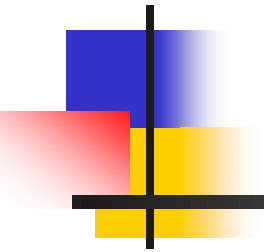


數位IC設計

Case Study

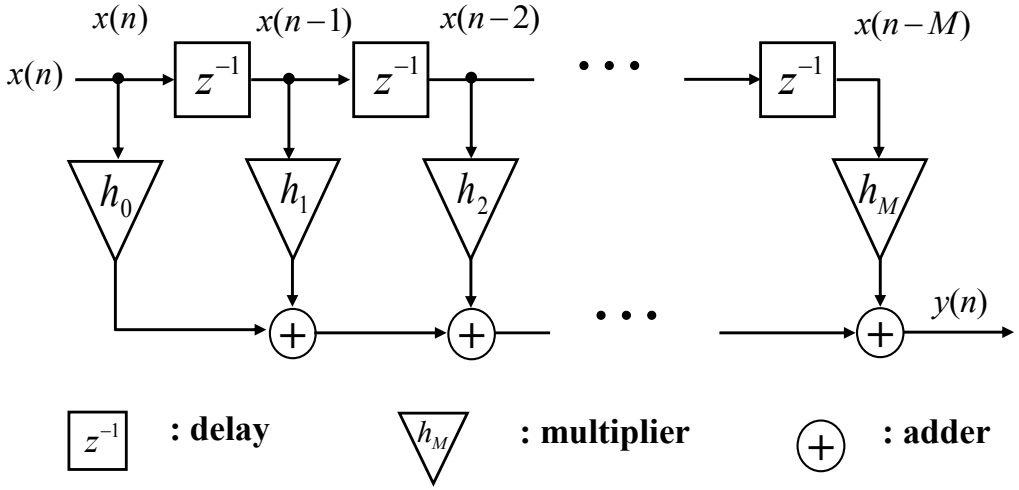
Case - Filter (1/8)

FIR 濾波器 $y(n)=\sum_{m=0}^{M} h_m x(n-m)$

Filter tap=4

output 和 前面4個輸入有關

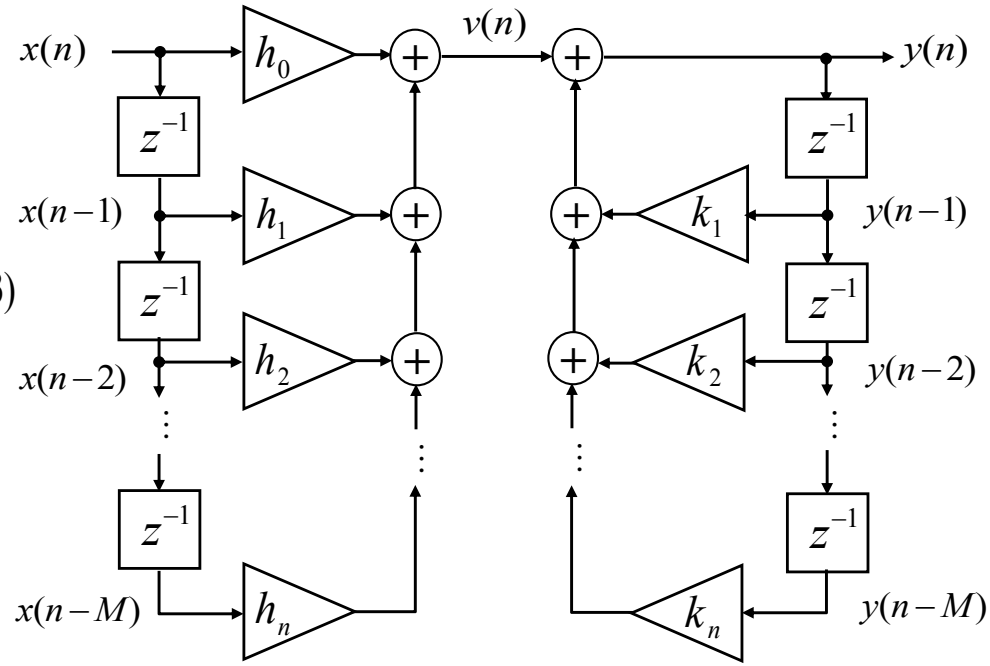
$$y(0)=h_0x(0)+h_1x(-1)+h_2x(-2)+h_3x(-3)$$
$$y(1)=h_0x(1)+h_1x(0)+h_2x(-1)+h_3x(-2)$$
$$y(2)=h_0x(2)+h_1x(1)+h_2x(0)+h_3x(-1)$$
$$y(3)=h_0x(3)+h_1x(2)+h_2x(1)+h_3x(0)$$
$$y(4)=h_0x(4)+h_1x(3)+h_2x(2)+h_3x(1)$$



output　現在 input　之前 input

IIR $y(n)-\sum_{m=1}^{M} k_m y(n-m)=\sum_{m=0}^{M} h_m x(n-m)$

$$y(0)-[k_1y(-1)+k_2y(-2)+k_3y(-3)]=h_0x(0)+h_1x(-1)+h_2x(-2)h_3x(-3)$$
$$y(1)-[k_1y(0)+k_2y(-1)+k_3y(-2)]=h_0x(1)+h_1x(0)+h_2x(-1)h_3x(-2)$$
$$y(2)-[k_1y(1)+k_2y(0)+k_3y(-1)]=h_0x(2)+h_1x(1)+h_2x(0)h_3x(-1)$$
$$y(3)-[k_1y(2)+k_2y(1)+k_3y(0)]=h_0x(3)+h_1x(2)+h_2x(1)h_3x(0)$$
$$y(4)-[k_1y(3)+k_2y(2)+k_3y(1)]=h_0x(4)+h_1x(3)+h_2x(2)h_3x(1)$$

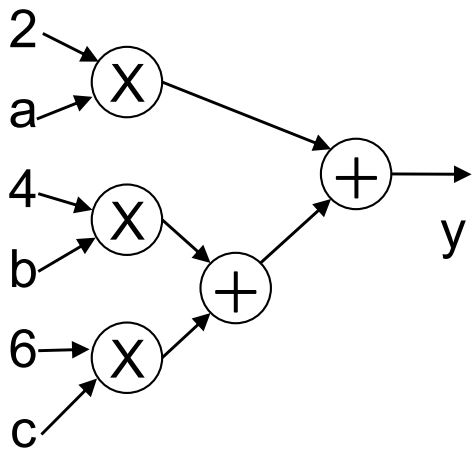


Case - Filter (2/8)

tap=3

h_0=2; h_1=4, h_2=6;

```
module fir1(a, b, c, y);
input [7:0] a, b, c;
output [11:0] y;
assign y = a*2+b*4+c*6;
endmodule
```

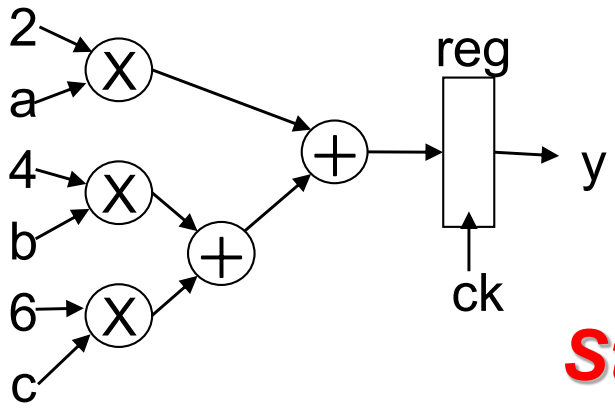



```
module fir2(a, b, c, y);
input [7:0] a, b, c;
output [11:0] y;
reg [11:0] y;
always @(a or b or c)
y = a*2+b*4+c*6;
endmodule
```

Output: 20, 32, 44, 56, 68, 80, 92, 104, ….

Case - Filter (3/8)

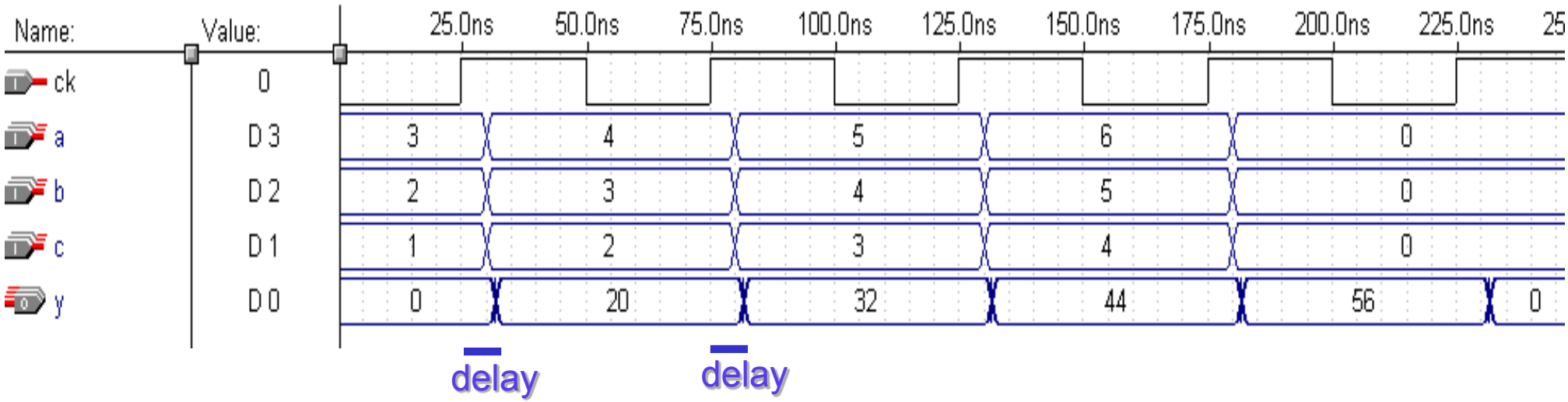
```
module fir3(a, b, c, y, ck);
input [7:0] a, b, c;
input ck;
output [11:0] y;
reg [11:0] y;
always @(posedge ck)
y = a*2+b*4+c*6;
endmodule
```



Stable output

Three inputs (a, b, c) must be entered concurrently (more pins, higher cost).

Output: 20, 32, 44, 56, 68, 80, 92, 104, ….



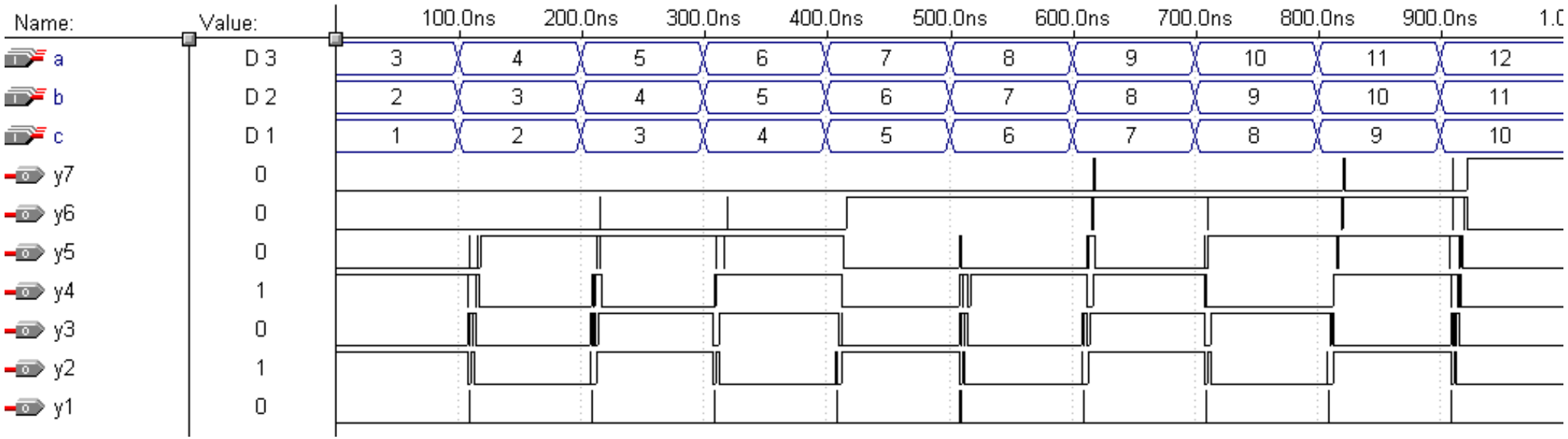
The stable output is generated at the positive edge of clock.

fir2

Case - Filter (4/8)

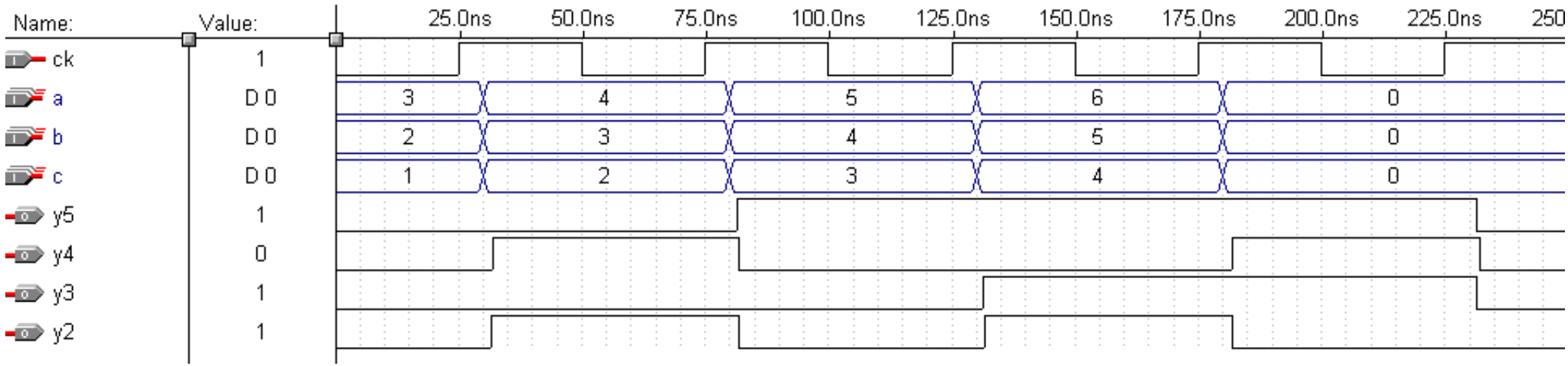
fir2

Unstable output



fir3

Stable output



Case - Filter (5/8)

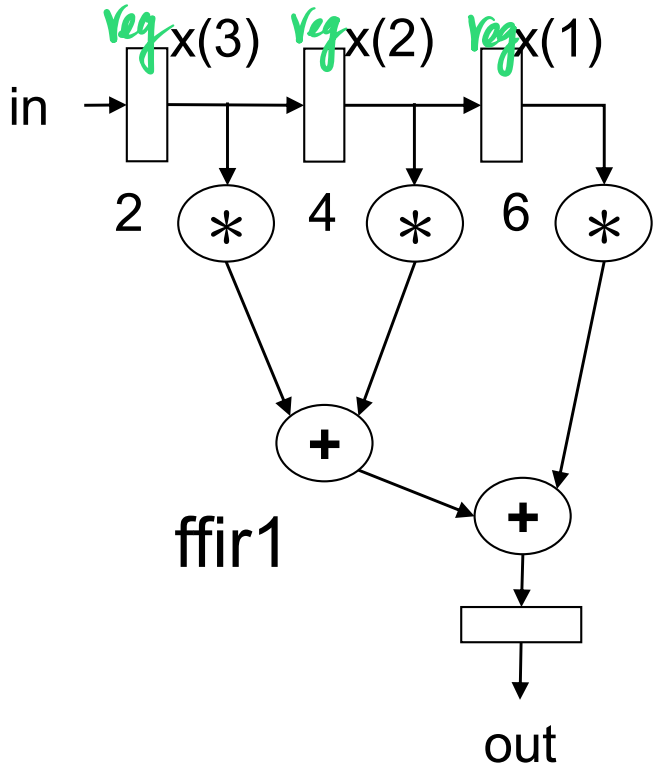
```
module register(in, out, ck);
input [11:0] in;
input ck;
output [11:0] out;
reg [11:0] out;
always @(posedge ck)
out=in;
endmodule
```

```
`include "register.v"
module ffir1(in, out, ck);
input [11:0] in;
input ck;
output [11:0] out;
wire [11:0] x4, x3, x2, x1;
register r1(in, x3, ck);
register r2(x3, x2, ck);
register r3(x2, x1, ck);
register r4(x4, out, ck)
assign x4=x1*6+x2*4+x3*2;
endmodule
```

利用reg，當此來的時候
把上一時刻的值傳到下一個reg
⇒ 一次只需吃一個input，成本較低

```
module ffir1_a(in, out, ck);
input                   ck;
input  [11:0] in;
output [11:0] out;
reg    [11:0] out;
reg    [11:0] x4, x3, x2, x1;
always @(posedge ck)
begin
        x3<=in;
        x2<=x3;
        x1<=x2;
        out<=(x3*2+x2*4)+x1*6;
end
endmodule
```

T(n+1) ➔ 4 3 2
T(n) ➔ 3 2 1 ⇨



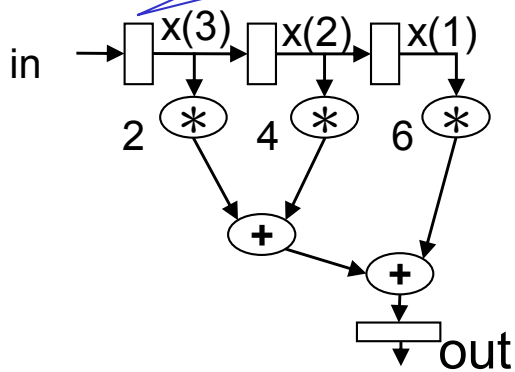
One input (in) is entered every clock cycle
(more suitable for memory access and pins' cost)

Case - Filter (6/8)

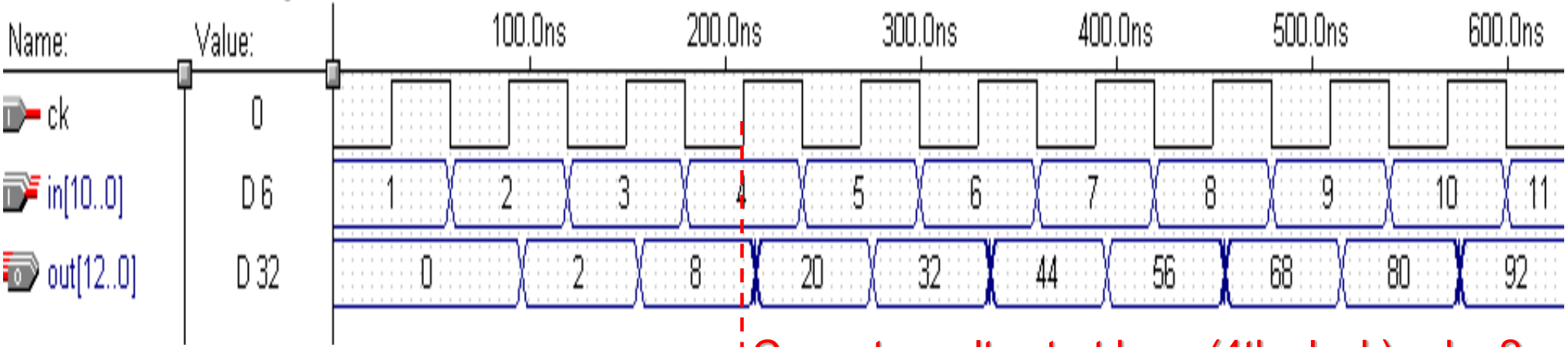
T(4)	→ 4	3	2
T(3)	→ 3	2	1
T(2)	→ 2	1	X
T(1)	→ 1	X	X
T(0)	→ X	X	X

To work well, every input must be ready before the positive edge of every clock

ffir1

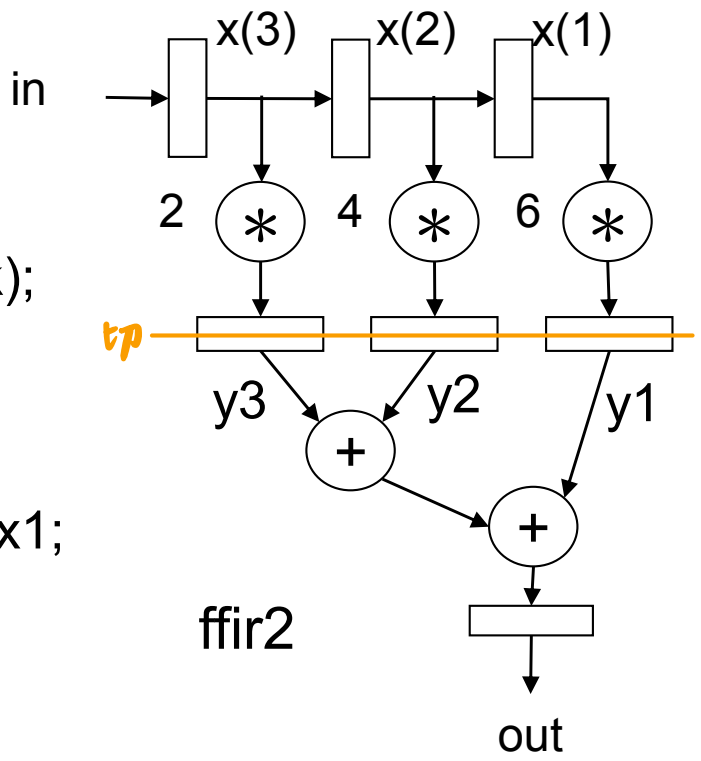


Output: 20, 32, 44, 56, 68, 80, 92, 104, ….



Correct results start here (4th clock), why ?

Case - Filter (7/8)

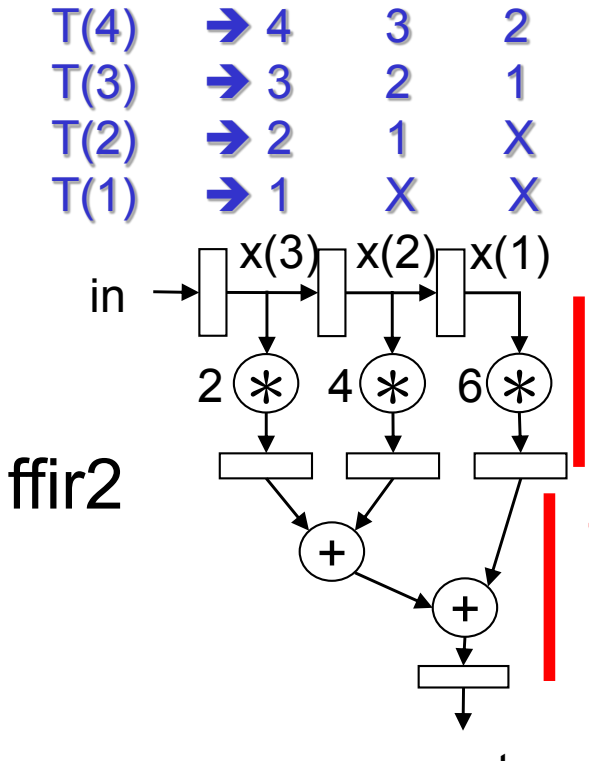



```
`include "register.v"
module ffir2(in, out, ck);
input [11:0] in;
input ck;
output [11:0] out;
wire [11:0] x4, x3, x2, x1;
wire [11:0] t3, t2, t1;
wire [11:0] y3, y2, y1;

register r1(in, x3, ck); register r2(x3, x2, ck);
register r3(x2, x1, ck);
assign t3=x3*2; assign t2=x2*4; assign t1=x1*6;
register r4(t3, y3, ck); register r5(t2, y2, ck);
register r6(t1, y1, ck);
assign x4=y1+y2+y3;
register r7(x4, out, ck);
endmodule
```

Datapath Pipelining

```
module ffir2_a(in, out, ck);
input         ck;
input  [11:0] in;
output [11:0] out;
reg    [11:0] out;
reg    [11:0] x3, x2, x1;
reg    [11:0] y3, y2, y1;
always @(posedge ck)
begin
              x3<=in;
              x2<=x3;
              x1<=x2;
              y3<=x3*2;
              y2<=x2*4;
              y1<=x1*6;
tp ___________________
              out<=(y3+y2)+y1;
end
endmodule
```

Case - Filter (8/8)



Delay for * is about 7.3 ns

Delay for register assign is about 6.1 ns

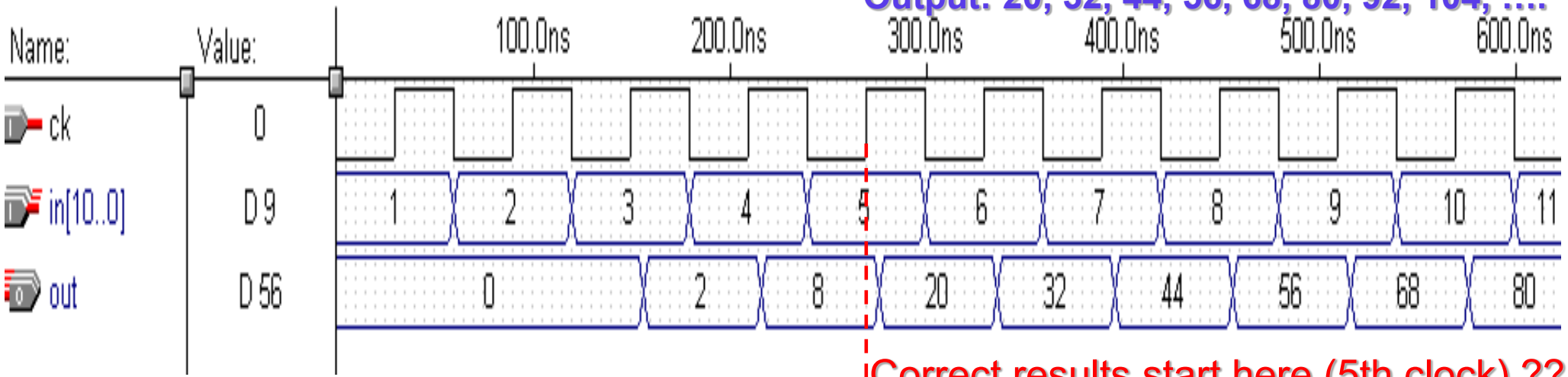
Delay for + is about 2.6 ns

Total delay=7.3+6.1=13.4 ns (+ little wire delay)

Total delay=2.6*2+6.1=11.3 ns

Critical path=13.4 ns => clock rate less than $1/(13.4*10^{-9})$ ~= 74.6 MHz 較快

Output: 20, 32, 44, 56, 68, 80, 92, 104, ….



Correct results start here (5th clock) ??

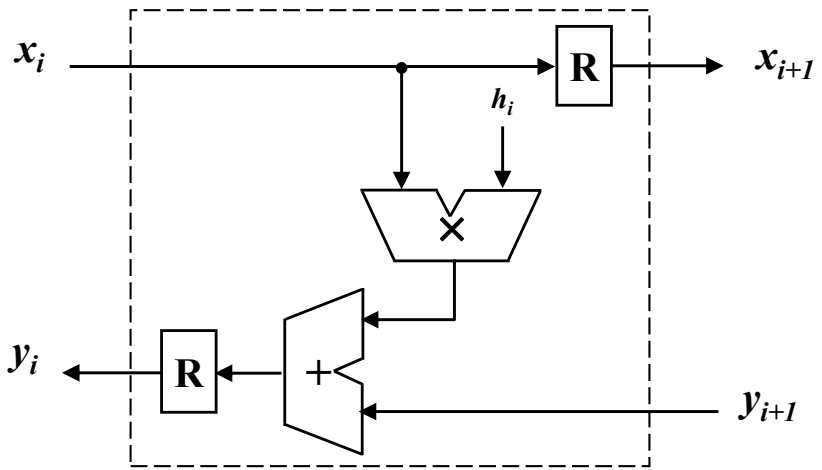
Case - Systolic Array (1/5)

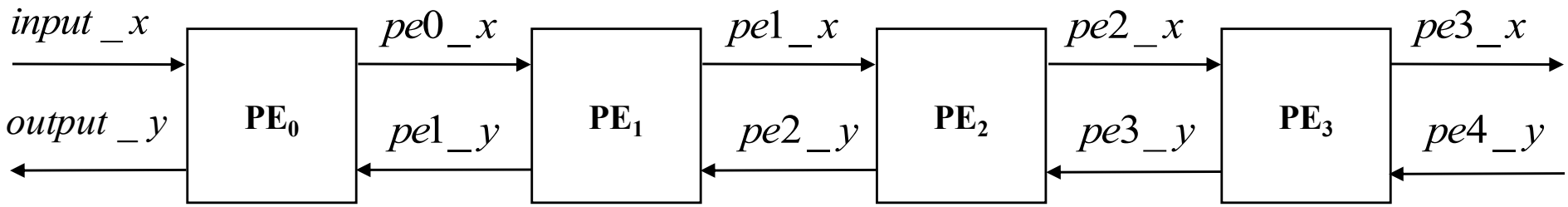
Systolic Array (FIR)

$$y(0) = h_0x(0) + h_1x(-1) + h_2x(-2) + h_3x(-3)$$
$$y(1) = h_0x(1) + h_1x(0) + h_2x(-1) + h_3x(-2)$$
$$y(2) = h_0x(2) + h_1x(1) + h_2x(0) + h_3x(-1)$$
$$y(3) = h_0x(3) + h_1x(2) + h_2x(1) + h_3x(0)$$
$$y(4) = h_0x(4) + h_1x(3) + h_2x(2) + h_3x(1)$$

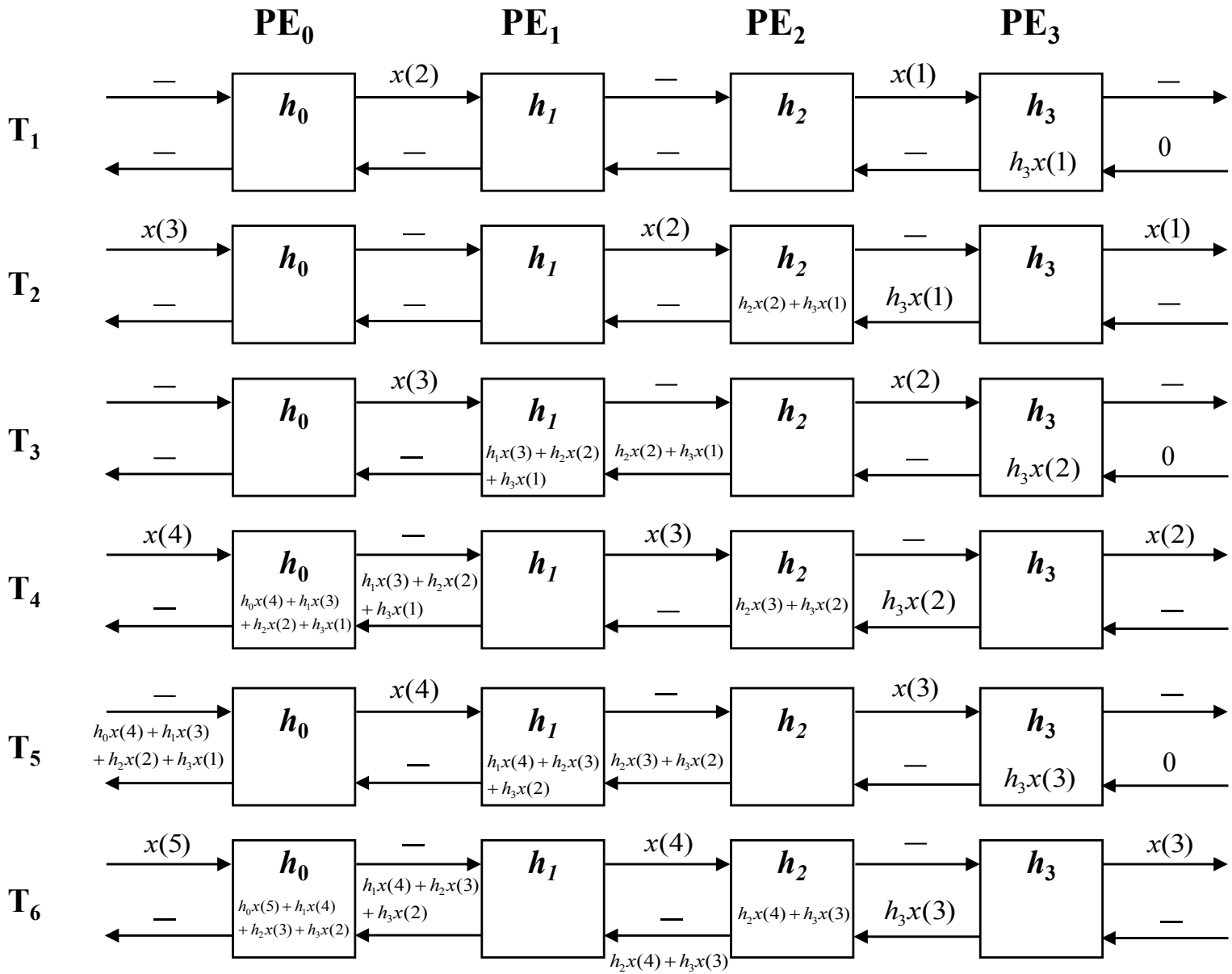
+ : adder **× : multiplier**
R : register **h_i : coefficient**

Processing Element





Case - Systolic Array (2/5)



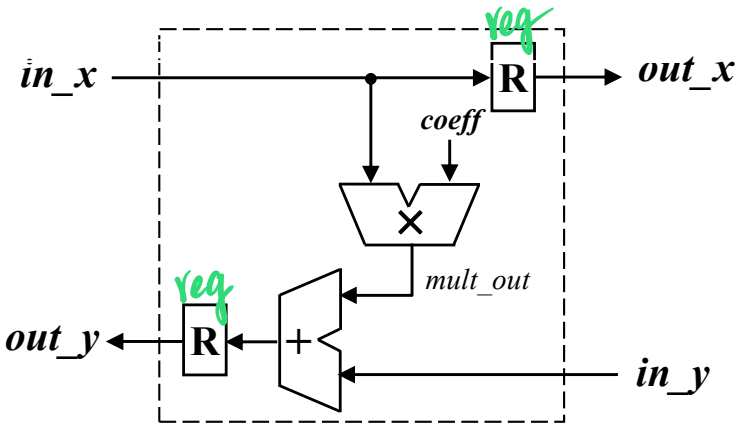
Case - Systolic Array (3/5)

	$input\_x$	$pe0\_x$	$pe1\_x$	$pe2\_x$	$pe3\_x$	$pe4\_y$	$pe3\_y$	$pe2\_y$	$pe1\_y$	$output\_y$
T_1	–	$x(2)$	–	$x(1)$	–	–	–	—	—	—
T_2	$x(3)$	–	$x(2)$	–	$x(1)$	0	$h_3x(1)$	—	—	—
T_3	–	$x(3)$	–	$x(2)$	–	–	–	$h_2x(2)$ $+ h_3x(1)$	—	
T_4	$x(4)$	–	$x(3)$	–	$x(2)$	0	$h_3x(2)$	—	$h_1x(3) + h_2x(2)$ $+ h_3x(1)$	—
T_5	–	$x(4)$	–	$x(3)$	–	–	–	$h_2x(3)$ $+ h_3x(2)$	—	$h_0x(4) + h_1x(3)$ $+ h_2x(2) + h_3x(1)$
T_6	$x(5)$	–	$x(4)$	–	$x(3)$	0	$h_3x(3)$	—	$h_1x(4) + h_2x(3)$ $+ h_3x(2)$	—
T_7	–	$x(5)$	–	$x(4)$	–	–	–	$h_2x(4)$ $+ h_3x(3)$	—	$h_0x(5) + h_1x(4)$ $+ h_2x(3) + h_3x(2)$

Case - Systolic Array (4/5)

Design_1 structural

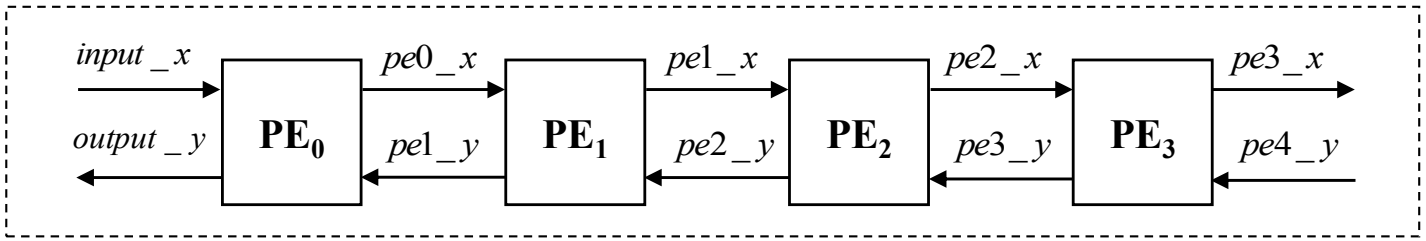
```
module pe(clk, reset, coeff, in_x, in_y, out_x, out_y);
parameter size = 8;
input   clk, reset;
input    [size-1:0]         in_x, coeff;
input    [size+size-1:0]  in_y;
output  [size-1:0]         out_x;
output  [size+size-1:0]  out_y;
wire     [size+size-1:0]  mult_out, add_out;
reg_8 r1(clk, reset, in_x, out_x);
reg_16 r2(clk, reset, add_out, out_y);
assign mult_out = in_x * coeff;
assign add_out = mult_out + in_y;
endmodule
```



behavior

```
module pe(clk, reset, coeff, in_x, in_y,
out_x, out_y);
parameter size = 8;
input              clk, reset;
input    [size-1:0]         in_x, coeff;
input    [size+size-1:0]  in_y;
output  [size-1:0]         out_x;
output  [size+size-1:0]  out_y;
reg       [size+size-1:0]  out_y;
reg       [size-1:0]         out_x;

always@(posedge clk)
 begin
  if(reset) begin
    out_x = 0;
    out_y = 0;
    end
  else begin
    out_y = in_y + (in_x * coeff);
    out_x = in_x;
    end
 end
endmodule
```

Case - Systolic Array (5/5)




```
//***** main ****************************
module systolic(clk, reset, input_x, output_y);
parameter   size = 8;
input   clk, reset;
input   [size-1:0]          input_x;
output [size+size-1:0]  output_y;
wire    [size-1:0]          pe0_x, pe1_x, pe2_x, pe3_x;
wire    [size+size-1:0]  pe1_y, pe2_y, pe3_y;

wire   [size-1:0]       h0 = 8'h01;
wire   [size-1:0]       h1 = 8'h01;
wire   [size-1:0]       h2 = 8'h01;
wire   [size-1:0]       h3 = 8'h01;
wire   [size+size-1:0]  pe4_y = 16'h0000;

pe pe_0(clk, reset, h0, input_x, pe1_y, pe0_x,
output_y);
pe pe_1(clk, reset, h1, pe0_x, pe2_y, pe1_x, pe1_y);
pe pe_2(clk, reset, h2, pe1_x, pe3_y, pe2_x, pe2_y);
pe pe_3(clk, reset, h3, pe2_x, pe4_y, pe3_x, pe3_y);
endmodule
```

```
//***** register_8bits *******
module  reg_8(clk,reset,in,out);
 parameter   size_in = 8;
 input          clk,reset;
 input          [size_in-1:0]  in;
 output         [size_in-1:0]  out;
 reg            [size_in-1:0]  out;

always @(posedge clk )
 begin
   if(reset)
      out=0;
   else
      out=in;
  end
endmodule
```

```
//***** register_16bits *******
module  reg_16(clk,reset,in,out);
 parameter   size_in = 16;
 input          clk,reset;
 input          [size_in-1:0]  in;
 output         [size_in-1:0]  out;
 reg            [size_in-1:0]  out;

always @(posedge clk )
 begin
  if(reset)
     out=0;
  else
     out=in;
 end
endmodule
```

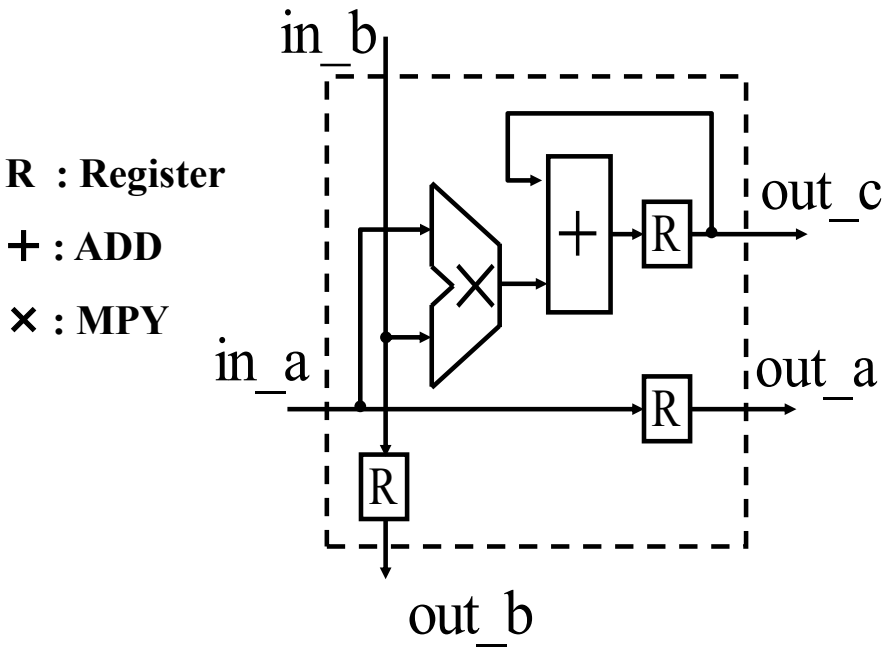
Case-Matrix Multiplication (1/2)

structural

Matrix Multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

c=c+(a*b)

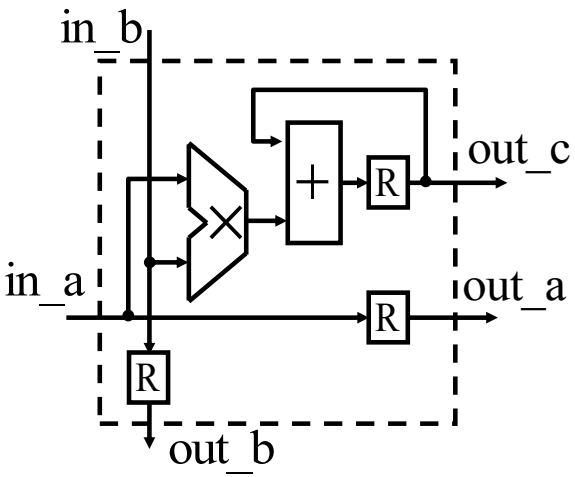



```
`include "reg_8.v"
`include "reg_16.v"

module PE(clk,reset,in_a,in_b,out_a,out_b,out_c);
parameter data_size=8;
input       reset,clk;
input       [data_size-1:0] in_a,in_b;
output      [data_size-1:0] out_a,out_b;
output      [2*data_size-1:0] out_c;
wire        [2*data_size-1:0] out_c,ADD_out,out_MPY;
wire        [data_size-1:0] out_a,out_b;

assign out_MPY=in_a*in_b;
assign ADD_out=out_MPY+out_c;
reg_16 reg_16_0(clk,reset,ADD_out,out_c);
reg_8 reg_delay_8_0(clk,reset,in_a,out_a);
reg_8 reg_delay_8_1(clk,reset,in_b,out_b);
endmodule
```

Case-Matrix Multiplication (2/2)

behavior

```
module PE_H(reset,clk,in_a,in_b,out_a,out_b,out_c);
parameter data_size=8;
input       reset,clk;
input       [data_size-1:0] in_a,in_b;
output      [2*data_size:0] out_c;
output      [data_size-1:0] out_a,out_b;
reg         [2*data_size:0] out_c;
reg         [data_size-1:0] out_a,out_b;
```




```
always @(posedge clk)
begin
    if(reset)
    begin
     out_a=0;
     out_b=0;
     out_c=0;
    end
   else
    begin
     out_c=out_c+in_a*in_b;
     out_a=in_a;
     out_b=in_b;
   end
end
endmodule
```

The rest of circuit can be designed easily….