Tapped Delay

Young Won Lim
11/13/24

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

Based on

Introduction to Signal Processing

S. J. Ofranidis

The necessities in DSP C Programming

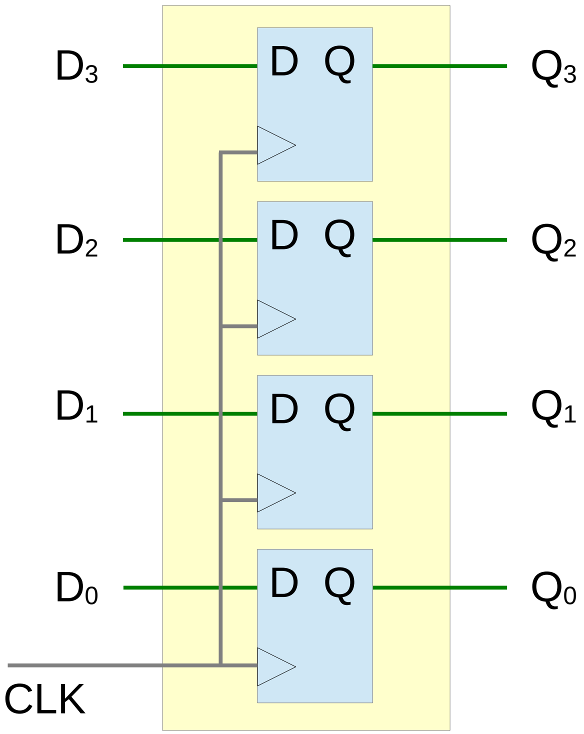
FIR Filter (A.pdf) 20191114

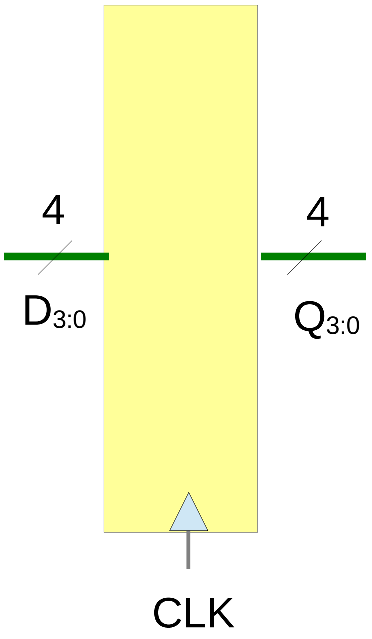
D Flip Flop

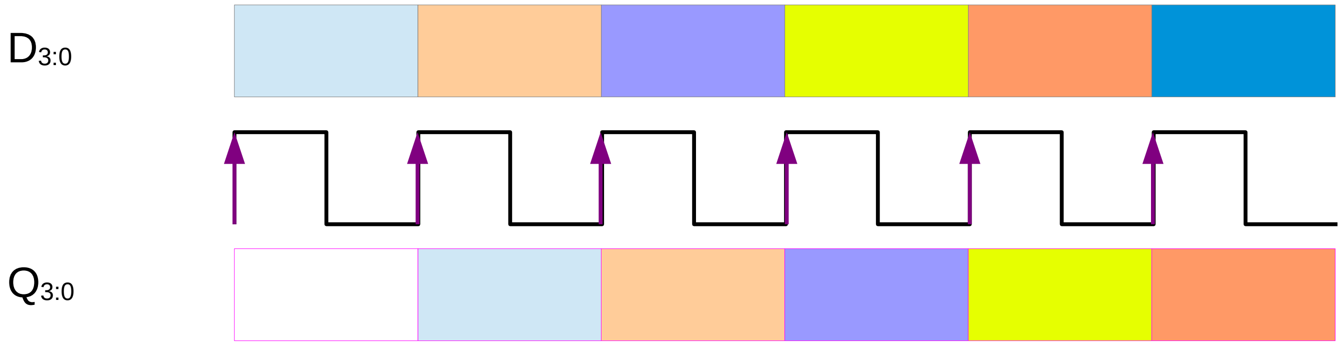
Considering the widely used
Edge triggered
D-type Flip Flops

Register

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design







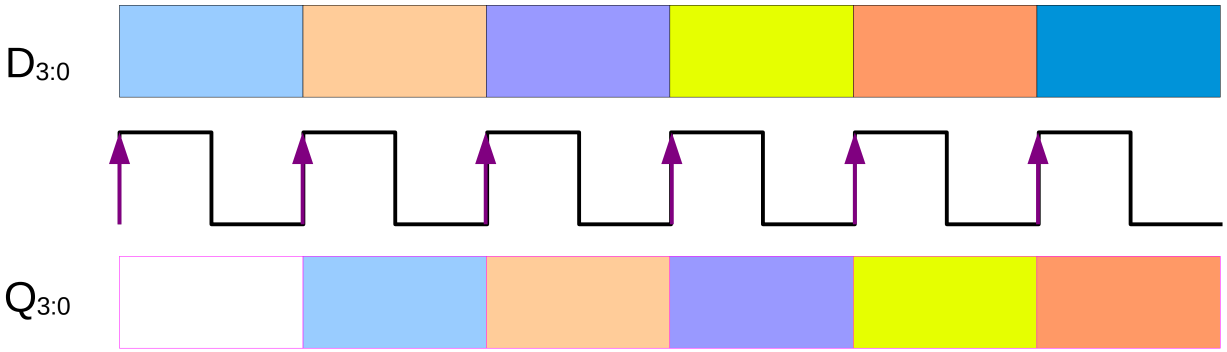
without delay

without delay

Types of Timing Diagrams

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

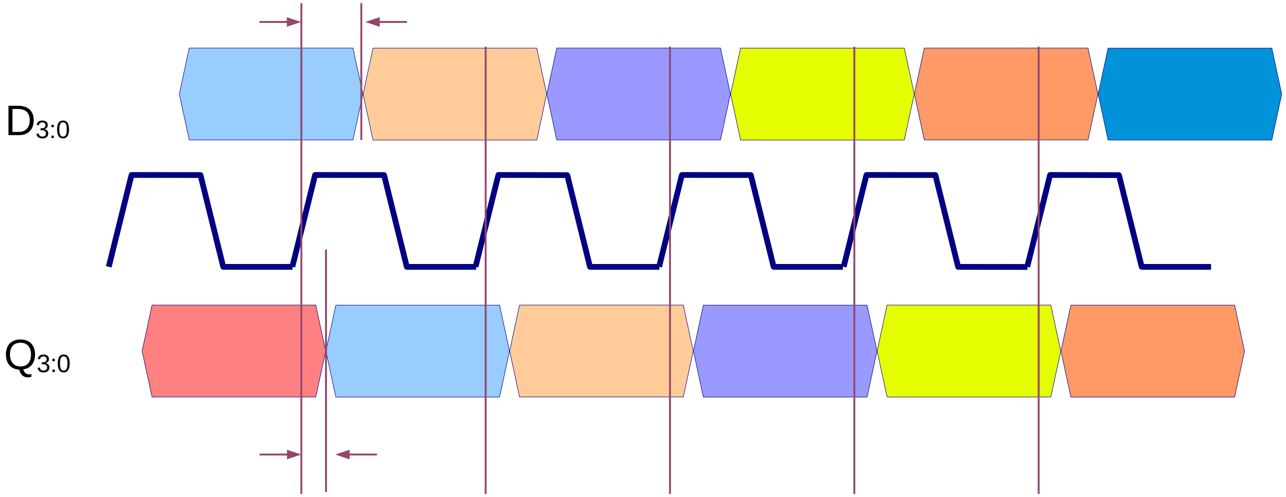
a timing diagram without delays



no delay (ideal case)

no delay (ideal case)

a timing diagram with delays

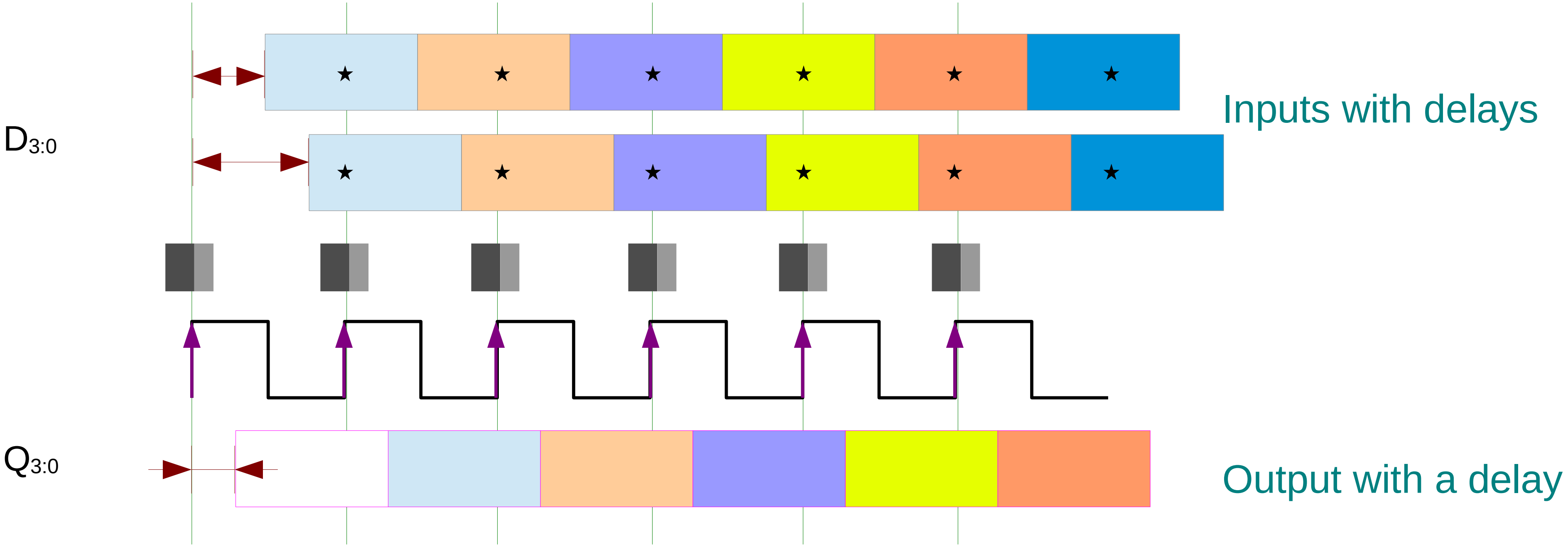


input delay

output delay Clk->Q

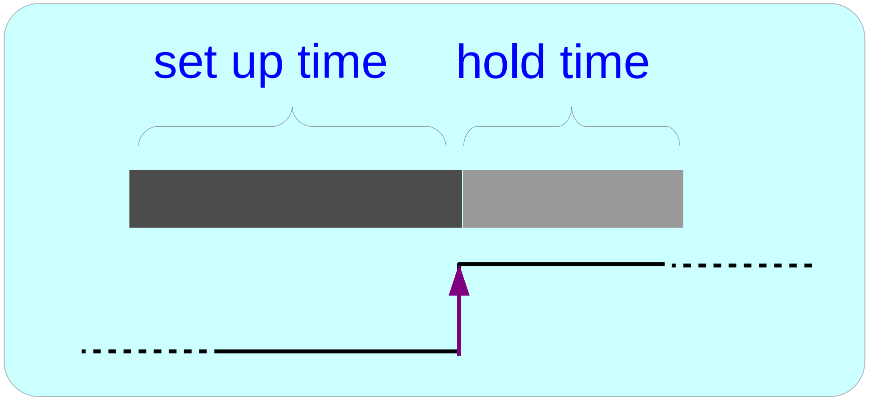
Setup & Hold Time (1)

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design



Inputs with delays

Output with a delay

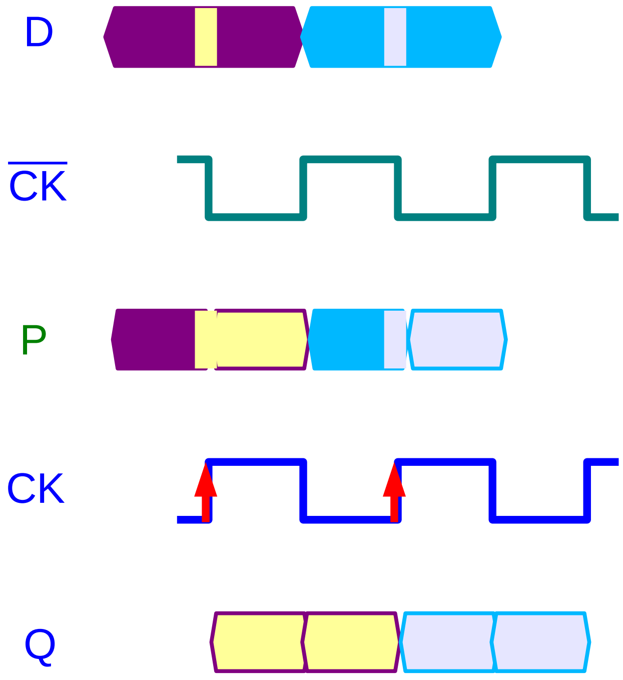


Master-Slave D FlipFlop – Rising Edge

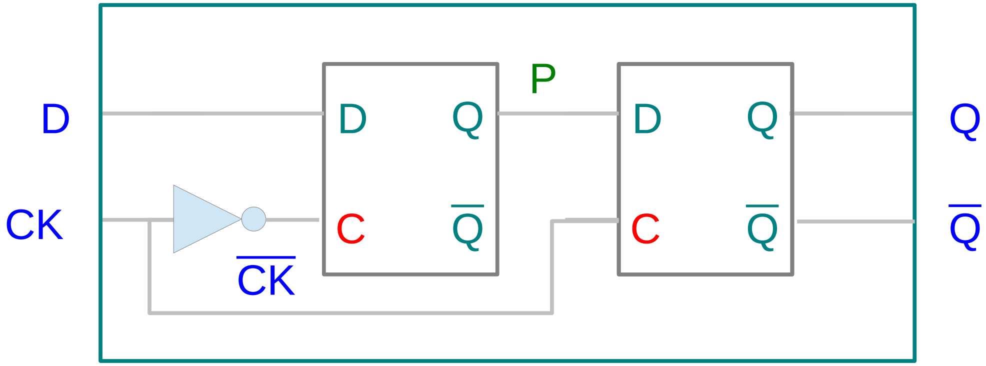
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

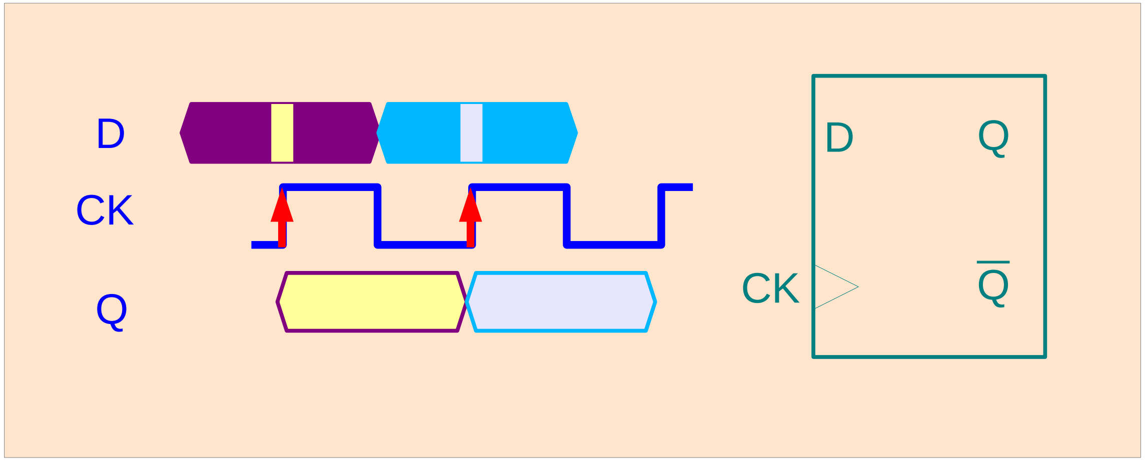
Master D Latch

most of D data in a time slot (dark shaded area) is not used but only the data around the rising edge of the clock CK



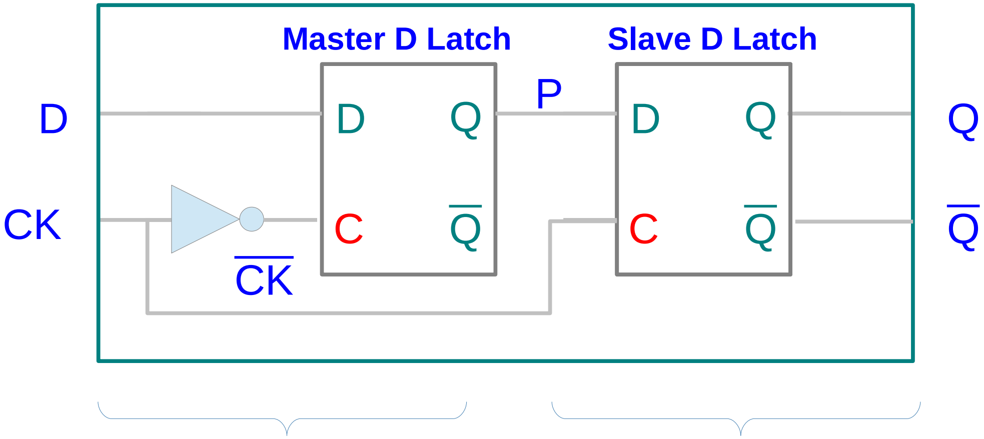
Slave D Latch





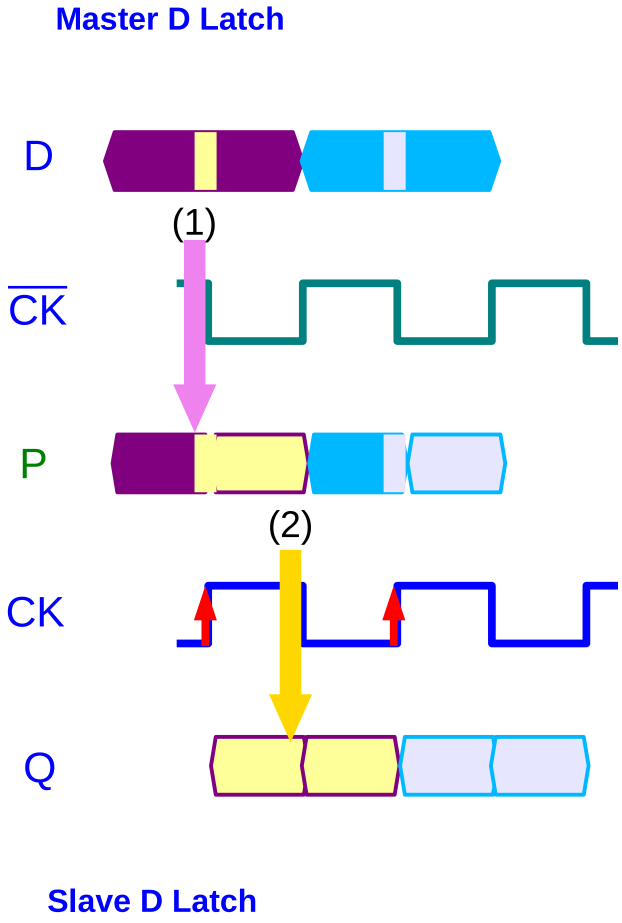
Master-Slave D FlipFlop – Rising Edge

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design



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

(2) the current <u>content</u>
P gets stored
in the slave latch

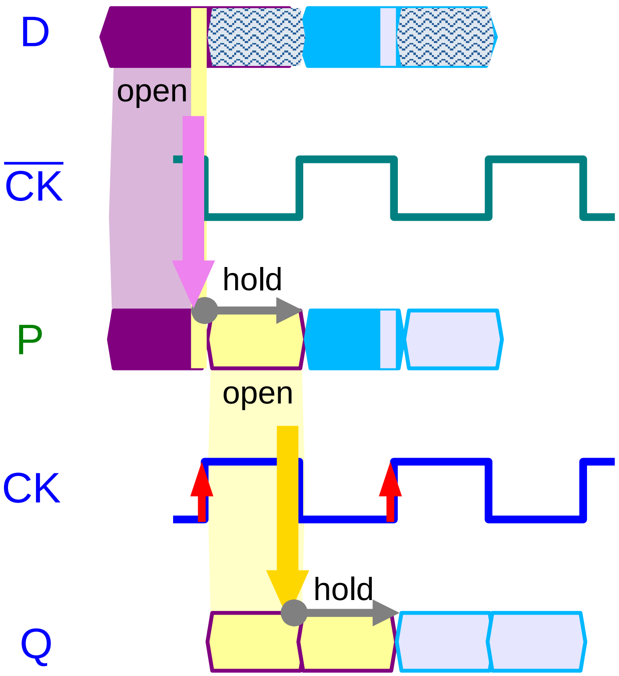


Using inverted clocks <u>enable</u>
(1) and (2) to be executed <u>sequentially</u>

Master-Slave D FlipFlop – open and hold

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

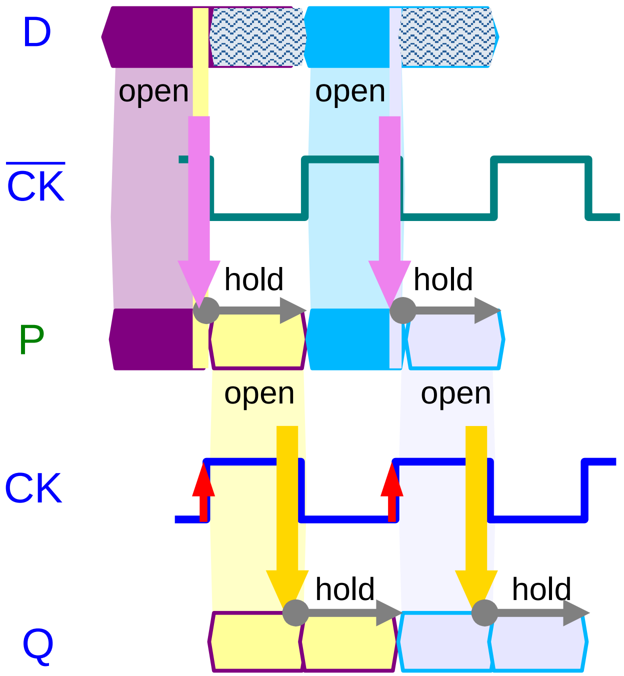
Master D Latch



Slave D Latch

▨ shaded area has <u>no</u> effect in operation

Master D Latch

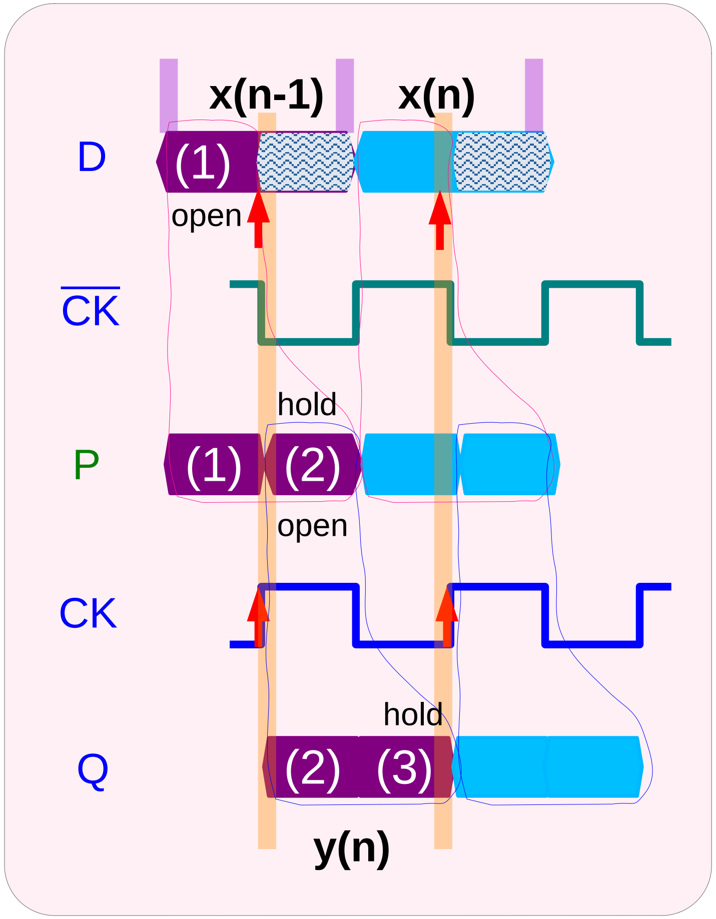


Slave D Latch

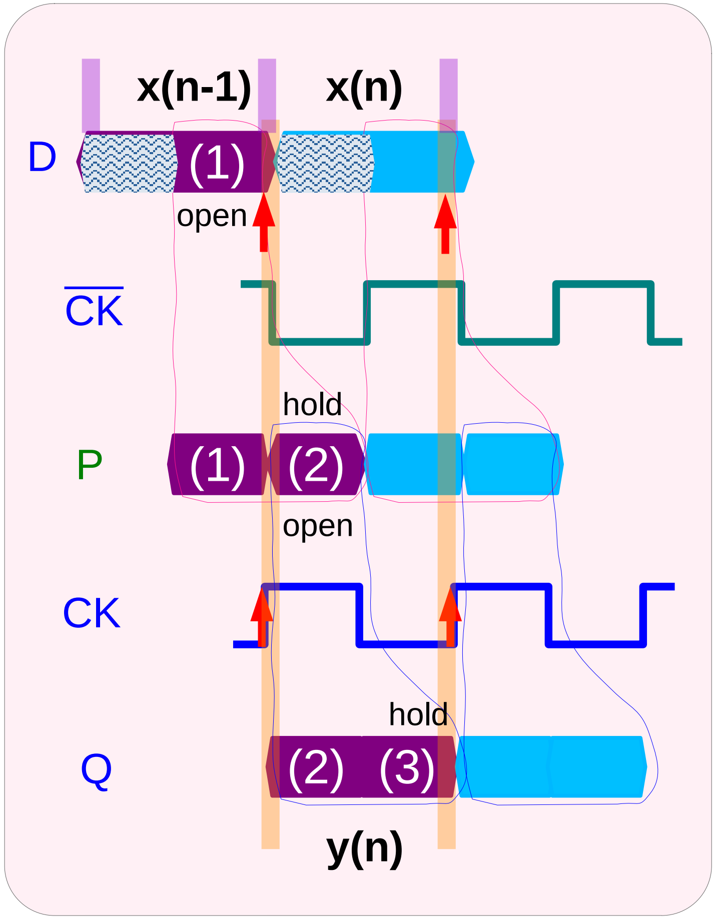
▨ shaded area has <u>no</u> effect in operation

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

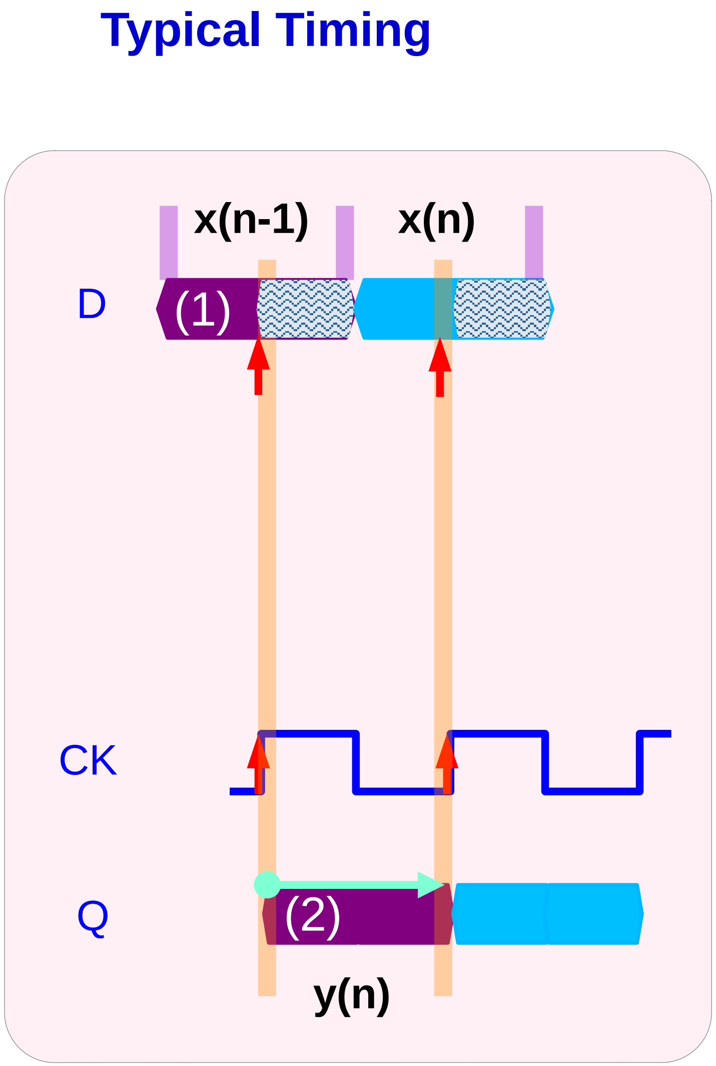
Typical Timing

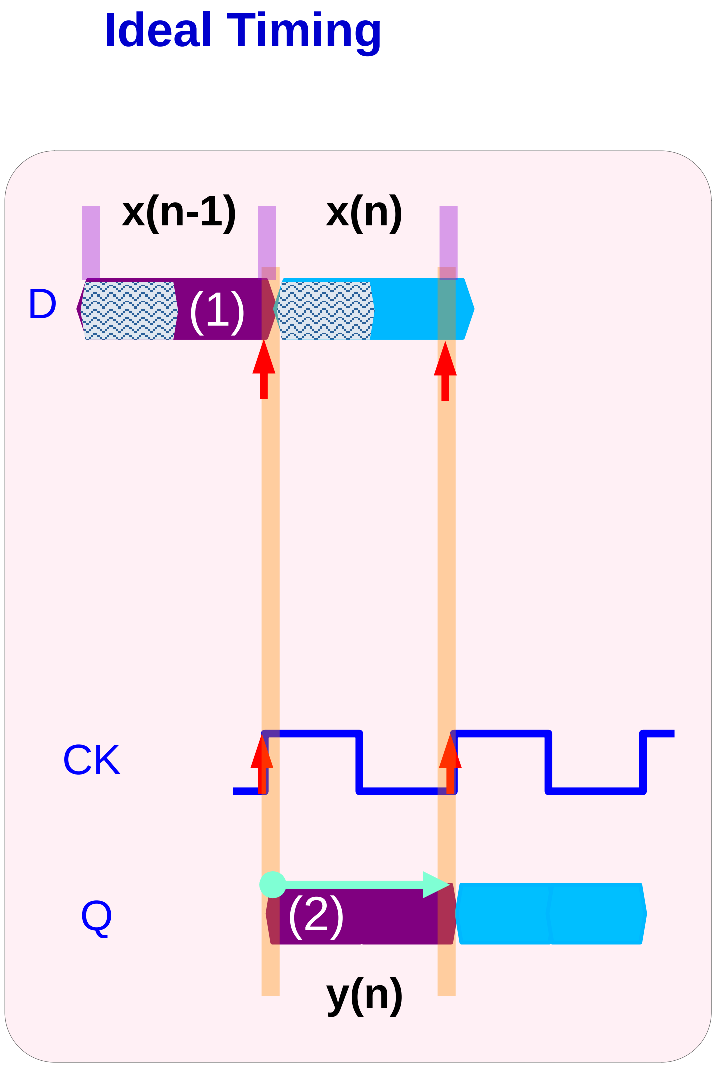


Ideal Timing



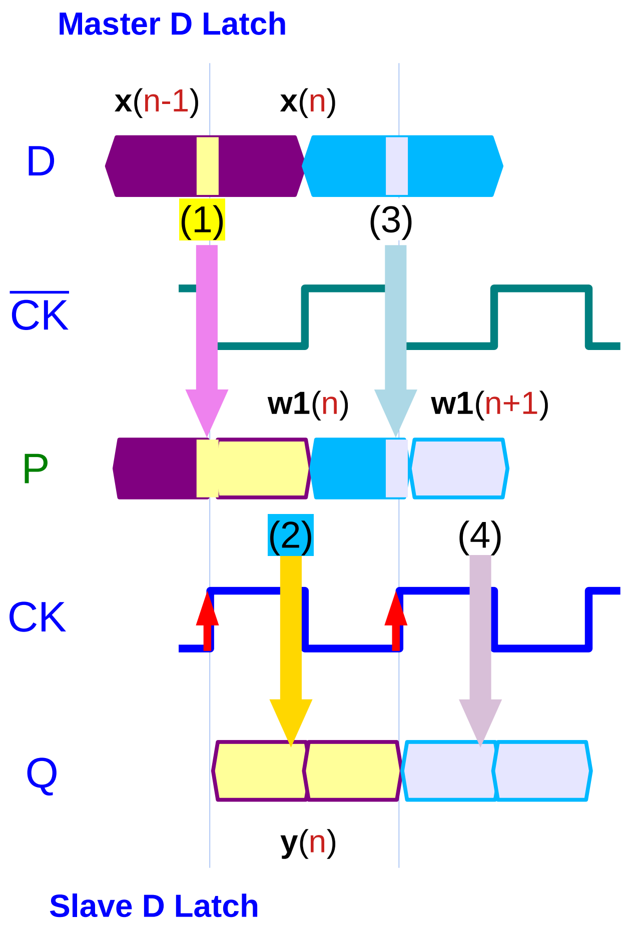
Master-Slave D FlipFlop – typical vs. ideal timing (2)

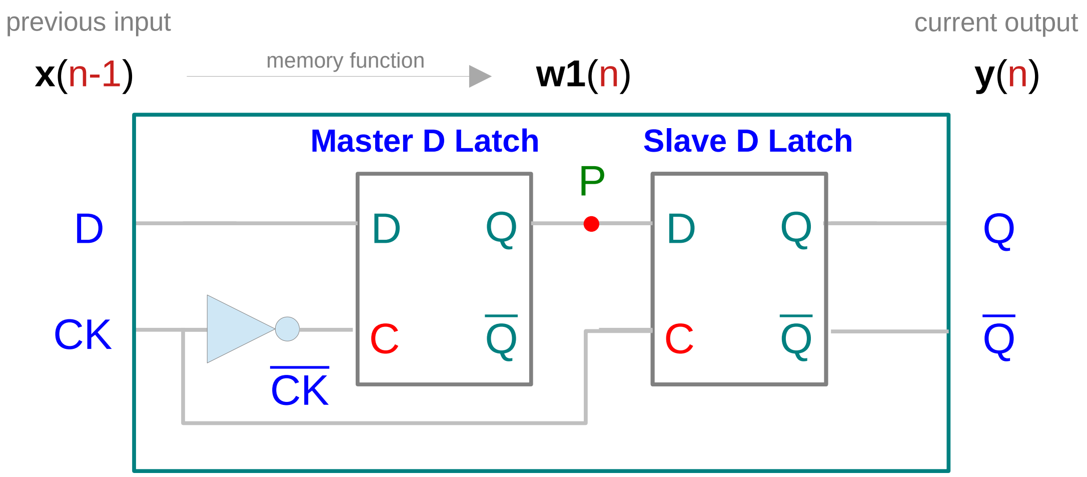




Master-Slave D FlipFlop – Rising Edge Sampling

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design





(1) the input **x**(n-1) gets stored in the master latch

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

***WR w1**(n)*

first, write

at the end of
the previous time slot n-1,
x(n-1) gets stored in **w1**(n)
for the current time slot n

(2) the content of **w1**(n) gets stored in the slave latch

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

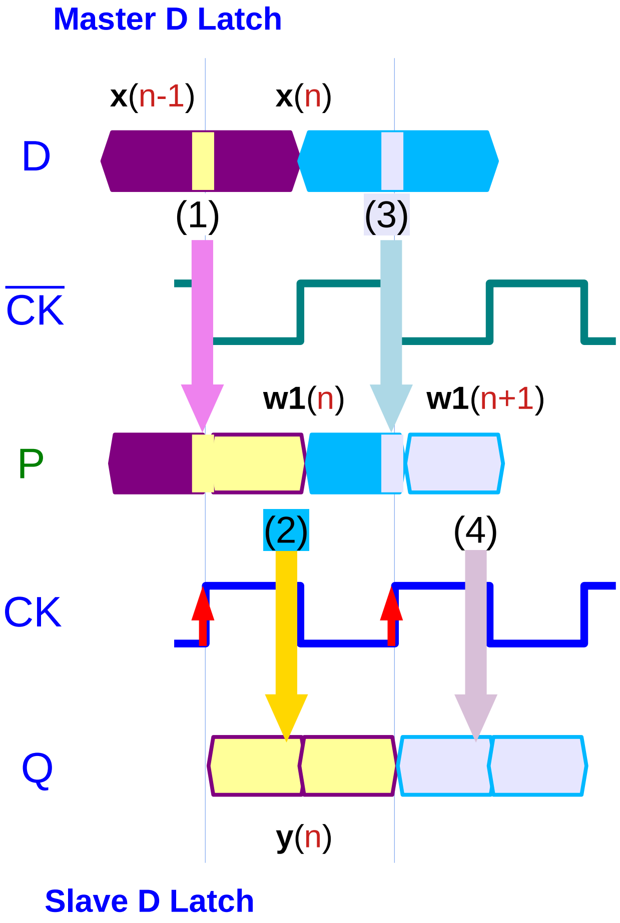
***RD w1**(n)*

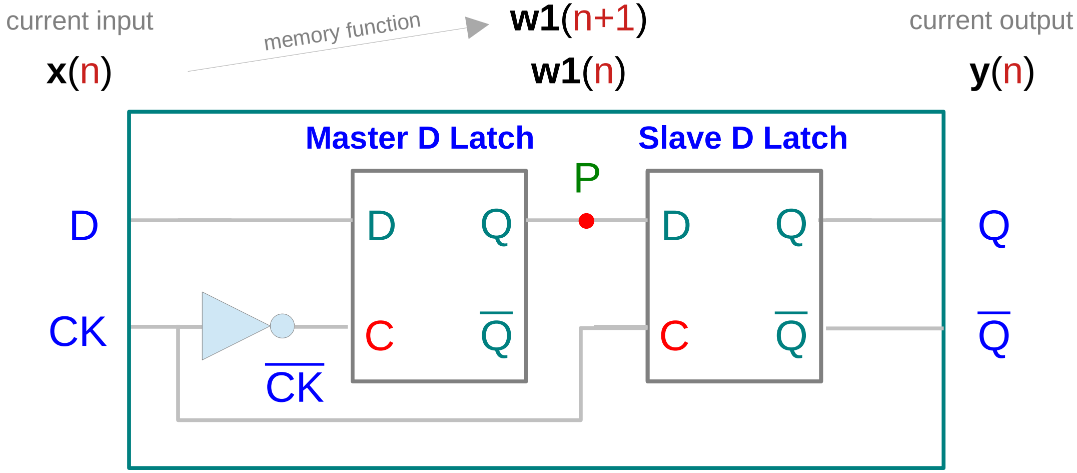
then, read

during the current time slot n,
y(n) outputs **w1**(n) which holds
the value of **x**(n-1) at the end
of the previous time slot n-1

Master-Slave D FlipFlop – Rising Edge Sampling

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design





(2) the content of **w1**(n) gets stored in the slave latch

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

***RD w1**(n)*

first, read

during the current time slot n, **y**(n) outputs **w1**(n) which holds the value of **x**(n-1) at the end of the previous time slot n-1

(3) the input **x**(n) gets stored in the master latch

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

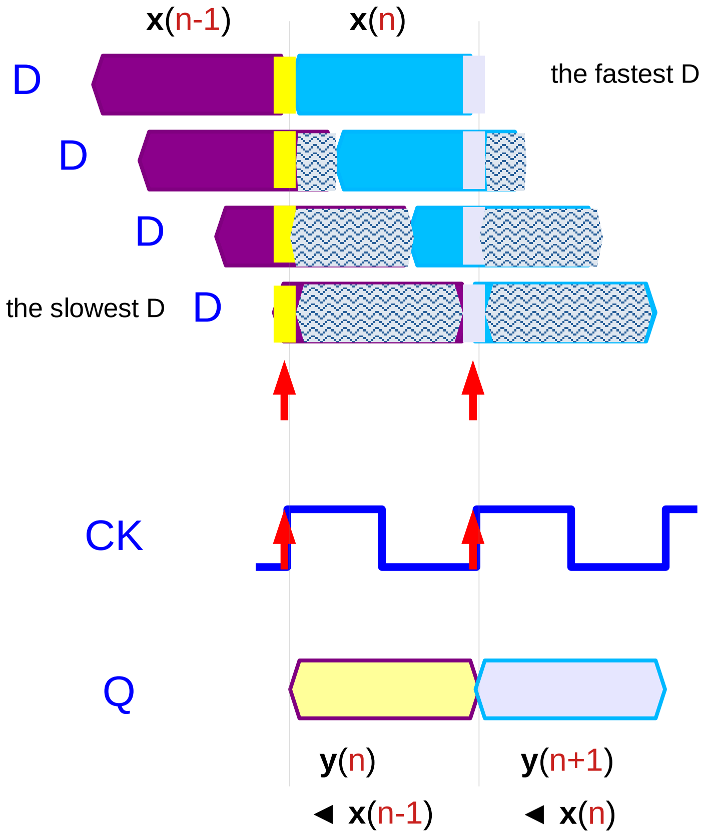
***WR w1**(n+1)*

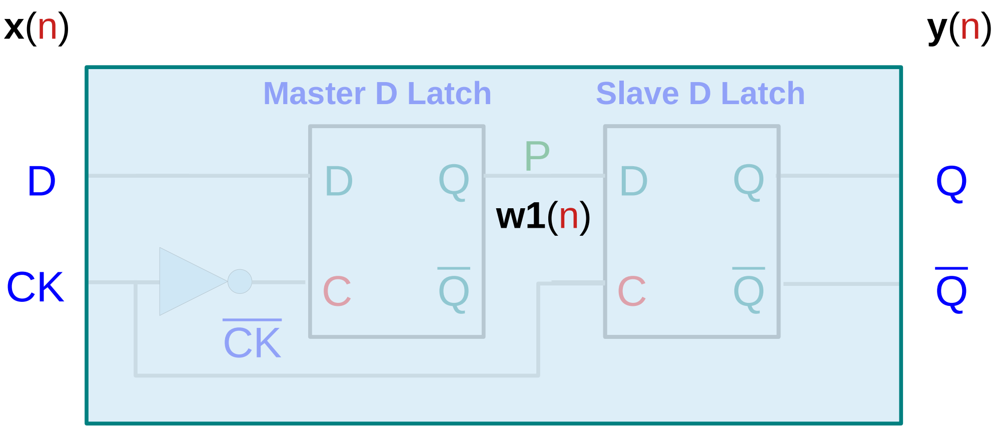
then, write

at the end of the current time slot n, **x**(n) gets stored in **w1**(n+1) for the next time slot n+1

Master-Slave D FlipFlop – Rising Edge Sampling

https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design





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

***RD w1**(n)*

first, read

during the current time slot n,
y(n) outputs **w1**(n) which holds
the value of **x**(n-1) at the end
of the previous time slot n-1

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

***WR w1**(n+1)*

then, write

at the end of
the current time slot n,
x(n) gets stored in **w1**(n+1)
for the next time slot n+1

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

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

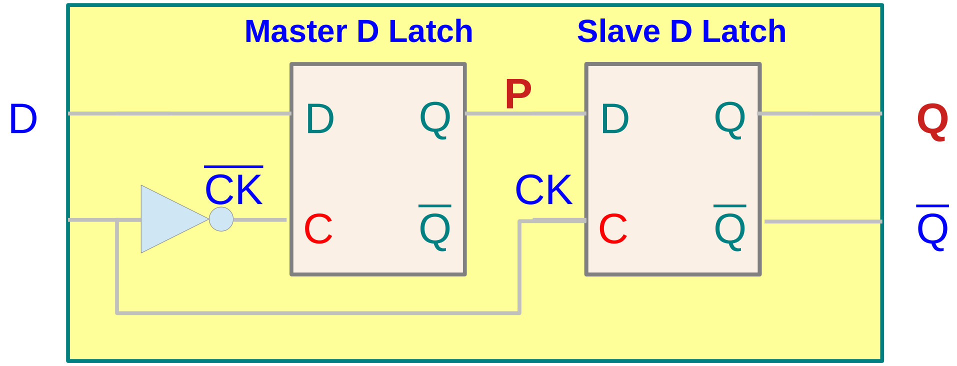
fractional numbers

x(n) → **X** *n-bit* wires at time n

y(n) → **Y** *n-bit* wires at time n

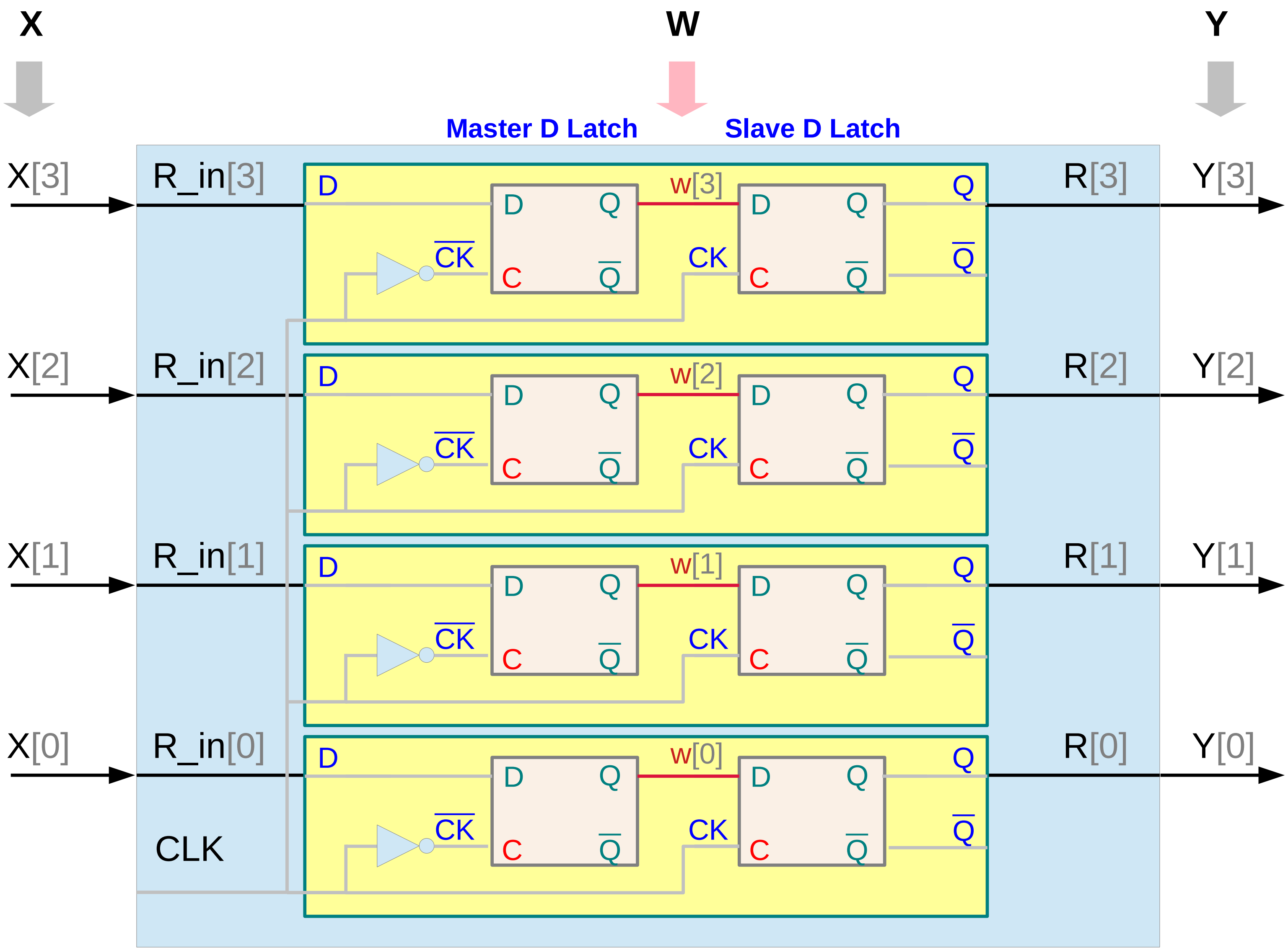
w1(n) → **W** *n-bit* registers at time n → memory elements

1. using the **P** outputs of master latches as the name for memory elements
2. using the **Q** outputs of flipflops as the name for memory elements

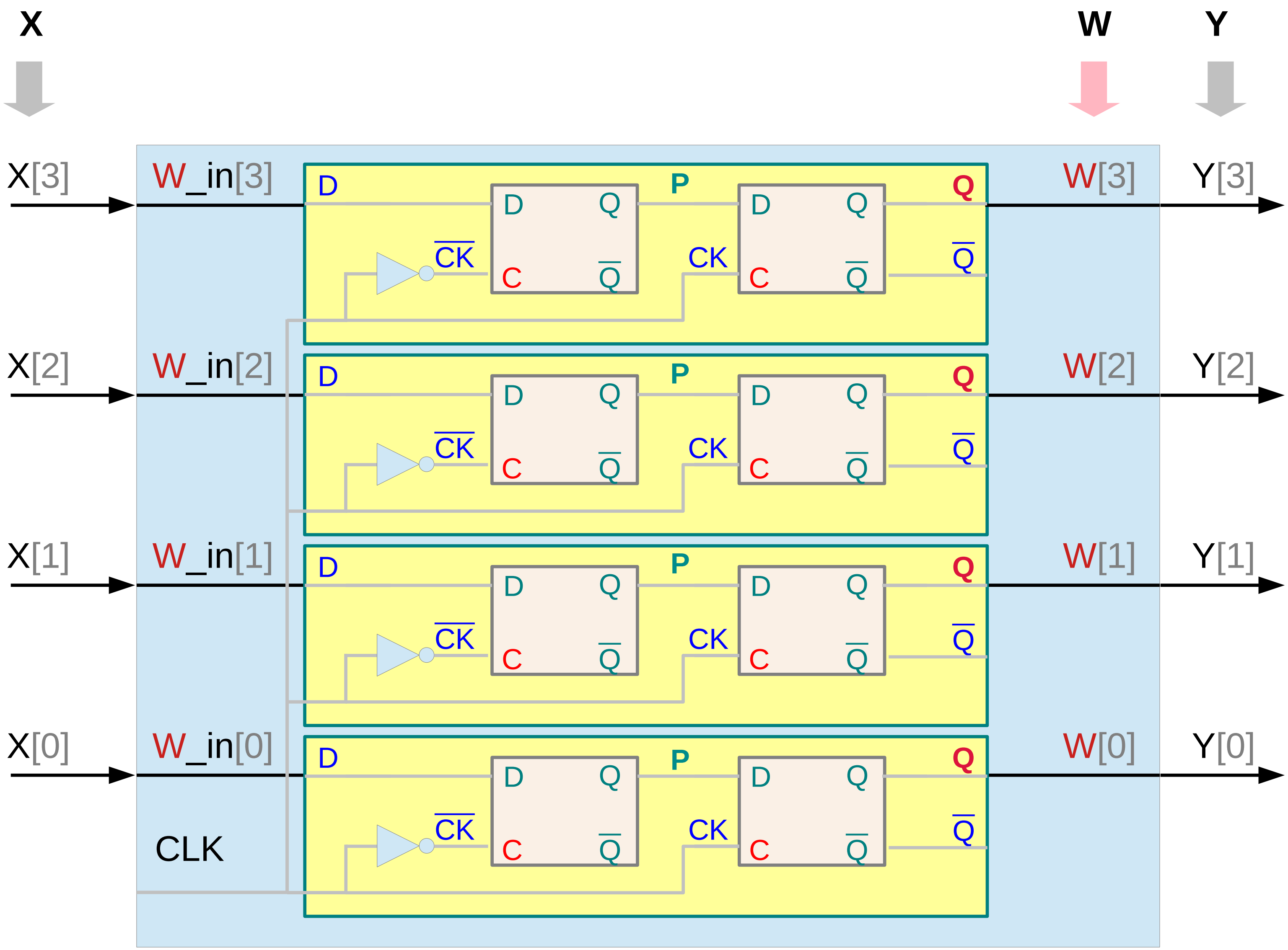


D FlipFlop using Master and Slave D Latches

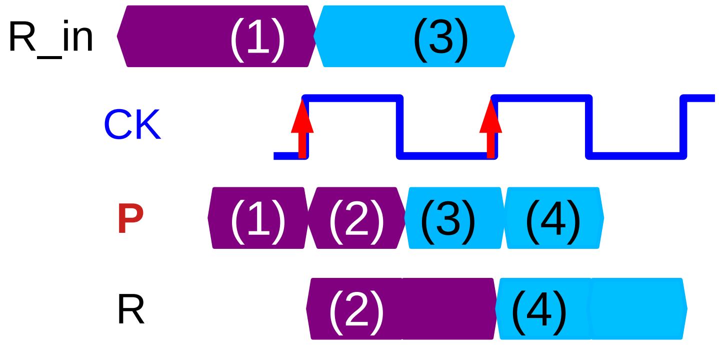
(1) Using master latch outputs **P**



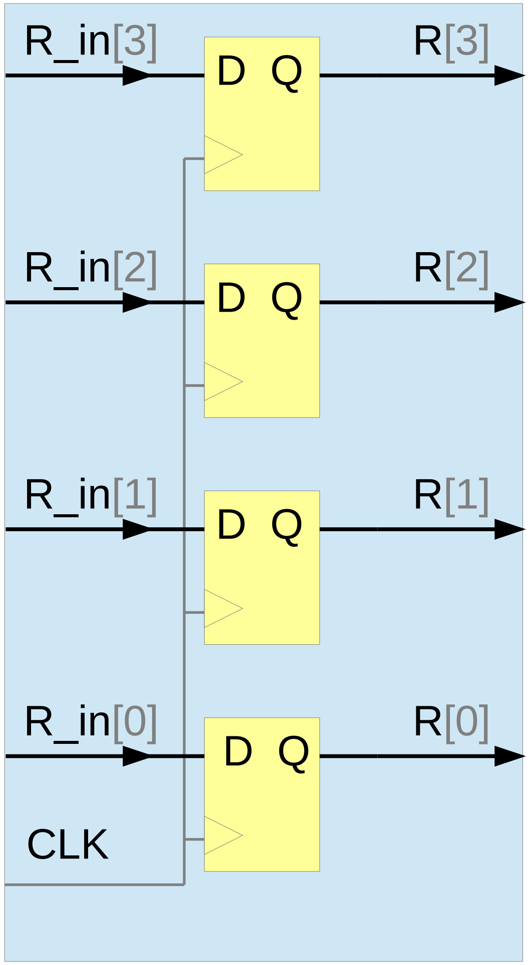
(2) Using flipflop outputs **Q**



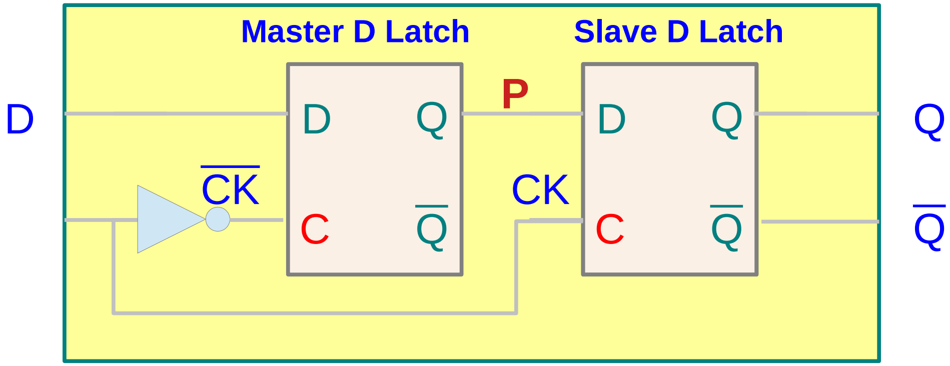
(1) 4-bit Register Using master latch outputs **P**



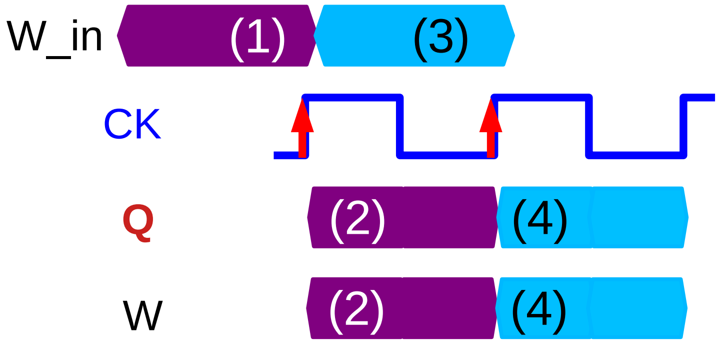
4-bit Register using 4 D flipflops



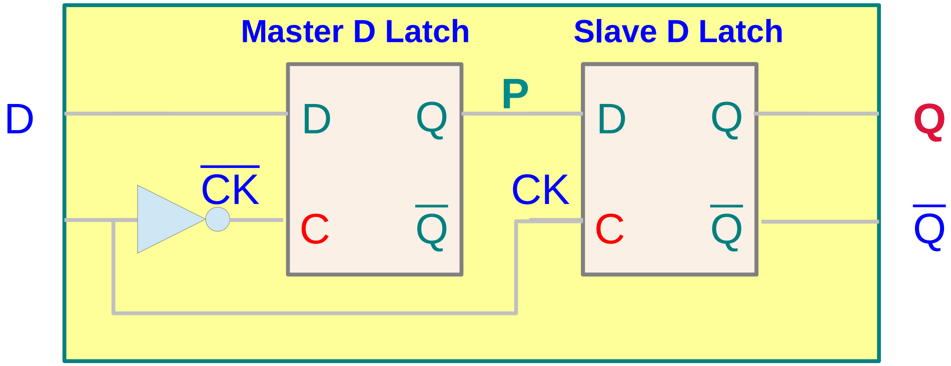
D FlipFlop using Master and Slave D Latches

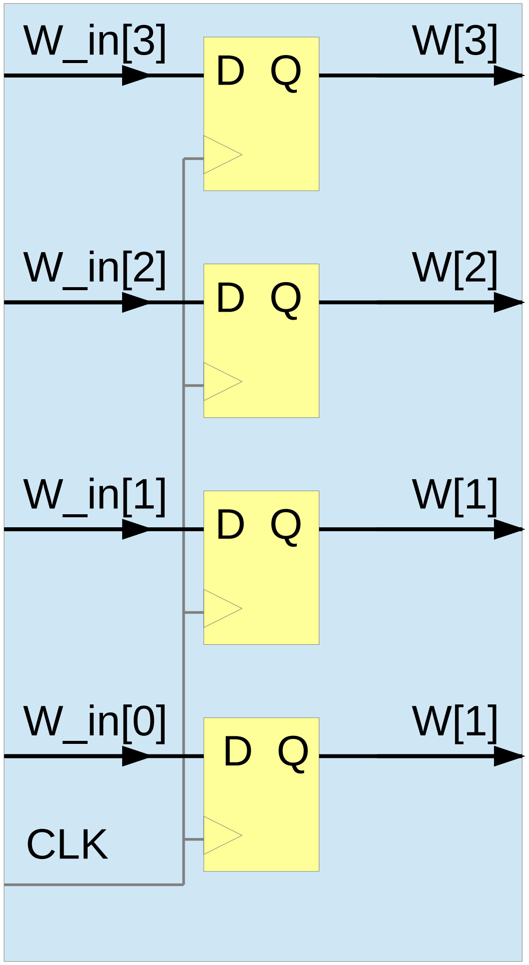


(2) 4-bit Register Using flipflop outputs **Q**



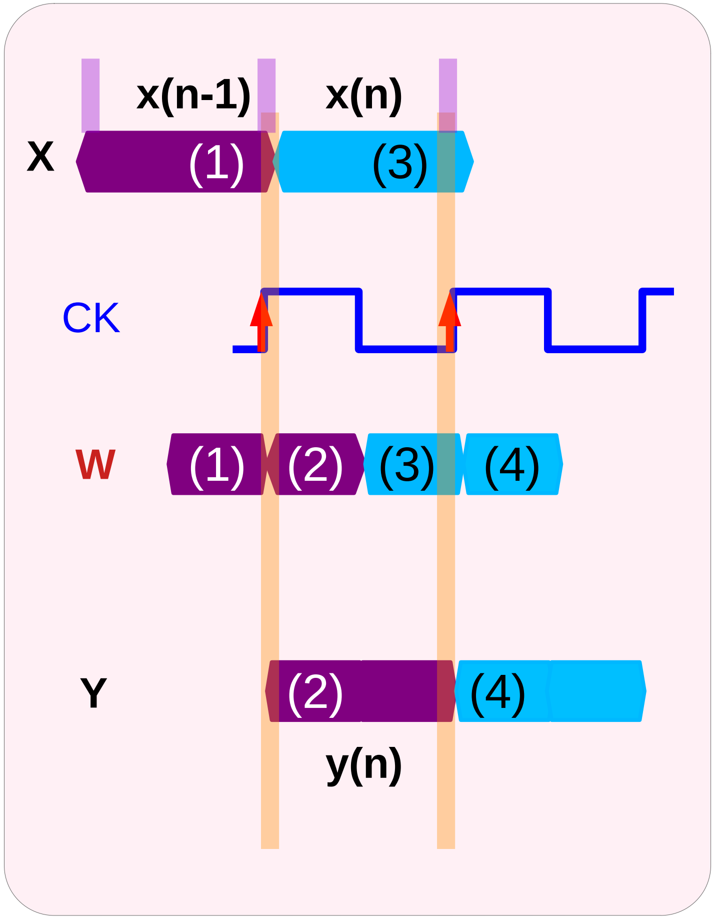
D FlipFlop using Master and Slave D Latches





(1) Timing diagrams with master latch outputs **P**

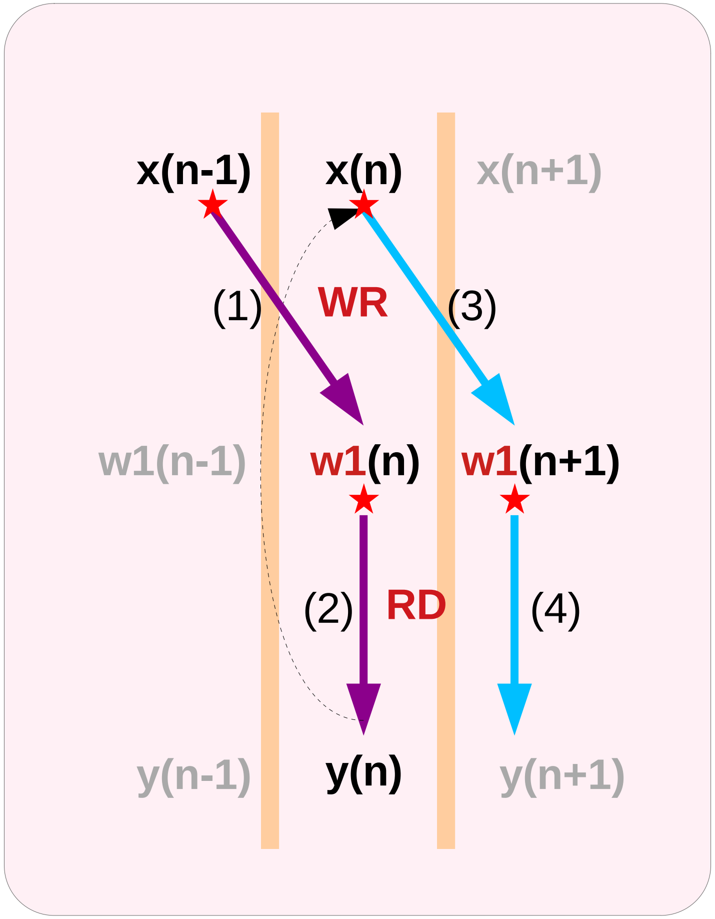
data centered view



Input

Internal State

Output

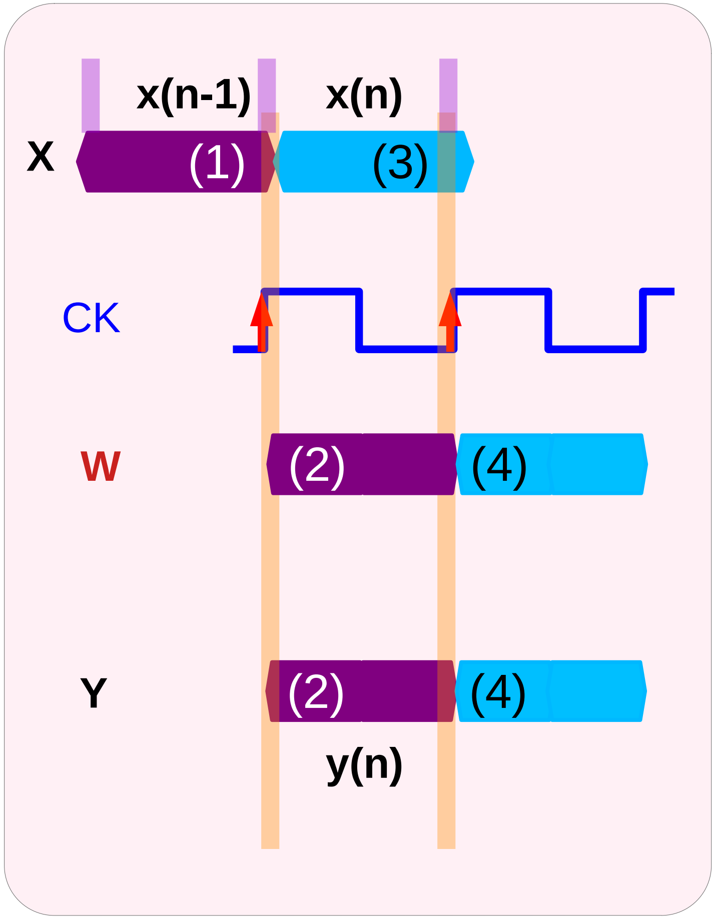


Hardware model

The same **DSP C model**

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

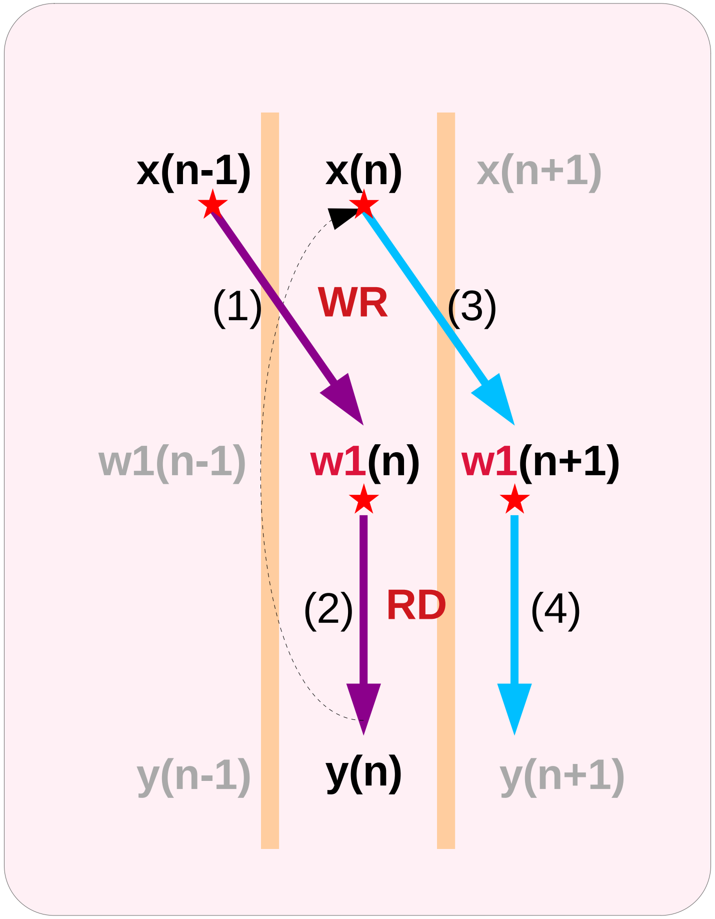
data centered view



Input

Simplified Internal State

Output

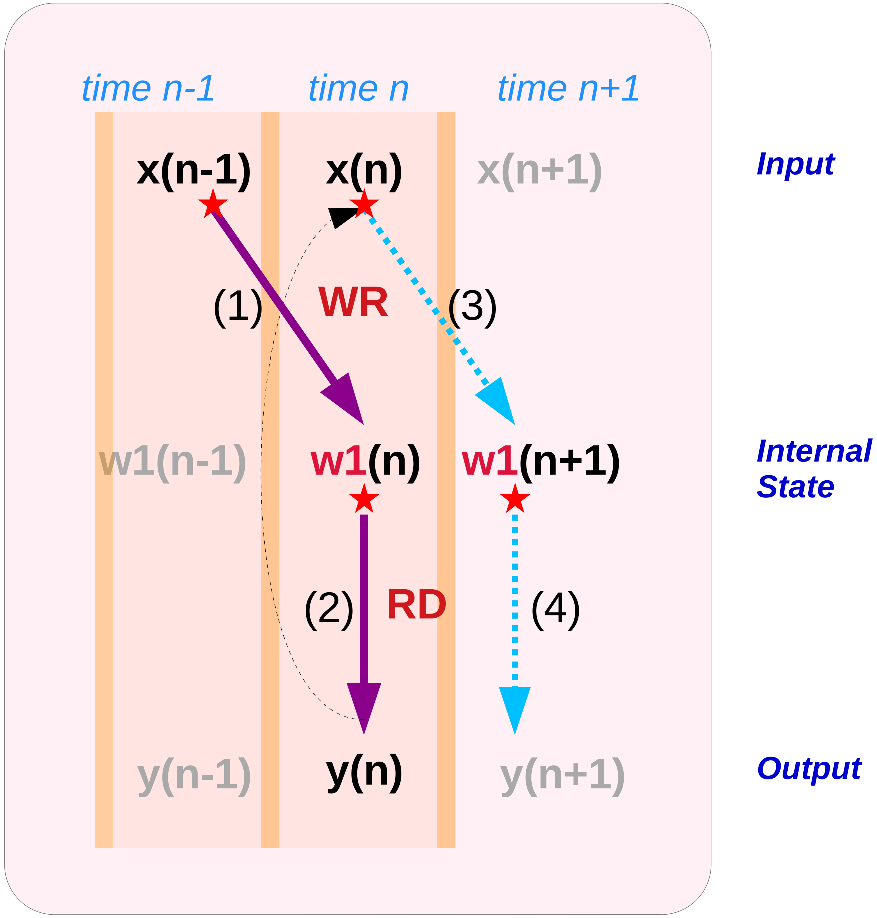


Hardware model

The same **DSP C model**

Data centered operations

data centered operations



DSP C model

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

(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

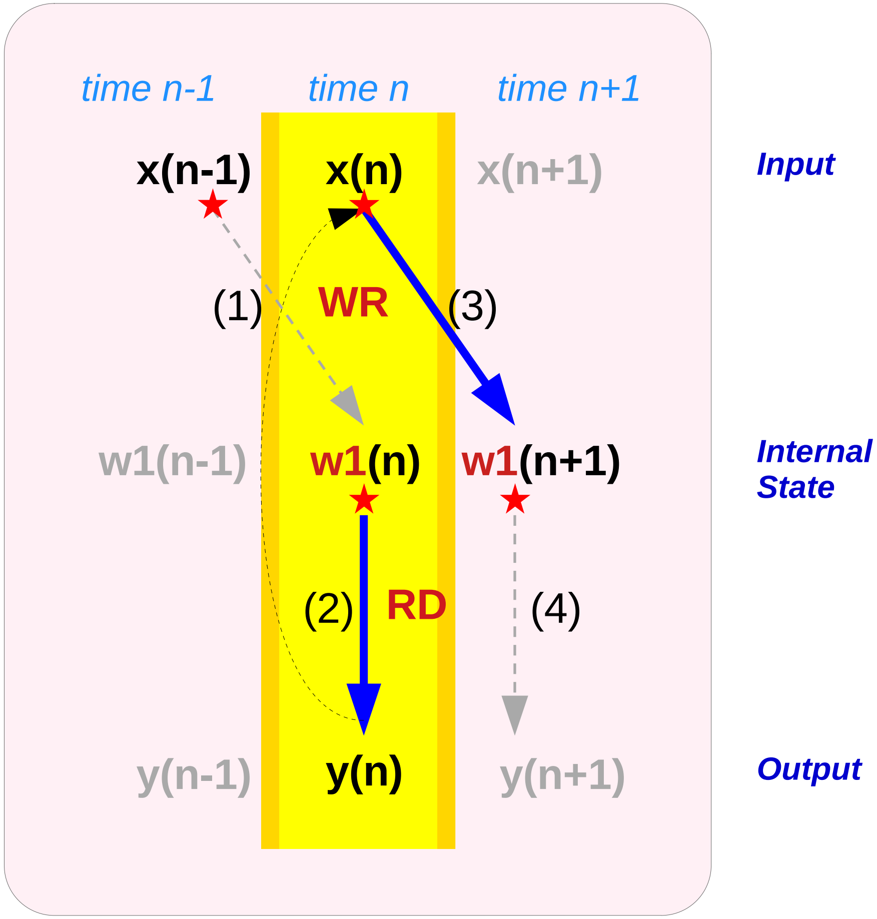
here, because the written data is read out,
yhere is no **RAW (read after write)** hazards

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

Time centered operations

time centered operations

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



DSP C model

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

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

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

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

Data vs. time centered operations (1)

data centered operations

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

time centered operations

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

time centered operations

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

time n-1

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

time n

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

time n+1

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

time n+2

W	= X	*WR W*
Y	= W	*RD 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)*	**Y = W1**	*output y(n-1) from W1*	*time n-1*
w1(n) = **x**(n-1)	*input x(n-1) to w1(n)*	**W1 = X**	*input x(n-1) to W1*	
y(n) = **w1**(n)	*output y(n) from w1(n)*	**Y = W1**	*output y(n) from W1*	*time n*
w1(n+1) = **x**(n)	*input x(n) to w1(n+1)*	**W1 = X**	*input x(n) to W1*	
y(n+1) = **w1**(n+1)	*output y(n+1) from w1(n+1)*	**Y = W1**	*output y(n+1) from W1*	*time n+1*
w1(n+2) = **x**(n+1)	*input x(n+1) to w1(n+2)*	**W1 = X**	*input x(n+1) to W1*	
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*)	**W1**	**= X**	***WR W1***
(2)	**y**(n) = **w1**(n)	***RD w1***(*n*)	**Y**	= **W1**	***RD W1***

Time centered operations

time centered operations

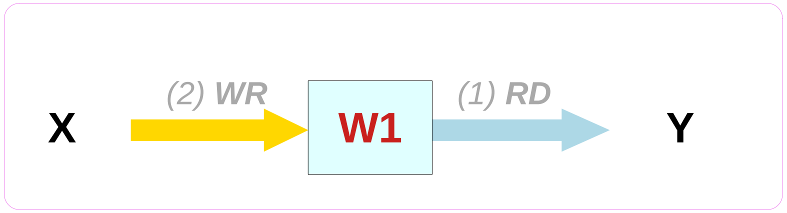
in the Register Transfer Level

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

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

Register Transfer Level



at time n

Y = **W1**

W1 = **X**

at time n+1

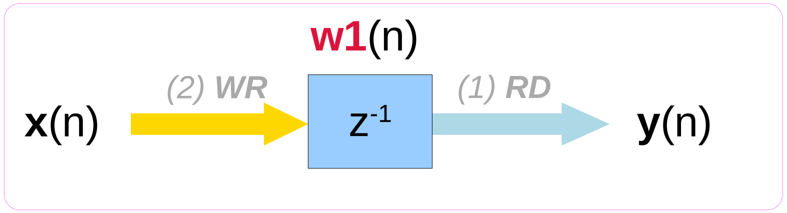
Y = **W1**

W1 = **X**

(1) **RD**

(2) **WR**

DSP C Model



at time n

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

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

at time n+1

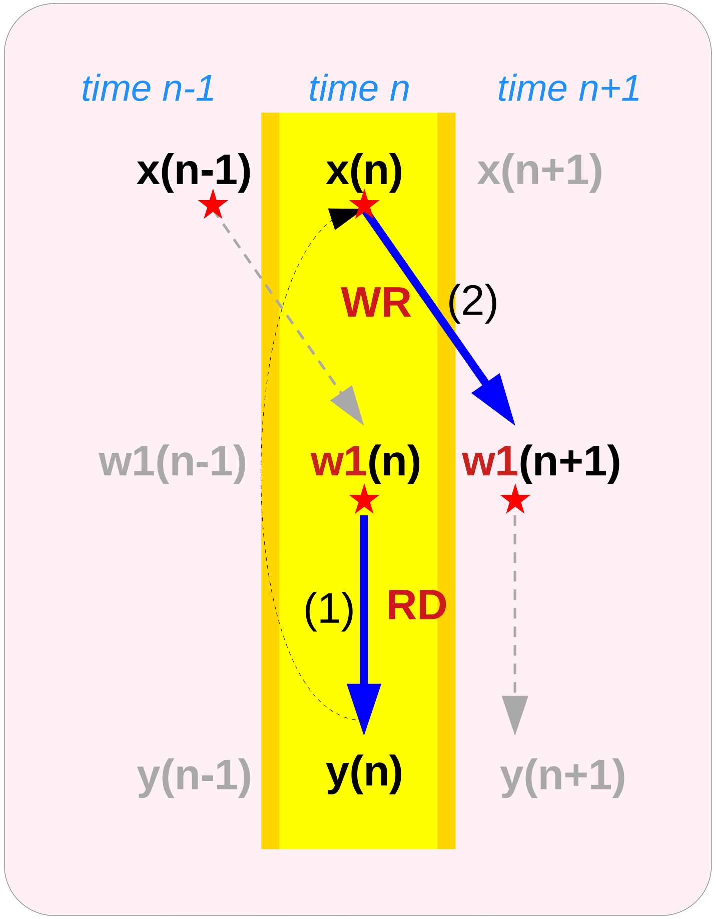
y(n+1) = **w1**(n+1)

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

(1) **RD**

(2) **WR**

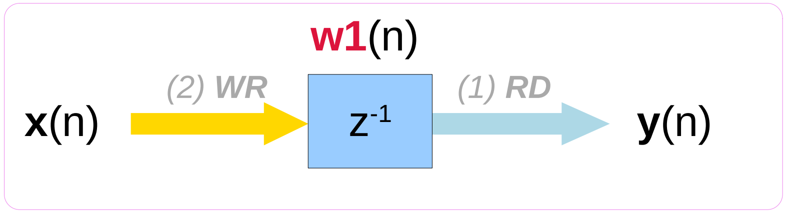
time centered operations



Input

DSP C model

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



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

at time n

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

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

at time n+1

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

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

(1) **RD** *read internal state*

(2) **WR** *update internal state*

at time n,

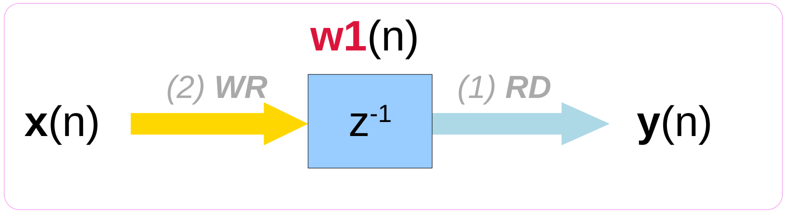
1) the content of the register **w1**(n)
becomes the output **y**(n)

2) the input **x**(n) is saved and
becomes the new content **w1**(n+1)

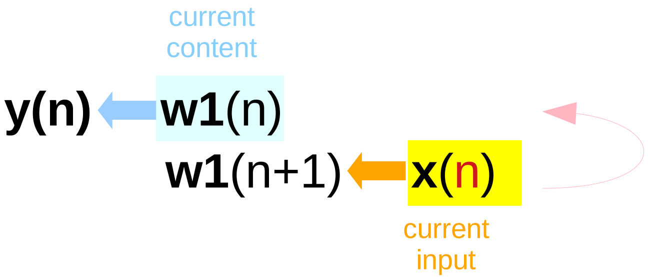
RD access of **w1**(n) = **x**(n-1)

WR access of **w1**(n+1) = **x**(n)

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



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

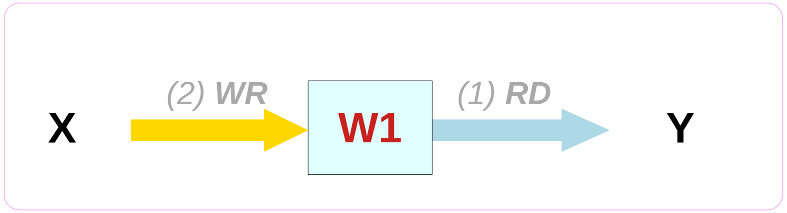


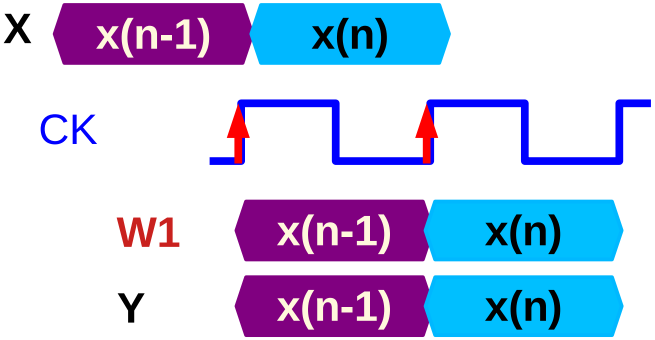
a register holding the previous input sample **x**(n-1)

(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 one sampling instant and
become the output at the next time n+1

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

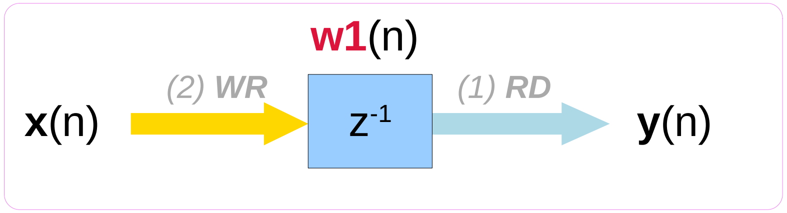




simulate a clocked hardware
ignoring delay constraints in hardware

zero-delay simulation

Delay element modeling

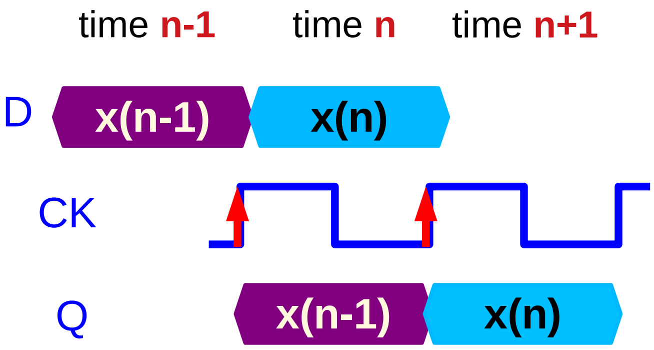


w1(n) → y(n)	y(n) = w1(n) (1) **RD** old w1
x(n) → w1(n+1)	w1(n+1) = x(n) (2) **WR** new w1

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**



simulate a clocked hardware
ignoring delay constraints in hardware

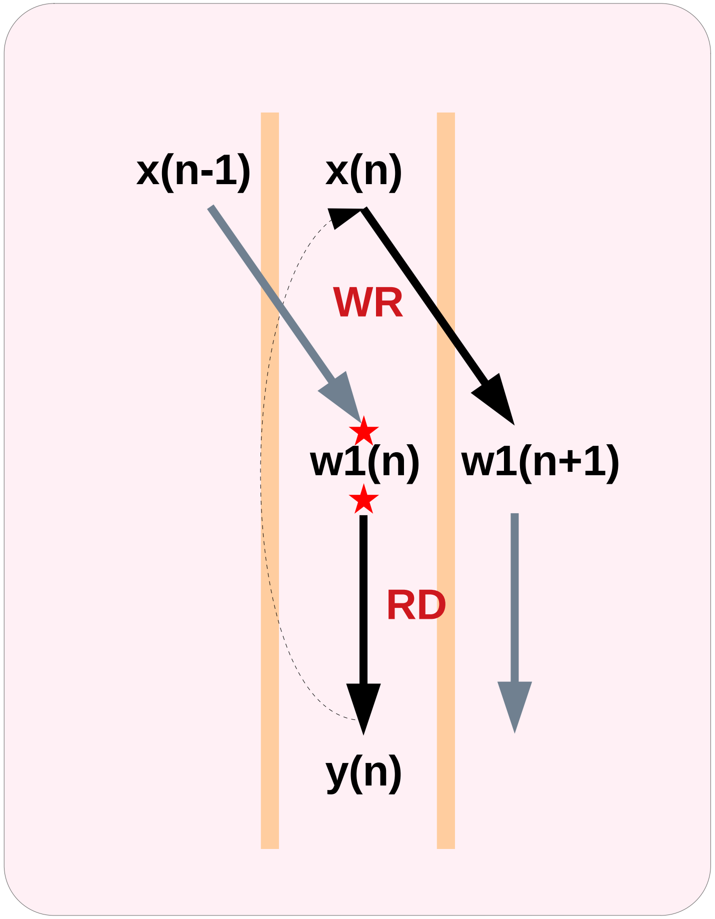
zero-delay simulation

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

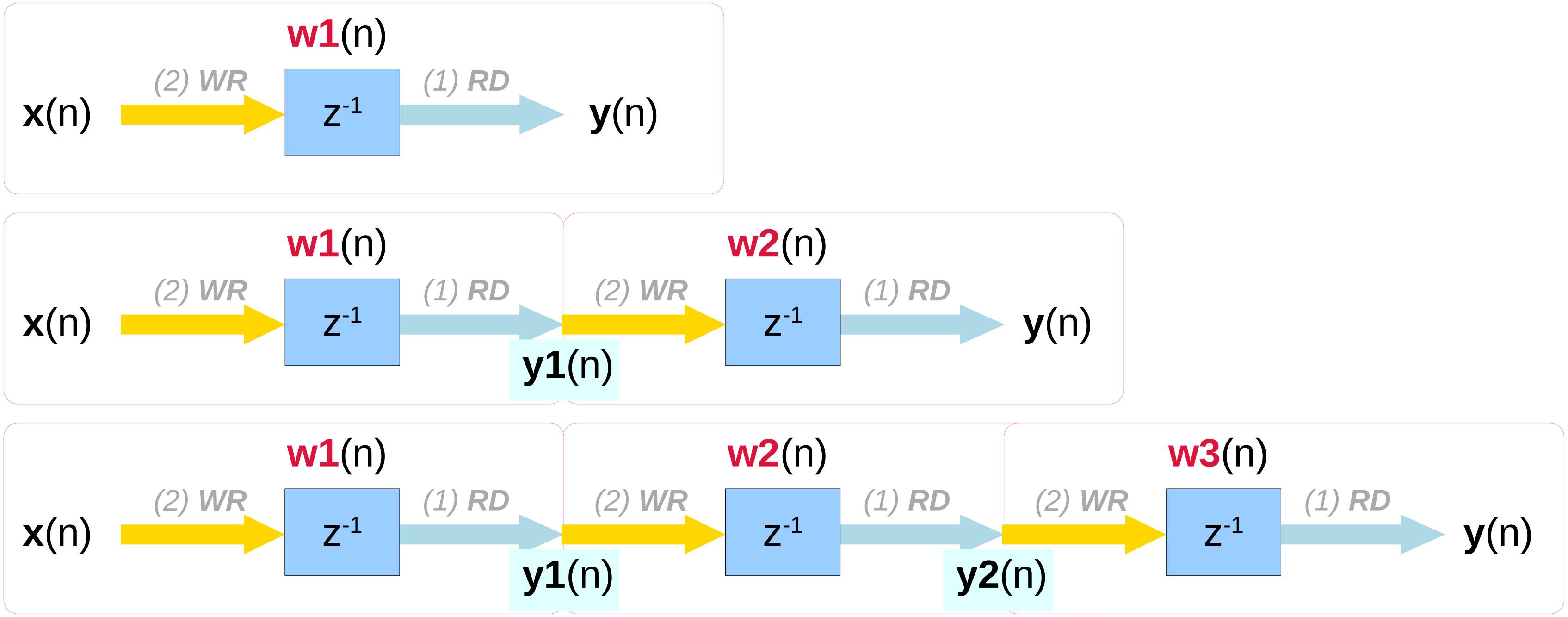
w1(n+1) = **x**(n)	(2) **WR**	new w1
y(n) = **w1**(n)	(1) **RD**	old w1
w1(n+2) = **x**(n+1)	(2) **WR**	new w1
y(n+1) = **w1**(n+1)	(1) **RD**	old w1

WAR (Write after Read) Violation



DSP C model

Single, Double, Triple Delay – Summary



single delay

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

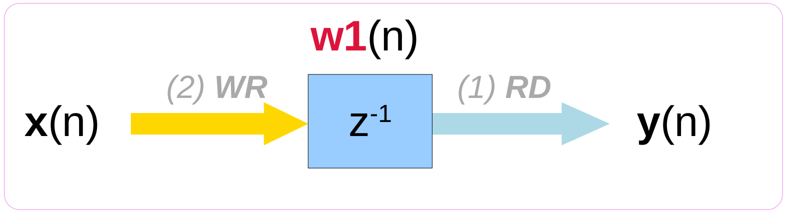
double delay

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

Single Delay – IO Equations



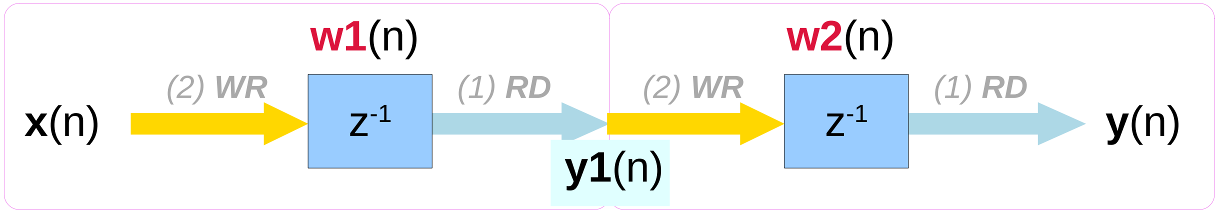
single delay

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

output

input

Double Delay – IO Equations



y1(n) = **w1**(n)
w1(n+1) = **x**(n)

→

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

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

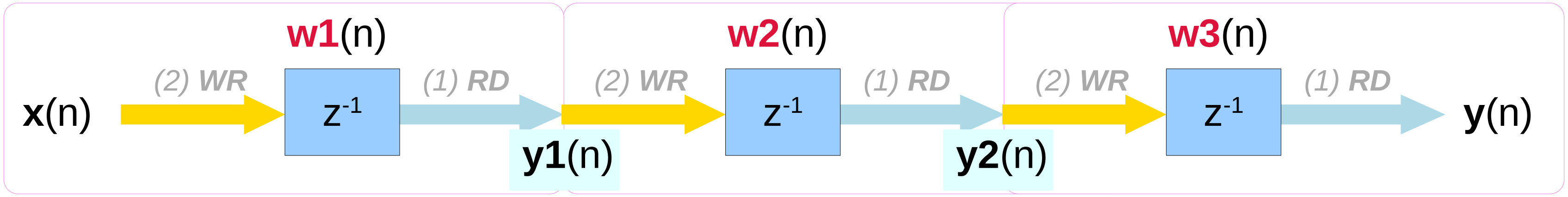
y(n) = **w2**(n)
w2(n+1) = **w1**(n)

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

output

input

Triple Delay – IO Equations



y1(n) = w1(n)
w1(n+1) = x(n)

y2(n) = w2(n)
w2(n+1) = y1(n)

y(n) = w3(n)
w3(n+1) = y2(n)

w1(n+1) = x(n)

w2(n+1) = w1(n)

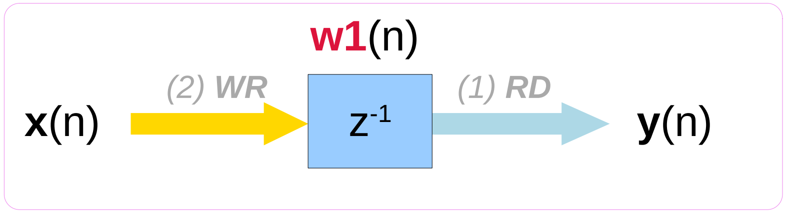
y(n) = w3(n)
w3(n+1) = w2(n)

y(n) = w3(n)
w3(n+1) = w2(n)
w2(n+1) = w1(n)
w1(n+1) = x(n)

output

input

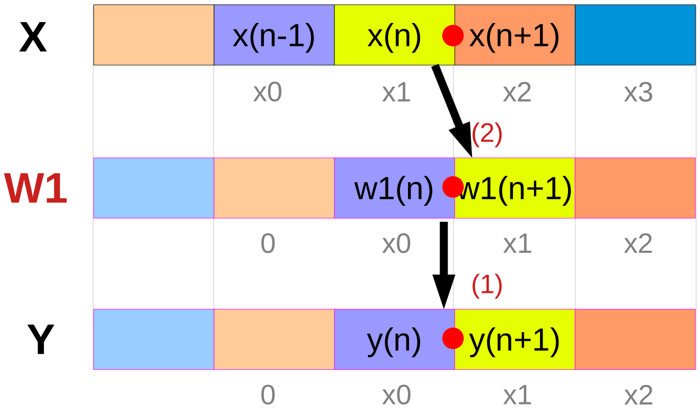
Single Delay



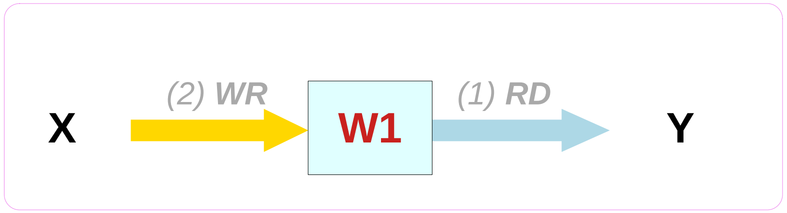
single delay

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

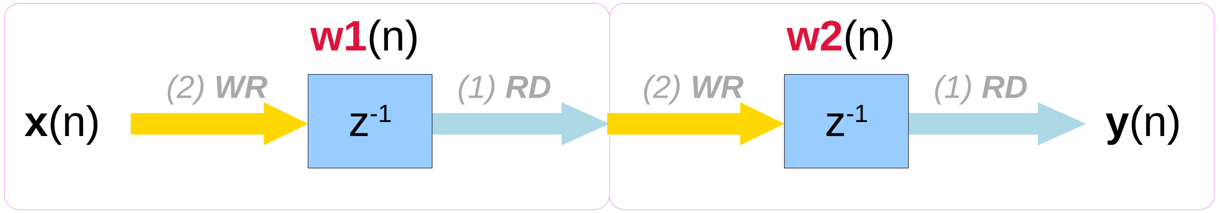
output
input



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

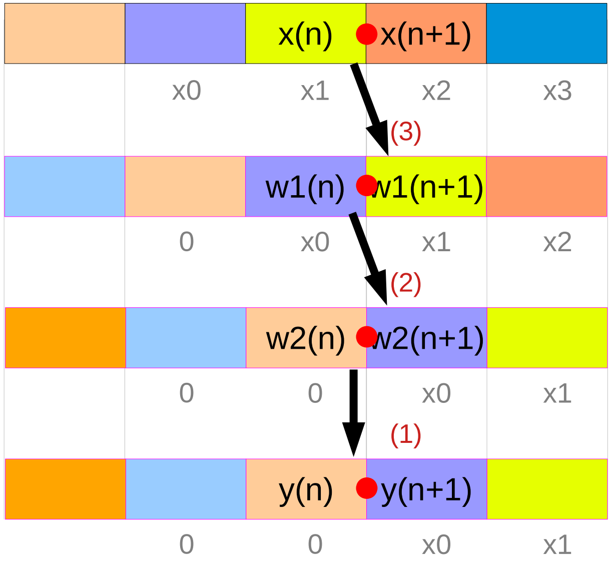


Double Delay



double delay

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



x(n) x(n+1)

x0 x1 x2 x3

(3)

w1(n) w1(n+1)

0 x0 x1 x2

(2)

w2(n) w2(n+1)

0 0 x0 x1

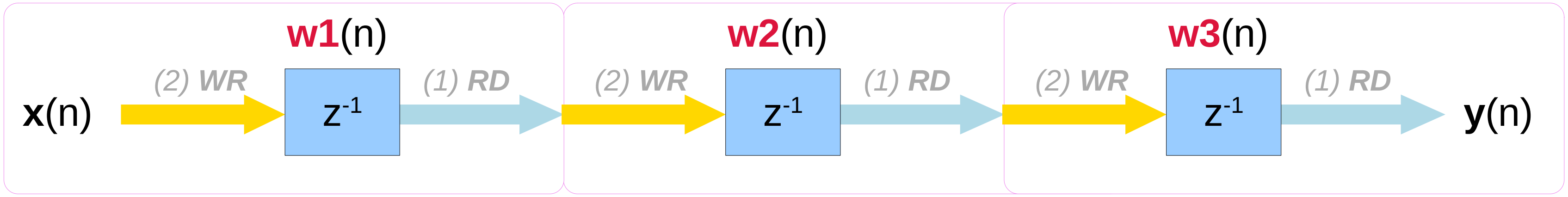
(1)

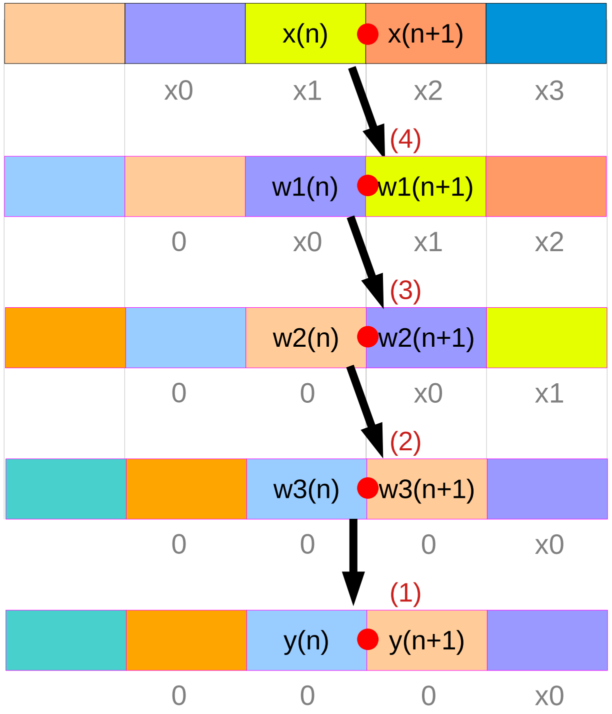
y(n) y(n+1)

0 0 x0 x1

n	**x**(n)		**w1**(n)		**w2**(n)		**y**(n)
0	x0	(3)	0	(2)	0		0
1	x1		x0		0	(1)	0
2	x2		x1		x0		x0
3	x3		x2		x1		x1
4	x4		x3		x2		x2

Triple Delay





triple delay

y(n) = **w3**(n) output

w3(n+1) = **w2**(n)

w2(n+1) = **w1**(n)

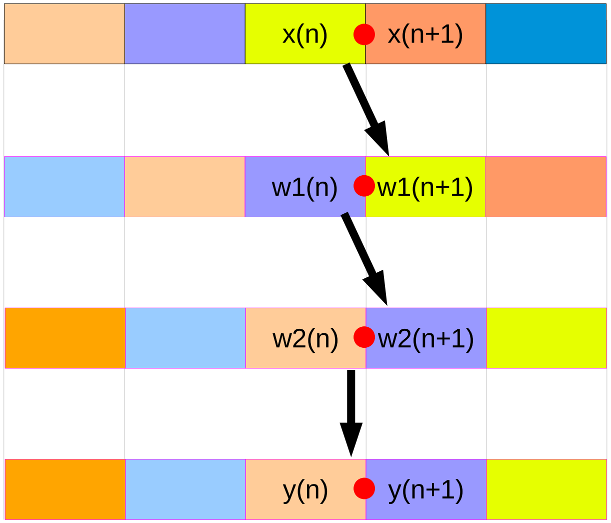
w1(n+1) = **x**(n) input

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

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

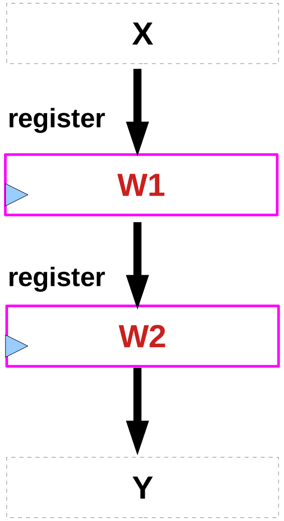
Delay C Model

Timing Chart



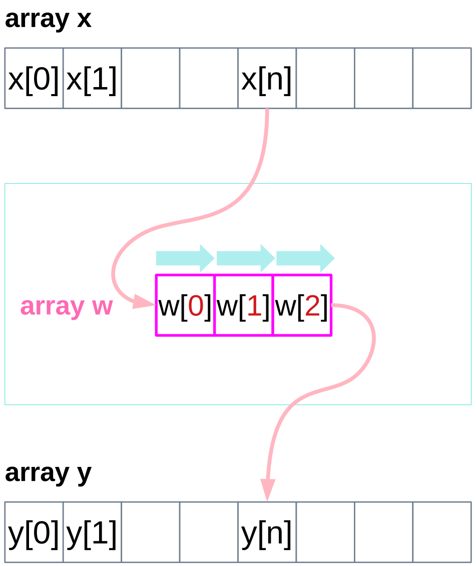
y(n) = **w2**(n)
w2(n+1) = **w1**(n)
w1(n+1) = x(n)

Register Transfer



Y = W2
W2 = W1
W1 = X

DSP C Model for simulation



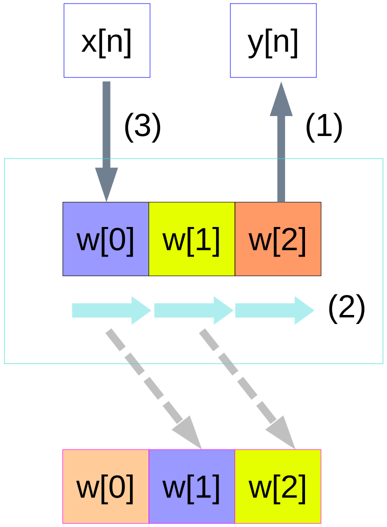
y[n] = w[2]
w[2] = w[1]
w[1] = w[0]
w[0] = x[n]

IO Equations for the Triple Delay (1)

$$\begin{aligned} y(n) &= w2(n) \\ w2(n+1) &= w1(n) \\ w1(n+1) &= w0(n) \\ w0(n+1) &= x(n) \end{aligned}$$

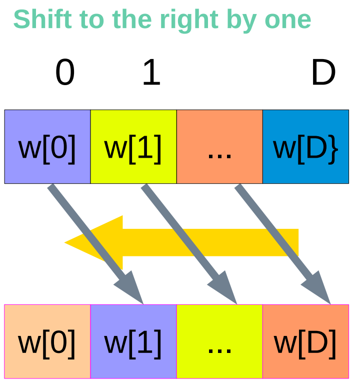
D = 2, 1

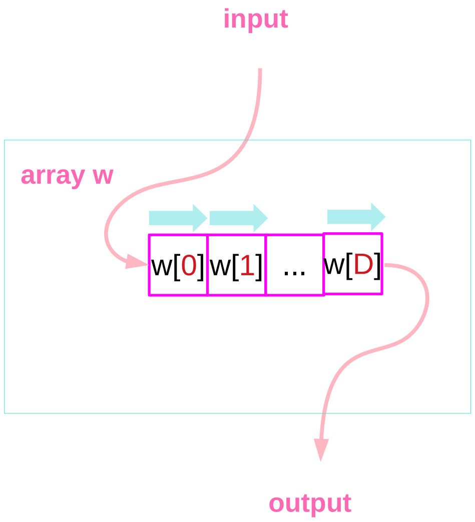
```
y[n]  = w[2]          // (1) get the output
w[2] = w[1]           // (2) shift
w[1] = w[0]           // (2) shift
w[0] = x[n]           // (3) put the input
```



delay.c

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

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

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

        // reverse-order updating
}
```



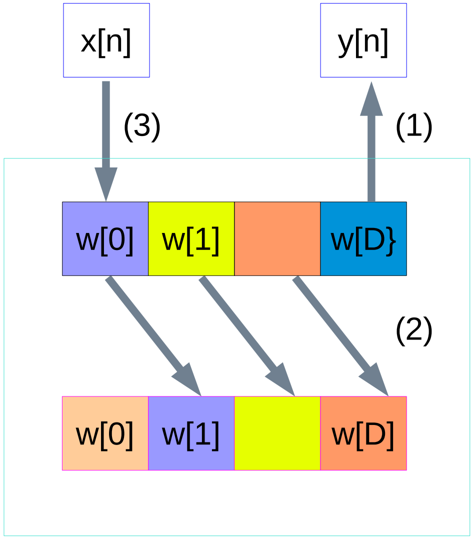


order of execution

```
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) get output y[n]
    w[0] = x[n];          // (3) read input x[n]
    delay(D, w);          // (2) update delay line
}


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

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

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

        // reverse-order updating
}
```



Delay Functions

$y(n) \quad = w_1(n)$
$w_1(n+1) = x(n)$

$y(n) \quad = w_2(n)$
$w_2(n+1) = w_1(n)$
$w_1(n+1) = x(n)$

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

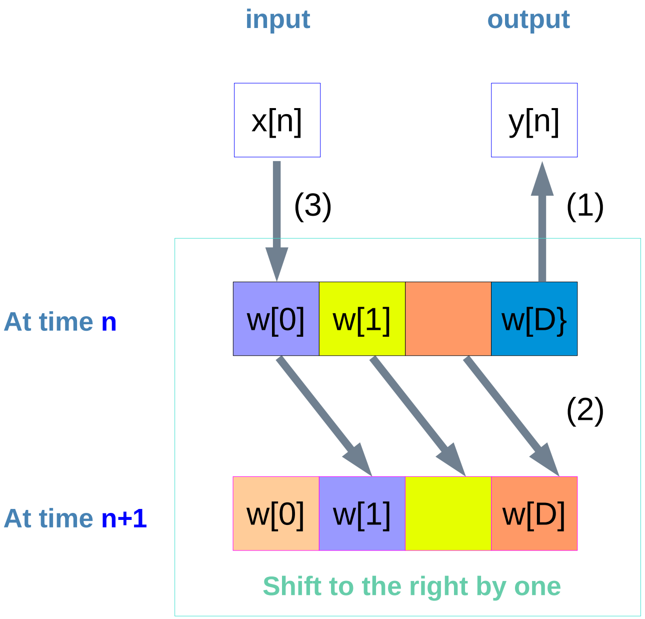
$y(n) \quad = w_D(n)$
$w_0(n) \quad = x(n)$
$w_i(n+1) = w_{i-1}(n),$
$i = D, D-1, \ldots, 2, 1$

time index : n

memory location : W_i
memory index : i

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

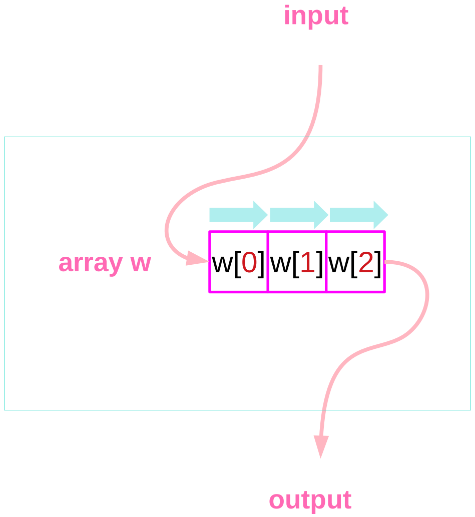
the current value at w_{i-1}
will become
the next value at w_i



Holding a delayed input sequence

$w_0(n)= x(n)$
$w_1(n)= x(n-1)= w_0(n-1)$
$w_2(n)= x(n-2)= w_1(n-1)$
$w_3(n)= x(n-3)= w_2(n-1)$

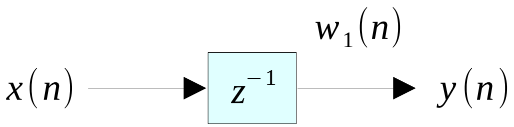
$y(n) \quad = w_3(n)$
$w_3(n+1) = w_2(n)$
$w_2(n+1) = w_1(n)$
$w_1(n+1) = x(n)$



$y(n) \quad = w_3(n) \quad = w_2(n-1) \;= w_1(n-2) \;= x(n-4)$
$w_3(n) \quad = w_2(n-1) \;= w_1(n-2) \;= x(n-3)$
$w_2(n) \quad = w_1(n-1) \;= x(n-2)$
$w_1(n) \quad = x(n-1)$

$y(n) \quad = w_3(n)$
$w_3(n) \quad = w_2(n-1)$
$w_2(n) \quad = w_1(n-1)$
$w_1(n) \quad = x(n-1)$

Single Delay (1)



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

$\Downarrow$

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

$$y(n) = w_1(n)$$

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

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

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

$$w_1(0) = 0$$

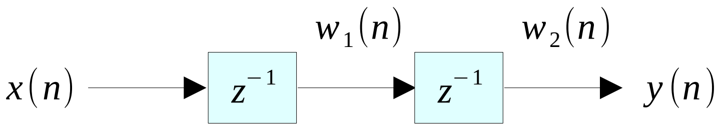
$[x_0, x_1, x_2, x_3, \ldots] \rightarrow [0, x_0, x_1, x_2, \ldots]$

for each input sample x do:

```
    $y := w_1$
    $w_1 := x$
```

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

Double Delay (1)



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

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

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

$$y(n) = w_2(n)$$

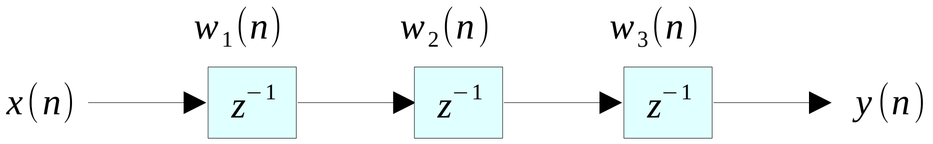
$$w_2(n+1) = w_1(n)$$
$$w_1(n+1) = x(n)$$

for each input sample x do:

```
    $y := w_2$
    $w_2 := w_1$
    $w_1 := x$
```

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

Triple Delay (1)



$$\begin{aligned} w_3(n) &= w_2(n-1) \\ w_2(n) &= w_1(n-1) \\ w_1(n) &= x(n-1) \end{aligned}$$

$$= w_1(n-2) = x(n-3)$$

$$\begin{aligned} w_3(n+1) &= w_2(n) \\ w_2(n+1) &= w_1(n) \\ w_1(n+1) &= x(n) \end{aligned}$$

$$y(n) = w_3(n)$$

$$\begin{aligned} w_3(n+1) &= w_2(n) \\ w_2(n+1) &= w_1(n) \\ w_1(n+1) &= x(n) \end{aligned}$$

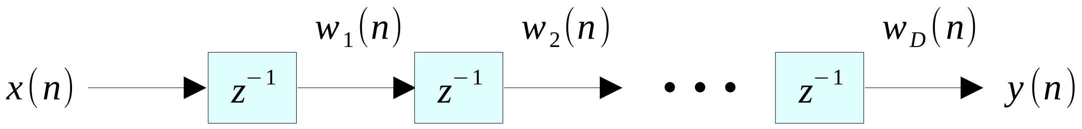
$$[x_0, x_1, x_2, x_3, \ldots] \rightarrow [0, 0, 0, x_0, \ldots]$$

$for\ each\ input\ sample\ x\ do:$

```
    $y := w_3$
    $w_3 := w_2$
    $w_2 := w_1$
    $w_1 := x$
```

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

D Unit Delay (1)



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

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

$for\ each\ input\ sample\ x\ do:$

```
    $y \ := \ w_D$
    $w_0 \ := \ x$
    $w_0 := x$
    $for\ i = D, D-1,\ldots,1\, do:$
        $w_i := w_{i-1}$
```

$for\ each\ input\ sample\ w_0\ do:$

```
    $for\ i = D, D-1,\ldots,1\, do:$
        $w_i := w_{i-1}$
```

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;
}
```

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

Direct Form

Considering the widely used
Edge triggered
D-type Flip Flops

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

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

References

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