Lecture 2: Introduction to System Design, VHDL Basics

TIE-50206 Logic Synthesis
Arto Perttula
Tampere University of Technology
Fall 2017
Contents

• 1. Introduction to System Design
  – Abstraction
  – Main phases

• 2. VHDL basics
  – Entity – the interface
  – Architecture – the behavior
  – Continues next week
Acknowledgements

• Prof. Pong P. Chu provided "official" slides for the book which is gratefully acknowledged
  – See also: http://academic.csuohio.edu/chu_p/

• Most slides were originally made by Ari Kulmala
  – and other previous lecturers (Teemu Pitkänen, Konsta Punkka, Mikko Alho, Erno Salminen…)

Arto Perttula 26.10.2017
1a. Representation (View) and Abstraction

1. INTRODUCTION TO SYSTEM DESIGN
Examples of Different Views

• View: different perspectives of a system

1. Behavioral view:
   – Describe functionalities and I/O behavior
   – Treat system as a black box

2. Structural view:
   – Describe the internal implementation
     (components and interconnections)
   – Essentially a block diagram (or schematic)

3. Physical view:
   – Add more info to structural view: component size, component locations, routing wires
   – E.g., layout of a print circuit board
Examples of Different Views (2)

1. Behavioral
2. Structural
3. Physical

higher abstraction

inputs:
button0_in, button1_in
...

outputs:
led0_out
audio_out
...

Function:
When user presses button1, then...
When...

Arto Perttula
26.10.2017
Complexity Management

• Q: How to manage complexity for a chip with 10 million transistors?
• A: Abstraction – a simplified model of a system
  – Show the selected features
  – Ignore many details
• E.g., timing of an inverter
Levels of Abstraction in HDL

1. Transistor level, lowest abstraction
2. Gate level
3. Register transfer level (RTL)
   - Typical level nowadays in addition to structural
4. Behavioral (Processor) level, highest abstraction
5. (Manager view: everything works just by snapping fingers…)

- Characteristics of each level
  - Basic building blocks
  - Signal representation
  - Time representation
  - Behavioral representation
  - Physical representation
Summary of Abstractions

<table>
<thead>
<tr>
<th>Level</th>
<th>Typical blocks</th>
<th>Signal representation</th>
<th>Time representation</th>
<th>Behavioral description</th>
<th>Physical description</th>
<th>Example block</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>transistor</td>
<td>transistor, resistor</td>
<td>voltage</td>
<td>continuous function</td>
<td>differential equation</td>
<td>transistor layout</td>
<td></td>
<td>ELT-xxxxxx</td>
</tr>
<tr>
<td>gate</td>
<td>and, or, xor, flip-flop</td>
<td>logic 0 or 1</td>
<td>propagation delay</td>
<td>Boolean equation</td>
<td>cell layout</td>
<td></td>
<td>DigiPer. Dig Suunn</td>
</tr>
<tr>
<td>RT</td>
<td>adder, mux, register</td>
<td>integer, system state</td>
<td>clock tick</td>
<td>extended FSM</td>
<td>RT level floor plan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>behavioral</td>
<td>processor, memory</td>
<td>abstract data type</td>
<td>event sequence</td>
<td>algorithm in C</td>
<td>IP level floor plan</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This course focuses on RTL
Behavioral Description

• An *untimed* algorithm description with no notation of time or registers (or even interface)
• The tools automatically place the registers according to the constraints set by the designer
• E.g., FFT described in Matlab/C
• The designer gives constraints to a behavioral synthesis tool
  – Maximum latency, clock frequency, throughput, area
  – Interface
• The tool explores the design space and creates the timing-aware circuit
• Not very well supported yet
Register-Transfer Level (RTL)

- Typically, HW description languages use RT level
- The idea is to represent the combinational logic before registers
  - The logic between registers, i.e., between register transfers
- The registers are ”implied” not explicitly defined in VHDL
  - Synchronous processes imply registers and are covered later lectures
- Combinatorial logic is created by synthesis tool and depends on
  1. Right-hand-side of the signal assignment (e.g. \( x_r \) <= \( a+b \));
  2. Preceding control structures (if \( \text{sel}='1' \), for\( (i=0;i<9;i++) \)…)

Note that you can create purely combinational logic using RTL abstraction.
Register-Transfer Level (RTL) (2)

- RT (Register Transfer) is a bit misleading term
- Two meanings:
  1. Loosely: represent the module level
  2. Formally: a design methodology in which the system operation is described by how the data is manipulated and moved among registers

<table>
<thead>
<tr>
<th>Time_instant</th>
<th>Value of ( a_r )</th>
<th>Value of ( b_r )</th>
<th>Value of ( c_r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_0 )</td>
<td>( x_0 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T_1 )</td>
<td>( x_1 )</td>
<td></td>
<td>( \text{foo}(x_0) )</td>
</tr>
<tr>
<td>( T_2 )</td>
<td>( x_2 )</td>
<td>( \text{foo}(x_1) )</td>
<td>( \text{bar}(\text{foo}(x_0)) )</td>
</tr>
<tr>
<td>... ( x_1, x_0 )</td>
<td>( \text{D} \ Q ) &amp; ( a_r )</td>
<td>( \text{D} \ Q ) &amp; ( b_r )</td>
<td>( \text{D} \ Q ) &amp; ( c_r )</td>
</tr>
</tbody>
</table>
Key for Success: Hierarchy

Hierarchical design: top level

- All systems are designed and implemented hierarchically
- The same component can be replicated and used in many products
- Usually only knowledge of external behavior is required, not the internals

Arto Perttula 26.10.2017
Structural VHDL Description

- Circuit is described in terms of its components
- High-level block diagram
- Black-box components, modularity
- For large circuits, low-level descriptions quickly become impractical
- **Hierarchy is very essential to manage complex designs**

Syntax:
Corresponds to the entity

**Parameters**

```vhdl
component component_name
  generic(
    generic_declaration;
    generic_declaration;
    . . .
  );
  port(
    port_declaraction;
    port_declaraction;
    . . .
  );
```

**Ports**
Example

• A hierarchical two-digit decimal counter
  – pulse=1 when q is 9
  – p100=1 when both q_ten and q_one are 9
  – However, if en goes 0 when q=9, something strange happens…
• Let’s concentrate on the structure…

Top-level block diagram, ”hundred_counter”
Example Implemented in VHDL

Top-level entity

library ieee;
use ieee.std_logic_1144.all;

entity hundred-counter is
port(
clk, reset : in std_logic;
en, q, one : out std_logic);
end entity hundred-counter;

architecture vhdl-87-arch of hundred-counter is
begin

one digit dec_counter
port map (clk=>clk, reset=>reset, en=en, 
ten_digit => p_ten, q=>q, one=>one, 
p00=>p00);

end hundred-counter;

end vhdl-87-arch;

w e will use an existing component called dec_counter

Iterate the counters as in schematic and connect the signals to ports.

Internal signals

module vhdl-87
end module vhdl-87

Internal signals

signal p00, p_ten, q, one : std_logic;
end component;

architecture vhdl-87-arch of hundred-counter is
begin

one digit dec_counter
port map (clk=>clk, reset=>reset, en=en, 
ten_digit => p_ten, q=>q, one=>one, 
p00=>p00);

end hundred-counter;

end vhdl-87-arch;

Internal signals

Top-level entity

library ieee;
use ieee.std_logic_1144.all;

entity hundred-counter is
port(
clk, reset : in std_logic;
en, q, one : out std_logic);
end entity hundred-counter;

architecture vhdl-87-arch of hundred-counter is
begin

one digit dec_counter
port map (clk=>clk, reset=>reset, en=en, 
ten_digit => p_ten, q=>q, one=>one, 
p00=>p00);

end hundred-counter;

end vhdl-87-arch;

Internal signals

signal p00, p_ten, q, one : std_logic;
end component;

architecture vhdl-87-arch of hundred-counter is
begin

one digit dec_counter
port map (clk=>clk, reset=>reset, en=en, 
ten_digit => p_ten, q=>q, one=>one, 
p00=>p00);

end hundred-counter;

end vhdl-87-arch;

Internal signals

signal p00, p_ten, q, one : std_logic;
end component;

architecture vhdl-87-arch of hundred-counter is
begin

one digit dec_counter
port map (clk=>clk, reset=>reset, en=en, 
ten_digit => p_ten, q=>q, one=>one, 
p00=>p00);

end hundred-counter;

end vhdl-87-arch;
1b. Development Tasks

1. INTRODUCTION TO SYSTEM DESIGN
System Development

• Developing a digital system is a refining and validating process
• Main tasks:

  I. requirements capture, specification
  II. design, synthesis
  III. physical design
  IV. fabrication, testing

(time)

I-III. verification
I. Specification

• Capture the
  1. use cases, requirements
  2. non-functional requirements (performance, cost, power consumption, silicon area)

• Usually informal, natural language (English, Finnish) completed with tables and illustrations
  – Formal methods are being studied and their importance will increase
II. Design, Synthesis

• A refinement process that realizes a description with components from the lower abstraction level
  – Manual/automated
• The resulting description is a structural view in the lower abstraction level
  – A synthesis from VHDL code obtains netlist (gates and flip-flops)
  – Estimates the size, maximum frequency and power consumption
• Type of synthesis:
  – High-level synthesis
  – RT level synthesis
  – Gate level synthesis
  – Technology mapping
• Emphasis of this course
III. Physical Design

• Placement of cells and routing of wires
  – Refinement from structural view to physical view
  – Derive layout of a netlist

• Circuit extraction:
  – Determine wire resistance and capacitance accurately to estimate timing and power

• Others
  – Derivation of power grid and clock distribution network, assurance of signal integrity etc.
I-III. Verification

• Check whether a design *meets the specification* and performance goals
• Concern the correctness of the initial design and the refinement processes
• Two aspects
  1. Functionality (e.g., is the answer 42?)
  2. Non-functional (e.g., performance)
• Takes ~40-80% of design time
I-III. Methods of Verification

1. Simulation
   – Spot check: cannot verify the absence of errors
   – Can be computationally intensive

2. Hardware emulation with reconfigurable HW
   – Almost real-time, connection to external devices

3. Timing analysis
   – Just check the worst case delay, automated

4. Formal verification
   – Apply formal mathematical techniques to determine certain properties, applicable only in small scale
   – E.g., equivalence checking between two models

5. Specification/code review
   – Explain the design/specification to others and they comment it
   – Surprisingly powerful!
IV. Testing

- Testing is the process of detecting physical defects of a die or a package occurred at the time of manufacturing
  - Testing and verification are different tasks in chip design
- Difficult for large circuit
  - Must add auxiliary testing circuit into design
  - E.g., built-in self test (BIST), scan chain etc.
  - Some tests with specialized test-SW running on chip
- Locating the fault is not always needed
  - Faulty chips simply discarded
- Basic tests are done at wafer-level
  - Sub-set of tests also for packaged chips
1c. Development Flow

1. INTRODUCTION TO SYSTEM DESIGN
EDA Software

• EDA (Electronic Design Automation) software can automate many tasks
• Mandatory for success together with re-use!
• Can software replace human hardware designer? (e.g., C-program to chip)
• Synthesis software
  – Should be treated as a tool to perform transformation and local optimization
  – Cannot alter the original architecture or convert a poor design into a good one
  – See also the so called ”Mead & Conway revolution”
• EDA tools abstraction level in functional description has not increased significantly since mid-90’s when RT-level gained popularity
  – Increased abstraction always causes some penalty in performance, area etc. when increasing abstraction, but significant improvement in time to design
Design Flow

- Medium design targeting FPGA
- Circuit up to 50,000 gates
- Note the test bench development at the same time as RTL (or before that)
- Large design targeting FPGA needs also
  - Design partition
  - More verification
  - I/O-verification, external interfaces
2a. Basics

2. VERY HIGH SPEED INTEGRATED CIRCUIT HARDWARE DESCRIPTION LANGUAGE (VHSIC HDL = VHDL)
Why (V)HDL?

- Interoperability
- Technology independence
- Design reuse
- Several levels of abstraction
- Readability
- Standard language
- Widely supported
  ➢ Improved productivity
What Is VHDL?

- **VHDL = VHSIC Hardware Description Language**
  - (VHSIC = Very High Speed IC)

  - Design specification language
  - Design entry language
  - Design simulation language
  - Design documentation language
  - An alternative to schematics
A Brief VHDL History

• Developed in the early 1980’s
  – For managing design problems that involved large circuits and multiple teams of engineers
  – Originally for documentation, synthesis developed soon after
  – Funded by U.S. Department of Defence
• First publicly available version released in 1985
• IEEE standard in 1987 (IEEE 1076-1987)
  – IEEE = Institute of Electrical and Electronics Engineers
• An improved version of the language was released in 1994
  – IEEE standard 1076-1993
  – No major differences to ’87, but some shortcuts added
VHDL

- Parallel programming language(!) for hardware
  - Allows sequential code portions also
- Modular
  - Interface specification is separated from the functional specification
- Allows many solutions to a problem
- The coding style matters!
  - Different solutions will be slower and/or larger than others
  - Save money!
- Case-insensitive language
  - Examples (usually) show reserved words in CAPITALS
- Widely used language
My First VHDL Example

ENTITY eg1 IS
  PORT (  
    clk   : IN STD_LOGIC;  
    rst_n : IN STD_LOGIC;  
    a, b  : IN STD_LOGIC_VECTOR(1 DOWNTO 0);  
    both_1_out: OUT STD_LOGIC;  
    siwa_out : OUT STD_LOGIC  
  );
END eg1;
ARCHITECTURE rtl OF eg1 IS
  SIGNAL c : STD_LOGIC;
BEGIN
  both_1_out <= c;
  c          <= a(0) AND b(0);
  PROCESS ( clk, rst_n )
  BEGIN
    IF rst_n = '0' THEN
      siwa_out <= '0';
    ELSIF clk'EVENT AND clk = '1' THEN
      IF a = '00' THEN
        siwa_out <= b(0);
      ELSIF a = '11' then
        siwa_out <= b(1);
      ELSE
        siwa_out <= '0';
      END IF;
    END IF;
  END PROCESS;
END rtl;
VHDL Environment

(Tools used in this course are shown in parentheses)
Entities – Interfaces

• A black box with interface definition
  – Functionality will be defined in architecture
• Defines the inputs/outputs of a component (pins)
• Defines the generic parameters (e.g., signal width)
• A way to represent modularity in VHDL
• Similar to symbol in schematic
• Reserved word ENTITY

ENTITY comparator IS
  PORT ( 
    a_in : IN STD_LOGIC_VECTOR(8-1 DOWNTO 0);
    b_in : IN STD_LOGIC VECTOR(8-1 DOWNTO 0);
    eq_out : OUT STD_LOGIC
  );
END comparator;

26.10.2017
Arto Perttula
Architecture – Internals

- Every entity has at least one architecture
- Architecture specifies the internals of a design unit and is coupled to a certain entity
  - Defines functionality
- One entity can have several architectures
- Architectures can describe design on many levels
  - Gate level
  - RTL (Register Transfer Level)
  - Structural
  - Behavioral level

```
entity X

A_in --> X
B_in --> X

C_out <- X

arch 1
  "It is logical AND"

arch 2
  "It is logical XOR"
```

etc.
Architecture (2)

- **Example:**
  ```vhdl
  ARCHITECTURE rtl OF comparator IS
  BEGIN
    eq_out <= '1' WHEN (a_in = b_in) ELSE '0';
  END rtl;
  ```

- **Two main approaches**
  1. Define new functionality with control statements, e.g., if-for-case, (rtl), shown above
  2. Instantiate existing components and define interconnections between them (structural)
Ports

• Provide communication channels (=pins) between the component and its environment
• Each port must have a name, direction and a type
  – An entity may omit port declaration, e.g., in test bench
• Port directions:
  1. **IN**: A value of a port can be read inside the component, but cannot be assigned. Multiple reads of port are allowed.
  2. **OUT**: Assignment can be made to a port, but data from a port cannot be read. Multiple assignments are allowed.
  3. **INOUT**: Bi-directional, assignments can be made and data can be read. Multiple assignments are allowed. (not recommended inside a chip)
  4. **BUFFER**: An out port with read capability. May have at most one assignment (not recommended)
Signals

- Used for communication inside the architecture, carry data
  - Ports behave like signals
- Can be interpreted as
  a) Wires (connecting logic gates)
  b) "wires with memory" (i.e., FF’s, latches etc.)
- VHDL allows many types of signals
  - Bit vectors, integers, even multidimensional arrays and records
- Declared in the architecture body’s declaration section
- Signal declaration:
  SIGNAL signal_name : data_type;
- Signal assignment:
  signal_name <= new_value;
Other Declarations

• Functions, procedures (subprograms)
  – Much like in conventional programming languages

• Component declaration
  – ”We will use an adder which looks like this”

• Configuration
  – ”We will use exactly this adder component instead of that other one”
  – Binds certain architecture to the component instance
Libraries And Packages

• Frequently used functions and types can be grouped in a package
• Libraries include several compiled packages and other design units
• Packages typically contain
  – Constants
    • Like header.h in conventional programming languages
  – General-purpose functions
    • E.g., Log2(x)
  – Design-specific definitions
    • E.g., own data types, records (structs)
Design Units

- Segments of VHDL code that can be compiled separately and stored in a library
- Library = directory of compiled VHDL files
Structure of VHDL Entity

- Usually 1 entity plus 1 architecture per file
  - File named according to entity

- Architectures contains usually either
  a) processes
  or
  b) instantiations

Arto Perttula
Relation Between Circuit and VHDL

Realization

Examples

GENERAL VHDL STRUCTURE

entity entity_name is
  port(
    A : IN;
    B : IN;
    Sel : IN;
    C : OUT;
  );
end entity_name;

architecture arch_name of entity_name is
begin
  declarations;
  begin_concurrent_statement;
  begin_concurrent_statement;
  begin_concurrent_statement;
  ... 
end arch_name;

signal add, mul
Components +, *
declaration

Mux realization
Component instantiation
Signal assignments

Adder
Multiplier

A
B
Sel
C
add
mul
0
1

sel

A
B
C
Even Parity Detection Circuit

- Input: $a(2)$, $a(1)$, $a(0)$
- Output: even

Boolean function:

$$even = a(2)' \cdot a(1)' \cdot a(0)' + a(2)' \cdot a(1) \cdot a(0) + a(2) \cdot a(1)' \cdot a(0) + a(2) \cdot a(1) \cdot a(0)'$$

Truth table:

<table>
<thead>
<tr>
<th>$a(2)$</th>
<th>$a(1)$</th>
<th>$a(0)$</th>
<th>even</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Even Parity Detection Circuit at Gate-Level VHDL

Defines packages that are used in the design.

The interface of a block "even_detector"
Input a, 3 bits
Output even, 1 bit

Signals used internally
Functionality of the block (gate level representation).
The order of assignments does NOT matter here (concurrent statements).

library ieee;
use ieee.std_logic_1164.all;

entity even_detector is
  port(
    a: in std_logic_vector(2 downto 0);
    even: out std_logic);
end even_detector;

architecture eg_arch of even_detector is
  signal p1, p2, p3, p4 : std_logic;
begin
  even <= (p1 or p2) or (p3 or p4);
  p1 <= (not a(0)) and (not a(1)) and (not a(2));
  p2 <= (not a(0)) and a(1) and a(2);
  p3 <= a(0) and (not a(1)) and a(2);
  p4 <= a(0) and a(1) and (not a(2));
end eg_arch;
<table>
<thead>
<tr>
<th>Language constructs in VHDL</th>
<th>Purpose</th>
<th>Other notes</th>
<th>C++ counterpart</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTITY</td>
<td>Defines interface. Includes generics and ports (their names, widths, and directions).</td>
<td>&quot;Public interface&quot;, the actual implementation is hidden into architecture.</td>
<td>Class definition</td>
</tr>
<tr>
<td>GENERIC</td>
<td>Instance-specific constant value</td>
<td>Excellent idea in HDL!</td>
<td>Constant parameters, templates</td>
</tr>
<tr>
<td>PORT</td>
<td>I/O pin of an entity. Defines direction and type.</td>
<td>See also signal.</td>
<td>Method of a class, inter-process message</td>
</tr>
<tr>
<td>ARCHITECTURE</td>
<td>Contains functionality.</td>
<td>One entity may have many architectures in the library</td>
<td>Class implementation</td>
</tr>
<tr>
<td>SIGNAL, (VARIABLE)</td>
<td>Communication channel between components/processes.</td>
<td>They are not the same! Variables only inside processes</td>
<td>Variable</td>
</tr>
<tr>
<td>COMPONENT</td>
<td>For instantiating a sub-block</td>
<td>Needed for hierarchy.</td>
<td>Class instance, object</td>
</tr>
<tr>
<td>PROCESS</td>
<td>These capture most of the functionality.</td>
<td>Processes are executed in parallel. Both seq. and comb.</td>
<td>Thread</td>
</tr>
<tr>
<td>IF, FOR, CASE, ASSIGNMENT</td>
<td>Control statements</td>
<td>Bounds must be known for loops at compile-time</td>
<td>The same</td>
</tr>
<tr>
<td>PACKAGE</td>
<td>Contains shared definitions.</td>
<td>Constants, functions, procedures, types</td>
<td>Header file (file.h)</td>
</tr>
<tr>
<td>LIBRARY</td>
<td>Holds analyzed ('compiled') codes</td>
<td>Standard ieee library is practically always used</td>
<td>Compiled object codes (file.o)</td>
</tr>
</tbody>
</table>
EXTRA SLIDES ON VHDL
Recap: Register Transfer Level (RTL)

- Circuit is described in terms of how data moves through the system.
- In the register-transfer level you describe how information flows between registers in the system.
- The combinational logic is described at a relatively high level, the placement and operation of registers is specified quite precisely.
- The behavior of the system over the clock prediods is defined by registers.