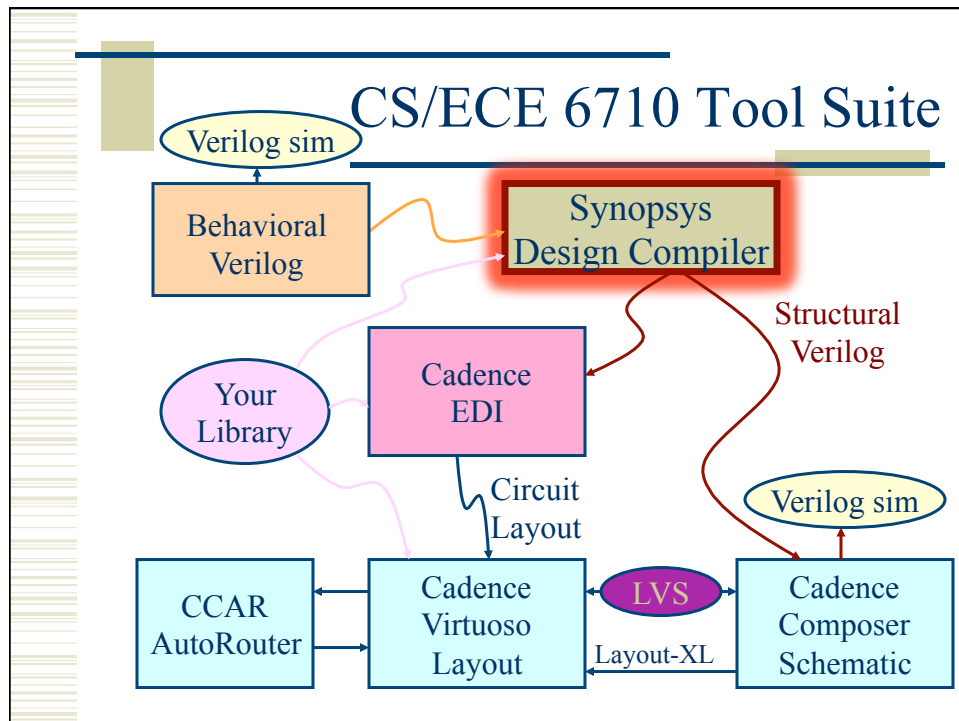


Synthesis and Place & Route

Synopsys design compiler
Cadence Encounter Digital
Implementation System (EDI)



Design Compiler

- ◆ Synthesis of behavioral to structural
- ◆ Three ways to go:
 1. **Type commands to the design compiler shell**
 - Start with syn-dc and start typing
 2. **Write a script**
 - Use syn-script.tcl as a starting point
 3. **Use the Design Vision GUI**
 - Friendly menus and graphics...

Synthesis Process: Design Compiler

1. Define synthesis environment
2. Read in your behavioral Verilog
3. Set synthesis constraints (speed, area, etc.)
4. Compile (synthesize) the design
5. Write out the results

Design Compiler – Basic Flow

1. Define environment
 - **target libraries** – your cell library
 - **synthetic libraries** – DesignWare libraries
 - **link-libraries** – libraries to link against
2. **Read** in your structural Verilog
 - Usually split into **analyze** and **elaborate**
3. Set constraints
 - timing – define **clock**, **loads**, etc.

Design Compiler – Basic Flow

4. Compile the design
 - **compile** or **compile_ultra**
 - Does the actual synthesis
5. Write out the results
 - Make sure to **change_names**
 - Write out **structural verilog**, **report**, **ddc**, **sdc** files

beh2str – the simplest script!

```
[elb@lab2-12 cadence]$ beh2str
beh2str - Synthesizes a verilog RTL code to a structural code
         based on the synopsys technology library specified.
Usage   : beh2str f1 f2 f3
         f1 is the input verilog RTL file
         f2 is the output verilog structural file
         f3 is the compiled synopsys technology library file
```

beh2str – the simplest script!

```
[elb@lab2-12]$ beh2str
beh2str - Synthesizes a verilog RTL code to a structural code
         based on the synopsys technology library specified.
Usage   : beh2str f1 f2 f3
         f1 is the input verilog RTL file
         f2 is the output verilog structural file
         f3 is the compiled synopsys technology library file

[elb@lab2-12]$ beh2str addsub.v addsub_dc.v Lib6710_00.db
....
Thank you...
[elb@lab2-12]$ ...results in addsub_dc.v and addsub_dc.v.rep
```

addsub.v

```
module addsub (a, b, addnsb, result);
  parameter SIZE = 8; // default word size is 8
  input [SIZE-1:0] a, b; // two SIZE-bit input
  input addnsb; // Control bit: 1 = add, 0 = sub
  output reg [SIZE:0] result; // SIZE+1 bit result

  always @(a, b, addnsb)
  begin
    if (addnsb)
      result = a + b;
    else result = a - b;
  end
endmodule
```

addsub_dc.v

```
module addsub ( a, b, addnsb, result );
  input [7:0] a;
  input [7:0] b;
  output [8:0] result;
  input addnsb;
  wire n8, n9, n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n20, n21,
    n22, n23, n24, n25, n26, n27, n28, n29, n30, n31, n32, n33, n34, n35,
    n36, n37, n38, n39, n40, n41, n42, n43, n44, n45, n46, n47, n48, n49,
    n50, n51, n52, n53, n54, n55, n56;
  XNOR2X2 U4 ( .A(addnsb), .B(n8), .Y(result[8]) );
  AOI21X2 U5 ( .A(n9), .B(n10), .C(n11), .Y(n8) );
  AOI21X2 U6 ( .A(n12), .B(n13), .C(a[7]), .Y(n11) );
  ...
  IN VX1 U60 ( .A(a[0]), .Y(n52) );
  XNOR2X2 U61 ( .A(b[0]), .B(addnsb), .Y(n54) );
endmodule
```

58 cells used

addsub_dc.v.rep

Operating Conditions: typical Library: foo_typ

Wire Load Model Mode: top

Startpoint: b[0] (input port)

Endpoint: result[8] (output port)

Path Group: (none)

Path Type: max

Point	Incr	Path
input external delay	0.00	0.00 r
b[0] (in)	0.00	0.00 r
U61/Y (XNOR2X2)	0.67	0.67 r
U56/Y (IN VX1)	0.57	1.24 f
...		
U5/Y (AOI21X2)	0.42	8.62 r
U4/Y (XNOR2X2)	0.47	9.09 r
result[8] (out)	0.00	9.09 r
data arrival time		9.09

(Path is unconstrained)

addsub_dc.v.rep

Library(s) Used:

foo_typ (File: /home/elb/VLSI/cadence-f13/syn-f13/trythis/Lib6710_00.db)

Number of ports: 26

Number of nets: 75

Number of cells: 58

Number of combinational cells: 58

Number of sequential cells: 0

Number of macros: 0

Number of buf/inv: 17

Number of references: 3

Combinational area: 331.000000

Buf/Inv area: 51.000000

Noncombinational area: 0.000000

Net Interconnect area: undefined (No wire load specified)

Total cell area: 331.000000

Total area: undefined

beh2str – the simplest script!

```
#!/bin/tcsh
setenv SYNLOCAL /uusoc/facility/cad_common/local/class/6710/F13/synopsys
#set the path of dc shell script file
setenv SCRIPTFILE ${SYNLOCAL}/beh2str.tcl
#store the arguments
setenv INFILE $1
setenv OUTFILE $2
setenv LIBFILE $3
#setup to run synopsys
source /uusoc/facility/cad_common/local/setups/F13/setup-synopsys
# run (very simple) Synopsys Design Compiler synthesis
dc_shell-xg-t -f $SCRIPTFILE
```

Beh2str.tcl – the actual script

```
# beh2str script
set target_library [list [getenv "LIBFILE"]]
set link_library [concat [concat "" $target_library] $synthetic_library]
read_file -f verilog [getenv "INFILE"]
/* This command will fix the problem of having */
/* assign statements left in your structural file. */
set_fix_multiple_port_nets -all -buffer_constants
#do the actual compilation (synthesis)
compile -ungroup_all
check_design
/* always do change_names before write... */
redirect change_names { change_names -rules verilog -hierarchy -verbose }
# write out the structural Verilog
write -f verilog -output [getenv "OUTFILE"]
quit
```

What beh2str leaves out...

- ◆ Timing!
 - No clock defined so no target speed
 - No wire load model, so fewer placement constraints
 - No input drive defined so assume infinite drive
 - No output load define so assume something

Copy this from /uusoc/facility/cad_common/local/class/6710/F13/synopsys

.synopsys_dc.setup

```
...
set SynopsysInstall [getenv "SYNOPSYS"]

set search_path [list . \
[format "%s%s" $SynopsysInstall /libraries/syn] \
[format "%s%s" $SynopsysInstall /dw/sim_ver] \
]
define_design_lib WORK -path ./WORK
set synthetic_library [list dw_foundation.sldb]
set synlib_wait_for_design_license [list "DesignWare-Foundation"]
set link_library [concat [concat "*" $target_library] $synthetic_library]
set symbol_library [list generic.sdb]
...
```

syn-script.tcl

```
♦ /uusoc/facility/cad_common/local/class/6710/F13/synopsys

#/* search path should include directories with memory .db files */
#/* as well as the standard cells */
set search_path [list . \
[format "%s%s" SynopsysInstall /libraries/syn] \
[format "%s%s" SynopsysInstall /dw/sim_ver] \
!!your-library-path-goes-here!!]
#/* target library list should include all target .db files */
set target_library [list !!your-library-name!.db]
#/* synthetic_library is set in .synopsys_dc.setup to be */
#/* the dw_foundation library. */
set link_library [concat [concat "*" $target_library] $synthetic_library]
```

syn-script.tcl

```
#/* below are parameters that you will want to set for each design */
#/* list of all HDL files in the design */
set myFiles [list !!all-your-structural-Verilog-files!! ]
set fileFormat verilog           ;# verilog or VHDL
set basename !!basename!!       ;# Name of top-level module
set myClk !!clk!!                ;# The name of your clock
set virtual 0                    ;# 1 if virtual clock, 0 if real clock
#/* compiler switches... */
set useUltra 1                   ;# 1 for compile_ultra, 0 for compile
                                ;# mapEffort, useUngroup are for
                                ;# non-ultra compile...
set mapEffort1 medium            ;# First pass - low, medium, or high
set mapEffort2 medium            ;# second pass - low, medium, or high
set useUngroup 1                 ;# 0 if no flatten, 1 if flatten
```

syn-script.tcl

```
#!/* Timing and loading information */
set myPeriod_ns !!10!!           ;# desired clock period (sets speed goal)
set myInDelay_ns !!0.25!!       ;# delay from clock to inputs valid
set myOutDelay_ns !!0.25!!      ;# delay from clock to output valid
set myInputBuf !!INVX4!!        ;# name of cell driving the inputs
set myLoadLibrary !!Lib!!       ;# name of library the cell comes from
set myLoadPin !!A!!             ;# pin that outputs drive

#!/* Control the writing of result files */
set runname struct ;# Name appended to output files
```

syn-script.tcl

```
#!/* the following control which output files you want. They */
#!/* should be set to 1 if you want the file, 0 if not */
set write_v 1 ;# compiled structural Verilog file
set write_ddc 0 ;# compiled file in ddc format
set write_sdf 0 ;# sdf file for back-annotated timing sim
set write_sdc 1 ;# sdc constraint file for place and route
set write_rep 1 ;# report file from compilation
set write_pow 0 ;# report file for power estimate
```

syn-script.tcl

```
# analyze and elaborate the files
analyze -format $fileFormat -lib WORK $myfiles
elaborate $basename -lib WORK -update
current_design $basename
# The link command makes sure that all the required design
# parts are linked together.
# The uniquify command makes unique copies of replicated modules.
link
uniquify
# now you can create clocks for the design
if { $virtual == 0 } {
    create_clock -period $myPeriod_ns $myClk
} else {
    create_clock -period $myPeriod_ns -name $myClk
}
```

syn-script.tcl

```
# Set the driving cell for all inputs except the clock
# The clock has infinite drive by default. This is usually
# what you want for synthesis because you will use other
# tools (like SOC Encounter) to build the clock tree (or define it by hand).
set_driving_cell -library $myLoadLibrary -lib_cell $myInputBuf \
    [remove_from_collection [all_inputs] $myClk]
# set the input and output delay relative to myclk
set_input_delay $myInDelay_ns -clock $myClk \
    [remove_from_collection [all_inputs] $myClk]
set_output_delay $myOutDelay_ns -clock $myClk [all_outputs]
# set the load of the circuit outputs in terms of the load
# of the next cell that they will drive, also try to fix hold time issues
set_load [load_of [format "%s%s%s%s%s" $myLoadLibrary \
    "/" $myInputBuf "/" $myLoadPin]] [all_outputs]
set_fix_hold $myClk
```

syn-script.tcl

```
# now compile the design with given mapping effort
# and do a second compile with incremental mapping
# or use the compile_ultra meta-command
if { $useUltra == 1 } {
    compile_ultra
} else {
    if { $useUngroup == 1 } {
        compile -ungroup_all -map_effort $mapEffort1
        compile -incremental_mapping -map_effort $mapEffort2
    } else {
        compile -map_effort $mapEffort1
        compile -incremental_mapping -map_effort $mapEffort2
    }
}
```

syn-script.tcl

```
# Check things for errors
check_design
report_constraint -all_violators
set filebase [format "%s%s%s" $basename "_" $runname]
# structural (synthesized) file as verilog
if { $write_v == 1 } {
    set filename [format "%s%s" $filebase ".v"]
    redirect change_names { change_names -rules verilog \
                            -hierarchy -verbose }
    write -format verilog -hierarchy -output $filename
}
# write the rest of the desired files... then quit
```

Using Scripts

- ◆ Modify syn-script.tcl or write your own
- ◆ `syn-dc -f scriptname.tcl`
- ◆ Make sure to check output!!!!

addsub_struct.v – 10ns spec

Startpoint: addsub (input port clocked by clk)
Endpoint: result[8] (output port clocked by clk)
Path Group: clk
Path Type: max

Point	Incr	Path		
clock clk (rise edge)	0.00	0.00		
clock network delay (ideal)	0.00	0.00		
input external delay	0.25	0.25 f		
addsub (in)	0.00	0.25 f		
U79/Y (INVX4)	0.37	0.62 r		
U20/Y (NOR2X1)	0.64	1.26 f		
...				
U16/Y (NOR2X1)	0.33	8.30 r	Number of ports:	26
U17/Y (NOR2X1)	0.53	8.83 f	Number of nets:	111
result[8] (out)	0.00	8.83 f	Number of cells:	94
data arrival time		8.83	Number of combinational cells:	94
			Number of sequential cells:	0
			Number of macros:	0
clock clk (rise edge)	10.00	10.00	Number of buf/inv:	21
clock network delay (ideal)	0.00	10.00	Number of references:	7
output external delay	-0.25	9.75		
data required time		9.75		

data required time		9.75		
data arrival time		-8.83		

slack (MET)		0.92		

addsub_struct.v – 4ns spec

Startpoint: b[3] (input port clocked by clk)
 Endpoint: result[6] (output port clocked by clk)
 Path Group: clk
 Path Type: max

Point	Incr	Path		

clock clk (rise edge)	0.00	0.00		
clock network delay (ideal)	0.00	0.00		
input external delay	0.25	0.25 r		
b[3] (in)	0.00	0.25		
U16/Y (INVX4)	0.07	0.32 f		
U117/Y (NAND2X1)	0.41	0.73 r		
...				
U152/Y (NAND2X1)	0.24	3.74 r		
result[6] (out)	0.00	3.74 r		
data arrival time		3.74		

clock clk (rise edge)	4.00	4.00	Number of ports:	26
clock network delay (ideal)	0.00	4.00	Number of nets:	248
output external delay	-0.25	3.75	Number of cells:	231
data required time		3.75	Number of combinational cells:	231
			Number of sequential cells:	0

data required time		3.75	Number of macros:	0
data arrival time		-3.74	Number of buf/inv:	79
			Number of references:	9

slack (MET)		0.01		

addsub_struct.v – 3ns spec

Startpoint: a[1] (input port clocked by clk)
 Endpoint: result[7] (output port clocked by clk)
 Path Group: clk
 Path Type: max

Point	Incr	Path		

input external delay	0.25	0.25 r		
a[1] (in)	0.00	0.25 r		
U147/Y (INVX4)	0.07	0.32 f		
U80/Y (NAND2X1)	0.24	0.55 r		
...				
U132/Y (NAND2X1)	0.30	3.59 r		
result[7] (out)	0.00	3.59 r		
data arrival time		3.59		
clock clk (rise edge)	3.00	3.00	Number of ports:	26
clock network delay (ideal)	0.00	3.00	Number of nets:	247
output external delay	-0.25	2.75	Number of cells:	230
data required time		2.75	Number of combinational cells:	230
			Number of sequential cells:	0

data required time		2.75	Number of macros:	0
data arrival time		-3.59	Number of buf/inv:	77
			Number of references:	9

slack (VIOLATED)		-0.84		

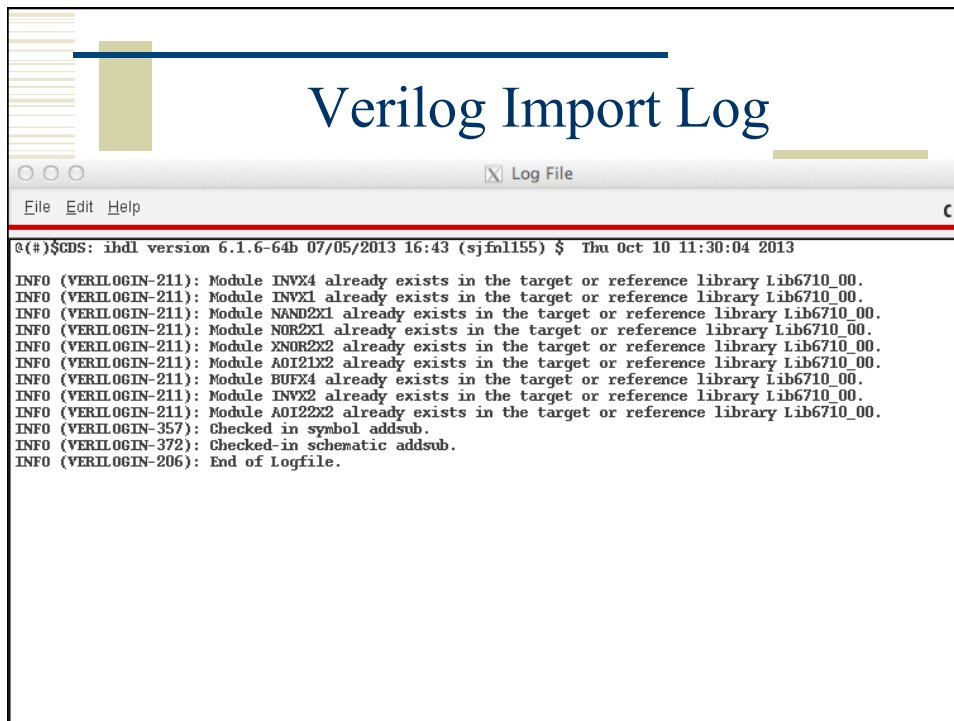
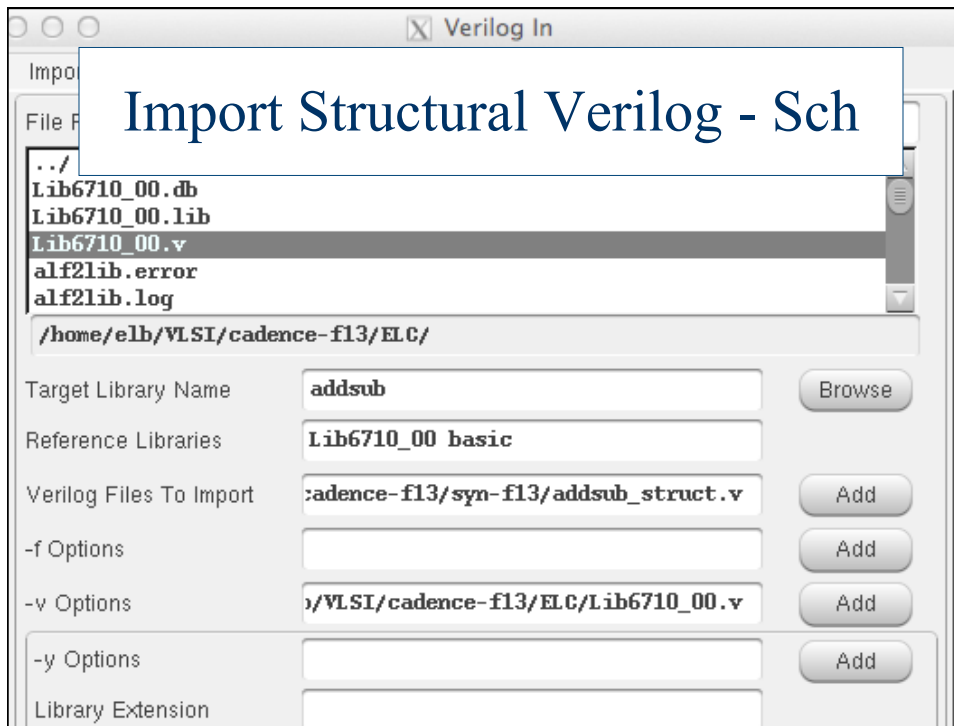
addsub_struct.v – 3ns spec

In the running log information...

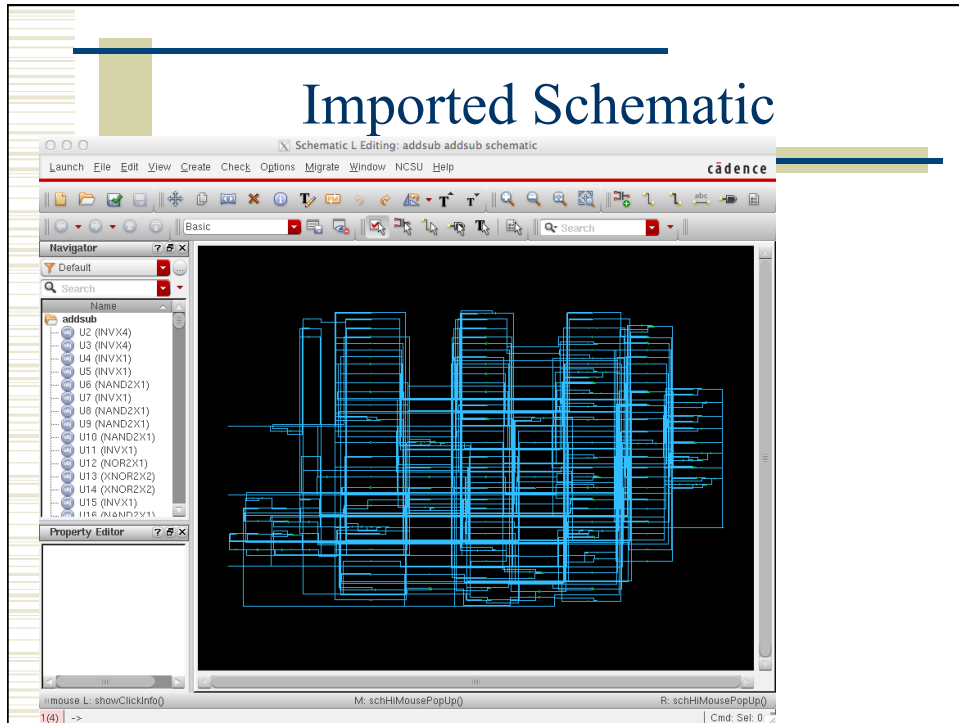
max_delay/setup ('clk' group)

Endpoint	Required Path Delay	Actual Path Delay	Slack
result[7]	2.75	3.59 x	-0.84 (VIOLATED)
result[6]	2.75	3.58 x	-0.83 (VIOLATED)
result[5]	2.75	3.56 x	-0.81 (VIOLATED)
result[3]	2.75	3.46 x	-0.71 (VIOLATED)
result[8]	2.75	3.40 x	-0.65 (VIOLATED)
result[4]	2.75	3.39 x	-0.64 (VIOLATED)
result[2]	2.75	3.29 x	-0.54 (VIOLATED)
result[1]	2.75	3.26 f	-0.51 (VIOLATED)

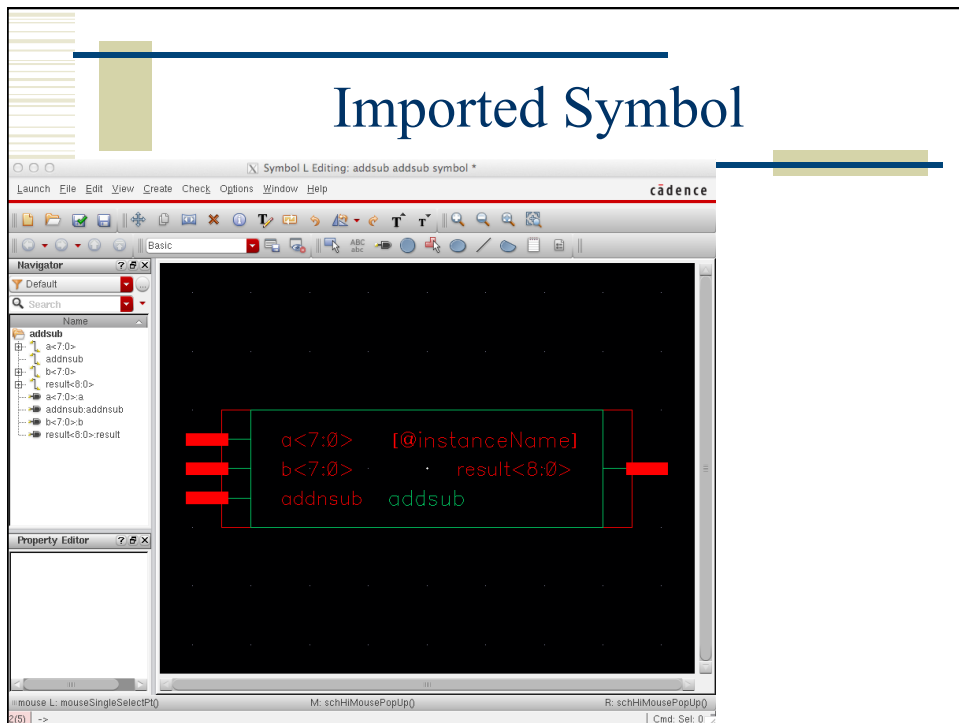
Import Structural Verilog - Sch



Imported Schematic



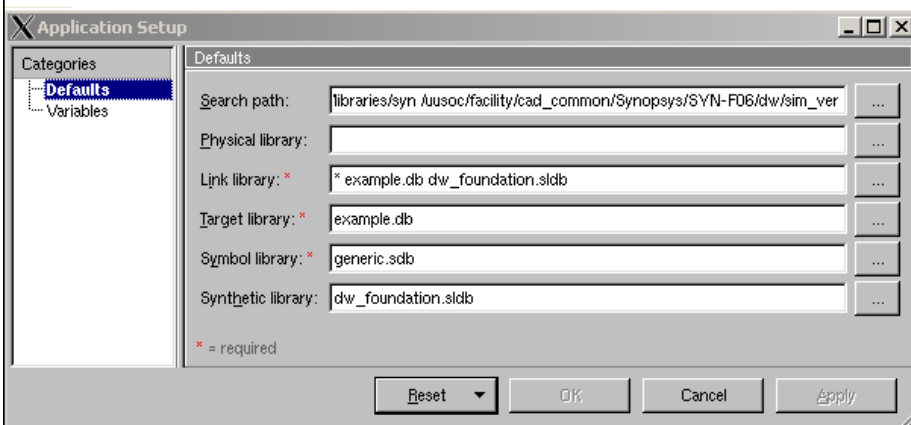
Imported Symbol



Using Design Vision

- ◆ You can do all of these commands from the design vision gui if you like
- ◆ **syn-dv**
- ◆ Follow the same steps as the script
 - Set libraries in your own **.synopsys_dc.setup**
 - analyze/elaborate
 - define clock and set constraints
 - compile
 - write out results

Setup



File ->Setup

analyze/elaborate

The image shows two overlapping dialog boxes in the Xilinx IDE. The top dialog is 'Analyze Designs', which has a list of file names in analysis order containing the path `/home/elb/IC_CAD/syn-f06/mips/mips.v`. Below the list are buttons for 'Add...', 'Delete', and 'Autom...'. At the bottom, there are fields for 'Format: Auto' and 'Work library: WORK', along with a checkbox for 'Create new library if it does not exist' and an 'OK' button. The bottom dialog is 'Elaborate Designs', which has a 'Library:' dropdown set to 'WORK' and a 'Design:' dropdown set to 'mips(verilog)'. It features a 'Parameters:' table with columns 'Name' and 'Value'. Below the table are checkboxes for 'Gate clock [command set_clock_gating_style must have been executed]' and 'Reanalyze out-of-date libraries', and 'OK' and 'Cancel' buttons at the bottom.

File -> Analyze

File ->Elaborate

Look at results...

The image shows a terminal window displaying the output of a compilation process. The output includes a table of resources, statistics for MUX_OPs, and several informational messages.

RAM_reg	Flip-flop	8	Y	N	N	N	N	N	N	N
RAM_reg	Flip-flop	8	Y	N	N	N	N	N	N	N
RAM_reg	Flip-flop	8	Y	N	N	N	N	N	N	N
RAM_reg	Flip-flop	8	Y	N	N	N	N	N	N	N
RAM_reg	Flip-flop	8	Y	N	N	N	N	N	N	N

```

Statistics for MUX_OPs
=====
| block name/line | Inputs | outputs | # sel inputs | MB |
=====
| regfile_WIDTH8_REGBITS3/304 | 8 | 8 | 3 | N |
| regfile_WIDTH8_REGBITS3/305 | 8 | 8 | 3 | N |
=====
Presto compilation completed successfully.
Information: Building the design 'alu' instantiated from design 'datapath_WIDTH8_REGBITS3' with
the parameters "8". (HDL-133)
statistics for case statements in always block at line 279 in file
'/home/elb/IC_CAD/syn-f06/mips/mips.v'
=====
| Line | full/ parallel |
=====
| 280 | auto/auto |
=====
Presto compilation completed successfully.
Information: Building the design 'zerodetect' instantiated from design 'datapath_WIDTH8_REGBITS3' with
the parameters "8". (HDL-133)
Presto compilation completed successfully.
design_vision-xg-t>
    
```

Define clock

attributes -> specify clock

Also look at other attributes...

Compile

Design -> Compile Ultra

Timing Reports

The screenshot shows a 'Report Timing' window with a table of timing data and a 'Report Timing Paths' dialog box.

Des/Clust/Port	Wire Load Model	Library
mips	5k	example
controller	5k	example

Point	Incr	Path
cont/state_reg[2]/0 (DFF_QB)	0.00	0.00 r
cont/state_reg[2]/0 (DFF_QB)	1.28	1.28 f
cont/V77/Y (NOR2)	0.51	1.80 r
cont/V76/Y (NAND2)	0.37	2.16 f
cont/V56/Y (NOR2)	1.08	3.24 r
cont/V54/Y (NOR2)	0.81	4.05 f
cont/V12/Y (INWX1)	0.50	4.56 r
cont/V35/Y (NOR2)	0.45	5.01 f
cont/V11/Y (INWX1)	0.33	5.33 r
cont/V34/Y (NOR2)	0.42	5.75 f
cont/V3/Y (INWX1)	0.48	6.24 r
cont/V28/Y (NOR2)	0.45	6.68 f
cont/V8/Y (INWX1)	0.24	6.93 r
cont/memread (controller)	0.00	6.93 r
memread (out)	0.00	6.93 r
data arrival time		6.93

(Path is unconstrained)

The 'Report Timing Paths' dialog shows configuration options for the report, including path type (Full), delay type (Max), and various reporting options like 'Report timing loops' and 'Justify paths with input vector'.

Timing -> Report Timing Path

Write Results

The screenshot shows a terminal window with the following text:

```

data arrival time                6.93
-----
(Path is unconstrained)
design_vision-xg-t>
Log History
design_vision-xg-t> change_names -rules verilog -hierarchy > change_names
ady
change_names
  
```

Below the terminal is a 'Save Design As' dialog box. The 'Look in' field shows the path `/home/rlb/IC_CAD/syn-106/`. The file list includes various Verilog files and databases. The 'File name' field contains `trythis.v`, and the 'File type' is set to `Database Files (*.ddc *.ddc.gz *.db *.db.gz *.gdb *.sdb *.pdb *.t)`. The 'Format' is set to `Auto`. There is a `SYNOPTIS` logo in the bottom right corner of the dialog.

File -> Save As...

Endpoint slack...

The screenshot displays the 'Endpoint Slack' tool interface. On the left, there are configuration options for delay type (set to 'max'), binning settings (8 bins), and histogram settings (title: 'Endpoint Slack', x-axis: 'Slack', y-axis: 'Number of Paths'). The central histogram shows the distribution of path slack values, with bars labeled with their respective counts: 8, 2, 30, 40, 15, 37, 1, 16. The x-axis ranges from -1.78141 (Worst) to 8.9648 (Best). On the right, a table lists paths and their slack values:

Slack	Name
-1.78141	dp_pcreg_q_reg_1_/D
-1.78141	dp_pcreg_q_reg_3_/D
-1.78141	dp_pcreg_q_reg_5_/D
-1.78141	dp_pcreg_q_reg_7_/D
-1.781	dp_pcreg_q_reg_0_/D
-1.781	dp_pcreg_q_reg_2_/D
-1.781	dp_pcreg_q_reg_4_/D
-1.781	dp_pcreg_q_reg_6_/D

Timing -> Endpoint Slack

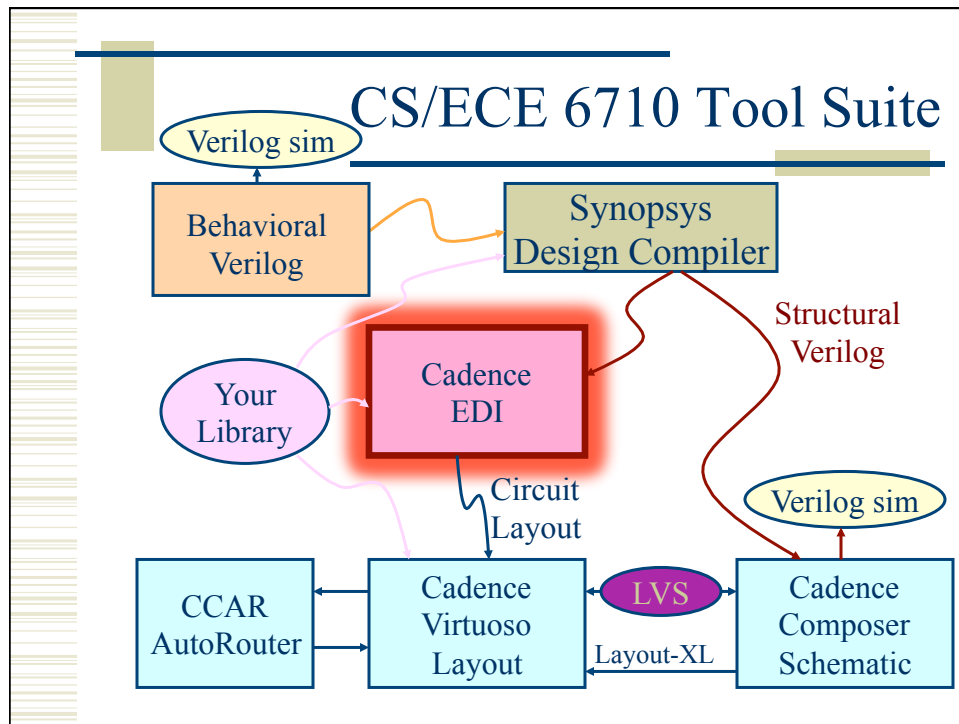
Path Slack

Timing -> Path Slack

The screenshot displays the 'Path Slack' tool interface. On the left, there are configuration options for path selection (From, Through, To pins), number of paths (10 worst, 50 max), and binning settings (8 bins). The central histogram shows the distribution of path slack values, with bars labeled with their respective counts: 8, 8, 8, 20, 6. The x-axis ranges from -1.78141 (Worst) to -1.77778 (Best). On the right, a table lists paths and their slack values:

Slack	From	To
-1.78141	cont_state_reg_3_/G	dp_pcreg_q_reg_1_/D
-1.78141	cont_state_reg_3_/G	dp_pcreg_q_reg_3_/D
-1.78141	cont_state_reg_3_/G	dp_pcreg_q_reg_5_/D
-1.78141	cont_state_reg_3_/G	dp_pcreg_q_reg_7_/D
-1.781	cont_state_reg_3_/G	dp_pcreg_q_reg_0_/D
-1.781	cont_state_reg_3_/G	dp_pcreg_q_reg_2_/D
-1.781	cont_state_reg_3_/G	dp_pcreg_q_reg_4_/D
-1.781	cont_state_reg_3_/G	dp_pcreg_q_reg_6_/D

A circuit diagram is visible in the background on the right side of the interface.



Encounter Digital Implementation (EDI)

1. Import Design
2. Floorplan
3. Power plan
4. Place cells
5. Synthesize clock tree
6. Route signal nets
7. Verify results
8. Write out results

Convert structural Verilog
(From Synopsys)
Into physical layout

Sometimes shorthand:
Place and Route

EDI Usage

- ◆ Need structural Verilog, `struct.sdc`, `library.lib`, `library.lef`
- ◆ Make a new dir for edi... (I call mine EDI)
- ◆ Make an `mmmc.tcl` file with timing/library info
- ◆ `<design>.globals` has design-specific settings
 - use `UofU_edi.globals` as starting point.
- ◆ Usual warnings about scripting...
 - `Top.tcl` and other `*.tcl` are in the class directory as starting points
 - `./uusoc/facility/cad_common/local/class/6710/F13/cadence/EDI`
- ◆ Call with `cad-edi`

cad-edi Flow

1. Import Design
 - `.v`, `.sdc`, `.lib`, `.lef` – can put these in a `<name>.globals` and `mmmc.tcl`
 - `mmmc` = multi-mode multi-corner
2. Floorplan
 - Choose physical size, ratio, utilization percentage, etc.
3. Power plan
 - rings, stripes, row-routing (`sroute`)

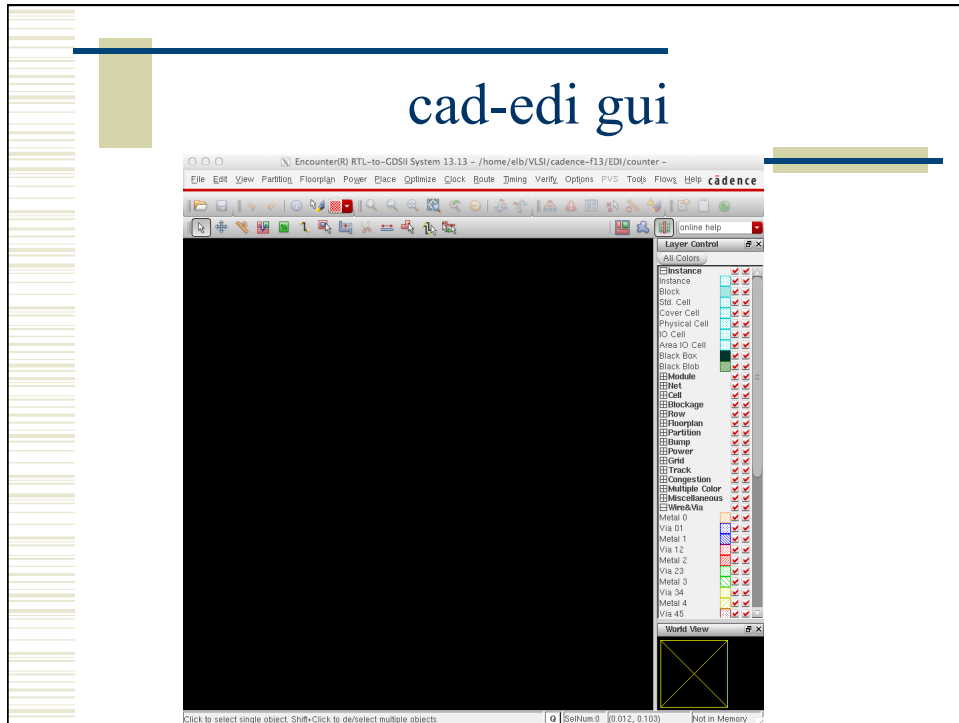
cad-edi Flow

4. Placement
 - place cells in the rows
 - Timing optimization – preCTS
5. Synthesize clock tree
 - use your buf or inv footprint cells
 - timing optimization – postCTS
7. global routing
 - NanoRoute
 - timing optimization – postRoute

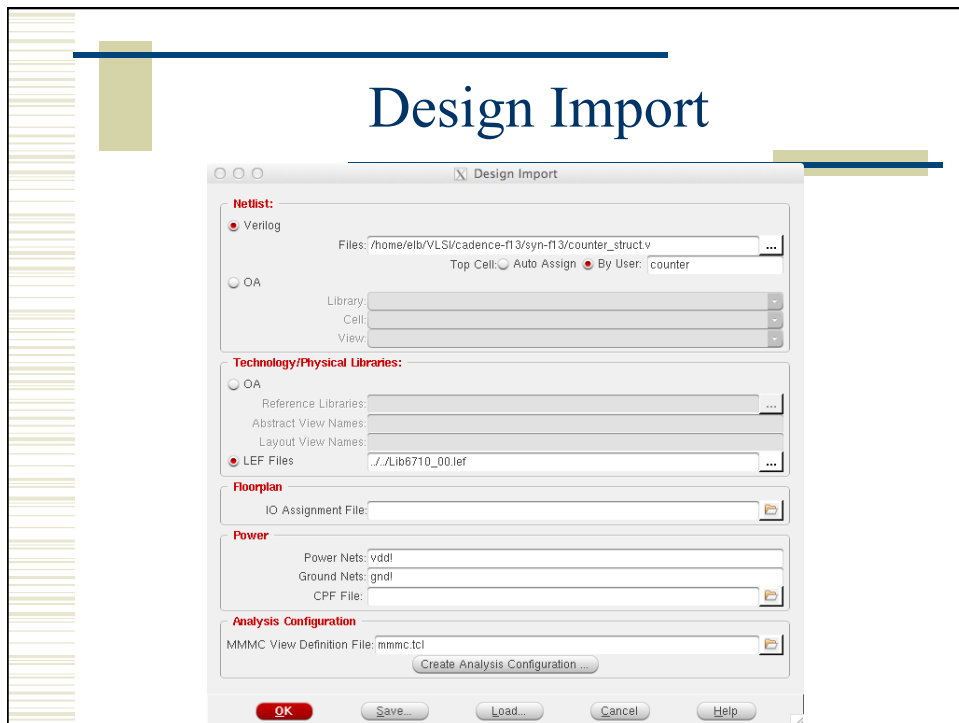
cad-edi Flow

8. Add filler cells
 - Fill in the spots in the row with no cells
 - Adds NWELL for continuity
10. Write out results
 - <name>.def can be imported as layout
 - <name>_edi.v is the placed and routed Verilog description
 - .spef, .sdc, _edi.lib have timing information

cad-edi gui



Design Import



Using a .globals file

- ◆ Put the load information into a .globals file
- ◆ Load it up without having to re-type
- ◆ Also need a mmmc.tcl file

UofU_edi.globals

```
#
# Set the name of your structural Verilog file
# This comes from Synopsys synthesis
set init_verilog {!!your-file-name.v!!}
# Set the name of your top module
set init_design {!!your-top-module-name!!}
# Set the name of your .lef file
# This comes from ELC
set init_lef_file {!!your-file-name.lef!!}
...
```

UofU_edi.globals

```
#####  
# below here you probably don't have to change anything  
#####  
# Set the name of your "multi-mode-multi-corner data file  
# You don't need to change this unless you're using a  
# different mmmc.tcl file.  
set init_mmmc_file {mmmc.tcl}  
# Some helpful input mode settings  
set init_import_mode {-treatUndefinedCellAsBbox 0 -keepEmptyModule 1 }  
# Set the names of your gnd and power nets  
set init_gnd_net {gnd!}  
set init_pwr_net {vdd!}
```

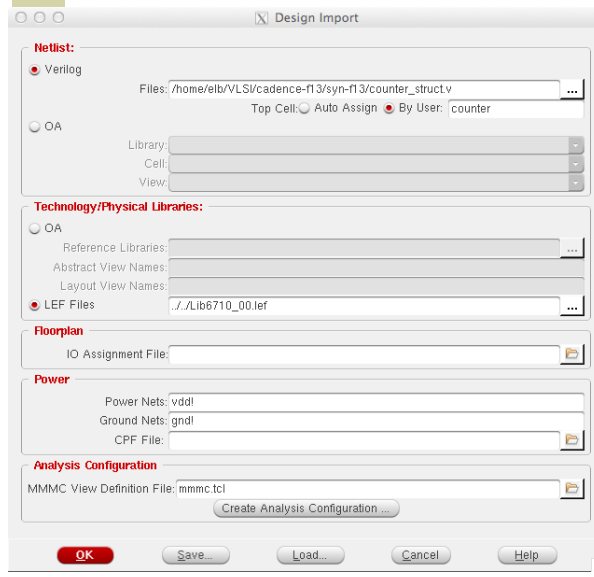
mmmc.tcl

```
# set the name of your .lib file (e.g. Lib6710_01.lib)  
# You can create multiple library sets if you have multiple libraries  
# such as fast, slow, and typ  
# If you have multiple .lib files put them in a [list lib1 lib2] structure  
create_library_set -name typical_lib \  
    -timing {!!your-lib-file!!.lib}  
# Specify the .sdc timing constraint file to use  
# This file comes from Synopsys synthesis. (e.g. design_struct.sdc)  
create_constraint_mode -name typical_constraint \  
    -sdc_files {!!your-sdc-file!!.sdc}  
...
```

mmmc.tcl

```
#####  
# Below here you shouldn't have to change, unless you're doing  
# something different than the basic EDI run...  
#####  
# Create an RC_corner that has specific capacitance info.  
create_rc_corner -name typical_rc\  
...  
# Define delay corners and analysis views.  
create_delay_corner -name typical_corner \  
-library_set {typical_lib} \  
-rc_corner {typical_rc}  
create_analysis_view -name typical_view \  
-constraint_mode {typical_constraint} \  
-delay_corner {typical_corner}  
# Now define which analysis view to use for setup and for hold.  
set_analysis_view -setup {typical_view} -hold {typical_view}
```

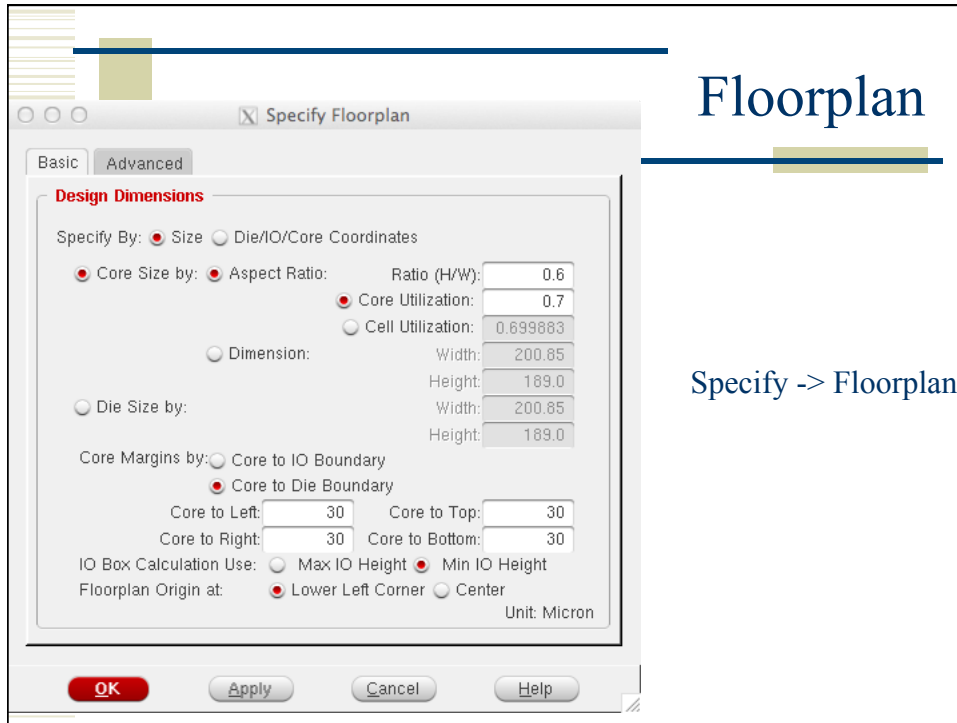
Design Import



Some screen captures are from a slightly older version of EDI...

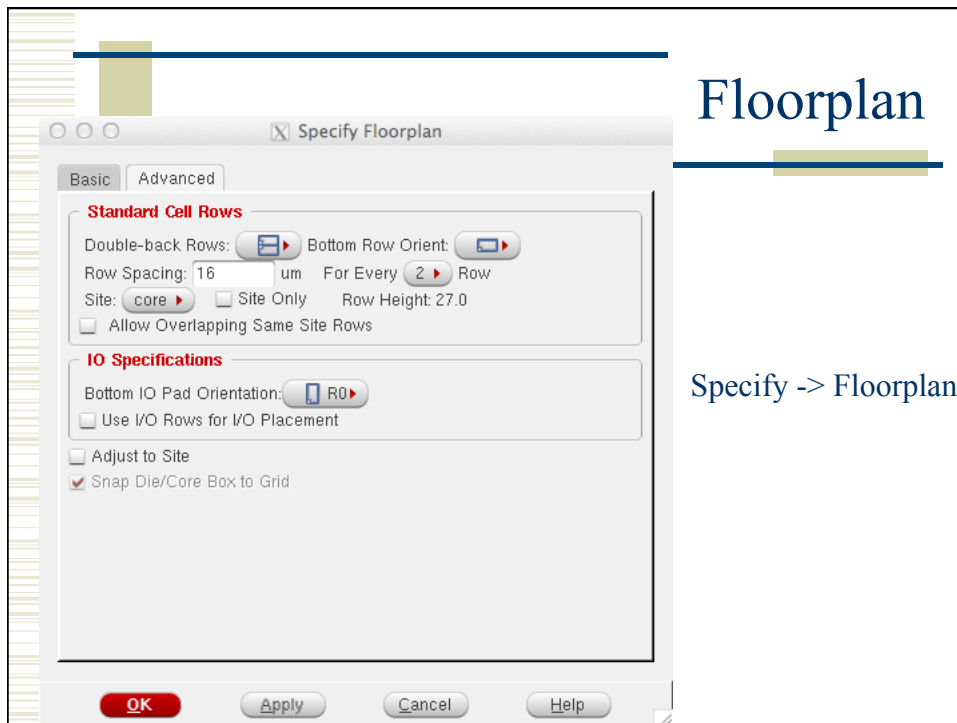
not this one...

Floorplan

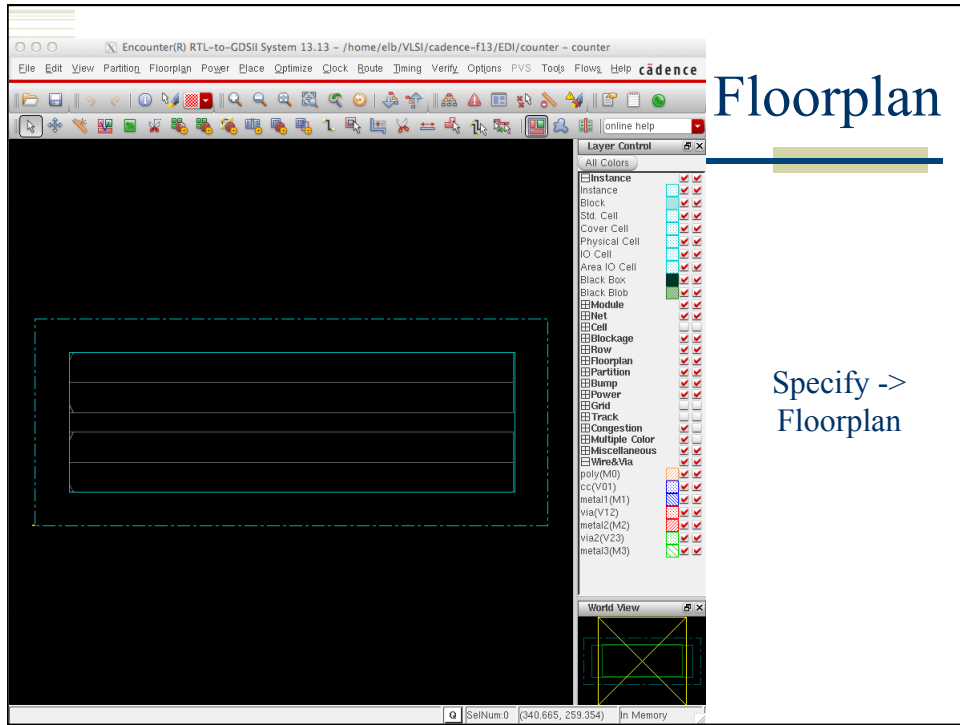


Specify -> Floorplan

Floorplan

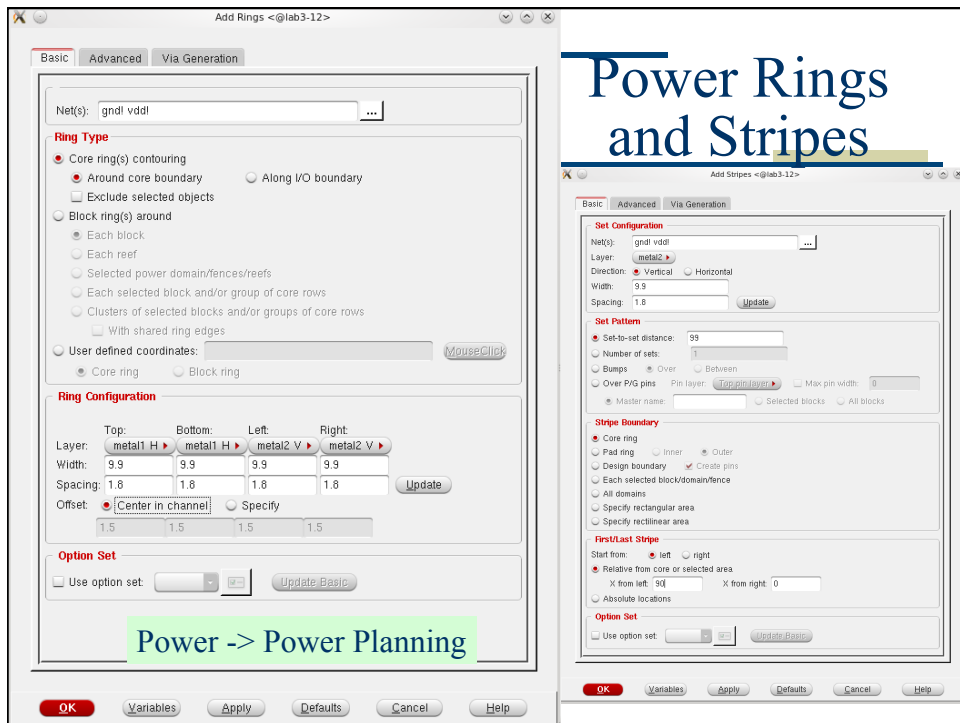


Specify -> Floorplan



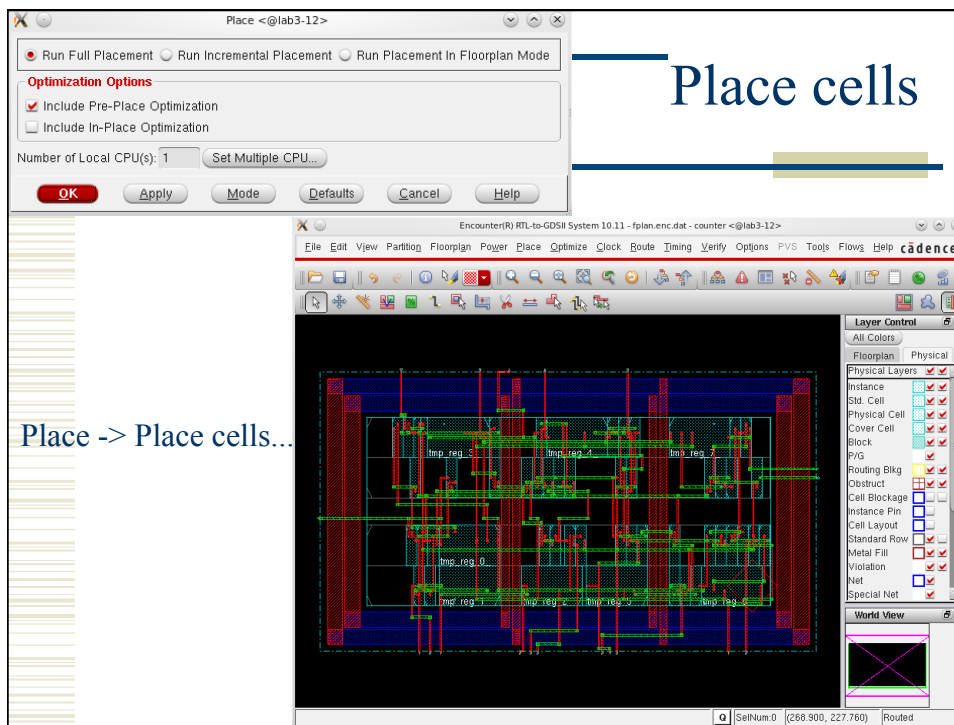
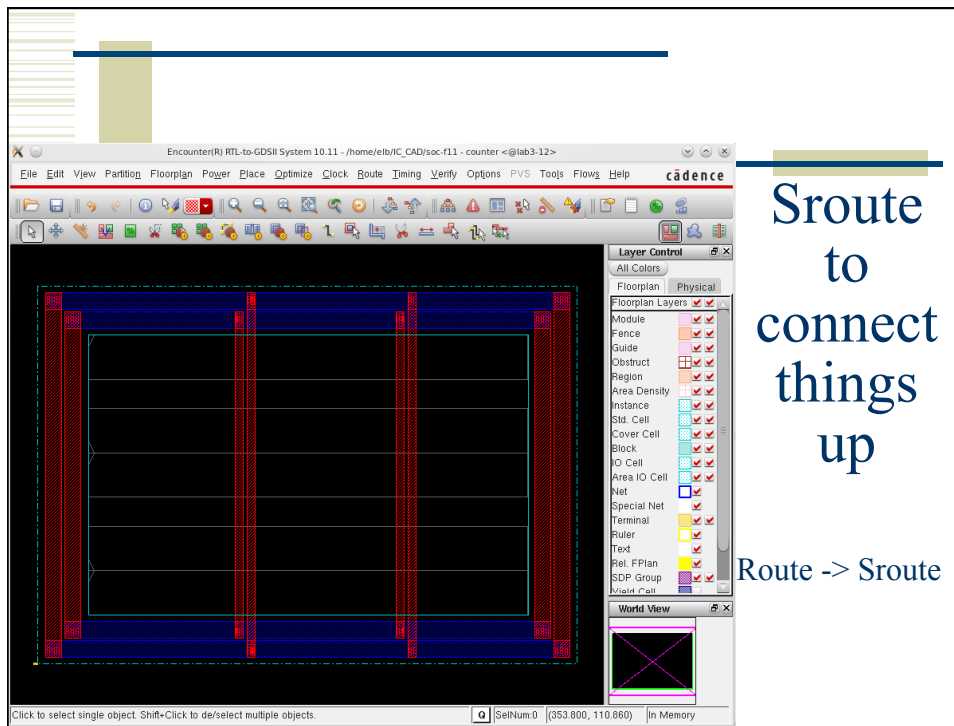
Floorplan

Specify ->
Floorplan



Power Rings and Stripes

Power -> Power Planning



pre-CTS timing optimization

Timing -> Optimization

Optimization <@lab3-12>

Design Stage

Pre-CTS Post-CTS Post-Route

Optimization Type

Setup Hold

Design Rules Violations

Max Cap

Max Tran

Max Fanout

Include SI

optDesign Final Summary

Setup mode	all	reg2reg	in2reg	reg2out	in2o
WNS (ns):	-2,063	-2,063	3,986	3,811	N/
TNS (ns):	-16,867	-16,867	0,000	0,000	N/
Violating Paths:	12	12	0	0	N/
All Paths:	149	123	43	18	N/

Clock Tree Synthesis

clock -> create clock tree spec

Generate Clock Spec <@lab3-12>

Specify Buffer/Inverter

Cells List

IN VX1

Selected Cells

IN VX1

Output Specification File: Clock.ctstch

Synthesize Clock Tree <@lab3-12>

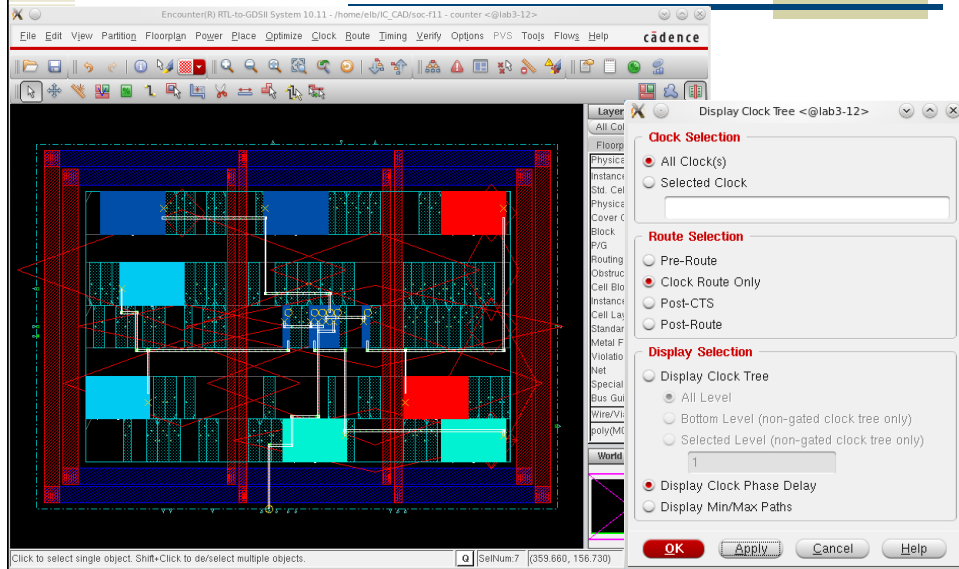
Basic | Advanced

Clock Specification Files: Clock.ctstch

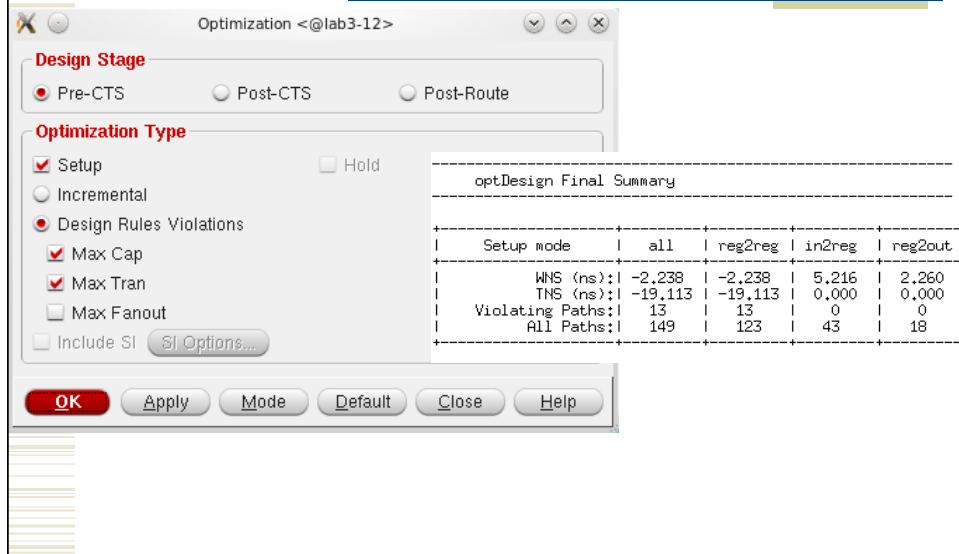
Results Directory: clock_report

clock -> Synthesize clock tree

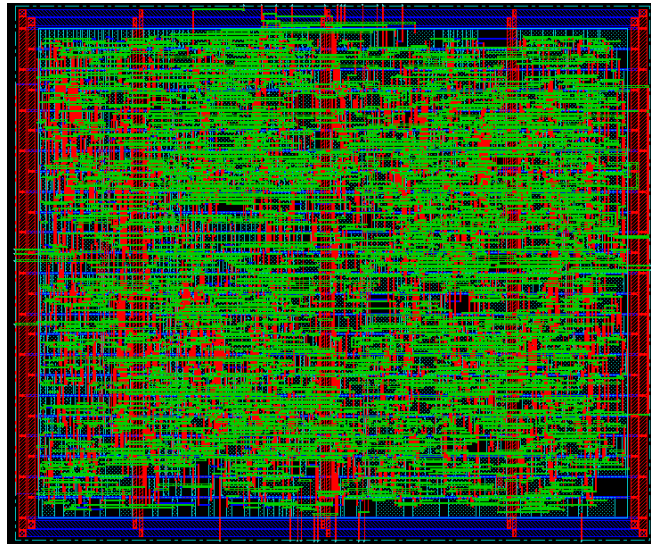
Display Clock Tree



post-CTS optimization



Routed circuit



postRoute optimization

Timing -> Optimization

Optimization <@lab3-12>

Design Stage

Pre-CTS Post-CTS Post-Route

Optimization Type

Setup Hold

Incremental

Design Rules Violations

Max Cap

Max Tran

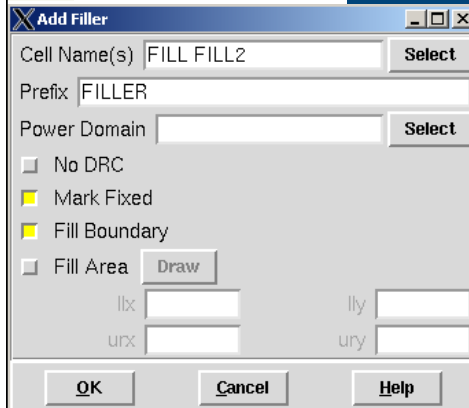
Max Fanout

Include SI

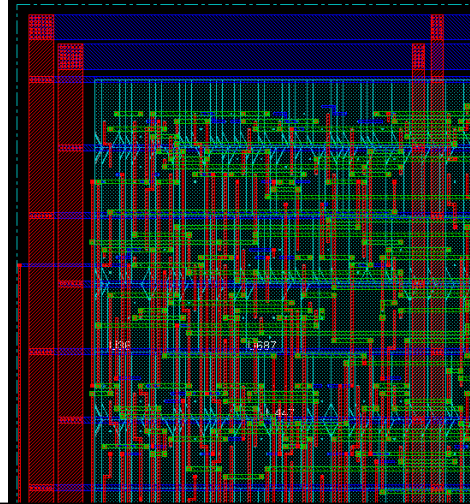
optDesign Final Summary

Setup mode	all	reg2reg	in2reg	reg2out
WNS (ns):	-3,080	-3,080	5,251	1,827
TNS (ns):	-32,195	-32,195	0,000	0,000
Violating Paths:	26	26	0	0
All Paths:	149	123	43	18

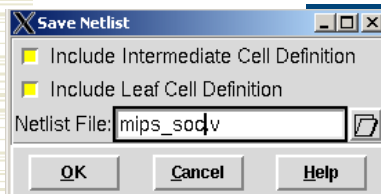
Add Filler



Place -> Filler -> Add...

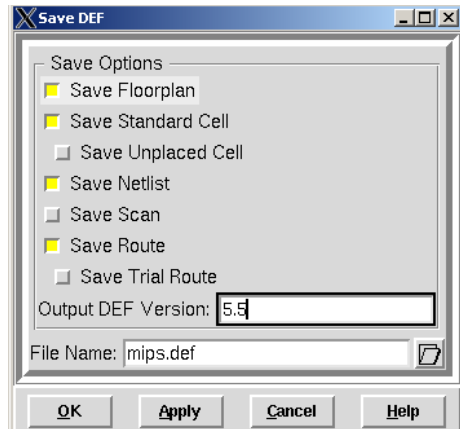


Write Results...



Design -> Save -> Netlist

Design -> Save -> DEF



Encounter Scripting

- ◆ Usual warnings – know what’s going on!
- ◆ Use **top.tcl** as a starting point
 - And the other **.tcl** files it calls...
- ◆ EDI has a floorplanning stage that you may want to do by hand
 - **write another script to read in the floorplan and go from there...**
- ◆ Use **encounter.cmd** to see the text versions of what you did in the GUI...

top.tcl

```
# set the basename for the config and floorplan files. This
# will also be used for the .lib, .lef, .v, and .spef files...
set basename "counter"
# The following variables are used in fplan.tcl...
# Note that rowgap and coregap should be divisible by the basic
# grid unit of 0.3 that our AMIC5N/F process uses.
#
set usepct 0.60 ;# percent utilization in placing cells
set rowgap 15 ;# gap (microns) between pairs of std cell rows
set aspect 0.60 ;# aspect ratio of overall cell (1 is square,
                ;# <1 is landscape, >1 is portrait)
set coregap 30.0 ;# gap (microns) between the core and the power rails
```

top.tcl

```
#####  
# You may not have to change things below this line - but check!  
#  
# You may want to do floorplanning by hand in which case you  
# have some modification to do!  
#####  
  
# Set some of the power and stripe parameters - you can change  
# these if you like - in particular check the stripe space (sspace)  
# and stripe offset (soffset)!  
set pwidth 9.9 ;# power rail width  
set pspace 1.8 ;# power rail space  
set swidth 4.8 ;# power stripe width  
set sspace 123 ;# power stripe spacing  
set soffset 120 ;# power stripe offset to first stripe
```

top.tcl

```
#  
# Set the flag for EDIto automatically figure out buf, inv, etc.  
set dbgGPSAutoCellFunction 1  
  
# Import design and floorplan  
# If the config file is not named $basename.globals, edit this line.  
source $BASENAME.globals  
init_design
```

top.tcl

```
# source the files that operate on the circuit
source fplan.tcl ;# create the floorplan (might be done by hand...)
source pplan.tcl ;# create the power rings and stripes
source place.tcl ;# Place the cells and optimize (pre-CTS)
source cts.tcl ;# Create the clock tree, and optimize (post-CTS)
source route.tcl ;# Route the design using nanoRoute
source verify.tcl ;# Verify the design and produce output files
exit
```

fplan.tcl

```
puts "-----Floorplanning-----"
#
# Make a floorplan - this works fine for projects that are all
# standard cells and include no blocks that need hand placement...
setDrawView fplan
setFPlanRowSpacingAndType $rowgap 2
floorPlan -site core -r $aspect $usepct \
          $coregap $coregap $coregap $coregap
fit

#
# Save design so far
saveDesign ${BASENAME}_fplan.enc
saveFPlan ${BASENAME}.fp
puts "-----Floorplanning done-----"
```

pplan.tcl

```
puts "-----Power Planning-----"
puts "-----Making power rings-----"
#
# Make power and ground rings - $pwidth microns wide
# with $pspace spacing between them and centered in the channel
addRing -spacing_bottom $pspace \
    -width_left $pwidth \
    -width_bottom $pwidth \
    -width_top $pwidth \
    -spacing_top $pspace \
    -layer_bottom metal1 \
    -center 1 \
    -stacked_via_top_layer metal3 \
    ...
```

pplan.tcl

```
puts "-----making power stripes-----"
# Make Power Stripes. This step is optional. If you keep it
# in remember to check the stripe spacing
# (set-to-set-distance = $sspace) and stripe offset
# (xleft-offset = $soffset)
addStripe -block_ring_top_layer_limit metal3 \
    -max_same_layer_jog_length 3.0 \
    -snap_wire_center_to_grid Grid \
    -padcore_ring_bottom_layer_limit metal1 \
    ...
# Use the special-router to route the vdd! and gnd! nets
route -allowJogging 1

# Save the design so far
saveDesign ${BASENAME}_pplan.enc
puts "-----Power Planning done-----"
```

top.tcl

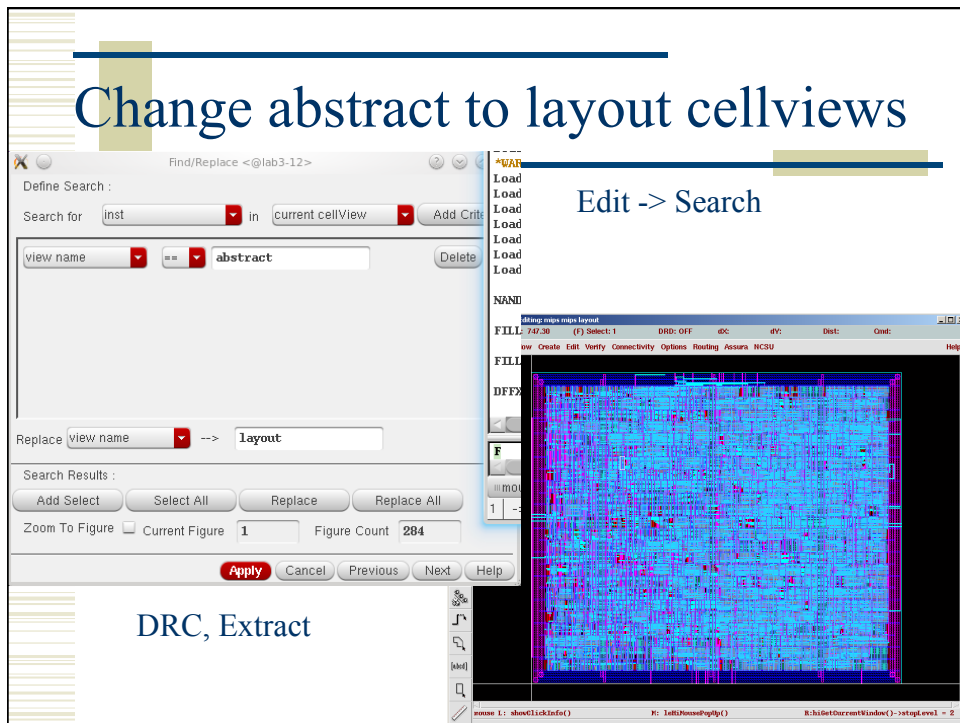
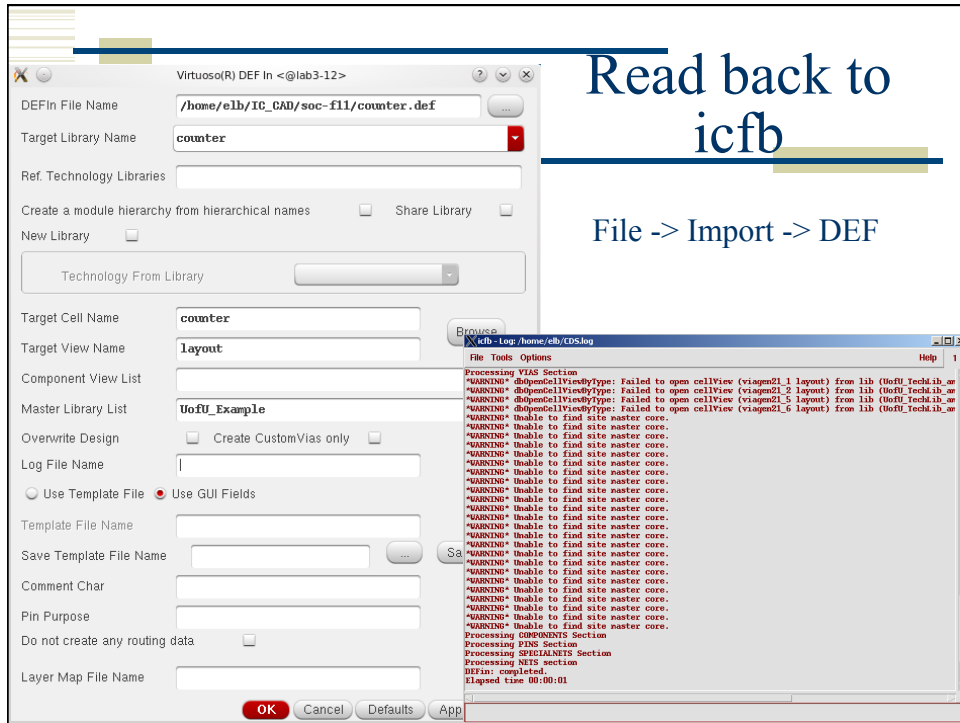
Read the script...

- place
- pre-CTS optimization
- clock tree synthesis
- post-CTS optimization
- routing
- post-ROUTE optimization
- add filler
- write out results

Report Files

- ◆ <topname>_Conn_regular.rpt
- ◆ <topname>_Conn_special.rpt
- ◆ <topname>_Geom.rpt

- ◆ Want 0 violations in all
 - If you have 1 or 2 in the geometry you might be able to fix them easily in Virtuoso...



Import Verilog

The screenshot shows the 'Verilog In' dialog box with the following fields:

- File Filter Name: [Empty]
- Target Library Name: **mips**
- Reference Libraries: **example basic**
- Verilog Files To Import: **../soc/mips/mips_soc.v**
- f Options: [Empty]
- v Options: **example.v**
- y Options: [Empty]
- Library Extension: [Empty]

Buttons: OK, Cancel, Defaults, Apply, Load, Save, Help

Log File window content:

```
@(#)GDS: ihdl.exe version 5.1.0 07/23/2006 23:42 (cicln01) $ Thu  
module MUX2_INV already in target/reference library example  
module INVX4 already in target/reference library example  
module INVX1 already in target/reference library example  
Checked in symbol mux2_WIDTH8_0  
Checked in schematic mux2_WIDTH8_0  
module DFF_QB already in target/reference library example  
module TIEH1 already in target/reference library example  
Checked in symbol flop_WIDTH8_2  
Checked in schematic flop_WIDTH8_2  
Checked in symbol mux2_WIDTH8_2  
Checked in schematic mux2_WIDTH8_2  
Checked in symbol flop_WIDTH8_3  
Checked in schematic flop_WIDTH8_3  
module INVX3 already in target/reference library example  
module NOR2 already in target/reference library example  
module NAND2 already in target/reference library example  
module DFF already in target/reference library example  
module XOR2 already in target/reference library example  
Checked in symbol mips  
Checked in schematic mips  
End of Logfile.
```

File -> Import -> Verilog
*Make SURE you import
The Verilog from EDI!*

LVS...

Schematic view

The screenshot shows the 'Verilog Schematic Editor' window with a complex circuit diagram. The diagram consists of numerous interconnected logic gates and components, rendered in blue lines on a black background. The window title is 'Verilog Schematic Editor: mips mips schematic'. The menu bar includes: Tools, Design, Window, Edit, Add, Check, Sheet, Options, Migrate, HCSU, Help. The status bar at the bottom shows: mouse L: showClickInfo() mouse R: schMousePopUp() mouse M: setExtForm().

```

/home/elb/IC_CAD/cadence-f06/LVS/si.out
File
4508      pmos
4508      mmos

Net-list summary for /home/elb/IC_CAD/cadence-f06/LVS/schematic/netList
count
4919      nets
30        terminals
4372      pmos
4372      mmos

Terminal correspondence points
N2452    N1137    adr<0>
N1980    N660     adr<1>
N2887    N345     adr<2>
N2149    N873     adr<3>
N1695    N401     adr<4>
N604     N603     adr<5>
N1066    N1024    adr<6>
N1900    N601     adr<7>
N566     N567     clk
N3328    N832     memdata<0>
N1598    N304     memdata<1>
N2069    N783     memdata<2>
N721     N695     memdata<3>
N165     N147     memdata<4>
N1814    N508     memdata<5>
N3974    N234     memdata<6>
N2485    N1167    memdata<7>
N1109    N1068    memread
N575     N575     memwrite
N4147    N379     reset
N4234    N497     writedata<0>
N3536    N1009    writedata<1>
N581     N584     writedata<2>
N3871    N132     writedata<3>
N4824    N1115    writedata<4>
N4527    N806     writedata<5>
N2534    N1218    writedata<6>
N2926    N394     writedata<7>

Devices in the rules but not in the netlist:
cap nfet pfet mmos4 pmos4

The net-lists match.

```

LVS Result

Yay!

Summary

- ◆ Behavioral -> Structural -> Layout
- ◆ Can be automated by scripting, but make sure you know what you're doing
 - on-line tutorials for TCL
 - Google "tcl tutorial"
 - Synopsys documentation for *design_compiler*
 - *encounter.cmd* (and documentation) for EDI
- ◆ End up with placed and routed core layout
 - or BLOCK for later use...