Novel Low Power 8*8 Booth Multiplier Verilog Implementation

Saurabh Gaglani
Department of Computer Science, California State University Northridge

COMP 621 Computer Arithmetic Design

Dr Shahnam Mirzaei

16/12/2024

ALL VERILOG CODE ATTACHED WITH REPORT

IMPLEMENTATION STEPS

Booth Multiplier Design:

- The multiplier implements radix-4 Booth encoding to minimize partial product computation.
- The 8-bit multiplier and multiplicand are processed to generate 4 partial products using Booth encoding logic.
- Stage-1:
  - The least significant 2 bits (LSBs) of the multiplier are combined with an assumed 0 to generate the first Booth code.
  - Based on the Booth code:
    - \texttt{3'b001, 3'b010}: Partial product = Multiplicand.
    - \texttt{3'b011}: Partial product = Multiplicand shifted left (×2).
    - \texttt{3'b100}: Partial product = -2 × Multiplicand.
    - \texttt{3'b101, 3'b110}: Partial product = -Multiplicand.
  - The 2 LSBs from Stage-1 are directly assigned to the product register.

Partial Product Generation:

- For Stages 2, 3, and 4, Booth codes are generated using groups of 3 multiplier bits
- Partial products are computed based on the Booth code and shifted left by 2
- LSBs of each partial product are assigned to the product register to ensure accurate summation.

Summation of Partial Products:

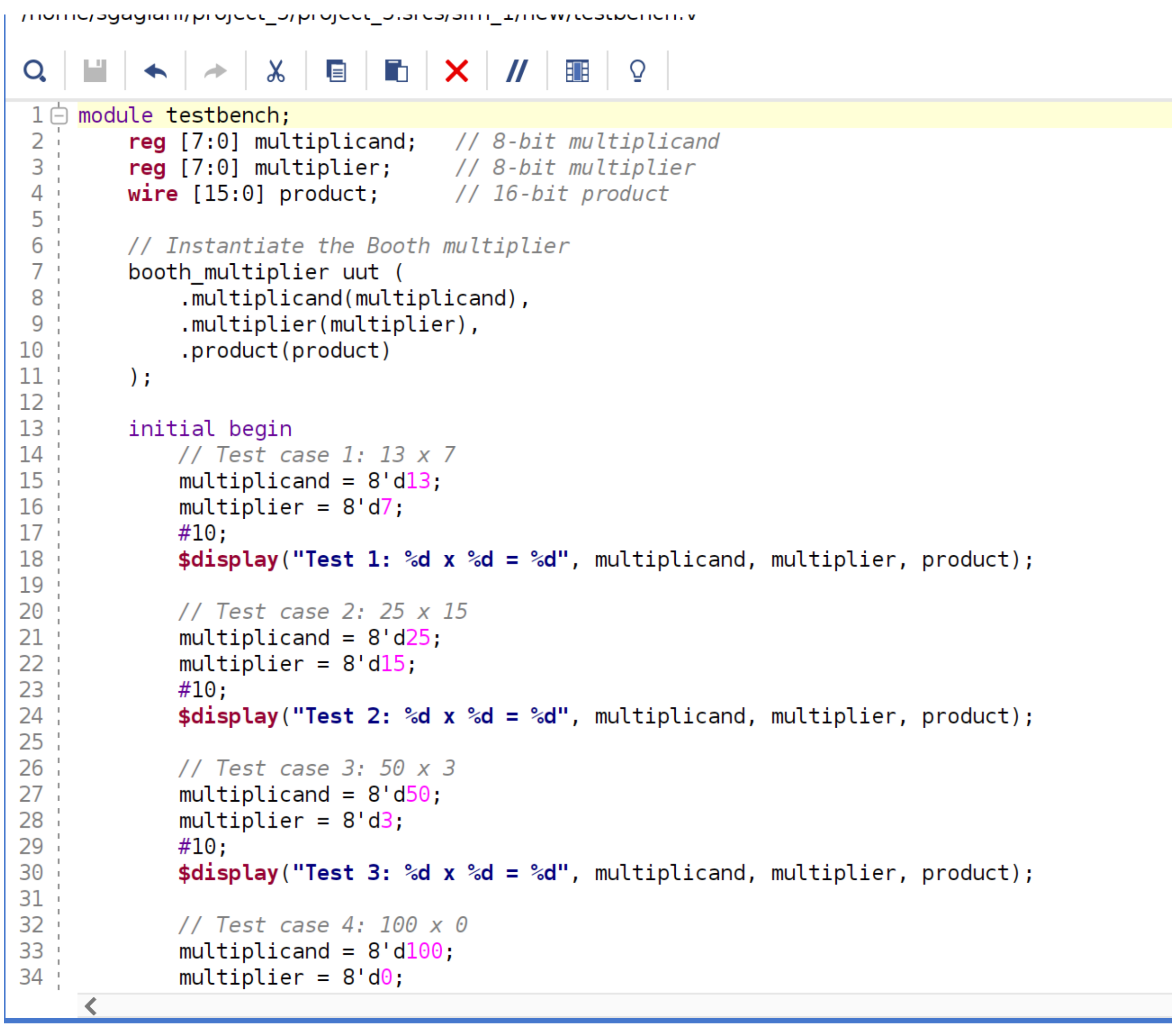
- The higher 14 bits of all partial products are summed
- LSBs are already handled in each stage, optimizing computation and eliminating redundant propagation.

Testbench Verification:

- The testbench runs five test cases to verify the multiplier's correctness:
  - 13×7=9113 \times 7 = 9113×7=91 (0x5B in HEX)
  - 25×15=37525 \times 15 = 37525×15=375 (0x177 in HEX)
  - 50×3=15050 \times 3 = 15050×3=150 (0x96 in HEX)
  - 100×0=0100 \times 0 = 0100×0=0 (0x00 in HEX)

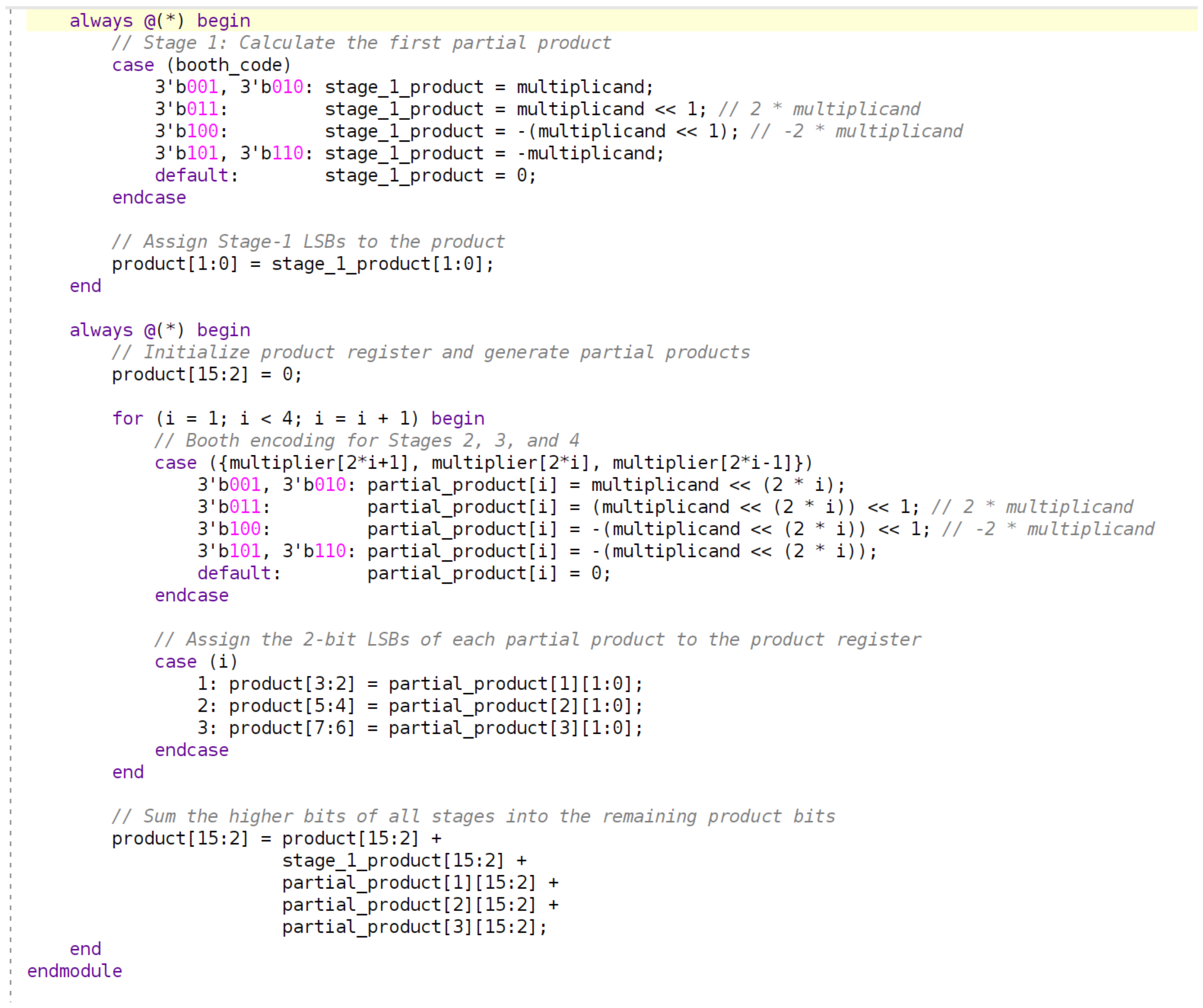
   - 8×2=168 \times 2 = 168×2=16 (0x10 in HEX).
- Results are displayed using \texttt{$display} in both decimal and hexadecimal formats.

Testbench



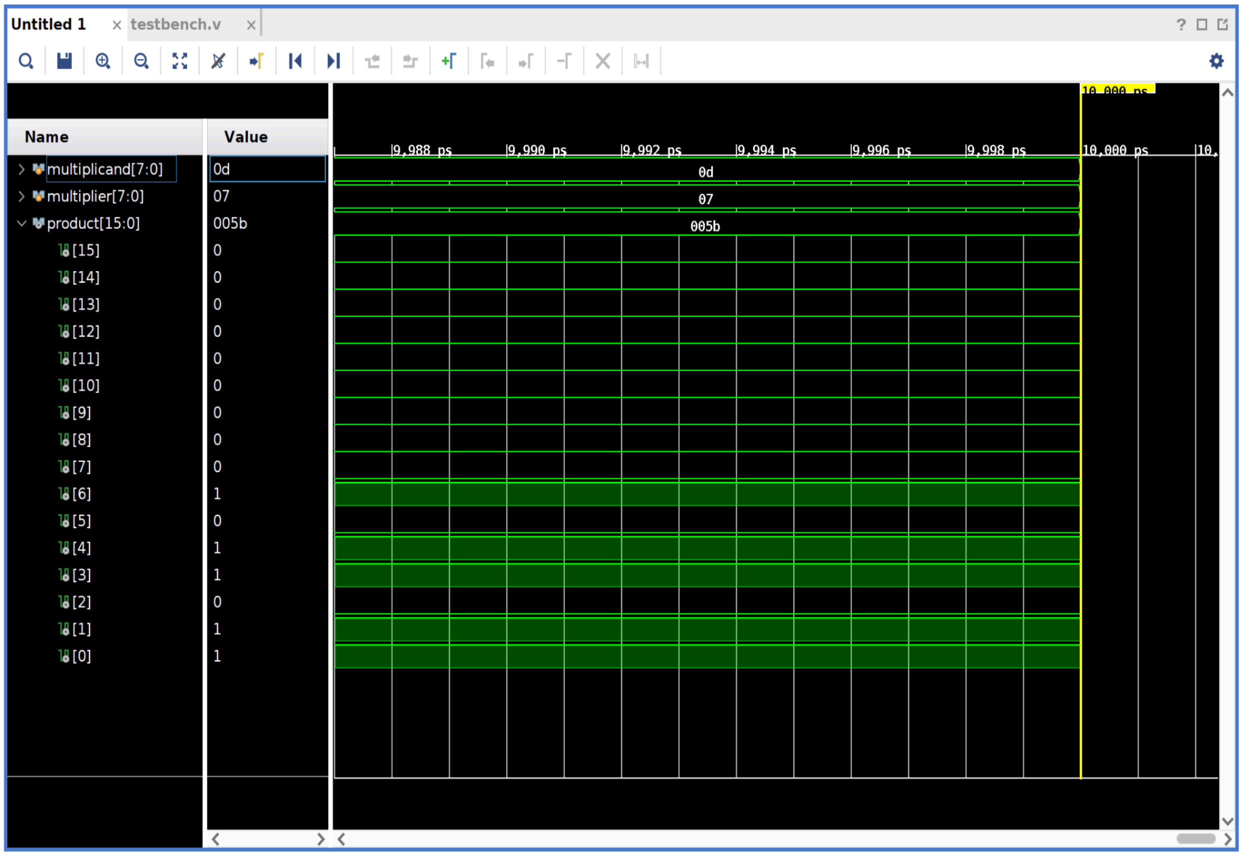
```
module testbench;
    reg [7:0] multiplicand;   // 8-bit multiplicand
    reg [7:0] multiplier;     // 8-bit multiplier
    wire [15:0] product;      // 16-bit product

    // Instantiate the Booth multiplier
    booth_multiplier uut (
        .multiplicand(multiplicand),
        .multiplier(multiplier),
        .product(product)
    );

    initial begin
        // Test case 1: 13 x 7
        multiplicand = 8'd13;
        multiplier = 8'd7;
        #10;
        $display("Test 1: %d x %d = %d", multiplicand, multiplier, product);

        // Test case 2: 25 x 15
        multiplicand = 8'd25;
        multiplier = 8'd15;
        #10;
        $display("Test 2: %d x %d = %d", multiplicand, multiplier, product);

        // Test case 3: 50 x 3
        multiplicand = 8'd50;
        multiplier = 8'd3;
        #10;
        $display("Test 3: %d x %d = %d", multiplicand, multiplier, product);

        // Test case 4: 100 x 0
        multiplicand = 8'd100;
        multiplier = 8'd0;
```

Algorithm


```
    always @(*) begin
        // Stage 1: Calculate the first partial product
        case (booth_code)
            3'b001, 3'b010: stage_1_product = multiplicand;
            3'b011:         stage_1_product = multiplicand << 1; // 2 * multiplicand
            3'b100:         stage_1_product = -(multiplicand << 1); // -2 * multiplicand
            3'b101, 3'b110: stage_1_product = -multiplicand;
            default:        stage_1_product = 0;
        endcase

        // Assign Stage-1 LSBs to the product
        product[1:0] = stage_1_product[1:0];
    end

    always @(*) begin
        // Initialize product register and generate partial products
        product[15:2] = 0;

        for (i = 1; i < 4; i = i + 1) begin
            // Booth encoding for Stages 2, 3, and 4
            case ({multiplier[2*i+1], multiplier[2*i], multiplier[2*i-1]})
                3'b001, 3'b010: partial_product[i] = multiplicand << (2 * i);
                3'b011:         partial_product[i] = (multiplicand << (2 * i)) << 1; // 2 * multiplicand
                3'b100:         partial_product[i] = -(multiplicand << (2 * i)) << 1; // -2 * multiplicand
                3'b101, 3'b110: partial_product[i] = -(multiplicand << (2 * i));
                default:        partial_product[i] = 0;
            endcase

            // Assign the 2-bit LSBs of each partial product to the product register
            case (i)
                1: product[3:2] = partial_product[1][1:0];
                2: product[5:4] = partial_product[2][1:0];
                3: product[7:6] = partial_product[3][1:0];
            endcase
        end

        // Sum the higher bits of all stages into the remaining product bits
        product[15:2] = product[15:2] +
                        stage_1_product[15:2] +
                        partial_product[1][15:2] +
                        partial_product[2][15:2] +
                        partial_product[3][15:2];
    end
endmodule
```

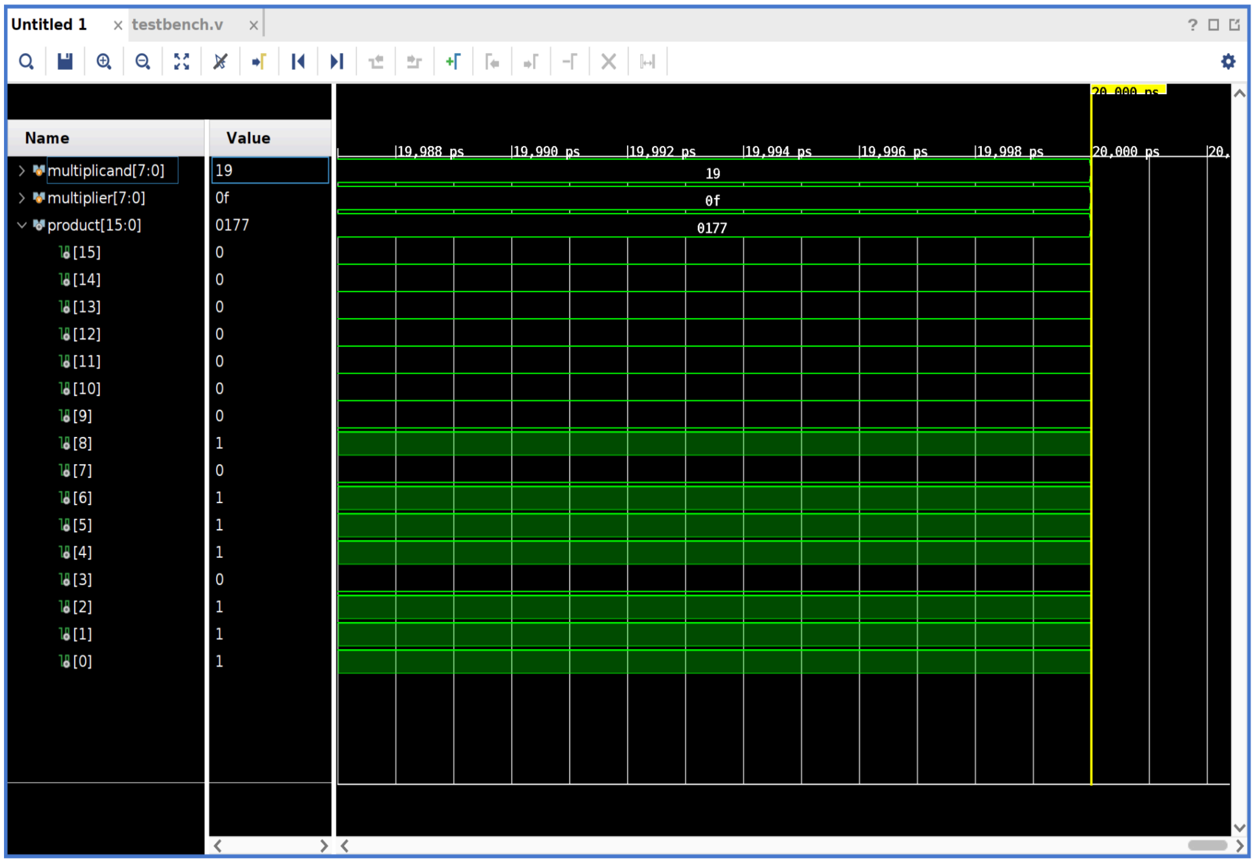
Evaluation - (Results in HEX)

1. **$(0d*07 = 005b)_{16}$ Is the same as $(13*7 = 91)_{10}$**



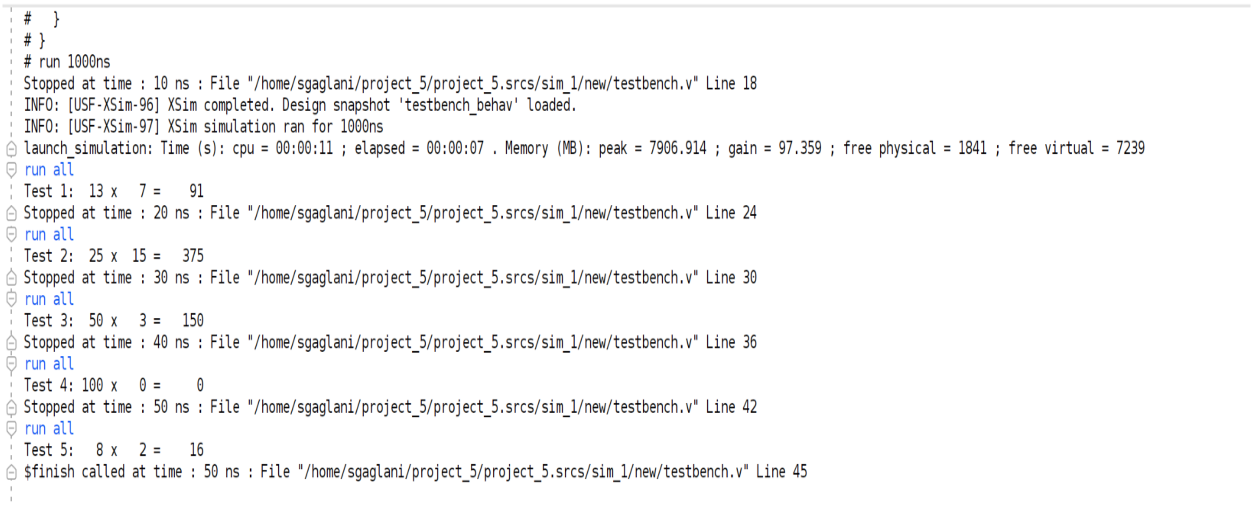
Name	Value
multiplicand[7:0]	0d
multiplier[7:0]	07
product[15:0]	005b
[15]	0
[14]	0
[13]	0
[12]	0
[11]	0
[10]	0
[9]	0
[8]	0
[7]	0
[6]	1
[5]	0
[4]	1
[3]	1
[2]	0
[1]	1
[0]	1

2. ￼(￼19*0f = 0177)$_{16}$ IS THE SAME AS (25*15 = 375)$_{10}$



Name	Value
multiplicand[7:0]	19
multiplier[7:0]	0f
product[15:0]	0177
[15]	0
[14]	0
[13]	0
[12]	0
[11]	0
[10]	0
[9]	0
[8]	1
[7]	0
[6]	1
[5]	1
[4]	1
[3]	0
[2]	1
[1]	1
[0]	1

3. All test cases from the testbench in decimal



```
#   }
# }
# run 1000ns
Stopped at time : 10 ns : File "/home/sgaglani/project_5/project_5.srcs/sim_1/new/testbench.v" Line 18
INFO: [USF-XSim-96] XSim completed. Design snapshot 'testbench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:11 ; elapsed = 00:00:07 . Memory (MB): peak = 7906.914 ; gain = 97.359 ; free physical = 1841 ; free virtual = 7239
run all
Test 1:  13 x   7 =    91
Stopped at time : 20 ns : File "/home/sgaglani/project_5/project_5.srcs/sim_1/new/testbench.v" Line 24
run all
Test 2:  25 x  15 =   375
Stopped at time : 30 ns : File "/home/sgaglani/project_5/project_5.srcs/sim_1/new/testbench.v" Line 30
run all
Test 3:  50 x   3 =   150
Stopped at time : 40 ns : File "/home/sgaglani/project_5/project_5.srcs/sim_1/new/testbench.v" Line 36
run all
Test 4: 100 x   0 =     0
Stopped at time : 50 ns : File "/home/sgaglani/project_5/project_5.srcs/sim_1/new/testbench.v" Line 42
run all
Test 5:   8 x   2 =    16
$finish called at time : 50 ns : File "/home/sgaglani/project_5/project_5.srcs/sim_1/new/testbench.v" Line 45
```

ALL VERILOG CODE ATTACHED WITH REPORT