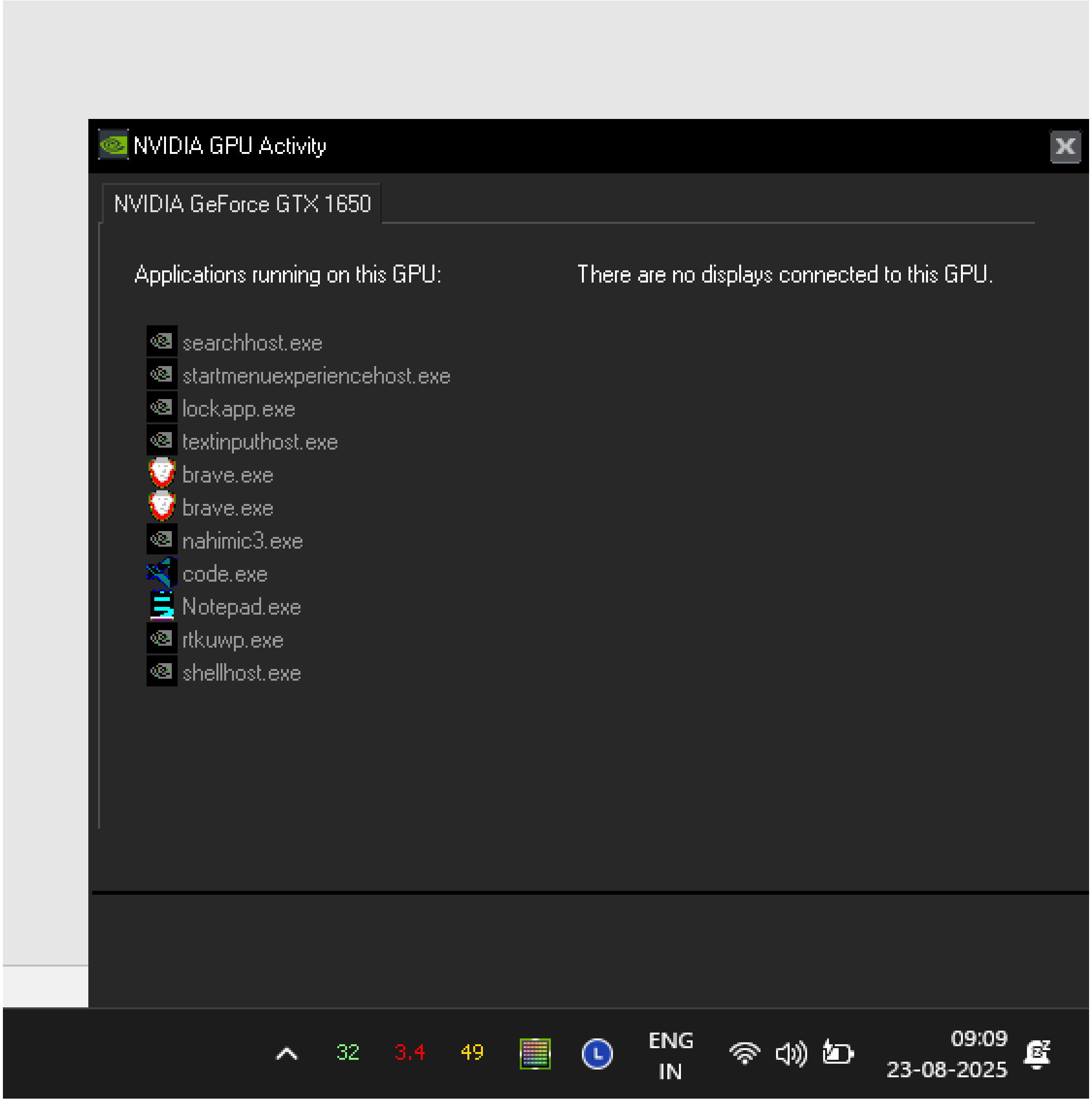
At the end somehow took me 2days but code is done

Ye error tha jo kafi parehsan kar rha tha to iske liye system jake changes karne honge, apne env variable me hardware acc enable kardena and XbG boost karke khuch hota hai nvidia ka wo on kardena



NVIDIA GPU Activity

NVIDIA GeForce GTX 1650

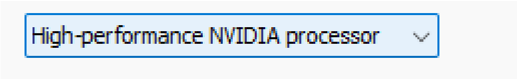
Applications running on this GPU:

There are no displays connected to this GPU.

- searchhost.exe
- startmenuexperiencehost.exe
- lockapp.exe
- textinputhost.exe
- brave.exe
- brave.exe
- nahimic3.exe
- code.exe
- Notepad.exe
- rtkuwp.exe
- shellhost.exe

Aisa try icon ayega

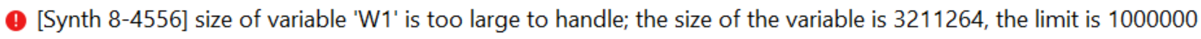
Isme vivado dhundna aisa pop up ayega usme AMD cpu ayega <mera wahi hia> and dusra



High-performance NVIDIA processor

NVIDIA ka GPU joke mera hai

Use select karna and finaly environment variable me jake work limit disable kardena

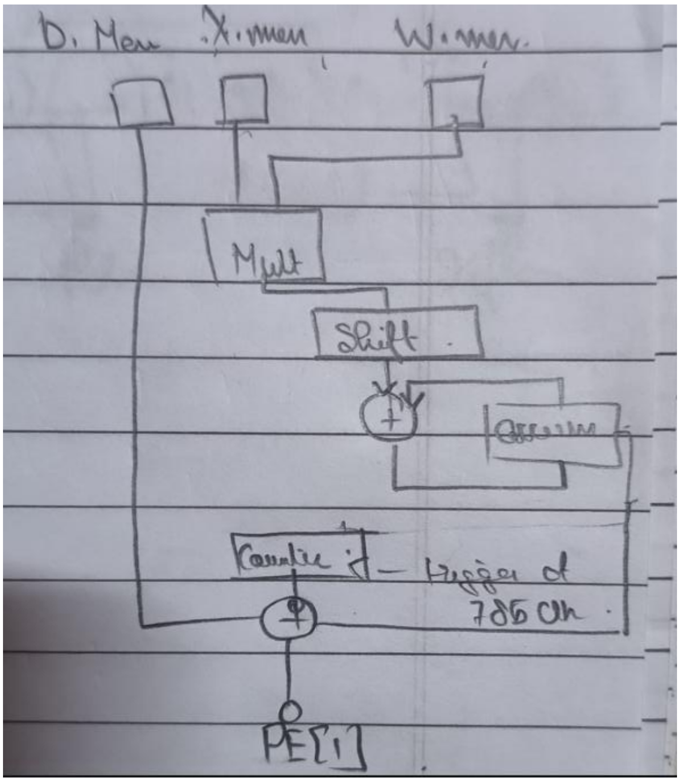


[Synth 8-4556] size of variable 'W1' is too large to handle; the size of the variable is 3211264, the limit is 1000000

Regarding code sab khuch system Verilog me hai (kyuki Verilog me 3D array nahi declare karsakte the & mene abhi naya sikha hai systemverilog :-))

Ok khuch khas batane ko hai nahi more or less scene aisa hia

Sstard pe logic which multiplies two value and adds it to accumulator for N times (counter chaltarhega when it reaches 784clk it triggers) tab wo acc apna out share karega with another adder which adds it to main file.



First operation

W1.mem B1.mem image_0000.mem load

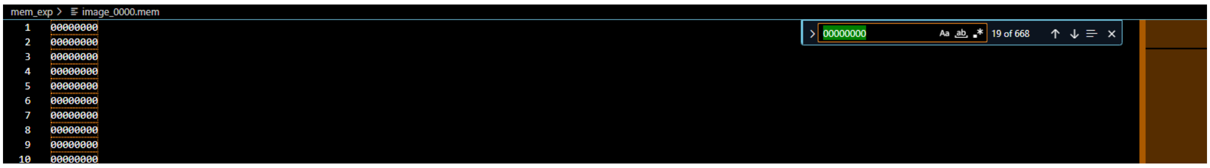
Layer1.sv
(1,784)*(1,128)+(1,128) //128 mac units
785 clk cycles **786 if relu optmizied
relu.sv
relu 129 cycle --X.reg (vector)

Layer2.sv
(1,128)*(128,10)+(1,10)
130 cycles
argmax.sv
10 cycle

Final output vector where one of the value would be 1 that one is output

//possible optimizations for later on

In layer 1 we can have pruning where we select only those values which are non zero



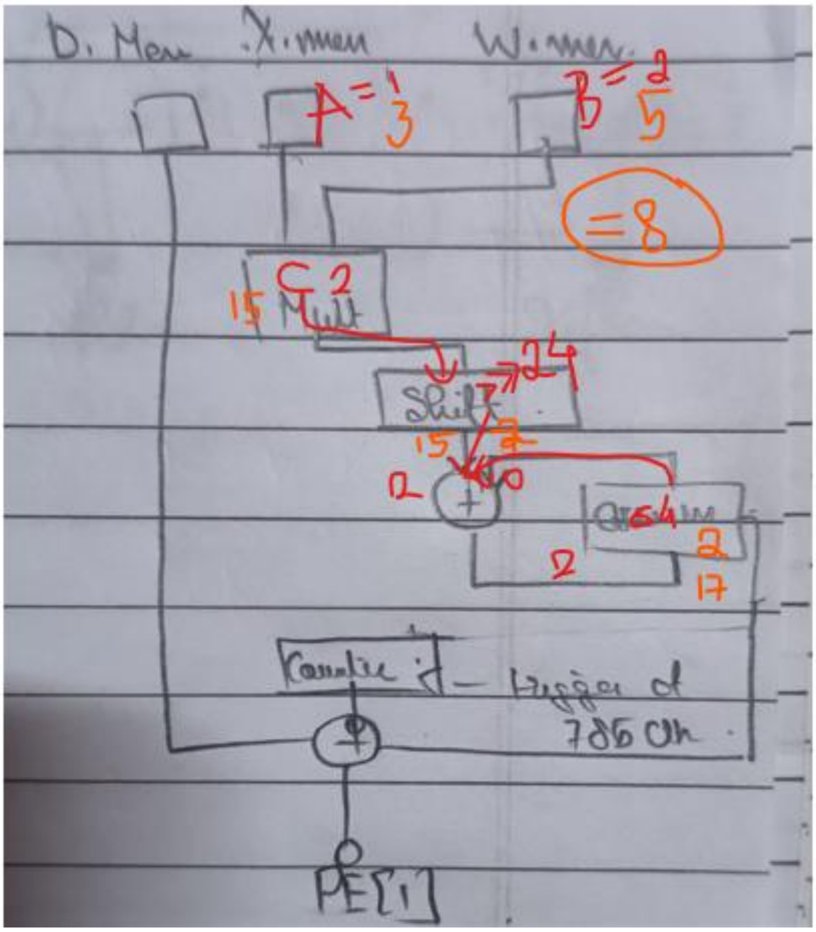
```
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

like in this scenario we have 600+ useless values that’s saving 600 cycles of time ie 6 X 10^-6 seconds

Have relu integrated in PE unit where we could have parallel operation of 128 cycle to 1

Dual port ram mem file, stream pipeline layer1 → layer2 184cycle layer 1 +10 cycle

Best case time 2.5Us



Just
how much time will it take cpu to compute this assuming no systolic array optimizaitions are done

ChatGPT said:

Great question.

Let’s estimate how much time a **CPU** would take to compute the same operations **without systolic array optimizations** — i.e., using regular scalar or vector (but non-pipelined) processing.

We’ll consider each operation:

☐ Step-by-Step CPU Computation Time Estimation

1. Matrix Multiply + Bias:

Shape:

- `(1 × 784)` × `(784 × 128)` + `(1 × 128)`
- Result: `(1 × 128)`

Each output element = dot product of 784-element row and column → 784 multiply-accumulate (MAC) operations.

So:

- Total MACs = `128 × 784 = 100352` operations

Assume:

- A **CPU** takes ~3–5 CPU cycles per MAC (with loop overhead and memory access)
- For this estimate, let's use **4 cycles per MAC**

Then:

- `100352 MACs × 4 cycles = 401,408 cycles`

---

2. ReLU (on 128 values):

Each ReLU is a simple comparison (`if x < 0 then x = 0`)

Assume 1 cycle per ReLU:

- `128 × 1 = 128 cycles`

---

3. Second Matrix Multiply + Bias:

Same shape `(1×784)*(784×128)` but now only takes 130 cycles in hardware.

On CPU, it’s the same size → **same cost** as first matmul.

- Again: `784 × 128 = 100352 MACs × 4 = 401,408 cycles`

---

4. Argmax (on 128 values):

Find the max out of 128 values.

- Sequential comparison: 127 comparisons
- Assume 1–2 cycles per comparison → ~150 cycles

✓ Total CPU Cycles (No Optimizations):

```
MatMul1:  401,408
ReLU:       128
MatMul2:  401,408
Argmax:     150
-----------------------
Total:    ~803,094 CPU cycles
```

☐ How much time is that?

Let’s assume the CPU clock speed:

CPU Clock	Time per cycle	Total time
1 GHz	1 ns	803 µs
2 GHz	0.5 ns	401.5 µs
3 GHz	0.33 ns	267.7 µs

✅ Final Answer:

⚠️ **Without any systolic or SIMD optimizations, a CPU might take between ~250 µs to 800 µs** to compute your full pipeline — **roughly 25x to 75x slower** than your 10.58 µs hardware implementation at 100 MHz.

Let me know your target CPU (e.g., ARM Cortex-A, Intel i7, etc.) or whether to factor in SIMD or cache effects — I can refine the estimate further.

You said:

tell me diference in clk cycles in booth approach like % terms

ChatGPT said:

Great — let’s directly compare the **clock cycle difference** between your **systolic array hardware implementation** and a **CPU-based implementation** (without optimizations), and express that as a **percentage**.

🔢 Step-by-Step Comparison

✅ Systolic Array (Your Hardware):

- **Total cycles:** 1058 (as previously calculated)

✅ CPU (Scalar, No Optimization):

- **Total cycles (estimated):** 803,094

(Based on: 2 × 100,352 MACs × 4 cycles each + ReLU + argmax)

📊 Percent Difference in Clock Cycles

We’ll compute the reduction percentage as:

Reduction %=(CPU cycles−HW cyclesCPU cycles)×100\text{Reduction \%} = \left(\frac{\text{CPU cycles} - \text{HW cycles}}{\text{CPU cycles}} \right) \times 100Reduction %=(CPU cyclesCPU cycles−HW cycles)×100 =(803094−1058803094)×100≈(802036803094)×100≈99.87%= \left(\frac{803094 - 1058}{803094} \right) \times 100 \approx \left(\frac{802036}{803094} \right) \times 100 \approx 99.87\%=(803094803094−1058)×100≈(803094802036)×100≈99.87%

✅ Final Answer:

Your **systolic array hardware** reduces the clock cycles by approximately **99.87%** compared to a scalar **CPU implementation without optimizations**.

That’s a **~780× speedup**, showing the **huge efficiency gain** of using parallel hardware for matrix-heavy workloads like neural nets.

Let me know if you want to compare this to a **SIMD CPU**, **GPU**, or **optimized CPU kernel (like BLAS)** next!