

## ***ECE 3401 Lecture 4***

---

### ***Introduction to A Hardware Description Language – VHDL***

#### **Outline**

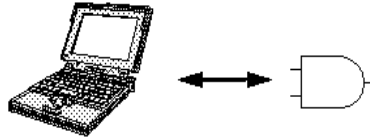
---

- HDL and VHDL Overview
  
- VHDL Language Organization
  
- Basic VHDL Modeling
  - Entity declaration
  - Architecture declaration
  - Data objects
  - Component

## HDL-Overview

---

- What is Hardware ?

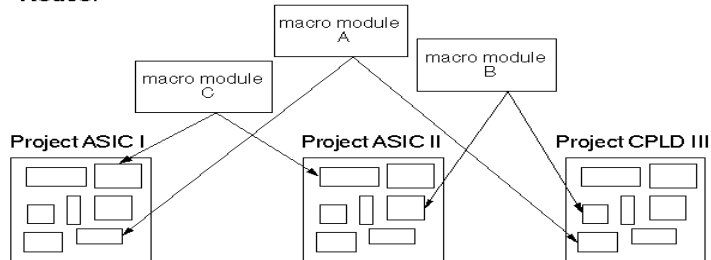


- Hardware Description Language (HDL) = "Programming" language for modeling of (digital) hardware.
- HDL: VHDL (very-high-speed integrated circuits (VHSIC)), Verilog

## Application of HDL

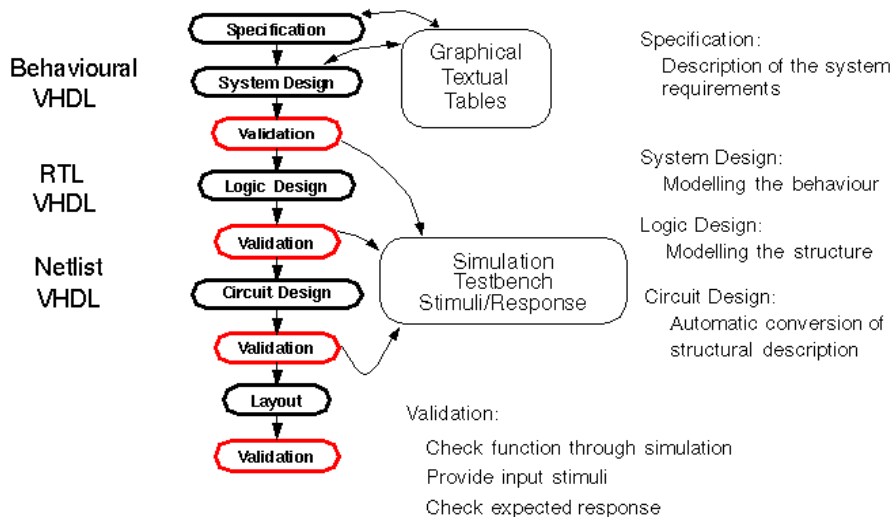
---

Reuse:



- HDL offer design reuse capabilities.
  - The corresponding HDL model can be **re-used in several designs/projects**.
  - Frequently needed function blocks (macros) are collected in **model libraries**.

## Range of Use



## VHDL – Overview

- Very-High-Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL):
  - Modeling of digital systems
  - **Concurrent** and **sequential** statements
  - Machine-readable specification
  - Man- and machine-readable documentation
- International Standards:
  - IEEE Std 1076-1987 - IEEE Std 1076-1993

## VHDL Information

---

- **Reference Books for VHDL:**

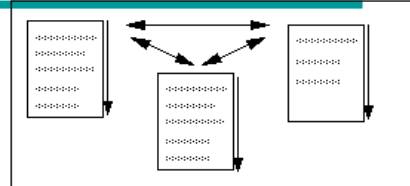
- Peter J. Ashenden, "The Student's Guide to VHDL (2<sup>nd</sup> Edition)," Morgan Kaufmann.
- Peter J. Ashenden, "The Designer's Guide to VHDL (2<sup>nd</sup> Edition)," Morgan Kaufmann.
- J. Bhasker "A VHDL Primer (3<sup>rd</sup> Edition)", Prentice Hall
- Yalamanchili "Introductory VHDL from Simulation to Synthesis"
- D. L. Perry "VHDL Programming by Example" McGrawHill

- **Useful web site for VHDL:**

- <http://www.eda.org/rassp/vhdl>
- <http://www.peakfpga.com/vhdlref/index.html>

## Concepts of VHDL

---



- **Execution of assignments:**

- **Sequential:**

- Executed one after another, like in software programming languages.
- Can override the effects of previous statements.

- **Concurrent:**

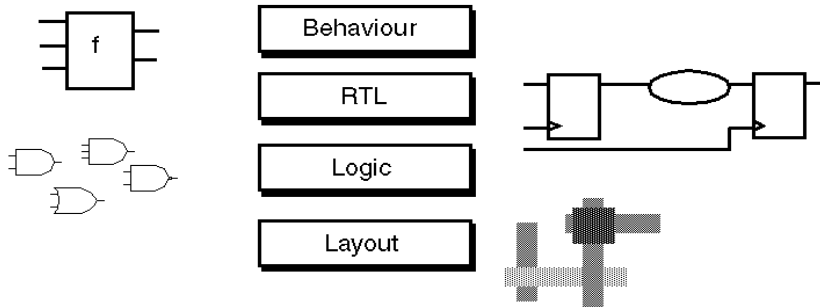
- Active continuously.
- The order of statements is not relevant.
- Suited to model the parallelism of hardware.

- **Abstraction:** description of different parts of a system.

- On every abstraction level, only the essential information is considered, nonessential information is left out.

## Abstraction levels in Digital Design

---



### ▪ Behavioral level:

- Functional description of the model is outlined.
- No system clock, signal transitions are asynchronous with respect to the switching time.
- *Simulation only*, but *typically not synthesizable*.

## Abstraction levels in Digital Design

---

### ▪ Register level (RTL):

- The design is divided into combinational logic and storage elements.
- Storage elements (Flip-Flops, latches) are controlled by a system clock.
- *Synthesizable*.

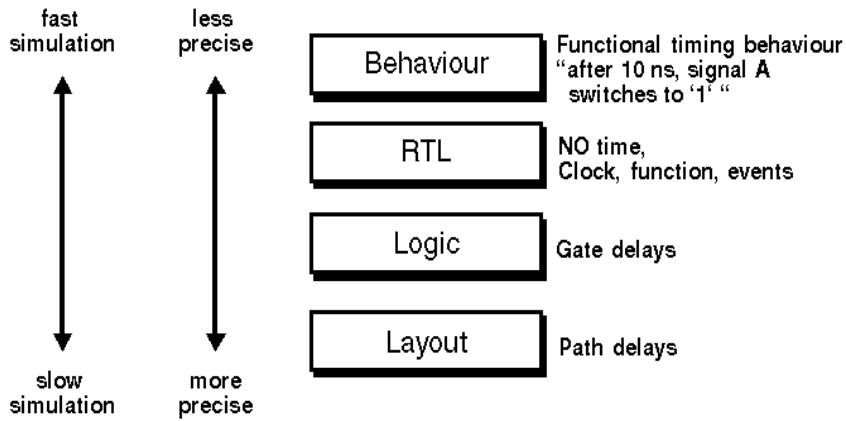
### ▪ Logic level:

- The design is represented as a *netlist* with logic gates (AND, OR, NOT,...) and storage elements.

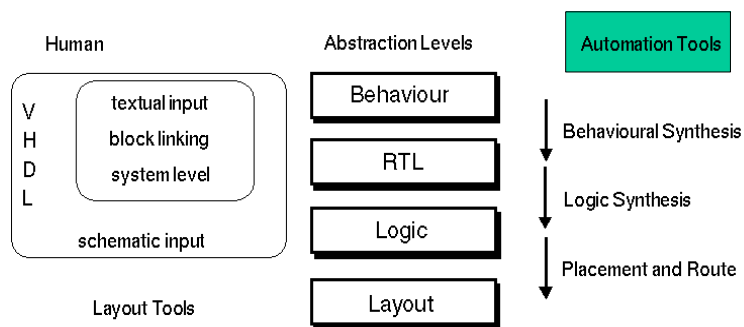
### ▪ Layout level:

- The different cells of the target technology are placed on the chip and the connections are routed.
- After the layout has been verified, the circuit is ready for the production process (submitted to manufacturing factory for chip fabrication).

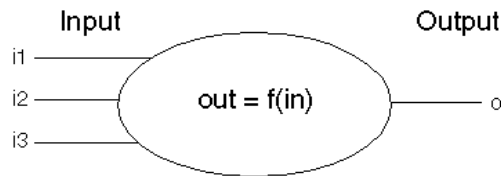
## Information Content of Abstraction Levels



## Abstraction levels and VHDL

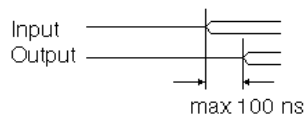


## Behavioral Description in VHDL



`o <= transport i1 + i2 * i3 after 100 ns;`

Specification:



- The function can be modeled as a simple equation (e.g.  $i1+i2*i3$ ) plus a delay of 100 ns.

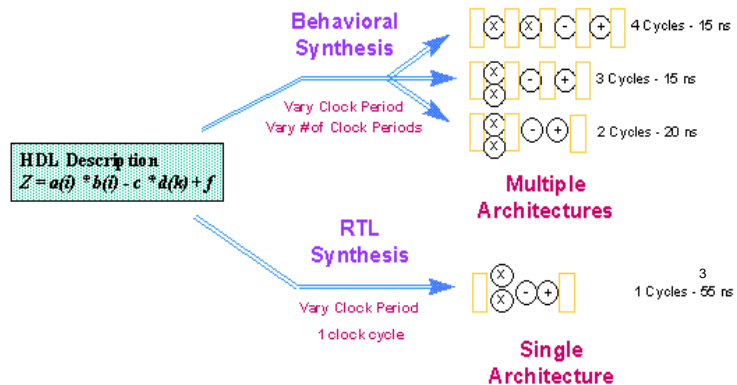
## Behavioral Synthesis

- Only very simple behavioral models are synthesizable.
- Applications: design of RAM cells for the target technology, where only the generic parameters (width, depth, number of ports, (a)synchronous,...) need to be specified.
- Advantages
  - Explore Architectural Tradeoffs
  - Automatic generation of state machines
  - Fewer lines of VHDL code
  - Faster simulation than RTL
- Disadvantages
  - No timing information
  - Tools are not mature
  - May not be appropriate for non-algorithmic designs

## RTL Synthesis

- RTL synthesis implements all functionality within a single clock cycle
- Behavioral synthesis automatically allocates the functionality across multiple clock cycles

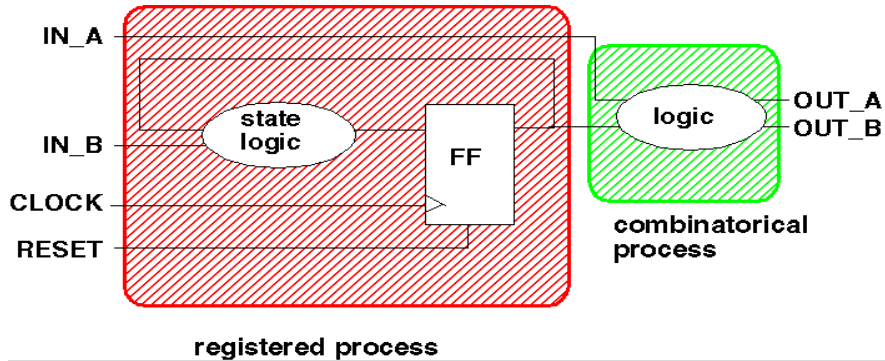
## Behavioral/RTL Synthesis



From Synopsys Behavioral Compiler Presentation

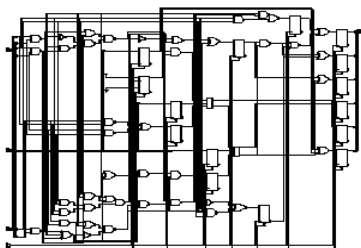


## Register Transfer Level in VHDL



- Functional behavior is modeled with registered process (clocked process) and combinational process.
- RTL VHDL code contains some sort of structural information in addition to the functional behavior.

## Gate Level in VHDL



```

U86 : ND2 port map( A => n192, B => n191, Z => n188);
U87 : ND2 port map( A => I3_2, B => I2_0, Z => n175);
U88 : ND2 port map( A => I2_2, B => I3_0, Z => n173);
U89 : NR2 port map( A => mul_36_PROD_not_0,
                  B => n174, Z => n185);
U90 : EN port map( A => n181, B => n182, Z => n180);
U91 : ND2 port map( A => I3_2, B => I2_1, Z => n181);
U92 : ND2 port map( A => I2_2, B => I3_1, Z => n182);
U93 : IVP port map( A => n180, Z => n192);
U94 : AO6 port map( A => n173, B => n174, C => n175,
                  Z => n172);
U95 : NR2 port map( A => n174, B => n173, Z => n176);
U96 : ND2 port map( A => I3_1, B => I2_1, Z => n174);
U97 : EN port map( A => n183, B => n178,
                  Z => product64_4);
U98 : ND3 port map( A => I2_2, B => I3_2, C => n174,
                  Z => n183);
    
```

- Contains a list of the gates components (e.g. ND2, NR2, AO6).
- Each single element of the circuit (e.g. U86) is instantiated as a component (e.g. ND2) and connected to corresponding signals (n192, n191, n188).

## VHDL Language & Syntax (General)

-----  
-- Example VHDL Code --  
-----

```
signal mySignal: bit;           -- an example signal

MYsignal <= '0',                -- start with '0'
           '1' after 10 ns,      -- and toggle
           '0' after 20 ns,      -- every 10 ns
           '1' after 30 ns;
```

- Signal assignment: '`<=`'
- User defined names:
  - Letters, numbers, underscores.
  - Start with a letter.
  - No VHDL keyword may be used.
  - Case insensitive
- List delimiter: ';'.
- Statements are terminated by ';' (may span multiple lines)
- Comments: '--' until end of line

## VHDL Language & Syntax (Identifier)

MySignal\_23 -- normal identifier  
rdy, RDY, Rdy -- identical identifiers  
vector\_&\_vector -- X : special character  
last of Zout -- X : white spaces  
idle\_\_state -- X : consecutive underscores  
24th\_signal -- X : begins with a numeral  
open, register -- X : VHDL keywords

- Normal Identifier:
  - Letters, numbers, underscores
  - Case insensitive.
  - The first character must be a letter.
  - The last character cannot be an underscore.
  - No two consecutive underscores.
  - VHDL reserved words may not be used as identifiers.

\mySignal\_23\ -- extended identifier  
\rdy\, \RDY\, \Rdy\ -- different identifiers  
\vector\_&\_vector\ -- legal  
\last of Zout\ -- legal  
\idle\_\_state\ -- legal  
\24th\_signal\ -- legal  
\open\, \register\ -- legal

- Extended Identifier (VHDL93)
  - Enclosed in back slashes
  - Case sensitive
  - Graphical characters allowed
  - May contain spaced and consecutive underscores.
  - VHDL keywords allowed

## Legal and Illegal Identifiers

---

- Legal Identifiers:
  - Uconn\_huskies
  - ECE\_252
  - Sel6B
  
- Illegal Identifiers:
  - `_time_is_9am` -- an identifier must start with a letter.
  - `8thsemester` -- an identifier must start with a letter.
  - `Homework#1` -- letter, digits, and underscore only.
  - `final__example` -- two underscore in succession not allowed
  - `Entity` -- keyword cannot be used as identifier
  - `Time_out_` -- last character cannot be an underscore.

## VHDL Reserved Words

---

abs	disconnect	label	package	sla
access	downto	library	port	sll
after	else	linkage	postponed	sra
alias	elsif	literal	procedure	srl
all	end	loop	process	subtype
and	entity	map	protected	then
architecture	exit	mod	pure	to
array	file	nand	range	transport
assert	for	new	record	type
attribute	function	next	register	unaffected
begin	generate	nor	reject	units
block	generic	not	rem	until
body	group	null	report	use
buffer	guarded	of	return	variable
bus	if	on	rol	wait
case	impure	open	ror	when
component	in	or	select	while
configuration	inertial	others	severity	with
constant	inout	out	shared	xnor
	is		signal	xor

## VHDL Structural Elements

---

- **Entity:** description of **interface** consisting of the port list.
  - The primary hardware abstraction in VHDL, analogous to a symbol in a block diagram.
- **Architecture:** description of the **function** of the corresponding module.
- **Process:** allows for a **sequential** execution of the assignments
- **Configuration:** used for simulation purposes.
- **Package:** hold the definition of commonly used data types, constants and subprograms.
- **Library:** the logical name of a collection of compiled VHDL units (object code).
  - Mapped by the simulation or synthesis tools.

## Basic VHDL Concepts

---

- Interfaces
- Behavior
- Structure
- Test Benches
- Analysis, elaboration, simulation
- Synthesis

*A VHDL Design Unit consists of an Entity Declaration and an Architecture Body.*

**Entity Declaration:**

Names entity and defines interfaces between entity and its environment.

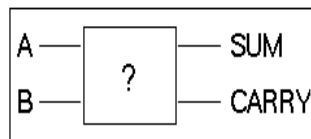
```
ENTITY entity_name IS
PORT ( name_list : mode type);
END entity_name ;
```

**Architecture Body:**

Establishes relationship between inputs and outputs of design.

```
ARCHITECTURE body_name OF entity_name IS
-- declarative_statements
BEGIN
-- activity_statements
END body_name;
```

## Entity statement



```
entity HALFADDER is
port(
  A, B:          in  bit;
  SUM, CARRY:    out bit);
end HALFADDER;
```

```
entity ADDER is
port(
  A, B:          in  integer range 0 to 3;
  SUM:          out integer range 0 to 3;
  CARRY:        out bit );
end ADDER;
```

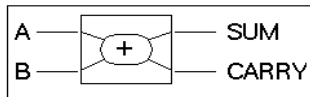
- **Interface** description
- Port clause identifies ports used by "entity" to communicate with its environment.
- Port signals
  - Data types: bit, int, ...
  - Signal width: 0 to 3
  - Signal direction: in, out, ...

## Architecture Body Statement

```
entity HALFADDER is
port(
  A, B:    in bit;
  SUM, CARRY: out bit);
end HALFADDER;
```

```
-- Architecture body
-- "Hadd" is user defined name
```

```
architecture Hadd of HALFADDER is
begin
  SUM <= A xor B;
  CARRY <= A and B;
end architecture Hadd;
```



- An architecture defines an entity's behavior from a simulation point of view.
- Implementation of design
- Always connected with a specified entity
  - One entity can have several architectures
  - Entity ports are available as signals within the architecture.
- Contains concurrent statements.

## Architecture Body Structure

```
-- architecture Body
```

```
architecture EXAMPLE of STRUCTURE is
```

```
-- Declarative part
```

```
subtype DIGIT is integer range 0 to 9;
constant BASE: integer := 10;
signal DIGIT_A, DIGIT_B: DIGIT;
signal CARRY: DIGIT;
```

```
begin
```

```
-- Statement part
```

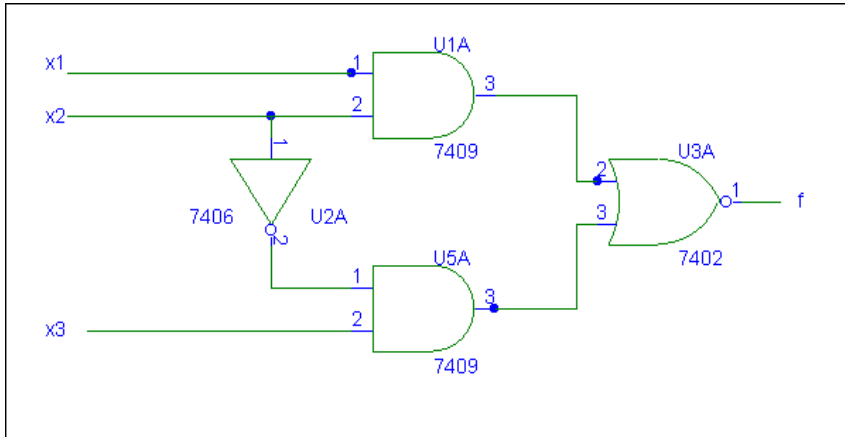
```
DIGIT_A <= 3;
SUM <= DIGIT_A + DIGIT_B;
DIGIT_B <= 7;
CARRY <= 0 when SUM < BASE else 1;
```

```
end EXAMPLE ;
```

- Declarative part:
  - data types
  - constants
  - additional signals ("actual" signals)
  - components
  - ...
- Statement part (after 'begin'):
  - signal assignments
  - processes
  - component instantiations
  - all concurrent statements that can be placed within the statement part..

## Example

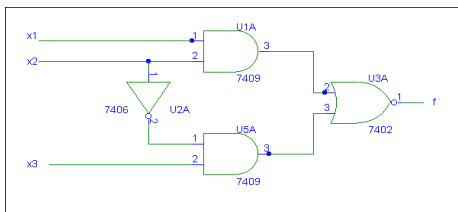
Write a VHDL code for the circuit shown below. The inputs to the circuit are x1, x2, x3, and the output is f ?



## Example: VHDL code

```
ENTITY example1 IS
    PORT (x1, x2, x3: IN BIT;
          f: OUT BIT);
END example1;

ARCHITECTURE logicFunc OF example1 IS
    BEGIN
        -- Architecture statement region
        f <= (x1 AND x2) NOR (NOT x2 AND x3);
    END logicFunc;
```



• **logicFunc**: user-defined name