In [62]:
```
import foldable_robotics.dxf as frd
import foldable_robotics as fr
import foldable_robotics.manufacturing as frm
from foldable_robotics.layer import Layer
from foldable_robotics.laminate import Laminate
import foldable_robotics.parts.castellated_hinge2 as frc
import shapely.geometry as sg
```

In [63]:
```
fr.display_height=300
fr.resolution = 4
desired_degrees = 120
thickness = 1
plain_width = frm.plain_hinge_width(desired_degrees,thickness)
plain_width
```

Out[63]:
```
1.7320508075688774
```

In [64]:
```
support_width = 2 # must be larger than hinge width
kerf = .05
is_adhesive = [False,True,False,True,False]
arc_approx = 10
NUM_LAYERS = 5
bridge_thickness = 2
bounding_box_padding = 10
jig_spacing = 10
jig_dia = 5
```

In [65]:
```
body_vertices = frd.read_lwpolylines('body.dxf',
layer='body',
arc_approx = arc_approx)
body_vertices
```

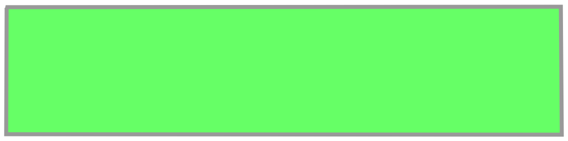
Out[65]:
```
[[[70.11354420113561, 179.2749391727498],
  [69.89781021897826, 119.9480940794812],
  [330.0729927007307, 119.7323600973239],
  [329.641524736416, 179.7064071370645],
  [69.68207623682093, 179.4906731549072]]]
```

In [66]:
```
body_polygons = [sg.Polygon(item) for item in body_vertices]
body_polygons[0]
```

Out[66]:

In [67]:
```
body_layer = Layer(*body_polygons)
body_layer
```

Out[67]:



In [68]:

```
cut_vertices = frd.read_lwpolylines('body.dxf', layer='cut', arc_approx=arc_approx)

#print(f"The number of original objects: {len(cut_vertices)}")

# Clean and verify the data
cleaned_polygons = []
for i, v in enumerate(cut_vertices):
    # Remove duplicate adjacent vertices
    unique_vertices = [v[0]]
    for point in v[1:]:
        last_point = unique_vertices[-1]
        # If the points are not repeated (the distance is greater than 0.001
        if abs(point[0] - last_point[0]) > 0.001 or abs(point[1] - last_point[1]) >
            unique_vertices.append(point)

    # Check if there are enough vertices
    if len(unique_vertices) >= 3:
        try:
            poly = sg.Polygon(unique_vertices)
            if poly.is_valid and poly.area > 0.001:  # Make sure the area is not ze
                cleaned_polygons.append(poly)
                print(f"object {i}: {len(unique_vertices)} effect vertex")
            else:
                print(f"Object {i}: invalid or the area is 0")
        except Exception as e:
            print(f"object {i}: error - {e}")
    else:
        print(f"object {i}: vertex not enough ({len(unique_vertices)} < 3)")

print(f"\nThe number of valid objects after cleaning: {len(cleaned_polygons)}")

# Create Layer
if cleaned_polygons:
    cut_layer = Layer(*cleaned_polygons)
    cut_layer.plot()
else:
    cut_layer = Layer()
```

```
object 0: vertex not enough (1 < 3)
object 1: 4 effect vertex
object 2: 4 effect vertex
object 3: 4 effect vertex
object 4: 4 effect vertex

The number of valid objects after cleaning: 4
```

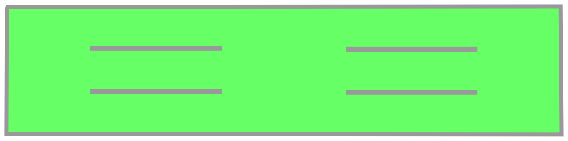
In [69]:
```
body_layer -= cut_layer
```

In [70]:
```
hole_vertices = frd.read_lwpolylines('body.dxf', layer='holes', arc_approx = arc_ap
hole_layer = Layer(*[sg.Polygon(item) for item in hole_vertices])
hole_layer
```

Out[70]:

In [71]:
```
body_layer -= hole_layer
body_layer
```

Out[71]:



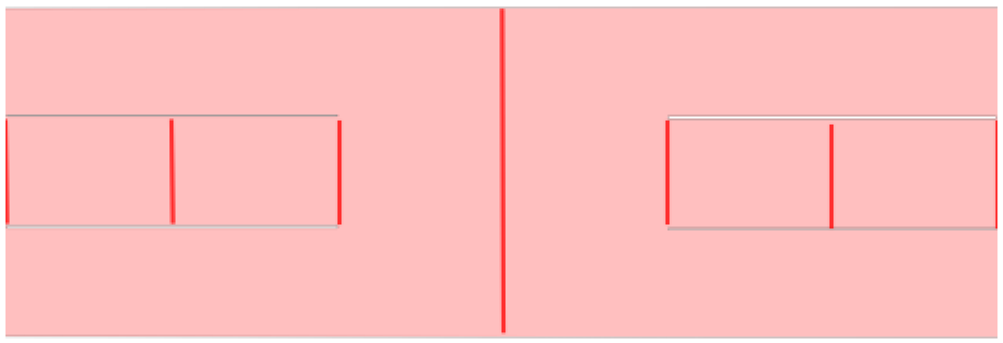
In [72]:

```python
joint_vertices = frd.read_lines('body.dxf', layer='joint')
# Create a joint line layer
joint_lines_original_layer = Layer(*[sg.LineString(item) for item in joint_vertices
joint_lines_original_layer.plot()

# Take the intersection with body_layer to obtain the trimmed joint line
joint_lines_modified_layer = joint_lines_original_layer & body_layer
body_layer.plot()
joint_lines_modified_layer.plot()

modified_joint_vertices = [list(item.coords) for item in joint_lines_modified_layer
print(f"Find {len(modified_joint_vertices)} joints line")
```

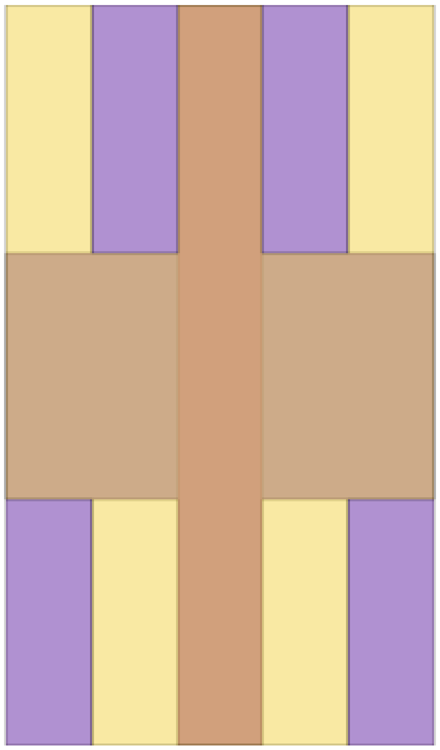
```
Find 7 joints line
```



In [73]:
```
castellated_width,castellated_gap = \
frm.castellated_hinge_width(desired_degrees,thickness)
print(plain_width,castellated_gap,castellated_width)
```

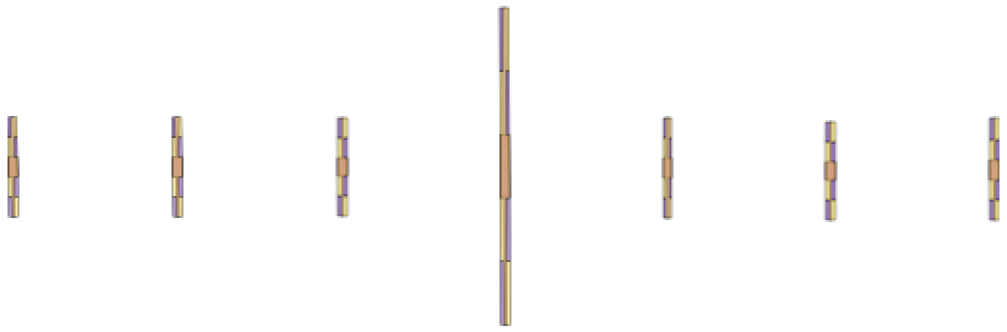
```
1.7320508075688774 1.7320508075688774 0.577350269189626
```

In [74]:
```
hinge = frc.generate(castellated_gap,castellated_width)
hinge.plot()
```

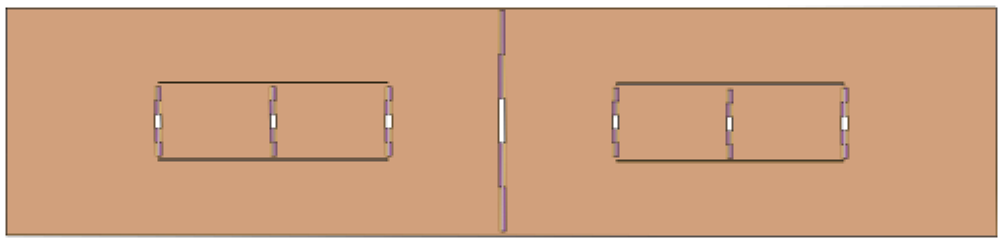


In [75]:
```
support_width = 1
```

In [76]:
```
lam = Layer().to_laminate(len(hinge))
all_hinges = []
for p3,p4 in modified_joint_vertices:
    all_hinges.append(hinge.map_line_stretch((0,0),(1,0),p3,p4))  # Properly indent
all_hinges = lam.unary_union(*all_hinges)
all_hinges.plot()
```



In [77]:
```
actual_final_device = Laminate(body_layer,body_layer,body_layer,body_layer,body_lay
actual_final_device -= all_hinges
actual_final_device.plot()
```

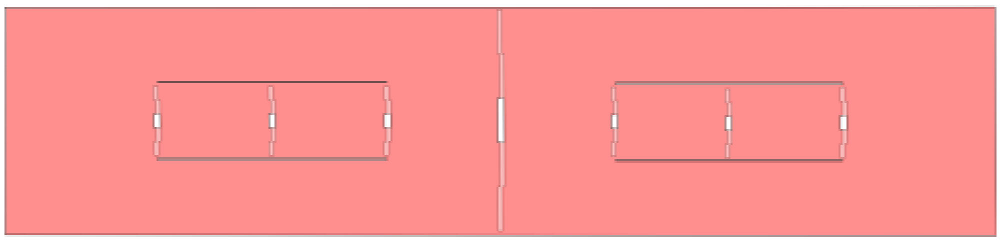


In [78]:
```
hole,dummy = frm.calc_hole(modified_joint_vertices,plain_width/2)
fr.my_line_width=0
holes = hole.to_laminate(NUM_LAYERS)
```

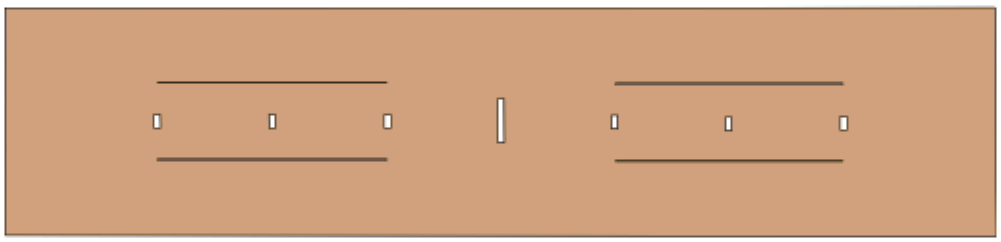
```
holes<<=.5 # add a little extra material to ensure we removed enough.
holes.plot()
```

```
('zero-size array to reduction operation minimum which has no identity',)
<Figure size 640x480 with 0 Axes>
```

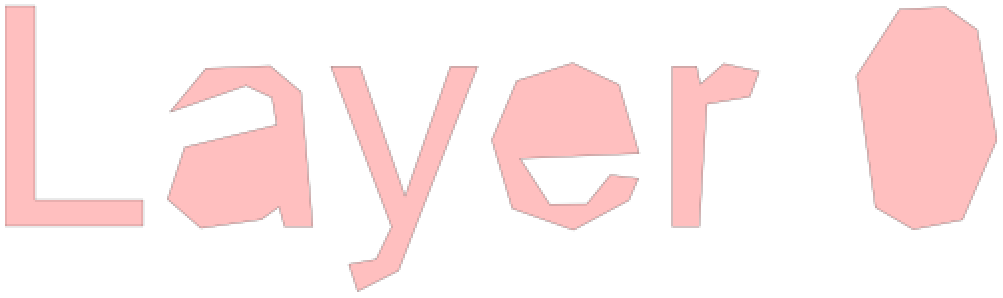
In [79]:
```
# Visualize each layer
actual_final_device[0].plot()
actual_final_device[2].plot()
```



In [80]:
```
keepout = frm.keepout_laser(actual_final_device)
keepout.plot()
```



In [81]:
```
layer_id = frm.build_layer_numbers(NUM_LAYERS, text_size=jig_dia)
layer_id = layer_id.simplify(.2)
layer_id[0].plot()
```

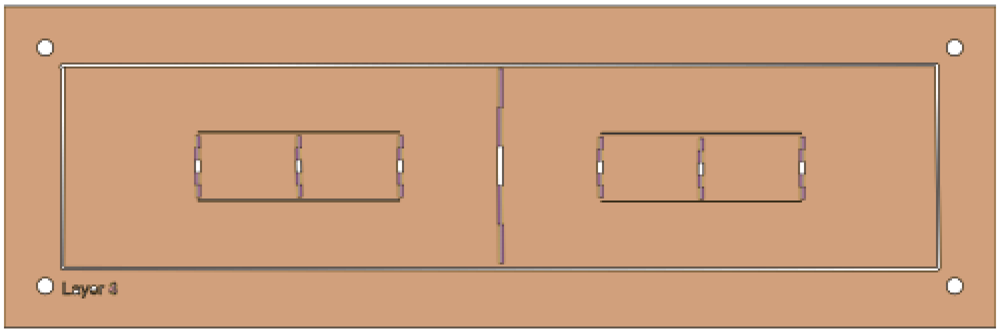


In [82]:
```
# Calculate alignment holes
(x1,y1),(x2,y2) = actual_final_device.bounding_box_coords()
w1,h1 = actual_final_device.get_dimensions()
w2 = round(w1/jig_spacing)*jig_spacing+jig_spacing+support_width
```

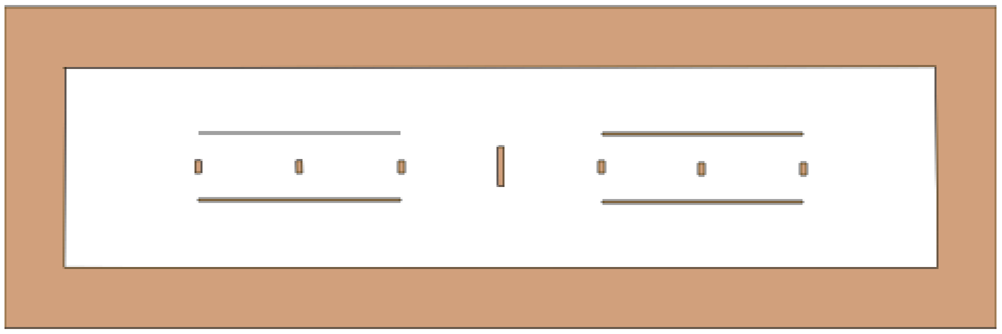
```
h2 = round(h1/jig_spacing)*jig_spacing+jig_spacing+support_width
x1 -= (w2-w1)/2
y1 -= (h2-h1)/2
x2 += (w2-w1)/2
y2 += (h2-h1)/2
points = []
points.append(sg.Point(x1,y1))
points.append(sg.Point(x2,y1))
points.append(sg.Point(x1,y2))
points.append(sg.Point(x2,y2))
alignment_holes_layer = Layer(*points)
alignment_holes_layer<<=(jig_dia/2)
alignment_holes=alignment_holes_layer.to_laminate(NUM_LAYERS)
alignment_holes.plot()
```

In [83]:
```
# Generate material board
sheet_layer = (alignment_holes_layer<<bounding_box_padding).bounding_box()
sheet=sheet_layer.to_laminate(NUM_LAYERS)
sheet.plot()
```

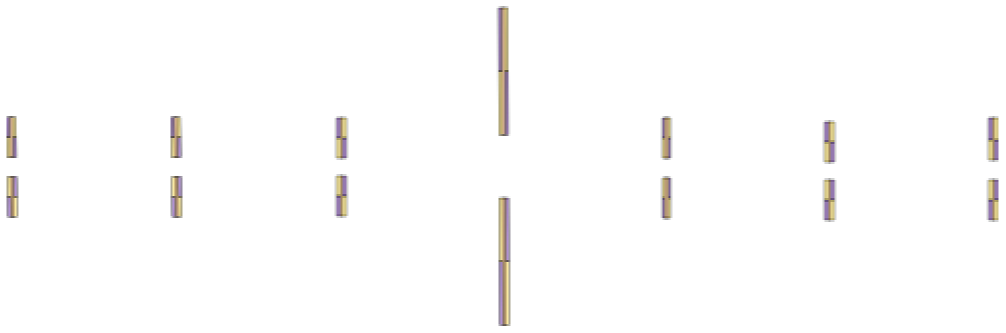
In [84]:
```
# Calculate removable waste
removable_scrap = frm.calculate_removable_scrap(actual_final_device,sheet,support_w
web = removable_scrap-alignment_holes-layer_id.translate(x1+jig_dia,y1-jig_dia/2)
(web | actual_final_device).plot()
```



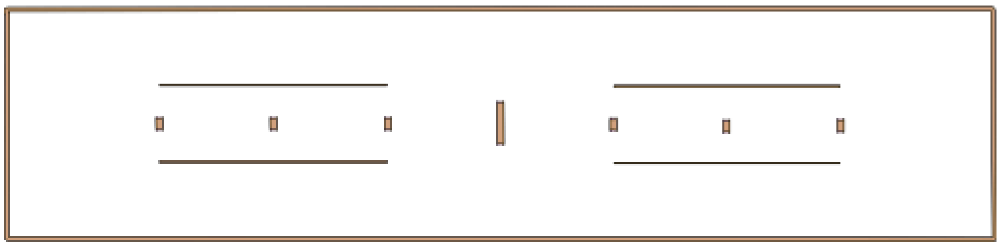
In [85]:
```
# The second cutting of waste materials
second_pass_scrap = sheet-keepout
second_pass_scrap.plot()
```



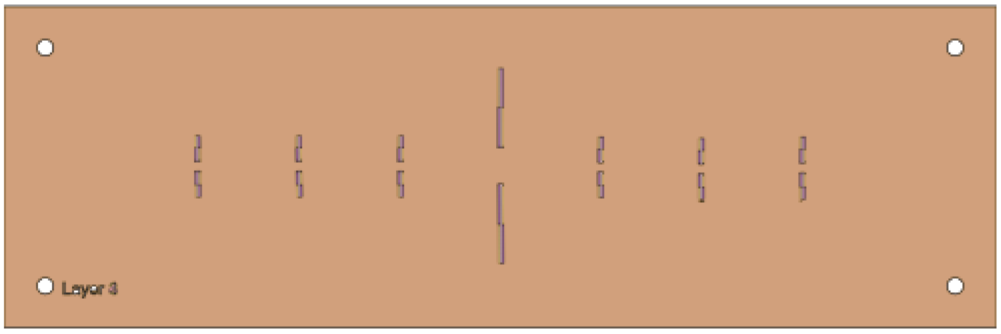
In [86]:
```
first_pass_scrap = sheet - second_pass_scrap - actual_final_device
first_pass_scrap = frm.cleanup(first_pass_scrap,.00001)
first_pass_scrap.plot()
```



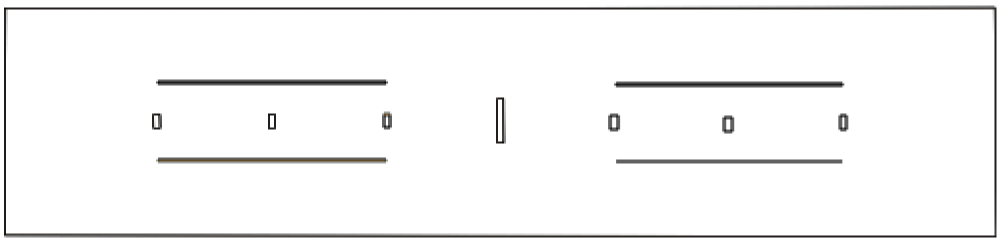
In [87]:
```
# Generate the supporting structure
support = frm.support(
    actual_final_device,
    frm.keepout_laser,
    support_width,
    support_width/2)
support.plot()
```



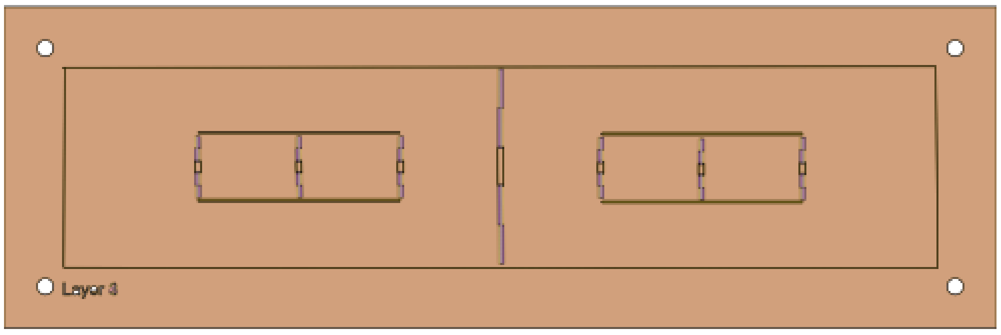
In [88]:
```
# support design
supported_design = web|actual_final_device|support
supported_design.plot()
```



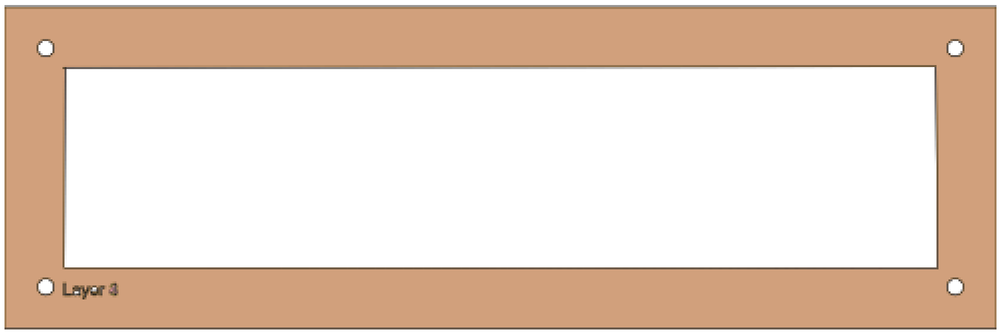
In [89]:
```
# cutting material
cut_material = (keepout<<kerf)-keepout
cut_material.plot()
```

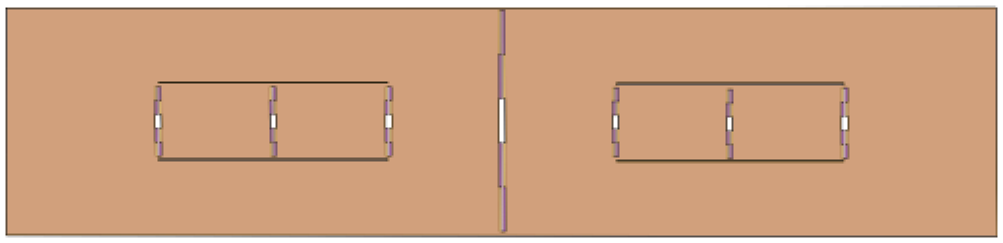


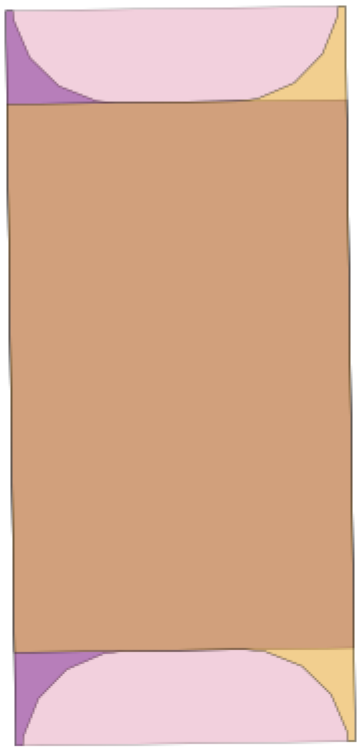
In [90]:
```
# surplus material
remaining_material = supported_design-cut_material
remaining_material.plot()
```

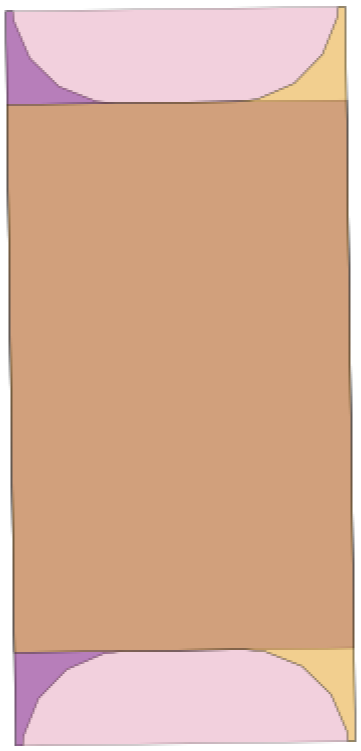


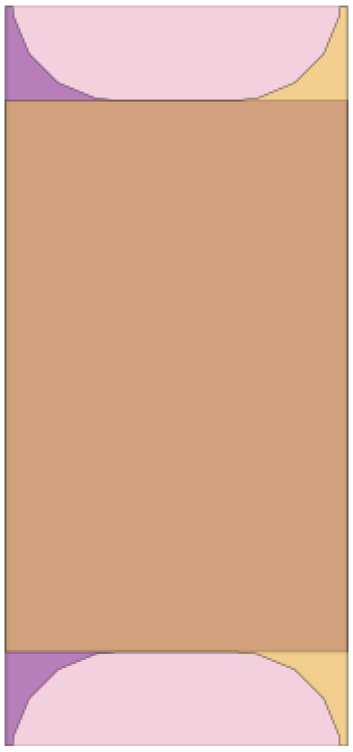
```
In [91]: # Search for the connected part
         remaining_parts = frm.find_connected(remaining_material, is_adhesive)
         for item in remaining_parts:
             item.plot(new=True)  # This line is now properly indented with 4 spaces
```

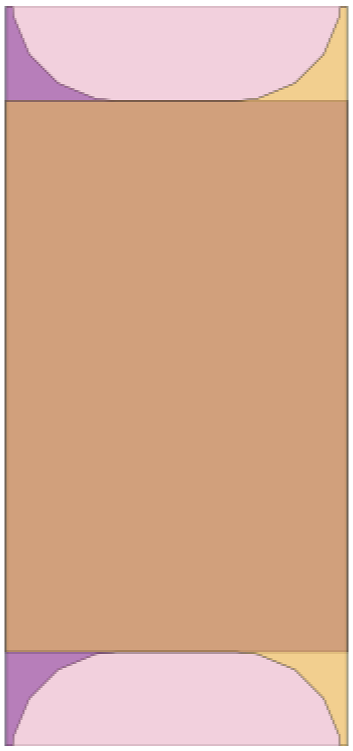


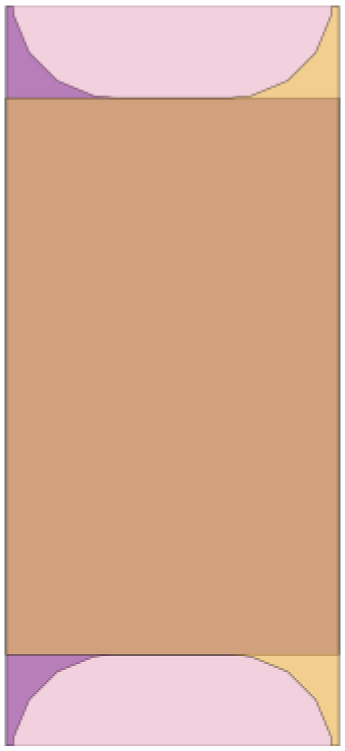


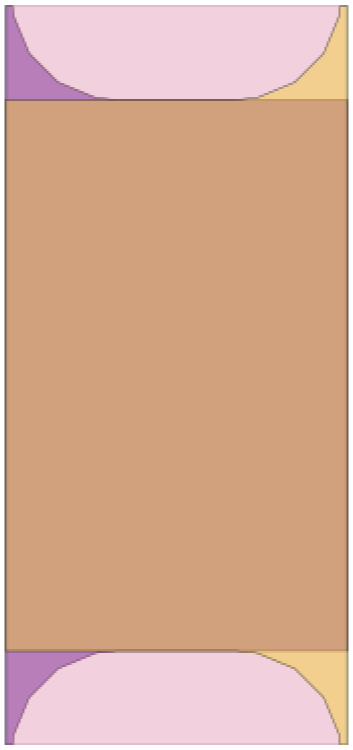


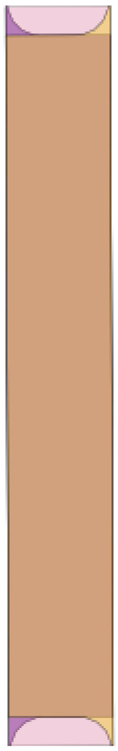


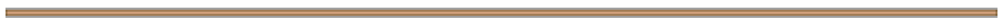




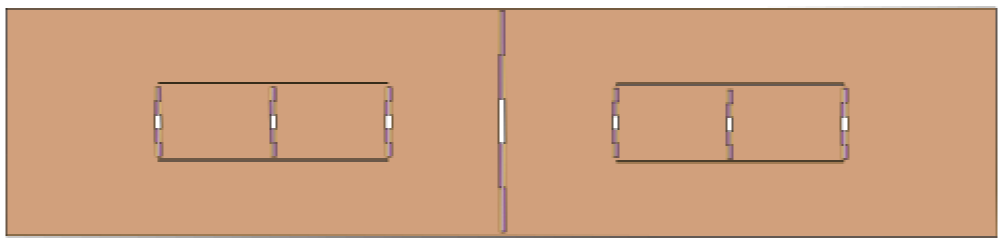






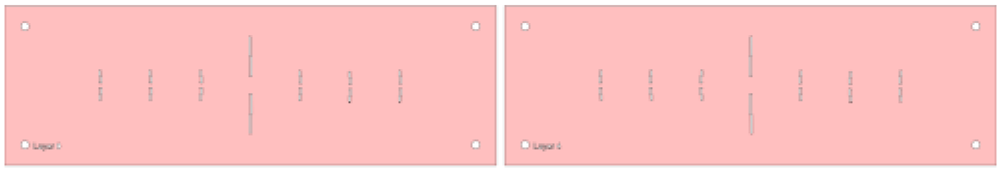


In [92]:
```python
# Assistant
test_part = actual_final_device >> 1
for result in remaining_parts:
    if not (result & test_part).is_null():
        break
result.plot()
```

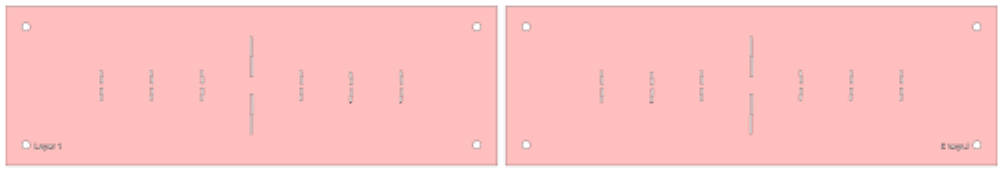


In [93]:
```
check = (result^actual_final_device)
check>>=1e-1
assert(check.is_null())
```

In [94]:
```
# Generate the final output file
w,h = supported_design.get_dimensions()
p0,p1 = supported_design.bounding_box_coords()
# rigid layer
rigid_layer = supported_design[0] | (supported_design[-1].translate(w+5,0))
rigid_layer.plot()
```



In [95]:
```
# adhesive phase
l4 = supported_design[3].scale(-1,1)
p2,p3 = l4.bounding_box_coords()
l4 = l4.translate(p0[0]-p2[0]+w+5,p0[1]-p2[1])
adhesive_layer = supported_design[1] | l4
adhesive_layer.plot()
```

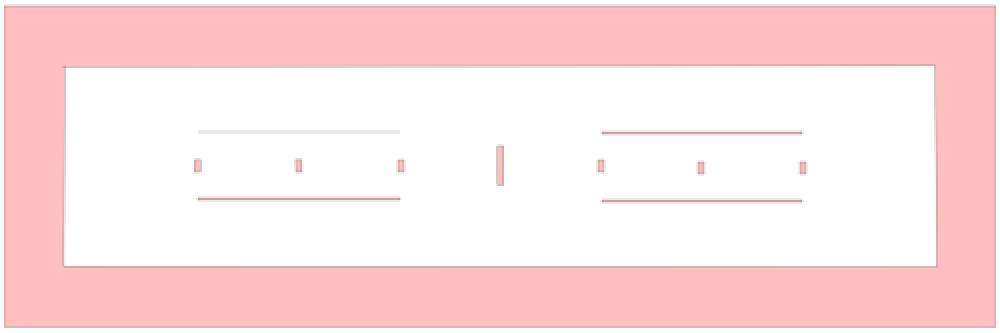


```
In [96]: # The first time cutting the file
         first_pass = Laminate(rigid_layer,adhesive_layer,supported_design[2])
         first_pass.export_dxf('first_pass')
         print("Created first_pass.dxf")
```

```
Created first_pass.dxf
```

```
In [97]: # Final cut file
         final_cut = sheet - keepout
         final_cut = final_cut[0]
         final_cut.export_dxf('final_cut')
         print("Created final_cut.dxf")
         final_cut.plot()
```

```
Created final_cut.dxf
```



```
In [ ]:
```

```
In [ ]:
```