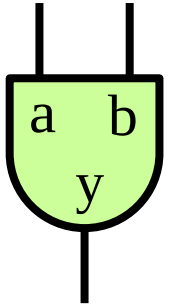
第4回 ハードウェア記述言語
～乗算器、除算器～

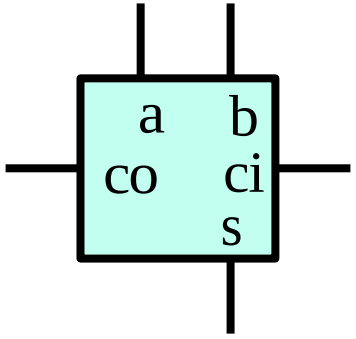
中野 秀洋

乗算器の設計

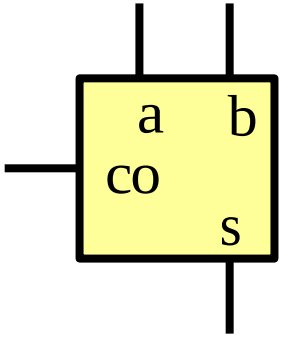
```
  11000100
x 11000001
----------
  11000100
 00000000 0
00000000 00
00000000 000
00000000 0000
00000000 00000
11000100 000000
11000100 0000000
----------------
1001001111000100
```



AND回路



全加算器



半加算器

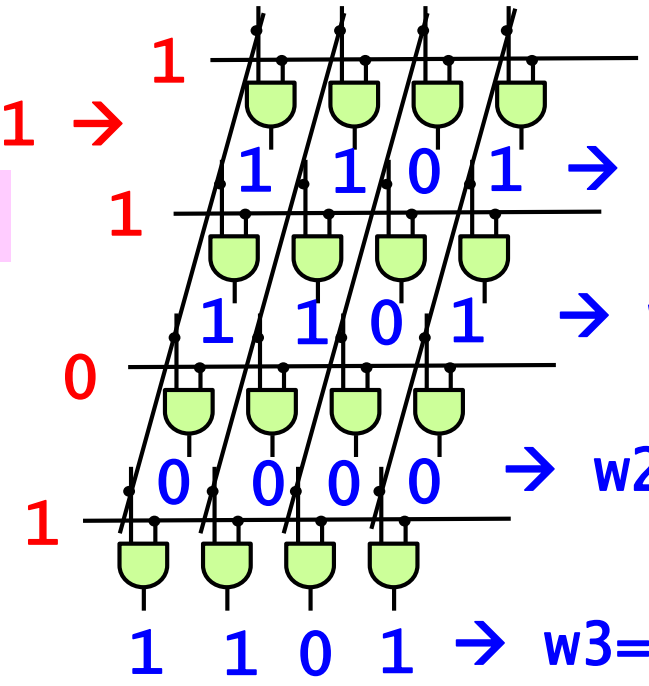
単一サイクル乗算器

被乗数

a=1101 → 1 1 0 1

b=1011 →

乗数



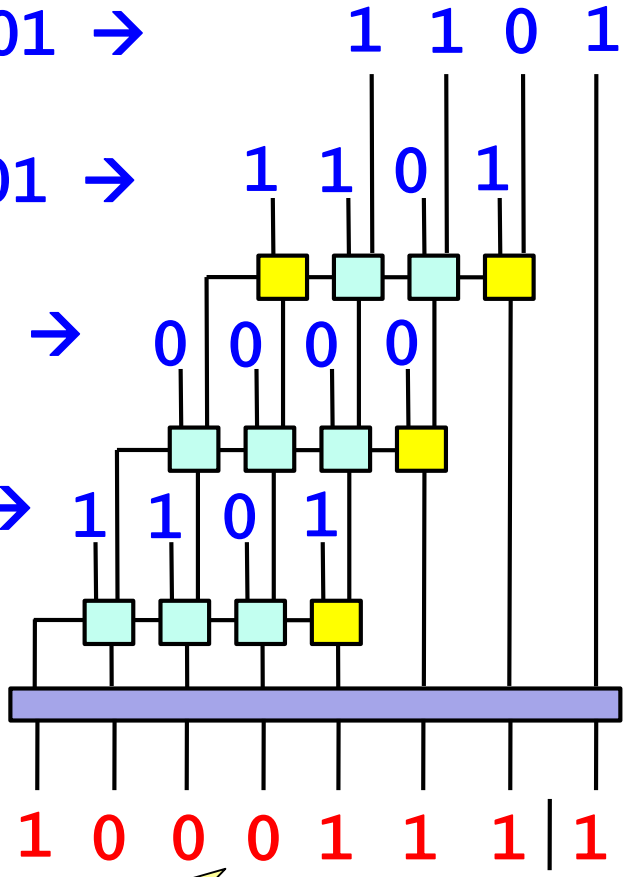
ANDで部分積

w0=1101 →

w1=1101 →

w2=0000 →

w3=1101 →



部分積の和

→ p=10001111

積

```
  1101
x 1011
------
  1101
 11010
 00000
1101000
--------
10001111
```

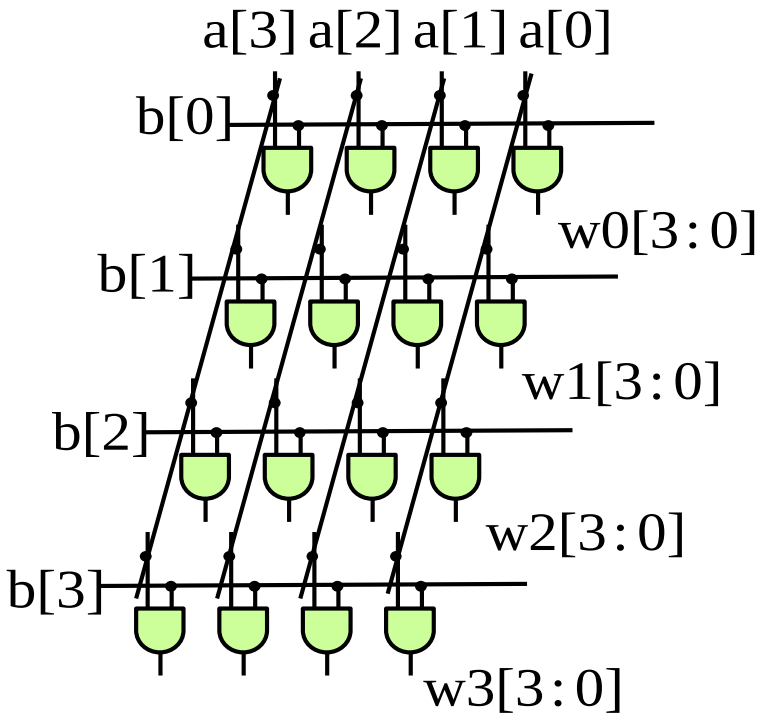
```
  1101
+ 11010
--------
 100111

 100111
+ 00000
--------
 0100111

 0100111
+1101000
--------
10001111
```

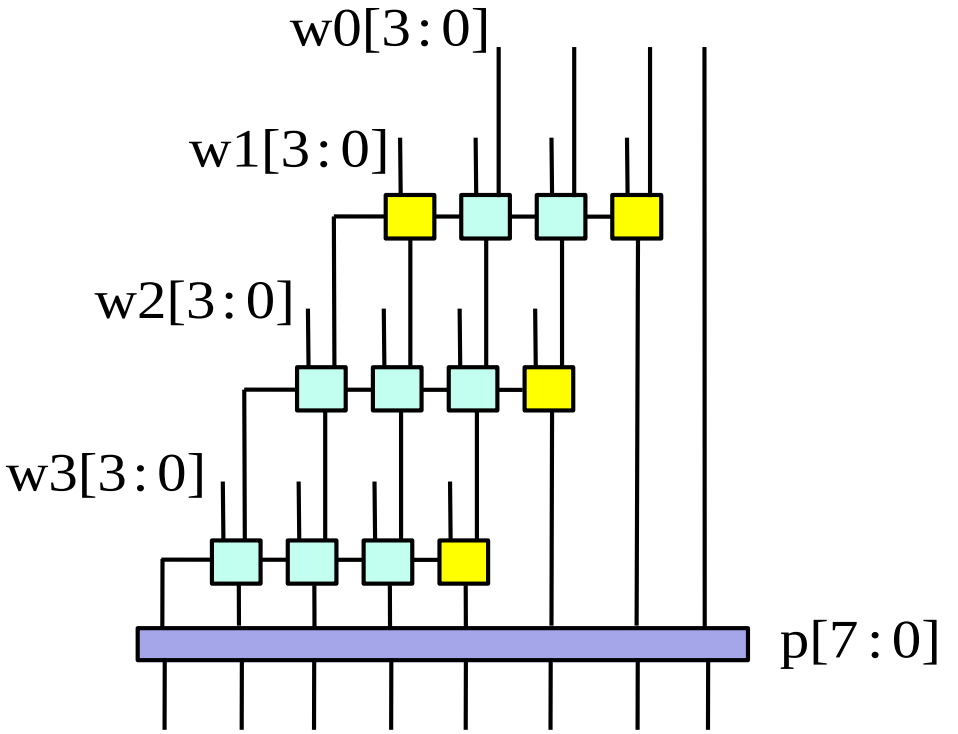
4bit単一サイクル乗算器

```
module mul_s(clk, a, b, p);

    input          clk;
    input  [3:0] a, b;
    output [7:0] p;
    reg    [7:0] p;
    wire   [3:0] w0, w1, w2, w3;

    assign w0 = a & {4{b[0]}};
    assign w1 = a & {4{b[1]}};
    assign w2 = a & {4{b[2]}};
    assign w3 = a & {4{b[3]}};

    always@(posedge clk) begin
        p <=  w0
            + {w1,1'b0}
            + {w2,2'b00}
            + {w3,3'b000};
    end
endmodule
```

{ n{w} }
信号線wをn本コピー

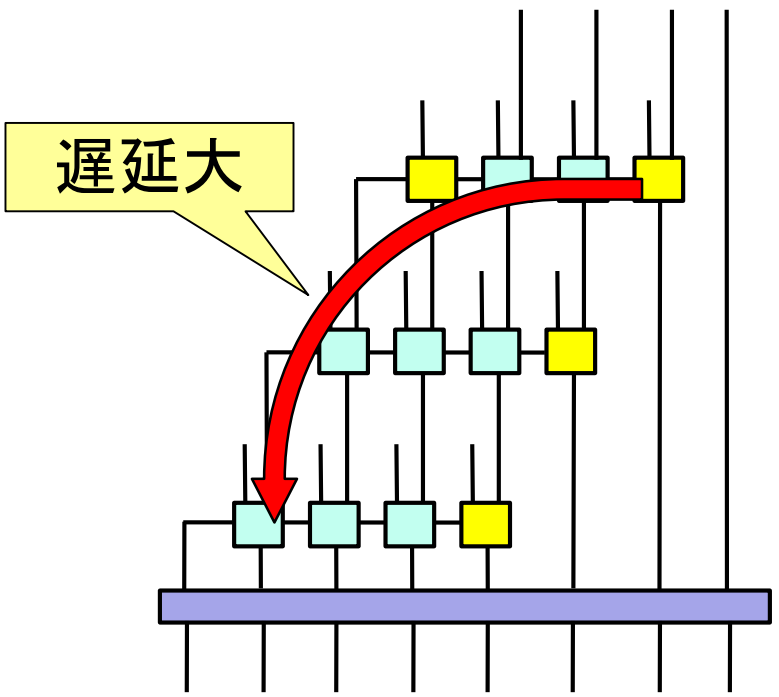
```
  1101
x 1011
------
  1101
 11010
 00000
1101000
--------
10001111
```

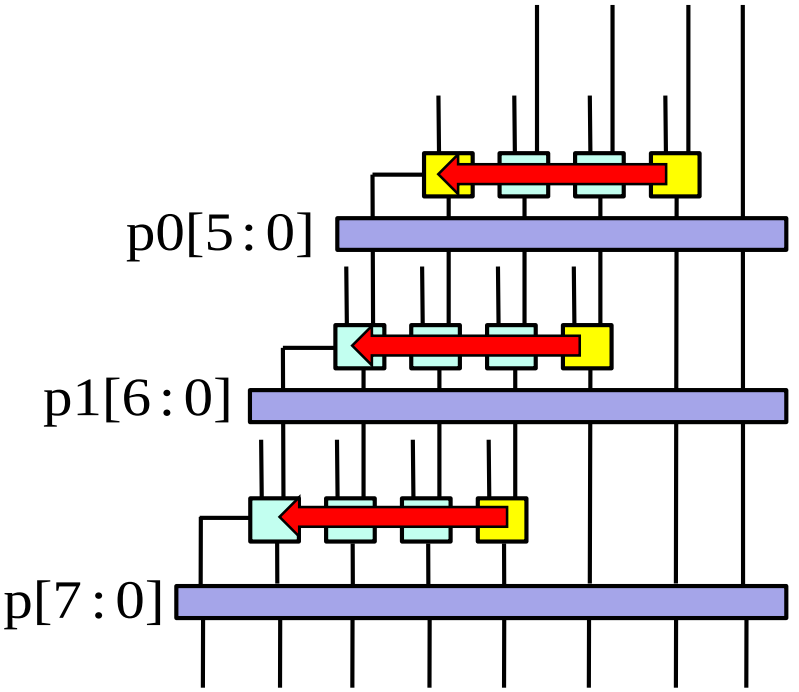




マルチサイクル乗算器

各段の加算結果をレジスタに格納
マルチサイクルで動作
遅延小 ➔ 動作周波数高



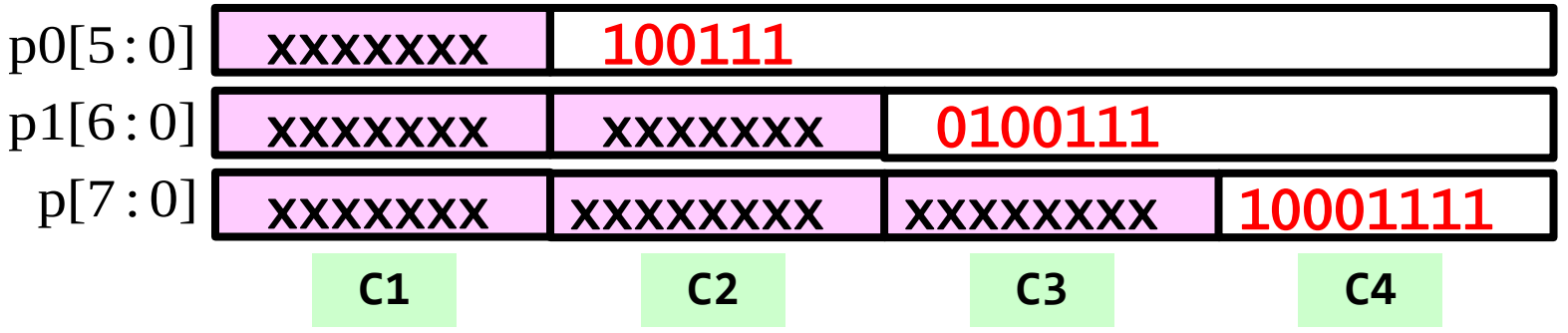


C1
```
  1101
+ 11010
------
 100111
```

C2
```
  100111
+ 000000
--------
 0100111
```

C3
```
  0100111
+ 1101000
---------
 10001111
```

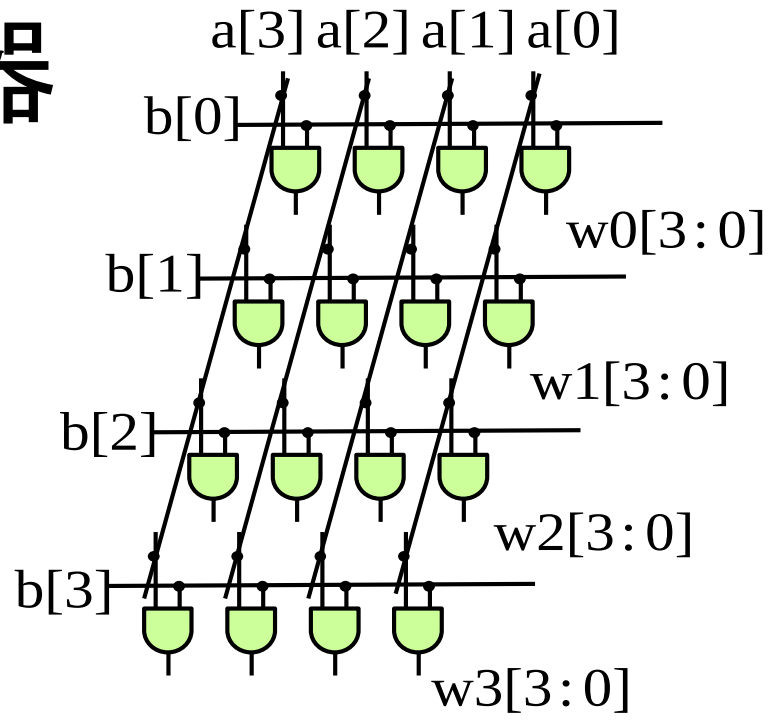
C4 10001111

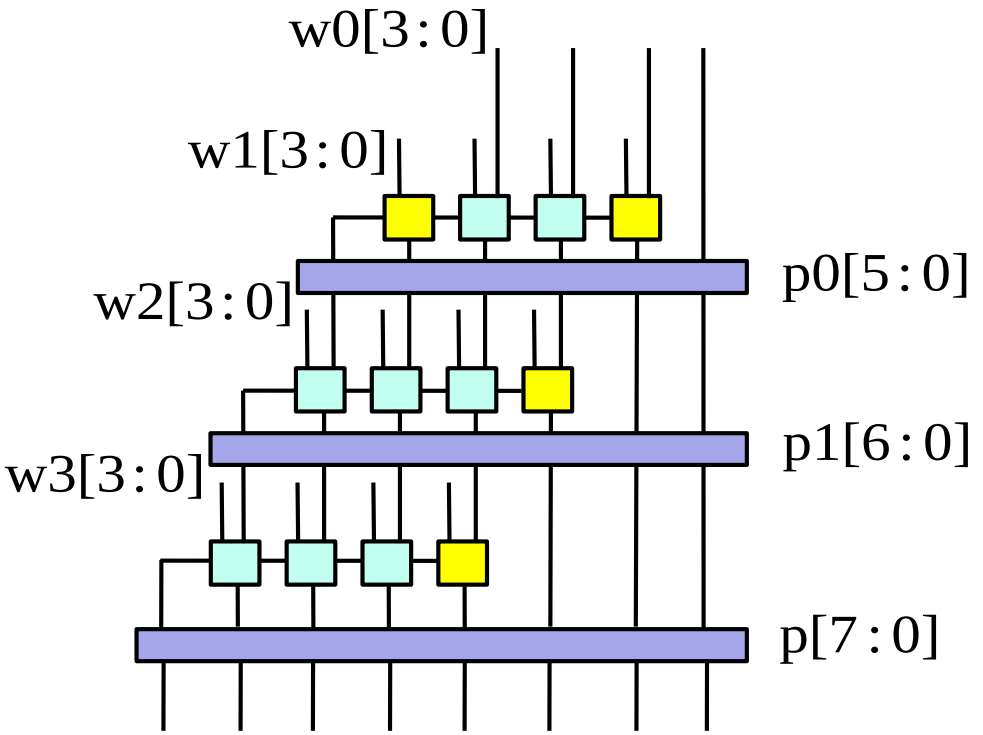


4bitマルチサイクル乗算器

```
module mul_m(clk, a, b, p);

    input          clk;
    input  [3:0] a, b;
    output [7:0] p;
    reg    [5:0] p0;
    reg    [6:0] p1;
    reg    [7:0] p;
    wire   [3:0] w0, w1, w2, w3;

    assign w0 = a & {4{b[0]}};
    assign w1 = a & {4{b[1]}};
    assign w2 = a & {4{b[2]}};
    assign w3 = a & {4{b[3]}};

    always@(posedge clk) begin
        p0 <= w0 + {w1,1'b0};
    end
    always@(posedge clk) begin
        p1 <= p0 + {w2,2'b00};
    end
    always@(posedge clk) begin
        p  <= p1 + {w3,3'b000};
    end
endmodule
```

```
   1101
x  1011
------
   1101
  11010
 000000
1101000
--------
10001111
```

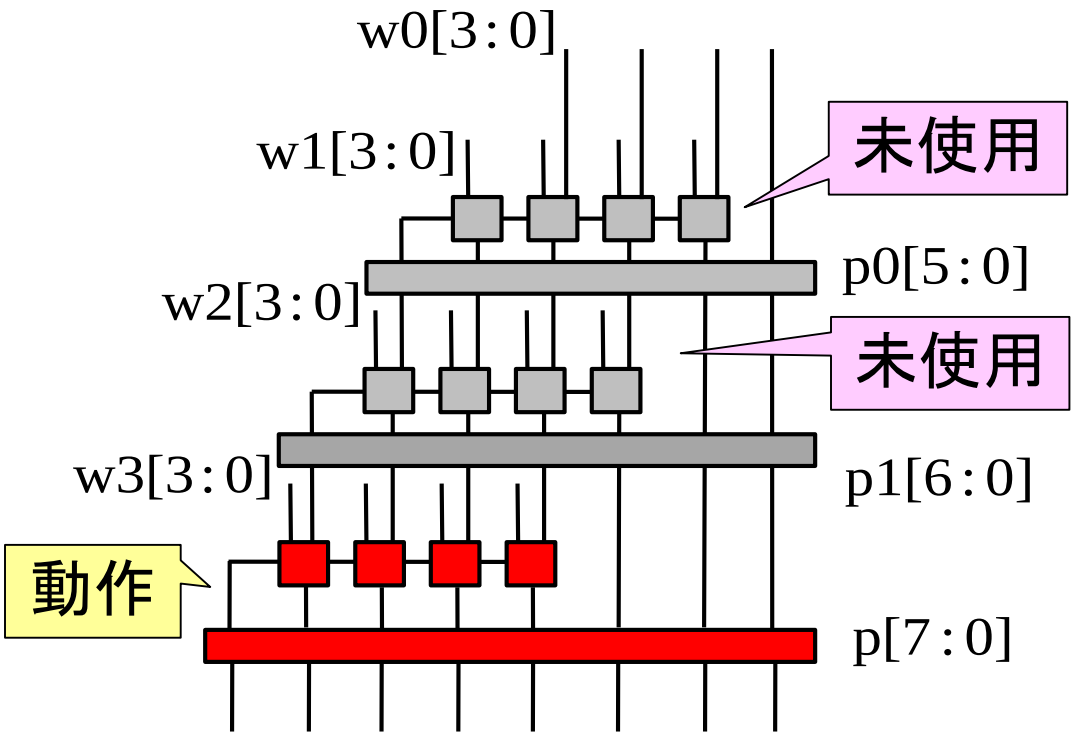




パイプライン乗算器

後段の回路の動作時
前段の回路は未使用 ➔ 無駄

未使用回路を後続の演算で利用 ➔ 高速化




```
  1101
+ 11010
------
 100111

 100111
+ 000000
------
0100111

 0100111
+1101000
--------
10001111
```

```
   1101
x  1011
-------
   1101
  1101 0
 0000 00
1101 000
--------
10001111
```

パイプライン乗算器

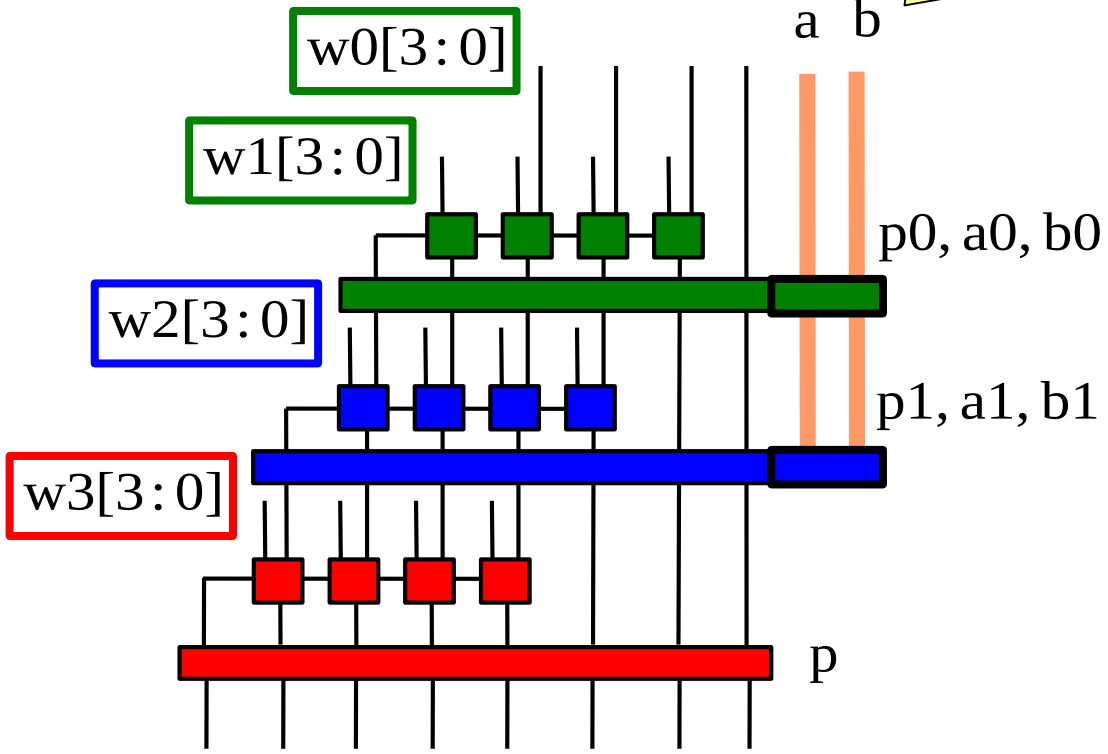
パイプライン

複数の演算をオーバーラップさせて実行

先行する演算の完了を待たずに

後続の演算を開始可能 ➔ 高速化

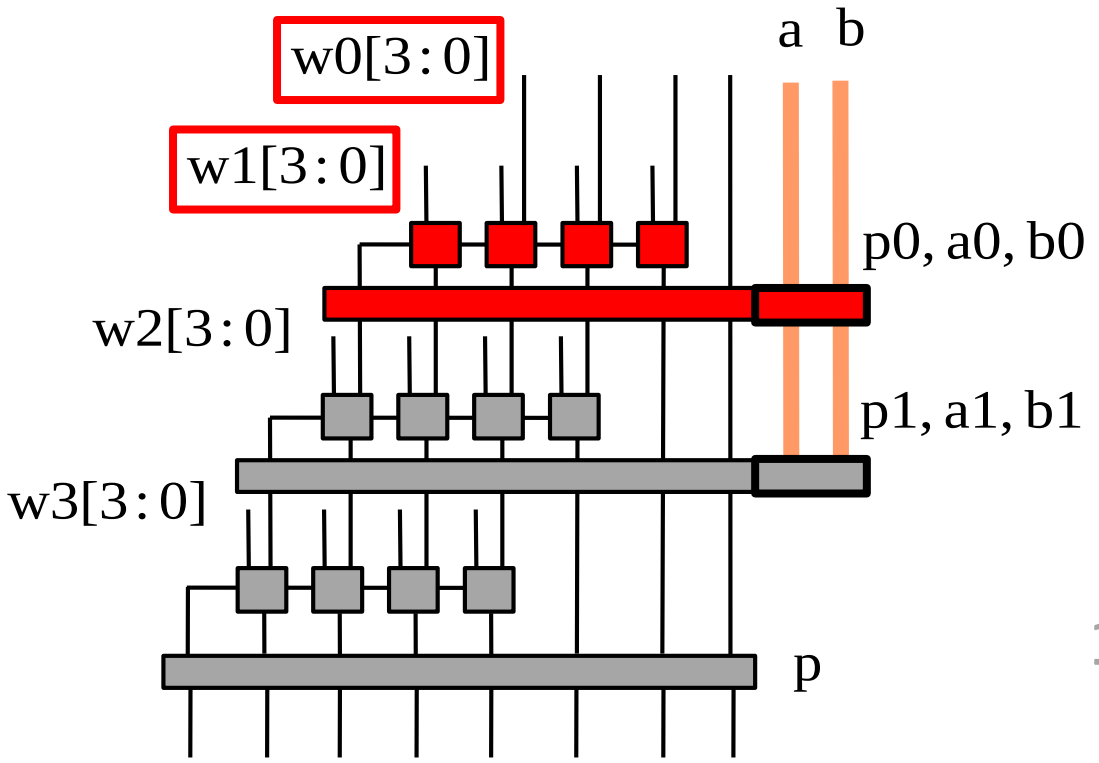
被乗数・乗数もパイプラインへ
部分積の計算で必要




```
  0010
+ 00100
 000110

 010010
+100100
0110110

 0100111
+1101000
10001111
```

```
   0010
x  0011
   0010
  00100
 000000
0000000
00000110

   1001
x  0110
   0000
  10010
 100100
0000000
00110110

   1101
x  1011
   1101
  11010
 000000
1101000
10001111
```

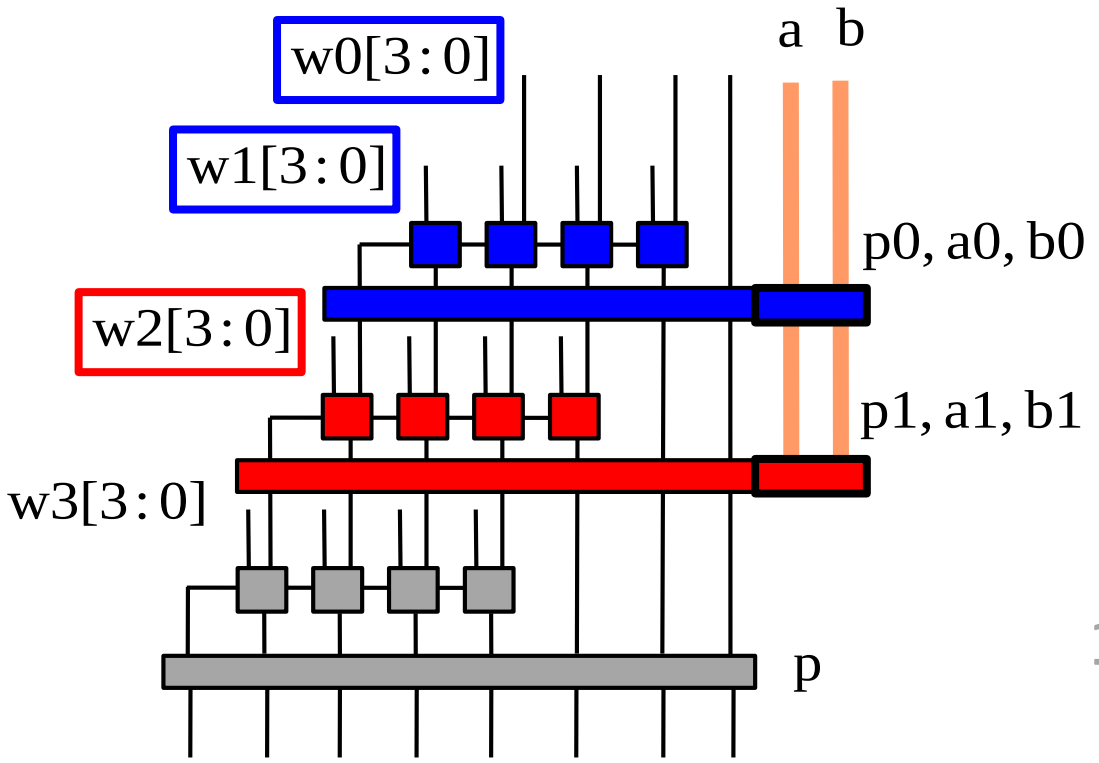
パイプライン乗算器（C1→C2）



1101	1001	0010
x 1011	x 0110	x 0011
1101	0000	0010
1101**0**	10010	00100
000000	100100	000000
1101000	0000000	0000000
10001111	00110110	00000110

a0[3:0]	1101	xxxx	xxxx	xxxx	xxxx
a1[3:0]	xxxx	xxxx	xxxx	xxxx	xxxx
b0[3:0]	1011	xxxx	xxxx	xxxx	xxxx
b1[3:0]	xxxx	xxxx	xxxx	xxxx	xxxx
p0[5:0]	100111	xxxxxx	xxxxxx	xxxxxx	xxxxx
p1[6:0]	xxxxxxx	xxxxxxx	xxxxxxx	xxxxxxx	xxxxxxx
p[7:0]	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx

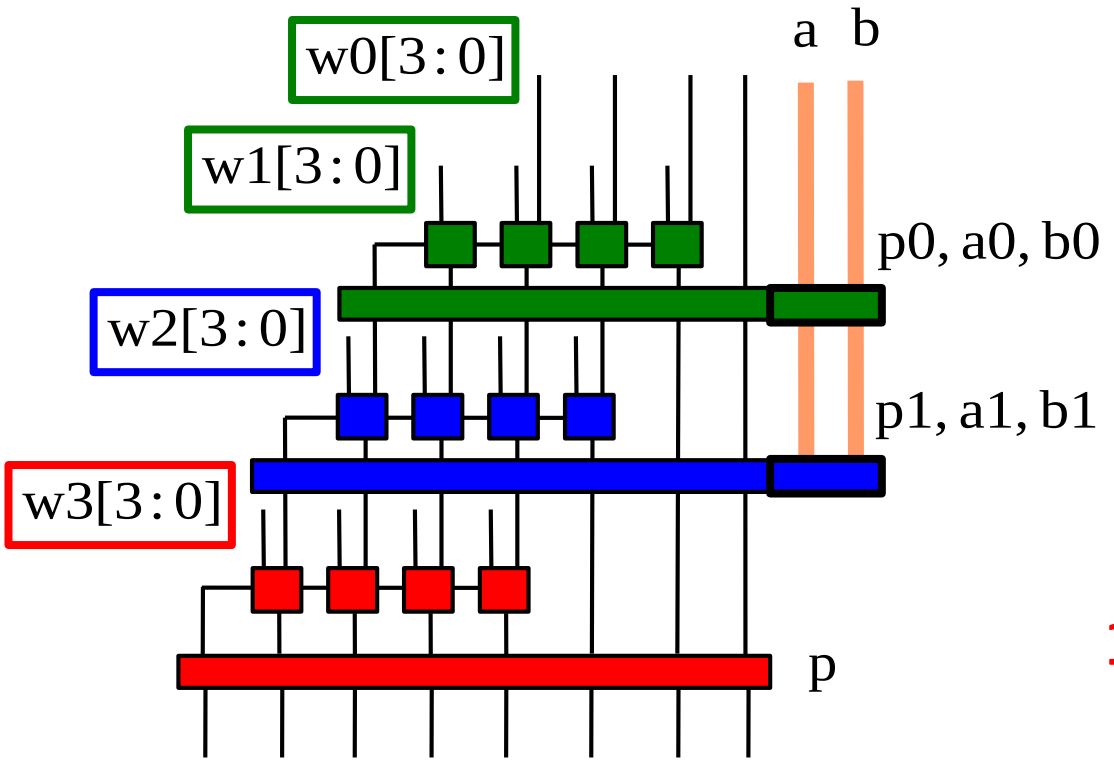
パイプライン乗算器（C2→C3）



1101	1001	0010
x 1011	x 0110	x 0011
1101	0000	0010
11010	10010	00100
000000	100100	000000
1101000	0000000	0000000
10001111	00110110	00000110

a0[3:0]	1101	1001	xxxx	xxxx	xxxx
a1[3:0]	xxxx	1101	xxxx	xxxx	xxxx
b0[3:0]	1011	0110	xxxx	xxxx	xxxx
b1[3:0]	xxxx	1011	xxxx	xxxx	xxxx
p0[5:0]	100111	010010	xxxxxx	xxxxxx	xxxxx
p1[6:0]	xxxxxxx	0100111	xxxxxxx	xxxxxxx	xxxxxxx
p[7:0]	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx

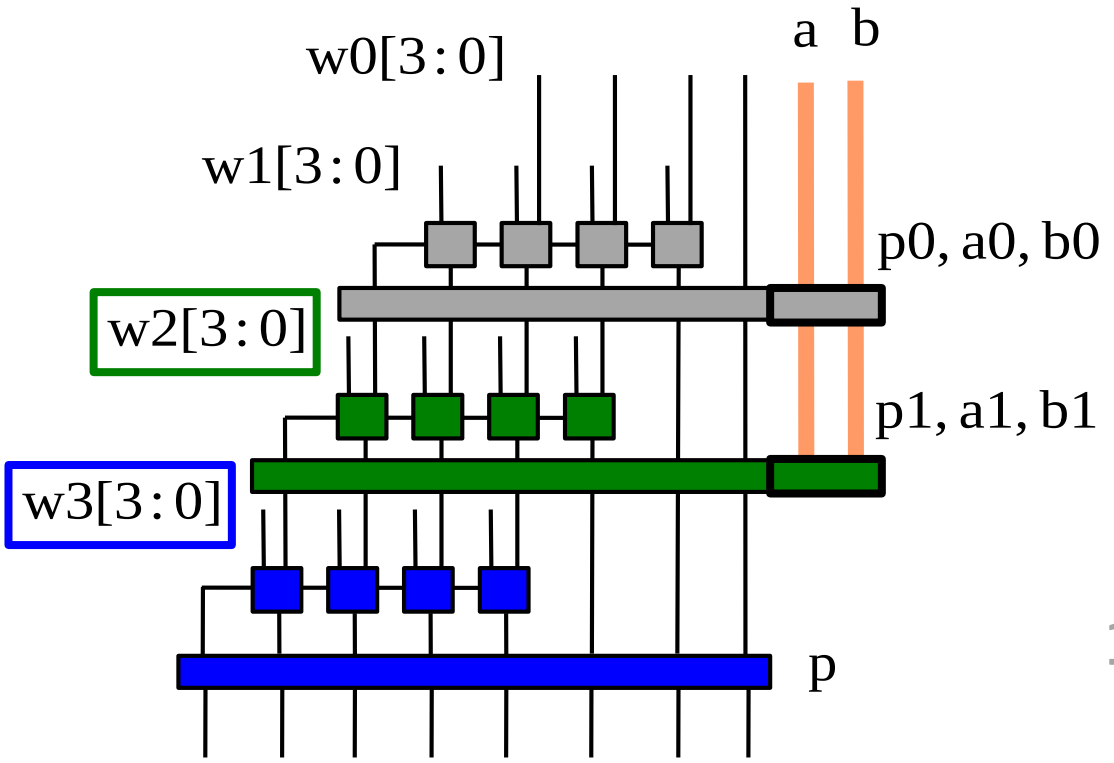
パイプライン乗算器（C3→C4）



1101	1001	0010
x 1011	x 0110	x 0011
1101	0000	0010
11010	10010	00100
000000	100100	000000
1101000	0000000	0000000
10001111	00110110	00000110

a0[3:0]	1101	1001	0010	xxxx	xxxx
a1[3:0]	xxxx	1101	1001	xxxx	xxxx
b0[3:0]	1011	0110	0011	xxxx	xxxx
b1[3:0]	xxxx	1011	0110	xxxx	xxxx
p0[5:0]	100111	010010	000110	xxxxxx	xxxxx
p1[6:0]	xxxxxxx	0100111	0110110	xxxxxxx	xxxxxxx
p[7:0]	xxxxxxxx	xxxxxxxx	10001111	xxxxxxxx	xxxxxxxx

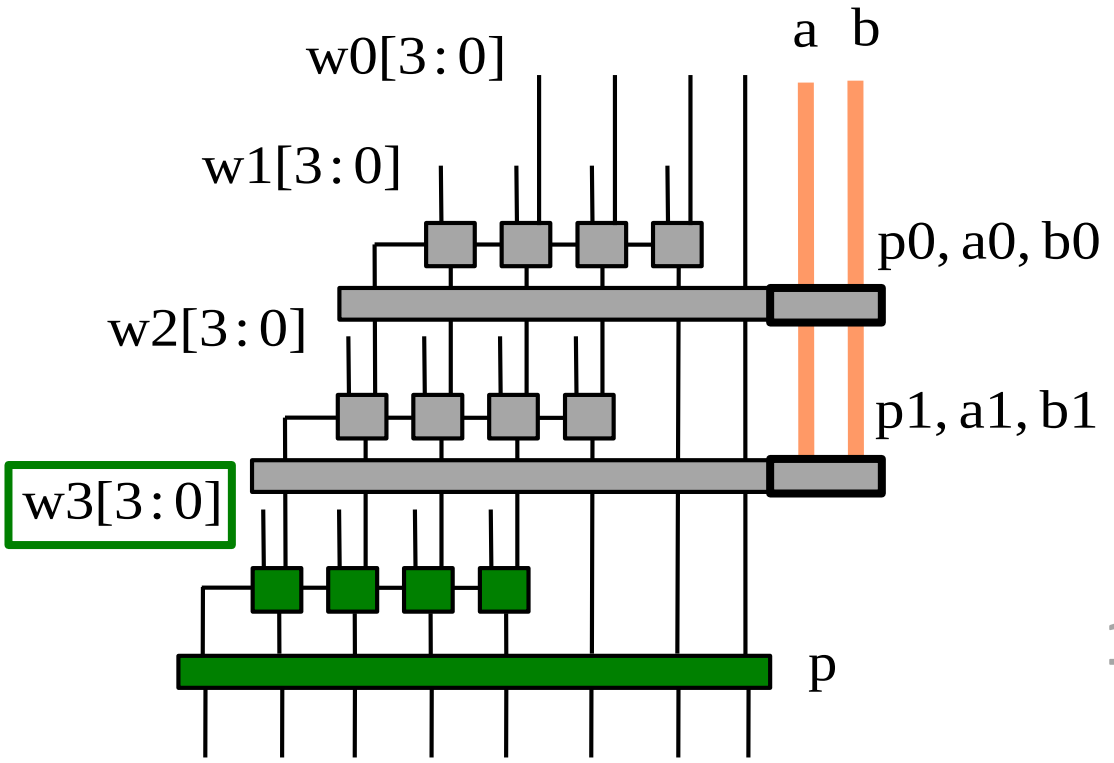
パイプライン乗算器（C4→C5）



1101	1001	0010
x 1011	x 0110	x 0011
1101	0000	0010
11010	10010	00100
000000	100100	000000
1101000	0000000	0000000
10001111	00110110	00000110

a0[3:0]	1101	1001	0010	xxxx	xxxx
a1[3:0]	xxxx	1101	1001	0010	xxxx
b0[3:0]	1011	0110	0011	xxxx	xxxx
b1[3:0]	xxxx	1011	0110	0011	xxxx
p0[5:0]	100111	010010	000110	xxxxxx	xxxxx
p1[6:0]	xxxxxxx	0100111	0110110	0000110	xxxxxxx
p[7:0]	xxxxxxxx	xxxxxxxx	10001111	00110110	xxxxxxxx

パイプライン乗算器（C5→C6）



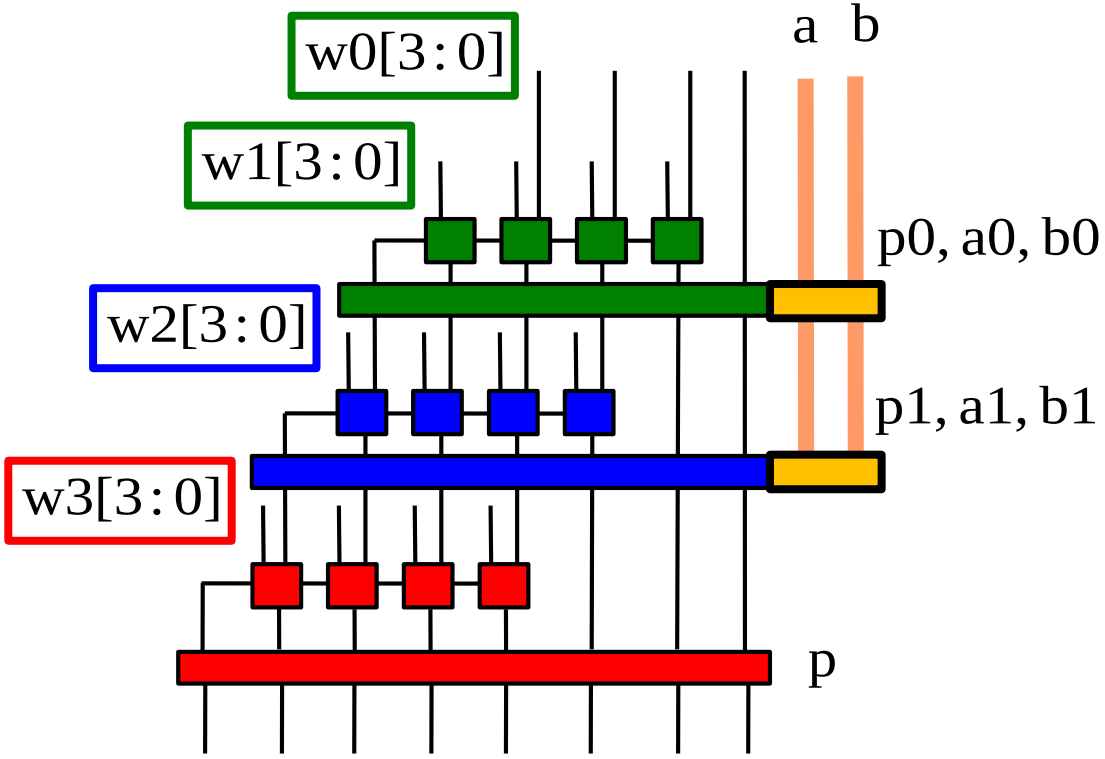
1101	1001	0010
x 1011	x 0110	x 0011
1101	0000	0010
11010	10010	00100
000000	100100	000000
1101000	0000000	0000000
10001111	00110110	00000110

a0[3:0]	1101	1001	0010	xxxx	xxxx
a1[3:0]	xxxx	1101	1001	0010	xxxx
b0[3:0]	1011	0110	0011	xxxx	xxxx
b1[3:0]	xxxx	1011	0110	0011	xxxx
p0[5:0]	100111	010010	000110	xxxxxx	xxxxx
p1[6:0]	xxxxxxx	0100111	0110110	0000110	xxxxxxx
p[7:0]	xxxxxxxx	xxxxxxxx	10001111	00110110	00000110

4bitパイプライン乗算器

```
module mul_p(clk, a, b, p);

    input         clk;
    input  [3:0] a, b;
    output [7:0] p;
    reg    [5:0] p0;
    reg    [6:0] p1;
    reg    [7:0] p;
    reg    [3:0] a0, b0, a1, b1;
    wire   [3:0] w0, w1, w2, w3;

    assign w0 = a  & {4{b[0]}};
    assign w1 = a  & {4{b[1]}};
    assign w2 = a0 & {4{b0[2]}};
    assign w3 = a1 & {4{b1[3]}};

    always@(posedge clk) begin
        p0 <= w0 + {w1,1'b0};
        a0 <= a;
        b0 <= b;
    end
    always@(posedge clk) begin
        p1 <= p0 + {w2,2'b00};
        a1 <= a0;
        b1 <= b0;
    end
    always@(posedge clk) begin
        p  <= p1 + {w3,3'b000};
    end
endmodule
```

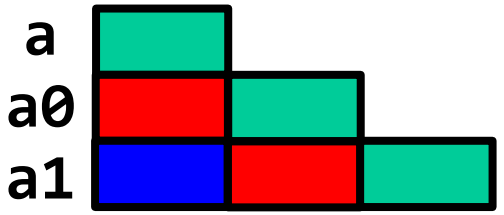
被乗数・乗数もパイプラインへ
部分積の計算で必要



2種類の代入文

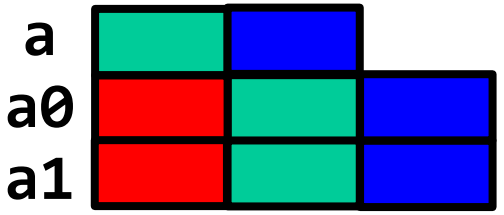
- ノンブロッキング代入
  - 全ての式の右辺を先に評価して左辺に代入
  - 以下の例では a0 ← a と a1 ← a0 の並列代入

```
always@(posedge clk) begin
   a0 <= a;
   a1 <= a0;
end
```

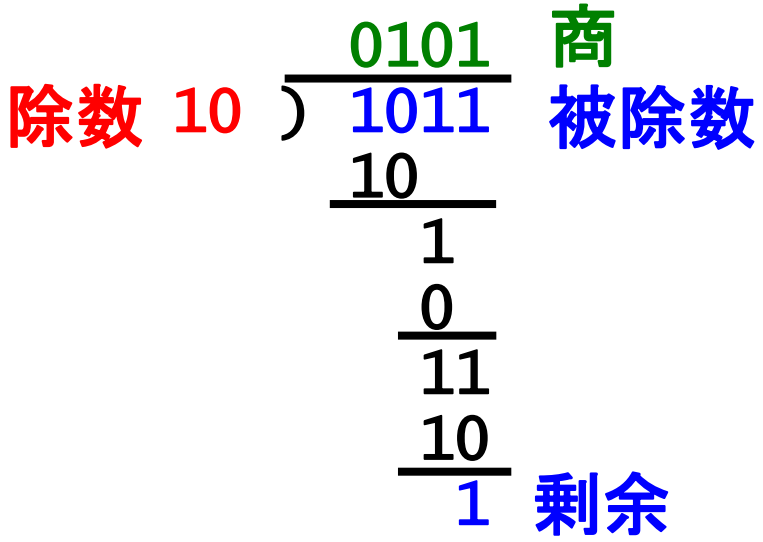


- ブロッキング代入
  - 上の式を評価してから下の式を評価
  - 以下の例では a1 ← a0 ← a の直列代入

```
always@(posedge clk) begin
   a0 = a;
   a1 = a0;
end
```

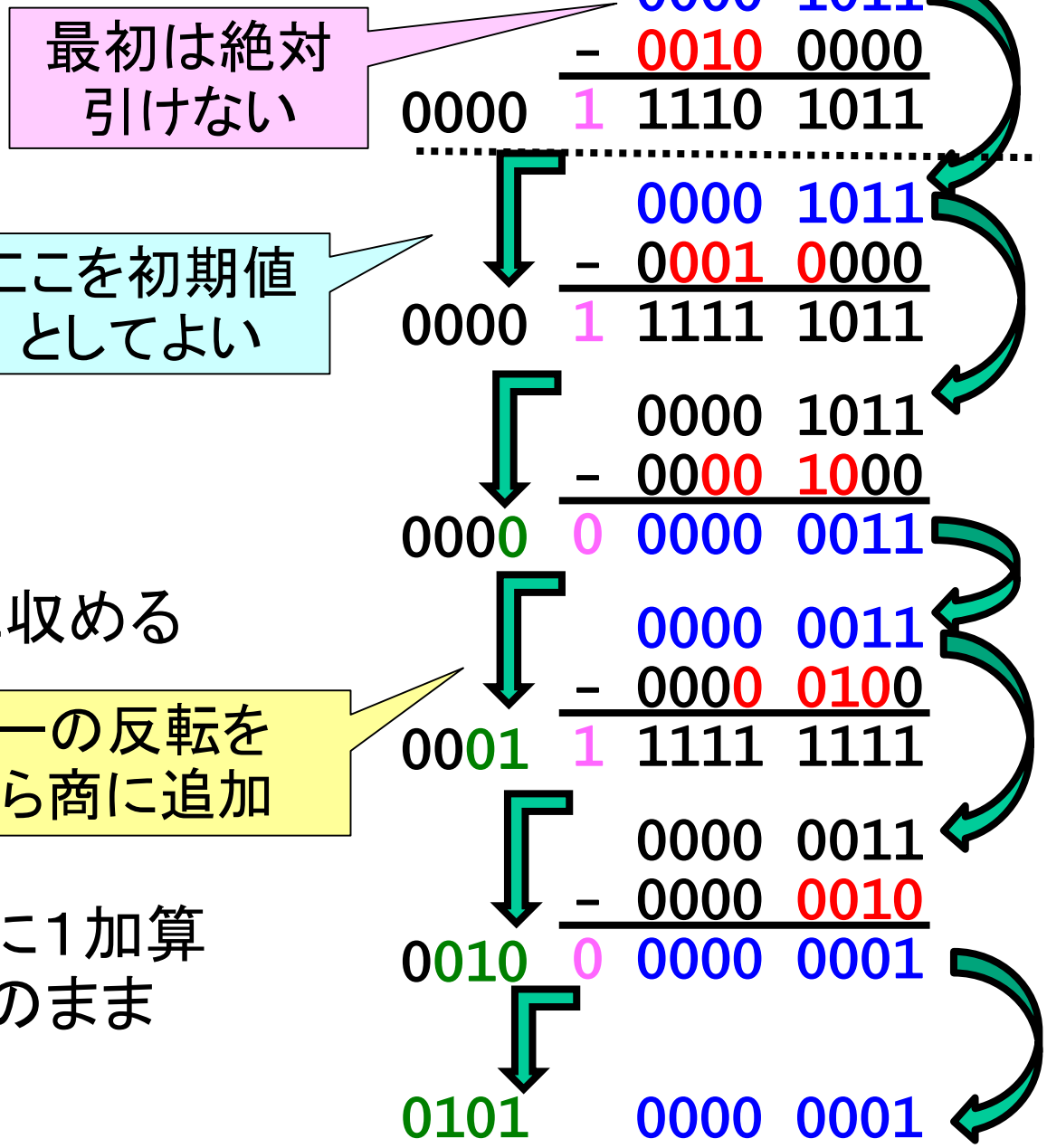


除算器の設計



0a. 剰余の初期値：被除数
0b. 除数の初期値：左半分に収める
0c. 商の初期値：0

1. 商を左シフト
2. 差＝剰余－除数
3a. 差＞0なら剰余＝差、商に1加算
3b. 差＜0なら剰余と商はそのまま
4. 除数を右シフト



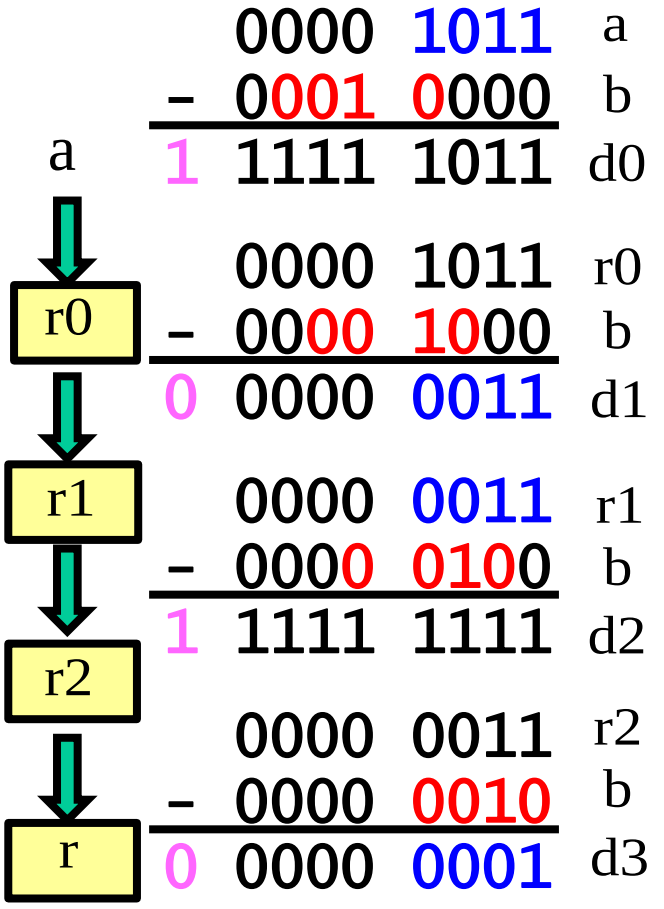
4bit単一サイクル除算器

```
module div(clk, a, b, q, r);
    input         clk;
    input  [3:0] a, b;
    output [3:0] q, r;
    reg    [3:0] q, r;
    wire   [8:0] d0, d1, d2, d3;
    wire   [3:0] r0, r1, r2;

    assign d0 = {4'b0000,a } - {1'b0,b,3'b000};
    assign d1 = {4'b0000,r0} - {2'b00,b,2'b00};
    assign d2 = {4'b0000,r1} - {3'b000,b,1'b0};
    assign d3 = {4'b0000,r2} - {4'b0000,b};
    assign r0 = (d0[8] == 1'b1) ? a  : d0[3:0];
    assign r1 = (d1[8] == 1'b1) ? r0 : d1[3:0];
    assign r2 = (d2[8] == 1'b1) ? r1 : d2[3:0];

    always@(posedge clk) begin
       r  <= (d3[8] == 1'b1) ? r2 : d3[3:0];
       q  <= {~d0[8],~d1[8],~d2[8],~d3[8]};
    end
endmodule
```

```
      0000 1011   a
    - 0001 0000   b
0←  1 1111 1011   d0

      0000 1011   r0
    - 0000 1000   b
1←  0 0000 0011   d1

      0000 0011   r1
    - 0000 0100   b
0←  1 1111 1111   d2

      0000 0011   r2
    - 0000 0010   b
1←  0 0000 0001   d3

0101 q            0001 r
```

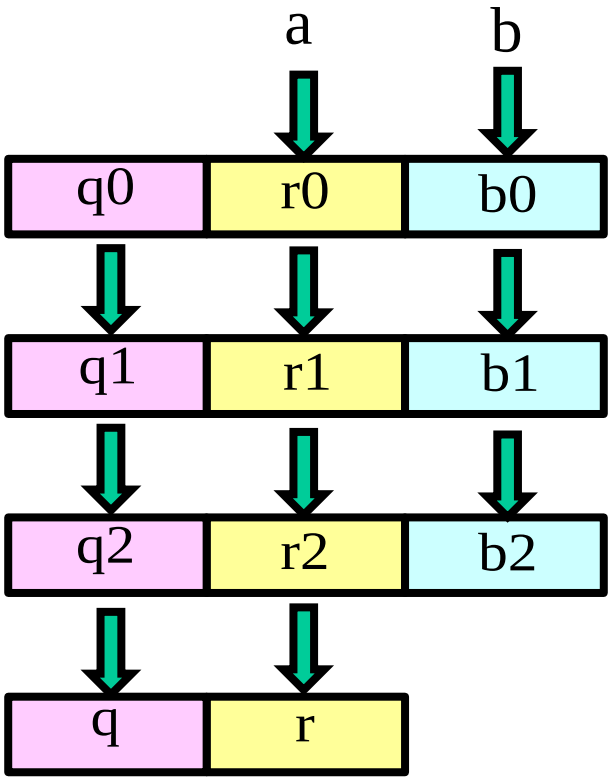
4bitマルチサイクル除算器

```
module div(clk, a, b, q, r);
    input         clk;
    input  [3:0] a, b;
    output [3:0] q, r;
    reg    [3:0] q, r;
    wire   [8:0] d0, d1, d2, d3;
    reg    [3:0] r0, r1, r2;

    assign d0 = {4'b0000,a } - {1'b0,b,3'b000};
    assign d1 = {4'b0000,r0} - {2'b00,b,2'b00};
    assign d2 = {4'b0000,r1} - {3'b000,b,1'b0};
    assign d3 = {4'b0000,r2} - {4'b0000,b};

    always@(posedge clk) begin
        r0 <= (d0[8] == 1'b1) ? a  : d0[3:0];
        r1 <= (d1[8] == 1'b1) ? r0 : d1[3:0];
        r2 <= (d2[8] == 1'b1) ? r1 : d2[3:0];
        r  <= (d3[8] == 1'b1) ? r2 : d3[3:0];
        q  <= {~d0[8],~d1[8],~d2[8],~d3[8]};
    end
endmodule
```

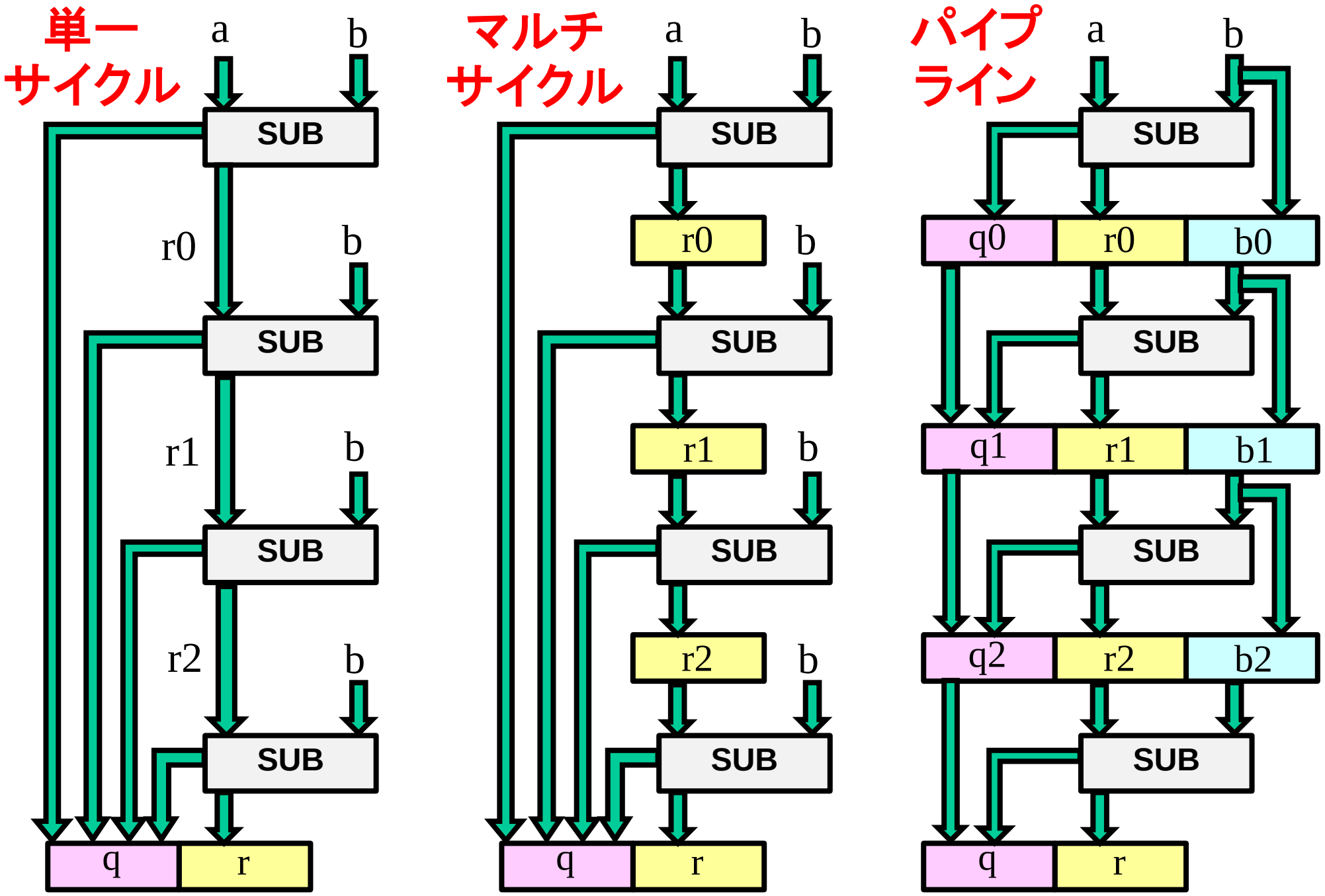


4bitパイプライン除算器

```
module div(clk, a, b, q, r);
  input         clk;
  input  [3:0] a, b;
  output [3:0] q, r;
  reg    [3:0] q, r;
  wire   [8:0] d0, d1, d2, d3;
  reg    [3:0] r0, r1, r2;
  reg    [3:0] q0, q1, q2;
  reg    [3:0] b0, b1, b2;

  assign d0 = {4'b0000,a } - {1'b0,b,3'b000};
  assign d1 = {4'b0000,r0} - {2'b00,b0,2'b00};
  assign d2 = {4'b0000,r1} - {3'b000,b1,1'b0};
  assign d3 = {4'b0000,r2} - {4'b00002,b2};

  always@(posedge clk) begin
    r0 <= (d0[8] == 1'b1) ? a  : d0[3:0];
    r1 <= (d1[8] == 1'b1) ? r0 : d1[3:0];
    r2 <= (d2[8] == 1'b1) ? r1 : d2[3:0];
    r  <= (d3[8] == 1'b1) ? r2 : d3[3:0];
    q0 <= {3'b0,~d0[8]};
    q1 <= {q0[2:0],~d1[8]};
    q2 <= {q1[2:0],~d2[8]};
    q  <= {q2[2:0],~d3[8]};

    b0 <= b;
    b1 <= b0;
    b2 <= b1;
  end
endmodule
```

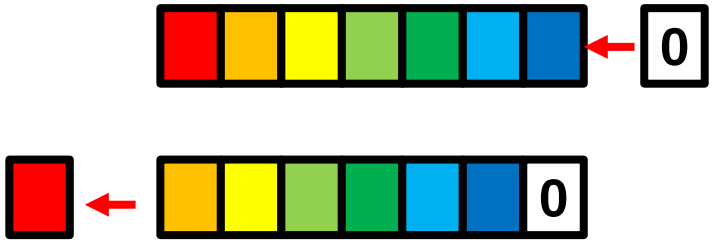


除算器の設計


単一
サイクル
a
b
SUB
r0
b
SUB
r1
b
SUB
r2
b
SUB
q
r
マルチ
サイクル
a
b
SUB
r0
b
SUB
r1
b
SUB
r2
b
SUB
q
r
パイプ
ライン
a
b
SUB
q0
r0
b0
SUB
q1
r1
b1
SUB
q2
r2
b2
SUB
q
r

シフトレジスタ（左シフト）

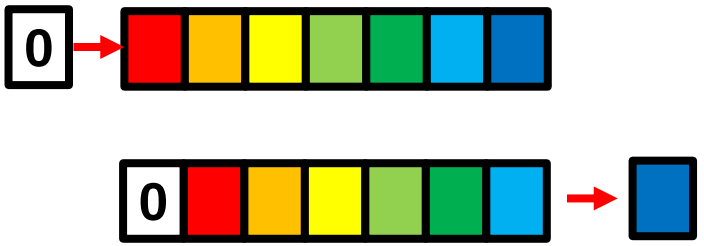
```
module shreg(
    clk, ld, d, q);

    input         clk, ld;
    input  [7:0] d;
    output [7:0] q;
    reg    [7:0] q;

    always@(posedge clk) begin
        if(ld == 1'b1) begin
            q <= d;
        end else begin
            q <= {q[6:0], 1'b0};
        end
    end

endmodule
```



1ビットずつ左シフト
最下位ビットに0を挿入
最上位ビットは消滅

シフトレジスタ(右シフト)

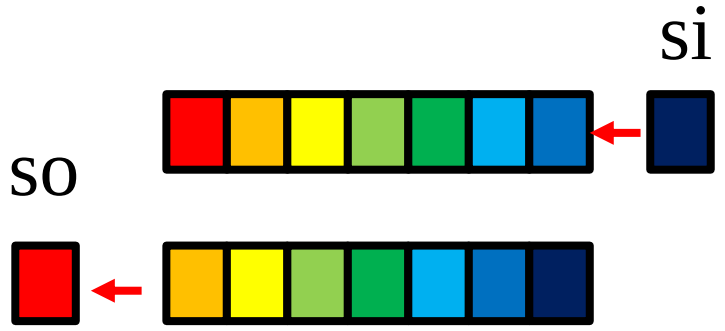
```
module shreg(
    clk, ld, d, q);

    input         clk, ld;
    input  [7:0] d;
    output [7:0] q;
    reg    [7:0] q;

    always@(posedge clk) begin
        if(ld == 1'b1) begin
            q <= d;
        end else begin
            q <= {1'b0, q[7:1]};
        end
    end

endmodule
```



1ビットずつ右シフト
最上位ビットに0を挿入
最下位ビットは消滅

入出力付きシフトレジスタ(左シフト)

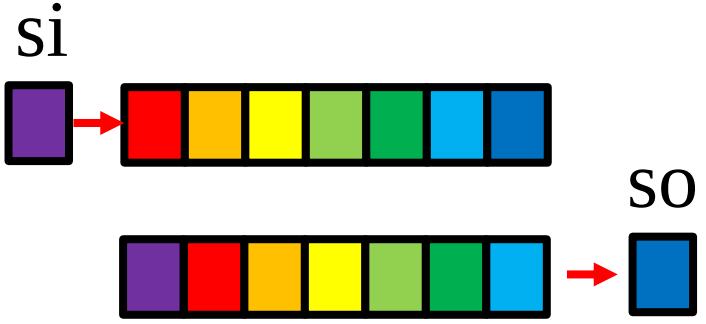
```
module shreg(
    clk, ld, d, si, so, q);

    input         clk, rst, ld;
    input  [7:0] d;
    input         si;
    output        so;
    output [7:0] q;
    reg    [7:0] q;

    always@(posedge clk) begin
        if(ld == 1'b1) begin
            q <= d;
        end else begin
            q <= {q[6:0], si};
        end
    end

    assign so = q[7];

endmodule
```



1ビットずつ左シフト
空いたビットに入力si
あふれたビットが出力so

入出力付きシフトレジスタ(右シフト)

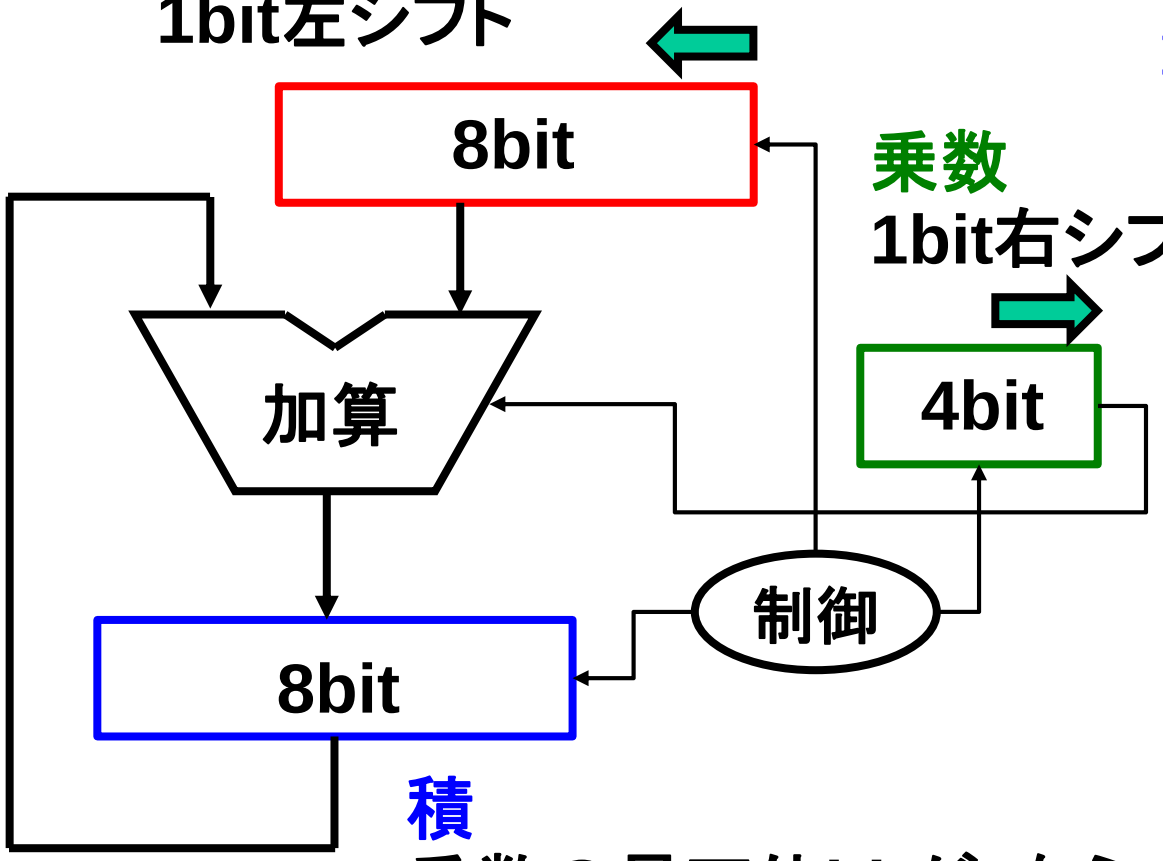
```
module shreg(
    clk, ld, d, si, so, q);

    input         clk, rst, ld;
    input  [7:0] d;
    input         si;
    output        so;
    output [7:0] q;
    reg    [7:0] q;

    always@(posedge clk) begin
        if(ld == 1'b1) begin
            q <= d;
        end else begin
            q <= {si, q[7:1]};
        end
    end

    assign so = q[0];

endmodule
```



1ビットずつ右シフト
空いたビットに入力si
あふれたビットが出力so

（1）逐次型乗算器

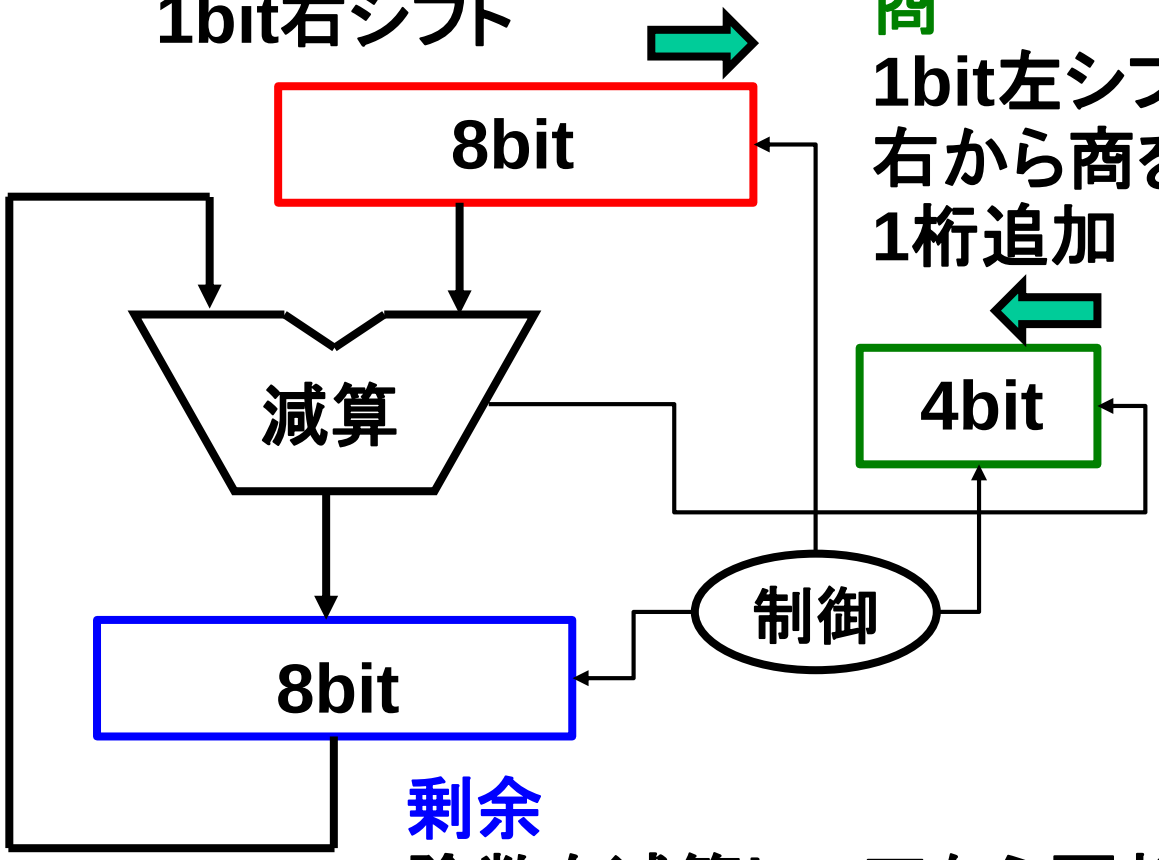
```
    1101
x   1011
------
    1101
   1101 0
  0000 00
 1101 000
------
10001111
```



乗数の最下位bitが1なら
被乗数を加算（初期値：0）

C1	00000000 + 00001101 = 00001101	1011
C2	00001101 + 00011010 = 00100111	0101
C3	00100111 + 00000000 = 00100111	0010
C4	00100111 + 01101000 = 10001111	0001
C5	10001111	0000

（2）逐次型除算器




```
      0101
10 ) 1011
     10
      1
      0
      11
      10
       1
```

	00001011	0000
C1	− 00010000	
	1 11111011	
	00001011	0000
C2	− 00001000	
	0 00000011	
	00000011	0001
C3	− 00000100	
	1 11111111	
	00000011	0010
C4	− 00000010	
	0 00000001	
C5	00000001	0101

課題

（1）逐次型乗算器

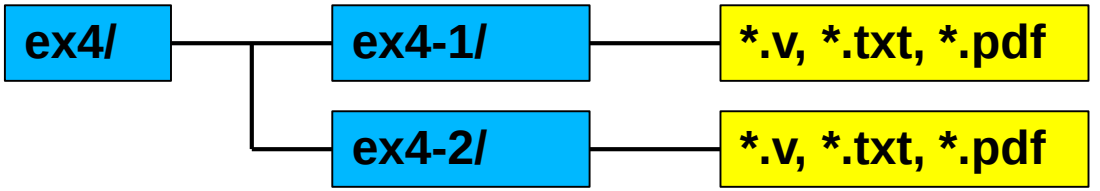
（2）逐次型除算器

verilogソースファイル(テストベンチを含む)を作成し，シミュレーションを実行せよ.

また，シミュレーション波形も表示させよ.

提出すべきファイル

- 課題の回路のHDLソースファイル（テストベンチを含む）
  課題（1）～（2）に対してディレクトリ（ex4-?）を作成すること
- 課題の回路のシミュレーション実行結果
  （ iverilog の実行結果をリダイレクトして txt ファイルを作成）
- 課題の回路のシミュレーション波形
  （gtkwaveから pdf ファイルを作成）
- 圧縮アーカイブファイル（ex4-?.tar.gz）を作成して
  Web上から提出せよ




```
cd ex4
tar zcvf ex4-1.tar.gz ex4-1/
tar zcvf ex4-2.tar.gz ex4-2/
```