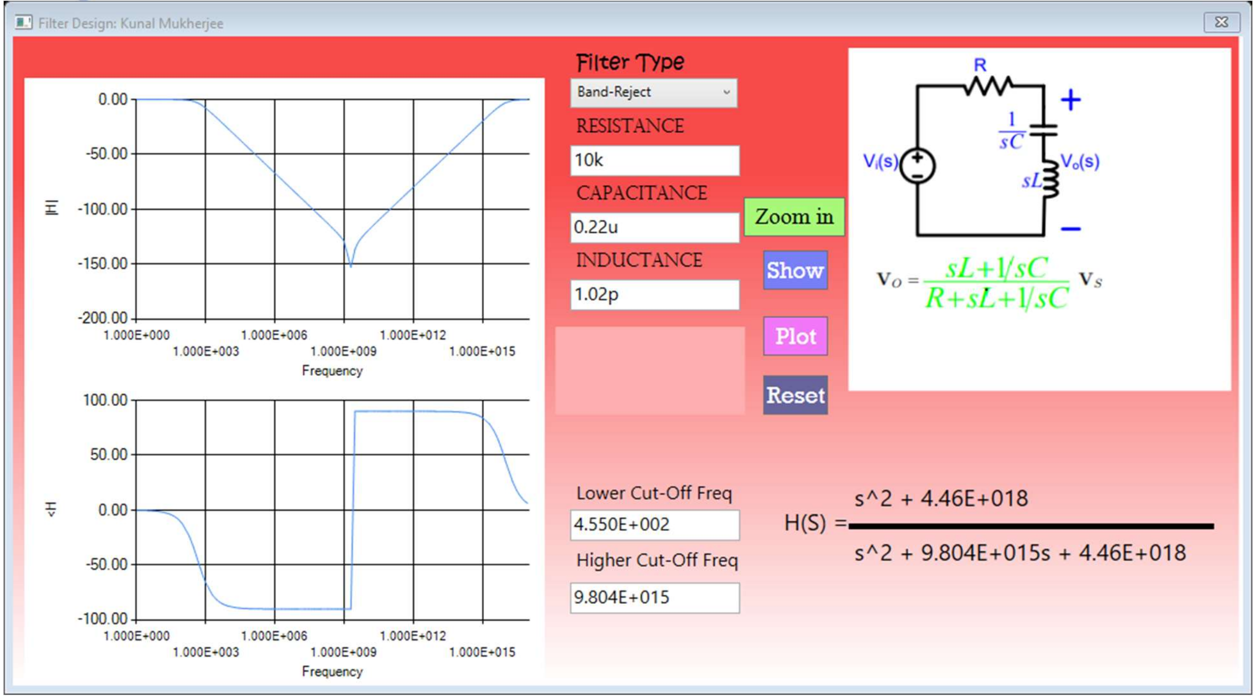
RC, RL, AND RLC FILTER DESIGN

Kunal Mukherjee



Filter Design: Kunal Mukherjee

Filter Type: Band-Reject

RESISTANCE: 10k

CAPACITANCE: 0.22u

INDUCTANCE: 1.02p

Lower Cut-Off Freq: 4.550E+002

Higher Cut-Off Freq: 9.804E+015

$$V_O = \frac{sL+1/sC}{R+sL+1/sC} V_S$$

$$H(S) = \frac{s\text{^}2 + 4.46E{+}018}{s\text{^}2 + 9.804E{+}015s + 4.46E{+}018}$$

MARCH 2, 2019

UNIVERSITY OF EVANSVILLE

Contents

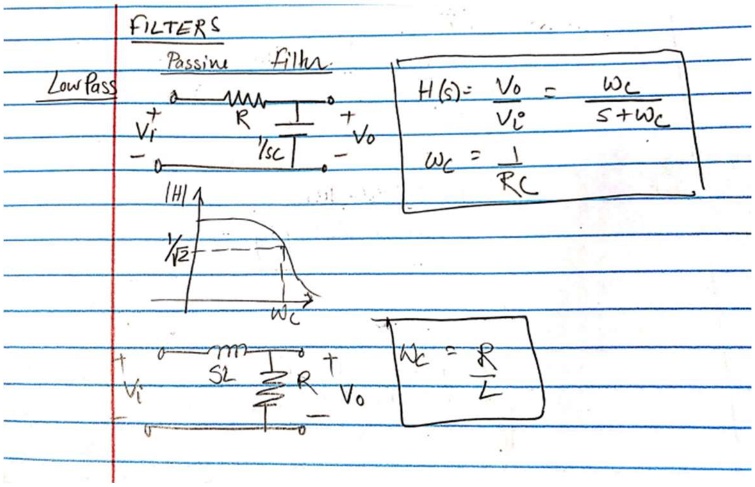
Problem Design

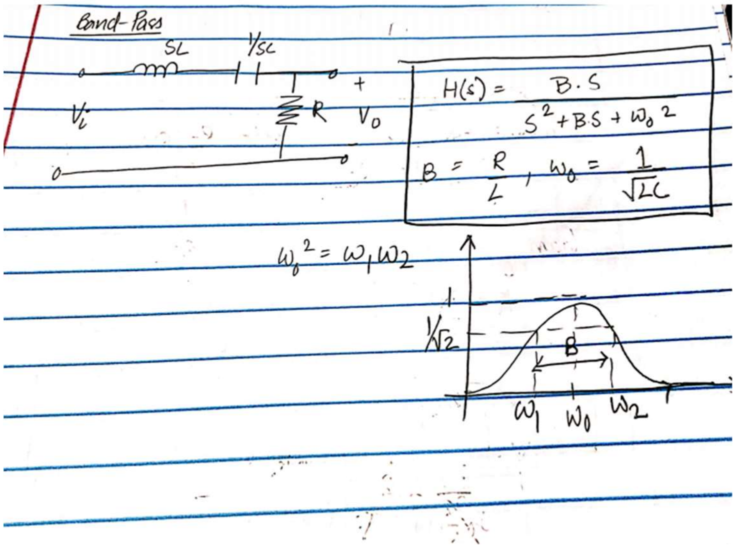
The problem we are trying to solve is how to accurately guess the frequency response from R, L or C value. Therefore, building a simulator to emulate different filter types is the best choice of action. In engineering sometimes, it is necessary to check a pre-built circuit's frequency response as well as to know what value of resistors, capacitors and inductors to use to build the filter. Therefore, I decided to design C# WPF application that shows the user the Bode magnitude plot and Bode phase plot, if the user enters the filter type then, R, L or C value, or center/cut-off frequency and any one of R, L or C value. The application also shows the transfer function and a sample circuit diagram.

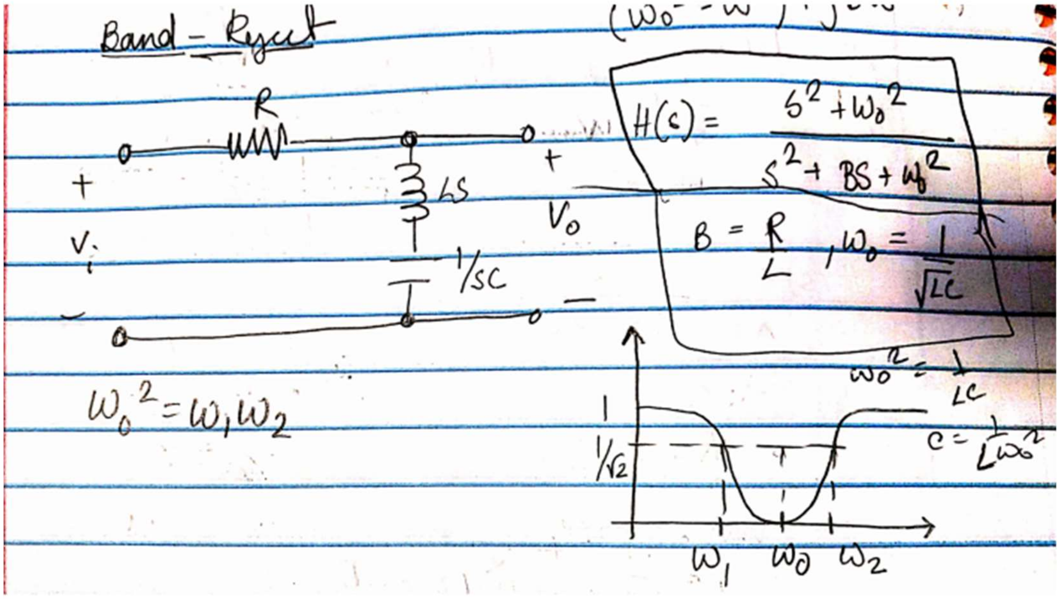
Design

There were two major design hurdles. The first design hurdle was to get all the transfer functions for RC low/high pass, RL low/high pass, RLC bandpass or RLC band stop filters. The second hurdle was to find the appropriate frequency increment value, so that all the necessary information of the bode plot is preserved but does not need a huge dataset.

The transfer function was calculated using the following equations:




Band-Pass
SL
1/SC
+
R
V_i
V_o
$H(s) = \frac{B \cdot S}{S^2 + B S + \omega_0^2}$
$B = \frac{R}{L}, \ \omega_0 = \frac{1}{\sqrt{LC}}$
$\omega_0^2 = \omega_1 \omega_2$
1
$1/\sqrt{2}$
B
ω_1
ω_0
ω_2


Band-Reject
R
+
V_i
LS
1/SC
V_o
−
$H(s) = \frac{S^2 + \omega_0^2}{S^2 + BS + \omega_0^2}$
$B = \frac{R}{L}, \ \omega_0 = \frac{1}{\sqrt{LC}}$
$\omega_0^2 = \omega_1 \omega_2$
$\omega_0^2 = \frac{1}{LC}$
$C = \frac{1}{L\omega_0^2}$
1
$1/\sqrt{2}$
ω_1
ω_0
ω_2

The second design issue was solved by using a multiplicative increment. For example, let say, we want to graph for a frequency of 10-1000 Hz, I will use the following frequency values e.g. 1,2,3...10,20,30…100,200,300…1000. In that way, we don’t need all the points in between 10-20, as the log scale of the frequency axis will not care, and this increment captures all the necessary information to plot the frequency response.

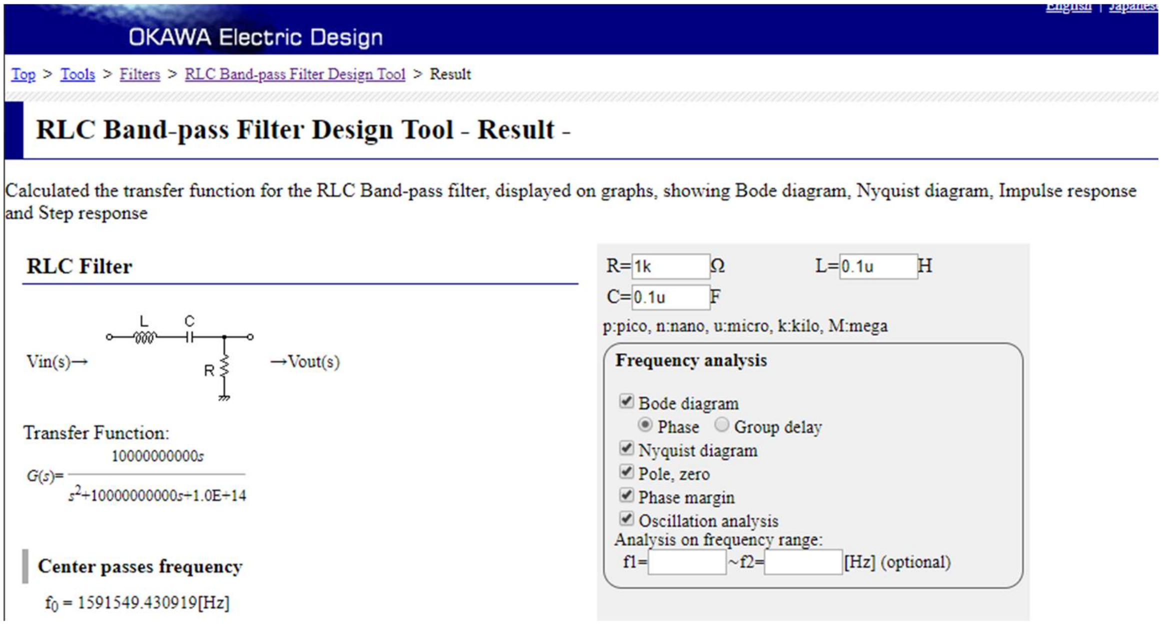
As extra feature, I have added, that you can put u-micro, p-pico, n-nano, f-femto, m-mili, k or K-kilo and M-mega along with numbers for R, L and C values. I have also added a zoom feature that will zoom in the center or the cut-off frequency depending on the filter type.

Theoretical Results

The project should be working as per the following specifications mentioned above as the calculated transfer function matches with the transfer function in the book.

Simulation Results

I used OKAWA Electric Design simulator software to test my design. The simulation result matched with the application results. The following illustration is showing the band pass filter for 1k, 0.1u C and 0.1u for I, from OKAWA electric and my application.



OKAWA Electric Design

English | Japanese

Top > Tools > Filters > RLC Band-pass Filter Design Tool > Result

RLC Band-pass Filter Design Tool - Result -

Calculated the transfer function for the RLC Band-pass filter, displayed on graphs, showing Bode diagram, Nyquist diagram, Impulse response and Step response

RLC Filter

Transfer Function:

$$G(s)=\frac{10000000000s}{s^2+10000000000s+1.0E+14}$$

Center passes frequency

f_0 = 1591549.430919[Hz]

R= 1k Ω L= 0.1u H

C= 0.1u F

p:pico, n:nano, u:micro, k:kilo, M:mega

Frequency analysis

☑ Bode diagram

● Phase ○ Group delay

☑ Nyquist diagram

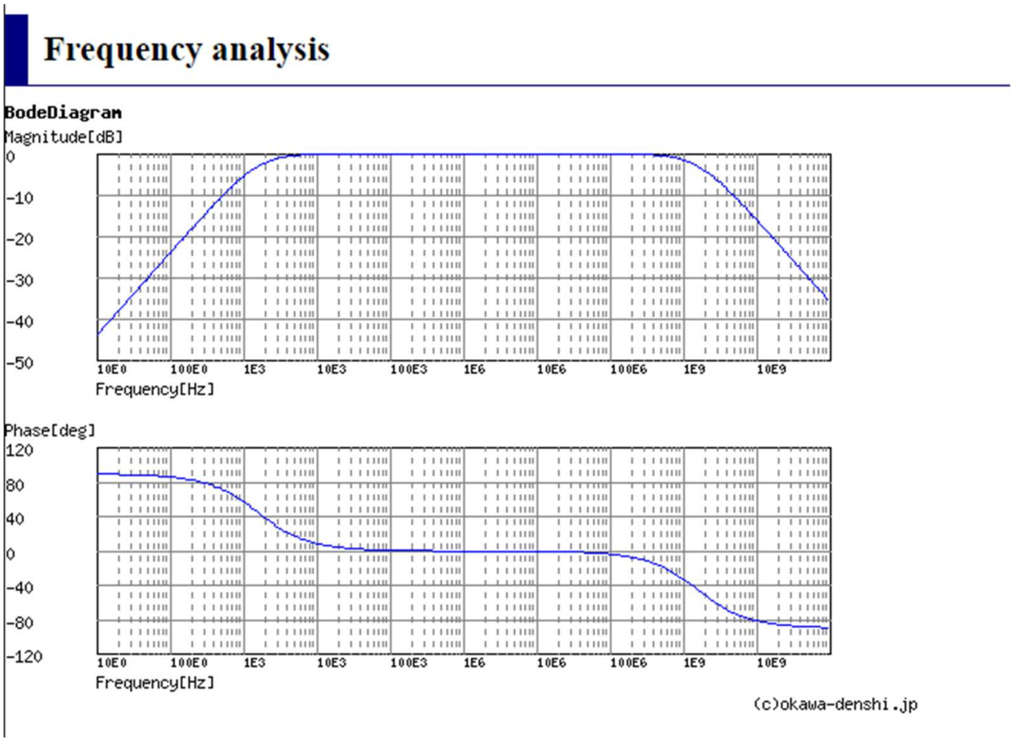
☑ Pole, zero

☑ Phase margin

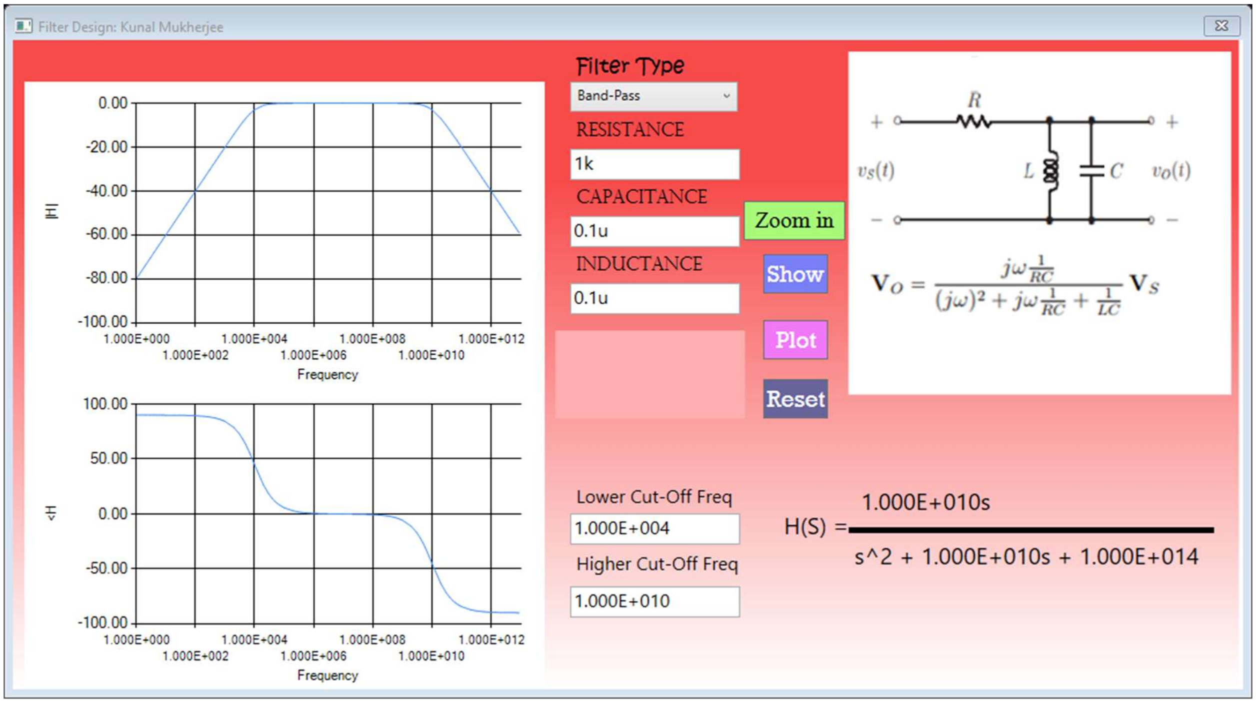
☑ Oscillation analysis

Analysis on frequency range:

f1= ~f2= [Hz] (optional)



Frequency analysis



Filter Design: Kunal Mukherjee

Filter Type: Band-Pass

RESISTANCE: 1k

CAPACITANCE: 0.1u

INDUCTANCE: 0.1u

Lower Cut-Off Freq: 1.000E+004

Higher Cut-Off Freq: 1.000E+010

$$\mathbf{V}_O = \frac{j\omega\frac{1}{RC}}{(j\omega)^2 + j\omega\frac{1}{RC} + \frac{1}{LC}}\mathbf{V}_S$$

$$H(S) = \frac{1.000E+010s}{s^2 + 1.000E+010s + 1.000E+014}$$

Experimental Design

The project did not need much experimentation, except for finding how much of a frequency range to show the relative trend of the graph.

Measure Results

The project did not need much measurement, but I checked all my simulation results e.g. frequency response, cut-off/center frequency and transfer function with OKAWA electric's simulation.

Conclusion

The project met the specifications of WPF C# application that shows the user the Bode magnitude plot and Bode phase plot, when the user enters the filter type then, R, L or C value, or center/cut-off frequency and any one of R, L or C value. The project also shows the sample circuit as well as the transfer function.

Code

```
//Kunal Mukherjee
//3/2/2019
//EE 380: Filter design

using System;
using System.Windows;
using System.Windows.Media.Imaging;
using System.Windows.Forms.DataVisualization.Charting;

namespace Project2_Filters
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            txtbxR.Text = "0";
            txtbxC.Text = "0";
            txtbxL.Text = "0";

        }

        //initializing variables
        private double r = 0;
        private double c = 0;
        private double l = 0;
        private double wc = 0;
        private double filter = 0;
        private double Mh = 0; //magnitude of H
        private double Ah = 0; //angle of H
        private double w0 = 0; //undamped natural frequency
        private double wl = 0; //lower cut-off frequency
        private double wu = 0; //upper cut-off frequency
        private double B = 0; //bandwidth
        private double Q = 0; //quality factor
        private double w = 0;
        private double tempVal = 0;
        private int nI = 0;
        private int nMax = 0;
        private System.Windows.Controls.Image img = new System.Windows.Controls.Image();
        public BitmapImage bmi;
        public WriteableBitmap wbm;
        public Series wASeries = new Series();
        public Series wPSeries = new Series();
        private Chart chtAmp = new Chart();
        private Chart chtPha = new Chart();

        //The function loads the |H| chart
        private void cnvChartAmp_Loaded(object sender, RoutedEventArgs e)
        {
            System.Windows.Forms.Integration.WindowsFormsHost host =
                   new System.Windows.Forms.Integration.WindowsFormsHost();
            host.Child = chtAmp;
            // Add the chart to the canvas so it can be displayed.
            this.cnvChartAmp.Children.Add(host);
        }

        //The function loads the <H chart
        private void cnvChartPha_Loaded(object sender, RoutedEventArgs e)
        {
            System.Windows.Forms.Integration.WindowsFormsHost host =
                   new System.Windows.Forms.Integration.WindowsFormsHost();
            host.Child = chtPha;
            // Add the chart to the canvas so it can be displayed.
            this.cnvChartPha.Children.Add(host);
        }

        //The function displays the appriopiate transfer function and the circut
        private void btnShow_Click(object sender, RoutedEventArgs e)
        {
            imgDrawing.Source = wbm;

            if (filter == 0)
            {
                lblHN.Content = wc.ToString( "E3");
                lblHD.Content = "s + " + wc.ToString("E3");
            }
            if (filter == 1)
            {
                lblHN.Content = "s";
                lblHD.Content = "s + " + wc.ToString("E3");
            }
            if (filter == 2)
            {
                lblHN.Content = " " + B.ToString("E3") + "s";
                lblHD.Content = "s^2 + " + B.ToString("E3") + "s + " + (Math.Pow(w0, 2)).ToString("E3");
            }
            if (filter == 3)
            {
                lblHN.Content = "s^2 + " + (Math.Pow(w0, 2)).ToString("E2");
                lblHD.Content = "s^2 + " + B.ToString("E3") + "s + " + (Math.Pow(w0, 2)).ToString("E2");
            }
        }

        //The function resets the R,L,C value to zero as well as the filter values
        private void btnReset_Click(object sender, RoutedEventArgs e)
        {
            txtbxR.Text = "0";
            txtbxC.Text = "0";
            txtbxL.Text = "0";
            txtbxCF.Text = "0";
            txtbxHCF.Text = "0";
            txtbxLCF.Text = "0";

            lblHD.Content = "D(s)";
            lblHN.Content = "H(s)";
        }

        //The function selects which canvas to display the necessary options
        private void CmbFilterType_DropDownClosed(object sender, EventArgs e)
        {
            if ((cmbFilterType.Text.Equals("Low-Pass")) ||
               (cmbFilterType.Text.Equals("High-Pass")))
            {
                cnvBP.Width = 164;
                cnvLP.Width = 0;
            }
            else
            {
                cnvBP.Width = 0;
                cnvLP.Width = 164;
            }

        }

        //The function recalculates the series acording to the zooming option
        //so that the x-axis can be zoomed in accordingly
        private void btnZoomIn_Click(object sender, RoutedEventArgs e)
        {
            double centFreq = 0;

            if (filter == 1 || filter == 2)
            {
                centFreq = wc;
            }
            else
            {
                centFreq = w0;
            }

            chtAmp.ChartAreas[0].AxisX.Maximum = centFreq + (0.95 * (chtAmp.ChartAreas[0].AxisX.Maximum - centFreq));
            chtAmp.ChartAreas[0].AxisX.Minimum = centFreq - (0.95 * (centFreq - chtAmp.ChartAreas[0].AxisX.Minimum));

            chtPha.ChartAreas[0].AxisX.Maximum = centFreq + (0.95 * (chtAmp.ChartAreas[0].AxisX.Maximum - centFreq));
            chtPha.ChartAreas[0].AxisX.Minimum = centFreq - (0.95 * (centFreq - chtAmp.ChartAreas[0].AxisX.Minimum));

            chtAmp.Series.Clear();
            chtAmp.Series.Add(wASeries);
            chtAmp.ChartAreas[0].AxisX.IsLogarithmic =  true;
            chtAmp.ChartAreas[0].AxisX.LogarithmBase = 10;
            chtAmp.ChartAreas[0].AxisX.Title = "Frequency";
            chtAmp.ChartAreas[0].AxisX.LabelStyle.Format =  "E3";
            chtAmp.ChartAreas[0].AxisY.Title = "|H|";
            chtAmp.ChartAreas[0].AxisY.LabelStyle.Format =  "{0.00}";

            chtPha.Series.Clear();
            chtPha.Series.Add(wPSeries);
            chtPha.ChartAreas[0].AxisX.IsLogarithmic =  true;
            chtPha.ChartAreas[0].AxisX.LogarithmBase = 10;
            chtPha.ChartAreas[0].AxisX.Title = "Frequency";
            chtPha.ChartAreas[0].AxisX.LabelStyle.Format =  "E3";
            chtPha.ChartAreas[0].AxisY.Title = "<H";
            chtPha.ChartAreas[0].AxisY.LabelStyle.Format =  "{0.00}";
        }

        //The function calls the appriopiate transfer function depending on the filter type
        //the function also get the R,L,C value as well as load the correct
        //curcit image
        private void btnPlot_Click(object sender, RoutedEventArgs e)
        {
            //select the filter type
            if (cmbFilterType.Text.Equals("Low-Pass"))
            {
                filter = 0;
            }
            else if (cmbFilterType.Text.Equals("High-Pass"))
            {
                filter = 1;
            }
            else if (cmbFilterType.Text.Equals("Band-Pass"))
            {
                filter = 2;
            }
            else if (cmbFilterType.Text.Equals("Band-Reject"))
            {
                filter = 3;
            }

            //get the r,c,l value
            if (txtbxR.Text != "")
            {
                r = getRLCValue(0);  // Convert.ToDouble(txtbxR.Text);
            }
            else
            {
                r = 10;
                txtbxR.Text = "10";
            }
            if (txtbxC.Text != "")
            {
                c = getRLCValue(1);  //Convert.ToDouble(txtbxC.Text);
            }
            else
            {
                c = 10;
                txtbxC.Text = "10";
            }
            if (txtbxL.Text != "")
            {
                l = getRLCValue(2);  //Convert.ToDouble(txtbxL.Text);
            }
            else
            {
                l = 10;
                txtbxL.Text = "10";
            }


            //depending on the choice select the filter type
            if (filter == 0)
            {
                if (c == 0)
                {
                    plotLowPassRLFilter();

                    bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents\Kunal\UE courses\EE-380\Project2_Filters\Project2_Filters\Image\LPRL.png"));
                    wbm = new WriteableBitmap(bmi);

                }
                else
                {
                    plotLowPassRCFilter();

                    bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents\Kunal\UE courses\EE-380\Project2_Filters\Project2_Filters\Image\LPRC.png"));
                    wbm = new WriteableBitmap(bmi);
                }
            }
            if (filter == 1)
            {
                if (c == 0)
                {
                    plotHighPassRLFilter();

                    bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents\Kunal\UE courses\EE-380\Project2_Filters\Project2_Filters\Image\HPRL.png"));
                    wbm = new WriteableBitmap(bmi);

                }
                else
                {
                    plotHighPassRCFilter();

                    bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents\Kunal\UE courses\EE-380\Project2_Filters\Project2_Filters\Image\HPRC.png"));
                    wbm = new WriteableBitmap(bmi);
                }
            }
            if (filter == 2)
            {
                plotBandPassFilter();

                bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents\Kunal\UE courses\EE-380\Project2_Filters\Project2_Filters\Image\BP.png"));
                wbm = new WriteableBitmap(bmi);

            }
            if (filter == 3)
            {
                plotBandRejectFilter();

                bmi = new BitmapImage(new Uri(@"C:\Users\kunmu\Documents\Kunal\UE courses\EE-380\Project2_Filters\Project2_Filters\Image\BR.png"));
                wbm = new WriteableBitmap(bmi);
            }

        }

        //The function allows u-micro, p-pico, n-nano, f-femto, m-mili,k,K-kilo and M-mega
        //character to be used for R,L,C values
        private double getRLCValue(int choice)
        {
            string num = "";
            if (choice == 0)
            {
                if (txtbxR.Text.Contains("K") || txtbxR.Text.Contains("k"))
                {
                    num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, 3);
                }
                if (txtbxR.Text.Contains("M"))
                {
                    num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, 6);
                }
                if (txtbxR.Text.Contains("m"))
                {
                    num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, -3);
                }
                if (txtbxR.Text.Contains("u"))
                {
                    num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, -6);
                }
                if (txtbxR.Text.Contains("n"))
                {
                    num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, -9);
                }
                if (txtbxR.Text.Contains("p"))
                {
                    num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, -12);
                }
                if (txtbxR.Text.Contains("f"))
                {
                    num = txtbxR.Text.Substring(0, txtbxR.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, -15);
                }
                else
                {
                    return Convert.ToDouble(txtbxR.Text);
                }
            }
            if(choice == 1)
            {
                if (txtbxC.Text.Contains("M"))
                {
                    num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, 6);
                }
                if (txtbxC.Text.Contains("K") || txtbxC.Text.Contains("k"))
                {
                    num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, 3);
                }
                if (txtbxC.Text.Contains("m"))
                {
                    num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, -3);
                }
                if (txtbxC.Text.Contains("u"))
                {
                    num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, -6);
                }
                if (txtbxC.Text.Contains("n"))
                {
                    num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, -9);
                }
                if (txtbxC.Text.Contains("p"))
                {
                    num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, -12);
                }
                if (txtbxC.Text.Contains("n"))
                {
                    num = txtbxC.Text.Substring(0, txtbxC.Text.Length - 1);
                    return Convert.ToDouble(num) * Math.Pow(10, -15);
                }
                else
                {
                    return Convert.ToDouble(txtbxC.Text);
                }
            }
            if(choice == 2)
            {
                if (txtbxL.Text.Contains("M"))
                {
                    num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
                    Console.WriteLine(num);
                    return Convert.ToDouble(num) * Math.Pow(10, 6);
                }
                if (txtbxL.Text.Contains("K") || txtbxL.Text.Contains("k"))
                {
                    num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
                    Console.WriteLine(num);
                    return Convert.ToDouble(num) * Math.Pow(10, 3);
                }
                if (txtbxL.Text.Contains("m"))
                {
                    num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
                    Console.WriteLine(num);
                    return Convert.ToDouble(num) * Math.Pow(10, -3);
                }
                if (txtbxL.Text.Contains("u"))
                {
                    num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
                    Console.WriteLine(num);
                    return Convert.ToDouble(num) * Math.Pow(10, -6);
                }
                if (txtbxL.Text.Contains("n"))
                {
                    num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
                    Console.WriteLine(num);
                    return Convert.ToDouble(num) * Math.Pow(10, -9);
                }
                if (txtbxL.Text.Contains("p"))
                {
                    num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
                    Console.WriteLine(num);
                    return Convert.ToDouble(num) * Math.Pow(10, -12);
                }
                if (txtbxL.Text.Contains("f"))
                {
                    num = txtbxL.Text.Substring(0, txtbxL.Text.Length - 1);
                    Console.WriteLine(num);
                    return Convert.ToDouble(num) * Math.Pow(10, -15);
                }
                else
                {
                    return Convert.ToDouble(txtbxL.Text);
                }
            }

            return 0;
        }

        //The function calculates the number of sig.fig for a number given
        private int magnitudeQuantificationofValue( double number)
        {
            int counter = 0;

            if (number > 1)
            {
                while (number > 10)
                {
                    number /= 10;
                    counter++;
                }
            }

            return counter;
        }

        //The function maximum and the minimum w value for the transfer function
        //for low and high pass filter
        private void setWCAxisMinandMax(double wTemp)
        {
            int x = magnitudeQuantificationofValue(wTemp);

            chtAmp.ChartAreas[0].AxisX.Minimum = Math.Pow(10, x - 2);
            chtPha.ChartAreas[0].AxisX.Minimum = Math.Pow(10, x - 2);

            chtAmp.ChartAreas[0].AxisX.Maximum = Math.Pow(10, x + 2);
            chtPha.ChartAreas[0].AxisX.Maximum = Math.Pow(10, x + 2);

            nI = magnitudeQuantificationofValue(chtAmp.ChartAreas[0].AxisX.Minimum);
            nMax = magnitudeQuantificationofValue(chtAmp.ChartAreas[0].AxisX.Maximum);
        }

        //The function maximum and the minimum w value for the transfer function
        //for band pass and stop filter
        private void setW0AxisMinandMax()
        {
            if (txtbxLCF.Text == "0")
            {
                w0 = 1 / Math.Sqrt(l * c);
                B = r / l;

                double rTemp = Math.Sqrt(Math.Pow((B / 2), 2) + Math.Pow(w0, 2));

                wl = (-(B / 2) + rTemp);
                wu = (+(B / 2) + rTemp);

                txtbxLCF.Text = (wl).ToString("E3");
                txtbxHCF.Text = (wu).ToString("E3");

            }
            else
            {
                wl = Convert.ToDouble(txtbxLCF.Text);
                wu = Convert.ToDouble(txtbxHCF.Text);

                w0 = Math.Sqrt(wl * wu);
                B = wu - wl;

                if (txtbxR.Text != "0")
                {
                    txtbxL.Text = (getRLCValue(0) / B).ToString( "E3");
                    txtbxC.Text = (1 / (getRLCValue(2) * Math.Pow(w0, 2))).ToString("E3"); ;
                }
                else if (txtbxC.Text != "0")
                {
                    txtbxL.Text = (1 / Math.Pow(w0, 2) * getRLCValue(1)).ToString("E3");
                    txtbxR.Text = (getRLCValue(2) * B).ToString( "E3");
                }
                else
                {
                    txtbxC.Text = (1 / Math.Pow(w0, 2) * getRLCValue(2)).ToString("E3");
                    txtbxR.Text = (getRLCValue(2) * B).ToString( "E3");
                }

            }

            int xMin = magnitudeQuantificationofValue(wl);
            int xMax = magnitudeQuantificationofValue(wu);

            chtAmp.ChartAreas[0].AxisX.Minimum = Math.Pow(10, xMin - 3);
            chtPha.ChartAreas[0].AxisX.Minimum = Math.Pow(10, xMin - 3);

            chtAmp.ChartAreas[0].AxisX.Maximum = Math.Pow(10, xMax + 3);
            chtPha.ChartAreas[0].AxisX.Maximum = Math.Pow(10, xMax + 3);

            nI = magnitudeQuantificationofValue(chtAmp.ChartAreas[0].AxisX.Minimum);
            nMax = magnitudeQuantificationofValue(chtAmp.ChartAreas[0].AxisX.Maximum);
```

```

        }

        //The function clears the series up and sets it up for data population
        private void initializeSeries()
        {
            //clear the chart area
            chtAmp.ChartAreas.Clear();
            chtAmp.ChartAreas.Add("Default");
            chtPha.ChartAreas.Clear();
            chtPha.ChartAreas.Add("Default");

            chtAmp.Width = 450;
            chtAmp.Height = 270;
            chtPha.Width = 450;
            chtPha.Height = 270;

            chtAmp.Location = new System.Drawing.Point(0, 0);
            chtPha.Location = new System.Drawing.Point(0, 0);

            //w for |H|
            wASeries = new Series();
            //w for <H
            wPSeries = new Series();

            wASeries.ChartType = SeriesChartType.Line;
            wPSeries.ChartType = SeriesChartType.Line;
        }

        //The function fills up the graph after series is calculated
        //as well as formats the graph
        private void fillSeriesGraph()
        {
            chtAmp.Series.Clear();
            chtAmp.Series.Add(wASeries);
            chtAmp.ChartAreas[0].AxisX.IsLogarithmic =  true;
            chtAmp.ChartAreas[0].AxisX.LogarithmBase = 10;
            chtAmp.ChartAreas[0].AxisX.Title =  "Frequency";
            chtAmp.ChartAreas[0].AxisX.LabelStyle.Format =  "E3";
            chtAmp.ChartAreas[0].AxisY.Title =  "|H|";
            chtAmp.ChartAreas[0].AxisY.LabelStyle.Format =  "{0.00}";



            chtPha.Series.Clear();
            chtPha.Series.Add(wPSeries);
            chtPha.ChartAreas[0].AxisX.IsLogarithmic =  true;
            chtPha.ChartAreas[0].AxisX.LogarithmBase = 10;
            chtPha.ChartAreas[0].AxisX.Title =  "Frequency";
            chtPha.ChartAreas[0].AxisX.LabelStyle.Format = "E3";
            chtPha.ChartAreas[0].AxisY.Title = "<H";
            chtPha.ChartAreas[0].AxisY.LabelStyle.Format = "{0.00}";
        }

        //The function calculates the transfer function for
        //low pass RL filter
        private void plotLowPassRLFilter()
        {
            initializeSeries();

            //calculate the cutoff frequency
            wc = r / l;

            if (txtbxCF.Text == "0")
            {
                txtbxCF.Text = wc.ToString("E3");
            }
            else
            {
                wc = Convert.ToDouble(txtbxCF.Text);

                if(txtbxR.Text == "0")
                {
                    txtbxR.Text = (wc * getRLCValue(2)).ToString("E3");
                }
                else
                {
                    txtbxL.Text = (getRLCValue(0) / wc).ToString("E3");
                }
            }

            setWCAxisMinandMax(wc);

            //generate the graph points |H| and <H
            for (int n = nI; n <= nMax; n++)
            {
                for (int m = 1; m < 10; m++)
                {
                    w = m * Math.Pow(10, n);

                    Mh = 20 * Math.Log10(1 / Math.Sqrt(1 + Math.Pow((w / wc), 2)));
                    Ah = (-Math.Atan2(w, wc)) * (180 / Math.PI);

                    wASeries.Points.AddXY(w, Mh);
                    wPSeries.Points.AddXY(w, Ah);

                }

            }

            fillSeriesGraph();
        }

        //The function calculates the transfer function for
        //low pass RC filter
        private void plotLowPassRCFilter()
        {
            initializeSeries();

            wc = 1 / (r * c);

            if (txtbxCF.Text == "0")
            {
                txtbxCF.Text = wc.ToString("E3");
            }
            else
            {
                wc = Convert.ToDouble(txtbxCF.Text);

                if (txtbxR.Text == "0")
                {
                    txtbxR.Text = (1 / wc * getRLCValue(1)).ToString("E3");
                }
                else
                {
                    txtbxC.Text = (1 / wc * getRLCValue(0)).ToString("E3");
                }
            }

            setWCAxisMinandMax(wc);

            //generate the graph points |H| and <H
            for (int n = nI; n <= nMax; n++)
            {
                for (int m = 1; m < 10; m++)
                {
                    w = m * Math.Pow(10, n);

                    Mh = 20 * Math.Log10(1 / Math.Sqrt(1 + Math.Pow((w / wc), 2)));
                    Ah = (-Math.Atan2(w, wc)) * (180 / Math.PI);


                    wASeries.Points.AddXY(w, Mh);
                    wPSeries.Points.AddXY(w, Ah);
                }
            }

            fillSeriesGraph();
        }

        //The function calculates the transfer function for
        //high pass RL filter
        private void plotHighPassRLFilter()
        {
            initializeSeries();

            wc = r / l;

            if (txtbxCF.Text == "0")
            {
                txtbxCF.Text = wc.ToString( "E3");
            }
            else
            {
                wc = Convert.ToDouble(txtbxCF.Text);

                if (txtbxR.Text == "0")
                {
                    txtbxR.Text = (wc * getRLCValue(2)).ToString( "E3");
                }
                else
                {
                    txtbxL.Text = (getRLCValue(0) / wc).ToString( "E3");
                }
            }

            setWCAxisMinandMax(wc);

            //generate the graph points |H| and <H
            for (int n = nI; n <= nMax; n++)
            {
                for (int m = 1; m < 10; m++)
                {
                    w = m * Math.Pow(10, n);

                    Mh = 20 * Math.Log10(1 / Math.Sqrt(1 + Math.Pow((wc / w), 2)));
                    Ah = (Math.Atan2(wc, w)) * (180 / Math.PI);


                    wASeries.Points.AddXY(w, Mh);
                    wPSeries.Points.AddXY(w, Ah);
                }
            }

            fillSeriesGraph();
        }

        //The function calculates the transfer function for
        //high pass RC filter
        private void plotHighPassRCFilter()
        {
            initializeSeries();

            wc = 1 / (r * c);

            if (txtbxCF.Text == "0")
            {
                txtbxCF.Text = wc.ToString("E3");
            }
            else
            {
                wc = Convert.ToDouble(txtbxCF.Text);

                if (txtbxR.Text == "0")
                {
                    txtbxR.Text = (1 / wc * getRLCValue(1)).ToString("E3");
                }
                else
                {
                    txtbxC.Text = (1 / wc * getRLCValue(0)).ToString("E3");
                }
            }

            setWCAxisMinandMax(wc);

            //generate the graph points |H| and <H
            for (int n = nI; n <= nMax; n++)
            {
                for (int m = 1; m < 10; m++)
                {
                    w = m * Math.Pow(10, n);

                    Mh = 20 * Math.Log10(1 / Math.Sqrt(1 + Math.Pow((wc / w), 2)));
                    Ah = (Math.Atan2(wc, w)) * (180 / Math.PI);


                    wASeries.Points.AddXY(w, Mh);
                    wPSeries.Points.AddXY(w, Ah);
                }

            }

            fillSeriesGraph();
        }

        //The function calculates the transfer function for
        //band pass filter
        private void plotBandPassFilter()
        {
            initializeSeries();

            setW0AxisMinandMax();

            for (int n = nI; n <= nMax; n++)
            {
                for (int m = 1; m < 10; m++)
                {
                    w = m * Math.Pow(10, n);

                    tempVal = (Math.Pow(w0, 2)) - (Math.Pow(w, 2));

                    Mh = 20 * Math.Log10(Math.Sqrt(Math.Pow(B * w, 2)) / Math.Sqrt(Math.Pow(tempVal, 2) + Math.Pow(B * w, 2)));
                    Ah = (Math.Atan2(B * w, 0) - Math.Atan2(B * w, tempVal)) * (180 / Math.PI);

                    wASeries.Points.AddXY(w, Mh);
                    wPSeries.Points.AddXY(w, Ah);
                }

            }

            fillSeriesGraph();
        }

        //The function calculates the transfer function for
        //band stop filter
        private void plotBandRejectFilter()
        {
            initializeSeries();

            setW0AxisMinandMax();

            for (int n = nI; n <= nMax; n++)
            {
                for (int m = 1; m < 10; m++)
                {
                    w = m * Math.Pow(10, n);

                    tempVal = (Math.Pow(w0, 2)) - (Math.Pow(w, 2));

                    if (Math.Sqrt(Math.Pow(tempVal, 2))  != 0)
                    {
                        Mh = 20 * Math.Log10(Math.Sqrt(Math.Pow(tempVal, 2)) / Math.Sqrt(Math.Pow(tempVal, 2) + Math.Pow(B * w, 2)));
                    }

                    Ah = (Math.Atan2(0, tempVal) - Math.Atan2(B * w, tempVal)) * (180 / Math.PI);

                    wASeries.Points.AddXY(w, Mh);
                    wPSeries.Points.AddXY(w, Ah);
                }

            }

            fillSeriesGraph();
        }

    }
}
```