Basic Fully Connected Neural Network Hardware Design

Objective

Design a compact hardware accelerator that performs *inference* for a small fully connected (FC) neural network (1 hidden layer). Only the inference path—that is, forward propagation with fixed weights and biases obtained from offline training—is considered in scope.

What is Inference in a Fully Connected Neural Network?

Inference is the forward-pass computation that maps an input vector $\mathbf{x}$ to an output vector $\mathbf{y}$ using parameters $\{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{L}$ learned during training. For a single-hidden-layer FC network the operations are

$$\mathbf{h} = f\big(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big),$$
$$\mathbf{y} = g\big(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}\big),$$

where f and g are activation functions (e.g. ReLU or sigmoid). During inference the weights and biases are *constants*, so there is no back-propagation, weight update, or gradient storage.

Why Dedicated Hardware Instead of Running on a CPU?

a) **Parallel MACs.** Each neuron performs many multiply–accumulate (MAC) operations. A custom datapath can instantiate several MAC units that run concurrently, cutting latency from $\mathcal{O}(NH)$ cycles on a scalar CPU to $\mathcal{O}(\lceil N/\text{MACs} \rceil)$.

b) **Deterministic Low Latency.** Hardware finite-state machines (FSMs) have fixed cycle counts, avoiding cache misses, context switches, and OS jitter found in general-purpose processors.

c) **Energy Efficiency.** Specialized logic eliminates instruction fetch/decode overhead, reducing dynamic power. Activity is focused on arithmetic units and on-chip memories sized precisely for the model.

d) **Throughput Scalability.** Multiple accelerator instances can be tiled or time-multiplexed in an FPGA/ASIC fabric to serve batched inputs at high frame rates.

e) **Software vs. Hardware Description.** Writing Python or C uses software abstractions; designing in VHDL/Verilog describes transistor-level data paths, pipelines, and timing, giving freedom to tune wordlengths, pipeline depths, and routing for maximum speed.

Recommended Background Resources

Before implementing the hardware, review the following concise video primers on neural-network fundamentals:

- 3Blue1Brown: "But what is a neural network?"
- 3Blue1Brown: "Gradient descent, how neural networks learn"
- 3Blue1Brown: "Backpropagation calculus"
- 3Blue1Brown: "What is backpropagation really doing?"

Although back-prop is out of scope, understanding the forward path in context will help you optimize datapaths and word-lengths.

Problem Statement (*Complete in ˜1 week*)

Assumptions

- All weights $\mathbf{W}$ and biases $\mathbf{b}$ are pre-loaded into on-chip registers (8-bit signed, range $-128 \ldots 127$).
- Only inference is required; no training logic is present.

Network Specification

Layer	Dimension	MAC Units Allocated
Input	3×1	—
Hidden	4×1	4 (one per neuron)
Output	2×1	2 (one per neuron)

Finite-State Machine (FSM) Outline

`S1`: **RESET** Clear registers; wait for configuration.
`S2`: **LOAD_W** Load weights and biases from host interface into dedicated registers.
`S3`: **IDLE** Await valid input vector.
`S4`: **FETCH_X**Latch the 3×1 input into local registers.
`S5`: **MAC_H** Perform 3 parallel MAC cycles to compute hidden pre-activations.
`S6`: **ACT_H** Apply activation $f(\cdot)$ (e.g. ReLU) element-wise.
`S7`: **MAC_Y** Compute output pre-activations using hidden results.
`S8`: **ACT_Y** Apply output activation $g(\cdot)$ if required.
`S9`: **WRITE_Y**Expose output vector; assert `valid`.

Each state is single-cycle except the MAC phases, which iterate once per weight column when word-serial multiplication is used.

Dataflow and Components

- **MAC Block**: 8-bit multiplier + 16-bit accumulator + adder for bias injection.
- **Registers**: store $\mathbf{W}^{(1)}$ (4×3), $\mathbf{b}^{(1)}$ (4), $\mathbf{W}^{(2)}$ (2×4), $\mathbf{b}^{(2)}$ (2), intermediary activations.
- **Multiplexers**: select between weight load path and computation path.
- **Activation Unit**: simple combinational ReLU (`y = max(0,x)`) to minimize area.
- **Control FSM**: drives enable signals, address counters, and `valid/ready` handshake with the host.

A high-level hand-drawn block diagram—showing MAC clusters, register files, and control sequencing—is recommended before coding in VHDL.

Beyond the Baseline

The one-MAC-per-neuron architecture provides clarity but is area-heavy. Real accelerators often share a single MAC array across layers, schedule operations over time, or exploit sparsity to skip multiplications. Such optimizations shrink area/energy at the cost of a more complex FSM and memory subsystem—perfect extensions once the baseline design is complete.

NOTE!

Make sure to keep your weights low in range for now, as we are using 8-bit registers, they can't store values for than 8-bits, hence all the intermediate result mist also comply with this length
Happy Learning!