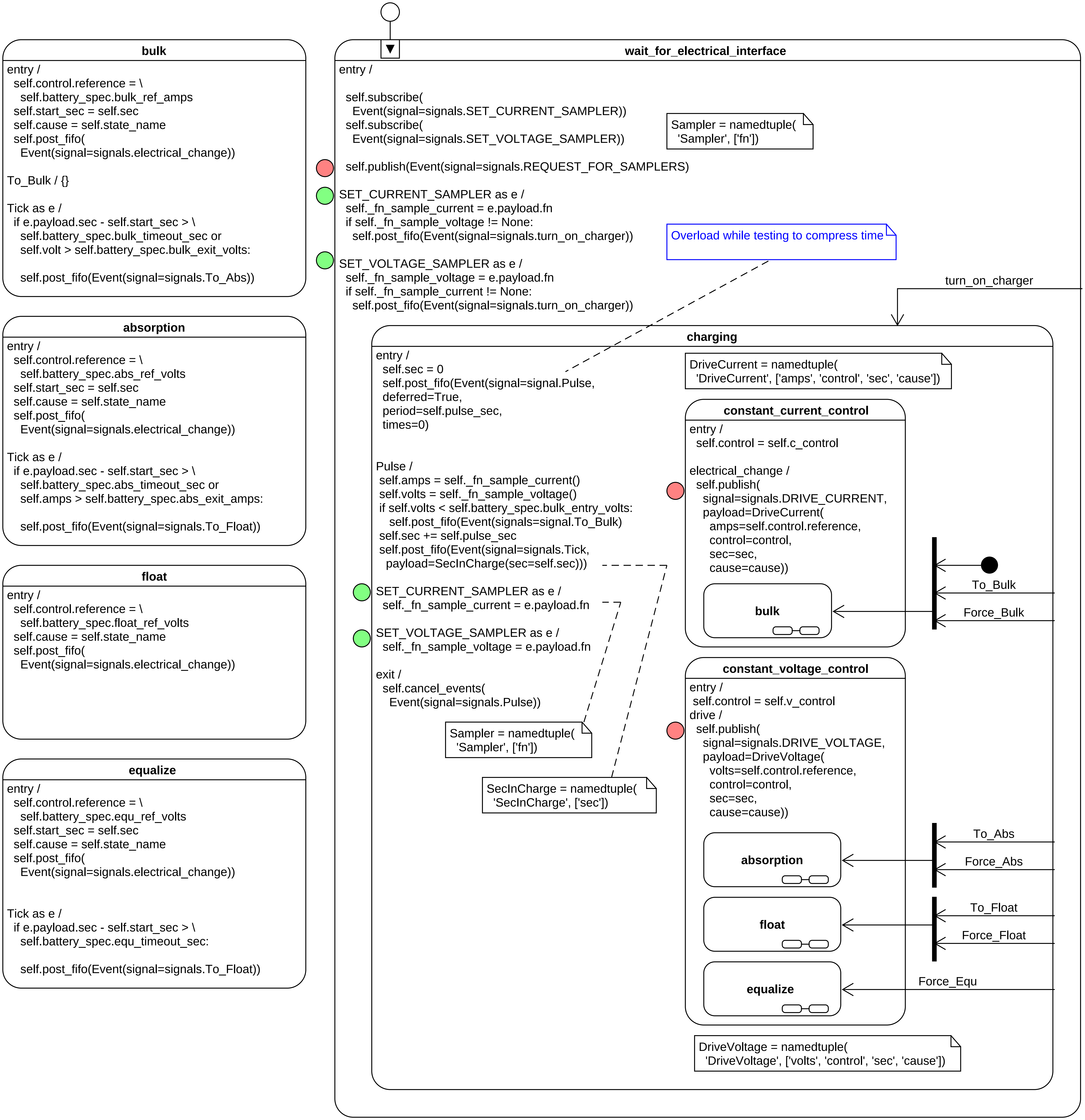

bulk

```
entry /
  self.control.reference = \
    self.battery_spec.bulk_ref_amps
  self.start_sec = self.sec
  self.cause = self.state_name
  self.post_fifo(
    Event(signal=signals.electrical_change))

To_Bulk / {}

Tick as e /
  if e.payload.sec - self.start_sec > \
    self.battery_spec.bulk_timeout_sec or
    self.volt > self.battery_spec.bulk_exit_volts:

    self.post_fifo(Event(signal=signals.To_Abs))
```

absorption

```
entry /
  self.control.reference = \
    self.battery_spec.abs_ref_volts
  self.start_sec = self.sec
  self.cause = self.state_name
  self.post_fifo(
    Event(signal=signals.electrical_change))

Tick as e /
  if e.payload.sec - self.start_sec > \
    self.battery_spec.abs_timeout_sec or
    self.amps > self.battery_spec.abs_exit_amps:

    self.post_fifo(Event(signal=signals.To_Float))
```

float

```
entry /
  self.control.reference = \
    self.battery_spec.float_ref_volts
  self.cause = self.state_name
  self.post_fifo(
    Event(signal=signals.electrical_change))
```

equalize

```
entry /
  self.control.reference = \
    self.battery_spec.equ_ref_volts
  self.start_sec = self.sec
  self.cause = self.state_name
  self.post_fifo(
    Event(signal=signals.electrical_change))


Tick as e /
  if e.payload.sec - self.start_sec > \
    self.battery_spec.equ_timeout_sec:

    self.post_fifo(Event(signal=signals.To_Float))
```

wait_for_electrical_interface

```
entry /

  self.subscribe(
    Event(signal=signals.SET_CURRENT_SAMPLER))
  self.subscribe(
    Event(signal=signals.SET_VOLTAGE_SAMPLER))

  self.publish(Event(signal=signals.REQUEST_FOR_SAMPLERS)

SET_CURRENT_SAMPLER as e /
  self._fn_sample_current = e.payload.fn
  if self._fn_sample_voltage != None:
    self.post_fifo(Event(signal=signals.turn_on_charger))

SET_VOLTAGE_SAMPLER as e /
  self._fn_sample_voltage = e.payload.fn
  if self._fn_sample_current != None:
    self.post_fifo(Event(signal=signals.turn_on_charger))
```

```
Sampler = namedtuple(
  'Sampler', ['fn'])
```

Overload while testing to compress time

charging

```
entry /
  self.sec = 0
  self.post_fifo(Event(signal=signal.Pulse,
  deferred=True,
  period=self.pulse_sec,
  times=0)


Pulse /
 self.amps = self._fn_sample_current()
 self.volts = self._fn_sample_voltage()
 if self.volts < self.battery_spec.bulk_entry_volts:
    self.post_fifo(Event(signals=signal.To_Bulk)
 self.sec += self.pulse_sec
 self.post_fifo(Event(signal=signals.Tick,
   payload=SecInCharge(sec=self.sec)))

SET_CURRENT_SAMPLER as e /
  self._fn_sample_current = e.payload.fn

SET_VOLTAGE_SAMPLER as e /
  self._fn_sample_voltage = e.payload.fn

exit /
  self.cancel_events(
    Event(signal=signals.Pulse))
```

```
Sampler = namedtuple(
  'Sampler', ['fn'])
```

```
SecInCharge = namedtuple(
  'SecInCharge', ['sec'])
```

```
DriveCurrent = namedtuple(
  'DriveCurrent', ['amps', 'control', 'sec', 'cause'])
```

constant_current_control

```
entry /
  self.control = self.c_control

electrical_change /
  self.publish(
    signal=signals.DRIVE_CURRENT,
    payload=DriveCurrent(
      amps=self.control.reference,
      control=control,
      sec=sec,
      cause=cause))
```

bulk

constant_voltage_control

```
entry /
 self.control = self.v_control
drive /
  self.publish(
    signal=signals.DRIVE_VOLTAGE,
    payload=DriveVoltage(
      volts=self.control.reference,
      control=control,
      sec=sec,
      cause=cause))
```

absorption

float

equalize

```
DriveVoltage = namedtuple(
  'DriveVoltage', ['volts', 'control', 'sec', 'cause'])
```