Proposal: Streamlined Measurement Type Registration System

I suggest creating a more automated approach to adding new Measurement types using a registry pattern. Here's the proposed solution:

1. Create a measurement registration system using decorators and automatic discovery:

```python
# Create a new file: src/qscope/meas/registry.py

from typing import Type, Dict
from .measurement import Measurement

class MeasurementRegistry:
    _measurements: Dict[str, Type[Measurement]] = {}

    @classmethod
    def register(cls, name: str = None):
        def decorator(meas_class: Type[Measurement]):
            nonlocal name
            if name is None:
                name = meas_class.__name__
            cls._measurements[name] = meas_class
            return meas_class
        return decorator

    @classmethod
    def get_all_measurements(cls) -> Dict[str, Type[Measurement]]:
        return cls._measurements.copy()
```

2. Use the decorator to register measurements:

```python
# In measurement files:
from .registry import MeasurementRegistry

@MeasurementRegistry.register("ESR")
class ESR(Measurement):
    # Your measurement implementation
    pass
```

3. Create a base configuration class that measurements can extend:

```python
# In src/qscope/types.py
from dataclasses import dataclass
from typing import Optional

@dataclass
class BaseMeasurementConfig:
    sequence_module: Optional[str] = None  # Path to pulseblaster sequence if needed
    gui_parameters: dict = field(default_factory=dict)  # GUI-specific settings
```

4. Modify the GUI to automatically discover and add measurements:

```
# In qmeas_opts.py
from qscope.meas.registry import MeasurementRegistry

class QuantumMeasurementOpts(QGroupBox):
    def __init__(self):
        super().__init__()
        self.setup_measurement_types()

    def setup_measurement_types(self):
        # Automatically populate dropdown with registered measurements
        measurements = MeasurementRegistry.get_all_measurements()
        self.measurement_type_dropdown.addItems(measurements.keys())
```

Benefits: - Eliminates manual registration in multiple places - Automatically handles GUI updates when new measurements are added - Provides a single source of truth for measurement types - Makes it easier to maintain and extend

New Process for Adding Measurements: 1. Create your measurement class and decorate it with `@MeasurementRegistry.register()` 2. Add your configuration class extending `BaseMeasurementConfig` 3. Add pulseblaster sequence if needed

The GUI will automatically pick up new measurement types without additional changes.

Would you like to see more detailed examples of this implementation?