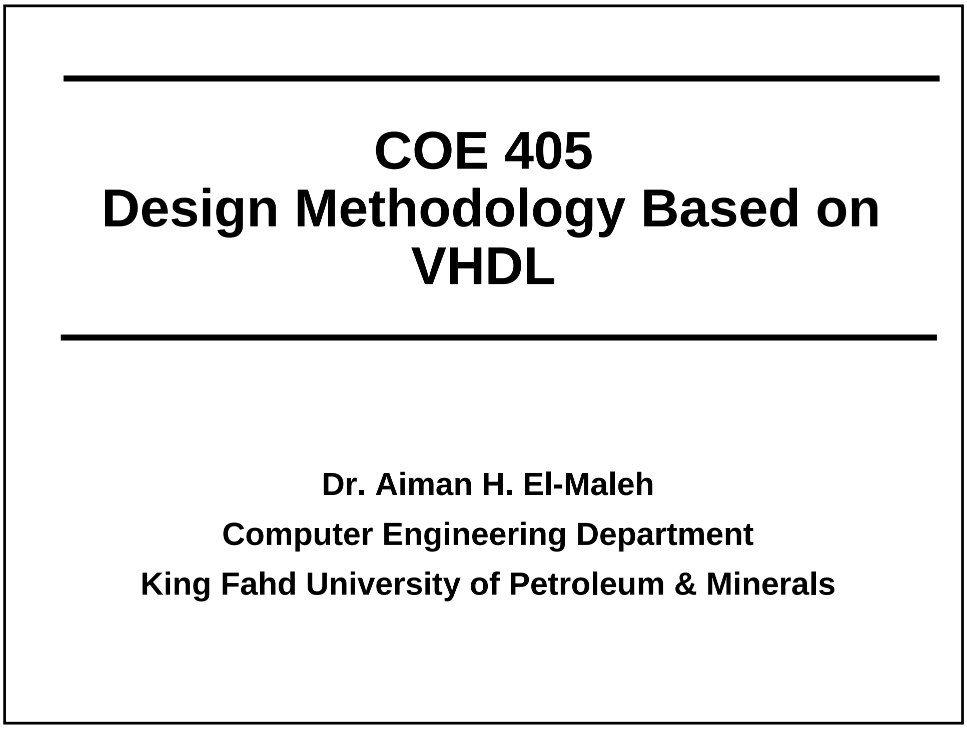
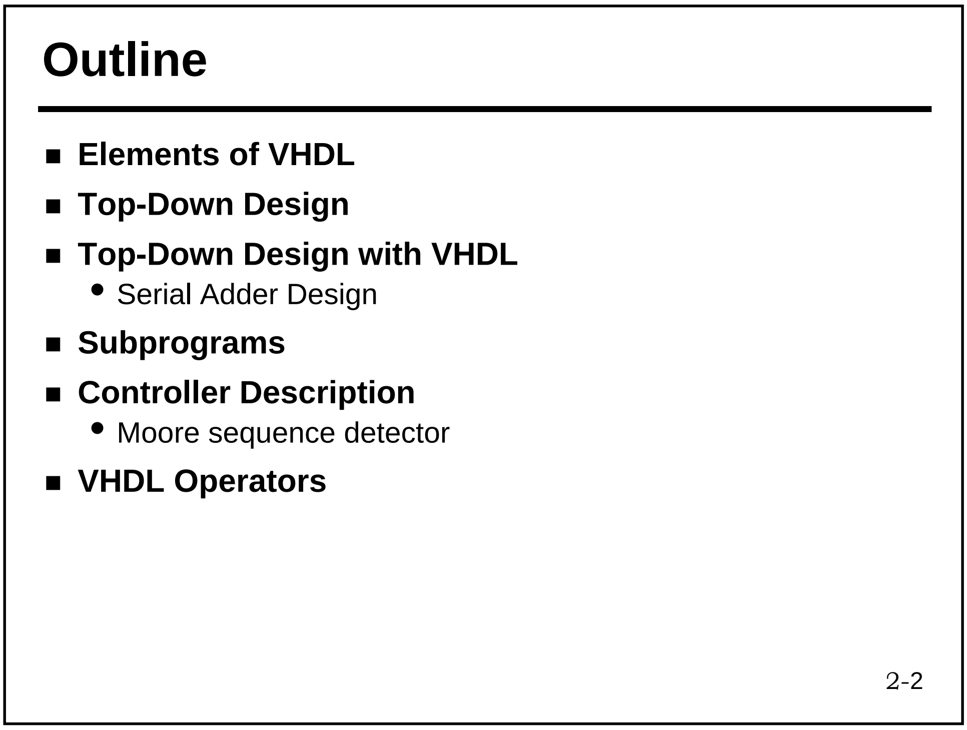


COE 405 Design Methodology Based on VHDL

Dr. Aiman H. El-Maleh

Computer Engineering Department

King Fahd University of Petroleum & Minerals



Outline

- **Elements of VHDL**
- **Top-Down Design**
- **Top-Down Design with VHDL**
  - Serial Adder Design
- **Subprograms**
- **Controller Description**
  - Moore sequence detector
- **VHDL Operators**

Interface and Architectural Specifications

```
Entity component_name IS
        input and output ports
        physical and other parameters
END component_name;
```

```
Architecture identifier of component_name IS
  declarations
BEGIN
        specification of the functionality of the
        component in terms its inputs & influenced
        by physical and other parameters.
END identifier;
```

2-3

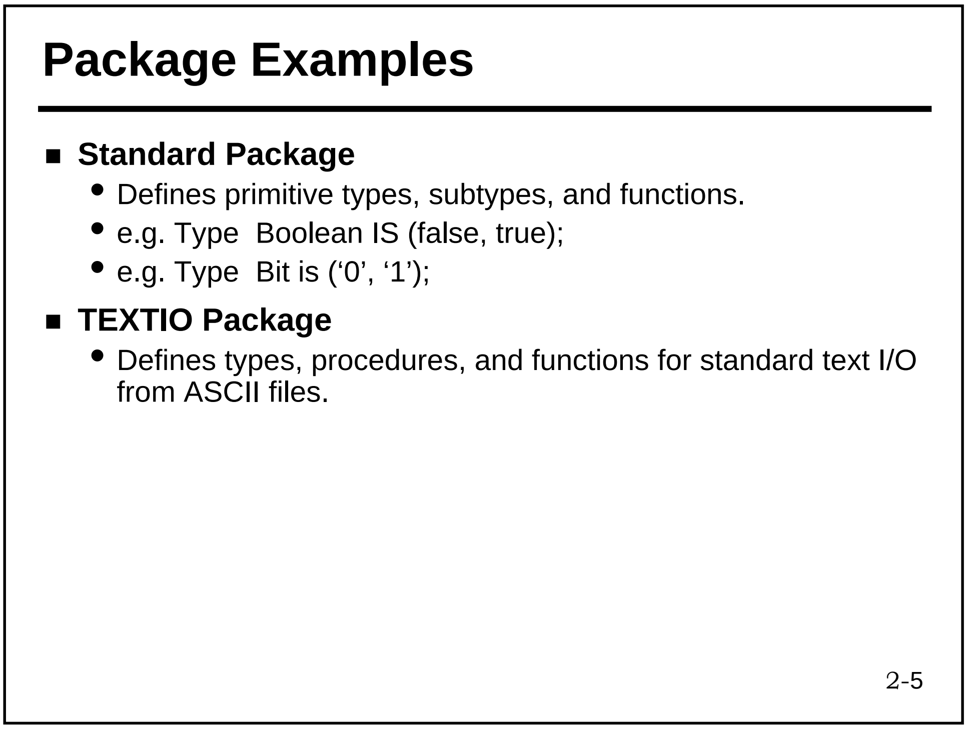
Packages

- Packages are used to encapsulate information that is to be shared among multiple design units.
- A package is used by the USE clause
  - e.g. Use work.package_name.all

```
Package package_name IS
        component declarations
        sub-program parameters
END package_name;
```

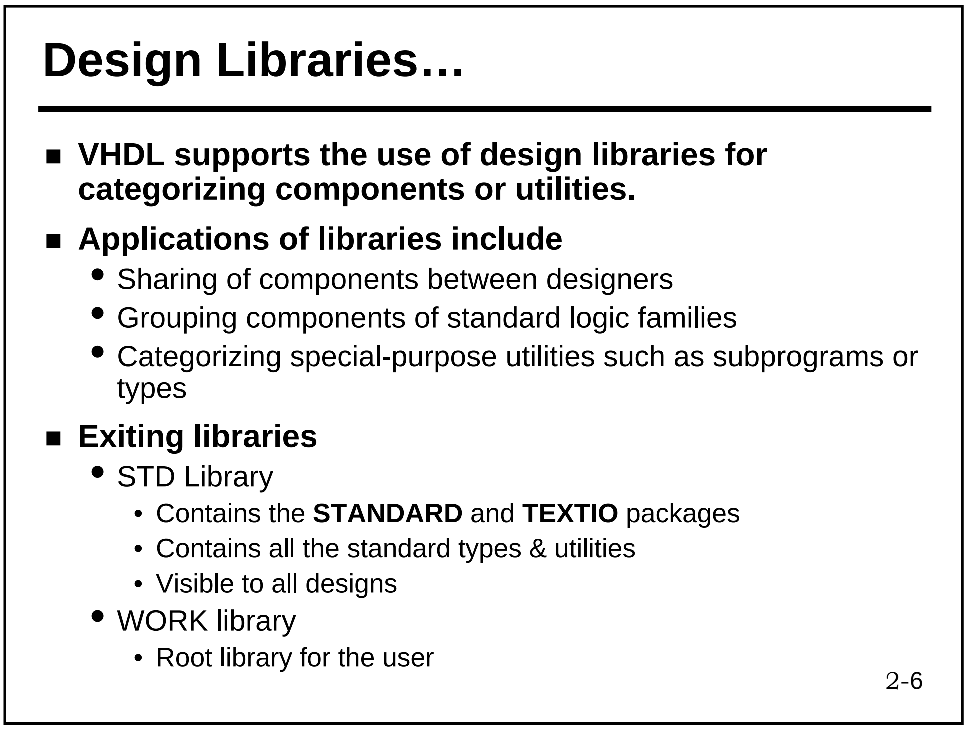
```
Package Body package_name IS
        type definitions
        sub-programs
END package_name;
```

2-4



Package Examples

- **Standard Package**
  - Defines primitive types, subtypes, and functions.
  - e.g. Type Boolean IS (false, true);
  - e.g. Type Bit is (‘0’, ‘1’);
- **TEXTIO Package**
  - Defines types, procedures, and functions for standard text I/O from ASCII files.



Design Libraries…

- **VHDL supports the use of design libraries for categorizing components or utilities.**
- **Applications of libraries include**
  - Sharing of components between designers
  - Grouping components of standard logic families
  - Categorizing special-purpose utilities such as subprograms or types
- **Exiting libraries**
  - STD Library
    - Contains the **STANDARD** and **TEXTIO** packages
    - Contains all the standard types & utilities
    - Visible to all designs
  - WORK library
    - Root library for the user

…Design Libraries

- **IEEE library**
  - Contains VHDL-related standards
  - Contains the std_logic_1164 (IEEE 1164.1) package
    - Defines a nine values logic system
- **To make a library visible to a design**
  - LIBRARY libname;
- **The following statement is assumed by all designs**
  - LIBRARY WORK;
- **To use the std_logic_1164 package**
  - LIBRARY IEEE
  - USE IEEE.std_logic_1164.ALL

Design Binding

- **Using Configuration, VHDL allows**
  - Binding of Entities and Architectures.
  - binding of subcomponents of a design to elements of various libraries.

```
Configuration config_name of component_name IS
        binding of Entities and Architectures
        specifying parameters of a design
        binding components of a library to subcomponents
END config_name;
```

Top-Down Design

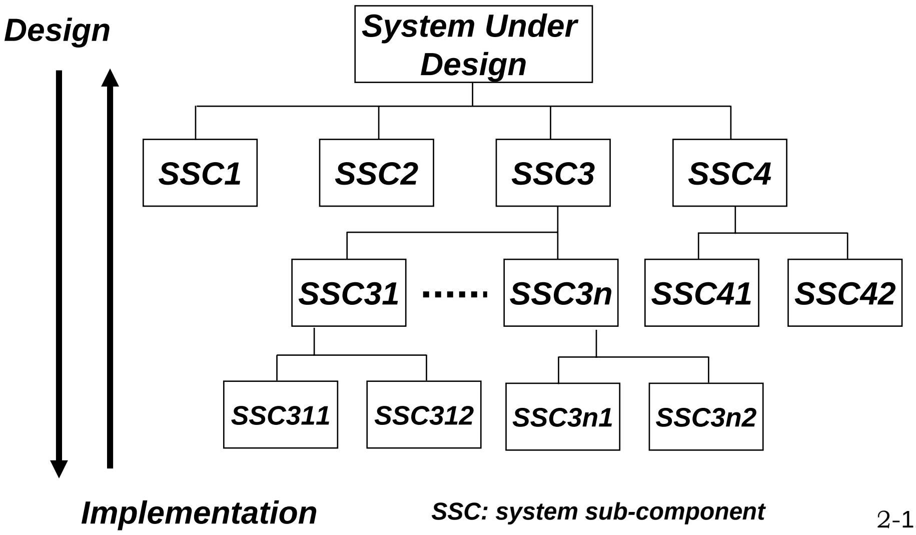
- **Top-down design process uses a divide-and-conquer strategy**
- **Top-down design involves recursive partitioning of a system into subcomponents until**
  - all subcomponents are manageable design parts
- **Design of a component is manageable if the component is**
  - Available as part of a library
  - Can be implemented by modifying an already available part
  - Can be described for a synthesis program or an automatic hardware generator.

Recursive Partitioning Procedure

```
Parition (system)
   IF HardwareMappingOf(system) is done Then
        SaveHardwareof(system)
   ELSE
      FOR EVERY Functionally-Distinct Part_I of System
        Partition (Part_I)
      END FOR;
   END IF;
END Parition;
```

- **Mapping to hardware, depends on target technology, available libraries, and available tools.**

Top-Down Design, Bottom-Up Implementation

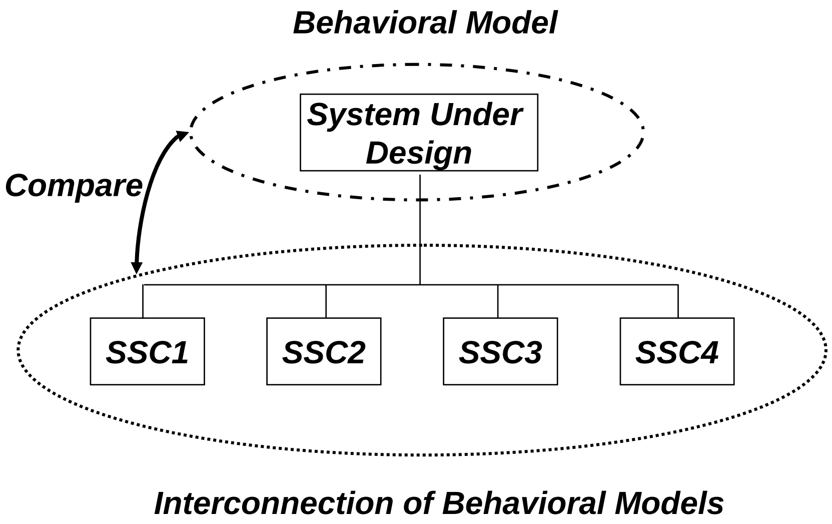


SSC: system sub-component

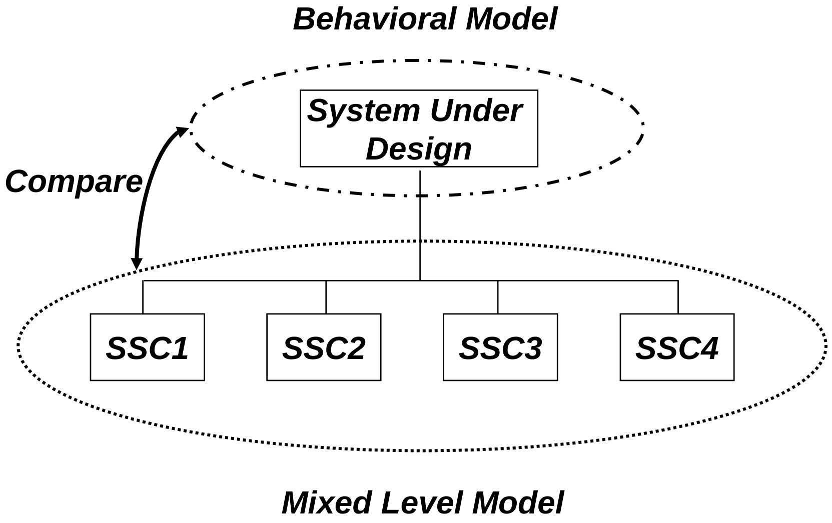
Design Verification

- **Initially, a behavioral description of system under design (SUD) must be simulated to verify design.**
- **After first level of partitioning**
  - A behavioral description of each of the subcomponents must be developed
  - A structural hardware model of the SUD is formed by wiring subcomponents behavioral descriptions
  - Simulating the new model and comparing it with original SUD description verifies correctness of first level of partitioning

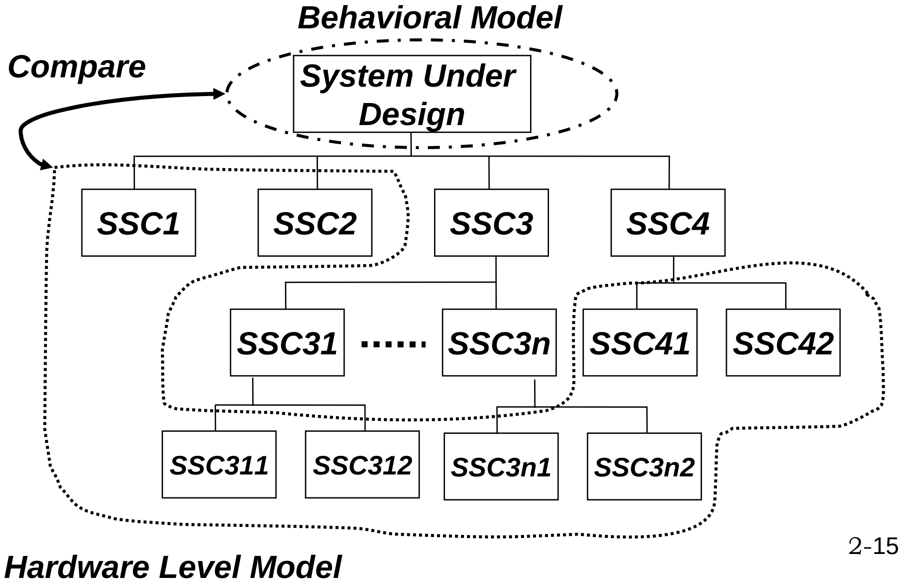
Verifying First Level of Partitioning



Verifying Hardware Implementation of SSC1 and SSC2

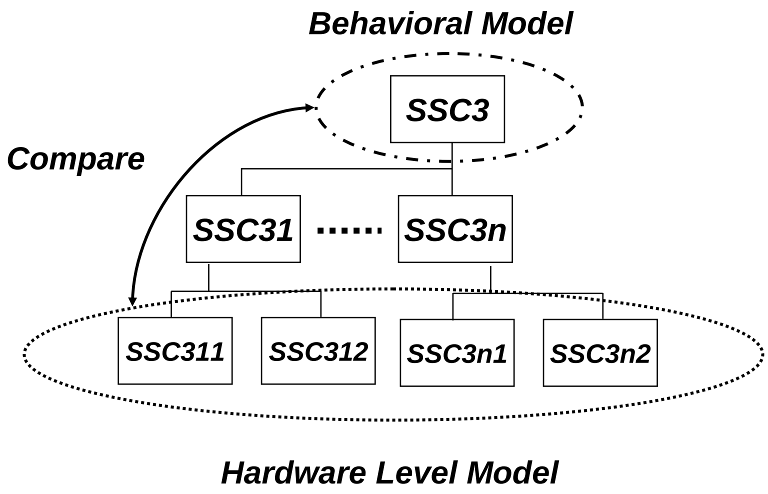


Verifying the Final Design



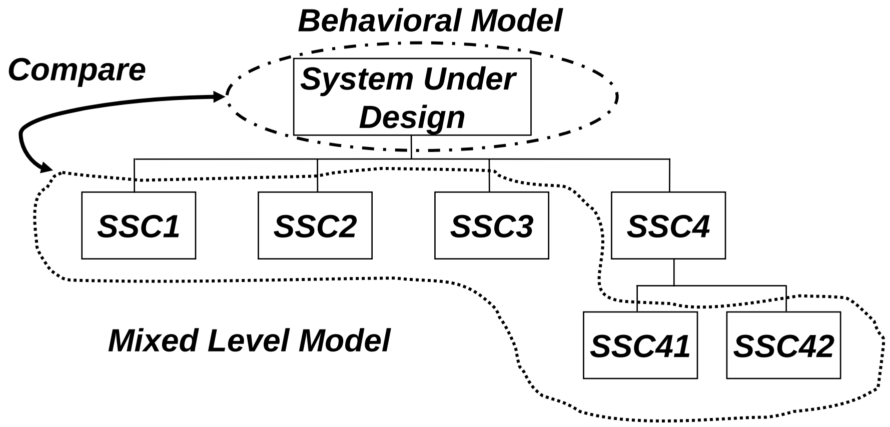
2-15

Verifying Hardware Implementation of SSC3



2-16

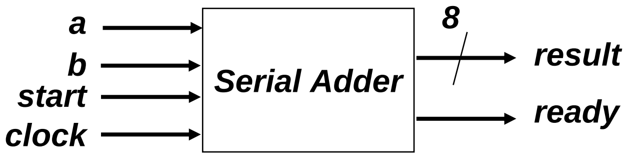
Verifying the Final Design



Back Annotation: Behavioral models of subcomponents can be adjusted to mimic properties of hardware-level models.

Top Down Design of Serial Adder

- Serial data is synchronized with clock and appear on a and b inputs
- Valid data bits on a and b appear after a synchronous pulse on start
- After eight clock pulses, the addition of a and b will appear on result
- The ready output is set to 1 to indicate that the result of addition is ready and remains one until another start pulse is applied



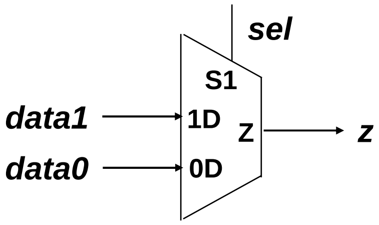
Design Assumptions

- **Synthesis tools:**
  - Capable of synthesizing logic expressions to combinational logic blocks
- **Design Library:**
  - A 2x1 multiplexer with active high inputs and outputs
  - A synchronous D-FF with synchronous active high reset input
  - VHDL models of library elements are provided
- **Parts from previous designs**
  - Divide-by-8 counter (3-bit counter)
    - Synchronous reset
    - Single bit output remains high as long as the circuit is counting; it goes to low when reset is pressed or 8 clock edges are counted

2-19

VHDL Model of 2x1 Multiplexer

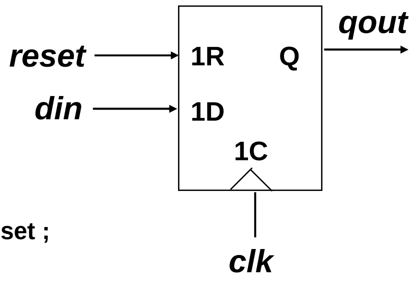
```
Entity mux2_1 IS
   Generic (dz_delay: TIME := 6 NS);
   PORT (sel, data1, data0: IN BIT; z: OUT BIT);
END mux2_1;

Architecture dataflow OF mux2_1 IS
Begin
   z <= data1 AFTER dz_delay WHEN sel=‘1’ ELSE
         data0 AFTER dz_delay;
END dataflow;
```



2-20

VHDL Model of D-FF

```
Entity flop IS
    Generic (td_reset, td_in: TIME := 8 NS);
    PORT (reset, din, clk: IN BIT; qout: OUT BIT :=‘0’);
END flop;
Architecture behavioral OF flop IS
Begin
    Process(clk)
    Begin
        IF (clk = ‘0’ AND clk’Event ) Then
            IF reset = ‘1’ Then
                    qout <= ‘0’  AFTER    td_reset ;
            ELSE
                    qout <= din  AFTER   td_in ;
           END IF;
       END IF;
     END process;
END behavioral ;
```



Divide-by-8 Counter

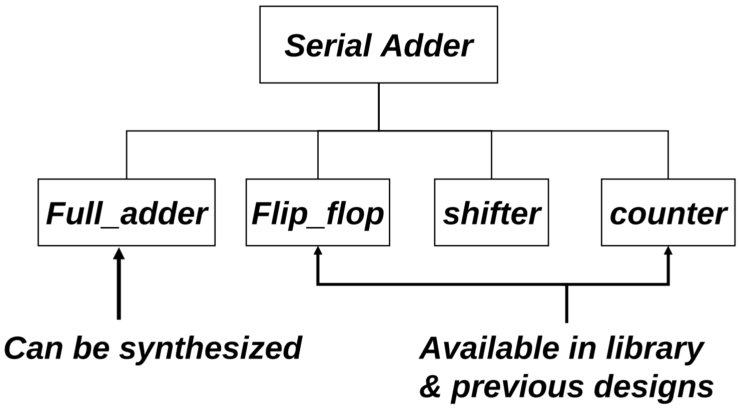
```
Entity counter IS
    Generic (td_cnt: TIME := 8 NS);
    PORT (reset, clk: IN BIT; counting: OUT BIT :=‘0’);
    Constant limit: INTEGER :=8;
END counter ;
Architecture behavioral OF counter IS
Begin
    Process(clk)
          Variable count: INTEGER := limit;
    Begin
       IF (clk = ‘0’  AND  clk’Event ) THEN
            IF reset = ‘1’ THEN           count := 0 ;
            ELSE  IF   count < limit THEN count:= count+1; END IF;
            END IF;
            IF count = limit Then         counting <= ‘0’ AFTER td_cnt;
            ELSE                          counting <= ‘1’ AFTER td_cnt;
           END IF;
      END IF;
   END process;
END behavioral ;
```

11

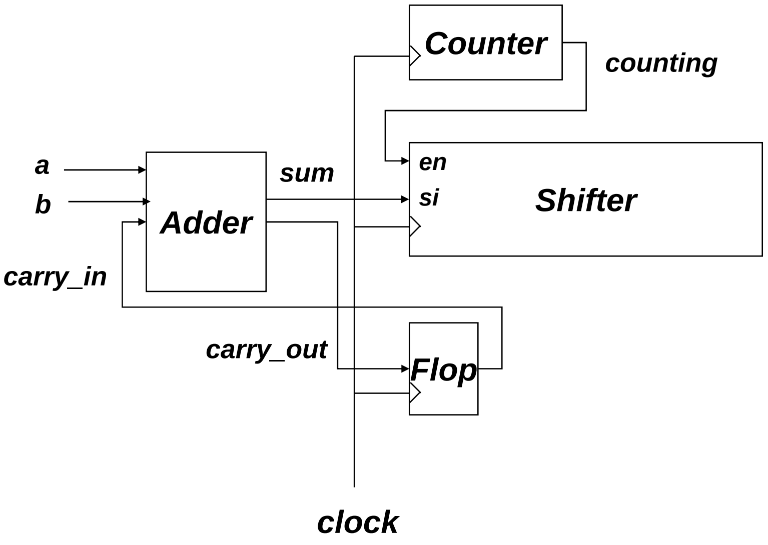
Serial Adder Behavioral Description

```
Entity serial_adder IS
    PORT (a, b, start, clock: IN BIT; ready: OUT BIT; result: BUFFER Bit_Vector (7 downto 0));
END serial_adder ;
Architecture behavioral OF serial_adder IS
Begin
    Process(clock)
          Variable count: INTEGER := 8;          Variable sum, carry: BIT;
    Begin
       IF (clock = ‘0’  AND  clock’Event ) THEN
            IF start = ‘1’ THEN           count := 0 ; carry:=‘0’;
            ELSE  IF   count < 8 THEN
                    count:= count+1;
                    sum := a XOR b XOR carry;
                    carry := (a AND b) OR (a AND carry) OR (b AND carry);
                    result <= sum & result(7 downto 1);
                   END IF;
            END IF;
            IF count = 8 Then  ready <= ‘1’; ELSE ready <= ‘0’;      END IF;
      END IF;
    END process;
END behavioral;
```

Serial Adder First Level of Partitioning

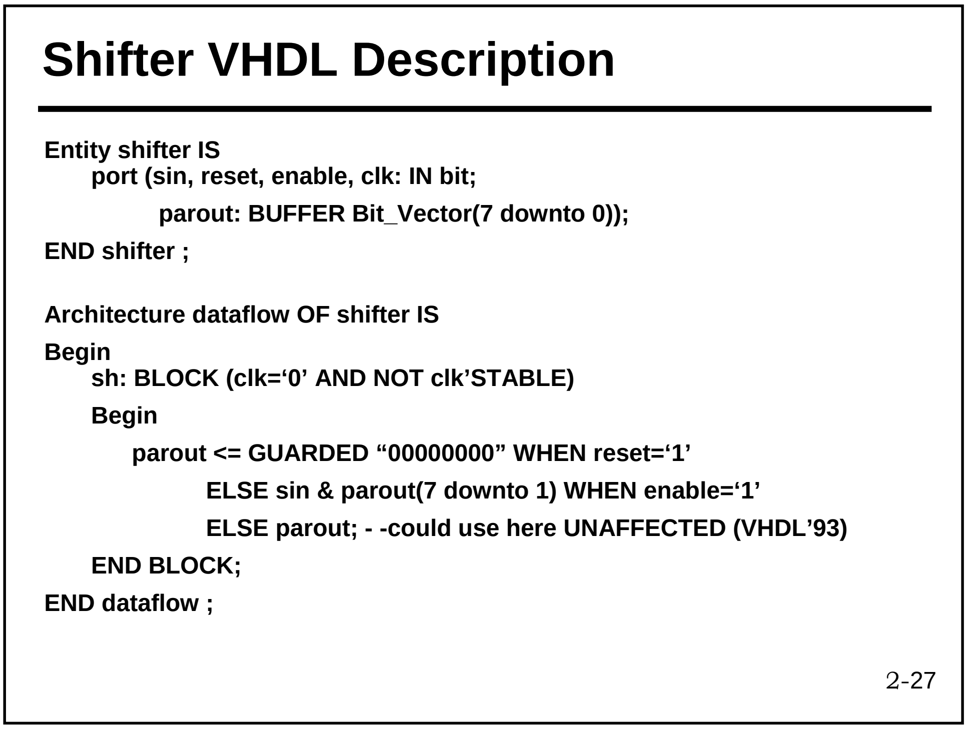


General Layout of Serial Adder



Full-Adder VHDL Description

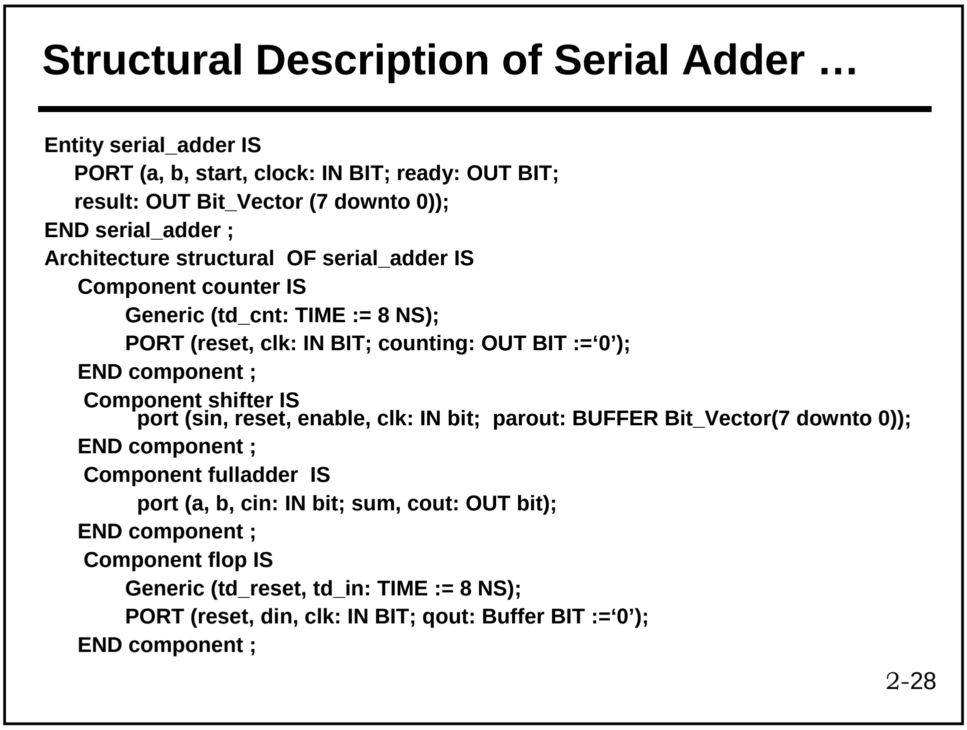
```
Entity fulladder IS
     port (a, b, cin: IN bit; sum, cout: OUT bit);
END fulladder;

Architecture behavioral OF fulladder IS
Begin
     sum <= a xor a xor cin;
     cout <= (a and b) or (b and cin) or (a and cin);
END behavioral ;
```



Shifter VHDL Description

```
Entity shifter IS
      port (sin, reset, enable, clk: IN bit;
               parout: BUFFER Bit_Vector(7 downto 0));
END shifter ;

Architecture dataflow OF shifter IS
Begin
      sh: BLOCK (clk='0' AND NOT clk'STABLE)
      Begin
           parout <= GUARDED "00000000" WHEN reset='1'
                     ELSE sin & parout(7 downto 1) WHEN enable='1'
                     ELSE parout; - -could use here UNAFFECTED (VHDL'93)
      END BLOCK;
END dataflow ;
```

2-27



Structural Description of Serial Adder …

```
Entity serial_adder IS
    PORT (a, b, start, clock: IN BIT; ready: OUT BIT;
    result: OUT Bit_Vector (7 downto 0));
END serial_adder ;
Architecture structural  OF serial_adder IS
     Component counter IS
           Generic (td_cnt: TIME := 8 NS);
           PORT (reset, clk: IN BIT; counting: OUT BIT :='0');
     END component ;
      Component shifter IS
            port (sin, reset, enable, clk: IN bit;  parout: BUFFER Bit_Vector(7 downto 0));
     END component ;
      Component fulladder  IS
            port (a, b, cin: IN bit; sum, cout: OUT bit);
     END component ;
      Component flop IS
           Generic (td_reset, td_in: TIME := 8 NS);
           PORT (reset, din, clk: IN BIT; qout: Buffer BIT :='0');
     END component ;
```

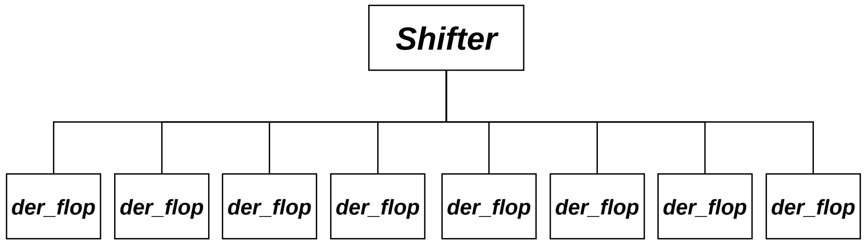
2-28

… Structural Description of Serial Adder

```
SIGNAL sum, carry_in, carry_out, counting: BIT;
Begin
   u1: fulladder port map (a, b, carry_in, sum, carry_out);
   u2: flop  port map (start, carry_out, clock, carry_in);
   u3: counter port map (start, clock, counting);
   u4: shifter port map (sum, start, counting, clock,
   result);
   u5: ready <= NOT counting;
END structural ;
```

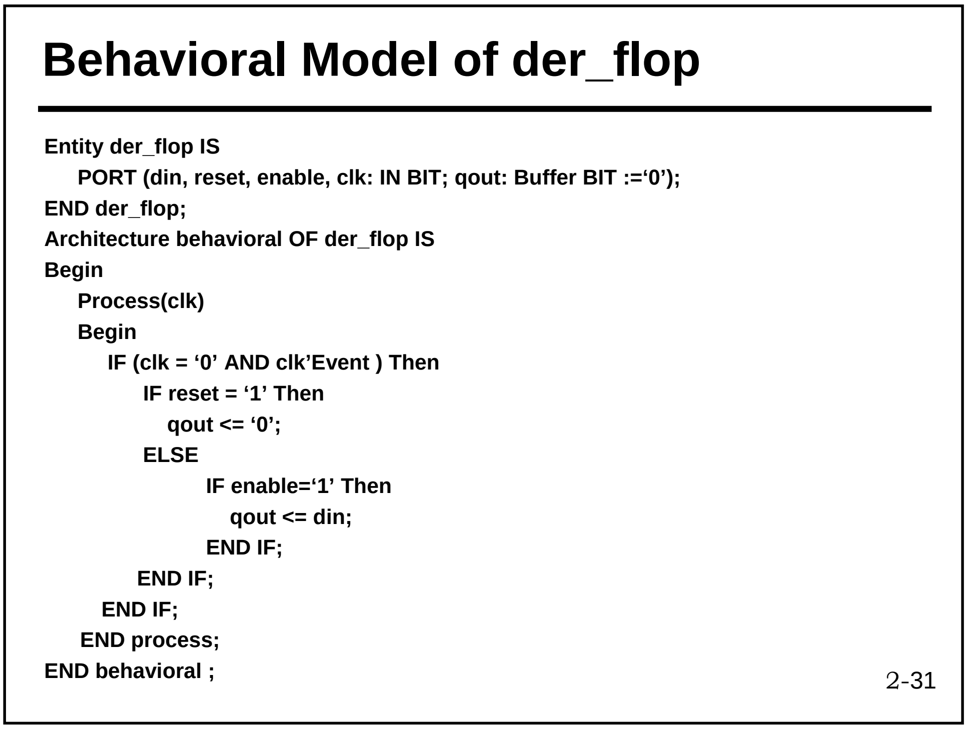
Second Level of Partitioning: Partitioning Shifter

- Shifter needs eight flip-flops with synchronous reset and clock enabling (der_flop)
- der_flop cannot be directly implemented with given tools and library



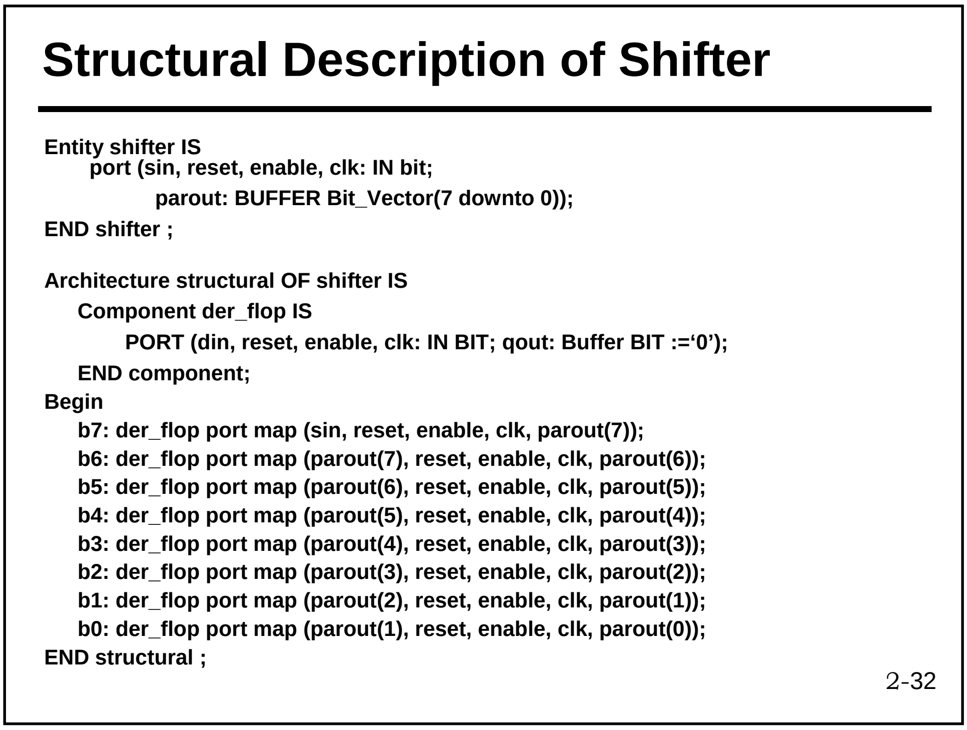
Shifter

der_flop *der_flop* *der_flop* *der_flop* *der_flop* *der_flop* *der_flop* *der_flop*



Behavioral Model of der_flop

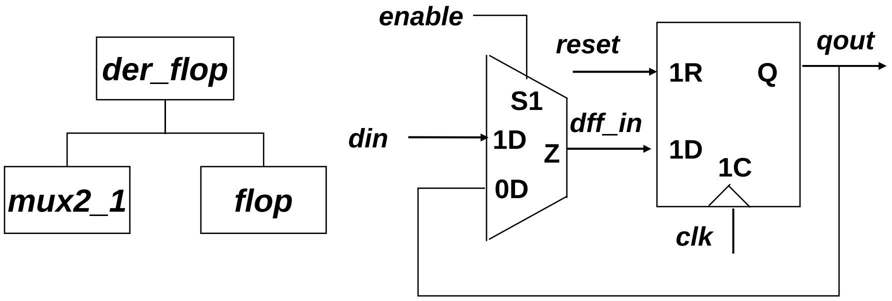
```
Entity der_flop IS
    PORT (din, reset, enable, clk: IN BIT; qout: Buffer BIT :=‘0’);
END der_flop;
Architecture behavioral OF der_flop IS
Begin
    Process(clk)
    Begin
        IF (clk = ‘0’ AND clk’Event ) Then
            IF reset = ‘1’ Then
               qout <= ‘0’;
            ELSE
                    IF enable=‘1’ Then
                       qout <= din;
                    END IF;
            END IF;
        END IF;
    END process;
END behavioral ;
```



Structural Description of Shifter

```
Entity shifter IS
     port (sin, reset, enable, clk: IN bit;
             parout: BUFFER Bit_Vector(7 downto 0));
END shifter ;

Architecture structural OF shifter IS
    Component der_flop IS
          PORT (din, reset, enable, clk: IN BIT; qout: Buffer BIT :=‘0’);
    END component;
Begin
    b7: der_flop port map (sin, reset, enable, clk, parout(7));
    b6: der_flop port map (parout(7), reset, enable, clk, parout(6));
    b5: der_flop port map (parout(6), reset, enable, clk, parout(5));
    b4: der_flop port map (parout(5), reset, enable, clk, parout(4));
    b3: der_flop port map (parout(4), reset, enable, clk, parout(3));
    b2: der_flop port map (parout(3), reset, enable, clk, parout(2));
    b1: der_flop port map (parout(2), reset, enable, clk, parout(1));
    b0: der_flop port map (parout(1), reset, enable, clk, parout(0));
END structural ;
```

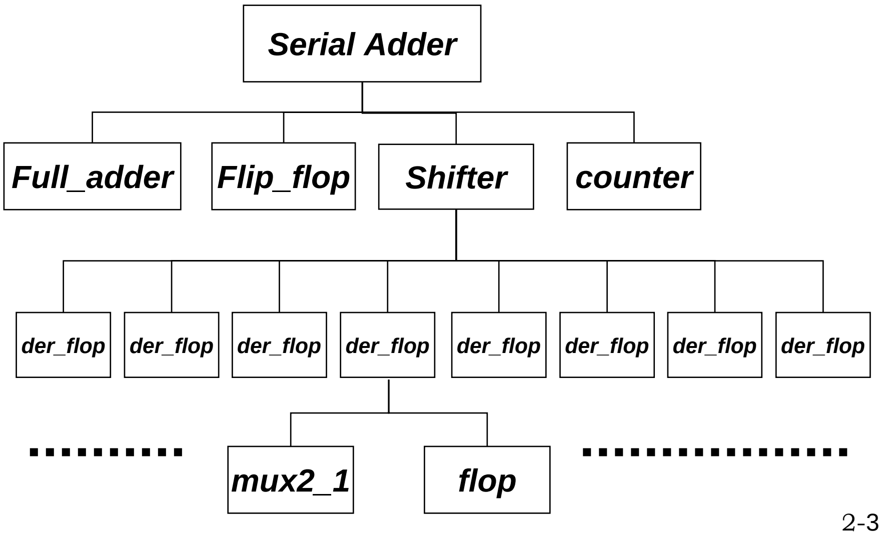
Partitioning der_flop



Structural Description of der_flop

```
Entity der_flop IS
    PORT (din, reset, enable, clk: IN BIT; qout: Buffer BIT :=‘0’);
END der_flop;
Architecture structural OF der_flop IS
    Component flop  IS
          Generic (td_reset, td_in: TIME := 8 NS);
          PORT (reset, din, clk: IN BIT; qout: Buffer BIT :=‘0’);
    END component ;
    Component mux2_1 IS
          Generic (dz_delay: TIME := 6 NS);
          PORT (sel, data1, data0: IN BIT; z: OUT BIT);
    END component ;
Signal dff_in: BIT;
Begin
    mx: mux2_1 port map (enable, din, qout, dff_in);
    ff: flop port map (reset, dff_in, clk, qout);
END structural ;
```

Complete Design of Serial Adder



2-35

Synthesizable Serial Adder

```
Entity serial_adder IS
    PORT (a, b, start, clock: IN BIT; ready: OUT BIT; result: BUFFER Bit_Vector (7 downto 0));
END serial_adder ;
Architecture syn OF serial_adder IS
Begin
    Process(clock)
          subtype CNT8 IS INTEGER Range 0 to 8;
          Variable count: CNT8 := 8;  Variable sum, carry: BIT;
    Begin
       IF (clock = ‘0’  AND  clock’Event ) THEN
            IF start = ‘1’ THEN          count := 0 ; carry:=‘0’;
            ELSE  IF   count < 8 THEN
                    count:= count+1;
                    sum := a XOR b XOR carry;
                    carry := (a AND b) OR (a AND carry) OR (b AND carry);
                    result <= sum & result(7 downto 1);
                   END IF;
            END IF;
            IF count = 8 Then  ready <= ‘1’; ELSE ready <= ‘0’;      END IF;
       END IF;
    END process;
END syn;
```

2-36

Subprograms

- VHDL allows the use of functions and procedures
- Most of high-level behavioral constructs in VHDL can be used in body of functions and procedures
- Subprograms can be declared, defined, and invoked as in software languages
- Functions can be used
  - to represent Boolean equations
  - for type conversions
  - for delay value calculations
- Procedures can be used for
  - type conversion
  - description of counters
  - outputting internal binary data in integer form

Procedure Example

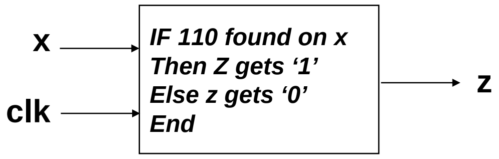
```
Type Byte ARRAY (7 Downto 0) of BIT;
Procedure byte_to_integer(ib: IN Byte; oi: OUT Integer) IS
Variable result: Integer :=0;
Begin
  For I IN 0 To 7 Loop
      IF ib(I) = ‘1’ Then
        result := result + 2**I;
     END IF;
   END Loop;
   oi := result;
END byte_to_integer;
```

Function Example

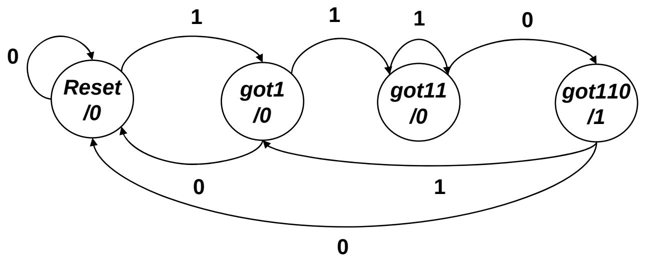
```
Entity fulladder IS
   Port (a, b, cin: IN BIT; sum, cout: OUT BIT);
END fulladder;
Architecture behavioral OF  fulladder IS
   Function fadd(a, b, c : IN BIT) RETURN Bit_Vector IS
        variable sc: Bit_Vector(1 downto 0);
   Begin
        sc(1) := a XOR b XOR c;
        sc(0) := (a AND b) OR (a AND c) OR (b AND c);
        Return sc;
   END;
Begin
   (sum, cout) <= fadd(a, b, cin);
END behavioral ;
```

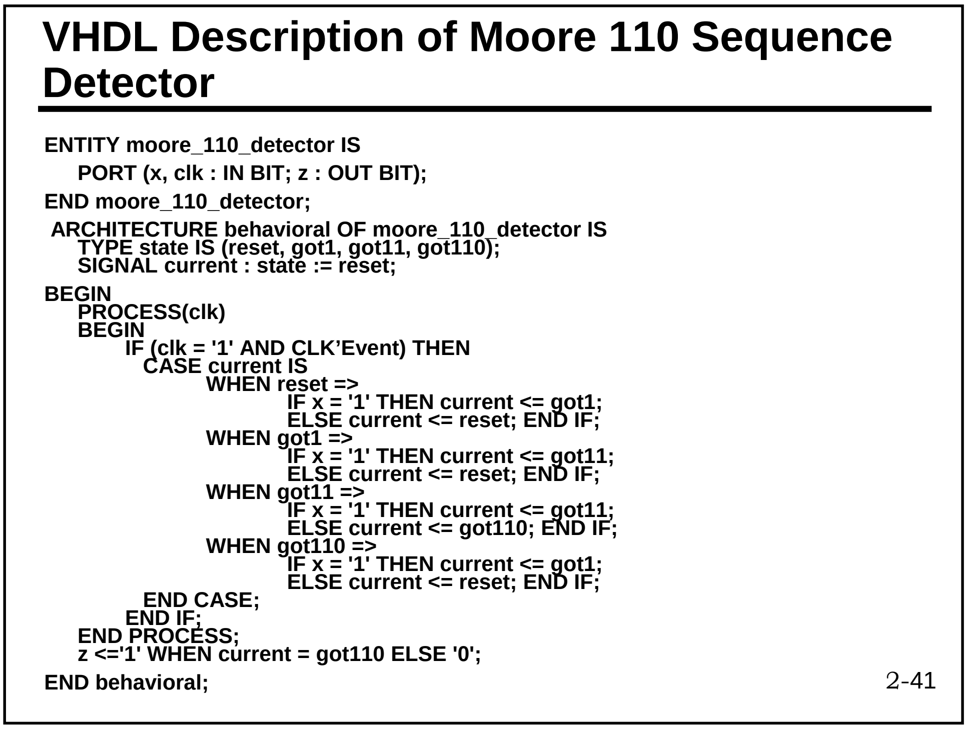
Controller Description

- **Moore Sequence Detector**
  - Detection sequence is 110



IF 110 found on x
Then Z gets ‘1’
Else z gets ‘0’
End

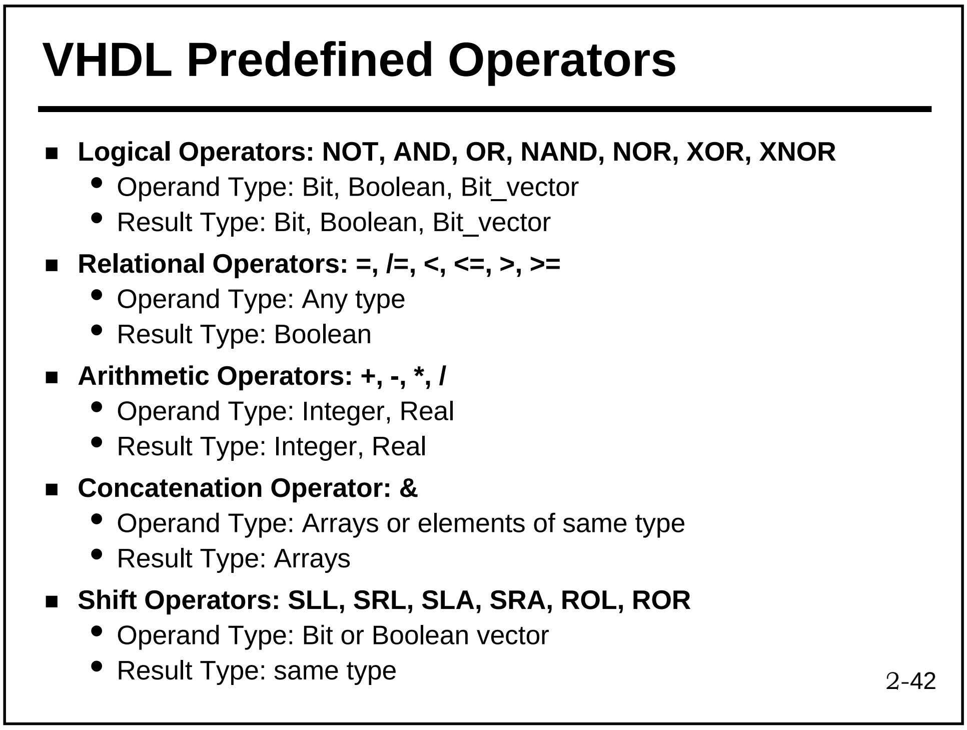




VHDL Description of Moore 110 Sequence Detector

```
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, got1, got11, got110);
    SIGNAL current : state := reset;
BEGIN
    PROCESS(clk)
    BEGIN
         IF (clk = '1' AND CLK'Event) THEN
           CASE current IS
                  WHEN reset =>
                           IF x = '1' THEN current <= got1;
                           ELSE current <= reset; END IF;
                  WHEN got1 =>
                           IF x = '1' THEN current <= got11;
                           ELSE current <= reset; END IF;
                  WHEN got11 =>
                           IF x = '1' THEN current <= got11;
                           ELSE current <= got110; END IF;
                  WHEN got110 =>
                           IF x = '1' THEN current <= got1;
                           ELSE current <= reset; END IF;
           END CASE;
         END IF;
    END PROCESS;
    z <='1' WHEN current = got110 ELSE '0';
END behavioral;
```

2-41



VHDL Predefined Operators

- **Logical Operators: NOT, AND, OR, NAND, NOR, XOR, XNOR**
  - Operand Type: Bit, Boolean, Bit_vector
  - Result Type: Bit, Boolean, Bit_vector
- **Relational Operators: =, /=, <, <=, >, >=**
  - Operand Type: Any type
  - Result Type: Boolean
- **Arithmetic Operators: +, -, *, /**
  - Operand Type: Integer, Real
  - Result Type: Integer, Real
- **Concatenation Operator: &**
  - Operand Type: Arrays or elements of same type
  - Result Type: Arrays
- **Shift Operators: SLL, SRL, SLA, SRA, ROL, ROR**
  - Operand Type: Bit or Boolean vector
  - Result Type: same type

2-42