Tapped Delay

Young Won Lim
7/24/24

Please send corrections (or suggestions) to youngwlim@hotmail.com.
This document was produced by using LibreOffice.

Young Won Lim
7/24/24

Based on

Introduction to Signal Processing

S. J. Ofranidis

The necessities in DSP C Programming

FIR Filter (A.pdf) 20191114

D Flip Flop

Considering the widely used
Edge triggered
D-type Flip Flops

Register

D_3 — D Q — Q_3

D_2 — D Q — Q_2

D_1 — D Q — Q_1

D_0 — D Q — Q_0

CLK

4 — $D_{3:0}$ 4 — $Q_{3:0}$

CLK

$D_{3:0}$ without delay
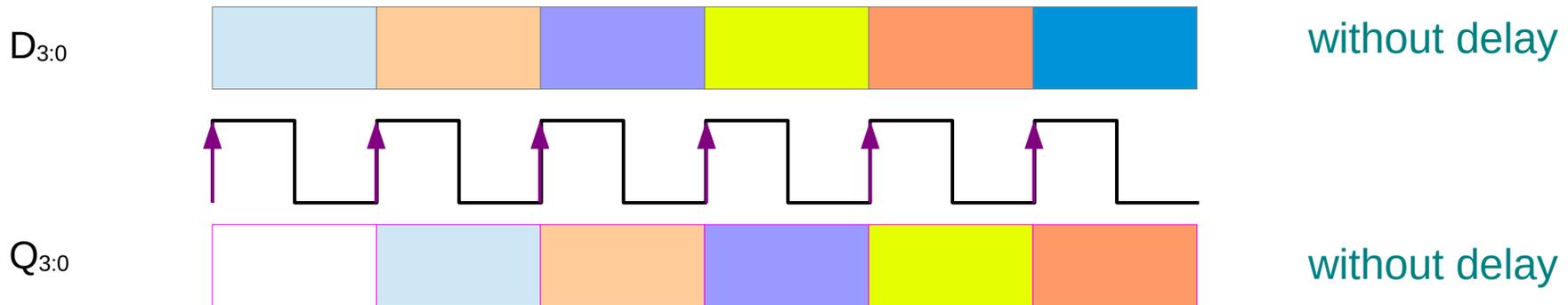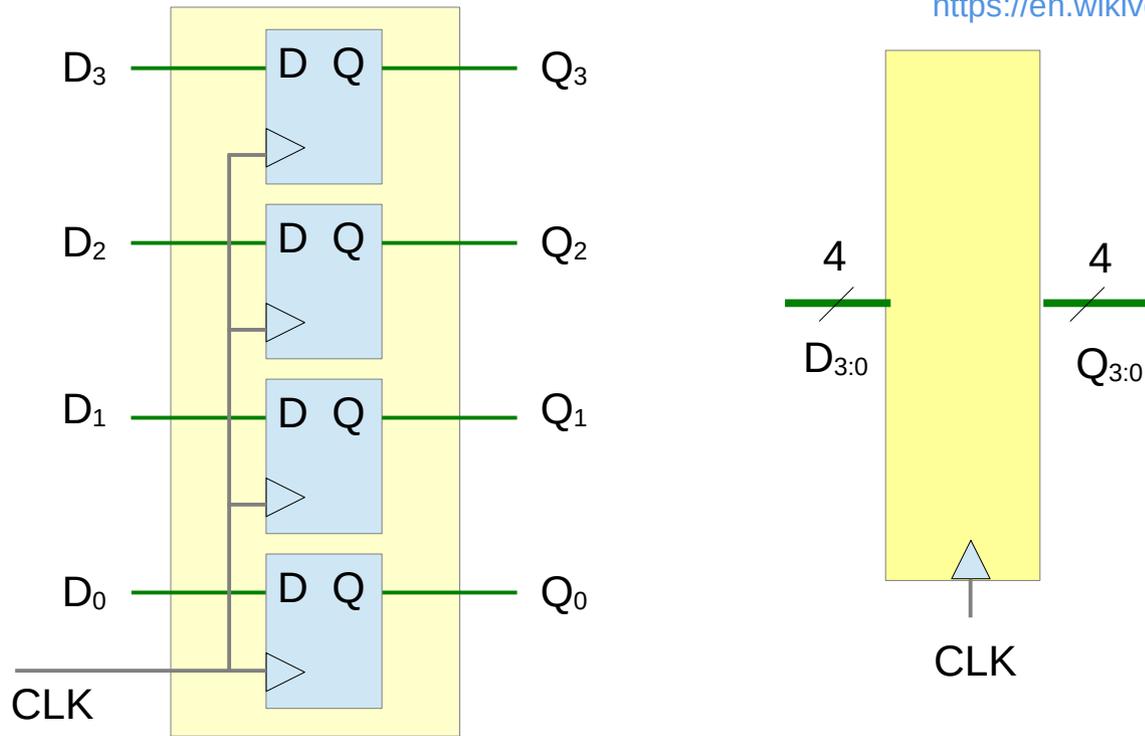
$Q_{3:0}$ without delay

Types of Timing Diagrams

a timing diagram without delays

$D_{3:0}$

no delay (ideal case)

$Q_{3:0}$

no delay (ideal case)
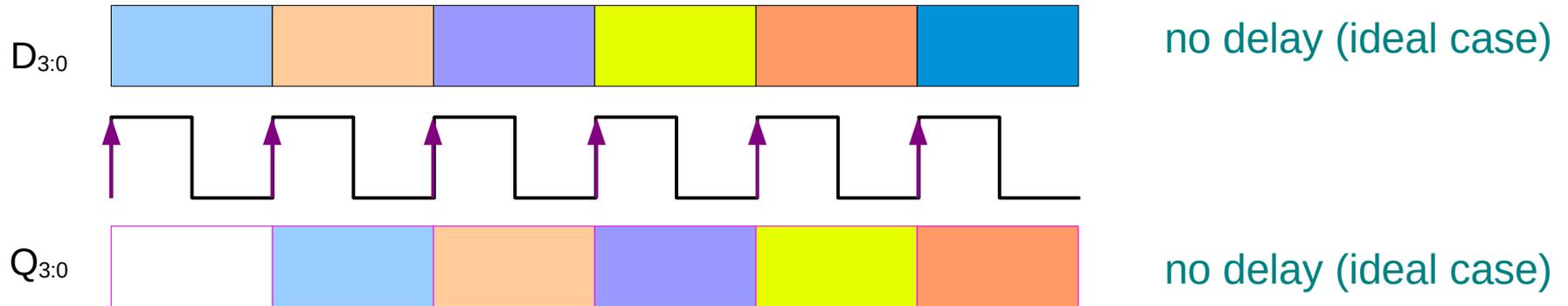
a timing diagram with delays
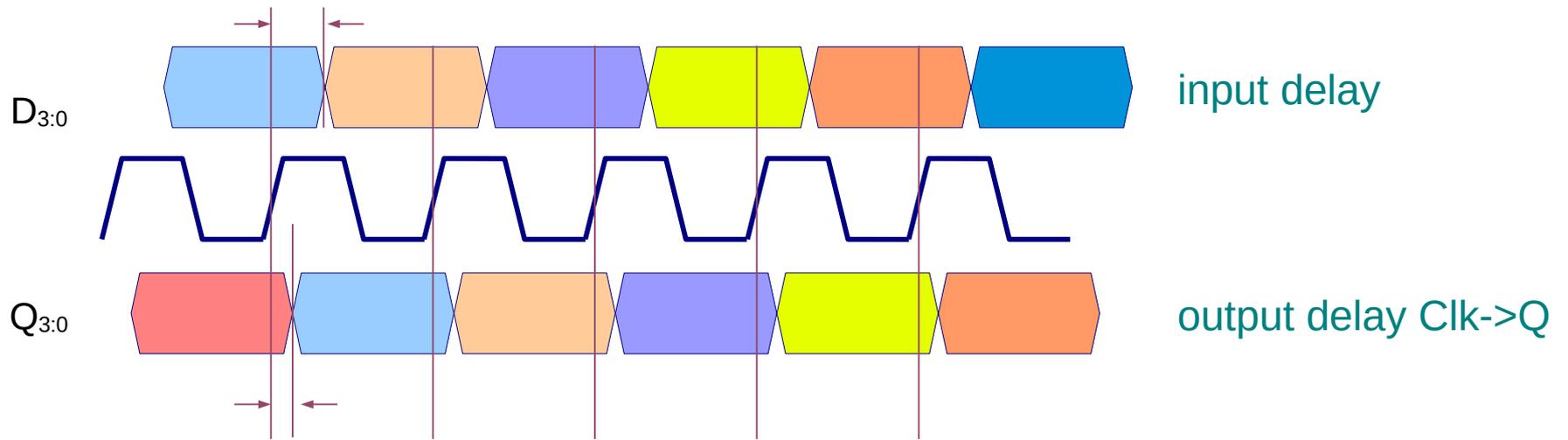
$D_{3:0}$

input delay

$Q_{3:0}$

output delay Clk->Q

Setup & Hold Time (1)

$D_{3:0}$

Inputs with delays

$Q_{3:0}$

Output with a delay

set up time   hold time
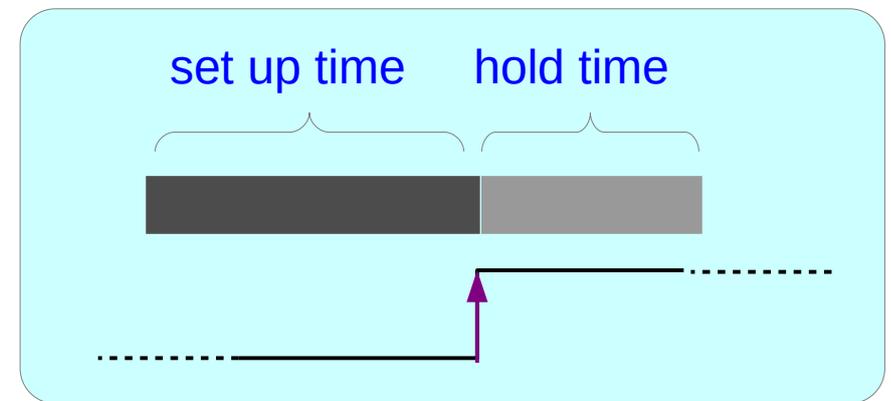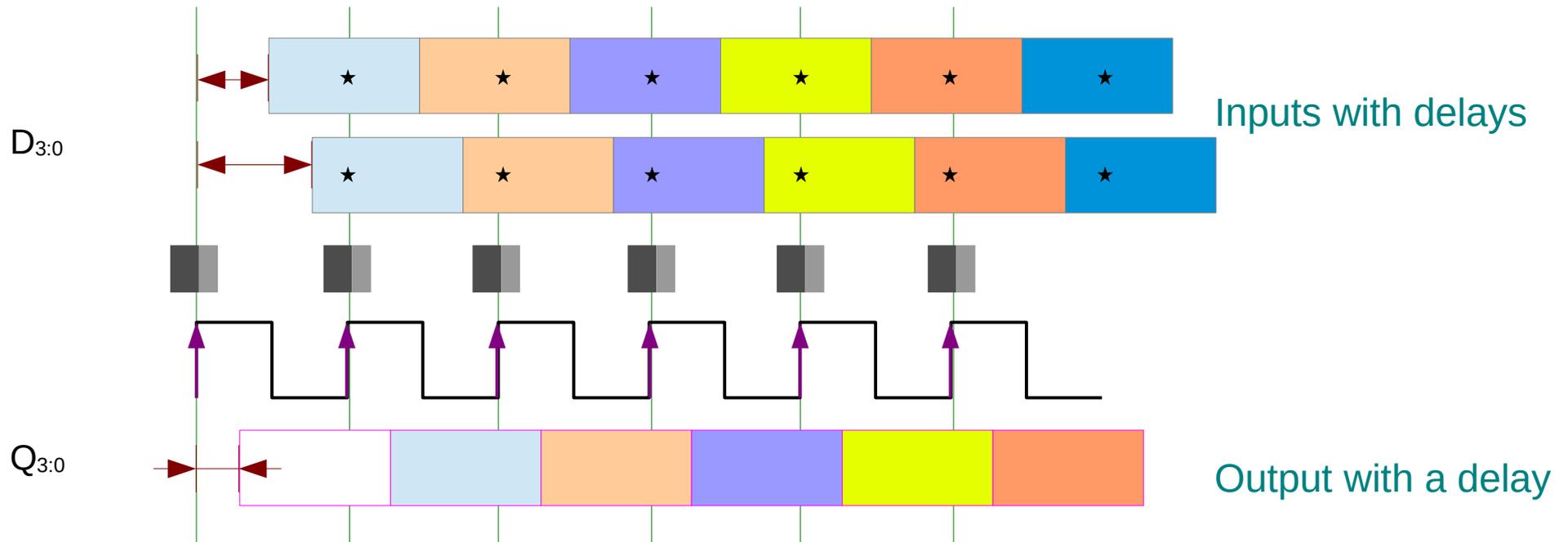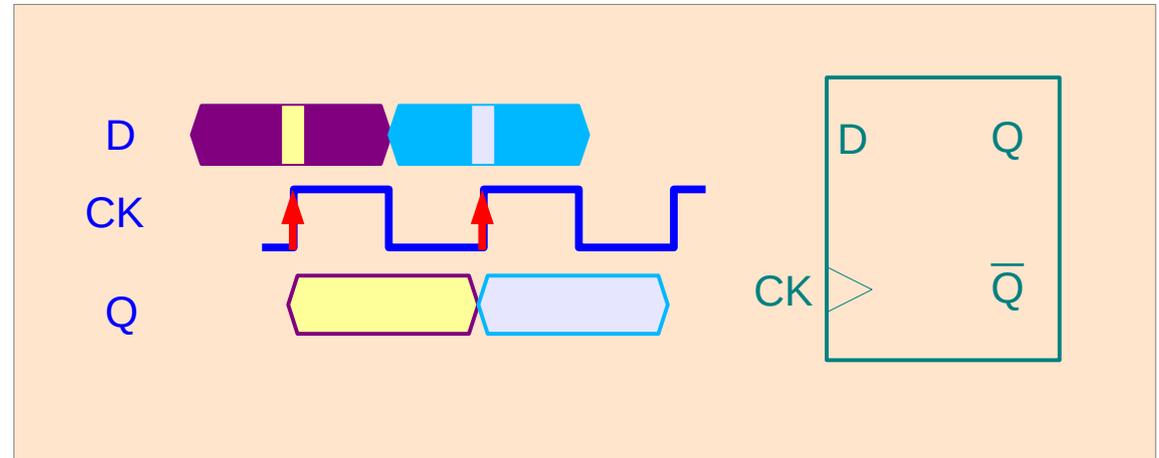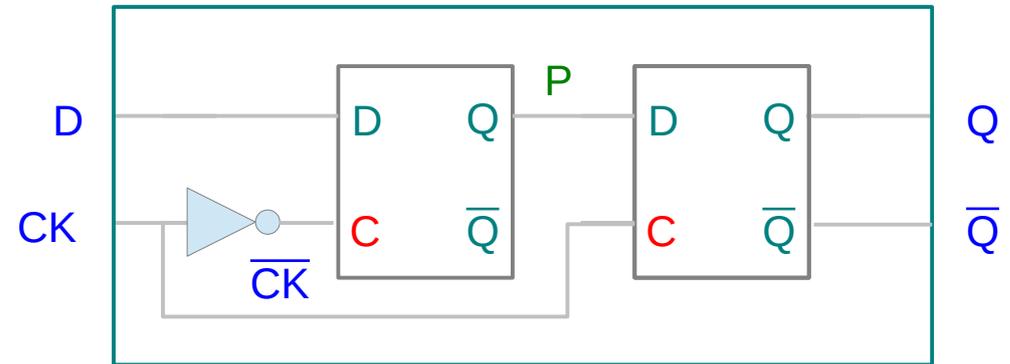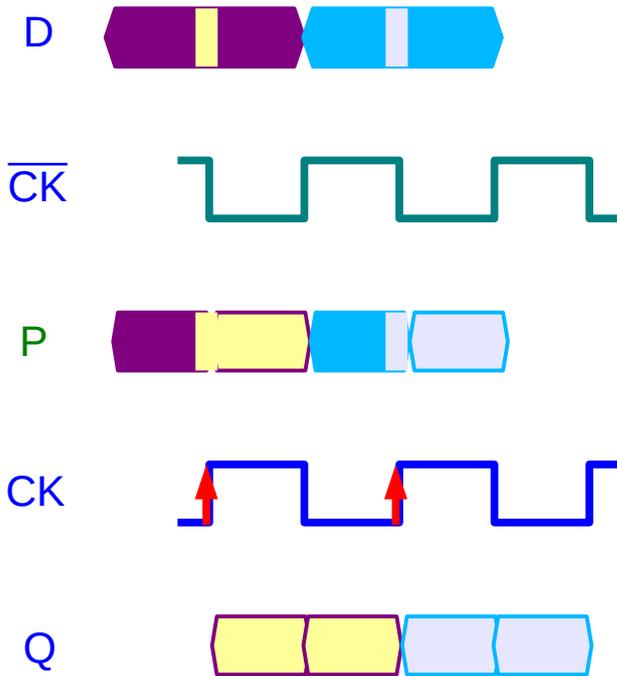
Master-Slave D FlipFlop – Rising Edge

Master D Latch

Slave D Latch

Master-Slave D FlipFlop – Rising Edge
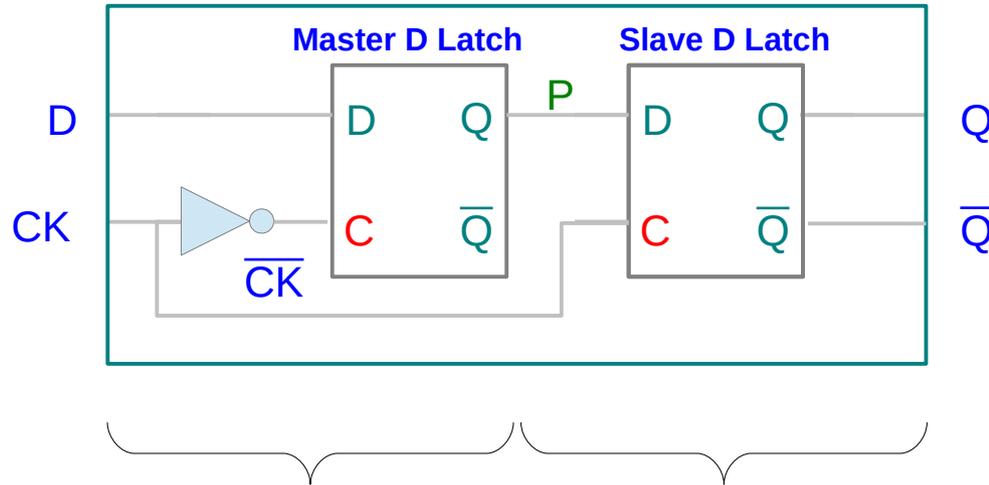
Master D Latch

(1) the current <u>input</u>
    D gets stored
    in the master latch

(2) the current <u>content</u>
    P is clocked out
    to the output Q

Using inverted clocks <u>enable</u>
(1) and (2) to be executed <u>sequentially</u>

**Tapped Delay
(1A)**

9

Young Won Lim
7/24/24
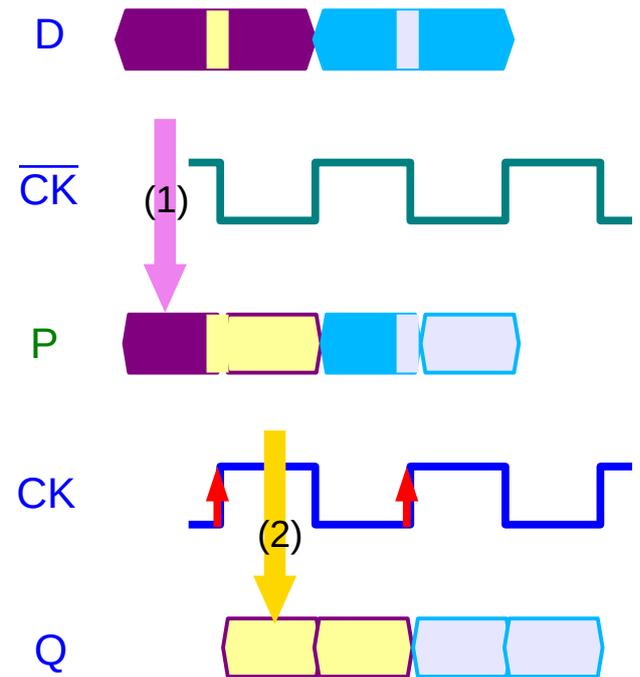
Master-Slave D FlipFlop – open and hold

Master D Latch

Master D Latch

D

$\overline{CK}$

P

CK

Q

open

hold

open

hold
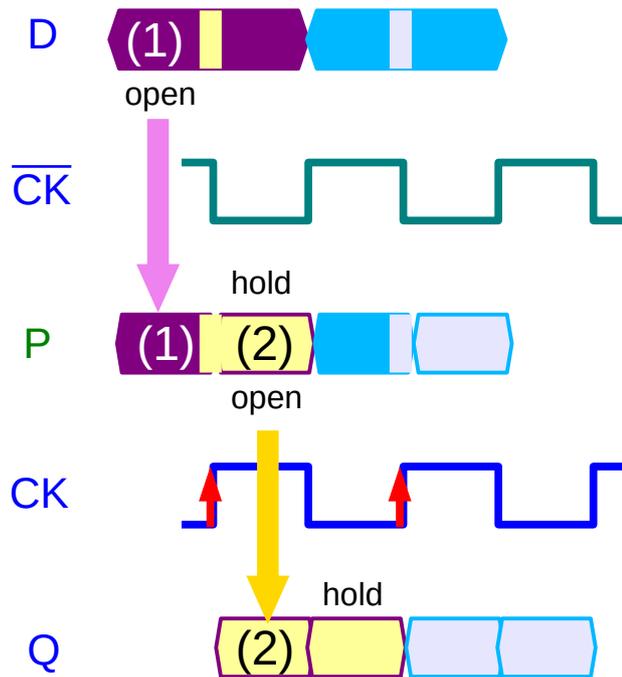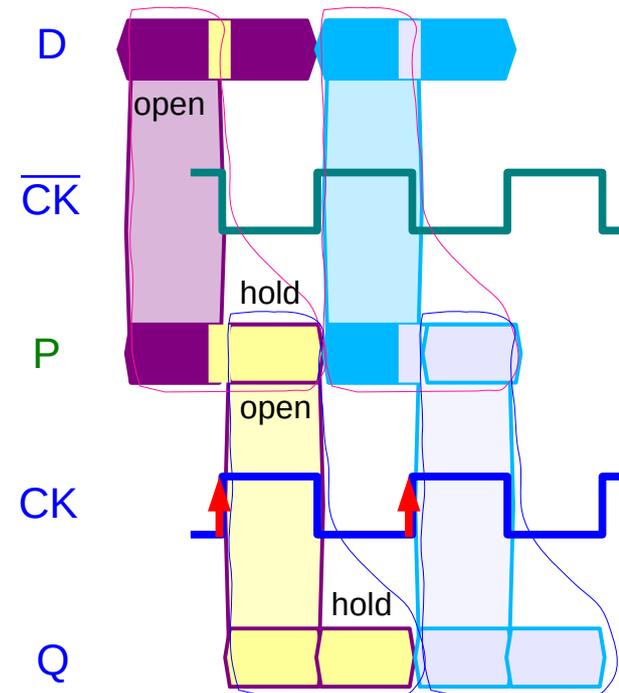
Slave D Latch

Slave D Latch

Master-Slave D FlipFlop – typical vs. ideal timing (1)

Typical Timing

Ideal Timing

Tapped Delay (1A)

Young Won Lim
7/24/24
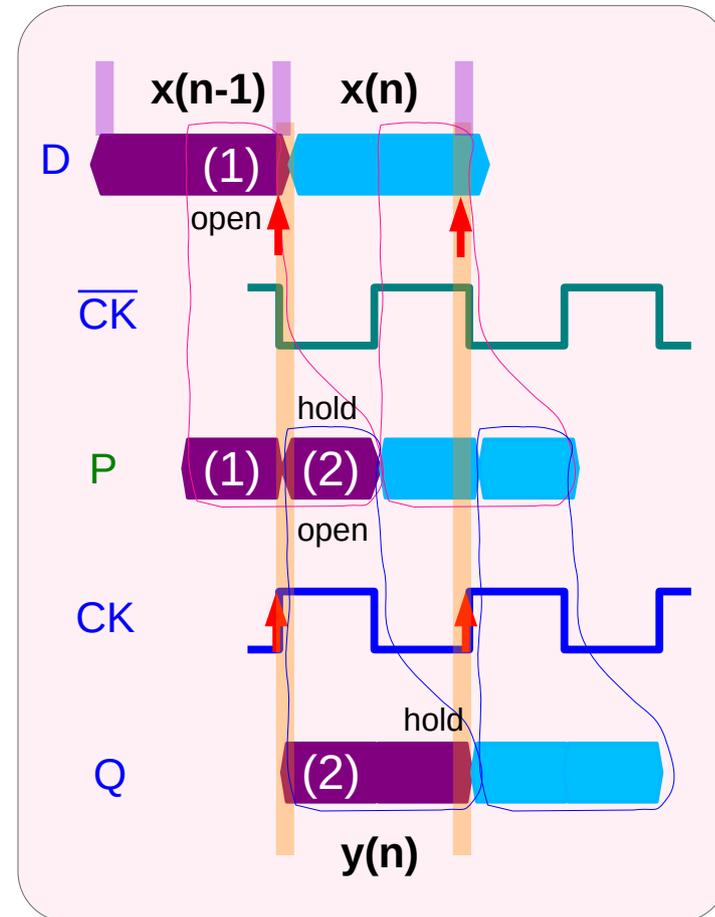
Master-Slave D FlipFlop – typical vs. ideal timing (2)

Typical Timing
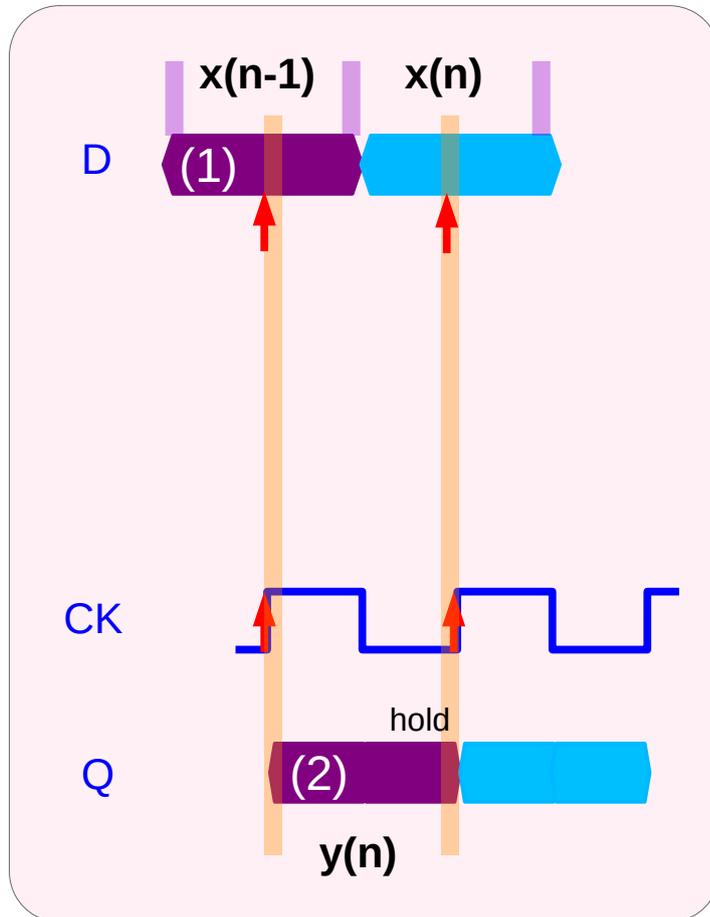
Ideal Timing
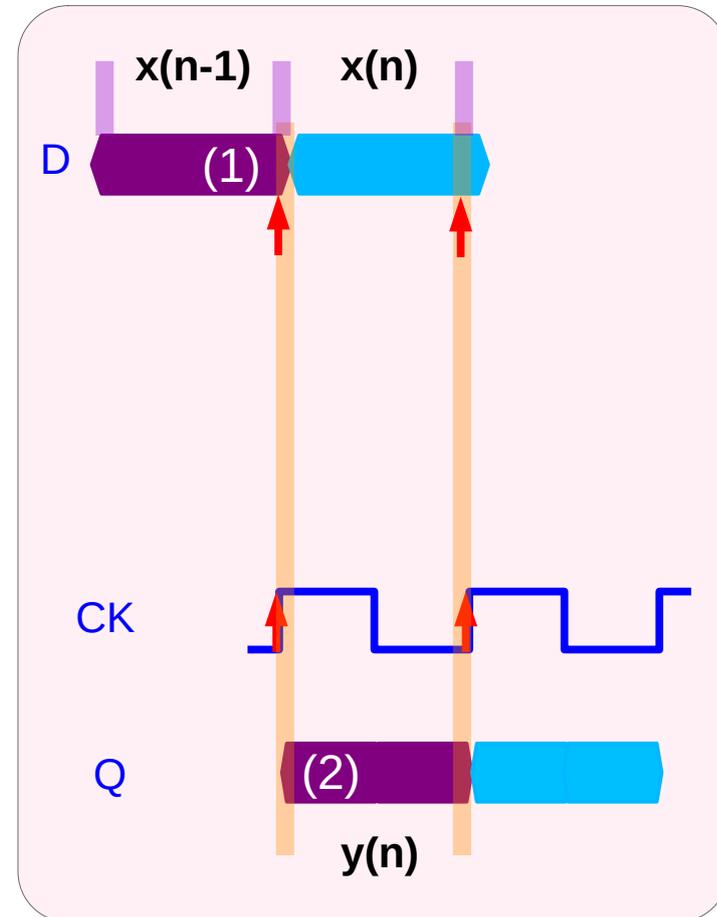
Master-Slave D FlipFlop – Rising Edge Sampling

Master D Latch

x(n)

D

$\overline{CK}$ (1)

P

w1(n)

CK

(2)

Q

y(n)

Slave D Latch

x(n)　　　　　　　　　　w1(n)　　　　　　　y(n)

Master D Latch　　**Slave D Latch**

D　　　　　　D　　Q　　P　　D　　Q　　Q

CK　　　　　C　　$\overline{Q}$　　　C　　$\overline{Q}$　　$\overline{Q}$

$\overline{CK}$

(1) the current <u>input</u>　　　(2) the current <u>content</u>
　　x(n) gets stored　　　　　　w1(n) is clocked out
　　in the master latch　　　　　to the output y(n)

Fixed point representation

fractional
numbers

Floating Point Representation

Fixed Point Representation

integer + implicit fixed scaling factor

a 2's complement number $(11110101)_2 = -11$

could represent $-11 \cdot 2^{-3} = -88$,
$\qquad\qquad\qquad\quad -11 \cdot 2^{-5} = -0.343\,75$
$\qquad\qquad\qquad\quad -11 \cdot 2^{-12} = -0.002\,685\,546\,875$

with implied scaling factors -3, -5, -12

fractional
numbers

x(n)	➡	**X**	*n-bit* wires	at time n
y(n)	➡	**Y**	*n-bit* wires	at time n
w1(n)	➡	**W**	*n-bit* registers	at time n

Memory Elements

fractional numbers

$x(n)$ ➡ **X** n-bit wires   at time n
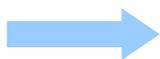
$y(n)$ ➡ **Y** n-bit wires   at time n

$w1(n)$ ➡ **W** n-bit registers   at time n ➡ memory elements
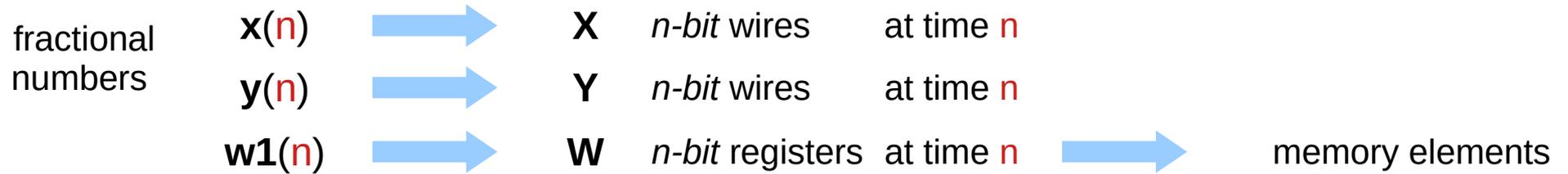
1. using the **P** <u>outputs</u> of master latches as the <u>name</u> for memory elements

2. using the **Q** <u>outputs</u> of flipflops as the <u>name</u> for memory elements
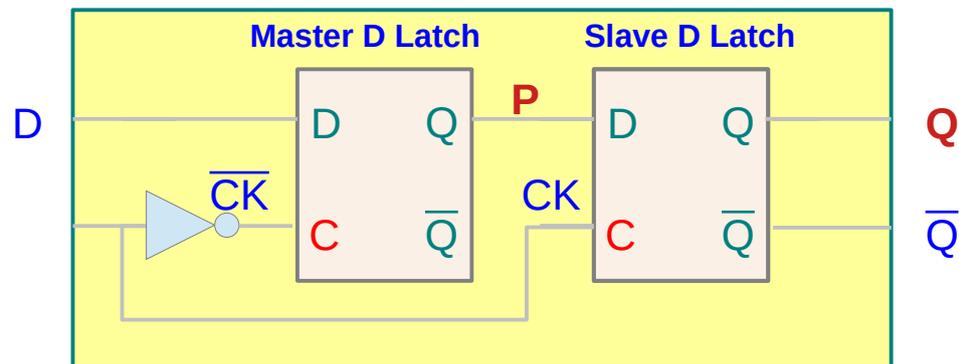
D FlipFlop using Master and Slave D Latches

(1) Using master latch outputs **P**
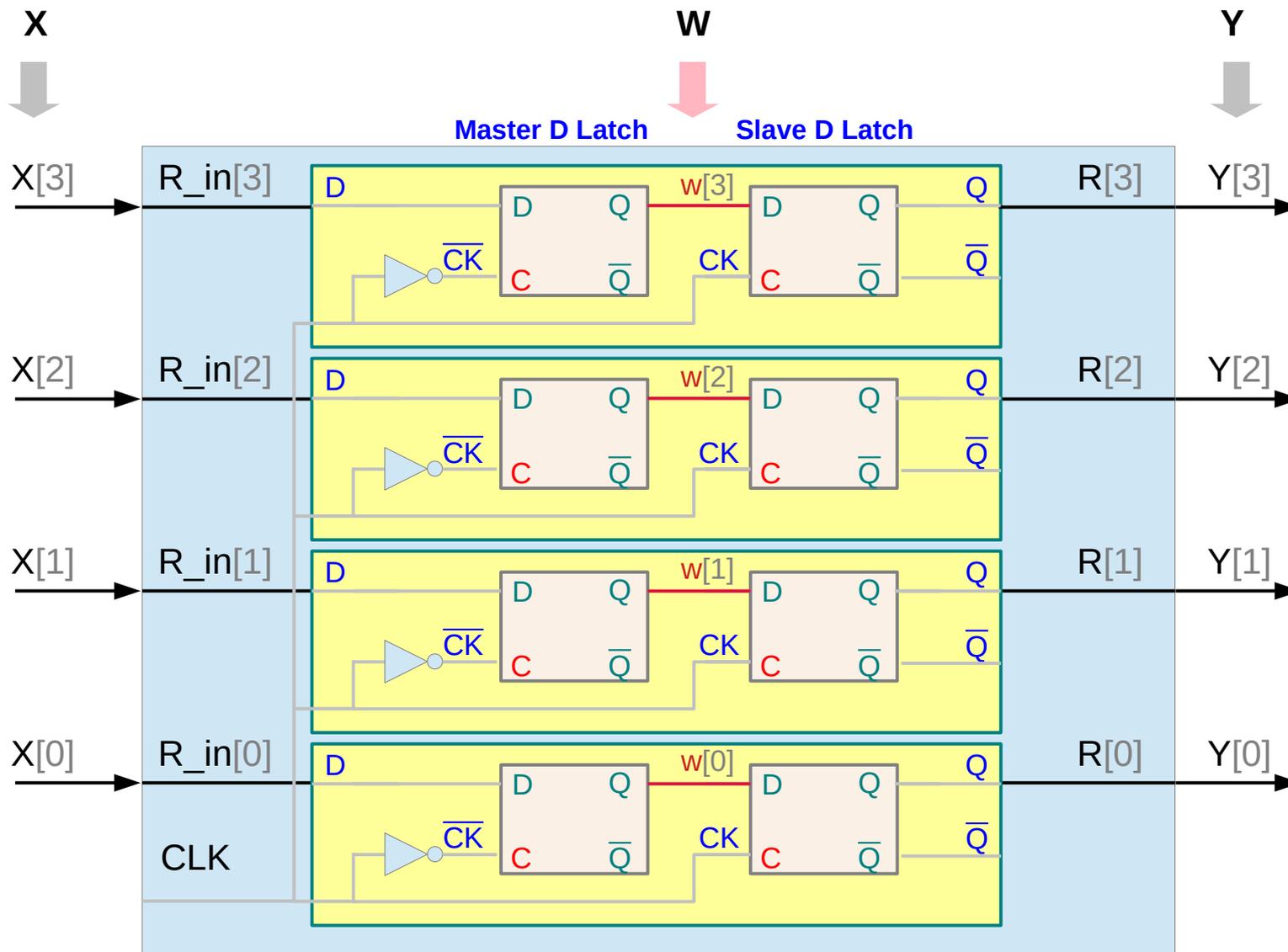
(2) Using flipflop outputs **Q**
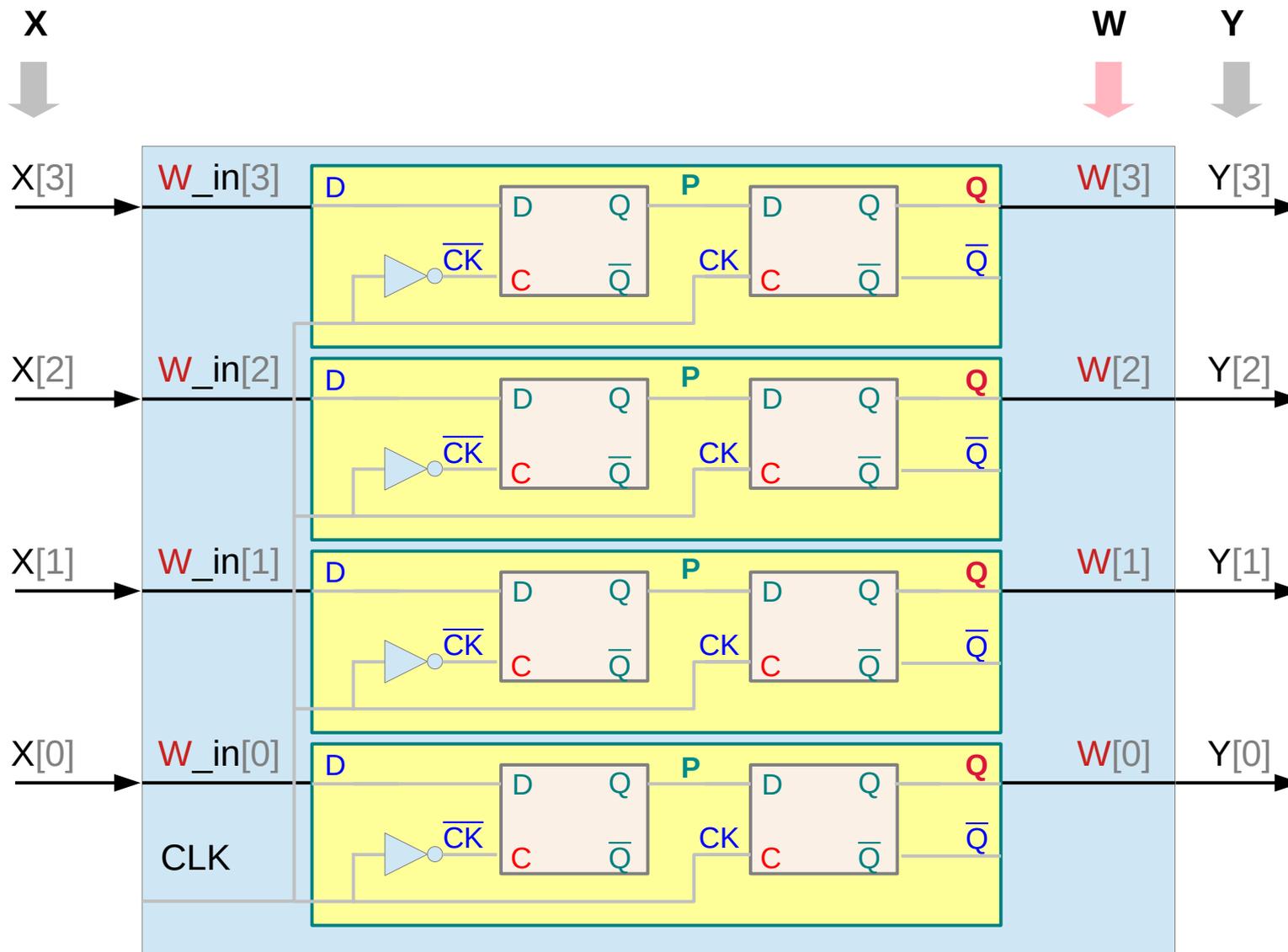
(1) 4-bit Register Using master latch outputs **P**

R_in (1) (3)

CK

P (1) (2) (3) (4)

R (2) (4)

D FlipFlop using Master and Slave D Latches

Master D Latch   Slave D Latch

D   D   Q   **P**   D   Q   Q

$\overline{CK}$ CK

C $\overline{Q}$ C $\overline{Q}$ $\overline{Q}$
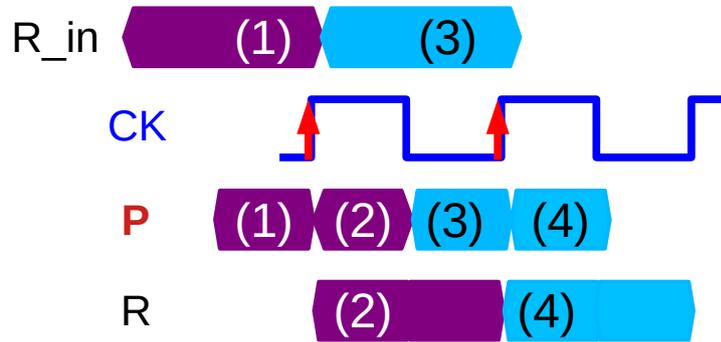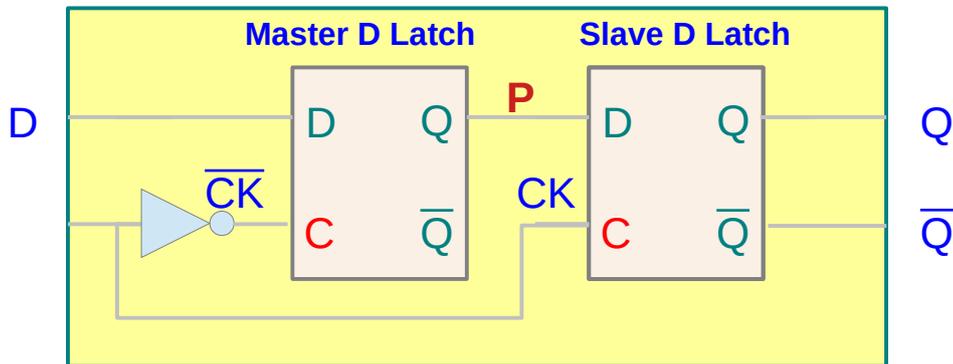
4-bit Register using 4 D flipflops

R_in[3]   D Q   R[3]

R_in[2]   D Q   R[2]

R_in[1]   D Q   R[1]

R_in[0]   D Q   R[0]

CLK
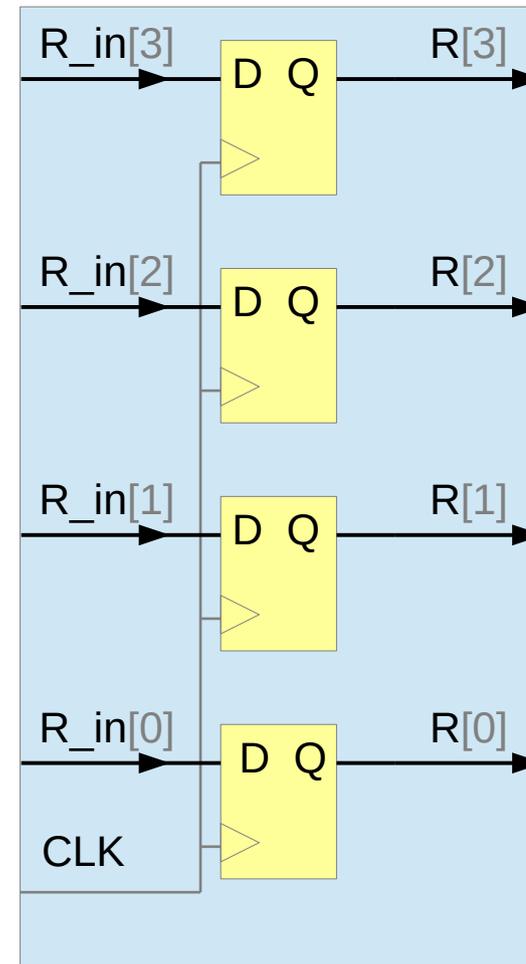
(2) 4-bit Register Using flipflop outputs **Q**
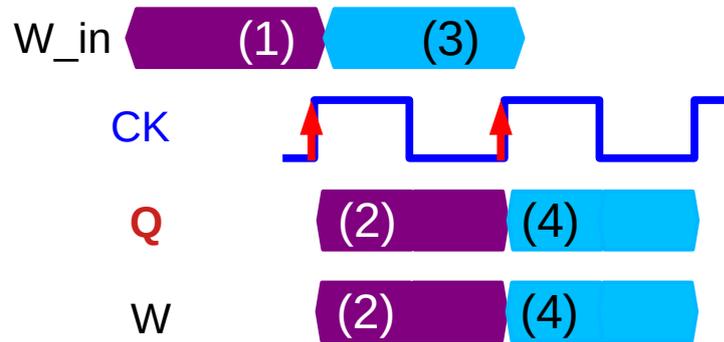
W_in (1) (3)

CK

Q (2) (4)

W (2) (4)

D FlipFlop using Master and Slave D Latches

| Master D Latch | Slave D Latch |

D

$\overline{CK}$

C $\overline{Q}$ CK   C $\overline{Q}$

P

Q

$\overline{Q}$

W_in[3]   D   Q   W[3]

W_in[2]   D   Q   W[2]

W_in[1]   D   Q   W[1]

W_in[0]   D   Q   W[1]

CLK
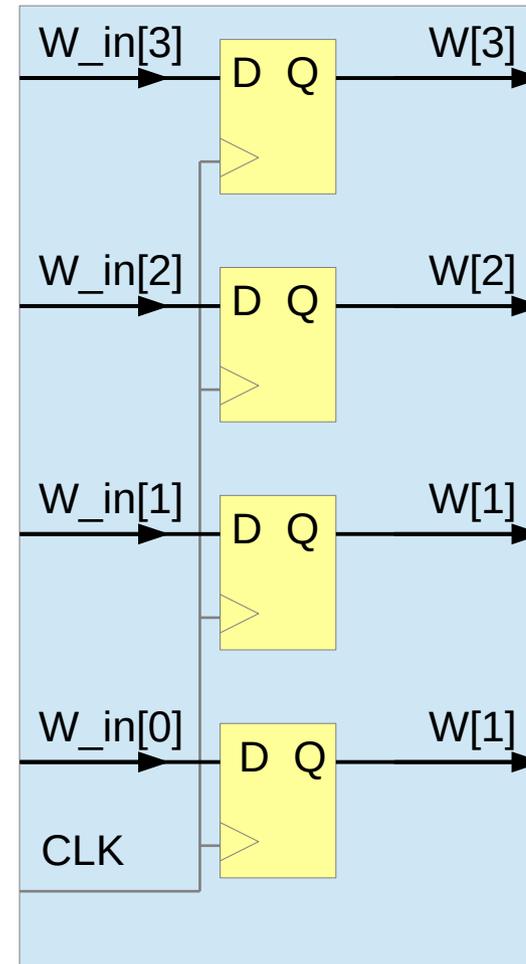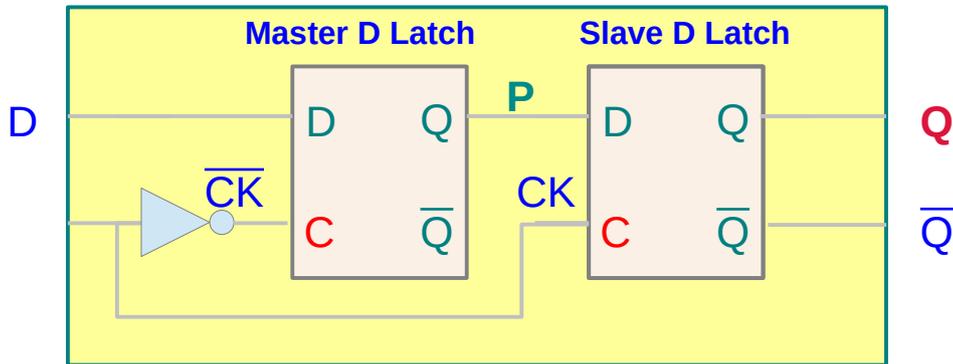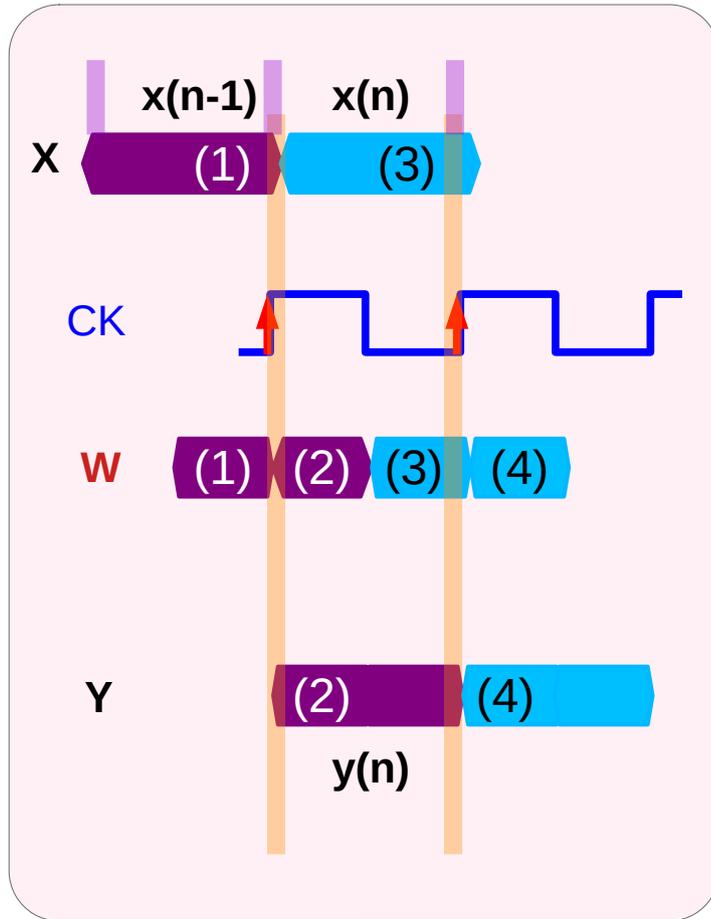
data centered view

Input

CK

Internal State

Output
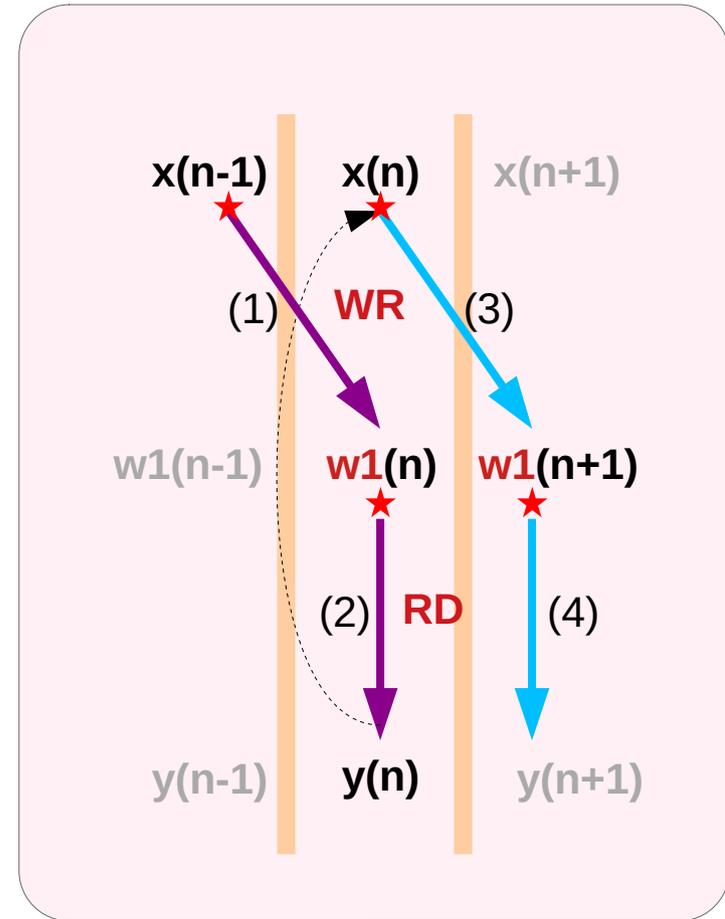
Hardware model

The same **DSP C model**

(2) Timing diagrams with flipflop outputs **Q**

data centered view

Hardware model
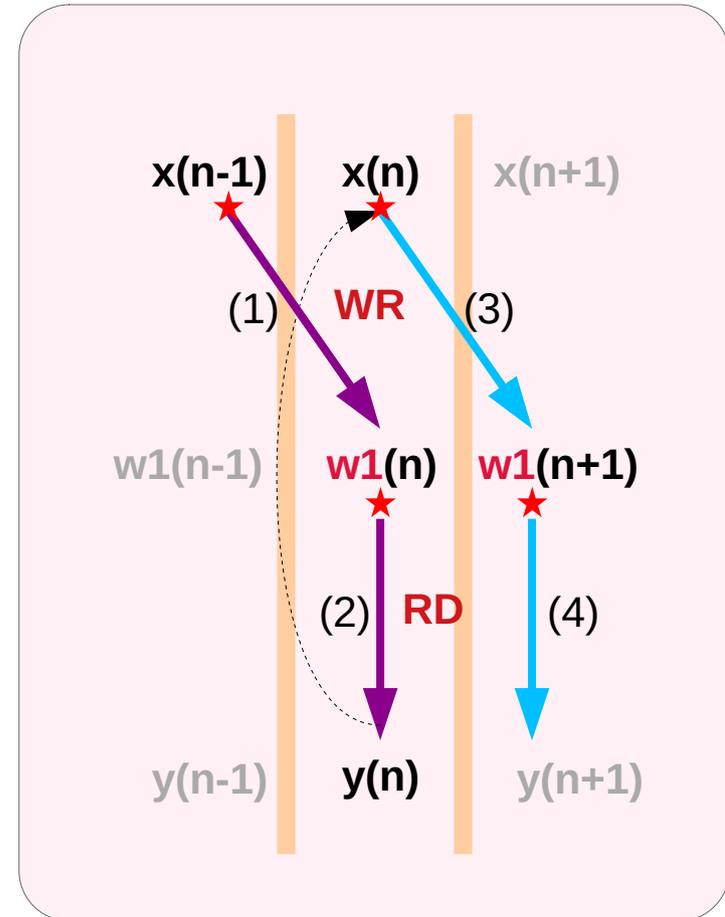
The same **DSP C model**

Data centered operations

data centered operations
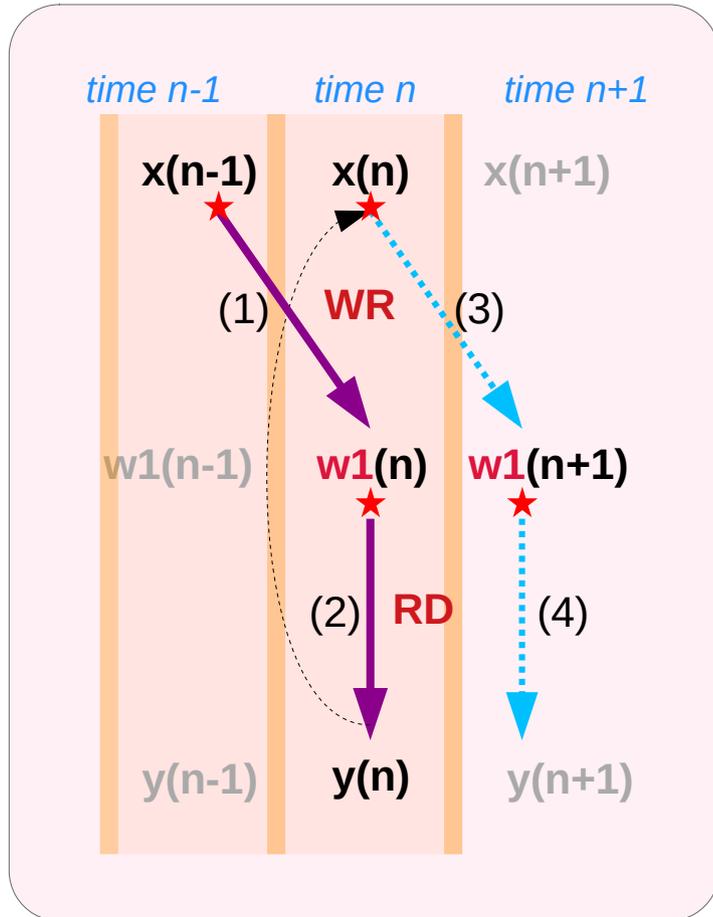
operation spans two time steps (**n-1**, **n**)

DSP C model

(1)   **w1**(n) **= x**(n-1)   *WR* **w1**(*n*)

(2)   **y**(n) = **w1**(n)   *RD* **w1**(*n*)

first **WR,** then **RD**
must avoid **RAW (read after write)** hazards

here, because the written data is read out,
yhere is no **RAW (read after write)** hazards

y(n-1) = **w1**(n-1)
——————————————   **w1**(n) = **w1**(n-1)
w1(n) **= x**(n-1)

y(n) = **w1**(n)
——————————————   **w1**(n+1) = **w1**(n)
w1(n+1) **= x**(n)

y(n+1) = **w1**(n+1)
——————————————   **w1**(n+2) = **w1**(n+1)
w1(n+2) **= x**(n+1)

y(n+2) = **w1**(n+2)

Time centered operations
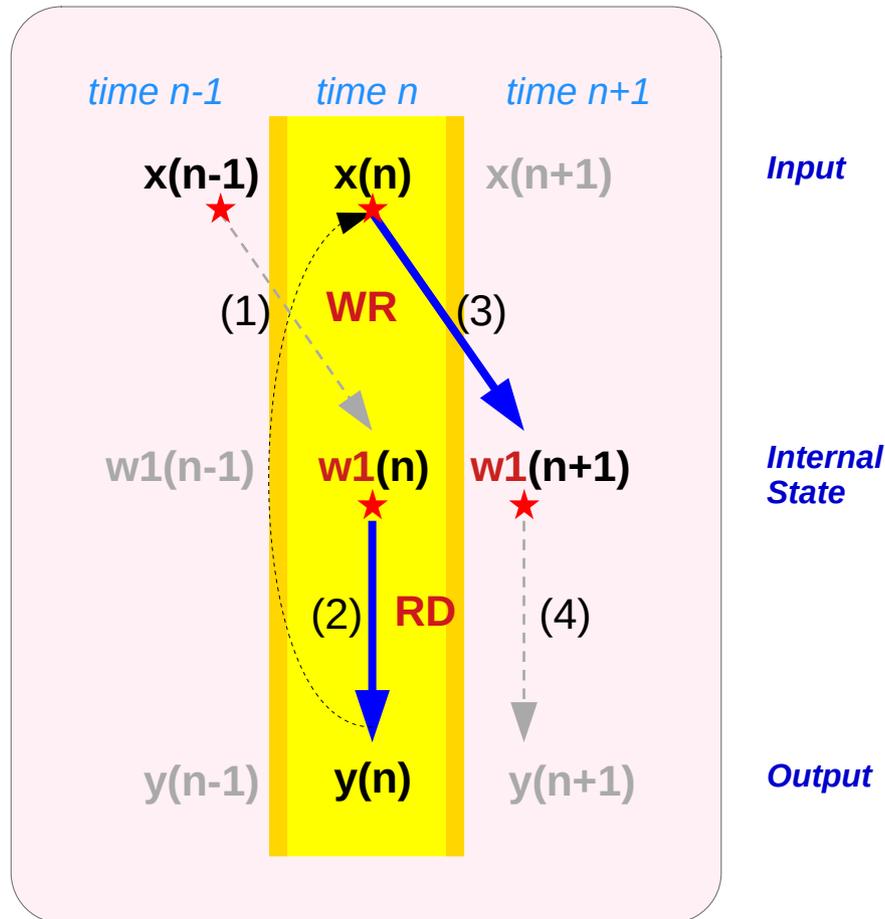
time centered operations

operation spans only one time step (**n**)

time n-1   *time n*   *time n+1*

x(n-1)   **x(n)**   x(n+1)   *Input*

(1)   **WR**   (3)

w1(n-1)   **w1(n)**   **w1(n+1)**   *Internal State*

(2)   **RD**   (4)

y(n-1)   **y(n)**   y(n+1)   *Output*

DSP C model

(2) $\mathbf{y}(n) = \mathbf{w1}(n)$ *RD **w1**(n)*

(3) $\mathbf{w1}(n+1) = \mathbf{x}(n)$ *WR **w1**(n)*

first **RD**, then **WR**
no **RAW (read after write)** hazards

$\mathbf{y}(n-1) = \mathbf{w1}(n-1)$

$\mathbf{w1}(n) = \mathbf{x}(n-1)$

$\mathbf{w1}(n) = \mathbf{w1}(n-1)$

$\mathbf{y}(n) = \mathbf{w1}(n)$

$\mathbf{w1}(n+1) = \mathbf{x}(n)$

$\mathbf{w1}(n+1) = \mathbf{w1}(n)$

$\mathbf{y}(n+1) = \mathbf{w1}(n+1)$

$\mathbf{w1}(n+2) = \mathbf{x}(n+1)$

$\mathbf{w1}(n+2) = \mathbf{w1}(n+1)$

$\mathbf{y}(n+2) = \mathbf{w1}(n+2)$

Data vs. time centered operations (1)

data centered operations

y(n-1) = **w1**(n-1)	*output from w1(n-1)*
w1(n) = **w1**(n-1)	
w1(n) = **x**(n-1)	*input to w1(n)*
y(n) = **w1**(n)	*output from w1(n)*
w1(n+1) = **w1**(n)	
w1(n+1) = **x**(n)	*input to w1(n+1)*
y(n+1) = **w1**(n+1)	*output from w1(n+1)*
w1(n+2) = **w1**(n+1)	
w1(n+2) = **x**(n+1)	*input to w1(n+2)*
y(n+2) = **w1**(n+2)	*output from w1(n+2)*

time centered operations

y(n-1) = **w1**(n-1)	*output from w1(n-1)*	*time n-1*
w1(n) = **w1**(n-1)		
w1(n) = **x**(n-1)	*input to w1(n)*	
y(n) = **w1**(n)	*output from w1(n)*	*time n*
w1(n+1) = **w1**(n)		
w1(n+1) = **x**(n)	*input to w1(n+1)*	
y(n+1) = **w1**(n+1)	*output from w1(n+1)*	*time n+1*
w1(n+2) = **w1**(n+1)		
w1(n+2) = **x**(n+1)	*input to w1(n+2)*	
y(n+2) = **w1**(n+2)	*output from w1(n+2)*	*time n+2*

(1)   **w1**(n) = **x**(n-1)   ***WR w1(n)***

(2)   y(n) = **w1**(n)   ***RD w1(n)***

(2)   y(n) = **w1**(n)   ***RD w1(n)***

(1)   **w1**(n) = **x**(n)   ***WR w1(n)***

W1 = X	***WR W1***	
Y = **W1**	***RD W1***	

Y = **W1**	***RD W1***	
W1 = X	***WR W1***	

← ***Register Transfer Level***

Data vs. time centered operations (2)

data centered operations

y(n-1) = **w1**(n-1)

w1(n) = **x**(n-1)

y(n) = **w1**(n)

w1(n+1) = **x**(n)

y(n+1) = **w1**(n+1)

w1(n+2) = **x**(n+1)

y(n+2) = **w1**(n+2)

w1(n) = **w1**(n-1)

w1(n+1) = **w1**(n)

w1(n+2) = **w1**(n+1)

these conditions are met
by using a <u>single</u> register **W**

W	= X	*WR* **W**
Y	= **W**	*RD* **W**

time centered operations

y(n-1) = **w1**(n-1)

w1(n) = **x**(n-1)

y(n) = **w1**(n)

w1(n+1) = **x**(n)

y(n+1) = **w1**(n+1)

w1(n+2) = **x**(n+1)

y(n+2) = **w1**(n+2)

w1(n) = **w1**(n-1)

w1(n+1) = **w1**(n)

w1(n+2) = **w1**(n+1)

time n-1

time n

time n+1

time n+2

these conditions are met
by using a <u>single</u> register **W**

Y	= **W**	*RD* **W**
W	= X	*WR* **W**

Data centered operations

data centered operations　　　　　*in the Register Transfer Level*

y(n-1) = **w1**(n-1)	*output y(n-1) from w1(n-1)*	

w1(n) = **w1**(n-1)

w1(n) = **x**(n-1)　　*input x(n-1) to w1(n)*

Y = W1　　*output y(n-1) from W1*

W1 = X　　*input x(n-1) to W1*

⎫ *time n-1*

y(n) = **w1**(n)　　*output y(n) from w1(n)*

w1(n+1) = **w1**(n)

w1(n+1) = **x**(n)　　*input x(n) to w1(n+1)*

Y = W1　　*output y(n) from W1*

W1 = X　　*input x(n) to W1*

⎫ *time n*

y(n+1) = **w1**(n+1)　　*output y(n+1) from w1(n+1)*

w1(n+2) = **w1**(n+1)

w1(n+2) = **x**(n+1)　　*input x(n+1) to w1(n+2)*

Y = W1　　*output y(n+1) from W1*

W1 = X　　*input x(n+1) to W1*

⎫ *time n+1*

y(n+2) = **w1**(n+2)　　*output y(n+2) from w1(n+2)*

Y = W1　　*output y(n+2) from W1*

⎫ *time n+2*

(1)　**w1**(n) = **x**(n-1)　　***WR w1(n)***

(2)　**y**(n) = **w1**(n)　　***RD w1(n)***

W1 = X　　***WR W1***

Y = W1　　***RD W1***

Time centered operations

time centered operations *in the Register Transfer Level*

y(n-1) = **w1**(n-1) *output from w1(n-1)*	**Y** = **W1** *output y(n-1) from W1*
w1(n) = **w1**(n-1)	
w1(n) = **x**(n-1) *input to w1(n)*	**W1** = **X** *input x(n-1) to W1*

time n-1

y(n) = **w1**(n) *output from w1(n)*	**Y** = **W1** *output y(n) from W1*
w1(n+1) = **w1**(n)	
w1(n+1) = **x**(n) *input to w1(n+1)*	**W1** = **X** *input x(n) to W1*

time n

y(n+1) = **w1**(n+1) *output from w1(n+1)*	**Y** = **W1** *output y(n+1) from W1*
w1(n+2) = **w1**(n+1)	
w1(n+2) = **x**(n+1) *input to w1(n+2)*	**W1** = **X** *input x(n+1) to W1*

time n+1

y(n+2) = **w1**(n+2) *output from w1(n+2)*	**Y** = **W1** *output y(n+2) from W1*

time n+2

(2)   **y**(n) = **w1**(n)   *RD w1(n)*

(1)   **w1**(n) = **x**(n)   *WR w1(n)*

Y   = **W1**   *RD W1*

W1   = **X**   *WR W1*

Simultaneous **RD** and **WR** actions

w1(n)

x(n) → z^{-1} → **y**(n)

(2) **WR**   (1) **RD**
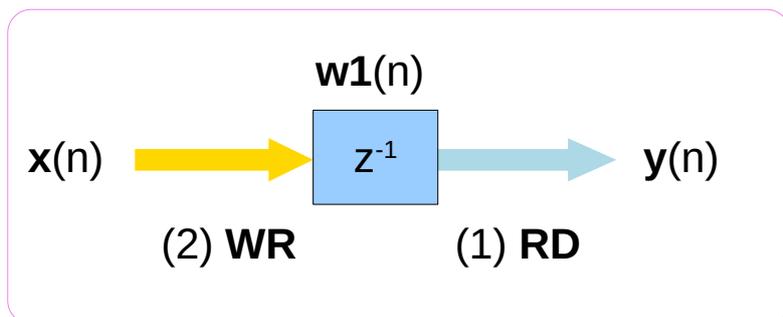
y(n) = **w1**(n)	**y**(n+1) = **w1**(n+1)
w1(n+1) = **x**(n)	**w1**(n+2) = **x**(n+1)

(1) **RD**
(2) **WR**

at time n   at time n+1

   y(n) = **w1**(n)   **y**(n+1) = **w1**(n+1)

   w1(n+1) = **x**(n)   **w1**(n+2) = **x**(n+1)

time centered operations

time n-1   *time n*   *time n+1*

x(n-1)   **x(n)**   x(n+1)   *Input*

(1)   **WR**   (3)

w1(n-1)   **w1(n)**   **w1(n+1)**

(2)   **RD**   (4)

y(n-1)   **y(n)**   y(n+1)

read internal state

update internal state

DSP C model
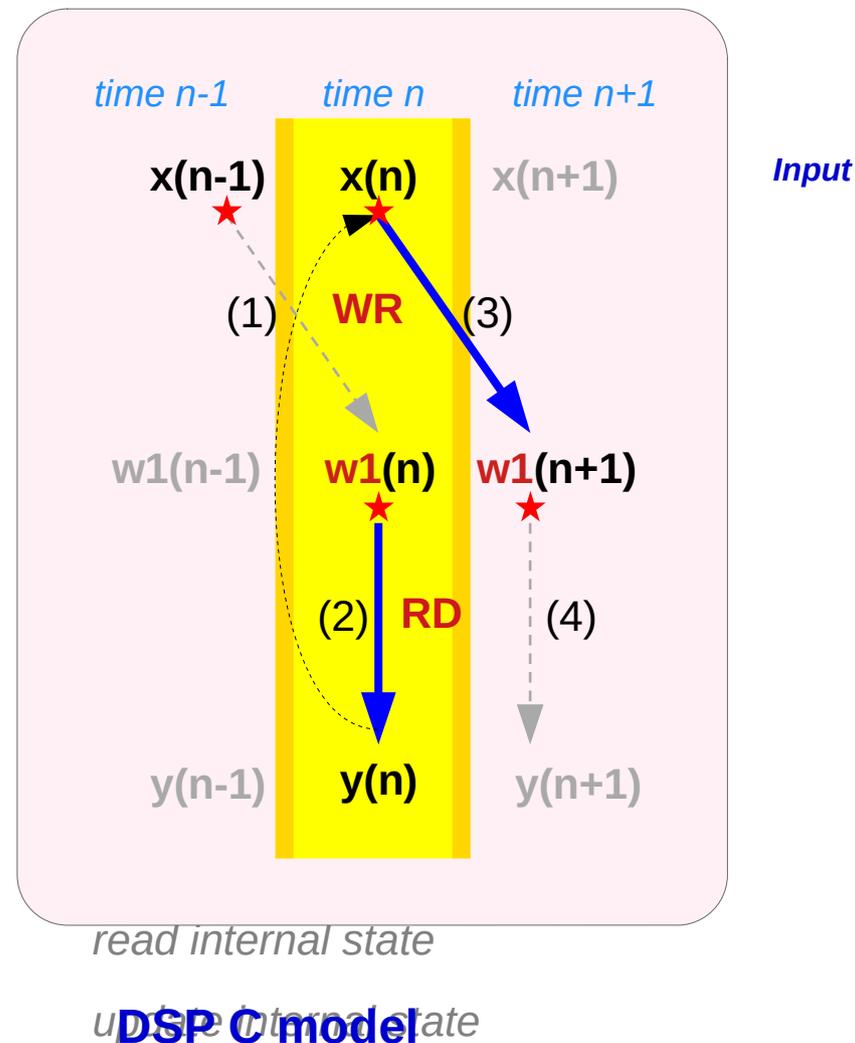
at time n,
  1) the <u>content</u> of the register **w1**(n)

RD access of **w1**(n) = **x**(n-1)

Simultaneous **RD** and **WR** actions

w1(n)

x(n) $\xrightarrow{\hspace{1cm}}$ z^{-1} $\xrightarrow{\hspace{1cm}}$ **y**(n)

(2) **WR**   (1) **RD**
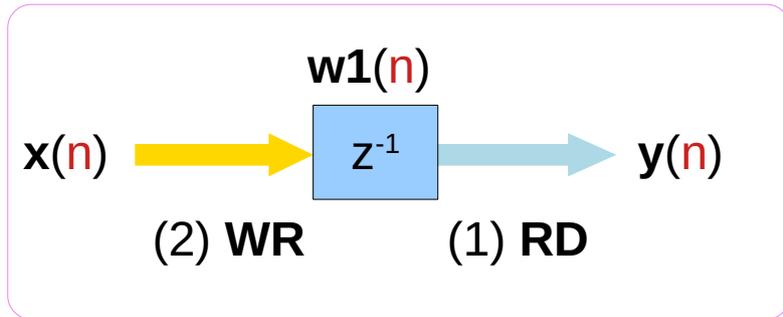
current content	**w1**(n)	= **x**(n-1)
next content	**w1**(n+1)	= **x**(n)

at time n   at time n+1

y(n) = **w1**(n)
w1(n+1) = **x**(n)

$\longrightarrow$

y(n+1) = **w1**(n+1)
w1(n+2) = **x**(n+1)

read internal state

update internal state

at time n,

1) the <u>content</u> of the register **w1**(n)   **RD** access of **w1**(n) = **x**(n-1)

   becomes the output **y**(n)

2) the input **x**(n) is saved and   **WR** access of **w1**(n+1) = **x**(n)

   becomes the <u>new</u> <u>content</u> **w1**(n+1)

Current content **w1(n)** and current input **x(n)**

$$w1(n)$$

$$x(n) \longrightarrow \boxed{z^{-1}} \longrightarrow y(n)$$

(2) **WR** (1) **RD**

current content	**w1**(n)	= **x**(n-1)
next content	**w1**(n+1)	= **x**(n)
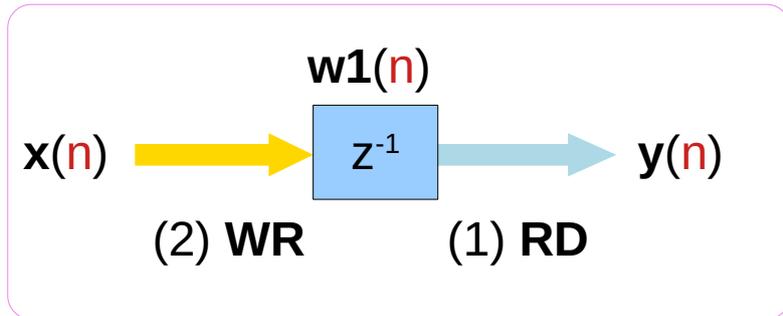
current content

$$y(n) \longleftarrow w1(n) = x(n-1)$$
$$w1(n+1) \longleftarrow x(n)$$

current input

a register holding the previous input sample **x**(n-1)

(1) the current content **x**(n-1) is clocked out to the output
(2) the current input **x**(n) gets stored in the register

It will be held for <u>one</u> sampling instant and
become the output at the <u>next</u> time n+1
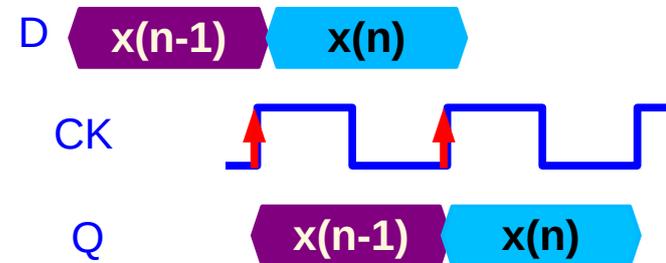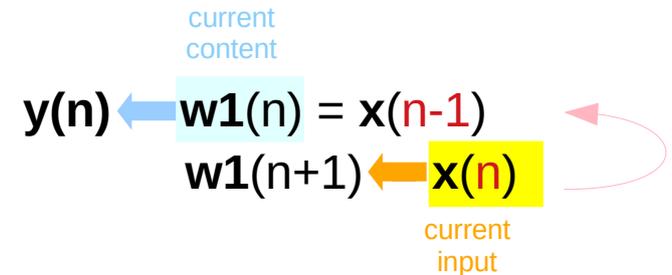
D   x(n-1)   **x(n)**

CK

Q   x(n-1)   **x(n)**

Current content **w1(n)** and current input **x(n)**

$x(n)$

$\xrightarrow{\text{(2) WR}}$ $\boxed{z^{-1}}$ **w1**(n) $\xrightarrow{\text{(1) RD}}$ $y(n)$

current content	**w1**(n)	= **x**(n-1)
next content	**w1**(n+1)	= **x**(n)

current content

$y(n) \leftarrow$ **w1**(n) = **x**(n-1)

$\qquad$ **w1**(n+1) $\leftarrow$ **x(n)**

current input

$\bullet$ **x**(n)

D $\langle$ **x(n-1)** $\rangle\langle$ **x(n)** $\rangle$

CK
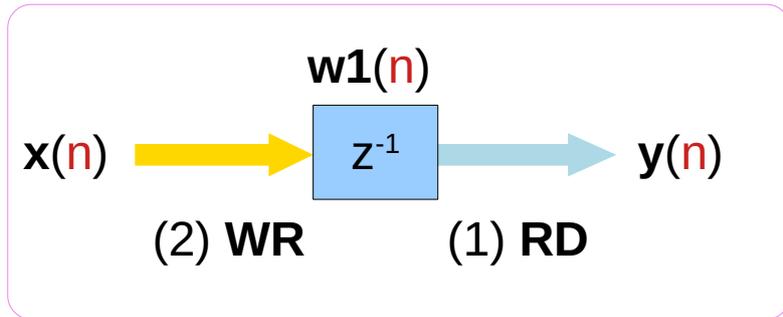
Q $\langle$ **x(n-1)** $\rangle\langle$ **x(n)** $\rangle$
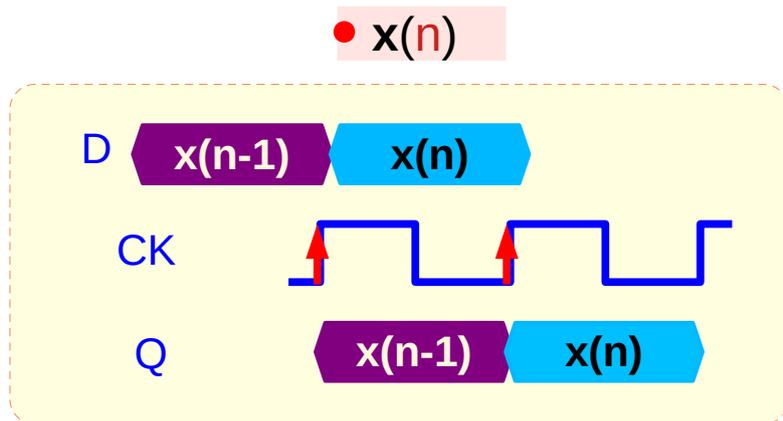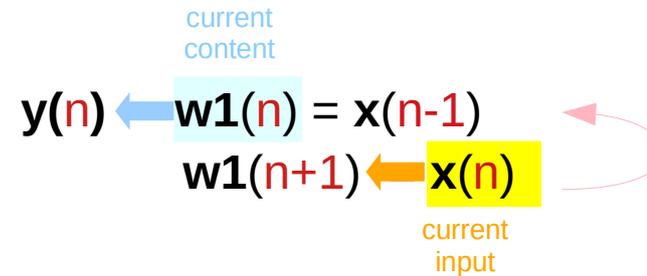
$\bullet$ **w1**(n)

$\bullet$ **y**(n)

simulate a clocked hardware
ignoring delay constraints in hardware

zero-delay simulation

Delay element modeling

$w1(n)$

$x(n)$ → z^{-1} → $y(n)$

(2) **WR**   (1) **RD**
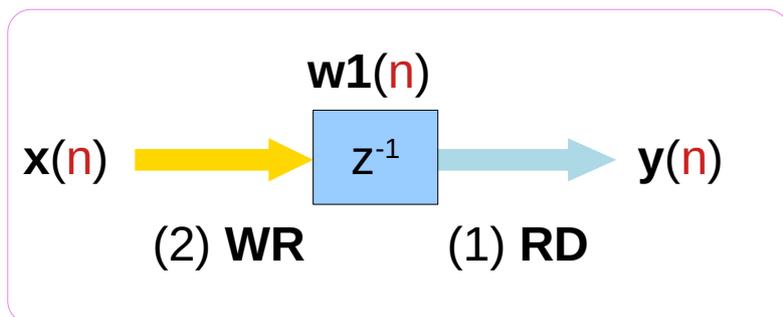
$w1(n)$ ⟶ $y(n)$	$y(n) = w1(n)$ (1) **RD** old w1
$x(n)$ ⟶ $w1(n+1)$	$w1(n+1) = x(n)$ (2) **WR** new w1

The content of the delay register at time n
as the internal state of the filter by

internal state at time n	$w1(n) = x(n-1)$	**WR** at time n-1	
internal state at time n+1	$w1(n+1) = x(n)$	**WR** at time n	
output at time n	$y(n) = w1(n)$	**RD** at time n	

RD before **WR**   **WAR** (Write after Read) Access
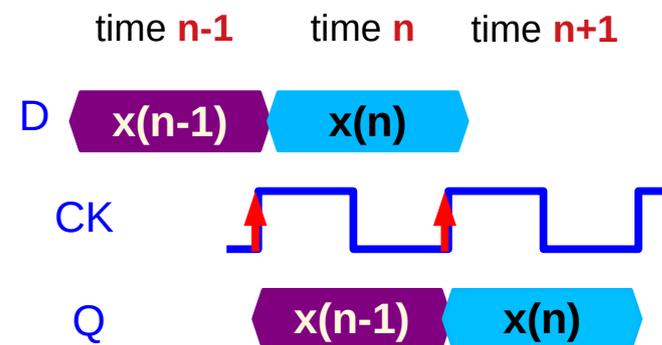
time **n-1**   time **n**   time **n+1**

D $x(n-1)$ $x(n)$

CK

Q $x(n-1)$ $x(n)$

simulate a clocked hardware
ignoring delay constraints in hardware

zero-delay simulation

WAR (Write after Read)

y(n) = **w1**(n)	(1) **RD**	old w1
w1(n+1) = x(n)	(2) **WR**	new w1
y(n+1) = **w1**(n+1)	(1) **RD**	old w1
w1(n+2) = **x**(n+1)	(2) **WR**	new w1

w1(n+1) = x(n)	(2) **WR**	new w1
y(n) = **w1**(n)	(1) **RD**	old w1
w1(n+2) = **x**(n+1)	(2) **WR**	new w1
y(n+1) = **w1**(n+1)	(1) **RD**	old w1

WAR (Write after Read) Violation
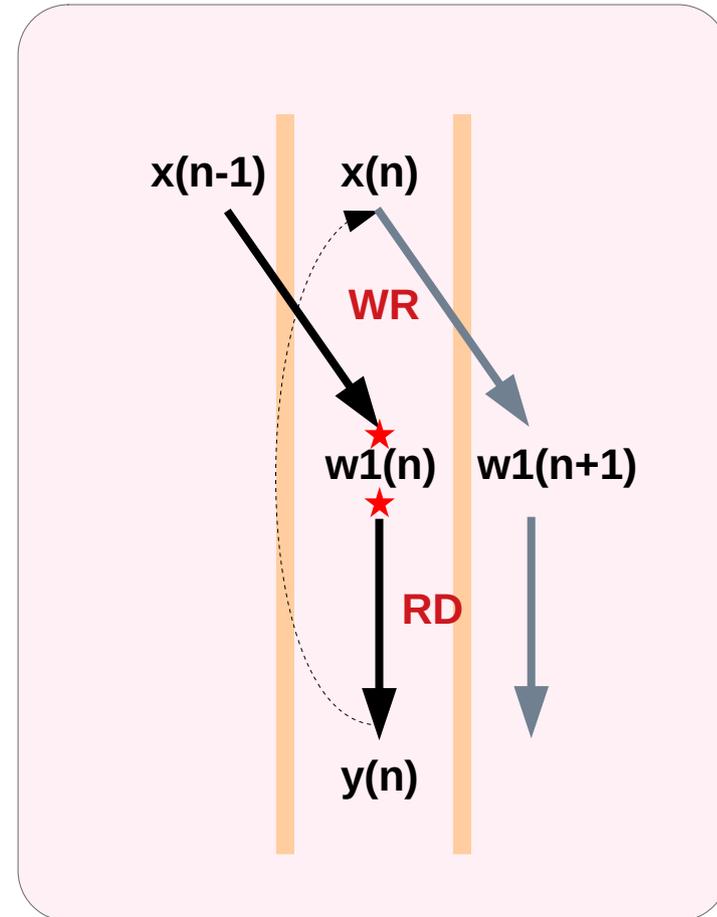
DSP C model

Single Delay

w1(n)

x(n)   **y**(n)

z⁻¹

(2)   (1)
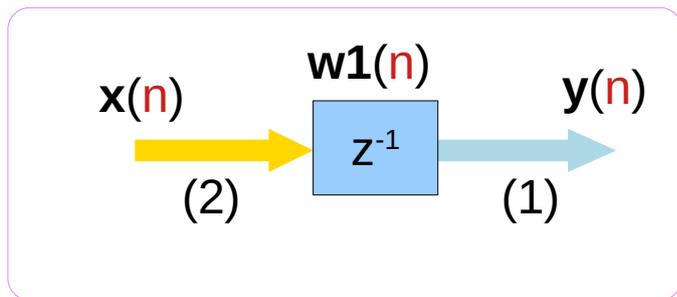
$$y(n) = w1(n) \quad (1) \textbf{ RD}$$
$$w1(n+1) = x(n) \quad (2) \textbf{ WR}$$
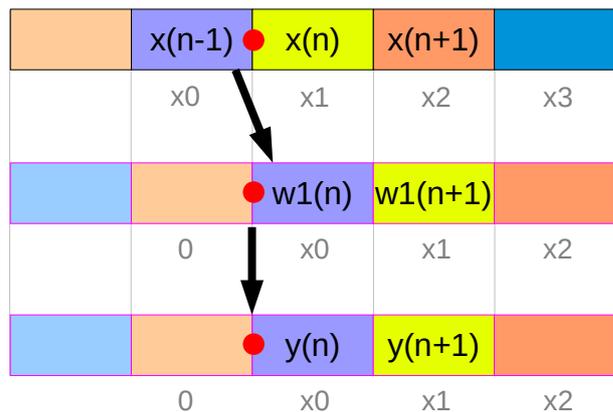
	x(n-1) •	x(n)	x(n+1)	
	x0	x1	x2	x3
		• w1(n)	w1(n+1)	
	0	x0	x1	x2
		• y(n)	y(n+1)	
	0	x0	x1	x2

n	x(n)	w1(n)	y(n)
0	x0	0	0
1	x1	x0	x0
2	x2	x1	x1
3	x3	x2	x2
4	x4	x3	x3

Double Delay
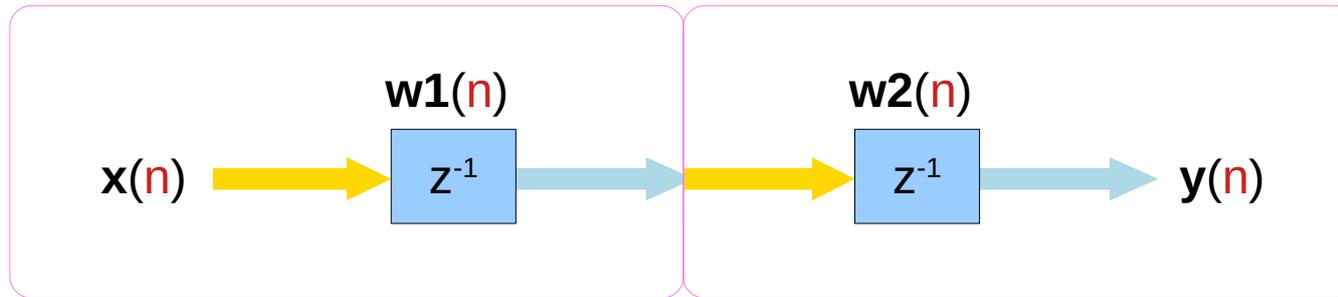
$$\mathbf{y}(n) = \mathbf{w2}(n)$$
$$\mathbf{w2}(n+1) = \mathbf{w1}(n)$$
$$\mathbf{w1}(n+1) = \mathbf{x}(n)$$
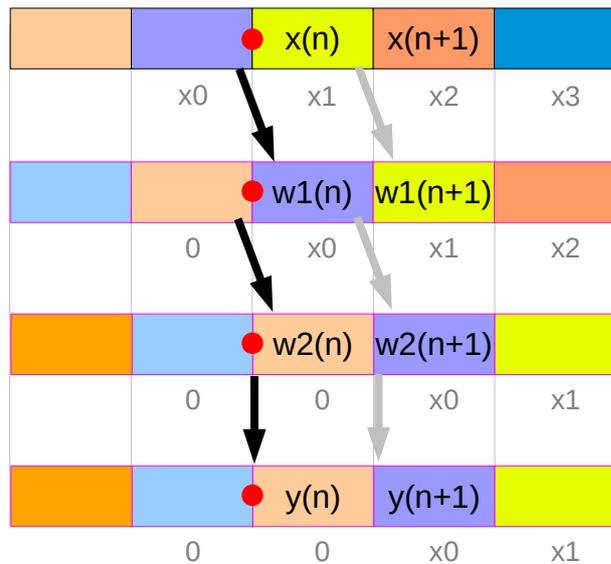
n	x(n)	w1(n)	w2(n)	y(n)
0	x0	0	0	0
1	x1	x0	0	0
2	x2	x1	x0	x0
3	x3	x2	x1	x1
4	x4	x3	x2	x2
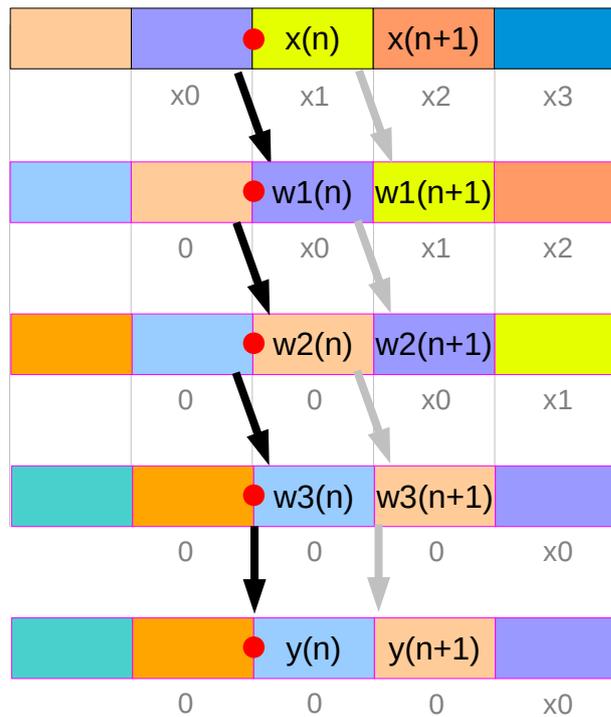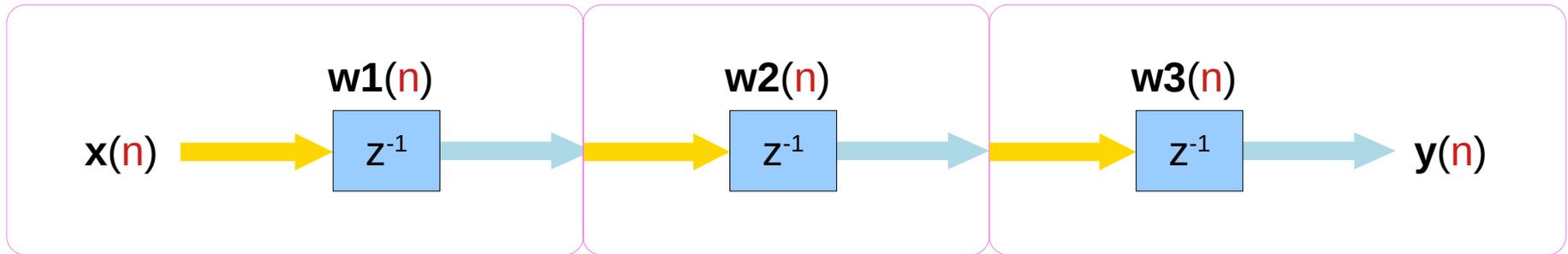
n	x(n)	w1(n)	w2(n)	w3(n)	y(n)
0	x0	0	0	0	0
1	x1	x0	0	0	0
2	x2	x1	x0	0	0
3	x3	x2	x1	x0	x0
4	x4	x3	x2	x1	x1

$$\mathbf{y}(n) \qquad = \mathbf{w3}(n)$$
$$\mathbf{w3}(n+1) = \mathbf{w2}(n)$$
$$\mathbf{w2}(n+1) = \mathbf{w1}(n)$$
$$\mathbf{w1}(n+1) = \mathbf{x}(n)$$

Single Delay – IO Equations
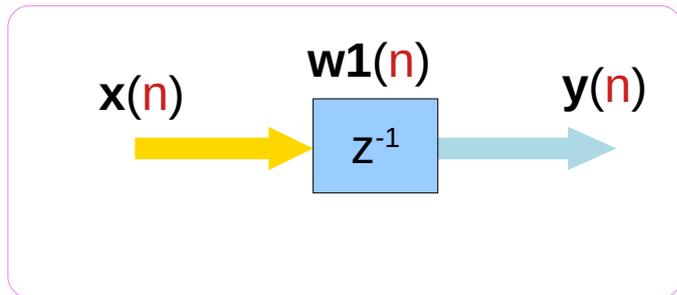
single delay

y(n) = **w1**(n)	output
w1(n+1) = **x**(n)	input

double delay

y(n) = **w2**(n)	output
w2(n+1) = **w1**(n)	
w1(n+1) = **x**(n)	input

triple delay

y(n) = **w3**(n)	output
w3(n+1) = **w2**(n)	
w2(n+1) = **w1**(n)	
w1(n+1) = **x**(n)	input
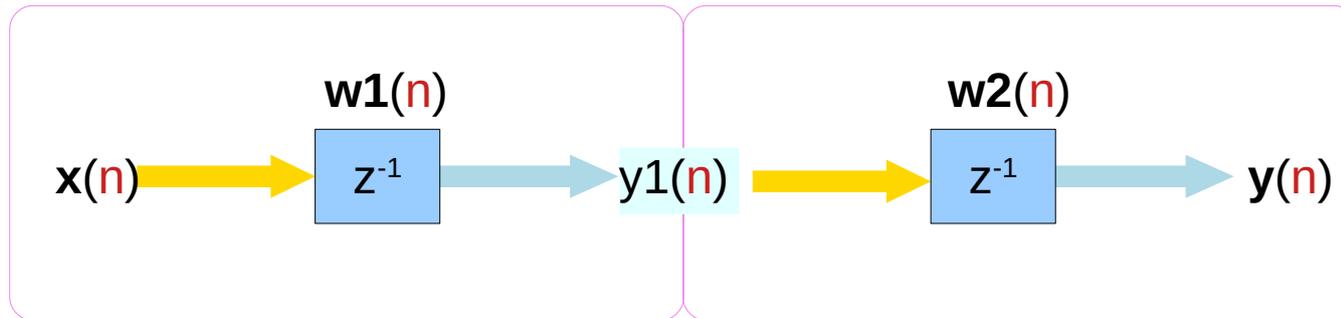
$$y1(n) = w1(n)$$
$$w1(n+1) = x(n)$$

$$y(n) = w2(n)$$
$$w2(n+1) = y1(n)$$

$$y1(n) = w1(n)$$
$$w2(n+1) = y1(n)$$
$$\overline{w2(n+1) = w1(n)}$$

$$y(n) \qquad = w2(n)$$ output
$$w2(n+1) = w1(n)$$
$$w1(n+1) = x(n)$$ input

Triple Delay – IO Equations
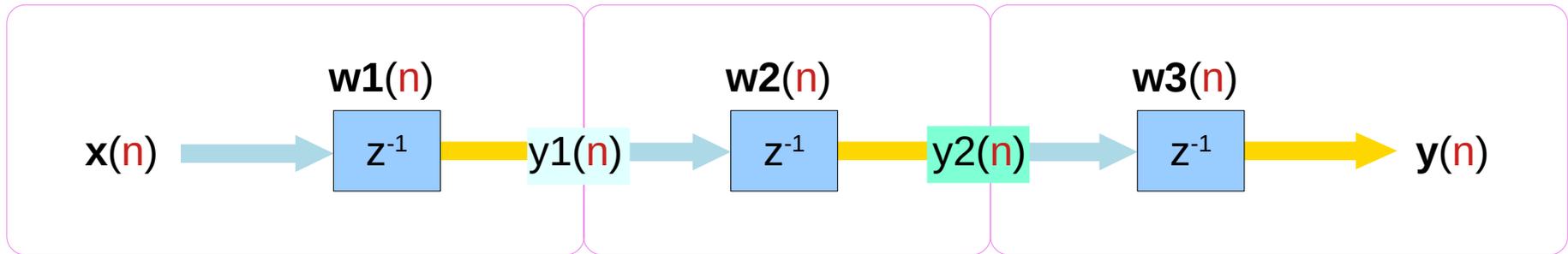
$$y1(n) = \mathbf{w1}(n)$$
$$\mathbf{w1}(n+1) = \mathbf{x}(n)$$

$$y2(n) = \mathbf{w2}(n)$$
$$\mathbf{w2}(n+1) = y1(n)$$

$$\mathbf{y}(n) = \mathbf{w3}(n)$$
$$\mathbf{w3}(n+1) = y2(n)$$

$$y1(n) = \mathbf{w1}(n)$$
$$\underline{\mathbf{w2}(n+1) = y1(n)}$$
$$\mathbf{w2}(n+1) = \mathbf{w1}(n)$$

$$y2(n) = \mathbf{w2}(n)$$
$$\underline{\mathbf{w3}(n+1) = y2(n)}$$
$$\mathbf{w3}(n+1) = \mathbf{w2}(n)$$

$$\mathbf{y}(n) \quad\quad = \mathbf{w3}(n) \quad \text{output}$$
$$\mathbf{w3}(n+1) = \mathbf{w2}(n)$$
$$\mathbf{w2}(n+1) = \mathbf{w1}(n)$$
$$\mathbf{w1}(n+1) = \mathbf{x}(n) \quad \text{input}$$

Delay C Model

Timing Chart
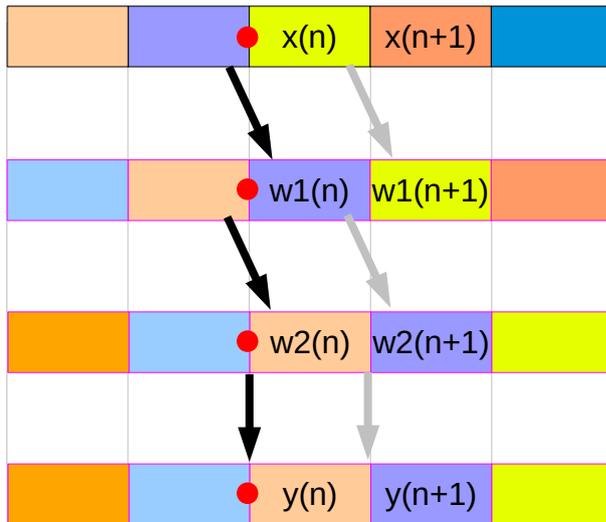
y(n)   = w2(n)
w2(n+1) = w1(n)
w1(n+1) = x(n)

Register Transfer

x

register

w1

register

w2

y

y   = w2
w2 = w1
w1 = x
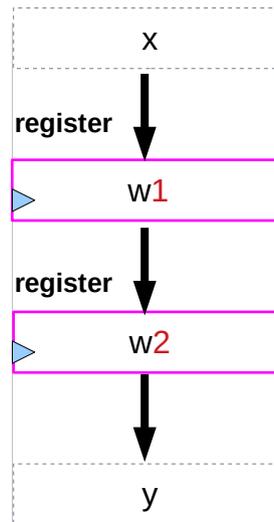
DSP C Model for simulation
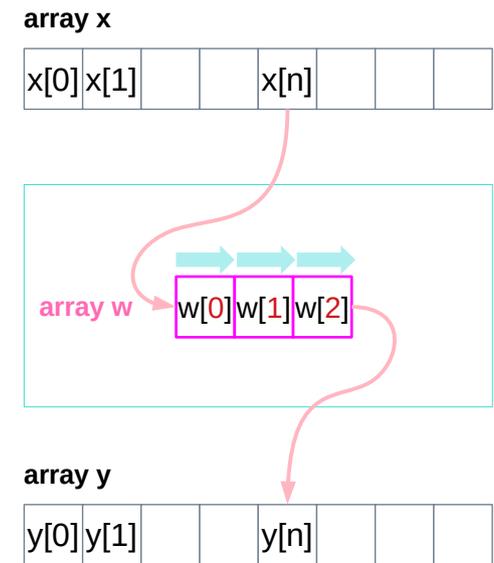
array x

x[0]	x[1]		x[n]			

array w   w[0] w[1] w[2]

array y

y[0]	y[1]		y[n]			

y[n] = w[2]
w[0] = x[n]
w[2] = w[1]
w[1] = w[0]

$$y(n) = w2(n)$$
$$w0(n) = x(n)$$
$$w2(n+1) = w1(n)$$
$$w1(n+1) = w2(n) \qquad D = 2, 1$$

```
y[n] = w[2]        // get the output
w[0] = x[n]        // put the input
w[2] = w[1]        // shift
w[1] = w[0]        // shift
```
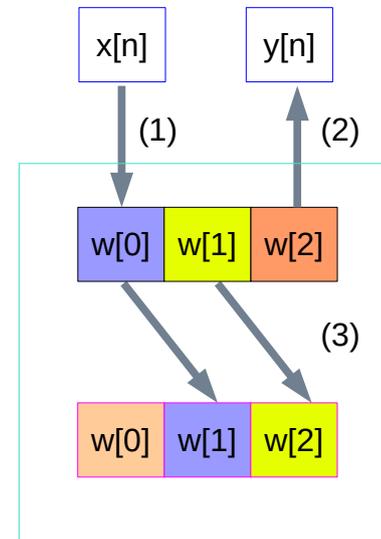
delay.c

/* delay.c - delay by D time samples */
/* w[0] = input, w[D] = output */

```c
void delay(int D, double *w)
{
    int i;

    for (i=D; i>=1; i--)
        w[i] = w[i-1];

    // reverse-order updating
}
```

Shift to the right by one

0   1   D

| w[0] | w[1] | | w[D} |

| w[0] | w[1] | | w[D] |

input

array w | w[0] | w[1] | w[2] |
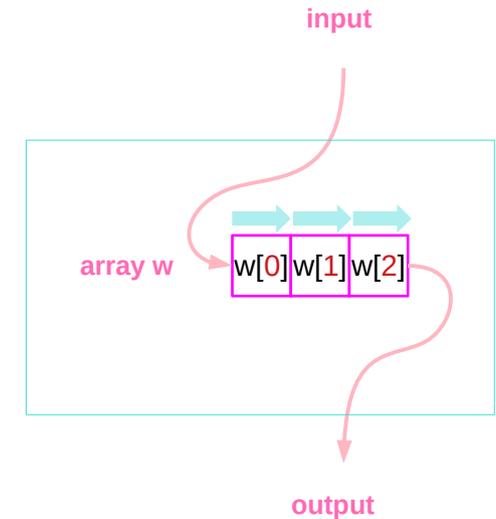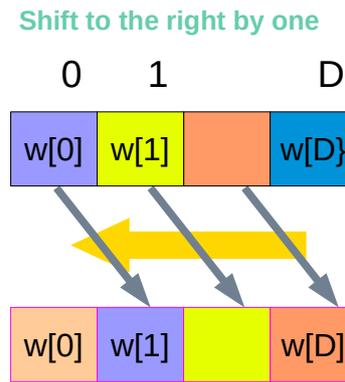
output

order of execution

$$w[D] = w[D-1]$$

… …

$$w[2] = w[1]$$

$$w[1] = w[0]$$

Using the delay function

```
double *w;
w = (double *) calloc(D+1, sizeof(double));    // (D+1)-dimensional


for (n = 0; n < Ntot; n++) {
    y[n] = w[D];            // (1) write output
    w[0] = x[n];            // (2) read input
    delay(D, w);            // (3) update delay line
}
```
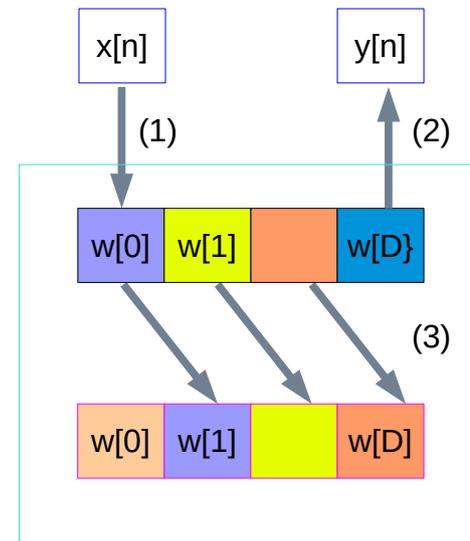
$y(n) = w_1(n)$

$w_1(n+1) = x(n)$


$y(n) = w_2(n)$

$w_2(n+1) = w_1(n)$

$w_1(n+1) = x(n)$

$y(n) = w_3(n)$

$w_3(n+1) = w_2(n)$

$w_2(n+1) = w_1(n)$

$w_1(n+1) = x(n)$

$y(n) = w_D(n)$

$w_0(n) = x(n)$

$w_i(n+1) = w_{i-1}(n),$

$i = D, D-1, \ldots, 2, 1$

---

time index : n

memory location : W_i
memory index : i

$w_i(n+1) = w_{i-1}(n)$

the current value at w_{i-1}
will become
the next value at w_i

input   output

x[n]   y[n]

(1)   (2)

At time **n** | w[0] | w[1] | | w[D} |

(3)

At time **n+1** | w[0] | w[1] | | w[D] |

Shift to the right by one
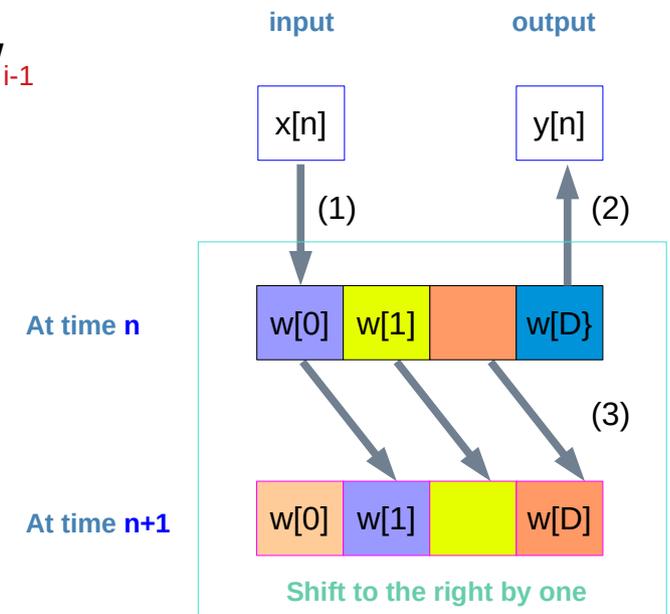
Holding a delayed input sequence

$w_0(n) = x(n)$

$w_1(n) = x(n-1) = w_0(n-1)$

$w_2(n) = x(n-2) = w_1(n-1)$

$w_3(n) = x(n-3) = w_2(n-1)$

input

array w   w[0] w[1] w[2]

output

Single Delay (1)

$$w_1(n)$$

$$x(n) \longrightarrow \boxed{z^{-1}} \longrightarrow y(n)$$

$$[x_0, x_1, x_2, x_3, \ldots] \rightarrow [0, x_0, x_1, x_2, x_3, \ldots]$$

for each input sample x *do*:

$$y := w_1$$

$$w_1 := x$$
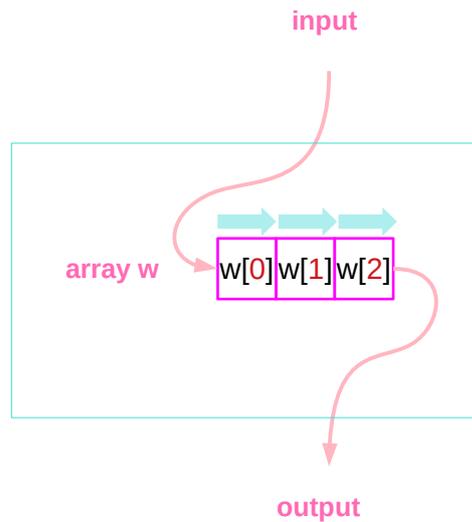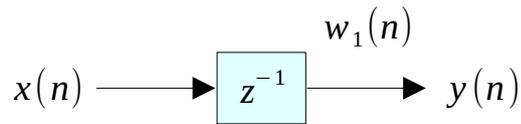
$w_1(n) = x(n-1)$ (internal state at time n)

$w_1(n+1) = x(n)$ (internal state at time n+1)

$y(n) = w_1(n)$

$w_1(n+1) = x(n)$

n	$x(n)$	$w_1(n)$	$y(n)$
0	x_0	0	0
1	x_1	x_0	x_0
2	x_2	x_1	x_1
3	x_3	x_2	x_2
4	x_4	x_3	x_3

$y(n+1) = w_1(n+1)$

$w_1(n+2) = x(n+1)$

$w_1(0) = 0$

Double Delay (1)

$$w_1(n) \qquad w_2(n)$$

$$x(n) \longrightarrow \boxed{z^{-1}} \longrightarrow \boxed{z^{-1}} \longrightarrow y(n)$$

$$[x_0, x_1, x_2, x_3, \ldots] \rightarrow [0, x_0, x_1, x_2, x_3, \ldots]$$

for each input sample x do:

$$y := w_2$$
$$w_2 := w_1$$
$$w_1 := x$$
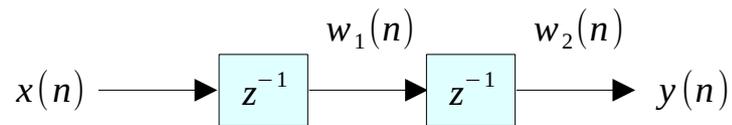
$$w_2(n) = w_1(n-1) \qquad = x((n-1)-1) = x(n-2)$$
$$w_1(n) = x(n-1)$$

$$w_2(n+1) = w_1(n)$$
$$w_1(n+1) = x(n)$$

n	$x(n)$	$w_1(n)$	$w_2(n)$	$y(n)$
0	x_0	0	0	0
1	x_1	x_0	0	0
2	x_2	x_1	x_0	x_0
3	x_3	x_2	x_1	x_1
4	x_4	x_3	x_2	x_2

$$y(n) = w_2(n)$$
$$w_2(n+1) = w_1(n)$$
$$w_1(n+1) = x(n)$$
$$w_1(0) = 0$$

Triple Delay (1)

$$x(n) \longrightarrow \boxed{z^{-1}} \xrightarrow{w_1(n)} \boxed{z^{-1}} \xrightarrow{w_2(n)} \boxed{z^{-1}} \xrightarrow{w_3(n)} y(n)$$

$$[x_0, x_1, x_2, x_3, \ldots] \rightarrow [0, x_0, x_1, x_2, x_3, \ldots]$$

for each input sample x do:

$$y := w_3$$
$$w_3 := w_2$$
$$w_2 := w_1$$
$$w_1 := x$$
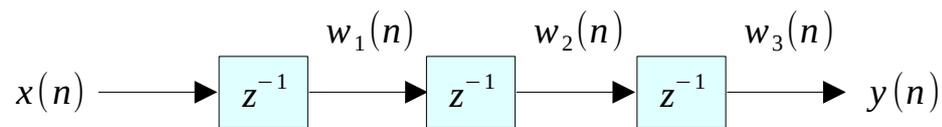
$$w_3(n) = w_2(n-1) \qquad = w_1(n-2) = x(n-3)$$
$$w_2(n) = w_1(n-1)$$
$$w_1(n) = x(n-1)$$

$$w_3(n+1) = w_2(n)$$
$$w_2(n+1) = w_1(n)$$
$$w_1(n+1) = x(n)$$

n	$x(n)$	$w_1(n)$	$w_2(n)$	$y(n)$
0	x_0	0	0	0
1	x_1	x_0	0	0
2	x_2	x_1	x_0	x_0
3	x_3	x_2	x_1	x_1
4	x_4	x_3	x_2	x_2

$$y(n) = w_3(n)$$
$$w_3(n+1) = w_2(n)$$
$$w_2(n+1) = w_1(n)$$
$$w_1(n+1) = x(n)$$

D Unit Delay (1)
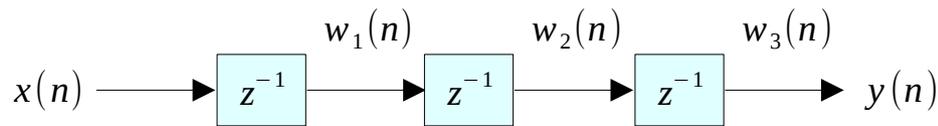
$$[x_0, x_1, x_2, x_3, \ldots] \rightarrow [0, x_0, x_1, x_2, x_3, \ldots]$$

$x(n)$ → z^{-1} → $w_1(n)$ → z^{-1} → $w_2(n)$ → z^{-1} → $w_3(n)$ → $y(n)$

$$w_i(n) = w_{i-1}(n-1) \qquad for\ i = 1, 2, \ldots, D$$

for each input sample x do:

$$y := w_D$$

$$w_0 := x$$

$$w_0 := x$$

for $i = D, D-1, \ldots, 1\, do$:

$$w_i := w_{i-1}$$

$$w_3(n+1) = w_2(n)$$
$$w_2(n+1) = w_1(n)$$
$$w_1(n+1) = x(n)$$

n	$x(n)$	$w_1(n)$	$w_2(n)$	$y(n)$
0	x_0	0	0	0
1	x_1	x_0	0	0
2	x_2	x_1	x_0	x_0
3	x_3	x_2	x_1	x_1
4	x_4	x_3	x_2	x_2

for each input sample w_0 *do*:

for $i = D, D-1, \ldots, 1\, do$:

$$w_i := w_{i-1}$$

$$y(n) = w_D(n)$$
$$w_0(n) = x(n)$$
$$w_i(n+1) = w_{i-1}(n)$$
$$i = D, D-1, \ldots, 2, 1$$

D Unit Delay (1)

```
/* delay.c - delay by D time samples */
void delay(int D, double *w)          w[0] = input, w[D] = output
{
        int i;

        for (i=D; i>=1; i--)                      reverse-order updating
                w[i] = w[i-1];

}
```

dot

```
/* dot.c - dot product of two length-(M+1) vectors */
double dot(int M, double *h, double *w)        Usage: y = dot(M, h, w);
{                                                      h = filter vector, w = state vector
    int i;                                             M = filter order
    double y;

    for (y=0, i=0; i<=M; i++)                     compute dot product
    y += h[i] * w[i];

    return y;
}
```

$$y \;=\; h_0 w_0 + h_1 w_1 + \ldots + h_M w_M \;=\; [h_0, h_1, \cdots, h_M]\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix} \;=\; \boldsymbol{h^T w}$$

Direct Form

Considering the widely used
Edge triggered
D-type Flip Flops

$$H(z) = \frac{N(Z)}{D(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

$$y_n = -a_1 y_{n-1} - a_2 y_{n-2} + b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2}$$

References

[1]   S. J. Ofranidis , Introduction to Signal Processing