Report on UVLO Comparator Simulation and Detection Using Digital Scope and Power Supply

Overview

This Python code simulates and detects the Under-Voltage Lockout (UVLO) comparator trigger signal using a digital oscilloscope (channel 4) and a power supply (channel 1). The power supply channel is configured to act as a digital comparator signal generator for simulation purposes. The code integrates hardware control via instrument drivers and implements a trimming routine to optimize the UVLO threshold voltage.

Hardware and Instrument Setup

The hardware layer is abstracted in the `HWL` class, which initializes three instruments:

- **A34461**: A digital multimeter (DMM) for voltage measurements.
- **N67xx**: A programmable power supply used here to generate digital signals and force voltages.
- **DPO4054**: A digital oscilloscope used to detect comparator trigger signals on channel 4.

```
from Instruments.a34461 import A34461
from Instruments.dpo4054 import DPO4054
from Instruments.n67xx import N67xx
from time import sleep
import math

class HWL:
    def __init__(self):
        self.meter = A34461('USB0::0x2A8D::0x1401::MY57200246::INSTR')
        self.ps = N67xx('USB0::0x0957::0x0F07::MY50002157::INSTR')
        self.scope = DPO4054('USB0::0x0699::0x0401::C020132::INSTR')

        # Setup digital scope channel 4 for comparator detection
        self.scope.setup_channel(channel=4, label='uvlo', display=True, scale=1,
                                 position=0.0, offset=0.0, coupling='DC',
                                 bandwidth='FULL', invert=False)
        self.scope.set_channel_display(channel=1, display=False)
        # Set timebase to half the input signal frequency (1 MHz)
        self.scope.set_timebase_scale(1 / (2 * 1E6))
```

Explanation:

- **Channel 4** of the scope is configured to monitor the UVLO comparator output signal.
- The timebase is set to capture signals at 1 MHz input frequency.
- Channel 1 display is disabled as it is used for power supply digital signal generation.

Comparator Trigger Simulation and Detection

The core logic to simulate and detect the comparator trigger is implemented in the `voltage_trigger_LH_callback` method:

```
def voltage_trigger_LH_callback(self, signal, reference, threshold,
measure_value):
    # Configure scope trigger on channel 4 for rising edge at 90% of 1.8V
    self.scope.trigger_setup(channel=4, level=1.8 * 0.9, slope='RISE',
mode='EDGE')
    self.scope.set_acquire_sequence()  # Prepare scope to acquire data

    # Setup power supply channel 1 as digital signal generator for simulation
    self.ps.configure_voltage_source(channel=1)
    self.ps.output_state(state=True, chan=1)

    # Generate digital trigger signal if measured value is close to threshold
(within 8%)
    if math.isclose(threshold, measure_value, rel_tol=0.08):
        self.ps.set_voltage(channel=1, voltage=1.8)  # Simulate comparator output
high
    else:
        self.ps.set_voltage(channel=1, voltage=0)    # Comparator output low

    # Check if comparator triggered on scope
    if self.scope.get_operation_status():
        self.ps.set_voltage(channel=1, voltage=0)
        self.ps.output_state(state=False, chan=1)
        return True, True  # Trigger detected
    else:
        self.ps.set_voltage(channel=1, voltage=0)
        self.ps.output_state(state=False, chan=1)
        return True, False  # No trigger detected
```

Explanation:

- The oscilloscope is set to trigger on a rising edge at 1.62 V (90% of 1.8 V).
- Power supply channel 1 simulates the comparator output by outputting 1.8 V if the measured voltage is close to the threshold.
- The function returns a tuple indicating if the operation was successful and if the trigger was detected.

Voltage Forcing and Measurement

The `voltage_source` method forces a voltage on power supply channel 1 and measures it:

```
def voltage_source(self, signal, reference, value):
    self.ps.set_voltage(channel=1, voltage=value)
    self.ps.output_state(state=True, chan=1)
    return False, float(self.ps.measure_voltage_dc(chan=1))
```

- This method is used to apply a specific voltage to the device under test (DUT) and read back the actual voltage.

UVLO Threshold Trimming Routine

The main script performs a trimming loop to find the optimal trimming code that sets the UVLO threshold voltage closest to a target value (`LH_Th = 2.25 V`):

```
if __name__ == "__main__":
    hwl = HWL()
    g.hardware_callbacks = {
        'voltage_trigger_lh': hwl.voltage_trigger_LH_callback
    }

    LH_Th = 2.25  # Target threshold voltage
    error_spread = LH_Th * 0.1  # 10% error margin
    code_width = 5  # 5-bit trimming code
    min_error = float('inf')
    optimal_code = None
    optimal_measured_value = None
    force_voltage_low_limit = 2
    force_voltage_high_limit = 3

    for Code in range(2**code_width):
        force_voltage = force_voltage_low_limit
        trigger = False

        while True:
            pvdd_forced_voltage = VFORCE(signal='PVDD', reference='GND',
value=force_voltage, error_spread=error_spread)
            trigger = VTRIG_LH(signal='IODATA0', reference='GND', threshold=LH_Th,
expected_value=force_voltage)

            if trigger:
                break
            elif pvdd_forced_voltage >= force_voltage_high_limit:
                print(f'..... Voltage max limit {force_voltage_high_limit}V
crossed ........')
                break

            force_voltage += 0.01  # Increment voltage by 10mV
            sleep(0.001)  # 1 ms delay

        error = abs(pvdd_forced_voltage - LH_Th) / abs(LH_Th) * 100
        if error < min_error:
            min_error = error
            optimal_code = hex(Code)
            optimal_measured_value = pvdd_forced_voltage

    # Final pass/fail report
    if force_voltage_low_limit < optimal_measured_value <
force_voltage_high_limit:
```

```
            print(f'............ UVLO_Trim Test Passed ........')
            # TODO: Burn optimal trimming code to OTP via I2C
        else:
            print(f'............ UVLO_Trim Test Failed ........')

        print(f"Optimal Code: {optimal_code}")
        print(f"Optimal Measured Value: {optimal_measured_value:.4f} V (Target:
{LH_Th} V)")
        print(f"Minimum Error: {min_error:.6f} %")
```

Explanation:

- The trimming code iterates over all possible 5-bit codes (0 to 31).
- For each code, it forces a voltage starting from 2 V and increments by 10 mV until the comparator triggers or voltage exceeds 3 V.
- The difference between forced voltage and threshold is calculated as error.
- The code with the minimum error is selected as optimal.
- Final pass/fail is decided based on whether the optimal voltage lies within the allowed range.

Summary

Aspect	Description
Purpose	Simulate and detect UVLO comparator trigger using a digital oscilloscope and power supply.
Scope Channel 4	Configured to detect comparator output digital signal (UVLO trigger).
Power Supply Channel 1	Used as digital signal generator to simulate comparator output for testing.
Trigger Detection	Scope triggers on rising edge at 90% of 1.8 V digital signal.
Trimming Routine	Iterates through trimming codes to find optimal UVLO threshold voltage with minimal error.
Callbacks	Hardware callbacks attached for voltage forcing and trigger detection.

Conclusion

This code provides a comprehensive hardware-in-the-loop simulation and detection framework for UVLO comparator signals. By combining power supply control, oscilloscope triggering, and voltage measurement, it enables automated trimming and verification of UVLO thresholds in a chip. The modular design with callbacks and instrument abstraction allows easy extension for other tests or devices.