Introduction to Structured VLSI Design

- VHDL I

Joachim Rodrigues
Two HDLs used today

– VHDL and Verilog
– Syntax and ``appearance'' of the two languages are very different
– Capabilities and scopes are quite similar
– Both are industrial standards and are supported by most software tools
VHDL

Very High Speed Integrated Circuit (VHSIC) Hardware Description Language

A Technology Independent, Standard Hardware description Language (HDL), used for digital system modeling, simulation, and synthesis
Why VHDL?

There are several hardware description languages available; VHDL (Europe), Verilog (USA), and System C are the most common.

Advantages of VHDL
• IEEE standard.
• Supported by all CAD Tools.
• Technology independent.
• Common – Specially in Europe.
• Flexible – Delay modeling, Matrices, etc.
• Supports easy modeling of various abstraction levels.
VHDL History

- 1981 – VHSIC Initiated (US DoD)
- 1985 – VHDL version 7.2 (IBM and TI)
- 2008 - Accellera approved VHDL 4.0 also informally known as VHDL 2008
VHDL was developed as a language for modeling and simulation.

Consequence: Mismatch between simulation and synthesis -- Most constructs in VHDL are fine for simulation, but cannot be synthesized, e.g., *after*, *time*, etc.

With restrictions, VHDL can be used for synthesis.
Design of Digital Systems

• VHDL is used to design digital systems
  – Simulation
  – Synthesis

• Goal
  – Reliable design process, with minimum cost and time
  – Minimal design errors
  – Synthesizable code
Basic Design Methodology

1. Requirements
2. RTL Model
   - Simulate
3. Synthesize
4. Gate-level Model
   - Simulate
5. Test Bench
A digital design in VHDL requires

- Definition of required packages (libraries), e.g., `std_logic_1164`.
- An `ENTITY` (corresponds to the interface of a component).
- An `ARCHITECTURE` (describes its behavior).

An entity may have several architectures

Optional:
A `CONFIGURATION` that connects an `ARCHITECTURE` to an `ENTITY`.

Joachim Rodrigues, EIT, LTH, Introduction to Structured VLSI Design    jrs@eit.lth.se    VHDL I
The ENTITY is the interface of a component. It contains all IO-ports (port map) and possibly *generics*. 

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity Adder is
  generic (N: integer);
  port(
    A : in std_logic_vector (N-1 downto 0);
    B : in std_logic_vector (N-1 downto 0);
    S : out std_logic_vector (N-1 downto 0)
  );
end Adder;
```

Adder
Packages - Datatypes

Recommended types: integer, std_logic, and std_logic_vector.
  
- **Integer** – to model generics or constants
- **std_logic** – for one bit signals
- **std_logic_vector** – A bus of std_logic, e.g., counters, addresses, etc.

A std_logic may have **ONE** out of nine values, of which five are important here.

- ’U’ uninitialized - when the simulator starts
- ’X’ forced unknown – two signals driving the same output, short circuit
- ’0’ forced logic zero
- ’1’ forced logic one
- ’Z’ high impedance
**Entity- Port Declaration**

**Examples of declarations**

- **1-bit input port**
  - `clk: in std_logic;`

- **8-bit input port, MSB left (commonly used!!)**
  - `a: in std_logic_vector (7 downto 0);`

- **8-bit output port**
  - `S: out std_logic_vector (7 downto 0);`

Possible values of `std_logic` are: 'U', '0', '1', '-', 'Z', 'X`

“clk” port is special for clock pins and no “in” or “out” for signal names
Entity - Generics

Used to pass certain properties into a design to make it more general. Typically:

- Bus widths
- Delays

The value can be set in the entity declaration (default value), component declaration, or component instantiation.
Architecture

• An architecture is:
  – a pattern, a template, a way of doing it

• Developing a good architecture involves:
  – Coordination and optimization across many levels of abstraction.
  – ...under a large set of constraints and requirements (that is changing over time).
  – An iterative process involving design and analysis. “Exploring the design space”.

Joachim Rodrigues, EIT, LTH, Introduction to Structured VLSI Design jrs@eit.lth.se VHDL I
Basically two types of architectures:

- Behavioral: *using sequential processes*
- Structural: top level, component instantiation, *concurrent processes*
Architecture - behavioral

```
architecture behavioral of ADDER is
begin
    add_a_b : process (A,B)
    begin
        s <= A+B;
    end process add_a_b;
end architecture behavioral;
```

Architecture defines behavior of the circuit
• **Behavioral** architecture
  – Describes the algorithm performed by the module, FSM
  – May contain
    • *Process statements*
    • *Sequential statements*
    • *Signal assignment statements*
    • *Wait statements (not synthesizable)*
**Structural** architecture

– Implements a module as a composition of components (modules)

– contains
  
  • *signal declarations*, for internal interconnections
    – the entity ports are also treated as signals
  
  • *component instances*
    – instances of previously declared entity/architecture pairs
  
  • *port maps* in component instances
    – connect signals to component ports
Structural description

• In structural view, a circuit is constructed by smaller parts.
• Structural description specifies the types of parts and connections.
• Essentially a textual description of a schematic
• Done by using “component” in VHDL
  – First declared (make known)
  – Then instantiated (used)
Example - Structural description

architecture str_arch of even_detector is
component xor2
port(
  i1, i2: in std_logic;
o1: out std_logic);
end component;
component not1
port(
  i1: in std_logic;
o1: out std_logic);
end component;
signal sig1, sig2: std_logic;

begin
  unit1: xor2
    port map (i1 => a(0), i2 => a(1), o1 => sig1);
  unit2: xor2
    port map (i1 => a(2), i2 => sig1, o1 => sig2);
end begin;
Mixing Behavioral and Structural

An architecture may contain both behavioral and structural parts

– process statements and component instances
  • collectively called *concurrent statements*
– processes can read and assign to signals

Ex: Register-Transfer-Logic (RTL) model

– data path described structurally
– control section described behaviorally
library IEEE;
use IEEE.std_logic_1164.all;

entity TWOREG is
    generic( N : integer )
    port( DD : in std_logic_vector(N-1 downto 0);
    QQ : out std_logic_vector(N-1 downto 0);
    CLK : in std_logic );
end TWOREG;

architecture ARCH of TWOREG is
    component FF
        generic (N:integer)
        port(
            D : in std_logic_vector(N-1 downto 0);
            Q : out std_logic_vector(N-1 downto 0);
            CLK : in std_logic
        )
    end component;
    signal SLADD : std_logic_vector(N-1 downto 0);
begin
    FF1: FF
        generic map(N=>N)
        port map(D=>DD, Q=>SLADD, CLK=>CLK);
    FF2: FF
        generic map(N=>N)
        port map(D=>SLADD, Q=>QQ, CLK=>CLK);
end ARCH;
Combinational and Sequential Parts

In practice, processes come in two kinds,
- one with \texttt{CLK only} in the sensitivity list and
- one without \texttt{CLK} in the sensitivity list.

This state machine is modeled using two processes:
• **Concurrent statements (simple processes):**
  - `a <= b;`
  - `c <= a + b;`
  - `d <= a And B;`

• **Process statements:**

  namelabel: process (a, b, ... sensitivity list)
  variable declarations...
  begin
  sequential statements...
  – if ... then ... [else | elsif ...] end if;
  – for n in 0 to 7 loop...
  – case b is ...
  – s := z sll shamt;
  – i := a + b; --variable assignment, only in processes
  – c <= i; --concurrent signal assignment!
  end process namelabel;

• All processes are “executed” in parallel (think of gates and wires, not variables)
Writing combinational components:

```vhdl
architecture BEHAV of MUX2 is
begin
  process(IN0, IN1, SEL);
  begin
    Q <= IN0;
    if (SEL = '0') then
      Q <= IN0;
    elsif (SEL = '1') then
      Q <= IN1;
    end if;
  end process;
end BEHAV;
```

For a component to be combinational:

1. All inputs MUST be present in the sensitivity list!
2. All outputs MUST be assigned on every run!

If conditions 1 and 2 above are not fulfilled, THE SYNTHESIZED DESIGN WILL NOT WORK!
Writing **combinational** components:

```vhdl
architecture BEHAV of MUX2 is
begin
process(IN0, IN1, SEL)
begin
    Q <= IN0;
    if (SEL = '1') then
        Q <= IN1;
    end if;
end process;
end BEHAV;
```

For a component to be combinational:

1. All inputs MUST be present in the sensitivity list!
2. All outputs MUST be assigned on every run!

If conditions 1 and 2 above are not fulfilled, THE SYNTHESIZED DESIGN WILL NOT WORK!
Process – Example II

Writing **sequential** (clocked) components:

```vhdl
architecture GOOSE of FLIPFLOP is
begin
  process (CLK)
  begin
    if (CLK = '1') and (CLK'event) then
      Q<=D;
    end if;
  end process;
end GOOSE;
```

Enable register with synchronous reset

```vhdl
process (clk, reset)
begin
  if clk'event and clk='1'
  then
    if (Reset = '0') then
      Q <= '0';
    elseif enable='1' then
      Q <= D;
    end if;
  end if;
end process;
```

The **SENSITIVITY LIST**

When a signal present in the sensitivity list changes, the process is run once, top to bottom.
Case command

- **Example: Multiplexer**

```vhdl
architecture behv1 of Mux is
begin
process(I3,I2,I1,I0,S) --nested in process
begin -- use case statement
  case S is
    when "00" => Op <= I0; --sequential statements
    when "01" => Op <= I1;
    when "10" => Op <= I2;
    when "11" => Op <= I3;
    when others => Op <= "ZZZ"; --avoid inferred latches
  end case;
end process;
end behv1;
```
IF vs. CASE statements

If and case statements generate different HW

Case statement
Case c is
  when "01" => q <= a;
  when "10" => q <= b;
  when others => q <= c;
End case;

If statement
If (c1= '1') then
  q <= a;
Elseif (c2 = '1') then
  q <= b;
Else
  q <= c;
End if;
Finite State Machines

Why FSMs?
- Models different behaviour at different times (states)

A state machine requires:
- An initial state (Reset)
- Transitions with stable states
- Default values (Case statement)

Realizes:
- Datapath
- Controller
- Datapath+Controller
Basic State Machine

A Typical state machine

Output of a Mealy machine is state and input dependent
Transforming a State Machine into HW

Typical FSM

Generic Architecture for FSMs
Realization of FSFs

Entity declaration

```vhdl
library IEEE;use IEEE.STD_LOGIC_1164.all;
entity state_machine is
    generic (m : integer := 2) -- bus width
    port (clk : in STD_LOGIC;
          reset : in STD_LOGIC;
          input : in STD_LOGIC_VECTOR(m-1 downto 0);
          output : out STD_LOGIC_VECTOR(m-1 downto 0)
    );
end state_machine;
```
Architecture declaration (combinatorial part)

```vhdl
architecture implementation of state_machine is
  type state_type is (st0, st1, st2, st3); -- defines states;
  signal state, next_state : state_type;
  signal output, next_output STD_LOGIC_VECTOR (m-1 downto 0);
begin
combinatorial : process (input, state, next_state)
begin
  case (state) is -- Current state and input dependent
    when st0 => if (input = '01') then
      next_state <= st1;
      next_output <= "01"
    end if;
when ....
when others =>
  next_state <= next_state; -- Default
  next_output <= "00";
end case;
end process;
```

Realization of FSMs- cont’d
Sequential part:

```vhdl
synchronous : process (clk, reset)
begin
    if clk'event and clk = '1' then
        if reset = '1' then
            state <= st0;
            output <= "00";
        else
            state <= next_state;
            output <= next_output; -- registered outputs
        end if;
    end if;
end if;
end process;
end architecture;
```

---

Joachim Rodrigues, EIT, LTH, Introduction to Structured VLSI Design  
jrs@eit.lth.se  
VHDL I
• A FSM can be split in three parts:
  – State Transition Logic block
  – State Memory block (register)
  – Output logic
Summary

The knowledge you have gained today is sufficient to implement a simple combinational or structural architecture.
What’s next?

- Continue sequence detector
- Find a lab buddy
- 2\textsuperscript{nd} VHDL presentation Monday next week

First Deadline: Preparation of sequence detector Tuesday 7th