Coursework 2 - Annunciator Panel

You are required to implement a simulation of an annunciator panel which is reporting, in real time, data being read from a Data AQuisition (DAQ) system, to help opperate a plant. The DAQ Python Class (included in the Notebook) will report very noisy data from a level sensor. You must use the Blinkstic to display the following information:

LED		Light
3	Low level warning, v < -2.56 volts	RED
4	Level in opperating range	Green
5	Plant Running	Blue
6	High level warning, v > 4.27 volts	RED

Optionally you may used LEDs 7, 0, 1 and 2 to indicate the fluid level

LED		Light
7	80% full	Amber
0	60% full	Amber
1	40% full	Amber
2	20% full	Amber

You must initialise the DAQ as follows

```
from scada import DAQ
mydaq = DAQ()
mydaq.connect('coursework')
mydaq.trigger()
```

and use the function

```
time_of_reading, reading = mydaq.next_reading()
```

to obtain data from the DAQ (which runs at 2 Hz).

Your program must run for 60 seconds, processing 120 readings. At the end of the run a graph showing the readings obtained (with the high and low levels marked) must be plotted. It is **critical** that the warning lights do not flicker (i.e. come on for one reading, go off for the next, and come on again). Your Python code must use an appropriate strategy to prevent this. Please Note:

- **Anyone** using `mydaq.voltage()` in their Python programme for the coursework will *fail* automatically.
- the reading returned by `mydaq.next_reading()` is an integer value between 0 and 1024.
- the `coursework` sensor is *personalised* using the matriculation number recorded on *your* blinkstick. If this does not match *your* matriculation number we will assume you are cheating.
- You can communicate with the blinkstick using the `mydaq.bstick` object in the normal way.
- you may find the `constant`, `ramp up` and `ramp down` sensors useful to debug and test your code.

You must save a pdf of this workbook and upload it to Turn-it-In, by the deadline. It must be uploaded together with a narrated video showing your simulation running. *Extensions are not permitted unless you have an approved learning adjustment*.

- 24 marks are avaiable for the design,
- 32 marks for the python programming,
- 24 marks for code documentation, and
- 20 marks for the video (including the commentary).

Please remember that programmes which exhibit dangerous behaviour (e.g. high and low levels at the same time) or where the warning lights flicker will be an automatic fail.

Part 1 - Design (24 marks)

I designed this program to simulate the alarm panel of a factory by acquiring real-time data from the level sensors through a data acquisition system (DAQ) and using the LEDs on the blinkstick to display the operational status of the factory and alarm messages. Firstly, initialising the Data Acquisition Device (DAQ) and the blinkstick ensures that the sensor data can be acquired and the corresponding LEDs can be controlled. Then, the upper and lower voltage limits are defined to determine if an alarm is triggered when below the lower limit or above the upper limit.

In the main loop, the program will continuously read the voltage data from the DAQ device, convert the read value (from 0 to 1024) to the actual voltage (from -5V to 5V), and judge the current operation status by the voltage value. Depending on the status, the corresponding LEDs are illuminated: red for low and high potential alarms, green for regular operation, and blue for plant in operation. In addition, the current level height is indicated by a level indicator. As the level rises, the indicator lights up in sequence from 20% to 80%, which represents the relative height of the current level. The red light will be illuminated when the voltage value is detected to be above the set upper limit or below the lower limit. At the same time, the green light will go out, which means that the unit is in a dangerous condition. When the voltage drops or rises to within safe limits, the green light will be restored.

To prevent the LED from flickering frequently, the status of the LED is updated by comparing the status of the previous and this time and then updating the status of the LED when the status changes. This ensures that the LED will not blink. I also added a delay to each loop in the program so that the LEDs can remain stable for a while in each state.

Finally, the program plots the read voltage data on a graph and shows the voltage trend over time. I also plotted threshold lines for high and low voltages in the graph to see if the voltages were within safe limits.

In [ ]:
```
Initialize DAQ and CONNECT to 'coursework'
    Initialize blinkstick

    Define thresholds (LOW = -2.56, HIGH = 4.27) and LED positions and colors
    SET previous state to NONE
    REPEAT 120 times:
        READ time and voltage from DAQ
        DETERMINE current state based on thresholds

        IF state changed THEN
            UPDATE LEDs for current state
            UPDATE fluid level LEDs based on voltage

        SET previous state to current state

PLOT voltage and time readings with thresholds
```

Part 2 - Python code (66 marks)

Write your code in the next cell. Please remember (for a B grade) we are looking for code which is an elegant model solution for the problem, including the use of LEDs 7, 0, 1 and 2 to indicate the fluid level. It should demonstrate the use of the elements of Python needed to implement the anunciator panel as described above. Code should be clearly written, easy to read, make consistent use of spacing, be fully self-documenting and follow the Python naming conventions for constatnts, variables, functions, classes etc. Grades A3 to A1 require you to demonstrate learning that goes beyond what we have taught you and uses more advanced technqiues. If you choose to use such techniques, remember that code still needs to work propperly and be elegant. Overly complicated solutions could lose you marks!

32 marks for programming, 24 marks for code documentation

In [4]:
```
import time
import matplotlib.pyplot as plt
from scada import DAQ
from blinkstick import blinkstick

# Initialising the DAQ (Data Acquisition Device)
mydaq = DAQ()
mydaq.connect('coursework')  # Connecting to data source
mydaq.trigger()

# Initialising Blinkstick for controlling LEDs
bstick = mydaq.bstick

# Define upper and lower voltage limits
LV_T = -2.56  # Low Voltage Alarm Threshold. Triggers a low voltage alarm if the voltage falls below this value.
HV_T = 4.27  # High Voltage Alarm Threshold. Triggers a high voltage alarm if the voltage is higher than this value

# Define the number and colour of the LEDs
LV_L = 3  # Red light for low Voltage alarm
NO_L = 4  # Green light for normal operation status
PR_L = 5  # Blue light, indicating that the plant is running
HV_L = 6  # Red light for high Voltage alarm
F_L = [2, 1, 0, 7]  # Fluid Level LED No: 2 -- 20%, 1 -- 40%, 0 -- 60%, 7 -- 80%

# Defining LED colours (Use RGB values)
LED_COLORS = {
    'red': [128, 0, 0],  # red for alarm indicators
    'green': [0, 128, 0],  # Green for normal operation indicator
    'blue': [0, 0, 128],  # Blue colour for plant operating status indicator
    'amber': [128, 95, 0],  # amber, used for fluid level
    'off': [0, 0, 0]  # turn off the light
}

# Set the blue light (plant running) to always be on
bstick.set_color(index=PR_L, red=LED_COLORS['blue'][0], green=LED_COLORS['blue'][1], blue=LED_COLORS['blue'][2])

# Define a list for storing read times and read data
readings = []  # Store voltage readout for subsequent graphing of voltage changes
times = []  # Stores the time of each read for subsequent drawing of the timeline

# Read and process DAQ data 120 times
previous_state = None  # Stores the last state to avoid unnecessary flickering
for _ in range(120):
    # Reading data from DAQ
    time_of_reading, reading = mydaq.next_reading()  # time_of_reading is the time of reading, reading is an integer between 0 and 1024
    voltage = (reading / 1024.0) * 10 - 5  # Converts readings to voltage (range: -5V to 5V)
    readings.append(voltage)  # Recording voltage values in 'reading' list
    times.append(time_of_reading)  # Record reading time in 'time' list

    # Determine current voltage status
    if voltage < LV_T:
        current_state = 'low'  # Voltage below low threshold, state 'low'
    elif voltage > HV_T:
        current_state = 'high'  # Voltage above high threshold, state ‘high’
    else:
        current_state = 'normal'  # Voltage is within normal range, status is ‘normal’.

    # Update the status LED only when the status changes to avoid LED flickering
    if current_state != previous_state:
        # Turn off all status LEDs
        for led in range(8):
            if led != PR_L:
                bstick.set_color(index=led, red=LED_COLORS['off'][0], green=LED_COLORS['off'][1], blue=LED_COLORS['off'][2])
```

```
        # Set the corresponding status LED according to the current state
        if current_state == 'low':
            # Setting the low voltage alarm LED to red
            bstick.set_color(index=LV_L, red=LED_COLORS['red'][0], green=LED_COLORS['red'][1], blue=LED_COLORS['red'][2])
        elif current_state == 'high':
            # Set the high potential alarm LED to red
            bstick.set_color(index=HV_L, red=LED_COLORS['red'][0], green=LED_COLORS['red'][1], blue=LED_COLORS['red'][2])
        elif current_state == 'normal':
            # Setting the green light for normal operation status and the blue light for factory operation
            bstick.set_color(index=NO_L, red=LED_COLORS['green'][0], green=LED_COLORS['green'][1], blue=LED_COLORS['green'][2])

    # Setting the Level LED, judging from 20% to 80% in sequence
    for i in range(len(F_L)):
        if voltage >= (i + 1) * 2 - 5:  # Determine if the voltage reaches the corresponding level thresholds
            bstick.set_color(index=F_L[i], red=LED_COLORS['amber'][0], green=LED_COLORS['amber'][1], blue=LED_COLORS['amber'][2])
            # Set the corresponding level LED to amber
        else:
            bstick.set_color(index=F_L[i], red=LED_COLORS['off'][0], green=LED_COLORS['off'][1], blue=LED_COLORS['off'][2])
            # If the threshold is not reached, the corresponding LED is switched off

    # Update last status
    previous_state = current_state

    # Maintains stability and prevents frequent flashing of LEDs
    time.sleep(0.5)

# Charting the read data
plt.figure(figsize=(10, 6))
plt.plot(times, readings, label='Reading')  # Plotting voltage over time
plt.axhline(y=LV_T, color='r', linestyle='--', label='Low Level Threshold')  # Plotting the low voltage threshold line
plt.axhline(y=HV_T, color='r', linestyle='--', label='High Level Threshold')  # Plotting the High Voltage Threshold Lines
plt.xlabel('Time (s)')  # Setting the x-axis labels
plt.ylabel('Voltage (V)')  # Setting the y-axis labels
plt.title('DAQ Readings Over Time')  # Setting the Chart Title
plt.legend()  # Show legend
plt.grid(True)  # Show gridlines
plt.show()  # Show charts
```

```
DAQ s2490971 Initialised 2024-11-26 10:21:48.275274 Q=0.009765625
licensed to ZhongKai Zhang.
coursework connected.
triggered at 2024-11-26 10:21:48.277376
```