Introduction to the Digital Flow In Mixed Environment 1 – Front End

Christophe FLOUZAT
CEA / DRF / IRFU

PRESENTATION OBJECTIVES

Presentation intended to Analog Developers:

- who need to interact with digital blocs
- who need to develop (simple) digital blocs

Who am I ?

- Initially, an Analog (RF) developer
- Went to digital to develop FPGAs and SoC
- Finally digital developer for micro electronics

→ I have the experience of this path you foresee ☺

Summary

- Firsts steps in a HDL language: Verilog
- Digital Simulation
- Simulation with Analog blocs

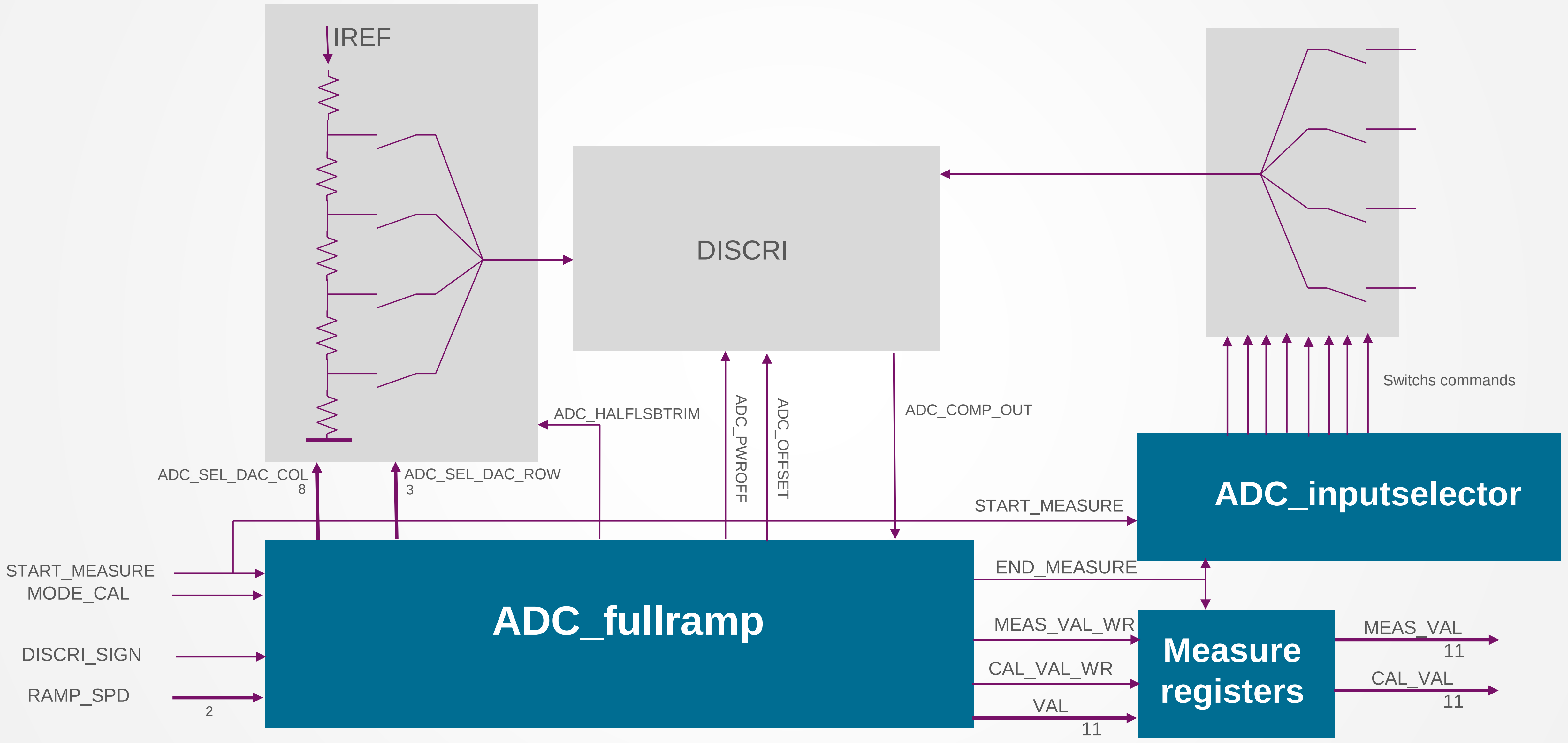
TWO TYPICAL CASES

Either you need to develop a small digital block

Or someone developed this bloc for (with) you

- Understand what’s inside
- Maybe add some little changes
- Simulate it with other (analog) units

TYPICAL EXAMPLE DESIGN: A SIMPLE ADC



Hardware Description Language: Verilog

HARDWARE DESIGN LANGUAGES

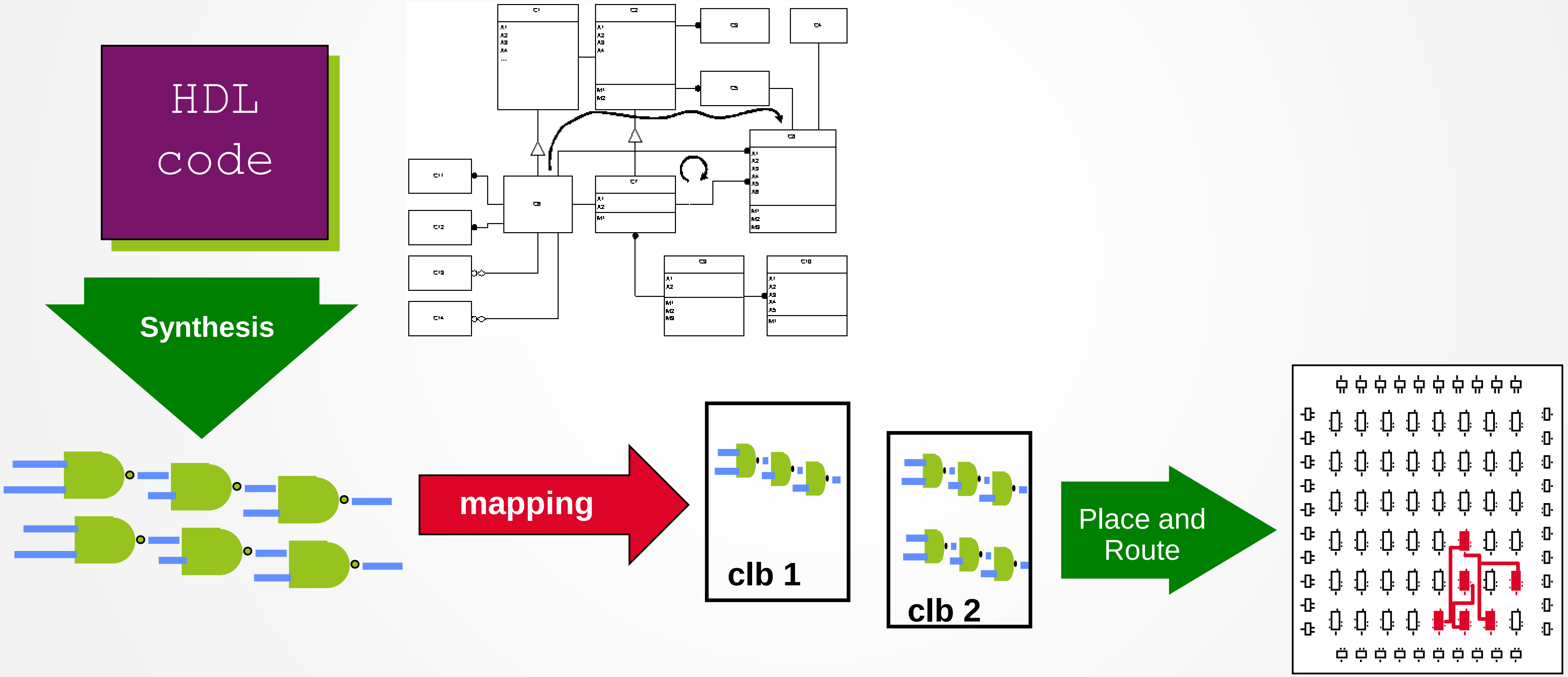
Hardware **D**escription **L**anguage (HDL) used to describe digital hardware elements

Needed as Designs become more and more Complex (millions of gates)

Various levels of design abstractions are used:

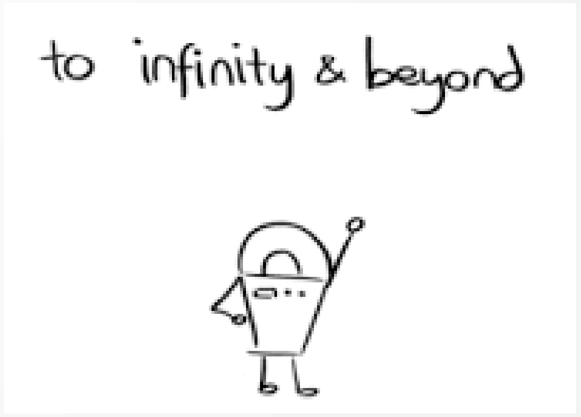
- **Behavioral**:
  - flow control, arithmetic operators, complex delays
- **Register Transfer Level** (RTL):
  - Structural description of the registers and the signal changes between registers
- **Gate level**:
  - combinatorial logic gates (and, or, not,…)
- **Switch level**:
  - layout description of the wires, resistors and transistors (CMOS,PMOS, etc)

From Text Files to Silicium

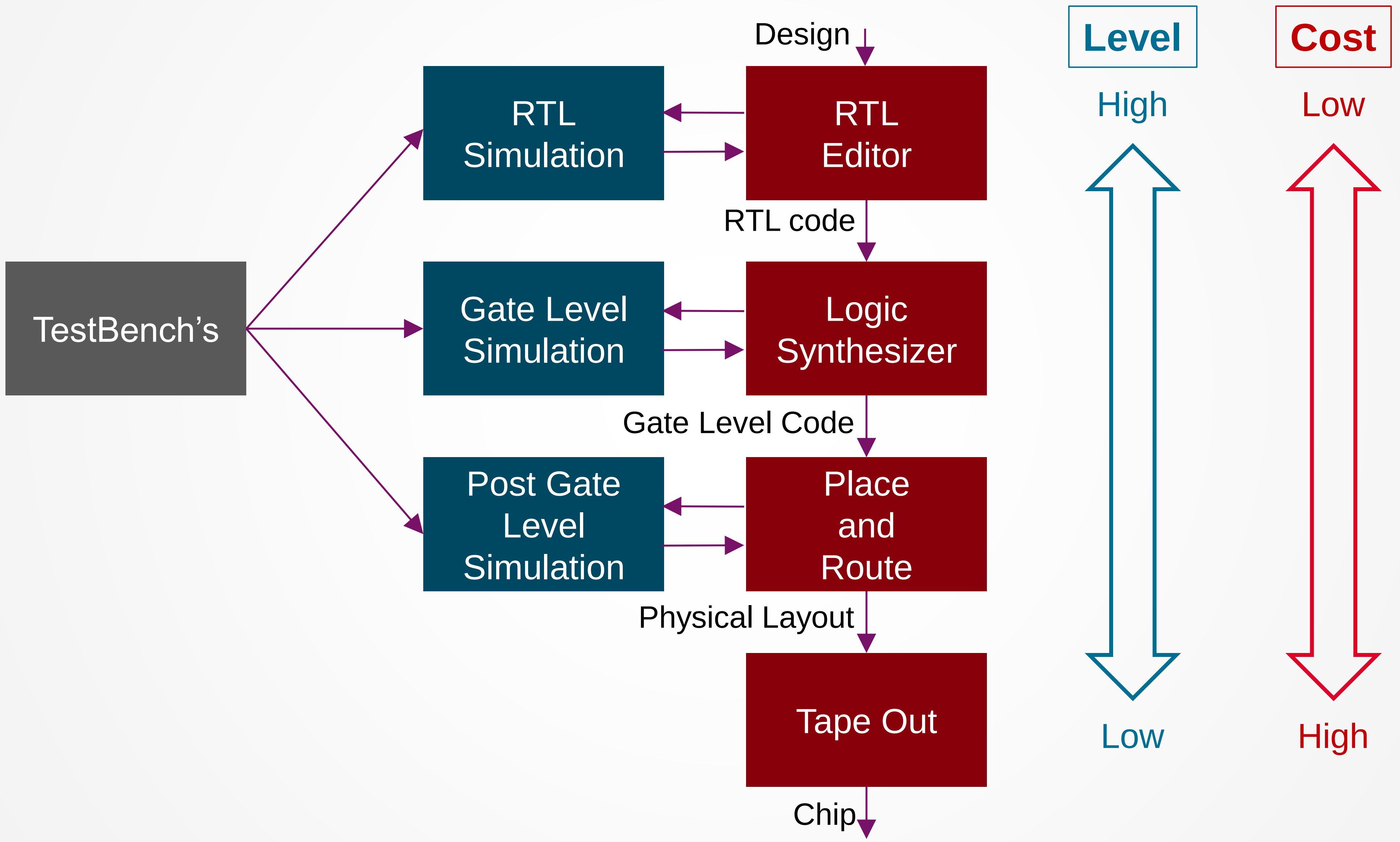


HDL Brief History

Year	Event
1980	'Gateway Design Automation' Company started to develop Verilog
1980	VHDL development by US defense dpt
1990	Cadence bought Gateway and pushed Verilog
1993	VHDL-1993 1st IEEE Std
1995	Verilog-1995 1st IEEE Std
2000	VHDL-2000 After review
2001	Verilog-2001 2nd IEEE Std
2002	SystemVerilog 3.0 Accelera Extension
2005	SystemVerilog-2005 IEEE Standard
2007	VHPI Procedural interface extension to VHDL



HDL DEVELOPMENT



HDL DEVELOPMENT

It looks like code but describes structures like the one used for custom-level design.

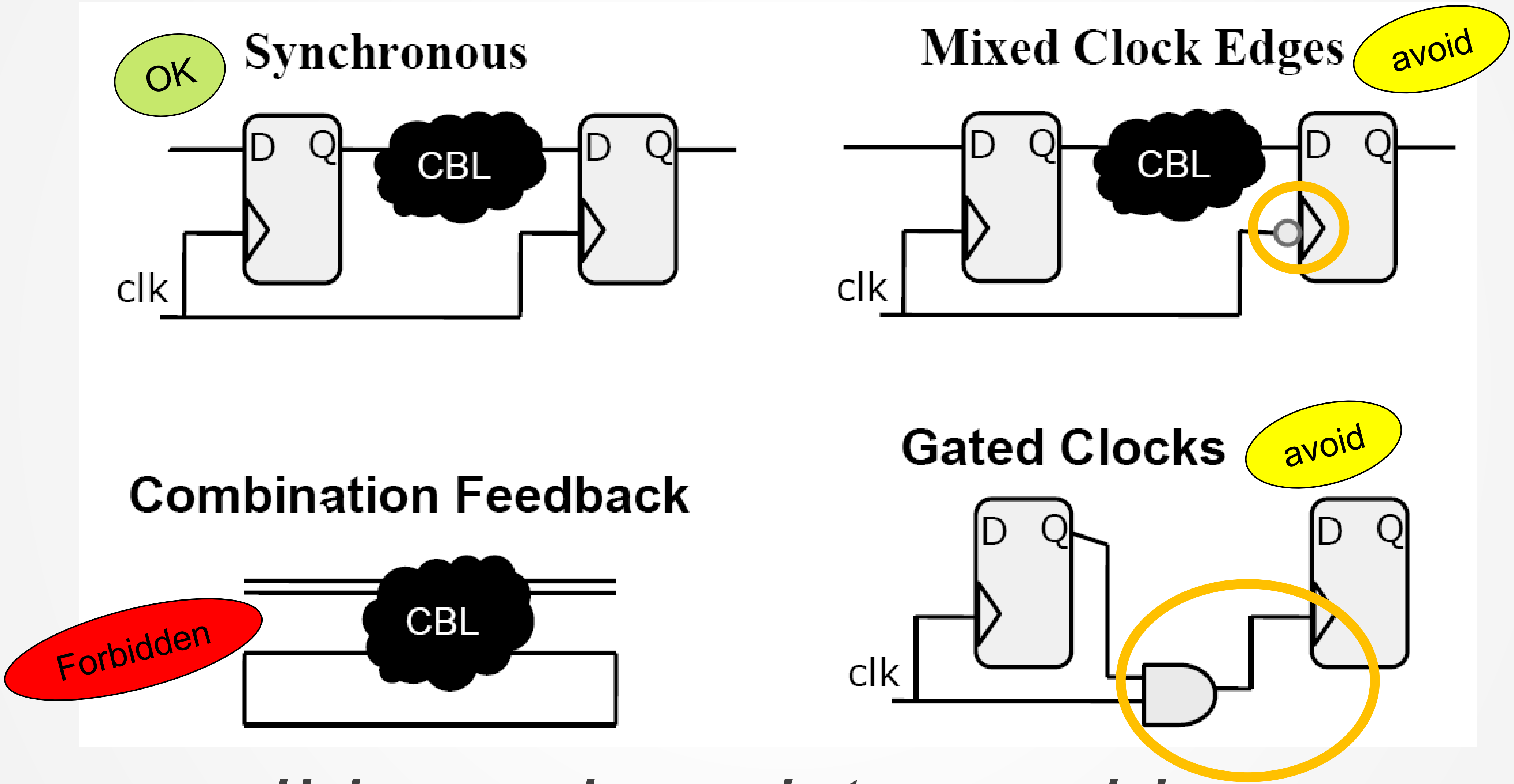
→ Imposes to use the right coding style

Unlike a program which executes sequentially, HDL's execute concurrently.

Two main subsets:
– Synthesizable – reflecting HW / Silicon
– Non-Synthesizable – reflecting *instrumentation* code

Why use Verilog ?
– Relatively simple and close to C
– 60% of the world digital design market

SYNCHRONOUS DESIGNS RECOMMENDED



Unless you know what you are doing…

C-LIKE SYNTAX

- **Case Sensitive**
- **Identifiers can contain**
  - Numbers 0 1 … 9
  - Underscore _
  - UPPER and lower case letters
- **Each Line is terminated with a “;”**
- **Define a name and give a constant value**
  - `define RAM_SIZE 16
- **Include another file**
  - `include adder.v
- **Comments:**
  - // single line
  - /* multiline */

Verilog Coding Styles

There are 3 Major Coding Styles:

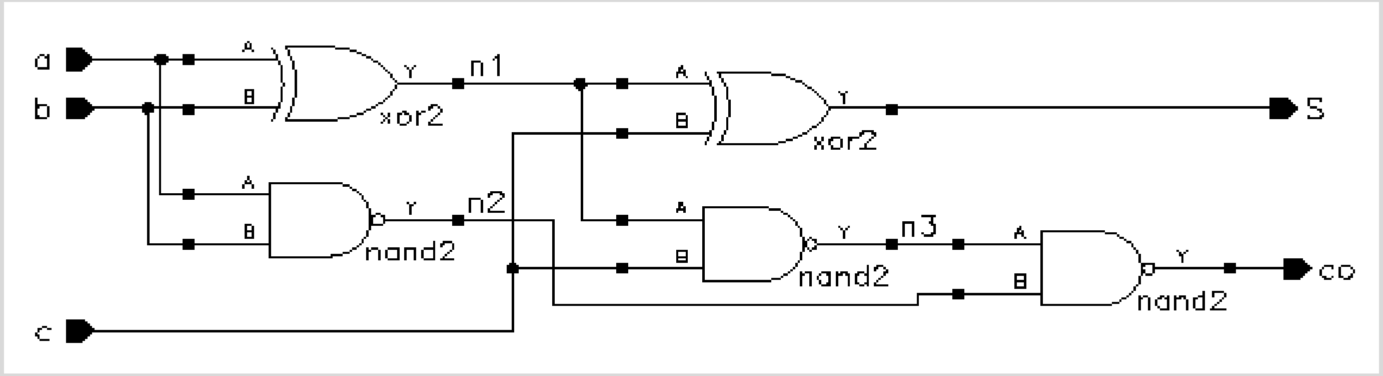
→ **Structural** : Instances + Wires

→ **Combinational** : Continuous Assignment

→ **Procedural** : “always” Blocks

Structural

```
input  a, b ,c;
output co, s;
wire   n1, n2, n3;
xor    (n1, a, b);
xor    (s, n1, c);
nand   (n2, a, b);
nand  (n3, n1, c);
nand  (co, n3,n2);
```



Behavioral

```
input  a, b ,c;
output co, s;
wire   n1, n2, n3;
assign n1 = a^b;
assign s = n1^c;
assign n2 = ~(a&b);
assign n3 = ~(n1&c);
assign co = ~(n3&n2);
```

MODELING STRUCTURE: THE MODULE

The module is the basic building block in Verilog.

Modules can be interconnected to describe the structure of your digital system

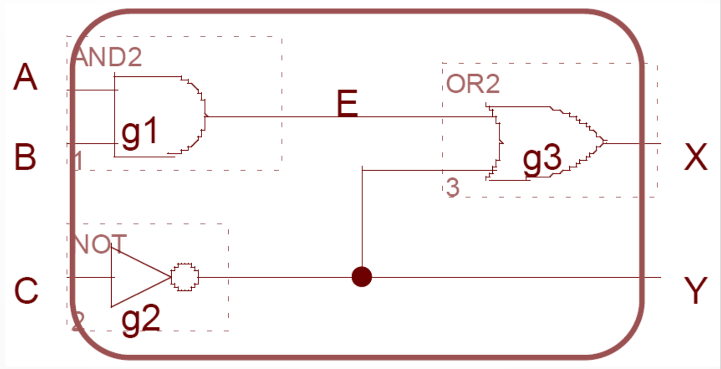
Think hierarchical: module by module

Interface is defined by ports (input, output or inout)

All modules run concurrently

```
module smpl2 (X,Y,A,B,C);
   input  a, b ,c;
   output X, Y;
   assign X = (A&B) | ~C;
   assign Y = ~C;
endmodule
```

assign statement ordering doesn't matter because they execute in parallel



Preferred (verilog2001) Port Declaration Style

```
module smpl2 (
   output X,  // You should
   output Y,  // document each
   input  A,  // and every input
   input  B,  // and output in
   input  C); // the port list

   assign X = (A&B) | ~C;
   assign Y = ~C;
endmodule
```

Module Content

- **Net-list / Call other Modules**
  structural description for the top level
- **Continuous assignments (combination circuits)**
  Data flow specification for simple combinational
  Verilog operators
- **Procedural blocs (RTL)**
  always and *initial* blocks (allow timing control and concurrency)
  C-like procedure statements
- **Primitives (=truth table, state transition table)**
- **Function and task (≈function and subroutine)**

Module Instantiation

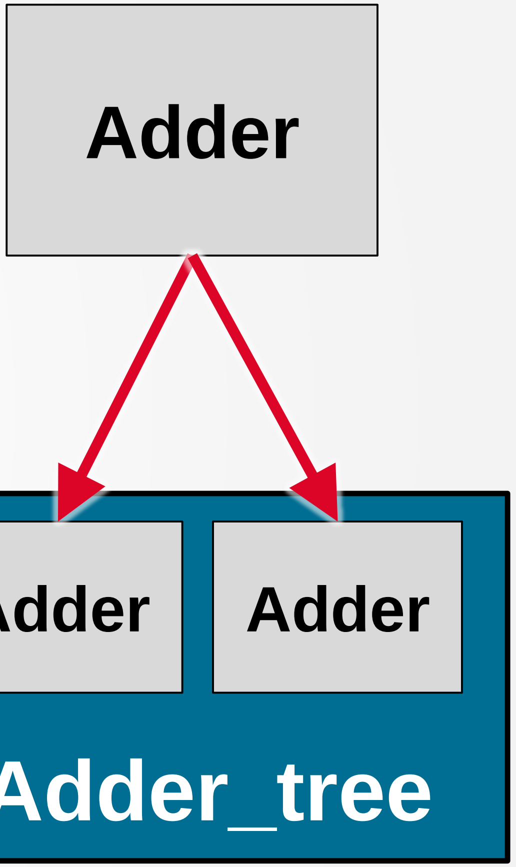
Module Creation = Declaration (can't be nested)

```
module adder(out,in1,in2);
    output  out;
    input   in1,in2,sel;

    assign  out=in1 + in2;
endmodule
```

Module Use = Instantiation

instance
example

```
module adder_tree (out0,out1,in1,in2,in3,in4);
    output  out0,out1;
    input   in1,in2,in3,in4;

    adder   add_0 (out0,in1,in2);
    adder   add_1 (out1,in3,in4);
endmodule
```



PARAMETRIZED MODULE

*Parameterized register model - **Verilog-1995** style*

```
module myreg(q, d, clk, rst_n);
   parameter  SZ=8;
   output [SZ-1:0] q;
   input  [SZ-1:0] d;
   input  clk, rst_n;
   reg    [SZ-1:0] q;

   always @(posedge clk or negedge rst_n)
   begin
      if (!rst_n)  q <= 0;
      else         q <= d;
   end
endmodule
```

*Parameterized register model - **Verilog-2001** style*

```
module myreg
#(
   parameter  SZ=8
)
(
   output reg [SZ-1:0]  q;
   input      [SZ-1:0]  d;
   input                clk
   input                rst_n;
);
   always @(posedge clk or negedge rst_n)
   begin
      if (!rst_n)  q <= 0;
      else         q <= d;
   end
endmodule
```

EXPLICIT PORT ASSIGNMENTS (RECOMMENDED)

```
module mytop
#(
   parameter  SIZE=8
)
(
   output reg [SIZE-1:0] q;
   input      [SIZE-1:0] d;
   input                 clk
   input                 rst_n;
);

   ~~myreg #(SIZE) toto(q,d,clk,rst_n);~~

endmodule
```

```
module mytop
#(
   parameter  SIZE=8
)
(
   output reg [SIZE-1:0] q;
   input      [SIZE-1:0] d;
   input                 clk
   input                 rst_n;
);

   myreg
          #(.SZ(SIZE))
          toto(.q(q),.d(d),
               .clk(clk),.rst_n(rst_n));
endmodule
```

Explicit assignments may avoid you some headaches if parameter order changes…

“Generate” Feature

Lets you generate code automatically

Starts with generate, ends with endgenerate
You can use if statements, always blocks, for loops

```
genvar i;
generate
   if (MODE==INNER_BARREL) begin
          for (i=0;i<3;i=i+1) begin
              evtMgr multEvt(.rst(rst), .clk(clk), .out(out[i]), .trig(trig));
          end
   else
          evtMgr multEvt(.rst(rst), .clk(clk), .out(out[0]), .trig(trig));
   end
endgenerate
```

Just be careful, errors in a generate block are hard to track sometimes.

PROCEDURAL BLOCS: SEQUENTIAL LOGIC (1/2)

Behavioral model statements to define sequential actions

"*always*" keyword infers a register (flip-flop or latch)

- @(event-expression) means the statement below are executed
  - when the expression is true (i.e. event-driven)
    - Example:
      - @(posedge clk) means that the statement is only executed on the positive edge of the clock
- "*reg*" declaration needed for always block assignment

```
module dff(
   output Q,   // data output
   input  D,   // data input
   input  CLK);// clock input

   reg Q;
   always @(posedge(CLK)) begin
          Q <= D;
   end
endmodule
```

Input D is sampled, and assigned to Q only when the clock ticks

PROCEDURAL BLOCS: SEQUENTIAL LOGIC (2/2)

- **C-like statements inside always block**
  - if and case (see typical structures)
  - Very useful for describing conditional logic

```
always @(posedge clk)
begin
   statement1;
   statement2;
   …
end
```

- **Statements are executed sequentially to describe the intended behavior of the circuit**

  → Last assignment to variable is value put into the register

  → Use with caution!

NET LIST: REGISTERS AND NETS

NETS

Nets are physical connections between devices
Nets always reflect the logic value of the driving device
Many types of nets, but all we care about is **`wire`**

Can't appear inside an always bloc

Types of nets

wire, tri	: default
wor, trior	: wire-ORed
wand, triand	: wire-ANDed
trireg	: with capacitive storage
tri1	: pull high
tri0	: pull low
supply1	: power
supply0	: ground

REGISTER TYPE

Implicit storage – unless variable of this type is modified it retains previously assigned value
Does not necessarily imply a hardware register
Register type is denoted by **`reg`**

Assignment in always bloc

NET DECLARATION

Declaring a net

```
wire [<range>] <net_name> [<net_name>*];
```

Range is specified as `[MSb:LSb]`. *Default is one bit wide*

Declaring a register variable

```
reg [<range>] <reg_name> [<reg_name>*];
```

Declaring memory

```
reg [<range>] <memory_name> [<start_addr>:<end_addr>];
```

Examples

```
reg r; // 1-bit reg variable
wire w1, w2; // 2 1-bit wire variable
reg [7:0] vreg; // 8-bit register
reg [7:0] memory [0:1023]; a 1 KB memory
```

LOGICAL VALUES AND NUMBER BASES

LOGIC VALUES

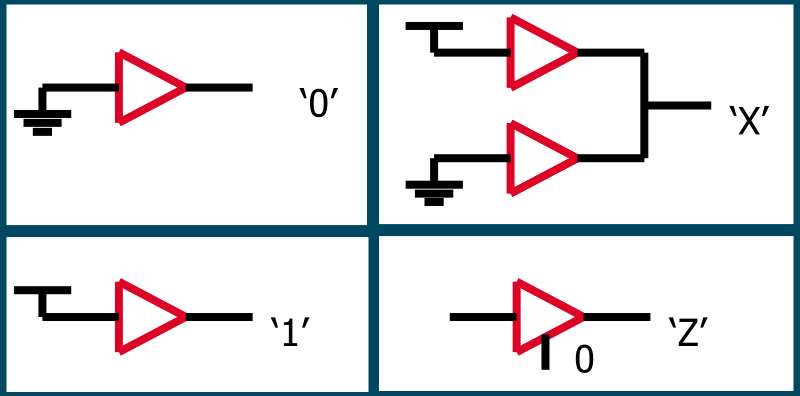
0: zero, logic low, false, ground

1: one, logic high, power

X: unknown

Z: high impedance, unconnected, tri-state

4-value logic system in Verilog :



Format : **<size>'<base><value>**

Examples :

8'd16

8'h10

8'o20

'hCAFE

12'b0000_0100_0110 - binary number with 12 bits (_ is ignored)

SIGNAL CONCATENATION

Representations	**Meanings**
{b[3:0],c[2:0]}	{b[3] ,b[2] ,b[1] ,b[0], c[2] ,c[1] ,c[0]}
{a,b[3:0],w,3'b101}	{a,b[3] ,b[2] ,b[1] ,b[0],w,1'b1,1'b0,1'b1}
{4{w}}	{w,w,w,w}
{b,{3{a,b}}}	{b,a,b,a,b,a,b}

SOME TRAPS WITH VECTORS OF BITS

- A[3:0] - vector of 4 bits: A[3], A[2], A[1], A[0]
- Treated as an *unsigned* integer value by default,
    - **e.g. A < 0 is *never* true!**
    - e.g. beware of sums
        - C[4:0] = A[3:0] + B[3:0]; with A = 0110 (6) and B = 1010(-6)
        - You get C = 10000 : **different from expected 00000**,
        - as B is zero-padded, not sign-extended (unless declared signed)

- Concatenating bits/vectors into a vector, e.g. sign extend

  B[7:0] = {A[3], A[3], A[3], A[3], A[3:0]};

  B[7:0] = {4{A[3]}, A[3:0]};

- Vectors can be declared as “signed”

CONTINUOUS ASSIGNMENT

Describe combinational logic

Operands + operators

Drive values to a net

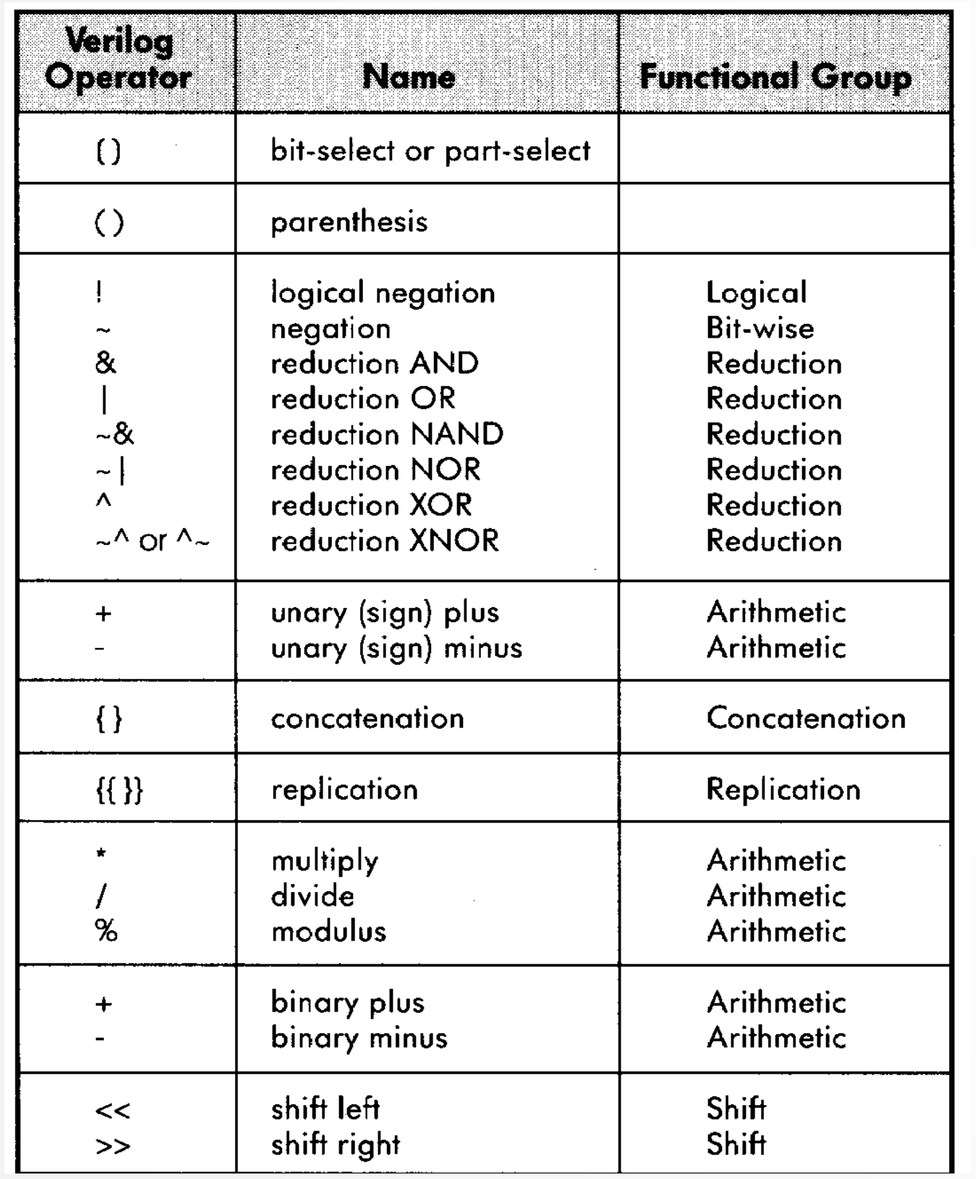
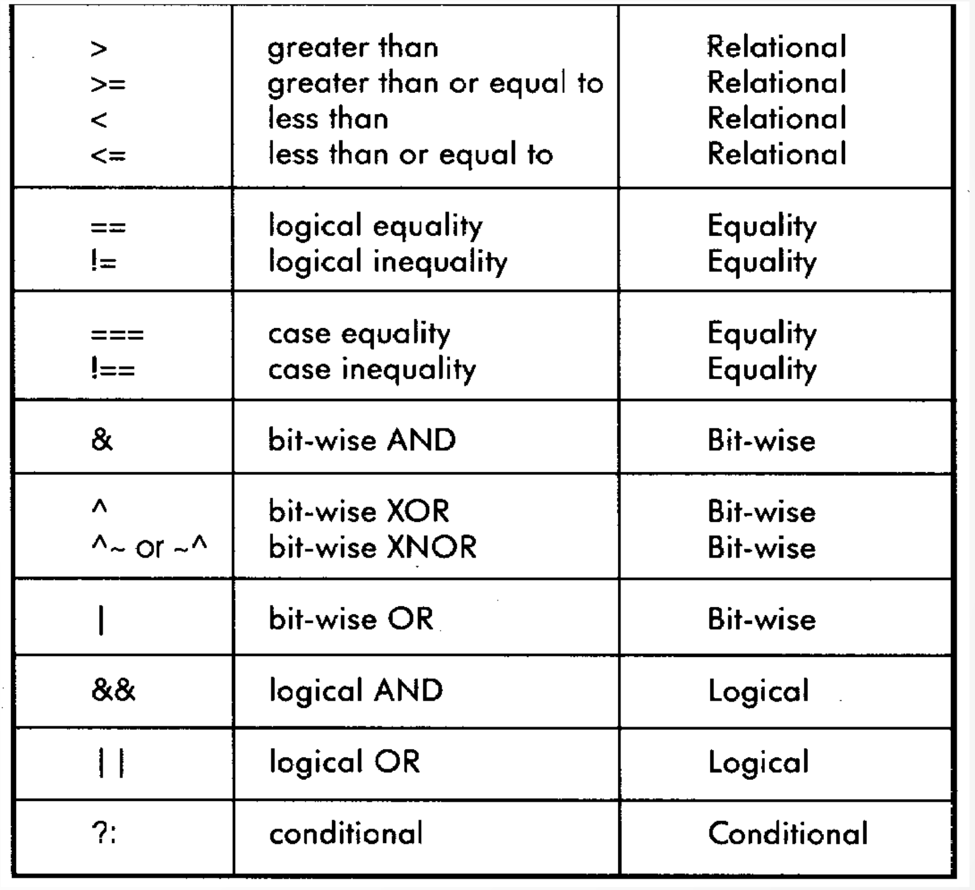
```
assign      out  = a&b ;     // and gate
assign      eq   = (a==b) ;  // comparator
wire #10    inv  = ~in ;     // inverter with delay
wire [7:0]  c    = a+b ;     // 8-bit adder
```

Avoid logic loops

```
assign a = b + a ;
```

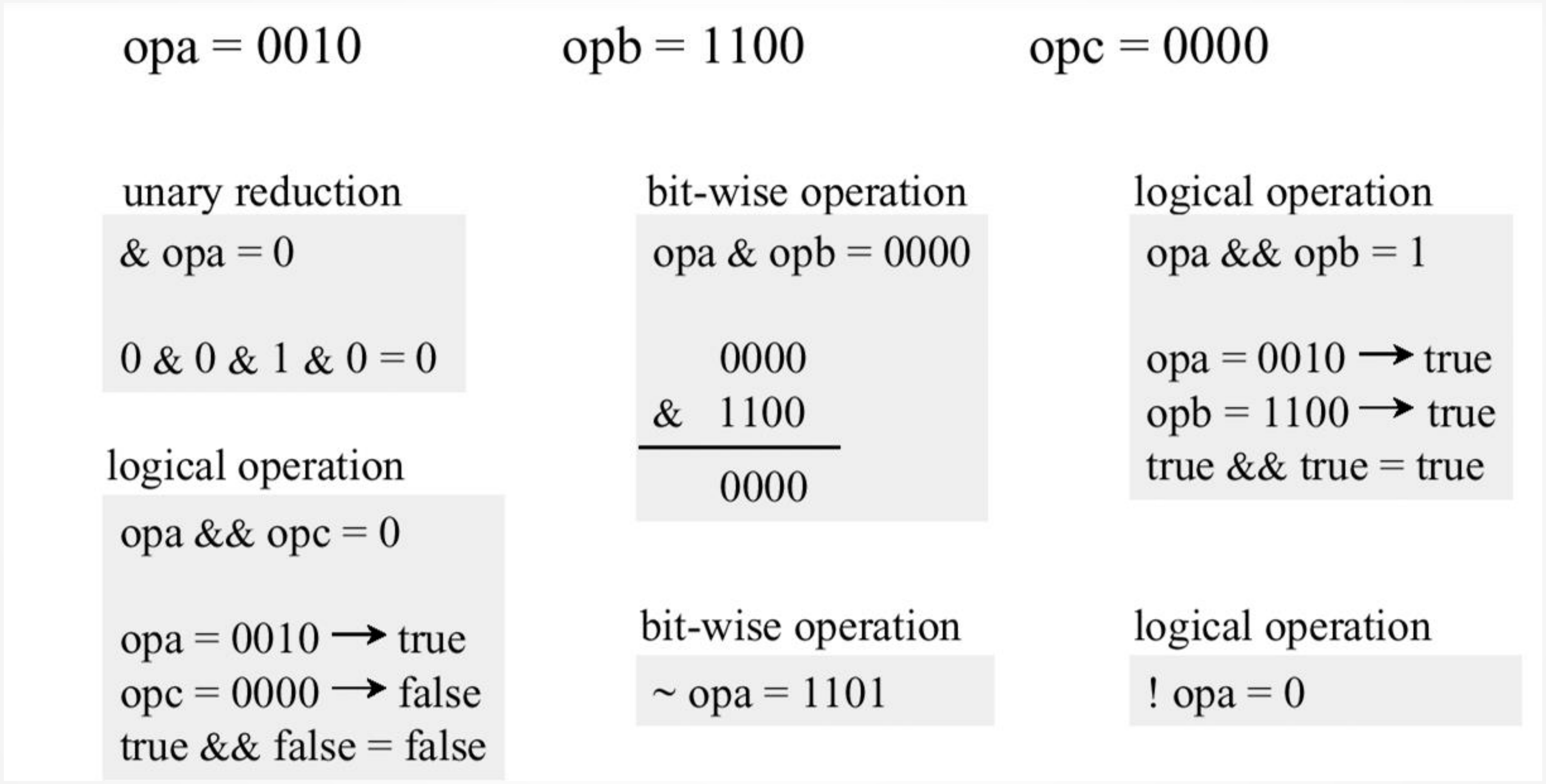
asynchronous design

OPERATORS

Verilog Operator	Name	Functional Group
[ ]	bit-select or part-select	
( )	parenthesis	
!	logical negation	Logical
~	negation	Bit-wise
&	reduction AND	Reduction
|	reduction OR	Reduction
~&	reduction NAND	Reduction
~|	reduction NOR	Reduction
^	reduction XOR	Reduction
~^ or ^~	reduction XNOR	Reduction
+	unary (sign) plus	Arithmetic
-	unary (sign) minus	Arithmetic
{ }	concatenation	Concatenation
{{ }}	replication	Replication
*	multiply	Arithmetic
/	divide	Arithmetic
%	modulus	Arithmetic
+	binary plus	Arithmetic
-	binary minus	Arithmetic
<<	shift left	Shift
>>	shift right	Shift
>	greater than	Relational
>=	greater than or equal to	Relational
<	less than	Relational
<=	less than or equal to	Relational
==	logical equality	Equality
!=	logical inequality	Equality
===	case equality	Equality
!==	case inequality	Equality
&	bit-wise AND	Bit-wise
^	bit-wise XOR	Bit-wise
^~ or ~^	bit-wise XNOR	Bit-wise
|	bit-wise OR	Bit-wise
&&	logical AND	Logical
||	logical OR	Logical
?:	conditional	Conditional

OPERATORS: SOME TRAPS?



opa = 0010 opb = 1100 opc = 0000

unary reduction

& opa = 0

0 & 0 & 1 & 0 = 0

logical operation

opa && opc = 0

opa = 0010 $\rightarrow$ true
opc = 0000 $\rightarrow$ false
true && false = false

bit-wise operation

opa & opb = 0000

```
    0000
&   1100
--------
    0000
```

bit-wise operation

~ opa = 1101

logical operation

opa && opb = 1

opa = 0010 $\rightarrow$ true
opb = 1100 $\rightarrow$ true
true && true = true

logical operation

! opa = 0

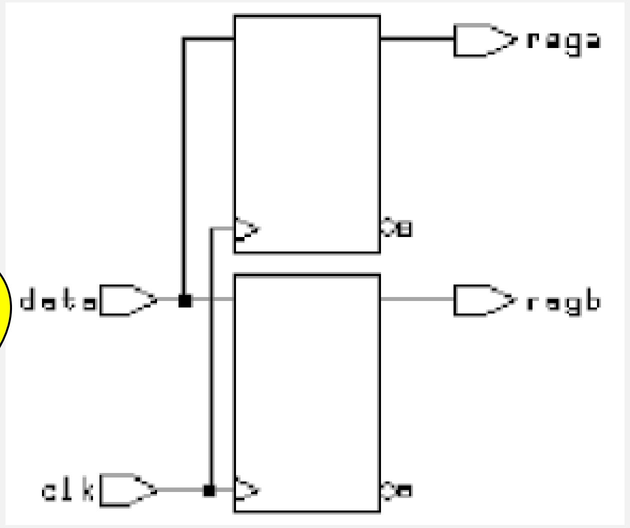
BLOCKING / NON-BLOCKING ASSIGNMENT

Blocking assignment

evaluation and assignment are immediate

```
always @(posedge clk)
begin
    rega = data;
    regb = rega;
end
```

⇒ Use for Combinational Logic (always or assign)

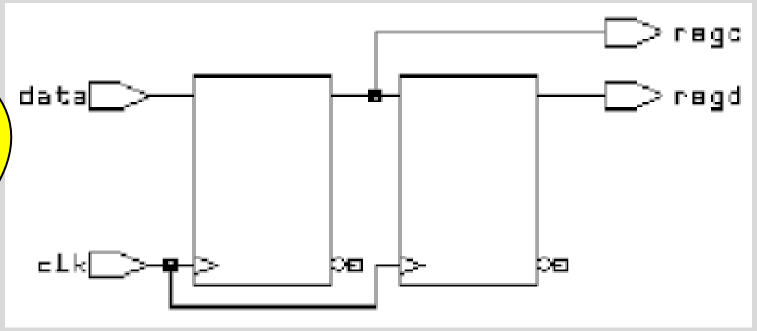


Non-Blocking Assignment

all assignments deferred until all right-hand sides have been evaluated (end of simulation timestep)

```
always @(posedge clk)
begin
    regc <= data;
    regd <= regc;
end
```

⇒ Use for Sequential Style (always)



Guidelines for efficient Verilog Coding

Separate Combinational and Sequential

Separate also Structural Description and Random Logic

- Structured: data path, XORs, MUXs
- Random logic: control logic, decoder, encoder

Use Parentheses Operands + operators

Use assign statements when possible

Guidelines for Synthesizable Verilog

Always know your Target Circuit

Often, your only allowed storage is the DFF (avoid latchs)

No case statements without default case

No if statements without an else case

A net assigned in one case must be assigned to in all cases (no implicit storage)

No loops

No initial blocks

Limited operators: + and – are the only arithmetic operators allowed

Try to avoid relational operators (>, ==) in favor of simpler logic

CLASSICAL STRUCTURES

D FLIP FLOP

```verilog
module dff(
   output [7:0] Q,  // data output
   input  [7:0] D,  // data input
   input  RST,  // Asynchronous Reset
   input  CLK); // clock input

   reg [7:0] Q;
   always @(posedge CLK or posedge RST)
   begin
      if (RST)
          Q <= 0;
      else
          Q <= D;
   end
endmodule
```

```verilog
module dff(
   output [7:0] Q,  // data output
   input  [7:0] D,  // data input
   input  RST,  // Synchronous Reset
   input  CLK); // clock input

   reg [7:0] Q;
   always @(posedge CLK)
   begin
      if (RST)
          Q <= 0;
      else
          Q <= D;
   end
endmodule
```

MULTIPLEXOR

```
module mux4_1(
   output        out,   // data output
   input  [1:0] sel,   // input select
   input        in0,   // data input 0
   input        in1,   // data input 1
   input        in2,   // data input 2
   input        in3);  // data input 3

   assign out =
      (sel==2'b00)?     in0 :
      (sel==2'b01)?     in1 :
      (sel==2'b10)?     in2 :
      (sel==2'b11)?     in3 :
      1'bX;
endmodule
```

```
module mux4_1(
   output reg   out,   // data output
   input  [1:0] sel,   // input select
   input  [3:0] in);   // data input

   always @(sel or in)
   begin
      case (sel)
         2'd0:     out = in[0];
         2'd1:     out = in[1];
         2'd2:     out = in[2];
         2'd3:     out = in[3];
         default:  out = 1'bX;
      endcase
   end
endmodule
```

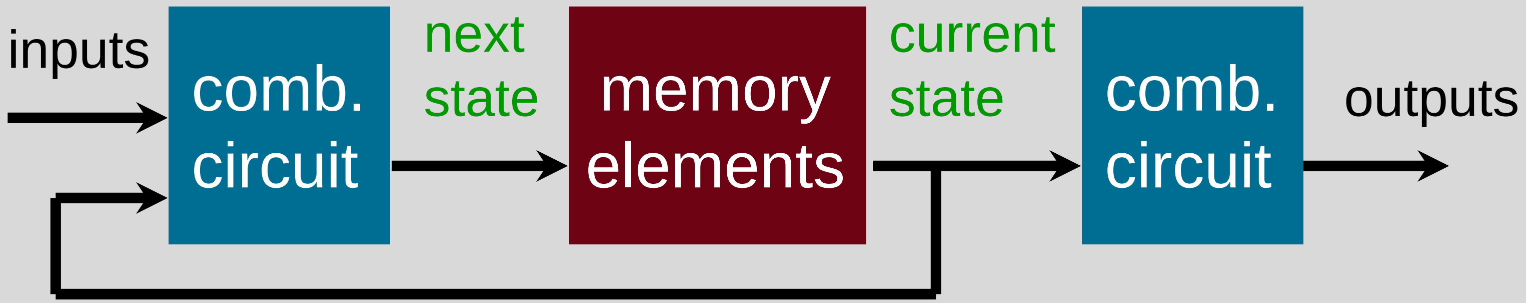
COUNTER

```
module counter(
   output [7:0] Q,  // data output
   input  RST,  // Synchronous Reset
   input  CLK,  // clock input
   input  EN);  // enable count

   reg [7:0] Q;
   always @(posedge CLK)
   begin
      if (RST)
          Q <= 8’b0;
      else if (EN)
          Q <= Q + 1;
      end
   end
endmodule
```

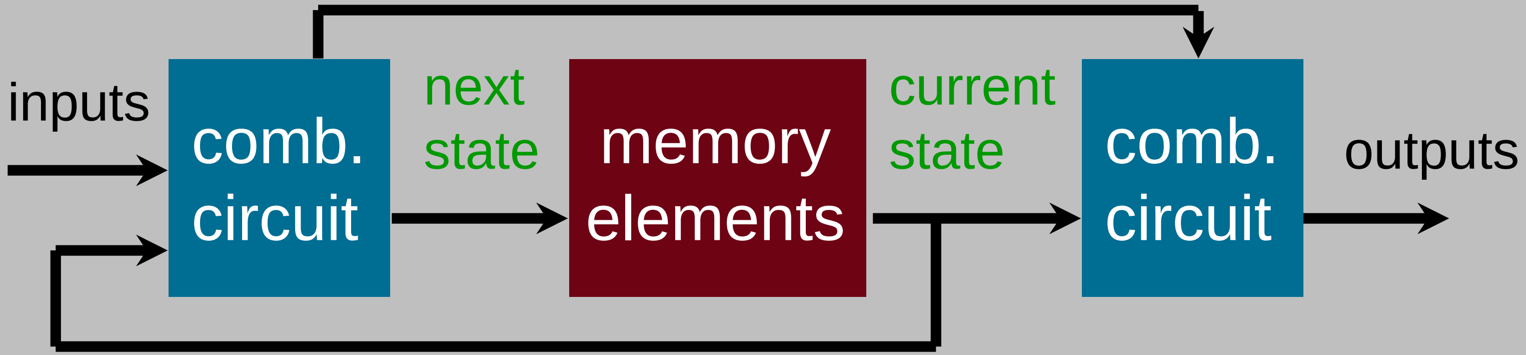
```
module counter_noloop(
   output [7:0] Q,  // data output
   input  RST,  // Synchronous Reset
   input  CLK,  // clock input
   input  EN);  // enable count

   reg [7:0] Q;
   always @(posedge CLK)
   begin
      if (RST)
          Q <= 0;
      else if ((EN) and !(&Q))
          Q <= Q + 1;
      end
   end
endmodule
```

Finite State Machine 1/2

Moore: output determined by current state only



Mealy: output determined by current state and inputs



FINITE STATE MACHINE 2/2

State Register

```
localparam IDLE=0,WAITFORB=1,DONE=2,ERROR=3;
reg [1:0] state, // Current state
          nxtState; // Next state
always @(posedge clk) begin
   if (reset) begin
      state <= IDLE; // Initial state
   end else begin
      state <= nxtState;
   end
end
```

Output Computation

```
always @(*) begin
   out = 0; // Default output value
   case (state)
      IDLE : begin
      end
      WAITFORB : begin
      end
      DONE : begin
          out = 1;
      end
      ERROR : begin
      end
   endcase
end
```

Next State Computation

```
always @(*) begin
   nxtState = state; // Default: don't move
   case (state)
      IDLE : begin
          if (B) nxtState = ERROR;
          else if (A) nxtState = WAITFORB;
      end
      WAITFORB : begin
          if (B) nxtState = DONE;
      end
      DONE : begin
      end
      ERROR : begin
      end
   endcase
end
```

VERILOG TEST BENCH

TESTBENCH'S

- **To simulate the whole environment surrounding your module:**
  - Either well targeted scenarios,
  - or (better) explore random scenarios,
  - or (even better) explore all possible scenarios

$\left.\begin{array}{l} \\ \\ \end{array}\right\}$ move to SystemVerilog ?

- **Of course, you can use non-synthesizable structures in your code**
  - For example, to include error detection code (encouraged)

As good practices, you are encouraged
to test all unitary blocks constituting the hierarchy of your module

SYSTEM AND COMPILER COMMANDS

- **Compiler Directives (always preceded by a back-quote `)**
  - `define – defines a compiler time constant or macro
  - `ifdef, `else, `endif – conditional compilation
  - `include – include the whole content of a text file
  - `timescale – set the reference time unit and time precision of your simulation.

    ```
    `timescale 100ns/1ns // only 1, 10, and 100 are legal values.
    ```
- **System Tasks (always preceded by a $)**
  - $time – returns the current simulation time
  - $display – similar to printf in C

    ```
    $display ("%d %d %d", address, sinout, cosout);
    ```
  - $monitor – print whenever any of the arguments change except $time.

    ```
    $monitor ($time, "%d %d %d", address, sinout, cosout);
    ```
  - $finish – ends simulation
  - $readmemh – load memory array from text file in hex format
  - $sdf_annotate – useful for back-annotated simulations

Timing Control in Testbench's 1/2

- **Delay (#)**
  Used to delay statement by specified amount of simulation time

```
always
begin
    #10 clk = 1;
    #10 clk = 0;
end
```

A confusing Subtlety

```
Always @(a or b)
begin
    #10 if(a) out=b;
end
```

Ten Delta cycles after a change in a or b,
check value of a before updating out (Weird!)

```
Always @(a or b)
begin
    if(a) #10 out=b;
end
```

After a change in a or b, sample
and update out, ten Delta cycles later

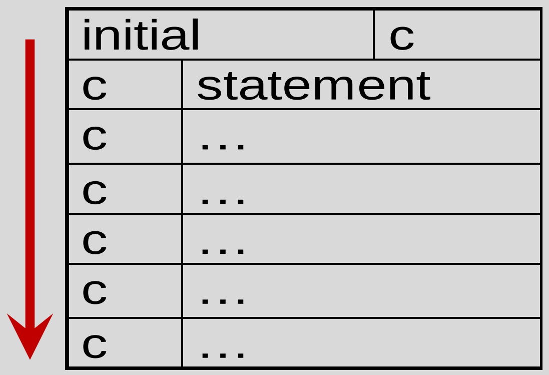
TIMING CONTROL IN TESTBENCH'S 2/2

- **Event Control (@)**
  - Delay execution until event occurs
  - Event may be single signal/expression change
  - Multiple events linked by or

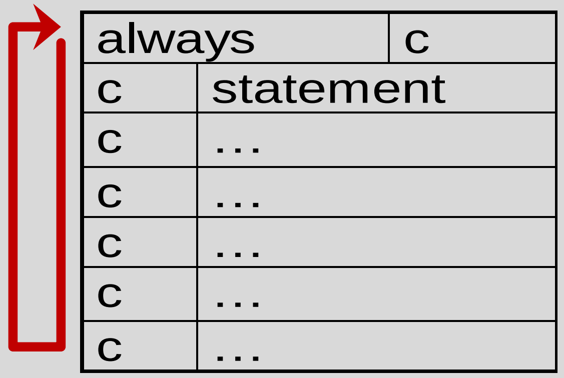
```
always
@(posedge clk)
begin
    q <= d;
end
```

```
always @(x or y)
begin
    s = x ^ y;
    c = x & y;
end
```

- **"initial" procedural block**
  - No activation list
  - Runs only once
  - Often, Initialize Simulation Environment



initial	c
c	statement
c	...
c	...
c	...
c	...
c	...



always	c
c	statement
c	...
c	...
c	...
c	...
c	...

Guidelines to avoid Strange Simulation Behaviors

- When modeling sequential logic, use non-blocking assignments
- When modeling combinational logic with always block, use blocking assignments.

  Make sure all RHS variables in block appear in @ expression
- If you mix sequential and combinational logic within the same always block

  => use nonblocking assignments
- Don’t mix blocking and nonblocking assignments in the same always block
- Don’t make assignments to same variable from more than one always block
- Don’t make assignments using #0 delays

TESTBENCH EXAMPLE

counter.v

```verilog
module counter
    #(
        parameter                DW = 4
    )
    (
        input                    clk,
        input                    rst,
        output  reg [DW-1:0]     count
    );

    always@(posedge clk, negedge rst) begin
        if(~rst) begin
            count <= {DW{1'b0}};
        end else begin
            count <= count + 1'b1;
        end
    end
endmodule
```

tb_counter.v

```verilog
`timescale 1ns/1ns
module tb_counter;
    localparam  DW = 5;
    reg             clk, rst;
    wire [DW-1:0]   cnt;
    reg [DW-1:0]    expectedCnt;

always #10 clk = ~clk; // clock generation

always@(posedge clk) begin
    if(cnt==expectedCnt)
        $write("%t: clk[%b] rst[%b] cnt[%b]\n", $time, clk, rst, cnt);
    else
        $write("error count %b != %b\n",cnt,expectedCnt);
end

always@(posedge clk, negedge rst) begin
    if(~rst) begin
        expectedCnt=0;
    end else begin
        expectedCnt = cnt+1;
    end
end

initial begin
    clk = 1'b0; rst = 1'b0;
    #100 rst = 1'b1;
    #1000 $stop;
end

counter #(.DW(DW)) dut( .clk(clk), .rst(rst), .count(cnt));

endmodule
```

DIGITAL SIMULATION

SIMULATORS

- ModelSim
- QuestaSim
- NCSim
- VCS
- Icarus + GTKWave - Free
- ISim

Simulation using Cadence NCSim/NCVerilog

Depending on the complexity of your design

- **Either 1 step with *irun***

- ***Or 3 steps with ncvlog/ncelab and call snapshots***
  - Compile your design files and also your testbench's with *ncvlog*
  - Elaborate the designs(s) with *ncelab*
  - Simulate by calling the right snapshot (*ncls* & *ncsim* tools)

ONE STEP SIMULATION WITH *irun* (1/2)

```
cflouzat> irun ./source/hdl/*.v ./source/tb/tb_counter.v -gui -access +rwc
[…]
ncsim> run
                  10: clk[1] rst[0] cnt[00000]
                  30: clk[1] rst[0] cnt[00000]
                  50: clk[1] rst[0] cnt[00000]
                  70: clk[1] rst[0] cnt[00000]
                  90: clk[1] rst[0] cnt[00000]
                 110: clk[1] rst[1] cnt[00000]
                 130: clk[1] rst[1] cnt[00001]
                 150: clk[1] rst[1] cnt[00010]
                 170: clk[1] rst[1] cnt[00011]
[…]
                 970: clk[1] rst[1] cnt[01011]
                 990: clk[1] rst[1] cnt[01100]
                1010: clk[1] rst[1] cnt[01101]
                1030: clk[1] rst[1] cnt[01110]
                1050: clk[1] rst[1] cnt[01111]
                1070: clk[1] rst[1] cnt[10000]
                1090: clk[1] rst[1] cnt[10001]
Simulation stopped via $stop(1) at time 1100 NS + 0
./tb_counter.v:29 #1000 $stop;
```

-sdf_file adc_ctrl.sdf
for back annotated simulation

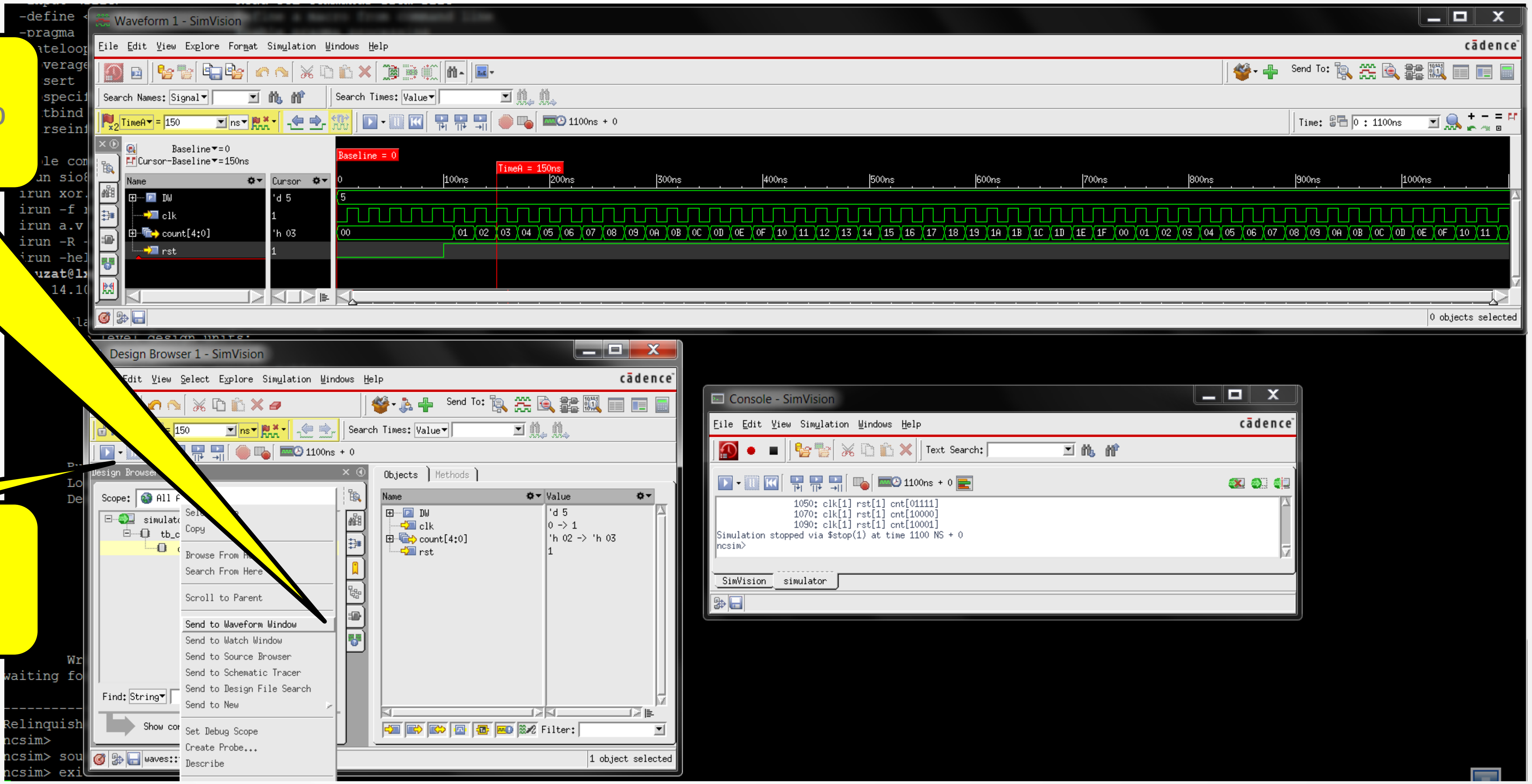
ONE STEP SIMULATION WITH *irun* (2/2)

And in case you discover a bug…

```
irun ./source/hdl/counter.v ./source/tb/tb_counter.v +nctimescale+1ns/1ns -gui -access +rwc
```

Right click on signals or modules to send signals to waveform window

Then RUN simulation, or REWIND if you added some new signals…



```
        1050: clk[1] rst[1] cnt[01111]
        1070: clk[1] rst[1] cnt[10000]
        1090: clk[1] rst[1] cnt[10001]
Simulation stopped via $stop(1) at time 1100 NS + 0
ncsim>
```

THREE STEPS SIMULATION (1/3)

First, setup hdl.var and cds.lib files:

- **cds.lib**
  - maps logical lib name to physical location
  - can include previously define 'cds.lib' file
  - example

```
# Define logical name work to physical library wrk
include $CDS_INST_DIR/tools/inca/files/cds.lib
DEFINE work ./wrk
```

- **hdl.var**
  - optional configuration file
  - to setup compiler, elaborator and simulator command line options and arguments

```
include $CDS_INST_DIR/tools/bin/files/hdl.var
DEFINE NCVLOGOPTS -messages -errormax 10
DEFINE NCELABOPTS -messages -errormax 10
```

THREE STEPS SIMULATION (2/3)

- **Compile all your design files with *ncvlog***

```
ncvlog -messages -WORK work -incdir ./source/hdl ./source/hdl/*.v
```

- **Compile all your testbench's with *ncvlog***

```
ncvlog -messages -WORK work -incdir ./source/hdl ./source/tb/*.v
```

- **Elaborate all test scenarios with *ncelab***

```
ncelab -messages -timescale 1ns/1ps -access +wc -work work work.adc_ctrl_rampgen_tb1
ncelab -messages -timescale 1ns/1ps -access +wc -work work work.adc_ctrl_rampgen_tb2
ncelab -messages -timescale 1ns/1ps -access +wc -work work work.adc_ctrl_rampgen_tb3
ncelab -messages -timescale 1ns/1ps -access +wc -work work work.adc_ctrl_rampgen_tb4
ncelab -messages -timescale 1ns/1ps -access +wc -work work work.adc_ctrl_rampgen_tb5
ncelab -messages -timescale 1ns/1ps -access +wc -work work work.adc_ctrl_rampgen_tb6
```

- **List all compilated snapshots with *ncls***

```
ncls -snap
```

- **Then call it with *ncsim***

```
ncsim -gui -update work.adc_ctrl_rampgen_tb1:module
```

THREE STEPS SIMULATION (3/3)

For back annotated simulation

- **Compile your sdf file → generates a sdf.X file**

```
ncsdfc adc_ctrl.sdf
```

- **Add the right option to ncelab stage**

```
-SDF_CMD_FILE adc_ctrl_rampgen_tb1.sdfcmd
```

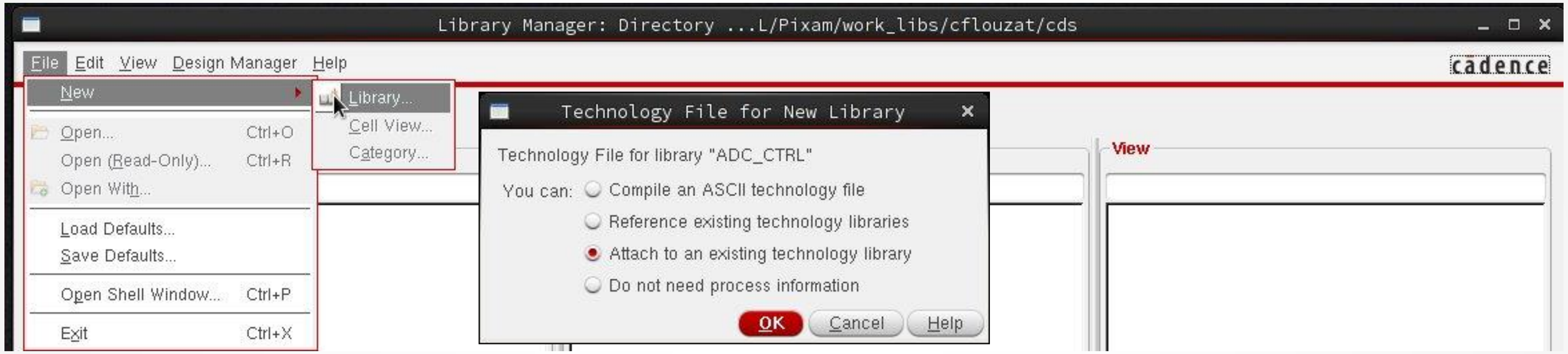
- **With** `adc_ctrl_rampgen_tb1.sdfcmd` **containing the text below**

```
// SDF command file adc_ctrl_rampgen_tb1.sdfcmd
COMPILED_SDF_FILE = “adc_ctrl.sdf.X”,
SCOPE = adc_ctrl_rampgen_tb1.uut,
MTM_CONTROL = “TYPICAL”; // or MAXIMUM or MINIMUM
// END OF FILE: adc_ctrl_rampgen_tb1.sdfcmd
```

IMPORT FILES INTO VIRTUOSO LIBRARY

IMPORT VERILOG FILES (1/2)

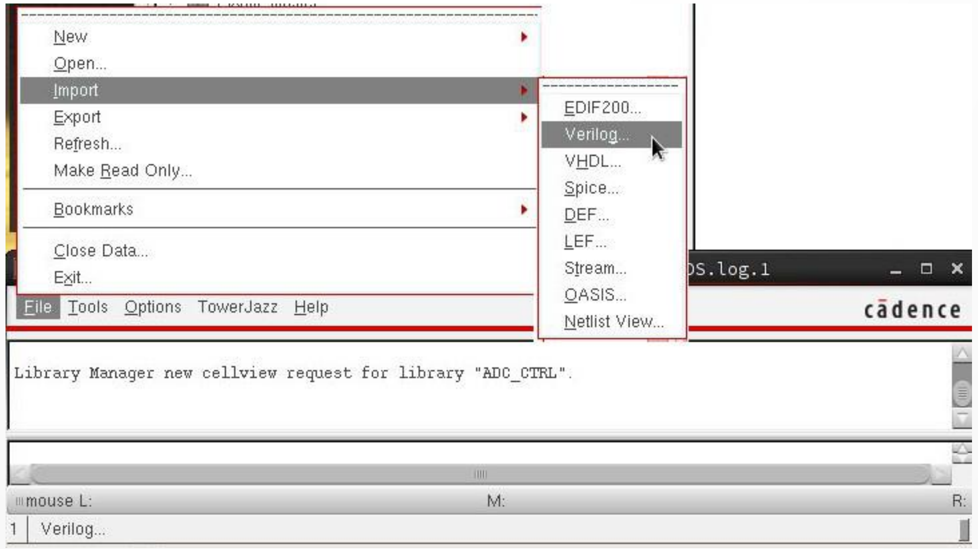
Create new Library


Library Manager: Directory ...L/Pixam/work_libs/cflouzat/cds
File Edit View Design Manager Help
cadence
New
Open... Ctrl+O
Open (Read-Only)... Ctrl+R
Open With...
Load Defaults...
Save Defaults...
Open Shell Window... Ctrl+P
Exit Ctrl+X
Library...
Cell View...
Category...
View
Technology File for New Library
OK
Cancel
Help

Technology File for library "ADC_CTRL"

You can:
- ○ Compile an ASCII technology file
- ○ Reference existing technology libraries
- ● Attach to an existing technology library
- ○ Do not need process information

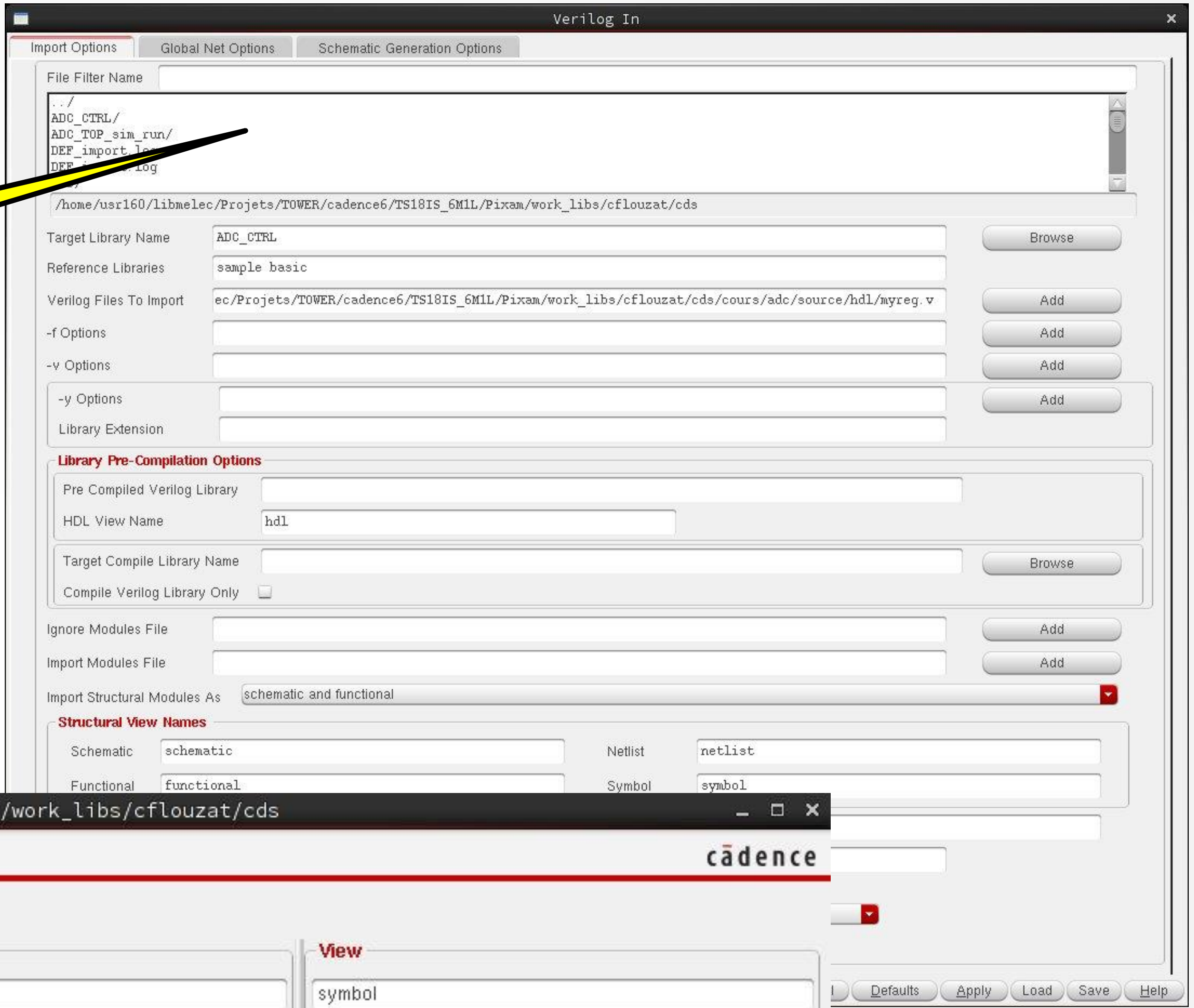
Virtuoso > Import Verilog



```
Library Manager new cellview request for library "ADC_CTRL".
```

```
1 | Verilog...
```

Import Verilog Files (2/2)



File Filter Name:

```
../
ADC_CTRL/
ADC_TOP_sim_run/
DEF_import.lo
DEF_     .log
```

/home/usr160/libmelec/Projets/TOWER/cadence6/TS18IS_6M1L/Pixam/work_libs/cflouzat/cds

Target Library Name	ADC_CTRL
Reference Libraries	sample basic
Verilog Files To Import	ec/Projets/TOWER/cadence6/TS18IS_6M1L/Pixam/work_libs/cflouzat/cds/cours/adc/source/hdl/myreg.v
-f Options	
-v Options	
-y Options	
Library Extension	

Library Pre-Compilation Options

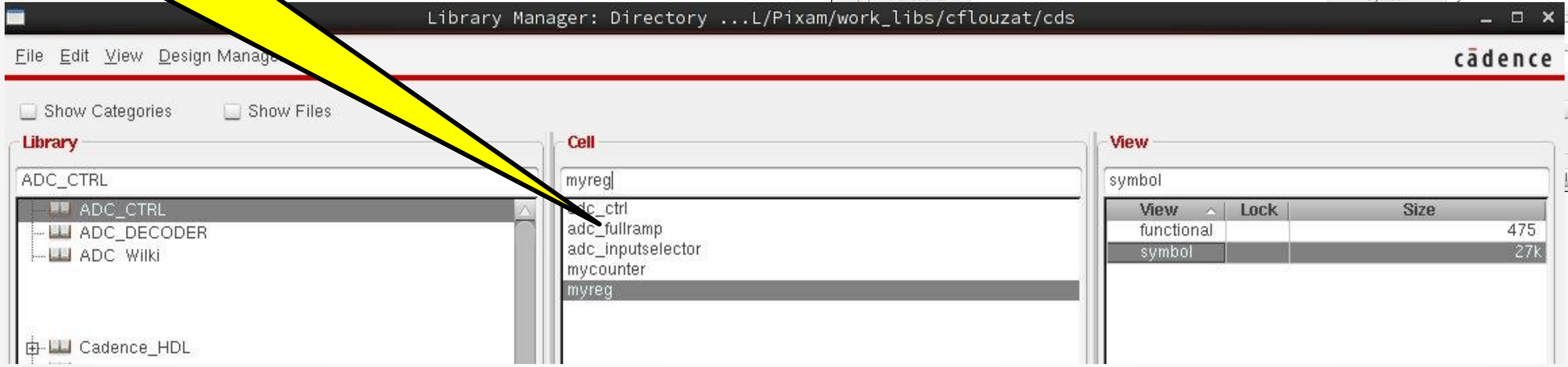
Pre Compiled Verilog Library	
HDL View Name	hdl
Target Compile Library Name	
Compile Verilog Library Only	☐
Ignore Modules File	
Import Modules File	
Import Structural Modules As	schematic and functional

Structural View Names

Schematic	schematic	Netlist	netlist
Functional	functional	Symbol	symbol

Select Files here

Your modules were added


Library Manager: Directory ...L/Pixam/work_libs/cflouzat/cds
File
Edit
View
Design Manage
cādence
Show Categories
Show Files

Library: ADC_CTRL

- ADC_CTRL
- ADC_DECODER
- ADC_Wilki
- Cadence_HDL

Cell: myreg

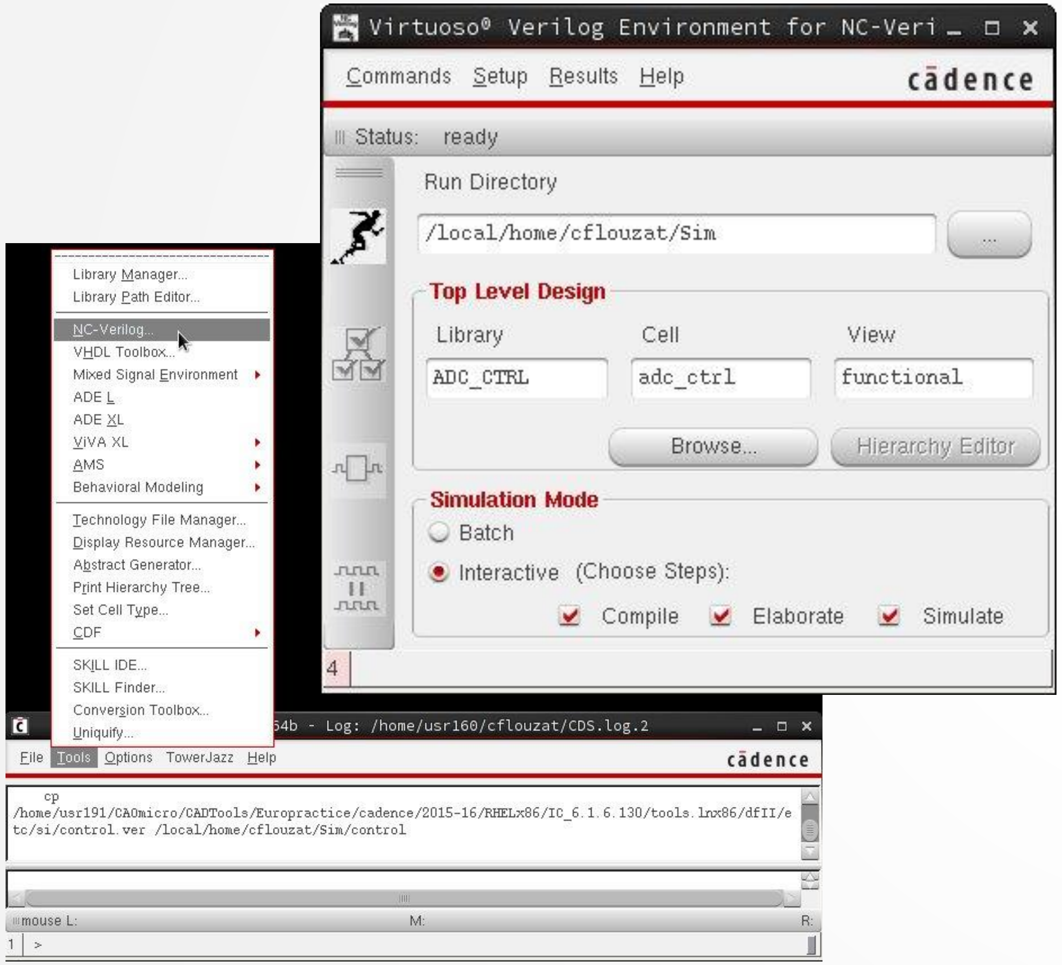
- adc_ctrl
- adc_fullramp
- adc_inputselector
- mycounter
- myreg

View: symbol

View	Lock	Size
functional		475
symbol		27k

START NC-VERILOG INTERFACE

Virtuoso > Tools > NC-Verilog



Virtuoso® Verilog Environment for NC-Veri

Status: ready

Run Directory: /local/home/cflouzat/Sim

Top Level Design

Library	Cell	View
ADC_CTRL	adc_ctrl	functional

Simulation Mode

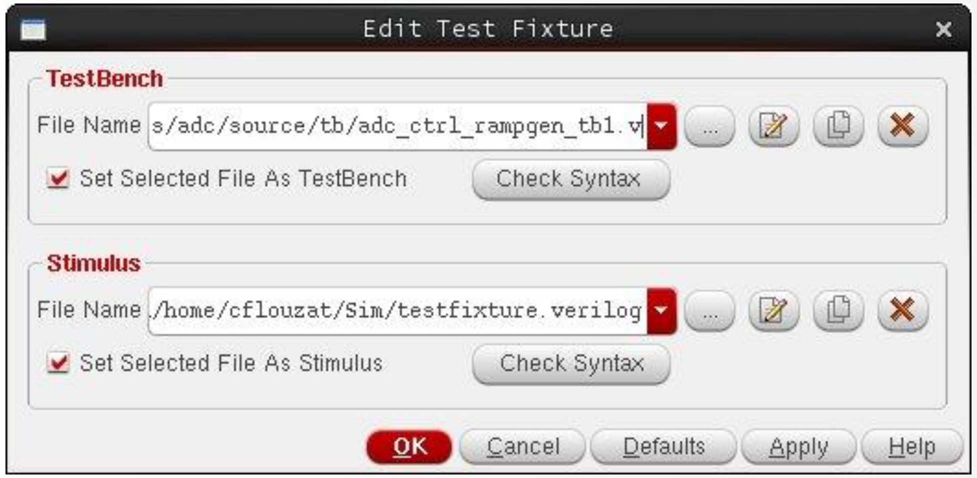
Batch [✓ Interactive] (Choose Steps):

✓ Compile ✓ Elaborate ✓ Simulate

64b - Log: /home/usr160/cflouzat/CDS.log.2

```
    cp
/home/usr191/CAOmicro/CADTools/Europractice/cadence/2015-16/RHELx86/IC_6.1.6.130/tools.lnx86/dfII/e
tc/si/control.ver /local/home/cflouzat/Sim/control
```

Initialize Design
Generate NetList
Ten select your testbench



Edit Test Fixture

TestBench

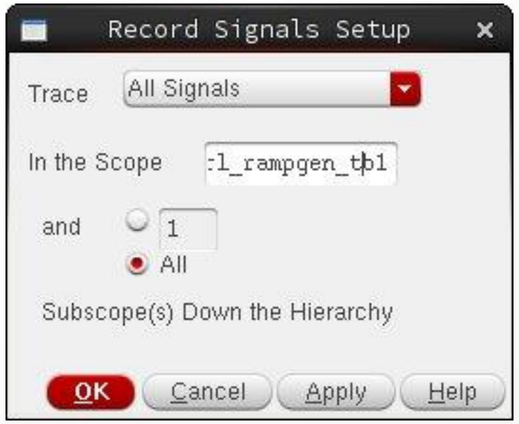
File Name s/adc/source/tb/adc_ctrl_rampgen_tb1.v

✓ Set Selected File As TestBench

Stimulus

File Name /home/cflouzat/Sim/testfixture.verilog

✓ Set Selected File As Stimulus



Record Signals Setup

Trace: All Signals

In the Scope: :l_rampgen_tb1

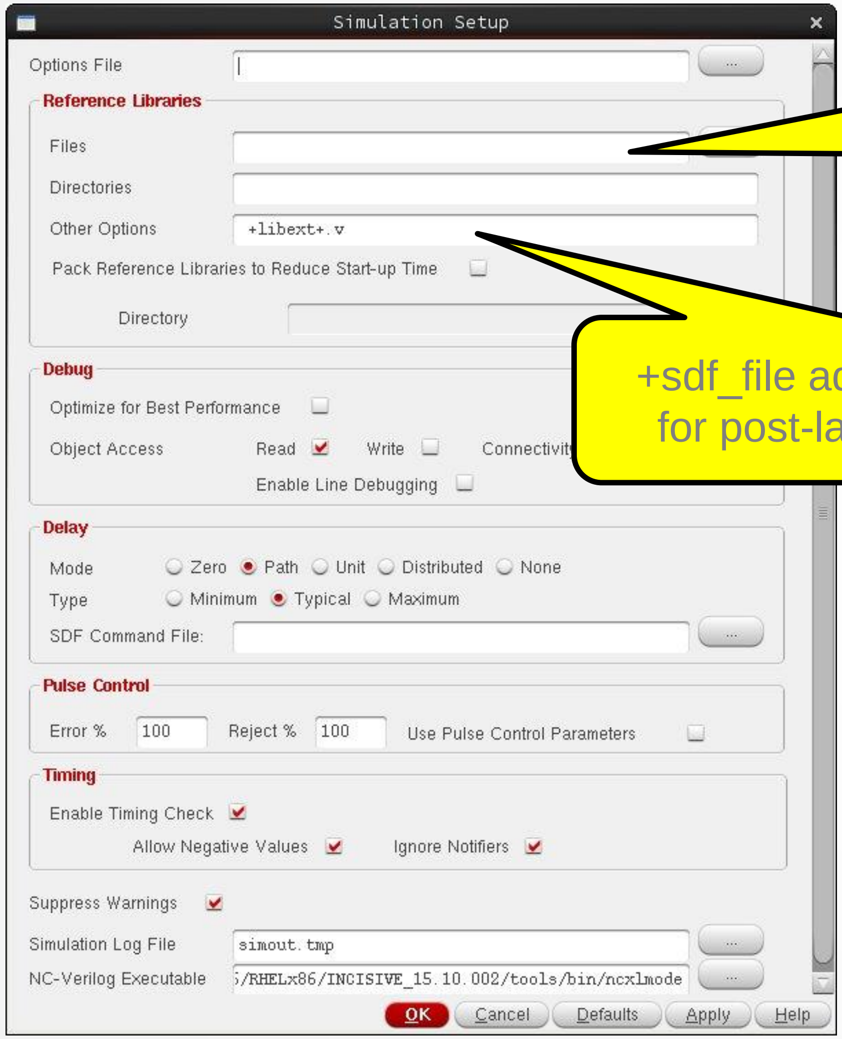
and 1 [✓ All]

Subscope(s) Down the Hierarchy

Select the signals you would store

START NC-VERILOG INTERFACE

Simulation Setup



Simulation Setup

Options File: |

Reference Libraries

Files:

Directories:

Other Options: +libext+.v

Pack Reference Libraries to Reduce Start-up Time [ ]

Directory:

Debug

Optimize for Best Performance [ ]

Object Access: [✓ Read] Write [ ] Connectivit

Enable Line Debugging [ ]

Delay

Mode: Zero [✓ Path] Unit Distributed None

Type: Minimum [✓ Typical] Maximum

SDF Command File:

Pulse Control

Error % 100 Reject % 100 Use Pulse Control Parameters [ ]

Timing

Enable Timing Check [✓]

Allow Negative Values [✓] Ignore Notifiers [✓]

Suppress Warnings [✓]

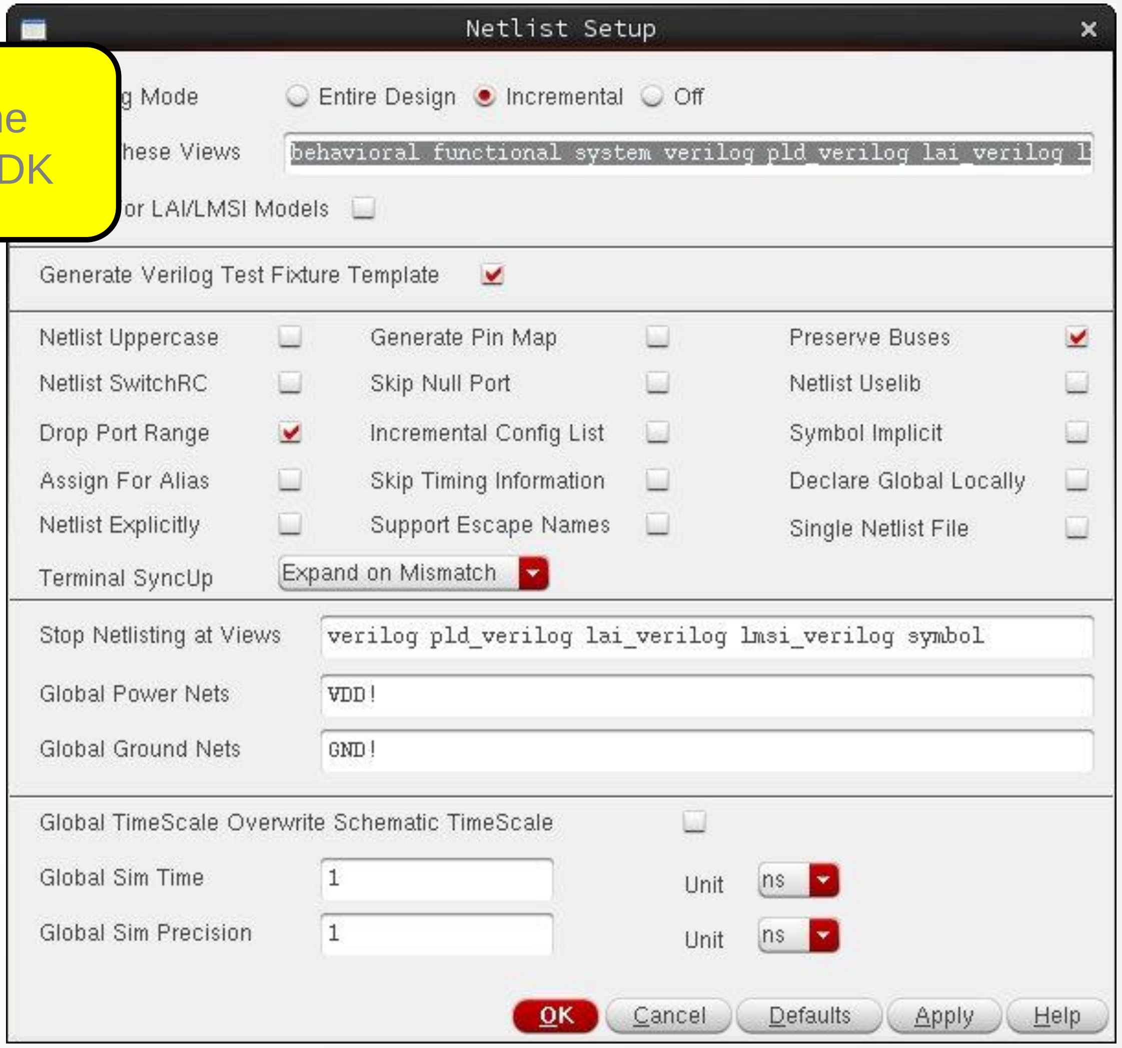
Simulation Log File: simout.tmp

NC-Verilog Executable: ;/RHELx86/INCISIVE_15.10.002/tools/bin/ncxlmode

Here you can add the All_StdCells.v from PDK

+sdf_file adc_ctrl_signoff.sdf for post-layout simulations

Netlister Options



Netlist Setup

g Mode: Entire Design [✓ Incremental] Off

hese Views: behavioral functional system verilog pld_verilog lai_verilog l

or LAI/LMSI Models [ ]

Generate Verilog Test Fixture Template [✓]

Netlist Uppercase	[ ]	Generate Pin Map	[ ]	Preserve Buses	[✓]
Netlist SwitchRC	[ ]	Skip Null Port	[ ]	Netlist Uselib	[ ]
Drop Port Range	[✓]	Incremental Config List	[ ]	Symbol Implicit	[ ]
Assign For Alias	[ ]	Skip Timing Information	[ ]	Declare Global Locally	[ ]
Netlist Explicitly	[ ]	Support Escape Names	[ ]	Single Netlist File	[ ]

Terminal SyncUp: Expand on Mismatch

Stop Netlisting at Views: verilog pld_verilog lai_verilog lmsi_verilog symbol

Global Power Nets: VDD!

Global Ground Nets: GND!

Global TimeScale Overwrite Schematic TimeScale [ ]

Global Sim Time: 1 Unit: ns

Global Sim Precision: 1 Unit: ns

MIXED SIMULATION

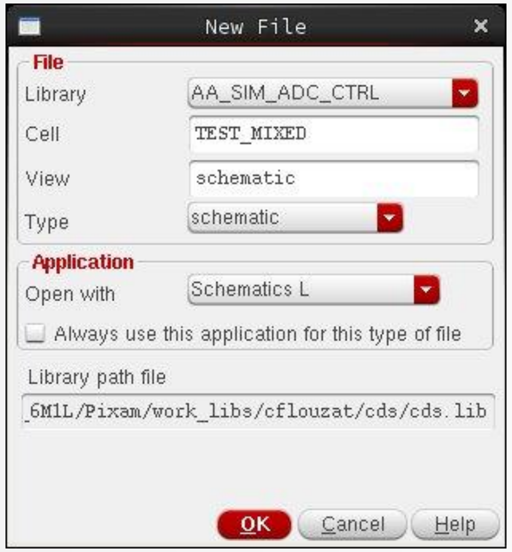
METHODOLOGY

- **Can be very slow**
  - You’d better double check in full digital before moving to mixed simulation
  - Often you work with a simple model for the digital blocs as a first approach
  - Add more and more details (simulation time grows up)

- **Possibility to reuse your digital testbench**

DRAW MIXED SCHEMATICS TEST BENCH

Create new Library



New File

File

Library	AA_SIM_ADC_CTRL
Cell	TEST_MIXED
View	schematic
Type	schematic

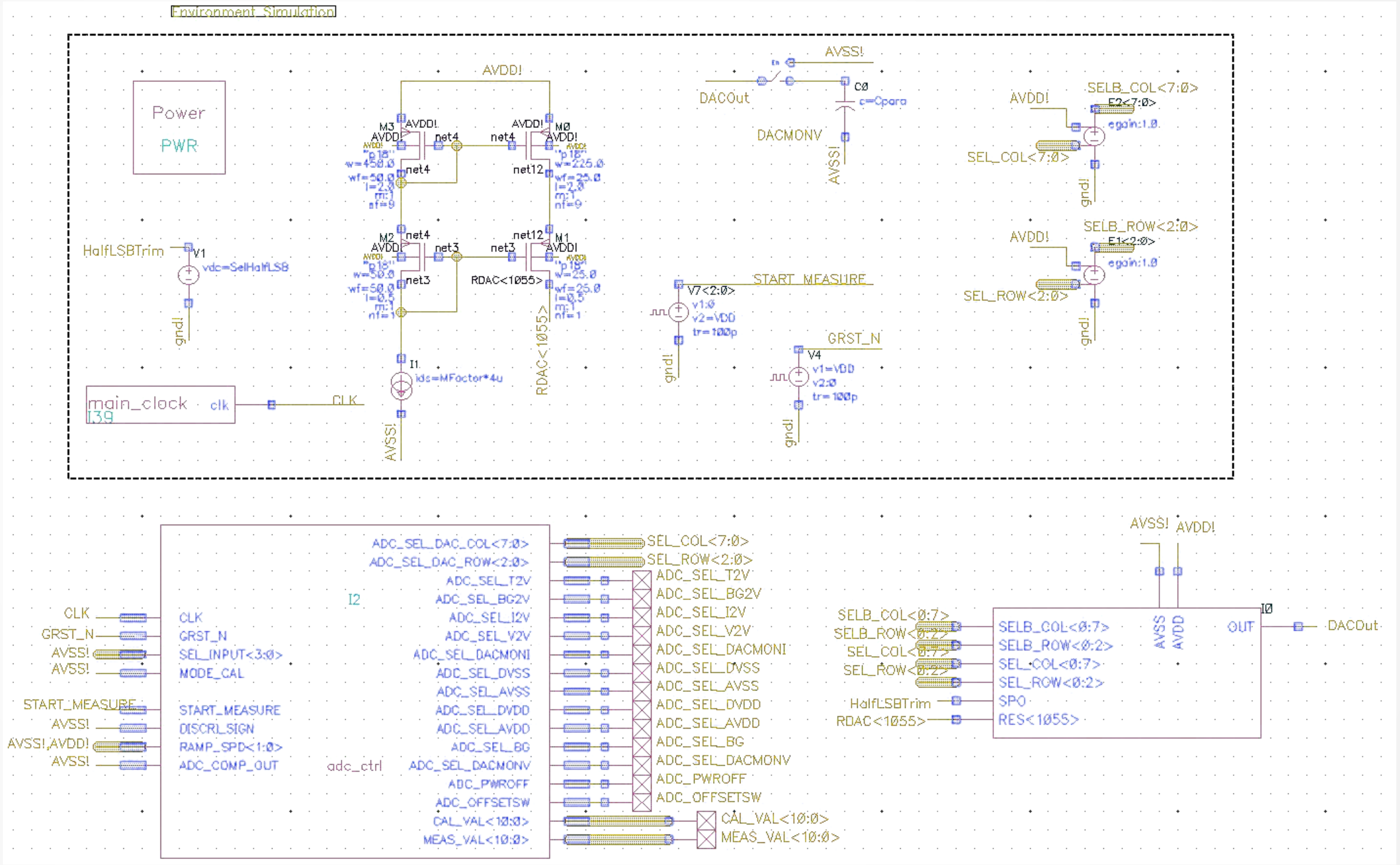
Application

Open with	Schematics L

Always use this application for this type of file

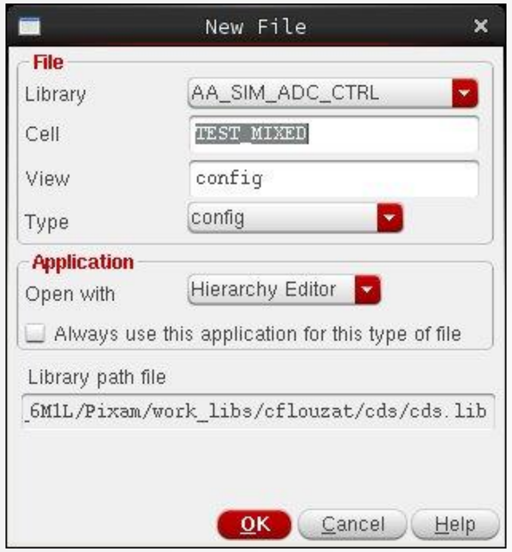
Library path file

_6M1L/Pixam/work_libs/cflouzat/cds/cds.lib



and draw
new schematics

CREATE A CONFIG VIEW



New File

File

Library	AA_SIM_ADC_CTRL
Cell	TEST_MIXED
View	config
Type	config

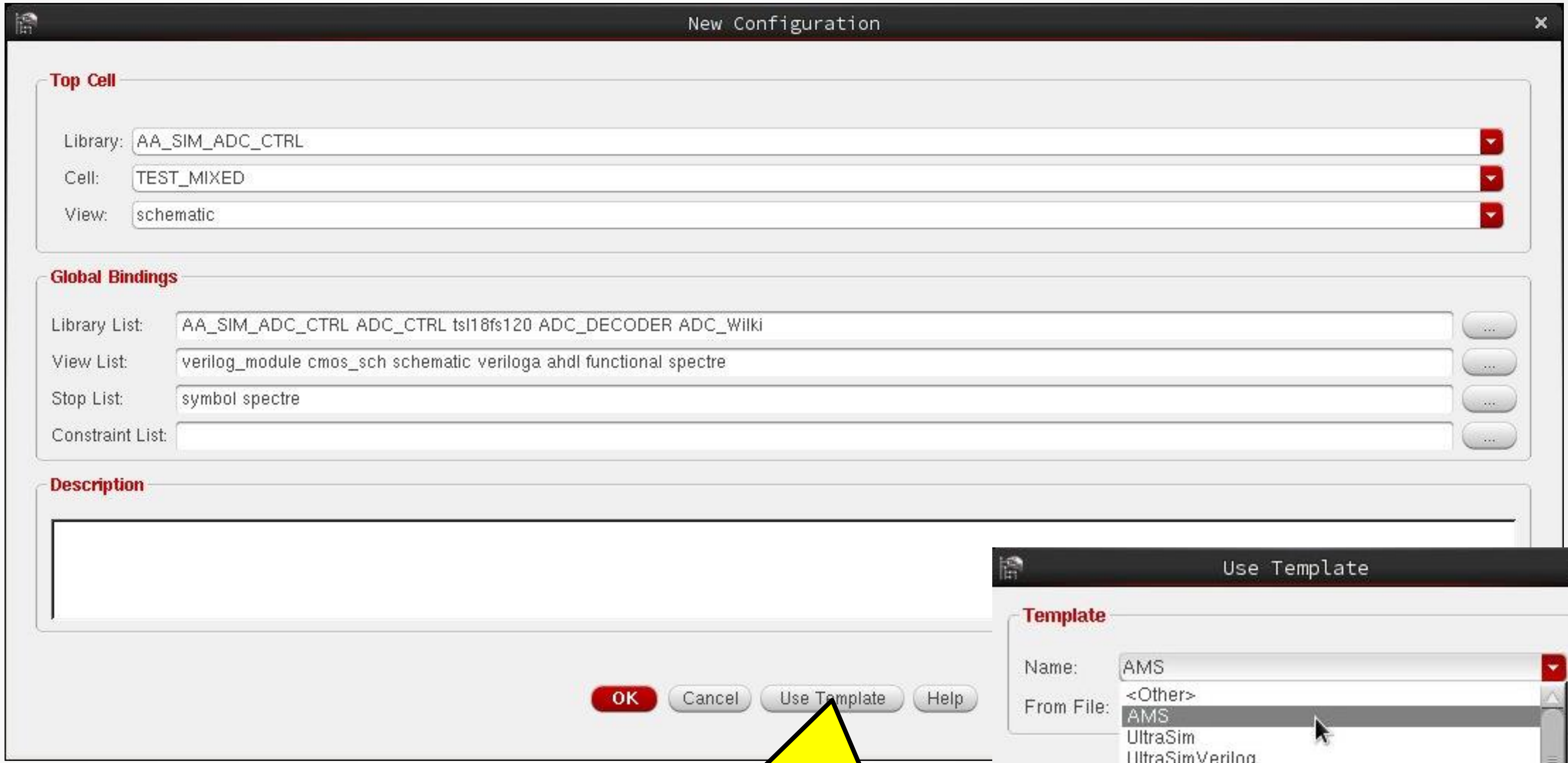
Application

Open with: Hierarchy Editor

☐ Always use this application for this type of file

Library path file

_6M1L/Pixam/work_libs/cflouzat/cds/cds.lib



New Configuration

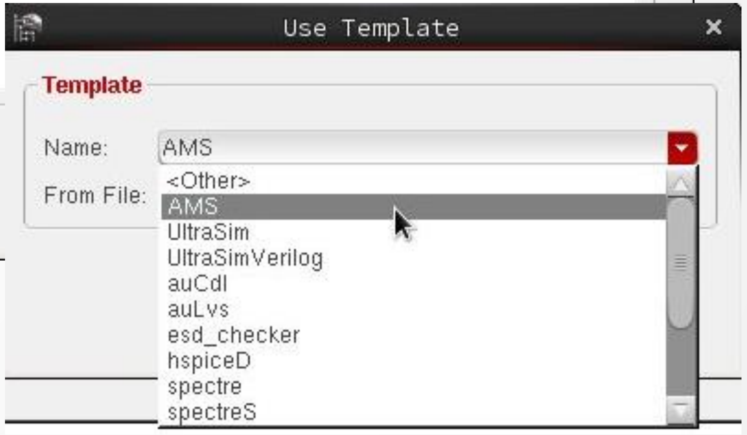
Top Cell

Library:	AA_SIM_ADC_CTRL
Cell:	TEST_MIXED
View:	schematic

Global Bindings

Library List:	AA_SIM_ADC_CTRL ADC_CTRL tsl18fs120 ADC_DECODER ADC_Wilki
View List:	verilog_module cmos_sch schematic veriloga ahdl functional spectre
Stop List:	symbol spectre
Constraint List:	

Description



Use Template

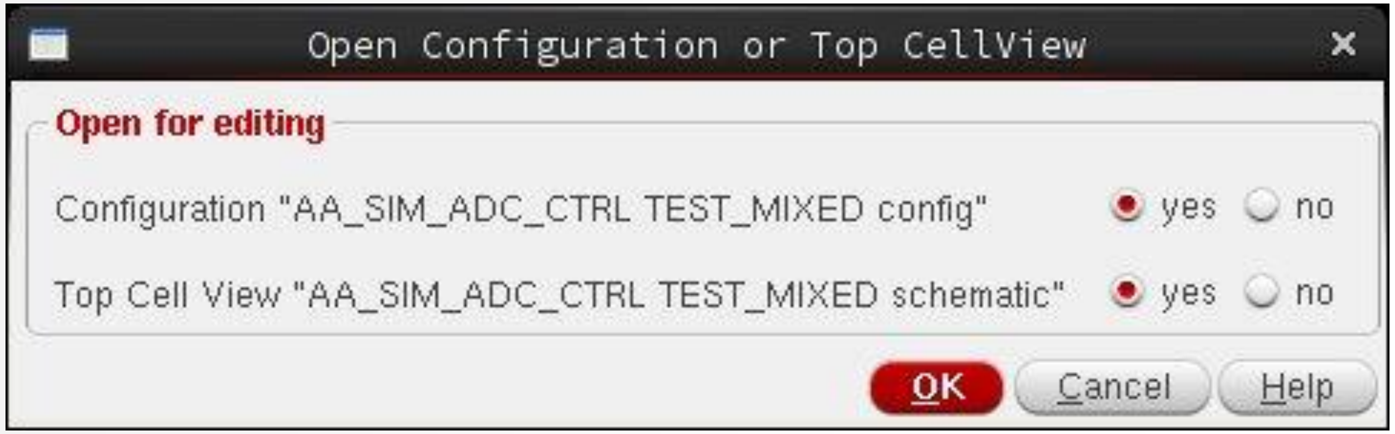
Template

Name: AMS

From File:

Use Template

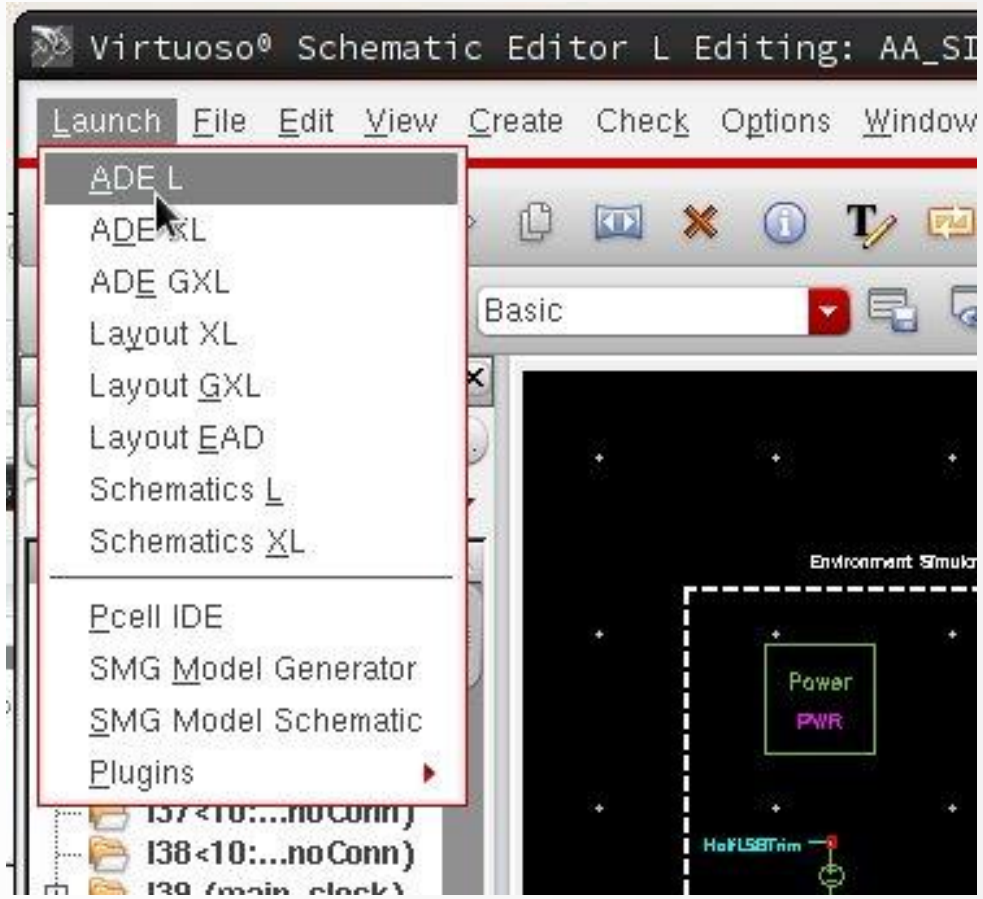
OPEN CONFIG VIEW AND START ADE L


Open Configuration or Top CellView

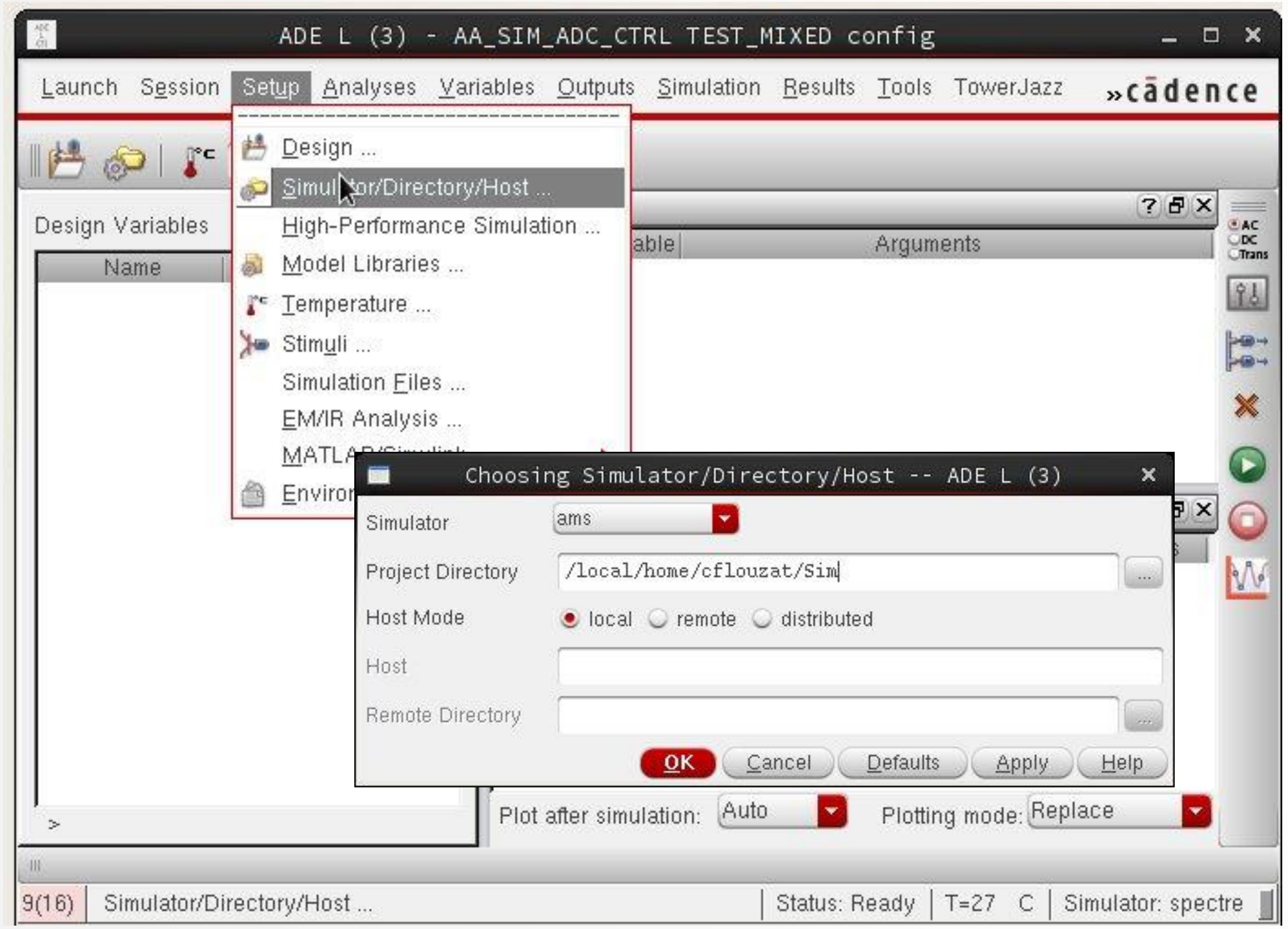
Open for editing

Configuration "AA_SIM_ADC_CTRL TEST_MIXED config" [✓ yes] no

Top Cell View "AA_SIM_ADC_CTRL TEST_MIXED schematic" [✓ yes] no



Configure Simulation (1/2)

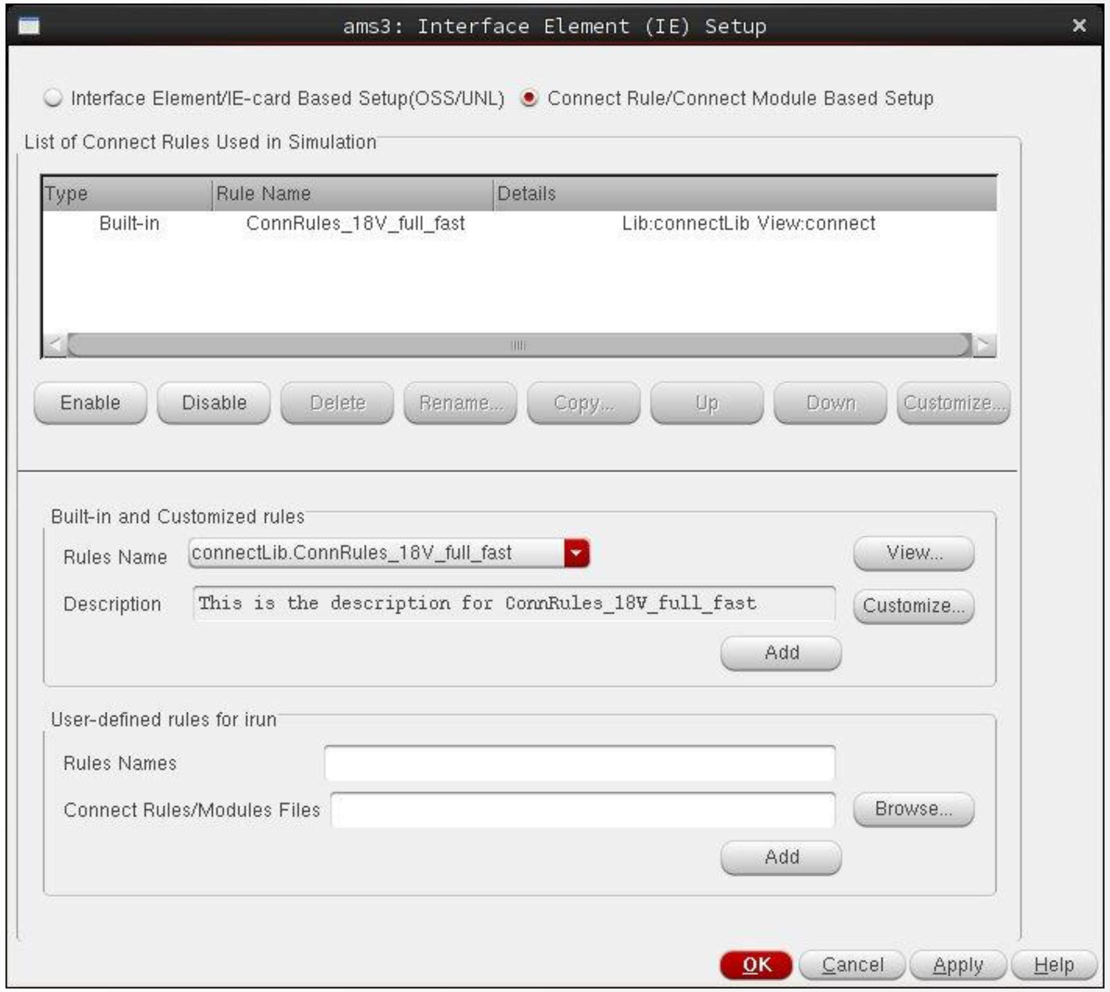

ADE L (3) - AA_SIM_ADC_CTRL TEST_MIXED config
Launch Session Setup Analyses Variables Outputs Simulation Results Tools TowerJazz
»cādence
Design ...
Simulator/Directory/Host ...
High-Performance Simulation ...
Model Libraries ...
Temperature ...
Stimuli ...
Simulation Files ...
EM/IR Analysis ...
MATLAB/Simulink
Environ
Design Variables
Name
Arguments
OK Cancel Defaults Apply Help

Choosing Simulator/Directory/Host -- ADE L (3)

Simulator	ams
Project Directory	/local/home/cflouzat/Sim
Host Mode	[✓ local] remote distributed
Host	
Remote Directory	

Plot after simulation: Auto   Plotting mode: Replace

9(16) Simulator/Directory/Host ... | Status: Ready | T=27 C | Simulator: spectre


ams3: Interface Element (IE) Setup
Enable Disable Delete Rename... Copy... Up Down Customize...
View... Customize... Add
Browse... Add
OK Cancel Apply Help

Interface Element/IE-card Based Setup(OSS/UNL) [✓ Connect Rule/Connect Module Based Setup]

List of Connect Rules Used in Simulation

Type	Rule Name	Details
Built-in	ConnRules_18V_full_fast	Lib:connectLib View:connect

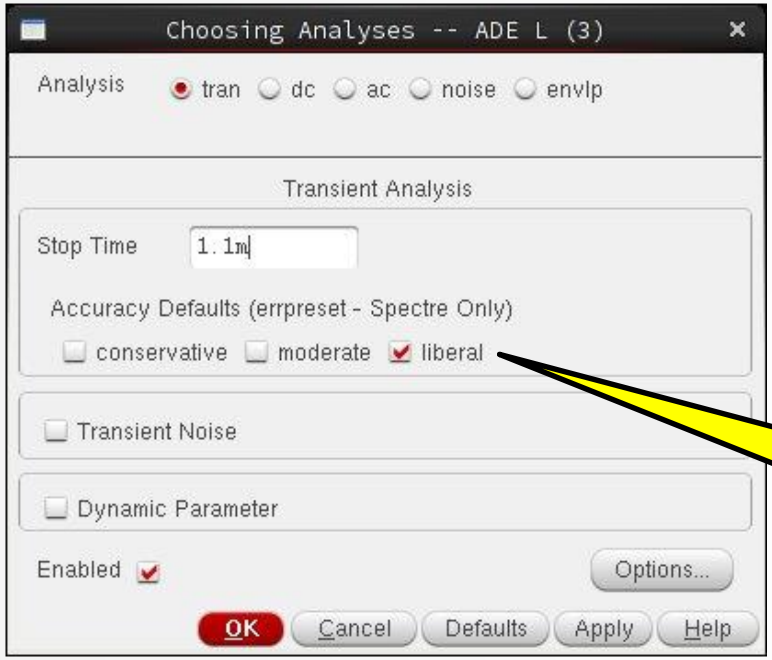
Built-in and Customized rules

Rules Name	connectLib.ConnRules_18V_full_fast
Description	This is the description for ConnRules_18V_full_fast

User-defined rules for irun

Rules Names	
Connect Rules/Modules Files	

Configure Simulation (2/2)



Choosing Analyses -- ADE L (3)

Analysis [✓ tran] dc ac noise envlp

Transient Analysis

Stop Time: 1.1m

Accuracy Defaults (errpreset - Spectre Only)

conservative moderate [✓ liberal]

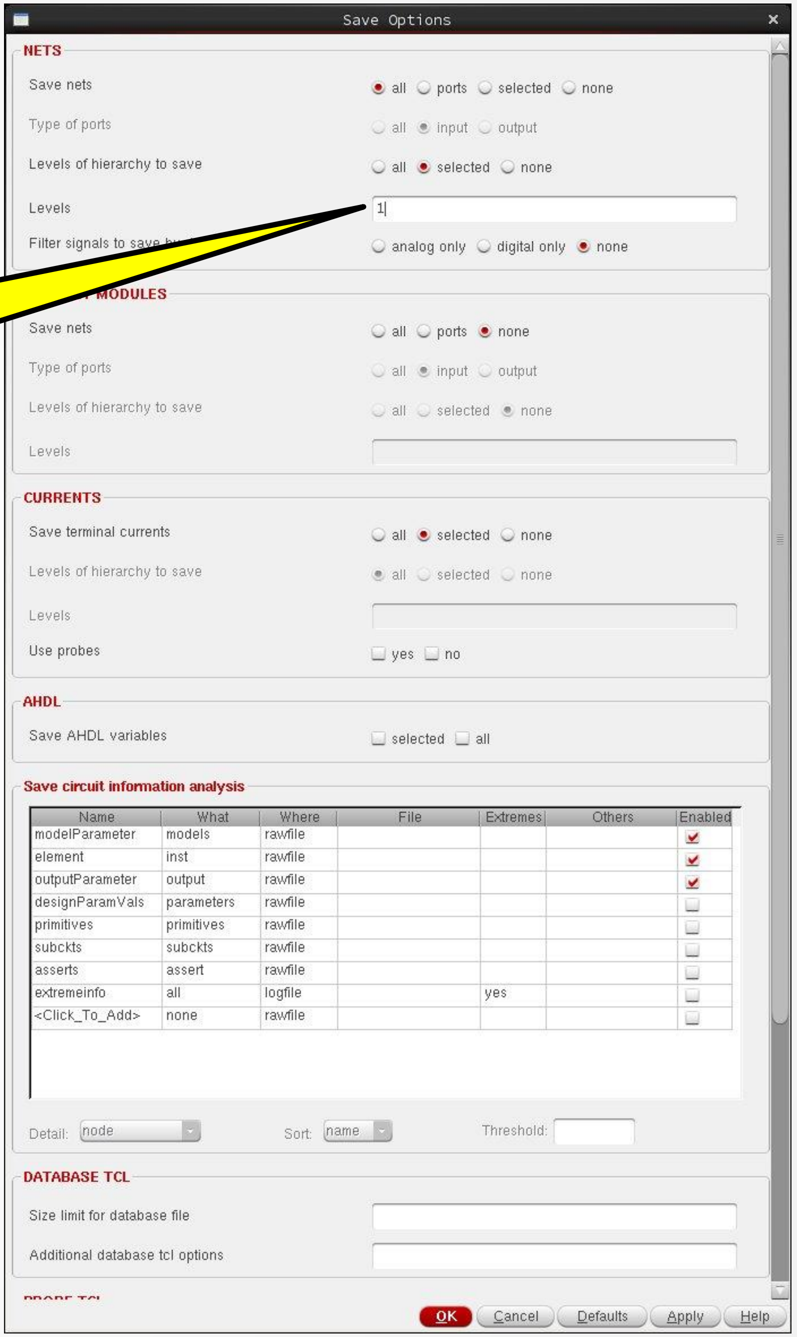
Transient Noise

Dynamic Parameter

Enabled [✓]

Set detail level

Save all signals in hierarchical levels up to.. 1



Save Options

NETS

Save nets: [✓ all] ports selected none

Type of ports: all [✓ input] output

Levels of hierarchy to save: all [✓ selected] none

Levels: 1

Filter signals to save by: analog only digital only [✓ none]

MODULES

Save nets: all ports [✓ none]

Type of ports: all [✓ input] output

Levels of hierarchy to save: all selected [✓ none]

Levels:

CURRENTS

Save terminal currents: all [✓ selected] none

Levels of hierarchy to save: [✓ all] selected none

Levels:

Use probes: yes no

AHDL

Save AHDL variables: selected all

Save circuit information analysis

Name	What	Where	File	Extremes	Others	Enabled
modelParameter	models	rawfile				✓
element	inst	rawfile				✓
outputParameter	output	rawfile				✓
designParamVals	parameters	rawfile				
primitives	primitives	rawfile				
subckts	subckts	rawfile				
asserts	assert	rawfile				
extremeinfo	all	logfile		yes		
<Click_To_Add>	none	rawfile				

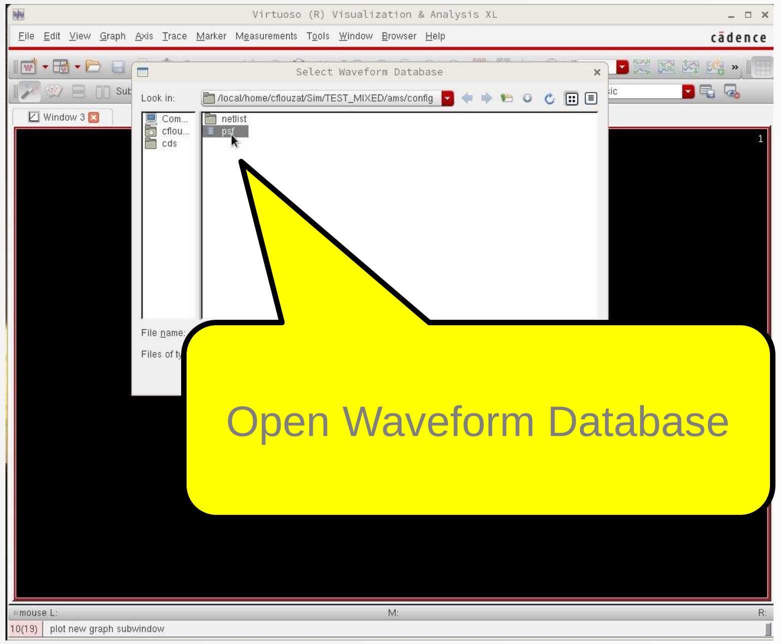
Detail: node Sort: name Threshold:

DATABASE TCL

Size limit for database file:

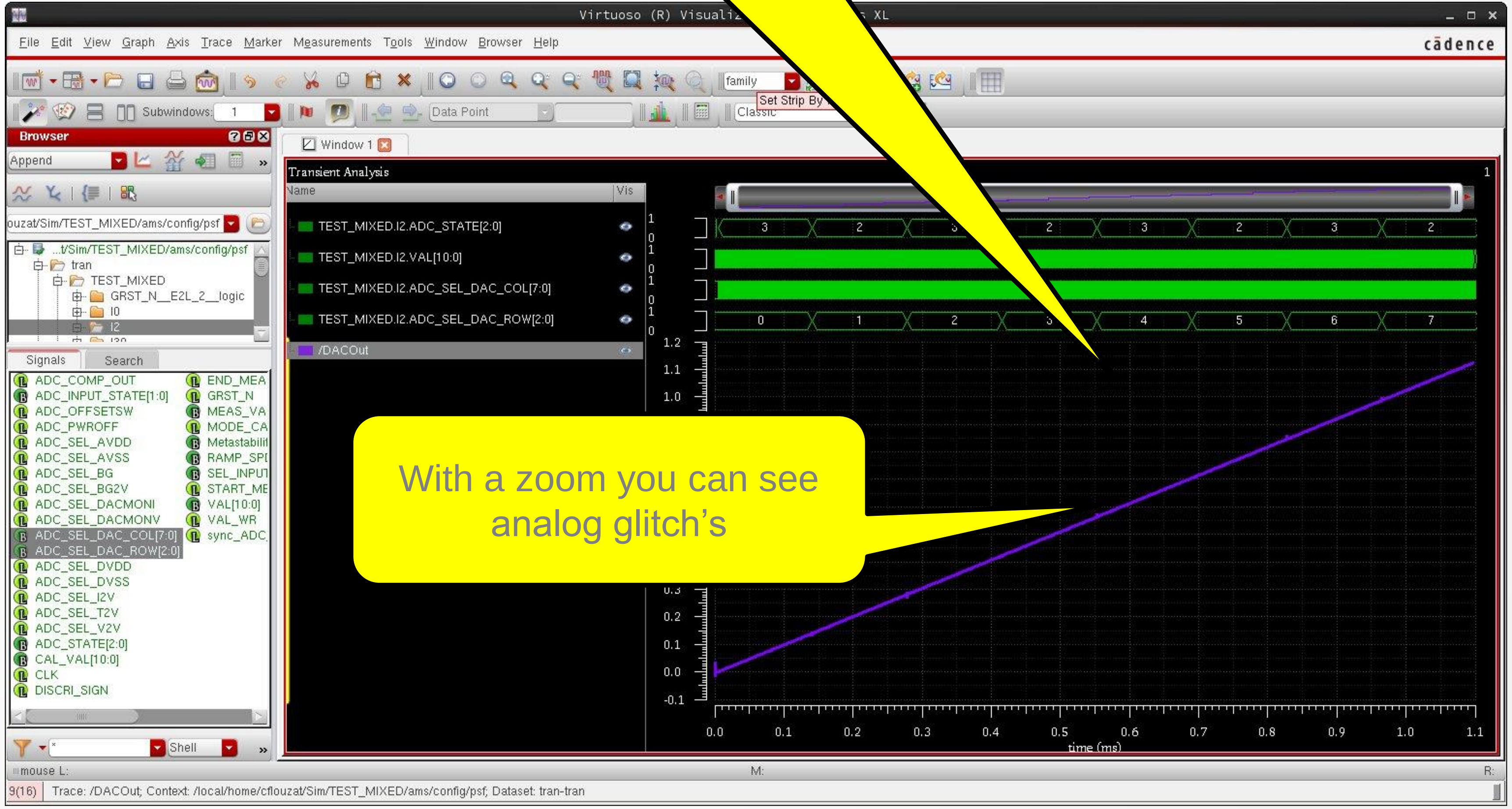
Additional database tcl options:

TOOLS – WAVEFORM VIEWING


Virtuoso (R) Visualization & Analysis XL
File Edit View Graph Axis Trace Marker Measurements Tools Window Browser Help
cādence
Select Waveform Database
Look in:
/local/home/cflouzat/Sim/TEST_MIXED/ams/config
netlist
psf
Window 3
File name:
Files of ty
mouse L:
M:
R:

10(19) plot new graph subwindow

Open Waveform Database



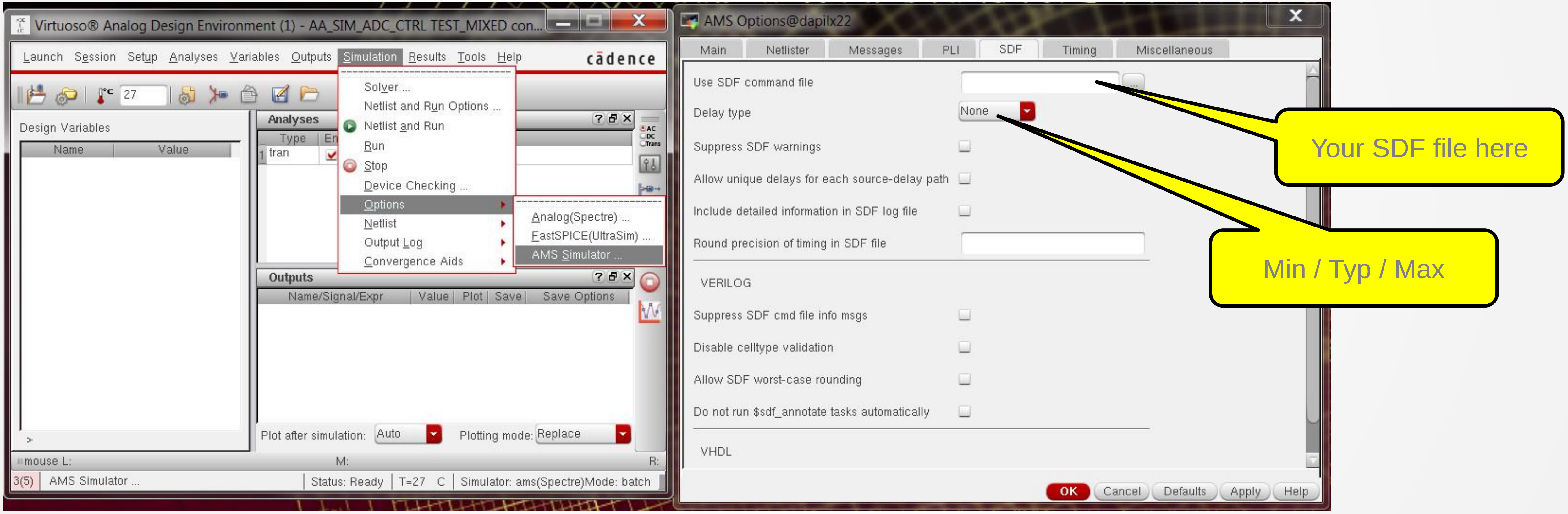
9(16) Trace: /DACOut; Context: /local/home/cflouzat/Sim/TEST_MIXED/ams/config/psf; Dataset: tran-tran

You can watch both
Digital signals
And Analog outputs

With a zoom you can see
analog glitch's

WHAT'S NEXT?

- Move to Gate-Level Synthesis using the SDF file after synthesis
- Move to simulation with Placed and Routed modules



Virtuoso® Analog Design Environment (1) - AA_SIM_ADC_CTRL TEST_MIXED con...
Launch Session Setup Analyses Variables Outputs Simulation Results Tools Help
cādence
Solver ...
Netlist and Run Options ...
Netlist and Run
Run
Stop
Device Checking ...
Options
Netlist
Output Log
Convergence Aids
Analog(Spectre) ...
FastSPICE(UltraSim) ...
AMS Simulator ...
Design Variables
Name
Value
Analyses
Type
tran
Outputs
Name/Signal/Expr
Value
Plot
Save
Save Options
Plot after simulation: Auto
Plotting mode: Replace
Status: Ready
T=27 C
Simulator: ams(Spectre)Mode: batch
AMS Options@dapilx22
Main
Netlister
Messages
PLI
SDF
Timing
Miscellaneous
Use SDF command file
Delay type
None
Suppress SDF warnings
Allow unique delays for each source-delay path
Include detailed information in SDF log file
Round precision of timing in SDF file
VERILOG
Suppress SDF cmd file info msgs
Disable celltype validation
Allow SDF worst-case rounding
Do not run $sdf_annotate tasks automatically
VHDL
OK
Cancel
Defaults
Apply
Help
Your SDF file here
Min / Typ / Max

SUMMARY

Main Points to Remind

- **Hdl is not only coding: Important to use the right structures**
- **Changes in the digital interfaces adds a lot of work**
- **Test strategy has to be thought early**
- **A digital model of analog blocs may accelerate design time**
- **Version control of the design files may help you a lot**
  (svn / mercurial / git / gitlab…)
- **Think to switch to SystemVerilog, even without using classes**
  - Give you access to always_comb, always_ff and assert