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

2022 年 11 月 15 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 CSAs 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})

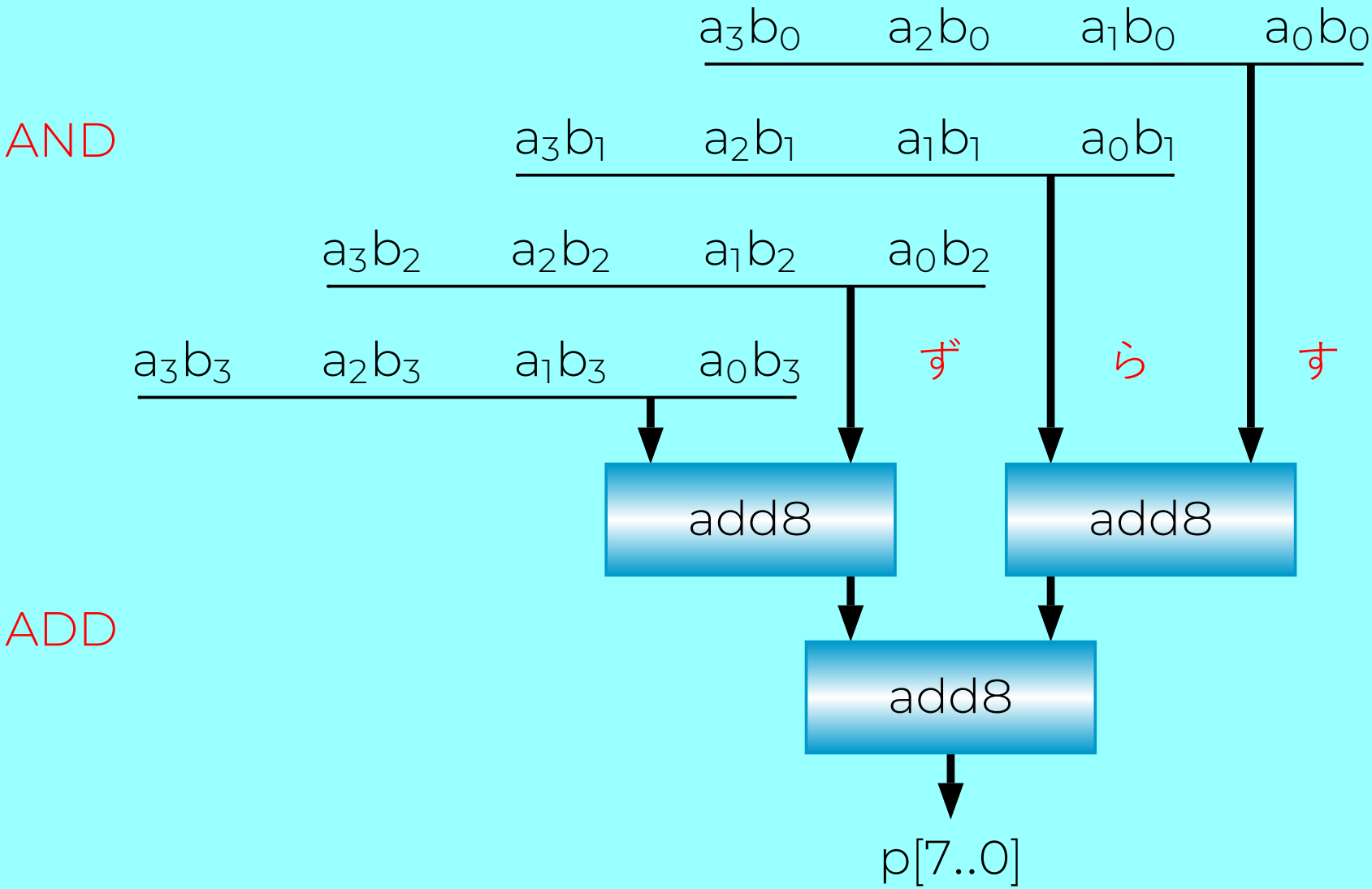
2 倍のビット数が必要

符号なし数の掛け算

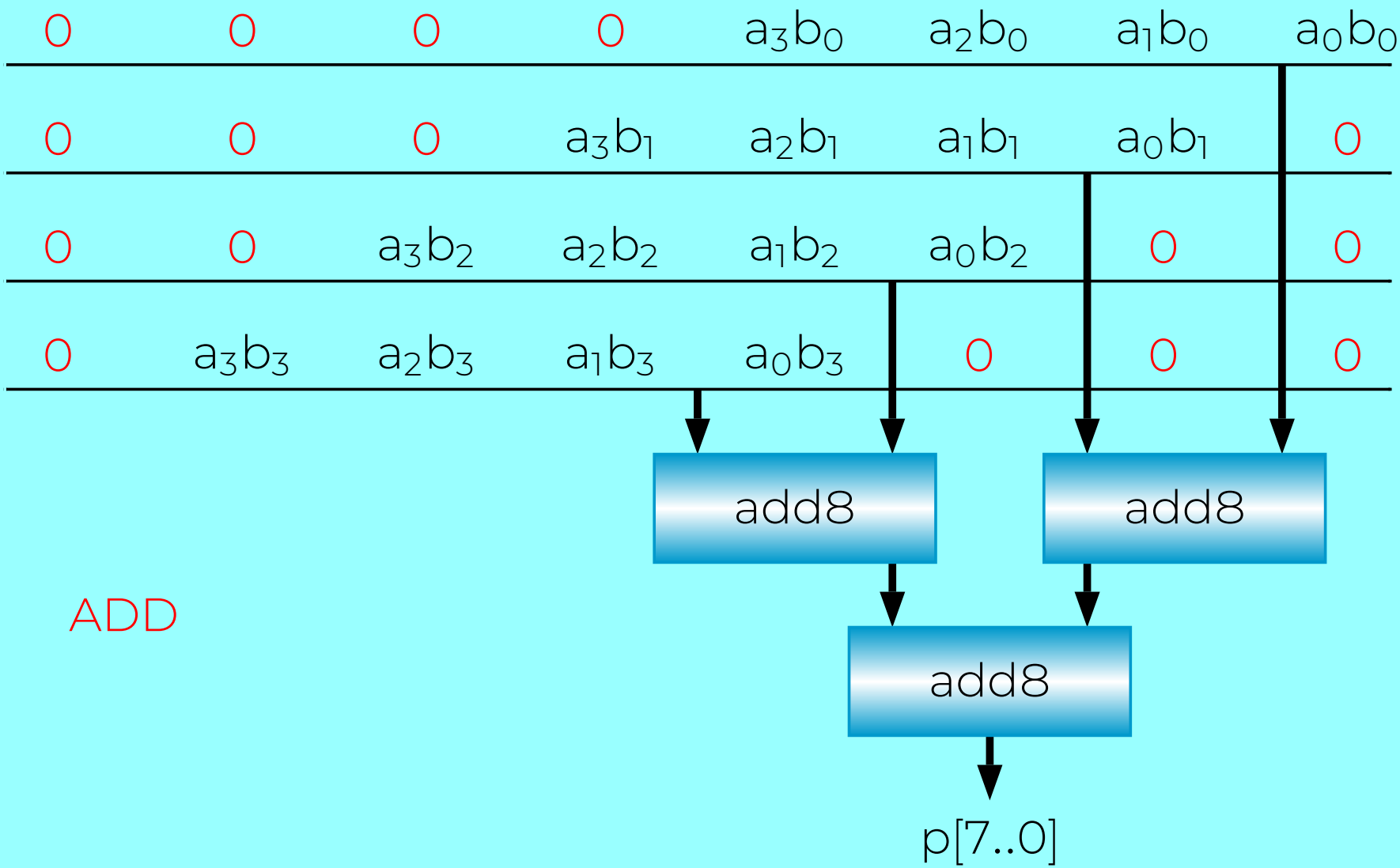
$$
\begin{array}{rcccccc}
 & & a_3 & a_2 & a_1 & a_0 & \text{かけられる数} \\
 & \times & b_3 & b_2 & b_1 & b_0 & \text{かける数} \\
\hline
 & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 & \left.\begin{array}{l}\\ \\ \\ \\ \end{array}\right\} 4 \times 4 = 16 \\
 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & & \text{AND gates} \\
a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & & \\
+\ a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 \quad p_0 \quad \text{積}
\end{array}
$$

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


AND
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
ず
ら
す
add8
add8
ADD
add8
p[7..0]

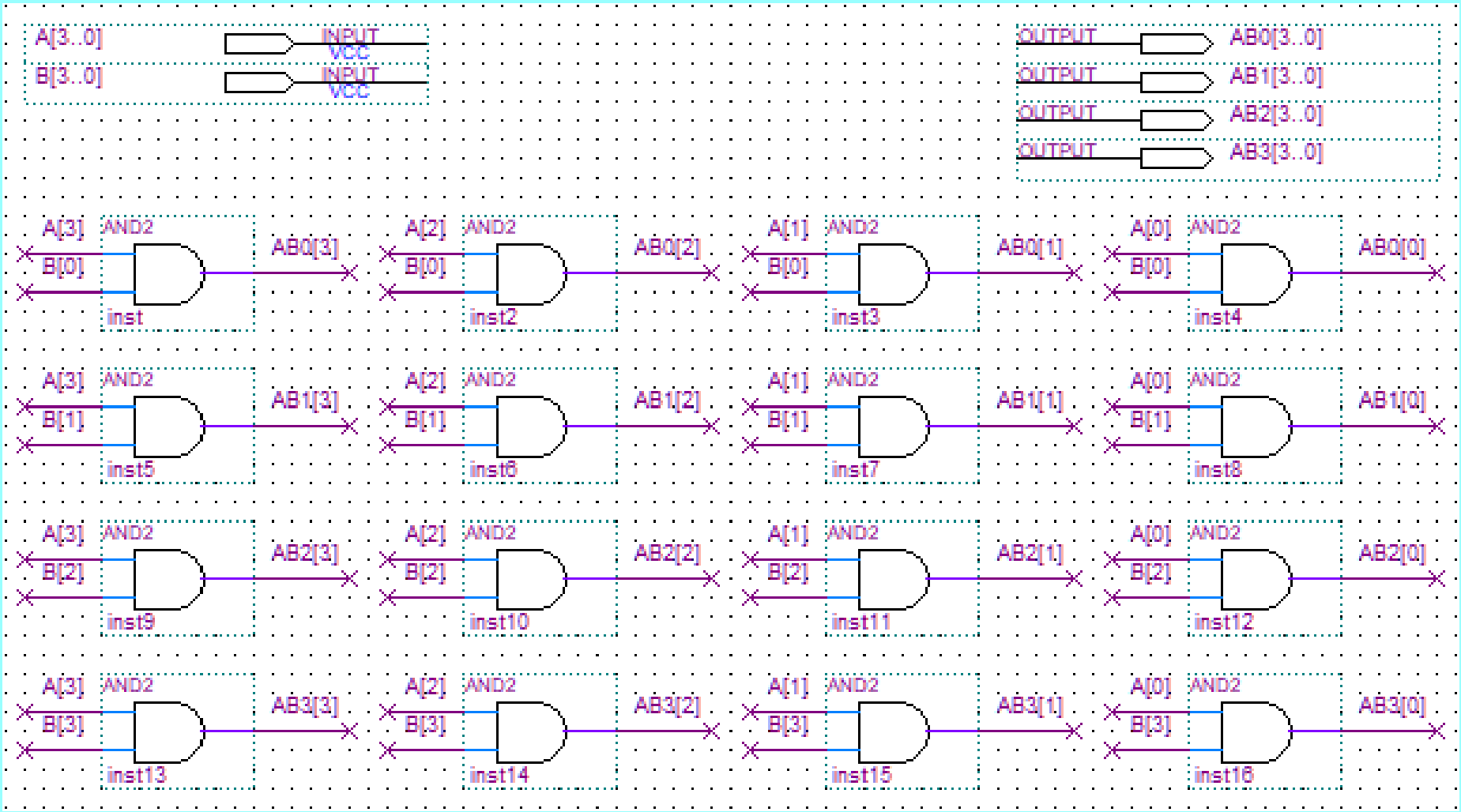
符号なし数の掛け算



ADD

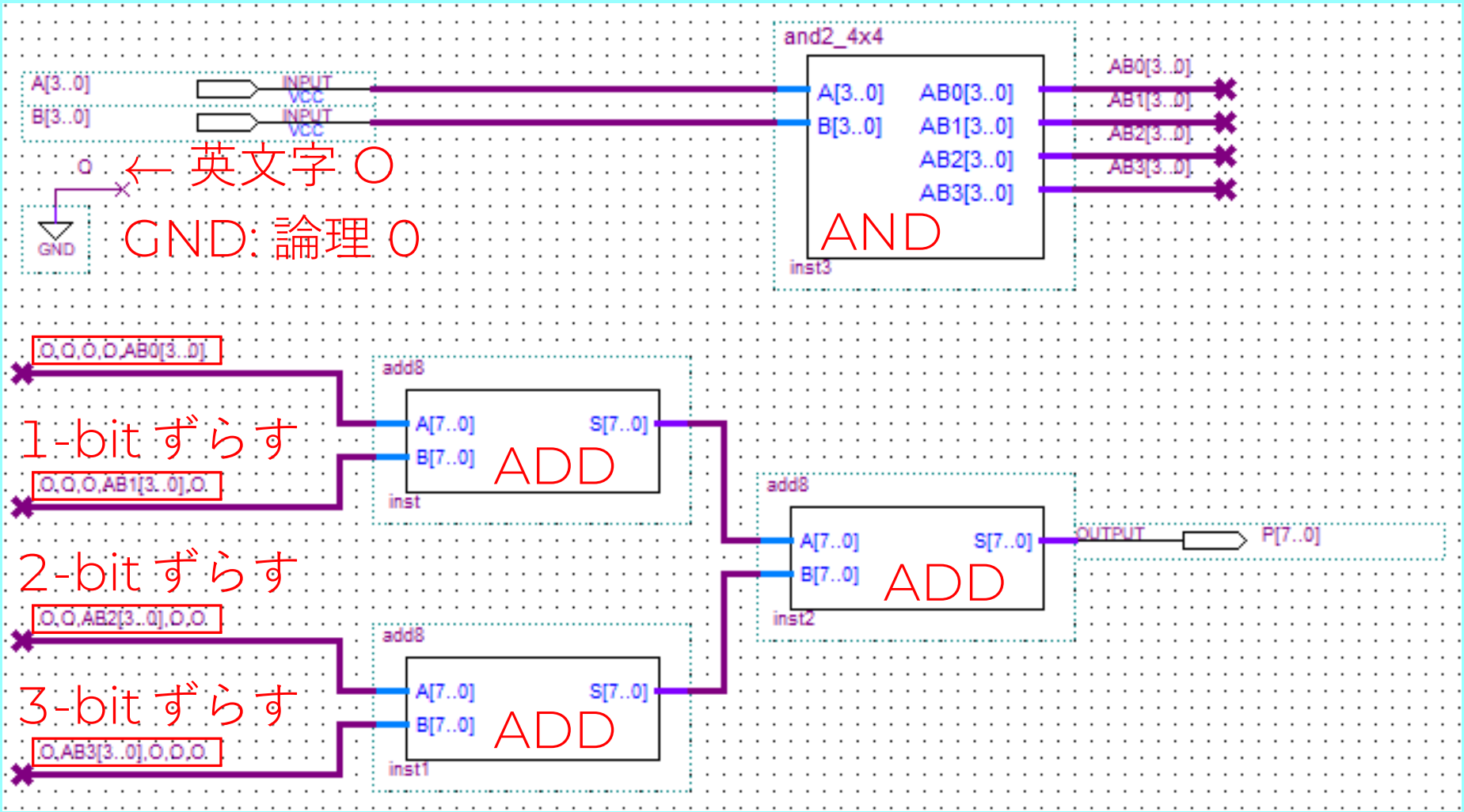
符号なし数の乗算器回路

and2_4x4.bdf



符号なし数の乗算器回路

mul4x4_tree.bdf

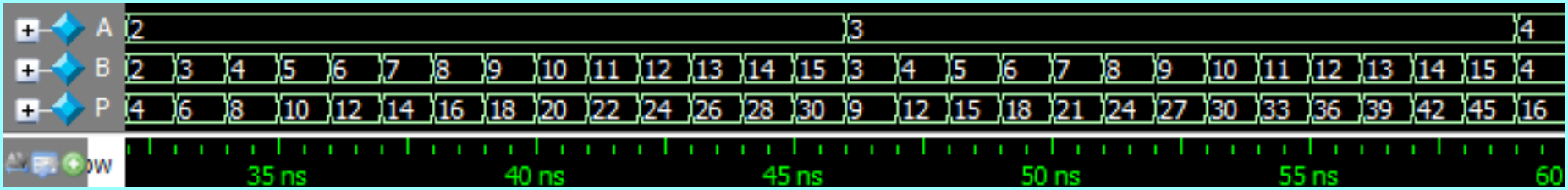


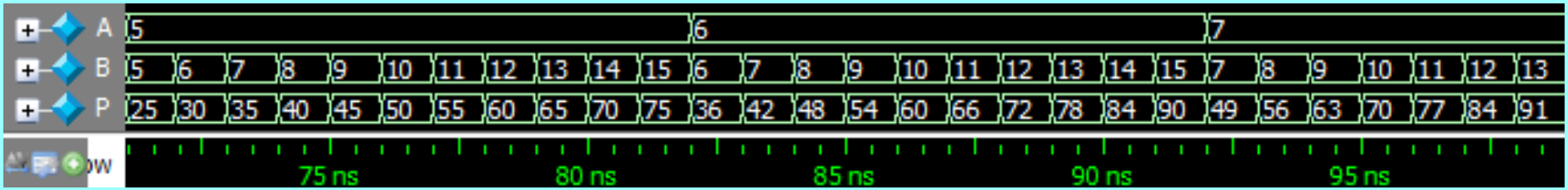
テストベンチ

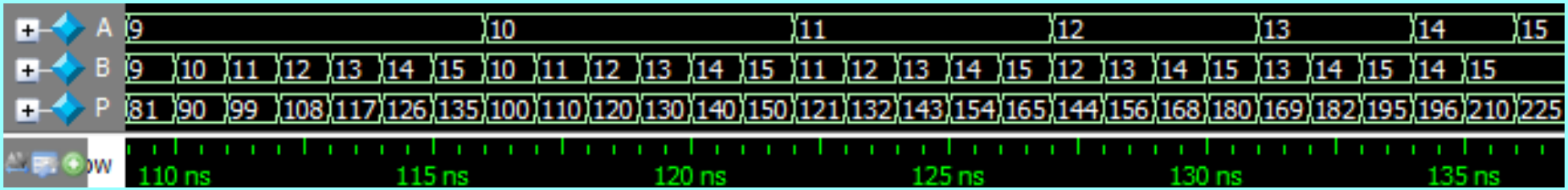
回路のモジュール mul4x4_tree を呼び出し、入力のパターンを用意する。

```
‘timescale 1ns/1ps // unit = 1 ns; accuracy = 1 ps
module mul4x4_tree_tb;
   reg  [3:0] A,B;
   wire [7:0] P;
   mul4x4_tree i1 (.A(A), .B(B), .P(P));
   integer i,j;
   initial begin
      #0 A = 0; B = 0;
      for (i = 0; i < 16; i = i + 1) begin
         for (j = i; j < 16; j = j + 1) begin
            #1 A = i; B = j;
         end
      end
      #1 $stop;
   end
endmodule
```

乗算器回路のシミュレーション







⇑

Radix: Unsigned (符号なし)

$15 \times 15 = 225$

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) × (n − 1) = 9 個)

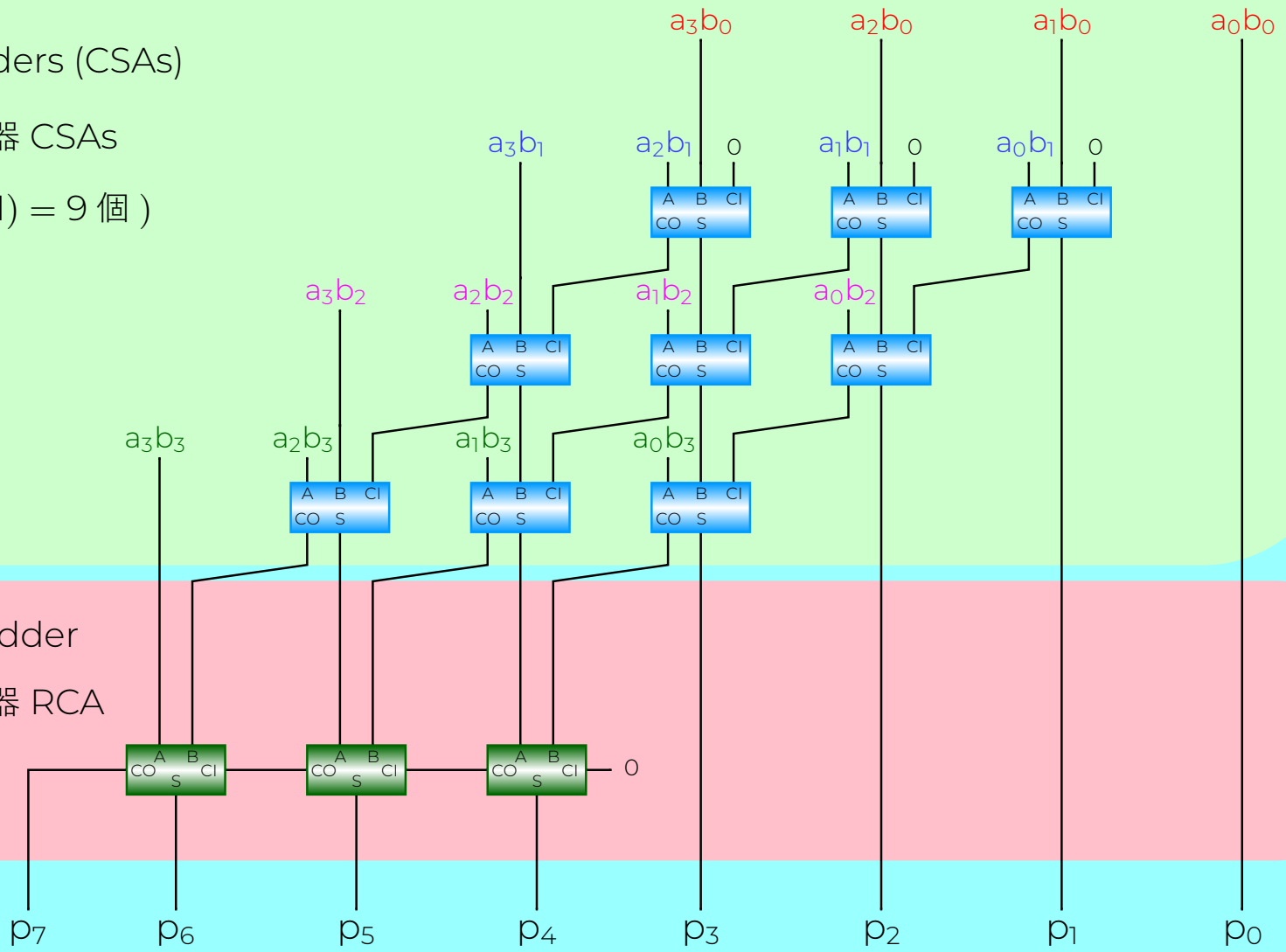
段数: n − 1 = 3

n = 4: ビット数

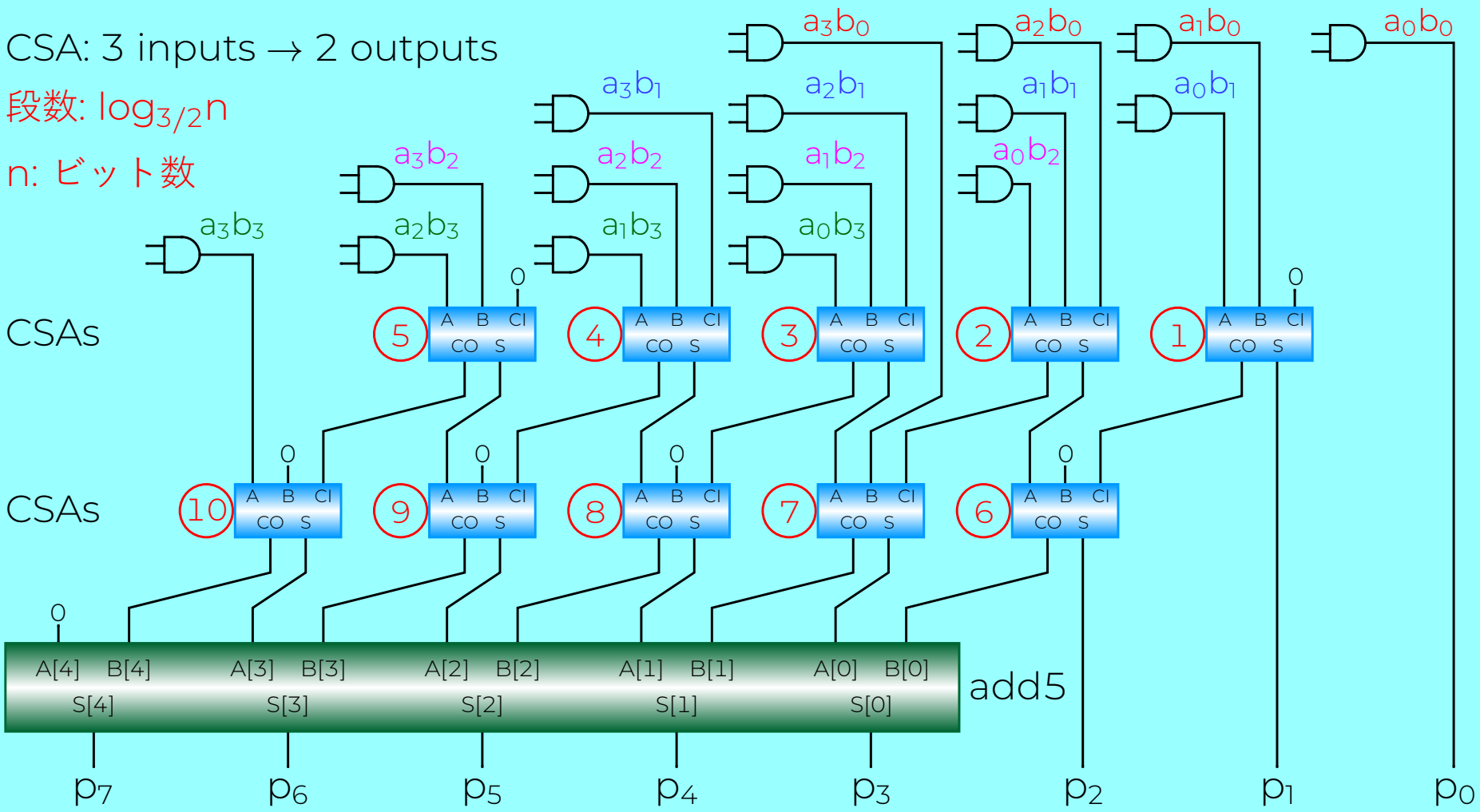
Ripple Carry Adder

桁上げ伝播加算器 RCA

(1 個)



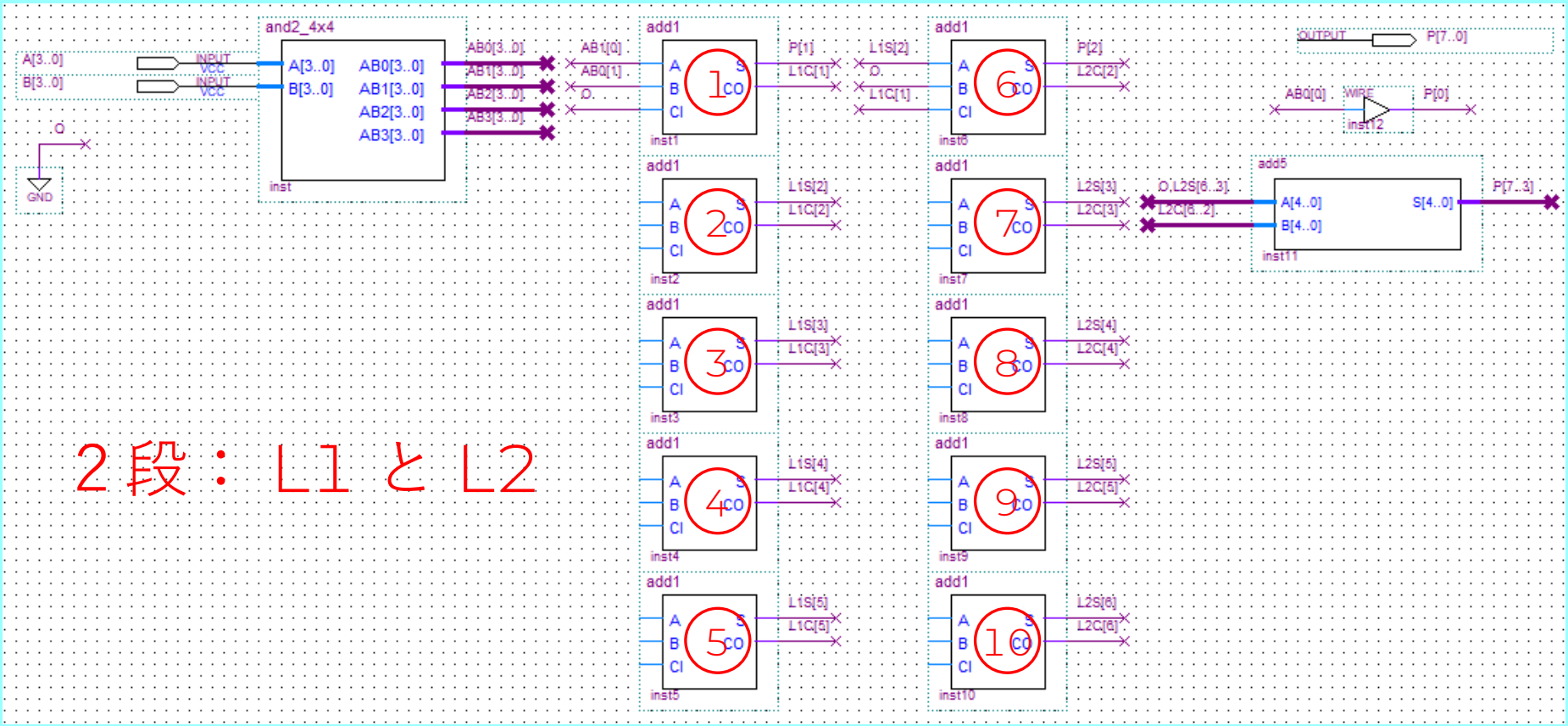
Wallace Tree (ウォレスツリー)



計算可能であれば、大量の CSA を使用して並列に計算する。

Wallace Tree の回路

mul4x4_wallace_tree.bdf

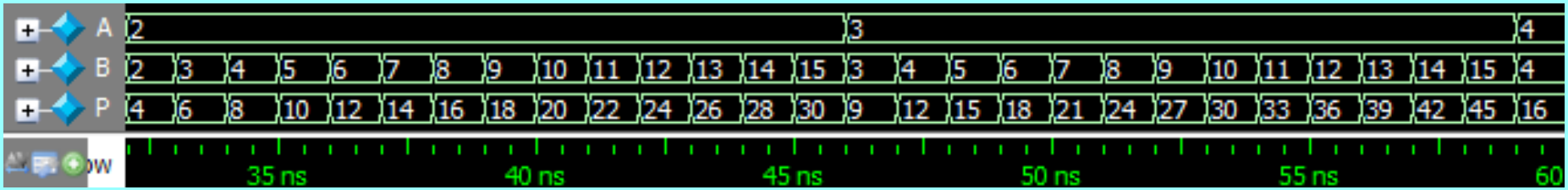


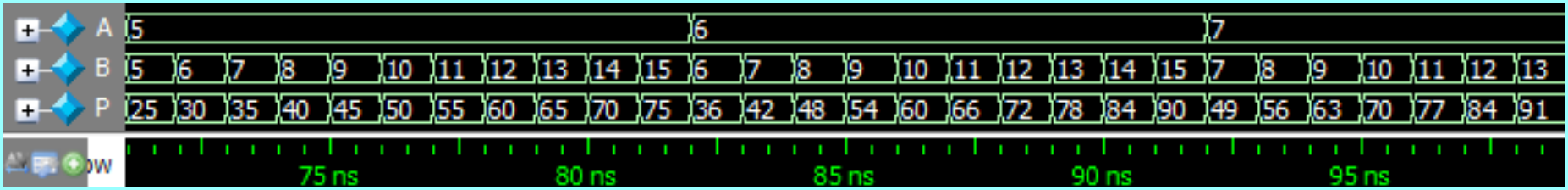
注意: 一部のワイヤーが省略された。

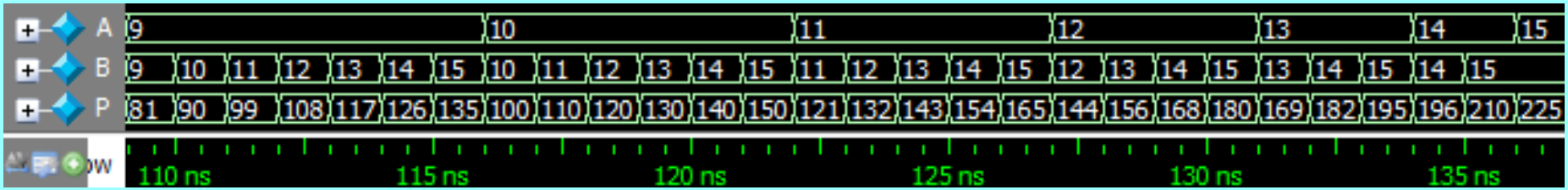
テストベンチ

```
‘timescale 1ns/1ps // unit = 1 ns; accuracy = 1 ps
module mul4x4_wallace_tree_tb;
   reg  [3:0] A,B;
   wire [7:0] P;

   mul4x4_wallace_tree i1 (.A(A), .B(B), .P(P));

   integer i,j;
   initial begin
      #0 A = 0; B = 0;
      for (i = 0; i < 16; i = i + 1) begin
         for (j = i; j < 16; j = j + 1) begin
            #1 A = i; B = j;
         end
      end
      #1 $stop;
   end
endmodule
```

Wallace Tree の波形







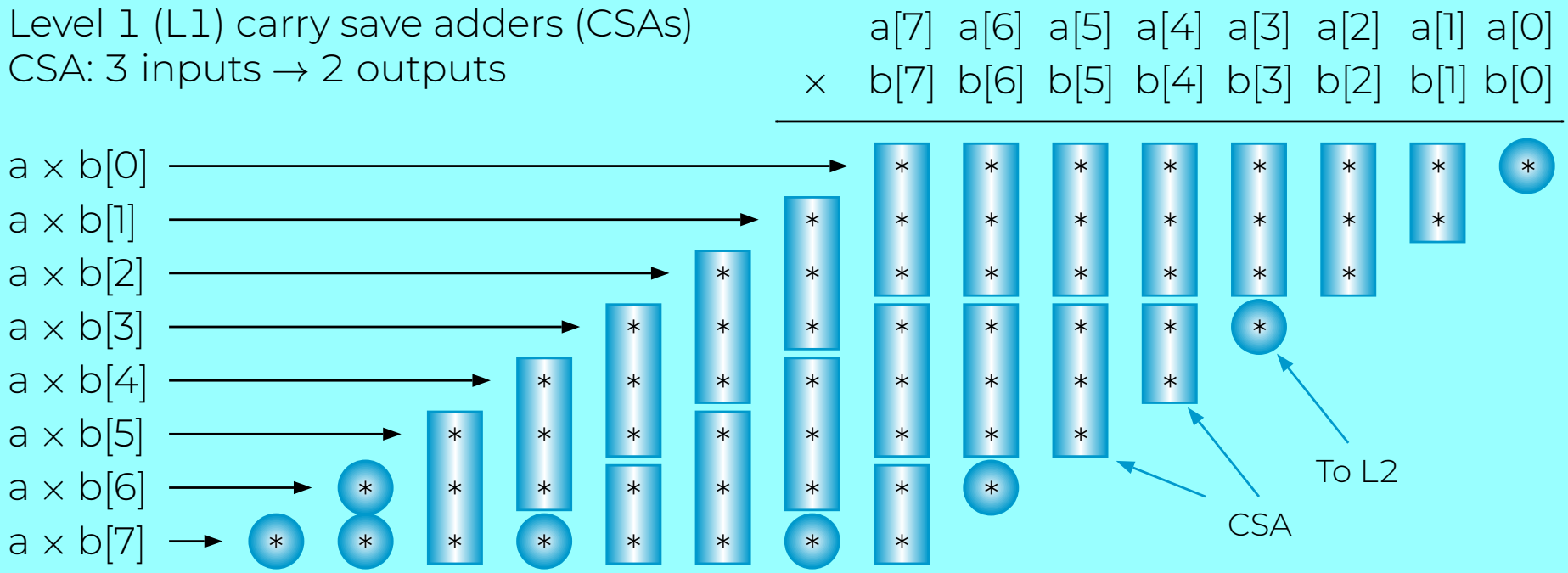
⇑

Radix: Unsigned (符号なし)

$15 \times 15 = 225$

Wallace Tree (8 ビット)

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

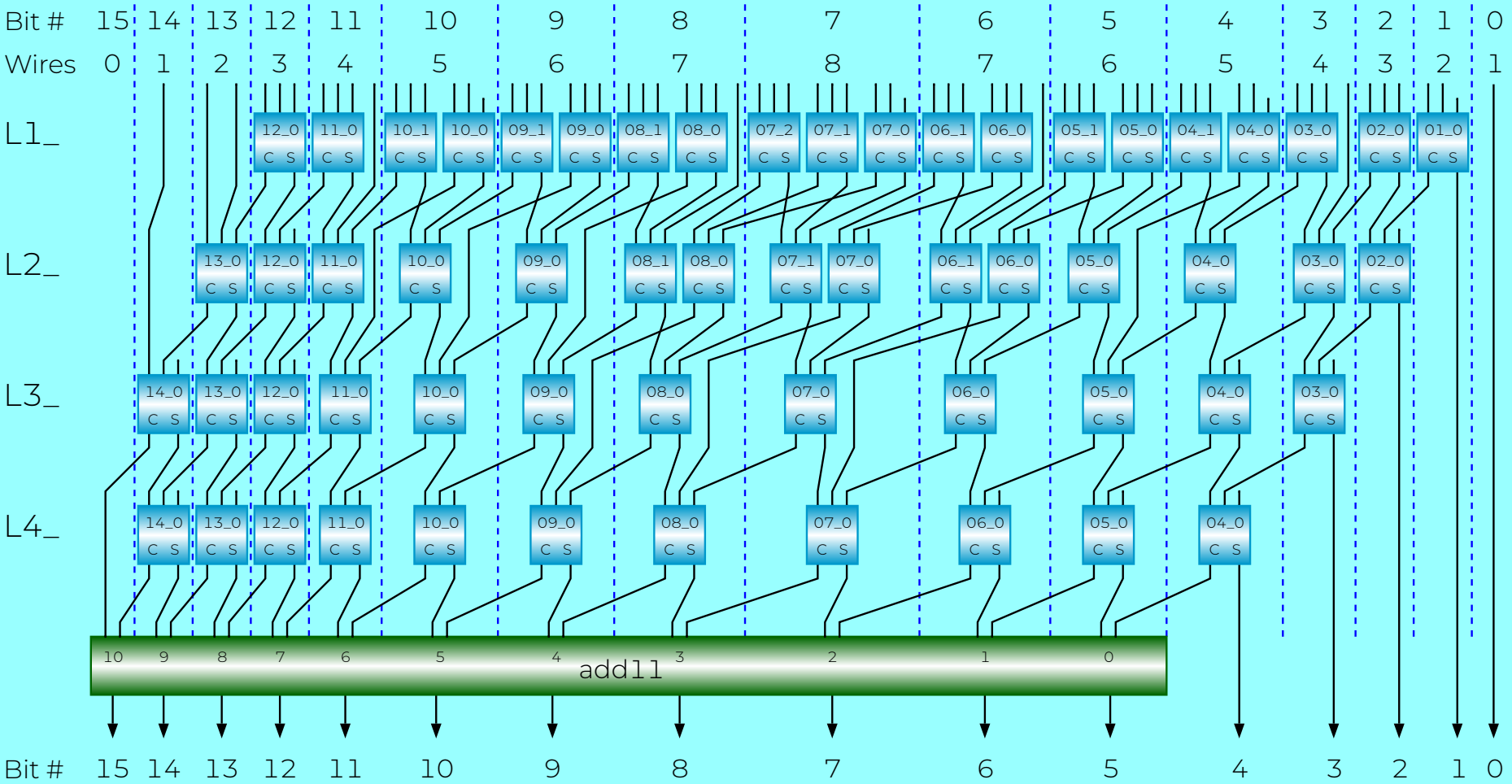


Bit number	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
# of wires (*)	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1
# of CSAs	0	0	1	1	2	2	2	3	2	2	2	1	1	1	0

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。

Wallace Tree (8 ビット)

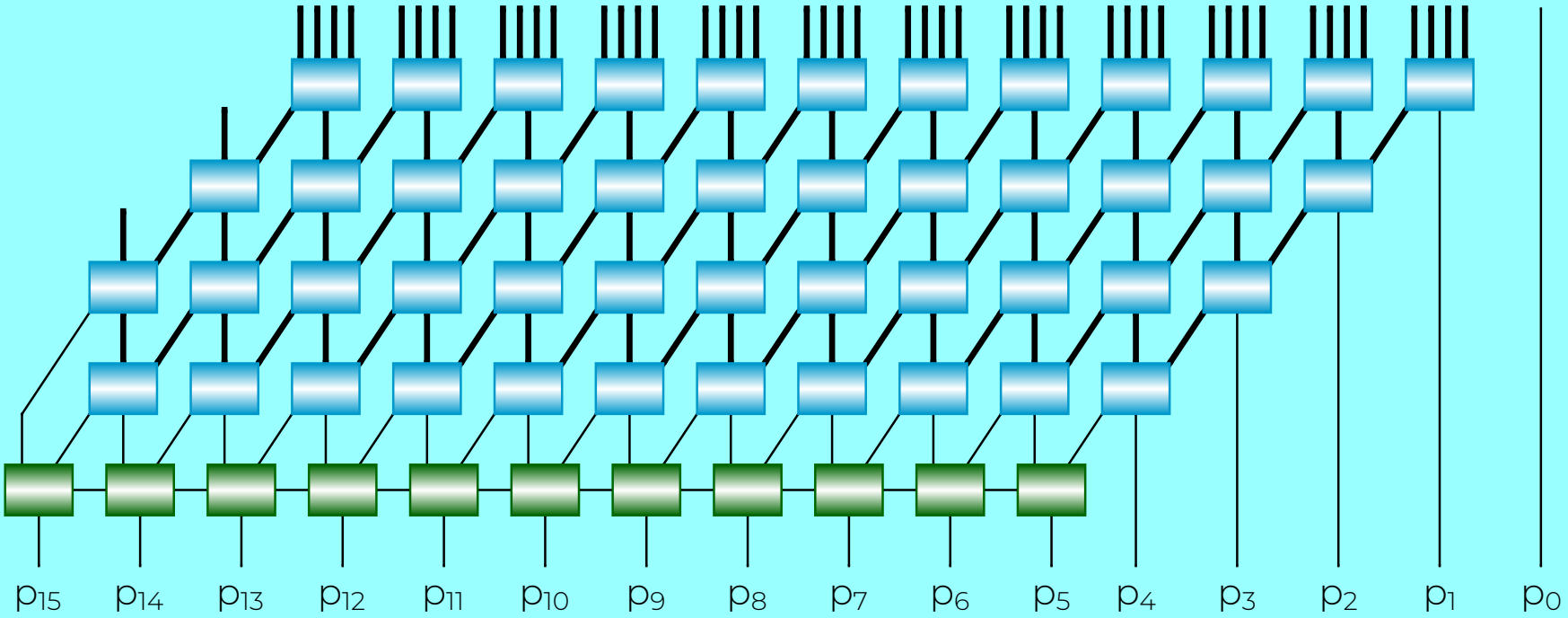


CSA: 3 inputs → 2 outputs ;　　段数 $= \log_{3/2} n$

計算可能であれば、大量の CSA を使用して並列に計算する。

Wallace Tree (8 ビット)

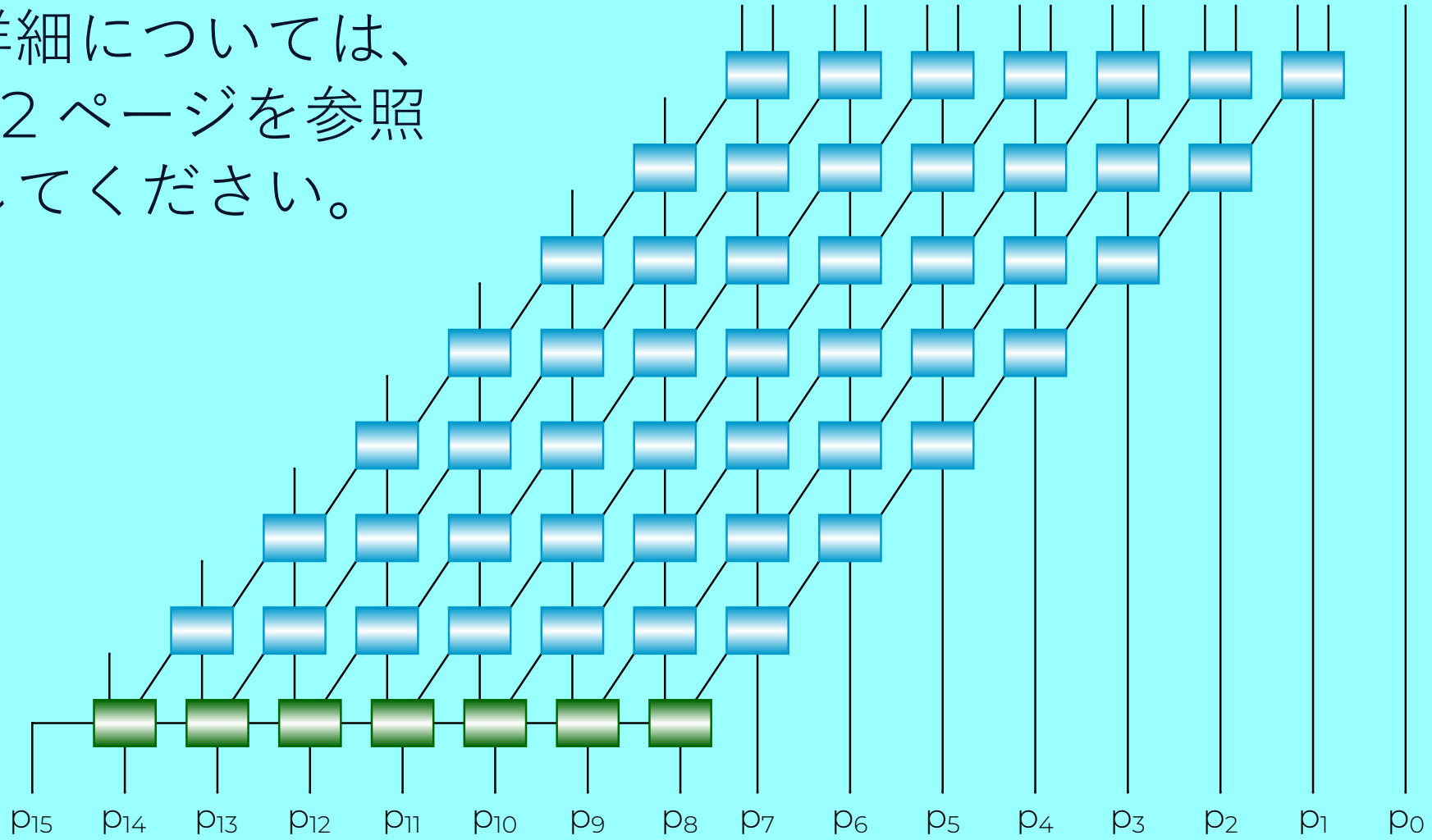
綺麗に整列されたウォレスツリーの回路図。
詳細については、前のページを参照してください。



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、
12 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

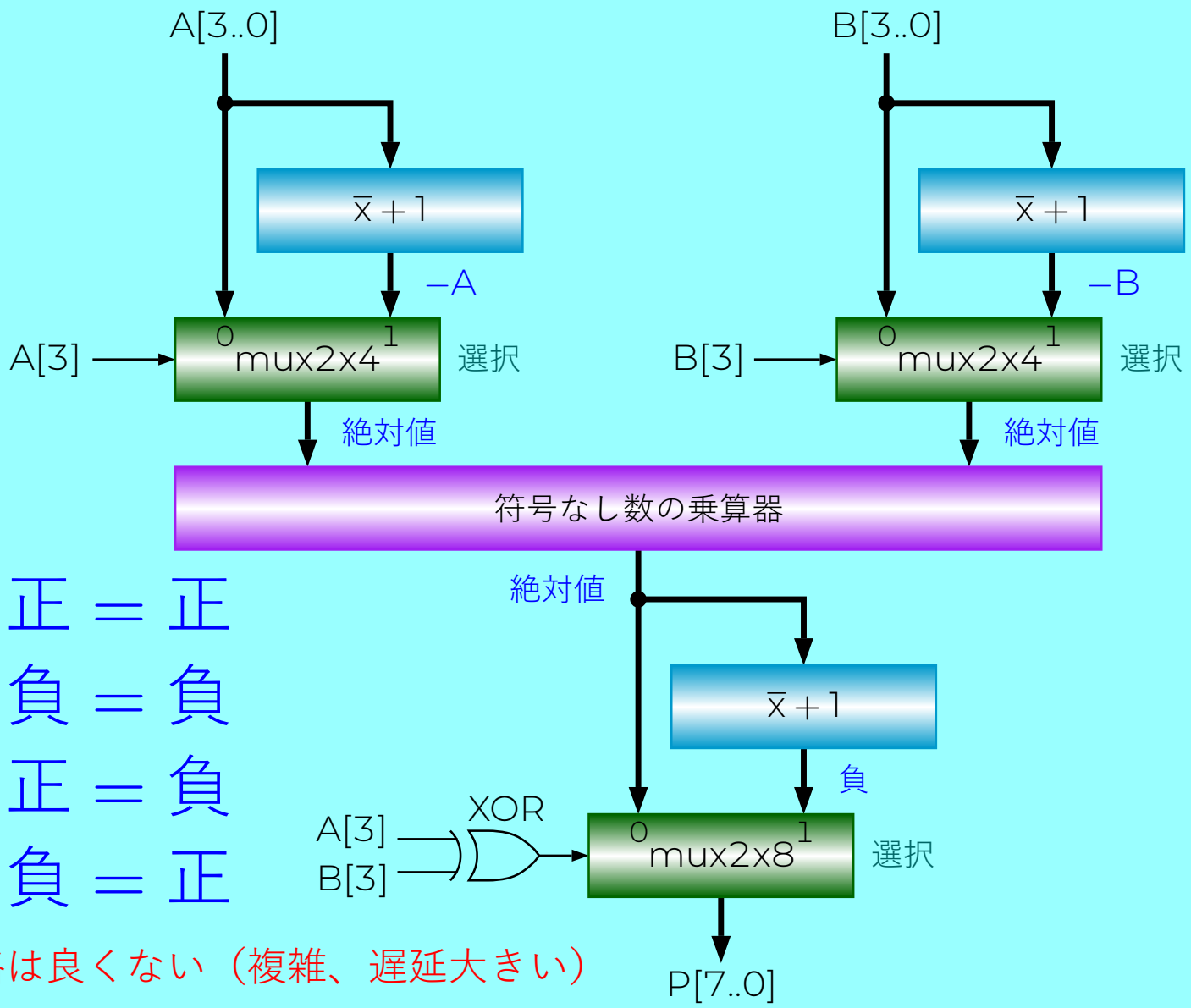
  負 × 負 = 正

↑

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

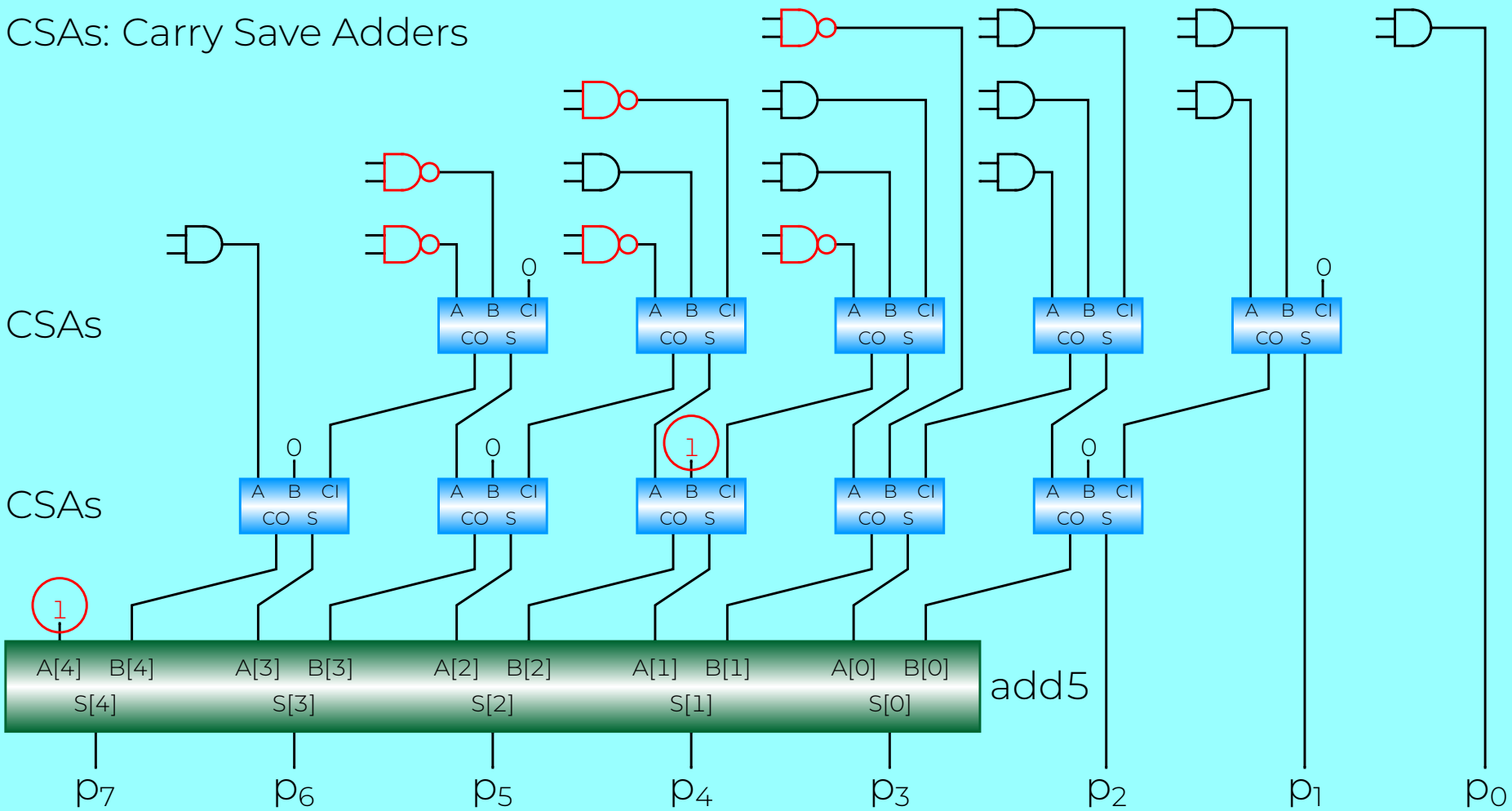
2 の補数で表現する数の掛け算

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

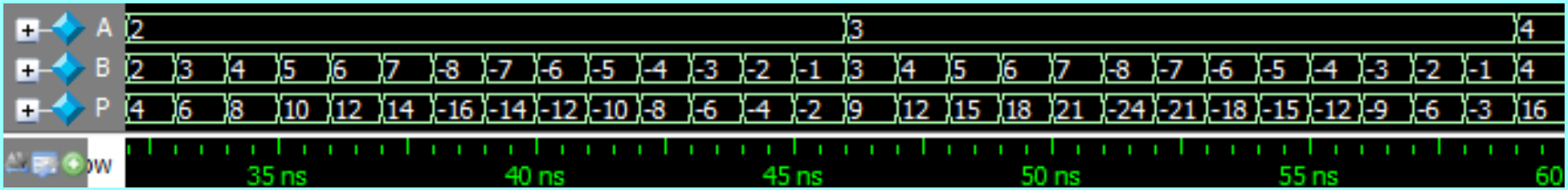


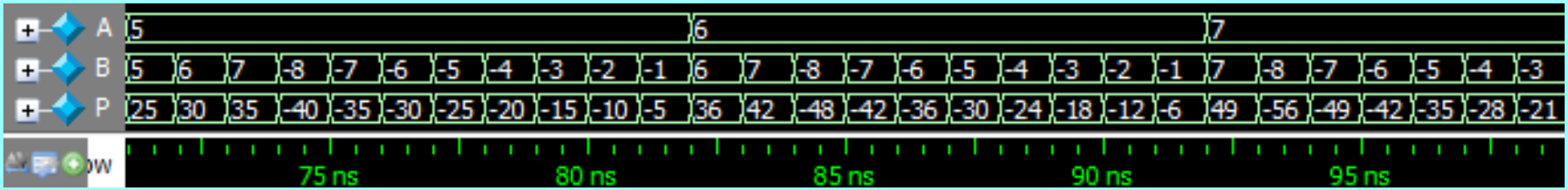
a_3b_2、a_3b_1、a_3b_0、a_2b_3、a_1b_3、a_0b_3：P6 を参照

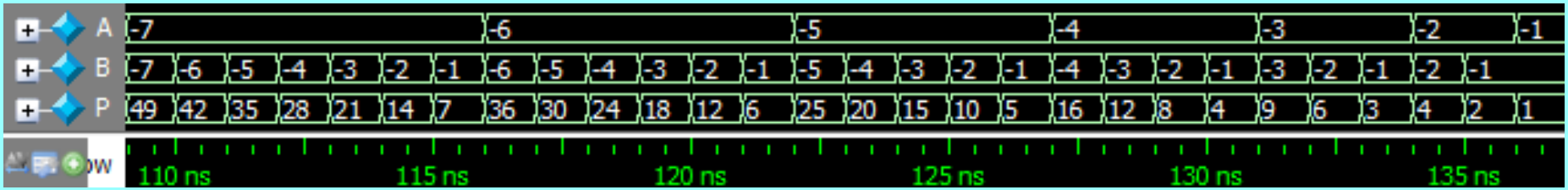
テストベンチ

```
‘timescale 1ns/1ps // unit = 1 ns; accuracy = 1 ps
module mul4x4_wallace_tree_signed_tb;
   reg  [3:0] A,B;
   wire [7:0] P;

   mul4x4_wallace_tree_signed i1 (.A(A), .B(B), .P(P));

   integer i,j;
   initial begin
      #0 A = 0; B = 0;
      for (i = 0; i < 16; i = i + 1) begin
         for (j = i; j < 16; j = j + 1) begin
            #1 A = i; B = j;
         end
      end
      #1 $stop;
   end
endmodule
```

2 の補数のウォレスツリーの波形



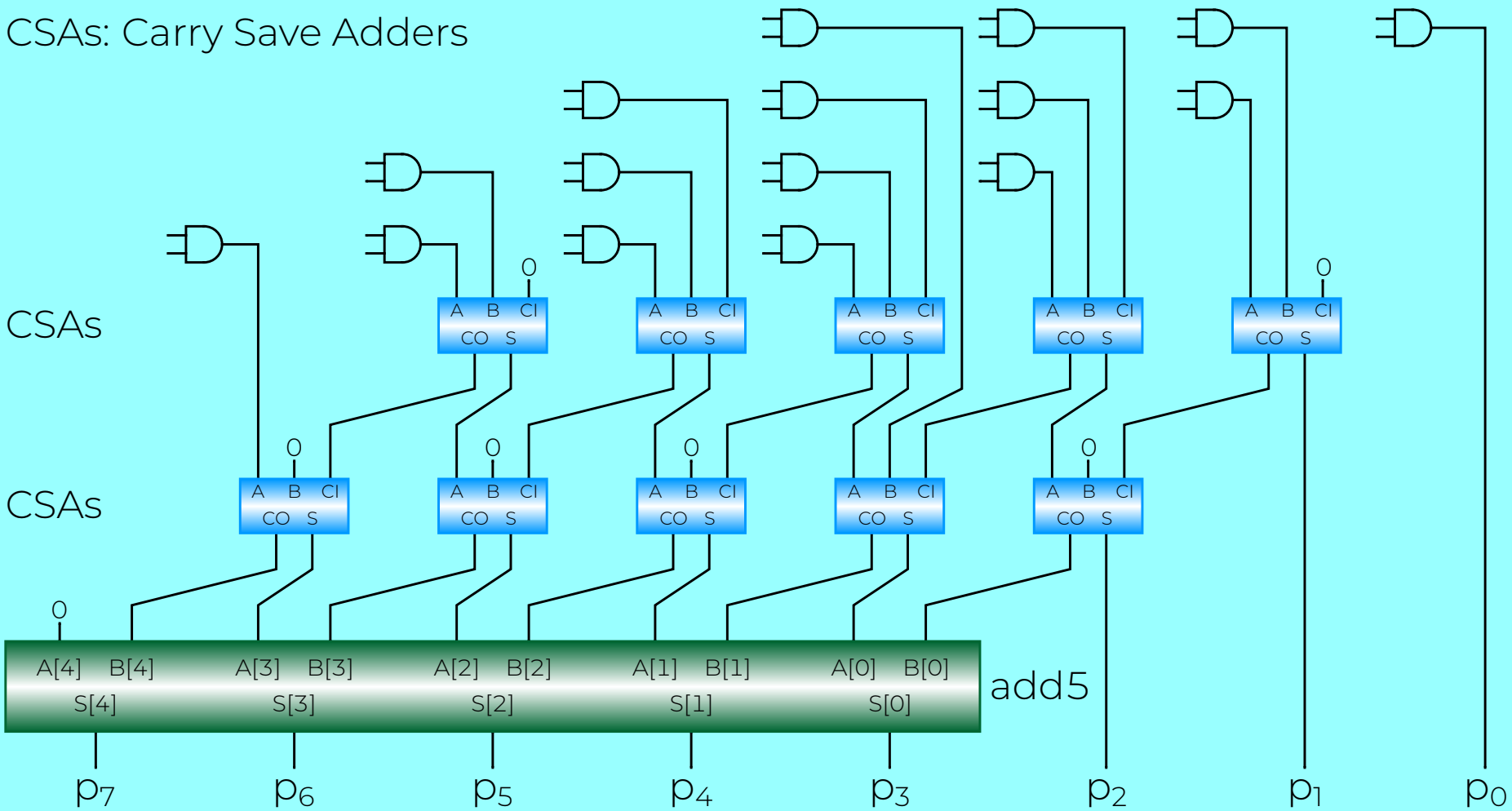




⇑

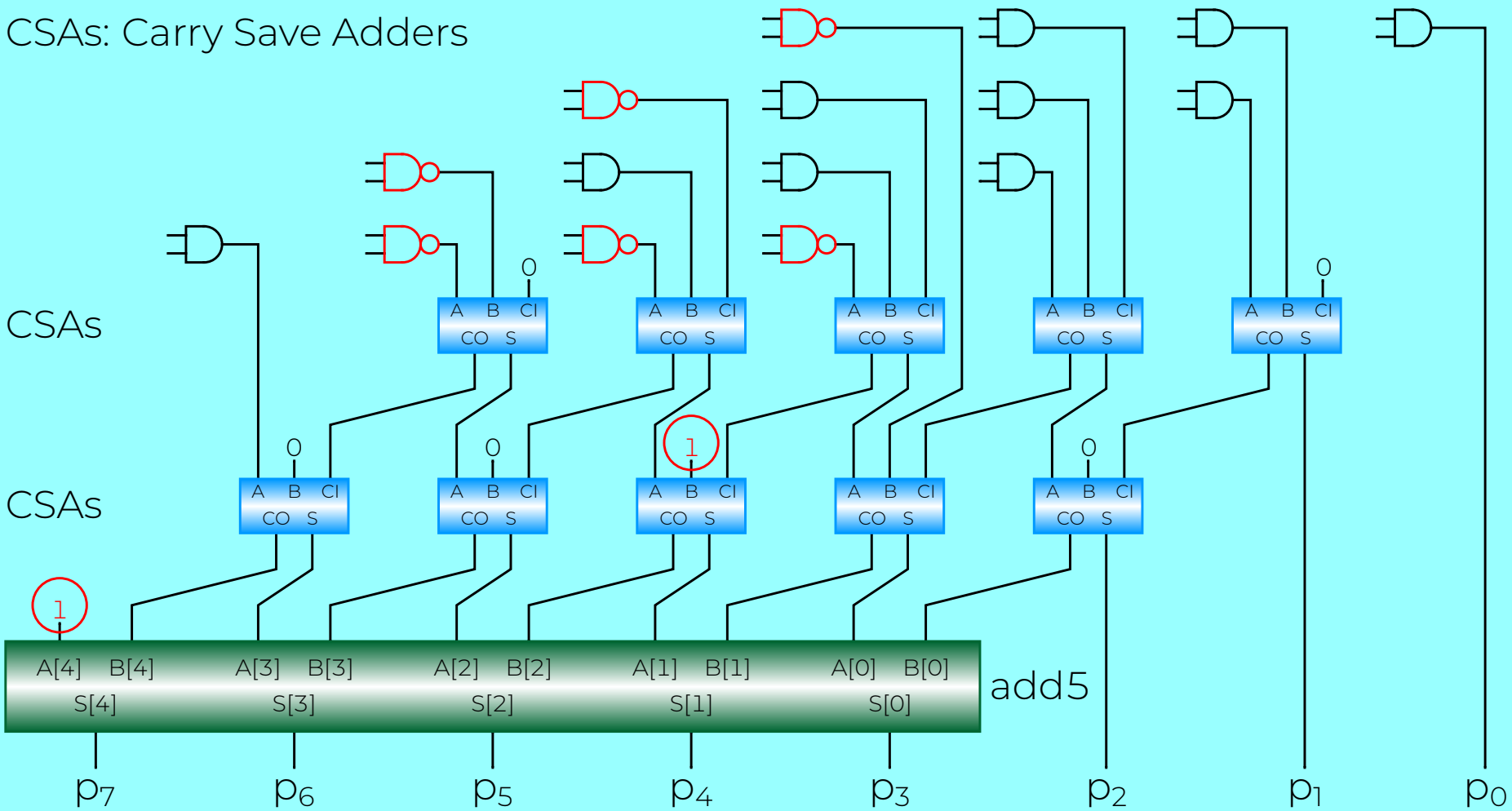
Radix: Decimal (符号付き 10 進) $(-1) \times (-1) = +1$

符号なしのウォレスツリーの回路



a_3b_2、a_3b_1、a_3b_0、a_2b_3、a_1b_3、a_0b_3：P6 を参照

2 の補数のウォレスツリーの回路



a_3b_2、a_3b_1、a_3b_0、a_2b_3、a_1b_3、a_0b_3：P6 を参照

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 CSAs 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

プロジェクト名は mul4x4_wallace_tree にすること。
テストベンチ mul4x4_wallace_tree_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。(注意：gnd = 0; vcc = 1) プロジェクト名は mul4x4_wallace_tree_signed にすること。テストベンチ mul4x4_wallace_tree_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

プロジェクト名は mul8x8_wallace_tree 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。(注意：gnd = 0; vcc = 1) プロジェクト名は mul8x8_wallace_tree_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

Verilog HDL による課題の実装

5 ビット加算器の設計（第 6 回資料 P14 を参照）

```
module add5b (A, B, S);
    input  [4:0] A, B;
    output [4:0] S;
    wire   [4:0] c; // internal wires
    add1b i0 (A[0], B[0], 1'b0, c[0], S[0]); // full adder 0
    add1b i1 (A[1], B[1], c[0], c[1], S[1]); // full adder 1
    add1b i2 (A[2], B[2], c[1], c[2], S[2]); // full adder 2
    add1b i3 (A[3], B[3], c[2], c[3], S[3]); // full adder 3
    add1b i4 (A[4], B[4], c[3], c[4], S[4]); // full adder 3
endmodule
```

発展：自由練習

Verilog HDL による課題の実装

AND2 マトリクス（行列）（P8 を参照）

```
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]}};
    assign AB1 = A & {4{B[1]}};
    assign AB2 = A & {4{B[2]}};
    assign AB3 = A & {4{B[3]}};
endmodule
```

発展：自由練習

Verilog HDL による課題の実装

符号なしのウォレスツリーの回路（P13、P14 を参照）

```
module wallace4x4 (A, B, P);
    input  [3:0] A, B;
    output [7:0] P;
    wire   [3:0] AB0, AB1, AB2, AB3;
    wire   [5:1] L1C;
    wire   [5:2] L1S;
    wire   [6:2] L2C;
    wire   [6:3] L2S;
    and4x4 i0 (A, B, AB0, AB1, AB2, AB3);
    assign P[0] = AB0[0];
    add1b i1 (AB1[0], AB0[1],   1'b0, L1C[1],   P[1]); //  1 L1
```

発展：自由練習

Verilog HDL による課題の実装

符号なしのウォレスツリーの回路（P13、P14 を参照）

```
    add1b i2 (AB2[0], AB1[1], AB0[2], L1C[2], L1S[2]); //  2 L1
    add1b i3 (      ,       ,       , L1C[3], L1S[3]); //  3 L1
    add1b i4 (      ,       ,       , L1C[4], L1S[4]); //  4 L1
    add1b i5 (      ,       ,       , L1C[5], L1S[5]); //  5 L1
    add1b i6 (L1S[2], L1C[1],   1'b0, L2C[2],   P[2]); //  6 L2
    add1b i7 (      ,       ,       , L2C[3], L2S[3]); //  7 L2
    add1b i8 (      ,       ,       , L2C[4], L2S[4]); //  8 L2
    add1b i9 (      ,       ,       , L2C[5], L2S[5]); //  9 L2
    add1b ia (      ,       ,       , L2C[6], L2S[6]); // 10 L2
    add5b ib ({1'b0,L2S[6:3]}, L2C[6:2], P[7:3]);      // add5b
endmodule
```

上記コードを完成し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：

$$S_3S_2S_1S_0 = T_3T_2T_1T_0$$