1. glob/__init__.py

```
class GlobalContext:
    """
    Global context managing hardware availability flags, callback functions,
    and output logs for trigger events.
    """
    def __init__(self):
        self.output = []  # Log of trigger events and measurements
        self.instructions = {}
        self.dut_description = None

        # Hardware availability flags
        self.voltage_force_hardware_available = False
        self.current_force_hardware_available = False

        # Callback functions for triggers and measurements
        self.hardware_callbacks = {
            'voltage_trigger_hl': None,
            'voltage_trigger_lh': None,
            'voltage_trigger_lg': None,
            'current_trigger_hl': None,
            'current_trigger_lh': None,
            'current_trigger_lg': None,
        }

    @property
    def callback_keys(self):
        return self.hardware_callbacks.keys()

# Instantiate global context
g = GlobalContext()
```

2. trig_callback_functions.py

```
import random

def vtrig_hl_callback(g, signal, reference, threshold):
    hardware_available = g.voltage_force_hardware_available
    measured_voltage = threshold - 0.1 + random.uniform(-0.2, 0.2) if
hardware_available else 0.0
    triggered = measured_voltage < threshold if hardware_available else False
    print(f"[vtrig_hl_callback] HW avail: {hardware_available}, Measured:
{measured_voltage:.3f}, Threshold: {threshold}, Triggered: {triggered}")
    return hardware_available, triggered
```

```
def vtrig_lh_callback(g, signal, reference, threshold):
    hardware_available = g.voltage_force_hardware_available
    measured_voltage = threshold + 0.1 + random.uniform(-0.2, 0.2) if
hardware_available else 0.0
    triggered = measured_voltage > threshold if hardware_available else False
    print(f"[vtrig_lh_callback] HW avail: {hardware_available}, Measured:
{measured_voltage:.3f}, Threshold: {threshold}, Triggered: {triggered}")
    return hardware_available, triggered

def vtrig_lg_callback(g, signal, reference, _):
    hardware_available = g.voltage_force_hardware_available
    measured_voltage = random.uniform(0, 0.1) if hardware_available else 0.0
    triggered = measured_voltage < 0.05 if hardware_available else False
    print(f"[vtrig_lg_callback] HW avail: {hardware_available}, Measured:
{measured_voltage:.3f}, Triggered: {triggered}")
    return hardware_available, triggered

def atrig_hl_callback(g, signal, reference, threshold):
    hardware_available = g.current_force_hardware_available
    measured_current = threshold - 0.001 + random.uniform(-0.002, 0.002) if
hardware_available else 0.0
    triggered = measured_current < threshold if hardware_available else False
    print(f"[atrig_hl_callback] HW avail: {hardware_available}, Measured:
{measured_current:.6f}, Threshold: {threshold}, Triggered: {triggered}")
    return hardware_available, triggered

def atrig_lh_callback(g, signal, reference, threshold):
    hardware_available = g.current_force_hardware_available
    measured_current = threshold + 0.001 + random.uniform(-0.002, 0.002) if
hardware_available else 0.0
    triggered = measured_current > threshold if hardware_available else False
    print(f"[atrig_lh_callback] HW avail: {hardware_available}, Measured:
{measured_current:.6f}, Threshold: {threshold}, Triggered: {triggered}")
    return hardware_available, triggered

def atrig_lg_callback(g, signal, reference, _):
    hardware_available = g.current_force_hardware_available
    measured_current = random.uniform(0, 0.001) if hardware_available else 0.0
    triggered = measured_current < 0.001 if hardware_available else False
    print(f"[atrig_lg_callback] HW avail: {hardware_available}, Measured:
{measured_current:.6f}, Triggered: {triggered}")
    return hardware_available, triggered
```

3. hadware/trig.py

```
def apply_trig_and_measure(g, signal, reference, threshold, force_type,
simulated_measured_value=None):
    """
```

```
    Apply trigger check and measure using hardware callback if available.
    If hardware unavailable, use simulated_measured_value to compare with
threshold:
      - HL triggers: True if measured_value < threshold
      - LH triggers: True if measured_value > threshold
      - LG triggers: True if measured_value < fixed small threshold

    Returns:
        (hardware_available: bool, triggered: bool)
    """
    callback = g.hardware_callbacks.get(force_type)
    if callback:
        hardware_available, triggered = callback(g, signal, reference, threshold)
        if hardware_available:
            return hardware_available, triggered

    if simulated_measured_value is None:
        raise ValueError("Simulated measured value must be provided if hardware is
unavailable.")

    if force_type.endswith('_hl'):
        triggered = simulated_measured_value < threshold
    elif force_type.endswith('_lh'):
        triggered = simulated_measured_value > threshold
    elif force_type.endswith('_lg'):
        if 'voltage' in force_type:
            triggered = simulated_measured_value < 0.05
        elif 'current' in force_type:
            triggered = simulated_measured_value < 0.001
        else:
            triggered = False
    else:
        triggered = False

    print(f"[apply_trig_and_measure] HW avail: False, Simulated:
{simulated_measured_value}, Threshold: {threshold}, Triggered: {triggered}")
    return False, triggered
```

4. instructions/trigger.py

```
from apply_trig_and_measure import apply_trig_and_measure
from global_context import g

def VTRIG_HL(signal: str, threshold: float, reference: str = 'GND',
expected_value: float = 2.7):
    hw_avail, triggered = apply_trig_and_measure(
        g, signal, reference, threshold, 'voltage_trigger_hl', expected_value
    )
    g.output.append({'type': 'TRIGGER', 'trigger': 'VTRIG_HL', 'signal': signal,
```

```
'triggered': triggered})
    return triggered

def VTRIG_LH(signal: str, threshold: float, reference: str = 'GND',
expected_value: float = 3.3):
    hw_avail, triggered = apply_trig_and_measure(
        g, signal, reference, threshold, 'voltage_trigger_lh', expected_value
    )
    g.output.append({'type': 'TRIGGER', 'trigger': 'VTRIG_LH', 'signal': signal,
'triggered': triggered})
    return triggered

def VTRIG_LG(signal: str, reference: str = 'GND', expected_value: float = 0.02):
    hw_avail, triggered = apply_trig_and_measure(
        g, signal, reference, None, 'voltage_trigger_lg', expected_value
    )
    g.output.append({'type': 'TRIGGER', 'trigger': 'VTRIG_LG', 'signal': signal,
'triggered': triggered})
    return triggered

def ATRIG_HL(signal: str, threshold: float, reference: str = 'GND',
expected_value: float = 0.005):
    hw_avail, triggered = apply_trig_and_measure(
        g, signal, reference, threshold, 'current_trigger_hl', expected_value
    )
    g.output.append({'type': 'TRIGGER', 'trigger': 'ATRIG_HL', 'signal': signal,
'triggered': triggered})
    return triggered

def ATRIG_LH(signal: str, threshold: float, reference: str = 'GND',
expected_value: float = 0.15):
    hw_avail, triggered = apply_trig_and_measure(
        g, signal, reference, threshold, 'current_trigger_lh', expected_value
    )
    g.output.append({'type': 'TRIGGER', 'trigger': 'ATRIG_LH', 'signal': signal,
'triggered': triggered})
    return triggered

def ATRIG_LG(signal: str, reference: str = 'GND', expected_value: float = 0.0005):
    hw_avail, triggered = apply_trig_and_measure(
        g, signal, reference, None, 'current_trigger_lg', expected_value
    )
    g.output.append({'type': 'TRIGGER', 'trigger': 'ATRIG_LG', 'signal': signal,
'triggered': triggered})
    return triggered
```

5. test_triggers.py

```
import random
from global_context import g
from callback_functions import *
from trigger_functions import *

def assign_callbacks(assign: bool):
    if assign:
        g.hardware_callbacks.update({
            'voltage_trigger_hl': vtrig_hl_callback,
            'voltage_trigger_lh': vtrig_lh_callback,
            'voltage_trigger_lg': vtrig_lg_callback,
            'current_trigger_hl': atrig_hl_callback,
            'current_trigger_lh': atrig_lh_callback,
            'current_trigger_lg': atrig_lg_callback,
        })
    else:
        for key in g.hardware_callbacks.keys():
            g.hardware_callbacks[key] = None

def test_scenario(hw_voltage: bool, hw_current: bool, callbacks_assigned: bool):
    print("\n--- Test Scenario ---")
    print(f"Voltage Hardware Available: {hw_voltage}")
    print(f"Current Hardware Available: {hw_current}")
    print(f"Callbacks Assigned: {callbacks_assigned}")

    g.voltage_force_hardware_available = hw_voltage
    g.current_force_hardware_available = hw_current

    assign_callbacks(callbacks_assigned)

    triggers = [
        ('VTRIG_HL', VTRIG_HL, 3.0),
        ('VTRIG_LH', VTRIG_LH, 3.0),
        ('VTRIG_LG', VTRIG_LG, None),
        ('ATRIG_HL', ATRIG_HL, 0.01),
        ('ATRIG_LH', ATRIG_LH, 0.1),
        ('ATRIG_LG', ATRIG_LG, None),
    ]

    signals = ['VCC_CORE', 'PWR_RAIL']
    references = ['GND']

    for name, func, threshold in triggers:
        signal = random.choice(signals)
        reference = random.choice(references)

        if threshold is not None:
            noise = random.uniform(-0.2 * threshold, 0.2 * threshold)
            expected_value = threshold + noise
            triggered = func(signal, threshold, reference, expected_value)
        else:
            expected_value = random.uniform(0.0, 0.05) if 'VTRIG' in name else
random.uniform(0.0, 0.001)
```

```
            triggered = func(signal, reference, expected_value)

        print(f"\nTrigger: {name}, Signal: {signal}, Threshold: {threshold},
Expected Value: {expected_value:.4f}")
        print(f"Triggered: {triggered}")

if __name__ == "__main__":
    for hw_v in [True, False]:
        for hw_c in [True, False]:
            for cb_assigned in [True, False]:
                test_scenario(hw_v, hw_c, cb_assigned)
```

Detailed Report

Overview

This modular trigger system supports voltage and current trigger detection with fallback simulation when hardware is unavailable. It is designed for flexible integration with hardware or simulation environments.

Components

1. **GlobalContext**

- Centralizes hardware availability flags and callback registrations.
- Stores output logs for trigger events.
- Easily extendable to add more hardware types or callbacks.

2. **Callback Functions**

- Simulate or interface with hardware to measure signals.
- Return whether hardware is available and if the trigger condition is met.
- Include noise simulation for realistic testing.
- Separate callbacks for each trigger type:
  - Voltage: HL, LH, LG
  - Current: HL, LH, LG

3. **apply_trig_and_measure**

- Core logic that tries hardware callbacks first.
- If hardware unavailable, uses provided simulated value to evaluate trigger.
- Supports HL (High-to-Low), LH (Low-to-High), LG (Low-to-Ground) triggers.
- Raises error if no simulated value provided when hardware unavailable.

4. **Trigger Functions**

- User-facing functions that wrap `hadware/trig.py`.
- Accept parameters:

    - `signal`: signal line name
    - `threshold`: trigger threshold (not for LG triggers)
    - `reference`: reference signal, default `'GND'`
    - `expected_value`: simulated measurement if hardware unavailable
- Log trigger results to global output.

5. **Test Harness**

- Automates testing of all triggers under all combinations of:
    - Voltage hardware available/not
    - Current hardware available/not
    - Callbacks assigned/not
- Uses randomized noisy expected values.
- Prints detailed results for debugging and verification.

Keywords and Concepts

- **Trigger Types**: HL (High-to-Low), LH (Low-to-High), LG (Low-to-Ground)
- **Signal & Reference**: Names of signals and their reference points for measurement
- **Threshold**: Value to compare against measured or simulated signal
- **Expected Value**: Simulated measurement used when hardware is unavailable
- **Hardware Availability**: Flags indicating if physical measurement hardware is present
- **Callbacks**: Functions that perform actual or simulated measurements

---

Applications

- Automated hardware testing and validation
- Simulation environments without hardware present
- Flexible trigger detection for power rails, signals, and currents
- Integration with production test frameworks or lab measurement setups

---

Customization

- Add new trigger types by extending `GlobalContext.hardware_callbacks` and implementing callbacks and trigger functions.
- Replace simulated noise models with real hardware measurement APIs.
- Extend `apply_trig_and_measure` for advanced trigger logic like hysteresis or timing.
- Customize output logging for detailed traceability or reporting.
- Adapt test harness for unit testing or continuous integration.

Summary

This modular design cleanly separates concerns:

- **Global state and configuration** (`GlobalContext`)
- **Hardware interaction and simulation** (`callback_functions.py`)

- **Core trigger logic** (`trig.py`)
- **User API for triggers** (`trigger.py`)
- **Comprehensive testing** (`test_triggers.py`)

It supports robust, flexible, and maintainable trigger detection suitable for a wide range of hardware and simulation scenarios.