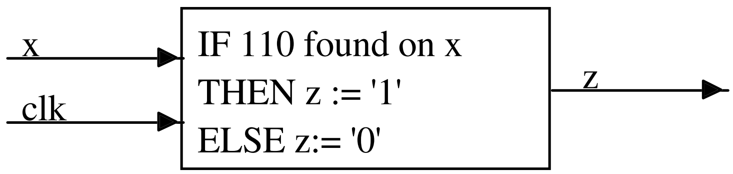
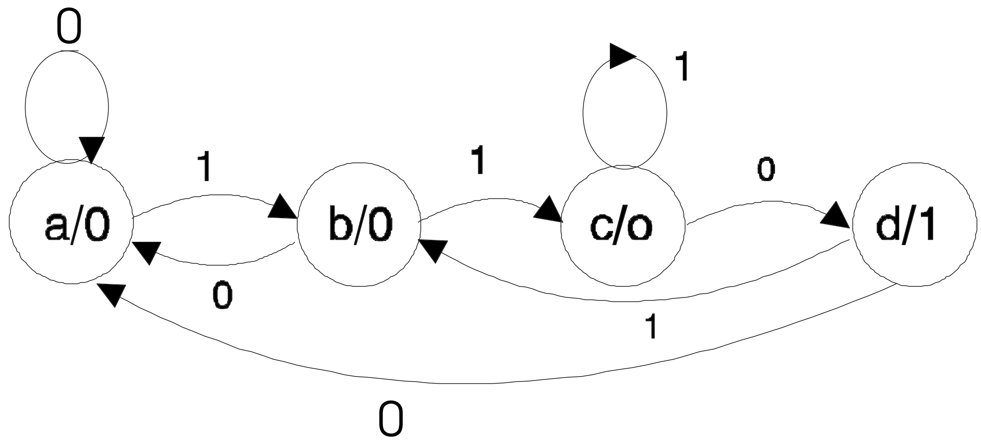
PART 4

VHDL_Overview

- A Moore 110 sequence detector
- Behavioral description
- Using process statements
- Top-down design of the sequence detector
- Using available components
- Wiring predefined components
- Wiring from bottom to top
- Generation of data for testbench
- Using procedures
- A simulatable design
- Other VHDL operators

VHDL_Overview.moore_110_detector





- Design a 110 detector
- Detect 110 on *x*
- Output becomes 1 when found

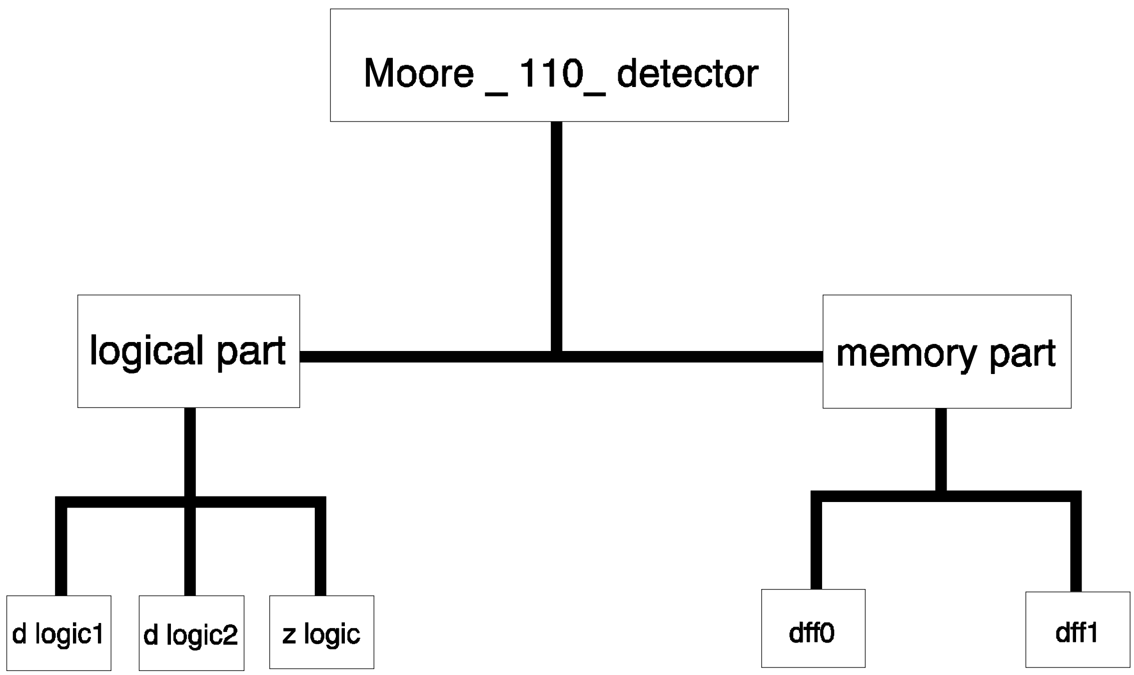
© Zainalabedin Navabi, 1994

VHDL_Overview.moore_110_detector.behavioral

```
ENTITY moore_110_detector IS    PORT (x, clk : IN BIT; z : OUT BIT);
END moore_110_detector;
--
ARCHITECTURE behavioral OF moore_110_detector IS
    TYPE state IS (reset, goto1, goto11, goto110);
SIGNAL current : state := reset;
BEGIN
  PROCESS(clk)
  BEGIN
     IF clk = '1' THEN
       CASE current IS
           WHEN reset =>
              IF x = '1' THEN current <= goto1;
              ELSE current <= reset;  END IF;
           WHEN goto1 =>
              IF x = '1' THEN current <= goto11;
              ELSE current <= reset;   END IF;
           WHEN goto11 =>
              IF x = '1' THEN current <= goto11;
              ELSE current <= goto110;  END IF;
           WHEN goto110 =>
              IF x = '1' THEN current <= goto1;
              ELSE current <= reset;  END IF;
       END CASE;
     END IF;
  END PROCESS;
  z <='1' WHEN current = goto110 ELSE '0';
END behavioral;
```

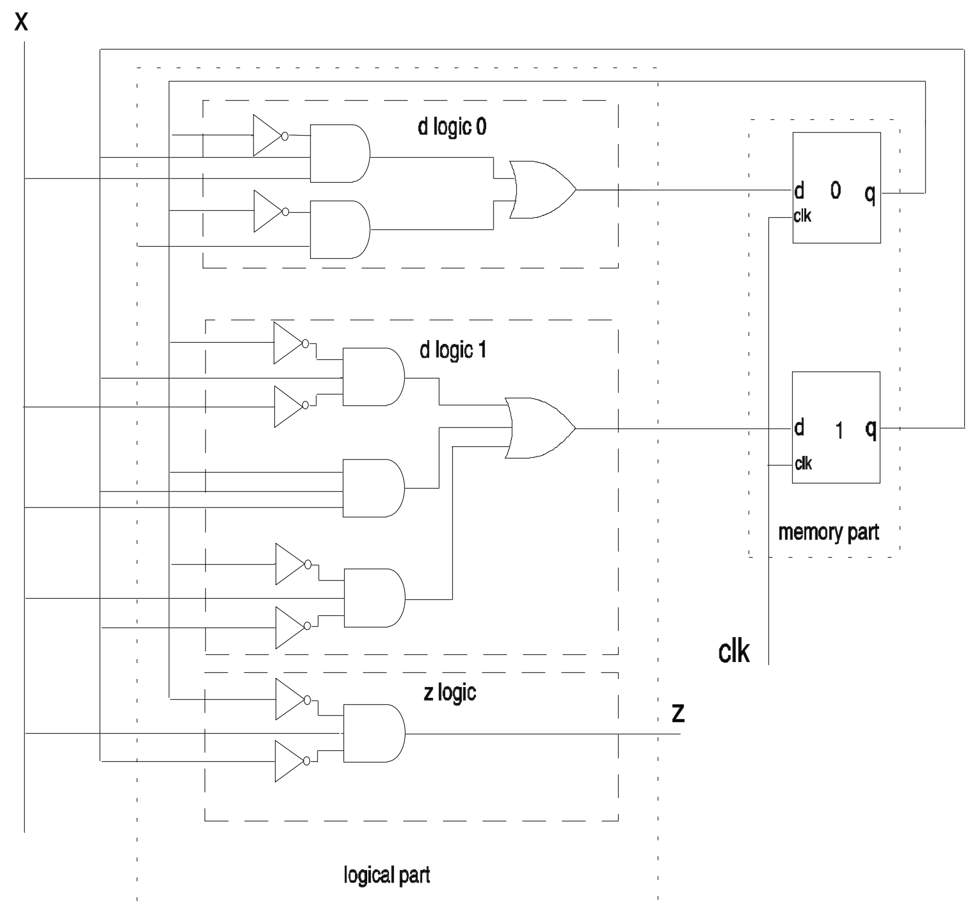
- Entity and architecture for interface and operation
- A single process includes a case statement
- Statements in a process statement execute sequentially

VHDL_Overview.moore_110_detector.top_down_design



- Partition design to available components

VHDL_Overview.moore_110_...top_down_...overall_structure



- Logic circuit for the sequence detector
- Three logical functions are used

VHDL_Overview.moore_110_...top_down...available_components

```
ENTITY dff IS
  PORT(d : IN BIT; clk : IN BIT; q : OUT BIT);
END dff;
--
ARCHITECTURE dataflow OF dff IS
BEGIN
  b:BLOCK (clk = '1' AND NOT clk'STABLE)
  BEGIN
    q <= GUARDED d AFTER 11 NS;
  END BLOCK;
END dataflow;
```

- Available parts include DFF, and f, g, and z logical functions
- VHDL Entity & Architecture for each unit is shown

VHDL_Overview.moore_110_...top_down...available_components

```
ENTITY logicfunction_f IS
  PORT (i1, i2, i3 : IN BIT; o1 : OUT BIT);
END logicfunction_f;
--
ARCHITECTURE dataflow OF logicfunction_f IS
BEGIN
  o1 <= ((NOT i1) AND i2) OR ((NOT i2) AND i1 AND i3) AFTER 8 NS;
END dataflow;

ENTITY logicfunction_g IS
  PORT (i1, i2, i3 : IN BIT; o1 : OUT BIT);
END logicfunction_g;
--
ARCHITECTURE dataflow OF logicfunction_g IS
BEGIN
  o1    <=      (i2 AND (NOT i1) AND (NOT i3)) OR (i2 AND i1 AND i3) OR
                ((NOT i2) AND (NOT i1) AND i3) AFTER 8 NS;
END dataflow;

ENTITY logicfunction_z IS
  PORT (i1, i2, i3 : IN BIT; o1 : OUT BIT);
END logicfunction_z;
--
ARCHITECTURE dataflow OF logicfunction_z IS
BEGIN
  o1 <= (i2 AND (NOT i1) AND (NOT i3)) AFTER 8 NS;
END dataflow;
```

- VHDL Entity & Architecture for logical functions

VHDL_Overview.moore_110...top_down_...first_level_wiring

```
ENTITY logical_part IS
  PORT (x, q0_in, q1_in : IN BIT; d0_out, d1_out, z_out  : OUT BIT);
END logical_part;
--
ARCHITECTURE structural OF logical_part IS
  COMPONENT  c1 PORT (i1, i2, i3 : IN BIT ; o1 : OUT BIT);
  END COMPONENT;
  FOR d_logic0 : c1 USE ENTITY WORK.logicfunction_g (dataflow);
  COMPONENT   c2 PORT (i1, i2, i3 : IN BIT; o1 : OUT BIT);
  END COMPONENT;
  FOR d_logic1 : c2 USE ENTITY WORK.logicfunction_f (dataflow);
  COMPONENT   c3 PORT(i1,i2,i3 : IN BIT; o1 : OUT BIT);
  END COMPONENT;
  FOR z_logic :c3 USE ENTITY WORK.logicfunction_z(dataflow);
BEGIN
  d_logic0: c1 PORT MAP(q0_in,q1_in,x,d0_out);
  d_logic1: c2 PORT MAP(q0_in,q1_in,x,d1_out);
  z_logic : c3 PORT MAP(q0_in,q1_in,x,z_out);
END structural;
```

- Wiring the logical part
- Statements in architecture execute concurrently

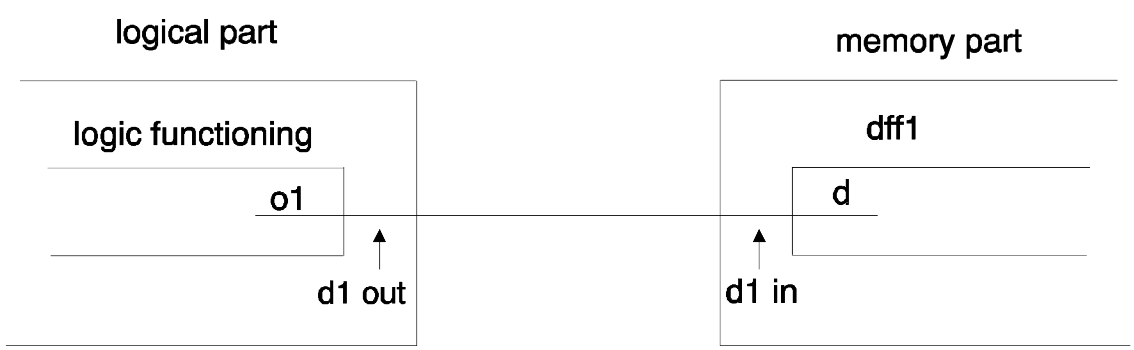
VHDL_Overview.moore_110...top_down_...first_level_wiring

```
ENTITY memory_part IS
  PORT(d0_in,d1_in,clk : IN BIT;q0_out,q1_out : OUT BIT);
END memory_part;
--
ARCHITECTURE structural OF memory_part IS
  COMPONENT  m
    PORT(d : IN BIT; clk : IN BIT; q : OUT BIT);
  END COMPONENT;
  FOR dff0, dff1 : m USE ENTITY WORK.dff (dataflow);
BEGIN
  dff0 : m PORT MAP (d0_in, clk, q0_out);
  dff1 : m PORT MAP (d1_in, clk, q1_out);
END structural;
```

- Wiring memory part
- Memory part consists of two flip-flops
- Statements in Architecture execute concurrently

VHDL_Overview.moore_110...top_down_...wiring_complete_system

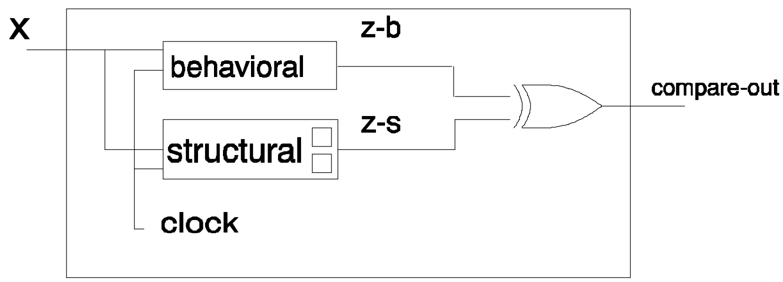
```
ARCHITECTURE structural OF moore_110_detector IS
  COMPONENT
    l PORT (x, q0_in, q1_in : IN BIT; d0_out, d1_out, z_out : OUT BIT);
  END COMPONENT;
  FOR lpart : l USE ENTITY WORK.logical_part (structural);
  COMPONENT
    m PORT (d0_in, d1_in, clk : IN BIT; q0_out, q1_out : OUT BIT);
  END COMPONENT;
  FOR mpart : m USE ENTITY WORK.memory_part(structural);
  SIGNAL conn0, conn1, conn2, conn3 : BIT;
BEGIN
  lpart : l PORT MAP (x, conn0, conn1, conn2, conn3, z);
  mpart : m PORT MAP (conn2, conn3, clk, conn0, conn1);
END structural;
```



- Same entity as declared
- Use structural instead of behavioral architecture

© Zainalabedin Navabi, 1994

VHDL_Overview.moore_110_detector.testbench




```
ENTITY  moore_test IS   END moore_test;
--
ARCHITECTURE input_output OF moore_test IS
  COMPONENT comp1      PORT (x, clk : IN BIT; z : OUT BIT);
  END COMPONENT;
  FOR c1 : comp1 USE ENTITY WORK.moore_110_detector (behavioral);
  FOR c2 : comp1 USE ENTITY WORK.moore_110_detector (structural);
  SIGNAL x_in, clock, z_beh, z_struct, compare_out : BIT;
BEGIN
  x_in <= '0',  '1' AFTER 500 NS,
                '0' AFTER 900 NS,           '1' AFTER 1100 NS,
                '0' AFTER 1300 NS,          '1' AFTER 1500 NS,
                '0' AFTER 1900 NS,          '1' AFTER 2100 NS,
                '0' AFTER 2300 NS,          '1' AFTER 2500 NS;
  clock <= NOT clock AFTER 100 NS WHEN NOW < 3000 NS ELSE clock;
  c1 : comp1 PORT MAP (x_in, clock, z_beh);
  c2 : comp1 PORT MAP (x_in, clock, z_struct);
  compare_out <= z_beh XOR z_struct;
END input_output;
```

- A testbench tests and compares two architectures

VHDL_Overview.moore_110_detector.data_generation

```
PROCEDURE   serial_data_generation
(SIGNAL target : OUT BIT; int : IN INTEGER; t : IN TIME) IS
   VARIABLE i : INTEGER;
   VARIABLE bit_val : BIT;
   VARIABLE current : TIME :=100 NS;
BEGIN
  i := int;
  WHILE i >0 LOOP
    IF (i MOD 2 = 1) THEN
      bit_val := '1';
    ELSE
      bit_val := '0';
    END IF;
    target <= TRANSPORT bit_val AFTER current;
    current := current + t;
    i := i/2;
  END LOOP;
END serial_data_generation;
```

- Procedure is a sequential body
- It takes zero real time to execute the entire procedure

VHDL_Overview.moore_110_detector.procedural_test

```
ARCHITECTURE procedural OF moore_test IS
  PROCEDURE serial_data_generation
  (SIGNAL target : OUT BIT; int : IN INTEGER; t : IN TIME) IS
     VARIABLE i : INTEGER;
     VARIABLE bit_val : BIT;
     VARIABLE current : TIME :=100 NS;
  BEGIN
    i := int;
    WHILE i > 0 LOOP
      IF (i MOD 2 = 1) THEN  bit_val :='1';
      ELSE  bit_val := '0';
      END IF;
      target <= TRANSPORT bit_val AFTER current;
      current:=current + t;
      i := i / 2;
    END LOOP;
  END serial_data_generation;
  COMPONENT comp  PORT (x,clk:IN BIT;z:OUT BIT);  END COMPONENT;
  FOR c1 : comp USE ENTITY WORK.moore_110_detector(structural);
  SIGNAL x_in, clock, z_struct : BIT;
 BEGIN
  serial_data_generation (x_in, 5548, 200 NS);
  clock <= NOT clock AFTER 100 NS WHEN NOW < 3000 NS ELSE clock;
  c1 : comp PORT MAP (x_in, clock, z_struct);
END procedural;
```

- The three statements execute concurrently

VHDL_Overview.moore_110_detector.testbench

- Test may be done for various purposes
  - Verify the design
  - Check the delays
  - Find maximum clock speed
  - Compare behavioral & dataflow
  - A testbench can instantiate two versions of a component
  - XOR gates can be used to flag discrepancies

© Zainalabedin Navabi, 1994

VHDL_Overview.other_operators

PREDEFINED OPERATORS

```
      LOGICAL OPERATORS: NOT AND OR NAND NOR XOR
           OPERAND TYPE: BIT BOOLEAN
            RESULT TYPE: BIT BOOLEAN

   RELATIONAL OPERATORS: = / = < <= > >=
           OPERAND TYPE: any type
            RESULT TYPE: BOOLEAN

   ARITHMETIC OPERATORS: + - * / **
                       : MOD REM ABS
           OPERAND TYPE: INTEGER REAL physical
            RESULT TYPE: INTEGER REAL physical

 CONCATENATION OPERATOR: &
           OPERAND TYPE: array of any type
            RESULT TYPE: array of any type
```

- Operators exist for logical and arithmetic operations
- Concatenation operator produces array

VHDL_Overview.conclusions

1. Outline: Introduction, Organization, Outline

2. Design_Environments: Digital system design process, Hardware description languages, Hardware simulation, Hardware synthesis

3. VHDL_Background: VHDL initiation, Existing languages, VHDL requirements, The VHDL language, VHDL based design process, Levels of abstraction

4. VHDL_Overview: 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

ERROR: undefined
OFFENDING COMMAND: ;

STACK: