Virtual Neuron Implementation in Python using the NEST Simulator

```python
import nest

class VirtualNeuron:
    """ Virtual neuron is a neuromorphic primitive that mimicks the behavior of an artificial neuron using a collection of spiking neurons. """

    def __init__(self, precision=(4, 4, 4, 4)):
        """ Initializes a virtual neuron object """
        self.precision = precision
        self.positive_precision = precision[0] + precision[1]
        self.negative_precision = precision[2] + precision[3]
        self.total_precision = sum(precision)
        self.higher_precision = max(self.positive_precision, self.negative_precision)

        # Setup incoming neurons
        self.x_positive = {}
        self.x_negative = {}
        self.y_positive = {}
        self.y_negative = {}

        for i in range(self.positive_precision):
            self.x_positive[i] = self.create_neuron(threshold=0)
            self.y_positive[i] = self.create_neuron(threshold=0)

        for i in range(self.negative_precision):
            self.x_negative[i] = self.create_neuron(threshold=0)
            self.y_negative[i] = self.create_neuron(threshold=0)

        # Setup positive bit neurons
        self.bits_positive = {}
        if self.positive_precision > 0:
            for i in range(self.positive_precision + 1):
                self.bits_positive[i] = {}
                self.bits_positive[i][0] = self.create_neuron(threshold=0)
                self.bits_positive[i][1] = self.create_neuron(threshold=1)

                if i > 0:
                    self.bits_positive[i][2] = self.create_neuron(threshold=2)

        # Setup negative bit neurons
        self.bits_negative = {}
        if self.negative_precision > 0:
            for i in range(self.negative_precision + 1):
                self.bits_negative[i] = {}
                self.bits_negative[i][0] = self.create_neuron(threshold=0)
                self.bits_negative[i][1] = self.create_neuron(threshold=1)

                if i > 0:
                    self.bits_negative[i][2] = self.create_neuron(threshold=2)

        # Setup outgoing neurons
        self.z_positive = {}
        self.z_negative = {}

        if self.positive_precision > 0:
            for i in range(self.positive_precision + 1):
                self.z_positive[i] = self.create_neuron(threshold=0)
```

1/3
if self.negative_precision > 0:
    for i in range(self.negative_precision + 1):
        self.z_negative[i] = self.create_neuron(threshold=0)

# Neurons created
print("Neurons created...")

# Setup synapses between positive incoming neurons and positive bit neurons
for i in range(self.positive_precision):
    nest.Connect(self.x_positive[i], self.bits_positive[i][0], syn_spec={"weight": 1.0, "delay": float(i+1)})
    nest.Connect(self.x_positive[i], self.bits_positive[i][1], syn_spec={"weight": 1.0, "delay": float(i+1)})
    nest.Connect(self.y_positive[i], self.bits_positive[i][0], syn_spec={"weight": 1.0, "delay": float(i+1)})
    nest.Connect(self.y_positive[i], self.bits_positive[i][1], syn_spec={"weight": 1.0, "delay": float(i+1)})
    if i > 0:
        nest.Connect(self.x_positive[i], self.bits_positive[i][2], syn_spec={"weight": 1.0, "delay": float(i+1)})
        nest.Connect(self.y_positive[i], self.bits_positive[i][2], syn_spec={"weight": 1.0, "delay": float(i+1)})

# Setup synapses between negative incoming neurons and negative bit neurons
for i in range(self.negative_precision):
    nest.Connect(self.x_negative[i], self.bits_negative[i][0], syn_spec={"weight": 1.0, "delay": float(i+1)})
    nest.Connect(self.x_negative[i], self.bits_negative[i][1], syn_spec={"weight": 1.0, "delay": float(i+1)})
    nest.Connect(self.y_negative[i], self.bits_negative[i][0], syn_spec={"weight": 1.0, "delay": float(i+1)})
    nest.Connect(self.y_negative[i], self.bits_negative[i][1], syn_spec={"weight": 1.0, "delay": float(i+1)})
    if i > 0:
        nest.Connect(self.x_negative[i], self.bits_negative[i][2], syn_spec={"weight": 1.0, "delay": float(i+1)})
        nest.Connect(self.y_negative[i], self.bits_negative[i][2], syn_spec={"weight": 1.0, "delay": float(i+1)})

# Setup carry synapses in positive bits
for i in range(self.positive_precision):
    nest.Connect(self.bits_positive[i][1], self.bits_positive[i+1][0], syn_spec={"weight": 1.0, "delay": float(1.0)})
    nest.Connect(self.bits_positive[i][1], self.bits_positive[i+1][1], syn_spec={"weight": 1.0, "delay": float(1.0)})
    nest.Connect(self.bits_positive[i][1], self.bits_positive[i+1][2], syn_spec={"weight": 1.0, "delay": float(1.0)})

# Setup carry synapses in negative bits
for i in range(self.negative_precision):
    nest.Connect(self.bits_negative[i][1], self.bits_negative[i+1][0], syn_spec={"weight": 1.0, "delay": float(1.0)})
    nest.Connect(self.bits_negative[i][1], self.bits_negative[i+1][1], syn_spec={"weight": 1.0, "delay": float(1.0)})
    nest.Connect(self.bits_negative[i][1], self.bits_negative[i+1][2], syn_spec={"weight": 1.0, "delay": float(1.0)})

# Setup synapses between positive bit neurons and positive outgoing neurons
if self.positive_precision > 0:
    for i in range(self.positive_precision + 1):
        nest.Connect(self.bits_positive[i][0], self.z_positive[i], syn_spec={"weight": 1.0, "delay": float(1/3)}, syn_spec={"weight": 1.0, "delay": float(1/3)}, syn_spec={"weight": 1.0, "delay": float(1/3)})
```python
float(self.higher_precision - i + 1))
    nest.Connect(self.bits_positive[i][1], self.z_positive[i], syn_spec={"weight": -1.0, "delay":
float(self.higher_precision - i + 1))
    if i > 0:
        nest.Connect(self.bits_positive[i][2], self.z_positive[i], syn_spec={"weight": 1.0, "delay":
float(self.higher_precision - i + 1))

# Setup synapses between negative bit neurons and negative outgoing neurons
if self.negative_precision > 0:
    for i in range(self.negative_precision + 1):
        nest.Connect(self.bits_negative[i][0], self.z_negative[i], syn_spec={"weight": 1.0, "delay":
float(self.higher_precision - i + 1))
        nest.Connect(self.bits_negative[i][1], self.z_negative[i], syn_spec={"weight": -1.0, "delay":
float(self.higher_precision - i + 1))
        if i > 0:
            nest.Connect(self.bits_negative[i][2], self.z_negative[i], syn_spec={"weight": 1.0, "delay":
float(self.higher_precision - i + 1))

# Synapses created
print("Synapses created...")
print("Virtual neuron created...")

def create_neuron(self, V_th=0, internal_state=-1.0):
    """ Creates an "iaf_psc_delta" neuron in NEST
    Params:
    V_th: Threshold voltage
    internal_state: Default internal state of the neuron
    """
    neuron = nest.Create("iaf_psc_delta")
    neuron.V_th = V_th
    neuron.V_m = internal_state  # Membrane potential
    neuron.V_reset = -1e-6  # Leak
    neuron.tau_m = 1e-6  # Leak
    neuron.t_ref = 0.0
    neuron.E_L = neuron.V_m # Resting membrane potential
    refractory_input = False
    return neuron
```
