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

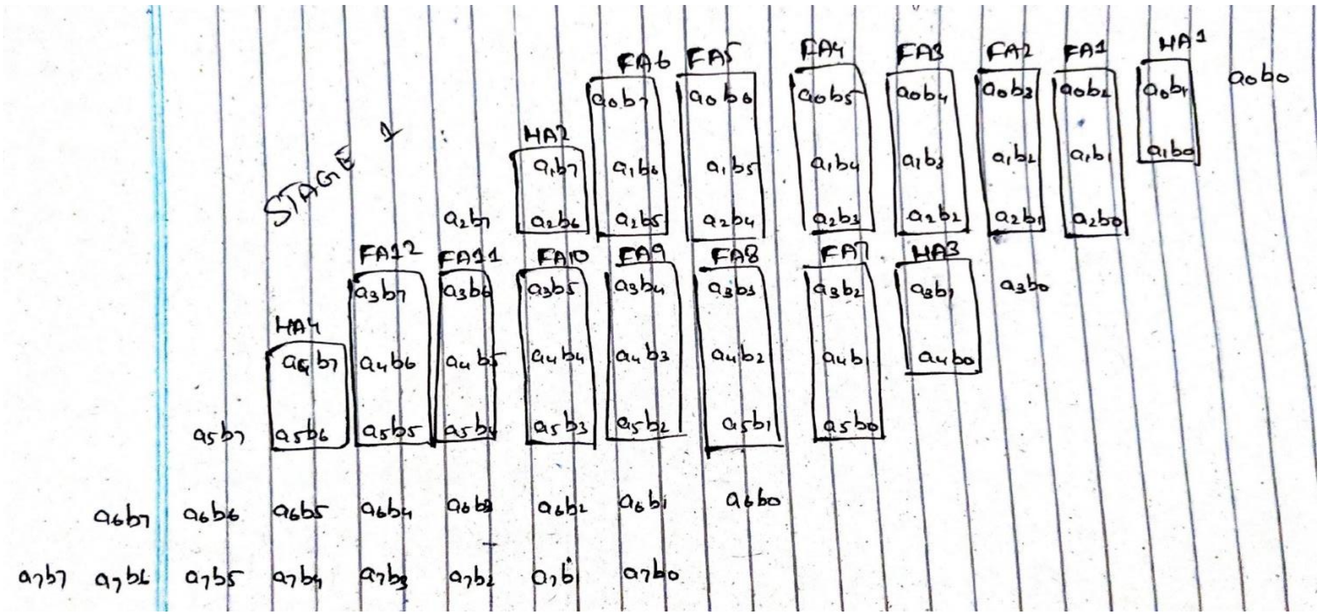


Figure 1: Stage One

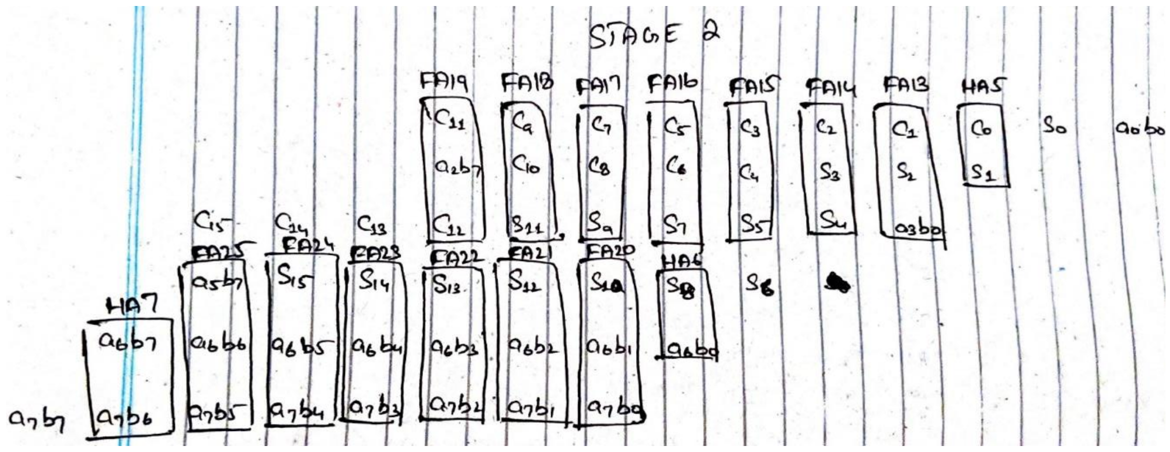


Figure 2: Stage Two

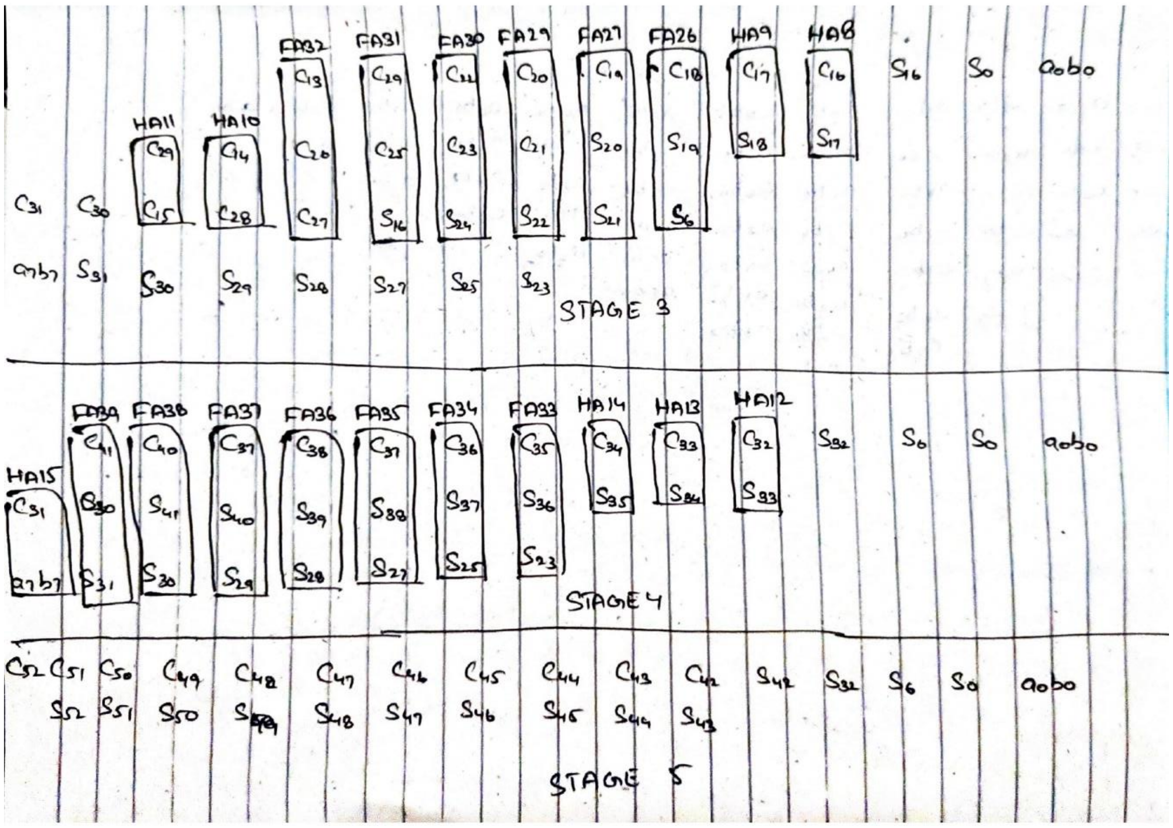


Figure 3: Stage Three Four and Five

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);
```

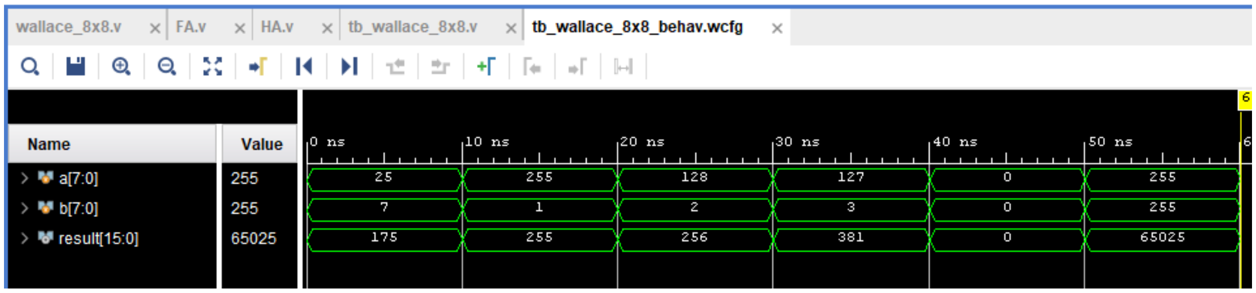
```
   FA result13_FA_2(result13_c, result[13], result13_temp_1, result12_c_temp_2,
result12_c_temp_1);

   // 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:



Name	Value	0 ns	10 ns	20 ns	30 ns	40 ns	50 ns
a[7:0]	255	25	255	128	127	0	255
b[7:0]	255	7	1	2	3	0	255
result[15:0]	65025	175	255	256	381	0	65025

Figure 4: Wallace Tree Simulation Results