



Effective PCB Design CS449 Project

Ananthapadmanabhan A - 20D070010

Aditya Byju - 20D070004

Course instructor : Prof. G. Sivakumar



Aim

- Create a constraint model to find a plausible PCB design for a given circuit using least area and components.
- Plan to find a suitable model using a set of constraints and use an UI for display and output.
- The output should be able to display a PCB layout which represents the given circuit as well as make sure the layout is the most efficient one (least area).



Program Components

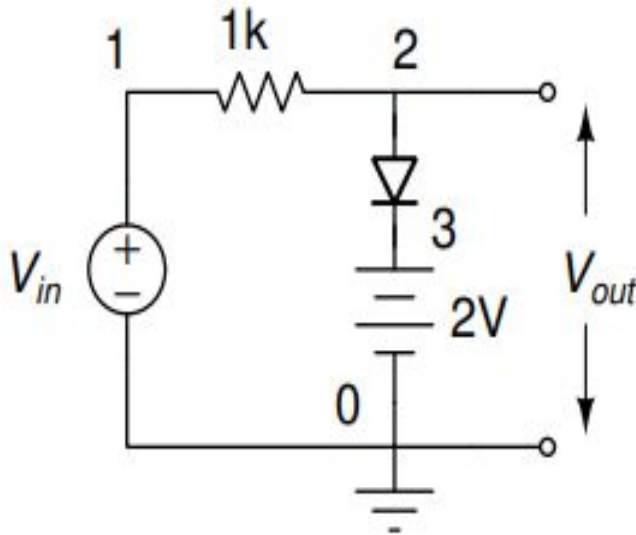
- The frontend is implemented using a Python interface. Inputs and output printing will be taken care of by a Python program.
- The rest of the program is written in Clingo. The Python interface gives the input data to the Clingo terminal which produces the required output for the user.



Inputs/Outputs

- Inputs will be a number of circuit components with the connections specified with each component. The components may also have some extra information like its size or number of ports which will be used by clingo to determine the output. Some tentative components designed are resistors, capacitors, inductors, transformers, sensors, transistors etc.
- Output will be a PCB diagram drawn using python consisting of the required components.

Input Format (NGSpice inspired)



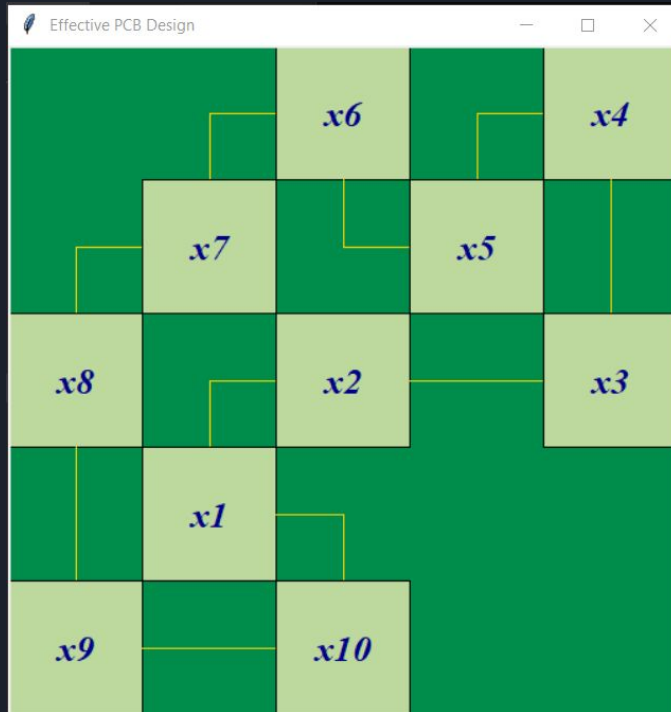
```
r1 1 2
d1 2 3
VDC 3 0 2x3
vin 1 0 3x4
```



Input format(contd.)

The above slide shows a simple circuit and its python input format. Each line represents a component and contains data about that component. The resistor and diode just has the connection points specified as the rest of the data is predefined but the voltage sources have an extra area component as there is no predefined area for the specific component.

Output Format



The picture on the side shows an output from python making use of our Clingo code. We can see that the circuit area has been minimized and the length of the wires used for its construction is also minimized.



Design

We will be using some constraints for the PCB to make it efficient and minimize it. The constraints are mentioned in the next slide. The clingo program will consider two layers of the PCB, on-board layer and the above-board layer. Some components like the wires stay only on the on-board layer while other components like the resistors stay only on the above-board layer. There are some components that overlap on both layers like transformers. This splitting allows us to implement certain designs which may help us reduce PCB size like wire jumpers. For simplicity we use only one pcb layer but in practice we can scale the no of PCB layers to any number using the same logic we use for the two layers.



Constraints

- Minimum area for the PCB designed
- Wires do not cross each other in the same layer
- Components do not overlap
- Short circuits are not allowed
- Wiring length needs to be minimized (secondary to area)
- Circuit conditions need to be satisfied
- Wires may pass under resistors/capacitors/inductors
- Power supply will be put on the edge (for practical ease)



Python code for input taking

```
# taking input from input.txt
with open('input2.txt') as f:
    lines = f.readlines()

c = 1
s = ""
l = []
no = 1

compDict = {}

for x in lines:
    string = x
    string = string.split(' ', 1)
    compDict[c] = string[0]
    nodes = [int(s) for s in x.split() if s.isdigit()]
    no = max(no, nodes[0])
    no = max(no, nodes[1])
    s = "comp("+str(c)+", "+str(nodes[0])+", "+str(nodes[1])+")."
    l.append(s)
    c = c+1
```



Python code for input taking

- The above python code will convert the circuit description involving the components and nodes into suitable input file for processing by clingo.
- For example the below input description will be converted into clingo input as follows:

Input circuit description:

```
r1 1 2  
r2 2 3  
d1 3 0  
v1 1 0
```

Input Clingo file:


```
comp(1,1,2).  
comp(2,2,3).  
comp(3,3,0).  
comp(4,1,0).
```



Python code for running clingo

```
# run clingo and generate answers
ctl = clingo.Control("")
n = int(input("What is n? -> "))
with open("data.lp", 'w') as f:
    for L in l:
        f.writelines(L)
        f.writelines("\n")
    f.write("#const n="+str(n)+".")
    f.writelines("\n")
    f.write("#const no="+str(no)+".")
ctl.load("data.lp")
ctl.load("project.lp")
ctl.configuration.solve.models = "0"
ctl.ground(["base", []])

with ctl.solve(on_model=lambda m: myfn(m), async_=True) as handle:
    while not handle.wait(0):
        pass
    handle.get()
```



Python function for reading and storing computed answer models

```
# function to extract stable models from clingo and store it
def myfn(ans):
    global ctr
    global h
    global w
    global wireLocations
    global compLocations
    ctr += 1
    print("Answer number: ", ctr, " is printed below")
    atoms = ans.symbols(atoms=True)
    print("Atoms are: ", atoms)
    w = 0
    h = 0
    compLocations = []
    wireLocations = []
```



Python function for reading and storing computed answer models

```
for atom in atoms:
    if(atom.name == "place"):
        s = str(atom.arguments[3])
        w = max(w, int(s)-1)
        s = str(atom.arguments[4])
        h = max(w, int(s)-1)
        comp = []
        for arg in atom.arguments:
            comp.append(int(str(arg)))
        compLocations.append(comp)

    elif(atom.name == "wplace"):
        s = str(atom.arguments[4])
        w = max(w, int(s)-1)
        s = str(atom.arguments[5])
        h = max(w, int(s)-1)
        wire = []
        for arg in atom.arguments:
            wire.append(int(str(arg)))
        wireLocations.append(wire)
    else:
        continue
```



Python function for displaying final PCB design using Tkinter library

```
# function to display the final PCB design
def pcblayout():
    global compLocations
    global wireLocations
    global compDict
    global w
    global h

    # changeable thickness of the cell
    t = 101

    myw = Tk()
    myw.title(" Effective PCB Design")
    myc = Canvas(myw, width=(w)*t, height=(h)*t, background='#008C4A')
    myc.grid(row=0, column=0)

    # placement of components
    for components in compLocations:
        stringTemp = str(compDict[components[0]])
        y = components[1]-1
        x = components[2]-1
        i = myc.create_text(x*t + (t+1)/2, y*t + (t+1)/2, fill="darkblue", font="Times 20 italic bold",
                           text=stringTemp)
        r = myc.create_rectangle(x*t + 0, y*t + 0, x*t + t,
                                y*t + t, fill="#bdd99e")
        myc.tag_lower(r, i)
```

Python function for displaying final PCB design using Tkinter library

```
# placement of wires
for wires in wireLocations:
    y = wires[2]-1
    x = wires[3]-1
    a = wires[0]
    b = wires[1]
    if (a == 1 and b == 2) or (a == 2 and b == 1):
        myc.create_line(x*t + (t+1)/2, y*t + (t+1)/2, x*t + (t+1)/2,
                        y*t + t, fill='#FFD700')
        myc.create_line(x*t + (t+1)/2, y*t + (t+1)/2, x*t +
                        t, y*t + (t+1)/2, fill='#FFD700')
    elif (a == 1 and b == 3) or (a == 3 and b == 1):
        myc.create_line(x*t + 0, y*t + (t+1)/2, x*t + t,
                        y*t + (t+1)/2, fill='#FFD700')
    elif (a == 1 and b == 4) or (a == 4 and b == 1):
        myc.create_line(x*t + (t+1)/2, y*t + 0, x*t + (t+1)/2,
                        y*t + (t+1)/2, fill='#FFD700')
        myc.create_line(x*t + (t+1)/2, y*t + (t+1)/2, x*t +
                        t, y*t + (t+1)/2, fill='#FFD700')
    elif (a == 2 and b == 3) or (a == 3 and b == 2):
        myc.create_line(x*t + 0, y*t + (t+1)/2, x*t + (t+1)/2,
                        y*t + (t+1)/2, fill='#FFD700')
        myc.create_line(x*t + (t+1)/2, y*t + (t+1)/2, x*t + (t+1)/2,
                        y*t + t, fill='#FFD700')
    elif (a == 2 and b == 4) or (a == 4 and b == 2):
        myc.create_line(x*t + (t+1)/2, y*t + 0, x*t + (t+1)/2,
                        y*t + t, fill='#FFD700')
    elif (a == 3 and b == 4) or (a == 4 and b == 3):
        myc.create_line(x*t + (t+1)/2, y*t + 0, x*t + (t+1)/2,
                        y*t + (t+1)/2, fill='#FFD700')
        myc.create_line(x*t + (t+1)/2, y*t + (t+1)/2, x*t +
                        0, y*t + (t+1)/2, fill='#FFD700')

myw.mainloop()
```


Clingo code

```
% Effective PCB Design
```

```
% placement of components everywhere
```

```
1{place(Name,1..n,1..n,1..n,1..n)}1 :- comp(Name,Con1,Con2).
```

```
:-place(X,A,B,C,D),(A+1,B+1)!=(C,D).
```

```
:-place(X1,A1,B1,C1,D1),place(X2,A2,B2,C2,D2),A1=A2,B1=B2,X1!=X2.
```

```
% placement of wires everywhere
```

```
2{wplace(1..4,1..4,1..n,1..n,1..n,1..n)}.
```

```
:-wplace(X,Y,A,B,C,D),(A+1,B+1)!=(C,D).
```

```
:-wplace(X,Y,A,B,C,D),X=Y.
```

```
:-wplace(A1,B1,X11,Y11,X21,Y21),wplace(A2,B2,X12,Y12,X22,Y22),X11=X12,Y11=Y12,X21=X22,Y21=Y22,B1!=B2.
```

```
:-wplace(A1,B1,X11,Y11,X21,Y21),wplace(A2,B2,X12,Y12,X22,Y22),X11=X12,Y11=Y12,X21=X22,Y21=Y22,A1!=A2.
```

```
:-wplace(A1,B1,X11,Y11,X21,Y21),place(A2,X12,Y12,X22,Y22),X11=X12,Y11=Y12,X21=X22,Y21=Y22.
```

```
% connect wires adjacently
```

```
:-wplace(A1,B1,X11,Y11,X21,Y21),wplace(A2,B2,X12,Y12,X22,Y22),X12=X11,Y12=Y11+1,X22=X21,Y22=Y21+1,A1=1,B2!=3.
```

```
:-wplace(A1,B1,X11,Y11,X21,Y21),wplace(A2,B2,X12,Y12,X22,Y22),X12=X11,Y12=Y11+1,X22=X21,Y22=Y21+1,A1!=1,B2=3.
```

```
:-wplace(A1,B1,X11,Y11,X21,Y21),wplace(A2,B2,X12,Y12,X22,Y22),X12=X11,Y12=Y11+1,X22=X21,Y22=Y21+1,B1=1,A2!=3.
```

```
:-wplace(A1,B1,X11,Y11,X21,Y21),wplace(A2,B2,X12,Y12,X22,Y22),X12=X11,Y12=Y11+1,X22=X21,Y22=Y21+1,B1!=1,A2=3.
```

```
:-wplace(A1,B1,X11,Y11,X21,Y21),wplace(A2,B2,X12,Y12,X22,Y22),X12=X11+1,Y12=Y11,X22=X21+1,Y22=Y21,A1=2,B2!=4.
```

```
:-wplace(A1,B1,X11,Y11,X21,Y21),wplace(A2,B2,X12,Y12,X22,Y22),X12=X11+1,Y12=Y11,X22=X21+1,Y22=Y21,A1!=2,B2=4.
```

```
:-wplace(A1,B1,X11,Y11,X21,Y21),wplace(A2,B2,X12,Y12,X22,Y22),X12=X11+1,Y12=Y11,X22=X21+1,Y22=Y21,B1=2,A2!=4.
```

```
:-wplace(A1,B1,X11,Y11,X21,Y21),wplace(A2,B2,X12,Y12,X22,Y22),X12=X11+1,Y12=Y11,X22=X21+1,Y22=Y21,B1!=2,A2=4.
```

Clingo code

% every component should be connected by 2 adjacent wires

```
:- place(X,A,B,C,D), not 2{wplace(4,_,A+1,B,C+1,D); wplace(3,_,A,B+1,C,D+1); wplace(2,_,A-1,B,C-1,D); wplace(1,_,A,B-1,C,D-1);  
wplace(_,4,A+1,B,C+1,D); wplace(_,3,A,B+1,C,D+1); wplace(_,2,A-1,B,C-1,D); wplace(_,1,A,B-1,C,D-1)}2.
```

% empty cells marked to eliminate models that have open loops

```
empty(A,B,C,D) :- not place(_,A,B,C,D), not wplace(_,_,A,B,C,D), A=0..n+1, B=0..n+1, C=0..n+1, D=0..n+1, C=A+1, D=B+1.
```

```
:- wplace(X,_,A,B,C,D), X=1, empty(A,B+1,C,D+1).
```

```
:- wplace(X,_,A,B,C,D), X=2, empty(A+1,B,C+1,D).
```

```
:- wplace(X,_,A,B,C,D), X=3, empty(A,B-1,C,D-1).
```

```
:- wplace(X,_,A,B,C,D), X=4, empty(A-1,B,C-1,D).
```

```
:- wplace(_,Y,A,B,C,D), Y=1, empty(A,B+1,C,D+1).
```

```
:- wplace(_,Y,A,B,C,D), Y=2, empty(A+1,B,C+1,D).
```

```
:- wplace(_,Y,A,B,C,D), Y=3, empty(A,B-1,C,D-1).
```

```
:- wplace(_,Y,A,B,C,D), Y=4, empty(A-1,B,C-1,D).
```

% circuit correctness

% same wire components have same node

```
1{node(X,Y,A,B,C,D,1..no)}1 :- wplace(X,Y,A,B,C,D).
```

```
:- node(A1,B1,X11,Y11,X21,Y21,N1), node(A2,B2,X12,Y12,X22,Y22,N2), X12=X11, Y12=Y11+1, X22=X21, Y22=Y21+1, A1=1, B2=3, N1!=N2.
```

```
:- node(A1,B1,X11,Y11,X21,Y21,N1), node(A2,B2,X12,Y12,X22,Y22,N2), X12=X11, Y12=Y11+1, X22=X21, Y22=Y21+1, B1=1, A2=3, N1!=N2.
```

```
:- node(A1,B1,X11,Y11,X21,Y21,N1), node(A2,B2,X12,Y12,X22,Y22,N2), X12=X11+1, Y12=Y11, X22=X21+1, Y22=Y21, A1=2, B2=4, N1!=N2.
```

```
:- node(A1,B1,X11,Y11,X21,Y21,N1), node(A2,B2,X12,Y12,X22,Y22,N2), X12=X11+1, Y12=Y11, X22=X21+1, Y22=Y21, B1=2, A2=4, N1!=N2.
```

Clingo code

```
% comp are connected to correct noded wires
:- #count{N : node(,,_,_,N)} != no.
:-node(A1,B1,X11,Y11,X21,Y21,N1),place(Name,X12,Y12,X22,Y22),comp(Name,Con1,Con2),X12=X11,Y12=Y11+1,X22=X21,Y22=Y21+1,A1=1,N1!=Con1,N1!=Con2.
:-node(A1,B1,X11,Y11,X21,Y21,N1),place(Name,X12,Y12,X22,Y22),comp(Name,Con1,Con2),X12=X11,Y12=Y11+1,X22=X21,Y22=Y21+1,B1=1,N1!=Con1,N1!=Con2.

:-node(A1,B1,X11,Y11,X21,Y21,N1),place(Name,X12,Y12,X22,Y22),comp(Name,Con1,Con2),X12=X11,Y12=Y11-1,X22=X21,Y22=Y21-1,A1=3,N1!=Con1,N1!=Con2.
:-node(A1,B1,X11,Y11,X21,Y21,N1),place(Name,X12,Y12,X22,Y22),comp(Name,Con1,Con2),X12=X11,Y12=Y11-1,X22=X21,Y22=Y21-1,B1=3,N1!=Con1,N1!=Con2.

:-node(A1,B1,X11,Y11,X21,Y21,N1),place(Name,X12,Y12,X22,Y22),comp(Name,Con1,Con2),X12=X11+1,Y12=Y11,X22=X21+1,Y22=Y21,A1=2,N1!=Con1,N1!=Con2.
:-node(A1,B1,X11,Y11,X21,Y21,N1),place(Name,X12,Y12,X22,Y22),comp(Name,Con1,Con2),X12=X11+1,Y12=Y11,X22=X21+1,Y22=Y21,B1=2,N1!=Con1,N1!=Con2.

:-node(A1,B1,X11,Y11,X21,Y21,N1),place(Name,X12,Y12,X22,Y22),comp(Name,Con1,Con2),X12=X11-1,Y12=Y11,X22=X21-1,Y22=Y21,A1=4,N1!=Con1,N1!=Con2.
:-node(A1,B1,X11,Y11,X21,Y21,N1),place(Name,X12,Y12,X22,Y22),comp(Name,Con1,Con2),X12=X11-1,Y12=Y11,X22=X21-1,Y22=Y21,B1=4,N1!=Con1,N1!=Con2.

% minimization of component location
#minimize{(A+B) : place(X,A,B,_,_)}.

% minimization of length of wires
#minimize{1,A,B,C,D : wplace(,,_,A,B,C,D)}.

% minimization of wire location
#minimize{(A+B+C+D) : wplace(,,_,A,B,C,D)}.

% show required output
#show place/5.
#show wplace/6.
#show node/7.
```



Explanation of Clingo code

- The components are described by the rule ***comp***(*Name*, *Node1*, *Node2*).
- The rule ***place*** is for deciding the placement of the different components.
- We have written constraints so that the different components do not overlap with each other.
- The rule ***wplace*** is for deciding the placement of the wires connecting the different components.
- We have written constraints so that the wires do not overlap with components or other wires.
- We have written constraints for proper connection of wires to components and other wires.
- The rule ***empty*** decides whether a 1x1 block is empty or not.
- The rule ***node*** assigns a number for all wires connected to each other and to components representing the node to which it is connected.
- Finally we have added a code to minimize the sum of coordinates of the components effectively minimizing area, and also to minimize the length of the wires.

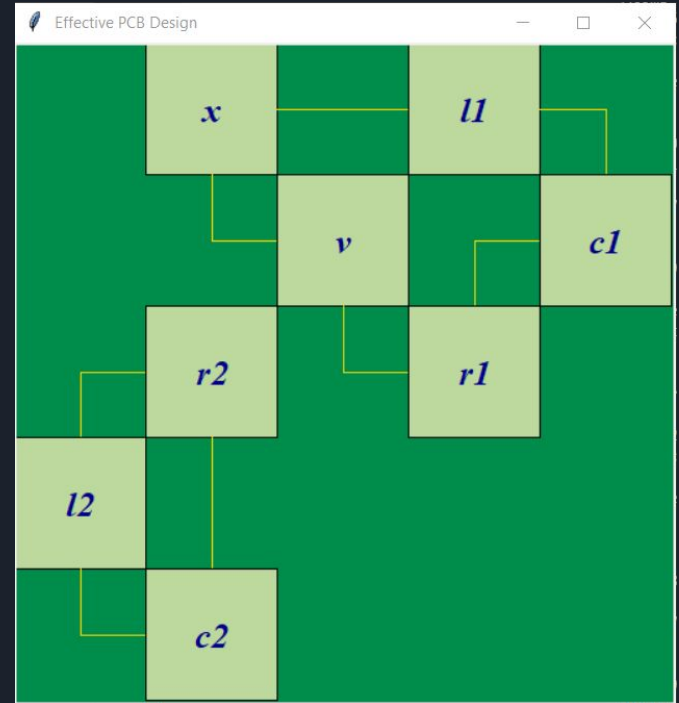
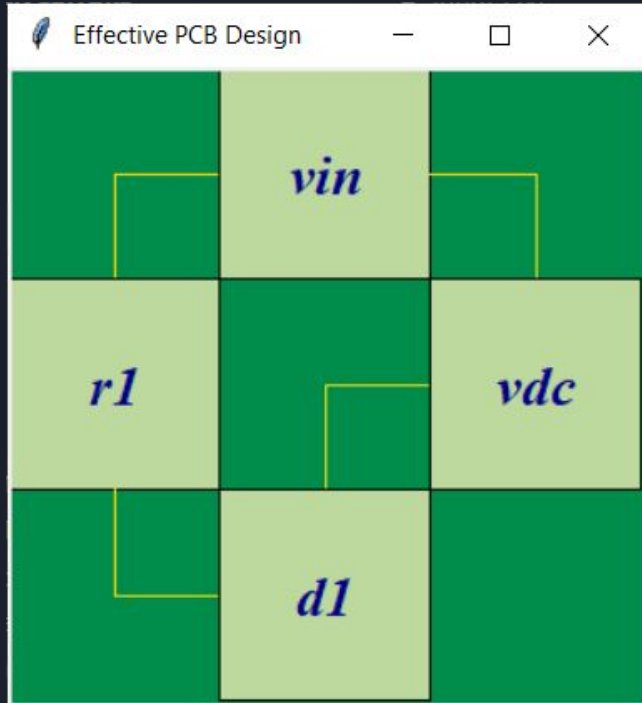


Clingo output

Answer number: 10 is printed below

```
Atoms are: [place(5,1,4,2,5), place(4,2,3,3,4), place(3,3,2,4,3), place(6,3,4,4,5), place(2,4,1,5,2), place(7,4,3,5,4), place(1,5,2,6,3), comp(1,1,2), comp(2,2,3), comp(3,3,4), comp(4,4,5), comp(5,5,6), comp(6,6,7), comp(7,7,1), wplace(1,2,1,3,2,4), wplace(1,2,3,1,4,2), wplace(1,2,4,2,5,3), wplace(1,4,5,1,6,2), wplace(2,1,2,2,3,3), wplace(2,1,3,3,4,4), wplace(4,2,2,4,3,5), empty(0,0,1,1), empty(0,1,1,2), empty(0,2,1,3), empty(0,3,1,4), empty(0,4,1,5), empty(0,5,1,6), empty(0,6,1,7), empty(0,7,1,8), empty(0,8,1,9), empty(0,9,1,10), empty(0,10,1,11), empty(1,0,2,1), empty(1,1,2,2), empty(1,2,2,3), empty(1,5,2,6), empty(1,6,2,7), empty(1,7,2,8), empty(1,8,2,9), empty(1,9,2,10), empty(1,10,2,11), empty(2,0,3,1), empty(2,1,3,2), empty(2,5,3,6), empty(2,6,3,7), empty(2,7,3,8), empty(2,8,3,9), empty(2,9,3,10), empty(2,10,3,11), empty(3,0,4,1), empty(3,5,4,6), empty(3,6,4,7), empty(3,7,4,8), empty(3,8,4,9), empty(3,9,4,10), empty(3,10,4,11), empty(4,0,5,1), empty(4,4,5,5), empty(4,5,5,6), empty(4,6,5,7), empty(4,7,5,8), empty(4,8,5,9), empty(4,9,5,10), empty(4,10,5,11), empty(5,0,6,1), empty(5,3,6,4), empty(5,4,6,5), empty(5,5,6,6), empty(5,6,6,7), empty(5,7,6,8), empty(5,8,6,9), empty(5,9,6,10), empty(5,10,6,11), empty(6,0,7,1), empty(6,1,7,2), empty(6,2,7,3), empty(6,3,7,4), empty(6,4,7,5), empty(6,5,7,6), empty(6,6,7,7), empty(6,7,7,8), empty(6,8,7,9), empty(6,9,7,10), empty(6,10,7,11), empty(7,0,8,1), empty(7,1,8,2), empty(7,2,8,3), empty(7,3,8,4), empty(7,4,8,5), empty(7,5,8,6), empty(7,6,8,7), empty(7,7,8,8), empty(7,8,8,9), empty(7,9,8,10), empty(7,10,8,11), empty(8,0,9,1), empty(8,1,9,2), empty(8,2,9,3), empty(8,3,9,4), empty(8,4,9,5), empty(8,5,9,6), empty(8,6,9,7), empty(8,7,9,8), empty(8,8,9,9), empty(8,9,9,10), empty(8,10,9,11), empty(9,0,10,1), empty(9,1,10,2), empty(9,2,10,3), empty(9,3,10,4), empty(9,4,10,5), empty(9,5,10,6), empty(9,6,10,7), empty(9,7,10,8), empty(9,8,10,9), empty(9,9,10,10), empty(9,10,10,11), empty(10,0,11,1), empty(10,1,11,2), empty(10,2,11,3), empty(10,3,11,4), empty(10,4,11,5), empty(10,5,11,6), empty(10,6,11,7), empty(10,7,11,8), empty(10,8,11,9), empty(10,9,11,10), empty(10,10,11,11), node(1,2,4,2,5,3,1), node(1,4,5,1,6,2,2), node(1,2,3,1,4,2,3), node(2,1,2,2,3,3,4), node(1,2,1,3,2,4,5), node(4,2,2,4,3,5,6), node(2,1,3,3,4,4,7)]
```

Python Tkinter output





Final Results

- We have successfully completed our objective of Effective PCB Design as part of our CS449 (Topics in AI) project.
- We have finished the clingo code for optimization of area + minimization of length of the wires of the PCB Design with 1x1 components.
- We have used Python Tkinter library to print the final PCB design.



Further inclusions that can be done

- We can include the condition of branching of wires. Now our code optimizes only single loop multiple circuits. With branching of wires integrated with the code, circuits with multiple loops can be processed.
- Our code can be expanded to consider bigger area components.



Thank You!

