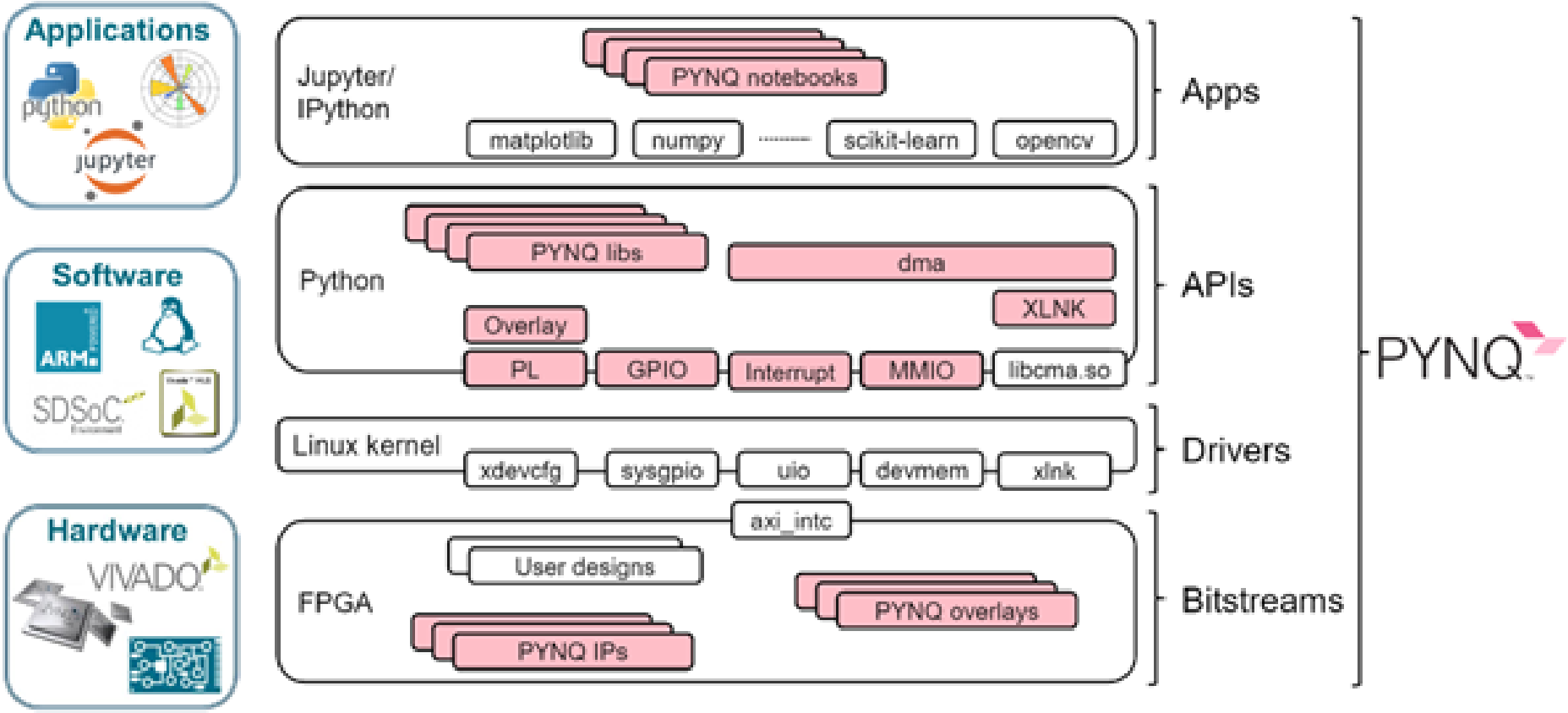
香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

Lab10: PYNQ Exercise

AXI IP Block Design Diagram



Lab10 Outline

- **PART 0: Hardware Setup**
- **PART 1: Flash Image File into SD Card**
- **PART 2: Write Python Code**

① Example

② Task

- **Appendix (Optional)**

Lab10 Outline

- **PART 0: Hardware Setup**
- **PART 1: Flash Image File into SD Card**
- **PART 2: Write Python Code**

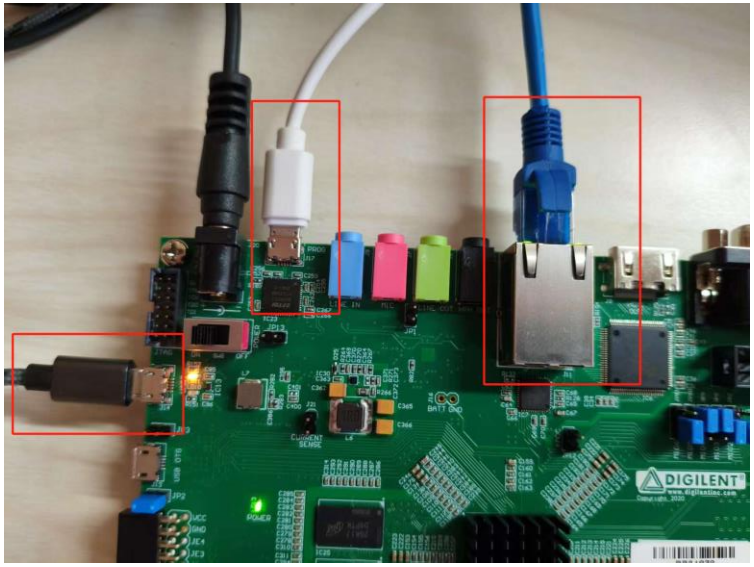
① Example

② Task

- **Appendix (Optional)**

Hardware Setup

- **In this lab, you need to connect two micro-b USB wires and the network cable to the Zedboard.**
  - **Connect the JTAG port on the top. This port is used to program the hardware bitstream.**
  - **Connect the UART port under the power switch. This port is used to program the software binary to the board and communicate with the software program.**
  - **Establish an ethernet between the computer and the Zedboard with network cable.**

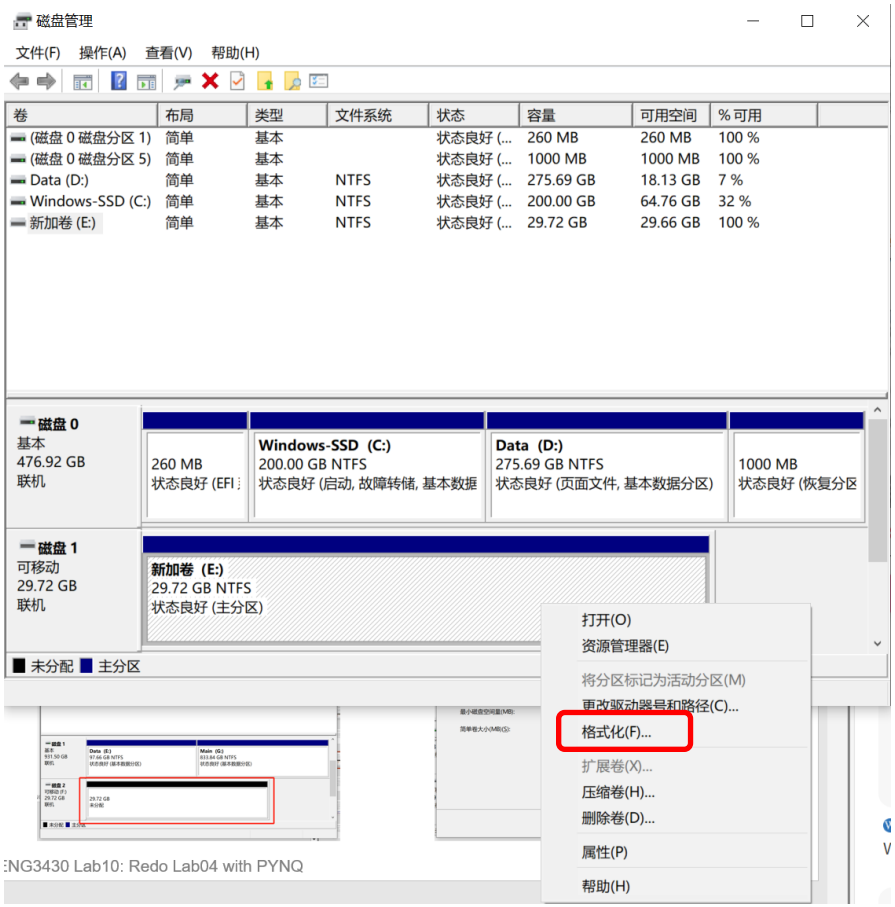


Lab10 Outline

- PART 0: Hardware Setup
- **PART 1: Flash Image File into SD Card**
- PART 2: Write Python Code
  - ① Example
  - ② Task
- Appendix (Optional)

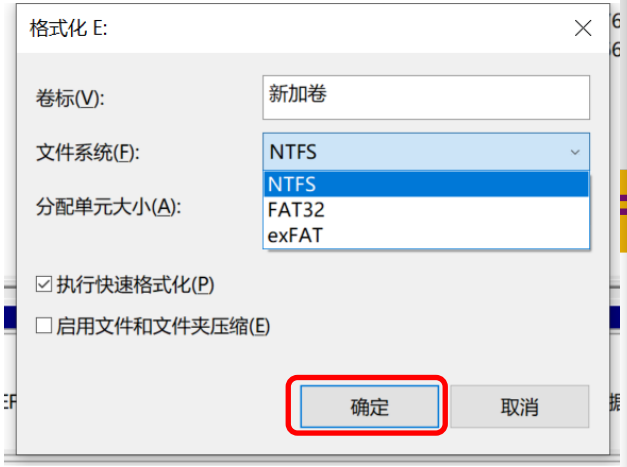
Flash Image file into SD Card

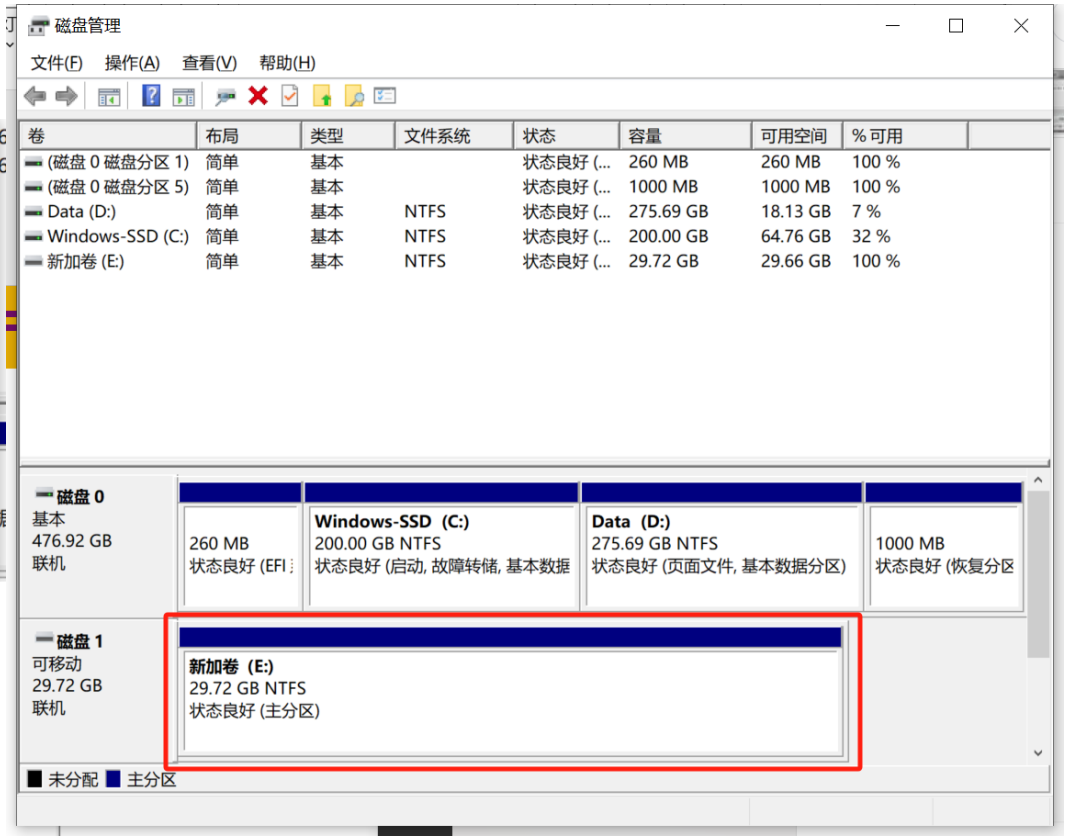
- Insert **SD card** into your computer
- Open **Disk Manager**
- Click **Format Hard Disk**



Flash Image file into SD Card

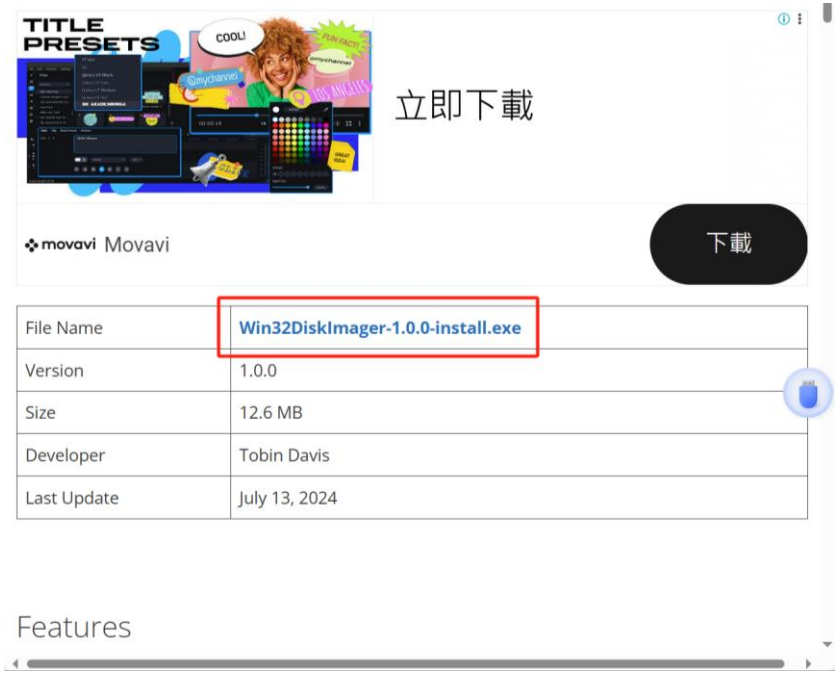
- Choose **NTFS** as file system
- Click **Confirm**
- You should find a **new volume** with almost 30GB

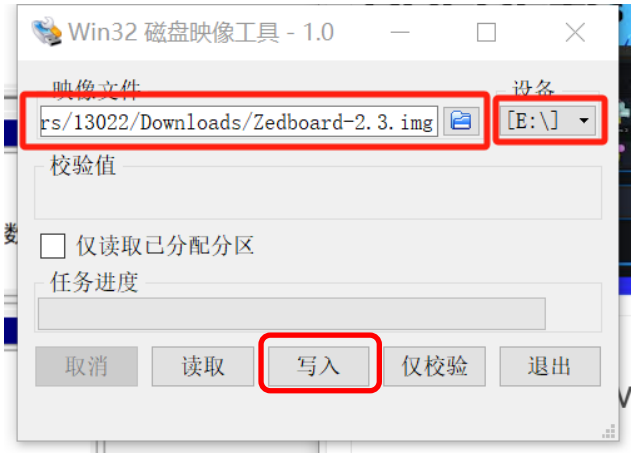


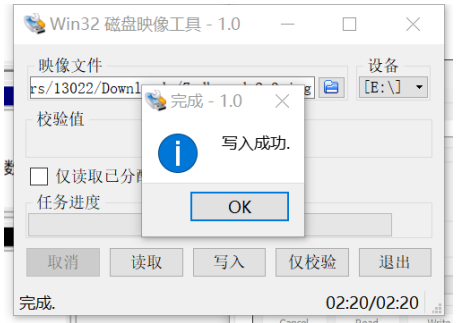


Flash Image file into SD Card

- Download and open **Win32DiskImager**
  - https://win32diskimager.org
- Select Zedboard-2.3.img as **image file**
- Select the new volume for **SD card** as **equipment**
- Click **Write** and wait for 3-5 mins







Lab10 Outline

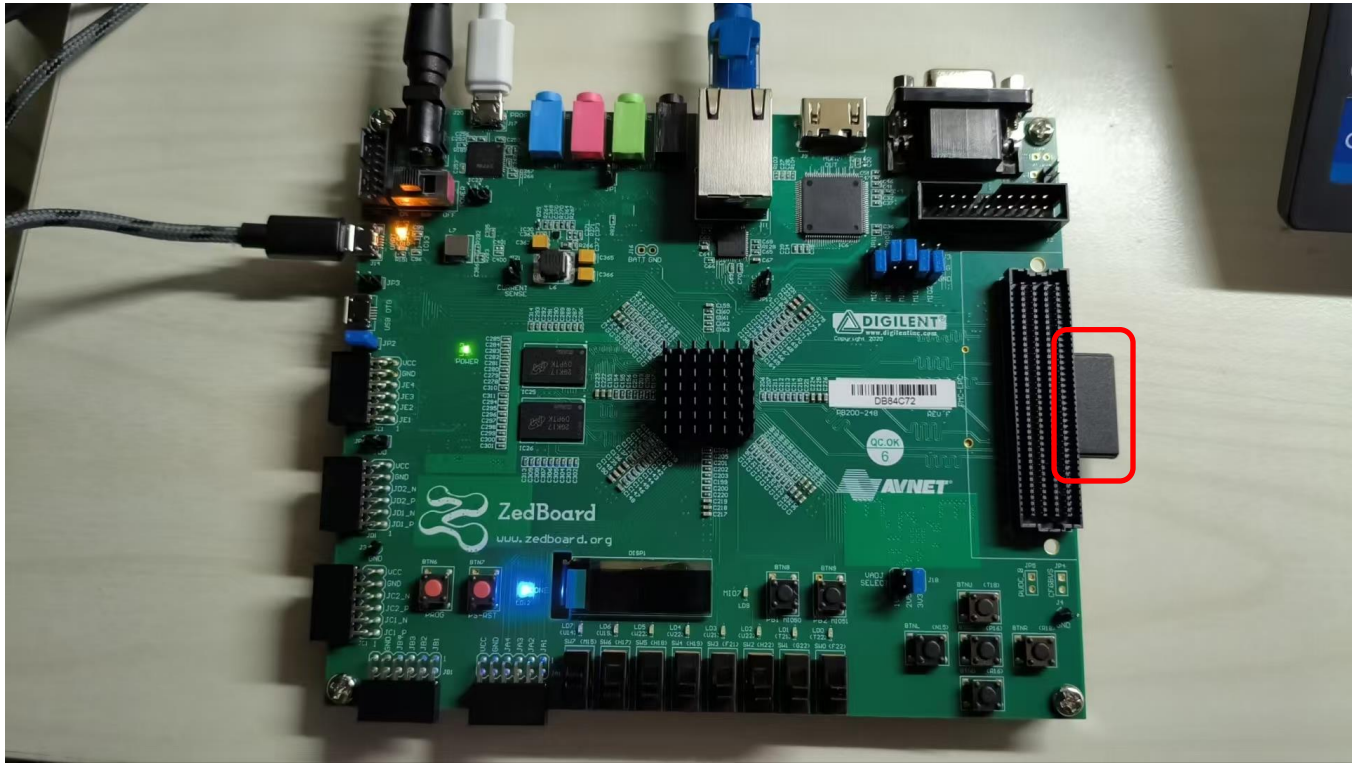
- PART 0: Hardware Setup
- PART 1: Flash Image File into SD Card
- **PART 2: Write Python Code**

  ① Example

  ② Task
- Appendix (Optional)

Hardware Setting

- Insert the **SD card** into card slot in the Zedboard
- Off the power switch
- **LD11** and **LD12** will be turned on

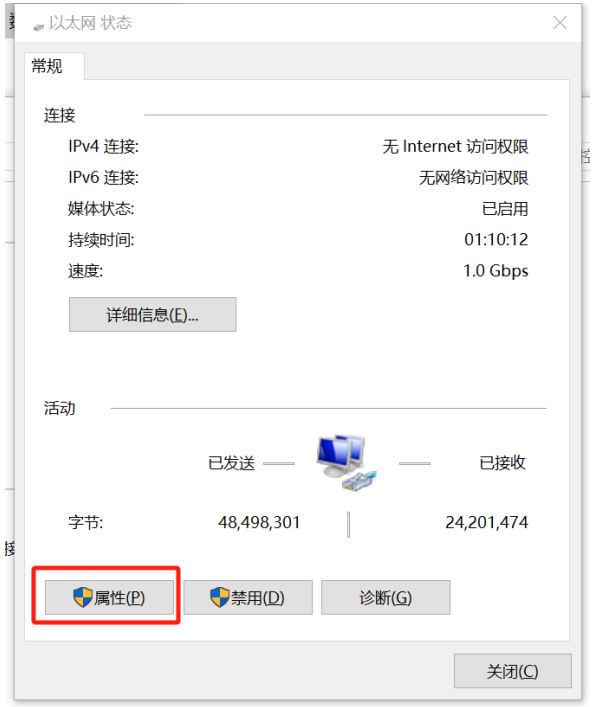


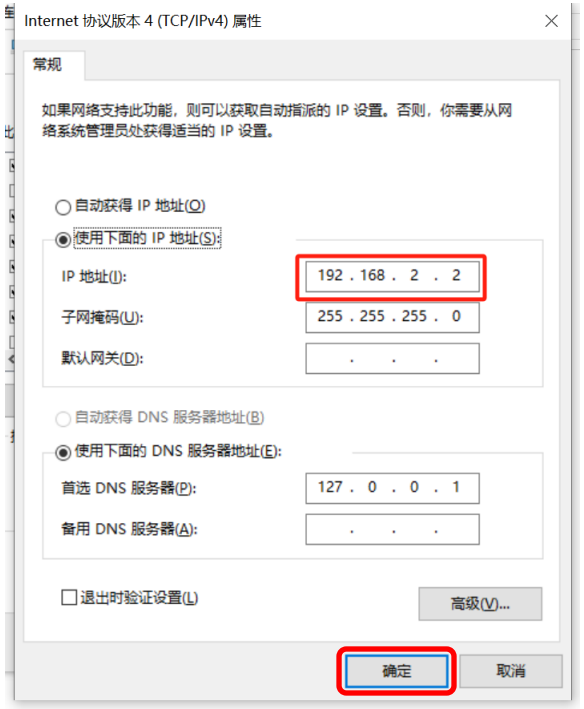
Ethernet Configuration

- Open **Network and Sharing Center**
- **Select Ethernet**

Ethernet Configuration

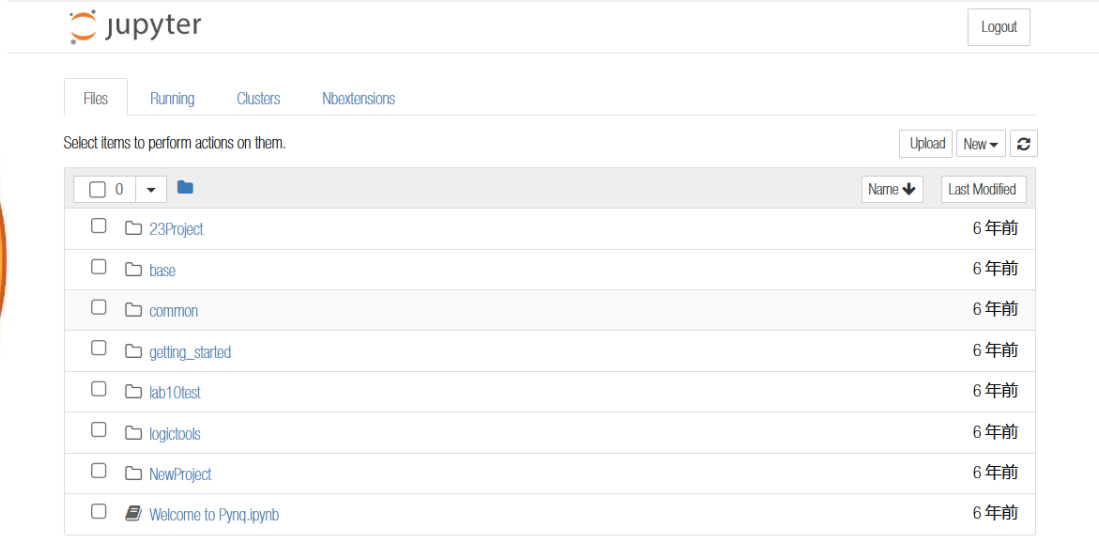
- Open **Properties**
- Select **TCP/IPv4**
- Set **IP address** to **192.168.2.2** and Confirm





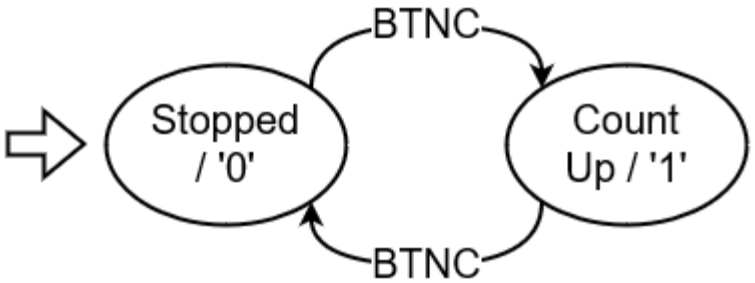
Open Jupyter

- Go to 192.168.2.99 in your browser
- Password is Xilinx
- Now you can program in Python



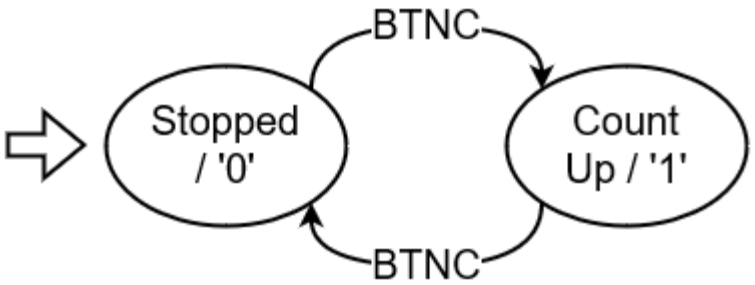
Stopwatch Behavior

- **Behavior**
  - **Two states**
    - **Stopped State** (Initial State)
      - Select the value using **SW0 ~ 3**
      - Display the value in binary format through **LED0 ~ 3**
    - **Count-Up State**



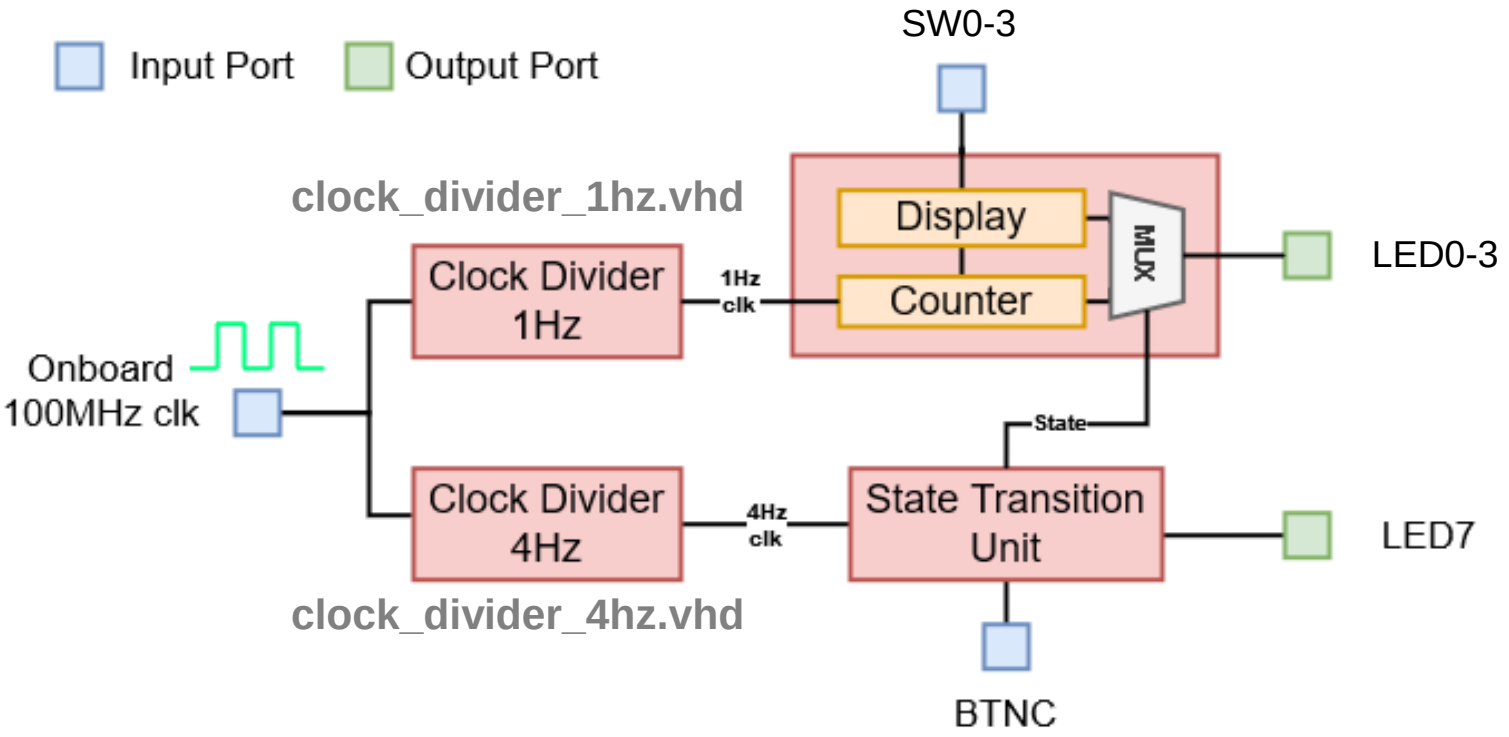
Stopwatch Behavior

- **Behavior**
  - **Two states**
    - **Stopped State**
    - **Count-Up State**
      - Count up from the value selected in Stopped State in 1Hz
      - If the value reaches “1111”, jump to “0000” and keep count up



Stopwatch Behavior

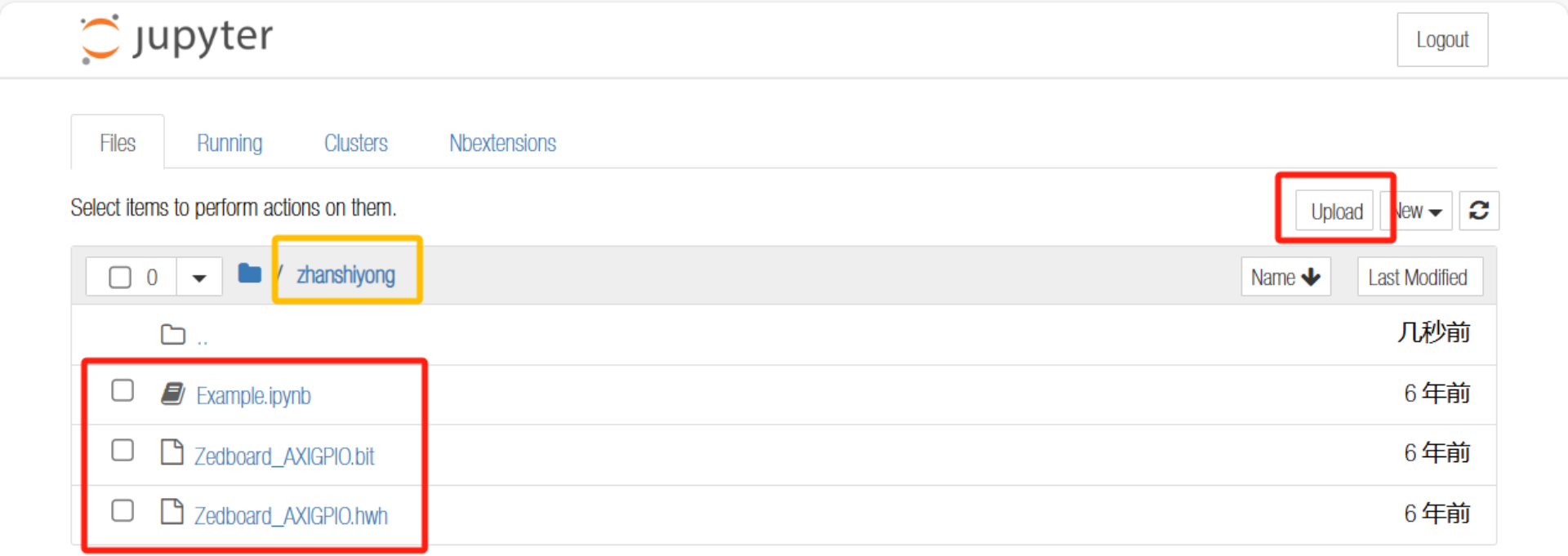
- **Logic Schematic**



lab04.vhd

Upload .bit, .hwh Files, and Ex. Code

- Download Zedboard_AXIGPIO.bit, Example.ipynb, and Zedboard_AXIGPIO.hwh from Blackboard
- Create a new folder and upload them
- Open Example.ipynb



Example Code (Part 1)

```
from pynq import Overlay
import time
class FSMController:
    def __init__(self, bitfile_path):
        self.ol = Overlay(bitfile_path)
        self.leds = self.ol.led.channel1
        self.switches = self.ol.sw.channel1
        self.btns = self.ol.btn.channel1
        self.state = "0"
        self.counter = 0
        self.last_1hz = time.time()
        self.last_4hz = time.time()
        self.last_btn_press = time.time()
        self.btn_pressed = False
        self.update_leds()

    def update_leds(self):
        state_val = int(self.state, 2) << 7 #State Position
        count_val = self.counter & 0x0F #0-3 led
        led_value = state_val | count_val
        self.leds.write(led_value, 0xFF)
        print(f"{bin(led_value)[2:].zfill(8)}")

    def read_inputs(self):
        current_time = time.time()
        self.last_btn_press = current_time
        btn_val = self.btns.read()
        return {
            'din': self.switches.read() & 0x0F,
            'btn_c': (btn_val & 0x01) == 0x01,  # bit0
        }
```

Declare Overlay

Set I/O Signals

Set Time and State-transition Variables

Just as Embedded System
☺ Programmer Friendly

Update led output with xxx.write()

Read sw and btn input with xxx.read()

Example Code (Part 2)

```
def run(self):
        while True:
            current_time = time.time()
            inputs = self.read_inputs()
            if current_time - self.last_4hz >= 0.25:
                self.last_4hz = current_time
                btn_c = inputs['btn_c']
                if btn_c and not self.btn_pressed:
                    if self.state == "0":
                        self.state = "1"
                    elif self.state == "1":
                        self.state = "0"
                    self.btn_pressed = True
                elif not btn_c:
                    self.btn_pressed = False
            if self.state == "0":
                self.counter = inputs['din']
            elif self.state == "1":
                if current_time - self.last_1hz >= 1.0:
                    self.last_1hz = current_time
                    if self.counter < 15:
                        self.counter += 1
                    else:
                        self.counter = 0
            self.update_leds()
            time.sleep(0.5)

if __name__ == "__main__":
    controller = FSMController("./design_1.bit")
    controller.run()
```

Check BtnC for every 0.25 sec.

State Transition

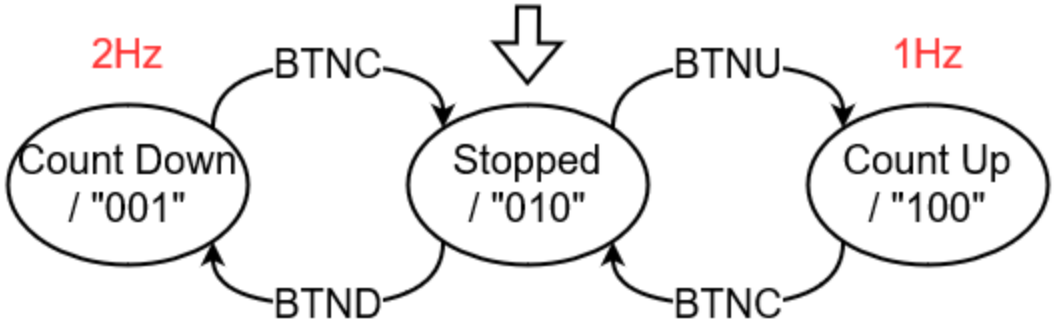
Avoid Oscillation

Behavior

Declare .bit file

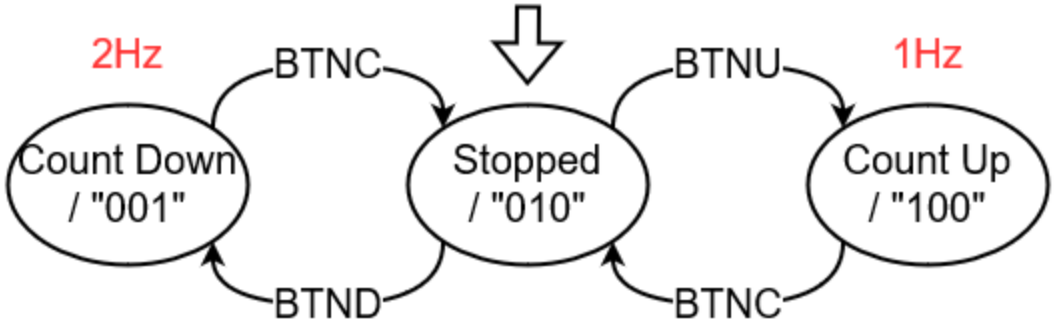
Three-state Stopwatch

- Compare to the two-state stopwatch in example
  - **Stopped state (“010”)** is the same
  - **Count Up state (“100”)** is no longer loop back
    - If the counter reaches “1111”, stops at “1111”
  - **Count Down state (“001”)**
    - Count down the value selected by the stopped state in 2Hz
    - If the counter reaches “0000”, stops at “0000”



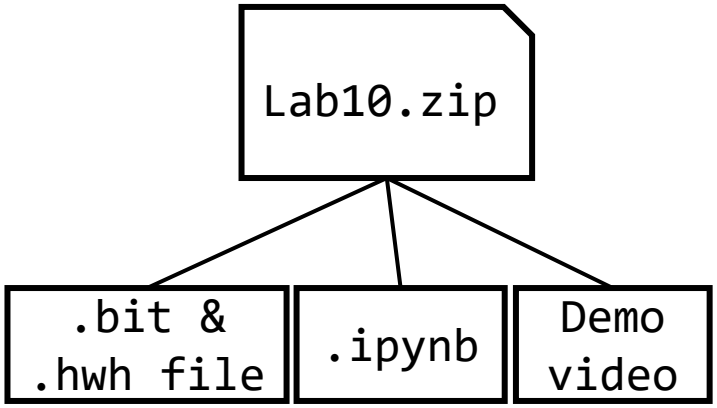
Three-state Stopwatch

- Compare to the two-state stopwatch in example
  - **LED 5-7** for showing the new state values (“001”, “010”, “100”)
  - **BTNU and BTND** are added for state transition
    - BTNC will no longer switch back from Count Up state to Stopped state
    - Non-specified transition will create no effects
      - E.g., Pressing BTND in Count Up state has no effect



Submission Guide

- Following the question specification to implement the Three-State Stopwatch
- Demo video instruction
  1. Select 1010b and press BTNU
  2. Wait until 1111b and press BTND
  3. Then press BTNC
  4. Select 1000b and press BTND
  5. Wait until 0000b and press BTNU
  6. Finally press BTNC



- Submit the zip file according to the figure
  - The video should clearly demonstrate LEDs, SWs and BTNs at the same time
  - Deadline: **12:30 on 9 Apr. 2025**
    - ➢ Late submission is NOT acceptable.

香港中文大學
The Chinese University of Hong Kong

Thank You!

Lab10 Outline

- PART 0: Hardware Setup
- PART 1: Flash Image File into SD Card
- PART 2: Write Python Code

① Example

② Task

- **Appendix (Optional)**

Lab10 Appendix Outline

- **PART 0: VM Setting**
- **PART 1: Create PYNQ Image**

① SDSoC Configuration
② Petalinux Configuration
③ Generate Image

- **PART 2: Create PYNQ Overlay**

AMD Vivado

⑤ IP Integration
⑥ HDL Wrapper & Generate Bitstream

VM Setting

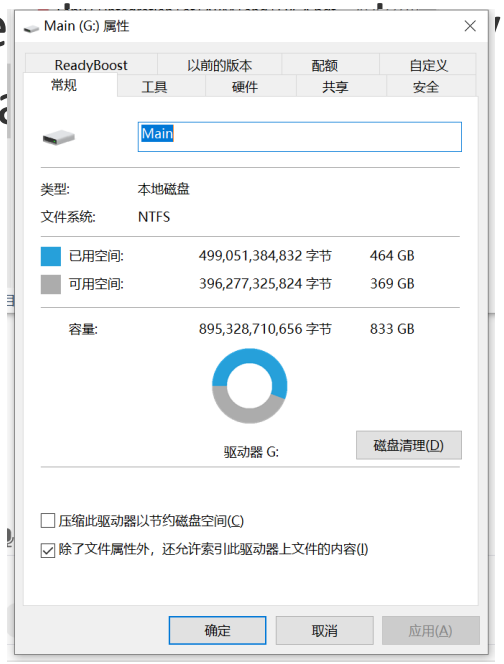
- If you already have the Linux 16.04 environment on your computer, ignore this section.

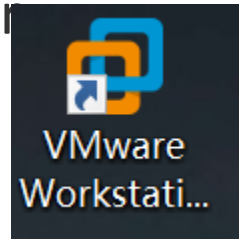
Objective: Create Linux environment for **Part 2**

Output: **Linux 16.04** Environment

VM Setting

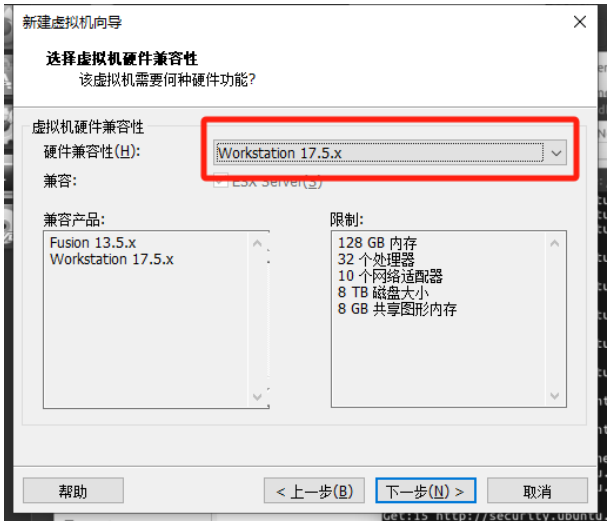
- Download **VMWare Workstation** and **ubuntu-16.04.7-desktop-amd64.iso**
  - www.vmware.com/products/desktop-hypervisor/workstation-and-fusion
  - https://releases.ubuntu.com/16.04/
- Open **VMWare** and Create a New Virtual Machine
  - Before starting this step, you ne… your computer has 80GB of free stora…

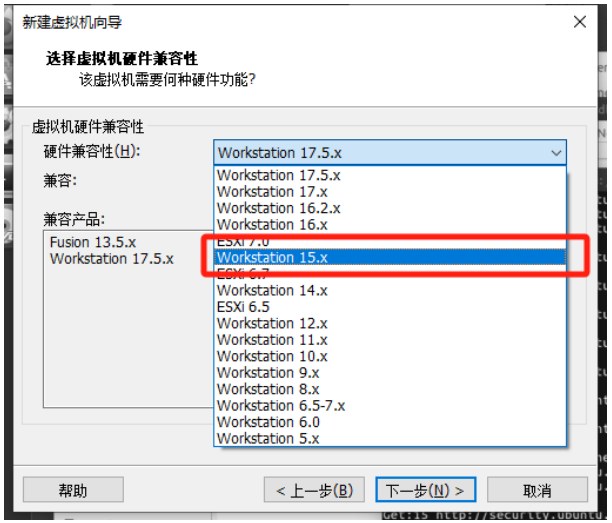


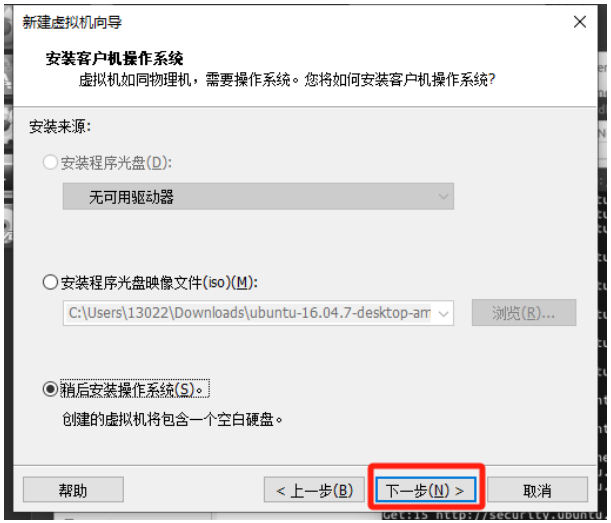


VM Setting

- Follow the Instruction in Following Slides

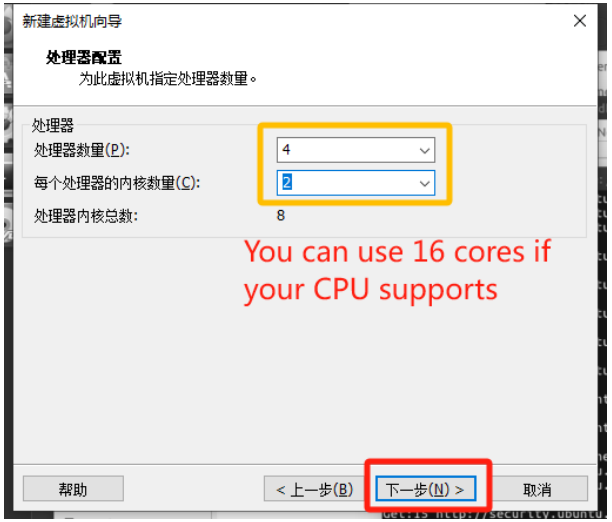






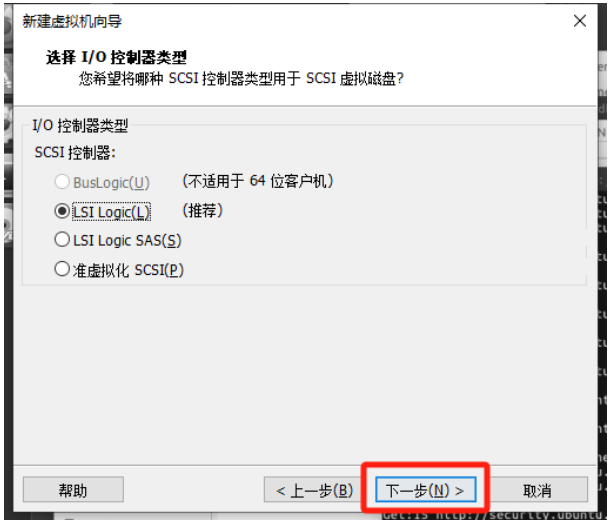
VM Setting

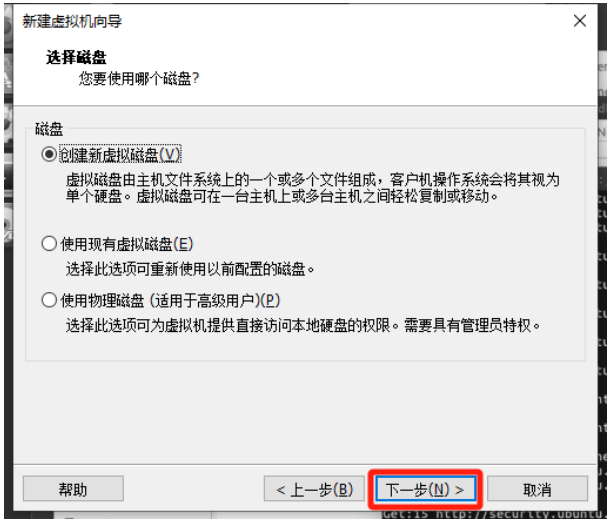
- Follow the Instruction in Following Slides



VM Setting

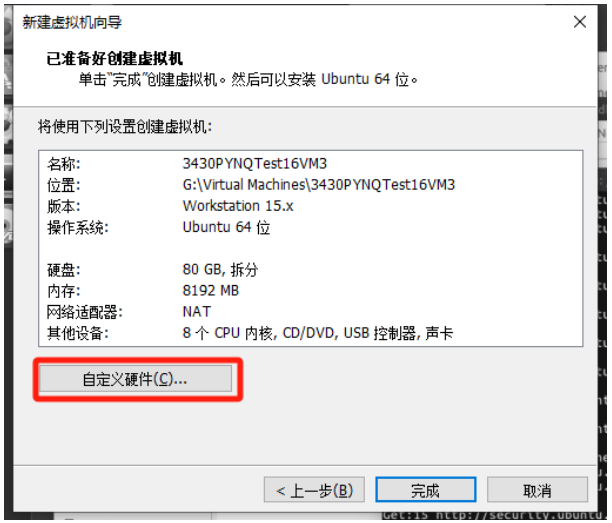
- Follow the Instruction in Following Slides

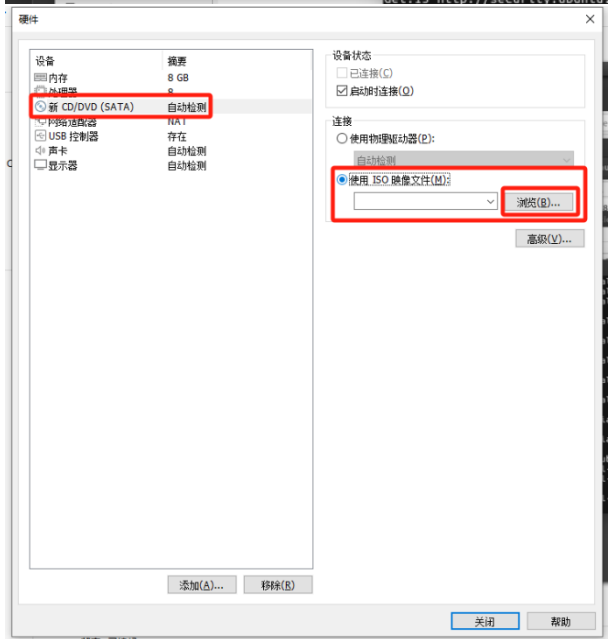




VM Setting

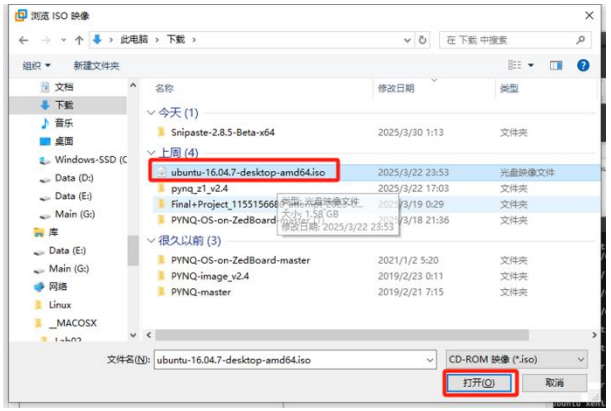
- Follow the Instruction in Following Slides

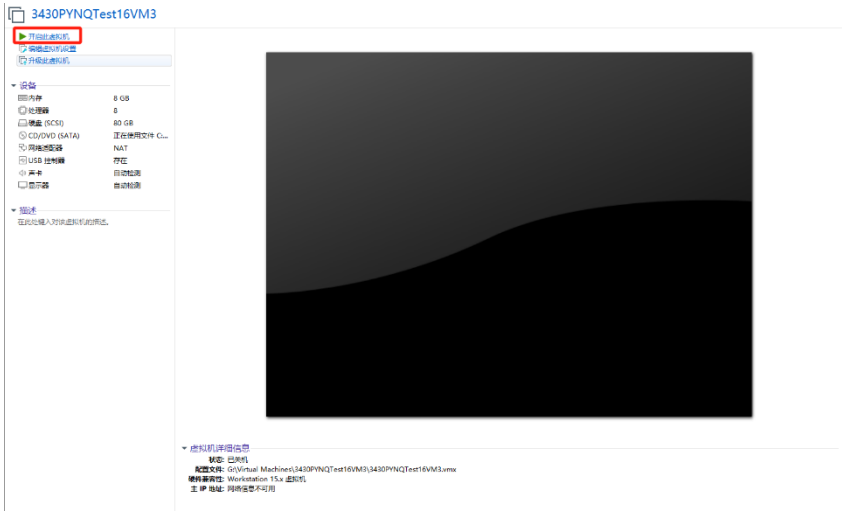


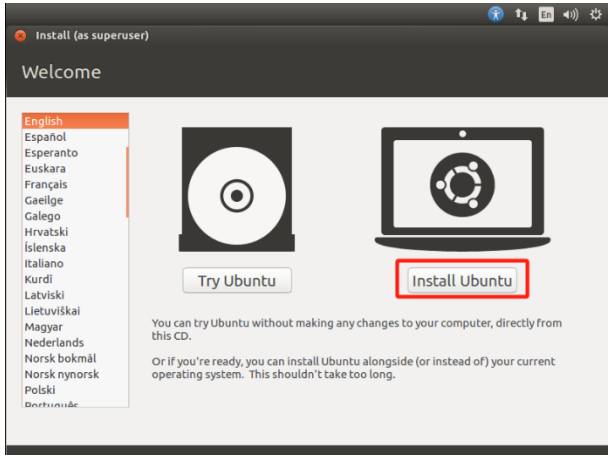


VM Setting

- Follow the Instruction in Following Slides

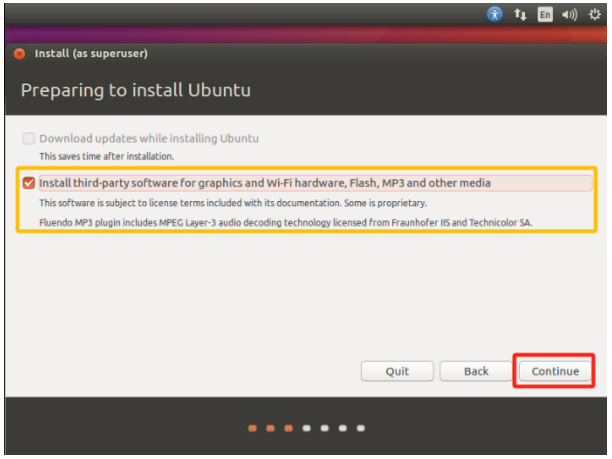


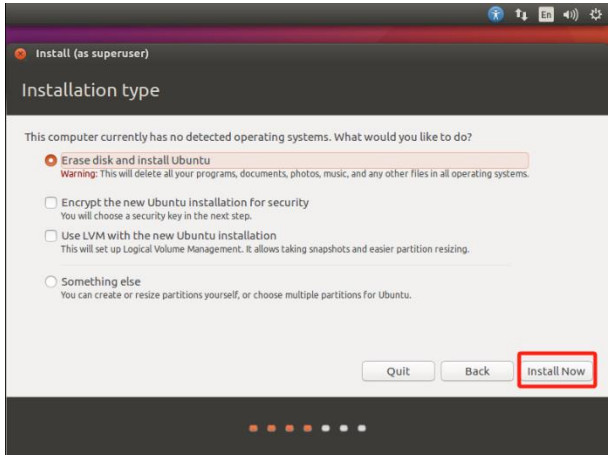


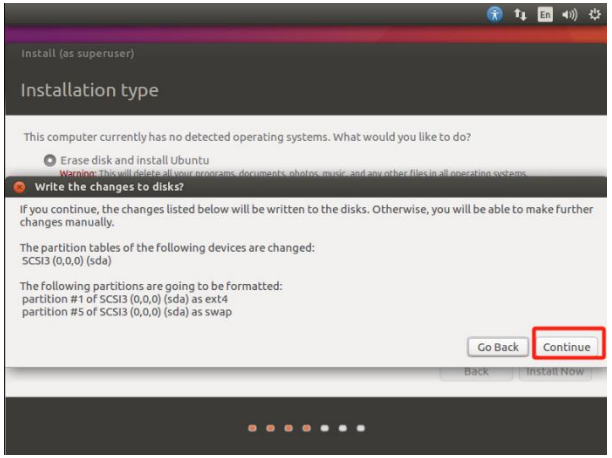


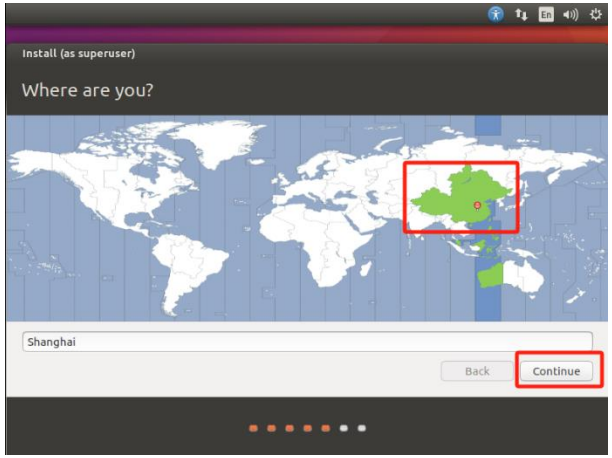
VM Setting

- Follow the Instruction in Following Slides



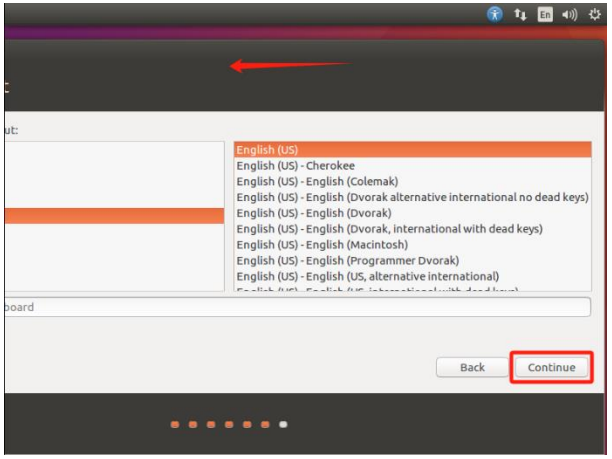


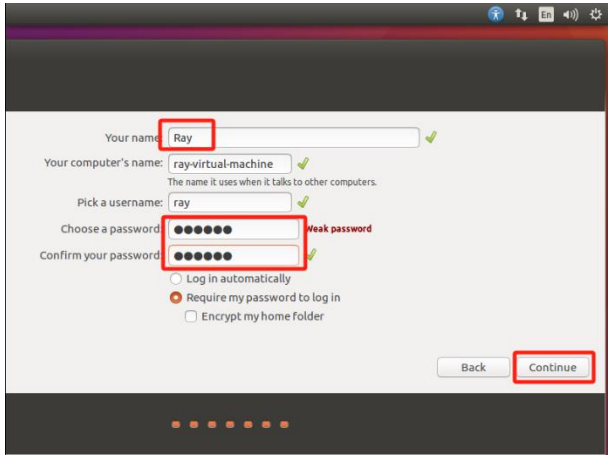


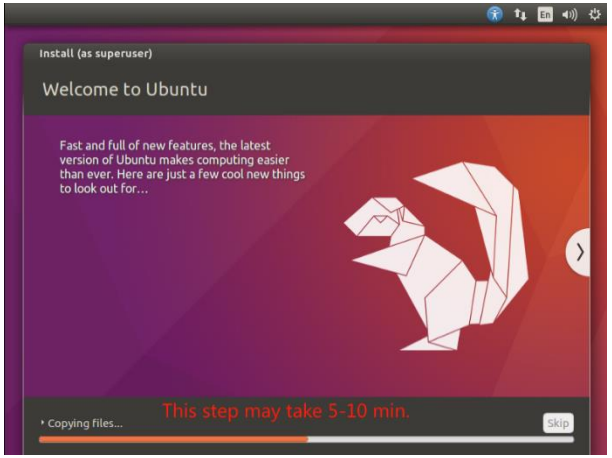


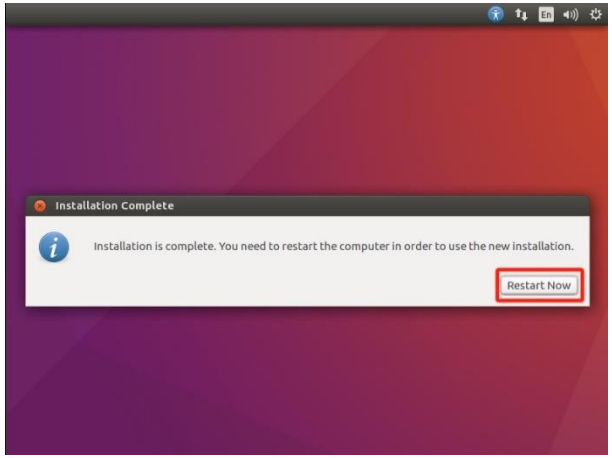
VM Setting

- Follow the Instruction in Following Slides









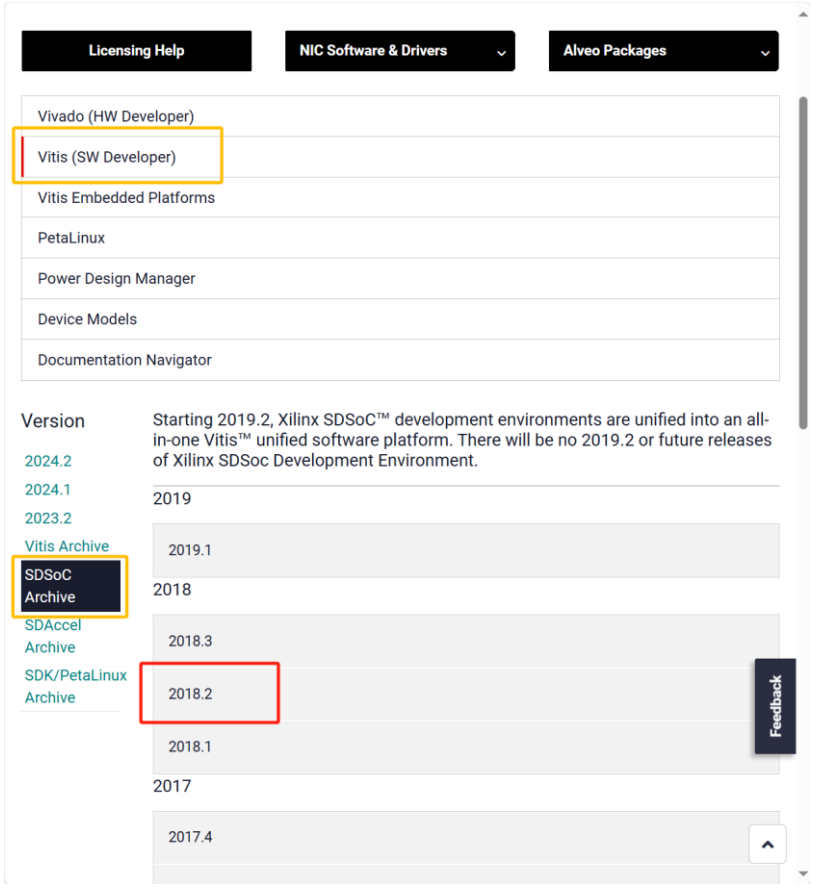
Create PYNQ Image

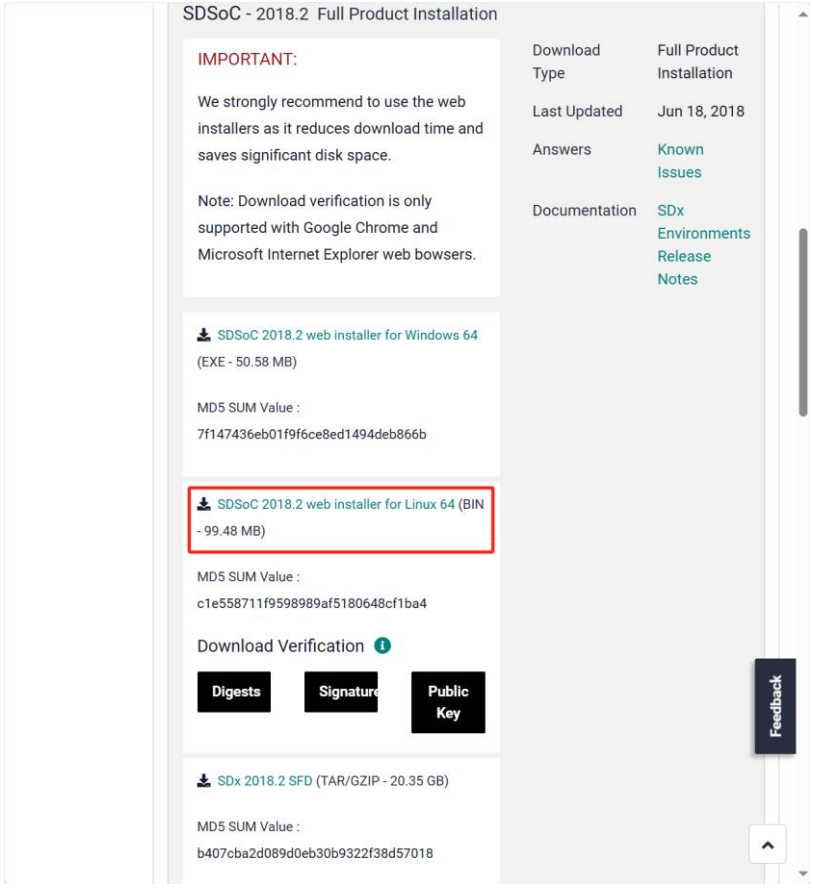
Objective: Support APIs, Drivers, and Apps

Output: **Zedboard_v2.3.img**

Download Required Documents

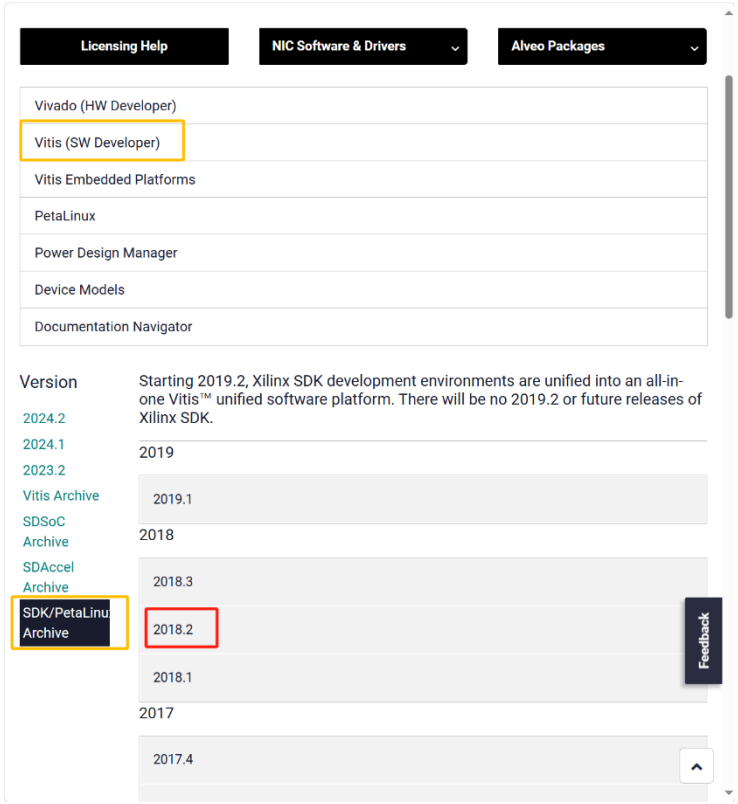
- Download **SDSoC 2018.2 for Linux** from AMD
  - www.xilinx.com/support/download.html/content/xilinx/en/downloadNav/vitis/archive-sdsoc.html

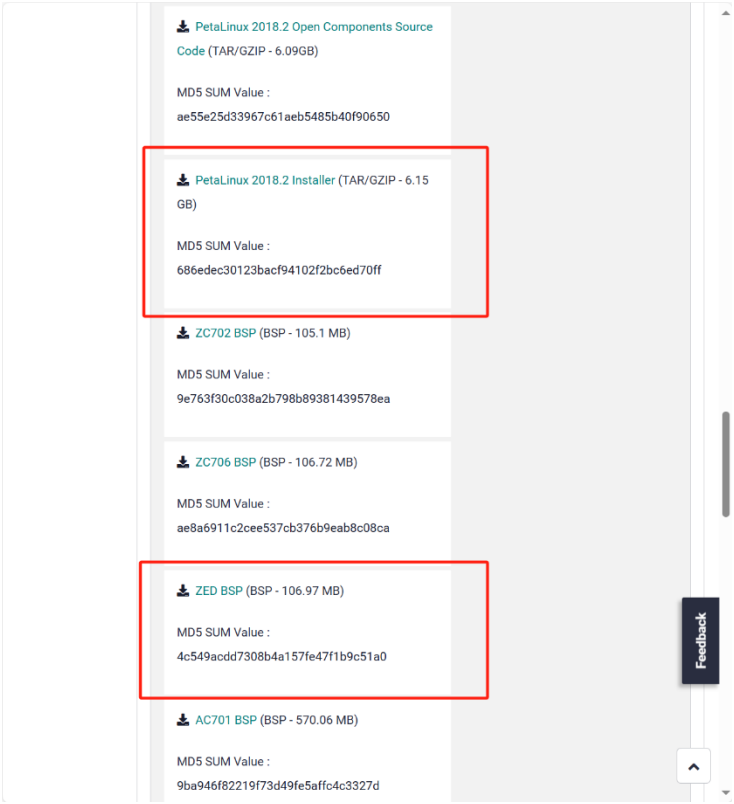




Download Required Documents

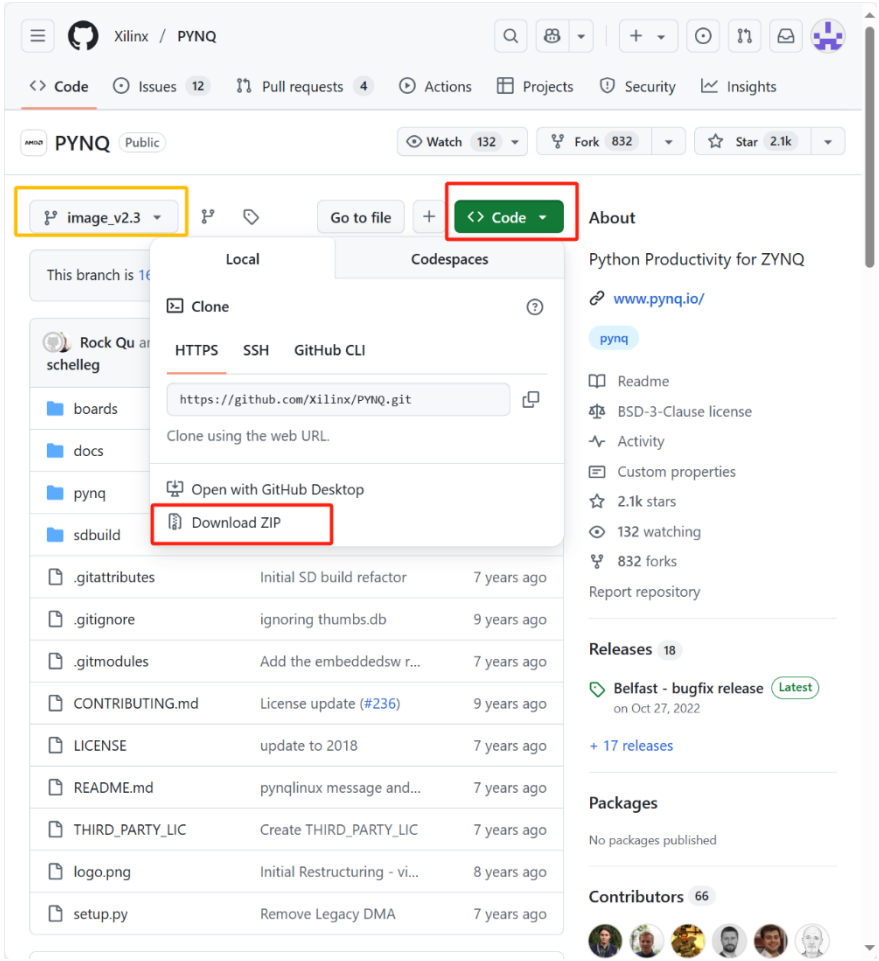
- Download **Petalinux 2018.2** and **.bsp file for Zedboard** from AMD
  - www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis/archive-sdk.html

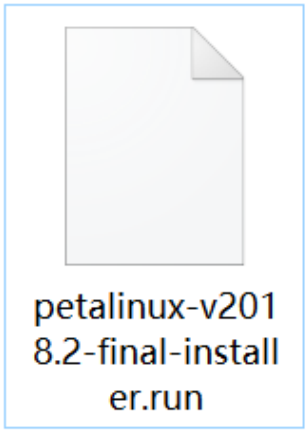


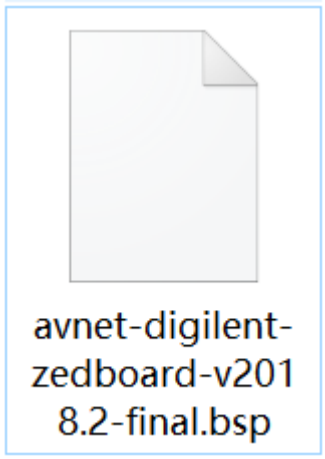


Download Required Documents

- Download **PYNQ image source code v2.3**
  - https://github.com/Xilinx/PYNQ/tree/image_v2.3

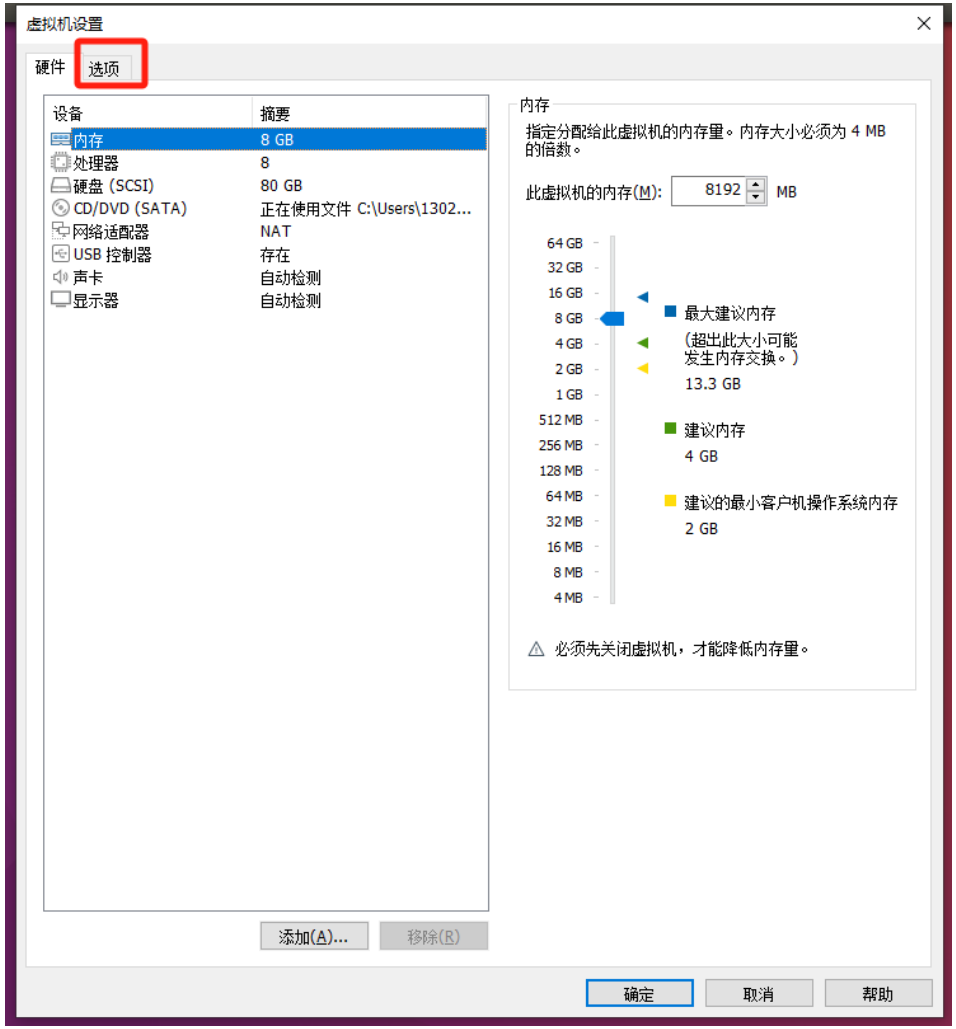


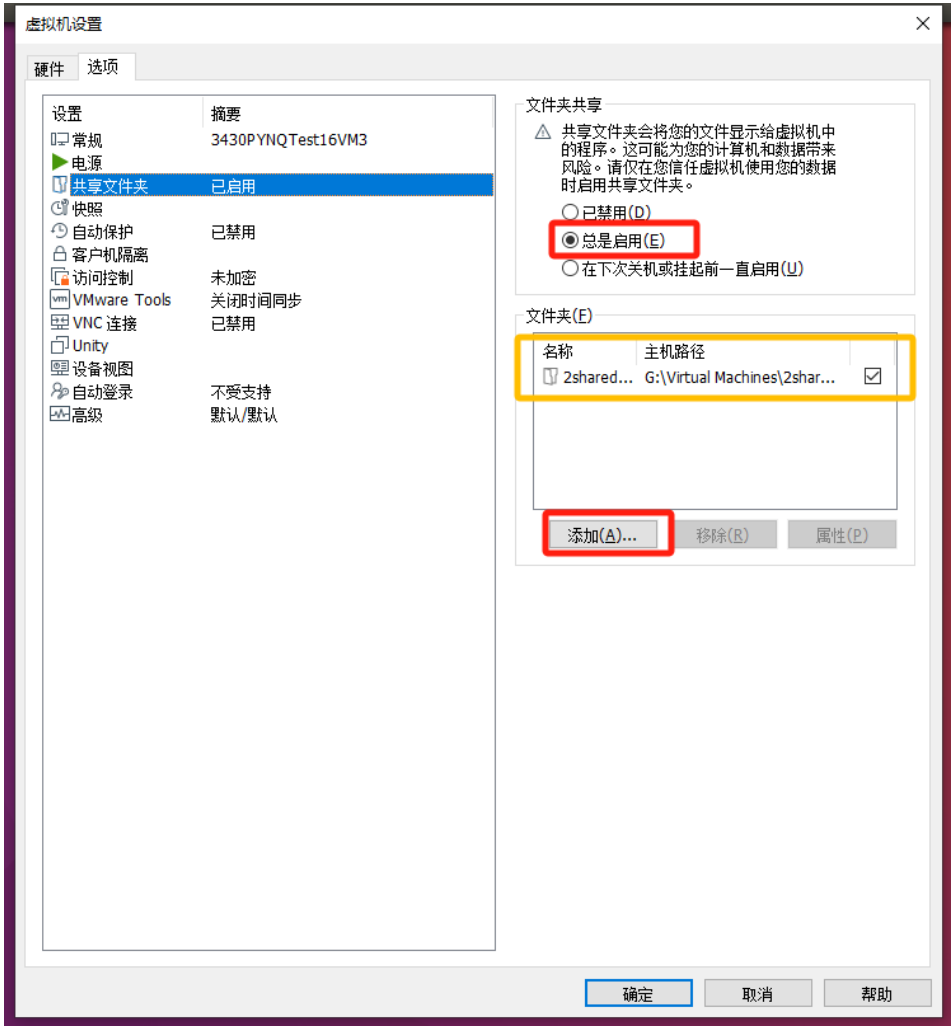




Install and Configure SDSoC

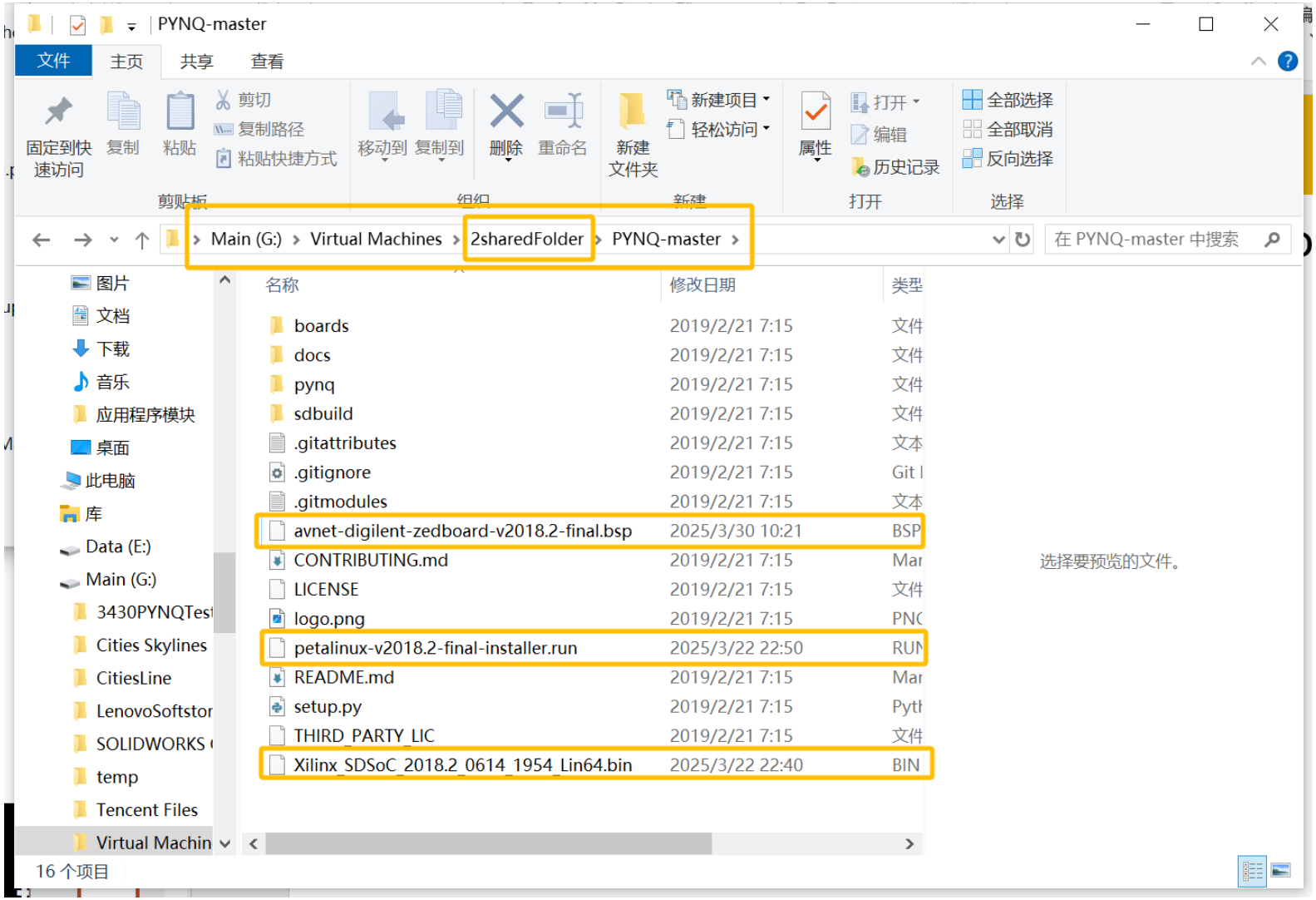
- Create a Shared Folder in VMware setting





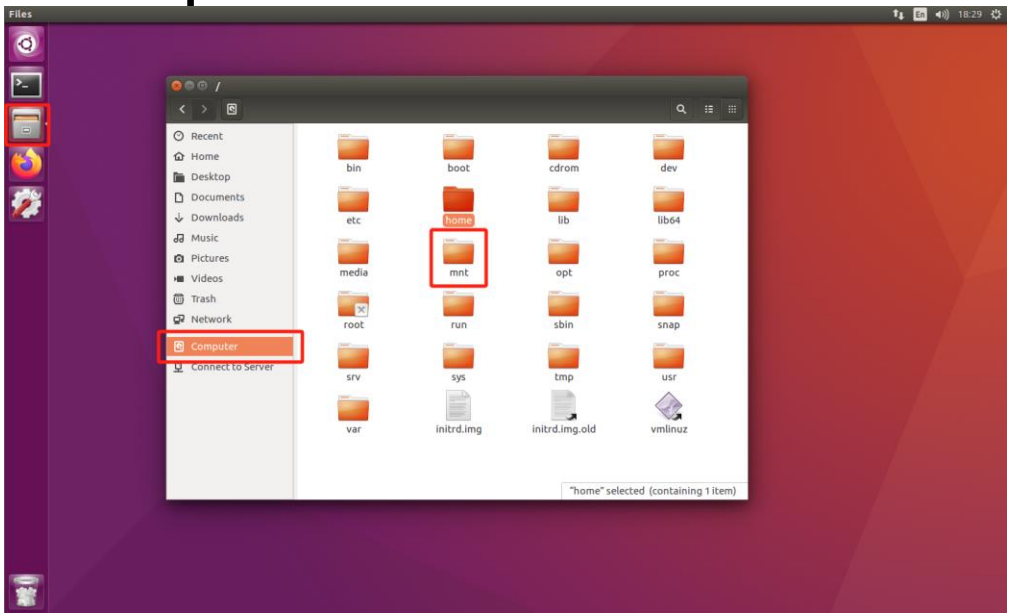
Install and Configure SDSoC

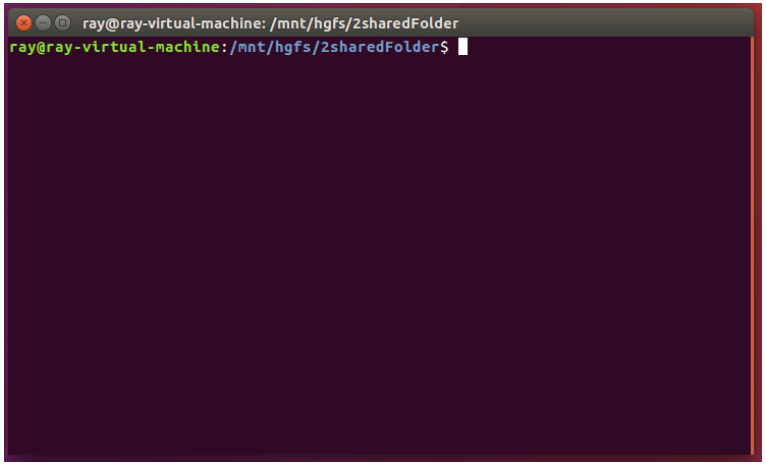
- Move All Downloaded Files into the Shared Folder

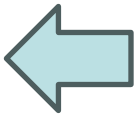


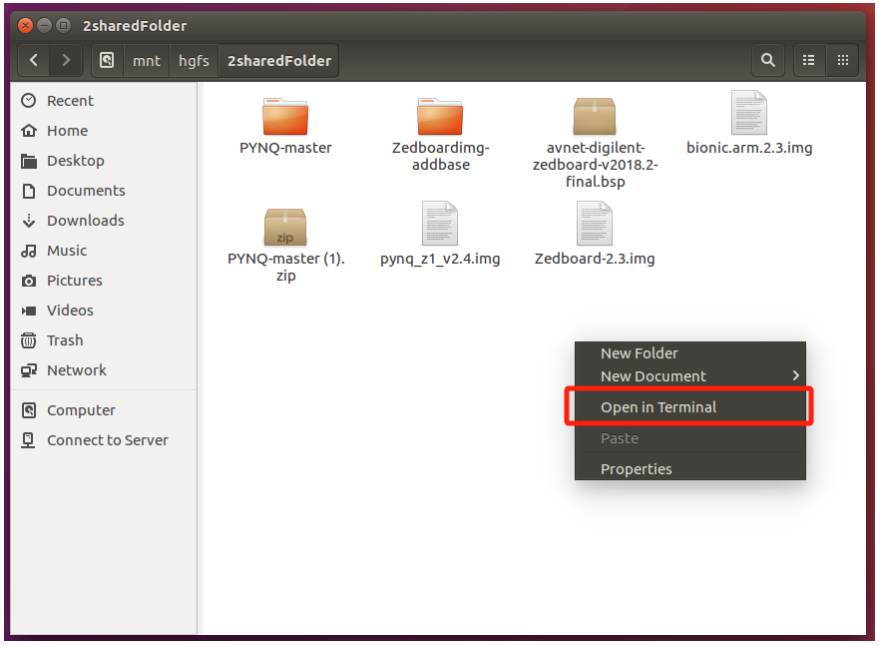
Install and Configure SDSoC

- Open the Shared Folder and Terminal in the VM



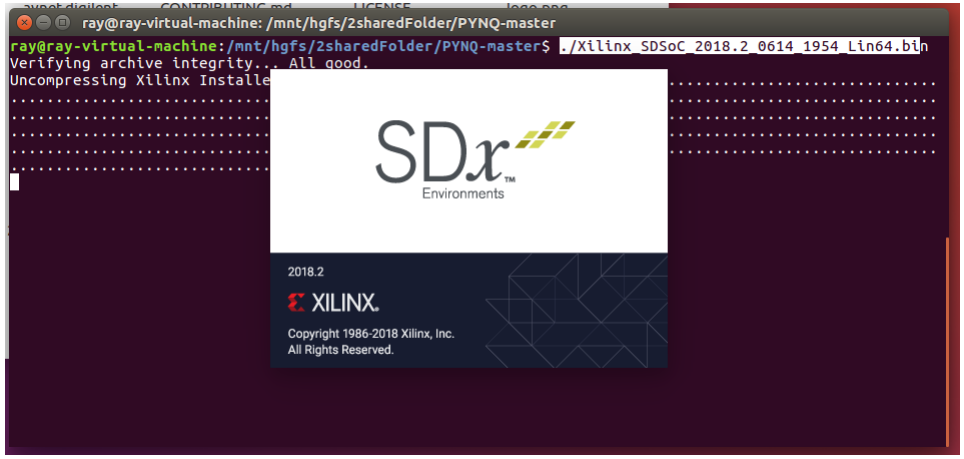


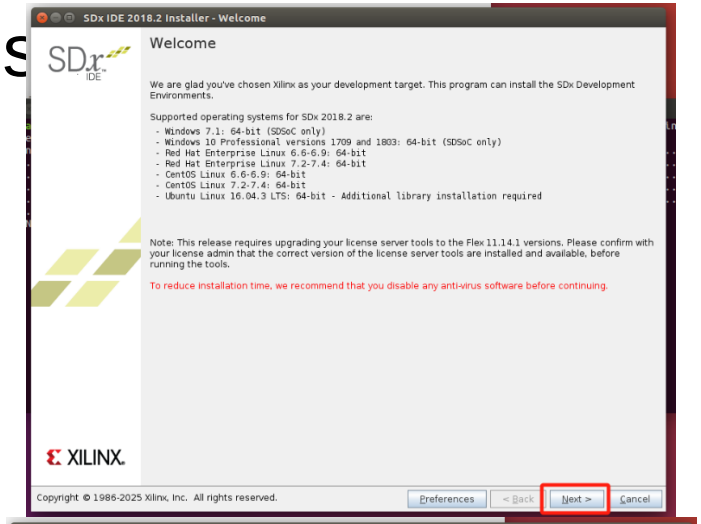


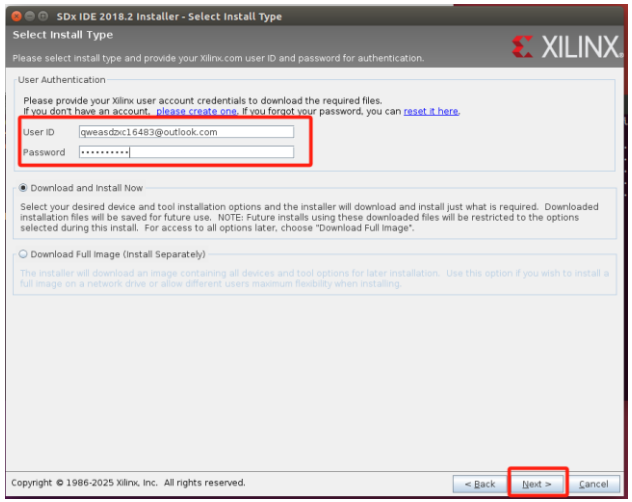


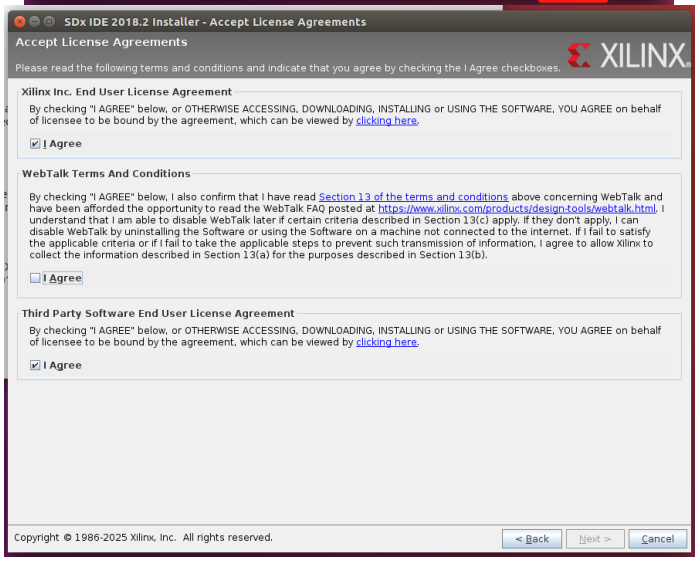
Install and Configure SDSoC

- Input ./Xilinx_SDSoC_2018.2_0614_1954_Lin64.bin
- Follow the default instructions



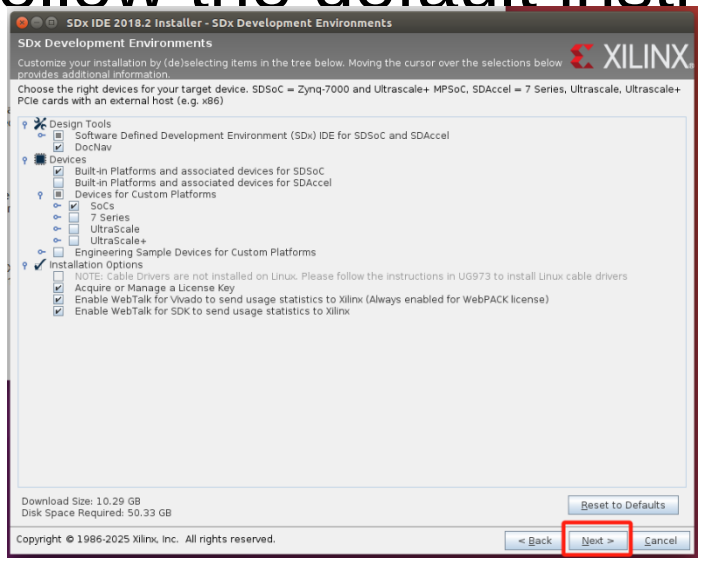


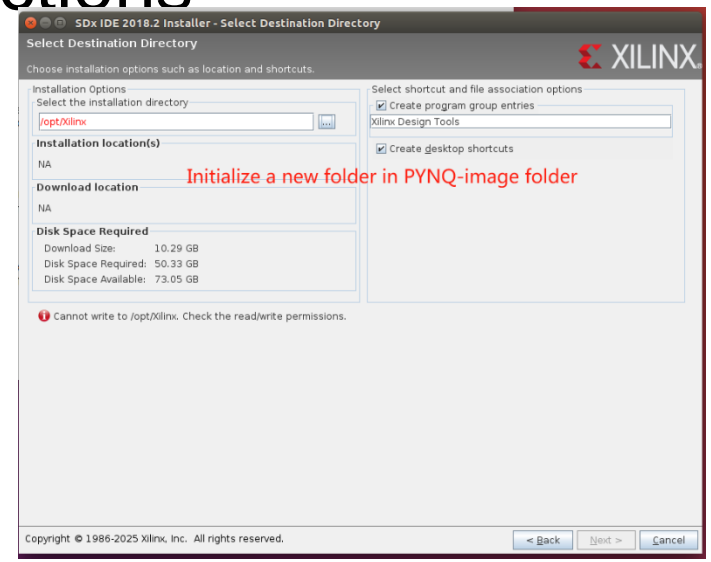


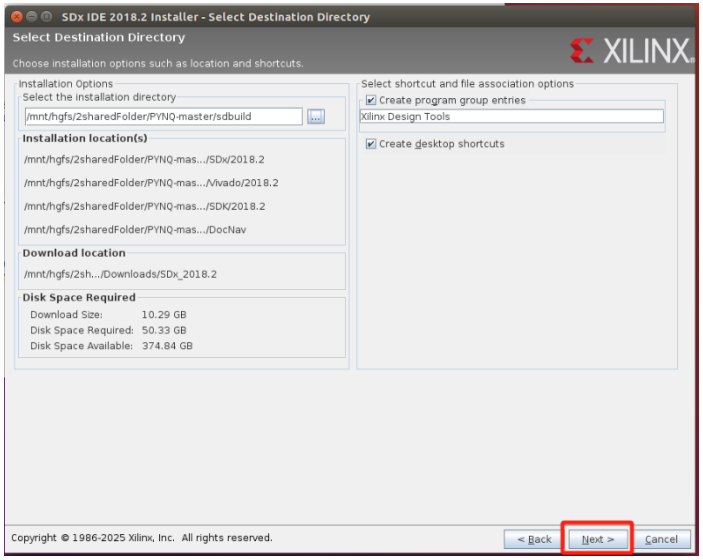


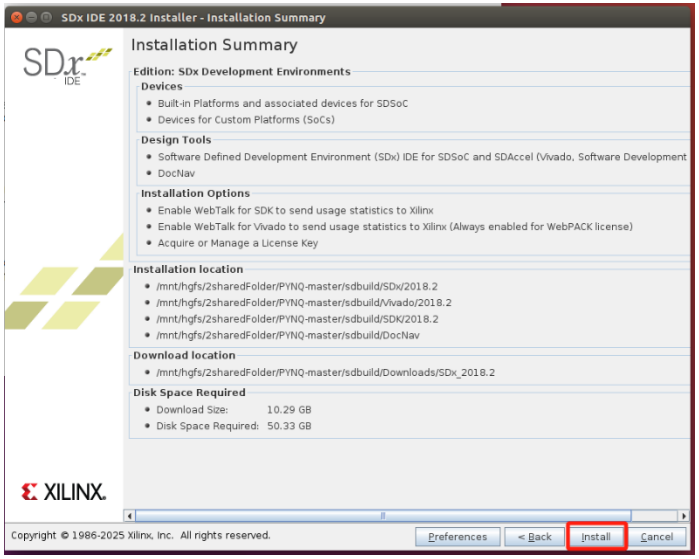
Install and Configure SDSoC

- Follow the default instructions



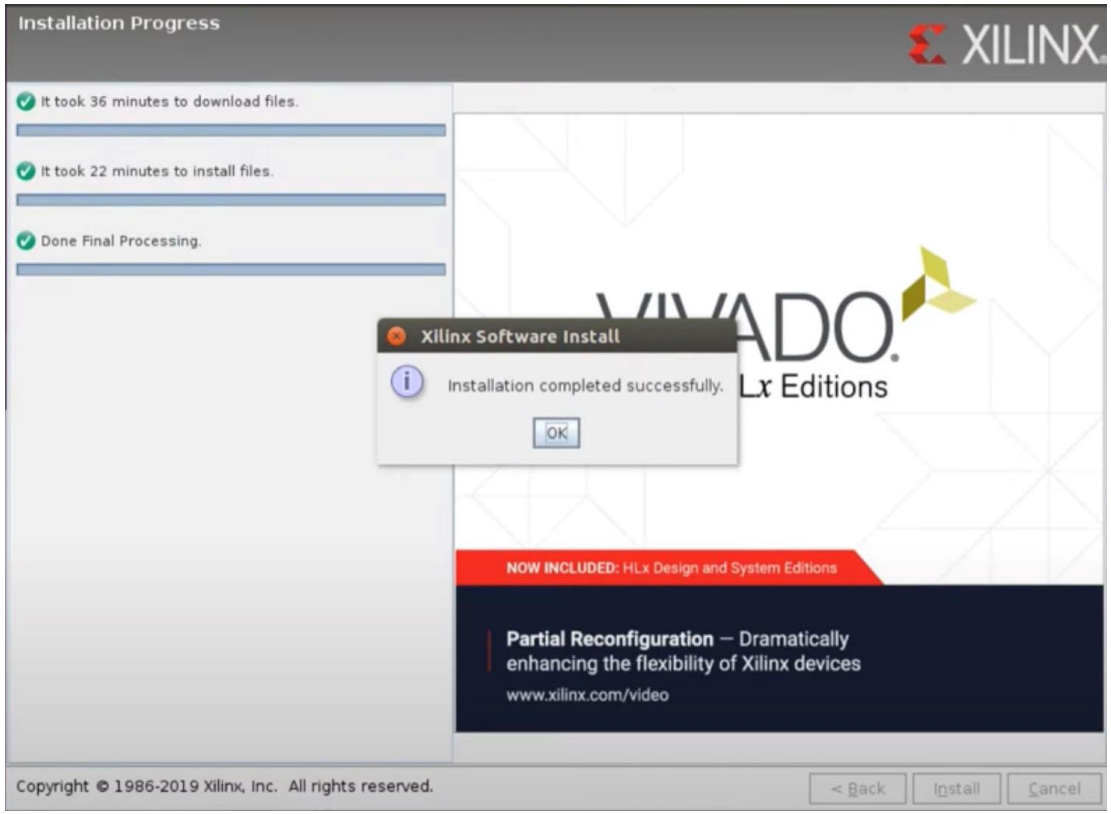


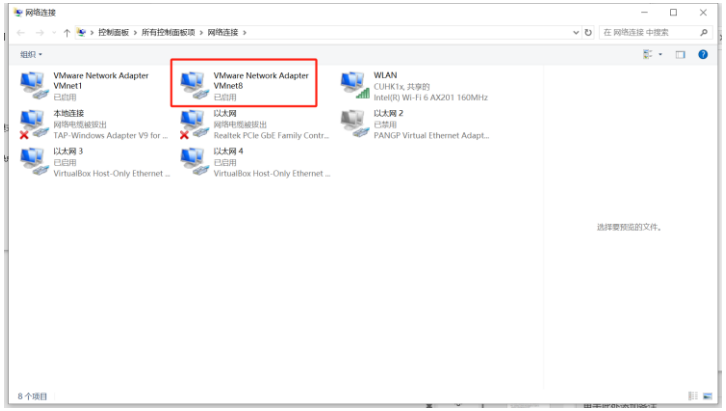




Install and Configure SDSoC

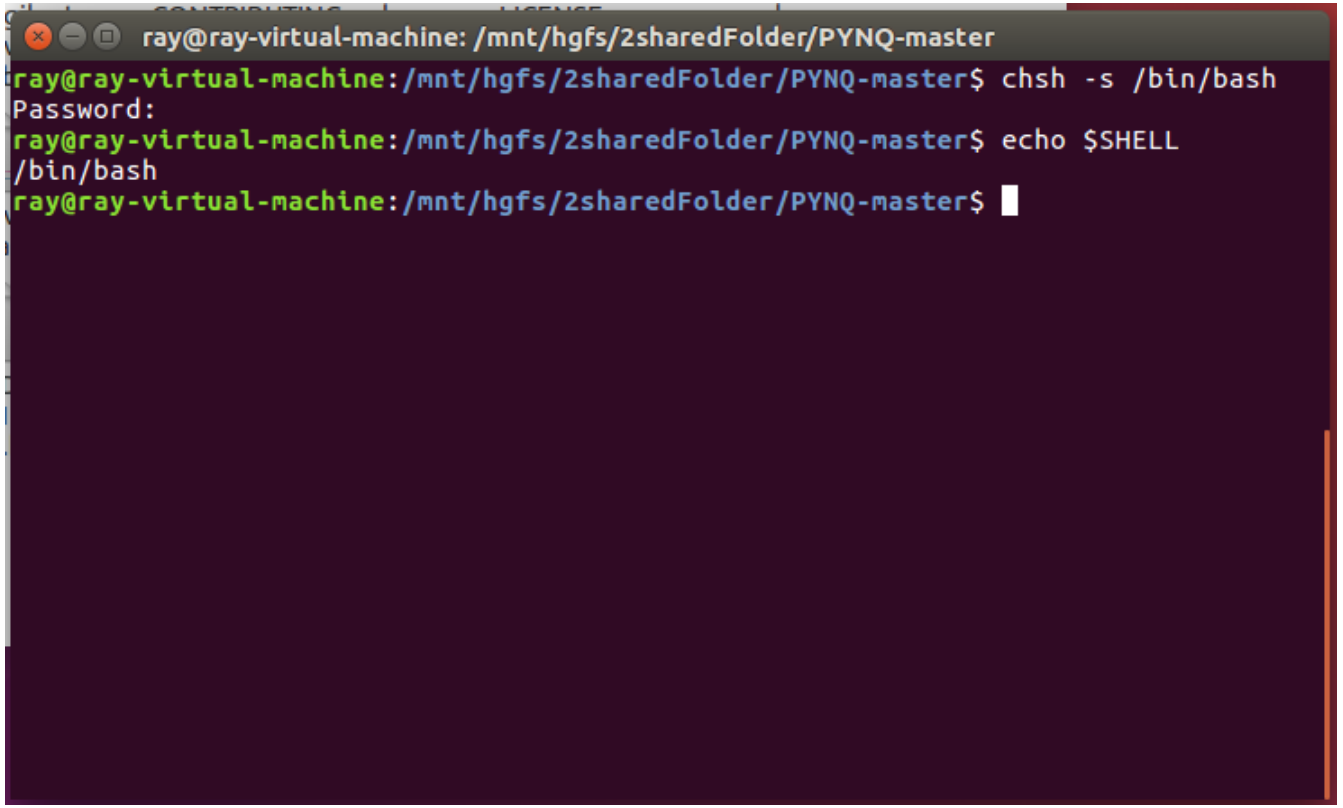
- The installation may take more than 15 min.
  - If you encountered unknown errors, you can check
    - Ethernet connection for NMVet8
    - Rest disk space in both host and virtual machine





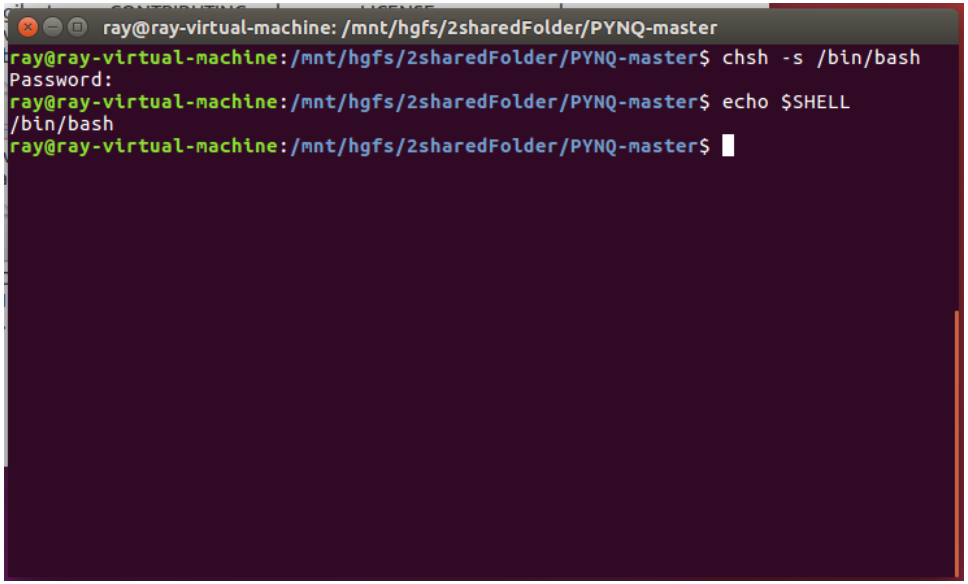
Install Dependencies for Petalinux

- Run **chsh -s /bin/bash** and restart the VM
  - ***echo $SHELL*** can be used to check if Shell has been changed to bash
  - The output should be */bin/bash*



Install Dependencies for Petalinux

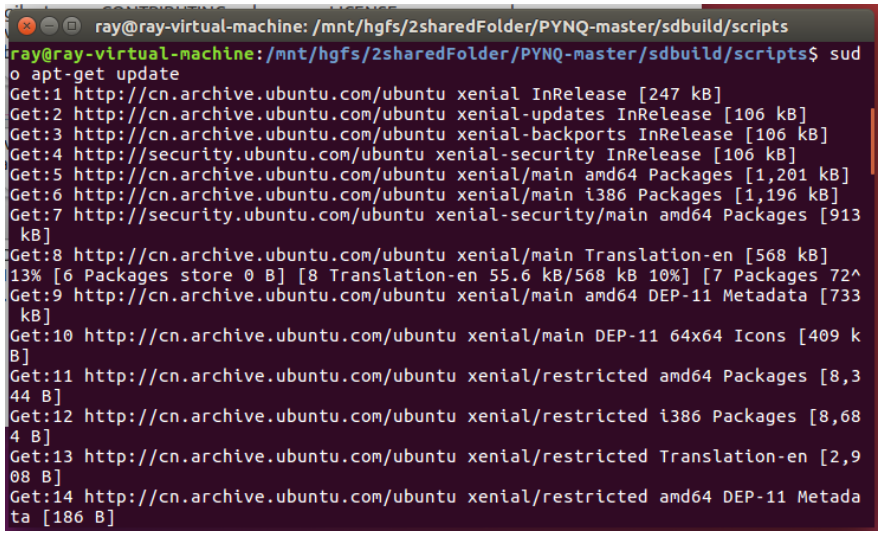
- Run **chsh -s /bin/bash** and restart the VM
  - ***echo $SHELL*** can be used to check if Shell has been changed to bash
  - The output should be */bin/bash*

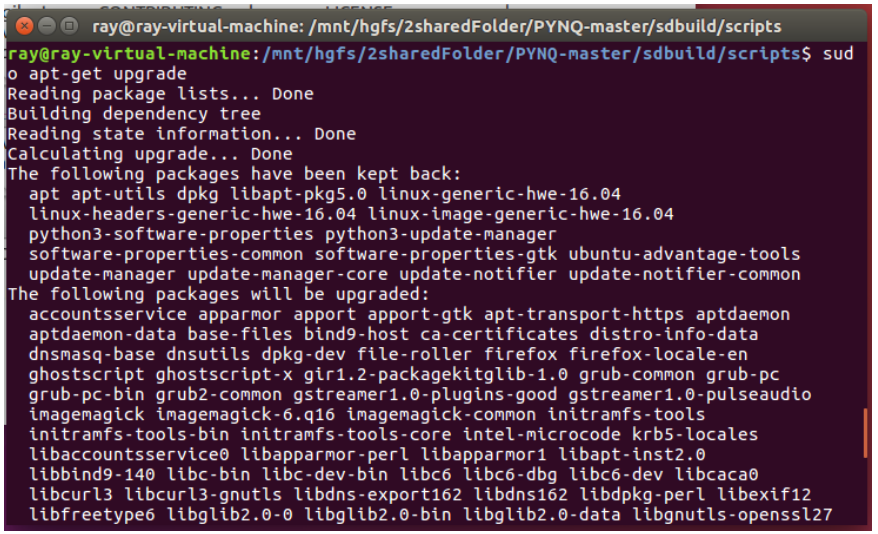


- This step ensures that the system is using Bash as the shell, which is commonly used in development environments due to its compatibility and features.

Install Dependencies for Petalinux

- Run **sudo apt-get update** and **sudo apt-get**



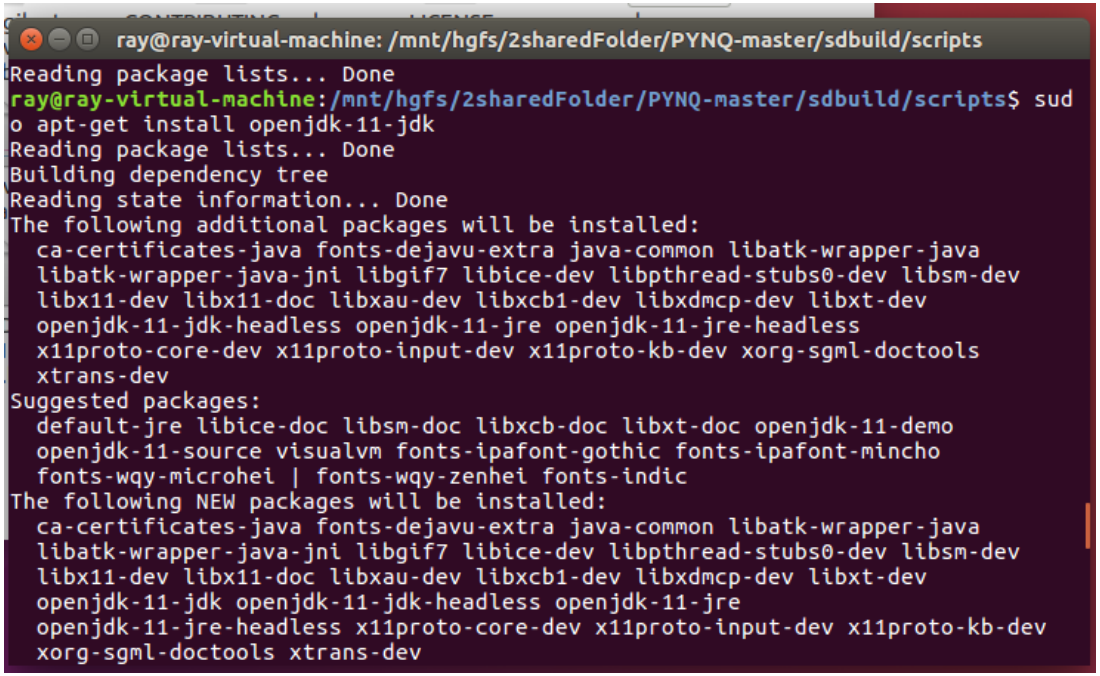


- sudo apt-get update: Updates the local package index with the latest information from the repositories, ensuring that your system is aware of the latest available software versions.

- sudo apt-get upgrade: Upgrades all installed packages on your system to their latest versions based on the updated package list.

Install Dependencies for Petalinux

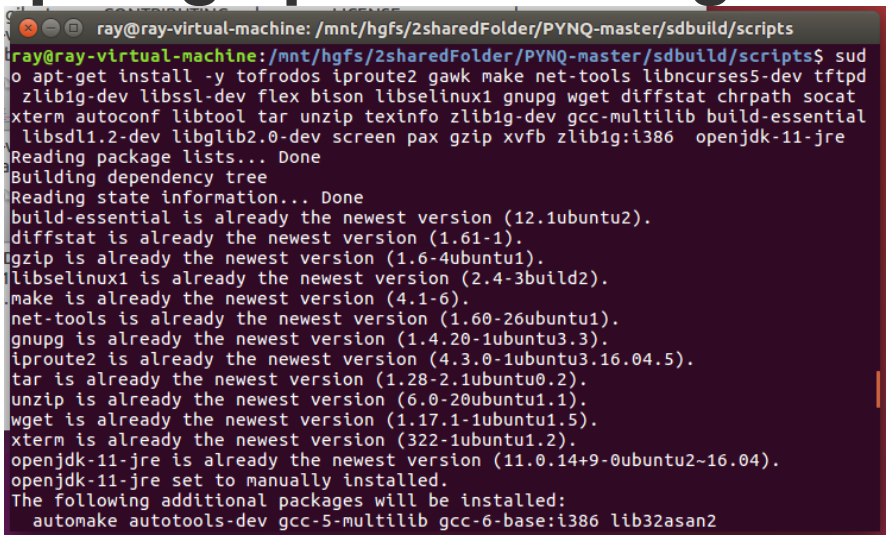
- Run **sudo add-apt-repository ppa:openjdk-r/ppa**
- Run **sudo apt-get update**



- Ubuntu 16.04's default repositories only include OpenJDK 8 (the package openjdk-8-jre), and OpenJDK 11 was not included in those repositories. So, we need to add a repository that includes it

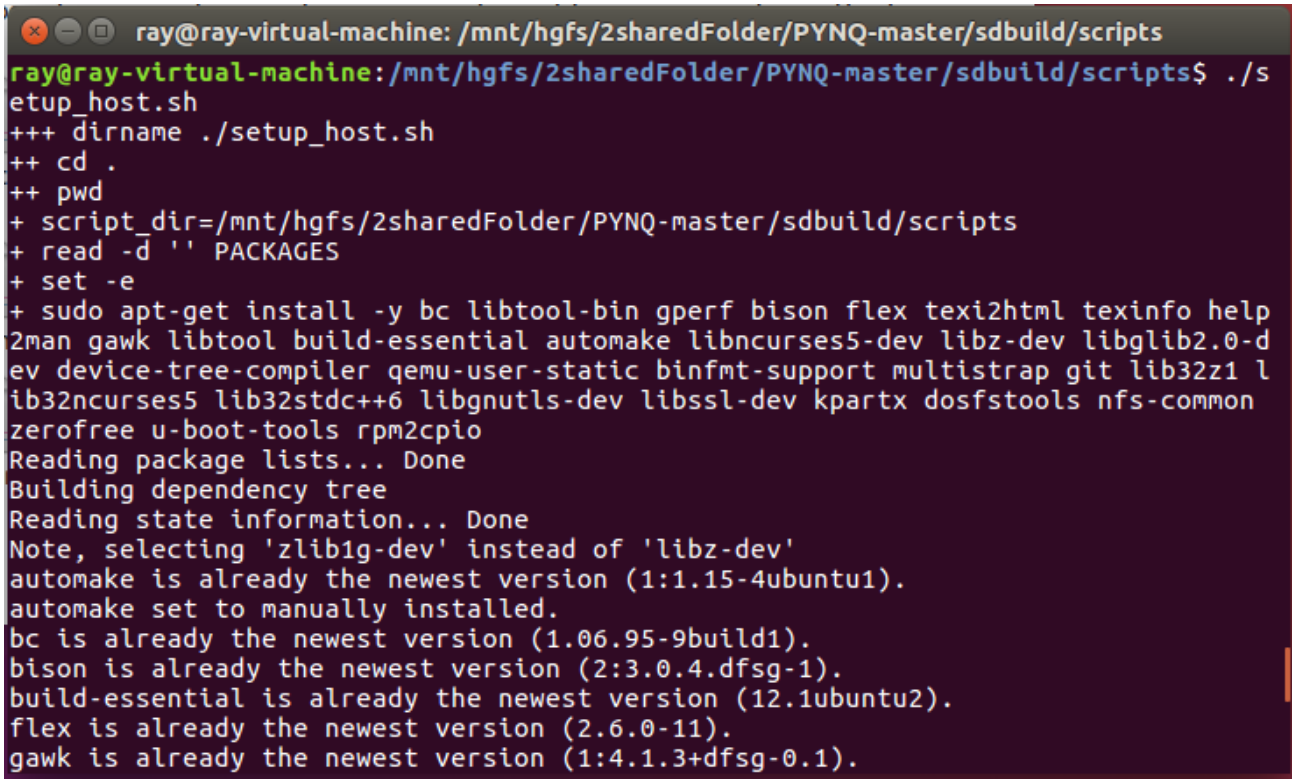
Install Dependencies for Petalinux

- Run **sudo apt-get install -y tofrodos iproute2 gawk make net-tools libncurses5-dev tftpd zlib1g-dev libssl-dev flex bison libselinux1 gnupg wget diffstat chrpath socat xterm autoconf libtool tar unzip texinfo zlib1g-dev gcc-multilib build-essential libsdl1.2-dev libglib2.0-dev screen pax gzip xvfb zlib1g:i386 openjdk-11-jre**



Install Dependencies for Petalinux

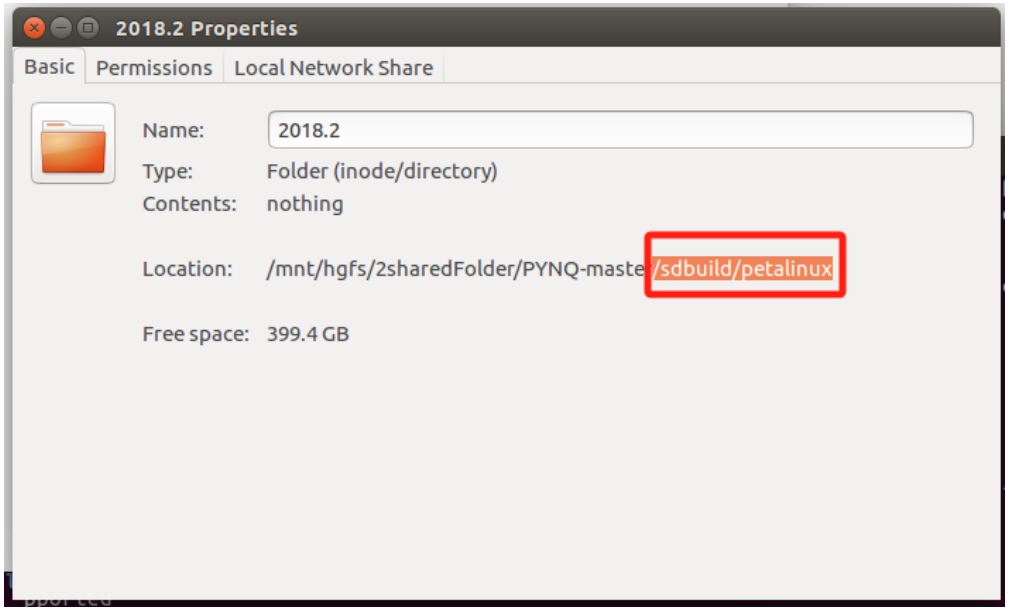
- Run **sdbuild/scripts/setup_hosts.sh**

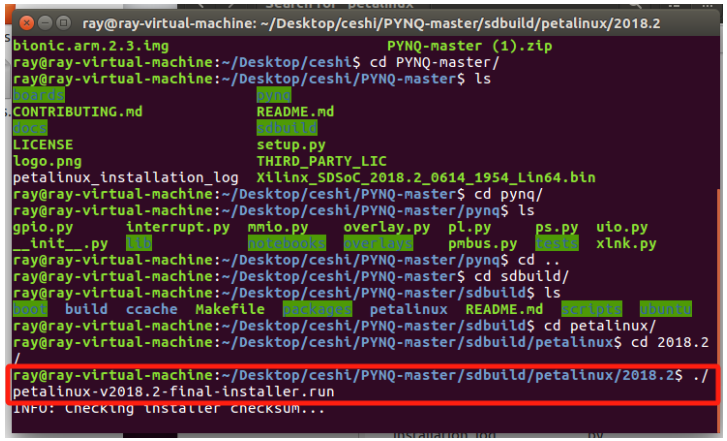


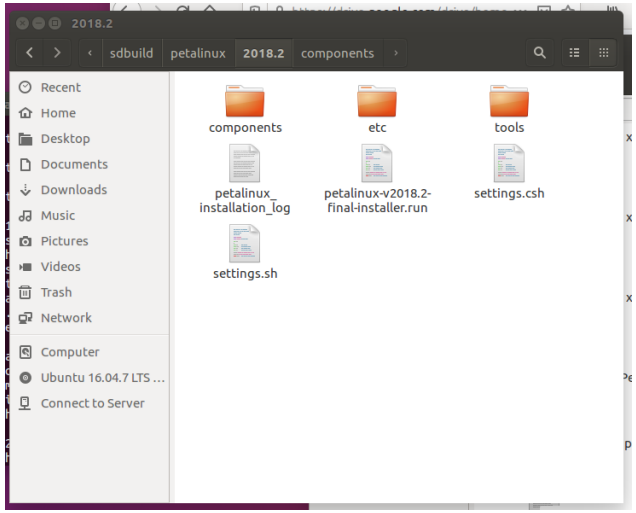
- If you encounter error such as *Cannot create symlink to 'COPYING',* you need to create a new folder in your VM and copy all files in the shared folder into it.

Install Petalinux (SUPER HARD)

- Initialize a new folder under sdbuild name as petalinux/2018.2
- Run **./petalinux-v2018.2-final-installer.run /sdbuild/petalinux/2018.2**



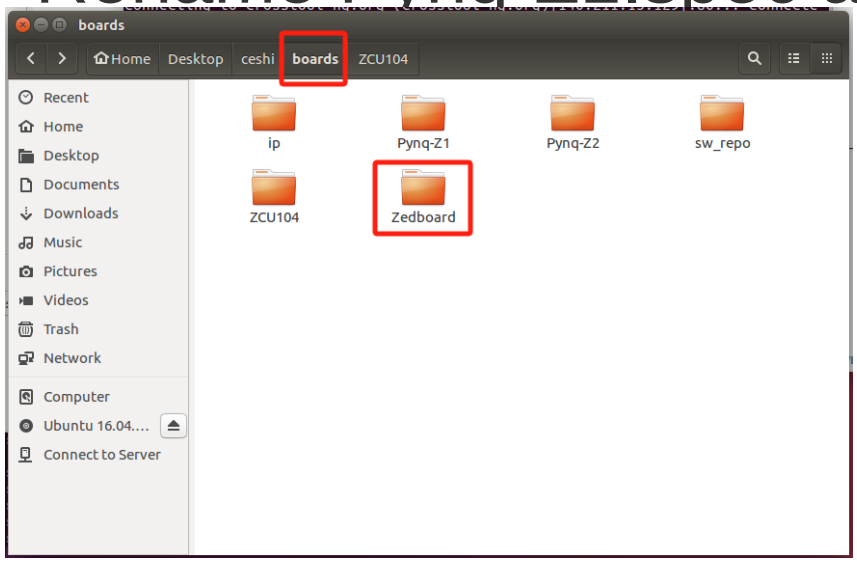


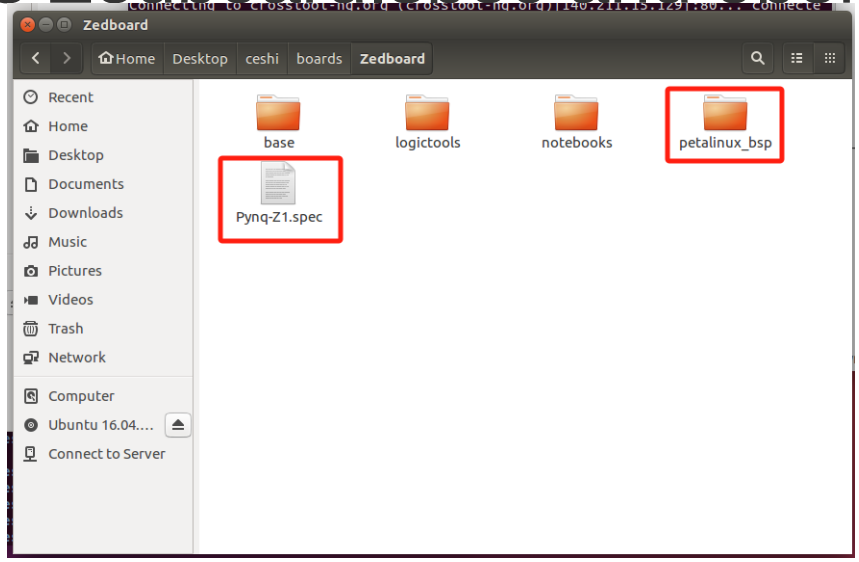


  - After installation, the folder used for installation should be the same as the right picture

Add .bsp file

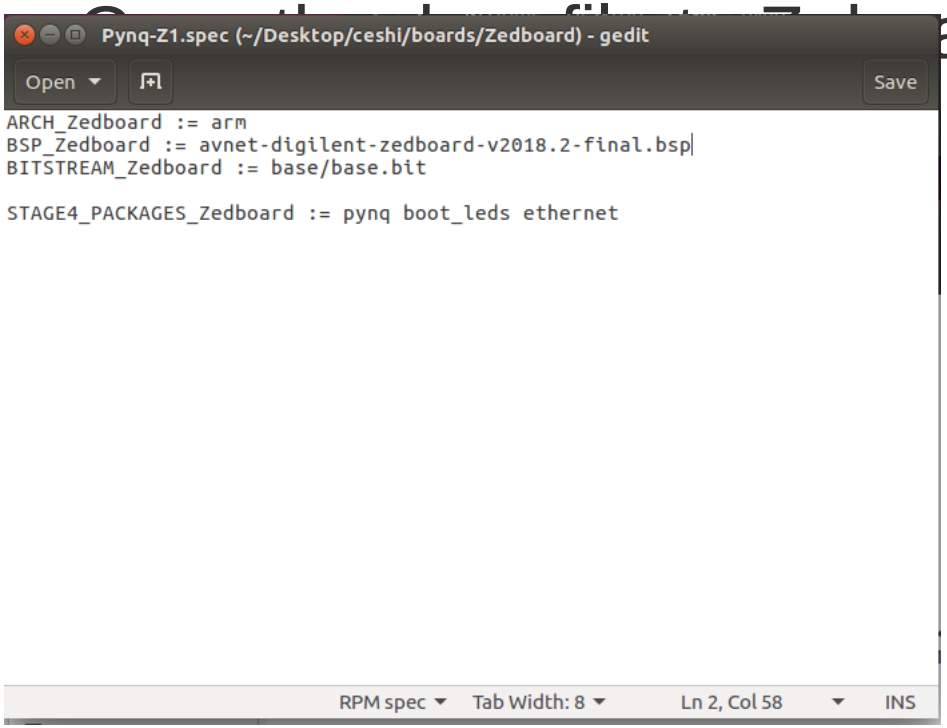
- Open boards under PYNQ-master folder
- Duplicate Pynq-Z1 and rename the folder as Zedboard
- Delete *petalinux_bsp* folder and add *avnet-digilent-zedboard-2018.2.bsp* to this folder
- Rename Pynq-Z1.spec as Zedboard.spec and open it

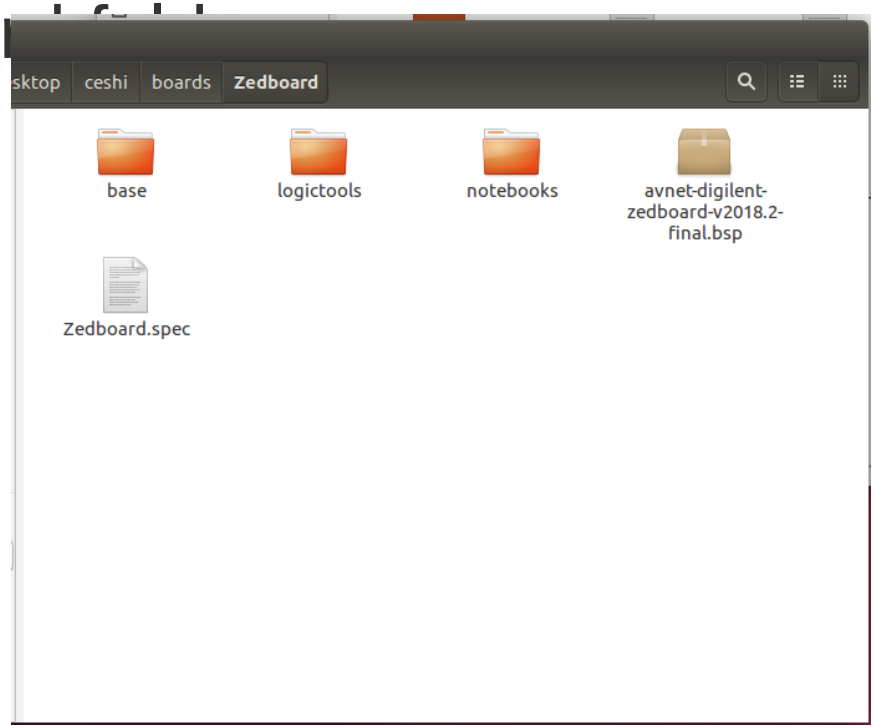




Add .bsp file

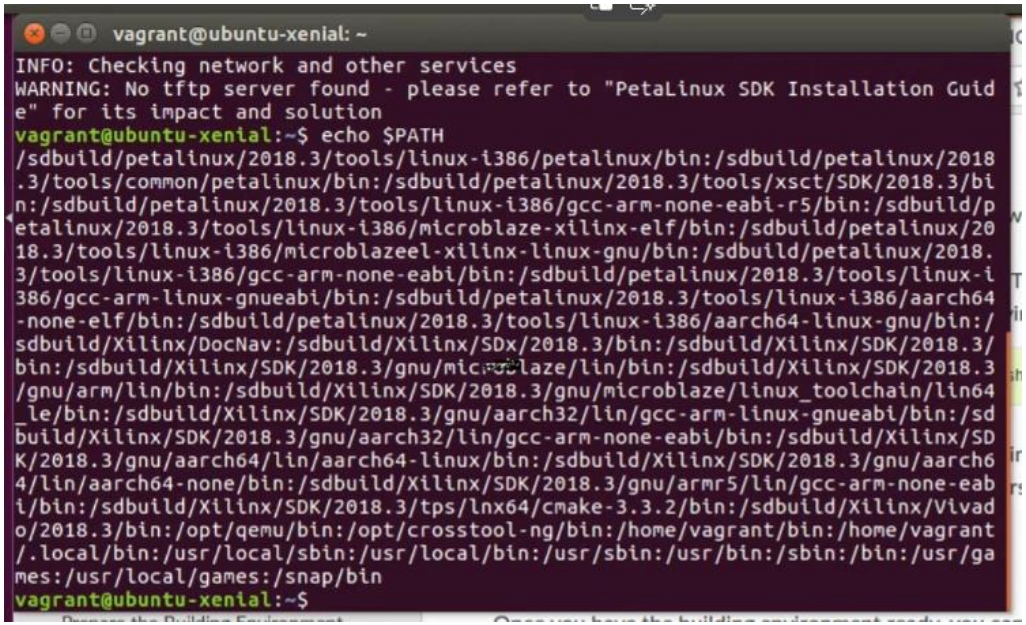
- Change all “PYNQ-Z1” to “zed” and fill BSP filename
- Attach the file name of .bsp file after BSP_Zedboard :=





Prepare Environment

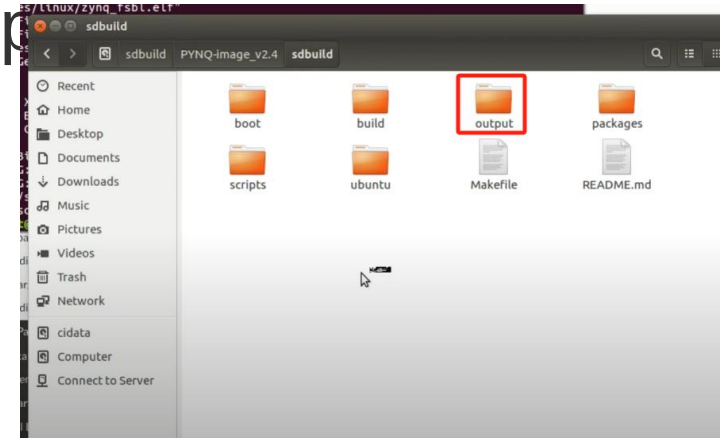
- Run **source sdbuild/Xilinx/Vivado/2018.2/setting64.sh**
- Run **source sdbuild/petalinux/2018.2/setting.sh**
- Run **source sdbuild/SDSoC/2018.2/setting64.sh**
  - echo $PATH can be used to check source variables
  - The output should be like

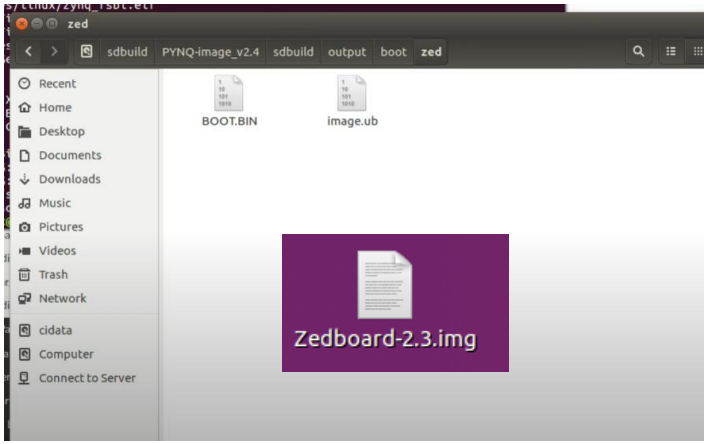


Build the Image

- Run **make boot_file BOARDS=zed** under sdbuild folder
  - It may take 4-5 hours
- The output are 1)a folder named output with image.ub inside and 2).img file at desktop

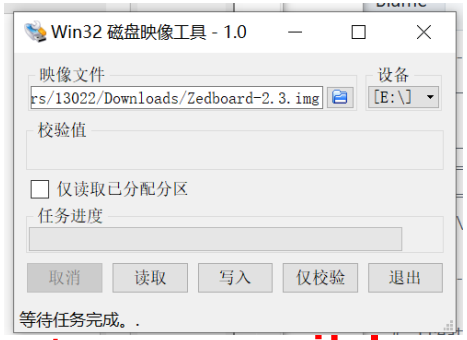
```
vagrant@ubuntu-xenial: /sdbuild/PYNQ-image_v2.4/sdbuild
vagrant@ubuntu-xenial:/sdbuild/PYNQ-image_v2.4/sdbuild$ make boot_files BOARDS=zed
fatal: Not a git repository (or any parent up to mount point /sdbuild)
Stopping at filesystem boundary (GIT_DISCOVERY_ACROSS_FILESYSTEM not set).
/opt/qemu/bin/qemu-arm-static -version | fgrep 2.8.0
qemu-arm version 2.8.0
which vivado | fgrep 2018.3
/sdbuild/Xilinx/Vivado/2018.3/bin/vivado
which sdx | fgrep 2018.3
/sdbuild/Xilinx/SDx/2018.3/bin/sdx
which petalinux-config | fgrep 2018.3
/sdbuild/petalinux/2018.3/tools/common/petalinux/bin/petalinux-config
which arm-linux-gnueabihf-gcc
/sdbuild/petalinux/2018.3/tools/linux-i386/gcc-arm-linux-gnueabi/bin/arm-linux-gnuea
bihf-gcc
which microblaze-xilinx-elf-gcc
/sdbuild/petalinux/2018.3/tools/linux-i386/microblaze-xilinx-elf/bin/microblaze-xili
nx-elf-gcc
which ct-ng
/opt/crosstool-ng/bin/ct-ng
which python | fgrep /usr/bin/python
/usr/bin/python
sudo -n mount > /dev/null
bash /sdbuild/PYNQ-image_v2.4/sdbuild/scripts/check_Env.sh
```

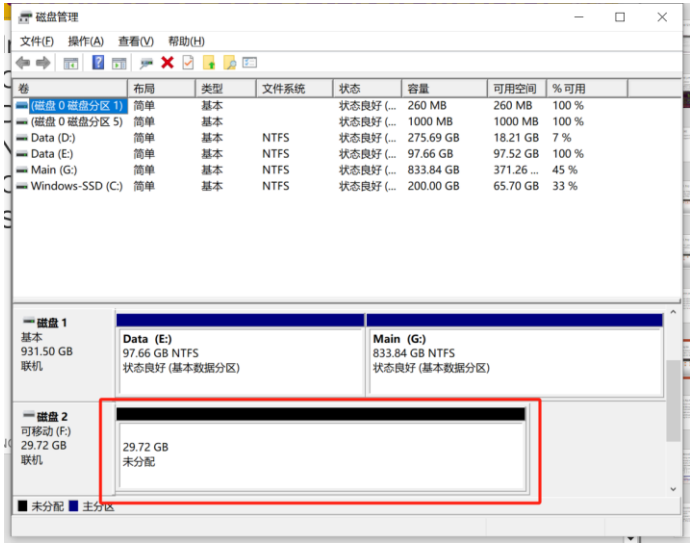


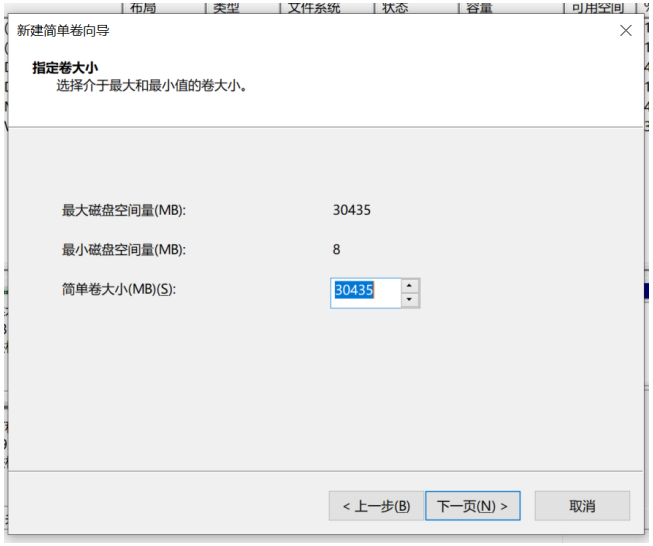


Flash file to SD card (Same as Before)

- Insert SD card into your computer
- Open **Disk Manager**
- Delete all **volumes** on the disk
- New a single volume with the biggest accessible volume
- Open **Win32DiskImager**
- Select .img file and flash it into SD card

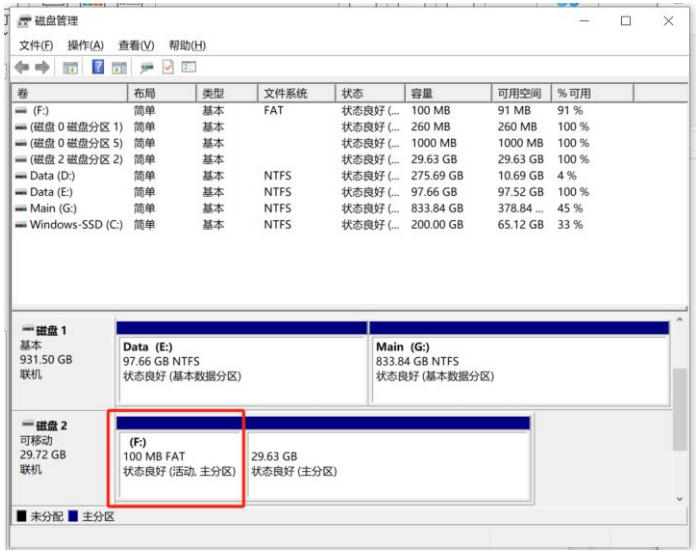


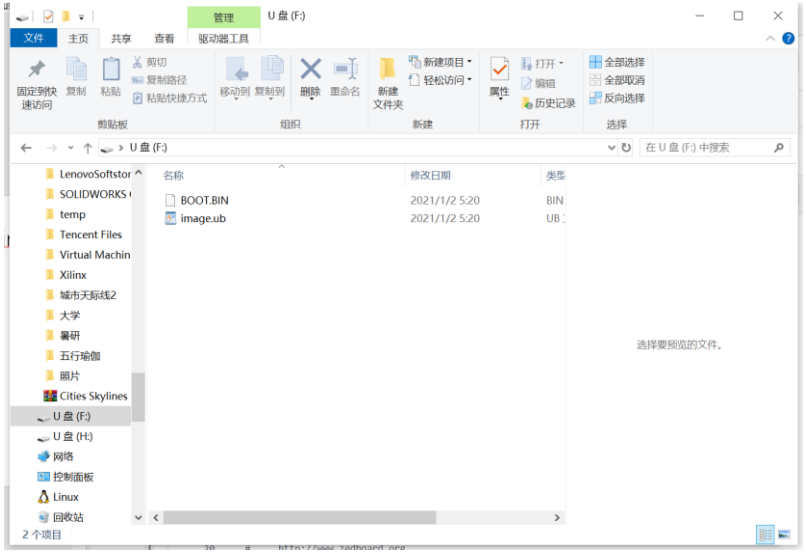




Change image.ub

- After burning, 2 volumes will be created.
- F: (smaller one) is accessible with **BOOT.BIN** and **image.ub** inside.
- If you face any unknown error, changing them with files inside output folder is one possible solution





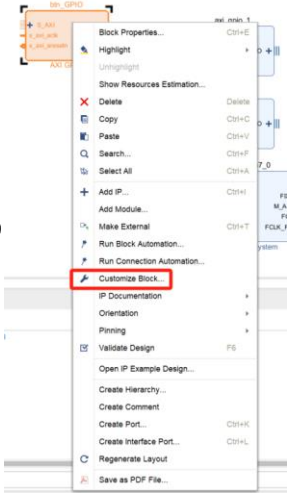
Create PYNQ Overlay

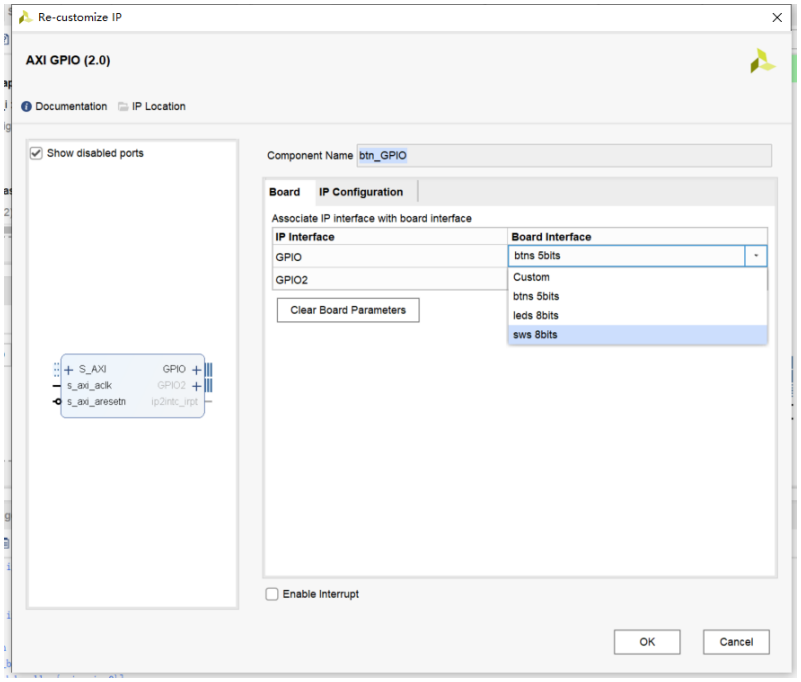
Objective: Support Bitstreams

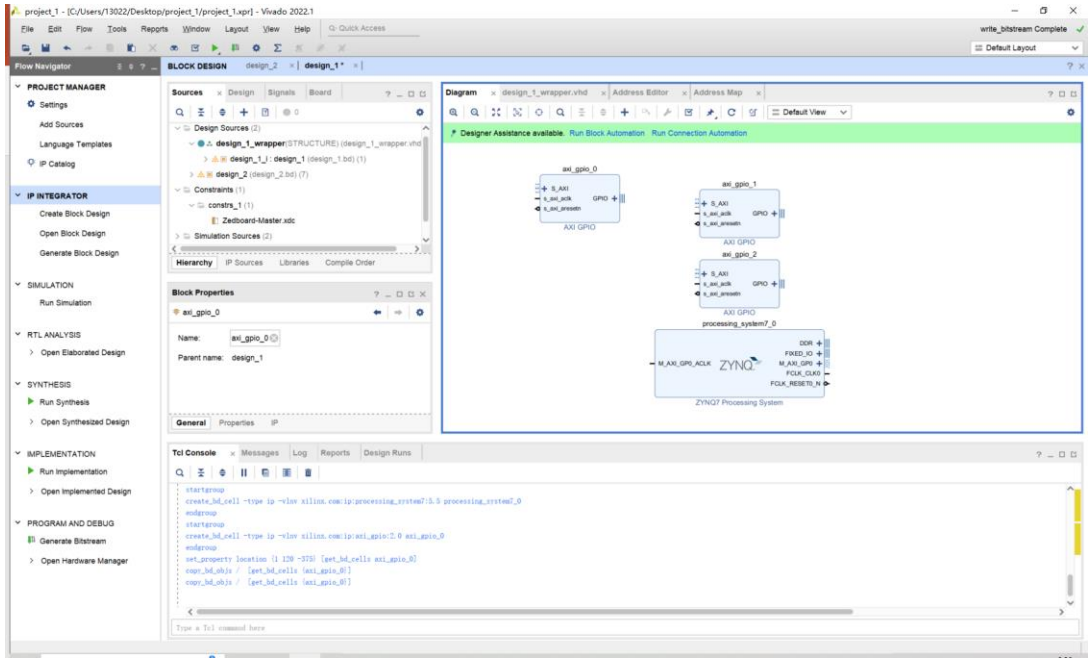
Output: **.bit** and **.hwh** files with Your Personal PYNQ Overlay

IP Integration

- Open Vivado and **Create a Block Deisgn** as Lab07
  - Only in this time, no need to create new IP block
- Add **Zynq Processing System** x1 and **AXI GPIO** x3
- Custom Blocks to **leds(8b), sws(5b),** and **btns(8b)**
- Rename the IP block as **btn**, **led**, and **sw**

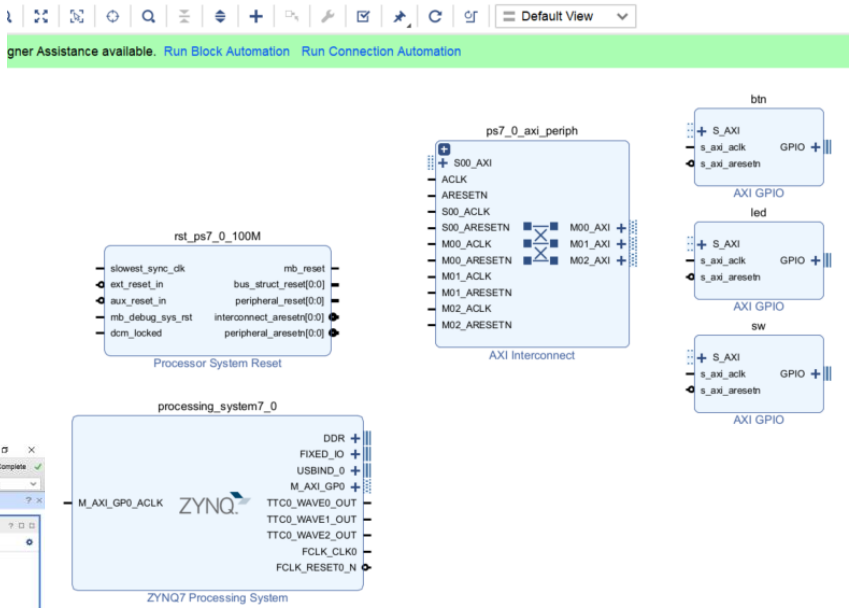


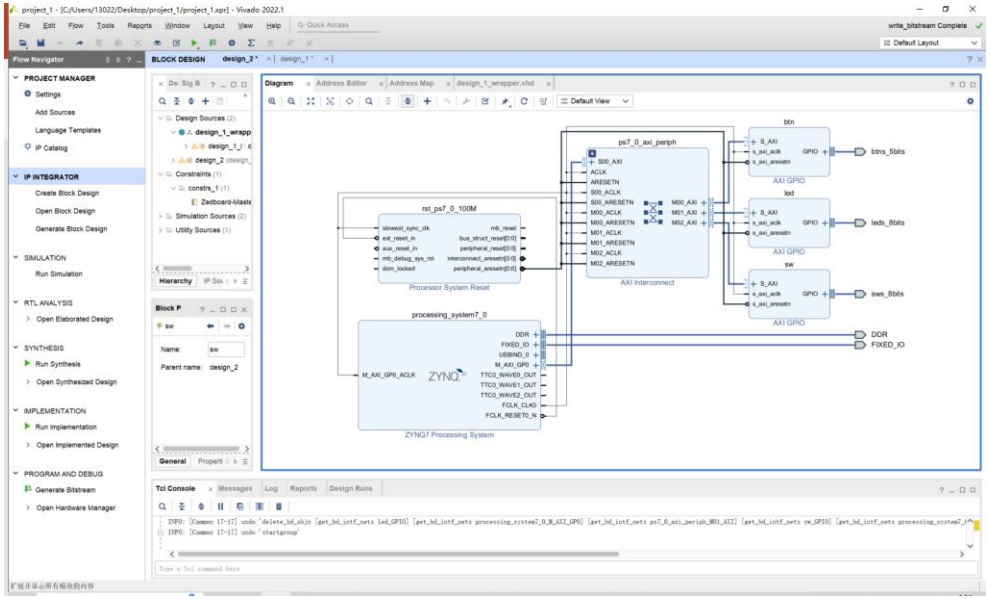




IP Integration

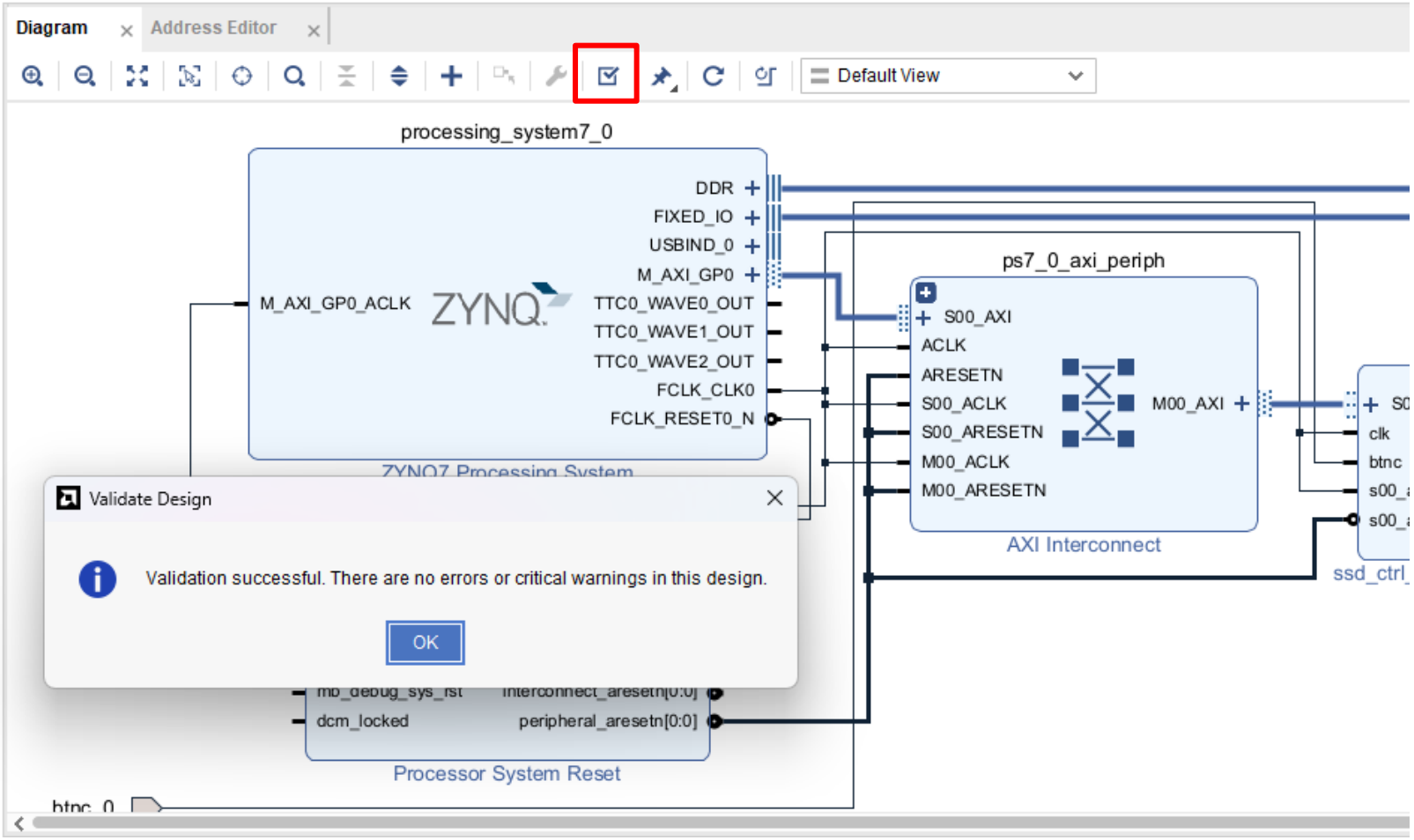
- Run **Block Automation** and Run **Connection Automation**





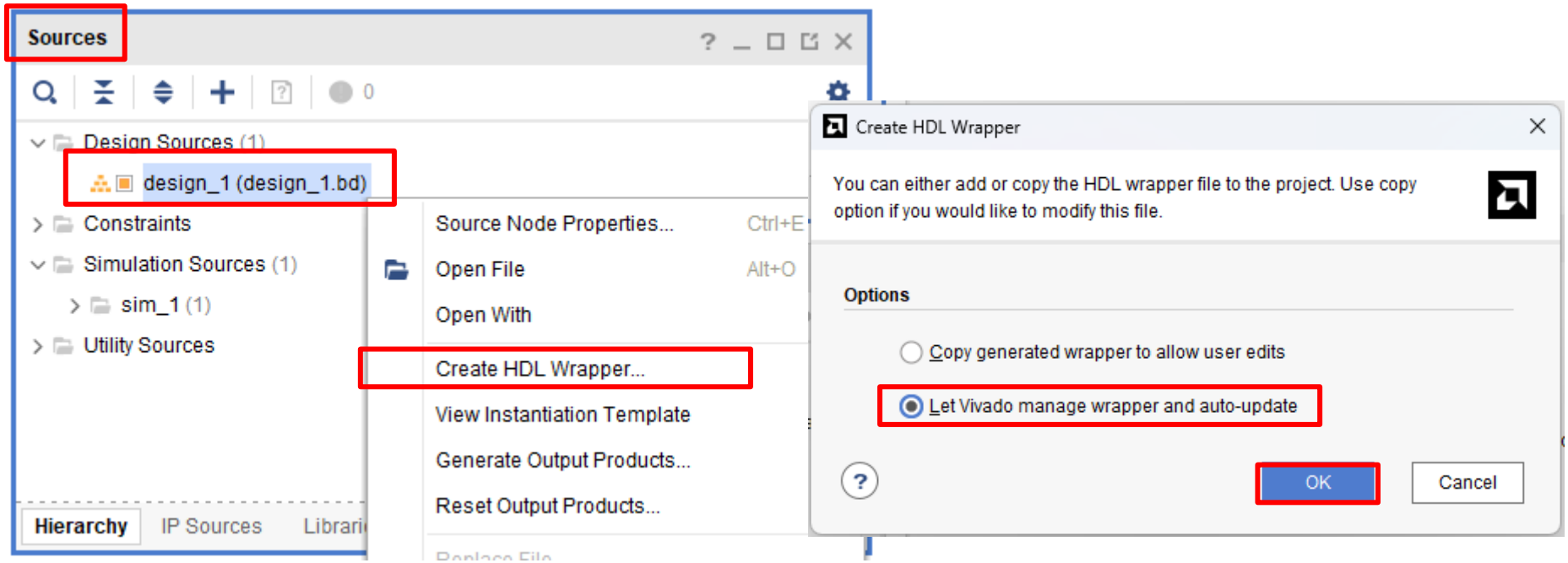
IP Integration

- Click **Validate Design**
- The result should have no errors



HDL Wrapper & Generate Bitstream

- Right click the **design_1** in **Design Source**
- Select **Create HDL Wrapper**



Export PYNQ Overlay

- **Export Hardware** (.xsa with .hwh file) and **Export Bitstream File** (.bit file)
- Change file name to be the same
- Upload them to **jupyter** under the same folder with .ipynb code