Наборы операций

- NOP (пустая операция)
- RESET (сброс состояния)
- Операции перемещения данных
- Логические операции
- Операции сдвига
- Операции сравнения
- Операции перехода
- Целочисленные арифметические операции
- Арифметические операции с плавающей точкой
- Специализированные операции

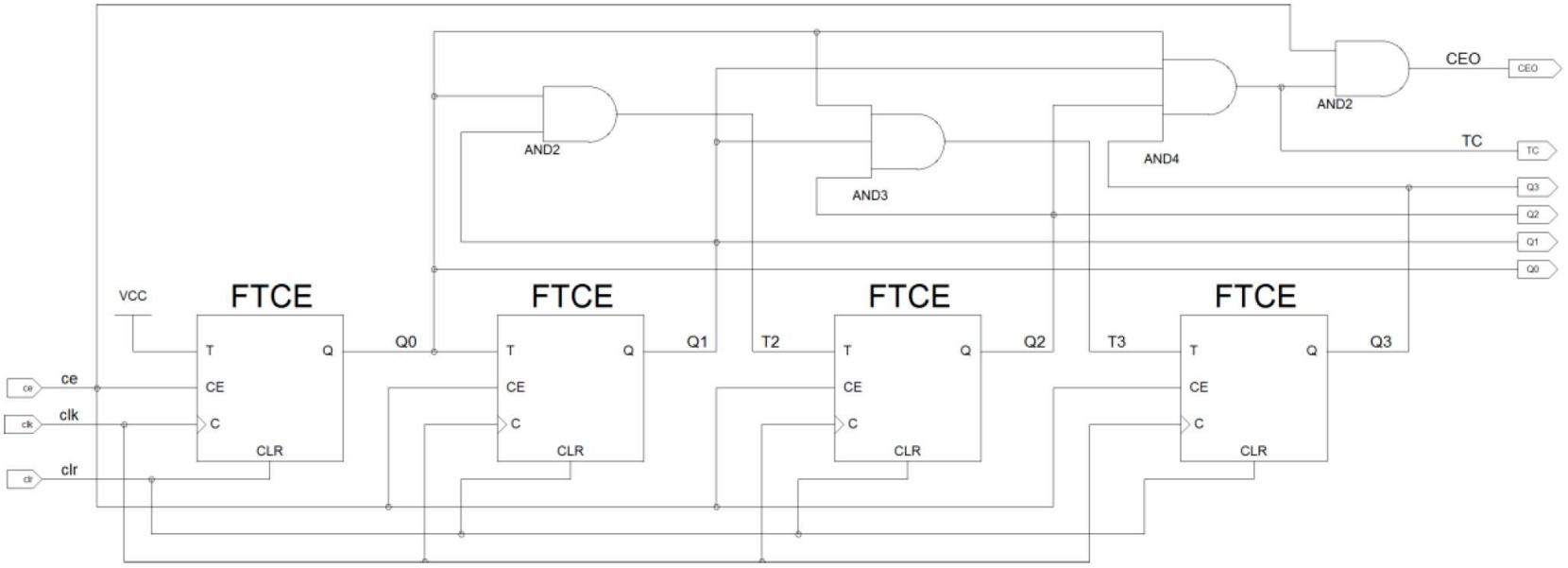
Оценки сложности

- Big O
- Little o
- Big Theta (Θ)
- Big Omega (Ω)
- Little Omega (ω)

- O(1)
- O(log n)
- O(n)
- O(n log n)
- $O(n^m)$
- O(exp n)
- O(n!)

Реализация логических элементов

Синхронный счетчик



Прямой код

$10_{10} = 01010_2$

$-10_{10} = 11010_2$

$0_{10} = 00000_2 = 10000_2$

Обратный код

$10_{10} = 01010_2$

$-10_{10} = 10101_2$

Дополнительный код

$$10_{10} = 01010_2$$

$$-10_{10} = 10101_2 + 0001_2 = 10110_2$$

$$15_{10} - 10_{10} = 01111_2 + 10110_2$$

$$= (1)\ 00101_2 = 5_{10}$$

Сложение двух двоичных чисел

0011 1011
1011 0110

1111 0001

Вычисление переноса

0011 1011

1011 0110

1111 0001

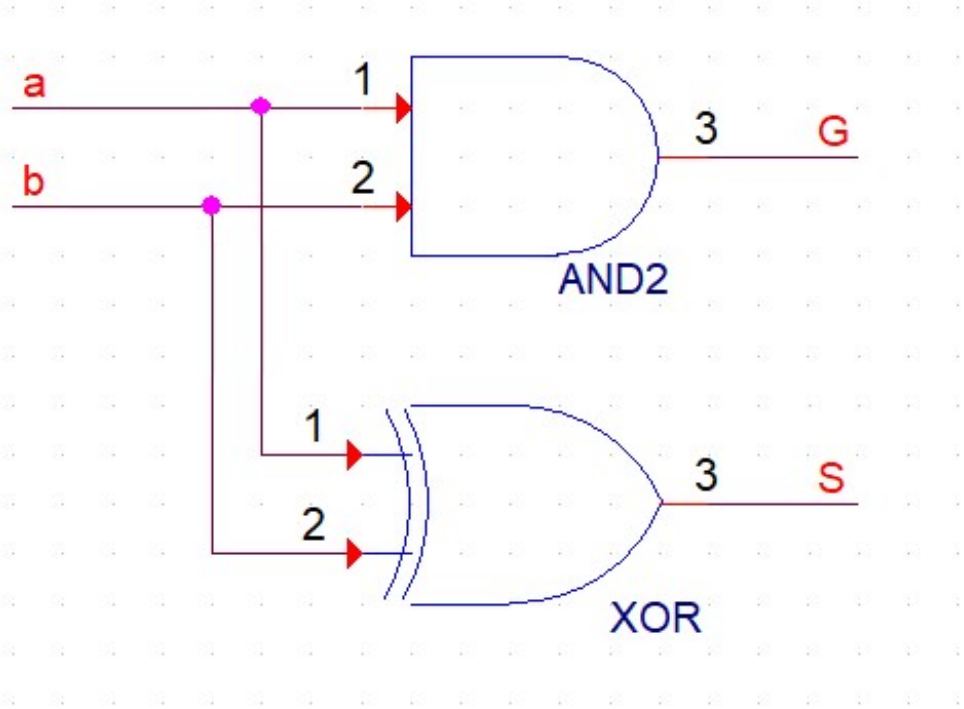
Однобитный полусумматор (неполный сумматор)

$G=ab$

$S=a \oplus b$

a	b	S	G
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Однобитный полусумматор (неполный сумматор)



Generate, Propagate и Carry

$G=ab$ – формирование переноса

$P=a \oplus b$ – распространение переноса

c – перенос между сумматорами

Группы:

$G_{3:0}=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0$

$P_{3:0}=P_3P_2P_1P_0$

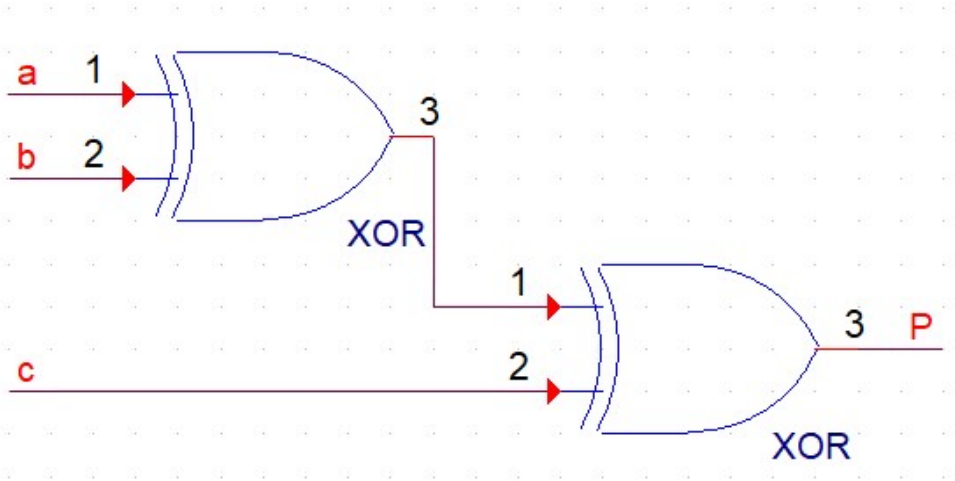
Однобитный полный сумматор

$$P = a \oplus b \oplus c$$

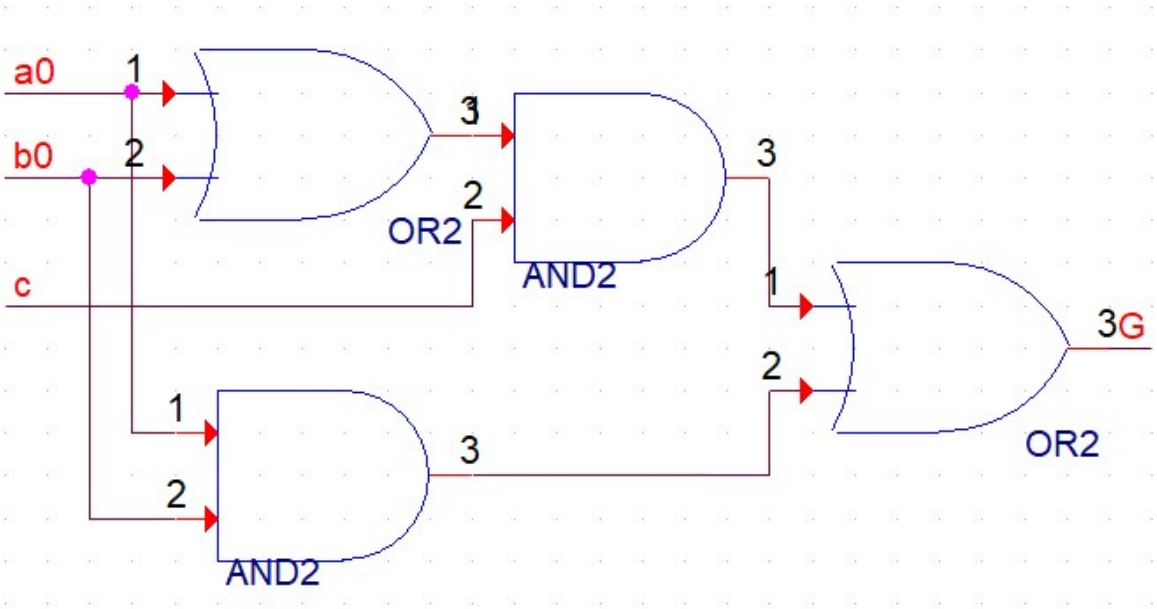
$$G = ab + (a+b)c$$

a	b	c	P	G
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

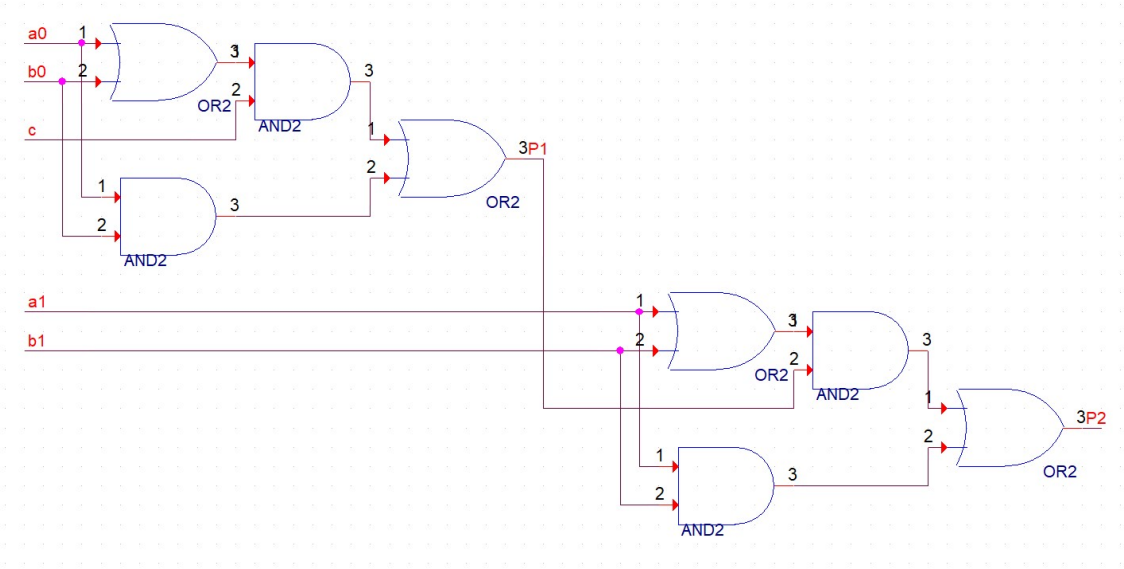
Однобитный полный сумматор



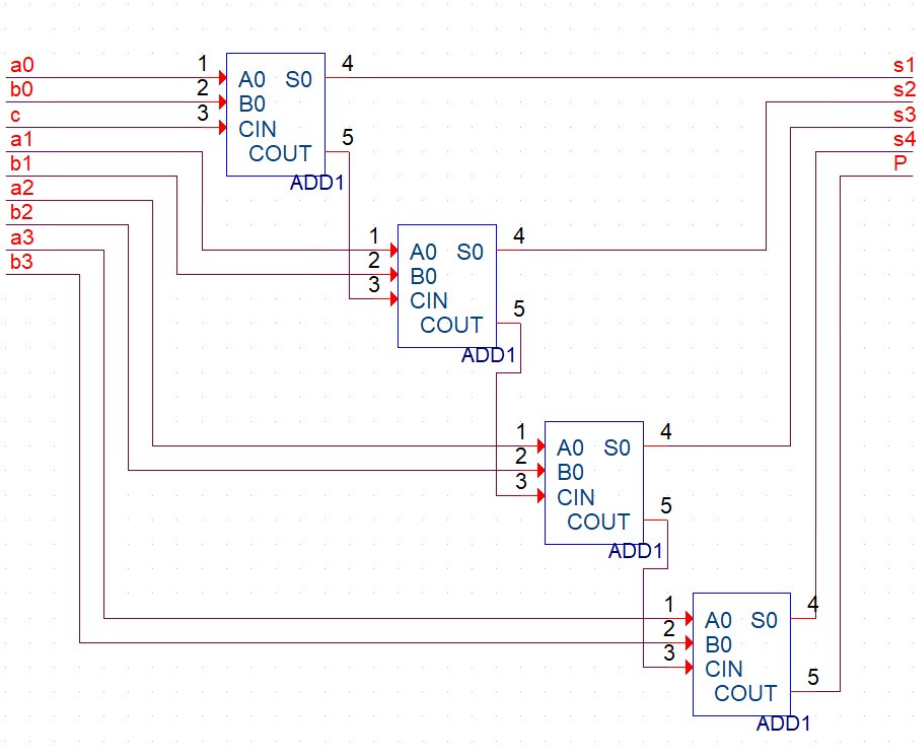
Однобитный полный сумматор



Каскадирование вычисления переноса



Каскадирование вычисления переноса

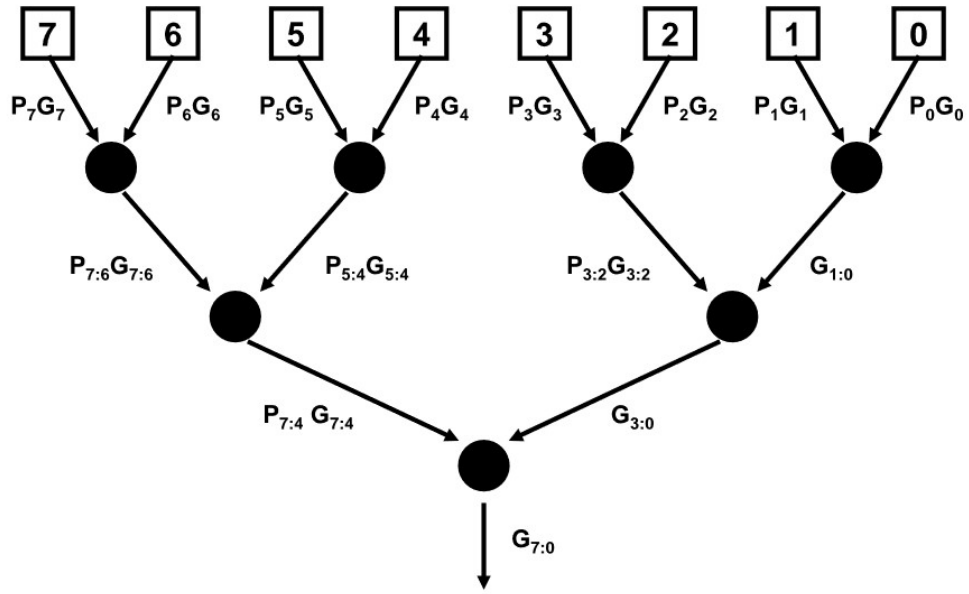


Одновременное вычисление переноса для нескольких разрядов

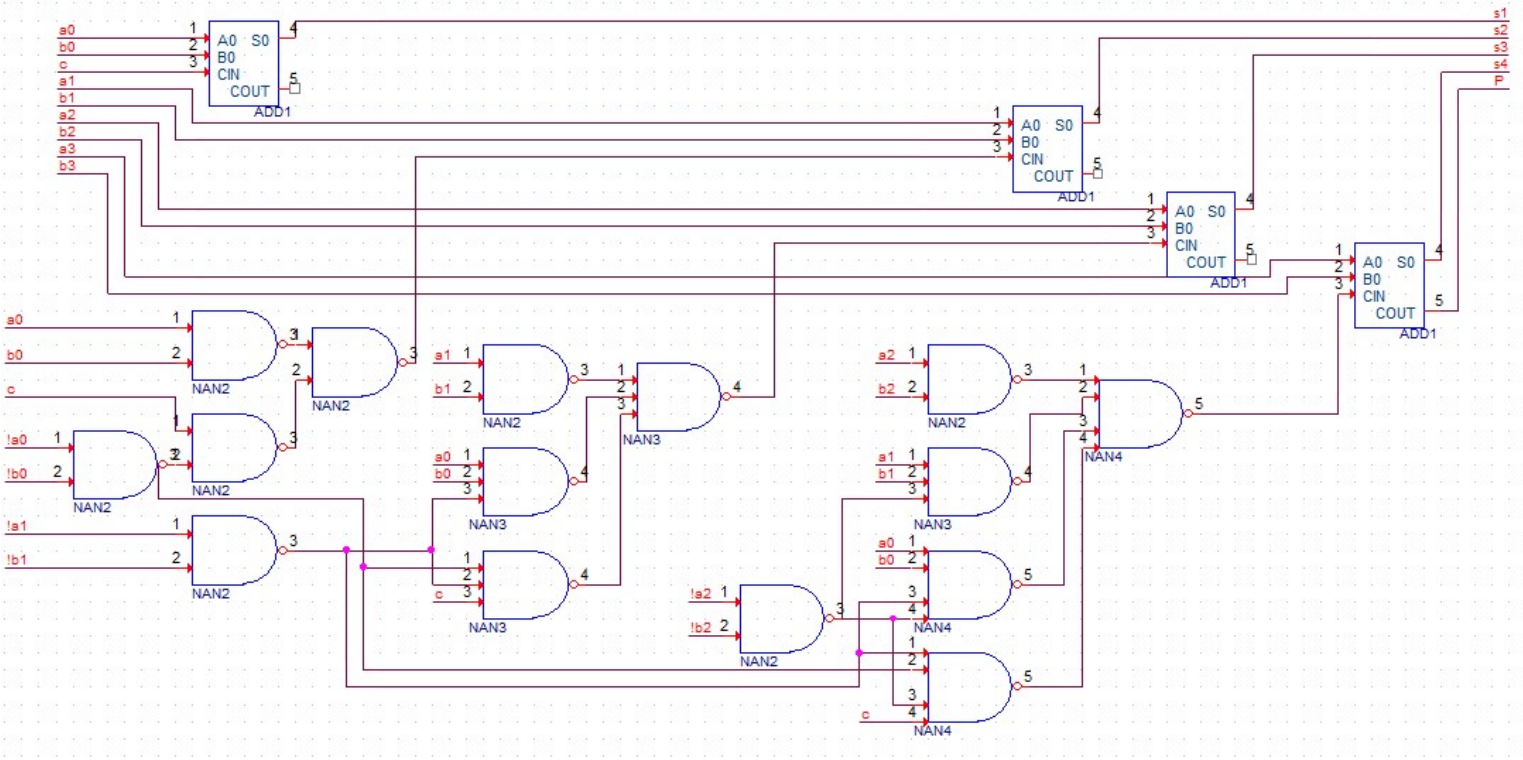
a0	b0	c	a1	b1	G
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	1

a0	b0	c	a1	b1	G
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	1
1	1	1	1	1	1
1	1	1	1	1	1

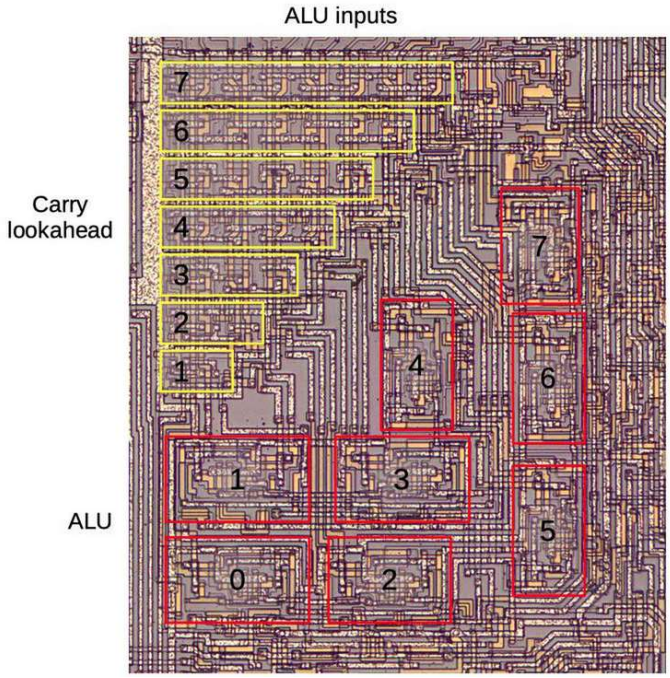
Вычисление переноса с помощью дерева логических элементов



Одновременное вычисление переноса для нескольких разрядов



Реализация сумматора в процессоре Intel 8008



Вычисление значения и переноса для нескольких разрядов

```
S0 = (A0 XOR B0) XOR Cin
S1 = (A1 XOR B1)
        XOR ((A0 AND B0)
        OR ((A0 XOR B0) AND Cin))
S2 = (A2 XOR B2)
        XOR ((A1 AND B1)
        OR ((A1 XOR B1) AND (A0 AND B0))
        OR ((A1 XOR B1) AND (A0 XOR B0) AND Cin))
S3 = (A3 XOR B3)
        XOR ((A2 AND B2)
        OR ((A2 XOR B2) AND (A1 AND B1))
        OR ((A2 XOR B2) AND (A1 XOR B1) AND (A0 AND B0))
        OR ((A2 XOR B2) AND (A1 XOR B1) AND (A0 XOR B0) AND Cin))
Cout = (A3 AND B3)
        OR ((A3 XOR B3) AND (A2 AND B2))
        OR ((A3 XOR B3) AND (A2 XOR B2) AND (A1 AND B1))
        OR ((A3 XOR B3) AND (A2 XOR B2) AND (A1 XOR B1) AND (A0 AND B0))
        OR ((A3 XOR B3) AND (A2 XOR B2) AND (A1 XOR B1) AND (A0 XOR B0) AND Cin)
```

Вычисление значения и переноса для нескольких разрядов

```
S0 = (A0 XOR B0) XOR Cin
S1 = (A1 XOR B1)
        XOR ((A0 AND B0)
        OR ((A0 XOR B0) AND Cin))
S2 = (A2 XOR B2)
        XOR ((A1 AND B1)
        OR ((A1 XOR B1) AND (A0 AND B0))
        OR ((A1 XOR B1) AND (A0 XOR B0) AND Cin))
S3 = (A3 XOR B3)
        XOR ((A2 AND B2)
        OR ((A2 XOR B2) AND (A1 AND B1))
        OR ((A2 XOR B2) AND (A1 XOR B1) AND (A0 AND B0))
        OR ((A2 XOR B2) AND (A1 XOR B1) AND (A0 XOR B0) AND Cin))
Cout = (A3 AND B3)
        OR ((A3 XOR B3) AND (A2 AND B2))
        OR ((A3 XOR B3) AND (A2 XOR B2) AND (A1 AND B1))
        OR ((A3 XOR B3) AND (A2 XOR B2) AND (A1 XOR B1) AND (A0 AND B0))
        OR ((A3 XOR B3) AND (A2 XOR B2) AND (A1 XOR B1) AND (A0 XOR B0) AND Cin)
```

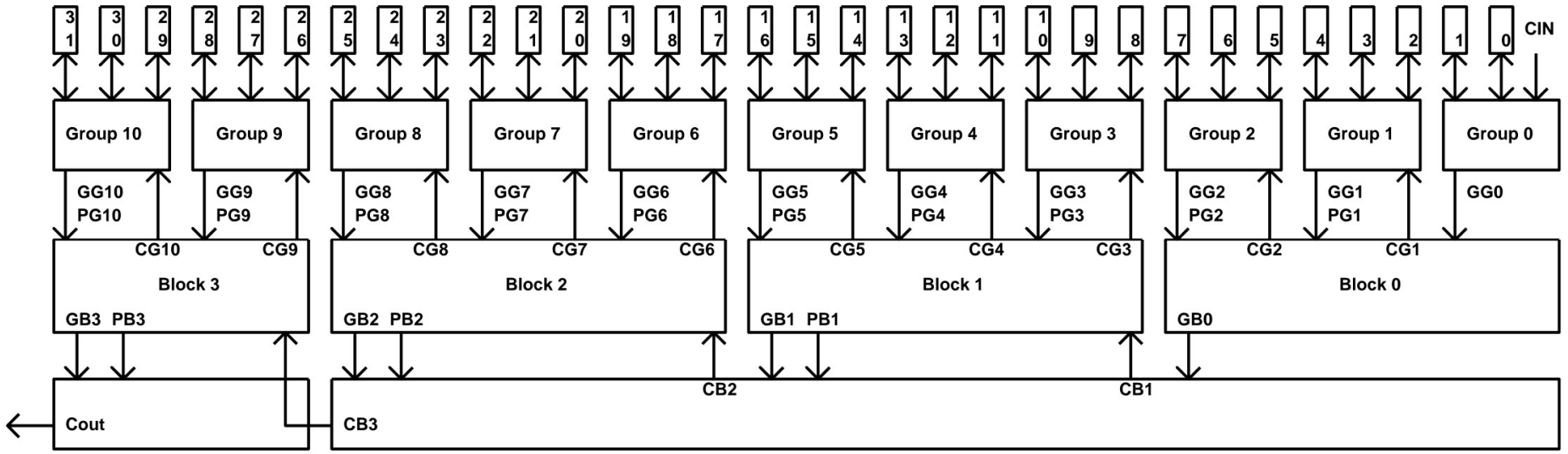
Общие методы формирования сумматора с древовидной структурой

Переменные древовидной структуры:

- Количество входов в узле (может быть переменным)
- Количество подключений выхода из узла
- Глубина дерева
- Основание системы счисления (группы)

- Возможность вычисления одновременно нескольких вариантов итогового значения, с выбором правильного результата в конце с учетом переносов
- Необходимость учитывать физику процесса

Сумматоры с ускоренным переносом



Вычисление переноса

Логика сумматора с предсказанием переноса

Для каждой пары бит формируется:

$g_i = a_i b_i$

$p_i = a_i + b_i$

Биты разбиваются на группы(например по 3 бита):

$GG_0=g_1+p_1(g_0+p_0Cin)$

$GG_1=g_4+p_4(g_3+p_3g_2)$

…

$PG_1=p_4p_3p_2$

$PG_2=p_7p_6p_5$

…

Из групп формируются группы второго уровня:

$GB_0=GG_2+PG_2(GG_1+PG_1GG_0)$

$GB_1=GG_5+PG_5(GG_4+PG_4GG_3)$

$GB_2=GG_8+PG_8(GG_7+PG_7GG_6)$

$GB_3=GG_{10}+PG_{10}GG_9$

$PB_1=PG_5PG_4PG_3$

$PB_2=PG_8PG_7PG_6$

$PB_3=PG_{10}PG_9$

Вычисление переноса

Логика сумматора с предсказанием переноса

Блоки формируют сигналы переноса блоков

$CB_1=GB_0$

$CB_2=GB_1+PB_1GB_0$

$CB_3=GB_2+PB_2(GB_1+PB_1GB_0)$

$CB_{out}=GB_3+PB_3CB_3$

Сигналы переноса блоков порождают сигналы переноса групп

…блок 2

$CG_3=CB_1$

$CG_4=GG_3+PG_3CB_1$

$CG_5=GG_5+PG_5(GG_4+PG_4CB_1)$

…

Наконец формируются побитовые переносы

…группа 1

$C_2=CG_1$

$C_3=g_2+p_2CG_1$

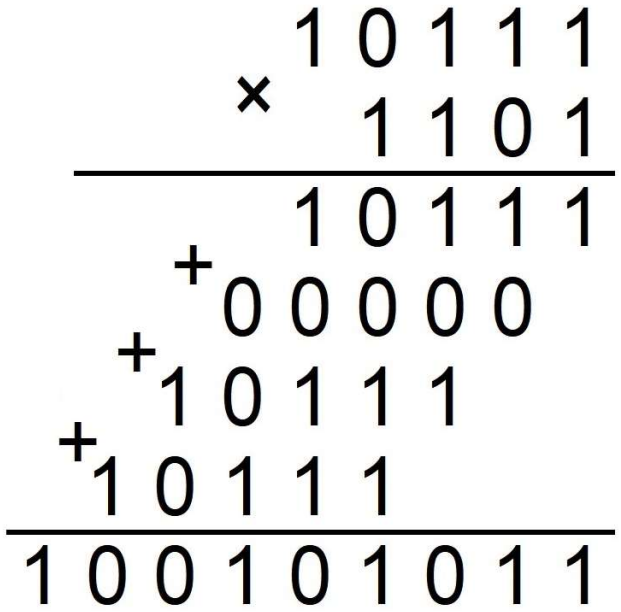
$C_4=g_3+p_3(g_2+p_2CG_1)$

…

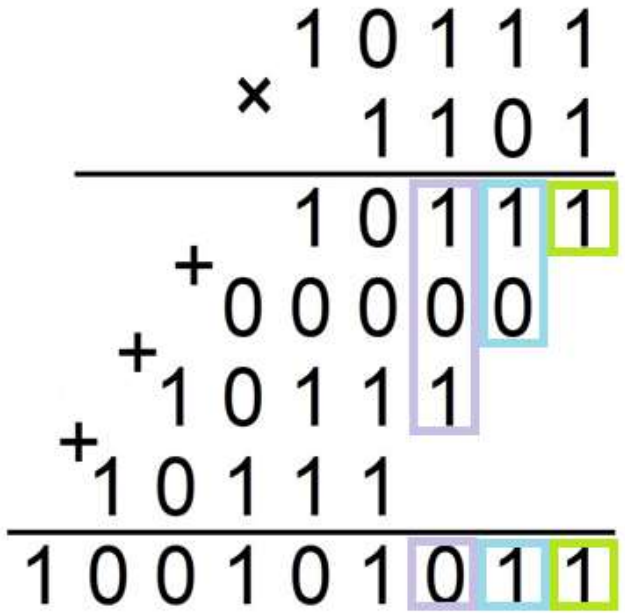
Итоговые биты суммы определяются следующим образом:

$S_i=a_i \oplus b_i \oplus c_i$

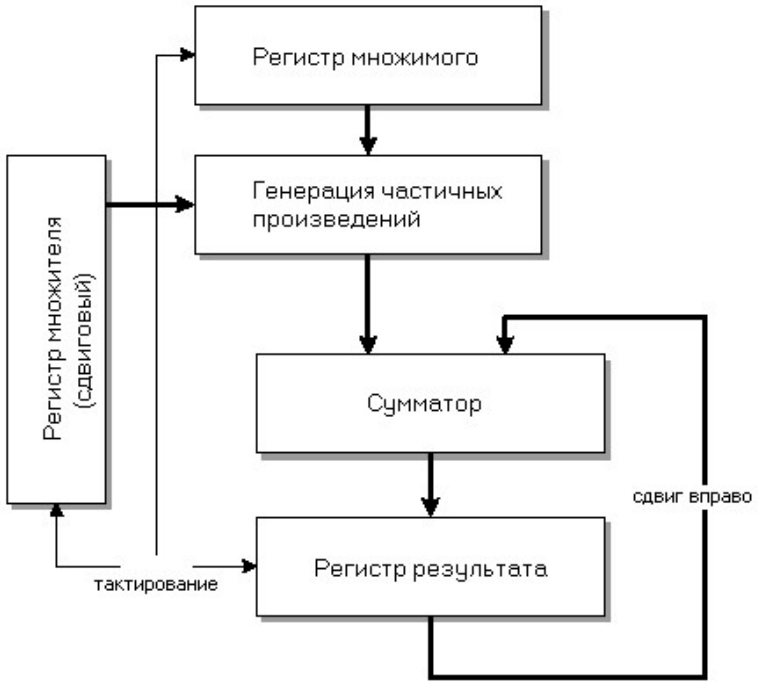
Двоичное умножение столбиком



Двоичное умножение столбиком



Итерационные умножители



Целочисленное умножение двоичных чисел с накоплением

```
`define m 24 //24 разряда необходимо для перемножения мантисс
`define n 24  //24 разряда необходимо для перемножения мантисс
module FIX_MULT(input [`m-1:0] A,       output reg[`m+`n-1:0] M=0, //Регистр множимого
                input [`n-1:0] B,       output wire end_mult,  //Конец умножения
                input clk,              output reg en_mult=0,  //Интервал умножения
                input st); //Импульс запуска умножителя
//M=(A*B)
reg [6:0]cb_tact=0 ;            //Счетчик тактов
reg [`m+`n-1:0] bf_A=0 ;        //Регистр сдвига сомножителя A
reg [`n-1:0] bf_B=0 ;           // Регистр сдвига сомножителя B
assign end_mult = (cb_tact==1) ; //Конец умножения

always @ (posedge clk) begin
en_mult <= st? 1 : end_mult? 0 : en_mult ; //Интервал умножения
cb_tact <= st? `n : en_mult? cb_tact-1 : cb_tact ; //Счет тактов
bf_A <= st? A : en_mult? bf_A<<1 : bf_A ; //Сдвиг A влево (умножение A на 2)
bf_B <= st? B : en_mult? bf_B>>1 : bf_B ; //Сдвиг B вправо для выделения младшего бита B
M <= st? 0 : (en_mult & (bf_B[0]==1))? M+bf_A : M ; //Накопление произведения
end
endmodule
```

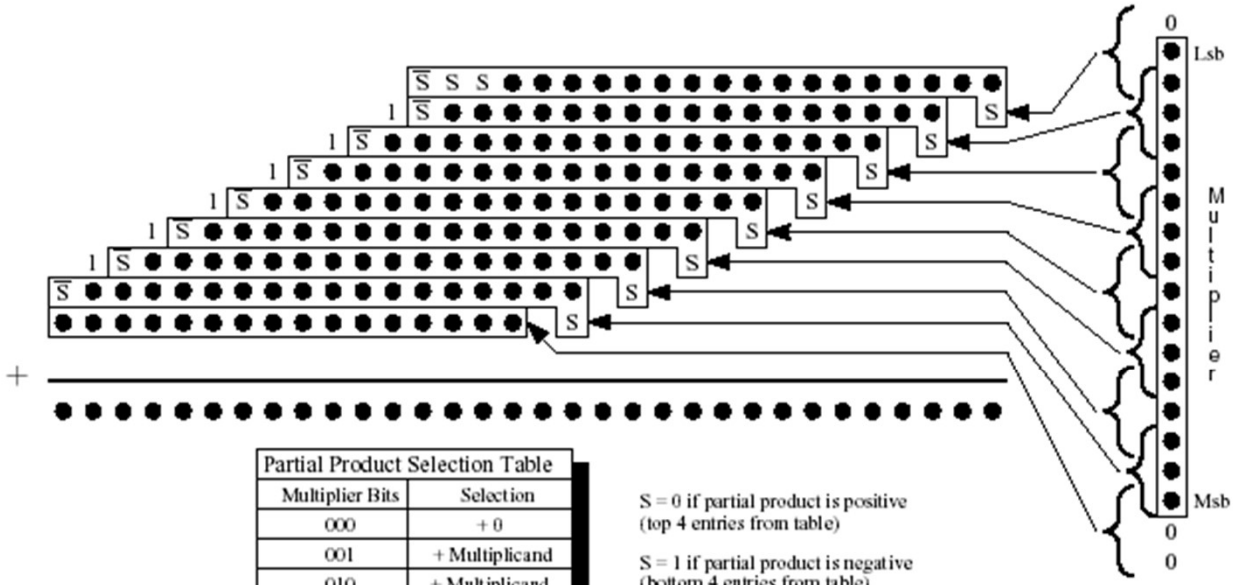
Умножение со знаком

```
`define m 18
module SMULT(
                input [`m-1:0] A,       output reg [2*`m-1:0] M=0, //Произведение
                input [`m-1:0] B,
                input clk,
                input st);

wire s_A = A[`m-1];             //Знак числа A
wire s_B = B[`m-1];             //Знак числа B
wire s_M = s_A ^ s_B ;          //Знак произведения M
wire [`m-2:0]mod_A= s_A? -A[`m-2:0] : A[`m-2:0]; //Модуль числа A
wire [`m-2:0]mod_B= s_B? -B[`m-2:0] : B[`m-2:0]; //Модуль числа B
wire [2*`m-3:0]mod_M =mod_A * mod_B; // Произведение модулей

always @ (posedge clk) begin
M <= st? 0 : (s_M & ce)? -mod_M : ce? mod_M : M; //Произведение
end
endmodule
```

Умножитель Бута

- $2^m+2^{m-1}+\ldots+2^k=2^{m+1}-2^k$
- m и k — номера крайних разрядов в группе из последовательных единиц
- 011110 =32-2=30

Умножитель Бута модифицированный алгоритм



Partial Product Selection Table	
Multiplier Bits	Selection
000	+ 0
001	+ Multiplicand
010	+ Multiplicand
011	+ 2 x Multiplicand
100	-2 x Multiplicand
101	- Multiplicand
110	- Multiplicand
111	- 0

S = 0 if partial product is positive
(top 4 entries from table)

S = 1 if partial product is negative
(bottom 4 entries from table)

Алгоритм Лемана

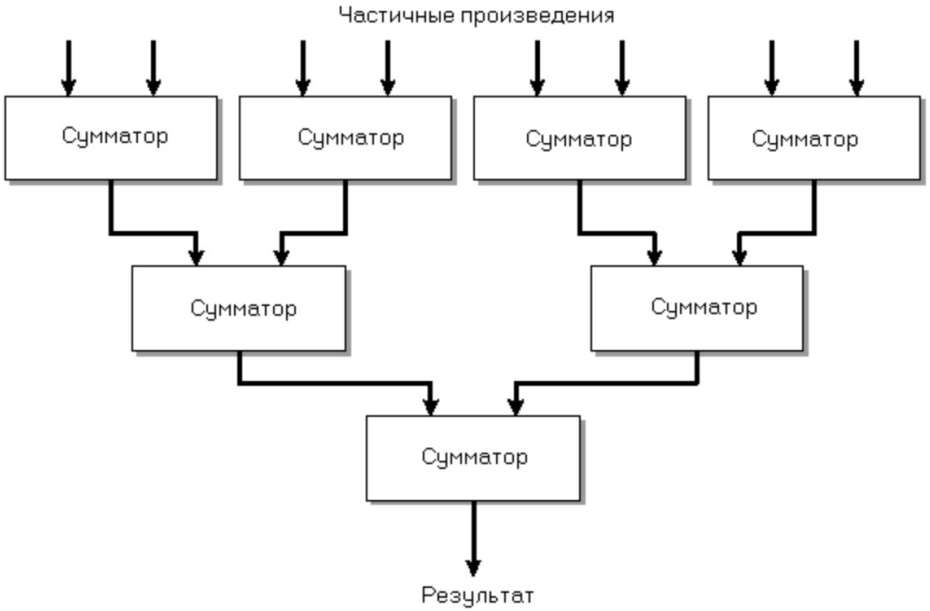
- если две группы нулей разделены единицей, стоящей в k-й позиции, то вместо вычитания в k-й позиции и сложения в (к + 1)-й позиции достаточно выполнить только сложение в k-й позиции;
- если две группы единиц разделены нулем, стоящим в k-й позиции, то вместо сложения в k-й позиции и вычитания в (k + 1)-й позиции достаточно выполнить только вычитание в k-й позиции.

Матричные умножители

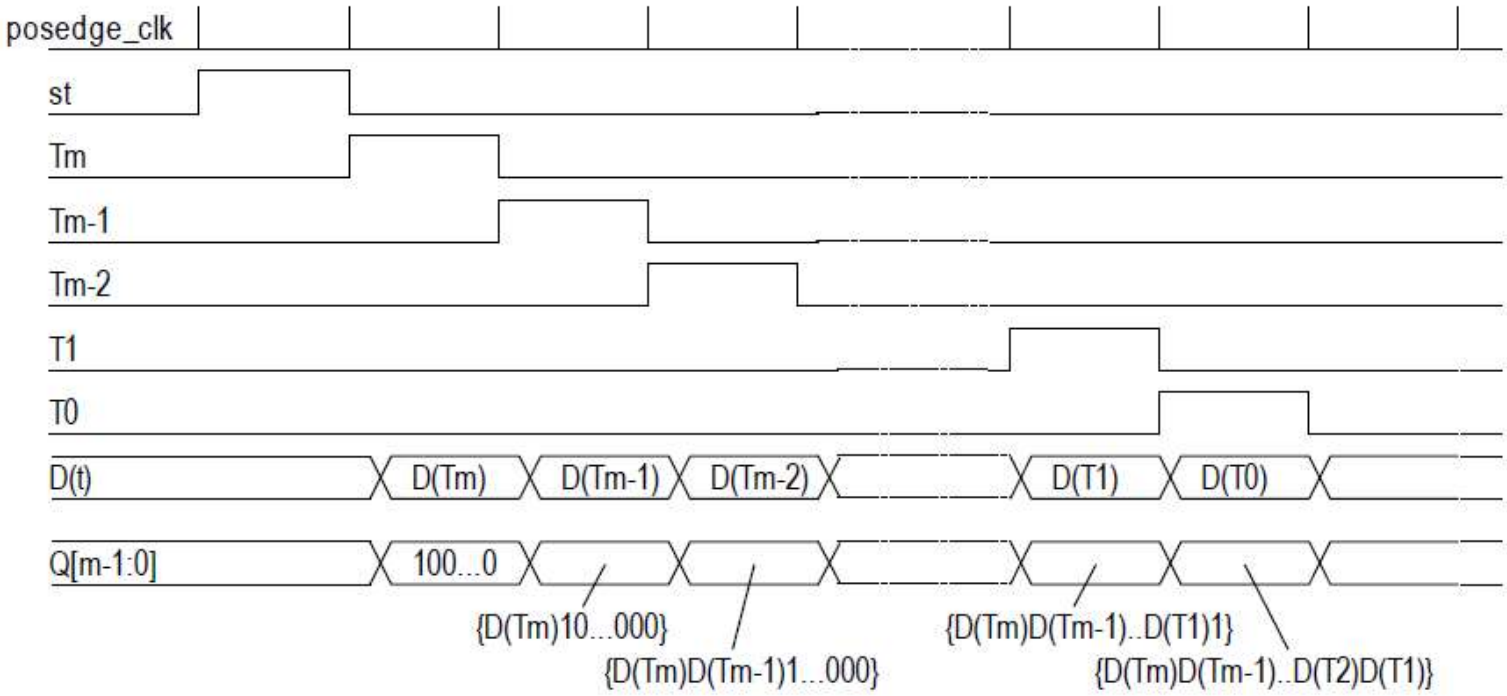
$$P = A \times B = \left(\sum_{i=0}^{n-1} a_i \times 2^i\right) \times \left(\sum_{j=0}^{n-1} b_j \times 2^j\right) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \times 2^{i+j}.$$

Умножитель n x n содержит n^2 схем «И», ПС и $(n^2 - 2n)$ СМ

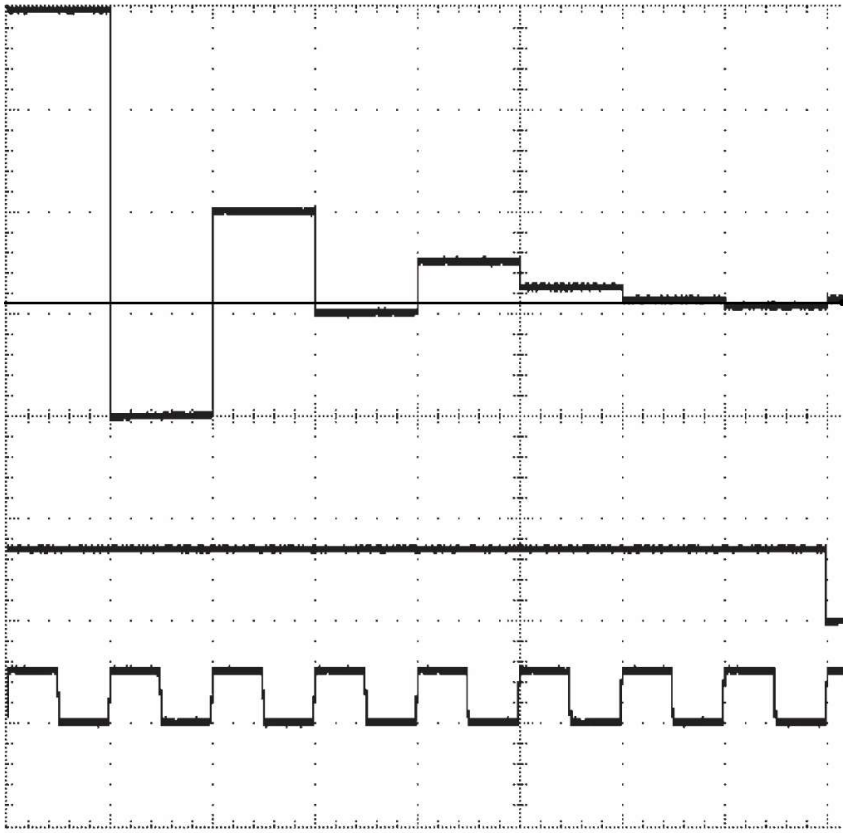
Параллельные умножители



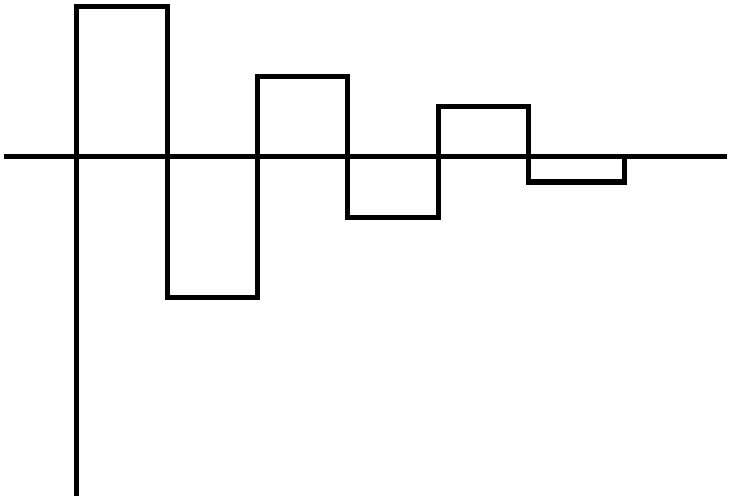
Последовательное приближение



Последовательное приближение



Метод вычисления обратных функций последовательным приближением



$\sqrt{x}$ =Y

Y^2=X

Пусть X=15

1000^2= 64 > X

0100^2=16 > X

0010^2= 4 < X

0011^2=9 < X

0011.1^2=12.25 <X

0011.11^2=14.0625 <X

0011.111^2=15.015625 >X

0011.1101^2=14.53515625 <X

0011.11011^2=14.8225 <X

0011.110111^2=14.9672265625 <X

…

Последовательное приближение

```
`define m 16
module SAR_SR (    input st,        output reg en=0, //Интервал приближения
                   input D,         output reg [`m-1:0] Q=0, // Выходной регистр
                   input clk,       output wire ok); //Конец приближения

reg[`m:0] T=0 ;          //Регистр сдвига
assign ok = T[0] ;       //Конец последовательного приближения
integer i ;              //Индекс цикла for
always @ (posedge clk)  begin
T <= st? 1<<`m : en? T>>1 : T ; //Сдвиг импульсов T вправо
for (i=`m-1 ; i >= 0; i=i-1) // Цикл for
Q[i] <= T[i+1]? 1 : T[i]? D : Q[i] ; //Загрузка очередного бита выходного регистра Q
en <= st? 1 : (T[0] & en)? 0 : en ;     //Интервал  последовательного приближения
end
endmodule
```

Вычисление квадратного корня методом последовательного приближения

```
module SQRT_BL (  input [31:0] X,        output wire[15:0]Q,
                  input st,              output wire en,
                  input clk,             output wire ok,
                                         output wire DI);

wire [31:0]M=Q*Q ; // Возведение Q в квадрат
assign DI =(M<=X); //Входной бит регистра последовательного приближения
SAR_SR  DD1 (     .st(st),        .Q(Q), // Выход регистра
                  .D(DI),         .en(en), //Интервал приближения
                  .clk(clk),      .ok(ok)); //Конец приближения

endmodule
```

Вычисление кубического корня методом последовательного приближения

```
module CUBRT_BL (input [47:0] X,        output wire[15:0]Q, // Выход результата
                 input st,              output wire en, //Интервал приближения
                 input clk,             output wire ok, //Конец приближения
                                        output wire DI); // Входной бит регистра
wire [47:0]M=Q*Q*Q ; // Возведение Q в куб
assign DI =(M<=X); //Входной бит регистра последовательного приближения
SAR_SR  DD1 (    .st(st),          .Q(Q), // Выход регистра
                 .D(DI),           .en(en), //Интервал приближения
                 .clk(clk),        .ok(ok)); //Конец приближения
endmodule
```

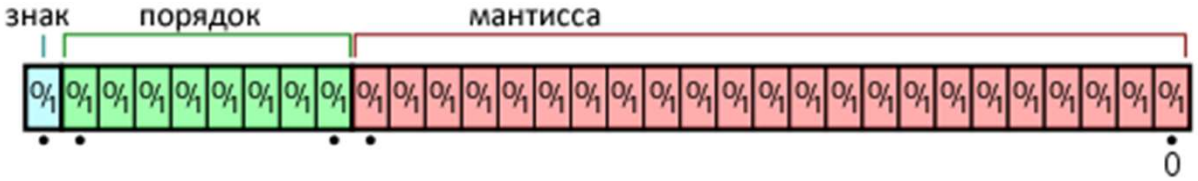
Числа с фиксированной точкой

0000 0000 0000 0000 . 0000 0000 0000 0000

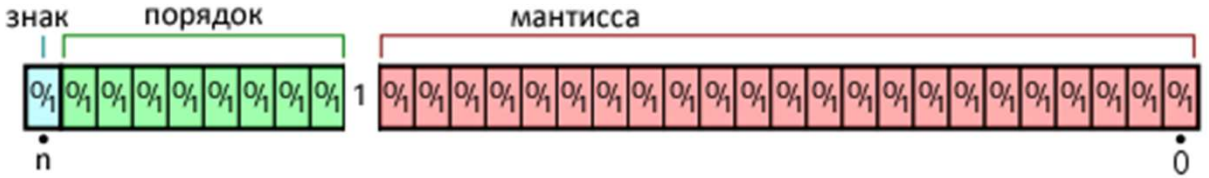
Максимальное целое 2^16

Числа с плавающей точкой

$(-1)^s \times M \times 2^E$



Нормализованный вид:
$(-1)^s \times 1.M \times 2^E$
+Денормализованный вид:
$(-1)^s \times 0.M \times 2^{Emin}$, если $E=E_{min}-1$



IEE754 (0: m=0, $e=e_{min}-1$; бесконечность: m=0, $e=e_{max}+1$; NaN: m – не 0, $e=e_{max}+1$)

s=0 (положительное)
$E=01111100_2-127_{10} = -3$
$M = 1.01_2$ (первая единица не явная)
$F = 1.01_2e-3 = 2^{-3}+2^{-5} =$
0,125 + 0,03125 = 0,15625

Числа с плавающей точкой

Округление в IEEE754:
0.5 округляется до четного: 1.5=2, 2.5=2

Проблема ассоциативности для чисел с плавающей точкой:
$(10^{20}+1)-10^{20}=0 \neq (10^{20}-10^{20})+1=1$

Сравнение специальных чисел IEEE754:

x	y	x < y? x: y	x <= y? x: y	x > y? y: x	x >= y? y: x
+0	-0	-0	+0	+0	-0
NaN	1	1	1	NaN	NaN

Умножение чисел с плавающей точкой

```
module MULT_FLOAT (output wire[47:0]MAxMB, // Произведение мантисс
input [31:0] A, output wire[31:0]M, //Произведение
input [31:0] B, output reg ok=0, //Конец умножения
input st,
input clk );
parameter E0 = 127;//E0=8'b01111111=8'h7F(смещение экспоненты)
wire ZA = (A==0) ; //Сомножитель A==0 ;
wire ZB = (B==0) ; //Сомножитель B==0 ;
wire [7:0]EA=A[30:23]- E0; //Экспонента A
wire [23:0]MA={1'b1,A[22:0]}; //Мантисса A
wire [7:0]EB=B[30:23]- E0; // Экспонента B
wire [23:0]MB={1'b1,B[22:0]}; // Мантисса B
assign MAxMB = MA*MB ; //Перемножение мантисс
wire SA=A[31] ; wire SB=B[31] ; //Знаки сомножителей
wire [7:0]EM= MAxMB[47]? (EA+EB)+ E0+1 : EA+EB+E0; //Экспонента M
wire [22:0]MM = MAxMB[47]? MAxMB>>22 : MAxMB>>23; // Модификация мантиссы M
wire SM = SA^SB ; // Знак произведения
assign M = (ZA | ZB)? 0 : {SM,EM,MM} ; //Сборка произведения
always @ (posedge clk) begin
ok <= st ; // Конец преобразования
end
endmodule
```

Сложение чисел с плавающей точкой

```
module ADD_FLOAT(
input st, output wire [31:0] SUMM, //Сумма чисел A и B
input [31:0] A, output wire [7:0] Exp_A, // Экспонента числа A
input [31:0] B, output wire [7:0] Exp_B, // Экспонента числа B
input clk, output wire [23:0] M_A, //Мантисса числа A
output wire [23:0] M_B, // Мантисса числа B
output wire [7:0] d_Exp, //Разность экспонент
output wire d_S, //Разность знаков слагаемых
output reg en_sh_M=0, //Разрешение сдвига мантиссы меньшего числа
output reg [7:0]cb_sh_M=0, //Счетчик сдвигов меньшей мантиссы
output wire ok_sh_M, // Конец сдвига мантиссы меньшего числа
output reg [23:0]sr_MA=0, //Регистр сдвига мантиссы A
output reg [23:0]sr_MB=0, // Регистр сдвига мантиссы B
output wire [24:0]Temp_SUMM, //Временная сумма/разность
output reg en_sh_TS=0, //Разрешение сдвига разности
output reg [23:0]sr_TS=0, //Регистр сдвига временной суммы/разности
output reg [6:0]cb_sh_TS=0, //Счетчик сдвига суммы/разности
output wire ok_sh_TS, //Конец сдвига разности
output reg ok_SUMM=0 ); //Конец суммирования
assign Exp_A=A[30:23]; assign M_A={1'b1,A[22:0]}; //Экспонента и мантисса числа A
assign Exp_B=B[30:23]; assign M_B={1'b1,B[22:0]}; // Экспонента и мантисса числа B
wire SA=A[31] ; wire SB=B[31] ; //Знаки чисел A и B
wire A_big_B = (A[30:0]>B[30:0]) ; //Модуль A больше модуля B
wire A_equ_B = (A[30:0]==B[30:0]) ; // Модуль A равен модулю B
assign d_S = SA^SB ; //Разность знаков слагаемых
assign d_Exp=A_big_B? Exp_A-Exp_B : Exp_B-Exp_A; // Разность экспонент
wire start= st & !(d_Exp==0); //Нет старта сдвига, если разность экспонент равна нулю
assign Temp_SUMM = !d_S? (sr_MA+sr_MB): A_big_B? sr_MA-sr_MB : sr_MB-sr_MA;
wire CO=Temp_SUMM[24] ; //Переполнение суммы мантисс
wire [7:0]max_Exp = A_big_B? Exp_A : Exp_B ; //Выбор максимальной экспоненты
wire S_SUMM = A_big_B? SA : SB ; //Знак суммы равен знаку большего числа
wire [7:0]Exp_SUMM = CO? max_Exp+1 : d_S? max_Exp-cb_sh_TS : max_Exp ;
wire [22:0]M_SUMM = CO? Temp_SUMM[23:1] : sr_TS[22:0] ;
assign SUMM = (A_equ_B & d_S)? 0 : //S=0 при равных модулях и разных знаках
(A==0)? B : //Если A==0, то SUMM=B
(B==0)? A : //Если B==0, то SUMM=A
{S_SUMM,Exp_SUMM,M_SUMM} ; //Сумма
reg ten_sh_M =0;
reg tst=0 ;
reg ten_sh_TS=0 ;
assign ok_sh_M = (d_Exp==0)? tst : !en_sh_M & ten_sh_M ; /* Конец сдвига мантиссы меньшего числа*/
assign ok_sh_TS= !en_sh_TS & ten_sh_TS ; // Конец сдвига временной суммы
always @ (posedge clk) begin
ten_sh_M <= en_sh_M ; tst <= st ; ten_sh_TS <= en_sh_TS ; //Задержка на такт
en_sh_M <= start? 1 : (cb_sh_M==1)? 0 : en_sh_M ; //Интервал сдвига мантисс
cb_sh_M <= start? d_Exp : en_sh_M? cb_sh_M-1 : cb_sh_M ; //Счет сдвигов мантиссы
sr_MA <= st? M_A : (en_sh_M & !A_big_B)? sr_MA>>1 : sr_MA ; //Сдвиг мантиссы A
sr_MB <= st? M_B : (en_sh_M & A_big_B)? sr_MB>>1 : sr_MB ; // Сдвиг мантиссы B
en_sh_TS <= (ok_sh_M & d_S & !(Temp_SUMM[23]))? 1 : (sr_TS[22])? 0 : en_sh_TS ; 6

cb_sh_TS <= ok_sh_M? 0 : en_sh_TS? cb_sh_TS+1 : cb_sh_TS ; // Счет сдвигов
sr_TS <= ok_sh_M? Temp_SUMM[23:0] : en_sh_TS? sr_TS<<1 : sr_TS ; //Сдвиг разности
ok_SUMM <= (!d_S | (d_S & (Temp_SUMM[23])))? ok_sh_M : ok_sh_TS ;
end
endmodule
```