Using Git with Vivado

University of Arizona Fall 2020 ECE 274A Digital Logic

Mitchell Dzurick

8/24/2020

Contents

1 Overview **3**

2 Intro to Git and Bash **3**

2.1 What is Git? 3

2.2 What is Bash? 3

3 Using Git with Windows **4**

3.1 Git Bash 4

3.1.1 Installation 4

3.1.2 Using Git Bash 5

4 Creating your own Github Repository **13**

5 Adding files in Vivado **15**

List of Figures

1 Command Flow 4

2 Git Bash Default Screen 5

3 Printing CWD in Git Bash 5

4 Changing Directory to Documents 5

5 viewing the example Github repo 6

6 Vivado 2019.2.1 Default Screen 7

7 Naming Example Project 7

8 Adding Sources Page 8

9 Adding Example Code into Vivado 8

10 Vivado Project Manager 9

11 Running Simulation of Example Project 10

12 Modifying Example Source Code 10

13 Viewing Modified Source Code in Git 11

14 Creating a new Github Repo 13
15 adding Example Code to Personal Repo and Pushing 14
16 Adding Sources in Verilog 15
17 Create Source File Screen 16
18 Selecting Target Location 16
19 Defining Module for New File 17
20 Checking For The New File in Git 17

1 Overview

This guide will provide supplementary information for ECE 274A Lab section. This guide will be extremely useful for managing and storing the source code of your project. This guide will assume no prior knowledge of git or bash is known. The two major concepts that this guide will explain are the following:

1. basic git usage
2. basic bash usage

This guide will most likely use language that you may not be familiar with. Most things are explained thoroughly but if you are not exactly sure what a word means you are encouraged to look it up or ask on Piazza!

NOTE: This is not information you will be tested on, but rather information that will be helpful in this class and your future career as an engineer.

2 Intro to Git and Bash

2.1 What is Git?

Git is a very common **VCS (Version Control Software)** that you will see in industry. Git has an emphasis on speed, data integrity and distributed workloads. All VCS systems generally have a similar goal of tracking changes in files and storing data on a remote server.

You can learn all about git through their official website https://git-scm.com/. The book ”Pro Git” is also completely free to view through your browser https://git-scm.com/book/en/v2.

2.2 What is Bash?

Bash is an acronym that stands for Bourne Again SHell. Bash is the unix shell that GNU Project maintains. A unix shell is simply just a command-line interpreter that sends commands directly to the operating system. You can think of Bash simply as your terminal.

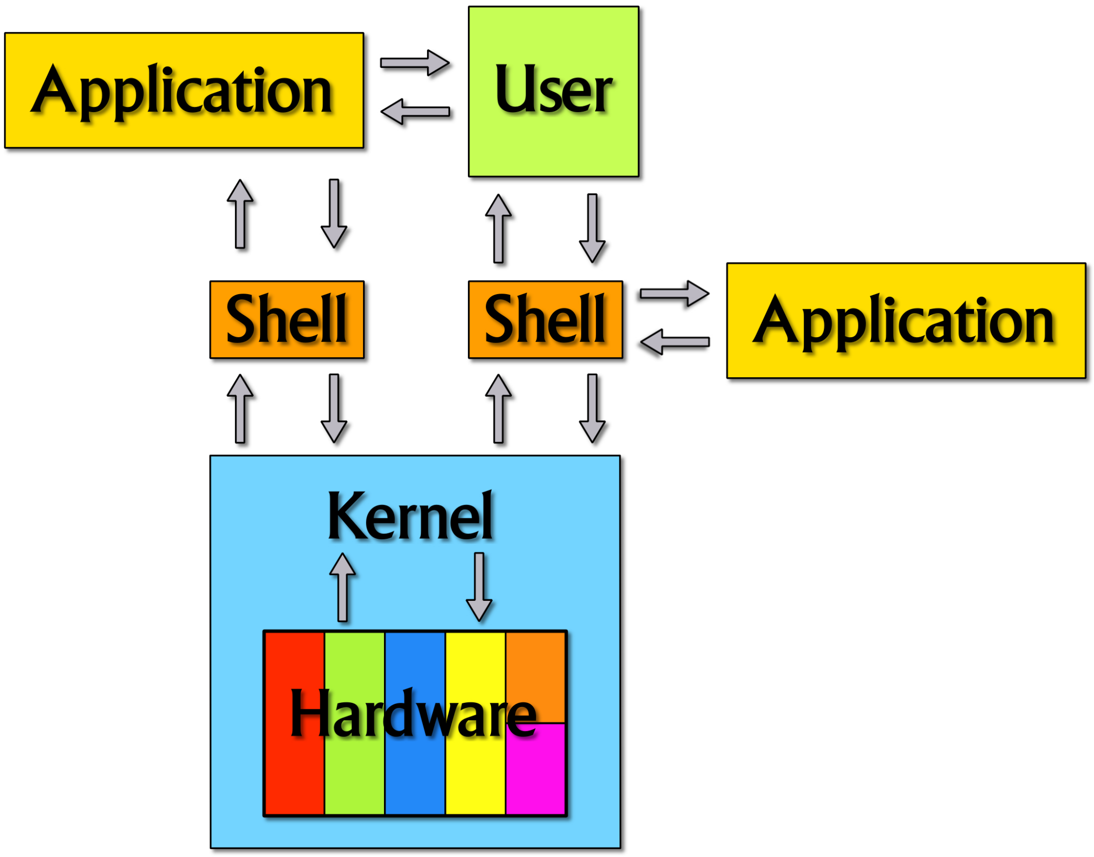


Figure 1: Command Flow

Figure 1 is a diagram provided from opensuse at https://en.opensuse.org/images/e/e2/Flow1.jpg which goes deeper into how the user interacts with the system. You as the User will send commands to the Kernel using a Shell (in our case, Bash), where the Kernel then controls the Hardware. Think of the Kernel as a resource manager for hardware.

3 Using Git with Windows

Windows does not have a git client by default, so we will need to download software for that. This guide will utilize the popular *git for windows*, or better known as Git Bash.

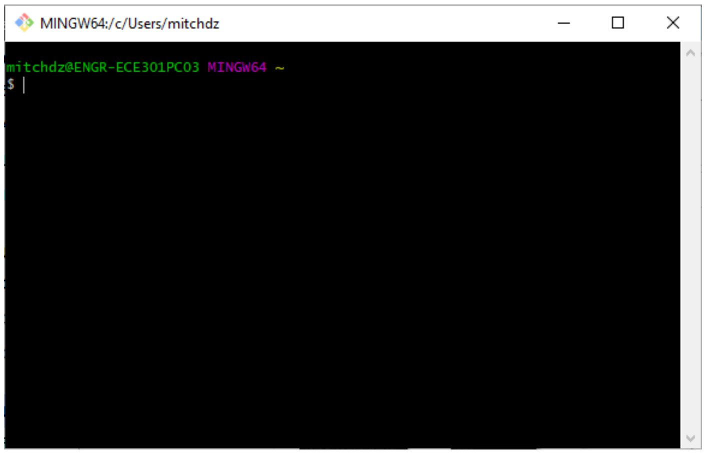
3.1 Git Bash

3.1.1 Installation

On the Lab PC, enter the website `https://gitforwindows.org/` and click "Download". Once the download is finished, open the installer. All of the default options are good, sane options, so just keep clicking "Next" Until the installation is done.

3.1.2 Using Git Bash

When you are finished downloading Git Bash from Section 3.1.1, you can open up Git Bash by typing in "git bash" after pressing the windows key. Upon pressing enter, you will be greeted with a terminal that looks like the Figure 2.

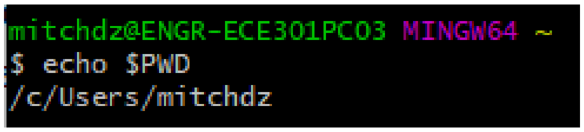


```
mitchdz@ENGR-ECE301PC03 MINGW64 ~
$ |
```

Figure 2: Git Bash Default Screen

Now that you are in the terminal, you can use any commands that any usual bash terminal would have. You can enter the following command **(do not type the $ at the start of the command, that is there by default)**:

```
$ echo $PWD
```



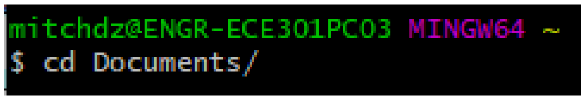
```
mitchdz@ENGR-ECE301PC03 MINGW64 ~
$ echo $PWD
/c/Users/mitchdz
```

Figure 3: Printing CWD in Git Bash

Figure 3 shows the result of running the above command. The result is "/c/Users/mitch". /c/ refers to the hard drive. "Users/mitch" refers to the path on that hard drive. The reason that you see the yellow tilde (that is the ~ character), is that ~ **is shorthand for "/c/Users/mitch"** (your username will show up instead of mitch).

You can change your directory into a folder where we can clone our repository. For this, we will be using the **cd** command. You can think of **cd** as standing for "Change Directory". Execute the following:

```
$ cd Documents/
```



```
mitchdz@ENGR-ECE301PC03 MINGW64 ~
$ cd Documents/
```

Figure 4: Changing Directory to Documents

Figure 4 shows the results of changing our directory to Documents. You can now note that the yellow text says "∼/Documents" which is actually referring to "/c/Users/mitch/Documents" in my case because the name for my User is mitch.

Let's create our own directory called 'git' in ∼Documents. This is where we will clone our git repository to be used for later.

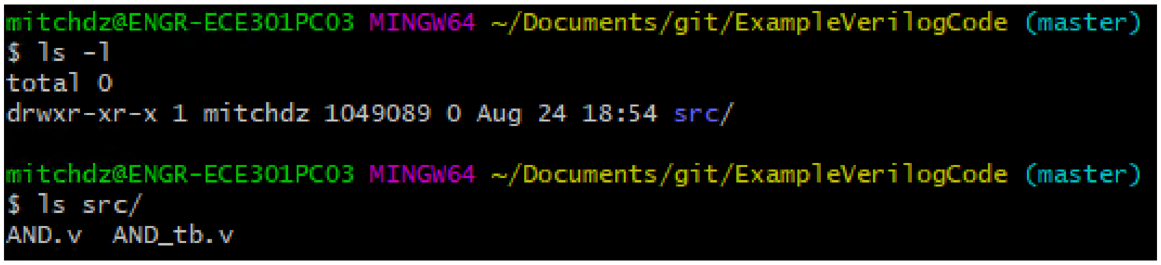
```
$ mkdir git
$ cd git
```

now you will be located in ∼/Documents/git. From here, let's clone a reference repository just to test everything. Execute the following command to clone an example repository with some Verilog code.

```
$ git clone https://github.com/mitchdz/ExampleVerilogCode
```

Now you have cloned a repository into your Documents folder that you will be able to see. In the terminal we can enter the new cloned repository and view the files with the **ls** command.

```
$ cd ExampleVerilogCode
$ ls -l
$ ls src/
```



```
mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/ExampleVerilogCode (master)
$ ls -l
total 0
drwxr-xr-x 1 mitchdz 1049089 0 Aug 24 18:54 src/

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/ExampleVerilogCode (master)
$ ls src/
AND.v  AND_tb.v
```

Figure 5: viewing the example Github repo

Figure 5 shows the results of the directory. We notice that one file exists named "src/". The blue name and the trailing "/" indicates that src is a directory. We can list the contents of the directory as shown in the same figure with "ls src/", where we see two files; AND.v and AND_tb.v

Now let's create a Vivado project that imports this folder. Open up Vivado and click Create a Project as shown in Figure 6

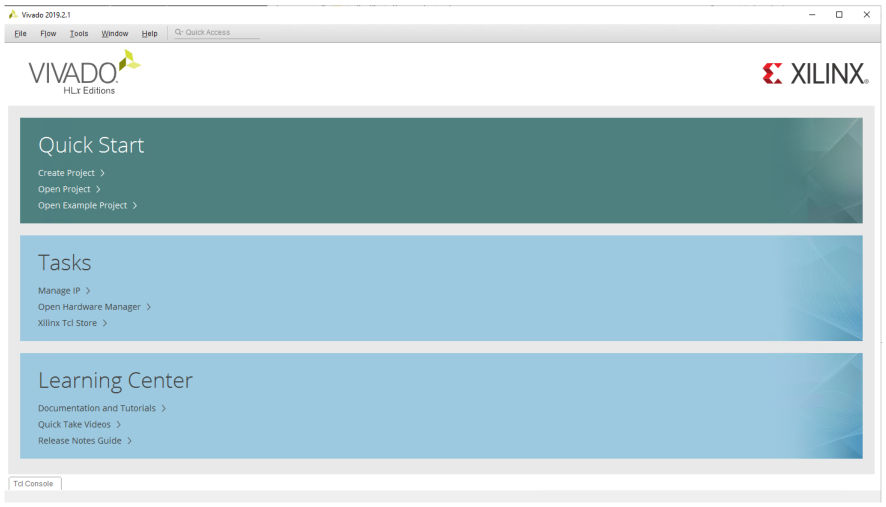
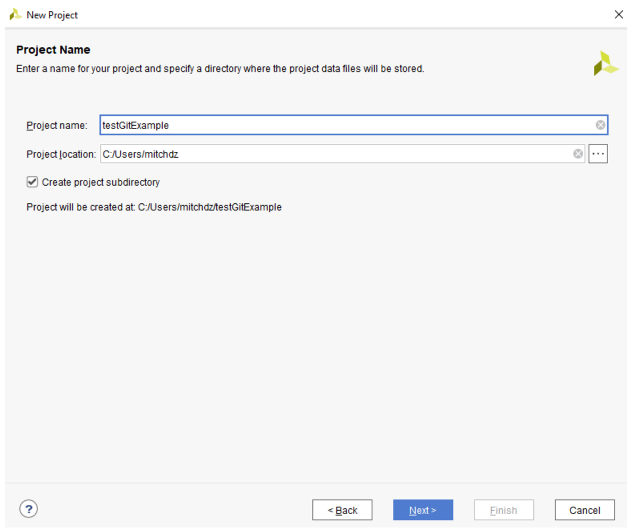


Figure 6: Vivado 2019.2.1 Default Screen

Name the Project "testGitExample" as shown in Figure 7



Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

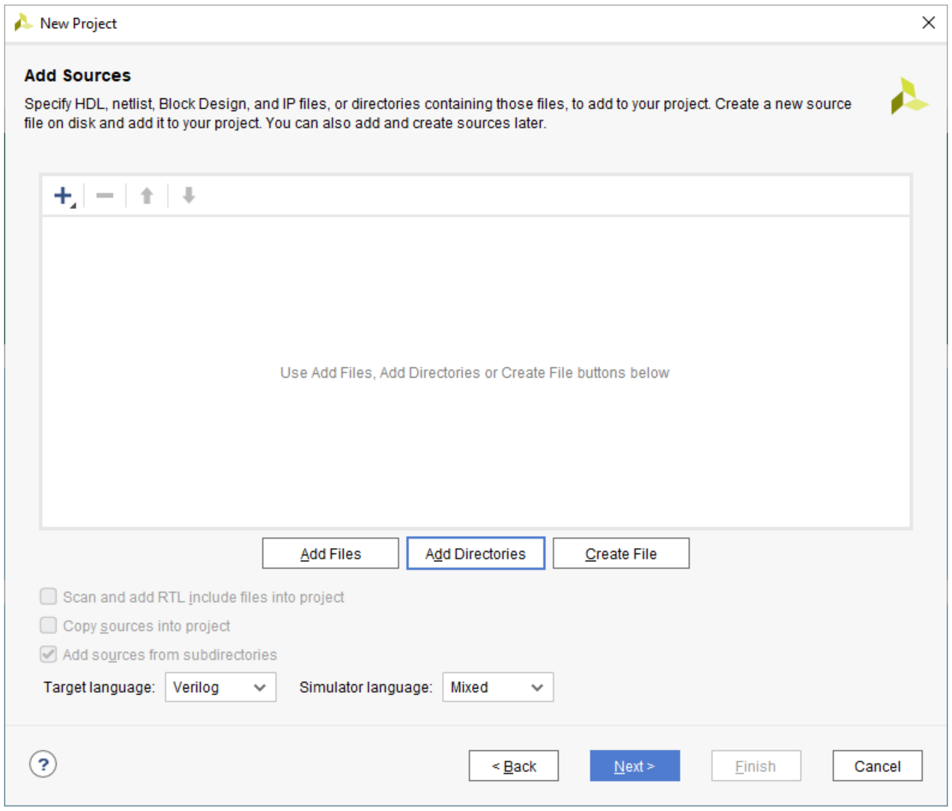
Project name: testGitExample

Project location: C:/Users/mitchdz

☑ Create project subdirectory

Project will be created at: C:/Users/mitchdz/testGitExample

Figure 7: Naming Example Project


New Project

Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

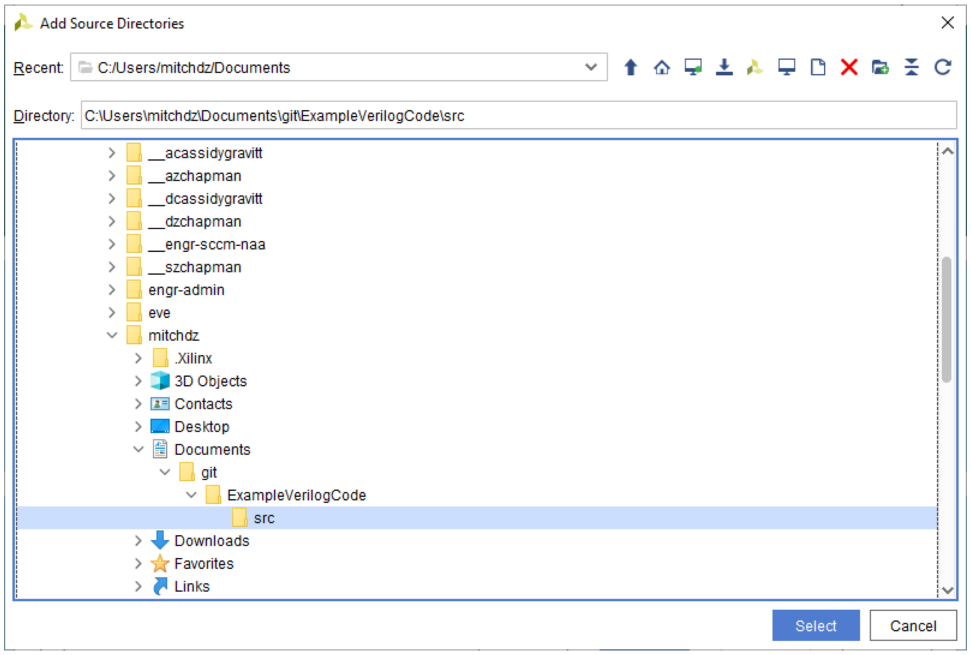
Use Add Files, Add Directories or Create File buttons below

- [ ] Scan and add RTL include files into project
- [ ] Copy sources into project
- [x] Add sources from subdirectories

Target language: Verilog   Simulator language: Mixed

Figure 8: Adding Sources Page

Choose RTL Project and then when it asks you to choose your sources, choose to 'Add Directories'. Figure 8 shows the window to select sources.


Add Source Directories

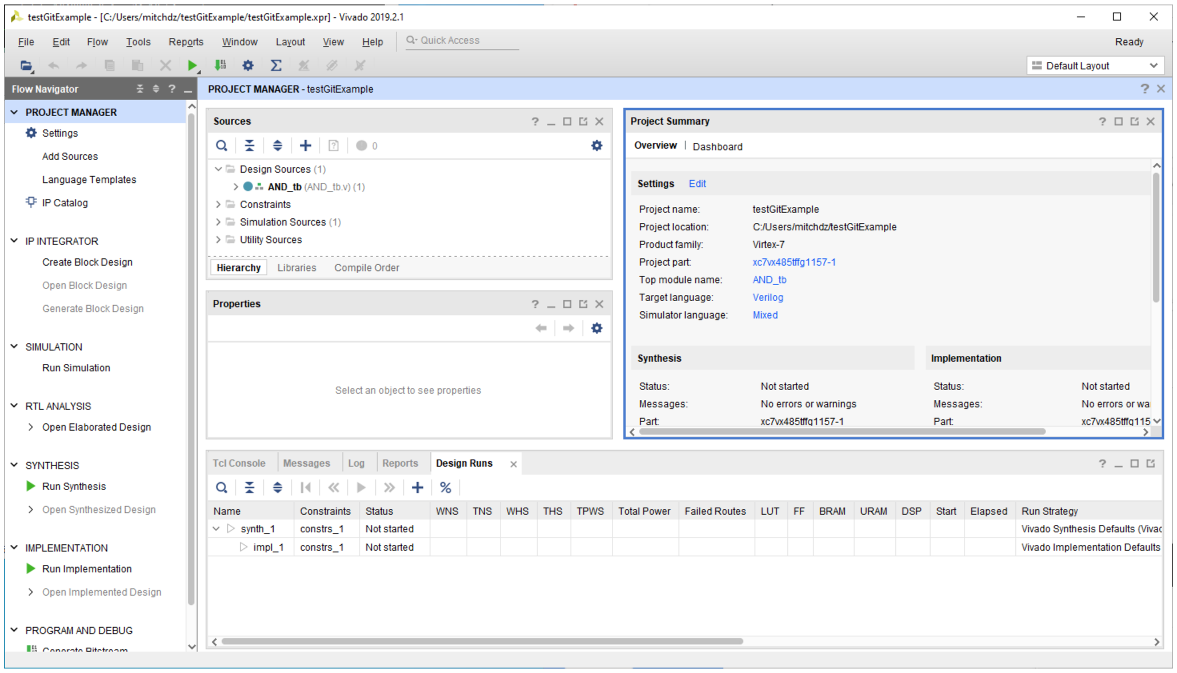
Recent: C:/Users/mitchdz/Documents

Directory: C:\Users\mitchdz\Documents\git\ExampleVerilogCode\src

Figure 9: Adding Example Code into Vivado

select the src/ directory of the ExampleVerilogCode just like how Figure 9 shows. Once you press Select, Vivado will pull this code into the project, and any changes made to this code will also be changed here. This allows us to easily source control the code because all of the modified code will be in a git directory.

Keep Pressing 'Next' And then you will get into the Vivado Project Manager as shown in Figure 10



Project name: testGitExample
Project location: C:/Users/mitchdz/testGitExample
Product family: Virtex-7
Project part: xc7vx485tffg1157-1
Top module name: AND_tb
Target language: Verilog
Simulator language: Mixed

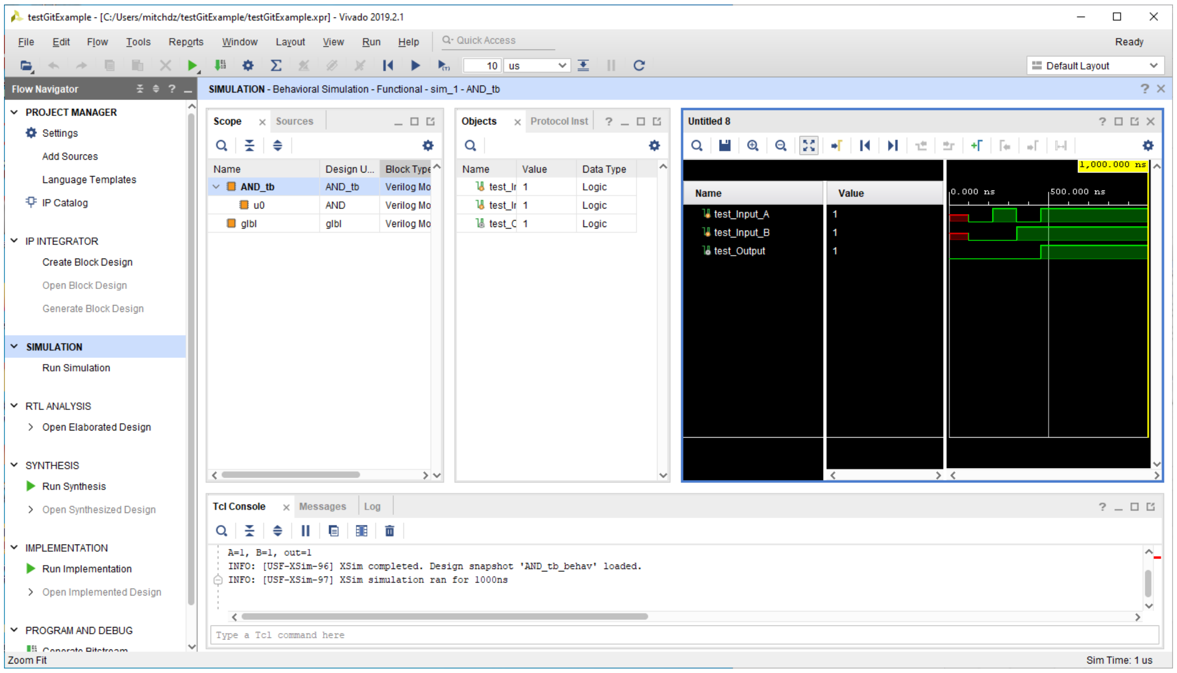
Synthesis
Status: Not started
Messages: No errors or warnings
Part: xc7vx485tffg1157-1

Implementation
Status: Not started
Messages: No errors or wa
Part: xc7vx485tffg115

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivac
impl_1	constrs_1	Not started															Vivado Implementation Defaults

Figure 10: Vivado Project Manager

Once you get into the Vivado Project Manager wait until you can see AND_tb in the Design Sources section as shwon in Figure 10. After that file shows up, press 'Run Simulation' on the left side. This will take a little bit.



Name	Design U...	Block Type
AND_tb	AND_tb	Verilog Mo
u0	AND	Verilog Mo
glbl	glbl	Verilog Mo

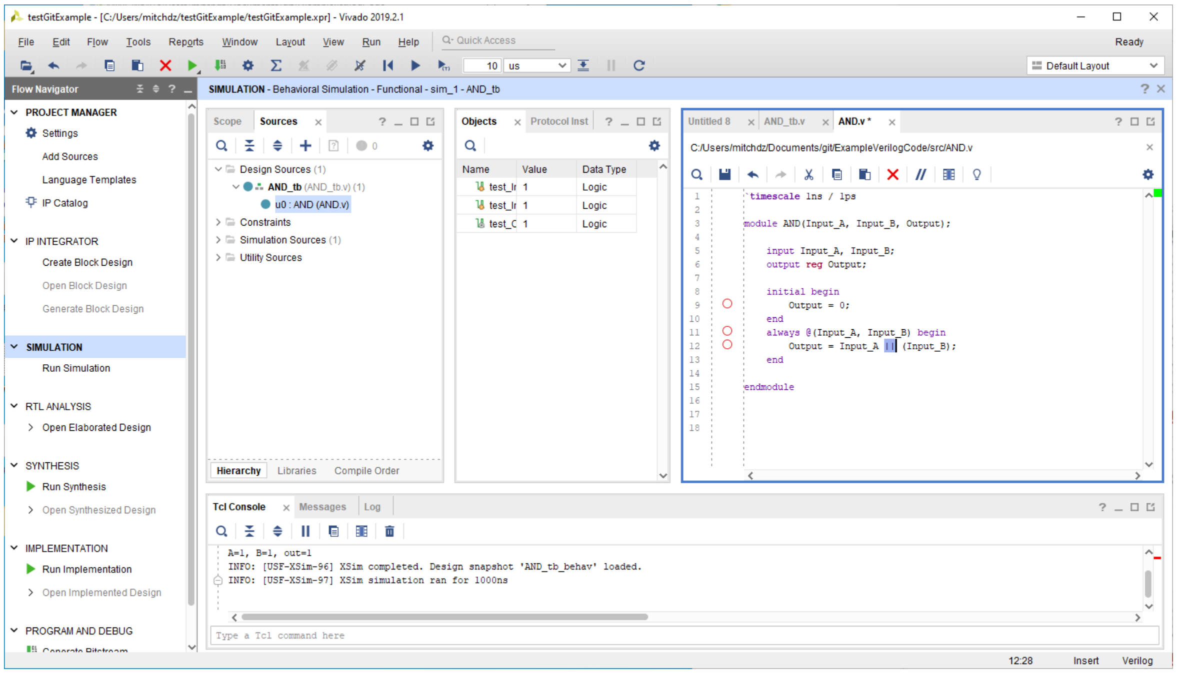
Name	Value	Data Type
test_Ir	1	Logic
test_Ir	1	Logic
test_C	1	Logic

```
A=1, B=1, out=1
INFO: [USF-XSim-96] XSim completed. Design snapshot 'AND_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

Figure 11: Running Simulation of Example Project

After the Simulation is done, you will see Figure 11. Click the "Zoom fit" button as shown being highlighted in the figure to be able to see the waveform better. The component being tested is an AND gate, so the results should make sense!

Let's make a change in the source code and see how to upload that to github.


testGitExample - [C:/Users/mitchdz/testGitExample/testGitExample.xpr] - Vivado 2019.2.1
File Edit Flow Tools Reports Window Layout View Run Help
Ready
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - AND_tb
Scope
Sources
Design Sources (1)
AND_tb (AND_tb.v) (1)
u0 : AND (AND.v)
Constraints
Simulation Sources (1)
Utility Sources
Hierarchy
Libraries
Compile Order
Objects
Protocol Inst
Untitled 8
AND_tb.v
AND.v *
C:/Users/mitchdz/Documents/git/ExampleVerilogCode/src/AND.v
Tcl Console
Messages
Log
12:28
Insert
Verilog

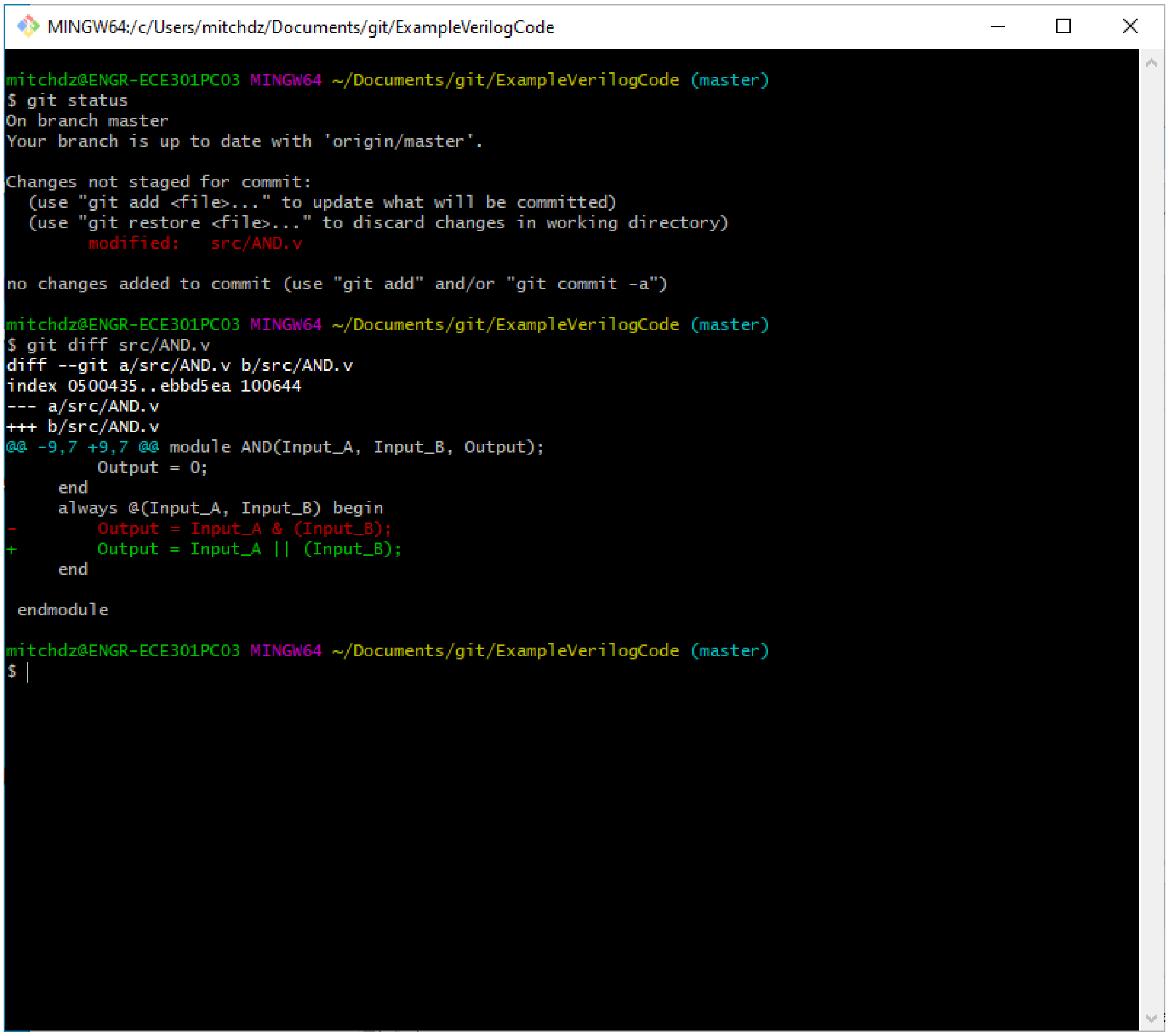
```
`timescale 1ns / 1ps

module AND(Input_A, Input_B, Output);

    input Input_A, Input_B;
    output reg Output;

    initial begin
        Output = 0;
    end
    always @(Input_A, Input_B) begin
        Output = Input_A || (Input_B);
    end

endmodule
```

```
A=1, B=1, out=1
INFO: [USF-XSim-96] XSim completed. Design snapshot 'AND_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

Figure 12: Modifying Example Source Code

Click the "Sources" Tab and then double click u0 as shown in Figure 12. Let's change the

AND gate to an OR gate. **Notice that There is a * next to AND.v* in the right highlighted section.** This means there is code that has not been saved. Pressing ctrl + s will save the code and make the asterick go away, then we can see the change in git.



```
mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/ExampleVerilogCode (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/AND.v

no changes added to commit (use "git add" and/or "git commit -a")

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/ExampleVerilogCode (master)
$ git diff src/AND.v
diff --git a/src/AND.v b/src/AND.v
index 0500435..ebbd5ea 100644
--- a/src/AND.v
+++ b/src/AND.v
@@ -9,7 +9,7 @@ module AND(Input_A, Input_B, Output);
         Output = 0;
     end
     always @(Input_A, Input_B) begin
-        Output = Input_A & (Input_B);
+        Output = Input_A || (Input_B);
     end

 endmodule

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/ExampleVerilogCode (master)
$ |
```

Figure 13: Viewing Modified Source Code in Git

Go back to Git Bash to check out the file changes. If you closed Git Bash and need to get back to the directory simply type ”cd /Documents/git/ExampleVerilogCode”. If not you can view the changes using git status and git diff as shown in Figure 13.

```
$ git status
$ git diff src/AND.v
```

So now we have a file that can be uploaded to git. The workflow for this as commands is as follows:

```
$ git add src/AND.v
$ git commit -m 'this text is the commit message text'
$ git push
```

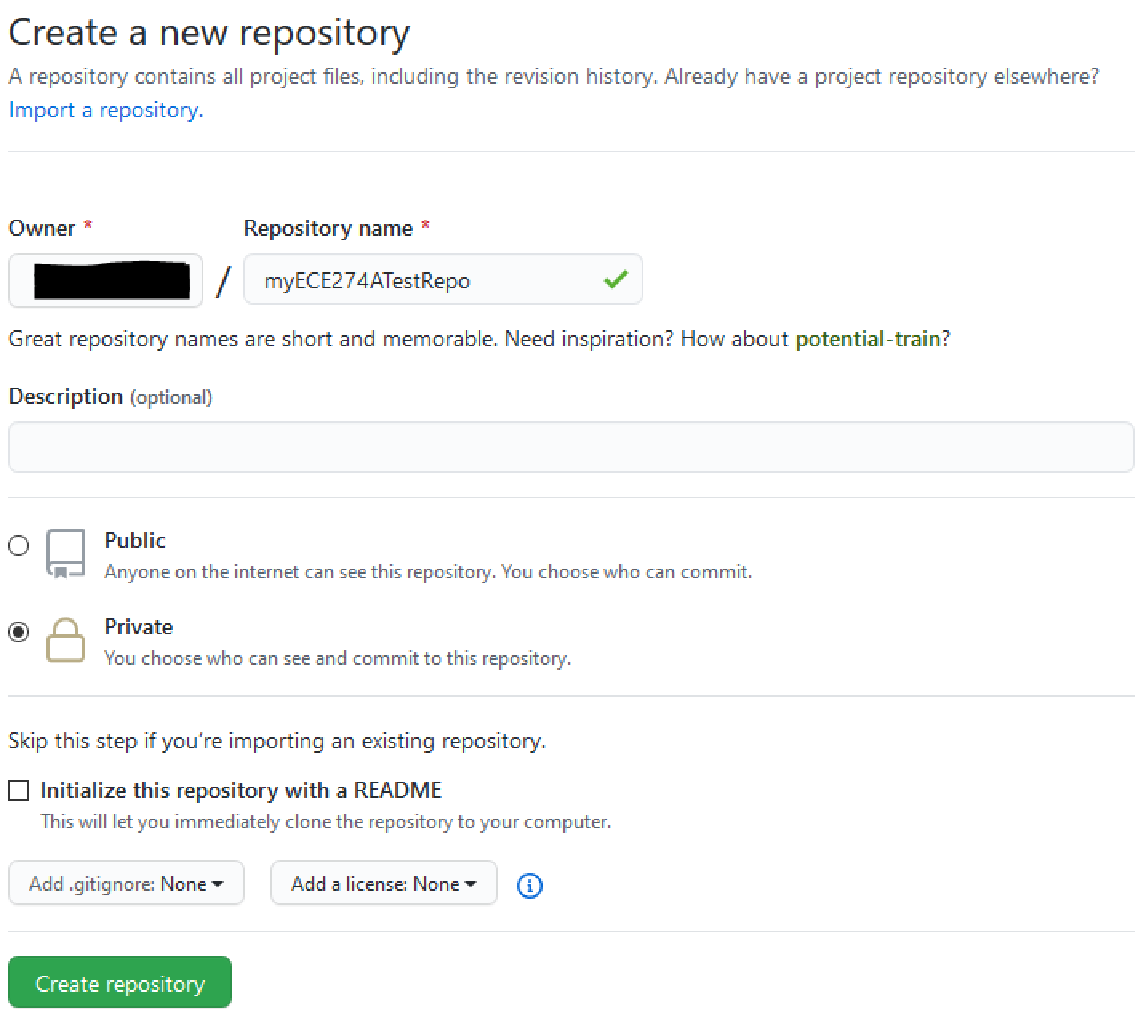
You won't be able to execute the last command because this repository does not belong to you. Proceed to Section 4 in order to create your own repo and test this out!

To recap, all you need to do in order to make the code appear from the computer to Github is:

```
$ git status # to check which files have changed
$ git add <file(s) to be commited>
$ git commit -m 'this text is the commit message text'
$ git push
```

4 Creating your own Github Repository

There is a lot of software that utilizes git such as GitHub, SourceForge, Bitbucket and GitLab. For this guide we will utilize Github. To get some cool bonuses for being a student you can even register for the student pack at https://education.github.com/pack.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * [redacted] / Repository name * myECE274ATestRepo ✓

Great repository names are short and memorable. Need inspiration? How about potential-train?

Description (optional)

○ Public
Anyone on the internet can see this repository. You choose who can commit.

● Private
You choose who can see and commit to this repository.

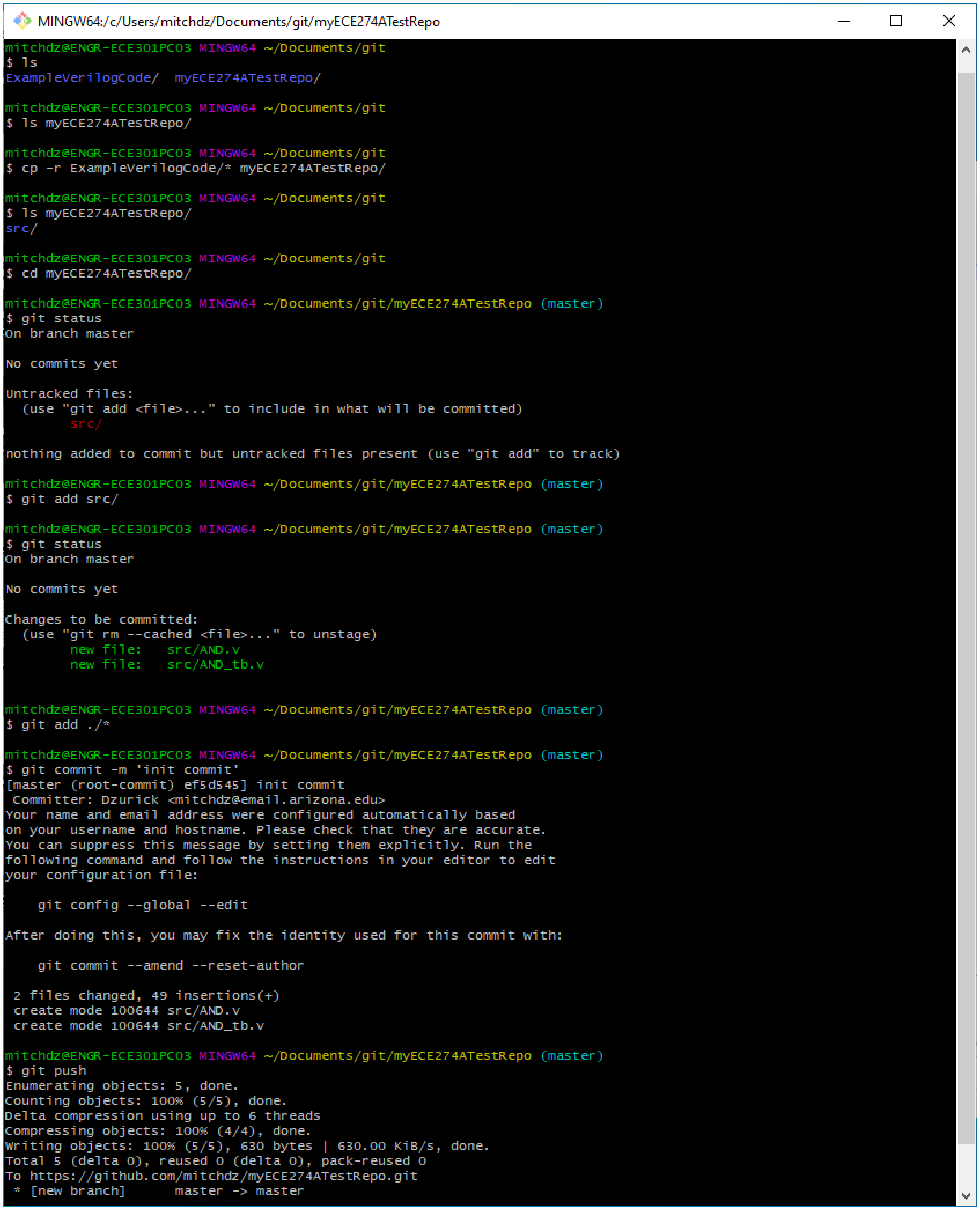
Skip this step if you're importing an existing repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Figure 14: Creating a new Github Repo

Go to your Github account after you create it and then click "New Project". You will be greeted with Figure 14. Name your repository whatever you want, but make sure that "Private" is selected. By default, repositories are Public. If you accidentally create a Public Repo it can be changed in Settings ¿ Options ¿ Change repository visibility.

After creating the repo, follow the steps from Section 3 with your own repository. Just copy the ExampleVerilogCode over to your new repository. An example of this is shown below in Figure 15


MINGW64:/c/Users/mitchdz/Documents/git/myECE274ATestRepo

```
mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git
$ ls
ExampleVerilogCode/  myECE274ATestRepo/

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git
$ ls myECE274ATestRepo/

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git
$ cp -r ExampleVerilogCode/* myECE274ATestRepo/

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git
$ ls myECE274ATestRepo/
src/

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git
$ cd myECE274ATestRepo/

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/myECE274ATestRepo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        src/

nothing added to commit but untracked files present (use "git add" to track)

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/myECE274ATestRepo (master)
$ git add src/

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/myECE274ATestRepo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   src/AND.v
        new file:   src/AND_tb.v


mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/myECE274ATestRepo (master)
$ git add ./*

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/myECE274ATestRepo (master)
$ git commit -m 'init commit'
[master (root-commit) ef5d545] init commit
 Committer: Dzurick <mitchdz@email.arizona.edu>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 2 files changed, 49 insertions(+)
 create mode 100644 src/AND.v
 create mode 100644 src/AND_tb.v

mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/myECE274ATestRepo (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 6 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 630 bytes | 630.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/mitchdz/myECE274ATestRepo.git
 * [new branch]      master -> master
```

Figure 15: adding Example Code to Personal Repo and Pushing

To review Figure 15, the following commands are used:

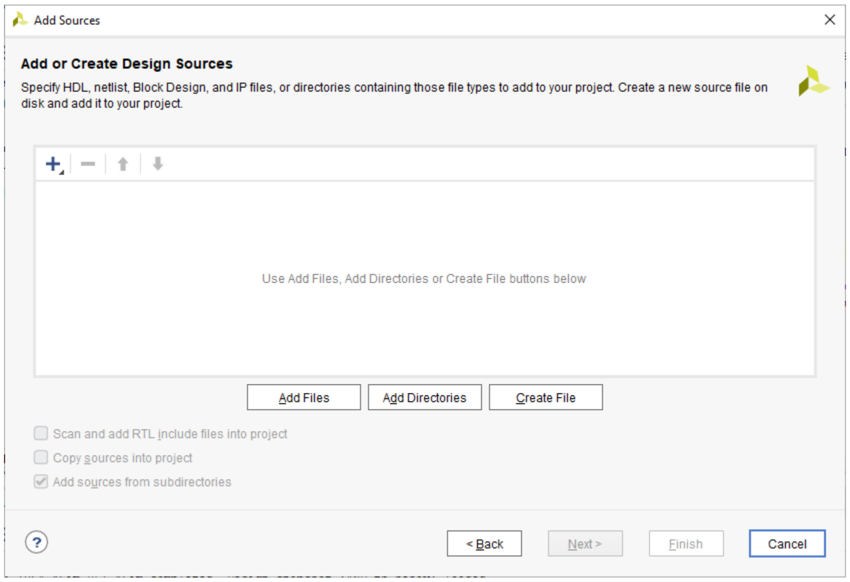
```
1 $ cp -r ExampleVerilogCode/* myECE274ATestRepo/
2 $ cd myECE274ATestRepo
3 $ git status
4 $ git add ./*
5 $ git commit -m 'init commit'
6 $ git push
```

Line 1 uses the cp command, which you can think of as "copy". This command copies the contents of ExampleVerilogCode into myECE274ATestRepo. Line 4 uses something new which is the *. The * is a special character called a wildcard. The * refers to matching any string, so what Line 4 is doing is adding every file in the directory to git.

5 Adding files in Vivado

Say you need to add a file in Vivado, but want that file to appear in your git directory. No problem!

In Vivado go to File ¿ Add Sources.. (or the shortcut Alt + A). Click **Add or create design sources**.



Add Sources

Add or Create Design Sources

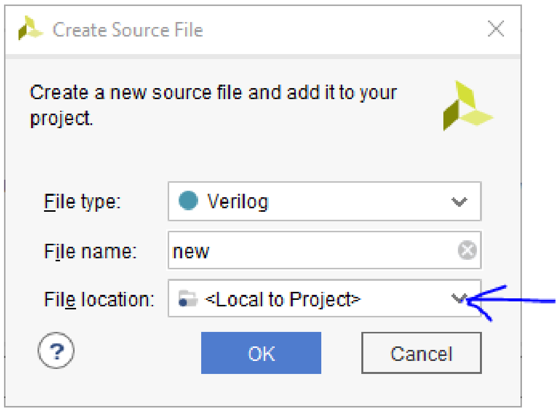
Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

Use Add Files, Add Directories or Create File buttons below

[ ] Scan and add RTL include files into project
[ ] Copy sources into project
[✓] Add sources from subdirectories

Figure 16: Adding Sources in Verilog

Click "Create File" in Figure 16.



Create a new source file and add it to your project.

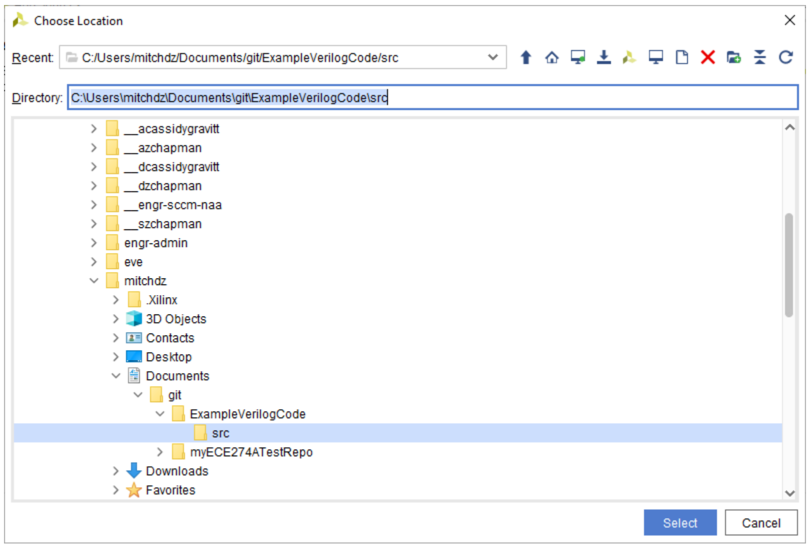
File type: Verilog

File name: new

File location: <Local to Project>

Figure 17: Create Source File Screen

Click the drop down arrow that is being pointed to in Figure 17. Make sure to make a name for this file, in this case the file name is ”new”.



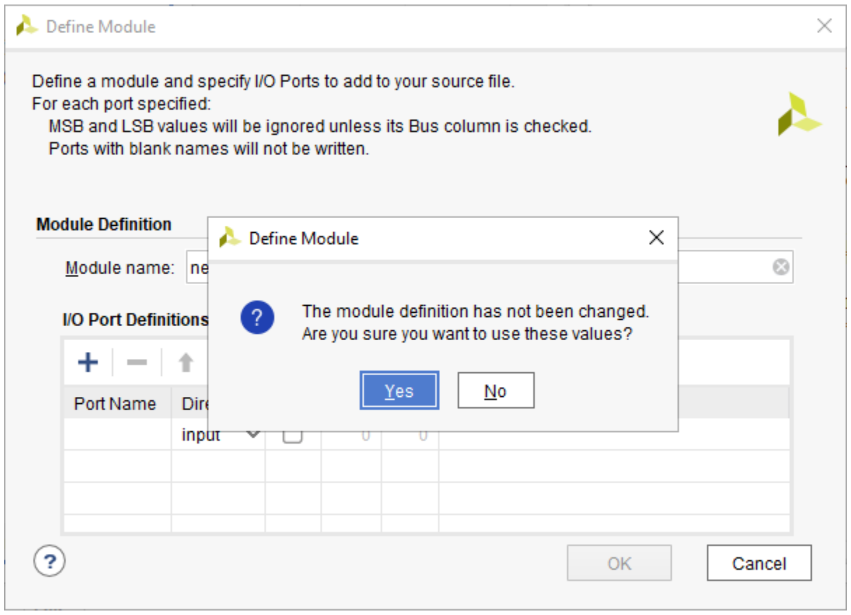
Recent: C:/Users/mitchdz/Documents/git/ExampleVerilogCode/src

Directory: C:\Users\mitchdz\Documents\git\ExampleVerilogCode\src

```
> __acassidygravitt
> __azchapman
> __dcassidygravitt
> __dzchapman
> __engr-sccm-naa
> __szchapman
> engr-admin
> eve
v mitchdz
    > .Xilinx
    > 3D Objects
    > Contacts
    > Desktop
    v Documents
        v git
            v ExampleVerilogCode
                src
            > myECE274ATestRepo
    > Downloads
    > Favorites
```

Figure 18: Selecting Target Location

Figure 18 shows the Location Highlights /Documents/git/ExampleVerilogCode/git which was highlighted by default! Simply press Select.


Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name: ne

I/O Port Definitions

Define Module

The module definition has not been changed.
Are you sure you want to use these values?

Figure 19: Defining Module for New File

You'll need to define a Module for your new file as shown in Figure 19. Just select "OK" and then "Yes". Wait a little bit and your new file will be in your git directory ready to be modified!

One final test...

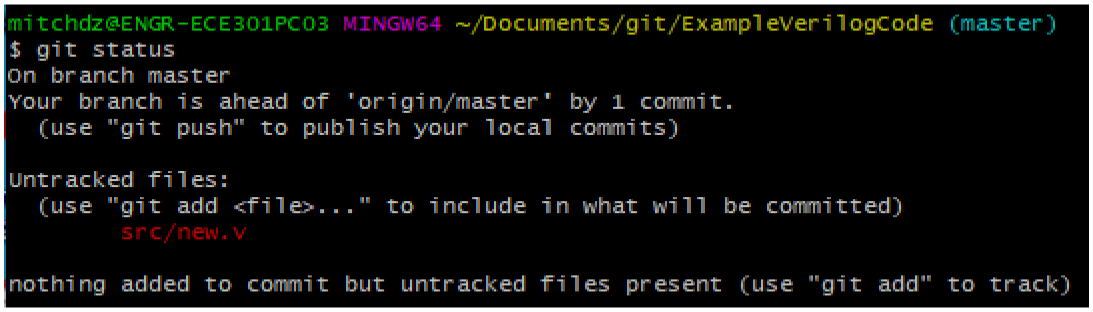

```
mitchdz@ENGR-ECE301PC03 MINGW64 ~/Documents/git/ExampleVerilogCode (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        src/new.v

nothing added to commit but untracked files present (use "git add" to track)
```

Figure 20: Checking For The New File in Git

The new file is there!