SLiCAP: Symbolic Linear Circuit Analysis Program

Python scripts for Analog Design Automation https://analog-electronics.tudelft.nl

Goal

Automation of Analog Design

1. Proper modeling of device behavior
2. Symbolic analysis
3. Derive design equations that relate circuit behavior to component properties
4. Find target values for component properties by solving these equations
5. Find components from libraries with preferred devices

Benefits

Guaranteed by design

- Speed up design process:
  - Find dominant design parameters and their interactions at an early stage of the design
- Improve design quality:
  - Structured design approach with stepwise increasing level of complexity
  - Facilitate symbolic analysis tasks that are cumbersome, error-sensitive and often omitted
  - Performance fixed with dominant physical mechanisms

Features

Design and design documentation

- Accepts SPICE-like netlists:
  - including sub circuits and models
- Symbol libraries available for:
  - gschem
  - LTSpice
  - Pulsonix
- Built-in EKV model for MOS transistors and GP model for bipolar transistors:
  - Small-signal parameters determined by process parameters, device geometry and operating conditions
  - Any of these parameters can be varied (stepped) during analysis
- Built-in small-signal models for:
  - Diode
  - 4-terminal vertical BJT and lateral BJT
  - differential-pair BJT
  - 4-terminal MOST
  - differential-pair MOST
  - Voltage-feedback Operational Amplifier
  - Current-feedback Operational Amplifier
- Design-oriented analysis:
  - Symbolic and numerical derivation and solution of design equations
  - Many topic-specific functions for extracting useful design information from complex expressions
- Combine results of different analysis:
  - Find show-stopper values and target values for design parameters from different performance aspects (noise, bandwidth, ...)
- Concurrent design and documentation:
  - One-click generation and update of html-based documentation
  - Include CSV tables, text files, images, MATLAB figures, beautifully typeset expressions and equations
- Easy to use:
  - Comprehensible and compact instructions

Capabilities

Linear(ized) Circuits

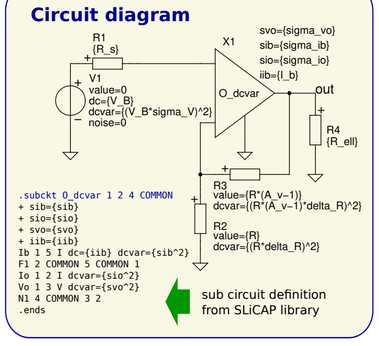
- Symbolic and numerical DC variance analysis:
  - Find requirements for standard deviation of offset and bias sources and of resistor tolerances
- Symbolic and numerical noise analysis:
  - Find requirements for noise sources of operational amplifiers and device geometry and operating conditions of MOS and bipolar transistors
  - Correlated double sampling as well as spectral noise integration can be included
- Symbolic and numerical analysis of linear dynamic circuits:
  - Symbolic and numerical solution of the complete network, of an individual branch voltage or current or of a transfer function
  - Pole-zero analysis with/without PZ cancellation
  - Find design equations for filter components
  - Plot PZ, frequency and time-domain responses from (Inverse) Laplace Transform
- Symbolic and numerical analysis of feedback circuits with the asymptotic-gain model:
  - Select any controlled source as loop gain reference
  - Derive design equations from expressions for:
    - Asymptotic-gain
    - Loop gain
    - Servo function
    - Direct transfer
    - Gain
  - Find requirements for DC gain and GB product of operational amplifiers
  - Find requirements for number of transistor stages in amplifiers and for their operating conditions
  - Investigate stability region with symbolic Routh Array
  - Find asymptotic servo bandwith and asymptotic value of mid-band loop gain or DC loop gain
  - Pole-zero analysis with/without PZ cancellation
  - Determine frequency compensation strategy and implementation
  - Make Nyquist plot of loop gain
  - Plot root-locus for any root-locus parameter
  - Plot PZ pattern, magnitude, phase, delay, impulse or step response from (Inverse) Laplace Transform

Technology

Python + Maxima + Internet

- Optional:
  - Python Sphinx, gschem, inkscape, LTspice

Circuit diagram



SLiCAP script

```
from SLiCAP import *
prj = initProject('myProject')
i1 = instruction()
makeNetlist('dcBehavior.sch', 'dcBehavior')
i1.setCircuit('dcBehavior.cir')
i1.setSource('V1')
i1.setDetector('V_out')
i1.setSimType('symbolic')
i1.setGainType('vi')
i1.setDataType('dcvar')
dcvarResults = i1.execute()
```

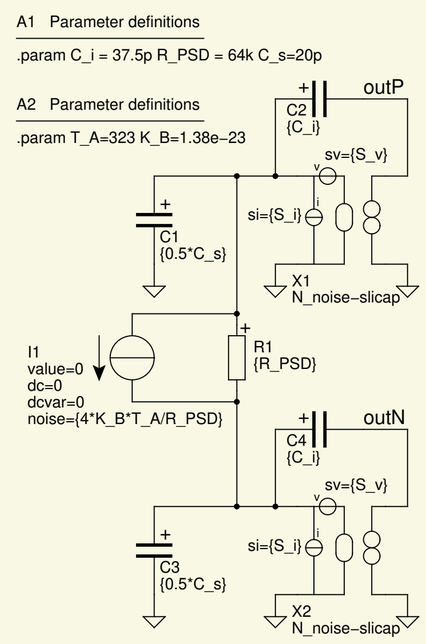
DC network solution

$$\begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_{3X1} \\ V_{5X1} \\ V_{out} \\ I_{V1} \\ I_{R2} \\ I_{R3} \\ I_{iF1X1} \\ I_{VoX1} \\ I_{oN1X1} \end{pmatrix} = \begin{pmatrix} V_B - I_b R_s \\ V_B \\ V_B - I_b R_s \\ V_B - I_b R_s \\ 0 \\ A_v V_B - I_b (R - R A_v) - I_b A_v R_s \\ -I_b \\ \frac{V_B}{R} - \frac{I_b R_s}{R} \\ \frac{I_b R_s}{R} - \frac{V_B}{R} - I_b \\ I_b \\ 0 \\ \frac{I_b (R_s R_\ell + R A_v R_s)}{R R_\ell} - \frac{V_B (R_\ell + R A_v)}{R R_\ell} - \frac{I_b (R_\ell - R + R A_v)}{R_\ell} \end{pmatrix}$$

Variance DC detector voltage

$$\begin{aligned} \sigma^2_{DET} = {} & \\ & \sigma^2_{\mathrm{ib}} (R - R A_v + A_v R_s)^2 \\ & + {A_v}^2 \sigma^2_{\mathrm{vo}} \\ & + \sigma^2_{\mathrm{io}} (R A_v - R + A_v R_s)^2 \\ & + {A_v}^2 {V_B}^2 \sigma^2_V \\ & + R^2 \delta^2_R (A_v - 1)^2 \left(I_b + \frac{V_B}{R} - \frac{I_b R_s}{R}\right)^2 \\ & + R^2 \delta^2_R (A_v - 1)^2 \left(\frac{V_B}{R} - \frac{I_b R_s}{R}\right)^2 [\mathrm{V}^2] \end{aligned}$$

PSD pulse position detection system design of noise behavior



Design Task

Determine show-stopper values for S_v and S_i such that the total differential RMS output noise over a bandwidth of 450kHz, after CDS with $\tau = 5\mu s$, is less than $50\mu V$.

SLiCAP script

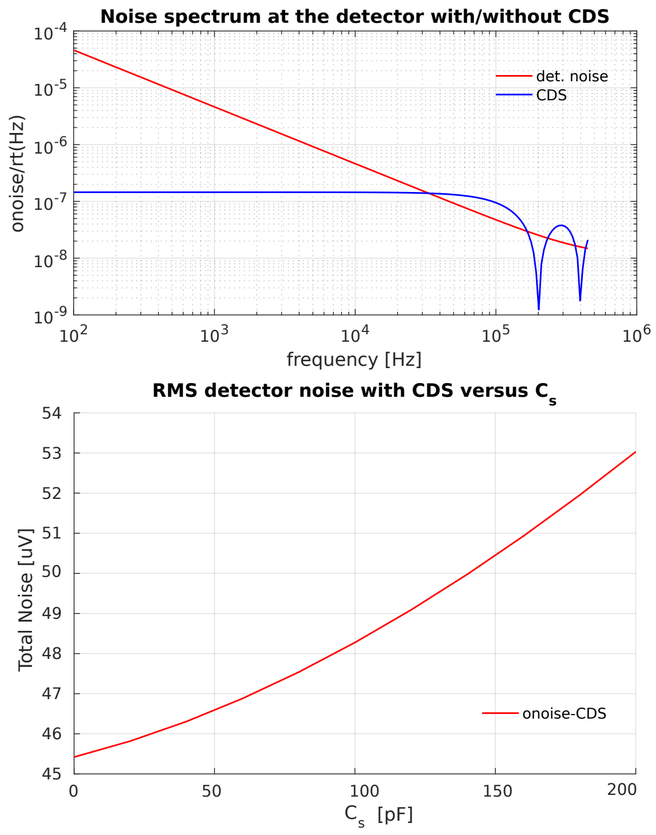
```
syms 'F' 'S_v' 'S_i';
F_min = 1     # Minimum frequency
F_max = 450e3 # maximum frequency
vn    = 50e-6 # RMS diff output noise after CDS
tau   = 5e-6  # CDS delay time
i1.setCircuit('PSDnoiseDesign')
i1.setSimType('numeric')
i1.setGainType('vi')
i1.setDataType('noise')
i1.setDetector(['V_outN', 'V_outP'])
onoise = i1.execute().onoise
rmsSv = doCDSint(onoise.subs(S_i, 0), tau, F_min, F_max)
Sv_max = sp.N(solve(rmsSi - vn, sp.Symbol('S_v'), 4)[0]
rmsSi = doCDSint(onoise.subs(S_v, 0), tau, F_min, F_max)
Si_max = sp.N(solve(rmsSi - vn, sp.Symbol('S_i')), 4)[0]
print('Sv_max =', Sv_max, ' V^2/Hz'))
print('Si_max =', Si_max, ' A^2/Hz'))
```

Result

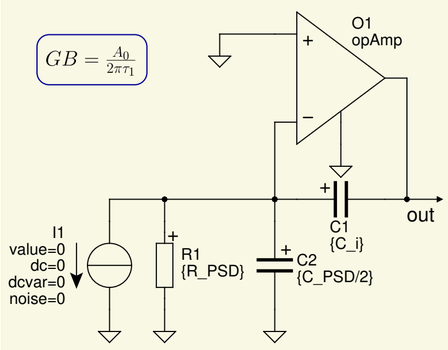
```
Sv_max = 1.03e-16 V^2/Hz
Si_max = 1.82e-25 A^2/Hz
```

Select opAmp with

```
S_v = 6 nV/rtHz
S_i = 5 fA/rtHz
```



PSD pulse position detection system design of dynamic behavior



SLiCAP results

Design dynamic behavior

This page gives the design equations for the high-pass and the low-pass cut-off.

High-pass cut-off

A high-pass cut-off frequency at 1Hz requires:

$$A_0 = 65299.0 \quad (1)$$

Low-pass cut-off

If all poles and zeros are dominant, a low-pass cut-off at 4.5e+05Hz requires:

$$\mathrm{GB} = 12.72\, R_o \left(\left(1.0 \cdot 10^{11}\right) c_d + 1.0\right) \quad (2)$$

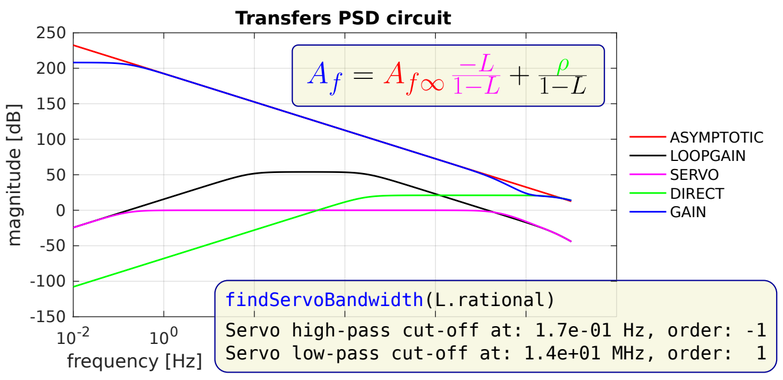
If the influence of a nonzero R_o on the dynamic behavior can be ignored, we need:

$$\mathrm{GB} = \left(1.2 \cdot 10^{16}\right) c_d + 5.7 \cdot 10^5 \quad (3)$$

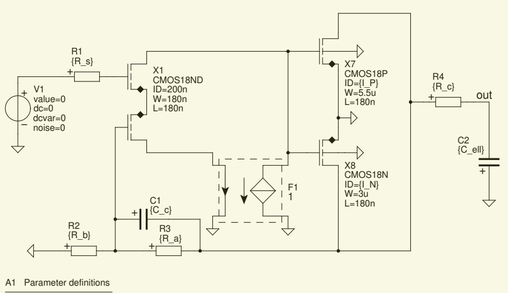
Go to main index

Select operational amplifier

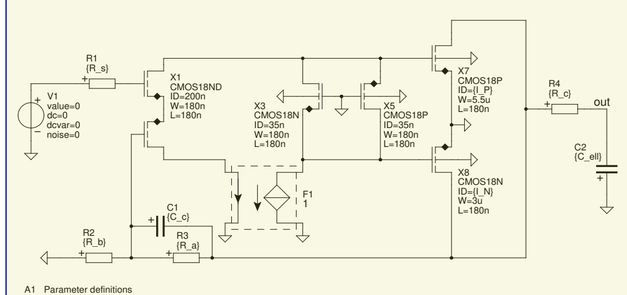
```
* file: AD8610.lib SLiCAP model for AD8610
.model AD8610 OV
+ cd = 15p
+ cc = 8p
+ av = {300k*(1-s/2/PI/120M)/(1+s*300k/2/PI/25M)/(1+s/2/PI/120M)}
+ zo = 20
```



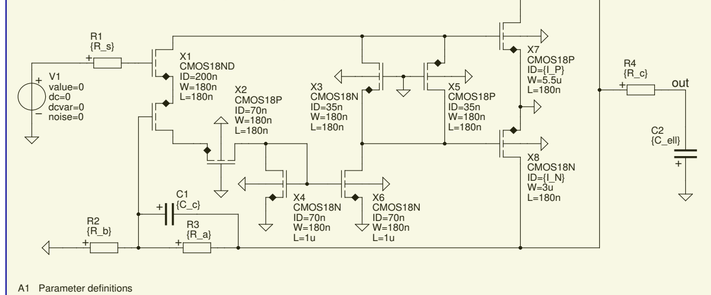
Vamp_0
CMOS amplifier with compl. parl. class AB output stage



Vamp_1
CMOS amplifier with split-signal compl. parl. class AB output stage



Vamp_2
CMOS amplifier with complete signal path



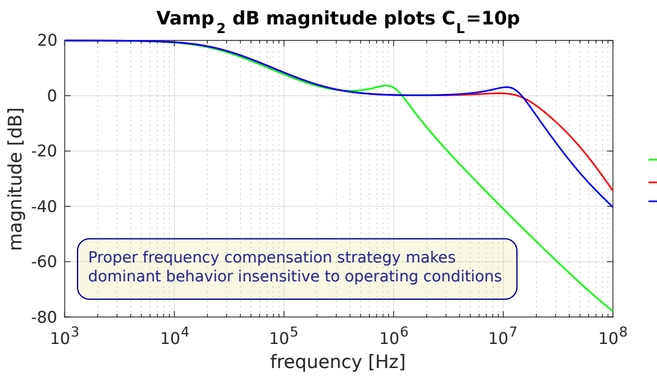

```
.subckt CMOS18N drain gate source bulk W={W} L={L} ID={ID}
M1 drain gate source bulk CMOS18N
.model CMOS18N M
+ cgs     = {W*L*C_0 + C_O*W}
+ gm      = {g_m}
+ go      = {ID/VAL/L}
+ cdg     = {C_O*W}
+ gb      = {(N-1)*g_m}

.param
+ TOX     = 4.1n  ; Oxide thickness                          [m]
+ C_O     = 700p  ; Overlap capacitance                      [F/m]
+ U_T     = 25.6m ; Thermal voltage at room temperature      [V]
+ Theta   = 0.28  ; Vertical field mobility reduction factor [V^-1]
+ E_CRIT  = 5.6M  ; Critical velocity saturation field       [V/m]
+ N       = 1.35  ; Substrate factor                         [-]
+ VAL     = 20M   ; Early voltage per unit of channel length [V/m]
+ mu_0    = 42.2m ; Low-field channel carrier mobility       [m^2/Vs]
+ C_0     = {8.8p * 3.9 / TOX}   ; Normalized oxide capacitance [F/m^2]
+ I_0     = {2*N*mu_0*C_0*U_T^2} ; Technology current        [A]
+ IC      = {ID*L/W/I_0}         ; Inversion coefficient     [-]
+ IC_CRIT = {1/(8*(N*U_T)^2*(Theta+1/L/E_CRIT))}
+ g_m     = {ID/(N*U_T*sqrt(IC*(1+IC/IC_CRIT)+0.5*sqrt(IC*(1+IC/IC_CRIT))+1))}
.ends
```

MOS EKV model in SLiCAP library

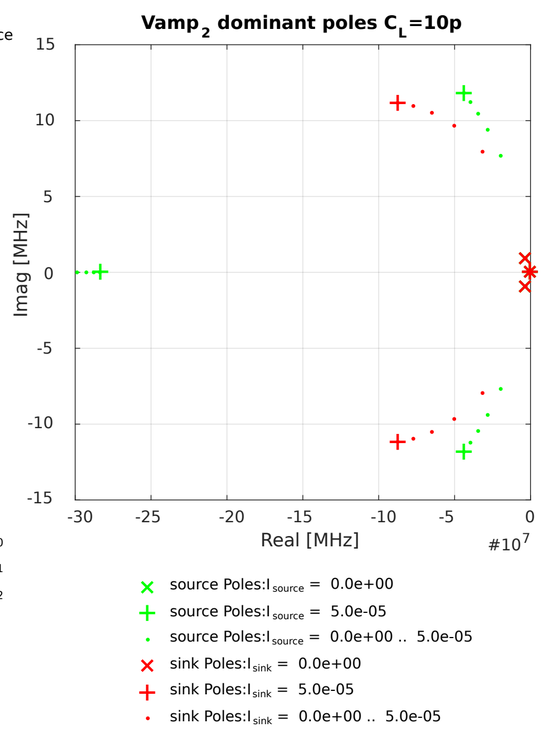
Small-signal model tracks with inversion coefficient from weak inversion to strong inversion with velocity saturation.
Combine:
- Large-signal static nonlinear behavior
- Small-signal dynamic behavior

Poles are function of sink / source current

```
makeNetlist('MOSamp_2.sch', 'Vamp_2')
i1.setCircuit('MOSamp_2.cir')
i1.setsource('V1')
i1.setdetector('V_out')
i1.setsimType('numeric')
i1.setgainType('gain')
i1.setdataType('poles')
i1.setstepVar('I_source')
i1.setstepStart(0)
i1.setstepStop('50u')
i1.setstepNum(5)
i1.setstepMethod('lin')
i1.stepOn()
amp2source10p = i1.execute()
amp2source10p.gainType = 'source'
i1.setstepVar('I_sink')
amp2sink10p = i1.execute()
amp2sink10p.gainType = 'sink'
```



Combine multiple root locus plots in one plot:
- Select any circuit variable as root-locus variable
- Separate pole and zero analysis can show non-observable poles and hidden instability



Combine results from different runs and from different circuits in one plot