# DAT116 (Mixed-signal system design)
# Lab 4: Continuous-time filter design

Lars Svensson
larssv@chalmers.se

Version 3.1, December 3, 2018

## 1 Introduction

In this lab session, you will design an active low-pass filter according to a given specification. You will use MATLAB to select a filter transfer function and then simulate the filter in Cadence to verify its functionality and to investigate the effects of component inaccuracies.

You will find that instructions for this lab are less detailed than for the previous ones. You will be required to use the MATLAB documentation to determine what command sequences to use.

## 2 Preparation

Operations required in this session have in many cases occurred also in previous sessions. In those cases, descriptions are quite brief. You may need to refer to previous lab PMs for details.

We will refer to the classical filter functions: the Butterworth, Chebyshev, and elliptical (Cauer) filters. These are well described in the Wanhammar filter textbook (as listed in the reading list and available as an eBook in the library).

We will use functions from the MATLAB Signal Processing Toolbox. In particular, we will use functions from the category *Digital and Analog Filters*. Find the corresponding section of the MATLAB documentation and study the functions `butter` and `buttord` in particular.

The filter section presented in section 5 can be used to implement a wide range of pole values. You will need to calculate component values starting from pole values; we will derive the necessary formulas in advance, in class, but you will need to insert values into the formulas during the lab session.

# 3 Filter order

Your first task is to find a transfer function[1] which meets the low-pass filter specification illustrated in figure 1.

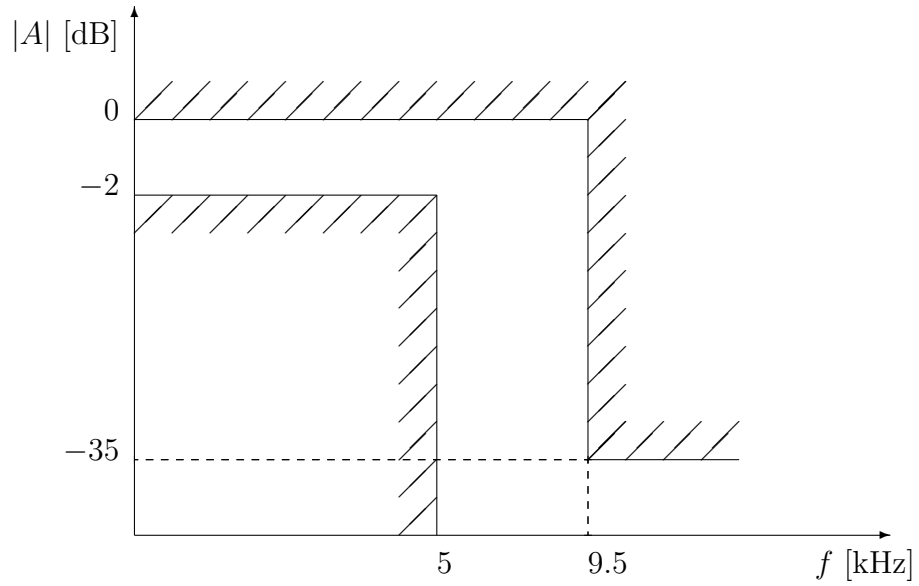


Figure 1: Low-pass filter magnitude specification.

- Launch MATLAB and select a working directory for this lab. Make sure MATLAB's search path includes your own directory for `.m` files.
- Use the `buttord` function to determine the necessary order for a Butterworth filter to meet the specifications in figure 1. (Don't leave out the extra argument `'s'` which specifies a time-continuous filter! Beware of the use of frequency vs. angular frequency!)

[1] In this lab, we will only consider the "classical" filter transfer functions. These are intended for "boxy" specifications like the one given here, and are well-supported by tools and literature.

- Repeat the operation for the other classical filters (the two Chebyshev types and the elliptical type). You will most easily find the necessary documentation by following links from the `butter` page (especially from the **See Also** section at the end).

The implementation style we will employ (see Section 5 below) is only usable for all-pole transfer functions. Thus, we should choose the transfer-function type which has no zeros (except at infinity) and allows the lowest-order implementation. Which one is this?

# 4 Transfer function selection

As you have seen during preparations, the MATLAB function `butter` constructs a Butterworth transfer function given a cutoff frequency and a filter order. Similar functions exist for the other classical filter types; these functions can return arrays of polynomial coefficients, *or* (equivalently) pole and zero arrays, depending on the number of output arguments. (Note: *you cannot use the same parameter list for the different filter-design functions;* refer to the documentation for each case!)

- Using the filter order arrived at above, generate a transfer function of the chosen type to meet the given specifications, and save it in pole-zero form.

Poles and zeros can be plotted with the provided function `splane` (which is modeled on the function `zplane` which is included with MATLAB). The `.m` file is available on the PingPong page.

- Plot poles and zeros of the transfer function using `splane`. Save the plot for your lab report.

The figure-1 filter specification was given as limits on the magnitude function. In order to compare the filter frequency response with the specification, we need to calculate the magnitude of the transfer function as a function of frequency. MATLAB provides the function `freqs` for this purpose. Somewhat inconveniently, `freqs` requires the filter specification to be in the polynomial-coefficient form rather than in the pole-zero form.

- Generate the transfer function again as above, but this time, save it in the polynomial-coefficients form.

- Use the `freqs` function to generate a plot of the filter magnitude response. Zoom into the plot to verify that the original specifications are fulfilled. Generate plots for your lab report.

# 5 Circuit implementation

The selected transfer function must now be implemented in circuit form. We will use a cascade of filter sections which each implement one complex-conjugate pole pair. A literature search will find myriad ways to implement such a second-order section. We have chosen one particularly simple form, based on the Sallen-Key topology[2] but with a variable gain which allows Q-value adjustment. The circuit in question is available in the `dat116` library under the name `sk`, and is shown in figure 2.

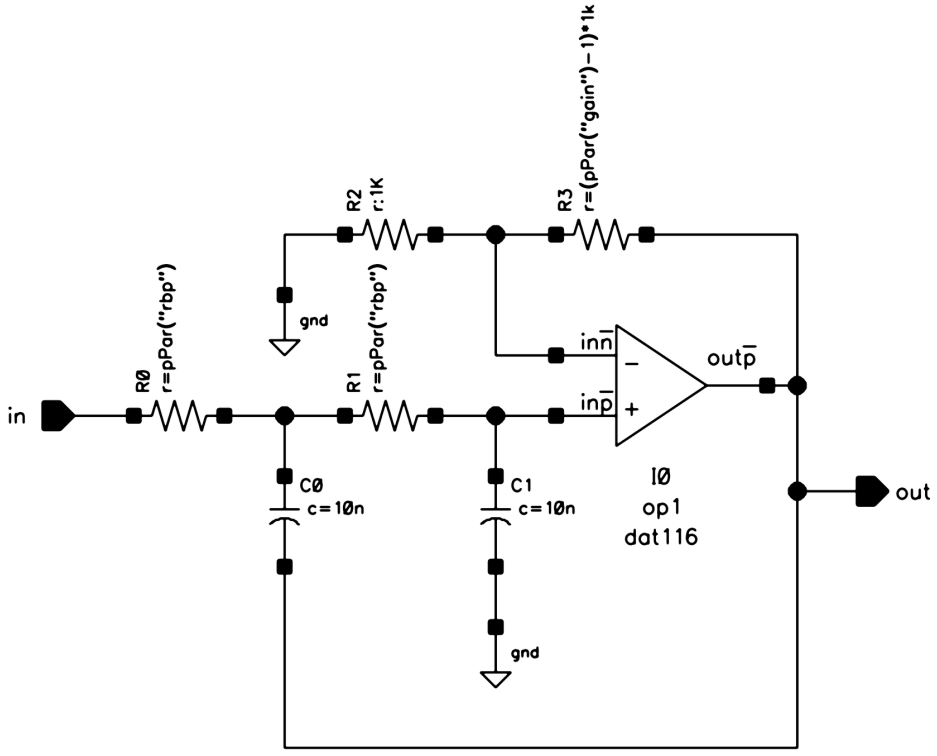


Figure 2: The `sk` cell implements a two-pole transfer function. Two identical resistors and two identical capacitors determine the pole frequency; the gain of a non-inverting amplifier sets the Q value.

[2] `http://en.wikipedia.org/wiki/Sallen-Key_topology`

- Launch Cadence. Launch the **Library Manager** and select the `dat116` library.
- Open the schematic view of `sk` and inspect the contents. How are the resistor values determined?

A good way to verify your understanding of the `sk` parameter handling is to include one instance of the circuit in a test bench cell and perform an AC simulation.

- Create or select a work library for this lab. In this library, create a schematic view of a test bench cell. Include an instance of `sk`. Connect a 10-k$\Omega$ resistance from the `sk` output to ground, and a `vsin` source from the input to ground. Save the schematic view.
- Since the `sk` behavior is defined by `verilogams` code (in `op1`), a `config` view is needed for simulation. Create a `config` view for the test bench, following the same procedure as in Lab 2.
- Launch the Analog Design Environment (**ADE L**), and set the simulator to `ams`.
- Carry out a few AC simulation frequency sweeps with different parameter combinations for `sk`. As you saw during preparations, the parameter `gain` should be strictly lower than 3! Compare the results with your expectations. Make plots for your report.

You are now equipped to implement a filter according to the specification in figure 1, using a cascade of `sk` stages.

- Construct a filter according to the specifications in figure 1. Use one `sk` cell for each complex-conjugated pole pair. Real-valued poles (if any) may be implemented as simple RC links. The overall gain will be larger than 0 dB and should be compensated for with a resistive divider at the output. Save a plot of your filter circuit for your report.
- Set up and run an AC analysis to verify your implementation. Use a logarithmic sweep for the frequency. You will want a high frequency resolution at the edges of the transition band; start out with 100 points per frequency decade. With an AC magnitude of 1 for the input signal, the output magnitude directly gives the filter magnitude function with no further calculations.

In the specification, the passband and stopband attenuations were given in decibels, so the simulated magnitude function should be plotted in the same way for easy comparison. We have seen previously (in Lab 2) how to do that using the **Calculator** tool. The same procedure should be followed here.

- Generate a plot of the magnitude function of your filter, on a dB scale and with a logarithmic frequency axis. Zoom in on the passband and stopband edges to check that the specification is fulfilled (if necessary for confidence, increase the number of points per decade and resimulate). Save plots for report.

# 6 Component inaccuracies

Simulations in the previous section have assumed that all component values are accurate. In practice, accuracies vary widely with the implementation technology. Circuits built with discrete components on PCBs are likely to be more accurate than on-chip implementations; for the latter case, ratios of component values may be significantly more accurate than absolute values. To be viable, a filter implementation must meet the specifications even with inaccurate component values.

We will now use the Monte-Carlo capabilities of the Cadence system to investigate the influence of filter component inaccuracy. The first step is to replace the `sk` implementation with a version which supports randomization of resistances and capacitances.

- Find the `sk1` cell in the `dat116` library. Inspect its schematic and verify that the topology corresponds to that of the `sk` cell.
- Select one of the resistances in `sk1` and open its property dialog (with `[q]`). Observe the presence of a value for the `Model Name` parameter, and make note of that name. Dismiss the dialog and close the `sk1` schematic.
- Create another filter test bench like the previous one, but using `sk1` cells rather than `sk` cells. Set the cell parameters to the values you used previously. Check and save the new test bench schematic.

The **ADE L** interface does not support Monte-Carlo simulation; thus it is necessary to launch a more complete simulator interface.

- In the schematics window, select **Launch→ADE XL**. Create a new `adexl` view for your filter testbench design. The schematics window changes to a tabbed window showing the **ADE XL** welcome screen; the menu bars etc also change to reveal a rich set of tools (we will only use a small subset of these). Verify that your schematic is still present in the original tab before returning to the `adexl` tab. Note that the available controls and menus change with the selected tab.

Find the `Data View` pane, in the top left part of the window, and the item `Tests` therein. Loosely speaking, a "test" is a (set of) simulation(s) together with evaluation criteria for the simulation results. In this lab, evaluation will be manual, so we can make do with a very simple test.

- Expand the `Tests` item by clicking on the `+` sign. Next, click on the emerging item `Click to add test`. A design-selection dialog window opens. Verify that the library and cell name correspond to your filter test bench cell, and click **OK**.

The other new window, **ADE XL Test Editor**, is very similar to the **ADE L** windows that you used previously, and the interaction is also similar. One difference from the **ADE L** tool is that the `Run` button has been renamed `Debug Test`; its intended use is to run a single simulation before embarking on the whole Monte-Carlo set.

- In the test editor window, set up an AC analysis with the same parameters as above, and arrange for the node voltages to be plotted and saved.

You will control the statistical component value distributions with a simple text file. A template file, `variations.scs`, should have been copied into your work directory when you launched Cadence.

- Inspect `variations.scs` in a text editor. Component parameters are declared at the top. The model name used for the resistor in `sk1` reappears in the template file; the model definition includes one of the component parameters. A `statistics` block specifies Gaussian distributions for the parameter values. We will reinspect these values later; for now, just close the file.

For the variations to be applied, we must arrange for the file to be included with the simulator netlist.

- In the Test Editor window, select **Setup→Model Libraries...** Click in an empty field under the heading `Global Model Files` in order to add `variations.scs` to the list (you may enter the full file-system path, starting with a `/`, or call up a browser with the button marked ... to the right of the field). Ensure that the check mark is set for the new entry. Click OK to exit the dialog.

When all this is done, it is finally possible to test the simulation setup.

- Click `Debug Test` to execute a single simulation run. Dismiss the warning about this being a debug run. Eventually a plot window will open to show the same magnitude function as before.

Once the test simulation performs as expected, it is time to move on to the actual Monte-Carlo runs. These are controlled from the main tabbed window with the `adexl` view.

- In the **ADE XL** window, select the `adexl` tab.
- From the main menu bar at the top of the window, select **Run→Monte Carlo Sampling...** A control panel opens to let you decide on specific parameters for the Monte Carlo runs. Under `Statistical Variation`, select `Mismatch`; under `Sampling Method`, select `Random` and 50 points; and under `Other Options`, make sure to check `Save Data to Allow Family Plots`. Finally, click **OK** to start the Monte Carlo runs. Progress is shown in the `History` pane in the lower left corner of the **ADE XL** window.
- When the Monte Carlo run set has completed, select the `Results` sub-tab in the `adexl` tab. Find and click the "plot all waveforms" button near the middle of the command/menu bar in the `Results` pane. Families of curves appear in the graph window for each of the signals selected for plotting.
- In the same way as before, use the **Calculator** functions to generate a plot of the curve families on a dB scale.
- Inspect the plot and compare the curves with the specifications shown in figure 1. Zoom in on the edges of passband and stopband. Most likely, you will find that some of your curves violate the specification. Save plots for your report.

Above, we chose `Mismatch` for the `Statistical Variation` control. This alternative means that each actual component value is to be *independently* selected

according to a Gaussian distribution centered on the nominal design value. In addition to this "small-scale" variation, many implementation technologies suffer from a larger-scale variation *correlated* across all components in the same circuit; the *ratio* of two component values in the same filter will not be affected by such variations. (Recall the `rglob` variation of Lab 2.)

Reopen the file `variations.scs`. The file specifies two categories of variations, labeled `mismatch` and `process`. The `process` specification models chip-to-chip variations; here, the capacitance variations have been especially emphasized[3]. The `Statistical Variation` alternative `All` will apply these large chip-to-chip capacitance variations *in addition to* the small-scale variations applied by `Mismatch`.

- Again select **Run→Monte Carlo Sampling...** in the **ADE XL** window. Change the value for `Statstical Variation` to `All` and rerun the Monte Carlo set. Create a dB-scaled plot as above. Comment on the qualitative appearance of the new curve family. Save a plot for your report.
- Zoom in on the passband and stopband edges as before. Save plots for report. Compare the result with those of the previous simulation; discuss.

These simulations have been intended to illustrate how to estimate the influence of component variations in filter design. The approach shown here does not cover all the issues, though. In particular, we have applied variations only to the passive filter components. Amplifier parameters may vary even more; but in circuits such as this one, the negative feedback ensures that overall performance is not significantly affected by such variations (as was illustrated in Lab 2). Additionally, any output voltage divider or other level-adapting circuits would also have limited precision; several parameter variations may show correlations; etc. With time and effort, it is possible to cover such more complex cases as well.

# 7 Wrap-up

After completing this lab session, you are supposed to be able to do the following:

- Use the MATLAB filter-design tools to select a transfer function to meet a simple specification.

[3]This has been done for illustration only; a real semiconductor process may have very different statistical properties.

- Create a cascade implementation of a transfer function using predefined filter stage topologies.
- Use Cadence Spectre simulations to verify that the filter implementation meets the specification.
- Use Monte-Carlo simulation to estimate the influence of component inaccuracy on a filter design.

**Reflection question:**

- The Monte-Carlo simulation results illustrate that even though a filter is designed to certain specifications, limited component precision may cause bad yield: many of the manufactured filters will fail to meet the original specs. What can we do *at the design stage* to improve the expected yield? How would such improvements influence the cost?
- Steep filters have poles close to the imaginary axis in the s plane. You have seen that the positions of such poles can be sensitive to component inaccuracies, and that large deviations from the intended magnitude function can result. How can you use this knowledge when selecting the sample rate in a data acquisition system?