1 mainpatch.py

```
1  def dload(fname):
2      #open DUMP_FILE
3      f = open(fname)
4
5      data = []
6
7      #add the data to data, line by line
8      for line in f:
9          data.append(line)
10
11     f.close()
12
13     import string
14
15     #split the data up by the | character
16     data = map(lambda x: string.split(x, "|"), data[:])
17     #turn all the strings into floats
18     data = map(lambda x: map(lambda y: float(y), x), data[:])
19
20     return data
21
22 def powercalc(line):
23     """assuming balanced circuit, find power consumption"""
24     #my excuse for it being this verbose is to make it easily readable
25     Ra = line[0]
26     Rb = line[1]
27     Rc = line[2]
28     Rd = line[3]
29     Ve = line[4]
30     Rf = line[5]
31     Ry = 2.0
32     #Using thevenins theorem, Vy is the point next to Rb
33     Vthy = (Ve*(Rd+Rf))/(Rd+Rc+Rf)
34     Rthy = Ry + (Rf*(Rd+Rc))/(Rf+Rd+Rc)
35     Ithy = Vthy/(Rthy+Rb)
36     Vy = (Rb*Vthy)/(Rthy+Rb)
37     Pb = Vy*Ithy
38     #Vz is the point next to Rc
39     Vthz = (Ve*(Ry+Rb))/(Rf+Ry+Rb)
40     Rthz = (Rf*(Ry+Rb))/(Rf+Ry+Rb) + Rd
41     Ithz = Ve/(Rthz + Rc)
42     Vz = (Vthz*Rc)/(Rthz+Rc)
43     Pc = Ithz*Vz
44     #Vx is the point next to Rf
45     Rbridge = ((Ry+Rb)*(Rd+Rc))/(Ry+Rb+Rd+Rc)
46     Vx = (Ve*Rbridge)/(Rbridge+Rf)
47     Pd = ((Vx-Vz)**2)/Rd
```

```
48      Pf = ((Ve–Vx)**2)/Rf
49      Rtot = Rf + ((Ry+Rb)*(Rd+Rc))/(Ry+Rb+Rd+Rc)
50      power = (Ve**2)/Rtot
51      return [power,Pb, Pc, Pd, Pf]
52
53  def iterator():
54      #monte carlo iterator to run the test many times with random values
55      import random
56      import math
57      import time
58      import pickle
59
60      data = dload("DUMP_FILE")
61      #infer what Rc should be on each line
62      data2= []
63      for line in data:
64          Rc = (line[2]*line[1])/2
65          data2.append(line[0:2]+[Rc]+line[2:])
66
67      #modulate with random variable
68      #query before doing this, so test can also work without this
69      #good old infinite loop
70      while 1:
71          #user query
72          uq = raw_input("Run Monte Carlo(mc) or nominal(n):\n")
73          if uq == "mc":
74              #query for resistor tolerance
75              tol = float(raw_input("Resistor Tolerance(in %):"))/100
76              psv = float(raw_input("Power supply standard deviation(in %):"))/100
77              #run Monte Carlo
78              datalist = []
79              for i in xrange(1000):
80                  print "iteration %s/1000"%(i+1)
81                  #modulate all resistor values and power supply as Gaussian random
                        variables
82                  mcdata = []
83                  for line in data2:
84                      #new line
85                      nl=[]
86                      for R in line[:4]:
87                          nl.append(random.gauss(R, tol*R/5.9))
88                      nl.append(random.gauss(line[4],line[4]*psv))
89                      nl.append(random.gauss(line[5], tol*line[5]/5.9))
90                      mcdata.append(nl)
91                  datalist.append(begin(mcdata))
92              print "Monte Carlo complete"
93              print "Processing..."
94              #use the output data to calculate variations in output variables
95              mcout = []
```

```
96              #the data is a list of list of lists at this point
97              for line in zip(*datalist):
98                  #so unpack it into lists and zip each row into a tuple of lists
99                  #then unpack the tuple into lists and then zip these lists into
                        tuples
100                 #so now each tuple in the data is each output of the Monte Carlo
101                 linz = zip(*line)
102                 avgs = []
103                 #calculate average of each tuple and add it to a list
104                 for x in linz:
105                     avgs.append(sum(x)/len(x))
106                 #do the same with standard deviations
107                 sigmas = []
108                 for x, avg in zip(linz, avgs):
109                     s = map(lambda y: (y-avg)**2, x)
110                     sigmas.append(math.sqrt(sum(s)/len(s)))
111                 #estimate yield from both dys
112                 ylds = []
113                 #make a list of the averages and sigmas in tuples, then throw away
                        all but the dys
114                 for x in zip(avgs, sigmas)[:2]:
115                     ylds.append(x[0]/x[1])
116                 #smallest yield is the worst yield
117                 yld = min(ylds)
118                 #estimate worst case power consumption
119                 pwc = avgs[2]+6*sigmas[2]
120                 #esimate worst case accuracy
121                 dywcs = []
122                 for x in zip(avgs, sigmas)[:2]:
123                     dywcs.append(x[0]+6*x[1])
124                 dywc = max(dywcs)
125                 #estimate worst case power consumption for each resistor
126                 Rwcs = []
127                 for x in zip(avgs,sigmas)[2:]:
128                     Rwcs.append(x[0]+6*x[1])
129                 #append results to list
130                 mcout.append([dywc,pwc,yld]+Rwcs)
131
132             #write results to output file
133             f = open("LOAD_FILE", "w")
134             for x in mcout:
135                 y = range(len(x))
136                 y.reverse()
137                 for z in zip(x,y):
138                     if z[1]==0:
139                         f.write("%f\n"%z[0])
140                     else:
141                         f.write("%f|"%z[0])
142                 #f.write("%f|%f|%f\n"%(x[0],x[1],x[2]))
```

```
143             f.close()
144             #combine input and output data
145             pdata = map(lambda x: x[0]+x[1], zip(data2,mcout))
146             #pickle results
147             #open pickle file
148             try:
149                 #open existing dictionary
150                 f = open("dblog.pickle")
151                 #and read it
152                 pdict = pickle.load(f)
153                 f.close()
154             except IOError:
155                 #initialise dictionary
156                 pdict = {}
157             #add to pickled dictionary
158             pdict[time.ctime(time.time())] = pdata
159             #rewrite pickle file
160             f = open("dblog.pickle", "wb")
161             pickle.dump(pdict, f)
162             f.close()
163             print "Processing complete"
164             break
165         elif uq == "n":
166             #run nominal
167             odata = begin(data2)
168             #probably just change this bit to a simple for loop
169             #odata = map(lambda x: [max(x[:2]),x[2:]], odata[:])
170             odata2 = []
171             for line in odata[:]:
172                 l2 = [max(line[:2])]
173                 l2.extend(line[2:])
174                 odata2.append(l2)
175             #write results to output file
176             f = open("LOAD_FILE", "w")
177             for x in odata2:
178                 y = range(len(x))
179                 y.reverse()
180                 for z in zip(x,y):
181                     if z[1]==0:
182                         f.write("%f\n"%z[0])
183                     else:
184                         f.write("%f|"%z[0])
185             f.close()
186             #pickle results
187             #combine output and input data
188             pdata = [] #I could do this with a map, but for some reason did not
189             for x in zip(data2,odata2):
190                 pdata.append(x[0]+x[1])
191             #open pickle file
```

```
192             try:
193                 #open existing dictionary
194                 f = open("dblog.pickle")
195                 #and read it
196                 pdict = pickle.load(f)
197                 f.close()
198             except IOError:
199                 #initialise dictionary
200                 pdict = {}
201             #add to pickled dictionary
202             pdict[time.ctime(time.time())] = pdata
203             #rewrite pickle file
204             f = open("dblog.pickle", "wb")
205             pickle.dump(pdict, f)
206             f.close()
207             break
208         elif uq == "\n":
209             break
210     return None
211
212 def begin(data):
213     #this code is terrible
214     import os
215     import pdb
216
217     data2= []
218     for line in data:
219         data2.append(line + [0.0002])
220
221     #add unbalanced current
222
223     f = open("DUMP_FILE_PATCH", "w")
224
225     for line in data2:
226         for x in line[:]:
227             if x == line[-1]:
228                 f.write("%f"%x)
229             else:
230                 f.write("%f|"%x)
231         f.write("\n")
232             #f.write("%f|%f|%f|%f|%f|%f|%f\n"%(line[0], line[1], line[2], line[3],
                    line[4], line[5], line[6]))
233
234     f.close()
235
236     #execute c code on command line
237     os.system("./bridge")
238
239     #read in results
```

```
240     hy = dload("LOAD_FILE")
241     dy1 = map(lambda x: 2.0-x[0], hy)
242
243     data2= []
244     for line in data:
245         Rc = (line[2]*line[1])/2
246         data2.append(line + [-0.0002])
247
248     f = open("DUMP_FILE_PATCH", "w")
249
250     for line in data2:
251         for x in line[:]:
252             if x == line[-1]:
253                 f.write("%f"%x)
254             else:
255                 f.write("%f|"%x)
256         f.write("\n")
257             #f.write("%f|%f|%f|%f|%f|%f|%f\n"%(line[0], line[1], line[2], line[3],
                    line[4], line[5], line[6]))
258
259     f.close()
260
261     #execute c code on command line
262     os.system("./bridge")
263
264     #read in results
265     ly = dload("LOAD_FILE")
266
267     dy2 = map(lambda x: x[0]-2.0, ly)
268
269     #dy = map(lambda x: (x[1][0]-x[0][0])/2, zip(hy,ly))
270
271     #dy=[]
272     #for x in zip(dy1,dy2):
273     #    dy.append(max(x))
274     #    if x[0] < 0:
275     #        print "Warning, circuit no longer detects 2 Ohm resistors"
276     #    else if x[1] < 0:
277     #        print "Warning, circuit no longer detects 2 Ohm resistors"
278
279
280     plist = []
281     #calculate power consumption
282     for line in data2:
283         plist.append(powercalc(line))
284
285     odata = zip(dy1, dy2, *zip(*plist))
286
287     #f = open("LOAD_FILE", "w")
```

```
288
289     #for x in odata:
290     #     f.write("%f|%f\n"%(x[0],x[1]))
291
292     #f.close()
293
294     return odata
295
296 if __name__ == "__main__":
297     iterator()
```

2 confirm.py

```
def crun(cfactors):
    """runs the c code on lists of input values, returns the resulting Ry"""
    import os
    import string

    f = open("DUMP_FILE_PATCH", "w")
    #write the list to a file in the expected way
    for line in cfactors:
        lnth = range(len(line))
        lnth.reverse()
        for x, l in zip(line,lnth):
            if l == 0:
                f.write("%f\n"%x)
            else:
                f.write("%f|"%x)
    f.close()

    #run the c code
    os.system("./bridge")

    #read the results from the other file
    f = open("LOAD_FILE")

    c = []
    for line in f:
        c.append(line)
    c = map(lambda x: string.split(x), c[:])

    d = map(lambda x: map(lambda y: float(y), x), c)
    d = map(lambda x: x[0], d[:])

    f.close()
    #return the resulting values
    return d

def cgraph(filename):
    """creates a csv that can be graphed easily from a csv of input parameters"""
    #open up and read csv of designs to test
    import mainpatch
    import string
    #cfactors = mainpatch.dload(filename)
    f = open(filename)
    cfactors = []
    for line in f:
        cfactors.append(map(lambda x: float(x), string.split(line)))
    f.close()
    cf2= []
```

```
48      for line in cfactors:
49          Rc = (line[2]*line[1])/2
50          cf2.append(line[0:2]+[Rc]+line[2:])
51      #1000 points between plus and minus 0.2mA in a list
52      pnts = 400
53      oobc = map(lambda x: -0.002 + (0.004*x)/pnts, range(pnts))
54      print oobc
55      rd = []
56      #for every row create a list of lists of control factors to evaluate
57      for line in cf2:
58          cf3=[]
59          for c in oobc:
60              cf3.append(line + [c])
61          #evaluate all currents
62          out = crun(cf3)
63          #reset values between plus and minus 0.2 mA to zero
64          #slice up based on indexes of oobc
65          oobci = map(lambda x: int(round(x*1000000)), oobc)
66          #I really shouldn't be allowed to write code
67          out = out[:oobci.index(-200)] + map(lambda x: 2, out[oobci.index(-200):
                oobci.index(200)+1]) + out[oobci.index(200):]
68          #then save the results in a list
69          rd.append(out)
70      #then write the lists to a file
71      f = open("rgraphs.csv", "w")
72      for line in zip(oobc,*rd):
73          ln = range(len(line))
74          ln.reverse()
75          for x in zip(line, ln):
76              if x[1] == 0:
77                  f.write("%f\n"%x[0])
78              else:
79                  f.write("%f,"%x[0])
80      f.close()
81      return None
82
83  if __name__ == "__main__":
84      while 1:
85          q = raw_input("evaluate confirmation run(cr) or csv of runs(csv)?")
86          if q == "csv":
87              #query for file name
88              filename = raw_input("file name:")
89              cgraph(filename)
90              break
91          elif q == "cr":
92              import string
93              #query for circuit input values
94              icfactors = map(lambda x: float(x), string.split(raw_input("Ra,Rb,Rc,
                    Rd,Ve,Vf:"), ","))
```

```
95              cfactors = []
96              cfactors.append(icfactors + [-0.0002])
97              cfactors.append(icfactors + [0])
98              cfactors.append(icfactors + [0.0002])
99              #send this to crun
100             out = crun(cfactors)
101             #calculate delta y
102             mean = out[1]
103             dy1 = out[0]-mean
104             dy2 = mean-out[2]
105             print "circuit balanced at %s Ohms"%(mean)
106             print "worst case delta y: %s"%(max([dy1,dy2]))
107             break
```