Design Examples

2025. 11

Yongjin Jeong, KwangWoon University

A FIR Filter: Direct Form

- FIR and IIR

$$y[n] = \sum_{k=0}^{M} b_k \cdot x[n-k]$$

$$y[n] = \sum_{k=0}^{M} b_k \cdot x[n-k] - \sum_{j=1}^{N} a_j \cdot y[n-j]$$

- Direct Form FIR

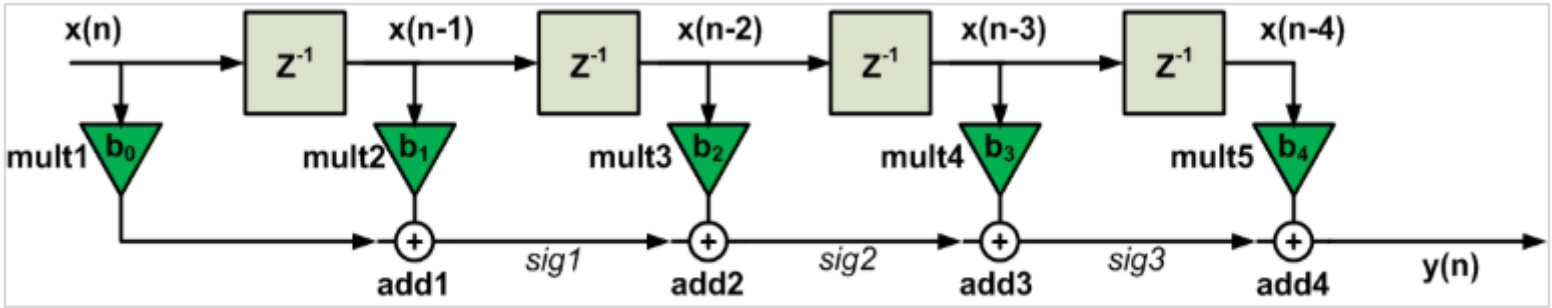
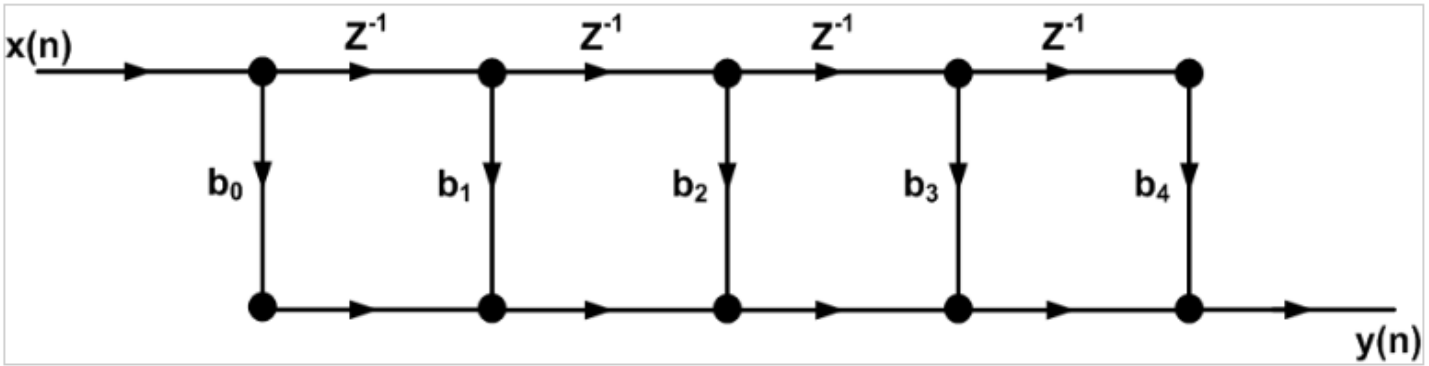


Figure 1. *The direct form of a five-tap FIR filter.*

$$f_{sampling} = \frac{1}{T_{mult} + 4T_{add}}$$

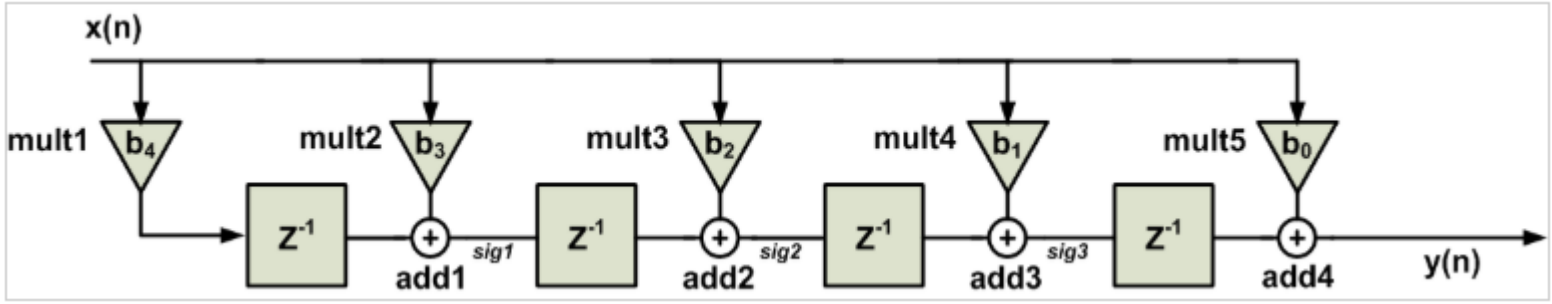
SFG(signal flow graph)



https://www.allaboutcircuits.com/technical-articles/ (by Steve Arar)

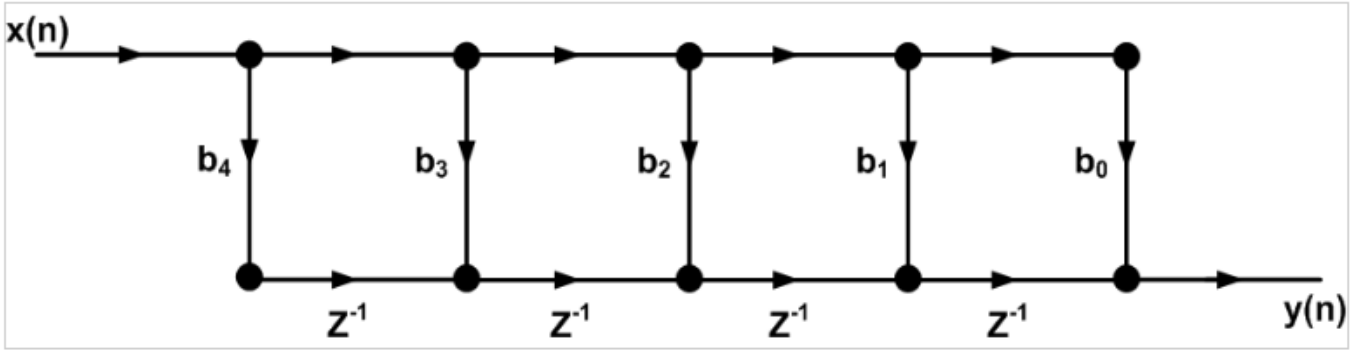
A FIR Filter: Transposed Form

- Transposed Form FIR



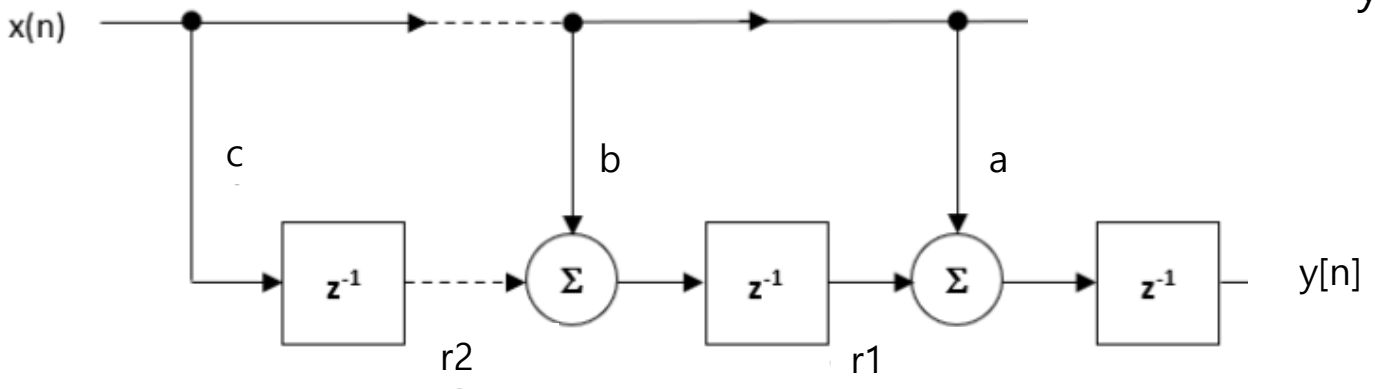
Transposed form of a FIR filter

- self-pipelined
- $1/f_{sampling} = T_{mult} + T_{add}$.



Design Example: Transposed FIR Filter

- **SISO (Single In Single Out) Transposed FIR Filter**
  - Direct implementation of the Transposed FIR Filter



$$y[n] = h0*x[n] + h1*x[n-1] + h2*x[n-2]$$
$$= a*x[n] + b*x[n-1] + c*x[n-2]$$

FIR Filter Configuration

- **Structure:** 3-tap transposed FIR
- **Coefficients:**
  - $h_0 = 1$
  - $h_1 = 2$
  - $h_2 = 3$
- **Input sequence:**
  $x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]$

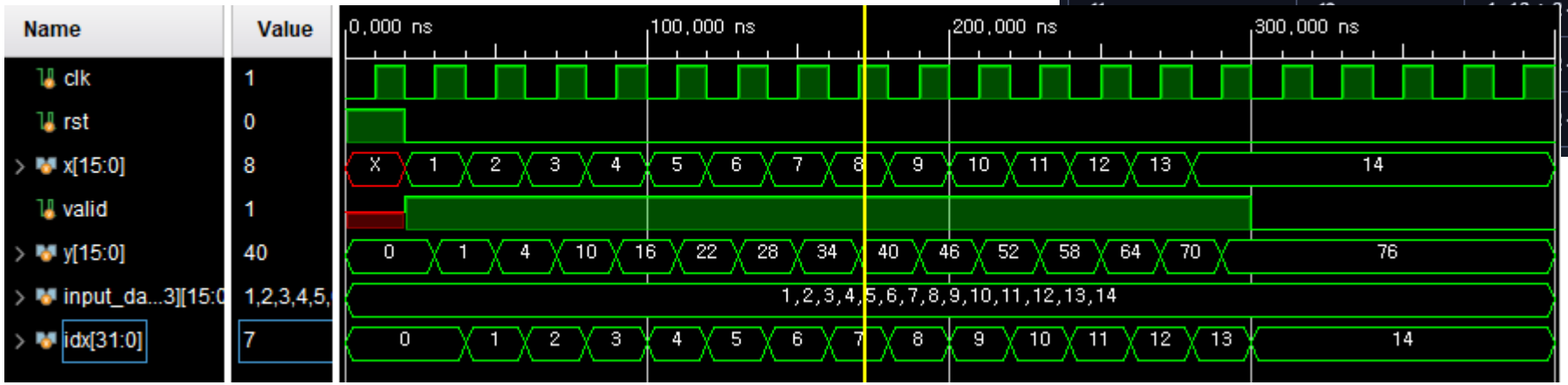
In transposed form, the output at time n is:

$$y[n] = h_0 \cdot x[n] + h_1 \cdot x[n-1] + h_2 \cdot x[n-2]$$

We assume initial conditions $x[-1] = x[-2] = 0$

Expected Output Values

Clock Cycle (n)	Input $x[n]$	Output $y[n] = h_0x[n] + h_1x[n-1] + h_2x[n-2]$
0	1	$1 \cdot 1 + 2 \cdot 0 + 3 \cdot 0 = 1$
1	2	$1 \cdot 2 + 2 \cdot 1 + 3 \cdot 0 = 4$
2	3	$1 \cdot 3 + 2 \cdot 2 + 3 \cdot 1 = 10$
3	4	$1 \cdot 4 + 2 \cdot 3 + 3 \cdot 2 = 16$
4	5	$1 \cdot 5 + 2 \cdot 4 + 3 \cdot 3 = 22$
5	6	$1 \cdot 6 + 2 \cdot 5 + 3 \cdot 4 = 28$
6	7	$1 \cdot 7 + 2 \cdot 6 + 3 \cdot 5 = 34$
7	8	$1 \cdot 8 + 2 \cdot 7 + 3 \cdot 6 = 40$
8	9	$1 \cdot 9 + 2 \cdot 8 + 3 \cdot 7 = 46$
9	10	$1 \cdot 10 + 2 \cdot 9 + 3 \cdot 8 = 52$
10	11	$1 \cdot 11 + 2 \cdot 10 + 3 \cdot 9 = 58$
		$\cdot 11 + 3 \cdot 10 = 64$
		$\cdot 12 + 3 \cdot 11 = 70$
		$\cdot 13 + 3 \cdot 12 = 76$



Design Example: FIR Polyphase

- **FIR 병렬화(2×) 수학적 유도:**
  - 짝수/홀수 샘플 분해(Serial→2-Parallel; S2P) 와 다항상(Polyphase) 분해

1) 원식

단일 FIR (3-tap, transposed든 direct든 동일):

$$y[n] = a\,x[n] + b\,x[n-1] + c\,x[n-2]$$

짝수/홀수 시퀀스를 정의하자:

$$x_e[k] = x[2k], \qquad x_o[k] = x[2k+1]$$

$$y_e[k] = y[2k], \qquad y_o[k] = y[2k+1]$$

2) 짝/홀 인덱스별로 전개

짝수 출력

$$\begin{aligned} y_e[k] &= y[2k] \\ &= a\,x[2k] \;+\; b\,x[2k-1] \;+\; c\,x[2k-2] \\ &= a\,x_e[k] \;+\; b\,x_o[k-1] \;+\; c\,x_e[k-1] \end{aligned}$$

홀수 출력

$$\begin{aligned} y_o[k] &= y[2k+1] \\ &= a\,x[2k+1] \;+\; b\,x[2k] \;+\; c\,x[2k-1] \\ &= a\,x_o[k] \;+\; b\,x_e[k] \;+\; c\,x_o[k-1] \end{aligned}$$

Design Example: FIR Polyphase

3) Polyphase(다항상) 형태로 쓰기

저역률(절반 속도, k-도메인) 지연 연산자를 z^{-1}라 하면,

$$y_e[k] = (a + c\,z^{-1})\,x_e[k] \;+\; (z^{-1}b)\,x_o[k],$$
$$y_o[k] = (b)\,x_e[k] \;+\; (a + c\,z^{-1})\,x_o[k].$$

행렬 형태:

$$\begin{bmatrix} y_e \\ y_o \end{bmatrix} = \underbrace{\begin{bmatrix} a + c\,z^{-1} & z^{-1}b \\ b & a + c\,z^{-1} \end{bmatrix}}_{\text{2-parallel polyphase}} \begin{bmatrix} x_e \\ x_o \end{bmatrix}$$

이게 의미하는 구현은:

- **서브필터** $E_0(z) = a + c\,z^{-1}$ (2-tap) 과 $E_1(z) = b$ (1-tap)을 준비하고,
- 짝수 출력은 $E_0\{x_e\} + z^{-1}\,E_1\{x_o\}$,
- 홀수 출력은 $E_1\{x_e\} + E_0\{x_o\}$,
- 그다음 $[y_e, y_o]$를 **P2S**로 인터리브하면 원래 단일 FIR의 $y[n]$와 **완전히 동일**해.

> 즉, **2개의 반속 필터 + 교차 경로의 (저속) 1샘플 지연**이 들어가면 2× 병렬화가 수학적으로 정확히 성립.

Design Example: FIR Polyphase

4) "같은 FIR 두 개"를 그냥 붙이면 안 되는 이유

많이 하는 착각: 짝수 스트림을 FIR#0, 홀수 스트림을 FIR#1에 **같은 계수** (a, b, c) 로 독립 처리하고 인터리브하면 될 것 같지만,

- 위 전개식에서 보듯 **짝수 출력에는** $x_o[k-1]$ **(홀수의 지연)가 들어가고, 홀수 출력에는** $x_e[k]$ **(짝수의 현재)가 들어간다.**
- 즉 **서로의 스트림이 교차로 섞인다**(cross-term).
  두 코어를 완전히 독립으로 돌리면 이 교차항을 놓쳐서 **단일 FIR과 동일 출력이 안 나옴**.

정확히 하려면 위 **polyphase 구조**처럼

- 한 서브필터는 $[a, c]$ (1-지연 포함 2-tap),
- 다른 서브필터는 $[b]$ (스칼라 게인),
- 그리고 **짝↔홀 간에 하나의 저속 지연**을 정확히 넣어줘야 해.

Design Example: FIR Polyphase

5) 빠른 감산 예시 (a=1, b=2, c=3)

$$y_e[k] = (1 + 3z^{-1})x_e[k] + (z^{-1} \cdot 2)\, x_o[k],$$
$$y_o[k] = 2\, x_e[k] + (1 + 3z^{-1})x_o[k].$$

이대로 구현하고 $[y_e, y_o]$를 인터리브하면 단일식 $y[n] = x[n] + 2x[n-1] + 3x[n-2]$와 일치.

Direct and Polyphase FIR

- **Direct (SISO) and "2x병렬화 (Polyphase)" 구현**
  - 2× 병렬화(S2P/E0/E1/P2S) : 하나의 입력 스트림을 짝·홀로 쪼개 내부에서 병렬 처리해 타이밍/Fmax를 개선하는 구현 최적화 (입력/출력 스트림 수는 여전히 1개(SISO), 기능은 같고 구조만 바꾼 것)
  - 단순 SISO 필터링과 처리량 (Throughput) 은 동일.
  - Polyphase의 진짜 가치는 샘플레이트 변환이나 멀티채널/병렬화 상황에서 나타남.
    - Decimation(다운샘플링): 필터 계수를 M개의 polyphase branch로 나누면, 필요한 출력만 계산하게 됨. (계산량이 1/M 수준으로 줄어듬)
    - Interpolation(업샘플링): 입력 사이에 삽입된 L-1개의 0 샘플까지 곱셈을 수행할 필요 없이 필터를 L개의 polyphase branch로 나누면, 각 branch가 실제 유효한 입력 샘플만 처리.

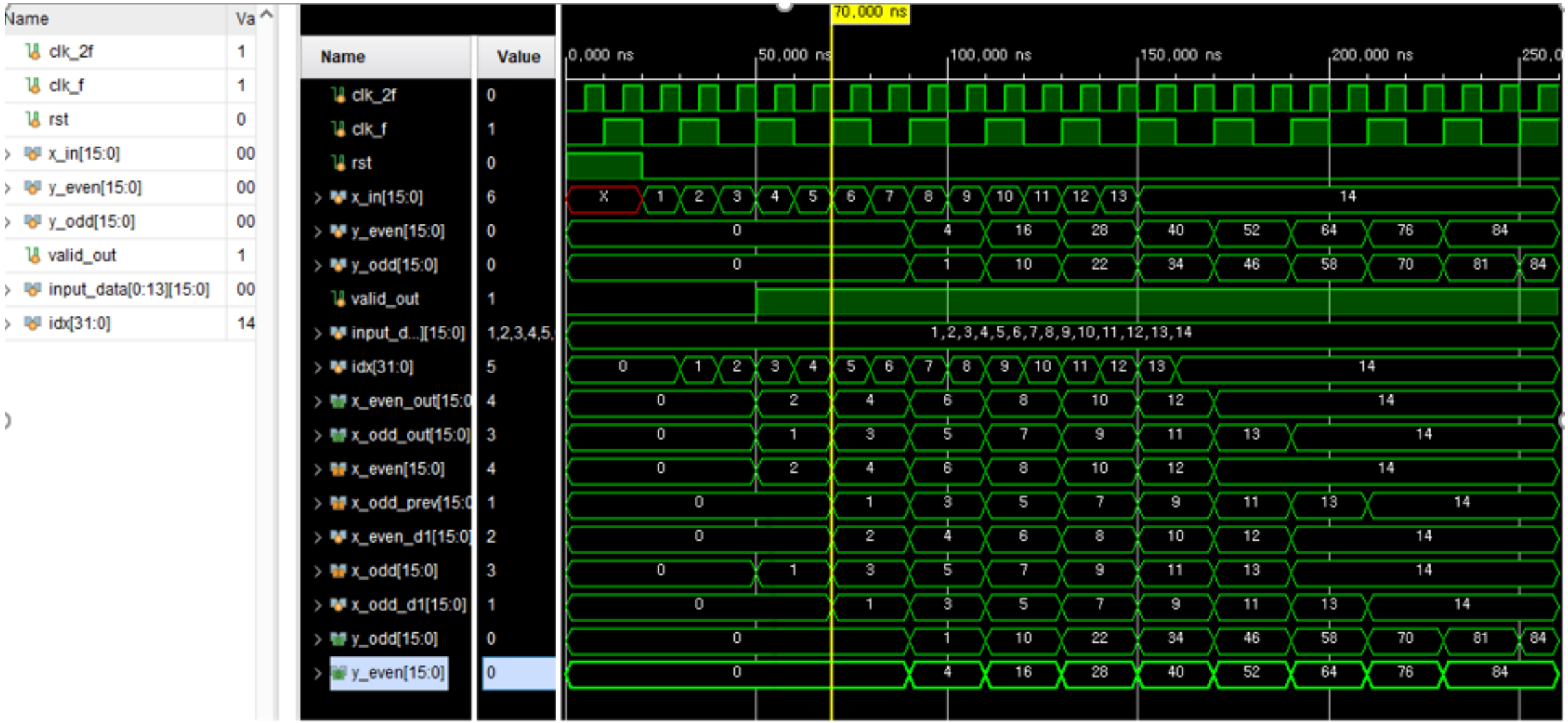
Polyphase FIR code

- 기본 가정
  - 입력 클럭 (2f): 외부에서 클럭당 1개 샘플 → 초당 2개 샘플 입력
  - 출력 클럭 (2f): FIR 결과를 순서대로 출력
  - 내부 FIR 처리 클럭 (f): 병렬 FIR 필터 2개가 f 클럭에서 동작
  - 목표: 입력 @2f → 병렬 FIR @f → 출력 @2f (or Choose one @f)

x_in (2f clk) —▶—

- $y[2k] = h_0 \cdot x[2k] + h_1 \cdot x[2k-1] + h_2 \cdot x[2k-2]$
- $y[2k+1] = h_0 \cdot x[2k+1] + h_1 \cdot x[2k] + h_2 \cdot x[2k-1]$

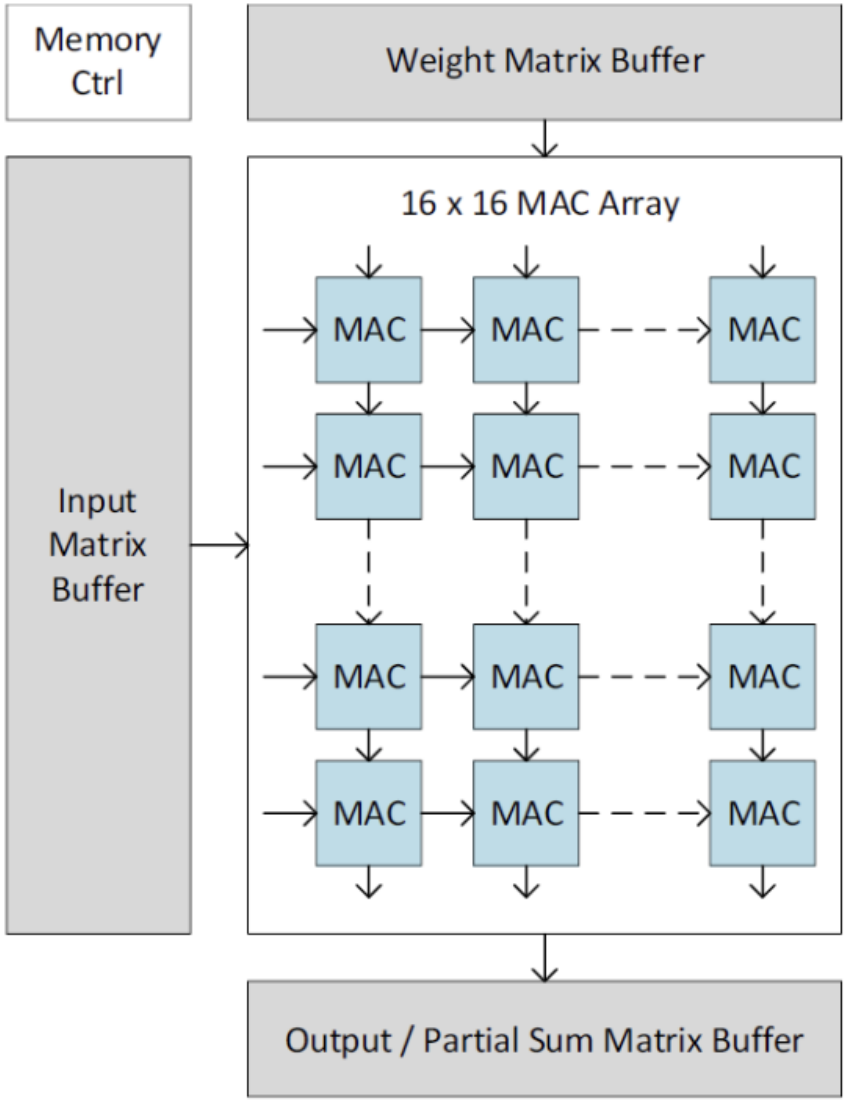
FIR 필터	입력	계수	출력 수식
FIR0	$x[2k], x[2k-2], x[2k-1]$	h_0, h_2, h_1	$y[2k] = h_0x[2k] + h_1x[2k-1] + h_2x[2k-2]$
FIR1	$x[2k+1], x[2k], x[2k-1]$	h_0, h_1, h_2	$y[2k+1] = h_0x[2k+1] + h_1x[2k] + h_2x[2k-1]$


Name
Va
clk_2f
1
clk_f
1
rst
0
x_in[15:0]
00
y_even[15:0]
00
y_odd[15:0]
00
valid_out
1
input_data[0:13][15:0]
00
idx[31:0]
14
Name
Value
clk_2f
0
clk_f
1
rst
0
x_in[15:0]
6
y_even[15:0]
0
y_odd[15:0]
0
valid_out
1
input_d...][15:0]
1,2,3,4,5,
idx[31:0]
5
x_even_out[15:0
4
x_odd_out[15:0]
3
x_even[15:0]
4
x_odd_prev[15:0
1
x_even_d1[15:0]
2
x_odd[15:0]
3
x_odd_d1[15:0]
1
y_odd[15:0]
0
y_even[15:0]
0
70,000 ns
0,000 ns
50,000 ns
100,000 ns
150,000 ns
200,000 ns
1,2,3,4,5,6,7,8,9,10,11,12,13,14

BLANK

Systolic Array

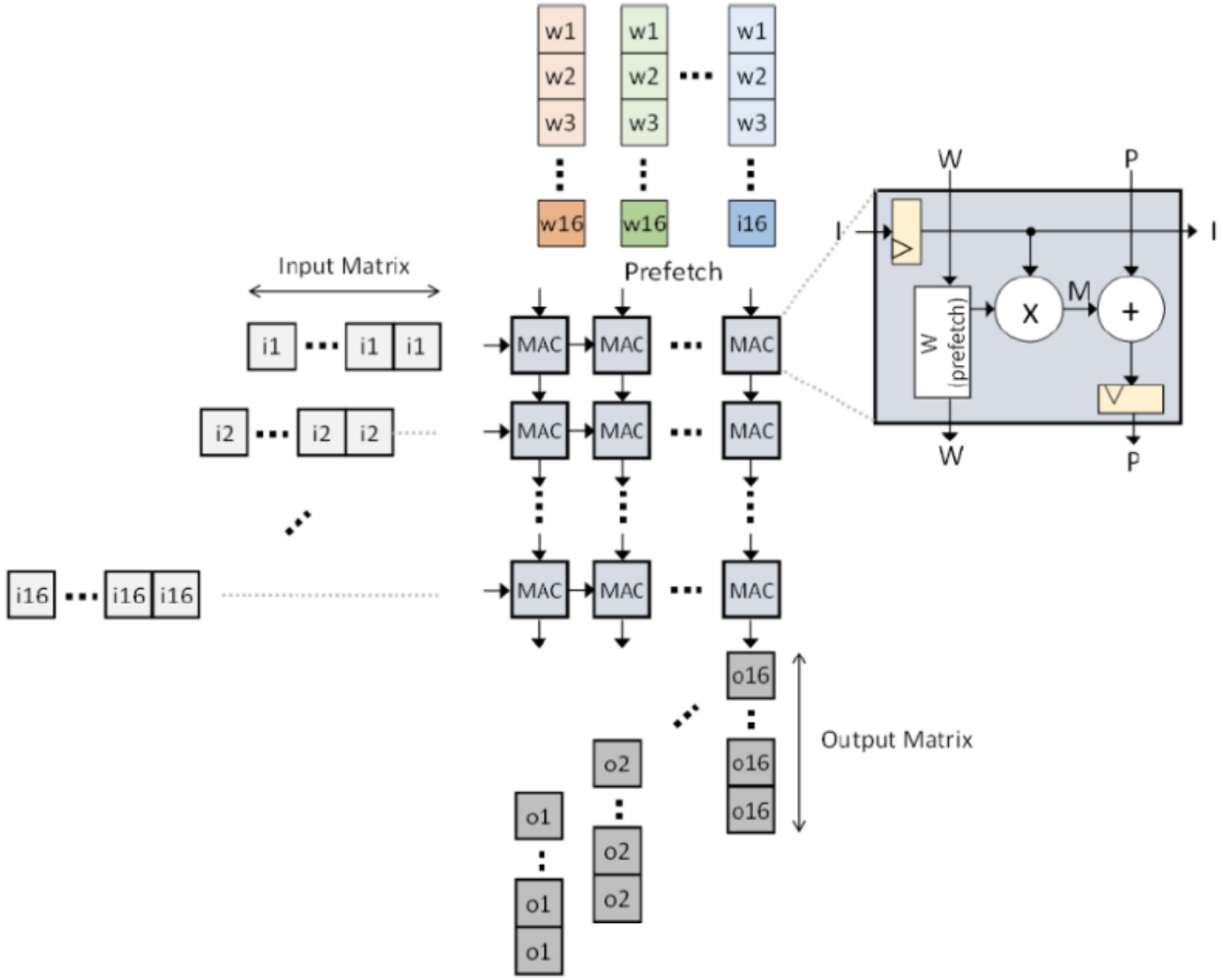
- Systolic MAC array

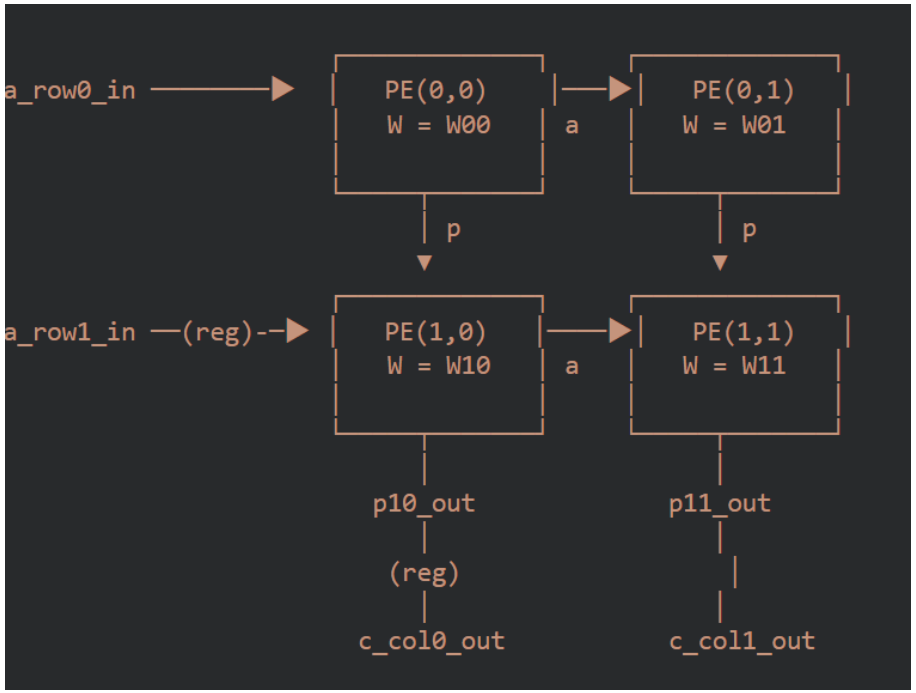


https://k-j-w.tistory.com/99

Systolic Array

- Data flow





https://k-j-w.tistory.com/99

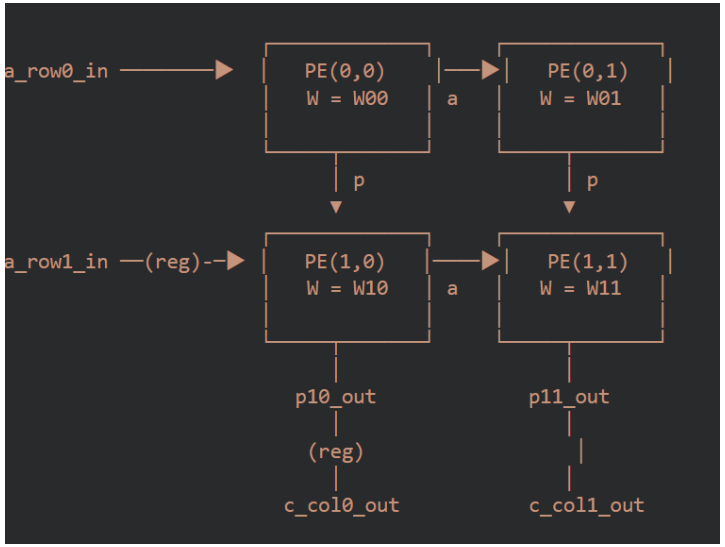
2x2 systolic array: Example

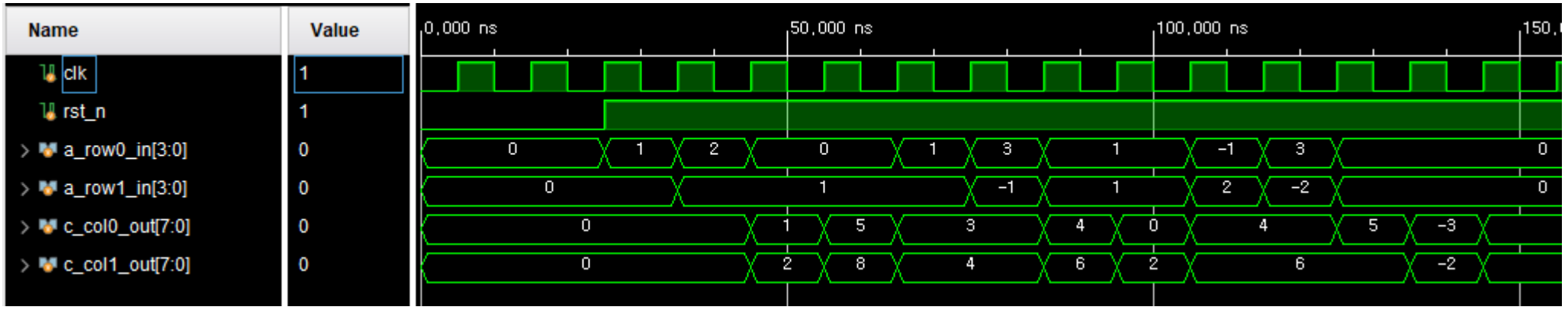
```python
import numpy as np

W  = np.array([[1,2], [3,4]])

A0 = np.array([[1,0], [2,1]])
A1 = np.array([[0,1], [0,1]])
A2 = np.array([[1,1], [3,-1]])
A3 = np.array([[1,1], [1,1]])
A4 = np.array([[-1,2], [3,-2]])
A0@W, A1@W, A2@W, A3@W, A4@W
```

```
(array([[1, 2],
        [5, 8]]),
 array([[3, 4],
        [3, 4]]),
 array([[4, 6],
        [0, 2]]),
 array([[4, 6],
        [4, 6]]),
 array([[ 5,  6],
        [-3, -2]]))
```





```
 module pe #(
    parameter int A_W = 4,
    parameter int P_W = 8,
    parameter logic signed [A_W-1:0] W_INIT = 4'sd1
)(
    input  logic clk,
    input  logic rst_n,

    input  logic signed [A_W-1:0] ain,
    input  logic signed [P_W-1:0] pin,

    output logic signed [A_W-1:0] aout,
    output logic signed [P_W-1:0] pout
);

    logic signed [A_W-1:0] w_reg;

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            w_reg <= W_INIT;
            aout  <= '0;
            pout  <= '0;
        end else begin
            // 좌->우 데이터 전달(1-cycle pipeline)
            aout <= ain;
            // MAC 누적(1-cycle pipeline)
            pout <= pin + (w_reg * ain);
        end
    end

endmodule
```

```
module systolic_2x2 #(
   parameter int A_W = 4,
   parameter int C_W = 8,

  // . . . (생략) . . .


   // Row1 skew(1-cycle delay)
   logic signed [A_W-1:0] a_row1_d;
   logic signed [C_W-1:0] c0_d;

   always_ff @(posedge clk or negedge rst_n) begin
      if (!rst_n) a_row1_d <= '0;
      else        a_row1_d <= a_row1_in;
   end

 // . . . (생략) . . .

   // 2x2 PE 배열
   pe #(.A_W(A_W), .P_W(C_W), .W_INIT(W00)) u_pe00 (...)
   pe #(.A_W(A_W), .P_W(C_W), .W_INIT(W01)) u_pe01 (...)
   pe #(.A_W(A_W), .P_W(C_W), .W_INIT(W10)) u_pe10 (...)
   pe #(.A_W(A_W), .P_W(C_W), .W_INIT(W11)) u_pe11 (...)

   // 출력 정렬:
   // col0(p10_out)가 col1(p11_out)보다 1클럭 빠름 -> col0를 1클럭 지연
   always_ff @(posedge clk or negedge rst_n) begin
      if (!rst_n) begin
         c0_d <= '0;
      end else begin
          c0_d <= p10_out;   // col0을 한 번 더 잡아서 늦춤
      end
   end

   assign c_col0_out = c0_d;
   assign c_col1_out = p11_out;  // col1은 레지스터 추가 없이 그대로 출력

endmodule
```

BLANK