

# Laboratory Assignment 4

## Objectives

This lab gives you opportunities to create the models for different real-world scenarios for the car manufacturing industry, electrical engineering industry, and process capacity modeling. In this lab, you will also design a simple traffic controller using the Finite State Machine and implement it using conventional flip-flops, gates, 7-segment displays and component drivers.

## Grading

Refer to the section **Python Programming** for grading criteria.

## Bibliography

Refer to the lecture notes for sample programs.

## Requirements

### 1 - Numerical Integration – Head Injury Criterion (HIC)

Help your colleague from the mechanical department to design a car airbag to meet the Head Injury Criterion (HIC). You can choose to at least match the Mercedes Benz HIC = 310 or the 1995 Audi 8 HIC = 142. Use this equation from the websites below:

$$H_{t,d} = d \left( \frac{1}{d} \int_t^{t+d} a(T) dT \right)^{2.5}$$

Without the airbag

$$a(t) = \frac{16400}{(t - 68)^2 + 400} + \frac{1480}{(t - 93)^2 + 18}$$

With the airbag

$$a(t) = \frac{22000}{(t - 74)^2 + 500}$$

- <https://www.intmath.com/applications-integration/hic-head-injury-criterion.php>
- <https://www.intmath.com/applications-integration/hic-part2.php>

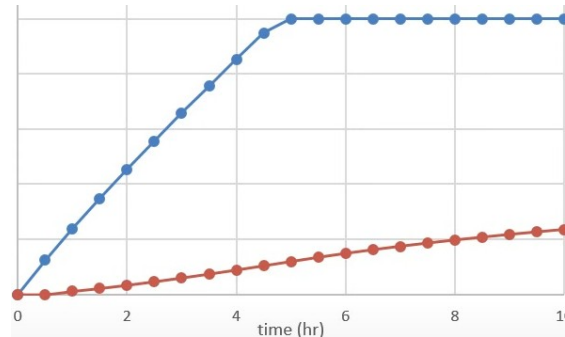
## 2 - Differential Equation – ER Modeling

Model a hospital ER (Emergency Room) with a general ERU (ER Unit) and an ICU (Intensive Care Unit) specializing in COVID-19 treatment with respiratory equipment **using the ODEINT method**.

For ER, the variables can be Number of Nurses, Number of Doctor, Number of beds, available slots for diagnosis equipment such as X-ray machine, CT Scanners, Number of Patients coming into the ER, Number of Patients being discharged vs being transferred to the ICU, death rate, etc. Your model should have at the minimum 3 variables.

For ICU specializing in COVID-19 treatment, again the variables can be Number of Nurses, Number of Doctor, Number of beds, Number of ventilators, Number of Patients coming into the ER, Number of Patients being discharged vs being transferred to the ICU, death rate, etc. Your model should have at the minimum 3 variables.

You will turn in your model along with different scenarios and graphs, including a graph that shows ER being saturated, ICU being saturated, and both ER & ICU being saturated. Show each scenario with the values of your program variables that cause that condition to occur.



You can leverage the concept and **ODEINT** code from this lecture below and provide your own model for the ERU and the ICU. In the model below, the Water Tank 1 can be your ERU and Water Tank 2 can be your ICU. <http://apmonitor.com/che263/index.php/Main/PythonDynamicSim>

## Python Programming

**Lab Submission:** see *Laboratory Hand-In Requirements* section below

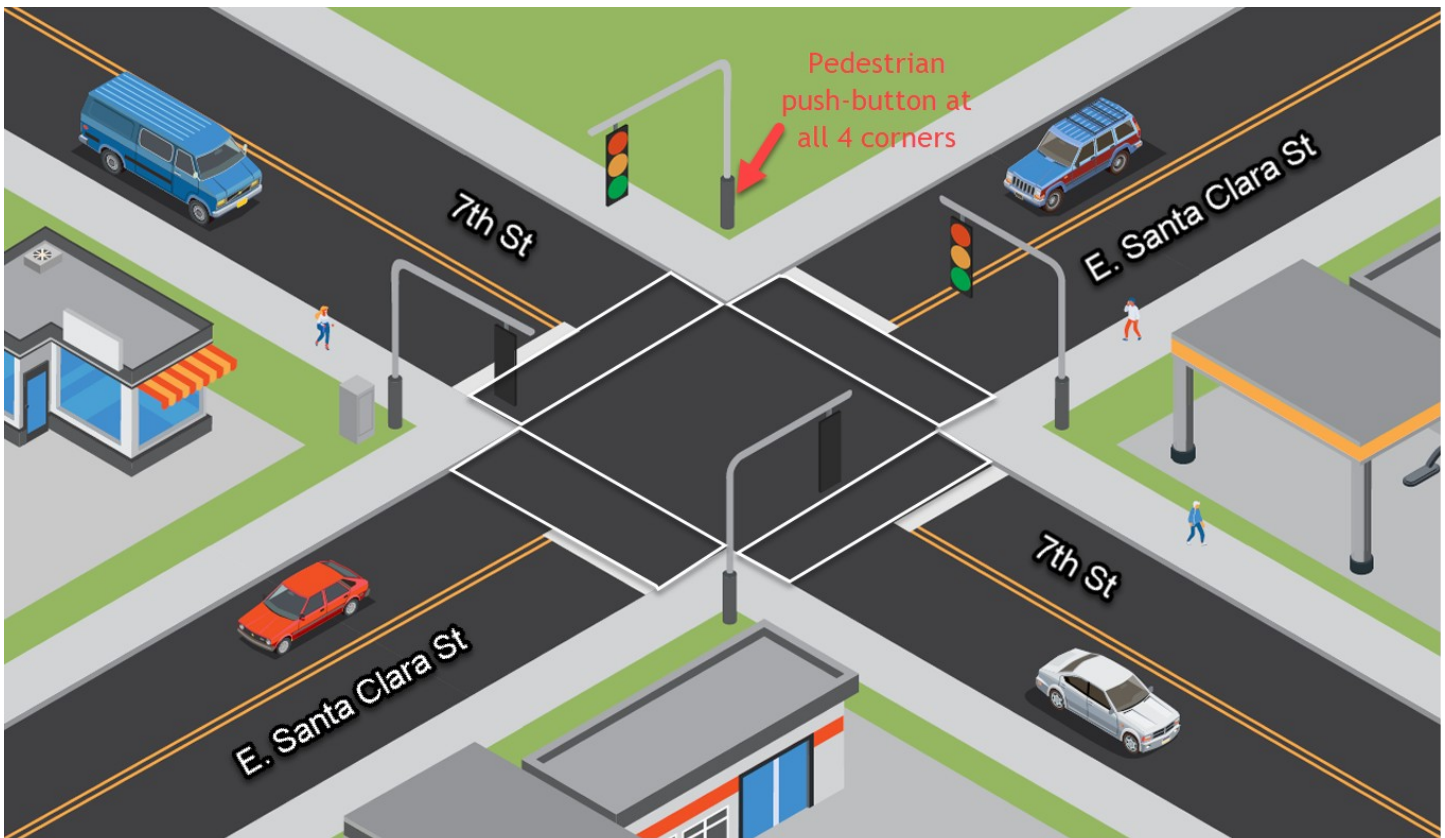
Once you learn the process and the code associate with each step in the process, you will be able to customize the program to do the followings.

Program or Requirement	Use Case	Earned Score / Max Score
Lab Report	<p><b>Turn in lab report (using the group report template) and the video recordings of all your work with your own voice narration for each requirement below and Submit Lab#_TeamName.Zip file on time to Canvas</b></p> <p>Design documentation:</p> <ul style="list-style-type: none"> <li>- Explain your ER model that includes ERU and ICU</li> <li>- Explain the HIC equation that you are using and defend why your answer is the best.</li> </ul>	____ / 10
Numerical Integration - HIC	Car airbag modeling. Explain the HIC equation that you are using and defend why your answer is the best.	____ / 10
Differential Equation – ER Modeling with 3+ variables	Production capacity modeling. You must show test scenarios for scenarios when: Both ERU and ICU are saturated. ERU 10/20, ICU 10/20	____ / 20
Simple Traffic Controller: Part 1, Hardware	Simple Traffic Controller: Part 1, Hardware Cut out a piece of paper to represent Santa Clara & 7 <sup>th</sup> Street. Place the LED lights at the intersection and the pedestrian button at the corner of 7 <sup>th</sup> street. Activate the pedestrian button to trigger the finite state machine state transition to demonstrate that the traffic lights are displaying properly along with the 7-	____ / 20

	segment for the walking lights. Also see notes below.	
Simple Traffic Controller: Part 1, Software	Simple Traffic Controller: Part 2, Software	____ / 25
Game development and Testing: Follow the Numbers	Entertaining Industry, Education Add hacks and tweaks: More dots, Level Up, No more chances, Multiple sets of dots, In record time (3 points each)	____ / 15
	<b>TOTAL</b>	<b>100%</b>

That's all for this lab. Hopefully you found it useful and increase your interest in the Python world! See you in the next lab.

## Simple Traffic Controller: Part 1, Hardware



East Santa Clara Street has the priority, i.e. it always has green light and pedestrian walkways along E. Santa Clara St. is always allowed, **except** for when there is a request for pedestrian crossing from the 7<sup>th</sup> Street or a car coming from 7<sup>th</sup> Street.

You will be given the following hardware to implement your hardware controller portion:

LED	For traffic signals.
7-Segment	For a count-down pedestrian signal along the 7 <sup>th</sup> Street
Push-button	To accept pedestrian's input request
74LS47	BCD to 7-Segment Decoder
74LS193	Binary Up/Down Counter with Clear
74LS90	Decade Counter
555 Timer	For use as clock
Resistors	For connecting LEDs, 7-Segments, and other open-collector signals
Capacitors	

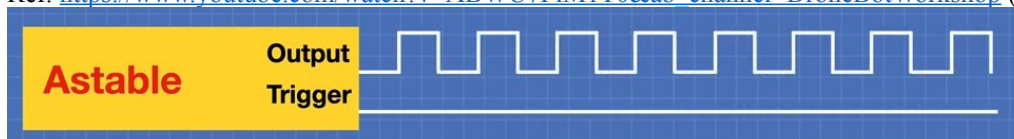
You will supply your own:

Breadboard	You will reuse your own breadboard from your previous courses
Power/Ground	Power/Ground from a battery or a power source. Use $3 \times 1.5V = 4.5V$

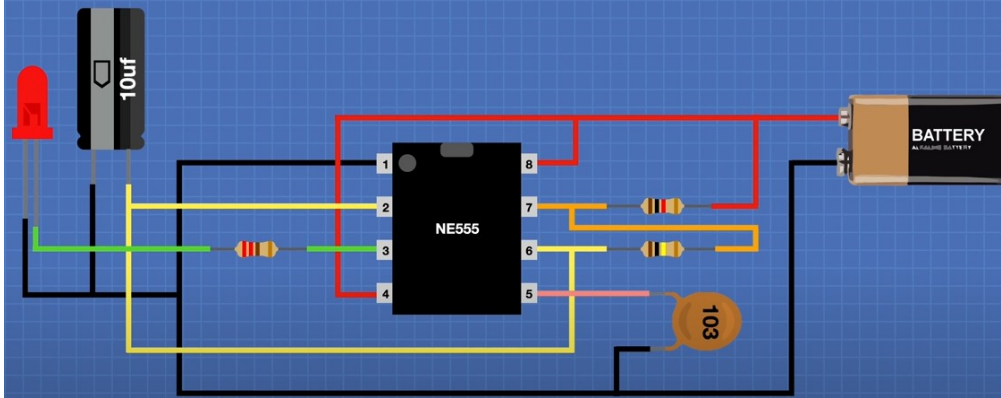
74-Series datasheet: <https://www.futurlec.com/IC74LS00Series.shtml>

Clock generation using 555 chipset using Astable Mode:

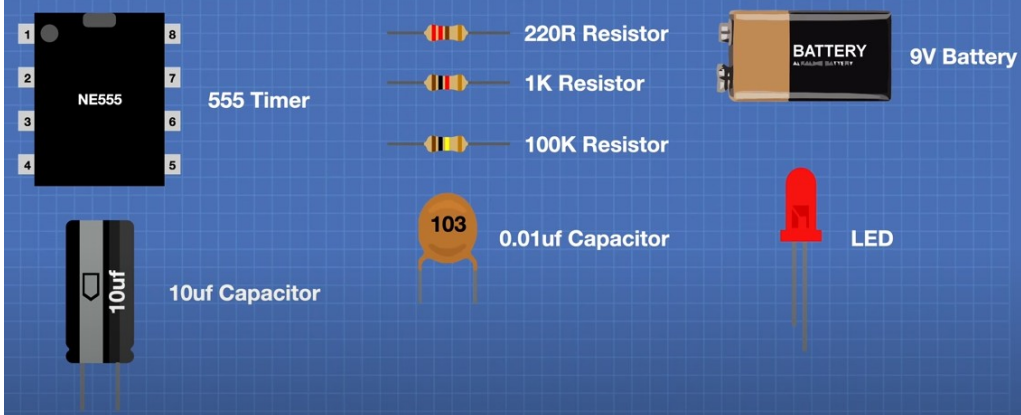
Ref: [https://www.youtube.com/watch?v=ABWU7FIM1T0&ab\\_channel=DroneBotWorkshop](https://www.youtube.com/watch?v=ABWU7FIM1T0&ab_channel=DroneBotWorkshop) (time=10:20 )



# Astable Mode



# Astable Mode

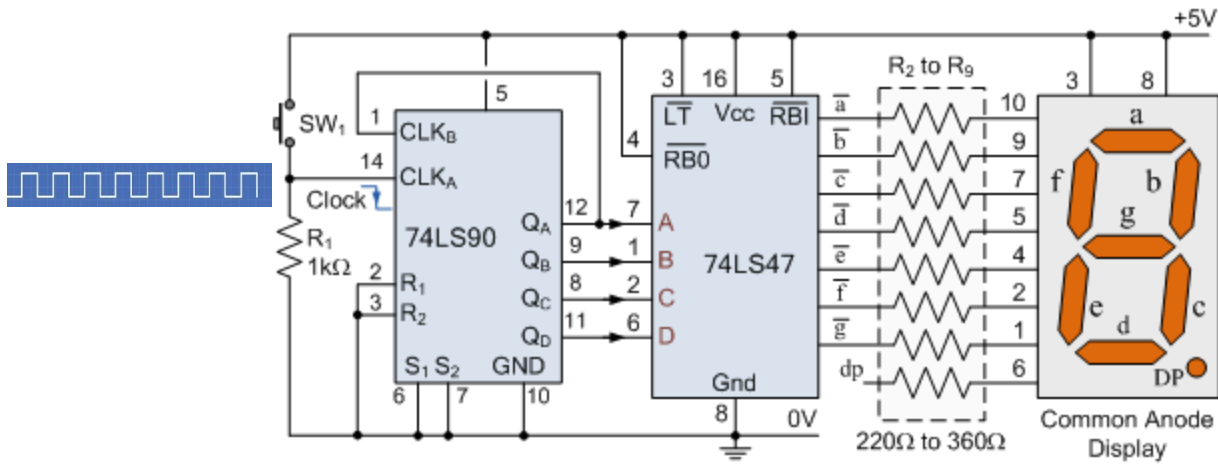


## 7-Segment circuit using clock, decade counter, BCD-to-7segment Decoder, and the 7-Segment Display:

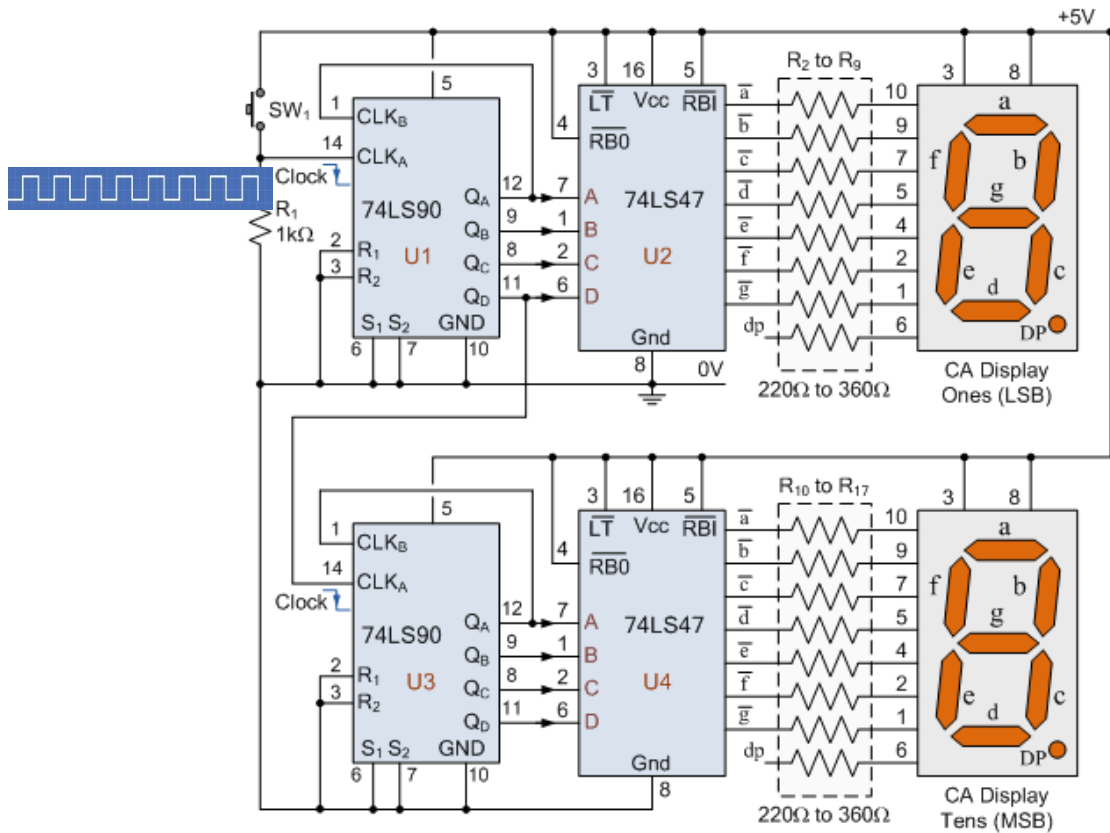
One-digit Ref: [https://www.youtube.com/watch?v=XCJqoae4hgY&ab\\_channel=element14presents](https://www.youtube.com/watch?v=XCJqoae4hgY&ab_channel=element14presents)

One and Two-digit Ref: <https://www.electronics-tutorials.ws/counter/7-segment-display.html>

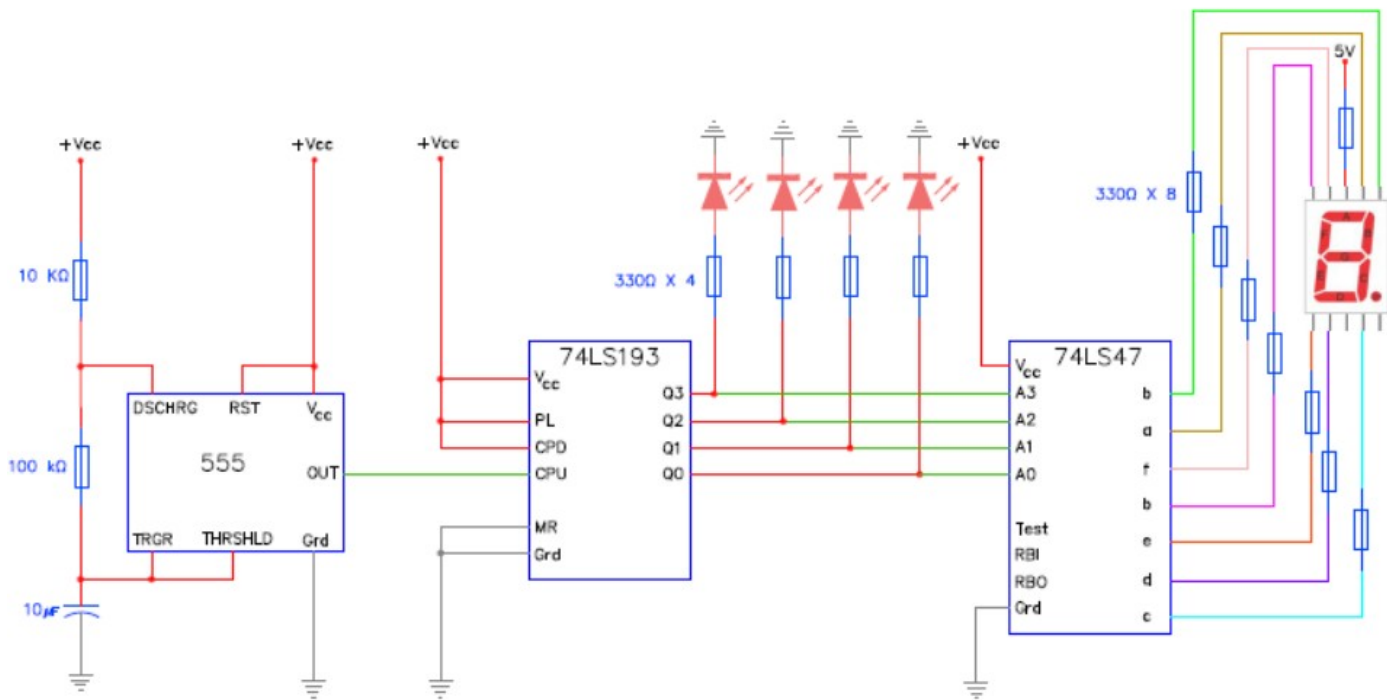
### Single Digit 7-segment Display Counter



### Two Digit 7-segment Display Counter



Alternatively, using 74LS193 up/down counter: [https://www.fwdskillzone.com/digital\\_logic\\_design.html](https://www.fwdskillzone.com/digital_logic_design.html)

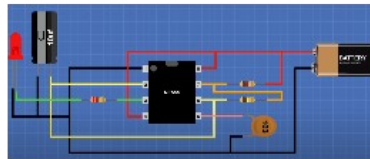


### State Machine Design:

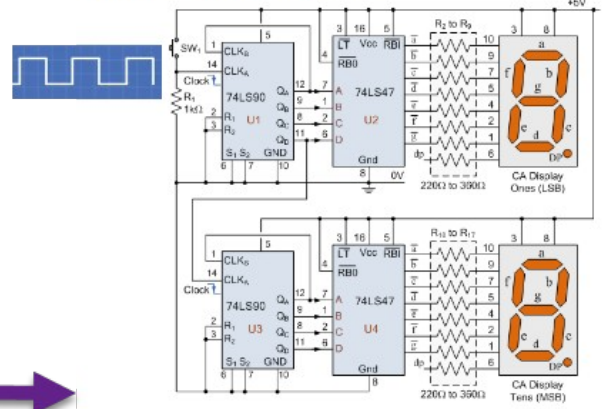
You can refresh your state machine circuit design using your old notes from previous classes or from this site showing a 4-state Mealy FSM implementation: [https://www.cpsc.ucalgary.ca/custom/321\\_challenge/04%20Finite%20State%20Machines.html](https://www.cpsc.ucalgary.ca/custom/321_challenge/04%20Finite%20State%20Machines.html) for from this video clip: <https://www.youtube.com/watch?v=Z4Zz7n-Lj0g>

Note: Your design may be a combination of Mealy & Moore FSM.

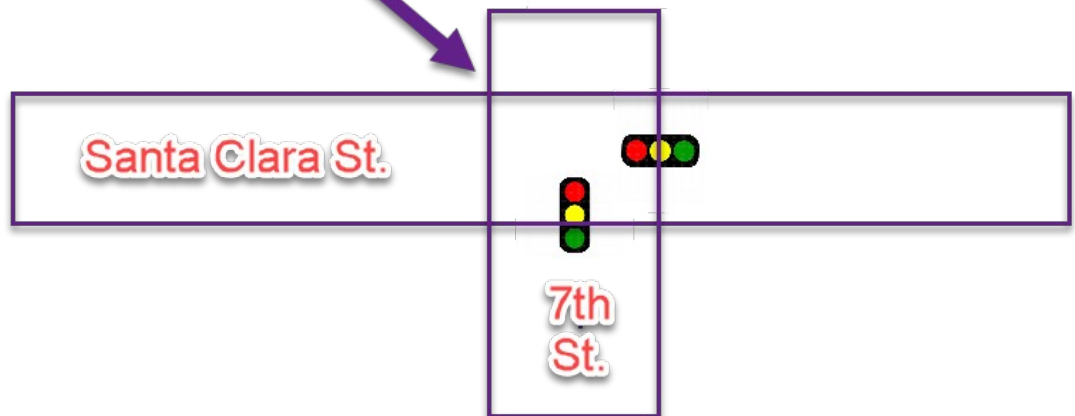
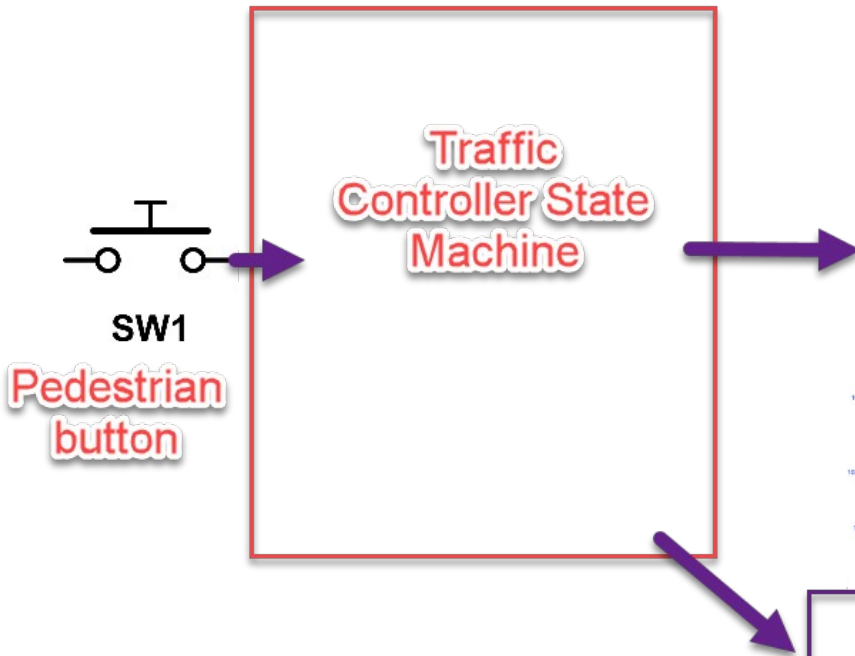
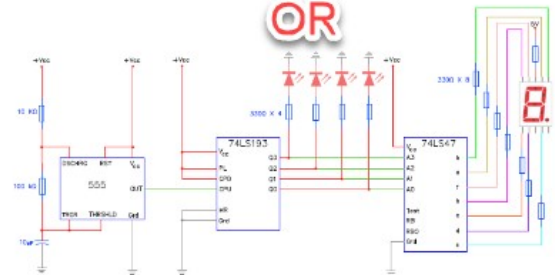




Two Digit 7-segment Display Counter



OR





# Connect Your Computer to the KV260 KRIA Board

## Download Tera Term or Putty

If you don't have TeraTerm or PuTTY, download and install it. Either one works fine.

- Tera Term download: [https://download.cnet.com/Tera-Term/3000-2094\\_4-75766675.html](https://download.cnet.com/Tera-Term/3000-2094_4-75766675.html)
- Putty download: [https://download.cnet.com/Putty/3000-7240\\_4-15129.html](https://download.cnet.com/Putty/3000-7240_4-15129.html)

## Connect computer to the KRIA board using the serial port

From your PC, use Device Manager => Ports: look for COM port. (My computer shows COM5 & COM6)

Set the following parameters:

Baud rate = 115200  
Data bits = 8  
Stop bits = 1  
Flow control = None  
Parity = None

=====  
KRIA KV260 username: ubuntu, password: ee104sjsu  
(on a new board, the original password is also ubuntu)

## Check the KRIA board IP Address using these command:

=====  
KRIA KV260 Board  
=====

```
ubuntu@kria:~$sudo netplan apply
```

```
---
```

```
ubuntu@kria:~$ip addr show dev eth0
```

```
ubuntu@kria:~$ping 8.8.8.8
```

=> verify that you can ping

If you cannot ping, then do the following steps:

```
ubuntu@kria:~$sudo nano /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg
```

```
network: {config: disabled}
```

```
---
```

#####NOTE: the indentation of any file in Linux is very important, especial for \*.yaml files and  
##### other configuration files.

##### You should show the file below in NotePad++ to see the indentations.

```
ubuntu@kria:~$sudo nano /etc/netplan/01-netcfg.yaml
```

```
network:
```

```
  version: 2
```

```
  renderer: networkd
```

```
  ethernet:
```

```
    eth0:
```

```
      dhcp4: no
```

```
      addresses:
```

```
        - 192.168.0.104/24
```

```
      gateway4: 192.168.0.1
```

```
      nameservers:
```

```
        addresses: [192.168.0.1]
```

```
---
```

```
ubuntu@kria:~$sudo netplan apply
```

```
---
```

```
ubuntu@kria:~$ip addr show dev eth0
```

```
ubuntu@kria:~$ping 8.8.8.8
```

=> verify that you can ping

## Download Angry IP Scanner for debug

You may also want to use Angry IP Scanner later, download it from here: <https://angryip.org/download/#windows>  
Then you can use this tool to scan the IP addresses, and find out the KRIA IP address.

## Connect computer to the KRIA board from a browser

# NOTE:

# If your laptop does not have an ethernet RJ45 port, then purchase a USB-to-RJ45 converter. When you connect your PC to the KRIA KV260 board using the ethernet cable via the RJ45-to-USB conversion, the laptop will show a new Ethernet connection.  
# Configure that Ethernet port similarly to the method below as if you Ethernet is on board.

# Laptop: Configure your ethernet port as below

# IP Address: 192.168.0.1, Subnet Mask: 255.255.255.0, DNS server address 8.8.8.8

Configure Control Panel => Network and Internet => Network and Sharing Center ==> Change Adapter Setting => Right click on Ethernet => Property => Internet Protocol Version 4 (TCP/IPv4) => Click on the button "Property" ==> Select use the following IP address: ==> IP address 192.168.0.1, Subnet mask: 255.255.255.0 and Use the following DNS server address 8.8.8.8 => Click OK => click Close

#You may have to force the address to 192.168.0.1 using instruction from here

# <https://forums.tomshardware.com/faq/how-to-change-default-internet-connection-sharing-ip-address-range.1606758/>

#Now configure WIFI to be shared by devices connected to the PC Ethernet port:

Configure Control Panel => Network and Internet => Network and Sharing Center ==> Change Adapter Setting => Right click on WIFI ==> Property => Sharing: set to share using Ethernet ==> press the Settings button ==> Select the services running on your network that Internet users can access (pick a number, such as 1700) ==> Service Setting popup: name of the computer is your computer name, use TCP port 21

Now open a browser, type the following URL:

<http://kria:9090/lab>

password is "xilinx"

## Simple Traffic Controller: Part 2, Software

### Python hardware programming

Download the following files from this GitHub to your windows computer using the Open in GitHub Desktop method:

[https://github.com/cathalmccabe/PYNQ\\_tutorials/tree/master/ps\\_gpio/kria\\_kv260\\_example](https://github.com/cathalmccabe/PYNQ_tutorials/tree/master/ps_gpio/kria_kv260_example)

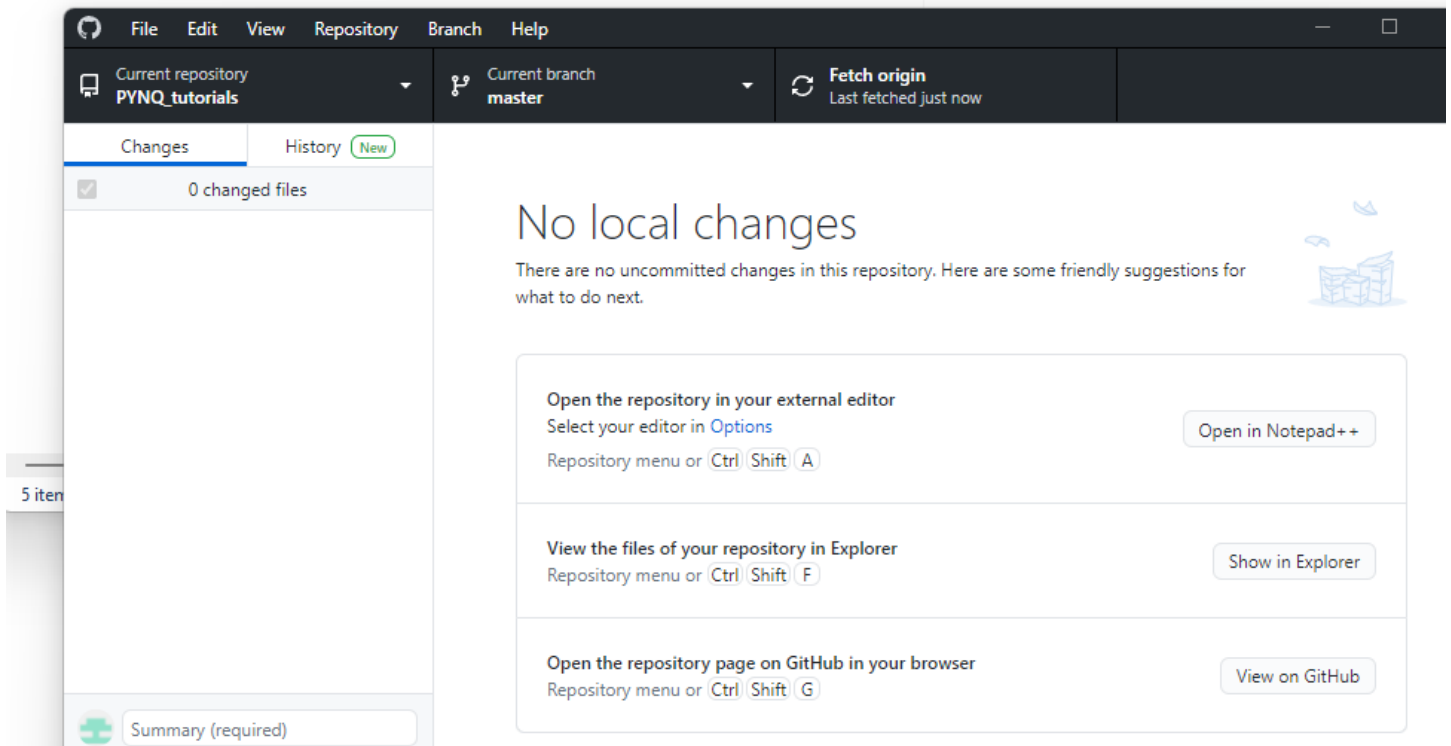
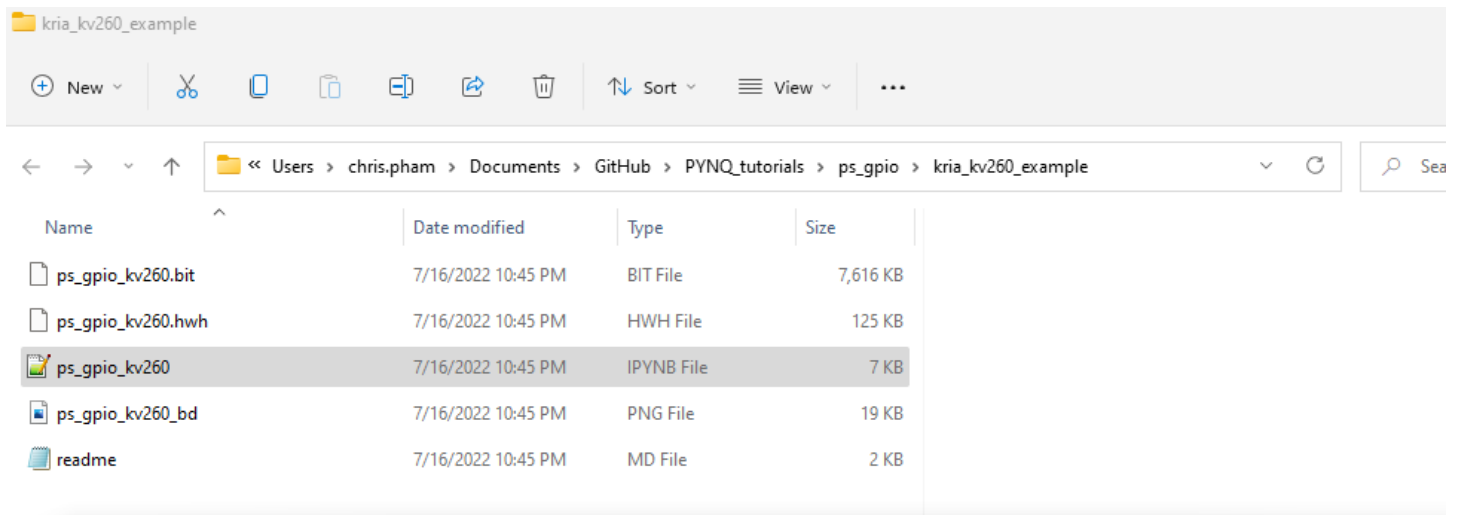
The image shows a GitHub repository page for `cathalmccabe / PYNQ_tutorials`. The repository is public and has 2 issues, 1 pull request, and 1 action. The file structure is as follows:

File	Description
<code>ps_gpio_kv260.bit</code>	Add PS GPIO example on KV260
<code>ps_gpio_kv260.hwh</code>	Add PS GPIO example on KV260
<code>ps_gpio_kv260.ipynb</code>	Add PS GPIO example on KV260
<code>ps_gpio_kv260_bd.png</code>	Add PS GPIO example on KV260
<code>readme.md</code>	Add PS GPIO example on KV260

A red arrow points to the `ps_gpio_kv260.ipynb` file with the text "click here".

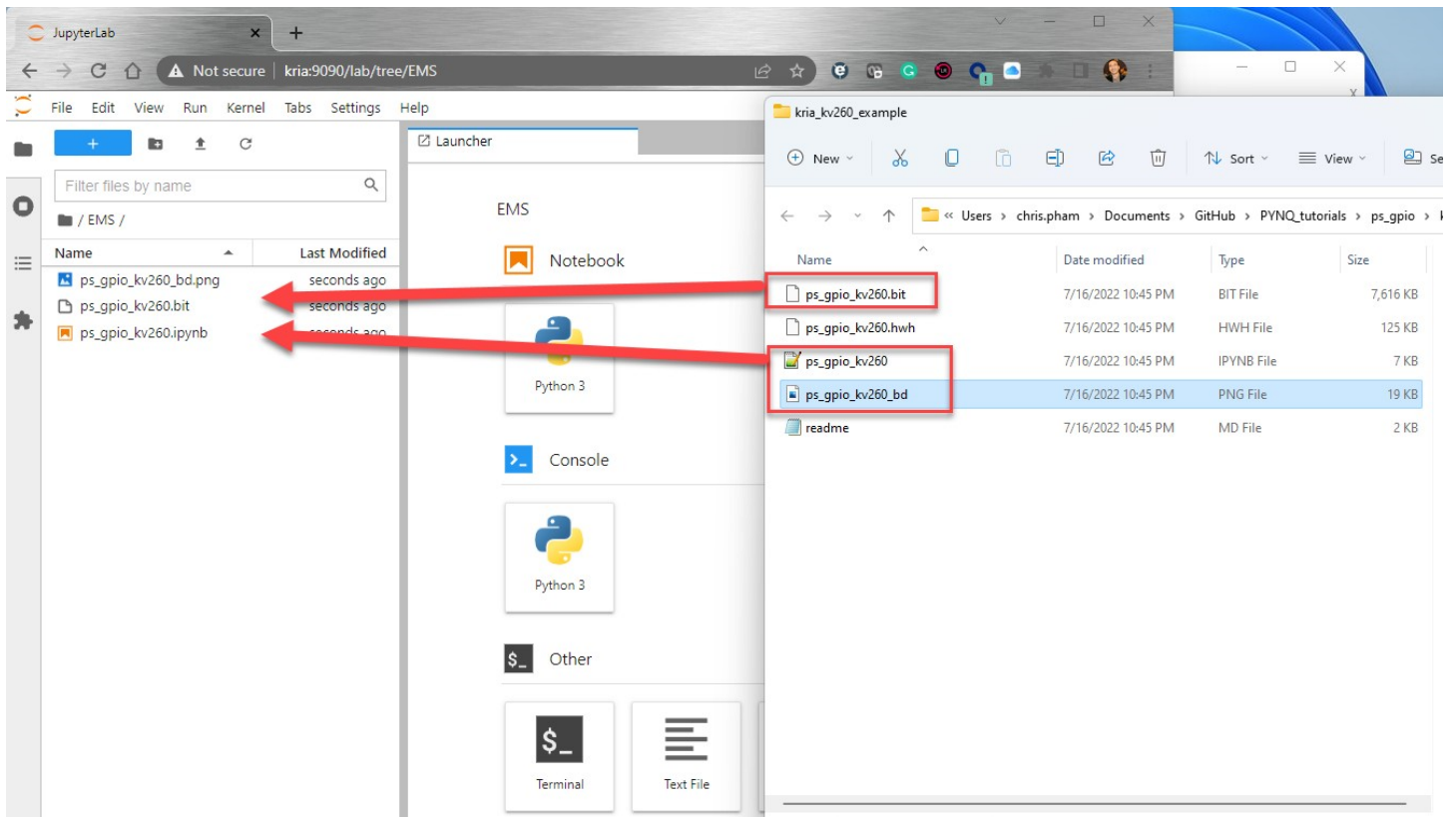
The `ps_gpio_kv260.ipynb` file is open, showing the title "Using PS GPIO with PYNQ" and the goal "Goal". The file has 320 lines (320 sloc) and is 6.62 KB. The commit history shows the latest commit by `cathalmccabe` on Sep 30, 2021.

In the bottom right corner, a red arrow points to the "Open in GitHub Desktop" option in the file actions menu.



Now, create a new folder EMS from the KRIA JupyterLab environment, and copy these 3 files by drag-and-drop method to the new EMS folder:

[ps\\_gpio\\_kv260.bit](#)  
[ps\\_gpio\\_kv260.ipynb](#)  
[ps\\_gpio\\_kv260\\_bd.png](#)



In the Jupyter directory, double click on the file [ps\\_gpio\\_kv260.ipynb](#)  
You will open the Jupyter Editor:

JupyterLab interface showing a notebook titled "Using PS GPIO with PYNQ". The notebook content includes:

## Using PS GPIO with PYNQ


### Goal

The aim of this notebook is to show how to use the Zynq PS GPIO from PYNQ. The PS GPIO are simple wires from the PS, and don't need a controller in the programmable logic.

Up to 96 input, output and tri-state PS GPIO are available via the EMIO in the Zynq Ultrascale+. They can be used to connect simple control and data signals to IP or external Inputs/Outputs in the PL.


### Hardware

This example uses a bitstream that connects PS GPIO to the Pmod on the KV260.

 PS GPIO Design

### External Peripherals

An LED, a Slider switch and a Buzzer are connected via the Pmod connector and a Grove adapter. These will be used to demonstrate the PS GPIO are working.



### Download the tutorial overlay

The `ps_gpio_kv260.bit` and `ps_gpio_kv260.hwh` files are in the `ps_gpio` directory local to this folder. The bitstream can be downloaded using the `PYNQ Overlay` class.

```
[ ]: from pynq import Overlay
ps_gpio_design = Overlay("./ps_gpio/ps_gpio_kv260.bit")
```

### PYNQ GPIO class

The PYNQ GPIO class will be used to access the PS GPIO.

```
[ ]: from pynq import GPIO
```

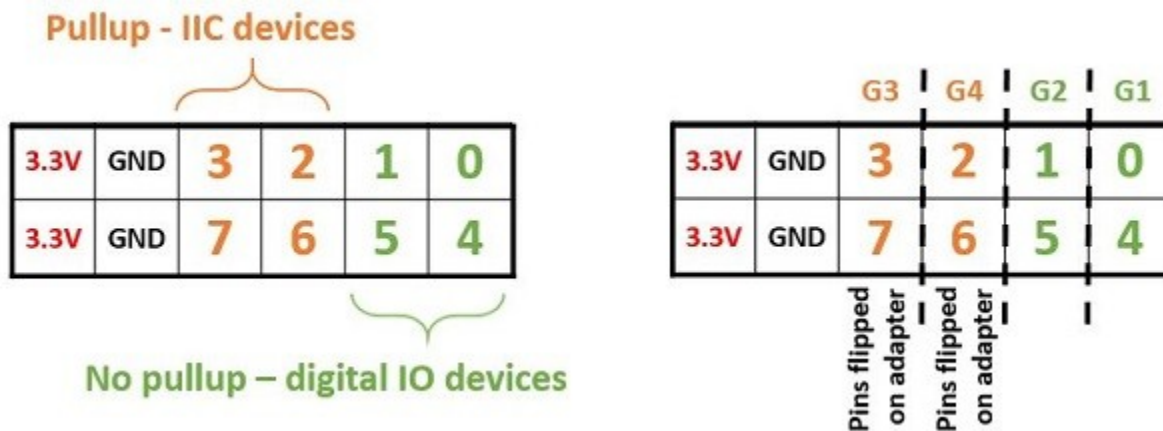
### GPIO help

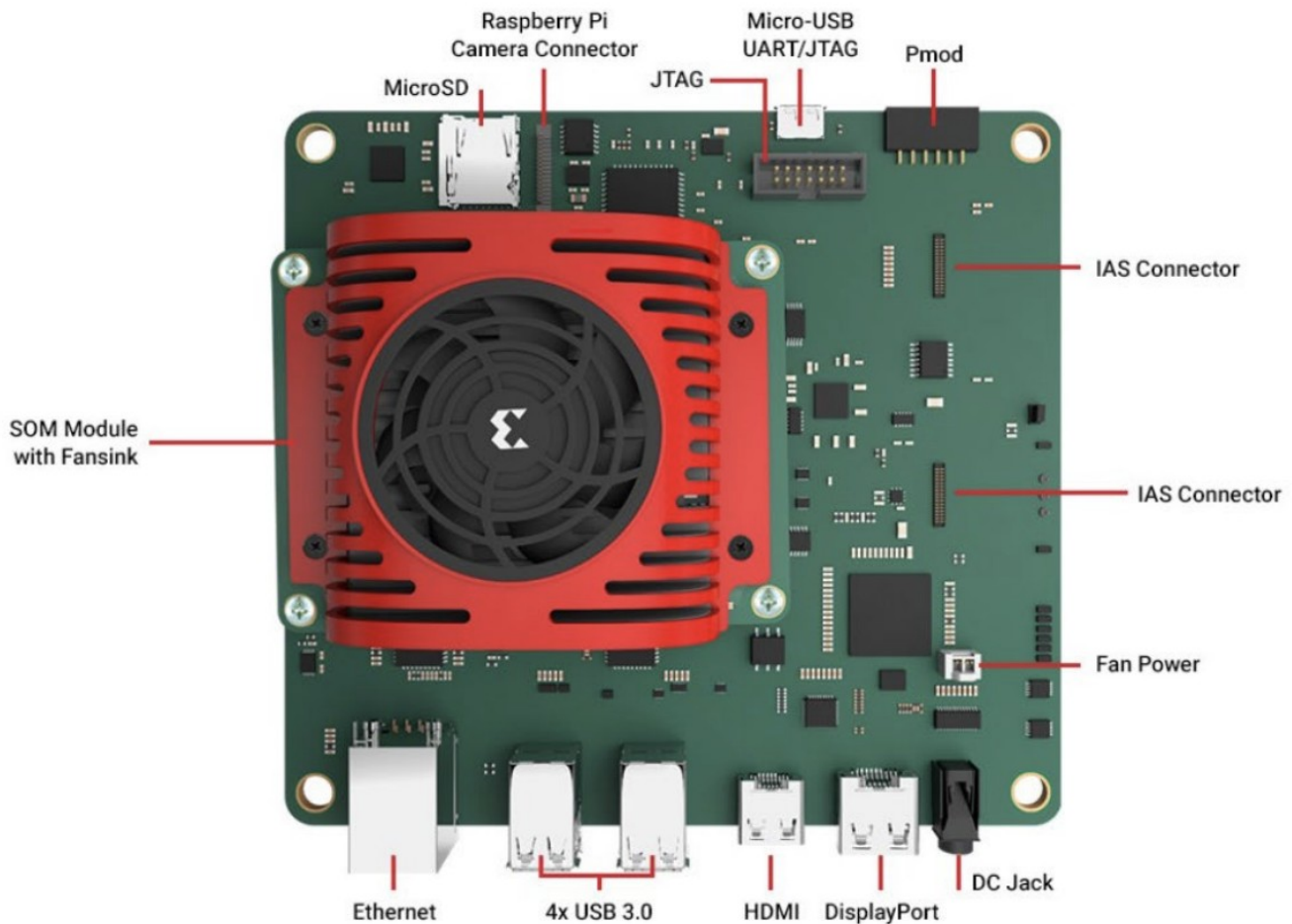
```
[ ]:
```

### Create Python GPIO objects for the led, slider and buzzer and set the direction:

```
[ ]: led = GPIO(GPIO.get_gpio_pin(6), 'out')
buzzer = GPIO(GPIO.get_gpio_pin(0), 'out')
```

This is the PMOD pinouts





According to the code below, connect the PMOD pins to the LED, Buzzer, and the Slider.

## PYNQ GPIO class

The PYNQ GPIO class will be used to access the PS GPIO.

```
[ ]: from pynq import GPIO
```

## GPIO help

```
[ ]:
```

```
### Create Python GPIO objects for the led, slider and buzzer and set the direction:
```

```
[ ]: led = GPIO(GPIO.get_gpio_pin(6), 'out')
      buzzer = GPIO(GPIO.get_gpio_pin(0), 'out')
      slider = GPIO(GPIO.get_gpio_pin(1), 'in')|
      slider_led = GPIO(GPIO.get_gpio_pin(5), 'out')
```



**PMOD  
pins**



## Wiring:

**LED1:** Positive terminal connects to PMOD pin 6. Negative terminal connects to a 1K-Ohm resistor then to GRD.

**Buzzer:** According to this spec, <https://wiki.seeedstudio.com/Grove-Buzzer/>, connect Red wire to VCC, Black wire to GND, and Yellow wire to PMOD pin 0.

**Slider:** According to this spec, [https://wiki.seeedstudio.com/Grove-Slide\\_Potentiometer/](https://wiki.seeedstudio.com/Grove-Slide_Potentiometer/), connect Red wire to VCC, Black wire to GND, and Yellow wire to PMOD pin 1.

**Slider LED2:** You can connect the positive terminal of LED2 to PMOD pin 5. Connect negative LED 2 terminal to a 1K-Ohm resistor then to GND.

## Code Execution:

Beginning from the first cell of the Jupyter Notebook, click on the PLAY button to execute every single cell from top down.

Experiment by changing the pin out from the 4<sup>th</sup> cell and re-run that cell. Experiment by changing code from other cells and run it.

## Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. **You are required to submit a video to demonstrate the process of data conversion and show that you achieved a working design.** You are also required to submit an archive of your project in the form of a ZIP file. Use 7-Zip option to create the ZIP file. Name the archive **lab#\_GroupName.zip**. Refer to Lab 1 for detail instructions.

You will submit your zip file to the instructor through Canvas by the due date and time. If your program is not completely functional by the due date, you should demonstrate and turn in what you have accomplished to receive partial credit. See the syllabus for the late penalty guideline