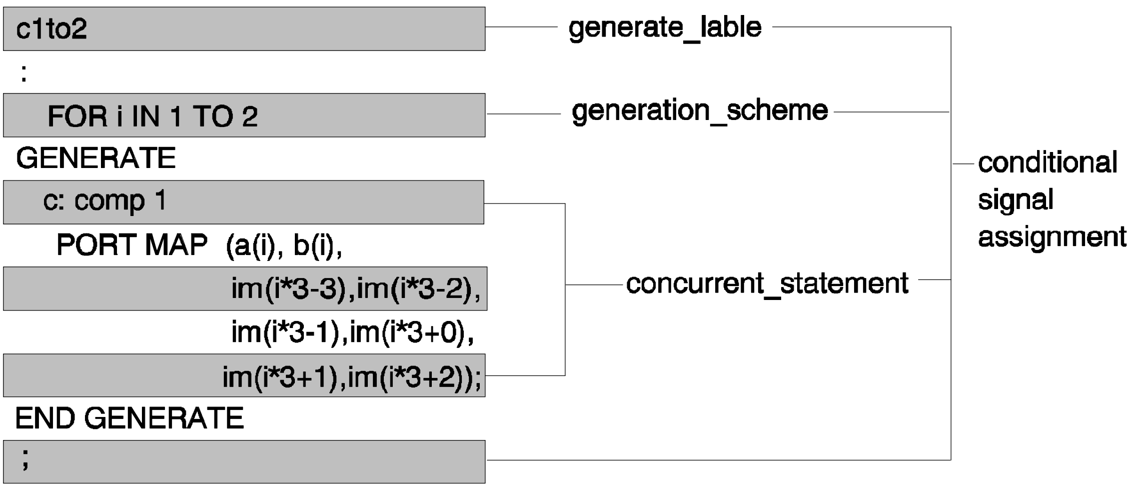
Structural...iterative_wiring.VHDL_description

```
ARCHITECTURE iterative OF nibble_comparator IS
  COMPONENT comp1
    PORT (a, b, gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
  END COMPONENT;
  FOR ALL : comp1 USE ENTITY WORK.bit_comparator (gate_level);
  SIGNAL im : BIT_VECTOR ( 0 TO 8);
BEGIN
  c0: comp1 PORT MAP (a(0), b(0), gt, eq, lt, im(0), im(1), im(2));
  c1to2: FOR i IN 1 TO 2 GENERATE
    c: comp1 PORT MAP (a(i), b(i), im(i*3-3), im(i*3-2), im(i*3-1),
                   im(i*3+0), im(i*3+1), im(i*3+2) );
  END GENERATE;
  c3: comp1 PORT MAP (a(3), b(3), im(6), im(7), im(8),
                  a_gt_b, a_eq_b, a_lt_b);
END iterative;
```

PORT MAP (	a(i),	b(i),	im(i*3-3),	im(i*3-2),	im(i*3-1),	im(i*3+0),	im(i*3+1),	im(i*3+2))
	i=1	i=1	i=1	i=1	i=1	i=1	i=1	i=1
PORT MAP (	a(1),	b(1),	im(0),	im(1),	im(2),	im(3),	im(4),	im(5))

- Figure shows mapping when c=1

Structural...iterative_wiring.syntax_details



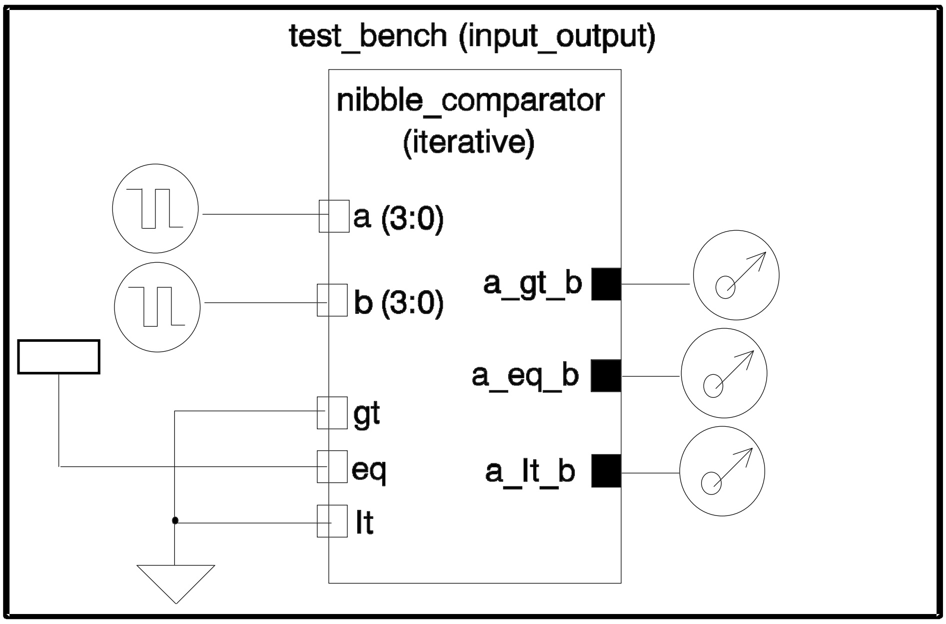
- Generate statement syntax
- Generate statement is a concurrent statement
- Generate statement brackets concurrent statements

Structural...iterative_wiring.VHDL_description

```
ARCHITECTURE iterative OF nibble_comparator IS
  COMPONENT comp1
    PORT (a, b, gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
  END COMPONENT;
  FOR ALL : comp1 USE ENTITY WORK.bit_comparator (gate_level);
  CONSTANT n : INTEGER := 4;
  SIGNAL im : BIT_VECTOR ( 0 TO (n-1)*3-1);
BEGIN
  c_all: FOR i IN 0 TO n-1 GENERATE
    l: IF i = 0 GENERATE
      least: comp1 PORT MAP (a(i), b(i), gt, eq, lt, im(0), im(1), im(2) );
    END GENERATE;
    m: IF i = n-1 GENERATE
      most: comp1 PORT MAP (a(i), b(i),
          im(i*3-3), im(i*3-2), im(i*3-1), a_gt_b, a_eq_b, a_lt_b);
    END GENERATE;
    r: IF i > 0 AND i < n-1 GENERATE
      rest: comp1 PORT MAP (a(i), b(i),
         im(i*3-3), im(i*3-2), im(i*3-1), im(i*3+0), im(i*3+1), im(i*3+2) );
    END GENERATE;
  END GENERATE;
END iterative;
```

- Architecture replaces the previous one with the same name
- This forms an *n* bit comparator
- Size depends on *n* only
- Use IF-Generate scheme as well as FOR scheme

Structural...test_bench.test_environment



- Aspect notation of a testbench is shown
- Testbench applies random data to *A* & *B* inputs
- Control inputs configure comparator FOR $A > B$, $A = B$, $A < B$
- Three outputs are monitored

Structural...testbench.VHDL_description

```
ARCHITECTURE input_output OF nibble_comparator_test_bench IS
  COMPONENT comp4 PORT (a, b : IN bit_vector (3 DOWNTO 0);
                 a_gt_b, a_eq_b, a_lt_b : IN BIT;
                 a_gt_b_out, a_eq_b_out, a_lt_b_out : OUT BIT);
  END COMPONENT;
  FOR a1 : comp4 USE ENTITY WORK.nibble_comparator(iterative);
  SIGNAL a, b : BIT_VECTOR (3 DOWNTO 0);
  SIGNAL eql, lss, gtr, gnd : BIT;  SIGNAL vdd : BIT := '1';
BEGIN
  a1: comp4 PORT MAP (a, b, gnd, vdd, gnd, gtr, eql, lss);
  a2:  a <= "0000",           ---- a = b (steady state)
       "1111" AFTER 0500 NS, -- a > b (worst case)
       "1110" AFTER 1500 NS, -- a < b (worst case)
       "1110" AFTER 2500 NS, -- a > b (need bit 1 info)
       "1010" AFTER 3500 NS, -- a < b (need bit 2 info)
       "0000" AFTER 4000 NS, -- a < b (steady state, prepare for next)
       "1111" AFTER 4500 NS, -- a = b (worst case)
       "0000" AFTER 5000 NS, -- a < b (need bit 3 only, best case)
       "0000" AFTER 5500 NS, -- a = b (worst case)
       "1111" AFTER 6000 NS; -- a > b (need bit 3 only, best case)
  a3 : b <= "0000",           ---- a = b (steady state)
       "1110" AFTER 0500 NS, -- a > b (worst case)
       "1111" AFTER 1500 NS, -- a < b (worst case)
       "1100" AFTER 2500 NS, -- a > b (need bit 1 info)
       "1100"  AFTER 3500 NS, -- a < b (need bit 2 info)
       "1101"  AFTER 4000 NS, -- a < b (steady state, prepare for next)
       "1102"  AFTER 4500 NS, -- a = b (worst case)
       "1103"  AFTER 5000 NS, -- a < b (need bit 3 only, best case)
       "1104"  AFTER 5500 NS, -- a = b (worst case)
       "1105"  AFTER 6000 NS; -- a > b (need bit 3 only, best case)
END input_output;
```

- *A*=1111, *B*=1110 @ 500 NS tests for worst case
- Best delay occurs at 5000 NS or 6000 NS

Structural...testbench.results

TIME (NS)	SIGNAL NAMES a(3:0)	b(3:0)	gtr	eql	lss
0	"0000"	"0000"	'0'	'0'	'0'
5	......	......	...	'1'	...
500	"1111"	"1110"	...	...	...
544	......	......	'1'	...	...
548	......	......	...	'0'	...
1500	"1110"	"1111"	...	...	...
1544	......	......	'0'	...	...
1548	......	......	...	...	'1'
2500	......	"1100"	...	...	...
2533	......	......	...	...	'0'
2537	......	......	'1'	...	...
3500	**"1010"**	......	...	...	...
3522	......	......	'0'	...	...
3526	......	......	...	...	'1'
4000	"0000"	"1111"	...	...	...
4500	"1111"	......	...	...	...
4544	......	......	...	'1'	...
4548	......	......	...	...	'0'
5000	"0000"	......	...	...	...
5011	......	......	...	'0'	...
5015	......	......	...	...	'1'
5500	......	"0000"	...	...	...
5544	......	......	...	...	'0'
5548	......	......	...	'1'	...
6000	"1111"	......	...	...	...

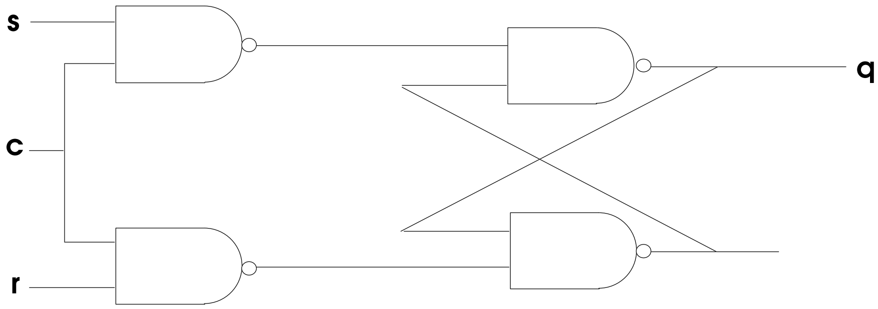
Events begins on bit 2

- Activity flows from:

Testbench => nibble_comparator => bit_comparator => gates...

- AT 2500NS: - testbench activates nibble comparator
  - nibble comparator activates bit_comparator
  - bit comparator activates its primitive gate
  - last 2 steps repeat while there are activities

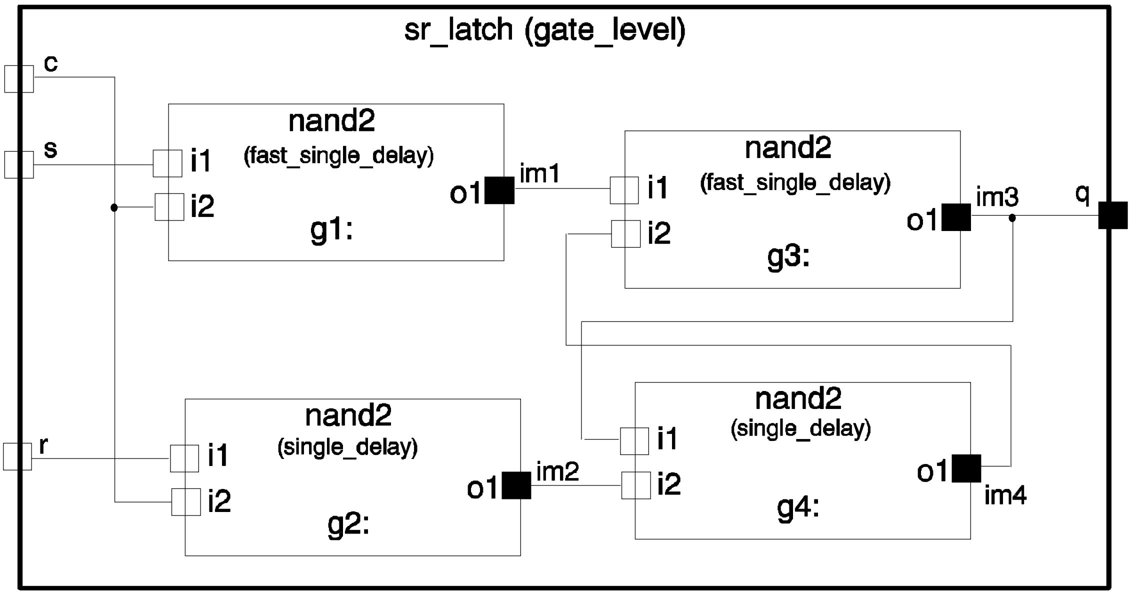
Structural...latch_design.single_bit_SR_latch



- Use four NAND gates to design a latch
- If all delays are equal then (0,0) on (q, qbar) will oscillate
- Remedy by careful initial value setting, or
- Remedy by using gates of different delay values

Structural...latch_design.single_bit_SR_latch

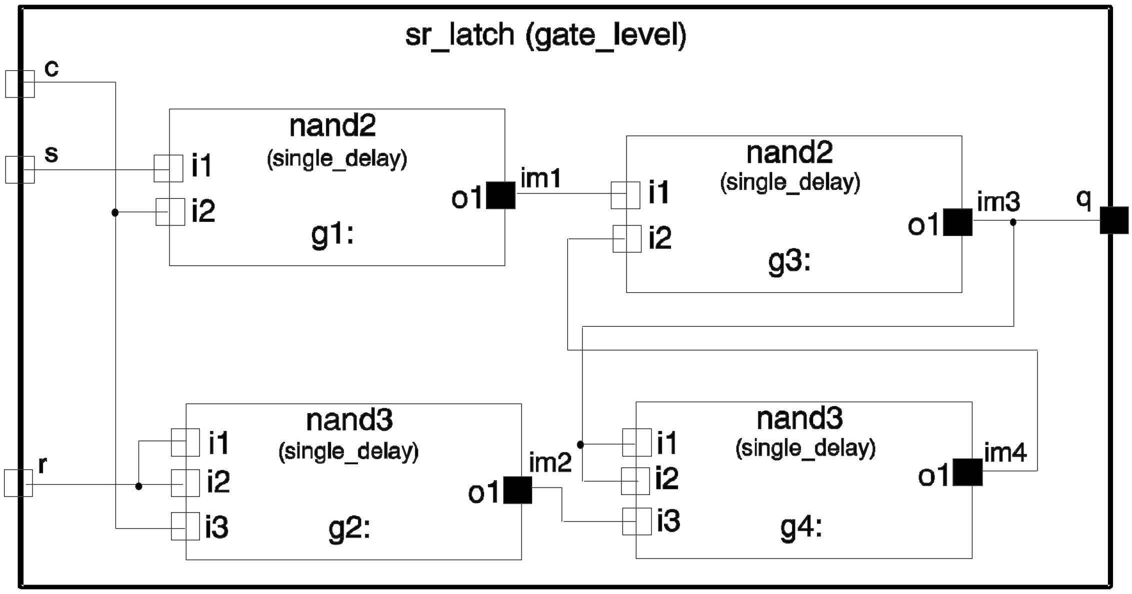
```
ARCHITECTURE fast_single_delay OF nand2 IS
BEGIN    o1 <= i1 NAND i2 AFTER 3 NS;
END fast_single_delay;
```




```
ARCHITECTURE gate_level OF sr_flipflop IS
  COMPONENT n2 PORT (i1, i2: IN BIT; o1: OUT BIT); END COMPONENT;
  FOR g1, g3 : n2 USE ENTITY WORK.nand2 (fast_single_delay);
  FOR g2, g4 : n2 USE ENTITY WORK.nand2 (single_delay);
  SIGNAL im1, im2, im3, im4 : BIT;
BEGIN
   g1 : n2 PORT MAP (s, c, im1);          g3 : n2 PORT MAP (im1, im4, im3);
   g2 : n2 PORT MAP (r, c, im2);          g4 : n2 PORT MAP (im3, im2, im4);
  q <= im3;
END gate_level;
```

- Use *fast_single_delay* with different delay value
- Composition aspect illustrates VHDL wiring
- Two separate configuration specifications are used

Structural...latch_design.single_bit_SR_latch

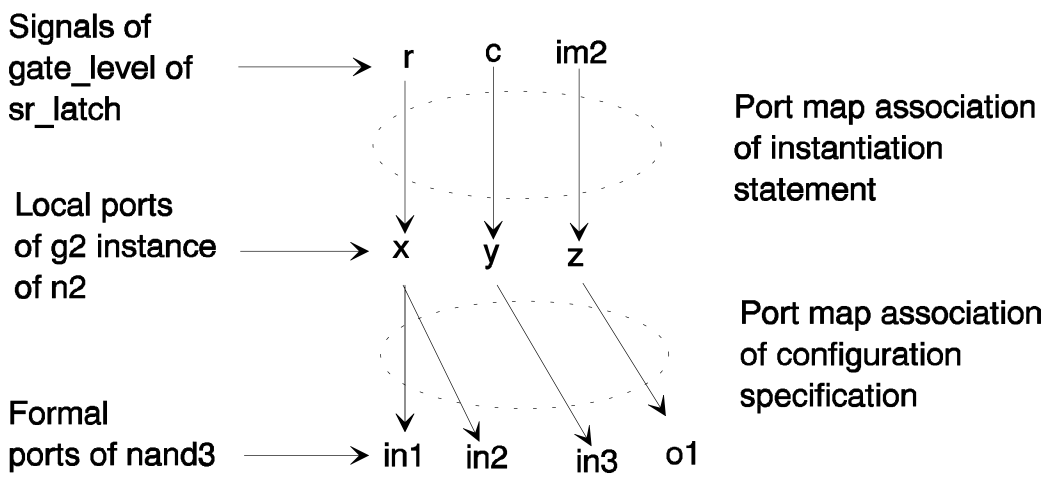



```
ARCHITECTURE gate_level OF sr_flipflop IS
  COMPONENT n2 PORT (x, y: IN BIT; z: OUT BIT); END COMPONENT;
  FOR g1, g3 : n2
      USE ENTITY WORK.nand2 (single_delay) PORT MAP (x, y, z);
  FOR g2, g4 : n2
      USE ENTITY WORK.nand3 (single_delay) PORT MAP (x, x, y, z);
  SIGNAL im1, im2, im3, im4 : BIT;
BEGIN
   g1 : n2 PORT MAP (s, c, im1);          g3 : n2 PORT MAP (im1, im4, im3);
   g2 : n2 PORT MAP (r, c, im2);          g4 : n2 PORT MAP (im3, im2, im4);
  q <= im3;
END gate_level;
```

- *nand3* has different delay value than *nand2*
- Use *nand3* to remedy oscillation problem

Structural...latch design.single_bit_SR_latch

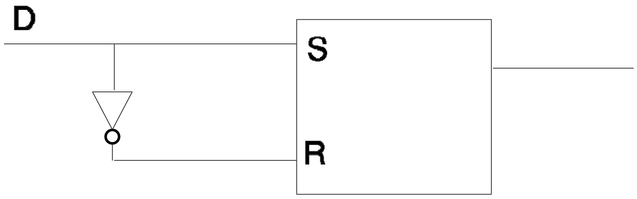
```
ARCHITECTURE gate_level OF sr_flipflop IS
  COMPONENT n2 PORT (x, y: IN BIT; z: OUT BIT); END COMPONENT;
  FOR g1, g3 : n2
      USE ENTITY WORK.nand2 (single_delay) PORT MAP (x, y, z);
  FOR g2, g4 : n2
      USE ENTITY WORK.nand3 (single_delay) PORT MAP (x, x, y, z);
  SIGNAL im1, im2, im3, im4 : BIT;
BEGIN
   g1 : n2 PORT MAP (s, c, im1);          g3 : n2 PORT MAP (im1, im4, im3);
   g2 : n2 PORT MAP (r, c, im2);          g4 : n2 PORT MAP (im3, im2, im4);
  q <= im3;
END gate_level;
```



- Association is done in two steps
- Port map of component declaration is default
- Use Port map in configuration declaration instead of default

Structural...old_new_comparator.latch_design.d_latch

ASSEMBLING PARTS FOR A NEW DESIGN




```
ENTITY d_latch IS    PORT(d,c : IN BIT;q: OUT BIT);   END d_latch;
--
ARCHITECTURE sr_based OF d_latch IS
  COMPONENT sr_latch    PORT (s, r, clk : IN BIT; q : OUT BIT);
  END COMPONENT;
  COMPONENT inv_t     PORT (i1 : IN BIT; o1 : OUT BIT);
  END COMPONENT;
SIGNAL dbar: BIT;
BEGIN
  c1 : sr_latch PORT MAP (d, dbar, c, q);
  c2 : inv_t PORT MAP (d, dbar);
END sr_based;
```

- New design will use parts developed so far
- *Old_new_comparator* needs a *byte_latch* and a *comparator*

In the *sr_based* ARCHITECTURE of *d_latch*:

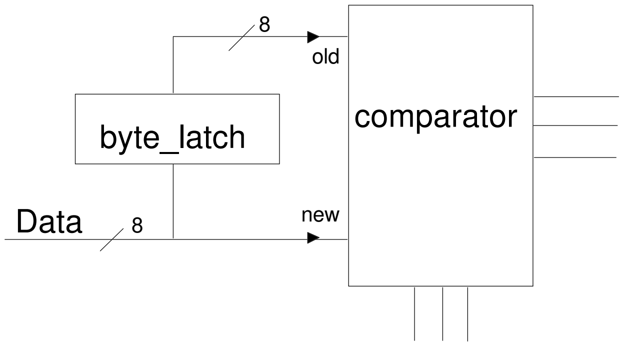
- Binding alternative uses component declaration as default
- Architecture most recently analyzed will be used

Structural...old_new_comparator.latch_design

```
ENTITY byte_latch IS
  PORT (di : IN BIT_VECTOR (3 DOWNTO 0);
        clk : IN BIT; qo : OUT BIT_VECTOR( 3 DOWNTO 0));
END byte_latch;
--
ARCHITECTURE iterative OF byte_latch IS
  COMPONENT d_latch
     PORT (d, c : IN BIT; q : OUT BIT);
  END COMPONENT;
BEGIN
  g : FOR i IN di'RANGE GENERATE
     L7DT0 : dlatch PORT MAP (di(i), clk, qo(i));
  END GENERATE;
END iterative;
```

- Wire *n* latches to form an n-bit latch
- Configuration specification is not required
- Component declaration defaults are used
- Generate statement wires bits of D-latch

Structural...old_new_comparator.system_wiring




```
ENTITY old_new_comparator IS
   PORT (i : IN BIT_VECTOR (7 DOWNTO 0);
   clk : IN BIT; compare : OUT BIT);
END old_new_comparator;
--
ARCHITECTURE wiring OF old_new_comparator IS
  COMPONENT latch    PORT (di : IN BIT_VECTOR (7 DOWNTO 0);
                clk : IN BIT; qo : OUT BIT_VECTOR (7 DOWNTO 0));
  END COMPONENT;
  COMPONENT byte_comparator
                PORT(a, b : BIT_VECTOR (7 DOWNTO 0);
                gt, eq, lt : IN   BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
  END COMPONENT;
  SIGNAL con1 : BIT_VECTOR (7 DOWNTO 0);
  SIGNAL vdd : BIT := '1';    SIGNAL gnd : BIT := '0';
BEGIN
  l : latch PORT MAP(i, clk, con1);
  c : byte_comparator
     PORT  MAP(con1, i, gnd, vdd, gnd, OPEN, compare, OPEN);
END wiring;
```

- Circuit compares old and new data
- Default binding is used

1. Outline: Introduction, Organization, Outline

2. Review: Behavioral description, Using process statements, Top-down design, Using available components, Wiring predefined components, Wiring from bottom to top, Generation of testbench data, Using procedures

3. VHDL_Timing: Modeling requirements, Objects & classes, Signals & variables, Concurrent & sequential assignments, Events, transactions & delta delays, Delay modeling, Sequential placement of transactions, Conventions

4. Structural_Description_of_Hardware: Wiring parts into larger designs, Start with primitives, Wire gates into general purpose components, Use iterative constructs, Generate testbenches, Show binding alternatives, Use gate-based components for a larger design

5. Design_Organization: Subprograms, Packaging, Parameter specification, Parametrization, Top level configuration, Design libraries, A complete example

6. Utilities_for_high_level_descriptions: Language aspects, Types, Over loading subprograms operators, Predefined Attributes, User defined attributes

7. Dataflow: Constructs for dataflow descriptions, Multiplexing and clocking, Multiple assignments, State machines, Open collector gates, A complete dataflow example, Load dependent timing

8. Behavioral_Descriptions: Constructs for sequential descriptions, Assertion for behavioral checks, Handshaking constructs, Timing control, Formatted I/O, MSI parts, A complete MSI based design

9. STANDARDS: MVL9: logic value system, Logic type, Operators, Conversions; VHDL'93: Operators, Delay model, Instantiation, Binding, Attributes, Signal assignments, Report, Postponed process

