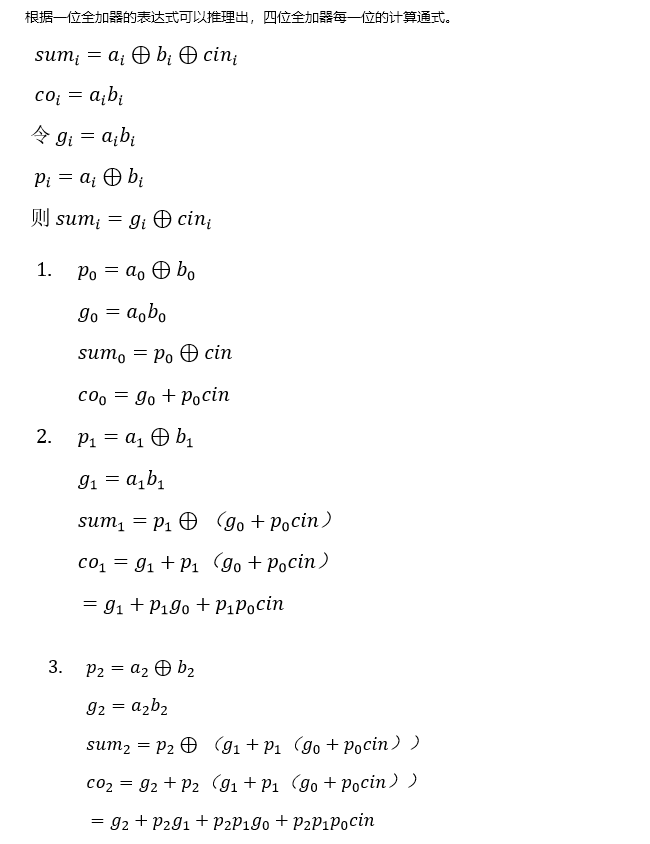
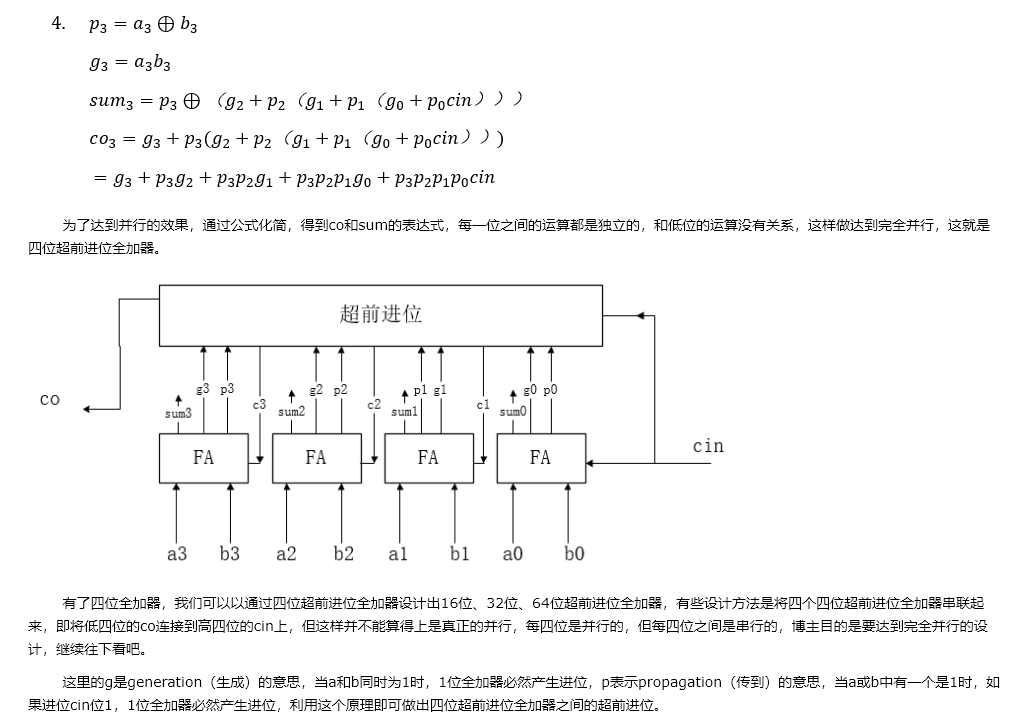
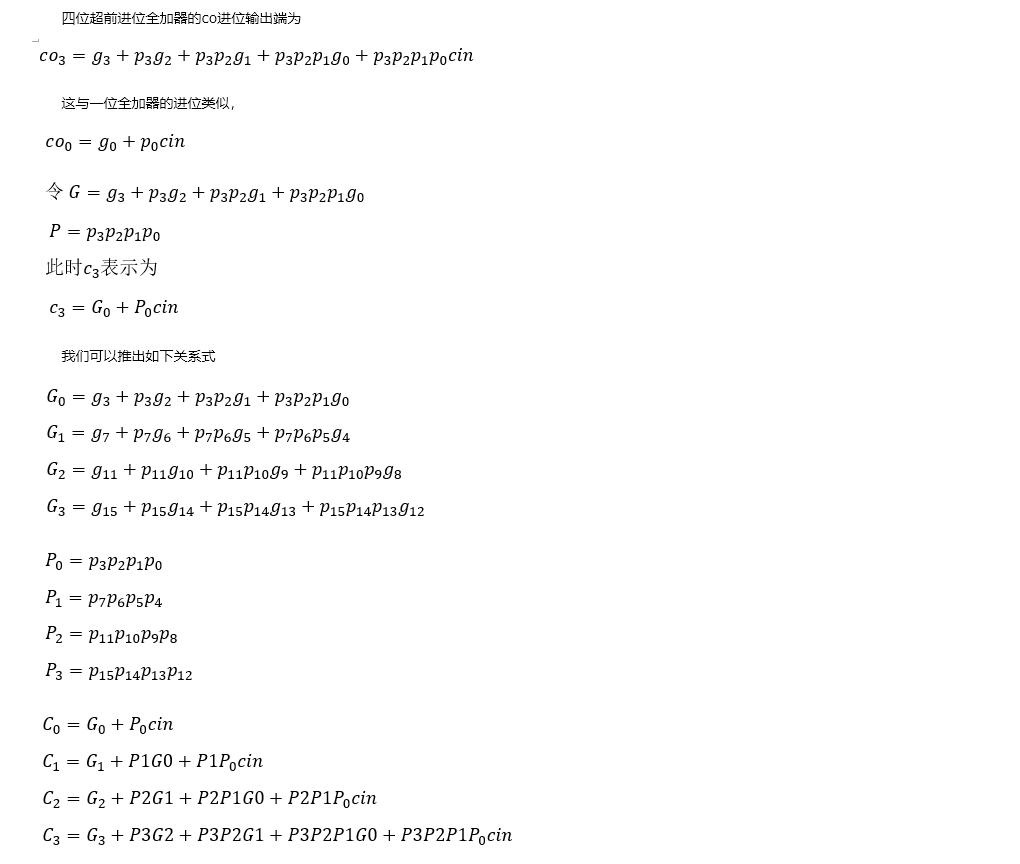
**四位超前进位加法器**

原理如下：







在Verilog中表示为:

module full4\_adder3(

input [3:0] a,b,

input cin,

output [3:0] sum,

output cout

);

wire [3:0] G, P;

wire [3:0] sum;

assign G[0] = a[0] & b[0];

assign P[0] = a[0] ^ b[0];

assign sum[0] = P[0] ^ cin;

assign P[1] = a[1] ^ b[1];

assign G[1] = a[1] & b[1];

assign sum[1] = P[1] ^ (G[0] | (P[0] & cin));

assign P[2] = a[2] ^ b[2];

assign G[2] = a[2] & b[2];

assign sum[2] = P[2] ^ (G[1] | (P[1] & (G[0] | (P[0] & cin))));

assign P[3] = a[3] ^ b[3];

assign G[3] = a[3] & b[3];

assign sum[3] = P[3] ^ (G[2] | P[2] & (G[1] | (P[1] & (G[0] | (P[0] & cin)))));

assign cout = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) | (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1] & P[0] & cin);

endmodule

测试文件为：（就用了四个样例，没有全部测试）

module tb\_full4\_adder();

wire [3:0] Sum;

wire cout;

reg [3:0] Ain;

reg [3:0] Bin;

reg cin;

initial

begin

//0000 0001 1

Ain = 4'b0000;

Bin = 4'b0001;

cin = 1;

//0001 00110

#10

Ain = 4'b0001;

Bin = 4'b0011;

cin = 0;

//0111 0010 1

#10

Ain = 4'b0111;

Bin = 4'b0010;

cin = 1;

//1111 1111 0

#10

Ain = 4'b1111;

Bin = 4'b1111;

cin = 0;

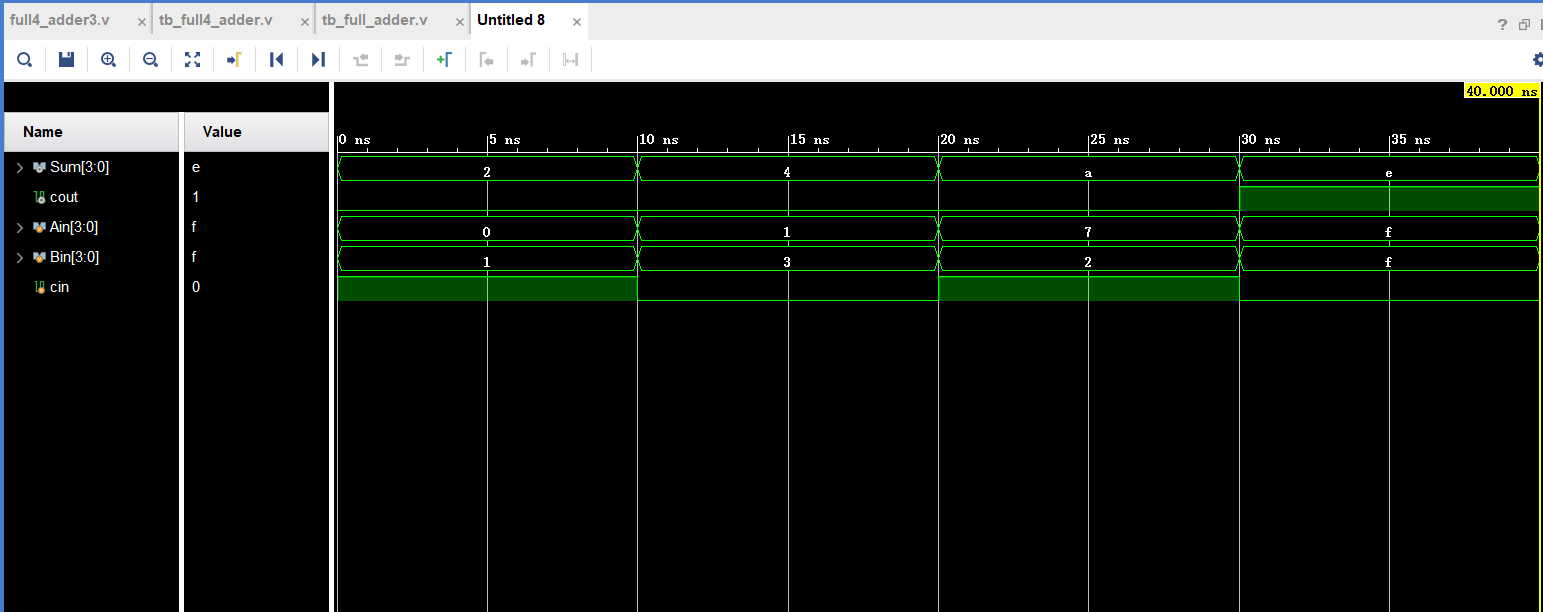
#10 $stop;

end

full4\_adder3 f4adder3(Ain, Bin, cin, Sum, cout);

endmodule

仿真结果为：



**乘法器**

串行乘法器：

module multiply1(clk, x, y, result);

input clk;

input [7:0] x, y;

output [15:0] result;

reg [15:0] result;

parameter s0 = 0, s1 = 1, s2 = 2;

reg [2:0] count = 0;

reg [1:0] state = 0;

reg [15:0] P, T;

reg [7:0] y\_reg;

always @(posedge clk) begin

case (state)

s0: begin

count <= 0;

P <= 0;

y\_reg <= y;

T <= {{8{1'b0}}, x};

state <= s1;

end

s1: begin

if(count == 3'b111)

state <= s2;

else begin

if(y\_reg[0] == 1'b1)

P <= P + T;

else

P <= P;

y\_reg <= y\_reg >> 1;

T <= T << 1;

count <= count + 1;

state <= s1;

end

end

s2: begin

result <= P;

state <= s0;

end

default: ;

endcase

end

endmodule

流水线乘法器：

module multiply2(mul\_a, mul\_b, clk, rst\_n, mul\_out);

input [3:0] mul\_a, mul\_b;

input clk;

input rst\_n;

output [15:0] mul\_out;

reg [15:0] mul\_out;

reg [15:0] stored0;

reg [15:0] stored1;

reg [15:0] stored2;

reg [15:0] stored3;

reg [15:0] stored4;

reg [15:0] stored5;

reg [15:0] stored6;

reg [15:0] stored7;

reg [15:0] mul\_out01;

reg [15:0] mul\_out23;

reg [15:0] add01;

reg [15:0] add23;

reg [15:0] add45;

reg [15:0] add67;

always @(posedge clk or negedge rst\_n)

begin

if(!rst\_n)

begin

mul\_out <= 0;

stored0 <= 0;

stored1 <= 0;

stored2 <= 0;

stored3 <= 0;

stored4 <= 0;

stored5 <= 0;

stored6 <= 0;

stored7 <= 0;

add01 <= 0;

add23 <= 0;

add45 <= 0;

add67 <= 0;

end

else begin

stored0 <= mul\_b[0]? {8'b0, mul\_a} : 16'b0;

stored1 <= mul\_b[1]? {7'b0, mul\_a, 1'b0} : 16'b0;

stored2 <= mul\_b[2]? {6'b0, mul\_a, 2'b0} : 16'b0;

stored3 <= mul\_b[3]? {5'b0, mul\_a, 3'b0} : 16'b0;

stored4 <= mul\_b[0]? {4'b0, mul\_a, 4'b0} : 16'b0;

stored5 <= mul\_b[1]? {3'b0, mul\_a, 5'b0} : 16'b0;

stored6 <= mul\_b[2]? {2'b0, mul\_a, 6'b0} : 16'b0;

stored7 <= mul\_b[3]? {1'b0, mul\_a, 7'b0} : 16'b0;

add01 <= stored1 + stored0;

add23 <= stored3 + stored2;

add45 <= stored5 + stored4;

add67 <= stored7 + stored6;

mul\_out01 <= add01 + add23;

mul\_out23 <= add45 + add67;

mul\_out <= mul\_out01 + mul\_out23;

end

end

endmodule