4. Full-Custom IC Design Methodology

Francesc Serra Graells

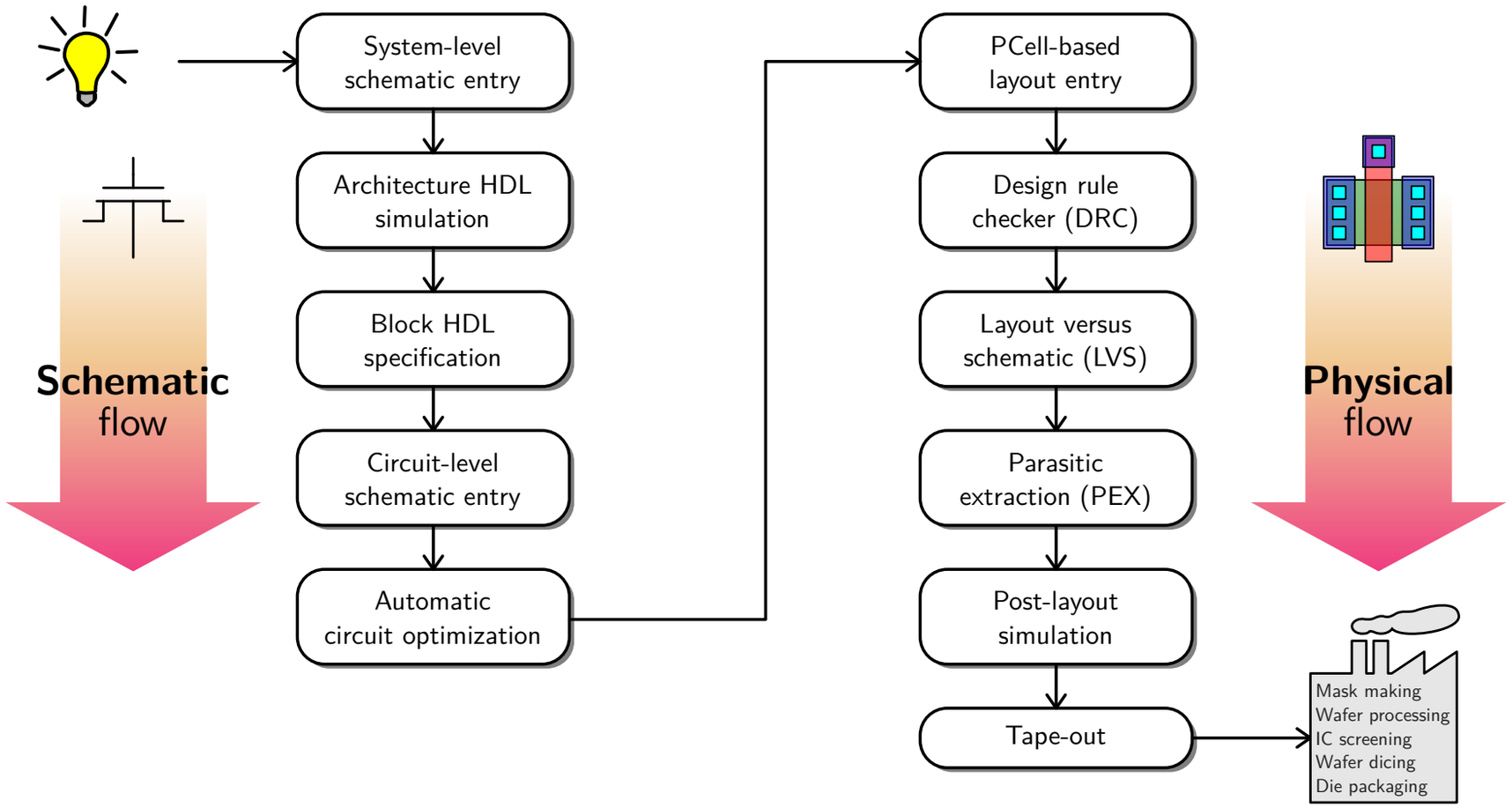
francesc.serra.graells@uab.cat
Departament de Microelectrònica i Sistemes Electrònics
Universitat Autònoma de Barcelona

paco.serra@imb-cnm.csic.es
Integrated Circuits and Systems
IMB-CNM(CSIC)

Full-Custom Analog IC Design Flow

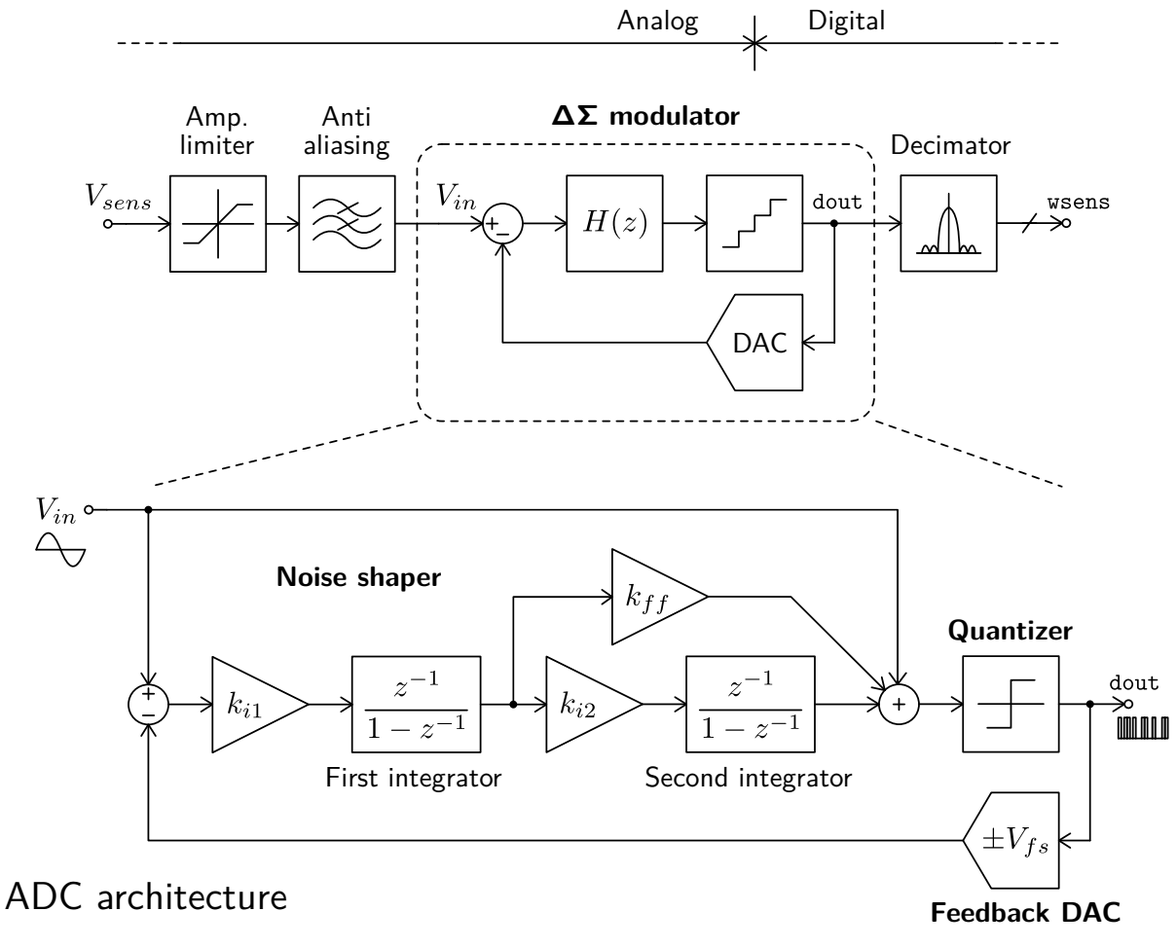
- From circuit **idea to layout** masks...

▲ **EDA**-assisted methodology

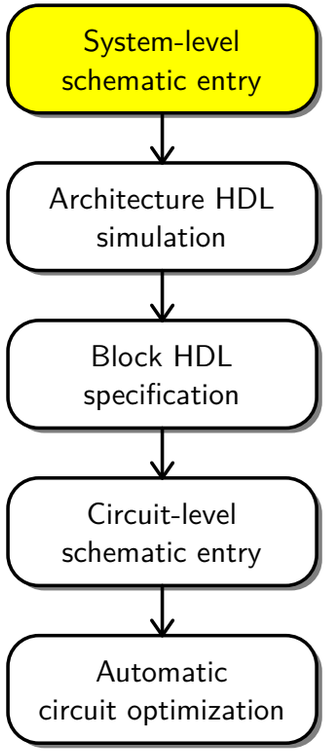


Full-Custom Schematic Design

- **Architecture** selection according to system and application specifications:



e.g. **ΔΣ** modulation ADC architecture



- Signal processing performance
- Communications standards
- Power constraints
- Testbility requirements
- ... and many more!

Full-Custom Schematic Design

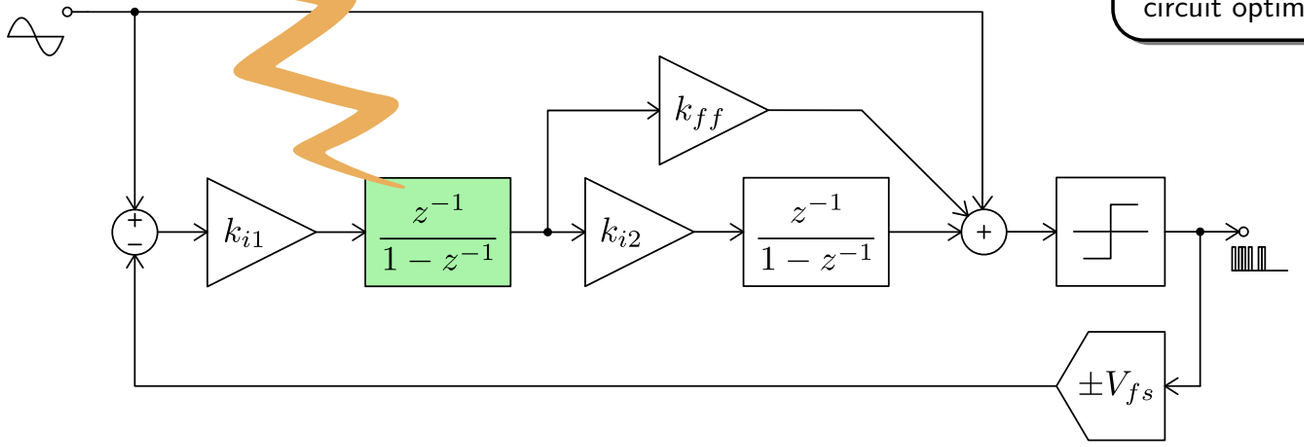
- System modeling through any hardware descitpion language (**HDL**):

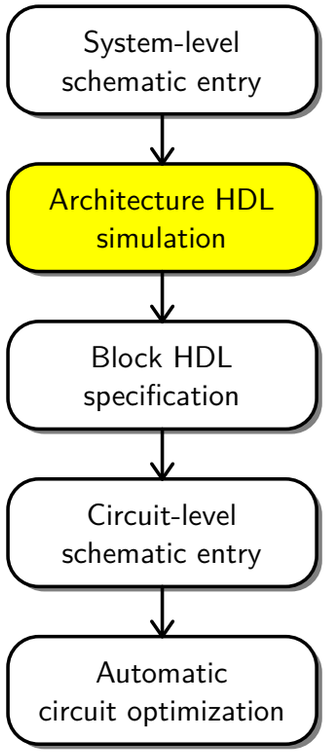
zinteg2lim.mod

```
void cm_zinteg2lim(ARGS) {
 inp = INPUT(inp);            /* Retriving input values */
 clk = INPUT_STATE(clk);
 pos_edge = PARAM(pos_edge);  /* Retrieving parameters */
 out_max = PARAM(out_ax);
 ...
 switch (ANALYSIS) {
  case TRANSIENT:
   if ((*clk_mem==ONE)&&(clk==ZERO)) {/* Neg. clk edge */
    if (pos_edge==FALSE)
     action = SAMPLING_INTEGRATION;
   } else {
    if ((*clk_mem==ZERO)&&(clk==ONE)) {/* Pos. clk edge*/
     if (pos_edge==TRUE)
      action = SAMPLING_INTEGRATION;
    } else {                          /* No clock edge */
     action = HOLDING;
   }
 ...
   switch (action) {
    case SAMPLING_INTEGRATION:
     *inp_mem = inp;
     out = *out_mem+*inp_mem;
     if (out<out_min) { out = out_min; }     /* Limiter */
     if (out>out_max) { out = out_max; }
     *out_mem = out;
     break;
    case HOLDING:
     out = *out_mem;
}}}
```

e.g. **XSpice**

- Architecture descritpion
- **Simplified** functional modeling of each block
- Second-order circuit effects still **neglected**





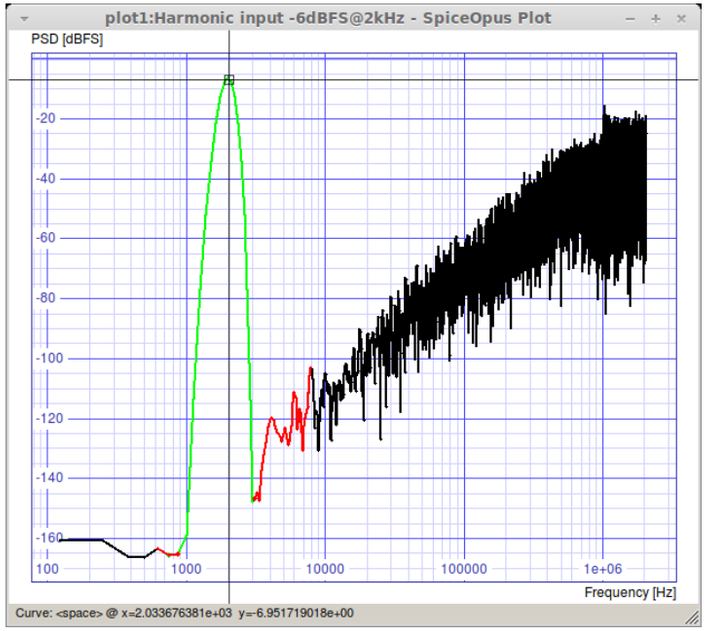
Full-Custom Schematic Design

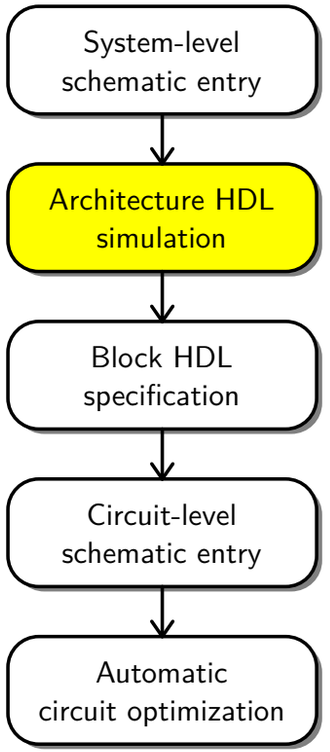
- HDL numerical simulation of the full system using **event-based** engines:
  - Architecture evaluation
  - Simulation **speed-up** by orders of magnitude!
  - **Ideal** response (maximum possible performance)

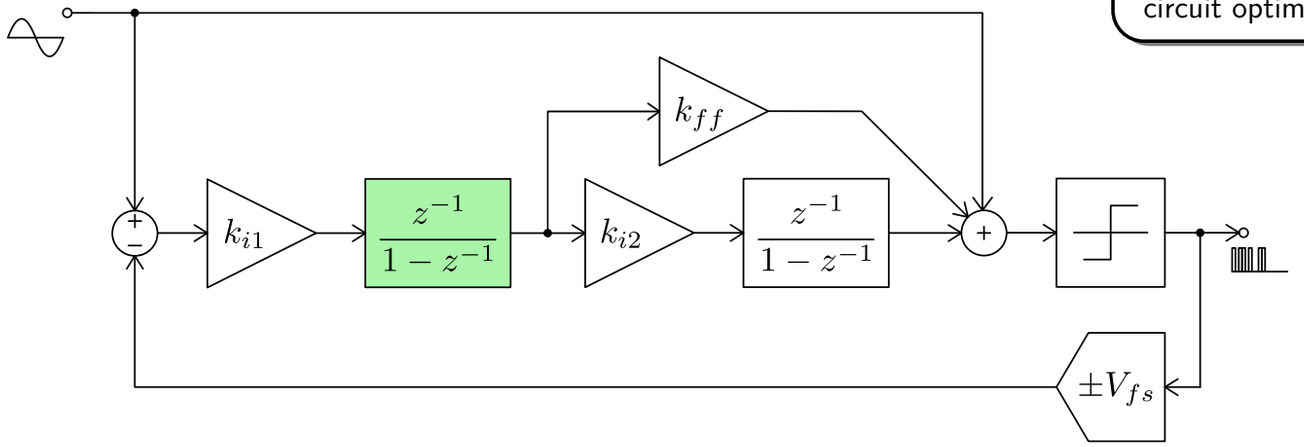
dsm-arch.sub

```
.subckt dsm_arch vin dclk dout
asumin [%v(vin) %v(vdac)] %v(verr) msumin
.model msumin usummer(sign=[1.0 -1.0])
aki1 %v(verr) %v(vint1in) mki1
.model mki1 kgain(k=0.3)
azint1 %v(vint1in) %d(dclk) %v(vint1out) mzint1
.model mzint1 zinteg2lim(pos_edge=0 out_ic=0.0
 + out_min=-5.0 out_max=5.0)
aki2 %v(vint1out) %v(vint2in) mki2
.model mki2 kgain(k=0.7)
azint2 %v(vint2in) %d(dclk) %v(vint2out) mzint2
.model mzint2 zinteg2lim(pos_edge=0 out_ic=0.0
 + out_min=-5.0 out_max=5.0)
akff %v(vint1out) %v(vkffout) mkff
.model mkff kgain(k=2.0)
asumout [%v(vint2out) %v(vkffout) %v(vin)]
 + %v(vquantin) msumout
.model msumout usummer(sign=[1.0 1.0 1.0])
aquant %v(vquantin) %d(~dclk) %d(dout) mquant
.model mquant quant2lsh(inp_th=0.0 out_ic=0 pos_edge=0
 + t_rise=1e-9 t_fall=1e-9)
adac %d(dout) %v(vdac) mdac
.model mdac dac2lsym(out_level=2.0)
.ends
```

e.g. **SpiceOpus**

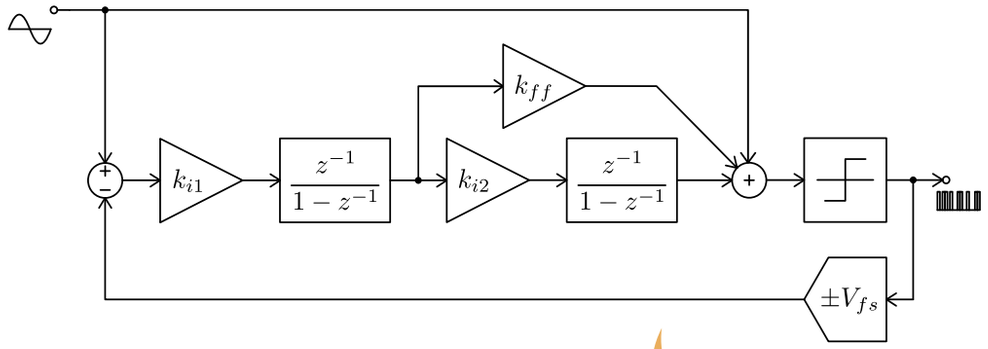


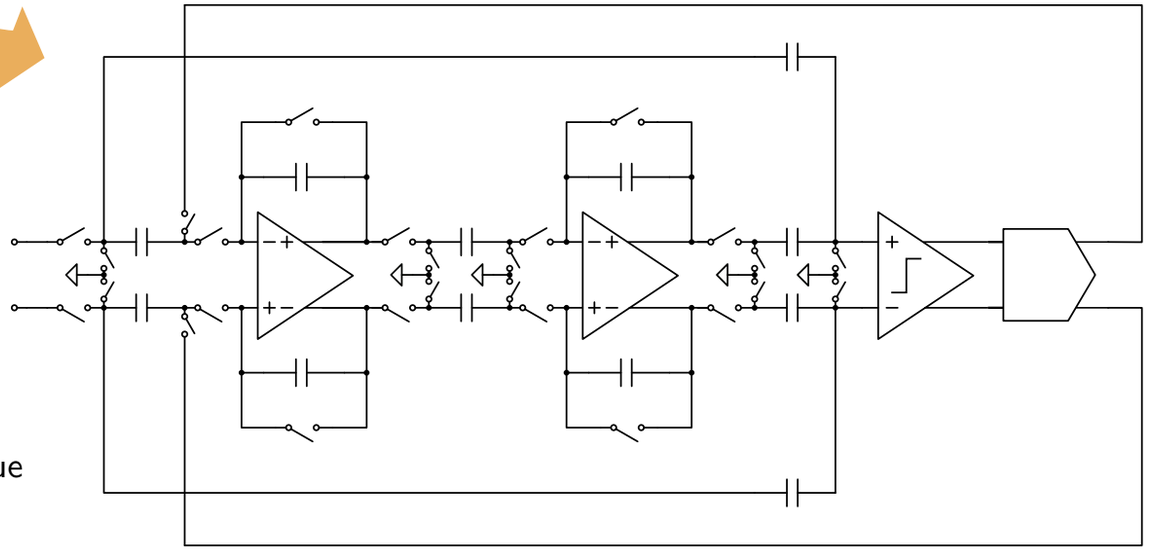




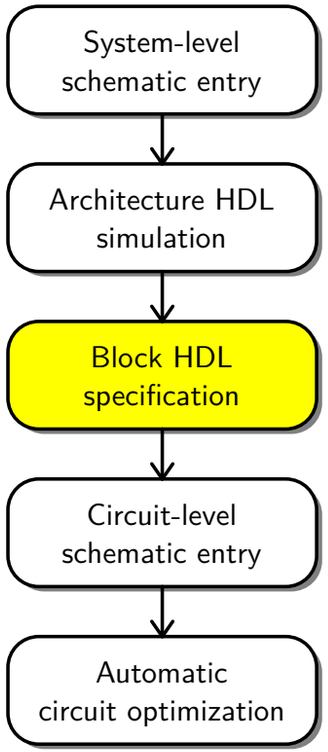
Full-Custom Schematic Design

- Choosing most suitable **circuit technique** according to:
  - **CMOS** technology options
  - Circuit sensitivity against process, supply and temperature (**PVT**)
  - IC **operation** conditions (calibration, testability...)
  - **External** components available



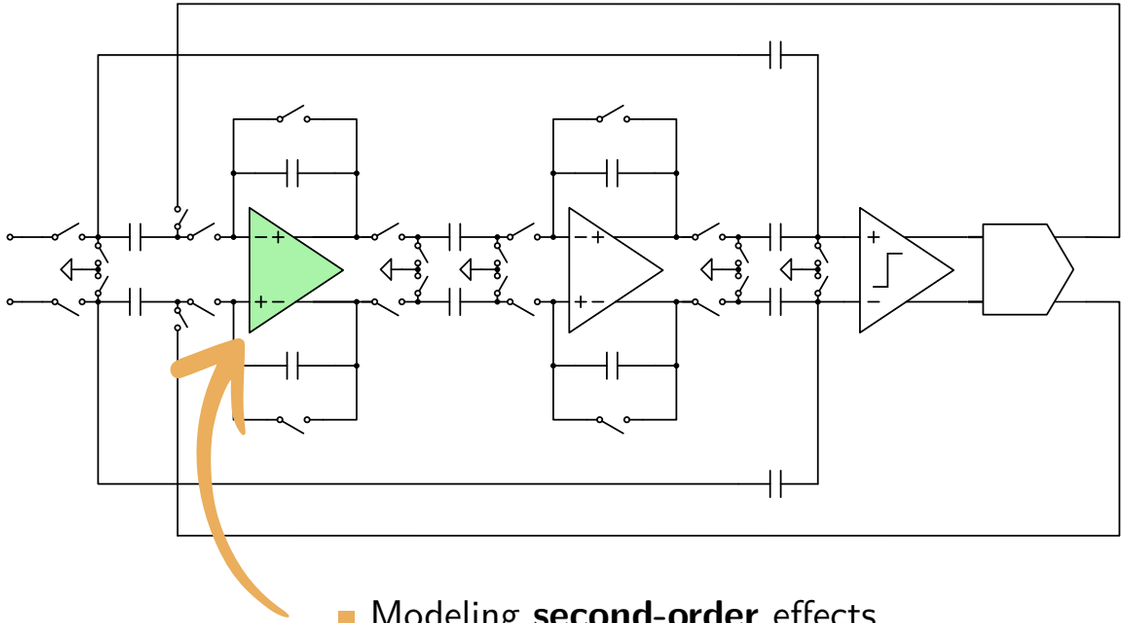


e.g. switched-capacitor (**SC**) technique



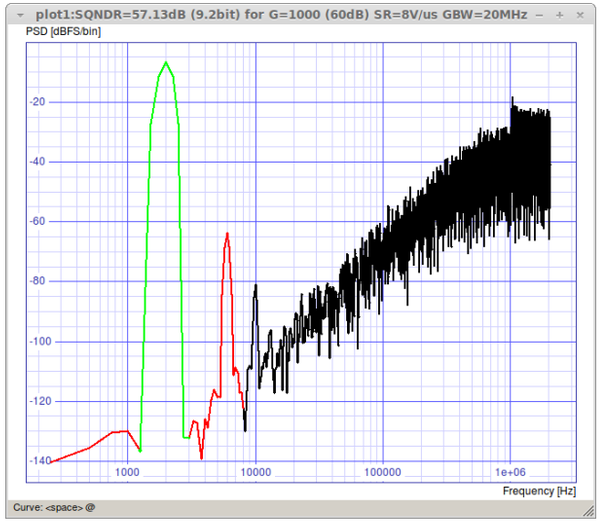
Full-Custom Schematic Design

- **Splitting** system design problem into independent **blocks**:



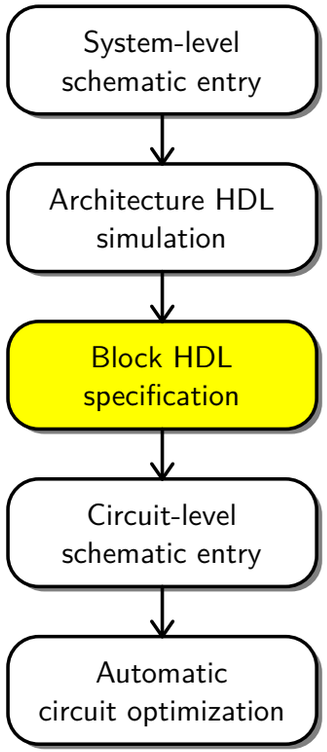
- Modeling **second-order** effects due to limited circuit performance
- Electrical simulation of each individual **block** is feasable
- Transistor-level **final** simulation of overall system is still required!

e.g. limited {G,GBW,SR} in **Opamp** model



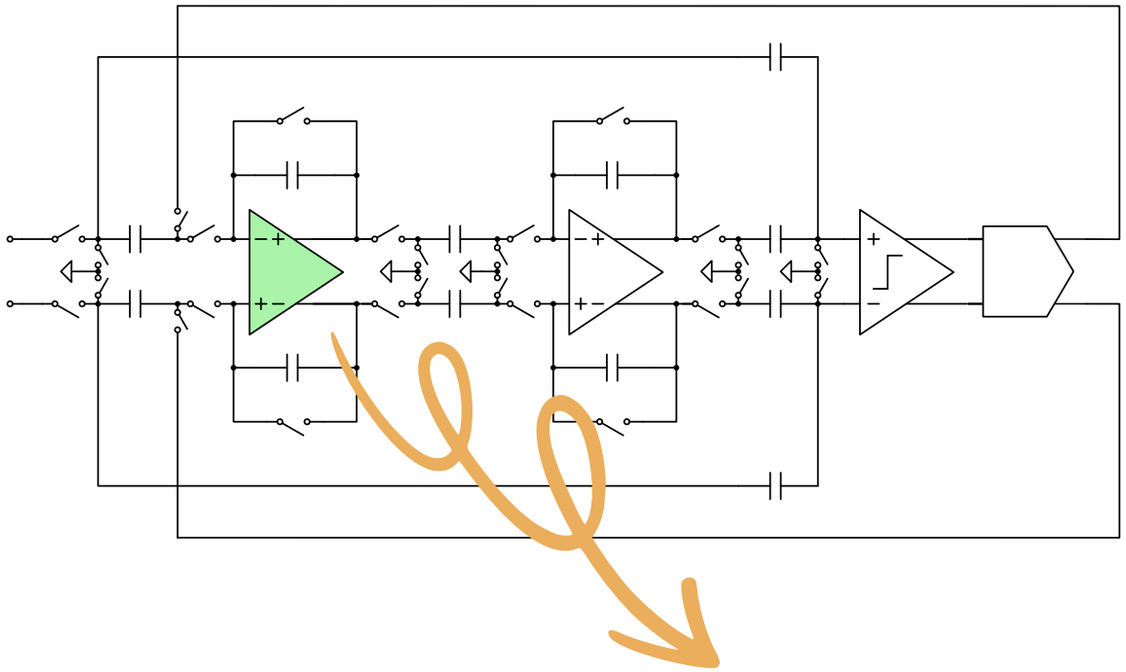
opamp.ifs

```
NAME_TABLE:
Spice_Model_Name:  opamp
C_Function_Name:   cm_opamp
Description:       "OpAmp macro"

PORT_TABLE:
Port_Name:         inp           inn           out
Description:       "pos. input"  "neg. input"  "output"
Direction:         in            in            out
Default_Type:      v             v             v
...
PARAMETER_TABLE:
Parameter_Name:    G             GBW           SR
Description:       "DC OL gain"  "GBW"         "slew-rate"
Data_Type:         real          real          real
Default_Value:     1000          1e6           1e6
Limits:            [0 -]         [0 -]         [0 -]
...
PARAMETER_TABLE:
Parameter_Name:    out_min                 out_max
Description:       "lower out limit"       "upper out limit"
Data_Type:         real                    real
Default_Value:     0                       5.0
Limits:            [0 5.0]                 [0 5.0]
...
```



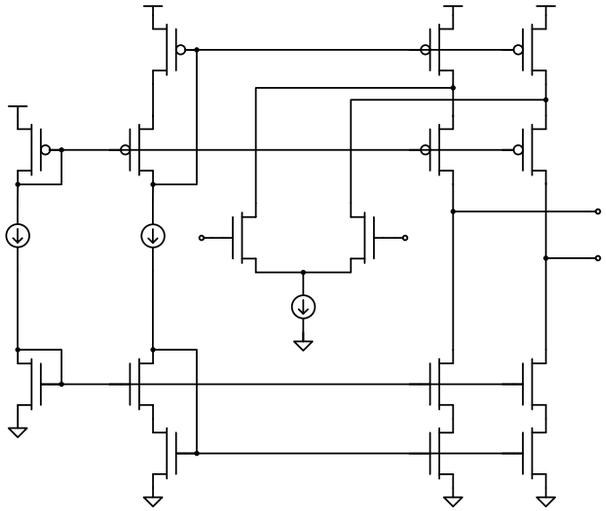
Full-Custom Schematic Design

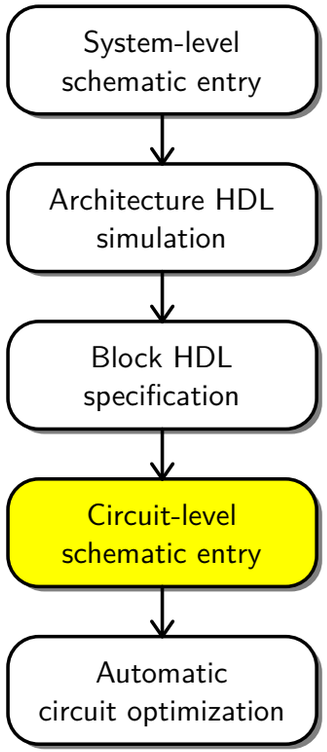
- Choosing the particular **circuit topology** for each system block:



e.g. fully-differential cascode and folded **OpAmp**

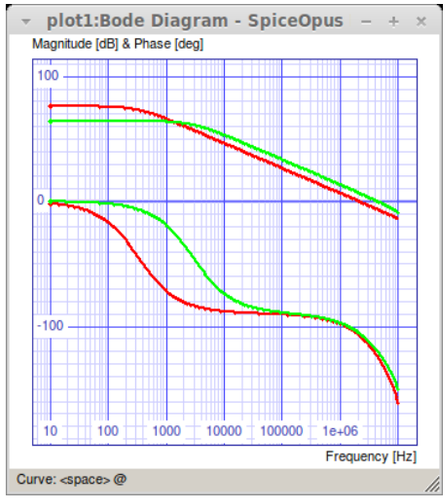
- **Separated** block testbenches
- **Target specs** from previous step e.g. {G,GBW,SR}
- Fast and accurate **electrical simulations**
- **Several** circuit topologies can be easily investigated for each functional block
- Inter-block **coupled** effects not covered!

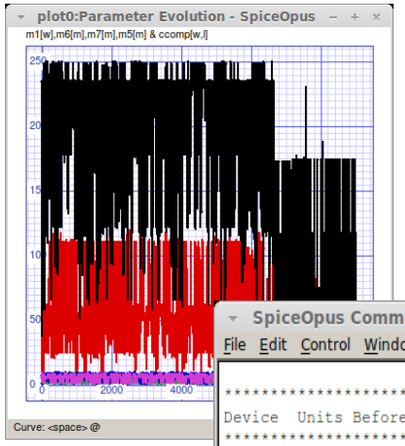


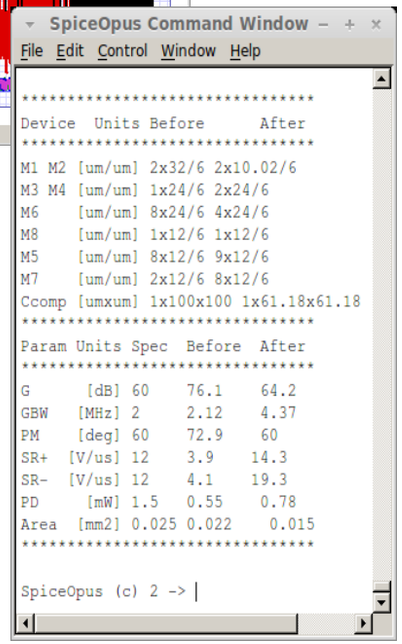


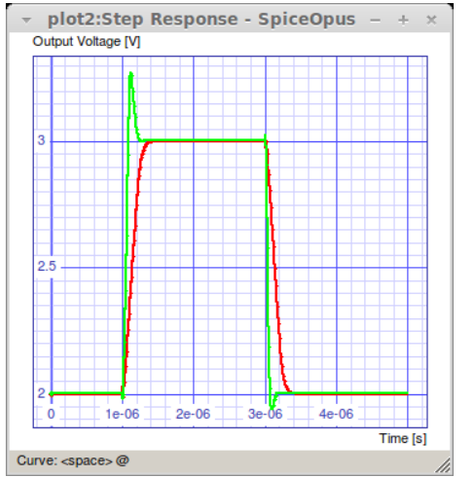
Full-Custom Schematic Design

- Device **size optimization** at block level:





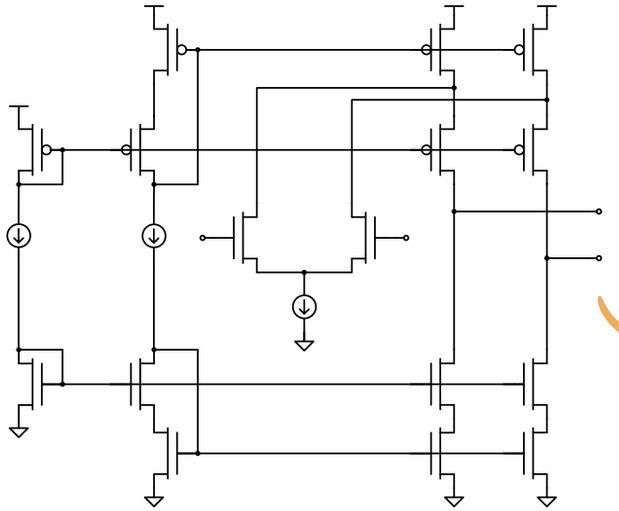




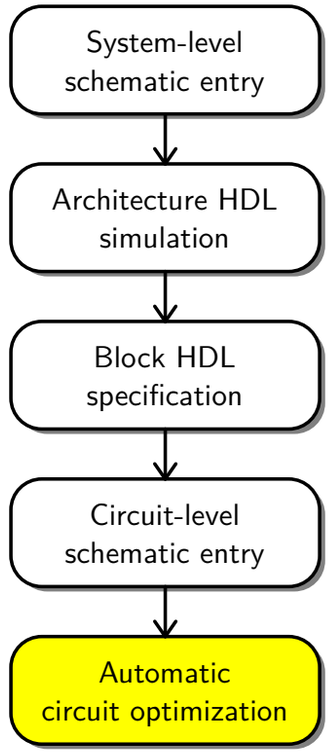
e.g. OpAmp **automatic** optimization

opamp_optimize.sp3

```
optimize
   parameter 0 @m1:xopamp[w] low 6u high 120u initial 32u
 " parameter 1 @m6:xopamp[m] low 1 high 10 initial 8
 " parameter 3 @ccomp:xopamp[w] low 25u high 250u
...
 " analysis 25 ac dec 50 10 10e6
 " analysis 26 let gmag=20*log10(mag(v(vout)))
 " analysis 27 let gph=phase(v(vout))
 " analysis 28 cursor c right gmag 0
 " analysis 29 let gbw=abs(frequency[%c])/1e6
 " analysis 30 let pm=180+gph[%c]
...
 " analysis 46 tran 1n 5u
 " analysis 47 cursor c right vout 2.1
 " analysis 48 let t1=time[%c]
 " analysis 49 cursor c right vout 2.9
 " analysis 50 let t2=time[%c]
 " analysis 51 let srpos=0.8/(t2-t1)*1e-6
...
 " implicit 0 op2.pd lt 1.5
 " implicit 1 op2.area lt 0.025
 " implicit 4 ac2.pm gt 60
 " implicit 5 tran2.srpos gt 12
 ...
 " cost 1/tran2.srneg+1/tran2.srpos+abs(60-ac2.pm)
 " method genetic elitism yes maxgen 1000
```

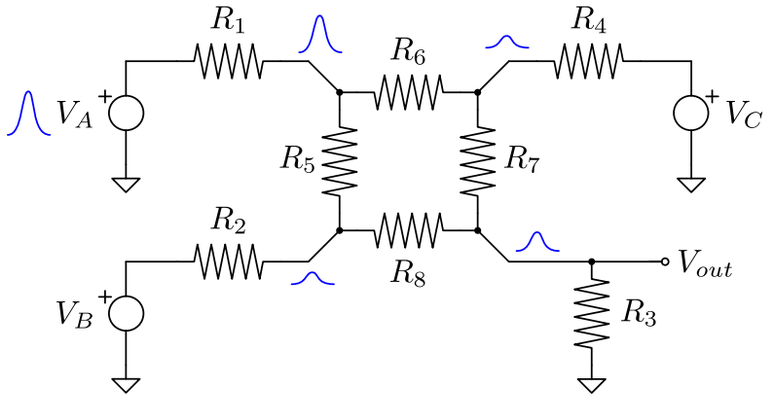


$$M \times \frac{W}{L}$$



Electrical vs Event Simulation

- Circuit as network of **physical devices**:

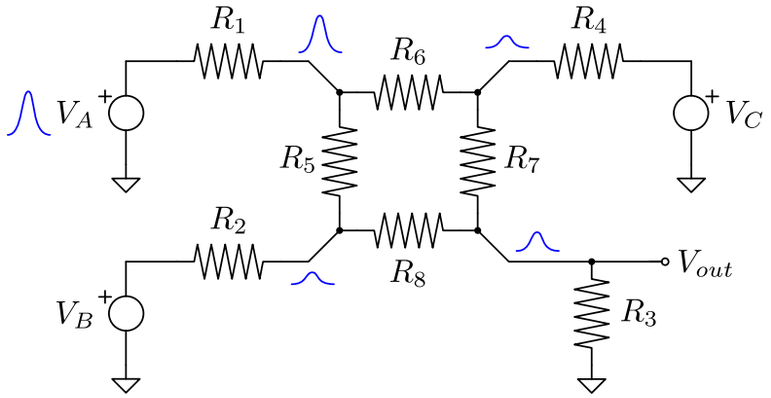


  - **Electrical** modeling
  - **Impedance** coupling
  - Any change requires resolving the **full net**

  - **Accurate** results
  - **Time consuming** simulation

Electrical vs Event Simulation

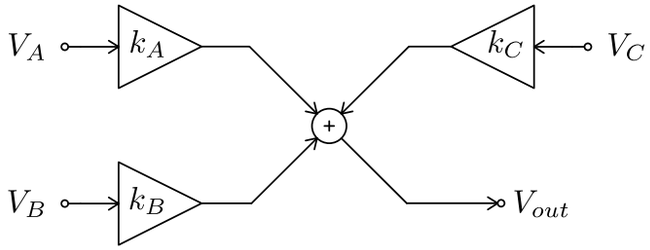
- Circuit as network of **physical devices**:



  - **Electrical** modeling
  - **Impedance** coupling
  - Any change requires resolving the **full net**

  - **Accurate** results
  - **Time consuming** simulation

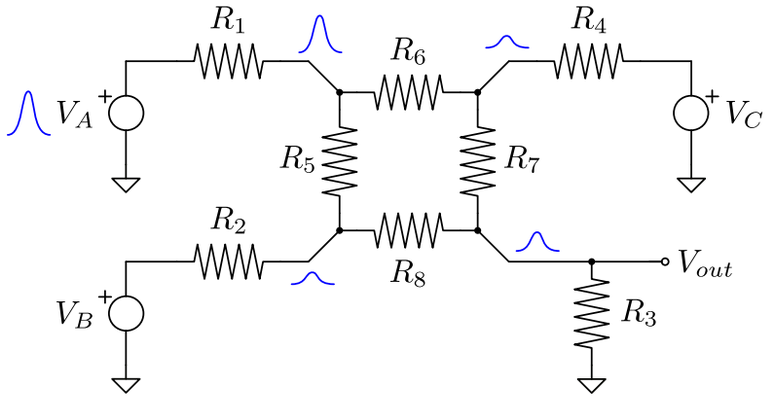
- Circuit as chain of **black boxes**:



$$V_{out} = k_A V_A + k_B V_B + k_C V_C$$

Electrical vs Event Simulation

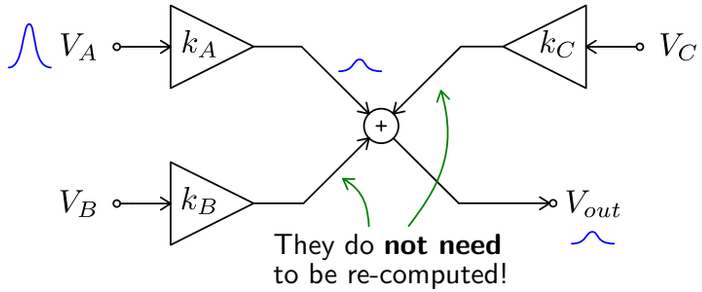
- Circuit as network of **physical devices**:



- **Electrical** modeling
- **Impedance** coupling
- Any change requires resolving the **full net**

- **Accurate** results
- **Time consuming** simulation

- Circuit as chain of **black boxes**:



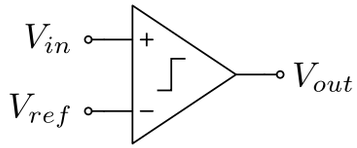
$$V_{out} = k_A V_A + k_B V_B + k_C V_C$$

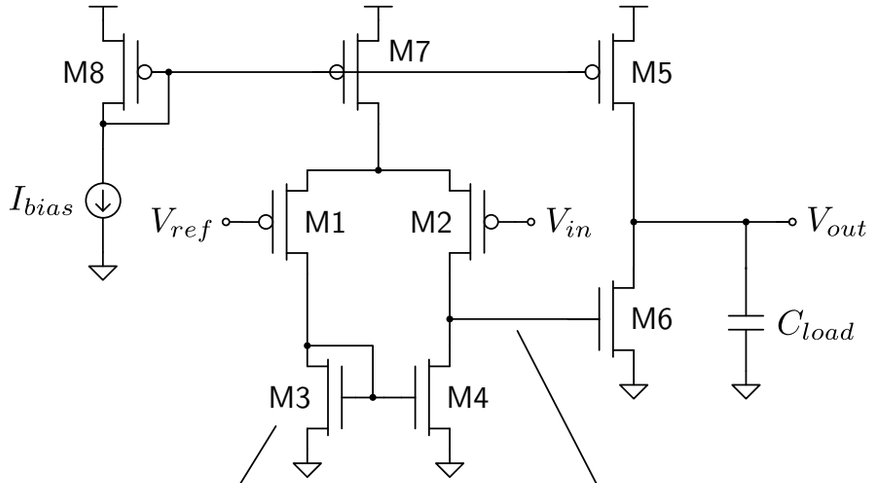
- **Behavioral** modeling
- Cause-effect **event** generation
- Only resolving event **propagation**

- **Fast** simulation
- **Simplified** predictions

Electrical vs Behavioral Modeling

- Example: **voltage comparator**



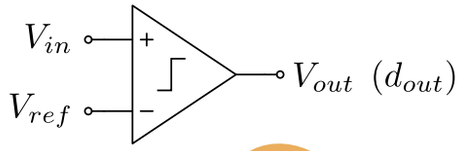


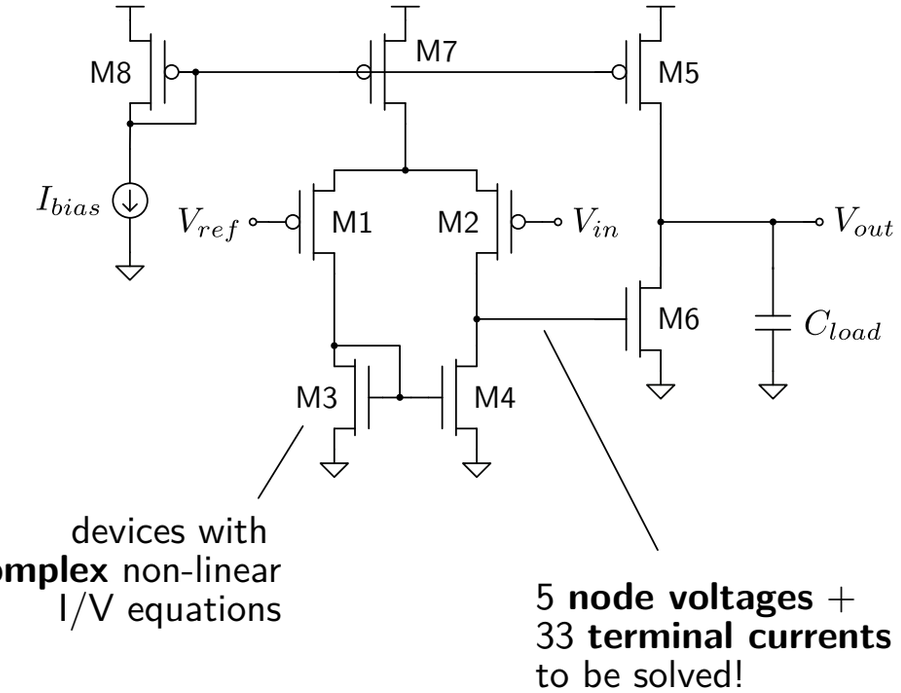
devices with **complex** non-linear I/V equations

5 **node voltages** + 33 **terminal currents** to be solved!

Electrical vs Behavioral Modeling

- Example: **voltage comparator**





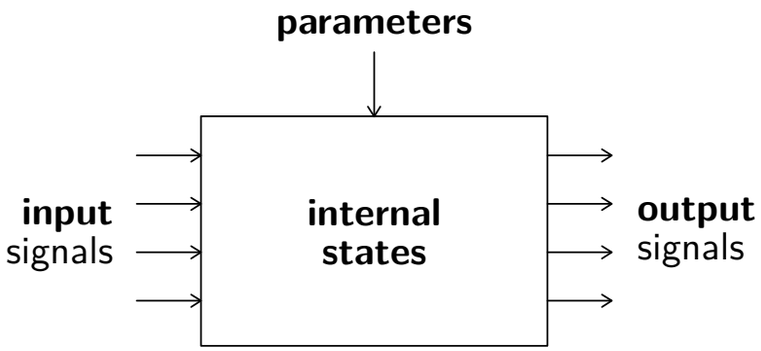
behavioral macro-model explicit equations:

$$d_{out} = \begin{cases} \text{H} & V_{in} \geq V_{ref} \\ \text{L} & V_{in} < V_{ref} \end{cases}$$

- Specific hardware description lenguage (**HDL**) required...

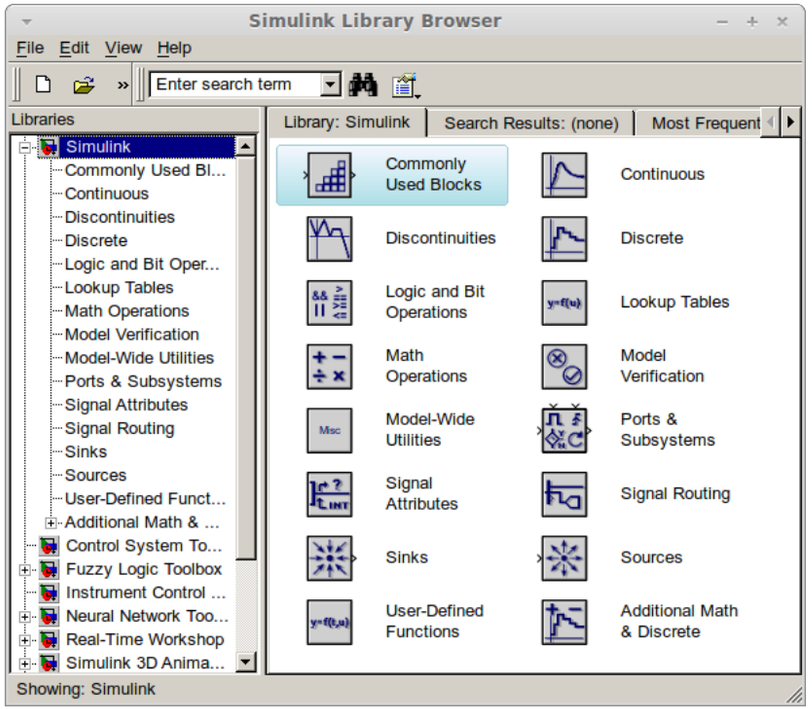
Simulink Engine

- Created in 1984
- **Multi** data type

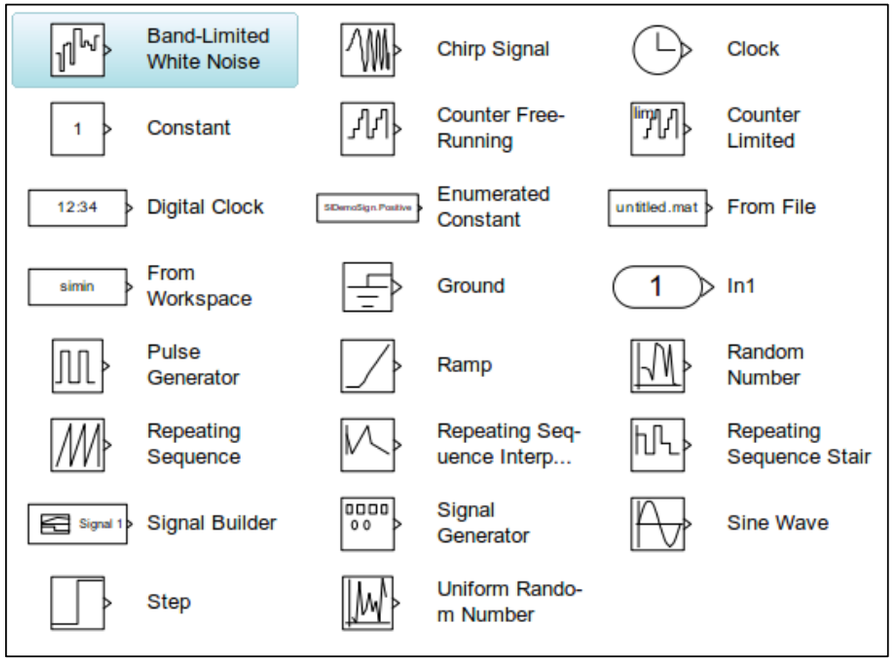


Simulink Engine

- Created in 1984
- **Multi** data type
- Several **libraries** of predefined blocks

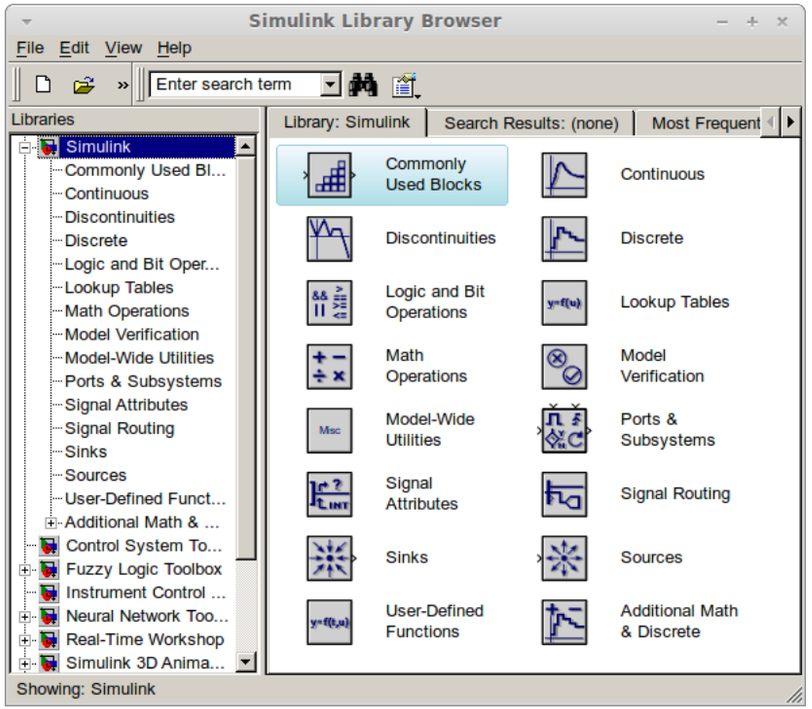


blocksets:

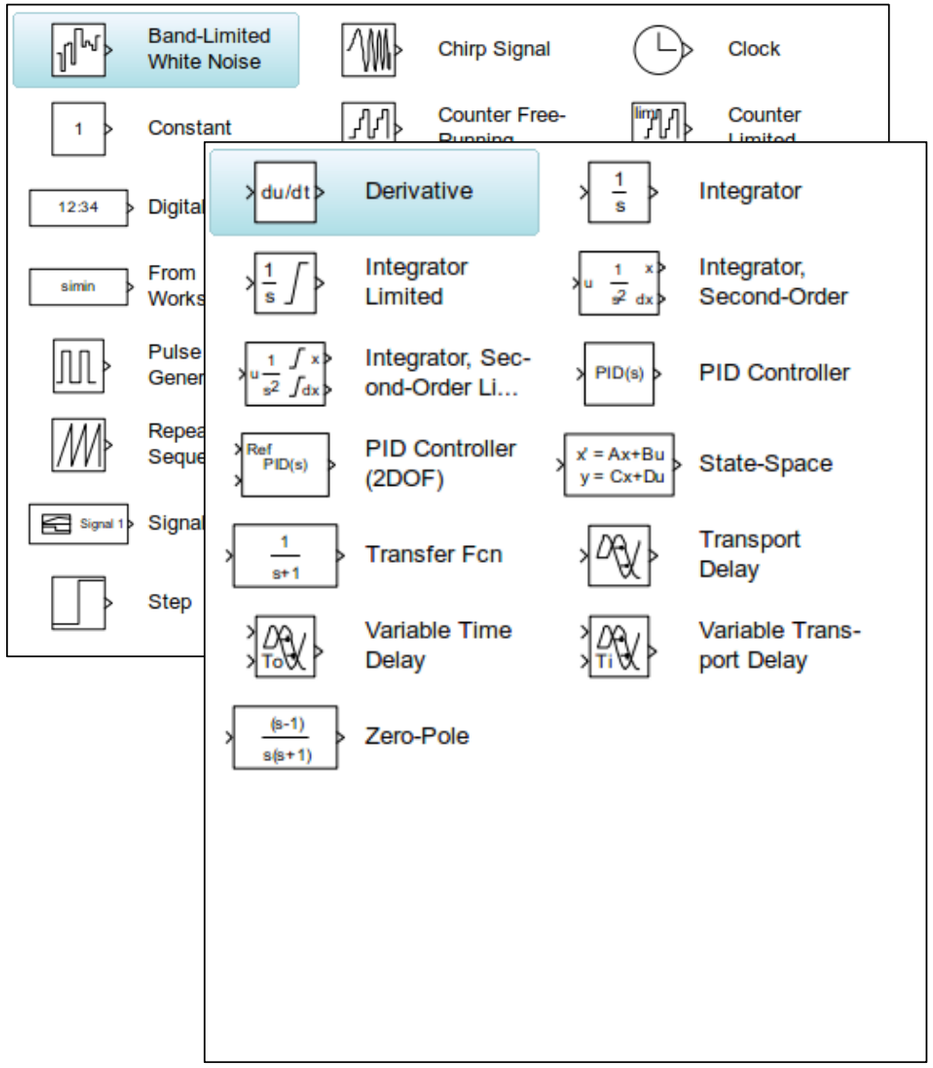


Simulink Engine

- Created in 1984
- **Multi** data type
- Several **libraries** of predefined blocks

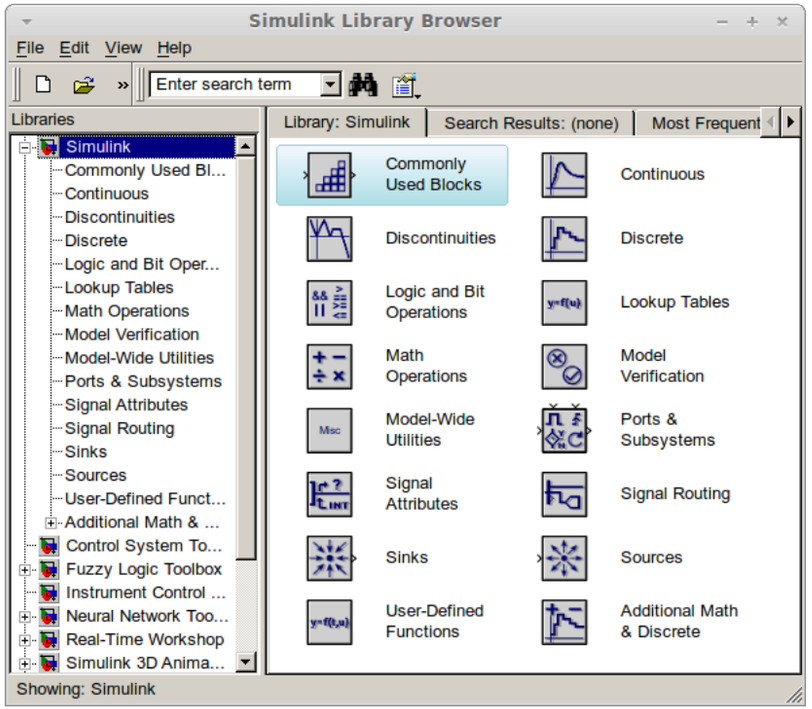


blocksets:

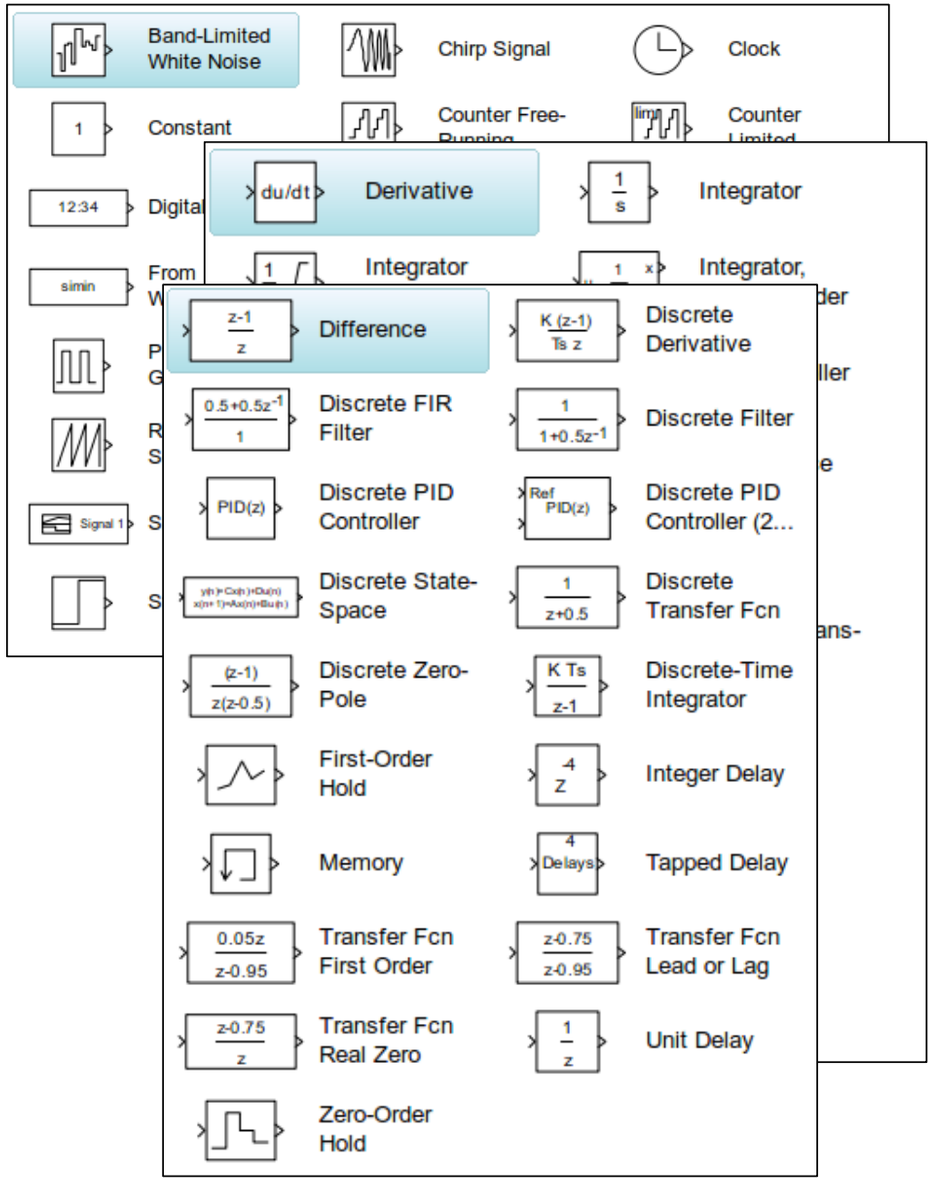


Simulink Engine

- Created in 1984
- **Multi** data type
- Several **libraries** of predefined blocks

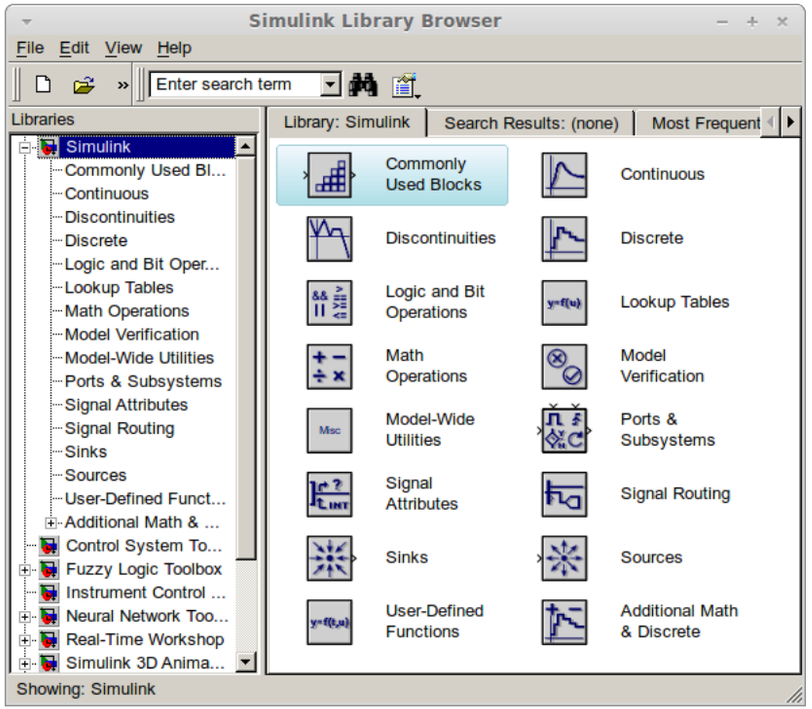


blocksets:

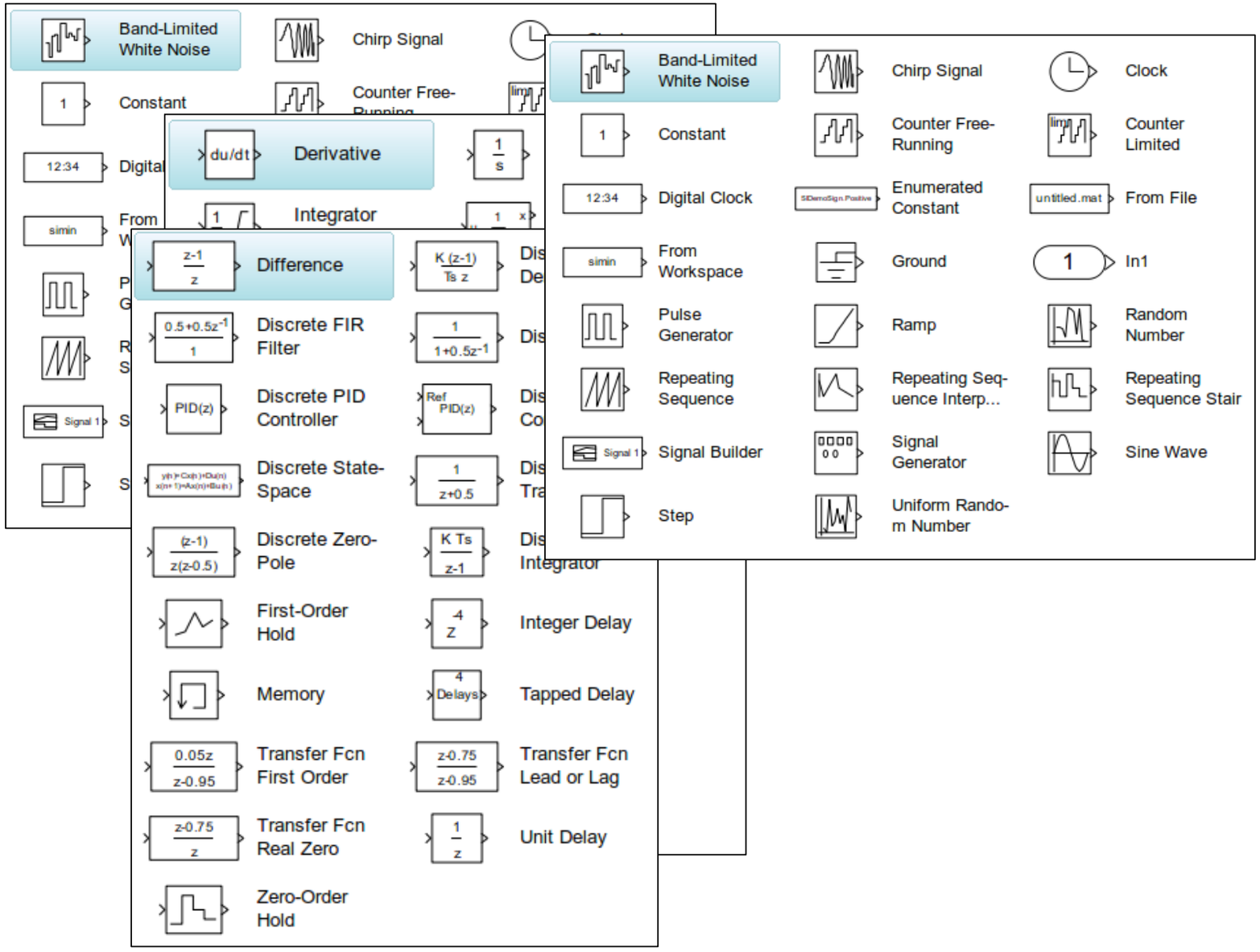


Simulink Engine

- Created in 1984
- **Multi** data type
- Several **libraries** of predefined blocks

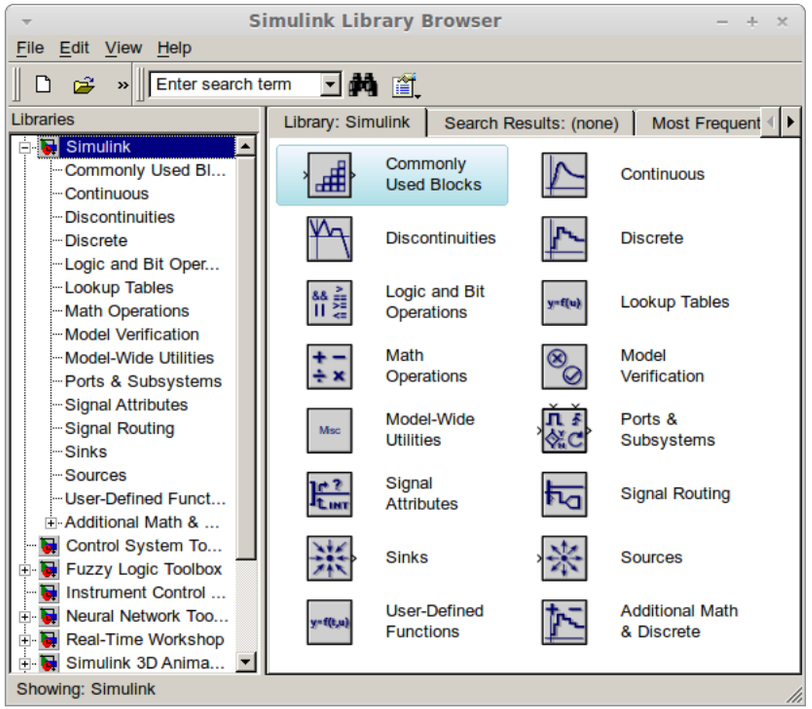


blocksets:

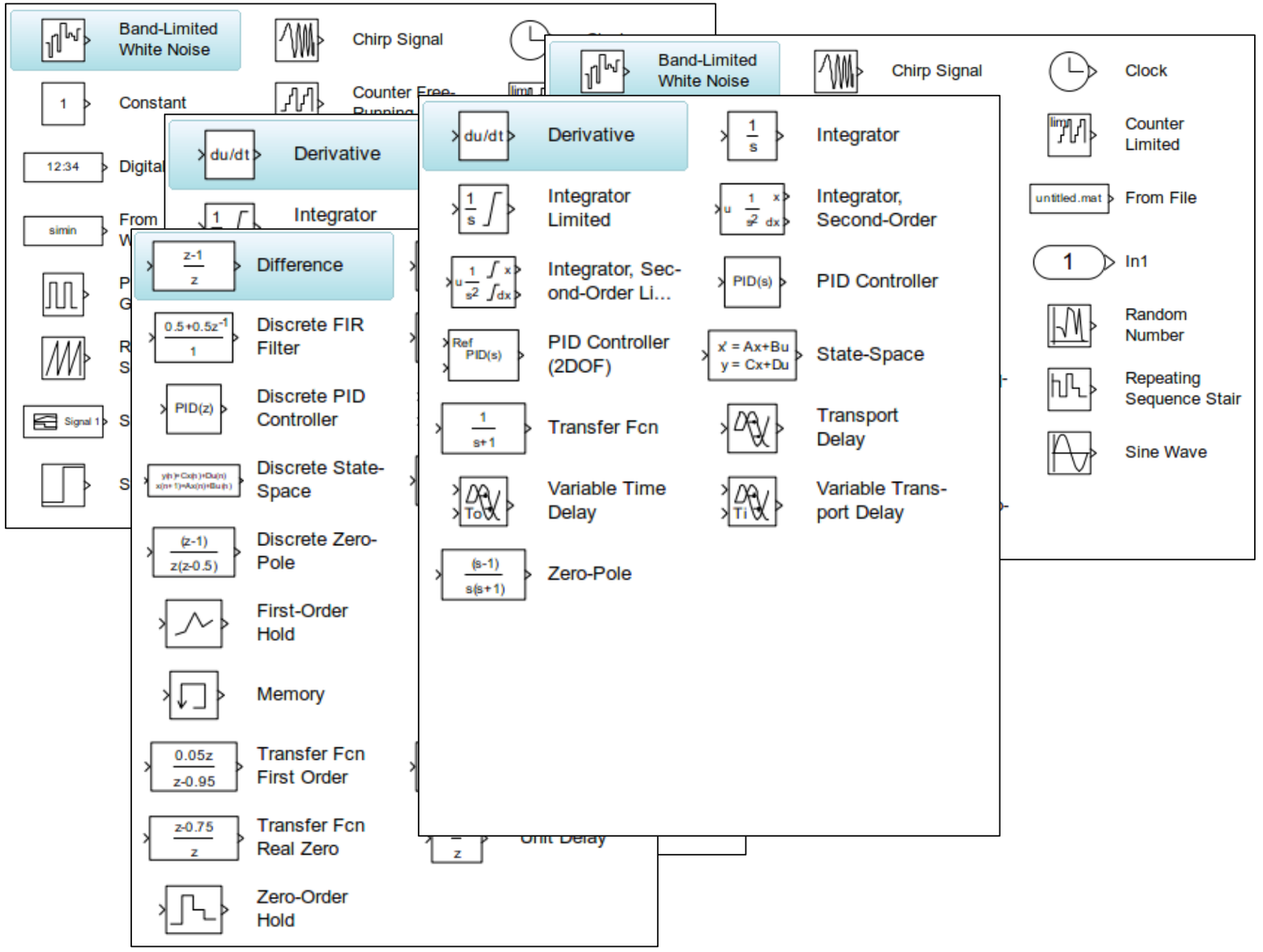


Simulink Engine

- Created in 1984
- **Multi** data type
- Several **libraries** of predefined blocks



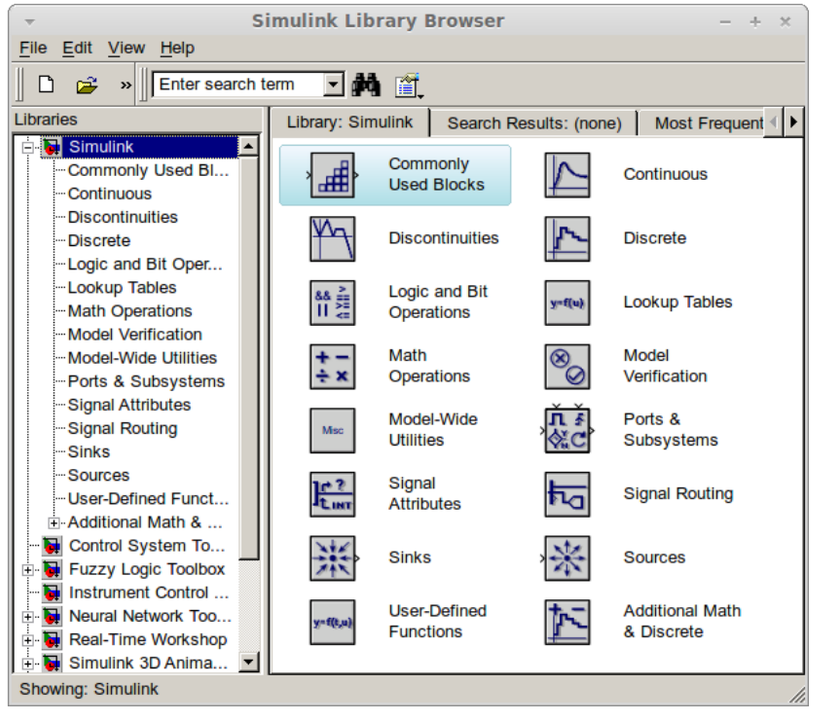
blocksets:

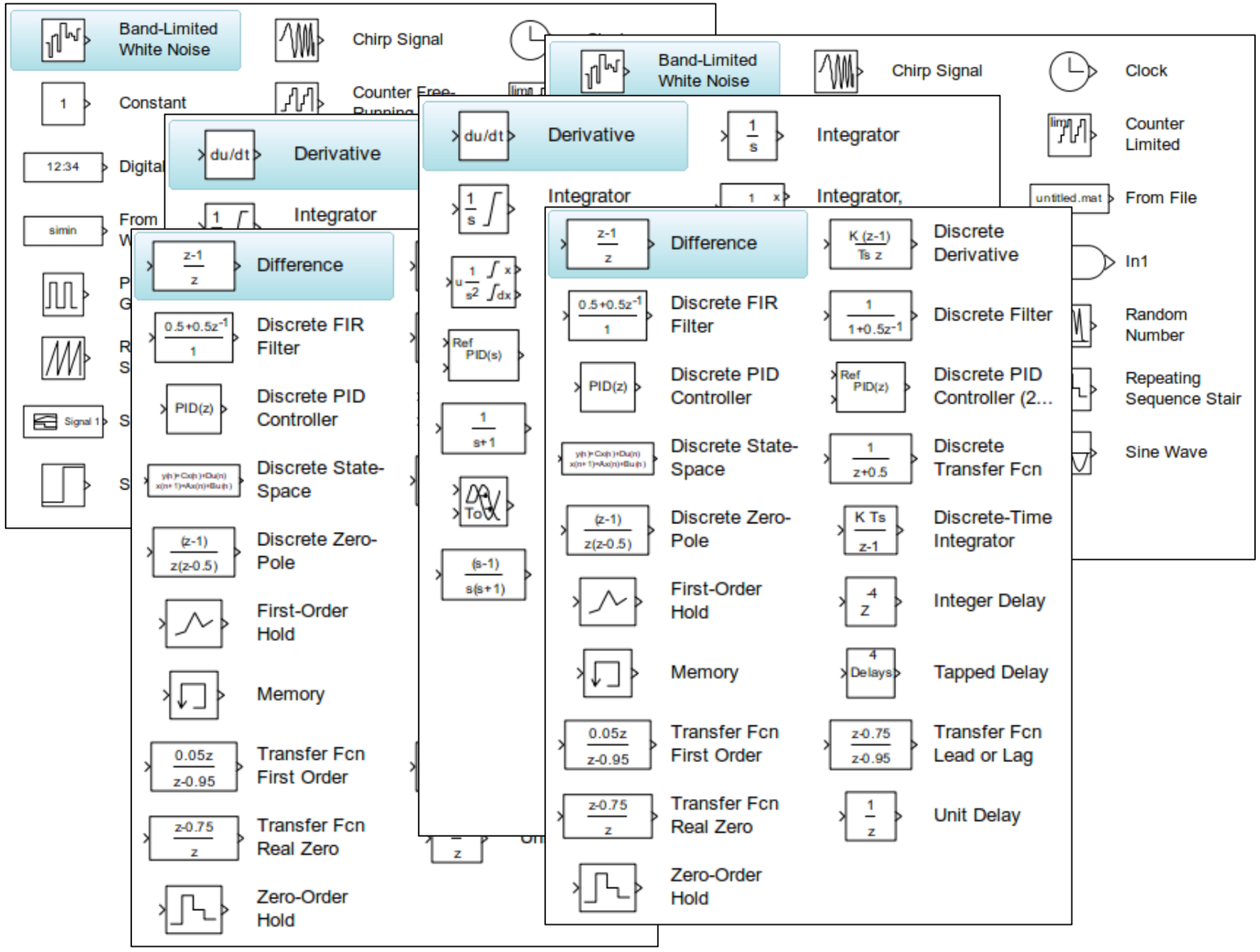


Simulink Engine

- Created in 1984
- **Multi** data type
- Several **libraries** of predefined blocks

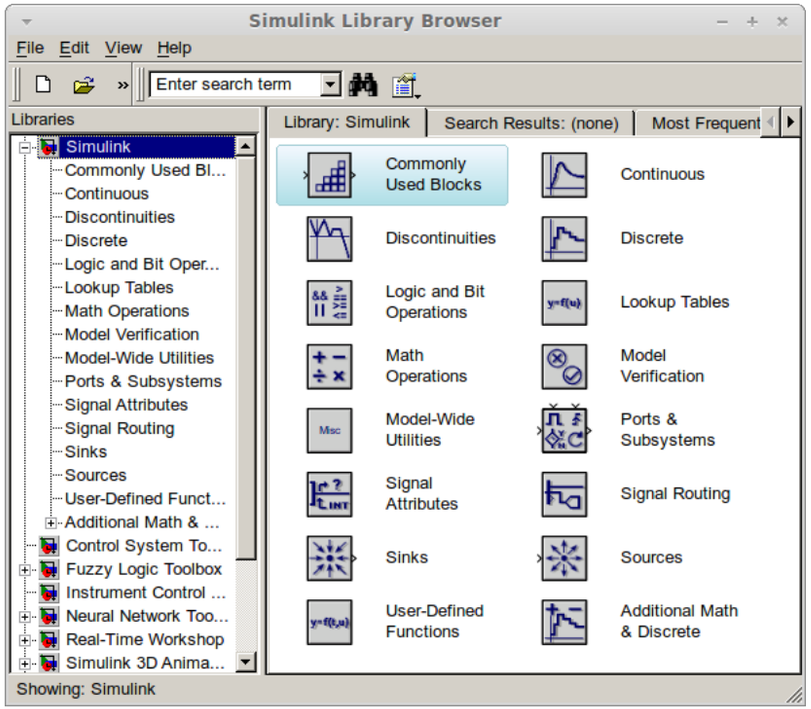
blocksets:



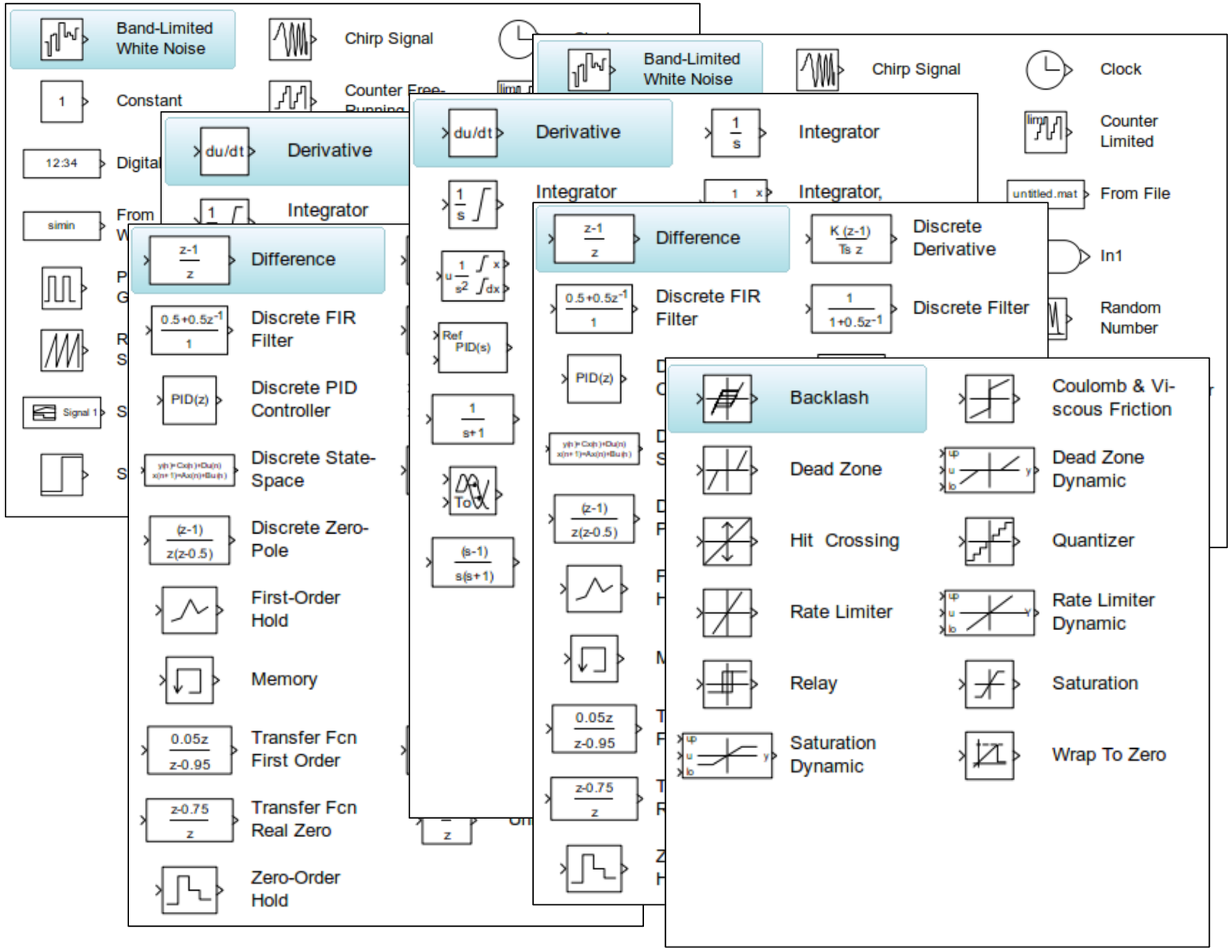


Simulink Engine

- Created in 1984
- **Multi** data type
- Several **libraries** of predefined blocks



blocksets:



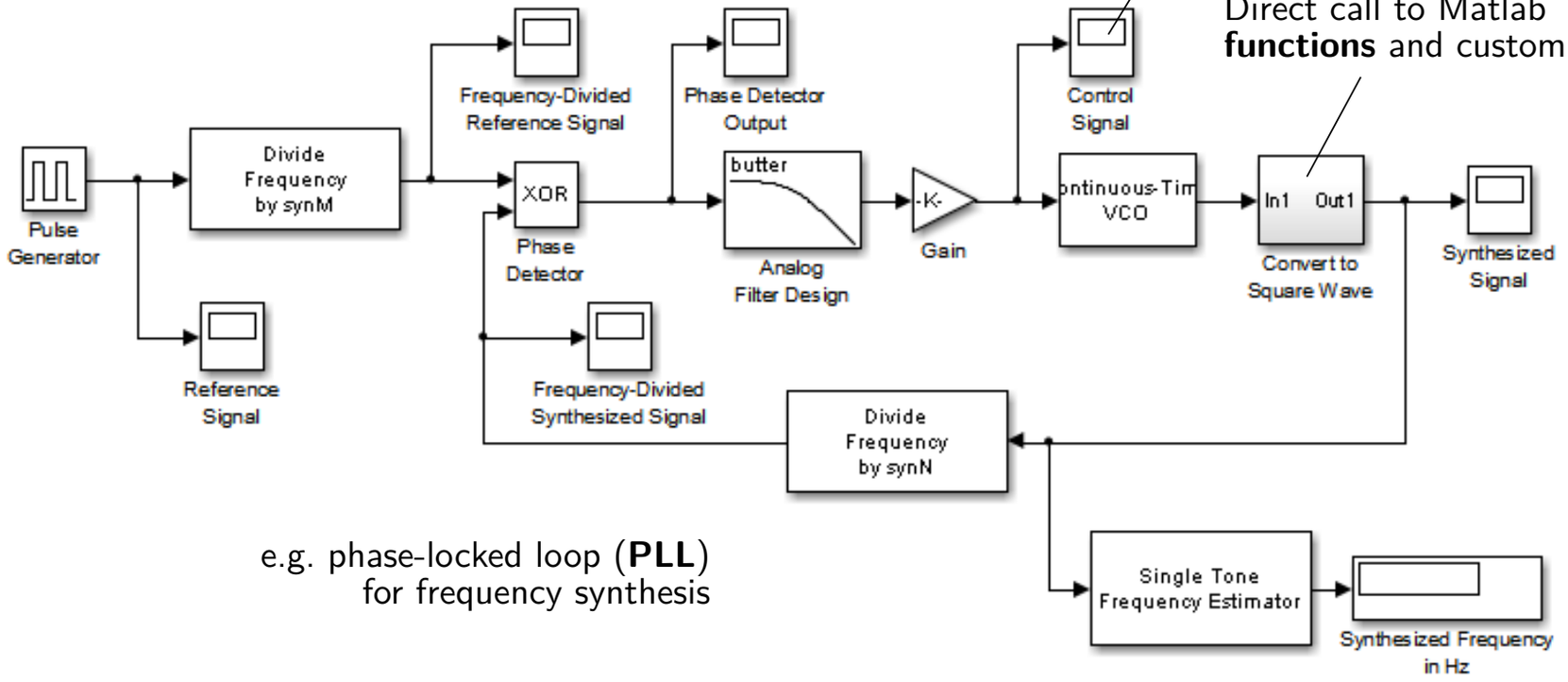
Simulink Engine

- Created in 1984
- **Multi** data type
- Several **libraries** of predefined blocks

- **Variable** and **fixed** step ordinary differential equation (**ODE**) solvers
- Continuous-time (**CT**) and discrete-time (**DT**) systems

Oscilloscopes and bridges to Matlab **workspace**

Direct call to Matlab **functions** and custom **scripts**

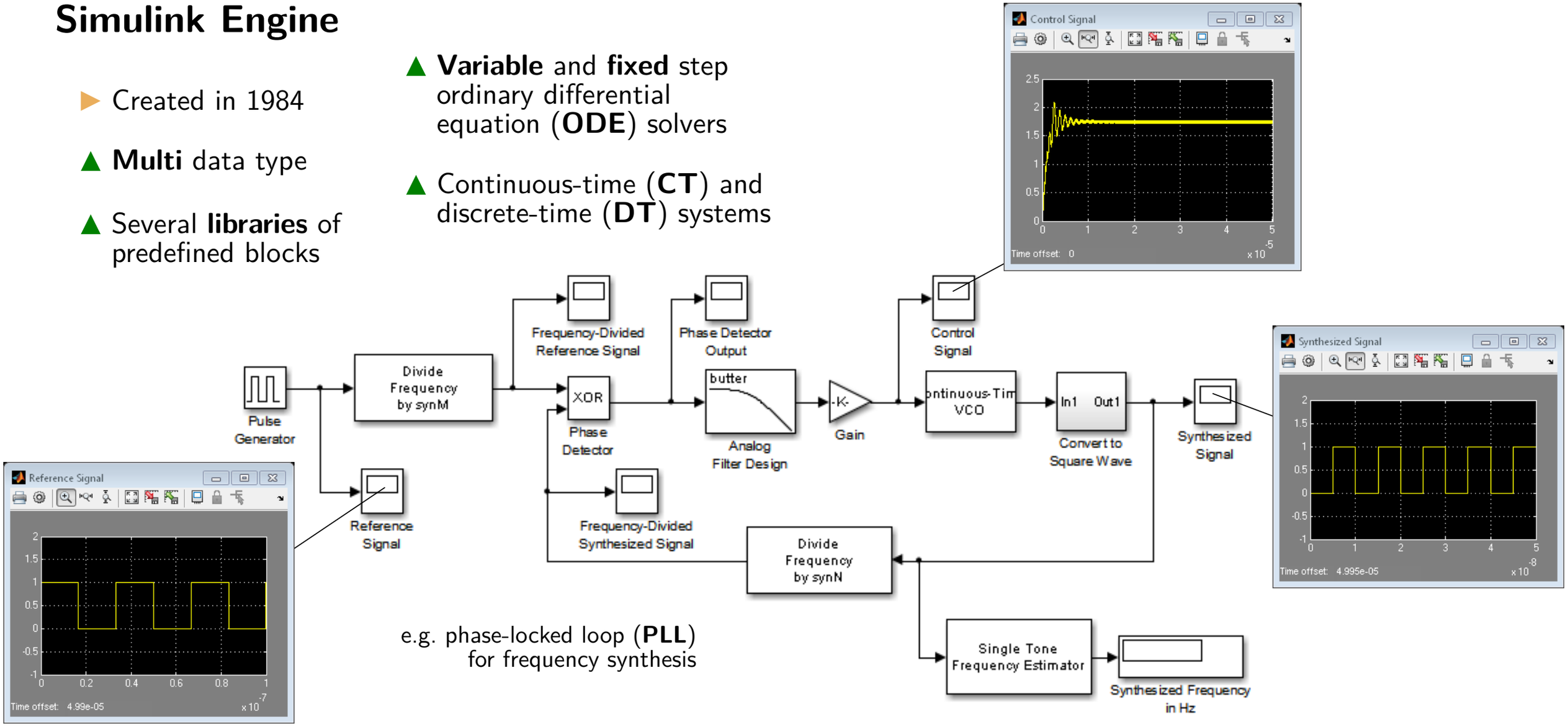


e.g. phase-locked loop (**PLL**) for frequency synthesis

Simulink Engine

- Created in 1984
- **Multi** data type
- Several **libraries** of predefined blocks

- **Variable** and **fixed** step ordinary differential equation (**ODE**) solvers
- Continuous-time (**CT**) and discrete-time (**DT**) systems



e.g. phase-locked loop (**PLL**) for frequency synthesis

Simulink Engine

- ▶ Created in 1984
- ▲ **Multi** data type
- ▲ Several **libraries** of predefined blocks
- ▲ **Variable** and **fixed** step ordinary differential equation (**ODE**) solvers
- ▲ Continuous-time (**CT**) and discrete-time (**DT**) systems
- ▲ Linked to **Matlab** scripting for the analysis of results
- ▼ Limited types of analysis

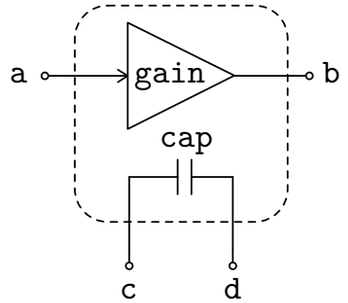
```
Block {
  BlockType        Reference
  Name         "Continuous-Time\nVCO"
  SID        3
  Ports        [1, 1]
  Position        [655, 357, 745, 403]
  BlockMirror       on
  NamePlacement       "alternate"
  LibraryVersion       "1.37"
  FontName        "Arial"
  SourceBlock       "commsynccomp2/Continuous-Time\nVCO"
  SourceType       "Continuous-Time VCO"
  ShowPortLabels       "FromPortIcon"
  SystemSampleTime       "-1"
  FunctionWithSeparateData off
  RTWMemSecFuncInitTerm   "Inherit from model"
  RTWMemSecFuncExecute    "Inherit from model"
  RTWMemSecDataConstants  "Inherit from model"
  RTWMemSecDataInternal   "Inherit from model"
  RTWMemSecDataParameters "Inherit from model"
  Ac        "1"
  Fc        "1e6"
  Kc        "1e5"
  Ph        "0"
}
```

↗ MathWorks
mathworks.com

Verilog Analog and Mixed Signal (AMS)

- Created in 1998
- Superset of Verilog digital HDL (**IEEE 1364**)
- Same as Verilog-A (1996), the AMS extension of Cadence Spectre
- **Analog/digital** circuits
- **Continuous/discrete** time
- All **SPICE analysis** supported (DC, AC, TRAN...)
- New **primitives** can be added in **C**

```
`include "constants.vams"
`include "disciplines.vams"

module example(a,b,c,d)

 parameter real cap = 1p;
 parameter integer gain = 2;

 input a;
 output b;
 inout c,d;

 electrical a,b,c,d;

 analog begin

  //Capacitor
  I(c,d) <+ cap * ddt(V(c,d));

  // Simple amplifier
  // Voltages referenced to ground if no second node is given
  V(b) <+ gain * V(a);

 end
endmodule
```

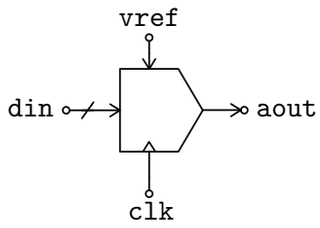


↗ Verilog-AMS Guide
verilogams.com

Verilog Analog and Mixed Signal (AMS)

- Created in 1998
- Superset of Verilog digital HDL (**IEEE 1364**)
- Same as Verilog-A (1996), the AMS extension of Cadence Spectre
- **Analog/digital** circuits
- **Continuous/discrete** time
- All **SPICE analysis** supported (DC, AC, TRAN...)
- New **primitives** can be added in **C**

```
`include "constants.vams"
`include "disciplines.vams"

module dac_simple(aout, clk, din, vref);

 parameter integer bits = 4 from [1:24];
 parameter integer td = 1n from[0:inf);

 input clk, vref;  // Define input/output
 input [bits-1:0] din;
 output aout;

 logic clk;  //Define port types
 logic [bits-1:0] din;
 electrical  aout, vref;

 real aout_new, ref;  // Internal variables
 integer i;

 analog begin
   ...
 end

endmodule
```

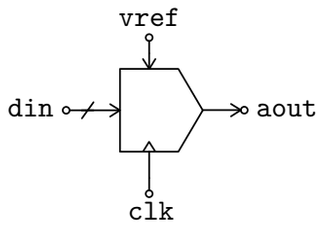


↗ Verilog-AMS Guide
verilogams.com

Verilog Analog and Mixed Signal (AMS)

- Created in 1998
- Superset of Verilog digital HDL (**IEEE 1364**)
- Same as Verilog-A (1996), the AMS extension of Cadence Spectre
- **Analog**/**digital** circuits
- **Continuous**/**discrete** time
- All **SPICE analysis** supported (DC, AC, TRAN...)
- New **primitives** can be added in **C**

```
`include "constants.vams"
`include "disciplines.vams"

module dac_simple(aout, clk, din, vref);

 ...

 analog begin
  @(initial_step) V(aout) <+ 0; // Initialization
  @(posedge clk) begin // Change only for rising clock edge
     aout_new = 0;
     ref = V(vref);
     for(i=0; i<bits; i=i+1) begin
        ref = ref/2;
        aout_new = aout_new + ref * din[i];
     end
  end
  V(aout) <+ transition(aout_new, td, 5n); // Smoothing
 end

endmodule
```

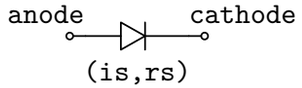


↗ Verilog-AMS Guide
verilogams.com

VHDL-AMS

- Created in 1999
- Incorporated into VHDL (**IEEE 1076**)
- **Similar** features to Verilog-AMS

```
library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;

-- this is the interface
entity DIODE is
   generic (is : current := 1.0e-14;  -- Saturation current
            rs : real    := 0.0);     -- Series resistance
   port (terminal anode, cathode : electrical);
end entity DIODE;
```

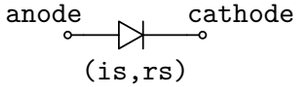


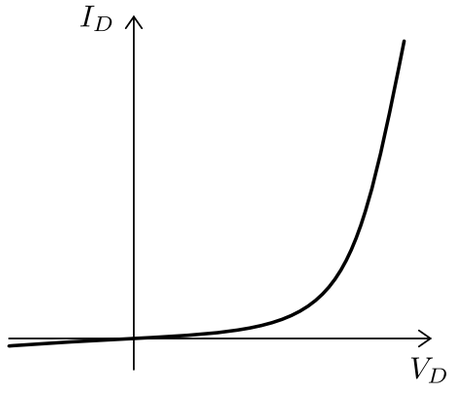
↗ IEEE 1076.1 Standard
standards.ieee.org

VHDL-AMS

- Created in 1999
- Incorporated into VHDL (**IEEE 1076**)
- **Similar** features to Verilog-AMS
- Splitting between interface (**entity**) and implementation (**architecture**)

```
library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;

-- this is the interface
entity DIODE is
   generic (is : current := 1.0e-14;  -- Saturation current
            rs : real    := 0.0);     -- Series resistance
   port (terminal anode, cathode : electrical);
end entity DIODE;

-- this is the implementation
architecture IV_CONTINUOUS of DIODE is
  quantity vd across id through anode to cathode;
  constant vt : voltage := 0.0258;     -- Thermal voltage at 300K
  begin

  id == is * (exp((vd-rs*id)/vt) - 1.0);

end architecture IV_CONTINUOUS;
```



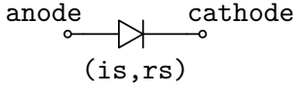


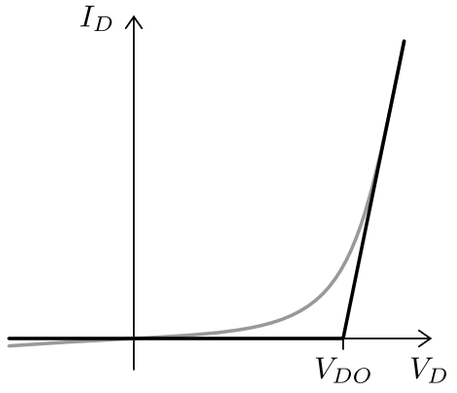
↗ IEEE 1076.1 Standard
standards.ieee.org

VHDL-AMS

- Created in 1999
- Incorporated into VHDL (**IEEE 1076**)
- **Similar** features to Verilog-AMS
- Splitting between interface (**entity**) and implementation (**architecture**)
- Same entity can hold **several** architectures...

```
library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;

-- this is the interface
entity DIODE is
   generic (is : current := 1.0e-14;  -- Saturation current
            rs : real    := 0.0);     -- Series resistance
   port (terminal anode, cathode : electrical);
end entity DIODE;

-- this is the implementation
architecture IV_PIECEWISE of DIODE is
  quantity vd across id through anode to cathode;
  variable vdo : real := 0.0;          -- Equivalent threshold
  begin

  vdo == ...;
  if (vd > vdo) use
    id == (vd - vdo) / rs;
  else
    id == 0;
  end use;

end architecture IV_PIECEWISE;
```



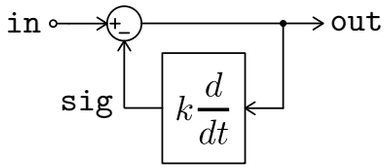


↗ IEEE 1076.1 Standard
standards.ieee.org

SystemC-AMS

- Created in 2010
- Incorporated into SystemC (**IEEE 1666**)
- Hierarchically, **on top** of Verliog-AMS and VHDL-AMS
- Based on **C++** class libraries:
  - Timed data flow (**TDF**)
  - Linear signal flow (**LSF**)
  - Electrical linear networks (**ELN**)
- **Concurrency** is optimized for simulation multithreading

```
SC_MODULE(mylsfmodel)   // model using LSF primitives
{
   sca_lsf::sca_in   in;   // LSF input port
   sca_lsf::sca_out   out; // LSF output port
   sca_lsf::sca_signal   sig; // LSF signal

   sca_lsf::sca_dot*   dot1; // declare module instances
   sca_lsf::sca_sub*   sub1;

   mylsfmodel(sc_module_name, double fc=1.0e3)
   {
       // instantiate predefined primitives
       dot1 = new sca_lsf::sca_dot("dot1",1.0/(2.0*M_PI*fc));
       dot1->x(out);
       dot1->y(sig);   // parameters
       sub1 = new sca_lsf::sca_sub("sub1");
       sub1->x1(in);
       sub1->x2(sig);
       sub1->y(out);
   }

};
```

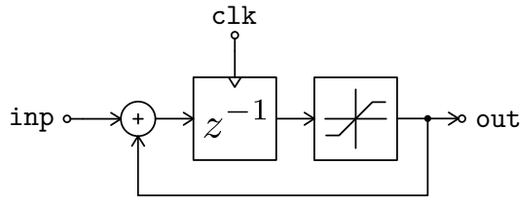


↗ SystemC-AMS Extensions
systemc.org

XSpice HDL

- Created in 1992
- Extension of SPICE (**a**-elements)
- **Analog**/**digital** circuits
- **Continuous**/**discrete** time
- New **primitives** can be added in **C** and compiled separately (**modular**):
  - Interface file specification (**IFS**)

```
NAME_TABLE:
Spice_Model_Name: zinteg2lim
C_Function_Name:  cm_zinteg2lim
Description:      "Z-domain integrator with limited output"

PORT_TABLE:
Port_Name:        inp        clk        out
Description:      "input"    "clock"    "output"
Direction:        in         in         out
Default_Type:     v          d          v
Allowed_Types:    [v]        [d]        [v]
Vector:           no         no         no
Vector_Bounds:    -          -          -
Null_Allowed:     no         no         no

PARAMETER_TABLE:
Parameter_Name:   pos_edge                    out_ic
Description:      "L->H edge output sync?"    "output initial condition"
Data_Type:        int                         real
Default_Value:    0                           0.0
Limits:           [0 1]                       -
Vector:           no                          no
Vector_Bounds:    -                           -
Null_Allowed:     no                          no

PARAMETER_TABLE:
Parameter_Name:   out_min                 out_max
Description:      "lower output limit"    "upper output limit"
Data_Type:        real                    real
Default_Value:    -1.0                    1.0
Limits:             -                       -
...
```

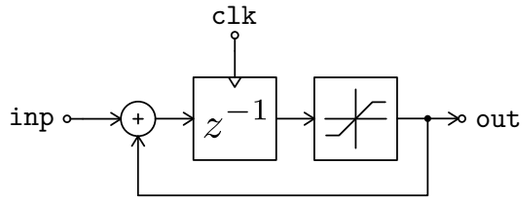


↗ F. L. Cox et al.
Code-level Modeling in XSPICE
IEEE ISCAS, May 1992
doi.org/10.1109/ISCAS.1992.230083

XSpice HDL

- Created in 1992
- Extension of SPICE (**a**-elements)
- **Analog**/**digital** circuits
- **Continuous**/**discrete** time
- New **primitives** can be added in **C** and compiled separately (**modular**):
  - Interface file specification (**IFS**)
  - Code model (**CM**)

```
#define SAMPLING_INTEGRATION 1
#define HOLDING 0

void cm_zinteg2lim(ARGS)
{
            double inp,      /* analog voltage input */
                   out,      /* analog voltage output */
                   *inp_mem, /* sampled input */
                   *out_mem, /* integrated output */
                   out_ic,   /* output initial condition */
                   out_min,  /* minimum output limit */
                   out_max;  /* maximum output limit */
  Digital_State_t clk,       /* current clock level */
                   *clk_mem, /* previous clock level*/
                   pos_edge; /* L->H edge clock output? */
               int action;   /* action type */
              char *error;   /* error message */

  inp = INPUT(inp);              /* Retriving input values */
  clk = INPUT_STATE(clk);
  pos_edge = PARAM(pos_edge);    /* Retriving parameters */
  out_ic = PARAM(out_ic);
  out_min = PARAM(out_min);
  out_max = PARAM(out_max);


  ...
```

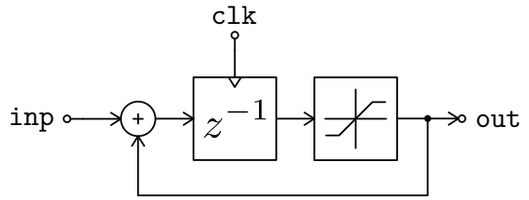


↗ F. L. Cox et al.
Code-level Modeling in XSPICE
IEEE ISCAS, May 1992
doi.org/10.1109/ISCAS.1992.230083

XSpice HDL

- Created in 1992
- Extension of SPICE (**a**-elements)
- **Analog**/**digital** circuits
- **Continuous**/**discrete** time
- New **primitives** can be added in **C** and compiled separately (**modular**):
  - Interface file specification (**IFS**)
  - Code model (**CM**)

```
...

  if (INIT==1) { /* Static storage allocation and checking */

    cm_analog_alloc(1,sizeof(double));
    cm_analog_alloc(2,sizeof(double));
    cm_event_alloc(3,sizeof(Digital_State_t));

    if (out_min>out_max) {
      error = "\n*** zinteg2lim error: out_min>out_max !\n";
      cm_message_send(error);
    }
    if ((out_ic>out_max)||(out_ic<out_min)) {
      error = "\n*** zinteg2lim error: out_ic exceeds
               [out_min,out_max] !\n";
      cm_message_send(error);
    }

  }

...
```

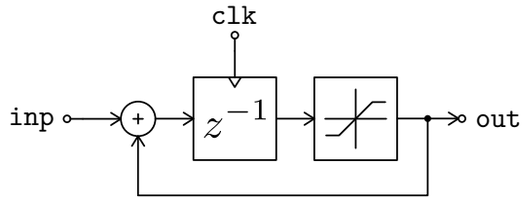


↗ F. L. Cox et al.
Code-level Modeling in XSPICE
IEEE ISCAS, May 1992
doi.org/10.1109/ISCAS.1992.230083

XSpice HDL

- Created in 1992
- Extension of SPICE (**a**-elements)
- **Analog**/**digital** circuits
- **Continuous**/**discrete** time
- New **primitives** can be added in **C** and compiled separately (**modular**):
  - Interface file specification (**IFS**)
  - Code model (**CM**)

```
...
  switch (ANALYSIS) {
    case TRANSIENT: /* Transient analysis */

      inp_mem = cm_analog_get_ptr(1,0); /* Previous state */
      out_mem = cm_analog_get_ptr(2,0);
      clk_mem = cm_event_get_ptr(3,0);

      if (TIME==0) { /* Initialization */

        *inp_mem = inp;
        *out_mem = out_ic;
        out = out_ic;

      } else { /* Regular operation */

        if ((*clk_mem==ONE)&&(clk==ZERO)) { /* Neg clk edge */
          if (pos_edge==FALSE)
            action = SAMPLING_INTEGRATION;
        } else {
          if ((*clk_mem==ZERO)&&(clk==ONE)) {/* Pos edge */
            if (pos_edge==TRUE)
              action = SAMPLING_INTEGRATION;
            } else {  /* No clock edge */
                action = HOLDING;
           }
          }
...
```

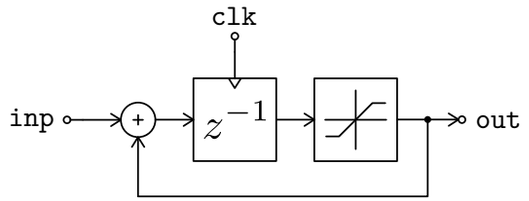


↗ F. L. Cox et al.
Code-level Modeling in XSPICE
IEEE ISCAS, May 1992
doi.org/10.1109/ISCAS.1992.230083

XSpice HDL

- Created in 1992
- Extension of SPICE (**a**-elements)
- **Analog**/**digital** circuits
- **Continuous**/**discrete** time
- New **primitives** can be added in **C** and compiled separately (**modular**):
  - Interface file specification (**IFS**)
  - Code model (**CM**)

```
...
        switch (action) {
           case SAMPLING_INTEGRATION: /* Samp and integ */
            *inp_mem = inp;
            out = *out_mem+*inp_mem;
            if (out<out_min) { out = out_min; } /* Limiter */
            if (out>out_max) { out = out_max; }
            *out_mem = out;
            break;
          case HOLDING:    /* Holding */
            out = *out_mem;
        }
      }

      *clk_mem = clk;
      OUTPUT(out) = out;

      break;

    case DC:         /* DC analysis */
      OUTPUT(out) = out_ic;
      break;

    default:         /* Analysis not supported */
      error = "\n*** zinteg2lim error: analysis not
                supported !\n";
      cm_message_send(error);
}}
```



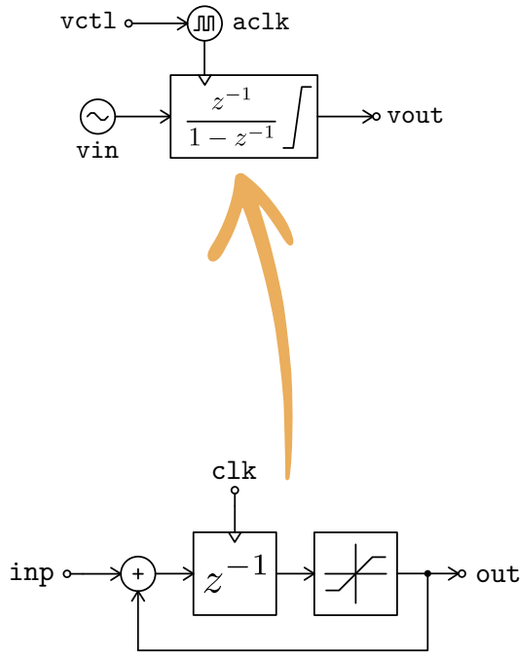
↗ F. L. Cox et al.
Code-level Modeling in XSPICE
IEEE ISCAS, May 1992
doi.org/10.1109/ISCAS.1992.230083

XSpice HDL

- Created in 1992
- Extension of SPICE (**a**-elements)
- **Analog**/**digital** circuits
- **Continuous**/**discrete** time
- New **primitives** can be added in **C** and compiled separately (**modular**):
  - Interface file specification (**IFS**)
  - Code model (**CM**)
- XSpice + SPICE3 co-simulation

```
vin 1 0 sin(0 1 1e4 0 0 90)
vctl 2 0 dc=1
aclk %v(2) %d(3) myclock
.model myclock d_osc(cntl_array=[0 1] freq_array=[0 1e6])

azinteg2lim %v(1) %d(3) %v(5) myzinteg2lim
.model myzinteg2lim zinteg2lim(pos_edge=0 out_ic=0.0
out_min=-1.0 out_max=1.0)

aprobe [%d(3)] [%v(4)] myprobe
.model myprobe dac_bridge(out_low=0 out_high=1 t_rise=1e-9
t_fall=1e-9)
.end
```

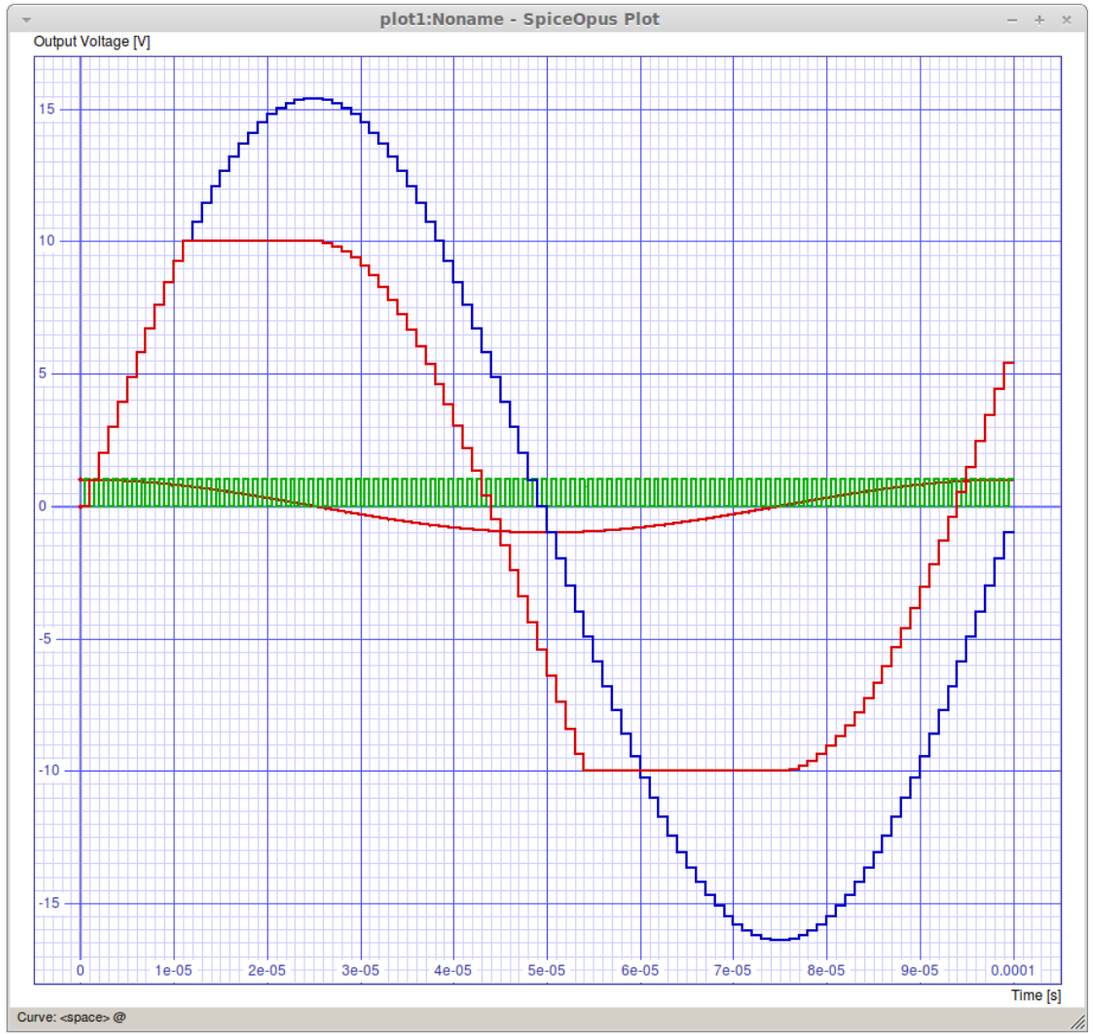
```
.control

source test_zinteg2lim.cir
let @@myzinteg2lim[out_min]=-10
let @@myzinteg2lim[out_max]=10
tran 1e-9 1e-4
let vin=v(1)
let vclk=v(4)
let vout=v(5)
plot create plot1 vin vclk vout vs time xlabel 'Time [s]'
ylabel 'Output Voltage [V]' xlimit 0 1e-4 ylimit -20 20
...

.endc
.end
```

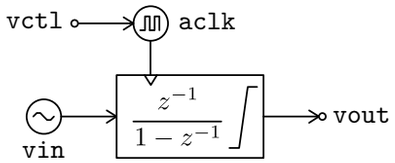


↗ F. L. Cox et al.
Code-level Modeling in XSPICE
IEEE ISCAS, May 1992
doi.org/10.1109/ISCAS.1992.230083

XSpice HDL

- Created in 1992
- Extension of SPICE (**a**-elements)
- **Analog**/**digital** circuits
- **Continuous**/**discrete** time
- New **primitives** can be added in **C** and compiled separately (**modular**):
  - Interface file specification (**IFS**)
  - Code model (**CM**)
- XSpice + SPICE3 co-simulation





↗ F. L. Cox et al.
Code-level Modeling in XSPICE
IEEE ISCAS, May 1992
doi.org/10.1109/ISCAS.1992.230083

XSpice HDL

- Macro definitions for **circuit data**:

Name	Description	Example
`ARGS`	Passing arguments to the CM.	`void dsm_opamp(ARGS)`
`CALL_TYPE`	Returns the simulator type used for the CM (`EVENT` or `ANALOG`).	`if (CALL_TYPE==ANALOG) {...}`
`INIT`	Returns 1 when first call of the CM.	`if (INIT==1) {...}`
`ANALYSIS`	Returns the current analysis type (`AC`, `DC` or `TRANSIENT`).	`if (ANALYSIS!=AC) {...}`
`FIRST_TIMEPOINT`	Returns 1 when first call of CM during the current analysis step.	`if (FIRST_TIMEPOINT==0) {...}`
`TIME`	Double returning current time point of TRANSIENT analysis in s.	`t2 = TIME-t1;`
`T(n)`	Double vector returning [current previous] time points of TRANSIENT analysis.	`dt = T(0)-T(1);`
`RAD_FREQ`	Double returning current frequency of AC analysis in rad/s.	`f = RAD_FREQ/(2*pi);`
`TEMPERATURE`	Double vector returning current analysis temperature in °C.	`TK = TEMPERATURE+273;`

↗ F. L. Cox et al.
Code-level Modeling in XSPICE
IEEE ISCAS, May 1992
doi.org/10.1109/ISCAS.1992.230083

XSpice HDL

- Macro definitions for **parameter** and **port data**:

Name	Description	Example
`PARAM(param)`	Returns CM parameter value.	`k = PARAM(gain);`
`PARAM_SIZE(param)`	Returns CM parameter vector size.	`num_coeff = PARAM_SIZE(coeff);`
`PARAM_NULL(param)`	Returns 1 when no value specified.	`if (PARAM_NULL(gain)==1) {...}`
`PORT_SIZE(a)`	Returns port size.	`num_out = PORT_SIZE(out);`
`PORT_NULL(a)`	Returns 1 when port is not connected.	`if (PORT_NULL(inp)==1) {...}`
`LOAD(a)`	Adds load capacitance in F to digital port.	`LOAD(inp) = 1e-12;`
`TOTAL_LOAD(a)`	Reads total load capacitance in F at digital port due to all attached CMs.	`delay = TOTAL_LOAD(out)*...`

- Macro definitions for **partial derivatives**, **gains** and **static variables**:

Name	Description	Example
`PARTIAL(y,x)`	Sets the partial derivative of output port `y` with respect to input port `x`. Needed by the simulator to solve non-linear equations. The `cm_analog_auto_partial()` function of Table 10 may be used instead.	`PARTIAL(out,in) = 1;`
`AC_GAIN(y,x)`	Sets the gain from input port `x` to output port `y` in `AC` analysis. Gain follows the complex data structure `Complex_t` def ned in Table 11.	`AC_GAIN(out,in) = gain_complex;`
`STATIC_VAR(a)`	Provides access to the static variables defined in the IFS file.	`last_x = STATIC_VAR(x);` `STATIC_VAR(x) = x;`

↗ F. L. Cox et al.
Code-level Modeling in XSPICE
IEEE ISCAS, May 1992
doi.org/10.1109/ISCAS.1992.230083

XSpice HDL

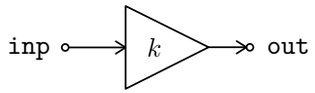
- Macro definitions for **I/O data**:

Name	Description	Example
`INPUT(x)`	Reads value from input port.	`signal = INPUT(inp);`
`INPUT_STATE(x)`	Reads the state of a digital input port (`ZERO`, `ONE` or `UNKNOWN`).	`if (INPUT_STATE(inp)!=ONE) {...}`
`INPUT_STRENGTH(x)`	Reads the strength with which a digital input port is externally driven (`STRONG`, `RESISTIVE`, `HI_IMPEDANCE` or `UNDETERMINATED`).	`if (INPUT_STRENGTH(inp)!= HI_IMPEDANCE) {...}`
`OUTPUT(y)`	Writes value to output port.	`OUTPUT(out) = result;`
`OUTPUT_CHANGED(y)`	Flags a digital output port as modified. If `TRUE` (default) then state, strength and delay need to be defined.	`OUTPUT_CHANGED(out) = FALSE;`
`OUTPUT_DELAY(y)`	Defines the delay in s (>0) of a digital output port.	`OUTPUT_DELAY(out) = 1e-9;`
`OUTPUT_STATE(y)`	Writes the state of a digital output port (`ZERO`, `ONE` or `UNKNOWN`).	`OUTPUT_STATE(out) = ZERO;`
`OUTPUT_STRENGTH(y)`	Writes the strength with which a digital output port is internally driven (`STRONG`, `RESISTIVE`, `HI_IMPEDANCE` or `UNDETERMINATED`).	`OUTPUT_STRENGTH(out) == STRONG;`

↗ F. L. Cox et al.
Code-level Modeling in XSPICE
IEEE ISCAS, May 1992
doi.org/10.1109/ISCAS.1992.230083

XSpice Examples

- Simple **gain** block:

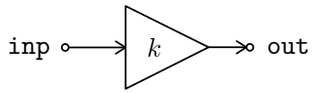


  - Interface file specification

```
NAME_TABLE:

Spice_Model_Name: kgain
C_Function_Name:  cm_kgain
Description:      "Scalar gain"

PORT_TABLE:

Port_Name:        inp       out
Description:      "input"   "output"
Direction:        in        out
Default_Type:     v         v
Allowed_Types:    [v]       [v]
Vector:           no        no
Vector_Bounds:    -         -
Null_Allowed:     no        no

PARAMETER_TABLE:

Parameter_Name:   k
Description:      "gain factor"
Data_Type:        real
Default_Value:    1.0
Limits:           -
```

XSpice Examples

- Simple **gain** block:

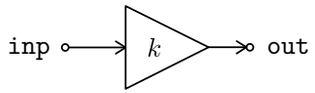


  - Interface file specification
  - Code model

```
void cm_kgain(ARGS)
{
            double inp,  /* analog voltage input */
                   out,  /* analog voltage output */
                   k;    /* gain factor */
          Complex_t k_ac; /* AC gain factor */

  inp = INPUT(inp);      /* Retriving input values */
  k = PARAM(k);          /* Retrieving parameters */

  if (ANALYSIS!=AC) {    /* DC and TRANSIENT analysis */
    OUTPUT(out) = k*inp;
    PARTIAL(out,inp) = k;
  } else {               /* AC analysis */
    k_ac.real = k;
    k_ac.imag = 0.0;
    AC_GAIN(out,inp) = k_ac;
  }

}
```

XSpice Examples

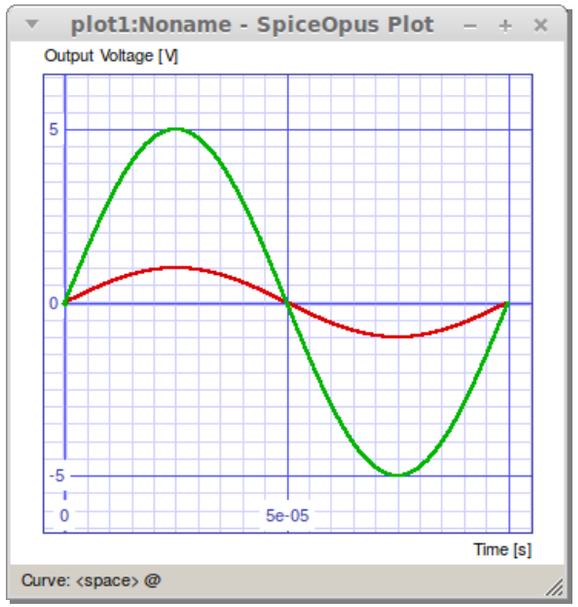
- Simple **gain** block:



  - Interface file specification
  - Code model
  - Simulation example

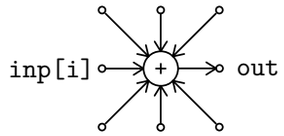
```
test_kgain.sp3

* Test script for kgain CM
* Requires cmload deltasigma.cm


.control

delcirc all
destroy all
delete all
save all

source test_kgain.cir
let @@mykgain[k]=5
tran 1e-9 1e-4
let vin=v(1)
let vout=v(2)
plot create plot1 vin vout vs time xlabel 'Time [s]'
ylabel 'Output Voltage [V]' xlimit 0 1e-4 ylimit -6 6

.endc

.end
```

```
test_kgain.cir
* Test circuit for kgain CM

vin 1 0 sin(0 1 1e4)

akgain %v(1) %v(2) mykgain
.model mykgain kgain(k=10)

.end
```



XSpice Examples

- Unity-gain **summer/substractor**:

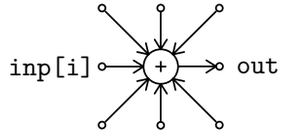


  - Interface file specification

```
NAME_TABLE:

Spice_Model_Name: usummer
C_Function_Name:  cm_usummer
Description:      "Unity-gain summer/substractor"

PORT_TABLE:

Port_Name:        inp           out
Description:      "input array" "output"
Direction:        in            out
Default_Type:     v             v
Allowed_Types:    [v]           [v]
Vector:           yes           no
Vector_Bounds:    [2 -]         -
Null_Allowed:     no            no

PARAMETER_TABLE:

Parameter_Name:   sign
Description:      "input sign array {-1,1}"
Data_Type:        int
Default_Value:    1
Limits:           [-1 1]
```

XSpice Examples

- Unity-gain **summer/substractor**:

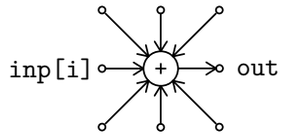


  - Interface file specification
  - Code model

```
void cm_usummer(ARGS)
{
          double acc;  /* summation accumulator */
       Complex_t k_ac; /* equivalent AC gain factor */
            int  ninp, /* number of inputs */
                 i;    /* generic loop counter */

  ninp = PORT_SIZE(inp);

  if (ANALYSIS!=AC) {  /* DC and TRANSIENT analysis */
    acc = 0.0;
    for (i=0; i<ninp; i++) {
      if (PARAM(sign[i])<0) {
        acc = acc-INPUT(inp[i]);
        PARTIAL(out,inp[i]) = -1.0;
      } else {
        acc = acc+INPUT(inp[i]);
        PARTIAL(out,inp[i]) = 1.0;
      }
    }
    OUTPUT(out) = acc;
  } else {             /* AC analysis */
    for (i=0; i<ninp; i++) {
      if (PARAM(sign[i])<0) {
        k_ac.real = -1;
      } else {
        k_ac.real = 1;
      }
      k_ac.imag = 0.0;
      AC_GAIN(out,inp[i]) = k_ac;
    }
  }
}
```

XSpice Examples

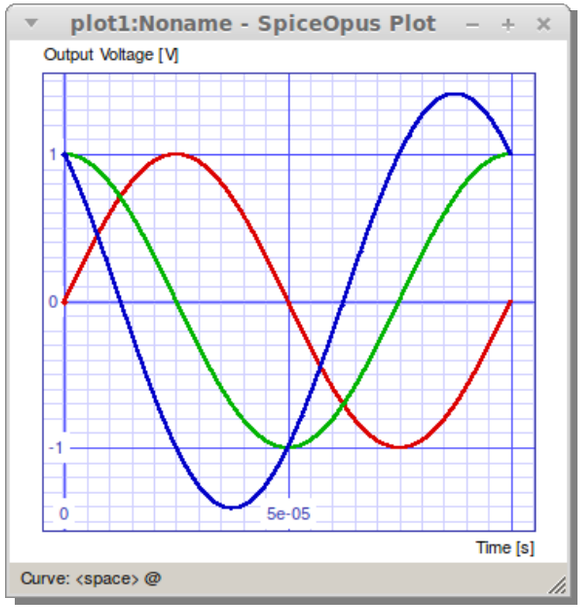
- Unity-gain **summer/substractor**:



  - Interface file specification
  - Code model
  - Simulation example

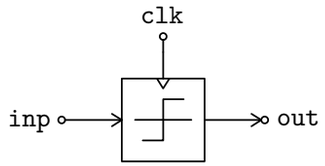
```
test_usummer.sp3
* Test script for usummer CM
* Requires cmload deltasigma.cm

.control

delcirc all
destroy all
delete all
save all

source test_usummer.cir
let @@myusummer[sign]=(-1;1)
tran 1e-9 1e-4
let vin1=v(1)
let vin2=v(2)
let vout=v(3)
plot create plot1 vin1 vin2 vout vs time xlabel 'Time
[s]' ylabel 'Output Voltage [V]'

.endc

.end
```

```
test_usummer.cir
* Test circuit for usummer CM

vin1 1 0 sin(0 1 1e4)
vin2 2 0 sin(0 1 1e4 0 0 90)

ausummer [%v(1) %v(2)] %v(3) myusummer
.model myusummer usummer(sign=[1 -1])

.end
```



XSpice Examples

- 2-level **quantizer** with S/H:

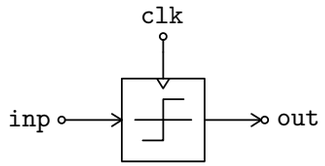


  - Interface file specification

```
NAME_TABLE:

Spice_Model_Name: quant2lsh
C_Function_Name:  cm_quant2lsh
Description:      "2-level quantizer with S/H"

PORT_TABLE:

Port_Name:        inp       clk       out
Description:      "input"   "clock"   "output"
Direction:        in        in        out
Default_Type:     v         d         d
Allowed_Types:    [v]       [d]       [d]
Vector:           no        no        no
Vector_Bounds:    -         -         -
Null_Allowed:     no        no        no

PARAMETER_TABLE:

Parameter_Name:     inp_th             out_ic
Description:        "input threshold"  "output initial condition"
Data_Type:          real               int
Default_Value:      0.0                0
Limits:             -                  [0 1]
Vector:             no                 no
Vector_Bounds:      -                  -
Null_Allowed:       no                 no

PARAMETER_TABLE:

Parameter_Name:     pos_edge          t_rise        t_fall
Description:        "L->H edge out?"  "rise delay"  "fall delay"
Data_Type:          int               real          real
Default_Value:      0                 1.0e-9        1.0e-9
Limits:             [0 1]             [1e-12 -]     [1e-12 -]
Vector:             no                no            no
```

XSpice Examples

- 2-level **quantizer** with S/H:

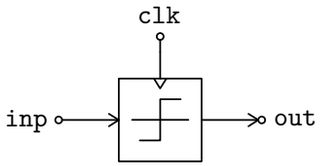


  - Interface file specification
  - Code model

```
#define SAMPLING_QUANTIZATION 1
#define HOLDING 0

void cm_quant2lsh(ARGS)
{           double inp,      /* analog voltage input */
                   *inp_mem, /* sampled input */
                   inp_th,   /* input threshold */
                   t_rise,   /* output rise time */
                   t_fall;   /* output fall time */

  Digital_State_t out,      /* digital output */
                   *out_mem, /* holded output */
                   clk,      /* current clock level */
                   *clk_mem, /* previous clock level*/
                   out_ic,   /* output initial condition */
                   pos_edge; /* L->H edge clock output? */
              int action;   /* action type */
             char *error;   /* error message */

  inp = INPUT(inp);            /* Retriving input values */
  clk = INPUT_STATE(clk);
  inp_th = PARAM(inp_th);      /* Retrieving parameters */
  t_rise = PARAM(t_rise);
  t_fall = PARAM(t_fall);
  out_ic = PARAM(out_ic);
  pos_edge = PARAM(pos_edge);

  if (INIT==1) { /* Static storage allocation and checking */
    cm_analog_alloc(1,sizeof(double));
    cm_event_alloc(2,sizeof(Digital_State_t));
    cm_event_alloc(3,sizeof(Digital_State_t));
    if (t_rise<1e-12) {
      error = "\n*** quant2lsh error: t_rise<1ps !\n";
      cm_message_send(error);
    }
    if (t_fall<1e-12) {
      error = "\n*** quant2lsh error: t_fall<1ps !\n";
      cm_message_send(error);
    }}
```

XSpice Examples

- 2-level **quantizer** with S/H:

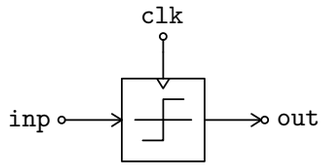


  - Interface file specification
  - Code model

```
switch (ANALYSIS) {

  case TRANSIENT: /* Transient analysis */

    inp_mem = cm_analog_get_ptr(1,0); /* Retriving previous state */
    out_mem = cm_event_get_ptr(2,0);
    clk_mem = cm_event_get_ptr(3,0);

    if (TIME==0) { /* Initialization */

      *inp_mem = inp;
      *out_mem = out_ic;
      out = out_ic;
      OUTPUT_CHANGED(out) = TRUE;
      OUTPUT_STATE(out) = out;
      OUTPUT_STRENGTH(out) = STRONG;

    } else { /* Regular operation */

      if ((*clk_mem==ONE)&&(clk==ZERO)) { /* Negative clk edge */
        if (pos_edge==FALSE)
        {
          action = SAMPLING_QUANTIZATION;
        }
      } else {
        if ((*clk_mem==ZERO)&&(clk==ONE)) { /* Positive clk edge */
          if (pos_edge==TRUE)
          {
            action = SAMPLING_QUANTIZATION;
          }
        } else {  /* No clock edge */
            action = HOLDING;
        }
      }
```

XSpice Examples

- 2-level **quantizer** with S/H:

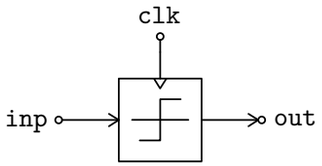


  - Interface file specification
  - Code model

```
        switch (action) {
          case SAMPLING_QUANTIZATION: /* Quantization action */
            OUTPUT_CHANGED(out) = TRUE;
            *inp_mem = inp;
            if (*inp_mem>inp_th) {
              out = ONE;
              OUTPUT_DELAY(out) = t_rise;
            } else {
              out = ZERO;
              OUTPUT_DELAY(out) = t_fall;
            }
            OUTPUT_STATE(out) = out;
            OUTPUT_STRENGTH(out) = STRONG;
            *out_mem = out;
            break;
          case HOLDING:             /* Holding action */
            OUTPUT_CHANGED(out) = FALSE;
        }
      }

      *clk_mem = clk;

      break;

    case DC:        /* DC analysis */
      OUTPUT_STATE(out) = out_ic;
      OUTPUT_STRENGTH(out) = STRONG;
      break;

    default:        /* Analysis not supported */
      error = "\n*** quant2lsh error: analysis not supported !\n";
      cm_message_send(error);
  }
}
```

XSpice Examples

- 2-level **quantizer** with S/H:



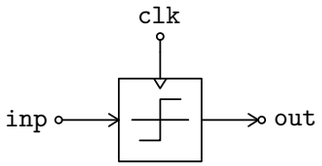
  - Interface file specification
  - Code model
  - Simulation example

```
test_quant2lsh.cir
* Test circuit for quant2lsh CM

vin 1 0 sin(0 1 1e4)
vctl 2 0 dc=1

aclk %v(2) %d(3) myclock
.model myclock d_osc(cntl_array=[0 1] freq_array=[0 1e5])

aquant2lsh %v(1) %d(3) %d(4) myquant2lsh
.model myquant2lsh quant2lsh(inp_th=0.5 out_ic=0 pos_edge=0 t_rise=1e-9
t_fall=1e-9)

aprobe1 [%d(3)] [%v(5)] myprobe
aprobe2 [%d(4)] [%v(6)] myprobe
.model myprobe dac_bridge(out_low=0 out_high=1 t_rise=1e-9 t_fall=1e-9)

.end
```

```
test_quant2lsh.sp3
* Test script for quant2lsh CM
.control
delcirc all
destroy all
delete all
save all

source test_quant2lsh.cir
let @@myquant2lsh[t_rise]=1e-6
let @@myquant2lsh[t_fall]=1e-6
tran 1e-9 1e-4
let vin=v(1)
let vclk=v(5)
let vout=v(6)
plot create plot1 vin vclk vout vs time xlabel 'Time [s]' ylabel 'Output ...

.endc
.end
```

XSpice Examples

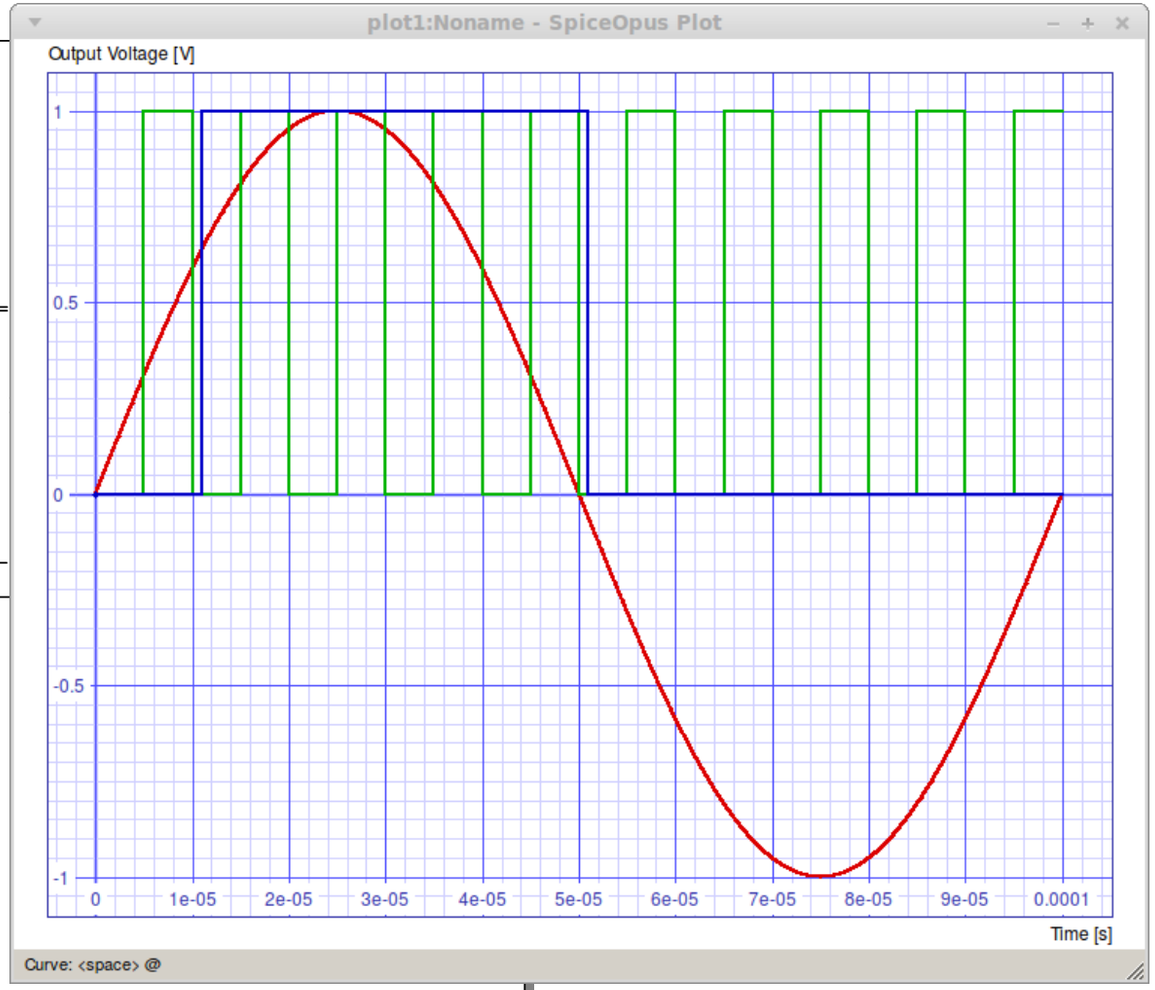
- 2-level **quantizer** with S/H:



  - Interface file specification
  - Code model
  - Simulation example

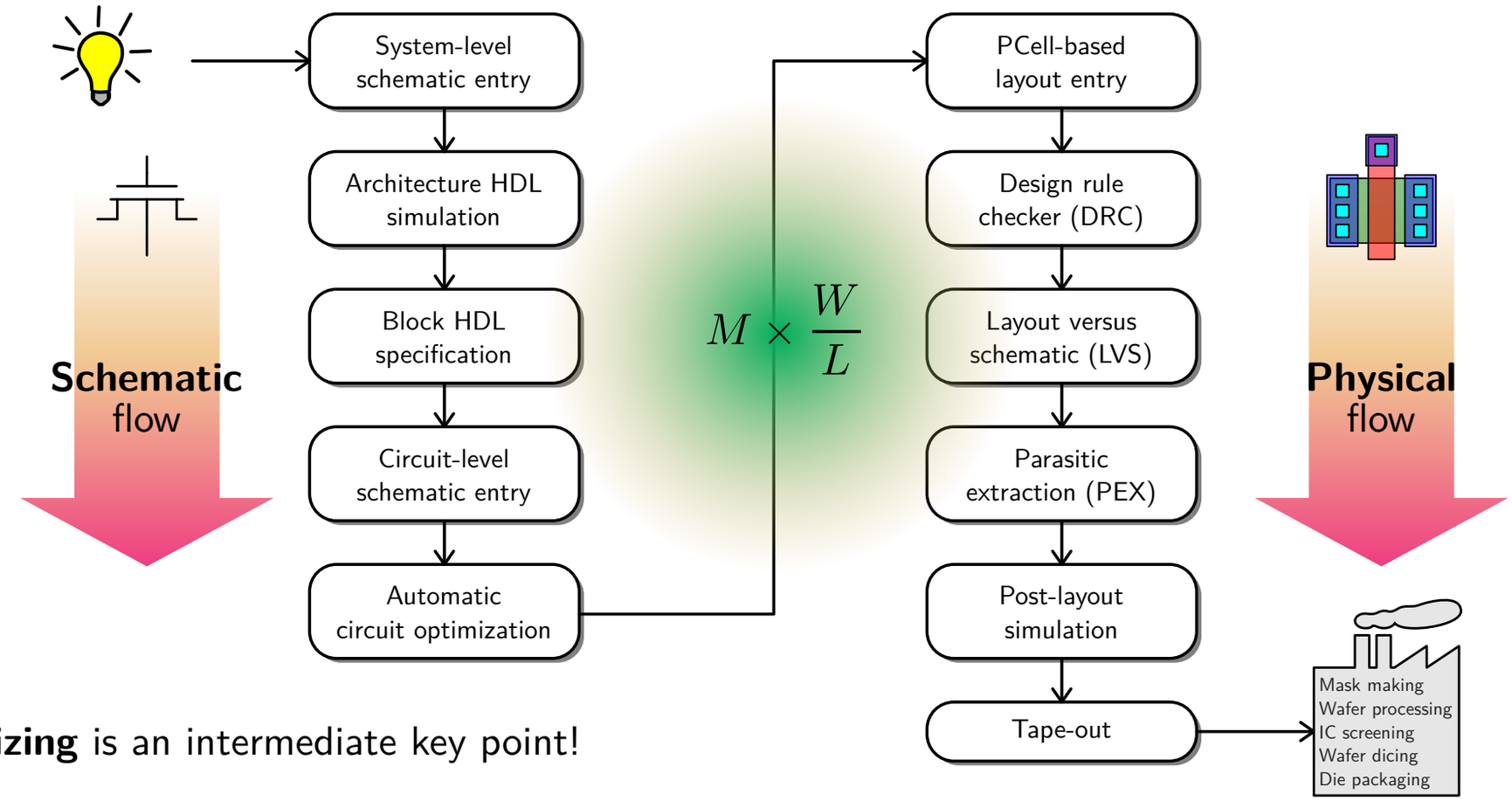
```
test_quant2lsh.cir
* Test circuit for quant2lsh CM

vin 1 0 sin(0 1 1e4)
vctl 2 0 dc=1

aclk %v(2) %d(3) myclock
.model myclock d_osc(cntl_array=[0 1] freq_array=

aquant2lsh %v(1) %d(3) %d(4) myquant2lsh
.model myquant2lsh quant2lsh(inp_th=0.5 out_ic=0
t_fall=1e-9)

aprobe1 [%d(3)] [%v(5)] myprobe
aprobe2 [%d(4)] [%v(6)] myprobe
.model myprobe dac_bridge(out_low=0 out_high=1 t_

.end
```

```
test_quant2lsh.sp3
* Test script for quant2lsh CM
.control
delcirc all
destroy all
delete all
save all

source test_quant2lsh.cir
let @@myquant2lsh[t_rise]=1e-6
let @@myquant2lsh[t_fall]=1e-6
tran 1e-9 1e-4
let vin=v(1)
let vclk=v(5)
let vout=v(6)
plot create plot1 vin vclk vout vs time xlabel 'Time [s]' ylabel 'Output ...

.endc
.end
```



Full-Custom Analog IC Design Flow

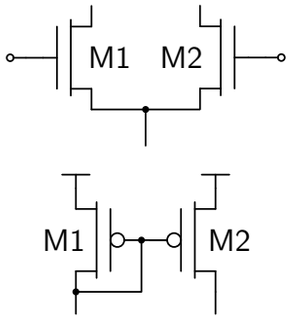
- From circuit **idea to layout** masks...
- **EDA**-assisted methodology



- **Device sizing** is an intermediate key point!

MOSFET Sizing Analog Guidelines

- Transistor (also other devices) matching **ratios**:



multiplicity (**M**) design for a common channel aspect ratio (W/L)

non-minimum channel length (**L**) above design rules for a given W/L

$$M \times \frac{W}{L}$$

overall **M** and **W/L** design for a given absolute L value

- Device output **impedance**:

$$\lambda \propto \frac{1}{L} \downarrow$$

- MOS **transconductance**:

$$M\frac{W}{L} \uparrow \quad g_{mg,s} \uparrow$$

- **Bandwidth**, technology **mismatching** and **flicker** noise:

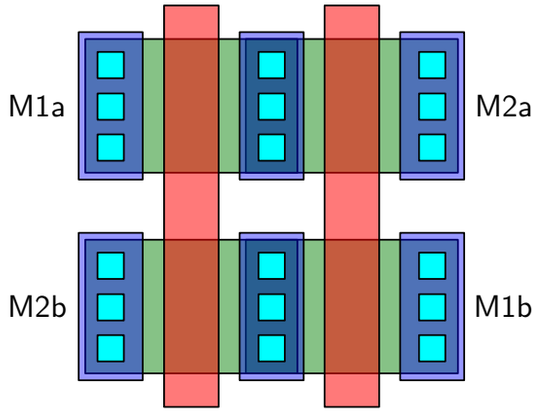
$$C_{ox} = \frac{\epsilon_{ox}}{t_{ox}} MWL$$

$$\sigma\left(\Delta P\right) \simeq \frac{A_P}{\sqrt{MWL}}$$

device area (**MWL**) design for a given M and W/L

$$\frac{dv_{nfk}^2}{df} = \frac{K_{fk}}{MWL}\frac{1}{f}$$

- Physical **layout** considerations:

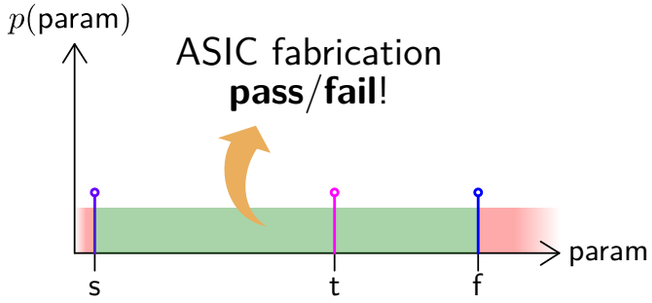


M integer design for an overall MW/L

e.g. M **even** values for common centroid layout techniques

Process Simulation

- **Global** deviations of model parameters:
  - Same change in **all devices** of the ASIC
  - Worse-case **corner** analysis: (t)yp, (f)ast, (s)low...

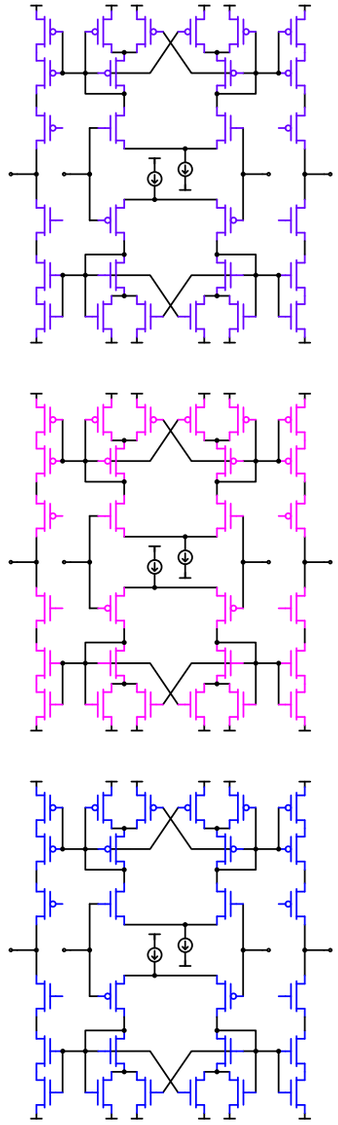


cnm25proc.lib

```
* CNM25 process corners

.lib common
.model cnm25modn nmos LEVEL = 2
+ TOX = 380E-10     VTO = {vton}     NSUB = 2.64E16   UO = {uon}
+ UCRIT = 1E4       UEXP = 6.86E-2   NFS = 7.11E11
+ DELTA = 2.20      RS = 93.8        LD = 9.13E-7     XJ = 8.24E-8
+ VMAX = 5.96E4     NEFF = 1.48      CJ = 3.50E-4     MJ = .40
+ CJSW = 5.95E-10   MJSW = .29       PB = .65
+ AF   =1.33  KF    =1e-29
.model cnm25modp pmos LEVEL = 2
+ TOX = 380E-10     VTO = {vtop}     NSUB = 1.36E16   UO = {uop}
+ UCRIT = 1E4       UEXP = 1.16E-1   NFS = 6.62E11
+ DELTA = 1.82      RS = 134.9       LD = 8.10E-7     XJ = 2.78E-9
+ VMAX = 1.20E5     NEFF = 6.67E-2   CJ = 3.82E-4     MJ = .35
+ CJSW = 7.38E-10   MJSW = .39       PB = .56
+ AF   =1.33  KF    =1e-29
.model cnm25cpoly c CJ= 4.227E-4  CJSW=0.0
.endl

.lib tt
.param vton=.942
.param vtop=-1.139
.param uon=648
.param uop=213
.lib 'cnm25proc.lib' common
.endl

.lib ss
.param vton=1.1
.param vtop=-1.3
.param uon=415
.param uop=131
.lib 'cnm25proc.lib' common
.endl

.lib ff
.param vton=0.7
.param vtop=-0.9
.param uon=881
.param uop=295
.lib 'cnm25proc.lib' common
.endl
```

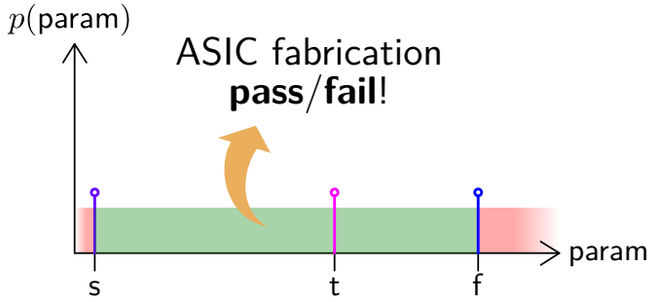
$V_{TH} \uparrow$, $\beta \downarrow$ slow

$V_{TH} \downarrow$, $\beta \uparrow$ fast

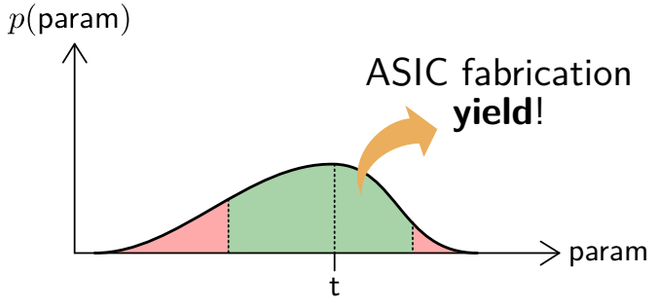


Process Simulation

- **Global** deviations of model parameters:
  - Same change in **all devices** of the ASIC
  - Worse-case **corner** analysis: (t)yp, (f)ast, (s)low...
  - Combined corners: process/voltage/temperature (**PVT**)



  - **Montecarlo** statistical analysis:

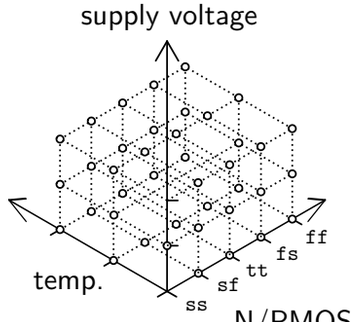


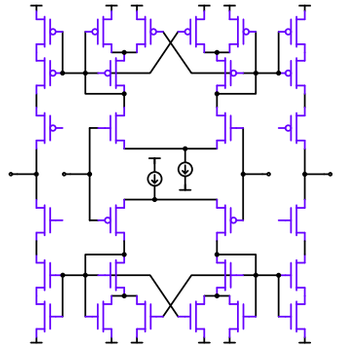
cnm25proc.lib

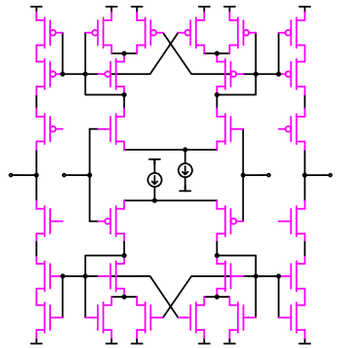
```
* CNM25 process corners

.lib common
.model cnm25modn nmos LEVEL = 2
+ TOX = 380E-10     VTO = {vton}     NSUB = 2.64E16   UO = {uon}
+ UCRIT = 1E4       UEXP = 6.86E-2   NFS = 7.11E11
+ DELTA = 2.20      RS = 93.8        LD = 9.13E-7     XJ = 8.24E-8
+ VMAX = 5.96E4     NEFF = 1.48      CJ = 3.50E-4     MJ = .40
+ CJSW = 5.95E-10   MJSW = .29       PB = .65
+ AF   =1.33  KF    =1e-29
.model cnm25modp pmos LEVEL = 2
+ TOX = 380E-10     VTO = {vtop}     NSUB = 1.36E16   UO = {uop}
+ UCRIT = 1E4       UEXP = 1.16E-1   NFS = 6.62E11
+ DELTA = 1.82      RS = 134.9       LD = 8.10E-7     XJ = 2.78E-9
+ VMAX = 1.20E5     NEFF = 6.67E-2   CJ = 3.82E-4     MJ = .35
+ CJSW = 7.38E-10   MJSW = .39       PB = .56
+ AF   =1.33  KF    =1e-29
.model cnm25cpoly c CJ= 4.227E-4  CJSW=0.0
.endl

.lib tt
.param vton=.942
.param vtop=-1.139
.param uon=648
.param uop=213
.lib 'cnm25proc.lib' common
.endl

.lib ss
.param vton=1.1
.param vtop=-1.3
.param uon=415
.param uop=131
.lib 'cnm25proc.lib' common
.endl

.lib ff
.param vton=0.7
.param vtop=-0.9
.param uon=881
.param uop=295
.lib 'cnm25proc.lib' common
.endl
```

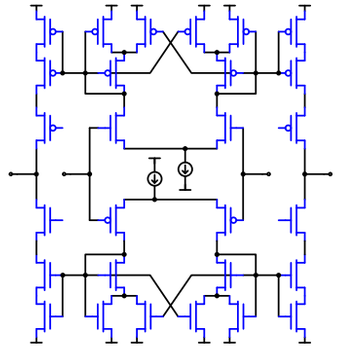
$V_{TH} \uparrow$ $\beta \downarrow$ slow

$V_{TH} \downarrow$ $\beta \uparrow$ fast



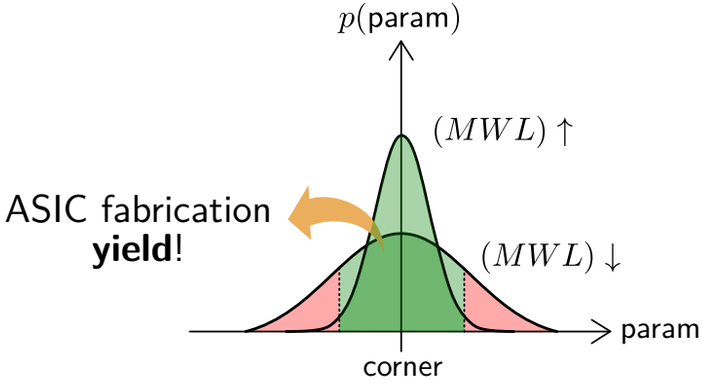






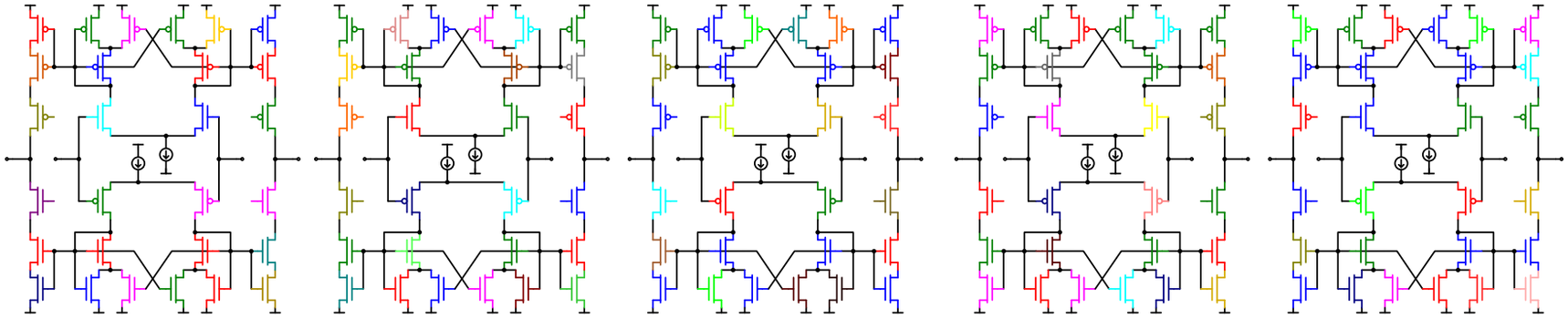
Mismatching Simulation

- **Local** deviations of model parameters:
  - Different change for **each device** of the ASIC
  - **Pelgrom Law**
  - **Montecarlo** statistical analysis:



cnm25match.lib

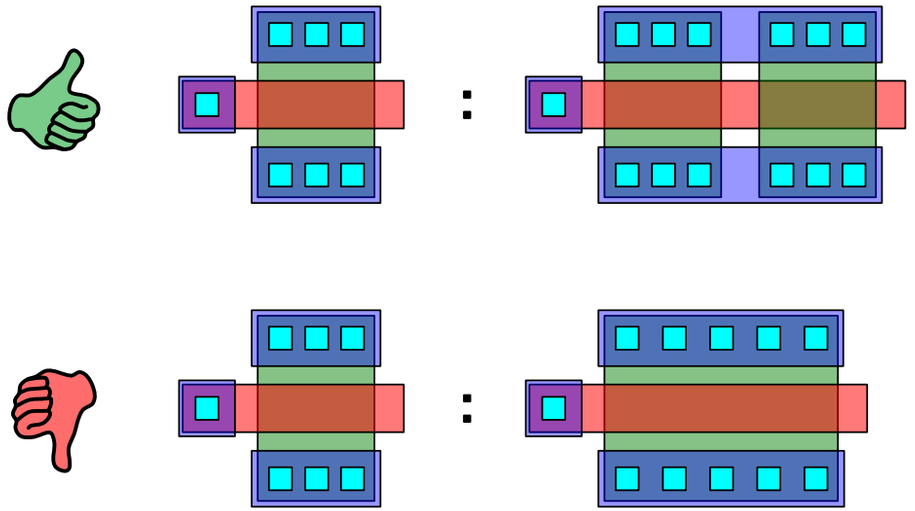
```
.param avto=30e-9
.param rauo=5e-8
.param rac=7.3e-8

.subckt cnm25modn d g s b param: w=3u l=3u
 + ad=0 as=0 pd=0 ps=0 m=1
.param vton=.942+rndgauss(avto/sqrt(m*w*l))
.param uon=648*(1+rndgauss(rauo/sqrt(m*w*l)))
.model modnlocal nmos level=2 vto={vton} uo={uon} tox=...
mn d g s b modnlocal w={w} l={l}
 + ad={ad} as={as} pd={pd} ps={ps} m={m}
.ends

.subckt cnm25cpoly t b param: w=30u l=30u m=1
.param cj=4.227E-4*(1+rndgauss(rac/sqrt(m*w*l)))
.model cpolylocal c cj={cj} cjsw=0.0
cpip t b cpolylocal w={w} l={l} m={m}
.ends
```



General Matching Rules

- **Unitary** elements

e.g. 1 : 2 ratio

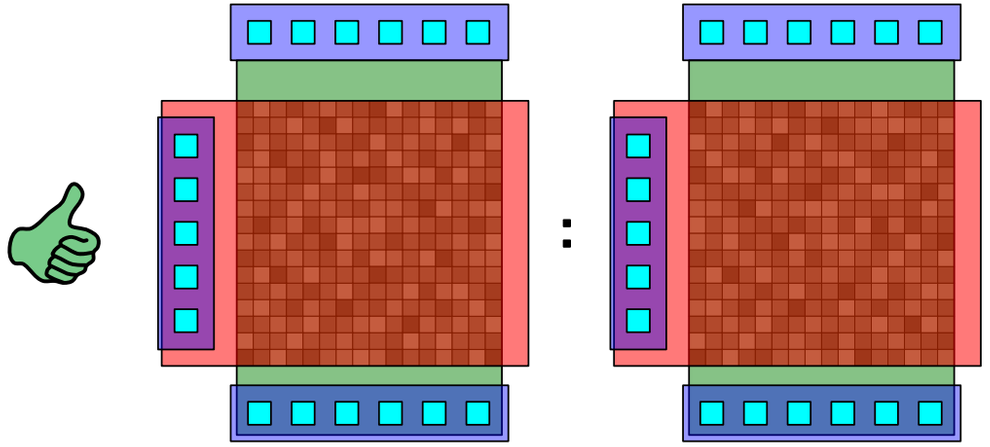


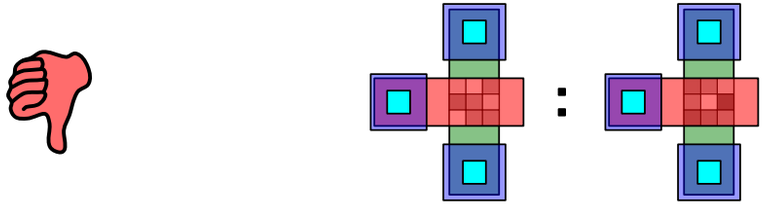
- Play with **multiplicity** only
- Same ratio for **second order** effects also (e.g. area and perimeter ratio in caps)
- **Larger area** for the overall array

General Matching Rules

- **Unitary** elements
- Large **area** devices

e.g. 1 : 1 ratio



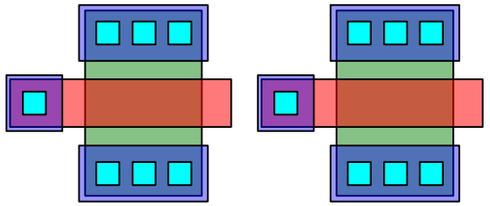


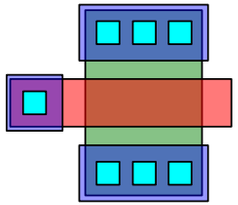
- Technology local **granularity** (e.g. distribution of dopants)
- **Pelgrom Law** $\propto \frac{1}{\sqrt{WL}}$

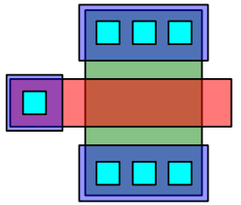
General Matching Rules

- **Unitary** elements
- Large **area** devices
- Minimum **distance**

e.g. 1 : 1 ratio



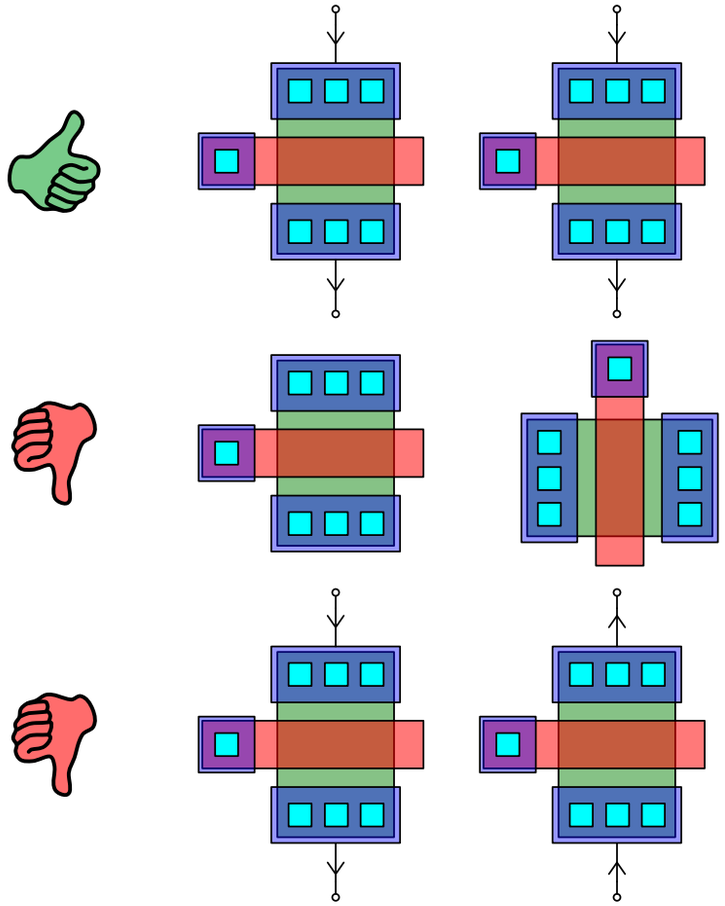




- Technology global **drifts**
  (e.g. gate oxide thickness slope)
- **Design rule** spacing limits

General Matching Rules

- **Unitary** elements
- Large **area** devices
- Minimum **distance**
- Same **orientation**

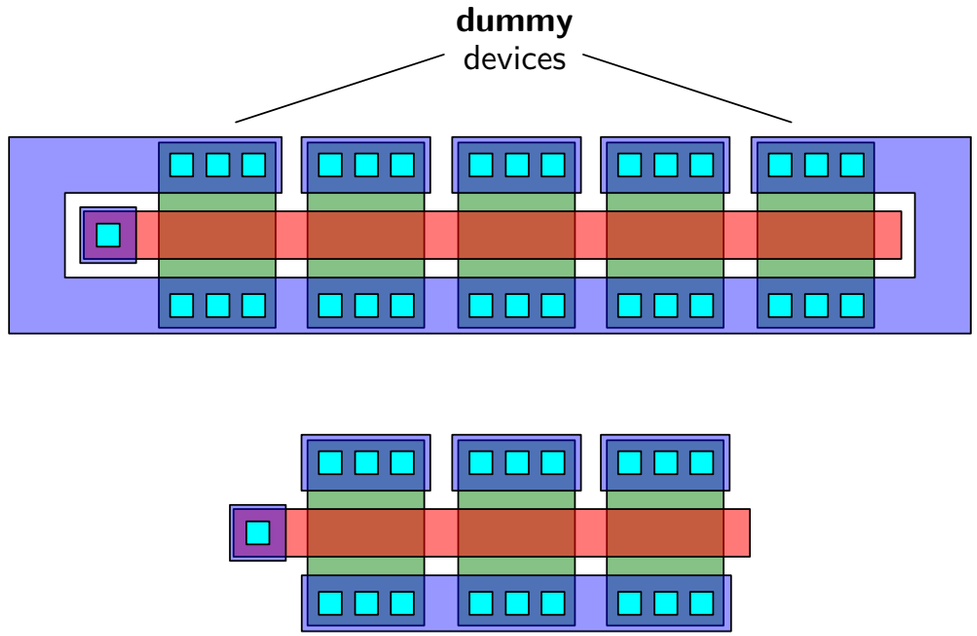


- **Anisotropic** materials (e.g. wafer crystal lattice orientation)
- Longer **routing** may be required...

General Matching Rules

- **Unitary** elements
- Large **area** devices
- Minimum **distance**
- Same **orientation**
- Same **sorround**

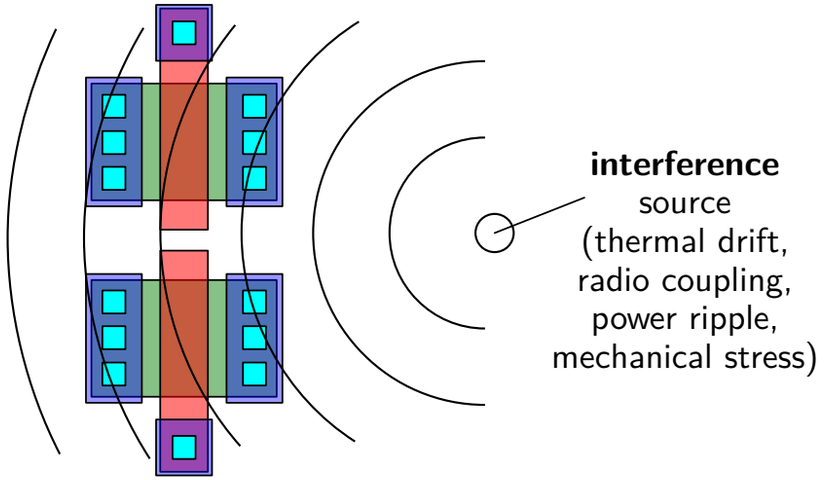
e.g. 1 : 1 : 1 ratios

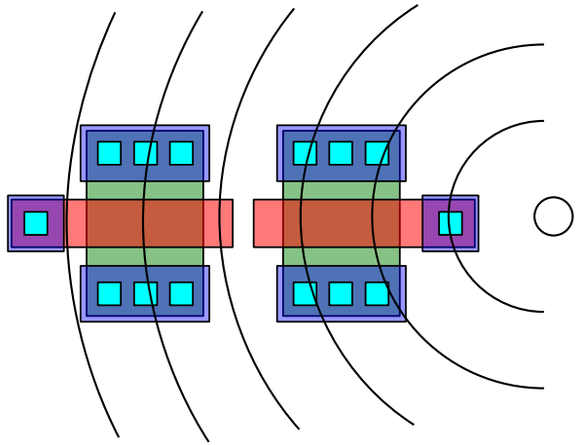


- Attenuation of **proximity** effects
- **Inter-device** second order effects (e.g. parasitic RLC)
- **Larger area** for the overall array

General Matching Rules

- **Unitary** elements
- Large **area** devices
- Minimum **distance**
- Same **orientation**
- Same **sorround**
- Same **symmetry**



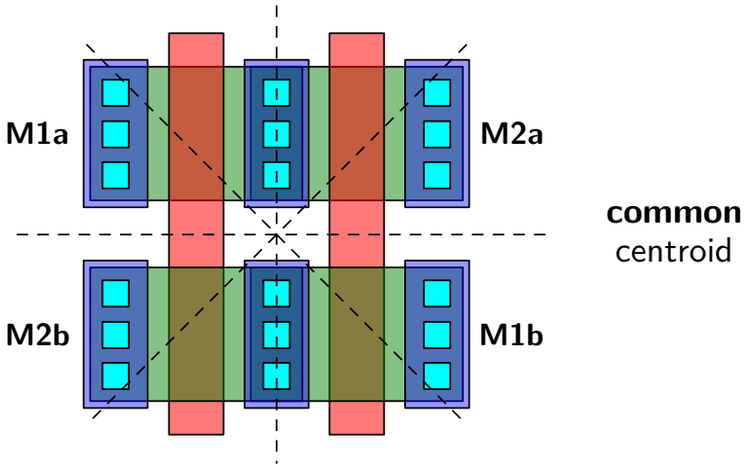


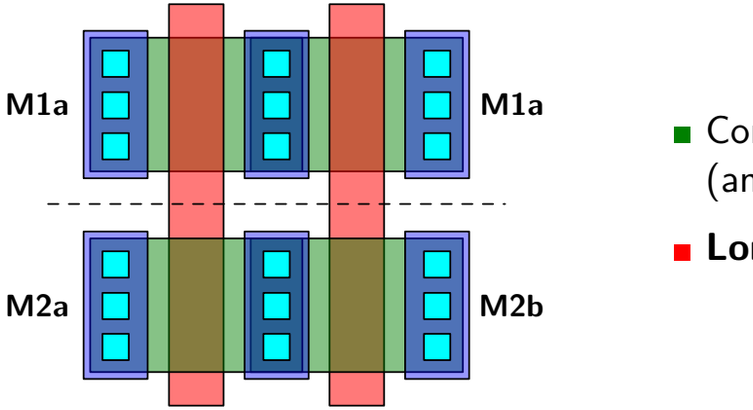
- **Differential** circuits (common mode interference)
- Complex **floorplan**

General Matching Rules

- **Unitary** elements
- Large **area** devices
- Minimum **distance**
- Same **orientation**
- Same **sorround**
- Same **symmetry**

e.g. 2 : 2 ratio



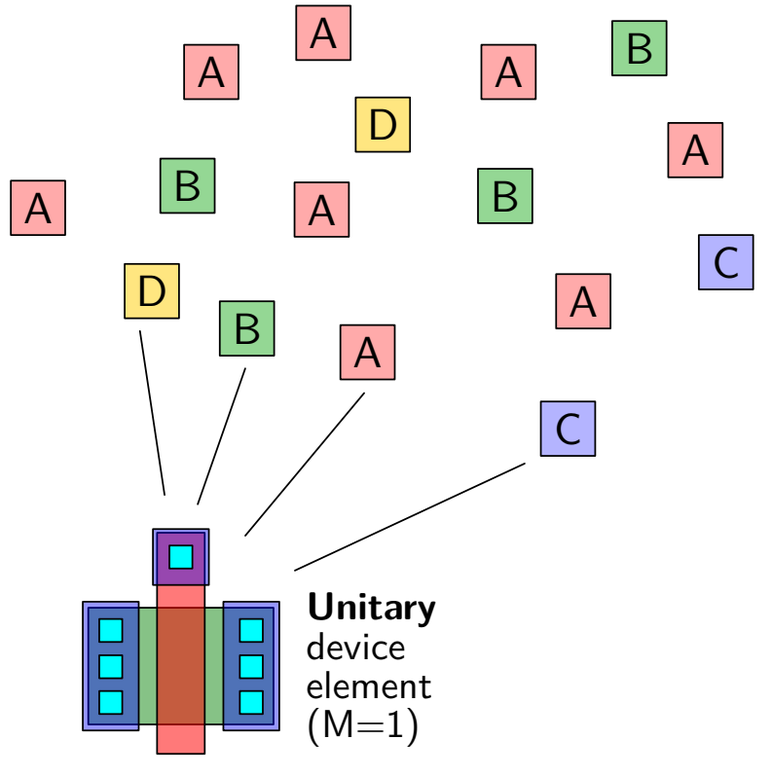


- Compensation of **linear** gradients (and non-linear at short distance)
- **Longer routing** is usually required...

Common Centroid Arrays

- Difficult to achieve for **large** and **multiple groups** of unitary elements

e.g. A : B : C : D ratios
8 4 2 2



Common Centroid Arrays

- Difficult to achieve for **large** and **multiple groups** of unitary elements
- Centroid as a **center-of-mass** concept, but with area weights...

each element: area A_i, center-of-mass coordinates C_i

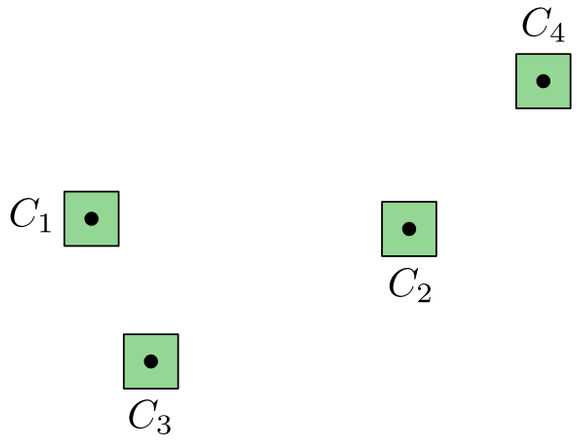
$$\sum_{i=1}^{M} A_i \left(C_i - C_0\right) = 0$$

centroid coordinates

$$C_0 = \frac{\sum_{i=1}^{M} A_i C_i}{\sum_{i=1}^{M} A_i}$$

If all elements are of **same area**:

$$C_0 \equiv \frac{1}{M} \sum_{i=1}^{M} C_i$$



Common Centroid Arrays

- Difficult to achieve for **large** and **multiple groups** of unitary elements
- Centroid as a **center-of-mass** concept, but with area weights...

each element: area A_i, center-of-mass coordinates C_i

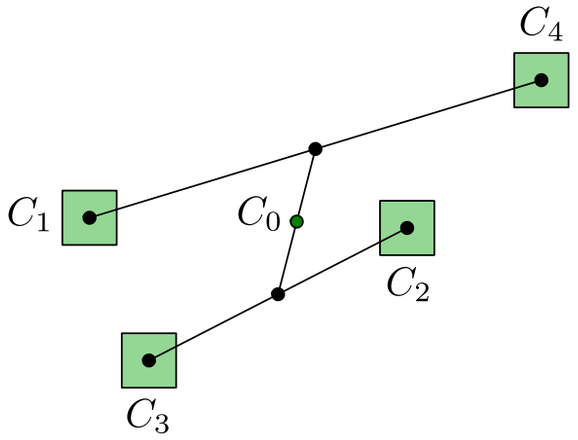
$$\sum_{i=1}^{M} A_i \left(C_i - C_0\right) = 0$$

centroid coordinates

$$C_0 = \frac{\sum_{i=1}^{M} A_i C_i}{\sum_{i=1}^{M} A_i}$$

If all elements are of **same area**:

$$C_0 \equiv \frac{1}{M} \sum_{i=1}^{M} C_i$$

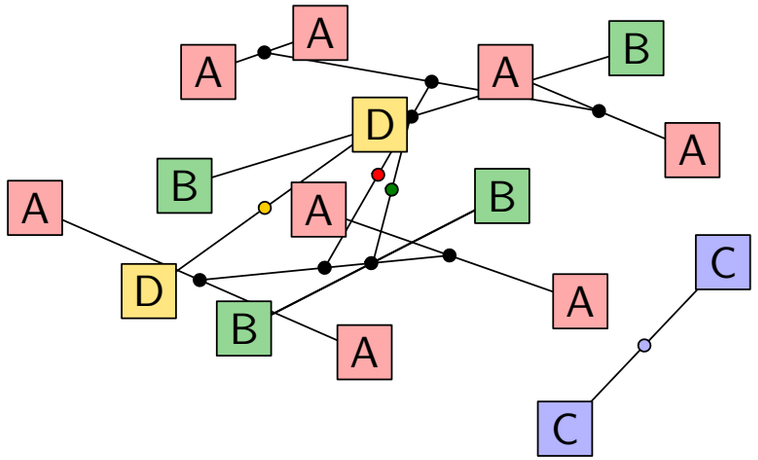


Center-of-area is the **average** of elements positions...

Common Centroid Arrays

- Difficult to achieve for **large** and **multiple groups** of unitary elements
- Centroid as a **center-of-mass** concept, but with area weights...
- **Centroid**-based golden rules:
  - **Coincidence** of all centroids

e.g. A : B : C : D ratios
8 4 2 2

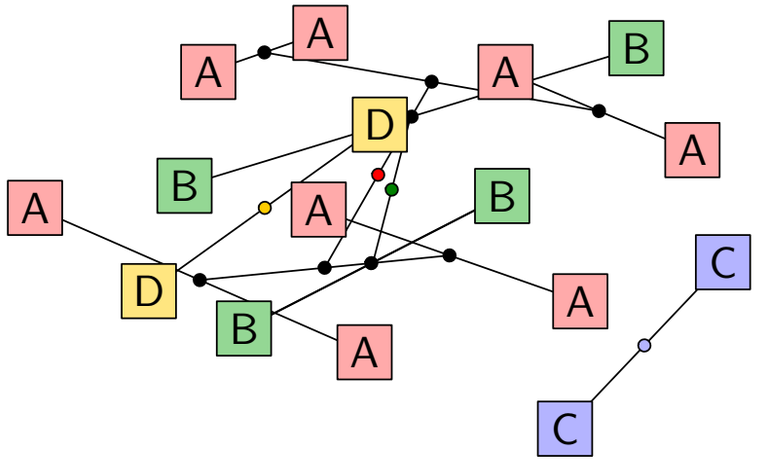


Not a common centroid arrangement!

Common Centroid Arrays

- Difficult to achieve for **large** and **multiple groups** of unitary elements
- Centroid as a **center-of-mass** concept, but with area weights...
- **Centroid**-based golden rules:
  - **Coincidence** of all centroids
  - **Symmetry** for X and Y axes

e.g. A : B : C : D ratios
8 4 2 2

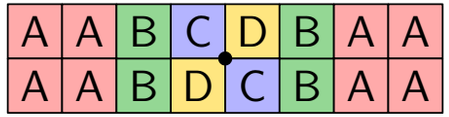


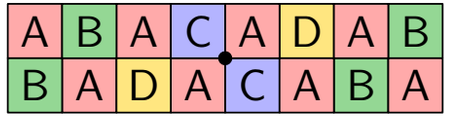
A	A	B	C	D	B	A	A
A	A	B	D	C	B	A	A

Common Centroid Arrays

- Difficult to achieve for **large** and **multiple groups** of unitary elements
- Centroid as a **center-of-mass** concept, but with area weights...
- **Centroid**-based golden rules:
  - **Coincidence** of all centroids
  - **Symmetry** for X and Y axes
  - **Dispersion** of groups as uniformly as possible

e.g. A : B : C : D ratios
8 4 2 2

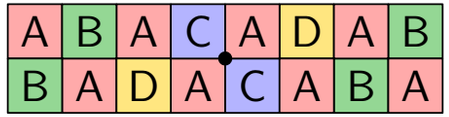


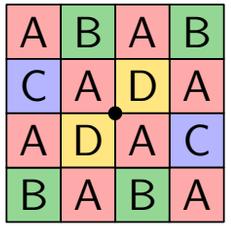


Common Centroid Arrays

- Difficult to achieve for **large** and **multiple groups** of unitary elements
- Centroid as a **center-of-mass** concept, but with area weights...
- **Centroid**-based golden rules:
  - **Coincidence** of all centroids
  - **Symmetry** for X and Y axes
  - **Dispersion** of groups as uniformly as possible
  - **Compactness** of overall array to ideal square shape

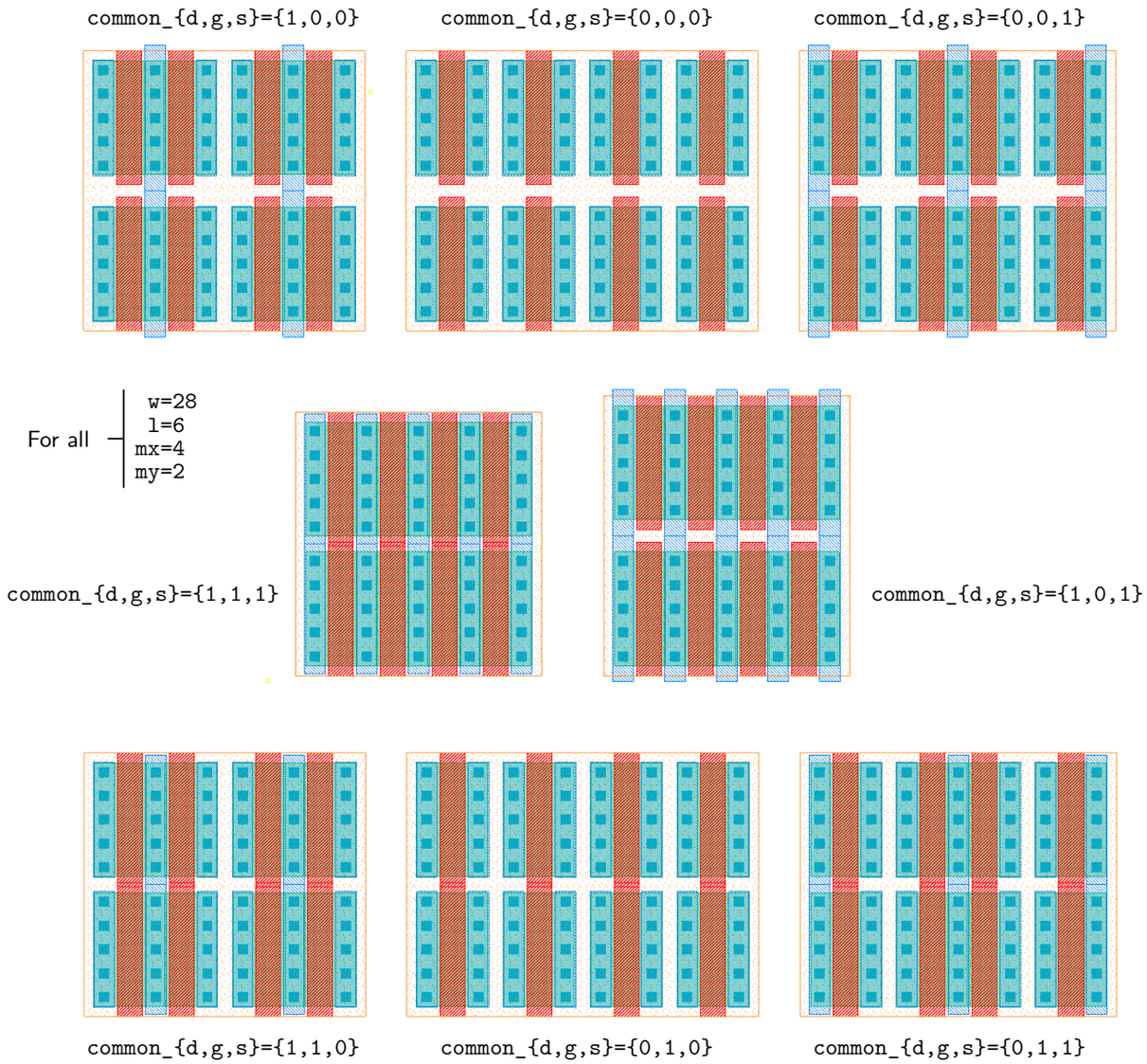
e.g. A : B : C : D ratios
8 4 2 2





PCell-Based Layout

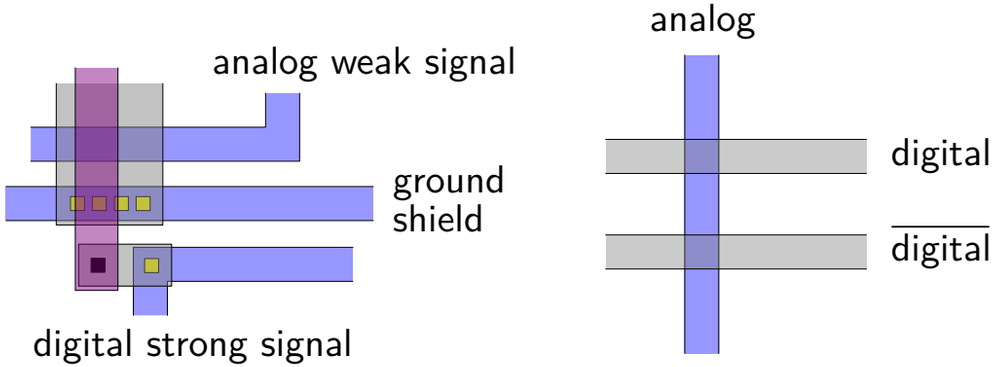
- **Unitary** elements
- **Regular** geometry
- **Design rule** compliant



e.g. CNM25 NMOSFET parameterized cell (PCell)

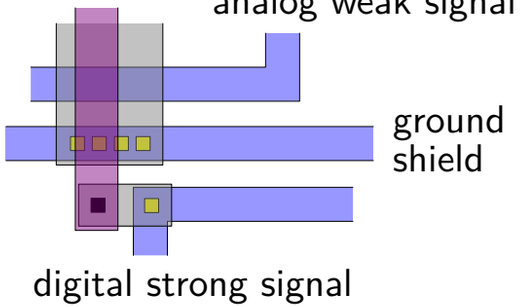
Decoupling Guidelines

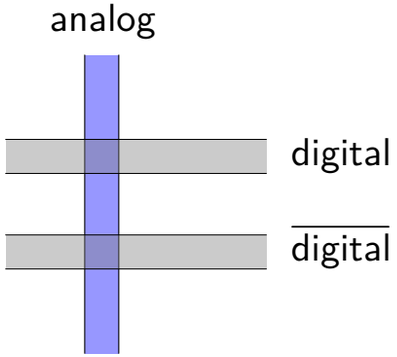
- **Signal integrity** between analog, digital, RF... domains



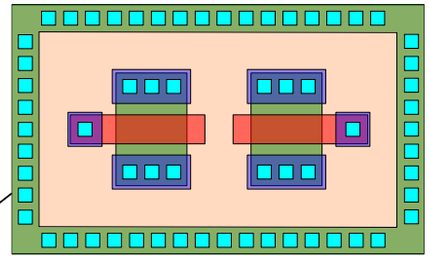
Decoupling Guidelines

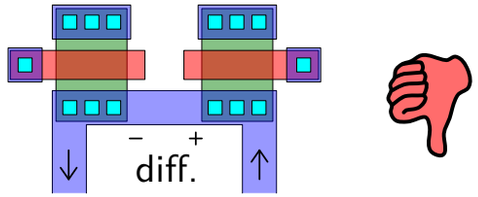
- **Signal integrity** between analog, digital, RF... domains

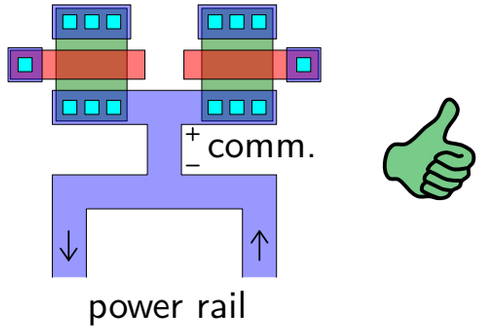




- Avoiding signal coupling through **power rails**

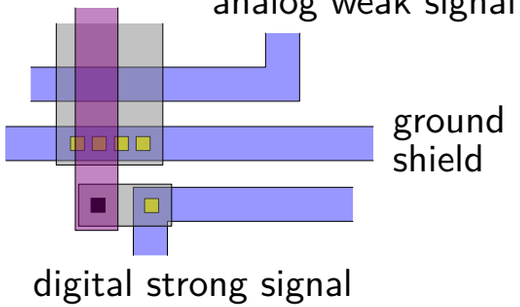


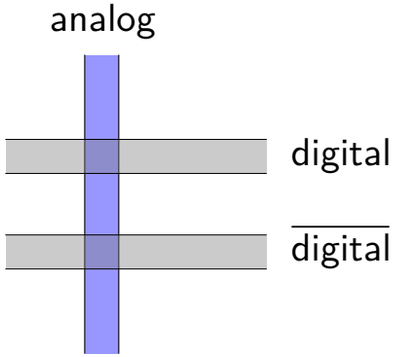




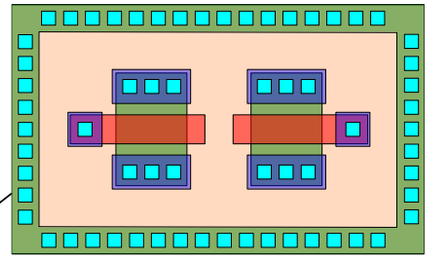
Decoupling Guidelines

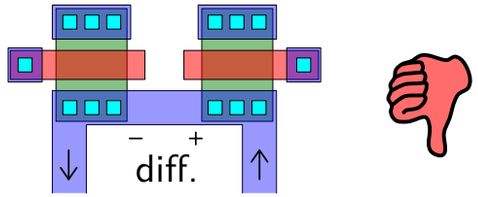
- **Signal integrity** between analog, digital, RF... domains



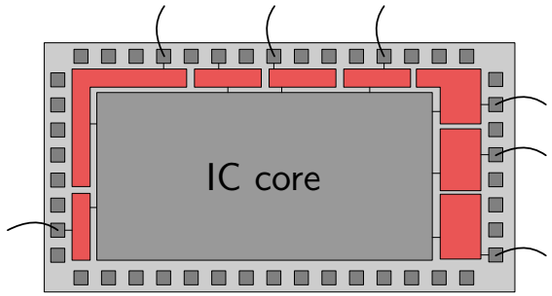


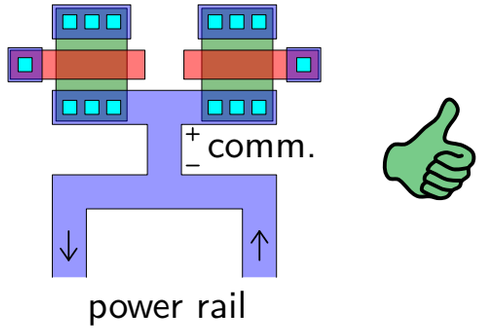
- Avoiding signal coupling through **power rails**



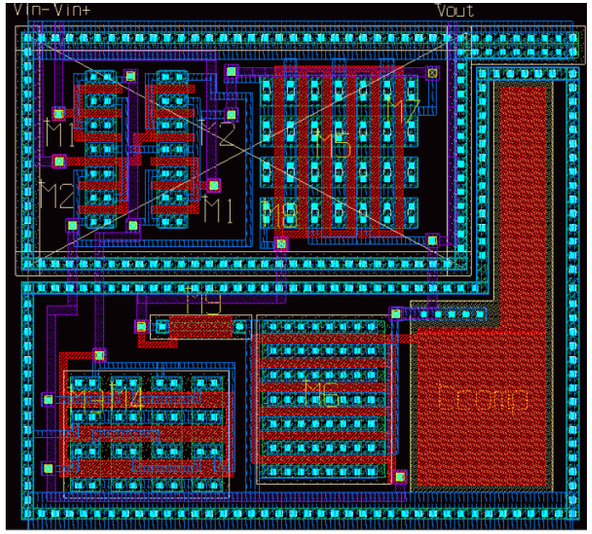


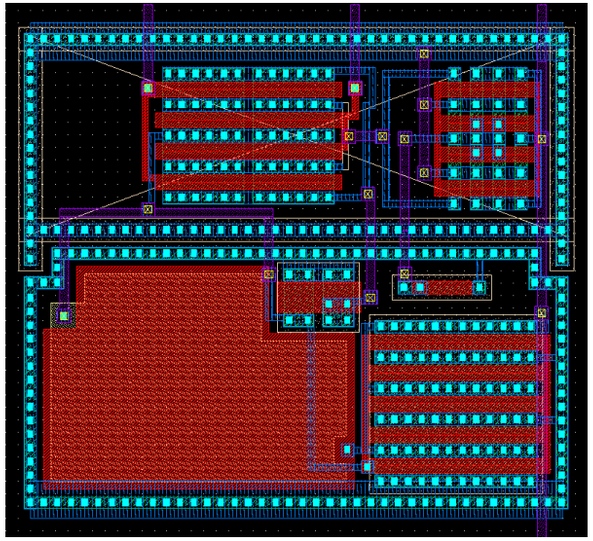
- Investing peripheral free area for on-chip **decoupling capacitors**

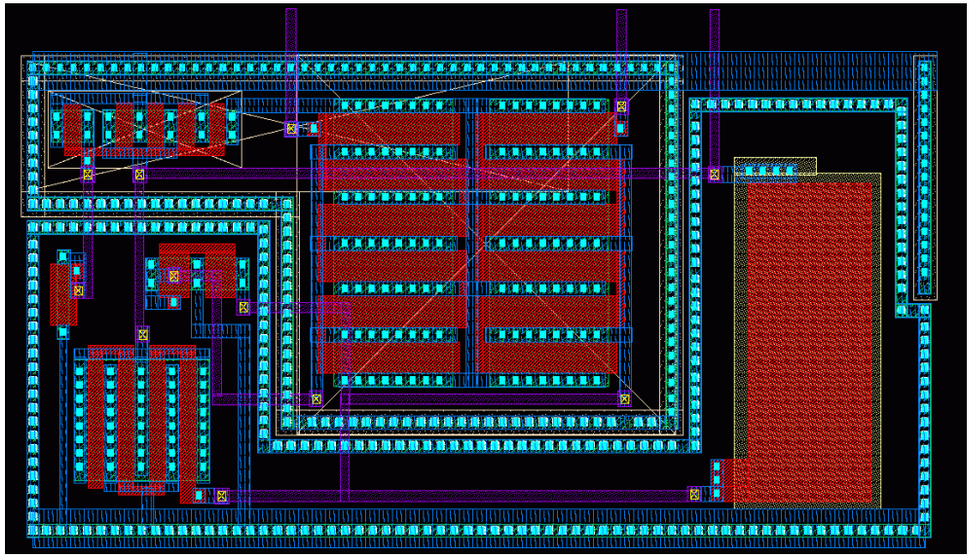


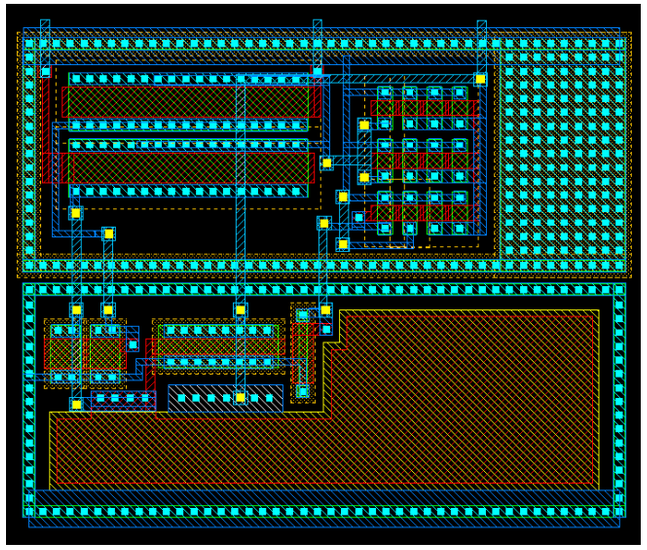


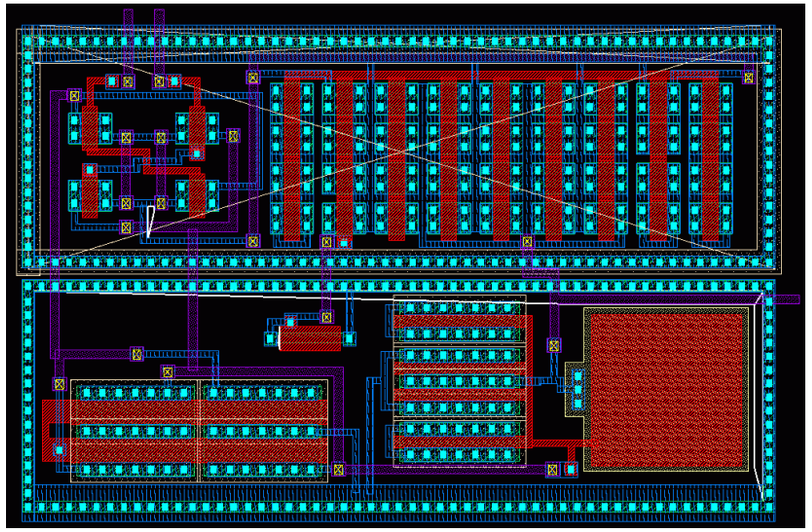
OpAmp Layout Examples (real cases from past students)

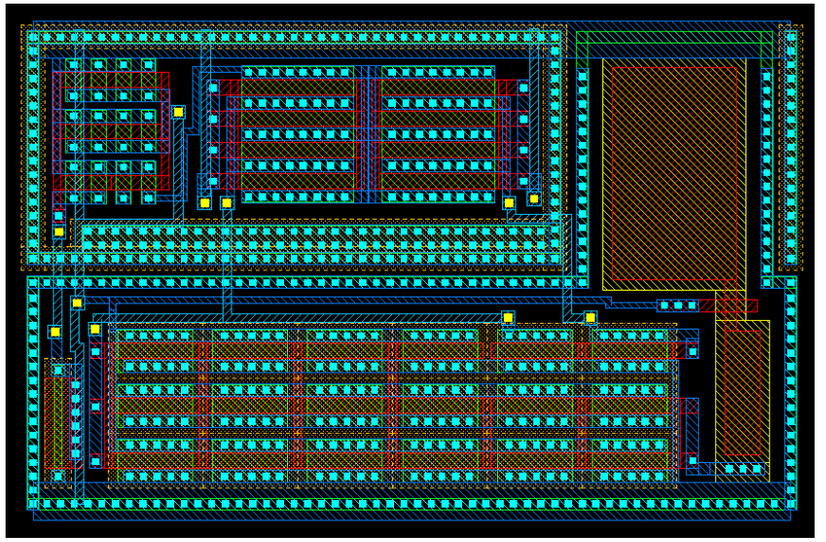




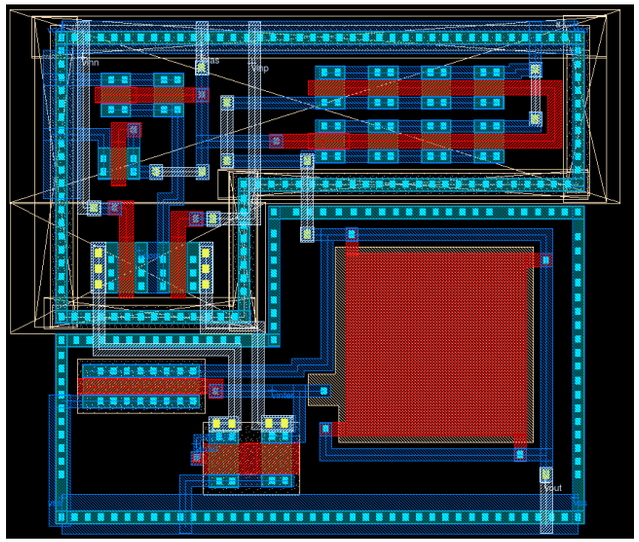


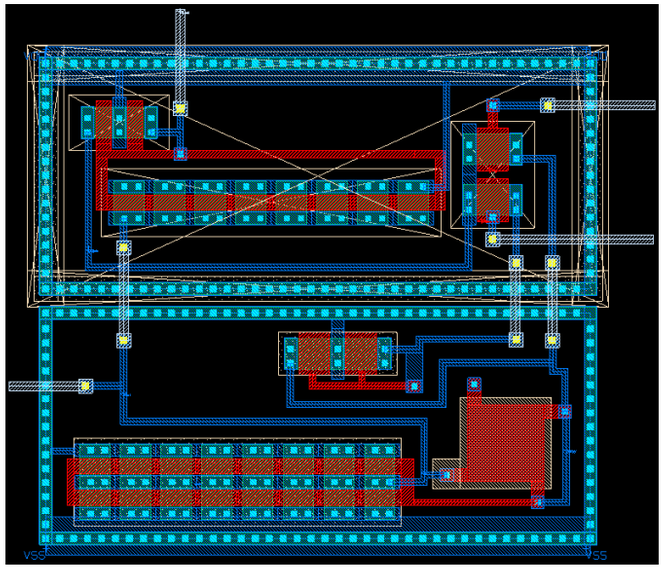


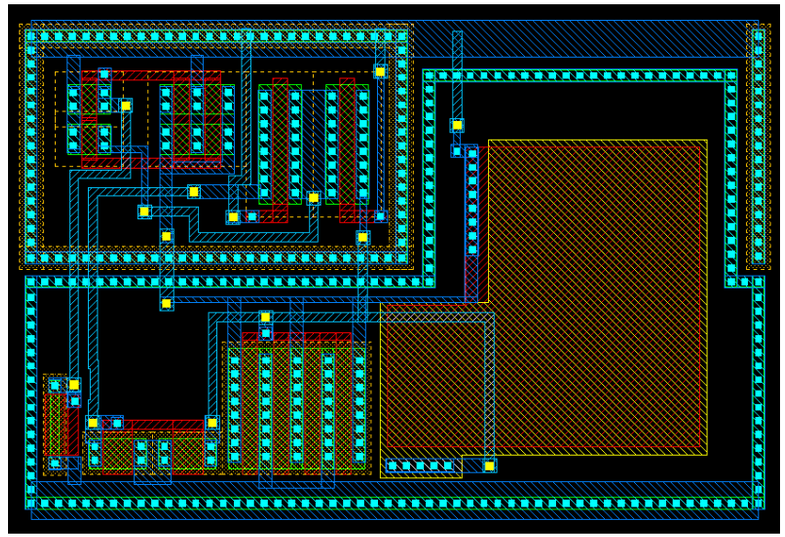


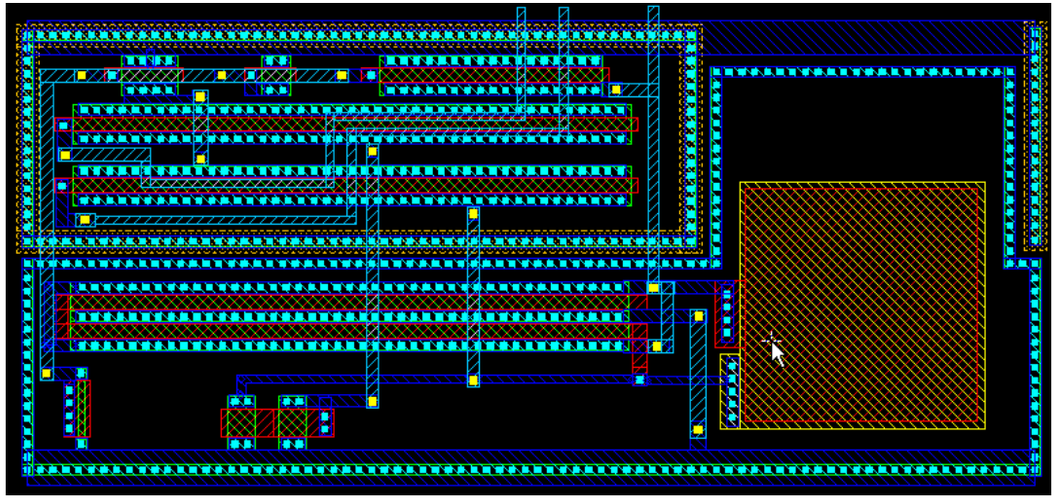


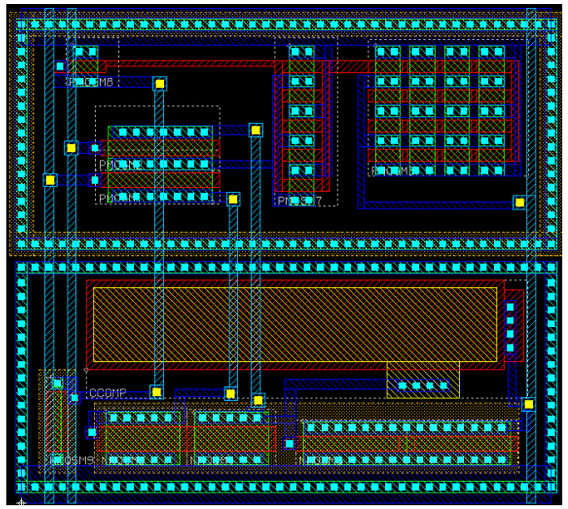
OpAmp Layout Examples (real cases from past students)

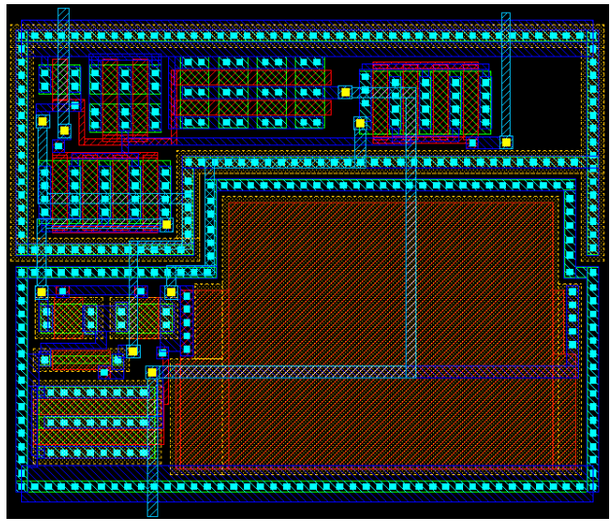








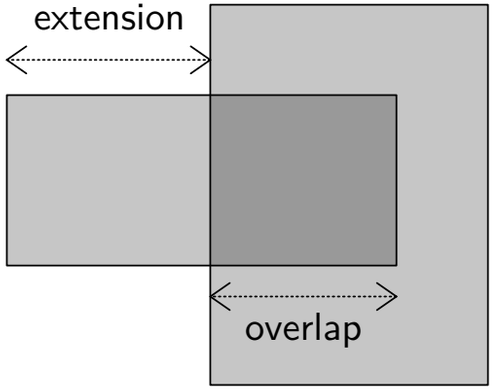


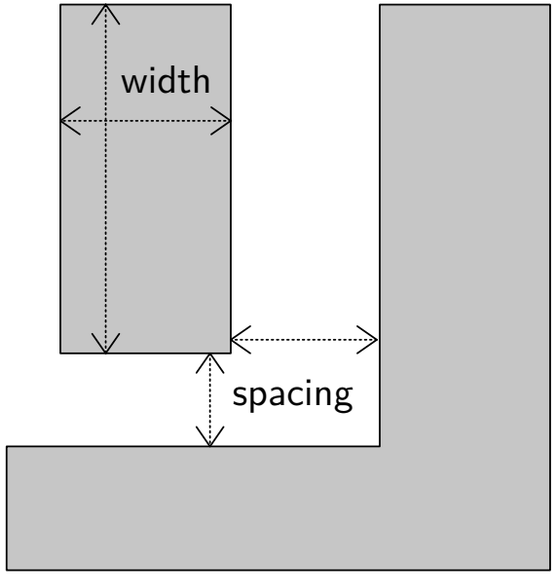


Geometrical Rules

- Basic **2D** concepts

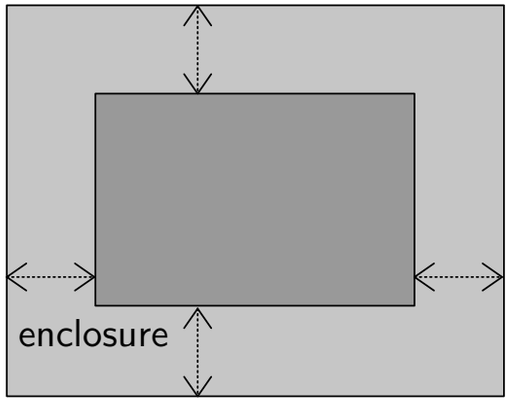
e.g. MOSFET gate

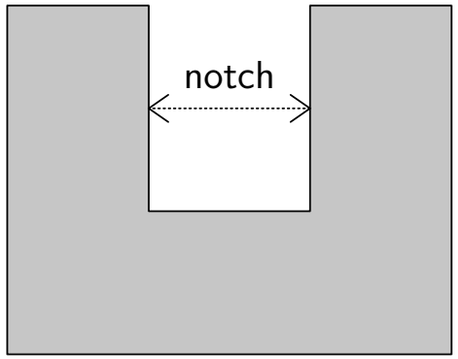




e.g. signal routing

e.g. MiM capacitor



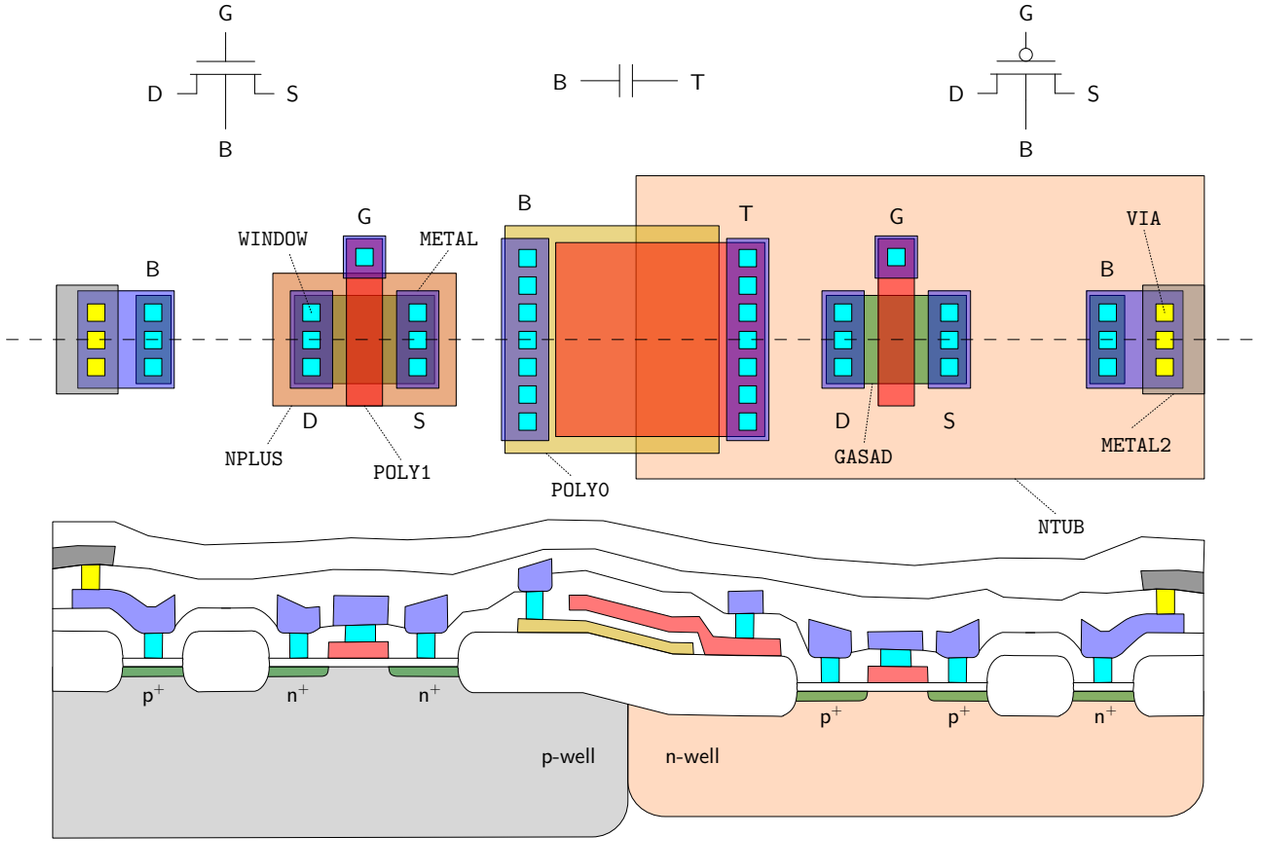


e.g. serpentine resistor

Geometrical Rules

- **Technology** rules set

e.g. 2.5um 2P2M CMOS (**CNM25**)



0.0	Design grid is 0.25um x 0.25um
1.1	N-well width >= 8um
1.2	N-well spacing and notch >= 8um
2.1	GASAD width >= 2um
2.2	GASAD spacing and notch >= 4um
2.3	N-well enclosure of P-plus active >= 5um
2.4	N-well spacing to N-plus active >= 5um
3.1	Poly0 width >= 2.5um
3.2	Poly0 spacing and notch >= 6um
3.3	Poly0 spacing to GASAD >= 6um
4.1.a	Poly1 width inside GASAD >= 3um
4.1.b	Poly1 width outside GASAD >= 2.5um
4.2	Poly1 spacing and notch >= 3um
4.3	GASAD extension of Poly1 >= 3um
4.4	Poly1 extension of GASAD >= 2.5um
4.5	Poly1 spacing to GASAD >= 1.25um
4.6	Poly0 enclosure of Poly1 >= 3um
5.1	N-plus enclosure of GASAD >= 2.5um
5.2	N-plus spacing to P-plus active >= 2.5um
5.3	N-plus spacing to Poly1 inside P-plus active >= 2um
5.4	N-plus extension of Poly1 inside N-plus active >= 1.5um
5.5	N-plus width >= 2.5um
5.6	N-plus spacing and notch >= 2.5um
6.1	Exact contact size = 2.5um x 2.5um
6.2	Contact spacing >= 3um
6.3	GASAD enclosure of Contact >= 1um
6.4	Poly1 enclosure of Contact >= 1.25um
6.5	Poly1 Contact spacing to GASAD >= 2.5um
6.6	Contact spacing to Poly1 inside GASAD >= 2um
6.9	Poly0 enclosure of Contact >= 4um
6.10	Contact spacing to Poly1 & Poly0 >= 4um
7.1	Metal1 width >= 2.5um
7.2	Metal1 spacing and notch >= 3um
7.3	Metal1 enclosure of Contact >= 1.25um
8.1	Exact via size = 3um x 3um
8.2	Via spacing >= 3.5um
8.3	Metal1 enclosure of Via >= 1.25um
8.4	Via spacing to Contact >= 2.5um
8.5	Via spacing to Poly1 >= 2.5um
9.1	Metal2 width >= 3.5um
9.2	Metal2 spacing and notch >= 3.5um
9.3	Metal2 enclosure of Via >= 1.25um
10.1	Exact passivation window size = 100um x 100um

Geometrical Rules

- **Technology** rules set

Process flow

Front end of line (**FEOL**)

Back end of line (**BEOL**)

0.0	Design grid is 0.25um x 0.25um
1.1	N-well width >= 8um
1.2	N-well spacing and notch >= 8um
2.1	GASAD width >= 2um
2.2	GASAD spacing and notch >= 4um
2.3	N-well enclosure of P-plus active >= 5um
2.4	N-well spacing to N-plus active >= 5um
3.1	Poly0 width >= 2.5um
3.2	Poly0 spacing and notch >= 6um
3.3	Poly0 spacing to GASAD >= 6um
4.1.a	Poly1 width inside GASAD >= 3um
4.1.b	Poly1 width outside GASAD >= 2.5um
4.2	Poly1 spacing and notch >= 3um
4.3	GASAD extension of Poly1 >= 3um
4.4	Poly1 extension of GASAD >= 2.5um
4.5	Poly1 spacing to GASAD >= 1.25um
4.6	Poly0 enclosure of Poly1 >= 3um
5.1	N-plus enclosure of GASAD >= 2.5um
5.2	N-plus spacing to P-plus active >= 2.5um
5.3	N-plus spacing to Poly1 inside P-plus active >= 2um
5.4	N-plus extension of Poly1 inside N-plus active >= 1.5um
5.5	N-plus width >= 2.5um
5.6	N-plus spacing and notch >= 2.5um
6.1	Exact contact size = 2.5um x 2.5um
6.2	Contact spacing >= 3um
6.3	GASAD enclosure of Contact >= 1um
6.4	Poly1 enclosure of Contact >= 1.25um
6.5	Poly1 Contact spacing to GASAD >= 2.5um
6.6	Contact spacing to Poly1 inside GASAD >= 2um
6.9	Poly0 enclosure of Contact >= 4um
6.10	Contact spacing to Poly1 & Poly0 >= 4um
7.1	Metal1 width >= 2.5um
7.2	Metal1 spacing and notch >= 3um
7.3	Metal1 enclosure of Contact >= 1.25um
8.1	Exact via size = 3um x 3um
8.2	Via spacing >= 3.5um
8.3	Metal1 enclosure of Via >= 1.25um
8.4	Via spacing to Contact >= 2.5um
8.5	Via spacing to Poly1 >= 2.5um
9.1	Metal2 width >= 3.5um
9.2	Metal2 spacing and notch >= 3.5um
9.3	Metal2 enclosure of Via >= 1.25um
10.1	Exact passivation window size = 100um x 100um

Design Rule Checker

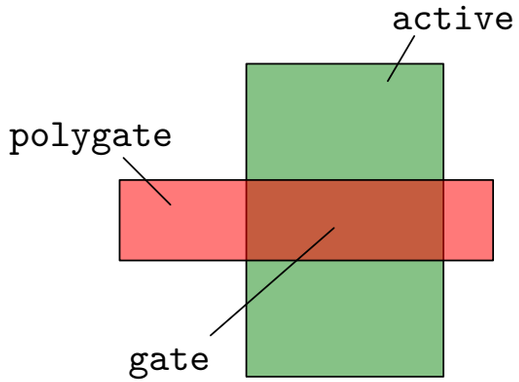
- **Programming** a rules set...

boolean operations

derived layers

cnm25drc.py

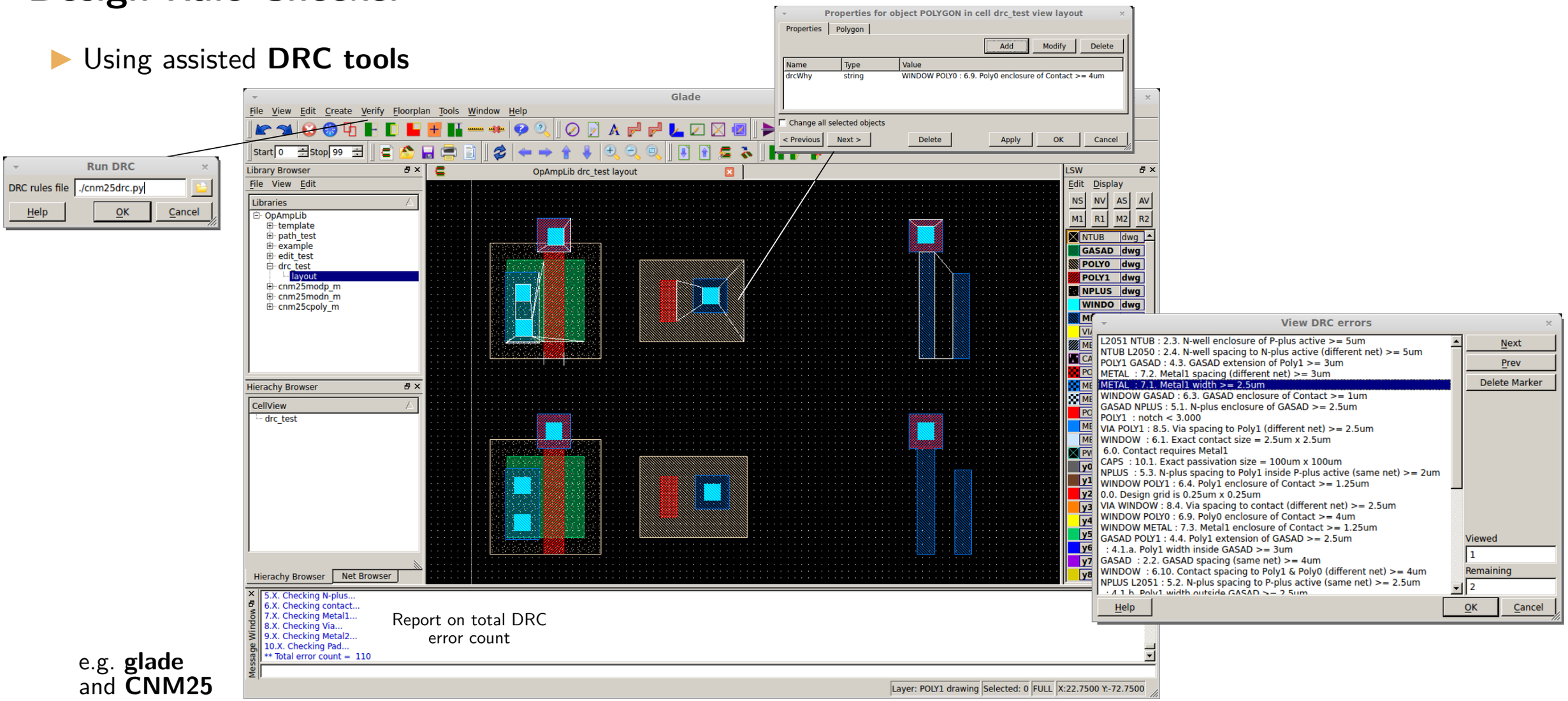
```
...
active   = geomGetShapes("GASAD", "drawing")
polygate = geomGetShapes("POLY1", "drawing")
polycap  = geomGetShapes("POLY0", "drawing")
gate     = geomAnd(polygate, active)
cpoly    = geomAnd(polygate, polycap)
geomOffGrid(polygate, 0.25, 1, "0.0. Design grid is ...
geomWidth(gate, 3, "4.1.a. Poly1 width inside GASAD >= ...
geomSpace(polygate, 3, diffnet, "4.2. Poly1 spacing...
geomNotch(polygate, 3, "4.2. Poly1 notch >= 3um")
geomExtension(polygate, active, 2.5, "4.4. Poly1 ext...
geomEnclose(polycap, cpoly, 3, "4.6. Poly0 enclosure...
...
```



e.g. 2.5um 2P2M CMOS (**CNM25**)

Design Rule Checker

- Using assisted **DRC tools**

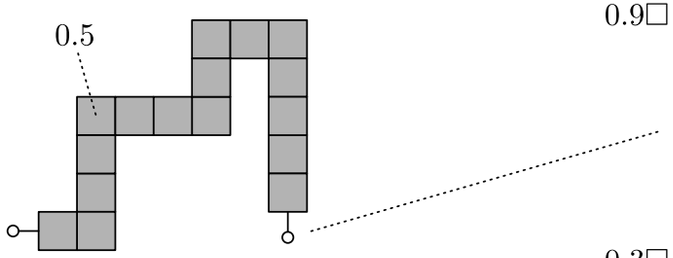


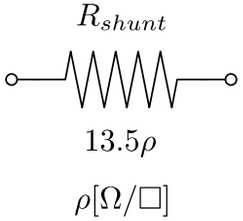
e.g. **glade** and **CNM25**

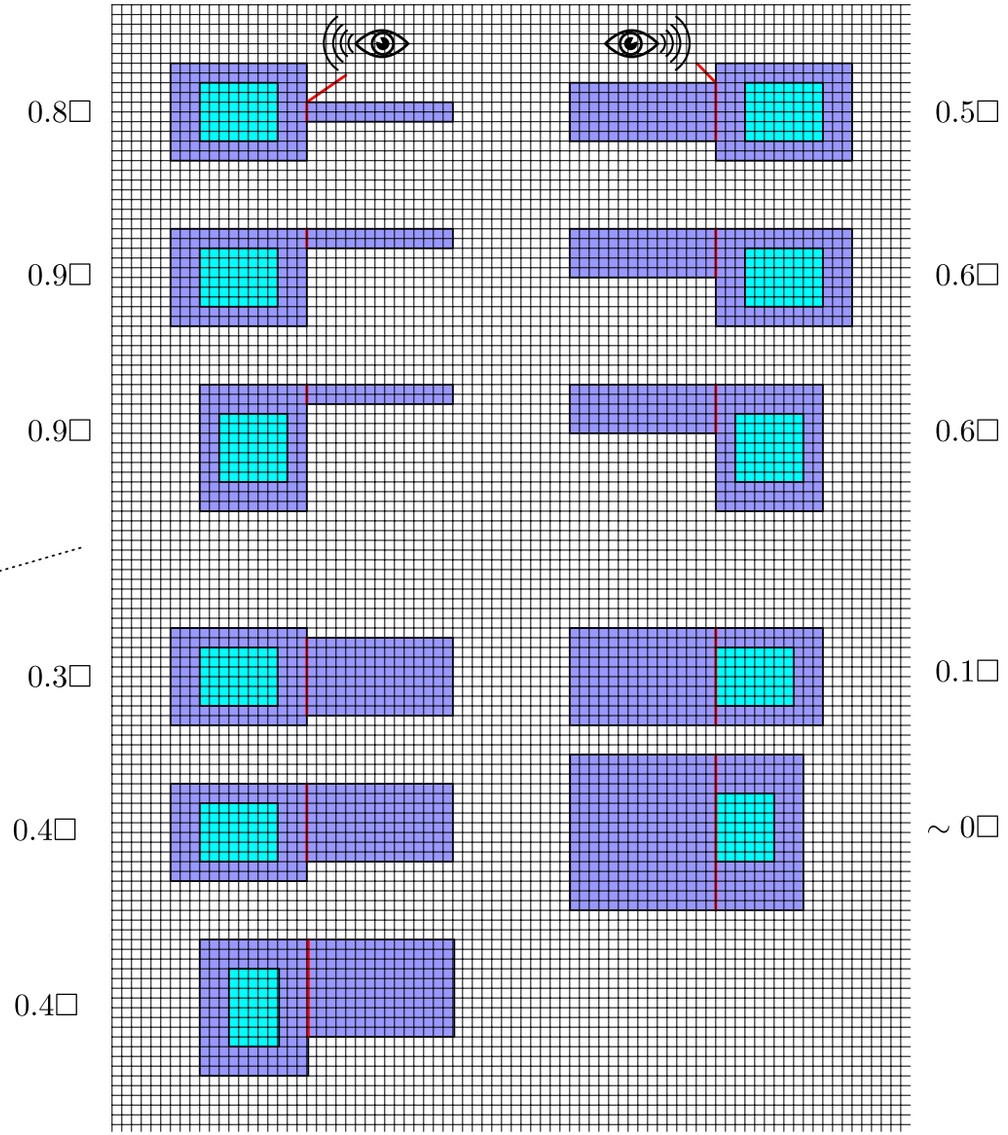
Motivation

- **Planar** technology parasitics

(2D layout view)



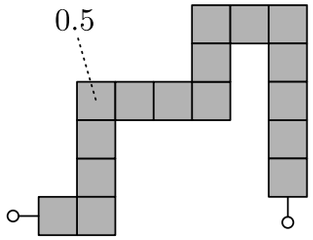




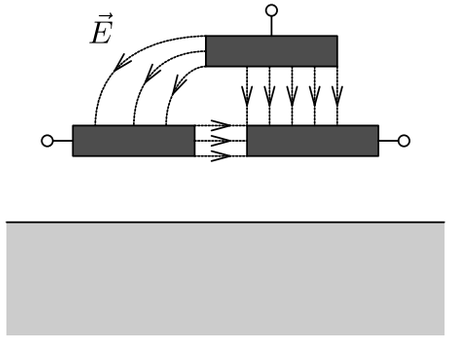
Motivation

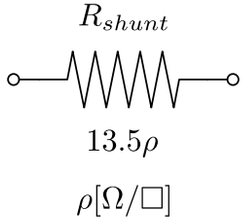
- **Planar** technology parasitics

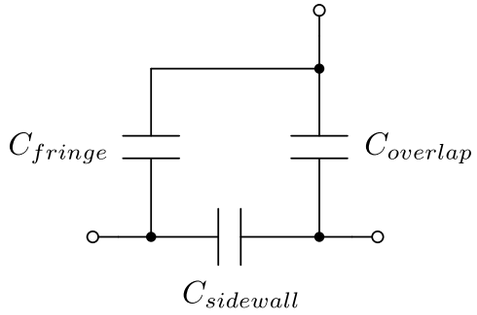
(2D layout view)



(2D cross section view)



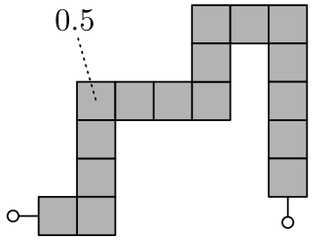




Motivation

- **Planar** technology parasitics
- **R** and **L** require node **splitting**!

(2D layout view)

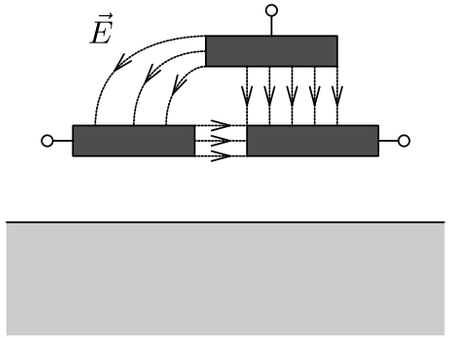


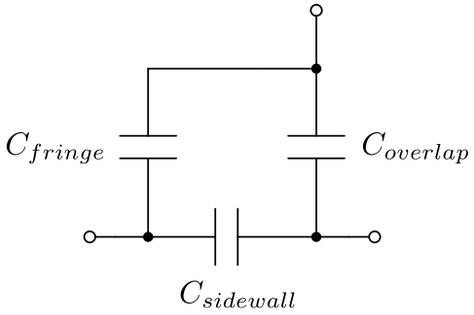
R_{shunt}

13.5ρ

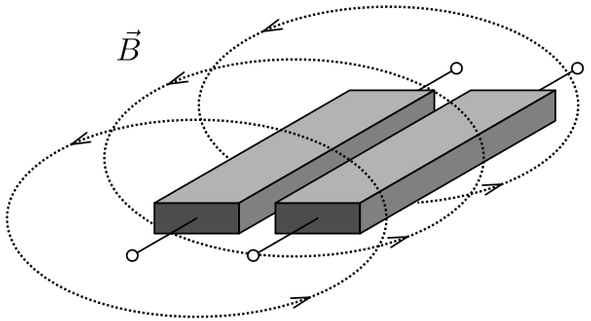
$\rho[\Omega/\square]$

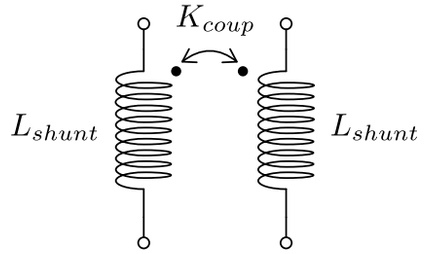
(2D cross section view)





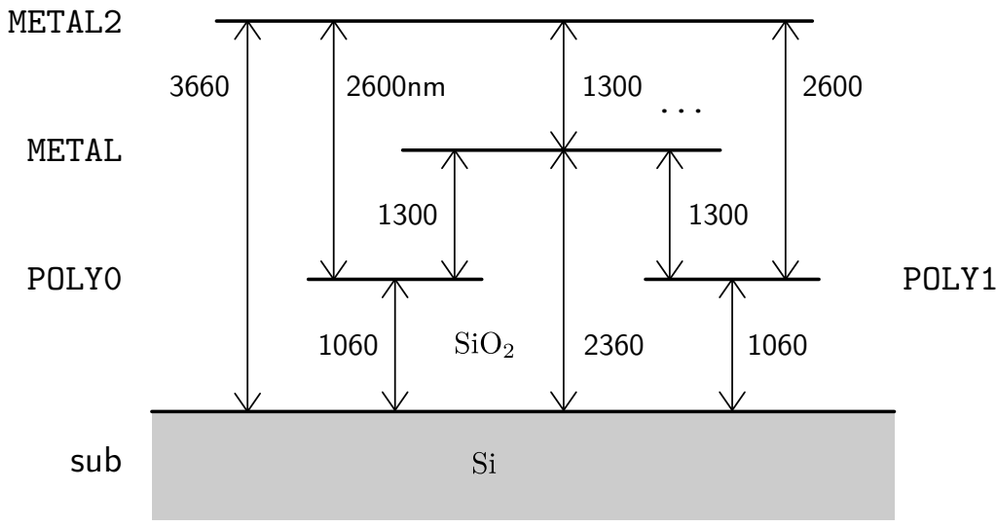
(3D view)





Extraction Tools

- **Programming** a simple rules set for parasitic overlap capacitance only...



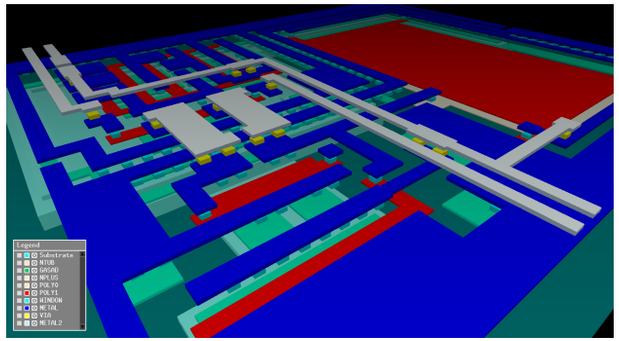
cnm25xtr.py

```
...
geomLabel(polygate, "POLY1", "pin", 1)
geomLabel(polygate, "POLY1", "net", 0)
geomConnect([
  [cont, ndiff, pdiff, polygate, polycap, metal1],
  [via12, metal1, metal2]... ] )
extractMOS("cnm25modn", ngate, polygate, ndiff, pwell)
extractParasitic3(pdiff, metal2, cmetal2diff, 0,
  [metal1, polygate, polycap])
...
```

e.g. 2.5um 2P2M CMOS (**CNM25**)

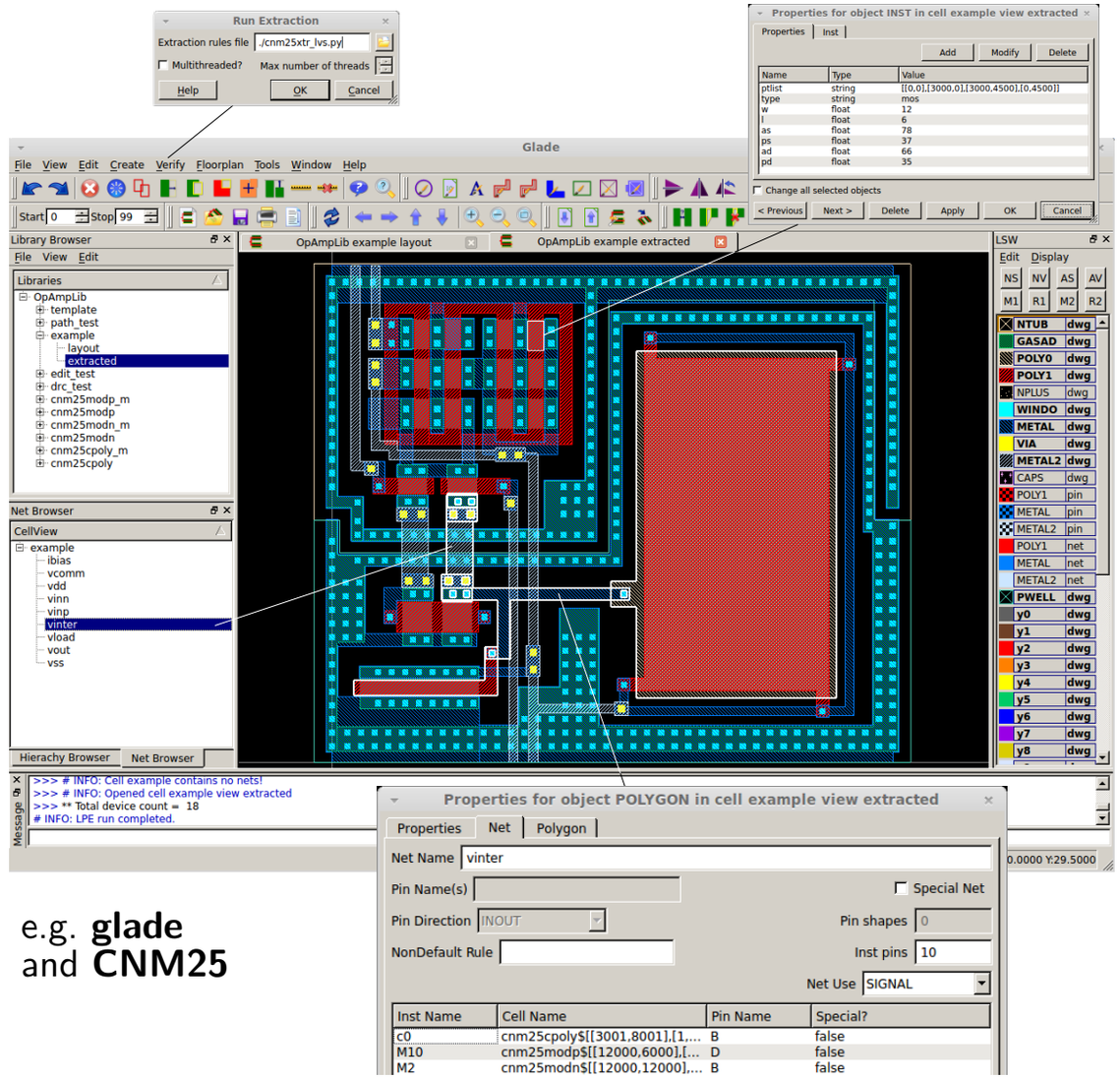
Extraction Tools

- **Programming** a simple rules set for parasitic overlap capacitance only...
- Using assisted **extraction tools**...



opamp_par.sub

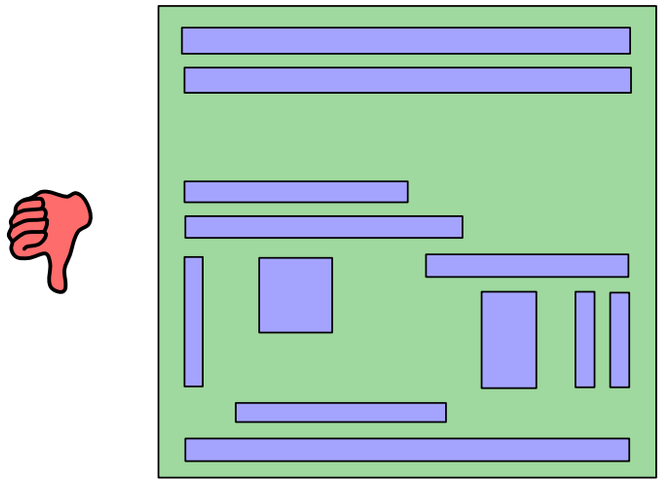
```
.SUBCKT opamp vinn vinp vout vdd vss ibias
MM0 vdd ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=...
MM1 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=...
Cc0 vinter vout cnm25cpoly w=6.42928e-05 l=0.000156207
MM8 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=...
...
CP1 vinter vss  C=3.8582e-13
CP2 vout ibias  C=3.33692e-15
CP3 vinp vss  C=1.85938e-15
CP4 vout vcomm  C=2.0918e-15
...
.ENDS
```



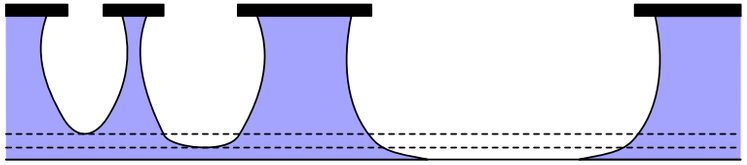
e.g. **glade** and **CNM25**

Going Into Production

- From few IC **prototypes** to small and medium **series**
- Design for manufacturing (**DFM**) layout rules to increase manufacturing **yield**
- Some **examples**:
  - Dummy **filling**

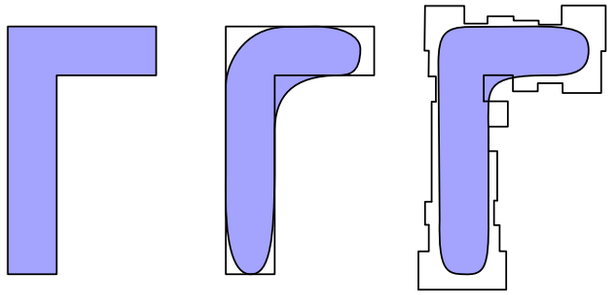


non-uniform layer area = **etching** issues

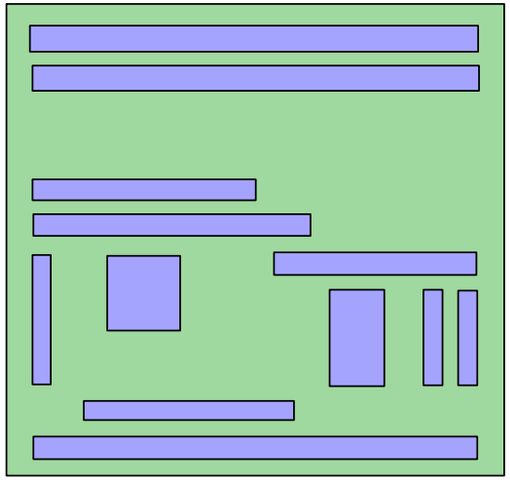


Going Into Production

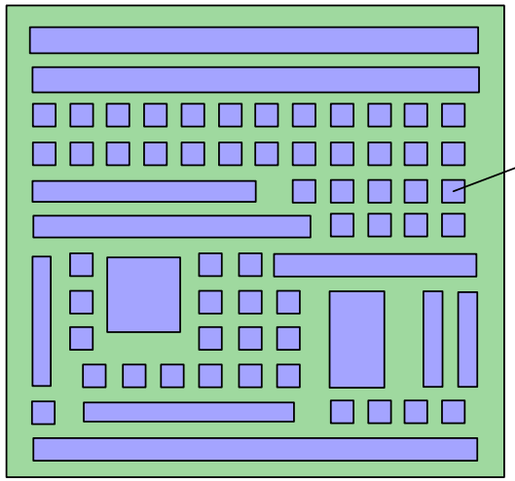
- From few IC **prototypes** to small and medium **series**
- Design for manufacturing (**DFM**) layout rules to increase manufacturing **yield**
- Some **examples**:
  - Dummy **filling**



Using few shapes also simplifies optical proximity correction (**OPC**)



non-uniform layer area = **etching** issues



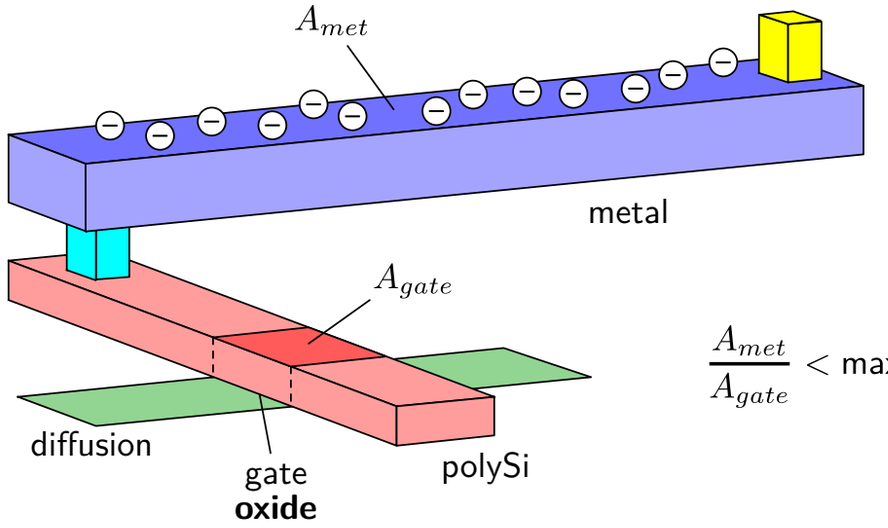
floating **dummy** structures

automatic filling for a target **% coverage**

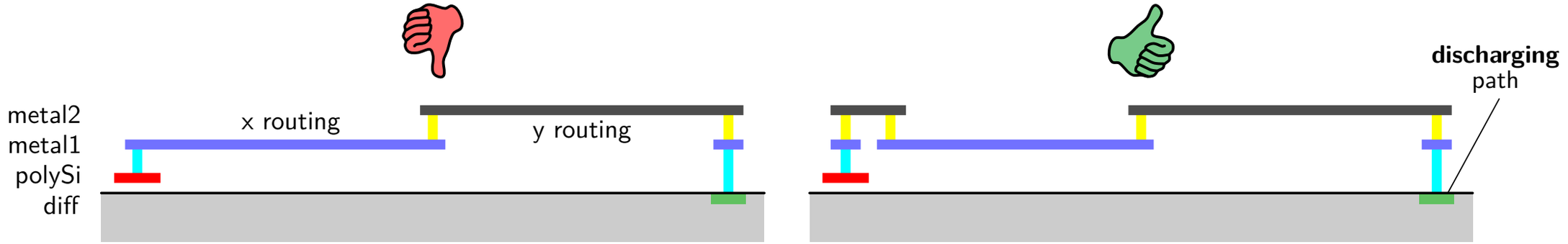
Going Into Production

- From few IC **prototypes** to small and medium **series**
- Design for manufacturing (**DFM**) layout rules to increase manufacturing **yield**
- Some **examples**:
  - Dummy **filling**
  - **Antenna** reduction

charge stored during metal patterning can **break** thin gate oxide:

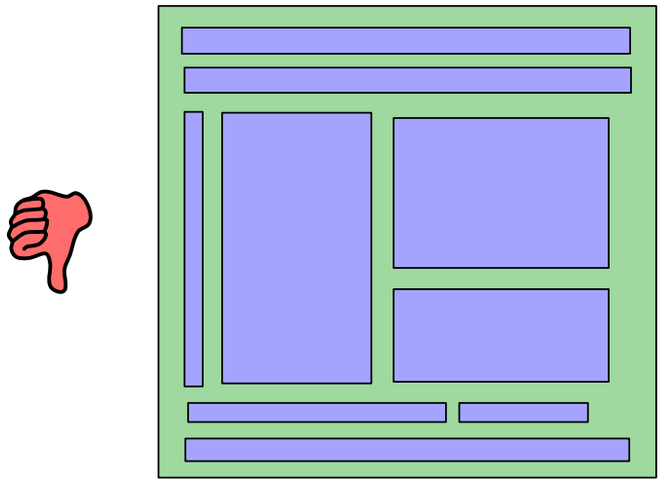


$$\frac{A_{met}}{A_{gate}} < \text{max}$$

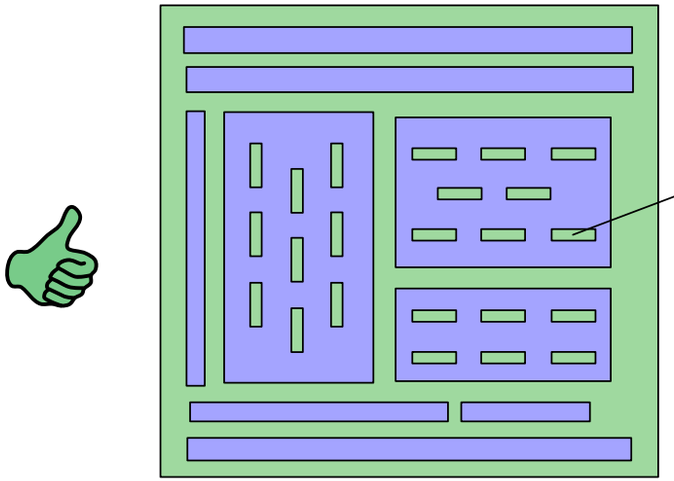


Going Into Production

- From few IC **prototypes** to small and medium **series**
- Design for manufacturing (**DFM**) layout rules to increase manufacturing **yield**
- Some **examples**:
  - Dummy **filling**
  - **Antenna** reduction
  - Metal **slotting**
  - **Multiple** contacts
  - Extra **guard rings**



large metal area = **mechanical** stress issues



stress-relief **holes**