**DDCO Mini Project**

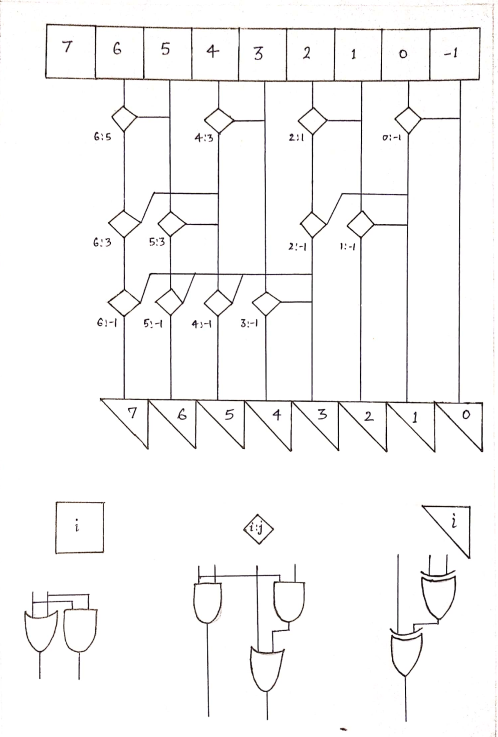
**Project Title: Design & implement 8–bit prefix–adder.**

**Section : H**

**Name : Sushma G Herakal**

**SRN : PES1UG19CS528**

**Circuit Diagram:**



**Algorithm (if any):**

Parallel prefix adders

The PPA is like a Carry Look Ahead Adder. The production of the carriers the prefix adders can be designed in many different ways based on the different requirements. We use tree structure form to increase the speed of arithmetic operation. Parallel prefix adders are faster adders and these are faster adders and used for high performance arithmetic structures in industries. The parallel prefix addition is done in 3 steps.

1. Pre-processing stage

In this stage we compute, the generate and propagate signals are used to generate carry input of each adder. A and B are inputs.

Pi = Ai +Bi

Gi = Ai **.**Bi

2. Carry generation network

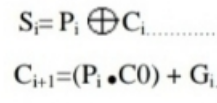
In this stage we compute carries corresponding to each bit. Execution is done in parallel form.After the computation of carries in parallel they are divided into smaller pieces. carry operator contain two AND gates, one OR gate. It uses propagate and generate as intermediate signals.

P(i:k) = P(i:j) **.**P(j-1:k)

G(i:k) = G(i:j) (G (j-1:k)**.**P(i:j))

3. Post processing stage

This is the final stage to compute the summation of input bits.



**Implementation (Code):**

module pg(input ai, bi, output pi, gi);

and2 and\_0(ai, bi, gi);

or2 or\_0(ai, bi, pi);

endmodule

module pg\_grouped(input pi\_k, gi\_k, pk\_j, gk\_j, output pi\_j, gi\_j);

wire temp;

and2 and\_0(pi\_k, pk\_j, pi\_j);

and2 and\_1(pi\_k, gk\_j, temp);

or2 or\_1(gi\_k, temp, gi\_j);

endmodule

module sum(input gi\_blk, ai, bi, output sum\_i);

wire temp;

xor2 xor\_0(ai, bi, temp);

xor2 xor\_1(gi\_blk, temp, sum\_i);

endmodule

module carry(input p\_block, g\_block, cin, output cout);

wire temp;

and2 and\_0(p\_block, cin, temp);

or2 or\_0(g\_block, temp, cout);

endmodule

//8-bit Prefix Adder

module prefixAdder(input [7:0]ai, bi, output [7:0]sum);

//p\_minus1 and g\_minus1 are = 0 for the first prefix adder block

assign p\_minus1 = 0;

assign g\_minus1 = 0;

wire [7:0]pi, gi;

wire p0\_minus1, p2\_1, p4\_3, p6\_5, p1\_minus1, p2\_minus1, p5\_3, p6\_3, p3\_minus1, p4\_minus1, p5\_minus1, p6\_minus1, p7\_minus1;

wire g0\_minus1, g2\_1, g4\_3, g6\_5, g1\_minus1, g2\_minus1, g5\_3, g6\_3, g3\_minus1, g4\_minus1, g5\_minus1, g6\_minus1, g7\_minus1;

/\*

assign p\_minus1=1'b0;

assign g\_minus1=1'b0; // pi and gi for -1th column is 0

\*/

//Generation of pi, gi

pg p0g0(ai[0], bi[0], pi[0], gi[0]);

pg p1g1(ai[1], bi[1], pi[1], gi[1]);

pg p2g2(ai[2], bi[2], pi[2], gi[2]);

pg p3g3(ai[3], bi[3], pi[3], gi[3]);

pg p4g4(ai[4], bi[4], pi[4], gi[4]);

pg p5g5(ai[5], bi[5], pi[5], gi[5]);

pg p6g6(ai[6], bi[6], pi[6], gi[6]);

pg p7g7(ai[7], bi[7], pi[7], gi[7]);

//Generation of gi:-1 for all i

//Level 1

pg\_grouped pg0\_minus1(pi[0], gi[0], p\_minus1, g\_minus1, p0\_minus1, g0\_minus1);

pg\_grouped pg2\_1(pi[2], gi[2], pi[1], gi[1], p2\_1, g2\_1);

pg\_grouped pg4\_3(pi[4], gi[4], pi[3], gi[3] ,p4\_3, g4\_3);

pg\_grouped pg6\_5(pi[6], gi[6], pi[5], gi[5] ,p6\_5, g6\_5);

//Level 2

pg\_grouped pg1\_minus1(pi[1], gi[1], p0\_minus1, g0\_minus1, p1\_minus1, g1\_minus1);

pg\_grouped pg2\_minus1(p2\_1, g2\_1, p0\_minus1, g0\_minus1, p2\_minus1, g2\_minus1);

pg\_grouped pg5\_3(pi[5], gi[5], p4\_3, g4\_3, p5\_3,g5\_3);

pg\_grouped pg6\_3(p6\_5, g6\_5, p4\_3, g4\_3, p6\_3,g6\_3);

//Level 3

pg\_grouped pg3\_minus1(pi[3], gi[3], p2\_minus1, g2\_minus1,p3\_minus1,g3\_minus1);

pg\_grouped pg4\_minus1(p4\_3, g4\_3, p2\_minus1, g2\_minus1,p4\_minus1,g4\_minus1);

pg\_grouped pg5\_minus1(p5\_3,g5\_3, p2\_minus1, g2\_minus1,p5\_minus1,g5\_minus1);

pg\_grouped pg6\_minus1(p6\_3,g6\_3, p2\_minus1, g2\_minus1,p6\_minus1,g6\_minus1);

//Level 4

pg\_grouped pg7\_minus1(pi[7], gi[7], p6\_minus1, g6\_minus1,p7\_minus1,g7\_minus1);

sum s0(g\_minus1, ai[0], bi[0], sum[0]);

sum s1(g0\_minus1, ai[1], bi[1], sum[1]);

sum s2(g1\_minus1, ai[2], bi[2], sum[2]);

sum s3(g2\_minus1, ai[3], bi[3], sum[3]);

sum s4(g3\_minus1, ai[4], bi[4], sum[4]);

sum s5(g4\_minus1, ai[5], bi[5], sum[5]);

sum s6(g5\_minus1, ai[6], bi[6], sum[6]);

sum s7(g6\_minus1, ai[7], bi[7], sum[7]);

//assign cout=g7\_minus1 + p7\_minus1\*g\_minus1;

carry c\_0(p7\_minus1, g7\_minus1, g\_minus1, cout);

endmodule

**Output:**

