NC State University　　　　　　　　　　　　　　　　　　　　Spring 2014
ECE Department　　　　　　　　　ECE 720　　　　　　　　　W. Rhett Davis

Place & Route Tutorial #1

In this tutorial you will use Cadence Encounter to place, route, and analyze the timing and wire-length of two simple designs. This tutorial assumes that you have worked through *Tutorial #1: Introduction to Simulation and Synthesis* on the ECE 520 ASIC Design Tutorials Page and that you know how to simulate, synthesize and analyze timing on basic designs.

I. Setup

- Log in to a Linux machine.
- Download and unpack the file **pr_tut1.tar.gz**. This archive contains a directory called "pr_tut1" with two subdirectories called “counter” and “xbar”. Each of these directories contains two subdirectories called "v" and "pr" with files needed to complete this tutorial. We will start with the counter design and move on to the xbar design.
- Change to the counter/v/synth directory and synthesize the simple "counter.v" design with the command “make”. This design is copied from the ECE 520 tutorial #1 mentioned above. When complete, you should have a file called “counter/v/src/gate/counter_final.v”, which will serve as the starting point for this tutorial.

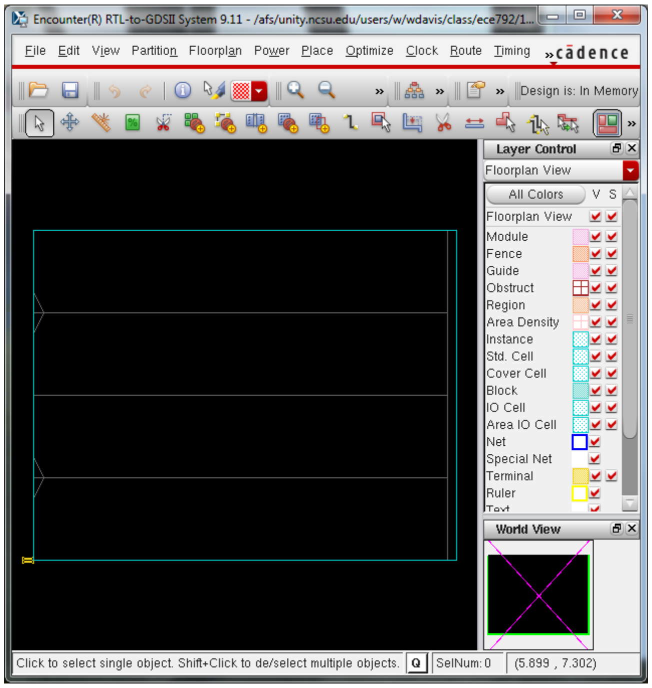
II. Initialize the Design

1. Change to the PR directory and Start Encounter with the commands

```
add cadence2012
encounter
```

2. Choose **File→Import Design** and click "Load" at the bottom of the next pop-up window. Browse and choose the file "design.conf". You'll notice that the "Design Import" window has been updated with various bits of information, including the following: (You should update this file as needed to for your own designs)
    - **Netlist file** - (set to "../v/src/gate/counter_final.v") This file gives the standard-cell netlist that will be placed and routed.
    - **Top Cell** – (set to “counter”) This gives the name of the top-level cell in the verilog file.
    - **Timing Constraint File** - (set to "design.tc") This file gives the timing constraint(s) that will be used for the design. The syntax for this file mimics Synopsys Design Compiler's syntax. Look in this file and you'll see that it defines the clock port and a 10 ns clock period.
    - **LEF Files** - a list of LEF files defining the design rules and abstract layout views for the standard cells
    - **Timing Libraries** - a list of Synopsys LIBERTY files defining the behavior and timing of the standard cells
3. Click "OK" to close the dialog box and import the design.
4. At this point, your design should look like the figure below. Note the message on the top-right that says "Design Is: In Memory", signifying the overall status of your design. You can zoom in by right-clicking and draging a box, and pan by middle-clicking and dragging. Some convenient

key-bindings are included below.


Encounter(R) RTL-to-GDSII System 9.11 - /afs/unity.ncsu.edu/users/w/wdavis/class/ece792/1...
File Edit View Partition Floorplan Power Place Optimize Clock Route Timing
cadence
Design is: In Memory
Layer Control
Floorplan View
All Colors V S
Floorplan View
Module
Fence
Guide
Obstruct
Region
Area Density
Instance
Std. Cell
Cover Cell
Block
IO Cell
Area IO Cell
Net
Special Net
Terminal
Ruler
World View
Click to select single object. Shift+Click to de/select multiple objects.
SelNum: 0
(5.899 , 7.302)

Key	Action	Description
q	Attribute	Display the object attribute form on selected object.
f	Fit Display	Zooms the display to fit the design area.
g	Group	Moves up the hierarchy on the highlighted Hinstance.
Shift-g	Ungroup	Moves down the hierarchy on the highlighted Hinstance.
z	Zoom-in	Zooms in the display, 2x.
Z	Zoom-out	Zooms out the display, 2x.
Shift	Select	Allows multiple selections of objects.
Arrows	Pan	Pans the display in direction of arrow.
k	Ruler	Creates a ruler for measuring distance.
Shift-k	Clear Rulers	Clears all rulers in the view.
Space Bar	Focus	Changes the focus of overlapping objects

5. Now that the design has been loaded, you can analyze the timing of the design. Choose **Timing→Report Timing...** and set the "Design Stage" to "Pre-Place" (which should be the only possible option at this point). Note the output directory is set to "timingReports" by default, and the file-name prefix is set to "counter_preCTS". Report files with this prefix will be put in the named directory. Click "Ok" to run the timing analysis. You should see the following output appear in the UNIX shell window:

```
+--------------------+---------+---------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|           WNS (ns):|  7.986  |  7.986  |  8.579  |   N/A   |   N/A   |   N/A   |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |   N/A   |   N/A   |   N/A   |
|    Violating Paths:|    0    |    0    |    0    |   N/A   |   N/A   |   N/A   |
|          All Paths:|    4    |    4    |    4    |   N/A   |   N/A   |   N/A  |
+--------------------+---------+---------+---------+---------+---------+---------+
```

This report shows that the worst negative slack (critical-path, labeled "WNS") slack is 7.986 ns. This is pretty good! Our clock period is not set very aggressively for this design. If we reduce the clock period in the design.tc file by more than 7.986ns, then we will see a negative number here. The number of violating paths is 0, which is consistent with the fact that the worst negative slack is positive. The total negative slack (labeled TNS) is also provided, which is the sum of the negative slack on all paths. This is an important metric, because it gives us an indication of how hard a problem is to fix. If the WNS and TNS are about the same, then there is a small number of violating paths, and we may be able to fix the problem with careful floorplanning. If the TNS is far greater than the WNS, then a tremendous number of paths are violating the timing constraints, and we may have no alternative except to increase the clock period.

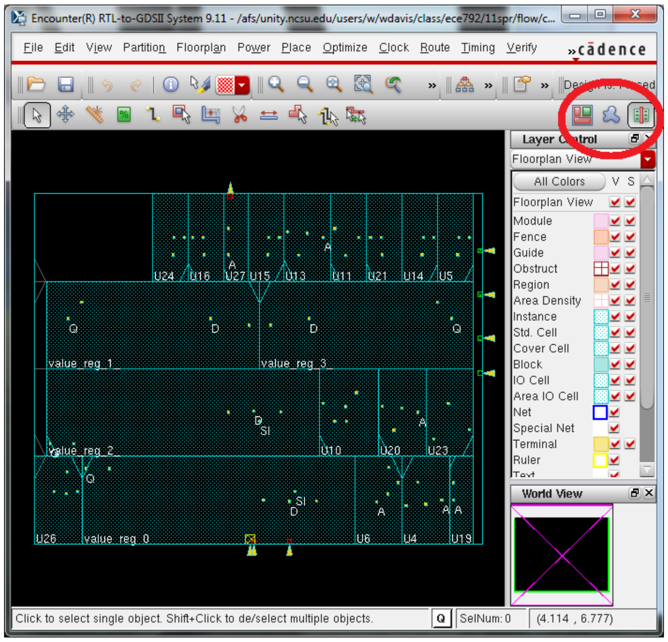
For more detail, look in the file "timingReports/counter_preCTS_in2reg.tarpt". The first path listed in this file is the longest path originating from an input pin. Compare this to the critical path given in the file "../v/synth/timing_max_slow_holdfixed_tut1.rpt", and you'll note that the paths are nearly identical, but the delays are not exactly the same. This is partly because no input delay was annotated in *Encounter*, and also because different timing engines and assumptions will produce different results. Lastly, the overall critical path recognized by *Encounter* is given in the file "timingReports/counter_preCTS_all.tarpt". This path differs significantly from the path identified by Design Compiler, again because no input delay was annotated.

6. Lastly, note that all of the output that you see in the UNIX shell window is also recorded in the file *encounter.log*. By default, this file is never over-written, so if you exit and start encounter again, you will see multiple files named *encounter.log1*, *encounter.log2*, and so on. We'll be referring to this file throughout the rest of this tutorial.

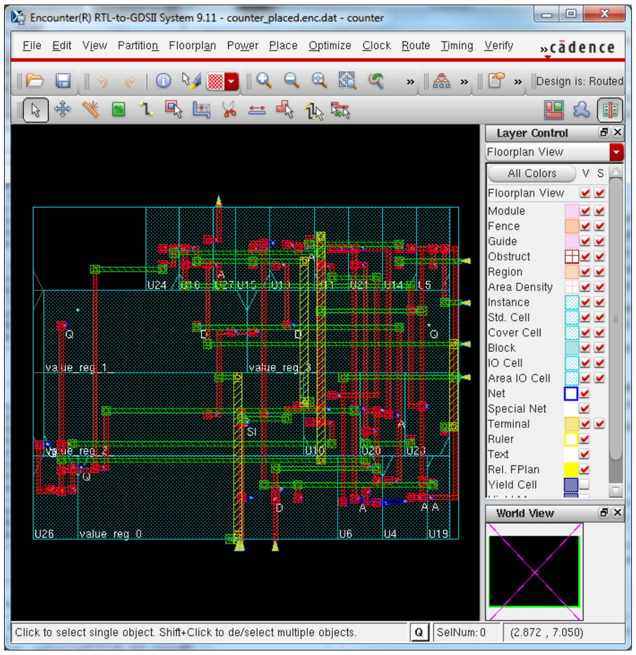
III. Place & Route the Design

Choose **Place → Place Standard Cell...** to run the Amoeba Placer. Click OK on the next pop-up to run placement with default options. Once complete, you will notice that the pins (small yellow triangles) have moved around the edges of the design, and that the status indicator has been updated to "Design Is: Placed". But you won't see anything else. This is because you are in the "Floorplan View", which shows only your initial planning view. To see the placed cells, you'll need to switch to the "Physical View". Toggle between these views with the three buttons on the far right of the buttons bar (below the menu bar). These three buttons are circled in the picture below. From left to right, these buttons activate the "Floorplan View", "Amoeba View", and "Physical View".

Choose **File → Save Design...** and save as "counter_placed.enc". Once saved, you can open the design with **File → Restore Design....**


Encounter(R) RTL-to-GDSII System 9.11 - /afs/unity.ncsu.edu/users/w/wdavis/class/ece792/11spr/flow/c...
File Edit View Partition Floorplan Power Place Optimize Clock Route Timing Verify
cadence
Layer Control
Floorplan View
All Colors V S
Module
Fence
Guide
Obstruct
Region
Area Density
Instance
Std. Cell
Cover Cell
Block
IO Cell
Area IO Cell
Net
Special Net
Terminal
Ruler
World View
Click to select single object. Shift+Click to de/select multiple objects.
SelNum: 0
(4.114 , 6.777)

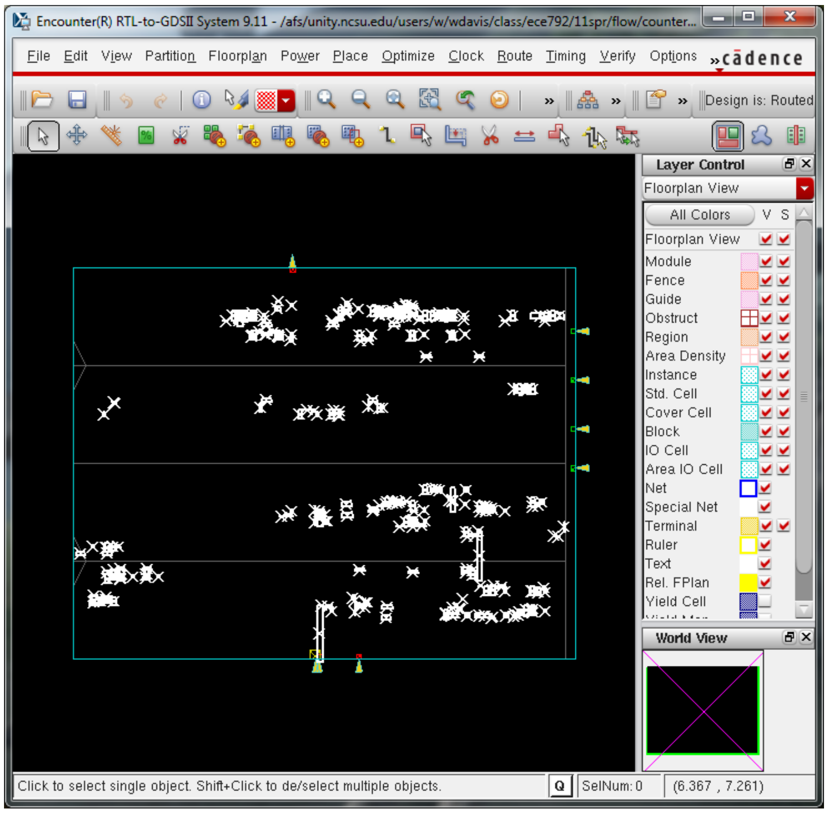
Choose **Route → Trial Route** and click OK on the next pop-up to route with default options. Note that the status updates to "Design Is: Routed". Your routed design should look like the one below:



The trial route is a fast, approximate routing used for estimating wire-lengths and planning where wires will go at a high level (similar to a "Global Route", except that wires are actually drawn). One of the main short-cuts that is taken during a trial route is to leave lots of short-circuits between wires. We can illustrate this by running a geometry check, which is normally done after a "Detail Route"). Choose

Verify→Verify Geometry... and click Ok on the pop-up window. When done you will see a number of white crosses appear in the view, marking the violations. Switch back to the floorplan view, and you should see the image below. You should also see the following messages appear in the *encounter.log* file:

```
  VERIFY GEOMETRY ...... Starting Verification
  VERIFY GEOMETRY ...... Initializing
  VERIFY GEOMETRY ...... Deleting Existing Violations
  VERIFY GEOMETRY ...... Creating Sub-Areas
                  ...... bin size: 2080
  VERIFY GEOMETRY ...... SubArea : 1 of 1
  VERIFY GEOMETRY ...... Cells          :  0 Viols.
  VERIFY GEOMETRY ...... SameNet        :  37 Viols.
  VERIFY GEOMETRY ...... Wiring         :  257 Viols.
  VERIFY GEOMETRY ...... Antenna        :  0 Viols.
```



Ideally, a completed design will have zero violations. Any violations that cannot be fixed within Encounter will need to be fixed manually, using a tool such as Cadence Virtuoso. For now, however, we won't worry about them, because these violations are likely to be fixed once we perform a Detail Route.

The main reason for performing a trial route is so that we can get a more accurate idea of delays due to wire parasitic capacitances and resistances. Choose **Timing→Report Timing...** and set the "Design Stage" to "Pre-CTS". This time the critical-path slack decreases slightly to 7.847 ns. Look through the *encounter.log* file, and you'll see the following message:

```
Extraction called for design 'counter' of instances=24 and nets=36
using extraction engine 'preRoute'
```

There are many parasitic extractors, each with varying fidelity and computation-time. Here, the fast, pre-route extractor is used. To see the effect of changing the extractor, choose **Timing→Report Timing...** a third time and set the "Design Stage" to "Post-Route"

```
Extraction called for design 'counter' of instances=24 and nets=36
using extraction engine 'postRoute' at effort level ‘low’
```

Now the slack decreases slightly (again) to 7.730 ns. The choice of the timing analyzer to use is not always easy. It's a matter of how much time you want to spend in calculation, what effects you're looking for, and whether or not you have created the necessary technology files for the extractor. For our purposes, the important thing is to make note of the kind of extractor that was used for a particular timing report, because it is difficult to tell if a particular design change has had a positive or negative effect, if the timing reports were created with different types of extractors.

Choose **Design→Save Design** and save as "counter_trialrouted.enc".

IV. Writing Scripts

So far in this tutorial, we have been exploring the Encounter user interface. This is important, because when you’re trying to figure out how to fix problems, you need to have a way to explore lots of information and try possible fixes very quickly. However, as we have seen, critical issues such as the design’s timing and short circuits depend on the *precise details* of which commands were used and also the *order* in which they were done. It is nearly impossible to guarantee that you will issue the same commands in the same order every time you work through a design, especially on large designs, for which it can take hours to run some of these steps. To guarantee consistency from one run to the next, we must get in the habit of using scripts to automate the design process.

Encounter, like most other VLSI CAD tools, uses the Tool Command Language (Tcl) as a scripting framework. The first step in writing scripts is to learn more about Tcl. The best way to do that is to work through the tutorial, which can be found at http://www.tcl.tk → Documentation → Tcl 8.5 Tutorial. This tutorial is broken into about 50 sections, each about 1 page in length. If you ever get confused about the Tcl syntax, then go back to this tutorial and work through a few more sections.

The Tcl tutorial will teach you about the basics, but to automate *Encounter*, we need to know about the procedures that have been automated for *Encounter*. There are three ways to figure out what these commnds are:

1. **encounter.cmd file** - To aid in creating scripts, *Encounter* prints the equivalent Tcl commands for each command you issue through the GUI. These commands are collected in the file *encounter.cmd*, (and *encounter.cmd1*, *encounter.cmd2*, and so on). They are also printed in the

encounter.log file, with the prefix “<CMD>” to help you separate the commands from the output. Here are the contents of my encounter.cmd file, after running this tutorial up to this point:

```
loadConfig design.conf 0
commitConfig
fit
setDrawView fplan
clearClockDomains
setClockDomains –all
timeDesign -prePlace -idealClock -pathReports -drvReports -slackReports
     -numPaths 50 -prefix counter_preCTS -outDir timingReports
getMultiCpuUsage –localCpu
setPlaceMode -fp false
placeDesign –prePlaceOpt
setDrawView place
saveDesign counter_placed.enc
trialRoute -maxRouteLayer 10
setDrawView place
fit
verifyGeometry
setDrawView fplan
setDrawView place
clearClockDomains
setClockDomains –all
timeDesign -preCTS -idealClock -pathReports -drvReports -slackReports
     -numPaths 50 -prefix counter_preCTS -outDir timingReports
clearClockDomains
setClockDomains –all
timeDesign -postRoute -pathReports -drvReports -slackReports -numPaths 50
     -prefix counter_postRoute -outDir timingReports
saveDesign counter_trialrouted.enc
```

2. **Encounter Documentation** – The encounter documentation is located in the following directory:

   /afs/eos.ncsu.edu/dist/cadence2012/edi/doc

   From this directory, there are a number of subdirectories that contain HTML and PDF versions of each document. The following are the documents that you will probably use the most:
   - fetxtcmdref (First Encounter Text Command Reference) – This file gives detail on all of the Tcl commands that are available.
   - soceUG (SoC Encounter User Guide) – This file gives the best documentation on how Encounter is intended to be used to place and route your design.
   - encounter (Encounter Menu Reference) – This file gives a break-down of the GUI commands, which can sometimes be more helpful to use when exploring for a solution to a problem

3. **info command** – The Tcl “info” command can be used to print out detail about the available commands, including the lists of the available commands (with *info commands ?pattern?*) and the

arguments expected by a command (with *info args procname*). This is especially important, since the documentation is often out-of-date, and it can be maddening to explore for a solution when you don't know what the options are. See sections 33-35 of the Tcl tutorial for more detail on using the info command.

Using this approach, you can fairly quickly assemble a Tcl script to execute your design. The preferred way to organize a flow is to break it into a set of steps (*e.g.* Initialize, Place, Route, etc.) that can be run independently, thereby allowing you to better manage your time by learning which steps need to be run and how long it takes to run each step. Each script is best executed from the UNIX command line as follows:

```
encounter -nowin -overwrite -replay run_[step].tcl |& tee run_[step].log
```

The *nowin* option avoids running the GUI, which takes extra time to load, and *overwrite* overwrites the log files, so that you don't get hundreds of *encounter.log* and *encounter.cmd* files as you run the scripts repeatedly. The last arguments direct both *stdout* and *stderr* to a log file with a known name, so that you always have a record of the current status of the design and a place to go back to when debugging. Another reason for using *stdout* (rather than the *encounter.log* file) is that user-defined log messages will not show up in the encounter.log file, but they will show up in *stdout*.

The remainder of this tutorial will take you through the base-line flow that we will use for this course.

V. Execute the Baseline Flow

Before we execute the flow, look at the file setup.tcl. This file contains three things:

- **modname variable** – The top-level module name is referenced quite often. Putting this variable here makes the scripts reusable on other designs.
- **topmetal variable** – When placing and routing large designs, we often want to reserve higher levels of metal for later steps. This variable allows us to specify to maximum number of metal layers to use, currently set to 10, which is the maximum.
- **timef procedure** – This procedure is defined to convert seconds into hours and minutes, so that we can have a convenient message at the end of each log file telling us how long it took to run.

Note also the file README.txt, which contains a brief list of the commands that should be run to execute the entire flow. It also includes a reminder of all the things you'll need to change when re-targeting this flow for a new design. For brevity, we won't include the commands to execute in the remainder of this file. Refer to the README.txt file for these commands.

The baseline design flow that we will use is organized in the following steps:

1. **init** – *Loads the design.conf file and generates an initial floorplan.*

   This step may seem trivial, but as designs get larger floorplans will become more complex, and we'll need to issue a lot of commands before placing the design. For now, go ahead and run the step. Then examine the run_init.tcl file, and you'll see the following:

a. **Header** – the setup.tcl file is sourced, and the current time is recorded for noting the total run-time.
b. **Design Import** – The *design.conf* file is loaded with the *loadConfig* command
c. **Initial Floorplan** – An initial floorplan is created with the *floorplan* command. We skipped this step before, because the default floorplan was enough for us. Now, we want to do some basic power routing, so we'll need to add some space to create power and ground rings. Refer to the *fetxtcmdref* documentation for info on what the parameters of this command mean. Do note, however, that two versions of the *floorplan* are included, one with the *–s* option, and another (which is commented out) with the *–su* option. The –s option is used to specify the exact dimensions of the floorplan, which is necessary for a design that is so tiny as our counter example. With a larger, however, it is better to specify an aspect ratio and density. The first two arguments of the *floorplan –su* command give the aspect ratio (1.0) and density (0.95) for the desired initial floorplan. You should comment out the *–s* version and use the *–su* version on a larger design.
d. **Power Routing** – The *addRing* and *addStripe* commands connect power in the design. Unconected power rails won't cause many problems for us in this course, because the tools we use will generally infer power connections. However, unconnected power and ground rails will cause a design to fail and are necessary for detailed parasitic extraction with tools such as Mentor Graphics Calibre. We will come back to this issue in a later tutorial.
e. **Timing Analysis** – A pre-place timing analysis is done
f. **Save Design** – The design is saved, so that we can start from this point on the next step. We can also start up *Encounter* and restore this design to view its status after this step.
g. **Footer** – The elapsed time is printed, along with a message that the command completed successfully. This message is important, because when running a lot of scripts, it can be hard to find where things went wrong sometimes. The first thing that we should always check is whether or not a script completed successfully, or whether the script failed for some reason during execution.

2. **place** – *Places the standard cells*

This command is much like the previous step, except that it places the cells, as we did interactively earlier. The one big difference is the IO placement file, which specifies how the inputs and outputs should be organized around the periphery of this design. The authoring of these placement files can be time-consuming, partly because the syntax is hard to remember, and also because it simply takes a lot of time to type out the names of all the pins. This step checks for an IO placement file named *io.place*. If the file exists, it is loaded. If not, then it dumps a template called *io.tmpl*. You can copy the file *io.tmpl to io.place* and modify it to alter the positions of the pins. Then re-run the **place** step.

Go ahead and run this step now. For now, we will ignore the *io.place* file, so there is no need to run the **place** step a second time.

3. **cts** – *Clock-Tree Synthesis*

   This command performs the following:

   a. **Pre-CTS Optimization** – In order to guarantee the best performance of the design, it's best to re-optimize the placement and/or routing every time more detail is added. Otherwise, it may be too complicated for *Encounter* to find a good solution. This is more time-consuming, but leads to much better solutions. This script starts with a pre-clock-tree-synthesis placement optimization.
   b. **Clock Tree Synthesis** - We skipped earlier for simplicity. We will come back to clock-tree synthesis in a later tutorial, but it is such an important part of the flow that it's best not to skip completely. This step looks for a clock-tree specification file called *clock.ctstch*. If not found, it generates a template called *clock.tmpl*. As with the **place** step, we can copy *clock.tmpl* to *clock.ctstch* and re-run the **cts** step to generate the clock-tree. Unlike the **place** step, we must re-run the **cts** step, or the design will remain essentially unaltered from the previous step.
   c. **Repeater Insertion** - Repeater insertion could be a step on its own, but we include it here for convenience. The file *repeater.rule* is provided for this purpose.
   d. **Post-CTS Optimization** – Two post-CTS placement optimizations are executed to help fix setup-time and hold-time violations, respectively.

      Go ahead and run this step, copy the file and re-run the step. We will discuss clock-trees in more detail in a future tutorial.

4. **trialroute** – *Trial route, extraction, and timing analysis*

   This step performs a trial-route, extraction, and timing analysis as we did earlier. Note also that a verilog netlist and SPEF file are created, to allow timing and power analysis in other tools (such as *Synopsys Design Compiler* or *Prime Time*). This is the same behavior that we see in the ECE 520 *PAD_Flow.pl* tutorial.

   Note that the **trialroute** step restores the design *counter_cts* if it exists, or *counter_placed* otherwise. This means that we are free to skip the **cts** step, if we want to get a fast estimate of the wire-delays for a design.

5. **route** – *Detail route, extraction, and timing analysis*

   This step performs a detail route, which we did not do earlier. The detail route shouqld perform all of the details needed to make the design error-free, meaning no short or open circuits. Run this step now, and you'll see the following in the *run_route.log* file:

```
#Start Detail Routing.
#start initial detail routing ...
#    number of violations = 0
#cpu time = 00:00:00, elapsed time = 00:00:00, memory = 262.00 (Mb)
#start 1st optimization iteration ...
#    number of violations = 0
#cpu time = 00:00:00, elapsed time = 00:00:00, memory = 262.00 (Mb)
#Complete Detail Routing.
```

This output is telling us the detail route completed with essentially no violations (which are mainly short and open circuits) in virtually no time at all. In a more complex, congested design, you would see a large number of violations and a number of iterations, as the router attempts to connect all of the wires. In general, it's fine to start out with a huge number of violations, as long as the total number drops significantly from one iteration to the next. But in an overly-congested design, the number of violations will reach a minimum values and remain unchanged from one iterations to the next. This means that the route step can take many hours to run, before the router finally gives up. In such cases, you generally need to solve the problem by increasing the number of routing layers, reducing the density (giving the router more area to work with), or using special floor-planning commands to relieve the congention in a problem area.

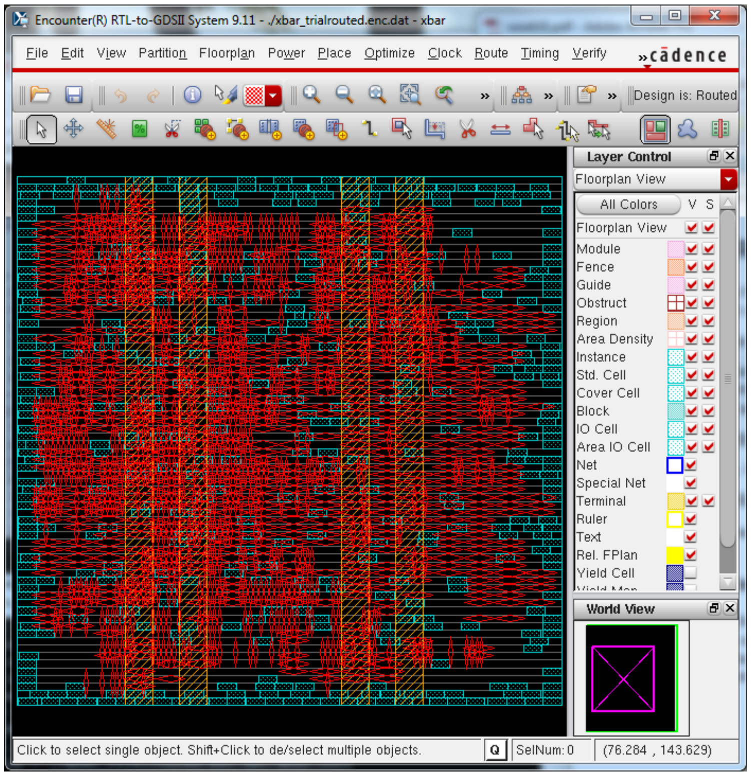
Examine the rest of the run_route.tcl file, and you'll see three more things happening that are new:

a. **Routing power nets** – the *sroute* command is used to route the power rails. We drew the rings in the init step, but here is where they get connected to the cells.
b. **Add Filler Cells** – The gaps between cells need to be filled in order to ensure that no design-rules are violated. The standard-cell library provides filler cells for this purpose. They need to be added at the last minute, since the gaps are needed for clock-tree-synthesis and other optimizations.
c. **Post-Route Optimization** - Two post-route placement & routing optimizations are executed to help fix setup-time and hold-time violations, respectively
d. **DEF file output** – The design is saved in a Design Exchange Format (DEF) file. DEF files are used for interchanging physical design data between a variety of CAD tools, and we will use them to calculate various statistics on our design. For more information on the format of this file, refer to the **lefdefref** file in the documentation directory.

We're done! Look in the *counter/pr/expected* directory for a set of files that should be very similar to the output that you get when you run this tutorial.

VI. Place & Route the Crossbar Design

Now that you have completed the baseline flow on the simple counter design, re-run the flow on the crossbar design located in the *xbar* directory. You'll note that the *floorplan –su* command is used with 95% density in the *run_init.tcl* script, and that 6 metal layers are specified in the *setup.tcl* file. Go ahead and synthesize the crossbar, and then run the flow up through the **trialroute** step. Open up the *xbar_trialrouted.enc* design and you should see something like the following:



The red diamonds that you see are congestion markers, indicating how many more horizontal and vertical routing tracks are needed beyond what is available. You can zoom into these markers to see more detail. See the "Congestion Markers in the Display" section of the **soceUG** documentation for more info on how to read these markers. Look in the *run_trialroute.log* file, and you'll see the following output.

```
Congestion distribution:

Remain  cntH            cntV
--------------------------------------
 -6:    0        0.00%  1        0.01%
 -5:    0        0.00%  5        0.03%
 -4:    23       0.15%  63       0.42%
 -3:    151      1.01%  338      2.26%
 -2:    584      3.90%  1039     6.93%
 -1:    1370     9.14%  1677    11.19%
--------------------------------------
  0:    1616    10.78%  1659    11.07%
  1:    1130     7.54%  879      5.87%
  2:    653      4.36%  469      3.13%
  3:    479      3.20%  337      2.25%
  4:    337      2.25%  235      1.57%
  5:    255      1.70%  220      1.47%
  6:    200      1.33%  298      1.99%
  7:    199      1.33%  323      2.16%
```

The Global Router divides the system into Global Routing Cells (called GCells) which are usually 10x10 routing tracks. These lines say that 9% of all GCells have one-too-few horizontal routing tracks, 11% of all GCells have one-too-few vertical routing tracks, 4% of all GCells have two-too-few horizontal routing tracks, and so on. This is a rather congested design, but good Detail Routers can handle a lot of congestion, and our Detail Router (called *Nano Route*) is a pretty good one. You can try to run the route step, but watch the violations on each iteration in the run_route.log file to make sure that the step doesn't run forever.

Your task is to try to alter the scripts to allow the xbar design to complete the route step successfully with no violations. You can decrease the density of the design or increase the number of metal layers available. When you are done, turn in the following:

- your **xbar_routed.def** file
- your **run_route.log** file, showing zero violations
- all **Tcl scripts** from your **xbar/pr** directory
- Also include in your solution document a description of the changes that you needed to make to the scripts in order to route the design with no violations.