**What is VHDL ?**

**VHDL** stands for **V**HSIC (Very High Speed Integrated Circuits) **H**ardware **D**escription **L**anguage. In the mid-1980’s the U.S. Department of Defense and the IEEE *(Institute of Electrical and Electronics Engineers)* sponsored the development of this hardware description language. Its goal was to develop very high-speed integrated circuit. It has become now one of industry’s standard languages used to describe digital systems. The other widely used hardware description language is **Verilog**. Both are powerful languages that allow us to describe and simulate complex digital systems. A third HDL language is **ABEL** (Advanced Boolean Equation Language) which was specifically designed for Programmable Logic Devices (PLD). ABEL is less powerful than the other two languages and is less popular in industry.

The initial version of VHDL, designed to [IEEE](https://en.wikipedia.org/wiki/IEEE) standard IEEE 1076-1987, included a wide range of data types, including numerical ([integer](https://en.wikipedia.org/wiki/Integer) and [real](https://en.wikipedia.org/wiki/Real_data_type)), logical ([bit](https://en.wikipedia.org/wiki/Bit) and [boolean](https://en.wikipedia.org/wiki/Boolean_datatype)), [character](https://en.wikipedia.org/wiki/Character_(computing)) and [time](https://en.wikipedia.org/wiki/Time), plus [arrays](https://en.wikipedia.org/wiki/Array_data_type) of bit called bit vector and of character called [string](https://en.wikipedia.org/wiki/String_(computer_science)).

**7.1. RTL design with VHDL**

Register Transfer Level, or RTL design lies between a purely behavioral description of the desired circuit and a purely structural one. An RTL description describes a circuit’s registers and the sequence of transfers between these registers but does not describe the hardware used to carry out these operations.

The steps in RTL design are: (1) determine the number and sizes of registers needed to hold the data used by the device, (2) determine the logic and arithmetic operations that need to be performed on these register contents, and (3) design a state machine whose outputs control how the register contents are updated in order to obtain the desired results. Producing an RTL design is similar to writing a computer program in a conventional programming language. Choosing registers is the same as choosing variables. Designing the flow of data in the “datapath” is analogous to writing expressions involving the variables (registers) and operators (combinational functions). Designing the controller state machine is similar to deciding on the flow of control within the program (if/then/else, while-loops, etc).

In [digital circuit design](https://en.wikipedia.org/wiki/Integrated_circuit_design#Digital_design), **register-transfer level** (**RTL**) is a design **abstraction** which models a [synchronous](https://en.wikipedia.org/wiki/Synchronous_circuit) [digital circuit](https://en.wikipedia.org/wiki/Digital_circuit) in terms of the flow of digital signals (data) between [hardware registers](https://en.wikipedia.org/wiki/Hardware_register), and the [logical operations](https://en.wikipedia.org/wiki/Boolean_logic) performed on those signals. It is used in [hardware description languages](https://en.wikipedia.org/wiki/Hardware_description_language) (HDLs) like [Verilog](https://en.wikipedia.org/wiki/Verilog) and [VHDL](https://en.wikipedia.org/wiki/VHDL) to create high-level representations of a circuit. From this lower-level representations and finally **actual** wiring can be derived. Design at the RTL level is typical practice in modern digital design.

When designing digital integrated circuits with a [hardware description language](https://en.wikipedia.org/wiki/Hardware_description_language), the designs are usually engineered at a higher level of abstraction than transistor level ([logic families](https://en.wikipedia.org/wiki/Logic_family)) or logic gate level. In HDLs, **the designer declares the registers** (which roughly correspond to **variables** in computer programming languages), and describes the combinational logic by using **constructs** that are familiar from programming languages such as **if-then-else** and **arithmetic operations**. This level is called *register-transfer level*. The term refers to the fact that **RTL focuses on describing the flow of signals between registers.**

As an example, the circuit mentioned above can be described in VHDL as follows:

D <= not Q;

process(clk)

begin

if rising\_edge(clk) then

Q <= D;

end if;

end process;

[**RTL vs HDL? Whats the difference**](http://electronics.stackexchange.com/questions/69022/rtl-vs-hdl-whats-the-difference) **?**

HDL is the catch all name for all hardware description languages (Verilog,VHDL, etc.) in the same way Object Oriented can refer to C++, Java, etc. RTL, on the other hand, is **a way of describing a circuit**. You write your RTL level code in an HDL language which then is translated (by synthesis tools) to gate level description in the same HDL language or whatever your target device/process will take.

Here is a line of Verilog (HDL) describing a mux in RTL:

assign mux\_out = (sel) ? din\_1 : din\_0;

Your synthesis tool can take that and convert it to a set of logic gates, or just a mux macro that is supported by your end device. For example it might instantiate a mux macro

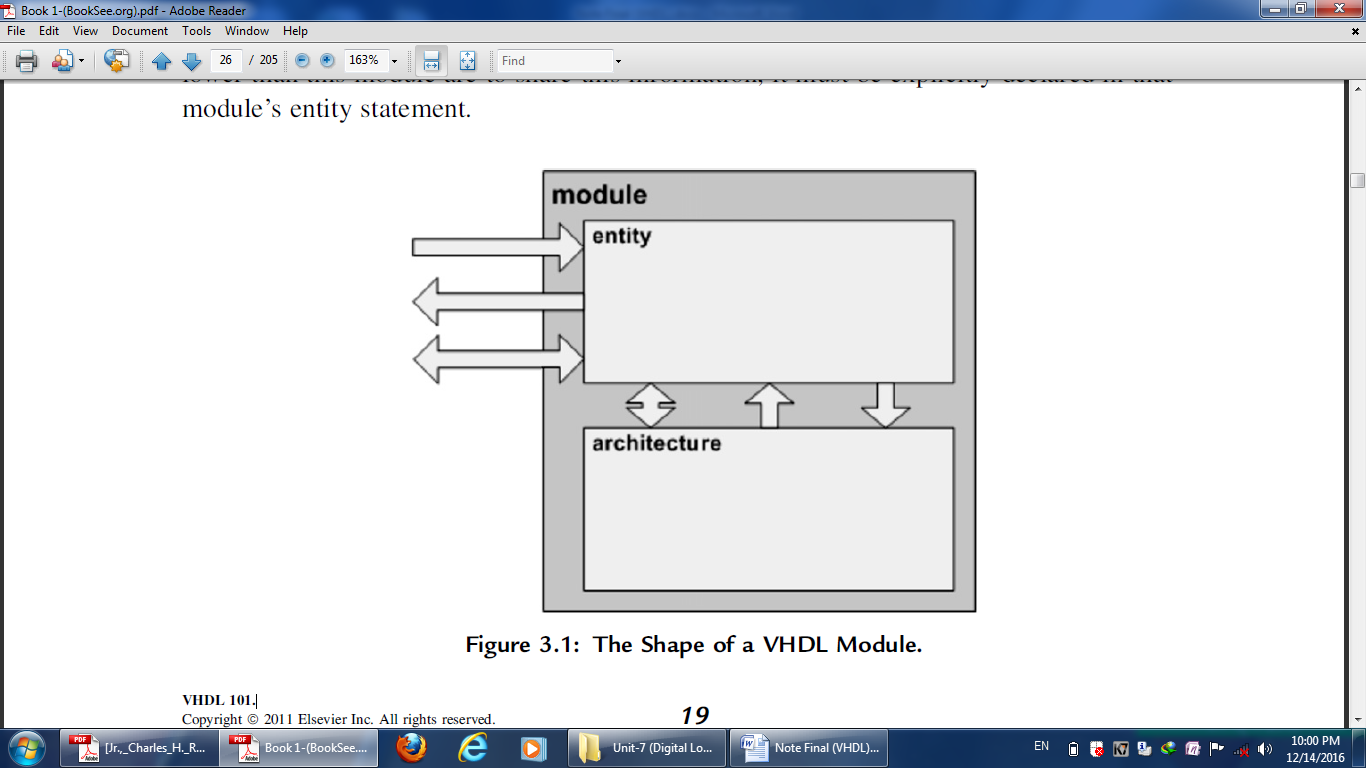
mux u3 (mux\_out, din\_1, din\_0);

In both cases you can feed the same inputs to the block (RTL, or gate-level) and your output should be the same. In fact there are tools that check the output of your synthesis against your RTL code. It makes sure the tool didn't accidentally optimize or change something during synthesis that caused a mismatch. This is called Formal Verification.

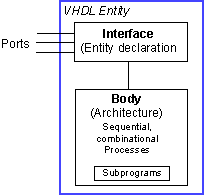
**7.1.1. Shape (Structure) of VHDL**

Shape refers to the typical parts of a single VHDL module. A VHDL module contains two basic parts –( a) one covers the connections made between this module and other modules (the ‘‘outside world’’ which may be the pins on an FPGA package or other modules within the design) and (b) the other part, an unambiguous description of the behavior of the module. Modules may be combined in parallel or hierarchically.

In the case of a module, the only information **from the outside world or higher level that this module can access (either see or modify) is listed in the entity statement.** Other information can be defined in both the entity and the architecture sections and can be used freely within this module. If modules hierarchically lower than this module are to share this information, it must be explicitly declared in that module’s entity statement.



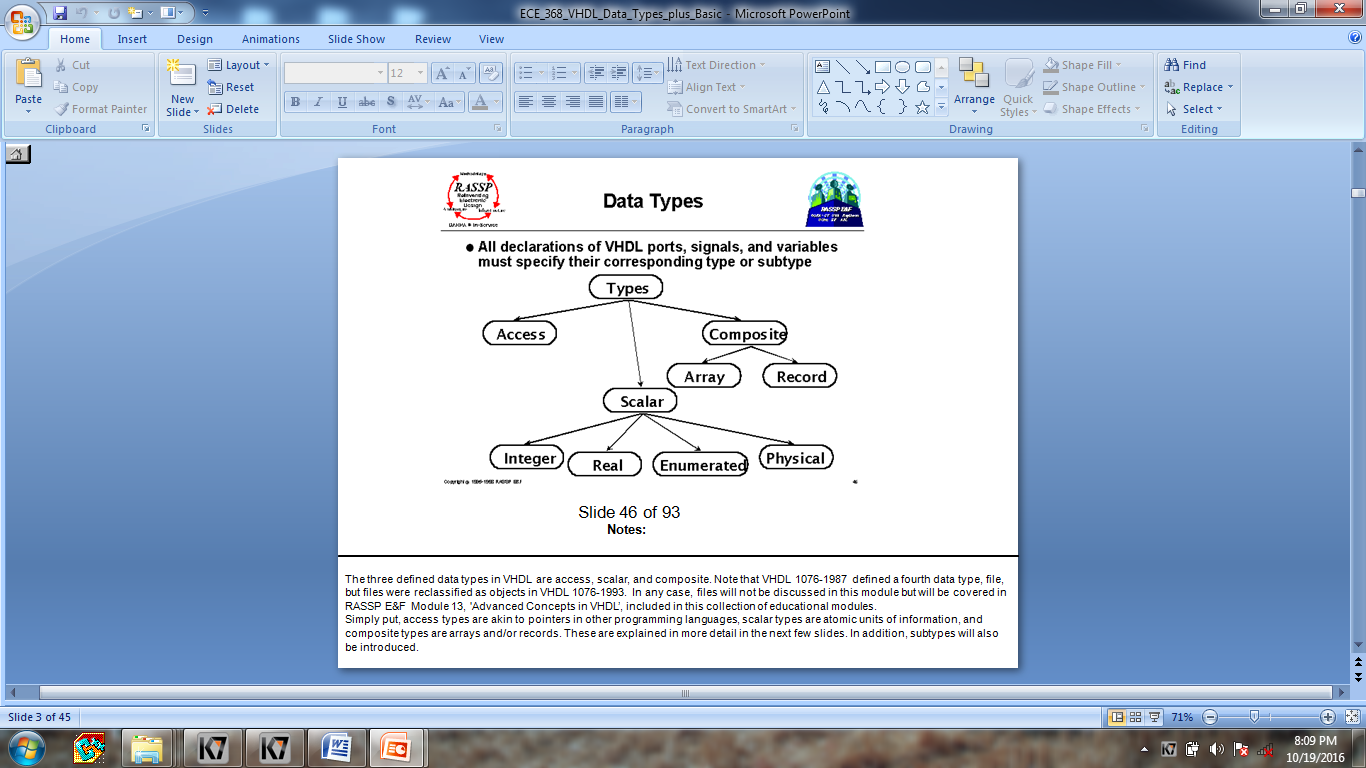
A digital system in VHDL consists of a design **entity** that can contain other entities that are then considered components of the top-level entity. Each entity is modeled by an *entity declaration* and an *architecture body*. We can consider the entity declaration as the **interface to the outside world that defines the input and output signals**, while **the architecture body contains the description of the entity and is composed of interconnected entities, processes and components, all operating concurrently, as schematically shown in Figure** below. In a typical design there will be many such entities connected together to perform the desired function.

****

VHDL uses reserved **keyword**s that cannot be used as signal names or identifiers. Keywords and user-defined identifiers are **case insensitive**. Lines with comments start with two adjacent hyphens (--) and will be ignored by the compiler. VHDL also ignores line breaks and extra spaces. VHDL is **a strongly typed** language which implies that one has always to declare the [type](http://www.seas.upenn.edu/~ese201/vhdl/#DataTypes) of every object that can have a value, such as signals, constants and variables.

**Data Types**

The type defines the set of values that the object can have and the set of operations that are allowed on it. The notion of *type* is key to VHDL since it is a strongly typed language that requires each object to be of a certain type. In general one is not allowed to assign a value of one type to an object of another data type (e.g. assigning an integer to a bit type is not allowed). There are four classes of data types: **scalar**, **composite**, **access** and **file** types. The scalar types represent a single value and are ordered so that [relational](http://www.seas.upenn.edu/~ese201/vhdl/#RelationalOperators) operations can be performed on them. The **scalar** type includes **integer, real, and enumerated types of Boolean and Characte**r.



**Concurrent Statements**

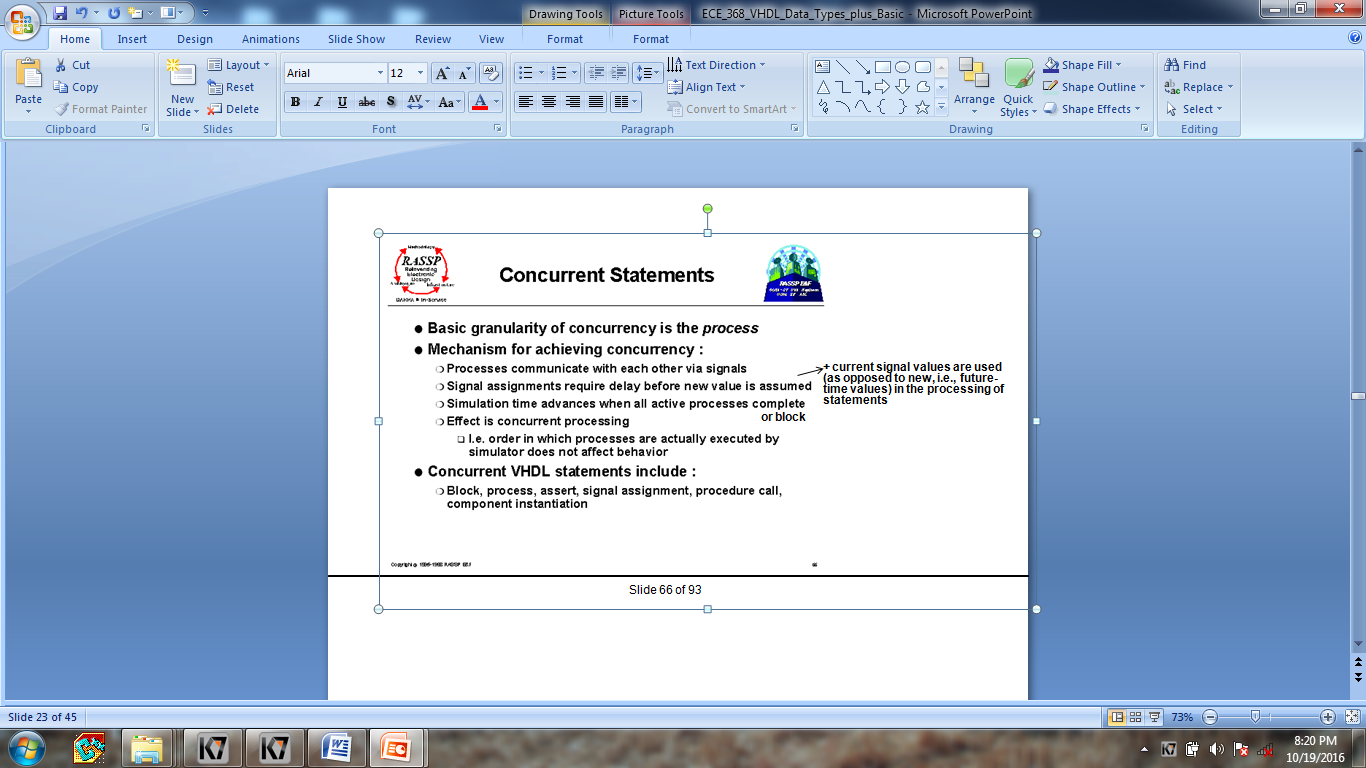
A VHDL architecture contains a set of concurrent statements. Each concurrent statement defines one of the interconnected blocks or processes that describe the overall behavior or structure of a design. Concurrent statements in a design execute continuously, unlike sequential statements, which execute one after another. The two main concurrent statements are:

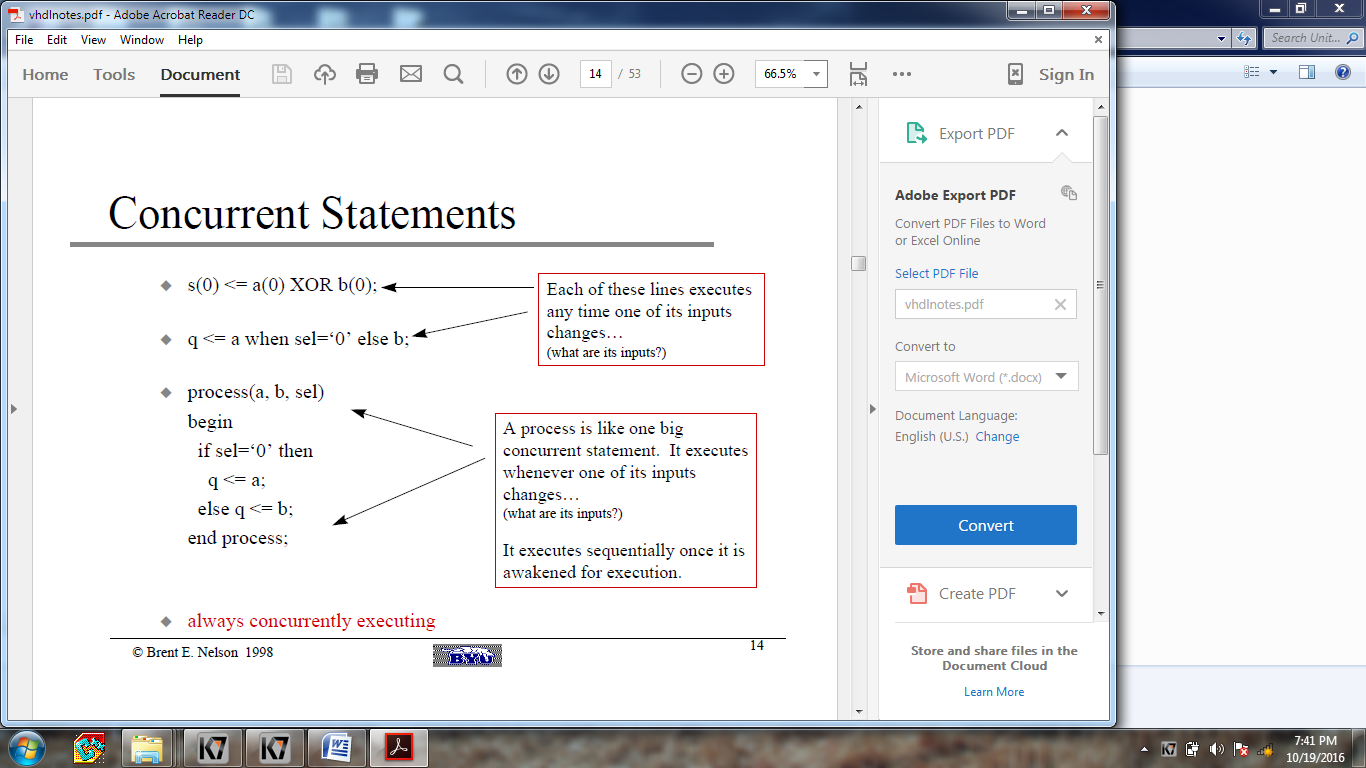
**process statement :**

A process statement defines a process. Processes are composed of sequential statements , but processes are themselves concurrent statements. All processes in a design execute concurrently. However, at any given time only one sequential statement is interpreted within each process. A process communicates with the rest of a design by reading or writing values to and from signals or ports declared outside the process.

**block statement:**

A block statement defines a block. Blocks are named collections of concurrent statements, optionally using locally defined types, signals, subprograms, and components.



****

**What is a process ?**

* **A process statement is a concurrent statement, but all statements contained in it are sequential statement (statements that executes serially, one after the other).**
* **The use of processes makes your code more modular, more readable, and allows you to separate combinational logic from sequential logic.**
* **List of all signals that the process is sensitive to. Sensitive means that a change in the value of these signals will cause the process to be invoked.**

**VHDL Object Types**

* **Constants**
* **Signals: It is a physical signal (you can think of it like a piece of wire. A signal is a sequence of time-value pairs. A signal assignment takes effect only after a certain delay (the smallest possible delay is called a “delta time”). It is possible to define global signals (signals that can be shared among entities). But more often signals are just locally defined for a given architecture**
* **Variables: Assignment to variables are scheduled immediately (the assignment takes effect immediately). If a variable is assigned a value, the corresponding location in memory is written with the new value while destroying the old value. This effectively happen immediately so if the next executing statement in the program uses the value of the variable, it is the new value that is used. Typically, variables are used as a local storage mechanism, visible only inside a process.**
* **Files**

**Signals vs. Variables**

* **Signals assignments are scheduled after a certain delay d**
* **Variables assignments happen immediately, there is no delay**

**Attributes**

Types and objects declared in a VHDL description can have additional information, called attributes, associated with them. In other words, attributes provide information about certain items in VHDL. They may be used to communicate information about many different items in VHDL. Similarly, attributes can return various types of information. For example, an attribute can be used to determine the depth of an array, its range, its leftmost index, etc. Additionally, the user may define new attributes to cover specific situations. This capability allows user-defined constructs and data types to use attributes. An example of the use of attributes is in assigning information to a VHDL construct, such as board location, revision number, etc.

**Some predefined attributes are :**‘left 🡪 the leftmost value of a type  
‘right  
‘high 🡪 the greatest value of a type  
‘low  
‘length 🡪 the number of elements in an array

‘event 🡪 a change on a signal or variable  
‘range 🡪 the range of the elements of an array object

**Finite State Machine (FSM)**

The FSMs consist of three parts: **state register**, **next-state logic**, and **output logic**. Based on the number of processes used to describe the FSM three approaches can be distinguished:

**one-process**: The parts of the FSM are described in the same clocked process. The outputs are buffered. That means, that the resource requirement is high but the timing is reliable (the control signals are glitch-free) and the clock-to-output delay is low.

**two-process**: The next-state logic and the state register are described in the same process but the output logic is modeled with a separate combinatorial process. The outputs are not buffered, so the resource requirement is lower but the timing is less reliable, because the mealy-inputs may cause timing violations.

**three-process**: An own process is assigned to the structural parts of the FSM respectively. The HDL code is difficult to read and the mealy-inputs may cause timing violations.