Bit-Serial Neural Computation Engine: Implementation and Resource Analysis

Sudipto Sarkar
Rickarya Das
Arka Das

Contents

- Introduction
- Why Neural Computation?
- What is a neuron?
- Proposed Architechture
- What is Input Buffer?
- Hidden Weight Memory
- Flattening and Latency
- Mac Engine Structure
- FSM-Mac Engine
- ReLU Activation
- Device Implementation
- Utilisation Report
- Timing Report
- Power Report
- Conclusion
- References

INTRODUCTION

The Challenge with Neural Networks on FPGAs

1. Heavy Dependence on MAC/DSP Units
2. Not Suitable for Edge Devices
3. Traditional Approaches Don't Scale Down

Can neural inference be done efficiently *without* using a single DSP or MAC unit...

?

What We Built?

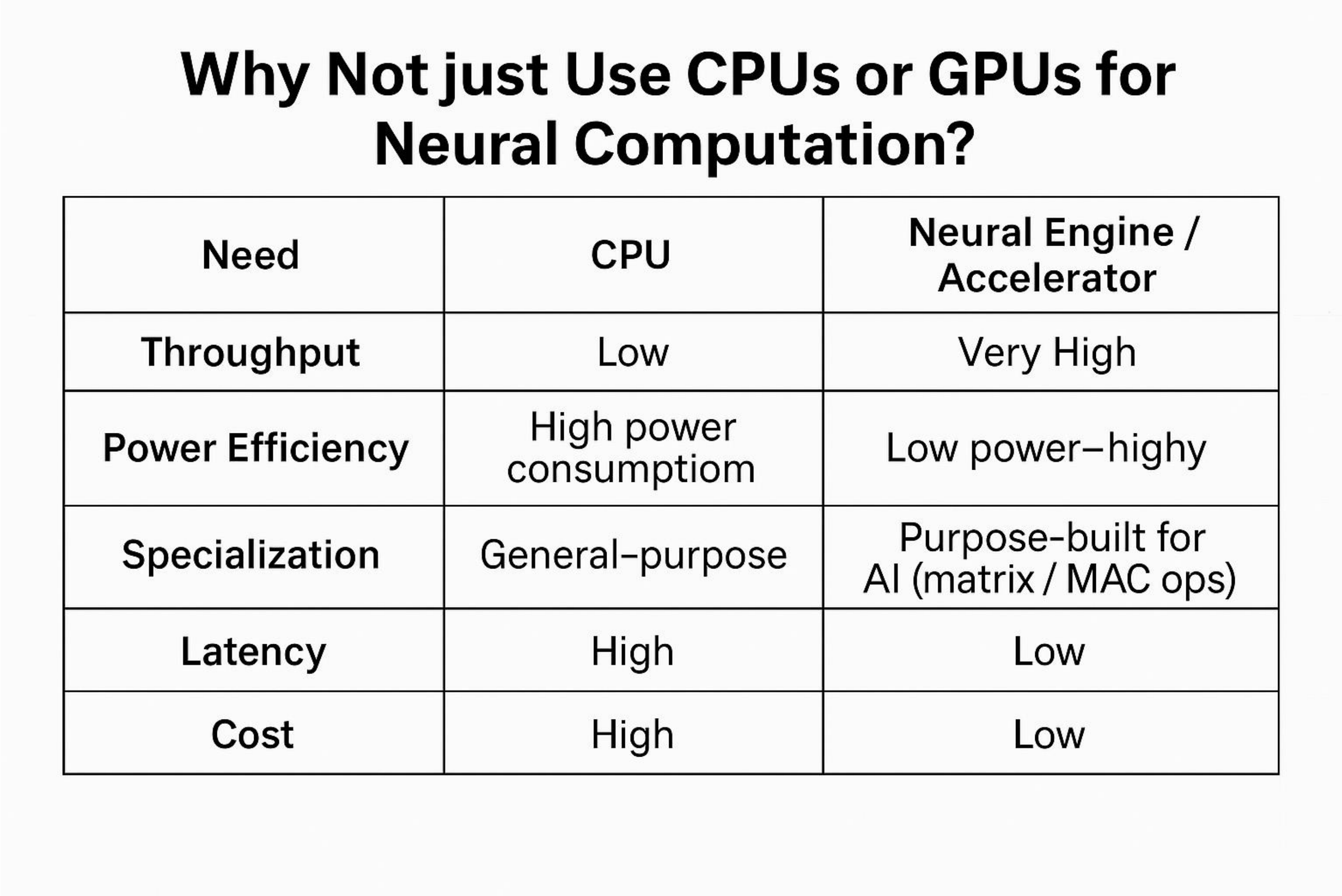
- ✓ A **neural network accelerator** implemented **entirely with LUTs, flip-flops, and shift-add logic**.
- ✓ A **bit-serial Processing Element** that performs 8-bit × 8-bit multiply-accumulate **one bit per cycle**, reducing hardware footprint dramatically.
- ✓ A modular, two-stage architecture that supports **streamed inputs, multi-neuron parallelism, and configurable layers**.

"We prove that neural networks don't need heavy MAC units—just clever architecture."

Why This Problem Matters?

- Enables **ML deployment on low-cost FPGAs** where DSPs are scarce.
- Reduces **power consumption**, **area footprint**, and **hardware complexity**.
- Opens the door for **scalable, customizable neural hardware** using only basic logic (LUTs + registers).

Why Neural Computation?



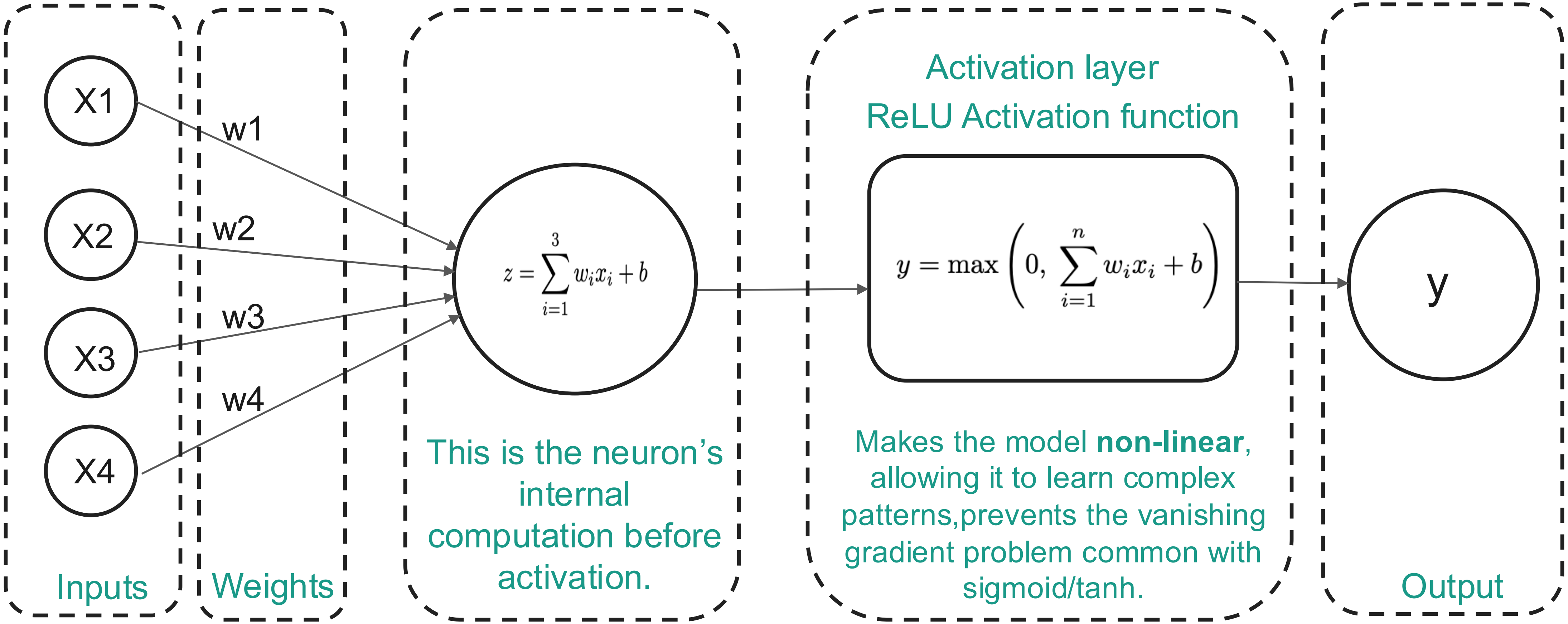
Why Not just Use CPUs or GPUs for Neural Computation?

Need	CPU	Neural Engine / Accelerator
Throughput	Low	Very High
Power Efficiency	High power consumptiom	Low power–highy
Specialization	General–purpose	Purpose-built for AI (matrix / MAC ops)
Latency	High	Low
Cost	High	Low

Why Neural Engines Win?

- Designed specifically for **massive parallel multiply-accumulate (MAC)** operations.
- **10×–100× better energy efficiency** compared to CPUs/GPUs.
- Optimized memory pipelines reduce data movement cost.
- Lower latency → suitable for real-time inference (speech, vision, robotics).

What is a neuron?



$$z = \sum_{i=1}^{3} w_i x_i + b$$

This is the neuron's internal computation before activation.

Activation layer

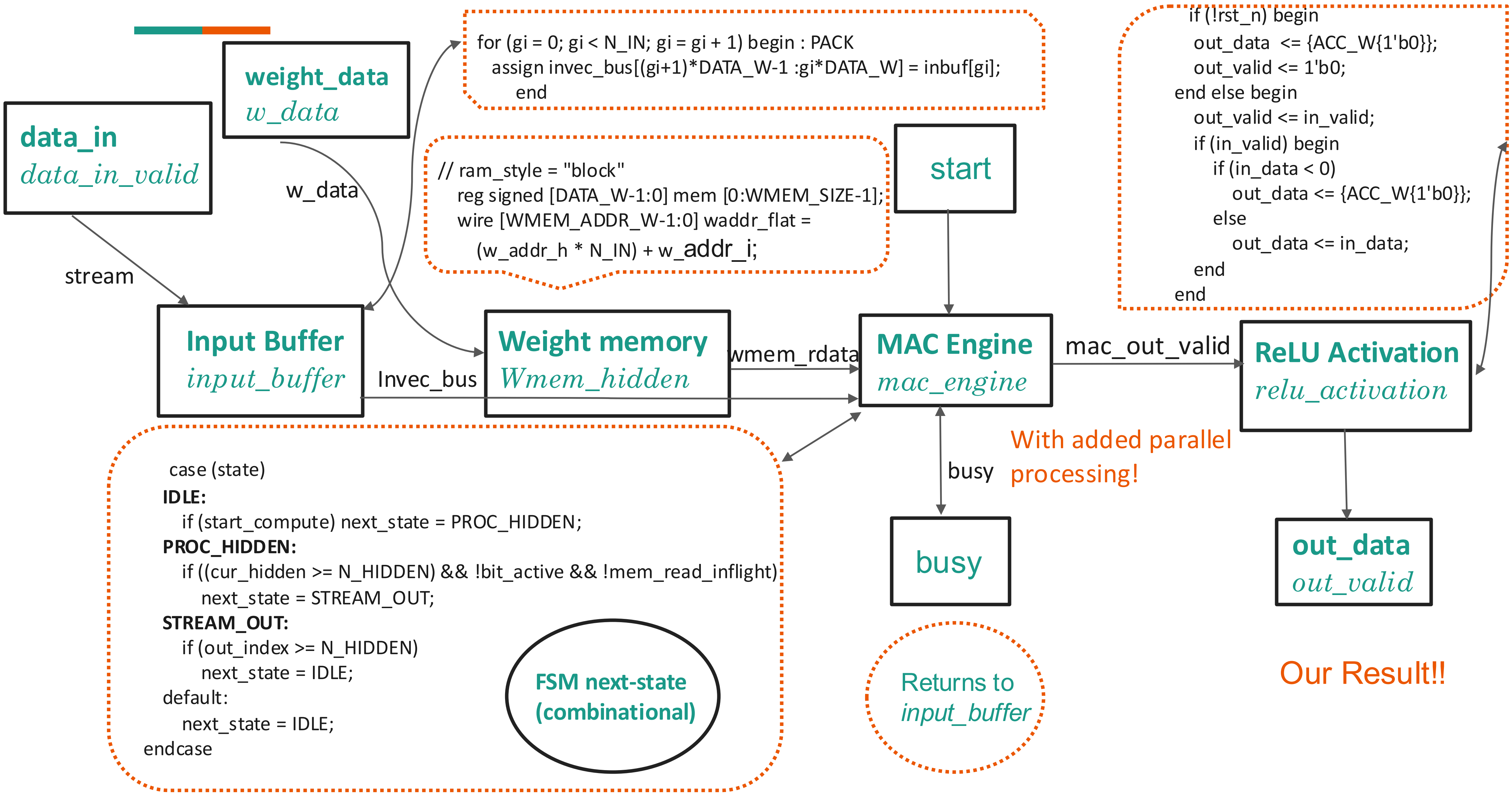
ReLU Activation function

$$y = \max\left(0, \sum_{i=1}^{n} w_i x_i + b\right)$$

Makes the model **non-linear**, allowing it to learn complex patterns,prevents the vanishing gradient problem common with sigmoid/tanh.

Mathematical definition of a single neuron !

Proposed architechture



```
for (gi = 0; gi < N_IN; gi = gi + 1) begin : PACK
  assign invec_bus[(gi+1)*DATA_W-1 :gi*DATA_W] = inbuf[gi];
     end
```

```
// ram_style = "block"
   reg signed [DATA_W-1:0] mem [0:WMEM_SIZE-1];
   wire [WMEM_ADDR_W-1:0] waddr_flat =
      (w_addr_h * N_IN) + w_addr_i;
```

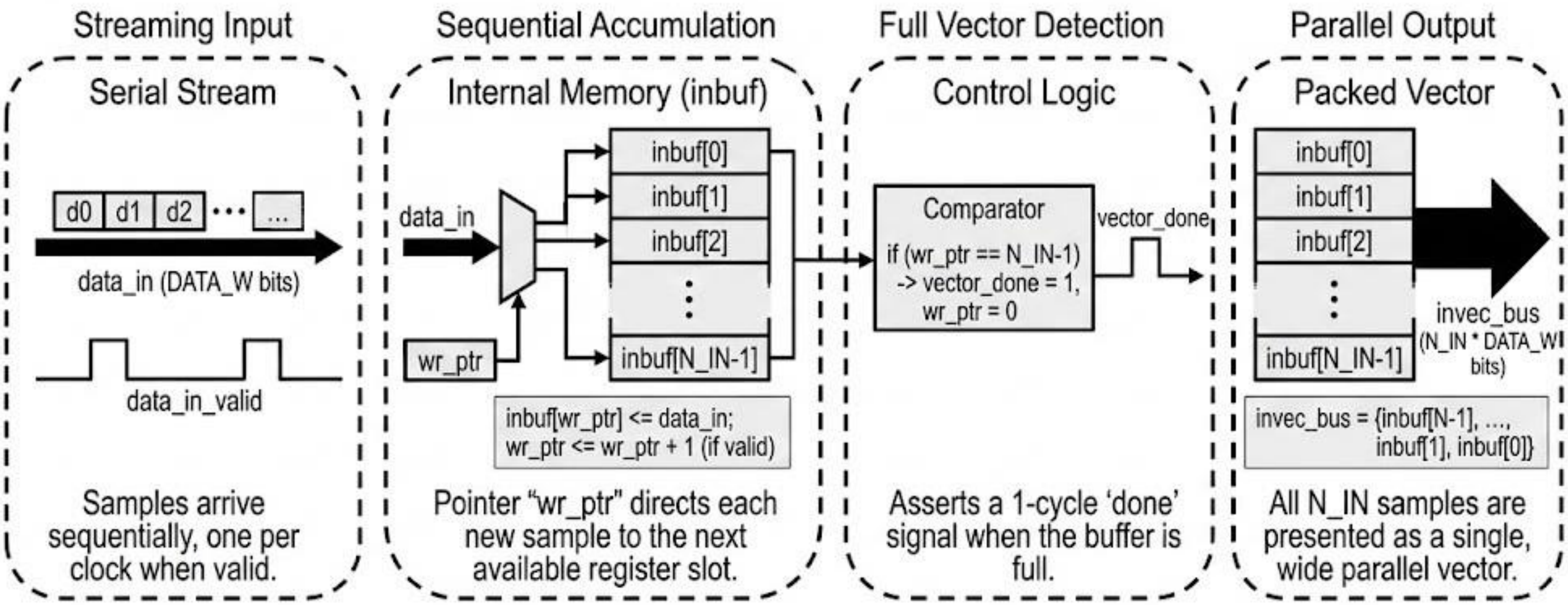
```
    if (!rst_n) begin
      out_data  <= {ACC_W{1'b0}};
      out_valid <= 1'b0;
    end else begin
      out_valid <= in_valid;
      if (in_valid) begin
        if (in_data < 0)
          out_data <= {ACC_W{1'b0}};
        else
          out_data <= in_data;
      end
    end
```

```
 case (state)
IDLE:
   if (start_compute) next_state = PROC_HIDDEN;
PROC_HIDDEN:
   if ((cur_hidden >= N_HIDDEN) && !bit_active && !mem_read_inflight)
      next_state = STREAM_OUT;
STREAM_OUT:
   if (out_index >= N_HIDDEN)
      next_state = IDLE;
default:
   next_state = IDLE;
endcase
```

With added parallel processing!

Our Result!!

What is Input Buffer?



```
inbuf[wr_ptr] <= data_in;
wr_ptr <= wr_ptr + 1 (if valid)
```

```
if (wr_ptr == N_IN-1)
 -> vector_done = 1,
 wr_ptr = 0
```

```
invec_bus = {inbuf[N-1], ...,
             inbuf[1], inbuf[0]}
```

Samples arrive sequentially, one per clock when valid.

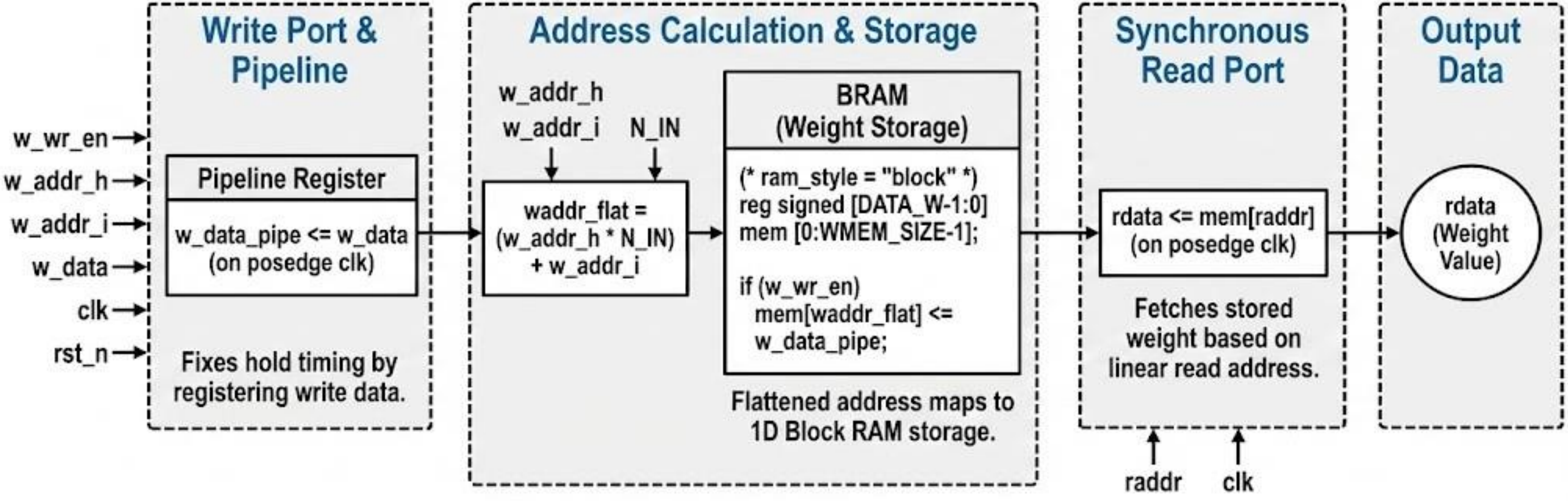
Pointer “wr_ptr” directs each new sample to the next available register slot.

Asserts a 1-cycle ‘done’ signal when the buffer is full.

All N_IN samples are presented as a single, wide parallel vector.

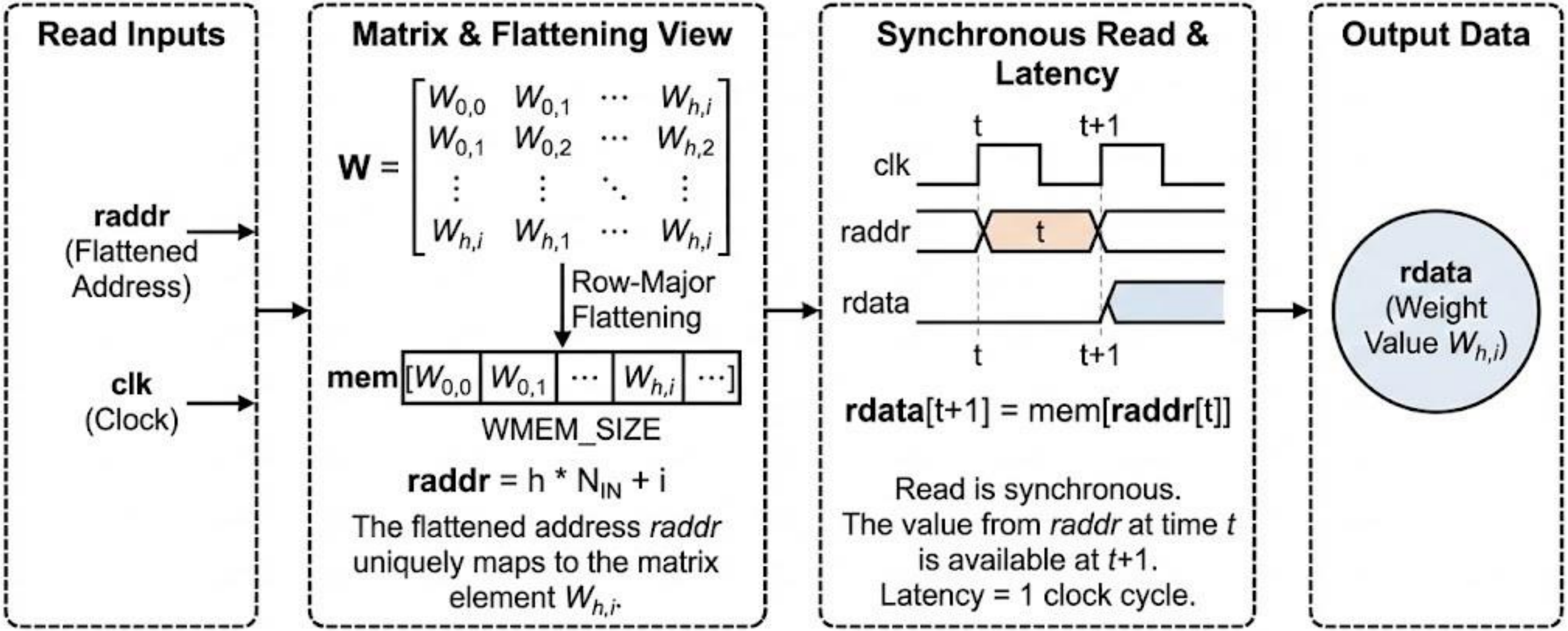
Converts streaming serial data into a wide parallel vector for high-throughput processing!

Hidden Weight Memory



Structure and Operation of the Hidden Layer Weight Memory Module.

Flattening and Latency



Read Inputs

raddr (Flattened Address)

clk (Clock)

Matrix & Flattening View

$$\mathbf{W} = \begin{bmatrix} W_{0,0} & W_{0,1} & \cdots & W_{h,i} \\ W_{0,1} & W_{0,2} & \cdots & W_{h,2} \\ \vdots & \vdots & \ddots & \vdots \\ W_{h,i} & W_{h,1} & \cdots & W_{h,i} \end{bmatrix}$$

Row-Major Flattening

mem $[W_{0,0} \mid W_{0,1} \mid \cdots \mid W_{h,i} \mid \cdots]$

WMEM_SIZE

raddr = h * N_{IN} + i

The flattened address *raddr* uniquely maps to the matrix element $W_{h,i}$.

Synchronous Read & Latency

rdata[t+1] = mem[**raddr**[t]]

Read is synchronous.
The value from *raddr* at time *t* is available at *t*+1.
Latency = 1 clock cycle.

Output Data

rdata (Weight Value $W_{h,i}$)

This slide details the mathematical and timing aspects of reading from the flattened weight memory.

Mac_Engine Structure

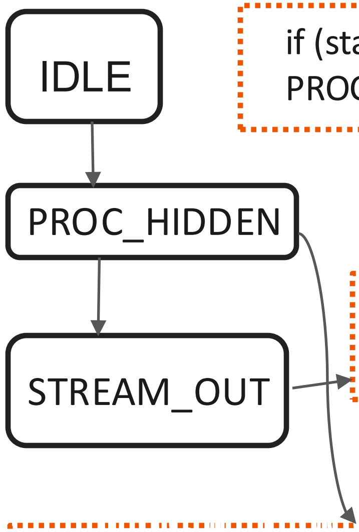
Input Signals and Control

clk →
rst_n →
start_compute →
invec_bus[DATA:0] →
wmem_rdata →

[ACC_W-1:0] hidden_accum [0:N_HIDDEN-1] →

[MEM_INDEX_W-1:0] mem_read_index; →

FSM STATE LOGIC (combinatorial)

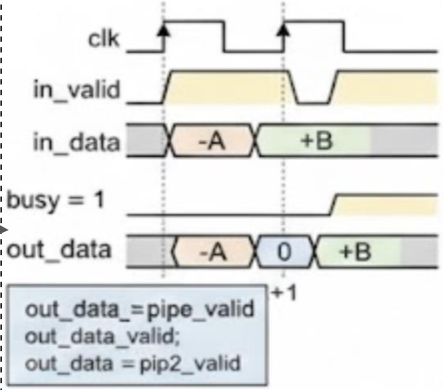


if (start_compute) next_state = PROC_HIDDEN;

if (out_index >= N_HIDDEN) next_state = IDLE;

if ((cur_hidden >= N_HIDDEN) && !bit_active && !mem_read_inflight) next_state = STREAM_OUT;

PIPELINING AND TIMING DIAGRAM



out_data_=pipe_valid
out_data_valid;
out_data = pip2_valid

Latency = 2 clock cycles
out_data_pipe
out_data_pipe_valid
out_data_pipe2
out_data_pipe2_valid
✓ reduces critical path length
✓ ensures the BRAM output path is clean

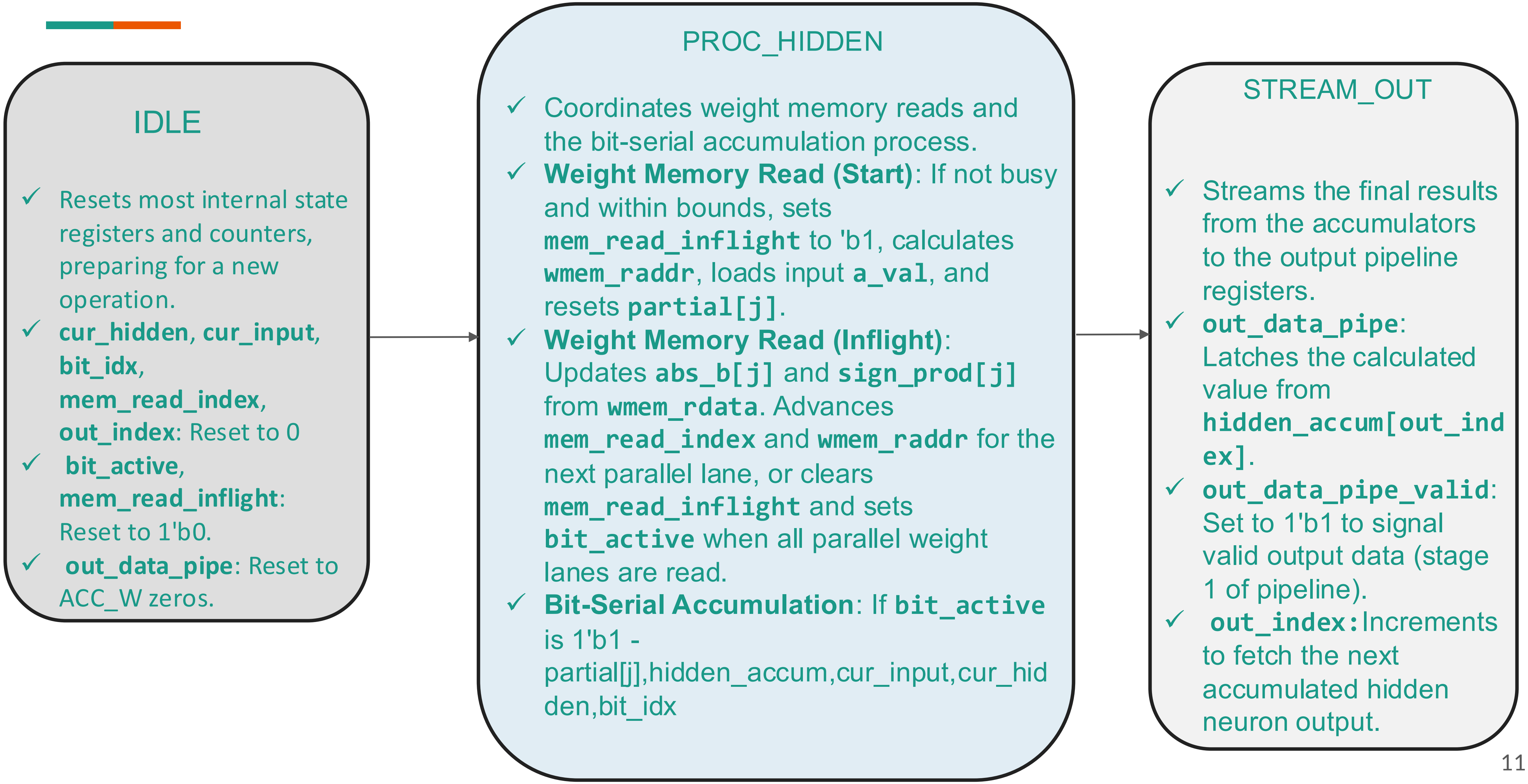
OUTPUT SIGNALS

busy= 0

Output data is valid after 2 clock cycles

output reg signed [(2*DATA_W)+$clog2((N_IN>1)?N_IN:2)-1:0] out_data
output reg out_valid,
output reg busy

FSM – MAC_ENGINE



IDLE

- ✓ Resets most internal state registers and counters, preparing for a new operation.
- ✓ **cur_hidden**, **cur_input**, **bit_idx**, **mem_read_index**, **out_index**: Reset to 0
- ✓ **bit_active**, **mem_read_inflight**: Reset to 1'b0.
- ✓ **out_data_pipe**: Reset to ACC_W zeros.

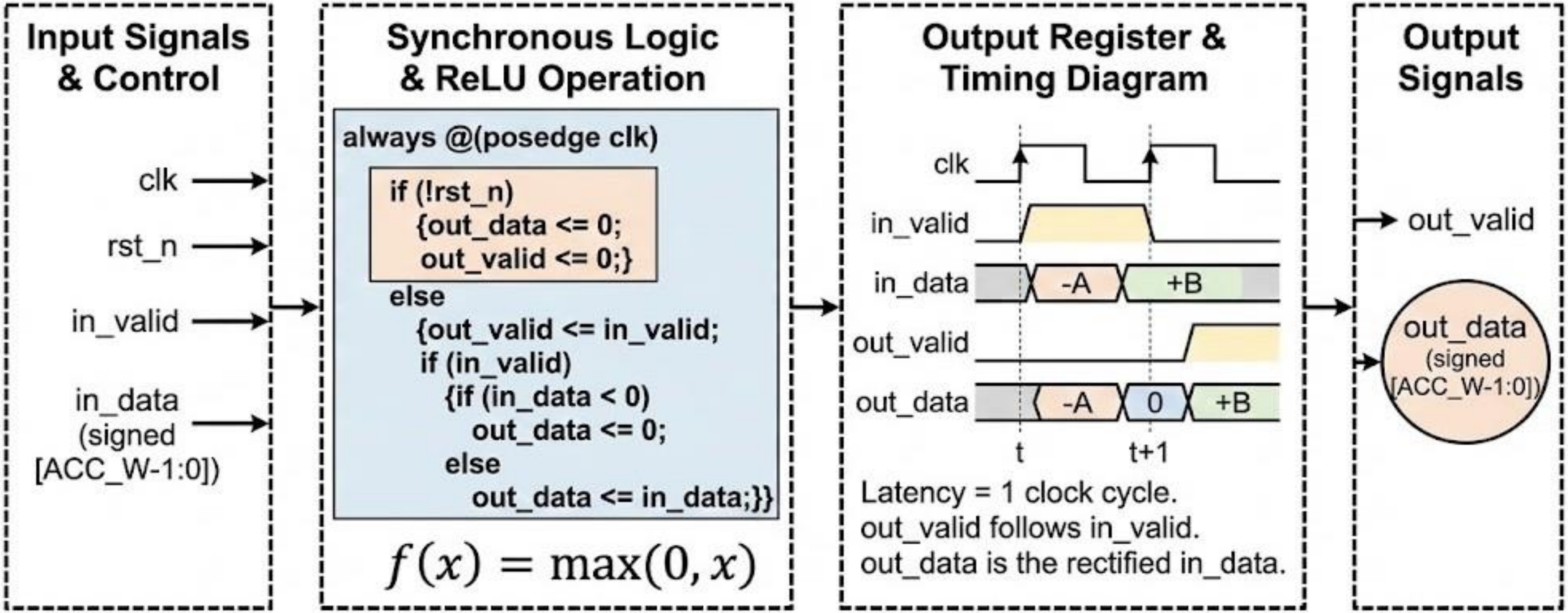
PROC_HIDDEN

- ✓ Coordinates weight memory reads and the bit-serial accumulation process.
- ✓ **Weight Memory Read (Start)**: If not busy and within bounds, sets **`mem_read_inflight`** to 'b1, calculates **`wmem_raddr`**, loads input **`a_val`**, and resets **`partial[j]`**.
- ✓ **Weight Memory Read (Inflight)**: Updates **`abs_b[j]`** and **`sign_prod[j]`** from **`wmem_rdata`**. Advances **`mem_read_index`** and **`wmem_raddr`** for the next parallel lane, or clears **`mem_read_inflight`** and sets **`bit_active`** when all parallel weight lanes are read.
- ✓ **Bit-Serial Accumulation**: If **`bit_active`** is 1'b1 - partial[j],hidden_accum,cur_input,cur_hidden,bit_idx

STREAM_OUT

- ✓ Streams the final results from the accumulators to the output pipeline registers.
- ✓ **`out_data_pipe`**: Latches the calculated value from **`hidden_accum[out_index]`**.
- ✓ **`out_data_pipe_valid`**: Set to 1'b1 to signal valid output data (stage 1 of pipeline).
- ✓ **`out_index`**:Increments to fetch the next accumulated hidden neuron output.

ReLU Activation



Input Signals & Control

Synchronous Logic & ReLU Operation

```
always @(posedge clk)
  if (!rst_n)
    {out_data <= 0;
     out_valid <= 0;}
  else
    {out_valid <= in_valid;
     if (in_valid)
       {if (in_data < 0)
          out_data <= 0;
        else
          out_data <= in_data;}}
```

$$f(x) = \max(0, x)$$

Output Register & Timing Diagram

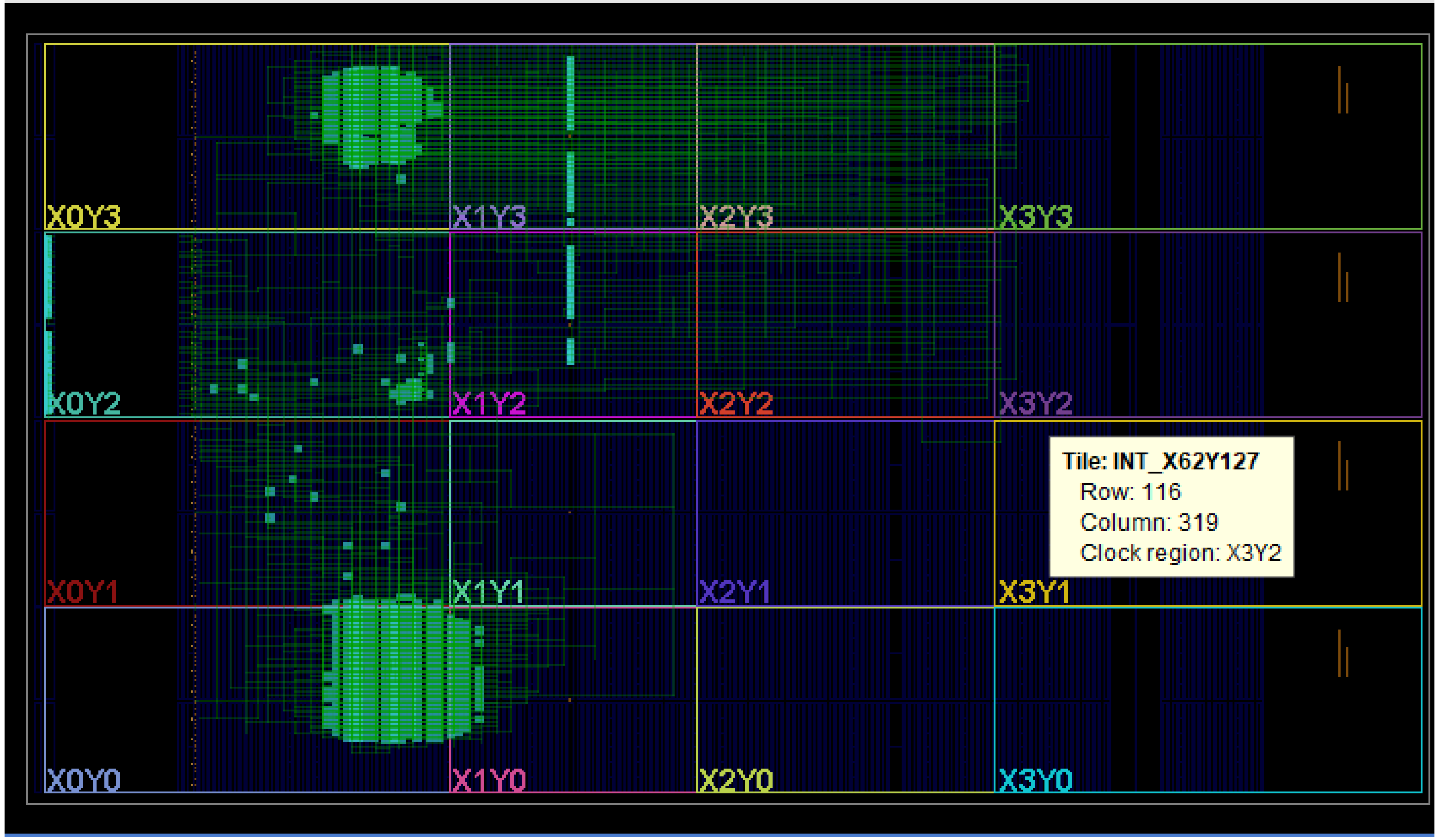
Latency = 1 clock cycle.
out_valid follows in_valid.
out_data is the rectified in_data.

Output Signals

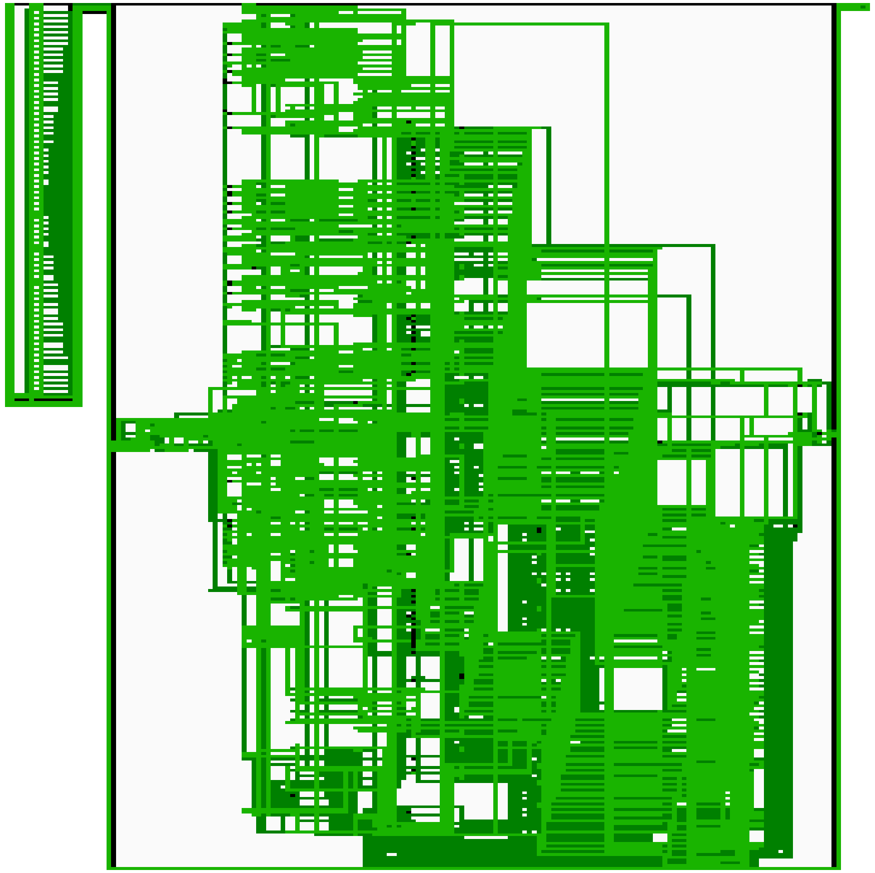
This module implements the Rectified Linear Unit (ReLU) activation function with a single clock cycle latency.

Device Implementation

Implemented with 128 neurons in hidden layer initially! Parallel Processing = 4



Tile: INT_X62Y127
Row: 116
Column: 319
Clock region: X3Y2

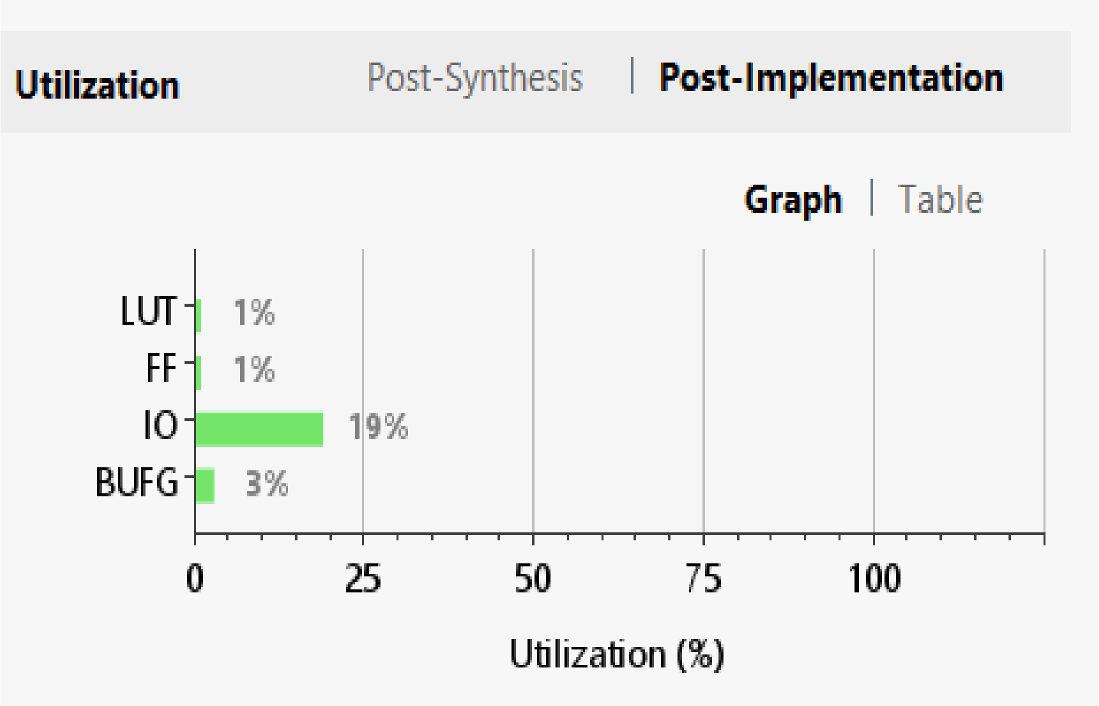


Floorplan Details - Scaled version of our Architecture on Artix Ultrascale+

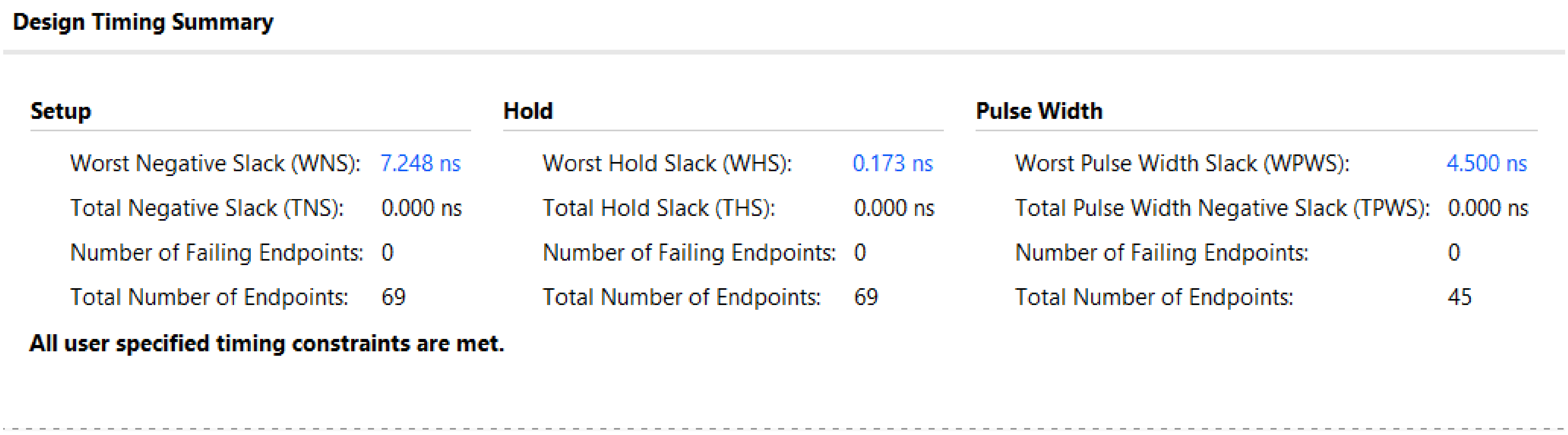
Utilization Report

Resource	Available	Used	Utilization
LUT	41000	34	0.08%
FF	82000	39	0.05%
IO	300	68	22.67%
DSP	240	0	0%

Utilization Graph for 4-3-2 neural computational Engine (A Simplified version)



Timing Report



Design Timing Summary

Setup

Worst Negative Slack (WNS):	7.248 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	69

Hold

Worst Hold Slack (WHS):	0.173 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	69

Pulse Width

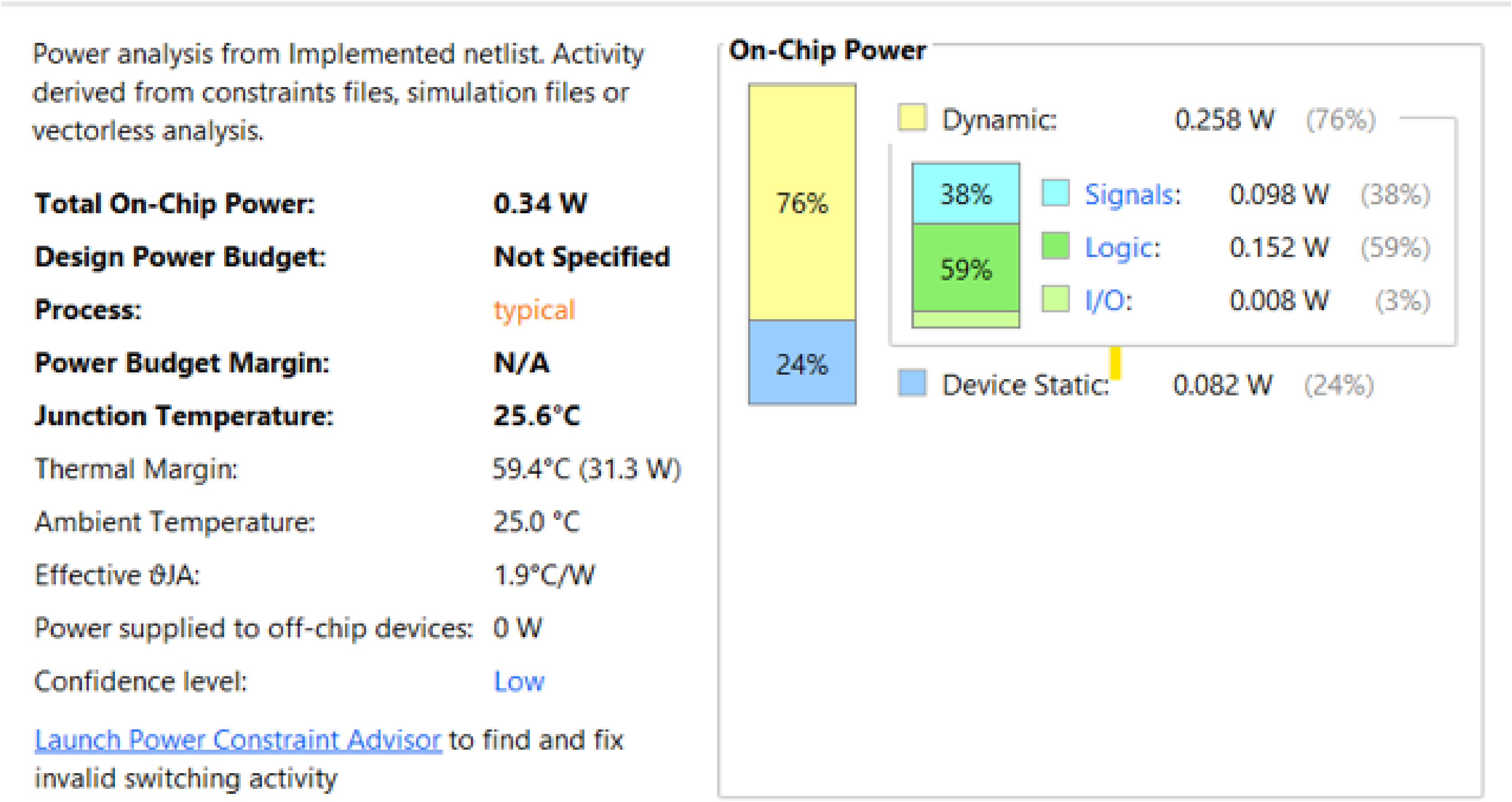
Worst Pulse Width Slack (WPWS):	4.500 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	45

All user specified timing constraints are met.

The design meets all timing requirements at 100 MHz.

There are no setup, hold, or pulse-width violations.

Power Report



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.34 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.6°C**
Thermal Margin:	59.4°C (31.3 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	1.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

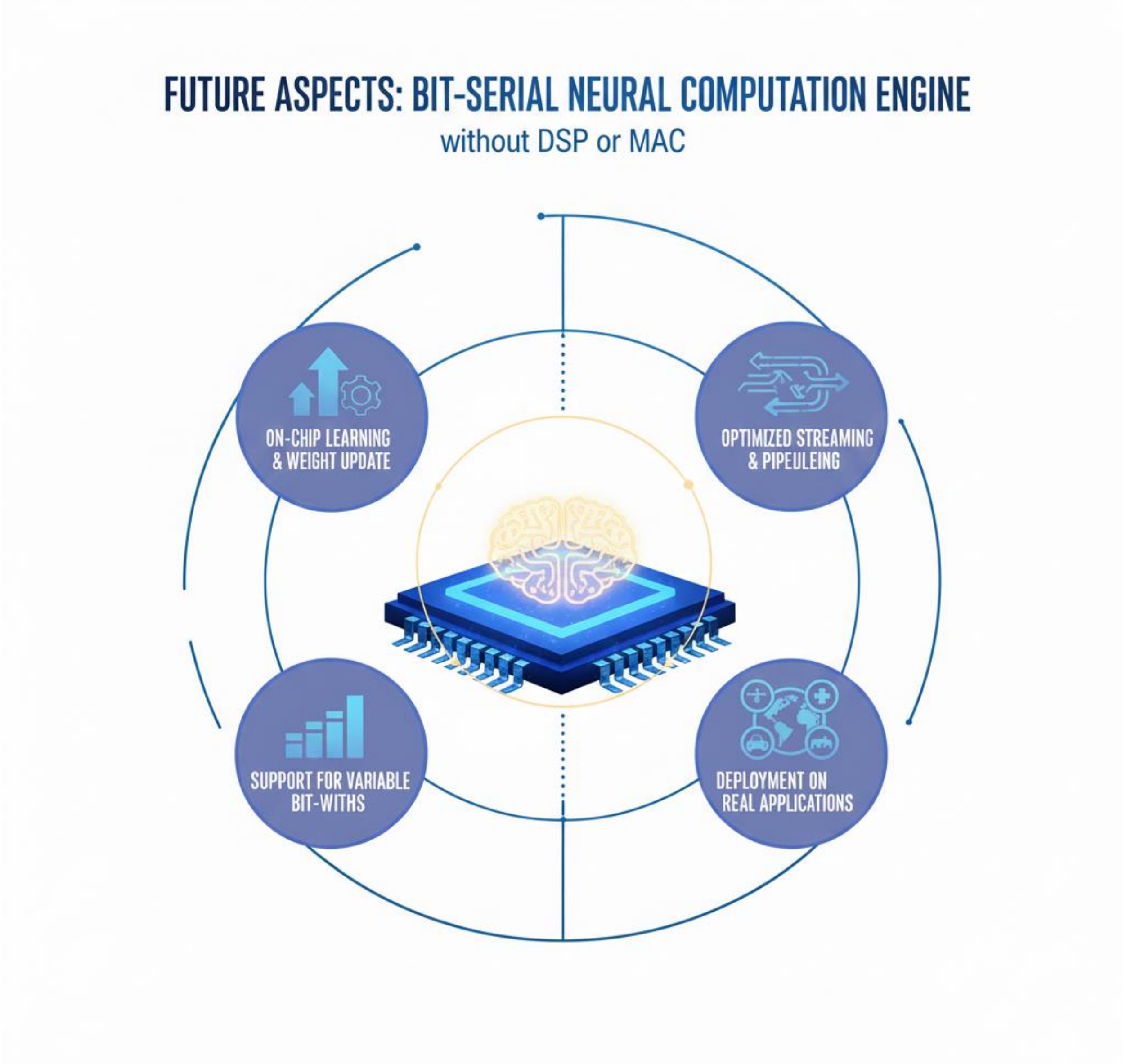
Dynamic:	0.258 W	(76%)
Signals:	0.098 W	(38%)
Logic:	0.152 W	(59%)
I/O:	0.008 W	(3%)
Device Static:	0.082 W	(24%)

The total on-chip power consumption is 0.34 W, with dynamic power being the major contributor.
Most of the dynamic power comes from logic and signal activity, with static power accounting for the rest.

Conclusion

What we achieved?

- ✓ The architecture achieves correct results while consuming **extremely low FPGA resources** (<0.1% LUTs, 0 DSPs) and operating with **very low power**.
- ✓ This confirms that **serialized arithmetic** is a viable alternative to traditional parallel MAC-based accelerators, especially for **edge-class and resource-constrained hardware platforms**.
- ✓ Through its multi-lane **parallel processing feature**, the design accelerates hidden-layer computation by evaluating **multiple neurons simultaneously**, boosting throughput while preserving low area usage.



FUTURE ASPECTS: BIT-SERIAL NEURAL COMPUTATION ENGINE

without DSP or MAC

ON-CHIP LEARNING & WEIGHT UPDATE
OPTIMIZED STREAMING & PIPEULEING
SUPPORT FOR VARIABLE BIT-WITHS
DEPLOYMENT ON REAL APPLICATIONS

References

[1] ”Low-power and low-cost dedicated bit-serial hardware neural network for epileptic seizure prediction system” SM Kueh, TJ Kazmierski
[2] ”BIT - SERIAL NEURAL NETWORKS” Alan F. Murray, Anthony V. W. Smith and Zoe F. Butler. Department of Electrical Engineering, University of Edinburgh, The King’s Buildings, Mayfield Road, Edinburgh, Scotland, EH93JL.
[3] AMD Xilinx, Vivado Design Suite User Guide*, 2024.
[4] Introduction to Neural Computation – MIT
[5] G. Csordas, B. Feher, and T. Kovacshazy, “Application of bit-serial arithmetic units for FPGA implementation of convolutional neural networks,” in *Proc. 19th Int. Conf. Appl. Electron. (AE)*, Pilsen, Czech Republic, 2018, pp. 23–28.

THANK YOU !