Quick python script for rearranging components in pcb file

April 26, 2013

1 What's the big deal?

KiCAD is an open source software for designing boards. Within the "Eeschema" editor you can create components in sheets and subsheets, which are connected to the outside. Once you load those into the PcB-editor, the problem is that those components are placed without any reference to the original subsheet. This can be quite some hassle if you are working with lots of components in several subsheets, which you would like to place together on the board.

Note that the following python-script is just a quick-and-dirty solution that fit to my needs and is far-off from being elaborate. Nevertheless, it might give anybody trying to script something similar a first idea of a possible approach):

```
import re
import math


#-------------------------------------------------------------------------------
#1 read in the data-file and get all the relevant information


#this module will summarize all the necessary information
class module_list:
        def __init__(self, path, module, coord_line):
                self.path = path
                self.module = module
                self.coord_line = coord_line
        def __repr__(self):
                return repr((self.path, self.module, self.coord_line))



module_items=[]     #here all information is listed
file_content=[]     #this is a list of all lines of the pcb-file document
```

```
22
23  module_indicator="path" #this is the key-word that will bring us down to each mo
24  m_dum=-1;           #just a mere counter
25
26  #here we open the pcb-file and search for the "path"-component; this should be u
27  #coordinates on the board we ultimately wish to change and 4 lines above is the
28  with open("Board_File.kicad_pcb") as file_pcb:
29          for line in file_pcb:
30              file_content.append(line)
31              m_dum=m_dum+1
32              matchObj = re.search( module_indicator, line, re.M|re.I)        #http
33              if matchObj:
34                  path_var=file_content[m_dum][11:19]     #get the path name; a bit
35                  module_var = re.search("module (\w+)", file_content[m_dum-4], re.M
36                  coord_line=m_dum-3                      #show where the coordinat
37                  module_items.append(module_list(path_var,module_var,coord_line))
38
39
40  #now the name attributes are sorted with respect to module_var http://wiki.pytho
41  module_items_sorted=sorted(module_items, key=lambda module_list: module_list.path)
42
43  #now search for the same modules created in the same subsheet;
44  module_identifier=[0]        #this variable is saving a crossing from one submodu
45  submodule_ref=module_items_sorted[1].path     #this variable is taken as initiali
46  module_length=[]    #length of the corresponding module of module_identifier
47  module_length_ref=0
48  m_dum=-1; #again another counter
49
50  for m_item in module_items_sorted:
51      m_dum=m_dum+1
52      module_var=m_item.path
53      if submodule_ref!=module_var:
54          module_identifier.append(m_dum)
55          module_length.append(m_dum-module_length_ref)
56          submodule_ref=module_var  #change the reference
57          module_length_ref=m_dum
58
59  module_length.append(m_dum-module_length_ref) #last one has to be appended as wel
```

```
60
61
62  #-------------------------------------------------------------------------
63  #2 calculate the grid on the board for the subregions
64
65
66  #divide a predefined board area into an equal amount of subregions
67  #in our case the coordinates are (90 60) (90 200) (360 200) (360 60)
68  # ideally the information of the shape of the modules should be considered!!!
69  #Moreover, some design rules could also be integrated (though they become more i
70
71  #solve the following equation to get the minimum number of rectangles fitting in
72  #y_sub*n=140=y_total
73  #x_sub*m=270=x_total
74  #n*m=number of subregions = 18
75  #x should be = y
76  #=> 4 equations with 4 unknowns
77
78  x_total=270
79  y_total=140
80  num_of_subregions=len(module_identifier)
81
82  m_tot=math.ceil(math.sqrt(x_total*num_of_subregions/y_total))
83  n_tot=math.ceil(num_of_subregions/math.ceil(m_tot))
84
85  y_sub=y_total/n_tot
86  x_sub=x_total/m_tot
87
88
89  #next get the number of components in a submodule, take a square-like area of su
90  x_0=90      #starting point in the coordinate system
91  y_0=60      #starting point in the coordinate system on y axis
92  m_count=-1     #this is the counter for getting the right coordinates in module_
93
94  for m_dum in range(len(module_identifier)):
95      x_middle=x_0+x_sub*(0.5+(m_dum % m_tot))                #get the middle of x-
96      y_middle=y_0+y_sub*(0.5+math.floor((m_dum)/(m_tot)))    #get the middle of y-
97
```

```
98      for m_dum2 in range(module_length[m_dum]):
99          m_count=m_count+1
100         x_dum=x_middle+0.2*x_sub*math.cos(2*math.pi/(m_dum2+1)) #here arrange aro
101         y_dum=y_middle+0.2*y_sub*math.sin(2*math.pi/(m_dum2+1))
102
103         #precision is up to 2 digits creating the corresponding string for the f
104         x_rounded= "%.2f" % round(x_dum,2)
105         y_rounded= "%.2f" % round(y_dum,2)
106         new_coord="    (at "+ x_rounded + " " + y_rounded +")\n"   #python will c
107
108         #now write the new coordinates into the respective line in the file:
109         coord_line=module_items_sorted[m_count].coord_line
110         file_content[coord_line]=new_coord
111
112
113 #now write everything into a new output file:
114 file_new = open('Board_File.kicad_pcb','w')
115 for item in file_content:
116     file_new.write(item)
117 file_new.close()
```