A short introduction on using Wave Digital Filters in circuit emulation

November 9, 2016

This notebook is a work in progress. Notes, comments and other suggestions should kindly be sent to olafur.bogason@mail.mcgill.ca.

Wave Digital Filter (WDF) theory is over fifty years old []. Originally it was used to digitize RCL ladder circuits [], that is circuits that can be seperated into a cascade of series and parallel connections. WDFs can be seen as a Finite Difference Scheme with some nice numerical properties []. With recent advances [2,3] Wave Digital Structures (more general than filters) allow an audio DSP designer to create algorithms that are *physically informed*. That is, the algorithms retain the underlying structure (topology) of the modelled circuit and first-order approximations of how circuit elements physically behave (lumped elements).

For each lumped element in the Kirchoff (K)-domain there exists a is a **port** in the Wave Digital domain. Similarly for series and parallel connections in the K-domain there exists series and parallel **adaptors** in the WD-domain. Many circuits [] cannot be split into a cascade of series and parallel adaptors and for such topologies we have **R-type** (Rigid) adaptors.

This notebook presents a derivation of Wave Digital (WD) representations of the more commonly used circuit elements that have been ported over from the Kirchoff, K-domain. Derivations of series and parallel adaptors follow.

Examples of how to implement WD-structures will be given by use of the programming language *Python* [?]. To keep things simple concepts from *Object Oriented Programming* are used to model individual ports (lumped elements in K-domain). Simple functions are used for adaptors to show how computation of the WD formalism is carried out. Instead of displaying all code at once it will be interleaved with the text. Consequently the concepts discussed are be put to practice straight away.

Few examples of WDF circuit simulations are given at the end of the notebook. First some auxillary code.

```
In [1]: %matplotlib inline

        import numpy as np
        import pandas as pd
        from scipy import signal
        from sympy import *
        import matplotlib.pyplot as plt

        FS = 96e3  # Sample frequency.
        N = 2**14  # Number of points to simulate.
```

```
input = np.zeros(N)
input[0] = 1  # Input is a delta function.
output = np.zeros(input.size)
steps  = np.arange(N)

def plot_freqz(x, title="Frequency response"):
    # Plot the frequency response of a signal x.
    w, h = signal.freqz(x, 1, 2048*2)
    H = 20 * np.log10(np.abs(h))
    f = w / (2 * np.pi) * FS
    plt.semilogx(f, H, label="WDF")
    plt.xlim([np.min(f), np.max(f)])
    plt.title(title)
    plt.xlabel('Frequency [Hz]')
    plt.ylabel('Magnitude [dB]')

def plot_ltspice_freqz(filename, title="Frequency response", out_label='V(v
    # Plot the frequency responce of a LTspice simulation containing
    # frequency and complex signal.
    def imag_to_mag(z):
        # Returns the maginude of an imaginary number.
        a, b = map(float, z.split(','))
        return 20*np.log10(np.sqrt(a*a + b*b))

    x = pd.read_csv(filename, delim_whitespace=True)
    x['H_dB'] = x[out_label].apply(imag_to_mag)

    f = np.array(x['Freq.'])
    H_db = np.array(x['H_dB'])
    plt.semilogx(f, H_db, label="LTspice")
```

1 WD-domain variables

One-port elements in electric circuits include electrical component such as resistors, capacitors and inductors. They also include ideal and resistive voltage and current sources along with short and open circuits. Many more one-port elements exist, but these are some of the elements that have already been ported over to the WD-domain.

We will carry out the derivation by borrowing concepts from circuit theory such as impedances and the Laplace transform. Another integral part of the derivation is the use of the bilinear transform [] to digitize reactive circuit elements []

$$s \leftarrow \frac{1}{2f_s}\frac{z-1}{z+1}$$

We map K-domain variables v and i (voltage and current) to WD-domain variables a and b (incident- and reflected wave) with the following linear transforms.

$$a = v + i \cdot R_p \quad b = v - i \cdot R_p$$

and back again

$$v = \frac{1}{2}(a+b) \quad i = \frac{1}{2R_p}(a-b)$$

R_p is called a **port resistance**. It is physically introduced into the system, meaning that in practice it can be set to any real value.

There is one thing to bear in mind during this derivation. For the resulting WDF structure to be computable we cannot have any delay-free loops inside it []. That means that for each one-port element we are looking for a function f that maps a -> b but b_t cannot depend on a_t. It can only on delayed versions of a (e.g. a_{t-1}) (1).

To deal with this issue there are two tools at our disposal. The free parameter, R_p, the port resistance. The other tool is to make one port of a single adaptor **reflection free**. That means that we choose the port resistance at that port so that the reflected wave does not depend directly on the incident wave at that same port. Note that the reflected wave can, and often will, depend on incident waves from other ports of the same adaptor.

Photo of K-domain and WD-domain ports and the mappings from v, i to a, bþ

```
In [2]: class WDFOnePort(object):
            def __init__(self):
                # Incident- and reflected wave variables.
                self.a, self.b = 0, 0

            def wave_to_voltage(self):
                voltage = (self.a + self.b) / 2
                return voltage
```

2 One port elements

In this section we start with the K-domain representaion of a one-port element and derive the WD-domain counterpart. In the derivation process we use the transformations between the K-domain and WD-domain extensively.

2.0.1 WDF Resistor

$$Z_R = R = \frac{v}{i} = \frac{\frac{1}{2}(a+b)}{\frac{1}{2R_p}(a-b)}$$

$$b = \frac{R - R_p}{R + R_p} a$$

Now for (1) to be fulfilled we need to have $R_p = R$. When $Z_R = 0$ then $b = 0$ for all resistor elements. In words that means that the reflected wave is always zero independent of the inciding wave! What does that mean? Will the resistors not contribute at all to the WDF structure?

Far from it. The port resistances contribute to the scattering of inciding waves. More about that later.

```
In [3]: class Resistor(WDFOnePort):
            def __init__(self, R):
                WDFOnePort.__init__(self)
```

```
        self.Rp = R

    def get_reflected_wave(self, a):
        self.a = a
        self.b = 0
        return self.b
```

2.0.2 WDF Capacitor

Now we move on to reactive elements. and describe the capacitor as a function of the Laplace variable s.

$$Z_C = \frac{1}{sC} = \frac{v}{i} = \frac{\frac{1}{2}(a+b)}{\frac{1}{2R_p}(a-b)}$$

$$b = Sa = \frac{1 - R_pCs}{1 + R_pCs}a$$

The bilinear transform is used to digitize the function S

$$b = Sa = \frac{\frac{1}{2f_s}\frac{z-1}{z+1}L - R_p}{\frac{1}{2f_s}\frac{z-1}{z+1}L + R_p}a = \frac{(1-2f_sCR_p) + (1+2f_sCR_p)z^{-1}}{(1+2f_sCR_p) + (1-2f_sCR_p)z^{-1}}a$$

The only way (1) be fulfilled is if $R_p = \frac{1}{2f_sC}$. For that value of R_p the reflected wave is $ b = z^{-1} a $, the reflected wave is simply a unit-delayed verion of the incident wave.

```
In [4]: class Capacitor(WDFOnePort):
            def __init__(self, C):
                WDFOnePort.__init__(self)
                self.Rp = 1 / (2 * FS * C)

            def get_reflected_wave(self, a):
                self.b = self.a
                self.a = a
                return self.b

            def set_incident_wave(self, a):
                self.a = a
```

2.0.3 WDF Inductor

Following the same approach as above we can easilly derive the WD-domain representation for an inductor.

$$Z_L = sL = \frac{v}{i} = \frac{\frac{1}{2}(a+b)}{\frac{1}{2R_p}(a-b)}$$

$$b = Sa = \frac{sL - R_p}{sL + R_p}a$$

Now we use the bilinear transform and digitize S.

$$b = Sa = \frac{\frac{1}{2f_s}\frac{z-1}{z+1}L - R_p}{\frac{1}{2f_s}\frac{z-1}{z+1}L + R_p}a = \frac{(2f_sL - R_p) - (2f_sL + R_p)z^{-1}}{(2f_sL + R_p) + (2f_sL - R_p)z^{-1}}a$$

Again we see that the only way (1) be fulfilled is if $R_p = 2f_sL$. Plugging that result into the equation above we get that $b = -z^{-1}a$, which means that the reflected wave is a unit-delayed inciding wave with it's phase inverted.

```
In [5]: class Inductor(WDFOnePort):
            def __init__(self, L):
                WDFOnePort.__init__(self)
                self.Rp = (2 * FS * L)

            def get_reflected_wave(self, a):
                self.b = self.a
                self.a = a
                return -self.b

            def set_incident_wave(self, a):
                self.a = a
```

2.0.4 Short circuit

A short circuit can most easily be seen as a resistor with no resistance, that is $R = 0$

$$b = \frac{R - R_p}{R + R_p}a = -a$$

which essentially means that all the inciding wave gets projected back through the port.

```
In [6]: class ShortCircuit(WDFOnePort):
            def __init__(self):
                WDFOnePort.__init__(self)

            def get_reflected_wave(self, a):
                self.a = a
                self.b = -a
                return self.b
```

2.0.5 Open circuit

A short circuit can most easily be seen as a resistor with infinite resitance, that is $R \to \infty$

$$b = \frac{R - R_p}{R + R_p}a = \frac{R/R_p - 1}{R/R_p + 1}a = a$$

The difference between a short- and open circuit is the additional minus sign. The sign in the short circuit case can be interpreted as a change in phase. Which is easy to visualize as a incident wave getting completely reflected at a port resembling an open circuit.

```
In [7]: class OpenCircuit(WDFOnePort):
            def __init__(self):
                WDFOnePort.__init__(self)

            def get_reflected_wave(self, a):
                self.a = a
                self.b = a
                return self.b
```

2.0.6 Ideal Voltage Source

Picture

We can get the relationship in the case of an ideal voltage source with voltage v_s directly from the definition of wave variables with respect to Kirchoff ones:

$$v = v_s = \frac{1}{2}(a+b)b = 2v_s - a$$

```
In [8]: class IdealVoltageSource(WDFOnePort):
            def __init__(self):
                WDFOnePort.__init__(self)

            def get_reflected_wave(self, a, vs=0):
                self.a = a
                self.b = 2*vs - a
                return self.b
```

2.0.7 Ideal Current Source

The same is possible for the ideal current source with current i_s

$$v = \frac{1}{2R_p}(a-b)b = a - 2R_p i_s$$

where R_p is the port resistance and must be matched, which is a concept that we will cover in the next section.

```
In [9]: class IdealCurrentSource(WDFOnePort):
            def __init__(self):
                WDFOnePort.__init__(self)

            def get_reflected_wave(self, a, i_s=0):
                self.a = a
                self.b = a - 2*R_p*i_s
                return self.b
```

2.0.8 Resistive Voltage Source

Now what if we have an ideal voltage source with voltage v_s connected in series with a resistor R? Then the Kirchoff voltage law gives us

$$v = v_s + Rib = 2v - a = 2(v_s + Ri) - a = 2v_s + 2R\frac{1}{2R_p}(a - b) - a$$

Now if we choose to have to have $R = R_p$ then we get

$$b = v_s$$

```
In [10]: class ResistiveVoltageSource(WDFOnePort):
             def __init__(self, R):
                 WDFOnePort.__init__(self)
                 self.Rp = R

             def get_reflected_wave(self, a, v_s=0):
                 self.a = a
                 self.b = v_s
                 return self.b
```

2.0.9 Resistive Current Source

What about a resistive current source? A resistive current source is essentially an ideal current source of magnitude i_s connected in parallel with a resistor R. The voltage over the resistor can be quantified by $v = (i - i_s)R$ where i is the current flowing into the port. We now have:

$$b = a - 2R_p i = a - 2R_p\frac{v}{R} + i_s R_p = a - \frac{R_p}{R}(a + b) + 2i_s R_p$$

Again if we choose to have $R = R_p$ then we get

$$b = Ri_s$$

```
In [11]: class ResistiveCurrentSource(WDFOnePort):
             def __init__(self, R):
                 WDFOnePort.__init__(self)
                 self.Rp = R

             def get_reflected_wave(self, a, i_s=0):
                 self.a = a
                 self.b = self.Rp*i_s
                 return self.b
```

2.0.10 Table of results and interpretation

Voltage waves. How does voltage over a capacitor, inductor and resistor behave? What is interesting about these results?

3 Adaptors for simple topologies

Now that we have a couple of one port elements laid out we cant start connecting them together. Historically circuits that have been emulated using WDFs have been connected in some seperable series and/or parallel conjunction. That is why we will start with deriving series and parallel adaptors.

3.0.1 Two-port simple adaptors

Now we will move to derive the relationship of reflected waves to incident waves of two-port series and parallel adaptors.

Series Two port series adaptors are described by the following two equations:

$$0 = v_1 + v_2 i_1 = i_2 = i_J$$

where i_J is the current flowing into the junction.

$$0 = v_1 + v_2 = \frac{1}{2}(a_1 + b_1) + \frac{1}{2}(a_2 + b_2) b_1 = -(a_2 + a_1) - b_2$$

$$0 = i_2 - i_1 = \frac{1}{2R_2}(a_2 - b_2) - \frac{1}{2R_1}(a_1 - b_1) a_2 - b_2 - \frac{R_2}{R_1}(a_1 = b_1)$$

$$b_2 = \frac{R_1 - R_2}{R_1 + R_2} a_2 - \frac{2R_2}{R_1 + R_2} a_1 = -a_1 + \gamma(a_1 + a_2)$$

where $\gamma = \frac{R_1 - R_2}{R_1 + R_2}$ as defined by Fettweis and used all over the place when scattering junctions appear. Now we can get the reflected wave at port 1.

$$b_1 = -a_2 + \gamma(a_1 + a_2) b_2 = -a_1 + \gamma(a_1 + a_2)$$

Parallel We can derive the scattering relationship for parallel adaptors in a similar fashion. Conversely to the series adaptor the currents in the parallel adaptor sum up to zero but the voltages are all equal.

$$0 = i_1 + i_2 v_1 = v_2 = v_J$$

where v_J is defined as the voltage at the junction. Now we can use the definition of the wave variables and derive the relationship of reflected waves to the incident ones:

$$0 = i_1 + i_2 = \frac{1}{2R_1}(a_1 - b_1) + \frac{1}{2R_2}(a_2 - b_2)$$

$$0 = R_2(a_1 - b_1) + R_1(a_2 - b_2) b_1 = a_1 + \frac{R_1}{R_2} a_2 - \frac{R_1}{R_2} b_2$$

And using equation ?? we can continue the derivation

$$0 = v_2 - v_1 = \frac{1}{2}(a_2 + b_2) - \frac{1}{2}(a_1 + b_1) = -a_1 + a_2 \frac{R_2 - R_1}{2R_2} + b_2 \frac{R_2 + R_1}{2R_2} b_2 \frac{R_2 + R_1}{2R_2} = a_1 - a_2 \frac{R_2 - R_1}{2R_2}$$

$$b_2 = \frac{2R_2}{R_1 + R_2} a_1 - \frac{-R_1 + R_2}{R_1 + R_2} = a_1 + \gamma(a_2 - a_1)$$

$$b_1 = a_1 + \frac{R_1}{R_2}a_2 - \frac{R_1}{R_2}b_2 = a_1(1 - \frac{2R_1}{R_1 + R_2}) + a_2(\frac{R_1}{R_2} - \frac{R_1}{R_2}\frac{R_1 - R_2}{R_1 + R_2}) = a_2 + \gamma(a_2 - a_1)$$

$$b_1 = a_2 + \gamma(a_2 - a_1) b_2 = a_1 + \gamma(a_2 - a_1)$$

Putting it together in vector format further shows us that we have a scattering matrix (rotation matrix + multiplication) in 2D space. In the case of a parallel adaptor we have:

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} -\gamma & 1+\gamma \\ 1-\gamma & \gamma \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

And for the series adaptor we have:

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} \gamma & 1-\gamma \\ 1-\gamma & \gamma \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

```
In [12]: def parallel_adaptor2(a1, Rp1, a2, Rp2):
             gamma = (Rp1 - Rp2) / (Rp1 + Rp2)
             S = np.array(((-gamma, 1+gamma), (1-gamma, gamma)))
             b1, b2 = np.dot(S, np.array((a1, a2)))

             return b1, b2

         def series_adaptor2(a1, Rp1, a2, Rp2):
             gamma = (Rp1 - Rp2) / (Rp1 + Rp2)
             S = np.array(((gamma, 1-gamma), (1-gamma, gamma)))
             b1, b2 = np.dot(S, np.array((a1, a2)))

             return b1, b2
```

3.0.2 N-port adaptors

Although N-port simple adaptors can always be broken up into 2- and 3-port adaptors we will still carry out the derivation for a variable N-port adaptors (because it isn't really any harder and we all love to generalize, right?).

Series As in the case of a 2-port series adaptor the adaptor is described by voltages and currents at each port.

$$0 = v_1 + v_2 + \cdots + v_N i_1 = i_2 = \cdots = i_N$$

So now we can start working…

$$0 = v_1 + v_2 + \cdots + v_N = (a_1 - R_1 i_1) + (a_2 - R_2 i_2) + \cdots + (a_N - R_N i_N) a_1 + a_2 + \cdots + a_N - i(R_1 + R_2 + \cdots + R_N) = 0$$

$$i = i_i = \frac{1}{2R_i}(a_i - b_i) = \frac{1}{R_1 + R_2 + \cdots + R_N}(a_1 + a_2 + \cdots + a_N) b_i = a_i - \frac{2R_i}{R_1 + R_2 + \cdots + R_N}(a_1 + a_2 + \cdots + a_N)$$

Parallel To save space and simplify calculations I put $G_i = \frac{1}{R_i}$

$$0 = i_1 + i_2 + \cdots + i_N v_1 = v_2 = \cdots = v_N = v$$

$$0 = i_1 + i_2 + \cdots + i_N = G_1(a_1 - v) + G_2(a_2 - v) + \cdots + G_N(a_N - v)$$

$$v = v_i = \frac{1}{2}(a_i + b_i) = \frac{1}{G_1 + G_2 + \cdots + G_N}(G_1 a_1 + G_2 a_2 + \cdots + G_N a_N)$$

$$b_i = \frac{2}{G_1 + G_2 + \cdots + G_N}(G_1 a_1 + G_2 a_2 + \cdots + G_N a_N) - a_i = (\gamma_1 a_1 + \gamma_2 a_2 + \cdots + \gamma_N a_N) - a_i$$

where γ_i is traditionally defined as

$$\gamma_i = \frac{2 \cdot G_i}{G_1 + G_2 + \cdots + G_N}$$

```
In [13]: def series_adaptor(A, R):
             Rtot = np.sum(R)
             Atot = np.sum(A)
             return [a - 2*r / Rtot * Atot for a, r in zip(A, R)]

         def parallel_adaptor(A, R):
             G = [1/r for r in R]
             Gtot = np.sum(G)
             Gamma = [2*g / Gtot for g in G]
             aDotGamma = np.dot(A, Gamma)
             return [aDotGamma - a for a in A]
```

4 On computability

Because we want our structures to be computable we cannot have any delay-free loops residing inside of them. That means that if we are to connect two adaptors together, or connect an element without its port resistance set, we are going to have to do something about it.

One way is to *match* the port resistance of a port that has a delay-free relationship between its incident wave and reflected one. That way, due to the series/parallel equations that we have derived above, the reflected wave of that given port will not depend on the inc ident wave at all.

Usually the way computalibity is ensured is to make up a SPQR tree of the circuit. SQPQ tree is an extension of the Binary Computation Tree as it allows for so called R-type adaptors which we will cover later. To see how to set-up a circuit in such a tree, please follow on to the next section.

5 Examples

Now we will use the basic theory that we've learned and compute the magnitude response of some simple circuits. The code will be very generic and not make use of any object oriented concepts. Parties interested in seeing an OO approach for WDFs should consider rt-wdf.

5.1 Example - Resistive voltage source connected in series with RC

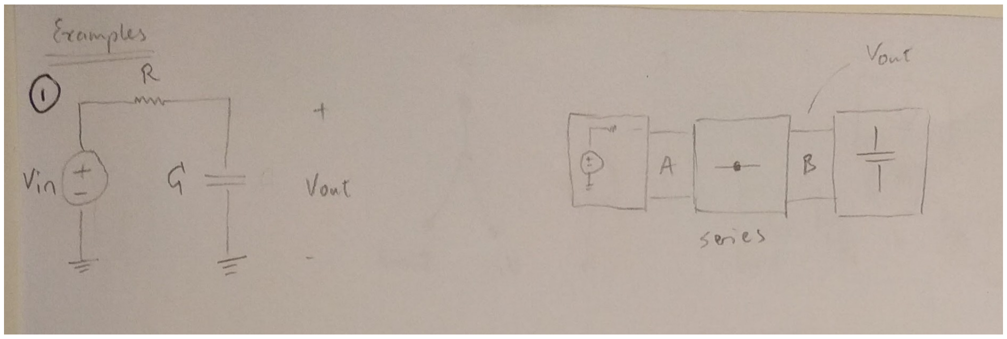
5.1.1 Reference circuit

5.1.2 WDF derivation

5.1.3 SPQR tree

5.1.4 Audio examples, we need them for fun!

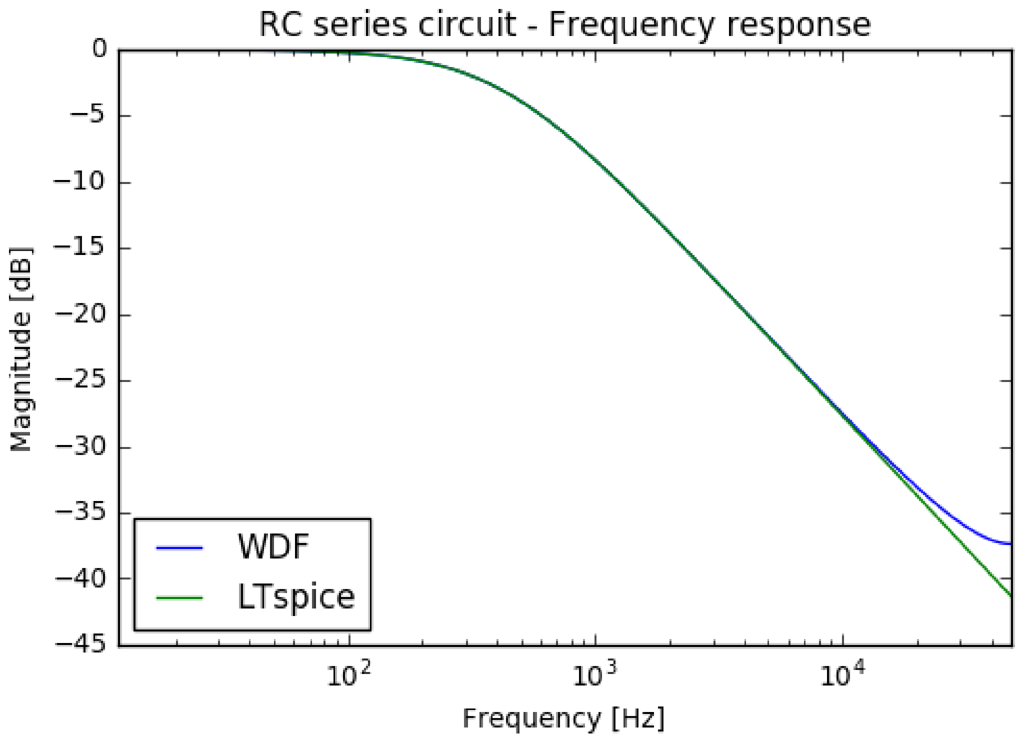
5.1.5 WDF derivation



WDF derivation for example 1

```
In [14]: # One-port parameters.
         V1 = ResistiveVoltageSource(1)  # 1 Ohm resistive volt. source
         C1 = Capacitor(3.5e-5)          # 0.35 uF capacitor.
         R1 = Resistor(10)               # 10 Ohm resistor.

         # Adaptor parameters.
         Rp1, Rp2, Rp3 = C1.Rp, R1.Rp, V1.Rp

         # Simulation loop.
         b1, b2, b3 = 0, 0, 0
         for i in steps:
             # 1. Gather inputs and scatter them.
             a1 = C1.get_reflected_wave(b1)
             a2 = R1.get_reflected_wave(0)
             a3 = V1.get_reflected_wave(0, input[i])  # Read input signal off volta

             a  = (a1, a2, a3)
             Rp = (Rp1, Rp2, Rp3)
             b1, b2, b3 = series_adaptor(a, Rp)

             # 2. Gather outputs.
             output[i] = C1.wave_to_voltage()  # Output is voltage over C1.
             C1.set_incident_wave(b1)          # Store new input inside Capacitor.

         # Plot frequency response of the WDF simulation.
```

```
        plot_freqz(output, title="RC series circuit - Frequency response")

        # Plot frequency response from LTspice
        plot_ltspice_freqz("data/ex01.txt", title="Frequency response", out_label=

        # Show the Frequency response
        plt.legend(loc=3)
        plt.show()
```



5.1.6 Audio examples

5.2 Example - Resistive voltage source connected to RC, RL branches

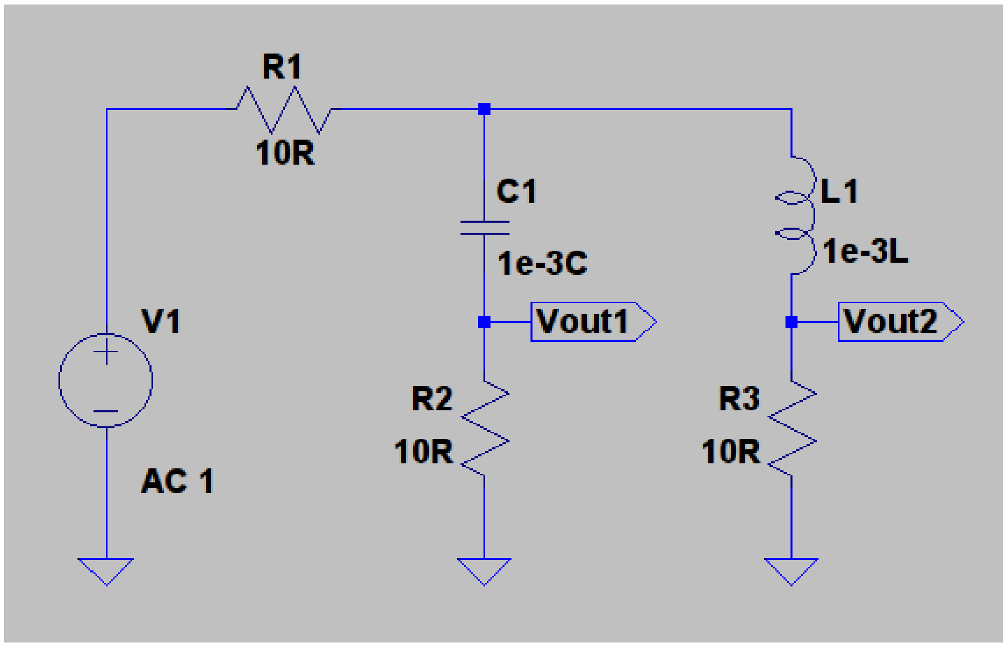
5.2.1 Reference circuit

5.2.2 WDF derivation

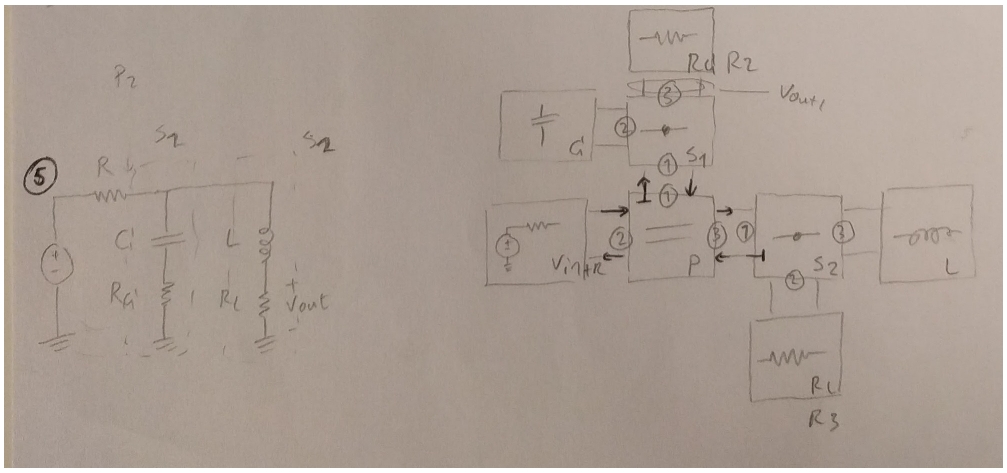
5.2.3 SPQR tree

5.2.4 WDF derivation

```
In [15]: # Two outputs.
         output_vout1 = np.zeros(input.size)
         output_vout2 = np.zeros(input.size)
```

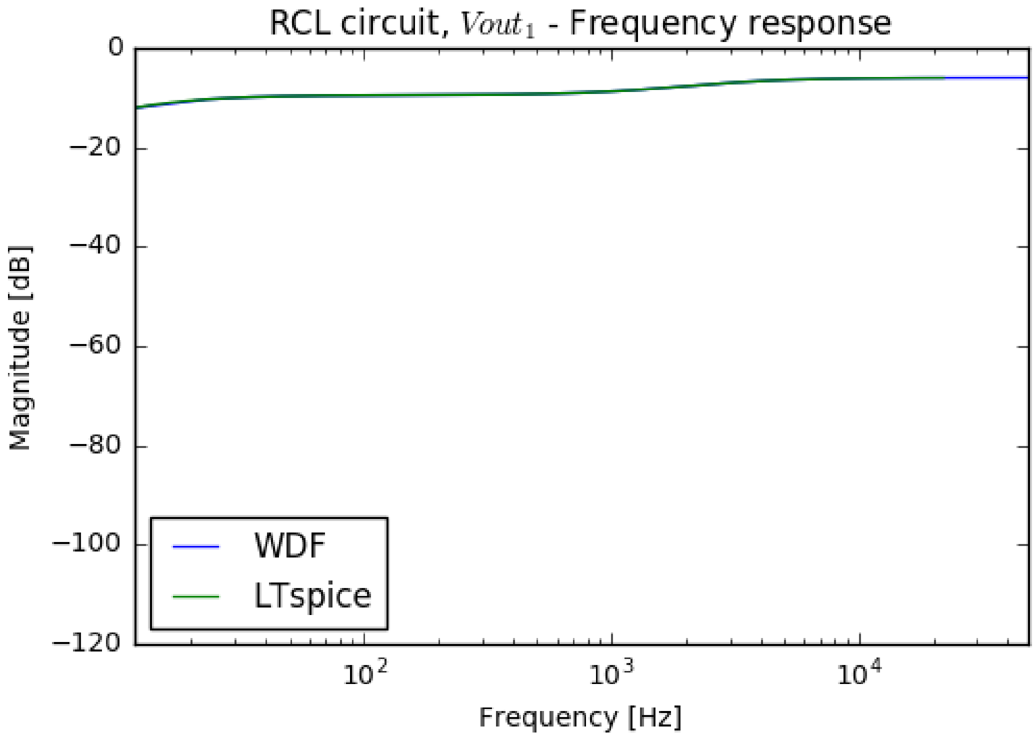


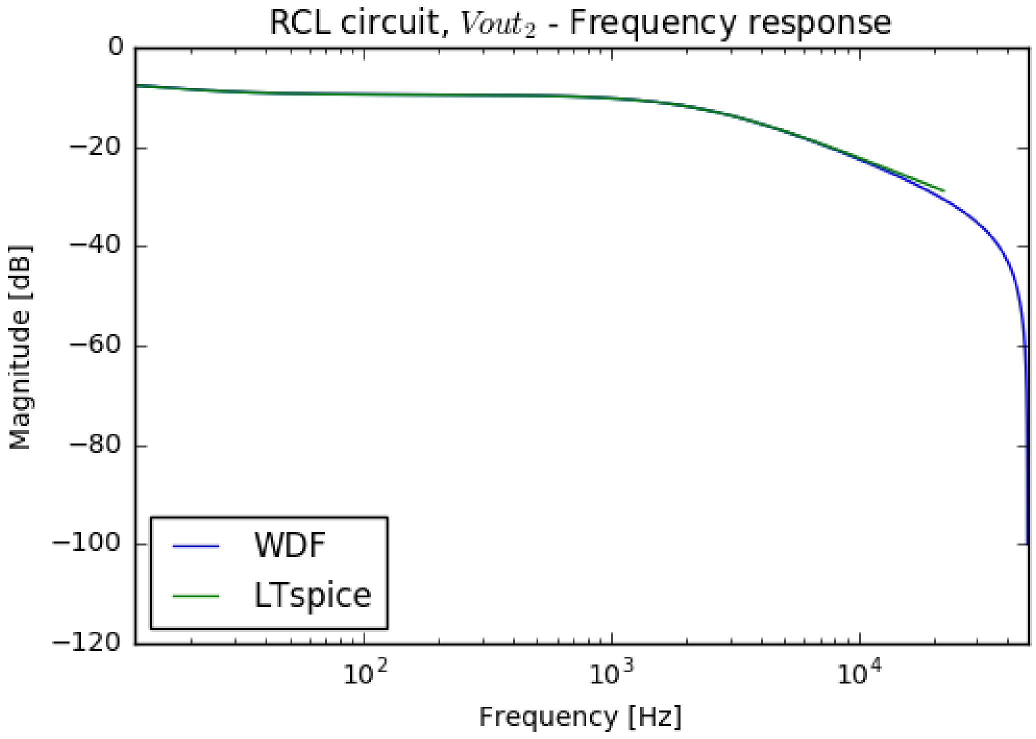
Reference circuit for example 1



WDF derivation for example 1

```
V1 = ResistiveVoltageSource(1)  # 1 Ohm resistive volt. source
C1 = Capacitor(3.5e-5)          # 0.35 uF capacitor.
R1 = Resistor(10)               # 10 Ohm resistor.


# One-port parameters.
V1 = ResistiveVoltageSource(10)  # Voltage source with 10 Ohm series resis
R2 = Resistor(10)     # 10 Ohm resistor.
R3 = Resistor(10)     # 10 Ohm resistor.
L1 = Inductor(1e-3)   # 1 mH Inductor.
C1 = Capacitor(1e-3)  # 1 mF Capacitor.

## Adaptor parameters.
# Series Port 2.
Rs22 = R3.Rp
Rs23 = L1.Rp
Rs21 = Rs22 + Rs23  # Matched port.
# Parallel Port.
Rp2 = V1.Rp
Rp3 = Rs21
Rp1 = 1 / (1/Rp2 + 1/Rp3)  # Matched port.
# Series Port 1.
Rs11 = Rp1
Rs12 = C1.Rp
Rs13 = R2.Rp

# Simulation loop.
ap1,  ap2,  ap3  = 0, 0, 0
as11, as12, as13 = 0, 0, 0
as21, as22, as23 = 0, 0, 0
bs11, bs12, bs13 = 0, 0, 0
bs21, bs22, bs23 = 0, 0, 0
bp3 = 0
for i in steps:
    # 1. Wave up.
    as22 = R3.get_reflected_wave(bs22)
    as23 = L1.get_reflected_wave(bs23)

    as2 = (as21, as22, as23)
    Rs2 = (Rs21, Rs22, Rs23)
    bs21, bs22, bs23 = series_adaptor(as2, Rs2)

    ap2 = V1.get_reflected_wave(bp3, input[i]) # Read signal from voltage
    ap3 = bs21

    ap = (ap1, ap2, ap3)
    Rp = (Rp1, Rp2, Rp3)
    bp1, bp2, bp3 = parallel_adaptor(ap, Rp)
    as11 = bp1
    as12 = C1.get_reflected_wave(bs12)
    as13 = R2.get_reflected_wave(bs13)

    as1 = (as11, as12, as13)
    Rs1 = (Rs11, Rs12, Rs13)
    bs11, bs12, bs13 = series_adaptor(as1, Rs1)

    # 2. Wave down.
    ap1 = bs11
    ap = (ap1, ap2, ap3)
    Rp = (Rp1, Rp2, Rp3)
    bp1, bp2, bp3 = parallel_adaptor(ap, Rp)

    as21 = bp3
    as2 = (as21, as22, as23)
    Rs2 = (Rs21, Rs22, Rs23)
    bs21, bs22, bs23 = series_adaptor(as2, Rs2)

    output_vout1[i] = R2.wave_to_voltage()  # Output is voltage through R
    output_vout2[i] = R3.wave_to_voltage()  # Output is voltage through R

    C1.set_incident_wave(bs12)
    L1.set_incident_wave(bs23)

# Plot frequency response of the WDF simulation.
plot_freqz(output_vout1, title="RCL circuit, $Vout_1$ - Frequency response

# Plot frequency response from LTspice
plot_ltspice_freqz("data/ex03_vout1.txt", title="Frequency response", out_

# Show the Frequency responses
plt.legend(loc=0)  # Find best location of legend box.
plt.show()

# Plot frequency response of the WDF simulation.
plot_freqz(output_vout2, title="RCL circuit, $Vout_2$ - Frequency response

# Plot frequency response from LTspice
plot_ltspice_freqz("data/ex03_vout2.txt", title="Frequency response", out_

# Show the Frequency responses
plt.legend(loc=0)  # Find best location of legend box.
plt.show()
```


RCL circuit, $Vout_1$ - Frequency response
Magnitude [dB]
Frequency [Hz]
0
−20
−40
−60
−80
−100
−120
10^2
10^3
10^4
WDF
LTspice


RCL circuit, $Vout_2$ - Frequency response
Magnitude [dB]
Frequency [Hz]
0
−20
−40
−60
−80
−100
−120
10^2
10^3
10^4
WDF
LTspice

6 More complicated topologies

Simple R-type adaptors derived using MNA. Show how three port parallel adaptor can be defined
Move onto more complicated topologies that **need** MNA + R-type adaptors.

7 Example - LP Sallen-Key filter

7.0.1 Reference circuit

7.0.2 WDF derivation

7.0.3 SPQR tree

8 Wave Digital Filter Adaptors for Arbitrary Topologies and Multiport Linear Elements

8.1 by Kurt James Werner, Julius O. Smith III, Jonathan S. Abe

Link to paper.

8.2 Main topics

Derive scattering matrix for arbitrary circuit topology using Modified Nodal Analysis in the W-domain.

8.3 Derivation of scattering matrix

We want to find a matrix S so that

$$b = Sa$$

For each port on a R-type adaptor we place a Thevenin equivalent circuit. We then apply the MNA method (circuit element stamps) in the K-domain and finally transfer that into the W-domain.

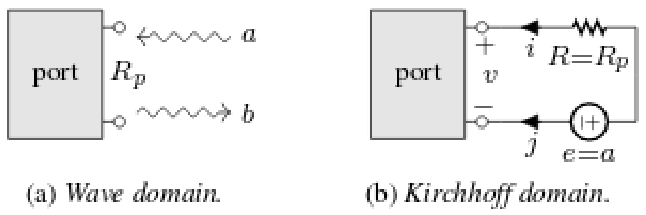


Figure 1: *Instantaneous Thévenin port equivalent.*

WDF derivation for example 1

R is the resistor value in the Thévenin equivalent circuit, R_p the WDF adaptor port resistance, i is the current going into the port and j is the source current.

Now we set $R = R_p, e = a, i = -j$ and remember the folowing equations that define the relationship between K-domain and the W-domain.

$$a = v + R_p i, b = v - R_p i$$

and use these two equations to eliminate v and i:

$$b = a + 2R_p j$$

$$\underbrace{\begin{bmatrix} \mathbf{Y} & \mathbf{A} \\ \mathbf{B} & \mathbf{D} \end{bmatrix}}_{\text{MNA matrix } \mathbf{X}} \begin{bmatrix} \boldsymbol{v}_n \\ \boldsymbol{j} \end{bmatrix} = \begin{bmatrix} \boldsymbol{i}_s \\ \boldsymbol{e} \end{bmatrix}$$

WDF derivation for example 1

Then all we need to do is get j out of this set of equations:

$$\boldsymbol{j} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{X}^{-1} \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix}^{\top} \boldsymbol{e} \,.$$

WDF derivation for example 1

Now we have the solution to the equation \$ b = S a \$ where

$$\mathbf{S} = \mathbf{I} + 2 \begin{bmatrix} \mathbf{0} & \mathbf{R} \end{bmatrix} \mathbf{X}^{-1} \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix}^{\top}$$

WDF derivation for example 1

8.4 Method applied to a previous example

9 Op amps

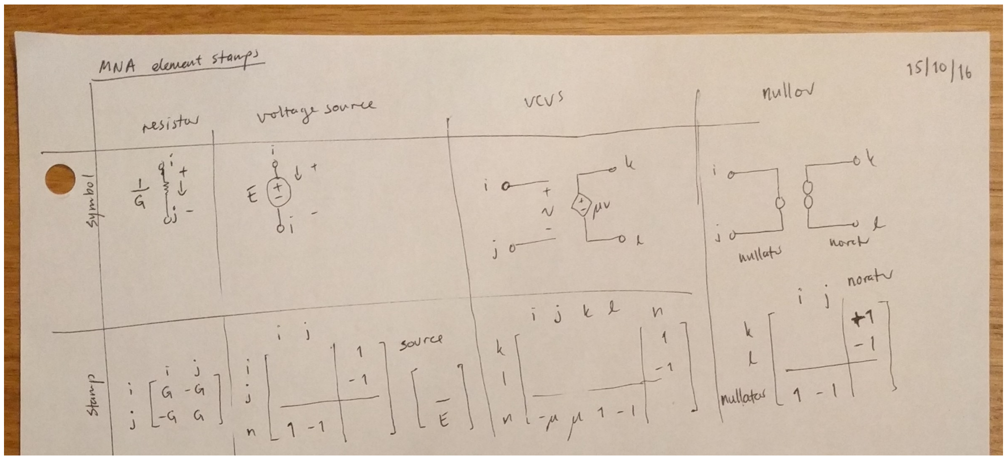
Nullator and macromodel discussion.
Examples: Buffer and Sallen-Key filter

10 Diodes

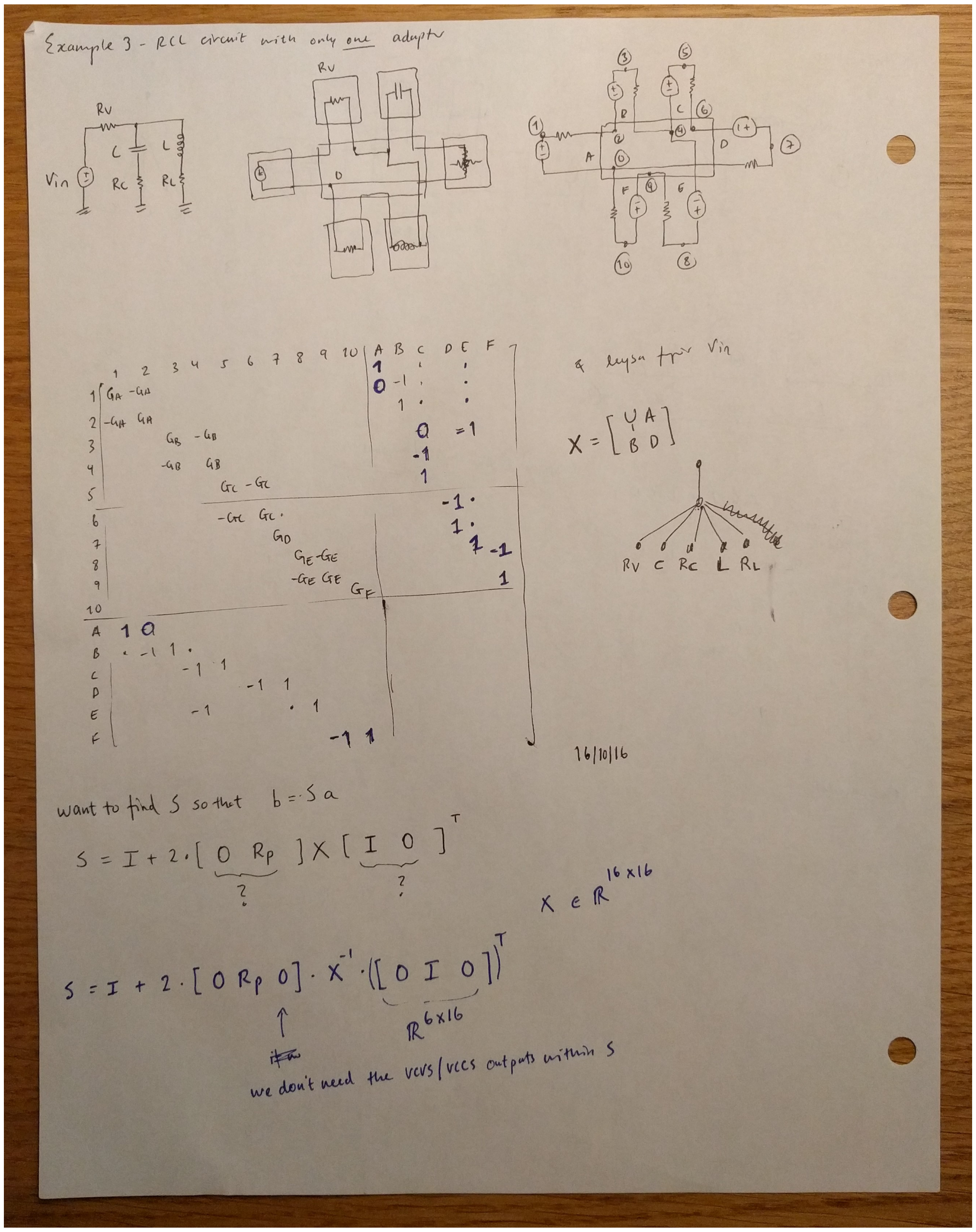
11 Final thoughts

12 Biliography and references

13 LTspice files and other handy things



WDF derivation for example 1


Rv
C
L
Vin
Rc
RL
D
A
B
C
D
E
F
Rv C Rc L RL

Example 3 - RCL circuit with only one adaptor

& leysa fyrir Vin

$$X = \begin{bmatrix} Y & A \\ B & D \end{bmatrix}$$

$$\begin{array}{c|cccccccccc|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & A & B & C & D & E & F \\
\hline
1 & G_A & -G_A & & & & & & & & & 1 & & & & & \\
2 & -G_A & G_A & & & & & & & & & 0 & -1 & & & & \\
3 & & & G_B & -G_B & & & & & & & & 1 & 0 & & =1 & \\
4 & & & -G_B & G_B & & & & & & & & & -1 & & & \\
5 & & & & & G_C & -G_C & & & & & & & 1 & & & \\
6 & & & & & -G_C & G_C & & & & & & & & -1 & & \\
7 & & & & & & & G_D & & & & & & & 1 & & \\
8 & & & & & & & & G_E & -G_E & & & & & & 1 & -1 \\
9 & & & & & & & & -G_E & G_E & & & & & & & 1 \\
10 & & & & & & & & & & G_F & & & & & & \\
\hline
A & 1 & 0 & & & & & & & & & & & & & & \\
B & & -1 & 1 & & & & & & & & & & & & & \\
C & & & & -1 & 1 & & & & & & & & & & & \\
D & & & & & & -1 & 1 & & & & & & & & & \\
E & & & & -1 & & & & & 1 & & & & & & & \\
F & & & & & & & & & -1 & 1 & & & & & & \\
\end{array}$$

16/10/16

want to find S so that $b = S a$

$$S = I + 2 \cdot [\underbrace{0 \quad R_p}_{?}] X [\underbrace{I \quad 0}_{?}]^T$$

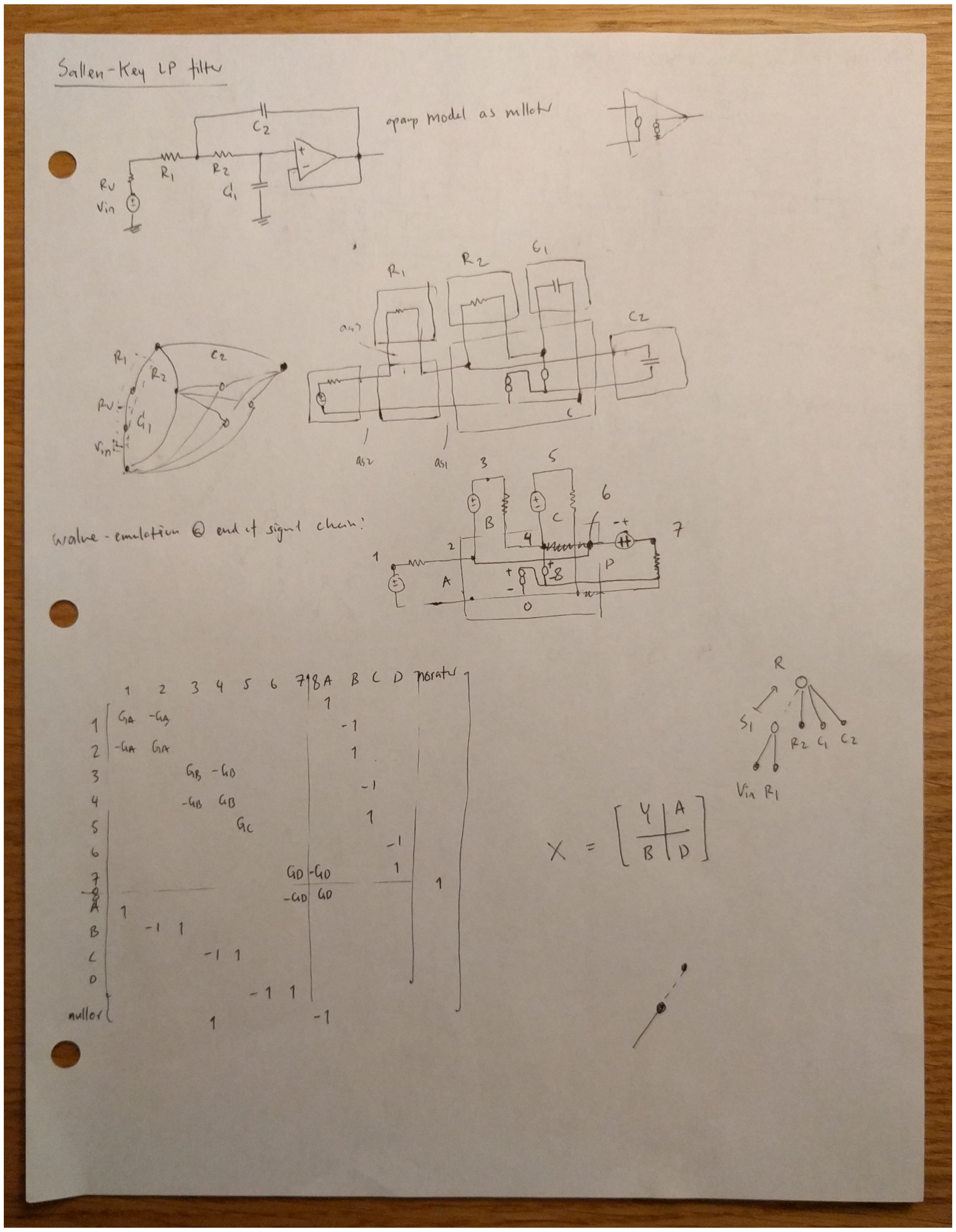
$$X \in \mathbb{R}^{16 \times 16}$$

$$S = I + 2 \cdot [0 \; R_p \; 0] \cdot X^{-1} \cdot (\underbrace{[0 \; I \; 0]}_{\mathbb{R}^{6 \times 16}})^T$$

$\uparrow$

~~it [illegible]~~ we don't need the vcvs/vccs outputs within S

WDF derivation for example 1



WDF derivation for example 1