Delay Model in Verilog

20151201

Some useful documents

gate

http://www.doe.carleton.ca/~jknight/97.478/SimVry2C.PDF

http://www.sunburst-design.com/papers/CummingsSNUG2002Boston_NBAwithDelays.pdf

http://www.asic-world.com/verilog/gate3.html

Noblocking Assignment:

http://www-inst.eecs.berkeley.edu/~cs152/fa06/handouts/CummingsHDLCON1999_BehavioralDelays_Rev1_1.pdf

http://cs.haifa.ac.il/courses/verilog/1996-CUG-presentation_nonblocking_assigns.pdf

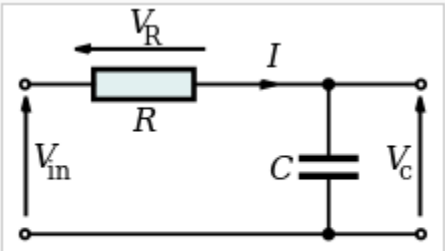
RC Circuits

By viewing the circuit as a voltage divider, the voltage across the capacitor is:

$$V_C(s) = \frac{1/Cs}{R + 1/Cs} V_{in}(s) = \frac{1}{1 + RCs} V_{in}(s)$$

and the voltage across the resistor is:

$$V_R(s) = \frac{R}{R + 1/Cs} V_{in}(s) = \frac{RCs}{1 + RCs} V_{in}(s).$$



Series RC circuit

Time-domain considerations [edit]

This section relies on knowledge of e, *the natural logarithmic constant.*

The most straightforward way to derive the time domain behaviour is to use the Laplace transforms of the expressions for V_C and V_R given above. This effectively transforms $j\omega \to s$. Assuming a step input (i.e. $V_{in} = 0$ before $t = 0$ and then $V_{in} = V$ afterwards):

$$V_{in}(s) = V\frac{1}{s}$$

$$V_C(s) = V\frac{1}{1 + sRC}\frac{1}{s}$$

and

$$V_R(s) = V\frac{sRC}{1 + sRC}\frac{1}{s}.$$

Partial fractions expansions and the inverse Laplace transform yield:

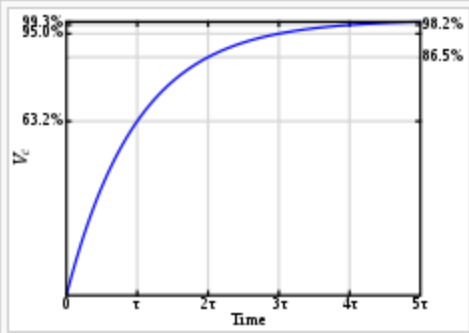
$$V_C(t) = V\left(1 - e^{-t/RC}\right)$$

$$V_R(t) = Ve^{-t/RC}.$$

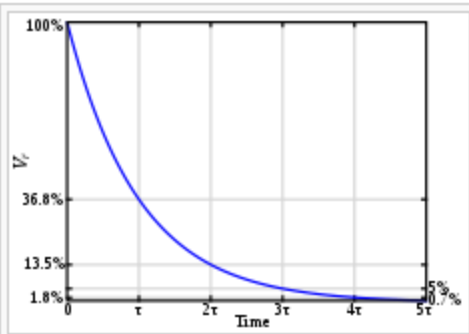
Transient Response

Partial fractions expansions and the inverse Laplace transform yield:

$$V_C(t) = V\left(1 - e^{-t/RC}\right)$$
$$V_R(t) = Ve^{-t/RC}.$$



Capacitor voltage step-response.



Resistor voltage step-response.

exponentially increase
exponentially decrease

$e^{-\frac{t}{RC}} \Rightarrow e^{-1} = 36.8\%$

$t = RC$ 63.2%

time constant

Time Constants

```
(%i1) f(x) := %e^-x;
(%o1) f(x):=%e^-x
(%i2) float(%e^-1);
(%o2) 0.36787944117144
(%i3) float(%e^-2);
(%o3) 0.13533528323661
(%i4) float(%e^-3);
(%o4) 0.049787068367863
(%i5) float(%e);
(%o5) 2.718281828459045
 -->
 -->  wxplot2d([f(x)], [x,0,5]);
```

$e^{-1} = 36.8\%$

$e^{-2} = 13.5\%$

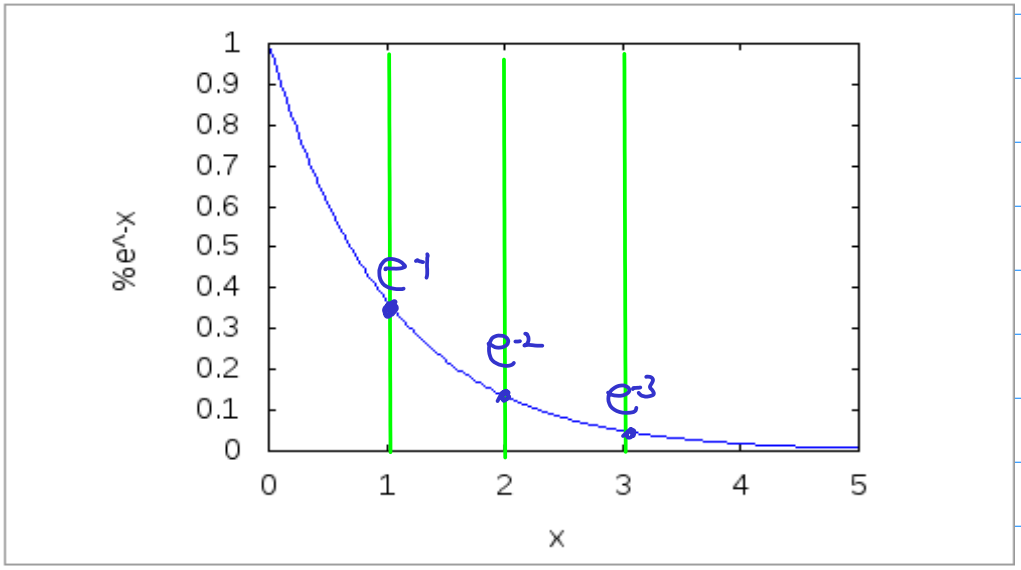
$e^{-3} = 5.0\%$

$1-e^{-1} = 63.2\%$

$1-e^{-2} = 86.5\%$

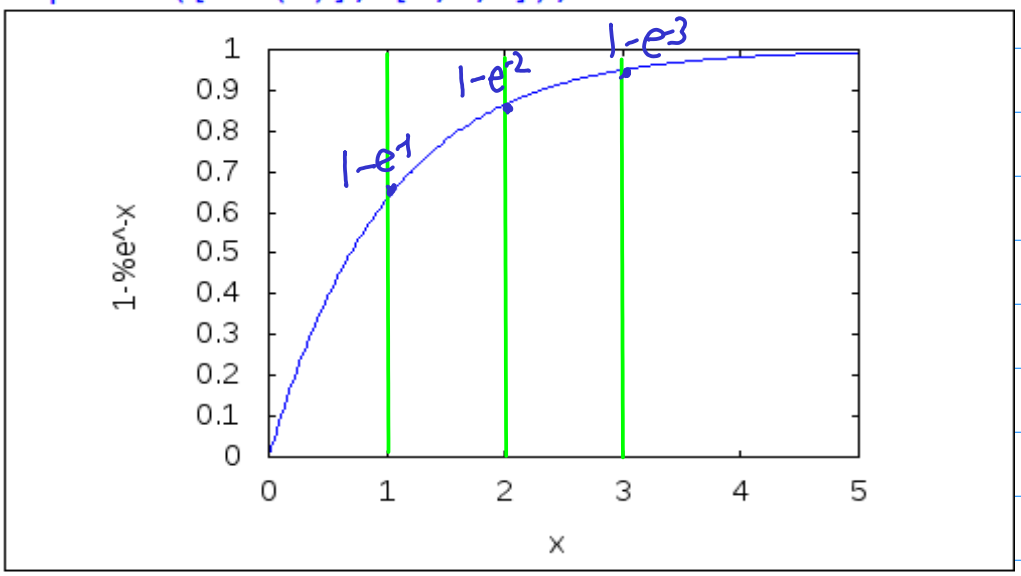
$1-e^{-3} = 95.0\%$

(%t7)




```
(%i8) wxplot2d([1-f(x)], [x,0,5]);
```

(%t8)



(%o8)

```
(%i9) f(x) := %e^-x;
```

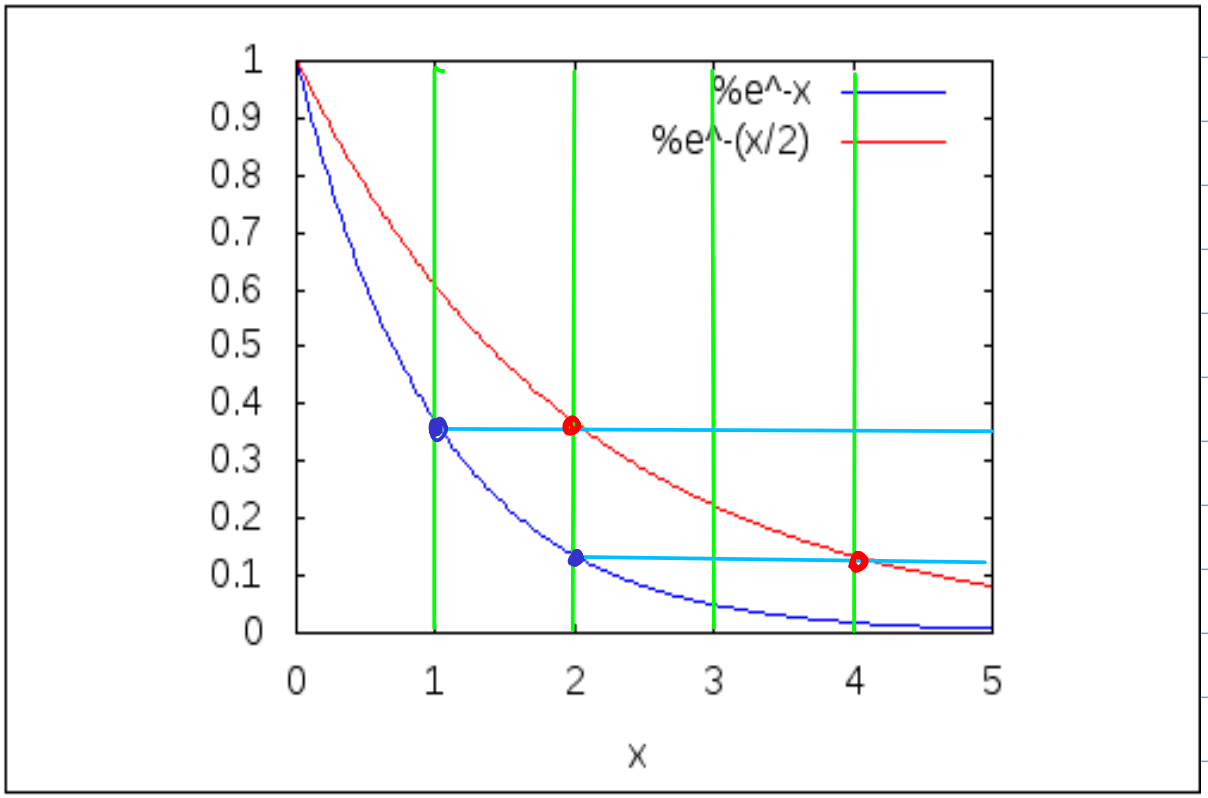
(%o9) $f(x) := \%e^{-x}$

```
(%i10) g(x) := %e^(-x/2);
```

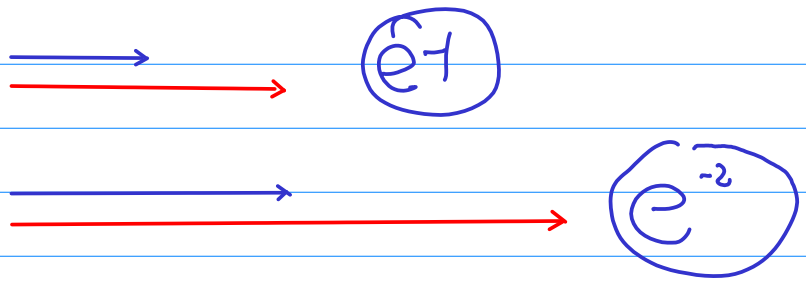
(%o10) $g(x) := \%e^{\frac{-x}{2}}$

```
(%i11) wxplot2d([f(x), g(x)], [x,0,5]);
```

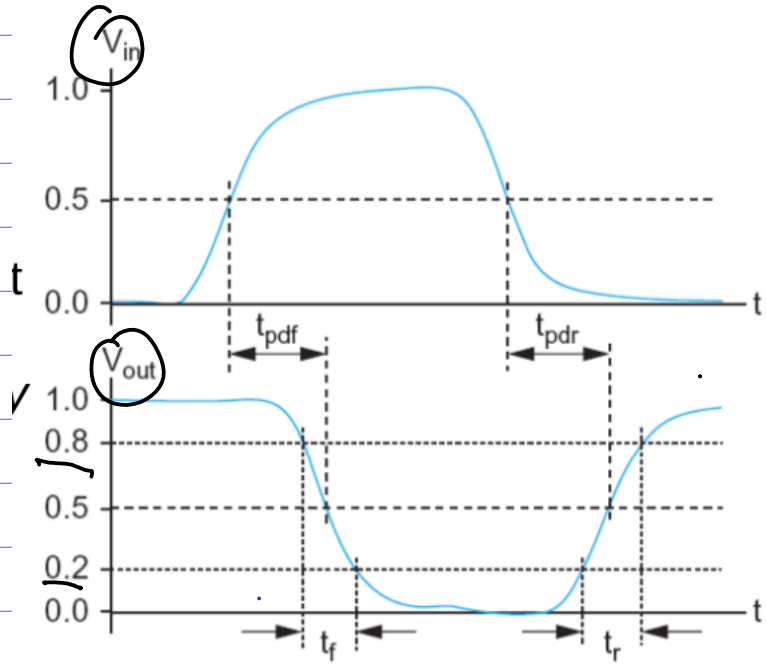
(%t11)

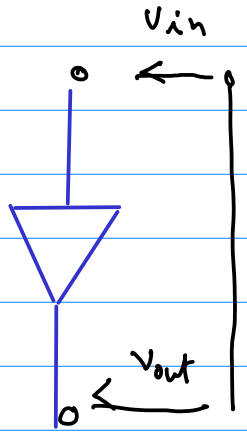


(%o11)

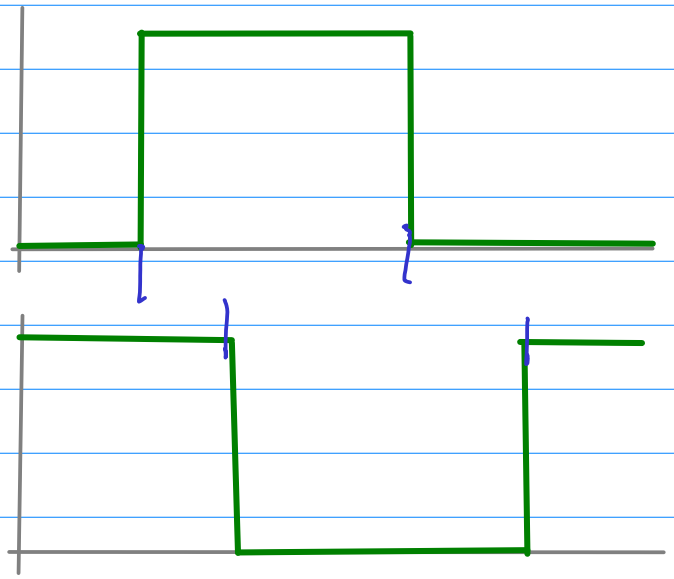


Spice Simulation : accurate, long time to simulate



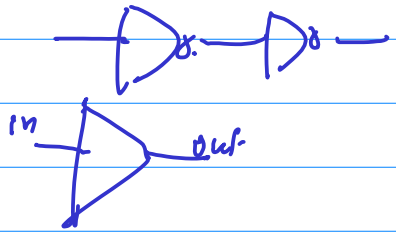


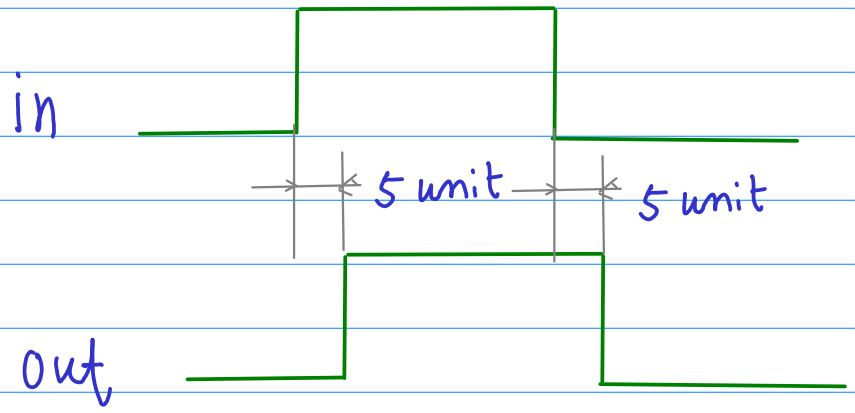
Verilog Simulation : Delay estimate, fast to simulate



Single delay

```
buf #(5) (out,in);
```





fall delay
= rise delay

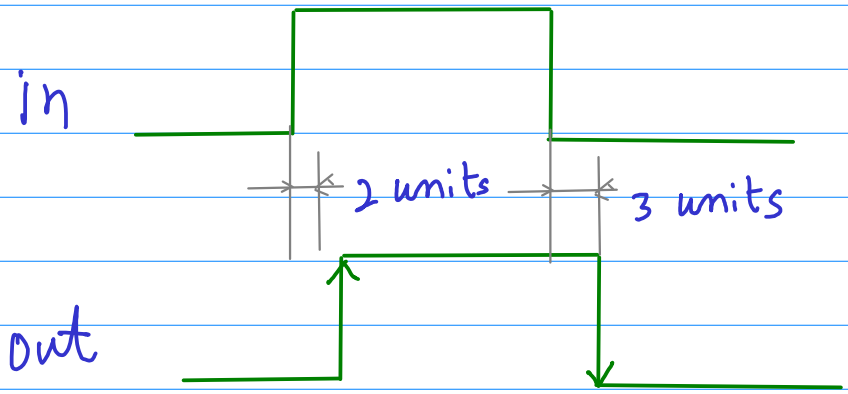
two delay

annotation

```
buf #(2,3) (out,in);
```

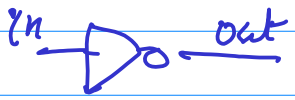
rise fall

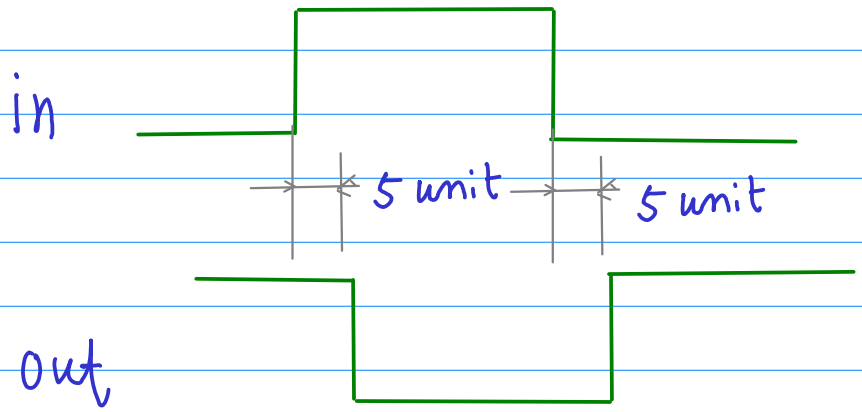
spf



Single delay

```
not #(5) (out,in);
```



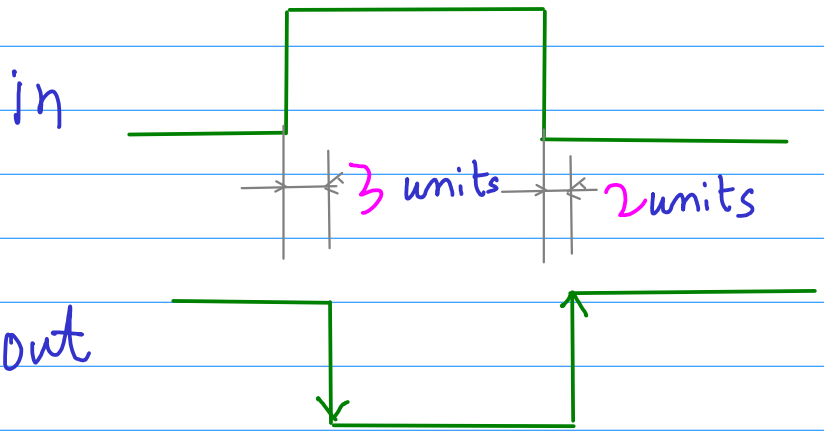


fall delay
= rise delay

two delay

```
not #(2,3) (out,in);
```

↑ rise ↑ fall



```
// Delay for all transitions
or     #5                u_or    (out1,b,c);

// Rise and fall delay
and    #(1,2)            u_and   (out2,b,c);


// Rise, fall and turn off delay
nor    #(1,2,3)          u_nor   (out3,b,c);


//One Delay, min, typ and max
nand   #(1:2:3)          u_nand  (out4,b,c);


//Two delays, min,typ and max
buf    #(1:4:8,4:5:6)     u_buf   (out5,b);


//Three delays, min, typ, and max
notif1 #(1:2:3,4:5:6,7:8:9) u_notif1 (out6,b,c);
```

1ns for a one time unit

```
`timescale 1ns/100ps

module adder_tb;

reg signed [3:0] i, j, k, flag;

wire signed [3:0] S;


adder4 A4 (i, j, 1'b0, Co, S);


endmodule
```

0.1 x 1ns
= 0.1ns

adder4.v

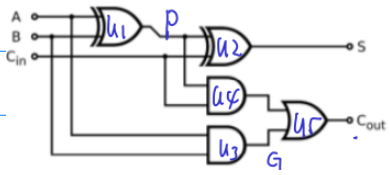
```
module adder4(A, B, Ci, Co, S);
  input [3:0] A, B;
  input Ci;

  output Co;
  output [3:0] S;

  wire[3:1] C;

  fa  fa0 (A[0], B[0], Ci,   C[1], S[0]);
  fa  fa1 (A[1], B[1], C[1], C[2], S[1]);
  fa  fa2 (A[2], B[2], C[2], C[3], S[2]);
  fa  fa3 (A[3], B[3], C[3], Co,   S[3]);


endmodule
```

fa_gate.v

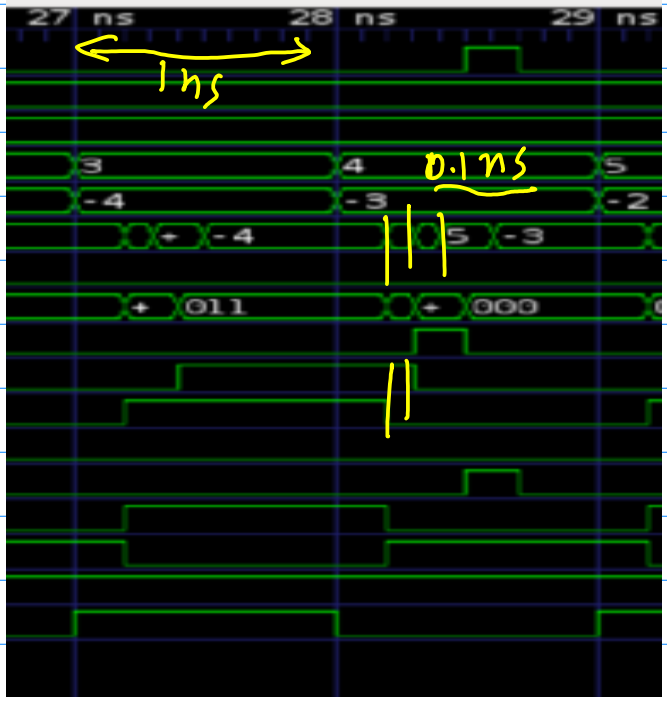
```
module fa(a, b, c, Co, S);
  input a, b, c;
  output Co, S;

  wire P, S, G, PC;

  xor #0.1 U1 (P, a, b);
  xor #0.1 U2 (S, P, c);
  and #0.1 U3 (G, a, b);
  and #0.1 U4 (PC, P, c);
  or  #0.1 U5 (Co, G, PC);
endmodule
```

fa_flow.v

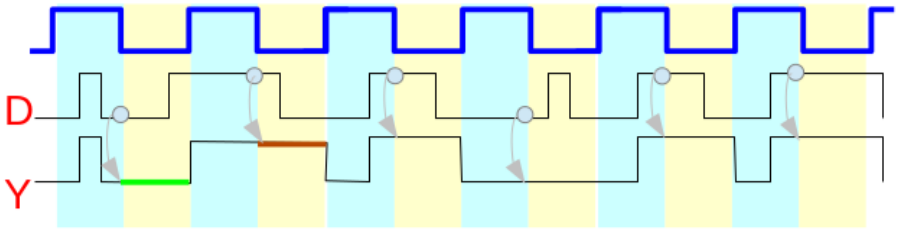
```
module fa(a, b, c, Co, S);
  input a, b, c;
  output Co, S;

  wire P, S, G, PC;

  assign #0.1 P = a ^ b;
  assign #0.1 S = P ^ c;
  assign #0.1 G = a & b;
  assign #0.1 PC = P & c;
  assign #0.1 Co = G | PC;
endmodule
```

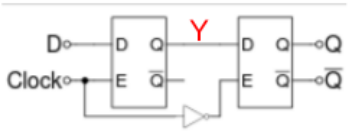


Delay

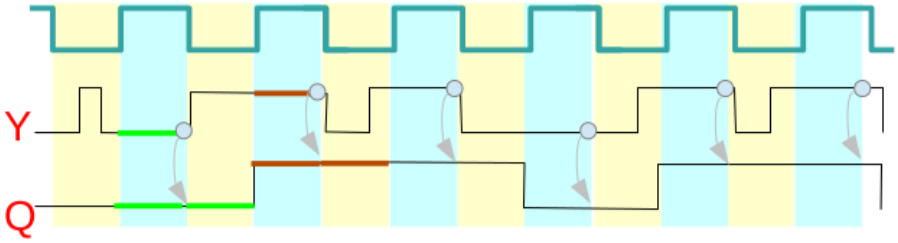


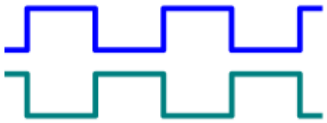
Master D Latch



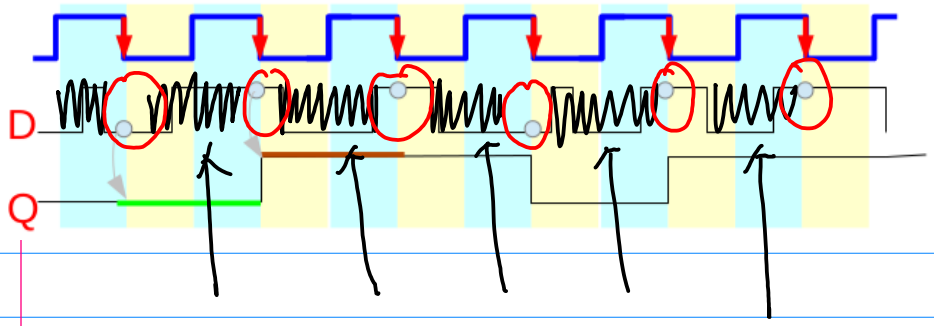


Slave D Latch





Master-Slave D F/F



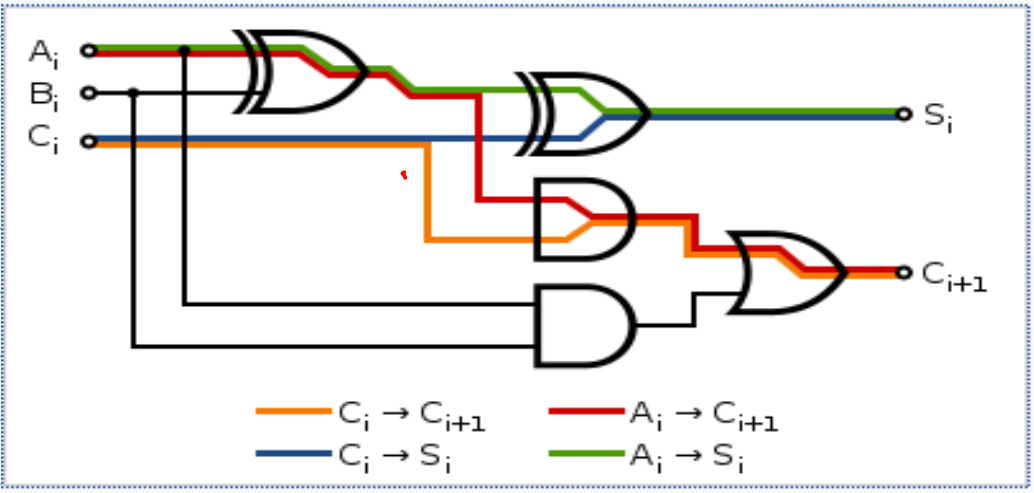
the hold output of the master is transparently reaches the output of the slave

this value is held for another half period

input D can have any arbitrary values

if the correct values are present at the negative edge of the clock

Gate Delay in RTL simulations

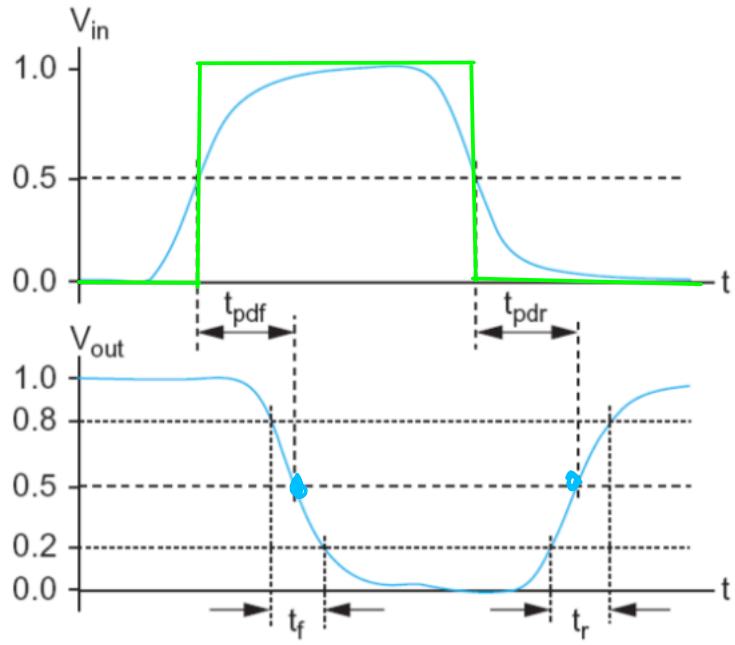


A full adder has an overall gate delay of 3 logic gates from the inputs *A* and *B* to the carry output C_{out} shown in red

In electronics, digital circuits and digital electronics, the propagation delay, or **gate delay**, is the length of time which starts when the input to a logic gate becomes stable and valid to change, to the time that the output of that logic gate is stable and valid to change. Often on manufacturers' datasheets this refers to the time required for the output to reach 50% of its final output level when the input changes to 50% of its final input level. Reducing gate delays in digital circuits allows them to process data at a faster rate and improve overall performance.

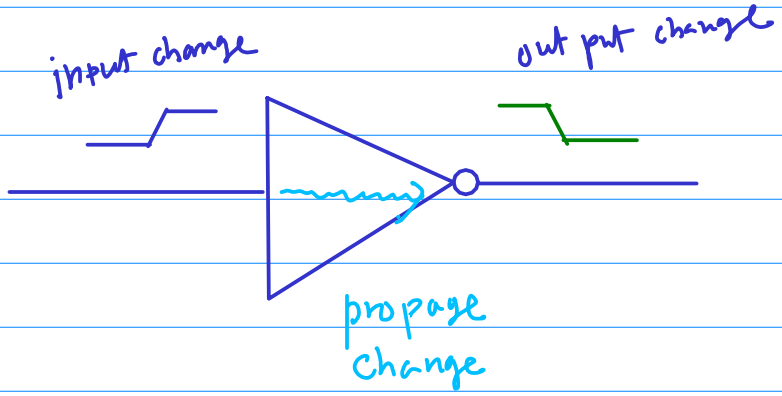
wikipedia.org

Propagation Delay in Spice Simulations



$$t_f = 2.2\,\tau_n$$

$$t_r = 2.2\,\tau_p$$



to estimate the reaction delay time from input to output

★ if input is ideal square shape,

$$t_p = \frac{1}{2}\left(t_{pf} + t_{pr}\right) = 0.35\left(\tau_n + \tau_p\right)$$

$$\tau_n = R_n C_{out}$$

$$\tau_p = R_p C_{out}$$

Delay	Eample
Absolute Delay #5	
Min, Typ, Max	#(4 : 6 : 8)
Risign, Falling	#(4, 6)
Rising, Falling, Turnoff	#(4, 6, 8)
	#(4:6:8, 5:7:9)

(min : typ : max)

(tr , tf , toff)

(4:6:8 , 5:7:9)

tr (min : typ : max)

tf (min : typ : max)

Verilog Path Delay Modeling : specify

specify block with path delay statements

Single-path
- Parallel connection : (=>)
- Full connection : (*>)

Edge-sensitive path : from clock to q
- posedge clock : (+:)
- negedge clock : (-:)

Level-sensitive path : (:)

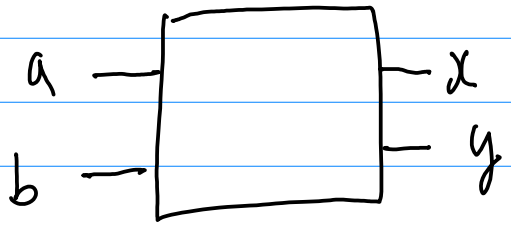
State-dependent path
- if (cond) simple_path
- if (cond) edge_sensitive_path
- ifnone simple_path

specparam statement
- to define parameters inside the specify block

http://www.wiley.com/legacy/wileychi/mblin/supp/student/LN07AdvancedModeling.pdf

```
module my_nor (a, b, out);
...
output out;

nor nor1 (out, a, b);

specify
      specparam      tr   = 1,  tf   = 2;
      specparam      tr_n = 2,   tf_n = 3;

      if (a)      (b => out)      = (tr,      tf);
      if (b)      (a => out)      = (tr,      tf);
      if (~a)     (b => out)      = (tr_n,    tf_n);
      if (~b)     (a => out)      = (tr_n,    tf_n);

endspecify
```

Simple path



Chosen paths

parallel connection

(a => x) = 5;

(a => y) = 5;

⋮

All paths

Full connection

(a, b *> x) = 6;

(a, b *> y) = 7;

delay info

Verilog Timing Checks

$setup
$hold
$setuphold
$width
$skew
$period
$recovery

```
sepcify
        $setup (D, posedge CK, 15);
        $hold   (posedge CK, D, 8);
endspecify
```