ENGT4303 Assignment

Bernard Blackham (10216872)

Due: November 1, 2006

At 500 MHz, the s-parameters for the device are: $\begin{bmatrix} 0.57\angle 176° & 0.05\angle 67° \\ 8.4\angle 86° & 0.14\angle -130° \end{bmatrix}$

At 1 GHz, the s-parameters for the device are: $\begin{bmatrix} 0.60\angle 156° & 0.09\angle 70° \\ 4.4\angle 75° & 0.11\angle -164° \end{bmatrix}$

Test for stability

The Rollert Stability Factor K is given by

$$\begin{aligned} K &= \frac{1 - |S_{11}|^2 - |S_{22}|^2 + |\Delta|^2}{2\,|S_{12}S_{21}|} && (1) \\ &= 1.0213 \text{ @ 500 MHz} && (2) \\ &= 1.0551 \text{ @ 1 GHz} && (3) \end{aligned}$$

B1 is given by

$$\begin{aligned} B1 &= 1 + |S_{11}|^2 - |S_{22}|^2 - |\Delta|^2 && (4) \\ &= 1.103 \text{ @ 500 MHz} && (5) \\ &= 1.140 \text{ @ 1 GHz} && (6) \end{aligned}$$

At both frequencies, $K > 1$ and $B1 > 0$. Therefore, the device is unconditionally stable. We examine the stability circles anyway.

The input stability circle is given by

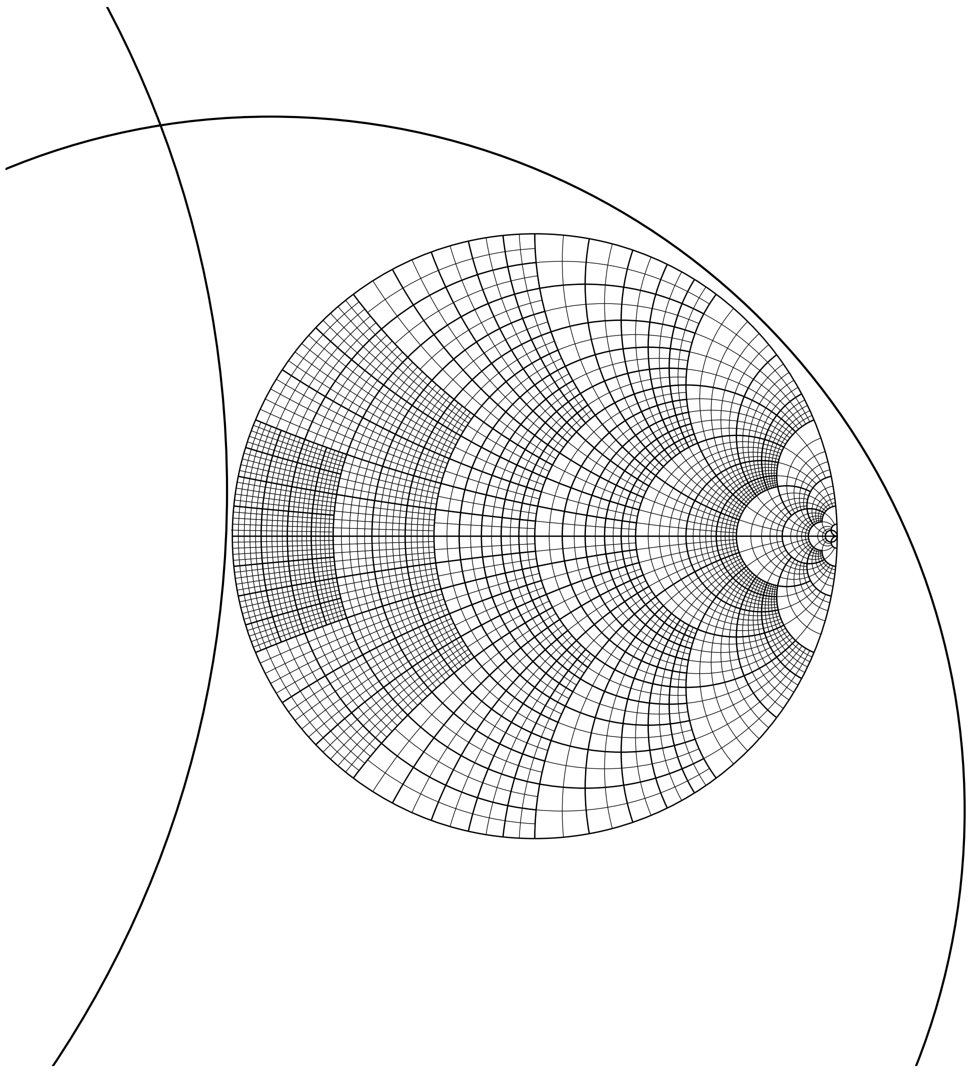
$$\begin{aligned} C_{Load} &= \frac{(S_{22} - S_{11}^{*}\Delta)^{*}}{|S_{22}|^2 - |\Delta|^2} \\ &= -0.873 - j0.907 \text{ @ 500 MHz} \\ &= -0.846 - j0.331 \text{ @ 1 GHz} \\ r_{Load} &= \frac{|S_{12}S_{21}|}{\left||S_{22}|^2 - |\Delta|^2\right|} \\ &= 2.298 \text{ @ 500 MHz} \\ &= 2.024 \text{ @ 1 GHz} \end{aligned}$$

In the case where $Z_L = Z_0$, as $S_{11} < 1$, then $|\Gamma_{IN}| < 1$ and thus the center of the Smith chart is the stable region.

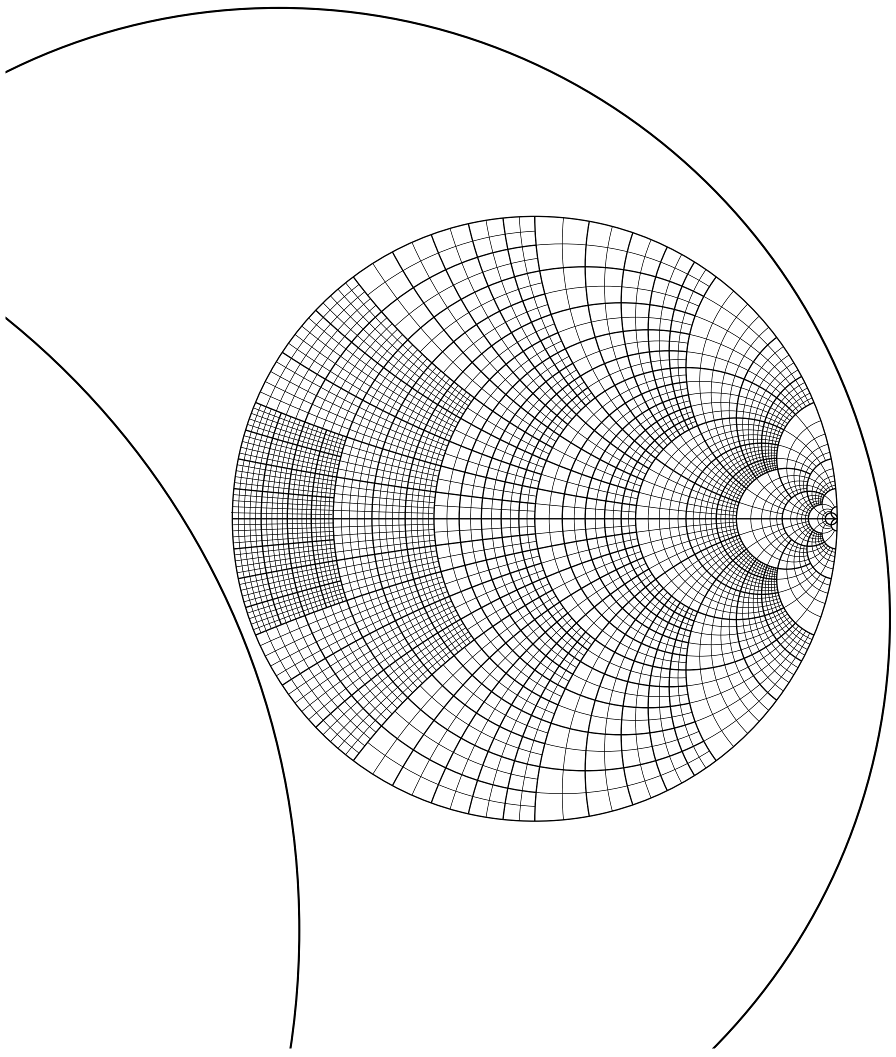
Similarly, the output stability circle is given by:

$$\begin{aligned}
C_{Source} &= \frac{(S_{11} - S_{22}^{*}\Delta)^{*}}{|S_{11}|^{2} - |\Delta|^{2}} \\
&= -4.442 + j0.149 \text{ @ } 500 \text{ MHz} \\
&= -3.376 - j1.361 \text{ @ } 1 \text{ GHz} \\
r_{Source} &= \frac{|S_{12}S_{21}|}{\left||S_{11}|^{2} - |\Delta|^{2}\right|} \\
&= 3.428 \text{ @ } 500 \text{ MHz} \\
&= 2.601 \text{ @ } 1 \text{ GHz}
\end{aligned}$$

At 500 Mhz, the input and output stability circles are below.



At 1 GHz, the input and output stability circles are below.



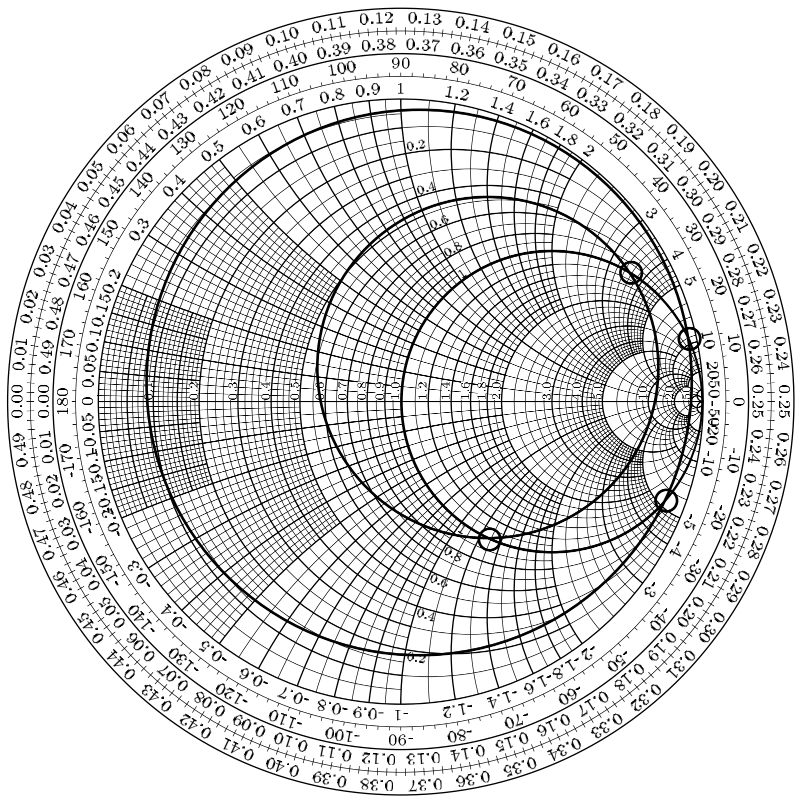
Circles of Constant Gain

We assume that the input is conjugately matched, so $G_T = G_P$.

For 14 dB, $G_P = 10^{1.4} = 25.12$. We find:

$$\begin{aligned}
g_p &= \frac{G_p}{|S_{21}|^2} && (7)\\
&= \frac{25.12}{8.4^2} && (8)\\
&= 0.356 && (9)\\
C_P &= \frac{g_p\left(S_{22} - \Delta S_{11}^*\right)^*}{1 + g_p\left(|S_{22}|^2 - |\Delta|^2\right)} && (10)\\
&= 0.0608 + j0.0632 \text{ @ 500 MHz} && (11)\\
&= 0.2879 + j0.1126 \text{ @ 1 GHz} && (12)\\
r_P &= \frac{\left(1 - 2K\,|S_{12}S_{21}|\,g_p + |S_{12}S_{21}|^2\,g_p^2\right)^{\frac{1}{2}}}{\left|1 + g_p\left(|S_{22}|^2 - |\Delta|^2\right)\right|} && (13)\\
&= 0.9057 \text{ @ 500 MHz} && (14)\\
&= 0.5682 \text{ @ 1 GHz} && (15)\\
& && (16)
\end{aligned}$$

These two circles are plotted below. Using $Z_{\text{REF}} = 50\Omega$, we aim for the intersections of the locus of values for which $r = 1$ intersects the constant gain curves. This occurs at two points for each circle of constant gain, and are also shown on the diagram. The intersections of the 500 MHz circle and the $Z_L = Z_{\text{REF}}$ are at (0.955,0.207) and (0.878,-0.327). These correspond to impedances of $1 + j9.207$ and $1 - j5.365$, respectively. The intersections of the 1 GHz circle and the $Z_L = Z_{\text{REF}}$ are at (0.762, 0.426) and (0.294, -0.456). These correspond to impedances of $1 + j3.578$ and $1 - j1.290$, respectively.



Output matching network design

In order to attain points on the circle of constant gain at both 500 MHz and 1 GHz, we use a capacitor and inductor in series. The series capacitor also eliminates and DC component from the load interfering with the operation of the circuit. As our targets, we choose the intersection points at $1 - j5.365$ on the 500 MHz circle and $1 - j1.290$ on the 1 GHz circle.

Thus the reactive component is given by:

$$X = j\omega L + \frac{1}{j\omega C} \tag{17}$$

$$\tag{18}$$

At 500 MHz:

$$\begin{aligned} -j5.365 &= j \times 2\pi 500 \times 10^6 \frac{L}{Z_{\mathrm{REF}}} + \frac{1}{j \times 2\pi 500 \times 10^6 C Z_{\mathrm{REF}}} && (19) \\ -5.365 &= 2\pi 500 \times 10^6 \frac{1}{50} L - \frac{1}{50 \times 2\pi 500 \times 10^6} \frac{1}{C} && (20) \\ & && (21) \end{aligned}$$

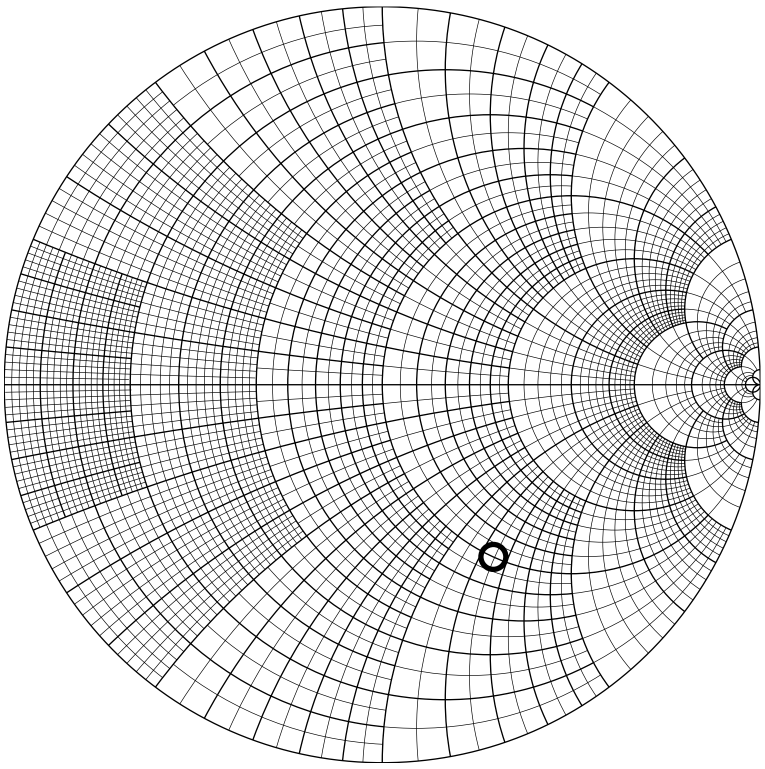
Similarly at 1 GHz:

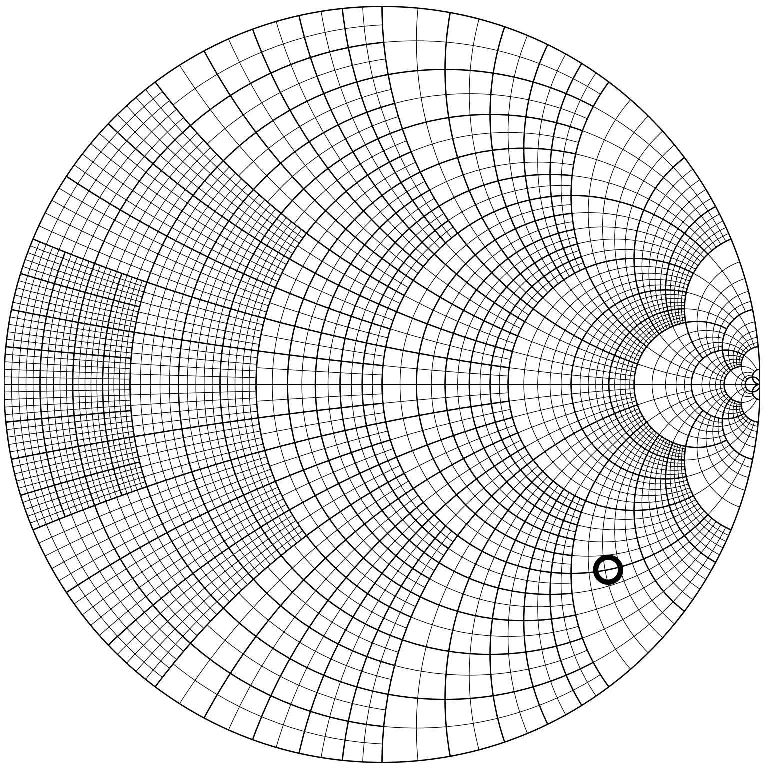
$$\begin{aligned} -j1.290 &= j2\pi \times 10^9 \frac{L}{Z_{\mathrm{REF}}} + \frac{1}{j2\pi \times 10^9 C Z_{\mathrm{REF}}} && (22) \\ -1.290 &= 2\pi 10^9 \frac{L}{50} - \frac{1}{50 \times 2\pi 10^9} \frac{1}{C} && (23) \\ & && (24) \end{aligned}$$

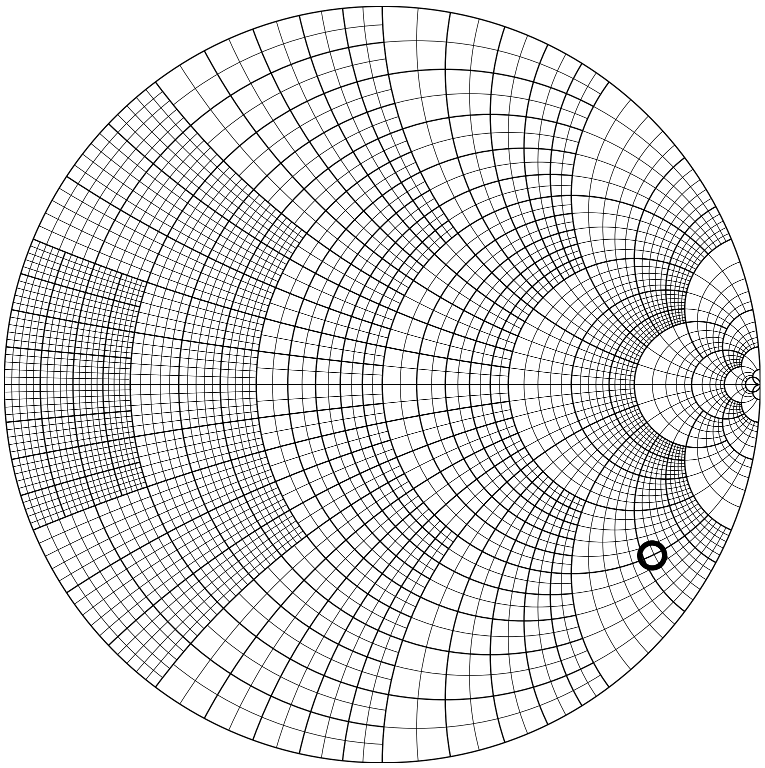
This gives a set of simultaneous equations, with solution:

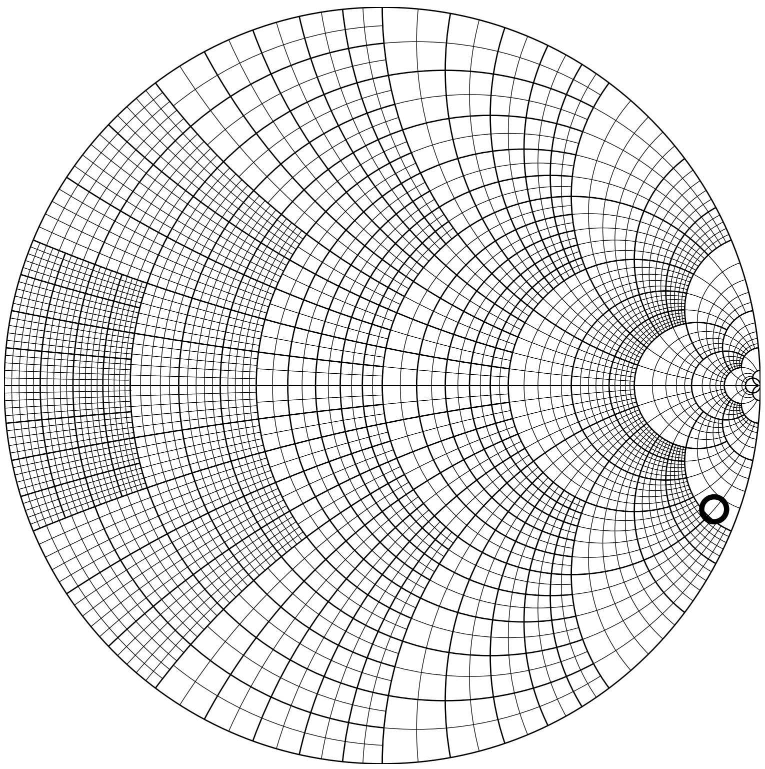
$$\begin{aligned} L &= 14.76\,\mathrm{nH} && (25) \\ \frac{1}{C} &= 9.879 \times 10^{11}\,\mathrm{F}^{-1} && (26) \\ C &= 1.012\,\mathrm{pF} && (27) \\ & && (28) \end{aligned}$$

The change in impedance is shown in the smith charts below for 500, 700, 800 and 1 000 GHz.


500 MHz


700 MHz


800 MHz


1 GHz

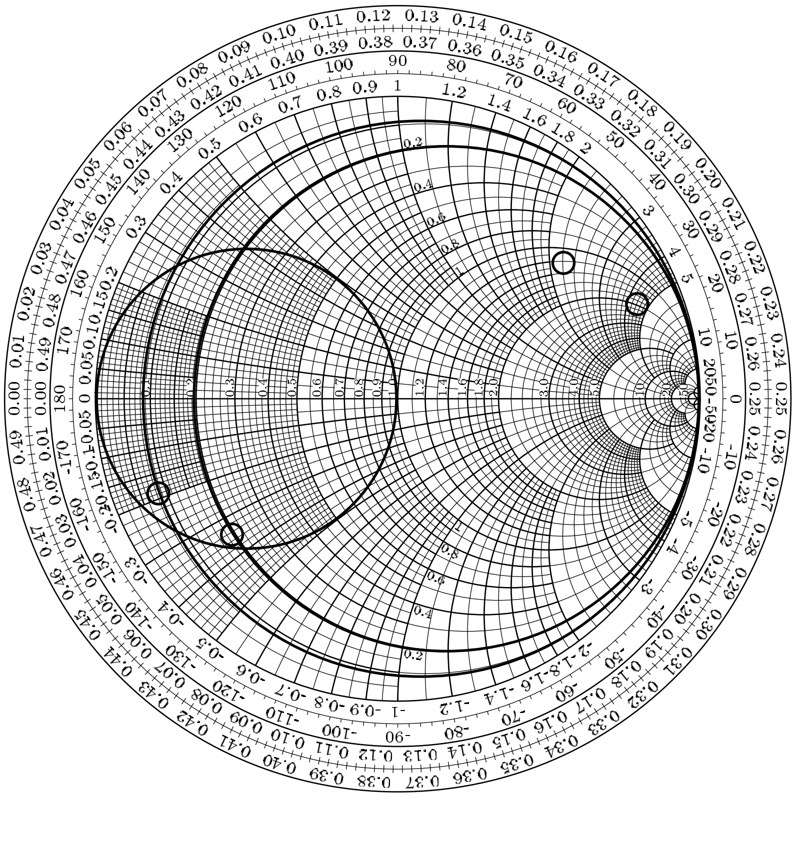
Input matching network design

As the constant gain curves were calculated under the assumption that the input was conjugately matched to the source, we have

$$
\begin{aligned}
\Gamma_S &= \Gamma_{in}^* && (29)\\
&= \left(S_{11} + \frac{S_{12}S_{21}\Gamma_L}{1 - S_{22}\Gamma_L}\right)^* && (30)\\
&= -0.792 - j0.313 \text{ @ } 500\text{ MHz} && (31)\\
&= -0.548 - j0.449 \text{ @ } 1\text{ GHz} && (32)\\
Z_S &= \frac{1+\Gamma_S}{1-\Gamma_S} && (33)\\
&= 0.0833 - j0.1892 \text{ @ } 500\text{ MHz} && (34)\\
&= 0.1916 - j0.3459 \text{ @ } 1\text{ GHz} && (35)\\
& && (36)
\end{aligned}
$$

After much trial and error and attemping various combinations of series and parallel LC circuits, no successful matching circuit could be found to achieve these values of Z_S. Numerous methods were used to search for a solution, including graphical manipulation with computer tools such as "linsmith", solving complex expressions for Z_S, and algebraic manipulation.

The load-matching circuit may have to be modified in order to give a result, but time did not allow this.



Code listing for solving equations

```
#!/usr/bin/python

from math import *
import sys

def to_cart(mag, deg):
    rad = radians(deg)
    return mag*(cos(rad) + 1j * sin(rad))

def to_polar(x):
    ang = 0
    if x.real == 0:
        ang = pi/2
    else:
        ang = atan(x.imag / x.real)
    if x.real < 0:
        ang += pi
    while ang > pi:
        ang -= 2*pi
    while ang < -pi:
        ang += 2*pi
    return (abs(x), degrees(ang))

def csqrt(x):
    (mag, deg) = to_polar(complex(x))
    return to_cart(sqrt(mag), deg/2.)

def dB(x):
    return 10 * log(x) / log(10)

def par(a,b):
    return 1./(1./a + 1./b)

def LogFunc(s):
    sys.stderr.write(s)

def is_sensible_range(f, lo, hi, sample_points = 50):
    '''Determines if a function goes up once and down once within the range
    [lo, hi] at a certain number of sample points.
    Returns a 2-tuple of booleans.
    If the first tuple is true, the condition holds.
    Otherwise, if the second tuple is True, the function was monotonic in the
    range [lo, hi] (ie, range too small).
    If the second tuple is False, the function went up and down at least once
    in the range[lo, hi] (ie, range too large).'''
    x = lo
    going_up = -2
    changed = False
    step = (hi - lo) / sample_points
    #print '\t\tis_sensible_range(f, %s, %s)'%(str(lo),str(hi))
    while x < hi:
        #print "Trying with", x, 'giving', f(x)
        res = f(x)
        if going_up == -2:
            # Don't have any data points yet. Will do next time.
            going_up = -1
        elif going_up == -1:
            # Now have a data point. Figure out which way we're going
            if prev_res < res:
                going_up = 1
            else:
                going_up = 0
        elif going_up == 0:
            if prev_res < res:
                if changed:
                    return (False, False) # 2 changes of direction already. Quit it.
                else:
                    changed = True
                    going_up = 1
        elif going_up == 1:
            if prev_res > res:
                if changed:
                    return (False, False) # 2 changes of direction already. Quit it.
                else:
                    changed = True
                    going_up = 0
        prev_res = res
        x += step
    # All went well.
    if changed:
        return (True, True)
    return (False, True)

def find_sensible_increment(f, start, start_incr):
    '''Determines an increment suitable to start solving for E with.'''
    lo = start
    hi = start + start_incr
    #print '\t\tfind_sensible_increment(%s,%s)'%(str(lo),str(hi))
    sample_points = 50.
    while True:
        (good, too_small) = is_sensible_range(f, lo, hi, sample_points)
        if good:
            return (hi-lo)/sample_points/10
        if too_small:
            #print '\ttoo small', lo, hi
            hi *= 2.
        else:
            #print '\ttoo big', lo, hi
            hi /= 11.

def solve(f, start_x, start_increment, limit):
    '''Solve for the function f(x) = 0 starting from start_x.'''
    # Firstly find a sensible range to search in.
    x = start_x
    incr = find_sensible_increment(f, x, start_increment)
    LogFunc(' .')
    sys.stdout.flush()

    # Find the point where the function crosses the axis.
    prev_y = None
    while True:
        y = f(x)
        if prev_y != None and (y > 0 and prev_y < 0) or (y < 0 and prev_y > 0):
            break
        prev_y = y
        x += incr
        if limit != None and x > limit:
            LogFunc('! at %g > %g\n'%(x, limit))
            sys.stdout.flush()
            return (None, start_increment, limit)
    LogFunc('.')
    sys.stdout.flush()

    # Solution lies between x-incr. and x. Go searching.
    res = solve_for_zero(f, x-incr, x)
    LogFunc('.')
    return (res, x, incr)


def solveit(w1, z1, w2, z2):
    func = lambda w,Cs,Ls,Cp,Lp: par(1+1j*w*Ls-1.j/(w*Cs), par(1j*w*Lp, -1j/(w*Cp)))

    Cs = 1e-12
    Ls = 1e-9
    Cp = 1e-12
    Lp = 1e-9

    #f = lambda w,Cs,Ls,Cp,Lp: par(1+1j*w*Ls-1.j/(w*Cs), par(1j*w*Lp, -1j/(w*Cp)))
    params = [Cs, Ls, Cp, Lp]
    things = ['Cs', 'Ls', 'Cp', 'Lp']
    while True:
        (Cs, Ls, Cp, Lp) = params
        print "Cs = %g Ls = %g Cp = %g Lp = %g"%(Cs, Ls, Cp, Lp)
        Z = func(w1, Cs, Ls, Cp, Lp)
        print "At 500 MHz:"
        print "   Want:", z1
        print "   Got :", Z
        Z = func(w2, Cs, Ls, Cp, Lp)
        print "At 1000 MHz:"
        print "   Want:", z2
        print "   Got :", Z
        print
        print "Change (1 = Cs, 2 = Ls, 3 = Cp, 4 = Lp) ?",
        s = sys.stdin.readline().strip()
        if s == '': break
        which = ord(s[0]) - ord('1')
        if which < 0 or which > 3: continue
        print "Enter %s:"%(things[which]),
        s = sys.stdin.readline().strip()
        if s == '': continue
        params[which] = float(s)

    return (Cs, Ls, Cp, Lp)


def solveit(w1, z1, w2, z2):
    func = lambda w,Cs,Ls,Cp,Lp: par(1+1j*w*Ls-1.j/(w*Cs), par(1j*w*Lp, -1j/(w*Cp)))

    Cs = 1e-12
    Ls = 1e-9
    Cp = 1e-12
    Lp = 1e-9

    #f = lambda w,Cs,Ls,Cp,Lp: par(1+1j*w*Ls-1.j/(w*Cs), par(1j*w*Lp, -1j/(w*Cp)))
    params = [Cs, Ls, Cp, Lp]
    things = ['Cs', 'Ls', 'Cp', 'Lp']
    while True:
        (Cs, Ls, Cp, Lp) = params
        print "Cs = %g Ls = %g Cp = %g Lp = %g"%(Cs, Ls, Cp, Lp)
        Z = func(w1, Cs, Ls, Cp, Lp)
        print "At 500 MHz:"
        print "   Want:", z1
        print "   Got :", Z
        Z = func(w2, Cs, Ls, Cp, Lp)
        print "At 1000 MHz:"
        print "   Want:", z2
        print "   Got :", Z
        print
        print "Change (1 = Cs, 2 = Ls, 3 = Cp, 4 = Lp) ?",
        s = sys.stdin.readline().strip()
        if s == '': break
        which = ord(s[0]) - ord('1')
        if which < 0 or which > 3: continue
        print "Enter %s:"%(things[which]),
        s = sys.stdin.readline().strip()
        if s == '': continue
        params[which] = float(s)
    return (Cs, Ls, Cp, Lp)


gainDb = 14
ranges = [
    [ "500 MHz",
        to_cart(0.57, 176), to_cart(0.05, 67),
        to_cart(8.4, 86), to_cart(0.14, -130),
        1 - 5.365j,
    ],
    [ "1 GHz",
        to_cart(0.60, 156), to_cart(0.09, 70),
        to_cart(4.4, 75), to_cart(0.11, -164),
        1 - 1.290j,
    ],
]
for (freq, S11, S12, S21, S22, Zl) in ranges:
    print " ------- @ %s"%(freq)
    Det = S11*S22 - S12*S21
    print " Det  =",Det
    print "|Det| =",abs(Det)
    C1 = S11 - Det * S22.conjugate()
    B1 = 1 + abs(S11)**2 - abs(S22)**2 - abs(Det)**2
    C2 = S22 - Det * S11.conjugate()
    B2 = 1 + abs(S22)**2 - abs(S11)**2 - abs(Det)**2
    K = (1 + abs(Det)**2 - abs(S11)**2 - abs(S22)**2) / (2. * abs(S12 * S21))
    D2 = abs(S22)**2 - abs(Det)**2
    GammaL = 0.9642304

    # Rollert Stability Factor K
    print 'K =', K
    print 'B1 =', B1

    # Input Stability Circle
    Cload = (S22 - S11.conjugate() * Det).conjugate() / (abs(S22)**2 - abs(Det)**2)
    rload = abs((S12*S21)) / abs(abs(S22)**2 - abs(Det)**2)
    print 'Cload =', Cload
    print 'rload =', rload

    # Output Stability Circle
    Csource = (S11 - S22.conjugate()*Det).conjugate() / (abs(S11)**2 - abs(Det)**2)
    rsource = abs(S12*S21) / abs(abs(S11)**2 - abs(Det)**2)
    print 'Csource =', Csource
    print 'rsource =', rsource

    # Maximum power gain
    Gmax = abs(S21)/abs(S12) * (K - csqrt(K**2 - 1))

    # Operating gain
    gp = (1 - abs(GammaL)**2) / \
     ( abs(1 - S22*GammaL)**2 - abs(S11 - Det*GammaL)**2)
Gp = abs(S21)**2 * gp
print 'Gp =', Gp, '=', dB(Gp), 'dB'

# Circle of constant gain
Gp = 10.**(gainDb/10.)
gp = Gp / (abs(S21)**2)
Cp = (gp * C2.conjugate()) / (1 + gp * (abs(S22)**2 - abs(Det)**2))
rp = csqrt(1 - 2*K*abs(S12*S21)*gp + (abs(S12*S21)**2)*(gp**2)) / \
     abs(1 + gp * (abs(S22)**2 - abs(Det)**2))
print 'gp =', gp
print 'Cp =', Cp
print 'rp =', rp

print 'Zl =', Zl

GammaL = (Zl-1)/(Zl+1)
print 'GammaL =', GammaL

GammaS = (S11 + (S12*S21*GammaL)/(1 - S22*GammaL)).conjugate()
print 'GammaS =', GammaS

Zs = (1+GammaS)/(1-GammaS)
print 'Zs =', Zs
```