Verilog vs. C for Matrix Multiplication: A Performance Comparison on FPGA and ARM CPU

Xiangbo Cai, Chenxin Zhang
Department of Electrical & Computer Engineering, Michigan State University

***Abstract*—This project investigates the performance difference between hardware and software implementations of matrix multiplication. A 3×3 matrix multiplication operation was implemented using Verilog on an FPGA (BlackBoard) and C programming on a processor. The operation was executed repeatedly to observe and compare processing speeds, and LED indicators were used to visualize the completion of multiple cycles. By monitoring the LED toggle rate and analyzing waveform outputs, the number of clock cycles required by each method was indirectly measured. The comparison highlights the efficiency gap between hardware-based parallel computation and software-based execution. [?]**

***Index Terms*—Matrix Multiplication, FPGA, Verilog, ARM CPU, Performance Analysis**

I. Introduction

Matrix multiplication is a core operation widely used in applications such as signal processing, machine learning, and computer graphics. The performance of this operation can vary significantly depending on whether it is executed in hardware or software, due to differences in parallelism, memory access, and clock speed. To explore these differences, a 3×3 matrix multiplication was implemented in both Verilog and C. The Verilog design was deployed on an FPGA using the BlackBoard development platform, leveraging hardware-level parallelism, while the C program was executed on an ARM processor, representing a typical software-based approach. To measure and compare performance, the multiplication operation was repeated many times, and an LED was toggled at set intervals to provide a visual indication of execution progress. Additionally, waveform simulations were used to verify the timing and correctness of the Verilog implementation. This experimental setup offers a practical means of evaluating the computational efficiency and real-world responsiveness of hardware versus software solutions for matrix operations.

II. Mathematical Background

Let $\mathbf{A} \in R^{N\times N}$ be an upper-triangular matrix whose entries are

$$A_{i,j} = \begin{cases} 1, & i \le j, \\ 0, & i > j. \end{cases}$$

Visually,

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 1 & \cdots & 1 \\ 0 & 0 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

Let $\mathbf{B} \in R^{N\times N}$ be a lower-triangular matrix whose entries are

$$B_{i,j} = \begin{cases} 1, & i \ge j, \\ 0, & i < j. \end{cases}$$

Visually,

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix}.$$

Their product $\mathbf{C} = \mathbf{AB}$ yields the full matrix

$$\mathbf{C} = \begin{pmatrix} N & N-1 & N-2 & \cdots & 1 \\ N-1 & N-1 & N-2 & \cdots & 1 \\ N-2 & N-2 & N-2 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix}.$$

For the special case $N = 3$, we have

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \quad \mathbf{C} = \mathbf{AB} = \begin{pmatrix} 3 & 2 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

III. Verilog Simulation

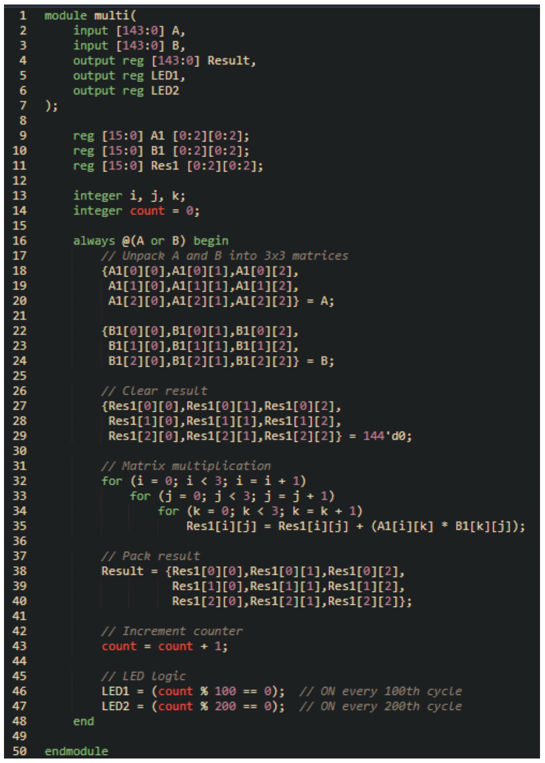
Based on the mathematical formulation of matrix multiplication discussed earlier, we implemented the core logic in Verilog as shown in the `multi` module. This module performs a 3×3 matrix multiplication and uses two LEDs as visual indicators of the operation's progress, making it suitable for hardware-level testing on the FPGA.

In the design, the inputs `A` and `B` are each 144-bit wide vectors, representing two packed 3×3 matrices, where each element is 16 bits. These vectors are unpacked into two-dimensional arrays `A1` and `B1` to allow standard matrix indexing. The result matrix `Res1` is initialized to zero, and the multiplication is carried out using three nested `for` loops, following the standard formula [2]

$$C_{i,j} = \sum_{k=0}^{2} A_{i,k} \cdot B_{k,j}.$$

After computing the result, the matrix is repacked into a single 144-bit vector and assigned to the output `Result`. A counter named `count` tracks how many times the multiplication has been executed. Based on the value of this counter, `LED1` is toggled every 100 cycles and `LED2` every 200 cycles. This provides a simple visual method to monitor execution progress and compare operation speed in real time. [3]

Overall, the Verilog implementation is functionally complete and logically structured. It not only performs matrix multiplication efficiently but also integrates hardware-based indicators, making it well-suited for evaluating and comparing computational performance on FPGA platforms.



```
module multi(
    input [143:0] A,
    input [143:0] B,
    output reg [143:0] Result,
    output reg LED1,
    output reg LED2
);

    reg [15:0] A1 [0:2][0:2];
    reg [15:0] B1 [0:2][0:2];
    reg [15:0] Res1 [0:2][0:2];

    integer i, j, k;
    integer count = 0;

    always @(A or B) begin
        // Unpack A and B into 3x3 matrices
        {A1[0][0],A1[0][1],A1[0][2],
         A1[1][0],A1[1][1],A1[1][2],
         A1[2][0],A1[2][1],A1[2][2]} = A;

        {B1[0][0],B1[0][1],B1[0][2],
         B1[1][0],B1[1][1],B1[1][2],
         B1[2][0],B1[2][1],B1[2][2]} = B;

        // Clear result
        {Res1[0][0],Res1[0][1],Res1[0][2],
         Res1[1][0],Res1[1][1],Res1[1][2],
         Res1[2][0],Res1[2][1],Res1[2][2]} = 144'd0;

        // Matrix multiplication
        for (i = 0; i < 3; i = i + 1)
            for (j = 0; j < 3; j = j + 1)
                for (k = 0; k < 3; k = k + 1)
                    Res1[i][j] = Res1[i][j] + (A1[i][k] * B1[k][j]);

        // Pack result
        Result = {Res1[0][0],Res1[0][1],Res1[0][2],
                  Res1[1][0],Res1[1][1],Res1[1][2],
                  Res1[2][0],Res1[2][1],Res1[2][2]};

        // Increment counter
        count = count + 1;

        // LED logic
        LED1 = (count % 100 == 0);  // ON every 100th cycle
        LED2 = (count % 200 == 0);  // ON every 200th cycle
    end

endmodule
```

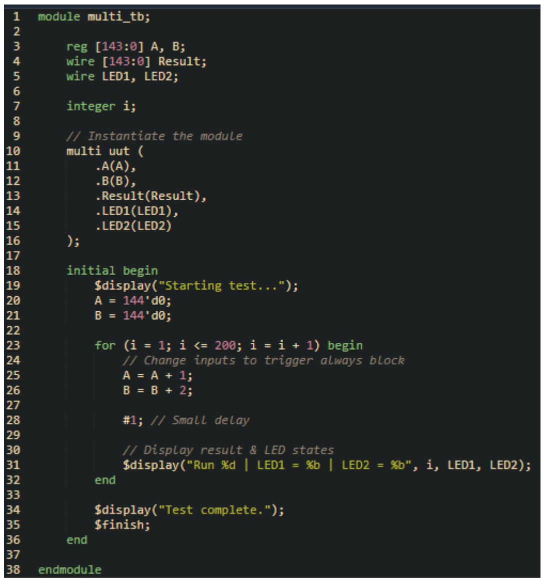
Fig. 1. multi.v code implementation

The related testbench is constructed to simulate and evaluate the functionality of the `multi` module. In this testbench, the inputs `A` and `B` are initialized to zero and incremented in each iteration to continuously trigger the `always` block inside the main module. The testbench runs the multiplication operation 200 times in a loop.

For each iteration, `A` is incremented by 1 and `B` by 2 to ensure the inputs are changing, thereby causing the module to recalculate the matrix product. A short delay (`#1`) is added between iterations to allow time for signal propagation in simulation. During each run, the testbench prints the current iteration number along with the status of the two LEDs, `LED1` and `LED2`.

This setup provides a simple but effective way to validate the toggling behavior of the LEDs, which are designed to light up every 100 and 200 cycles respectively. It also indirectly verifies that the matrix multiplication logic is being repeatedly triggered. Overall, the testbench serves as a functional verification tool to confirm both the computational and control aspects of the Verilog design in simulation.

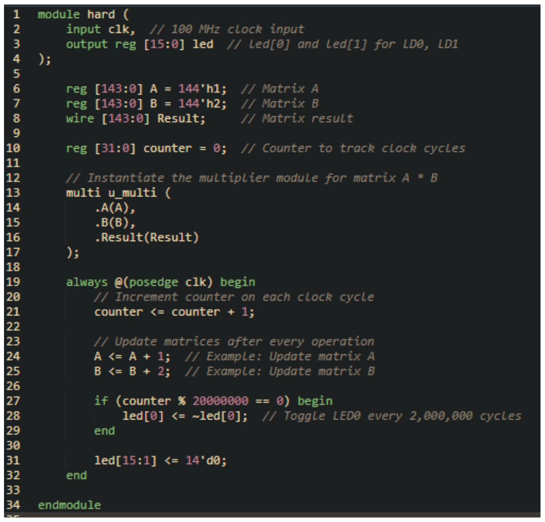
Now we implemented the hardware-level integration by creating a top-level module named `hard`. This module connects the matrix multiplication core (`multi`) to a 100 MHz clock signal, enabling continuous operation in real-time hardware.



```
module multi_tb;

    reg [143:0] A, B;
    wire [143:0] Result;
    wire LED1, LED2;

    integer i;

    // Instantiate the module
    multi uut (
        .A(A),
        .B(B),
        .Result(Result),
        .LED1(LED1),
        .LED2(LED2)
    );

    initial begin
        $display("Starting test...");
        A = 144'd0;
        B = 144'd0;

        for (i = 1; i <= 200; i = i + 1) begin
            // Change inputs to trigger always block
            A = A + 1;
            B = B + 2;

            #1; // Small delay

            // Display result & LED states
            $display("Run %d | LED1 = %b | LED2 = %b", i, LED1, LED2);
        end

        $display("Test complete.");
        $finish;
    end

endmodule
```

Fig. 2. multi_tb.v code implementation

Registers `A` and `B` are initialized and updated on each clock cycle to simulate changing matrix inputs. A 32-bit counter tracks clock cycles, and `led[0]` is toggled every 20 million cycles (approximately every 0.2 seconds) to provide a visual indication of ongoing computation. This setup enables practical deployment of the matrix multiplier on an FPGA board with real-time monitoring through onboard LEDs.



```
module hard (
    input clk,  // 100 MHz clock input
    output reg [15:0] led  // led[0] and led[1] for LD0, LD1
);

    reg [143:0] A = 144'h1;  // Matrix A
    reg [143:0] B = 144'h2;  // Matrix B
    wire [143:0] Result;     // Matrix result

    reg [31:0] counter = 0;  // Counter to track clock cycles

    // Instantiate the multiplier module for matrix A * B
    multi u_multi (
        .A(A),
        .B(B),
        .Result(Result)
    );

    always @(posedge clk) begin
        // Increment counter on each clock cycle
        counter <= counter + 1;

        // Update matrices after every operation
        A <= A + 1;  // Example: Update matrix A
        B <= B + 2;  // Example: Update matrix B

        if (counter % 20000000 == 0) begin
            led[0] <= ~led[0];  // Toggle LED0 every 2,000,000 cycles
        end

        led[15:1] <= 14'd0;
    end

endmodule
```

Fig. 3. hard.v code implementation

IV. Results & Analysis

In the final stage of the experiment, the matrix multiplication module was deployed onto an FPGA for hardware-level testing. To monitor the operation frequency, a 32-bit counter was implemented in the design. The counter increments with every clock cycle, and each time it reaches 20 000 000, it toggles `LED0`, creating a visible blink that represents one complete cycle of repeated matrix multiplication.

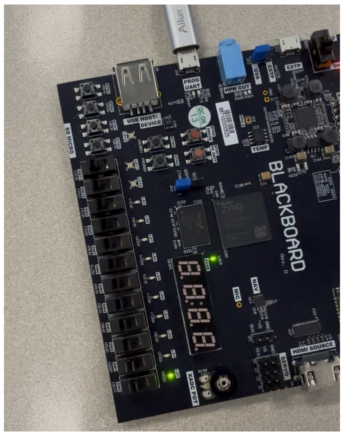


Fig. 4. BlackBoard FPGA showing running matrix-multiplier with LED indicator

To evaluate the system's performance, we recorded a 10-second video of the FPGA board in operation. Upon reviewing the footage, we observed that the LED blinked 27 times during the interval. This indicates that the board completed

$$\mathbf{27 \times 20{,}000{,}000 = 540{,}000{,}000}$$

iterations of the matrix multiplication module over 10 seconds.

The number of operations executed per second is therefore

$$\frac{\mathbf{540{,}000{,}000}}{\mathbf{10}\ \mathrm{s}} = \mathbf{54{,}000{,}000}\ \text{operations/s}$$

This corresponds to an average execution time per operation of

$$\boxed{\mathbf{T = 18.5}\ \mathrm{ns}}$$

Equivalently, the effective operation frequency of the matrix multiplication logic is

$$\boxed{\mathbf{f \approx 54}\ \mathrm{MHz}}$$

Although the system clock runs at 100 MHz, the actual throughput is slightly lower due to sequential and combinational logic delays. Nevertheless, these results demonstrate the high efficiency of the hardware implementation and validate the use of LED indicators as a practical method for performance monitoring. The module achieves a consistent and high operation rate, making it well-suited for accelerating real-time matrix computations in small to medium-scale applications.

V. Compare with C & Conclusion

In this project, we observed a significant performance difference between the Verilog-based hardware implementation and the software-based C implementation of 3×3 matrix multiplication. Although modern CPUs are capable of executing billions of instructions per second, the nature of software execution introduces several limitations when compared to dedicated hardware logic on an FPGA.

First, C programs are executed sequentially by the processor, with operations such as multiplication and addition occurring one after another unless explicitly parallelized. Even with compiler-level optimizations or SIMD extensions, the degree of parallelism remains limited by the CPU's architecture. In contrast, the Verilog implementation allows true parallel execution: the multiply-accumulate operations for different matrix elements can be processed simultaneously within the FPGA, leveraging its hardware-level parallelism and resulting in significantly higher throughput.

Second, Verilog enables precise control over timing and logic at the hardware level. Each computational step is defined explicitly in terms of clock cycles, providing predictable and consistent performance. C code, on the other hand, is subject to various runtime factors such as instruction scheduling, memory access delays, cache misses, and operating-system overhead, all of which introduce variability and reduce overall efficiency.

Third, on our specific test platform, the C code runs on a relatively low-power ARM processor embedded within the BlackBoard system, which is not optimized for high-performance arithmetic workloads. Without dedicated hardware acceleration or optimized numerical libraries, the processor handles each instruction sequentially and relies heavily on memory access, further slowing down performance.

In conclusion, the experiment confirms that hardware-based matrix multiplication using Verilog on an FPGA offers significant performance advantages over software-based C implementations. The FPGA design not only achieves higher execution speed but also ensures deterministic behavior, making it more suitable for real-time or embedded systems. This validates the effectiveness of FPGA acceleration for computationally intensive and repetitive tasks such as matrix multiplication.

References

[1] M. Meier, "ECE331 Honors Project Instruction Note," Department of Electrical & Computer Engineering, Michigan State University, Fall 2025.
[2] ARM Ltd., "Example – Matrix Multiplication Using NEON," Arm Developer Documentation, Issue 03, 2020. :contentReference[oaicite:0]index=0
[3] N. Bean, "Matrix Multiplication Speed Test: Python vs. C vs. FPGA (SystemVerilog) Benchmarks," *Medium*, Jan. 21, 2025. :contentReference[oaicite:1]index=1