

# Design Project 1: CMOS Image Sensor

TelcomBCN – UPC, DCISE QP09

## 0. Introduction

This exercise addresses the design of a mixed-signal system, including both analog and digital sections. During the first part of the project, the analog part will be developed using design techniques already introduced in previous Exercises. In the second part of the project, the design of the digital part using automatic synthesis will be demonstrated. Finally, the performance of the complete system will be analyzed using mixed-signal simulation.

The design project is based on the circuit explained in detail in the book [Dcise]<sup>1</sup>, Chap. 7.5. It is **necessary** that you read that section in detail before starting the design of the circuit.

The project Pre-Lab is divided in two parts. The first part addresses the analog section of the system, should be completed before 23rd April, and up-loaded to the Atenea web page. The second part addresses the design of the digital section of the system, should be completed before 10th May, and again uploaded to the Atenea web page. The Lab is done at the laboratory using Cadence tools for schematic capture and simulation. The results of the Lab should be written in a document delivered to the laboratory class professor. The lab section of this exercise should be completed by the end of the semester (1st June).

The evaluation of this final project amounts 2 points of the total qualification of the subject. Note that part of the evaluation of the project will be based on your answers to the Pre-Labs.

For the design of the analog section, technological data for AMS 0.35µm process is required. Such information is found in the Appendix of this document. For the design of the digital section, information of the AMS 0.35µm digital standard cells is required. The information on these cells is found in the datasheets provided by the manufacturer: [http://asic.austriamicrosystems.com/databooks/index\\_c35.html](http://asic.austriamicrosystems.com/databooks/index_c35.html)

## 4. Pre-Lab 2

In the first part of the project you designed the analog section of the image sensor, which had several control signals as inputs. These control signals were generated with simple voltage sources. In this second part of the project, you will design the digital section that allows to generate them from the inputs of the system, which are the following:

- Reset: Initializes the registers of the digital part, low-level active, asynchronous.
- CLK: Clock of the digital part, rising edge active, with period  $T=12.5\ \mu\text{s}$
- SelObt<1:0>: Selection of the shutter speed:

SelObt<1>	SelObt<0>	Shutter Speed	Shutter Time
0	0	1/8000	125 µs
0	1	1/4000	250 µs
1	0	1/2000	500 µs
1	1	1/1000	1000 µs

In order to reduce the complexity of the digital circuitry, in this second part of the project the image sensor will work with a simplified sequence of events, which is illustrated in the next figure:



Fig. 1 . Simplified sequence of actions during the reading of a 4-pixel sensor.

<sup>1</sup> [Dcise] *Diseño de Circuitos y Sistemas Integrados* A. Rubio, J. Altet, X. Aragonés, J.L. González, D. Mateo, F. Moll. Ed. Alfaomega, 2005.

Remember that the period of the CLK input is 12.5  $\mu$ s. The analog part of the sensor which was designed according to the specifications of the first part of the project will correctly work with the new, slower sequence.

15.- Draw the chronogram of the control inputs of the analog section according to this new sequence. Detail the duration of each pulse (in numbers of clock periods, not  $\mu$ s), including the shutter opening time for the different values of SelObt.

In order to generate the sequence of control signals, the digital circuitry is made of a Control Unit and a Process Unit, following the scheme in Fig. 2. The Process Unit consists of a programmable timer that, using SelObt and CLK as inputs, generates a pulse when the shutter opening time is finished. The Control Unit consists of a finite state machine, and will generate both the control inputs of the Process Unit, and the control signals of the analog part.

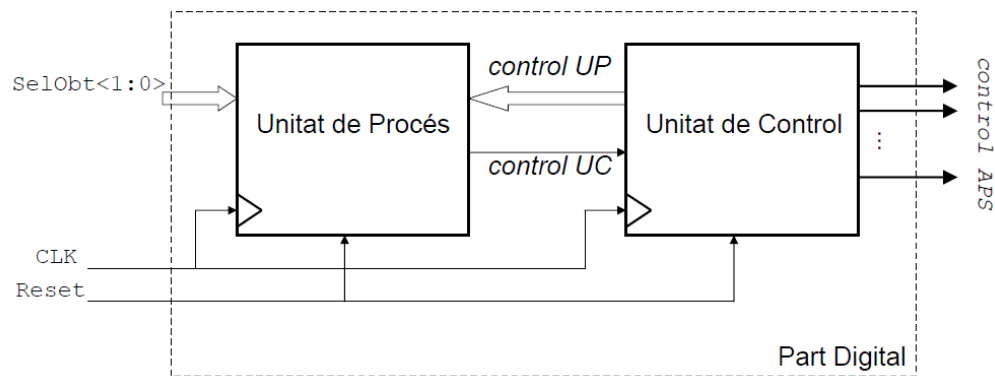


Fig. 2. Scheme of the digital section of the image sensor.

16.- Design the schematic of the Process Unit, using the following basic elements:

- Basic logic gates (AND, OR, NOT...)
- TFECL1 from the CORELIB library: toggle flip-flop with active low asynchronous reset.
- Comparator: combinational circuit that outputs '1' when both input buses have the same value, '0' otherwise.
- Decoder: combinational circuit that implements a look-up table with input SelObt<1:0> and outputs the shutter opening time, expressed in number of clock periods.

17.- Design a 7-bit comparator, which delivers a '1' output when both inputs have the same value, and '0' otherwise. Use simple gates like AND, OR, NAND, NOR, INV, XOR, NXOR as basic elements.

18.- Draw the states diagram of the finite state machine that, from the clock signal and the pulse output by the Process Unit, generates the control signals of the analog part and the Process Unit. Specify clearly the conditions of the transitions between the different states, and the states in which the outputs are active '1'.

## 5. Lab 2

### Synthesis of the digital section

In the Pre-Lab of this part of the project, you designed the schematics of the circuits in the digital section of the image sensor. The design was tedious and error-prone, and the next steps now would be to introduce the schematics in the Cadence environment using the schematic editor. This is again a tedious process, and obviously inefficient when designing large, digital circuitry.

In practical situations, the design flow of digital circuits is completely different, following a straightforward and efficient way. In a first step, circuits are described according to their *behavior* or *function*, using a Hardware Description Language (HDL). The following shows the description of a 7-bit comparator in VHDL<sup>2</sup>:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY comparador IS
  PORT
  (
    dataa      : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
    datab     : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
    AeB        : OUT STD_LOGIC
  );
END comparador;

ARCHITECTURE SYN OF comparador IS
BEGIN
  AeB <= 1 WHEN (dataa=datab) ELSE 0;
END SYN;
```

You can see that, beyond the syntactical burden, the description of the comparator is extremely simple, much simpler and faster to enter than the *structural* description that you designed in question 17 of the Pre-Lab2. Note that the circuit is exactly the same, the schematic that you designed and the VHDL code are just two different ways of describing or entering the same circuit.

The VHDL description of the circuit is then input to a CAD tool that automatically synthesizes the structure of the circuit, i.e. it generates the schematic for you. The structural description is generated in a Register Transfer Level (RTL), i.e. it only uses very basic elements like registers and basic logic gates. In the Cadence environment, the schematic of the circuit is synthesized in two steps. First, a tool known as RTL Compiler generates the structural description from a VHDL input file. The circuit synthesized is not a schematic, but a text description of the structure of the circuit in Verilog language. Next, you will import this structural description as a library cellview, and during this import process, the schematic and symbol views of the circuit will be automatically generated from the Verilog text description.

The digital section that you need consists of 4 circuits: a comparator, a decoder, a counter, and a control unit. We will complete the synthesis process for all these circuits, individually.

1.- Copy the `comparador.vhd` file into your working directory.

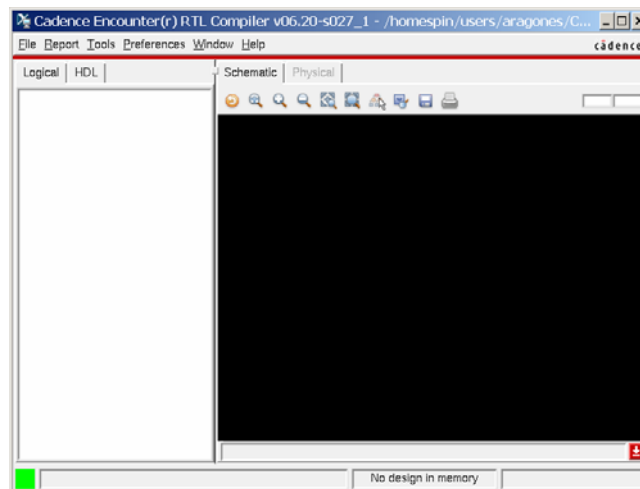
2.- In a terminal window, go to your working directory and launch the RTL Compiler by typing

- `rc -gui`

with no `&` sign in the end. You will get this screen:

---

<sup>2</sup> VHDL: VHSIC (very high speed integrated circuit) **H**ardware **D**escription **L**anguage is one of the two most used description languages for circuit synthesis. The other one is the language Verilog.



Note that terminal window has turned into the command console of the RTL compiler tool. First thing to do is to read the `comparador.vhd` input file. Type at the console prompt:

- `rc:/> read_hdl -vhd comparador.vhd`

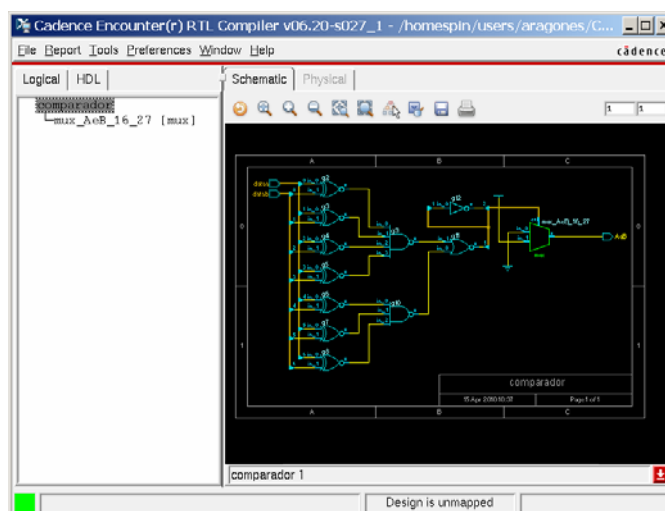
Next you need to specify the path and files of the library of the cells that will be used during the synthesis process<sup>3</sup>:

- `rc:/> set_attribute lib_search_path /usr/local/blend/Hit_kit16/ams035/ams_v3.70/liberty/c35_3.3V`
- `rc:/> set_attribute library {c35_CORELIB.lib}`

The next step is to elaborate a circuit structure making use of generic components:

- `rc:/> elaborate`

In the graphic window, you will see the schematic elaborated. In the **Logical** tab you can check the hierarchy of cells of the elaborated circuit, and in the **HDL** tab you can see the VHDL file that was read.

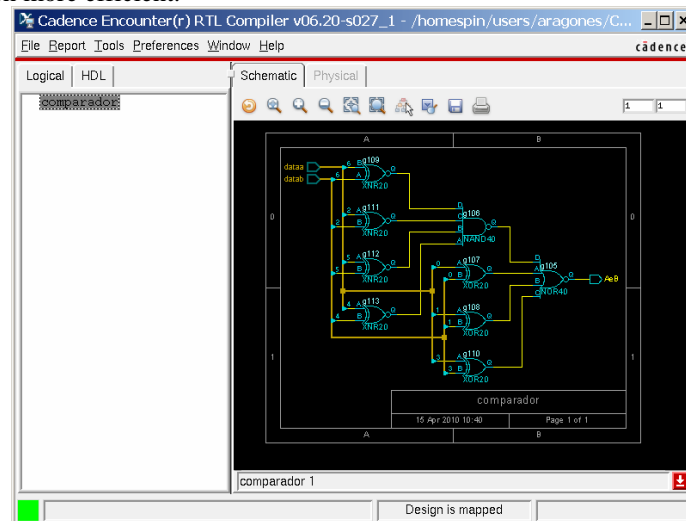


The schematic that has been elaborated is a pre-synthesis using generic components. The next step is now to map those generic components to the cells of the CORELIB library.

- `rc:/> synthesize -to_mapped`

<sup>3</sup> The library used is the CORELIB library that you know, but the synthesizer needs a specific file which identifies the functionality of each cell, and has area, timing, and power consumption information of the cells. This information can be used during the synthesis process to optimize any of these concepts.

You can see in the graphic window that the schematic has changed; now it makes use of the cells in the CORELIB library. Compare the schematic against your answer in question 17 of the Pre-Lab2. Probably the synthesizer tool has been more efficient.



The final step is now to write the result into an output file. The RTL Compiler tool is not able to generate a schematic cellview that can be read by the Cadence environment, but outputs the schematic description in a text file using the Verilog HDL language:

- `rc:/> write_hdl -mapped comparator > comparator_synth.v`

You can examine the `comparator_synth.v` file, you will distinguish a description of your schematic, where cells are connected according to their input and output assignments.

- `rc:/> exit`

3.- Examine the behavioral description of the rest of the digital circuits, contained in the `decoder.vhd`, `COMPTobt.vhd` and `unitat_control.vhd` files. Repeat the former procedure to synthesize the structural descriptions of these circuits.

### Importing the synthesized circuits as cellviews

Once the circuits have been synthesized, they must be imported into the Virtuoso Environment as schematic cellviews.

4.- Start the Virtuoso icfb environment as usual:

- `ams_cds -m fb -tech c35b3 &`

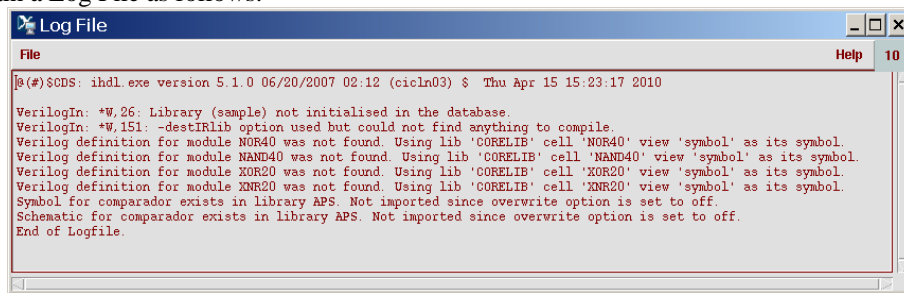
5.- From the icfb window:

- File → Import... → Verilog....

Fill the form as follows:

- **Target Library Name:** specify the library name of the project (you may use the Browse button to select it).
- **Reference libraries:** *sample basic CORELIB*
- **Verilog Files To Import:** select the `comparator_synth.v` file in the file browser of the upper part of the form; then click the Add button next to this field, the field should be filled with the complete path of the file (if the file is in your working directory, it is enough to write `comparator_synth.v` in the field)
- **Target Compile Library Name:** same as Target Library Name.
- **Import Structural Modules As:** select schematic and functional
- **Power Nets:** replace VDD! and GND! by *vdd!* and *gnd!*
- OK

You will obtain a Log File as follows.



You can verify in the Library Manager that a new cellview has been created named *comparador*, with views functional, schematic and symbol. Open the schematic and verify that it has the same structure as that synthesized by the RTL Compiler tool.

6.- Import the Verilog descriptions of the other digital circuits.

#### Creating the schematics of the digital section

7.- Create a new cellview called *Unitat\_Proces* and edit the schematic of the Process Unit using the symbols of the imported circuits. Create a symbol for this cell.

8.- Create a new cellview called *Seccio\_Digital* and edit the schematic of the digital section, including the Process Unit and the Control Unit. Identify all the nets in the schematic, including all the inputs and outputs of the Control Unit. Create a symbol for this cell.

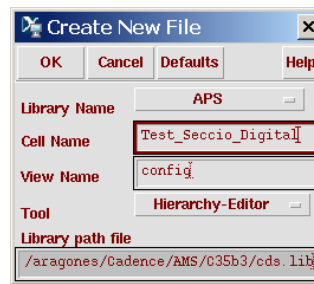
#### Simulation of the digital section

The design of the digital section has been completed and is now time to verify its correct functionality. We can follow a procedure analogous to that followed in former exercises and verify the time evolution of the different signals using a spectre simulation. Nevertheless, this procedure is quite inefficient because spectre is an analog simulator that obtains the voltages and currents in each node of the circuit, going down to transistor level. To verify the functionality of our digital circuit, we don't need such detail; a digital simulation is enough, i.e. a simulation that gives us the evolution of the logical levels ('0' or '1') at the inputs and outputs of the gates, without going down to transistor level. The simulation time with the digital simulator (called Verilog) will be orders of magnitude shorter than a simulation with spectre.

Digital simulations make use of specific stimulus files defined in Verilog language. Nevertheless, in our verification we will use voltage sources to define the inputs of the circuit. Since voltage sources are analog devices, we will not make a pure digital simulation, but a mixed-signal simulation, in which part of the circuit is simulated with spectre (the analog part) and part of the circuit is simulated with Verilog (the digital part). This mixed-signal strategy makes sense because at the end, we will need a mixed-signal simulation to verify the functionality of the complete image sensor.

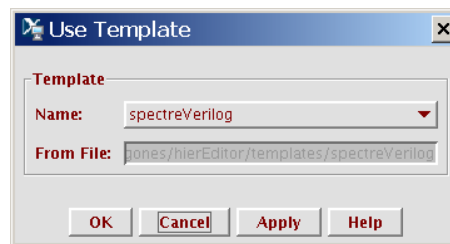
9.- Create a new cellview called *Test\_Seccio\_Digital* and edit the schematic of a testbench for the digital section. Make sure to include a DC voltage source connected to a vdd symbol from the *analogLib* library. Other voltage sources should generate the inputs of the digital section. Use a *vpwl* source to activate the reset signal during a few  $\mu$ s. Set the values of the *Se1Obt* sources such that they can be selected with simple 1 or 0 values from the ADE window. Identify all the nets in the schematic, including all the outputs of the Digital Section.

10.- Create a new cellview for the cell *Test\_Seccio\_Digital* using the tool Hierarchy-Editor:

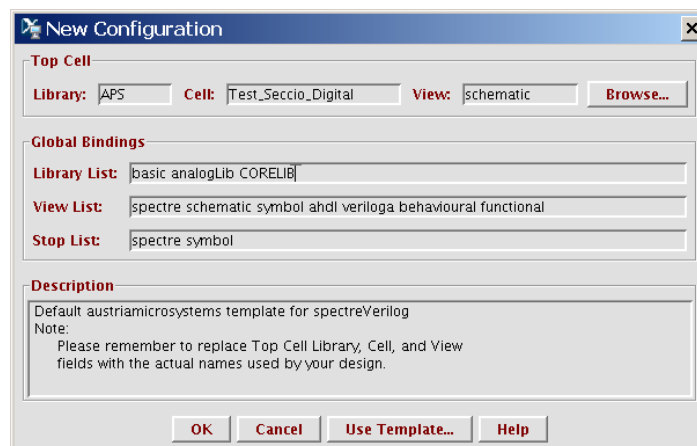


A couple of windows are opened.

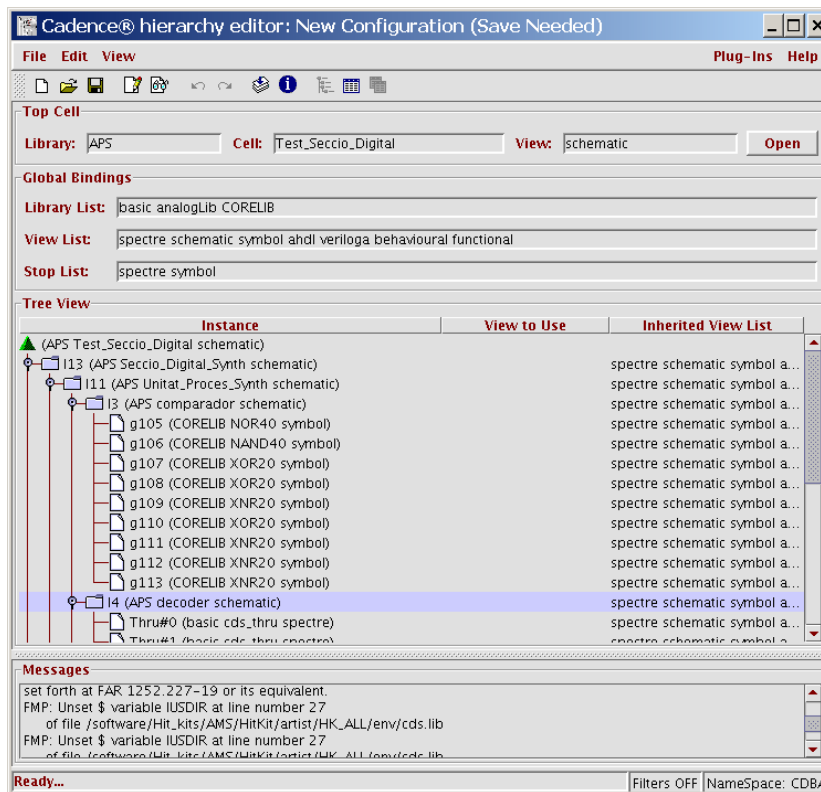
- Click **Use Template....** in the New Configuration window.
- Select **spectreVerilog**
- Apply
- OK



- In the New Configuration window, fill the Library List field with the following list: *basic analogLib CORELIB*
- OK



- In the main window of the Hierarchy Editor, click on the **View** menu and select **Tree**.
- Expand the hierarchy of the different cells until the finest level is shown



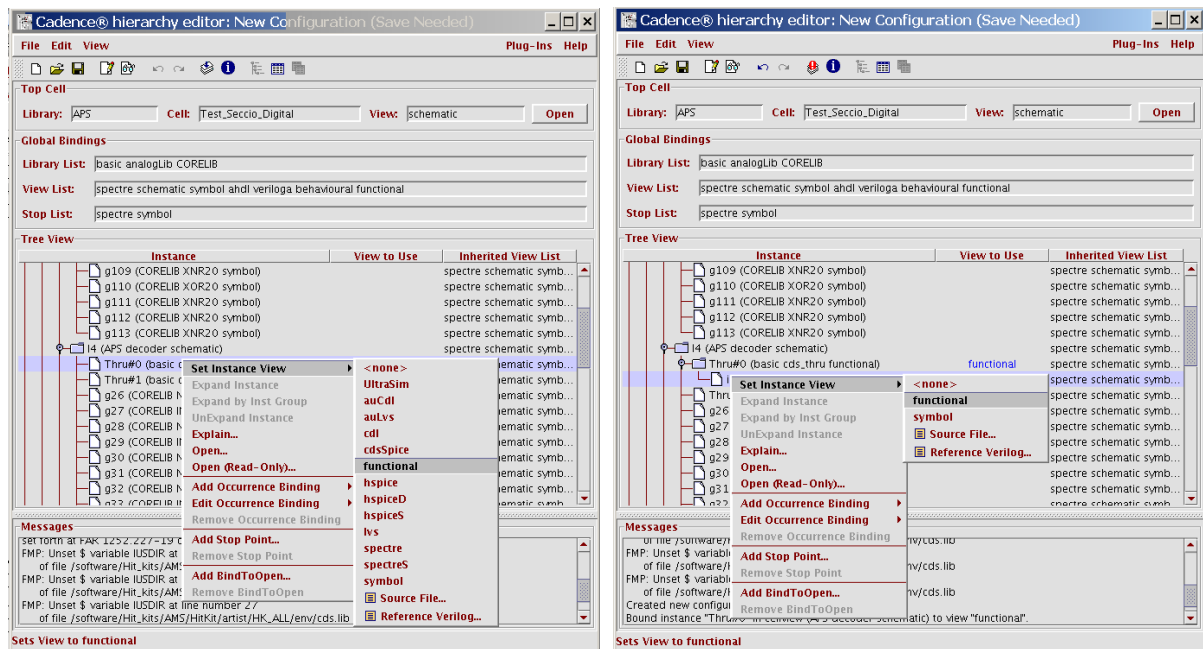
Observe the Tree View. It contains the hierarchical list of instances in your design which stops whenever a view suitable for simulation is found. Views suitable for simulation can be **spectre** or **symbol**. An instance with view **spectre** has a spectre model associated to it and will be simulated in the analog domain with the Spectre simulator. An instance with view **symbol** has a verilog model associated to it and will be simulated in the digital domain with the Verilog simulator.

11.- If you observe the list of instances in the Hierarchical Tree, you will notice that there are some **Thru#** instances inside the schematics of the digital circuits, which are **cds\_thru** cells with a **spectre** view. The synthesizer has added these *through* cells to connect two nets with different names (i.e., the **cds\_thru** cell is just a "utility", it has not any circuit associated to it). These cells will provoke some inconvenience when observing the waveforms, since their output will be considered an analog node when in fact it must be a digital node. To avoid this inconvenience, we must specify that these cells will not be simulated with their **spectre** view, but with a functional view.

In the Hierarchy Tree, identify the schematics where there are **Thru#** instances, and the number of them.

- Select the first of the **Thru#** instances and click the right button to expand the menu
- Set Instance View.... functional (left figure below)
- You will see that the **Thru#** instance has changed its view to functional. The instance can now be expanded into a lower level, which contains a **cds\_thrualias** cell with a symbol view.
- Select the instance, click the right button to expand the menu, and select the functional view also for this **cds\_thrualias** cell (right figure below).





Repeat these steps for all the instances of the cds\_thru cells. When finished, make sure that no Thru# instances are now listed in the Hierarchical Tree.

- File → Save

12.- In the Library Manager, double click the config view of the *Test\_Seccio\_Digital* cell to open it.



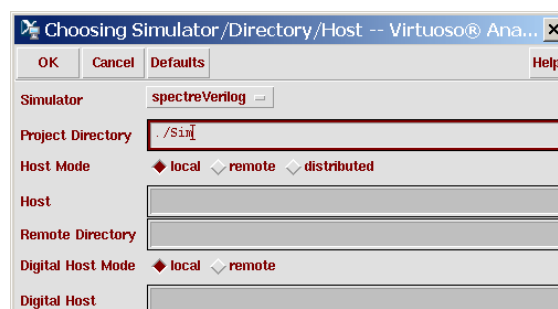
The schematic of the cell opens, but note that the window banner specifies that this is the Config view. This is necessary for mixed-signal simulation: you cannot make a mixed-signal simulation from an schematic view.

We will now start the simulation environment:

- Tools → Analog Environment

The familiar Analog Environment window opens.

- Setup → Simulator/Directory/Host
- Select spectreVerilog
- OK



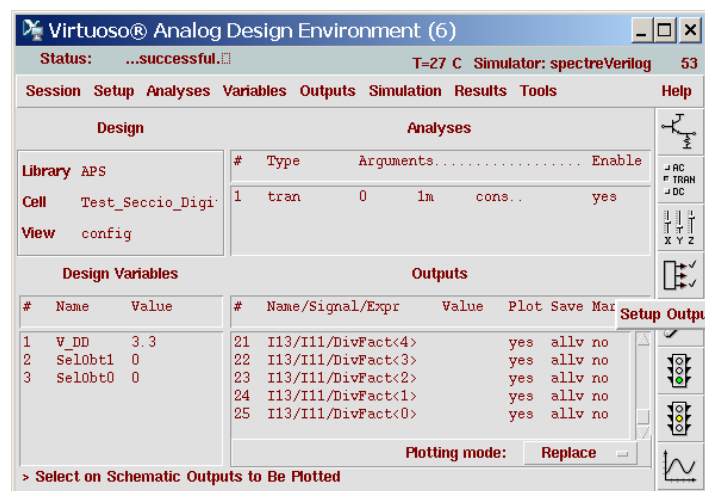
This option selects the list of views that will be used for this particular simulation, i.e. the simulation environment already knows whether a block is digital or analog. You can view this partitioning by selecting on the schematic (config view) window:

- Mixed-Signal → Display Partition → All Active

The nets and instances of the schematic are now highlighted in different color according to their consideration for simulation. Instances highlighted in green color are analog; elements highlighted in blue color are digital, and nets highlighted in orange color are mixed (ex., outputs of analog instances and inputs of digital ones). Go down the hierarchy and verify that the partitions are as expected.

Go back to the Analog Environment window and complete the rest of the fields:

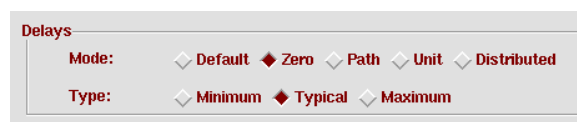
- Variables → Copy from Cellview. Specify appropriate values to the variables in your design.
- Analysis → Choose.... Specify a transient analysis with a suitable simulation time.
- Outputs → To Be Plotted... → Select on Schematic. Select all the inputs and outputs of the *Seccio\_Digital* cell, the output of the Process Unit. You can also select the input buses of the comparator, if you want.



Although the steps of the synthesis and mixed-signal definition that we have performed so far may seem complex to you, in fact we have followed a simplified flow. For example, one of the steps that we have ignored, and which is essential to any real verification, is the generation of the timing specification of each cell. The delays of each digital cell in our design should have been written to a file and the simulator would read this file to include the delays between signals, accounting for the fan-out of each net.

Since we have skipped these steps during the synthesis process, we must specify the simulator that we will ignore the delays of the digital cells:

- Simulation → Options → Digital.... Select Zero as Delay Mode.




Save the state. Note that, next time you open the ADE window, you will need to select the spectreVerilog simulator before loading a previous state.

You are now ready to create the Netlist and Run the simulation.

### Verification of the functionality of the digital section

13.- After the simulation is completed, the WaveScan window opens. Note that the waveforms in the upper part of the window are digital, while the waveforms in the lower part are analog. You can move the cursor along the waveforms, you will see that the readings of the analog nets are voltages, while the readings of the digital waveforms are logic values (St0 or St1).

For your convenience, you change the background color by selecting **Frame → Color Schemes → White**. Also you may on any digital waveform and increase the number of **Digital Visible Rows**.

Also, you may want to group the bits of your buses and provide a decimal reading. Make sure that the bits are properly ordered in the WaveScan window. First select the label of the MSB in the bus. Then select the label of the LSB while holding the Shift key in your keyboard. Next, select the **Create Bus** button . The **Create Bus** window opens. Make sure that the order of the Bit and Signal list matches. Write a bus name. Select **Signed Decimal** as Radix and **Replace**. When clicking **OK**, the bits of the bus are replaced by a signal waveform in which the values are expressed in decimal radix.

Verify and report the correct functionality of the digital section, particularly the sequence of the control signals of the analog section. Repeat the verification for the different shutter speed times.

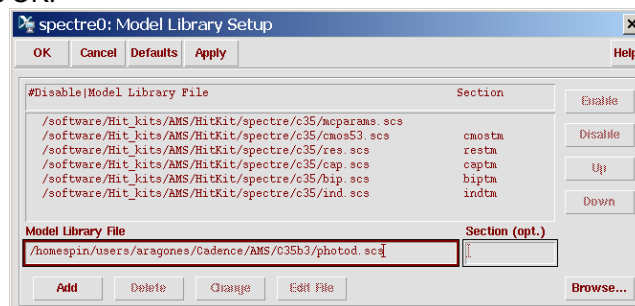
14.- It is now time to verify the correct functionality of the complete system. Edit a schematic called *Test\_All* that contains the digital section, the analog section, and the stimulus that you already used to test the digital part.

Repeat steps 10 and 11 to create a config view of this cell and replace the views of the *Thru#* cells.

Open the **config** view of the *Test\_All* cell, and repeat the step 12 to define a mixed-signal simulation of the complete system. Make sure to define the illumination values and plot the internal nodes of interest of the analog section. It is a good idea to make a simulation with exactly the same conditions of a simulation of only the analog section that you did in the first part of the project, in order to compare the results.

Remember also that you must add the model of the list of model files that are used for the simulation:

- **Setup → Model Libraries... Browse...**
- select the *photod.scs* file
- click on **Add** and **OK**.



It is also remarkable that now, as you observe **Display the Partition of the design**, the outputs of the digital circuitry are now considered mixed-signal. This is because now they are also inputs of the analog circuitry. The consequence of this is that, in the WaveScan window, they will be represented as analog waveforms instead of digital.

15.- Run the simulation and verify the correct functionality of the complete system, for the same illumination levels and shutter opening times that you simulated in the first part of the design. Compare the results obtained.

Optional improvements:

- Improve the linearity by canceling the body effect in M3 (force Bulk=Source).
- Optimize the dynamic range (try lowering  $V_{PN}$ ).
- Optimize the acquisition sequence (only analog part of the sensor)
- Introduce controllable ISO sensitivity (only analog part of the sensor)
- Increase the number of pixels (only analog part of the sensor)
- Report the power consumption of the analog and digital parts of the sensor.
- Report the power consumption of the digital part if  $V_{dd}$  is reduced to half its original value.
- Which modifications should be necessary in the analog part of the sensor if  $V_{dd}$  was reduced to half its original value?
- Report the power consumption of the complete circuitry if  $V_{dd}$  is reduced to half its original value.
- Include the A/D conversion and show its correct functionality.