Munteanu Mircea, Munteanu Daniel AC anul II CTI RO, sg. 4.2

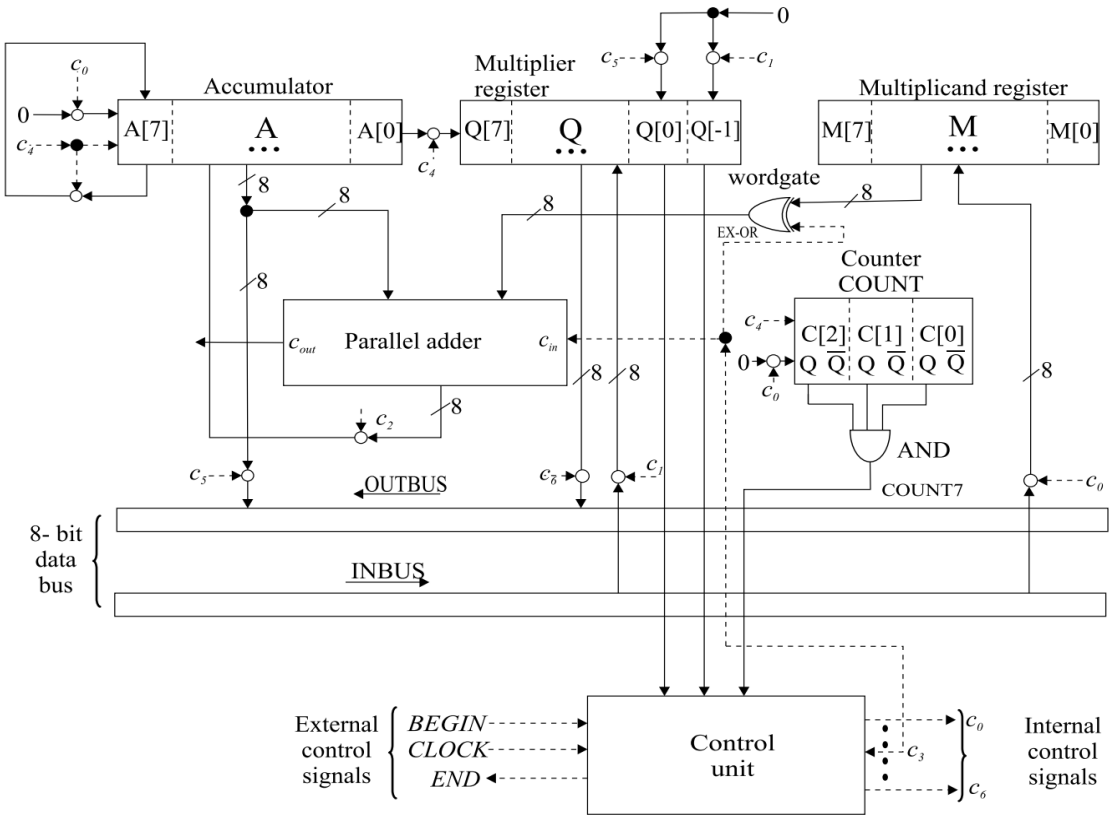
Project: Booth Multiplier Implementation in Verilog

The Booth multiplication algorithm is a method used to multiply binary numbers more efficiently.

Instead of performing additions for each bit, the algorithm identifies sequences of identical bits and replaces them with shift and add/subtract operations.

This helps reduce the total number of operations required and speeds up the multiplication process.

Schema Hardware



Computer Arithmetic - M. Vladutiu pg87

Carry Look Ahead

```
module CLA (input [7:0]x, y, input c_in, output [7:0]z);
  wire [6:0]g0;
  wire [6:0]gl;
  wire [6:0]p0;
  wire [6:0]pl;
  wire [6:0]c0;
  wire [6:0]cl;
  wire g,p;

  AC ac7(.x(x[7]),.y(y[7] ^ c_in),.c(cl[6]),.g(g0[6]),.p(p0[6]),.z(z[7]));
  AC ac6(.x(x[6]),.y(y[6] ^ c_in),.c(c0[6]),.g(gl[6]),.p(pl[6]),.z(z[6]));
  AC ac5(.x(x[5]),.y(y[5] ^ c_in),.c(cl[5]),.g(g0[5]),.p(p0[5]),.z(z[5]));
  AC ac4(.x(x[4]),.y(y[4] ^ c_in),.c(c0[5]),.g(gl[5]),.p(pl[5]),.z(z[4]));
  AC ac3(.x(x[3]),.y(y[3] ^ c_in),.c(cl[4]),.g(g0[4]),.p(p0[4]),.z(z[3]));
  AC ac2(.x(x[2]),.y(y[2] ^ c_in),.c(c0[4]),.g(gl[4]),.p(pl[4]),.z(z[2]));
  AC ac1(.x(x[1]),.y(y[1] ^ c_in),.c(cl[3]),.g(g0[3]),.p(p0[3]),.z(z[1]));
  AC ac0(.x(x[0]),.y(y[0] ^ c_in),.c(c0[3]),.g(gl[3]),.p(pl[3]),.z(z[0]));

  BC bc6(.g0_in(g0[6]),.p0_in(p0[6]),.gl_in(gl[6]),.pl_in(pl[6]),.c_in(cl[2]),.g_out(g0[2]),.p_out(p0[2]),.c0_out(c0[6]),.cl_out(cl[6]));
  BC bc5(.g0_in(g0[5]),.p0_in(p0[5]),.gl_in(gl[5]),.pl_in(pl[5]),.c_in(c0[2]),.g_out(gl[2]),.p_out(pl[2]),.c0_out(c0[5]),.cl_out(cl[5]));
  BC bc4(.g0_in(g0[4]),.p0_in(p0[4]),.gl_in(gl[4]),.pl_in(pl[4]),.c_in(cl[1]),.g_out(g0[1]),.p_out(p0[1]),.c0_out(c0[4]),.cl_out(cl[4]));
  BC bc3(.g0_in(g0[3]),.p0_in(p0[3]),.gl_in(gl[3]),.pl_in(pl[3]),.c_in(c0[1]),.g_out(gl[1]),.p_out(pl[1]),.c0_out(c0[3]),.cl_out(cl[3]));
  BC bc2(.g0_in(g0[2]),.p0_in(p0[2]),.gl_in(gl[2]),.pl_in(pl[2]),.c_in(cl[0]),.g_out(g0[0]),.p_out(p0[0]),.c0_out(c0[2]),.cl_out(cl[2]));
  BC bc1(.g0_in(g0[1]),.p0_in(p0[1]),.gl_in(gl[1]),.pl_in(pl[1]),.c_in(c0[0]),.g_out(gl[0]),.p_out(pl[0]),.c0_out(c0[1]),.cl_out(cl[1]));
  BC bc0(.g0_in(g0[0]),.p0_in(p0[0]),.gl_in(gl[0]),.pl_in(pl[0]),.c_in(c_in),.g_out(g),.p_out(p),.c0_out(c0[0]),.cl_out(cl[0]));

endmodule
```

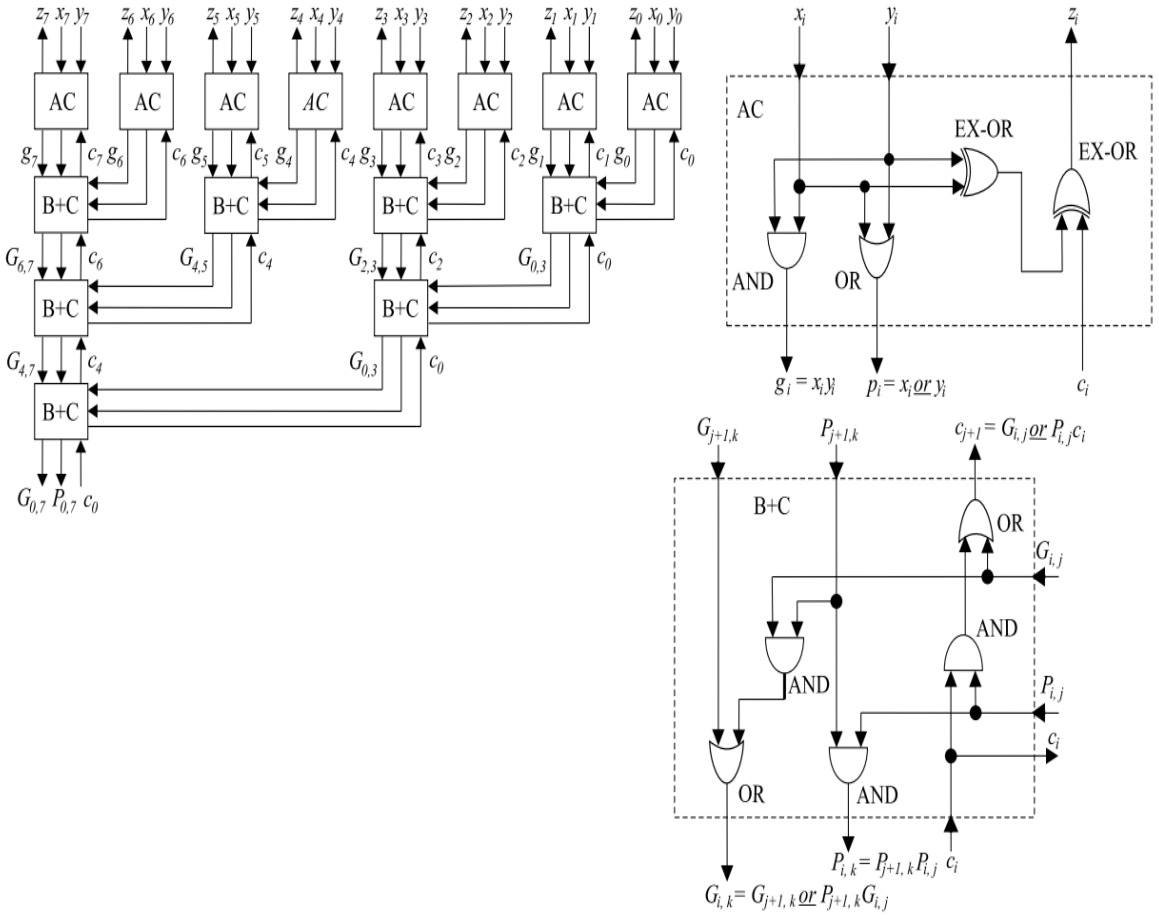
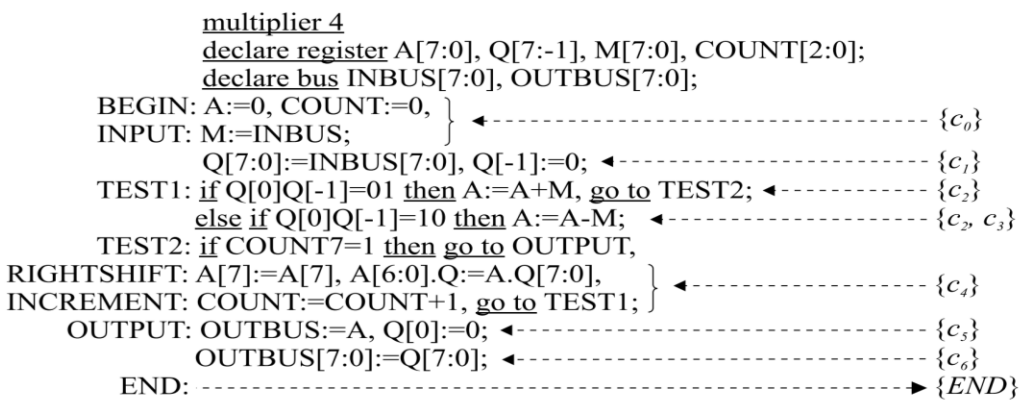


Fig. 2.21 Block diagram of an 8-bit multilevel CLA and gate level implementations of its cells

Control Unit



Computer Arithmetic - M. Vladutiu pg86

```
1   module control_unit (input clk, rst_b, start, counted7, input [0:-1]q, output reg [6:0]c);
2
3     // SHIFT <--> TEST
4     localparam START = 0;
5     localparam SCAN = 1;
6     localparam SHIFT = 2;
7     localparam TEST = 3;
8     localparam OUTPUT = 4;
9     localparam END = 5;
10
11    reg [2:0]state, state_next;
12
13    //Se stabileste urmatorea stare in functie de starea curenta si input
14    always@(*)
15      begin
16        state_next = state;
17        case(state)
18          START: if(start == 1) state_next = SCAN;
19          SCAN: state_next = SHIFT;
20          SHIFT: state_next = TEST;
21          TEST:
22            begin
23              if(counted7 == 1)
24                state_next = OUTPUT;
25              else
26                state_next = SCAN;
27            end
28          OUTPUT: state_next = END;
29        endcase
30      end
31
32    //Se stabilesc valorile semnalelor c in functie de valorile lui q
33    always@(*)
34      begin
35        c = 0;
36        case(state)
37          START:
38            begin
39              c[0] = 1;
```

```
36        case(state)
37          START:
38            begin
39              c[0] = 1;
40              c[1] = 1;
41            end
42          SCAN:
43            begin
44              if(q == 2'b01)
45                c[2] = 1;
46              else
47                if(q == 2'b10)
48                  begin
49                    c[2] = 1;
50                    c[3] = 1;
51                  end
52            end
53          SHIFT: c[4] = 1;
54          OUTPUT:
55            begin
56              c[5] = 1;
57              c[6] = 1;
58            end
59        endcase
60      end
61
62      always@(posedge clk, negedge rst_b)
63        begin
64          if(~rst_b)
65            begin
66              state <= START;
67              c = 0;
68            end
69          else
70            state <= state_next;
71        end
72
73  endmodule
```

Main Module

```
module multiplier (input clk, rst_b, start, input [15:0]INBUS, output [15:0]OUTBUS);

  // Daca vrei sa vezi valorile lui A,Q,c in simulare poti sa le pui la output in loc de wire

  wire [6:0]c;
  wire [7:0]addition_result;
  wire [7:0]A,M;
  wire [7:-1]Q;
  wire counter_is_7;

  control_unit controller(.clk(clk), .rst_b(rst_b), .start(start), .counted7(counter_is_7), .q({Q[0],Q[-1]}), .c(c));

  reg_m register_m(.clk(clk), .rst_b(rst_b), .ld_ibus(c[0]), .ibus(INBUS[15:8]), .q(M));
  reg_a register_a(.clk(clk), .rst_b(rst_b), .ld_sum(c[2]), .ld_obus(c[5]), .sh_r(c[4]), .sh_i(A[7]), .sum(addition_result), .obus(OUTBUS[15:8]), .q(A));
  reg_q register_q(.clk(clk), .rst_b(rst_b), .ld_ibus(c[1]), .ld_obus(c[6]), .clr_lsb(c[0]), .sh_r(c[4]), .sh_i(A[0]), .ibus(INBUS[7:0]), .obus(OUTBUS[7:0]), .q

  CLA adder(.x(A) ,.y(M) , .c_in(c[3]), .z(addition_result));

  counter counter7(.clk(clk), .rst_b(rst_b), .c_up(c[4]), .count_end(counter_is_7));

endmodule
```

```
module multiplier_tb;
  reg clk, rst_b, start;
  reg [15:0]INBUS;
  wire [15:0]OUTBUS;

  multiplier cut(.clk(clk), .rst_b(rst_b), .start(start), .INBUS(INBUS), .OUTBUS(OUTBUS));

  //clk
  initial begin
    clk = 0;
    forever #100 clk = ~clk;
  end

  //rst_b
  initial begin
    rst_b = 1;
    #10 rst_b = 0;
    #10 rst_b = 1;
  end

  //numere
  initial begin
    start = 0;
    INBUS[15:8] = 8'b01001010;
    INBUS[7:0] = 8'b11000001;
    #30 start = 1;
  end
endmodule
```

Bibliografie

Computer Arithmetic - M. Vladutiu