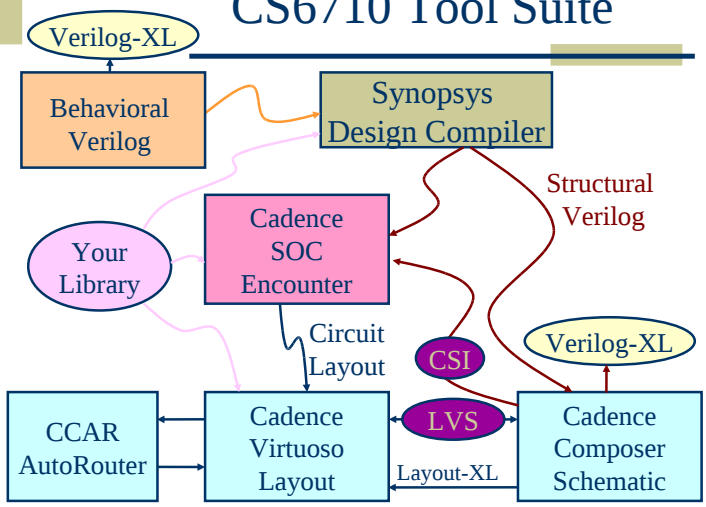
Synthesis and Place & Route

Synopsys design compiler

Cadence Encounter Digital Implementation System (EDI)

CS6710 Tool Suite



Design Compiler

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

Synthesis Process: Design Compiler

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

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 behavioral RTL 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

beh2str – Synthesizes a verilog RTL code to a structural code based on the synopsys technology library specified

Usage: beh2str <input.v> <output.v> <libfile>

beh2str addsub.v addsub_dc.v Lib5710_00.db

Results in addsub_dc.v, addsub_dc.v.rep

addsub.v

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

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

addsub_dc.v

```
module addsub ( a, b, addnsub, result );
  input [7:0] a;
  input [7:0] b;
  output [8:0] result;
  input addnsub;
  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(addnsub), .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) );
  …
  INVX1 U60 ( .A(a[0]), .Y(n52) );
  XNOR2X2 U61 ( .A(b[0]), .B(addnsub), .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 (INVX1)                            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)
```

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 design compiler
source /uusoc/facility/cad_common/local/setups/F13/setup-synopsys
# run (very simple) design compiler synthesis
dc_shell-xg-t -f $SCRIPTFILE
```

.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]
```

beh2str – 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
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
```

What beh2str leaves out...

- Timing!
  - No clock defined so no target speed
  - No wire load model so not as placement constrained
  - 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/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, 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 (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_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 $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 -ungoup_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!!!!

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

addsub_struct.v – 10ns target

```
Startpoint: addnsub (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
addnsub (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
U17/Y (NOR2X1)                           0.53       8.83 f
result[8] (out)                          0.00       8.83 f
data arrival time                                   8.83

clock clk (rise edge)                   10.00      10.00
clock network delay (ideal)              0.00      10.00
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
```

```
Number of ports:                 26
Number of nets:                 111
Number of cells:                 94
Number of combinational cells:   94
Number of sequential cells:       0
Number of macros:                 0
Number of buf/inv:               21
Number of references:             7
```

addsub_struct.v – 4ns target

```
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
clock network delay (ideal)              0.00       4.00
output external delay                   -0.25       3.75
data required time                                  3.75
-----------------------------------------------------------
data required time                                  3.75
data arrival time                                  -3.74
-----------------------------------------------------------
slack (MET)                                        0.01
```

```
Number of ports:                 26
Number of nets:                 248
Number of cells:                231
Number of combinational cells:  231
Number of sequential cells:       0
Number of macros:                 0
Number of buf/inv:               79
Number of references:             9
```

addsub_struct.v – 3ns target

```
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
clock network delay (ideal)              0.00       3.00
output external delay                   -0.25       2.75
data required time                                  2.75
-----------------------------------------------------------
data required time                                  2.75
data arrival time                                  -3.59
-----------------------------------------------------------
slack (VIOLATED)                                   -0.84
```

```
Number of ports:                 26
Number of nets:                 247
Number of cells:                230
Number of combinational cells:  230
Number of sequential cells:       0
Number of macros:                 0
Number of buf/inv:               77
Number of references:             9
```

addsub_struct.v – 3ns target

From the log file:

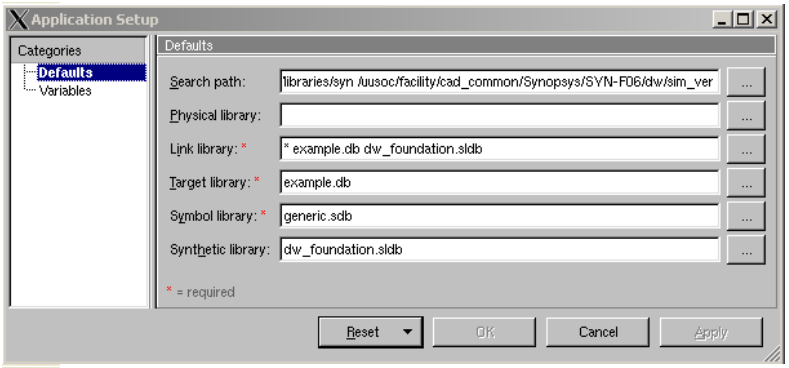
```
max_delay/setup ('clk' group)

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

Using Design Vision

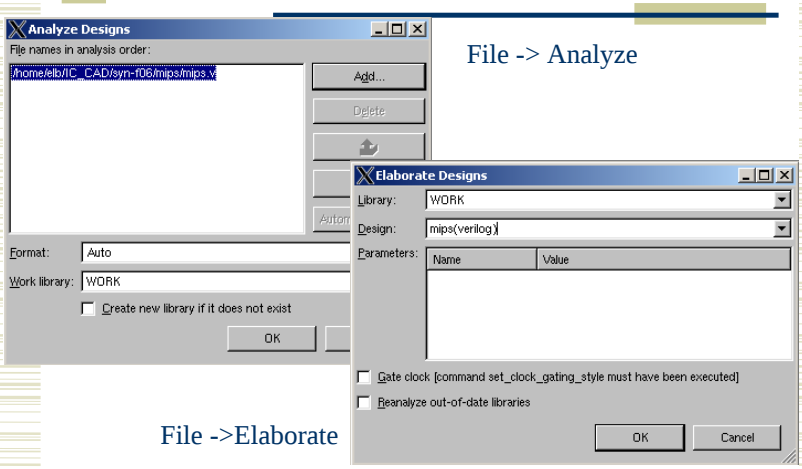
- You can do all of these commands from the design vision gui if you like
- **syn-dv**
- Follow the same steps as the script
  - Set libraries
  - analyze/elaborate
  - define clock and set constraints
  - compile
  - write out results

Setup



File ->Setup

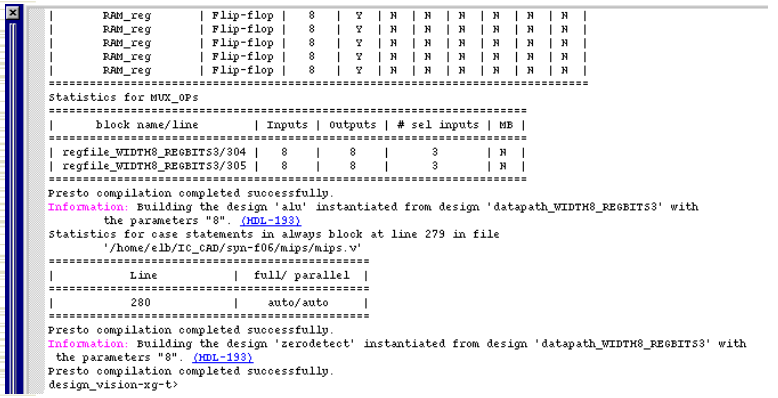
analyze/elaborate



File -> Analyze

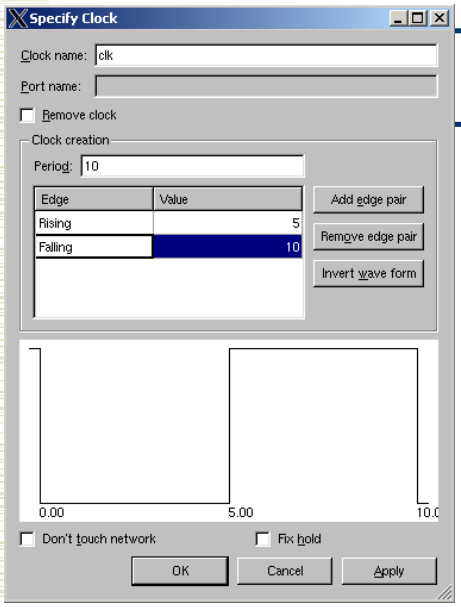
File ->Elaborate

Look at results...



```
|       RAM_reg       | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|       RAM_reg       | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|       RAM_reg       | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|       RAM_reg       | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|       RAM_reg       | Flip-flop |   8   |  Y  | 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-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_WIDTH8_REGBITS3' with
 the parameters "8". (HDL-193)
Presto compilation completed successfully.
design_vision-xg-t>
```

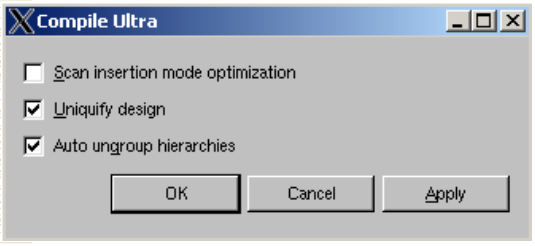
Define clock



attributes -> specify clock

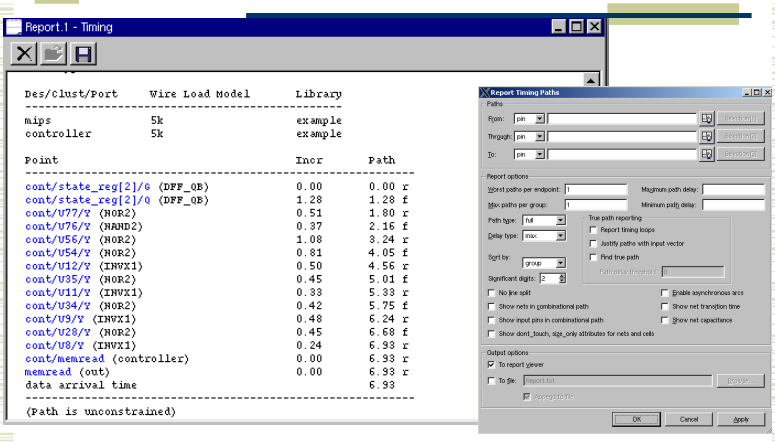
Also look at other attributes...

Compile



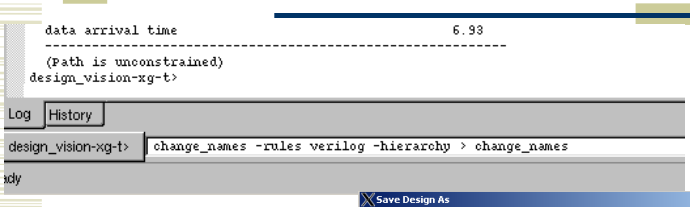
Design -> Compile Ultra

Timing Reports



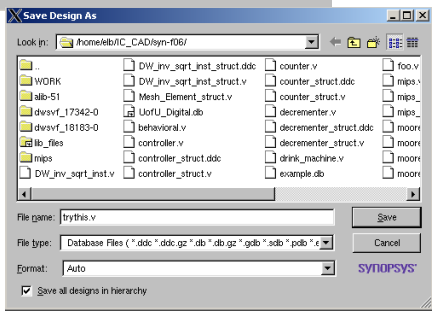
Timing -> Report Timing Path

Write Results



```
data arrival time                                        6.93
-----------------------------------------------------------
(Path is unconstrained)
design_vision-xg-t>
```


change_names

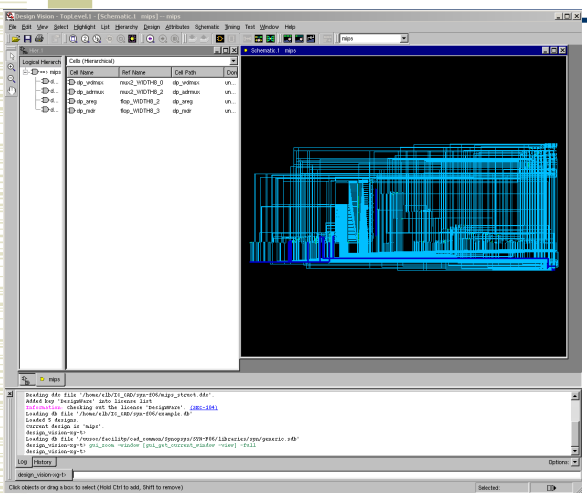


File -> Save As...

Or, use syn-dv after script...

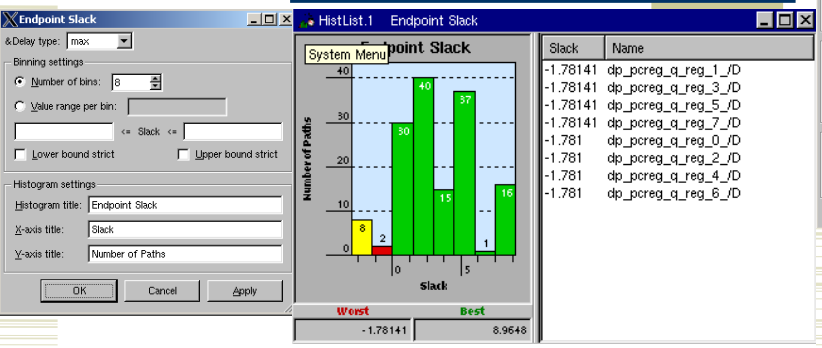
- syn-dc –f mips.tcl
- results in .v, .ddc, .sdc, .rep files
- Read the .ddc file into syn-dv and use it to explore timing...

syn-dv with mips_struct.v



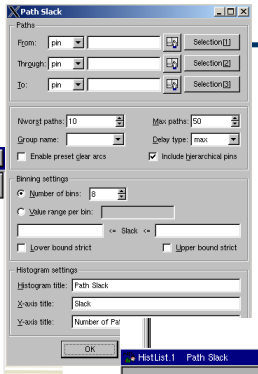
File -> Read

Endpoint slack...

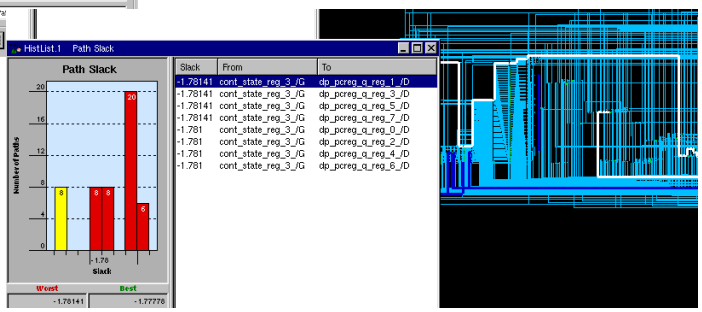


Timing -> Endpoint Slack

Path Slack



Timing -> Path Slack



Encounter Digital Implementation System

- Need structural Verilog, .sdc, library.lib, library.lef
- make a new directory for edi... (very chatty)
- Configuration file sets up names, etc.
  - use UofU_edi.globals as starting point.
- Usual warnings about scripting... top.tcl is the generic script
  - .../local/class/6710/F13/cadence/EDI
- cad-edi

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

Converts structural verilog into physical layout

Shorthand for this process: Place and Route

EDI Usage

- Need structural Verilog, struct.sdc, library.lib, library.lef
- Make a new directory for EDI (very chatty)
- Make an mmmc.tcl file with timing/lib 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 this in a <name>.globals and mmmc.tcl
   - mmmc = multi-mode multi-corner
2. Floorplan
   - Choose physical size, ratio, utilization, etc.
3. Power plan
   - Rings, stripes, row-routing (sroute)
4. Timing optimization – preCTS

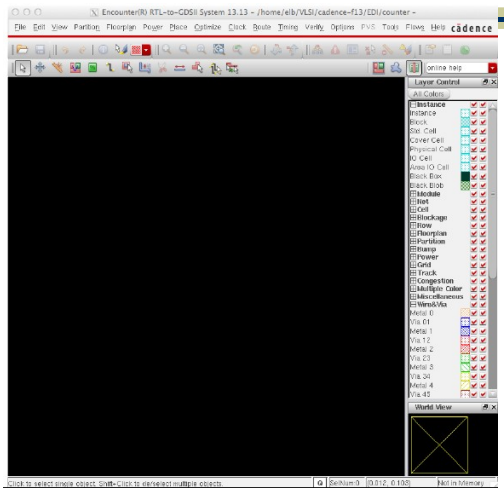
cad-edi Flow

5. Placement
   - Place cells in the rows
   - Timing optimization – preCTS
6. Synthesize clock tree
   - Use your buf or inv footprint cells
   - Timing optimization – postCTS
7. Global routing
   - Nanoroute
   - Timing optimization - postRoute

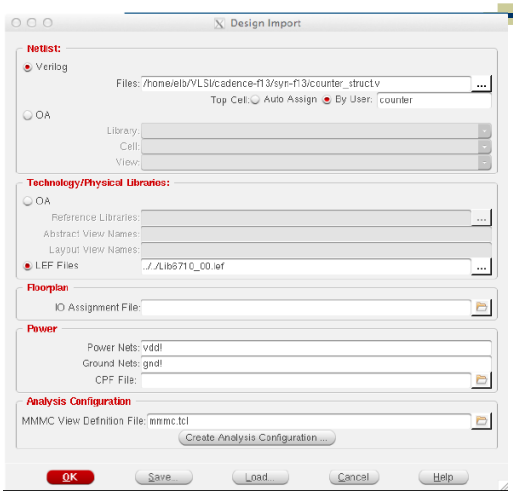
cad-edi Flow

8. Add filler cells
   - Fill in the spots in the row with no cells
   - Adds NWELL for continuity
9. Write out results
   - <name>.def can be imported as layout
   - <name>_edi.v is the placed and routed structural verilog file
   - .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.v!!}
# 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 ground and power nets
set init_gnd_net {gnd!}
set int_pwr_net {vdd!}
```

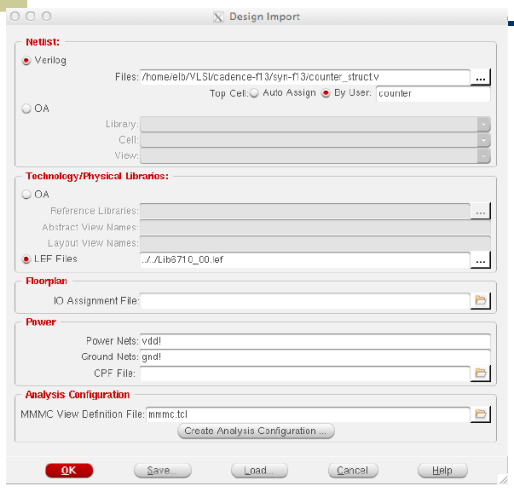
mmmc.tcl

```
# set the name of your .lib file (e.g. lib5710-01.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 constraints 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}
…
```

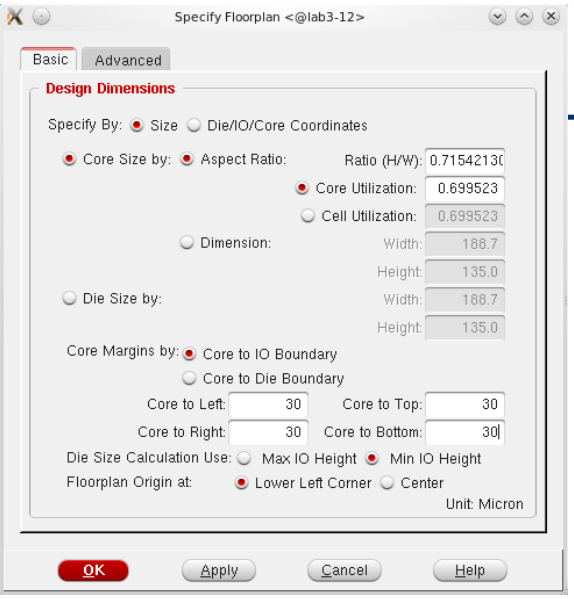
mmmc.tcl

```
#########################################################
# 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}
```

Design Import

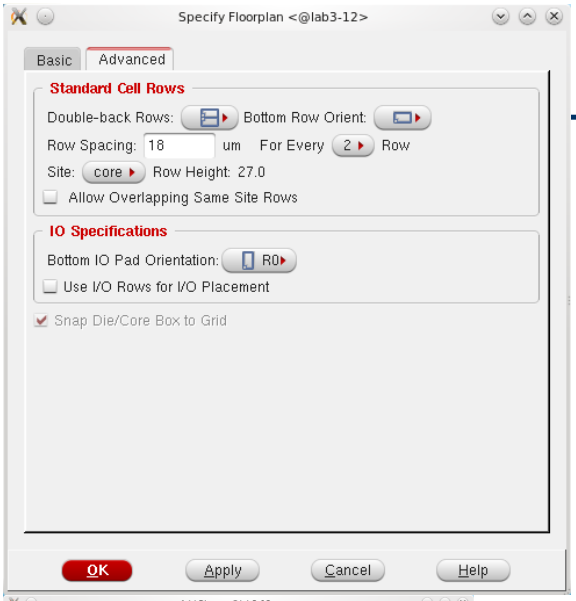


Some screen shots are from an older version of EDI, but not this one...



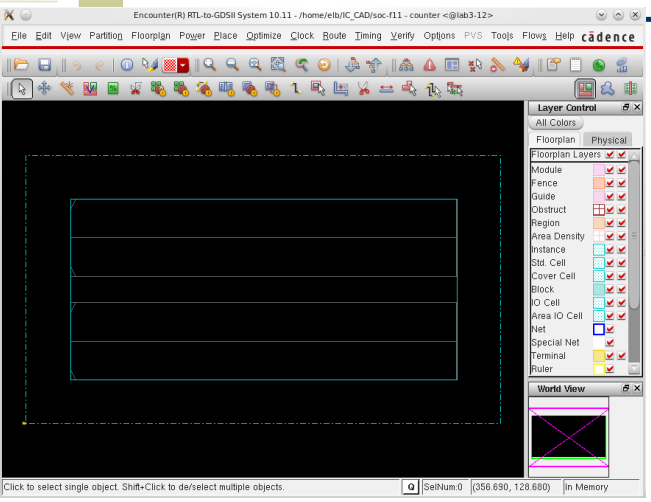
Floorplan

Specify -> Floorplan

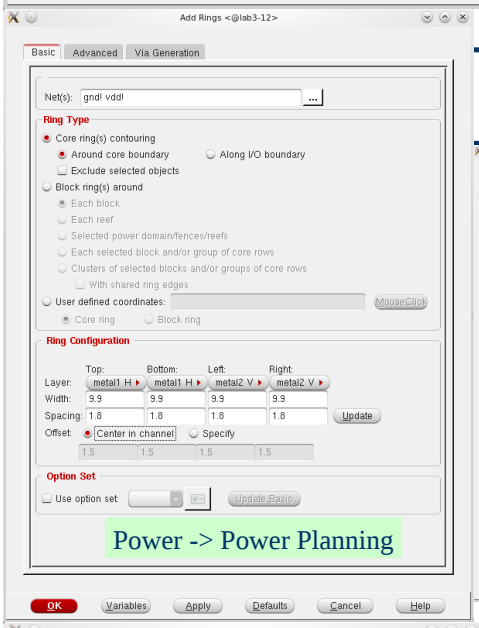


Floorplan

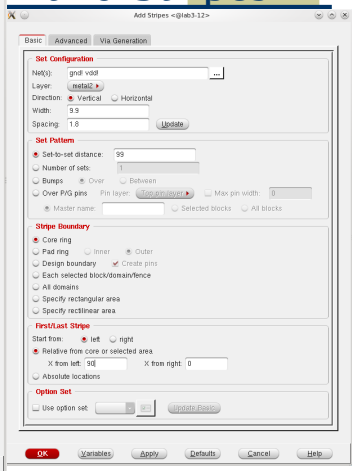
Specify -> Floorplan

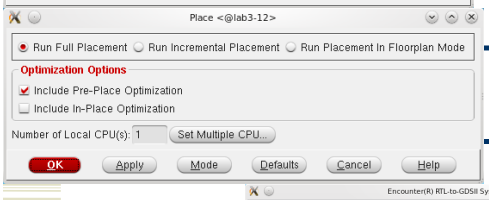


Specify -> Floorplan

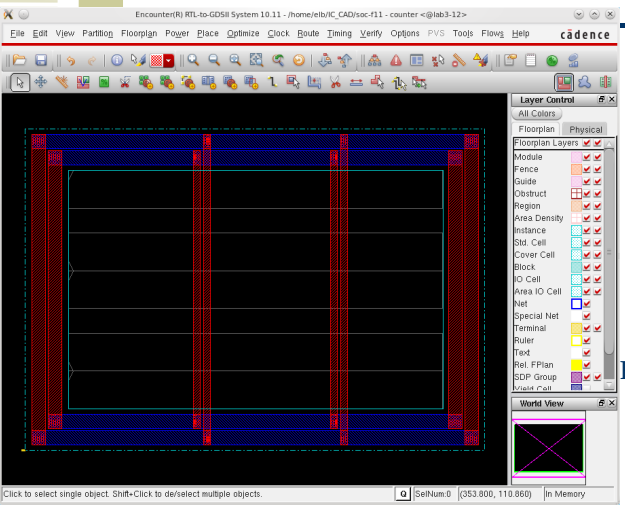


Power Rings and Stripes





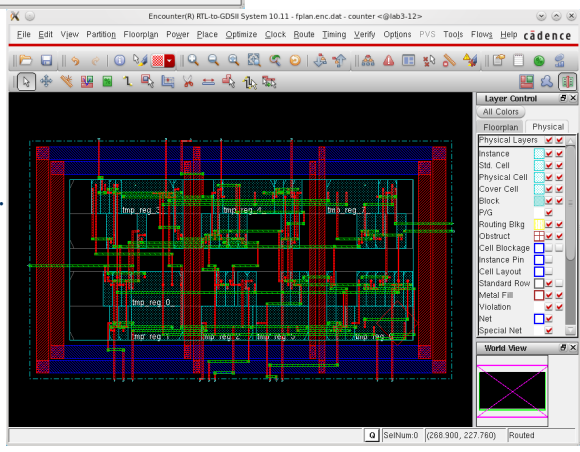
Place cells



Sroute to connect things up

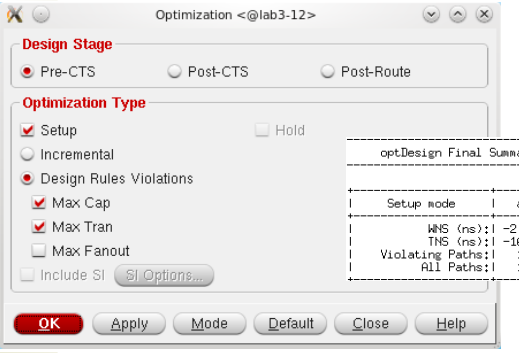
Route -> Sroute

Place -> Place cells...



pre-CTS timing optimization

Timing -> Optimization

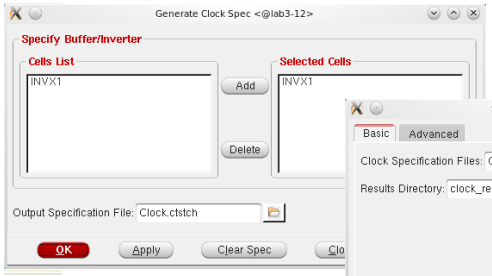


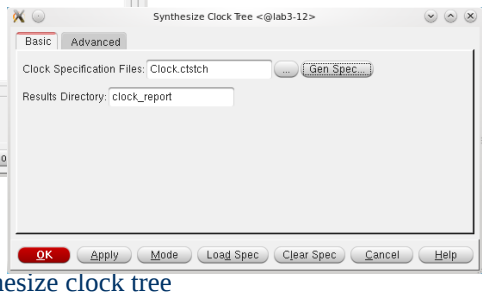
optDesign Final Summary

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

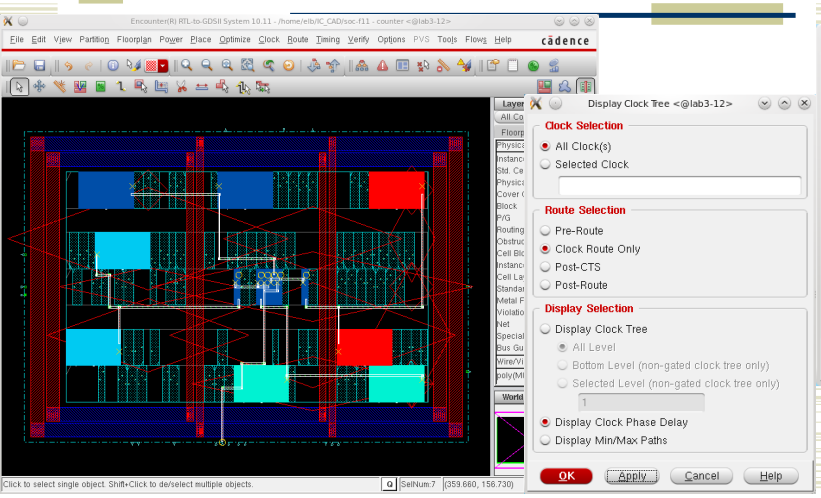
clock -> create clock tree spec



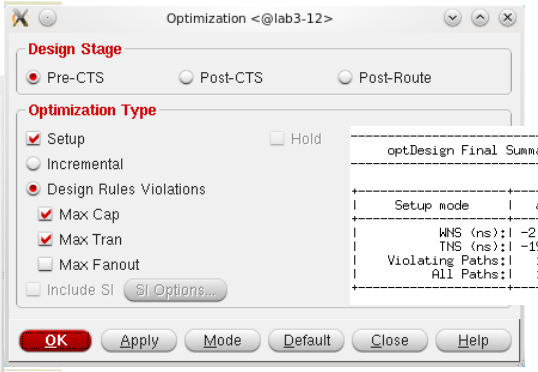


clock ->Synthesize clock tree

Display Clock Tree



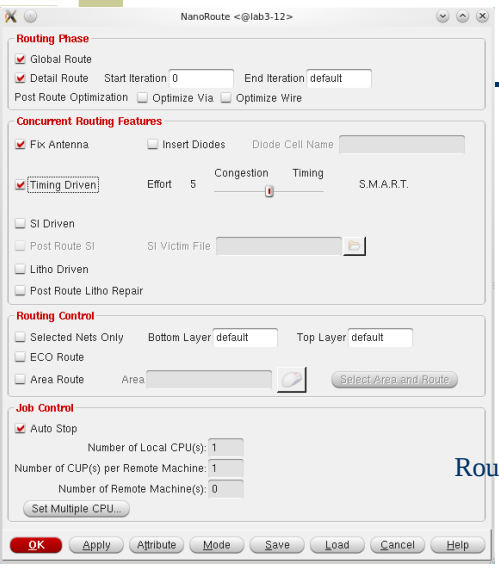
post-CTS optimization



optDesign Final Summary

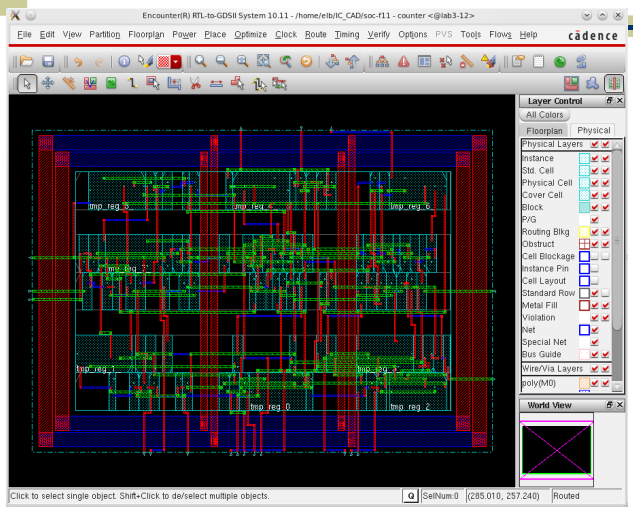
Setup mode	all	reg2reg	in2reg	reg2out
WNS (ns):	-2.238	-2.238	5.216	2.260
TNS (ns):	-19.113	-19.113	0.000	0.000
Violating Paths:	13	13	0	0
All Paths:	149	123	43	18

NanoRoute

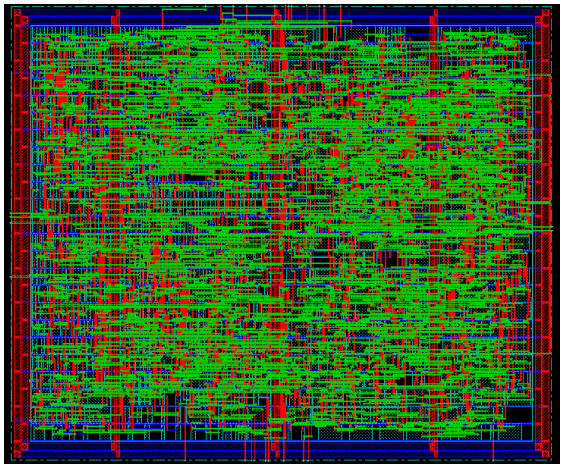


Route -> NanoRoute -> Route

Routed circuit

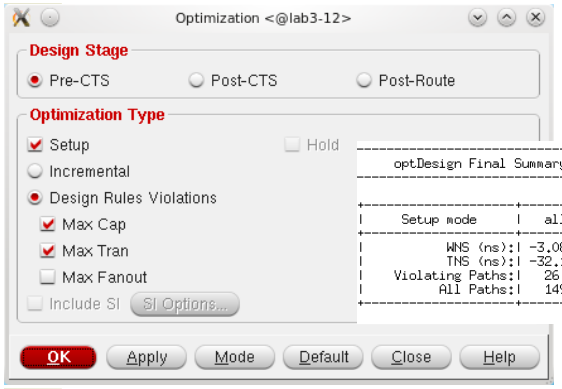


Routed circuit

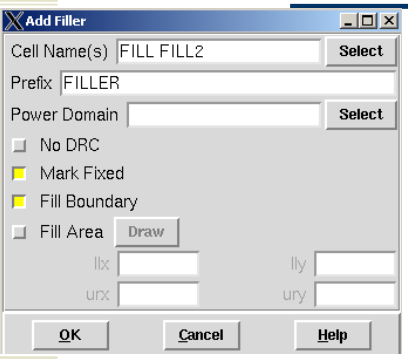


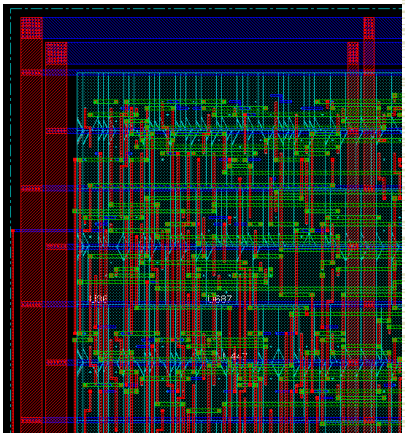
postRoute optimization

Timing -> Optimization



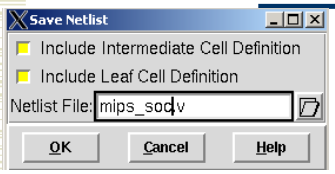
Add Filler





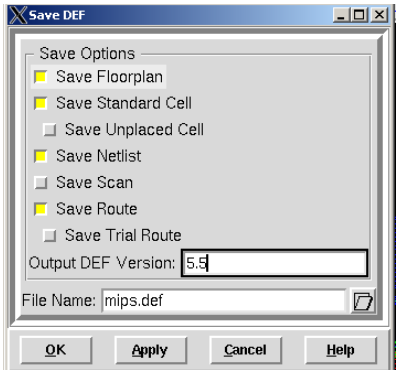
Place -> Filler -> Add...

Write Results...



Design -> Save -> Netlist

Design -> Save -> DEF



Encounter Scripting

- Usual warnings – know what's going on!
- Use opt.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 "mips"
# 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 process uses
set usepct 0.60 ; # percent utilization in placing cells
set rowgap 15  ; # gap (microns) between pairs of rows
set aspect 0.60 ; # aspect ratio ( 1 is square)
set coregap 30.0 ; # gap (microns) between core and 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
set coregap 30.0;     # gap between the core and the power rails
```

top.tcl

```
# Set the flag for SOC to automatically figure out
# buf, inv, etc.
set dbgGPSAutoCellFunction 1

# import design and floorplan
# if the config file is not named $basename.conf,
# edit this file.
loadConfig $basename.conf 0
commitConfig
```

top.tcl

```
# source the files that operate on the circuit
source fplan.tcl;    # percent utilization in placing cells
source pplan.tcl;    # create the power rings and stripes
source place.tcl;    # place the cells and optimize (pre-CTS)
source cts.tcl;      # create 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 for projecst 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
sroute -allowJogging 1

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

top.tcl

Read the script...

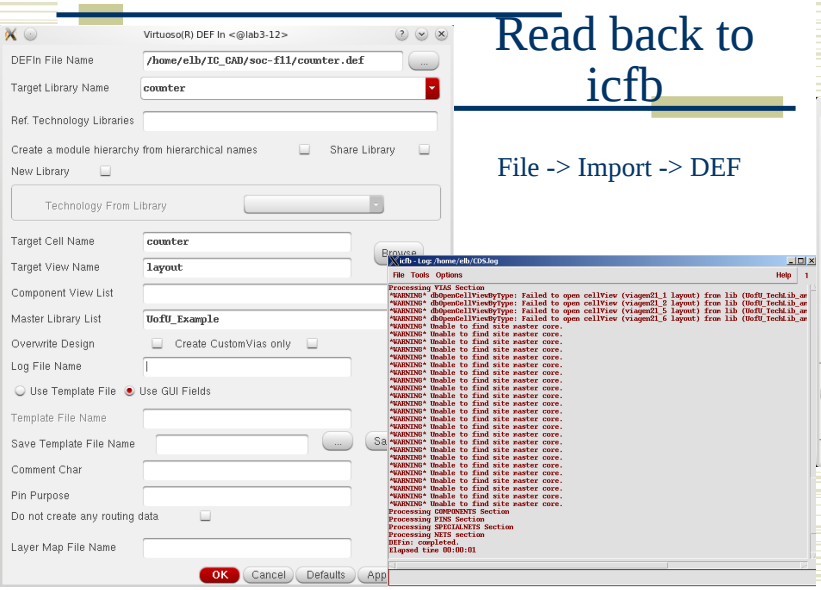
- place
- pre-CTS optimization
- clock tree synthesis
- post-CTS optimization
- routing
- post-ROUTE optimization
- add filler
- write out results

Report Files

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

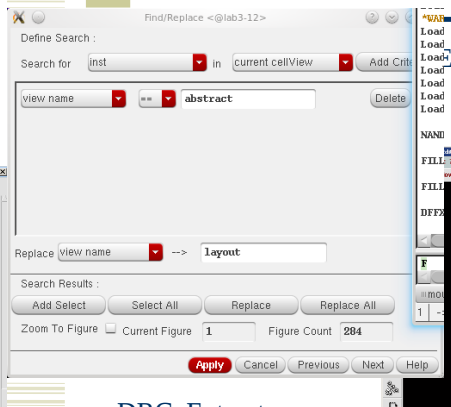
- Desire 0 violations
  - If you have 1 or 2 in the geometry, you might be able to fix them easily in Virtuoso…

Read back to icfb

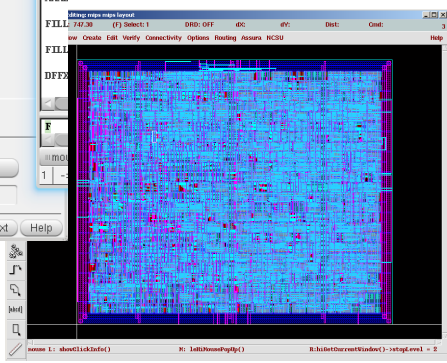


File -> Import -> DEF

Change abstract to layout cellviews

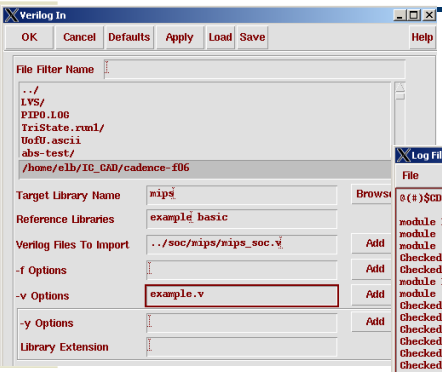


Edit -> Search

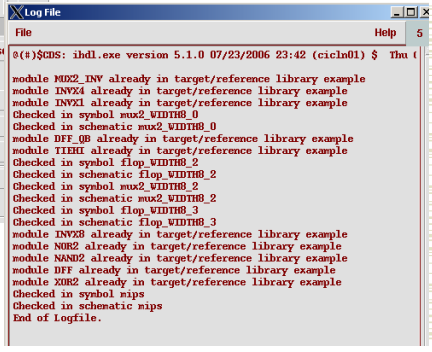


DRC, Extract

Import Verilog

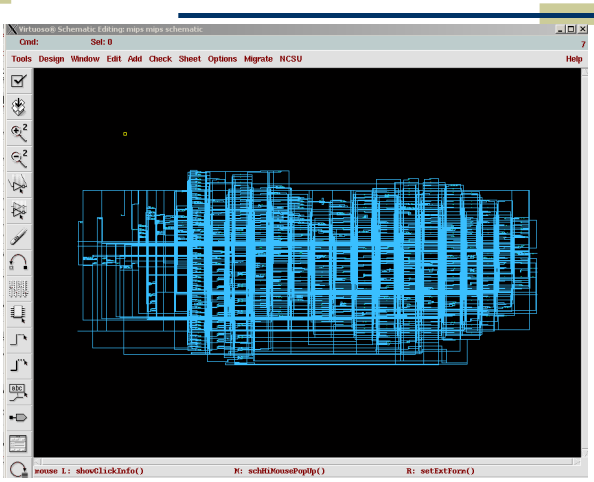


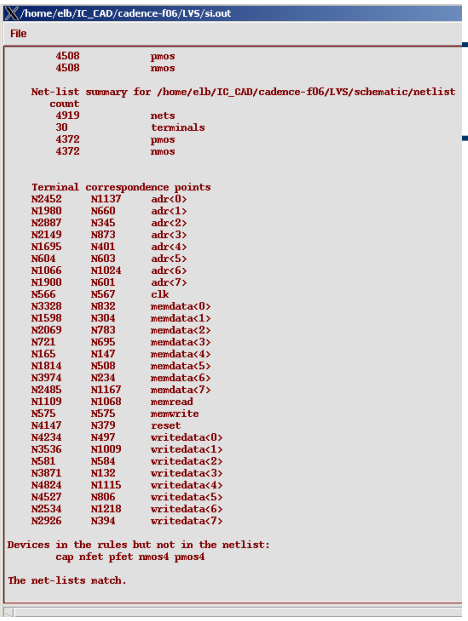
File -> Import -> Verilog



LVS...

Schematic view





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...