# Tapped Delay

Copyright (c) 2024 - 2019 Young W. Lim.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

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

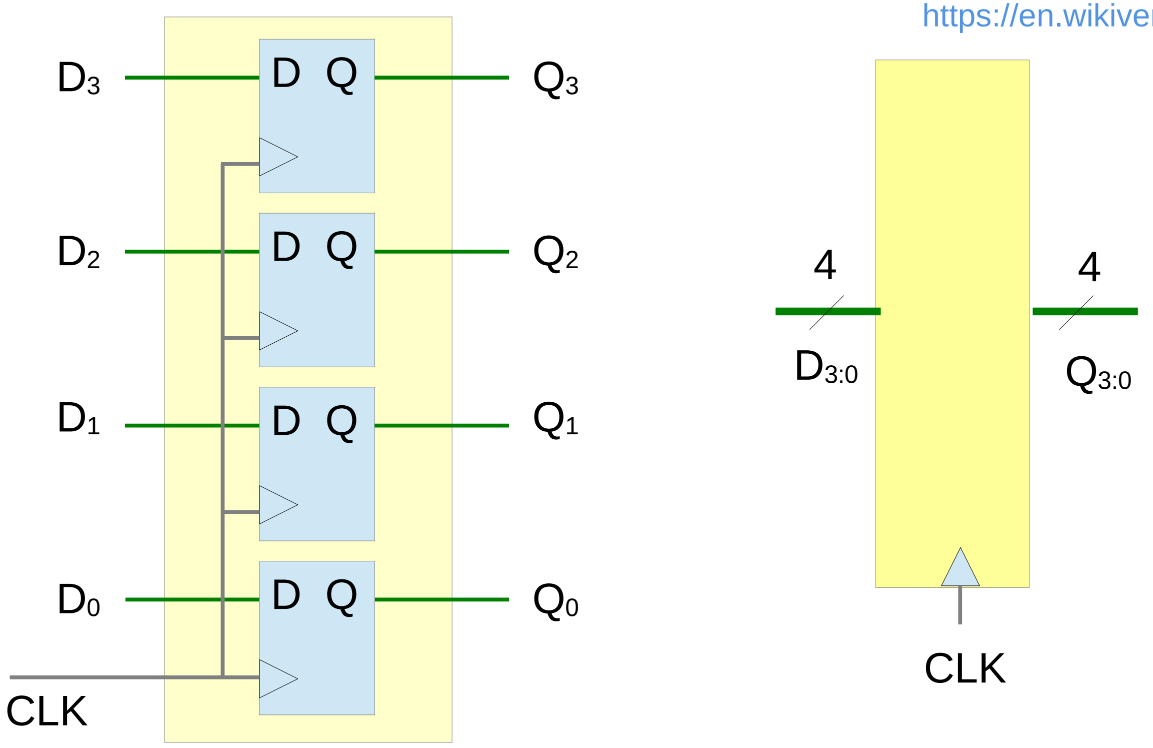
FIR Filter (A.pdf) 20191114

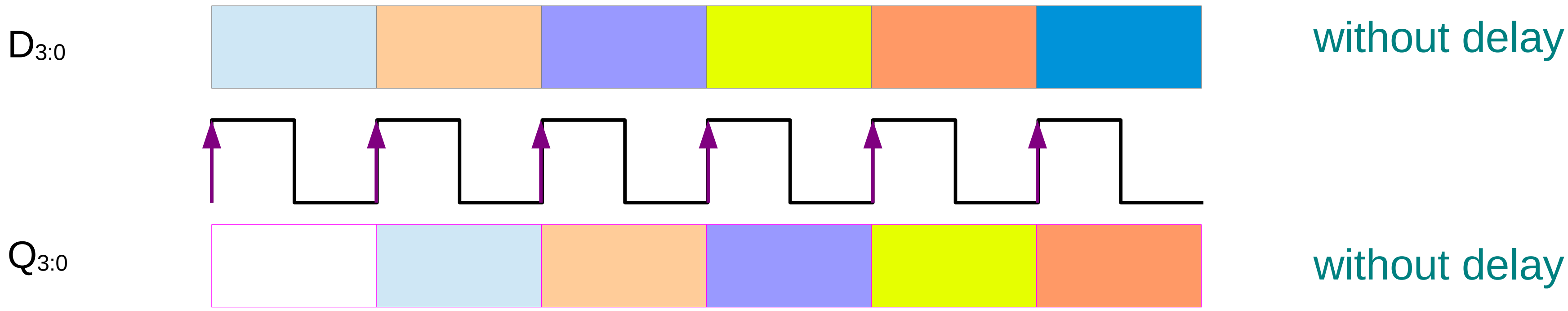
# D Flip Flop

Considering the widely used
Edge triggered
D-type Flip Flops

# Register

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

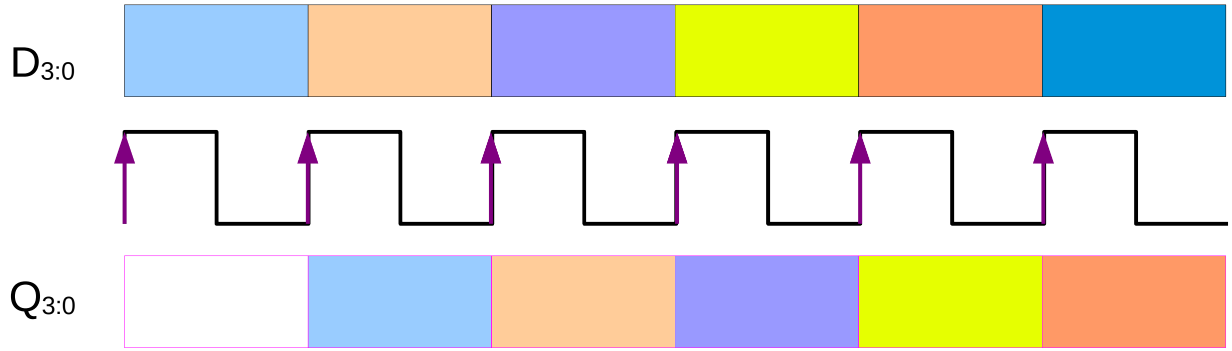


# Types of Timing Diagrams

## a timing diagram without delays

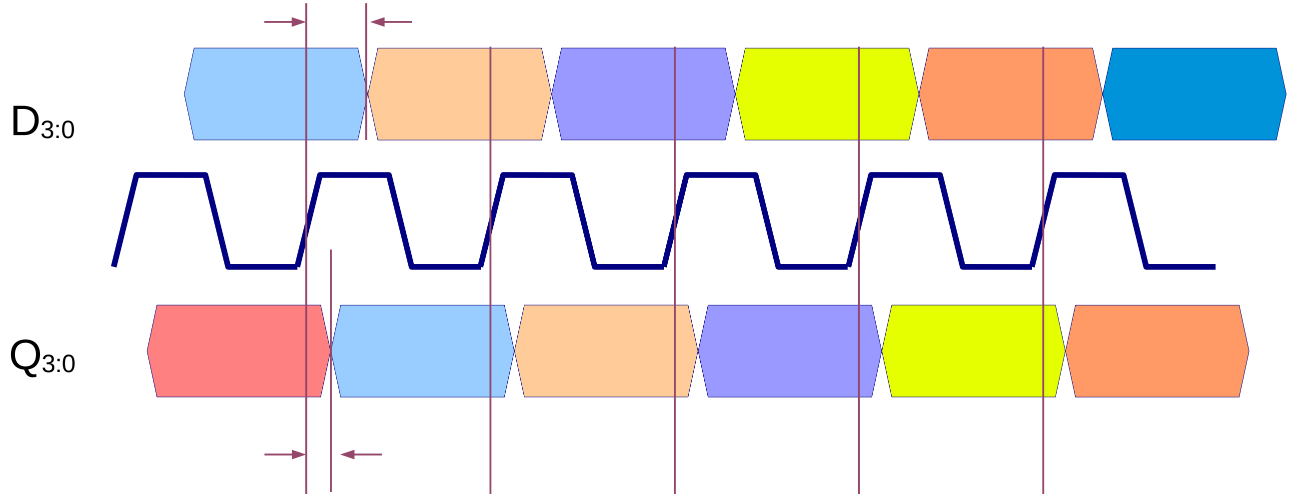
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design




no delay (ideal case)

no delay (ideal case)

## a timing diagram with delays

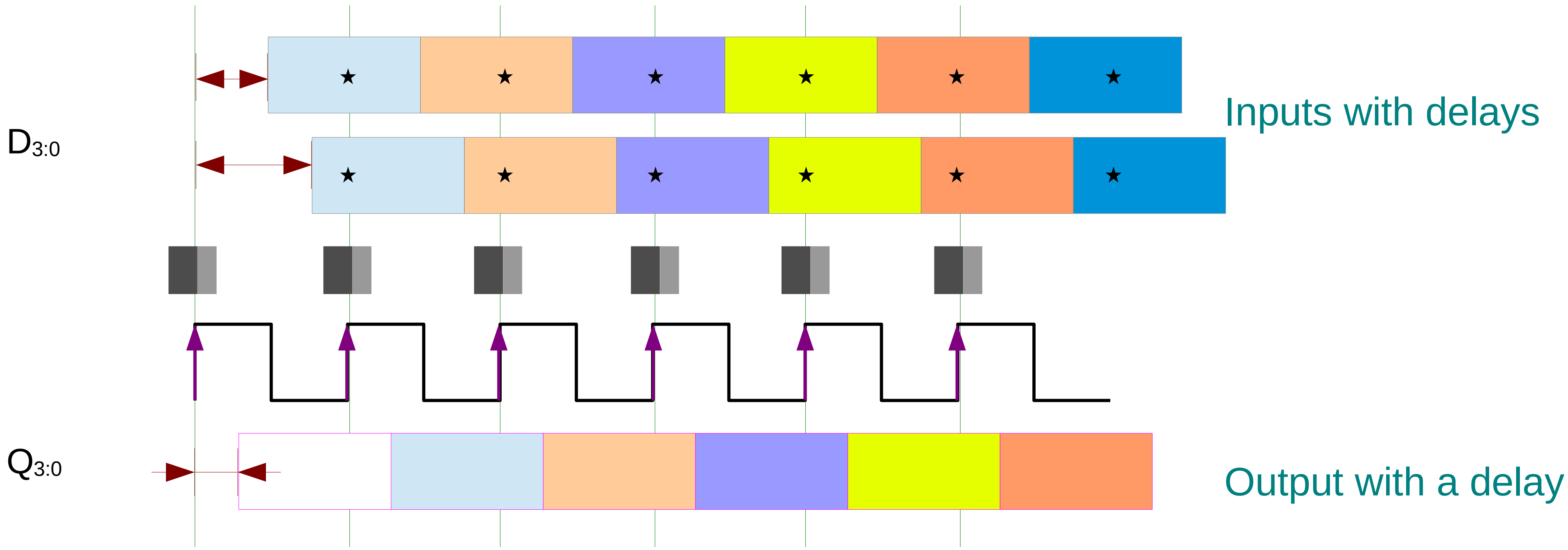


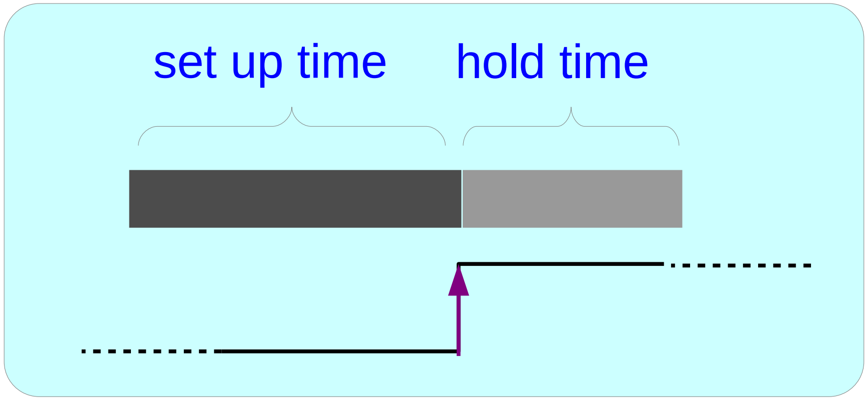

input delay

output delay Clk->Q

# Setup & Hold Time (1)

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

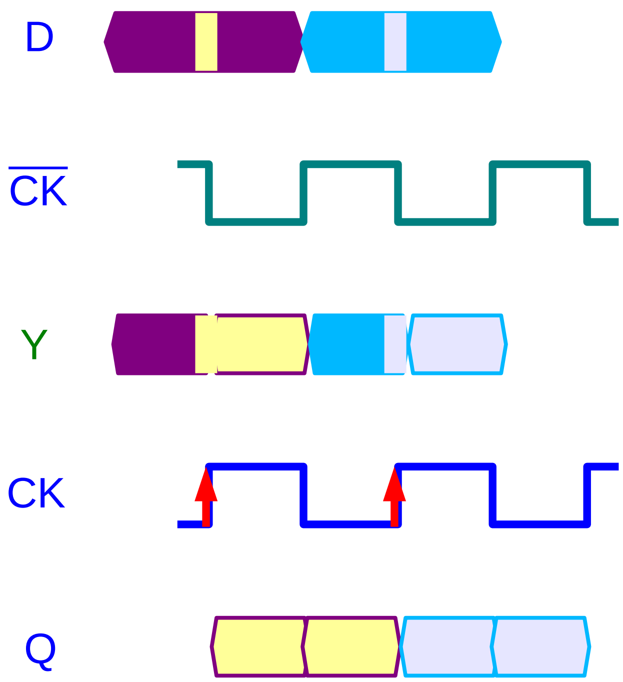


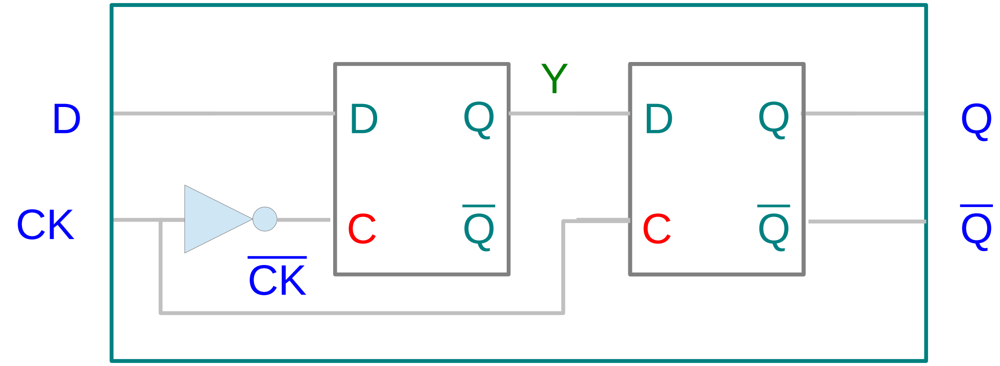
# Master-Slave D FlipFlop – Rising Edge

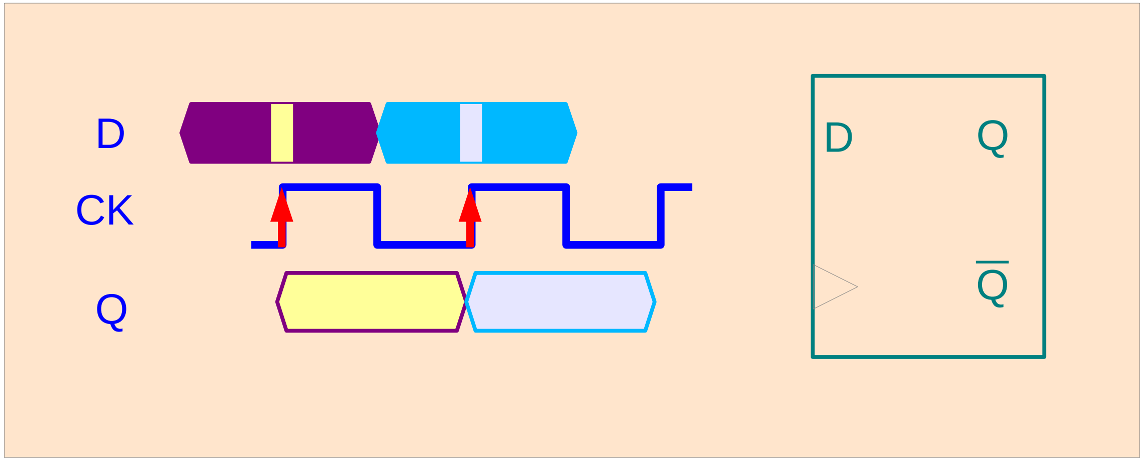
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

Master D Latch



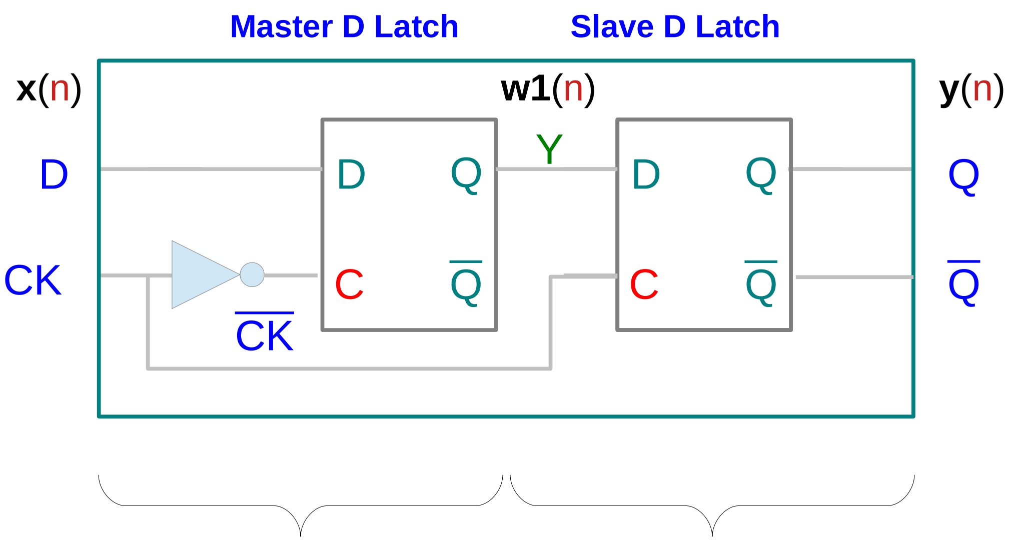

Slave D Latch



# Master-Slave D FlipFlop – Rising Edge

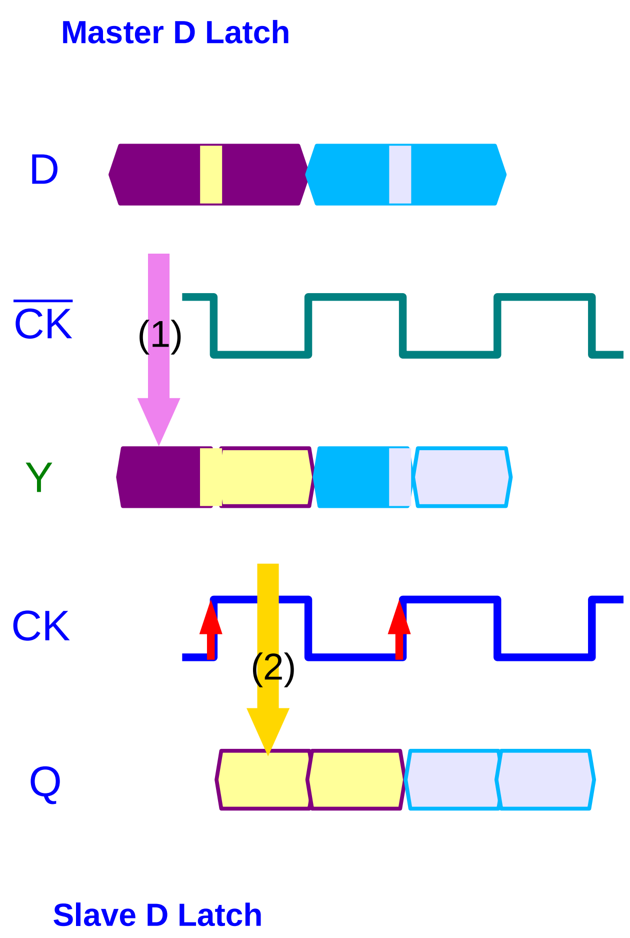
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design




(1) the current input
D gets stored
in the master latch

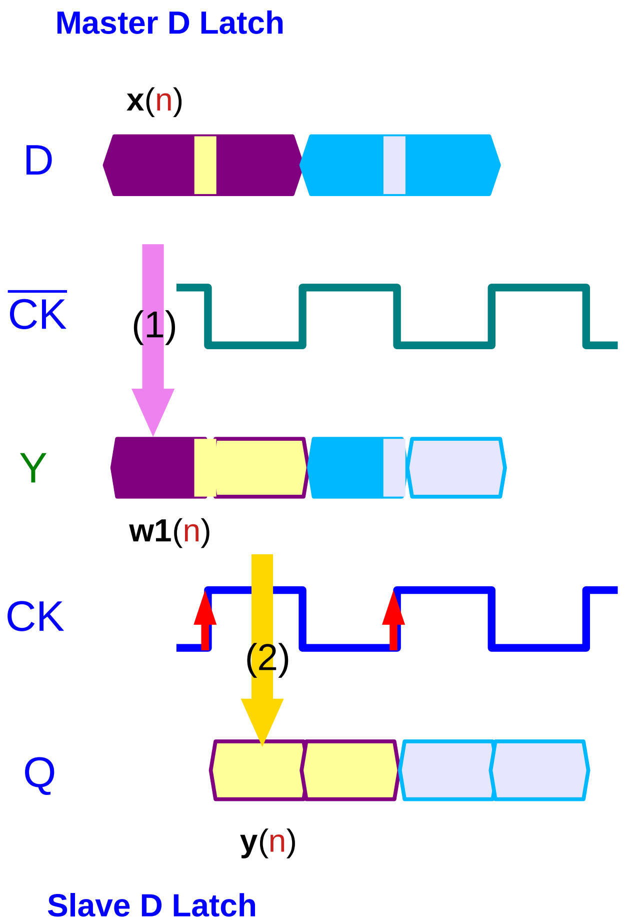
(2) the current content
Y is clocked out
to the output Q

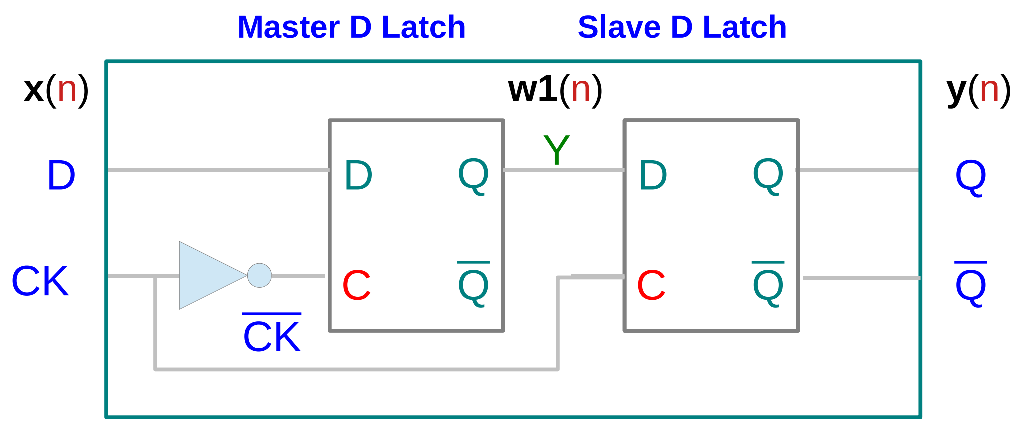
Using inverted clocks enable
(1) and (2) to be executed sequentially



# Master-Slave D FlipFlop – Rising Edge Sampling

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



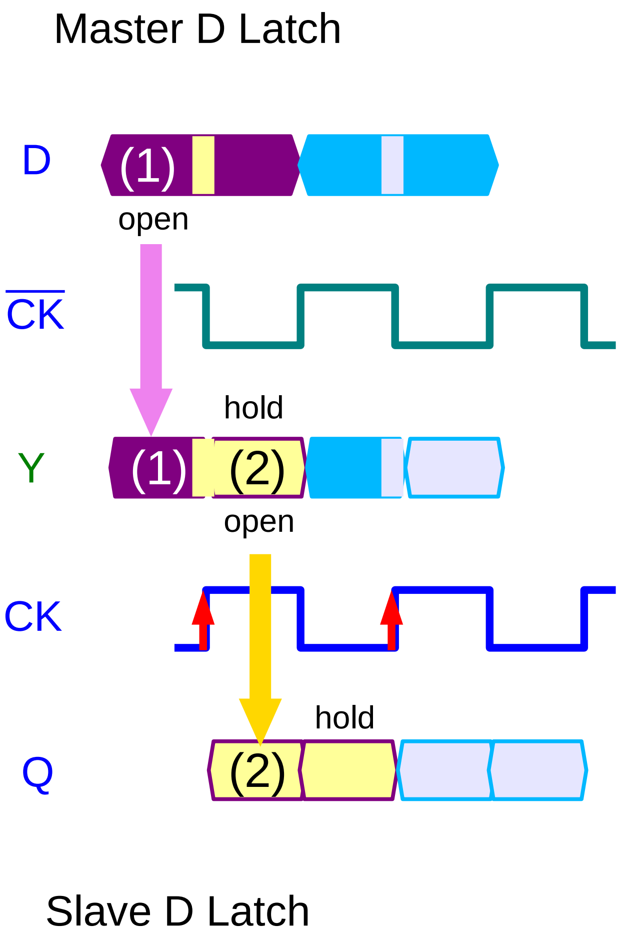



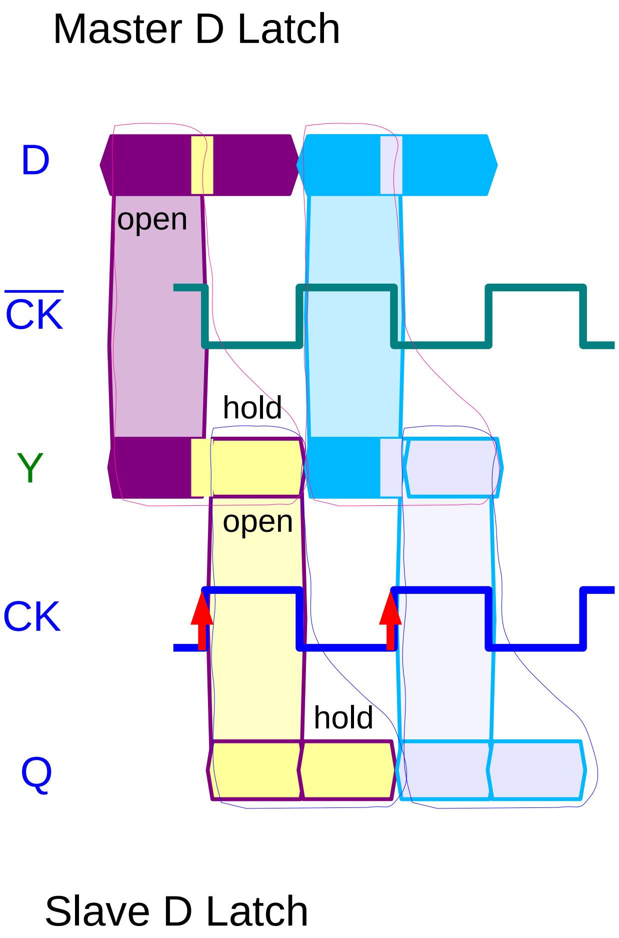

(1) the current <u>input</u>
**x**(n) gets stored
in the master latch

(2) the current <u>content</u>
**w1**(n) is clocked out
to the output **y**(n)

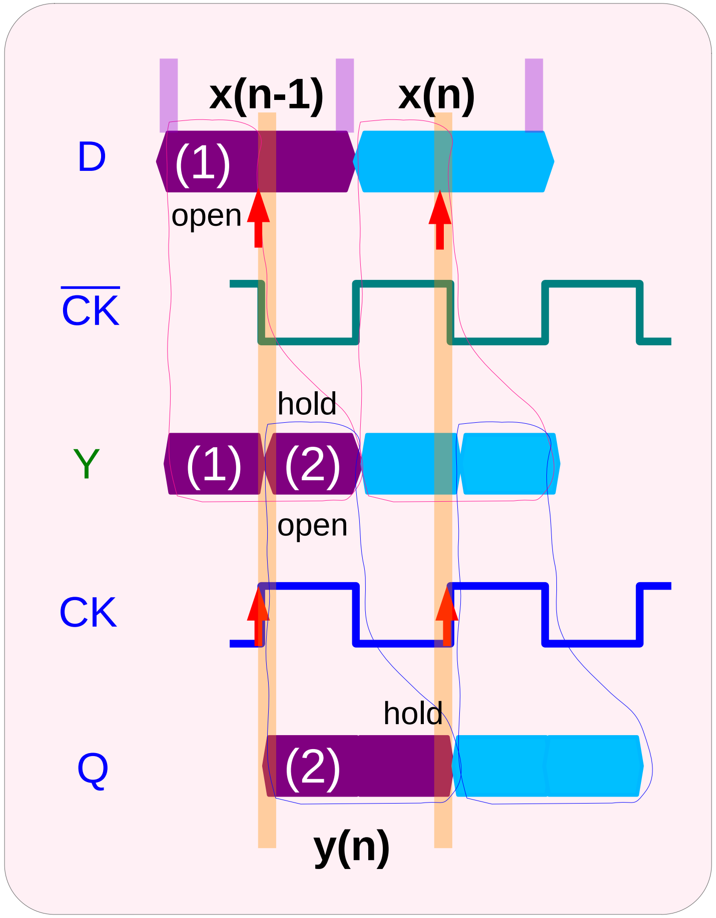
# Master-Slave D FlipFlop – open and hold

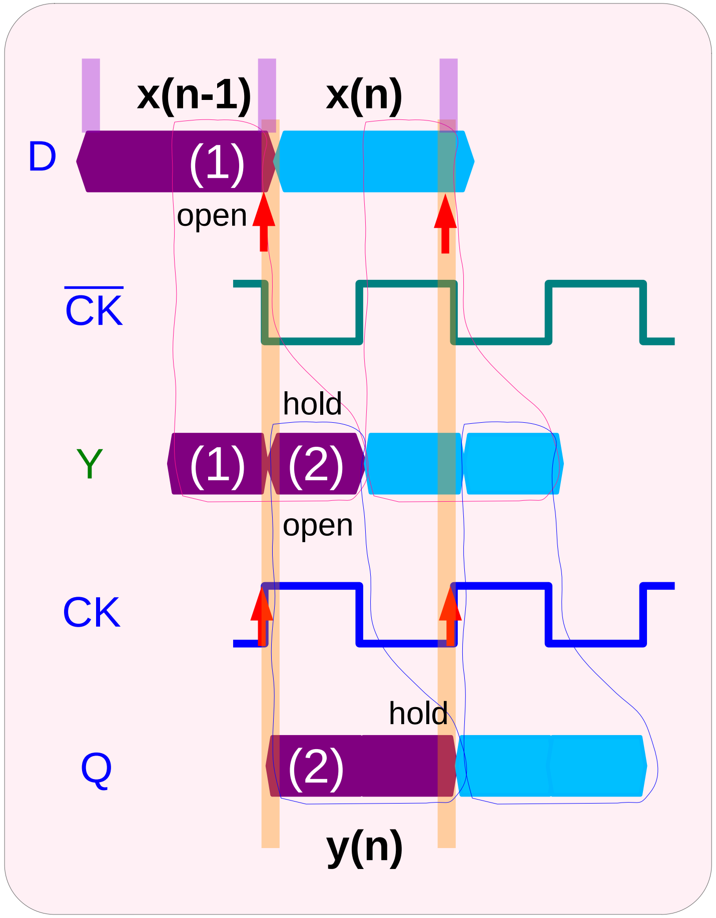
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design



# Master-Slave D FlipFlop – ideal timing

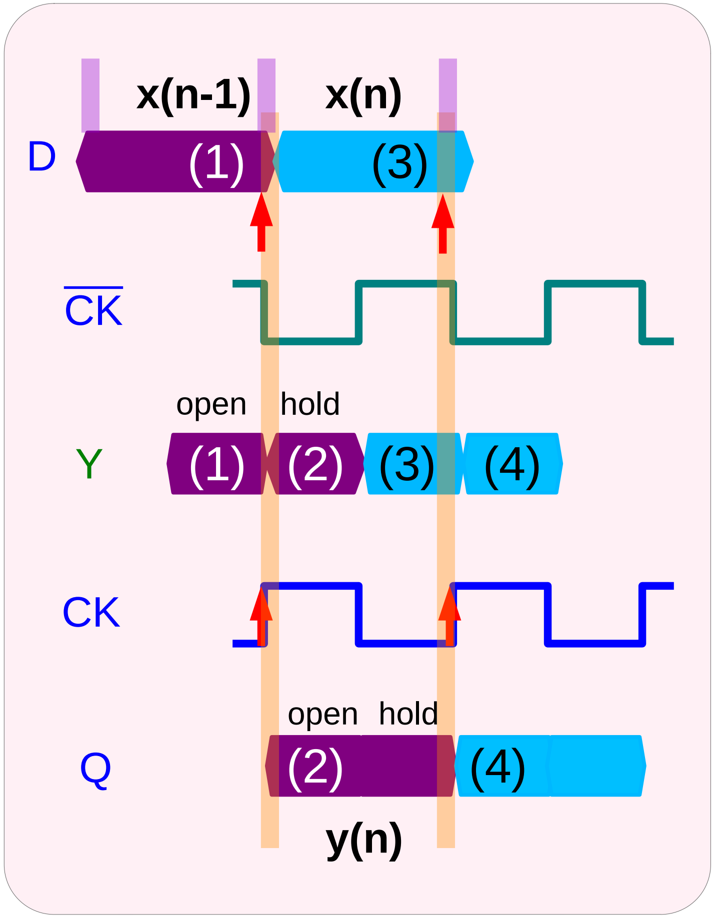



**Typical Timing**

**Ideal Timing**

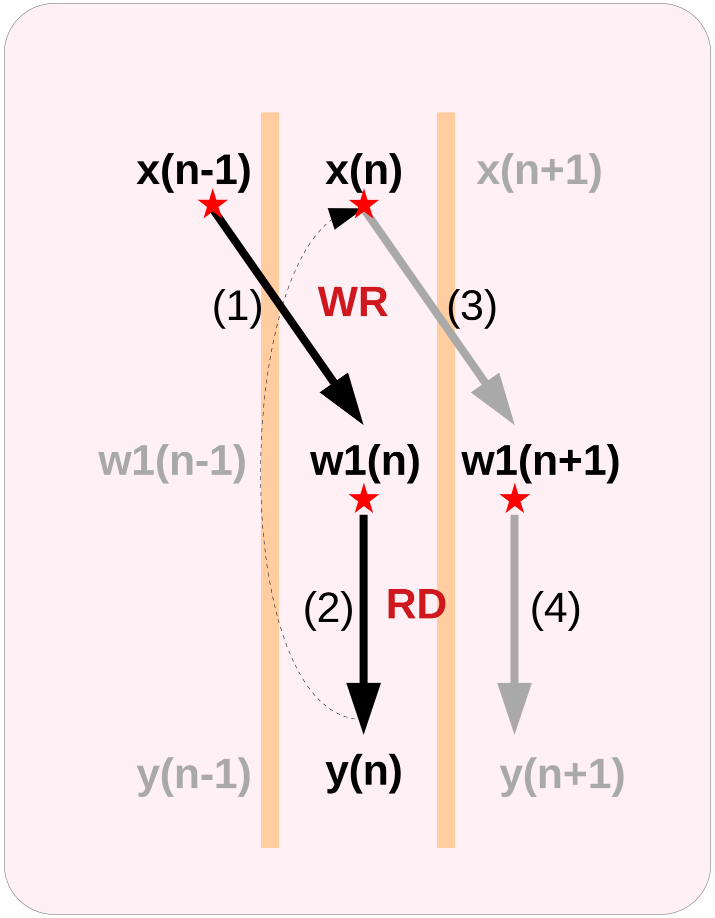
# Master-Slave D FlipFlop – DSP C model




*Input*

*Internal State*

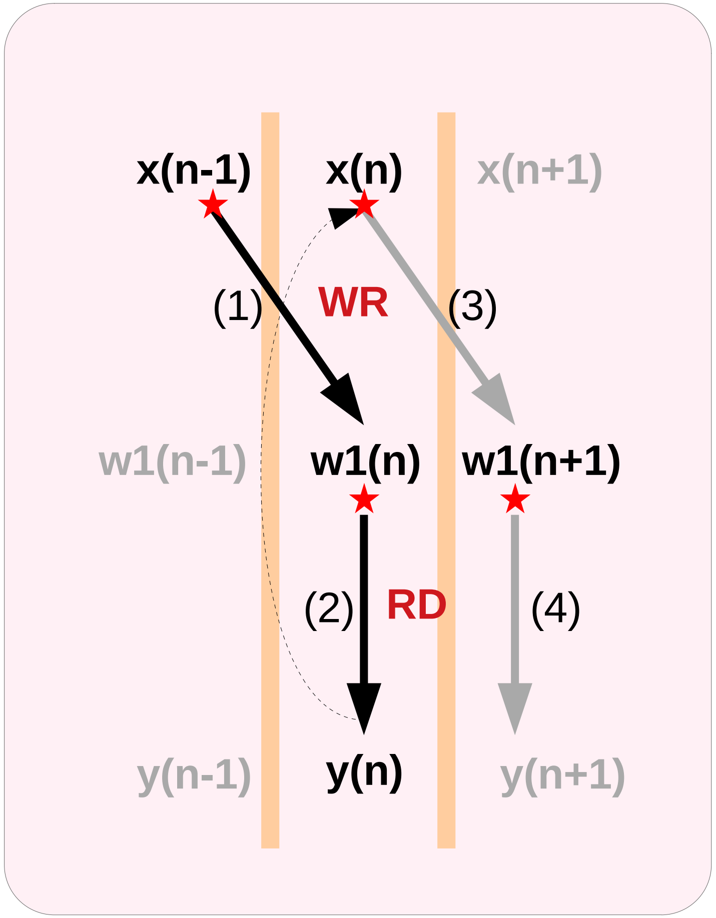
*Output*




**Hardware model**

**DSP C model**

# Master-Slave D FlipFlop – DSP C model




***Input***

***Internal State***

***Output***

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

WR **w1**(n)

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

RD **w1**(n)

No **RAW** (read after write) hazard

but we must simulate
the parallel hardware
by a sequential code

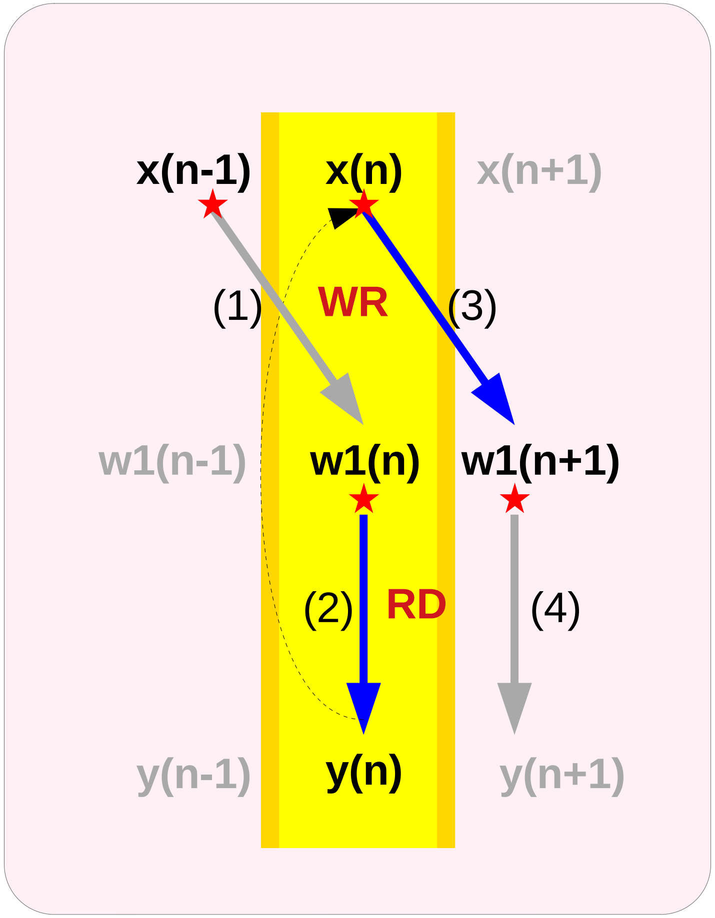
at time n,
use only **x**(n) and **y**(n)

**DSP C model**

# Master-Slave D FlipFlop – DSP C model

at the time **n**

**RAW (read after write)**




***Input***

***Internal State***

***Output***

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

**RD w1**(n)

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

**WR w1**(n)

No **RAW** (read after write) hazard

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

**w1**(n) = **x**(n)

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

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

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

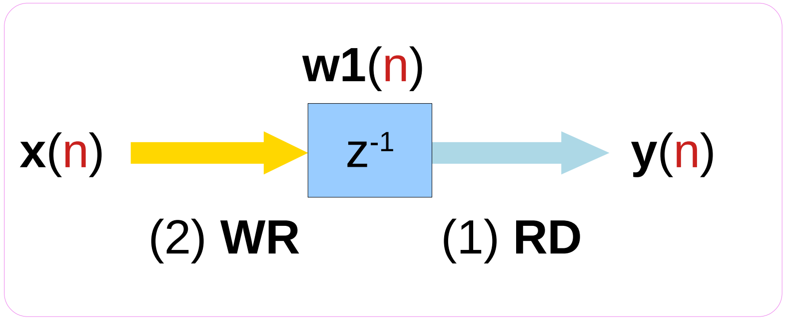
**w1**(n+2) = **w1**(n+1)

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

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

**DSP C model**

# Simultaneous RD and WR actions



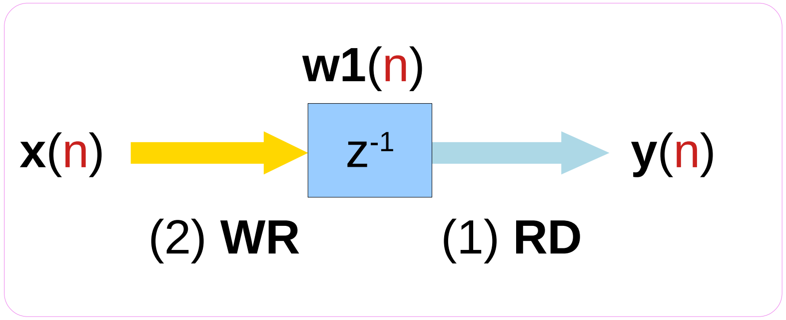

| | | |
|---|---|---|
| current content | **w1**(n) | = **x**(n-1) |
| next content | **w1**(n+1) | = **x**(n) |

| at time n | | at time n+1 | |
|---|---|---|---|
| **y**(n) = **w1**(n) | → | **y**(n+1) = **w1**(n+1) | *read internal state* |
| **w1**(n+1) = **x**(n) | | **w1**(n+2) = **x**(n+1) | *update internal state* |

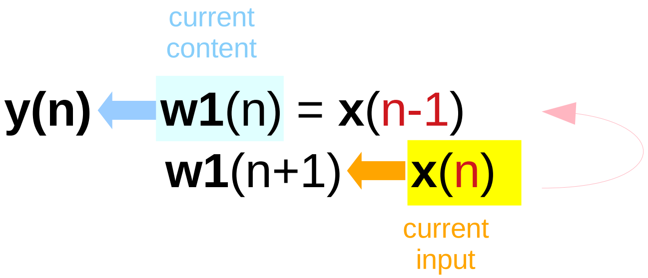
at time n,

1) the content of the register **w1**(n) becomes the output **y**(n) — **RD** access of **w1**(n) = **x**(n-1)
2) the input **x**(n) is saved and becomes the new content **w1**(n+1) — **WR** access of **w1**(n+1) = **x**(n)

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



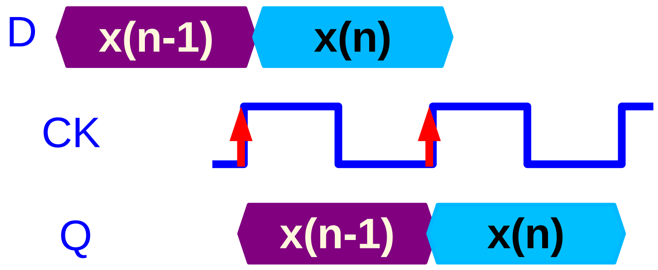

| | | |
|---|---|---|
| current content | **w1**(n) | = **x**(n-1) |
| next content | **w1**(n+1) | = **x**(n) |



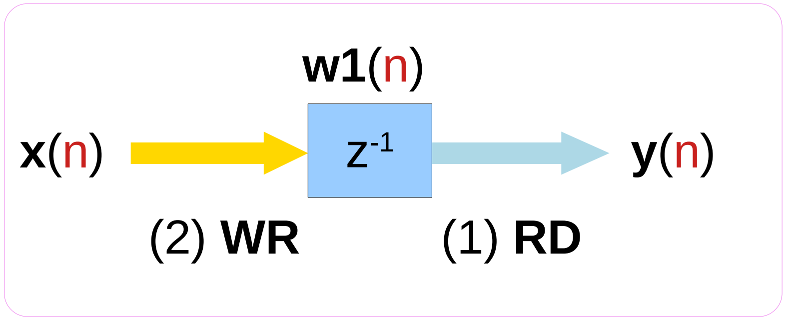

a register holding the previous input sample **x**(n-1)

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

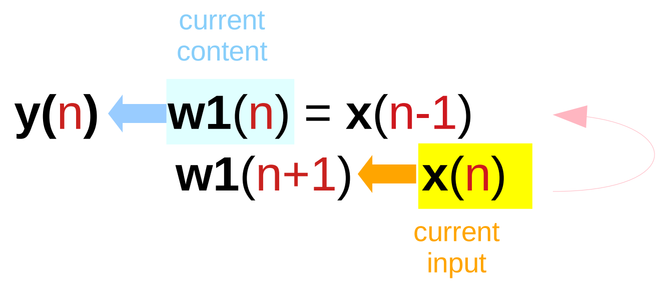
It will be held for one sampling instant and become the output at the next time n+1



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

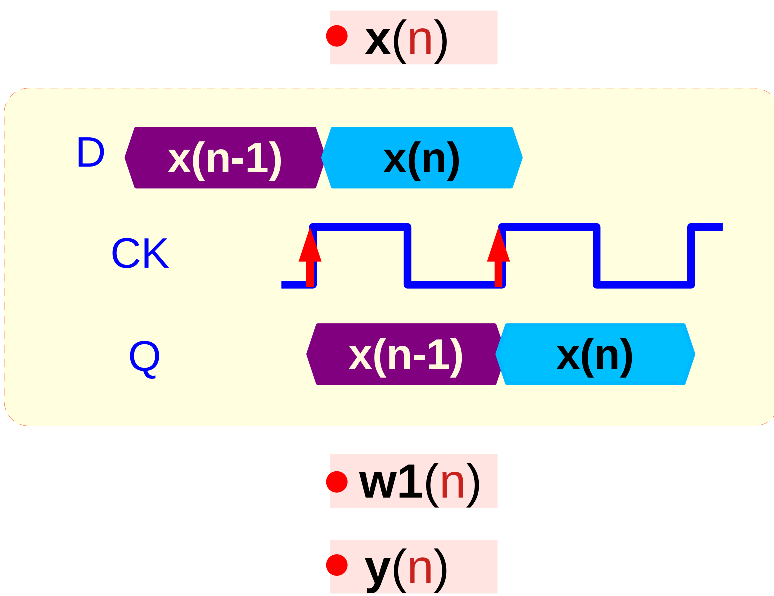



| | | |
|---|---|---|
| current content | **w1**(n) | = **x**(n-1) |
| next content | **w1**(n+1) | = **x**(n) |

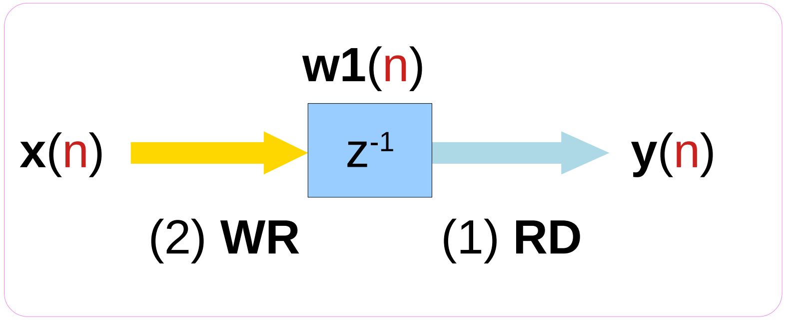



*simulate a clocked hardware*
*ignoring delay constraints in hardware*

***zero-delay simulation***



# Delay element modeling

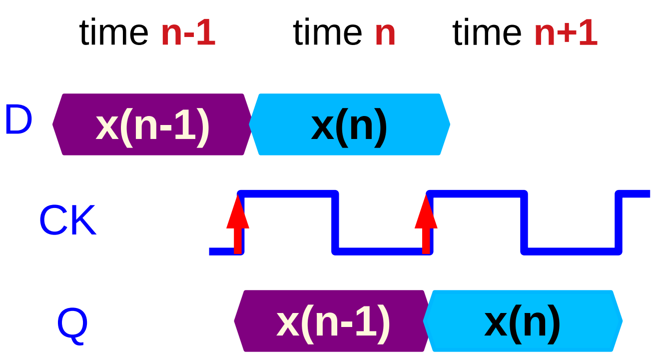



| | |
|---|---|
| w1(n) → y(n) | y(n) = w1(n)<br>(1) **RD** old w1 |
| x(n) → w1(n+1) | w1(n+1) = x(n)<br>(2) **WR** new w1 |

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*

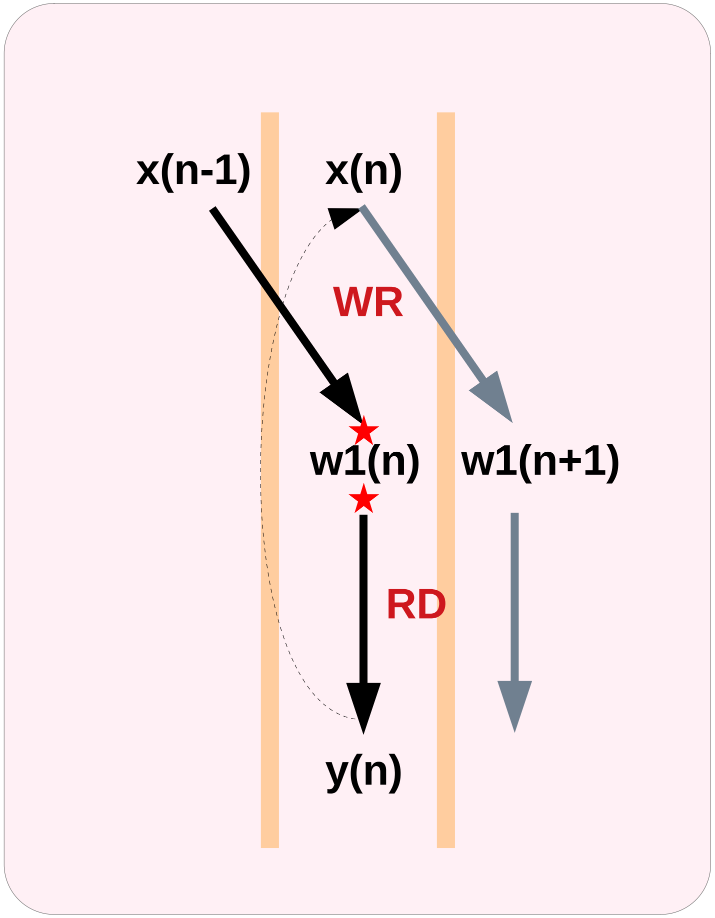
***zero-delay simulation***

# 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 |

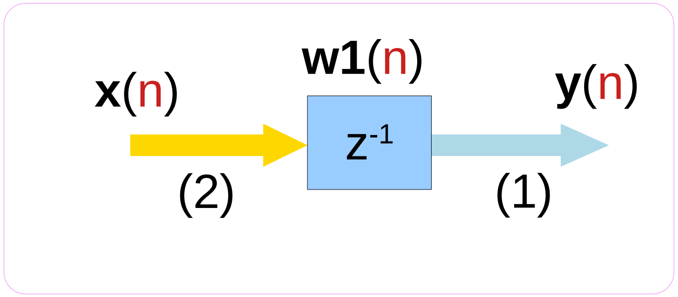
| | | |
|---|---|---|
| **w1**(n+1) = **x**(n) | (2) **WR** | new w1 |
| **y**(n) = **w1**(n) | (1) **RD** | old w1 |
| **w1**(n+2) = **x**(n+1) | (2) **WR** | new w1 |
| **y**(n+1) = **w1**(n+1) | (1) **RD** | old w1 |

**WAR (Write after Read) Violation**



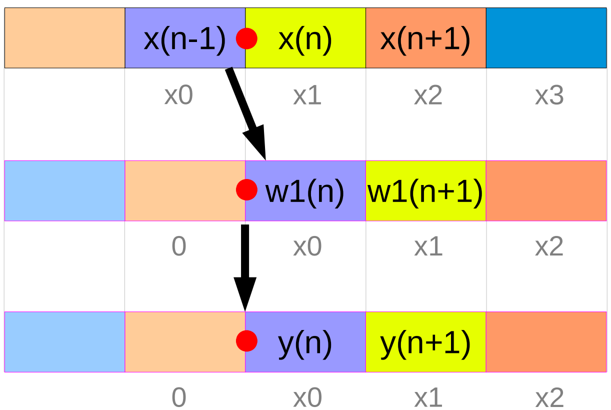

**DSP C model**

# **Single** Delay



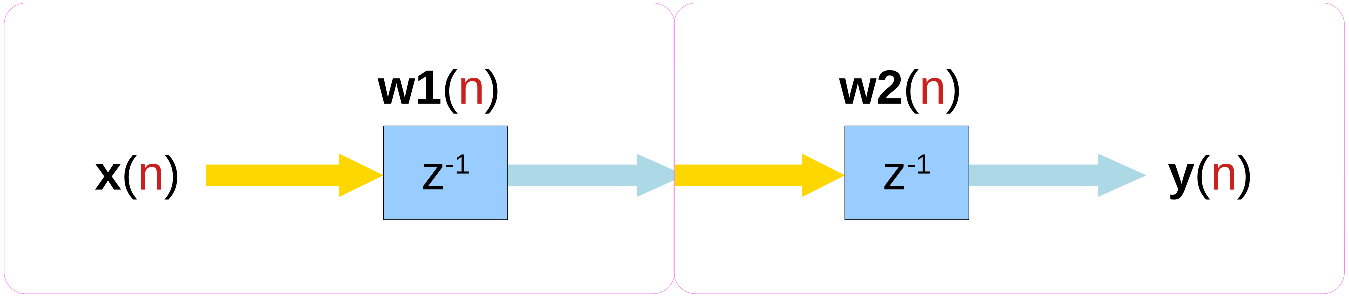

y(n) = w1(n) (1) **RD**

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

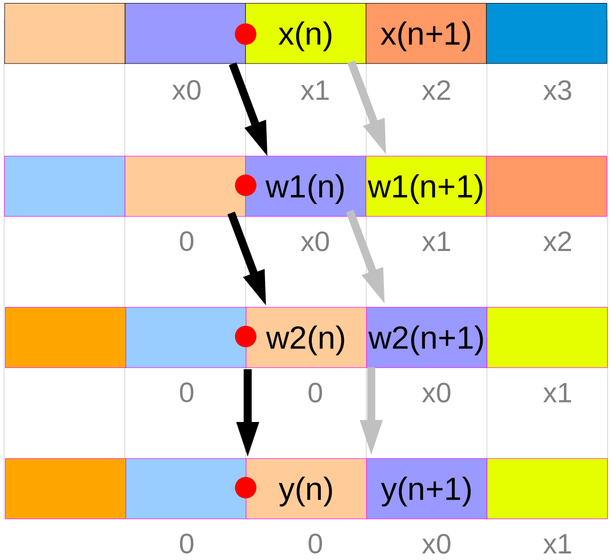



| n | x(n) | w1(n) | y(n) |
|---|---|---|---|
| 0 | x0 | 0 | 0 |
| 1 | x1 | x0 | x0 |
| 2 | x2 | x1 | x1 |
| 3 | x3 | x2 | x2 |
| 4 | x4 | x3 | x3 |

# Double Delay

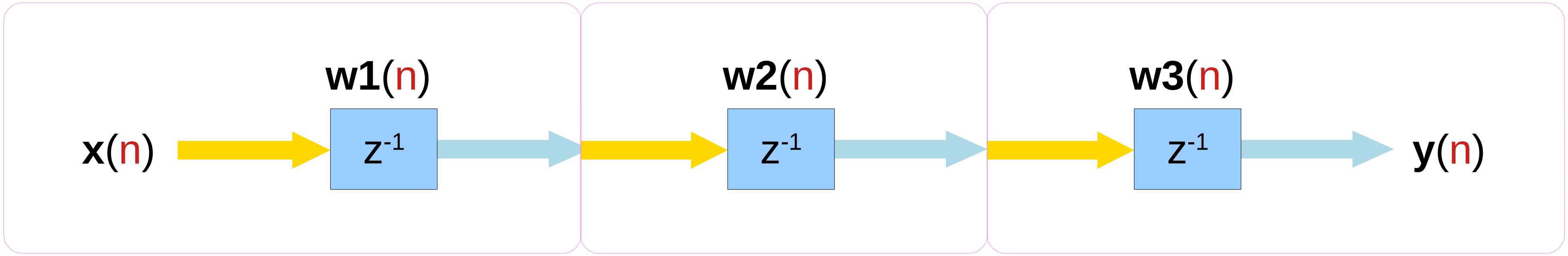



$y(n) = w2(n)$
$w2(n+1) = w1(n)$
$w1(n+1) = x(n)$




| 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 |

# Triple Delay

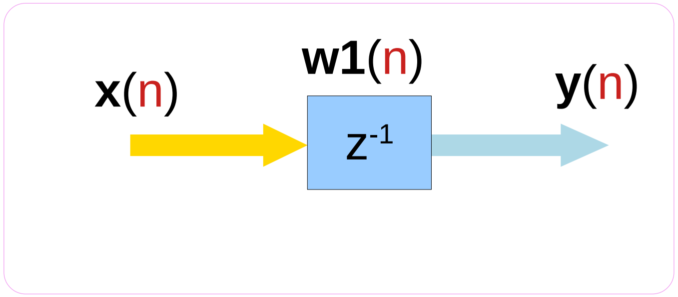




| n | x(n) | w1(n) | w2(n) | w3(n) | y(n) |
|---|---|---|---|---|---|
| 0 | x0 | 0 | 0 | 0 | 0 |
| 1 | x1 | x0 | 0 | 0 | 0 |
| 2 | x2 | x1 | x0 | 0 | 0 |
| 3 | x3 | x2 | x1 | x0 | x0 |
| 4 | x4 | x3 | x2 | x1 | x1 |

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

# **Single** Delay – IO Equations




**single delay**

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

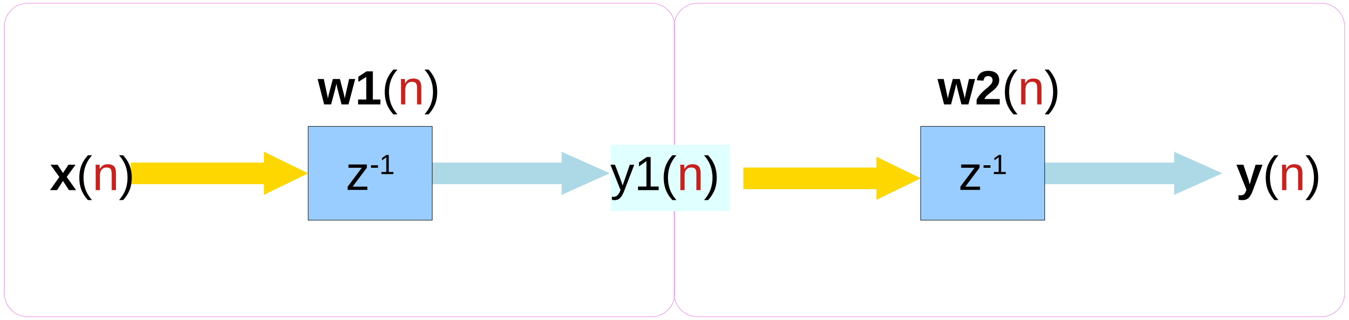
**double delay**

| | |
|---|---|
| **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 |

# **Double** Delay – IO Equations




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

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

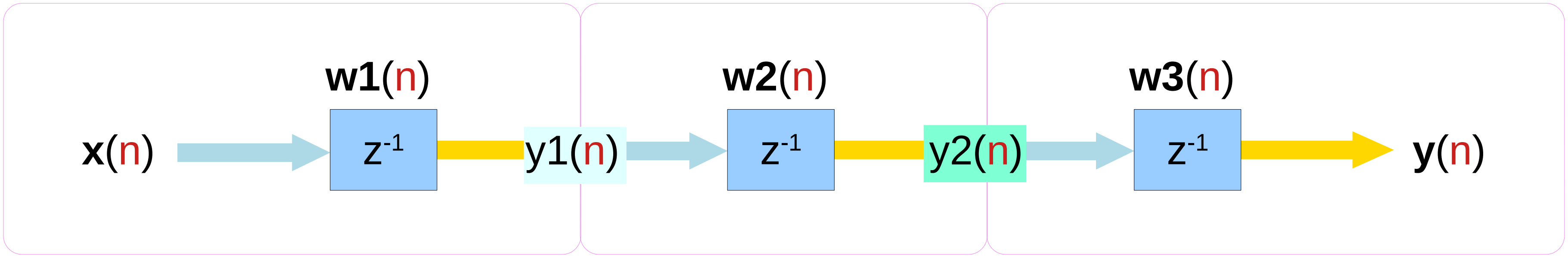
$y1(n) = \mathbf{w1}(n)$
$\mathbf{w2}(n+1) = y1(n)$

---

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

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

# Triple Delay – IO Equations




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

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

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

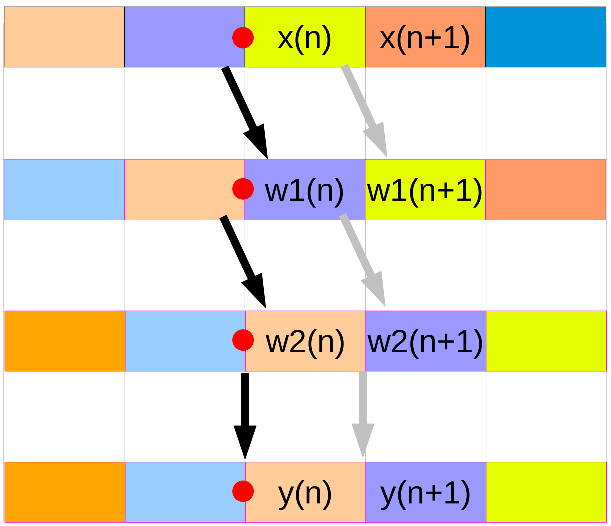
$y1(n) = \mathbf{w1}(n)$
$\mathbf{w2}(n+1) = y1(n)$
$\mathbf{w2}(n+1) = \mathbf{w1}(n)$

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

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

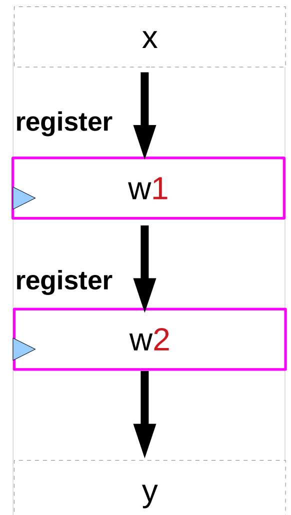
# Delay C Model

### Timing Chart



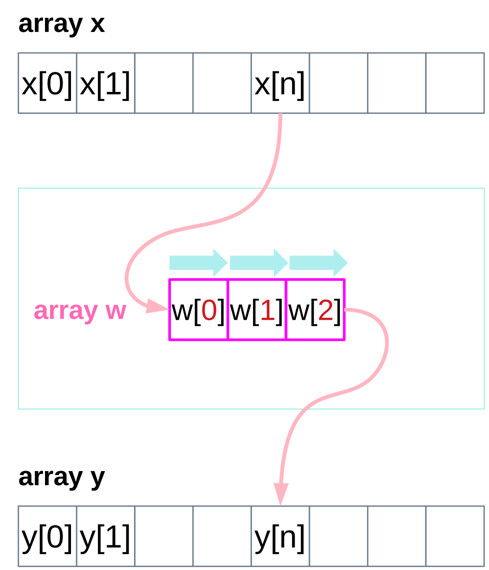

$y(n) = w2(n)$
$w2(n+1) = w1(n)$
$w1(n+1) = x(n)$

### Register Transfer




$y = w2$
$w2 = w1$
$w1 = x$

### DSP C Model for simulation



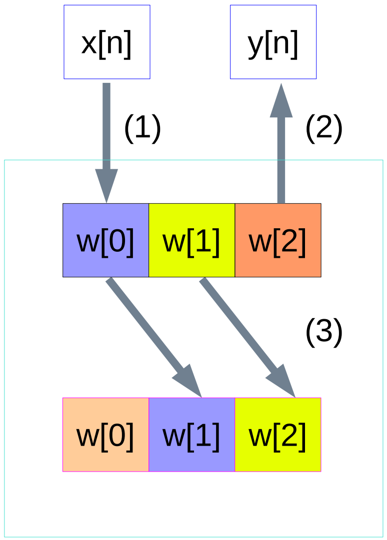

$y[n] = w[2]$
$w[0] = x[n]$
$w[2] = w[1]$
$w[1] = w[0]$

# IO Equations for the Triple Delay (1)

y(n) = w2(n)
w0(n) = x(n)
w2(n+1) = w1(n)
w1(n+1) = w2(n)

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



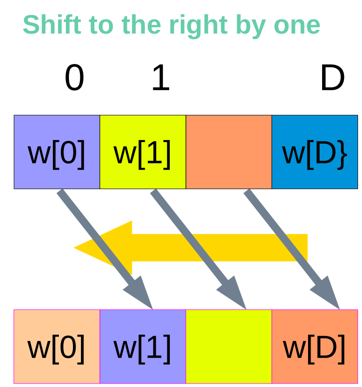
# delay.c

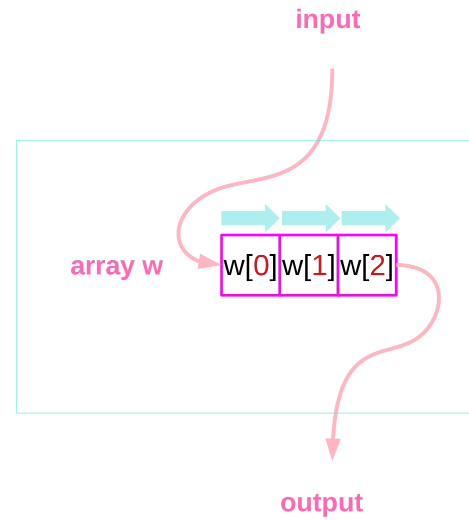
```
/* delay.c - delay by D time samples */
/* w[0] = input, w[D] = output */

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

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

    // reverse-order updating
}
```



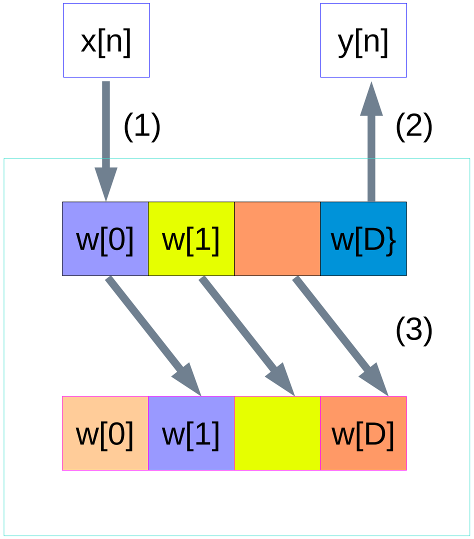

order of execution

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



# Delay Functions

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

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

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

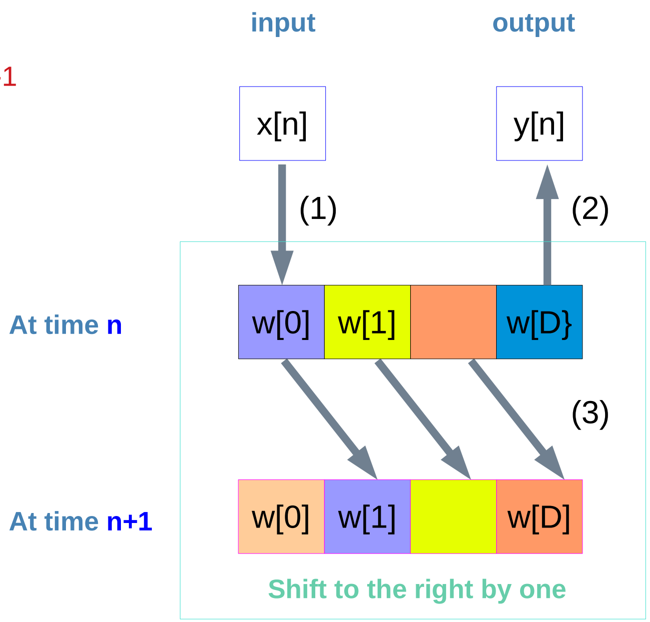
$y(n) = w_D(n)$
$w_0(n) = x(n)$
$w_i(n+1) = w_{i-1}(n),$
$i = D, D-1, \ldots, 2, 1$

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 $w_i$



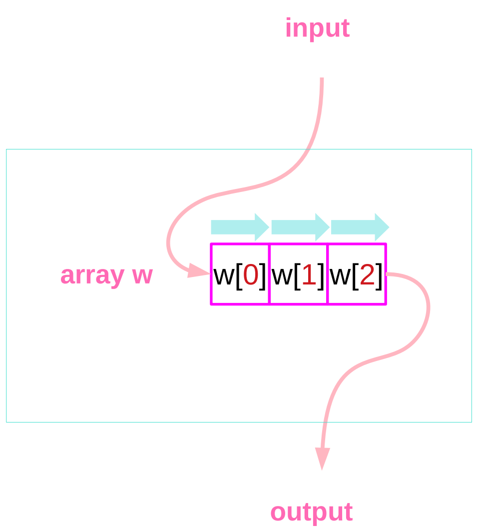
# Holding a delayed input sequence

$w_0(n) = x(n)$

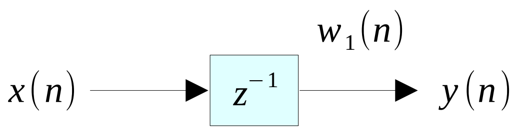
$w_1(n) = x(n-1) = w_0(n-1)$

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

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



# Single Delay (1)




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

$$\text{for each input sample } x \text{ do:}$$

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

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

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

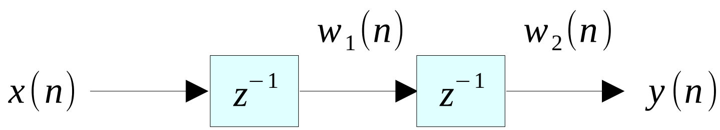
$$y(n) = w_1(n)$$
$$w_1(n+1) = x(n)$$

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

$$w_1(0) = 0$$

| $n$ | $x(n)$ | $w_1(n)$ | $y(n)$ |
|---|---|---|---|
| 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)




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

*for each input sample x do:*

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

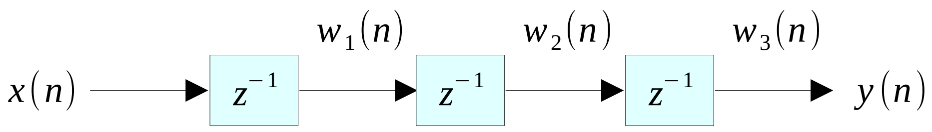
$$w_2(n) = w_1(n-1) = x((n-1)-1) = x(n-2)$$
$$w_1(n) = x(n-1)$$

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

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

| $n$ | $x(n)$ | $w_1(n)$ | $w_2(n)$ | $y(n)$ |
|---|---|---|---|---|
| 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)




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

*for each input sample x do:*

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

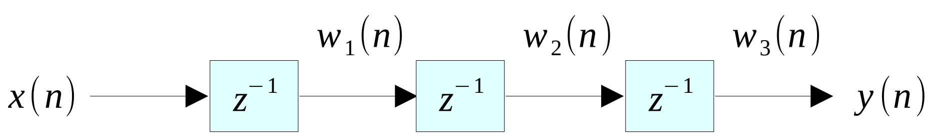
$$w_3(n) = w_2(n-1) \qquad = w_1(n-2) = x(n-3)$$
$$w_2(n) = w_1(n-1)$$
$$w_1(n) = x(n-1)$$

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

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

| $n$ | $x(n)$ | $w_1(n)$ | $w_2(n)$ | $y(n)$ |
|---|---|---|---|---|
| 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)




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

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

*for each input sample x do:*

$y := w_D$

$w_0 := x$

$w_0 := x$

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

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

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

| $n$ | $x(n)$ | $w_1(n)$ | $w_2(n)$ | $y(n)$ |
|---|---|---|---|---|
| 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 | $x_4$ | $x_3$ | $x_2$ | $x_2$ |

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

*for each input sample $w_0$ do:*

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

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

# 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