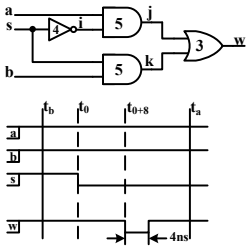
Chapter 7
VHDL Signal Model

Prepared by:
Nadereh Hatami

VHDL Signal Model

VHDL Signal Model

Timing and Concurrency of Operations



- Illustrating Timing and Concurrency

Sequential Modeling

```
i := NOT s;
j := a AND i;
K := s AND b;
W := j OR k;
```

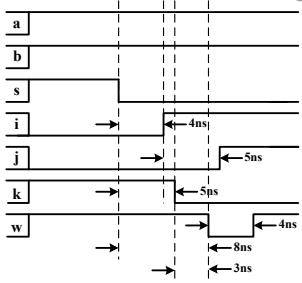
- Modeling a Multiplexer with Sequential Statements

Concurrent Modeling

```
ENTITY mux IS
   PORT (a, b, s : IN BIT; w : OUT BIT);
END ENTITY;
--
ARCHITECTURE concurrent of mux IS
   SIGNAL i, j, k : BIT;
BEGIN
   i <= NOT s AFTER 4 NS;
   j <= a AND i AFTER 5 NS;
   k <= b AND s AFTER 5 NS;
   w <= j OR k AFTER 3 NS;
END ARCHITECTURE concurrent;
```

- Modeling a Multiplexer with Concurrent Statements

Concurrent Modeling



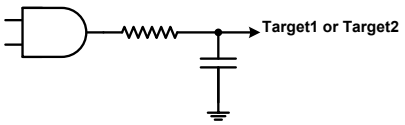
- Timing of Signals of Concurrent Description of previous Multiplexer

Signal Assignments

```
ENTITY example IS END ENTITY;
--
ARCHITECTURE delay OF example IS
   SIGNAL waveform : BIT;
   SIGNAL target1, target2, target3 : BIT;
BEGIN
   -- Inertial delay
   target1 <= waveform AFTER 5 NS;
   -- Inertial with reject
   target2 <= REJECT 3 NS INERTIAL waveform AFTER 5 NS;
   -- Illustrating transport delay
   target3 <= TRANSPORT waveform AFTER 5 NS;
   -- Creating waveform (not shown)
   waveform <= -- P5, N6, P4, N6, P3, N6, P2, P6,
               -- N5, P6, N4, P6, N3, P6, N2, N6;
END delay;
```

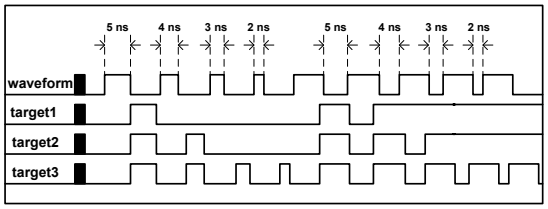
- VHDL Description for the Demonstration of Delay Mechanisms

Inertial Delay Mechanism



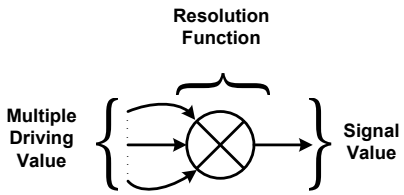
- The RC Delay is Best Represented by an Inertial Delay Mechanism

Comparing Inertial and Transport



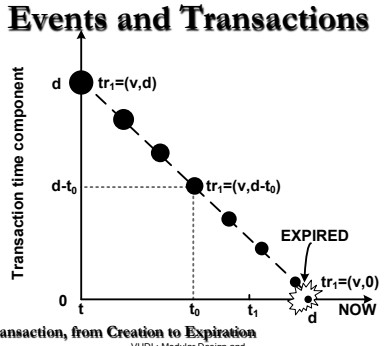
- Illustrating Differences between Delay Mechanisms in VHDL

Concurrent Assignments



- Resolving a Single Value from Multiple Driving Values

Events and Transactions



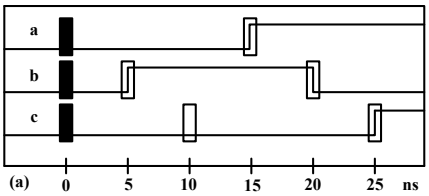
- A Transaction, from Creation to Expiration

Events and Transactions

```
ARCHITECTURE demo OF example IS
  SIGNAL a, b, c : BIT := '0';
BEGIN
   a <= '1' AFTER 15 NS;
   b <= NOT a AFTER 5 NS;
   c <= a AFTER 10 NS;
END demo;
```

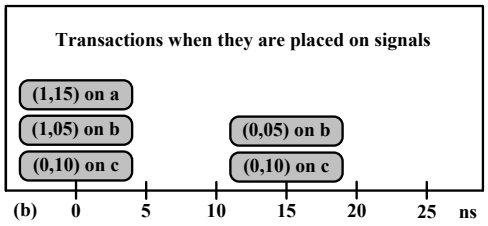
- A Simple Description for Illustrating Events and Transactions

Events and Transactions



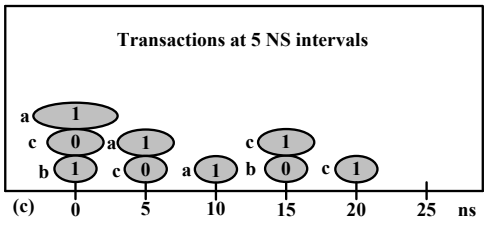
- Events and Transactions that Occur on Signals in previous slide : (a) The Resulting Timing Diagram Showing Transactions when they become Current;

Events and Transactions



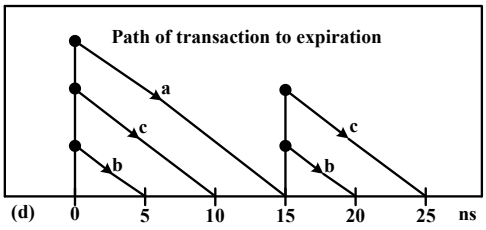
- Events and Transactions that Occur on Signals in previous slide : (b) Transactions when they are Placed on Signals;

Events and Transactions



- Events and Transactions that Occur on Signals in previous slide : (c) Transactions as their Time Values Approach Zero to Become Current;

Events and Transactions



- Events and Transactions that Occur on Signals in previous slide : (d) Transactions from Creation to Expiration

Delta Delay

```
ENTITY timing IS
   PORT (a, b : IN BIT; z, zbar : BUFFER BIT);
END ENTITY;
--
ARCHITECTURE delta of timing IS
BEGIN
   z_bar <= NOT z;
   z <= a AND b AFTER 10 NS;
END delta;
```

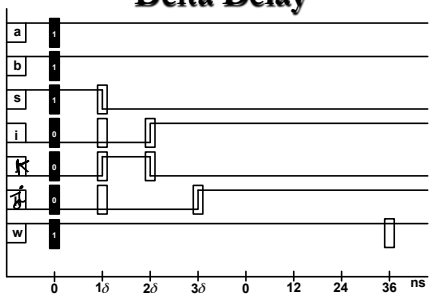
- Demonstrating Need for Delta Delay

Sequential Wait Statements

```
ENTITY mux IS
   PORT (a, b, s : IN BIT; w : OUT BIT);
END ENTITY;
--
ARCHITECTURE concurrent of mux IS
   SIGNAL i, j, k : BIT;
BEGIN
   i <= NOT s;
   j <= a AND i;
   k <= b AND s;
   w <= j OR k AFTER 36 NS;
END ARCHITECTURE concurrent;
```

- VHDL Description for Demonstrating the Delta Delay

Delta Delay



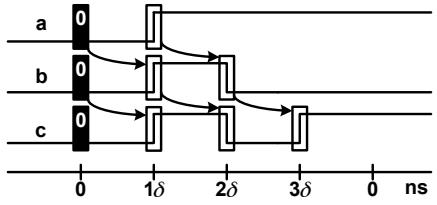
- Timing Diagram for the Description of Figure 7.13, Showing Delta Delays

Delta Delay

```
ARCHITECTURE concurrent OF timing_demo IS
   SIGNAL  a, b, c : BIT := '0';
BEGIN
   a <= '1';
   b <= NOT a;
   c <= NOT b;
END concurrent;
```

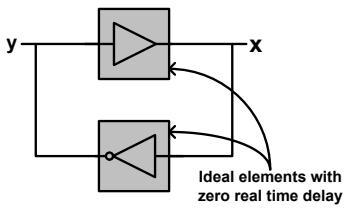
- Chain of Two Inverters, Delta Time, Transactions, and Concurrency

Delta Delay



- Timing Diagram for the *timing_demo* Description of previous slide

Delta Delay



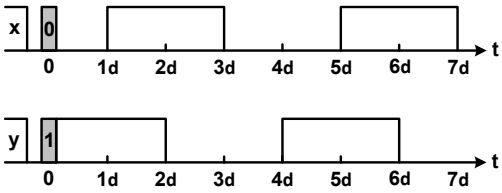
- Oscillation in Zero Real Time. (a) Circuit to Model;

Delta Delay

```
ARCHITECTURE forever OF oscillating IS
   SIGNAL  x: BIT := '0';
   SIGNAL  y: BIT := '1';
BEGIN
   x <= y;
   y <= NOT x;
END forever;
```

- Oscillation in Zero Real Time. (b) VHDL Representation;

Delta Delay



- Oscillation in Zero Real Time. (c) Signal Waveforms

Sequential Placement of Transactions

```
ARCHITECTURE sequential OF
sequential_placement IS
      ` ` `
BEGIN
   PROCESS
      x <= v1 AFTER t1;
      x <= v2 AFTER t2;
      WAIT;
   END PROCESS;
END sequential;
```

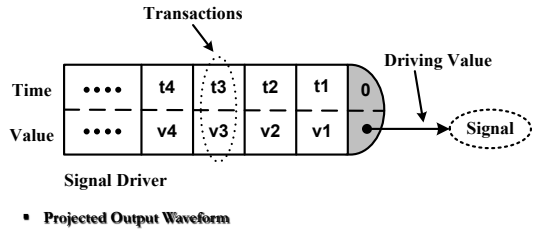
- Sequential Placement of Transactions in a Sequential Body of VHDL

Sequential Placement of Transactions

```
ARCHITECTURE concurrent OF
sequential_placement IS
   ` ` `
BEGIN
   a <= v1, v2 AFTER t1;
   x <= a AFTER t2;
END concurrent;
```

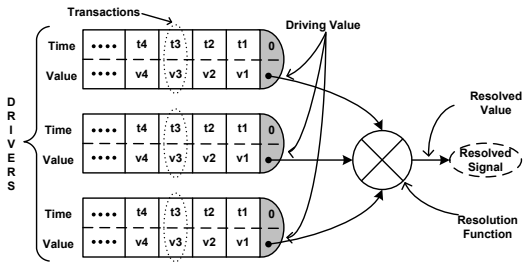
- Sequential Placement of Transactions in a Concurrent Body of VHDL

Signal Drivers



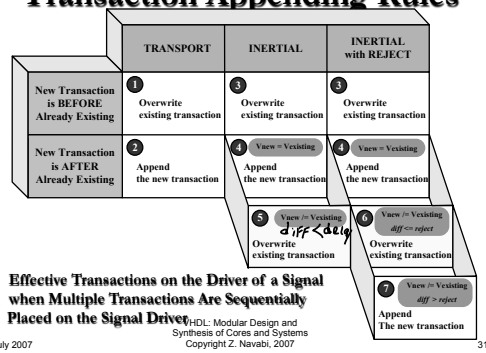
- Projected Output Waveform

Signal Drivers



- Multiple Drivers of a Resolved Signal

Transaction Appending Rules



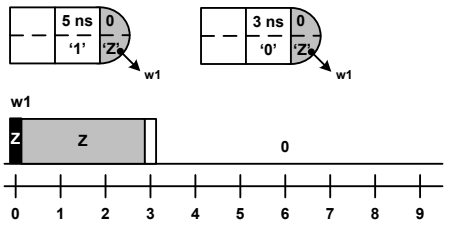
- Effective Transactions on the Driver of a Signal when Multiple Transactions Are Sequentially Placed on the Signal Driver

Transaction Appending Rules

```
case1: PROCESS BEGIN -- Transport, Before
   w1 <= '1' AFTER 5 NS;
   w1 <= TRANSPORT '0' AFTER 3 NS;
   WAIT;
END PROCESS case1; -- Overwrites existing
```

- Transport Delay, Before Existing Transactions

Transaction Appending Rules



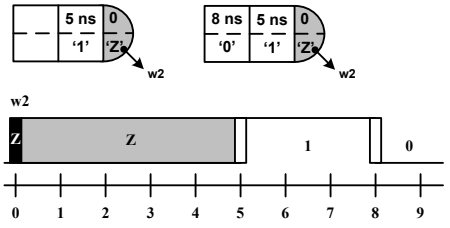
- Transport Delay, Before Existing Transactions (continued)

Transaction Appending Rules

```
case2: PROCESS BEGIN -- Transport, After
   w2 <= '1' AFTER 5 NS;
   w2 <= TRANSPORT '0' AFTER 8 NS;
   WAIT;
END PROCESS case2; -- Appends to existing
```

- Transport Delay After Existing Transaction

Transaction Appending Rules



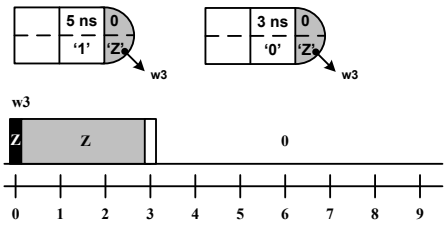
- Transport Delay After Existing Transaction

Transaction Appending Rules

```
case3a: PROCESS BEGIN -- Inertial, Before
   w3a <= '1' AFTER 5 NS;
   w3a <= INERTIAL '0' AFTER 3 NS;
   WAIT;
END PROCESS case3a; -- Overwrites existing
--
case3b: PROCESS BEGIN -- Reject, Before
   w3b <= '1' AFTER 5 NS;
   w3b <= REJECT 3 NS INERTIAL '0' AFTER 3 NS;
   WAIT;
END PROCESS case3b; -- Overwrites existing
```

- Inertial or Inertial with Reject, Before Existing

Transaction Appending Rules



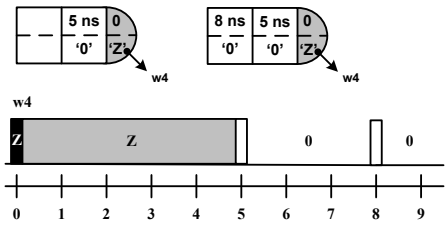
- Inertial or Inertial with Reject, Before Existing (continued)

Transaction Appending Rules

```
case4a: PROCESS BEGIN -- Inertial, After, Vn=Ve
   w4a <= '0' AFTER 5 NS;
   w4a <= INERTIAL '0' AFTER 8 NS;
   WAIT;
END PROCESS case4a; -- Appends to existing
--
case4b: PROCESS BEGIN -- Reject, After, Vn=Ve
   w4b <= '0' AFTER 5 NS;
   w4b <= REJECT 8 NS INERTIAL '0' AFTER 8 NS;
   WAIT;
END PROCESS case4b; -- Appends to existing
```

- Inertial or Inertial with Reject, Same Value Transaction After Existing

Transaction Appending Rules



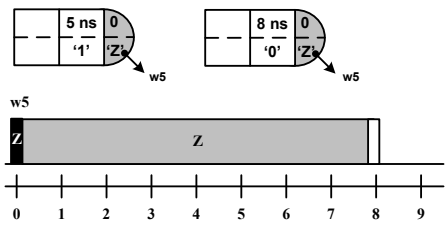
- Inertial or Inertial with Reject, Same Value Transaction After Existing (continued)

Transaction Appending Rules

```
case5: PROCESS BEGIN -- Inertial, After, Vn/=Ve
   w5 <= '1' AFTER 5 NS;
   w5 <= INERTIAL '0' AFTER 8 NS;
   WAIT;
END PROCESS case5; -- Overwrites existing
```

- Inertial Delay, Different Value Transaction After Existing

Transaction Appending Rules



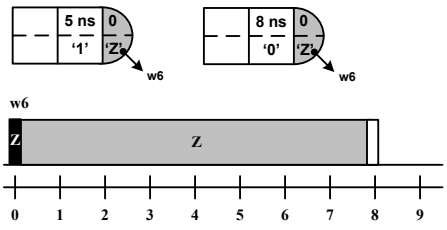
- Inertial Delay, Different Value Transaction After Existing (continued)

Transaction Appending Rules

```
case6: PROCESS BEGIN
   -- Reject, After, Vn/=Ve, Diff <= Reject
   w6 <= '1' AFTER 5 NS;
   w6 <= REJECT 4 NS INERTIAL '0' AFTER 8 NS;
   WAIT;
END PROCESS case6; -- Overwrites existing
```

- Inertial with Reject, Different Values, After Existing, Reject Occurs

Transaction Appending Rules



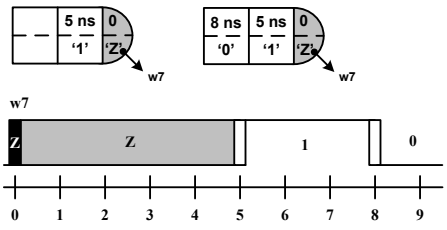
- Inertial with Reject, Different Values, After Existing, Reject Occurs

Transaction Appending Rules

```
case7: PROCESS BEGIN
   -- Reject, After, Vn/=Ve, Diff > Reject
   w7 <= '1' AFTER 5 NS;
   w7 <= REJECT 2 NS INERTIAL '0' AFTER 8 NS;
   WAIT;
END PROCESS case7; -- Appends to existing
```

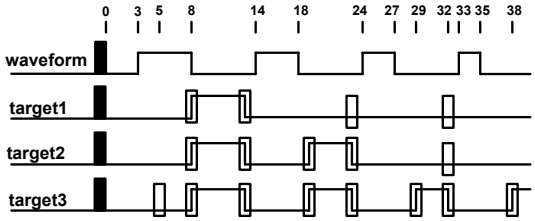
- with Reject, Different Values, After Existing, Reject Doesn't Occur

Transaction Appending Rules



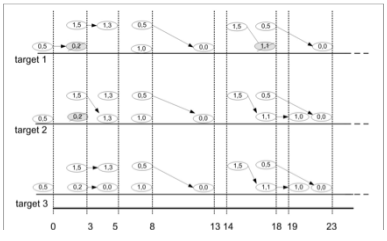
- Inertial with Reject, Different Values, After Existing, Reject Doesn't Occur

Pulse Rejection



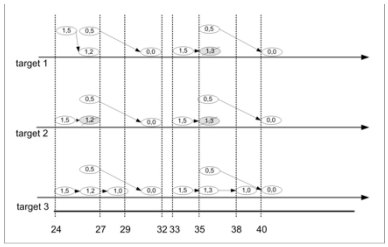
- Pulse Rejection in Inertial, Reject, and Transport Delay

Pulse Rejection



- New, Pending, and Expired Transactions on the Targets of previous slide

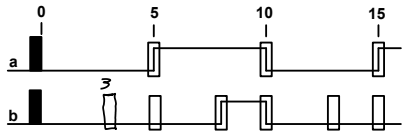
Pulse Rejection



- New, Pending, and Expired Transactions on the Targets of previous slide (Continued)

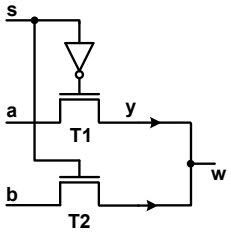
Placing Waveform Elements

```
ARCHITECTURE delay OF example IS
   SIGNAL a, b, BIT := '0';
BEGIN
   a <= '1' AFTER 5 NS, '0' AFTER 10 NS, '1' AFTER 15 NS;
   b <= '0', a AFTER 3 NS;
END ARCHITECTURE delay;
```



- Sequential Placement of Transactions by Concurrent Assignments

Resolving between Multiple Driving Values



- Pass Transistor Based Multiplexer

Resolving between Multiple Driving Values

```
ENTITY multiplexer IS
   PORT (a, b, s : IN v4l; w : OUT v4l);
END ENTITY;
==
-- Does not compile
ARCHITECTURE wired OF multiplexer IS
   SIGNAL y : wiring v4l;
BEGIN
   T1: y <= a WHEN s='0' ELSE 'Z';
   T2: y <= b WHEN s='1' ELSE 'Z';
   w <= y;
END ARCHITECTURE wired;
```

- Multiplexer Circuit, Two Concurrent Assignments (Does Not Compile)

Resolving a Pair of Values

```
FUNCTION wire (a, b : v4l) RETURN v4l IS
   CONSTANT v4l_wire_table : v4l_2d := (
      'X' => ('X', 'X', 'X', 'X'),
      '0' => ('X', '0', 'X', '0'),
      '1' => ('X', 'X', '1', '1'),
      'Z' => ('X', '0', '1', 'Z'));
BEGIN
   RETURN v4l_wire_table (a, b)
END wire;
```

- Resolving Every Pair of Values of *v4l* Type

Resolving Multiple Driving Values

```
FUNCTION wiring ( drivers : v4l_vector) RETURN v4l IS
   VARIABLE accumulate : v4l := 'Z';
BEGIN
   FOR i IN drivers'RANGE LOOP
      accumulate := wire (accumulate, drivers(i));
   END LOOP;
   RETURN accumulate;
END wiring;
```

- Wiring Resolution Function, an Array Version of *Wire*

Applying a Resolution Function

```
ARCHITECTURE wired OF multiplexer IS
   SIGNAL y : wiring v4l;
BEGIN
   T1: y <= a WHEN s='0' ELSE 'Z';
   T2: y <= b WHEN s='1' ELSE 'Z';
   w <= y;
END ARCHITECTURE wired;
```

- Working Architecture for Multiplexer

Resolution Package

```
FUNCTION wiring ( drivers : v4l_vector) RETURN v4l;
SUBTYPE wired_v4l IS wiring v4l;
TYPE wired_v4l_vector IS
            ARRAY (NATURAL RANGE <>) OF wired_v4l;
```

- Resolution Related Declarations

Resolution Package

```
LIBRARY utilities;
USE utilities.VerilogLogic.ALL;
ENTITY multiplexer_n4 IS
   PORT (a, b, c, d : IN wired_v4l_vector;
         s : IN wired_v4l_vector (1 DOWNTO 0);
         w : OUT wired_v4l_vector);
END ENTITY;
--
ARCHITECTURE wired OF multiplexer_n4 IS
BEGIN -- Four Bus Connections
  BC1: w <= a WHEN s="00" ELSE
          (a'RANGE => 'Z');
  BC2: w <= b WHEN s="01" ELSE
          (b'RANGE => 'Z');
  BC3: w <= c WHEN s="10" ELSE
          (c'RANGE => 'Z');
  BC4: w <= d WHEN s="11" ELSE
          (d'RANGE => 'Z');
END ARCHITECTURE wired;
```

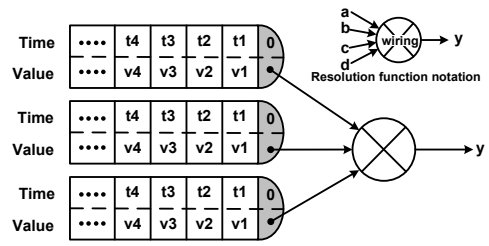
- Using Resolved Multi-bit Busses

A Resolution Package

```
PACKAGE VerilogLogic IS
FUNCTION oring ( drivers : v4l_vector) RETURN v4l;
   SUBTYPE ored_v4l IS oring v4l;
   TYPE ored_v4l_vector IS
                    ARRAY(NATURAL RANGE<>)OF ored_v4l;
   ' ' '
END PACKAGE VerilogLogic;
PACKAGE BODY VerilogLogic IS
   FUNCTION oring ( drivers : v4l_vector) RETURN v4l IS
      VARIABLE accumulate : v4l := '0';
   BEGIN
      FOR i IN drivers'RANGE LOOP
         accumulate := accumulate OR drivers(i);
      END LOOP;
      RETURN accumulate;
   END oring;
   ' ' '
END PACKAGE BODY VerilogLogic;
```

- Package Description for *Oring* Resolution Function

Relation to Sequential Transactions



- Projected Output Waveforms of Resolution Function

Resolutions with Guarded Assignments

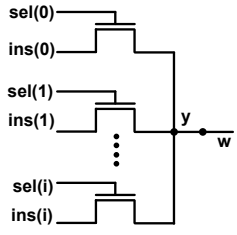
```
LIBRARY utilities;
USE utilities.VerilogLogic.ALL;
ENTITY selection_1ofn IS
   PORT (ins : IN wired_v4l_vector;
         sel : IN wired_v4l_vector;
         w : OUT wired_v4l);
END ENTITY;
--
ARCHITECTURE wired OF selection_1ofn IS
   SIGNAL y : wired_v4l BUS;
BEGIN
   Mi: FOR i IN ins'RANGE GENERATE
      Ti: BLOCK (sel(i) = '1') BEGIN

         y <= GUARDED ins (i);

      END BLOCK Ti;
   END GENERATE Mi;
   w <= y;
END ARCHITECTURE wired;
```

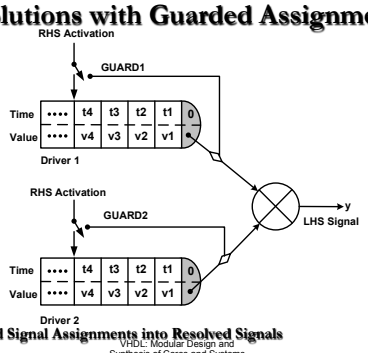
- 1-of-n Selection Logic Assign Guarded Assignments

Resolutions with Guarded Assignments



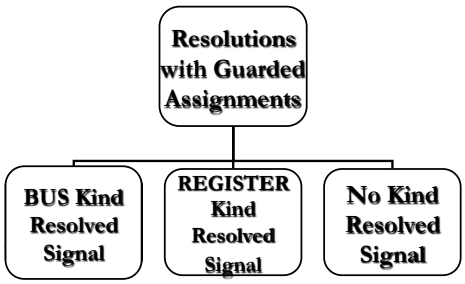
- NMOS Transistor Based Selection Logic

Resolutions with Guarded Assignments

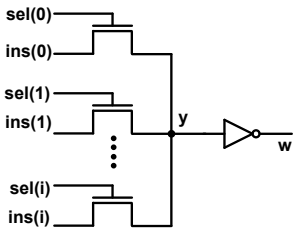


- Guarded Signal Assignments into Resolved Signals

Simple Assignments



REGISTER Kind Resolved Signal



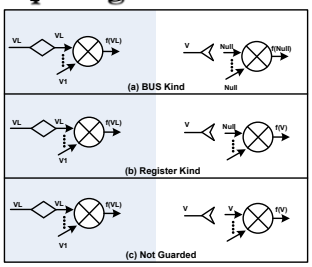
- NMOS Half-Register with Selection Logic

REGISTER Kind Resolved Signal

```
ARCHITECTURE wired_reg OF selection_lofn IS
   SIGNAL y : wired_v4l REGISTER;
BEGIN
   Mi: FOR i IN ins'RANGE GENERATE
      Ti: BLOCK (sel(i) = '1') BEGIN
         y <= GUARDED ins (i);
      END BLOCK Ti;
   END GENERATE Mi;
   w <= NOT y;
END ARCHITECTURE wired_reg;
```

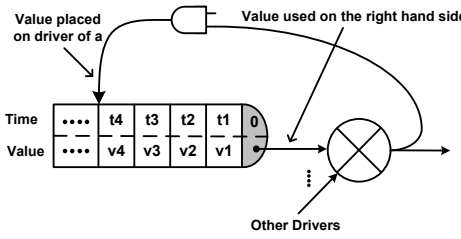
- Using REGISTER Kind for Selection Logic with Half-Register

Comparing Disconnections



- Last Disconnections. (a) BUS Kind; (b) REGISTER Kind; (c) no Kind

Resolving Right and Left Signals



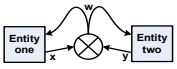
- Resolved Signals on Right- and Left-Hand Sides

Resolving INOUT Signals

```
ENTITY one( a : IN v4l; x: INOUT v4l )…
ENTITY two( b : IN v4l; y: INOUT v4l )…
ENTITY three IS END three;

ARCHITECTURE connecting OF three IS
  SIGNAL w: oring v4l;
BEGIN
  c1: ENTITY WORK.one PORT MAP( a, w );
  c2: ENTITY WORK.two PORT MAP( b, w );
END connecting;
```

(a)



(b)

- Connecting INOUT Ports Require Resolved Signals. (a) VHDL Code; (b) Graphical Notation

Summary

What we covered in this chapter were

- discuss sequential and concurrent assignments of values to signals
- took a limited look of a single driver and only discussed how sequential transactions affect a signal driver
- showed how multiple driving values interact for resolving a value for a signal with multiple concurrent drivers
- topics of sequential placement of transactions and resolution functions