Introduction to Structured VLSI Design

- VHDL

Joachim Rodrigues

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 History**

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.

---

**Basic Design Methodology**

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.
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;

The ENTITY is the interface of a component. It contains all IO-ports (port map) and possibly generics.

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 - 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.

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
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”.

Architecture - behavioral

```vhdl
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

**Basiclly two types of architectures:**

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

Fully behavioral
Partially beh. & struct.
Pipelined structural

**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

**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

**Example: Structural Architecture**

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

entity TWOREG is
  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 TWOREG;

architecture ARCH of TWOREG is
begin
  process (CLK)
  begin
    case S is
      when '0' => D <= D + 1;
      when '1' => Q <= not Q;
    end case;
  end process;
end ARCH;
```

**Combinational and Sequential Parts**

In practice, processes comes in two kinds,
- one with **CLK only** in the sensitivity list and
- one without **CLK** in the sensitivity list.

This state machine is modeled using two processes:
Concurrent Statements and Processes

- Concurrent statements (simple processes):
  - \( a \leftarrow b \)
  - \( c \leftarrow a + b \)
  - \( d \leftarrow a \text{ And } B \)

- 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)

Process – Example I

Writing combinational components:

architecture BEHAV of MUX2 is
begin
process(INO, IN1, SEL);
begin
Q <= IN0;
if (SEL = '0') then
Q <= IN0;
elseif (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

Enable register with synchronous reset

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

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 -- 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

```vhdl
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

Output of a Mealy machine is state and input dependent

A Typical state machine
Transforming a State Machine into HW

Realization of FSMs

Entity declaration

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)

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;
end architecture;

Realization of FSMs- cont’d

Sequential part:

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 process;
end architecture;
FSM Structure

- A FSM can be split in three parts:
  - State Transition Logic block
  - State Memory block (register)
  - Output logic

Summary

- Signals have a delta-delay if not other delay is specified
- Variables are updated instantaneously
- Statements in
  - an architecture body are concurrent
  - a process are sequential
- Components need to be declared before instantiation
- FSMs are implemented using CASE statements

Inferred Latches

- In case a process does not assign an output signal value:
  - The old value is retained
  - This is an inferred latch and the circuit is no longer combinational
  - The latch is not explicit but is inferred from the VHDL code
  - Normally caused by failure to consider all combinations
  - Bad programming practice

**Bad1**: process(sA,sB,a,b)
begin
if sA='1' then
  z<=a;
elsif sB='1' then
  z<=b;
end if;
end process Bad1;

**Bad2**: process(sA,a,b)
begin
if sA='1' then
  z<=a;
else
  z<=b;
end if;
end process Bad2;

**Good**: process(select,a,b)
begin
if select='1' then
  z<=a;
else
  z<=b;
end if;
end process Mux;

Simulation / Synthesis mismatch?

- For all combinational processes (no clock):
  - All process input signals must be in the sensitivity list.
  - Synthesis tools usually ignore the sensitivity list, but simulators don’t!
  - Wrong sensitivity list will cause mismatch in behavior between functional simulation and the synthesized circuit.

**Wrong**: process (select,b)
begin
if select='1' then
  z<=a;
else
  z<=b;
end if;
end process Wrong;

**Mux**: process (select,a,b)
begin
if select='1' then
  z<=a;
else
  z<=b;
end if;
end process Mux;

Both versions synthesize to the circuit to the right.
Summary

The knowledge you have gained today is sufficient to implement a simple combinational or structural architecture.

Testbench and Simulation

- Testing: Testbench and Circuit
  - The testbench models the environment our circuit is situated in.
  - Provides stimuli (input to circuit) during simulation.
  - May verify the output of our circuit against test vectors.
- The testbench (VHDL) consists of:
  - A top level entity connecting the circuit to the testbench
  - One or more behavioral architectures (matching the refined level of our circuit).
- Testing is done at every abstraction level.

Testbench and Simulation

• Test strategy -> documentation of design correctness.
• Good design = easy to test -> a good design is a well-structured well-partitioned hierarchical design.
  – Tip: Do not overdo it with too many entities!

• Design and test of pipelined circuits
  • With a good testbench it is also easy to verify that the final synthesized design works as specified.
Testbench Example

**Entity and Architecture**

```vhdl
entity testbench is
end testbench;
architecture test of testbench is
end test
```

**Component Declaration**

```vhdl
component circuit is
port(clk, reset, inputs, outputs);
end circuit;
signal inputs, outputs, clk, reset : type;
```

**Clock Generator**

```vhdl
begin
clock_gen: process
begin
if clk='1' then clk<='0'
else clk<='1'; end if;
wait for clk_period/2;
end process
```

**Reset Signal Generator**

```vhdl
reset_gen: process
begin
reset <= '1', '0' after 57 ns;
delay(clk_period, 57 ns);
end process
```

**Component Instantiation**

```vhdl
circuit: circuit
port map (clk, reset, inputs, outputs);
instantiation process
```

**Testbench**

```vhdl
tester: process(clk, reset)
begin
...
end process;
end testbench;
```

Joachim Rodrigues, EIT, LTH, Introduction to Structured VLSI Design  jrs@eit.lth.se  VHDL I

---

Testbench and Simulation Circuit Verification

- **Input driver and output verification:**
  - Test vectors

  - **Testvectors:**
    - Input stimuli
    - Correct circuit output

  - **Test vector file**
    - Circuit
    - Actual circuit output
    - Equal?

Joachim Rodrigues, EIT, LTH, Introduction to Structured VLSI Design  jrs@eit.lth.se  VHDL I

---

Simulation and Verification

**Functional simulation**

- Your initial specification:
  - While designing a digital system you will rely on functional simulation to verify that your VHDL specification works.
- Cycle-accurate RTL:
  - After refining your VHDL specification to the synthesizable RTL abstraction level, you need to re-verify, using the same testbench.
- After synthesis:
  - This testbench is again used to simulate the post-synthesis VHDL netlist (output of the synthesis tool), to verify that it actually works like your VHDL specification.

Joachim Rodrigues, EIT, LTH, Introduction to Structured VLSI Design  jrs@eit.lth.se  VHDL I

---

Simulation and Timing-Verification

- After synthesis and place & route:
  - Just after synthesis, the simulation is still only functional, we need to simulate the electrical timings of your circuit:
    - Clock-to-output delay, contamination delay, propagation delay, wire delay, setup time, hold time, clock skew, ...
    - The implementation tools (Synthesis + Place & Route) can generate a detailed timing model of your circuit which includes all the above parameters!
    - The tools can also calculate maximum operating frequency
  - Timing simulations with added test vectors are required to find bugs in the VHDL code, which were not caught by functional simulation or by the synthesis tool.
    - Such bugs usually reflect bad design practices such as: improper use of latches, unintended glitches, metastable flip-flops, reset/clock skew.

Joachim Rodrigues, EIT, LTH, Introduction to Structured VLSI Design  jrs@eit.lth.se  VHDL I
What’s next?

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

First Deadline: Preparation of sequence detector Monday 7th