525.612.81 – Computer Architecture
Setting up for the FPGA tools
Dr. Nicholas Beser

Welcome to the Spring Computer Architecture class. During this semester we will be studying RISC architectures by simulating them using Verilog code, running them on the Xilinx Artix-7 FPGA using the Nexys board. While the tools and code will run on Windows 10, and Linux, I find that the Linux systems seem to have the least issues in interfacing and tool development.

To help the class get started I recommend that we use the linux environment (Ubuntu 20.04). If you have a PC or Mac, it is possible to run linux using a VM package. The Mac (VMFusion) system works well. The VirtualBox software is free and runs on both Mac and PC. This memo will provide instructions on setting up a system that is sized to run Xilinx Vivado, and the MIPS assembler and C compiler. The plan is to get the class ready to work with the Nexys evaluation board, and have a system ready to go.

Please note that the new Mac does not support VM systems. There may be a version of VMFusion for the new Mac that could work, but I am not able to test it.

The following software packages will be installed as part of this exercise:

1. VirtualBox (PC or Mac)
2. Ubuntu 20.04 on the VirtualBox system
3. Xlinx Vivado (Latest Version) – 2020.2
4. Installation of missing library for Xilinx Code
5. Cable Drivers for linux
6. MIPS-MTI-ELF Assembler and C-Compiler
7. Make Utility
8. Git Utility
9. Python 3 (required for script that controls PMODUSBUart

If you have taken the JHU FPGA classes, you should have an account on www.xilinx.com. If you don't, you should go to the web site and get a free account. The Xilinx Vivado software can be operated with the free license, but you need the account to install.

You should also go to www.bitbucket.org and apply for a free account. The class keeps a repository of FPGA code at that site. Once you get your account, please send me your username. I will then add you to the class list. That will give you access to the repository. We have replaced the Bus Blaster with a Digilent PMODUSBUart card. You will need two USB ports on your computer to both download code to Nexys Board, and to reprogram the MIPS processor under software control.

Resources Needed to Run Linux on your system:

The Xilinx Vivado code is about 20Gbytes. I think that you will need at a minimum 100 Gybte free storage to setup VirtualBox with Ubunto. (More would be better). I do a lot of development on my VMFusion and operate with 250Gytes. I think 150Gbytes will probably be sufficient for the semester.

Your linux system will need RAM. If you have only 8Gbytes in your system, you may have a problem. I am currently running with 16Gbytes, with 8Gbytes allocated to VirtualBox.

Number of Processors: The more processors you have the faster Vivado will operate. I have 8 on my mac, and can allocate 4 to VirtualBox. I have 4 on my PC and can only allocate 2 for VirtualBox. The rule of thumb is that the more you have, the more you will use.

Install VirtualBox

Installing VirtualBox is relatively easy. Down load the software from:

- https://www.virtualbox.org/wiki/Downloads

In addition to providing either a PC or Mac version (they also have a linux version), there is a VirtualBox Extension Pack which is required to upgrade the USB ports to version 3.0. You need to install the VirtualBox Extension Pack by clicking on the module once VirtualBox is installed. Install the VirtualBox Extension Pack immediately after installing VirtualBox,

Installing Ubuntu 20.04 on Virtual Box

Download the Ubuntu 20.04 iso image from https://ubuntu.com/download/desktop.
The iso image will be used by VirtualBox to create the linux system. Start VirtualBox, and click on the New Symbol. Give the virtual OS a name (I called my Ubuntu 20.04).

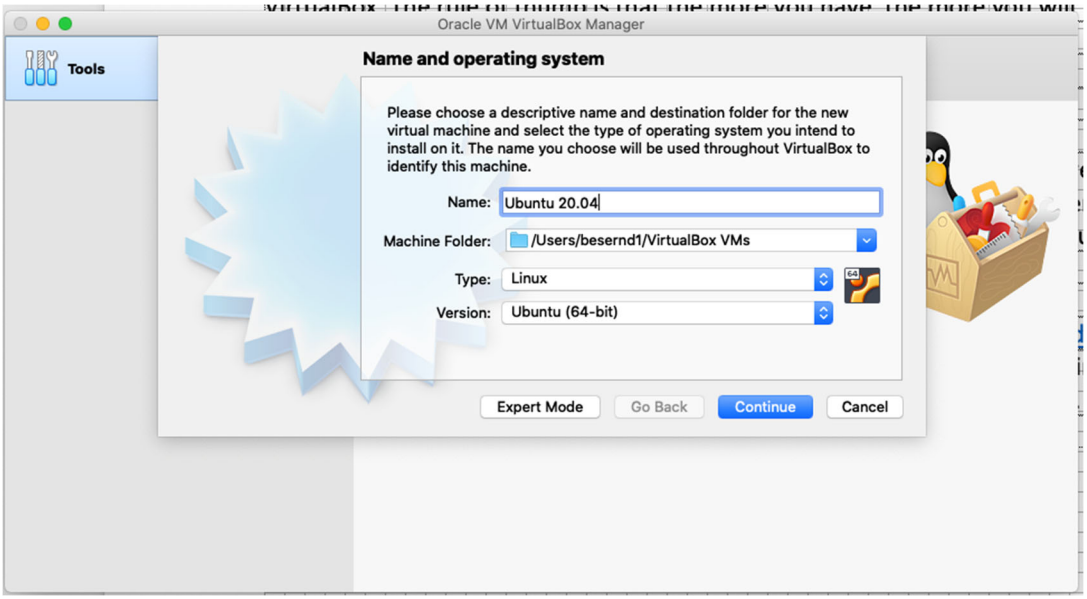

Figure 1 – Setting up the Linux Operating System Name

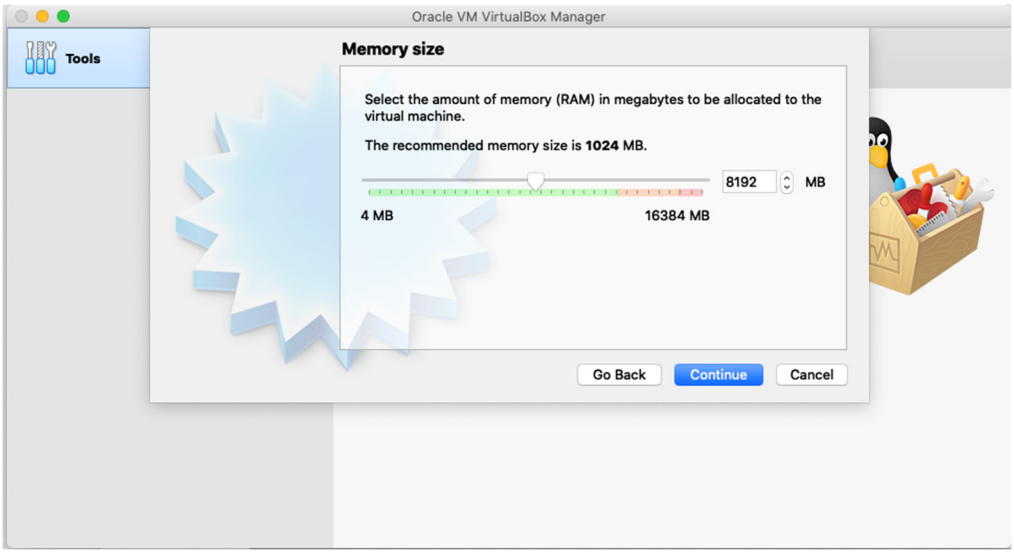

Figure 2 – Set the amount of memory for the VM system (recommend 8Gbyte)

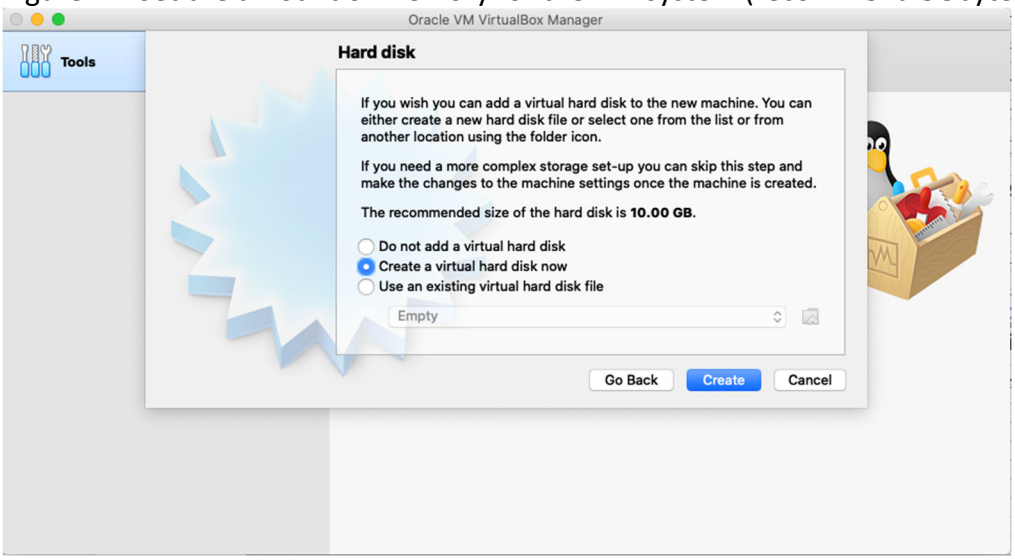

Figure 3 – Setup the Hard Disk as a Virtual Hard Disk

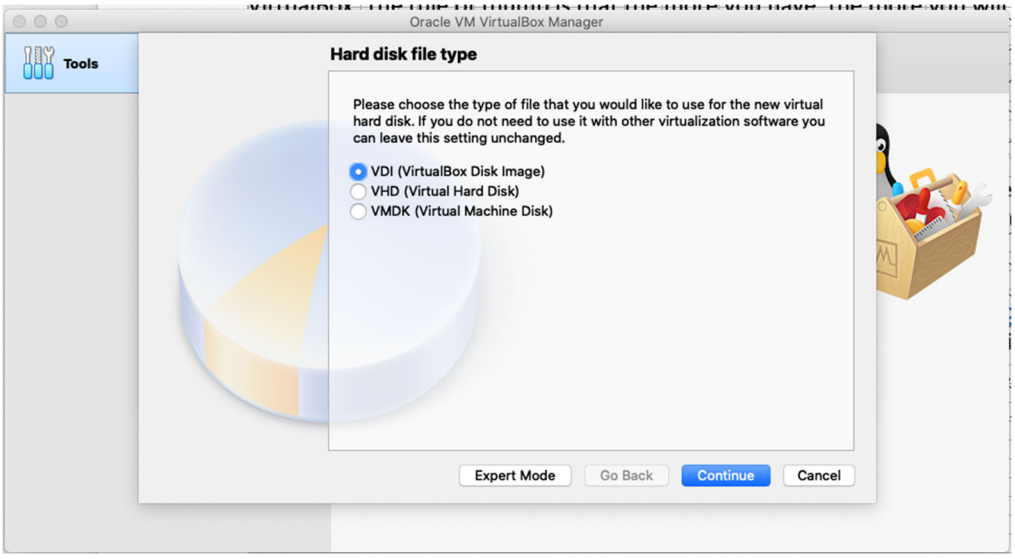

Figure 4 – Setup the Hard Disk Type as a VirtualBox Disk Image

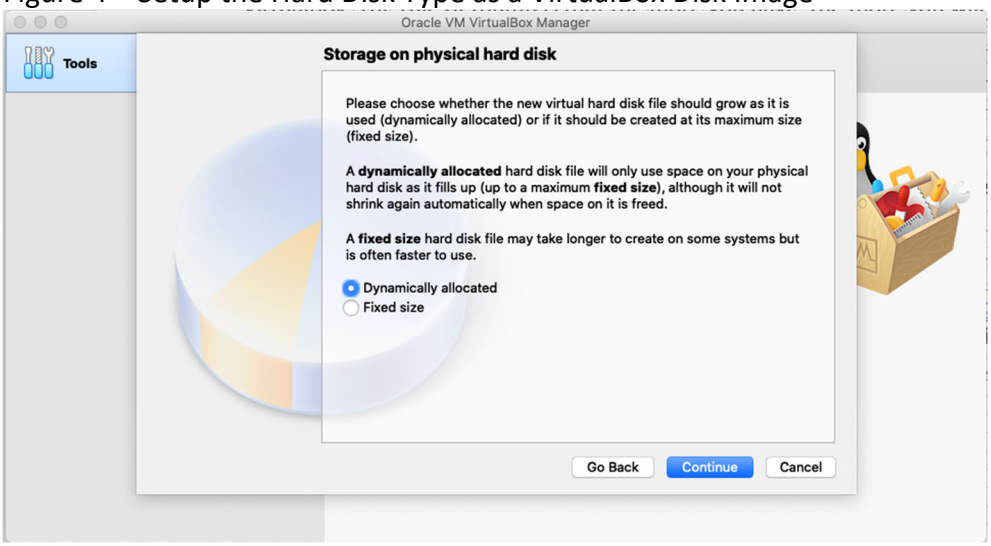

Figure 5 – Set the Storage to be Dynamically Allocated. It will grow up to 100Gyte. The upper cap can be changed later if required.

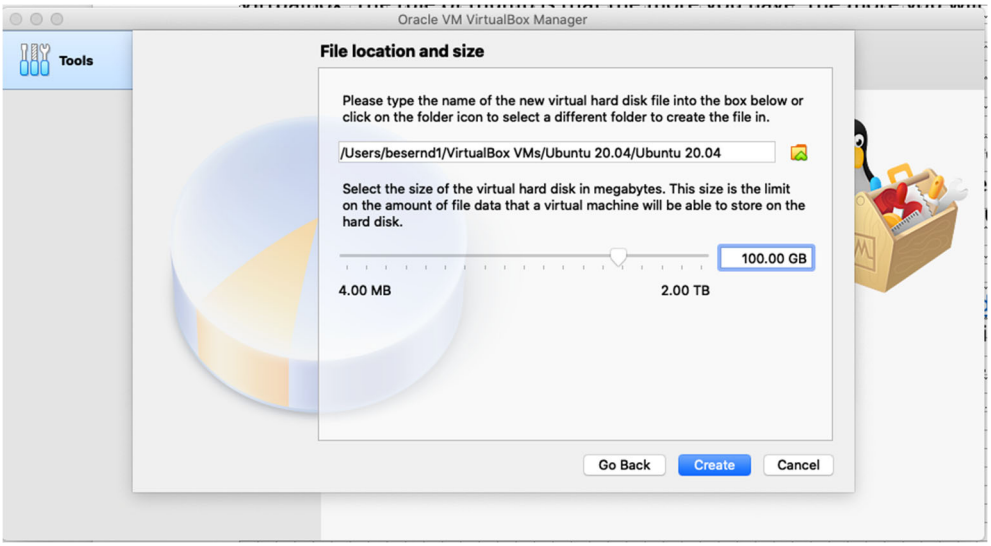

Figure 6 – Set the size of the VirtualBox drive to be 100 Gbyte. This can be modified later if more disk space is required.

Figure 7 – The basic system is now in place. Before installing linux, I recommend a number of parameters should be changed (video memory, USB port)

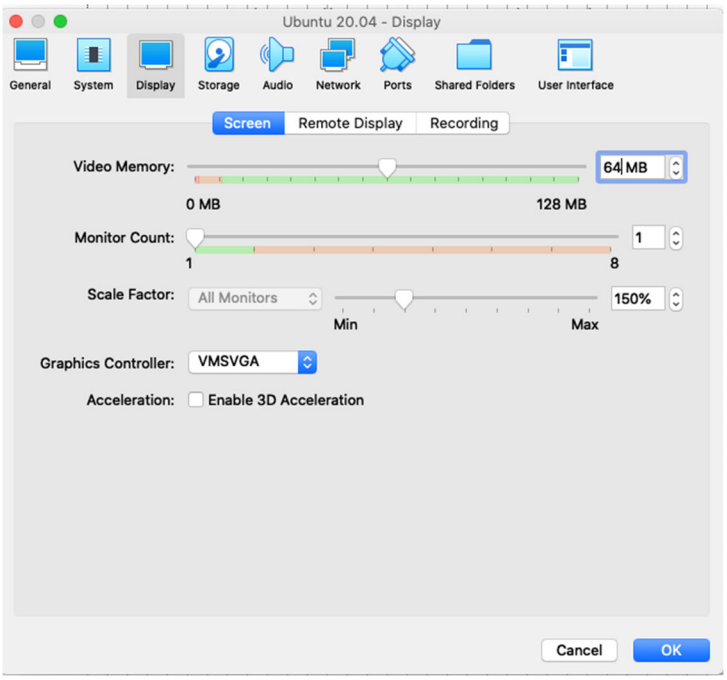

Figure 8. Go to settings, and select display. Change the default size to 64Mbyte. This will allow higher resolution to be selected (up to 1920x1080). If you have multiple monitors, you can also select that option here. You can also go back and change these as required.

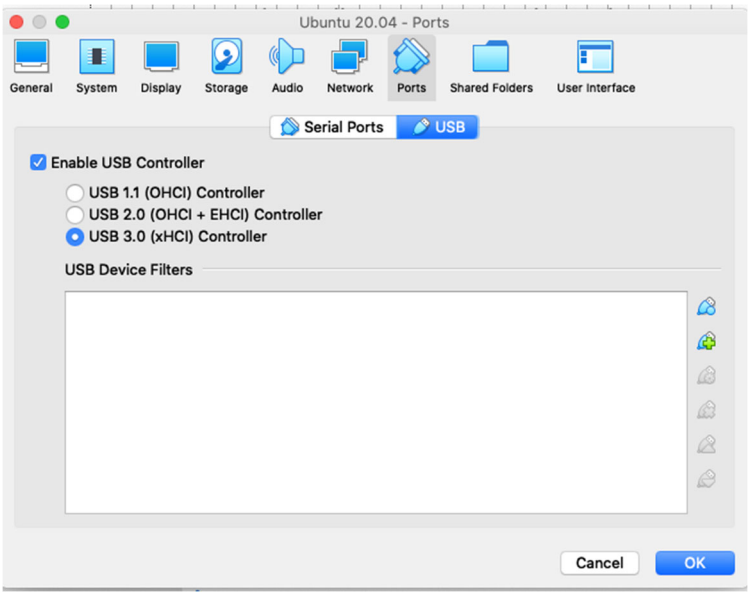

Figure 9 – Select Ports and then select USB. The default is USB 1.1. If you are not able to select USB 2.0 or USB 3.0, you need to exit and install the VirtualBox Extension Pack. The installation can be started by clicking on the VirtualBox Extension Pack. That will provide drivers for USB 3.0. Select 3.0 and then go to the filter operation (Button on the right.)

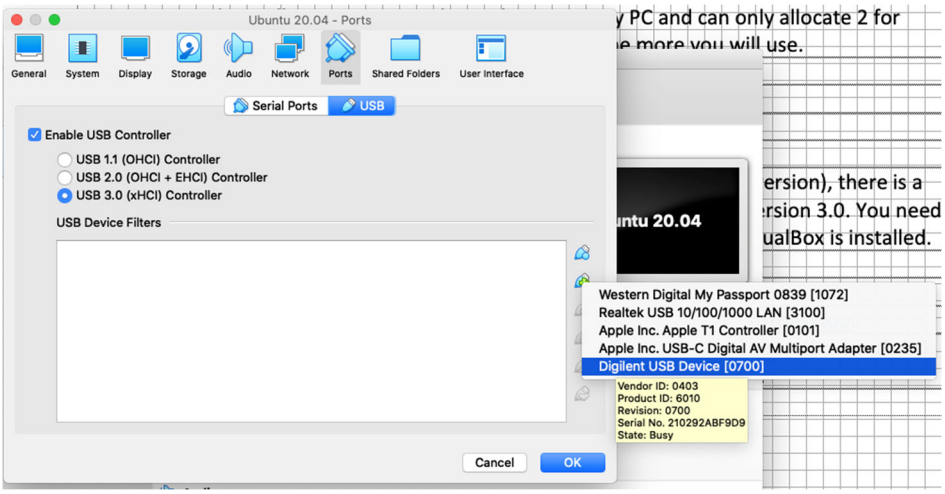

Figure 10 – The second button on the right adds new USB filters. If the Nexys Board is plugged into the system, it will appear on the right pull down menu. Select it.

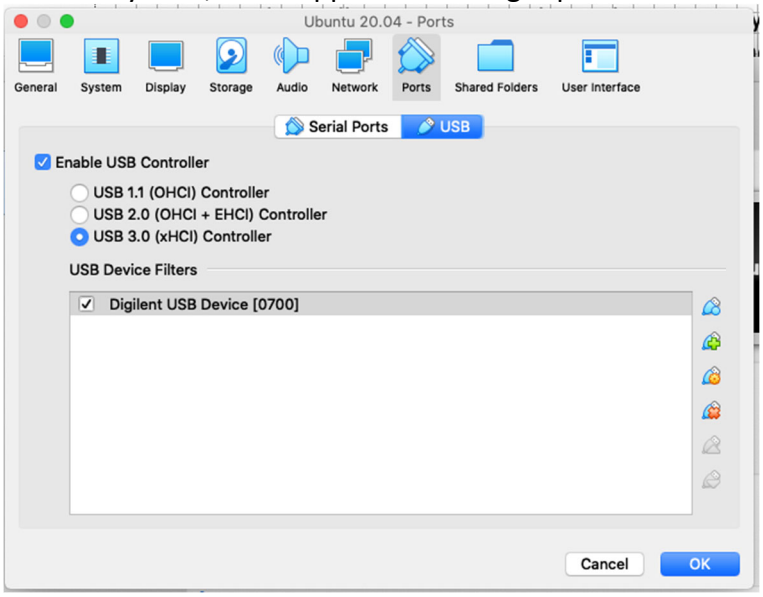

Figure 11 – Once the Digilent USB Device is set, the board will be visible to the OS installed with VirtualBox.

To start the install, click on the green start arrow. You will need to provide the location of the ISO file

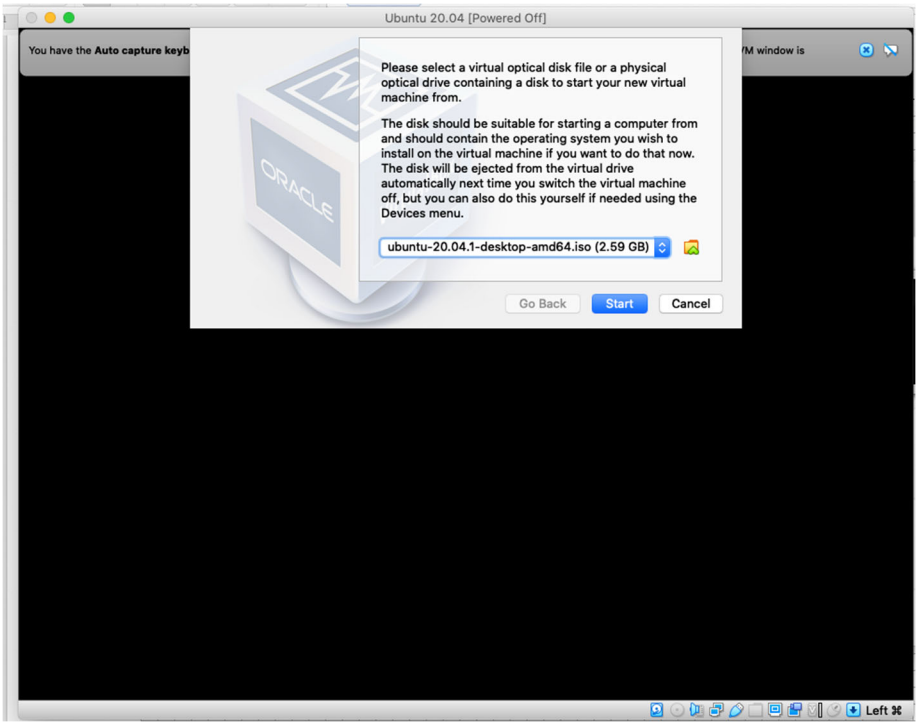

Figure 12 – Once the iso file is located, you can then click on start which installs Ubuntu 20.04 on the virtual hard drive.

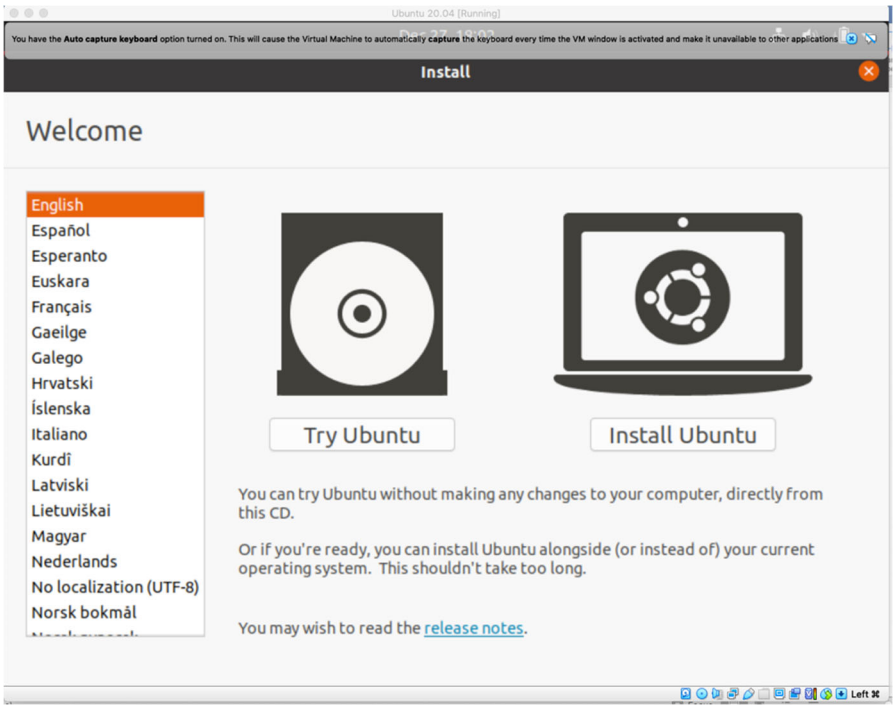

Figure 13 – Press Install Ubuntu to begin Linux installation

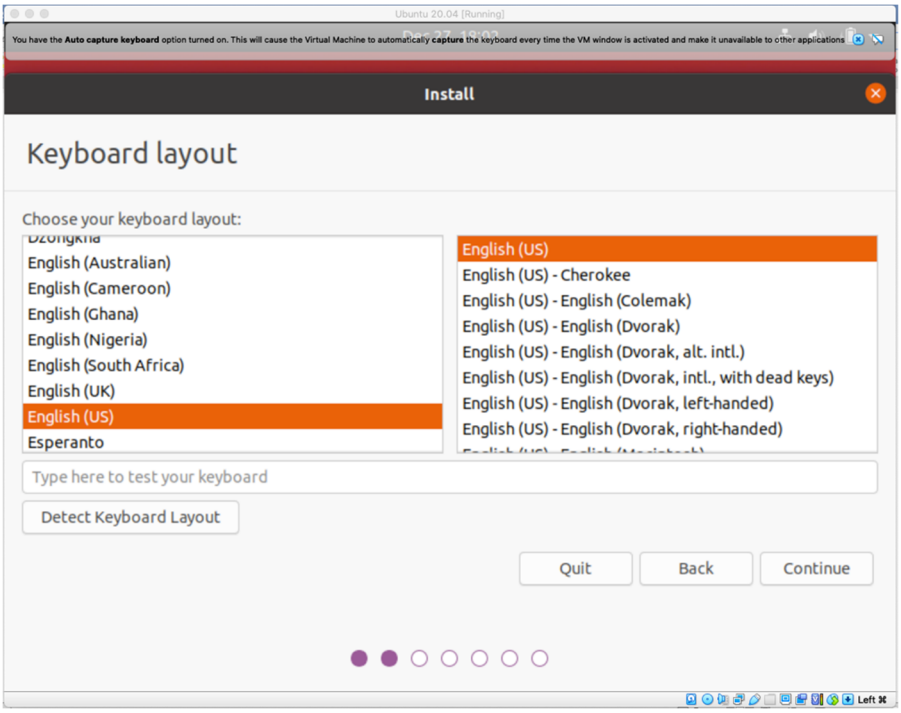

Figure 14 – Select English (US) Keyboard layout

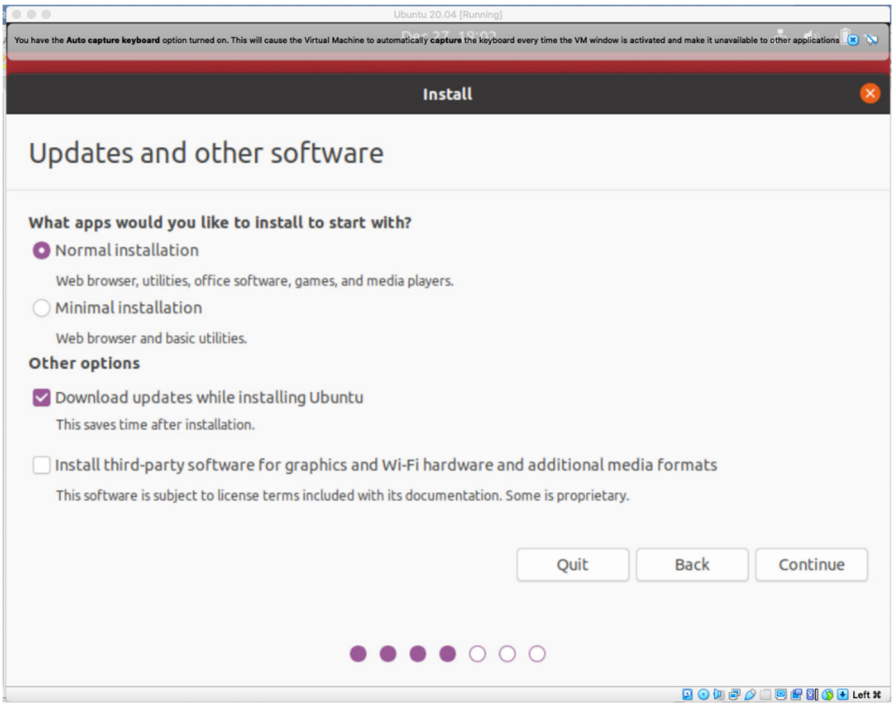

Figure 15 – Select Normal installation and Download updates while installing Ubuntu

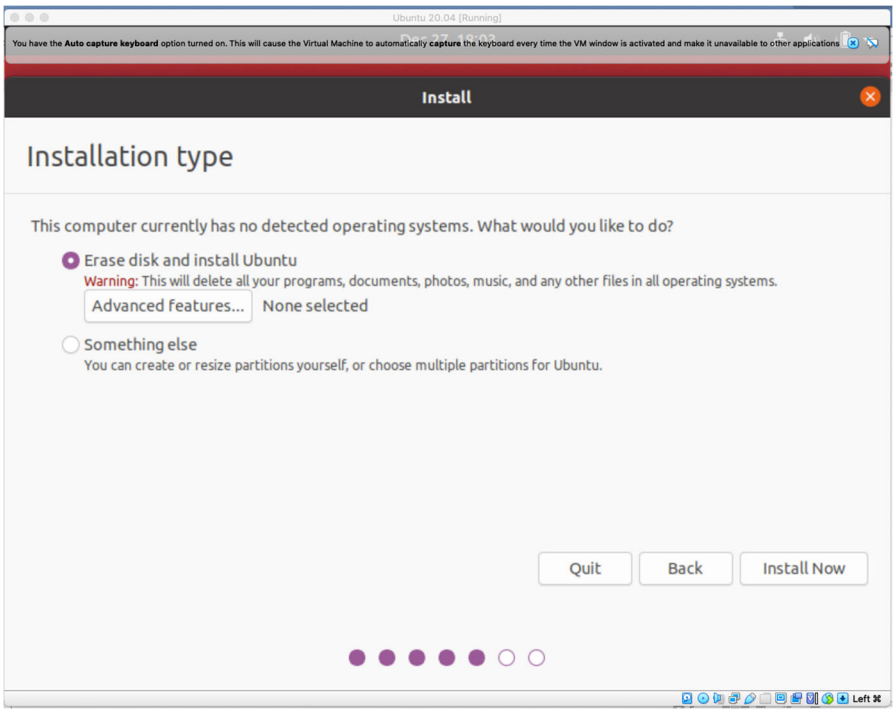

Figure 16 – Select Erase Disk and Install Ubuntu – Note this will only erase the VM disk, not your hard drive.

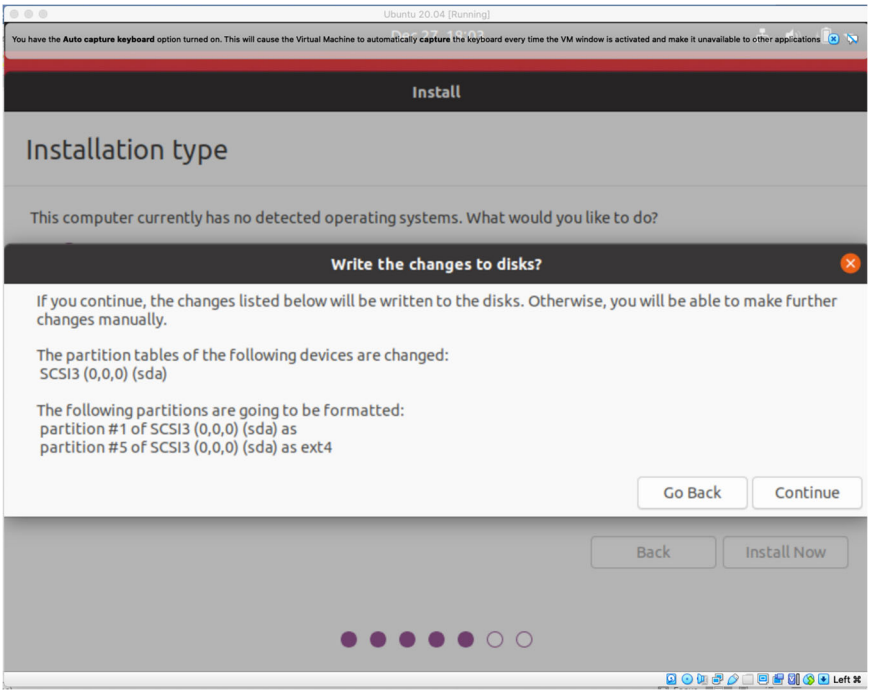

Figure 17 – Since the VM disk has no operating system on it, just continue to install.

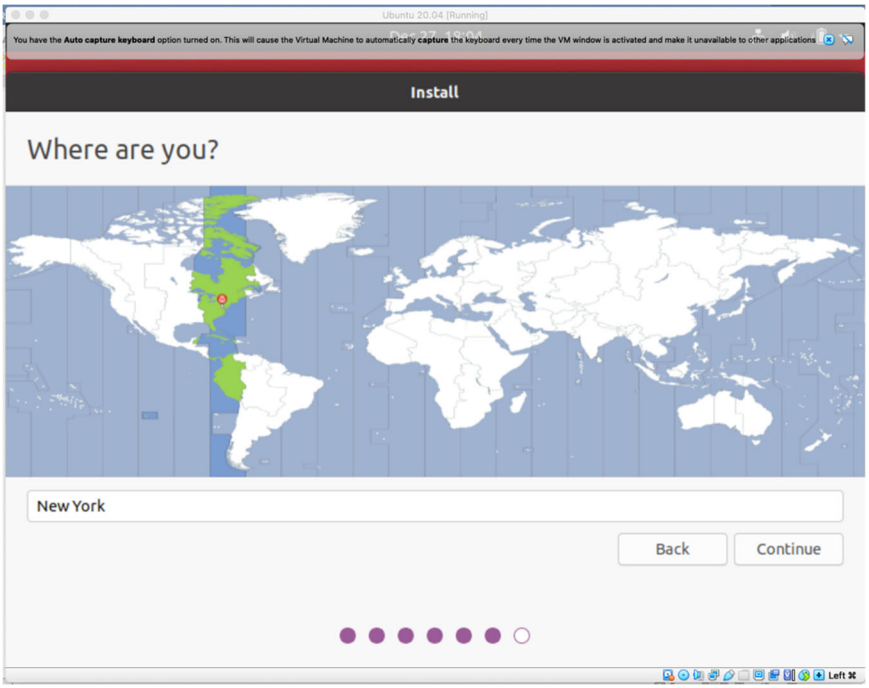

Figure 18 – Identify your time zone. This setting is for east coast.

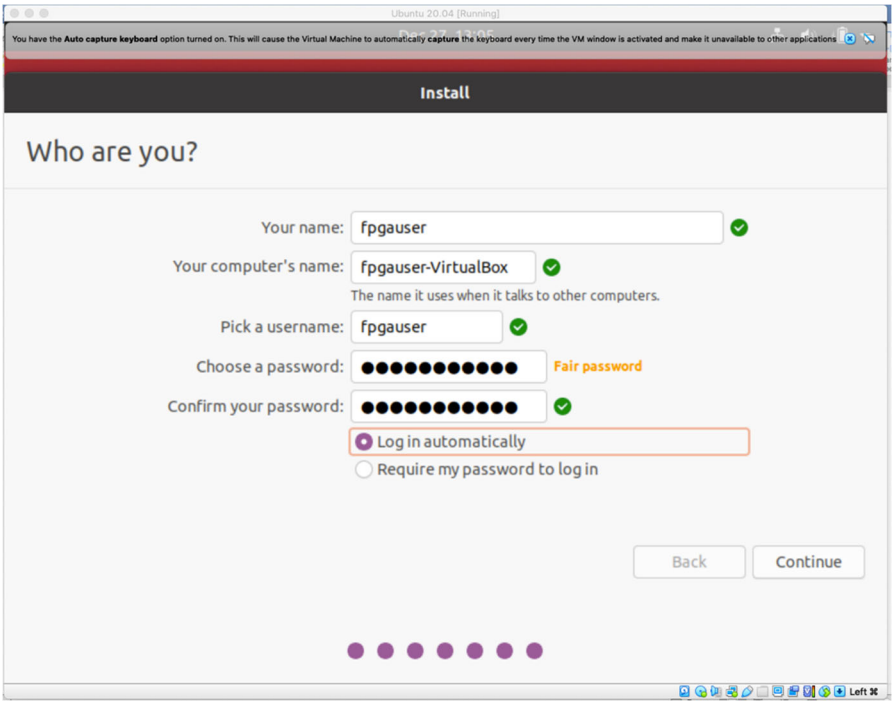

Figure 19 – These settings are up to the student. I chose fpguser with a common username. You should select your own username for security purposes.

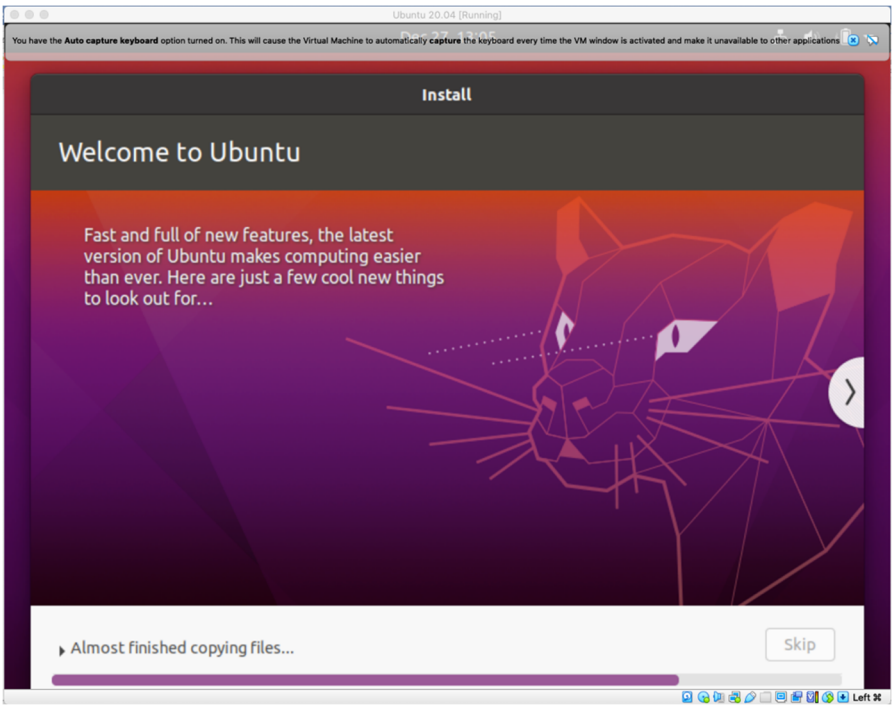

Figure 20 – Installation is proceeding normally

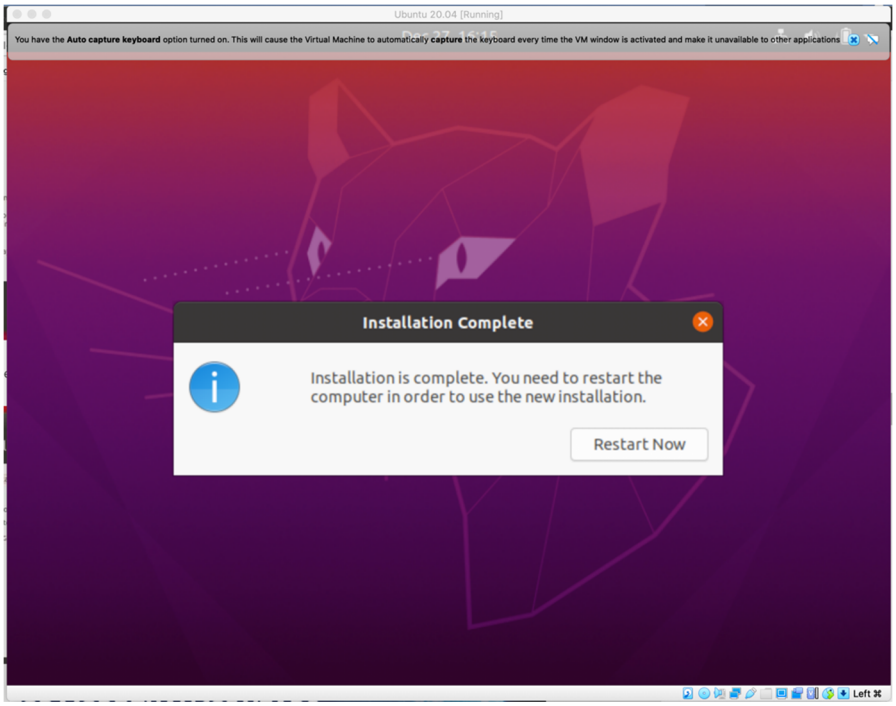

Figure 21 – Installation is complete. Restart the VM and answer the questions on startup. It is safe to skip most. Go to settings to change the resolution.

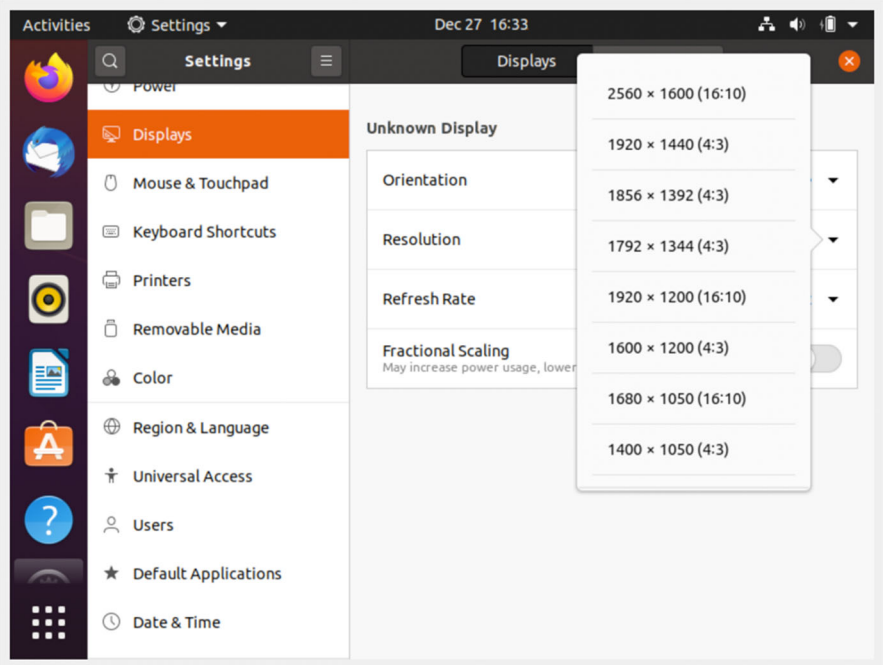

Figure 22 – Select Display and change resolution (I selected 1920x1200)

Your updater might also say that there are changes to be added to the Ubuntu 20.04. I accept the changes and have them added to the OS.

At this point, I would also exit Ubuntu, and go back to the settings. You should change the number of processors that are available to the user from the default of 1 to the maximum allowed. If your computer has 4 processors, you will be allowed to change it to 2. Since my macbook pro has 8 processors, I set it to 4. This will allow you to change the number of processors when processing vivado to the maximum number, speeding up the process.

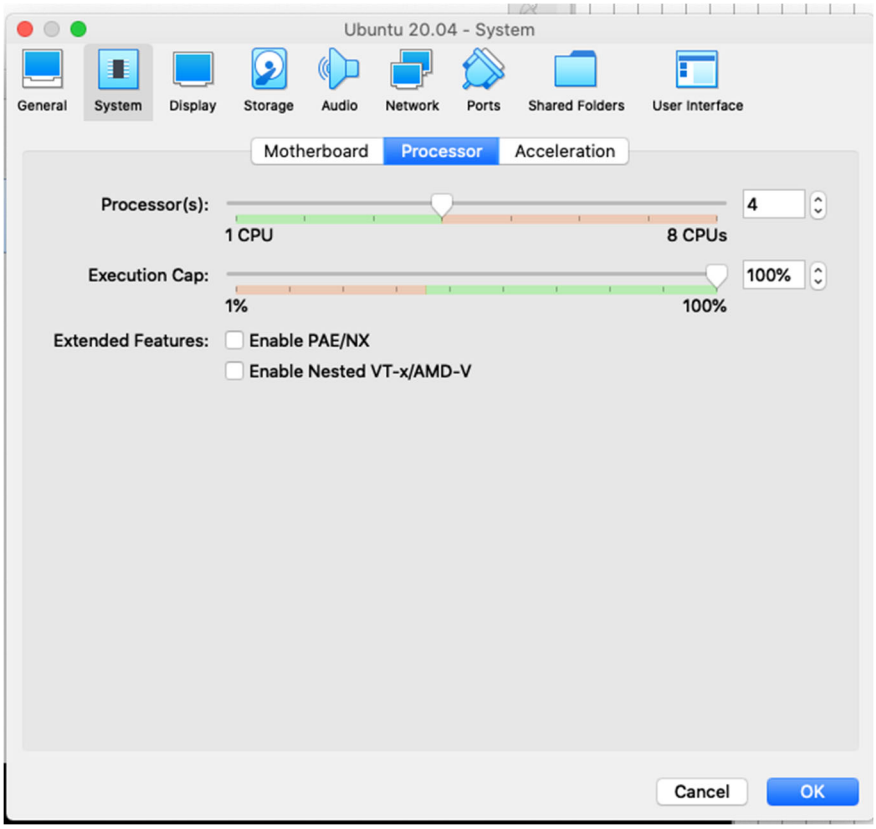

Figure 23 – The mac has 8 processors, my PC lapto has 4, the system will allow you to select half of them to be used in the VM system.

Adding Vivado:

Use Firefox on Ubuntu and go to **https://www.xilinx.com/support/download.html** , and log in using your Xilinx account. If you don't have one, you should sign up, it is free. Select Xilinx Unified Installer 2020.2: Linux Self Extracting Web Installer (356.08 Mbyte)

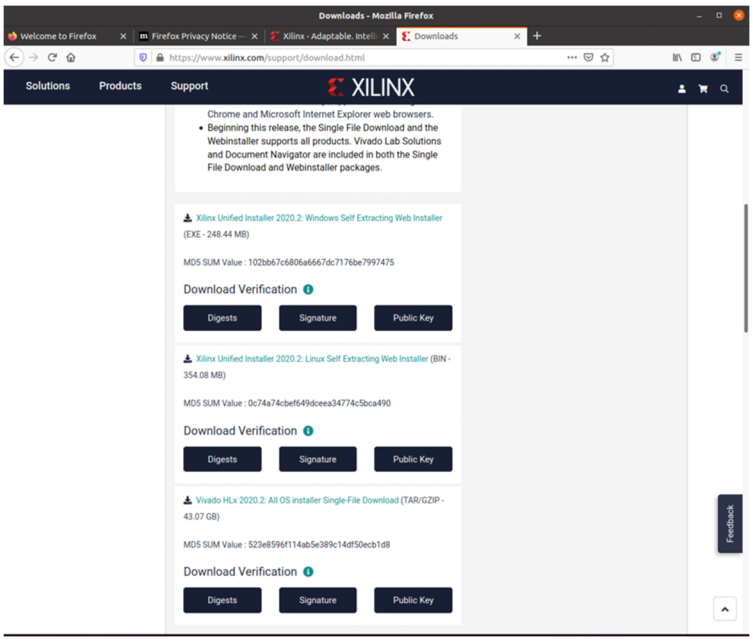

Figure 24 – Select Linux Self Extracting Web Installer. You will have to log in after this window, and then the system will present your Name and Address Verification:

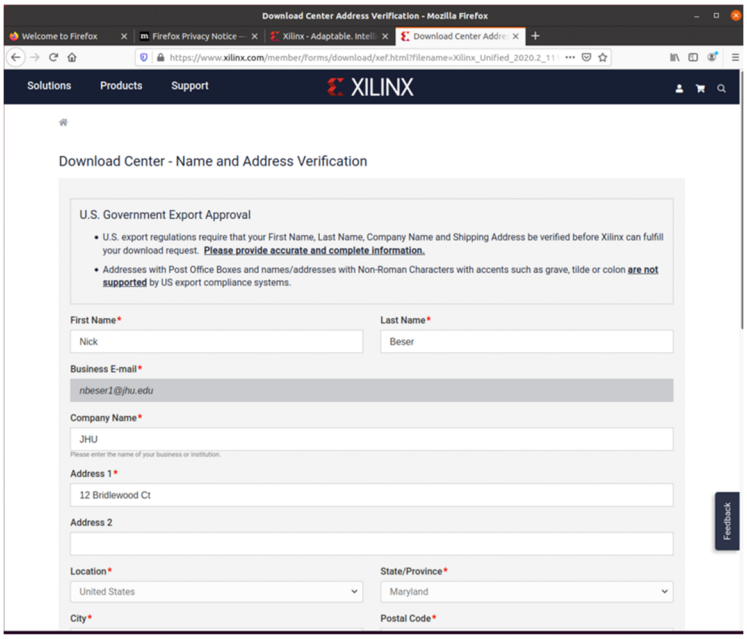

Figure 25 – At the bottom of the window is a download button, Select download.

Once the binary install is loaded into your Downloads directory, change the file status to executable by launching a terminal and:

```
chmod a+x Xilinx_Unified_2020.2_1118_1232_Lin64.bin
```

Then start the installation as root by typing:

```
sudo ./Xilinx_Unified_2020.2_1118_1232_Lin64.bin
```

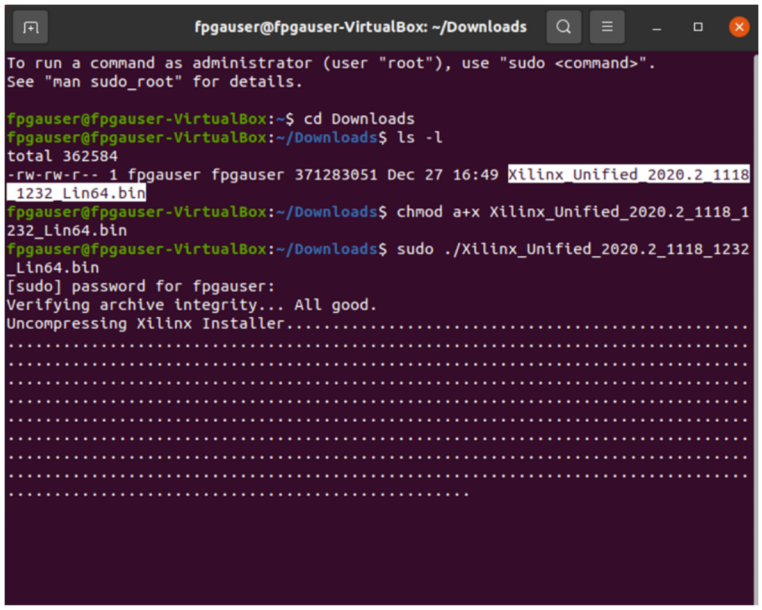

Figure 26 – Changing executable status and launching install as root

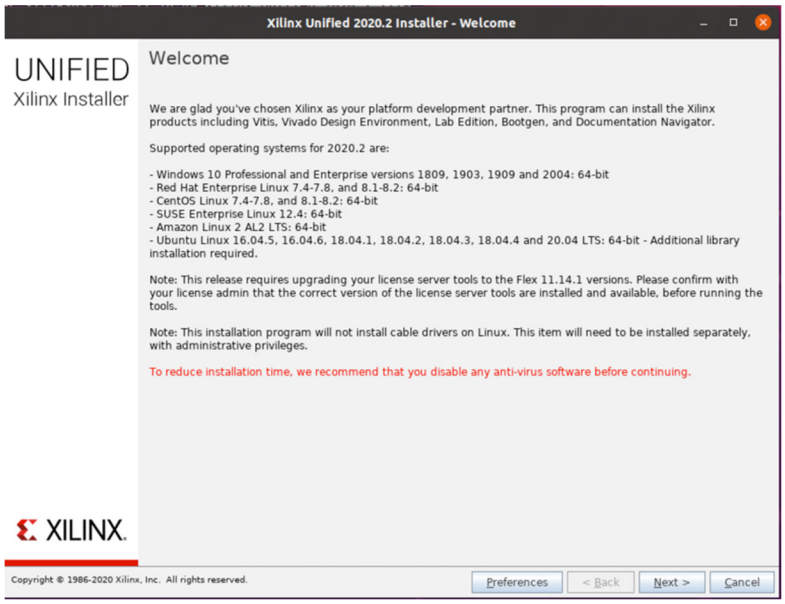

Figure 27 – Xilinx installation Starting

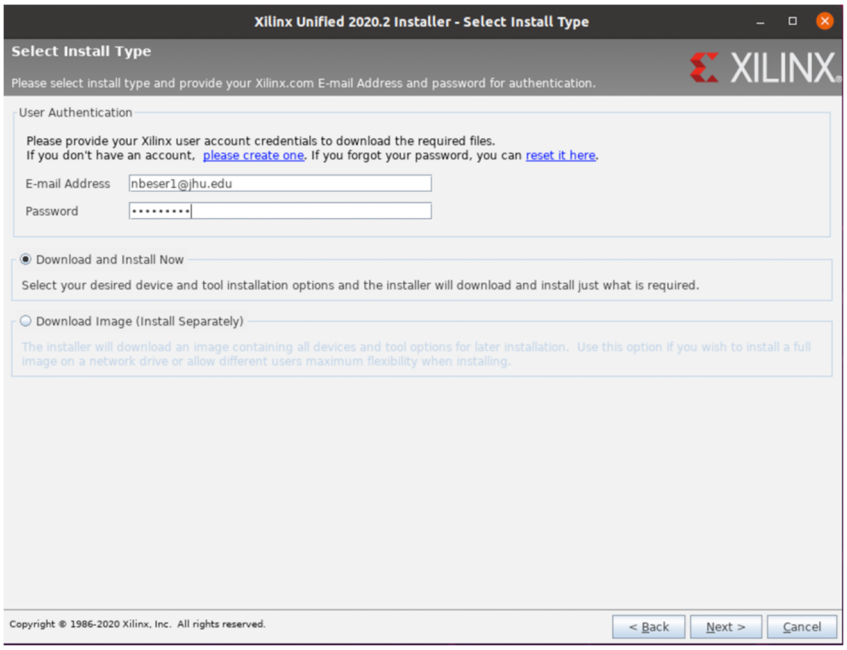

Figure 28 – Log into Xilinx.com to enable download

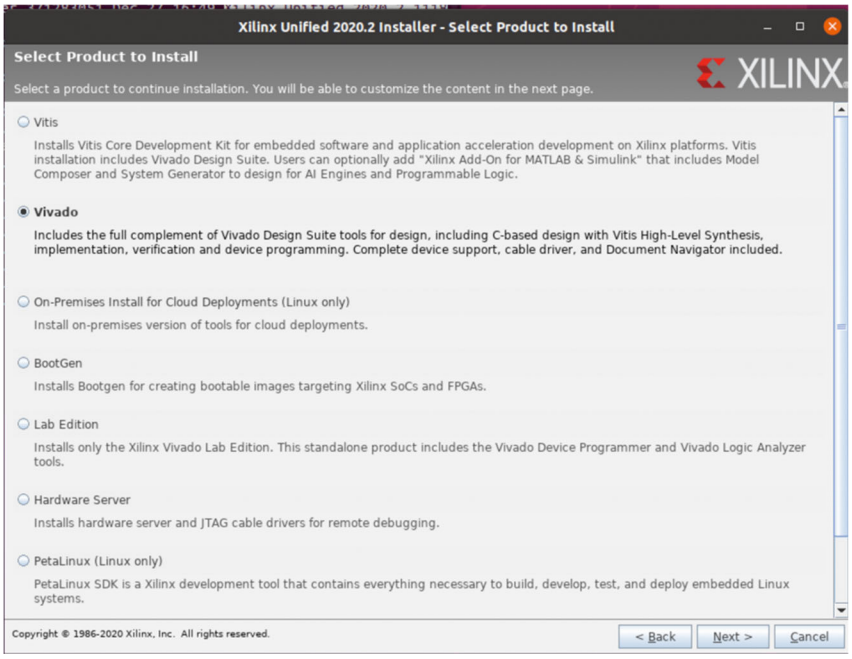

Figure 29. Vitis is very large. Vivado is just 20Gbytes. Select Vivado.

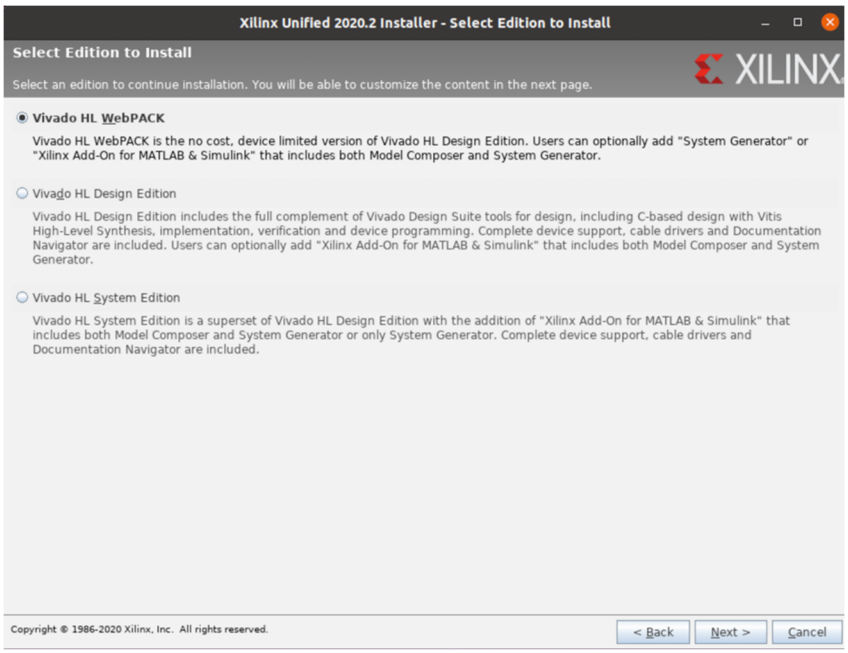

Figure 30 – Select the Vivado HL WebPack license. It's free and does not expire.

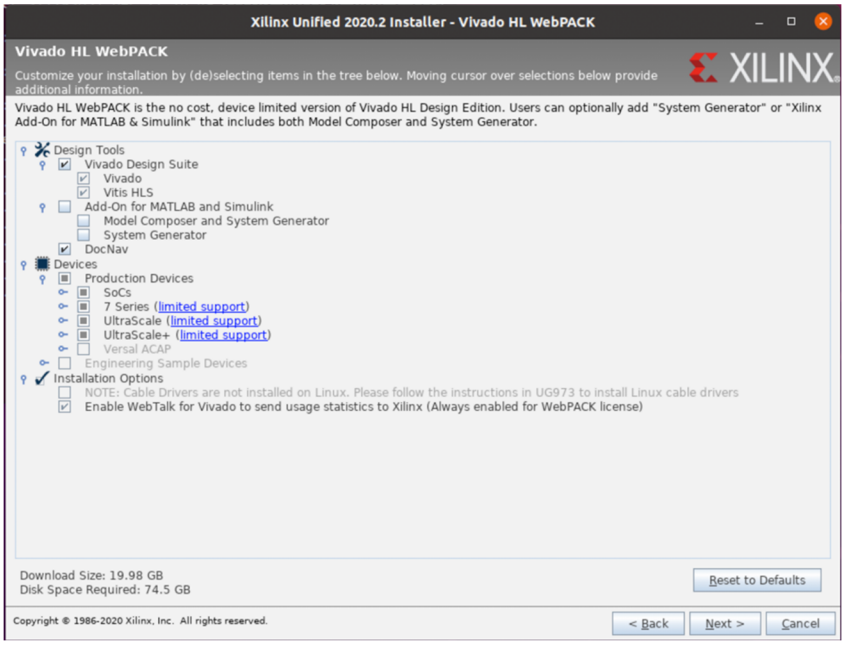

Figure 31 – This window lists what is being installed. As you can see the Vivado installation requires 74.5 GB free disk space.

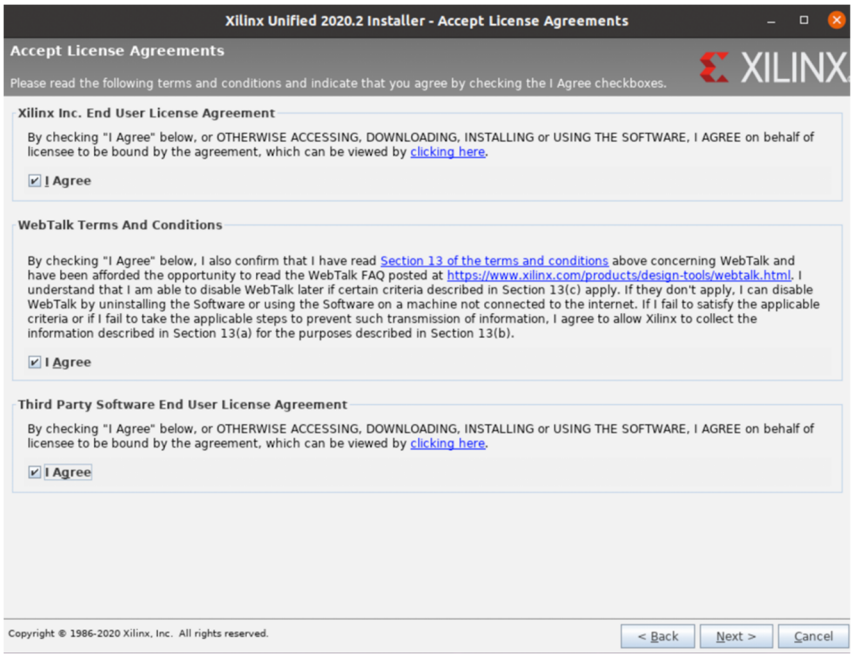

Figure 32 – You must agree to all terms to begin download

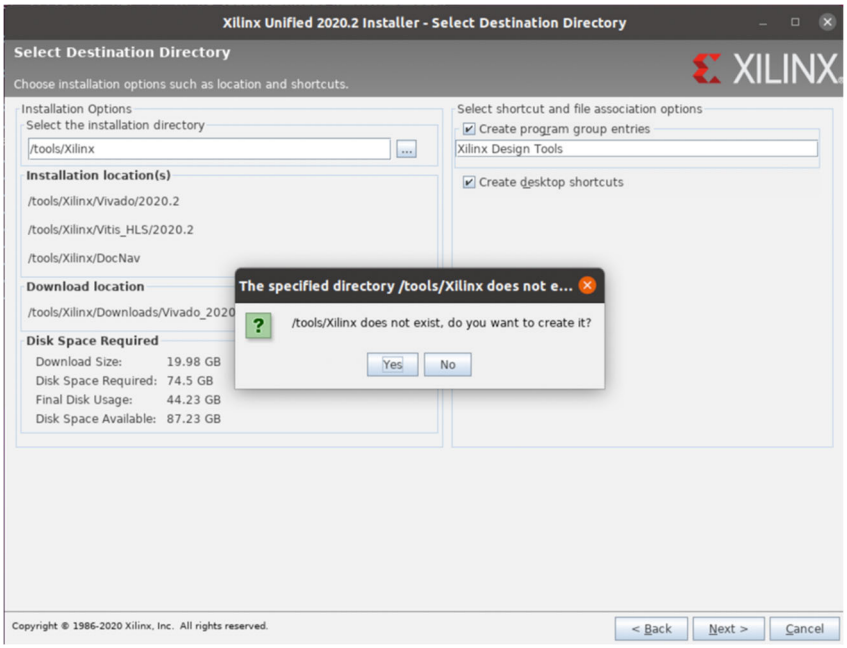

Figure 33 – The install directory will be in /tools/Xilinx. Since you are doing this as root, you should be able to do the install.

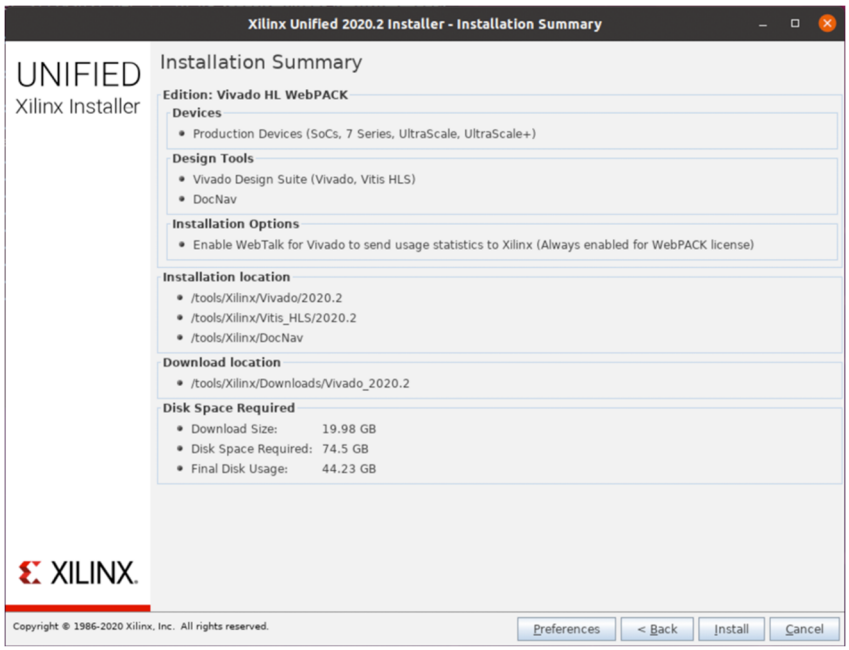

Figure 34 – Final window before install. The following steps will take multiple hours for download depending on your network speed.

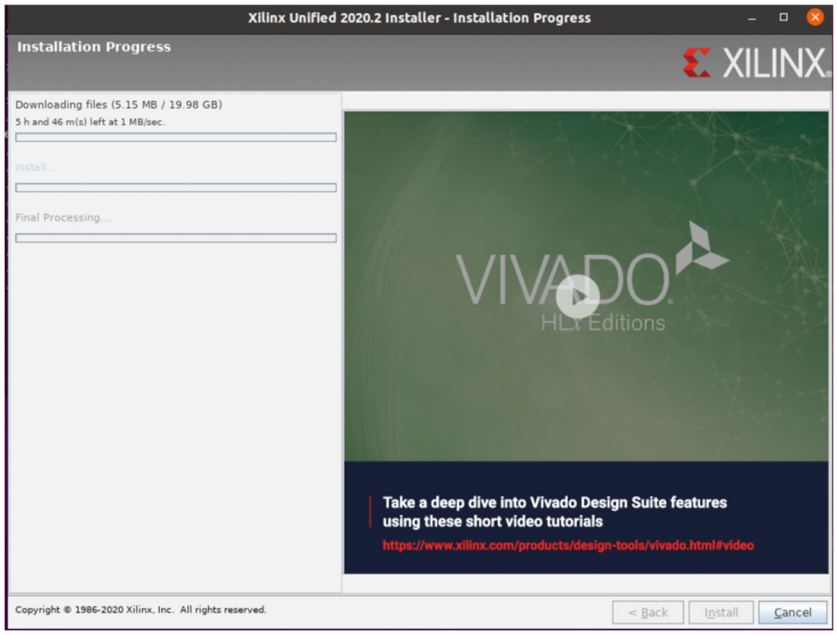

Figure 35 – Although this window says that it will take in excess of 5 hours, it is closer to three hours using my network (multiple users at home streaming), not the best speed.

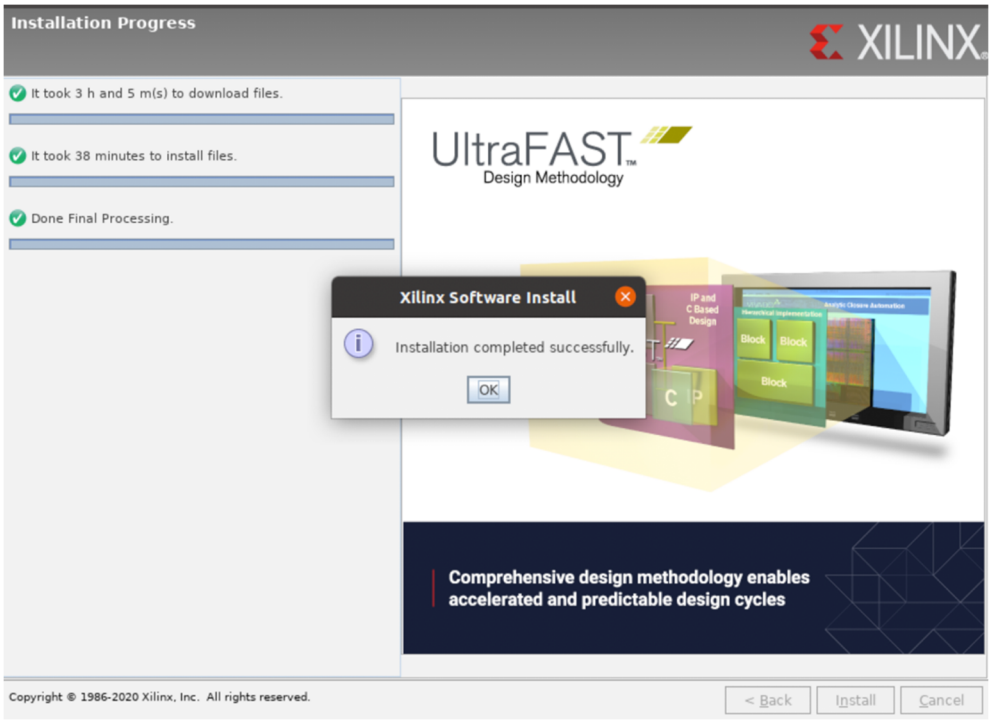

Figure 36 – Vivado is Installed on Linux system.

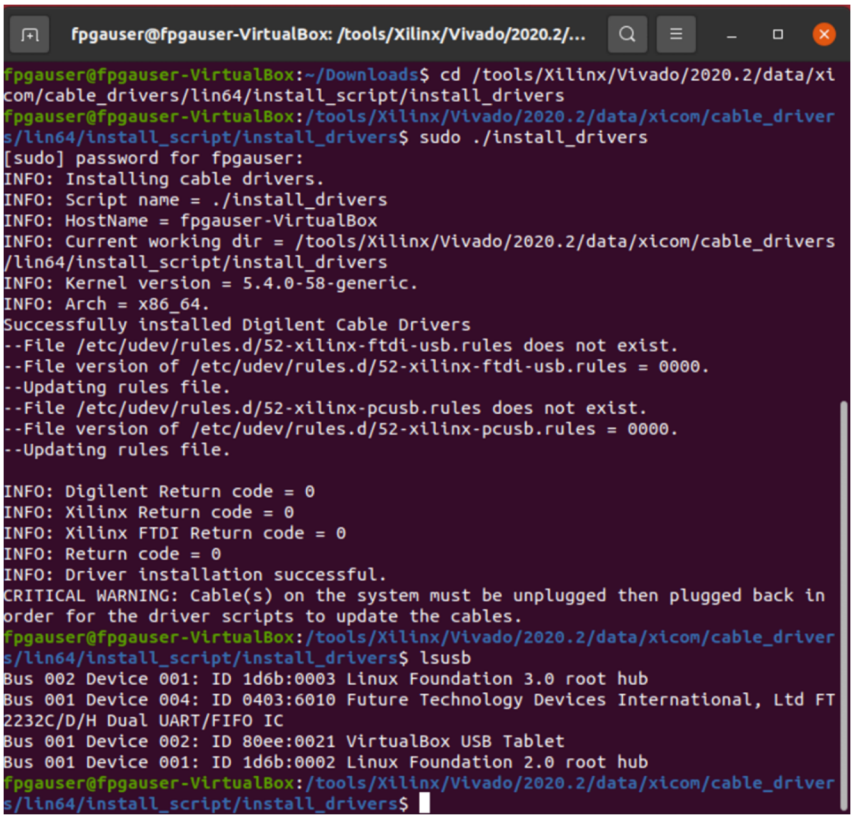

Figure 37 - After Installation of Vivado, go to the directory where the cable drivers are located and run the script for install

The Cable drivers are located at:
cd /tools/Xilinx/Vivado/2020.2/data/xicom/cable_drivers/lin64/install_script/install_drivers
sudo ./install_drivers

Missing library for Viviado
If you attempt to run Vivado just after installing, you may discover that a library is missing from Linux. I don't know why the install for linux does not correct this problem, but there are three command lines that need to be executed from the terminal. You have to be root to perform this task:

```
sudo apt update
sudo apt install libtinfo-dev
sudo ln -s /lib/x86_64-linux-gnu/libtinfo.so.6 /lib/x86_64-linux-gnu/libtinfo.so.5
```

The two lines install the libtinfo library and create a symbolic link to provide libtinfo.so.6 which is required by Vivado.

MIPS-MTI-ELF Assembler and C-Compiler
The RISC processors that we will be studying are based on the MIPS architecture as described by Patterson and Hennessy. The initial processor has a very small instruction set, and as we study the architectures we will add additional instructions. The MTI-ELF toolchain contains an Assembler and C compiler, plus additional utilities that will convert the various output files, link them, and create memory images. The code is available from:
https://codescape.mips.com/components/toolchain/2021.09-01/downloads.html

Download the MTI Bare Metal Toolchain for Linux x64 (303Mbytes). Unarchive it in the Downloads directory. Once it is available, modify the .bashrc file in the login directory to contain the following code at the end of the file. Please correct the location to reflect the actual directory of the archive:

```
export MIPS_ELF_ROOT="/home/fpgauser/Downloads/mips-mti-elf/2021.09-01/bin"
export PATH=$HOME/Downloads/mips-mti-elf/2021.09-01/bin:$PATH
```

Once the two lines have been added to .bashrc file, type:

```
source .bashrc
```

If there are no errors, the MIPS assembler is ready to go.
Note, An older version of the MIPS tools is also available (mips-mti-elf.tar.gz).

Make Utility
Add the make utility to linux by typing:

```
sudo apt install make
```

Git Utility
Add the Git Utility to linux by typing:

```
sudo apt install git
```

Download the mips-cpu.zip from from Module 1 Readings
Once the class is started, we will be using the bitbucket.org site to host our distribution of RISC designs. For now, go to the Blackboard site, Course Content, Module 1 folder, and select mips-cpu.zip from the readings section. This will provide two MIPS designs that were modified from the Li text to run on the Nexys FPGA board. The designs have also been modified to use the PMODUSBUart module from Diligent. I suggest you create a directory called Distro, and unarchive the zip file.

The basic structure of the files are:
mips-cpu
Which will also have two subdirectories:
RTL – Contains the Verilog designs
Software – Contains MIPS code

Assembling a new test routine
Go to the following directory in the mips-cpu respository:
mips-cpu/Software/Assembly/scintcode7segment

Examine the Makefile:
Contents of Makefile:

```
SOFTWARE_ROOT = ..
A_SOURCES = scintcode7segment.S
C_SOURCES =
PROGNAME = scintcode7segment
include $(SOFTWARE_ROOT)/makesystem/parent.mk
```

In a terminal in the scintcode7segment type:

```
make split_i_d_mem
```

This will create two new files: imem.mem (instruction memory text file) and dmem.mem (data memory text file). The two files must be read into the Verilog source file in Vivado. Notice that the make file also generated a listing of the assembly code. This can be used when running Vivado simulator to debug the program.

In order to test the USB device, to to the other directories (scintcodeSwitchLED7segment, and make the MIPS code in the same was as the first program.

Testing the Nexys Board Interface
Connect the Nexys board USB cable to the host computer. From a terminal in linux type:

```
lsusb
```

You should see the following output on the terminal as shown in Figure 37. If you also plug the pmodUSBUart module into the JB port (top pins) and connect the USB cable to your computer, you will see two Future Technology Devices International ports when you type lsusb from a terminal. You may need to go to the Devices menu option and select the two USB devices before Virtual Box see's the ports.

Running VIvado

I create a script file to run vivado since I run it as root:

```
sudo /tools/Xilinx/Vivado/2020.2/bin/vivado
```

I have been told that you do not have to run Vivado as root. There is a shell script located in /tools/Xilinx/Vivado/2020.2/settings64.ch that when run in the .bashrc file, will allow you to run vivado as a general user.

I also set the file to be executable by:

```
chmod a+x Viv2020.sh
```

Start Vivado by typing:
```
./Viv2020.sh
```

Once Vivado starts, go to the following directory (assuming that you put the repository in Documents):

```
mips-cpu/RTL/CPUs/Single Cycle with interrupts
```

Click on sccompintFPGA.xpr to open the single cycle with interrupts design. Since the project was created with an earlier version of Vivado, you will get a message that it has to be converted to the current version. You will also get a Project Upgraded message. I usually select Report IP Status. The conversion should be automatic; however we will need to update the instruction and data memory files.

Open the Project Manager, and click on mfp_nexys4_ddr file. We need to open the sc_interrupt.v file to update the links. Scroll to lines 70 and 71 and you will see the parameter IMEM_FILE and parameter D_MEM_FILE definitions. They need to be updated to reflect the newly assembled code for the scintcode7segment routine. Note, this is the most common mistake most people make when updating programs. The exact directory location must be given.

The new lines should be:

```
parameter IMEM_FILE = "/home/fpgauser/Documents/mips-cpu/Software/Assembly/7SegmentExamples/imem.mem";
parameter DMEM_FILE = "/home/fpgauser/Documents/mips-cpu/Software/Assembly/7SegmentExamples/dmem.mem";
```

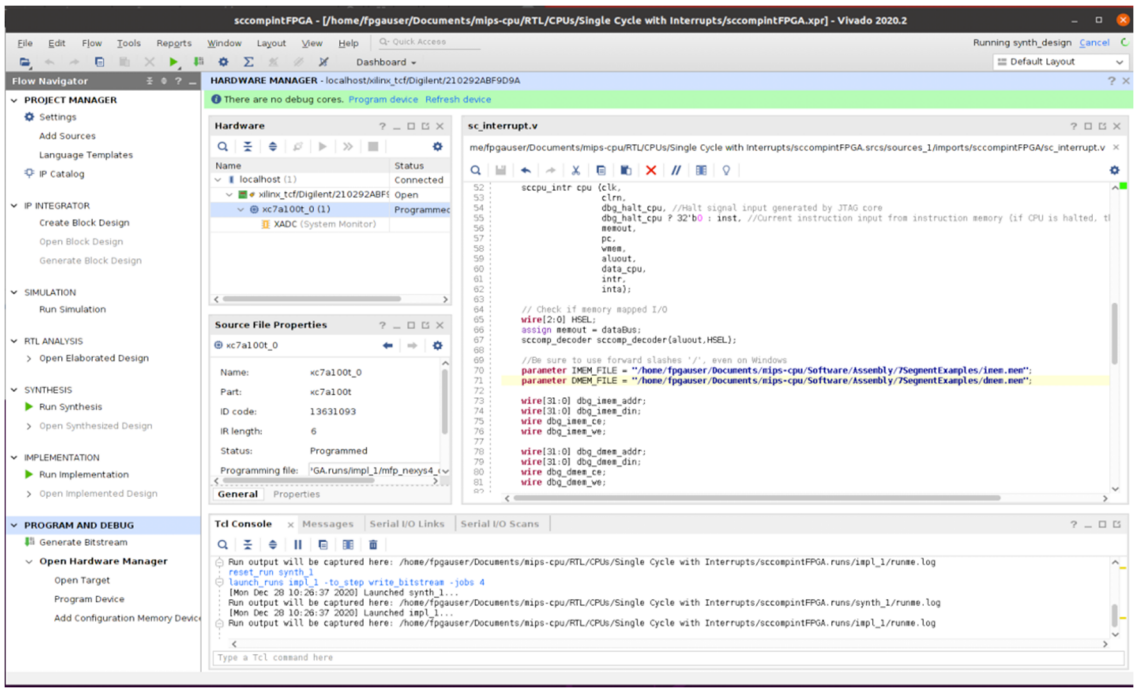

Figure 38 – Update the sc_interrupt.v file with the correct location of the imem.mem and dmem.mem files. Save the file.

Click on Generate Bitstream. Since this is the first time this has been run, Vivado will go through the entire process, synthesis, implementation and layout, generating the bit stream. The launch menu will provide an option to set the number of jobs. Set it to the maximum number of processors to speed up the compilation.

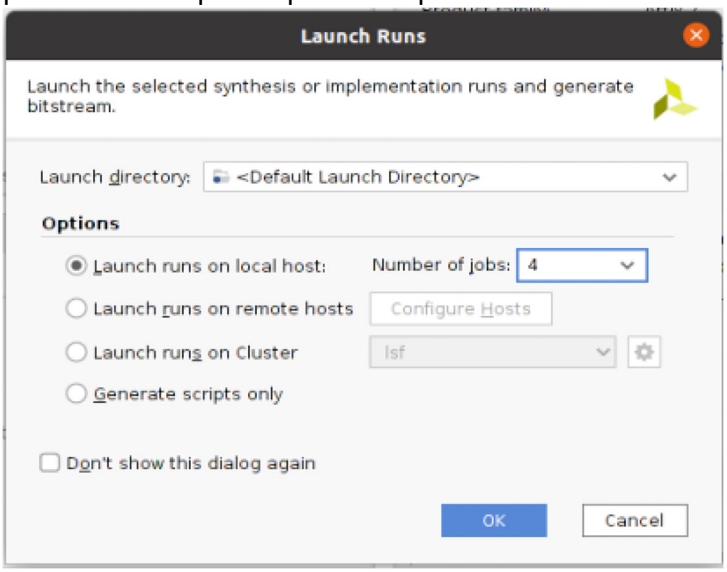

Figure 39 – This setting will allow 4 CPU's to work on the jobs in parallel.

After Vivado completes the run, it will produce a window with the option to Open Hardware manager:

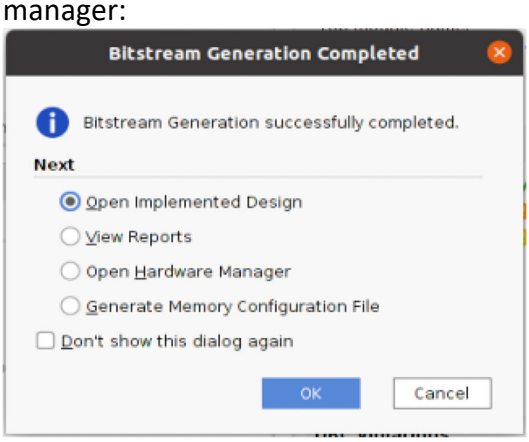

Figure 40 – Select Open Hardware Manger to load the Nexys board.

Make sure the Nexys board is on and connected to the computer. Select open target, and then auto connect. The system should return with the Digilent board visible, and Program Device as an option to be selected:

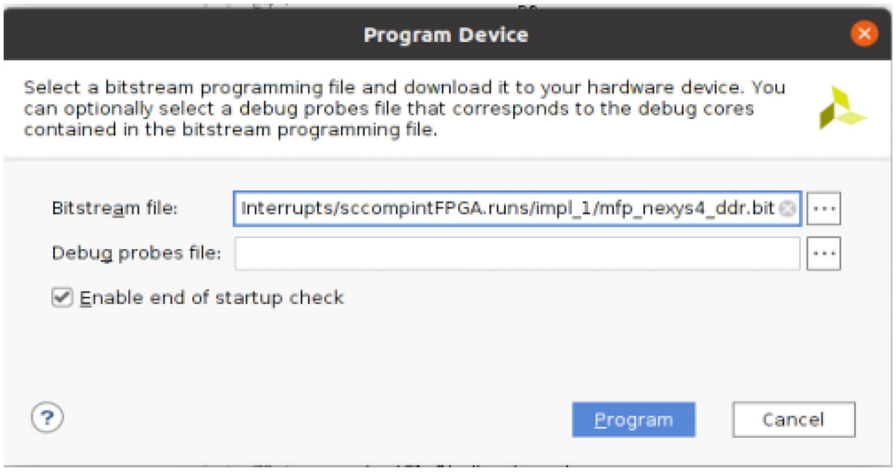

Figure 41 – Loading the bitstream to the Nexys Board

After loading the bitstream the board should produce:

Figure 42. The Nexys 4 DDR board running the single cycle with interrupts MIPS assembly code. Notice the pmodUSBUart module plugged into the top pins of JB. Connector JA has the pmodALS board (light detector), and Connector JC has the pmodOLEDrgb module. Connector JD has the speaker wired into pins 1 and 6 (signal and power).

PMODUSBUart programming

Once the MIPS design has been bit streamed into the FPGA board, the debug interface Verilog code can be accessed by a serial port to halt the processor, load and verify the programming memory, and then restart the processor. This will allow you to change the code without forcing a reprogramming of the FPGA.

In order to use the board, you will need python3. Along with Python 3, you should install pip3 so that any additional python libraries can be added.

Step 1: Program the FPGA (This was performed in the previous section).

Step 2: Install the PIP package for Python 3

The fastest way to do this on modern Ubuntu, which is recommended for this class, is to run

```
sudo apt-get install python3-pip
```

which will install pip and any dependencies for you.

Some python installs already come with pip, so you can try running "pip --version" or "pip3 --version" to see if you already have it.

For example:

```
$ pip3 --version
pip 20.2.4 from /usr/lib/python3.9/site-packages/pip (python 3.9)
```

Step 3: Install the required prerequisites

In this zip file, navigate to mips-cpu->Software->UART, e.g. "cd mips-cpu/Software/UART"
Now run pip against the requirements file provided by running "pip3 install -r requirements.txt":

```
$ pip3 install -r requirements.txt
Defaulting to user installation because normal site-packages is not writeable
Collecting intelhex
  Using cached intelhex-2.3.0-py2.py3-none-any.whl (50 kB)
Collecting pyserial
  Using cached pyserial-3.5-py2.py3-none-any.whl (90 kB)
Successfully installed intelhex-2.3.0 pyserial-3.5
```

You may be directed to add /home/fpgauser/.local/bin to the PATH (in .bashrc)

Step 4: Identify your USB-UART tty device

This step can be slightly tricky given our FPGA board and USB-UART adapter use a similar USB-UART chip, making identification harder.

One easy way to find the linux dev device (/dev/ttyUSB*) associated with the adapter is to unplug its USB cable, then plug it back in. You can then run "dmesg" to see what the kernel did with the device. For example on my system, reattaching the device resulted in:

```
$ sudo dmesg
<lots of text>
[95713.162977] usb 1-8: new full-speed USB device number 12 using xhci_hcd
[95713.511860] usb 1-8: New USB device found, idVendor=0403, idProduct=6001,
bcdDevice= 6.00
[95713.511863] usb 1-8: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[95713.511864] usb 1-8: Product: FT232R USB UART
[95713.511866] usb 1-8: Manufacturer: FTDI
[95713.511867] usb 1-8: SerialNumber: A904CXD3
[95713.534864] ftdi_sio 1-8:1.0: FTDI USB Serial Device converter detected
[95713.534882] usb 1-8: Detected FT232RL
```

```
[95713.542908] usb 1-8: FTDI USB Serial Device converter now attached to
ttyUSB0
```

This last line in the log tells us Linux has placed our device at /dev/ttyUSB0, which we will use in the next step.

In order to talk to the USB port from the python code, you also need to add your account to the dialout group. In linux, issue the following command (substituting your user name where I have fpgauser):

```
sudo usermod -a -G dialout fpgauser
```

Step 5: Download the software

The support files used for the make operation will automatically generate both the .mem files used by Vivado to load the code into the FPGA prior to simulating or building the bitstream file. The utility will also generate a intel Hex file that is used by the download python script to load the program through the USBUart interface. download.py is the only step you will need to routinely perform after loading an FPGA bitstream.

In the mips-cpu->Software->UART directory as before, there is a script "download-ihex.py" that interprets Intel Hex formatted data files and sends its content to the CPU over UART. The provided "single_cycle_sw2led.ihex" is a tiny program for the Single Cycle CPU which takes the 16 switch positions and sets the corresponding LEDs.

After the CPU bitstream is loaded on the FPGA, this program can be downloaded by running "./download-ihex.py /dev/ttyUSB0 single_cycle_sw2led.ihex" from the UART directory, making sure to substitute "ttyUSB0" for your appropriate device if different. Notice that the ihex file in this example is also in the same directory as the download-ihex.py file.

An example of this is:

```
$ ./download-ihex.py /dev/ttyUSB0 single_cycle_sw2led.ihex
h
i
a 00000000
w 3c0cbf80
a 00000004
w 358d0004
a 00000008
w 8daa0000
a 0000000c
w ad8a0000
a 00000010
w 1000fffd
a 00000014
w 00000000
e
g
```

The output of the download script can be interpreted as:

- h – halt
- i. – instruction (imem)
- a – address
- w – Write
- r – Read
- d – Data (dmem)
- e – Cmd Reset
- s – step
- g – go

The python script will read the intel hex file and output correctly to each memory model. If all goes well, the LEDs will take on the switch positions. Internally, the download mechanism is a simple command set sent to a monitor state-machine in the fpga (cmdproc.v) which takes characters like "w 3c0cbf80" and converts them into bus cycles to write data into RAM.

The entire stack of Python and Verilog is open-source and reasonably small, so poke around and see how it works.

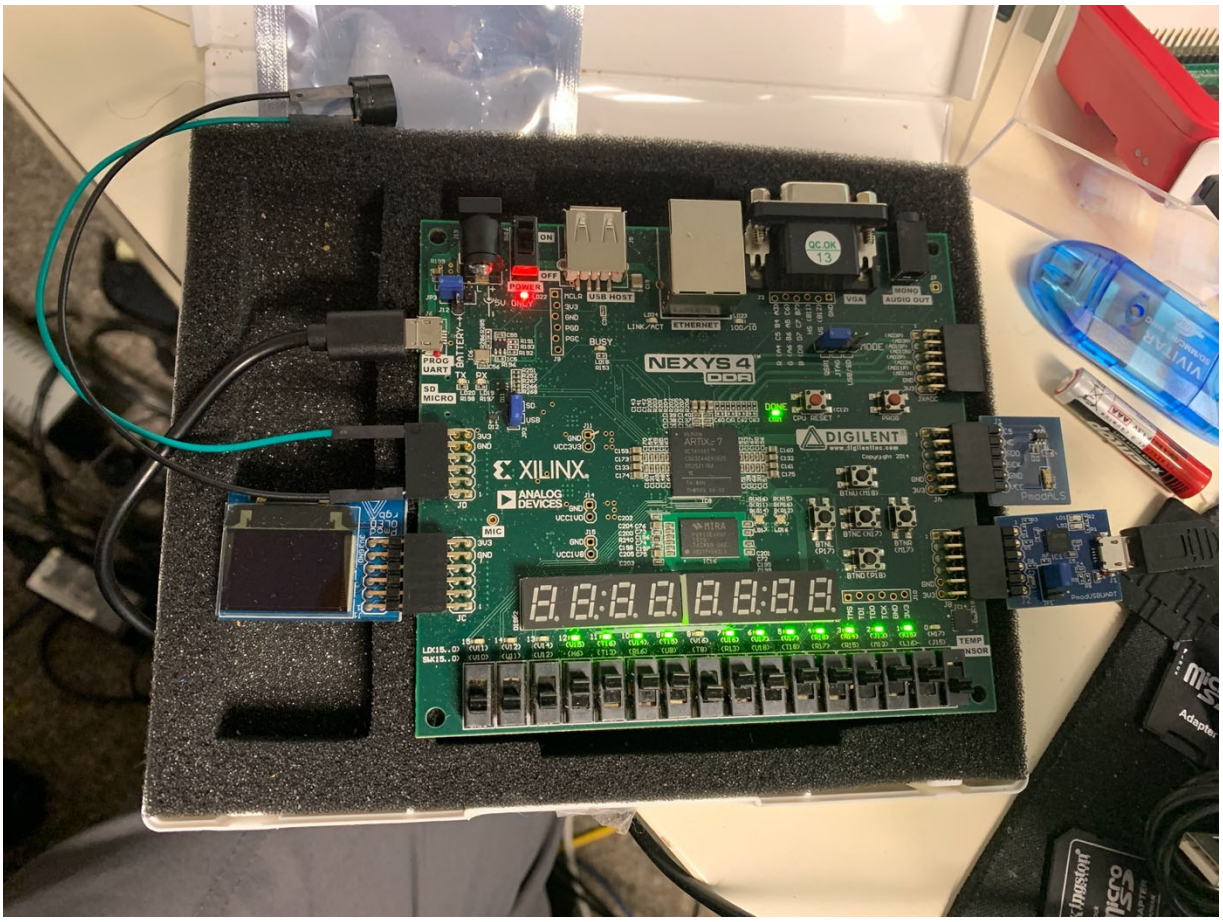

Figure 43 – The Nexys FPGA running the switch to LED program, downloaded using the USB Uart.