dfttools Force Sweep Functionality – Code Analysis & Customization Guide

Overview

The `dfttools` library supports **force sweeps** for voltage, current, resistance, and frequency signals. These functions perform the iterative application of a range of values, measure the hardware response at each step, and log the results. All force data or configuration like `from dfttools import*`.

All sweep functions rely on:

- `apply_force_and_measure_sweep()`: a core function that manages the step-by-step application and measurement.
- **Callback functions** registered in `g.hardware_callbacks` for each force type, which must be customized to interface with your hardware.

Core Components

1. Sweep Functions

The core part of defining how the sweeps instructions, and the initial state includes importing instructions from different hardware component,

```
from dfttools.hardware.force_sweep import apply_force_and_measure_sweep

def VFORCESWEEP(signal: str = 'VCC', reference: str = 'GND', initial_value: float
= 0.0, end_value: float = 5.0, step: float = 0.1, step_time: float = 0.01):
    """
    Perform a voltage sweep from initial_value to end_value with specified step
and step_time.
    """
    sweep_results = apply_force_and_measure_sweep(g, signal, reference,
initial_value, end_value, step, step_time, 'voltage_force_sweep')
    if not sweep_results:
        return {'signal': signal, 'reference': reference, 'sweep_results': []}

    g.output.append({'type': 'SWEEP', 'signal': signal, 'reference': reference,
'sweep_results': sweep_results})
    return sweep_results

def AFORCESWEEP(signal: str = 'VCC', reference: str = 'GND', initial_value: float
= 0.0, end_value: float = 5.0, step: float = 0.1, step_time: float = 0.01):
    """
    Perform a current sweep from initial_value to end_value with specified step
and step_time.
    """
    sweep_results = apply_force_and_measure_sweep(g, signal, reference,
initial_value, end_value, step, step_time, 'current_force_sweep')
    if not sweep_results:
```

```
        return {'signal': signal, 'reference': reference, 'sweep_results': []}

    g.output.append({'type': 'SWEEP', 'signal': signal, 'reference': reference,
'sweep_results': sweep_results})
    return sweep_results

def RESFORCESWEEP(signal: str = 'R1', reference: str = 'GND', initial_value: float
= 0.0, end_value: float = 1000.0, step: float = 10.0, step_time: float = 0.01):
    """
    Perform a resistance sweep from initial_value to end_value with specified step
and step_time.
    """
    sweep_results = apply_force_and_measure_sweep(g, signal, reference,
initial_value, end_value, step, step_time, 'resistance_force_sweep')
    if not sweep_results:
        return {'signal': signal, 'reference': reference, 'sweep_results': []}

    g.output.append({'type': 'SWEEP', 'signal': signal, 'reference': reference,
'sweep_results': sweep_results})
    return sweep_results

def FREQFORCESWEEP(signal: str = 'CLK', reference: str = 'GND', initial_value:
float = 0.0, end_value: float = 100.0, step: float = 1.0, step_time: float =
0.01):
    """
    Perform a frequency sweep from initial_value to end_value with specified step
and step_time.
    """
    sweep_results = apply_force_and_measure_sweep(g, signal, reference,
initial_value, end_value, step, step_time, 'frequency_force_sweep')
    if not sweep_results:
        return {'signal': signal, 'reference': reference, 'sweep_results': []}

    g.output.append({'type': 'SWEEP', 'signal': signal, 'reference': reference,
'sweep_results': sweep_results})
    return sweep_results
```

Explanation: The sweep function `VFORCESWEEP, AFORCESWEEP, RESFORCESWEEP, FREQFORCESWEEP`, takes multiple parameter configurations like signal, reference, value, step and step_time. with the values, signal and reference, the core instruction calls a callback with the arguments. the result is returned to the caller. if the call back is not exist then returns the signal parameters.

2. Sweep Implementation (`apply_force_and_measure_sweep()`)

```
from dfttools.hardware.force import apply_force_and_measure
import time

def apply_force_and_measure_sweep(g, signal, reference, initial_value, end_value,
step, step_time, force_type):
    sweep_results = []
```

```
    # Check hardware availability using the callback if defined
    if g.hardware_callbacks[force_type]:
        current_value = initial_value
        while current_value &lt;= end_value:
            hardware_available, measured_value = apply_force_and_measure(g,
signal, reference, current_value, force_type)
            if hardware_available:
                sweep_results.append(measured_value)
                # Simulate waiting for step_time
                time.sleep(step_time)
            current_value += step

    return sweep_results
```

Explanation: The core sweep is implemented in `apply_force_and_measure_sweep` this implements the whole sweep instruction. a loop runs from `initial_value` till the `end_value`.

Hardware Integration

```
from dfttools.hardware.force import apply_force_and_measure
```

This import statement indicates the core utility responsible for applying and measuring the force at each step of the sweep is leveraged.

3. Callback Functions for Sweeps

```
def voltage_force_sweep_callback(g, signal, reference, value):
    force_hardware_available = True  # Set hardware availability dynamically
    measured_value = value  # Example dynamic measurement

    return force_hardware_available, measured_value

def current_force_sweep_callback(g, signal, reference, value):
    force_hardware_available = True  # Set hardware availability dynamically
    measured_value = value  # Example dynamic measurement
    return force_hardware_available, measured_value

def resistance_force_sweep_callback(g, signal, reference, value):
    force_hardware_available = True  # Set hardware availability dynamically
    measured_value = value  # Example dynamic measurement
    return force_hardware_available, measured_value

def frequency_force_sweep_callback(g, signal, reference, value):
    force_hardware_available = True  # Set hardware availability dynamically
    measured_value = value  # Example dynamic measurement
    return force_hardware_available, measured_value
```

Explanation: The callback functions must be implemented to intract with hadware,

- The `force_hardware_available` should be dynamically be set to simulate/real time hardware presence status.

4. Hardware Callbacks Registration.

```
g.hardware_callbacks = {
    'voltage_force_sweep': voltage_force_sweep_callback,
    'current_force_sweep': current_force_sweep_callback,
    'resistance_force_sweep': resistance_force_sweep_callback,
    'frequency_force_sweep': frequency_force_sweep_callback,
}
```

Explanation: The `g.hardware_callbacks` in g, the global context, stores the name for each callbacks used in the code.

5. Hardware Callbacks Application.

```
# Set hardware availability dynamically
measured_value = value  # Example dynamic measurement

return force_hardware_available, measured_value
```

Explanation: The `value` parameter will contain the value that is currently been swept, after the hardware, the measured value must be captured and returned, along with the hardware availability status.

6. Putting All Things Together.

```
from dfttools import *
# Callbacks

print("Voltage Sweep Results:", VFORCESWEEP(signal='VCC', reference='GND',
initial_value=0.0, end_value=5.0, step=1, step_time=0.01))
print("Current Sweep Results:", AFORCESWEEP(signal='VCC', reference='GND',
initial_value=0.0, end_value=5.0, step=1, step_time=0.01))
print("Resistance Sweep Results:", RESFORCESWEEP(signal='R1', reference='GND',
initial_value=0.0, end_value=1000.0, step=100.0, step_time=0.01))
print("Frequency Sweep Results:", FREQFORCESWEEP(signal='CLK', reference='GND',
initial_value=0.0, end_value=100.0, step=100.0, step_time=0.01))
```

Explanation:

- The example code runs all the different hardware sweep function which are voltage, resistance, frequency and current.

dfttools Force Sweep Customization

```
1.  Callback functions should be developed considering all the hardware
specifications
2.  step_time should be properly configured to enable correct sweep operation.
```

Disclaimer: *The hardware force instructions are currently simple, but in the future, a feedback loop should implemented in hardware side to re capture any potential failures.*

7. Test Report.

```
Voltage Sweep Results: [0.0, 1, 2, 3, 4, 5.0]
Current Sweep Results: [0.0, 1, 2, 3, 4, 5.0]
Resistance Sweep Results: [0.0, 100.0, 200.0, 300.0, 400.0, 500.0, 600.0, 700.0,
800.0, 900.0, 1000.0]
Frequency Sweep Results: [0.0, 100.0]
```

Explanation:

- This is the summary for the hardware function code results, which displays the values, after running the test.

---

customize the call back functions with powersupply force sweep and read the values back and return.

Certainly! Below is a detailed example showing how to customize **callback functions for power supply force sweeps** in the `dfttools` framework. The callbacks will:

- Apply each sweep step value (voltage or current) to the power supply.
- Read back the actual value from the hardware.
- Return the hardware availability and the measured value.

Custom Power Supply Force Sweep Callbacks with Readback

1. Simulated Hardware Interface Functions

Replace these with your actual hardware API or communication code.

```
import time

def set_power_supply_voltage(channel, voltage):
    print(f"[Hardware] Setting voltage {voltage:.3f} V on channel {channel}")
    time.sleep(0.05)  # Simulate hardware delay
    return True  # Return True if successful

def read_power_supply_voltage(channel):
    # Simulate reading voltage back from hardware
    measured_voltage = 3.3  # Replace with actual read command
    print(f"[Hardware] Reading voltage {measured_voltage:.3f} V from channel
{channel}")
    return measured_voltage

def set_power_supply_current(channel, current):
    print(f"[Hardware] Setting current {current:.3f} A on channel {channel}")
    time.sleep(0.05)
    return True

def read_power_supply_current(channel):
    measured_current = 1.2  # Replace with actual read command
    print(f"[Hardware] Reading current {measured_current:.3f} A from channel
{channel}")
    return measured_current
```

---

2. Customized Sweep Callback Functions

```
def voltage_force_sweep_callback(g, signal, reference, value):
    """
    Callback for voltage force sweep step.
    Sets voltage and reads back actual voltage.
    """
    hardware_available = set_power_supply_voltage(signal, value)
    if not hardware_available:
        return False, 0.0
    measured_value = read_power_supply_voltage(signal)
    return True, measured_value

def current_force_sweep_callback(g, signal, reference, value):
    """
    Callback for current force sweep step.
    Sets current and reads back actual current.
    """
    hardware_available = set_power_supply_current(signal, value)
    if not hardware_available:
        return False, 0.0
    measured_value = read_power_supply_current(signal)
    return True, measured_value
```

3. Register Callbacks in Global Context

```
from dfttools.glob import g

g.hardware_callbacks['voltage_force_sweep'] = voltage_force_sweep_callback
g.hardware_callbacks['current_force_sweep'] = current_force_sweep_callback
```

4. Example Usage of Sweep Functions

```
from dfttools import *

print("Starting Voltage Sweep:")
voltage_sweep_results = VFORCESWEEP(signal='CH1', reference='GND',
initial_value=0.0, end_value=5.0, step=1.0, step_time=0.1)
print("Voltage Sweep Results:", voltage_sweep_results)

print("\nStarting Current Sweep:")
current_sweep_results = AFORCESWEEP(signal='CH1', reference='GND',
initial_value=0.0, end_value=2.0, step=0.5, step_time=0.1)
print("Current Sweep Results:", current_sweep_results)
```

5. Output Example (Simulated)

```
Starting Voltage Sweep:
[Hardware] Setting voltage 0.000 V on channel CH1
[Hardware] Reading voltage 3.300 V from channel CH1
[Hardware] Setting voltage 1.000 V on channel CH1
[Hardware] Reading voltage 3.300 V from channel CH1
[Hardware] Setting voltage 2.000 V on channel CH1
[Hardware] Reading voltage 3.300 V from channel CH1
[Hardware] Setting voltage 3.000 V on channel CH1
[Hardware] Reading voltage 3.300 V from channel CH1
[Hardware] Setting voltage 4.000 V on channel CH1
[Hardware] Reading voltage 3.300 V from channel CH1
[Hardware] Setting voltage 5.000 V on channel CH1
[Hardware] Reading voltage 3.300 V from channel CH1
Voltage Sweep Results: [3.3, 3.3, 3.3, 3.3, 3.3, 3.3]

Starting Current Sweep:
[Hardware] Setting current 0.000 A on channel CH1
[Hardware] Reading current 1.200 A from channel CH1
[Hardware] Setting current 0.500 A on channel CH1
[Hardware] Reading current 1.200 A from channel CH1
[Hardware] Setting current 1.000 A on channel CH1
```

```
[Hardware] Reading current 1.200 A from channel CH1
[Hardware] Setting current 1.500 A on channel CH1
[Hardware] Reading current 1.200 A from channel CH1
[Hardware] Setting current 2.000 A on channel CH1
[Hardware] Reading current 1.200 A from channel CH1
Current Sweep Results: [1.2, 1.2, 1.2, 1.2, 1.2]
```

---

6. Customization Tips

- Replace the simulated `set_power_supply_*` and `read_power_supply_*` functions with your actual hardware API calls.
- Use your device's SDK, SCPI commands, or communication protocol (e.g., VISA, pyserial).
- Ensure proper error handling and return `(False, 0)` if hardware is not reachable or command fails.
- Adjust `step_time` to allow hardware to stabilize after each step.
- Log or store additional metadata as needed in `g.output`.

Summary

- **Define your hardware-specific callbacks** for each sweep step.
- **Register callbacks** in `g.hardware_callbacks`.
- **Use sweep functions** to perform automated parameter sweeps with measurement.
- **Customize** the callbacks to interface with your hardware APIs (serial, VISA, SDK, etc.).