How to use the Project Template

Introduction

This is about a Linux Bash script created to setup a project and/or library structure for AMD FPGA designs.
The project structure is there to give full control over a design without the need to use Vivado or Vitis graphical user interface for design management. When using the VS-Code editor with selected extensions there is no need to let Vivado or another tool manage the design. The only thing Vivado is used for is for the tasks it is build for; synthesise and implement (place & route) a design.

Provided

The script and accompanying directories and files are delivered as a zip file (CreateProject_[date].zip). The setup of the script and its directories and files is as showed below:

```
.       (This is usually the /templates directory in the users home directory)
├─ /VhdlProjects
│   ├─ /FileHeaders                    File headers, project data and legal statements for all created files.
│   ├─ /Library_Name              The library template directory
│   │   ├─ /Documents
│   │   ├─ /SimScripts
│   │   ├─ /Simulation
│   │   └─ Vhdl
│   ├─ /Project_Name              The project template directory
│   │   ├─ /Constraints
│   │   ├─ /Documents
│   │   ├─ /Libraries
│   │   │   ├─ /Common_Lib
│   │   │   ├─ /HierarchyName_Lib
│   │   │   └─ /IpCoreName_Lib
│   │   ├─ /SimScripts
│   │   ├─ /Simulation
│   │   ├─ /Vhdl
│   │   ├─ /Vivado
│   │   └─ /ZipFiles
│   │      README_FIRST.md
│   │      README_FIRST.txt
│   ├─ CreateProject.desktop         File used in Cinnamon desktops to show the tool in the main menu
│   ├─ CreateProject.sh                    The bash script for project creation, discussed below.
│   └─ ProjectTemplateUse.pdf            PDF version of this markdown document
│
```

Each above presented directory contains one or more files.
The directories and files are discussed in this document.

Remark:

- All files in this zip file are delivered as is (standard/default setup) and can be modified by the user to the needs of the user.

Installation

The tool is aimed to be used in a Linux environment because the main element of this tool is a bash script.

- Copy the provided zip file in the directory used to store templates.
  - Most of the time the user default template folder is /home//Templates
  - When storing templates somewhere else, use that folder.
  - When not having a folder for templates create one or use the default of the system.
- Unzip the contents of the zip file in the chosen *Templates* folder.

Remark:

- The CreateProject.desktop file will need to be changed when using a different templates file than the one used in the Cinnamon desktop for Ubuntu, Linux-Mint or Manjaro.
- Change the path to the script in this line:

  ```
  Exec=/home/user/Templates/VhdlProjects/CreateProject.sh
  ```

For Linux do:

This is written for Ubuntu, Linux-Mint or Manjaro Gnome and/or Cinnamon desktops. The script itself should work on all systems running “the Bourne Again SHell” (Bash). The tool is setup to bring the script as an icon in the main menu of systems running Gnome or Cinnamon desktop environments.

- Copy the *CreateProject.desktop* file into the `.local/share/applications` directory.
  - Open the file with a text editor and change the word “**user**” in:

    ```
    Exec=/home/user/Templates/CreateProject/CreateProject.sh
    ```

    YOUR user name.
  - It is also possible that a complete other directory than the default Templates directory in the home directory is used to store the script. Make sure the path in above showed line behind **Exec=** reflects the correct path to the “CreateProject.sh” file.
- Make sure both the .desktop and .sh file are executable!

Now it's possible to just double click the icon to create a new project or library with all its directories
and files. Running the *.sh* script will setup a project structure like showed below.

Remark:

- One can run the script form within a terminal by calling the script as

  ```
  . /home/<user>/Templates/CreateProject/CreateProject.sh
  ```

Project Setup

A project when setup by the script is an automatically customised copy of the directories and files in the **~/Templates/CreateProject/Project_Name** directory. It is thus fairly easy to modify and tune the project setup to the needs of the user.
A basic projects shows a directory structure as:

```
.      (Project Root Directory)
├─ Project directory
│    ├─ Constraints
│    ├─ Documents
│    ├─ Libraries
│    ├─ PetaLinux
│    ├─ Pynq
│    ├─ SimScripts
│    ├─ Simulation
│    ├─ Vhdl
│    ├─ Vivado
│    ├─ Vitis
│    ├─ ZipFiles
```

Constraints

Directory for the design constraint files. Three constraint files are available in this directory. One global constraints file for placement, pin lock and timing constraints. A file for bitstream constraints and a file for implementation constraints. A ReadMe_Constraints.pdf file provides more and detailed information about the files and even more.

Documents

This is the directory were all document that have some connection with the design are stored. Store here design propositions, data sheets, application notes, email about the project/designs and the real design documentation. For that make custom sub-directories in this directory, a possible directory layout is shown below:

- DataSheets
- Contains the data sheet and all other information of the used FPGA and data sheets of components connected to the FPGA.
- Propositions
  - Fits all documents presenting propositions on how the project can be handled and worked out. This directory will be an important source at the start of any project because then propositions are key to the next process step of the design.
- Email
  - Store here prints of emails that have to do with the project. This makes following up of the project easier is later stages.
- Figures
  - If markdown or other documents are created under the Documents directory store here the figures used in these documents.
- Other

    - All other documents relevant to the project go here.

Libraries

This directory contains by default three directories for libraries

- Common_Lib : This is a basic library that can/should contain elements used throughout, on all hierarchical levels, the whole design.
- HierarchyName_Lib : This is a library for an hierarchical component of the design. Hierarchical component are simulated separately so that when a design is finished one is sure the hierarchical elements work.
- IpCoreName_Lib : Each created IP core is dropped in it's own library. The setup of the IP core library can look as the setup of the HierarchyName_Lib library or can have the look of a complete project.
  When using Vivado [Tools] - [Create and Package New IP] flow directories and file can/will be add.

PetaLinux

To create this directory one must answer yes to a question posed by the script. The goal of this directory is to contain all Linux stuff required. The content in the directory is used from the command line (Linux terminal).

Pynq

As the PetaLinux directory this directory is only created when a script option is answer with yes. When one answers yes when the script asks if Pynq is going to be used, the script will go and copy the latest version of the Pynq tool chain into this directory. The content in the directory is used from the command line (Linux terminal).

SimScripts

Simulation of designs is supposed to be done with Vivado Xsim. Xsim is used as stand alone tool from the command line by running a bash script The other files in this directory determine what needs to be simulated and what needs to be viewed as waveform.

Simulation

The result of a simulation run ends up in this directory.
This directory contains the compiled libraries in the xsim.dir sub-directory and further contains all files that are result of the opening an saving of the Xsim waveform viewer.

Vhdl

Find here the top level of the design as HDL source, testbench and tester files. All other HDL files of a design are stored in libraries. Each hierarchical component in a design is a liobrary set in the Libraries directory.

Vivado

This directory contains one or more Vivado projects created by the Vivado project management GUI tool. The main project should have the same name as the top level VHDL file in the Vhdl directory.
Read the ReadMe_Vivado.pdf file for extra information.

Vitis

Option to select during the rub of the script.
Directory use for everything that has to do with Vitis for the project.

ZipFiles

When not storing directories and files on Github or BitBucket one can on regularly base compress the ongoing design and keep the compressed design version here as backup.

Library Setup

A library when setup is an automatically customised copy of the directories and files in the **~/Templates/CreateProject/Library_Name** directory or is in selected cases a copy of the **~/Templates/CreateProject/Project_Name** directory. It is thus fairly easy to modify and tune the library setup to the needs of the user.

Two types of libraries can be created using the script. What's called a simple library and what's called a project library. The simple library setup only has a small selection of directories of a project while the project library is a library that looks and is a real project able to stand at itself.

Simple Library setup:

```
Libraries       (The libraries directory of a project)
├─ Library directory
│    ├─ Documents
│    ├─ SimScripts
│    ├─ Simulation
│    ├─ Vhdl
```

Project directory setup:

The library is created as all other libraries in the Libraries directory of a project but the setup is that of a project. For the structure look at the project setup

Using the Linux Script

Click the icon set by *CreateProject.desktop*, chose [CreateVhdlProject] in the applications menu structure or open a terminal/command window and type:
.\Path\To\The\Project_Template\CreateProject.sh.

This is what the script will show and ask:

When the script is started, below text is displayed for a while without possibility to enter any value. The text tells what's going to happen.

```
================================================================================
 This script creates a new project or new library in an existing project for
 AMD FPGA designs. The script generates a directory structure and coding and/or
 simulation template files to be used with Vivado, Petalinux, PynQ and/or Vitis
 Things:
       - The scripyt assumes the use of VS-Code editor
       - The script and directory setup assumes the use of the Vivado simulator
         run in standalone mode.
       - The Vivado tool is solely used for synthesis and inplementation
       - PetaLinux, when used, is used in command line (terminal) mode
       - Pynq, when used, is like PetatLinux used from the command line

 Each of the created directories in a project or library has a PDF document
 with instructions and guidelines about how to use the directory, tools and
 files.
================================================================================
```

Next is a text and question about what must be created:

```
============================================================================
 For the good order:
 Single character answers are case insensitive (y/n can also be Y/N).
 All other answers are case sensitive (Author, Project , ...).
=============================================================================
 This script needs GIT! Most likely GIT is already installed, but to be sure
 it is, type \"git --version\" in a command terminal window. If something
 like \"git version 2.46.0\" is reported, GIT is installed
 If nothing is reported, GIT needs to be installed
    On Ubuntu or Linux Mint, type \"sudo apt install git\"
    On Arch Linux or Manjaro, type \"sudo pacman -S git\"
 Configure GIT
    git config --global user.name \"Your Name\"
    git config --global user.email \"youremail@example.com\"
=============================================================================
 REMARK:
    At any moment the script can be left by hitting [CTRL]-[C].
=============================================================================
 What needs to be created?
```

Answer **P** when a project need to be generated or answer **L** for a library. The letters entered to make a selection are case insensitive.
Remark:
    One can exit the script, at any moment, by hitting CTRL-C.

Assume a project has to be created, answer thus **P**.
When a library, answer **L**, is the correct answer continue reading at: **"Using the Linux Script - Library"**.

When the choice has been made to create a new project the question is asked **where** the project **should** be created.

```
======================================================================
 Is <path/of/current/directy>, the path to create the new project (y/n)?
======================================================================
```

When answering *yes*, the project will be created in the directory where the script is started.
If the answer is *no*, the next question is asked.

```
Provide the path to the directory where the new project needs to be created.
The new project is created as a sub-directory in the given path
Use the \"tab\" for auto-complete commands, files, or directories.
DO NOT USE '-' or SPACES in directory names or file names!
DO NOT END THE PATH WITH A "/"!
```

Provide the path to the directory were the new project needs to be created. DO NOT USE "-" characters in the path name and DO NOT terminate the path with a / character. As can be used on the command line, the [tab] key can be used to complete path names.

The path information can be absolute or can be relative. When relative the directory where the script is started is taken as base.

Assume the entered path is: */home//Projects*

When the provided path already exist, what mostly will be the case, next is asked.

```
 Directory, /home/<user>/Projects, already exists!
 Use (u) the given path anyway or change (c) the path (u/c)?
```

When the provide path does not exist below text is displayed, when sure about the path enter **y** to create the project directory.

When using a directory as home for all projects, the answer will be u/U because a new project must be created near all other already existing projects in the Projects directory. When the answer is c/C the previous question will be asked again and one can provide a new path to create the project.
Assume the answer is **u**
The script will confirm the directory name and path that will be used as home for the new project and will at the same time ask the name of the new to create project.

```
======================================================================
Provide the name of the new project to create!  .  .  :
======================================================================
```

Enter the name of the project to create (like: MyNewProject).
After all next questions, the project will be created as directory with all it's sub-directories and files under the previous given path.

A check on the given project name will be done to detect if the name is not already used by another project. When that is the case, previous question will be asked again.

```
Checking if the project already exist in the provided path.
```

If the project already exist, the next question is for a new project name or project path

```
Project already exist in the provided path.
Enter a new project name to create as directory under the given path
or create a whole new project path.
Enter a New Project Name (npn) or enter a New Project Path (npp)  .  .  .  .  :
```

If the project does not exist it will be created after below remark

```
The project does not exist in the provided path.
```

Questions about the used FPGA, tools and other options are asked in order to be able to customise the project directory that is going to be created.

```
=================================================================================
 Gathering Required Project Information

 Remark:
    The user of the script is responsible for the correct installation and
    environment setting of all AMD/Xilinx tools (Vivado, Vitis, Petalinx, ...)!


 Name (without extension) of the top level design file (VHDL/verilog)?  .  .   :
 Name of company/vendor creating this Project/Library?  .  .  .  .  .  .  .  . :
 What is the target FPGA family for this project?
 Use , to separate different FPGA families used in the project
 Example: Zynq 7000, Ultrascale, Ultrascale+, Zynq-Ultrasacle+  .  .  .  .  .  :
 Vivado version going to be used for the project  .  .  .  .  .  .  .  .  .  . :
 Used Simulator and its version (ex: QuestaSim_10.7)  .  .  .  .  .  .  .  .   :
 Is PetaLinux going to be used?  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   :
 Is Pynq going to be used?
    Answering yes (y) will automatically clone the latest version
    of PYNQ from the Xilinx Pynq GitHub page.
    Find the official PYNQ webpage here http://www.pynq.io/
       Find the PYNQ documentation here http://pynq.readthedocs.io/  .  .  .  .:
 Is Vitis going to be used?  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  . .  :
 Is the project structure created for the person running this script? (y/n)  . :
```

- When answering **Y**, the name (Linux user name) is displayed and the question asked if that name really needs to be used. Answer again **Y** or answer **N** and enter an new name.
- When answering **N**, enter a name.

```
 Is the VS-Code editor used as development tool for the project? (y/n)  .  .   :
 Append or add the project to a editor workspace and/or project? (y/n)  .  .   :
```

- If the question about the use of the VS-Code editor is answer with no, no extra questions will be asked.
- If the question is answered with **Y**, the question about appending to a workspace or creating an new workspace is asked. Checks are done to determine if workspaces exist or not.

```
 Append the new project to the Project Manager project list? (y/n)  .  .  .  . :
==========================================================================
```

Project manager is a extension for the VS-Code editor and is mostly used when the design uses GitHub as repository.

Before creating the project structure on disk, a summary of all given answers is showed.
Answer *y/Y* when all data is correct and let the script generate the project or answer *n/N* wen some or all data is wrong. In this case all questions will be asked again.
The summay looks like:

```
 The Project Manager extension for VS-Code is installed.
 Checking if there is already a projects.json file
 if file doesn't exist, one is automatically created.
 There is already a projects.json file present

================================================================================
====
 Summary of gathered project information before creating the project:




 This script is stored in:
/home/marc/Templates/CreateProject
 The script is run from:
/home/marc/Desktop/Pynq/Projects
     Creating a new project in:
     The new project is called:                                        ddd
     The full project path is then:                                    /ddd
 The FPGA(s) used in this project are:                                 Zynq 7000

 The top level VHDL file is:                                           eee
 The Vivado version used is:                                           2024.2
 The Simulator and version used is:                                    Xsim 2024.2

 If Petalinux is going to be used, a directory for the OS is created y
 If Pynq is going to be used, a directory for the tool is created    n
 If Vitis is going to be used, a directory for the tool is created   n
 The project or design author is:                                      Marc
 The company/vendor creating the project or Library is:                fff
 Today is:                                                             22 nov 2024
 VS-Code editor used?                                                  y
     Workspace file is put in:
     Workspace file name:                                              ddd.code-
workspace
     Add to Project Manager                                            y
     A projects.json file is available or created

================================================================================
====
 Is the displayed information correct? (y/n) :
```

If the above information is correct answer **y** and the project will be created. When answering **n** the process of project information gathering is started again.
The final output of the script is this:

```
 Create the project.
 Pynq is not going to be used.
 Vitis is not going to be used.
 Add file headers to the basic template files.
 Name the files.
 Replace the keywords in all files of the created project.
 Create a new .code-workspace file in the root of the project directory
 Add project to a existing projects.json file
 Create .mdf project file.
================================================================================
 Project <ProjectName> is created in <Project/Path>
================================================================================
```

When hitting [Enter] the command/terminal window will close.
The project is now created and all files are in place, ready to be used.

Using the Linux Script - Library

A library is normally created in the */Libraries* directory of an existing project.
Using the script to do this is explained here.

Click the icon set by *CreateProject.desktop*, chose [CreateVhdlProject] in the applications menu structure or open a terminal/command window and type:
.\Path\To\The\Project_Template\CreateProject.sh.
The script will run in a terminal/command window and display at the start the same text as is displayed when a project is created.
When the question whether a project, **p** or a library **l** must be create is asked. Answer **L**

The script now asks for the path where to create the new library

```
=============================================================================
 Libraries reside in the project's 'libraries' directory and can be simple
 set of sub-directories for HDL and simulation or can take the format of a
 complete project
 Provide the path to the root directory of a project and indicate the type of
 library required:
    project (p) : Library is setup as if it's a project
    simple (s)  : Library contains only a basic set of directories.

 Use the "tab" for auto-complete commands, files, or directories.
 DO NOT USE '-' or SPACES in directory names or file names!
 DO NOT END THE PATH WITH A " / "!
=============================================================================
 Path to project were library needs created  .  .  .  .  .  .  .  .:
```

Libraries are stored in the Libraries directory of a project. The script will check if a Libraries directory is available in the given path. When there is a Libraries directory, that directory will be used as home for the new library. When where is no Libraries directory, one will be created.

If a library needs to be created in a project `MyNewProject` residing in a directory `/home/<user>/Projects` the answer to above question would be `/home/<user>/Projects/MyNewProject`.

Now the question i,s asked what kind of library needs to be created. A simple library made from a restricted set of directories or a project library figuring as a real project.

```
Library type: (p/s)?  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .    :
```

Let's answer **P**

The library is going to be generated in a project setup created by the same script as here used for creating libraries, thus there is a Libraries directory available. The tool check confirms this

```
 A "Libraries" directory is available in given project path.
 A library can be created in: /home/marc/Desktop/Pynq/Projects/ddd/Libraries

==============================================================================
 What is the name of the library that needs to be created?
 REMARK: The entered name will automatically be extended with '_Lib'
==============================================================================
```

Enter the name of the library that needs to be created.
The script checks if a library with the given name already exist, if it is a new name will be asked else the library directory will be created.

```
 A library, <lbrry>_Lib, will be created in
/home/marc/Desktop/Pynq/Projects/ddd/Libraries
 Checking if the library already exist in the given project/Libraries path.
 A new library, qqq_Lib, will created in
/home/marc/Desktop/Pynq/Projects/ddd/Libraries
=================================================================================
 There is a .mdf available.
=================================================================================
=================================================================================
 Create the library as a project:
```

Since the library will have the format of a project, the questions about PetaLinux, Pynq and Vitis need to be asked

```
 Is PetaLinux going to be used?  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .   :
 Is Pynq going to be used?
     Answering yes (y) will automatically clone the latest version
     of PYNQ from the Xilinx Pynq GitHub page.
     Find the official PYNQ webpage here http://www.pynq.io/
     Find the PYNQ documentation here http://pynq.readthedocs.io/  .  .  .  .  :
 Is Vitis going to be used?  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  . .  :
=================================================================================
```

After answering these questions the library, as a project, is generated.

```
Create the project.
 Petalinux is not going to be used.
 Pynq is not going to be used.
 Vitis is not going to be used.
 Add file headers to the basic template files.
 Name the files.
 Replace the keywords in all files of the created project.
=================================================================================
 Library <...>_Lib is created in /path/project_name/Libraries

=================================================================================
Press enter to continue
```

Project and Library Template Setup.

The script file copies one of the directories, Library_name or Project_name, including sub-directories and files to a new given, in the script file, directory. The script file then renames the whole directory and file tree and changes keywords in the files so that headers and other settings reflect the correct names, dates, used tools and legal headers.
If wanted one can modify the setup of the *Project_Name* and/or *Library_Name* directory, respecting the names of directories and files, to reflect a different project setup.

- FileHeaders
  This directory contains the text of the legal headers visible in VHDL and other text files.
  One can adjust, adapt these files to his/her needs.
- Project_Name
  Root directory of the project.
  This will be created in the location given at the Project path question.
  This will get the requested Project Name name.
  The script will create all sub-directories and necessary files.
  - Constraints
    Shows three base XDC files; one for bit stream constraints one for implementation constraints and one for global design constraints (timing, pin lock, ...)
  - Documents
    This folder is the home of all documentation of the project.
    The *Figures* sub-directory hold figures used in the documentation Markdown text file.
    Read also the pdf file in the directory
  - Libraries
    Each hierarchical level in the design is represented as a library.
    This allows easy re-use of previous or parts of previous projects. A Library directory can thus contain a full sub-project, used in this project as instantiated component in the top level or as instantiated component in a sub-level. Or the Library directory can contain a simple library of simulated HDL and use that as component. The third option is that the Library contains a user generated IP block. IP cores are created as libraries because they act in the design is hierarchical levels. As with normal hierarchical levels, this setup makes it easy to re-use previous created IP cores. Read the **ReadMe_Libraries.pdf** on how to create IP-cores and get them in a library
     The minimum a library must contain is a VHDL or verilog source code file.

- SimScripts
  Functional simulation uses VHDL test-bench and tester files in the VhdlL directory.
  All simulation is pure VHDL based.

  Timing simulation uses TCL scripts to run. This is done because for a timing simulation the design is implemented a FPGA and the FPGA is simulated as some kind of back box with inputs and outputs.
  Read the **ReadMe_Simulation.pdf** to grasp how simulations can be run from here.
  The *transcript* file is kind of a simulator log file.
  The *Vivado_GenSimFiles.tcl* is a TCL file that need to be run from the Vivado-TCL command line. It generates the necessary files for timing simulation. The generated files are dropped in the *Simulation* directory.
- Simulation
  Directory used by the simulator for functional and timing simulations.
  The .do and Vivado_GenSimFiles create files that are stored in this directory.
- Vhdl
  This directory holds the top level or top levels of the design and project.
  The directory also holds the functional simulation test-bench and tester files.
- Vivado
  This is the directory holding the Vivado implementation project of the project.
  Read the **ReadMe_Vivado.pdf** document on how to proceed.
- ZipFiles
  Any zip files having some kind of band/bond with the project go here.
  Example: as version of the project to be kept/archived can be stored here.