8-Bit Successive Approximation Register (SAR) ADC: Exploring Mixed-Signal Design with Xschem

Phillip Ferreira Baade–Pedersen (Design Engineer)
Dr. Ing. Christian Wittke (Scientist)
Frankfurt (Oder), IHP - 22/05/2025

Projects: BMBF -> FMD-QNC (16ME0831)

Leibniz Institute
for high
performance
microelectronics

Agenda For today

09:00 - 09:15 | Recap | Questions?
Review of Key Takeaways from Yesterday

09:15 – 09:30| Module Overview
Overview and Introduction to Today's Module

09:30 - 10:00 | Introduction to Mixed Signal Design In Xschem
A small presentation on how to perform Mixed signal simulations in Xschem

10:00 - 12:00 | Building Blocks for 8-Bit SAR ADC
Implementing circuits based on a provided task list for the 8-bit SAR ADC.

12:00 - 13:00 | Lunch Brake | Catching Up

13:00 - 15:00 | Mixed Signal Simulation
Run simulations on the Verilog code and integrate it into the final ADC schematic.

15:00 – 15:30 | Coffee Break | Caching Up

15:30 - 17:00 | Mixed Signal Simulation
Pick up where we left off and complete the design work.

Leibniz Institute
for high
performance
microelectronics

Key Takeaways from Yesterday

- Overview of Qucs-S
- Understanding Issues with Xyce/QUCS-S
- Electromagnetic (EM) Simulation with OpenEMS
- Post-EM Simulation Analysis
- Developing RF Workflow Skills in OS

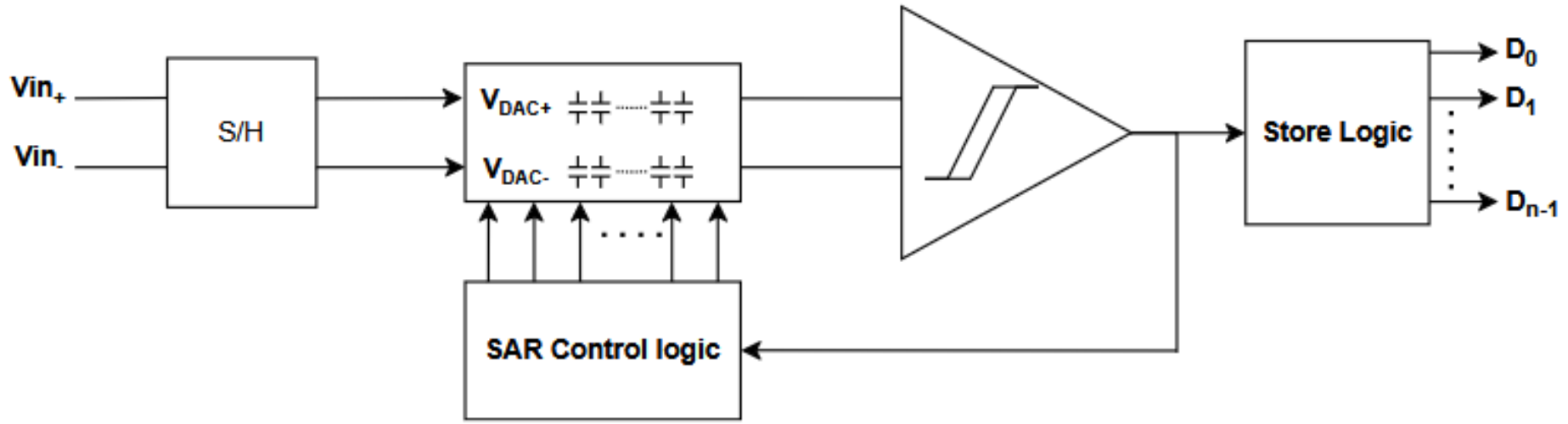
Questions?

- Was something from yesterday unclear?
- Are we going to fast?
- Do you need more time for the small exercise?

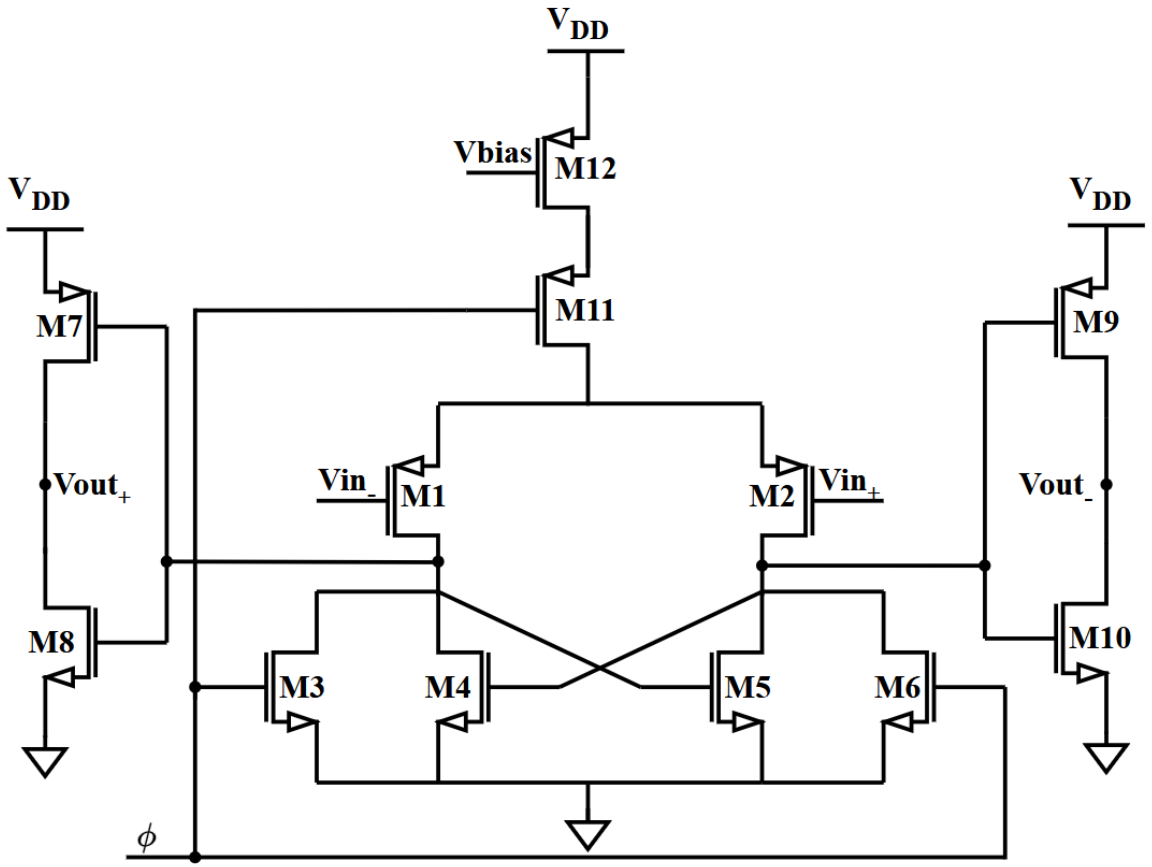
Overview Of Todays Plan

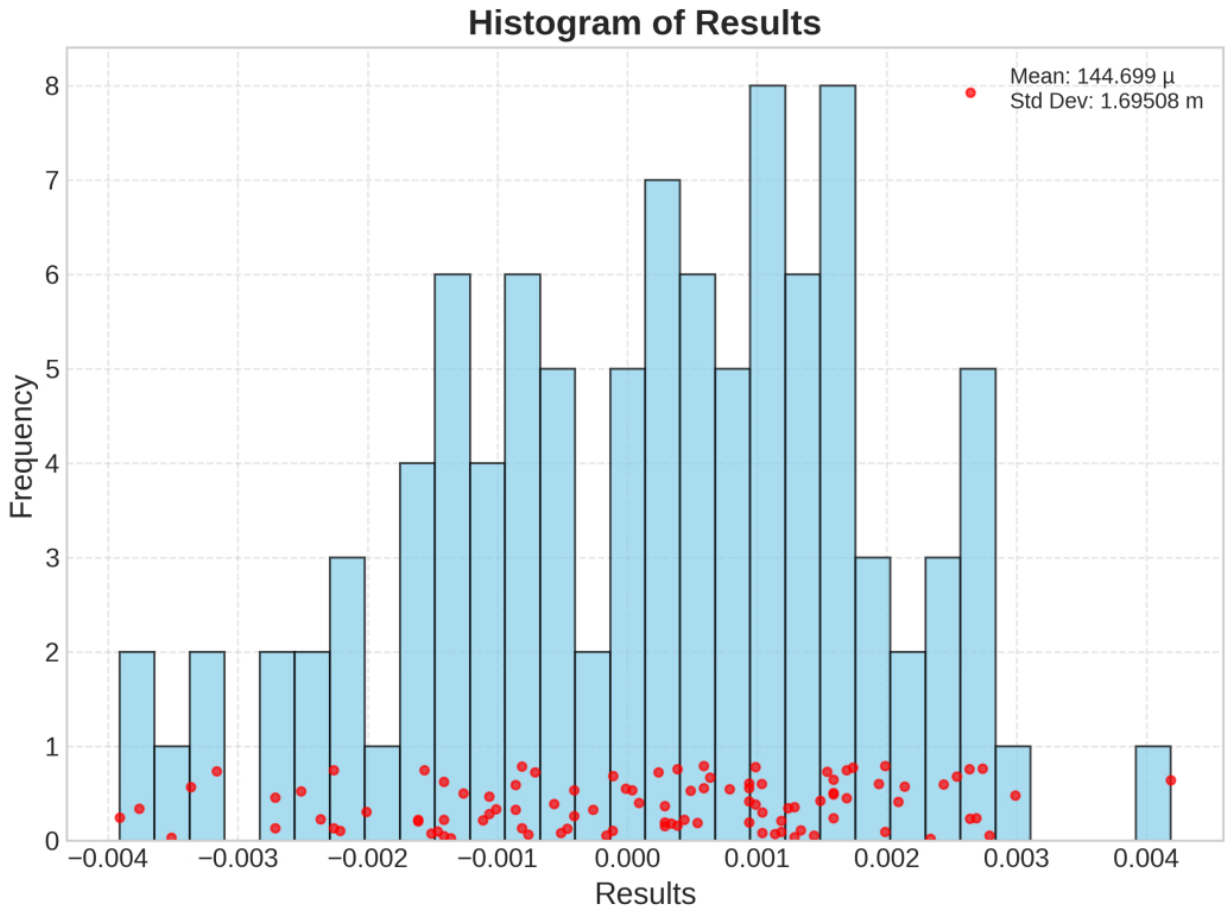
- **Review of Key Skills:** Applying knowledge from previous sessions to create small circuits for the SAR ADC, including Monte Carlo analysis.
- **Digital Simulations:** Writing and simulating Verilog code, including the development of testbenches
- **AMS Workflow:** Establishing an Analog-Mixed Signal (AMS) workflow using Xschem
- **Misc:** Continuing with the layout of the OTA

The SAR ADC

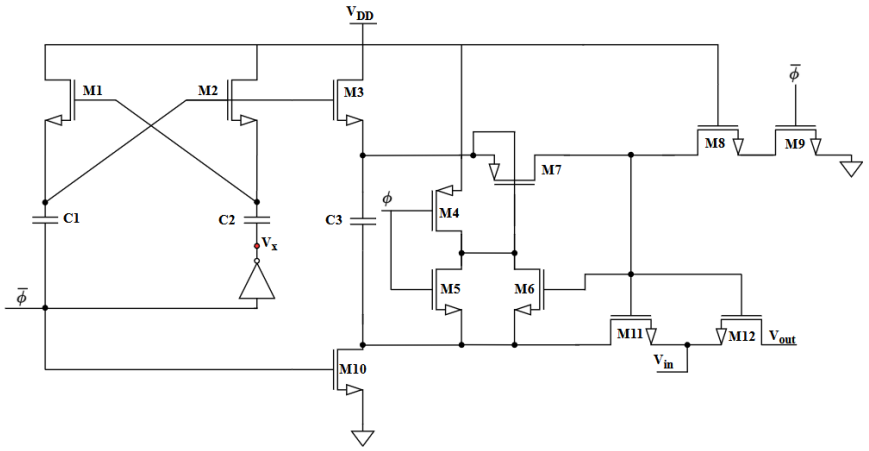


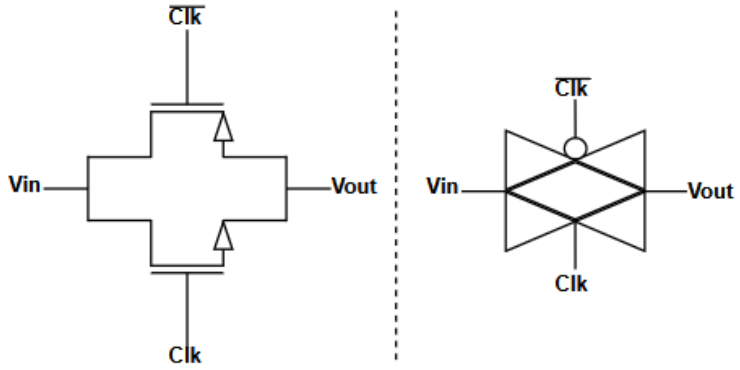
The SAR ADC

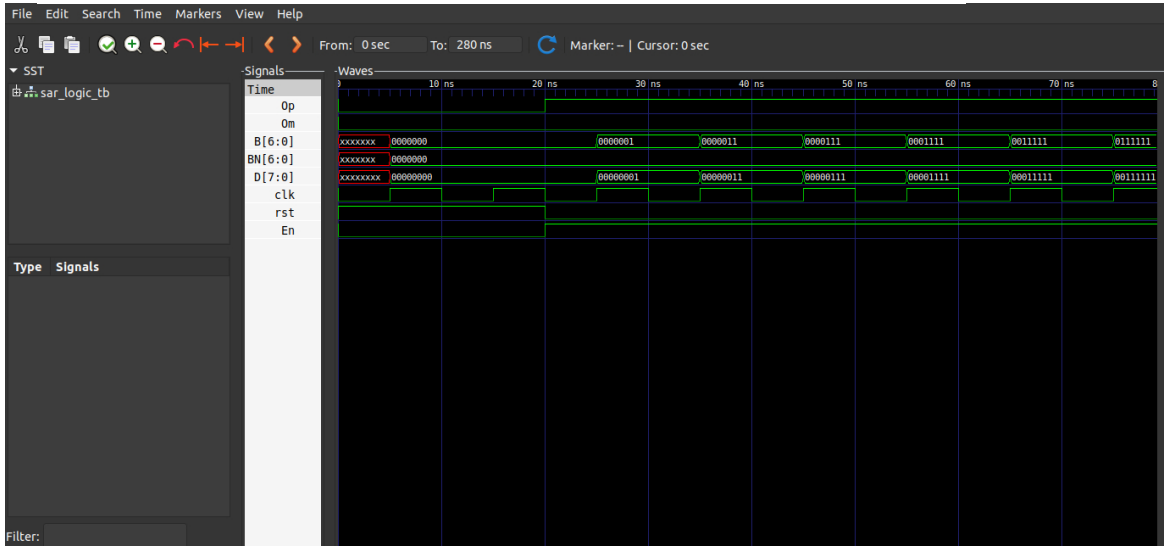

V_DD
Vbias
M12
M11
M7
M9
Vout+
Vin-
M1
M2
Vin+
Vout-
M8
M10
M3
M4
M5
M6
ϕ

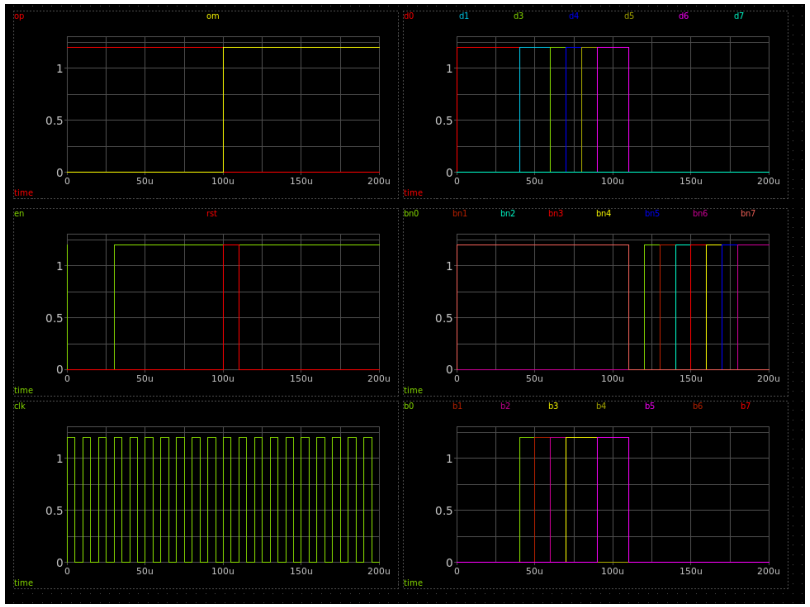

Histogram of Results
Mean: 144.699 µ
Std Dev: 1.69508 m
Frequency
Results
0
1
2
3
4
5
6
7
8
−0.004
−0.003
−0.002
−0.001
0.000
0.001
0.002
0.003
0.004

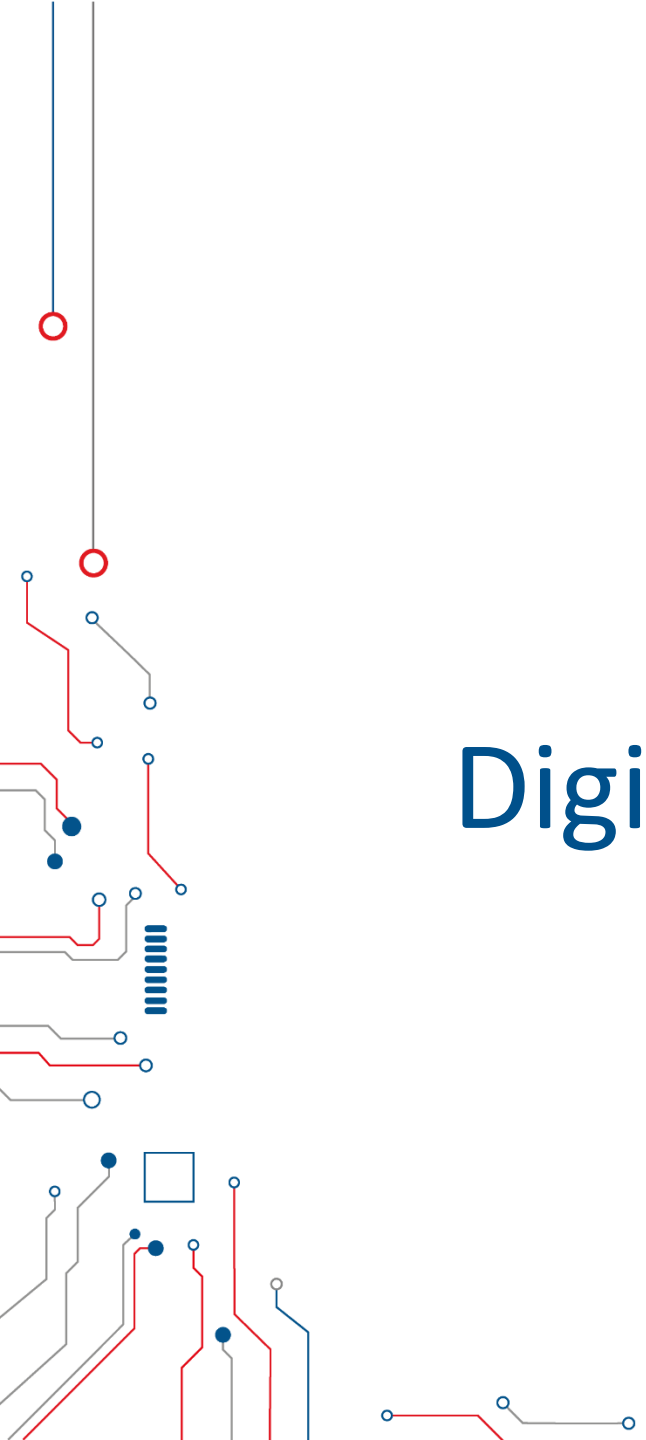
Digital Components & S/H











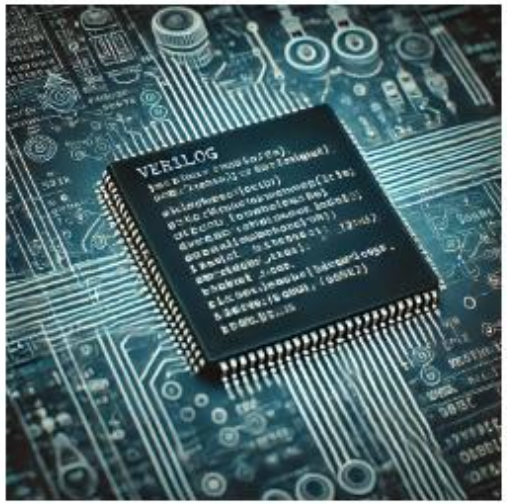
Leibniz Institute
for high
performance
microelectronics

Digital Integration In Xschem

Some open source EDA tools – digital design

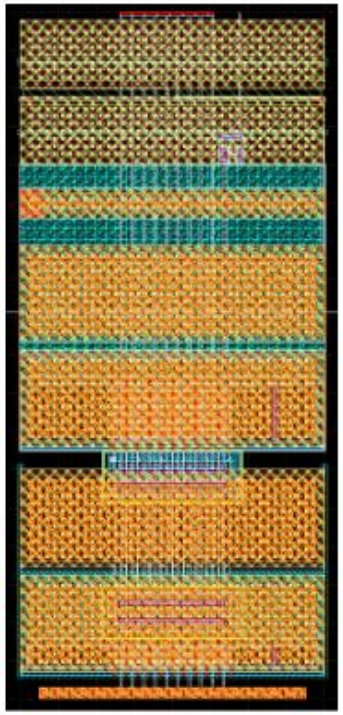
Simulation	Synthesis	PnR	GDS	Verification
Iverilog	Yosys	OpenROAD	Klayout	cocotb
Verilator		Coriolis	Magic	pyUVM
Ghdl				

```
always @(posedge clk) begin
  if (rst) begin
    done_q <= 1'b0;
    bit_ct_q <= 3'b0;
    dout_q <= 8'b0;
    miso_q <= 1'b1;
  end else begin
    done_q <= done_d;
    bit_ct_q <= bit_ct_d;
    dout_q <= dout_d;
    miso_q <= miso_d;
  end
```



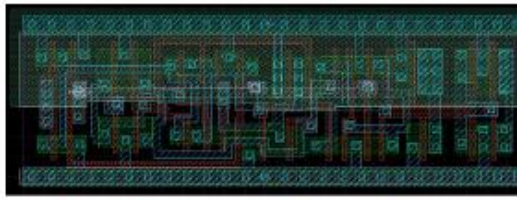
Digital primitives

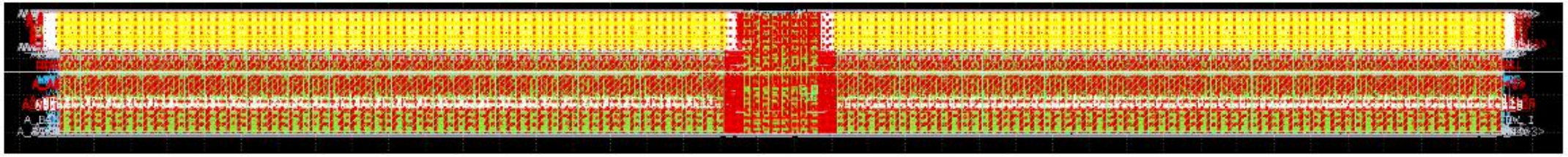
	StdCells	IOCells	SRAM
Verilog	x	x	x
LIB	x	x	x
LEF	x	x	x
CDL	x	x	x
SPICE	x	x	x
GDS	x	x	x



Digital components:

- stdcells
  - 78 cells,
  - combinatorial logic,
  - sequenntial elements,
  - scan flops,
  - gated cells.
- IOCells,
  - In, Out, InOut, Analog
  - different drive strengths
- SRAM
  - hard macros of a single port SRAM
  - different sizes
  - SRAM generator



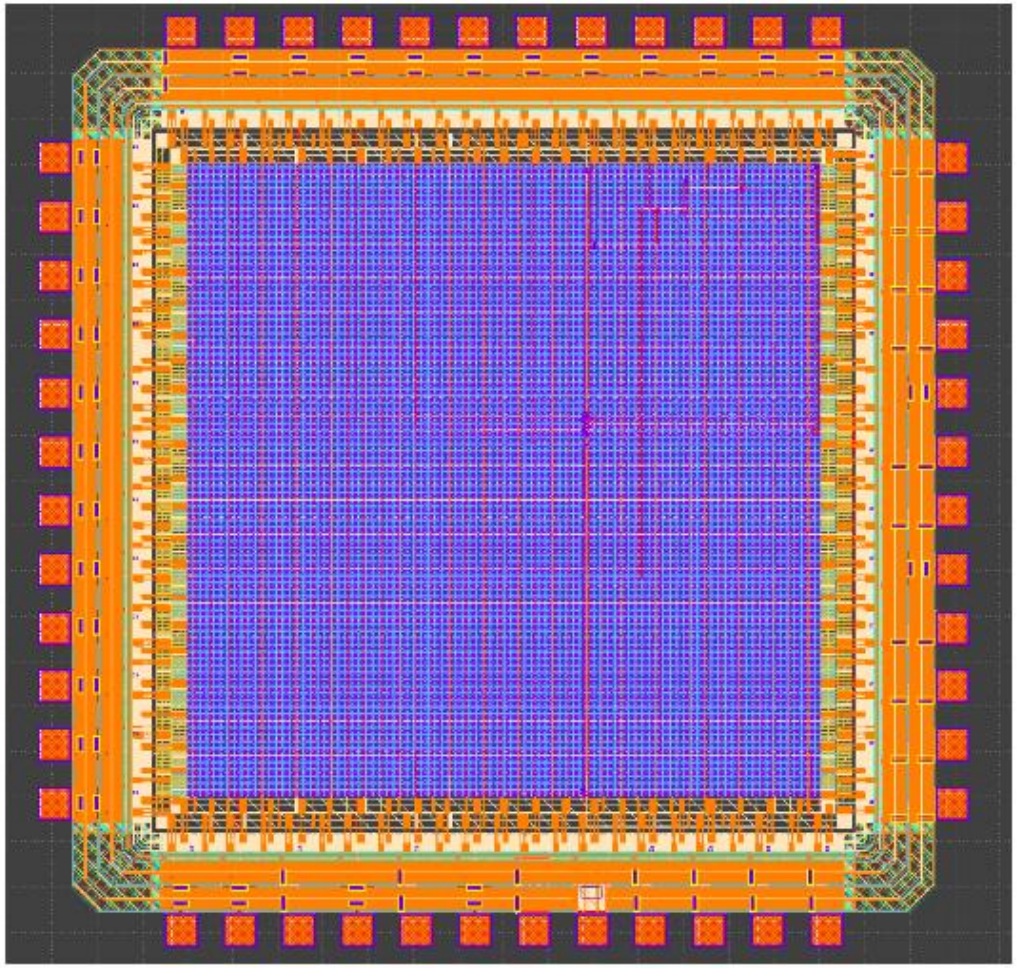


Mixed signal approach - goals

- Design a mixed signal chip
- Enable the simulation in two domains: analog and digital,
- Accurately model circuits behavior in both domains and at the interfaces.
- Automation of the design process

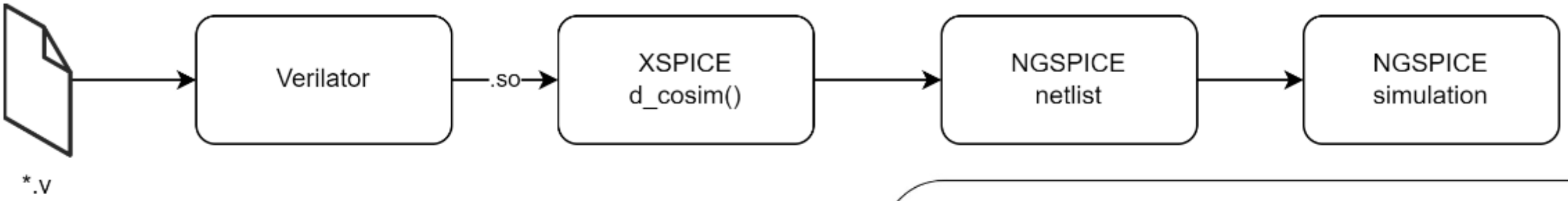
Key issues:

- Analog and digital behavioral models are handeled diferently
- Timing abstraction has to be handled as well



Mixed signal circuits development using OS tools

Event based acceleration in XSPICE, being part of the ngspice software

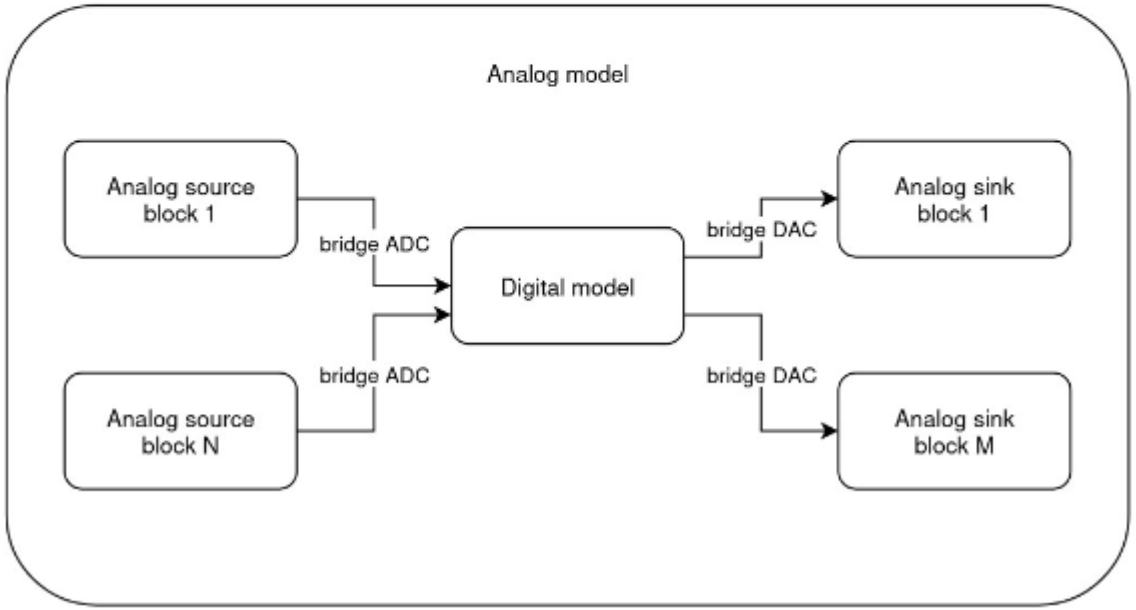


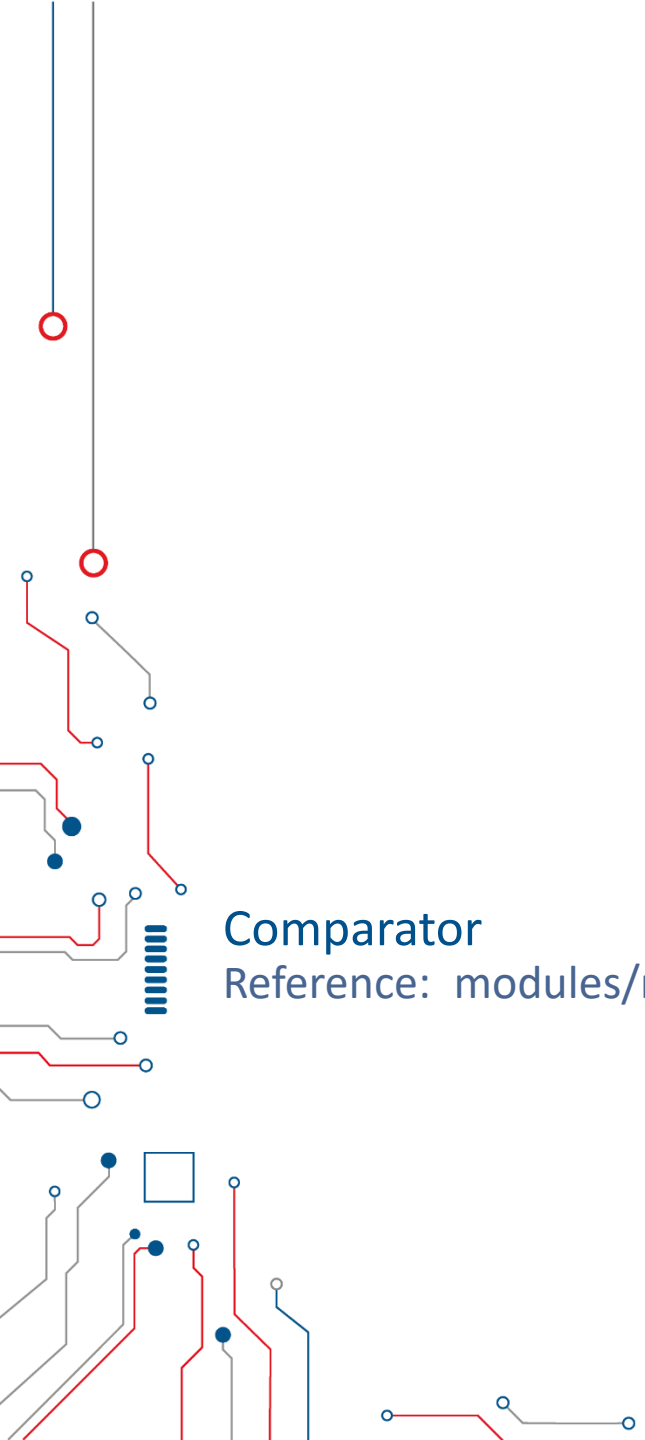
Many models of digital primitives, sources and sinks have been already modeled in XSPICE

Verilog and VHDL supportted

Verilator provides an elegant and scalable solution for encapsulation of a digital module

Gnucap as a future „post-spice" era mixed signal circuit simulator





Leibniz Institute
for high
performance
microelectronics

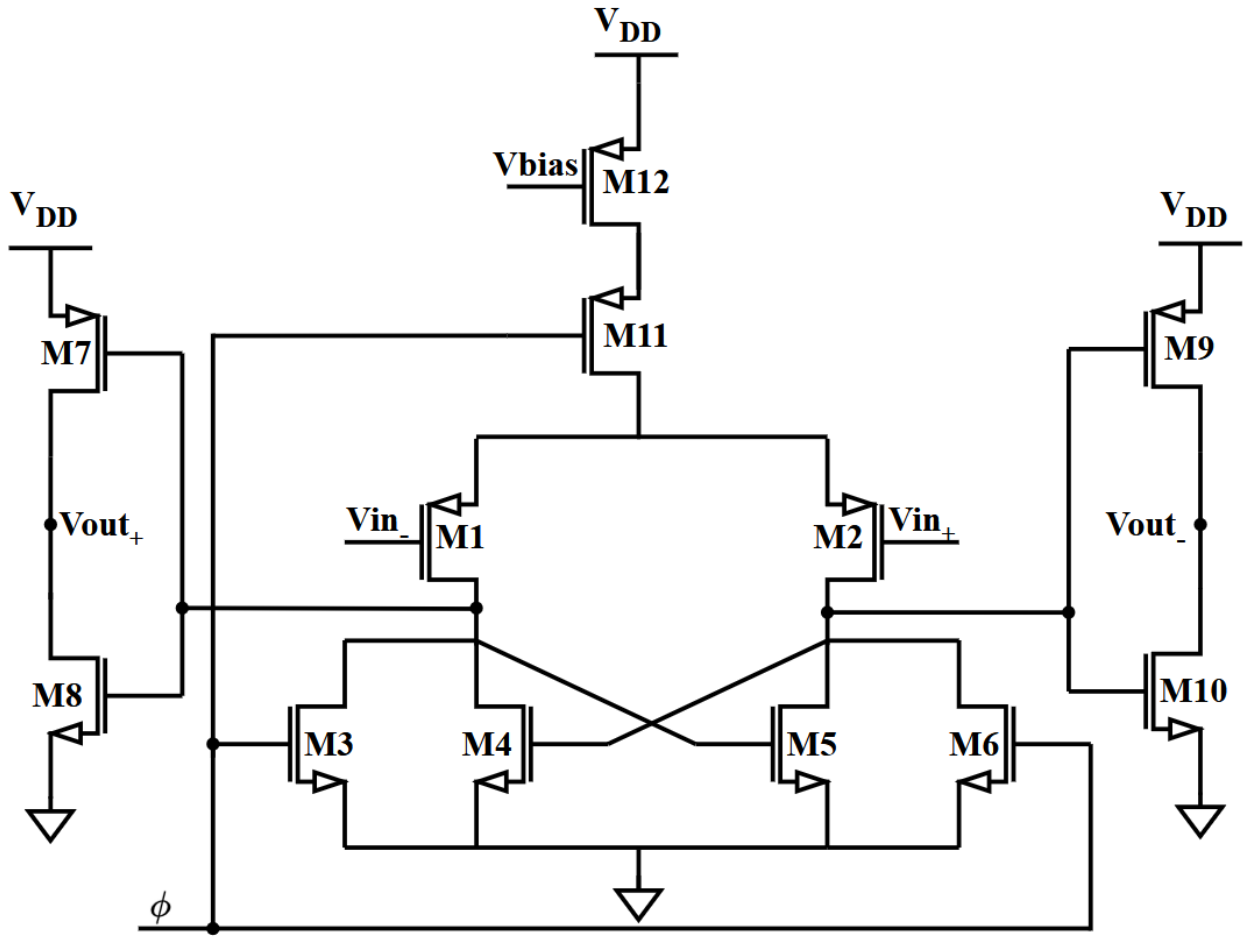
Part 1

Comparator

Reference: modules/module_3_8_bit_SAR_ADC/part_1_comparator

Dynamic Comparator

- Heart of the SAR ADC
- Makes decisions on falling edge!
- We want to evaluate the offset of this comparator!



Dynamic Comparator

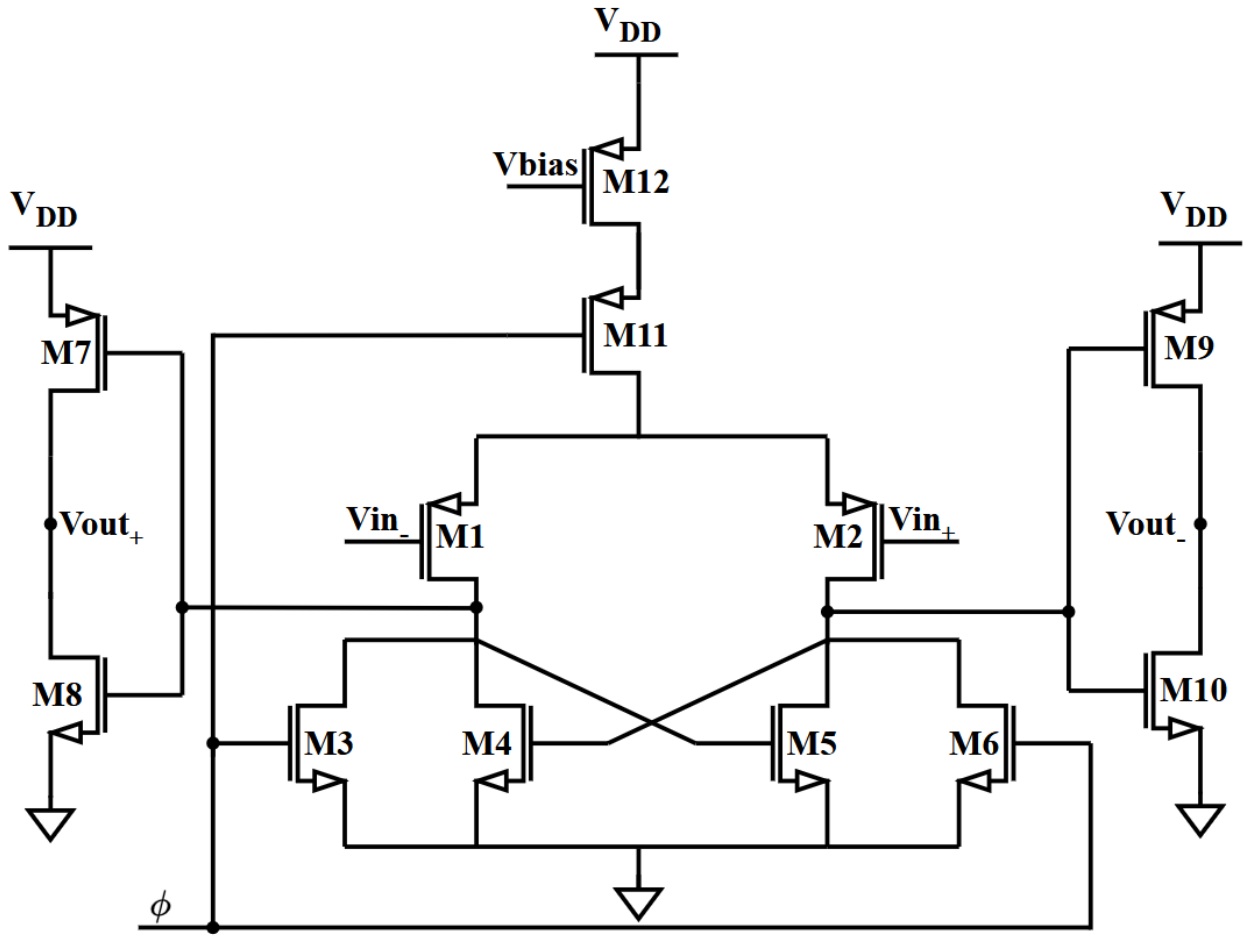
Specs

- **Vdd** is set to **1.2V**
- The **bias voltage** is set to **0.6V**
- A clock signal can be defined as follows:

PULSE(V1 V2 TD TR TF PW PER NP)

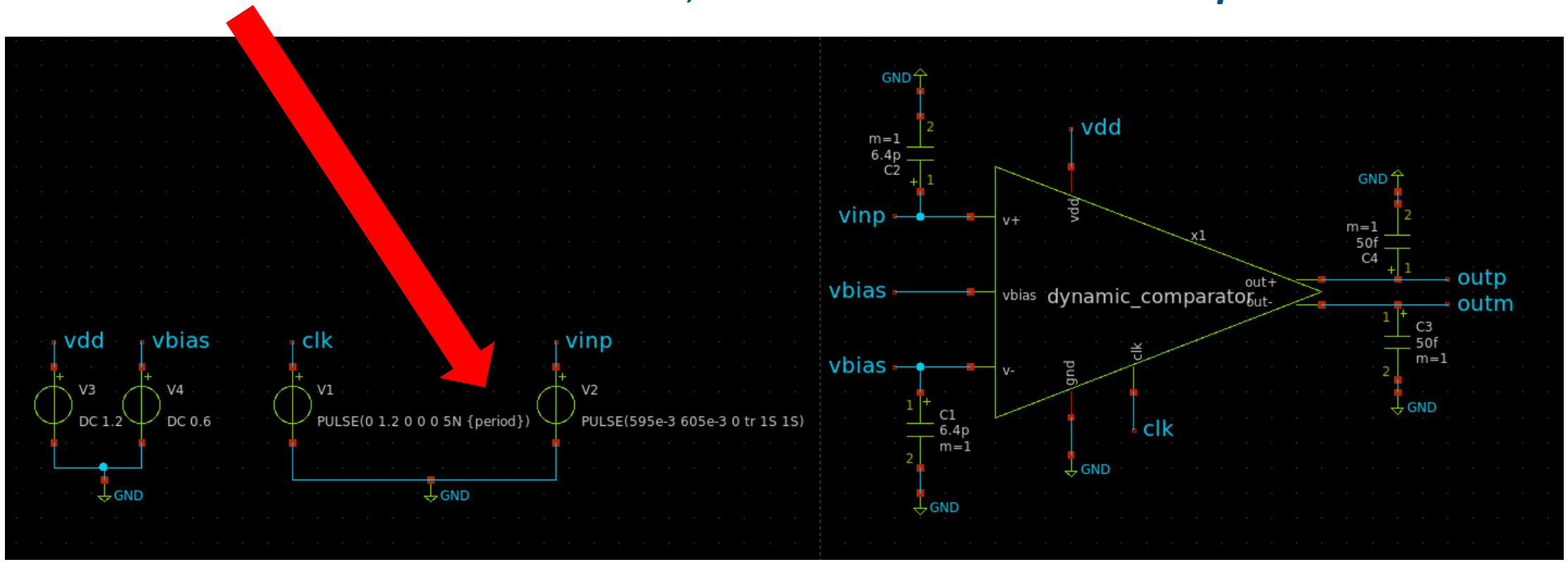
Name	Parameter	Default Value	Units
V1	Initial value	-	*V*, *A*
V2	Pulsed value	-	*V*, *A*
TD	Delay time	0.0	sec
TR	Rise time	TSTEP	sec
TF	Fall time	TSTEP	sec
PW	Pulse width	TSTOP	sec
PER	Period	TSTOP	sec
NP	Number of Pulses *)	unlimited	-

- **M1, M2**: W = 32 µm , L = 0.2 µm , ng = 4
- **M3, M4, M5, M6**: W = 4 µm , L = 0.2 µm
- **M8, M10**: W = 4 µm , L = 0.2 µm
- **M7, M9**: W = 8 µm , L = 0.2 µm
- **M11, M12**: W = 18 µm , L = 0.3 µm , ng = 4



Create The Testbench

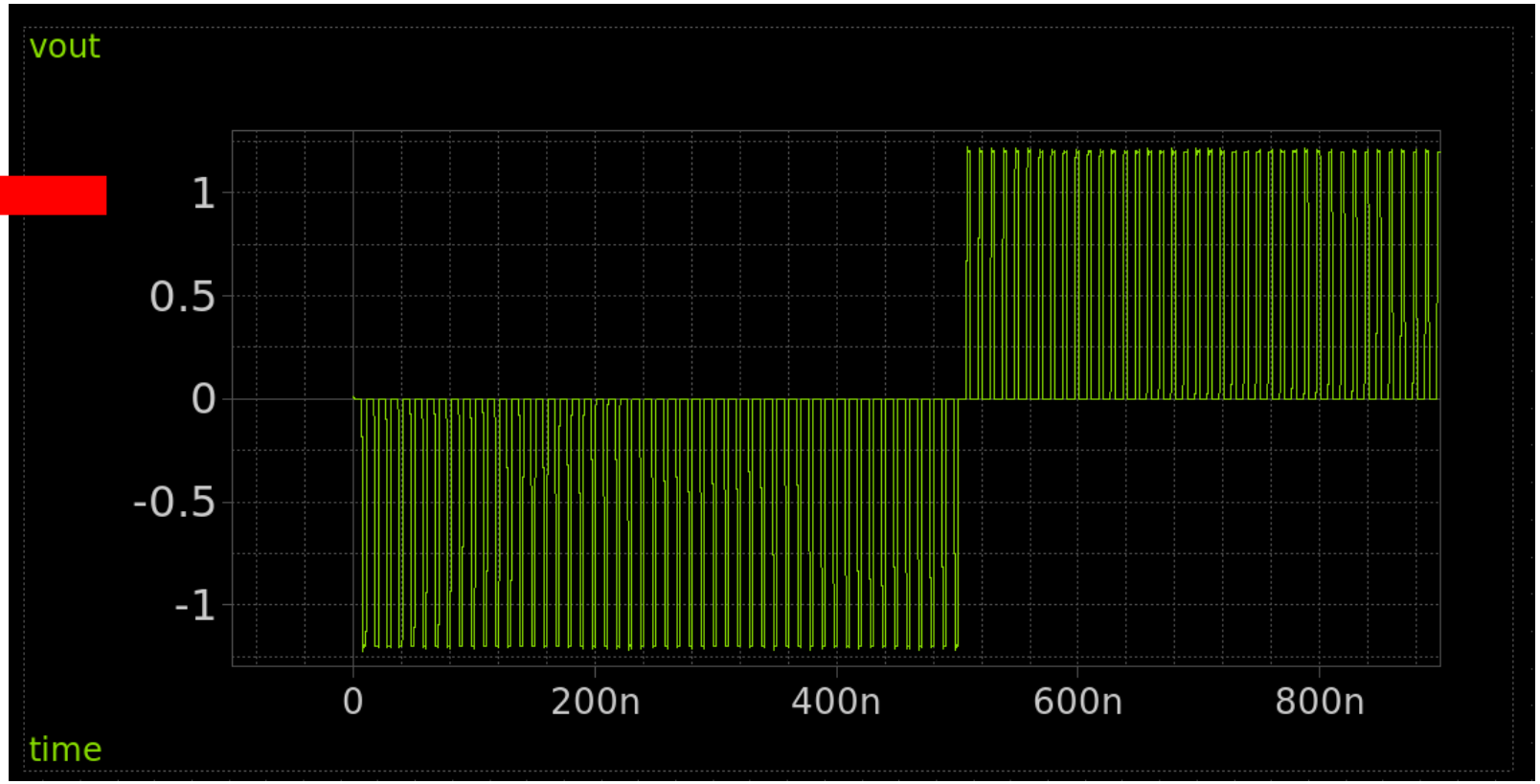
Period is defined in simulation code block, and same for risetime in ***vinp***



Plot the corresponding output from a transient analysis and verify the operation

Create The Testbench

```
name=NGSPICE only_toplevel=false
value="
.control
op
.param clock = 100e6    ; 100 MHz clock
.param period = 1 / clock
.param num_cycles = 100  ; cycles
.param tr = num_cycles * period
tran 500p 1u
.save all
let vindiff = (v(vinp))-(v(vbias))
let clk = v(clk)
let vout = (v(outp))-(v(outm))
write output_file.raw
.endc
"
```

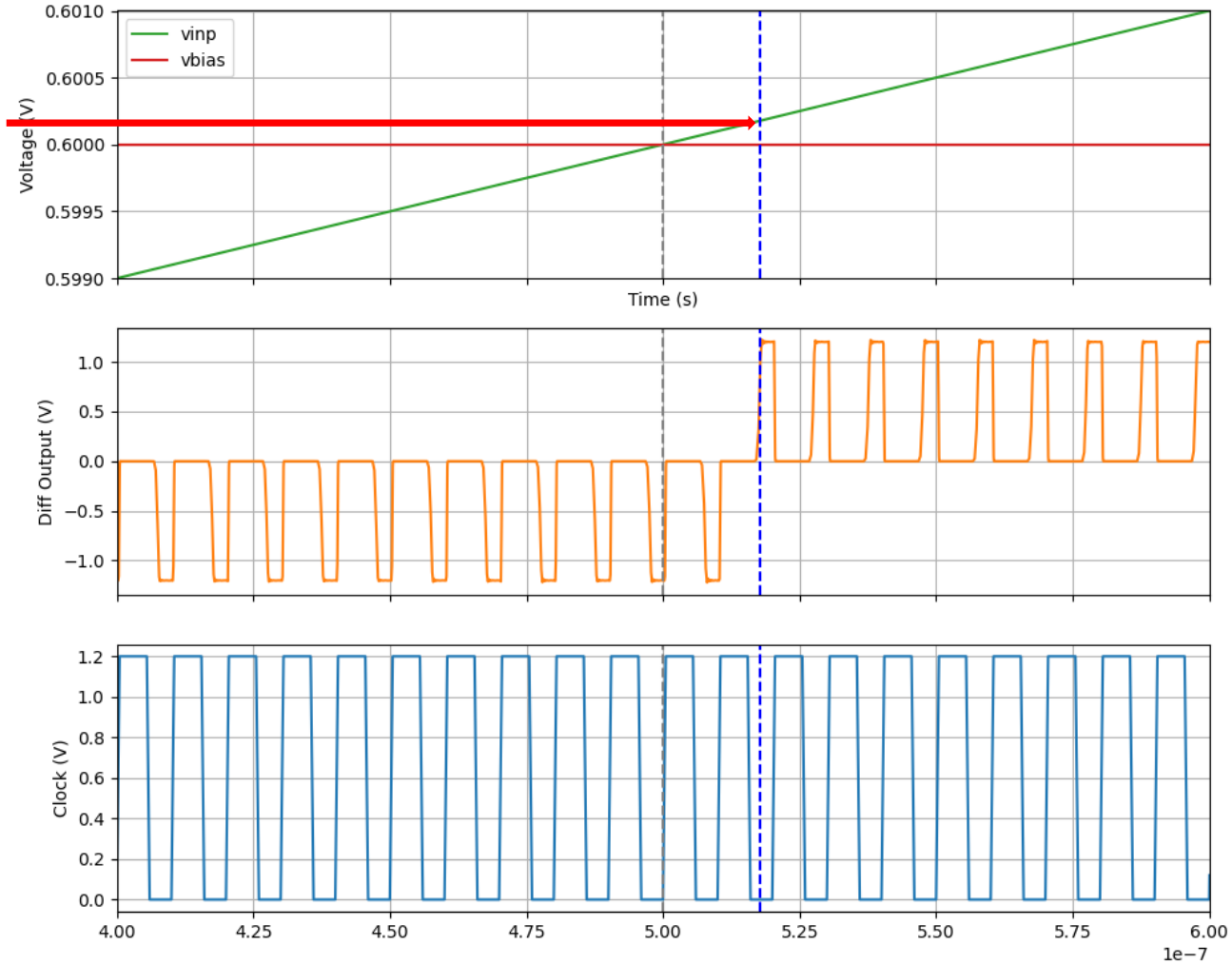


Mismatch Analysis of the Comparator

Why is it necessary

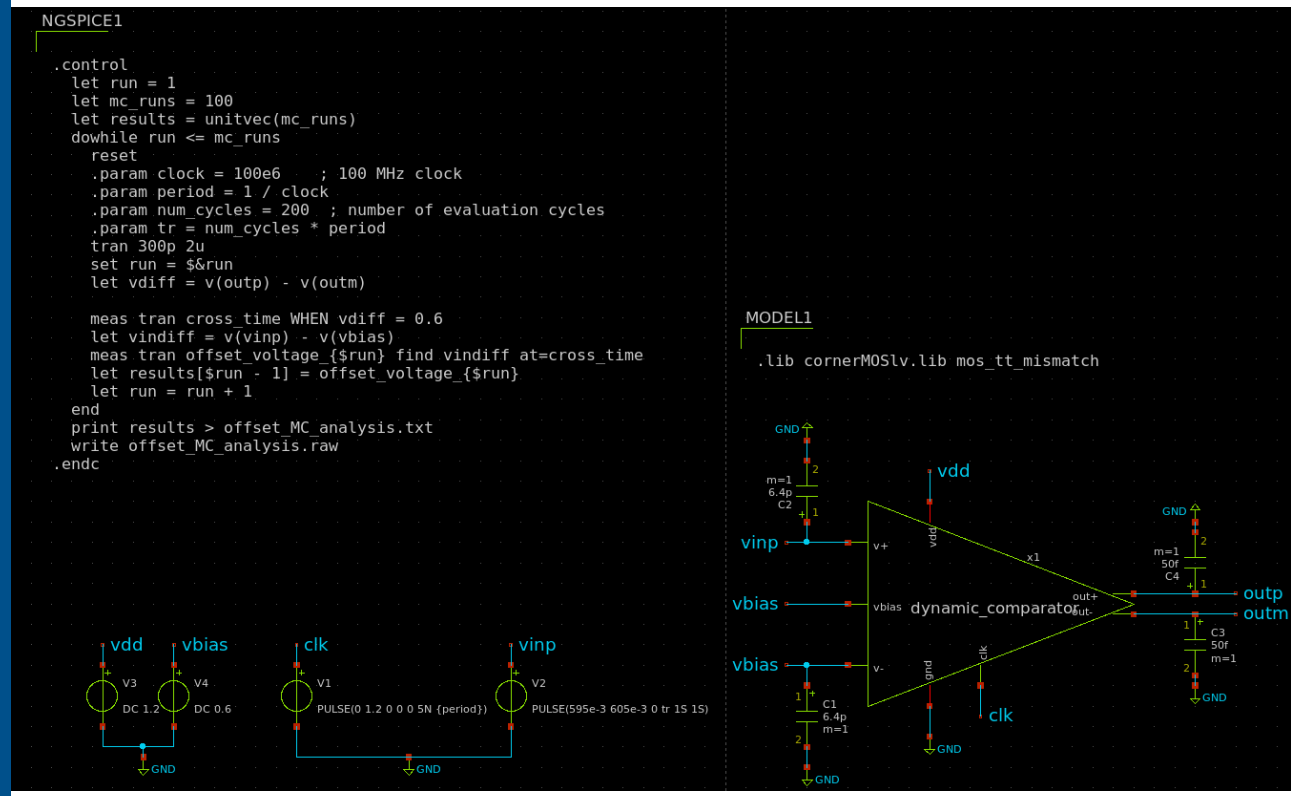
- Dynamic offset is highly sensitive to process mismatch

- We have to make sure its under 1/2LSB and thus cant be seen. Hidden by quantization error!



Mismatch Analysis of the Comparator

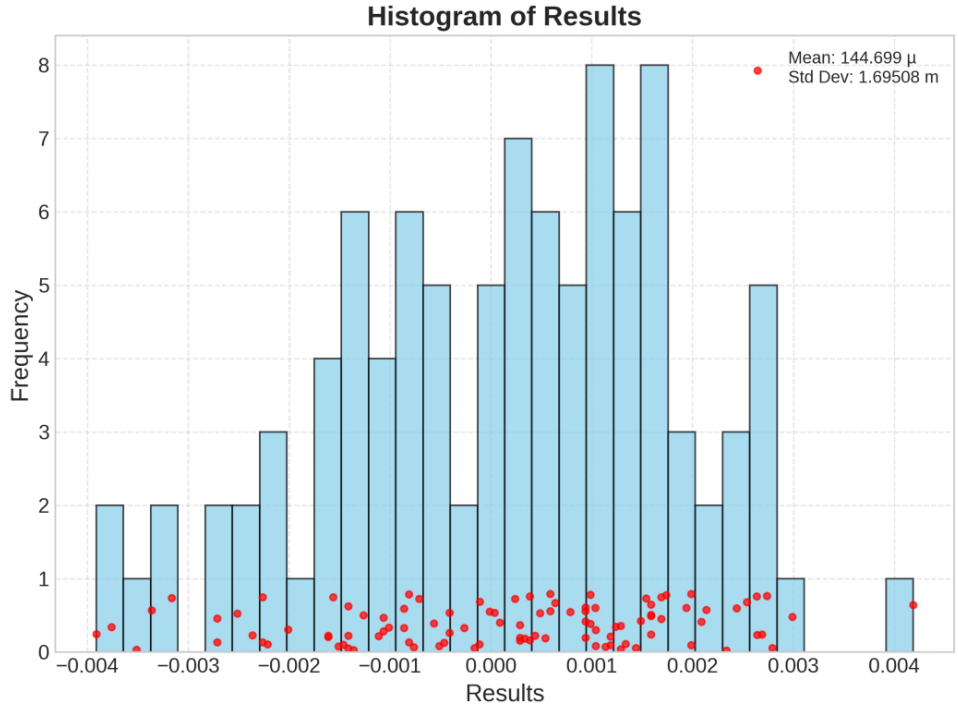
```
name=NGSPICE1 only_toplevel=false
value="
.control
 let run = 1
 let mc_runs = 100
 let results = unitvec(mc_runs)
 dowhile run <= mc_runs
  reset
  .param clock = 100e6    ; 100 MHz clock
  .param period = 1 / clock
  .param num_cycles = 200  ; number of evaluation cycles
  .param tr = num_cycles * period
  tran 300p 2u
  set run = $&run
  let vdiff = v(outp) - v(outm)
  meas tran cross_time WHEN vdiff = 0.6
  let vindiff = v(vinp) - v(vbias)
  meas tran offset_voltage_{$run} find vindiff at=cross_time
  let results[$run - 1] = offset_voltage_{$run}
  let run = run + 1
 end
 print results > offset_MC_analysis.txt
 write offset_MC_analysis.raw
.endc
"
```



Mismatch Analysis of the Comparator

- part_1_comparator/testbench/statistical_calculation
- python3 calculate_histogram.py ../simulations/offset_MC_analysis.txt
- **Histogram saved** to /statistical_calculation/histogram_plots/histogram.png
- xdg-open histogram_plots/histogram.png

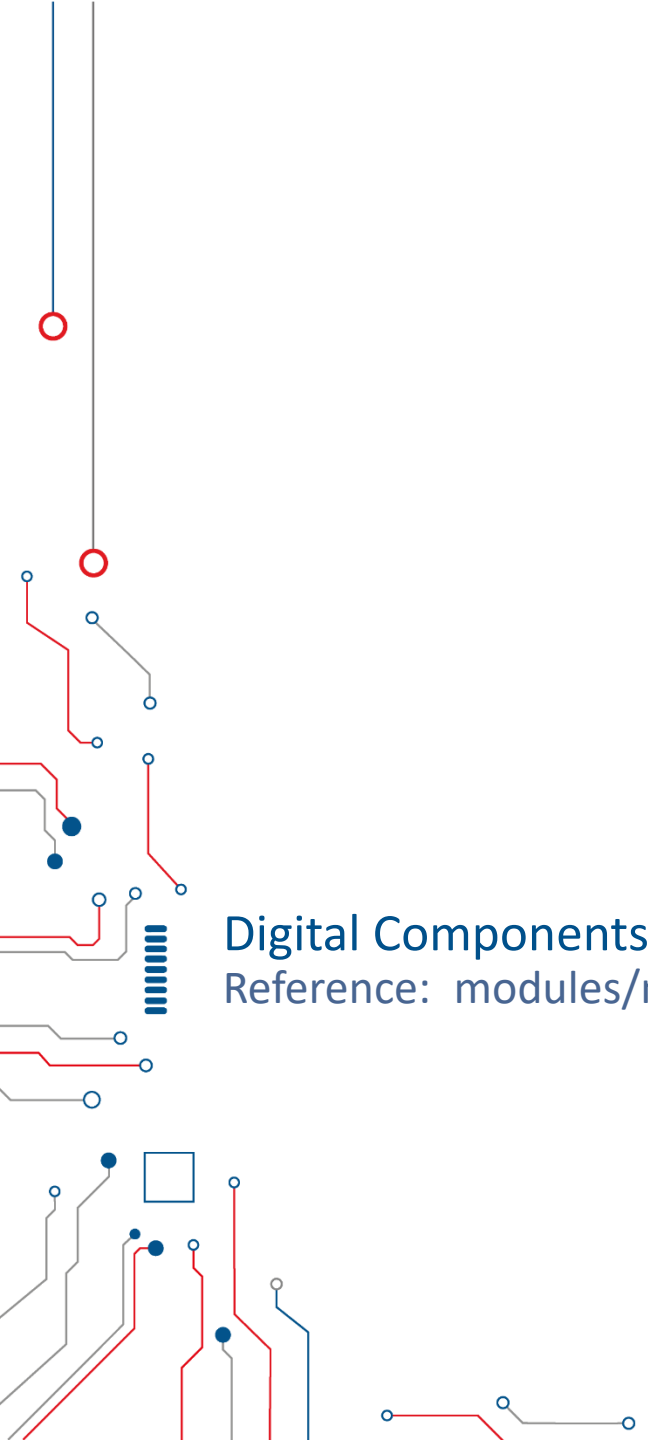
```
Total samples: 100
Mean: 144.699 µ
Standard Deviation: 1.69508 m
Do you want to create a histogram plot? (y/n):
```



Leibniz Institute
for high
performance
microelectronics

Catching Up / Lunch Time !

Next session:
SAR Algorithm



Leibniz Institute
for high
performance
microelectronics

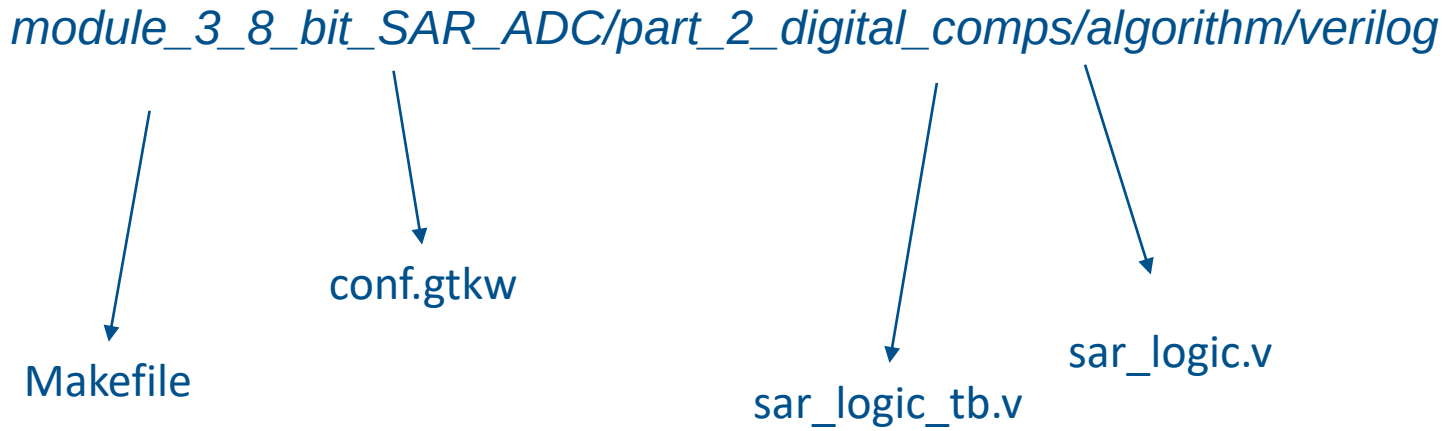
Part 2

Digital Components

Reference: modules/module_3_8_bit_SAR_ADC/part_2_digital_comps

SAR Algorithm

```
always @(posedge clk) begin
   if (rst) begin
      B <= 7'b0000000;
      BN <= 7'b0000000;
      D <= 8'b00000000;
      counter <= 4'b0000;
   end else if (En && (Op ^ Om)) begin
      if (counter < 7) begin
         D <= D | ({7'b0, Op} << counter);

         B[counter % 7] <= (Op) ? 1'b1 : 1'b0;
         BN[counter % 7] <= (Om) ? 1'b1 : 1'b0;

         counter <= counter + 1'b1;
      end
   end
end
```



SAR Algorithm

The algorithm folder in the repository can be copied directly to your own directory for simplicity

Makefile

```
# Define variables
IVERILOG = iverilog
VVP = vvp
GTKWAVE = gtkwave
NGSPICE = ngspice
OUTPUT = sar_logic.out
MIXED = sar_logic.so
MIXED_OBJ = sar_logic_obj_dir
VCD_FILE = sar_logic_tb.vcd
SRC_FILES = sar_logic.v sar_logic_tb.v
NGSPICE_FILE = sar_logic.v
DEST_DIR = ../xschem/simulations
```

```
# Default target
all: run_wave

# Compile using iverilog
$(OUTPUT): $(SRC_FILES)
	$(IVERILOG) -o $(OUTPUT) $(SRC_FILES)

# Run the simulation with vvp
run: $(OUTPUT)
	$(VVP) $(OUTPUT)

copy:
	cp $(MIXED) $(DEST_DIR)

# Open the wave file in GTKWave
wave: $(VCD_FILE)
	$(GTKWAVE) $(VCD_FILE) conf.gtkw

# Run ngspice vlnggen command
ngspice:
	$(NGSPICE) vlnggen $(NGSPICE_FILE)

# Execute both the simulation and wave viewer
run_wave: run wave

# Run ngspice and then the rest of the steps
full: ngspice run_wave copy

# Clean up generated files
clean:
	rm -f $(OUTPUT) $(VCD_FILE) $(MIXED)
	rm -rf $(MIXED_OBJ)
```

make run

make copy

make wave

make ngspice

make run_wave

make full

make clean

Conf.gtkw

```
[*]
[*] GTKWave Analyzer v3.3.116 (w)1999-2023 BSI
[*] Mon Sep 30 13:11:15 2024
[*]
[dumpfile] "sar_logic_tb.vcd"
[dumpfile_mtime] "Mon Sep 30 13:10:47 2024"
[dumpfile_size] 777
[savefile] "conf.gtkw"
[timestart] 0
[size] 1906 985
[pos] -1 -1
*-14.000000 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
[sst_width] 269
[signals_width] 102
[sst_expanded] 1
[sst_vpaned_height] 291
@28
sar_logic_tb.Op
sar_logic_tb.Om
sar_logic_tb.B[6:0]
sar_logic_tb.BN[6:0]
sar_logic_tb.D[7:0]
sar_logic_tb.clk
sar_logic_tb.rst
sar_logic_tb.En
@29
[pattern_trace] 1
[pattern_trace] 0
```

Sets initial waves to be plotted when running the make file

SAR Logic

```
module sar_logic (
    input wire clk,
    input wire Op,
    input wire En,
    input wire Om,
    input wire rst,
    output reg [6:0] B,   // 7-bit
    output reg [6:0] BN,  // 7-bit
    output reg [7:0] D    // 8-bit
);

reg [3:0] counter = 4'b0000; // 4-bit counter

always @(posedge clk) begin
    if (rst) begin
        B <= 7'b0000000;
        BN <= 7'b0000000;
        D <= 8'b00000000;
        counter <= 4'b0000;
    end else if (En && (Op ^ Om)) begin
        if (counter < 7) begin
            D <= D | ({7'b0, Op} << counter);

            B[counter % 7] <= (Op) ? 1'b1 : 1'b0;
            BN[counter % 7] <= (Om) ? 1'b1 : 1'b0;

            counter <= counter + 1'b1;
        end
    end
end

endmodule
```

SAR Logic Testbench

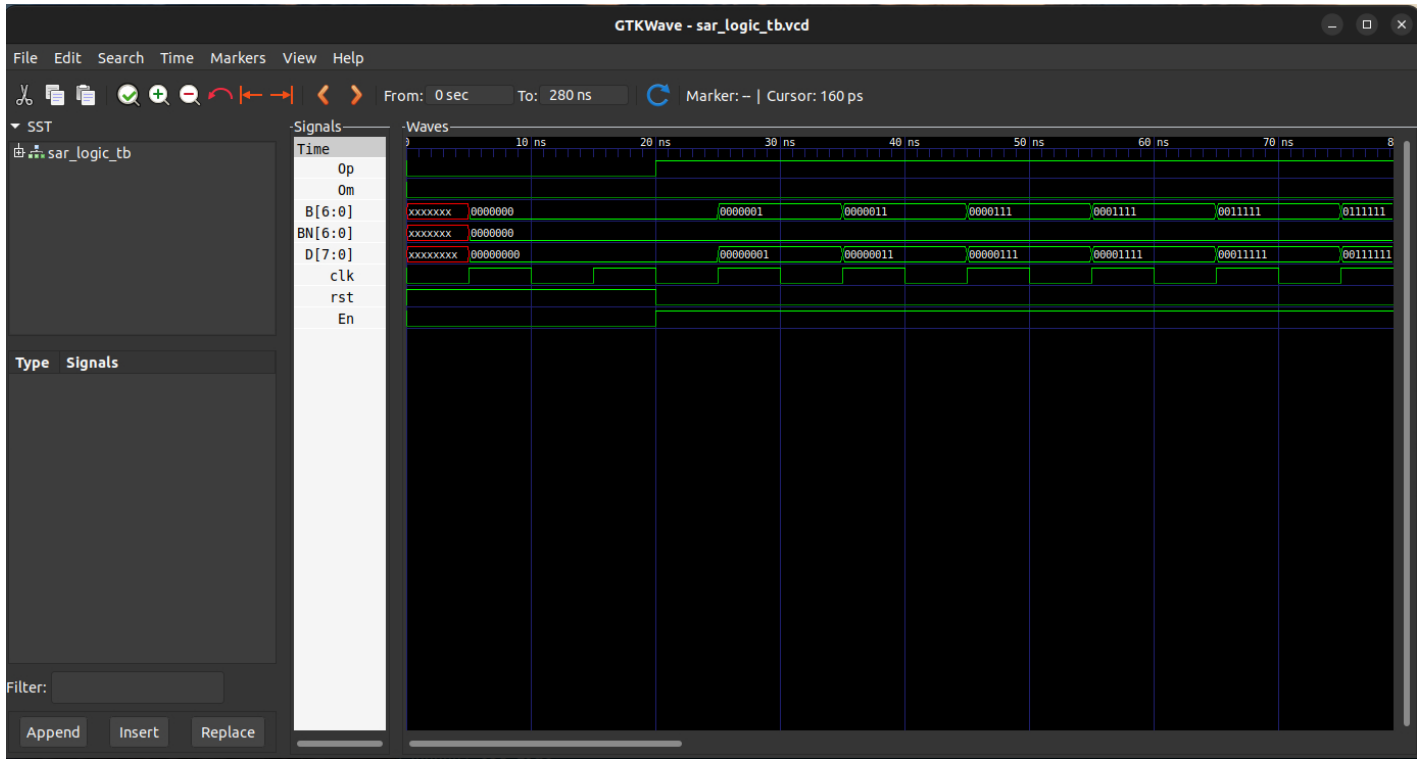
```
`timescale 1ns/1ps

module sar_logic_tb();

    // Inputs
    reg clk;
    reg rst;
    reg Op;
    reg Om;
    reg En;

    // Outputs
    wire [6:0] B;
    wire [6:0] BN;
    wire [7:0] D;

    // Instantiate the SAR Logic module
    sar_logic uut (
        .clk(clk),
        .rst(rst),
        .Op(Op),
        .Om(Om),
        .En(En),
        .B(B),
        .BN(BN),
        .D(D)
    );
```

```
    // Clock generation
    always #5 clk = ~clk;

    // Test sequence
    initial begin
        $dumpfile("sar_logic_tb.vcd");  // Name of the VCD file
        $dumpvars(0, sar_logic_tb);     // Dump all variables in the testbench

        // Initialize inputs
        clk = 1'b0;
        rst = 1'b1;
        Op = 1'b0;
        Om = 1'b0;
        En = 1'b0;

        // Apply reset
        #20 rst = 1'b0;
        En = 1'b1;
        Op = 1'b1;
        Om = 1'b0;

        // Apply reset again
        #80 rst = 1'b1;
        #10 rst = 1'b0;
        Op = 1'b0;
        Om = 1'b1;

        #70 rst = 1'b1;

        // End of simulation
        #100 $finish;
    end

endmodule
```

Running Make full

cp sar_logic.so ../xschem/simulations

Why??



conf.gtkw
Makefile
sar_logic_obj_dir
sar_logic.out
sar_logic.so
sar_logic_tb.v
sar_logic_tb.vcd
sar_logic.v
sar_logic.vcd

Creating Symbol For The Digital Block

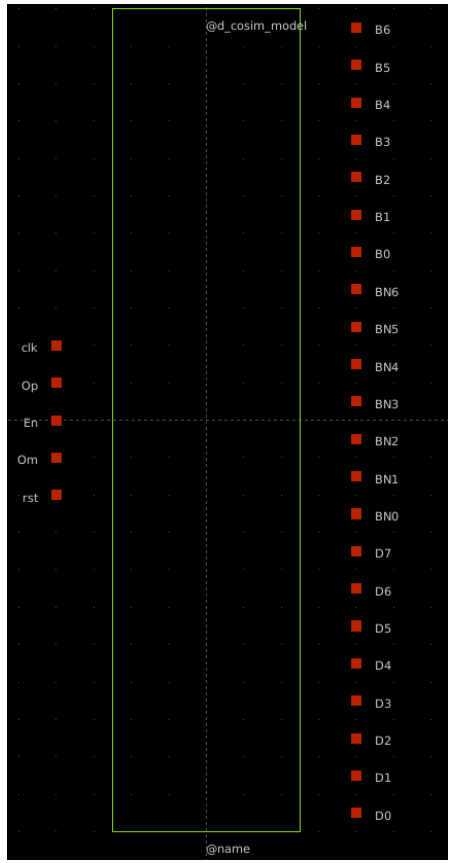
➡ module_3_8_bit_SAR_ADC/part_2_digital_comps/algorithm/python

➜ python git:(main) ✗ python3 generate_sym.py

Usage: python generate_sym.py <verilog_file> <symbol_file>
Example: python generate_sym.py control.v test.sym

Usage from Xschem directory:
python3 ../python/generate_sym.py ../verilog/sar_logic.v sar_logic.sym

Parsed module: sar_logic
Inputs: ['clk', 'Op', 'En', 'Om', 'rst']
Outputs: ['D7', 'D6', 'D5', 'D4', 'D3', 'D2', 'D1', 'D0', 'BN6', 'BN5',
'BN4', 'BN3', 'BN2', 'BN1', 'BN0', 'B6', 'B5', 'B4', 'B3', 'B2', 'B1', 'B0']
Xschem symbol written to sar_logic.sym



Creating Symbol For The Digital Block

```
v {xschem version=3.4.5 file_version=1.2
*
* This file is part of XSCHEM,
* a schematic capture and Spice/Vhdl/Verilog netlisting tool for circuit
* simulation.
* Copyright (C) 1998-2023 Stefan Frederik Schippers
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
}
G {}
K {type=delay
verilog_ignore=true
vhdl_ignore=true
format="@name [ @@clk @@Op @@En @@Om @@rst ] [ @@B6 @@B5 @@B4 @@B3 @@B2 @@B1 @@B0 @@BN6 @@BN5 @@BN4 @@BN3 @@BN2 @@BN1 @@BN0 @@D7 @@D6 @@D5 @@D4 @@D3 @@D2 @@D1 @@D0 ] null @dut
.model @dut @d_cosim_model simulation=@model"
template="name=adut
dut=dut
d_cosim_model= d_cosim
model=./sar_logic.so" }
V {}
S {}
E {}
```

Creating Symbol For The Digital Block

Stefan Schippers, the creator of Xschem, has made an in-depth video demonstrating how to create symbols for co-simulation. He also explains how to generate multidimensional outputs, which helps avoid large and cluttered symbol structures like the one in our example. Instead of manually expanding each signal, they can be represented more compactly—for example, as:
D[6..0]

https://www.youtube.com/watch?v=PPd7jkcHOgA

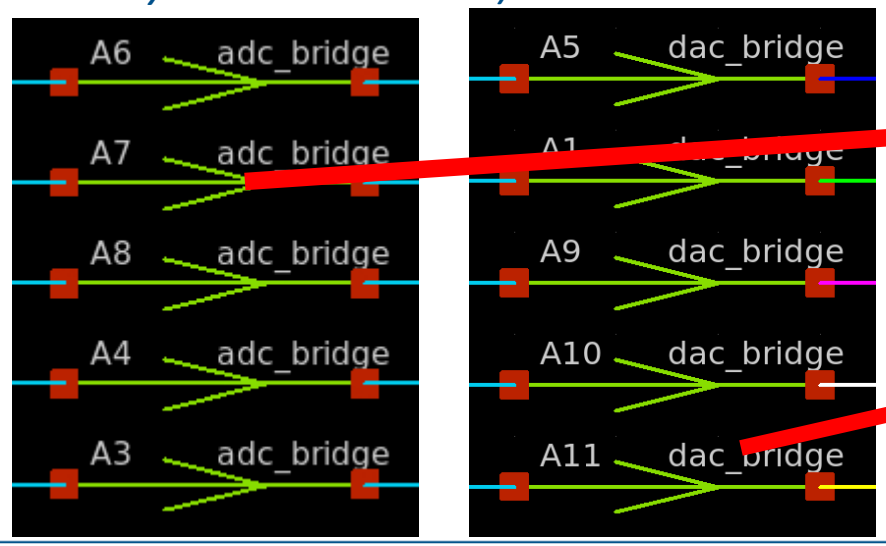
For this tutorial a script to generate such a symbol for our use-case specifically was made to save time!!

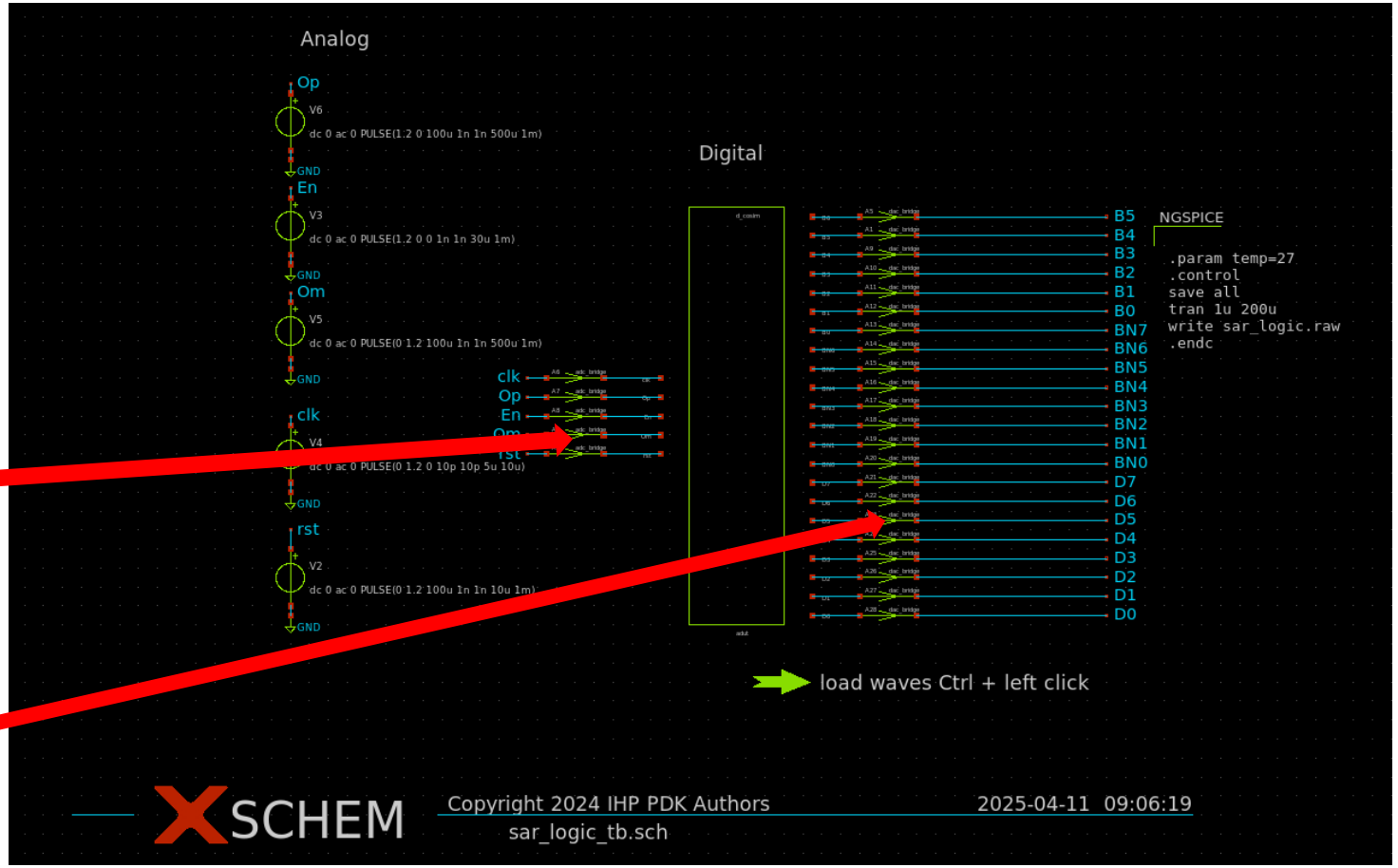
Digital Block In Xschem

Navigate to the following path:
module_3_8_bit_SAR_ADC/part_2_digital_comps/algorithm/xschem

→ xschem git:(main) ✗ xschem sar_logic_tb.sch

We need adc_bridge and dac_bridge to interface with the digital block *(can be found in same directory as the testbench)*





Digital Block In Xschem

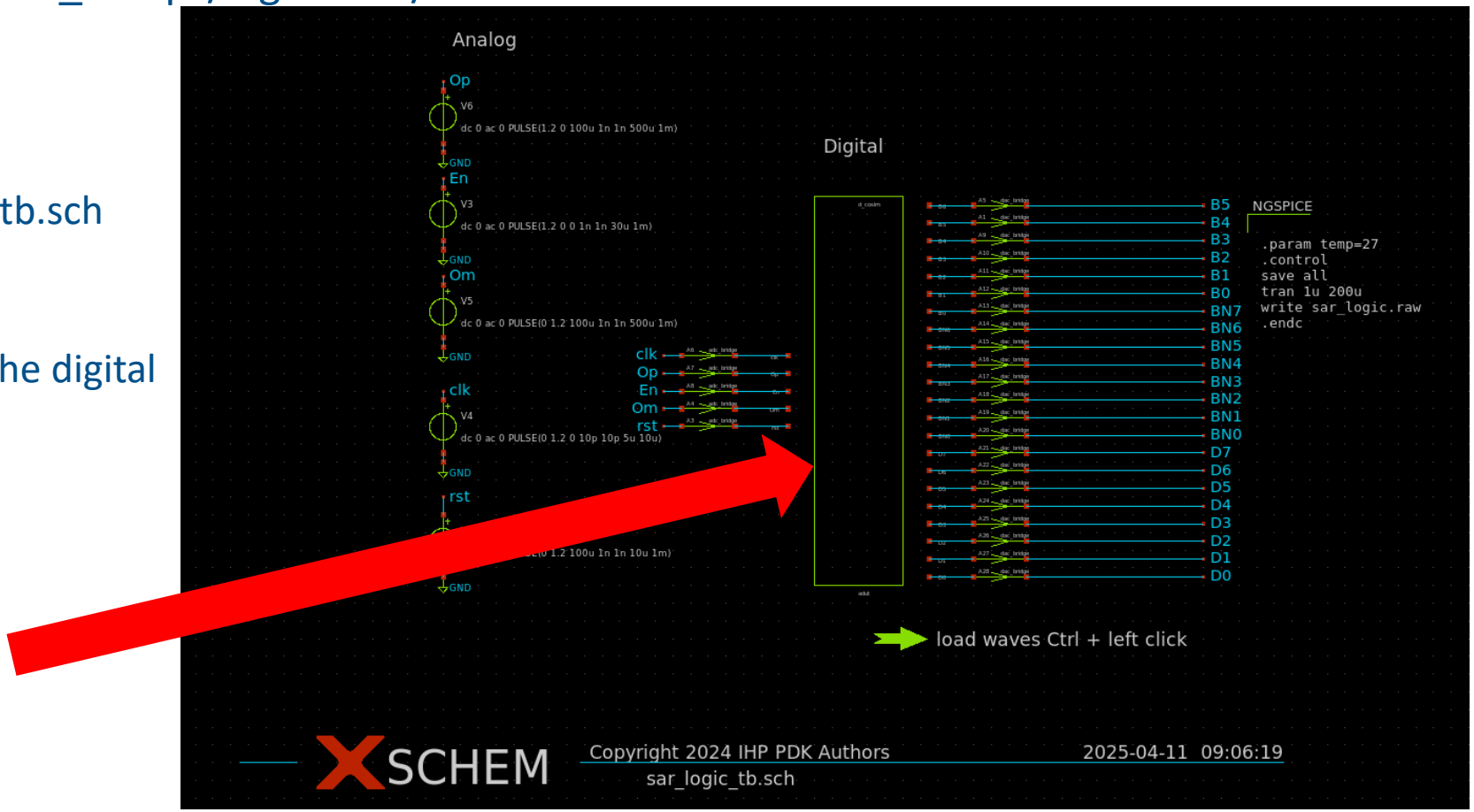
Navigate to the following path:

module_3_8_bit_SAR_ADC/part_2_digital_comps/algorithm/xschem

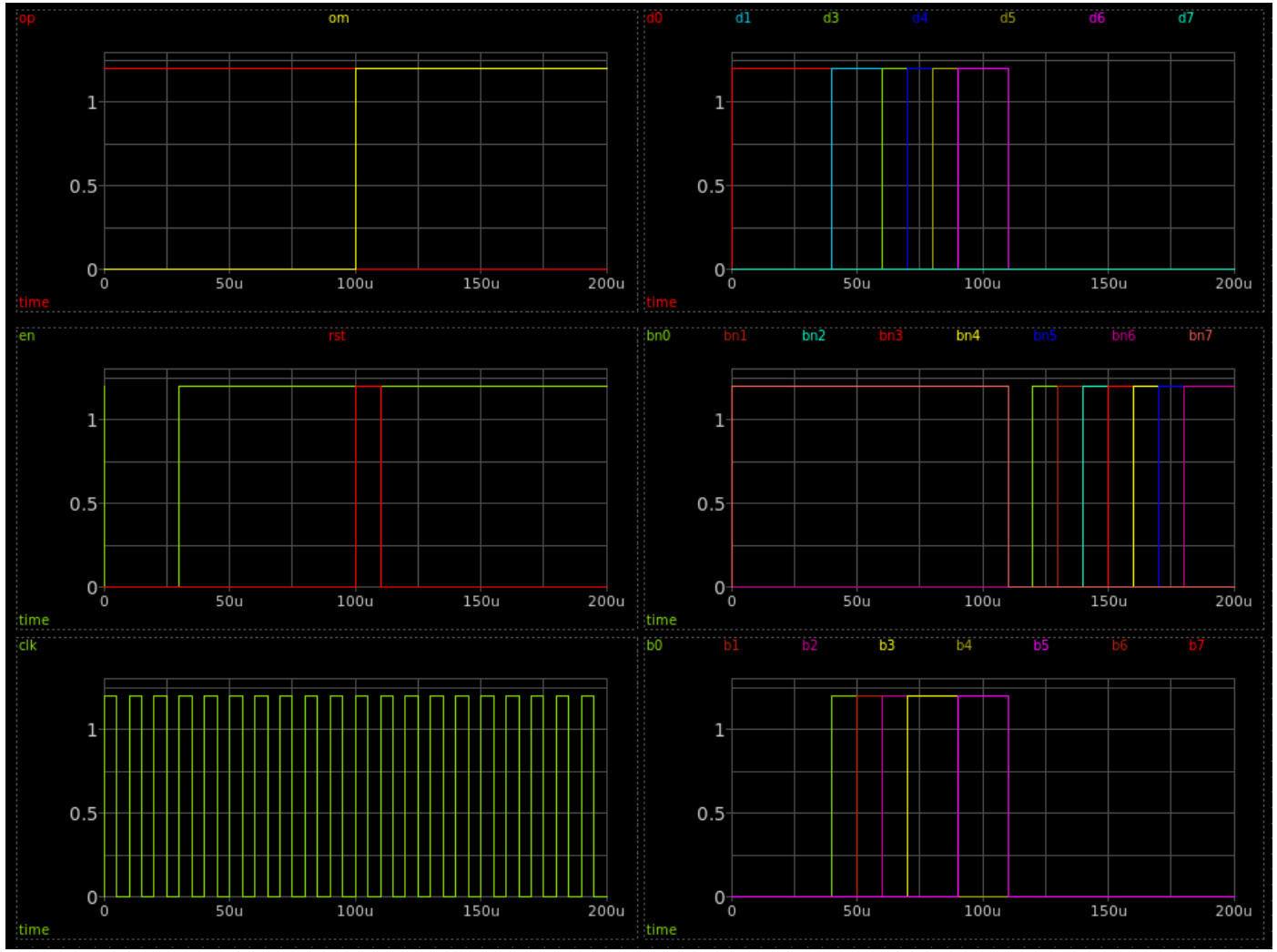
→ xschem git:(main) ✗ xschem sar_logic_tb.sch

- Simulate and confirm the operation of the digital block

```
name=adut
dut=dut
d_cosim_model= d_cosim
model=./sar_logic.so
```



Digital Block In Xschem



Remaining Components

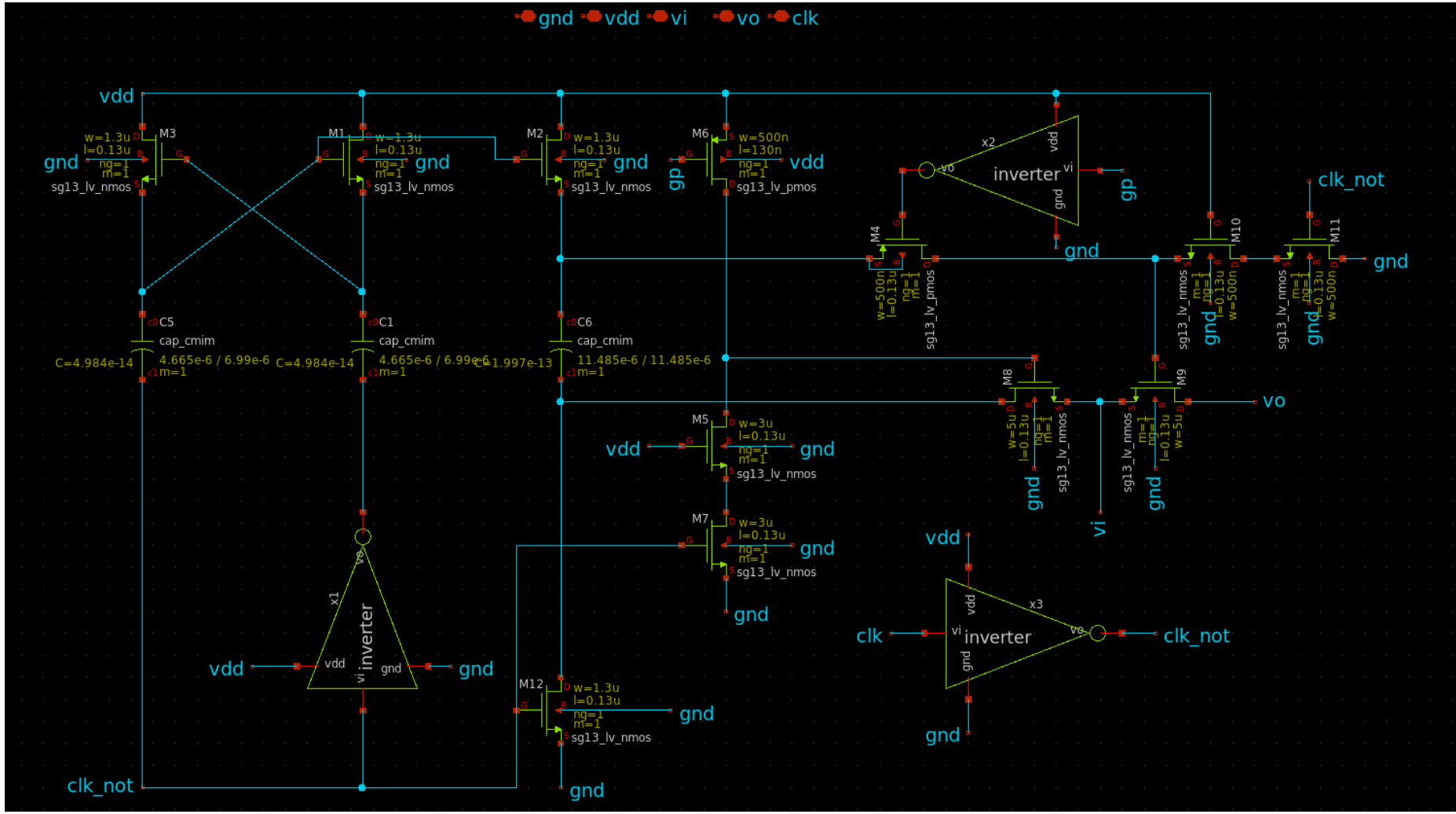
To realize the final schematic we have to create the following components:

- Boostrap
- Nand gate
- Transmission gate

On the next couple of slides the circuits can be seen, its up to you to test them and confirm operation.

Bootstrap Switch

- The following schematic is quite large
- you may copy the schematic from the repository
- Good practice to create the symbol yourself, along with the testbench!
- For the testbench explanation refer to the markdown file under this part



Bootstrap Switch

NMOS Transistors (sg13_lv_nmos):

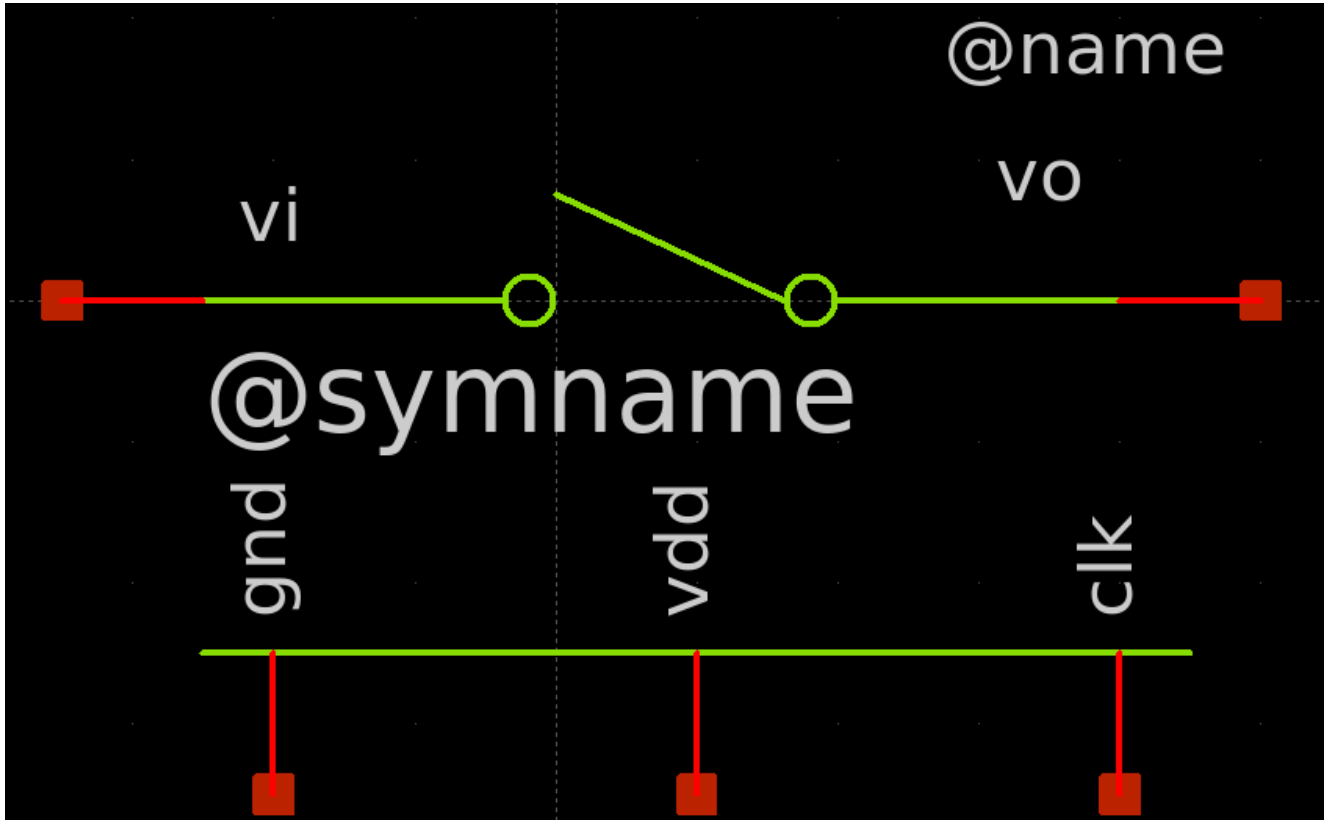
- M1: W = 1.3 µm, L = 0.13 µm, ng = 1, m = 1
- M2: W = 1.3 µm, L = 0.13 µm, ng = 1, m = 1
- M3: W = 1.3 µm, L = 0.13 µm, ng = 1, m = 1
- M5: W = 3.0 µm, L = 0.13 µm, ng = 1, m = 1
- M7: W = 3.0 µm, L = 0.13 µm, ng = 1, m = 1
- M8: W = 5.0 µm, L = 0.13 µm, ng = 1, m = 1
- M9: W = 5.0 µm, L = 0.13 µm, ng = 1, m = 1
- M10: W = 0.5 µm, L = 0.13 µm, ng = 1, m = 1
- M11: W = 0.5 µm, L = 0.13 µm, ng = 1, m = 1
- M12: W = 1.3 µm, L = 0.13 µm, ng = 1, m = 1

PMOS Transistors (sg13_lv_pmos):

- M4: W = 0.5 µm, L = 0.13 µm, ng = 1, m = 1
- M6: W = 0.5 µm, L = 0.13 µm, ng = 1, m = 1

Capacitors (cap_cmim):

- C1: W = 4.665 µm L = 6.99 µm m = 1
- C5: W = 4.665 µm L = 6.99 µm m = 1
- C6: W = 11.485 µm L = 11.485 µm m = 1



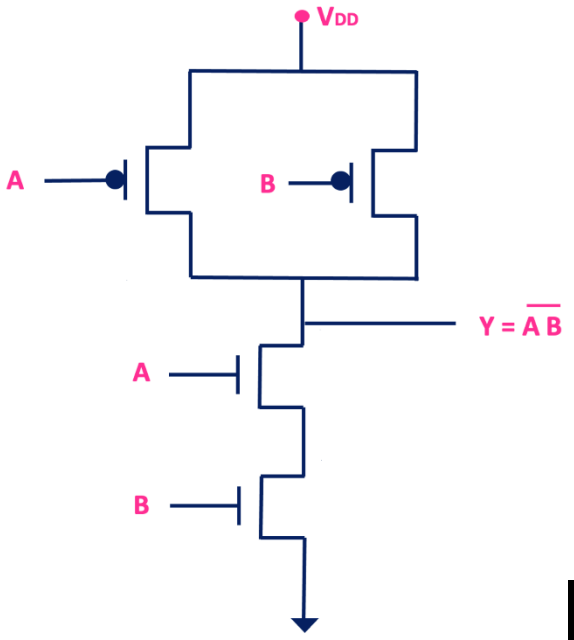
For the inverter the reader is needed to design this use the following parameters **NMOS Transistors (sg13_lv_nmos):**

M1: W = 1.3 µm, L = 0.4 µm, ng = 1, m = 1

PMOS Transistors (sg13_lv_pmos):

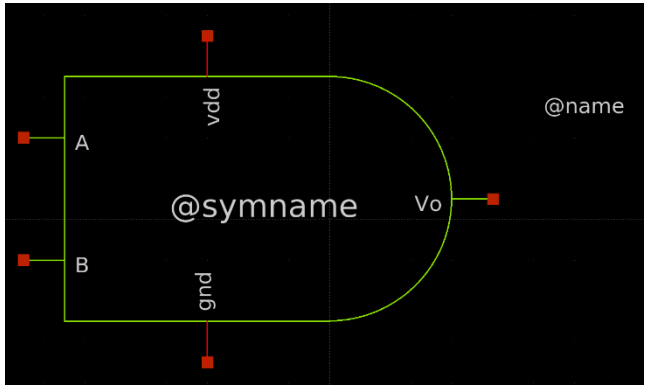
- M2: W = 1.3 µm, L = 0.8 µm, ng = 1, m = 1

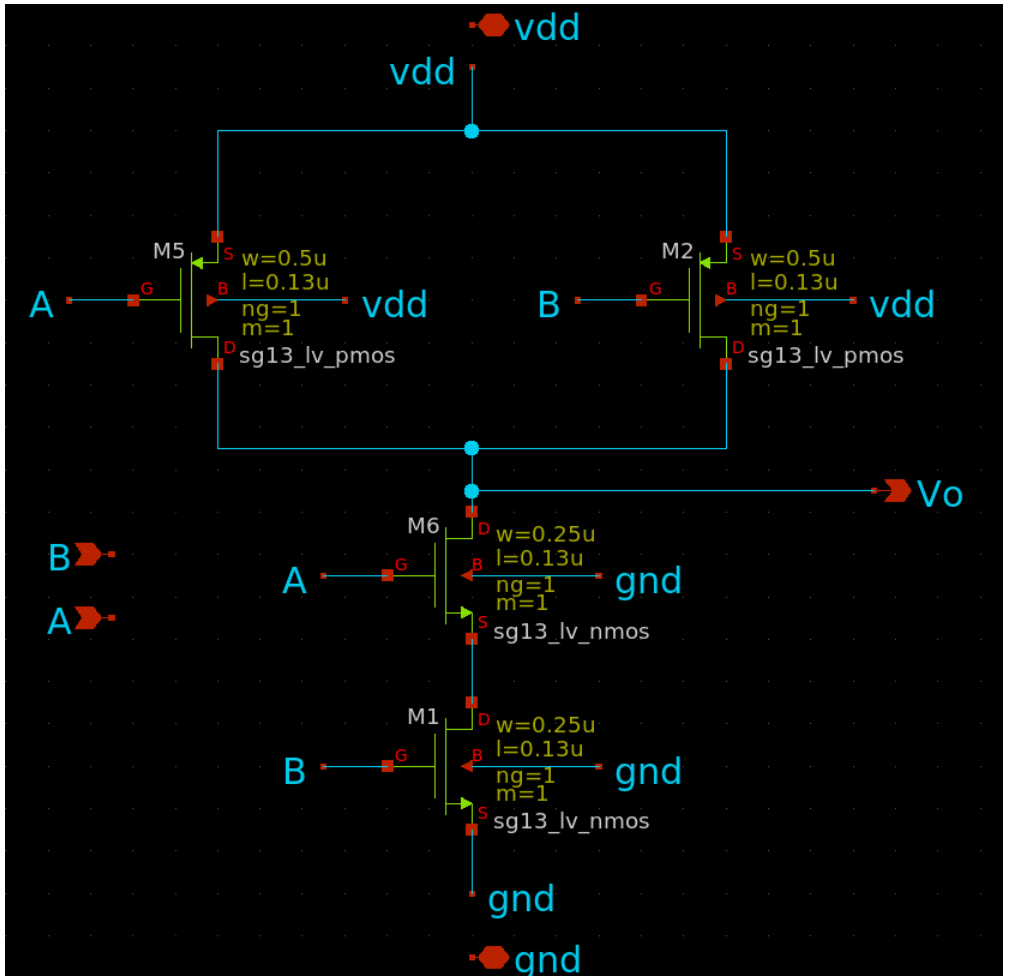
Nand Gate



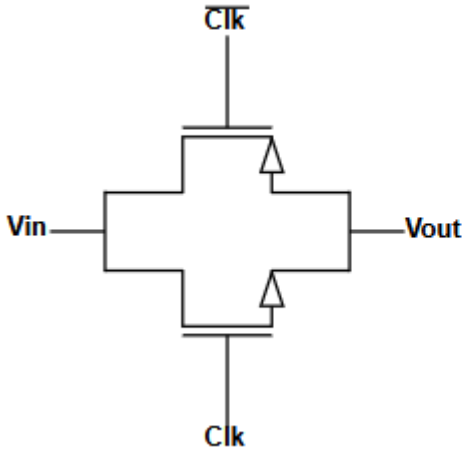
Truth Table

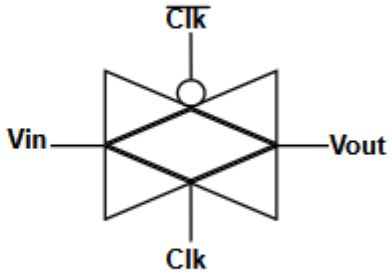
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

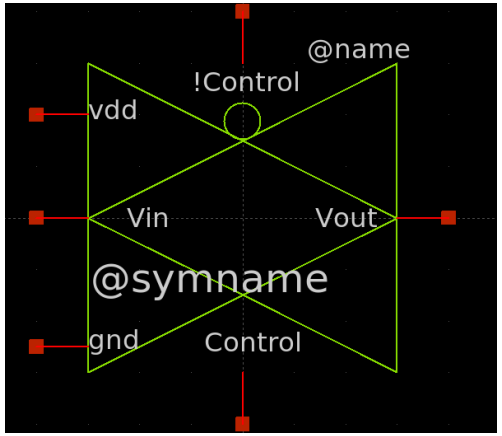


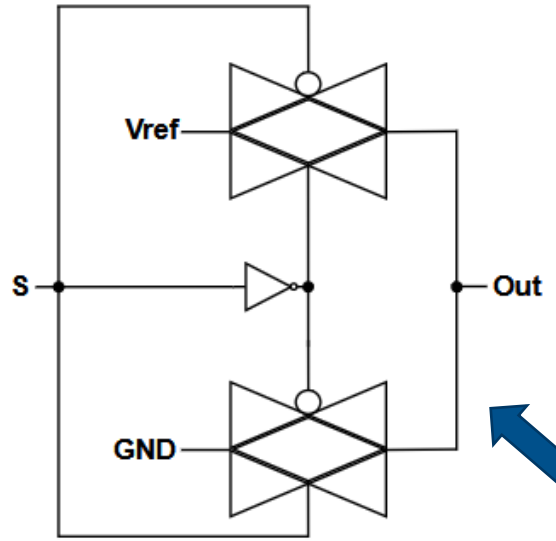


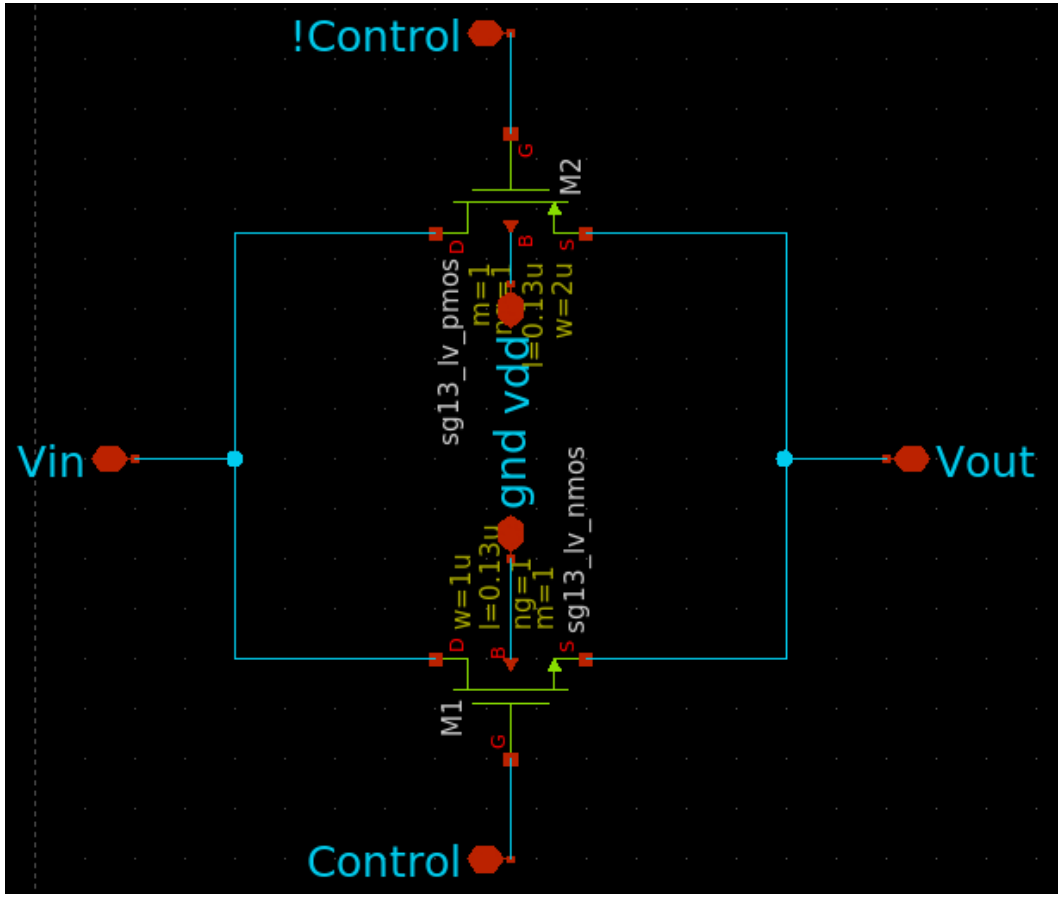
Transmission Gate



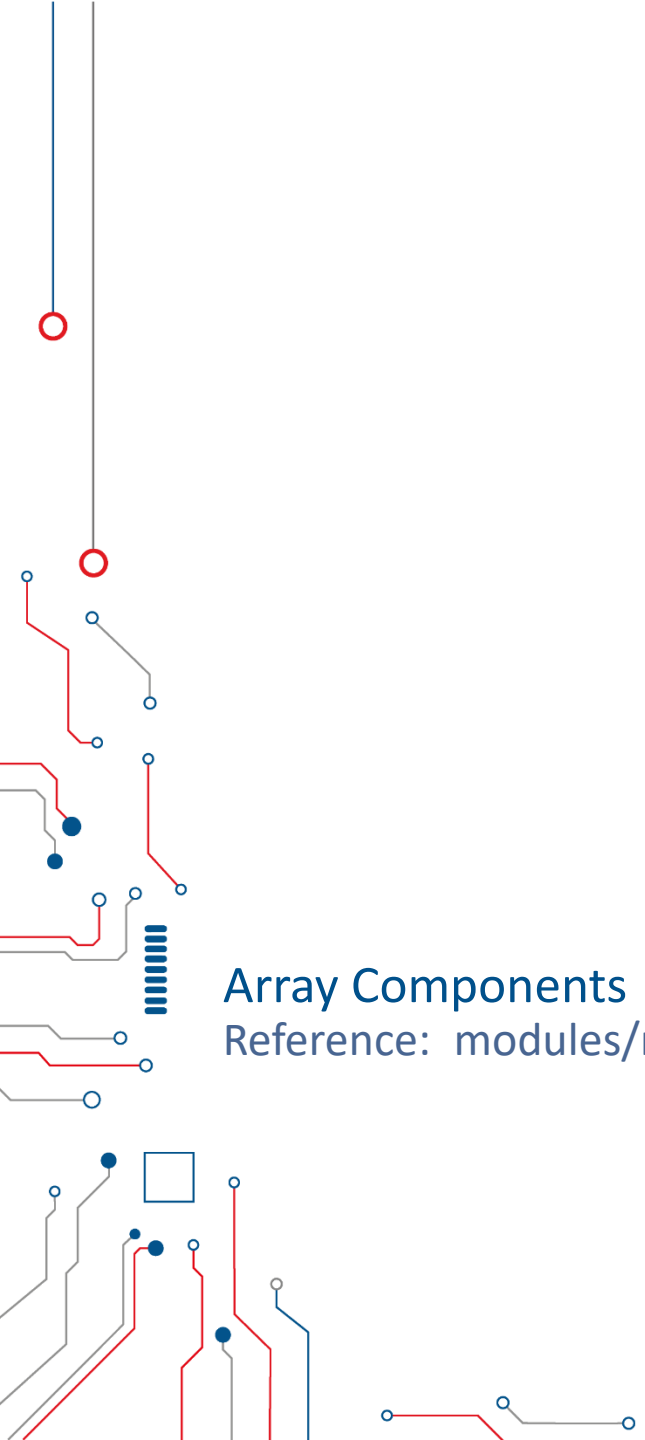








Switch Implementation



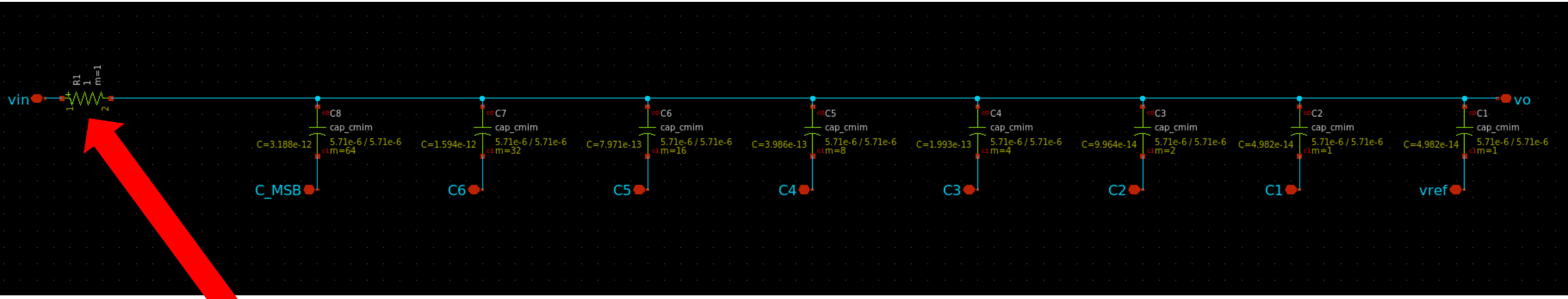
Leibniz Institute
for high
performance
microelectronics

Part 3

Array Components

Reference: modules/module_3_8_bit_SAR_ADC/part_3_array_components

C-DAC (Capacitive DAC)

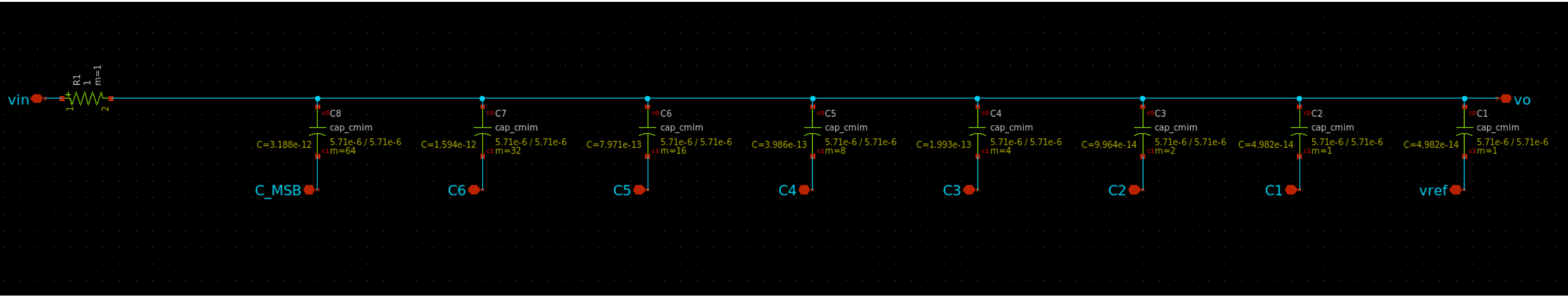


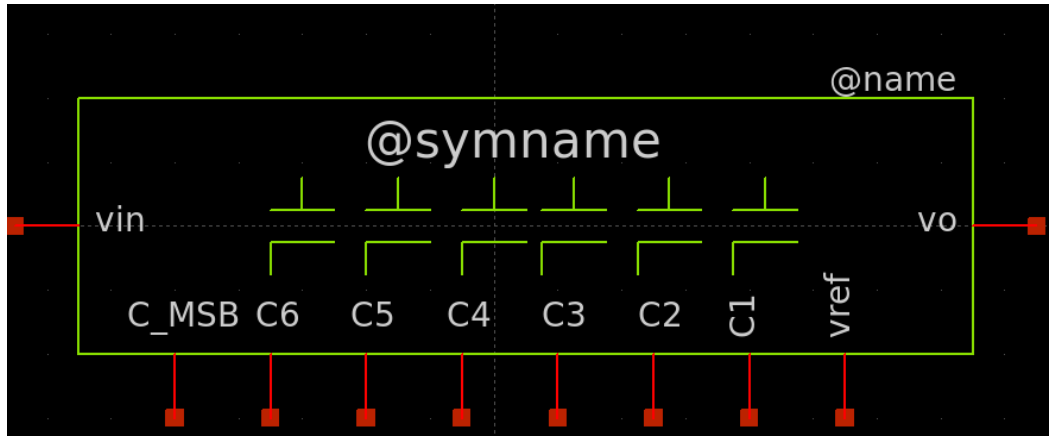
Why??

Capacitor Dimensions (cap_cmim):

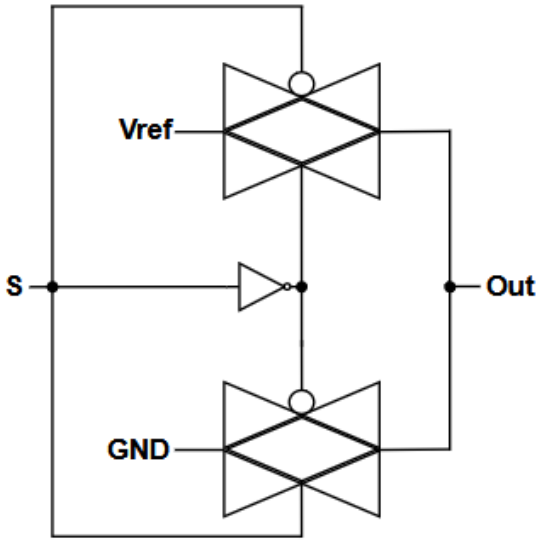
- C1 between vo and vref: W = 5.71 µm, L = 5.71 µm, m = 1
- C2 between vo and C1: W = 5.71 µm, L = 5.71 µm, m = 1
- C3 between vo and C2: W = 5.71 µm, L = 5.71 µm, m = 2
- C4 between vo and C3: W = 5.71 µm, L = 5.71 µm, m = 4
- C5 between vo and C4: W = 5.71 µm, L = 5.71 µm, m = 8
- C6 between vo and C5: W = 5.71 µm, L = 5.71 µm, m = 16
- C7 between vo and C6: W = 5.71 µm, L = 5.71 µm, m = 32
- C_MSB between vo and C_MSB: W = 5.71 µm, L = 5.71 µm, m = 64

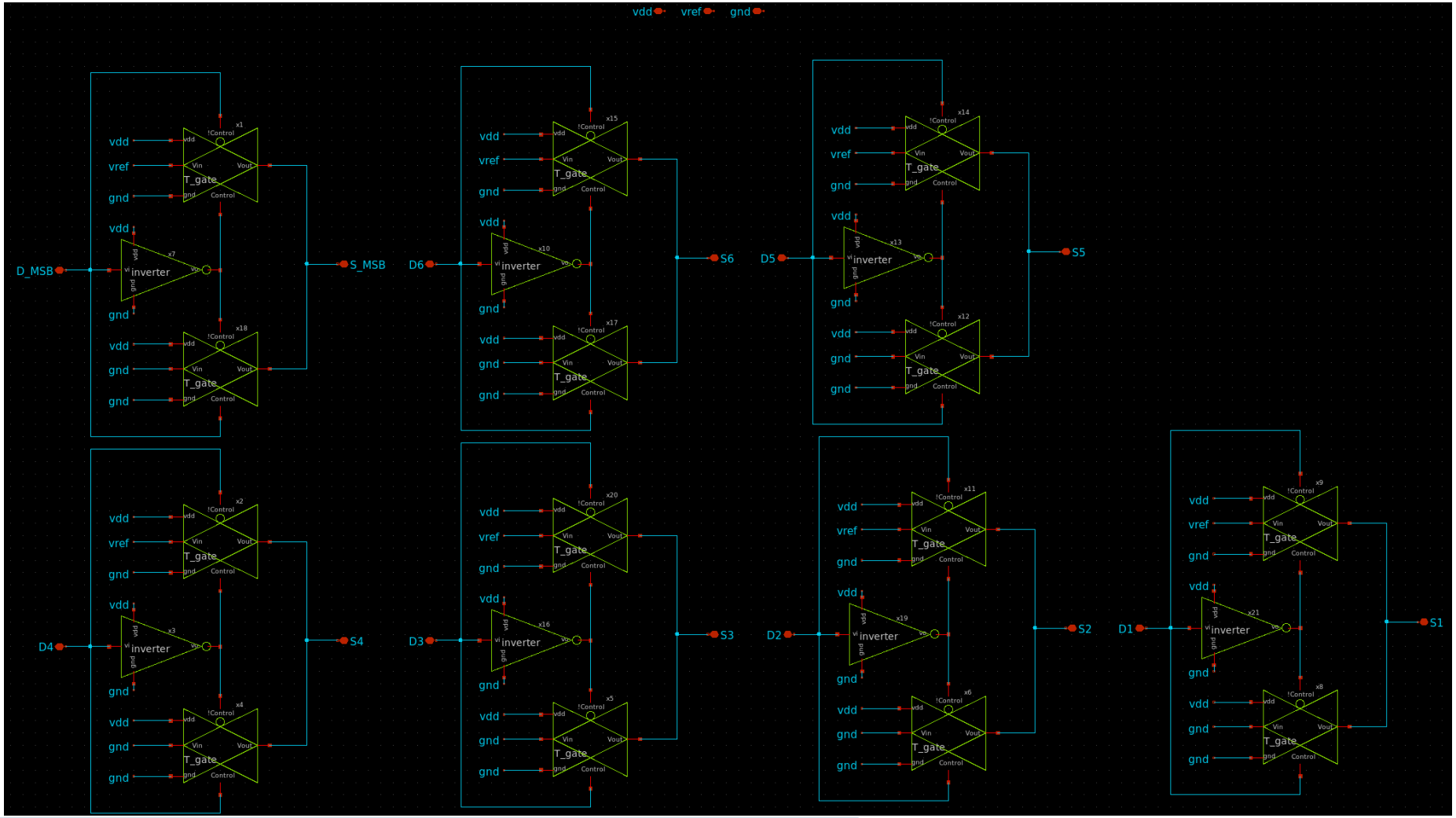
C-DAC (Capacitive DAC)



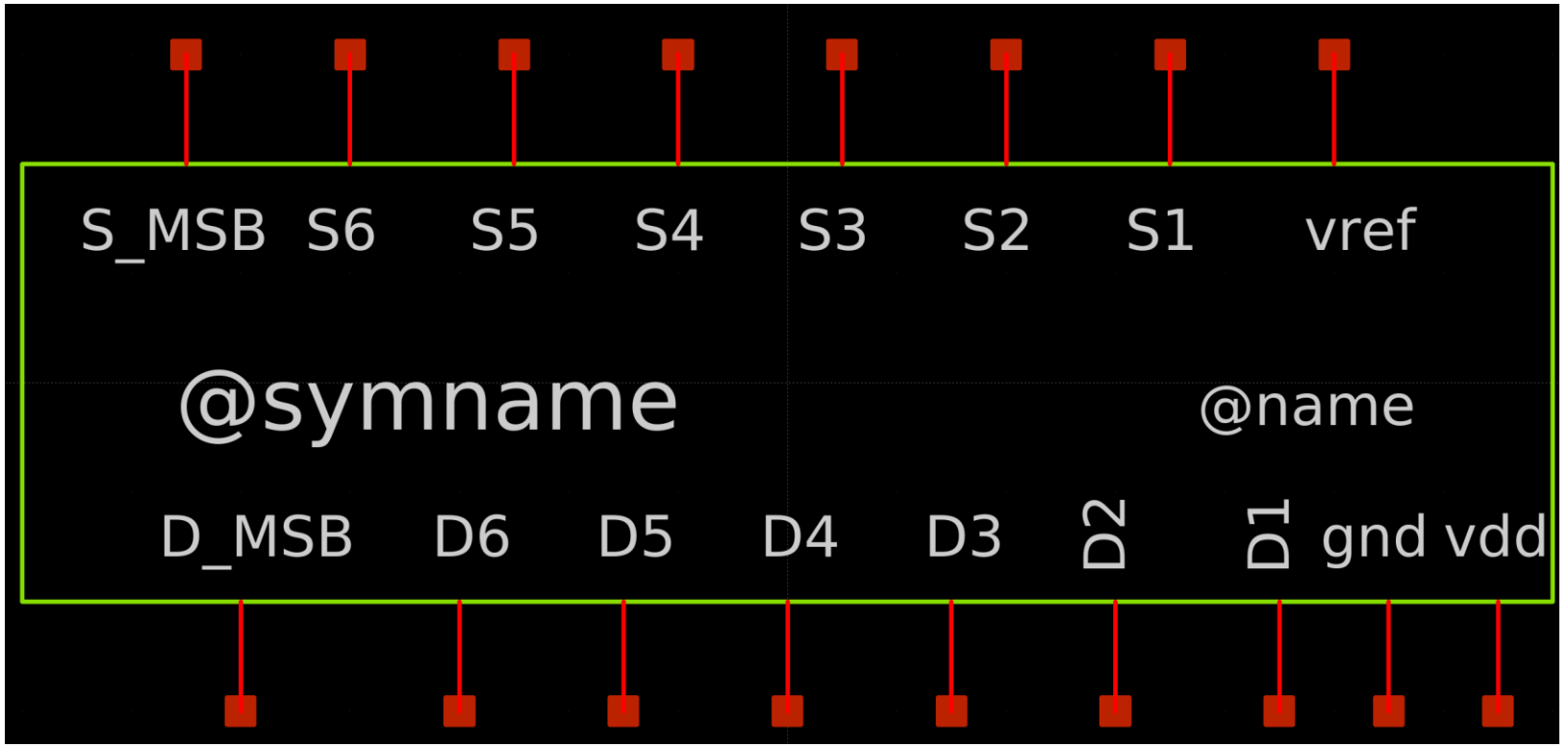


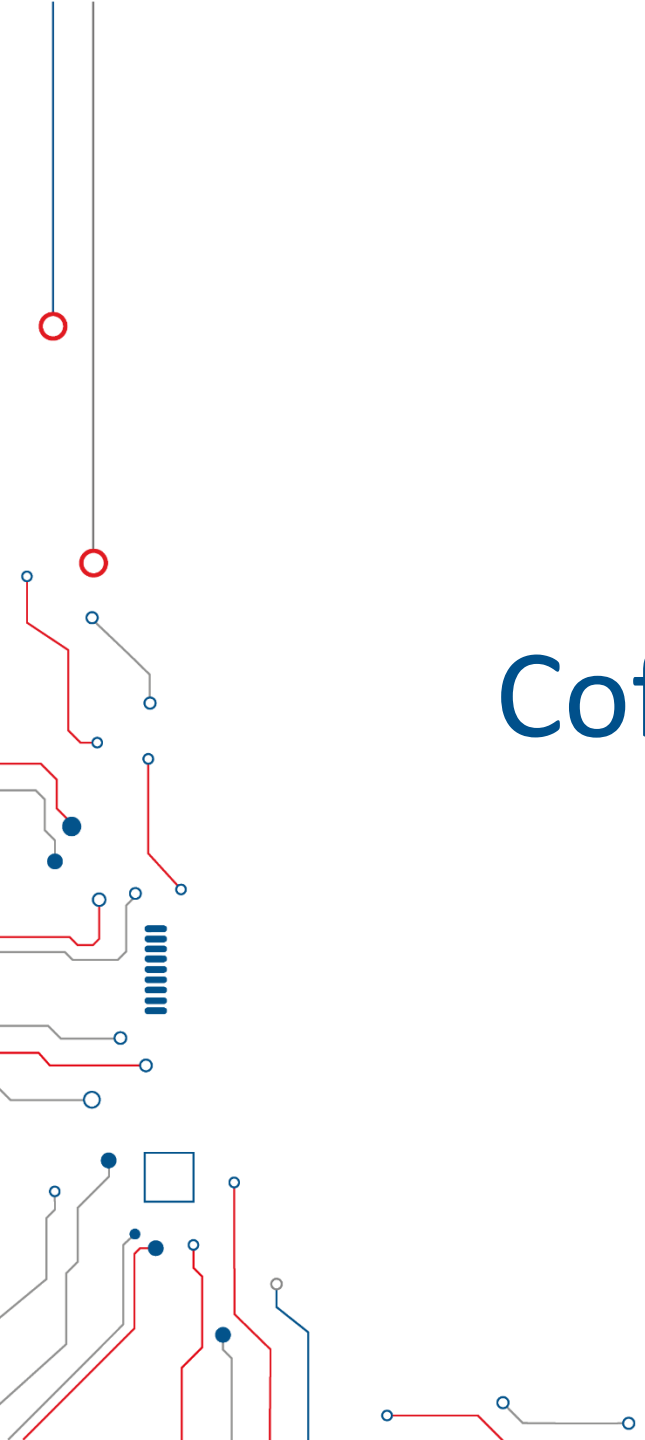
Switch Bank


Vref
S
Out
GND


vdd
vref
gnd
D_MSB
S_MSB
D6
S6
D5
S5
D4
S4
D3
S3
D2
S2
D1
S1
inverter
T_gate
!Control
Vin
Vout
Control

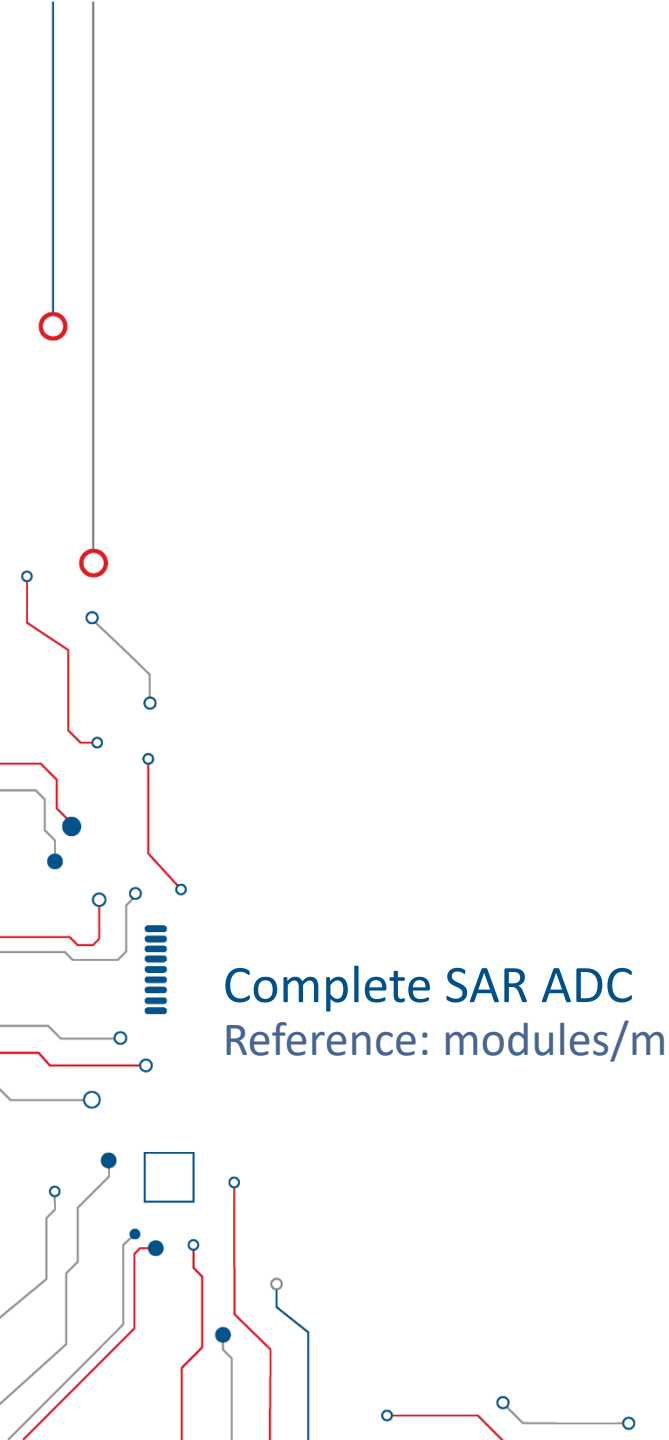
Switch Bank


S_MSB
S6
S5
S4
S3
S2
S1
vref
@symname
@name
D_MSB
D6
D5
D4
D3
D2
D1
gnd
vdd



Leibniz Institute
for high
performance
microelectronics

Coffee Break!! / Catching Up



Leibniz Institute
for high
performance
microelectronics

Part 4

Complete SAR ADC

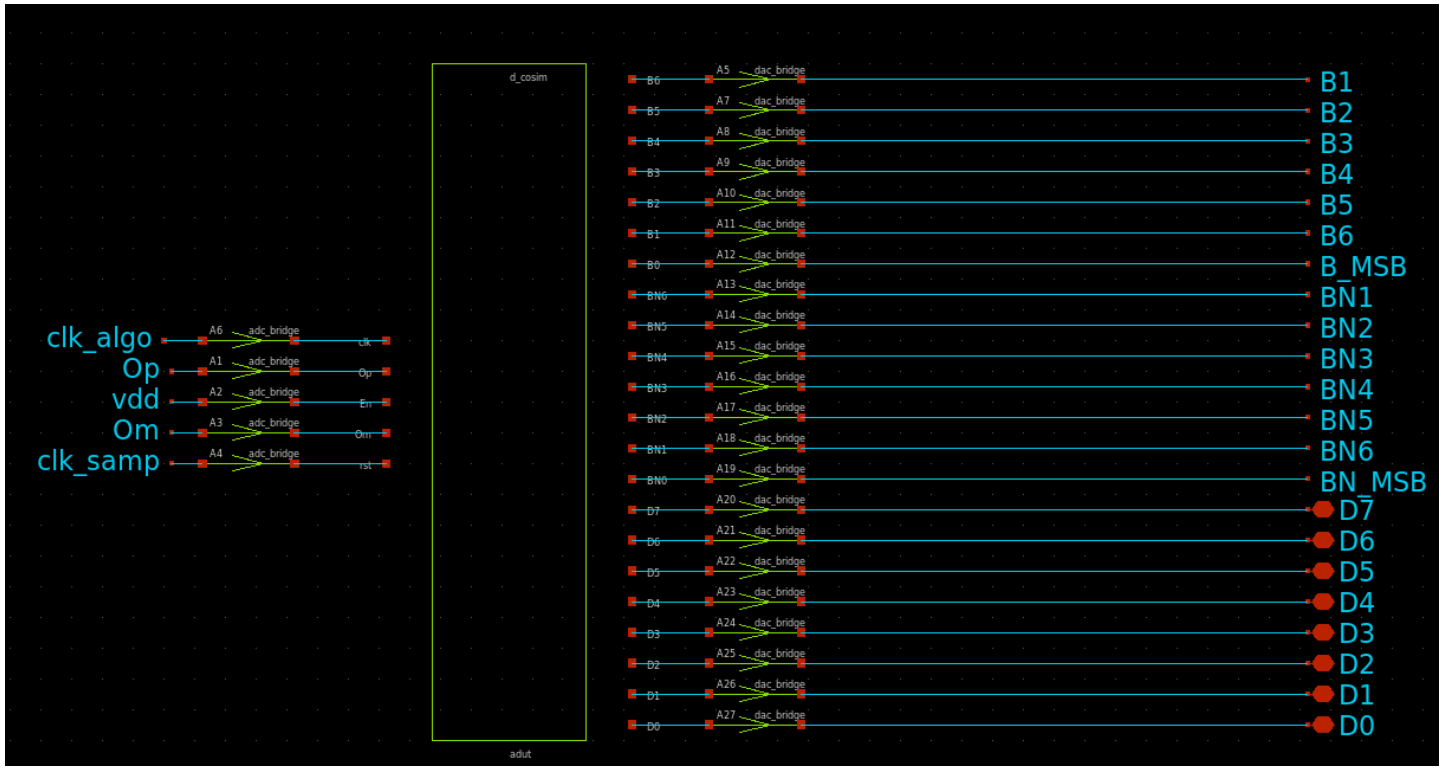
Reference: modules/module_3_8_bit_SAR_ADC/part_4_SAR_ADC

Importing SAR Algorithm

⚠ Note: Make sure to copy the shared object file (.so) and place it relative to the testbench, just like we did before. Create a simulation folder and include it there:

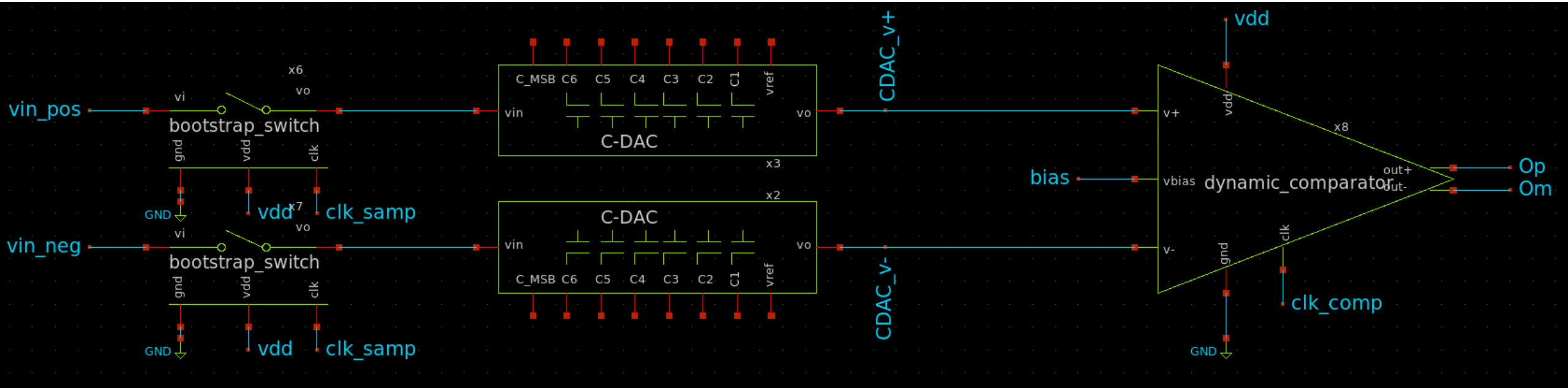
```
├── SAR_ADC_tb.sch   ⟵ Your Schematic
├── simulations      ⟵ Auto generated simulation folder
│   └── sar_logic.so
└── xschemrc
```

Optionally change the make file to automatically dump the .so file at simulations folder

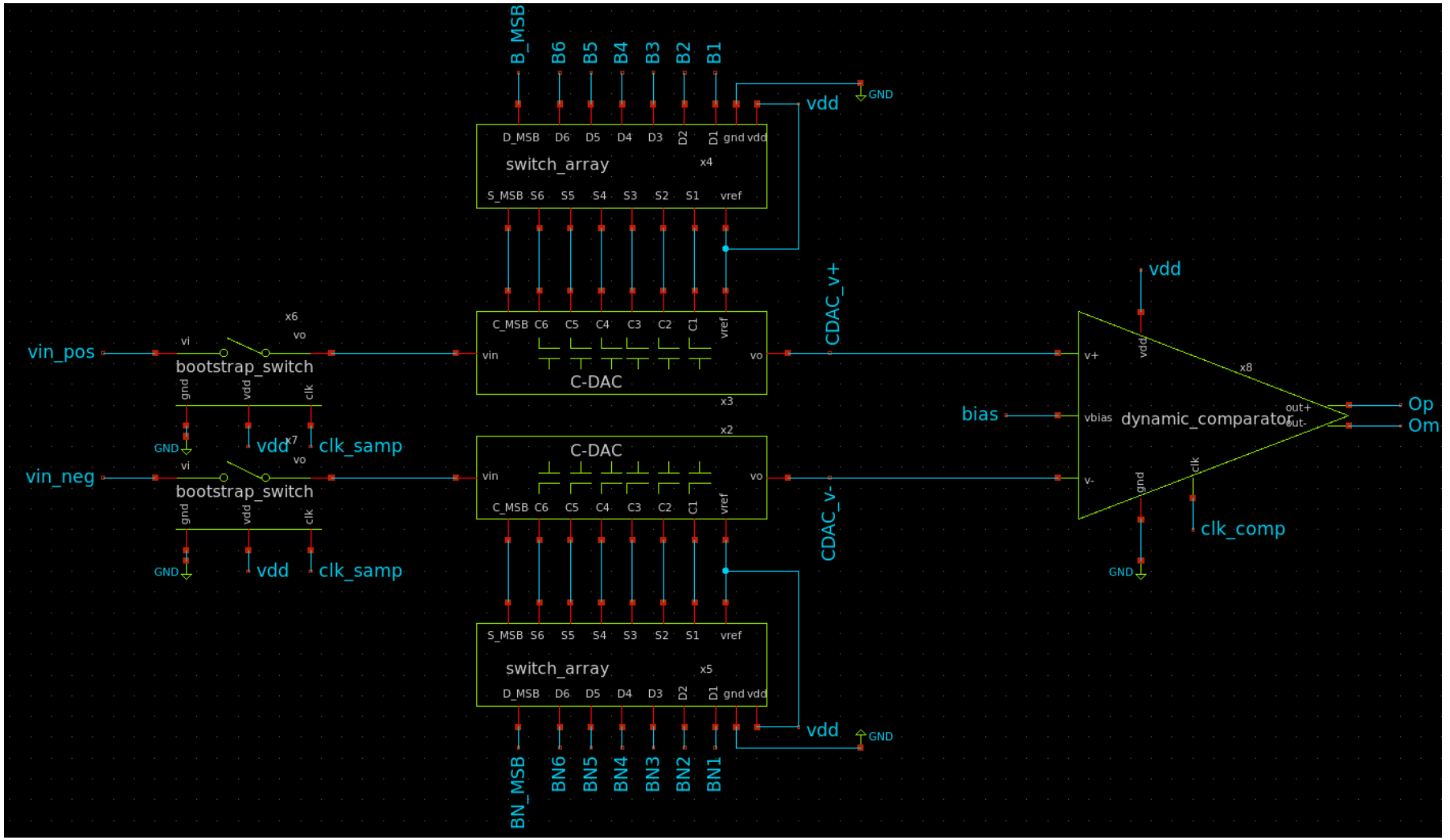


Make sure this works!

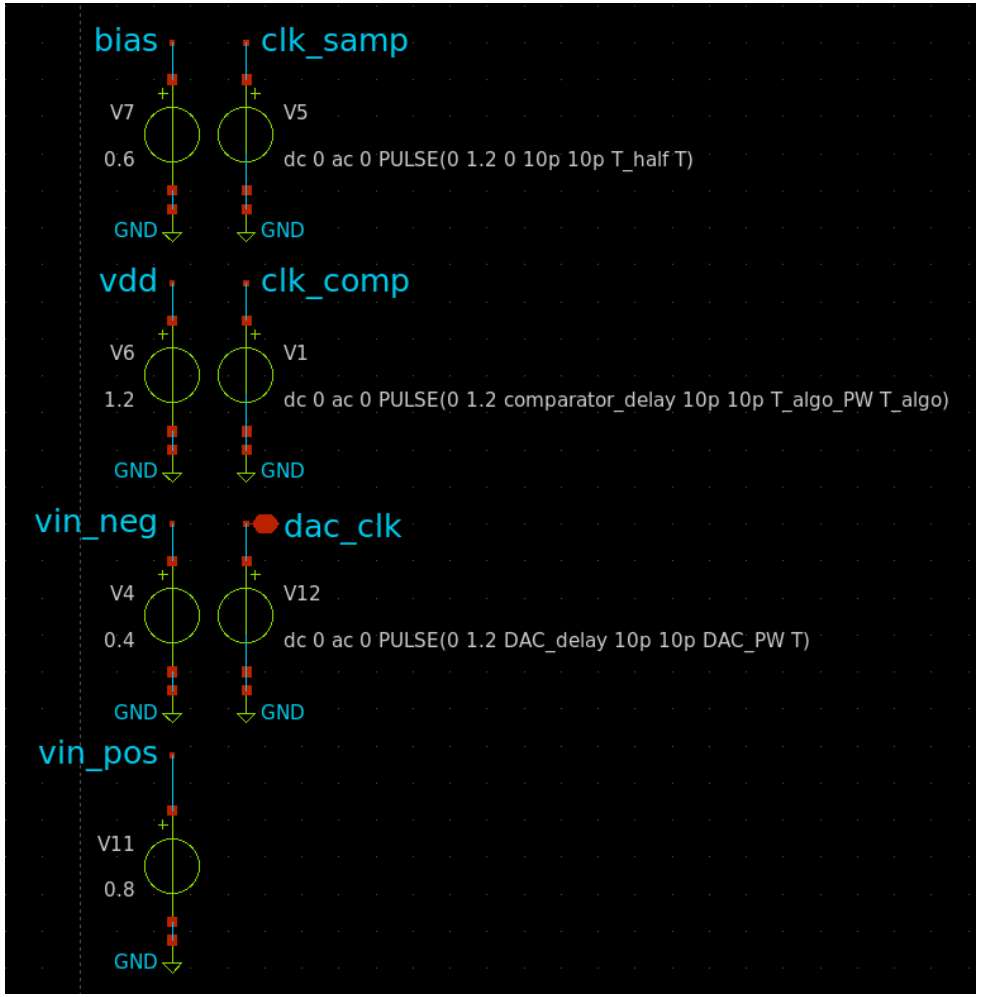
S/H, Comparator, and C-DAC



Switch Bank Incorporation



Testbench Setup



Signals:

bias ➜ *Simple Bias voltage*

clk_samp ➜ *sampling clock for S/H and Reset*

clk_comp ➜ *scaled clock signal for comparison cycles (covers all the bits within one sample window)*

vdd ➜ *Power supply*

vin_neg ➜ *negative input voltage*

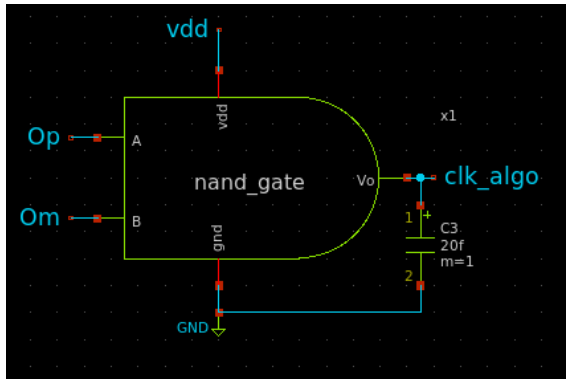
dac_clk➜ *clock for external processing in python*

vin_pos ➜ *positive input voltage*

Op ➜ *positive output of comparator*

Om ➜ *negative output of comparator*

clk_algo ➜ *provides the "clock" for the SAR algorithm*

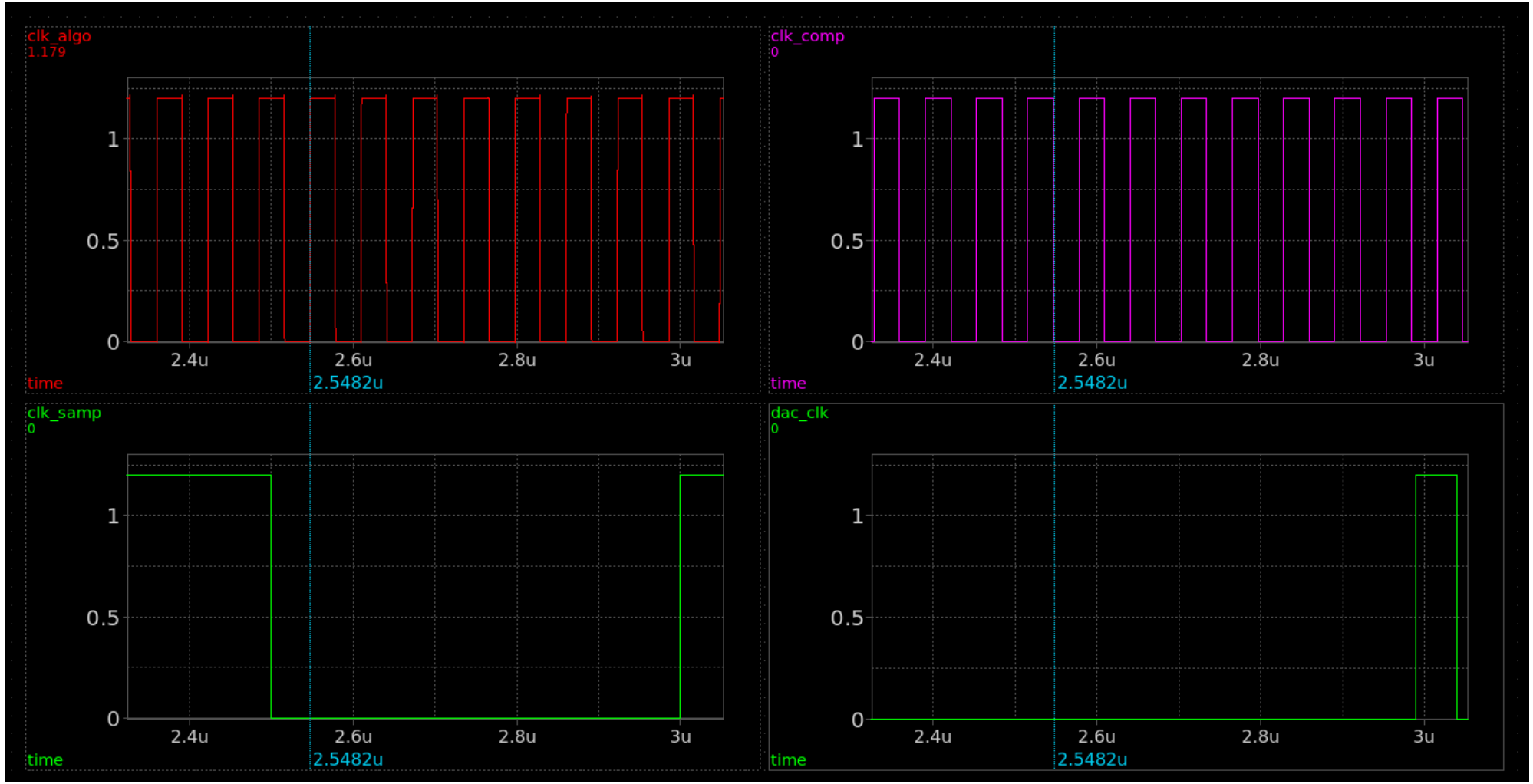


Testbench Setup

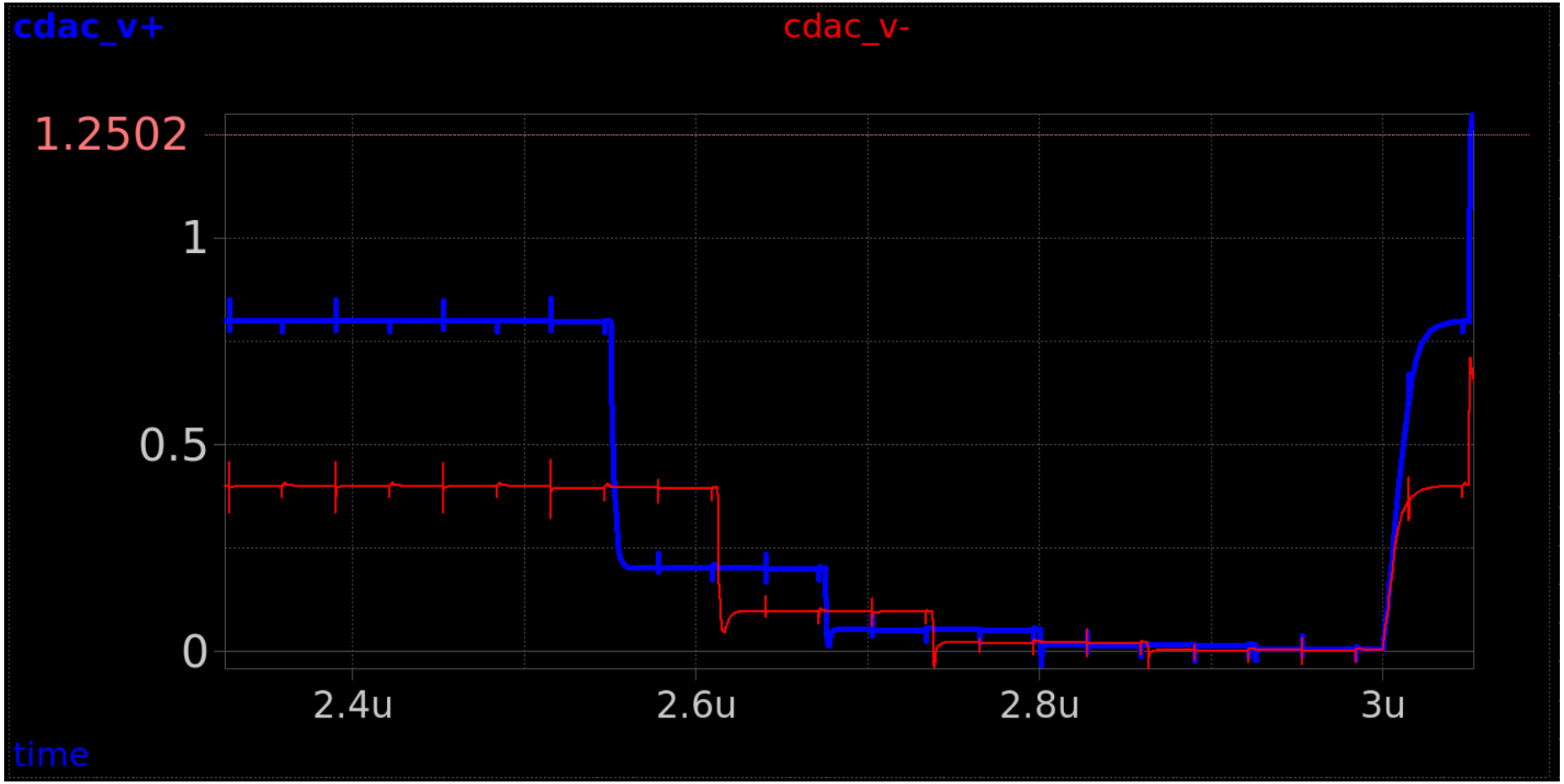
```
name=NGSPICE only_toplevel=true
value="
.param temp=27
.param T = 1u
.param T_half = T/2
.param T_algo = T/16
.param T_algo_delay = T/10
.param T_algo_PW = T/32
.param DAC_delay = 0.99*T
.param DAC_PW = T/20
.param comparator_delay = 0.328*T

.control
tran 1u 8u
let vin_diff = v(Vin_pos) - v(Vin_neg)
let comp_diff = v(op)- v(om)
write sar_adc_test.raw
.endc
"
```

```
name=MODEL only_toplevel=true
format="tcleval( @value )"
value=".lib cornerMOSlv.lib mos_tt
.lib $::SG13G2_MODELS/cornerCAP.lib cap_typ
"
```

Testbench Setup



Result



Post-Processing Using Ideal DAC

```
“
.param temp=27
.param T = 1u
.param T_half = T/2
.param T_algo = T/16
.param T_algo_delay = T/10
.param T_algo_PW = T/32
.param DAC_delay = 0.99*T
.param DAC_PW = T/20
.param comparator_delay = 0.328*T

.control
tran 1u 8u
let vin_diff = v(Vin_pos) - v(Vin_neg)
let comp_diff = v(op)- v(om)
set wr_singlescale
set wr_vecnames
wrdata bit_data.txt D0 D1 D2 D3 D4 D5 D6 D7 vin_diff dac_clk
.endc
"
```

The extracted data comes directly from the SAR logic and includes:

The 8 output bits (D0–D7)

The differential input voltage (vin_diff)

The DAC clock signal (dac_clk), which serves as a virtual clock for reconstructing the analog output using an ideal DAC in Python.

Adjustment to save the data!

Post-Processing Script Path: modules\module_3_8_bit_SAR_ADC\part_4_SAR_ADC\scripting

Post-Processing Using Ideal DAC

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Read the whitespace-separated data
df = pd.read_csv('../simulations/bit_data.txt', sep=r'\s+')

# Display the first few rows
df
```

[1]:

	time	D0	D1	D2	D3	D4	D5	D6	D7	vin_diff	dac_clk
0	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000e+00	0.0
1	1.000000e-13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.787787e-09	0.0
2	2.000000e-13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	9.575574e-09	0.0
3	4.000000e-13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.915115e-08	0.0
4	8.000000e-13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.830230e-08	0.0
...	...	...	...	...	...	...	...	...	...	...	...
601056	3.199908e-04	1.2	0.0	0.0	1.2	0.0	1.2	1.2	0.0	2.344200e-01	1.2
601057	3.199916e-04	1.2	0.0	0.0	1.2	0.0	1.2	1.2	0.0	2.344562e-01	1.2
601058	3.199933e-04	1.2	0.0	0.0	1.2	0.0	1.2	1.2	0.0	2.345284e-01	1.2
601059	3.199966e-04	1.2	0.0	0.0	1.2	0.0	1.2	1.2	0.0	2.346728e-01	1.2
601060	3.200000e-04	1.2	0.0	0.0	1.2	0.0	1.2	1.2	0.0	2.348242e-01	1.2

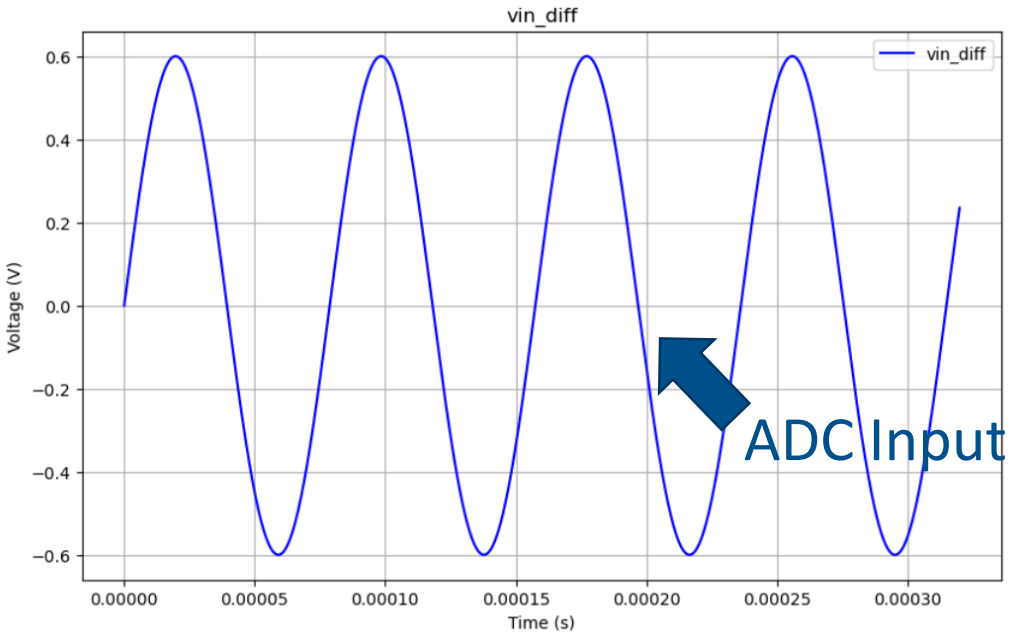
601061 rows × 11 columns

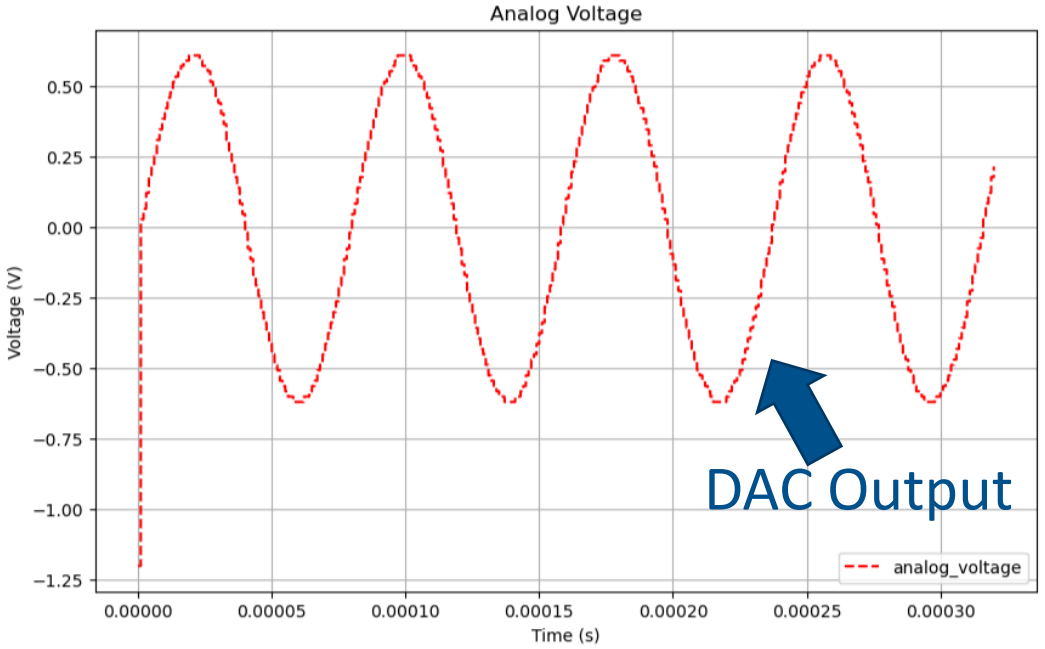
Remaining Task

Apply a differential sinusoidal signal and recover it using the ideal DAC in the python script.

The plot below can be used as inspiration on what to expect and the guide on how to do this is available in the markdown file if you get stuck:

module_3_8_bit_SAR_ADC/part_4_SAR_ADC/sar_adc.md





Alternatively

Analyze Performance Metrices: Use the python script to assess some of the performance metrices of the ADC (ENOB, SNR, DNL etc…)

Design Competition: Work on your layout for the big design competition!!

Catch-Up: Work on material from today or the other days to catch up or clarify points that wasn’t clear

Relax ☺