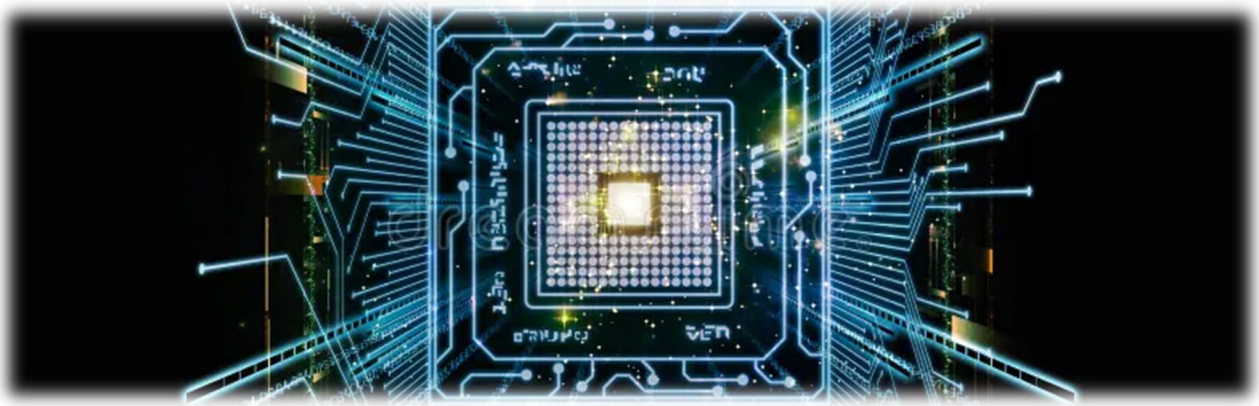


Name: Kirllos Atef Shawky Elia

Project: 64-bit Parallel Prefix Adder – Brent – Kung Method

GitHub: https://github.com/kirlloselia/ppa_brentkung

Table of Contents

1. Specs:

You are asked to design a 64-bit parallel prefix adder using Brent-kung Carry network through the following the steps below and attach all the required captures and codes
Fig.3 below show an example how to implement a size-8 prefix sum network using brent-kung
Fig.4 show the steps how the example in fig.3 is build step by step first layer one which contains (8-1 adders) and change the problem from 8-input problem to a smaller one with only 4-inputs then we apply the same idea by using (4-1 adders) and split the problem to half please spend some time to convince yourself and make sure you understand the pattern very well then start to map this into a carry network of 64-bit adder by removing each + sign with a carry operator you will find the implementation of carry operator below in fig.4

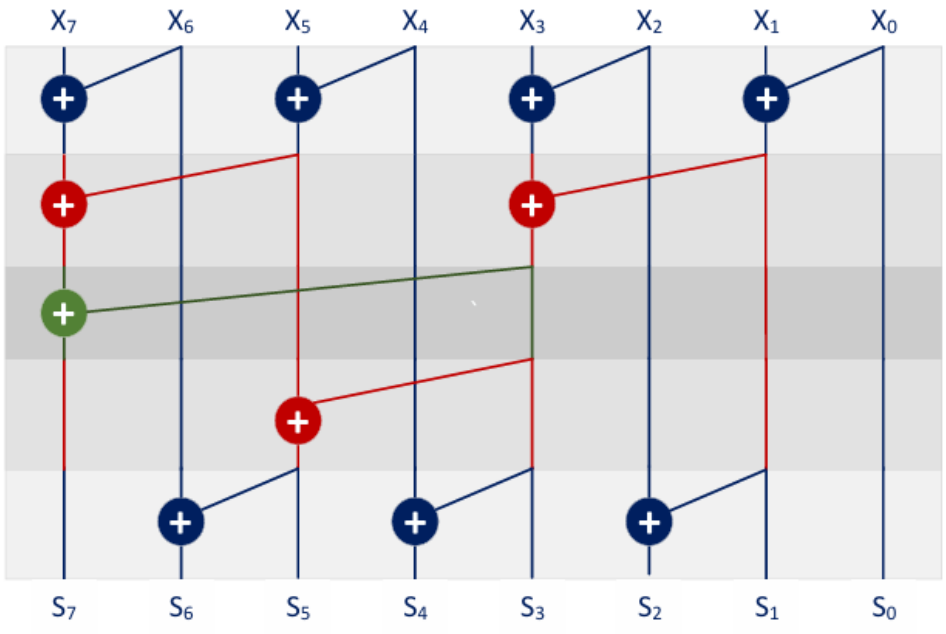


Figure 3

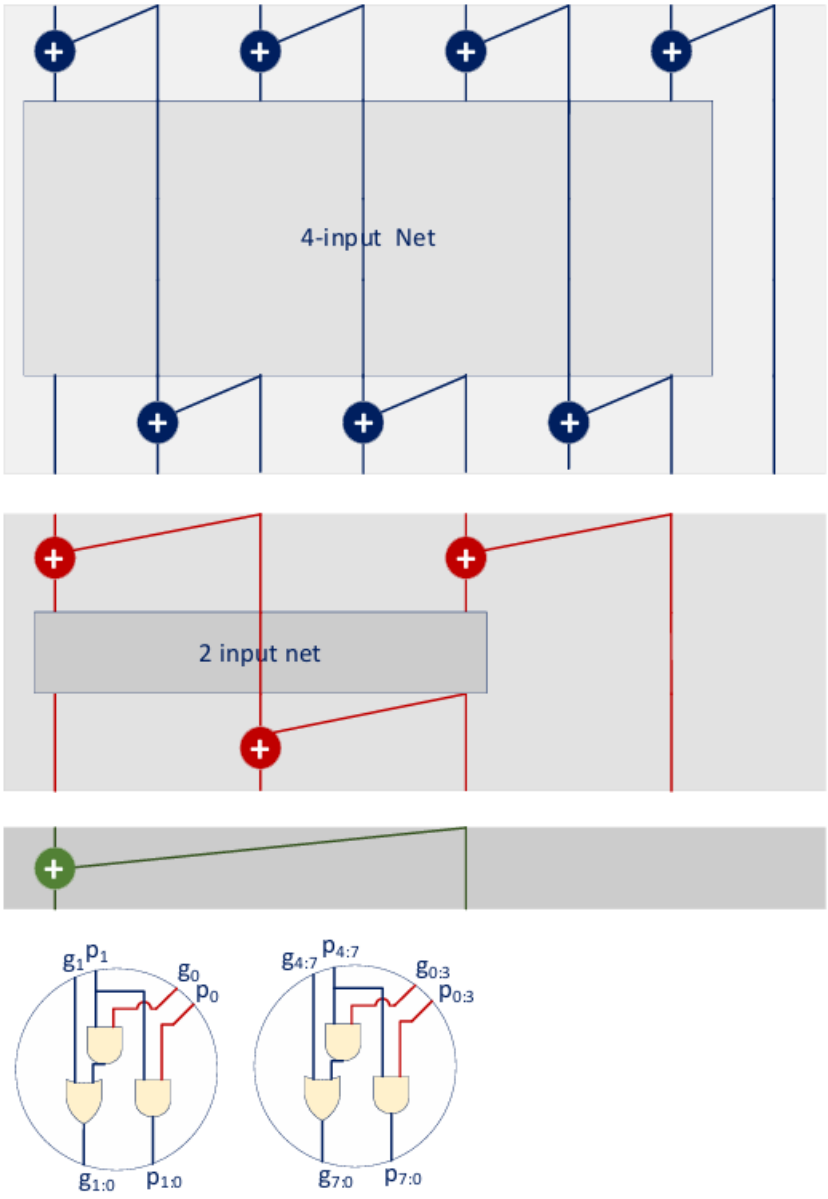
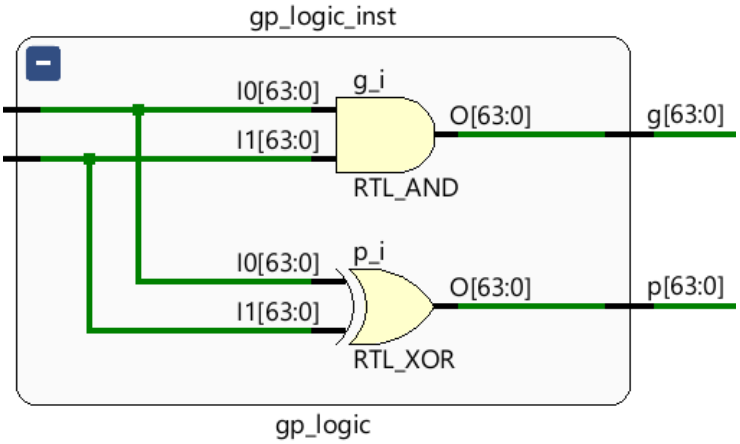


Figure 4

- **Inputs** {*A*, *B*, *Cin*} **(***A,B* **are 64-bits inputs and** *Cin* **is 1-bit)**
- **Outputs** {*Sum*, *Cout*} **(***sum* **is 64-bits output and** *Cout* **is 1-bit)**

2. G,P generation :

a) RTL schematic :

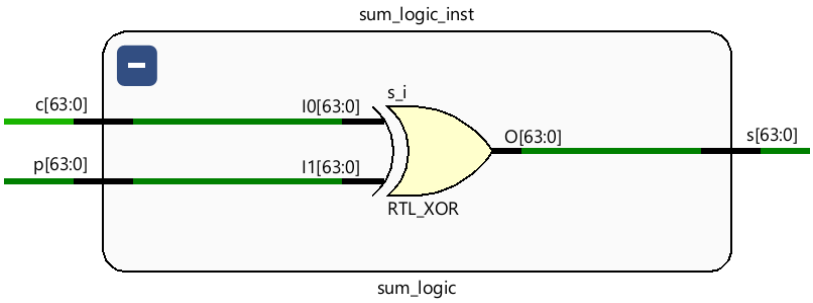


b) RTL code :

```
1 module gp_logic #(
2     parameter N = 64
3 )(
4     input [N-1:0] x, y,
5     output [N-1:0] g, p
6     );
7     assign g = x&y;
8     assign p = x^y;
9 endmodule
```

3. SUM

a) RTL schematic :

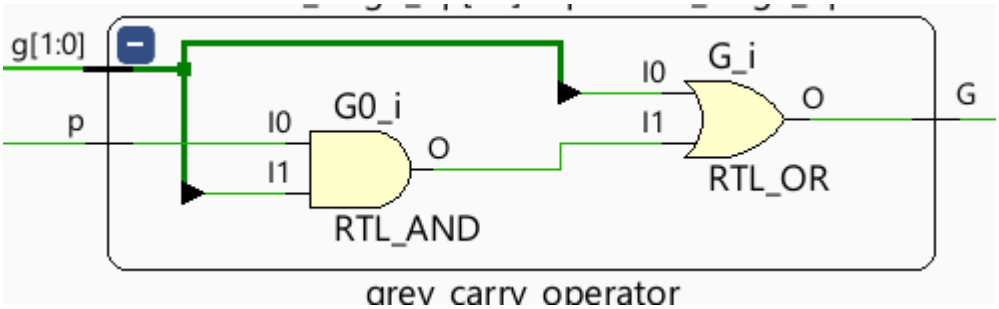


b) RTL code :

```
1 module sum_logic#(
2     parameter N = 64
3 )(
4     input [N-1:0] c, p,
5     output [N-1:0] s
6     );
7     assign s = c ^ p;
8 endmodule
```

4. Grey Operator

a) RTL schematic :

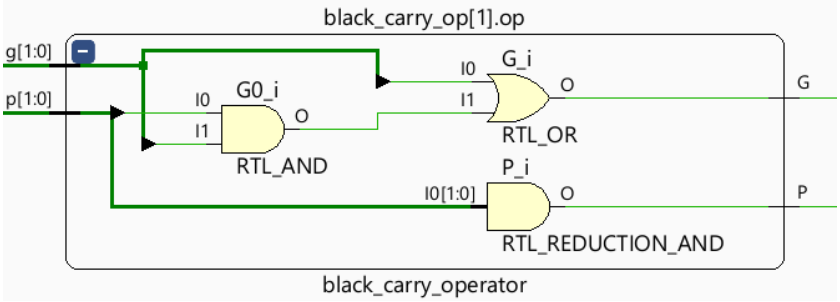


b) RTL code :

```
module grey_carry_operator (
    input p,                    // p''
    input [1:0] g,              // g'', g'
    output G
);

    assign G = g[1] | (p & g[0]);

endmodule
```

5. Black Operator

a) RTL schematic :

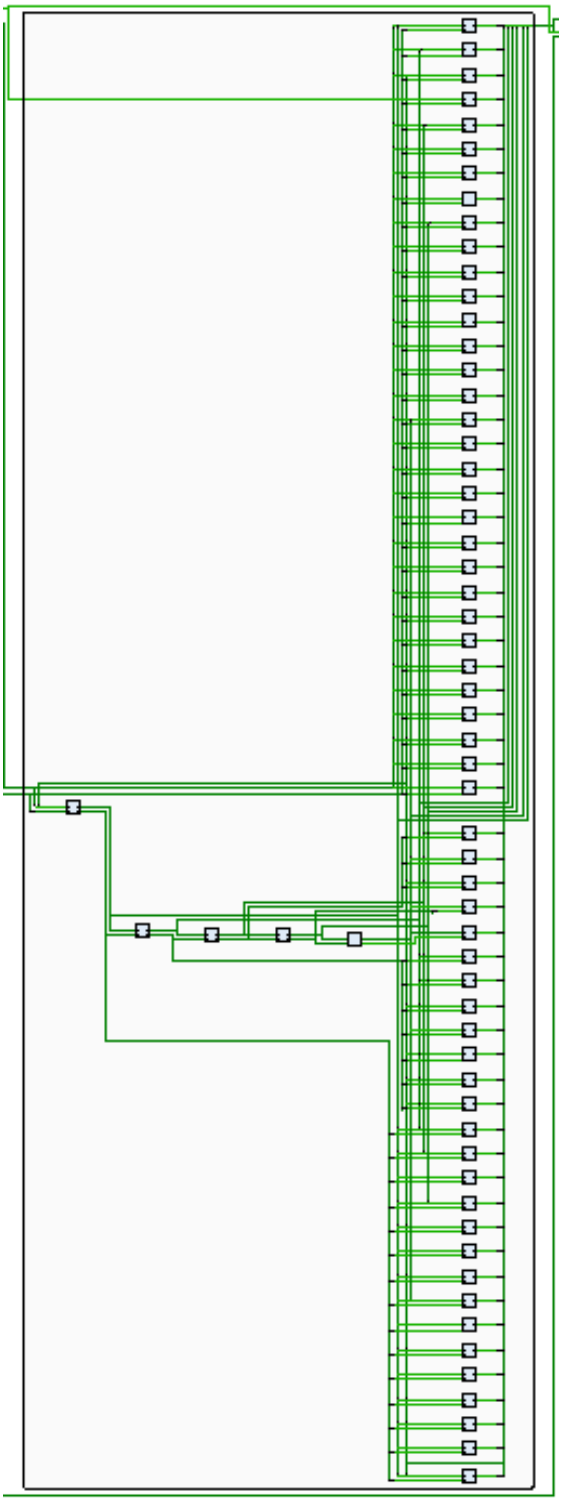


b)RTL code :

```
1  // black cell: pg outputs
2  module black_carry_operator (
3      input [1:0] p, g,
   // g'', p'', g', p'
4      output P, G
5  );
6
7      assign P = &p;
8      assign G = g[1] | (p[1] & g[0]);
9
10 endmodule
11
```

6. Carry Network

a) RTL schematic :

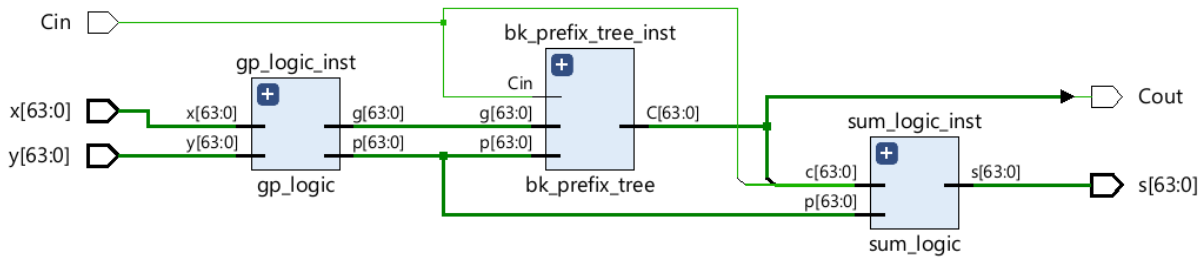


b) RTL code :

```
// carry network
module bk_prefix_tree #(
    parameter N = 16
)(
    input [N-1:0] p,g,
    input Cin,
    output [N-1:0] C       // C_-1 = Cin, C_N-1 = Cout
    );
    localparam red_stages_no = $clog2(N);
    localparam exp_stages_no = $clog2(N)-1;
    localparam total_stages_no = red_stages_no + exp_stages_no;
// total number of stages: 2log2(N)-1
    localparam NDIV2 = N >> 1;

    // assign C[-1] = Cin;

    wire [N-2:0] g_inter_red, p_inter_red;      // note that all intermediate g signals numbers

// in reduction levels converges to N-1 (N/2+N/4+..+1 = N-1)
    // handle Cin
    wire g0;
    grey_carry_operator op_minus1 (.g({g[0], Cin}), .p(p[0]), .G(g0));
    assign C[0] = g0;       //

    // first level, takes input from module's input
    bk_reduction_stage #(.N(N)) reduction_stage1 (.p(p[N-1:1]), .g({g[N-1:1], g0}),
                                                  .P(p_inter_red[NDIV2-1:1]), .G(g_inter_red[NDIV2-1:
0]));
    assign C[1] = g_inter_red[0];     // after level 1, x1 is ready

    genvar i;
    genvar j;
    generate
        for (i = 2; i <= red_stages_no; i = i + 1) begin: reduction_stage
            localparam in_idx = (1<<red_stages_no)-(1<<(red_stages_no-i+2));
// input index to current level = 2^levels-2^(levels-i+2)
            localparam in_size = N>>(i-1);
// input size = (N>>i)
            localparam out_idx = in_idx+in_size;
// input index of current level = input index to current level + input size
            localparam out_size = in_size>>1;
// output size = input size/2
            wire [in_size-1:0] g_inter_red_in = g_inter_red[in_size+in_idx-1:in_idx];
            wire [in_size-1:1] p_inter_red_in = p_inter_red[in_size+in_idx-1:in_idx+1];
            if (i == red_stages_no) begin
                bk_reduction_stage #(.N(in_size)) reduction_stage
 (.p(p_inter_red_in), .g(g_inter_red_in),
                                                        .G(g_inter_red[out_size+out_idx-1
:out_idx]));
            end
            else begin
                bk_reduction_stage #(.N(in_size)) reduction_stage
 (.p(p_inter_red_in), .g(g_inter_red_in),
                                                  .P(p_inter_red[out_size+out_idx-1:out_idx+1
]), .G(g_inter_red[out_size+out_idx-1:out_idx]));
            end
            localparam c_index = (1<<i) - 1;   // (2^level) - 1, level = i
            assign C[c_index] = g_inter_red[out_idx];
        end

    endgenerate
    generate
        for (i = 1; i <= exp_stages_no; i = i + 1) begin: expansion_stage
            // localparam stage_size = (1<<i) - 1;          // stage size = (2^level) - 1
            // localparam in_size = stage_size << 1;        // input size = stage size * 2
            localparam step = N>>(i+1);
            localparam c_index_start = 3*step-1;
            localparam c_index_end = N-step-1;
            if (i == exp_stages_no) begin
                for (j = c_index_start; j <= c_index_end; j = j + (step<<1)) begin
: expansion_stage_op
                    localparam c_base_idx = j - step;
                    grey_carry_operator expansion_stage_op
 (.p(p[j]), .g({g[j], C[c_base_idx]}), .G(C[j]));
                end
            end
            else begin
                localparam i_red_stage = $clog2(step);
                localparam start_idx_red = (1<<red_stages_no)-(1<<(red_stages_no-i_red_stage+1));
                for (j = c_index_start; j <= c_index_end; j = j + (step<<1)) begin
: expansion_stage_op
                    localparam c_base_idx = j - step;
                    localparam red_group = j>>(i_red_stage);
                    localparam red_idx = start_idx_red + red_group;
                    grey_carry_operator expansion_stage_op
 (.p(p_inter_red[red_idx]), .g({g_inter_red[red_idx], C[c_base_idx]}), .G(C[j]));
                end
            end
        end
    endgenerate

endmodule
```

7. 64 adder using Parallel Prefix Adder Brent Kung

a) RTL schematic :

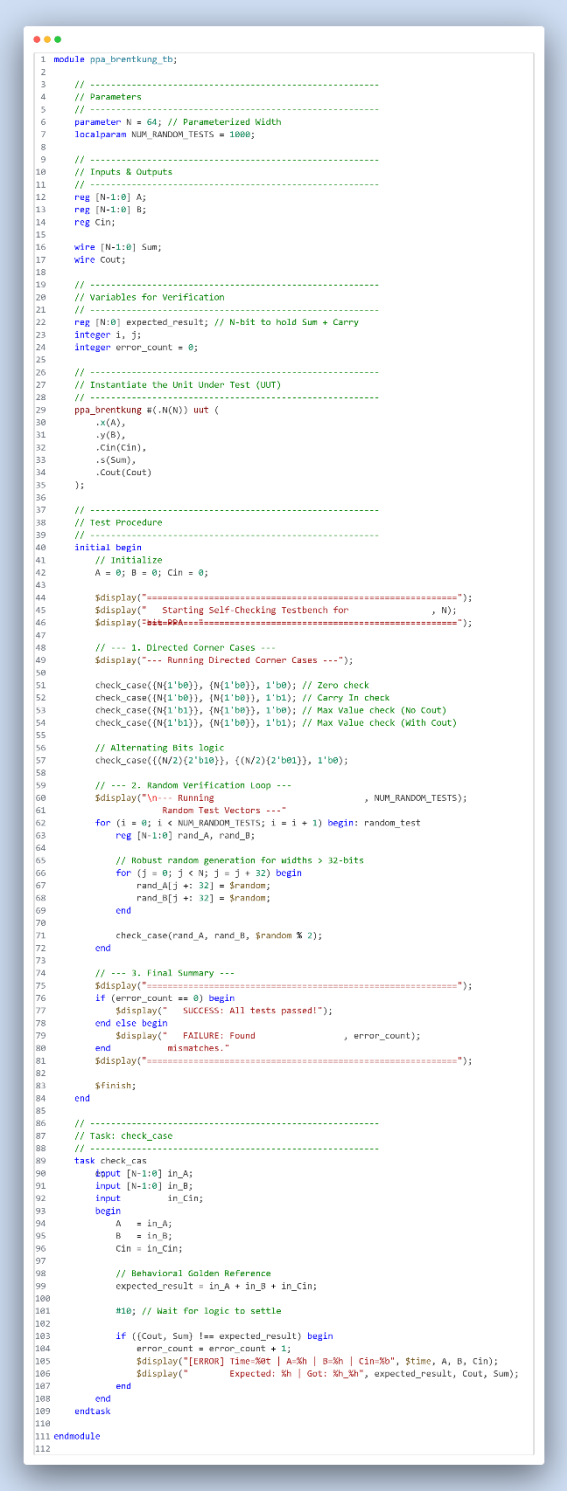


b) RTL code :

```
module ppa_brentkung #(
    parameter N = 64
)(
    input [N-1:0] x, y,
    input Cin,
    output [N-1:0] s,
    output Cout
    );
    wire [N-1:0] g, p, c;
    gp_logic #(.N(N)) gp_logic_inst (.x(x), .y(y), .g(g), .p(p));
    sum_logic #(.N(N)) sum_logic_inst (.c(c), .p(p), .s(s));
    wire [N-1:0] c_network;
    bk_prefix_tree #(.N(N)) bk_prefix_tree_inst (.g(g), .p(p), .Cin(Cin), .C(c_network));
    assign Cout = c_network  -1];
    assign c = {c[Network[N-2:0], Cin};  // Concatenate, not slice assignment
endmodule
```

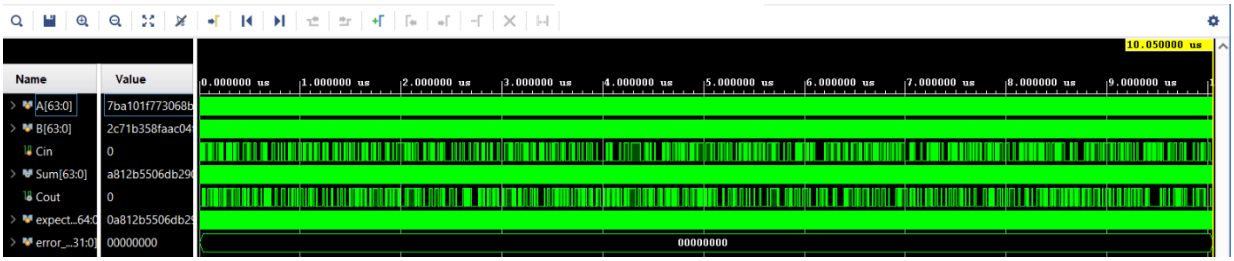
8. testbench code for 64 adder

a) code



```
module ppa_brentkung_tb;

    // ----------------------------------------------------------
    // Parameters
    // ----------------------------------------------------------
    parameter N = 64; // Parameterized Width
    localparam NUM_RANDOM_TESTS = 1000;

    // ----------------------------------------------------------
    // Inputs & Outputs
    // ----------------------------------------------------------
    reg [N-1:0] A;
    reg [N-1:0] B;
    reg Cin;

    wire [N-1:0] Sum;
    wire Cout;

    // ----------------------------------------------------------
    // Variables for Verification
    // ----------------------------------------------------------
    reg [N:0] expected_result; // N-bit to hold Sum + Carry
    integer i, j;
    integer error_count = 0;

    // ----------------------------------------------------------
    // Instantiate the Unit Under Test (UUT)
    // ----------------------------------------------------------
    ppa_brentkung #(.N(N)) uut (
        .x(A),
        .y(B),
        .Cin(Cin),
        .s(Sum),
        .Cout(Cout)
    );

    // ----------------------------------------------------------
    // Test Procedure
    // ----------------------------------------------------------
    initial begin
        // Initialize
        A = 0; B = 0; Cin = 0;

        $display("============================================================");
        $display("   Starting Self-Checking Testbench for                , N);
        $display("=bit=PPA=========================================================");

        // --- 1. Directed Corner Cases ---
        $display("--- Running Directed Corner Cases ---");

        check_case({N{1'b0}}, {N{1'b0}}, 1'b0); // Zero check
        check_case({N{1'b0}}, {N{1'b0}}, 1'b1); // Carry In check
        check_case({N{1'b1}}, {N{1'b0}}, 1'b0); // Max Value check (No Cout)
        check_case({N{1'b1}}, {N{1'b0}}, 1'b1); // Max Value check (With Cout)

        // Alternating Bits logic
        check_case({(N/2){2'b10}}, {(N/2){2'b01}}, 1'b0);

        // --- 2. Random Verification Loop ---
        $display("\n--- Running                             , NUM_RANDOM_TESTS);
                 Random Test Vectors ---"
        for (i = 0; i < NUM_RANDOM_TESTS; i = i + 1) begin: random_test
            reg [N-1:0] rand_A, rand_B;

            // Robust random generation for widths > 32-bits
            for (j = 0; j < N; j = j + 32) begin
                rand_A[j +: 32] = $random;
                rand_B[j +: 32] = $random;
            end

            check_case(rand_A, rand_B, $random % 2);
        end

        // --- 3. Final Summary ---
        $display("============================================================");
        if (error_count == 0) begin
            $display("   SUCCESS: All tests passed!");
        end else begin
            $display("   FAILURE: Found                 , error_count);
        end          mismatches."
        $display("============================================================");

        $finish;
    end

    // ----------------------------------------------------------
    // Task: check_case
    // ----------------------------------------------------------
    task check_cas
        input [N-1:0] in_A;
        input [N-1:0] in_B;
        input         in_Cin;
        begin
            A   = in_A;
            B   = in_B;
            Cin = in_Cin;

            // Behavioral Golden Reference
            expected_result = in_A + in_B + in_Cin;

            #10; // Wait for logic to settle

            if ((Cout, Sum) !== expected_result) begin
                error_count = error_count + 1;
                $display("[ERROR] Time=%0t | A=%h | B=%h | Cin=%b", $time, A, B, Cin);
                $display("        Expected: %h | Got: %h_%h", expected_result, Cout, Sum);
            end
        end
    endtask

endmodule

```

b) Waveform

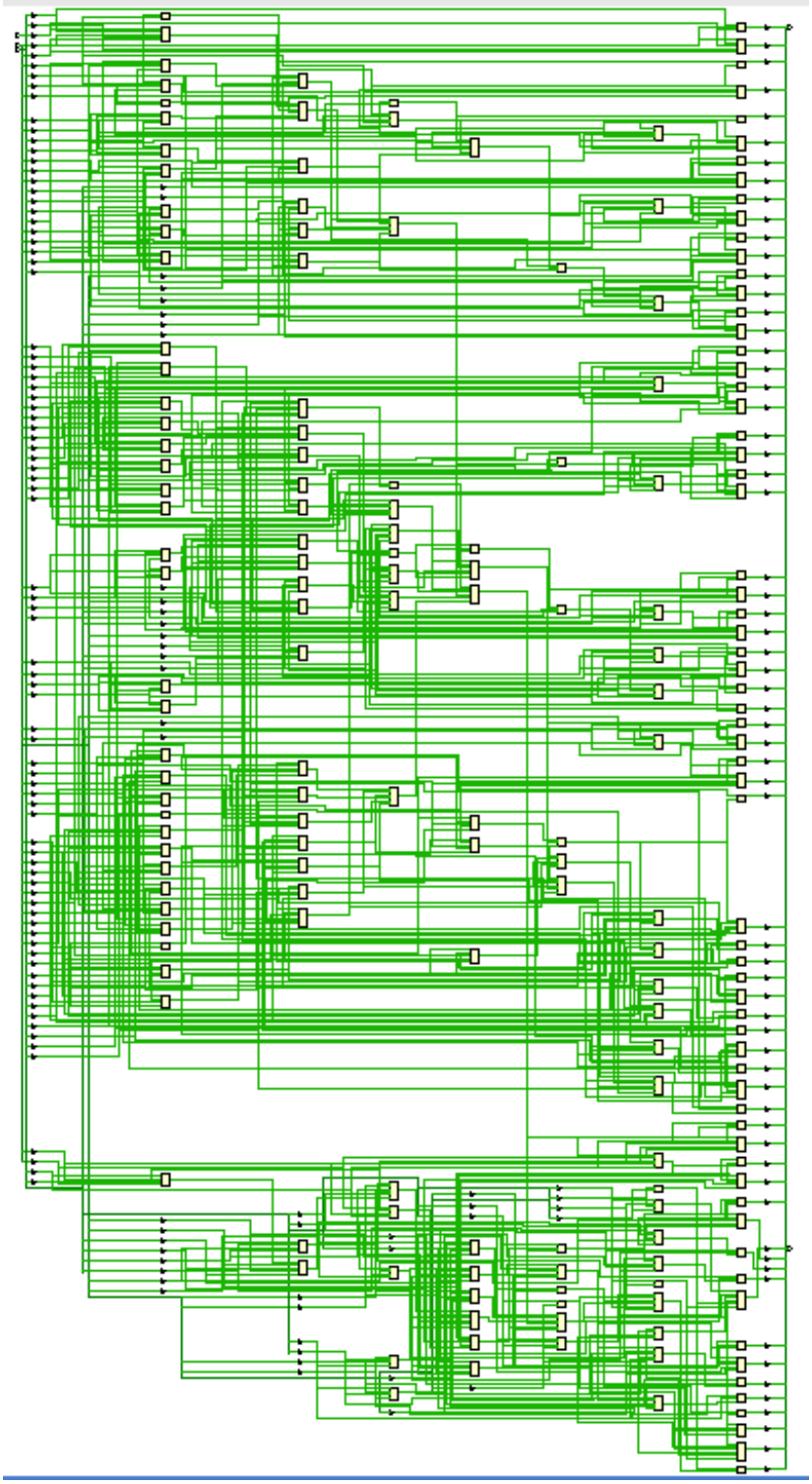


c) Test-Bench log

```
run all
================================================================
   Starting Self-Checking Testbench for 64-bit PPA
================================================================
--- Running Directed Corner Cases ---

--- Running 1000 Random Test Vectors ---
================================================================
   SUCCESS: All tests passed!
================================================================
```

12. IMPLEMENTATION

a) Implementation schematic



13. Utilization

Name	Slice LUTs (63400)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)
N ppa_brentkung	134	47	134	194

14. Timing Reports

a) setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	8	7	6	x[38]	Cout	22.960	4.619	18.341	∞	input port clock
Path 2	∞	9	8	6	x[38]	s[58]	22.137	5.182	16.955	∞	input port clock
Path 3	∞	9	8	6	x[38]	s[59]	21.805	4.954	16.851	∞	input port clock
Path 4	∞	9	8	6	x[38]	s[54]	21.673	4.932	16.741	∞	input port clock
Path 5	∞	9	8	6	x[38]	s[63]	21.261	4.662	16.599	∞	input port clock
Path 6	∞	8	7	6	x[38]	s[61]	21.215	4.839	16.376	∞	input port clock
Path 7	∞	8	7	6	x[38]	s[62]	21.174	4.592	16.583	∞	input port clock
Path 8	∞	8	7	6	x[38]	s[60]	21.117	4.600	16.517	∞	input port clock
Path 9	∞	8	7	6	x[38]	s[57]	21.115	4.828	16.287	∞	input port clock
Path 10	∞	9	8	6	x[38]	s[55]	21.087	4.729	16.358	∞	input port clock

b) hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	2	4	y[53]	s[53]	2.322	1.369	0.953	-∞	input port clock
Path 12	∞	3	2	5	y[55]	s[55]	2.344	1.420	0.924	-∞	input port clock
Path 13	∞	3	2	5	y[52]	s[52]	2.369	1.384	0.985	-∞	input port clock
Path 14	∞	3	2	4	y[43]	s[43]	2.374	1.376	0.998	-∞	input port clock
Path 15	∞	3	2	4	y[38]	s[38]	2.377	1.372	1.005	-∞	input port clock
Path 16	∞	3	2	4	y[50]	s[50]	2.403	1.401	1.002	-∞	input port clock
Path 17	∞	3	2	4	y[58]	s[58]	2.404	1.417	0.987	-∞	input port clock
Path 18	∞	3	2	4	y[44]	s[44]	2.407	1.388	1.019	-∞	input port clock
Path 19	∞	3	2	6	y[54]	s[54]	2.415	1.375	1.040	-∞	input port clock
Path 20	∞	3	2	5	y[56]	s[57]	2.415	1.402	1.013	-∞	input port clock