

Lab 1: Introductory session on Chisel:

Chisel is a powerful hardware description language (HDL) that is gaining popularity in the field of digital design and hardware engineering. It is used to design and describe digital circuits and systems at various levels of abstraction. Chisel stands out because it allows designers to write hardware descriptions using a high-level, Scala-based programming language, which provides several advantages over traditional HDLs like Verilog and VHDL. In this introduction to Chisel-based digital design, we'll explore the key concepts and benefits of Chisel.

1. High-Level Abstraction: Chisel allows designers to describe hardware using a high-level programming language, Scala. This means you can leverage the power of a modern, expressive language to specify digital circuits, making your designs more concise and readable compared to traditional HDLs.

2. Productivity and Maintainability: Writing hardware designs in Chisel often results in shorter and more modular code, which can significantly improve productivity and make it easier to maintain and debug your designs. The use of higher-level constructs can reduce the chances of introducing errors.

3. Re-usability: Chisel promotes the creation of reusable hardware components and libraries. You can define your custom modules, parameterize them, and reuse them across different projects, which can save time and promote good design practices.

4. Type Safety: Chisel takes advantage of the type system in Scala, which helps catch many design errors at compile-time rather than runtime. This can prevent costly mistakes in hardware design.

5. FPGA and ASIC Support: Chisel can target both field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs). This flexibility allows you to prototype your designs on FPGAs and then move to ASIC production if needed.

6. Integration with Verilog: Chisel can generate Verilog code from your Chisel designs, making it compatible with existing EDA (Electronic Design Automation) tools and allowing you to leverage the vast ecosystem of Verilog-based IP cores.

7. Community and Resources: Chisel has a growing and active community, which means you can find documentation, tutorials, and support to help you get started and overcome challenges as you learn and use Chisel for digital design.

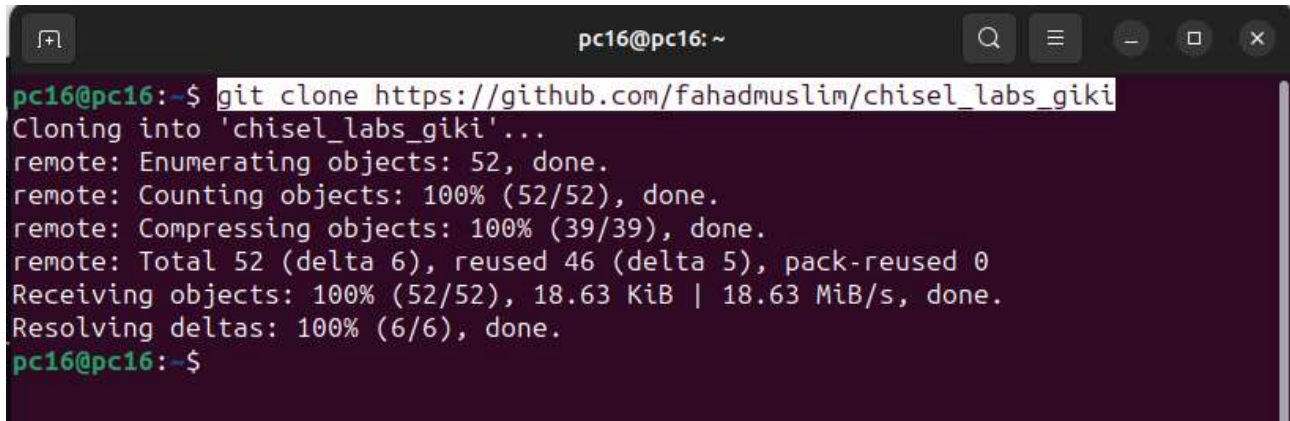
8. Chisel Ecosystem: Alongside Chisel, there are several tools and libraries that enhance the Chisel ecosystem, such as Firrtl (an intermediate representation for hardware), ChiselTest (for unit testing Chisel designs), and Chisel3 to Chisel4 migration tools.

9. Research and Innovation: Chisel is continually evolving, with ongoing research and development efforts, which means it can incorporate the latest design methodologies and best practices.

In summary, Chisel-based digital design offers a modern, high-level, and expressive approach to designing digital circuits and systems. Its focus on productivity, re-usability, type safety, and compatibility with existing tools makes it an attractive choice for hardware engineers and researchers looking to tackle complex digital design challenges. As you delve deeper into Chisel, you'll discover its capabilities and the benefits it brings to the world of digital design.

Step 1: Cloning Chisel Labs in Ubuntu:

Simply type git clone followed by the url to chisel labs. The url is as follows:
https://github.com/fahadmuslim/chisel_labs_giki

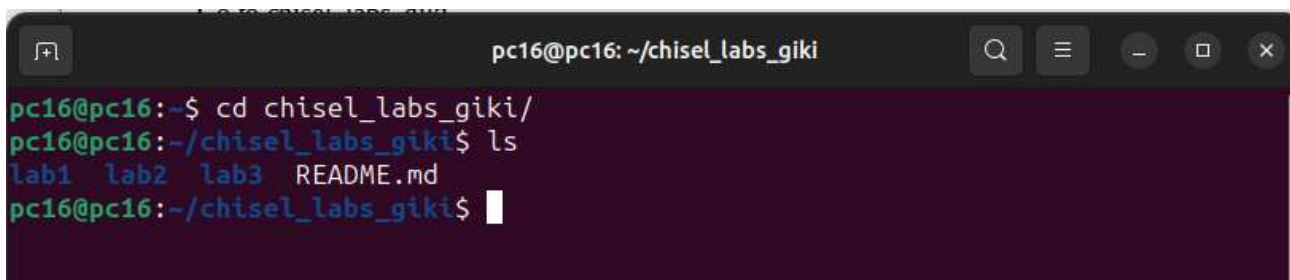


```
pc16@pc16: ~  
pc16@pc16:~$ git clone https://github.com/fahadmuslim/chisel_labs_giki  
Cloning into 'chisel_labs_giki'...  
remote: Enumerating objects: 52, done.  
remote: Counting objects: 100% (52/52), done.  
remote: Compressing objects: 100% (39/39), done.  
remote: Total 52 (delta 6), reused 46 (delta 5), pack-reused 0  
Receiving objects: 100% (52/52), 18.63 KiB | 18.63 MiB/s, done.  
Resolving deltas: 100% (6/6), done.  
pc16@pc16:~$
```

if git is not installed then, the install it on your machine using the the given link.
<https://www.digitalocean.com/community/tutorials/how-to-install-git-on-ubuntu-20-04>;

Step 2:

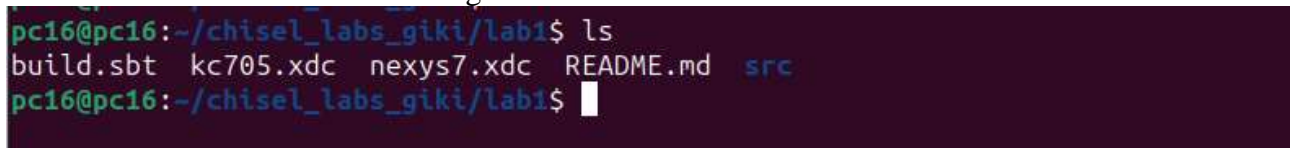
Go in to the **chisel_labs_giki** and then list all its contents using **ls** command.



```
pc16@pc16: ~/chisel_labs_giki  
pc16@pc16:~$ cd chisel_labs_giki/  
pc16@pc16:~/chisel_labs_giki$ ls  
lab1 lab2 lab3 README.md  
pc16@pc16:~/chisel_labs_giki$
```

Step 3:

Go to **lab1** and list the contents using **ls** command to see all the concerned files.



```
pc16@pc16:~/chisel_labs_giki/lab1$ ls  
build.sbt kc705.xdc nexys7.xdc README.md src  
pc16@pc16:~/chisel_labs_giki/lab1$
```

Build Process and Testing:

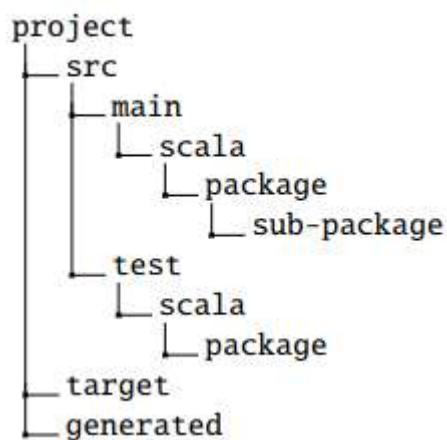
To get started with more interesting Chisel code we first need to learn how to compile Chisel programs, how to generate Verilog code for execution in an FPGA, and how to write tests for debugging and to verify that our circuits are correct. Chisel is written in Scala, so any build process that supports Scala is possible with a Chisel project. One popular build tool for Scala is **sbt**, which stands for the “Scala interactive build tool”. Besides driving the build and test process, **sbt** also downloads the correct version of Scala and the Chisel libraries.

Building your Project with sbt:

The Scala library that represents Chisel and the Chisel tester are automatically downloaded during the build process from a Maven repository. The libraries are referenced by **build.sbt**. It is possible to configure **build.sbt** with latest release to always use the most recent version of Chisel. However, this means that on each build, the version is looked up from the Maven repository. This lookup needs an Internet connection for the build to succeed. It is better to use a dedicated version of Chisel and all other Scala libraries in your **build.sbt**. Sometimes, it is good to be able to write hardware code and test it without an Internet connection.

Source Organization:

The figure below shows the organization of the source tree of a typical Chisel project. The root of the project is the project home, which contains **build.sbt**. It may also include a **Makefile** for the build process, a **README**, and a **LICENSE** file. Folder **src** contains all source code. From there it is split between **main**, which contains the hardware sources and **test** containing testers. The next folder in both cases is **scala**, as Chisel is based on Scala. If you want to include Java code, which may be useful for hardware generators, you would add a **java** folder. Chisel inherits from Scala, which itself inherits from Java, the organization of source in packages. Packages organize your Chisel code into namespaces. Packages can also contain sub-packages. The folder **target** contains the class files and other generated files.



How to execute Chisel using sbt to generate a Verilog code:

1. Go to Lab 1 and open the terminal with the following path.

```
pc16@pc16:~/chisel_labs_giki/lab1$
```

2. Type **ls**

```
pc16@pc16:~/chisel_labs_giki/lab1$ ls
build.sbt      kc705.xdc    project      src
build.sbt.save nexys7.xdc   README.md    target
pc16@pc16:~/chisel_labs_giki/lab1$
```

3. Type **sbt run** and it will generate all the necessary vivado files including the verilog code.

```
pc16@pc16:~/chisel_labs_giki/lab1$ sbt run
[info] welcome to sbt 1.9.6 (Private Build Java 17.0.8.1)
[info] loading project definition from /home/pc16/chisel_labs_giki/lab1/project
[info] loading settings for project lab1 from build.sbt ...
[info] set current project to lab1 (in build file:/home/pc16/chisel_labs_giki/lab1/)
[info] compiling 1 Scala source to /home/pc16/chisel_labs_giki/lab1/target/scala-2.13/classes ...
[info] running HelloMain
Hello World, I will now generate the Verilog file!
[success] Total time: 4 s, completed Sep 21, 2023, 10:46:50 AM
pc16@pc16:~/chisel_labs_giki/lab1$
```

4. If you want to save the files in user defined folder named **generated** then the following example must be followed i.e.

```
object <name> extends App {
  emitverilog (new <name> (), Array (" --target-dir" "generated"))
}
```

5. You can view the **<name>.v** and supporting files using the **ls** command as follows.

```
pc16@pc16:~/chisel_labs_giki/lab1$ ls
build.sbt      Hello.anno.json  Hello.v      nexys7.xdc  README.md  target
build.sbt.save Hello.fir        kc705.xdc   project     src
pc16@pc16:~/chisel_labs_giki/lab1$
```

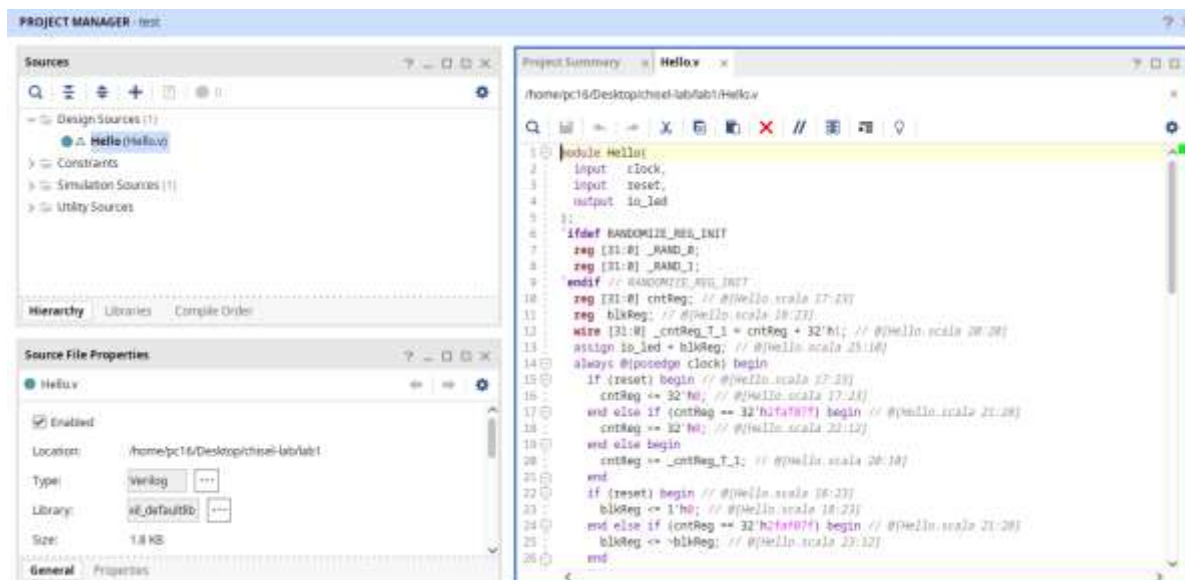
6. Open Vivado in terminal

```
pc16@pc16:~/Desktop/chisel-lab/lab1$ vivado

***** Vivado v2022.1 (64-bit)
**** SW Build 3526262 on Mon Apr 18 15:47:01 MDT 2022
**** IP Build 3524634 on Mon Apr 18 20:55:01 MDT 2022
** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.

start_gui
```

7. Create a new project and add the verilog file **<name>.v** to source files.



8. You can search the following link to see the step by step process of running and building the project in Vivado. Alternately the steps are also documented in the **Readme** file given in lab1.

https://diligent.com/reference/vivado/getting_started/2018.2