論理回路入門（８）

２進数の掛け算とウォレスツリー

李　亜民

2023 年 11 月 14 日 (火)

2 進数の掛け算

ポイント

- 符号なし数の掛け算（乗算）
- 符号なし乗算器回路
- 符号なし CSA を使用する乗算器回路
- 符号なしのウォレスツリー乗算器
- 2 の補数で表現する数の掛け算
- 2 の補数のウォレスツリー乗算器

2 進数掛け算の方式と掛け算命令

1. 直列掛け算
   - 筆算と同じ、各桁を順次計算。
2. 並列掛け算
   - 組合わせ論理回路を利用、各桁を同時に計算。
3. ROM 利用の掛け算
   - ROM にデータを用意、演算時に読み出し。

CPU の掛け算命令の例（32 ビット × 32 ビット）

- mulh ・・・・・64 ビット積の上位 32 ビットの計算
- mul ・・・・・・・64 ビット積の下位 32 ビットの計算

符号なし数の掛け算

1 1 1 0	かけられる数	(14_{10})
× 1 0 1 0	かける数	(10_{10})

0 0 0 0	桁に応じて
1 1 1 0	ずらす
0 0 0 0	
+ 1 1 1 0	
1 0 0 0 1 1 0 0	積 (140_{10})

2 倍のビット数が必要

符号なし数の掛け算

$$
\begin{array}{rcccc}
 & a_3 & a_2 & a_1 & a_0 \\
\times & b_3 & b_2 & b_1 & b_0 \\
\hline
 & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 \\
 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 \\
+ & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

かけられる数

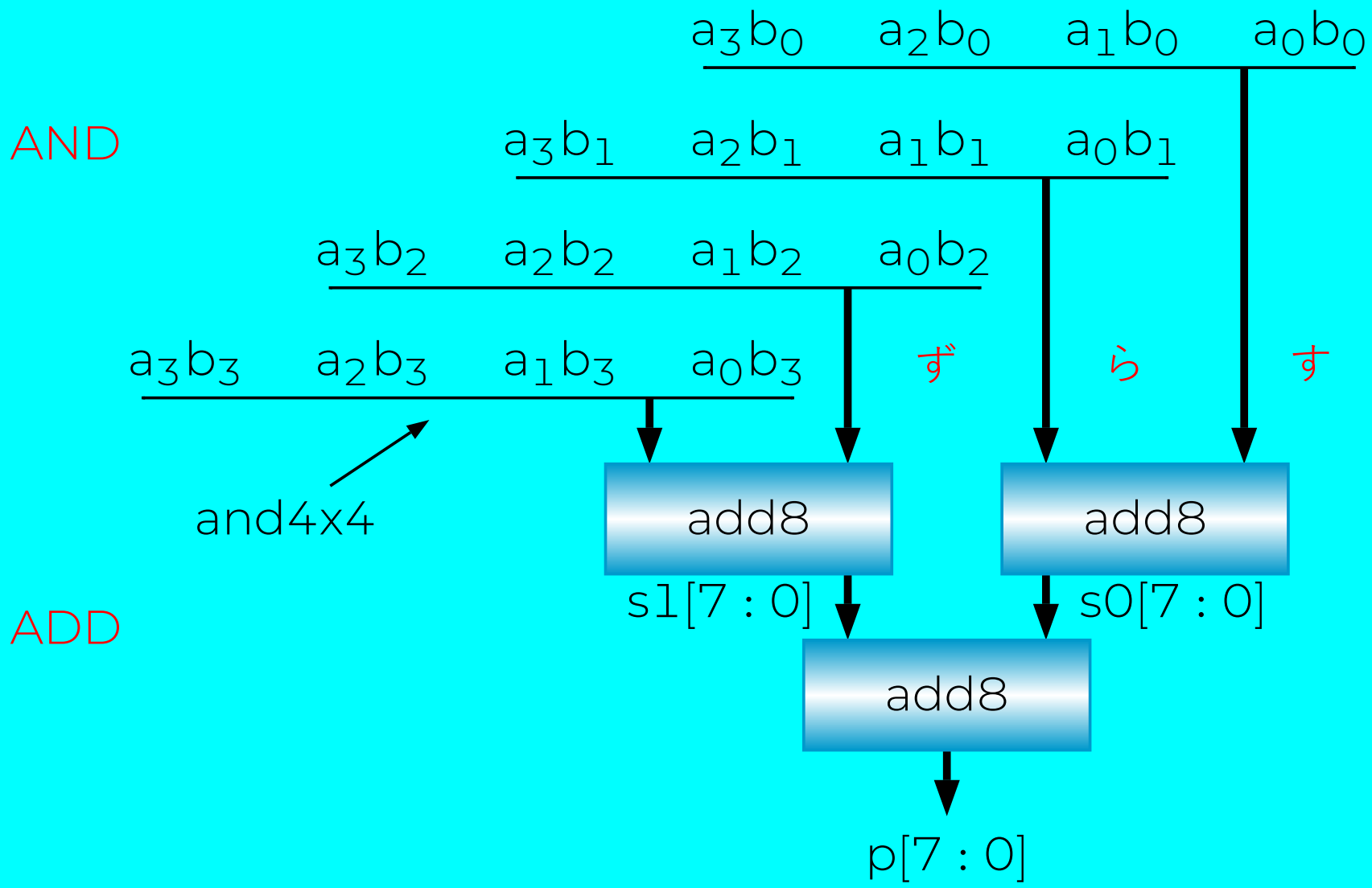
かける数

$4 \times 4 = 16$ AND gates

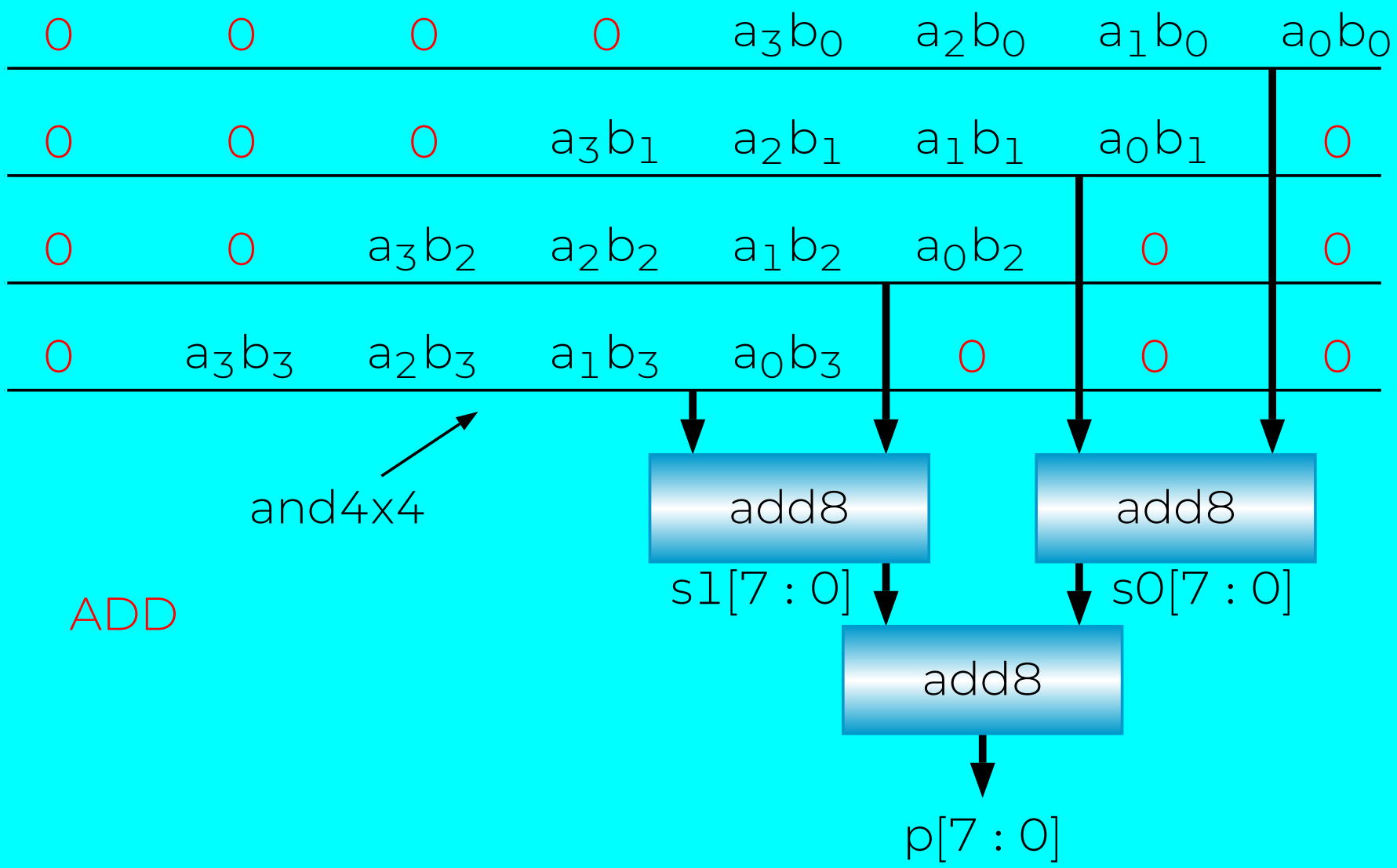
積

注意: $a_i \times b_i = a_i \cdot b_i = a_i b_i$ ・・・・・・・・・・・・・・・・・・・・・・・・・AND

符号なし数の掛け算


a_3b_0 a_2b_0 a_1b_0 a_0b_0
AND
a_3b_1 a_2b_1 a_1b_1 a_0b_1
a_3b_2 a_2b_2 a_1b_2 a_0b_2
a_3b_3 a_2b_3 a_1b_3 a_0b_3
ず
ら
す
and4x4
add8
add8
s1[7 : 0]
s0[7 : 0]
ADD
add8
p[7 : 0]

符号なし数の掛け算


0 0 0 0 a_3b_0 a_2b_0 a_1b_0 a_0b_0
0 0 0 a_3b_1 a_2b_1 a_1b_1 a_0b_1 0
0 0 a_3b_2 a_2b_2 a_1b_2 a_0b_2 0 0
0 a_3b_3 a_2b_3 a_1b_3 a_0b_3 0 0 0
and4x4
add8
add8
s1[7 : 0]
s0[7 : 0]
add8
p[7 : 0]
ADD

符号なし数の乗算器回路

```
‘timescale 1ns/1ns
module and4x4 (A, B, AB0, AB1, AB2, AB3);
    input  [3:0] A, B;
    output [3:0] AB0, AB1, AB2, AB3;

    assign AB0 = A & {4{B[0]}}; // A * B[0]
    assign AB1 = A & {4{B[1]}}; // A * B[1]
    assign AB2 = A & {4{B[2]}}; // A * B[2]
    assign AB3 = A & {4{B[3]}}; // A * B[3]

endmodule
```

注意:

```
{4{B[0]}} = {B[0],B[0],B[0],B[0]}
```

and4x4.v

符号なし数の乗算器回路

```
‘timescale 1ns/1ns
module mul4x4_tree (A, B, P);
    input  [3:0] A, B;
    output [7:0] P;

    wire   [3:0] AB0, AB1, AB2, AB3;
    and4x4 i0 (A, B, AB0, AB1, AB2, AB3);

    wire   [7:0] S0, S1;
    // add8 (a,                  b,                      s);
    add8 i1 ({4’b0000,AB0},     {3’b000,AB1,1’b0}, S0);
    add8 i2 ({2’b00,AB2,2’b00}, {1’b0,AB3,3’b000}, S1);
    add8 i3 (S0, S1, P);

endmodule
```

mul4x4_tree.v

符号なし数の乗算器回路

```
‘timescale 1ns/1ns
module add8 (A, B, S);
    input  [7:0] A, B;
    output [7:0] S;
    wire   [7:0] C;
    // add1 (a,    b,    ci,   co,   s);
    add1 i0 (A[0], B[0], 1’b0, C[0], S[0]);
    add1 i1 (A[1], B[1], C[0], C[1], S[1]);
    add1 i2 (A[2], B[2], C[1], C[2], S[2]);
    add1 i3 (A[3], B[3], C[2], C[3], S[3]);
    add1 i4 (A[4], B[4], C[3], C[4], S[4]);
    add1 i5 (A[5], B[5], C[4], C[5], S[5]);
    add1 i6 (A[6], B[6], C[5], C[6], S[6]);
    add1 i7 (A[7], B[7], C[6], C[7], S[7]);
endmodule
```

add8.v

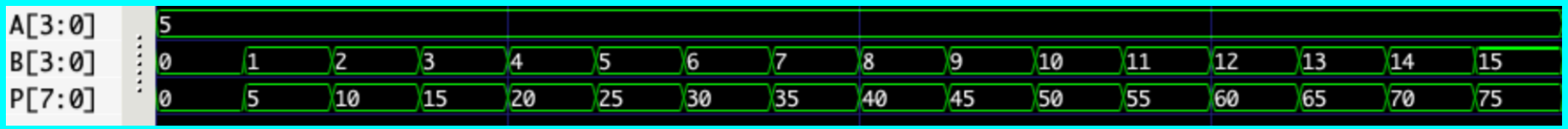
符号なし数の乗算器テストベンチ

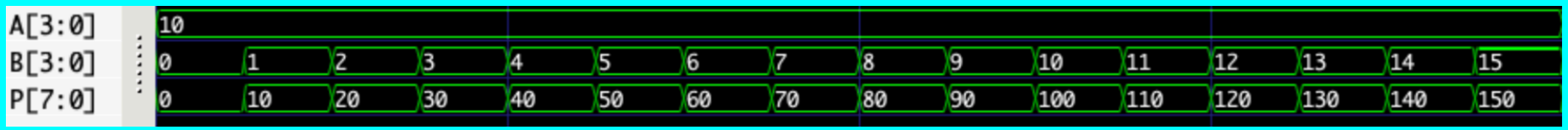
```
‘timescale 1ns/1ns
module mul4x4_tree_tb;
    reg  [3:0] A, B;
    wire [7:0] P;
    mul4x4_tree i0 (A, B, P);
    integer    i, j;
    initial begin
        for (i = 0; i < 16; i = i + 1) begin
            for (j = 0; j < 16; j = j + 1) begin
                A = i; B = j;
                #1 ;
            end
        end
        #1 $finish;
    end
    initial begin
        $dumpfile ("mul4x4_tree.vcd");
        $dumpvars;
    end
endmodule
```

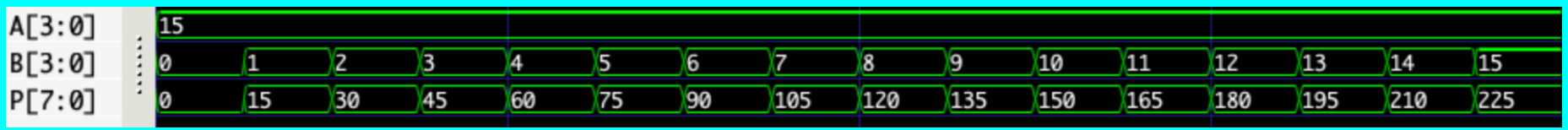
mul4x4_tree_tb.v

符号なし数の乗算器波形

```
% iverilog -Wall -o mul4x4_tree mul4x4_tree_tb.v \
  mul4x4_tree.v and4x4.v add8.v add1.v half_adder.v
% ./mul4x4_tree
% gtkwave mul4x4_tree.vcd
```







符号なし $P = A \times B$

$15 \times 15 = 225$

CSA を使用する乗算器回路 (4 ビット)

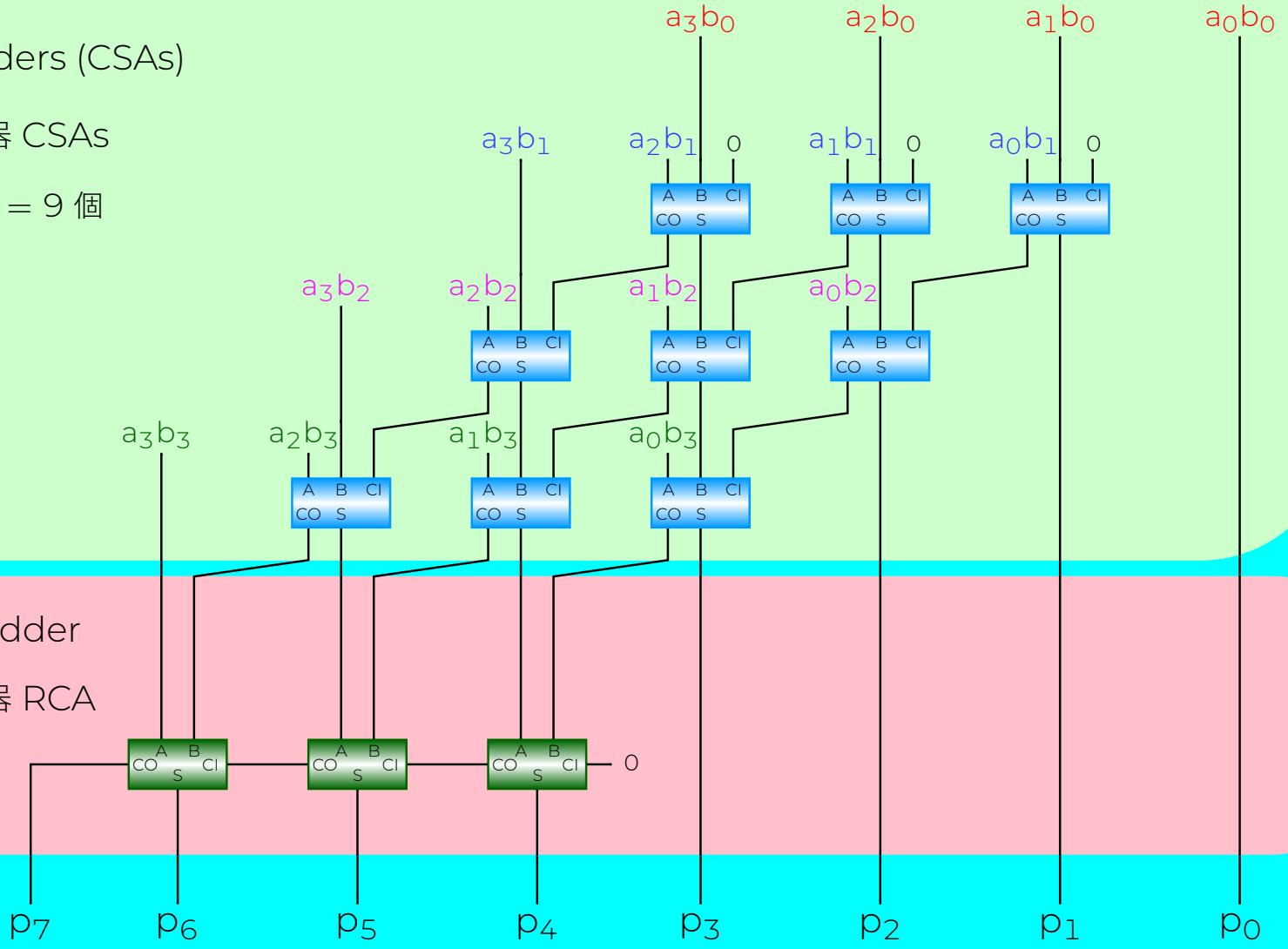
Carry Save Adders (CSAs)

桁上げ保存加算器 CSAs

$(n-1) \times (n-1) = 9$ 個

段数: $n-1=3$

$n=4$: ビット数

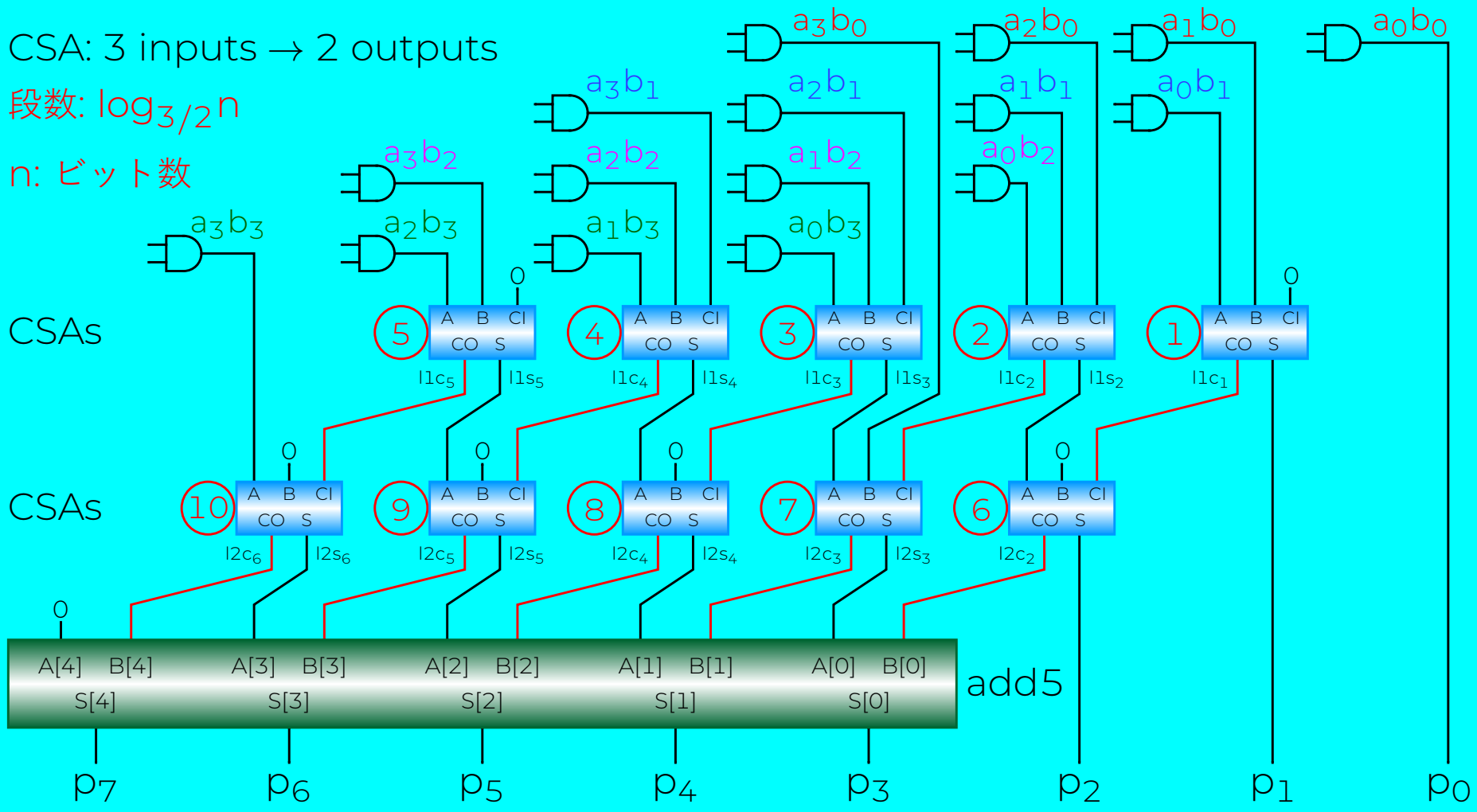


Ripple Carry Adder

桁上げ伝播加算器 RCA

(1個)

Wallace Tree (ウォレスツリー)



計算可能であれば、大量の CSA を使用して並列に計算する。

ウォレスツリー乗算器回路

```
‘timescale 1ns/1ns
module wallace4 (a, b, p);
    input  [3:0] a, b;
    output [7:0] p;
    wire   [3:0] ab0, ab1, ab2, ab3;
    and4x4 i0 (a, b, ab0, ab1, ab2, ab3);
    assign p[0] = ab0[0];
    wire   [5:1] l1c; // level 1 carry
    wire   [5:2] l1s; // level 1 sum
    wire   [6:2] l2c; // level 2 carry
    wire   [6:3] l2s; // level 2 sum
    // csa  (a,      b,      ci,     co,     s); // same as add1
    csa i1  (ab1[0], ab0[1], 1’b0,   l1c[1], p[1]);   // level 1
    csa i2  (      ,       ,       ,       ,       ); // level 1
    csa i3  (      ,       ,       ,       ,       ); // level 1
    csa i4  (      ,       ,       ,       ,       ); // level 1
    csa i5  (      ,       ,       ,       ,       ); // level 1
    csa i6  (l1s[2], 1’b0,   l1c[1], l2c[2], p[2]);   // level 2
    csa i7  (      ,       ,       ,       ,       ); // level 2
    csa i8  (      ,       ,       ,       ,       ); // level 2
    csa i9  (      ,       ,       ,       ,       ); // level 2
    csa i10 (      ,       ,       ,       ,       ); // level 2
    // add5  (a,               b,        s);
    add5 i11 ({1’b0,l2s[6:3]}, l2c[6:2], p[7:3]);
endmodule
```

csa.v

wallace4.v

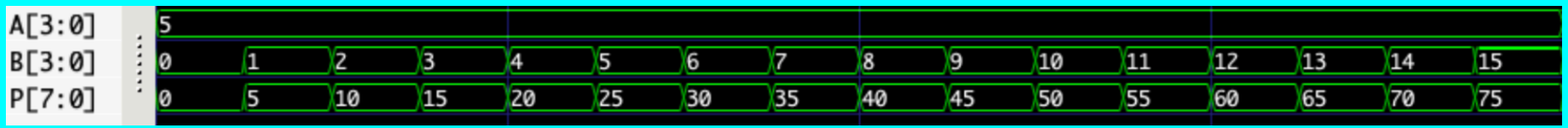
ウォレスツリー乗算器テストベンチ

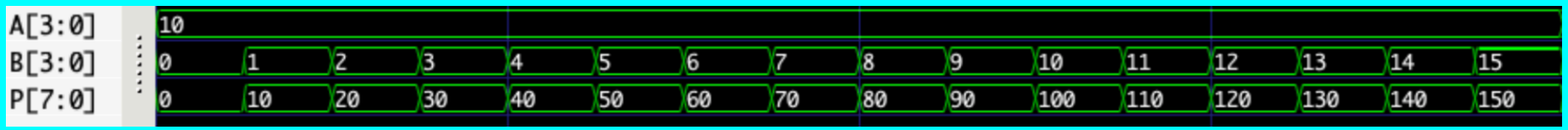
```
‘timescale 1ns/1ns
module wallace4_tb;
    reg  [3:0] A, B;
    wire [7:0] P;
    wallace4 i0 (A, B, P);
    integer    i, j;
    initial begin
        for (i = 0; i < 16; i = i + 1) begin
            for (j = 0; j < 16; j = j + 1) begin
                A = i; B = j;
                #1 ;
            end
        end
        #1 $finish;
    end
    initial begin
        $dumpfile ("wallace4.vcd");
        $dumpvars;
    end
endmodule
```

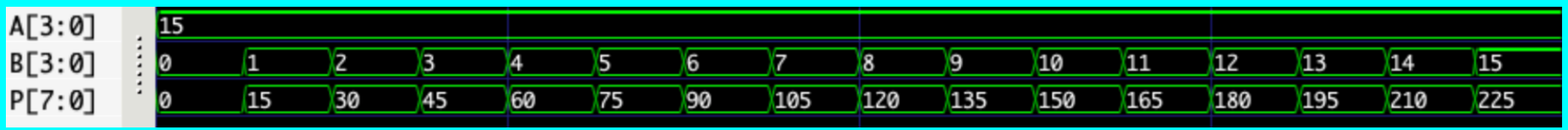
wallace4_tb.v

ウォレスツリー乗算器波形

```
% iverilog -Wall -o wallace4 wallace4_tb.v wallace4.v \
  add5.v add1.v csa.v half_adder.v and4x4.v
% ./wallace4
% gtkwave wallace4.vcd
```





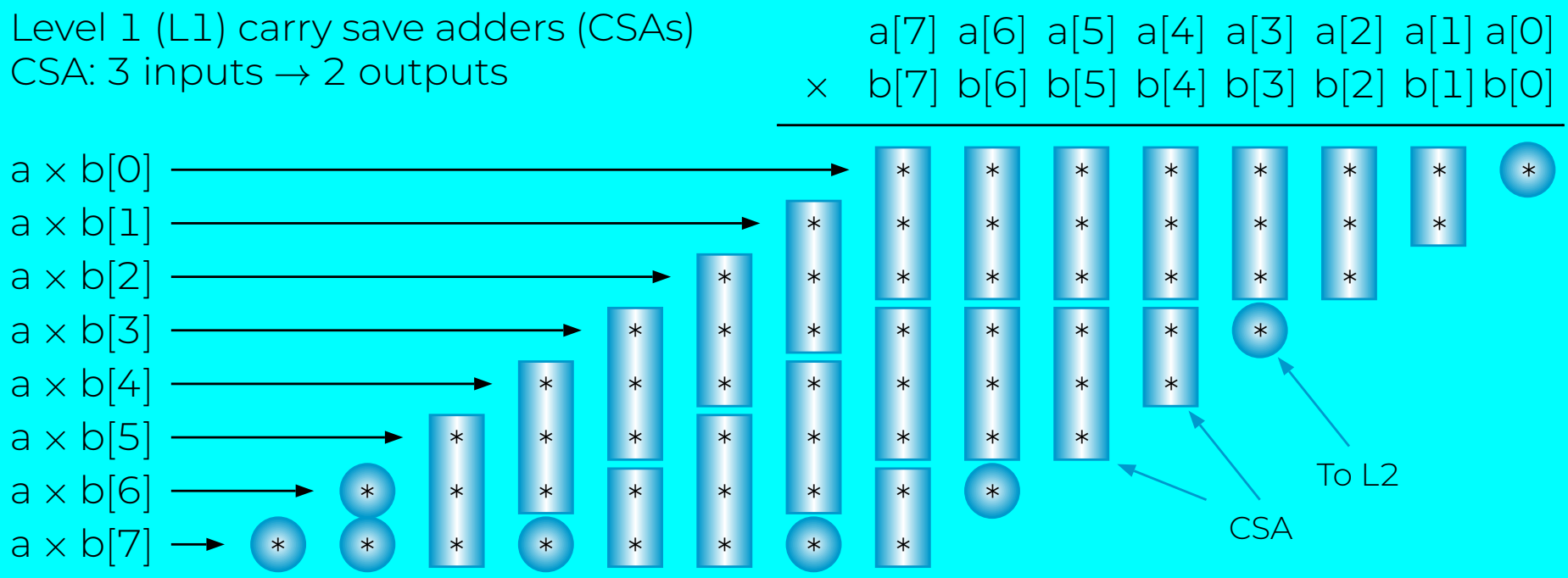


符号なし $P = A \times B$

$15 \times 15 = 225$

ウォレスツリー (8ビット)

Level 1 (L1) carry save adders (CSAs)
CSA: 3 inputs → 2 outputs

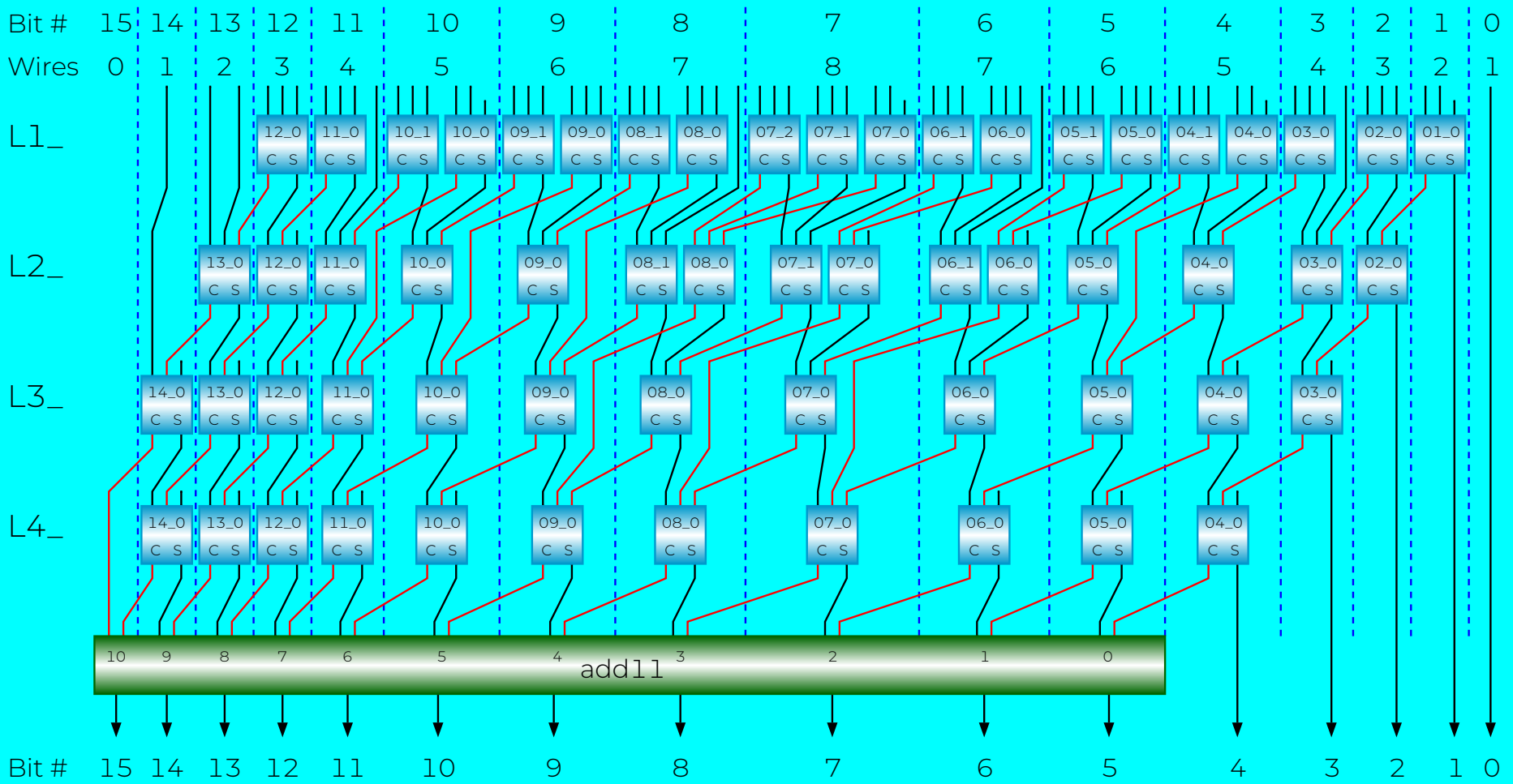


Bit number	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
# of wires (*)	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1
# of CSAs	0	0	1	1	2	2	2	3	2	2	2	1	1	1	0

段１（L1）の CSAs（# of CSAs：CSA の個数）
まるアスタリスク：段２（L2）に送る。

計算可能であれば、大量の CSA を使用して並列に計算する。

ウォレスツリー (8ビット)

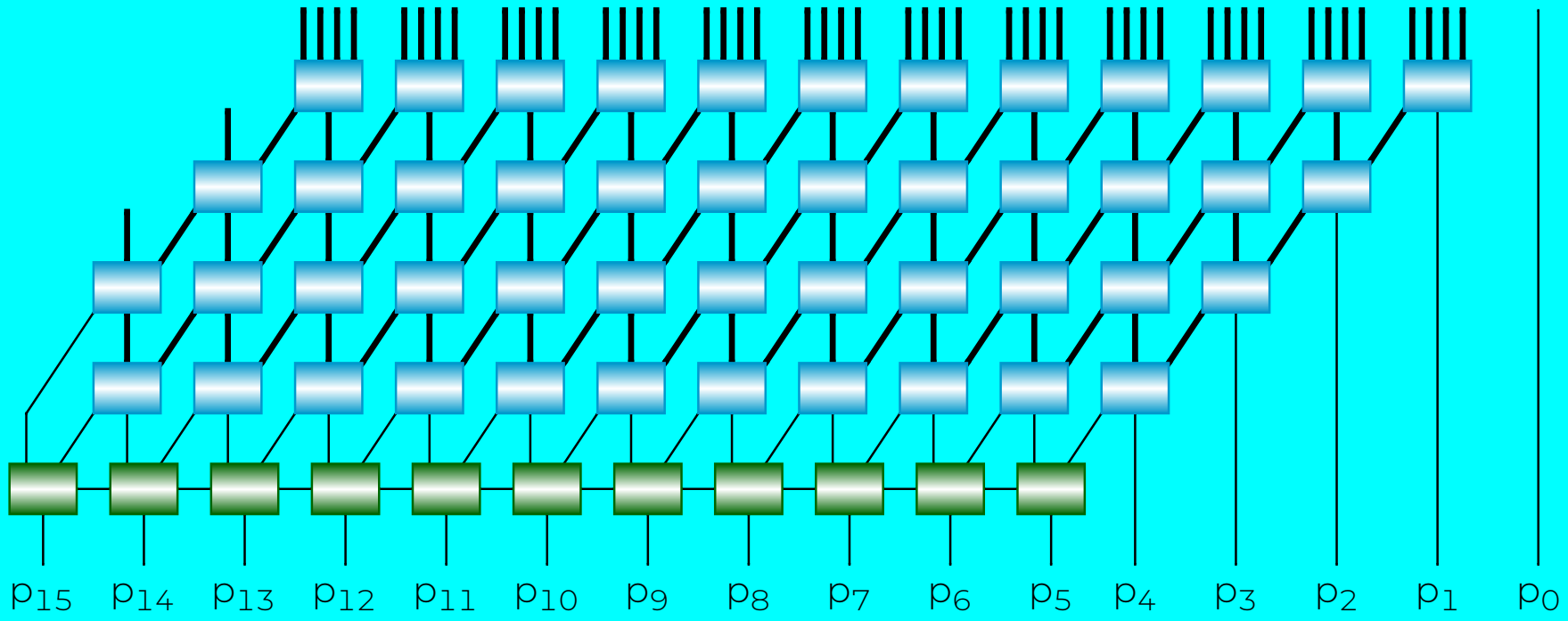


CSA: 3 inputs → 2 outputs;　　段数 $= \log_{3/2} n$

計算可能であれば、大量の CSA を使用して並列に計算する。

ウォレスツリー (8ビット)

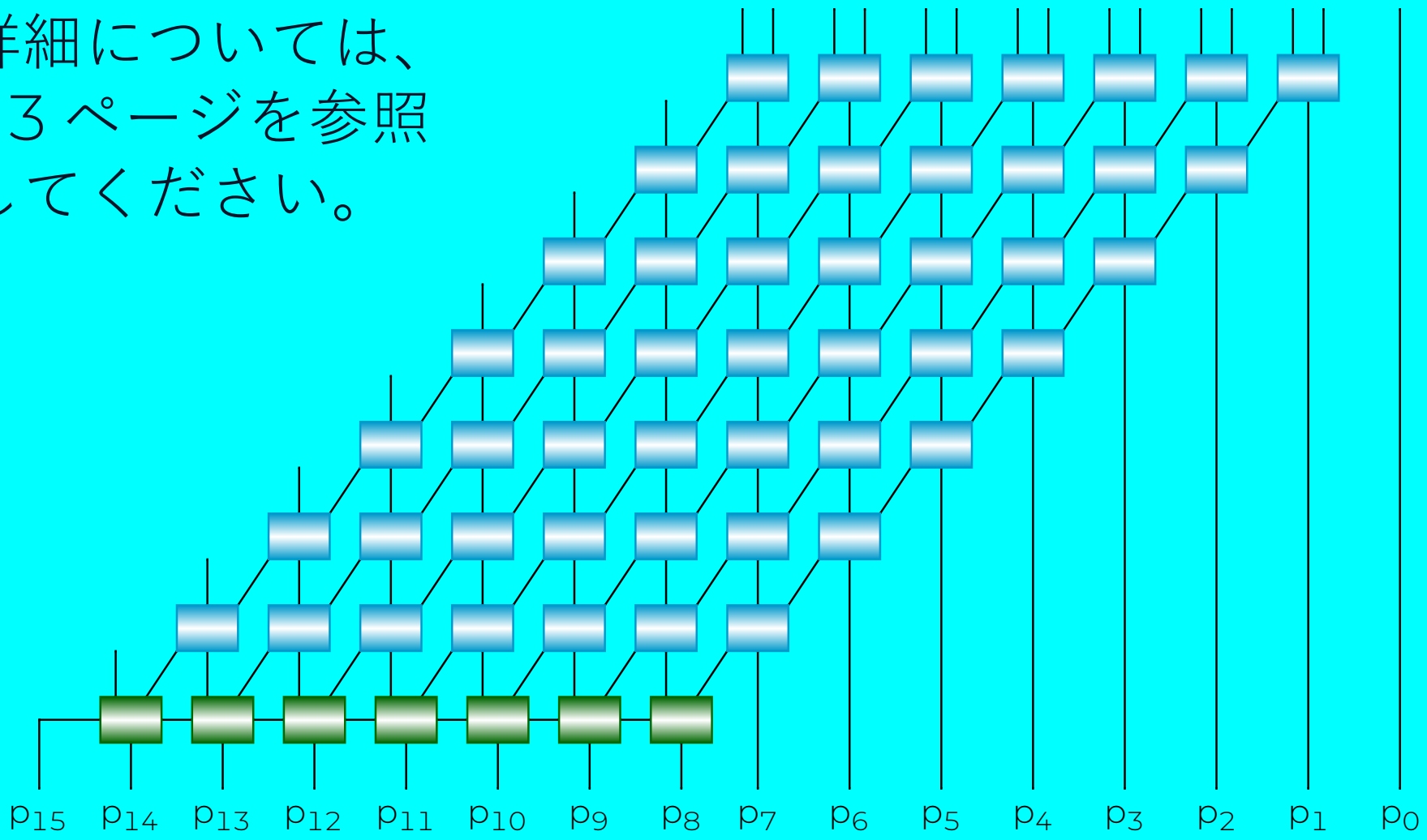
綺麗に整列されたウォレスツリーの回路図。
詳細については、前のページを参照してください。



4 段 CSAs

CSA を使用する乗算器回路 (8 ビット)

詳細については、
13 ページを参照
してください。



7 段 CSAs

2の補数で表現する数の掛け算

- 符号なし数の掛け算

  $1111 \times 1111 = 11100001$ ……$15 \times 15 = 225$

  $1111 \times 0010 = 00011110$ ……$15 \times 2 = 30$

- 2の補数で表現する数の掛け算

  $1111 \times 1111 = 00000001$ …$(-1) \times (-1) = +1$

  $1111 \times 0010 = 11111110$ …$(-1) \times (+2) = -2$

- 2の補数掛け算のルール

  正 × 正 = 正

  正 × 負 = 負

  負 × 正 = 負

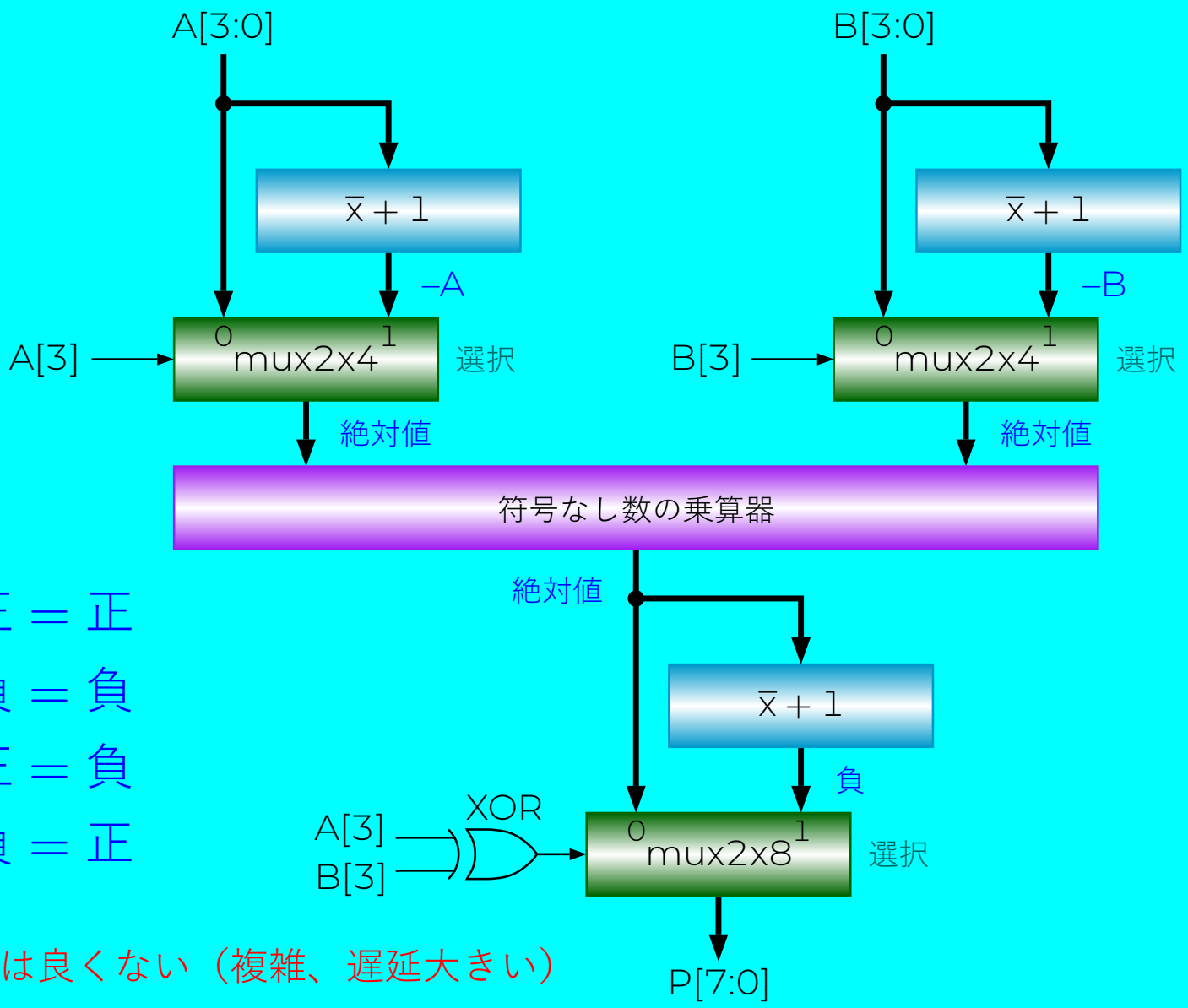
  負 × 負 = 正

↑

赤：下位4ビットは同じである

青：下位4ビットは同じである

2 の補数で表現する数の掛け算



正 × 正 = 正

正 × 負 = 負

負 × 正 = 負

負 × 負 = 正

この回路は良くない（複雑、遅延大きい）

2 の補数で表現する数の掛け算

$A_4 = a_3a_2a_1a_0 = -a_3 \times 2^3 + \sum_{i=0}^{2} a_i \times 2^i = -a_3 \times 2^3 + A_3$

$B_4 = b_3b_2b_1b_0 = -b_3 \times 2^3 + \sum_{i=0}^{2} b_i \times 2^i = -b_3 \times 2^3 + B_3$

A_3 と B_3: 符号なし数

$(a+b)(x+y) = ax + ay + bx + by$ なので、

$$
\begin{aligned}
P_8 &= A_4 \times B_4 \\
&= (-a_3 \times 2^3 + A_3) \times (-b_3 \times 2^3 + B_3) \\
&= a_3 \times b_3 \times 2^6 \\
&+ (-a_3 \times B_3) \times 2^3 \quad \text{(負)} \\
&+ (-b_3 \times A_3) \times 2^3 \quad \text{(負)} \\
&+ A_3 \times B_3
\end{aligned}
$$

2 の補数で表現する数の掛け算

$-X = \overline{X} + 1$ なので、

$$(-a_3 \times B_3) \times 2^3 = (\overline{0}, \overline{0}, \overline{a_3 b_2}, \overline{a_3 b_1}, \overline{a_3 b_0} + 1) \times 2^3$$

$$(-b_3 \times A_3) \times 2^3 = (\overline{0}, \overline{0}, \overline{a_2 b_3}, \overline{a_1 b_3}, \overline{a_0 b_3} + 1) \times 2^3$$

つまり

Bit#	7	6	5	4	3	
	1	1	$\overline{a_3 b_2}$	$\overline{a_3 b_1}$	$\overline{a_3 b_0}$	反転
	0	0	0	0	1	+1
	1	1	$\overline{a_2 b_3}$	$\overline{a_1 b_3}$	$\overline{a_0 b_3}$	反転
+	0	0	0	0	1	+1

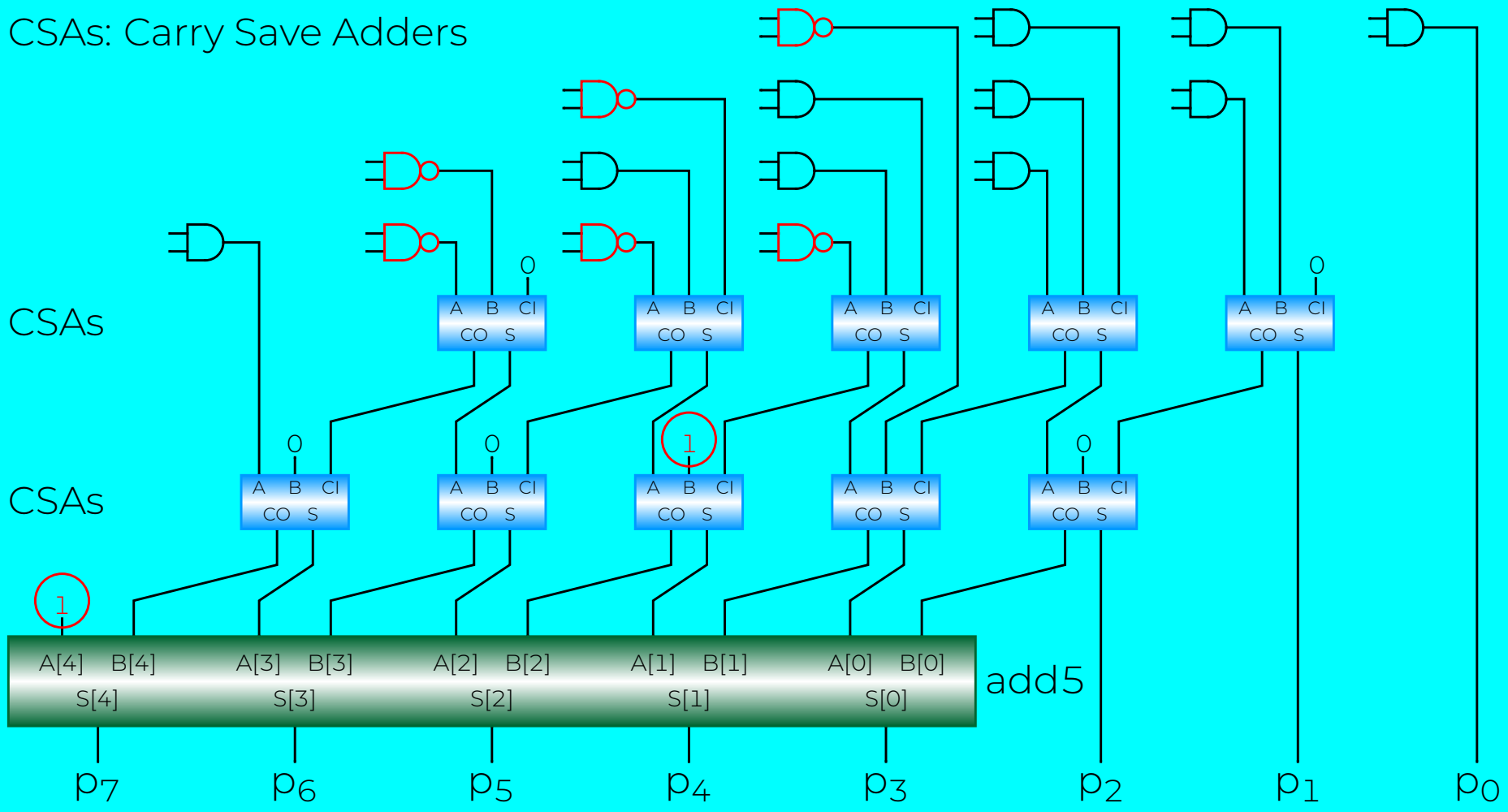
2 の補数で表現する数の掛け算

Bit#	7	6	5	4	3	
	1	1	$\overline{a_3b_2}$	$\overline{a_3b_1}$	$\overline{a_3b_0}$	反転
	0	0	0	0	1	+1
	1	1	$\overline{a_2b_3}$	$\overline{a_1b_3}$	$\overline{a_0b_3}$	反転
+	0	0	0	0	1	+1

整理

Bit#	7	6	5	4	3
	0	0	$\overline{a_3b_2}$	$\overline{a_3b_1}$	$\overline{a_3b_0}$
	0	0	$\overline{a_2b_3}$	$\overline{a_1b_3}$	$\overline{a_0b_3}$
+	1	0	0	1	0

2 の補数のウォレスツリーの回路



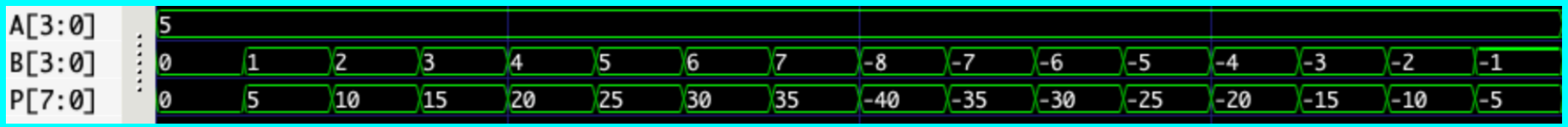
テストベンチ

```
‘timescale 1ns/1ns
module wallace4_signed_tb;
    reg  [3:0] A, B;
    wire [7:0] P;
    wallace4_signed i0 (A, B, P);
    integer    i, j;
    initial begin
        for (i = 0; i < 16; i = i + 1) begin
            for (j = 0; j < 16; j = j + 1) begin
                A = i; B = j;
                #1 ;
            end
        end
        #1 $finish;
    end
    initial begin
        $dumpfile ("wallace4_signed.vcd");
        $dumpvars;
    end
endmodule
```

wallace4_signed_tb.v

2 の補数のウォレスツリーの波形

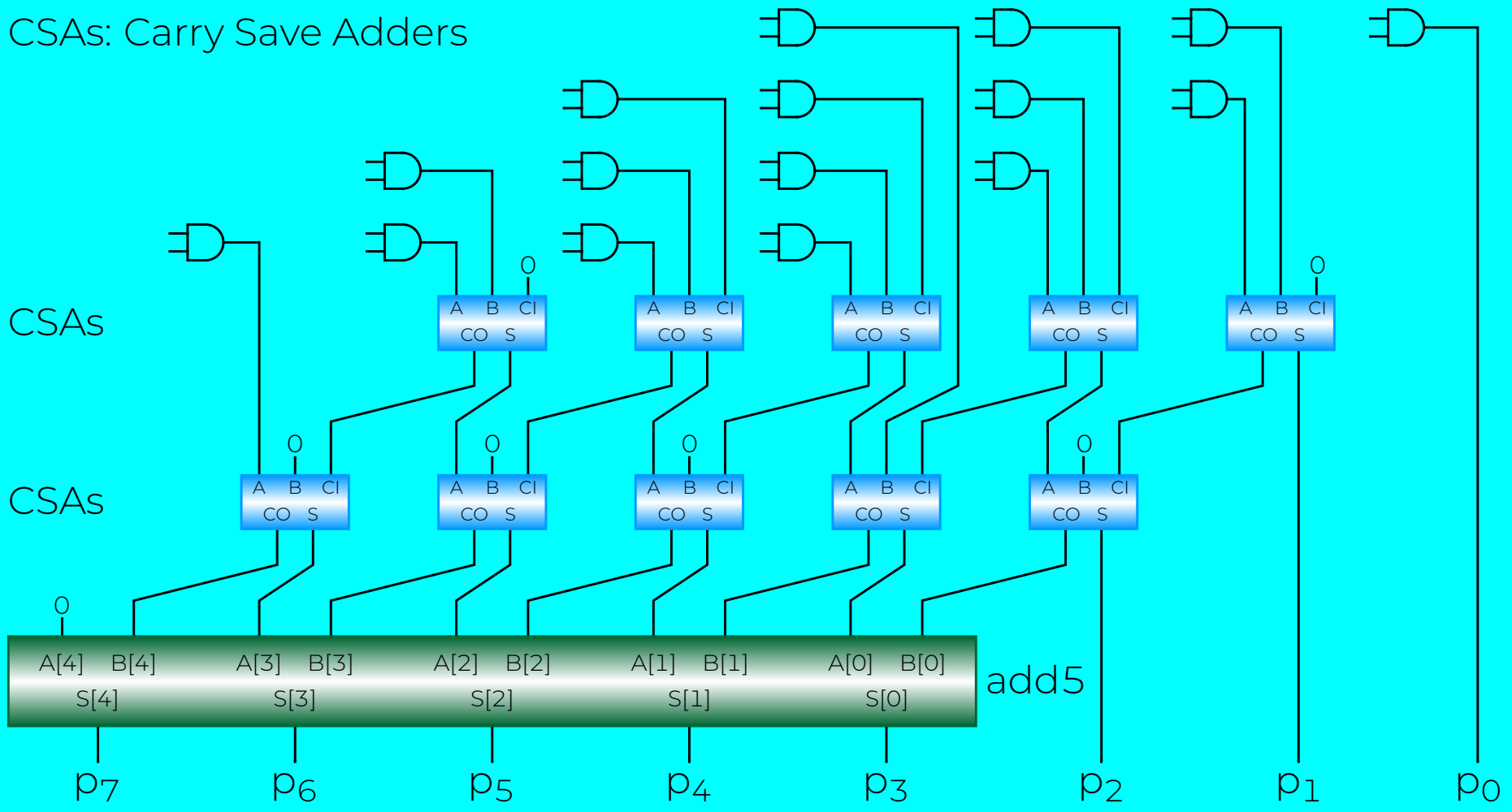
```
% iverilog -Wall -o wallace4_signed wallace4_signed_tb.v \
  wallace4_signed.v add5.v add1.v csa.v half_adder.v and4x4.v
% ./wallace4_signed
% gtkwave wallace4_signed.vcd
```



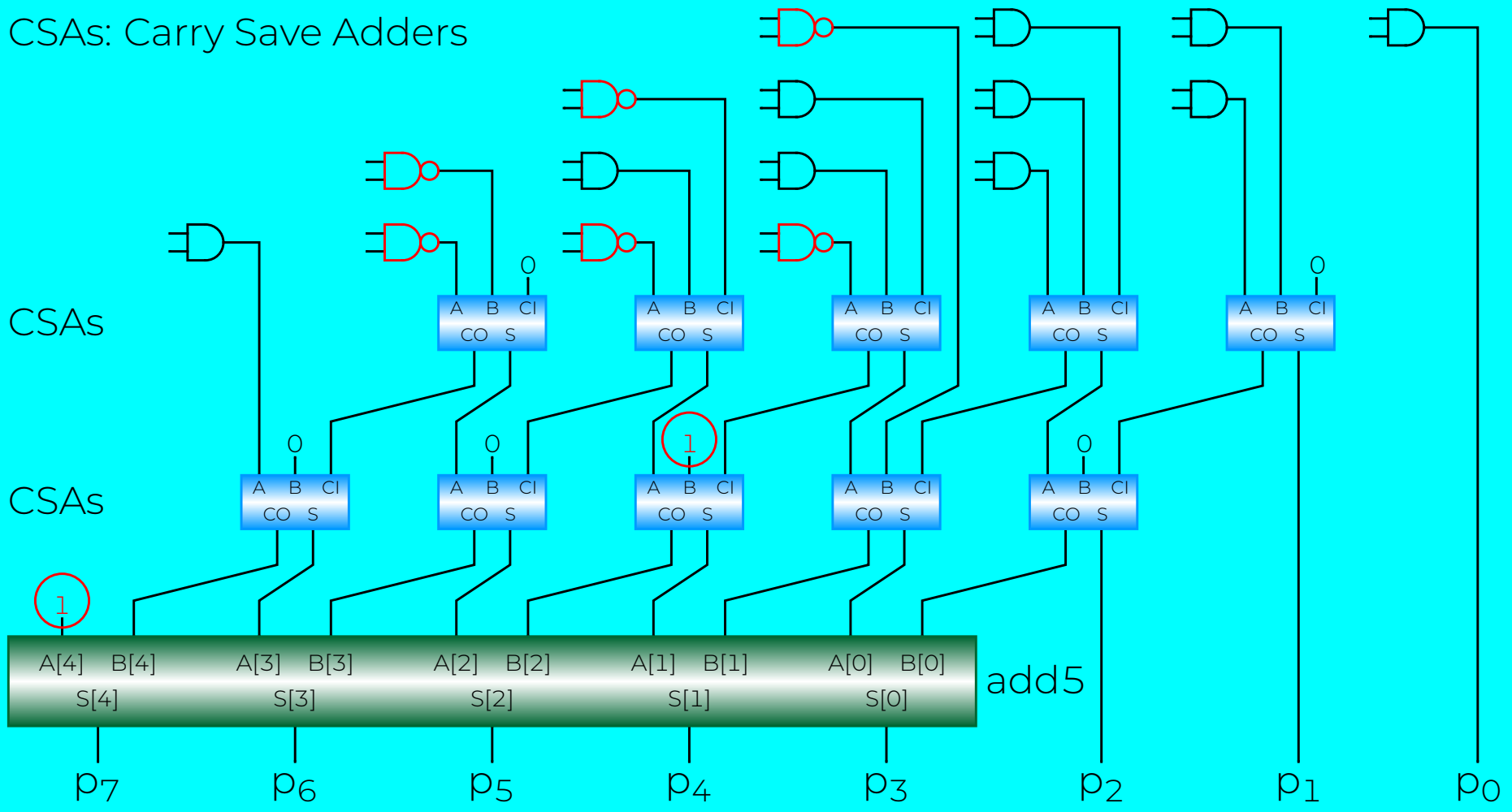
2 の補数 $P = A \times B$

$$(-1) \times (-1) = 1$$

符号なしのウォレスツリーの回路



2 の補数のウォレスツリーの回路



2 進数の掛け算

まとめ

- 符号なし数の掛け算（乗算）
- 符号なし乗算器回路
- 符号なし CSA を使用する乗算器回路
- 符号なしのウォレスツリー乗算器
- 2 の補数で表現する数の掛け算
- 2 の補数のウォレスツリー乗算器

課題 VIII (100 点 + 200 点)

４ビット × ４ビット符号なしのウォレスツリーを設計し動作検証シミュレーションして下さい。

入力信号: A[3:0] . . . . . 4-bit data
入力信号: B[3:0] . . . . . 4-bit data
出力信号: P[7:0] . . . . . 8-bit result

モジュール名は wallace4 にすること。
テストベンチ wallace4_tb.v を使って下さい。

オプション (+50 点): ４ビット × ４ビット２の補数のウォレスツリーを設計し動作検証シミュレーションして下さい。
モジュール名は wallace4_signed にすること。
テストベンチwallace4_signed_tb.v を使って下さい。

課題 VIII (100 点 + 200 点)

オプション (+100 点): 8 ビット × 8 ビット符号なしのウォレスツリーを設計し動作検証シミュレーションして下さい。

入力信号: A[7:0] . . . . . 8-bit data
入力信号: B[7:0] . . . . . 8-bit data
出力信号: P[15:0] . . . . . 16-bit result

モジュール名は wallace8 にすること。
テストベンチを自分で作って下さい。

オプション (+50 点): 8 ビット × 8 ビット 2 の補数のウォレスツリーを設計し動作検証シミュレーションして下さい。
モジュール名は wallace8_signed にすること。
テストベンチを自分で作って下さい。

発展：自由練習

P13 の CSA を使用する 4 ビット乗算器回路を設計し動作検証シミュレーションして下さい。

発展：自由練習

S を符号なしの 4 ビット A かける 4 ビット B の 8 ビット積とする

$$A_3A_2A_1A_0 \times B_3B_2B_1B_0 = S_7S_6S_5S_4S_3S_2S_1S_0$$

T を 2 の補数の 4 ビット A かける 4 ビット B の 8 ビット積とする

$$A_3A_2A_1A_0 \times B_3B_2B_1B_0 = T_7T_6T_5T_4T_3T_2T_1T_0$$

証明：

$$S_3S_2S_1S_0 = T_3T_2T_1T_0$$