

CPE/EE 427, CPE 527, VLSI Design I: Tutorial #3, Standard cell design flow (from schematic to layout, 8-bit accumulator)

Joel Wilder, Aleksandar Milenkovic, ECE Dept., The University of Alabama in Huntsville

Adapted in part from University of Utah, Dept. of Electrical Engineering
Author: Allen Tanner

1. INTRODUCTION

In this tutorial you will use Cadence's schematic capture tool to create a ripple-carry accumulator circuit. To accomplish this, you will first build the Full Adder circuit (adding two 1-bit signals). Next, you will build a symbol to contain your circuit, which will be the building block that higher bit adders can be constructed from. Thus, using this Full Adder symbol, you will build a 4-bit accumulator circuit. Next, you will perform a functional test of this circuit using Cadence's Verilog Simulation tool (which interacts with SimVision). In order to perform the verilog simulation, you will create a verilog stimulus file that will be used to test your circuit. Subsequently, you will construct an 8-bit accumulator circuit on your own, and then functionally test it for accuracy. Next, you will add pads to the 8-bit accumulator circuit in order to construct a padframe that can be used in an ASIC layout, thus preparing your design to be fabricated. After the schematic is complete, you will use the tools in the Cadence schematic editor to synthesize your design and create a gate-level Verilog netlist. You will then simulate your synthesized netlist to ensure proper operation. Subsequently, you will use Encounter to perform auto place-and-route. Finally, you will perform verifications on this design by importing it into the Cadence icfb tool as a schematic and a routed design.

You will base your design on the 0.5um AMI nwell process ($\lambda = 0.30\mu\text{m}$).

2. CADENCE STARTUP

From your home directory, change directories into your cadence working directory:

```
$ cd cadence
```

Make a directory for lab3 and change into that directory:

```
$ mkdir lab3
```

```
$ cd lab3
```

Start the cadence tool:

```
$ icfb&
```

3. CREATE LIBRARY

In the Library Manager window, create a library to contain your lab3 work (call it lab3), and attach the **AMI 0.60u C5N (3M, 2P, high-res)** technology process to that new library.

4. CREATE RC ADDER CIRCUIT/SYMBOL

As was done in tutorial 2, add a new cell to your new library to contain the schematic of the Full Adder circuit and build it as shown in Figure 1. Note that this will be different from tutorial 2 in that you will not be building your circuit from the transistor level, but you will be instantiating gates from a pre-existing library that contains basic circuit building blocks. Use the **OSU_ami05** library for this purpose.

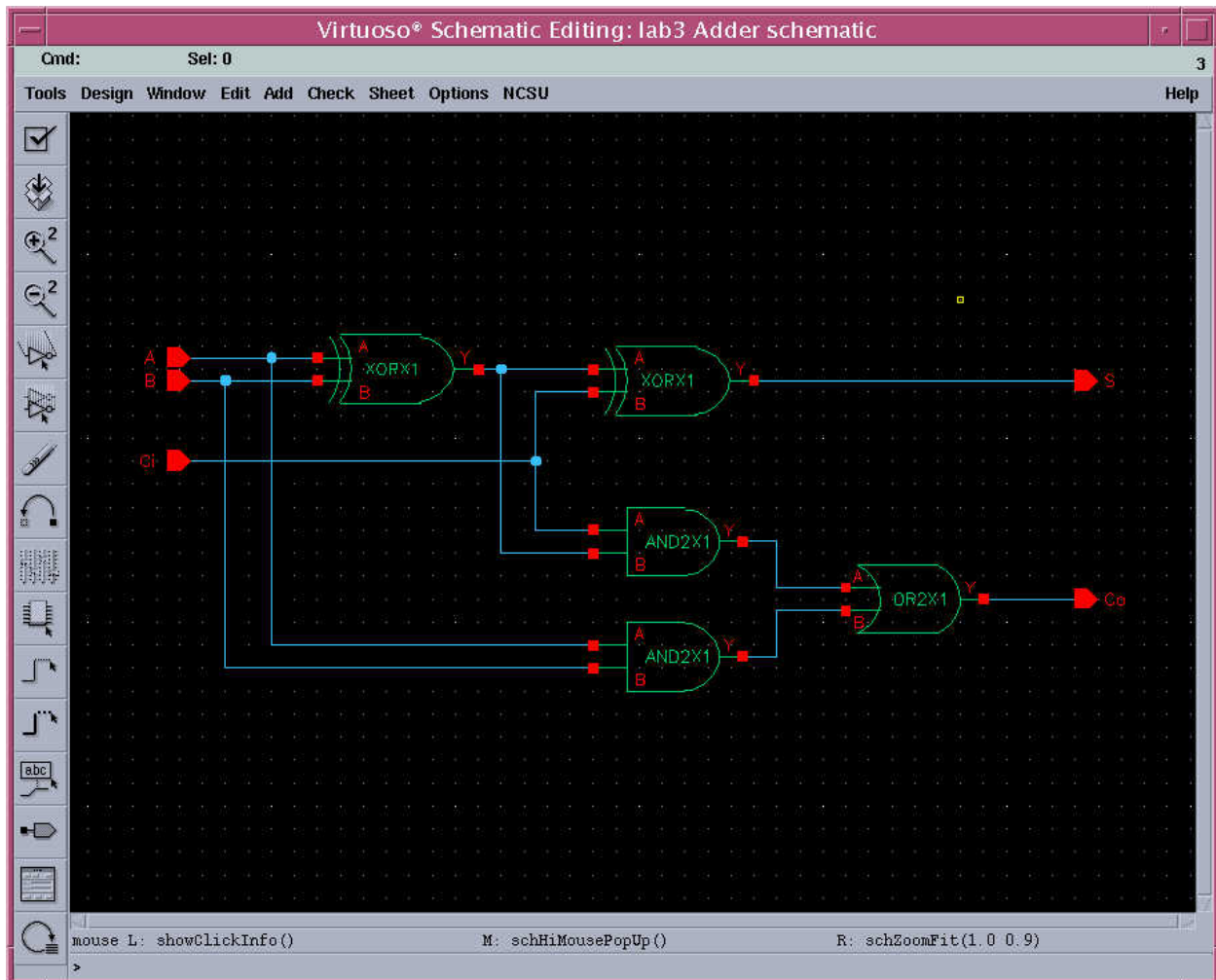


Figure 1. Full Adder Circuit.

Create the symbol for the Full Adder as shown in Figure 2. (Note that when you create the new cell in your library for the symbol, make sure that the name is the same as that used for the schematic cell. This will ensure that the circuit and symbol are “linked”.) In this tutorial, you will create a symbol from “scratch”, as opposed to importing the symbol as was done in tutorial 2.

The process of creating a custom symbol is done as follows:

- Go to **Add->Shape->Rectangle** to create the body of your symbol.
- Go to **Add-> Pin** to add the pins for your symbol, noting the direction for each pin you place (They should match the assignments of the pins that were made in the

schematic!). The rotate button also comes in handy here, so keep the pin creation interface window available.

- Go to **Add->Label**, and place the **[@instanceName]** label, which will serve as a placeholder for the reference designator. Repeat this once more, but change the label type to **nomalLabel**, type in the name of your symbol (i.e., RCAdder), and place this on your symbol.
- Go to **Add->Selection Box** and hit **Automatic**.
- Finally, go to **Design->Check and Save** and make sure there are no errors that occur in the CIW window.
- You can close the symbol editor window.

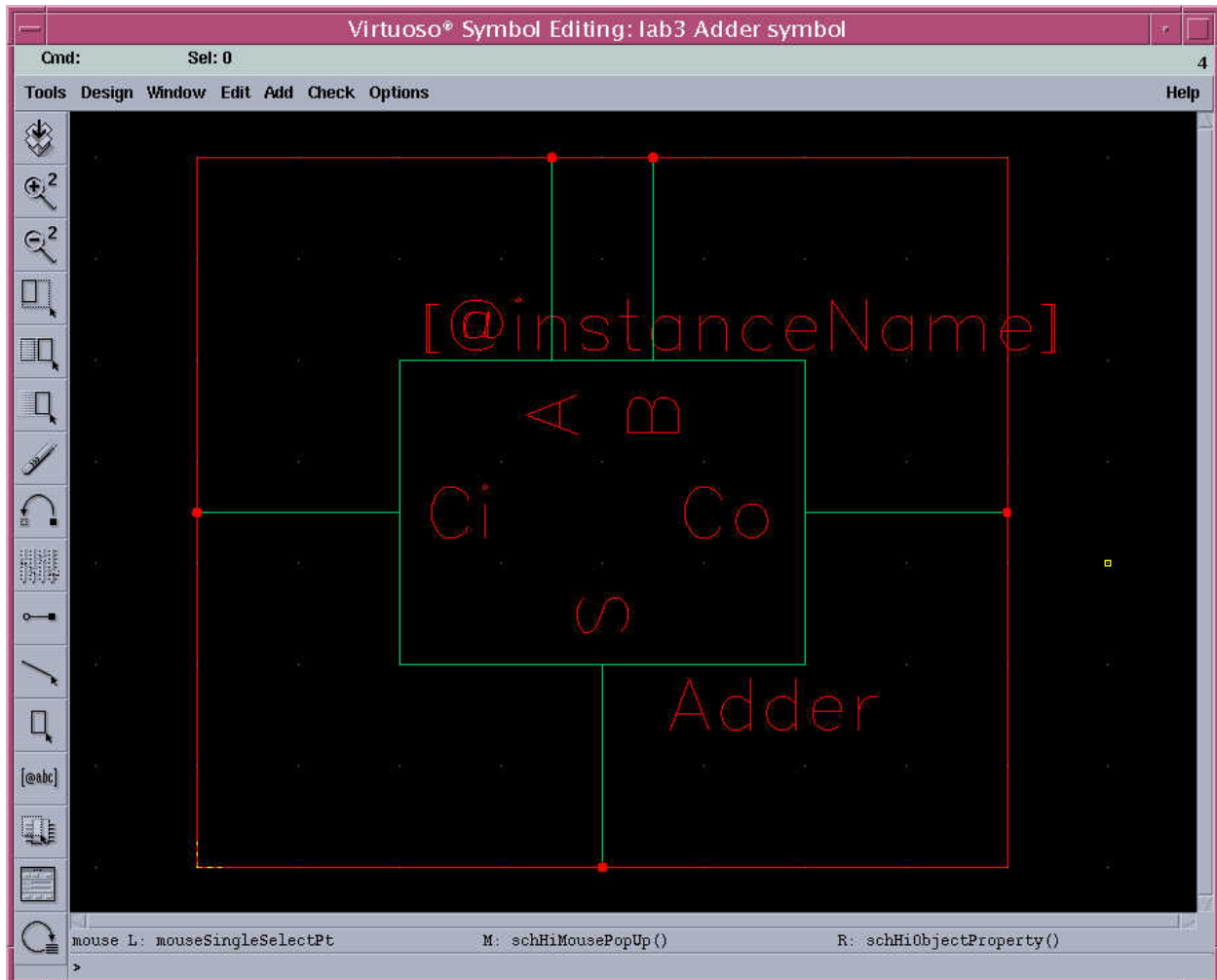


Figure 2. Full Adder Symbol.

5. CREATE 4-BIT ACCUMULATOR

Using your Full Adder symbol as a building block, you will create the 4-bit accumulator circuit as shown in Figure 3.

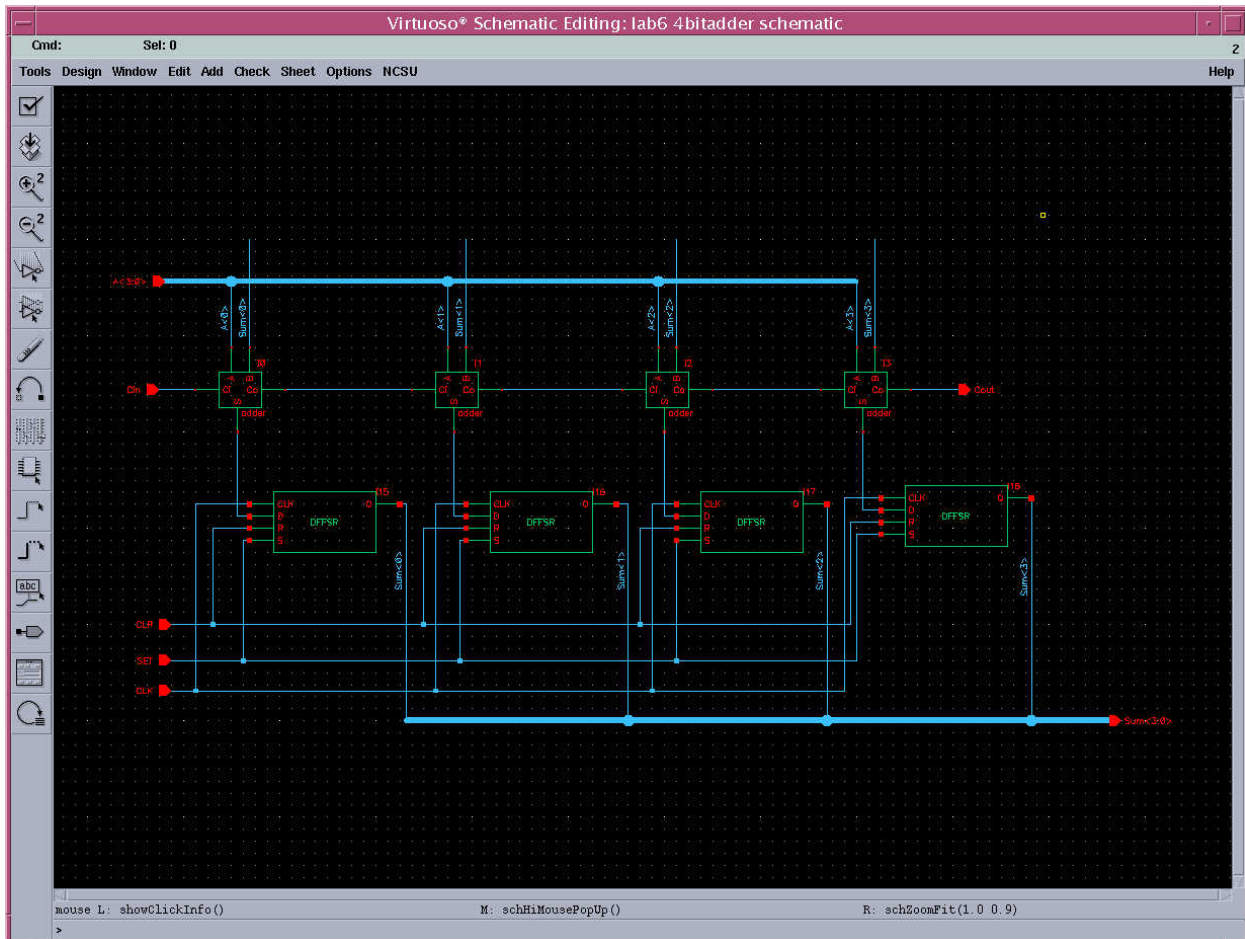


Figure 3. 4-bit Accumulator.

You will find the **DFFSR** symbol in the **OSU_ami05** library. The wide nets are buses, and they must be created with care, as follows (in the following order!):

1. Note the following terminology: wire equals a single bit; bus equals many bits, and can thus have several wires connected to it.
2. Extend single-bit wires from your symbol (you will have to double-click the “floating end”) that you want to group into a bus.
3. Label the wires according to how you want them grouped in the bus (i.e., Sum<1>, Sum<0>, etc). This will go ahead and assign net names to your wires.
4. Create a pin for your bus that will be named as you want the bus to be named (i.e., Sum<1:0>).
5. Position the pin so that it will be aligned with the ends of the wires that were created in #2.
6. Add a bus (type capital 'W' on your keyboard), first clicking on the pin, and then clicking on each wire that you want included in your bus. Clicking first on the pin will give the bus the same name as that of the pin.

Take note of the function of the accumulator, and that the outputs from the D flip-flops are “connected” back to the input by giving the 'B' input the same name as the output. (Naming a wire by the same name causes the netlist to join these two wires electrically.)

Check and save your schematic to ensure it is correct and free of errors.

6. SYNTHESIZE SCHEMATIC INTO A VERILOG NETLIST

In order to simulate your design, you first need to synthesize it into a verilog netlist. This is done in the following manner:

1. Go to **Tools->Design Synthesis->CSI** and a window will open as shown in Figure 4.

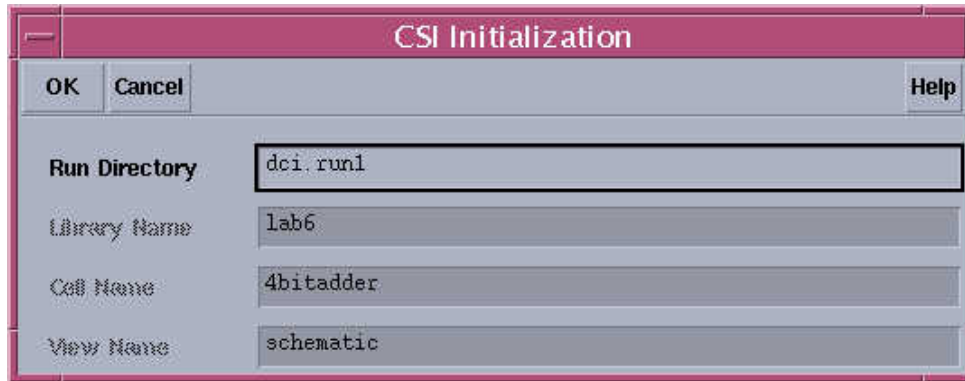


Figure 4. CSI Initialization.

This window specifies that you will generate the design synthesis in the directory dci.run1 (relative to your existing working directory). Select **OK**. After doing so, observe that there are several more pulldown menus available in the Schematic editor (having to do with synthesis).

2. Go to **Session->Setup->CSI...** and a window will appear as shown in Figure 5.

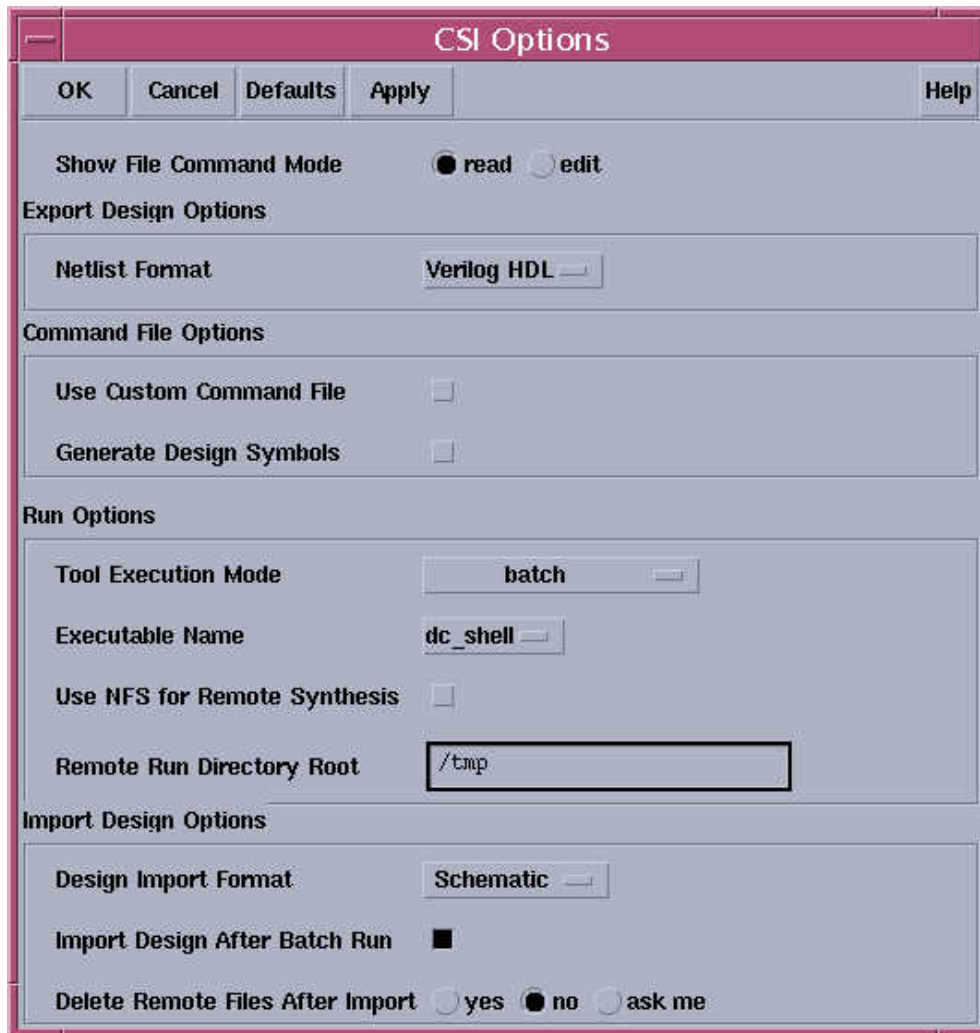


Figure 5. CSI Options.

Make the changes as shown in Figure 5 (in order to generate a Verilog netlist) and select **OK**.

3. Go to **Session->Setup->Verilog Netlister...** and a window will appear as shown in Figure 6.

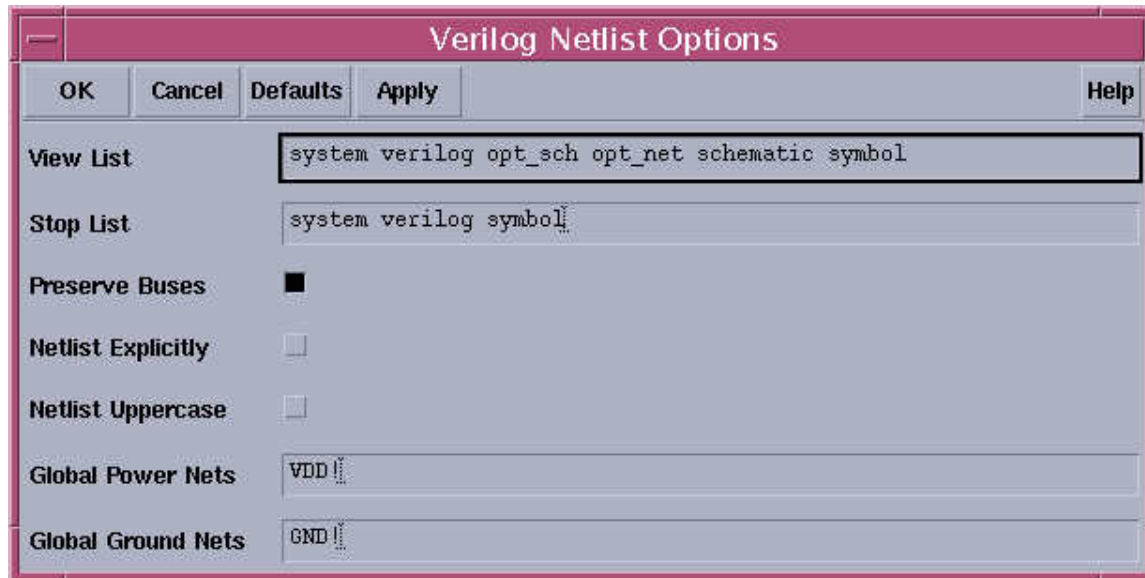


Figure 6. Verilog Netlist Options.

Remove behavioral and functional from the View List (this has to do with the hierarchical levels that are descended when performing the netlisting process) and select **OK**.

- Now that the settings have been made, go to **Run->Export Design** in order to initiate the synthesis process. This will create a file called “netlist” in the dci.run1 directory.

Once the netlisting has been completed (you can view the CIW window in order to check this), go to the dci.run1 directory in a terminal window and copy “netlist” up a directory and rename it accu.vh:

```
$ cd dci.run1
```

```
$ cp netlist ../
```

Go up a directory and open up the netlist for editing:

```
$ xemacs netlist&
```

Find the toplevel module in the code (should be named cdsModule_0 or something similar), which is basically the toplevel of your schematic where you instantiated the adders and DFFSR's and rename this module from cdsModule_0 to accu. In addition to the accu module, there is an adder module defined (from the adder symbol/circuit that you created), as well as the netlisting describing how the modules are connected. Save this file and close it.

Within your synthesized netlist, there are several adder modules. You might realize that these modules should have unique names and currently they do not (i.e., as is, the multiple instances of the adder module are all called the same name, meaning that the current netlist is wrong). You will fix this using an Encounter command to make the instances unique:

```
$ uniquifyNetlist -top accu accu.vh netlist
```

This command takes the netlist called “netlist”, where the accu module is the top module, and makes a unique netlist saved to the file accu.vh. You should view the accu.vh file to verify that the adder modules are now instantiated with unique names.

Now, this file is the synthesized, gate-level netlist of your schematic. Note that when synthesis is performed, it doesn't do any optimization, but just creates the netlist based on how the gates

are wired in the schematic. (This is different from when you create your design from an RTL-level verilog netlist and then have a synthesis program to generate a gate-level netlist for you – this will be done in the next tutorial. In that case optimization is done according to the timing constraints that you set for your clock net. Here, in synthesizing the netlist directly from the schematic, no such optimization is performed.)

7. FUNCTIONAL SIMULATION

Now that you have created your circuit and have synthesized a gate-level netlist, you will perform a functional simulation on it to verify that it is correct. This process is explained as follows.

1. The first thing you need to do is copy in a file you will need for simulation (copy this file into your lab3 working directory):

```
$ cp /apps/iit_lib/osu/osu_stdcells/flow/ami05/osu05_stdcells.v .
```

The osu05_stdcells.v file is necessary because it is a library file that contains definitions of the modules that are used in your accumulator design (i.e., the gates from the OSU library that you placed in your schematic).

2. Make sure you download the accu_test.v file into your lab3 working directory (accu_test.v is a verilog testbench that you will use to generate the test vectors to prove your design is functioning properly). Please be careful when you save the download. Do not leave an extra space at the end of the file name, or you won't be able to access it just with the name accu_test.v!! One thing you need to check is the ordering of the I/O for the accu module in your testbench relative to the ordering of the I/O for the same module as defined in your accu.vh netlist. Look for the following line in the accu_test.v file:

```
accu accu1(Cout, Sum, A, CLK, CLR, Cin, SET );
```

Find the module definition in your accu.vh file, and if necessary, adjust the accu_test.v file so that the I/O matches!

3. Add the following timescale definition to the top of your accu.vh file:

```
`timescale 1ns/10ps
```

This will simulate your design at 1 nano-second time units with a resolution of 10 pico-seconds.

4. Simulate the design:

```
$ verilog osu05_stdcells.v accu.vh accu_test.v
```


The testbench is set up to display results in the terminal window, which will give you a quick look to determine if your design is working properly. (you should notice the accumulator counting up!) The testbench also creates a database file containing all signals, which can be viewed in a graphical setting to provide a closer look at the performance of your design. SimVision is the tool that provides this graphical interpretation of the simulation of your design.

- Open up simvision:

\$ simvision&

In the Simvision Design Browser window, open the Waveform database by clicking on the “Open” symbol. Then double-click on “shm.db”, which is the folder where the file is located.

Inside the folder is only one file, shm.trn. Double-click on the file to open it. To see the contents of the waveform database, click on “stimulus” in the scope tree (as shown in Figure 7).

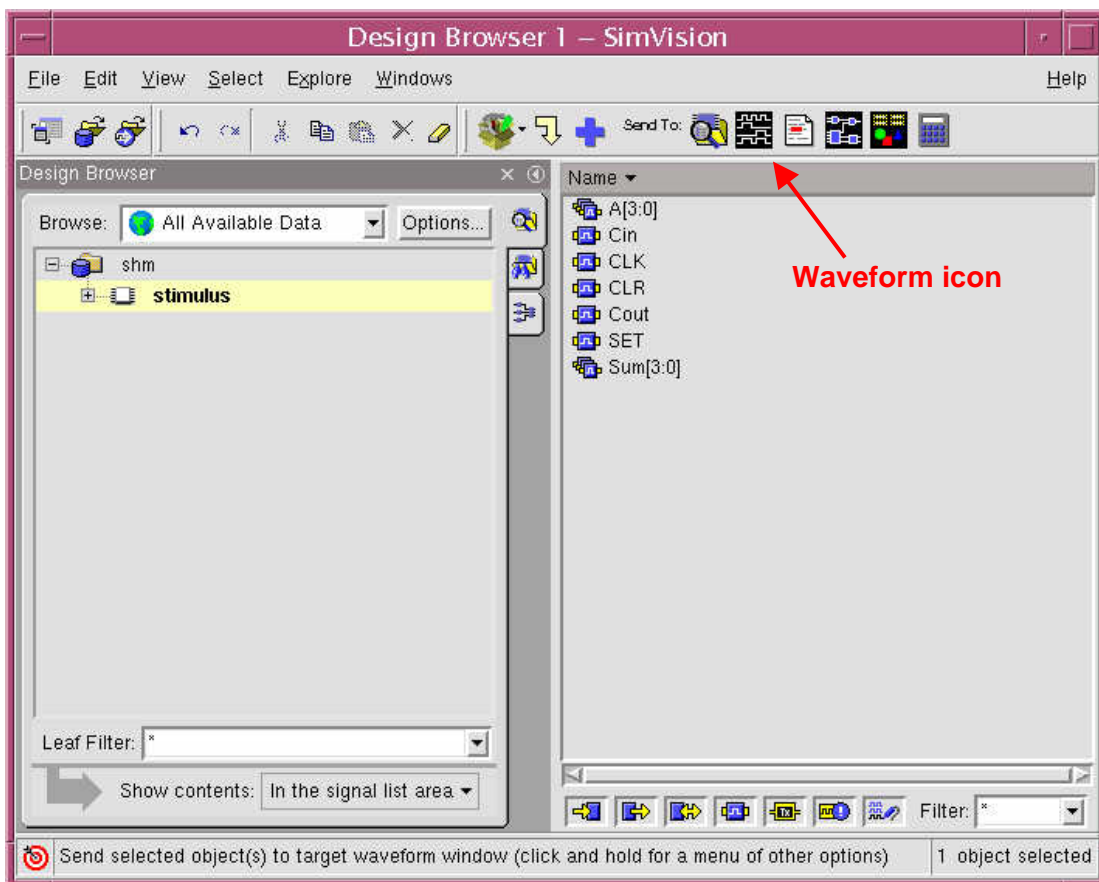


Figure 7. Design Browser window.

Now, to plot your waveforms, you need to select which signals you are interested in. In this case you will look at all waveforms. Select all waveforms on the right and display the waveform window (by clicking on the second icon from the right of “send to”, the waveform icon – see Figure 8). You will see your output similar to Figure 9. Verify that the accumulator is operating as expected.

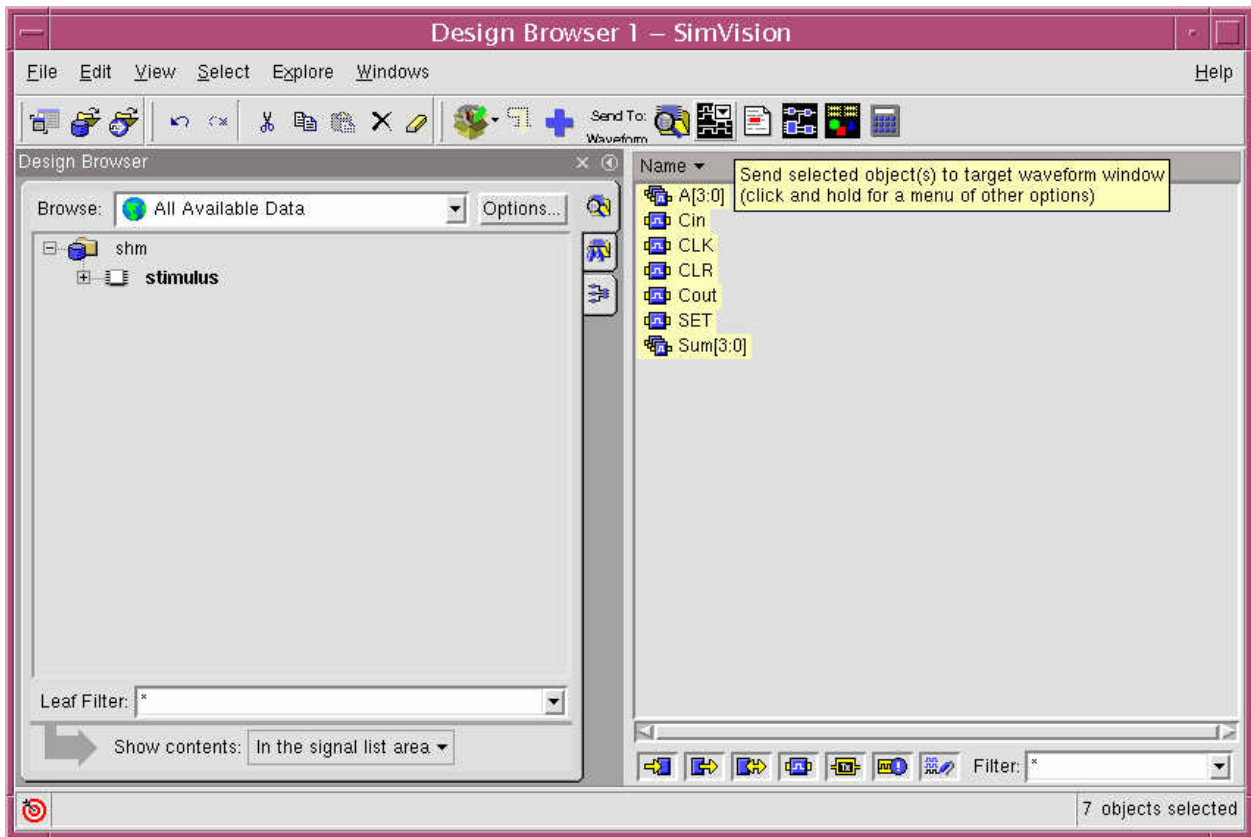


Figure 8. Selected signals to send to waveform window.

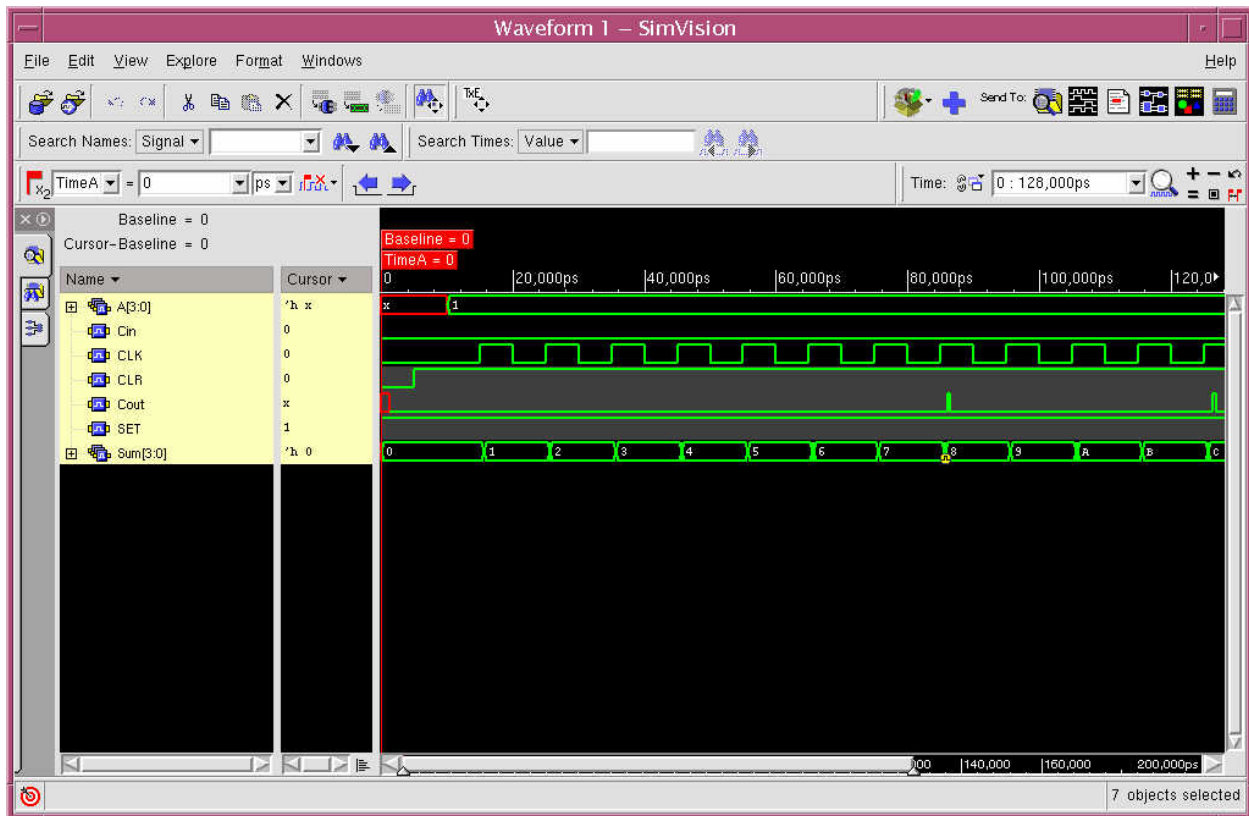


Figure 9. SimVision Waveform Window.

Note: The Waveforms in the **SimVision** window can be printed by selecting **File** → **Print Window...** option. While printing you can add Designer Name, Company Name, etc.

8. CREATE 8-BIT ACCUMULATOR

Now that you have the work flow down, go back and create an 8-bit accumulator, synthesize the gate-level verilog netlist, and simulate it to ensure it's working properly.

9. AUTO PLACE-AND-ROUTE OF THE CORE

Before adding a padframe to the design, you should auto place-and-route your core (base design without pads) to make sure it is routing as expected.

From the lab3 directory, use the mkdir command to create a folder called encounter, which will contain the technology files necessary for running auto place-and-route. Then copy the templates, as shown below:

```
$ mkdir encounter
$ cd encounter
$ cp /apps/iit_lib/osu/osu_stdcells/flow/ami05/* . (there is a space before the trailing '.')
```

The “encounter” folder now contains the template files for the AMI 0.5um technology.

Now you can use Cadence Encounter to place the standard cells and route them.

The result will be the final mask layout that could be shipped to the AMI foundry for fabrication. Cadence Encounter has a graphical user interface, however, it is faster and more convenient to execute it with a command file. This file is called "encounter.conf". Open this file for editing by typing:

\$ xemacs encounter.conf&

In the encounter.conf file, edit the following line as shown:

From:	To:
set my_toplevel MY_TOPLEVEL	set my_toplevel accu

Editing this file in this way points the tool to your accu design (Note: You will need to have your design, i.e., the accu.vh file, located in the same directory where you are running the Encounter scripts. So, place a copy of accu.vh into the encounter directory.). Overall, encounter.conf is a command file that is executed line by line. It includes commands to input the gate-level netlist, floorplan the chip, place the cells, route the cells and to verify the layout. In the end, the final layout is written in GDS-II format, which is the most popular format for IC layouts. It can be imported into Cadence Virtuoso and many other layout tools.

Before you can run encounter, you need a timing constraints file, which encounter will use as an input during the place-and-route process. Download the provided accu.sdc file into your encounter directory. (Please be careful when you save the download. Do not leave an extra space at the end of the file name, or you won't be able to access it just with the name accu.sdc!!) Depending on how you named your nets, you may need to edit that file. The existing file is shown in Figure 10 (compare your clock and I/O nets in your design to this constraints file, and modify it as necessary).

```

accu.sdc
##
## Cadence pks_shell "v5.14-s078 (May 29 2004 05:46:40)"
## Fri Sep 30 19:51:46 2005
##
## sdc_write_unambiguous_names global on
##
set sdc_version 1.4
current_design accu
create_clock [get_ports {CLK3}] -name vclk -period 10 -waveform {0 53}
set_driving_cell -lib_cell INVX8 -library osu05_stdcells -no_design_rule -max -pin Y -from_pin A [get_ports {A[3][3]}]
set_driving_cell -lib_cell INVX8 -library osu05_stdcells -no_design_rule -max -pin Y -from_pin A [get_ports {A[2][3]}]
set_driving_cell -lib_cell INVX8 -library osu05_stdcells -no_design_rule -max -pin Y -from_pin A [get_ports {A[1][3]}]
set_driving_cell -lib_cell INVX8 -library osu05_stdcells -no_design_rule -max -pin Y -from_pin A [get_ports {A[0][3]}]
set_driving_cell -lib_cell INVX8 -library osu05_stdcells -no_design_rule -max -pin Y -from_pin A [get_ports {CLK3}]
set_driving_cell -lib_cell INVX8 -library osu05_stdcells -no_design_rule -min -pin Y -from_pin A [get_ports {CLR3}]
set_driving_cell -lib_cell INVX8 -library osu05_stdcells -no_design_rule -max -pin Y -from_pin A [get_ports {CLR3}]
set_driving_cell -lib_cell INVX8 -library osu05_stdcells -no_design_rule -min -pin Y -from_pin A [get_ports {SET3}]
set_driving_cell -lib_cell INVX8 -library osu05_stdcells -no_design_rule -min -pin Y -from_pin A [get_ports {SET3}]
set_driving_cell -lib_cell INVX8 -library osu05_stdcells -no_design_rule -max -pin Y -from_pin A [get_ports {Cin3}]
set_driving_cell -lib_cell INVX8 -library osu05_stdcells -no_design_rule -min -pin Y -from_pin A [get_ports {Cin3}]
set_input_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {A[3][3]}]
set_input_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {A[2][3]}]
set_input_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {A[1][3]}]
set_input_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {A[0][3]}]
set_input_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {CLR3}]
set_input_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {SET3}]
set_input_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {Cin3}]
set_output_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {Sum[3][3]}]
set_output_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {Sum[2][3]}]
set_output_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {Sum[1][3]}]
set_output_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {Sum[0][3]}]
set_output_delay -add_delay 1 -clock [get_clocks {vclk3}] [get_ports {Cout3}]

```

Figure 10. Timing Constraints File.

This timing constraints file has a 100MHz clock frequency (10 nsec period) requirement.

Before you run encounter, let's discuss slack time. The most important number in digital logic design is the "Slack", or the safety margin when comparing data arrival time with respect to the clock frequency (as shown in Figure 11).

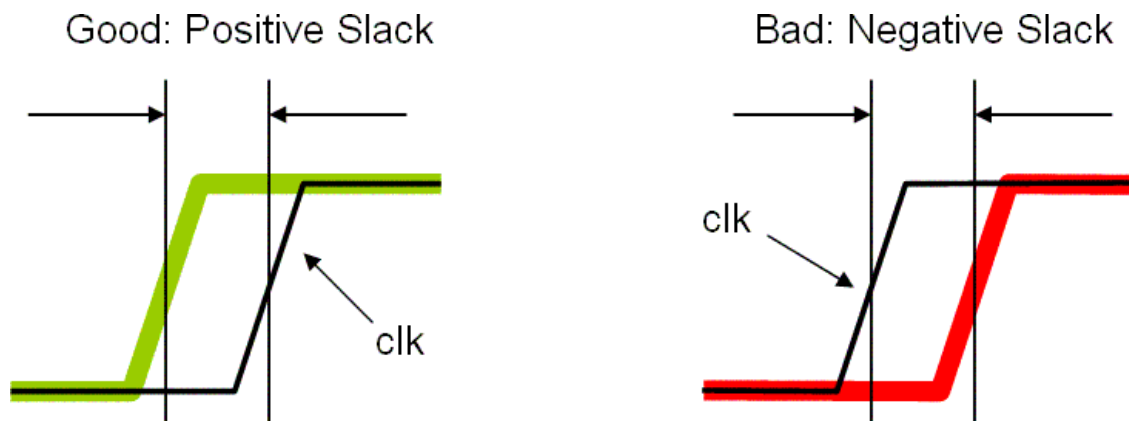


Figure 11. Slack Time.

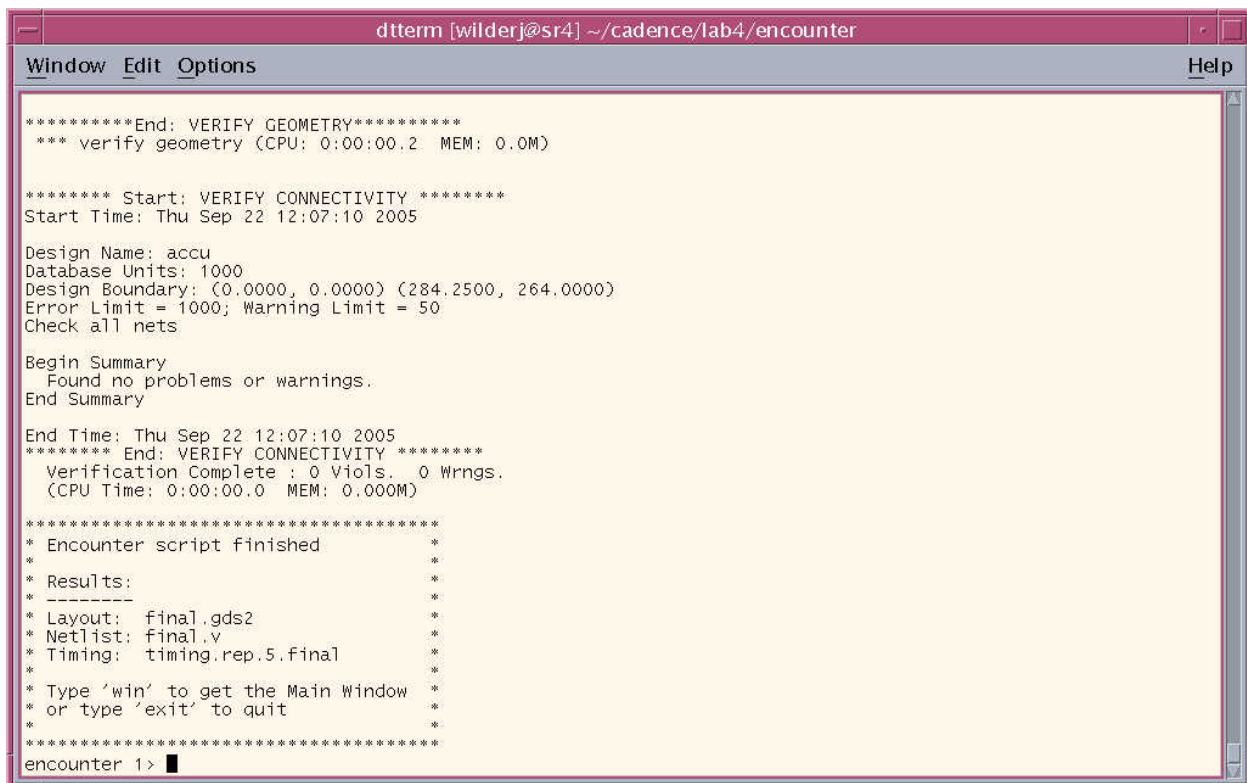
When you have a clock in your design, the data must arrive at the latch and meet the setup time of the latch before the rising edge of the clock. There's a certain amount of delay as the signal

flows through the circuit and arrives at the output latch. As shown in the picture on the left of figure 11, if the data signal arrives before the rising clock edge and meets the setup time requirement, this is what you want and results in a positive slack. If the data signal arrives after the rising edge of the clock, the circuit will not function at the frequency specified in the timing constraints file and the resulting slack is negative (propagation delay of the signal through the circuit is too slow and results in the signal arriving at the output flip-flop *after* the rising edge of the clock).

You are now ready to perform automatic place and route by typing the following command:

\$ encounter -init encounter.tcl

The results of this execution are shown in Figure 12. (If you have set everything up to this point, this should be a seamless, carefree process.)



```

dtterm [wilderj@sr4] ~/cadence/lab4/encounter
Window Edit Options Help
*****End: VERIFY GEOMETRY*****
*** verify geometry (CPU: 0:00:00.2 MEM: 0.0M)

***** Start: VERIFY CONNECTIVITY *****
Start Time: Thu Sep 22 12:07:10 2005

Design Name: accu
Database Units: 1000
Design Boundary: (0.0000, 0.0000) (284.2500, 264.0000)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Thu Sep 22 12:07:10 2005
***** End: VERIFY CONNECTIVITY *****
Verification Complete : 0 Viols. 0 Wrngs.
(CPU Time: 0:00:00.0 MEM: 0.000M)

*****
* Encounter script finished          *
* Results:                          *
* -----                          *
* Layout:  final.gds2                *
* Netlist: final.v                   *
* Timing:  timing.rep.5.final        *
*                                     *
* Type 'win' to get the Main Window *
* or type 'exit' to quit            *
*                                     *
*****
encounter 1> █

```

Figure 12. Auto place and route processing.

At the encounter 1 prompt in the terminal, type '**win**' to take a look at the resulting layout. Your layout should look similar to Figure 13.

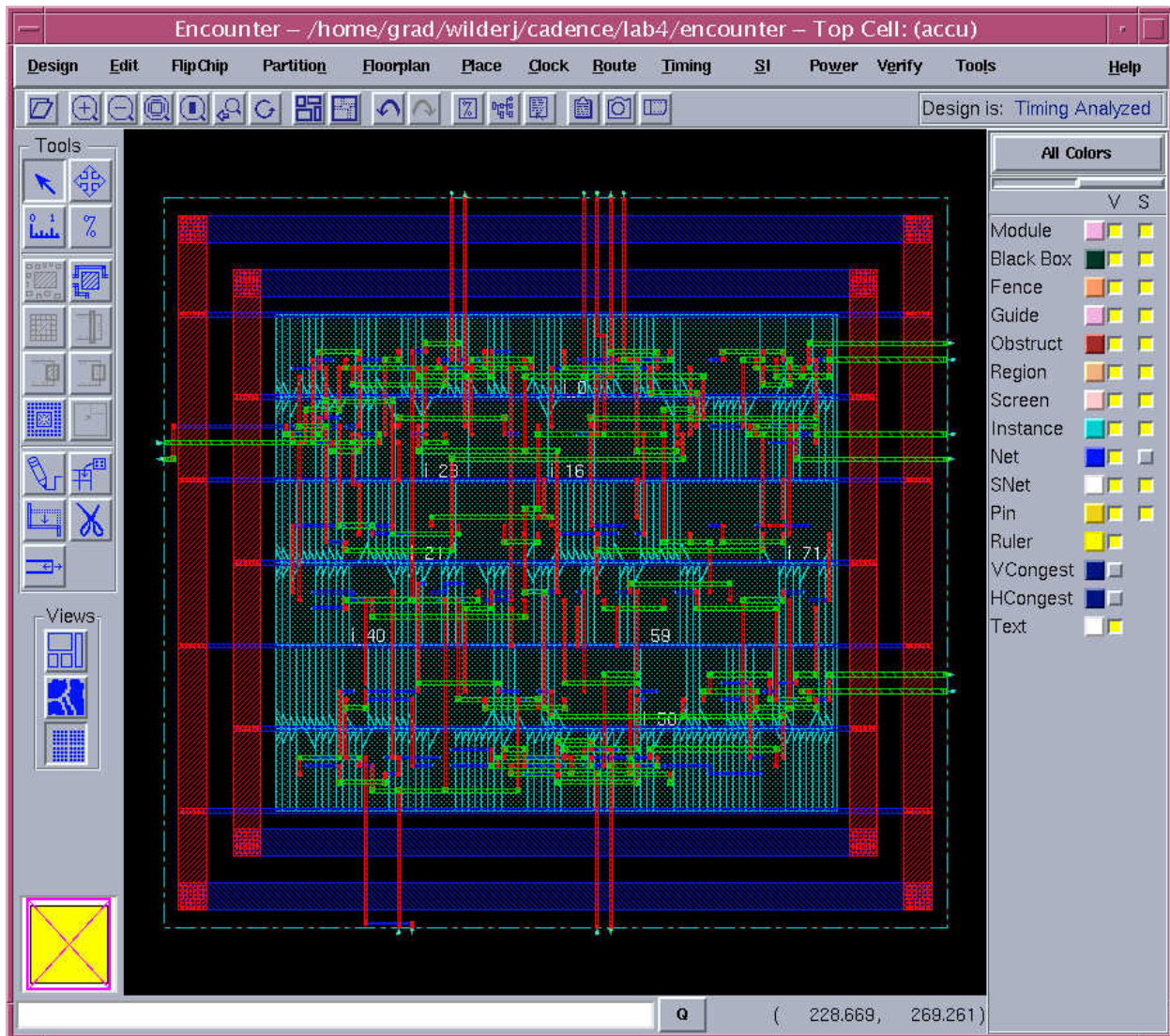


Figure 13. Encounter Layout.

Observe that the layout has been done for you by way of placing standardized cells from a pre-existing library. This is the way to go, since we don't have to do custom layouts from the transistor level! In Encounter, go to **Tools->Summary Report**. **Record the core size and chip size of this design** (you will be asked to hand this in). Note that while Encounter has a GUI interface for doing different step-by-step procedures to perform the auto place and route, we have done most of these things in the encounter.conf and encounter.tcl files. Further, Encounter can be used to analyze your layout via its GUI interface. Spend some time going through the tool menus, and then you can close Encounter and exit the program in the terminal.

Encounter performs a timing analysis for you, and uses this internally as it's optimizing the layout. The final timing report before actual layout can be found in file **timing.rep.5.final**. Have a look by typing

\$ more timing.rep.5.final

A snapshot of these results is shown in Figure 14. This file contains the 10 worst paths (relative to slack), with the worst offender shown **first**. (note that the picture shows the 10th worst path – and may be different for your design)

```

dtterm [wilderj@sr4] ~/cadence/lab4/encounter
Window Edit Options Help
| i_71/A | v | n_96 | XOR2X1 | 0.001 | 2.868 | 9.321 |
| i_71/Y | v | n_78 | XOR2X1 | 0.202 | 3.070 | 9.523 |
| i_132/B | v | n_78 | OAI21X1 | 0.000 | 3.070 | 9.524 |
| i_132/Y | ^ | n_112 | OAI21X1 | 0.129 | 3.199 | 9.653 |
| acc_reg_7/D | ^ | n_112 | DFFPOSX1 | 0.000 | 3.199 | 9.653 |
-----
Path 10: MET Setup Check with Pin acc_reg_7/CLK
Endpoint: acc_reg_7/D (^) checked with leading edge of 'vclk'
Beginpoint: in[1] (v) triggered by leading edge of 'vclk'
Other End Arrival Time 0.000
- Setup 0.347
+ Phase Shift 10.000
= Required Time 9.653
- Arrival Time 3.171
= Slack Time 6.482
Clock Rise Edge 0.000
+ Input Delay 1.000
+ Drive Adjustment 0.026
= Beginpoint Arrival Time 1.026
-----
| Pin | Edge | Net | Cell | Delay | Arrival Time | Required Time | |
|---|---|---|---|---|---|---|---|
| in[1] | v | in[1] | | | | 1.026 | 7.508 |
| i_15/B | v | in[1] | NOR2X1 | 0.001 | 1.027 | 7.509 |
| i_15/Y | ^ | n_33 | NOR2X1 | 0.113 | 1.140 | 7.622 |
| i_26/A | ^ | n_33 | OAI21X1 | 0.001 | 1.141 | 7.623 |
| i_26/Y | v | n_41 | OAI21X1 | 0.292 | 1.433 | 7.915 |
| i_192/A | v | n_41 | INVX1 | 0.004 | 1.437 | 7.918 |
| i_192/Y | ^ | n_113 | INVX1 | 0.183 | 1.619 | 8.101 |
| i_48/B | ^ | n_113 | OAI21X1 | 0.001 | 1.620 | 8.102 |
| i_48/Y | v | n_86 | OAI21X1 | 0.208 | 1.828 | 8.310 |
| i_45/C | v | n_86 | OAI21X1 | 0.001 | 1.829 | 8.311 |
| i_45/Y | ^ | n_54 | OAI21X1 | 0.214 | 2.043 | 8.525 |
| i_46/A | ^ | n_54 | A0I21X1 | 0.001 | 2.043 | 8.525 |
| i_46/Y | v | n_56 | A0I21X1 | 0.144 | 2.188 | 8.670 |
| i_64/C | v | n_56 | A0I21X1 | 0.001 | 2.188 | 8.670 |
| i_64/Y | ^ | n_91 | A0I21X1 | 0.239 | 2.427 | 8.909 |
| i_67/A | ^ | n_91 | OAI21X1 | 0.002 | 2.429 | 8.911 |
| i_67/Y | v | n_77 | OAI21X1 | 0.129 | 2.558 | 9.040 |
| i_70/C | v | n_77 | OAI21X1 | 0.000 | 2.558 | 9.040 |
| i_70/Y | ^ | n_76 | OAI21X1 | 0.155 | 2.713 | 9.195 |
| i_6614/A | ^ | n_76 | NAND2X1 | 0.001 | 2.714 | 9.196 |
| i_6614/Y | v | n_96 | NAND2X1 | 0.125 | 2.839 | 9.321 |
| i_71/A | v | n_96 | XOR2X1 | 0.001 | 2.839 | 9.321 |
| i_71/Y | v | n_78 | XOR2X1 | 0.202 | 3.041 | 9.523 |
| i_132/B | v | n_78 | OAI21X1 | 0.000 | 3.042 | 9.524 |
| i_132/Y | ^ | n_112 | OAI21X1 | 0.129 | 3.171 | 9.653 |
| acc_reg_7/D | ^ | n_112 | DFFPOSX1 | 0.000 | 3.171 | 9.653 |
-----
[VA 0.04]
[sr4 11] ~/cadence/lab4/encounter >

```

Figure 14. Post-layout slack time.

Locate the worst offender in the file and record the slack time (you will be asked to turn this in). Will the design function properly at 100MHz?

10. IMPORT LAYOUT DESIGN INTO CADENCE VIRTUOSO AND SCHEMATIC TOOLS

Let's generate a schematic and a layout design from the automated Encounter layout design to perform some final checks on the design. Encounter exports the routed design into a GDS2 stream, which is the standard for describing mask geometry. You will use this file to perform the conversion. First, you should download the following provided Cadence library files into your existing directory (should be \$HOME/cadence/lab3/encounter):

cds.lib

osu.lib

ncsu.lib

The Cadence library files are normally created as you build a schematic from scratch, but since the schematic and layout are being automatically generated from the routed design, you need to create these library files so that the Cadence schematic and layout can be viewed properly.

Next, perform the Encounter-to-icfb conversion by typing:

\$ osucells_enc2icfb

The results of this conversion should be similar to Figure 15.

```

dtterm [wilderj@sr4] ~/cadence/test
Window Edit Options Help
[\\A 0.09]
[sr4 91] ~/cadence/lab4/encounter > osucells_enc2icfb
/apps/cadence2005/local/lib
/apps/cadence2005/local/lib
/apps/iit_lib/osu/osu_stdcells/lib
Checking if final.gds2 exists.....OK
Determining top-level name.....OK (accu)
Creating temporary cds.lib.....OK
Determining Technology.....OK (AMI 0.5um)
Removing old library.....OK (accu)
Creating new DFII library.....OK (accu)
Creating PIP0 script file.....OK
Running PIP0 (GDS Stream-In).....
*****
*      CADENCE Design Systems, Inc.      *
*      Virtuoso(R) STREAM Interface 5.1.0 *
*      EXEC TIME : 22-Sep-2005  13:46:19  *
*      @(#) $CDS: pipo.exe version 5.1.0 06/11/2004 20:28 (cds125839) $      *
*****
Reading Stream File ...
*** There were 0 error and 1 warning messages ***
Statistic and more information, please check /home/grad/wilderj/cadence/lab4/encounter/PIP0.LOG file.
NORMAL EXIT ...

Creating IHDL script file.....OK
Running IHDL (Verilog In).....
@(#) $CDS: ihdl version 5.1.0 06/11/2004 20:23 (cds125839) $: (c) Copyright 1994-1995, Cadence Design Syst
*WARNING* techOpenTechFile: unable to open file techfile.cds in library accu in r mode
Analyzing design file(s) ...
Done
Creating Schematic ...
Generated sheet 1 for schematic accu
Cleaning up.....OK
Good by.

```

Figure 15. GDS2 conversion to Cadence layout/schematic.

You can start icfb at the prompt:

\$ icfb&

First, open up the schematic in your accu library. This schematic was automatically generated from the post-synthesized netlist. Perform a DRC on this schematic (check and save) and verify that there are no errors. **Print out the results of the DRC from the CIW window** (*you will hand in this print out to the instructor*). Close the schematic view.

Next, open up the layout view in your accu library (should be identical to what was generated in Encounter). You will need to go to **Options->Display** and change the setup from 0 to 32 (remember that you have had to do this in the past). You should now be able to see the routed

design. Have a look around the design. Zoom in and try to select a single diffusion region for one of the transistors. Notice that this is not possible. Why? The transistors have been imported as cells from a standardized library and are packaged as one complete object. (recall the benefits of using a standardized library rather than doing this from scratch!) Perform a DRC on the layout (**Verify->DRC->OK**). In the CIW window, if you see any errors you have done something horribly wrong and will need to go back and start this lab over. OK, just kidding. You will see 20 or so errors having to do with improperly formed shapes. You can fix this. Go to **Verify->Markers->Find** and select **Zoom to Markers**, and then click **Next**. The marker will point to some text label. Select the text and type 'q' to bring up the properties. Notice that this text label has been assigned to a metal layer and will need to be changed to the text-dg layer. You can do this with the menu selection. Change all of these (go back and select **Next** in the find markers window to find the next one), and then re-run DRC and verify that there are no errors in the CIW window. **Print out the results of the DRC from the CIW window** (you will hand in this print out to the instructor).

Next, you will extract parasitic capacitances from your layout by going to **Verify->Extract**. Click on **Set Switches** and select **Extract_parasitic_caps** in the resulting window and click OK. Your Extractor window should now look like Figure 16.

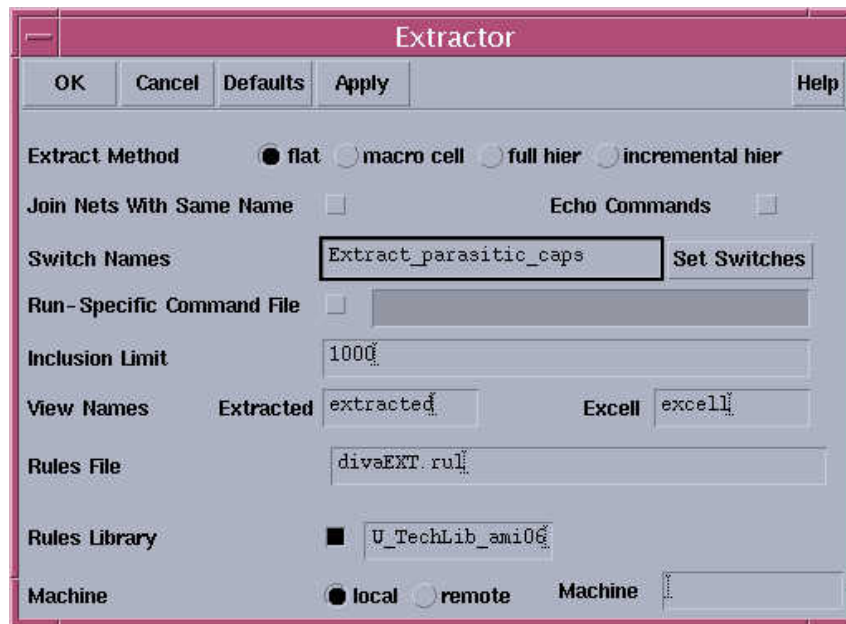


Figure 16. Extractor Window.

Click on OK to perform this operation. Now close the layout view of your design. Bring up the extracted view of your layout by double-clicking it from the library manager (Figure 17).

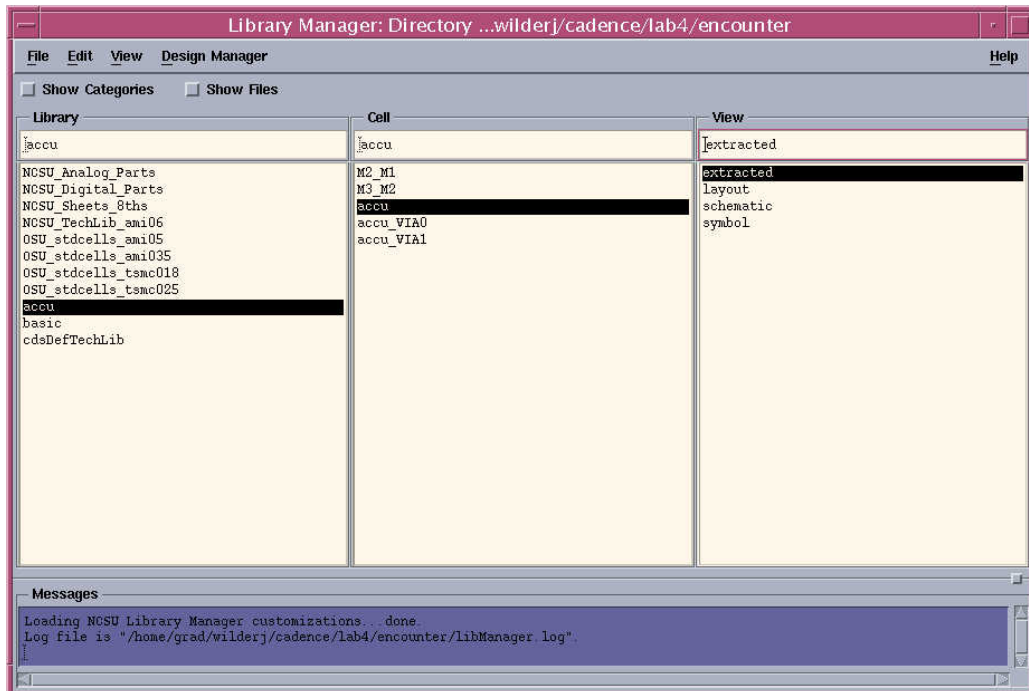


Figure 17. Find accu extracted view.

Zoom in on the extracted layout and find capacitors that have been placed in the design to represent the parasitic capacitances.

Next, you will perform a layout-versus-schematic comparison as another design check. Go to **Verify->LVS** and you will see the window as shown in Figure 18. Fill it out according to this window.

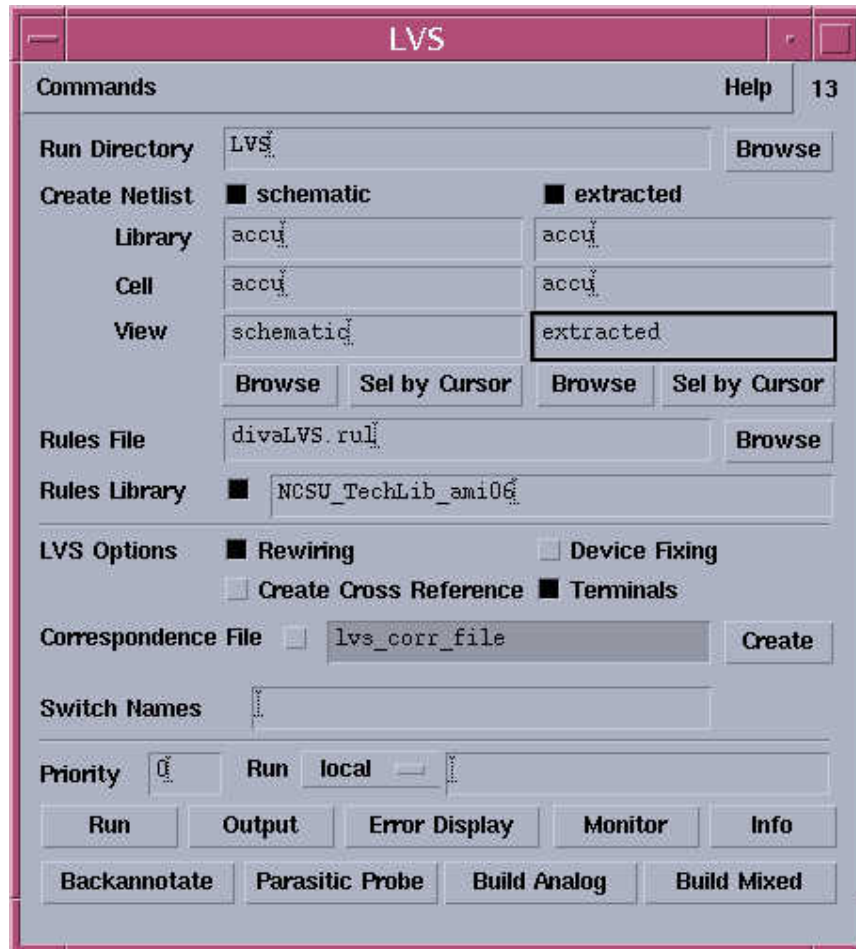


Figure 18. LVS Window.

Click **Run** and wait *patiently* for a popup window to tell you that the LVS process has completed successfully. Note that this is not telling you that the layout and schematic netlists are identical. To find out the results from the LVS, click on **Output**. Determine if the netlists match from the resulting window. **Print out these results** (*you will hand this in to the instructor*).

11. ADD PADFRAME TO DESIGN

Now that you've proven out your 8-bit accumulator design through simulation, routed the core, and checked for any errors in the place-and-route design through icfb schematic/layout conversion, you are ready to go back and add pads to the design, which are necessary for fabricating a chip. From the **OSU_ami05** library, you will add the following pads to your 8-bit accumulator schematic design:

- Corner pads (a total of 4): PADFC
- Power and ground pads (one each): PADVDD, PADGND
- Input pads (however many needed): PADINC
- Output pads (however many needed): PADOUT
- No-connect pads (however many needed): PADNC

A Mosis fabrication run requires 4 corner pads and 40 other pads, for a total of 44 overall pads. The instance names (reference designators) of the pads should be as follows (you will have to change the existing instance names in the schematic to match these – select the instantiated pad, press ‘q’, and change the name of the pad as follows):

- Corner pads: c01, c02, c03, c04
- All other pads: p01, p02, p03, ..., p39, p40

When you perform auto place-and-route, it will become apparent why these names have to be strictly named in this way.

The input and output pads should be connected so that the input/output pin has a net running to the pad, and then there’s a net on the other side of the pad running to the core design. Please be careful with the accumulator design and that your feedback lines stay within the core (and that they don’t run all the way to the pad and back – this would not be good!)

After you’ve added the padframe to your 8-bit accumulator design, ensure there are no design errors with your schematic, then generate the netlist and simulate it in order to prove it is working with the pads added.

12. AUTO PLACE-AND-ROUTE OF THE CHIP

Now, you are ready to place-and-route the chip design with the padframe added. You can perform this task in the same directory where the core was routed, or you can create a new encounter directory if you so desire (i.e., encounter_padframe). You can use the same timing constraints file here that you used for the core.

There are two additional steps that have to be taken when routing the chip (core plus pads). The first is to modify encounter.conf according to Figure 19. This modification tells Encounter to use encounter.io when orienting the pads for the chip. If the names of your pads (as placed in your schematic) do not match the names of the pads in the encounter.io file, you will have a problem (c01, c02, c03, c04, p01, p02, ..., p39, p40)! Change the name of the pads through the instance name. NOTE: The pads are accessed as normal through the **OSU_ami05** library by placing an instance, in the same way that you placed a MOSFET into your design. Further, for the pads that you must connect into your design, i.e., the input pads and the output pads, there are two connections for each pad: a PAD connection and a DI or DO connection. The DI/DO connection must be on your “core-side” and the PAD connection must be on the side of your pin (the input pin A<7:0> or the output pin Sum<7:0>). Also, as mentioned in the preceding section, be careful that when you feedback the Sum<7:0> lines back to your adder inputs, that they don’t feedback from the PAD pin, but from the DO pin. This keeps your feedback within the core and not running all the way from the core to the pad, and then back to the core. This would drastically increase your routing delays and would be very poor design practice!! Please also note how the encounter.io file controls the orientation of your pads. Note how the pad names correspond to the location in the padframe, i.e., on the “north side” or the “south side” or the “east side” or the “west side”. You may want to control on which side of the padframe the input pads are located (perhaps on the “west side”), and which side the output pads are located (perhaps on the “east side”). Further, maybe you would want to located your power pad on the “north side” and your ground pad on the “south side”. Place the other control pins where ever you like. These designations would depend on how your ASIC design interfaces with the overall system, were this design to be used as a sub-component in an overall system.

```

emacs: encounter.conf
File Edit View Cnds Tools Options Buffers Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News

encounter.conf
#####
#
# FirstEncounter Input configuration file      #
#                                             #
#####
#
# Specify the name of your toplevel module
set my_toplevel accu

# Only if your design as pads:
# - make sure the names in 'encounter.io' match your pad names
# - uncomment the following line
set rda_input(ui_io_file) "encounter.io"
#####
# No changes required below
#####

global env
set OSUcells $env(OSUcells)

global rda_input
set rda_input(ui_netlist) $my_toplevel.vh
set rda_input(ui_timingcon_file) $my_toplevel.sdc
set rda_input(ui_topcell) $my_toplevel

set rda_input(ui_netlisttype) {Verilog}
set rda_input(ui_11m1list) {}
set rda_input(ui_settop) {1}
set rda_input(ui_cell111b) {}
set rda_input(ui_iolib) {}
set rda_input(ui_areaolib) {}
set rda_input(ui_blklib) {}
set rda_input(ui_kboxlib) ""
set rda_input(ui_time1b) "$OSUcells/11b/ami05/11b/osu05_stdcells.t1f"

-----XEmacs: encounter.conf (Fundamental)-----Top-----
Loading efs-cu...done

```

Figure 19. Modify encounter.conf as shown.

The next required modification is to the encounter.tcl file, which should be changed according to Figure 20. These changes adjust the spacing from the padframe to the core, as well as keeping the supply rings from being centered on the chip. If you fail to make these changes, your routed design will not be correct.

```

emacs: encounter.tcl
File Edit View Cmds Tools Options Buffers Tcl Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News
encounter.tcl
#####
# Run the design through Encounter
#####
# Setup design and create floorplan
loadConfig ./encounter.conf
commitConfig

# Create Floorplan
floorplan -r 1.0 0.6 450 450 450 450

# Add supply rings around core
addRing -spacing_bottom 9.9 -width_left 9.9 -width_bottom 9.9 -width_top 9.9 -spacing_top 9.9
.9 -layer_bottom metal1 -width_right 9.9 -around core -layer_top metal1 -spacing_right 9.9
-spacing_left 9.9 -layer_right metal2 -layer_left metal2 -offset_top 9.9 -offset_bottom 9.9
-offset_left 9.9 -offset_right 9.9 -nets { gnd vdd }

# Place standard cells
amoebaPlace

# Route power nets
sroute -noBlockPins -noPadRings

# Perform trial route and get initial timing results
trialroute
buildTimingGraph
setCteReport
reportTA -nworst 10 -net > timing.rep.1.placed

# Run in-place optimization
# to fix setup problems
setIPOMode -mediumEffort -fixDRC -addPortAsNeeded
initECO ./ipo1.txt
fixSetupViolation
endECO
buildTimingGraph
setCteReport

-----XEmacs: encounter.tcl (Tcl)-----Top-----
Loading tcl...done

```

Figure 20. Modify encounter.tcl as shown.

After routing your design with Encounter (view the core size/chip size of your integrated circuit, **you will hand this in**) and verifying its correctness, import the design into Cadence Virtuoso and Schematic editors (as was done for the core). Check for any DRC errors in the schematic and layout views (Note: Don't worry about fixing errors in the layout view. The pad designs don't meet the conservative DRC requirements of your core, and since they came from a standard library, you can't do anything about them. They will result in thousands of errors! The schematic view should **not** have any errors.)

Make printouts of your post-routed schematic and extracted layout (*you will hand these in*)

13. ASSIGNMENT

Make an 8-bit accumulator circuit based on the preceding work flow and hand in the following:

- For core design only (no pads):
 - Core size, chip size of routed design (10%)

- Post-routed, worse offender slack time (10%)
- Schematic DRC results (10%)
- Layout DRC results (10%)
- Results from LVS (10%)
- For integrated circuit design (core plus pads):
 - Printouts of post-routed schematic and extracted layout (10%)
 - Core size, chip size (10%)
 - Answer the question: Will the 8-bit accumulator integrated circuit design operate at 350MHz? What is the resulting slack time of the post-routed design? (30%)