Exercise 4

Understanding Timing in the Back-end Design Flow

Prof. Dr. H. Kaeslin
Dr. N. Felber

SVN Rev.: 1768
Last Changed: 2016-10-04

Reminder:
With the execution of this training you declare that you understand and accept the regulations about using CAE/CAD software installations at the ETH Zurich. These regulations can be read anytime at [http://eda.ee.ethz.ch/index.php/Regulations](http://eda.ee.ethz.ch/index.php/Regulations).
1 Overview

One of the main parameters of an integrated circuit is the speed at which it is able to operate. The maximum clock rate at which the circuit can operate is usually determined by a so called setup constraint. However there is far more to timing than achieving maximum performance. An ASIC is never designed to work on its own, it needs to be able to properly communicate with other components. This requires that the chip respects the limitations set by these other components. These limitations include when the input signals will be available to the circuit and when the output signals need to be present so that they can be sampled by the connected circuits. Together these two form the input/output timing constraints. Furthermore, synchronous design relies that the hold time constraint is fulfilled for all possible timing paths.

Modern back-end design tools are able to optimize a circuit so that the timing constraints are fulfilled. However, there are no magic solutions. The tools require you to specify these constraints in a proper way, they rely on timing characterization provided through libraries and most importantly, the designer still has to go and verify that the resulting circuit fulfills the constraints. If this is not the case, the designer can choose to use different optimization parameters, or modify the architecture so that the resulting design satisfies all timing conditions.

This exercise is designed to show you:

- How timing analysis in the back-end design flow works
- How the constraints can be specified
- How you can analyze the timing reports
- How the design can be optimized

on a series of examples.

The timing constraint files will be used in the the following exercise 5 where you will start a back-end design flow from scratch. We will continue to use this setup in exercise 8 where you will complete the entire back-end design flow.

1.1 About the Style

We will try to use a number of different styles to identify different types of actions. These are summarized below:

**Student Task:** Parts of the text that have a gray background, like the current paragraph, indicate steps required to complete the exercise.

Actions that require you to select a specific menu fill be shown like the following:

menu→sub-menu→sub-sub-menu

Whenever there is an option or a tab that can be found in the current view/menu we will use a **BUTTON** to indicate such an option.

Throughout the exercise you will be asked to enter certain commands using the command line. The following is an example of the Linux command line.

---

1 There are many reasons for using a command line. Some functionality can not not be accessed through GUI commands, and in some cases, using the command line will be much faster. Most importantly, things you enter on the command line can be converted into a script and executed repeatedly.
Whereas some of the commands will be entered on the command line of the CADENCE INNOVUS tool such as:

```
inn> this command is an Innovus command
```

2 How digital timing is calculated.

In this section we first want to investigate how digital timing is calculated. For this reason we are going to load a small example circuit into CADENCE INNOVUS. If you have trouble with using CADENCE INNOVUS refer to the previous exercise 2.

**Student Task 1:** Start by installing the exercise

```
sh> /home/vlsi2/ex04/install_ex04
sh> cd ex04
sh> icdesign umcL65
```

You can now start CADENCE INNOVUS by selecting the ENCOUNTER/INNOVUS button on the cockpit interface or by directly changing into encounter directory and typing

```
sh> cds_innovus innovus
```

on the command line.

Load the design save/small_example.enc by selecting File→Restore Design and changing the DATA TYPE to ENCOUNTER.

The small example is a very simple circuit as seen in the following figure.
We will start the exercise by investigating the timing involving a particular gate: *i_sbox/U357*.

**Student Task 2:** Using the CADENCE INNOVUS command line type in the following command and answer the following questions

```
inn> report_timing -through i_sbox/U357/Z
```

• What type of standard cell is *i_sbox/U357*?

• How long is the timing path that goes through this cell?

• Where does the path start?

• Where does it end?

• How long does it take to pass through the gate *i_sbox/U357*?

• Is the timing OK, how long was this path supposed to be?

The command `report_timing` is very similar to the command to the one we have used in SYNOPSYSDESIGNCOMPILER, it allows you to specify a beginning point (`-from`) and end point (`-to`) to find the timing information about a specific path.

**Student Task 3:** We want to explore how CADENCE INNOVUS calculates the timing in this example. By using the DESIGN BROWSER

• locate the instance *i_sbox/U357*.

• determine which instances are driven by this gate and draw a simple schematic for these gates. Indicate the name of the instances as well as the name of the cell in your schematic.

As you know from your classes, the delay of a digital gate depends on

1. the current driving capability of the cell.

2. the capacitive load that is driven by the cell.

To switch a load faster, you need either a cell that can drive more current (larger transistors) or a load that is smaller.

**Student Task 4:** Let us find out about the delay parameters from the data-book of the cells which you can access under `./docs/stdcell_low_vt_b03_databook.pdf`. Using your schematic from the previous step;

• Find the input capacitance of the gates that are driven by *i_sbox/U357*.

• Determine the timing by finding the corresponding delay in the delay table.

• Does this match the delay from the report, why?
The information that you found in the data-sheet is tabulated for the tools in a lib file. Both SYNOPSYS DESIGN COMPILER and CADENCE INNOVUS use exactly the same file to determine the timing for individual cells. You can find this file under: 
./encounter/tech/lib/uk65lsclmvbbl_120c25_tc.lib

### Student Task 5:
Let us change the driving strength of our cell. Using the data-sheet find the cell with the strongest output drive. Use this cell name in the command below, then run the same `report_timing` command again and answer the questions.

```
inn > ecoChangeCell -inst i_sbox/U357 -cell NAME_OF_THE_CELL
```

- Did the delay through `i_sbox/U357` change, how much? 
- Did the overall delay of the entire path change, how much? 
- Did the delay of the gate driving `i_sbox/U357` change? 

**Note:** If you are using command line tools, the history shortcuts can be very practical. In most command line interpreters, the `!` is used to call commands from the history. This saves typing effort. You can type the following to execute the last command that started with `re` by typing

```
in > !re
```
In our case this would be the command `report_timing -through i_sbox/U357/Z`.

As mentioned earlier, we can either change the driving strength (which we just did), or reduce the loading capacitance. We will now try to reduce the loading capacitance of our gate `i_sbox/U357`.

### Student Task 6:
First let us reset the situation, you can either reverse the last command, or restart CADENCE INNOVUS and load the design again.

- One of the gates driven by `i_sbox/U357` is quite large. Identify this gate. 
- Using a `ecoChangeCell` command change this gate with an equivalent gate with a driving strength of 2. Use the data-book if necessary. 
- Compare the delay to before, which tactic worked better? 

What we just did was optimizing the delay of one path by resizing one of the cells. It is one of several tricks that can be used to optimize timing during back-end design. Once you use a higher driving strength the cell becomes faster and can drive more load. At the same time, as it requires larger transistors for the increased current, its input capacitance also increases, which in turn makes it more difficult to drive the cell. Optimizing timing is an exercise in balancing the loading capacitance with the input capacitance.

### 2.1 Process, Voltage, Temperature (PVT) variations

You may have noticed that the delay numbers reported so far are very accurate. However the delay can not be calculated as accurately as you think. Several additional factors influence the delay of a circuit. First of all the manufacturing process has a large variation. Operating voltage and temperature also have a large influence on the delays.
At the moment the analysis was made using typical values for the process, voltage was assumed to be nominal and the temperature was taken as room temperature. Let us now see what happens when we change these. For the current design three different corners were defined:

- **Typical** Typical process values, Voltage 1.20 V, Temperature 25 °C
- **Worst** Worst case process values, Voltage 1.08 V, Temperature 125 °C
- **Best** Best case process values, Voltage 1.32 V, Temperature 0 °C

### Student Task 7:

Use the following command to change the conditions to worst case.

```bash
inn> update_analysis_view -name func_view \
inn>     -delay_corner worst_delay \
inn>     -constraint_mode func_mode
inn> set_analysis_view -update_timing \
inn>     -setup { func_view } \
inn>     -hold { func_view }
```

- Repeat the `report_timing`, how much did the overall delay change? ________________
- Adapt the command above so that best case timing is used. How much did the overall delay change? ________________

As you can see the delay can change quite a bit depending on the PVT conditions. The manufacturer actually specifies that a certain percentage of the chips (the yield) will perform above the worst case conditions.

**Note:** If you are designing a circuit that needs to operate at a certain speed, make sure to use worst case timings during setup analysis. This will ensure that the timing is met even if the circuit is operating at its worst condition. If you are working on a research design where you compare the performance of alternative architectures, the choice of the corner is not so important.

### 3 How to specify timing

In the previous example we saw what factors affect timing, but we only examined one of the timing paths (the one that goes through `isbox/U357`) and we actually did not show how the timing was defined for the analysis. In this part we will cover these aspects:

- How to analyze the overall delay of the circuit
- How to specify the timing constraints.

---

2 we will see how these definitions are made in the next sections.
3.1 Four different timing paths

3.1.1 Register-to-register timing

In the previous example the timing path started from an internal register (Reg1 DP) and ended in a second internal register (Reg2 DP). For many circuits, this internal critical path from one register to another register is the most important timing path, as it is directly determined by the internal structure of the circuit that you have designed. In CADENCE INNOVUS this path will be abbreviated as reg2reg. The following diagram shows a circuit that has three parts that each have such paths. In the first part, the reg2reg delay is the combination of the delays of the combinational blocks a and b, the second part consists of c and the third path consists of d and e.

The maximum reg2reg delay in this system will determine the maximum clock frequency of the clock in this system, which in turn will determine the performance of the system. Ideally we would like to balance these delays so that all three delays are equal, which would allow us to run the system at its maximum frequency. In other words, the clock frequency at which we wish to run this system will be the main timing constraint for reg2reg paths.

**Note:** In principle it is the architecture design that determines what is in the blocks a to e. While small scale optimizations such as retiming can be used to move the registers slightly forward and backward, it is the architectural transformations (such as pipelining, unrolling etc.) that determine what needs to be done in between each register. In this exercise we will not consider such architectural transformations which were covered in VLSI I.

3.1.2 Input-to-register and register-to-output timing

A real-life circuit can not exist without an external interface. At some point data has to come in to our circuit from the environment and once it is processed it will have to be sampled by another circuit in the environment. In principle this creates timing paths that are exactly the same as reg2reg paths. The circuits in the environment also use the same clock as our circuits and we can see the system that we had before as the following.

---

Note: For this exercise we will assume that the clock used in the environment is exactly the same as the one used within our design. However, regardless of the clock rate the basic principles will still be the same for synchronous circuits.
Actually, nothing has changed in this picture when compared to the case before. We still have three paths that go from one register to another one. However, now only the combinational blocks \(b\), \(c\) and \(d\) belong to our circuit, the other two blocks \(a\) and \(e\) as well as the two registers in gray represent the environment.

Remember that the timing constraints determine the timing for our circuit, we can not influence the environment in any way. The timing parameters of the environment are given, and we have to make sure that our circuit complies with them, i.e. when it is connected to the environment it works with the given timing.

For the inputs, what will be given is the input delay which corresponds to the delay of the combinational block \(a\). This will constrain the path that starts at the inputs and ends at the first register. These paths will be called in2reg in CADELINE INNOVUS.

Similarly for the outputs, the timing constraints will specify what the output delay is. This delay will constrain the path that starts from the last register and goes through the combinational block \(d\) to reach the output. CADELINE INNOVUS will call these paths reg2out.

Note: Although most physical interpretations of the input output delays make only sense when these are given as positive numbers, it is actually possible that they have negative values as well. An IC-tester can apply inputs and sample outputs with arbitrary delays, so if you design a chip that will only work on the IC tester you could assign values with negative values as well.

Note: The in2reg delay of your circuit will end up being the output delay of another circuit, and the reg2out delay of your circuit will end up being the input delay of another circuit. A friendly circuit would want to put as little constraints as possible to other circuits, this is why most circuits would sample the inputs as soon as possible (eliminate \(b\)), and add a register to the outputs (eliminate \(d\)).

\[\text{Note:} \quad \text{plus the propagation delay of the flip-flop } t_{pd,ff}.\]
3.1.3 Input-to-output timing

It is also possible that there is a direct combinational path connecting the input of your circuit directly to one of the outputs. CADENCE INNOVUS labels such paths as \textit{in2out}. The following picture shows such an additional path going through the combinational block $f$.

Such \textit{in2out} delays are actually highly undesirable, as these paths have to satisfy both the \textit{input delay} and the \textit{output delay} in addition to the delay of the combinational block and still finish within a clock period. However there are plenty of cases where such paths appear unintentionally.

3.2 Performing timing analysis

Now that we know what different timing paths are, and how timing is calculated let us perform a timing analysis of the circuit that considers not only a single path but all the paths together.

\textbf{Student Task 8:} Let us first reset the circuit to its initial conditions. Please exit CADENCE INNOVUS and restart the program and load \texttt{./encounter/save/small_example.enc} again. Now let us perform a standard timing analysis by typing the following command

\begin{verbatim}
in> timeDesign -preCTS -outDir reports/timing
\end{verbatim}
You should see the following report on the command line (the actual numbers may differ slightly)

<table>
<thead>
<tr>
<th>Setup mode</th>
<th>all</th>
<th>reg2reg</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>WNS (ns):</td>
<td>-0.548</td>
<td>-0.548</td>
<td>0.176</td>
</tr>
<tr>
<td>TNS (ns):</td>
<td>-3.874</td>
<td>-3.874</td>
<td>0.000</td>
</tr>
<tr>
<td>Violating Paths:</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>All Paths:</td>
<td>113</td>
<td>47</td>
<td>66</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DRVs</th>
<th>Real</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nr nets(terms)</td>
<td>Worst Vio</td>
<td>Nr nets(terms)</td>
</tr>
<tr>
<td>max_cap</td>
<td>0 (0)</td>
<td>0.000</td>
</tr>
<tr>
<td>max_tran</td>
<td>0 (0)</td>
<td>0.000</td>
</tr>
<tr>
<td>max_fanout</td>
<td>0 (0)</td>
<td>0</td>
</tr>
<tr>
<td>max_length</td>
<td>0 (0)</td>
<td>0</td>
</tr>
</tbody>
</table>

Density: 28.773%
Routing Overflow: 0.00% H and 0.00% V

The default column summarizes all the paths groups which are not listed. In order to enable the visualization of the paths groups, you should use the command setOptMode -backwardCompatibleModePathGroups true before launching the command timeDesign.

The new report will show:

<table>
<thead>
<tr>
<th>Setup mode</th>
<th>all</th>
<th>reg2reg</th>
<th>in2reg</th>
<th>reg2out</th>
<th>in2out</th>
<th>clkgate</th>
</tr>
</thead>
<tbody>
<tr>
<td>WNS (ns):</td>
<td>-0.548</td>
<td>-0.548</td>
<td>0.176</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TNS (ns):</td>
<td>-3.874</td>
<td>-3.874</td>
<td>0.000</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Violating Paths:</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All Paths:</td>
<td>113</td>
<td>47</td>
<td>66</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DRVs</th>
<th>Real</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nr nets(terms)</td>
<td>Worst Vio</td>
<td>Nr nets(terms)</td>
</tr>
<tr>
<td>max_cap</td>
<td>0 (0)</td>
<td>0.000</td>
</tr>
<tr>
<td>max_tran</td>
<td>0 (0)</td>
<td>0.000</td>
</tr>
<tr>
<td>max_fanout</td>
<td>0 (0)</td>
<td>0</td>
</tr>
<tr>
<td>max_length</td>
<td>0 (0)</td>
<td>0</td>
</tr>
</tbody>
</table>

Density: 28.773%
Routing Overflow: 0.00% H and 0.00% V
The summary gives a very good overview of the timing of the current design. Some explanations:

- The analysis was run in setup mode, i.e. setup time checks were performed. For a complete analysis we will also have to consider hold timing checks, we will learn about hold timing checks towards the end of the exercise.
- The columns contain numbers for all paths in the design (ALL) or for specific path groups, e.g. reg2reg for all register to register paths.
- Worst negative slack (WNS) reports the slack for the most critical path. Negative numbers mean that the constraints are violated by this value.
- Total negative slack (TNS) is the sum of WNS for all violating paths. Together with the number of violating paths this figure helps to see how severe the violations are.
- Real/Total DRV show (electrical) Design Rule Violations. These are electrical constraints specified in a library that sets constraints that have to be fulfilled regardless of the timing of the circuit. These include, the maximum capacitance that an output is driving ($\text{max\_cap}$), the maximum time a signal is allowed to change from 1-to-0 or 0-to-1 ($\text{max\_tran}$), the maximum number of inputs driven by an output ($\text{max\_fanout}$) and the maximum length of the interconnect driven by one output ($\text{max\_length}$). Since our design is very small, there are no DRV violations in this design.
- Density and Routing Overflow show the placement utilization and routing resources, i.e. are a measure for the feasibility of the current floorplan/placement.

You will have noticed the -preCTS parameter in the command line. This instructs the timing engine to calculate the delays in the pre-Clock Tree Synthesis mode which corresponds to the back-end design mode that the present circuit is in. We will see different modes of the back-end design modes later in the exercise.

The command that we have used calculated the timing and has generated reports under the `/encounter/reports/timing` directory as instructed by the -outDir option. If you go to this directory you will see that several files have been generated. These are:

- *.summary: contains the summary file displayed on the console
- *.clkgate.tarpt: detailed reports for timing paths that end at a clock gating element.
- *.all.tarpt: detailed reports for the paths with the critical timing regardless of the type of path.

Each of these reports contain a list of timing paths as specified by an optional -numPath parameters. By default 50 most critical paths are shown. It is important to be able to find, analyze and understand these reports.

<table>
<thead>
<tr>
<th>Student Task 9: Examine the timing reports and answer the following questions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• What is the critical path of this circuit? ________________________________</td>
</tr>
<tr>
<td>• As it stands now, what is the fastest clock rate where this circuit would work properly? ___</td>
</tr>
<tr>
<td>• Why aren’t there any paths in the reg2out category? __________________________</td>
</tr>
</tbody>
</table>

11
Note: This exercise explains most commands using command line options, as these are the main underlying commands and it is important to understand them. The CADENCE INNOVUS GUI provides interfaces for some of these commands. For timing analysis you can use Timing→Report Timing. There is also the Timing→Debug Timing option which can help you with a graphical interpretation of the timing reports. However, all these commands are actually based on the command line tools we have just described and will not be covered in detail during this exercise.

3.3 Timing constraints

Until now we saw how we can analyze the timing of a circuit with given timing constraints. If we want to have the correct timing, we need to know how to tell the system about the timing. The timing constraint file contains definitions for:

- The clocks used in the system
- The input and output timing constraints
- Information about the signals driving the inputs and the capacitive output load
- Timing exceptions, constants, paths that should not be part of the analysis etc.

For the most part, CADENCE INNOVUS uses the same constructs for these specifications as SYNOPSIS DESIGN COMPILER, so you should be familiar with most of them from the earlier exercises of VLSI I. You can use the command line of CADENCE INNOVUS to get brief help about the timing constraint specification commands. using the command man, for example:

inn> man set_load

For the UMC 65 nm technology we use in this exercise, timing unit is ‘ns’ and capacitance unit is ‘pF’
For this exercise we will be using the following constraints:

- `create_clock`
- `set_input_transition`
- `set_output_delay`
- `set_output_delay`
- `set_load`

And we will make use of the following statements to build our expressions:

- `all_inputs`
- `all_outputs`
- `get_ports`
- `remove_from_collection`

Use the `man` command to get information on how to use these commands if necessary.

As we will see in a while, the timing setup we are using is a bit more complicated and involves more than one file. But the constraints that were used in our analyzes until now are located in a single file ./encounter/src/small_example_functional.sdc. For the moment let us concentrate on this one file, and we will soon learn about the rest of the story as well. At the moment there is not much defined in this file. Let us modify the constraints of our example circuit so that there are at least some constraints for the inputs and outputs.

**Student Task 10:** Open the timing constraint file that we use for this example ./encounter/src/small_example_functional.sdc using a text editor and modify the constraint file to add the following constraints:

- Define the clock period to be 720 ps.
- Add a constraint that sets the input transition time to 150 ps.
- Add an output load of 20 fF to all outputs.
- Add an input delay of 100ps to all inputs (except the clock) and an output delay of 200 ps to all outputs.
- Under the given constraints what will be the maximum allowable
  - `in2reg` delay ?
  - `reg2reg` delay ?
  - `reg2out` delay ?
  - `in2out` delay ?

for our circuit.

Save your file and read the constraint file by typing the following commands in the CADENCE INNOVUS shell:

```sh
inn> update_constraint_mode -name func_mode \\
in> -sdc_files [list src/small_example_functional.sdc]
in> set_analysis_view -update_timing
```

Make sure that there are no errors in the `sdc` file. If CADENCE INNOVUS reports errors, repeat the above process until there are no more errors.
Note that just changing the timing constraints does not change the circuit at all. The netlist stays the same, but the timing analysis reports will now be different as they will take into account the new constraints you have provided.

**Student Task 11:** Let us perform a timing analysis and see what the result of the timing will be and answer the following questions. The report files will be located under `/encounter/reports/timing`

```bash
inn> timeDesign -preCTS -outDir reports/timing
```

- Where is the critical path of the circuit? ________________________________
- Do you think it is realistic to expect that after optimization the timing as specified could be met? ________________________________

### 3.4 Optimization

As you can see in the previous example, the timing that we desired could not be met. This is perhaps not what you have expected, as most probably the timing of the circuit was OK after synthesis with SYNOPSYS DESIGN COMPILER. Problem is that the netlist generated by SYNOPSYS DESIGN COMPILER was only able to make an *estimate* of the parasitic capacitance due to interconnections. As we progress in the back-end design we will get more and more accurate numbers for this parasitic capacitance. If this parasitic load is different than what was initially expected, the timing will be off, sometimes significantly.

CADENCE INNOVUS can try to optimize your netlist so that the timing can be met. To do this, it will resort to resizing gates, just like we did in the first part of the exercise. In addition, it can add and delete buffers, replicate logic, and even re-synthesize parts of the critical path. Let us see if this will solve our problems.

**Student Task 12:** Run the optimization using the following command which will also run a timing analysis directly afterwards. Answer the questions by examining these reports

```bash
inn> optDesign -preCTS -outDir reports/timing
```

- Was the timing met? ________________________________
- Where does the critical path go through? ________________________________
- Is there a way to solve this? ________________________________

### 3.5 Different modes of operation

Most modern chips have functions that are designed to work under radically different circumstances. Most chips would have a *test* or *debug* mode that will only be used in controlled environments and will not be used during normal operation. It can be that these additional operational modes have different timing constraints that interfere with the *normal* mode of operation. In this case, we can tell CADENCE INNOVUS how to constrain the circuit in a specific mode. We can set inputs (or even internal nets) to a specific value using the `set_case_analysis` command, or even disable certain timing paths using `set_false_path`. 
In our case, if you look at the block diagram, you will notice that we have implemented a test mode that copies the output of the RAM block. This mode is activated by the Test_TI signal. By default CADENCE INNOVUS will look at all possible combinations of input signals when deciding what the critical paths in the design are. We can influence this behavior by adding specific timing constraints.

**Student Task 13:** Keep the design you have just optimized, and modify the timing constraints in the file `/encounter/src/small_example_functional.sdc` and add a constraint that makes sure that Test_TI remains at '0' during the timing analysis. Reload the timing constraints using the following commands and perform a timing analysis.

```
inn> update_constraint_mode -name func_mode \\
inn> -sdc_files [list src/small_example_functional.sdc]
inn> set_analysis_view -update_timing
inn>
inn> timeDesign -preCTS -outDir reports/timing
```

What effect did the constraint have? __________

**Note:** As you see we are repeating similar commands over and over. We hope you have already started writing a small script that you can source or have a cheat sheet file where you can copy the most frequently used commands during your design.

At this point, you might be happy about the results, however, we have now ignored the timing when the Test_TI signal is active. What we can do is to specify a different set of timing constraints for this mode of operation, and verify the timing for this second mode as well.

**Student Task 14:** Create a new constraint file called `/encounter/src/small_example_test.sdc` and add the following constraints:

- input and output delay of 1 ns
- clock period of 2.5 ns
- output load of 20 fF
- input transition time of 100 ps
- Set the Test_TI input to '1'

Load the new timing constraints using the following commands and perform a timing analysis.

```
in> update_constraint_mode -name func_mode \\
in> -sdc_files [list src/small_example_test.sdc]
in> set_analysis_view -update_timing
inin> timeDesign -preCTS -outDir reports/timing
```

Does this circuit fulfill the timing constraints __________

**Note:** Please be careful with how you specify constraints, especially `set_false_path` and `set_case_analysis` for certain views. These commands will disable certain timing paths and...
as a result CADENCE INNOVUS will not consider these paths during analysis. If there are actually problems with these paths (i.e. a hold violation) you will not detect them. Do not automatically assume that you should set all test related signals to constant ‘1’ just because we set a signal called Test_TI to ‘1’ in this particular exercise.

The optimization phase uses the constraints to determine which paths to optimize and how. There are two important aspects of this:

- Optimization algorithms used in a back-end design tool like CADENCE INNOVUS are heuristic. They are designed to reduce the overall WNS (Worst Negative Slack) as much as possible. Poorly constructed constraints may result in the optimizer trying to reduce the timing of a path that can not be properly optimized, while neglecting real paths that could actually be optimized. This was more or less the case during our first optimization.

- If the circuit has different modes of operation, by default the circuit will be optimized for only one mode of operation.

We have to be aware of what constraints we are assigning, we have seen that people simply copy the constraints from the example files we use in these exercises for their designs without really considering their own. It is really important that you understand the environment in which your circuit is supposed to work, and provide the appropriate constraints for the tools. This is important for the first aspect we have mentioned above. As for the second aspect, fortunately CADENCE INNOVUS provides some help in the form of Multi-Mode Multi-Corner (MMMC) Analysis as we will see in the next section.

4 Multi-Mode Multi-Corner (MMMC) Analysis

So far we have seen that the timing can significantly vary according to the

- process corner (i.e. typical, worst, best case)
- operational mode (i.e. test, functional)

If you use a simple timing setup, you are forced to make a decision on the corner and the mode and perform your optimization actions for a given set of constraints. The MMMC analysis is designed to help in this regard. It allows you to define a set of operating modes, and a set of corners, and will allow you to optimize and analyze the circuit using a set of conditions.

The price that you have to pay for this is that we will need a slightly more complex setup than a simple sdc file to achieve the definitions. Until now we asked you to enter some cryptic commands to update the timing constraints (using the update_constraint_mode command). Now it is time to learn how the setup is made.

---

6 As you have seen until now, you can still perform timing analysis for different corner and mode combinations, however, you can not optimize the circuit, for different corners and modes
4.1 MMMC configuration

The MMMC configuration consists of several separate definitions:

- **library set**
  contains the set of standard cell libraries which are characterized for a specific operating point.

- **rc corner**
  The parasitic capacitances and resistances are also affected by process variations. Therefore you can specify look-up tables for different operational corners that are used to extract the parasitic capacitances.

- **delay corner**
  The delay calculation combines an rc corner with a library set, as the delay of a circuit is a combination of both the delay of the cells, and the delay of the interconnect due to parasitic capacitances and resistances.

- **constraint mode**
  The constraint mode defines different modes of operation and attaches a set of constraints in sdc files to each mode. This allows you to put the general constraints (for example the output load) that are common for all operational modes in a separate file and to share it with other constraint modes.

- **analysis view**
  Finally the analysis view combines the delay corner with the constraint mode. These views are then used by CADENCE INNOVUS for optimization and timing analysis.

This configuration gives us a lot of flexibility in defining corners and operation modes. However, for simple designs where you are only interested in a single corner and operation mode, all the setup has still be constructed the same way. By default the cockpit setup provides you a sample MMMC configuration file under ./encounter/src/sample/mmmc.view.tcl and several associated sdc files. You should copy this file to the ./encounter/src directory and adapt it to your design. The particular MMMC control file that will be used in the setup is specified by the following line in the *.globals file that you use for initializing the design.

```tcl
set init_mmmc_file {./src/small_example_mmmc.tcl}
```

In this exercise, we want you to modify the MMMC setup so that we have three analysis views

- **func_view**: standard functional mode, using typical delay corner, has Test_TI set to '0' and the associated constraints that we already defined in ./encounter/src/small_example_functional.sdc

- **test_view**: An operational mode using typical delay corner and constraints that are used when the Test_TI is set to '1' as we already defined in ./encounter/src/small_example_test.sdc

- **hold_view**: Same constraints as the test_view but with a best case delay corner.

Normally we would just modify the initial MMMC file, in our case ./encounter/src/small_example_mmmc.tcl and re-run the design script with the new constraints. Since we are using a design that has been saved, the library set, rc corner and delay corner etc. are already defined and loaded. Therefore we will use the appropriate update_* commands to change the content of the configuration that already exists and add new configurations by using appropriate create_* commands.

---

7 We will cover this in more detail during the next exercise
Student Task 15: Let us reset the *constraint mode* that we had for the functional mode.

```bash
inn> update_constraint_mode -name func_mode \\
inn>     -sdc_files [list src/small_example_functional.sdc ]
```

Now let us define a new *constraint mode* for the test mode

```bash
inn> create_constraint_mode -name test_mode \\
inn>     -sdc_files [list src/small_example_test.sdc ]
```

We want to create two additional analysis modes a *test_view* that uses the *test_mode* with *typical_delay* and a *hold_view* that uses the same *test_mode* with *best_delay*. By using `create_analysis_view` commands create these two views. Look into `/encounter/src/small_example_mmmc.tcl` if you need inspiration. As the last point we want to define which *analysis views* will be used during the timing analysis. Add the following command

```bash
inn> set_analysis_view -update_timing \\
inn>     -setup { func_view test_view} \\
inn>     -hold { hold_view}
```

Now we have completed our MMMC setup. Note that, for your own design, you would directly modify the `/encounter/src/small_example_mmmc.tcl` definition and re-run the design flow, instead of using `update_*` series of commands that we have used in this exercise.

Student Task 16: Now that we have a complete MMMC setup, let us put it into action, run the timing analysis command with the additional `-expandedViews` option, that will generate reports for each setup *analysis view* under a separate directory within the `/encounter/reports/\ timing` directory.

```bash
inn> timeDesign -preCTS -expandedViews -outDir reports/timing
```

In order to dump also the *hold view* reports, use the following command.

```bash
inn> timeDesign -preCTS -hold -outDir reports/timing
```

Examine the reports and make sure that constraints for all modes are set properly, and these constraints are met.

5 Design Phases

Until now, we have always talked that the design is in the `-preCTS` phase. We will now investigate what the different design phases are, and what effect they have on the timing of our circuit. The back-end design mainly consists of two actions: placement and routing. Both actions are computationally

---

8 We will cover these steps in detail during exercise 8.
intensive and there is no algorithm that can present the perfect solution for these problems. As a result, heuristic algorithms are used for both placement and routing that are known to have a good chance of finding results with acceptable quality. This acceptable quality is defined by the constraints we provide the design tools.

5.1 preCTS

Initially we start the design with a netlist that we have imported from SYNOPSYS DESIGN COMPILER. The next step is preparing a floorplan, and then we can start with placement. This will position the individual standard cells that comprise the netlist on the area defined by the floorplan. How these cells are placed will have an effect on our timing. In a modern design technology, such as the 65 nm technology used in this exercise, the interconnection between the cells can account for significant delay due to parasitic capacitances. From a timing perspective, cells that are placed closer together will result in shorter delays. The goal of the placement stage is to make sure that the cells are placed within the area restrictions imposed by the floorplan and fulfill the timing constraints that you have specified. In CADENCE INNOVUS this stage of the design is called preCTS (pre Clock Tree Synthesis), in reference to the next design phase.

As mentioned earlier, the parasitic interconnects can have a significant impact on the timing of the circuit, but at this time of the design the routing has not yet been completed, so how are the delays due to parasitic interconnects calculated. We use two tricks:

5.1.1 Trial route

Routing is a time consuming and difficult task. However, just for the purpose of estimating the parasitic capacitance, a simpler and much faster routing can be used. In CADENCE INNOVUS this is called trial route.

Student Task 17: Let us investigate how the trial routing looks like. The design that you have currently loaded uses this type of routing. Examine the connections closely, What kind of errors do you see?

5.1.2 Simplified timing for high fan-out nets

There are certain nets in a typical circuit that will reach many nodes. The clock signal for example should be routed to the clock pins of all the flip-flops and latches that are triggered by this clock. In a typical design this could result in thousands of connections. Similarly there are several control signals such as reset, scan enable that also fan-out to a high number of outputs. Calculating the timing of these nets in the same fashion as normal nets is not very efficient. The main problem is that on such a large net, the signal will not arrive to all pins at the same time, a balanced distribution tree is necessary for such a distribution, and at this stage of the design, this is not there yet. So CADENCE INNOVUS by default would use special tricks to make it easier to calculate the timing:

• All clocks will be considered ideal, and there will not be any delay in distributing the clock signal. As a result all clock pins will receive the clock signal at exactly the same time. This will not be the case later on, but for a start it will help us with the flow.

---

9 topic of the following exercise 5.
10 These restrictions also includes position of large macro cells and connections to outside.
There is a set limit of fan-out. Once this limit is exceeded, CADENCE INNOVUS does not compute proper timing for this particular net, and assumes a fixed amount of delay. By default this fan-out number is 1000 and the delay used for nets that exceed this fan-out number is 1.0 ns. You can use `delaycal_use_default_delay_limit` and `delaycal_default_net_delay` variables to change these values. If you want to find out which nets have a fan-out higher than for example 10, you can use the following command `report_net -min_fanout 10`.

5.2 postCTS

Once the cells are all placed, and the timing has been optimized for this placement, it is time to deal with special high fan-out nets (such as Reset and Scan Enable) and the clock tree that will be used to distribute the clock signal. This is the clock tree synthesis phase. As soon as the clock tree is synthesized, CADENCE INNOVUS considers the design to be in the postCTS phase. In this phase the clock is no longer ideal:

- The clock distribution network introduces a certain clock insertion delay. The clock signal does not immediately arrive to all the pins, but there is some delay.
- Not all pins get the clock signal at exactly the same time, there is some clock skew.

5.2.1 Changes to input and output timing

Once the clock tree is inserted, the timing will change slightly again. Due to the clock insertion delay $t_{di}$ the internal clock will be slightly offset when compared to the external clock. At the input, the data traveling towards the first flip-flop inside the chip, will have more time, since this flip-flop will be triggered by a clock signal that has been delayed by $t_{di}$. At the output however, the data that is coming from the chip will be launched with the internal clock, but will have to be sampled by the external clock. Consequently there will be less time for this signal. This is illustrated in the following figure.

---

11 \textsuperscript{11} We will cover this in exercise 8
Note that, although for clarity in the figure the clock insertion delay is shown to be much smaller than the overall clock period, for practical circuits, this may not necessarily be true. Clock insertion delay is usually in the order of 1-2 ns, any circuit that is internally clocked at more than 500 MHz (which means a 2 ns period) will be significantly affected by this delay.

5.2.2 The long story about clock timing

There is a marked difference in the timing between preCTS and postCTS due to the introduction of the clock tree. Both SYNOPSYS DESIGN COMPILER and CADENCE INNOVUS accept several parameters to deal with this problem automatically. In the following we will discuss on how CADENCE INNOVUS calculates delays in the presence and absence of clock tree. The following table summarizes the most important settings:

<table>
<thead>
<tr>
<th>Timing analysis mode (setAnalysisMode)</th>
<th>Clock propagation mode (set_propagated_clock)</th>
<th>Clock latency (set_clock_latency)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-noSkew</td>
<td>forced ideal</td>
<td>no effect</td>
</tr>
<tr>
<td>-skew -noClockTree</td>
<td>forced ideal</td>
<td>SDCs in effect</td>
</tr>
<tr>
<td>-skew -clockTree</td>
<td>SDCs in effect</td>
<td>SDCs in effect</td>
</tr>
</tbody>
</table>

The timing analysis mode is automatically updated by CADENCE INNOVUS to match the design stage, i.e. before clock tree insertion it is set to '-skew -noClockTree' and afterwards to '-skew -ClockTree'. The analysis mode can also be changed manually with the setAnalysisMode command.

The two Synopsys design constraints (SDC) set_propagated_clock and set_clock_latency are usually specified by the designer in the sdc file. Furthermore, CTS tries to add a set_propagated_clock constraint on-the-fly (in memory). Since this will not be written back to your sdc file this can cause inconsistencies during reload.

Now, as can be seen from the table above, to get the actual timing of the buffers/inverters on the clock tree instead of ideal mode, setting both '-skew -ClockTree' and set_propagated_clock is required. Also note that set_propagated_clock gets overridden for all pre-CTS design stages and could therefore be set right from the start.

In ideal mode, the clock tree insertion delay is zero unless the set_clock_latency command is used to specify a different number, preferably close to the delay of the real tree (that is still to be inserted). While this placeholder delay has the advantage that the I/O timing doesn’t change between pre-CTS and post-CTS phases, it renders timing reports more difficult to understand and is not handled exactly the same way across different tools. Therefore, do not use this command unless you know what you are doing.

In conclusion, it is recommended to include set_propagated_clock right from the start, not use set_clock_latency and load modified timing constraints after CTS only if required, i.e. when the I/O timing numbers (set_input_delay, set_output_delay) need to be adjusted to account for the actual clock tree.

12 For slower clock speeds and/or uncritical I/O timing this is often not required.
5.3 postRoute

Even after the clock tree has been inserted, the parasitic effects of the routing are still estimated by the fast trial route. The last phase of the design is when the circuit is completely routed. At this stage, called post route, the exact delays due to routing can be calculated directly. This is then the most accurate estimation of the delay. On the down side, the optimizations have much more work to do. Moving or changing a cell means that all its connections have to be re-routed making such changes costlier and slower.

5.4 signOff

In tech-speak sign off refers to the last stage of the timing checks, when we are completely finished with the design, and are ready to send the chip to fabrication. This would require more accurate estimation of the parasitics (that will improve the overall timing accuracy) and possibly verifying the timing for a larger number of corner cases then what we were doing until now. For simple semester thesis projects, our experiments show very little difference between the timing obtained between post route and sign off. So we will skip this step in this exercise.

5.5 When to optimize

You can optimize the design at all phases: before placing (prePlace), after placement (preCTS), after clock tree synthesis (postCTS), and after routing (postRoute). Or in the extreme case you can defer all optimizations until after the routing and then run a single optimization step. There is unfortunately not a single solution to this question, but you will have to see what works best for you for each individual design. In principle

- The further along you are in the design phases, the more accurate the timing information, and therefore the more effective the optimization.

- The further along you are in the design phase, the more difficult it is to make changes to the design, which increases the amount of time the tool has to spend during optimization.

- Once the clock tree is placed, the cells associated with it will be fixed, making it impossible to move or change them. This usually also ties all the flip-flops to a certain position. If the placement prior to the clock tree insertion is not very good (i.e. not optimized), it can not be improved further after clock tree insertion.

- We suggest that you first do a quick and dirty complete design flow for your circuits, to get a feeling of how the design behaves throughout the design flow. During this run, try to see if optimization brings something, but if it fails, continue to the next stage anyway. Sometimes a design that can not be properly optimized at an early phase will meet its timing in the postRoute phase.\footnote{This sometimes happens because the trial routing at the earlier stages delivers an inaccurate parasitic estimation for the circuit.}

- We suggest you to consult the timing reports after each phase, and run an optimization if there are significant violations (i.e. more than 5% of the constraint). If necessary you can also run the optimization more than once.
• Some designs will not be able to meet the timing with optimization tricks, you may have to generate the netlist with different constraints or adapt the architecture in the RTL description you have.

6 Hold Time

Until now for timing analyzes we have only considered the setup constraints. As you know the setup constraint determines the slowest timing path in the circuit and also determines the operating frequency. However, the hold constraints are just as important if not even more important than the setup constraints.

Student Task 18: Just to refresh your memory, state what you would do to fix the timing problems of a circuit running with 100 MHz that has a

- Setup violation of 0.5 ns
- Hold violation of 0.2 ns

Once identified the hold violations can be easily fixed. However, in most cases, you will have hold violations due to inadequate input and output timing. In this case the optimizer will insert unnecessary delays into your circuit, increasing your area, power consumption and delay unnecessarily. Therefore the most important part of fixing hold violations is to making sure that the input and output constraints that you have specified are correct.

Contrary to the setup timing, in the hold timing, the problems are the fast paths. This is why, the worst case analysis for a hold path would generally use the fast delay corner. Fortunately we added just one such analysis corner in the MMC setup, and we will be using this analysis view for the hold analysis mode. The hold timing analysis and optimization only makes sense after the clock tree has been synthesized, as the clock skew is one of the main reasons for hold violations.

Student Task 19: Restart CADENCE INNOVUS and load the design ./encounter/save/clock_tree\enc. This is the same design that we have covered so far, with the clock tree added. First run a standard (setup) timing analysis and make sure that everything is OK.

```
inn> timedesign -postCTS -outDir reports/timing
```

Now it is time to run a hold time analysis, we do this by adding the -hold parameter to the timing analysis command.

```
inn> timedesign -postCTS -hold -outDir reports/timing
```

Examine the results, are there any hold violations?

Sketch a timing graph showing where the hold violation starts, which gates it follows through and where it ends. Add the timing information to your sketch to illustrate the problem and then propose a solution.

Show your sketch to an assistant before continuing!
**Note:** When constructing input and output delay constraints, instead of using a fixed delay value, you may want to define a window in which you define a minimum delay and a maximum delay. The maximum delay will then be used for setup checks and the minimum delay will be used for hold checks. For example:

```plaintext
set_input_delay 0.1 -min -clock Clk_CI [all_inputs]
set_input_delay 0.5 -max -clock Clk_CI [all_inputs]
```

**Student Task 20:** Let us now optimize the design for hold violations, do not forget to add `-hold` to the command line.

```
in> optDesign -postCTS -hold -outDir reports/timing
```

Examine the results, and make sure that you do not have any hold and setup violations.

The hold optimization will automatically report both setup and hold timing at the same time. When fixing hold violations, make sure that these are real problems, and are not caused by carelessly defined input and output constraints. In the past, a very high percentage of the designs that were completed by the students had unnecessary delay elements inserted to fix supposed hold violations that were not really there.

## 7 Effect of I/O pads

The example we have used throughout this exercise is a small block rather than a complete chip which can be manufactured on its own. The difference between the two lies in the I/O drivers and bonding pads that are needed to interface with the outside environment as you have learned in the previous exercise 3. In the figure below you can see the same diagram that we have been using throughout this exercise with the additional I/O drivers and bonding pads added.
As you can see, these structures introduce combinational delay and need to be taken into account when determining the input and output timing. You may recall from exercise 3 that such drivers (especially the output drivers) can easily add 1 ns to 2 ns or more combinational delay. As mentioned earlier, combinational input to output paths through a chip will have to go through both an input and an output driver which in most practical cases will exceed 3 ns.

**Note:** Make sure to not accidentally use constraints that were written for the core level (chip without pads) at the chip level (with pads) and vice versa. The pads affect the I/O timing quite a bit and the drive capabilities of a standard cell and an output pad are entirely different, i.e. set_load needs to be very different in both cases. Typical core level loads are in the range of tens of fF, while external loads are typically between 5-20 pF.

### Student Task 21: Restart CADECE INNOVUS and load the design ./encounter/save/example_chip/enc. This is the same design that we have covered so far, completely routed and with the I/O drivers and bond pads. Note that we have to use the option -postRoute during the timing analysis. If you want to have detailed reports, add -expandedViews to the command.

```
inn> timedesign -postRoute -outDir reports/timing
```

Examine the results, describe the type of problems we have?

---

### 7.1 Dealing with large output delay

The lower speed of the I/O drivers mainly comes from the difficulty in driving large external capacitances. While the capacitance being driven internally decreases with newer technology nodes, this is not necessarily the same for external loads which are due to packaging, and routing on the PCB. As these parameters do not scale the same way, the discrepancy between the external delays and internal delays keep on increasing. For the 65 nm technology that we use in this example, many circuits will be able to work with a clock period that is much lower that the propagation delay of an output pad.

Apart from lowering your clock speed to match the I/O constraints, you have a couple of options. In many cases you will not require one output every clock cycle. For example you may have an internal clock running at 1 GHz, but sample the output every fourth cycle with a 250 MHz clock. Such timing paths are called **multicycle** paths and can be specified with the set_multicycle_path.

However if you need to sample the output at every clock cycle, then this method will not work. What you can do in this case is to subtract multiples of the clock period from the output delay constraint. Note that you can easily specify negative numbers for your output delay. Taking the same example as before, if you want to sample the output of a system running at 1 GHz with a latency of three cycles you can subtract 3 ns (3·$T_{clk}$) from the original output delay you have specified.

Larger SoCs most often use a PLL to generate the internal clock from an external reference. In this case, it is possible to compensate for the I/O driver delay and clock insertion delay by using an appropriately delayed clock signal as the feedback signal for the PLL. However such systems would

---

14 For example we will use the same package that we have used for 180 nm designs also for our 65 nm designs, for someone interfacing with these chips, they will have very similar input capacitances.

15 In most cases this is not smart, your circuit will just work much slower than it could. However, sometimes you can favor an alternative architecture that has a longer critical path.
still use a system what effectively is a multicycle path as the clock for the I/O interface is often a fraction of the main clock which runs internally.

**Student Task 22:** Adapt one of the solutions described above and get the design to pass the timing analysis without errors.

## 8 Final Words

We have spent a rather long time describing the timing constraints and how they should be specified in this exercise. Our experience is that, although the theory is relatively simple, most students have difficulties with these concepts in practice. Please make sure that you understand these concepts fully.

In this exercise we have almost exclusively used command line arguments of CADENCE INNOVUS. As you have seen many aspects of the design are repeated over and over again. Command line arguments can be easily copied to a text file and can also be converted to batch scripts. This will also serve as a documentation of what steps you have used in the future. Most of the commands we have used can also be activated through various GUI elements. Some of these elements have practical features that can help you (DEBUG TIMING), some are one to one interfaces for the corresponding command (REPORT TIMING), while some we believe are not extremely practical (CONFIGURE MMMC). It is important to understand the principles and you are welcome to investigate the GUI elements on your own.