Analysys parameters: slow 1200mV 85C

Наши сумматоры!

Тип сумматора	Разрядность	Total logic el	Total reg	Fmax, MHz	Restricted Fmax, MHz
Serial adder	8	38	26	110.12	110.12
	16	74	50	70.65	70.65
	32	146	98	56.45	56.45
	64	290	194	30.65	30.65
	128	578	386	15.17	15.17
Carry lookahead	8	36	26	129.94	129.94
	16	72	50	69.77	69.77
	32	144	98	54.9	54.9
	64	288	194	31.46	31.46
	128	576	386	15.1	15.1

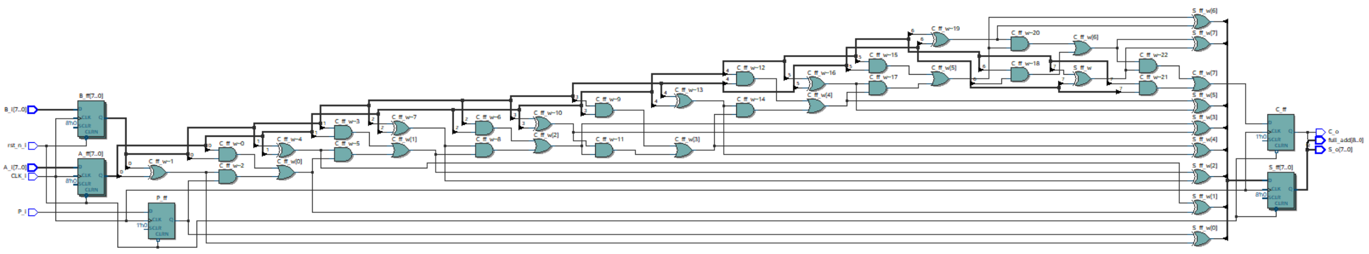
Сумматоры по примерам Панчула

Тип сумматора	Разрядность	Total logic el	Total reg	Fmax, MHz	Restricted Fmax, MHz
Serial adder	8	35	26	165.81	165.81
	16	69	50	75.8	75.8
	32	138	98	62.55	52.55
	64	276	194	45.3	45.33
	128	552	386	21.56	21.56
Carry lookahead	8	38	26	110.12	110.12
	16	74	50	70.65	70.65
	32	146	98	56.45	56.45
	64	290	194	30.65	30.65
	128	578	386	15.17	15.17

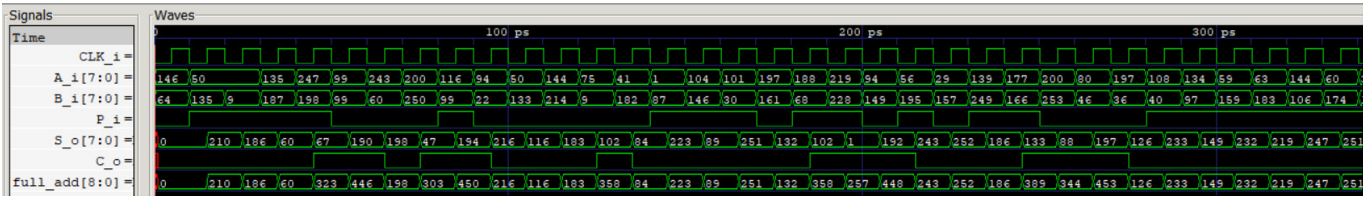
Если делать Carry lokahead по книге Панчула, то он полностью совпадает с нашим Serial adder по кол-ву logic и частотам, хотя по RTL-view они разные.

Последовательный сумматор

```
module serial_adder
    #(
        parameter W = 8
    )
    (
        input   logic             CLK_i,
        input   logic             rst_n_i,
        input   logic [ W-1 : 0 ] A_i,
        input   logic [ W-1 : 0 ] B_i,
        input   logic             P_i,
        output  logic [ W-1 : 0 ] S_o,
        output  logic             C_o,
        output  logic [ W   : 0 ] full_add
    );
        logic [ W-1 : 0 ] A_ff;
        logic [ W-1 : 0 ] B_ff;
        logic             P_ff;
        logic             C_ff;
        logic [ W-1 : 0 ] S_ff;
        logic [ W-1 : 0 ] A_ff_w;
        logic [ W-1 : 0 ] B_ff_w;
        logic [ W   : 0 ] P_ff_w;
        logic [ W-1 : 0 ] S_ff_w;
        logic [ W   : 0 ] C_ff_w;
    always_ff @(posedge CLK_i or negedge rst_n_i) begin
        if(!rst_n_i) begin
            A_ff <= 0;
            B_ff <= 0;
            P_ff <= 0;
        end
        else begin
            A_ff <= A_i;
            B_ff <= B_i;
            P_ff <= P_i;
        end
    end
    always_ff @(posedge CLK_i or negedge rst_n_i) begin
        if(!rst_n_i) begin
            S_ff <= 0;
            C_ff <= 0;
        end
        else begin
            S_ff <= S_ff_w;
            C_ff <= C_ff_w[W-1];
        end
    end
    assign A_ff_w   = A_ff;
    assign B_ff_w   = B_ff;
    assign P_ff_w[0] = P_ff;
    assign C_o       = C_ff;
    assign S_o       = S_ff;
    assign full_add  = {C_o, S_o};
    generate
        genvar i;
        for ( i = 0; i < W; i++) begin
            assign S_ff_w[i] = A_ff_w[i] ^ B_ff_w[i] ^ P_ff_w[i];
            assign C_ff_w[i] = (A_ff_w[i] & B_ff_w[i]) | (P_ff_w[i] & (A_ff_w[i] ^
B_ff_w[i]));
            assign P_ff_w[i+1] = C_ff_w[i];
        end
    endgenerate

endmodule
```

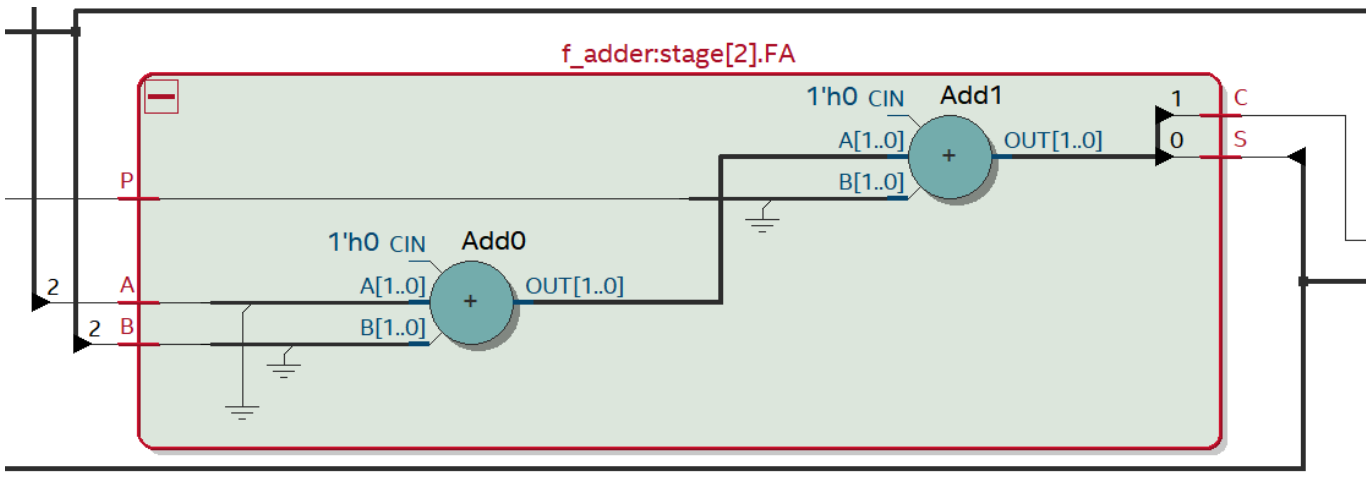


Waveform


Signals
Waves
Time 0 100 ps 200 ps 300 ps
CLK_i =
A_i[7:0] = 146 50 135 247 99 243 200 116 94 50 144 75 41 1 104 101 197 188 219 94 56 29 139 177 200 80 197 108 134 59 63 144 60
B_i[7:0] = 64 135 9 187 198 99 60 250 99 22 133 214 9 182 87 146 30 161 68 228 149 195 157 249 166 253 46 36 40 97 159 183 106 174
P_i =
S_o[7:0] = 0 210 186 60 67 190 198 47 194 216 116 183 102 84 223 89 251 132 102 1 192 243 252 186 133 88 197 126 233 149 232 219 247 251
C_o =
full_add[8:0] = 0 210 186 60 323 446 198 303 450 216 116 183 358 84 223 89 251 132 358 257 448 243 252 186 389 344 453 126 233 149 232 219 247 251

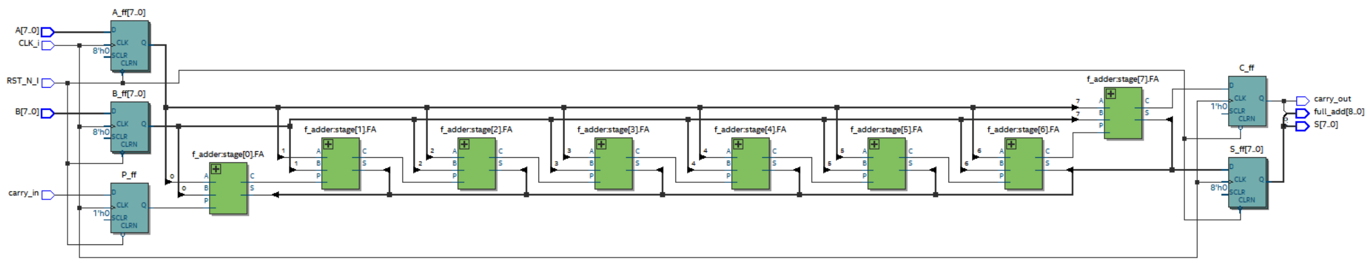
Ripple_carry_adder

```
module f_adder (A,B,P,S,C);
    input  logic   A;
    input  logic   B;
    input  logic   P;
    output logic   S;
    output logic   C;

    assign {C, S}= A+B+P;
endmodule
```

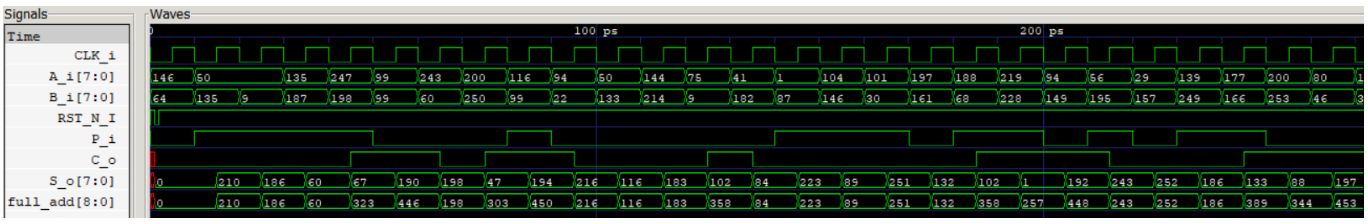


```
`include "f_adder.sv"
module ripple_carry_adder
# (
    parameter W = 8
)
(
    input                    carry_in,
    input  logic [W - 1 : 0]  A,
    input  logic [W - 1 : 0]  B,
    input  logic             CLK_i,
    input  logic             RST_N_I,
    output logic [W - 1 : 0]  S,
    output                   carry_out,
    output logic [W     : 0]  full_add
);
    logic  [W : 0] carry;

    logic [ W-1 : 0 ] A_ff;
    logic [ W-1 : 0 ] B_ff;
    logic             P_ff;
    logic             C_ff;
    logic [ W-1 : 0 ] S_ff;
    logic [ W-1 : 0 ] A_ff_w;
    logic [ W-1 : 0 ] B_ff_w;
    logic [ W   : 0 ] P_ff_w;
    logic [ W-1 : 0 ] S_ff_w;

    always_ff @(posedge CLK_i or negedge RST_N_I) begin
            if(!RST_N_I) begin
                A_ff <= 0;
                B_ff <= 0;
                P_ff <= 0;
                S_ff <= 0;
                C_ff <= 0;
            end
            else begin
                A_ff <= A;
                B_ff <= B;
                P_ff <= carry_in;
```

```
        S_ff <= S_ff_w;
        C_ff <= carry[W];
      end
    end

  assign carry[0]  = P_ff;
  assign A_ff_w    = A_ff;
  assign B_ff_w    = B_ff;
  assign S         = S_ff;
  assign carry_out = C_ff;
  generate
  genvar i;
    for (i = 0; i <= W - 1; i = i + 1) begin : stage
      f_adder
      FA(
        .A (A_ff_w [i] ),
        .B (B_ff_w [i] ),
        .S (S_ff_w [i] ),
        .P (carry[i] ),
        .C(carry[i + 1])
      );
    end
  endgenerate

  assign full_add = {carry_out, S};
endmodule
```



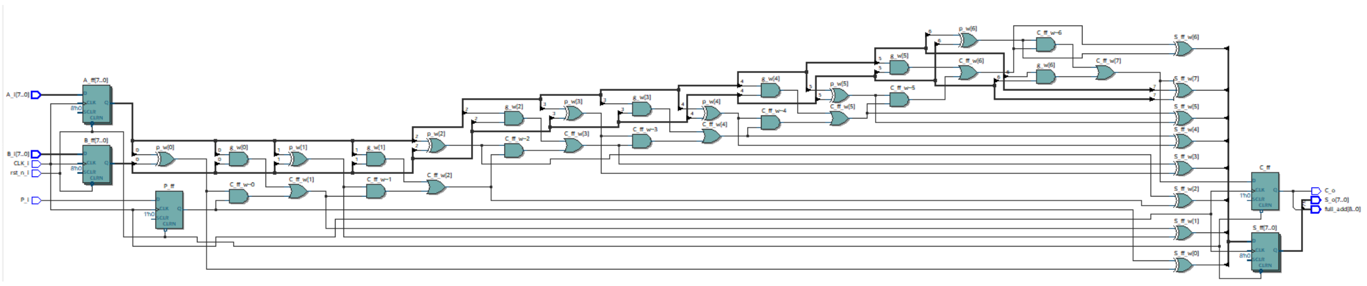
Waveform



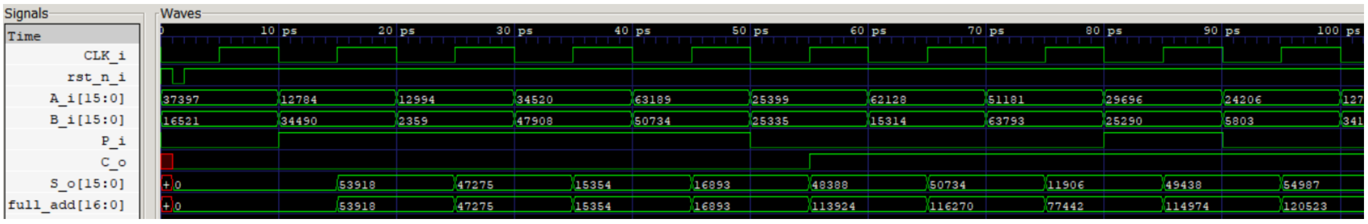
Сумматор с ускоренным переносом

Версия №1 и видимо не совсем правильная, хотя по waveform видно, что считает правильно

```
module carry_lookahead_adder
#(
    parameter W = 4
)
(
    input   logic             CLK_i,
    input   logic             rst_n_i,
    input   logic [ W-1 : 0 ] A_i,
    input   logic [ W-1 : 0 ] B_i,
    output  logic [ W-1 : 0 ] S_o,
    input   logic             P_i,

    output  logic             C_o,
    output  logic [ W   : 0 ] full_add
);



    logic [ W-1 : 0 ] A_ff;
    logic [ W-1 : 0 ] B_ff;
    logic             P_ff;
    logic             C_ff;
    logic [ W-1 : 0 ] S_ff;

    logic [ W-1 : 0 ] A_ff_w;
    logic [ W-1 : 0 ] B_ff_w;
    logic [ W-1 : 0 ] S_ff_w;
    logic [ W   : 0 ] C_ff_w;
    logic [ W-1 : 0 ] p_w;
    logic [ W-1 : 0 ] g_w;

    always_ff @(posedge CLK_i or negedge rst_n_i) begin
        if(!rst_n_i) begin
            A_ff <= 0;
            B_ff <= 0;
            P_ff <= 0;
        end
        else begin
            A_ff <= A_i;
            B_ff <= B_i;
            P_ff <= P_i;
        end
    end

    always_ff @(posedge CLK_i or negedge rst_n_i) begin
        if(!rst_n_i) begin
            S_ff <= 0;
            C_ff <= 0;
        end

        else begin
            S_ff <= S_ff_w;
            C_ff <= C_ff_w[W-1];
        end
    end

    assign A_ff_w   = A_ff;
    assign B_ff_w   = B_ff;

    assign C_ff_w[0] = P_ff;
    assign C_o       = C_ff;
    assign S_o       = S_ff;

    assign full_add  = {C_o, S_o};

    generate
        genvar i;
        for ( i = 0; i < W; i++) begin
            assign g_w[i] = A_ff_w[i] & B_ff_w[i] ;
            assign p_w[i] = A_ff_w[i] ^ B_ff_w[i] ;
            assign C_ff_w[i+1] = g_w[i] | (p_w[i] & C_ff_w[i]) ;
            assign S_ff_w[i] = A_ff_w[i] ^ B_ff_w[i] ^ C_ff_w[i];
        end
    endgenerate

endmodule
```



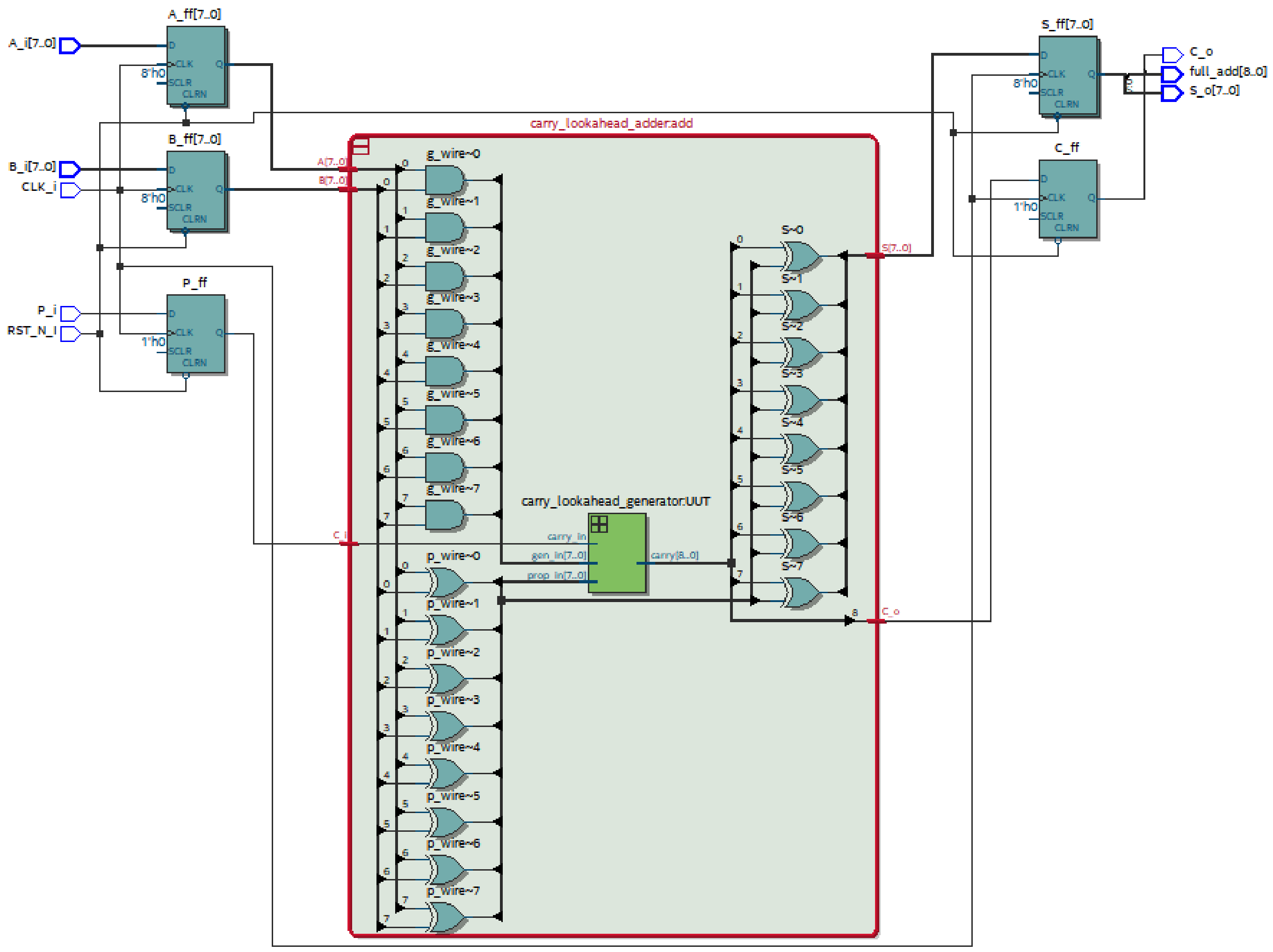
Waveform



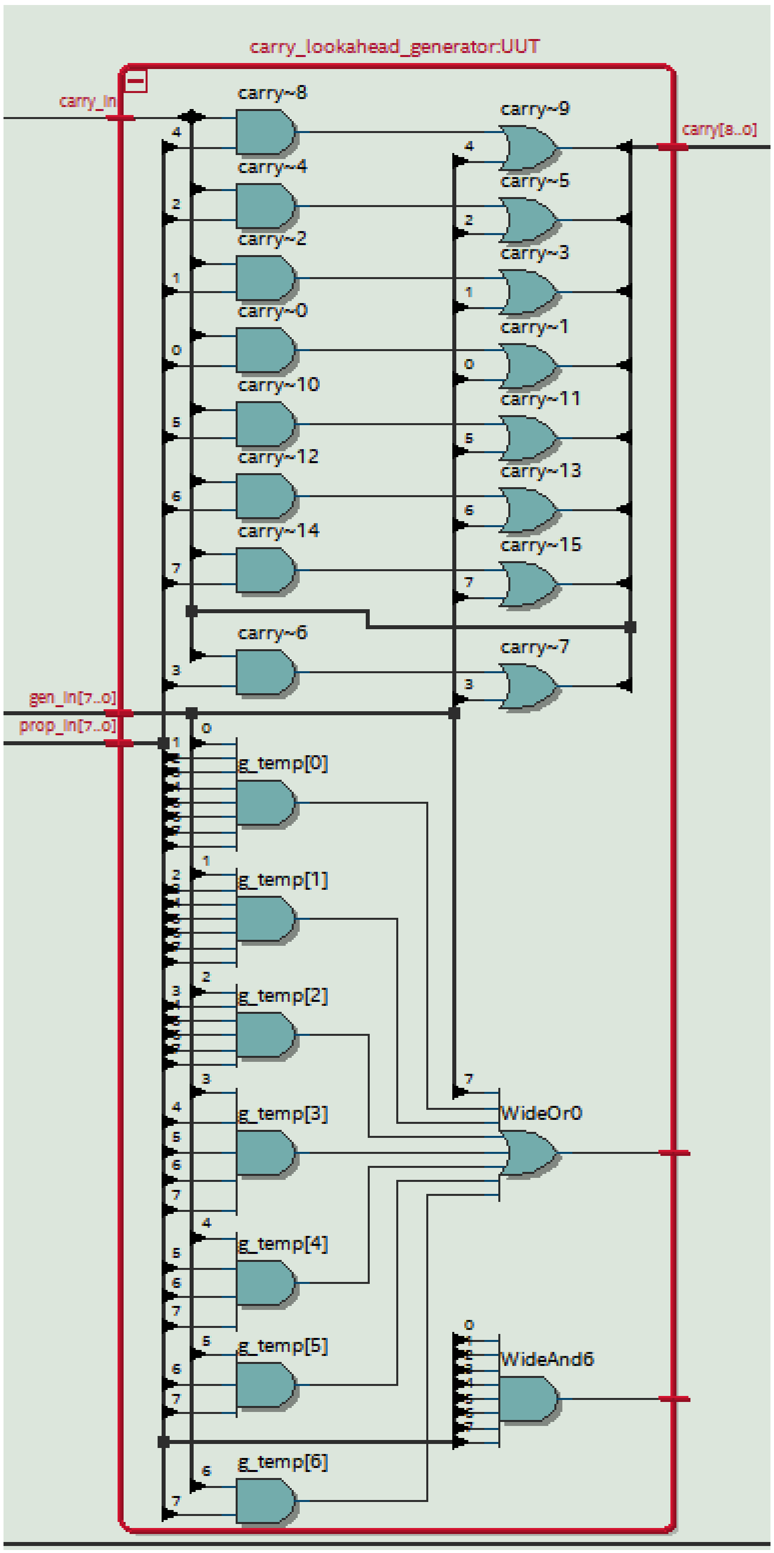
Сумматор с ускоренным переносом

Версия №2 по книге Панчула

```
`include "carry_lookahead_generator.sv"
module carry_lookahead_adder
    #(
        parameter WIDTH = 8
    )
    (
        input  logic                C_i,
        input  logic [WIDTH-1 : 0]  A,
        input  logic [WIDTH-1 : 0]  B,
        output logic [WIDTH-1 : 0]   S,
        output logic                 C_o
    );

    logic [ WIDTH   : 0 ] carry;
    logic [ WIDTH-1 : 0 ] g_wire, p_wire;

    carry_lookahead_generator
    #(
        .WIDTH(WIDTH)
    )
    UUT(
        .carry_in(C_i),
        .gen_in(g_wire),
        .prop_in(p_wire),
        .carry(carry),
        .group_gen(),
        .group_prop()
    );

    generate
        genvar i;
        for (i = 0; i <= WIDTH-1; ++i) begin : generateblock
            assign g_wire[i] = A[i] & B[i];
            assign p_wire[i] = A[i] ^ B[i];
            assign S[i]      = carry[i] ^ p_wire[i];
        end
    endgenerate

    assign C_o = carry[WIDTH];

endmodule
```

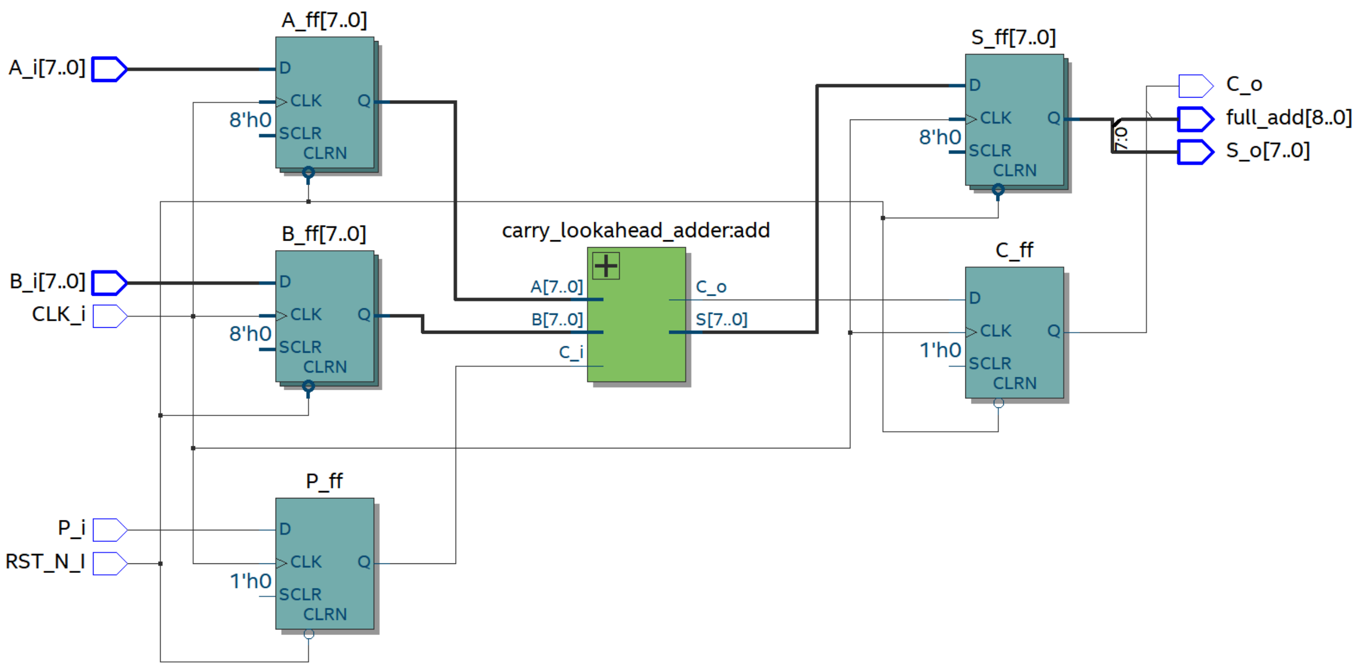


```
module carry_lookahead_generator
    #(
        parameter WIDTH = 8
    )
    (
        input  logic               carry_in,
        input  logic [WIDTH-1:0]  gen_in, prop_in,
        output logic [WIDTH:0  ]  carry,
        output logic              group_gen,
        output logic              group_prop
    );

    logic [WIDTH-1:0]  g_temp;
    logic [WIDTH-2:0]  p_temp;

    assign carry[0] = carry_in;

    generate
        genvar i;
        for ( i=0; i <= WIDTH-1; i++) begin
            assign carry[i+1] = gen_in[i] | prop_in[i] & carry[i];
            case (i)
                WIDTH-1 :
                 assign g_temp[i] = gen_in[i];
            default: begin
                 assign p_temp[i] = & prop_in[WIDTH-1 : i+1];
                 assign g_temp[i] = p_temp[i]& gen_in[i];
                end
            endcase
        end
    endgenerate

    assign group_gen = | g_temp;
    assign group_prop = & prop_in;

endmodule
```

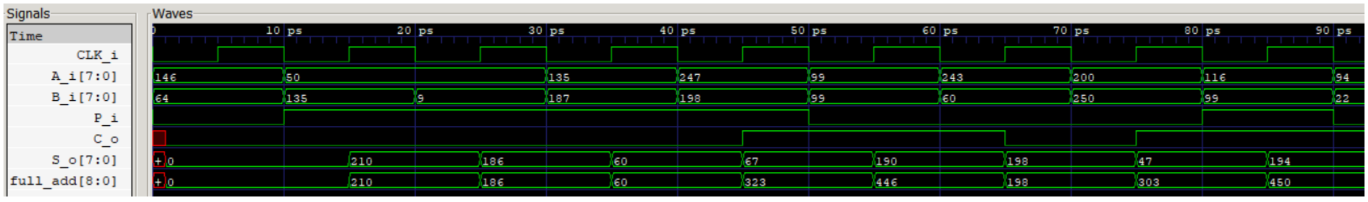


```
`include "carry_lookahead_adder.sv"
module top
    #(
        parameter WIDTH = 8
    )
    (
        input  logic [ WIDTH-1 : 0 ] A_i,
        input  logic [ WIDTH-1 : 0 ] B_i,
        input  logic                 P_i,
        input  logic                 RST_N_I,
        input  logic                 CLK_i,
        output logic [ WIDTH-1 : 0 ] S_o,
        output logic                 C_o,
        output logic [ WIDTH   : 0 ] full_add
    );

    logic [ WIDTH-1 : 0 ] A_ff;
    logic [ WIDTH-1 : 0 ] B_ff;
    logic                 P_ff;
    logic [ WIDTH-1 : 0 ] S_ff;
    logic                 C_ff;

    logic [ WIDTH-1 : 0 ] A_ff_w;
    logic [ WIDTH-1 : 0 ] B_ff_w;
    logic [ WIDTH-1 : 0 ] S_ff_w;
    logic                 C_ff_w;

    assign A_ff_w    = A_ff;
    assign B_ff_w    = B_ff;
    assign P_ff_w    = P_ff;

    assign C_o       = C_ff;
    assign S_o       = S_ff;

    always_ff @(posedge CLK_i or negedge RST_N_I) begin
        if(!RST_N_I) begin
            A_ff <= 0;
            B_ff <= 0;
            P_ff <= 0;
            S_ff <= 0;
            C_ff <= 0;
        end
        else begin
            A_ff <= A_i;
            B_ff <= B_i;
            P_ff <= P_i;
            S_ff <= S_ff_w;
            C_ff <= C_ff_w;
        end
    end

    carry_lookahead_adder
    #(
        .WIDTH(WIDTH)
    )
    add
    (
        .A(A_ff_w),
        .B(B_ff_w),
        .C_i(P_ff_w),
        .S(S_ff_w),
        .C_o(C_ff_w)
    );

    assign full_add = {C_o, S_o};
```

```
endmodule
```



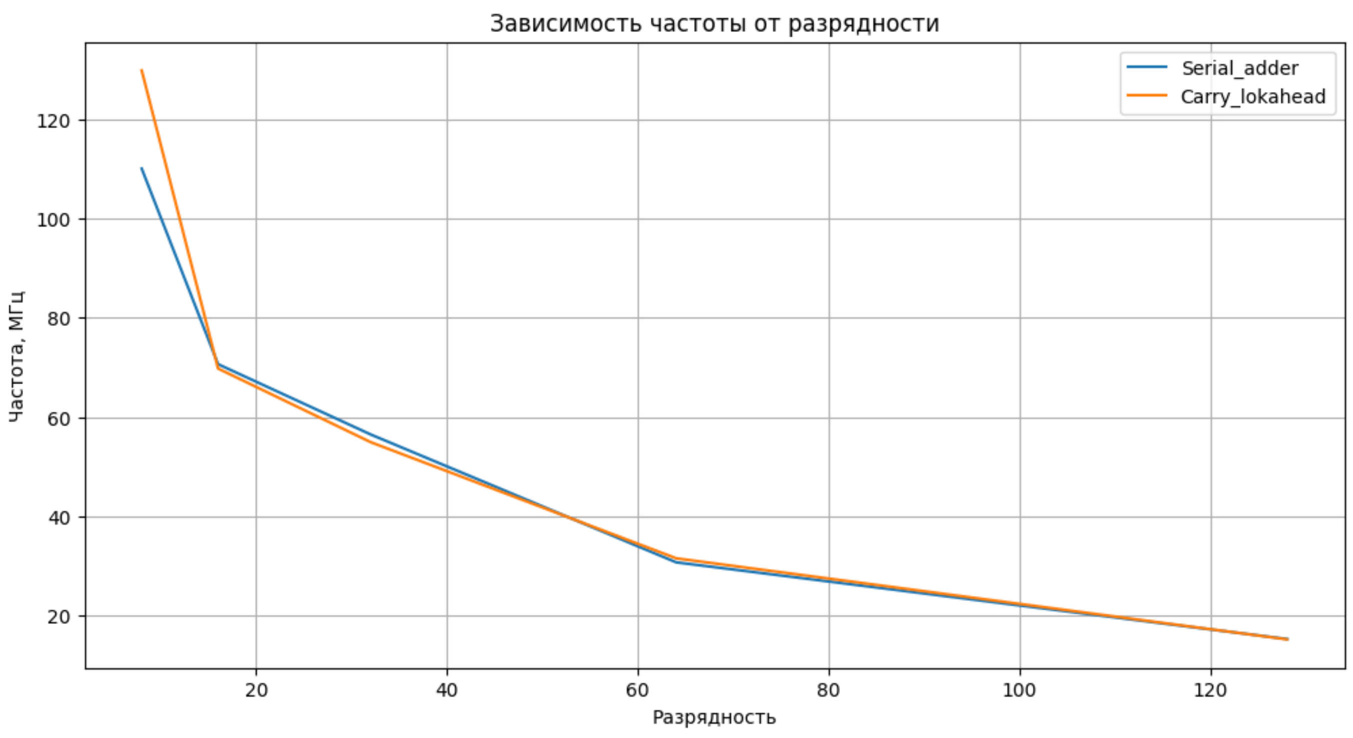
Waveform



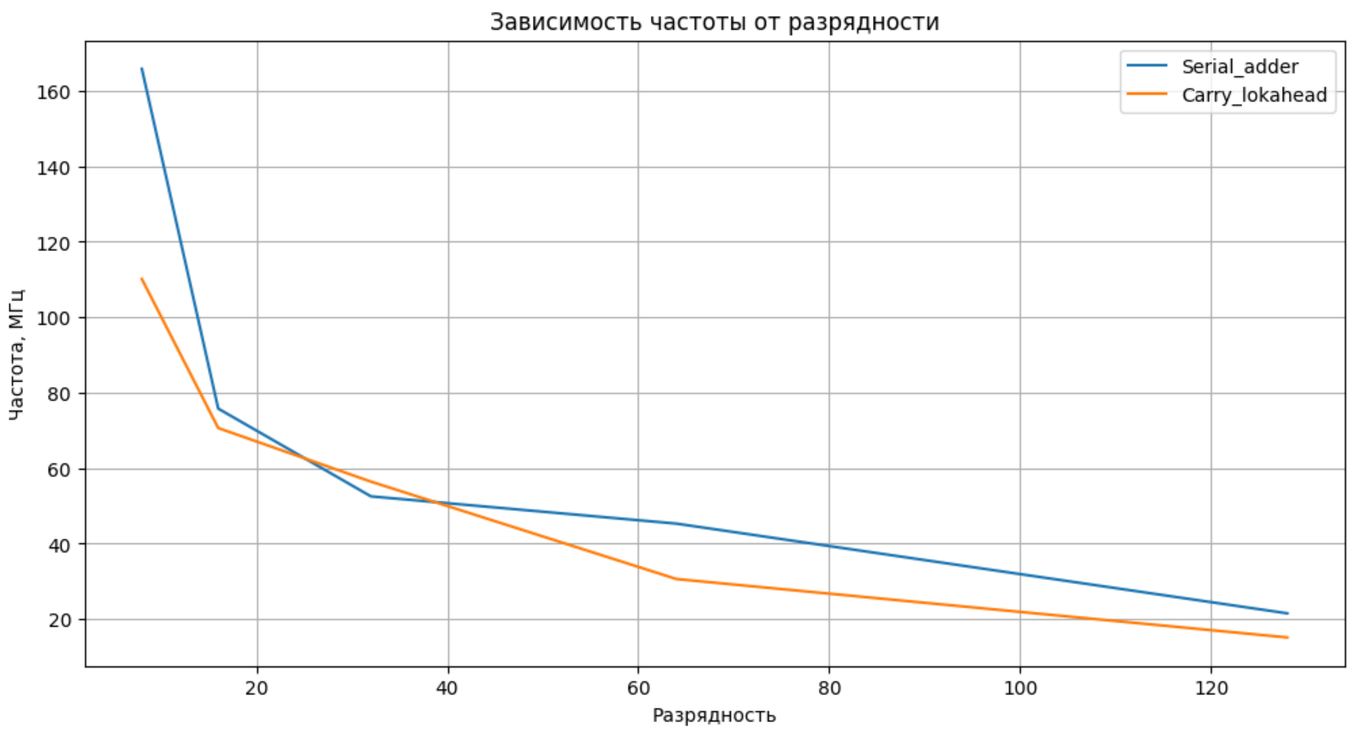
Зависимость частоты от разрядности

```
#Файл для временного анализа

create_clock -name {clock} -period 50MHz [get_ports {CLK_i}]

derive_clock_uncertainty

set_false_path -from [get_clocks {clock}] -to [get_ports {S_o[*]}]
set_false_path -from [get_clocks {clock}] -to [get_ports {C_o}]

set_false_path -from [get_clocks {clock}] -to [get_nets {A_i[*]}]
set_false_path -from [get_clocks {clock}] -to [get_nets {B_i[*]}]
set_false_path -from [get_clocks {clock}] -to [get_nets {P_i}]
```

Сравнение наших сумматоров



Сравнение сумматоров Панчула


Зависимость частоты от разрядности
Serial_adder
Carry_lokahead
Частота, МГц
Разрядность