Libraries Folder

This folder is used to:

- Store here all different HDL libraries, global/common and dedicated, creating this way the hierarchy of a design.
- IP cores generated with Vivado IP-wizards.

Libraries

Libraries are used in HDL to make designs clear and sound, they divide a design in smaller parts so that it's easier to get an oversight of all functionality in a design. Since libraries describe parts of a design it provides the possibility to easily simulate those parts ans signs them off before they get integrated into the whole of the design or project. When library parts are well designed and tested they are good candidates for re-use.

Instantiating a library component in a HDL design can be done in different ways. There is the classic **component instantiation** method. Here one needs to first declare the component in the declarative scope of the source code (file) where the instance is required. That usually means in the VHDL file's declarative region, but it can also be defined in packages. Then the component need to be instantiated and connected in the flow of the HDL code. Using this method means a lot of typing because a component needs to be declared first and then the same component need to be instantiated and used.
The better and preferred method is ***entity or direct instantiation***. This method is and will be used in all designs. It saves all the component declaration work and as such saves a lot of typing and creates shorted and more clear source code files. With the direct or entity instantiation the code refers at the instantiation of the component immediately to it's entity and possible library.

Component instantiation versus entity instantiation.

Component Instantiation

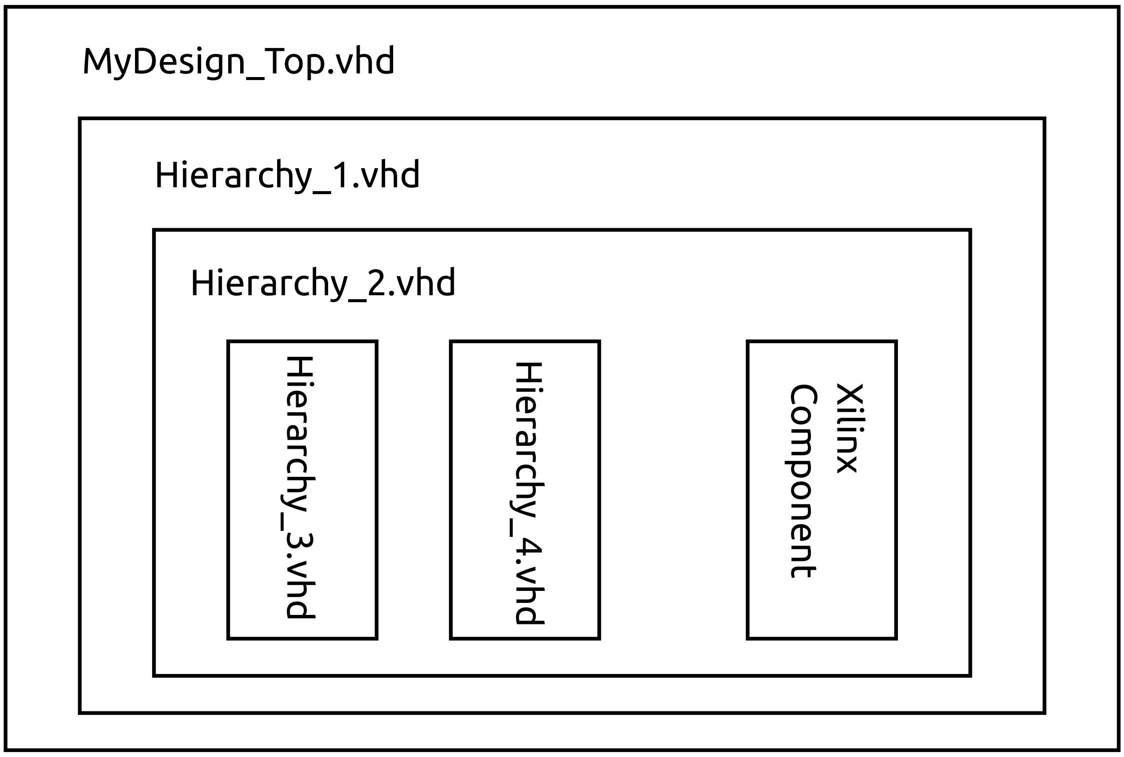
> IEEE and Xilinx Library declaration
>   declare here all standard libraries used in the design
>
> Entity declaration
>   Declare here the input and output ports and optional generics.
>
> Architecture part of the HDL code
>
> Component instantiation
>   Declare here the components that will be used in the design itself.
>
> Declaration of functions, constants, signals and attributes
>
> begin
>   Description of the functionality of the design.
>   Here the earlier declared components will be used and connected.
> end

Entity Instantiation

> IEEE and Xilinx Library declaration
>   Declaration of component libraries

> Entity declaration
>   Declare here the input and output ports and optional generics.
>
> Architecture part of the HDL code
>
> Declaration of functions, constants, signals and attributes
>
> begin
>   Description of the functionality of the design.
>   Here the earlier declared components will be used and connected.
> end

Library structure in a design



The file "MyDesign_Top.vhd" goes into the project or design /Vhdl top level directory.
Each of the "Hierarchy_*n*.vhd" files goes in its own library/Vhdl directory under the main /Libraries directory. The Xilinx component is instantiated as component in the design from the Xilin delivered HDL primitive component directories. The "Xilinx Component" can also be the top level HDL file created by the IP-Manager or a IP-Wizard.

```
/Project
    /Libraries
        /Hierarchy_1_Lib
            /Vhdl
                Hierarchy_1.vhd
        /Hierarchy_2_Lib
            /Vhdl
                Hierarchy_2.vhd
        /Hierarchy_3_Lib
            /Vhdl
                Hierarchy_3.vhd
        /Hierarchy_4_Lib
```

> /Vhdl
> Hierarchy_4.vhd
> /Vhdl
> MyDesign_Top.vhd

Find below VHDL libraries that are default created libraries. No one will stop designers to not use these names or libraries and create their own libraries with custom names. It's even so that when a design grows new libraries should be created, each of them with its own unique name.

VHDL Libraries

Common_Lib is dedicated VHDL library to store function packages. It will always be in the project even
when it is not used Each hierarchical level gets his own **_Lib** directory in this Libraries directory. Of course a hierarchy can consist out of multiple VHDL files and a **_Lib** directory can be a complete project (copied, as re-use, from somewhere else).

This way of working allows easy re-use of projects and libraries. As written above, created libraries can in
their turn be full projects. To reuse another full project as a hierarchical level in a new project, do:

- Create a **_Lib** directory in this Libraries directory.
- Where **_Lib** gets the name of the top level of the project that's going to be used.
- Copy the entire project into the newly created **_Lib**.
- From the entity of the top level of the imported project, create a instantiation component.
    - When using the Atom.io editor, install a package called VHDL Entity Converter and use that to
      automatically convert and entity to component.
- instantiate that component in the hierarchy of the new project.

IP-Cores

When using Vivado to manage all and everything of a project, all files and directories are created under a in Vivado generated project. I like to use Vivado only for what it is made, and that is place and route a design in a FPGA and create a bit stream for it. I like to keep all the rest outside a Vivado project, the reason:

- Do not create a garbage bin of your design.
- Vivado creates path of enormous length, especially when using IP cores. This is a problem on Microsoft machines.

With this approach using IP created with Vivado goes into a directory created under the /Libraries directory, keeping the length of paths in sizable lengths.
The big advantage of this is that these created IP-cores can easily be re-used in other projects.
Example: an ILA core created for a project can easily be copied, pasted and used under the Libraries directory of another project.

- Create an IP core in */Libraries*
- Create under */Libraries* a new **_Lib**. directory. Give it a meaningful name, use the name of the top level entity as easy meaningful name followed by ***_Lib***.
- Open a terminal/command window.
- Change directory (cd) to: /Libraries/*Lib.*
- Start Vivado in that directory.

- In the main Vivado window select:
    - "Manage IP"
    - "New IP Location"
    - Click [Next]
    - Enter: part, target language, Target simulator, Simulator language and for IP location click the
      browse button at the right of the entry line.
    - In the pop up window select/browse to the /Libraries/_Lib folder.
    - Press [Select]
    - Press [Finish]
- The IP project manager starts and creates the new project. Then displays the IP project manager
  window with a full choice of all Xilinx IP.
- Create the IP core.
    - Select in the right pane the wanted core. Example: Under Debug & Verification select Debug and
      then double click on: ILA
    - A core (ILA) toolbox will popup.
        - Give the core a meaningful name in Component Name
        - Configure the core as required.
        - Hit [Ok]
        - Hit [Generate]
    - The core is now generated and when finished it can be used in a design.
- The directory structure will look as:

  > /Libraries
  > /<CoreFoldername>_Lib
  > /.Xil
  > /ip_user_files
  > /managed_ip_project
  > /<Created_IP_Core_with_meaningful_name>
  > vivado.jou
  > vivado.log

- Under <Created_IP_Core_with_meaningful_name> find a .vho file that can be used to instantiate the
  created IP core in the design.

Modify an IP core in */Libraries*

If it is necessary to change something to the ILA core, do it this way:

- Open a terminal/command window.
- Change directory (cd) to: /*Libraries*/nnn_**Lib**
- Start Vivado
- In the main windows select:
    - "Manage IP"
    - "Open IP Location"
    - Select/browse in the pop-up file selector the /Libraries/_Lib folder.
    - Press [Select]

- The IP project manager starts and displays in the [Sources] window the IP.
    - Click on ''.
    - It should now display in a left pane a number of generated files.
- Right click on .
    - Select "Re-customize IP".
    - Perform the modifications needed for the design.
    - Click [OK].
    - In the popup window click on [Generate]
    - Click [OK] on the second pop-up about Out-Of-Context running.
    - Wait till the "Design Runs" window shows a 'V' in front of the IP core.
- Close Vivado
- The IP core is now re-generated and if required ready for instantiation.