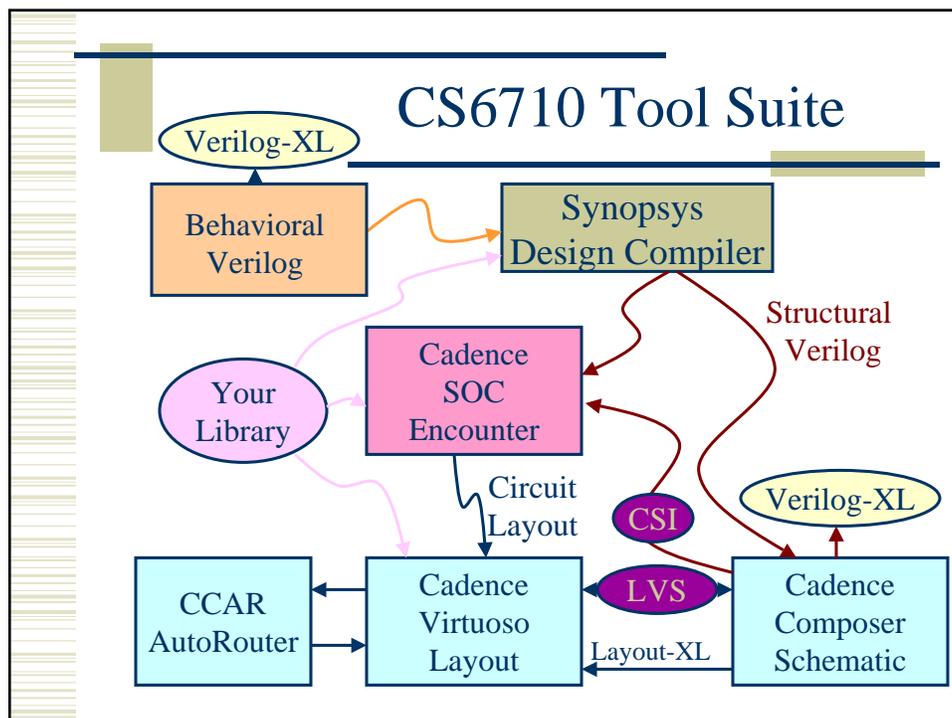


Synthesis and Place & Route

Synopsys design compiler
Cadence SOC Encounter



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...

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!

```
# 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
compile -ungroup_all
check_design
#/* always do change_names before write... */
redirect change_names { change_names -rules verilog -hierarchy -verbose }
write -f verilog -output [getenv "OUTFILE"]
quit
```

.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]
```

What beh2str leaves out...

- ◆ Timing!
 - No clock defined so no target speed
 - No input drive defined so assume infinite drive
 - No output load define so assume something

syn-script.tcl

```
♦ /uusoc/facility/cad_common/local/class/6710/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-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 !!0.5!!    ;# delay from clock to inputs valid
set myoutdelay_ns !!0.5!!   ;# delay from clock to output valid
set myinputbuf !!invX4!!    ;# name of cell driving the inputs
set myloadcell !!UofU_Digital/invX4/A!! ;# pin that outputs drive
set mylibrary !!UofU_Digital!! ;# name of library the cell comes from

#!/* 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_db 0     ;# compiled file in db format (obsolete)
set write_ddc 0    ;# compiled file in ddc format (XG-mode)
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 $mylibrary -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 $myloadcell] [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" [format "%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
```

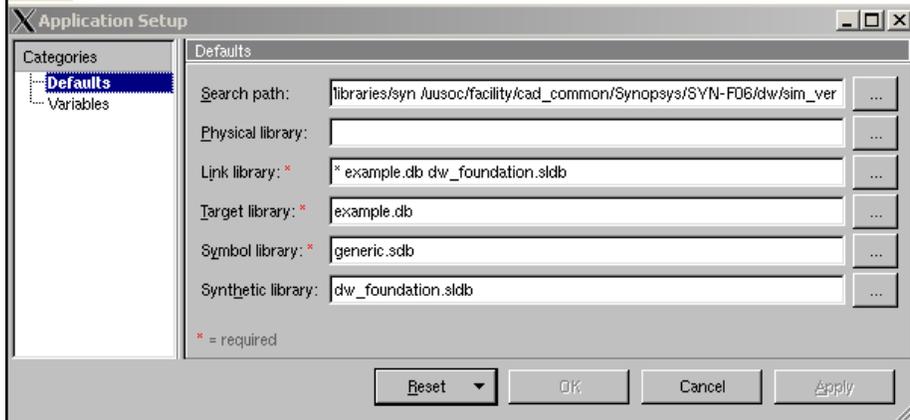
Using Scripts

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

Using Design Vision

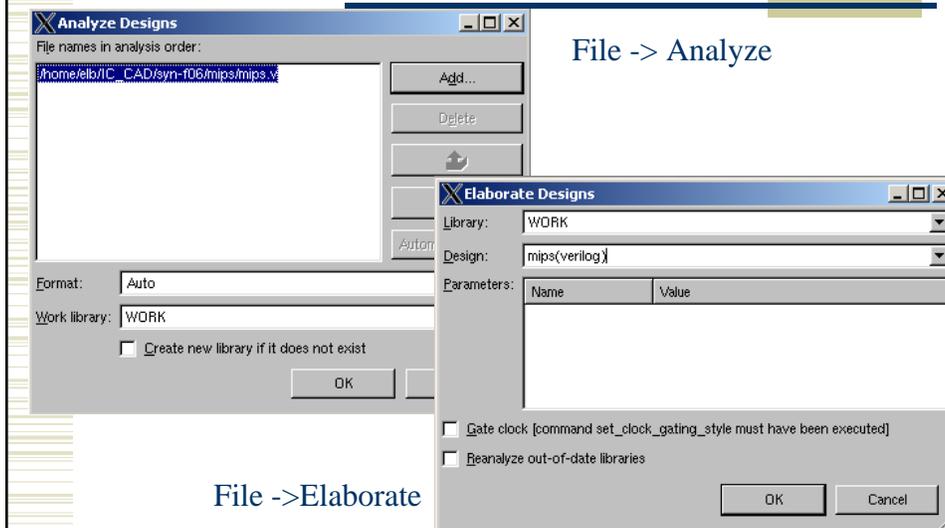
- ◆ You can do all of these commands from the design vision gui if you like
- ◆ `syn-dv`
- ◆ Follow the same steps as the script
 - Set libraries in your own `.synopsys_dc.setup`
 - analyze/elaborate
 - define clock and set constraints
 - compile
 - write out results

Setup



File ->Setup

analyze/elaborate



File -> Analyze

File ->Elaborate

Look at results...

```

RAM_reg | Flip-flop | 8 | 2 | N | N | N | N | N | N |
RAM_reg | Flip-flop | 8 | 2 | N | N | N | N | N | N |
RAM_reg | Flip-flop | 8 | 2 | N | N | N | N | N | N |
RAM_reg | Flip-flop | 8 | 2 | N | N | N | N | N | N |
RAM_reg | Flip-flop | 8 | 2 | N | N | N | N | N | N |
-----
Statistics for MUX_Ops
-----
| block name/line | Inputs | Outputs | # sel inputs | MB |
-----
| regfile_WIDTH0_REGBITS3/304 | 8 | 8 | 3 | N |
| regfile_WIDTH0_REGBITS3/305 | 8 | 8 | 3 | N |
-----
Presto compilation completed successfully.
Information: Building the design 'alu' instantiated from design 'datapath_WIDTH0_REGBITS3' with
the parameters "8". \(DDL-193\)
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_WIDTH0_REGBITS3' with
the parameters "8". \(DDL-193\)
Presto compilation completed successfully.
design_vision>ng-t>

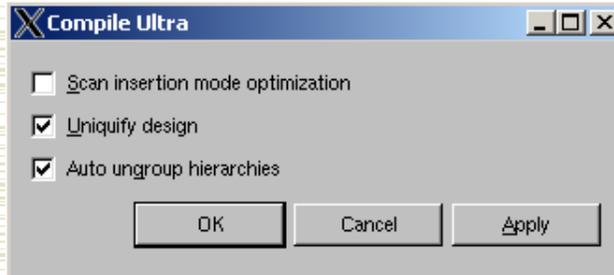
```

Define clock

attributes -> specify clock

Also look at other attributes...

Compile



Design -> Compile Ultra

Timing Reports

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

Point	Incr	Path
ocont/state_reg[2]/0 (DFF_QE)	0.00	0.00 r
ocont/state_reg[2]/0 (DFF_QE)	1.28	1.28 f
ocont/V77/Y (NOR2)	0.51	1.80 r
ocont/V76/Y (NOR2)	0.37	2.16 f
ocont/V56/Y (NOR2)	1.08	3.24 r
ocont/V54/Y (NOR2)	0.81	4.05 f
ocont/V12/Y (INWX1)	0.50	4.56 r
ocont/V35/Y (NOR2)	0.45	5.01 f
ocont/V11/Y (INWX1)	0.33	5.33 r
ocont/V34/Y (NOR2)	0.42	5.75 f
ocont/V9/Y (INWX1)	0.48	6.24 r
ocont/V28/Y (NOR2)	0.45	6.68 f
ocont/V8/Y (INWX1)	0.24	6.93 r
ocont/memread (controller)	0.00	6.93 r
memread (out)	0.00	6.93 r
data arrival time		6.93

(Path is unconstrained)

Report Timing Paths

From: [in] Through: [in] To: [in]

Report options

Max paths per endpoint: [1] Minimum path delay: []

Max paths per group: [] Minimum path delay: []

Path type: [full] True path reporting

Delay type: [max] Report timing loops

Significant digits: [2] Justify paths with input vector

Sort by: [group] Find true path

No fan split Enable asynchronous arcs

Show nets in combinational path Show net transition time

Show input pins in combinational path Show net capacitance

Show don't_touch, vgl_only attributes for nets and cells

Output options

To report viewer

To file [report.txt]

Append to file

OK Cancel Apply

Timing -> Report Timing Path

Write Results

data arrival time 6.93

(Path is unconstrained)
design_vision-xg-t>

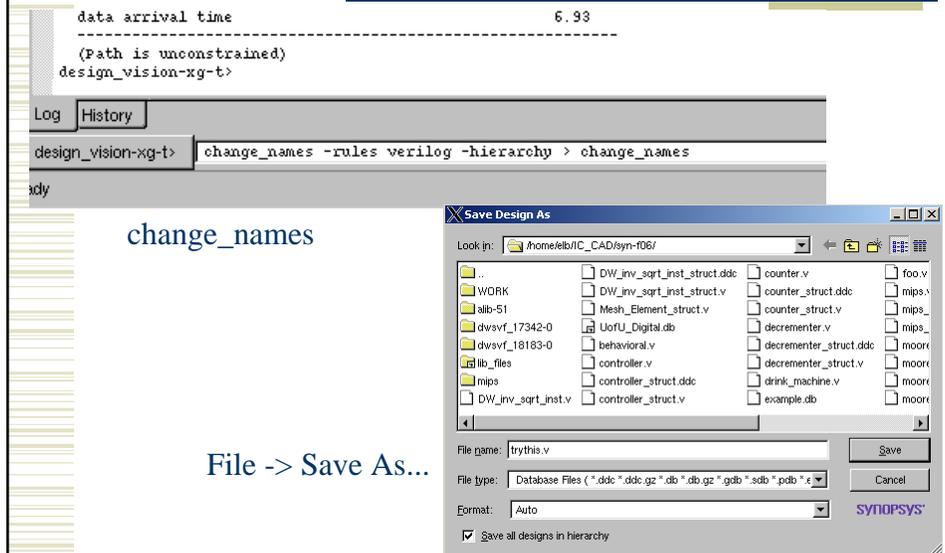
Log History

```
design_vision-xg-t> change_names -rules verilog -hierarchy > change_names
```

cdly

change_names

File -> Save As...

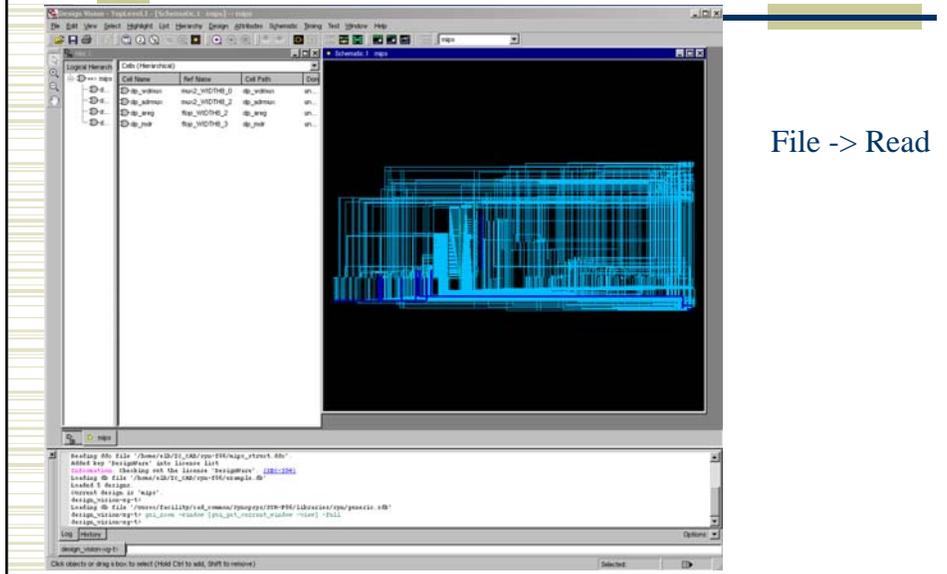


The screenshot shows a Synopsys IDE interface. At the top, a terminal window displays the command 'change_names -rules verilog -hierarchy > change_names' and the output 'data arrival time 6.93'. Below the terminal, there are buttons for 'Log' and 'History'. A 'Save Design As' dialog box is open, showing a file list with 'trythis.v' selected. The dialog box has a 'File name' field containing 'trythis.v', a 'File type' dropdown set to 'Database Files (*.ddc *.ddc.gz *.db *.db.gz *.gdb *.sdc *.sdc.gz *.pdc *.pdc.gz)', and a 'Format' dropdown set to 'Auto'. The 'Save' button is highlighted.

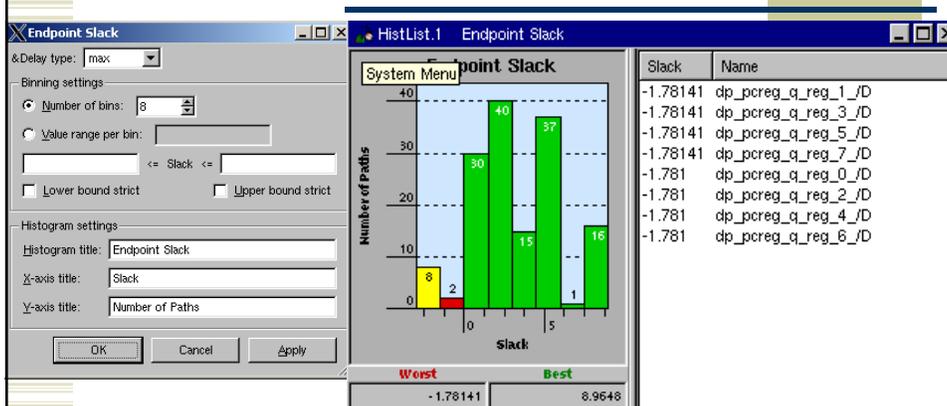
Or, use syn-dv after script...

- ◆ `syn-dc -f mips.tcl`
- ◆ results in .v, .ddc, .sdc, .rep files
- ◆ Read the .ddc file into syn-dv and use it to explore timing...

syn-dv with mips_struct.v



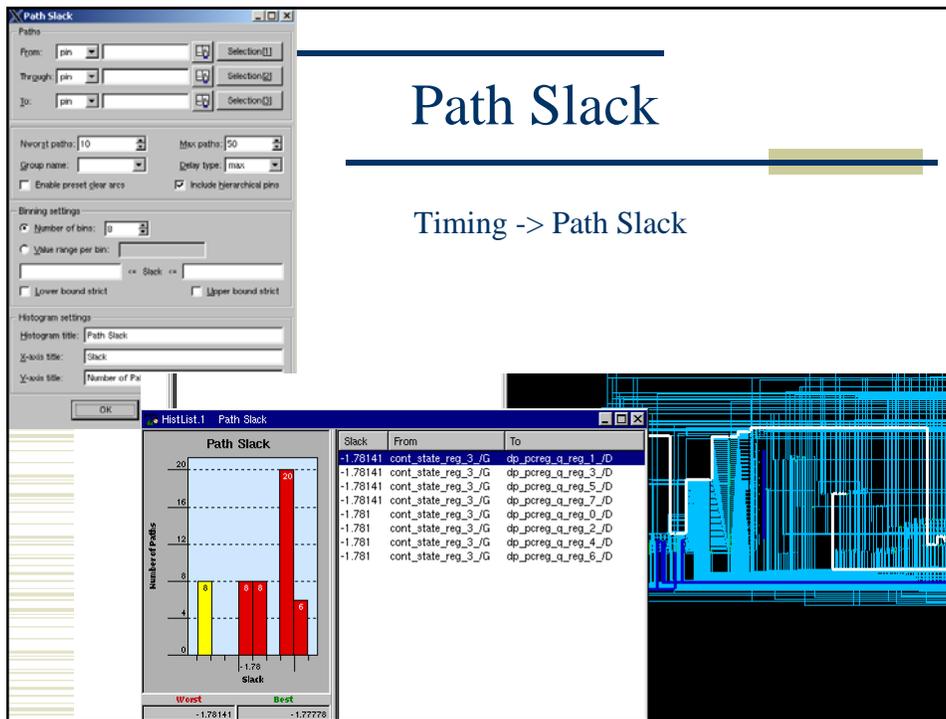
Endpoint slack...



Timing -> Endpoint Slack

Path Slack

Timing -> Path Slack



SOC Encounter

- ◆ Need structural Verilog, .sdc, library.lib, library.lef
- ◆ make a new dir for soc...
- ◆ <design>.conf is also very helpful
 - use UofU_soc.conf as starting point.
- ◆ Usual warnings about scripting...
UofU_opt.tcl is the generic script
 - .../local/class/6710/cadence/SOC
- ◆ cad-soc

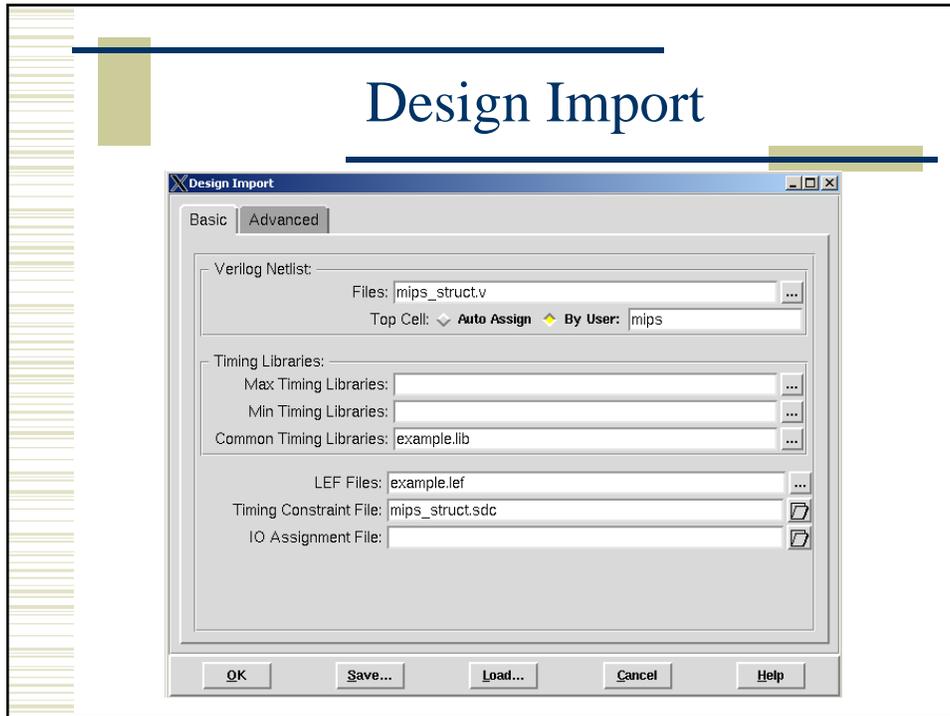
SOC Flow

1. Import Design
 - .v, .sdc, .lib, .lef – can put this in a .conf
2. Power plan
 - rings, stripes, row-routing (sroute)
3. Placement
 - place cells in the rows
4. Timing optimization – preCTS

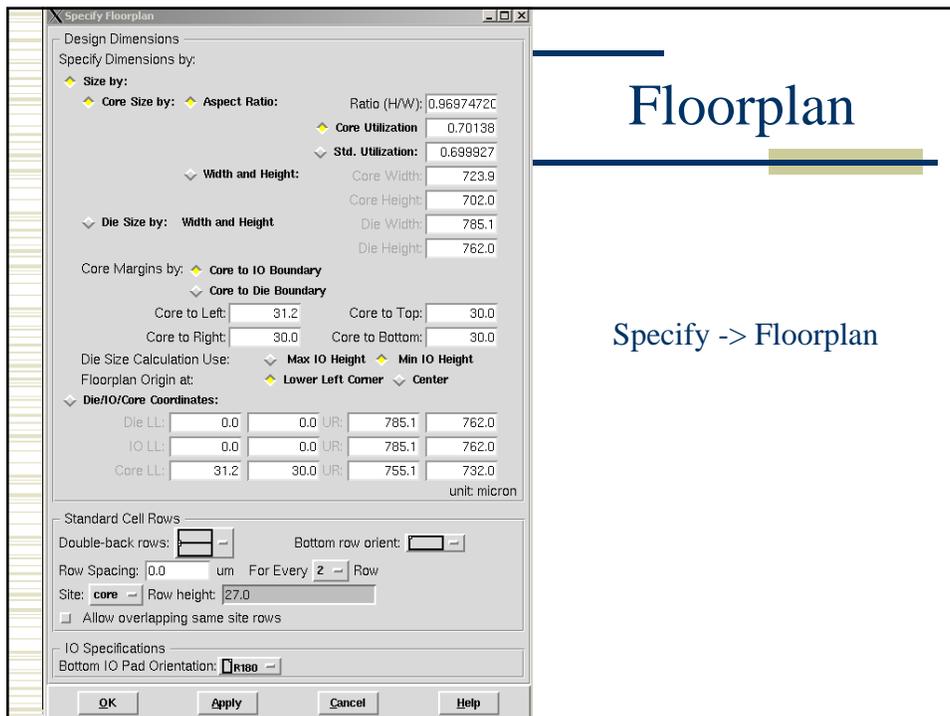
SOC Flow

5. Synthesize clock tree
 - use your buf or inv footprint cells
6. timing optimization – postCTS
7. global routing
 - NanoRoute
8. timing optimization – postRoute
9. Add filler cells
10. Write out results
 - .def, _soc.v, .spef, .sdc, .lef

Design Import



Floorplan



Specify -> Floorplan

Place cells

Place -> Place cells...

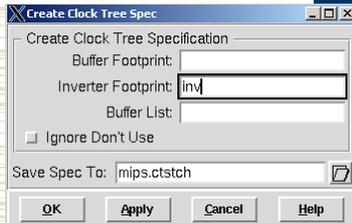
pre-CTS timing optimization

Timing -> Optimization

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/

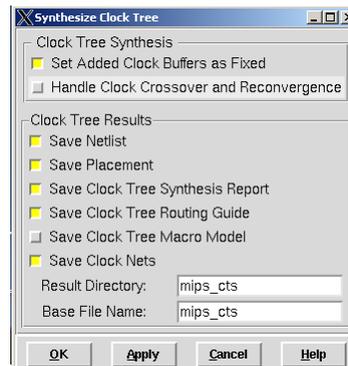
Clock Tree Synthesis



clock -> create clock tree spec

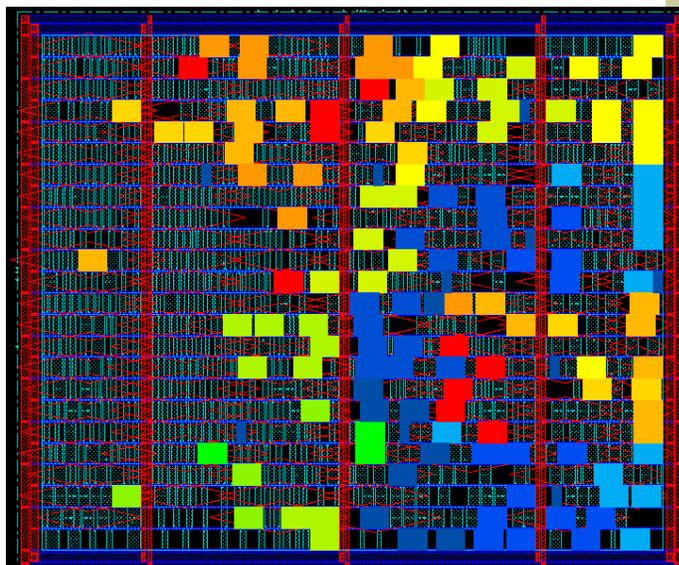


clock -> specify clock tree



clock -> Synthesize clock tree

Display Clock Tree



post-CTS optimization

The Optimization dialog box is shown with the 'Basic' tab selected. The 'Design Stage' is set to 'Post-CTS', 'Effort' is 'High', and 'Optimization Type' includes 'Setup', 'Design Rule Violations', and 'Path Groups'.

optDesign Final Summary

Setup mode	all	reg2reg	in2reg	reg2out
WNS (ns):	-2,238	-2,238	5,216	2,260
TNS (ns):	-19,113	-19,113	0,000	0,000
Violating Paths:	13	13	0	0
All Paths:	149	123	43	18

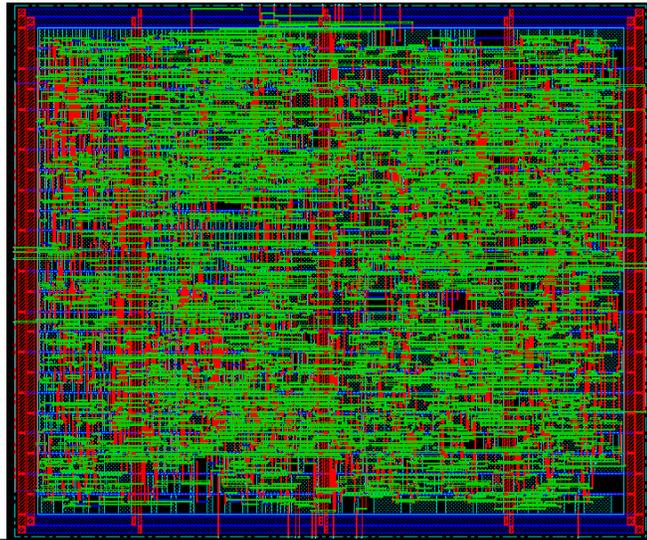
NanoRoute

The NanoRoute dialog box is shown with the following settings:

- Mode:** Global Route, Detail Route (Start Iteration: default, End Iteration: default)
- Concurrent Routing Features:** Fix Antenna, Insert Diodes (Diode Cell Name:), Timing Driven (Effort: 9, Congestion: , Timing: S.M.A.R.T.), SI Driven, Post Route SI (SI Victim File:)
- Routing Control:** Selected Nets Only (Bottom Layer: default, Top Layer: default), ECO Route, Area Route (Area: , Select Area and Route)
- Job Control:** Auto Stop, Multi Threading (CPUs: 1), Superthreading (Clients: , Edit Clients)

Route -> NanoRoute -> Route

Routed circuit



postRoute optimization

Optimization dialog box settings:

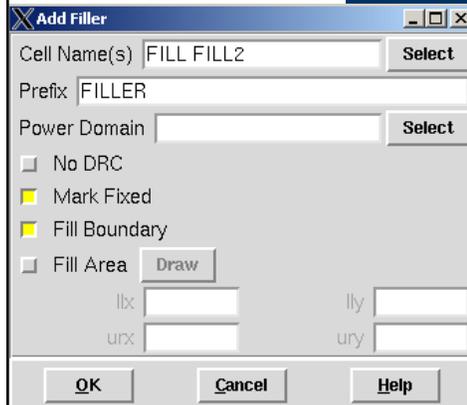
- Design Stage: **Post-Route**
- Effort: **High**
- Optimization Type:
 - Setup
 - Incremental
 - Hold
 - Design Rule Violations
 - Max Cap
 - Max Tran
 - Max Fanout
 - Include SI
- Path Groups: **All Path Groups**

Timing -> Optimization

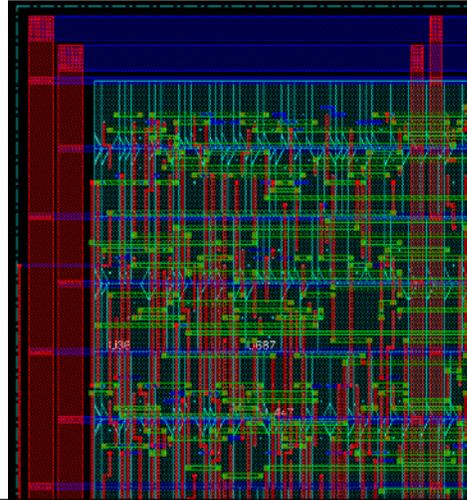
optDesign Final Summary

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

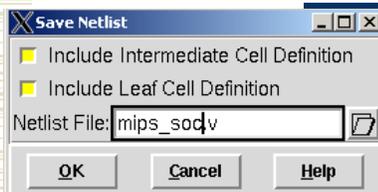
Add Filler



Place -> Filler -> Add...

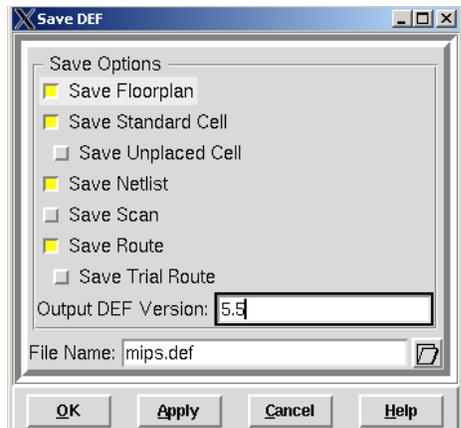


Write Results...



Design -> Save -> Netlist

Design -> Save -> DEF



Encounter Scripting

- ◆ Usual warnings – know what’s going on!
- ◆ Use `UofU_opt.tcl` as a starting point
- ◆ SOC 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...

UofU_opt.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 "mips"

# set the name of the footprint of the clock buffers
# in your .lib file
set clockBufName inv

# set the name of the filler cells - you don't need a list
# if you only have one
set fillerCells FILL
#set fillerCells [list FILL FILL2]
```

UofU_opt.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  
set pspace 1.8  
set swidth 4.8  
set sspace 249  
set soffset 126
```

UofU_opt.tcl

```
# Import design and floorplan  
# If the config file is not named $basename.conf, edit this line.  
loadConfig $basename.conf 0  
commitConfig  
# Make a floorplan - this works fine for projects that are all  
# standard cells and include no blocks that need hand placement...  
setDrawMode fplan  
floorPlan -site core -r 1.0 0.70 30.0 30.0 30.0 30.0  
fit  
# Save design so far  
saveDesign "fplan.enc"  
saveFPlan [format "%s.fp" $basename]
```

UofU_opt.tcl

```
# 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 -width_right $pwidth -
    around_core -jog_distance $pspace -offset_bottom $pspace -layer_top
    metal1 -threshold $pspace -offset_left $pspace -spacing_right $pspace -
    spacing_left $pspace -offset_right $pspace -offset_top $pspace -
    layer_right metal2 -nets {gnd! vdd! } -stacked_via_bottom_layer metal1 -
    layer_left metal2
```

UofU_opt.tcl

```
# 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
    -set_to_set_distance $sspace -stacked_via_top_layer metal3
    -padcore_ring_top_layer_limit metal3 -spacing $pspace -xleft_offset $soffset
    -merge_stripes_value 1.5 -layer metal2 -block_ring_bottom_layer_limit metal1
    -width $pwidth -nets {gnd! vdd! } -stacked_via_bottom_layer metal1
#
# Use the special-router to route the vdd! and gnd! nets
sroute -jogControl { preferWithChanges differentLayer }
#
# Save the design so far
saveDesign "pplan.enc"
```

UofU_opt.tcl

Read the script...

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

Read back to icfb

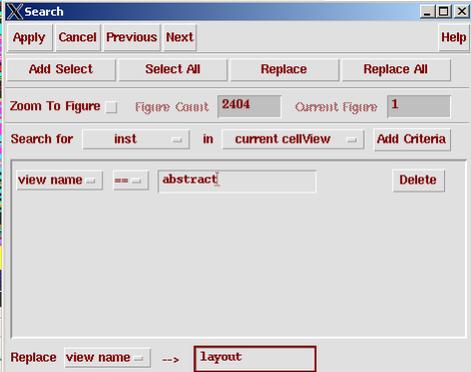
The screenshot shows the 'Virtuosos(R) DEF In' dialog box with the following fields filled in:

- Library Name: mips
- Cell Name: mips
- View Name: layout
- Use Ref. Library Names example
- DEF File Name: ../soc/mips/mips.def
- Map Names From: VERILOG
- Startup Name Mapping:
- Target P&R Engine: Gate Ensemble Silicon Ensemble

To the right of the dialog box, the text 'File -> Import -> DEF' is displayed. Below the dialog box, a terminal window shows the output of a script, including several warnings and a final 'DEFIn: completed' message.

Change abstract to layout cellviews

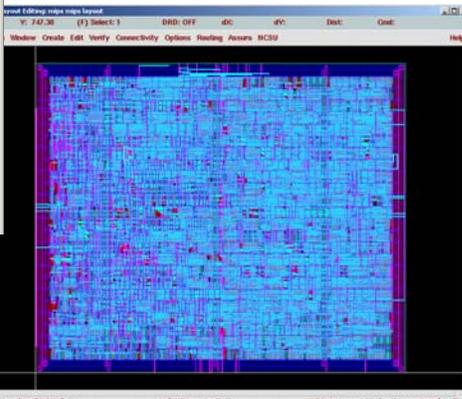
Search Edit -> Search



view name abstract

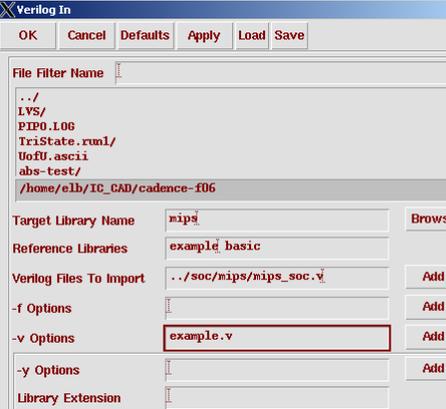
Replace view name --> layout

DRC, Extract



Import Verilog

Verilog In File -> Import -> Verilog



File Filter Name

Target Library Name mips

Reference Libraries example_basic

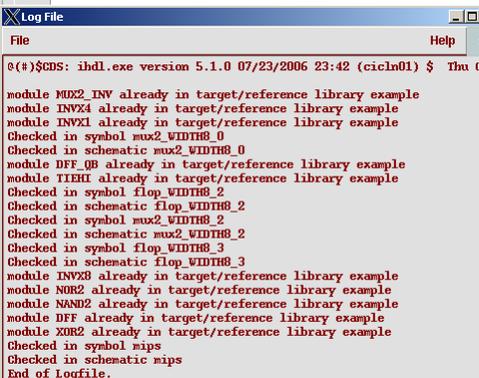
Verilog Files To Import ../soc/mips/mips_soc.v

-f Options

-v Options example.v

-y Options

Library Extension



Log File

```
@(#)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 INVX8 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.
```

LVS...

Schematic view



```
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
N3388 N632 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 SOC
- ◆ End up with placed and routed core layout
 - or BLOCK for later use...