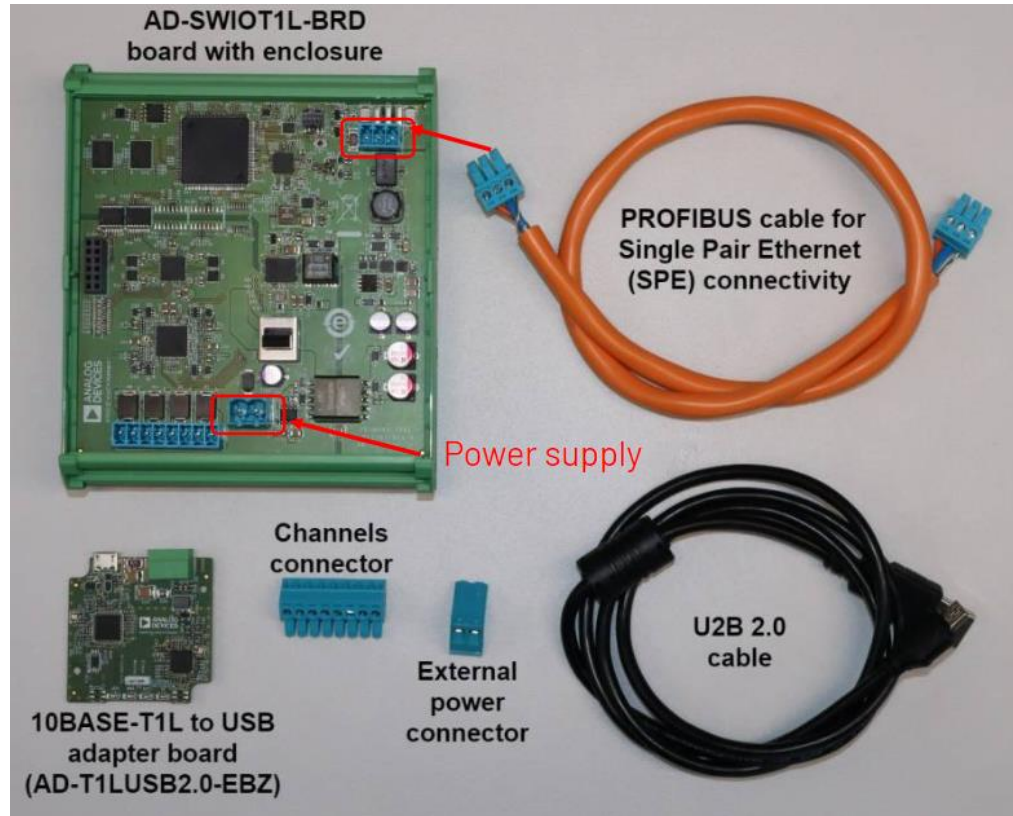


Applied Systems Control Workshop

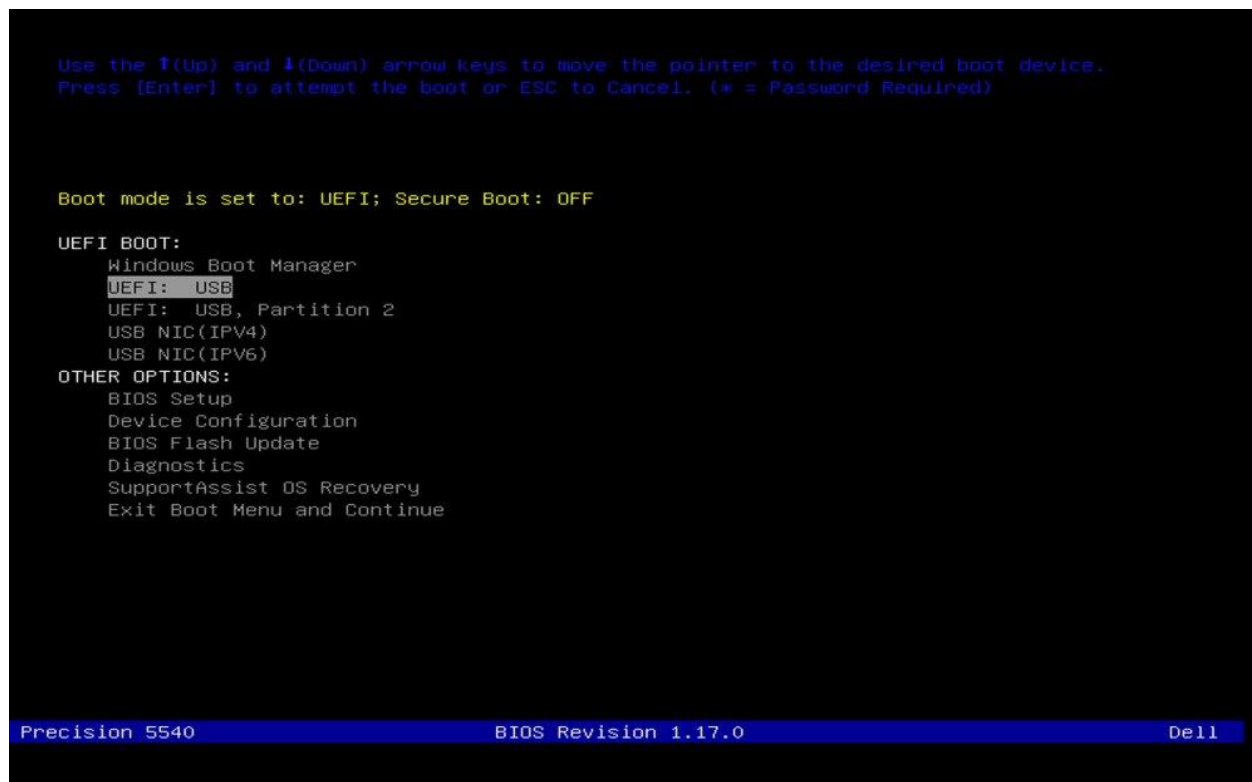
Workshop pre-requisites

Setup:

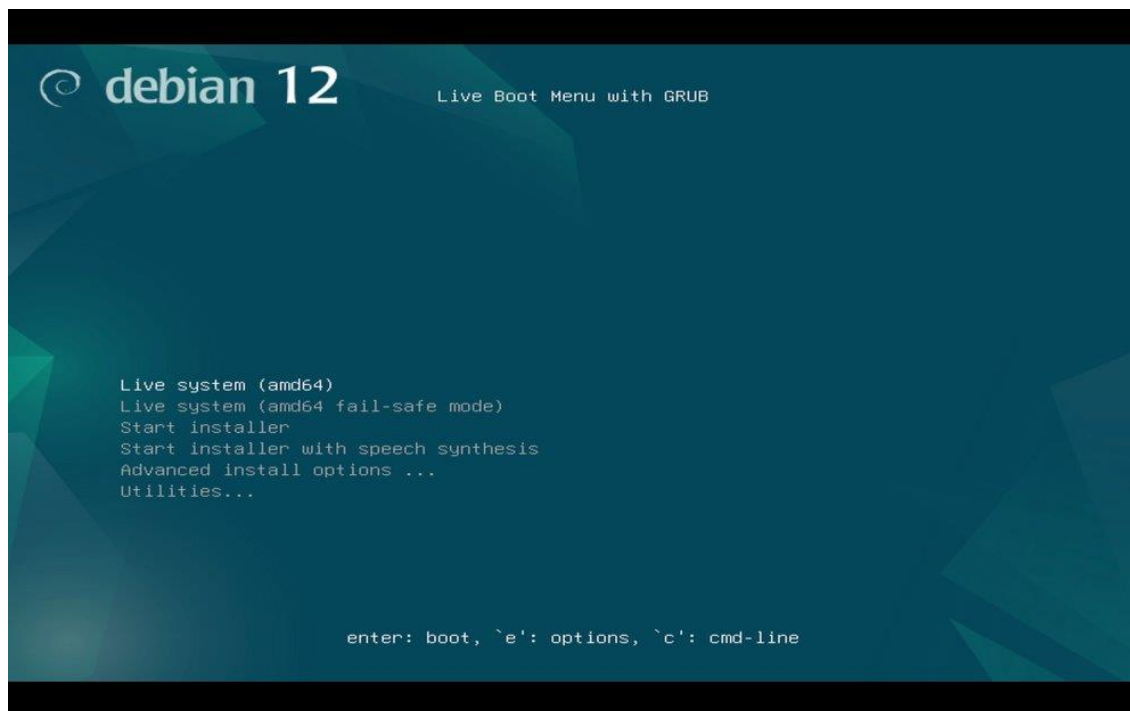
1. Power on the AD-SWIOT1L-SL board by plugging in the power cable from the power supply into the board's power connector – see the indication on the picture below.



2. Connect the USB to T1L media converter to your laptop using a **micro-USB cable**
3. Connect the USB to T1L media converter to the AD-SWIOT1L-SL board using a **PROFIBUS cable** – the orange one (see the picture above). After a short time, both link status LEDs (on the media converter and the AD-SWIOT1L-SL board) should be on.
4. Insert the USB stick into the PC USB port
5. Power on the computer and keep pressing F12 to enter the boot menu. The below or similar screen should appear once the boot menu is entered:



6. When prompted, select the **Live system (amd64)** and press the **ENTER** Key



Testing the board connectivity

The connection to the board can be tested by trying to run the *ping* command in the terminal. This usually rules out the host (PC) network configuration issues. Open a terminal and ping the board:

```
→ ~ ping 169.254.97.40
PING 169.254.97.40 (169.254.97.40) 56(84) bytes of data.
64 bytes from 169.254.97.40: icmp_seq=1 ttl=255 time=3.01 ms
64 bytes from 169.254.97.40: icmp_seq=2 ttl=255 time=1.57 ms
64 bytes from 169.254.97.40: icmp_seq=3 ttl=255 time=1.66 ms
^C
--- 169.254.97.40 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.574/2.082/3.011/0.657 ms
→ ~
```

Interrupt the ping process using CTRL+C keys combination

Hands-on exercises

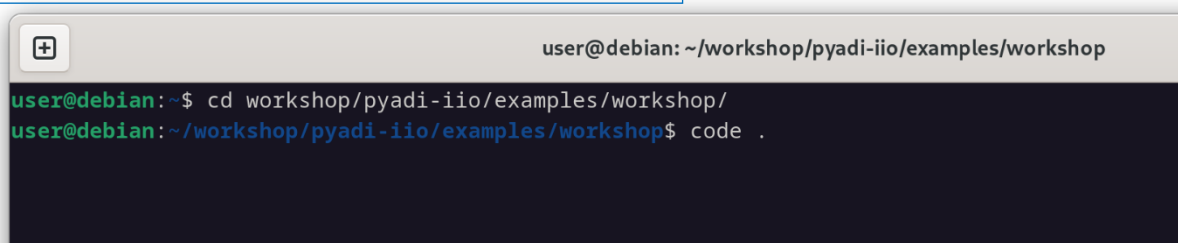
Example 1: During this exercise, students will connect and control an RGB LED using SWIOT board. The aim is to use Python to create a “for” loop that cycles through different color combinations by powering the RGB LED’s red, green and blue channels.

This example powers an RGB LED red, green and blue in a loop.

Steps

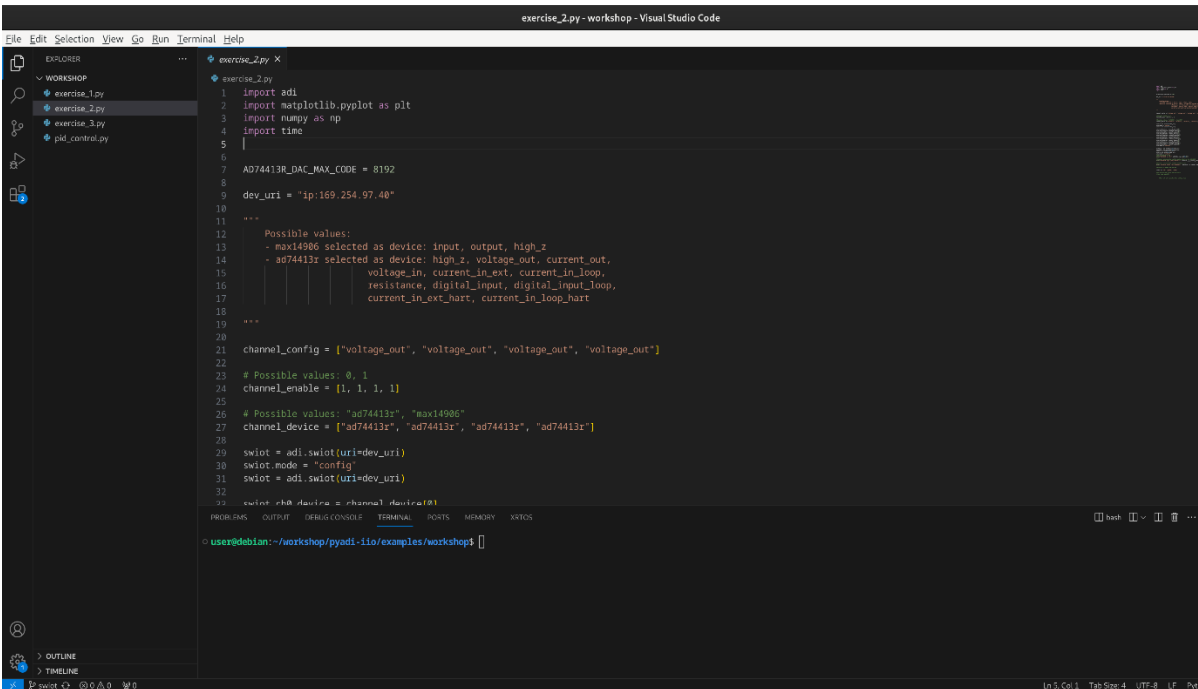
- Use the blue connector labeled as “1” and insert it into the board.
- Open a terminal from **Activities** menu – upper left corner of the screen, search for Terminal there
- You are in the ~home directory, so move to the workshop project location using the command:

```
$ cd automation_workshop/pyadi-iio/examples/workshop
```




The command “code .” will open VSCode editor.


Here you have **exercise_2.py** where you will write your code.



You are provided with an object **ad74413r** that controls voltage channels on the LED. Each channel is identified by "voltage0", "voltage1", and "voltage2". Your goal is to write a loop that will:

1. Set the raw value of each voltage channel (voltage0, voltage1, and voltage2) to 8000, one at a time e.g. **ad74413r.channel["voltage0"].raw = 8000**
2. After setting each channel's voltage, pause for 1 second.
3. Once all channels have been set, turn off all the channels by setting their raw values back to 0.





Tips: Use a **“for loop”** to iterate over the three voltage channels. Inside the loop, set the raw value for each channel to 8000 and then wait for 1 second. After the loop completes setting the voltages, use another for loop to reset the raw values of all channels to 0.

d. After writing your code, run the python file in the terminal:

```
$ ~/pyadi-iiio/examples/workshop> python3 exercise_2.py
```

Example 2: In this exercise, students will learn to control the brightness of an LED by adjusting a potentiometer. Using an analog-to-digital converter (ADC) on the microcontroller, they will read the potentiometer's value and assign it to the LED's brightness level

This example uses a potentiometer to dim the light of an LED.

Steps

- Use the blue connector labeled as "1" and insert it into the board.
- In the same directory, the code for the example can be found in ***exercise_3.py***.

```
while True:
    for channels in ad74413r._rx_channel_names:
        # Get the potentiometer's resistance value in Ohms
        resistance = ad74413r.channel["resistance3"].processed / 10000
        red_channel_val = (resistance / 11000) * AD74413R_DAC_MAX_CODE

        # Your code goes here
        # Assign the value of potentiometer value to the adc channel
```

- Convert and Assign the Value: Write the code to convert the value stored in "red_channel_val" to an integer, and assign it to the raw attribute of the "voltage0" channel
- Now, run the python file: ***python3 exercise_3.py***
- Slide the potentiometer and see how the LED increases or decreases its intensity.

Example 3: During this exercise, students will learn how a PID control system manages the speed of a fan based on temperature readings. They will allow the user to adjust the fan's speed by making the "pwm_output" variable take user input.

This example uses a closed control loop for temperature adjustment using a fan.

The code for this example is ***pid_control.py***

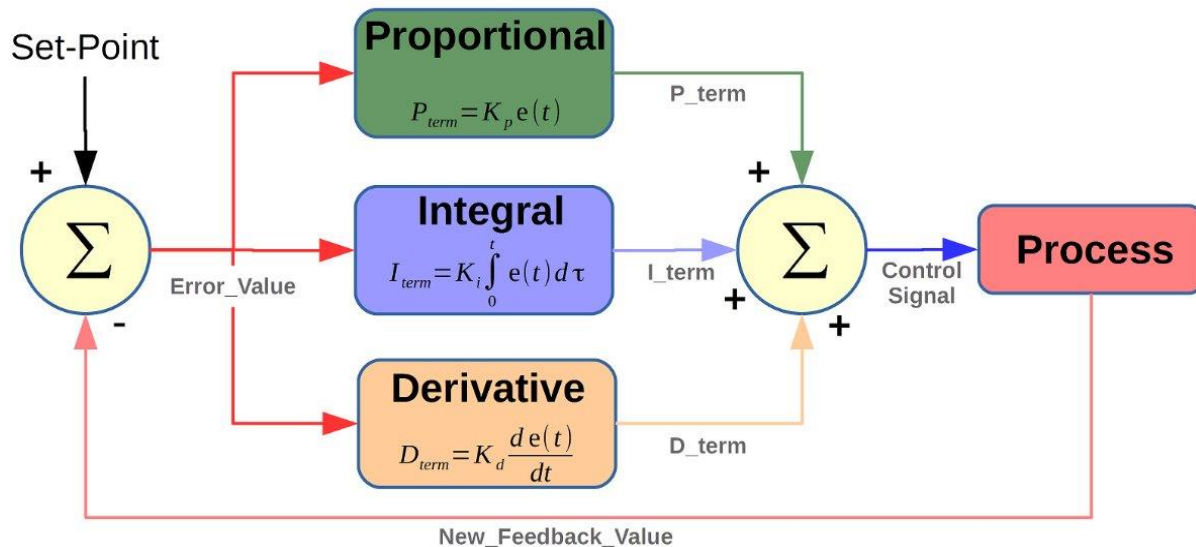
Description: The code reads the temperature from the ADT75 that is located on the board. The control loop has a setpoint temperature that is ideal for the board. Keeping the fan on top of the temp sensor we can see how the speed of the motor adjusts to cool the system and prevents overheating. When the sensor reaches the preferred temperature, the fan will keep running at a lower speed.

PID controllers represent a sophisticated feedback mechanism vital for controlling dynamic systems. At their core, they operate using three basic terms: **Proportional (P)**, **Integral (I)**, and **Derivative (D)**. Each term uniquely modulates the output signal based on the difference between the desired setpoint and the actual measured value, commonly known as the error.

Proportional Gain (Kp): This parameter determines the proportion of the error signal contributing to the controller output. A higher **Kp** value results in a stronger response to the current error. Too high a **Kp** can lead to oscillations or instability, while too low a value can result in a sluggish response.

Integral Gain (Ki): The integral term considers the accumulation of past errors and amplifies them over time. It helps eliminate steady-state error by continuously adjusting the control signal. A higher **Ki** value helps reduce steady-state error but can lead to overshoot or instability if set too high.

Derivative Gain (Kd): The derivative term predicts the future behavior of the error based on its current rate of change. It helps dampen oscillations by counteracting rapid changes in the error signal. Increasing **Kd** enhances damping and reduces overshoot, but too high a value can lead to instability or sensitivity to noise.



```
# Get the PID-adjusted PWM duty cycle based on the temperature
pwm_output = pid_control(current_temperature)
```

Steps

- Use the connector labeled as "2" and insert it into the board.
- Run the **pid_control.py** file and see how the PWM signal and speed adjust based on temperature.
- Make the **"pwm_output"** variable an input from the user and see how the duty cycle affects the speed of the fan.
- Try inputting duty cycles of your choice (e.g. 20, 50, 100)

```
# Python program showing
# a use of input()

val = input("Enter your value: ")
print(val)
```