KÄRNTEN University of Applied Sciences

CARINTHIA INSTITUTE FOR MICROELECTRONICS

An easy to use Python framework for circuit sizing from designers for designers.

Wolfgang Scherr, DVCON 2024, Munich

Violeta Petrescu, Johannes Sturm, Dirk Hammerschmidt, Santiago Martin Sondon

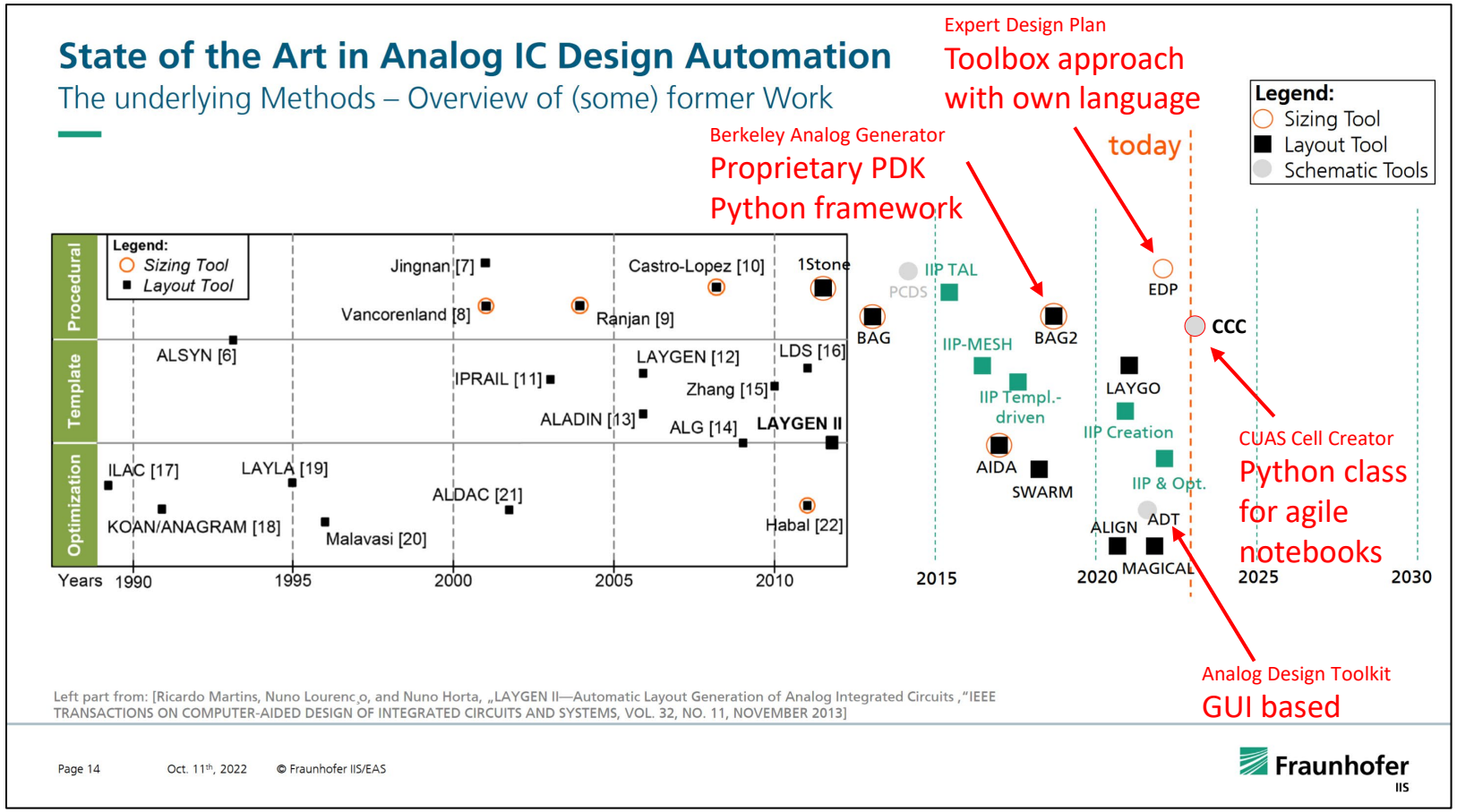
Josef Ressel Centre for System-on-Chip Design Automation

Federal Ministry Republic of Austria Labour and Economy

Christian Doppler Forschungsgesellschaft

V1.1

Automation is not new…



Source: Keynote Austrochip 2022, Benjamin Prautsch, Fraunhofer IIS
https://publica-rest.fraunhofer.de/server/api/core/bitstreams/384d05a3-60b7-4216-aa63-dbcb1dd35a6d/content

Why yet another approach?

- Electrical engineering ≠ Computer science
- Analogue design competence ≠ Programming competence
- Ensure the focus on the needs of analogue designers (not programmers)
- Make exchange of design procedures as simple as passing on a schematic
- Lightweight setup, PDK & tool agnostic, open for any tool extension

Assume you need a **Schmitt-trigger** for TSMC 65nm – given spec: Vil, Vih, Vihys.

- One might probably think it is trivial, but…
- … how to choose the design (and from where)?

- How long would it take to design it?
  Or just let an optimiser like e.g. Wicked find them?

- How long would it take to verify (at least PVT)?

- How to post-process and document the results?

- What about quickly reacting on a spec update later on?
- Or what about re-use for e.g. TSMC 28nm?

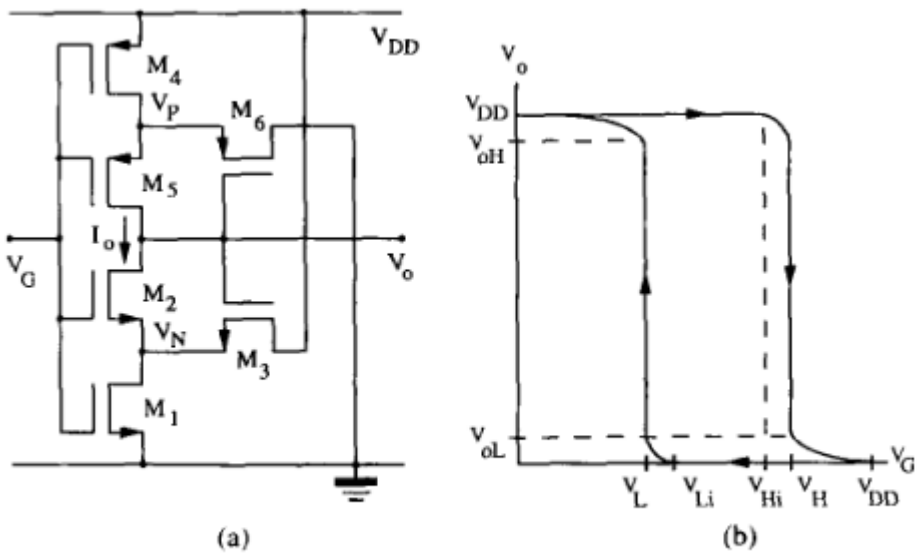


Fig. 1. CMOS Schmitt trigger and its transfer characteristic.

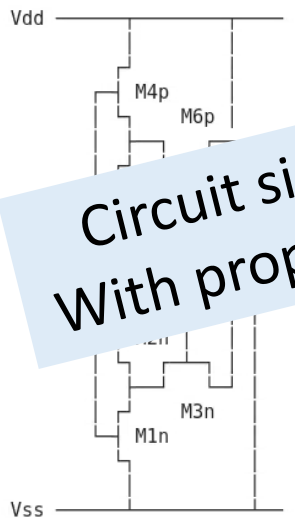
I. M. Filanovsky and H. Baltes, "CMOS Schmitt trigger design," in *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 41, no. 1, pp. 46-49, Jan. 1994

Circuit generation with CCC

- Get the Jupyter notebook of this generator **from a lib or just a colleague**
- Open a terminal, start Spyder (as tool to use Jupyter notebooks)
- Open/run the notebook → take the TB, schematic

Circuit sizing, documentation and verification results all in a single, share-able file! With proper commenting, it also fully explains how/why the sizing was done that way.

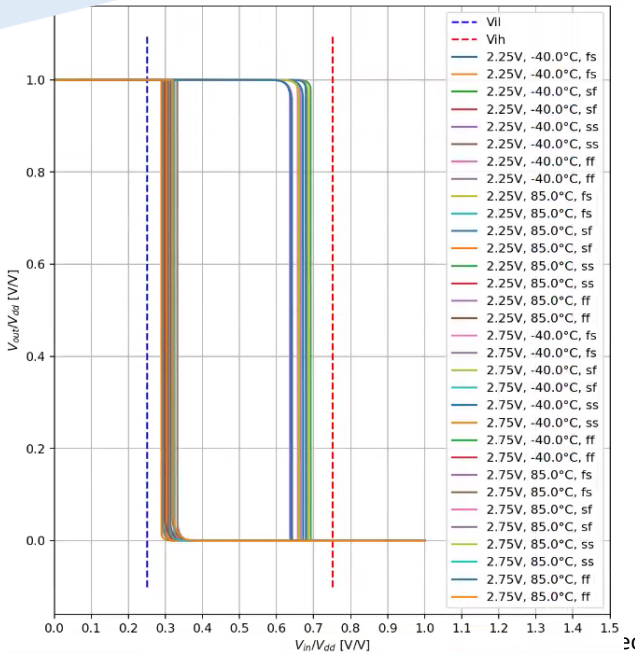
Circuit to set up




```
[2]: # We design relative to any technology core supply, thus parameters
     # Typical CMOS switching levels can be retrieved from JEDEC
     # E.g.: JESD8-12A.01, November 2005, JC-16 Committ
     # "1.2 V +/- 0.1 V (NORMAL RANGE) AND 0.8
     #  AND INTERFACE STANDARD FOR NONTE
     #  JEDEC Board Ballot JCB
     #
     # ST spec                                   ....0.5Vdd
                                                owed spec (application engineer):
                                                range)
                                                N28"
                                                = "tsmcN65"
     tech = "tsmc18"

     # basic (generic) elements to use, if one wants e.g. a specific 1.2V design, use MN12 and MP12
     nmos = "MN"
     pmos = "MP"

     # note, the generator is PDK INDEPENDENT! Thus, we
     # ideally deal with multiples of minimum feature sizes
     # we will focus on a small, minimum W/L design...
     # so: Wmin*2^Wpot=Wmin and Lmin*2^Lpot=Lmin
     Wpot=0
     Lpot=0
```

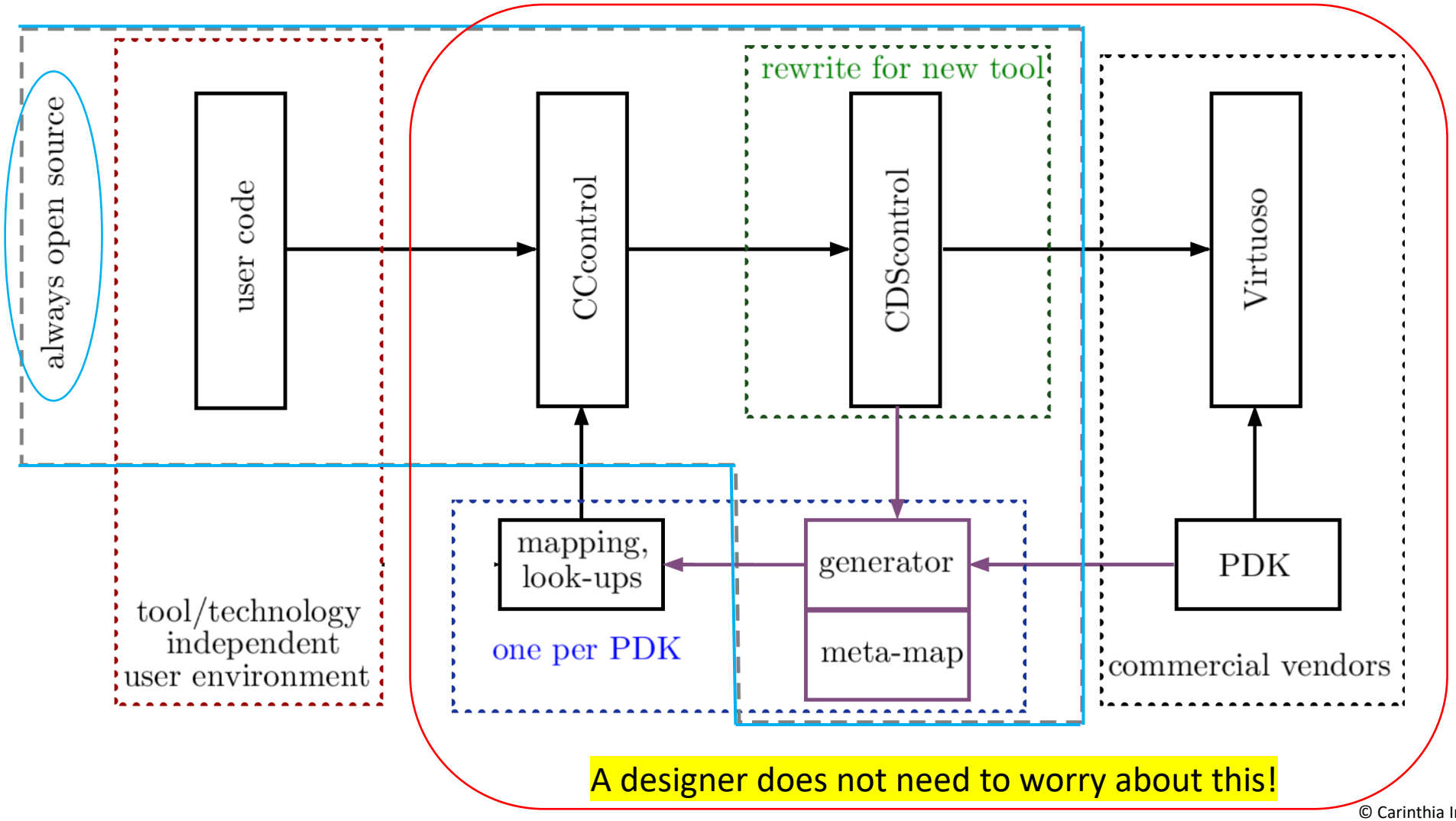
tsmcN65

MN25/MP25



How can CCC handle “any” technology and tool?

Tested for TSMC:
- **180nm**
- **65nm**
- **28nm**



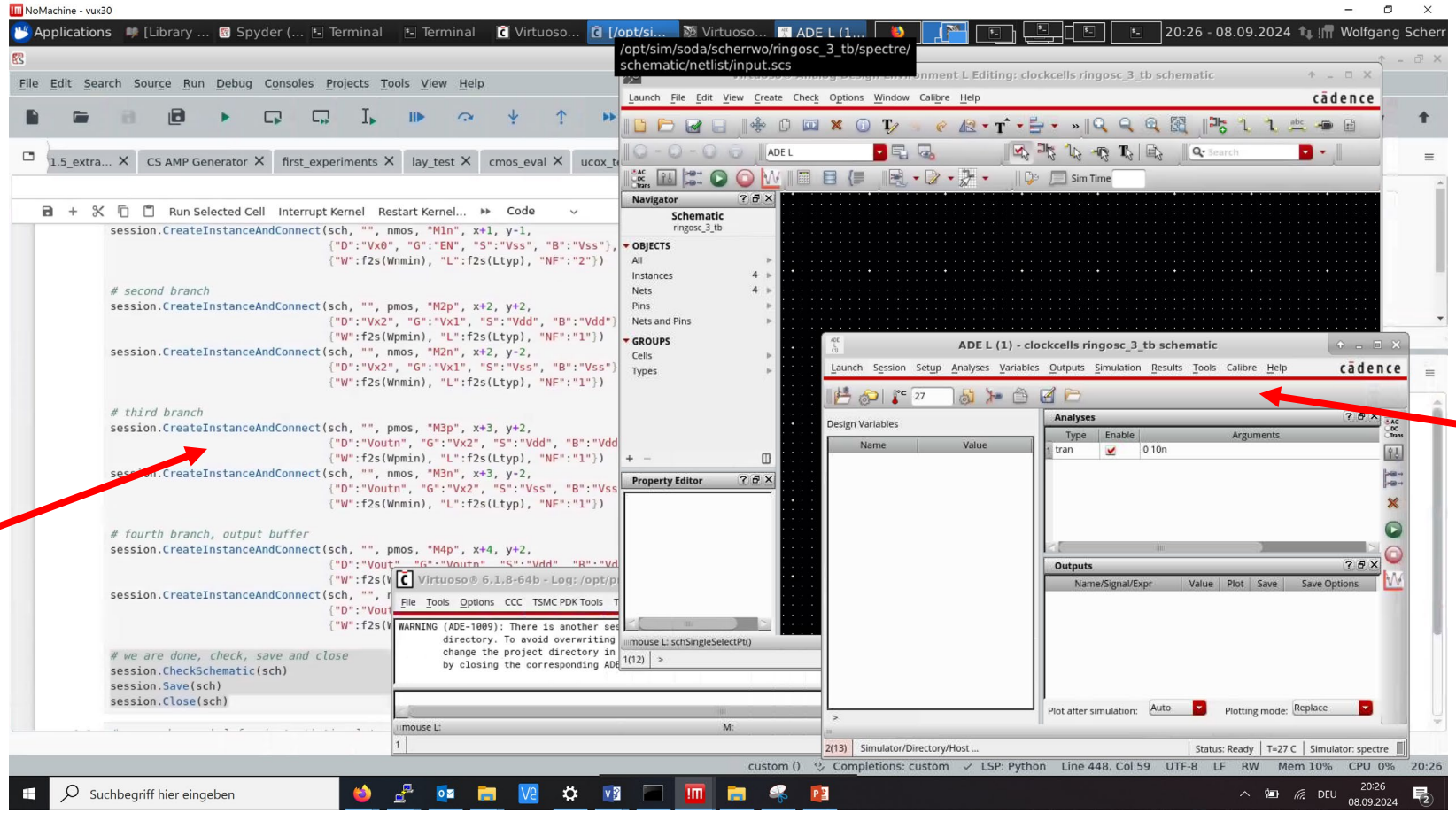
Exemplary mapping for TSMC PDKs

	tsmcN28	tsmcN65	tsmc18
MN	nch_mac	nch_mac	nmos2v_mac
MP	pch_mac	pch_mac	pmos2v_mac
MNL		nch_lvt_mac	
MPL		pch_lvt_mac	
MNH		nch_hvt_mac	
MPH		pch_hvt_mac	
MN09	nch_mac		
MP09	pch_mac		
MN12	nch_18ud12_mac	nch_mac	
MP12	pch_18ud12_mac	pch_mac	
MN15	nch_18ud15_mac		
MP15	pch_18ud15_mac		
MN18	nch_18_mac		nmos2v_mac
MP18	pch_18_mac		pmos2v_mac
MN25		nch_25_mac	
MP25		pch_25_mac	
MN33		nch_25od33_mac	nmos3v_mac
MP33		pch_25od33_mac	pmos3v_mac
MN50			
MP50			
MNH25		nch_hv25_mac	
MPH25		pch_hv25_mac	
RP	rpodwo_m	rppolywo_m	rphpoly_dis
RN	rnodwo_m	rnpolywo_m	rnhpoly_dis
CM			
CP	...	...	...
QN			
QP			

This list is not yet complete (further char. of even more devices - e.g. RF - possible).

Note: one can still use ANY library/cell from a PDK, but note that your notebook will not be technology-independent anymore.

What does agile design mean in CCC context?

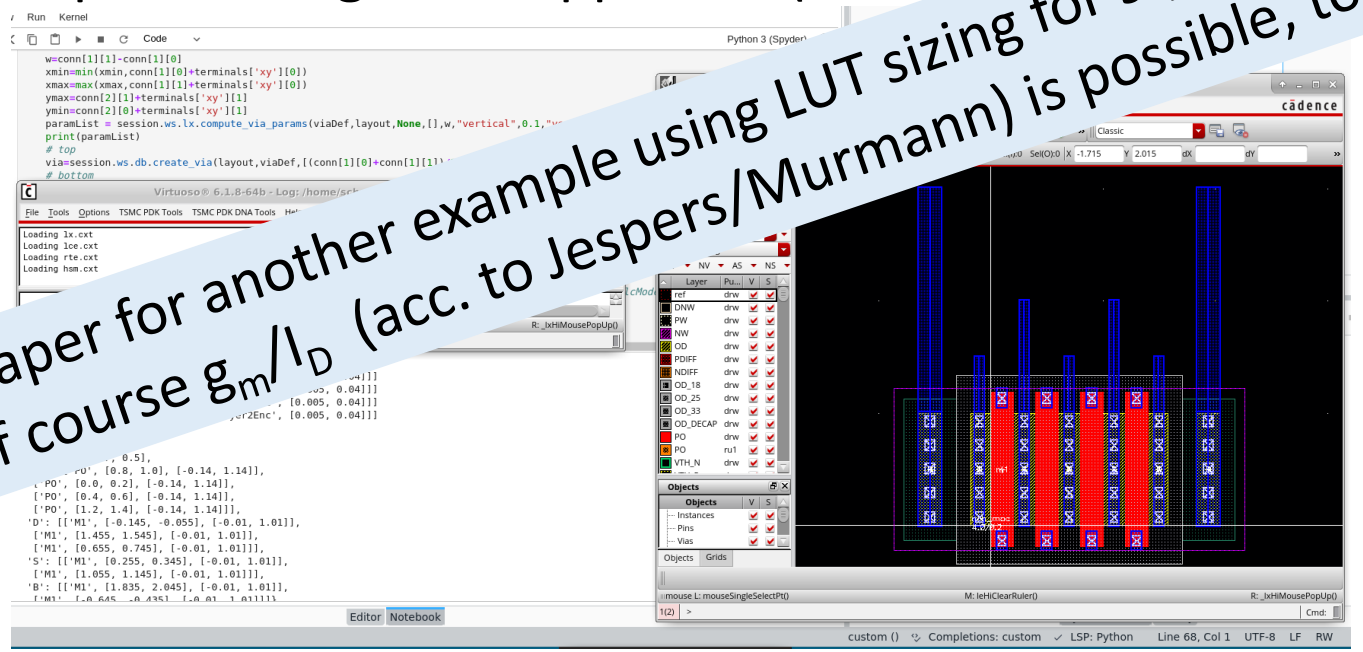


.) use scripts to
- create
- modify
schematics
.) run simulations
.) post process

.) interactive tool usage to debug the circuit
.) manual modify, simulate, etc.
.) ideally, bring correct solution back to the script

Outlook

- Open-source educational version with ng-spice planned
- Potential use of CCC for layout generation - not for re
  - A very simple “analogbase” approach (like BAG2) ple:



See also the paper for another example using LUT sizing for square-law equations.
Of course g_m/I_D (acc. to Jespers/Murmann) is possible, too.

- Own (PhD) work on layout ongoing with similar philosophy as CCC

Thank you for your attention!

- Acknowledgements:
  - SODA (System-on-Chip Design Automation) Josef-Ressel Centre for funding this work

- Special acknowledgements:
  - Skillbridge: Diss. Tobias Markus / R.-K. University Heidelberg
  - Skill/tool details: Andrew Beckett via Cadence forum

Backup slides

Typical designers' (manual) work tasks…

- Get familiar with a new technology
  - extract relevant sizing parameters
  - check out device performance
  - work out advantages and limitations

automation can significantly improve speed by avoiding to re-do the same task for every PDK (and it gets more critical for a modern tech.)

- Evaluate different circuit architectures
  - literature research, "well known" designs
  - hand calculations / estimations
  - trial setups of designs and initial simulations

automation can significantly improve speed by re-using existing designs from the past and directly map it to new PDK - also use symbolic solvers, do calculations reproducible in scripts…

- Set up final implementation
  - proper circuit sizing
  - proper circuit verification (m.c., PVT, …)
  - exhaustive documentation (allow re-use)

automation can significantly improve speed by executable optimisation and verification steps (for square-law sizing and especially gm/I_D)

Also: re-use is often limited to own circuits made in the past – or maybe some blocks done in the same group…

CCC Environment

- Allow a setup to generate with the same code circuits in multiple technologies
- Modular approach - what does this mean:
  - The framework is centrally maintained/released, users never modify that
  - Script is technology-neutral and contains the actual IP, keep it that way
  - Tool setup is also technology-dependent, encapsulate properly

this is where the user works (with data management)

.../my_project:
generator.py (or)
generator.ipynb

$PYTHONPATH

.../framework_v1/ccc:
cdsctrl.py
ccctrl.py
...

central installation like other Python packages, versioned roll-out based on release mechanism (development with data management)

separated DFII environments allowing different TECH setups (fully auto-generated and thus reproducible, no data management required, but can easily "hook" to any project DM)

.../my_project/generated1
.cdsinit
cds.lib
.../my_project/generated1/generated_lib
.../my_project/generated1/sim_data

.../my_project/generated2

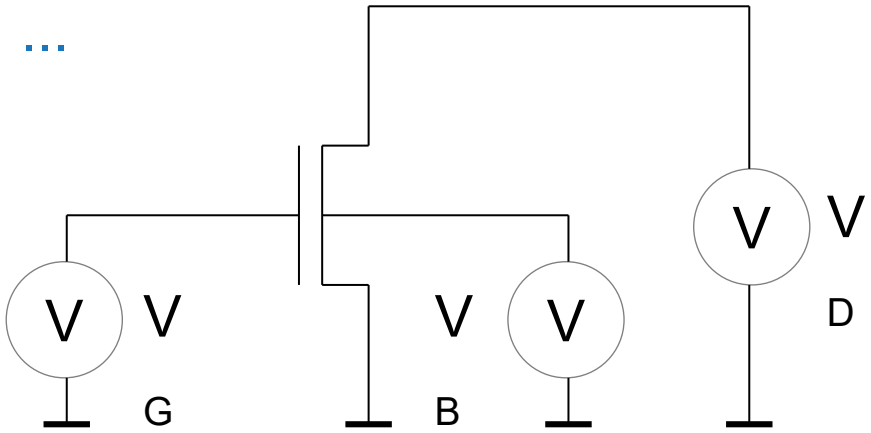
.../framework_v1/ccc/templates/tsmc65
specs.yaml
mchar_nch_mac_tsmcN65.hdf5
.cdsinit
cds.lib
...

device mapping and base tech data, characterization (LUT) data, tool setup data as required by local install

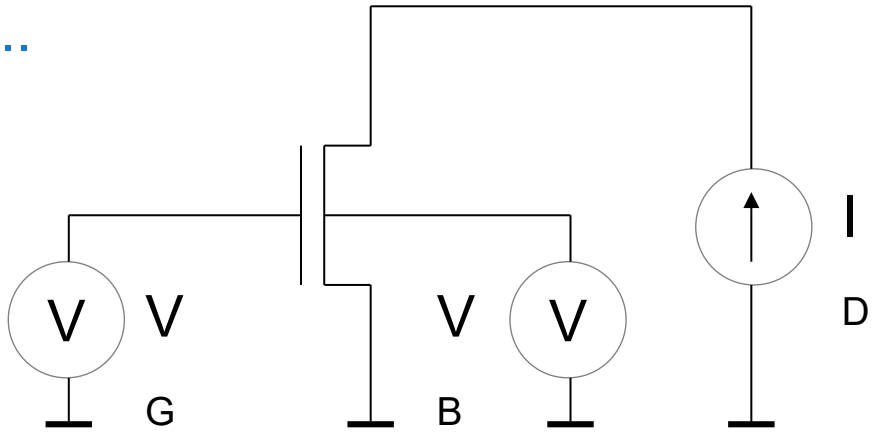
Device characterisation setups

- Exemplary MOS FET characterisation (any device/type), V-driven and I-driven channel

I_D, g_m, g_m/I_D, g_m/g_{DS}, r_o, V_{th}, ...



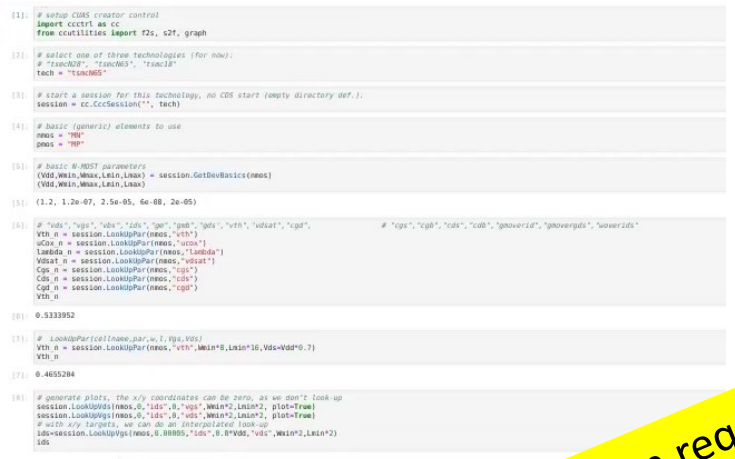
V_{DS}, g_m, g_m/I_D, g_m/g_{DS}, r_o, V_{th}, ...

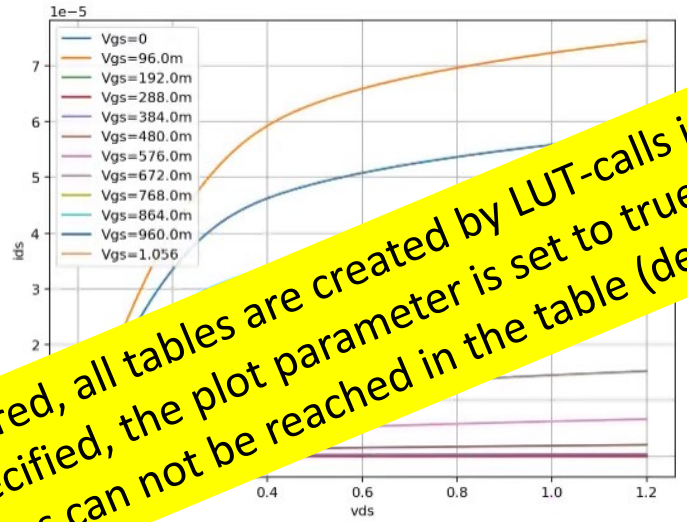


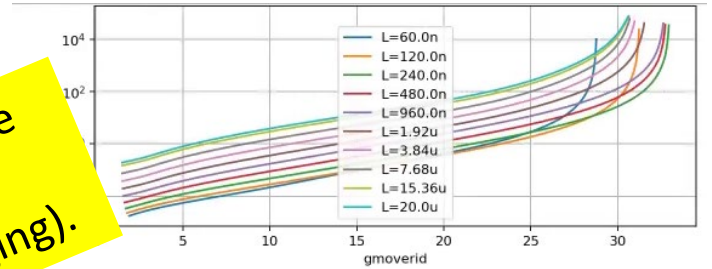
Textbook stuff (square law, gm/Id, …)

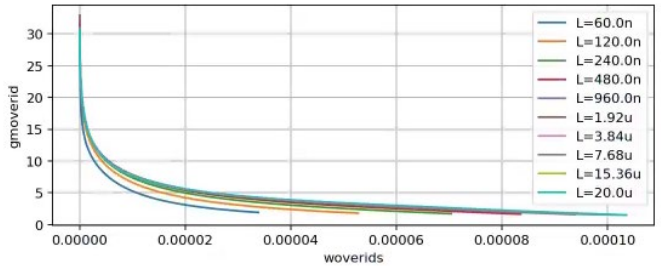
- extract all relevant MOS device parameters (large and small signal) and store them as value tables (versus the varied source) in **HDF5**
- use very same CCC framework for PDK independent characterization scripts

“Tech-learning” notebooks – get parameters for script- or hand calculations

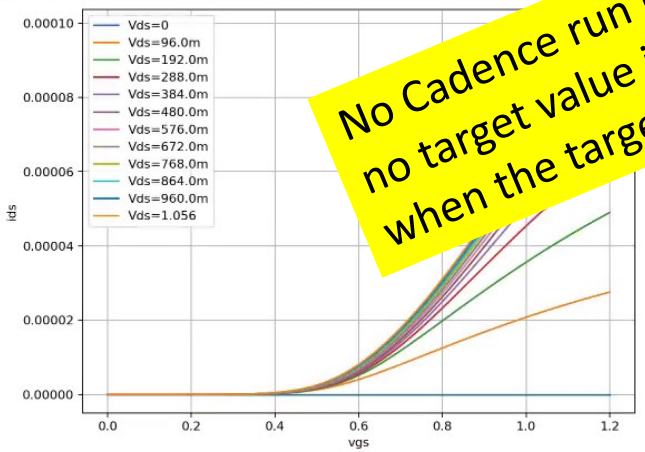


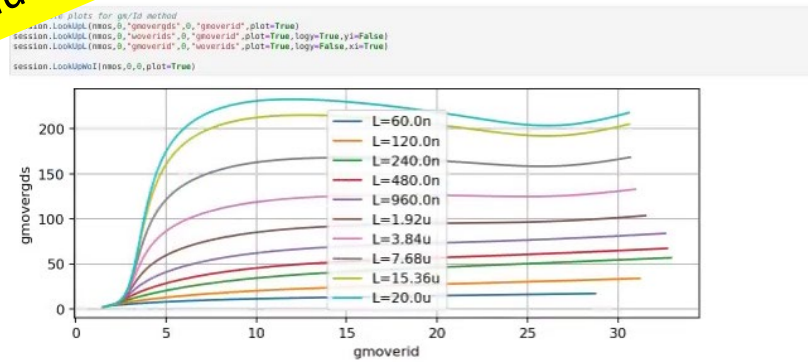


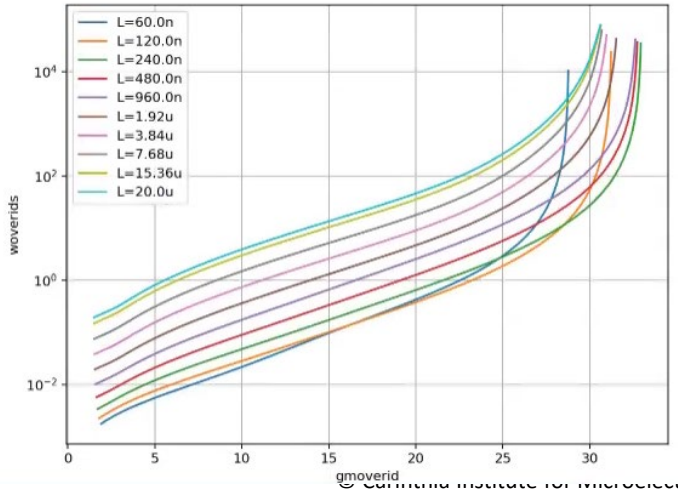




No Cadence run required, all tables are created by LUT-calls in case no target value is specified, the plot parameter is set to true or when the target values can not be reached in the table (debugging).







Examples: agile “work mode” in Spyder

```
-rw-r----- 1 scherrwo iscd-all  954680 Aug 31 20:56 first_experiments.ipynb
-rw-r----- 1 scherrwo iscd-all  368816 Aug 31 20:56 ST_design.ipynb
-rw-r----- 1 scherrwo iscd-all  117033 Aug 31 20:56 biasgen_sizing.ipynb
-rw-r----- 1 scherrwo iscd-all   89880 Aug 31 20:56 cs_amp_sizing.ipynb
```

Spyder (Python 3.10)

File Edit Search Source Run Debug Consoles Projects Tools View Help

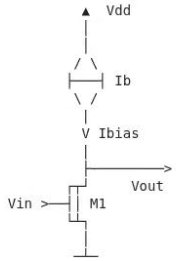
first_experiments | ST_design | cs_amp_sizing

Edit View Run Kernel

Markdown

```python
# some utilities
import numpy as np
from ccutilities import f2s, graph

# setup CUAS creator control
import ccctrl as cc
```

Circuit to set up



Specification

```python
[68]: Vo_dc = 0.5  # relative to supply voltage
Av = 50       # voltage gain
GBW = 10e6  # gain-bandwidth
Cl = 1e-12   # capacitive load

print("Av: "+f2s(Av)+"V/V; GBW="+f2s(GBW)+"Hz; Vo_DC="+f2s(Vo_dc*100)+"%Vdd; Cl="+f2s(Cl)+"F")

# target OA lib/cell
libname = "genlib"
```

Notebook | Editor

custom () Con

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

Help | Variable Explorer | Plots | Files

Console 1/A

```python
W = WoI * target_i
L=session.LookUpL("MP",target_gain,"gmovergds",target_gmoverid,"gmoverid",0,Vo_dc*Vdd,plot=False)
print("L",f2s(L)+"m")
print("W:",f2s(W)+"m")
```

```
gm_p: 30.0uS
L 208.8nm
W: 1.346um
```

```python
[8]: # ==========
# We are done with the session
del session
```

Virtuoso® 6.1.8-64b - Log: /home/ma/scherrwo/ccc/dev/ccc/examples/my_first_project_tsmcN65/

File Tools Options CCC Help — cādence

StartSrv
StopSrv
StopSrv/Exit

```
pyRunScript pa ?python "python"
start skillbridge on data in /opt/sim/iscd/scherrwo
Loading pe.cxt
*WARNING* could no -courier-medium-r-*-*-12-*", using font "fixed"
```

mouse L: M: R:

Create circuit and symbol

```python
[9]: session = cc.CccSession("my_first_project",tech)
```

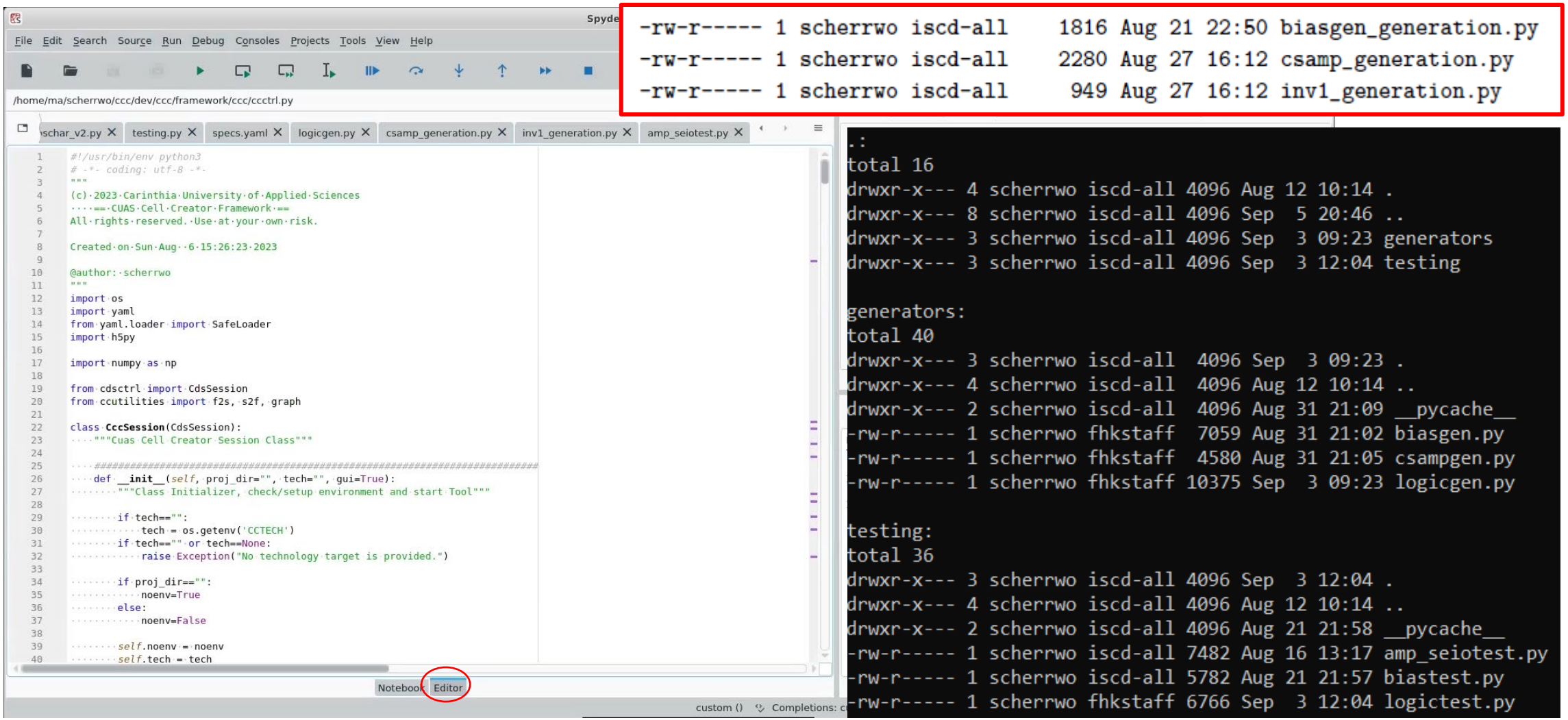
```
Launch Virtuoso and Python server...
@(#)$CDS: virtuoso version 6.1.8-64b 10/01/2018 20:02 (ip-172-18-22-57) $
...done.
```

```python
[11]: # If the library does not exist, it will be created. An existing schematic will be overwritten.
x=0
y=0
```

Con

```
Python 3.
Type "cop
IPython 8
In [1]:
```

Examples: waterfall “work mode” in Spyder



ST design – sizing 1/2

Josef Ressel Centre for System-on-Chip Design Automation

Schmitt-Trigger Design

```
(c) 2023 Carinthia University of Applied Sciences
    == CUAS Cell Creator Framework ==
All rights reserved. Use at your own risk.

Author: W. Scherr
```

[1]:
```python
import sys
# FRAMEWORK
sys.path.insert(0,"/opt/cadadm/dessup/scherrwo/CUAS/ccc_v0.3/framework/ccc")

# setup CUAS creator control
import ccctrl as cc
from ccutilities import f2s, graph
```

[2]:
```python
# We design relative to any technology core supply, thus parameters are relative as well.
# Typical CMOS switching levels can be retrieved from JEDEC:
# E.g.: JESD8-12A.01, November 2005, JC-16 Committee on Interface Technology,
# "1.2 V +/- 0.1 V (NORMAL RANGE) AND 0.8 - 1.3 V (WIDE RANGE) POWER SUPPLY VOLTAGE
#  AND INTERFACE STANDARD FOR NONTERMINATED DIGITAL INTEGRATED CIRCUITS",
#  JEDEC Board Ballot JCB-00-74 and JCB-05-80.
#
# ST spec: upper Vth: 0.35...0.75Vdd, lower Vth: 0.25...0.65Vdd, Vh: 0.1...0.5Vdd
# Thus, we set up everything nicely "in the middle" of the allowed spec (application engineer):
Vh = 0.3    #  (Hysteresis)
Vm = 0.5
Vih = Vm+Vh/2  #    (min wide range)
Vil = Vm-Vh/2  #    (max wide range)

#tech = "tsmcN28"
#tech = "tsmcN65"
tech = "tsmc18"

# basic (generic) elements to use, if one wants e.g. a specific 1.2V design, use MN12 and MP12
nmos = "MN"
pmos = "MP"

# note, the generator is PDK INDEPENDENT! Thus, we
# ideally deal with multiples of minimum feature sizes
# we will focus on a small, minimum W/L design...
# so: Wmin*2^Wpot=Wmin and Lmin*2^Lpot=Lmin
Wpot=0
Lpot=0
```

[3]:
```python
# start a session for this technology, no CDS start, so we need no environment dir.
session = cc.CccSession("", tech, noenv=True)

# basic MOST parameters, you may also do some plausibility checks here (just in case...)
(Vdd,Wmin,Wmax,Lmin,Lmax)=session.GetDevBasics(nmos)
(Vdd2,Wmin,Wmax,Lmin,Lmax)=session.GetDevBasics(pmos)
if not Vdd==Vdd2:
    Vdd=min(Vdd,Vdd2)
    print("NMOS and PMOS have different voltage classes, selecting lower one.")

# now get absolute sizing values
Wtyp = Wmin*pow(2,Wpot)
Ltyp = Lmin*pow(2,Lpot)

# this gives us relatively close parameters already
Vth_n = session.LookUpPar(nmos,"vth",Wtyp,Ltyp)
k_n = session.LookUpPar(nmos,"ucox",Wtyp,Ltyp)
Vdsat_n = session.LookUpPar(nmos,"vdsat",Wtyp,Ltyp)
Vth_p = session.LookUpPar(pmos,"vth",Wtyp,Ltyp)
k_p = session.LookUpPar(pmos,"ucox",Wtyp,Ltyp)
Vdsat_p = session.LookUpPar(pmos,"vdsat",Wtyp,Ltyp)

#print("Technology specification ('"+tech+"'):")
print("         Vdd="+f2s(Vdd)+"V, Wmin="+f2s(Wmin)+"m, Lmin="+f2s(Lmin)+"m")
print("         Wmax="+f2s(Wmax)+"m, Lmax="+f2s(Lmax)+"m")
print("         Wtyp="+f2s(Wtyp)+"m, Ltyp="+f2s(Ltyp)+"m")
(lib,cell) = session.look_up_cell(nmos)
print("         "+nmos+": "+lib+", "+cell+" vdsat="+f2s(Vdsat_n)+"V")
print("             k="+f2s(k_n)+"A/V², vth="+f2s(Vth_n)+"V")
(lib,cell) = session.look_up_cell(pmos)
print("         "+pmos+": "+lib+", "+cell+" vdsat="+f2s(Vdsat_p)+"V")
print("             k="+f2s(k_p)+"A/V², vth="+f2s(Vth_p)+"V")
print("\n  Spec:  Vih:",f2s(Vih*Vdd)+"V Vil:",f2s(Vil*Vdd)+"V")
```

```
         Vdd=1.8V, Wmin=220.0nm, Lmin=180.0nm
         Wmax=900.0um, Lmax=19.9um
         Wtyp=220.0nm, Ltyp=180.0nm
         MN: tsmc18, nmos2v_mac vdsat=223.3mV
             k=252.1uA/V², vth=485.6mV
         MP: tsmc18, pmos2v_mac vdsat=251.8mV
             k=102.7uA/V², vth=547.9mV

  Spec:  Vih: 1.17V Vil: 630.0mV
```

ST design - sizing 2/2

Example results:

```
180nm:
    W1-6: 220.0nm 220.0nm 259.6nm 657.3nm 657.3nm 640.8nm
    All L: 180.0nm    area:  477896.6nm²
65nm:
    W1-6: 348.1nm 348.1nm 120.0nm 699.3nm 699.3nm 241.2nm
    All L: 60.0nm     area:  147353.6nm²
28nm:
    W1-6: 112.6nm 112.6nm 100.0nm 451.8nm 451.8nm 164.2nm
    All L: 30.0nm     area:   41789.1nm²
```

[4]:
```
# we don't need the session anymore
del session
```

[5]:
```
# Any textbook show the same here, e.g. the classics:
# J. M. Rabaey, "Digital Integrated Circuits", 1st ed., Prentice Hall, pg 149ff
#
# check, if both devices are in velocity saturation
vsat=True
if (Vdd*Vm-Vth_n)<=Vdsat_n:
    vsat=False
if (Vdd-Vdd*Vm-Vth_p)<=Vdsat_p:
    vsat=False
if vsat:
    print("Calculate ratio for velocity saturation")
    k5overk2 = (k_n*Vdsat_n*(Vdd*Vm-Vth_n-Vdsat_n/2))/(k_p*Vdsat_p*(Vdd-Vdd*Vm-Vth_p-Vdsat_p/2))
else:
    k5overk2 = (k_n*(Vdd*Vm-Vth_n))/(k_p*(Vdd-Vdd*Vm-Vth_p))
print("k5/k2: "+f2s(k5overk2))
```

```
Calculate ratio for velocity saturation
k5/k2: 2.913
```

[6]:
```
# I. M. Filanovsky, H. Baltes, "CMOS Schmitt Trigger Design",
# IEEE transactions on circuits and systems, Vol. 41, Nr. 1, Jan. 1994, pg. 46ff
#
#
k1overk3 = pow((Vdd-Vdd*Vih)/(Vdd*Vih-Vth_n),2)
k4overk6 = pow(Vdd*Vil/(Vdd-Vdd*Vil-Vth_p),2)
print("k4/k6: "+f2s(k4overk6)+", k1/k3: "+f2s(k1overk3))
```

```
k4/k6: 1.026, k1/k3: 847.4m
```

[7]:
```
L=Ltyp
if k5overk2>=1:
    # P is larger than N (standard...)
    Wpmin=Wtyp*k5overk2
    Wnmin=Wtyp
else:
    # N is larger than P (strange...)
    printf("Hmmm, N is weaker than P devices?")
    Wpmin=Wtyp
    Wnmin=Wtyp/k5overk2
if k4overk6>=1:
    # P branch is larger
    W4=k4overk6*Wpmin
    W5=k4overk6*Wpmin
    W6=Wpmin
else:
    # P feedback is larger
    W4=Wpmin
    W5=Wpmin
    W6=Wpmin/k4overk6
if k1overk3>=1:
    # N branch is larger
    W1=k1overk3*Wnmin
    W2=k1overk3*Wnmin
    W3=Wnmin
else:
    # N feedback is larger
    W1=Wnmin
    W2=Wnmin
    W3=Wnmin/k1overk3
area = (W1+W2+W3+W4+W5+W6)*L
print("W1-6:",f2s(W1)+"m",f2s(W2)+"m",f2s(W3)+"m",f2s(W4)+"m",f2s(W5)+"m",f2s(W6)+"m")
print("All L:",f2s(L)+"m   ","area: ",f2s(area,2)+"m²")
```

```
W1-6: 220.0nm 220.0nm 259.6nm 657.3nm 657.3nm 640.8nm
All L: 180.0nm    area:  477896.6nm²
```

ST design – implementation & TB

Josef Ressel Centre for System-on-Chip Design Automation

```
[8]: # start a session for this technology, now with CDS.
session = cc.CccSession("st_project", tech)
libname="digcells"
cellname="st_inv"
```

```
Setup project directory: ./st_project_tsmc18/
Adding Virtuoso control code.
Launch Virtuoso and Python server...
@(#)$CDS: virtuoso version 6.1.8-64b 10/01/2018 20:02 (ip-172-18-22-57) $
...done.
```

```
[9]: # If the library does not exist, it will be created. An existing schematic will be overwritten.
x=0
y=0
sch=session.CreateSchematic(libname, cellname)
# supply pins
session.CreatePin(sch, "Vdd", "input", x,y+1)
session.CreatePin(sch, "Vss", "input", x,y-1)
# I/O pins
session.CreatePin(sch, "Vin", "input", x,y)
session.CreatePin(sch, "Vout", "output", x+3,y)
# first branch
session.CreateInstanceAndConnect(sch, "", pmos, "M4", x+1, y+2,
                                 {"D":"Vp", "G":"Vin", "S":"Vdd", "B":"Vdd"},
                                 {"W":f2s(W4), "L":f2s(L), "NF":"1"})
session.CreateInstanceAndConnect(sch, "", pmos, "M5", x+1, y+1,
                                 {"D":"Vout", "G":"Vin", "S":"Vp", "B":"Vdd"},
                                 {"W":f2s(W5), "L":f2s(L), "NF":"1"})
session.CreateInstanceAndConnect(sch, "", nmos, "M2", x+1, y-1,
                                 {"D":"Vout", "G":"Vin", "S":"Vn", "B":"Vss"},
                                 {"W":f2s(W2), "L":f2s(L), "NF":"1"})
session.CreateInstanceAndConnect(sch, "", nmos, "M1", x+1, y-2,
                                 {"D":"Vn", "G":"Vin", "S":"Vss", "B":"Vss"},
                                 {"W":f2s(W1), "L":f2s(L), "NF":"1"})
# feedback branch
session.CreateInstanceAndConnect(sch, "", pmos, "M6", x+2, y+1,
                                 {"D":"Vss", "G":"Vout", "S":"Vp", "B":"Vdd"},
                                 {"W":f2s(W6), "L":f2s(L), "NF":"1"})
session.CreateInstanceAndConnect(sch, "", nmos, "M3", x+2, y-1,
                                 {"D":"Vdd", "G":"Vout", "S":"Vn", "B":"Vss"},
                                 {"W":f2s(W3), "L":f2s(L), "NF":"1"})
# we are done, check, save and close
session.CheckSchematic(sch)
session.Save(sch)
session.Close(sch)
```

Outlook: use just a SPICE representation instead of an programmatic API for instances & connections.

(this API will be anyhow used in the background by the SPICE reader…)

```
[10]: # we need a symbol for instantiation later
session.CreateSymbol(libname, cellname)
```

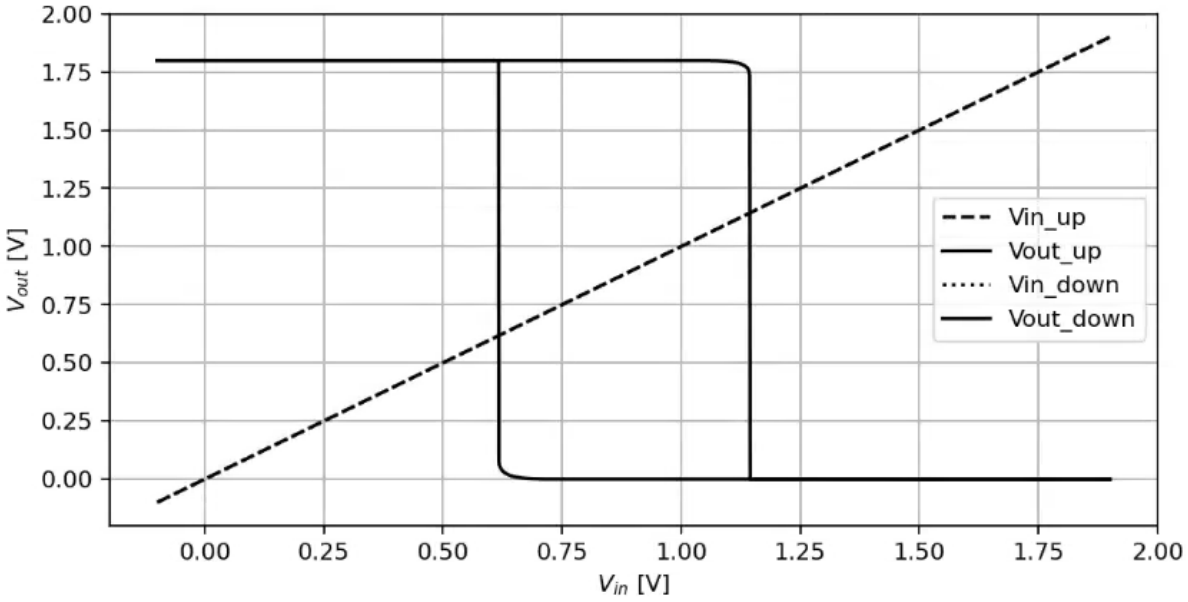
```
[11]: # setup the testbench
x=0
y=0
tb=session.CreateSchematic(libname, cellname+"_tb")
# supply
session.CreateInstanceAndConnect(tb, "analogLib", "vdc", "Vsup", x, y+1,
                                 {"PLUS":"Vdd", "MINUS":"Vss"},
                                 {"vdc":"v_sup"})
session.CreateInstanceAndConnect(tb, "analogLib", "vdc", "Vgnd", x, y,
                                 {"PLUS":"Vss", "MINUS":"gnd!"},
                                 {"vdc":"0.0"})
session.CreateInstanceAndConnect(tb, "analogLib", "gnd", "Refnode", x, y-1,
                                 {"gnd!":"gnd!"})
# stimulus
session.CreateInstanceAndConnect(tb, "analogLib", "vdc", "Vin", x+1, y,
                                 {"PLUS":"Vin", "MINUS":"Vss"},
                                 {"vdc":"v_in"})
# DUT
session.CreateInstanceAndConnect(tb, libname, cellname, "DUT", x+2, y,
                                 {"Vdd":"Vdd","Vss":"Vss","Vin":"Vin","Vout":"Vout"})
# we are done, check, save and close
session.CheckSchematic(tb)
session.Save(tb)
session.Close(tb)
```

ST design – verification (simple DC)

It is obvious, this code can be easily re-used for many purposes of I/O diagram simulation/extraction…

```
[12]: # DC sweep over supply (up)
session.SimulateDC(libname, cellname+"_tb", "Vin", "dc", -0.1, Vdd+0.1, 0.001,
                   {"v_sup":f2s(Vdd),"v_in":f2s(0.0)})
# fetch results
(vin1,vout1)=session.GetResult("Vout")
# DC sweep over supply (down)
session.SimulateDC(libname, cellname+"_tb", "Vin", "dc", Vdd+0.1, -0.1, -0.001,
                   {"v_sup":f2s(Vdd),"v_in":f2s(0.0)})
# fetch results
(vin2,vout2)=session.GetResult("Vout")
```

```
[13]: data = [
        [vin1,vin1,'k--','Vin_up'],
        [vin1,vout1,'k','Vout_up'],
        [vin2,vin2,'k:','Vin_down'],
        [vin2,vout2,'k','Vout_down']
        ]
graph(data,'$V_{in}$ [V]','$V_{out}$ [V]',size=(8,4))

for vi, vo in zip(vin1, vout1):
    if vo<=Vdd/2:
        Vih_m = vi
        break
for vi, vo in zip(vin2, vout2):
    if vo>=Vdd/2:
        Vil_m = vi
        break
print("\n  Result:  Vih:",f2s(Vih_m)+"V   Vil:",f2s(Vil_m)+"V   Vh:",f2s(Vih_m-Vil_m)+"V")
print("\n           Vih:",str(int(1000*Vih_m/Vdd)/10)+"%   Vil:",str(int(1000*Vil_m/Vdd)/10)+
                                                "%   Vh:",str(int(1000*(Vih_m-Vil_m)/Vdd)/10)+"%")
```

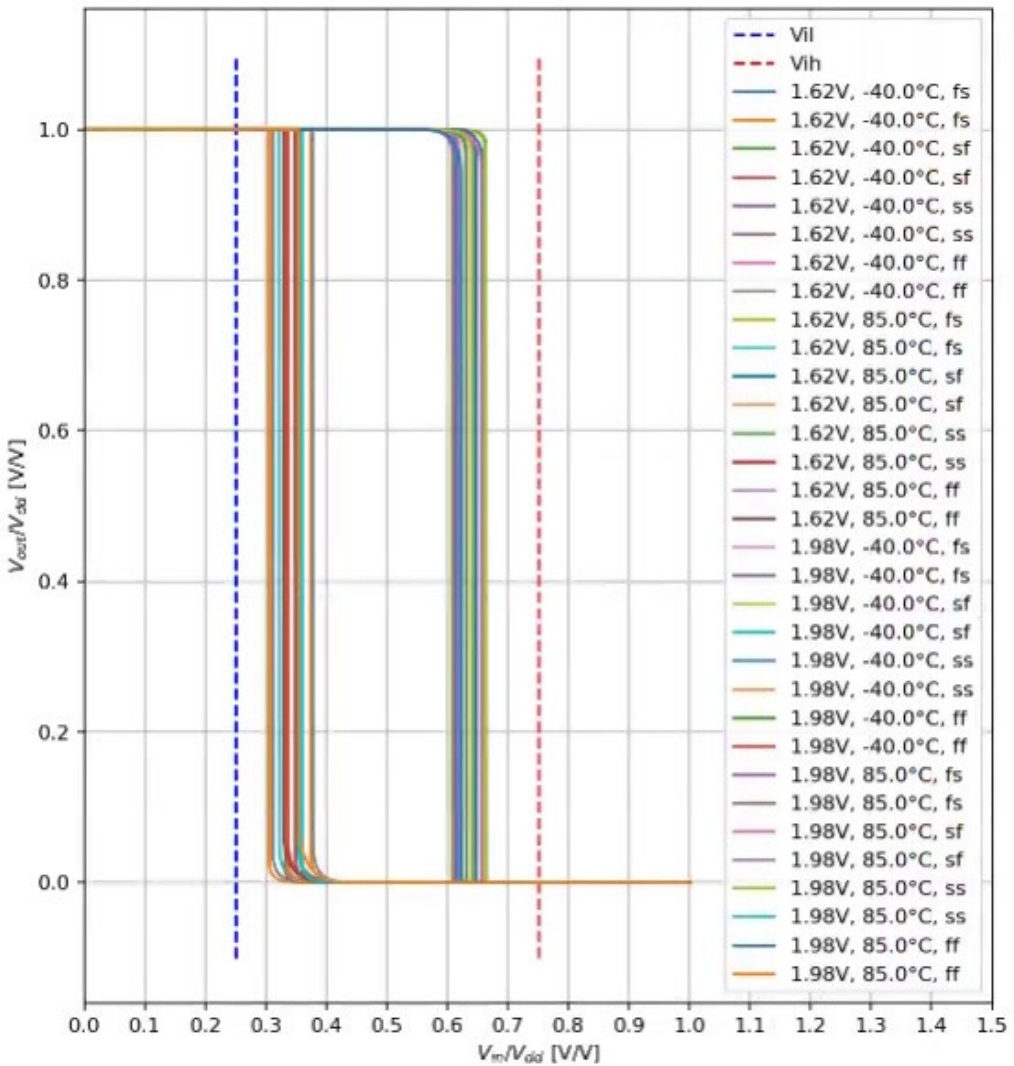



```
Result:  Vih: 1.145V   Vil: 617.0mV   Vh: 528.0mV

         Vih: 63.6%   Vil: 34.2%   Vh: 29.3%
```

ST design – verification (PVT DC)

```
[14]: # PVT analysis, normalised plot
d = []
d.append([[0.25,0.25],[-0.1,1.1],'b--','Vil'])
d.append([[0.75,0.75],[-0.1,1.1],'r--','Vih'])
for v in [Vdd*0.9,Vdd*1.1]:
    for t in [-40.0,85.0]:
        for c in ['fs','sf','ss','ff']:
            # DC sweep over supply (up)
            session.SimulateDC(libname, cellname+"_tb", "Vin", "dc", 0.0, v, 0.001,
                               {"v_sup":f2s(v),"v_in":f2s(0.0)}, temp=t, corner=c)
            # fetch results, scale to 100%
            (vi,vo)=session.GetResult("Vout")
            vi = [i/v for i in vi]
            vo = [i/v for i in vo]
            d.append([vi,vo,'',f2s(v)+'V, '+f2s(t)+'°C, '+c])
            # DC sweep over supply (down)
            session.SimulateDC(libname, cellname+"_tb", "Vin", "dc", v, 0.0, -0.001,
                               {"v_sup":f2s(v),"v_in":f2s(0.0)}, temp=t, corner=c)
            # fetch results, scale to 100%
            (vi,vo)=session.GetResult("Vout")
            vi = [i/v for i in vi]
            vo = [i/v for i in vo]
            d.append([vi,vo,'',f2s(v)+'V, '+f2s(t)+'°C, '+c])
graph(d,'$V_{in}/V_{dd}$ [V/V]','$V_{out}/V_{dd}$ [V/V]',limx=[0.0,1.5,16],size=(8,9))
```

```
[15]: del session

All closed.
```



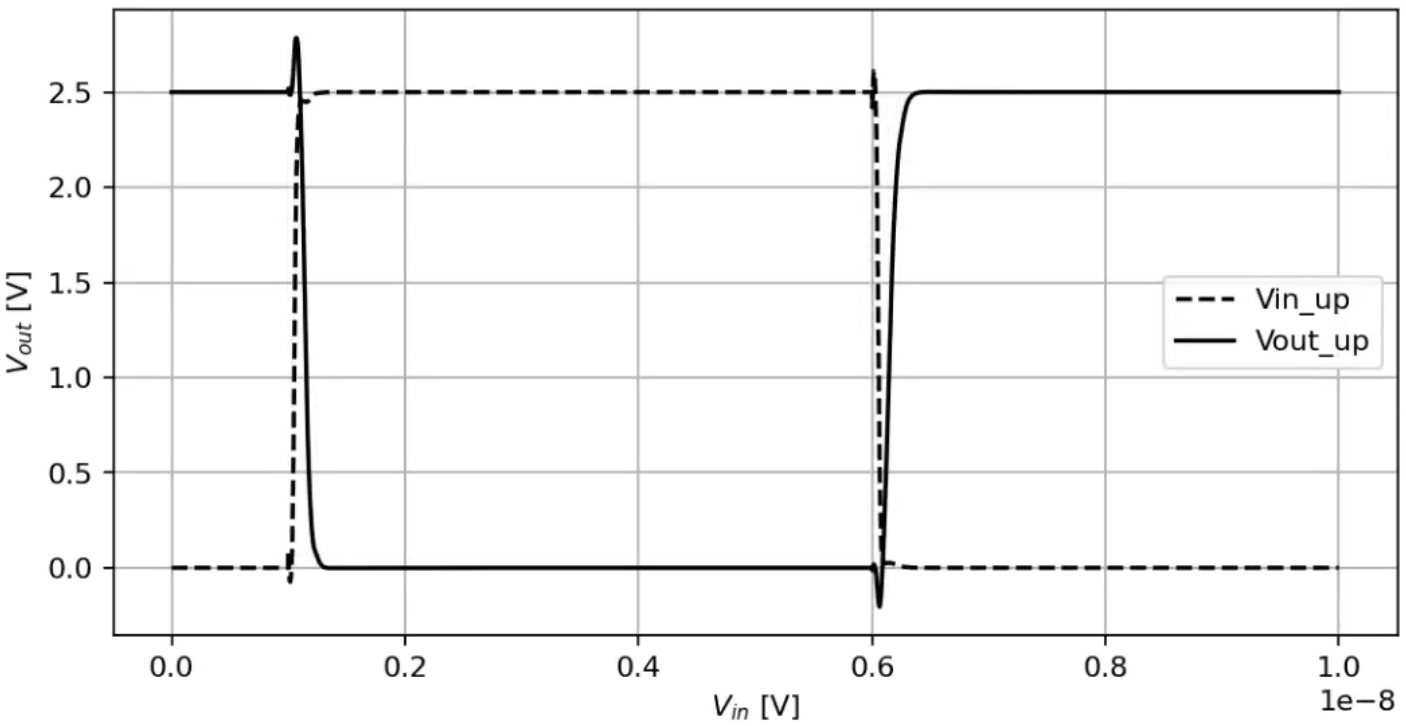
We do a nice normalisation, so we can compare results between technologies with different supply voltages….

ST design – verification (simple timing)

Again, one can see that this is easy to extend for PVT runs as well…

```
[15]: # TRAN (we measure between "DRIVER" and "LOAD")
session.SimulateTRAN(libname, cellname+"_tb", "10n",
                     {"v_sup":f2s(Vdd),"v_in":f2s(Vdd)})
# fetch results
(t,vin)=session.GetResult("Vin","tran")
(t,vout)=session.GetResult("Vout","tran")
```

```
[16]: data = [
        [t,vin,'k--','Vin_up'],
        [t,vout,'k','Vout_up']        ]
graph(data,'$V_{in}$ [V]','$V_{out}$ [V]',size=(8,4))

pos=0
for tt, vi, vo in zip(t, vin, vout):
    if vi>=Vdd/2 and pos==0:
        t1=tt
        pos=1
    if vo<=Vdd/2 and pos==1:
        t2=tt
        pos=2
    if vi<=Vdd/2 and pos==2:
        t3=tt
        pos=3
    if vo>=Vdd/2 and pos==3:
        t4=tt
        pos=4
        break

if pos==4:
    print("Result:  Delay falling:",f2s(t2-t1)+"s, rising:",f2s(t4-t3)+"s")
else:
    raise Exception("Did not detect two edges.")
```

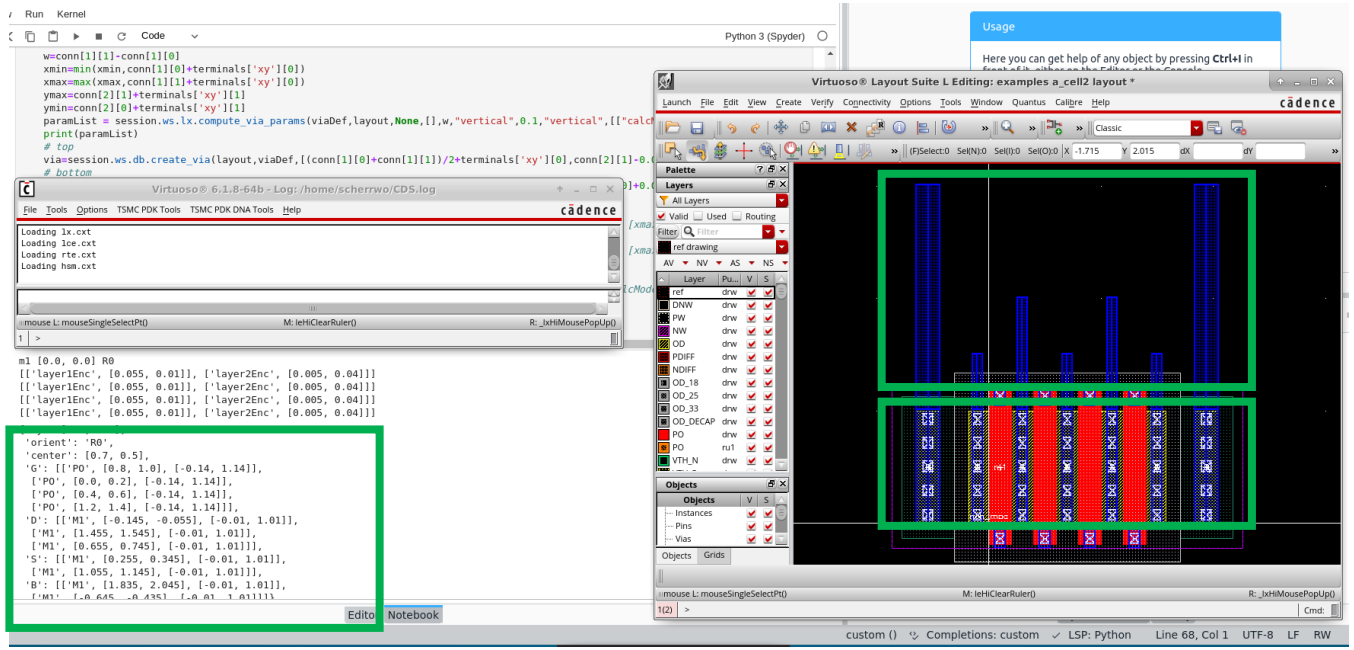



```
Result:  Delay falling: 95.61ps, rising: 117.4ps
```

Later, one can “copy over” this notebook into a Python asset class for verification of any basic digital I/O block…

Outlook

- Potential use of CCC for layout generation - not for release yet
- Own (PhD) work on layout ongoing with same philosophy as CCC
- A very simple “analogbase” approach (like BAG) is shown as example:



CCC can also extract all relevant Pcell coords.

CCC can place & connect Paths, Vias, …

CCC can place & configure PCells