

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

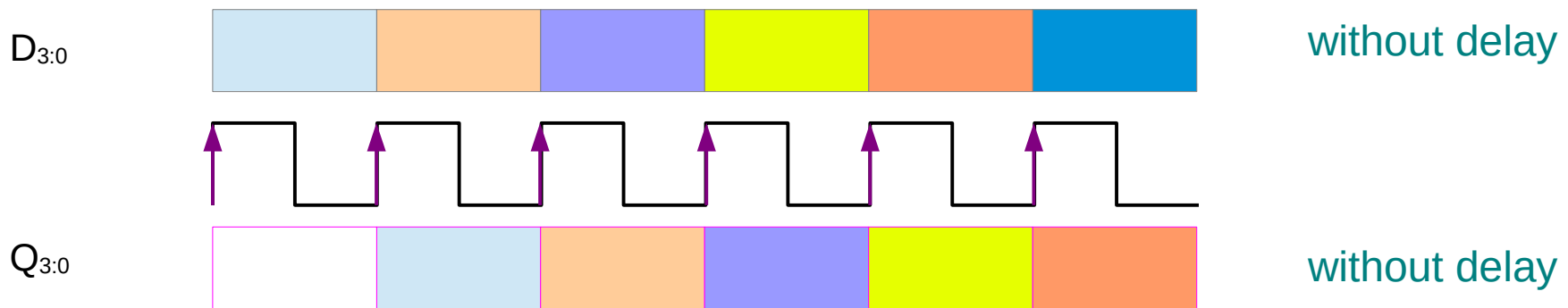
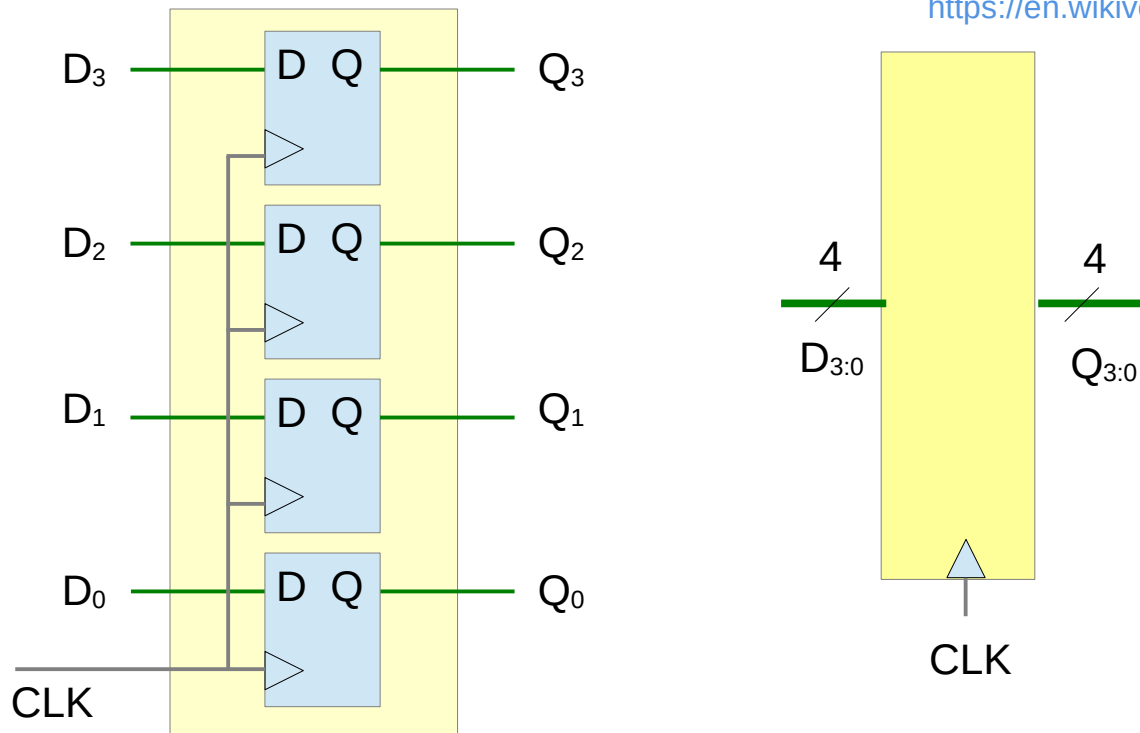
FIR Filter (A.pdf) 20191114

D Flip Flop

Considering the widely used
Edge triggered
D-type Flip Flops

Register

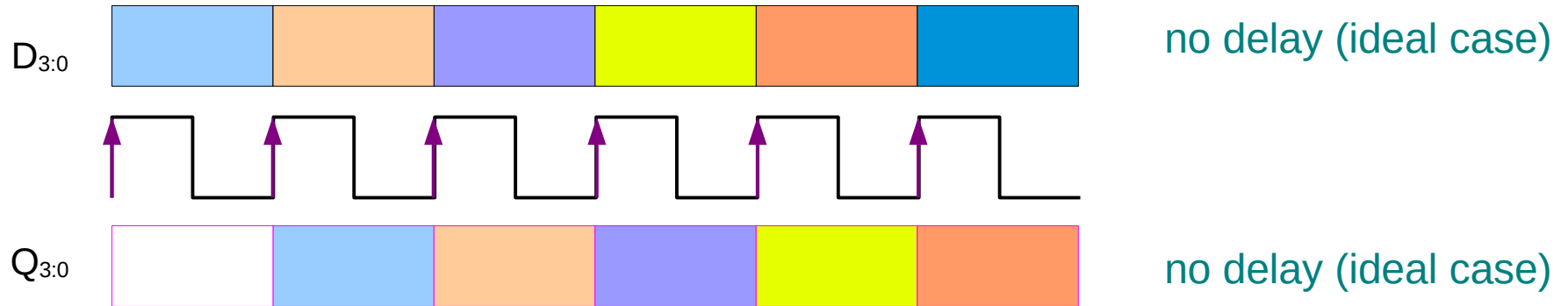
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design



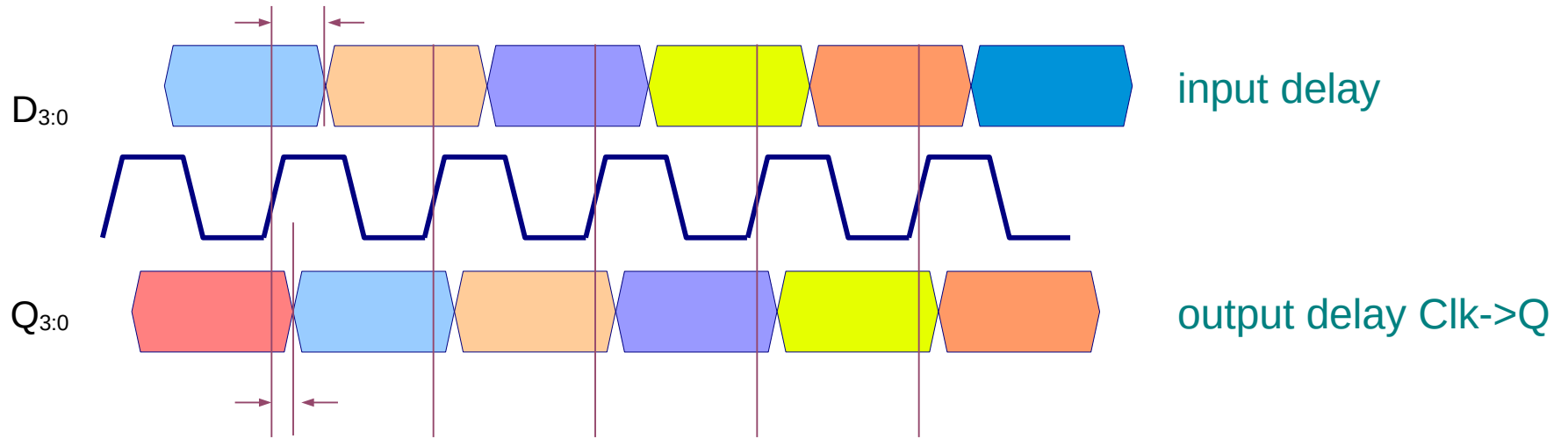
Types of Timing Diagrams

a timing diagram without delays

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

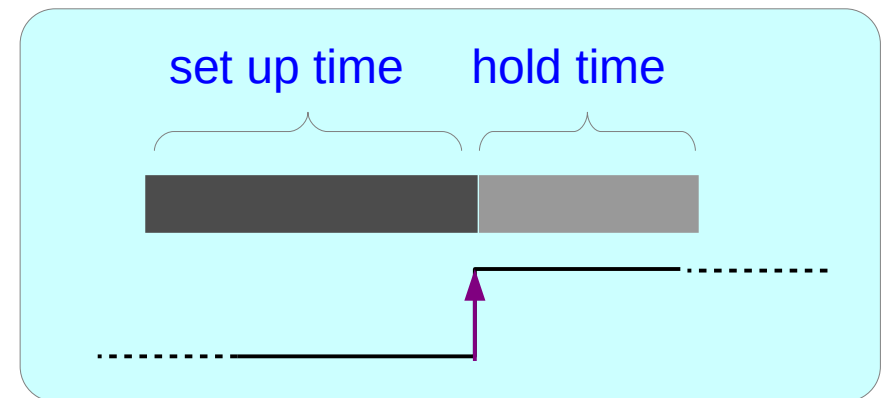
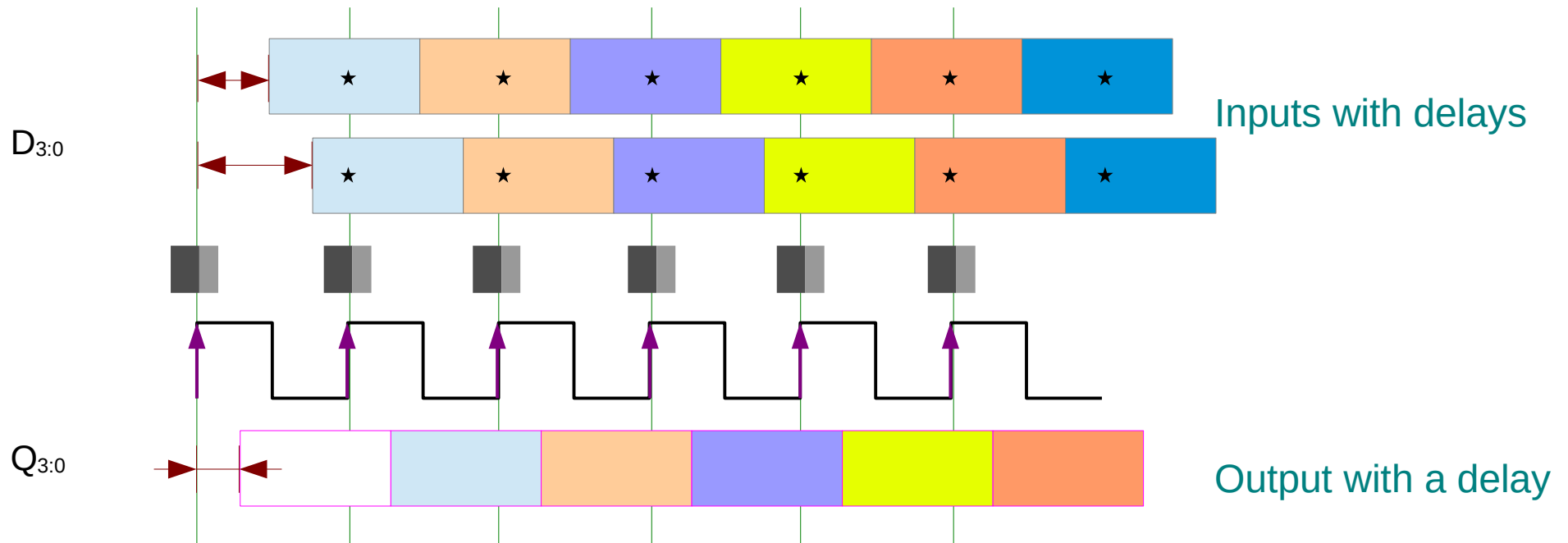


a timing diagram with delays



Setup & Hold Time (1)

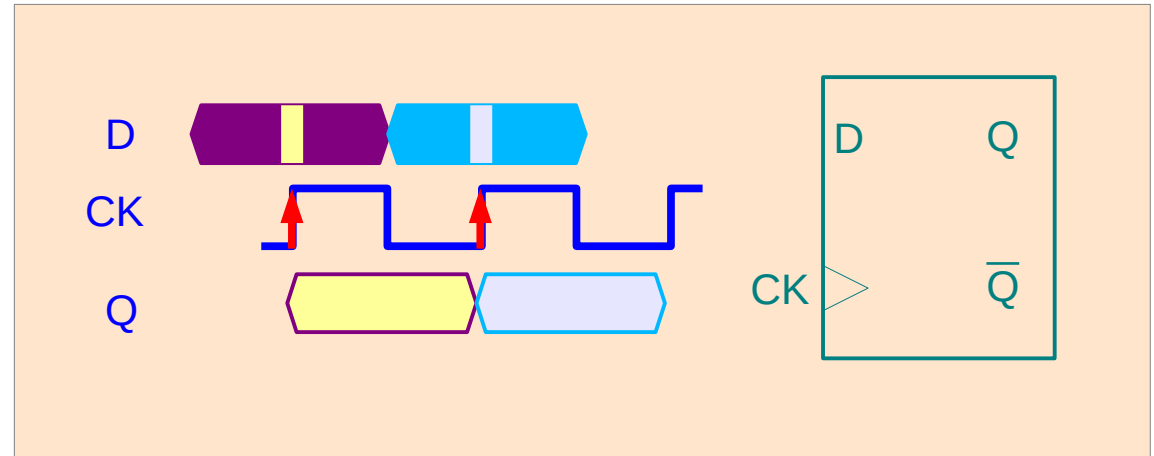
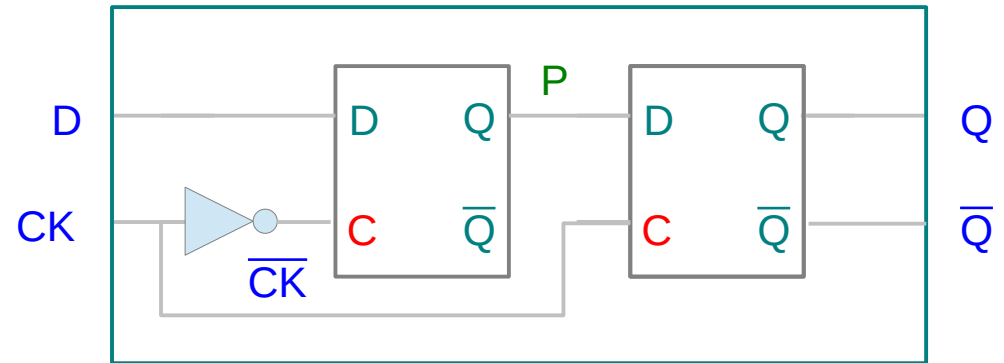
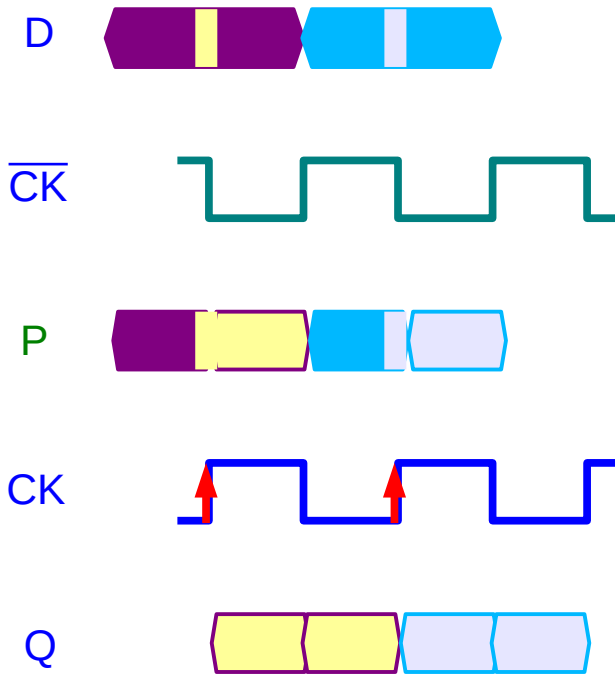
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design



Master-Slave D FlipFlop – Rising Edge

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

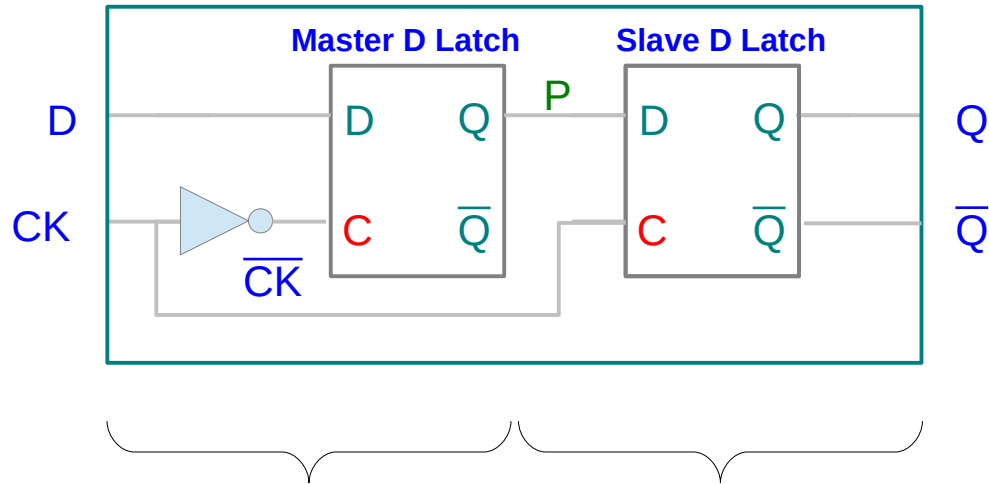
Master D Latch



Slave D Latch

Master-Slave D FlipFlop – Rising Edge

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

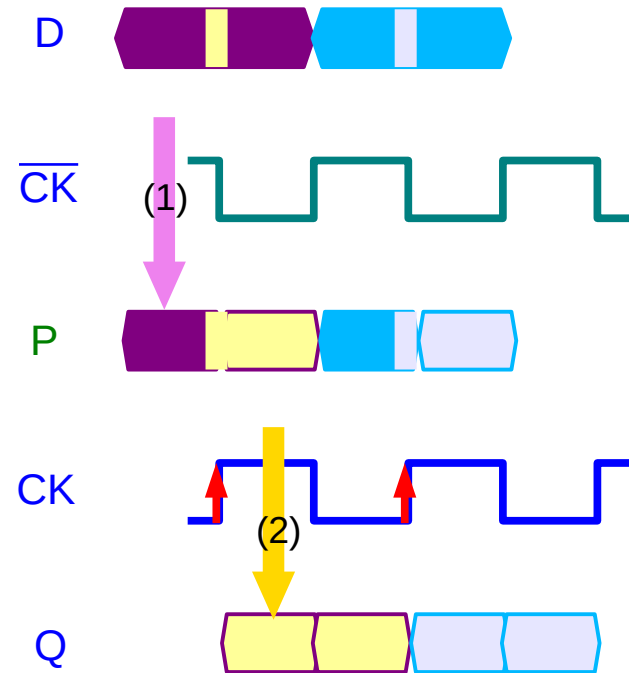


(1) the current input D gets stored in the master latch

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

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

Master D Latch

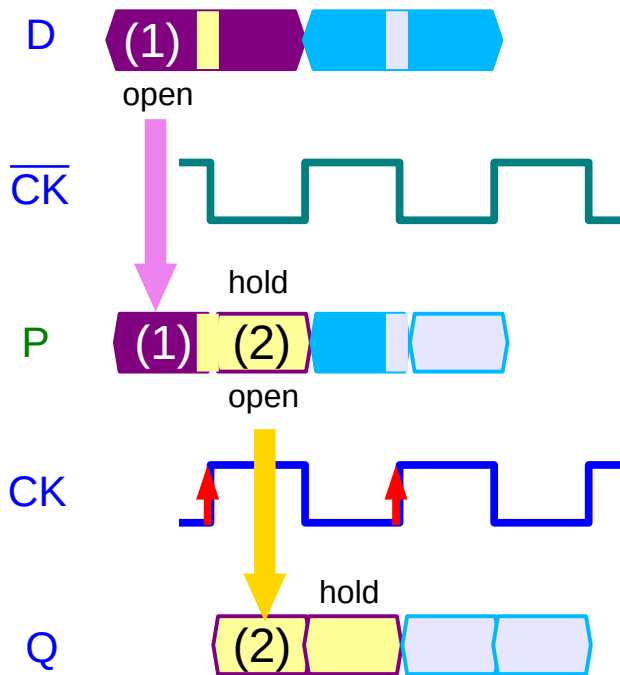


Slave D Latch

Master-Slave D FlipFlop – open and hold

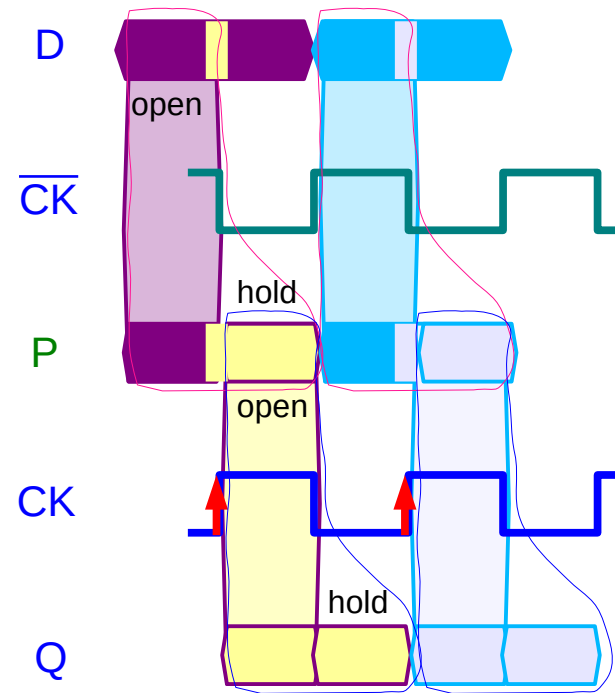
https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design

Master D Latch



Slave D Latch

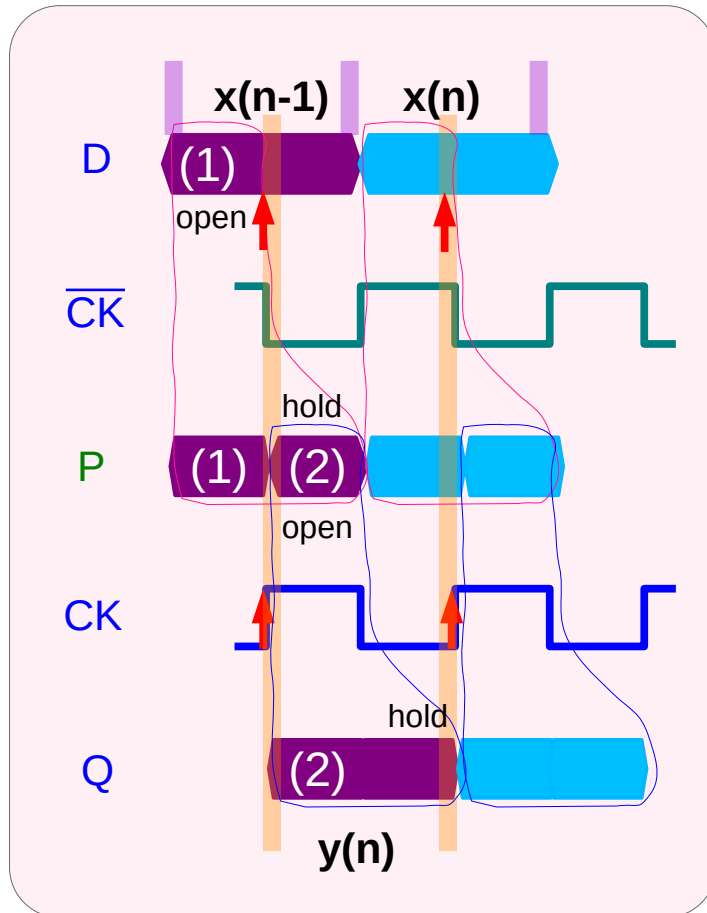
Master D Latch



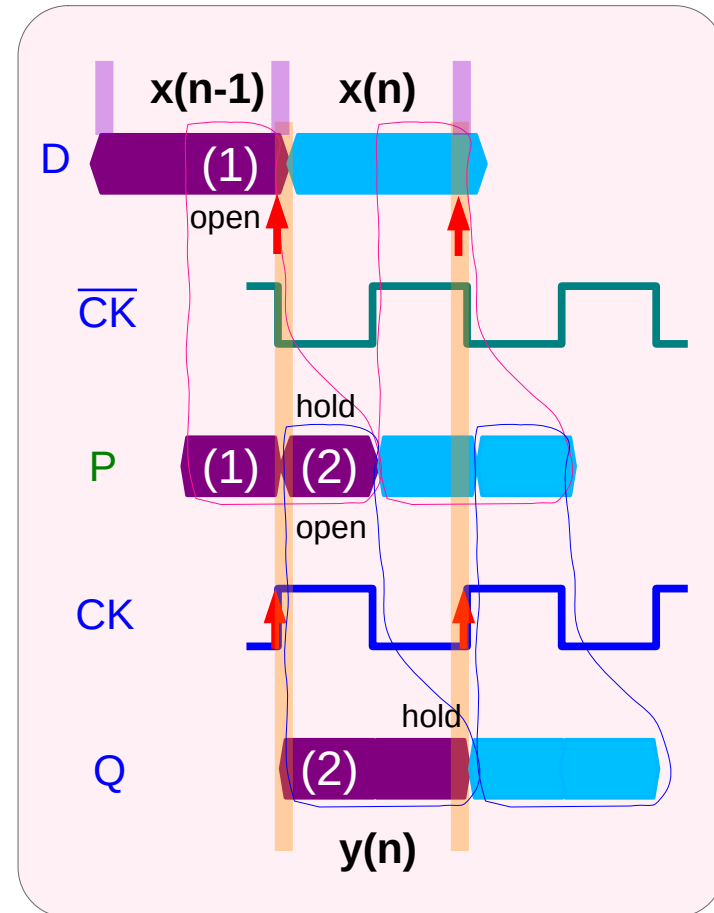
Slave D Latch

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

Typical Timing

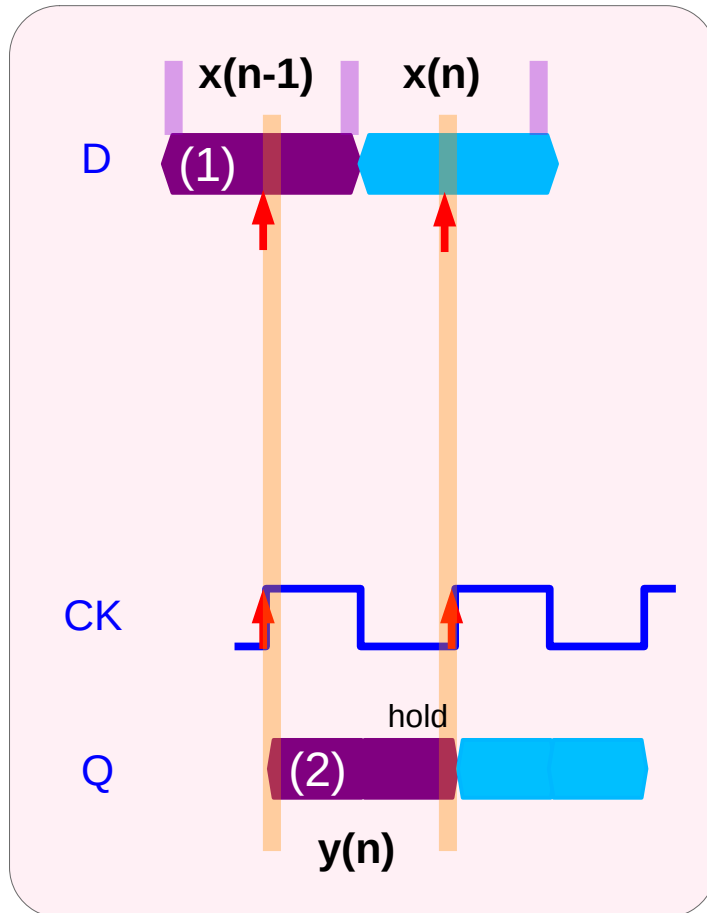


Ideal Timing

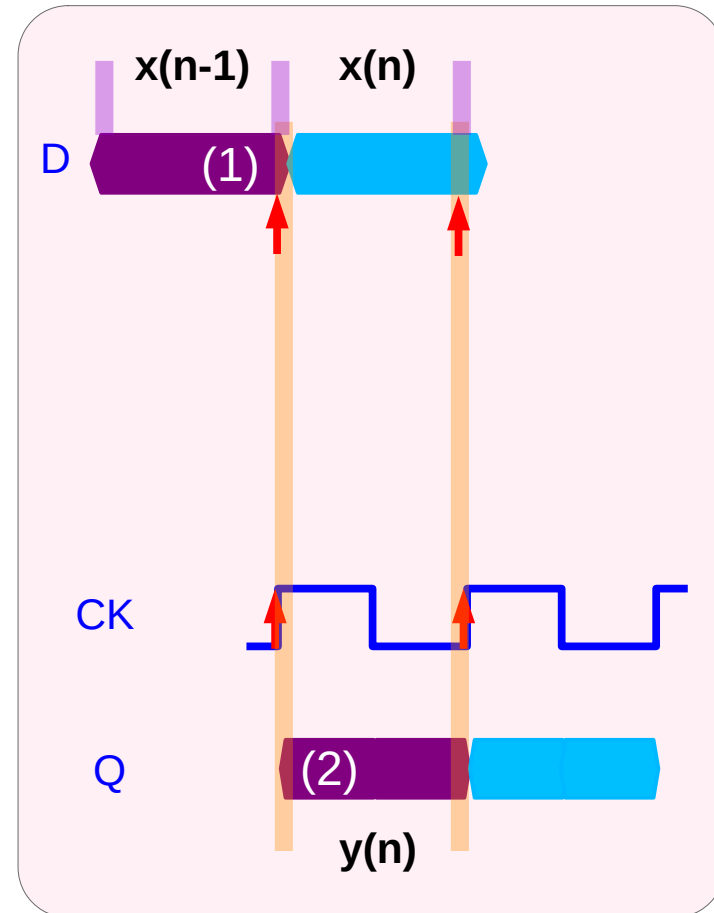


Master-Slave D FlipFlop – typical vs. ideal timing (2)

Typical Timing

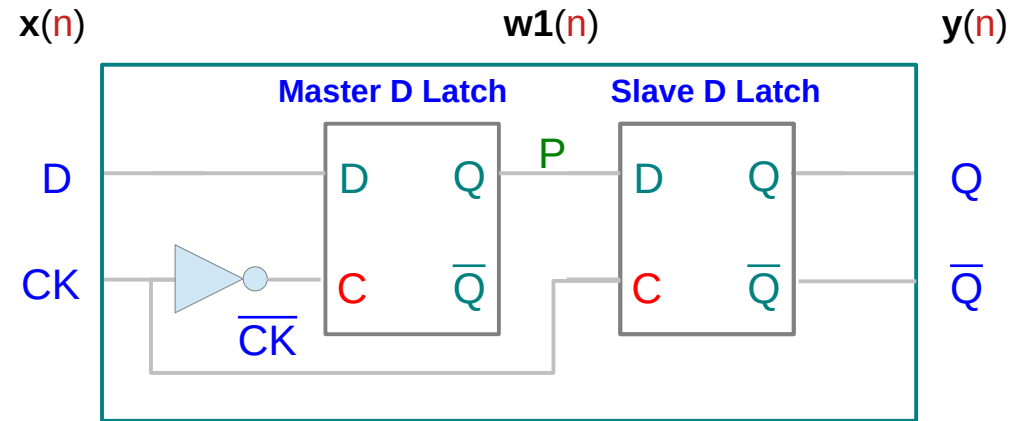
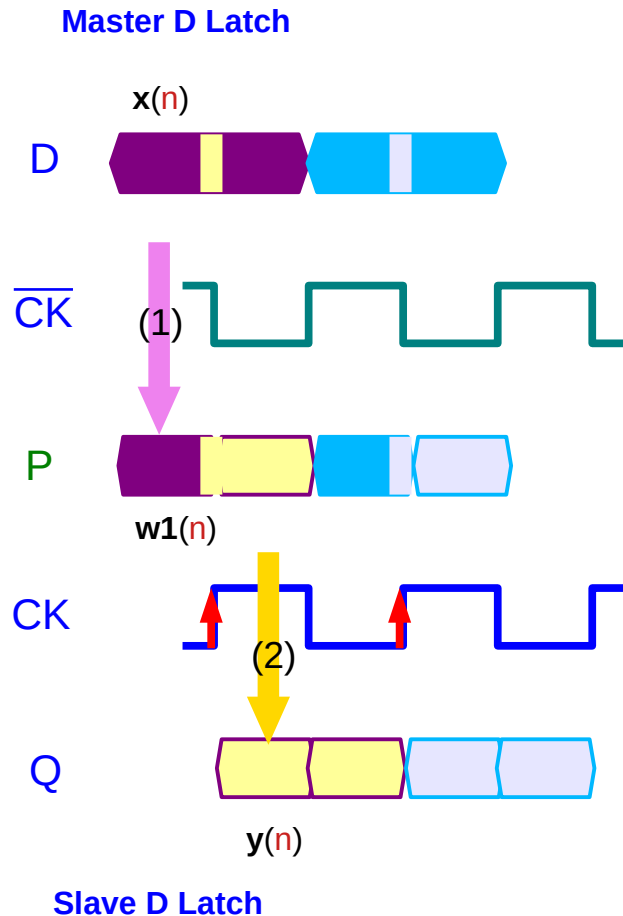


Ideal Timing



Master-Slave D FlipFlop – Rising Edge Sampling

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



(1) the current input $x(n)$ gets stored in the master latch

(2) the current content $w1(n)$ is clocked out to the output $y(n)$

Fixed point representation

fractional numbers

Floating Point Representation

Fixed Point Representation

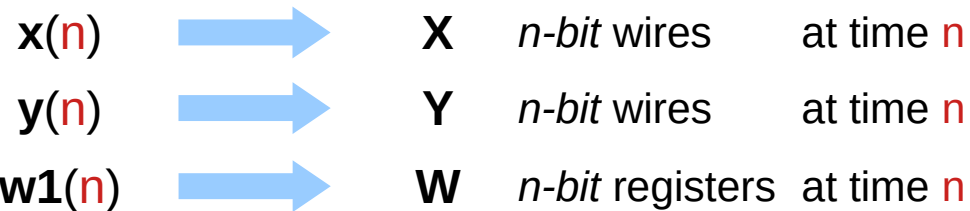
integer + implicit fixed scaling factor

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

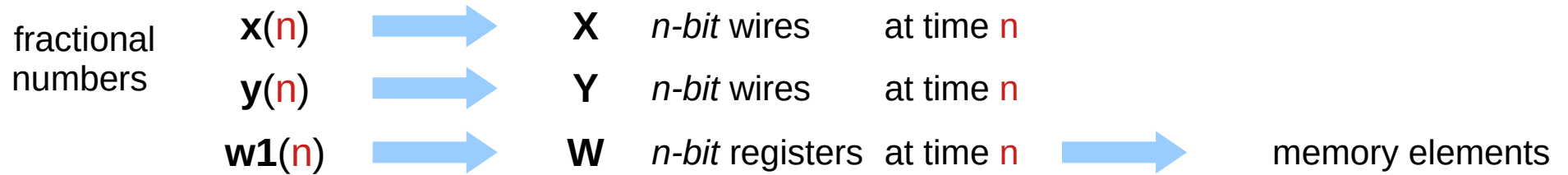
could represent $-11 \cdot 2^{-3} = -88$,
 $-11 \cdot 2^{-5} = -0.34375$
 $-11 \cdot 2^{-12} = -0.002685546875$

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

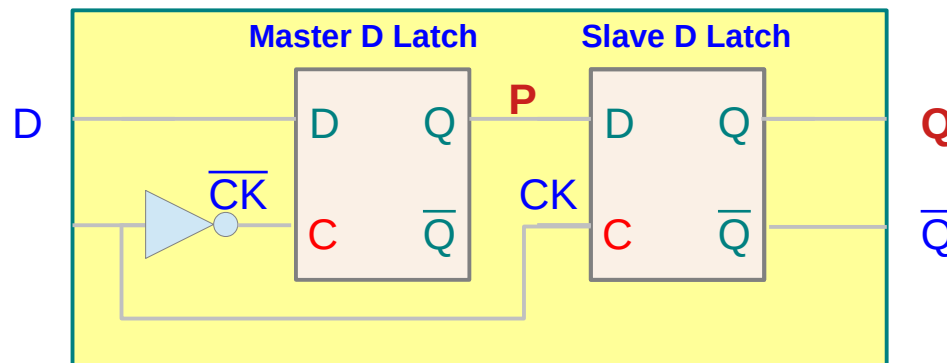
fractional numbers



Memory Elements

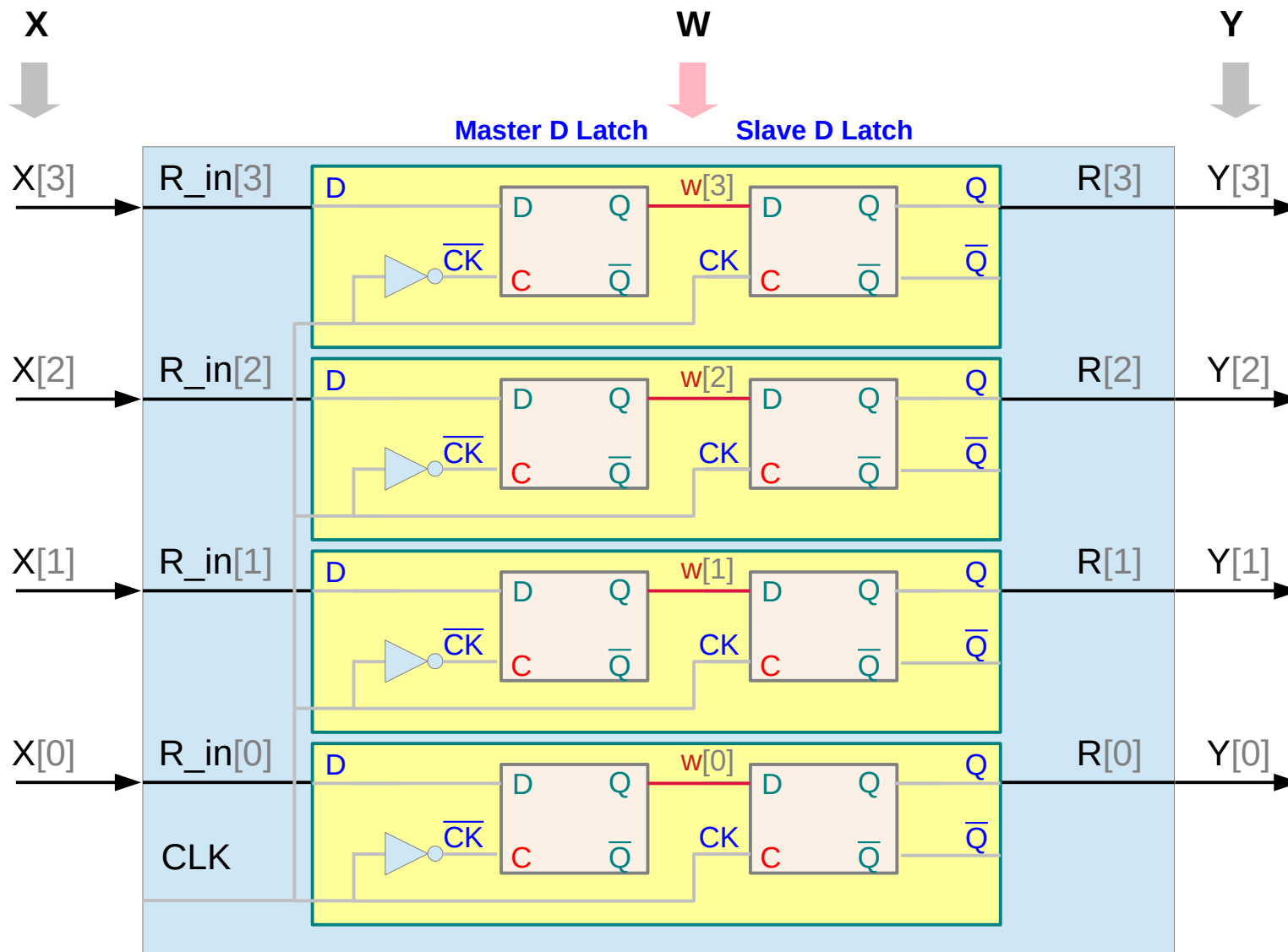


1. using the **P** outputs of **master latches** as the name for **memory elements**
2. using the **Q** outputs of **flipflops** as the name for **memory elements**

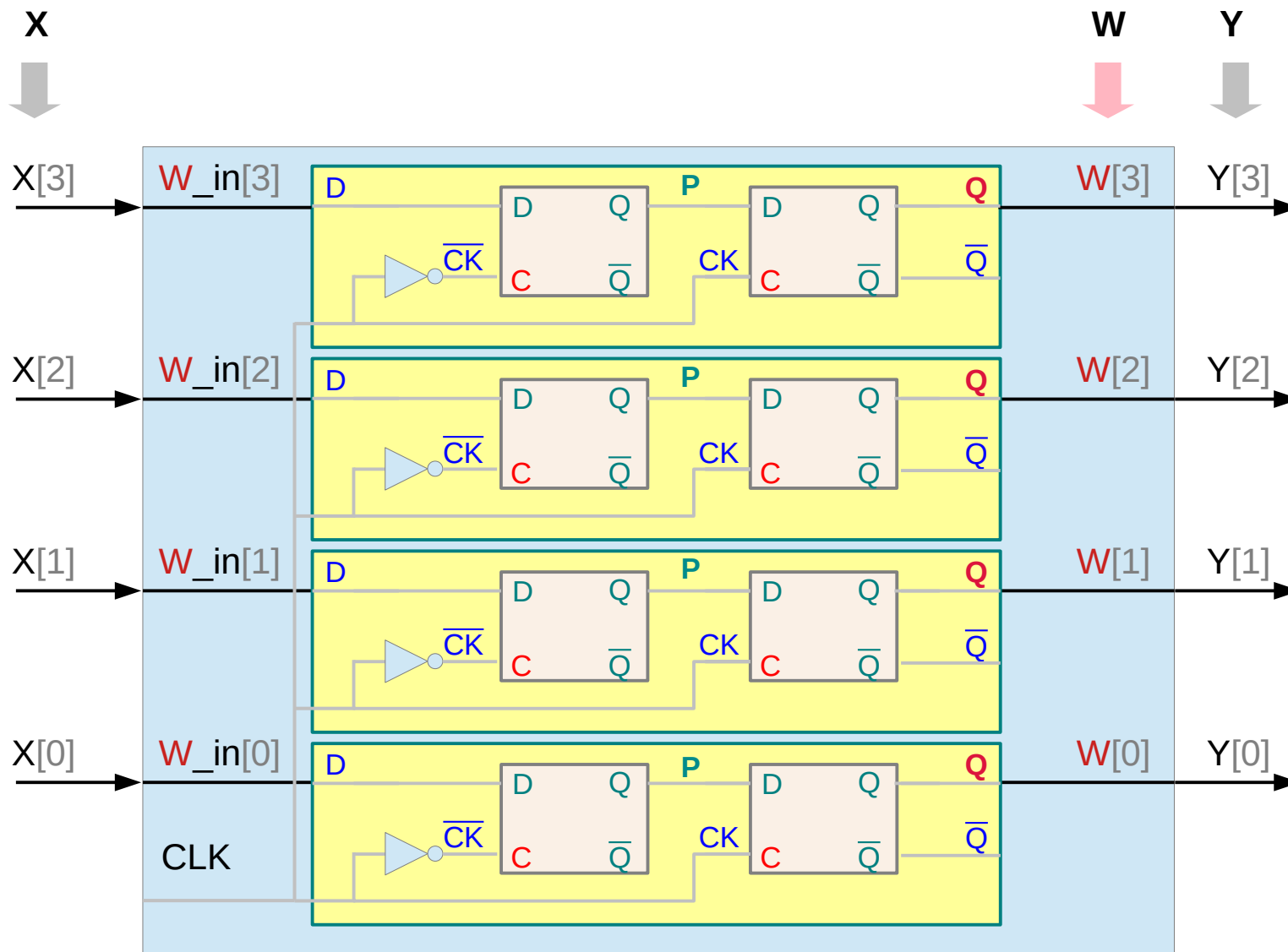


D FlipFlop using Master and Slave D Latches

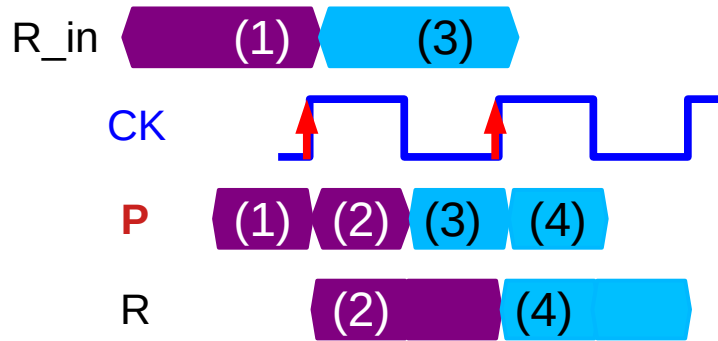
(1) Using master latch outputs P



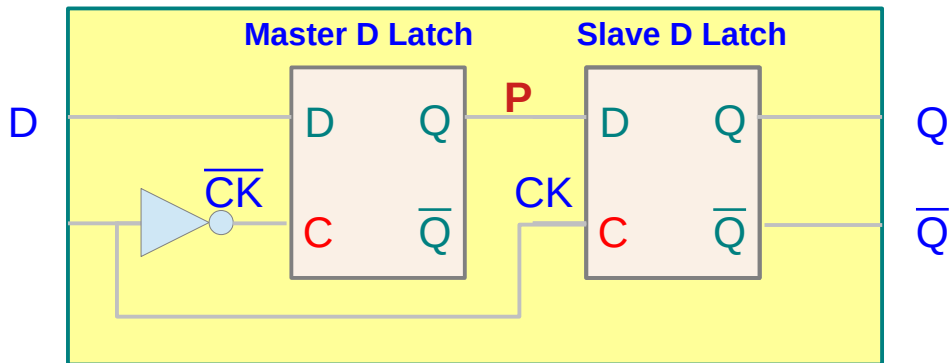
(2) Using flipflop outputs Q



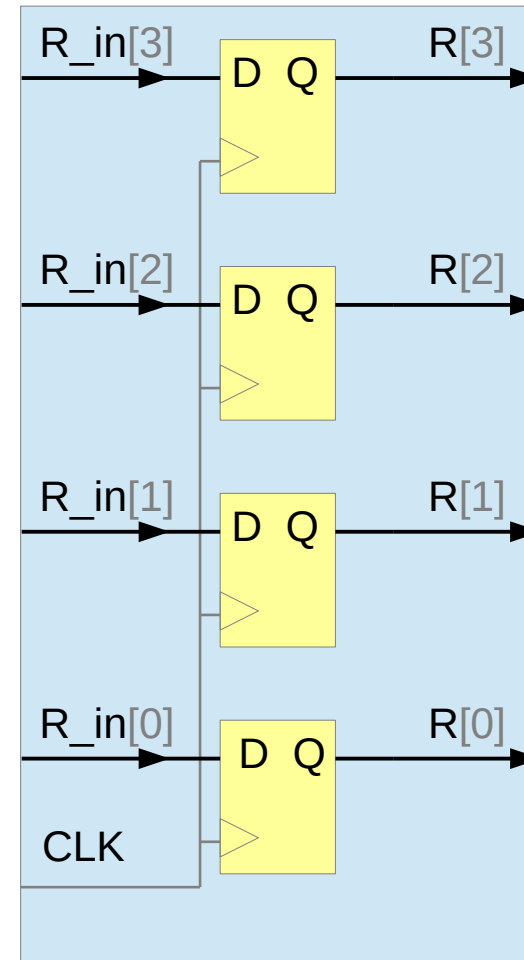
(1) 4-bit Register Using master latch outputs **P**



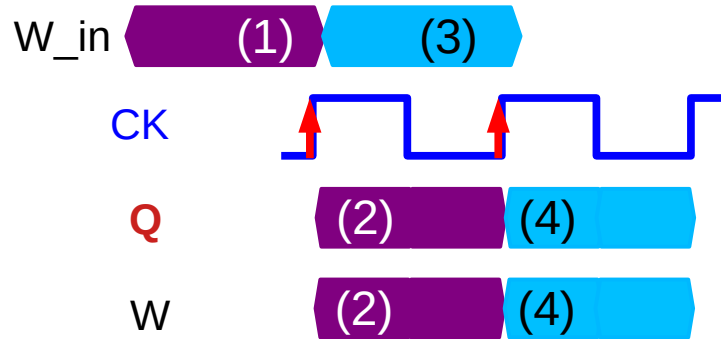
D FlipFlop using Master and Slave D Latches



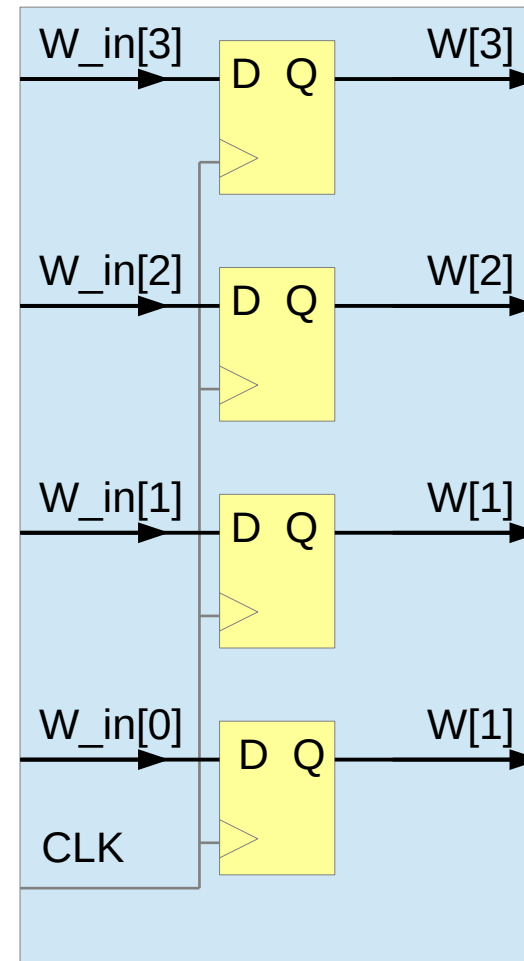
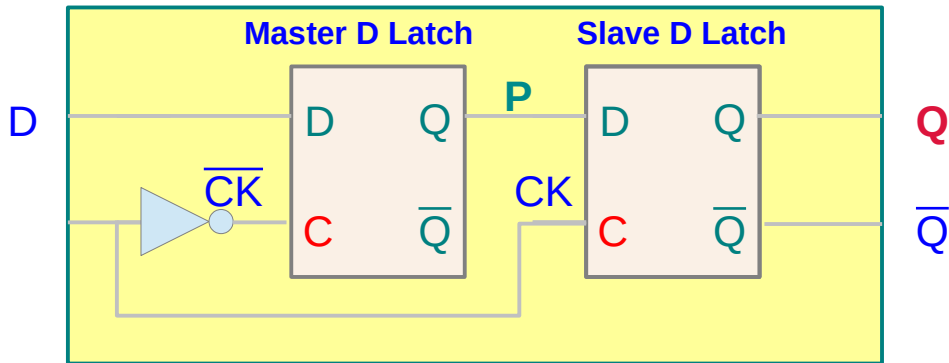
4-bit Register using 4 D flipflops



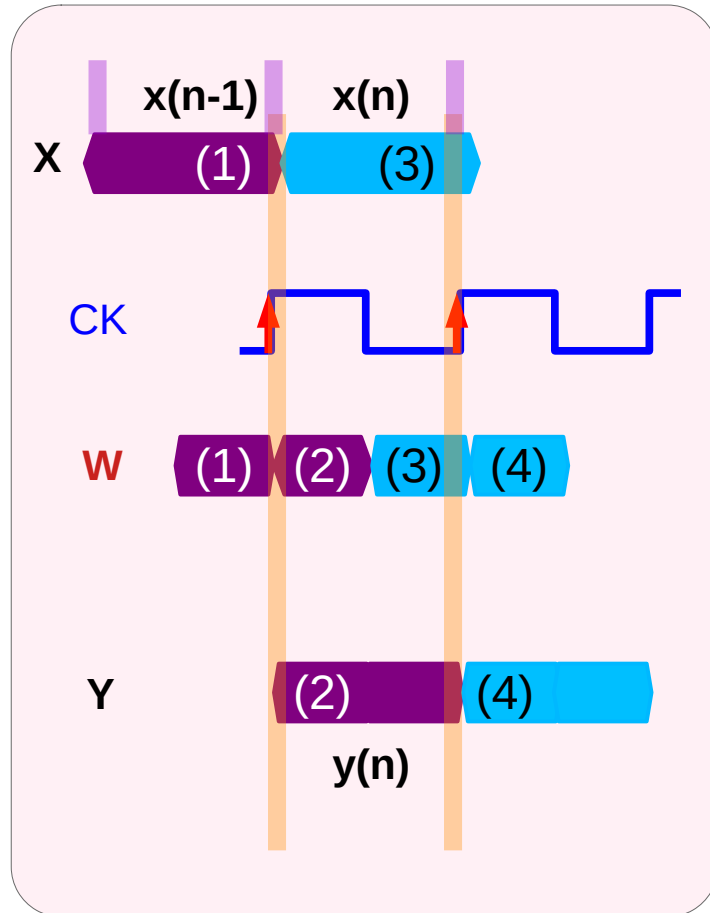
(2) 4-bit Register Using flipflop outputs Q



D FlipFlop using Master and Slave D Latches



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



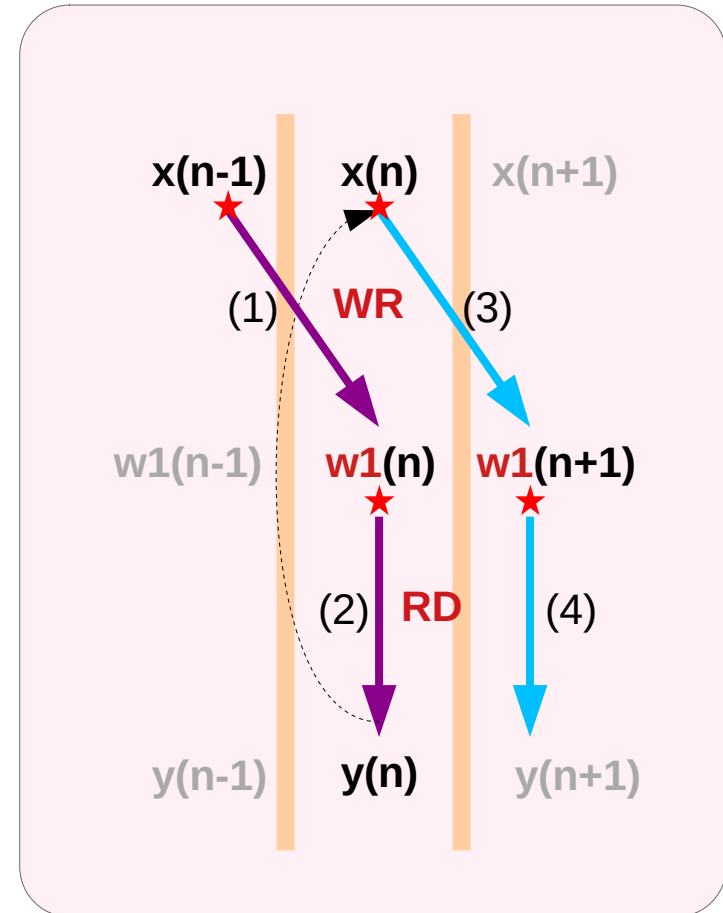
Hardware model

data centered view

Input

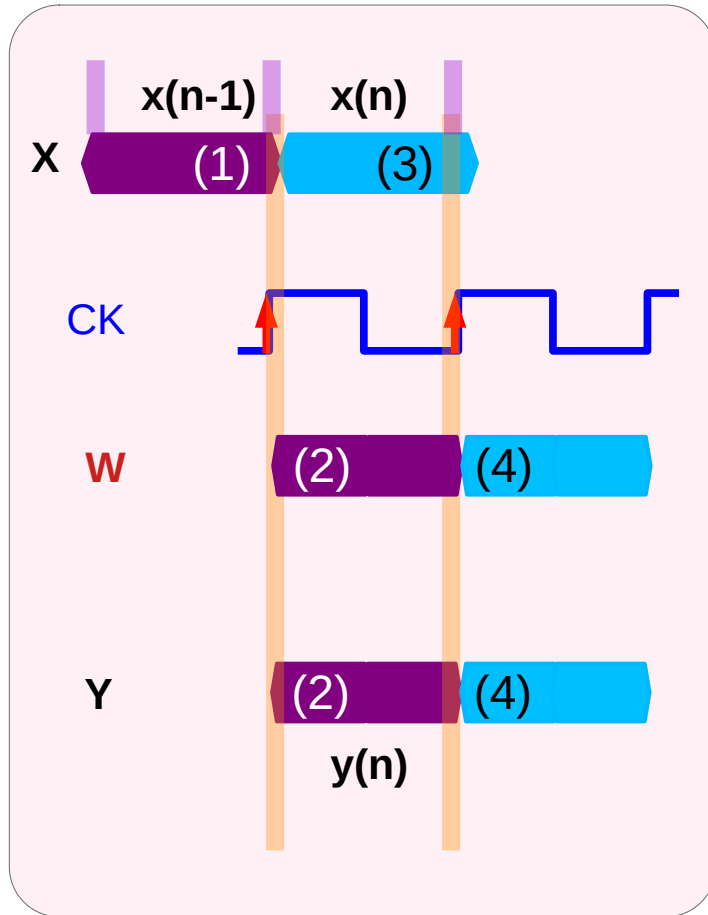
Internal State

Output



The same DSP C model

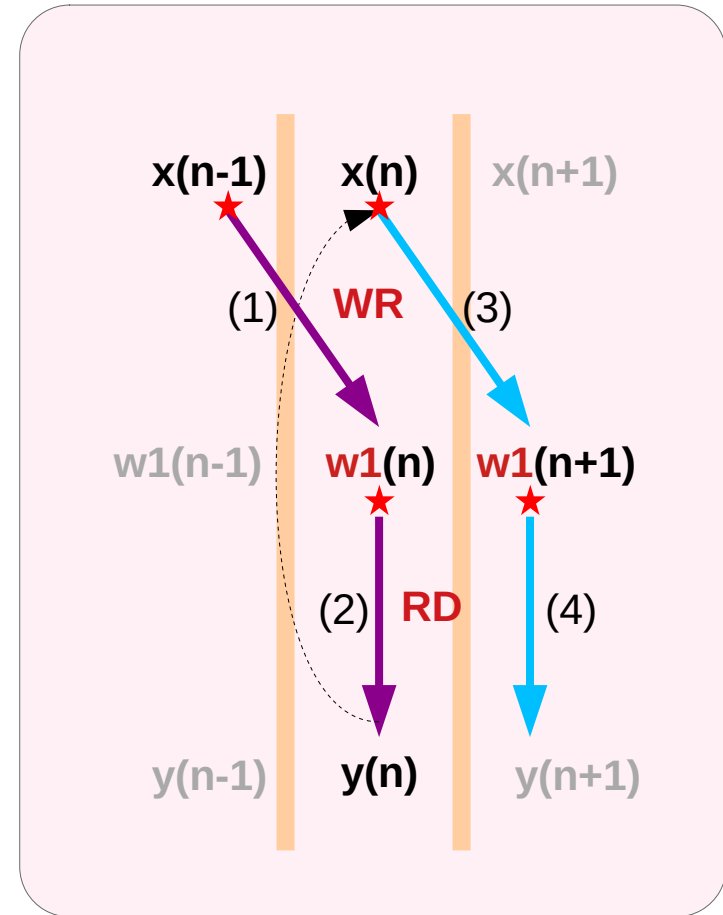
(2) Timing diagrams with flipflop outputs Q



Hardware model

data centered view

Input

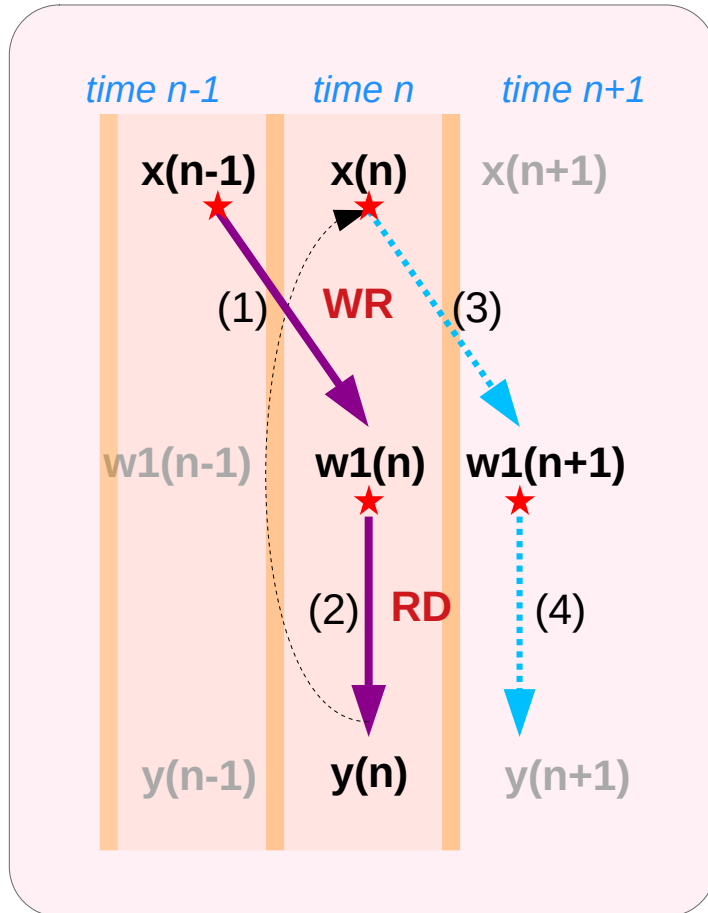


Output

The same DSP C model

Master-Slave D FlipFlop – DSP C model

data centered view



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** $w1(n)$

Input

Internal State

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

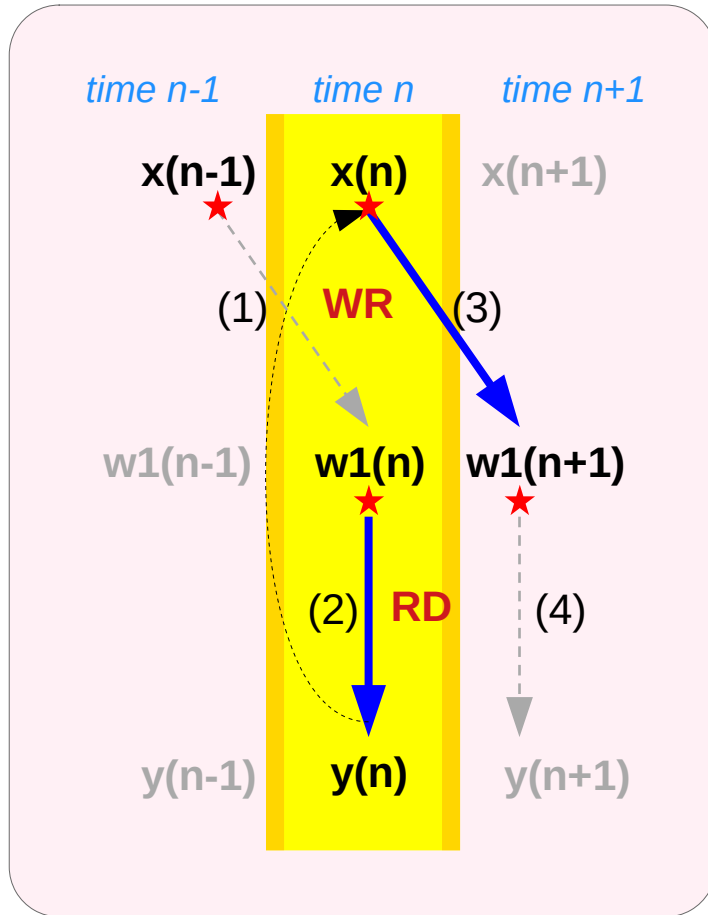
but we must simulate the parallel hardware by a sequential code

Output

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

Master-Slave D FlipFlop – DSP C model

time centered view



DSP C model

operation spans only one time step (n)

Input

- (2) $y(n) = w1(n)$ RD $w1(n)$
- (1) $w1(n) = x(n)$ WR $w1(n)$

Internal State

at the time n RAW (read after write)

No RAW (read after write) hazard

$$y(n) = w1(n)$$

$$w1(n) = x(n)$$

$$w1(n+1) = w1(n)$$

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

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

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

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

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

Output

Master-Slave D FlipFlop – DSP C model

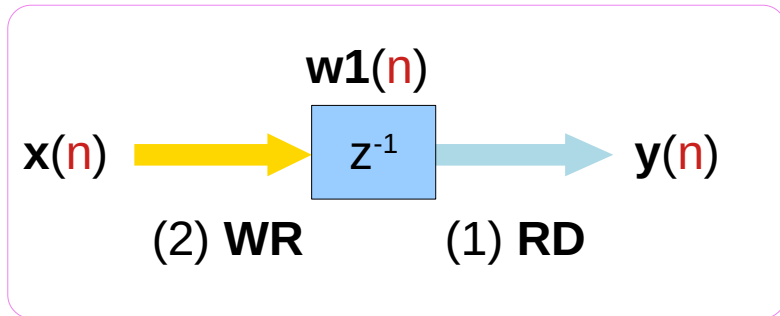
$y(n-1) = w1(n-1)$	$Y = W$
$w1(n-1) = x(n-1)$	$W = X$
$w1(n) = w1(n-1)$	
$y(n) = w1(n)$	$Y = W$
$w1(n) = x(n)$	$W = X$
$w1(n+1) = w1(n)$	
$y(n+1) = w1(n+1)$	$Y = W$
$w1(n+1) = x(n+1)$	$W = X$
$w1(n+2) = w1(n+1)$	
$y(n+2) = w1(n+2)$	$Y = W$
$y(n+2) = w1(n+2)$	$W = X$

$y(n-1) = w1(n-1)$	$Y = W$	} <i>time n-1</i>
$w1(n-1) = x(n-1)$	$W = X$	
$w1(n) = w1(n-1)$		} <i>time n</i>
$y(n) = w1(n)$	$Y = W$	
$w1(n) = x(n)$	$W = X$	} <i>time n+1</i>
$w1(n+1) = w1(n)$		
$y(n+1) = w1(n+1)$	$Y = W$	} <i>time n+2</i>
$w1(n+1) = x(n+1)$	$W = X$	
$w1(n) = w1(n-1)$		
$y(n+2) = w1(n+2)$	$Y = W$	
$y(n+2) = w1(n+2)$	$W = X$	

(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 $w1(n)$	

Simultaneous RD and WR actions



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

at time n

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



at time $n+1$

$$\begin{aligned} y(n+1) &= w1(n+1) \\ w1(n+2) &= x(n+1) \end{aligned}$$

read internal state

update internal state

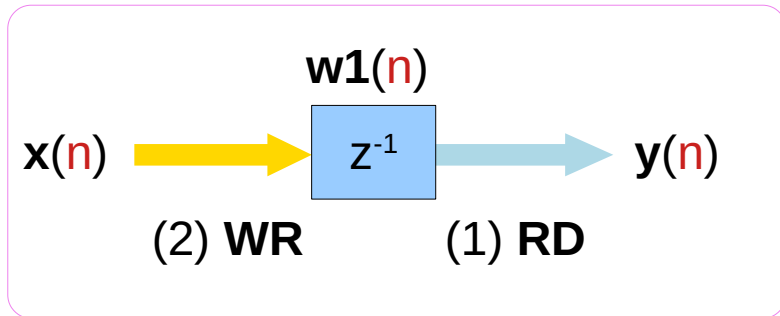
at time n ,

- 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 $x(n)$

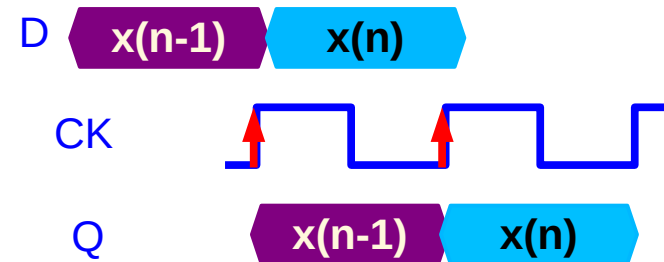
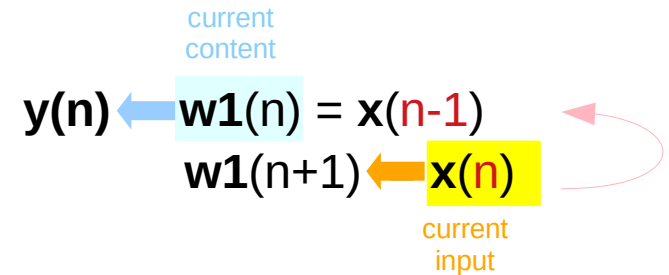


current content	$w1(n)$	$= x(n-1)$
next content	$w1(n+1)$	$= x(n)$

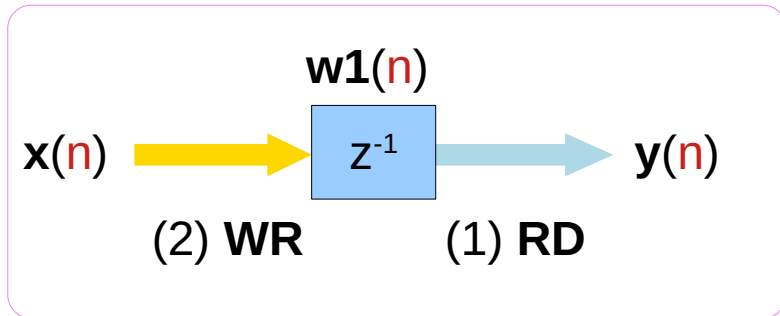
a register holding the previous input sample $x(n-1)$

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

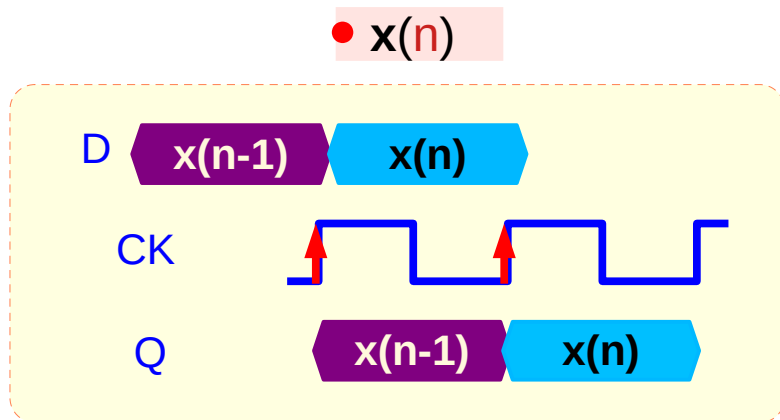
It will be held for one sampling instant and become the output at the next time $n+1$



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

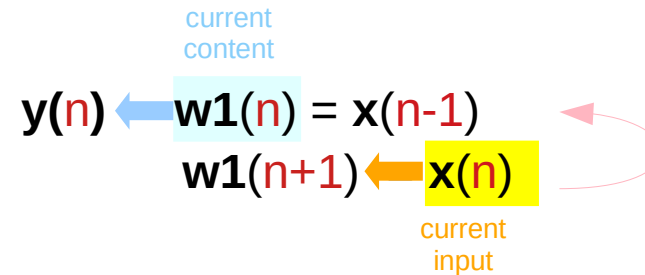


current content	$w1(n)$	$= x(n-1)$
next content	$w1(n+1)$	$= x(n)$



• $w1(n)$

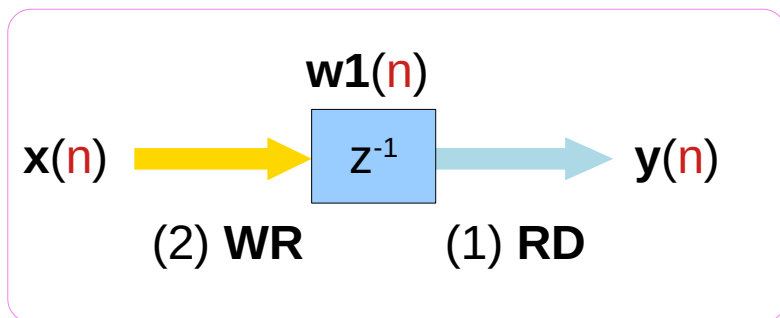
• $y(n)$



simulate a clocked hardware ignoring delay constraints in hardware

zero-delay simulation

Delay element modeling



$w1(n) \longrightarrow y(n)$	$y(n) = w1(n)$ (1) RD old w1
$x(n) \longrightarrow w1(n+1)$	$w1(n+1) = x(n)$ (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

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

internal state at time $n+1$

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

output at time n

$$y(n) = w1(n)$$

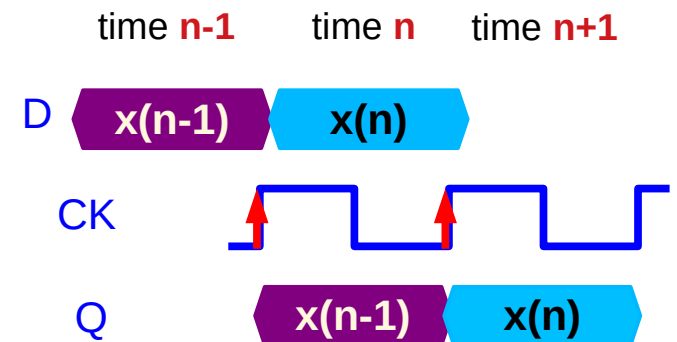
RD before WR

WAR (Write after Read) Access

WR at time $n-1$

WR at time n

RD at time n



*simulate a clocked hardware
ignoring delay constraints in hardware*

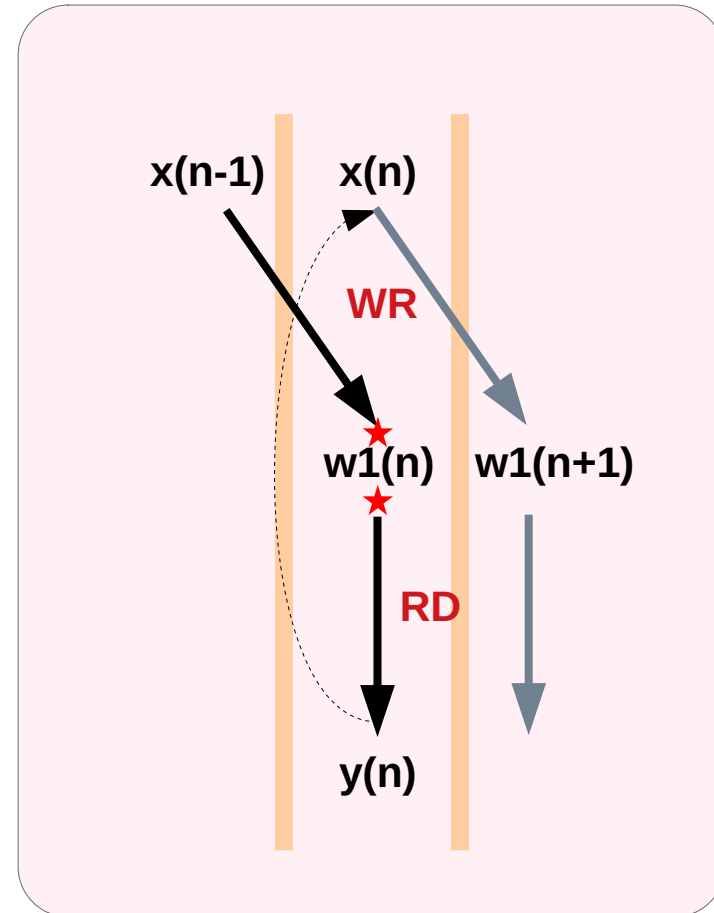
zero-delay simulation

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

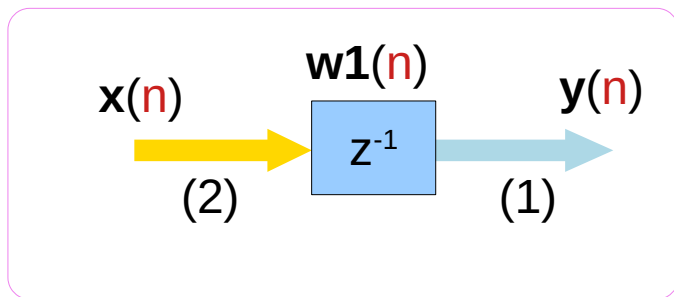
$w1(n+1) = x(n)$	(2) WR	new w1
$y(n) = w1(n)$	(1) RD	old w1
$w1(n+2) = x(n+1)$	(2) WR	new w1
$y(n+1) = w1(n+1)$	(1) RD	old w1

WAR (Write after Read) Violation



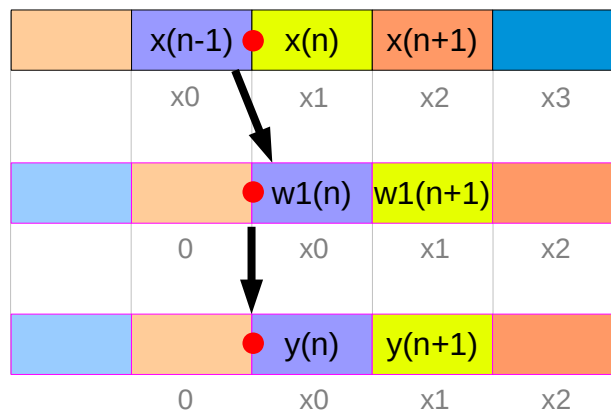
DSP C model

Single Delay



$$y(n) = w1(n) \quad (1) \text{ RD } \rightarrow$$

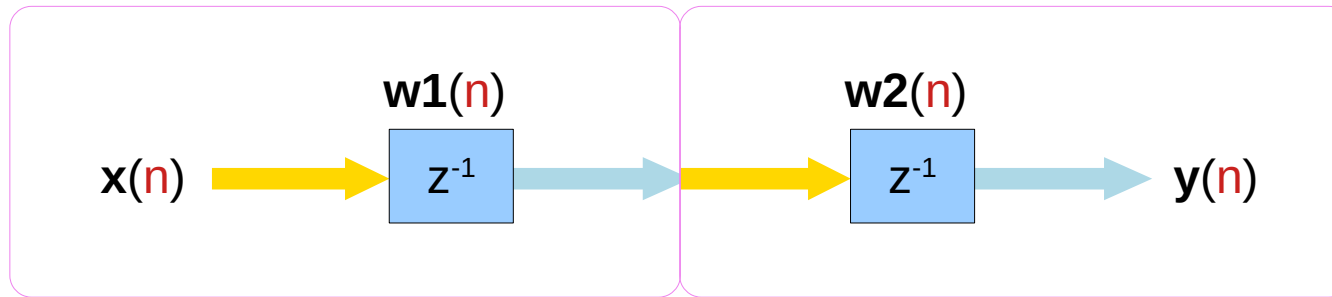
$$w1(n+1) = x(n) \quad (2) \text{ WR } \rightarrow$$



n	$x(n)$	$w1(n)$	$y(n)$
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

Arrows show the relationship between columns: $x(n) \rightarrow w1(n)$ and $w1(n) \rightarrow y(n)$.

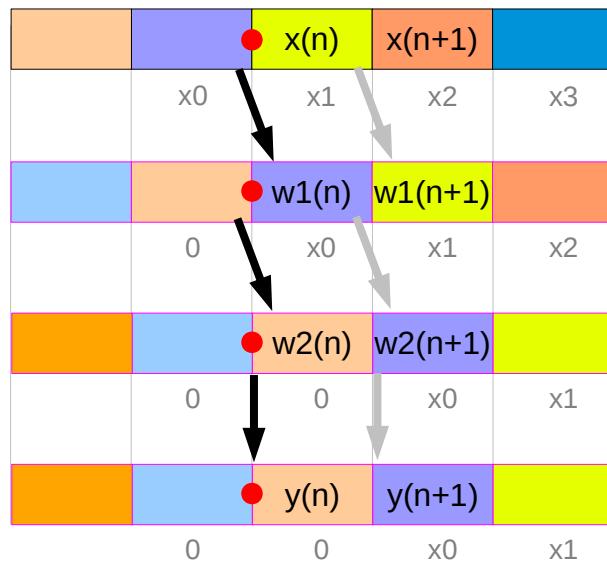
Double Delay



$$y(n) = w2(n)$$

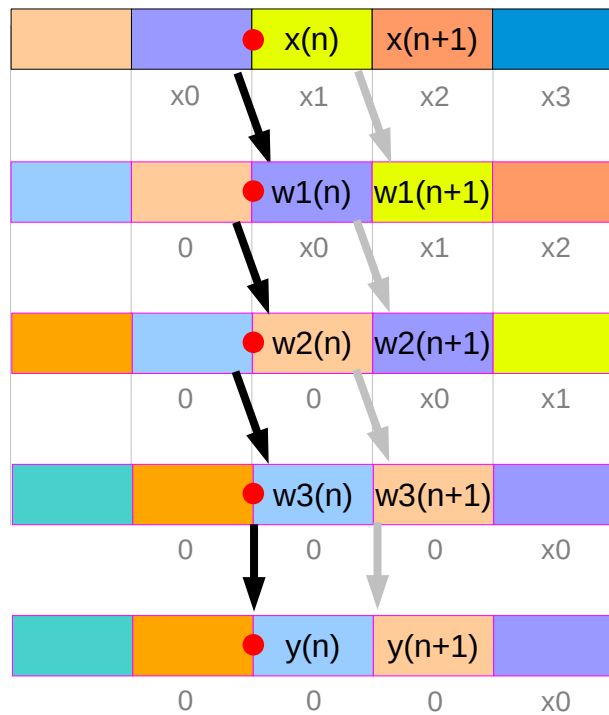
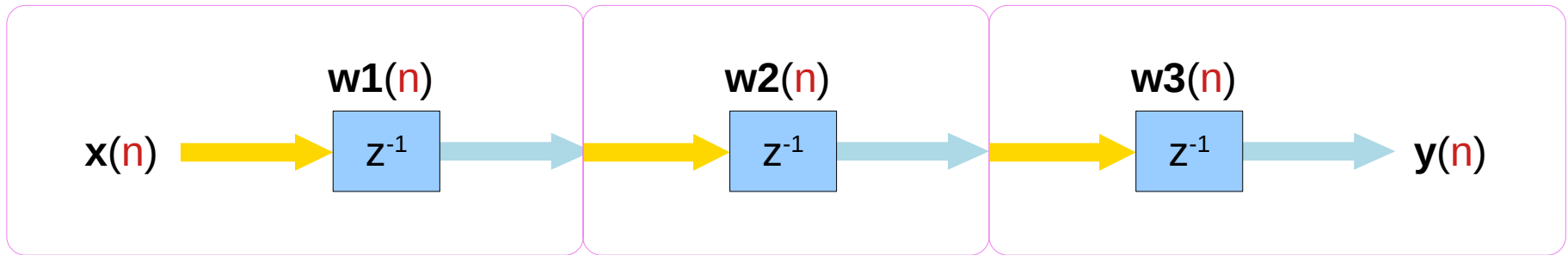
$$w2(n+1) = w1(n)$$

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



n	$x(n)$	$w1(n)$	$w2(n)$	$y(n)$
0	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

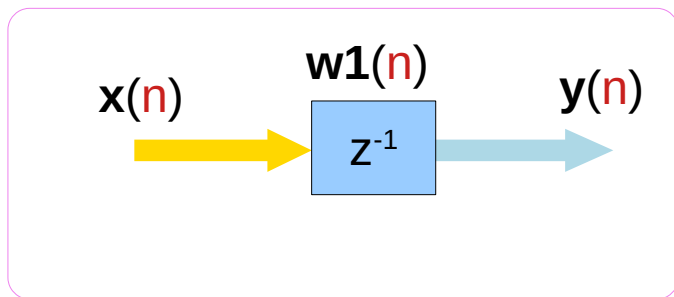
Triple Delay



n	$x(n)$	$w1(n)$	$w2(n)$	$w3(n)$	$y(n)$
0	x_0	0	0	0	0
1	x_1	x_0	0	0	0
2	x_2	x_1	x_0	0	0
3	x_3	x_2	x_1	x_0	x_0
4	x_4	x_3	x_2	x_1	x_1

$$\begin{aligned}
 y(n) &= w3(n) \\
 w3(n+1) &= w2(n) \\
 w2(n+1) &= w1(n) \\
 w1(n+1) &= x(n)
 \end{aligned}$$

Single Delay – IO Equations



single delay

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

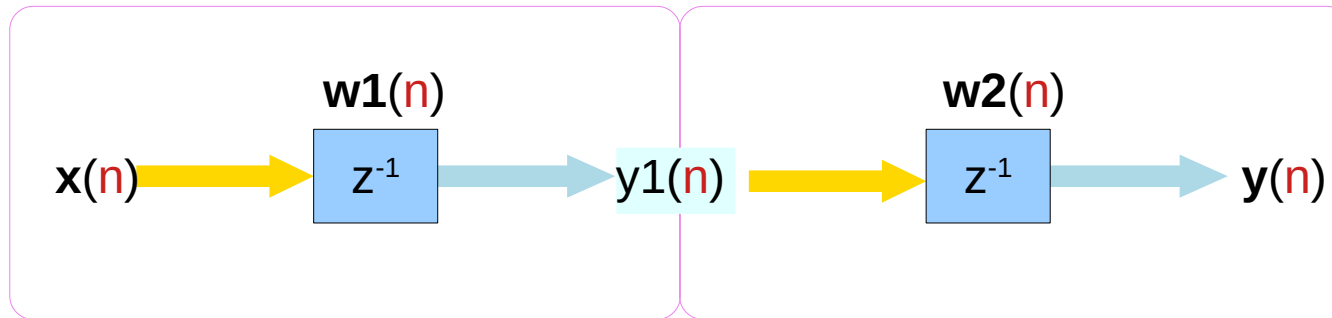
double delay

$y(n) = w2(n)$	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



$$y1(n) = w1(n)$$

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

$$y(n) = w2(n)$$

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

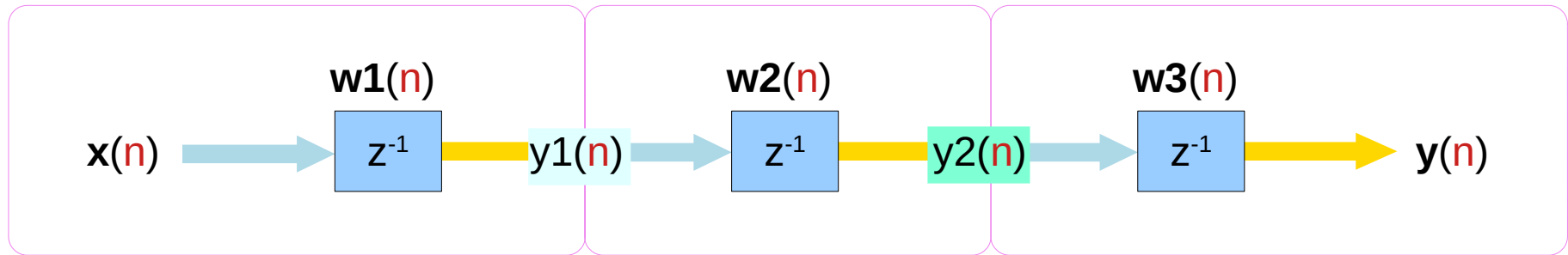
$$y1(n) = w1(n)$$

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

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

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

Triple Delay – IO Equations



$$y1(n) = w1(n)$$

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

$$y2(n) = w2(n)$$

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

$$y(n) = w3(n)$$

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

$$y1(n) = w1(n)$$

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

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

$$y2(n) = w2(n)$$

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

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

$$y(n) = w3(n) \quad \text{output}$$

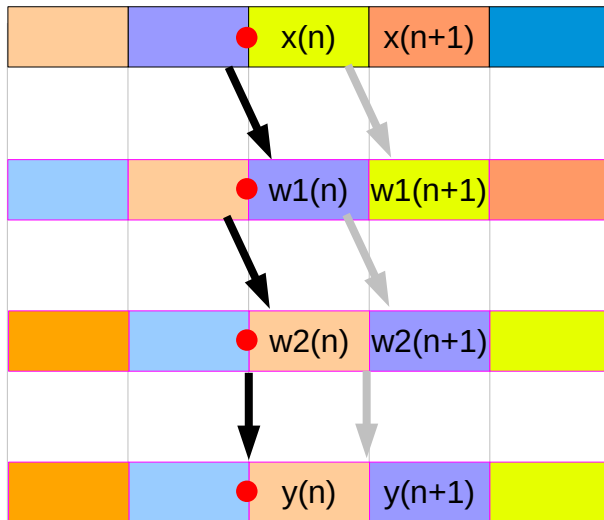
$$w3(n+1) = w2(n)$$

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

$$w1(n+1) = x(n) \quad \text{input}$$

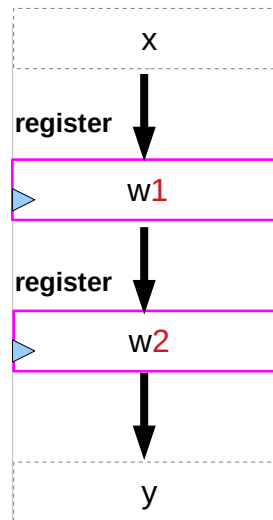
Delay C Model

Timing Chart



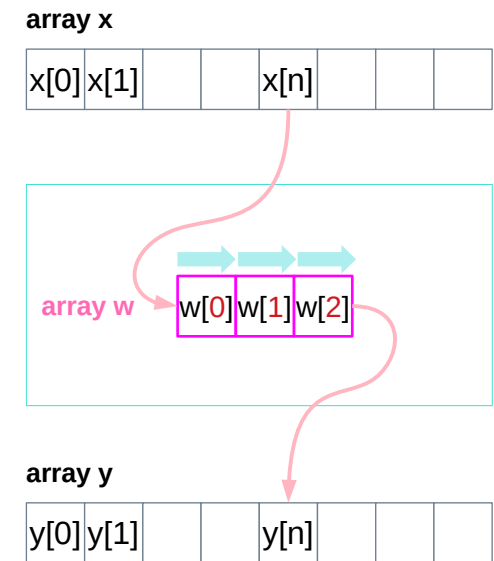
$$\begin{aligned}
 y(n) &= w2(n) \\
 w2(n+1) &= w1(n) \\
 w1(n+1) &= x(n)
 \end{aligned}$$

Register Transfer



$$\begin{aligned}
 y &= w2 \\
 w2 &= w1 \\
 w1 &= x
 \end{aligned}$$

DSP C Model for simulation



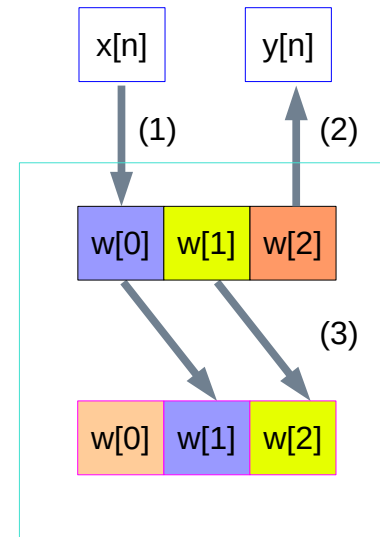
$$\begin{aligned}
 y[n] &= w[2] \\
 w[0] &= x[n] \\
 w[2] &= w[1] \\
 w[1] &= w[0]
 \end{aligned}$$

IO Equations for the Triple Delay (1)

$$\begin{aligned}y(n) &= w_2(n) \\w_0(n) &= x(n) \\w_2(n+1) &= w_1(n) \\w_1(n+1) &= w_2(n)\end{aligned}$$

$$D = 2, 1$$

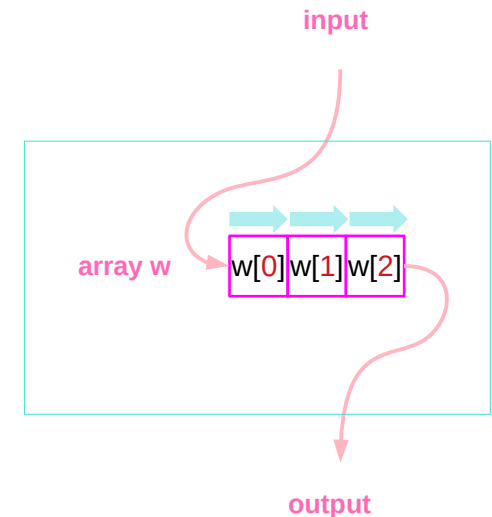
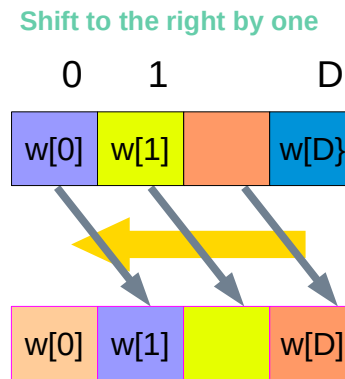
```
y[n] = w[2]           // get the output
w[0] = x[n]           // put the input
w[2] = w[1]           // shift
w[1] = w[0]           // shift
```



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  
}
```



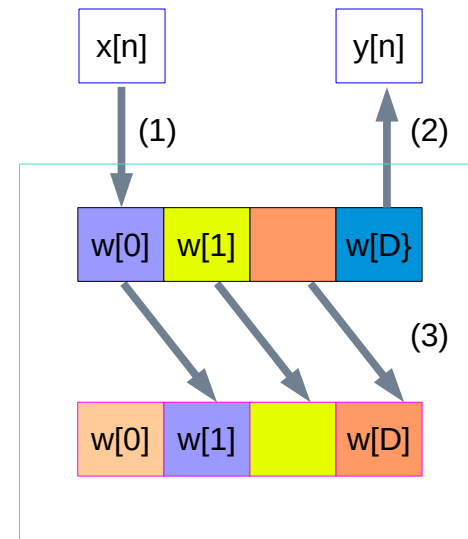
order of execution

$$\begin{aligned} w[D] &= w[D-1] \\ \dots & \quad \dots \\ w[2] &= w[1] \\ w[1] &= w[0] \end{aligned}$$

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

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

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

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

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

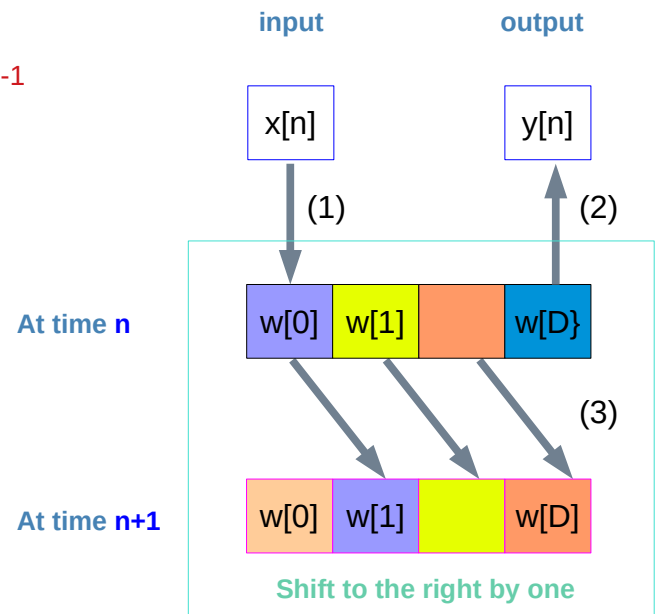
time index : n

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



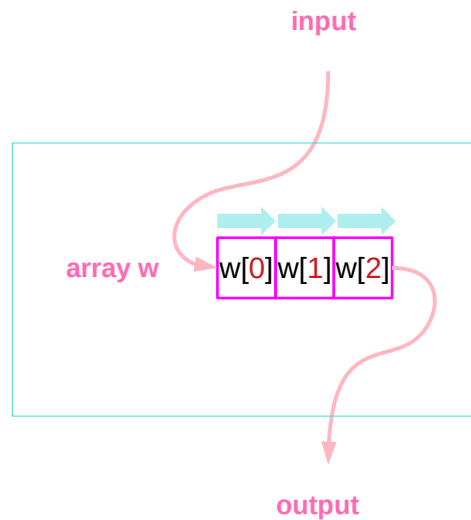
Holding a delayed input sequence

$$w_0(n) = x(n)$$

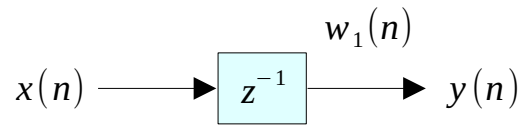
$$w_1(n) = x(n-1) = w_0(n-1)$$

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

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



Single Delay (1)



$$w_1(n) = x(n-1) \quad (\text{internal state at time } n)$$

$$w_1(n+1) = x(n) \quad (\text{internal state at time } n+1)$$

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

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

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

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

$$w_1(0) = 0$$

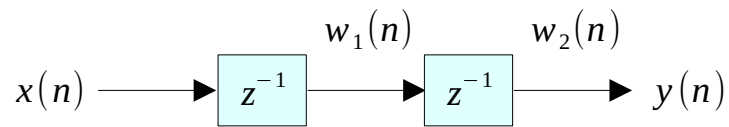
$$[x_0, x_1, x_2, x_3, \dots] \rightarrow [0, x_0, x_1, x_2, x_3, \dots]$$

for each input sample x do:

$$y := w_1$$

$$w_1 := x$$

Double Delay (1)



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

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

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

$$w_1(n+1) = x(n) \quad 0 \quad x_0 \quad 0 \quad 0 \quad 0$$

$$1 \quad x_1 \quad x_0 \quad 0 \quad 0$$

$$y(n) = w_2(n) \quad 2 \quad x_2 \quad x_1 \quad x_0 \quad x_0$$

$$w_2(n+1) = w_1(n) \quad 3 \quad x_3 \quad x_2 \quad x_1 \quad x_1$$

$$w_1(n+1) = x(n) \quad 4 \quad x_4 \quad x_3 \quad x_2 \quad x_2$$

$$w_1(0) = 0$$

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

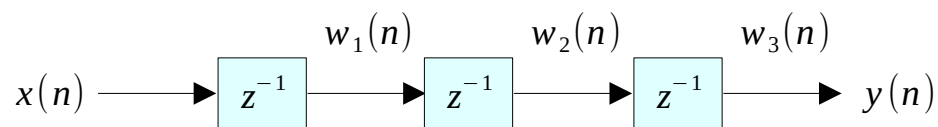
for each input sample x do:

$$y := w_2$$

$$w_2 := w_1$$

$$w_1 := x$$

Triple Delay (1)



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

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

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

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

for each input sample x do:

$$y := w_3$$

$$w_3 := w_2$$

$$w_2 := w_1$$

$$w_1 := x$$

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

$$w_2(n+1) = w_1(n) \quad 0 \quad x_0 \quad 0 \quad 0 \quad 0$$

$$w_1(n+1) = x(n) \quad 1 \quad x_1 \quad x_0 \quad 0 \quad 0$$

$$2 \quad x_2 \quad x_1 \quad x_0 \quad x_0$$

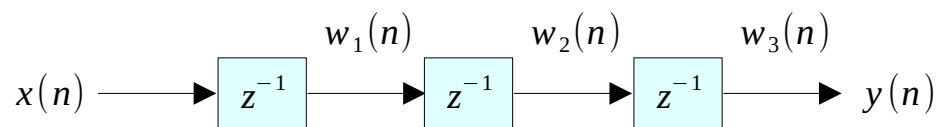
$$y(n) = w_3(n) \quad 3 \quad x_3 \quad x_2 \quad x_1 \quad x_1$$

$$w_3(n+1) = w_2(n) \quad 4 \quad x_4 \quad x_3 \quad x_2 \quad x_2$$

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

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

D Unit Delay (1)



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

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

$$w_2(n+1) = w_1(n) \quad 0 \quad x_0 \quad 0 \quad 0 \quad 0$$

$$w_1(n+1) = x(n) \quad 1 \quad x_1 \quad x_0 \quad 0 \quad 0$$

$$2 \quad x_2 \quad x_1 \quad x_0 \quad x_0$$

$$y(n) = w_D(n) \quad 3 \quad x_3 \quad x_2 \quad x_1 \quad x_1$$

$$w_0(n) = x(n) \quad 4 \quad x_4 \quad x_3 \quad x_2 \quad x_2$$

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

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

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

for each input sample x do:

$$y := w_D$$

$$w_0 := x$$

$$w_0 := x$$

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

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

for each input sample w_0 do:

for $i = D, D-1, \dots, 1$ 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)
```

```
{
```

```
    int i;
```

```
    double y;
```

```
    for (y=0, i=0; i<=M; i++)
```

```
        y += h[i] * w[i];
```

```
    return y;
```

```
}
```

Usage: y = dot(M, h, w);

h = filter vector, w = state vector

M = filter order

compute dot product

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

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