# Setup

We will be using the following tools for these labs:

* Git revision control
* UNIX make utility
* Vivado 2014.2

Following are instructions to install and set up Git and make, and to configure Vivado to run from the Windows command shell.

## Install GitHub for Windows

Git will be used as the revision control tool for these labs. It is free and one of the most widely-used tools and GitHub also serves as the revision control platform for the Vivado Tcl Store.

### GitHub Info and Installation

See the Windows GitHub introduction page here:

[https://windows.GitHub.com](https://windows.github.com)

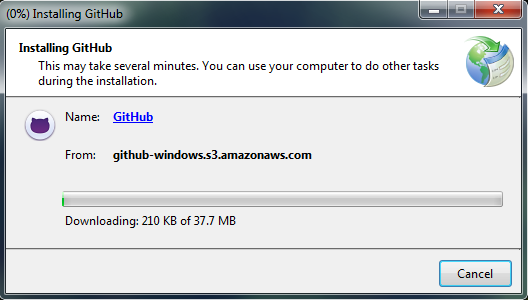
From there, click on

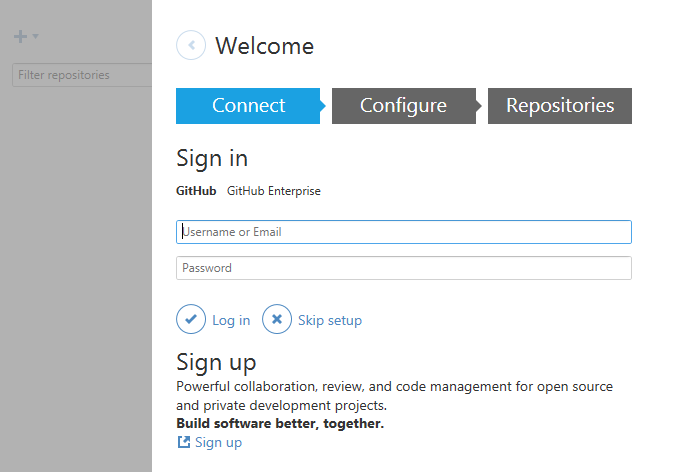


to download the setup program, or go to this URL:

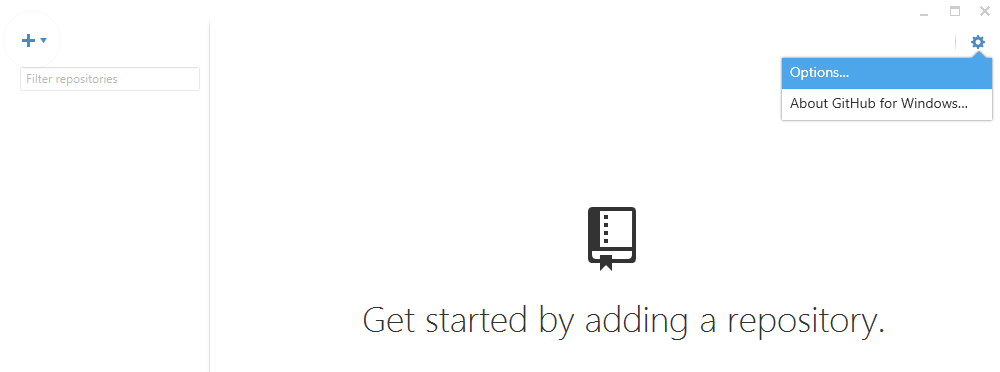
[https://GitHub-windows.s3.amazonaws.com/GitHubSetup.exe](https://github-windows.s3.amazonaws.com/GitHubSetup.exe)

Once GitHubSetup.exe is downloaded, run it to install GitHub.



Once the installation is complete you will see the Welcome page. Here you can **Log in** if you already have a GitHub account, or click the **Sign up** link at the bottom to create a new account. Or you may just continue without logging in by clicking on **Skip setup**. 

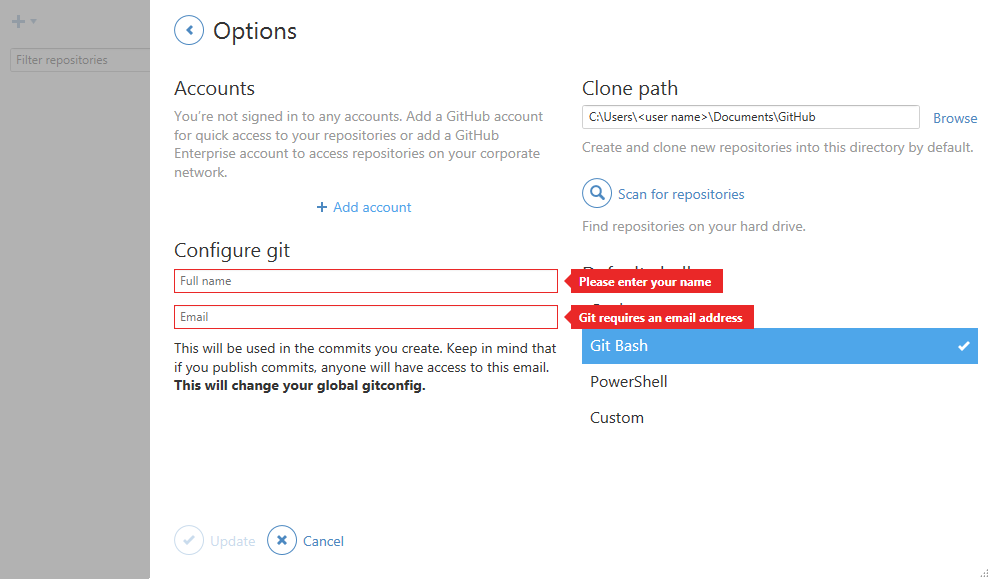
The next page shows a start page where no repositories have been set up, which should be typical for most. We will skip adding repositories for now and instead right-click on the blue Settings icon in the upper right corner and choose **Options…**



Here we will configure the following:

1. In the lower right, choose Git Bash as the default shell instead of PowerShell.
2. Enter your name and email which Git uses to track changes.

Once finished, click on the Update button at the bottom to save changes.



Now you may exit the GitHub GUI. We will be using the Git Shell for revision control.



To test Git, double-click the Git Shell icon to launch the Git Shell. It should open a command shell with a bash prompt ($). Run git status:

$ git status

# On branch master

#

# Initial commit

#

nothing to commit (create/copy files and use "git add" to track)

If you encounter errors or other dubious output please ask your instructor for assistance.

## Lab Files

Here are brief descriptions of the lab file folders:

* doc : lab instructions and presentation
* lab\_files : contains the initial version of files for each lab
* lab\_solutions: contains the final versions of the files for each lab

Please do not jump directly to the lab\_solutions without first trying the labs. Notify your instructor if any lab instructions are unclear.

# Lab 1 : Getting started with a basic RTL project

In this lab we will start with a very basic set of files and introduce you the basic processes involved in using Vivado with revision control. This includes creating:

1. Vivado run scripts to create and run projects
2. A Makefile that calls the run scripts to build projects
3. A Git repository that keeps files under revision control

## Working in shells

We will use two different command shells for the labs:

* Windows command shell - to run Vivado and make
* Git bash shell - for everything else

Upon opening a Windows command shell, use the **env.bat** script to set up Vivado to run in the shell:

call <path to scripts>\env.bat

This also adds the MinGW UNIX-like commands to your path so you can run the **make** utility to call Makefiles in the Windows shell. You can also run other common UNIX commands such ls, cp, rm, mv, and the UNIX find command within the Windows command shell.

Test the make command. In the command shell, run **make test**. It should fail with a message:



If you get a different message or experience some other difficulty, ask your instructor for assistance.

To launch the Git shell, double-click the icon as described earlier during Git installation:



The Git shell functions just like a bash shell and is pre-configured to run git commands. On Windows the pathname to the **C:\** drive is **/c/**, so for example the folder **C:\tsc14** is accessed using **/c/tsc14**.

## Setting up the working directory

For these labs, we will start with a few files in a simple directory structure and add to it as the design grows with each successive lab. In the lab1 directory you’ll find a snapshot of the working area:

* hdl
  + threeFlop
    - threeFlop.v
* xdc
  + top.xdc
* scripts
  + setup.tcl
  + Makefile
  + env.bat
  + utils.tcl

Copy these files to a new, empty directory for example **C:/tsc14.** We will call this the root folder or root directory. Inside root we will place the Git repository and a work folder to hold Vivado results. Create a directory **work** inside root.

When working on files in this lab, it is recommended to keep script files in the **script** directory and test them by running vivado or make in the **work** directory. To do this place the Makefile in the work directory for editing and test by calling **make** on the command line. When finished the Makefile can be checked in to the scripts directory.

## Lab Procedure

1. Complete the Tcl script named **setup.tcl** to recreate a Vivado project for the **threeFlop** design. Make sure:
   1. the touch proc is called last to create a Makefile target - This tells **make** that the setup script ran successfully. If setup fails then no target file is generated.
   2. the sources are referenced remotely, not added to the project
   3. when testing the script, run Vivado in the **work** directory
2. Create a new Tcl script called **compile.tcl.**
   1. It should open the threeFlop design and generate a bitstream.
   2. Use the touch proc to create a target similar to the target file used for the setup target.
3. Complete the Makefile to build a bitstream using setup and compile targets.
   1. The setup target has been completed as an example.
   2. Add a target **compile** that calls compile.tcl to generate the bitstream.
   3. Add a target **clean** that removes all the files in the current directory which is expected to be the **work** directory.
      1. Note that UNIX commands are used to remove files.
      2. Pay special attention to the Makefile target files.
4. Check the files into the Git repository.
   1. Create the Git repository in the root folder using the **git init** command.
   2. Check the status using the **git status** command. It should reflect a newly created repository.
   3. Check in the hdl, scripts, and xdc directories, but not work. The commands to use are:
      1. **git add <filename(s) or directory name(s)>**  to stage files for a commit.
      2. **git commit -m “Comments on the checkin”**

It is good to run git status after each step to ensure the project state is as expected.

1. Once all has been checked in successfully, clean the work directory.

## Conclusion

This first lab has taken you through the complete process of:

1. Creating Tcl scripts to create and run Vivado projects.
2. Preparing and calling a Makefile to run those scripts to build projects.
3. Creating and using a Git repository to keep recommended files under revision control.

We will build on the concepts learned in this lab to learn how to handle different types of design data under revision control.

# Lab 2 Using Revision Control with Managed IP

This lab covers the process of generating a simple managed IP AXI IIC and placing it under revision control. The lab objectives include:

* Creating a Managed IP project for AXI IIC.
* Create a script to fully generate the IP with OOC flow.
* Update the Git repository with the IP products and updated scripts into revision control.
* Iterate and change an IP customization option and update the Git repository as needed.

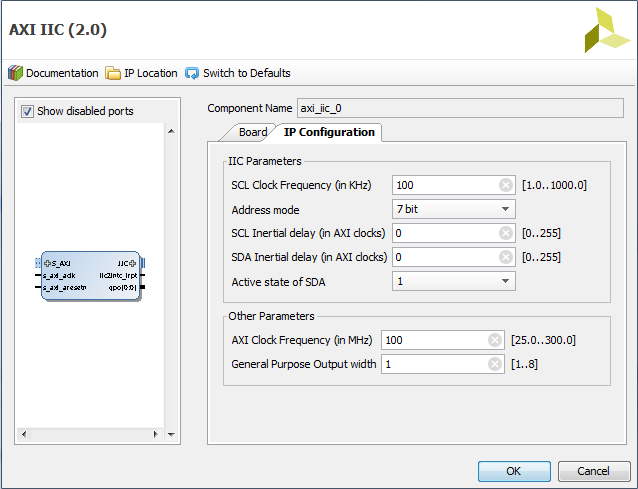
## Lab files

The lab files contain script files to get started:

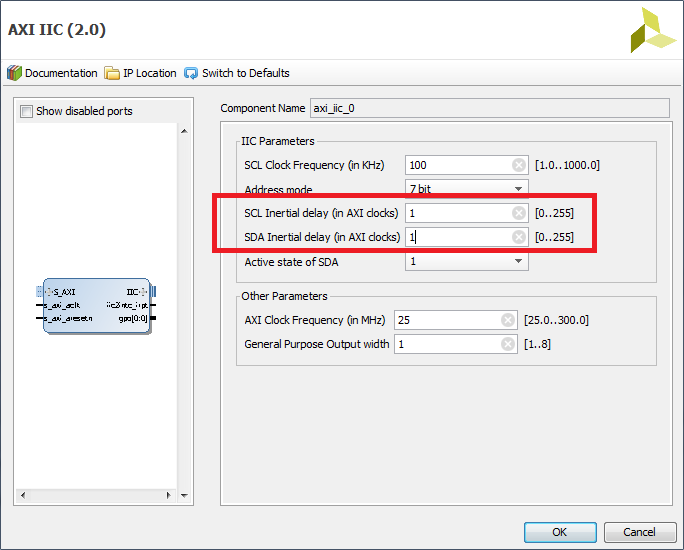
* env.bat, env.sh, utils.tcl: carried over from the previous lab
* Makefile: This is a template file where you will add the rule to make the IP.
* ip.tcl: This is a template file where you will add the Tcl required to build the IP from scratch.

## Lab Procedure

1. Create a new directory called **ip** in the root directory. This will be used to store the IP output products.
2. Create the AXI IIC IP in the work directory.
   1. Target part is xc7z020clg484-1.
   2. The target language is Verilog.
   3. Use the default settings for the IP including the name axi\_iic\_0



1. Complete the **ip.tcl** Tcl script that recreates the IP from scratch, all the way to output product generation.
   1. Avoid using absolute paths so that the IP can be recreated in a different location in the future.
   2. Make sure the utils.tcl script is sourced and use the touch proc to create a file **.ip.done** that can be used for a Makefile target similar to **compile** and **setup**.
   3. Since ip.tcl may be called in a sequence with other scripts, remember to wait on a run before creating .ip.done.
2. Add the IP to the Makefile
   1. Add the new target called **ip** that regenerates the axi\_iic\_0 output products from scratch.
   2. Add the rule to create the ip target. You can use the compile and setup targets as examples but note they are commented-out for this lab.
3. Test the Makefile by making the **clean** target followed by **ip**.
4. Copy the IP output products into the new directory inside the root directory called **ip**. The ip directory should contain the axi\_iic\_0 directory and everything below it.
5. Check in the IP and the changed scripts:
   1. Use **git status** to check the project status.
   2. Use **git add** to add the ip directory contents and changed files.
   3. Use **git status** to check the project status again.
   4. Use **git commi**t to check in the files.
6. Once the IP is under revision control, make a change. Change these options:



These are the IP properties:

* CONFIG.C\_SCL\_INERTIAL\_DELAY
* CONFIG.C\_SDA\_INERTIAL\_DELAY

1. Then regenerate the IP, update scripts, and check in files as needed. You may need to clean the work directory first.
2. Once all has been checked in successfully, clean the work directory.

## Conclusion

In this lab you have covered how to manage IP under revision control. You should now be comfortable performing the following:

* Creating and modifying Tcl scripts to fully generate the IP from scratch.
* Using a Makefile to build IP.
* Managing IP files under revision control with Git,

# Lab 3: Revision Control of a Block Design

This lab focuses on block designs from IP Integrator. In terms of revision control, block designs are somewhat similar to IP as they have associated output products. Block design management may also include recreating the block design itself from an empty block design. In this lab you will cover:

1. Use of write\_bd\_tcl to generate a Tcl script to recreate a block design.
2. Creating a script to generate the block design output products.
3. Using a Makefile to build a block design output products from scratch.
4. Checking in the block design under revision control.

## Lab files

The lab files contain the following files to get started:

* scripts:
  + env.bat, env.sh, utils.tcl: carried over from the previous lab
  + Makefile: This is a template file where you will add the rule to make the block design.
  + bd\_gen.tcl: This is a template file where you will add the Tcl required to build the block design from scratch.
* zynq\_bd\_project: contains a project with the block design used for the lab

This lab is a bit more complex than previous labs, so we will review the revision control scenario involved. The original block design is a local source in a project in zynq\_bd\_project.xpr. The block design is packaged and used in other designs. The sequence of updating the block design is:

* The block design is modified in IP Integrator in zynq\_bd\_project.xpr.
* The block design is saved and packaged in a source directory containing block designs. Recall the recommendation for block designs is to check in the directory containing the .bd file: that directory and all files and directories underneath.
* Other designs instantiate the block design which implies that those designs depend on the corresponding .bd file.
* When the .bd file changes, the entire block design package must be regenerated from scratch and checked back into the bd directory.

## Lab Procedure

1. Copy zynq\_bd\_project into the work directory and open the project from work. The block design should look similar to:



1. Use write\_bd\_tcl to write a script that creates the block design. Name the file **bd.tcl**.
2. Create a Tcl script **bd\_gen.tcl** that generates the block design output, similar to an IP. Some hints:
   1. Check what Tcl commands are issued when launching **Generate Block Design**.
   2. Review the IP generation script ip.tcl which uses a similar process to generate the IP.
   3. Include the creation of a Makefile target when finished.
3. Next update the Makefile.
   1. Be cautious of choosing the appropriate dependencies. Assume the block design project resides in **zynq\_bd\_project** as opposed to the working area **work**.
   2. Include a target bd\_gen that results in the entire block design being generated from scratch.
4. Recall the recommendation of files to check in for block designs, the block design directory and everything underneath it. Check in the recommended files to the bd directory.
5. Once all has been checked in successfully, clean the work directory.

## Summary

This lab demonstrates how IP Integrator Block Designs can be managed using revision control. Some key points to remember:

* A block design .bd file can be generated using the write\_bd\_tcl command.
* The generate\_target command generates block design output products from a .bd file, similar to Managed IP.

# Packaged Custom RTL IP with IPI

# System Generator Projects

# HLS

# Other - TBD

## DCP

## EDK/XPS

# Appendix A: Solutions