EE214 Short Notes

Aditya Byju

Course Professor: Prof. D. K. Sharma, Prof. Mariam Shojaei Baghini
Ref: Prof's video lectures and slides
The best is yet to be!

Digital Circuits Lab

September 2021

Contents

Introduction

- **ceramic capacitors** - a type of low capacitance capacitors that have no polarity

- **electrolytic capacitors** - these type of capacitors have higher capacitance values than ceramic capacitors and they do have polarity

Polarized Electrolytic Capacitor and its electric Symbol

Non-Polarized Ceramic Capacitor and its electric Symbol
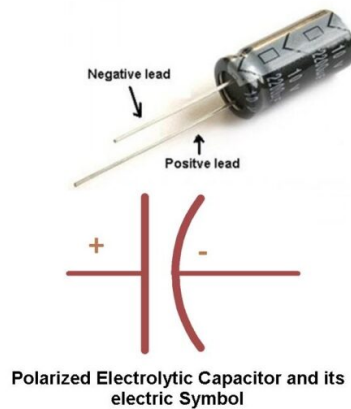
- Single-turn and multi-turn potentiometers are variable resistances, also multi-turn potentiometers provide finer tuning of resistances

- **hardware description language (HDL)** - is a specialized computer language used to describe the structure and behaviour of electronic circuits, and most commonly, digital logic circuits

- Very High Speed Integrated Circuit HDL (VHDL) - is a hardware description language which uses the syntax of ADA

- Uses of HDL:
  - For describing hardware
  - As a modelling language
  - For simulation of hardware
  - For early performance estimation of system architecture
  - For synthesis of hardware
  - For fault simulation, test and verification of designs

- **entity** - represents a template for a hardware block, describing its interface with other modules in terms of input and output signals

- **testbench** - includes the circuit being designed, blocks which apply test signals to it and those which monitor its output

- **architecture** - describes how an entity operates. An architecture is always associates with an entity. There can be multiple architectures associated with an entity.

```
entity name is
    generic(list);
    port(list);
end entity name;
```

```
architecture name of name_of_entity is
    (declarations)
begin      (concurrent statements)
end architecture name;
```

Figure : Entity declaration ($\geq$ VHDL 93)

Figure : Architecture declaration ($\geq$ VHDL 93)

- The architecture inherits the port signals from its entity. Concurrent statements constituting the architecture can be placed in any order.

- Signals are carried by wires, variables are used for array indices, loop counters, etc.

- When you assign a value to a variable the assignment becomes effective immediately, however when you assign a value to a signal there is an associated delay for this assignment aas it is there in a real circuit

- **component** - an entity ↔ architecture pair

```
component name is
    generic(list);
    port(list);
end component name;
```

Figure : Component declaration (≥ VHDL 93)

- **configuration** - describes linkages between component types and entity ↔ architecture pairs

- Related declarations and design elements like subprograms and procedures can be placed in a package for re-use. A package has a declarative part and an implementation part.

- Objects in a package can be referred to by a packagename.objectname syntax

- A description can include a use clause to incorporate the package in the design. Objects in the package then becomes visible to the description without having to use the dot reference as above.

- Many design elements such as packages, definitions and entire entity ↔ architecture pairs can be placed in a library

- Any description invokes a library by first declaring it (For e.g. Library IEEE;)

- Objects in the library can then be incorporated in the design by a use clause (For e.g. Use IEEE.std_logic_1164.all;)

- Various types of objects in VHDL are constants, variables, signals and files.

- Declaration of objects include their object type as well as the data type of values that they can acquire (For e.g. signal Enable: BIT;)
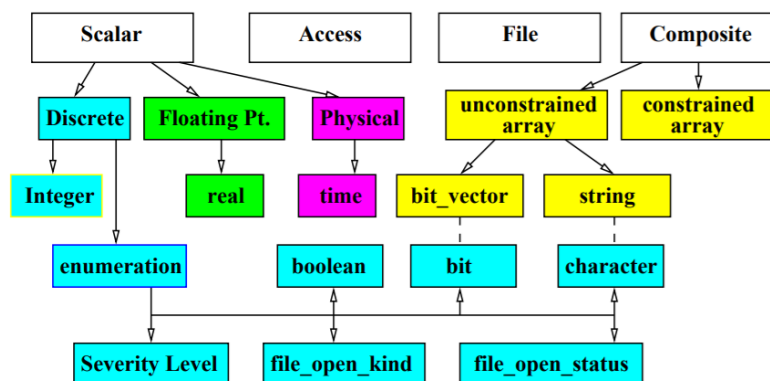

Figure : VHDL Data types

- VHDL enumeration types allow us to define a set of values that a variable of this type can acquire. For example, we can define a data type by the following declaration:

    type instr **is** (add, sub, adc, sbb, rotl, rotr);

- Enumeration types pre-defined in the language are:

    - **type** bit **is** ('0', '1');
    - **type** boolean **is** (false, true);
    - **type** severity_level **is** (note, warning, error, failure);
    - **type** file_open_kind **is** (read_mode, write_mode, append_mode);
    - **type** file_open_status **is** (open_ok, status_error, name_error, mode_error);
    - In addition to the above types, the character type enumerates all the ASCII characters

- **std_logic** - is a signal which can take 1 of 9 possible values. It is defined by:

    type std_logic **is** ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');

- Object which are declared to be of Physical type, carry a value as well as a unit. These are used to represent physical quantities such as time, resistance and capacitance. Time is the only physical type, which is pre-defined in the language. We can define other physical types when required.

- Composite data types are collections of scalar types. VHDL recognizes records and arrays as composite data types. Records are like structures in C. Arrays are indexed collections of scalar types.

- Arrays can be constrained or unconstrained:

    - In constrained arrays, the type definition itself places bounds on index values
    - In unconstrained arrays, no bounds are placed on index values

- VHDL defines two built-in types of arrays:

    - bit_vector

        type bit_vector **is array** (natural range <>) **of** bit;

    - strings

        type string_vector **is array** (positive range <>) **of** character;

Structural description in VHDL

- Structural style describes a design in terms of components and their interconnections. Each component declares its ports and the type and direction of signals that it expects through them.

- A purely structural architecture for an entity will consist of:

    - **Component declarations:** to associate component types with their port lists.
    - **Signal declarations:** to declare the signals used.
    - **Component instantiations:** to place component instances and to portmap their ports to signals. Signals can be internal or port signals declared by the entity.
    - **Configurations:** to bind component types to entity ↔ architecture pairs.
    - **Repetition grammar:** for describing multiple instances of the same component type – for example, memory cells or bus buffers.

- When we associate a component type with a previously defined entity ↔ architecture pair, the chosen architecture could itself contain other components - and these components in turn would be associated with other entity ↔ architecture pairs. This hierarchical association can be described by a standalone design unit called a configuration.

```
configuration config-name of name_of_entity is
    for name_of_architecture
        for componentinstance-namelist: componenttype-name
            use entity name_of_entity(name_of_architecture);
        end for
    end for
end configuration config-name;
```

Figure : Simplified configuration construct

- In VHDL, as we describe entities and architectures, these are compiled into a special library called work. This library is always included and does not have to be declared.

File I-O in VHDL

- In VHDL, in order to use files, we use a two step procedure:
  - We declare a file type first. This associates a file type with the kind of objects that files of this type will contain.
  - We can then declare files of this file type. The file declaration associates a VHDL filename with a file type and optionally, with a Physical file name and file mode (read, write or append).

- File type declaration:
  type FileType **is file of** DataType;

- If a file has not been opened during its declaration, it can be opened later by specific statements

- Opening files:
  procedure file_open(**file** f: FileType;
         Phys_name: in string;
         open_kind: in file_open_kind:=read_mode);

- Closing files:
  procedure file_close(**file** f: FileType);

- Reading from files:
  procedure read(**file** f: FileType; value: out Data_type);

- Writing to files:
  procedure write(**file** f: FileType; value: in Data_type);

Behavioural description in VHDL

- Behavioural style describes a design in terms of its behaviour, and not in terms of a netlist of components. We describe behaviour through "if-then-else" type of constructs, loops, sequential and concurrent assignment statements. Statements like "if-then-else" are inherently sequential. These must therefore occur only inside sequential bodies like processes.

- A concurrent assignment can be made conditionally by using 'when' clauses:

  name <= [delay-mechanism]
  waveform **when** Boolean-expression **else**
  waveform **when** Boolean-expression;

- VHDL operators:

  - **Logical operators:** AND, OR, NAND, NOR, OR, XNOR and NOT
  - **Relational operators:** =, /, <, <=, >, >=
  - **Shift operators:** SLL (logical left), SLA (arithmetic left) SRL (logical right), SRA (Arithmetic right), ROL (rotate left) and ROR (rotate right)

- Sequential constructs need to be placed inside a process. A process uses the syntax:

  [process-label:] **process** [(sensitivity-list)][**is**]
  [declarations]
  begin
      [sequential statements]
  end process [process-label];

  Sequential statements include "if" constructs, case statements, looping constructs, assertions, wait statements etc.

JTAG and Scanchain

JTAG:

- Joint Test Action Group

- Industry standard for verifying designs and testing PCBs

- Connects to an on-chip Test Access Port (TAP) controller

- TAP controller implements a protocol to access a set of registers or pins of a system

- Response is then stored into Flip-Flops and then compared with a golden response

Scanchain:

- Scan chain is a technique used in design for testing. The objective is to make testing easier by providing a simple way to set and observe every flip-flop in an IC.

Behavioural modelling

- In the behavioral modeling style in VHDL, we describe the behavior of an entity using sequential statements. The primary mechanism to write a program in behavioral style is by using something called a process.

- **Process block:**

  - a Process block is a concurrent block, inside which statements are executed sequentially
  - it exists between the begin and end keywords of the architecture
  - it can use all signals and ports, and we can define variables inside them

Syntax for process is:

```
optional_label: process (optional sensitivity list)
    declarations
begin
    sequential statements
end process optional_label;
```

- Syntax for function is:

```
function function_name (parameter_list) return type is
    declarations
begin
    sequential statements
end function_name;
```

- **Variables:**
  - defined inside a process, after the process declaration before the 'begin' keyword. Only valid inside that process.
  - signals and ports, like wires in any physical system have an inherent delay, after which their values are updated. Variables do not have such a delay, and thus can be used like we use variables in Python, C++ etc.

- **Sensitivity list of a process:**
  - a process block is a concurrent block, which will be executed again and again unless told to stop, thus, we have a sensitivity list (or a wait statement) which only allows the process to fire when some condition is true
  - a sensitivity list can contain multiple inputs, outputs, signals
  - in case of a sensitivity list, the process will be fired if and only if there is any change in value of any of the values in the list

- **wait statement:** it can be used in one of four ways:
  - wait on: it is a long form version of the sensitivity list. The process will fire only if there is a change in any of the items in the statement.
  - wait until: this involves a boolean condition to be true. There is a boolean expression which is given with this statement, and when it evaluates to true, the process is fired. This is used largely to make a system sensitive to a single edge or a single level.
  - wait for: this is used to make the system wait for a fixed time. The time has to be specified. This is used more in testbenches where an input to a system needs to be applied for a fixed time, until a new one can be applied to get a reliable output.
  - wait: just tell the process to die and never fire again. Usually used after application of the last input, to formally kill the process. Every process has to compulsorily have atleast one of these wait statements else the compiler will throw an error (so that the process need not be running all the time, thereby loading the simulation software).

- **if statement:** it can be used only inside a process. It's syntax is:

```
optional_label: for parameter in range loop
    sequential statements
end loop label;
```

- **for statement:** like if statement, it can also be used only inside a process. It's syntax is:

```
if condition_1 then
    sequential statements
elsif condition2 then
    sequential statements
else
    sequential statements
end if;
```

Sequential circuits and FSMs

- In sequential systems output could depend on what are the inputs you get now, as well as what was your past output

- Between each clock positive edge, the flip-flops have a value at their output, which is called the holding output. At each instant, it gives some information. This is called state of a system. If there are a defined number of states in the system (8 in the previous case), it is called a Finite State Machine (FSM)

- **State diagram:**

  - contains all states of the systems

  - arrows represent what is the next state the system should go to when it reaches current state

  - must have all possible states of the system (exhaustive)

- **State table:** is a tabular representation of the state diagram

- **case statement:** it can be used only inside a process. Unlike if condition, cases cannot overlap with each other (hence flipflop cannot be designed using case statement). It's syntax is:

```
case input/signal/variable is
    when val0 =>
    sequential statements
    when val1 | val2 =>
    sequential statements
    when val3 =>
    sequential statements
    when others =>
    sequential statement (sort of default case)
end case;
```