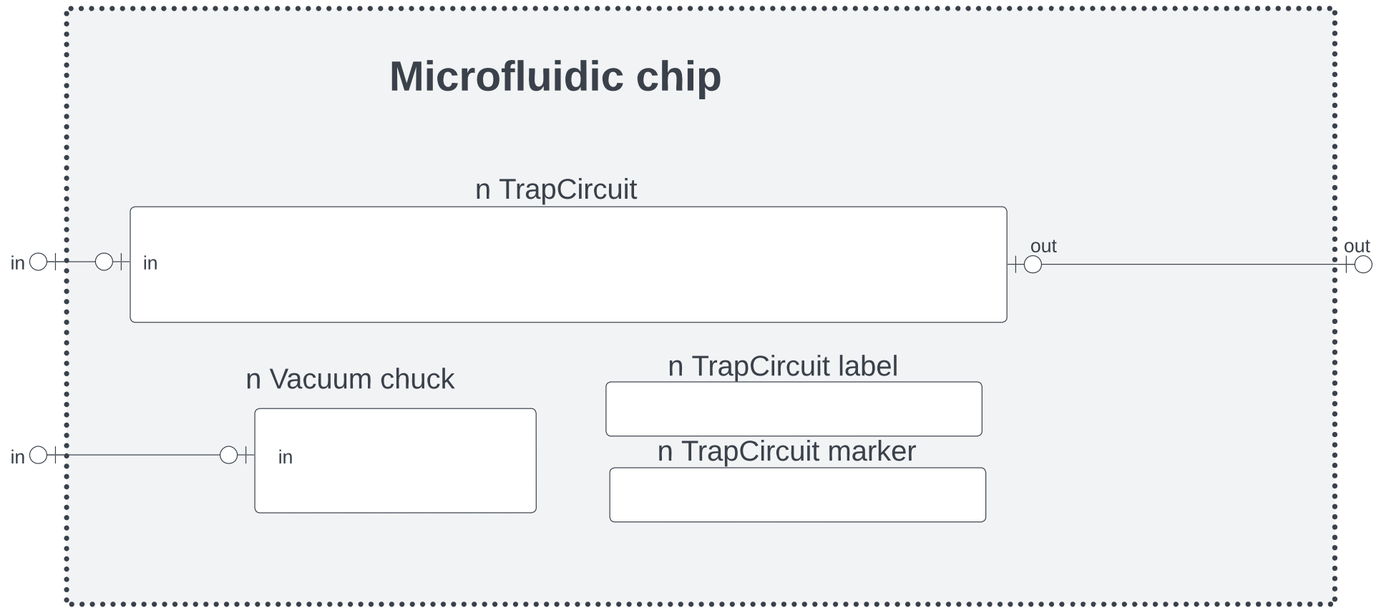
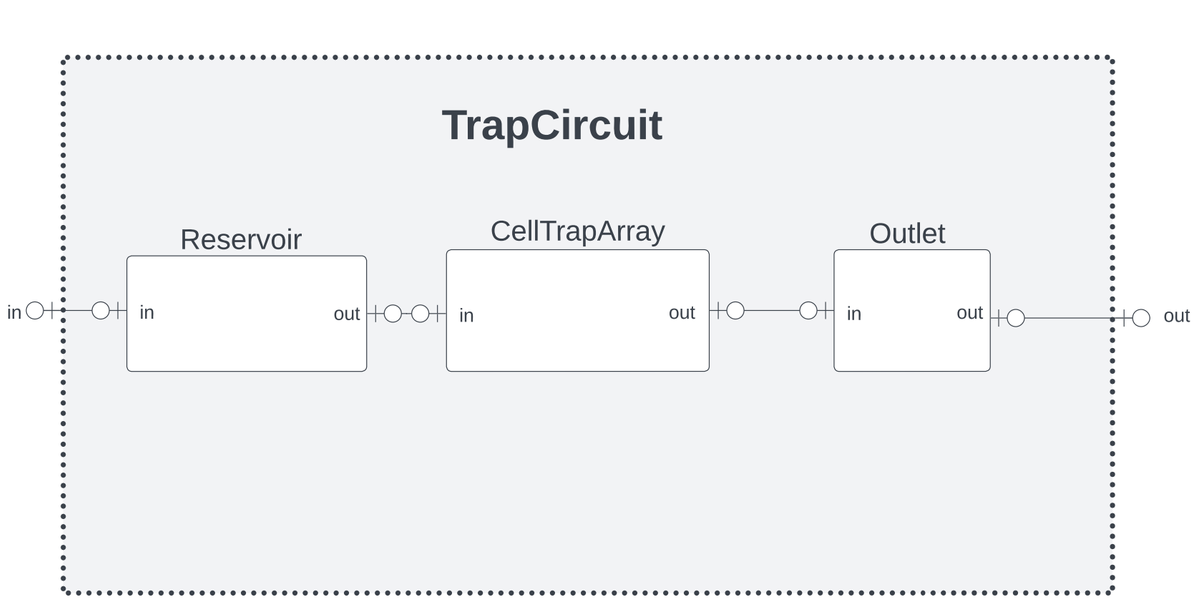
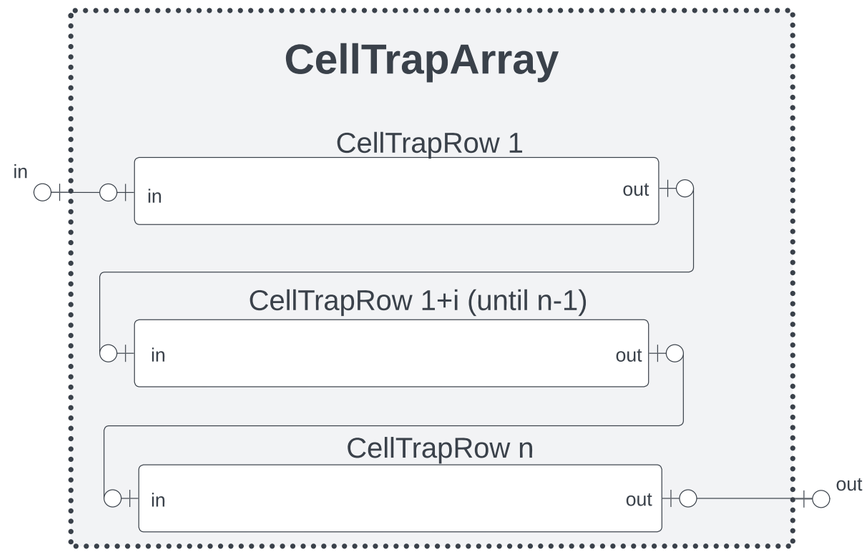
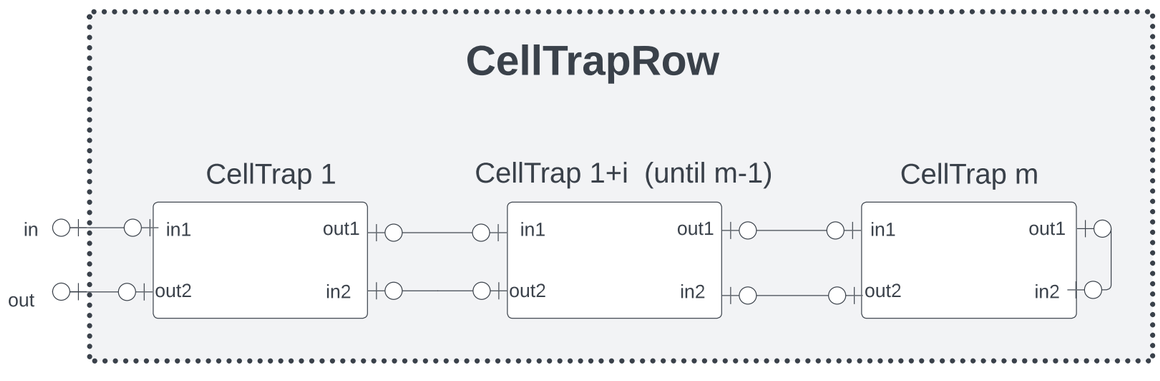
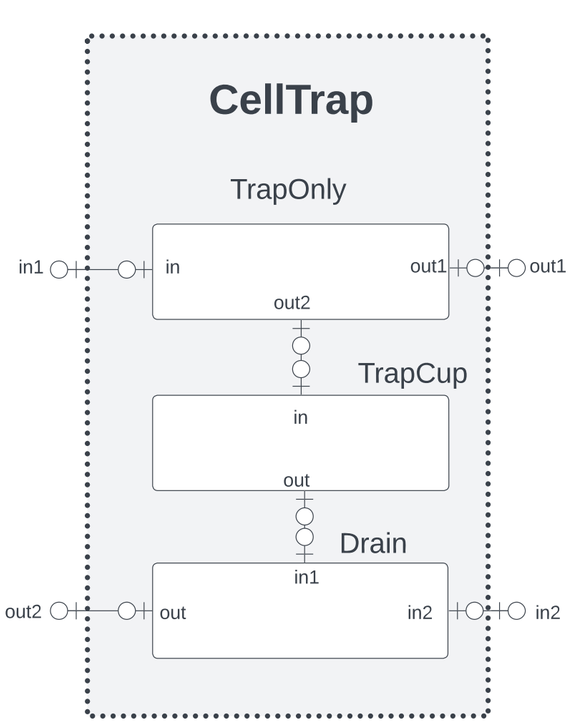

Microfluidic chip
n TrapCircuit
in
in
out
out
n Vacuum chuck
in
in
n TrapCircuit label
n TrapCircuit marker

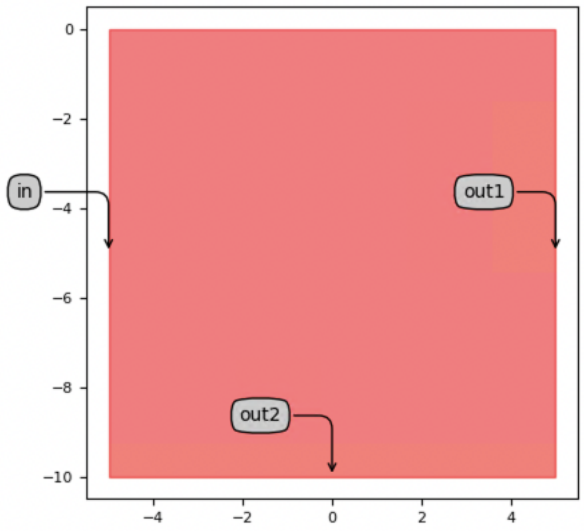

TrapCircuit
Reservoir
CellTrapArray
Outlet
in
in
out
in
out
in
out
out


CellTrapArray
CellTrapRow 1
in
in
out
CellTrapRow 1+i (until n-1)
in
out
CellTrapRow n
in
out
out


CellTrapRow
CellTrap 1
CellTrap 1+i (until m-1)
CellTrap m
in
out
in1
out2
out1
in2
in1
out2
out1
in2
in1
out2
out1
in2


CellTrap
TrapOnly
in1
in
out1
out1
out2
TrapCup
in
out
Drain
in1
out2
out
in2
in2

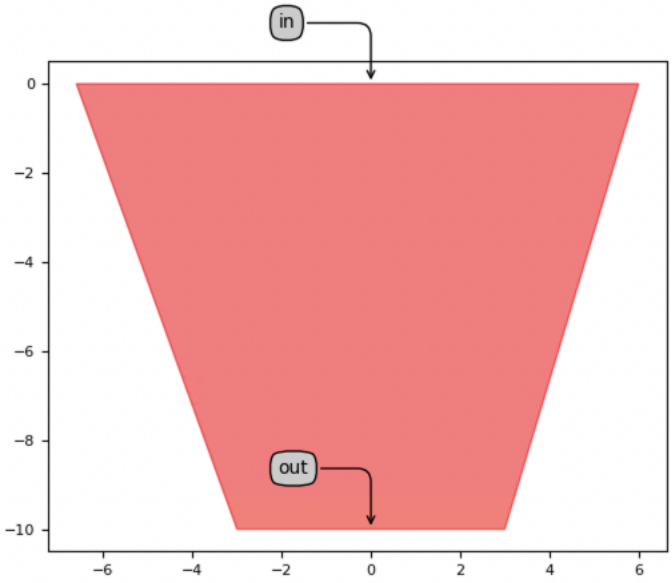
TrapOnly




```
def _generate_ports(self, ports):
    # input port
    ports += microfluidics.FluidicPort(name='in', position = (-self.in_length*0.5, -self.in_length*0.5),
                                       direction = i3.PORT_DIRECTION.IN,
                                       angle_deg=90,
                                       trace_template=self.in_channel_template
                                       )

    ports += microfluidics.FluidicPort(name='out1', position = (self.in_length*0.5, -self.out_length*0.5),
                                       direction = i3.PORT_DIRECTION.OUT,
                                       angle_deg=270,
                                       trace_template=self.out_channel_template
                                       )

    ports += microfluidics.FluidicPort(name='out2', position = (0.0, -self.out_length),
                                       direction = i3.PORT_DIRECTION.OUT,
                                       angle_deg=270,
                                       trace_template=self.out_channel_template
                                       )

    return ports
```

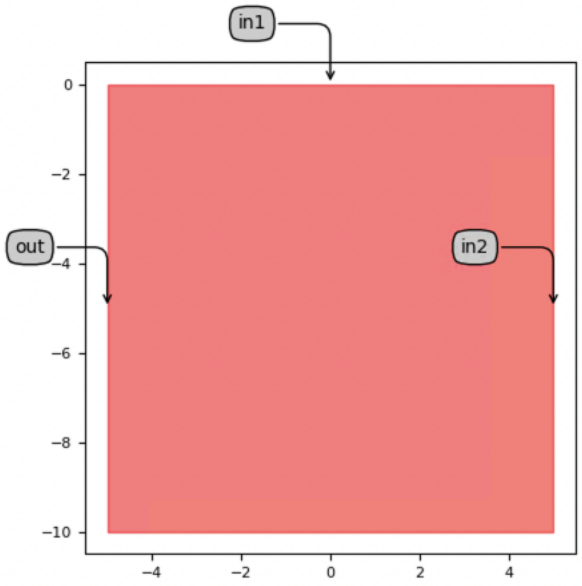
TrapCup

```
class Layout(_TrapCup.Layout):
    def _generate_elements(self, elems):
        in_width = self.in_channel_template.channel_width
        out_width = self.out_channel_template.channel_width
        point_list = [(in_width * 0.5, 0),
                      (in_width * 0.25, -self.in_length),
                      (-in_width * 0.25, -self.in_length),
                      (-in_width * 0.55, 0)
                      ]

        shape = i3.Shape(point_list, closed=True)
        boundary = i3.Boundary(self.in_channel_template.layer, shape)
        elems += boundary

        return elems
```



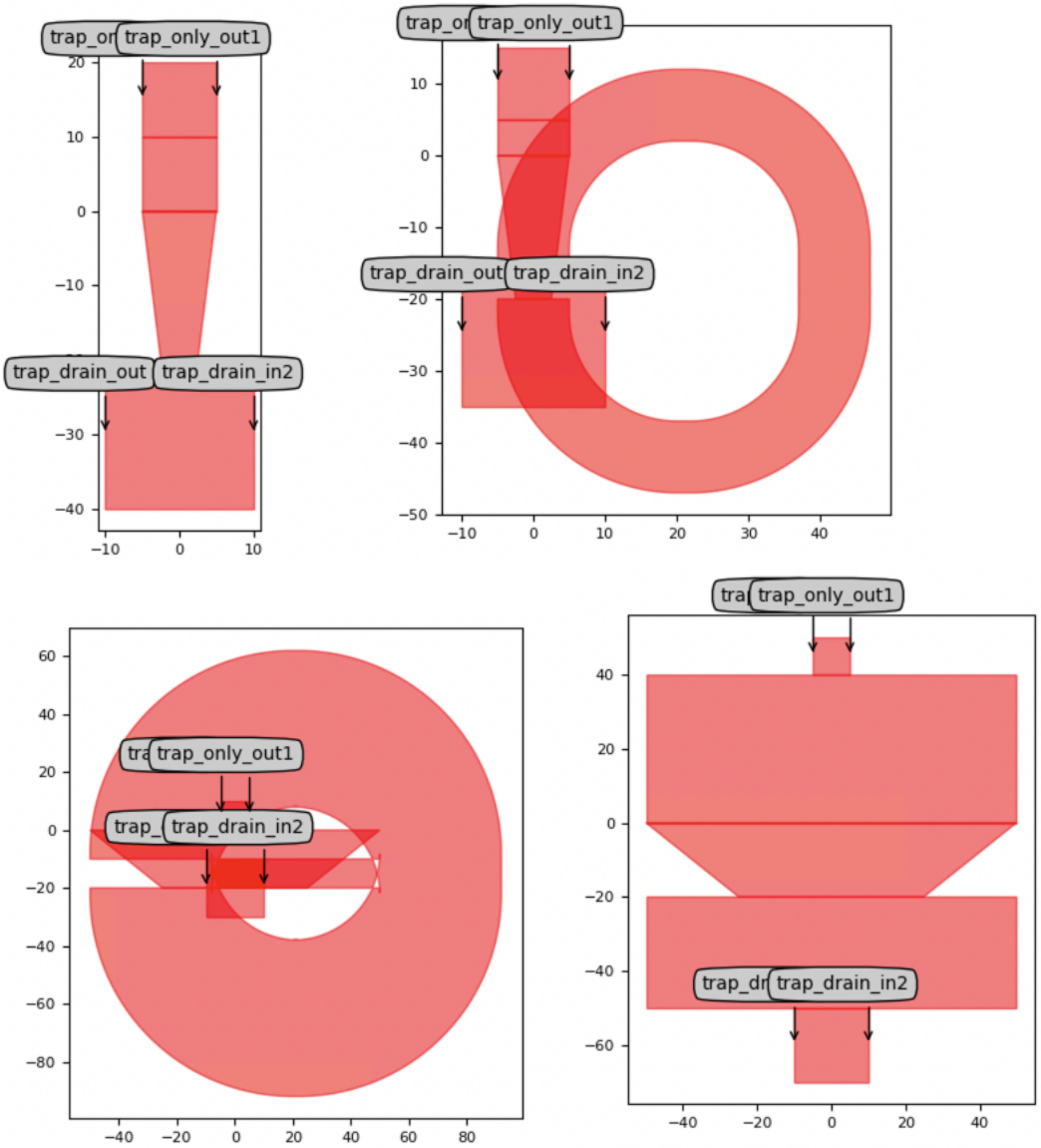

```
class Layout(i3.LayoutView):
    in_length = i3.PositiveNumberProperty(default=20, doc="Input part of the trap")
    out_length = i3.PositiveNumberProperty(default=20, doc="Output part of the trap")

    def _generate_ports(self, ports):
        # input port
        ports += microfluidics.FluidicPort(name='in', position = (0.0, 0.0),
                                           direction = i3.PORT_DIRECTION.IN,
                                           angle_deg=90,
                                           trace_template=self.in_channel_template
                                           )

        ports += microfluidics.FluidicPort(name='out', position = (0.0, -self.out_length),
                                           direction = i3.PORT_DIRECTION.OUT,
                                           angle_deg=270,
                                           trace_template=self.out_channel_template
                                           )

        return ports


class Netlist(i3.NetlistFromLayout):
    pass
```

TrapDrain




```
3 usages
class TrapDrain(_TrapDrain):
    """
    Trap drain with rectangle shapes
    """

    class Layout(_TrapDrain.Layout):
        def _generate_elements(self, elems):
            in_width = self.in_length #self.in_channel_template.channel_width
            out_width = self.out_length #self.out_channel_template.channel_width
            point_list = [(in_width * 0.5, 0),
                          (in_width * 0.5, -self.in_length),
                          (-in_width * 0.5, -self.in_length),
                          (-in_width * 0.5, 0)
                          ]

            shape = i3.Shape(point_list, closed=True)
            boundary = i3.Boundary(self.in_channel_template.layer, shape)
            elems += boundary

            return elems
```

CellTrap


trap_only_out1
trap_drain_out
trap_drain_in2
trap_only_out1
trap_drain_out
trap_drain_in2
trap_only_out1
trap_drain_in2
trap_only_out1
trap_drain_in2