In [10]:
```python
import time
from pynq.overlays.base import BaseOverlay
import socket

base = BaseOverlay("base.bit")
btns = base.btns_gpio
leds = base.leds
```

Sockets

This notebook has both a client and a server functionality. One PYNQ board in the group will be the client and SENDS the message. Another PYNQ board will be the server and RECEIVES the message.

Server

Here, we'll build the server code to LISTEN for a message from a specific PYNQ board.

When we send/receive messages, we need to pieces of information which will tell us where to send the information. First, we need the IP address of our friend. Second, we need to chose a port to listen on. For an analogy, Alice expects her friend, Bob, to deliver a package to our back door. With this information, ALICE (server ip address) can wait at the BACK DOOR (port) for BOB (client ip address) to deliver the package.

Format of the information ipv4 address: ###.###.###.### (no need for leading zeros if the number is less than three digits) port: ##### (it could be 4 or 5 digits long, but must be >1024)

Use the socket documentation (Section 18.1.3) to find the appropriate functions https://python.readthedocs.io/en/latest/library/socket.html

In [ ]:
```python
#clientAddress = 192.168.8.162

# TODO:
serverPort = 1234
clientAddress = "192.168.8.162"  # PYNQ board IP

# Create TCP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 1: Bind the socket to the pynq board IP and port
sock.bind(('', serverPort))

# Start listening
sock.listen(1)
print("The server is ready to receive...")

while True:
    # 2: Accept connections
    connectionSocket, addr = sock.accept()
    print(f"Connected by {addr}")

    try:
        # 3: Receive bytes from the connection
        data = connectionSocket.recv(serverPort)

        if data:
            message = data.decode()

            # 4: Print the received message
            print("Received:", message)

            # Optional: send back uppercase version
            modifiedMessage = message.upper()
            connectionSocket.send(modifiedMessage.encode())

    finally:
        connectionSocket.close()
```

Client

Now, we can implement the CLIENT code.

Back to the analogy, now we're interested in delivering a package to our friend's back door. This means BOB (client ip address) is delivering a package to ALICE (server ip address) at her BACK DOOR (port)

Remember to start the server before running the client code

In [9]:
```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# TODO:
serverName = "192.168.0.167"   # server IP
serverPort = 12345

# 1: Connect the socket (sock) to the <SERVER-IP> and choosen port <LISTENING-PORT>
sock.connect((serverName, serverPort))

# 2: Send 5 messages
for i in range(5):
    msg = "WES 237A!"
    sock.send(msg.encode())

# 3: Close the socket
sock.close()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [9], in <cell line: 7>()
      4 serverPort = 12345
      6 # 1: Connect the socket (sock) to the <SERVER-IP> and choosen port <LISTENING
-PORT>
----> 7 sock.connect((serverName, serverPort))
      9 # 2: Send 5 messages
     10 for i in range(5):

KeyboardInterrupt: 
```

On your server, you should see the message and then the server will shutdown! When we close a socket, both the client and the server are disconnected from the port.

Instead, change the function above to send 5 messages before closing.

The pseudocode looks like this

- connect the socket
- for i in range(5)
    - msg = input("Message to send: ")
    - send the message (msg)
- close the socket