Overview

- Design structure organization
  - Packages
  - Configurations
  - Libraries
- Design modeling
  - Simulation aspects
  - Synthesis annotations
- Verification
  - Mixed level simulation
  - Test bench

Language Elements

- Modularity is the principle of partitioning the hardware design and associated VHDL description into smaller units, into many files that can be separately compiled and verified
  - VHDL supports **five kinds** of design units that can be independently handled (compiled)

Design Units

- Primary Unit ➠ a library unit which can exist in a design library
  - Package declaration
  - Entity
  - Configuration
- Primary Units provide the definition of the interface of the system being designed, with the outside world

Design Units

- Secondary Unit ➠ a library unit that defines the body associated with a primary unit that has already been analyzed into the same design library
  - Package body (associated with a Package)
  - Architecture (associated with an Entity)

Package Declaration

- It is a Primary Library Unit
- Declarations include any of the following:
  - type
  - subtype
  - constant
  - file
  - alias
  - component
  - attribute
  - function
  - procedure
- A **`use`** clause is necessary to access the package elements from other design units

Entity

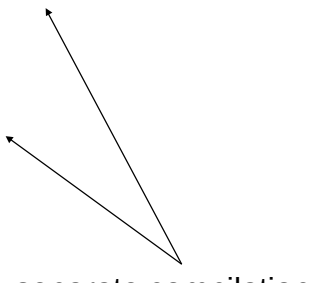
- It is a Primary Library Unit
- Defines the component (name) and its interface in terms of inputs, outputs and the types of signals
- Allows the definition of parametric aspects by means of the **`generic`** clause

Configuration

- It is a Primary Library Unit
- Defines the ***binding*** of each component to an **`entity`**, and each entity to an **`architecture`**
- If not used, default binding will occur:
  - For each unbound instance of every component, an entity matching name, port names and types will be selected
  - The last analyzed architecture will be selected, if several are available

Secondary Library Units

- They are expression of the primary units
- **`Package body`**
  - code performed by the subprograms
- **`Architecture`**
  - different approaches
  - different details
  - different specification
  - different abstraction



Library

- It can be used in:
  - Entity
  - Package
  - Configuration
- It is valid only for:
  - the primary unit ***immediately*** following it
  - any of its secondary units

to be repeated for ***any*** primary unit

```
LIBRARY lib_name, another_lib_name;
```


Library

- Each tool provides a way to map the library logical name to its physical position:
  - shell variables
  - file system links
  - configuration file
- Default libraries have been compiled by the tools vendor
- Libraries **WORK** and **STD** are never declared

```
LIBRARY work;
LIBRARY std;
USE std.standard.ALL;
```

implicitly assumed

Library Scope

- Save design efforts by ***reusing*** objects
  - Hardware components (design entities):
    - Entity declaration
    - Architecture body
    - Configuration
  - Software items (types, procedures, …):
    - Package declaration
    - Package body

Beh. GCD Exemplification

```
ENTITY gcd IS
       PORT (  clock, reset : IN bit;
               xi,yi : IN INTEGER;
               ou : OUT INTEGER
       );
END gcd;
ARCHITECTURE behavioral OF gcd IS
BEGIN
       PROCESS
               VARIABLE x, y,temp : INTEGER;
       BEGIN
               WAIT UNTIL clock = '1';
               x := xi;
               y := yi;
               WHILE (x > 0) LOOP
                       IF (x <= y) THEN
                               temp := y;
                               y := x;
                               x := temp;
                       END IF;
                       x := x - y;
               END LOOP;
               ou <= y;
       END PROCESS;
END behavioral;
```

Beh. GCD Exemplification

- Size modularity:
  - package definition

```
PACKAGE gcd_pack IS
       CONSTANT SIZE : INTEGER := 8;
END gcd_pack;

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.gcd_pack.ALL;
ENTITY gcd IS
       PORT (  clock, reset : IN STD_LOGIC;
               xi,yi : IN  UNSIGNED (SIZE-1 DOWNTO 0);
               ou    : OUT UNSIGNED (SIZE-1 DOWNTO 0)
       );
END gcd;
```

even if the package is in the same file

Beh. GCD Exemplification

- Package definition alternatives:
  - same file of the design entity
    - only if definitions concern **one** design entity
      - \+ simple managing (compilation, backup, ...)
      - – useless recompilations
  - separate file
    - definitions shared between multiple design entities
      - \+ modularity, avoid useless recompilations
      - – compilation order to be respected

Signal/Variable Type

- **`integer`**, **`boolean`** ➠ higher abstraction levels
  - \+ easy simulation
  - – no complete synthesis control
- **`bit`**, **`bit_vector`** ➠ lower abstraction levels
  - \+ difficult simulation
  - – complete synthesis control

Signal/Variable Type

- **`Bit`** versus **`std_logic`**
  - faster simulation / inaccurate results

```
LIBRARY STD;
```

automatically linked

  - slower simulation / more information

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
```

std_logic and std_logic_vector definition

std_logic_vector operations definition

Signal/Variable Type

- Signed versus **unsigned**
  - **bit_vector** and **std_logic_vector** are considered by default representing signed number (2-complement representation)
  - unsigned operation required **unsigned** type

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
```

unsigned type and operations definition

why?

Beh. GCD Exemplification

- Algorithm works with positive numbers only:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE WORK.gcd_pack.ALL;
ENTITY gcd IS
       PORT (  clock, reset : IN STD_LOGIC;
               xi,yi : IN  UNSIGNED (SIZE-1 DOWNTO 0);
               ou    : OUT UNSIGNED (SIZE-1 DOWNTO 0));
END gcd;
```

- Operations are correct if bit vectors are considered without sign

```
WHILE (x > 0) LOOP
      IF (x <= y) THEN
```

Size Customization

- Signal variable size can be defined during component instantiation:
  - use **`GENERIC`** keyword
  - default values assigned if component not instantiated
  - \+ reuse of the same component in a different contest
  - – no synthesis rules compliant (all used synthesis tools accept it)

Beh. GCD Exemplification

- Entity modification:

```
ENTITY gcd IS
       GENERIC (SIZE : INTEGER := 8);
       PORT (  clock, reset : IN STD_LOGIC;
              xi,yi : IN  UNSIGNED (SIZE-1 DOWNTO 0);
              ou    : OUT UNSIGNED (SIZE-1 DOWNTO 0));
END gcd;
```

- Component definition

Where is component definition?

```
COMPONENT gcd_comp
       GENERIC (SIZE : INTEGER := 8);
       PORT ( clock, reset : IN STD_LOGIC;
              xi,yi : IN  UNSIGNED (SIZE-1 DOWNTO 0);
              ou    : OUT UNSIGNED (SIZE-1 DOWNTO 0));
END COMPONENT;
```

Beh. GCD Exemplification

- Component/design-entity binding:

```
ARCHITECTURE structural OF ...
        ...
        FOR gcd8, gcd16: gcd_comp USE ENTITY gcd (behavioral);
        ...
```

Not mandatory if component name equal to design-entity name

- Component instantiation

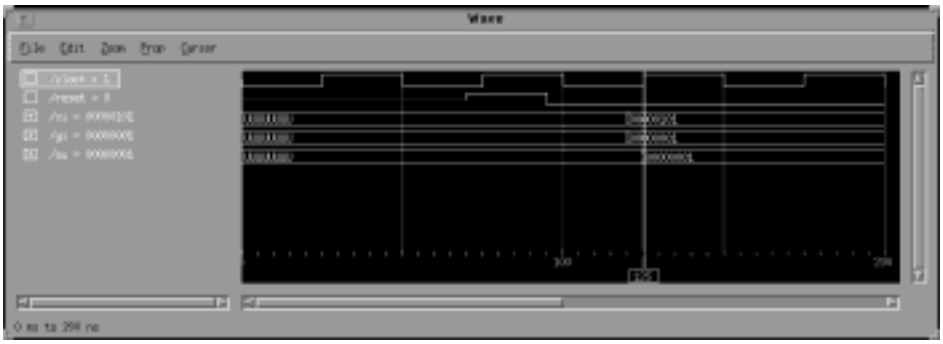
```
BEGIN
        gcd8: gcd_comp
                PORT MAP (clock, reset, x1_8, x2_8, o1_8);
        gcd16: gcd_comp
                GENERIC MAP (16)
                PORT MAP (clock, reset, x1_16, x2_16, o1_16);
END structural;
```

Default generic value

Behavioral Description Style

- Useful for:
  - specification analysis
  - rapid prototyping
  - work-group design
  - approximated simulation
- Useless for
  - automatic synthesis (up to now)
  - exact simulation

Behavioral simulation



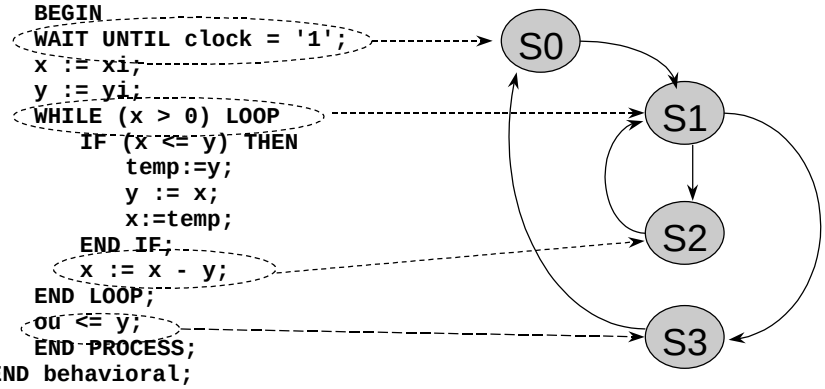
- Output is generated in the same time the clock is asserted and new values are ready
- What's the signal type?

RTL Description Style

- Useful for:
  - automatic synthesis
    - complete control of the synthesis results
    - acceptable estimation of structural properties
  - test bench generation and reuse
- Drawbacks
  - modeling of details concerning the chosen architecture
    - e.g. states of the FSM

RTL GCD Exemplification

```
ARCHITECTURE behavioral OF gcd IS
BEGIN
   PROCESS
      VARIABLE x, y,temp : unsigned (size-1 DOWNTO 0);
   BEGIN
   WAIT UNTIL clock = '1';
   x := xi;
   y := yi;
   WHILE (x > 0) LOOP
       IF (x <= y) THEN
           temp:=y;
           y := x;
           x:=temp;
       END IF;
       x := x - y;
   END LOOP;
   ou <= y;
   END PROCESS;
END behavioral;
```



Finite State Machine Description

- Characteristics:
  - an explicit finite set of states is identified
  - at each clock cycle a new state is selected
  - at each clock cycle outputs are evaluated
  - a reset signal put the FSM in the reset state
- VHDL consideration:
  - a sequence of operations required a *sequential description style* (e.g. a process)

Finite State Machine Description

- Synthesis tool considerations:
  - there are some templates to describe FSMs which are recognized by synthesis tools
  - each template produces different gate-level implementations
- FSM versus FSMD:
  - complex operations can be inserted into each transition instead of direct input/output mapping

RTL GCD Exemplification

- Clock synchronization:
  - behavioral style:

```
PROCESS
        VARIABLE x, y, temp : INTEGER;
BEGIN
        WAIT UNTIL clock = '1';
```

  - RTL style:

```
PROCESS(clock, reset)
        VARIABLE x, y, temp : UNSIGNED (SIZE-1 DOWNTO 0);
BEGIN
        IF (clock'EVENT AND clock = '1') THEN
                IF (reset = '1') THEN
                        reset state
                ELSE
                        next-state evolution
                END IF;
        END IF;
END PROCESS;
```

RTL GCD Exemplification

- States enumeration:

```
PACKAGE gcd_pack IS
        CONSTANT SIZE : INTEGER := 8;
        TYPE state_code IS (S0, S1, S2, S3);
END gcd_pack;
...
PROCESS(clock, reset)
        VARIABLE x, y, temp : UNSIGNED (SIZE-1 DOWNTO 0);
        VARIABLE state : state_code;
BEGIN
        IF (clock'EVENT AND clock = '1') THEN
                IF (reset = '1') THEN
                        state := S0;
```

- Symbolic states or coded states?

RTL GCD Exemplification

- States evolution:

```
IF (clock'EVENT AND clock = '1') THEN
        IF (reset = '1') THEN
                state := S0;
        ELSE
                CASE state IS
                WHEN S0 =>
                        state := S1;
                WHEN S1 =>
                        IF (condition) THEN
                                state := S2;
                        ELSE
                                state := S3;
                        END IF;
                WHEN S2 =>
                        state := S1;
                WHEN S3 =>
                        ...
                END CASE;
        END IF;
END IF;
```

Equivalent to behavioral `(WHILE x > 0) LOOP`

behavioral `LOOP exit`

RTL GCD Exemplification

- Behavioral condition:

```
WHILE (x > 0) LOOP
```

Integer values

- RTL condition:

```
IF (x > 0) THEN
```

Wrong comparison
UNSIGNED / integer

```
IF (x > conv_unsigned(0, SIZE)) THEN
```

Converts integer to a vector of **std_logic** values of size **SIZE**

Defined in package **std_logic_arith**

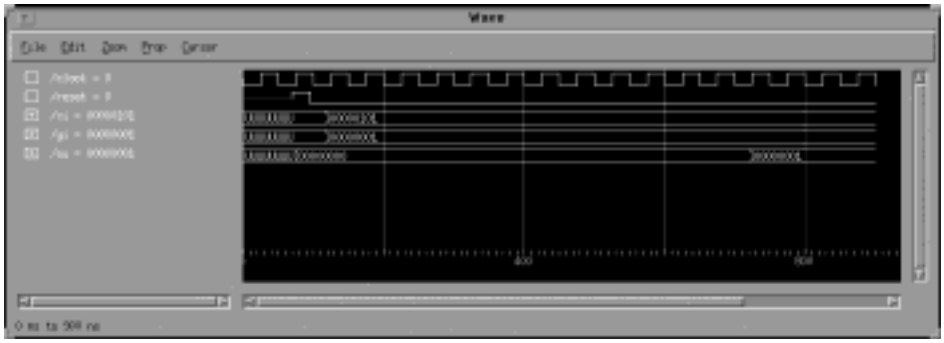
RTL GCD Exemplification

- Signal initialization:
  - Behavioral level
    - **OU** is initialized to the default level
  - RTL level
    - signals initialization implies or not memory elements (see synthesis lecture)

```
IF (reset = '1') THEN
       state := S0;
       ou <= (others => '0');
ELSE
```

Independent on signal size

RTL Simulation

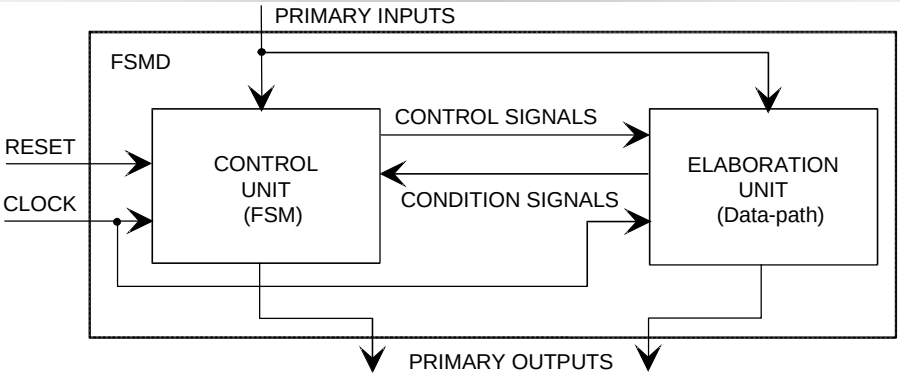


- Output generation required some clock cycles

Synthesis Annotation

- FSMD model is difficult to be automatically synthesized:
  - synthesis tool does not automatically isolate the FSM from the data-path
  - "Ad-hoc" synthesis algorithms for FSMs cannot be applied ⇛ low optimization level
- manual separation of FSM from Data-path

Synthesis Annotation



- Identification of control/condition signals
- Instantiation of two components (FSM + Data-Path)

Condition signals

```
...
CASE state IS
        ...
        WHEN S1 =>
                IF (x > conv_unsigned(0, SIZE)) ) THEN
                        state := S2;
                ELSE
                        state := S3;
                END IF;

        ...
        WHEN S1 =>
                IF (x_cond = '1' ) THEN
                        state := S2;
                ELSE
                        state := S3;
                END IF;
```

FSMD *condition* (some bits)

FSM *condition* (1 bit size)

Control signals

```
...
CASE state IS
        ...
        WHEN S2 =>
                x := x -y;

        ...
        WHEN S2 =>
                sub_control <= '1';
```

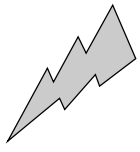
FSMD
operation
(some bits)

FSM
control
(1 bit size)

- control signals must be asserted only in such states where the operation must be executed
- control signal must be set to '0' in all other states

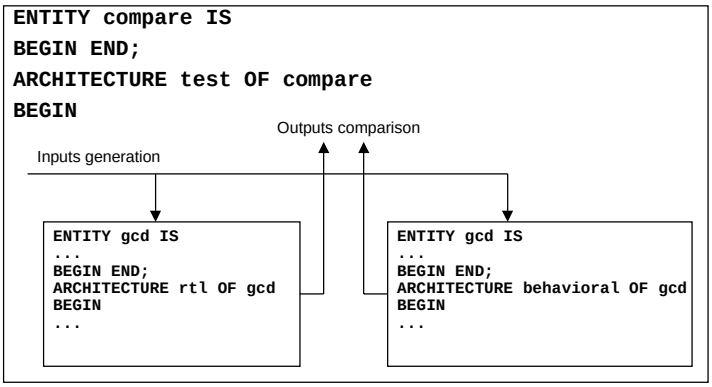
Mixed Level Verification

- Verification of the RT level description
- Comparison specification/implementation
- Hypotheses:
  - behavioral level specification is correct
  - timing aspects are not considered
  - an *acceptable* amount of input vectors guarantee the correctness of the implementation



Mixed Level Verification

- Strategy:



Mixed Level Verification

- Inputs generation:
  - a process without sensitivity list with one or more `wait` statements for each test vector

```
vectors_generator: PROCESS
BEGIN
       reset <= '0';
       xi_rtl <= conv_unsigned(5, SIZE));
       xi_beh <= conv_unsigned(5, SIZE));
       yi_rtl <= conv_unsigned(1, SIZE));
       yi_beh <= conv_unsigned(1, SIZE));
       wait for 1000 ns;
       ...
```

Assumption: RTL module has completed the computation

Mixed Level Verification

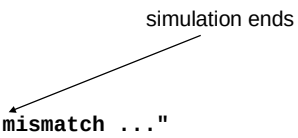
- Inputs generation:
  - the clock signal can be generated by using an "ad-hoc" process

```
ARCHITECTURE test OF compare
        SIGNAL clock : STD_LOGIC;
BEGIN
...
clock_generator: PROCESS
BEGIN
       clock <= '0';
       wait for 25 ns;
       clock <= '1';
       wait for 25 ns;
END PROCESS;
...
```


Mixed Level Verification

- Outputs comparison:
  - the *correct* specification output is compared to the possibly wrong implementation output

```
vectors_generator: PROCESS
       VARIABLE o_rtl, o_beh: UNSIGNED (SIZE-1 DOWNTO 0);
BEGIN
       ...
       wait for 1000 ns;
       o_rtl := ou_rtl;
       o_beh := ou_beh;
       ASSERT o_rtl = o_beh
              REPORT "outputs mismatch ..."
              SEVERITY error;
       ...
```



Mixed Level Verification

- Multiple test vectors application:
  - build a textual file with pairs of inputs (e.g., 5 1)
  - read pairs of inputs and apply them

```
vectors_generator: PROCESS
        VARIABLE in_line : LINE;
        VARIABLE x1, x2 : INTEGER;
BEGIN
        reset <= '0';
        WHILE NOT ( endfile(data_in)) LOOP
                READLINE (data_in, in_line);
                READ (in_line, x1); READ (in_line, x2);
                xi_rtl <= conv_unsigned(x1, SIZE));
                xi_beh <= conv_unsigned(x1, SIZE));
                yi_rtl <= conv_unsigned(x2, SIZE));
                yi_beh <= conv_unsigned(x2, SIZE));
                ...
```