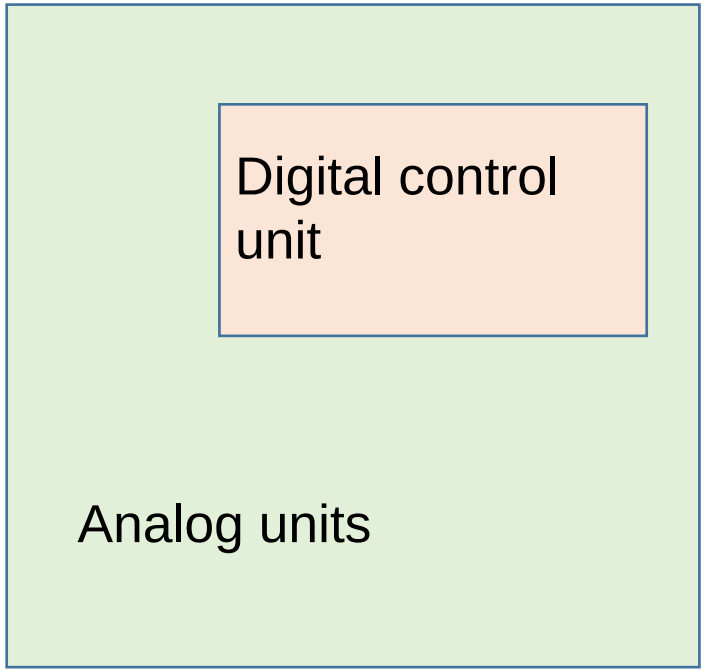
Mixed-Signal Design Flow and Example of SAR ADC Design

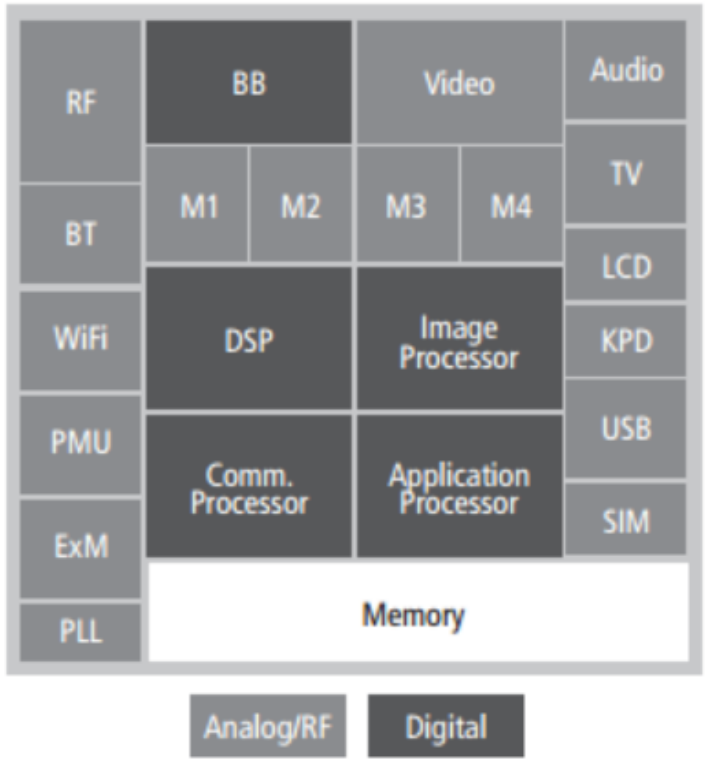
Dr. Alessandro Catania and Dr. Michele Dei

Mixed Signal Design Flow

Traditional mixed circuit system
(e.g. interface for a MEMS sensor)

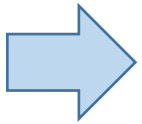


System on a chip with distribution of analog and digital units



Both are examples of Mixed Signal integrated circuits

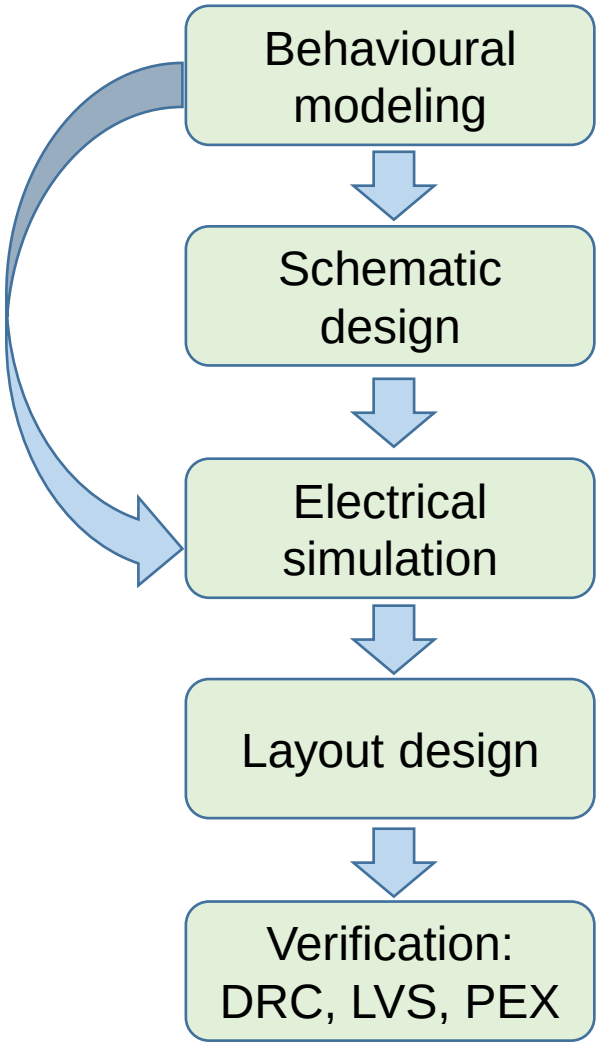
Possible design flows



- **Analog centric (or analog on top):** the analog and digital units are designed with their proper tools and integration is performed using the analog tool.

- **Digital centric (or digital on top):** the analog and digital units are designed with their proper tools and integration is performed using the (highly automated) digital tool.

Analog Design Flow and examples of CAD tools



Matlab/Simulink, Python, HDL (Verilog-A, VHDL-AMS)

Virtuoso schematic editor

Analog Design Environment (ADE)
Spectre electrical simulator

Virtuoso layout editor

Assura

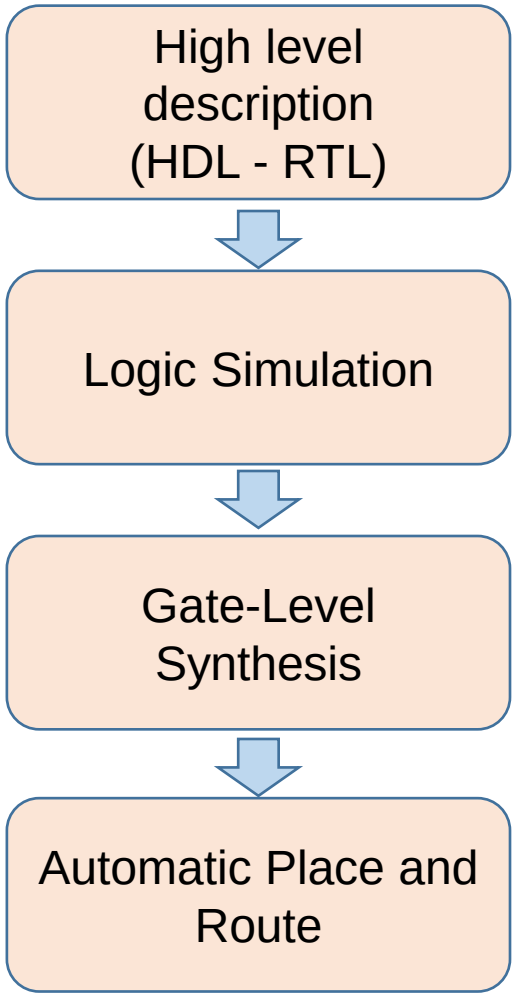
CADENCE

Calibre

Mentor Graphics

PEX: Parasitic extraction (for post-layout simulations)

Digital Design Flow and CADENCE tools



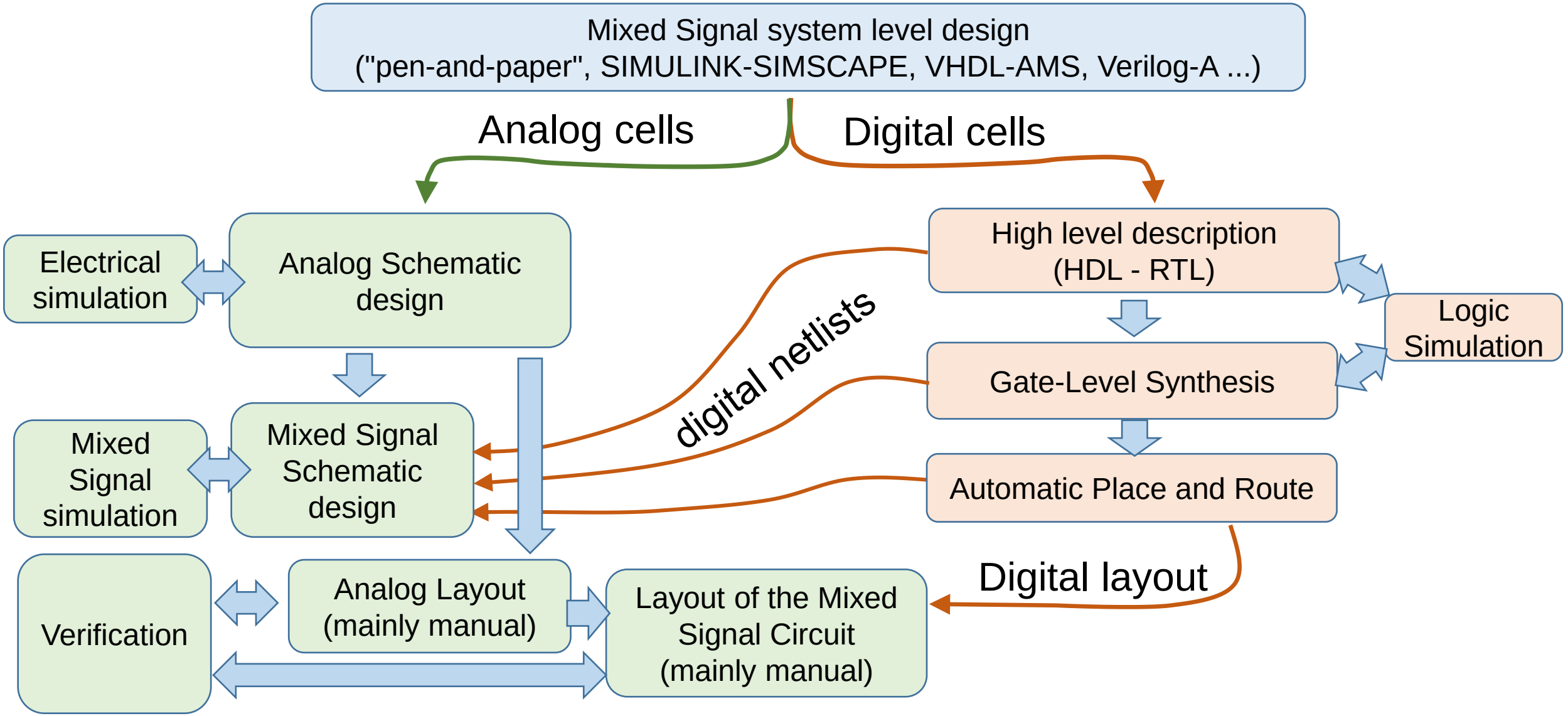
Text Editor (e.g. "Gedit")

CADENCE “Xcelium”

CADENCE “Genus”

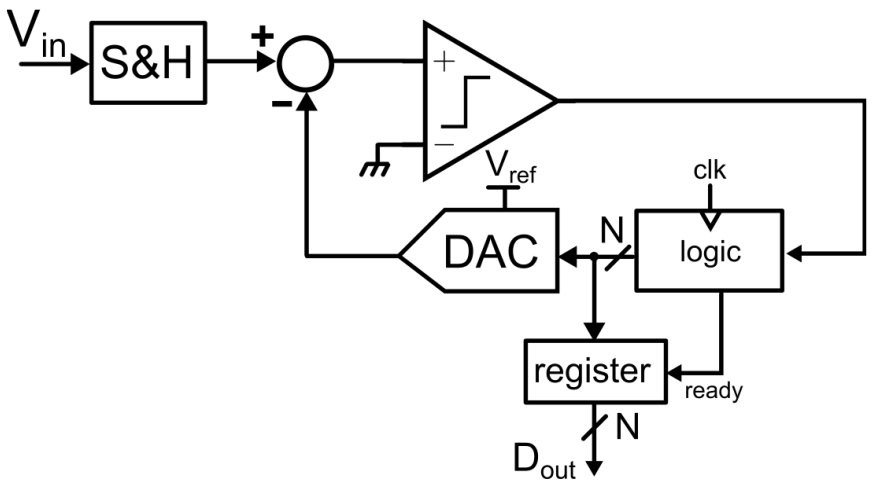
CADENCE “Innovus”

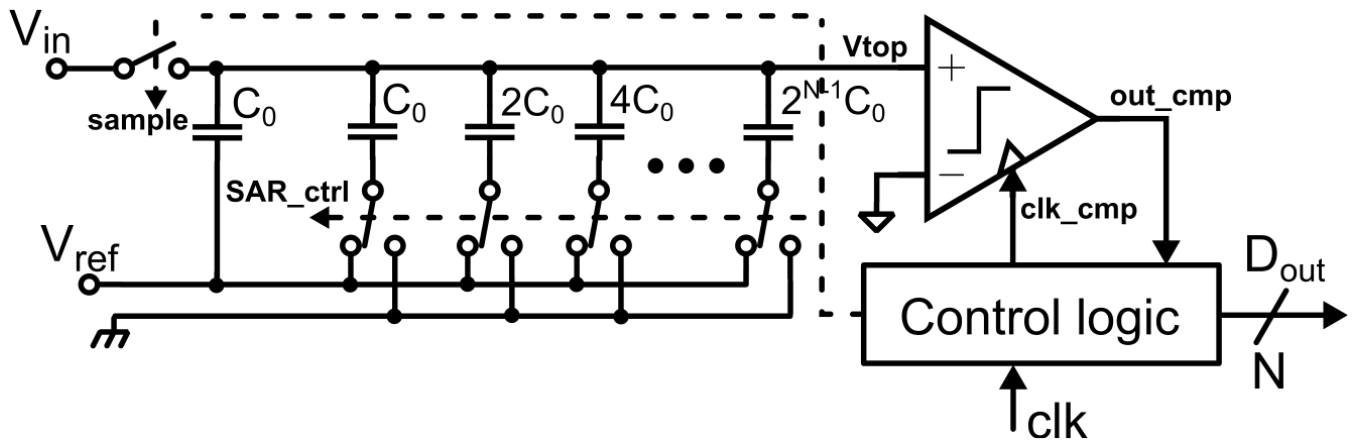
Mixed Signal Design Flow: Analog Centric Approach



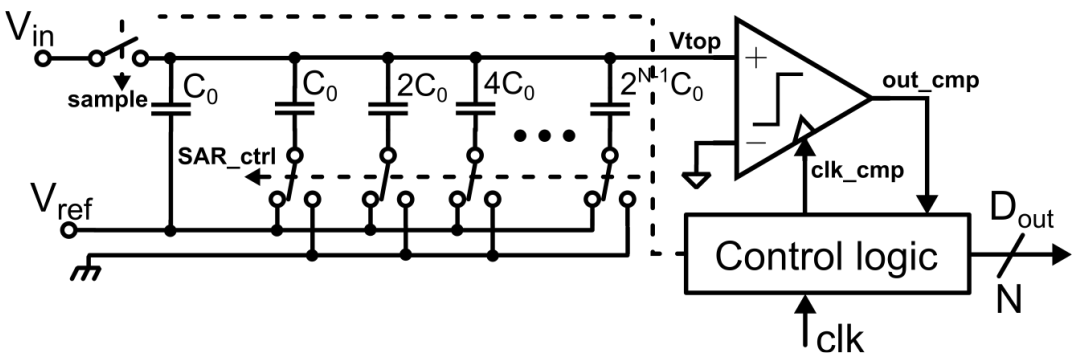
Example: Design of an SAR ADC

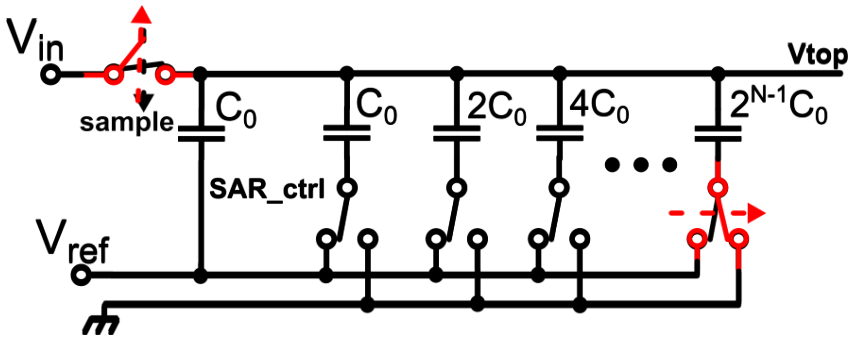
- High-level behavioural description of the SAR algorithm (e.g. MATLAB, python)
- Transistor-level design and simulations of the comparator and the DAC
- HDL description and digital simulations of the SAR control logic
- Mixed-signal simulations of the whole ADC
- Layout of the analog blocks / synthesis and place-and-route of the SAR logic



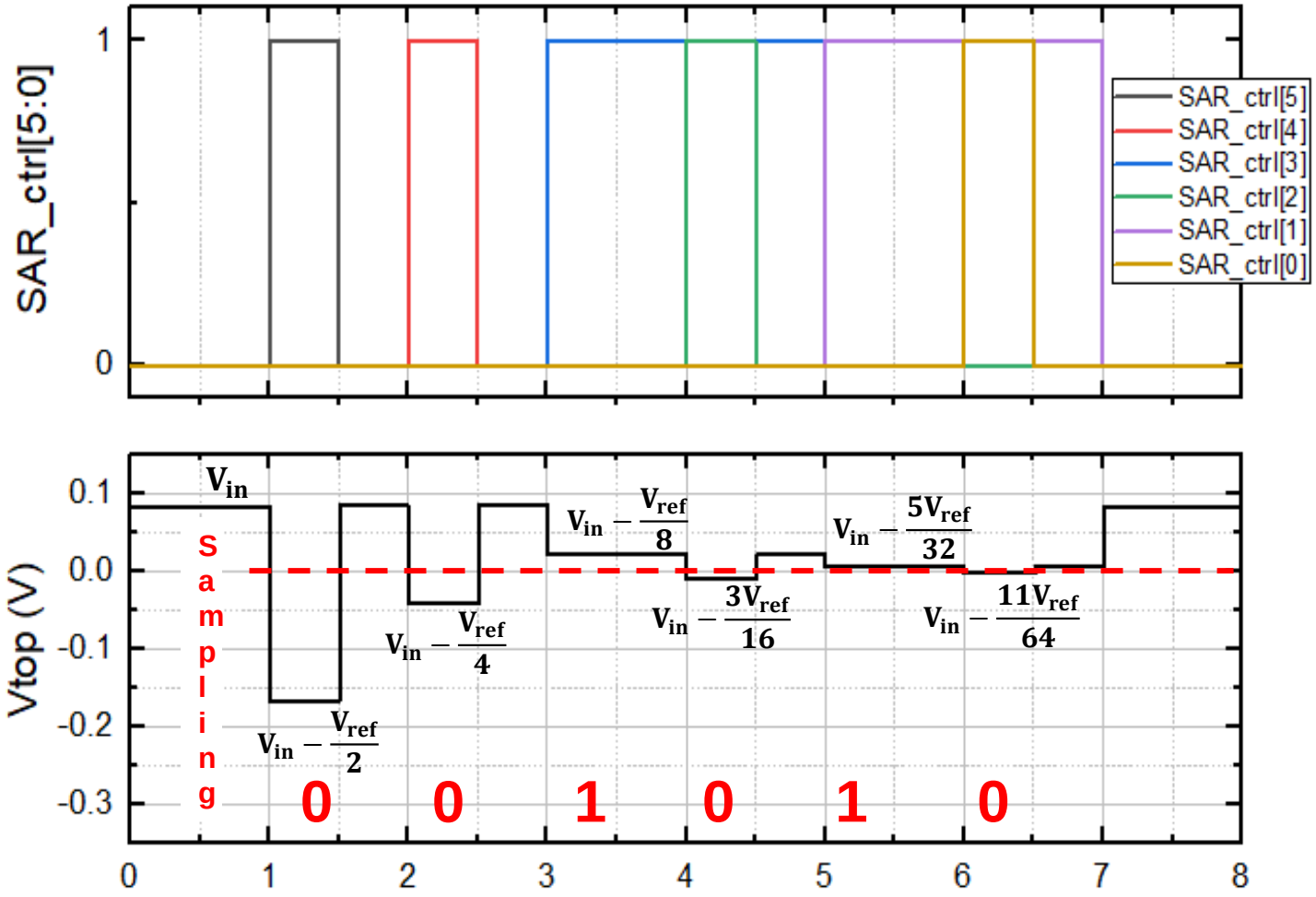


Successive Approximation Register ADC (6 bit)





$$V_{top} = V_{in} \Rightarrow \Delta V_{top} = \Delta V \frac{2^{N-1}C_0}{C_{tot}} = -V_{ref} \frac{2^{N-1}C_0}{2^N C_0} = -\frac{V_{ref}}{2}$$



V_{in} = 82.7 mV

V_{ref} = 500 mV

A/D (N = 6)

D_{out}=001010
(10 in decimale)

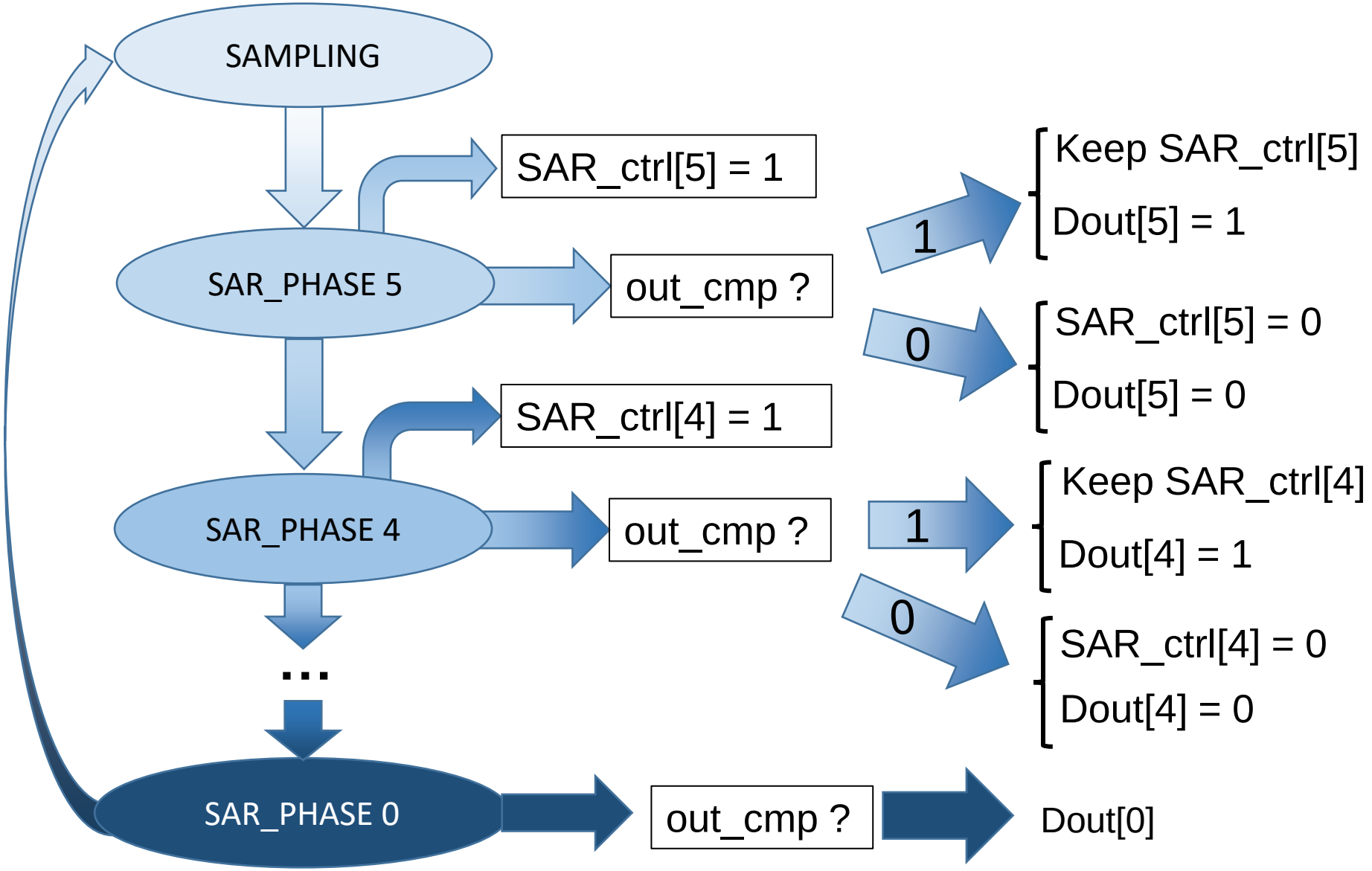
Behavioural description of the SAR algorithm

SAR algorithm:

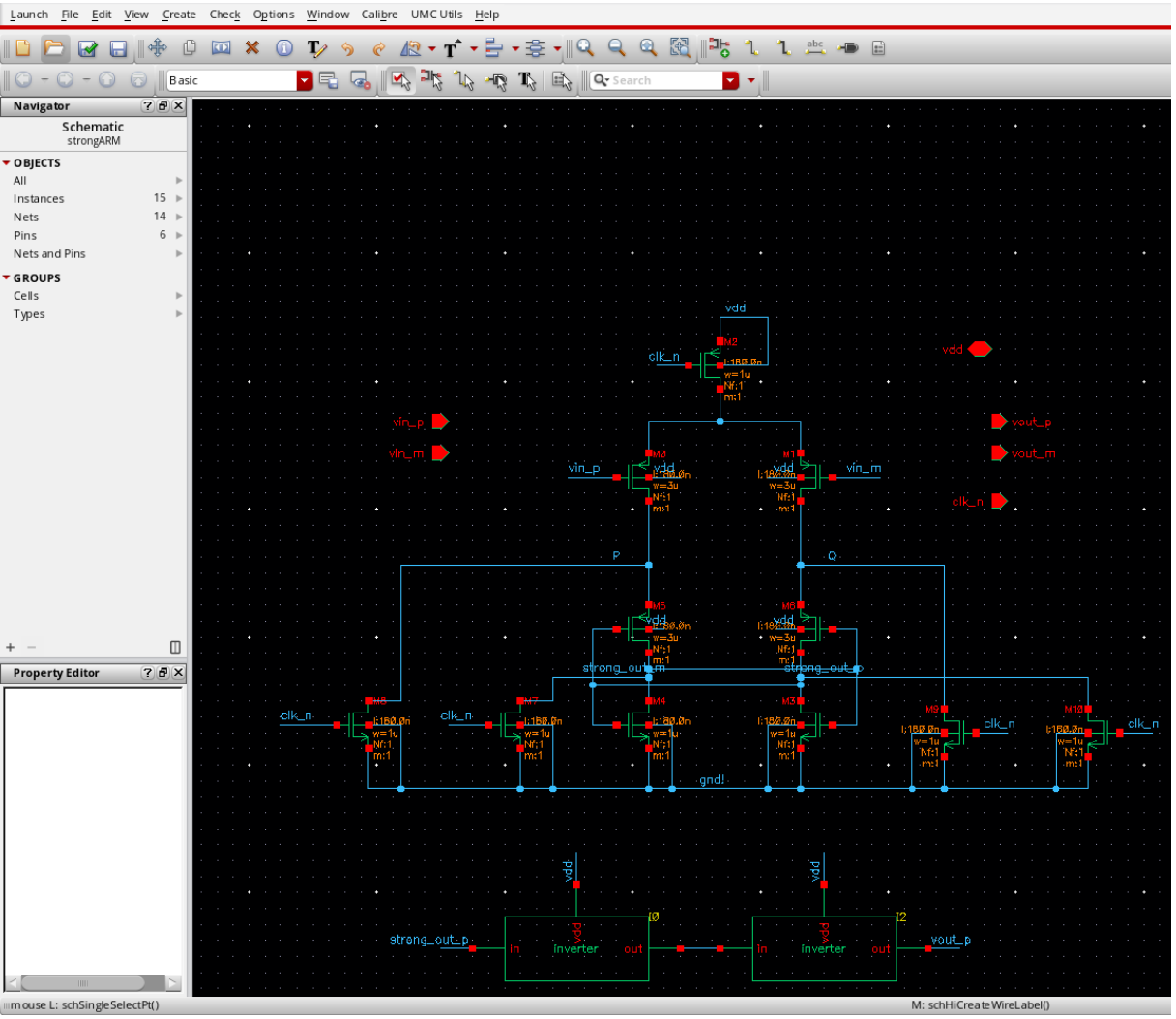
- Signal sampling
- SAR conversion (N phases)

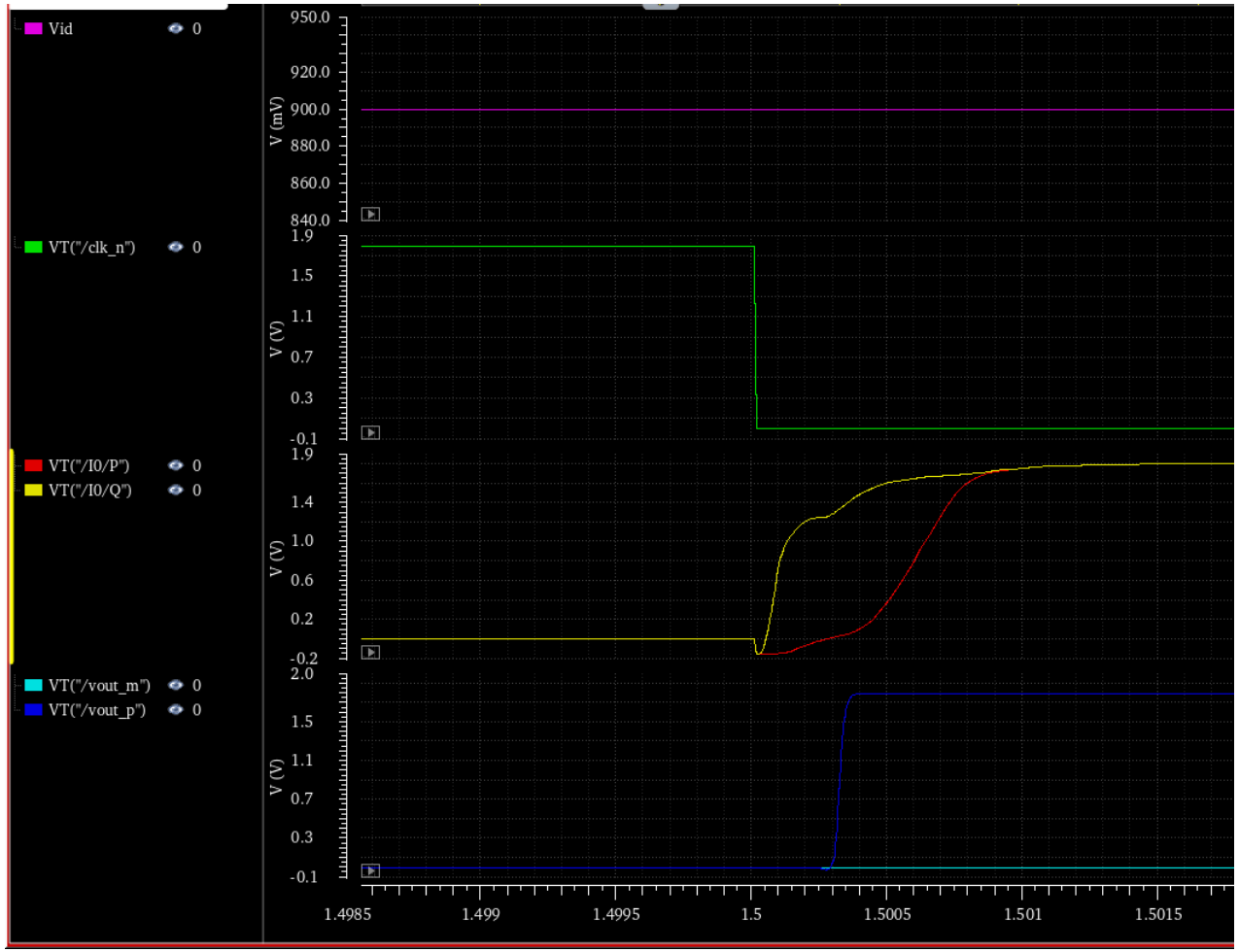
SAR phase:

1. DAC update (initial guess)
2. Verify the comparator output
3. DAC correction (if needed)



Analog Design and Simulations (Spectre simulator)





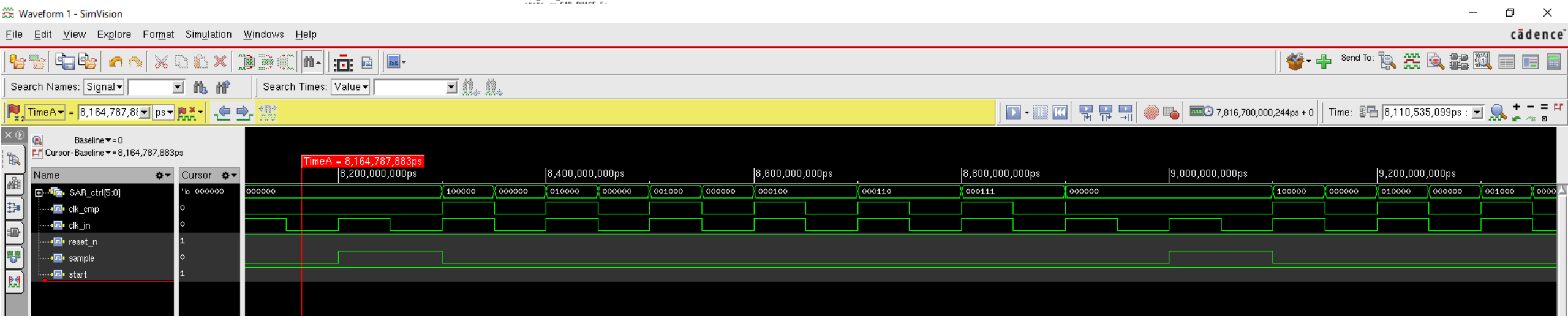
VHDL description of the SAR digital control block

VHDL Code

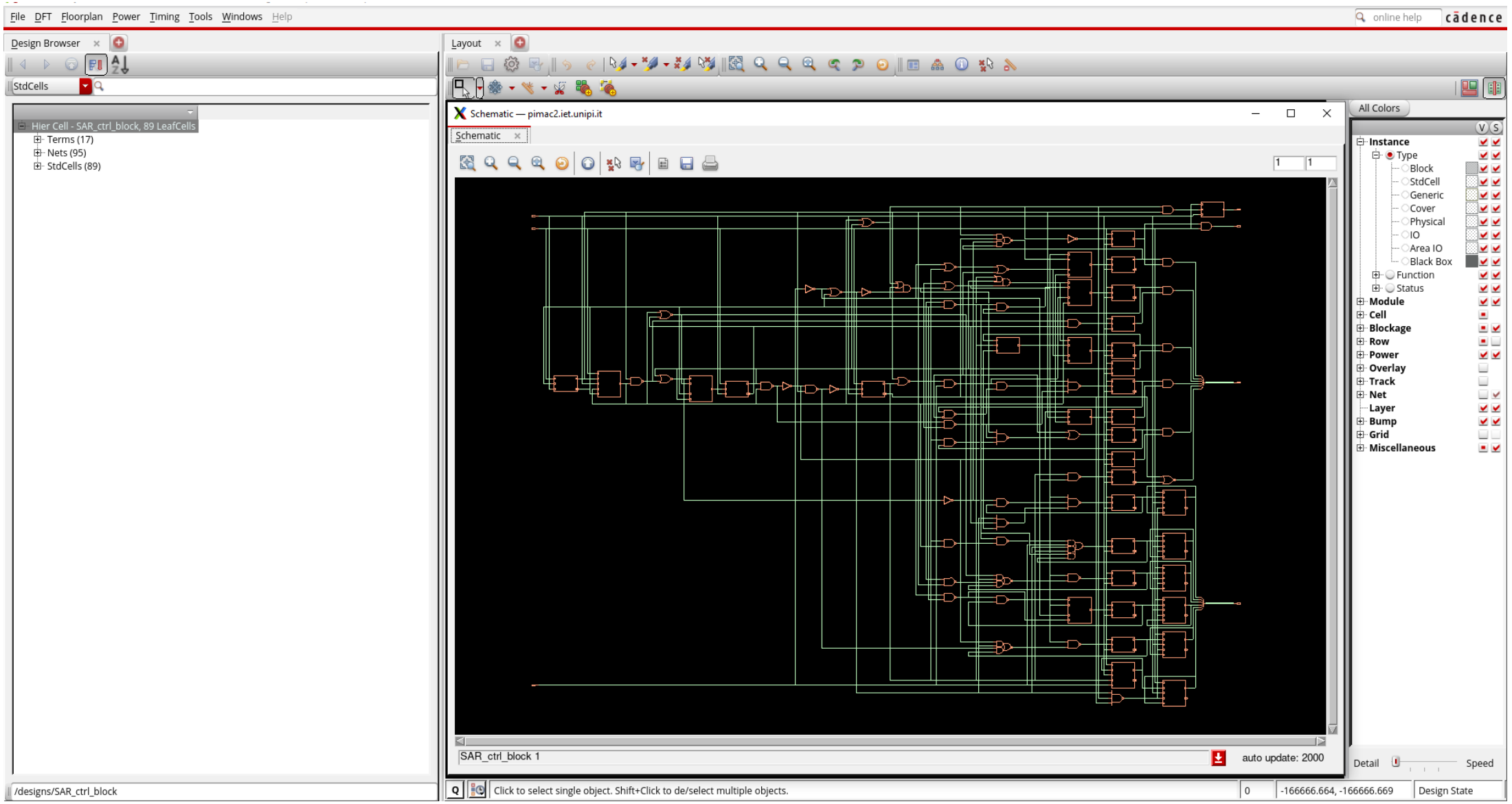
SAR_ctrl_block.vhd

```
library IEEE;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.std_logic_1164.all;

entity SAR_ctrl_block is
        port(
                clk_in: in std_logic;
                cmp_out: in std_logic;
                reset_n: in std_logic;
                sample: out std_logic;
                clk_cmp: out std_logic;
                SAR_ctrl: out std_logic_vector (5 downto 0);
                output_word: out std_logic_vector (5 downto 0)
        );
end SAR_ctrl_block;

architecture fsm of SAR_ctrl_block is

        type T_STATE is (RESET, SAMPLING, SAR_PHASE_5, SAR_PHASE_4, SAR_PHASE_3, SAR_PHASE_2, SAR_PHASE_1, SAR_PHASE_0, FINISH);
        signal state: T_STATE := RESET;
        signal aux_word: std_logic_vector (5 downto 0) := (others => '0');
        signal SAR_ctrl_1:std_logic_vector (5 downto 0) := (others => '1');
        signal SAR_ctrl_2:std_logic_vector (5 downto 0) := (others => '1');
        signal aux_clk:std_logic := '0';
        signal clk_aux:std_logic := '0';
        begin

                SAR_ctrl <= SAR_ctrl_1 and SAR_ctrl_2;
                clk_aux <= clk_in and aux_clk;
                clk_cmp <= inertial clk_aux after 1ns;

                process (clk_in, reset_n)
                        begin
                                if (reset_n='0') then
                                        sample <= '0';
                                        SAR_ctrl_1 <= (others => '0');
                                        state <= RESET;
                                        aux_clk <= '0';
                                elsif rising_edge(clk_in) then
                                        case state is
                                                when RESET =>  sample <= '0';
                                                               SAR_ctrl_1 <= (others => '0');
                                                               state <= SAMPLING;
                                                               aux_clk <= '0';
                                                when SAMPLING => sample <= '1';
                                                               SAR_ctrl_1 <= (others => '0');
```

Digital simulations (XCELIUM)



Digital synthesis (GENUS)



Digital synthesis (GENUS)

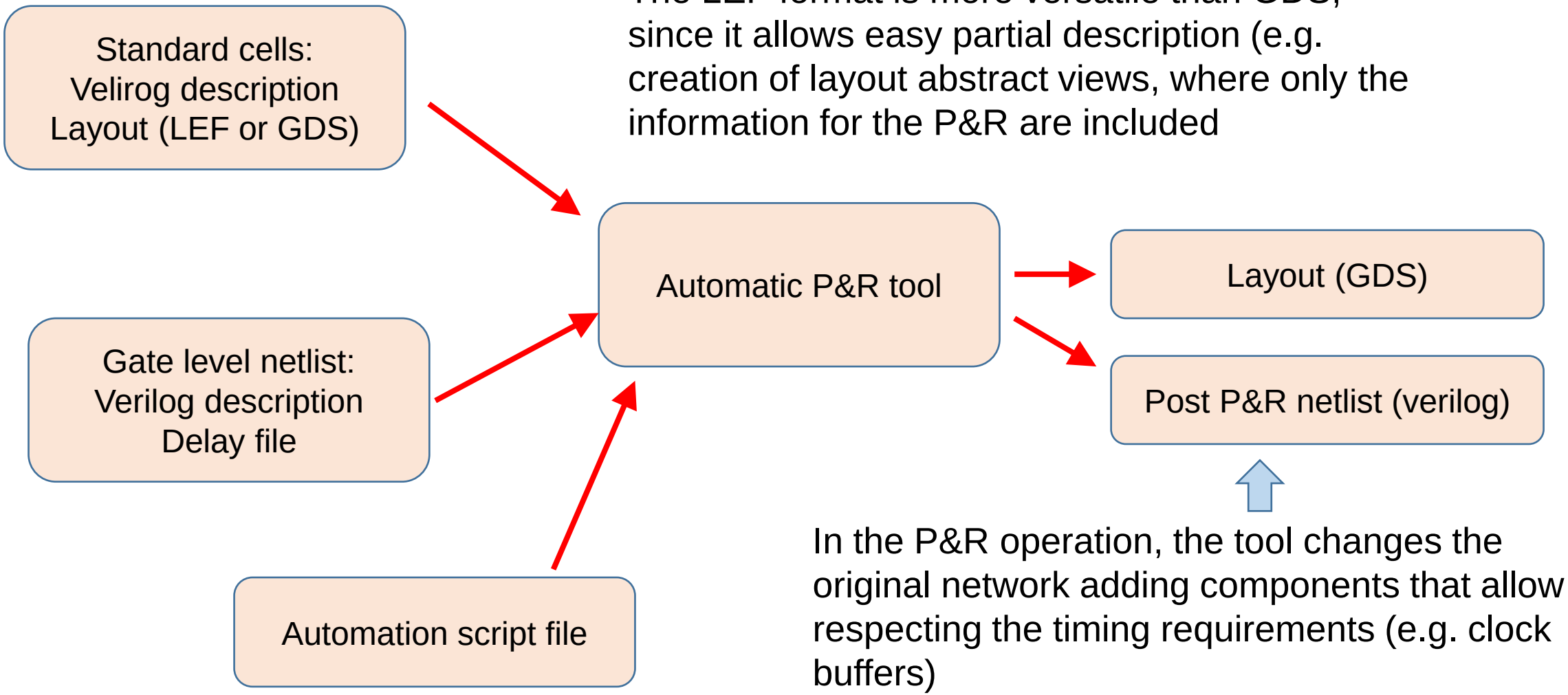
RTL (Register Transfer Level) Description
(GENUS input)

```
library IEEE;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.std_logic_1164.all;

entity SAR_ctrl_block is
        port(
                clk_in: in std_logic;
                cmp_out: in std_logic;
                reset_n: in std_logic;
                sample: out std_logic;
                clk_cmp: out std_logic;
                SAR_ctrl: out std_logic_vector (5 downto 0);
                output_word: out std_logic_vector (5 downto 0)
        );
end SAR_ctrl_block;

architecture fsm of SAR_ctrl_block is

        type T_STATE is (RESET, SAMPLING, SAR_PHASE_5, SAR_PHASE_4, SAR_PHASE_3, SAR_PHASE_2, SAR_PHASE_1, SAR_PHASE_0, FINISH);
        signal state: T_STATE := RESET;
        signal aux_word: std_logic_vector (5 downto 0) := (others => '0');
        signal SAR_ctrl_1:std_logic_vector (5 downto 0) := (others => '1');
        signal SAR_ctrl_2:std_logic_vector (5 downto 0) := (others => '1');
        signal aux_clk:std_logic := '0';
        signal clk_aux:std_logic := '0';
        begin

                SAR_ctrl <= SAR_ctrl_1 and SAR_ctrl_2;
                clk_aux <= clk_in and aux_clk;
                clk_cmp <= inertial clk_aux after 1ns;

                process (clk_in, reset_n)
                        begin
                                if (reset_n='0') then
                                        sample <= '0';
                                        SAR_ctrl_1 <= (others => '0');
                                        state <= RESET;
                                        aux_clk <= '0';
                                elsif rising_edge(clk_in) then
                                        case state is
                                                when RESET =>  sample <= '0';
                                                               SAR_ctrl_1 <= (others => '0');
                                                               state <= SAMPLING;
                                                               aux_clk <= '0';
                                                when SAMPLING => sample <= '1';
                                                               SAR_ctrl_1 <= (others => '0');
                                                               state <= SAR_PHASE_5;
```

Gate Level Description
(GENUS output)

```
module SAR_ctrl_block(clk_in, cmp_out, reset_n, sample, clk_cmp,
     SAR_ctrl, output_word);
  input clk_in, cmp_out, reset_n;
  output sample, clk_cmp;
  output [5:0] SAR_ctrl, output_word;
  wire clk_in, cmp_out, reset_n;
  wire sample, clk_cmp;
  wire [5:0] SAR_ctrl, output_word;
  wire [5:0] SAR_ctrl_2;
  wire [5:0] aux_word;
  wire [3:0] state;
  wire [5:0] SAR_ctrl_1;
  wire UNCONNECTED, UNCONNECTED0, UNCONNECTED1, UNCONNECTED2,
       UNCONNECTED3, UNCONNECTED4, UNCONNECTED5, UNCONNECTED6;
  wire UNCONNECTED7, UNCONNECTED8, UNCONNECTED9, UNCONNECTED10,
       UNCONNECTED11, UNCONNECTED12, UNCONNECTED13, UNCONNECTED14;
  wire UNCONNECTED15, UNCONNECTED16, n_0, n_1, n_2, n_3, n_5, n_6;
  wire n_7, n_8, n_9, n_10, n_11, n_12, n_13, n_14;
  wire n_15, n_16, n_18, n_19, n_20, n_21, n_23, n_24;
  wire n_27, n_29, n_31, n_32, n_33, n_34, n_35, n_36;
  wire n_37, n_38, n_39, n_40, n_41, n_42, n_43, n_44;
  wire n_45, n_46, n_47, n_48, n_50, n_51, n_52, n_53;
  wire n_55, n_56, n_57, n_58, n_59, n_60, n_61, n_62;
  wire n_63, n_64, n_68;
  DBFRBN \SAR_ctrl_2_reg[2] (.RB (reset_n), .CKB (clk_in), .D (n_64),
       .Q (SAR_ctrl_2[2]), .QB (UNCONNECTED));
  DBFRBN \SAR_ctrl_2_reg[4] (.RB (reset_n), .CKB (clk_in), .D (n_62),
       .Q (SAR_ctrl_2[4]), .QB (UNCONNECTED0));
  DBFRBN \SAR_ctrl_2_reg[3] (.RB (reset_n), .CKB (clk_in), .D (n_60),
       .Q (SAR_ctrl_2[3]), .QB (UNCONNECTED1));
  DBFRBN \aux_word_reg[4] (.RB (reset_n), .CKB (clk_in), .D (n_58), .Q
       (aux_word[4]), .QB (UNCONNECTED2));
  DBFRBN \SAR_ctrl_2_reg[1] (.RB (reset_n), .CKB (clk_in), .D (n_63),
       .Q (SAR_ctrl_2[1]), .QB (UNCONNECTED3));
  ND3S g1595__2398(.I1 (n_56), .I2 (n_51), .I3 (n_52), .O (n_64));
  DBFRBN \SAR_ctrl_2_reg[5] (.RB (reset_n), .CKB (clk_in), .D (n_57),
       .Q (SAR_ctrl_2[5]), .QB (UNCONNECTED4));
  OR2S g1601__5107(.I1 (n_42), .I2 (n_53), .O (n_63));
  OAI112HS g1590__6260(.A1 (n_59), .B1 (n_21), .C1 (n_50), .C2
       (state[1]), .O (n_62));
  DBZRBN \output_word_reg[1] (.RB (reset_n), .CKB (clk_in), .D
       (aux_word[1]), .TD (output_word[1]), .SEL (n_61), .Q
       (output_word[1]), .QB (UNCONNECTED5));
```

Automatic layout generation (Automatic Place and Route, P&R)

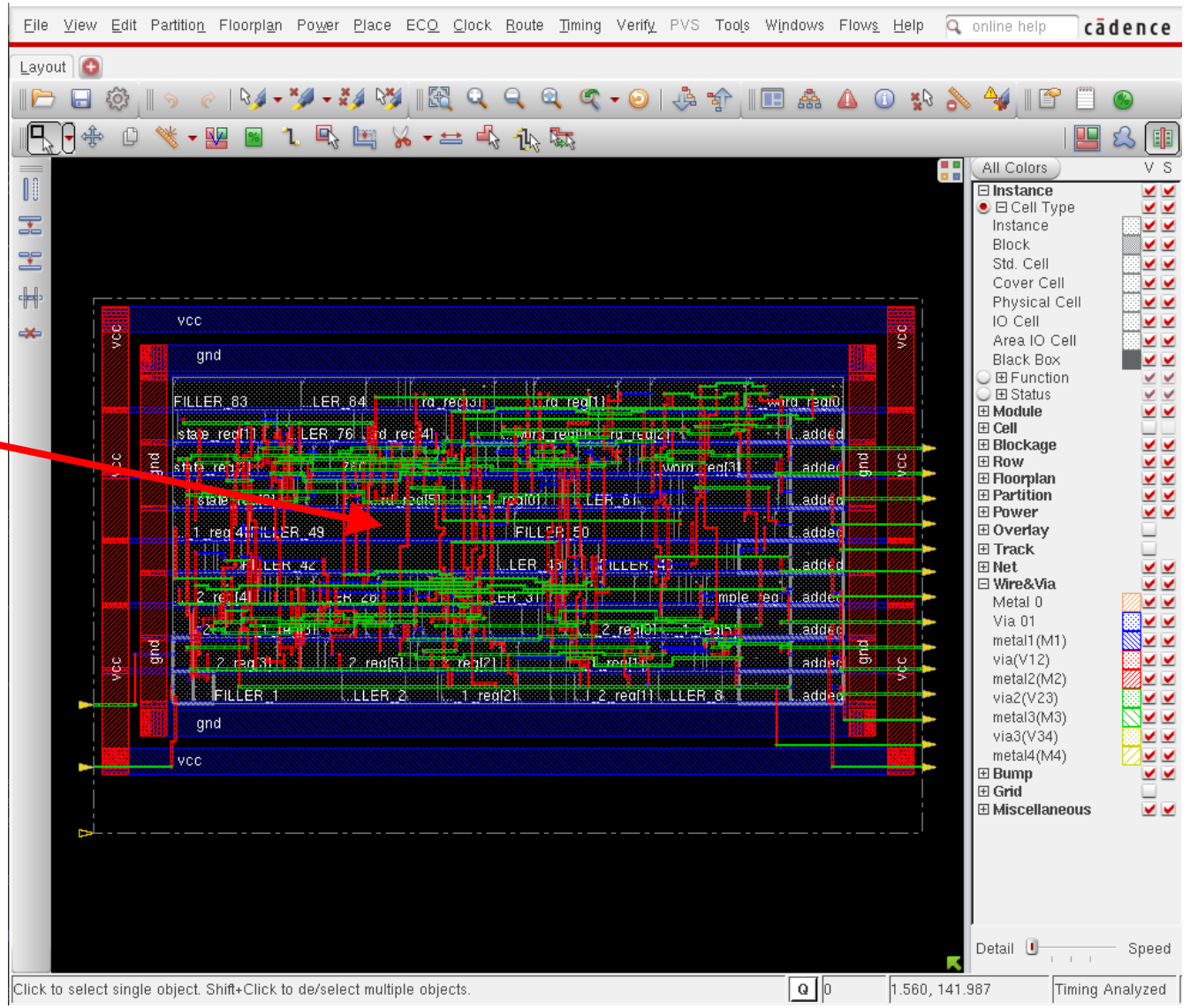


The LEF format is more versatile than GDS, since it allows easy partial description (e.g. creation of layout abstract views, where only the information for the P&R are included

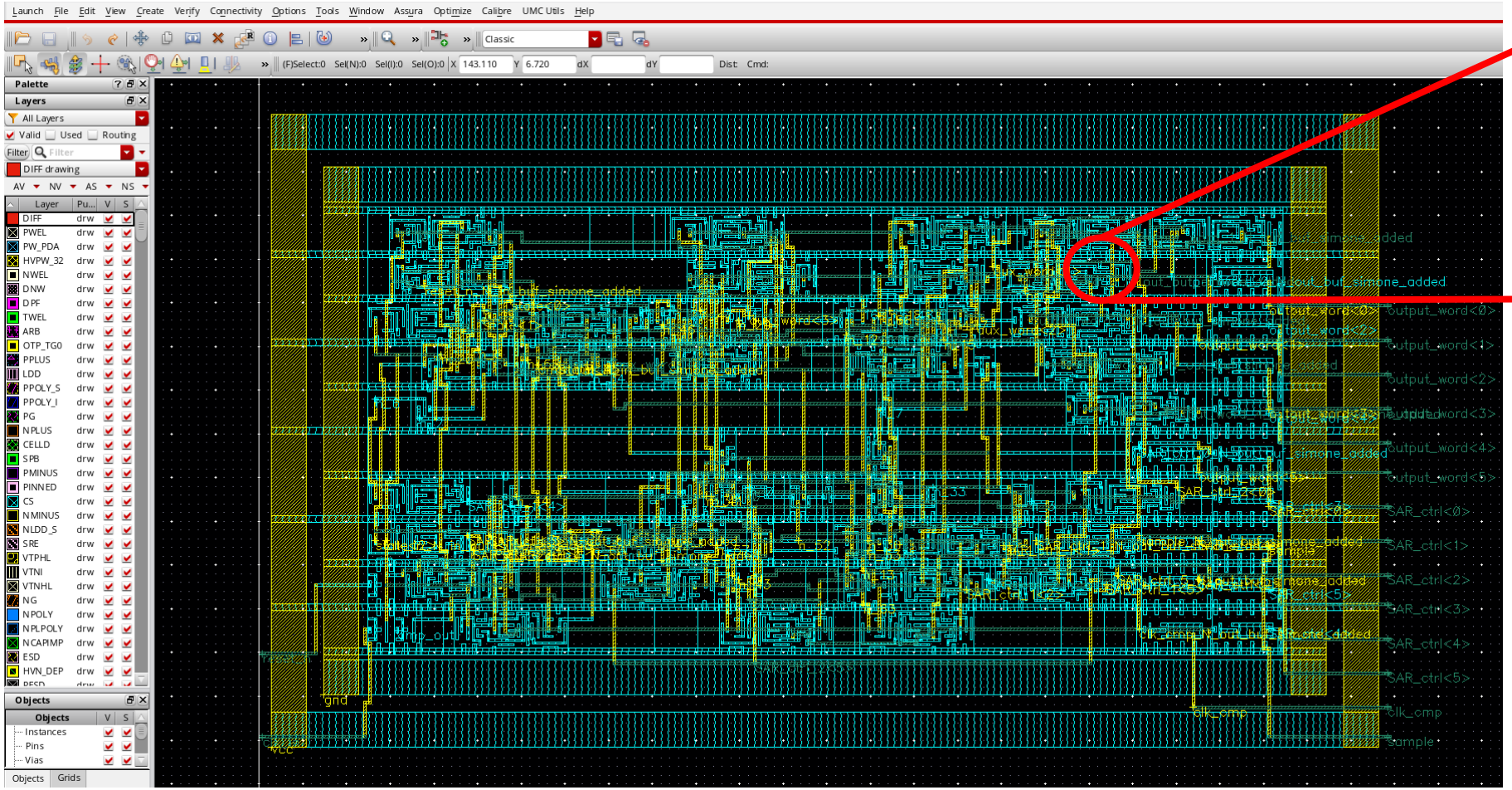
In the P&R operation, the tool changes the original network adding components that allow respecting the timing requirements (e.g. clock buffers)

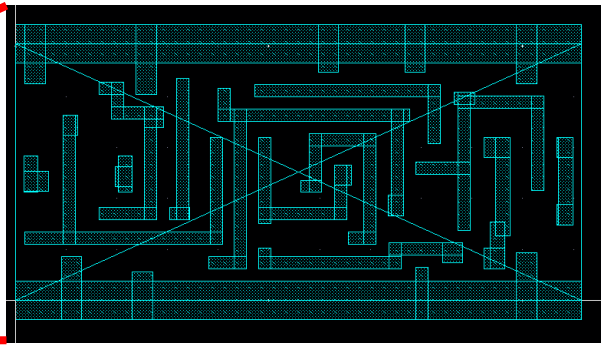
Automatic Place-and-Route (INNOVUS)

Empty spaces between cells are covered by "filler" elements in the final layout. Fillers include n-wells and other layers that improve continuity between cells



Layout imported in Virtuoso

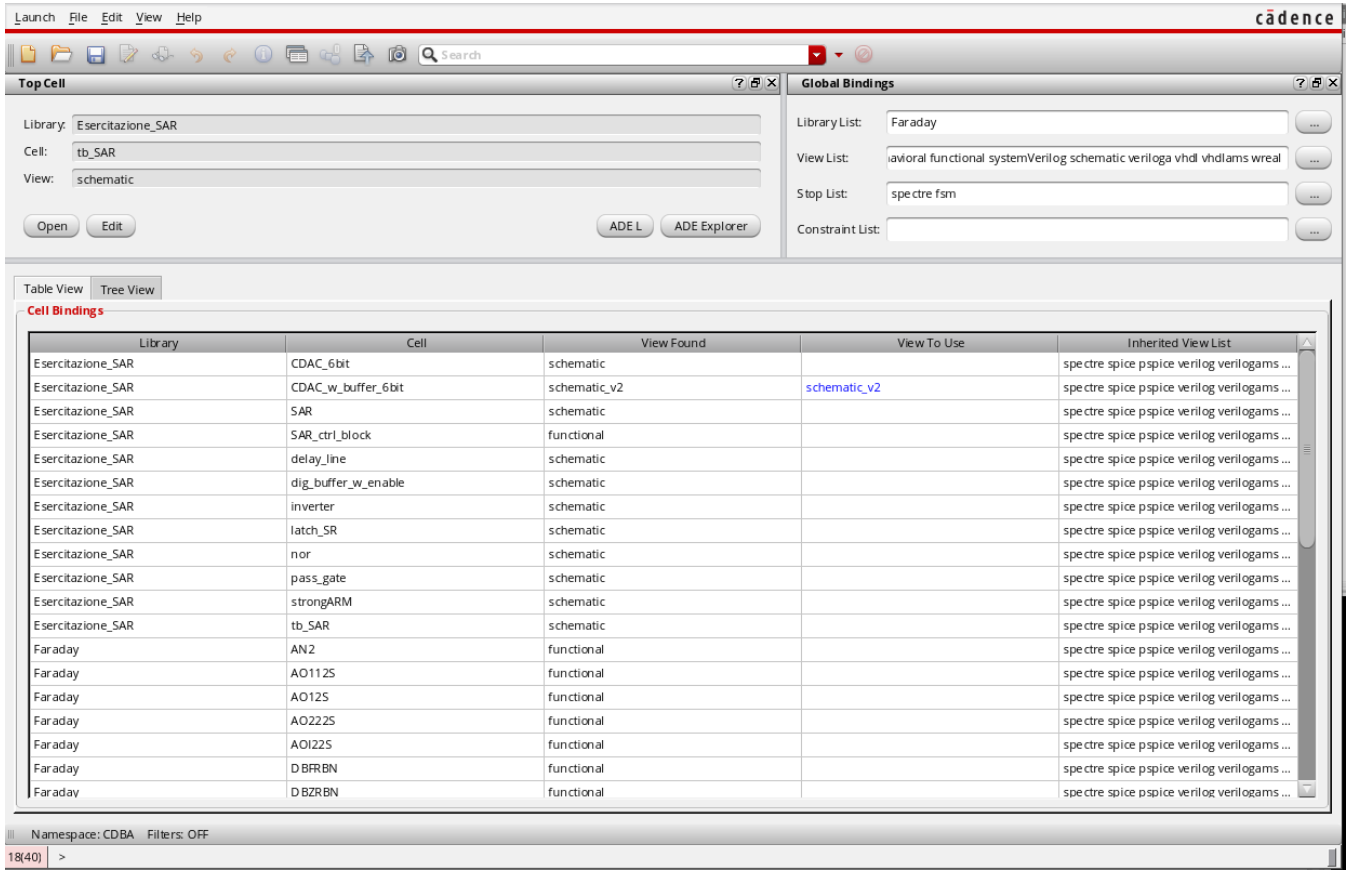




Abstract view of
D Flip-Flop

Mixed-Signal Simulations (ams simulator)

Config view of the testbench



Connect rules

File Edit View Help cadence

```
`define CONNRULES_18V_FULL_FAST
`define CONNRULES_18V_FULL
`define CONNRULES_18V_MID
`define CONNRULES_18V_BASIC
`define CONNRULES_FULL_FAST
`define CONNRULES_FULL
`define CONNRULES_MID
`define CONNRULES_BASIC
`define Vsup  1.8
`define Vthi  1.2
`define Vtlo  0.6
`define Vtrhi `Vthi/`Vsup
`define Vtrlo `Vtlo/`Vsup
`define Vlow  0
`define Tr    0.2n
`define Rlo   200
`define Rhi   200
`define Rx    40
`define Rz    10M
`define Vdelta      `Vsup/64
`define Vdelta_tol  `Vdelta/4
`define Tr_delta    `Tr/20
`define rsupply     4
`define rpull       1.5e3
`define rlarge      9.0e3
`define rweak       5.5e4
`define rmedium     3.2e5
`define rsmall      1.9e6
```

Mixed-Signal Simulations