Lund University

Department of Electroscience

Analog IC Projects
Laboratory Manual

Stefan Molund

October 2002

Contents

Abstract

The tools described and used in this manual; *Hierarchy Editor, Spectre, Verilog,* and *Silicon Ensemble* are trademarks of **Cadence Design Systems, Inc**.

The Design Kit used is from **Austria Mikro Systeme International AG**.

Based on previous work by **Pietro Andreani, Rifat Zejnic,** and **Bengt-Arne Molin** at the department of **Applied Electronics** at **Lund University**.

Stefan Molund

Introduction

This is the laboratory manual for the course Analog IC projects. The goal of these laboratories is to convey some knowledge on how to design and verify larger projects.

It is assumed that some knowlede on how to use Cadence still remains from the previous course of Analog IC design. Further information can be found in the departmet produced manual *Cadence Condensed*[2], and in the on-line set.

There are two laboratories which will cover the following topics.

1. Functional verification by using analog and digital simulators on various parts of the design, to shorten time spent in the simulator, called **mixed-mode** simulation.
2. **Floorplanning** and **Routing** of an entire chip using the tool *Silicon Ensemble*.

The environment is set up and the tools are started by the same procedure used in the basic course, probably **inittde ana2002**.

Laboratory Parts

A laboratory is divided into three parts:

- Preparation and homework exercises.
- Execution of the laboratory tasks.
- Submission of a complete laboratory report.

A laboratory is not approved until all parts are satisfactorily fulfilled.

Laboratory Report

In addition to the solution to the given homework exercises, the report should also contain an evaluation of the laboratory, what were the major obstacles in the laboratory, and suggestions of how to improve the different laboratory tasks.

Fell free to submit criticism regarding the manuals. It should not affect the evaluation of the laboratory reports, much.

Laboratory A : Verification of an AD-converter

This laboratory starts by introducing current mode technology that is used to construct a one-bit AD-converter. This is later extended up to a full six-bit converter along with a digital control block. This design is then simulated by using the mixed mode simulator environment. In this both the analog and digital simulators are active in order to achieve a high performance of the simulation.

Current Mode Technology

The current mode technology (in which the signal is a current) offers a number of advantages over the usual voltage mode. As a rule, the current mode circuits have no demand on high gain amplifiers which minimizes the need for high performance operational amplifiers. Neither is there any need for high precision resistors or capacitances. The consequences of this is that current based circuits are designed almost by transistors which makes them compatible with most digital processes.

Algorithmic AD Conversion

The advantage of the algorithmic AD-conversion principle is that it can be realized by relatively simple circuits [1]. An example of an algorithmic conversion is shown in figure 1.

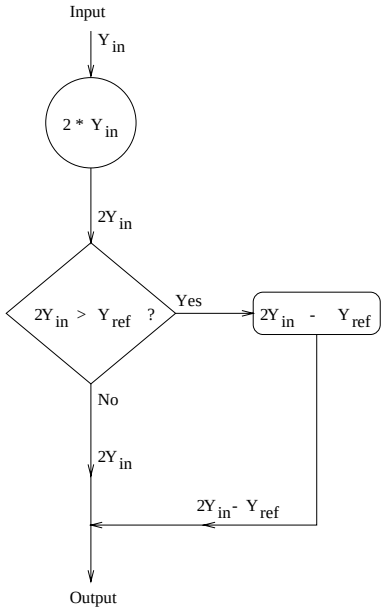

Figure 1: Flowchart for an algorithmic AD-conversion.

The input current, Y_{in}, which can assume values from zero up to a reference value, is doubbled to create $2Y_{in}$. If this new current, $2Y_{in}$, is less than the reference, Y_{ref}, the digital output is set to 0 and $2Y_{in}$ is the new Y_{in}. Otherwise, if $2Y_{in}$ is greater than Y_{ref} the output goes high and the new Y_{in} will be the difference between $2Y_{in}$ and Y_{ref}. The resulting Y_{in} is then connected to an identical cell, which will perform the same operations and generate a new digital bit. This can then be repeated until the needed resolution is achieved.

Basic Current Mirror Based AD-converter

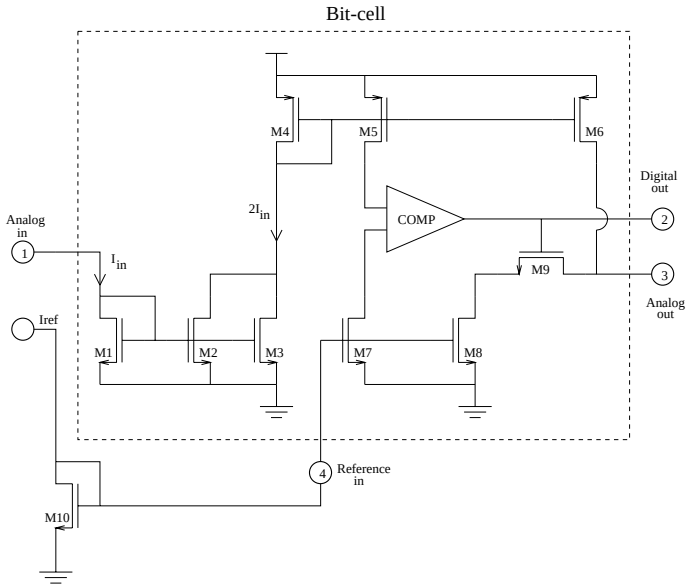

Figure 2: A bit-cell implementing a one-bit algorithmic conversion.

The circuit in figure 2 is used to implement an algorithmic AD-converter based on common current mirrors. The design performs a one bit AD-conversion in the following way: The input, I_{in}, is multiplied by 2 by a current mirror consisting of M1, M2, and M3. The resulting $2I_{in}$ is then mirrored from M4 to the comparator, COMP, by M5, and by M6 to the output. COMP is used to compare $2I_{in}$, from M5, with the reference current, mirrored in by M7. If $2I_{in}$ is less than I_{ref} the digital output will go low, thus switching off M9, and the output current will be $2I_{in}$, from M6. If, on the other hand, $2I_{in}$ is greater than I_{ref}, the digital output will be set high, which opens M9 and the current I_{ref} will be subtracted from the current from M6. The output current will then be $2I_{in}$–I_{ref}. This completes the one bit algorithmic conversion.

An N-bit AD-converter can be designed by connecting N one-bit converters in a cascade so that the analog output from one cell connects to the input of the next as

shown in figure 3. The transistor M10 is used to mirror the reference current into all the cells. The circuit does not need any controlling signals and the configuration will result in a compact design, easily modified for different resolutions. The maximum resolution is limited by the accuracy of the current mirrors.

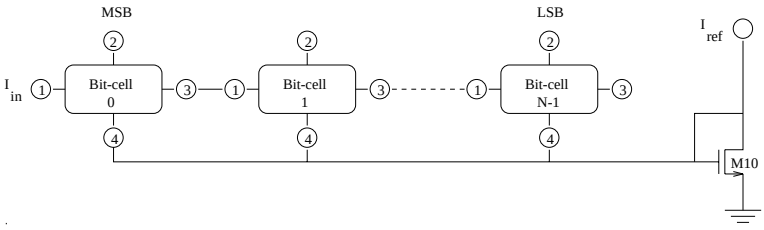

Figure 3: Cascading one-bit cells forming an N-bit converter.

Figure 4 shows the structure of the current comparator. It consists of two inverters in series. The output will switch when the resulting current I_{in}–I_{ref} changes direction. If the transistors are made small the comparator will be very fast.

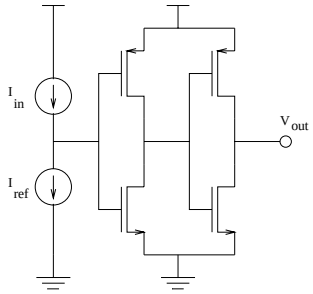

Figure 4: Current comparator composed by two inverters.

Laboratory

This laboratory starts by simulating a bit-cell, called an AD-block. Then a six-bit converter is to be designed by using six AD-blocks in series. Finally, a converter is created by using one modified version of the AD-block together with a digital control block.

By partitioning the AD-block (figure 5) into smaller cells a better overview is accomplished and further work made easier. The natural sub-cells are *ndiode* (M1), *powx2* (M2, M3, and M4), *pmirr* (M5 and M6), *anapart* (M7, M8, and M9), and finally *comp*. The transistor designations are from figure 5.

When the AD-block is to be simulated, some extra components has to be added. The voltage for the reference input (node 4) is created by forming a current mirror by adding a transistor (M10) which are fed with the reference current. The output

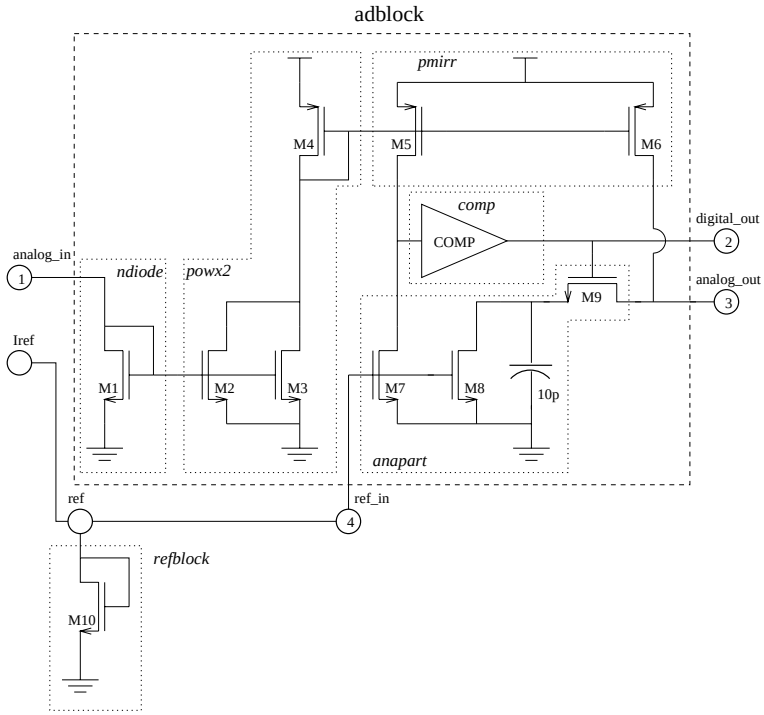

Figure 5: The AD-block split into sub-cells.

current (node 3) will be fed into another transistor of suitable size. This means that the load on the output will be the same as if another identical stage was connected on the output, compare with M1. The digital output can be left unconnected but it is recommended that it is loaded down by a resistor.

Laboratory Exercises

Part 1 : AD-block

1. Copy the design data according to the following procedure. Scrutinize the hierarchical structure and make sure that the function of each block is understood.

   - Create a library link *labbA* pointing to the library *$AMS_DIR/icp2002/labbA* by using the path editor (*CIW: Tools > Library Path Editor*).
   - Copy the entire library *labbA* to a local library (labA, for instance) which will be created. Check the box **Update Instances**. Attach the library to the technology **TECH_CSI**.

- **Do not** delete the link yet, it will cause problems. Wait until the next time Cadence is started.

2. Create the schematic for the current sink in the input, *refblock* in fig. 5. It should consist of one diode connected nmos-transistor (W/L=4/4) with a contact (*Pin*) connected to drain and gate. Name the contact *ref*. This forms a current mirror together with M7 and M8 for the reference current.

3. Generate the symbol view of the *adblock* from the schematic.

4. Create the test-bench and include the adblock and reference. Also add the sources **vdc** (3.3V) and **idc** (64uA). The signal source **ipulse** should generate a 6us long current pulse with an amplitude of 60uA. Set the rise and fall time to 1us.

   Mind the direction of the currents!

   The output current should be sinked by a structure identical to the one on the input.

Start the simulation environment from the test-bench and analyze the block with a transient analysis for 6us. Remember that currents that is to be saved must be selected before simulation starts. What happens on the outputs and why?

Part 2 : Six bit AD-converter

Now a six-bit converter is to be constructed by using the one-bit six times as in figure 3. The reference current is mirrored into all of the one-bit converters by one *refblock*. The digital outputs will form a bus with the most significant bit coming from the first of the converters.

1. Create a new schematic for the converter. Include six instances of the one-bit converter and connect them in cascade. Name the contacts on the digital outputs *D<5>*, ..., *D<0>*. The input current, *analog_in*, enters the most significant block and the output, *analog_out*, comes from the least significant.

2. Generate a symbol view of the converter.

3. Create a suitable test-bench. Include what are needed, dont't forget the current loads. Use the **ipwl** unit as current source. This generates a linear ramp between pairs of coordinates. Set **Number of pairs of points** to two and enter the coordinates (0, 0) and (33u, 33u) to form a ramp from 0 to 33uA with a width of 33us.

Perform a transient simulation over the ramp and study the outputs. How well does the digital outputs represent the analog input?

Part 3 : Clocked six-bit AD-converter

Instead of using six blocks in cascade it is possible to direct the output back to the input of the same block as in figure 6.

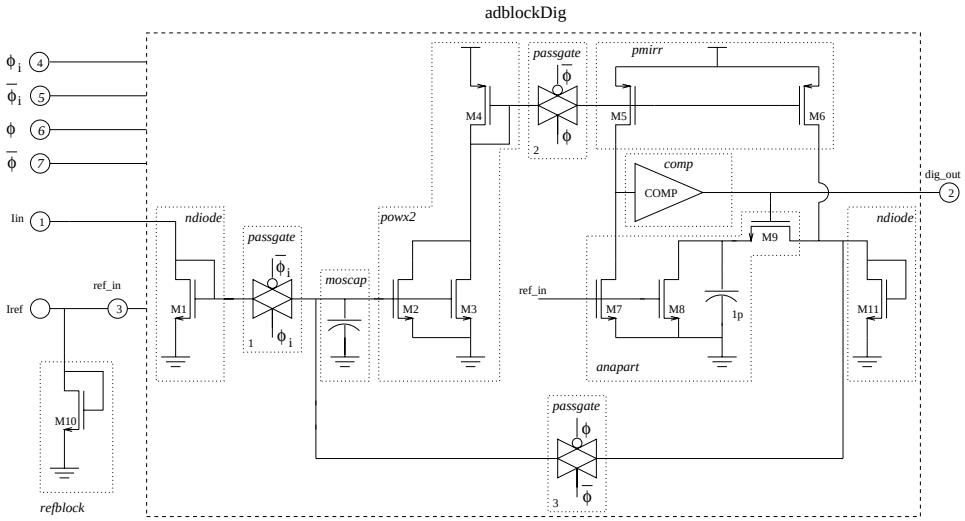

Figure 6: Digitally controlled AD-block.

It is essential that the current amplification (done by the cell *powx2*) and the current subtraction by the blocks *pmirr, comp,* and *anapart* does not interact with each other. This is solved by splitting the AD-block into two parts, isolated from each other by a pair of transmission gates ($passgate_2$ and $passgate_3$ that are clocked alternately to each other.

The values of the input- and output currents are saved as voltage levels on the capacitances on the gates of the transistors M5 and M6, on the second stage, and M2, M3, and the extra block (*moscap*) on the input. When the conversion is done a new sample of the input current is read in through the $passgate_1$ block. Note that this is controlled by a separate clock, ϕ_i.

The control block *control* in figure 7, which is composed entirely by digital standard cells, generates the clocks needed to control the AD-block, ϕ, $\overline{\phi}$, ϕ_i, and $\overline{\phi_i}$. It also saves the results, as the bits are calculated, and stores them in a shift register. When then conversion is done and a new sample is loaded the result is fed to an output bus, *res<5:0>*.

The behaviour of a design that consists of both analog and digital parts can be simulated by a *mixed-mode* simulator, i.e. one that runs the digital parts in a digital simulator (*verilog*) and the analog ones in a analog simulator (*spectre*). This means that a time-consuming simulation of the digital blocks can be avoided since the digital simulator is much faster. Some of the time saved is of course spent by the

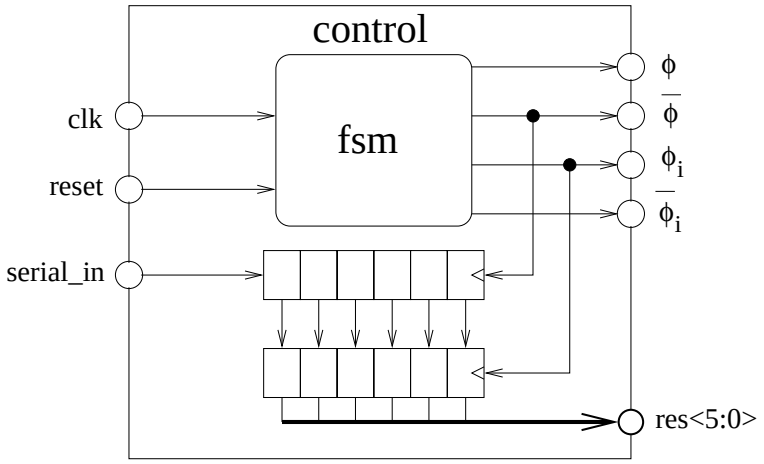

Figure 7: The control block.

interface between the simulators.

1. The building blocks *adblockDig and* control has already been copied to the local library (**labA**).
2. Make sure that the structure is understood and change the value of the capacitance in *anapart* to 1 pF.
3. Create a testbench and instanciate the control- and AD-blocks into it.
4. Connect the components to each other. Do not forget the input current sink.
5. Include signal sources as for the six-bit converter. The clock (**vpulse**) should have a frequency of 2MHz whith flanks of 1ns. The reset segnal, active low, should be on the first 400ns.

   Ramp the input current as before.

The netlister (the program that partitions the design) must be told which part of the construction that should be considered analog and which are digital.

1. Create a configuration view according to the procedure described in the layout chapter of the cadence manual [2], the library and cell names must be the same as for the testbench but the view is set to config.
2. Select **Verilog_Spectre** as template.
3. Make sure that you have the schematic connected to the config view active before starting the simulation environment.

In the **Hierarchy Editor** it is now possible to select (partition) what simulator is to be used for the selected block. As default, for the analog parts the normal simulator spectre will be used, of course. The digital standard cells will be simulated in the digital simulator which is shown by the view selected, *symbol*. If there is need for a more thorough simulation of the digital parts this can be achieved by changing the view to *cmos_sch*. Now the selected instance can be opened up down to transistor level. For this laboratory, however, it is sufficient with the default behaviour.

At last the simulation can be performed.

1. Start the simulator (*Tools > Analog Environment*).
2. Make sure that the **spectreVerilog** simulator is the selected one (*Setup > Simulator...*).
3. Check that **Preserve Buses** is on and **Netlist Explicitly**is off in the form activated by *Setup > Environment...* **Verilog Netlist Option**.
4. Select signals that should be saved. It is essential to select at least one analog and one digital. Currents must be selected ahead of simulation.
5. Perform a 30us transient analysis and contemplate the results.

Laboratory Report

Compile a set of the analyses printouts along with some description of what has been done together with the answer to the questions in the laboratory manual.

Homework

1. Make sure you understand the theory of the AD-converter.
2. Go through the entire laboratory manual.
3. Repeat the parts from the Cadence manual [2] that are used in this laboratory.

Laboratory B : Floorplanning, Place & Route.

This is more of a tutorial, to prepare for the tasks needed to complete the project, than a laboratory so no report will be required.

In this laboratory the tool Silicon Ensemble will be used to place and connect standard cells, homemade blocks, and pads on a design. The current mode AD converter from the previous laboratory will be used.

Pads and standard cells are usually supplied by the vendor who will fabricate the design, and are therefore prepared in the correct fashion to interact with Silicon Ensemble (SE). In this course in analog design you will need more than the digital cells, therefore you will have to prepare your own structures and schematics to be used in SE.

Silicon Ensemble is a stand alone tool so you will need some intermediate files, that describes the various aspects of the design, in an appropriate format. These files are called LEF (library exchange format) which describes the rules on how wires can be drawn by the router and what the available cells and blocks look like. The other file is called DEF (design exchange format) and contains the same information as the schematic on how things should be connected. After routing it will also have the coordinates of where the blocks are placed along with all wire information.

The Laboratory Design

Set up a link to the design library *$AMS_DIR/icp2002/labbB* with the path editor, as in the previous laboratory. Copy the entire library to a local one (*LabB*). Do not forget to select **Update Instances**.

As shown in figure 8 the highest level in the schematic consists of homemade blocks plus some pads to connect to the outside of the chip. The outside world is a harsh environment so the pads usually contains protective circuits and big buffers. Because of that the pads tend to be large structures and are often what decides the size of the finished chip.

For the current inputs (**Iin** and **Iref**) the analog pads, with direct connection from pad area and input, are used. For the other inputs, digital buffered pads. Note also the powerpads and how to connect the vdd- and gnd symbols to them. It is also a good idea to give names to the pads instead of letting the system choose one. It will be easier when placing the pads.

In the layout view of the cell adblockDig layout all the blocks has been incorporated and manually connected together. This is the recommended procedure since the tool (SE) are not too good at handling a large number of non-standard blocks. The connections (pins) are created as for a single cell in the **pn** purpose layer.

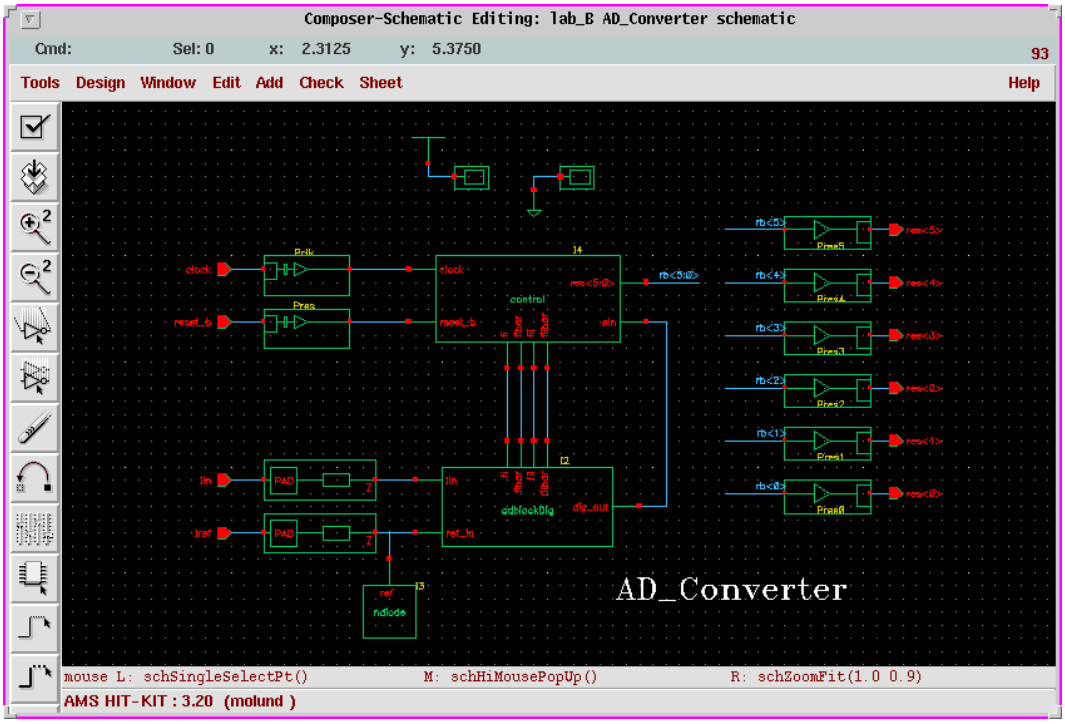

Figure 8: The AD_Converter schematic.

Silicon Ensemble does not need to know the details of what a cell contains. In a large design with thousands of cells the amount of data would be too big to handle. What it needs is some information of the outline of the block and where the connections are. All this dwells in the **abstract** view of the cell. Here, as shown in the adblockDig abstract view, most of the internal area is covered by the **METx by** layers. This is a blocking layer that tells the router that it can not use that layer over the covered parts of the block.

Abstract and LEF file Generation

This is the recommended prodedure for generating the **abstract** view from a **layout**. The cell **ndiode** will be used as an example.

- Open the layout view of ndiode. In the layout the connection points are drawn in the **pn** purpose layer as described in [2].
- Add a rectangle in the layer **prBndry dg** that just about encloses the whole cell. This will be the outline of the cell as Silicon Ensemble sees it.
- Make sure that the origin of the cell is the lover left corner of the structure. If not use the command *Edit* > *Other* > *Move Origin* to move it.

- Some properties needs to be added to the design. They are **prCellType : macro** and **placementClass : blockSite**. The easiest way to to this is by the short command *Ctrl+Shift+b*. Check the result by using *Q*.

- Start **AutoAbgen**, which generates the abstract view, from the main window (icfb) with the command *AutoAbgen > Auto Abstract Generation* and click on the **Setup** button. Click on **Load** to read in the setup file and enter the name of the library (labB) where it says *CHANGE_THIS*. Press the **Apply/Save** and click yes in the two following small boxes followed by an **OK** in the Setup form.

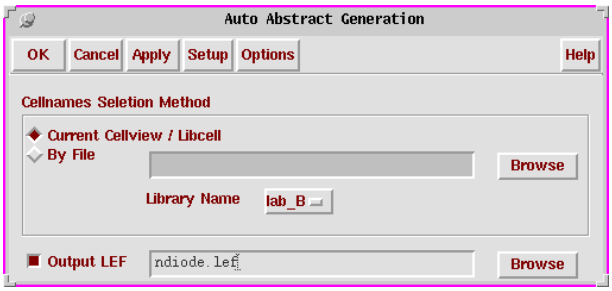

Figure 9: The Auto Abstract Generator Form.

- Fill in the form as in figure 9. Make sure that the correct library name is shown and that the layout view is visible before pressing **OK**.

- **OK** any window labeled *Refresh Data From Disk* that pops up during the run.

If there are no problems (look in the log-window) the abstract view will be created, have a look at it and make sure that there are no flashing error markers. You will also get a lef-file of the ndiode. The lef-file has the same information as the abstract view but in a form suitable for Silicon Ensemble. The abstract will be used later in Cadence.

Unfortunately the generator puts some rubbish in the file that has to be removed. This is done by the command *$AMS_DIR/editlef ndiode.lef*. Have a look at the file afterwards and try to understand its contents.

Transferring the Schematic to Silicon Ensemble

Before place- and routing with Silicon Ensemble may commence, the information from the schematic view, i.e. how everything is connected together, has to be transfered to a format that SE understands. In the **control** block (q.v.) some flip-flops have bussed names, **SR<5:0>** for example. This means that there are actually six of them, with the names **SR<5>, SR<4>, ... SR<0>**. Constructs like this has to be resolved so that every building block of the design has a unique name. The process of doing this is usually called netlisting. A similar process is used when the design is simulated.

- The process is started from the **icfb** window by applying *File* > *Export* > *PRflatten*. Fill in Library, Cell, and View (schematic) in the form that pops up and click on OK. This command builds a view called **autoLayout** which contains all necessary cells and information on how they are connected.
- Next, use *File* > *Export* > *DEF* and fill out the form as in figure 10. Note that it is from the **autoLayout** view the output def file is created.

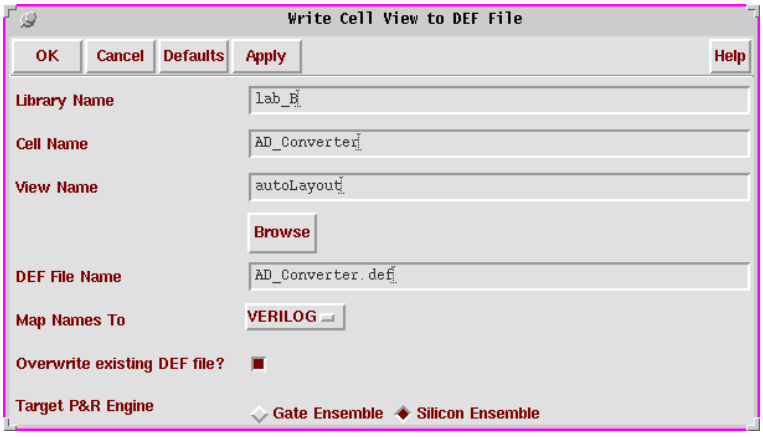

Figure 10: DEF file Generation.

- The def file also contains some errors but they are easily remedied with *$AMS_DIR/defpul AD_Converter.def*. A **_mod** will be inserted in the file name before the extension.

Now, everything that is needed for the Place- and Route stage are finished so Cadence dfII can be closed down.

Silicon Ensemble

As stated earlier Silicon Ensemble is a tool for Place and Route. It does not need or want the full layout of the cells since this would increase the amount of data to be handeled. This is done by using a set of **abstract** views, that contains information on size and how to connect and route, of the cells. For the standard cells and pads the manufacturer supplies them in a **LEF** (Library Exchange Format) file. For the design itself a **DEF** (Design Exchange Format) file is used.

Apart from the obvious steps of defining the environment and starting the program, the following steps are needed for a design cycle. Naturally, this is an iterative process which means that sometimes the designer has to back up and redo some steps since some selections may turn out not so good in one of the following stages.

- **Floorplanning:** Decisions on the size, square or rectangular, where cells and blocks can be placed.
- **Placing Pads and Blocks:** Creating placement file for pads, and manual placement of the blocks.
- **Powerplanning:** Where the main power lines should be located and their size and layer.
- **Placing Cells:** Standard cell placement.
- **Routing:** Connection of power and signals.
- **Generating Output Data:** The final design can be saved in some different formats depending of its readiness. Maybe it needs further editing in some other tool or it cat be sent directly for fabrication.

Setting Up the Environment

All search paths and variables needed to run Silicon Ensemble are already set up by the initialization command for this course. What needs to be done are to define a specific directory for the run and to download some definition- and startup files.

The environment can be initialized with **inittde ana2002 nostart** without invoking Cadence dfII.

- Create a library (**labB_SE**) and descend into it.
- Build the necessary files by running the script **$AMS_DIR/build_se_dir**. A couple of command files and new directories are now built, the most important of them is the **dbs** which is where the database will be written when you

save the design. This can be very big and you might consider linking it to some temporary area.

- Start SE with the command **sedsm -m=150 &**. The command means that that the program should not allocate more than 150 MB of memory from the system. This is enough for the laboratory but if you need more, valid selections can be found by the option **-help**.

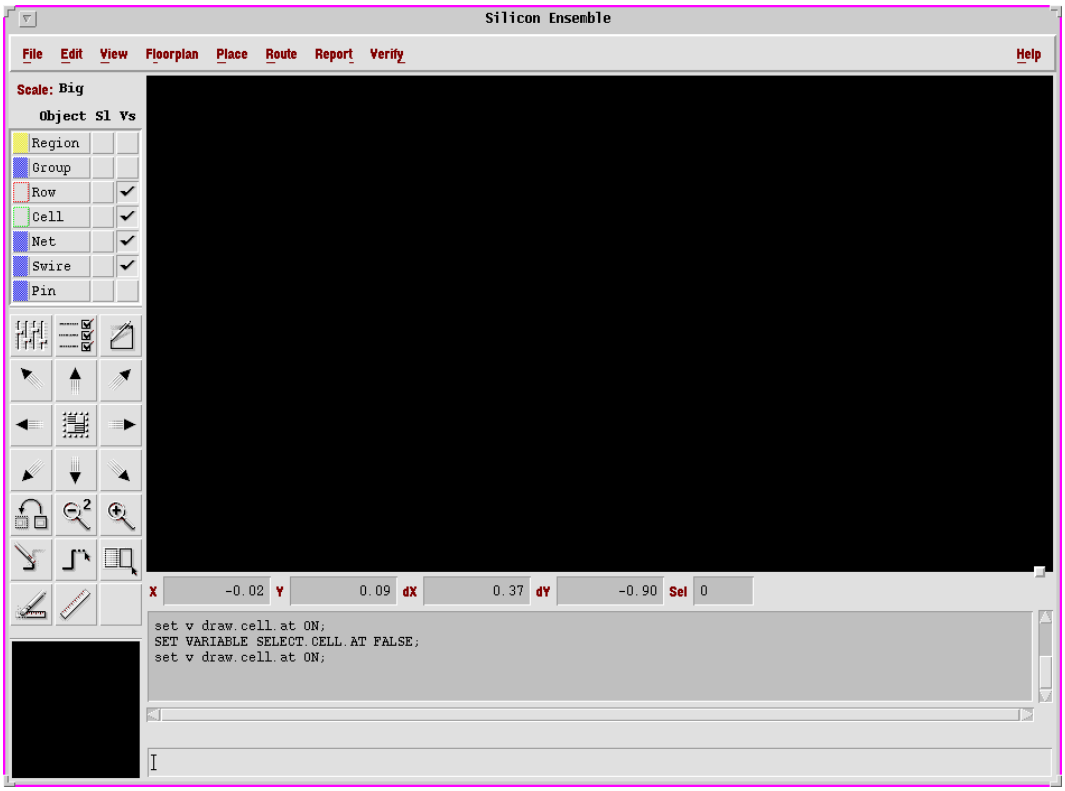

Figure 11: The Graphical Interface Window.

Now the Silicon Ensemble Graphical Interface window should be visible. As shown in figure 11 it looks like the other tool windows. There is a **menu bar** at the top, some **icons** for frequently used commands, a **message area**, a **command line** at the bottom where text commands can be written. The big **artwork window** is empty since nothing have been read into SE yet. In the top left area is the **Object Selection Control**, where the selectability and visibility of the different kind of objects can be enabled or disabled.

Input the Design

Start by calling on the supplied command file. Select *File* > *Execute* and doubleclick on **analefin.mac**. This file (q.v.) imports the library lef files and the specific def file for the construction. It must be modified for the project but the format should be obvious. Check the message window for errors and if none, proceed!

The above command could have been written on the command line, and it would have looked like *exec analefin.mac;*. Note that all written commands must end with a semi.

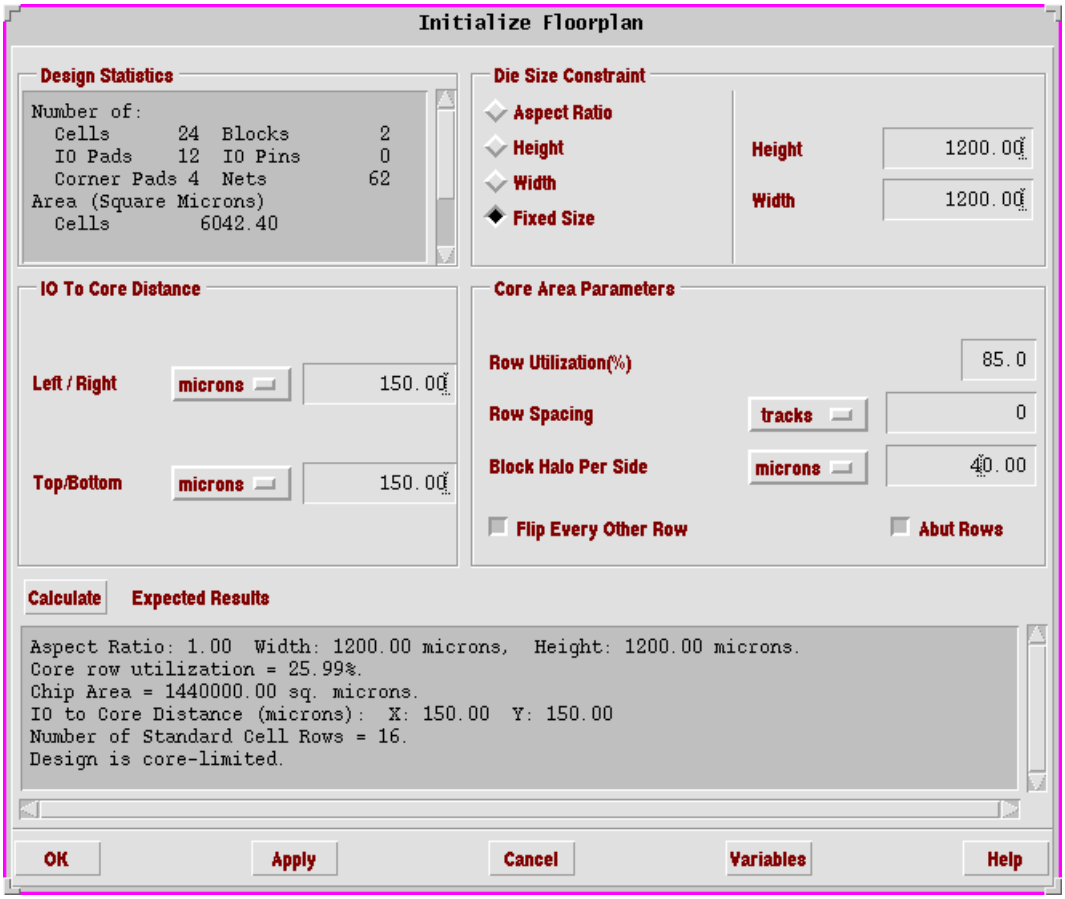

Figure 12: The Floorplanning Window.

Floorplanning

The floorplanning window is opened by *Floorplan* > *Initialize Floorplan* and should be filled out as in figure 12.

In the upper left part of the window there is a small area that lists some statistics of the design; number of cells, pads , and blocks. The number of nets and total area of the participating parts can also be found.

Under the heading **Die Size Constraints** some limiting parameters that affects the size of the design can be set. Usually you set a fixed size since you probably want to decide yourself exactly how the pads are to be distributed. **IO to Core Distance** is the space left between the pads and the **core area** of the chip. The core area will be filled with **Rows** in which the standard cells will be placed. **Row Utilization** is the ratio of standard cell area and row area. To pack the rows as close as possible use **Flip Every Other Row**, which puts p-area against p-area of the cells, and **Abut Rows** to push the rows into close contact. The free area that you want around the blocks for power supply is choosen with **Block Halo Per Side**.

When the form has been filled out click on **Calculate** and make sure that there are no problems reported, if not **OK**.

Placing Pads and Blocks

Now, there are some structures visible in the artwork window. Some big red rows for the pads and smaller rows in the middle for the standard cells. Also two green structures should be visible in the lower left part. These are the two blocks (ndiode and adblockDig).

To place the pads a **placement file** is used. There is no need to create one from scratch, the system can do that. Click on *Place > IOs* and select **I/O Constraint File** followed by **Write** and **Edit**. Now you have the file in an editor where you can rearrange the pads. The one for the lab can be used by the command *ioplace file padplace.mac;* from the command line.

The blocks are placed manually. Make the blocks selectable by checking the **Sl** column in the **Cell** row in the upper left part of the window. Then do *Edit > Move* and place the blocks rotated 90 degrees inside the row area as in figure 13.

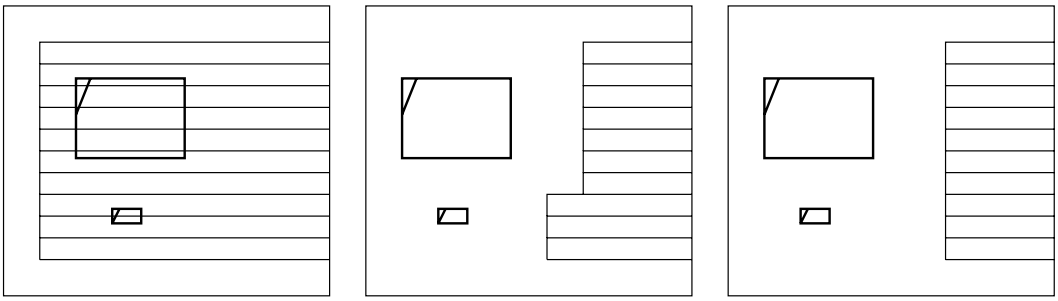

Figure 13: Various stages of Block Placing.

Floorplan > Update Core Rows will remove the rows from the block area. After that it will probably look as in the middle figure. To fix this, select the row and drag on the middle of the squares on the short edge. Zoom in to make sure you get the rows aligned.

Write *save bplaced;* on the command line to save the design under the selected name. This should be done once in a while in case something unpleasant happens with the system. To recall use *File > Open* or write *fload bplace;*.

Plan for the Power Distribution

Since the power lines has to feed a lot of current to the various parts of the design it is handeled separately. By *Route > Plan Power* you start the power planning session. The yellow dotted line is where the system will build power wires. This can of course be altered but the row area should always be enclosed by power. Make sure it looks like figure 14 and click on **Add Rings**. Fill in the form as in the figure and press **OK**.

If you have a long row area, power to the middle cells might be bad. To counter this click on **Add Stripes** and enter the specifications.

If something goes wrong in this step, all the power rings can be removed by *Floorplan > Reset Floorplan* and selecting **Delete all special wires**.

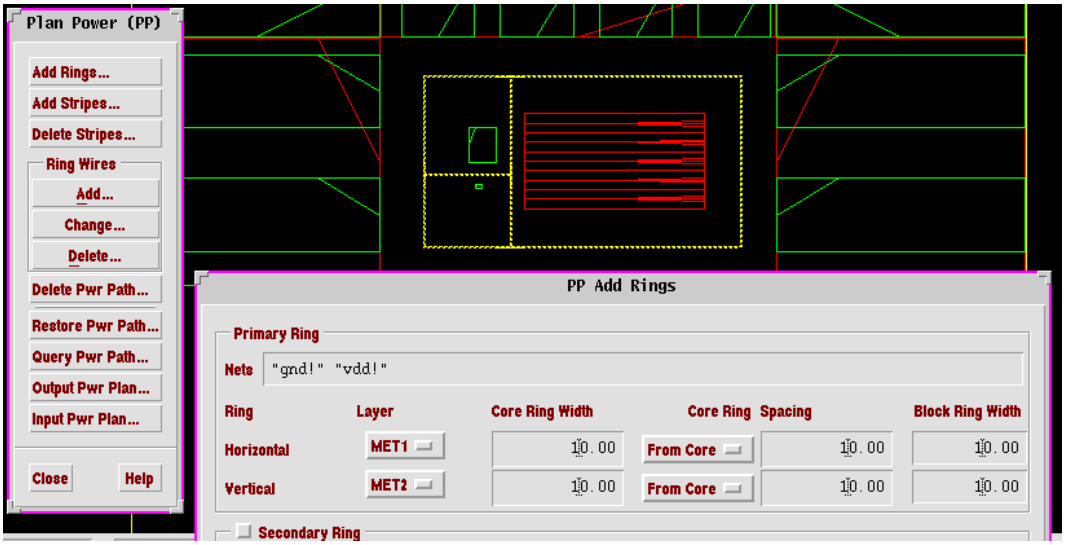

Figure 14: Power Planning.

Placing Cells

The standard cells are placed with *Place > Cells*. Leave all the options blank. The cells are now distributed in the row area. It is possible to influence the placement with a numerous of different constraints, delays, wire length, grouping of cells, etc.

Since the cells are not abutted (in contact) there might be problem with the layout rules. The n-tub layer in one cell could end up on an illegal distance from one in an other cell. To avoid this the space has to be filled with dummy cells. Just write *execute fillcore.mac;*

These filler cells are now included in the netlist are are not easily removed. Not even a new floorplan will get rid of them. A good idea is therefore to save the design *before* this step.

Routing

First the power has to be routed. A command file with all the necessary commands is named **allpower.mac;**. It could be executed, or the commands could be copied and pasted to the command line one by one. Usually there are problems in each step. They are presented here in the same order as in the file. Start the session with *Route > Connect Ring*, deselcect all types but **Stripes**, if there are stripes in the design, and click on **Apply**. The stripes will be connected to the rings.

Next Select **Block** and **All Ports**. Note the possibility of routing selected blocks and affecting the size of the wires. Default is the size of the pin in the block. The system prefers MET1 (blue) in the horizontal direction which is why you were recommended to rotate the blocks.

Followpins are supposed to connect the standard cells to the power ring. This is done on the left and right of the rows.

IO Pad and **Allports** takes care of the supply from the power pads to the power ring. Now, it should be obvious that the power pads are best placed in the top and bottom rows, for easy connection to the rings.

The sideways connection from pad to pad, that will form the **IO Ring**, which are used for the pads own power supply are not routed in this way. Instead the empty space between pads are filled out with small cells that will form the connections needed. This routine is called upon by the command *exec fillperi.mac;*.

To route the ordinary wires select *Route > Wroute*. Make sure that **Global and Final Route** is selected before pressing **OK**.

At this point it might look something like figure 15.

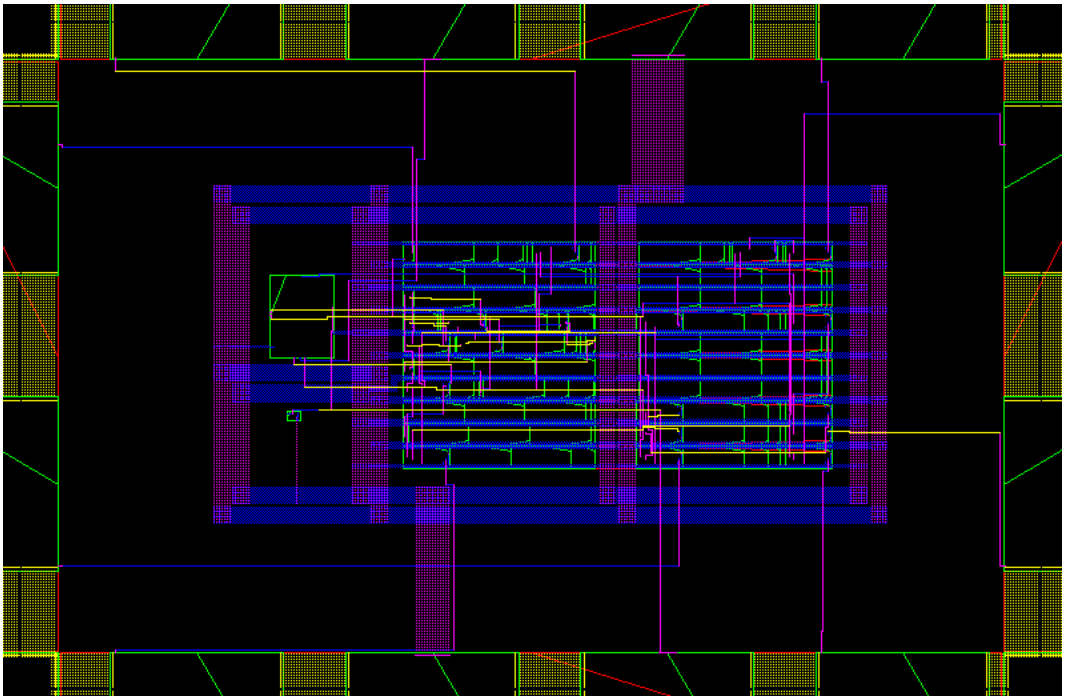

Figure 15: The final design.

Verification of the Design

Silicon Ensemble is not perfect and is capable of making errors. Some of these can be detected by running the various checking commands available.

Verify > *Connectivity* will report on open nets (i.e. something is not connected) or some segment of a net that has not been routed.

Verify > *Geometry* will examine the routing and report on any layout rules broken, wire to wire spacing, wire to cell content, minimum sizes, etc.

The errors, along with a short description, can then be viewed by *Edit* > *Find Info*.

Generating Output Data

Start by setting the design name which have disappeared. Use *Edit* > *Properties* and scroll down to **NAME.DESIGN** and change it to **AD_Converter**.

Since the design has to be brought back into Cadence, in order to replace some of the abstract views with the layout, a def file has to be generated. Select *File* > *Export* > *DEF* and click on **All**. Since there already exists a def files of the design (generated by Cadence) another name for the output file must be selected

(AD_ConverterSE.def).

Use *quit;* to leave Silicon Ensemble. It will not accept the command if there are non-saved modifications.

Reading a DEF file into Cadence dfII

The command *File* > *Import* > *DEF* will bring up the form showed in figure 16. Note how it is filled in. The same library and cell name as before should be chosen. The view should be **layout**.

Figure 16: Reading DEF file into dfII.

Exchanging views

In order to run a **Design Rule Check** without to many faulty errors some of the views in the design has to be changed. The **abstract** views contains layers that should not exist on a layout, f. ex. all metal with the **nt** purpose.

The view exchanging is easier done in another tool window, which are activated by *Tools > Floorplan/P&R > Silicon Ensemble*. Usually this is the tool in which the design is opened, which is shown by the letters **SE** in the top right corner. Make sure that only **Instance** is selected in the small **OSW** window. Select all the cells by dragging a box over the entire design with the left button. Deselect your own blocks by holding down **Ctrl** and clicking on them. *Floorplan > Replace View* will bring up a form in which you can change the view of all the selected cells at the same time. Chose the view **abstract_mlvs**. Next selct yout own blocks (**Shift** + click adds to select list) and change the view to **layout**.

Rulechecking

Go back to the layout editor (*Tools > Layout*) and start the DRC (*Verify > DRC*). The following switches can be used to minimize the number of errors, **no_antenna no_coverage no_erc no_metal_slots**. Here **Ctrl** is used to make multiple choises. Study the errors carefully, talk to someone who knows what they mean if you don't. Ignore errors in the pad area.

Make an extra run with the switch **reset_DRC** when you are ready. This is to clear the error markers from the design so that they won't end up in the final output.

Dont forget to save the design!

Generating Final Output

Before the design can be sent for fabrication, the views has to be changed to the correct ones. **abstract** is used for the pads and standard cells and **layout** for everything homemade with a complete layout available.

The final output file is usually called a **stream** or **gds** file. The correct form is opened by the command *CIW: File > Export > Stream* and should be filled in as in figure 17. The same name should be used on the file as the top structure of the design. Check the log file afterwards for errors.

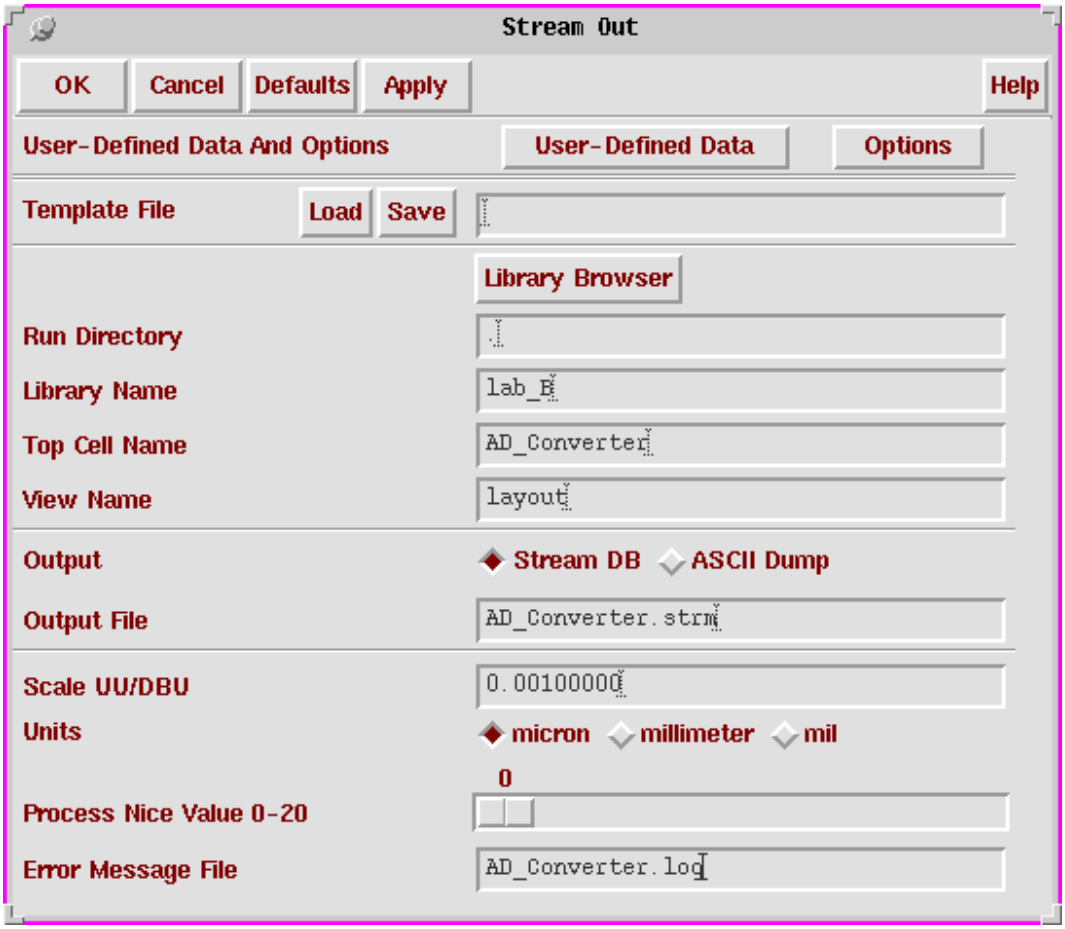

Figure 17: The Stream Out Window.

Project Instructions and Tips

When the construction is sent for fabrication it does not contain all necessary layout. This is filled in before the masks are generated. To avoid problem in that process **all blocks must have unique names.** This is easily done by using lowercase letters in the blockname.

When designing blocks, make the size of the pins at least 1x1 um and put them on the border of the block. It is also a good idea to separate them a little more than minimum allowed distance. This makes it easier for the abstract generator to create the abstract view of the block.

No strange characters like **+, -, #, $** etc. in library or cell names. Some of the tool used can not handle them.

Keep the name of the top structure short and simple, underscore (_) are not allowed.

References

[1] C. Andre, T. Salama, David G. Nairn and Henry W. Singor.
Current Mode A/D and D/A Converters.
IEE circuits and systems series 2, Analogue ICdesign: the current-mode approach, pp. 491-503, April 1990.

[2] S. Molund
CADENCE Condensed.
Department of Electroscience, Lund University 2001.
http://www.tde.lth.se/ugradcourses/cadsys/cadence.html