VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING

—o0o—

HOMEWORK REPORT

Chương 3 - Thiết kế mạch tổ hợp

SUPERVISOR: Nguyễn Trung Hiếu

SUBJECT: Digital System Design and Verification (EE3213)

GROUP: 04

List of Members

STT	MSSV	Họ Và Tên	Lớp
1	2213874	Nguyễn Thanh Tùng	L01
2	2210780	Nguyễn Đại Đồng	L01
3	2213496	Nguyễn Quốc Tín	L01

Ho Chi Minh, ../../20..

Mục lục

Danh sách hình vẽ

Danh sách bảng

List of Listings

Câu 3

Thiết kế mạch tính tổng của 2 số 32-bit sử dụng giải thuật CLA (Carry Look-Ahead Adder) trong lý thuyết. Lưu ý: tách thành các bộ cộng 4-bit CLA.

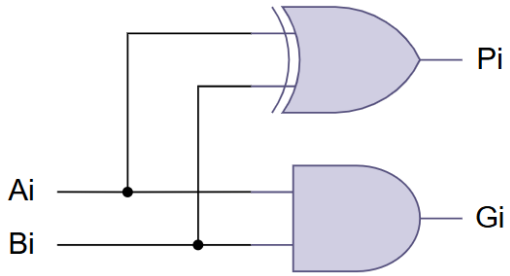
Cho các standard cell là: cổng not, các cổng logic 2, 3, 4 ngõ vào.

a) Thiết kế mạch theo phương pháp đã cho và chỉ được dùng các standard cell trên.

Theo lý thuyết, ta định nghĩa được 2 tín hiệu quan trọng là:

- **Generate ($\mathbf{G_i}$)**: Sinh carry ngay tại bit đó: $G_i = A_i \& B_i$.
- **Propagate ($\mathbf{P_i}$)**: Cho phép carry từ bit trước đi qua: $P_i = A_i \oplus B_i$.

Dựa vào đó, ta thiết kế PG Generator (Propagate–Generate Generator) như sau:

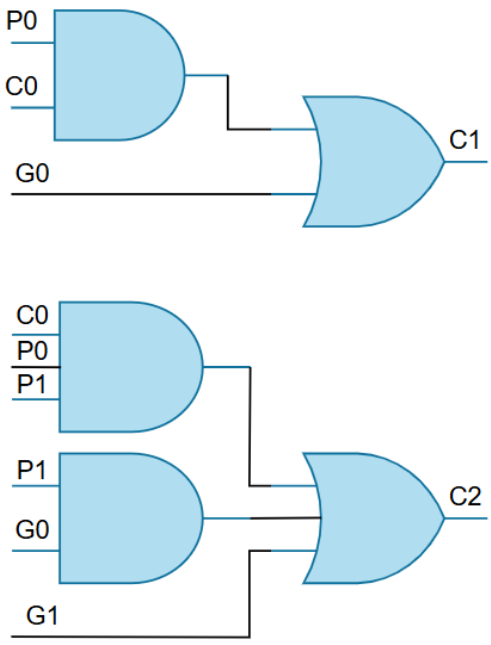


Hình 1: Sơ đồ logic của PG Generator.

Các tín hiệu G_i và P_i chính là đầu vào cho **Carry Generator** ở cấp kế tiếp. Biểu thức cho carry kế tiếp: $C_{i+1} = G_i | (P_i \& C_{in})$. Nếu thực hiện thế vào liên tục, ta được biểu thức như sau:

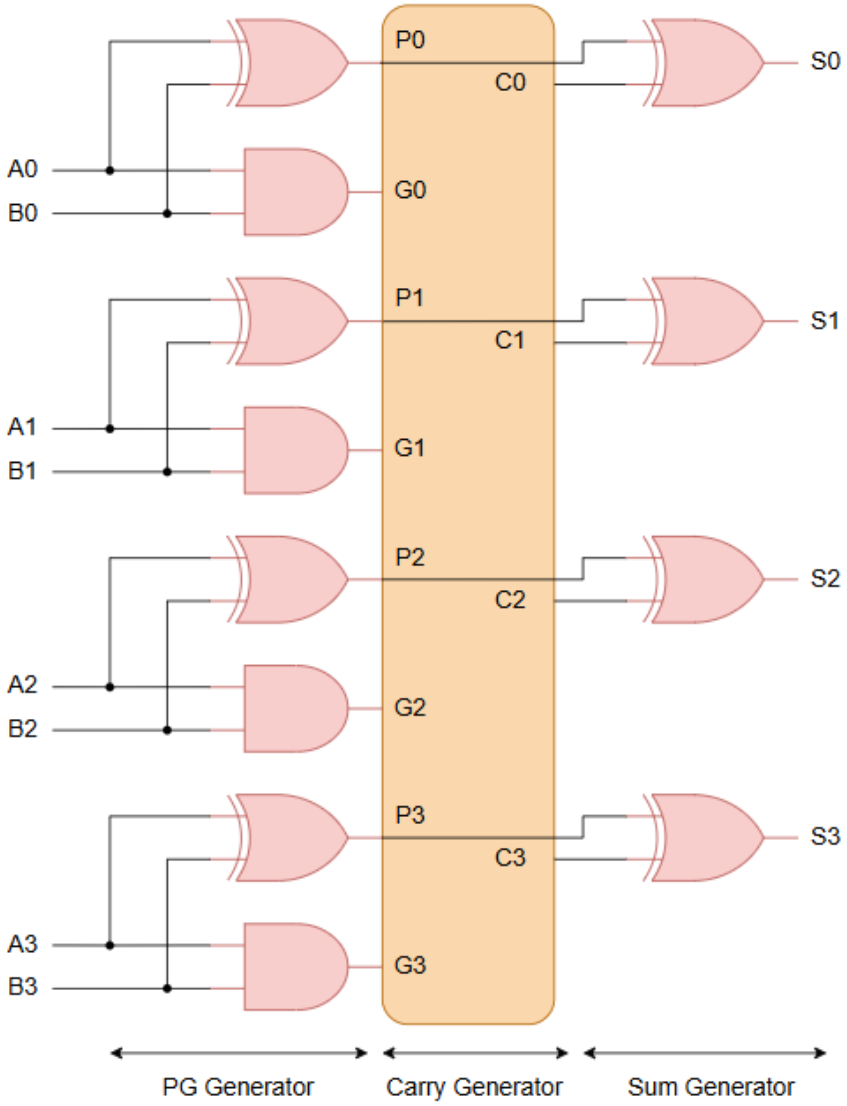
$$C_1 = G_0 | (P_0 \& C_{in}) \rightarrow C_2 = G_2 | (G_0 \& P_1) | (P_1 \& P_2 \& C_{in}) \dots$$

Tức là *tất cả các carry có thể tính song song* bằng các phép logic, không cần đợi ripple từng bit. Từ đó ta có thiết kế Carry Generator như sau:



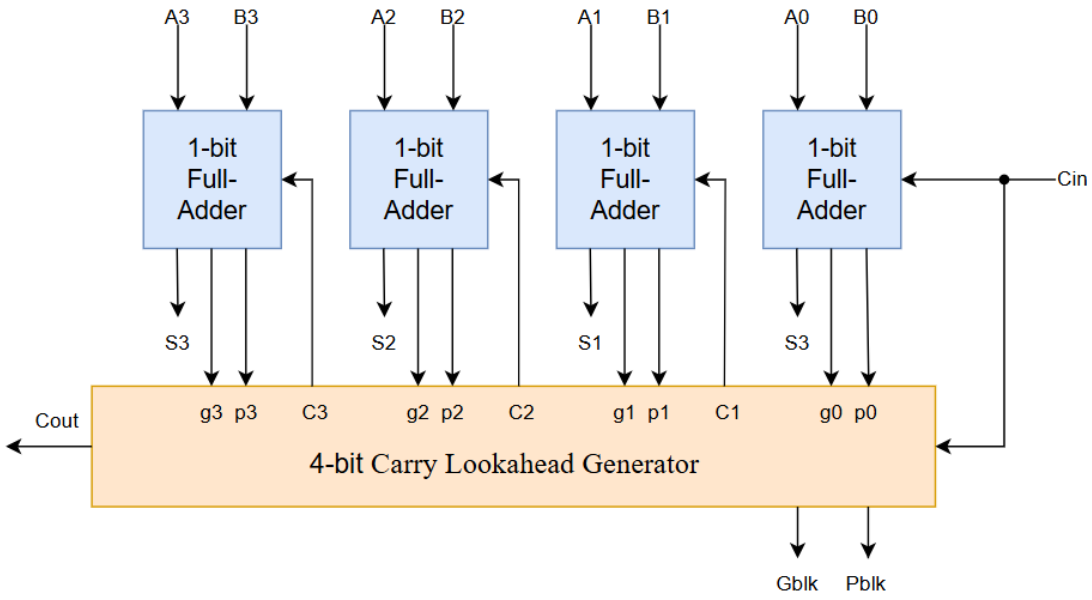
Hình 2: Sơ đồ logic của Carry Generator.

Sau khi biết các biết carry, ta tính tổng: $S_i = P_i \oplus C_{in}$. Từ đó, ta thiết kế bộ cộng CLA 4-bit như sau:



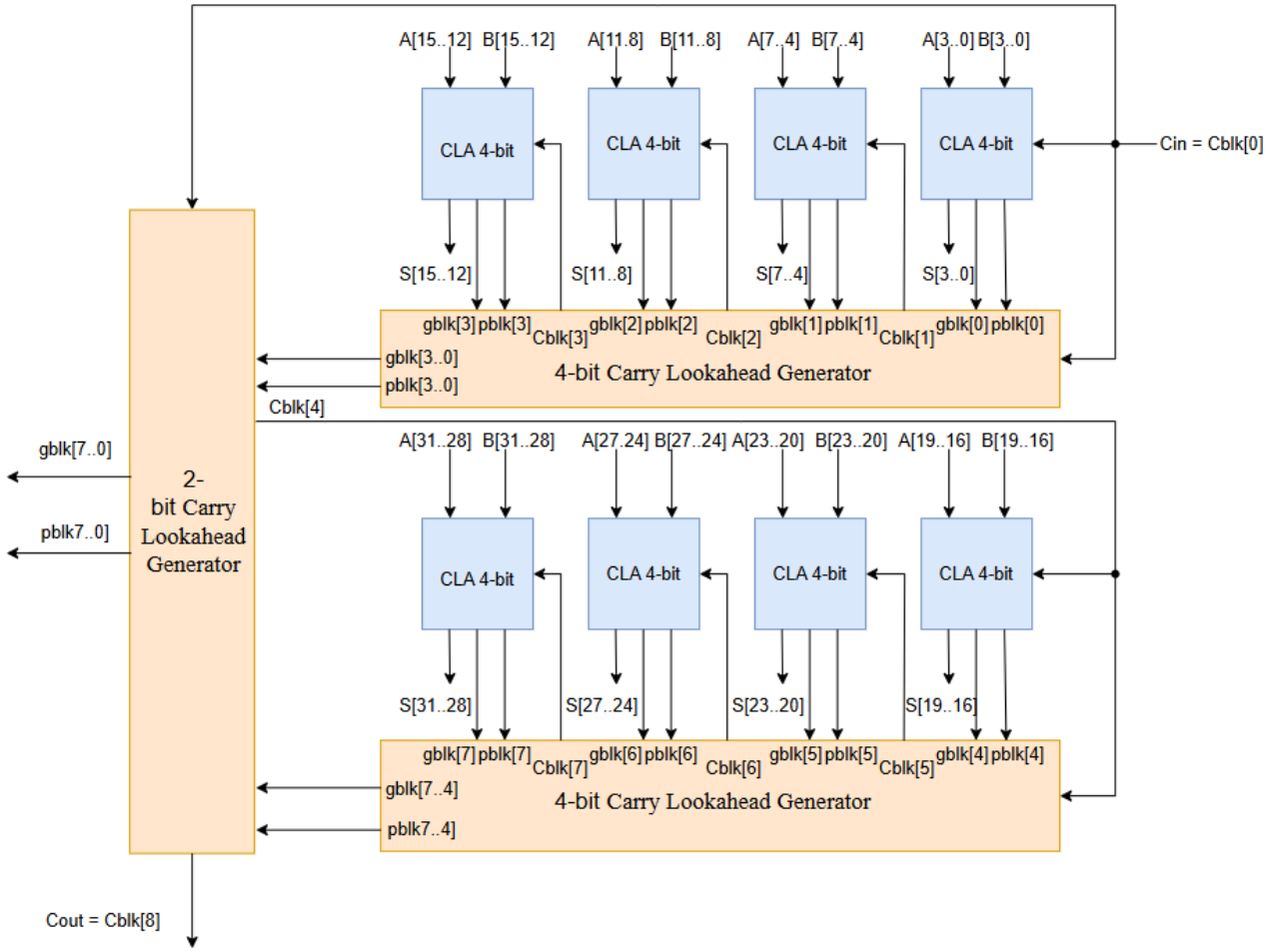
Hình 3: Sơ đồ logic của bộ cộng 4-bit CLA.

Ngoài ra, cũng là sơ đồ logic ấy, ta có thể biểu diễn thiết kế khác như hình bên dưới:



Hình 4: Block diagram của bộ cộng 4-bit CLA.

Từ những bộ trên, ta có được bộ tính tổng 2 số 32-bit bằng bộ cộng theo phương pháp CLA.



Hình 5: Block diagram của bộ cộng 32-bit CLA.

b) Viết chương trình HDL mô tả mạch đã cho.

```
module cla_4bit (
    input  logic [3:0]  A,
   input  logic [3:0]   B,
   input  logic         Cin,
   output logic         Gblk,
   output logic         Pblk,
   output logic [3:0]   S,
   output logic         Cout
);
    logic [3:0]p,g ;
    logic c1, c2, c3;
    assign p    = A ^ B ;
   assign g    = A & B ;
   assign c1   = g[0] | (p[0] & Cin);
   assign c2   = g[1] | (g[0] & p[1]) | (p[1] & p[0] & Cin);
    assign c3   = g[2] | (g[1] & p[2]) | (g[0] & p[1] & p[2]) | (p[2] & p[1] & p[0] & Cin);

   logic   and4, or3;
   assign  and4 = p[3] & p[2] & p[1] & p[0] ;
   assign  or3  = (g[2] & p[3]) | (g[1] & p[3] & p[2]) | (g[0] & p[3] & p[2] & p[1]);
```

```
21
22      assign Cout = g[3] | or3 | (and4 & Cin) ;
23      assign S[0] = p[0] ^ Cin;
24      assign S[1] = p[1] ^ c1;
25      assign S[2] = p[2] ^ c2;
26      assign S[3] = p[3] ^ c3;
27
28      assign Gblk     =  g[3] | or3;
29      assign Pblk     =  and4;
30
31
32  endmodule
```

Listing 1: Chương trình mô tả CLA 4-bit.

```
1   module cla_32bit (
2       input  logic        clk,
3       input  logic        rst_n,
4       input  logic [31:0] A,
5       input  logic [31:0] B,
6       input  logic        Cin,
7       output logic [31:0] Sum,
8       output logic        Cout
9   );
10
11      logic [31:0] A_r, B_r;
12      logic        Cin_r;
13
14      always_ff @(posedge clk or negedge rst_n) begin
15          if (!rst_n) begin
16              A_r   <= '0;
17              B_r   <= '0;
18              Cin_r <= 1'b0;
19          end else begin
20              A_r   <= A;
21              B_r   <= B;
22              Cin_r <= Cin;
23          end
24      end
25
26      logic [7:0] Pblk, Gblk;
27      logic [8:0] Cblk;
28      logic [31:0] Sum_c;
29      logic        Cout_c;
30
31      assign Cblk[0] = Cin_r;
32
33      genvar i;
34      generate
35          for (i = 0; i < 8; i++) begin : BLK
36              cla_4bit u4 (
37                  .A    (A_r[4*i +: 4]),
38                  .B    (B_r[4*i +: 4]),
39                  .cin  (Cblk[i]),
40                  .sum  (Sum_c[4*i +: 4]),
41                  .cout (),
42                  .Gblk (Gblk[i]),
43                  .Pblk (Pblk[i])
44              );
45              assign Cblk[i+1] = Gblk[i] | (Pblk[i] & Cblk[i]);
46          end
47      endgenerate
48
```

```
49      assign Cout_c = Cblk[8];
50
51
52      always_ff @(posedge clk or negedge rst_n) begin
53          if (!rst_n) begin
54              Sum  <= '0;
55              Cout <= 1'b0;
56          end else begin
57              Sum  <= Sum_c;
58              Cout <= Cout_c;
59          end
60      end
61  endmodule
```

Listing 2: Chương trình mô tả CLA 32-bit.

c) Viết testbench cho mạch. Testbench thực hiện rải 100 mẫu và tính scoreboard của 100 mẫu đó.

Nhóm em sử dụng dụng dấu + làm golden model để thực hiện kiểm tra kết quả test như sau:

```
1     expected = {1'b0, tv_a} + {1'b0, tv_b} + tv_cin;
```

Listing 3: Giải thuật chứng minh kết quả của bộ cộng CLA 32-bit.

- Thực hiện kiểm thử (self-checking) với 100 mẫu random để áp vào DUT. So sánh với expected để cập nhật scoreboard.

```
1         $display("=== Start run_test (100 samples) ===");
2         for (idx = 0; idx < 100; idx++) begin
3            // --- generate testcase ---
4            tv_a   = $urandom();
5            tv_b   = $urandom();
6            tv_cin = $urandom_range(0,1);
7            A_tb   = tv_a;
8            B_tb   = tv_b;
9            Cin_tb = tv_cin;
10           @(posedge clk);
11           @(posedge clk);
12           #1;
13           // --- compute expected ---
14           expected = {1'b0, tv_a} + {1'b0, tv_b} + tv_cin;
15           got      = {Cout_tb, Sum_tb};
16           // --- compare and display ---
17           if (got == expected) begin
18              pass_count++;
19              $display("PASS [%0d] A=0x%08h B=0x%08h Cin=%0d => {Cout,Sum}=0x%09h",
20              idx, tv_a, tv_b, tv_cin, got);
21           end else begin
22              fail_count++;
23              $display("FAIL [%0d] A=0x%08h B=0x%08h Cin=%0d => got=0x%09h (exp=0x%09h)",
24              idx, tv_a, tv_b, tv_cin, got, expected);
```

```
25            end
26        end
27
```

Listing 4: Sinh 100 mẫu random và thực hiện kiểm tra.

Kết quả

```
# === Start run_test (100 samples) ===
# PASS [0] A=0x2e45b278 B=0xd7aeae53 Cin=1 => {Cout,Sum}=0x105f460cc
# PASS [1] A=0x1e3a4d4a B=0x4ba544f7 Cin=1 => {Cout,Sum}=0x069df9242
# PASS [2] A=0xf331e3ec B=0x55820c15 Cin=0 => {Cout,Sum}=0x148b3f001
# PASS [3] A=0xfb325326 B=0x4cd94c49 Cin=0 => {Cout,Sum}=0x1480b9f6f
...
# PASS [97] A=0xb68bc493 B=0x18b2dfcc Cin=0 => {Cout,Sum}=0x0cf3ea45f
# PASS [98] A=0x3662a0de B=0x9cc1dc3f Cin=0 => {Cout,Sum}=0x0d3247d1d
# PASS [99] A=0xafde0bca B=0xcf523f4e Cin=1 => {Cout,Sum}=0x17f304b19
```

Listing 5: Kết quả test từng mẫu.

- Kết quả tổng kết.

```
# === Test summary ===
# Total samples = 100
# PASS = 100
# FAIL = 0
# === End run_test ===
```

Listing 6: Kết quả của tổng kết của bài test.