Overview

Tacking hardware security problems requires a thorough knowledge in the areas of:

- Computer-Aided Design (VLSI) from soft IP to layout
- Manufacturing process variations and testing methods
- FPGA and embedded sytem design
- Hardware encryption algorithms, key exchange mechanisms

Hardware security threats as they relate to ASICs and FPGAs are distinct under each of these classes of devices

Although there is some commonality, e.g., hardware encryption mechanisms and key storage issues, other issues such as Hardware Trojans are different

In order to appreciate these differences, we need first to understand their associated design flows

It is impossible to cover the details of these design flows because of their complexity -- this lecture instead serves as an overview

The Nature of Hardware and Software

Let's first consider the trade-offs of building a function in hardware (HW) versus using a software (SW) implementation

Choosing between implementing a design in HW or implementing it in SW may seem like a no-brainer -- clearly writing software is easier!

Proponents of **hardware** implementation will argue that performance is a big plus of hardware (an ASIC) over software (running on a microprocessor)

Unfortunately, the speed advantage of ASICs over software is fading

High-end processor have VERY high clk frequencies (much faster than ASICs)

Therefore, absolute performance is **not** a very good metric to compare hardware and software

A much better metric (that is independent of clk frequency) is **energy efficiency**, i.e., the amount of useful work done per unit of energy

This metric can be applied to all architectures

The Nature of Hardware and Software

Consider the energy consumption of an **AES** engine (encryption) on different architectures.

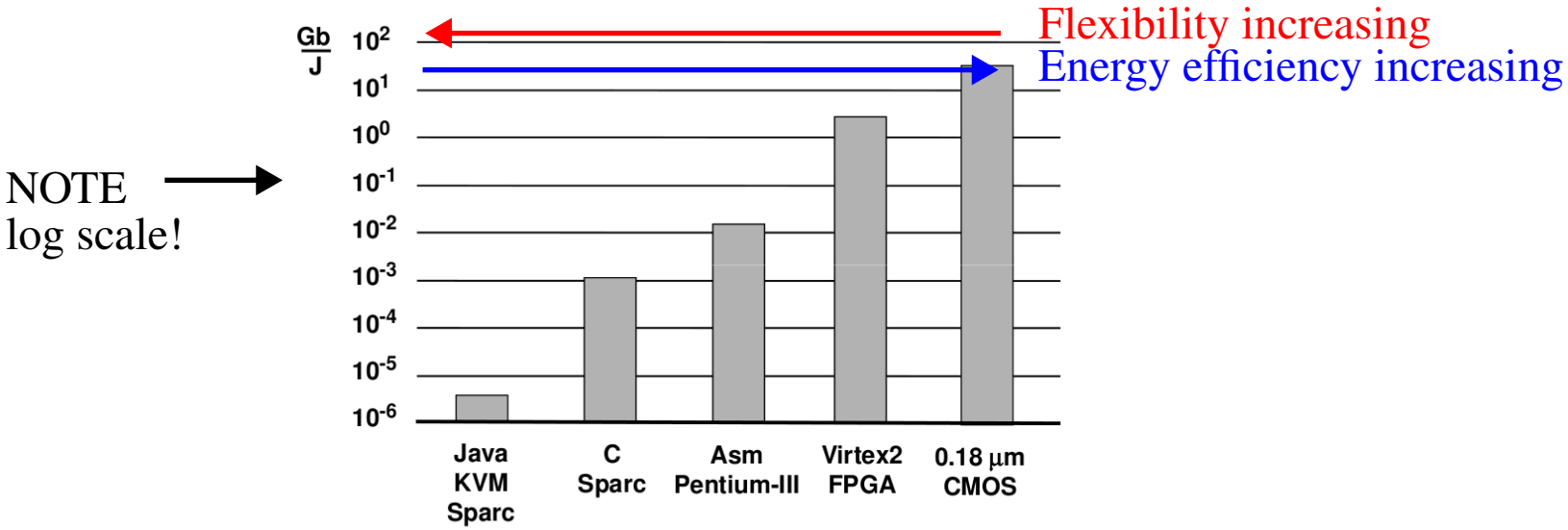


Fig. 1.5 Energy Efficiency

Encryption *throughput* using microprocessors is on order with throughput using dedicated ASICs

This is true b/c of the shorter clock period of microprocessors

When evaluated from a total energy, dedicated hardware engines (ASICs) win hands-down

The Nature of Hardware and Software

This is true b/c there is a **large overhead** associated with executing software instructions in the microprocessor implementation

- Instruction and operand fetch from memory
- Complex state machine for control of the datapath, etc.

Flexibility comes with a significant energy cost -- one which energy optimized applications cannot tolerate

Therefore, you will **never find** a Pentium processor in a cell phone, nor will you find a gigahertz processor inside of an iPod

Specialized hardware architectures are usually more efficient than software from a *relative perspective*

Relative performance means the amount of useful work done per clock cycle

The Nature of Hardware and Software

Highly **parallel implementations** are at an advantage b/c they do many things at the same time

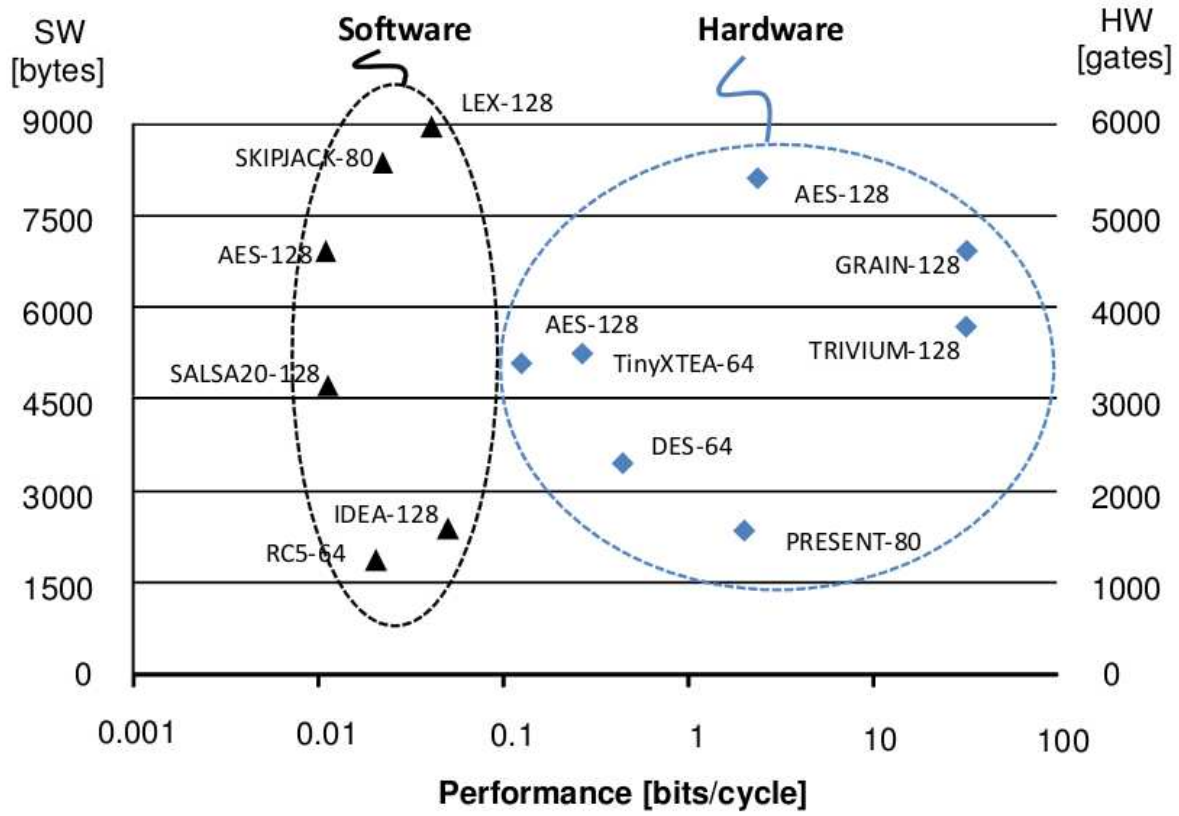


Fig. 1.4 Cryptography on Small Embedded Platforms

HW crypto implementations have a higher *relative performance* when compared to embedded processors

Hardware vs. Software

Arguments in favor of using dedicated HW:

• **Energy Efficiency**:

Nearly every electronic consumer product today carries a battery, e.g., iPod, PDA, mobile phone, Bluetooth device, etc.

Components moved from flexible SW into fixed HW

•**Power Density Limitations**:

Performance in modern high-end processors can no longer be scaled by speeding up the clk

Instead, there is a broad and fundamental shift towards **parallel** architectures

Arguments in favor of using dedicated SW:

• **Design Complexity**

High-end SoCs are extremely complex -- they contain multiple processors, large embedded memories, multiple peripherals and input-output devices

Software bugs are easier to address than hardware bugs

Hardware vs. Software

- **Design Cost**

  New chips are very expensive to design -- designers make chips **programmable** so they can be reused over multiple products or product generations

- **Shrinking Design Schedules**

  Each new technologies is exponentially more complex than the previous generation, and the move to the next generation happens more quickly

  For the designer, this means that each new product generation brings more work that needs to be completed in a shorter amount of time

- **Deep-submicron Effects**

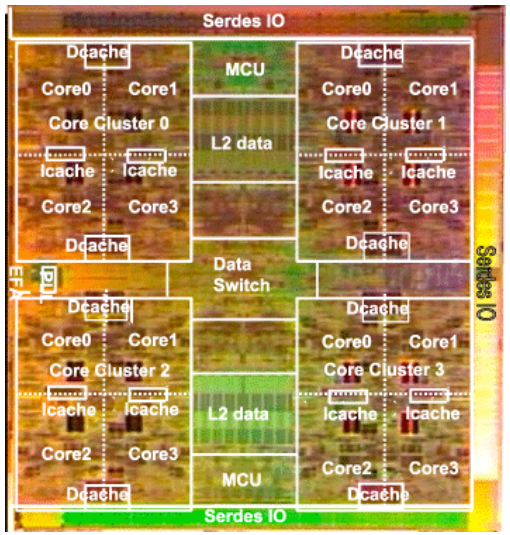
  Designing new hardware from-scratch in high-end silicon processes is difficult b/c of technology-related second-order effects

  - Increased *variability*
  - Decreased *reliability*

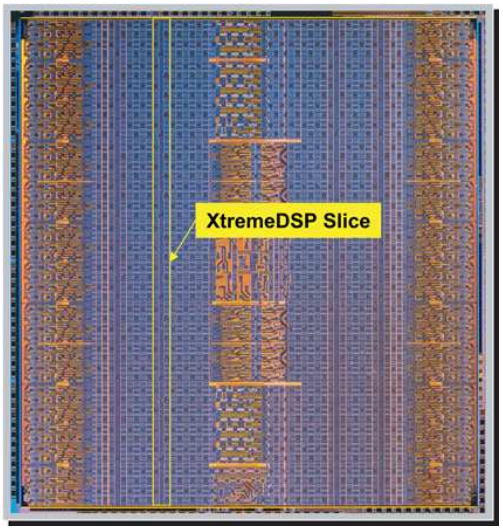
  It is easier to leverage tried-and-true embedded cores and implement functionality in SW

Flavors of Integrated Circuits

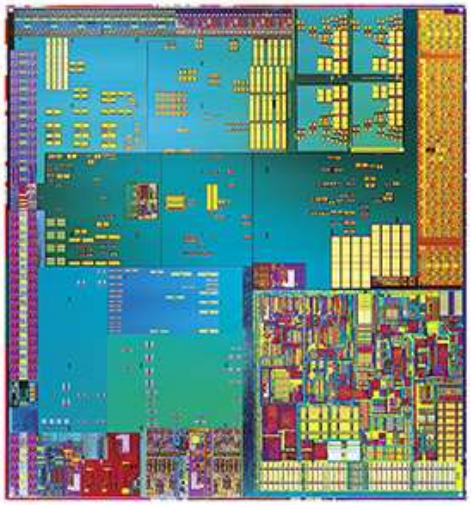
Microprocessor



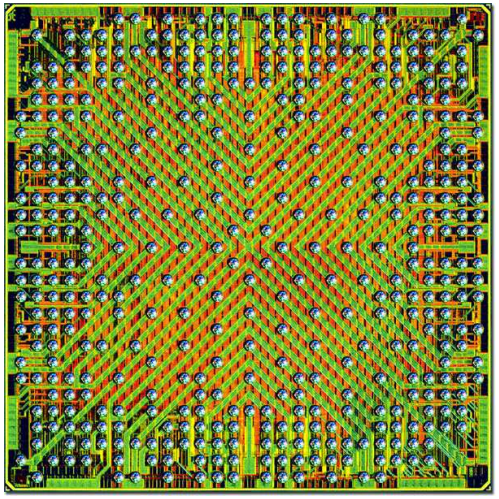
FPGA



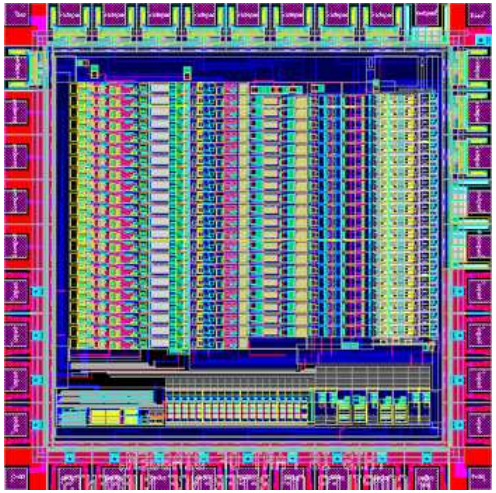
SoC



DSP



Microcontroller

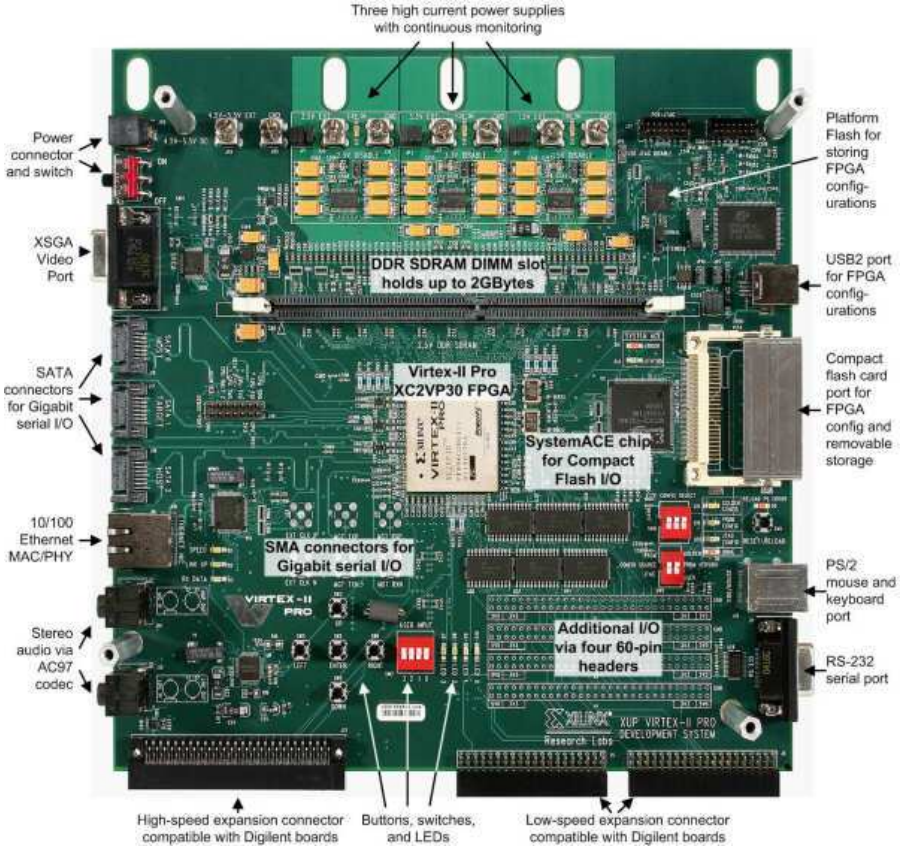


FPGAs

Of these, let's focus on the design flows for FPGAs and ASICs (SoCs)

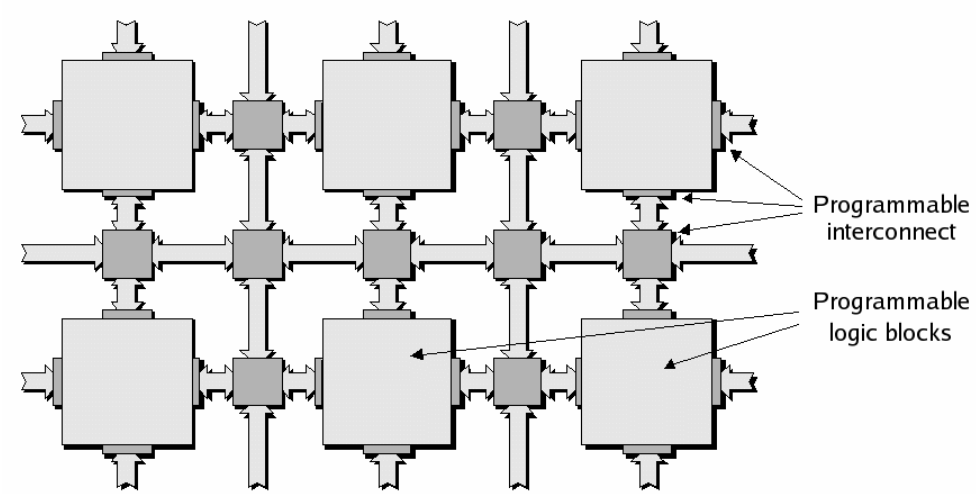
FPGA definition:

Digital integrated circuit that contains *configurable* blocks of logic (CLBs) and *configurable* interconnects between these blocks.



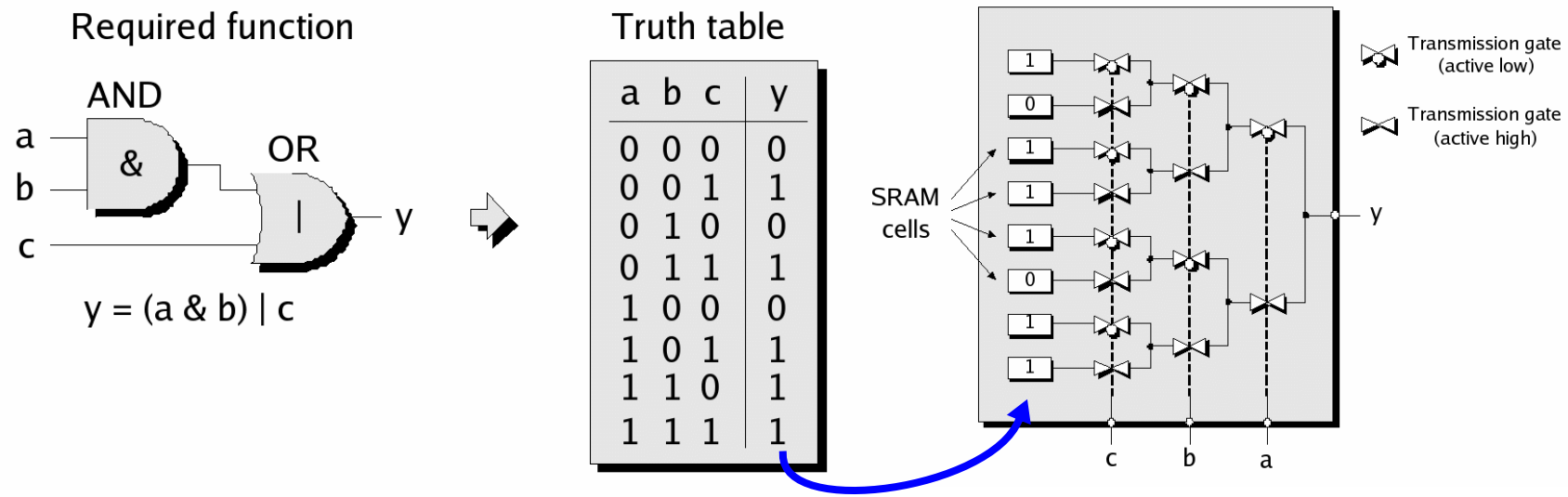
Digilent board w/ V2Pro
130 nm

FPGA Internals



Configurable logic blocks w/ programmable interconnect

Most FPGAs today are LUT-based -- here, the input signals are used as a pointer into a lookup table



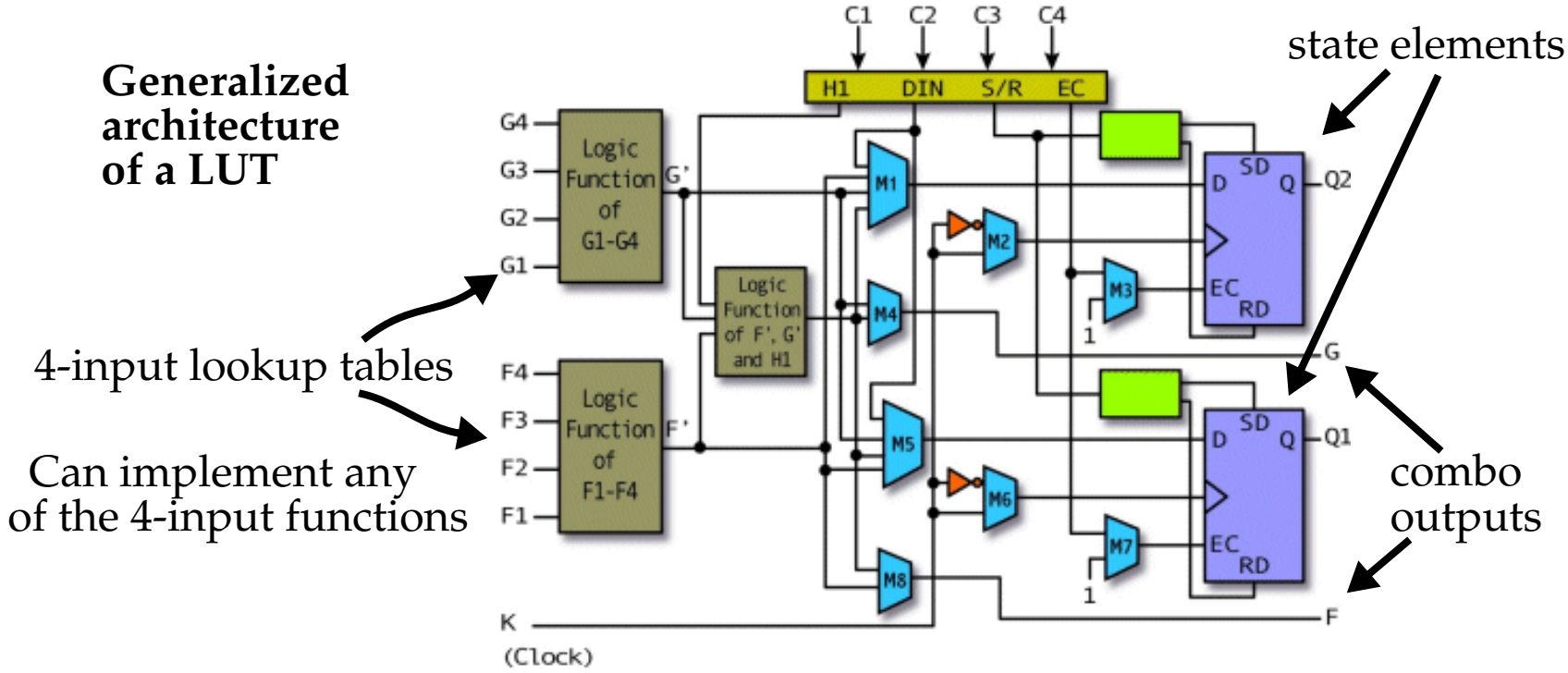
FPGA Internals

Input signals can be decoded using a hierarchy of *transmission-gate* MUXs.

Transmission gates either pass the value on their inputs or are in a *high-impedance* state.

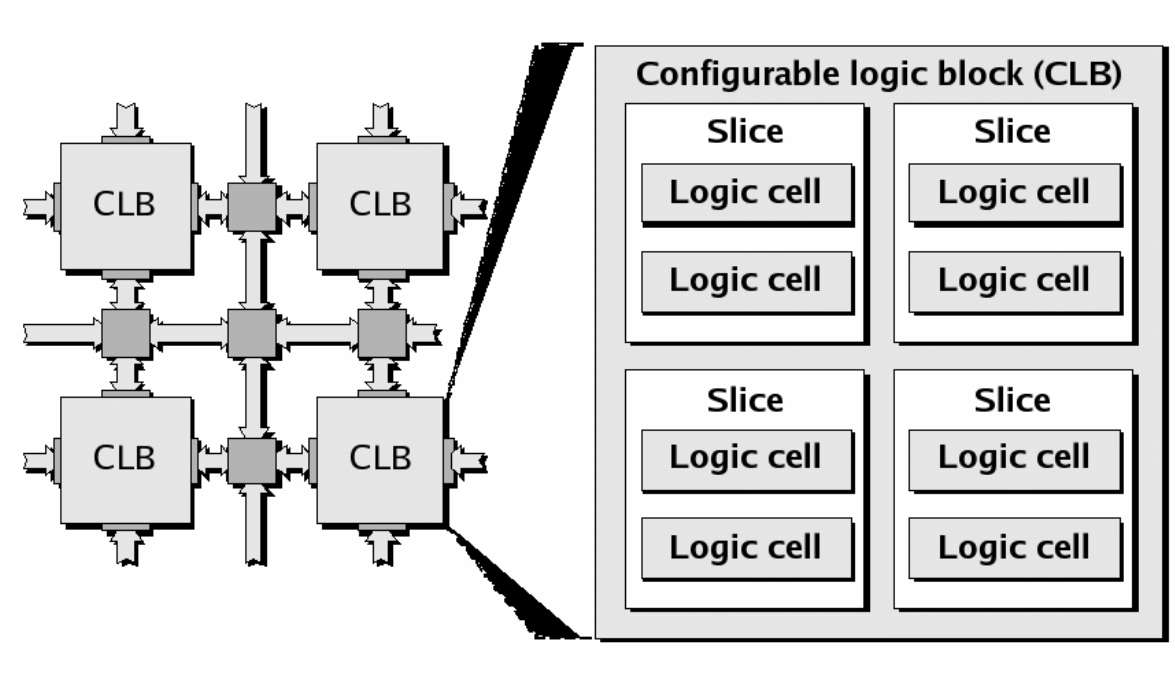
Note that the diagram does not show the serial connection of the cells (scan chain) for simplicity.

Generalized architecture of a LUT



FPGA Internals

Groups of *logic cells* (LCs) use fast local routing.



The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

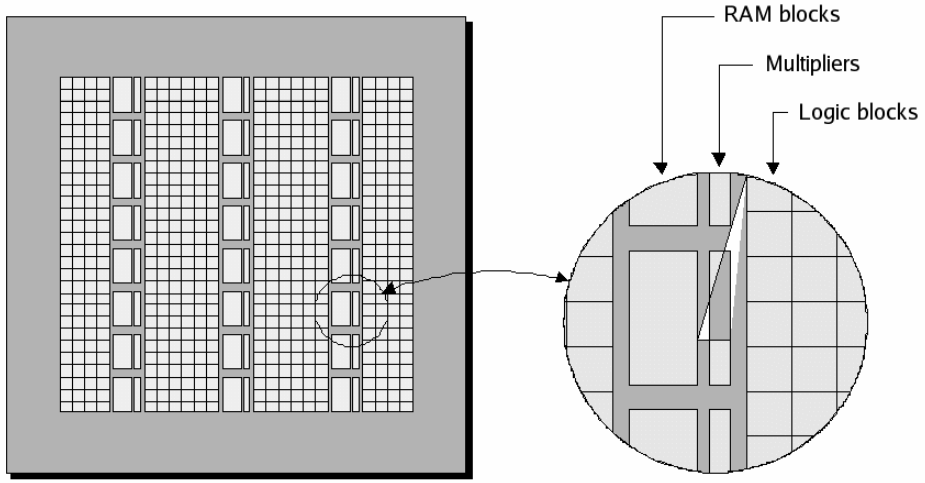
CLBs are implemeted as groups of LCs (*slices* of 4).

This reduces the complexity of providing re-programmable global routing as shown above

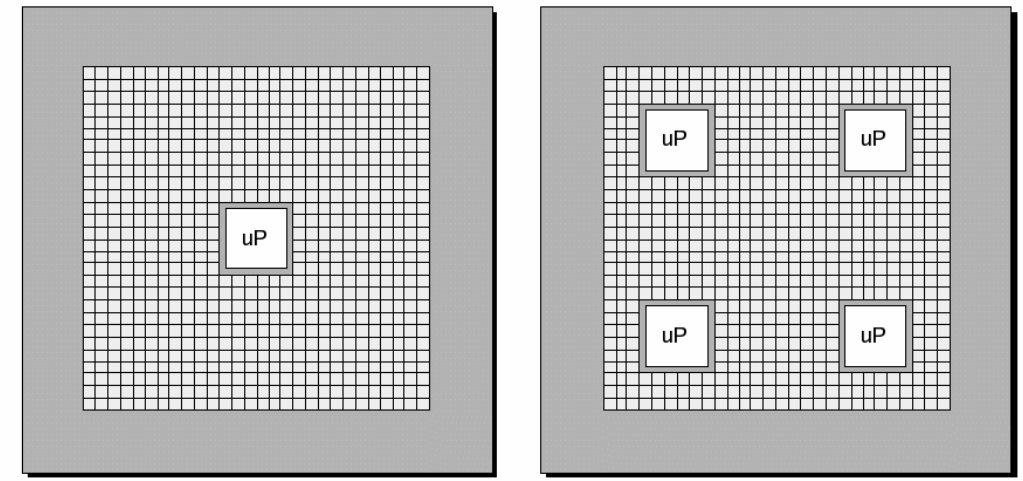
Many applications require memory, so FPGAs now include *embedded RAM* called **e-RAM** or **block RAM**.

FPGA Internals

Embedded FPGA resources



RAMs, Multipliers, Clock managers, etc



Embedded Microprocessors

FPGA Advantages

Key points:

- Manufacturer does NOT determine functionality, rather it is the designer who defines it after the device is fabricated via programming.
- Contain millions of logic gates, allowing large and complex functions to be implemented
- The cost of an FPGA design is **much lower** than that of an ASIC (given the ASIC is not produced in large numbers and cannot amortize this cost).
- Changing the design is also **much easier** with an FPGA, and the time-to-market much shorter (than an ASIC).

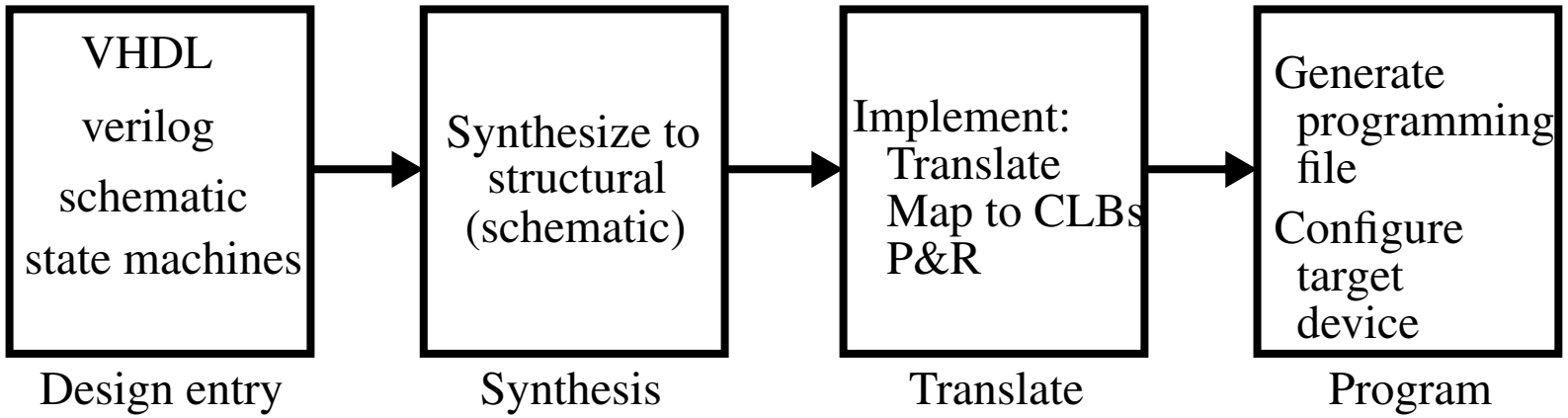
FPGA CAD Flows

FPGAs companies, such as Xilinx and Altera, also provide CAD software to enable designers to generate bitstreams for programming the FPGA

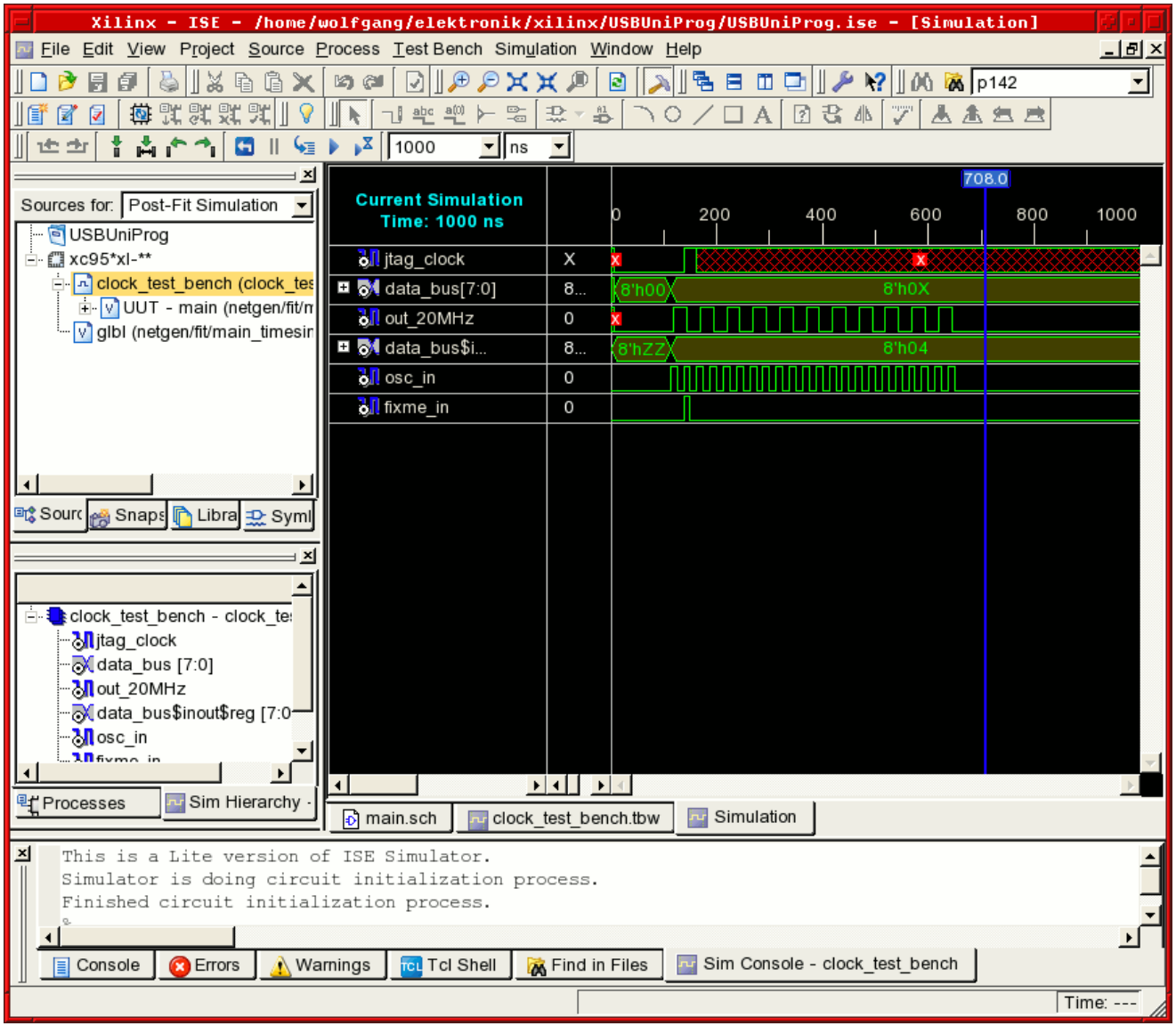
Xilinx provides ISE (Integrated Software Environment) for

- Design entry and synthesis supporting Verilog or VHDL
- Place-and-route
- Verification and debug tools (ChipScope Pro)
- Creation of the bit files to configure the FPGA

Basic Flow



FPGA CAD Flows



FPGA CAD Flows

Flows also include *simulation*, both Behavioral (high level) and Post-route (low level).

Behavioral VHDL code for a 8-bit counter

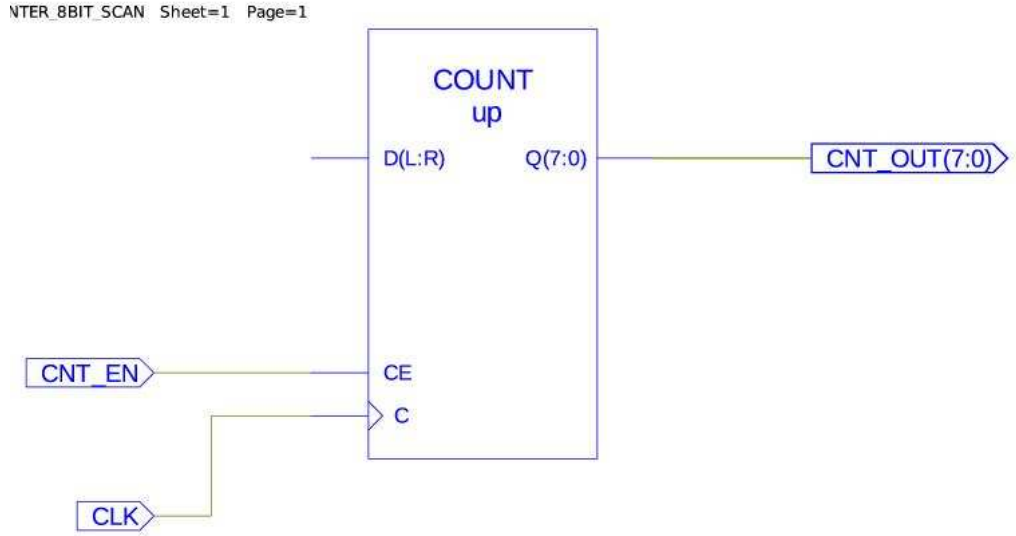
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

entity CNTER_8BIT_SCAN is
   port(
      CNT_EN, CLK : in std_logic;
      CNT_OUT : out std_logic_vector (7 downto 0)
   );
end entity;
```

FPGA CAD Flows

```
architecture beh of CNTER_8BIT_SCAN is

   signal cnter_reg, cnter_next: unsigned(7 downto 0);
   begin

   process(CLK)
      begin
      if (CLK'event and CLK = '1') then
         cnter_reg <= cnter_next;
      end if;
   end process;

   with CNT_EN select
      cnter_next <= cnter_reg + 1 when '1',
                    cnter_reg when others;

   CNT_OUT <= std_logic_vector(cnter_reg);
end beh;
```

FPGA CAD Flows

Synthesizing to structural produces:

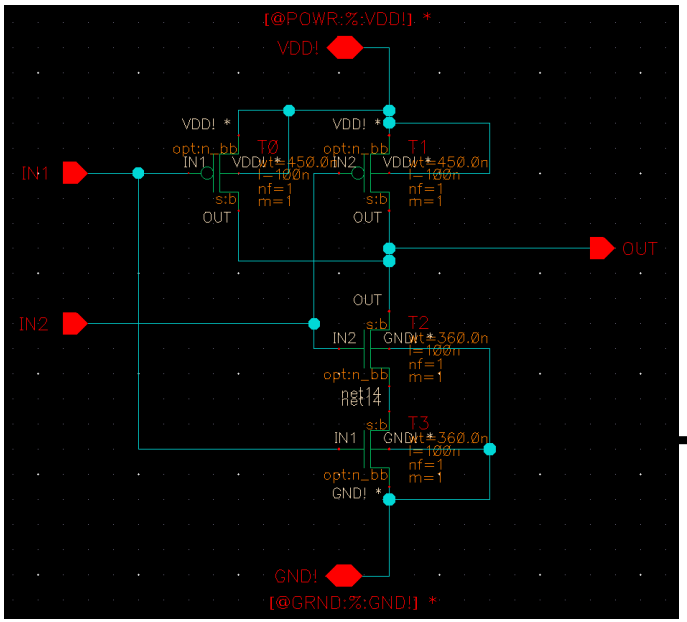


See demo for look at the actual implementation/mapping to the FPGA fabric

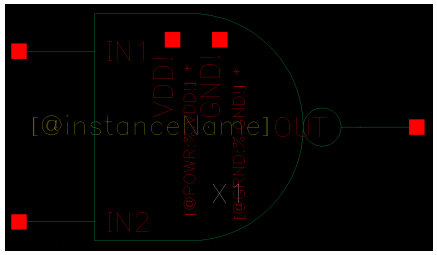
ASIC Design Flow

Designing a chip using a Standard Cell Flow

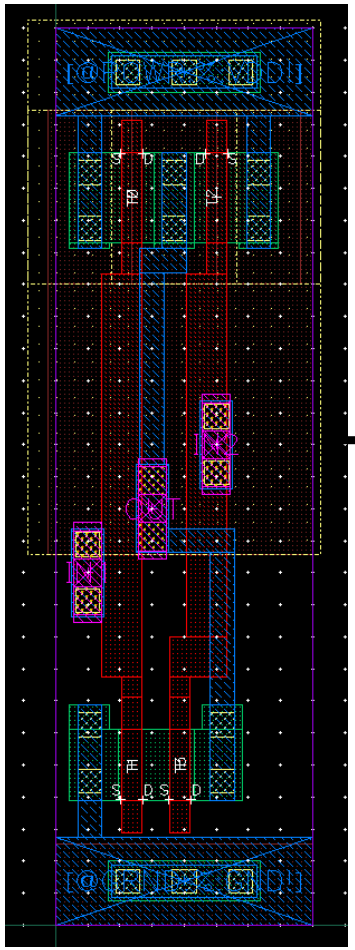
NAND 2X1



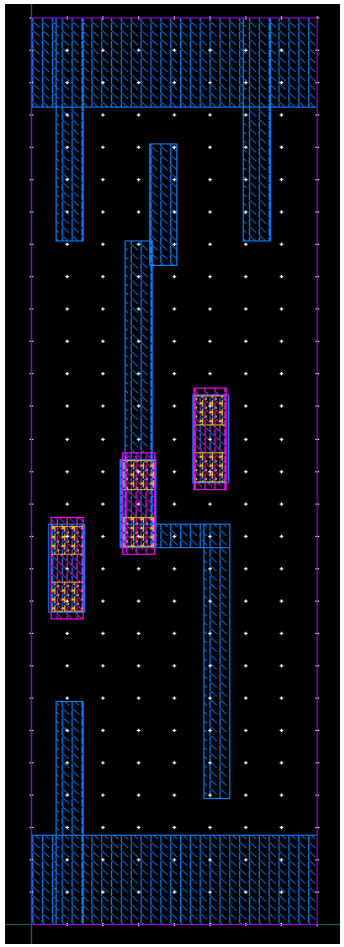
Schematic



Symbol

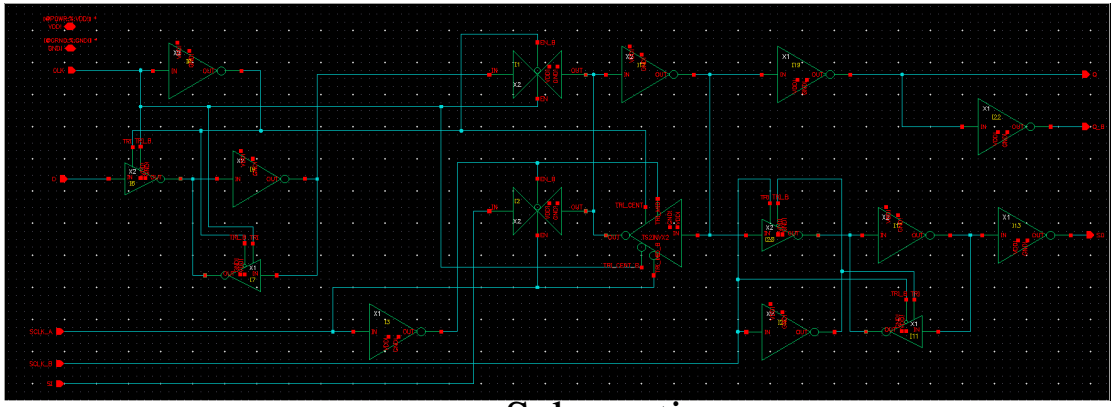


Layout

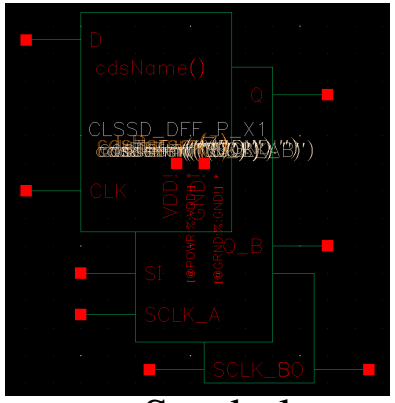


Abstract

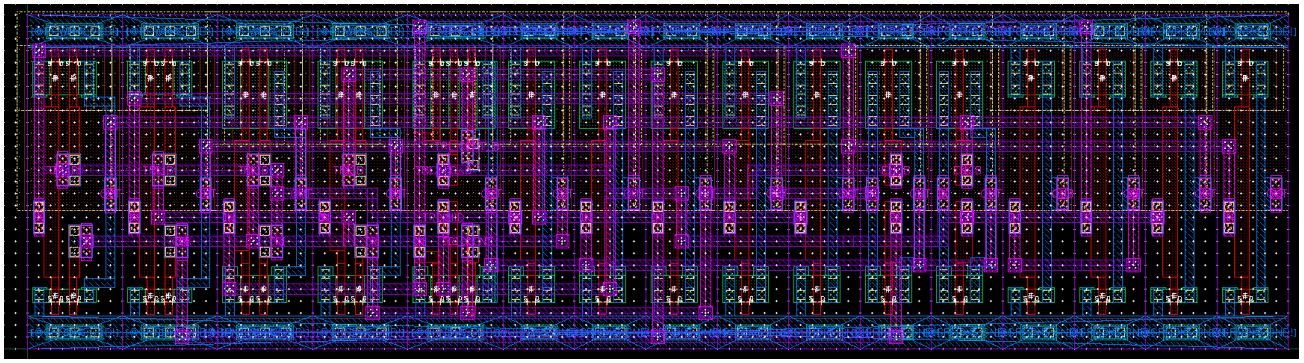
ASIC Design Flow (Clocked LSSD Scan Flop)



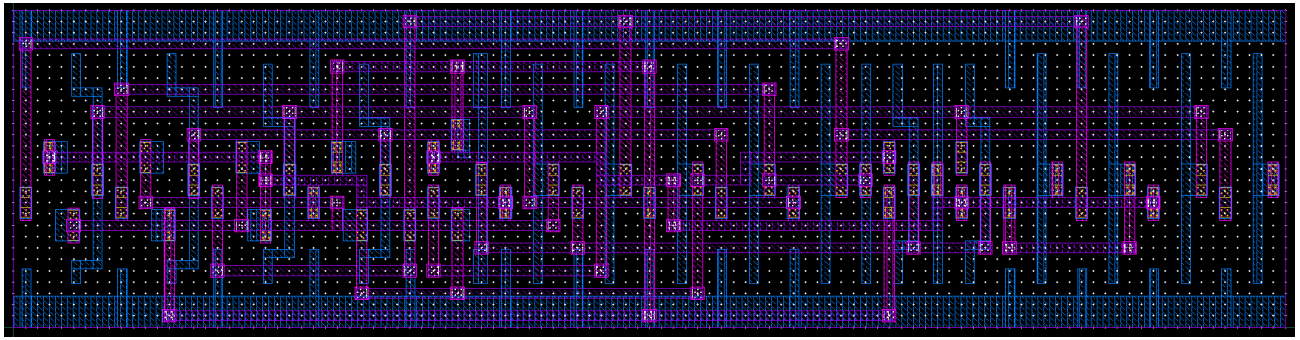
Schematic



Symbol



Layout

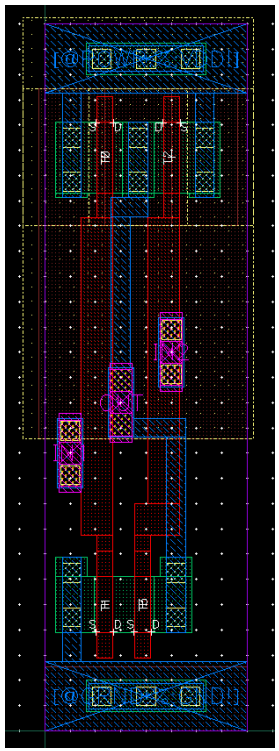


Abstract

ASIC Design Flow

Timing Characterization

Extract RC model

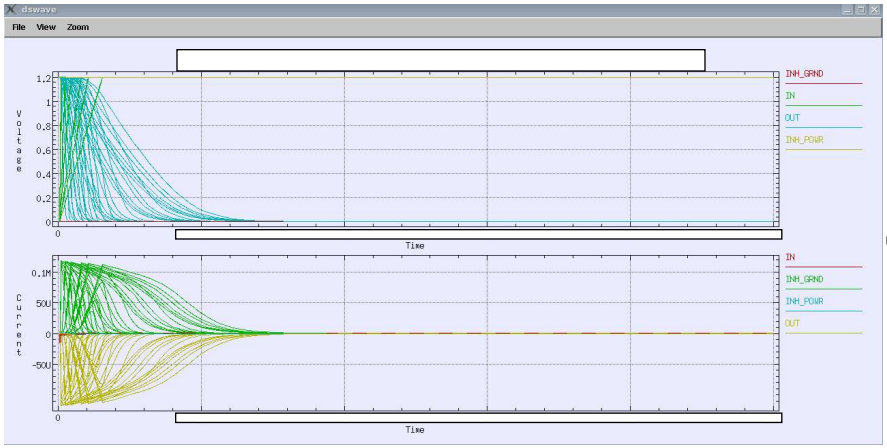


Layout

```
Session  Edit  View  Bookmarks  Settings  Help
// Library name: MosisNewChipProject
// Cell name: NAND2X1
// View name: calibre
subckt NAND2X1 inh_GRND IN1 IN2 OUT inh_POWR
    MT2 (MT2_d MT2_g net14 MT2_b) nfet w=3.6e-07 l=1e-07 par=1 m=1 \

    MT3 (net14 MT3_g MT3_s MT3_b) nfet w=3.6e-07 l=1e-07 par=1 m=1 \

    MT1 (MT1_d MT1_g MT1_s MT1_b) pfet w=4.5e-07 l=1e-07 par=1 m=1 \

    MT0 (MT0_d MT0_g MT0_s MT0_b) pfet w=4.5e-07 l=1e-07 par=1 m=1 \

    rVDD!_19 (MT0_b MT1_b) resistor r=0.01
    rVDD!_18 (MT0_b D4_noxref_pos) resistor r=0.01
    rVDD!_17 (VDD!_2 VDD!_1) resistor r=0.100594
    rVDD!_16 (VDD!_24 VDD!_1) resistor r=0.0387341
    rVDD!_15 (VDD!_20 VDD!_2) resistor r=0.809667
    rVDD!_14 (VDD!_4 VDD!_3) resistor r=0.100594
    rVDD!_13 (VDD!_27 VDD!_3) resistor r=0.00893864
    rVDD!_12 (VDD!_22 VDD!_4) resistor r=0.809667
    rVDD!_11 (VDD!_24 MT0_b) resistor r=10
    rVDD!_10 (VDD!_27 MT0_b) resistor r=10
    rVDD!_9 (VDD!_20 MT0_s) resistor r=7.5
    rVDD!_8 (VDD!_22 MT1_s) resistor r=7.5
    rVDD!_7 (inh_POWR VDD!_24) resistor r=0.0834273
    rVDD!_6 (VDD!_27 inh_POWR) resistor r=0.0834273
    rOUT_27 (MT0_d MT1_d) resistor r=0.01
    rOUT_26 (OUT_2 OUT_1) resistor r=0.23415
    rOUT_25 (OUT_15 OUT_1) resistor r=0.0196693
    rOUT_24 (OUT_3 OUT_2) resistor r=0.0831032
    rOUT_23 (OUT_4 OUT_3) resistor r=1.388
    rOUT_22 (OUT_7 OUT_4) resistor r=0.00704746
    rOUT_21 (OUT_6 OUT_5) resistor r=0.303625
    rOUT_20 (OUT_12 OUT_5) resistor r=0.0670475
    rOUT_19 (OUT_15 OUT_6) resistor r=0.0491733
    rOUT_18 (OUT_9 OUT_7) resistor r=0.159042
    rOUT_17 (OUT_10 OUT_9) resistor r=0.00704746
    rOUT_16 (OUT_29 OUT_10) resistor r=0.144583
                                                    706,15        7%
```

Automatic timing characterization through spice simulation runs

```
Session  Edit  View  Bookmarks  Settings  Help
/* ---------------------- *
 * Design : NAND2X1 *
 * ---------------------- */
cell (NAND2X1) {
  cell_footprint : nand2;
  area : 5.7344;
  cell_leakage_power : 0.081951;
  rail_connection( INH_GRND, RAIL_INH_GRND );
  rail_connection( INH_POWR, RAIL_INH_POWR );
  pin(IN1)  {
    direction : input;
    input_signal_level : RAIL_INH_POWR;
    capacitance : 0.0023529;
    rise_capacitance : 0.00235293;
    fall_capacitance : 0.00233986;
    rise_capacitance_range ( 0.00235286, 0.00254153) ;
    fall_capacitance_range ( 0.00233963, 0.0025395) ;
    ecsm_capacitance(rise) {
      index_1 : "0.08, 0.32, 0.64, 1.2, 1.6, 2.4" ;
      values  :                                                    ;
    }
    ecsm_capacitance(fall) {
      index_1 : "0.08, 0.32, 0.64, 1.2, 1.6, 2.4" ;
      values  :                                                    ;
    }
    max_transition : 2.4;
    internal_power() {
      rise_power(passive_energy_template_6x1) {
        index_1 ("0.08, 0.32, 0.64, 1.2, 1.6, 2.4");
        values (                                                   );
      }
      ecsm_power( passive_energy_template_6x1 ) {
        ecsm_power_type : rise ;
        index_1 ("0.08, 0.32, 0.64, 1.2, 1.6, 2.4");
        ecsm_current_waveform( "0" , "INH_GRND" ) {
          index_1   : "0, 0.00241764, 0.00208302, 0.00229607, 0.00205356, 0.00311081, 0.00813066, 0.00820692, 0.0046
7592, 0.00528148, 0.00460656, 0.00533302, 0.0046686, 0.00538005, 0";
          values
                                                                   ;
        }
        ecsm_current_waveform( "1" , "INH_GRND" ) {
          index_1   : "0, 0.000559496, 0.000524095, 0.000545635, 0.000529801, 0.000593198, 0.000688007, 0.000978764,
0.00228905, 0.00232638, 0.00138789, 0.001157, 0.00115794, 0.00119958, 0.00119174, 0.00124966, 0.00124444, 0";
          values  :
                                                                   ;
        }
        ecsm_current_waveform( "2" , "INH_GRND" ) {
          index_1   : "0, 0.000277048, 0.0002656, 0.000269614, 0.000266809, 0.000288596, 0.000331454, 0.000466801, 0
.00108583, 0.00106163, 0.000580653, 0.000556119, 0.000549, 0.000586701, 0.000571637, 0.000626882, 0.000608097, 0.000
64497, 0";
          values  :
                                                                   ;
        }
        ecsm_current_waveform( "3" , "INH_GRND" ) {
@
@
                                                    17301,13      37%
```



Timing characterization file (synopsys lib format)

ASIC Design Flow

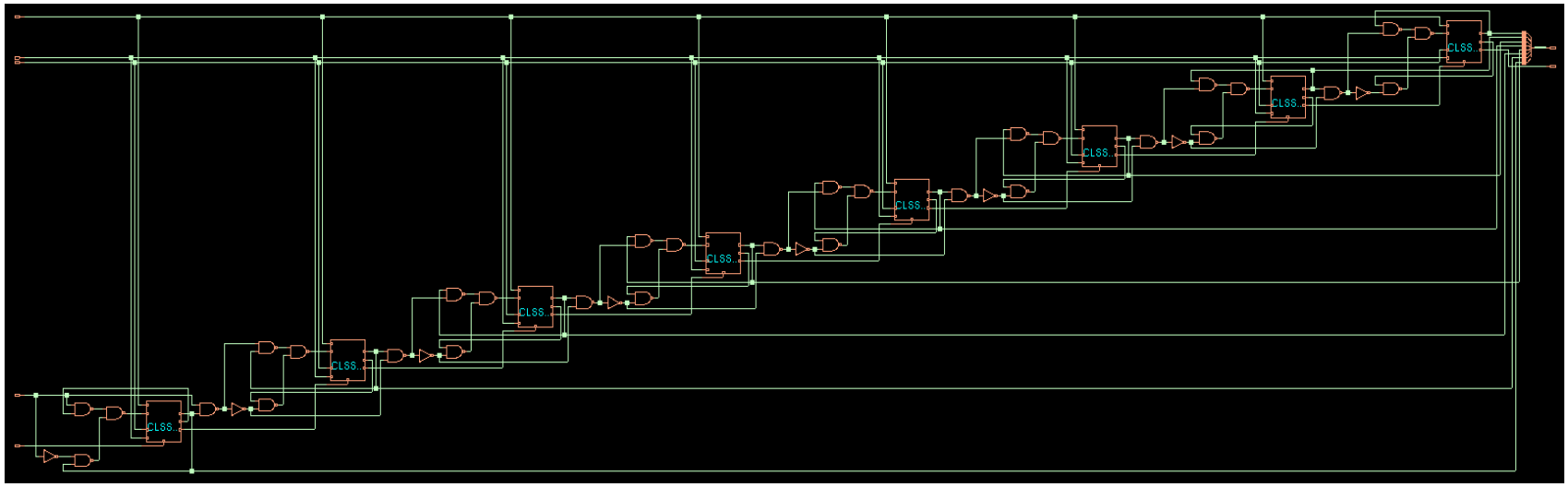
Behavioral VHDL code

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: Jim Plusquellic
--
-- Create Date:    16:04:58 01/25/2011
-- Design Name:
-- Module Name:    CNTER_8BIT_CLSSD_SCAN
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

entity CNTER_8BIT_CLSSD_SCAN is
    port(
        CNT_EN, CLK: in std_logic;
        CNT_OUT : out std_logic_vector (7 downto 0)
    );
end entity;

architecture beh of CNTER_8BIT_CLSSD_SCAN is

    signal cnter_reg, cnter_next: unsigned(7 downto 0);
    begin

    process(CLK)
        begin
        if ( CLK'event AND CLK = '1') then
            cnter_reg <= cnter_next;
        end if;
    end process;

    with CNT_EN select
        cnter_next <= cnter_reg + 1 when '1',
                      cnter_reg when others;

    CNT_OUT <= std_logic_vector(cnter_reg);
end beh;
~
~
~
~
~
~
~
"CNTER_8BIT_CLSSD_SCAN.vhd" 48L, 1104C                    26,7          All
```

Behavioral Synthesis

RC TCL script

```
Session  Edit  View  Bookmarks  Settings  Help
# INITIALIZE
# ================================================================================
# Different flows covered in doc, section 'Encounter RTL Compiler Synthesis Flows'
# set_attribute hdl_search_path ./vhdl/
set_attribute hdl_vhdl_environment common
set_attribute hdl_vhdl_read_version 1993
set_attribute hdl_vhdl_case original

#set_attribute map_to_master_slave_lssd true /

# Information level -- set from 0 (lowest) to 9 (highest)
set_attribute information_level 9

set_attribute lib_search_path /home/jimp/cadence_IBM_9rf/ELC/
set_attribute library std_cells.lib /

find CLSSD_DFF_P_X1/Q -libarc *

#set_attribute preserve false CLSSD_DFF_P_X1
#set_attribute avoid false CLSSD_DFF_P_X1
get_attribute preserve CLSSD_DFF_P_X1
get_attribute avoid CLSSD_DFF_P_X1

# Report other possible problems with scan cells.
filter avoid true [find /libraries -libcell *]
filter preserve true [find /des* -instance *]

# Two modes to synthesize a design: interconnect mode and ple (which uses LEF for better closure
# with back-end tools).
#set_attribute lef_library {cms9flp.lef std_cells.lef}

#set_attribute cap_table_file cap_file

# DFM -- requires a coefficients file
#read_dfm file.dfm

# READ DESIGN
# ================================================================================
read_hdl -vhdl {CNTER_8BIT_SCAN.vhd}

# To allow tracking of any DFT violations (identified later in the flow by the RC-DFT engine) to the
# RTL file name and line number at which the violation occurred, set the following root attribute:

set_att hdl_track_filename_row_col true /

# ELABORATE
# ================================================================================
# Builds data structures, infers registers, performs opt (dead code removeal) checks semantics
elaborate

# APPLY CONSTRAINTS
# ================================================================================
# Constraints include operating conditions, clk wfms and I/O timing
# -- read in SDC constraints. TO use SDC commands from within rc, prefix with dc::
# Use dc::set_time_unit -picoseconds and dc::set_load_unit -femptofarads to set default units --
# ns and pF are default - however, rc assumes ps and fF
                                                                          1,1           Top
```



ASIC Design Flow

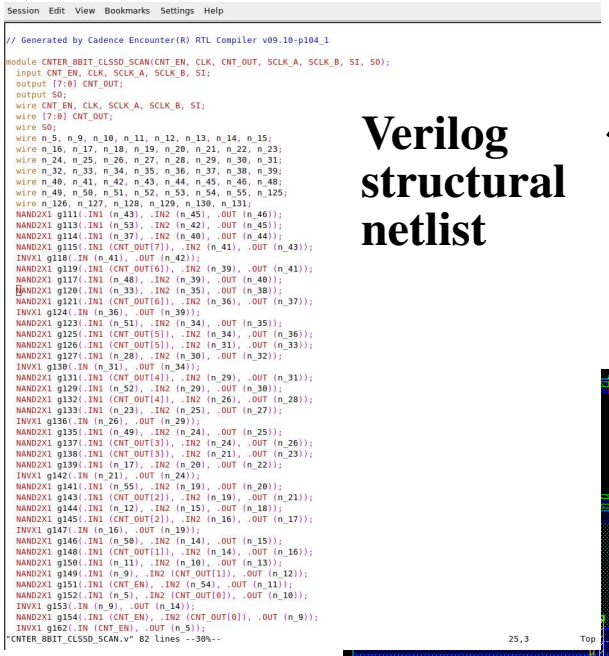
Verilog structural netlist

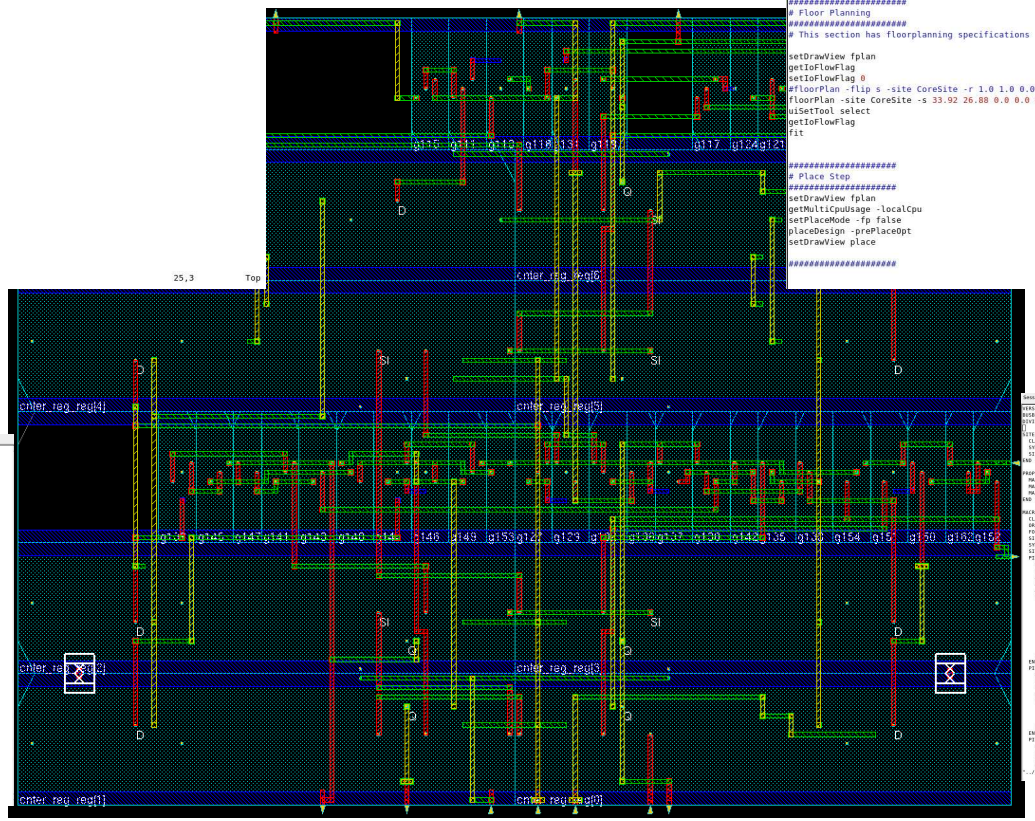
Physical Synthesis

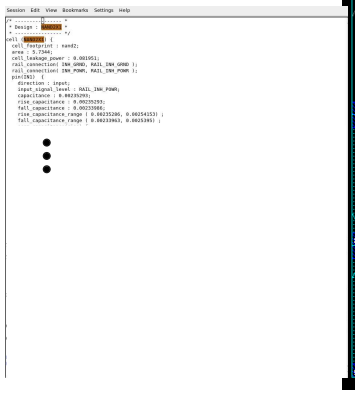
Encounter TCL file

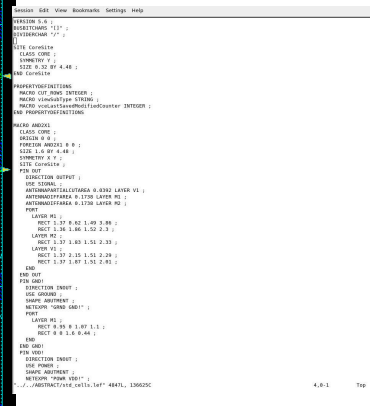
std_cells.lib

std_cells.lef



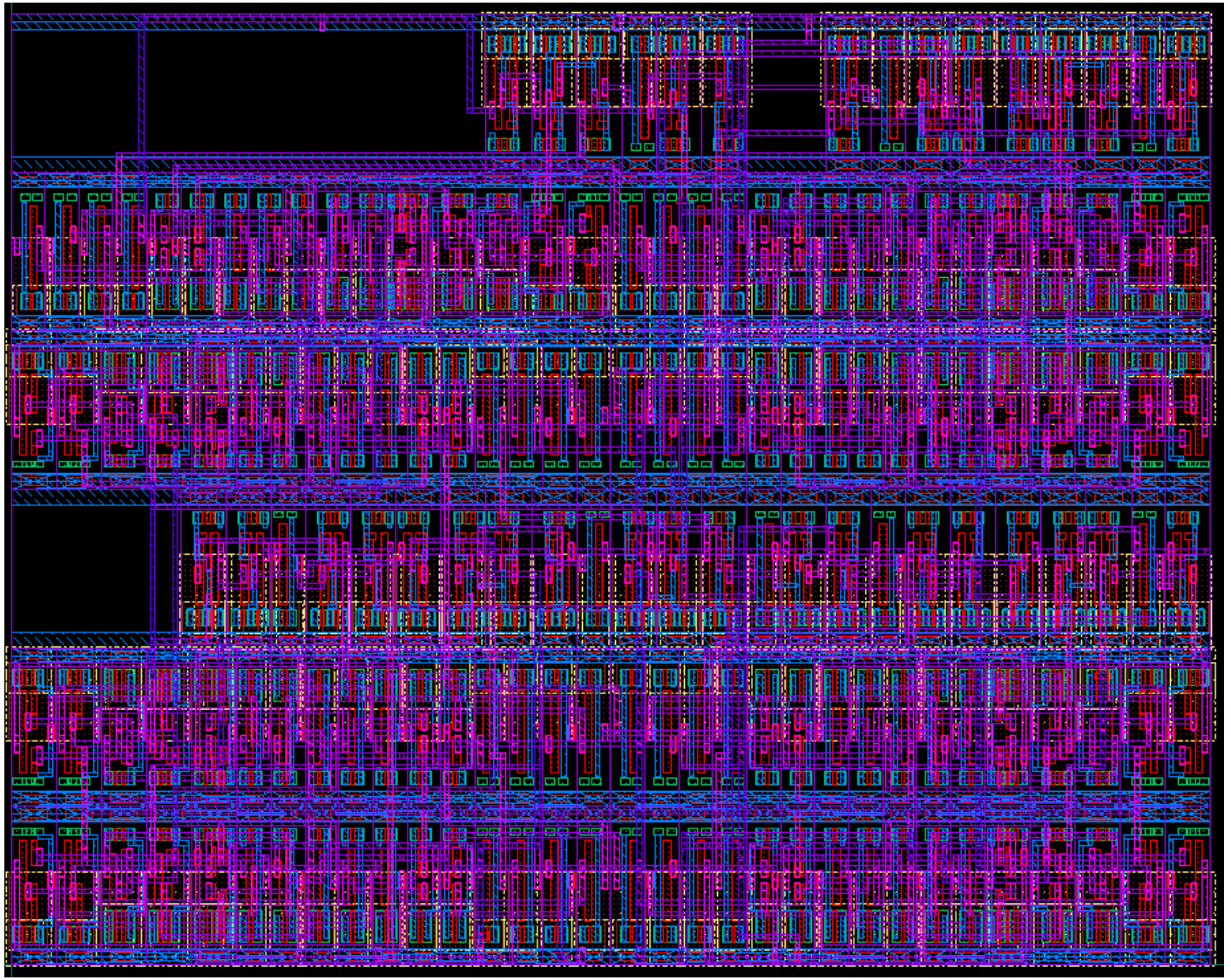






ASIC Design Flow

Final Layout



ASICs

Key points:

- ASICs offer the ultimate in size, number of transistors, complexity and performance.
- However, they are extremely time-consuming and expensive to design.
- Plus, the design is frozen in silicon, requiring a new design/fab iteration if changes are needed.

See demo showing power of the place and route tools