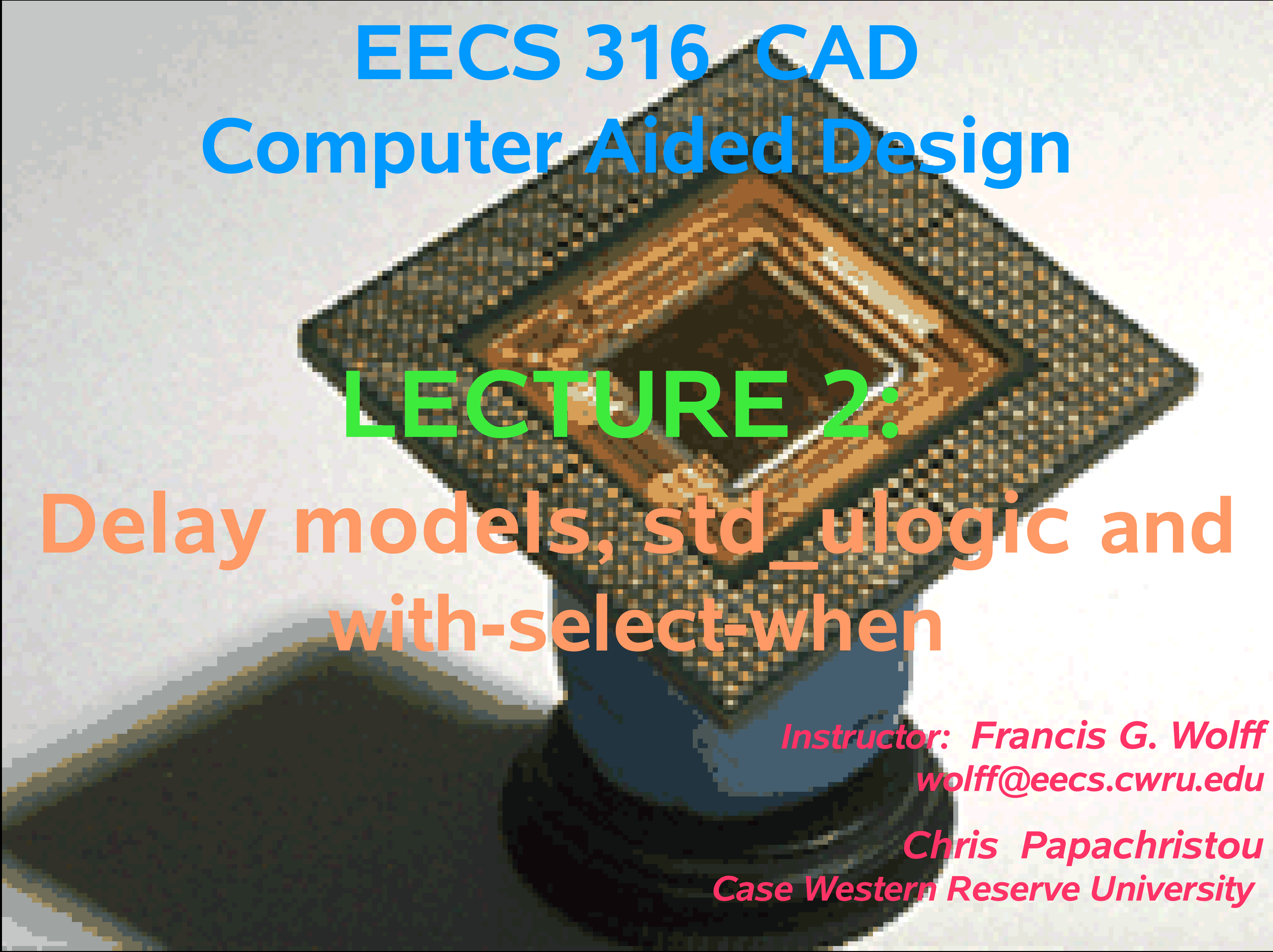


EECS 316 CAD Computer Aided Design

LECTURE 2:

Delay models, std_ulogic and with-select-when

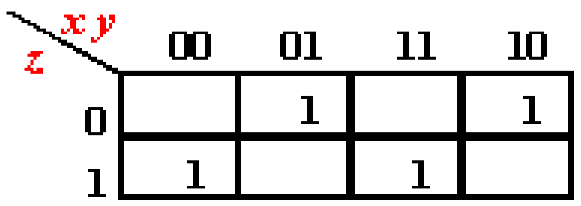
Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Chris Papachristou
Case Western Reserve University

Review: Full Adder: Truth Table

- A *Full-Adder* is a *Combinational circuit* that forms the arithmetic sum of three input bits.
- It consists of three inputs (z, x, y) and two outputs (*Carry*, *Sum*) as shown.

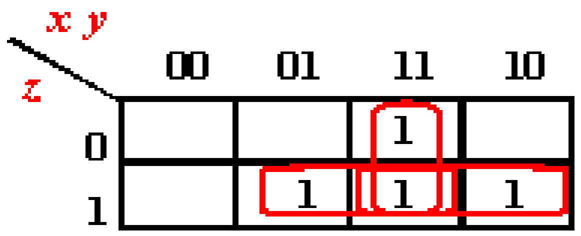
z	x	y	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth Table



$$\begin{array}{c|cccc} z \backslash xy & 00 & 01 & 11 & 10 \\ \hline 0 & & 1 & & 1 \\ 1 & 1 & & 1 & \end{array}$$

$$s = x \oplus y \oplus z$$



$$\begin{array}{c|cccc} z \backslash xy & 00 & 01 & 11 & 10 \\ \hline 0 & & & 1 & \\ 1 & & 1 & 1 & 1 \end{array}$$

$$c = xy + xz + yz = xy + z(x \oplus y)$$

Karnaugh maps

Review: Full Adder: Archite

Entity Declaration

```
ENTITY full_adder IS
    PORT (x, y, z:          IN std_logic;
               Sum, Carry:  OUT std_logic
); END full_adder;
```

Optional Entity END name;

Architecture Declaration

```
ARCHITECTURE full_adder_arch_1 OF  full_adder  IS

BEGIN

        Sum <= ( ( x XOR y ) XOR z );

        Carry <= (( x AND y ) OR (z AND (x AND y)));

END full_adder_arch_1;
```

Optional Architecture END name;

Review: SIGNAL: Scheduled Event

- SIGNAL

  Like variables in a programming language such as C, signals can be assigned values, e.g. 0, 1

- However, SIGNALs also have an associated time value

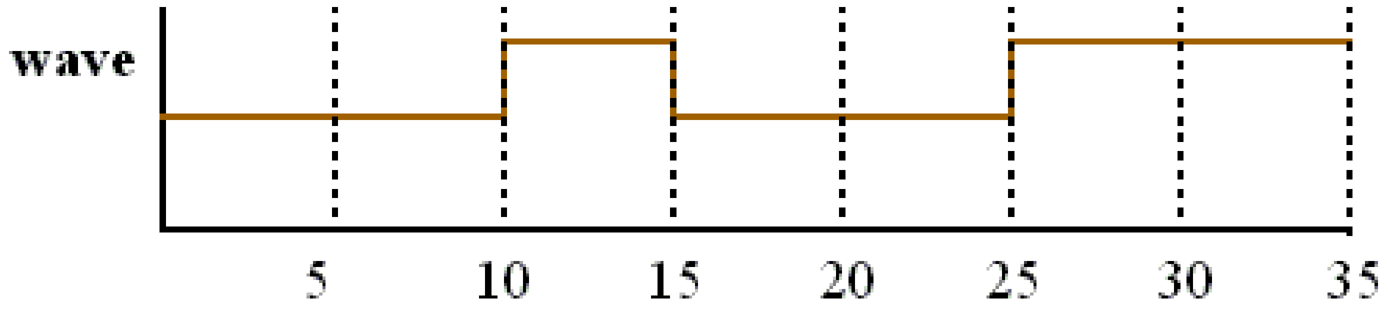
  A signal receives a value at a specific point in time and retains that value until it receives a new value at a future point in time (i.e. scheduled event)

- The waveform of the signal is

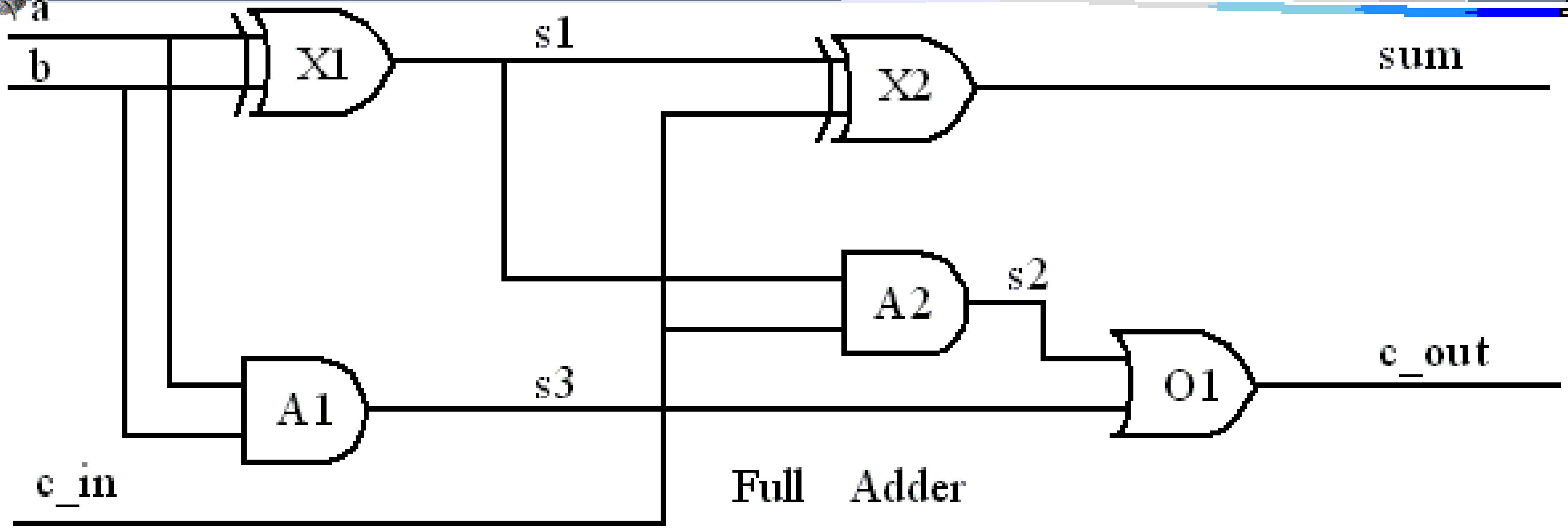
  a sequence of values assigned to a signal over time

- For example

  wave <= ‘0’, ‘1’ after 10 ns, ‘0’ after 15 ns, ‘1’ after 25 ns;



Review: Full Adder: Architecture with Delay



```
ARCHITECTURE full_adder_arch_2 OF full_adder IS
        SIGNAL S1, S2, S3: std_logic;
BEGIN
        s1     <= ( a XOR b )      after 15 ns;
        s2     <= ( c_in AND s1 ) after 5 ns;
        s3     <= ( a AND b )      after 5 ns;
        Sum   <= ( s1 XOR c_in ) after 15 ns;
        Carry <= ( s2 OR s3 )     after 5 ns;
END;
```

Signals (like wires) are not PORTs they do not have direction (i.e. IN, OUT)

Signal order: Does it matter? No

```
ARCHITECTURE full_adder_arch_2 OF  full_adder  IS
        SIGNAL S1, S2, S3: std_logic;
BEGIN
        s1     <= ( a XOR b )        after 15 ns;
        s2     <= ( c_in AND s1 ) after 5 ns;
        s3     <= ( a AND b )        after 5 ns;
        Sum   <= ( s1 XOR c_in ) after 15 ns;
        Carry <= ( s2 OR s3 )       after 5 ns;
END;
```

```
ARCHITECTURE full_adder_arch_3 OF  full_adder  IS
        SIGNAL S1, S2, S3: std_logic;
BEGIN
        Carry <= ( s2 OR s3 )       after 5 ns;
        Sum   <= ( s1 XOR c_in ) after 15 ns;
        s3     <= ( a AND b )        after 5 ns;
        s2     <= ( c_in AND s1 ) after 5 ns;
        s1     <= ( a XOR b )        after 15 ns;
END;
```

No, this is not C!

Net-lists have same behavior & parallel

Delta Delay

- Default signal assignment propagation delay if no delay is explicitly prescribed
  - VHDL signal assignments do not take place immediately
  - Delta is an infinitesimal VHDL time unit so that all signal assignments can result in signals assuming their values at a future time
  - E.g.

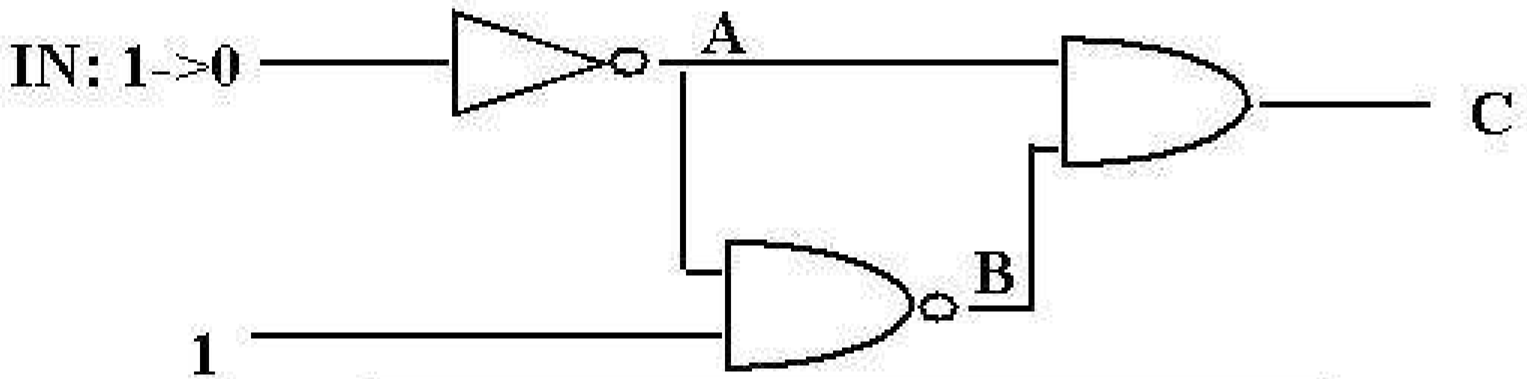
```
Output <= NOT Input;
-- Output assumes new value in one delta cycle
```

- Supports a model of concurrent VHDL process execution
  - Order in which processes are executed by simulator does not affect simulation output

Delta Delay

An Example with Delta Delay

- What is the behavior of C?



Using delta delay scheduling

Time	Delta	Event
0 ns	1	IN: 1->0 eval INVERTER
	2	A: 0->1 eval NAND, AND
	3	B: 1->0 C: 0->1 eval AND
	4	C: 1->0
1 ns		

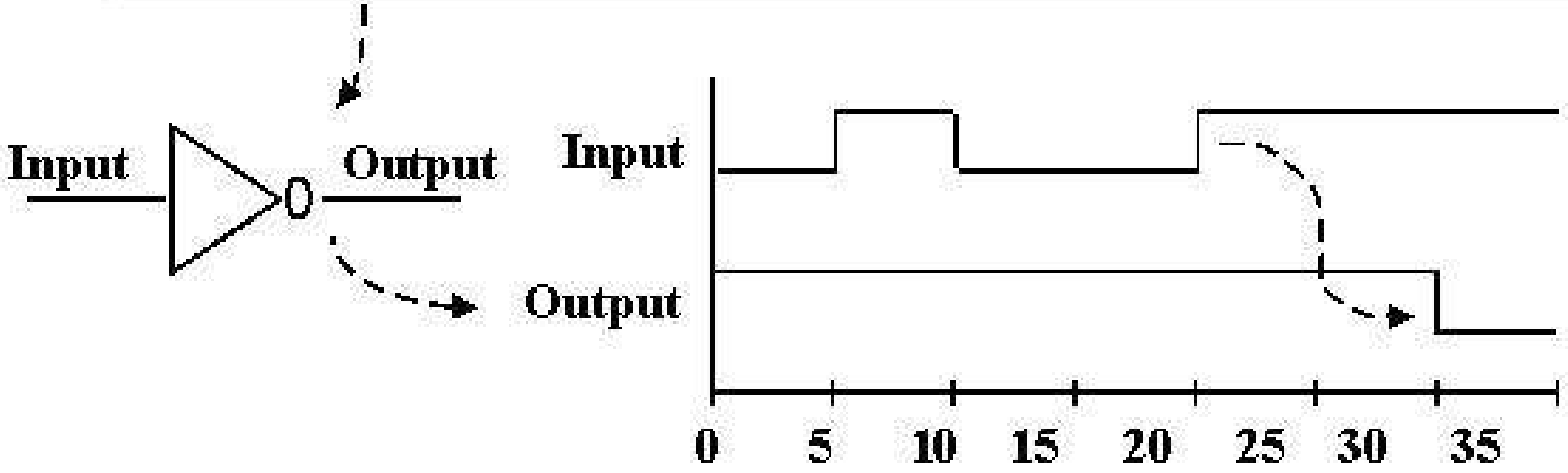
Inertial Delay

- Provides for specification propagation delay and input pulse width, i.e. ‘inertia’ of output:

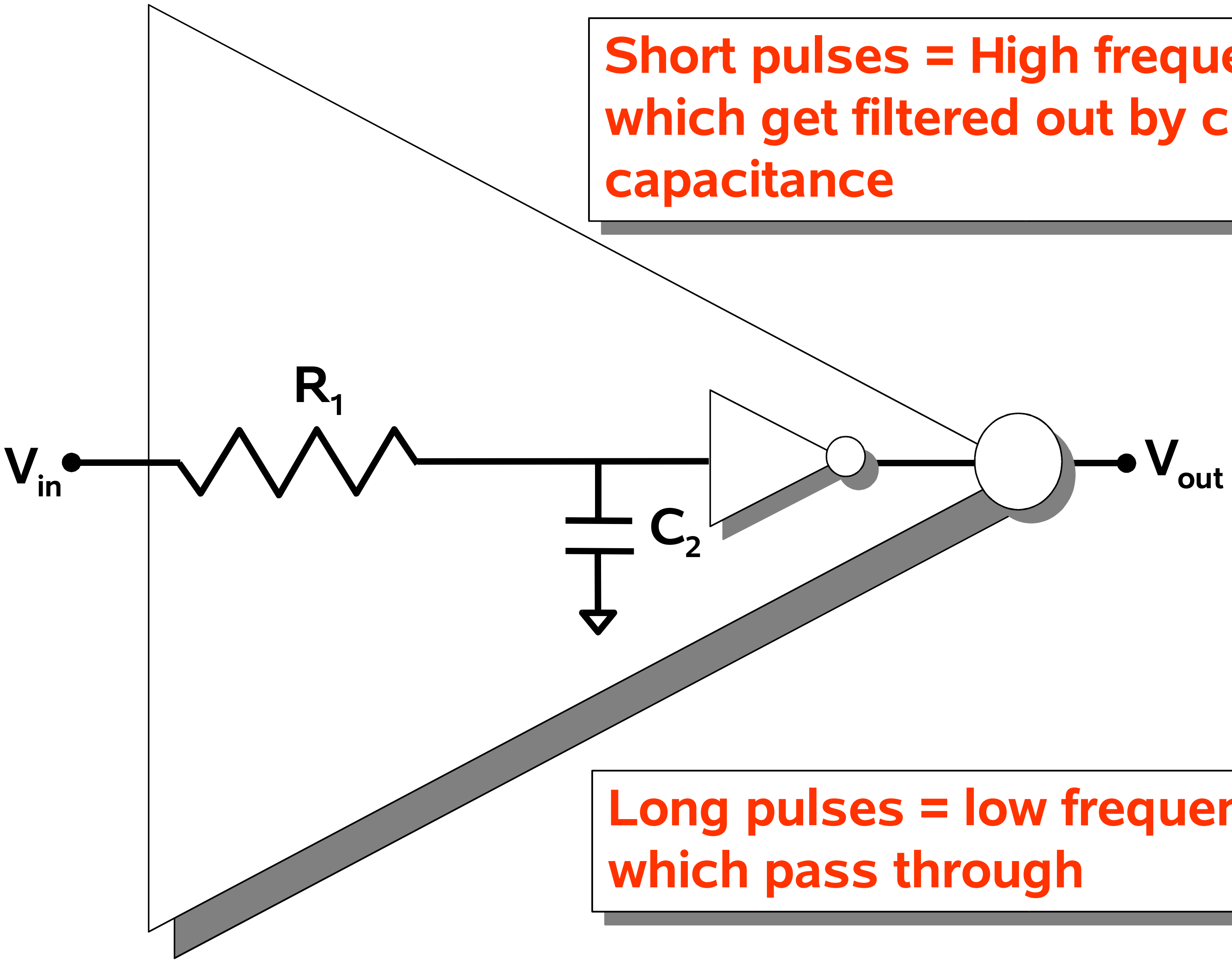
```
target <= [REJECT time_expression] INERTIAL waveform;
```

- Inertial delay is default and REJECT is optional :

```
Output <= NOT Input AFTER 10 ns;
-- Propagation delay and minimum pulse width are 10ns
```



Inverter model: lowpass filter (inertial)



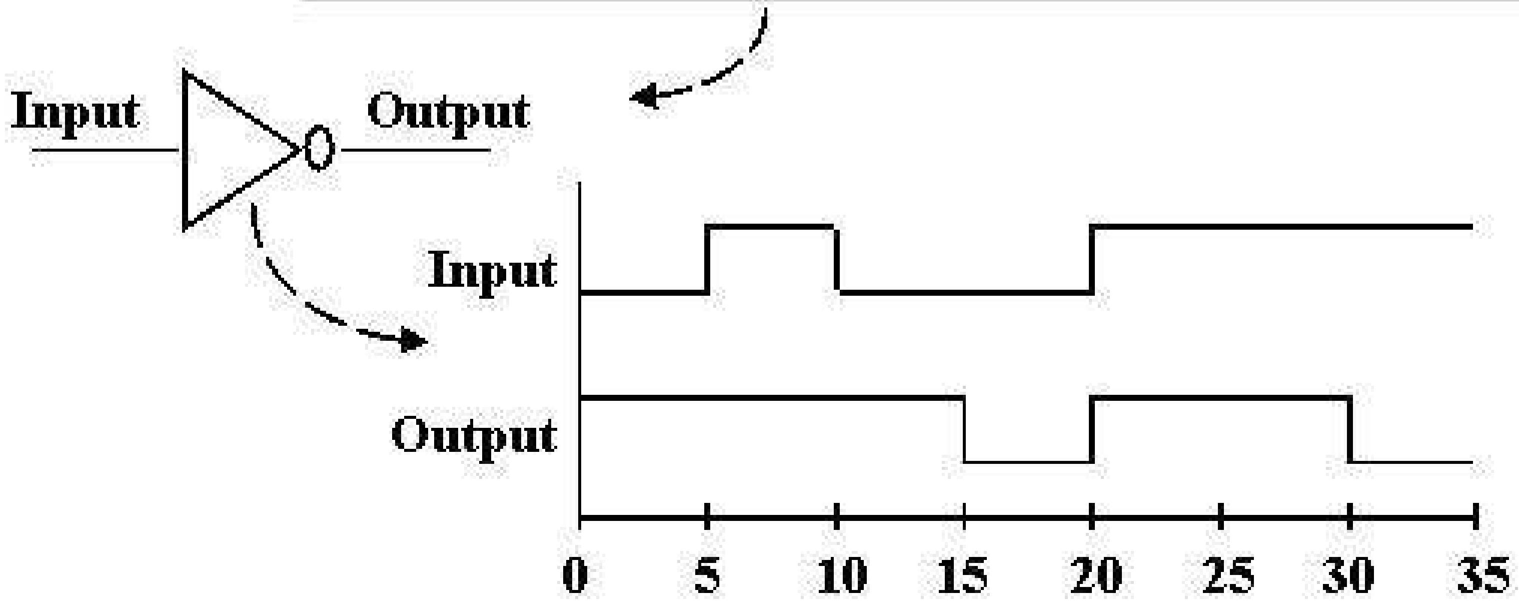
Short pulses = High frequency which get filtered out by cmos capacitance

Long pulses = low frequency which pass through

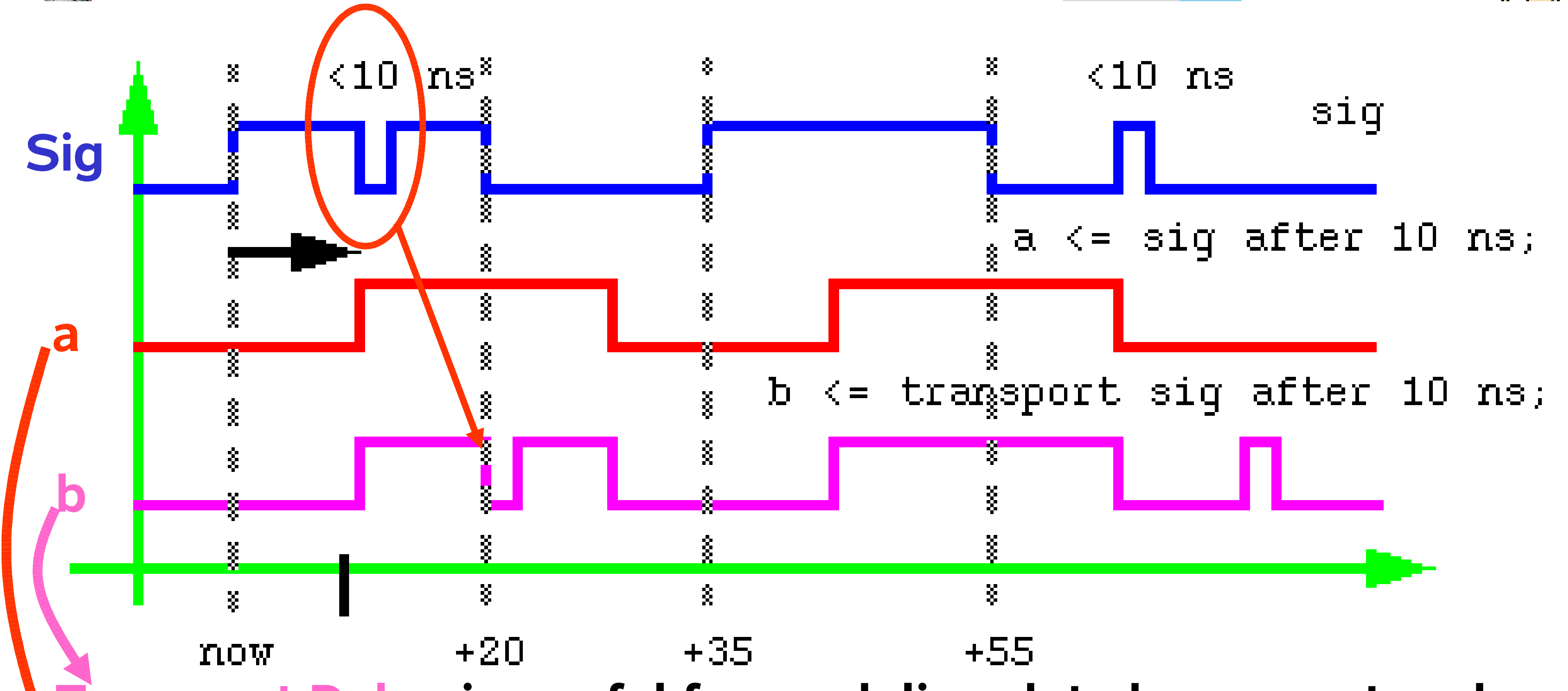
Transport Delay

- Transport delay must be explicitly specified
  - I.e. keyword "TRANSPORT" must be used
- Signal will assume its new value after specified delay

```
-- TRANSPORT delay example
Output <= TRANSPORT NOT Input AFTER 10 ns;
```



Inertial and Transport Delay



Transport Delay is useful for modeling data buses, networks

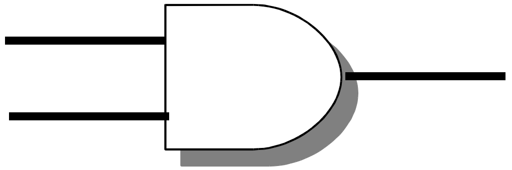
Inertial Delay is useful for modeling logic gates

Combinatorial Logic Operators

#Transistors		
2	NOT	z <= NOT (x); z<= NOT x;
2+2*i*	AND	z <= x AND y;
2*i*	NAND	z <= NOT (x AND y);
2+2*i*	OR	z <= x OR y;
2*i*	NOR	z <= NOT (x OR Y);
10	XOR	z <= (x and NOT y) OR (NOT x AND y); *z <= (x AND y) NOR (x NOR y); --AOI*
12	XNOR	z <= (x and y) OR (NOT x AND NOT y); *z <= (x NAND y) NAND (x OR y); --OAI*

Footnote: (i=#inputs) *We are only referring to CMOS static transistor ASIC gate designs*
Exotic XOR designs can be done in 6 (J. W. Wang, IEEE J. Solid State Circuits, 29, July 1994)

Std_logic AND: Un-initialized value



NOT	0	1	U
	1	0	U

AND	0	1	U
0	0	0	0
1	0	1	U
U	0	U	U

OR	0	1	U
0	0	1	U
1	1	1	1
U	U	1	U

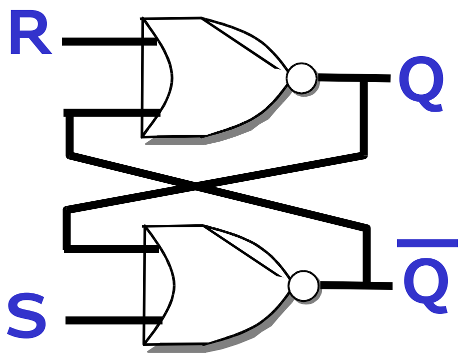
0 AND <anything> is 0

0 NAND <anything> is 1

1 OR <anything> is 1

1 NOR <anything> is 0

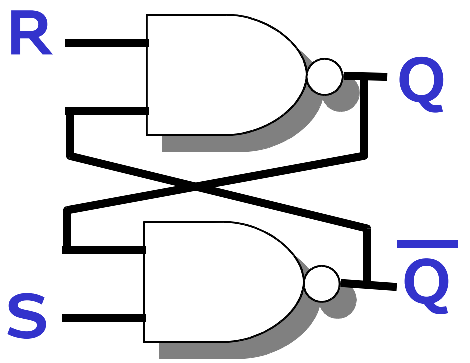
SR Flip-Flop (Latch)



NOR

R	S	Q_{n+1}
0	0	Q_n
0	1	1
1	0	0
1	1	U

```
Q  <= R NOR NQ;
NQ <= S NOR Q;
```



NAND

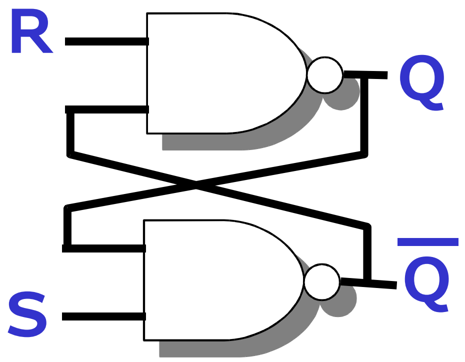
R	S	Q_{n+1}
0	0	U
0	1	1
1	0	0
1	1	Q_n

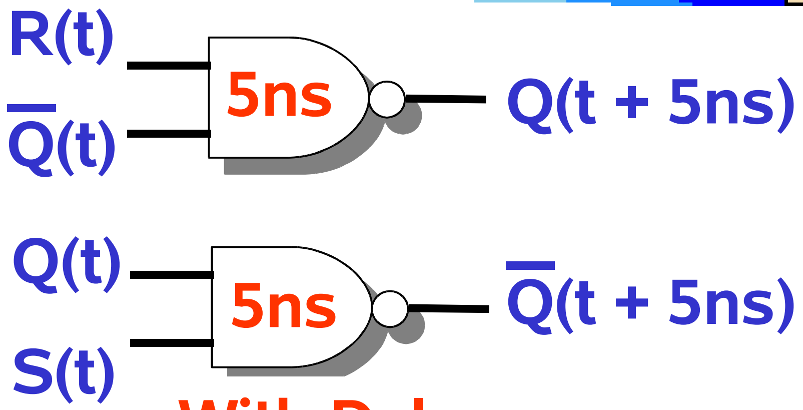
```
Q  <= R NAND NQ;
NQ <= S NAND Q;
```

SR Flip-Flop (Latch)

NAND

R	S	Q_{n+1}
0	0	U
0	1	1
1	0	0
1	1	Q_n



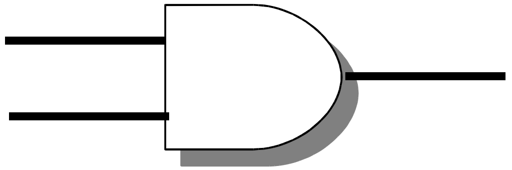


With Delay

Example: R <= ‘1’, ‘0’ after 10ns, ‘1’ after 30ns; S <= ‘1’;

t	0	5ns	10ns	15ns	20ns	25ns	30ns	35ns	40ns
R	1	1	0	0	0	0	1	1	1
$\overline{Q}$	U	U	U	U	0	0	0	0	0
Q	U	U	U	1	1	1	1	1	1
S	1	1	1	1	1	1	1	1	1

Std_logic AND: X Forcing Unknown Value



NOT	0	X	1	U
	1	X	0	U

AND	0	X	1	U
0	0	0	0	0
X	0	X	X	U
1	0	X	1	U
U	0	U	U	U

OR	0	X	1	U
0	0	X	1	U
X	X	X	1	U
1	1	1	1	1
U	U	U	1	U

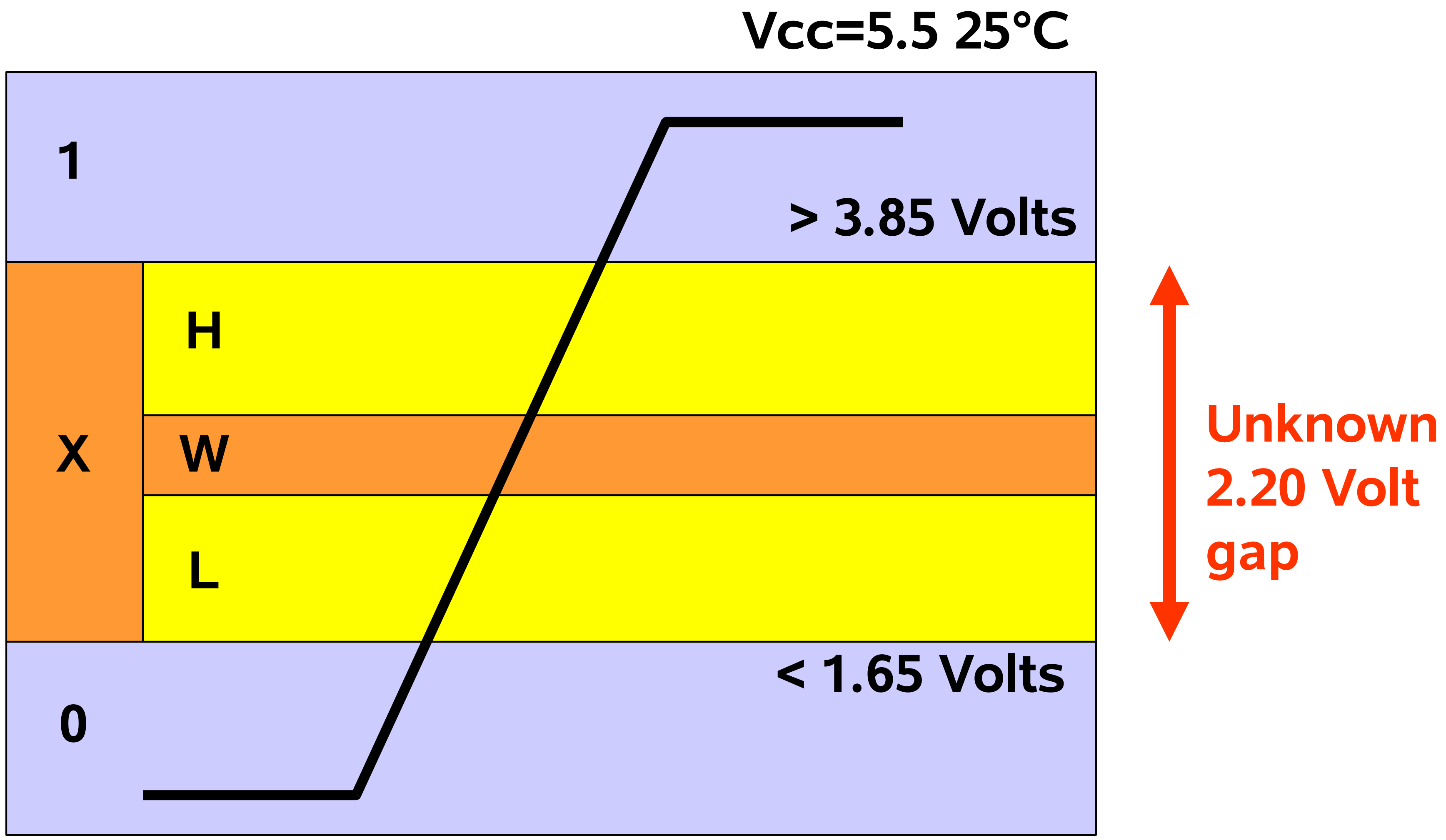
0 AND <anything> is 0

0 NAND <anything> is 1

1 OR <anything> is 0

0 NOR <anything> is 1

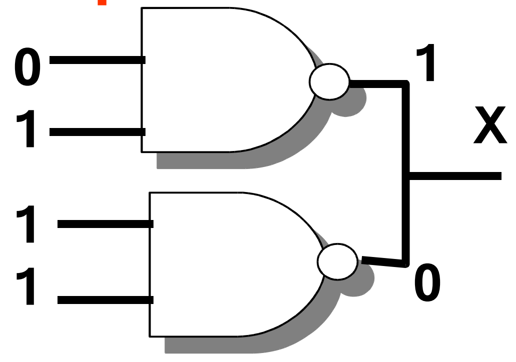
The rising transition signal



Modeling logic gate values: std_ulogic

```
TYPE std_ulogic IS ( -- Unresolved LOGIC
        'Z',    -- High Impedance (Tri-State)
        '1',    -- Forcing 1
        'H',    -- Weak 1
        'X',    -- Forcing Unknown: i.e. combining 0 and 1
        'W',    -- Weak Unknown: i.e. combining H and L
        'L',    -- Weak 0
        '0',    -- Forcing 0
        'U',    -- Un-initialized
        '-',    -- Don't care
);
```

Example:
multiple drivers



Multiple output drivers: Resolution Function

	U	X	0	L	Z	W	H	1	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	0	0	0	0	X	X
L							W	1	X
Z							H	1	X
W							W	1	X
H							H	1	X
1							1	1	X
-	U	X	X	X	X	X	X	X	X

Suppose that

the first gate outputs a 1

the second gate outputs a 0

then

the mult-driver output is X

X: forcing unknown value by combining 1 and 0 together

Multiple output drivers: Resolution Function

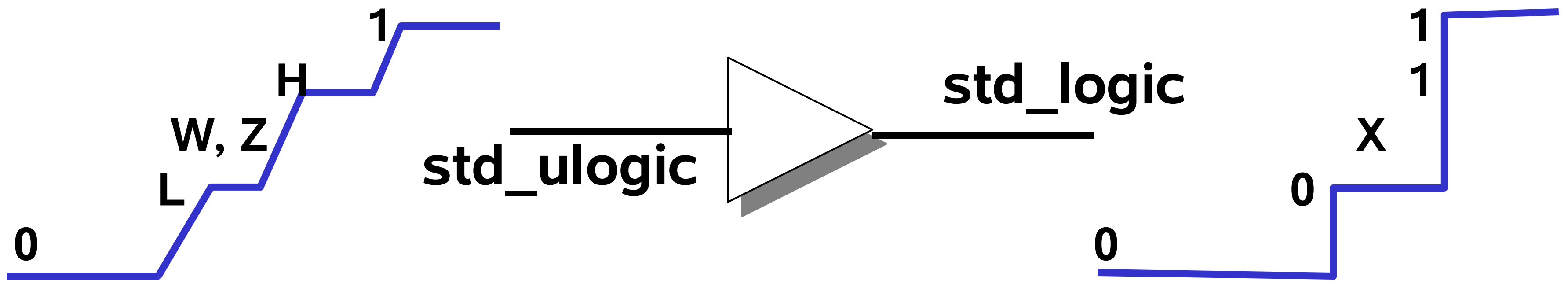
	U	X	0	L	Z	W	H	1	-
U	U	U	U	U	U	U	U	U	U
X		X	X	X	X	X	X	X	X
0			0	0	0	0	0	X	X
L				L	L	W	W	1	X
Z					Z	W	H	1	X
W						W	W	1	X
H							H	1	X
1								1	X
-									X

Observe that 0 pulls down all weak signals to 0

H <driving> L => W

- Note the multi-driver resolution table is symmetrical

Resolution Function: std_logic buffer gate



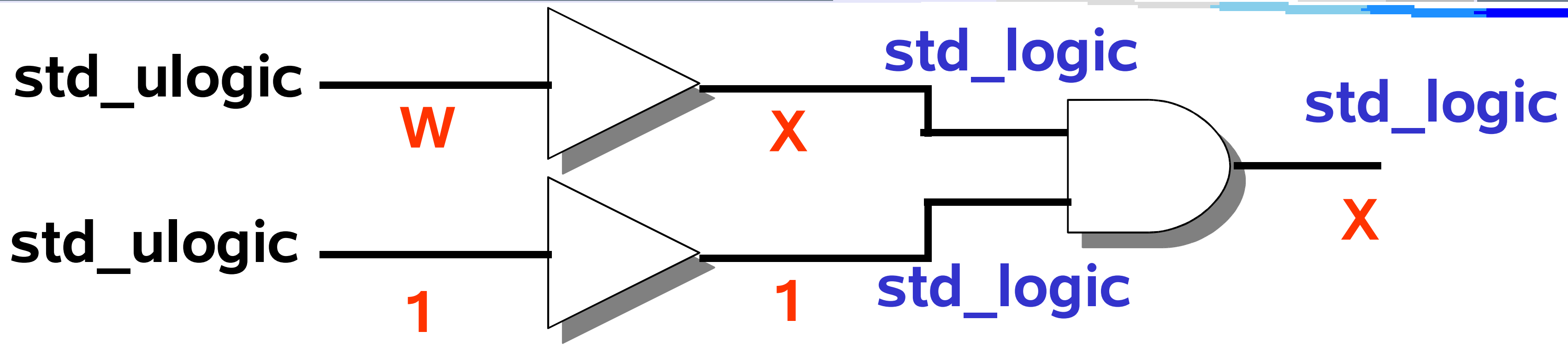
input:	U	0	L	W	X	Z	H	1	-
output:	U	0	0	X	X	X	1	1	X

0 or L becomes 0

H or 1 becomes 1

Transition zone becomes X

Resolving input: std_logic AND GATE



Process each input as an unresolved to resolved buffer.

Then process the gate as a standard logic gate { 0, X, 1, U }

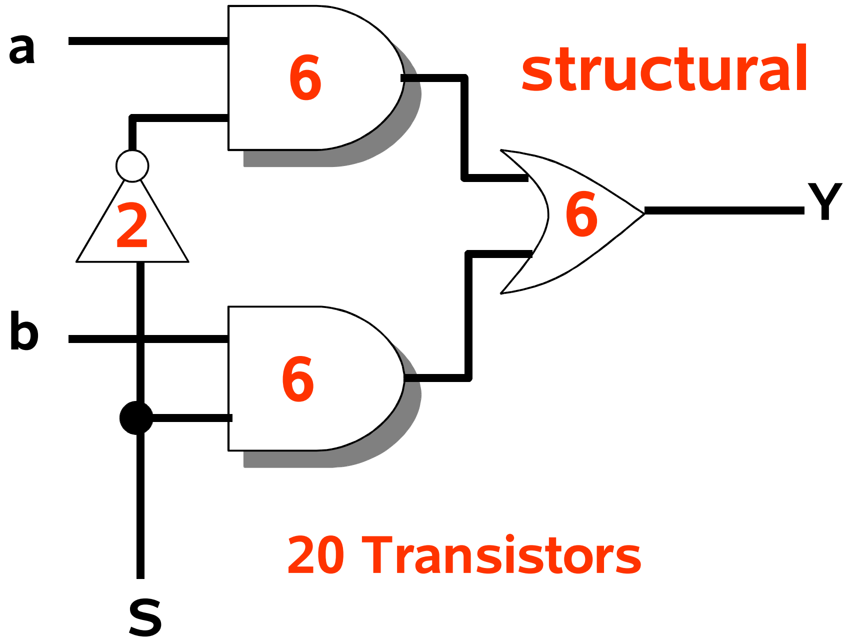
For example, let’s transform z <= ‘W’ AND ‘1’;

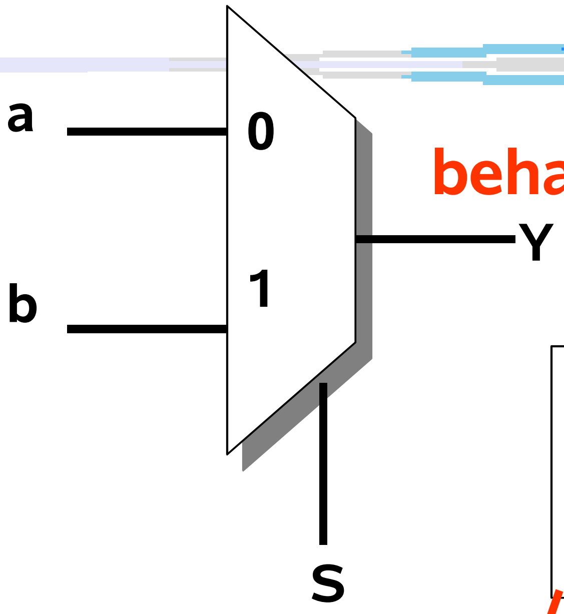
z <= ‘W’ AND ‘1’; -- convert std_ulogic ‘W’ to std_logic ‘X’

z <= ‘X’ AND ‘1’; -- now compute the std_logic AND

z <= ‘X’;

2-to-1 Multiplexor: with-select-when





combinatorial logic

```
Y <= (a AND NOT s)
        OR
     (b AND s);
```

Only values allowed

```
WITH s SELECT
    Y <= a WHEN '0',
         b WHEN '1';
```

or more general

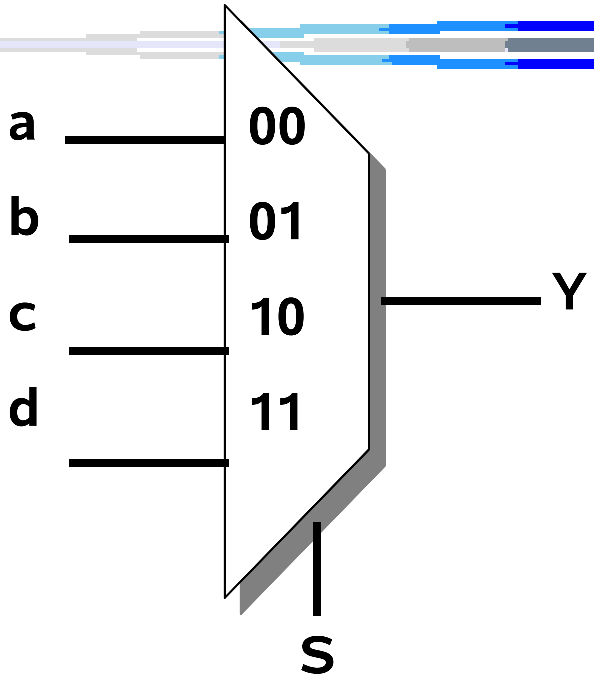
```
WITH s SELECT
    Y <= a WHEN '0',
         b WHEN OTHERS;
```

OTHERS includes 1,U,L,W,X,H,Z

4-to-1 Multiplexor: with-select-when

Structural Combinatorial logic

```
Y <= sa OR sb OR sc OR sd;

sa <= a AND ( NOT s(1) AND NOT s(0) );

sb <= b AND ( NOT s(1) AND s(0) );

sc <= c AND ( s(1) AND NOT s(0) );

sd <= d AND ( s(1) AND s(0) );
```



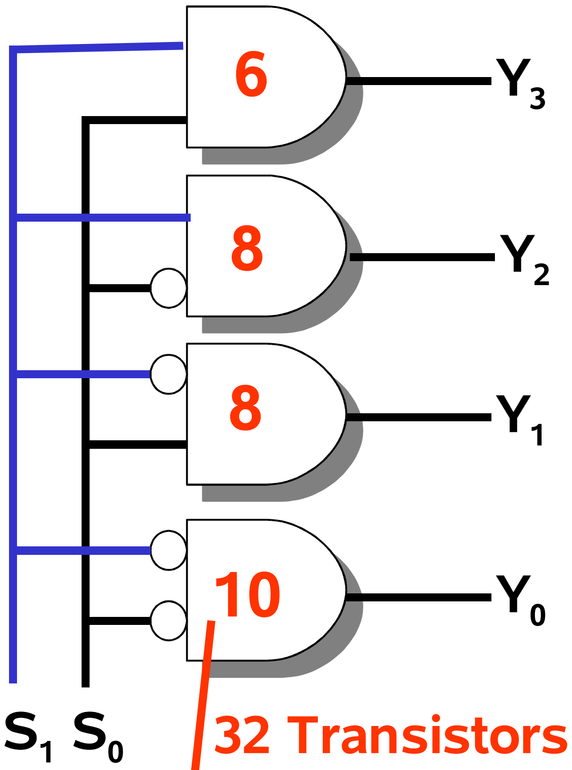
As the complexity of the combinatorial logic grows, the SELECT statement, simplifies logic design but at a loss of structural information

```
WITH s SELECT
  Y <= a WHEN "00",
       b WHEN "01",
       c WHEN "10",
       d WHEN OTHERS;
```

Note the comma after WHEN

behavioral

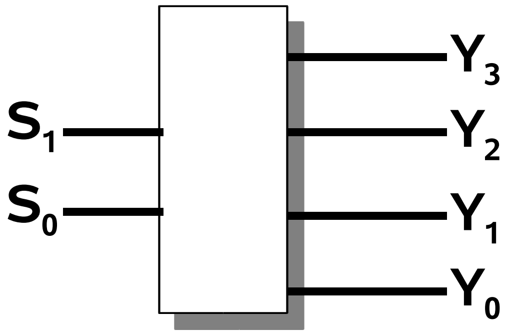
with-select-when: 2 to 4-line Decoder



32 Transistors

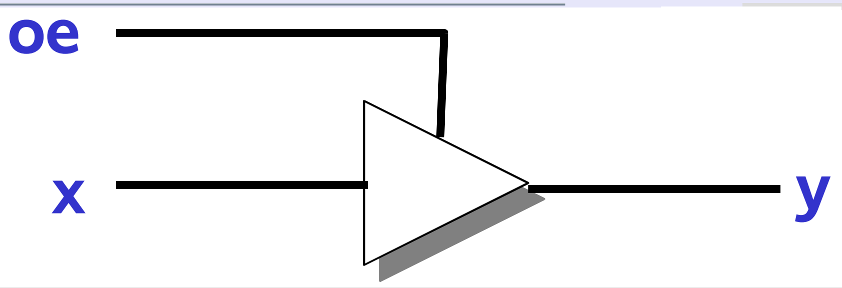
Replace this with a NOR, then 26 total transistors

```
SIGNAL S: std_logic_vector(1 downto 0);

SIGNAL Y: std_logic_vector(3 downto 0);
```



```
WITH S SELECT
      Y <= “1000” WHEN “11”,
           “0100” WHEN “10”,
           “0010” WHEN “01”,
           “0001” WHEN OTHERS;
```

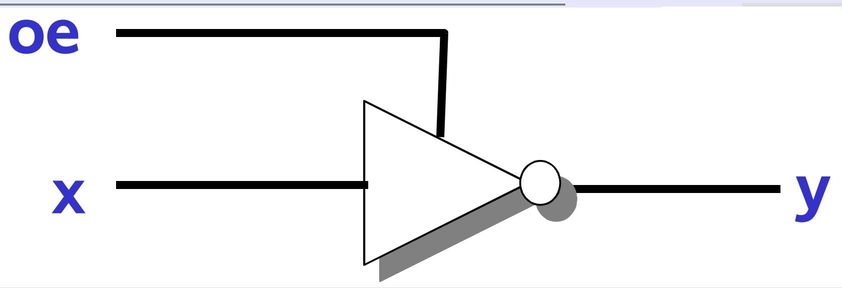
Tri-State buffer



```
ENTITY TriStateBuffer IS
          PORT(x:       IN      std_logic;
                    y:       OUT   std_logic;
                    oe:     IN      std_logic
); END;
```

```
ARCHITECTURE Buffer3 OF TriStateBuffer IS
BEGIN

         WITH oe SELECT
         y <=    x   WHEN '1',    -- Enabled:  y <= x;
                 'Z' WHEN OTHERS; -- Disabled: output a tri-state

END;
```

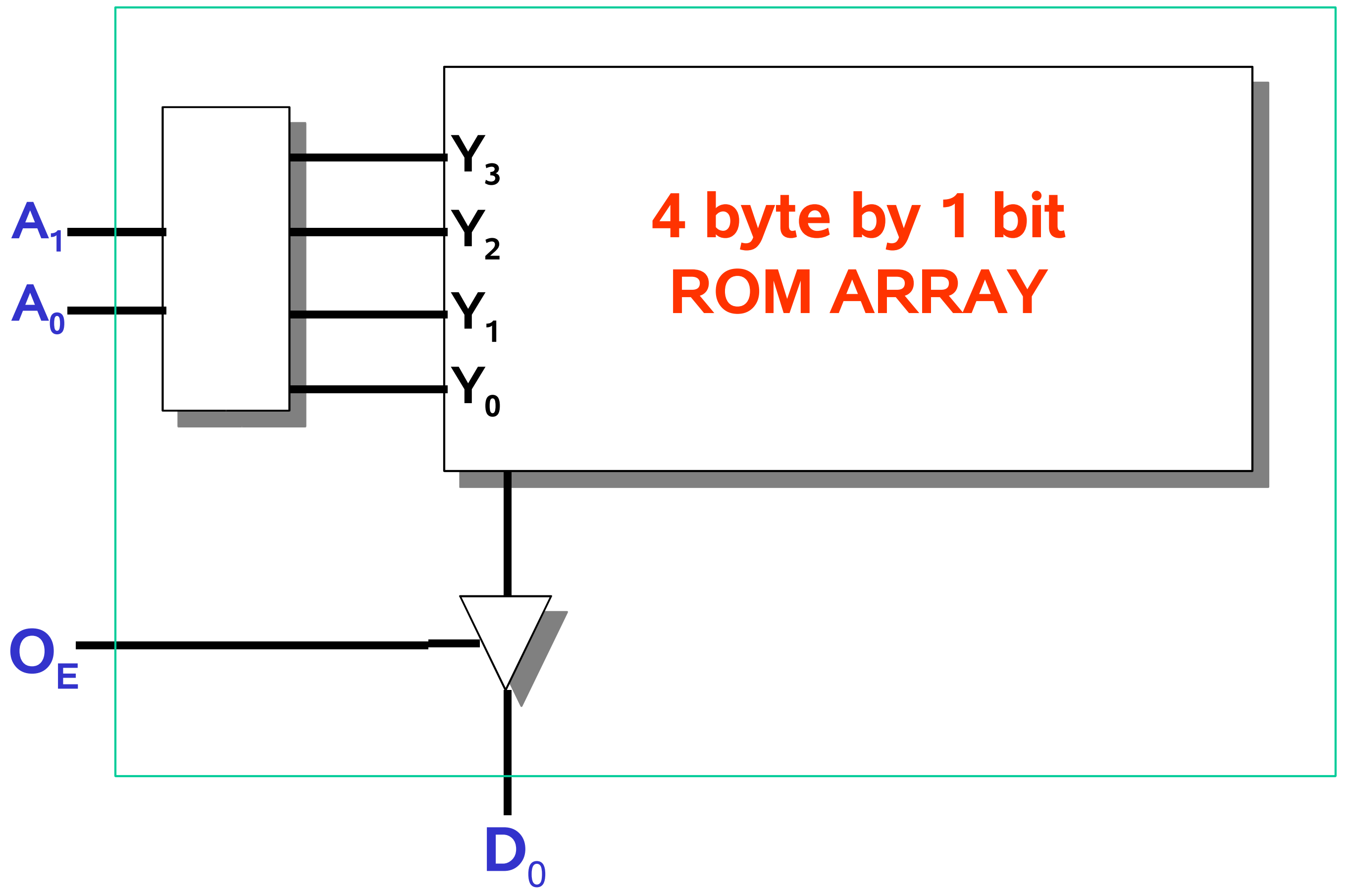
Inverted Tri-State buffer



```
ENTITY TriStateBufferNot IS
        PORT(x:      IN     std_logic;
             y:      OUT    std_logic;
             oe:     IN     std_logic
); END;
```

```
ARCHITECTURE Buffer3 OF TriStateBufferNot IS
BEGIN

        WITH oe SELECT
        y <=    NOT(x) WHEN '1',    -- Enabled:  y <= Not(x);
                'Z'    WHEN OTHERS; -- Disabled

END;
```

ROM: 4 byte Read Only Memory



ROM: 4 byte Read Only Memory

```
ENTITY rom_4x1 IS
        PORT(A:     IN std_logic_vector(1 downto 0);
             OE:    IN std_logic; -- Tri-State Output
             D:     OUT std_logic
); END;
```

```
ARCHITECTURE rom_4x1_arch OF rom_4x1 IS
  SIGNAL ROMout: std_logic;
BEGIN
  BufferOut: TriStateBuffer PORT MAP(ROMout, D, OE);
  WITH A SELECT
        ROMout <=  ‘1’ WHEN “00”,
                   ‘0’ WHEN “01”,
                   ‘0’ WHEN “10”,
                   ‘1’ WHEN “11”;
```

Component Instance

Component declaration name

Component Declaration/Instance relationship

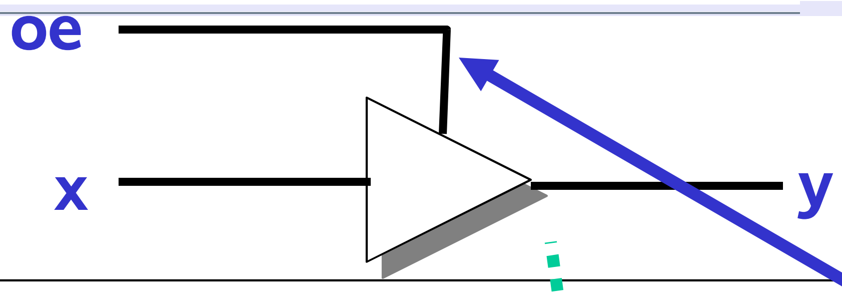
```
ARCHITECTURE rom_4x1_arch OF rom_4x1 IS

  COMPONENT TriStateBuffer
  PORT (x: IN std_logic; y: OUT std_logic, oe: IN std_logic);
  END COMPONENT;

  SIGNAL ROMout: std_logic;
BEGIN
   BufferOut: TriStateBuffer PORT MAP(ROMout, D, OE);

   WITH A SELECT
        ROMout <=   '1' WHEN "00",
                    '0' WHEN "01",
                    '0' WHEN "10",
                    '1' WHEN "11";

END;
```

Component Declaration

Colon (:) says make a Component Instance

Component Instance Name: *BufferOut*

Component Port relationship



OE → IN → oe → IN

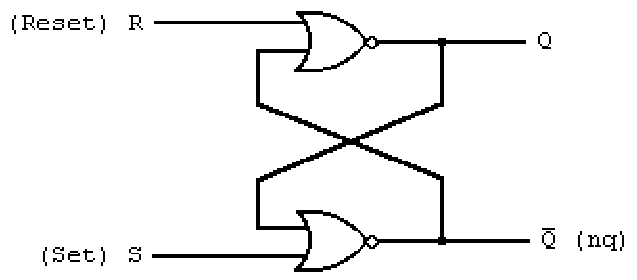
D → OUT → y → OUT

```
COMPONENT TriStateBuffer
  PORT (x: IN std_logic; y: OUT std_logic, oe: IN std_logic);
END COMPONENT;
```

```
BufferOut: TriStateBuffer     PORT MAP(ROMout, D, OE);
```

```
ENTITY rom_4x1 IS
        PORT(A:     IN std_logic_vector(1 downto 0);
             OE:    IN std_logic; -- Tri-State Output
             D:     OUT std_logic
); END;
```

Assignment #2 (Part 1 of 3)



SR Latch

1) Assume each gate is 5 ns delay for the above circuit.

(a) Write entity-architecture for a inertial model

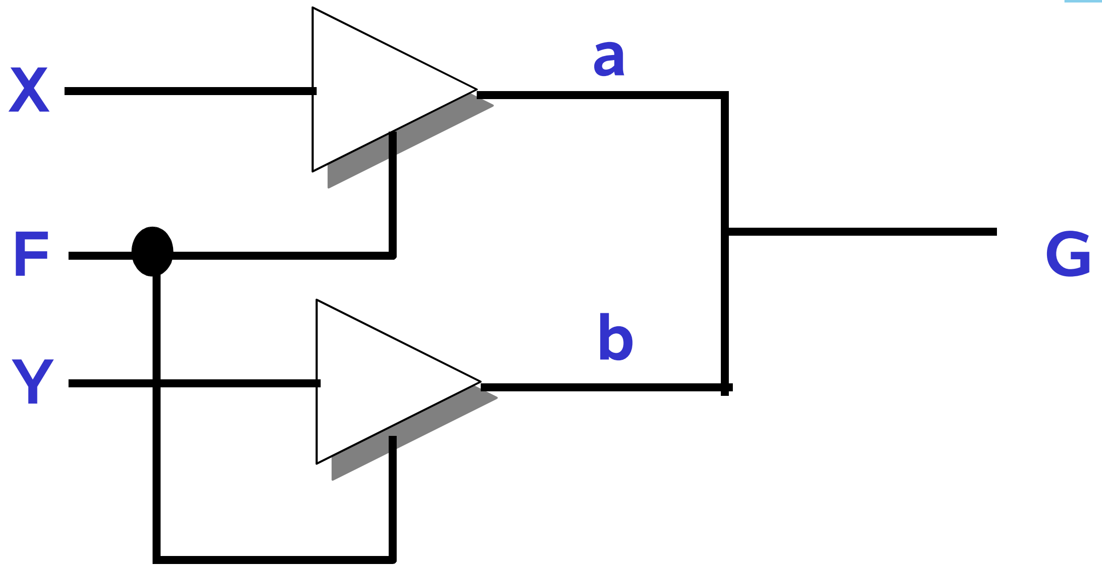
(b) Given the following waveform, draw, R, S, Q, NQ (inertial)
R <= ‘1’, ‘0’ after 25 ns, ‘1’ after 30 ns, ‘1’ after 50 ns;
S <= ‘0’, ‘1’ after 20 ns, ‘0’ after 35 ns, ‘1’ after 50 ns;

(c) Repeat (b) but now assume each gate is 20 ns delay

(d) Write entity-architecture for a transport model

(e) Given the waveform in (b) draw, R, S, Q, NQ (transport)

Assignment #2 (Part 2 of 3)



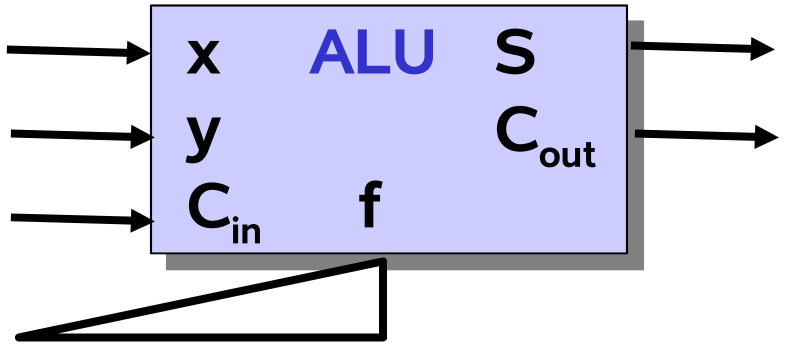
(2) Given the above two tri-state buffers connected together (assume transport model of 5ns per gate), draw X, Y, F, a, b, G for the following input waveforms:

X <= '1', '0' after 10 ns, 'X' after 20 ns, 'L' after 30 ns, '1' after 40 ns;
Y <= '0', 'L' after 10 ns, 'W' after 20 ns, '0' after 30 ns, 'Z' after 40 ns;
F <= '0', '1' after 10 ns, '0' after 50 ns;

Assignment #2 (Part 3 of 3)

3) Write (no programming) a entity-architecture for a 1-bit ALU. The input will consist of x, y, Cin, f and the output will be S and Cout. Make components for 1-bit add/sub. The input function f (with-select) will enable the following operations:

function f	ALU bit operation
000	S = 0; Cout = 0
001	S = x
010	S = y; Cout =1;
011	S = Cin; Cout = x
100	S = x OR y; Cout=x;
101	S = x AND y; Cout=x;
110	(Cout, S) = x + y + Cin; (component)
111	(Cout, S) = full subtractor (component)


x
ALU
S
y
C_{out}
C_{in}
f