**What does HDL stand for?**

HDL is short for **Hardware Description Language** (VHSIC Hardware Description Language) (Very High Speed Integrated Circuit).

---

**Why use an HDL?**

**Question:**
How do we know that we have not made a mistake when we manually draw a schematic and connect components to implement a function?

**Answer:**
By describing the design in a high-level (easy to understand) language, we can simulate our design before we manufacture it. This allows us to catch design errors, i.e., that the design does not work as we thought it would.

- Simulation guarantees that the design behaves as it should.
**Why use an HDL?**

Corollary:

By letting a tool convert the high-level description into hardware (assuming that we can mathematically prove that the tool works correctly - which is impossible today), we can then guarantee that what we have simulated is actually what we get.

• Synthesis guarantees that the translation to hardware is done correctly

---

**What is a Hardware Description Language?**

HDL is short for Hardware Description Language (VHDL - Very High Speed Integrated Circuit)

---

**What do we mean with hardware?**

**The Cabinet**

Network of processors with system software.

---

**Printed Circuit Boards (PCB)**

Single/Parallel processor systems with dedicated software, for instance for controlling a robot’s arm.

---

**MultiChip Modules (MCM)**

Single processors with software, RAM and ROM memories for storage.

---

**Circuits - ASIC/FPGAs**

---
What do we mean with hardware?

- **Functional blocks**
- **Gates**

What can we do with an HDL?

HDL is short for **Hardware Description Language**

(VHDL - **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit)

**Managing Design Complexity**

- Field Programmable Gate Arrays (FPGAs)
- Application Specific Integrated Circuits (ASICs)
- System on Chip (SoC)
- Network on Chip (NoC)

**Future Chips**

1 Billion transistors on a single Chip!

- Design methods and CAD tools
- How do we design a circuit with 1 G transistors?
- How do we test the circuit working?

**This course - part 1**

- Learning VHDL coding
  - Focus will be on applications that fit in or are part of
    - Application Specific Integrated Circuits (ASICs)
    - Field Programmable Gate Arrays (FPGAs)

- From gate level up to Functional level
  - Specification (understanding the problem)
  - Implementation (coding in VHDL)
  - Validation (simulating in VHDL)
Part 2 - Digital Design Methodology

- Learning how to design, i.e., we will focus on
  - methods how to divide and conquer a problem to implement virtually anything (including an electronic brain)
  - methods how to estimate the size, speed and power consumption of any given design

Digital Design Constraints

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Area</th>
<th>Time: Clock Period</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>-</td>
<td>*</td>
</tr>
</tbody>
</table>

Algorithm

```
WHILE G < K LOOP
  F := E*(A+B);
  G := (A+B)*(C+D);
END LOOP;
```

Some Administrative Details

- Course Responsible
  Johnny Öberg johnny@imit.kth.se
- Laboration Assistants
  mdd_asse@imit.kth.se
  Erland Nilsson, Rikard Thid, Mikael Millberg, Zhonghai Lu
- System Administrators
  Owe Thessén (Campus), Julio Mercado (Kista)

Visiting Address

- We are working in the
  Systems, Architectures and Methodologies Group (SAM), Laboratory of Electronic and Computer Systems (LECS), Dept. of Micro Electronics and Information Technology (IMIT)
- Visiting Address:
  KTH Forum, Isafjordsgatan 39, Kista (Elevator C, 8th Floor)
- Postal Address:
  KTH/IMIT/LECS, Electrum 229, SE-164 40 Kista

Course Overview

- The course is divided in two parts
  - 1) VHDL (~1st Quarter)
  - 2) Digital Design Methods (~2nd Quarter)
- Examination
  - 1) We use continuous examination. A lab passed gives points to the exam. All labs passed gives 10 points. 10 points is the minimum to pass the exam with a grade 3. Max on exam is 20 points. Grade 4 - 20 points. Grade 5 - 25 points.
  - 2) The written exam will be on the Digital Design Methods part of the course, i.e., there will be no VHDL on the exam. VHDL programming is done in labs only.
Course Overview

- There are 6 obligatory labs (Exam points in parenthesis):
  0) Introduction to VHDL and Modelsim (voluntary - 0p)
  1) Combinational circuits (1p)
  2) Finite State Machines (1p)
  3) VGA Controller (1p)
  4) Resolution Functions & Databusses (2p)
  5) The Micro controller (mini processor) (2p)
  6) Mini-project - the FIR-filter (three normal labs - 3p)

Course Overview

- Labs (4h scheduled every week - 14 in total)
  - Labs 0-3 take ~4-8 hours each to complete, depending on the student’s programming skills.
  - Labs 4-5 takes 8-12 hours to complete.
  - The mini-project labs take ~12-24 hours to complete

- Students must prepare labs at home to manage!

NOTE!

- Lab hours are for examination and to ask questions from assistants if you get stuck. Not for coding. The students should think about the problem(s) first and prepare code and questions before coming to the lab. Otherwise, the labs will take a very long time to complete.

- You don’t need to finish a lab within the scheduled time. If you do not finish one lab in time, continue where you were the next time you come. You may of course also do the labs on your own at other hours (unless occupied by others) or do the labs at home.

- During the scheduled lab hours, there will be assistants around to answer/ask questions and to check your progress.

VHDL/Modeling Lectures

1 - VHDL Basics
2 - Modeling Styles
3 - Data Types & Conversion Functions
4 - Latches, Flip-flops and Memories
5 - FSM Modeling
6 - Resolved Functions & Databases
7 - FSMDs: FSMs with a Datapath
8 - Advanced Features & Dataflow Modeling

Digital Design Methods’ Lectures

- 9 - Asynchronous State Machines
- 10 - Implementing Algorithms I
- 11 - Implementing Algorithms II
- 12 - Implementing Algorithms III
- 13 - Implementing Algorithms IV (Bit/Digit Serial Design)

Note!

- The course is given twice in parallel on Campus for E- and D-students (year 4) and in Kista for IT, and SoC-students.

  - If you miss a lecture, you can always attend the other lecture on Kista or Campus.
Course information on the web!

- Course PM available at 
  http://www.imit.kth.se/courses/2B1512/
- Updates to labs posted as soon as possible (previous years labs available)
- Lecture notes (copy of the slides) posted the day before the lecture.

**2B1512 Digital Design with HDL**

**F1 : Introduction to VHDL**

VHDL Basics

There are two types of VHDL code:
- VHDL targeted for synthesis (the synthesizable subset)
- VHDL targeted for simulation (used for high-level modeling and in test benches, i.e., for testing purposes)

How does the simulation work?

How does the simulation work?

1) Go through all functions. Compute the next value to appear on the output using current input values and store it in a local data area (a value table inside the function).

2) Go through all functions. Transfer the new value from the local table inside to the data area holding the values of the outputs (=inputs to the next circuit)

What is the output of C?

The two-phase simulation cycle

<table>
<thead>
<tr>
<th>NAND gate evaluated first:</th>
<th>AND gate evaluated first:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N:1\rightarrow0)</td>
<td>(N:1\rightarrow0)</td>
</tr>
<tr>
<td>(A:0\rightarrow1)</td>
<td>(A:0\rightarrow1)</td>
</tr>
<tr>
<td>(B:1\rightarrow0)</td>
<td>(B:1\rightarrow0)</td>
</tr>
<tr>
<td>(C:0\rightarrow0)</td>
<td>(C:1\rightarrow0)</td>
</tr>
</tbody>
</table>
Cycle-based simulators

Go through all functions using current inputs and compute next output

Update outputs & increase time with 1 delay unit

Event-based Simulators

Go through all functions whose inputs has changed and compute next output

Update outputs & increase time with 1 delay unit

Event-based simulators with event queues

Go through all functions whose inputs has changed and compute value and time for next output change

Increase time to first scheduled event & update signals

VHDL Simulation Cycle

- VHDL uses a simulation cycle to model the stimulus and response nature of digital hardware

Start Simulation

Update Signals

Delay

Execute Processes

End Simulation

VHDL Delay Models

- Delay is created by scheduling a signal assignment for a future time
- Delay in a VHDL cycle can be of several types
  - Inertial
  - Transport
  - Delta

Inertial Delay

- Default delay type
- Allows for user specified delay
- Absorbs pulses of shorter duration than the specified delay
**Transport Delay**

- Must be explicitly specified by user
- Allows user-specified delay
- Passes all input transitions w/ delay

```vhdl
-- TRANSPORT must be specified
output <= TRANSPORT NOT input AFTER 10 ns;
```

**Delta Delay**

- Delta delay needed to provide support for concurrent operations with zero delay
  - The order of execution for components with zero delay is not clear
- Scheduling of zero delay devices requires the delta delay
  - A delta delay is necessary if no other delay is specified
  - One delta delay is an infinitesimal amount of time
  - The delta is a scheduling device to ensure repeatability

**Example - Delta Delay**

<table>
<thead>
<tr>
<th>Time (ns)</th>
<th>Delta</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>IN: 1 -&gt; 0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>eval inverter</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>A: 0 -&gt; 1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>eval NAND, AND</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>B: 1 -&gt; 0, C: 0 -&gt; 1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>eval AND</td>
</tr>
</tbody>
</table>

**How do we write code?**

```
x y enable
```

**The black box model**

- Primary level of abstraction in VHDL is the entity
- In a behavioral description, the entity is defined by its responses to signals or input
- A behavioral model is similar to a "black box"
  - Interior is hidden from view
  - Behavior of entity is defined by the relationship of the input to the output

```vhdl
ENTITY half_adder IS
    PORT( x, y, enable: IN bit;
         carry, result: OUT bit);
END half_adder;
```

**Entity Declaration**

- An entity declaration describes the interface of the component
- PORT clause indicates input and output ports
- An entity can be thought of as a symbol for a component
Port Declaration

- PORT declaration establishes the interface of the object to the outside world
- Three parts of the PORT declaration
  - Name
  - Mode
  - Data type
- Sample PORT declaration:

```
ENTITY test IS
  PORT(name : mode data_type);
END test;
```

Port Parameters

- Name: any identifier that is not a reserved word
- Mode: in, out, buffer, inout, linkage
- Data type: any declared or predefined data type

Predefined Data types

- bit (‘0’ or ‘1’)
- bit_vector (array of bits)
- integer
- real
- time (physical data type)

Architecture Declaration

- Architecture declarations describe the operation of the component
- Many architectures may exist for one entity, but only one may be active at a time
- An architecture is similar to a schematic of the component

```
ARCHITECTURE behavior1 OF half_adder IS
BEGIN
  PROCESS (enable, x, y)
  BEGIN
    IF (enable = '1') THEN
      result <= x XOR y;
      carry  <= x AND y;
    ELSE
      carry, result <= '0';
    END IF;
  END PROCESS;
END behavior1;
```

Signals - Declaration & Assignments

```
ARCHITECTURE <architecture_name> OF <entity_name> IS
  - The signal declaration is used inside architectures to
    declare internal signals:
    signal a,b,c,d:bit;
    signal a,b,sum:bit_vector(31 downto 0);
  BEGIN
    - The signal assignment is used to describe behaviour:
      sum<=a xor b;
  END <architecture_name>;
```

Testbenches

- Testbenches have three main purposes
  - Generate stimulus for simulation
  - Apply stimulus to the entity under test
  - Compare output responses with expected values

```
ENTITY testbench IS
  -- no PORT statement necessary
END testbench;
ARCHITECTURE example OF testbench IS
  COMPONENT entity_under_test
    PORT(...);
  END COMPONENT;
BEGIN
  Generate_waveforms_for_test;
  Instantiate_component;
  Monitoring_statements;
END example;
```