# Parallel Prefix Adder-Subtractor (16-bit)

## Abstract:

Addition and subtraction are the most fundamental mathematical operations. Thus, Adders are essential building blocks for most combinational circuits. Every logic gate operates on binary digits, so addition, albeit simple on paper, turns out to be a pretty complex problem in Digital Design.

For addition of n-bit binary numbers, more than one full/half adder is required. The computation has to be performed bit-by-bit.

### Ripple Carry Adder:

The Ripple Carry Adder consists of a series of full adders, arranged linearly, each taking in the carry-out from the previous bit-addition, from LSB to MSB. This design is simple, but is inefficient in time-complexity. Each adder must wait for the previous bits' addition to compute and produce its carry-out for the current bits' addition to begin.

### Carry Look-ahead Adder:

The Carry Look-ahead Adder solves the Ripple Carry Adder's inefficiency by computing the carry-out bits beforehand and executing additions simultaneously. There are two signals: generate, gi = ai & bi and propagate, pi = ai | bi. The carry-out bits are computed as:

c1 = g0

c2 = g1 | p1&g0

c3 = g2 | p2&g1 | p2&p1&g0  and so on.

However, the circuitry required for this adder is extensive as compared to the Ripple Carry Adder, but minimal in time-complexity.

| Adder | Area-complexity | Time-Complexity |
|---|---|---|
| Ripple Carry | O(n) | O(n) |
| Carry Look-ahead | $O(n^2)$ | O(logn) |

Subtractors work in a similar fashion, the subtrahend is 2's complemented.

So our problem is: How to minimize the components required for the adder while maintaining the minimal time-complexity.

# Literature Survey:

## Parallel Prefix Circuits:

Parallel Prefix Circuits implement downsizing. For example:

y0 = x0

y1 = x1 ^ x0

y2 = x2 ^ x1 ^ x0  and so on, can be optimized as,

y0 = x0

y1 = x1 ^ y0

y2 = x2 ^ y1  and so on.

This significantly reduces components and retains the same speed.

## Parallel Prefix Adder:

We implement our adder by using the sum generated from the previous bits to find the carry-in for the current bits. The sum in a full adder is si = ai ^ bi ^ ci (where ci is the carry-in from the previous circuit), so the prefix circuit is perfectly feasible.

Different architectures have been devised for the calculation of the carry bits.

1960: J. Skansky – conditional Adder

1973: Kogge -Stone Adder

1980: Ladner – Fisher Adder

1982: Brent – Kung Adder

1987: Han Carlson Adder

1999: S. Knowles Adder

But the most modular design is that of the J. Sklansky – conditional Adder (1960), and the Ladner – Fischer Adder (1980) (implementation with low depth and high fan-out nodes). This is the architecture we shall implement.

The propagate and generate signals are first computed using the input bits as:

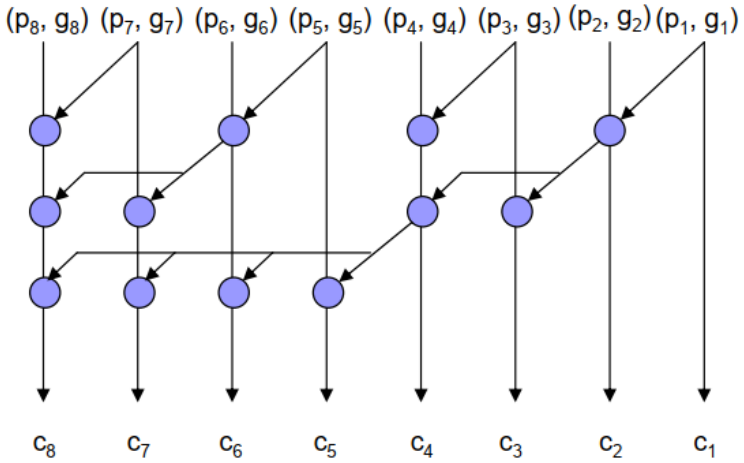
gi = ai & bi

pi = ai | bi.

Henceforth, for every ‘circle’ in the below diagram, the propagate and generate signals are:

pi = pi&pj

gi= pi&gj | gi.




And the final sum is calculated as: si = g(i-1) ^ ai ^ bi.

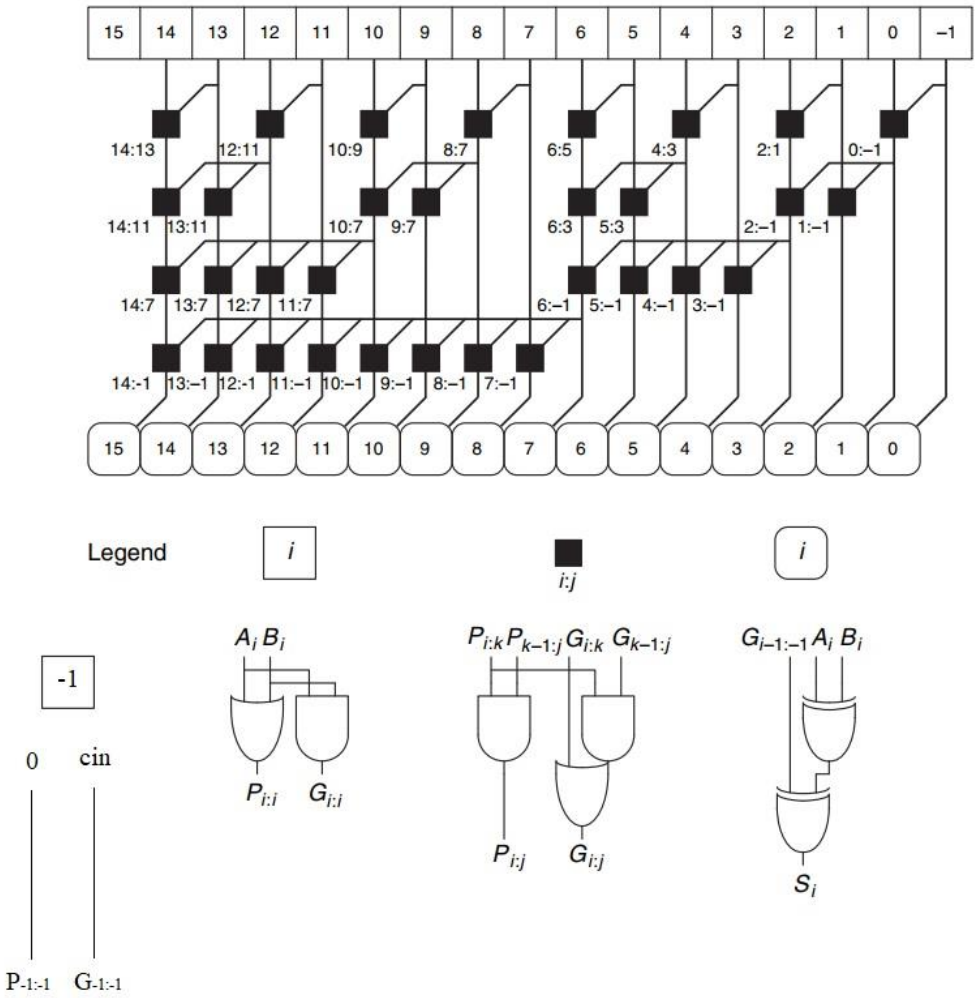
The final circuit diagram is given in the next section.

## Adder-Subtractor:

In a subtractor, the subtrahend of the input is to be 2’s complemented. This is achieved by taking a carry-in bit that indicates whether subtraction is True(1) or not. This carry-in bit is sent into XOR gates along with the subtrahend’s bits (this inverts the bits). Then the carry-in bit is added to the sum of the minuend and the inverted subtrahend (this 2’s complements it). The first generate signal is set to the carry-in, and the first propagate signal to 0. This accomplishes subtraction.

This 16- bit Parallel Prefix Adder-Subtractor is to be implemented in Verilog, which is a Hardware Description Language.

## Circuit:




## Verilog Code:

Library File:

```
module and2 (input wire i0, i1, output wire o);
	assign o = i0 & i1;
endmodule

module or2 (input wire i0, i1, output wire o);
	assign o = i0 | i1;
endmodule
```

```
module xor2 (input wire i0, i1, output wire o);
	assign o = i0 ^ i1;
endmodule

module xor3 (input wire i0, i1, i2, output wire o);
	wire temp;
	xor2 xor2_0 (i0, i1, temp);
	xor2 xor2_1 (i2, temp, o);
endmodule
```

## Adder File:

```
module triangle (input wire a, b, output wire p, g);
	or2 or_1 (a, b, p);
	and2 and_1 (a, b, g);
endmodule

module box (input wire pi, gi, pj, gj, output wire P, G);
	wire temp;
	and2 and_1 (pi, pj, P);
	and2 and_2 (pi, gj, temp);
	or2 or_0 (gi, temp, G);
endmodule

module circle (input wire a, b, gi, output wire s);
	xor3 xor3_1 (a, b, gi, s);
endmodule



module addsub16 (input wire [15:0] a, b, input wire cin, output wire [15:0] S);
	wire [15:0] p, g;
	wire [15:0] b_res;
	xor2 xor_0 (b[0], cin, b_res[0]);
	xor2 xor_1 (b[1], cin, b_res[1]);
```

```
xor2 xor_2 (b[2], cin, b_res[2]);
xor2 xor_3 (b[3], cin, b_res[3]);
xor2 xor_4 (b[4], cin, b_res[4]);
xor2 xor_5 (b[5], cin, b_res[5]);
xor2 xor_6 (b[6], cin, b_res[6]);
xor2 xor_7 (b[7], cin, b_res[7]);
xor2 xor_8 (b[8], cin, b_res[8]);
xor2 xor_9 (b[9], cin, b_res[9]);
xor2 xor_10 (b[10], cin, b_res[10]);
xor2 xor_11 (b[11], cin, b_res[11]);
xor2 xor_12 (b[12], cin, b_res[12]);
xor2 xor_13 (b[13], cin, b_res[13]);
xor2 xor_14 (b[14], cin, b_res[14]);
xor2 xor_15 (b[15], cin, b_res[15]);
triangle triangle_0 (a[0], b_res[0], p[0], g[0]);
triangle triangle_1 (a[1], b_res[1], p[1], g[1]);
triangle triangle_2 (a[2], b_res[2], p[2], g[2]);
triangle triangle_3 (a[3], b_res[3], p[3], g[3]);
triangle triangle_4 (a[4], b_res[4], p[4], g[4]);
triangle triangle_5 (a[5], b_res[5], p[5], g[5]);
triangle triangle_6 (a[6], b_res[6], p[6], g[6]);
triangle triangle_7 (a[7], b_res[7], p[7], g[7]);
triangle triangle_8 (a[8], b_res[8], p[8], g[8]);
triangle triangle_9 (a[9], b_res[9], p[9], g[9]);
triangle triangle_10 (a[10], b_res[10], p[10], g[10]);
triangle triangle_11 (a[11], b_res[11], p[11], g[11]);
triangle triangle_12 (a[12], b_res[12], p[12], g[12]);
triangle triangle_13 (a[13], b_res[13], p[13], g[13]);
triangle triangle_14 (a[14], b_res[14], p[14], g[14]);
triangle triangle_15 (a[15], b_res[15], p[15], g[15]);
wire [7:0] lvl1_P, lvl1_G;
box box_lvl1_0 (p[0], g[0], 1'b0, cin, lvl1_P[0], lvl1_G[0]);
box box_lvl1_1 (p[2], g[2], p[1], g[1], lvl1_P[1], lvl1_G[1]);
box box_lvl1_2 (p[4], g[4], p[3], g[3], lvl1_P[2], lvl1_G[2]);
```

```
box box_lvl1_3 (p[6], g[6], p[5], g[5], lvl1_P[3], lvl1_G[3]);
box box_lvl1_4 (p[8], g[8], p[7], g[7], lvl1_P[4], lvl1_G[4]);
box box_lvl1_5 (p[10], g[10], p[9], g[9], lvl1_P[5], lvl1_G[5]);
box box_lvl1_6 (p[12], g[12], p[11], g[11], lvl1_P[6], lvl1_G[6]);
box box_lvl1_7 (p[14], g[14], p[13], g[13], lvl1_P[7], lvl1_G[7]);
wire [7:0] lvl2_P, lvl2_G;
box box_lvl2_0 (p[1], g[1], lvl1_P[0], lvl1_G[0], lvl2_P[0], lvl2_G[0]);
box box_lvl2_1 (lvl1_P[1], lvl1_G[1], lvl1_P[0], lvl1_G[0], lvl2_P[1], lvl2_G[1]);
box box_lvl2_2 (p[5], g[5], lvl1_P[2], lvl1_G[2], lvl2_P[2], lvl2_G[2]);
box box_lvl2_3 (lvl1_P[3], lvl1_G[3], lvl1_P[2], lvl1_G[2], lvl2_P[3], lvl2_G[3]);
box box_lvl2_4 (p[9], g[9], lvl1_P[4], lvl1_G[4], lvl2_P[4], lvl2_G[4]);
box box_lvl2_5 (lvl1_P[5], lvl1_G[5], lvl1_P[4], lvl1_G[4], lvl2_P[5], lvl2_G[5]);
box box_lvl2_6 (p[13], g[13], lvl1_P[6], lvl1_G[6], lvl2_P[6], lvl2_G[6]);
box box_lvl2_7 (lvl1_P[7], lvl1_G[7], lvl1_P[6], lvl1_G[6], lvl2_P[7], lvl2_G[7]);
wire [7:0] lvl3_P, lvl3_G;
box box_lvl3_0 (p[3], g[3], lvl2_P[1], lvl2_G[1], lvl3_P[0], lvl3_G[0]);
box box_lvl3_1 (lvl1_P[2], lvl1_G[2], lvl2_P[1], lvl2_G[1], lvl3_P[1], lvl3_G[1]);
box box_lvl3_2 (lvl2_P[2], lvl2_G[2], lvl2_P[1], lvl2_G[1], lvl3_P[2], lvl3_G[2]);
box box_lvl3_3 (lvl2_P[3], lvl2_G[3], lvl2_P[1], lvl2_G[1], lvl3_P[3], lvl3_G[3]);
box box_lvl3_4 (p[11], g[11], lvl2_P[5], lvl2_G[5], lvl3_P[4], lvl3_G[4]);
box box_lvl3_5 (lvl1_P[6], lvl1_G[6], lvl2_P[5], lvl2_G[5], lvl3_P[5], lvl3_G[5]);
box box_lvl3_6 (lvl2_P[6], lvl2_G[6], lvl2_P[5], lvl2_G[5], lvl3_P[6], lvl3_G[6]);
box box_lvl3_7 (lvl2_P[7], lvl2_G[7], lvl2_P[5], lvl2_G[5], lvl3_P[7], lvl3_G[7]);
wire [7:0] lvl4_P, lvl4_G;
box box_lvl4_0 (p[7], g[7], lvl3_P[3], lvl3_G[3], lvl4_P[0], lvl4_G[0]);
box box_lvl4_1 (lvl1_P[4], lvl1_G[4], lvl3_P[3], lvl3_G[3], lvl4_P[1], lvl4_G[1]);
box box_lvl4_2 (lvl2_P[4], lvl2_G[4], lvl3_P[3], lvl3_G[3], lvl4_P[2], lvl4_G[2]);
box box_lvl4_3 (lvl2_P[5], lvl2_G[5], lvl3_P[3], lvl3_G[3], lvl4_P[3], lvl4_G[3]);
box box_lvl4_4 (lvl3_P[4], lvl3_G[4], lvl3_P[3], lvl3_G[3], lvl4_P[4], lvl4_G[4]);
box box_lvl4_5 (lvl3_P[5], lvl3_G[5], lvl3_P[3], lvl3_G[3], lvl4_P[5], lvl4_G[5]);
box box_lvl4_6 (lvl3_P[6], lvl3_G[6], lvl3_P[3], lvl3_G[3], lvl4_P[6], lvl4_G[6]);
box box_lvl4_7 (lvl3_P[7], lvl3_G[7], lvl3_P[3], lvl3_G[3], lvl4_P[7], lvl4_G[7]);
circle circle_0 (a[0], b_res[0], cin, S[0]);
circle circle_1 (a[1], b_res[1], lvl1_G[0], S[1]);
circle circle_2 (a[2], b_res[2], lvl2_G[0], S[2]);
circle circle_3 (a[3], b_res[3], lvl2_G[1], S[3]);
```

```
        circle circle_4 (a[4], b_res[4], lvl3_G[0], S[4]);
        circle circle_5 (a[5], b_res[5], lvl3_G[1], S[5]);
        circle circle_6 (a[6], b_res[6], lvl3_G[2], S[6]);
        circle circle_7 (a[7], b_res[7], lvl3_G[3], S[7]);
        circle circle_8 (a[8], b_res[8], lvl4_G[0], S[8]);
        circle circle_9 (a[9], b_res[9], lvl4_G[1], S[9]);
        circle circle_10 (a[10], b_res[10], lvl4_G[2], S[10]);
        circle circle_11 (a[11], b_res[11], lvl4_G[3], S[11]);
        circle circle_12 (a[12], b_res[12], lvl4_G[4], S[12]);
        circle circle_13 (a[13], b_res[13], lvl4_G[5], S[13]);
        circle circle_14 (a[14], b_res[14], lvl4_G[6], S[14]);
        circle circle_15 (a[15], b_res[15], lvl4_G[7], S[15]);
endmodule
```

## Testbench File:

```
module prefix_tb;
        reg [15:0] t_a,t_b;
        reg t_cin;
        wire [15:0] t_S;
        initial begin $dumpfile("testbench.vcd"); $dumpvars(0, prefix_tb); end
        addsub16 as16 (.a(t_a), .b(t_b), .cin(t_cin), .S(t_S));
        initial
        begin
                t_a [15:0] = 16'h0000; t_b [15:0] = 16'h0001; t_cin = 1'b0; #5
                t_a [15:0] = 16'h0069; t_b [15:0] = 16'h0069; t_cin = 1'b0; #5
                t_a [15:0] = 16'h0100; t_b [15:0] = 16'h0000; t_cin = 1'b0; #5
                t_a [15:0] = 16'h0110; t_b [15:0] = 16'h1001; t_cin = 1'b0; #5
                t_a [15:0] = 16'hffff; t_b [15:0] = 16'h0001; t_cin = 1'b0; #5
                t_a [15:0] = 16'h55aa; t_b [15:0] = 16'haa55; t_cin = 1'b0; #5
                t_a [15:0] = 16'h1010; t_b [15:0] = 16'h0101; t_cin = 1'b0; #5
                t_a [15:0] = 16'h0000; t_b [15:0] = 16'h0001; t_cin = 1'b0; #5
                t_a [15:0] = 16'h0000; t_b [15:0] = 16'h0001; t_cin = 1'b1; #5
                t_a [15:0] = 16'h0069; t_b [15:0] = 16'h0069; t_cin = 1'b1; #5
```

```
        t_a [15:0] = 16'h0100; t_b [15:0] = 16'h0000; t_cin = 1'b1; #5
        t_a [15:0] = 16'h0110; t_b [15:0] = 16'h1001; t_cin = 1'b1; #5
        t_a [15:0] = 16'hffff; t_b [15:0] = 16'h0001; t_cin = 1'b1; #5
        t_a [15:0] = 16'h55aa; t_b [15:0] = 16'haa55; t_cin = 1'b1; #5
        t_a [15:0] = 16'h1010; t_b [15:0] = 16'h0101; t_cin = 1'b1; #5
        t_a [15:0] = 16'h0000; t_b [15:0] = 16'h0001; t_cin = 1'b1;
    end
    initial
    begin
        $monitor ($time, "a = %h, b = %h, cin = %b, sum=%h", t_a, t_b, t_cin, t_S);
    end
endmodule
```