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?

```
IN: 1 -> 0
```

```
1
```

```
A
```

```
B
```

```
C
```

Using delta delay scheduling

<table>
<thead>
<tr>
<th>Time (ns)</th>
<th>Delta</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>IN: 1 -&gt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eval INVERTER</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>A: 0 -&gt; 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eval NAND, AND</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>B: 1 -&gt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C: 0 -&gt; 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eval AND</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>C: 1 -&gt; 0</td>
</tr>
</tbody>
</table>

1 ns
Inertial Delay

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

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

- Inertial delay is default and REJECT is optional:

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

---

[Diagram of an input and output waveform with timing highlights]
Transport Delay

- Transport delay must be explicitly specified
  - i.e. keyword “TRANSPORT” must be used
- Signal will assume its new value after specified delay

```plaintext
-- TRANSPORT delay example
Output <= TRANSPORT NOT Input AFTER 10 ns;
```
Inertial and Transport Delay

Inertial Delay is useful for modeling logic gates.

Transport Delay is useful for modeling data buses, networks.

Inertial Delay is useful for modeling logic gates.
### Combinatorial Logic Operators

<table>
<thead>
<tr>
<th>Function</th>
<th>Input</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z \leq \text{NOT}(x)$</td>
<td>2</td>
<td>$z \leq \text{NOT}(x)$; $z \leq \text{NOT} x$;</td>
</tr>
<tr>
<td>$z \leq x \text{AND} y$</td>
<td>$2 + 2i$</td>
<td>$z \leq x \text{AND} y$;</td>
</tr>
<tr>
<td>$z \leq \text{NOT}(x \text{AND} y)$</td>
<td>$2i$</td>
<td>$z \leq \text{NOT}(x \text{AND} y)$;</td>
</tr>
<tr>
<td>$z \leq x \text{OR} y$</td>
<td>$2 + 2i$</td>
<td>$z \leq x \text{OR} y$;</td>
</tr>
<tr>
<td>$z \leq \text{NOT}(x \text{OR} y)$</td>
<td>$2i$</td>
<td>$z \leq \text{NOT}(x \text{OR} y)$;</td>
</tr>
<tr>
<td>$z \leq (x \text{AND} \text{NOT} y) \text{OR} (\text{NOT} x \text{AND} y)$</td>
<td>10</td>
<td>$z \leq (x \text{AND} \text{NOT} y) \text{OR} (\text{NOT} x \text{AND} y)$; $z \leq (x \text{AND} y) \text{NOR} (x \text{NOR} y)$; --AOI</td>
</tr>
<tr>
<td>$z \leq (x \text{AND} y) \text{OR} (\text{NOT} x \text{AND} \text{NOT} y)$</td>
<td>10</td>
<td>$z \leq (x \text{AND} y) \text{OR} (\text{NOT} x \text{AND} \text{NOT} y)$; $z \leq (x \text{NAND} y) \text{NAND} (x \text{OR} y)$; --OAI</td>
</tr>
</tbody>
</table>

Footnote: $(i=\#\text{inputs})$ 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

**AND**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>U</td>
</tr>
<tr>
<td>U</td>
<td>0</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

0 AND <anything> is 0

0 NAND <anything> is 1

**OR**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>U</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>1</td>
<td>U</td>
</tr>
</tbody>
</table>

1 OR <anything> is 1

1 NOR <anything> is 0

**NOT**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>U</td>
</tr>
<tr>
<td>1</td>
<td>U</td>
<td>0</td>
<td>U</td>
</tr>
</tbody>
</table>

11 NOR <anything> is 0
**Std_logic AND: X Forcing Unknown Value**

- **AND**
  - 0 AND <anything> is 0
  - 0 NAND <anything> is 1

- **OR**
  - 1 OR <anything> is 0
  - 0 NOR <anything> is 1

- **NOT**
  - 0 NOT <anything> is 1

---

**Truth Tables:***

**AND Truth Table:**

- 0 AND 0 = 0
- 0 AND X = X
- 0 AND 1 = 0
- 0 AND U = U
- X AND 0 = 0
- X AND X = X
- X AND 1 = X
- X AND U = U
- 1 AND 0 = 0
- 1 AND X = X
- 1 AND 1 = 1
- 1 AND U = U
- U AND 0 = 0
- U AND X = U
- U AND 1 = U
- U AND U = U

**OR Truth Table:**

- 0 OR 0 = 0
- 0 OR X = X
- 0 OR 1 = 1
- 0 OR U = U
- X OR 0 = X
- X OR X = X
- X OR 1 = 1
- X OR U = U
- 1 OR 0 = 1
- 1 OR X = 1
- 1 OR 1 = 1
- 1 OR U = 1
- U OR 0 = U
- U OR X = U
- U OR 1 = 1
- U OR U = U

**NOT Truth Table:**

- 0 NOT 0 = 1
- 0 NOT X = X
- 0 NOT 1 = 0
- 0 NOT U = U
- X NOT 0 = X
- X NOT X = X
- X NOT 1 = 0
- X NOT U = U
- 1 NOT 0 = 1
- 1 NOT X = X
- 1 NOT 1 = 0
- 1 NOT U = U
- U NOT 0 = U
- U NOT X = U
- U NOT 1 = 1
- U NOT U = U
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
The rising transition signal

\[ V_{cc} = 5.5 \text{ } 25^\circ C \]

- **1**: \( > 3.85 \text{ Volts} \)
- **X**
  - **H**: Unknown 2.20 Volt gap
  - **W**:
  - **L**: \( < 1.65 \text{ Volts} \)
- **0**:
### Multiple output drivers: Resolution Function

<table>
<thead>
<tr>
<th></th>
<th>U</th>
<th>X</th>
<th>0</th>
<th>L</th>
<th>Z</th>
<th>W</th>
<th>H</th>
<th>1</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>X</td>
<td>U</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>U</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>L</td>
<td>W</td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>H</td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>U</th>
<th>X</th>
<th>0</th>
<th>L</th>
<th>Z</th>
<th>W</th>
<th>H</th>
<th>1</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>Z</td>
<td>W</td>
<td>H</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>W</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- **Observe that 0 pulls down all weak signals to 0**
- **Observe that H <driving> L => W**
- **Note the multi-driver resolution table is symmetrical**
Resolution Function: std_logic buffer gate

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 \( z <= 'W' \ AND '1'; \)

\( z <= 'W' \ AND '1'; \quad \text{-- convert std_ulogic 'W' to std_logic 'X'} \)

\( z <= 'X' \ AND '1'; \quad \text{-- now compute the std_logic AND} \)

\( z <= 'X'; \)
2-to-1 Multiplexor: with-select-when

**Structural Logic**

- \( Y \leq sa \text{ OR } sb; \)
- \( sa \leq a \text{ AND NOT } s; \)
- \( sb \leq b \text{ AND } s; \)

**Combinatorial Logic**

- \( Y \leq a \text{ WHEN ‘0’}, \)
- \( b \text{ WHEN ‘1’}; \)

**Behavioral Logic**

- \( \text{WITH } s \text{ SELECT} \)
  - \( Y \leq a \text{ WHEN ‘0’}, \)
  - \( b \text{ WHEN ‘1’}; \)

- or alternatively

- \( \text{WITH } s \text{ SELECT} \)
  - \( Y \leq a \text{ WHEN ‘0’}, \)
  - \( b \text{ WHEN OTHERS}; \)

- Only values allowed
4-to-1 Multiplexor: with-select-when

\[ Y \leq sa \text{ OR } sb \text{ OR } sc \text{ OR } sd; \]
\[ sa \leq a \text{ AND } (\text{ NOT } s(1) \text{ AND } \text{ NOT } s(0)); \]
\[ sb \leq b \text{ AND } (\text{ NOT } s(1) \text{ AND } s(0)); \]
\[ sc \leq c \text{ AND } (s(1) \text{ AND } \text{ NOT } s(0)); \]
\[ sd \leq d \text{ AND } (s(1) \text{ AND } s(0)); \]

WITH \( s \) SELECT
\[ Y \leq a \text{ WHEN } "00", \]
\[ b \text{ WHEN } "01", \]
\[ c \text{ WHEN } "10", \]
\[ d \text{ WHEN OTHERS}; \]

Structural Combinatorial logic

As the complexity of the combinatorial logic grows, the SELECT statement, simplifies logic design but at a loss of structural information.
ENTITY Buffer_Tri_State IS
  PORT(x: IN std_logic;
       y: OUT std_logic;
       oe: IN std_logic);
END;

ARCHITECTURE Buffer3 OF Buffer_Tri_State IS
BEGIN
  WITH oe SELECT
  y <= x WHEN '1', -- Enabled: y <= x;
       'Z' WHEN '0'; -- Disabled: output a tri-state
END;
1) Assume each gate is 10 ns delay for the above circuit.

(a) Write entity-architecture for an inertial model
(b) Given the following waveform, draw, R, S, Q, NQ (inertial)
   \[
   R \leftarrow '0', '1' \text{ after } 25 \text{ ns}, '0' \text{ after } 30 \text{ ns}; \\
   S \leftarrow '1', '0' \text{ after } 20 \text{ ns}, '1' \text{ after } 35 \text{ ns}, '0' \text{ after } 50 \text{ ns};
   \]

(c) Write entity-architecture for a transport model
(d) Given the waveform in (b) draw, R, S, Q, NQ (transport)
(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:

- $X \leq \text{"1"}$, \text{"0"} after 10 ns, \text{"1"} after 20 ns, \text{"L"} after 30 ns, \text{"1"} after 40 ns;
- $Y \leq \text{"0"}$, \text{"L"} after 10 ns, \text{"W"} after 20 ns, \text{"Z"} after 30 ns, 0 after 40 ns;
- $F \leq \text{"0"}$, \text{"1"} after 10 ns, \text{"0"} after 50 ns;
Assignment #2 (Part 3 of 3)

3a) Write (no programming) a entity-architecture for a 1-bit ALU. The input will consist of \(x\), \(y\), \(C_{in}\), \(f\) and the output will be \(S\) and \(C_{out}\). Use as many sub-components as possible. The input function \(f\) will enable the following operations:

<table>
<thead>
<tr>
<th>function (f)</th>
<th>ALU bit operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>(S = 0) (C_{out} = 0)</td>
</tr>
<tr>
<td>001</td>
<td>(S = x)</td>
</tr>
<tr>
<td>010</td>
<td>(S = y)</td>
</tr>
<tr>
<td>011</td>
<td>(S = x \text{ AND } y)</td>
</tr>
<tr>
<td>100</td>
<td>(S = x \text{ OR } y)</td>
</tr>
<tr>
<td>101</td>
<td>(S = x \text{ XOR } y)</td>
</tr>
<tr>
<td>110</td>
<td>((C_{out}, S) = x + y + C_{in});</td>
</tr>
<tr>
<td>111</td>
<td>((C_{out}, S) = \text{ full subtractor})</td>
</tr>
</tbody>
</table>

3b) Calculate the number of transistors for the 1-bit ALU

3c) Write a entity-architecture for a N-bit ALU (for generate)