Report for Wallace Tree Multiplier 8x8

Wallace Tree Level Diagram

A **Wallace multiplier** is a hardware implementation of a binary multiplier, a digital circuit that multiplies two integers. It uses a selection of full and half adders (the **Wallace tree** or **Wallace reduction**) to sum partial products in stages until two numbers are left. Wallace multipliers reduce as much as possible on each layer, whereas Dadda multipliers try to minimize the required number of gates by postponing the reduction to the upper layers.

The Wallace tree has three steps:

1. Multiply each bit of one of the arguments, by each bit of the other.
2. Reduce the number of partial products to two by layers of full and half adders.
3. Group the wires in two numbers, and add them with a conventional adder.

Wallace Tree Diagram for 8x8 Multiplier

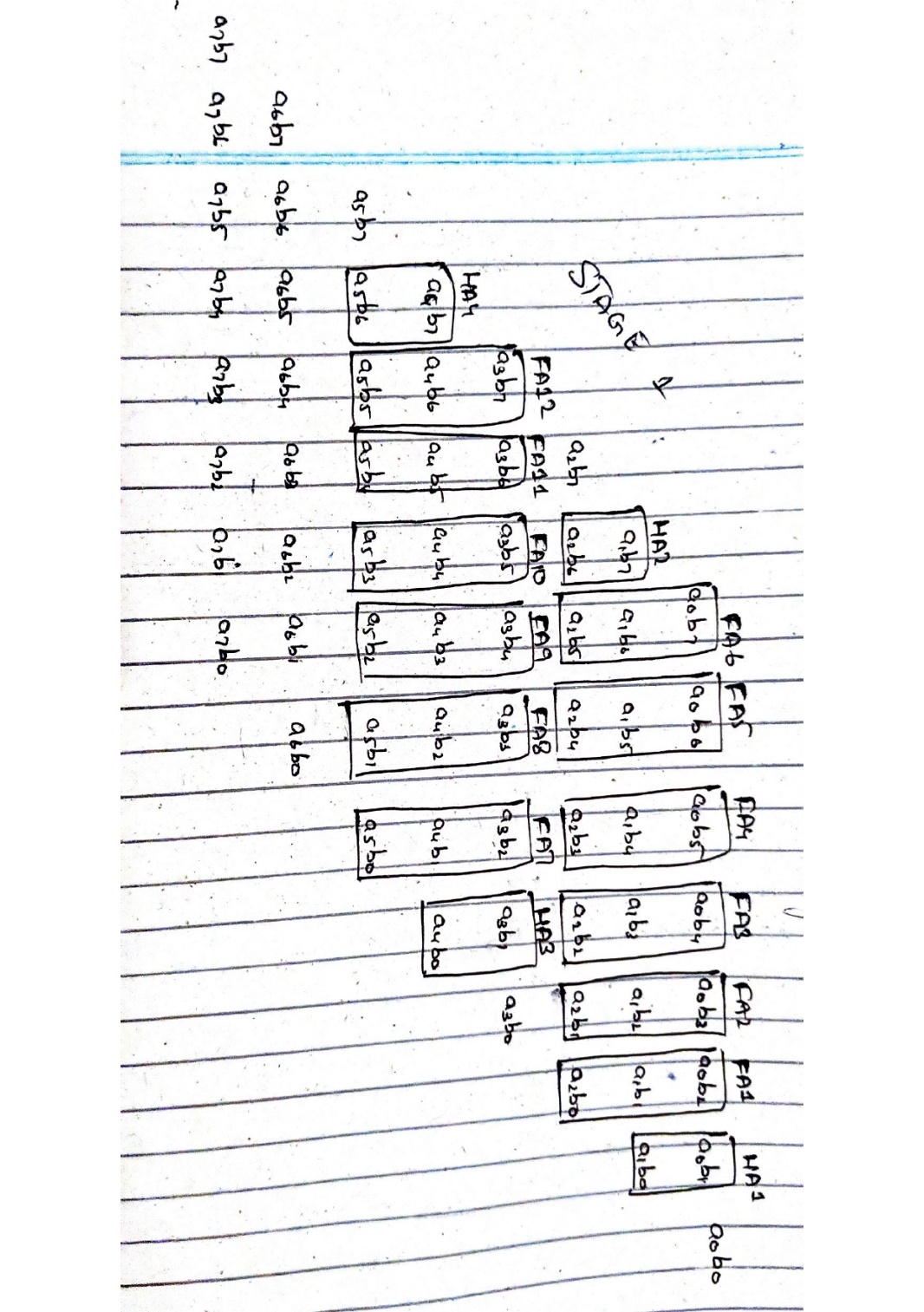


Figure 1: Stage One

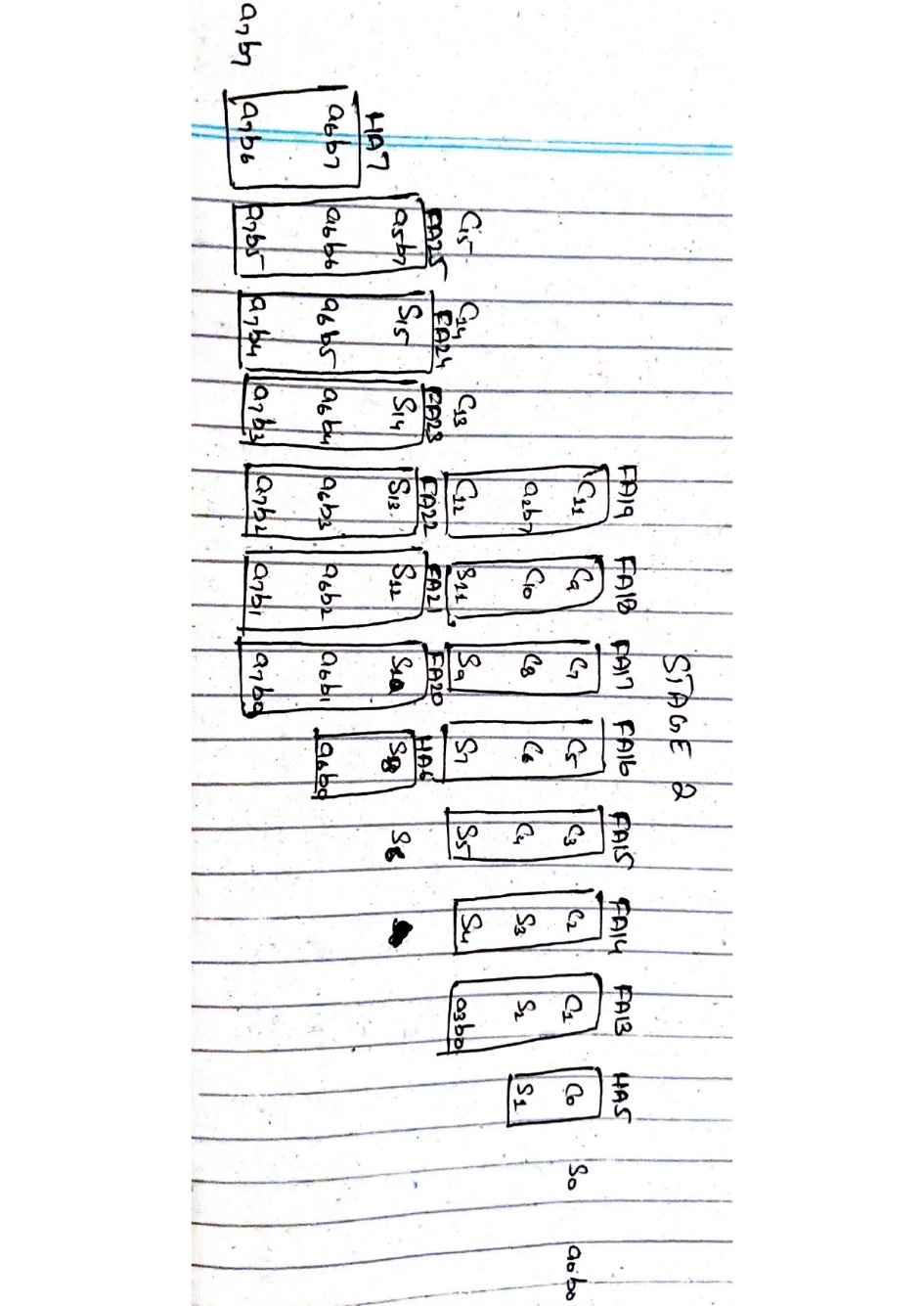


Figure 2: Stage Two

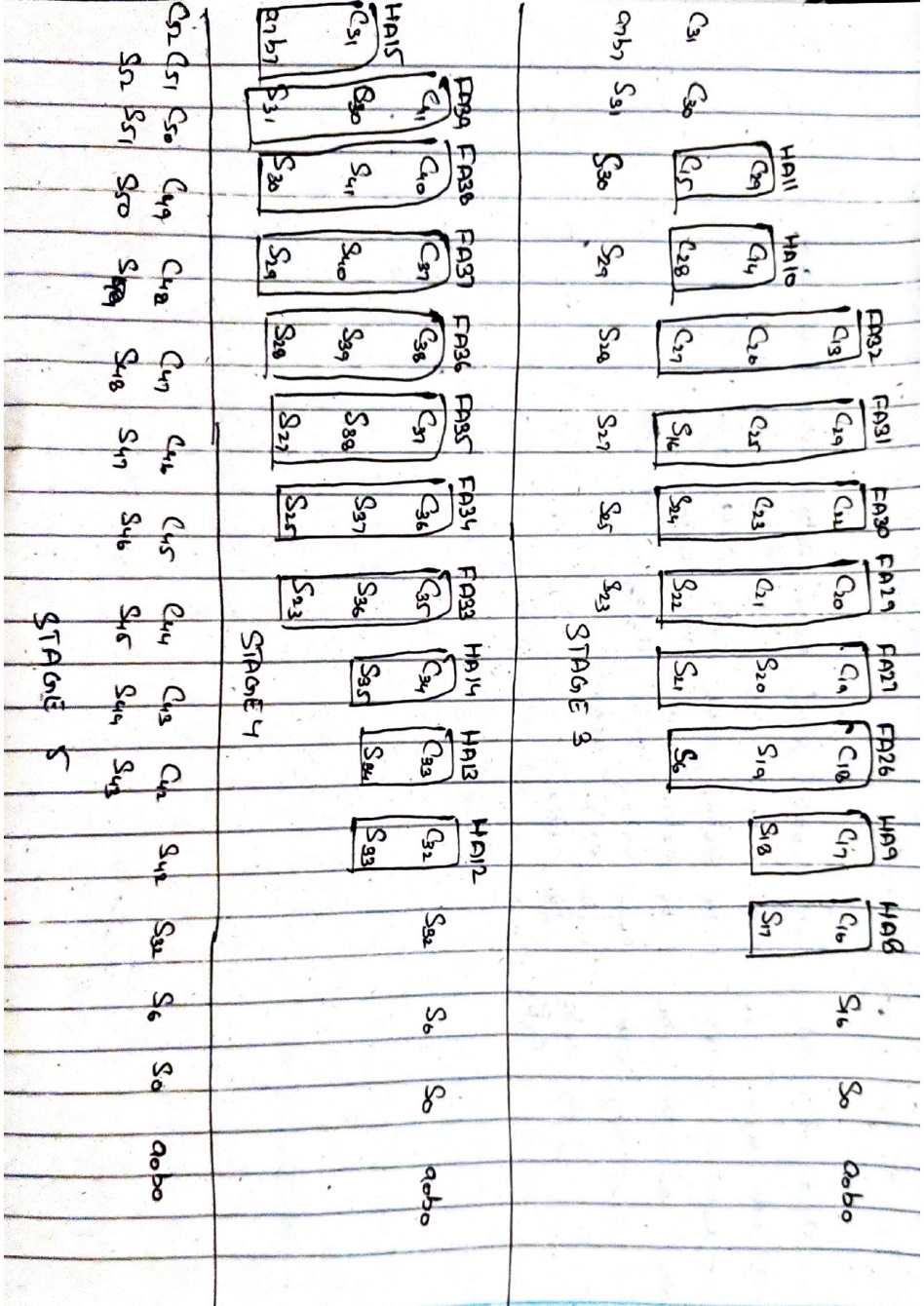


Figure 3: Stage Three Four and Five

Now, Wallace Multiplier needs to be implemented in gate level so we are going to be using lots of Full Adders and Half adders to achieve this goal.

Wallace Adder is fast multiplier but it requires a lot of hardware which is the drawback of the Wallace multiplier.

The Verilog Code and Testbench are as follows for the Wallace multiplier.

**Full Adder Verilog Code:**

|  |
| --- |
| module FA (  output cout,sum,  input a,b,cin  );  assign sum = a ^ b ^ cin;  assign cout = (a & b) | (b & cin) | (a & cin);  endmodule |

**Half Adder Verilog Code:**

|  |
| --- |
| module HA (    output cout,sum,  input a, b  );  assign sum = a ^ b;  assign cout = a & b;  endmodule |

**Wallace Tree Multiplier Verilog Code:**

|  |
| --- |
| module wallace\_8x8 (output[15:0] result,  input[7:0] a,  input[7:0] b);  wire[7:0] wallaceTree[7:0];    genvar i, j;  generate  for (i = 0; i < 8; i = i + 1) begin  for (j = 0; j < 8; j = j + 1) begin  assign wallaceTree[i][j] = a[i] & b[j];  end  end  endgenerate  assign result[0] = wallaceTree[0][0];  // result[1]  wire result1\_c;  HA result1\_HA\_1(result1\_c, result[1], wallaceTree[0][1], wallaceTree[1][0]);  // result[2]  wire result2\_c\_temp\_1, result2\_c, result2\_temp\_1;  FA result2\_FA\_1(result2\_c\_temp\_1, result2\_temp\_1, wallaceTree[0][2], wallaceTree[1][1], result1\_c);  HA result2\_HA\_1(result2\_c, result[2], wallaceTree[2][0], result2\_temp\_1);  // result[3]  wire result3\_c\_temp\_1, result3\_c\_temp\_2, result3\_c, result3\_temp\_1, result3\_temp\_2;  FA result3\_FA\_1(result3\_c\_temp\_1, result3\_temp\_1, wallaceTree[0][3], wallaceTree[1][2], result2\_c);  FA result3\_FA\_2(result3\_c\_temp\_2, result3\_temp\_2, wallaceTree[2][1], result3\_temp\_1, result2\_c\_temp\_1);  HA result3\_HA\_1(result3\_c, result[3], wallaceTree[3][0], result3\_temp\_2);  // result[4]  wire result4\_c\_temp\_1, result4\_c\_temp\_2, result4\_c\_temp\_3, result4\_c, result4\_temp\_1, result4\_temp\_2, result4\_temp\_3;  FA result4\_FA\_1(result4\_c\_temp\_1, result4\_temp\_1, wallaceTree[0][4], wallaceTree[1][3], result3\_c);  FA result4\_FA\_2(result4\_c\_temp\_2, result4\_temp\_2, wallaceTree[2][2], result4\_temp\_1, result3\_c\_temp\_1);  FA result4\_FA\_3(result4\_c\_temp\_3, result4\_temp\_3, wallaceTree[3][1], result4\_temp\_2, result3\_c\_temp\_2);  HA result4\_HA\_1(result4\_c, result[4], wallaceTree[4][0], result4\_temp\_3);  // result[5]  wire result5\_c\_temp\_1, result5\_c\_temp\_2, result5\_c\_temp\_3, result5\_c\_temp\_4, result5\_c, result5\_temp\_1, result5\_temp\_2, result5\_temp\_3, result5\_temp\_4;  FA result5\_FA\_1(result5\_c\_temp\_1, result5\_temp\_1, wallaceTree[0][5], wallaceTree[1][4], result4\_c);  FA result5\_FA\_2(result5\_c\_temp\_2, result5\_temp\_2, wallaceTree[2][3], result5\_temp\_1, result4\_c\_temp\_1);  FA result5\_FA\_3(result5\_c\_temp\_3, result5\_temp\_3, wallaceTree[3][2], result5\_temp\_2, result4\_c\_temp\_2);  FA result5\_FA\_4(result5\_c\_temp\_4, result5\_temp\_4, wallaceTree[4][1], result5\_temp\_3, result4\_c\_temp\_3);  HA result5\_HA\_1(result5\_c, result[5], wallaceTree[5][0], result5\_temp\_4);  // result[6]  wire result6\_c\_temp\_1, result6\_c\_temp\_2, result6\_c\_temp\_3, result6\_c\_temp\_4, result6\_c\_temp\_5, result6\_c, result6\_temp\_1, result6\_temp\_2, result6\_temp\_3, result6\_temp\_4, result6\_temp\_5;  FA result6\_FA\_1(result6\_c\_temp\_1, result6\_temp\_1, wallaceTree[0][6], wallaceTree[1][5], result5\_c);  FA result6\_FA\_2(result6\_c\_temp\_2, result6\_temp\_2, wallaceTree[2][4], result6\_temp\_1, result5\_c\_temp\_1);  FA result6\_FA\_3(result6\_c\_temp\_3, result6\_temp\_3, wallaceTree[3][3], result6\_temp\_2, result5\_c\_temp\_2);  FA result6\_FA\_4(result6\_c\_temp\_4, result6\_temp\_4, wallaceTree[4][2], result6\_temp\_3, result5\_c\_temp\_3);  FA result6\_FA\_5(result6\_c\_temp\_5, result6\_temp\_5, wallaceTree[5][1], result6\_temp\_4, result5\_c\_temp\_4);  HA result6\_HA\_1(result6\_c, result[6], wallaceTree[6][0], result6\_temp\_5);  // result[7]  wire result7\_c\_temp\_1, result7\_c\_temp\_2, result7\_c\_temp\_3, result7\_c\_temp\_4, result7\_c\_temp\_5, result7\_c\_temp\_6, result7\_c, result7\_temp\_1, result7\_temp\_2, result7\_temp\_3, result7\_temp\_4, result7\_temp\_5, result7\_temp\_6;  FA result7\_FA\_1(result7\_c\_temp\_1, result7\_temp\_1, wallaceTree[0][7], wallaceTree[1][6], result6\_c);  FA result7\_FA\_2(result7\_c\_temp\_2, result7\_temp\_2, wallaceTree[2][5], result7\_temp\_1, result6\_c\_temp\_1);  FA result7\_FA\_3(result7\_c\_temp\_3, result7\_temp\_3, wallaceTree[3][4], result7\_temp\_2, result6\_c\_temp\_2);  FA result7\_FA\_4(result7\_c\_temp\_4, result7\_temp\_4, wallaceTree[4][3], result7\_temp\_3, result6\_c\_temp\_3);  FA result7\_FA\_5(result7\_c\_temp\_5, result7\_temp\_5, wallaceTree[5][2], result7\_temp\_4, result6\_c\_temp\_4);  FA result7\_FA\_6(result7\_c\_temp\_6, result7\_temp\_6, wallaceTree[6][1], result7\_temp\_5, result6\_c\_temp\_5);  HA result7\_HA\_1(result7\_c, result[7], wallaceTree[7][0], result7\_temp\_6);  // result[8]  wire result8\_c\_temp\_1, result8\_c\_temp\_2, result8\_c\_temp\_3, result8\_c\_temp\_4, result8\_c\_temp\_5, result8\_c\_temp\_6, result8\_c, result8\_temp\_1, result8\_temp\_2, result8\_temp\_3, result8\_temp\_4, result8\_temp\_5, result8\_temp\_6;  FA result8\_FA\_1(result8\_c\_temp\_1, result8\_temp\_1, wallaceTree[1][7], wallaceTree[2][6], result7\_c);  FA result8\_FA\_2(result8\_c\_temp\_2, result8\_temp\_2, wallaceTree[3][5], result8\_temp\_1, result7\_c\_temp\_1);  FA result8\_FA\_3(result8\_c\_temp\_3, result8\_temp\_3, wallaceTree[4][4], result8\_temp\_2, result7\_c\_temp\_2);  FA result8\_FA\_4(result8\_c\_temp\_4, result8\_temp\_4, wallaceTree[5][3], result8\_temp\_3, result7\_c\_temp\_3);  FA result8\_FA\_5(result8\_c\_temp\_5, result8\_temp\_5, wallaceTree[6][2], result8\_temp\_4, result7\_c\_temp\_4);  FA result8\_FA\_6(result8\_c\_temp\_6, result8\_temp\_6, wallaceTree[7][1], result8\_temp\_5, result7\_c\_temp\_5);  HA result8\_HA\_1(result8\_c, result[8], result8\_temp\_6, result7\_c\_temp\_6);  // result[9]  wire result9\_c\_temp\_1, result9\_c\_temp\_2, result9\_c\_temp\_3, result9\_c\_temp\_4, result9\_c, result9\_temp\_1, result9\_temp\_2, result9\_temp\_3, result9\_temp\_4;  FA result9\_FA\_1(result9\_c\_temp\_1, result9\_temp\_1, wallaceTree[2][7], wallaceTree[3][6], result8\_c);  FA result9\_FA\_2(result9\_c\_temp\_2, result9\_temp\_2, wallaceTree[4][5], result9\_temp\_1, result8\_c\_temp\_1);  FA result9\_FA\_3(result9\_c\_temp\_3, result9\_temp\_3, wallaceTree[5][4], result9\_temp\_2, result8\_c\_temp\_2);  FA result9\_FA\_4(result9\_c\_temp\_4, result9\_temp\_4, wallaceTree[6][3], result9\_temp\_3, result8\_c\_temp\_3);  FA result9\_FA\_5(result9\_c\_temp\_5, result9\_temp\_5, wallaceTree[7][2], result9\_temp\_4, result8\_c\_temp\_4);  FA result9\_FA\_6(result9\_c, result[9], result9\_temp\_5, result8\_c\_temp\_5, result8\_c\_temp\_6);  // result[10]  wire result10\_c\_temp\_1, result10\_c\_temp\_2, result10\_c\_temp\_3, result10\_c, result10\_temp\_1, result10\_temp\_2, result10\_temp\_3;  FA result10\_FA\_1(result10\_c\_temp\_1, result10\_temp\_1, wallaceTree[3][7], wallaceTree[4][6], result9\_c);  FA result10\_FA\_2(result10\_c\_temp\_2, result10\_temp\_2, wallaceTree[5][5], result10\_temp\_1, result9\_c\_temp\_1);  FA result10\_FA\_3(result10\_c\_temp\_3, result10\_temp\_3, wallaceTree[6][4], result10\_temp\_2, result9\_c\_temp\_2);  FA result10\_FA\_4(result10\_c\_temp\_4, result10\_temp\_4, wallaceTree[7][3], result10\_temp\_3, result9\_c\_temp\_3);  FA result10\_FA\_5(result10\_c, result[10], result10\_temp\_4, result9\_c\_temp\_4, result9\_c\_temp\_5);  // result[11]  wire result11\_c\_temp\_1, result11\_c\_temp\_2, result11\_c, result11\_temp\_1, result11\_temp\_2;  FA result11\_FA\_1(result11\_c\_temp\_1, result11\_temp\_1, wallaceTree[4][7], wallaceTree[5][6], result10\_c);  FA result11\_FA\_2(result11\_c\_temp\_2, result11\_temp\_2, wallaceTree[6][5], result11\_temp\_1, result10\_c\_temp\_1);  FA result11\_FA\_3(result11\_c\_temp\_3, result11\_temp\_3, wallaceTree[7][4], result11\_temp\_2, result10\_c\_temp\_2);  FA result11\_FA\_4(result11\_c, result[11], result11\_temp\_3, result10\_c\_temp\_3, result10\_c\_temp\_4);  // result[12]  wire result12\_c\_temp\_1, result12\_c, result12\_temp\_1;  FA result12\_FA\_1(result12\_c\_temp\_1, result12\_temp\_1, wallaceTree[5][7], wallaceTree[6][6], result11\_c);  FA result12\_FA\_2(result12\_c\_temp\_2, result12\_temp\_2, wallaceTree[7][5], result12\_temp\_1, result11\_c\_temp\_1);  FA result12\_FA\_3(result12\_c, result[12], result12\_temp\_2, result11\_c\_temp\_2, result11\_c\_temp\_3);  // result[13]  wire result13\_c;  FA result13\_FA\_1(result13\_c\_temp\_1, result13\_temp\_1, wallaceTree[6][7], wallaceTree[7][6], result12\_c);  FA result13\_FA\_2(result13\_c, result[13], result13\_temp\_1, result12\_c\_temp\_2, result12\_c\_temp\_1);  // result[14]  FA result14\_FA\_1(result14\_c, result[14], wallaceTree[7][7], result13\_c, result13\_c\_temp\_1);  // result[15]  assign result[15] = result14\_c;  endmodule |

**Testbench Module for the Wallace Multiplier:**

|  |
| --- |
| module tb\_wallace\_8x8;  // Inputs  reg [7:0] a;  reg [7:0] b;  // Output  wire [15:0] result;  // Instantiate the wallace\_8x8 module  wallace\_8x8 uut (  .result(result),  .a(a),  .b(b)  );  // Test sequence  initial begin  // Display header  $display("Time\t\t a\t\t b\t\t Result");  // Test case 1  a = 8'b00011001; // 25  b = 8'b00000111; // 7  #10;  $display("%d\t\t %b\t %b\t %b", $time, a, b, result);  // Test case 2  a = 8'b11111111; // 255  b = 8'b00000001; // 1  #10;  $display("%d\t\t %b\t %b\t %b", $time, a, b, result);  // Test case 3  a = 8'b10000000; // 128  b = 8'b00000010; // 2  #10;  $display("%d\t\t %b\t %b\t %b", $time, a, b, result);  // Test case 4  a = 8'b01111111; // 127  b = 8'b00000011; // 3  #10;  $display("%d\t\t %b\t %b\t %b", $time, a, b, result);  // Test case 5 (Edge case: both inputs are zero)  a = 8'b00000000; // 0  b = 8'b00000000; // 0  #10;  $display("%d\t\t %b\t %b\t %b", $time, a, b, result);  // Test case 6 (Maximum values for both inputs)  a = 8'b11111111; // 255  b = 8'b11111111; // 255  #10;  $display("%d\t\t %b\t %b\t %b", $time, a, b, result);  // Finish simulation  $finish;  end  endmodule |

Simulation Results:

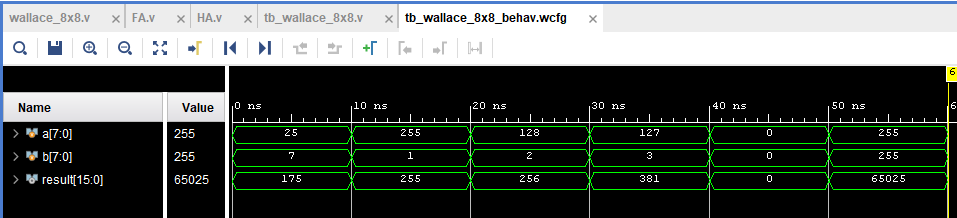


Figure 4: Wallace Tree Simulation Results