<u>Design of a Wireless Power Transfer System for a Pacemaker</u>

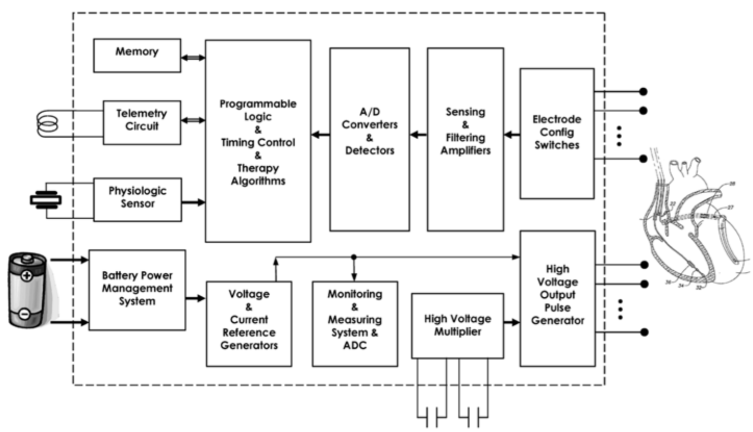
Lauren Adachi

In this project, a python script for designing a wireless power transfer system was created. Then, this script was used to design an inductive link for a pacemaker.

Pacemaker Wireless Power Transfer

I designed a wireless power transfer system for the pacemaker integrated circuit (IC) in [1] and [2]. Below (Figure 1) is a high-level block diagram of the pacemaker IC. The pacemaker consists of this IC, a pulse generator, and electrode pacing leads to provide electrical stimulation at a consistent frequency to set an appropriate pace for a human heart. This electrical stimulation consists of high voltage pulses (5V) that are generate via a high voltage generator (capacitive voltage multiplier).

Figure 1. Block Diagram of Pacemaker Integrated Circuit. [1]



Typical pacemakers are less than 1 oz in weight and less than 2 inches wide [1]. Typical operating frequencies for wireless power transfer include 125kHz for small coils (size of a pet's implanted RFID tag) and 13 MHz for larger coils (size of a credit card). The IC chosen requires 2.4 uW of power and 2.8 V [1]. The load impedance is therefore $Z_L = V_L/I_L$ = 2.8V ^ 2 / 8 uW = 980K Ohms.

Therefore, the design constraints are:

Overall size: < 1 inches

Load power: 2.4 uW

Load voltage: 2.8 V

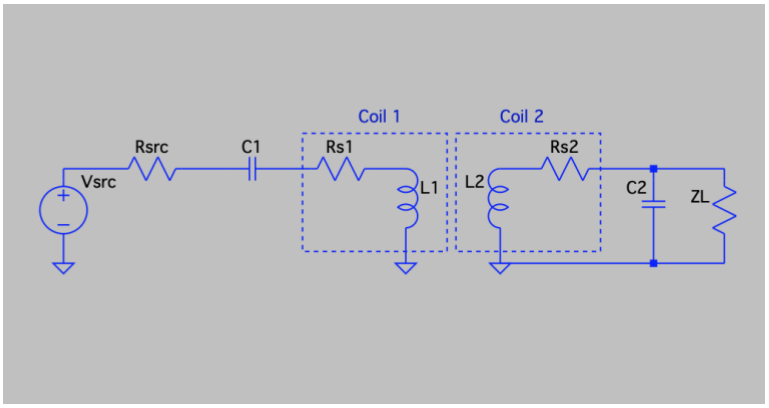
Load impedance (Z_L): 980K Ohms

Resonant frequency (f): 13 MHz

Inductive Link Model

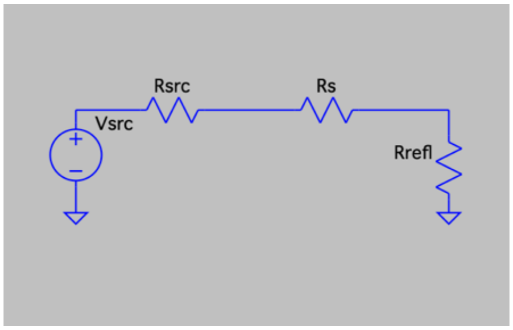
The inductive link is modelled as depicted in Figure 2. The model includes a voltage source (V_{src}) with corresponding source resistance (R_{src}). The coils are each modelled as an inductor (L) in series with a resistor (Rs). The load is modelled as an impedance (ZL). Lastly, resonant capacitors (C1, C2) are added to enable maximum power transfer via impedance matching.

Figure 2. Inductive link model.



The inductive link, at resonance (at resonant frequency f), is depicted in Figure 3. At resonance, the capacitors become shorted out, and the coils and load are represented by the equivalent "reflected" resistance through the coils (R_{refl}).

Figure 3. Simplified inductive link model at resonance.



Script

The script specifically calculates the coil parameters for a PCB spiral inductor.

The script takes coil, link, and requirement input parameters and determines potential coil designs by evaluating the Power Transfer Efficiency (PTE) and Power Delivered to the Load (PDE) of the power transfer system.

The input parameters for the script include coil parameters such as inner diameter of coil (din), copper track width (w), and track spacing (s) Figure 4 depicts these parameters graphically. Additionally, there are link parameters such as distance of link (D) desired resonant frequency for power transfer (f), load resistance ($R_L = Z_L$), source voltage (V_{src}), and source resistance (R_{src}) and requirements such as load power required (P_{Lreq}) and maximum diameter of coil (d_{max})

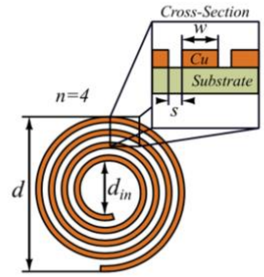


Figure 4. PCB spiral dimensions [3]

Script Parameters and Results for Pacemaker System

For the pacemaker system, my input parameters were as follows:

```
d_in1 = din_2 = 0.6 *10**(-3) # inner diameter, m

w1 = w2 = 0.6 *10**(-3)# track width, m

s1 = s2 = 2*w1 # track spacing, m



P_L_req = 2.4 * 10**(-6) # power required, W

d_max = 2 * 10**(-2) # maximum diameter of coil, m



D = 8 *10**(-3)# distance of link, m

f = 13*10**6 # desired resonant frequency, Hz

R_L = 980*10**3 # Load resistance, Ohms

V_src = 5 # from EM4095 driver output voltage, Volts

R_src = 7 # from EM4095 driver output resistance, Ohms
```

The values for V_{src} and R_{src} were taken from the EM4095 driver datasheet [5]. The equation for s1 came from [4].

The script produced:

Possible values of n1 and n2 for given input constraints...

n1 = 2, n2 = 4: d1(cm) = 0.72, d2(cm)= 1.44, PTE = 2.2466e-06, PDL(uW)= 3.29

n1 = 2, n2 = 5: d1(cm) = 0.72, d2(cm)= 1.8, PTE = 6.5281e-06, PDL(uW)= 9.3

n1 = 3, n2 = 4: d1(cm) = 1.08, d2(cm)= 1.44, PTE = 5.7183e-06, PDL(uW)= 6.26

n1 = 3, n2 = 5: d1(cm) = 1.08, d2(cm)= 1.8, PTE = 1.59016e-05, PDL(uW)= 16.27

n1 = 4, n2 = 4: d1(cm) = 1.44, d2(cm)= 1.44, PTE = 9.3413e-06, PDL(uW)= 7.03

n1 = 4, n2 = 5: d1(cm) = 1.44, d2(cm)= 1.8, PTE = 2.50039e-05, PDL(uW)= 16.88

n1 = 5, n2 = 4: d1(cm) = 1.8, d2(cm)= 1.44, PTE = 1.22802e-05, PDL(uW)= 6.19

n1 = 5, n2 = 5: d1(cm) = 1.8, d2(cm)= 1.8, PTE = 3.19304e-05, PDL(uW)= 13.91

To maximize PTE and PDL, I selected the final link design for the pacemaker system to be…

n1 = 5, n2 = 5: d1(cm) = 1.8, d2(cm)= 1.8, PTE = 3.19304e-05, PDL(uW)= 13.91

Physical Coil Design

Lastly, the coils were designed in KiCAD using the tutorial in [7]. I placed a 2-pin connector in the schematic, drew a wire between the two pins and associated a footprint with the connector. I ran the Copper Spiral Script twice (once for a spiral on the front copper layer, CCW, and once for a spiral on the back copper layer, CW) and pasted the results into the kicad_pcb file body in Atom. I then opened the kicad_pcb file in pcbnew and highlighted each of the spirals and clicked to see their properties, and changed the net for each to the net connecting the two pins on the connector together. Then I connected the spirals together at the center with a via and connected each outer end of the spiral to the connector.

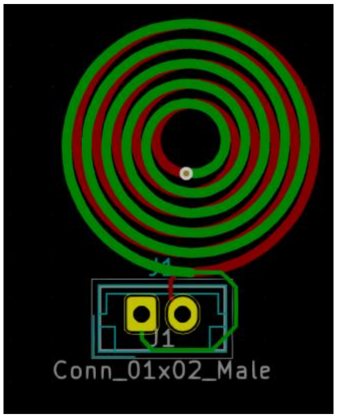


Figure 5. PCB Spiral Design in KiCAD

Conclusions

In summary, an adaptable, configurable python script was developed for designing inductive power transfer links. An inductive power transfer system was designed for a pacemaker system, and the designed PCB coils were created in KiCAD. It is important to note that this script is heavily dependent on the assumptions made in [3] and [8], so although this script serves as a good starting place for automating inductive power transfer design, further investigation is need to make sure it is accurate.

Appendix

Sources

[1] https://ieeexplore.ieee.org/document/1362855

[2] https://academic.oup.com/europace/article/16/10/1534/2426133

[3] https://ieeexplore.ieee.org/document/8410809/

[4] https://www.raypcb.com/pcb-coil/

[5] https://www.emmicroelectronic.com/sites/default/files/products/datasheets/4094-ds.pdf

[6] https://www.pcbuniverse.com/pcbu-tech-tips.php?a=4#:~:text=The%20most%20common%20unit%20of,1.37%20thousandths%20of%20an%20inch

[7] https://www.instructables.com/PCB-Coils-in-KiCad/

[8] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4449264/

Code

```
import scipy

import scipy.special

import os

import sys

import numpy as np

import matplotlib as mpl

import matplotlib.pyplot as plt


# CoilCalculator_v4

# this script takes in input parameters and power requirements and identifies

# viable coil parameters for an appropriate inductive link system.


if __name__ == '__main__' :

    # --- INPUT PARAMETERS

    # L1 parameters

    d_in1 = 0.6 *10**(-3) # inner diameter, m

    w1 = 0.6 *10**(-3)# track width, m

    s1 = 2*w1 # track spacing, m


    # L2 parameters
```

```
d_in2 = d_in1 # inner diameter, m
w2 = 0.6 *10**(-3)# track width, m
s2 = 2*w2 # track spacing, m

# constraints
P_L_req = 2.4 * 10**(-6) # power required, W
d_max = 2.54*1 * 10**(-2) # maximum diameter of coil, m
n_max = int((d_max - d_in1)/((s1+w1)*2))

# link parameters
D = 8 *10**(-3)# distance of link, m
f = 13*10**6 # desired resonant frequency, Hz
R_L = 980*10**3 # Load resistance, Ohms
V_src = 5 # from EM4095 driver output voltage, Volts
R_src = 7 # from EM4095 driver output resistance, Ohms

# define 2 arrays to hold PTE, and P_L values, respectively
PTE_array = [ [0]*n_max for i in range(n_max)]
P_L_array = [ [0]*n_max for i in range(n_max)]

# iterate through all possible n1 and n2 values
for n1 in range(1,n_max):
    for n2 in range (1,n_max):
        #print("n1:  " + str(n1))
        #print("n2 : " + str(n2))

        # ----- CALCULATIONS -----
        # calculate outer diameter of coil - derived via geometry
        d1 = d_in1 + (s1+w1)*n1*2
        #print("outer diameter of coil 1 (cm): " + str(d1*10**2))
        d2 = d_in2 + (s2+w2)*n2*2
        #print("outer diameter of coil 2 (cm): " + str(d2*10**2))

        # determine k, then find Kk, Ek, and Mk for that k -
        M = 0
        for i in range(0,n1):
```

```
    for j in range (0,n2):
        a = d_in1/2 + (s1+w1)*i
        b = d_in2/2 + (s2+w2)*j

        k = (4*a*b/( (a+b)**2 + D**2))**0.5 # equation 20, Schormans
        Kk = scipy.special.ellipk(k)
        Ek = scipy.special.ellipe(k)
        Mk = 4*3.14*10**(-7) * (a*b)**0.5 * ( (2/k - k)*Kk - 2/k*Ek ) # equations 18 & 19, Schormans
        M += Mk
#print("mutual inductance (H): " + str(M))

# calculate inductance
B1 = (d1 - d_in1)/(d1 + d_in1) # equation 3, Schormans
d_avg1 = 0.5*(d1+d_in1)
u = 4*3.14*10**(-7) # FR4 permeability, roughly equivalent to free space permeability
L1 = u * n1**2 * d_avg1/2 * (np.log(2.46/B1) + 0.2*B1**2 ) # inductance of each coil, equation 3, Schormans
#print("L1 (H): " + str(L1) )

B2 = (d2 - d_in2)/(d2 + d_in2) # equation 3, Schormans
d_avg2 = 0.5*(d2+d_in2)
L2 = u*n2**2*d_avg2/2*(np.log(2.46/B2) + 0.2*B2**2 ) # inductance of each coil, equation 3, Schormans
#print("L2 (H): " + str(L2) )

# calculate k
k_coupling = M/(L1*L2)**0.5
#print("k, coupling factor: " + str(k_coupling))

# calculate C
C1 = 1/(4*3.14**2*f**2*L1) # from f = 1/(2*PI*sqrt(L*C))
C2 = 1/(4*3.14**2*f**2*L2) # from f = 1/(2*PI*sqrt(L*C))
#print("C1 (F): "+  str(C1))
#print("C2 (F): "+  str(C2))

# calculate R_DC
t0_cu = 0.034798*10**(-3) # copper thickess, m via PCB Universe [6]
p = 1.68 *10**(-8) # resistivity of Cu, Ohm-m
```

```
A1 = t0_cu*w1    # cross sectional area, m

R_DC1 = p*3.14*(d1 - (w1+s1)*n1/2)*n1/A1 # equation 5, Schormans

A2 = t0_cu*w2    # cross sectional area, m

R_DC2 = p*3.14*(d2 - (w2+s2)*n2/2)*n2/A2 # equation 5, Schormans


# calculate R_skin

omega = 2*3.14*f

delta = (2*p/ (omega*u) )**0.5

R_skin1 = R_DC1*t0_cu/(delta*(1-2.71828**(-t0_cu/delta)))*1/(1+t0_cu/w1) # equation 10, Schormans

R_skin2 = R_DC2*t0_cu/(delta*(1-2.71828**(-t0_cu/delta)))*1/(1+t0_cu/w2) # equation 10, Schormans


# calculate R_prox

omega_crit1 = 3.1/u*(w1+s1)*p/(w1**2*t0_cu) # equation 13, Schormans

R_prox1 = R_DC1/10*(omega/omega_crit1)**2 # equation 12, Schormans

omega_crit2 = 3.1/u*(w2+s2)*p/(w2**2*t0_cu) # equation 13, Schormans

R_prox2 = R_DC2/10*(omega/omega_crit2)**2 # equation 12, Schormans


# calculate R_s from R_DC, R_skin, and R_prox

R_s1 = R_DC1 + R_skin1 + R_prox1 # equation 4, Schormans

R_s2 = R_DC2 + R_skin2 + R_prox2 # equation 4, Schormans

#print("Rs1 (Ohms): "+  str(R_s1))

#print("Rs2 (Ohms): "+  str(R_s2))


# -- calculate reflected R

Q_2 = omega*L2/R_s2  # page 3, Kiani

Q_L = R_L/(omega*L2) # page 3, Kiani

Q_2L =Q_2*Q_L/(Q_2 + Q_L)

R_refl = (M**2/(L1*L2)) * omega *  L1 * Q_2L # page 3, Kiani

#print ("reflected resistance (Ohms): "+str(R_refl))


#desired R_refl

#print ("desired relfected resistace (Ohms): " + str(R_s1 + R_src)) ## for impedance match


# ----- EVALUATION -----

# -- power transfer efficiency (PTE)
```

```
        PTE = R_refl/(R_s1 + R_src + R_refl)*(Q_2L/Q_L) # equation 2, Kiani

        #print("PTE: " + str(PTE))


        # -- power delivered to load (PDE)

        P_L = V_src**2*R_refl/(2*(R_s1 + R_src + R_refl)**2)*Q_2L/Q_L # equation 3, Kiani

        #print("PDL (uW): " + str(P_L*10**6))


        # max PDL -> assumes R_src + R_s1 = R_refl

        P_L_max = V_src**2 / (4*(R_src + R_s1)) # by hand

        #print("PDL max, assuming impedance match (uW): " + str(P_L_max*10**6))


        # upate PTE and P_L arrays with values calculated

        PTE_array[n1][n2] = PTE

        P_L_array[n1][n2] = P_L


print("Possible values of n1 and n2 for given input constraints...")

# for each n1, n2 combination

for n1 in range(1,n_max):

    for n2 in range (1,n_max):

        # check if P_L is greater than the required P_L

        if( P_L_array[n1][n2] > P_L_req):

            # print out n1, n2, d1, d2, PTE, and PDL

            d1 = str(round( (d_in1 + (s1+w1)*n1*2*10**2), 2) )

            d2 = str(round( (d_in2 + (s2+w2)*n2*2*10**2), 2 ) )

            PTE = str(round(PTE_array[n1][n2],10))

            PDL = str(round(P_L_array[n1][n2]*10**(6),2))

            print("n1 = " + str(n1) + ", n2 = " + str(n2) + ": d1(cm) = " + d1 + ", d2(cm)= " + d2 +  ", PTE = " + PTE + ", PDL(uW)= " + PDL )
```