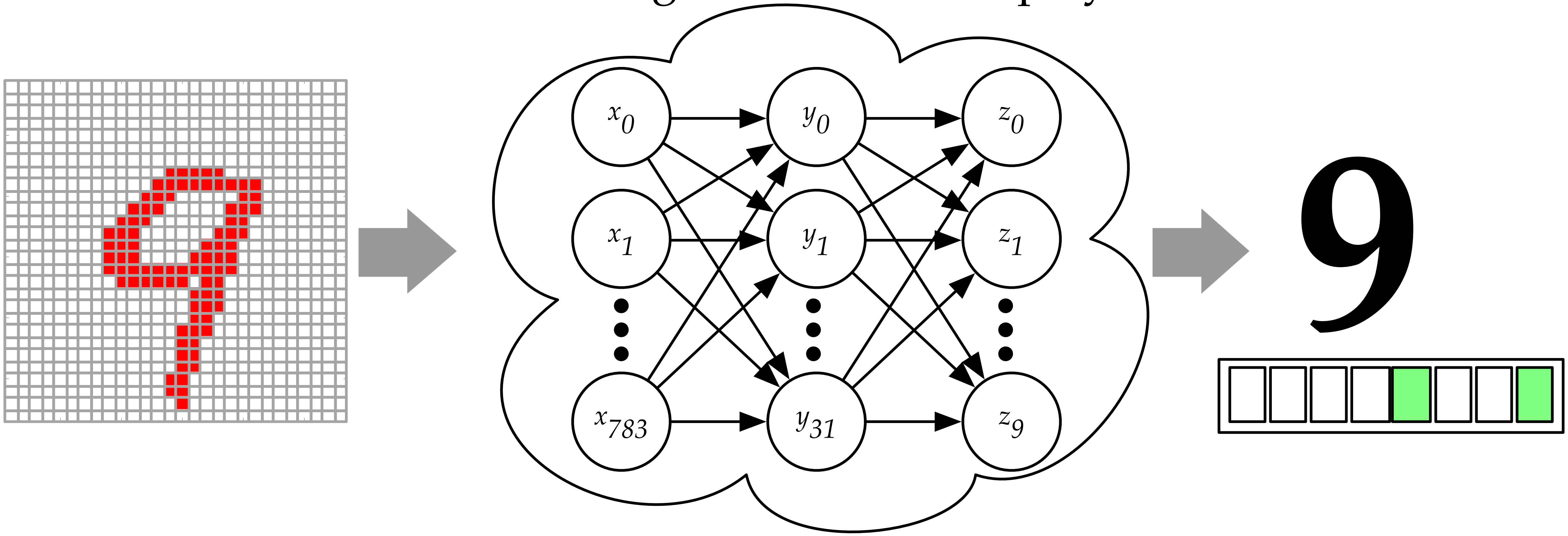
Revision 4

4/20/2018

Overview

Simple Neural Network

- Simple artificial neural network (ANN) for hand-written digit recognition (inference only)
- Input: 28x28 bitmap of hand-written digits
  - Transmitted from PC through UART
- Output: recognized digit, one of 0–9
  - Transmitted to PC through UART and displayed on LED



Simple Neural Network

- 1 input layer
  - 784 nodes
  - Each node represents one bit of input bitmaps
- 1 output layer
  - 10 nodes
  - Each node represents probability that the input is 0, 1, …, 9, respectively
- 1 hidden layer
  - 32 nodes
  - Connects input layer and output layer

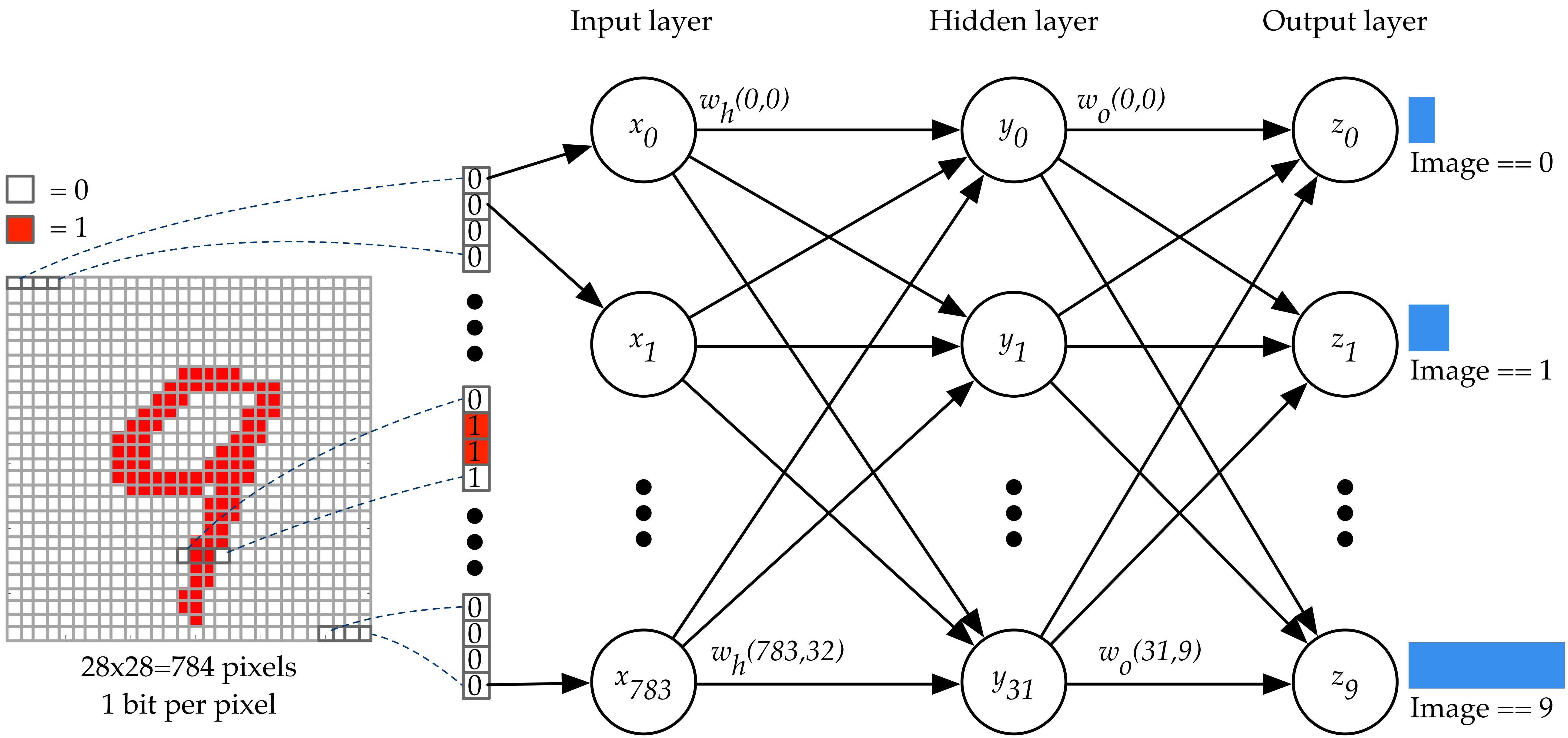
SNN Input

- 28x28 binary bitmap of a digit
- 28x28 = 784 bits = 98 bytes
- Ten digits (0–9) will be given
  - Some digits are correctly recognized, some others are incorrectly recognized, but the results are deterministic
  - Recognization results will be given
- Test will be done for thousands of digits
- Files will be provided
  - Files for initializing ROM in Modelsim (soon)
  - Files for transmitting from RealTerm (later)

SNN Output

- Recognized digit, one of 0, 1, …, 9
- Can be correct or incorrect, but the results should be the same as reference
- 1 byte in ASCII (e.g., ‘5’ is 8’h35 not 8’h05)
- Transmit to PC through UART
- Display on LED

Basic Operation



$$x_i = \begin{cases} 0, & \text{if pixel } i \text{ is empty} \\ 1, & \text{if pixel } i \text{ is filled} \end{cases} \qquad y_i = f\left(\sum_{k=0}^{783} w_h(i,k)x_k\right) \qquad z_i = f\left(\sum_{k=0}^{31} w_o(i,k)y_k\right)$$

f: activation function

Hidden and Output Unit Operation

$$y_i = f\left(\sum_{k=0}^{783} w_h(i,k)x_k\right)$$

- To calculate each *i*-th hidden unit result y_i,
  - multiply hidden weight $w_h(i, k)$ and input unit value x_k for k = 0, …, 733
  - apply activation function

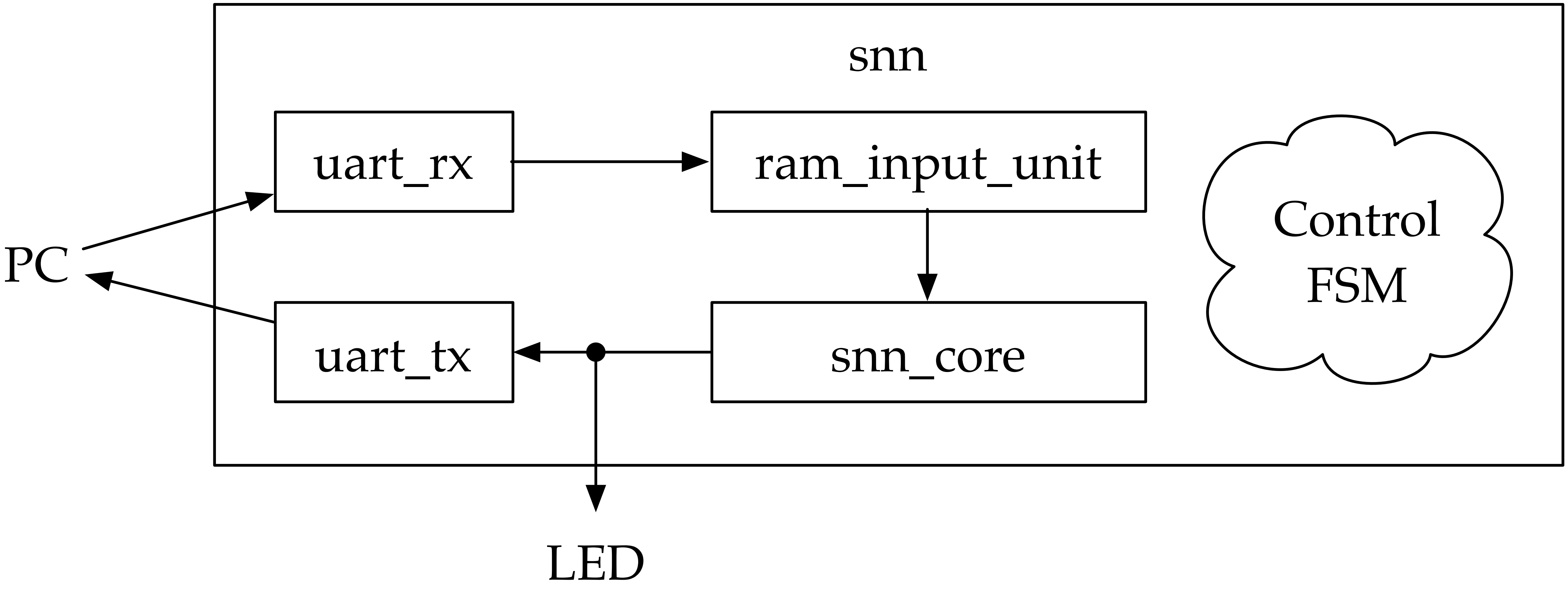
$$z_i = f\left(\sum_{k=0}^{31} w_o(i,k)y_k\right)$$

- To calculate each *i*-th output unit result z_i,
  - multiply output weight $w_o(i, k)$ and hidden unit value y_k for k = 0, …, 31
  - apply activation function

Final Output

- Among ten output unit values z_i, the largest value indicates the most likely digit
- e.g., if z_5 is the largest, the digit is recognized as 5

Top-Level Design



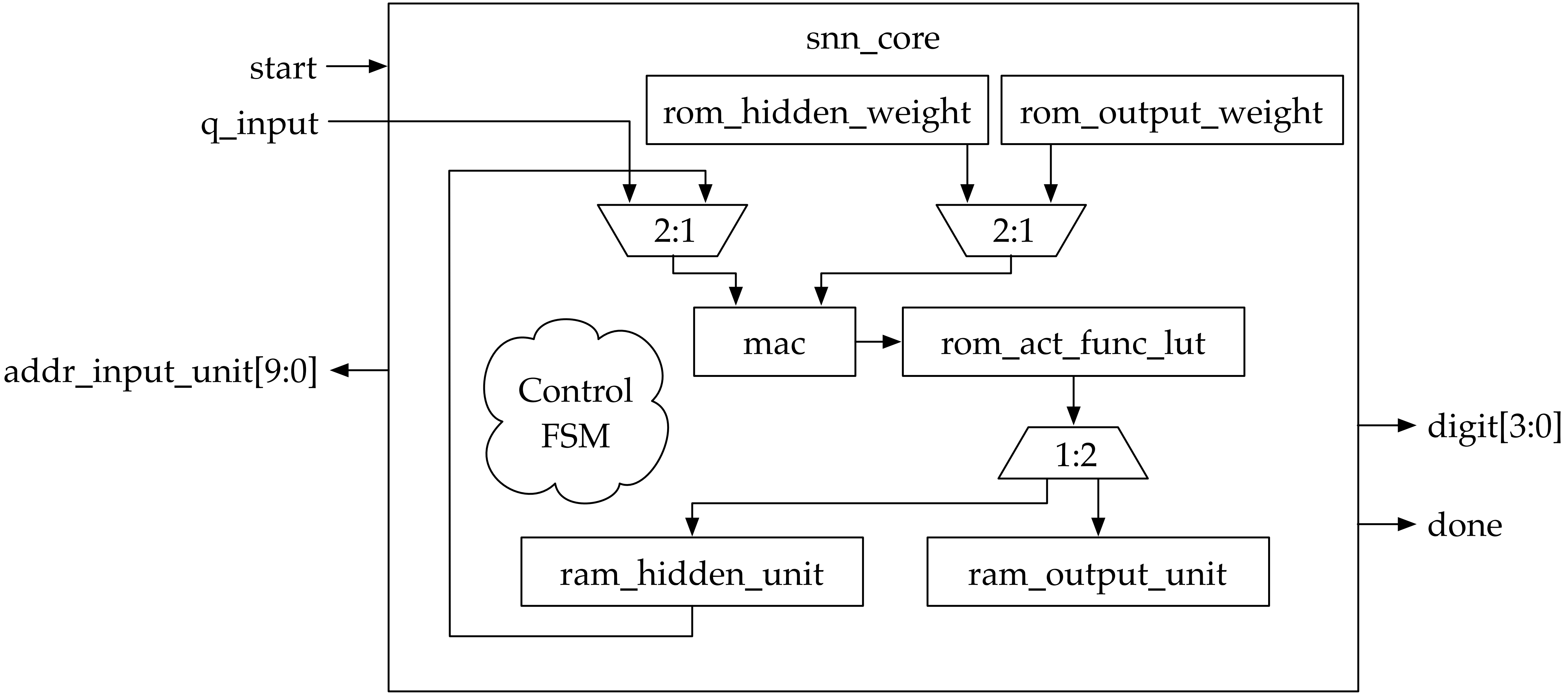
(a)98 byte input is loaded from PC to RAM **ram_input_unit**

(b)**snn_core** reads the input bit by bit and recognizes the digit

(c)2 byte result is sent to PC and displayed on LED

(d)This flow is controlled by Control FSM

SNN Core (snn_core) Design

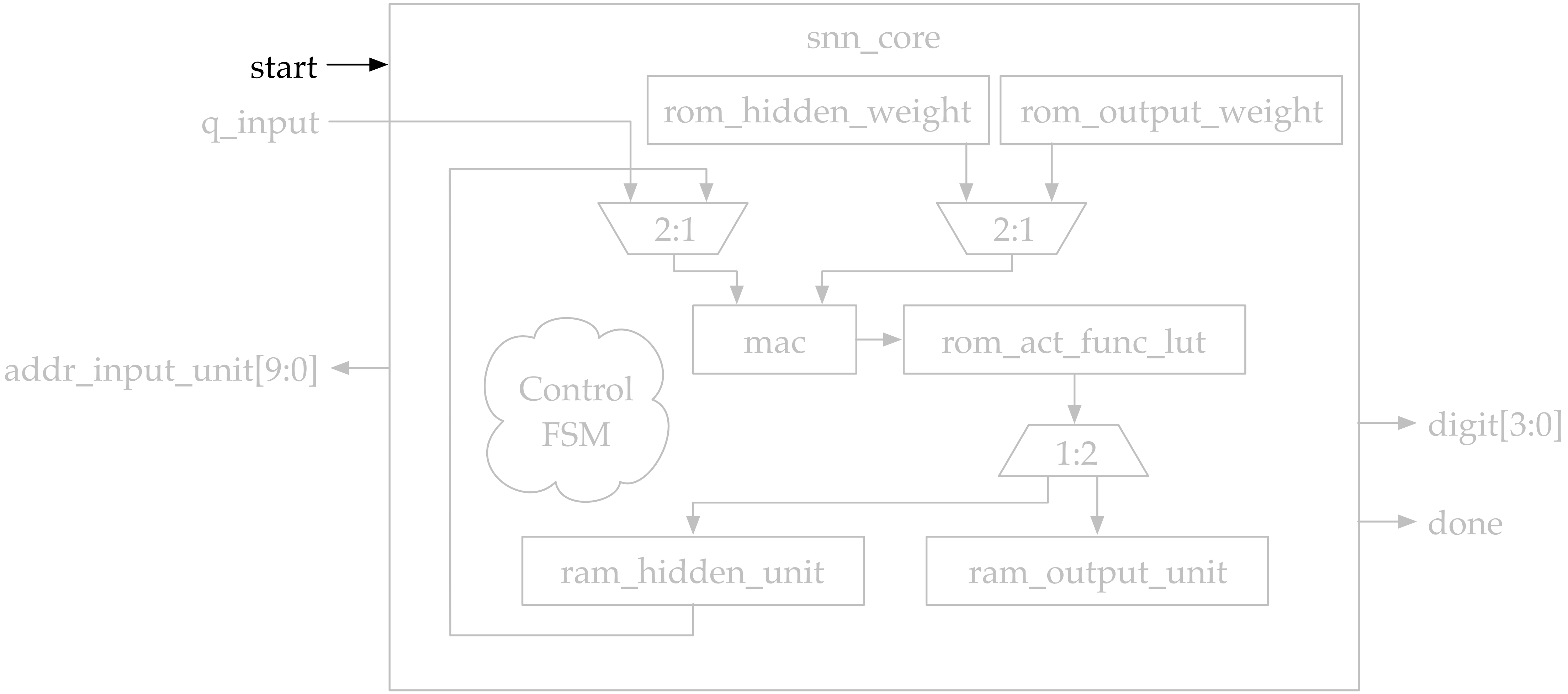


(a)Start when **start** is triggered

(b)Read input bit by bit, changing **addr_input_unit**

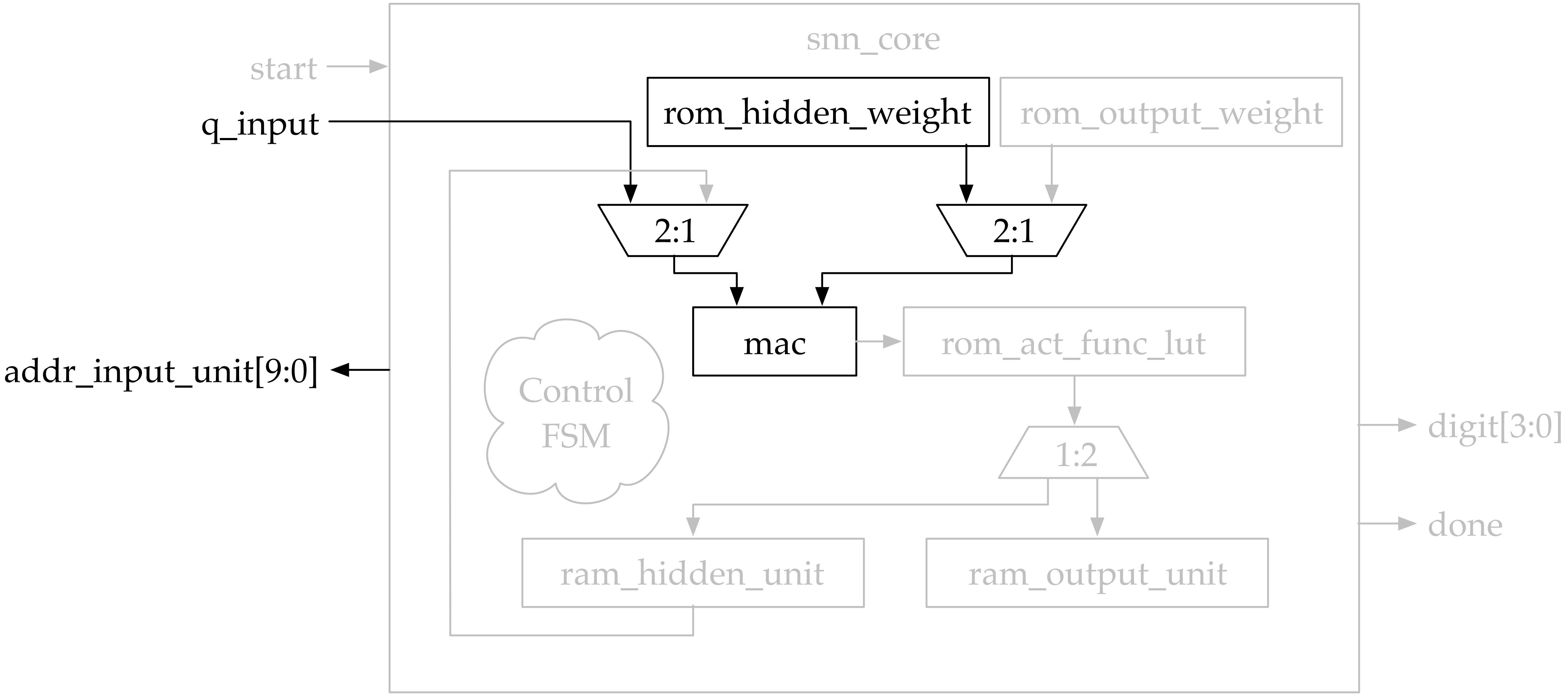
(c)When done, output **digit** and assert **done**

snn_core step 1: Start



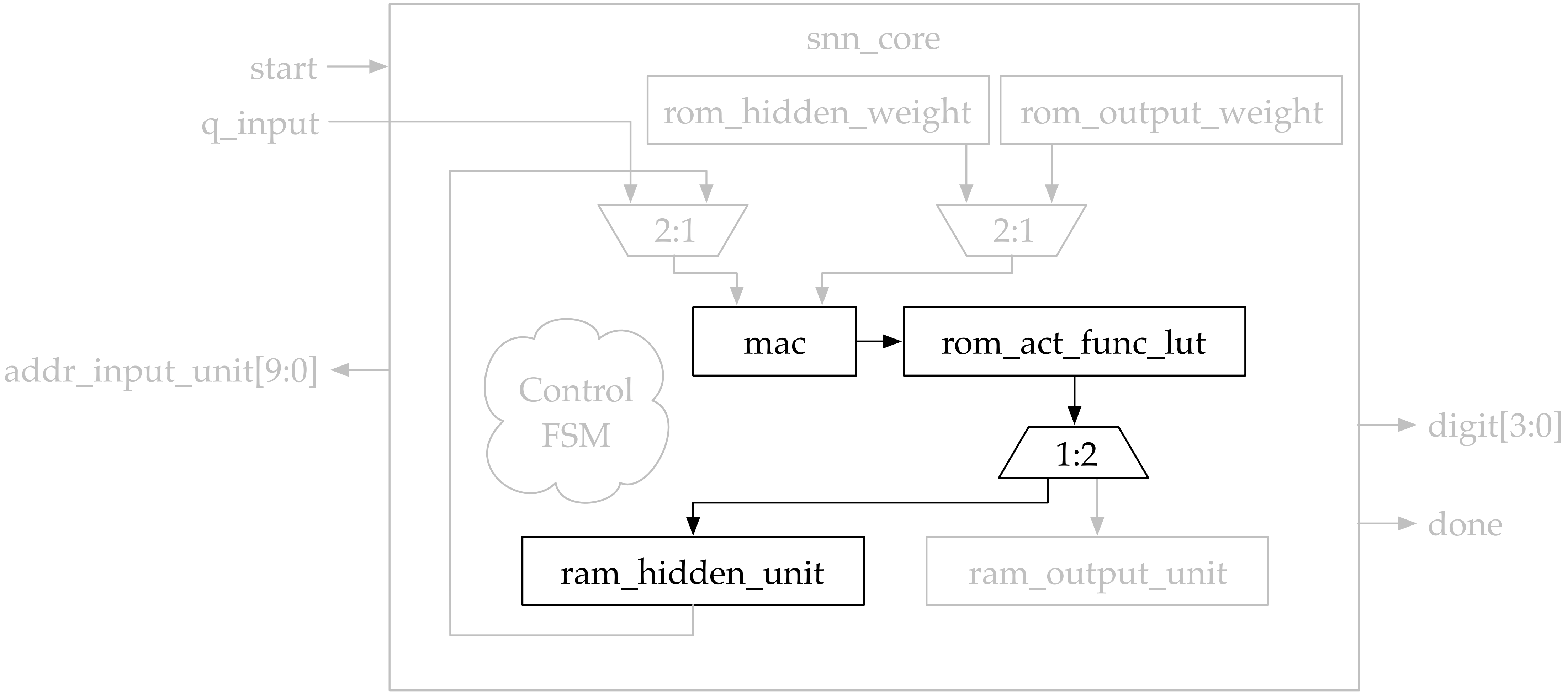
- **start** is asserted by **snn**

snn_core step 2: Mac Input Unit and Hidden Weight



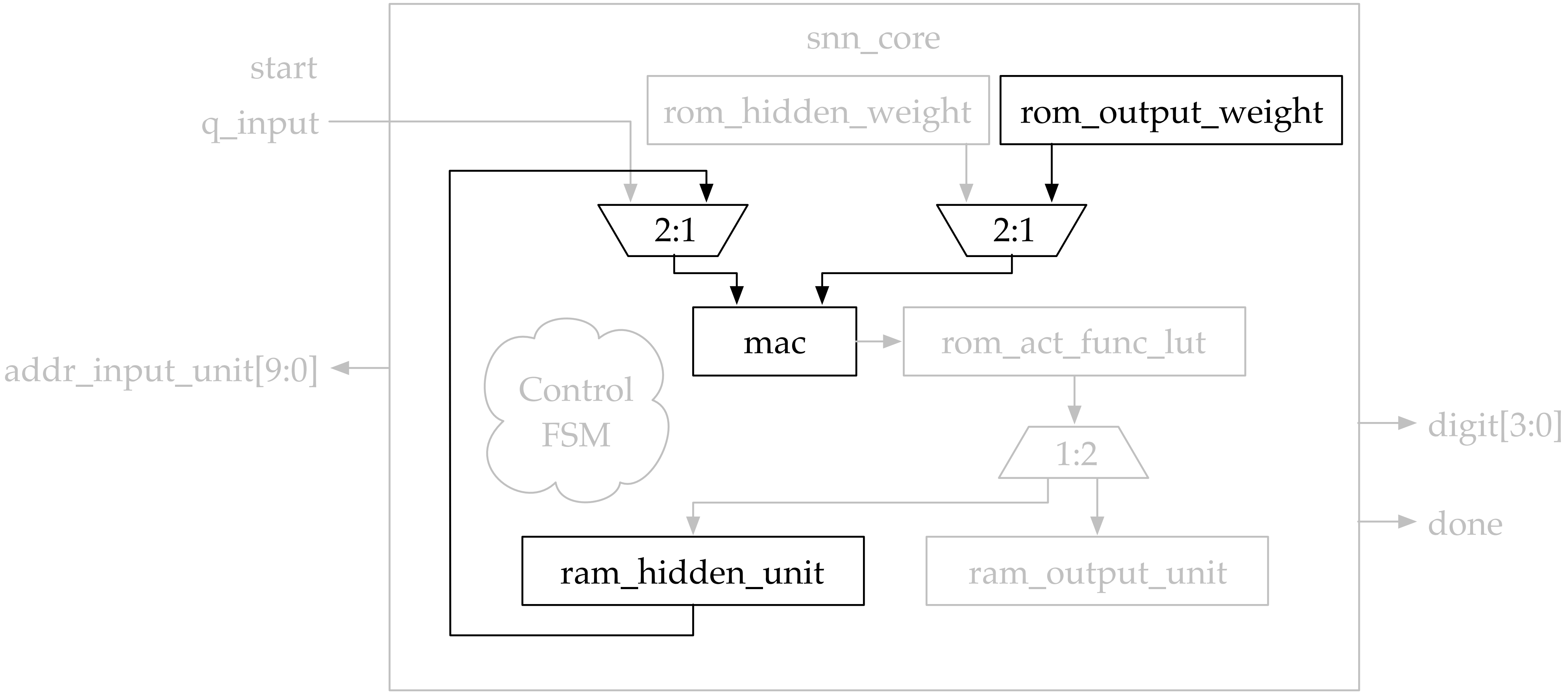
- Read **q_input** from **ram_input_unit** incrementing **addr_input_unit**
- Extend 1-bit **q_input** to 8-bit to make it either 0 (8′b00000000) or 127 (8′b01111111).
- Read **q_weight_hidden** from **ram_hidden_weight** incrementing **addr_hidden_weight**
- Mac results

snn_core step 3: Apply Activation Function



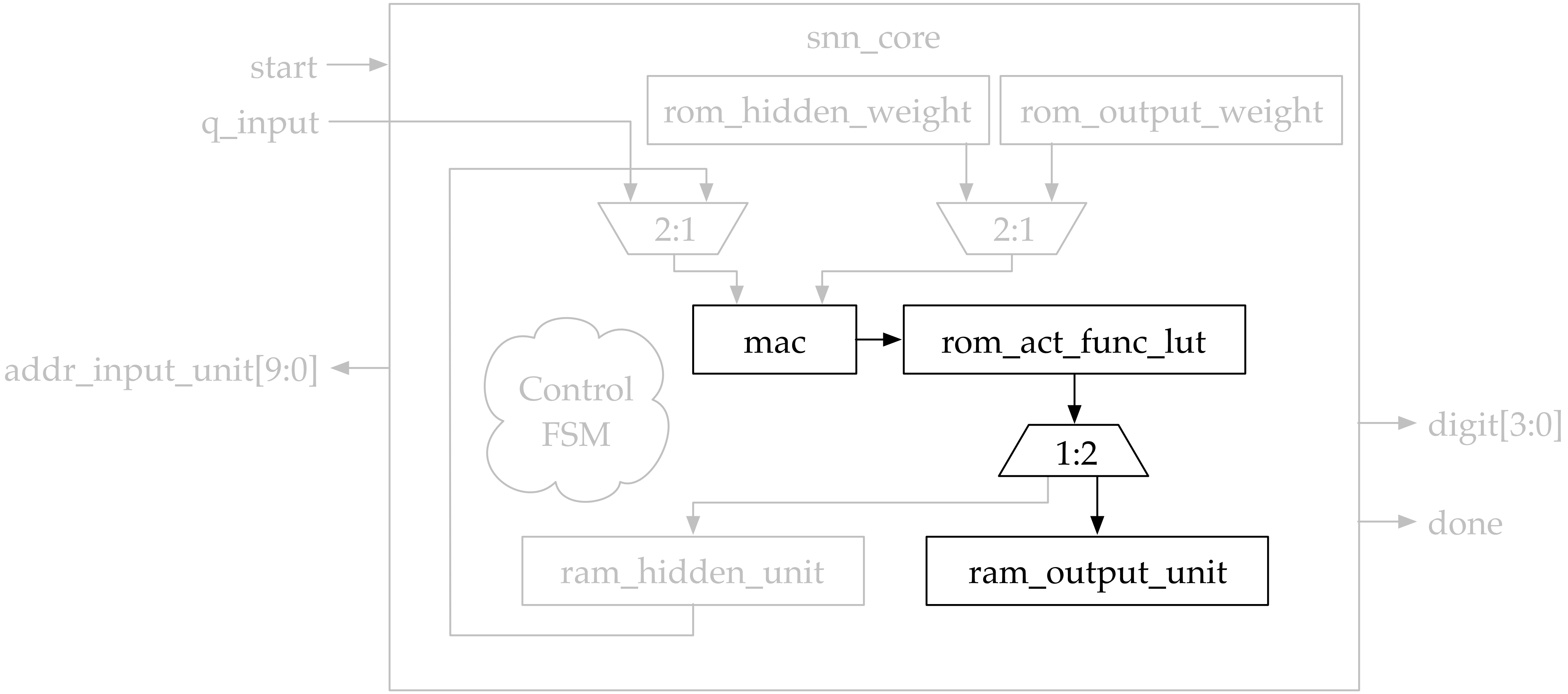
- Read **rom_act_func_lut** using (*rect*(**mac**)+1024) as address and read output (See “Mac Result Rectification” page for *rect*())
- The output is y_k. Write to **d_hidden_unit** port of **ram_hidden_unit.**

snn_core step 5: Mac Hidden Unit and Output Weight



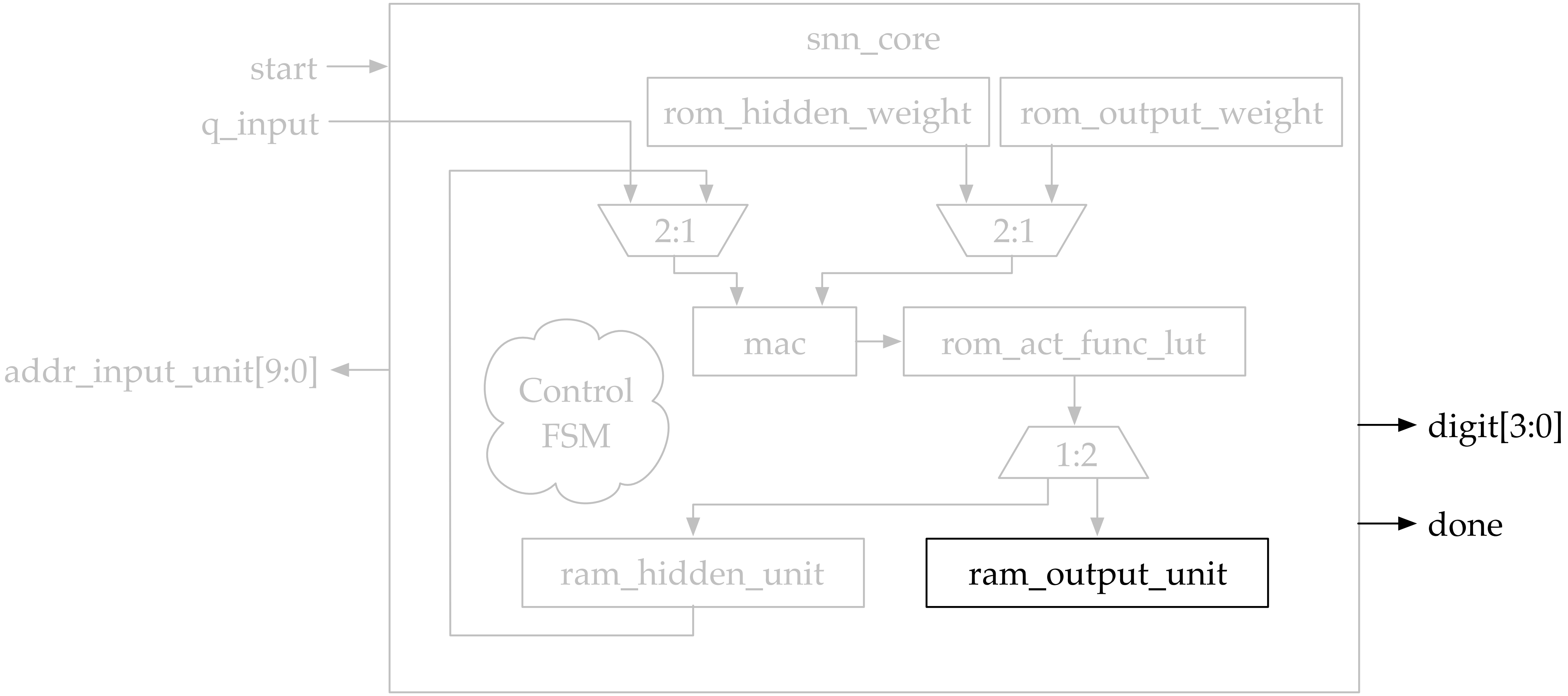
- Read **q_hidden_unit** from **ram_hidden_unit** incrementing **addr_hidden_unit**
- Read **q_weight_output** from **ram_output_weight** incrementing **addr_output_weight**
- Mac results

snn_core step 6: Apply Activation Function



- Read **rom_act_func_lut** using (*rect*(**mac**)+1024) as address and read output (See “Mac Result Rectification” page for *rect*())
- The output is z_k. Write to **d_output_unit** port of **ram_output_unit.**

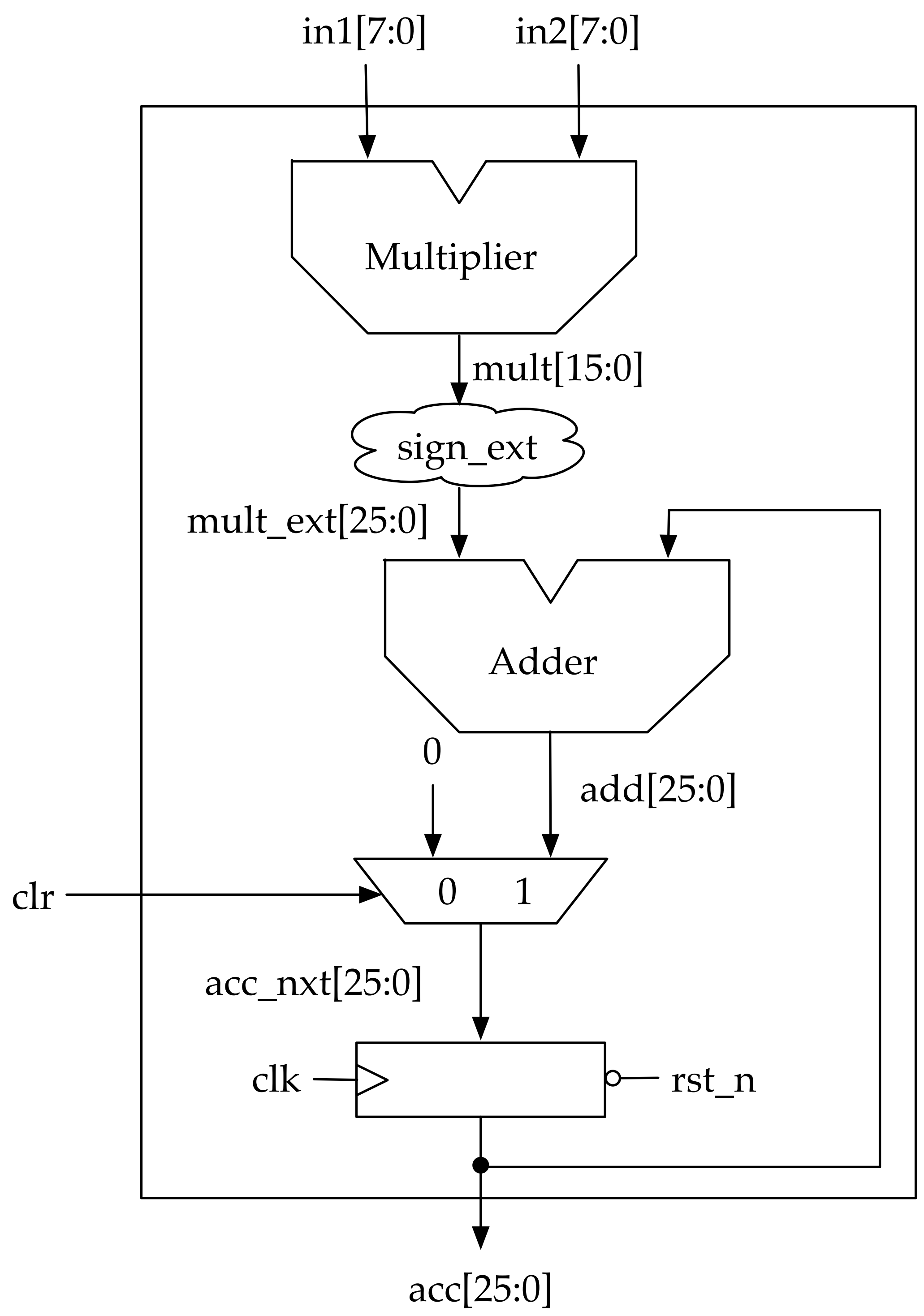
snn_core step 6: Find Maximum



- Find the index of the maximum value in **ram_output_unit** and set **digit**
- Assert **done** for one cycle

Module Details

Mac Operation

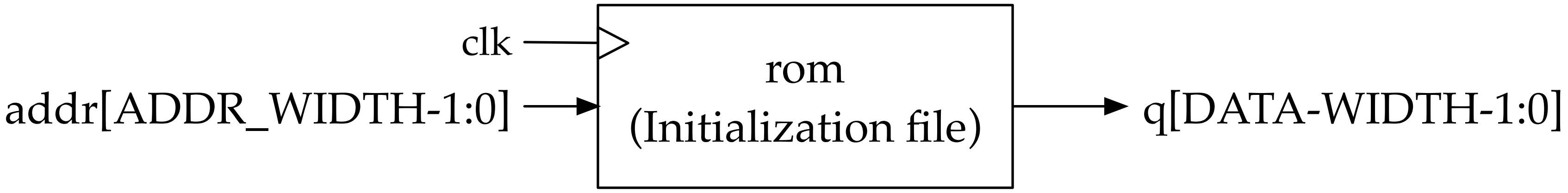


- No need to detect overflow or underflow since 26 bits are enough
- All input operands and intermediate values are signed
- Note that asserting **clr** will take effect one cycle later

Mac Result Rectification

- The output of **mac** is 26 bits, but the address width of **rom_act_func_lut** is 11 bits
- Take 11 bits from **acc**[17:7] if not overflows nor underflows
- If overflows, i.e., **acc** is positive (**acc**[25]==0) and any of **acc**[24:17] is 1, saturate to 11'b011_1111_1111
- If underflows, i.e., **acc** is negative (**acc**[25]==1) and any of **acc**[24:17] is 0, saturate to 11'b100_0000_0000

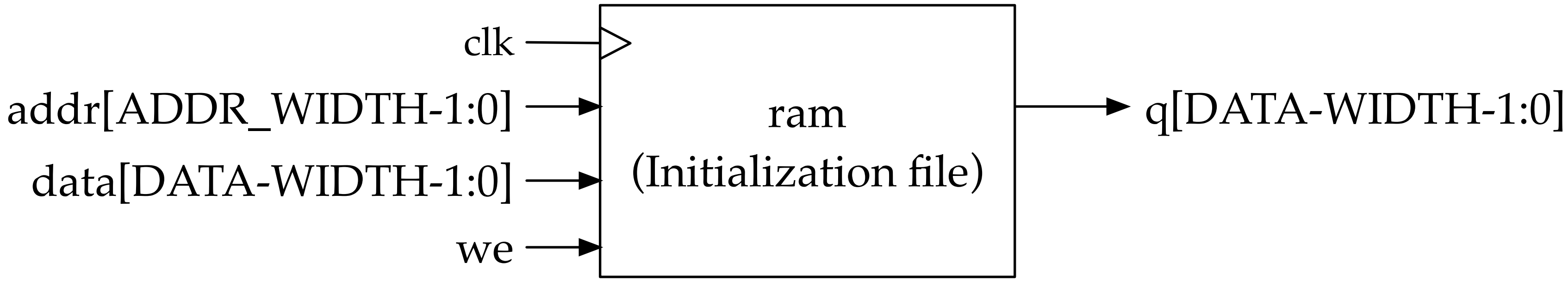
ROM



```
module rom (
  input [(ADDR_WIDTH-1):0] addr,
  input clk,
  output reg [(DATA_WIDTH-1):0] q);

  // Declare the ROM variable
  reg [DATA_WIDTH-1:0] rom[2**ADDR_WIDTH-1:0];

  initial
     readmemh("Initialization file", rom);
  end

  always @ (posedge clk)
  begin
     q <= rom[addr];
  end

endmodule
```

- Set **DATA_WIDTH** and **ADDR_WIDTH**
- No reset
- Initial contents loaded using **readmemh** in an **initial** block
- Read takes one cycle

RAM



```
module ram (
    input [(DATA_WIDTH-1):0] data,
    input [(ADDR_WIDTH-1):0] addr,
    input we, clk,
    output [(DATA_WIDTH-1):0] q);

    // Declare the RAM variable
    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    // Variable to hold the registered read address
    reg [ADDR_WIDTH-1:0] addr_reg;

    initial
         readmemh("Initialization file", ram);
    end

    always @ (posedge clk)
    begin
         if (we) // Write
              ram[addr] <= data;
         addr_reg <= addr;
    end

    assign q = ram[addr_reg];

endmodule
```

- Set **DATA_WIDTH** and **ADDR_WIDTH**
- No reset
- Assert **we** to enable write
- Separate data in (**data**) and data out (**q**) ports
- Initial contents loaded using **readmemh** in an **initial** block
- Read and write takes one cycle

Memory (Input Unit RAM)

Name	ROM/ RAM	Data width	Addr width	Description	Initialization file
ram_input_unit	RAM	1	10	Input unit value, x_0 to x_{783}	ram_input_contents.txt (all zeros)

- This RAM is in **snn**
- Ten sample inputs will be provided. Use one of them to initialize the RAM and test **snn_core**.

Input Unit RAM Contents

- **ram_input_unit** is loaded with 784-bit input bitmap
- When running on FPGA and when testing the entire **snn** module:
  - The contents of **ram_input_unit** is loaded using UART
  - Initialize it with **ram_input_contents.txt** with all zeros
- When testing **snn_core** module without UART:
  - Initialize **ram_input_unit** using sample input RAM contents files **ram_input_contents_sample_?.txt** (see “Test ROM Initialization File Set” page)

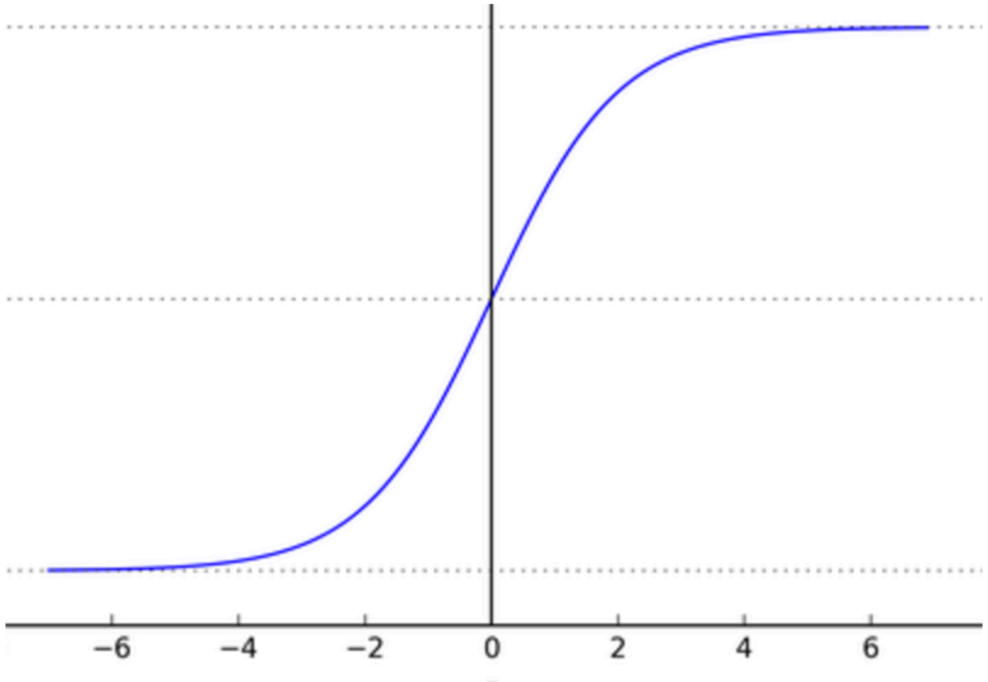
Memory (ROM and RAM)

Name	ROM/ RAM	Data width	Addr width	Description	Initialization file
rom_hidden_weight	ROM	8	5+10 =15 (*)	Hidden weight, $w_h(0,0)$ to $w_h(783,31)$	rom_hidden_weight_contents.txt
rom_output_weight	ROM	8	4+5 =9 (**)	Output weight, $w_o(0,0)$ to $w_o(31,9)$	rom_output_weight_contents.txt
ram_hidden_unit	RAM	8	5	Hidden unit result, y_0 to y_{31}	ram_hidden_contents.txt (all zeros)
ram_output_unit	RAM	8	4	Output unit result, z_0 to z_9	ram_output_contents.txt (all zeros)
rom_act_func_lut	ROM	8	11	Activation function LUT	rom_act_func_lut_contents.txt

- These ROMs and RAMs in **snn_core**
- (*) addr_hidden_weight[14:0] = {cnt_hidden[4:0], cnt_input[9:0]}
- (**) addr_output_weight[8:0] = {cnt_output[3:0], cnt_hidden[4:0]}

Activation Function LUT

- We use a sigmoid activation function for both $y = f(x)$ and $z = f(y)$



- This non-linear function is implemented using an LUT
  - Input: 11-bit mac output, (-1024 to 1023) + 1024 since address cannot be negative
  - Output: 8-bit y_k or z_k, 0 to 127

rom_hidden_weight_contents.txt

```
@0
00 // Hidden # 0, Input #   0, 0x00=   0
00 // Hidden # 0, Input #   1, 0x00=   0
00 // Hidden # 0, Input #   2, 0x00=   0
00 // Hidden # 0, Input #   3, 0x00=   0
00 // Hidden # 0, Input #   4, 0x00=   0
…
06 // Hidden # 0, Input #514, 0x06=   6
13 // Hidden # 0, Input #515, 0x13=  19
05 // Hidden # 0, Input #516, 0x05=   5
03 // Hidden # 0, Input #517, 0x03=   3
26 // Hidden # 0, Input #518, 0x26=  38
…

@1000
00 // Hidden # 4, Input #   0, 0x00=   0
00 // Hidden # 4, Input #   1, 0x00=   0
00 // Hidden # 4, Input #   2, 0x00=   0
00 // Hidden # 4, Input #   3, 0x00=   0
00 // Hidden # 4, Input #   4, 0x00=   0
…
@1C00
00 // Hidden # 7, Input #   0, 0x00=   0
00 // Hidden # 7, Input #   1, 0x00=   0
00 // Hidden # 7, Input #   2, 0x00=   0
00 // Hidden # 7, Input #   3, 0x00=   0
00 // Hidden # 7, Input #   4, 0x00=   0
```

$$w_h(0,0) = 0x00$$

$$w_h(514,0) = 0x06$$

$$w_h(0,4) = 0x00$$

$$w_h(0,7) = 0x00$$

rom_output_weight_contents.txt

```
@0
C3 // Output # 0, Hidden #  0, 0xC3= -61
```

$\longleftarrow w_o(0,0) = 0xC3$

```
16 // Output # 0, Hidden #  1, 0x16=  22
0E // Output # 0, Hidden #  2, 0x0E=  14
D9 // Output # 0, Hidden #  3, 0xD9= -39
A4 // Output # 0, Hidden #  4, 0xA4= -92
…
@20
0D // Output # 1, Hidden #  0, 0x0D=  13
```

$\longleftarrow w_o(0,1) = 0x0D$

```
EB // Output # 1, Hidden #  1, 0xEB= -21
E6 // Output # 1, Hidden #  2, 0xE6= -26
19 // Output # 1, Hidden #  3, 0x19=  25
16 // Output # 1, Hidden #  4, 0x16=  22
…
35 // Output # 8, Hidden # 15, 0x35=  53
```

$\longleftarrow w_o(15,8) = 0x35$

```
FF // Output # 8, Hidden # 16, 0xFF=  -1
B8 // Output # 8, Hidden # 17, 0xB8= -72
2D // Output # 8, Hidden # 18, 0x2D=  45
AF // Output # 8, Hidden # 19, 0xAF= -81
0E // Output # 8, Hidden # 20, 0x0E=  14
```

rom_act_func_lut_contents.txt

```
00 // f(-1024)=0
```

$\longleftarrow f(-1024) = 0$

```
00 // f(-1023)=0
00 // f(-1022)=0
...
3f // f(-5)=63
3f // f(-4)=63
3f // f(-3)=63
```

$\longleftarrow f(-3) = 63 = 0x3F$

```
40 // f(-2)=64
40 // f(-1)=64
40 // f(0)=64
40 // f(1)=64
40 // f(2)=64
41 // f(3)=65
41 // f(4)=65
...
7f // f(1020)=127
7f // f(1021)=127
7f // f(1022)=127
7f // f(1023)=127
```

$\longleftarrow f(-1023) = 127 = 0x7F$

Design Flow

Bottom-Up Design Flow

(a)Design and test **uart_tx**, **uart_rx** separately and together (HW3)

(b)Design and test **mac** (modify from EX7)

(c)Design and test **snn_core** using input pre-loaded on RAM **ram_input_contents_sample_?.txt** (not loaded via UART)

(d)Integrate **uart_tx**, **uart_rx**, and **snn_core** into **snn**

(e)Write a testbench for **snn**

- For testing purpose, use the pre-loaded RAM used in (c) and another **uart_tx** to load the empty input RAM inside **snn**
- Use another **uart_rx** to receive the final result and display on the console

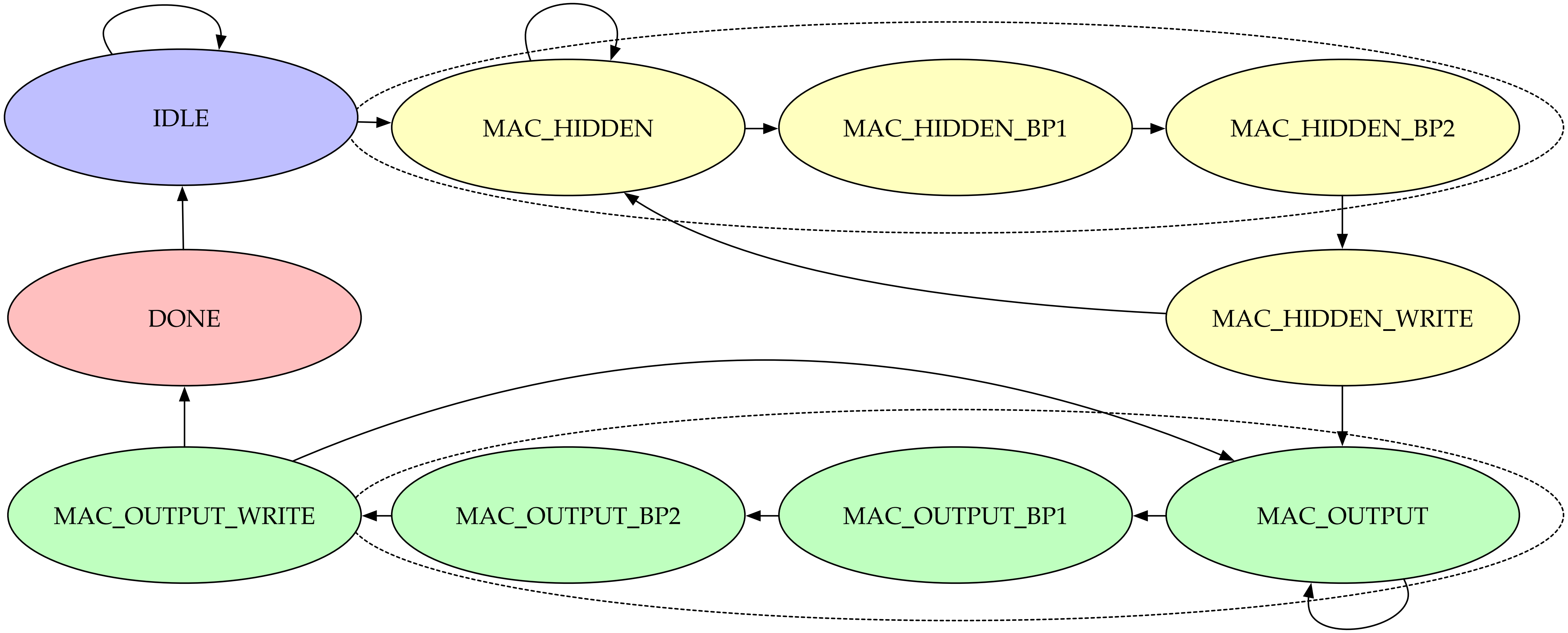
(f)Finally, run it on FPGA

Hints

Don’t Forget

- **mac** may take one cycle to be cleared
- **RAM** and **ROM** takes one cycle for read

FSM Design Example



- Note: No explicit state for finding maximum z_i. Maximum value is updated on the fly when written on **ram_output_unit**

Performance Improvement

Performance Improvement

- Improve performance by using more parallel mac operators
  - Split RAMs and ROMs for parallel access
  - Performance will dramatically improve
  - Area will increase (trade-off!)
- Find maximum as you calculate output unit
  - You will not need **ram_output_unit**, so you save some area too
- More optimization - share your ideas on Canvas for bonus points
- Describe your optimization ideas in the project report
  - What you have done and what you think you could do
- Use can modify only **snn_core** to improve performance. Performance evaluation will be done by simulation without UART transmission.

Test

Test ROM Initialization File Set

Name:	Digit	Recognized digit	Math file
ram_input_contents_sample_0.txt	0	0	mac_trace_hidden_0.txt mac_trace_output_0.txt
ram_input_contents_sample_1.txt	1	2 (incorrect)	mac_trace_hidden_1.txt mac_trace_output_1.txt
ram_input_contents_sample_2.txt	2	2	mac_trace_hidden_2.txt mac_trace_output_2.txt
ram_input_contents_sample_3.txt	3	3	mac_trace_hidden_3.txt mac_trace_output_3.txt
ram_input_contents_sample_4.txt	4	4	mac_trace_hidden_4.txt mac_trace_output_4.txt
ram_input_contents_sample_5.txt	5	1 (incorrect)	mac_trace_hidden_5.txt mac_trace_output_5.txt
ram_input_contents_sample_6.txt	6	6	mac_trace_hidden_6.txt mac_trace_output_6.txt
ram_input_contents_sample_7.txt	7	7	mac_trace_hidden_7.txt mac_trace_output_7.txt
ram_input_contents_sample_8.txt	8	8	mac_trace_hidden_8.txt mac_trace_output_8.txt
ram_input_contents_sample_9.txt	9	0 (incorrect)	mac_trace_hidden_9.txt mac_trace_output_9.txt

Serial Input Files

- Once you program on FPGA, input is provided through UART using RealTerm
- These input files will be provided later on purpose to get you to verify your design using ModelSim

ram_input_contents_sample_4.txt

```
//+----------------------------+
//|                            |
//|                            |
//|                            |
//|                            |
//|                            |
//|                     *      |
//|                     *      |
//|     *              **      |
//|    **              **      |
//|    **             **       |
//|    **             **       |
//|    **             **       |
//|   **             ***       |
//|   **            ***        |
//|   **       ********        |
//|   ************   **        |
//|     *****        **        |
//|                 ***        |
//|                 **         |
//|                 **         |
//|                 **         |
//|                 **         |
//|                 ***        |
//|                 ***        |
//|                  **        |
//|                            |
//|                            |
//|                            |
//+----------------------------+
0
0
0
0
0
```

← 28x28 bitmap

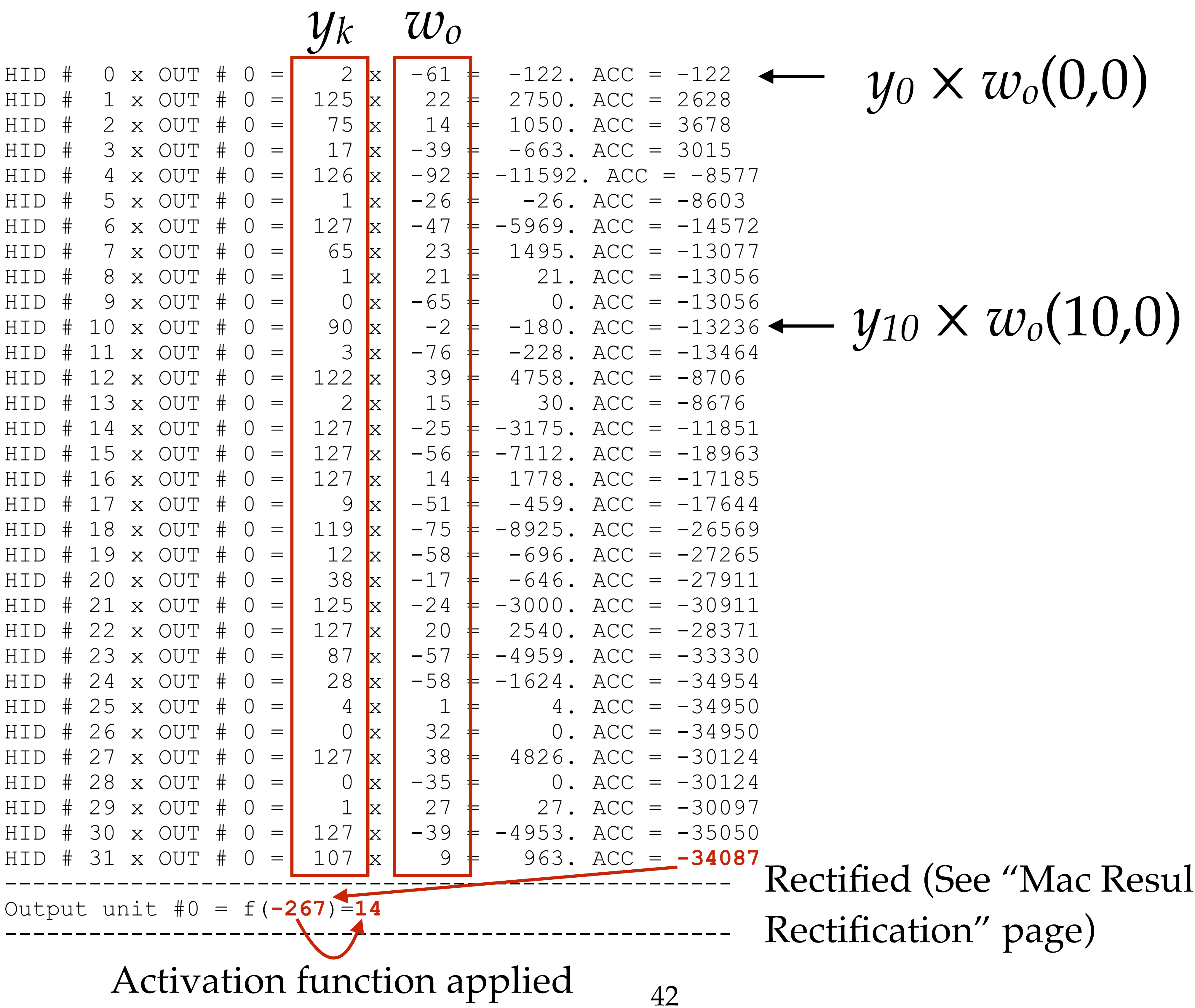
← Bit 0

← Bit 4

Check the Math

- **mac_trace_hidden_?.txt** is the trace of mac operations between input units and hidden weights
- **mac_trace_output_?.txt** is the trace of mac operations between hidden units and output weights
- **$display** your **mac** inputs and output and compare with the reference - comparing waveform will be very painful. Don’t do this.

mac_trace_output_4.txt



y_k w_o

```
HID #  0 x OUT # 0 =    2 x  -61 =   -122. ACC = -122
HID #  1 x OUT # 0 =  125 x   22 =   2750. ACC = 2628
HID #  2 x OUT # 0 =   75 x   14 =   1050. ACC = 3678
HID #  3 x OUT # 0 =   17 x  -39 =   -663. ACC = 3015
HID #  4 x OUT # 0 =  126 x  -92 = -11592. ACC = -8577
HID #  5 x OUT # 0 =    1 x  -26 =    -26. ACC = -8603
HID #  6 x OUT # 0 =  127 x  -47 =  -5969. ACC = -14572
HID #  7 x OUT # 0 =   65 x   23 =   1495. ACC = -13077
HID #  8 x OUT # 0 =    1 x   21 =     21. ACC = -13056
HID #  9 x OUT # 0 =    0 x  -65 =      0. ACC = -13056
HID # 10 x OUT # 0 =   90 x   -2 =   -180. ACC = -13236
HID # 11 x OUT # 0 =    3 x  -76 =   -228. ACC = -13464
HID # 12 x OUT # 0 =  122 x   39 =   4758. ACC = -8706
HID # 13 x OUT # 0 =    2 x   15 =     30. ACC = -8676
HID # 14 x OUT # 0 =  127 x  -25 =  -3175. ACC = -11851
HID # 15 x OUT # 0 =  127 x  -56 =  -7112. ACC = -18963
HID # 16 x OUT # 0 =  127 x   14 =   1778. ACC = -17185
HID # 17 x OUT # 0 =    9 x  -51 =   -459. ACC = -17644
HID # 18 x OUT # 0 =  119 x  -75 =  -8925. ACC = -26569
HID # 19 x OUT # 0 =   12 x  -58 =   -696. ACC = -27265
HID # 20 x OUT # 0 =   38 x  -17 =   -646. ACC = -27911
HID # 21 x OUT # 0 =  125 x  -24 =  -3000. ACC = -30911
HID # 22 x OUT # 0 =  127 x   20 =   2540. ACC = -28371
HID # 23 x OUT # 0 =   87 x  -57 =  -4959. ACC = -33330
HID # 24 x OUT # 0 =   28 x  -58 =  -1624. ACC = -34954
HID # 25 x OUT # 0 =    4 x    1 =      4. ACC = -34950
HID # 26 x OUT # 0 =    0 x   32 =      0. ACC = -34950
HID # 27 x OUT # 0 =  127 x   38 =   4826. ACC = -30124
HID # 28 x OUT # 0 =    0 x  -35 =      0. ACC = -30124
HID # 29 x OUT # 0 =    1 x   27 =     27. ACC = -30097
HID # 30 x OUT # 0 =  127 x  -39 =  -4953. ACC = -35050
HID # 31 x OUT # 0 =  107 x    9 =    963. ACC = -34087
-----------------------------------------------------
Output unit #0 = f(-267)=14
-----------------------------------------------------
```

$y_0 \times w_o(0,0)$

$y_{10} \times w_o(10,0)$

Rectified (See “Mac Result Rectification” page)

Activation function applied

Synthesis

Synthesize Your Design

- You have to be able to synthesize your design at the **`snn`** level of hierarchy.
- Your synthesis script should write out a gate level netlist of follower (**`snn.vg`**).
- You should be able to demonstrate at least one of your tests running on this post synthesis netlist successfully.
- Timing (400MHz) is mildly challenging. Your main synthesis objective is to minimize area.

Synthesis Constraints

Contraint	Value
Clock frequency	400MHz (yes, I know the project spec speaks of 50MHz, but that is for the FPGA mapped version. The TSMC mapped version needs to hit 400MHz.
Input delay	0.5ns after clock rise for all inputs
Output delay	0.5ns prior to next clock rise for all outputs
Drive strength of inputs	Equivalent to a ND2D2BWP gate from our library
Output load	0.1pF on all outputs
Wireload mode	TSMC32K_Lowk_Conservative
Max transition time	0.15ns
Clock uncertainty	0.10ns

- NOTE: Area should be taken after all hierarchy in the design has been smashed.

Grading

Grading Criteria (subject to change)

- Project demo: 55%
  - Code review: 20%
    - DUTs: Comment quality, conformity to the guidelines, etc.
    - Testbenches: Comment quality, coverage, etc.
  - Functionality: 20%
    - Functionality test using reference testbenches (not all reference testbenches will be provided)
  - Synthesis script review: 5%
  - Post-synthesis test results: 10%
- Report: 15%
  - 4-page report on your efforts to improve performance and reduce area
- Design performance: 15%
  - Number of cycles to complete test
- Design area: 15%
  - Cell area (Synopsys) and FPGA resource usage (Quartus)
- Bonus points: 3%
  - Discussion participation on Canvas (good questions and good answers): 1.5%
  - Use of a version control system: 1.5%

Bonus Points

- Up to 1.5% extra credit for making significant contributions on the Canvas Discussion Forum
- This specification is incomplete or even incorrect
  - In practice, you are never given a perfect specification for a new design
- Make contributions
  - Make definitions more clear
  - Correct inconsistencies
    - e.g., 15-bit output connected to 16-bit input
  - Participate in discussions. Share ideas and information.

Demo Plan

Project Demo (Tentative)

- Location: EH 3634 or 4613
- Date: 5/3 (Thu) and 5/4 (Fri)
  - 1.25% extra credit for demoing on Thursday
  - Reserve a time slot (to be announced later)
- Flash ROM file
- No need to bring your laptop and cable
- Short interview