Outline

1 General assumptions

2 IEEE Standard logic package (Std_logic_1164)

3 Synthesis
   - Combinational logic
   - Sequential logic
   - Finite State Machines (FSMs)

General assumptions

- only subset of VHDL is synthesizable,
- selected data types are supported,
- RTL and HLS synthesis (we concentrate mostly on RTL synthesis),
- description style has usually quite a big impact on the synthesized hardware,
- differences between different vendors.
IEEE Standard logic package

- VHDL defines only basic features to model digital devices, such as, simple BIT type, two types of a delay, etc.
- more complex data types for signals must be defined for realistic simulation.

An Example:

```vhdl
entity nand_2 is
    port (I1, I2: in bit; O: out bit);
end nand_2;

architecture standard of nand_2 is
begin
    O <= NOT (I1 AND I2) after 2.5 ns;
end standard;
```

Kris Kuchcinski (LTH) Synthesis from VHDL March 23, 2006 4 / 37

---

IEEE Standard logic package

Why we need more complex logic definition than BIT:

- we want to get more accurate simulation results,
- we use different technology and we want to simulate the same design in different technologies,
- real world is more complex than '0' and '1',
- more complex delays than standard VHDL models
  - inputs, output delays,
  - rising edge, falling edge delays,
  - min, max and typical delays,

Kris Kuchcinski (LTH) Synthesis from VHDL March 23, 2006 5 / 37

---

Std_logic_1164

Requirements:

- support for more realistic signal modeling,
- technology independence, allowing extensions for the future and providing support for all current technologies,
- provide consistent features which facilitate logic level modeling,
- hide the complexity of the package from the designer as much as possible, make models readable,
- provide for the timing accuracy continuum, giving the modeler flexibility to choose the appropriate level of model accuracy.

Intended use of the packages:

```vhdl
-- access standard logic facilities
use ieee.Std_logic_1164.all;
-- VHDL entity declaration
```

Kris Kuchcinski (LTH) Synthesis from VHDL March 23, 2006 6 / 37
Requirements:

- modeling signals with '0' and '1' simplifies the real circuit, because it only consider signal voltage without signals current,
- many simulators introduce, so called, signal strength related to the signal’s current,
- in std_logic_1164 we will introduce the following strengths of signals:
  - unknown,
  - forced (connected directly to ground or Vcc),
  - weak (connected to ground or Vcc through high resistive connection,
  - high impedance,
  - don’t care.

library IEEE;
PACKAGE Std_logic_1164 is
  TYPE std_ulogic is ( 'U', -- Uninitialized 'X', -- Forcing Unknown '0', -- Forcing 0 '1', -- Forcing 1 'Z', -- High Impedance 'W', -- Weak Unknown 'L', -- Weak 0 'H', -- Weak 1 '­', -- don’t care );

-- Unconstrained array of std_ulogic for use with the resolution function
TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;

FUNCTION resolved ( s : std_logic_vector ) RETURN std_logic;
-- *** Industry Standard Logic Type ***

SUBTYPE std_logic IS resolved std_ulogic;

-- Unconstrained array of std_logic for use in
-- declaring signal arrays

TYPE std_logic_vector IS ARRAY
    ( NATURAL RANGE <> ) of std_logic;

-- Basic states + Test

SUBTYPE X01 IS resolved std_ulogic range 'X' to '1';
    -- ('X','0','1')

SUBTYPE X01Z IS resolved std_ulogic range 'X' to 'Z';
    -- ('X','0','1','Z')

SUBTYPE UX01 IS resolved std_ulogic range 'U' to '1';
    -- ('U','X','0','1')

SUBTYPE UX01Z IS resolved std_logic range 'U' to 'Z';
    -- ('U','X','0','1','Z')

-- Overloaded Logical Operators

FUNCTION “and” ( l : std_ulogic; r : std_ulogic ) RETURN UX01;

FUNCTION “not” ( l : std_logic ) RETURN UX01;

Use of Std_logic_1164

Example:

entity nand_2 is
    port (I1, I2: in std_logic; O: out std_logic);
    -- interface description for
    -- two input nand gate using std_logic_1164
end nand_2;

architecture standard of nand_2 is
begin
    O <= NOT (I1 AND I2) after 2.5 ns;
end standard;
Synthesis

- different synthesis levels (behavioral, RTL, logic).
- different tools have different assumptions on specification styles,
- distinction between combinational logic and sequential logic.

Synthesis

Supported VHDL Language Constructs:
- Entity, Architecture and Package design units.
- Function and Procedure sub-programs
- IEEE Libraries - Std_Logic_1164, Std_LogicUnsigned, Std_Logic_Signed, Numeric_Std and Numeric_Bit
- Ports of mode in, out, inout and buffer
- Signals, Constants and Variables (the latter should be restricted to sub-programs and processes)
- Composite types - Arrays and Records
- Integer and subtypes Natural and Positive (Integer types should have a range constraint unless a 32-bit word is required)

Synthesis (cont'd)

Supported VHDL Language Constructs:
- User defined enumeration types (eg. type State_type is (s0, s1, s2, s3);).
- Operators - +, -, *, /, **, mod, rem, abs, not, =, /=, <, >, <=, >=, and, or, nand, nor, xor, xnor, sll, srl, sla, sra, rol, ror, & .
  (Notes: /, mod and rem are usually supported for compile-time constants or when the right-hand argument is a power of 2. The shifting operators are usually supported for compile-time constant shift values)
- Sequential statements - signal and variable assignments, wait, if, case, loop, for, while, return, null, function and procedure call.
  (Note: only a single wait statement is allowed in a process)
Supported VHDL Language Constructs:
- Concurrent statements - signal assignment, **process**, **block**, component instantiation, sub-program call, **generate**.
- **Generic** ports in entities.
- Predefined attributes - 'range', 'event
- Aggregates and others clause.

Unsupported VHDL Language Constructs:
- **Access** and **File** types - the former are similar to C’s pointers and files have no direct correspondence to hardware.
- **Register** and **Bus** kind signals - very rarely used VHDL constructs.
- Guarded blocks - as above, rarely used.
- **Next** and **Exit** loop control statements - A synthesis tool creates logic from a loop by 'unwinding' the loop into a series of iterations, often resulting in iterative circuits. Prematurely terminating a loop prevents this unwinding process.

Unsupported VHDL Language Constructs:
- Objects of type Real - floating point numbers cannot be mapped to hardware and therefore are not supported.
- User defined resolution functions - prior to IEEE Standard 1164, designers made up their own multi-valued logic systems and resolution functions to support technology related aspects of simulation. None of these custom solutions is standard, and therefore none are supported by synthesis.
Ignored Constructs:

- **Assert** and **report** statements - these are included in a design to send messages to the ‘console’ window relaying information about what is happening during a simulation. As such, they have nothing to do with the hardware of the design.

- **After** clause - this is used to specify inertial and transport hardware delays in a design, or alternatively in a test-bench to produce clocks and other control waveforms. Synthesis tools have no way of creating a specific delay time, unless it is created by means of counting clock pulses in hardware. Delays may be included in the pre-synthesis RTL design. However, they will be ignored during the synthesis process.

```vhdl
combi_process: process(A, B)
begin
  C <= not(A and B) after 20 ns;
  D <= not B after 20 ns;
end process combi_process;

Difference with

combi_process: process(A)
begin
  C <= not(A and B) after 20 ns;
  D <= not B after 20 ns;
end process combi_process;
```

**Sequential process**

```vhdl
A_process: process
begin
  wait until clk='0'
  C <= not(A and B);
  D <= not B after 10 ns;
end process A_process;

B_process: process
begin
  wait until clk='0'
  E <= not(D and G);
  F <= not G;
end process B_process;
```
Sequential process

Clocked processes lead to all signals, assigned inside the process, in a flip-flop.

```vhdl
example: process
begin
  wait until clk='1'
  dout <= din;
end process example;
```

If statement

```vhdl
if sel = '1'
  c <= b
else
  c <= a;
end if;
```

Asynchronous reset

```vhdl
process(clk, reset)
begin
  if reset = '1' then
    data <= "00";
  elseif clk'event and clk = '1' then
    data <= in_data;
  end if;
end process;
```
Synchronous reset

```vhdl
process(clk)
begin
    if clk'event and clk='1' then
        if reset = '1' then
            data <= "00";
        else
            data <= in_data;
        end if;
    end if;
end process;
```

Finite State Machines (FSMs)

- exist two types of FSMS, Moore and Mealy,
- contain register(s) for keeping FSM state,
- have logic for computing next state and output,
- can be modeled with combinational and sequential processes.

Moore machine

```vhdl
entity demo is
    port(clk, in1, reset: in std_logic;
        out1: out std_logic_vector(3 downto 0);
    end demo;

architecture moore of demo is
    type state_type is (s0, s1, s2, s3);
    signal state: state_type;
begin
    state_process: process(clk, reset)
        begin
            output_process: process(state)
                begin
            end process;
        end process;
end moore;
```
Moore machine (cont’d)

state_process: process(clk, reset) begin
  if reset = '1' then
    state <= s0;
  elseif clk'event and clk = '1' then
    case state is
    when s0 => if in1 = '1' then
              state <= s1; end if;
    when s1 => if in1 = '0' then
              state <= s2; end if;
    when s2 => if in1 = '1' then
              state <= s3; end if;
    when s3 => if in1 = '0' then
              state <= s0; end if;
    end case;
  end if;
end process;

Moore machine (cont’d)

output_process: process(state) begin
  case state is
    when s0 => out1 <= '0000';
    when s1 => out1 <= '1001';
    when s2 => out1 <= '1100';
    when s3 => out1 <= '1111';
  end case;
end process;

Mealy machine

```
rest/0000

S0
  1/1001

S1

0/0000
S2
  0/1100

S3
  1/1111
rest/1111

rest/1001
```
Mealy machine (cont’d)

- state process – the same as for Moore machine
- output process

```vhdl
output_process: process(state)
begin
  case state is
  when s0 =>
    if in1 = '1' then
      out1 <= '1001';
    else
      out1 <= '0000';
    end if;
  end process;
```

Incompletely defined combinational processes

```vhdl
architecture bad is
begin
  process(state)
  begin
    if a > b then
      q <= '0';
    elseif a < b then
      q <= '1';
    end if;
  end process
end;
```

```vhdl
architecture good is
begin
  process(state)
  begin
    if a > b then
      q <= '0';
    else
      q <= '1';
    end if;
  end process
end;
```

Bad synthesis

![Diagram](image)

Figure 14.2: Synthesis result of an incomplete process.
Good synthesis

High-level synthesis

High-level synthesis (behavioral, algorithm, architectural synthesis) means going from an algorithm specification to an RT level which implements the behavior.

- variables are allocated into registers or memory elements (sharing of registers is possible) - resource allocation,
- operators are allocated to functional units (several operators can be implemented by one functional unit, for example ALU) - resource allocation,
- operations are assigned to time slots for their execution in a synchronous implementation - scheduling,
- additional register are added if operations take more than one clock cycle.

High-level synthesis – example

Synthesis of the following code (inner loop of differential equation integrator)

```
while c do
    begin
        x1 := x + dx;
        u1 := u - (3*x+u+dx);
        y1 := y + u*dx;
        c := x < a;
        x := x1;  y := y1;  u := u1;
    end;
```
High-level synthesis – example

data-flow graph

scheduled data-flow graph

register allocation

Kris Kuchcinski (LTH)  Synthesis from VHDL  March 23, 2006  37 / 37