Physical planning of ASIC's in mobile systems

Master thesis performed in

Electronic Systems

by

Håkan Roos

LiTH-ISY-EX--07/3926--SE

Linköping 2007
Physical planning of ASIC's in mobile systems

Master thesis performed in Electronic Systems at Linköpings tekniska högskola by

Håkan Roos

LiTH-ISY-EX--07/3926--SE

Supervisors: Björn Forsberg, Ericsson AB

Jens Andersson, Ericsson AB

Prof. Mark Vesterbacka, ISY, Linköpings tekniska högskola

Examinator: Prof. Mark Vesterbacka, ISY, Linköpings tekniska högskola

Linköping, February 5, 2007
**Sammanfattning**

With increasing demands in terms of timing, area and power, today’s ASIC (Application Specific Integrated Circuit) designers are faced with new problems as technology emerges. Ericsson has started to work in 65 nm and realized that the methods used in previous, larger technologies, does not offer good enough correlation between synthesis and the results after physical placement. This leads to several expensive and time consuming iterations back and forth between Ericsson and the ASIC vendor.

In order to narrow the gap between Ericsson and the ASIC vendor, and hence increase correlation, physical planning has been identified as a possible solution. Cadence First Encounter, part of the Cadence Encounter digital IC design platform, is an advanced tool for silicon virtual prototyping. The tool basically brings back-end placement knowledge to front-end ASIC designers.

This master’s thesis main goal is to evaluate Cadence First Encounter and investigate how it could be integrated with Ericsson's design flow. The tool has been tested on previous designs with known issues and the results are positive. By using the prototype work flow in First Encounter that is described in this report, it is possible to identify and correct issues with the netlists in time, which will help shortening the lead time in projects and hence also the time to market.

**Nyckelord**

ASIC, Floorplanning, First Encounter, Prototyping, Back-end design, Chip layout
Abstract

With increasing demands in terms of timing, area and power, today’s ASIC (Application Specific Integrated Circuit) designers are faced with new problems as technology emerges. Ericsson has started to work in 65 nm and realized that the methods used in previous, larger technologies, does not offer good enough correlation between synthesis and the results after physical placement. This leads to several expensive and time-consuming iterations back and forth between Ericsson and the ASIC vendor.

In order to narrow the gap between Ericsson and the ASIC vendor, and hence increase correlation, physical planning has been identified as a possible solution. Cadence First Encounter, part of the Cadence Encounter digital IC design platform, is an advanced tool for silicon virtual prototyping. The tool basically brings back-end placement knowledge to front-end ASIC designers.

This master’s thesis main goal is to evaluate Cadence First Encounter and investigate how it could be integrated with Ericsson’s design flow. The tool has been tested on previous designs with known issues and the results are positive. By using the prototype work flow in First Encounter that is described in this report, it is possible to identify and correct issues with the netlists in time, which will help shortening the lead time in projects and hence also the time to market.
Acknowledgements

I would like to thank Ericsson AB for the opportunity to do my master’s thesis at their offices in Kista, Stockholm. I would especially like to acknowledge my supervisors Jens Andersson and Björn Forsberg, as well as the whole department PDR/UM, for supporting me in my work.

Furthermore I would like to thank Kevin Donnelly at Cadence design systems for technical expertise regarding First Encounter.

Finally I want to acknowledge my supervisor at the Department of Electrical Engineering at Tekniska Högskolan, Linköpings Universitet, Professor Mark Vesterbacka.

Håkan Roos

January 2007
# Table of contents

1 INTRODUCTION........................................................................................................... 1
  1.1 BACKGROUND........................................................................................................ 1
  1.2 METHODOLOGY...................................................................................................... 1
  1.3 GOAL....................................................................................................................... 2
  1.4 ABOUT CADENCE.................................................................................................. 2
  1.5 ABOUT FIRST ENCOUNTER.................................................................................. 2
  1.6 GLOSSARY............................................................................................................. 2

2 DESIGN FLOW ............................................................................................................. 5
  2.1 CURRENT DESIGN FLOW ..................................................................................... 5
  2.2 WORKFLOW IN FIRST ENCOUNTER....................................................................... 7
  2.3 POSSIBLE MODIFICATIONS IN FIRST ENCOUNTER – WHAT CAN BE DONE? ....... 7

3 THE PHOENIX-CORE .................................................................................................... 9
  3.1 THE PHOENIX-CORE IN FIRST ENCOUNTER....................................................... 9
  3.2 CTS ON THE PHOENIX-CORE.............................................................................. 11
  3.3 IMPLEMENTING POWER ON THE PHOENIX-CORE........................................... 12

4 THE HARDWARE ACCELERATOR .................................................................................. 15
  4.1 THE FIRST NETLIST SENT TO THE ASIC VENDOR........................................... 15
  4.2 THE FINAL VERSION OF THE NETLIST ............................................................... 17
  4.3 DIFFERENT FLOORPLANS .................................................................................. 17
  4.4 UTILIZATION ....................................................................................................... 19
  4.5 ROUTED HARDWARE ACCELERATOR................................................................. 21

5 TEST WITH A WHOLE ASIC ........................................................................................ 23
  5.1 GATE COUNT AND RUNT加工 ON ASIC_A ....................................................... 23
  5.2 ROUTING OF ASIC_A.......................................................................................... 24

6 CRA – THE CHIP RATE ACCELERATOR ....................................................................... 27
  6.1 FIRST VERSION OF THE CRA NETLIST............................................................. 27
  6.2 SECOND VERSION OF THE CRA NETLIST......................................................... 28
  6.3 THIRD VERSION OF THE CRA NETLIST............................................................... 29
  6.4 ATTEMPT TO SHORTEN THE LONG N2N RUNTIME........................................... 30

7 WIRE-LOAD MODELS (WLM) ....................................................................................... 33
  7.1 ZWL – ZERO WIRE-LOAD MODELS ................................................................... 33
  7.2 CWLM – CUSTOM WIRE-LOAD MODELS .......................................................... 33
  7.3 CWLM ON THE CRA............................................................................................ 34
  7.4 N2N WITH CWLM............................................................................................... 34
    7.4.1 Second netlist.................................................................................................. 35
    7.4.2 Third netlist.................................................................................................... 35

8 MULTI THRESHOLD VOLTAGE ..................................................................................... 37
  8.1 SUBTHRESHOLD LEAKAGE .................................................................................. 37
  8.2 MULTI-VT IN FIRST ENCOUNTER....................................................................... 38
    8.2.1 Multi-Vt during N2N....................................................................................... 38
  8.3 MULTI-VT ON THE CRA........................................................................................ 38
    8.3.1 Multi-Vt on the CRA during N2N ................................................................. 39
  8.4 MULTI-VT COST FUNCTION ................................................................................ 40
  8.5 MULTI-VT CONCLUSION..................................................................................... 40

9 TIMESAVING TECHNIQUES .......................................................................................... 43
  9.1 MULTI-THREADING ............................................................................................. 43
  9.2 SUPER-THREADING ............................................................................................. 43

10 USING FIRST ENCOUNTER ........................................................................................ 45
  10.1 RECOMMENDED USE OF FIRST ENCOUNTER.................................................. 45
  10.2 RUNT加工 ........................................................................................................... 46
Appendix A - Example run-script for First Encounter
Appendix B - Example output from the example run-script
Table of figures

FIGURE 1: ERICSSON S ASIC DESIGN FLOW SHOWING THE NEW AND OLD BORDER BETWEEN ERICSSON AND THE ASIC VENDOR. 6
FIGURE 2: SHORTENED WORKFLOW FOR FIRST ENCOUNTER. ................................................................. 7
FIGURE 3: THE FLOORPLAN THAT THE ASIC VENDOR USED FOR THE PHOENIX-CORE IN ASIC. A (ERICSSON DEVELOPED ASIC, SEE SECTION 5). ............................................................ 9
FIGURE 4: EXAMPLE OF A TESTED FLOORPLAN FOR THE PHOENIX-CORE. ........................................ 10
FIGURE 5: ANOTHER EXAMPLE OF A TESTED FLOORPLAN FOR THE PHOENIX-CORE. ............................ 10
FIGURE 6: BEST TIMING RESULTS FROM THE PHOENIX-CORE WITHOUT POWER NET. .......................... 12
FIGURE 7: FLOORPLAN VIEW OF THE PHOENIX-CORE WITH POWER NET. ........................................... 13
FIGURE 8: TIMING SUMMARY FROM A ROUTED PHOENIX-CORE WITH CTS AND POWER. ........................ 13
FIGURE 9: TIMING RESULTS FROM THE FIRST VERSION OF THE HARDWARE ACCELERATOR NETLIST ....... 15
FIGURE 10: EXAMPLE OF HOW THE RING OF INSTANCES CAN BE ADJUSTED. ...................................... 15
FIGURE 11: INTERCONNECTION BETWEEN TWO INSTANCES. ................................................................. 16
FIGURE 12: PIPELINED INTERCONNECTION BETWEEN TWO INSTANCES. ............................................... 16
FIGURE 13: SCHEMATIC VIEW OF THE ORGANIZATION OF INSTANCES IN THE HARDWARE ACCELERATOR. .... 17
FIGURE 14: SCHEMATIC VIEW OF THE HARDWARE ACCELERATOR'S INSTANCES WITH DIFFERENT SIZES OF THE RING. 18
FIGURE 15: FLOORPLAN OF THE HARDWARE ACCELERATOR IMPLEMENTING THE RING CONCEPT ............... 18
FIGURE 16: ACTUAL FLOORPLAN OF THE HARDWARE ACCELERATOR THAT THE ASIC VENDOR USED. EACH INSTANCE AND ITS CORRESPONDING LOGIC ARE INDICATED BY A SEPARATE COLOUR. ........................................... 19
FIGURE 17: NARROWER VERSION OF THE FLOORPLAN FOR THE HARDWARE ACCELERATOR ..................... 20
FIGURE 18: ANOTHER VERSION OF THE HARDWARE ACCELERATOR'S FLOORPLAN WITH MACROS AROUND THE WHOLE CORE BOUNDARY. ................................................................. 20
FIGURE 19: BEST TIMING RESULTS OF THE HARDWARE ACCELERATOR BEFORE ROUTE. ............................ 21
FIGURE 20: TIMING RESULTS FROM FIRST COMPLETE RUN ON THE FIRST VERSION OF THE CRA NETLIST ....... 27
FIGURE 21: RESULTS FROM A COMPLETE RUN INCLUDING N2N OPTIMIZATION ON THE SECOND VERSION OF THE CRA NETLIST. ................................................................. 28
FIGURE 22: N2N RUNTIMES ON DIFFERENT VERSIONS OF THE CRA NETLIST. ........................................ 30
FIGURE 23: TIMING RESULTS FOR THE THIRD VERSION OF THE CRA NETLIST. .................................... 30
FIGURE 24: EXAMPLE OUTPUT FROM "WIRELOAD” COMMAND ............................................................ 34
FIGURE 25: TIMING RESULTS FOR THE CRA WITH THE CWLM SWITCH FOR THE N2N OPTIMIZATION .......... 35
FIGURE 26: TIMING RESULTS FROM THE THIRD VERSION OF THE CRA NETLIST WITH CWLM GENERATION ACTIVATED FOR N2N. ................................................................. 36
FIGURE 27: LEAKAGE POWER GROWS EXPONENTIALLY WITH SHRINKING TECHNOLOGIES. [4] ............... 37
FIGURE 28: TIMING RESULTS FOR THE CRA (THIRD NETLIST VERSION) WITH MULTIvT LIBRARIES .......... 39
FIGURE 29: TIMING RESULT FOR THE CRA (THIRD NETLIST VERSION) WITH MULTIvT LIBRARIES AND THE MULTIvT SWITCH FOR N2N OPTIMIZATION ........................................... 40
FIGURE 30: EXAMPLE OF USEFUL SKEW. ................................................................................................ 51
1 Introduction

This section describes the background to why this evaluation was performed, the methodology used, the goals with it, a short presentation of Cadence and First Encounter and finally a glossary is given.

1.1 Background

The department PDR/UM at Ericsson AB in Kista, Stockholm designs ASIC:s that handles the greater part of the signal processing in base stations for WCDMA, GSM and other mobile standards. The technologies used are cutting edge and Ericsson is often first or among the first external customers in a new technology node.

Currently, the design is carried out by designing the blocks at RTL (Register Transfer Level) level in VHDL or System Verilog, and then synthesized to a netlist using ZWL (Zero Wire-load Model). Physical aspects like wire lengths and crosstalk are emulated by simply adding margins, that is shortening the clock period in different ways. Until 90 nm, this methodology has given good enough correlation between synthesis and the results after physical placement of the gates.

With shrinking technologies interconnects starts to consume a bigger and bigger part of the total delays on the silicon. Ericsson has started to work in 65 nm and realized that the previously used techniques does not offer satisfying correlation between synthesis and what is actually possible to implement physically. Also, the problems found after placement are not always the same as the most critical paths found after synthesis. This leads to time and cost consuming iterations. New processes nodes does not offer the same performance increase that previously nodes did and to compensate for this more and more data is processed in parallel, which leads to an increase in area and interconnect.

Physical planning might be a methodology that can simplify the design flow and save time by providing the ASIC vendors with placement information.

1.2 Methodology

The evaluation of physical placement was carried out using Cadence First Encounter. Different tools from different ASIC vendors were considered and Ericsson decided to test Cadence First Encounter. To get acquainted with the tool, a couple of tutorials were used and also a more extensive test case involving one of Ericsson’s own DSP-cores (Digital Signal Processor). The goal with the DSP-core was to try to meet the same timing requirements through prototyping using First Encounter, as the ASIC vendor did. The next step was to use First Encounter in a bigger context, namely one of Ericsson’s hardware accelerators containing no less than 96 memories. The first goal here was also to meet the same timing constraints as the ASIC vendor did, using the same floorplan, and second to try to improve this floorplan in terms of timing and area. This would hopefully be possible given the extra knowledge about the inner workings of the hardware accelerator that the ASIC vendor did not have.

The next step involved implementing both the DSP-core and the hardware accelerator in a whole ASIC, as black boxes, to see how First Encounter behaved on big designs.
The final step involved one of Ericsson's new designs that have not yet been manufactured. The purpose here was to test some new design techniques and investigate the possibilities to integrate and collaborate First Encounter with the ASIC vendors design flow.

1.3 Goal

The goal of this master thesis is to evaluate physical placement as a methodology and to investigate the pros and cons of using it in an ASIC design flow. The following questions will be answered:

- Is it possible to identify and prevent problems in 65 nm by using physical placement and fastprototyping?
- Is the tool easy to use and learn?
- How long are the runtimes on a typical Ericsson ASIC?
- How many licenses are needed?
- How is the current design flow affected if physical placement is introduced?
- How is the interface towards the ASIC vendors affected?

1.4 About Cadence

Cadence is (as of 2004) the world's largest supplier of electronic design technologies and engineering services. The company's revenue for 2005 was approximately 1.329 billion USD and they have around 5,100 employees in design centers, sales offices and research facilities spread around the world. Although Cadence are involved with computer systems, telecommunication equipment, networking equipment and consumer electronics, their primary product is software used for designing printed circuit boards and chips. The company's headquarter is located in San Jose, California and trades on the Nasdaq National Market under the symbol CDNS. Cadence's major competitors are Synopsys, Mentor Graphics and Magma Design Automation. [5], [6]

1.5 About First Encounter

First Encounter is a part of Cadence Encounter Digital IC Design Platform and its main purpose is basically to bring physical knowledge to front-end logic designers. The tool provides physical information for optimizing logic and also lets the designer use this information in downstream implementation. First Encounter is a very advanced tool with a lot of features that offers support for advanced 65 nm designs. The one feature in particular that Ericsson is interested in is the prototyping, or more precisely the ability to fairly easy determine if a certain circuit will meet timing or not. [5]

1.6 Glossary

The following section contains a glossary of technical terms and abbreviations used in this report.

ASIC
Application Specific Integrated Circuit. A microchip that is designed and customized for a specific application.
**CTS**  
Clock Tree Synthesis.

**CPU**  
Central Processing Unit.

**DC**  
Design Compiler. Synopsys tool for synthesizing the RTL code to a netlist.

**DSP**  
Digital Signal Processor. A specialized microprocessor designed specifically for digital signal processing.

**EDA**  
Electronic Design Automation. CAD-tools for electronics.

**Fanout**  
The number of inputs that an output is connected to.

**Gate count**  
Gate count is not an exact concept. It is rather a crude method of measuring the size of a design. Usually a “gate” is equal to a 2-input NAND gate (four transistors).

**I/O**  
Input/Output.

**LSF**  
Load Sharing Facility. A software job scheduler that distributes workload on networked machines.

**PLL**  
Phase-Locked Loop. Device that scales the clock signal to a desired level.

**RC**  
Cadence RTL Compiler. The synthesis tool used to run the N2N optimization.

**RTL**  
Register Transfer Level. A high-level representation of a circuit.

**TCL**  
Tool Command Language. A scripting language integrated with most EDA-tools.

**Wafer**  
The type of substrate used in semiconductor chip fabrication. Wafers are typically made of silicon.

**WLM**  
Wire-load Models, see section 7.

**ZWL**  
Zero Wire-load Model. See section 7.1, ZWL – Zero Wire-load Models, for further explanation.
2 Design flow

This section describes Ericsson’s current design flow and how it would change if First Encounter would be taken into use.

2.1 Current design flow

Currently Ericsson does not perform any layout and floorplanning. Once the netlist is created it is sent to the ASIC vendor for manufacturing. The netlist often contains errors that are not noticeable before the floorplan is created. The effect of this is that the ASIC vendor has to send the netlist back together with comments about what is wrong, so Ericsson can re-synthesize. This means crossing the dashed line (“Previous border”) in figure 1 below, which leads to increased costs in terms of time and money. By introducing First Encounter in Ericsson’s work flow, the border between Ericsson and the ASIC vendor is moved. The activities “Floorplanning” and “Prototype placement” can now be performed by both parts. Ericsson’s designers, who have detail knowledge of the inner workings of the netlist, can make an initial layout for the floorplan that the ASIC vendor later can refine and optimize. Prototype placement in First Encounter allows Ericsson to get an idea if a certain netlist will be possible to implement in hardware and hence decrease the number of expensive and time consuming iterations back and forth. More about profitability can be found in section 12 Profitability.
Figure 1: Ericsson’s ASIC design flow showing the new and old border between Ericsson and the ASIC vendor.
2.2 Workflow in First Encounter

The Encounter workflow offers a lot of features and can basically be used to place and route a whole design, from netlist to manufacturing. Many of these tasks are both time consuming and non-trivial to learn and perform quickly. Ericsson was interested in getting a quick and easy estimate if a certain ASIC would meet timing, and hence decrease the number of iterations back and forth to the ASIC vendor. Therefore, the workflow using Encounter was shortened to only include optimization of the netlist, floorplanning of the macros and pre-CTS (Clock Tree Synthesis) optimization (see figure 2). These steps are basically the features that are included in First Encounter and should be enough to perform fast prototyping, that is to get an idea if a certain netlist will meet timing or not. During the evaluation of First Encounter, several full licenses were used which gave access to all the features of SoC Encounter. Some of these extra features were used to examine if the timing results after prototyping correlated with the results after routing, with other words, to check the accuracy of the First Encounter workflow.

![Workflow Diagram]

Figure 2: Shortened workflow for First Encounter.

2.3 Possible modifications in First Encounter – What can be done?

As mentioned earlier, Ericsson’s interest in First Encounter is to get an overview of what is possible in terms of timing and also to keep the physical size of the chip reasonable. First Encounter lets you place macros and modify the size and shape of the floorplan. To meet timing, short wires are preferred, which leads to small chips. On the other hand, a small chip means less area for the logic, which in turn leads to routing congestion. Congestion occurs when more routes are requested in a specific layer than can actually fit. The tool also lets you decide where you want to place the I/O (Input/Output) pins. Doing this manually is generally required, but almost always compromise timing. Logic blocks can also be pre-placed, but it is not recommended unless strongly motivated. It was later learned that pre-placing of the logic blocks almost always compromised timing. The placing algorithm seems to work best when not interfered with placement guide lines for the logic.
It is also possible to rotate and mirror macros when placing them. This can be very useful in order to optimize utilization of the floorplan. However, in 65 nm rotating macros are no longer permitted to the same extent as in previously, technology nodes. The reason for this has to do with limitations in the manufacturing process.
3 The Phoenix-core

FlexAsic is an Ericsson designed DSP-concept that has been used in different ASIC projects for about 10 years. It is a signal processing DSP-core that is used in clusters on ASIC:s for radio base stations. Over the years, the DSP has been optimized and fitted for new technology nodes as they have emerged. Its current version is called the Phoenix-core and the one used here consists of eight memories, slightly more than two hundred thousand gates and is implemented in 65 nm technology. Because the DSP has been implemented many times before, its floorplan is probably near optimum. It was therefore a good circuit to test First Encounter on and also a challenge to try to meet the same timing constraints as the ASIC vendor did.

3.1 The Phoenix-core in First Encounter

Since variations of the Phoenix-core has been around for a while and been used in many of Ericsson's previous designs, its floorplan is well tested and verified. New problems arise every time a new technology node is taken into use and the requirements gets tighter. Figure 3 shows the floorplan that the ASIC vendor used when implementing the Phoenix-core in 65 nm.

![Floorplan](image)

Figure 3: The floorplan that the ASIC vendor used for the Phoenix-core in ASIC_A (Ericsson developed ASIC, see section 5).

The first tests with First Encounter were conducted using the tools prototyping feature on different floorplans. The prototyping performs a very fast placement and is basically an estimation tool. The first floorplan drafts were based on the following facts:

- The four similar, square memories are all connected to the same bus
- The two big memories connect through the same interface
- The two small memories also connect through the same interface
Based on these facts it was natural to try and place the respective group of memories close to and facing one another. Figure 4 and figure 5 shows two examples of tested floorplans.

Figure 4: Example of a tested floorplan for the Phoenix-core.

Figure 5: Another example of a tested floorplan for the Phoenix-core.

As mentioned earlier, prototyping was used with the intention to get a quick estimate if a certain floorplan would be useful or not. It turned out however, that the prototyping results varied a lot and gave results with worst slack around three to six times the clock period or worse. Further analysis led to the conclusion that the results were not reliable unless at least one optimization had been run. This was disappointing since an optimization run took around four hours on the Phoenix-core. It would have been great if the prototyping would have produced somewhat useful results, since it only took about five minutes. These results were later confirmed by the manual “Encounter Timing Closure Guide” [2], which states that you cannot trust post-placement slack results:

“After placement, there will be some basic buffering and resizing to get cells into their characterized region of timing as defined by their lookup tables. Until the cells are in that region, any timing analysis results are highly suspect.”
Some of the really bad results after prototype placement actually came out
good after optimization, while some results that showed promise after
prototype placement turned out really bad after optimization. The conclusion
from this is that at least one optimization is required after placement is
performed to get reliable results. Interesting to note is also that the more
advanced and thorough timing driven placement produced worse slack than
the prototyping on some occasions. This indicates that the prototyping
algorithm is too concentrated on speed rather than accuracy. A better balance
between speed and accuracy would make the prototype placement more
useful. It should not produce better results than what is actually possible. The
other way around is better.

The Phoenix-core netlist that was used in First Encounter was poorly
optimized. When the work in First Encounter was begun with a netlist-to-netlist
optimization by running the command

\texttt{runN2NOpt}

the results came out much better. The following optimizations also took less
time this way, but instead the \texttt{runN2NOpt} took about five hours to complete.
This indicates however, that the netlist that Synopsys Design Compiler (which
is the tool that Ericsson uses for logic synthesis) produces is no where near
optimized, or poorly compatible with Cadence First Encounter.

Thus, fast prototyping does not exist. To get any results that can be used as
an estimate, at least four hours are required on a design of the Phoenix-core’s
size. Still, that might be a price worth paying if the results are so accurate that
they can avoid involving the ASIC vendor to decide if a certain netlist will meet
timing or not.

The Phoenix-core was tested in First Encounter using a clock with a period of
3.2 ns, which translates to about 312 MHz. The ASIC vendor never managed
to meet timing using a 312 MHz clock and the floorplan in figure 3, which more
or less turned out to be the best one. Slightly better results were obtained
when the two smaller memories were moved down a bit, which avoided some
congestion, and when the four square memories were rearranged. Other than
that, the floorplan was kept the same as the one that the ASIC vendor used.
The chip was only meant for test purposes, so extra margins such as the age
factor of the silicon could be neglected and hence pushing the speed a little
further. The fact that it was a test chip also meant that only a limited number of
units were needed and therefore, through “speed sorting” (term for selecting
only the very best units), the ASIC should be able to deliver a chip that runs in
a speed close to 300 MHz. [1], [2]

\section*{3.2 \textbf{CTS on the Phoenix-core}}

CTS is not usually included in the prototyping work flow. It can be a very time
consuming, advanced and especially technology dependent task that allows
many options. Therefore it is better left for the ASIC vendor to do. However,
since the CTS will worsen timing a little, a certain margin in terms of extra
uncertainty (explained in section 4.1) should be introduced.

To check correlation between the prototyping results and the ones that the
ASIC vendor produced, a simple CTS was performed by running a CTS-script
provided by Cadence.
During the experiments on the Phoenix-core in First encounter, it was brought up to about 285 MHz, which is about the same as what the ASIC vendor did (given no removed margins and not compromising the yield). However, at this stage the power net had not yet been implemented, which should, according to Cadence, not affect the timing much. It might affect congestion though, since it needs some physical space and in turn that might affect timing. The timing results from the Phoenix-core without the power net are displayed in figure 6 below. Note the WNS that compromises timing.

<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>-0.329</td>
<td>-0.076</td>
<td>-0.329</td>
<td>0.662</td>
<td>5.234</td>
<td>-0.022</td>
</tr>
<tr>
<td>TNS (ns):</td>
<td>-18.925</td>
<td>-3.068</td>
<td>-15.857</td>
<td>0.000</td>
<td>0.000</td>
<td>-0.053</td>
</tr>
<tr>
<td>Violating Paths:</td>
<td>281</td>
<td>191</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>All Paths:</td>
<td>32307</td>
<td>21434</td>
<td>18811</td>
<td>132</td>
<td>6</td>
<td>536</td>
</tr>
</tbody>
</table>

Density: 48.401%
Real DRV (fanout, cap, tran): (487, 0, 0)
Total DRV (fanout, cap, tran): (619, 2, 217)

Figure 6: Best timing results from the Phoenix-core without power net.

### 3.3 Implementing power on the Phoenix-core

Implementing the power net is usually not included in the prototyping. Some choose to do it anyway just to see that it does not cause congestion or other problems. Since Ericsson is working with different ASIC vendors that implement the power net in their own unique way, it is defendable to exclude it from the prototyping flow. Also, according to Cadence, the power net should not affect timing too much. Since the power net will use some area and hence increase utilization a little, there is a possibility that the congestion might increase and in turn lead to slightly worse timing. In order to investigate this, a power net was implemented on the Phoenix-core to observe its effects.

To get comparable results, the same run-script that was used earlier on the Phoenix-core without the power net, was also used now. The script includes N2N optimization of the netlist, full timing-driven placement, pre-CTS optimizations, CTS, post-CTS optimizations, global detail route and finally a couple of optimizations post-route. Prior to executing the run-script, a simple power routing script that produced a standard power net, which can be seen in figure 7, was run.
The result was that utilization increased about one percent but the timing actually got better. This was probably just a lucky coincidence, but it indicates that the power net probably will not affect the timing very much. Figure 8 shows the timing information from First Encounter after a complete run on the Phoenix-core.

Figure 8: Timing summary from a routed Phoenix-core with CTS and power.

The results in figure 8 above show a very low utilization (density). Below 50% is very low at this point (pre-CTS) in the design flow. At this stage the chip is basically ready to be manufactured, since no big steps that should affect the utilization remains. A utilization of around 80% would be more appropriate for a block of this size. However, since this was the same floorplan layout and size as the one that the ASIC vendor used, congestion and timing probably got a lot worse if the floorplan was shrunken in order to increase utilization.
The hardware accelerator

The hardware accelerator contains 96 memories which makes it a lot bigger than the Phoenix-core. Because of its size it was an interesting test case to run First Encounter on, in order to see how the tool handled larger designs. It was also interesting to see if previous errors reported by the ASIC vendor could be detected.

4.1 The first netlist sent to the ASIC vendor

The exact same First Encounter workflow that was used on the final netlist lead to a worst slack of -732 ps (see figure 9), when applied to the first version of the hardware accelerator netlist. In the .sdc constraint file, an uncertainty of 790 ps was stated. The uncertainty works as a margin to cover things like extra slack introduced by the clock tree, the PLL (Phase-Locked Loop), the age factor of the silicon and other uncertainties. If the uncertainty was to be removed, the first netlist would actually meet timing, but unfortunately that is not an option. One also has to keep in mind that the floorplan used in this experiment was the final, “best” floorplan developed by the ASIC vendor. However, it is probably fair to say that had Ericsson used First Encounter on their first netlist of the hardware accelerator, they would have discovered many of the critical paths and could have corrected most, if not all, of them.

Figure 9: Timing results from the first version of the hardware accelerator netlist.

Density: 58.747%
Routing Overflow: 0.00% H and 0.25% V
Real DRV (fanout, cap, tran): (1330, 0, 0)
Total DRV (fanout, cap, tran): (1577, 196, 0)

Density: 58.747%
Routing Overflow: 0.00% H and 0.25% V
Real DRV (fanout, cap, tran): (1330, 0, 0)
Total DRV (fanout, cap, tran): (1577, 196, 0)

Figure 10: Example of how the ring of instances can be adjusted.

Each of the eight parts communicates with at least two others. This is done via memories called “left” and “right”. For example, memory “right” in part zero communicates with memory “right” in part one, see figure 11.
These paths, in between the instances, were too long and caused a negative slack, which left the ASIC vendor unable to implement them. Ericsson solved the problem by pipelining these paths (see figure 12), which is a common measure that easily could have been taken earlier if only the information would have been at hand.

The problem with negative slack between the instances was just one of many problems that were reported on the first version of the hardware accelerator netlist. A couple of other problem areas that the ASIC vendor had previously pointed out were also possible to identify. Due to heavy workload at the time, there are unfortunately no detailed change logs and documentation available on the problems that arose. The important conclusion is that all of the problems that the designers could remember, were possible to identify using First Encounter.

Another benefit is that one can get a full report on all the problems in a design from the beginning, which is something that the ASIC vendor did not produce. The ASIC vendor only reported the 50 worst paths, which is the default setting in First Encounter. It is often valuable to get an overview of all the paths with violations, in case they affect each other and for example a fix for one path might make another one worse.
4.2 The final version of the netlist

Timing was never a problem using the latest netlist of the hardware accelerator and the same floorplan as the ASIC vendor did. Despite an uncertainty of 790 ps there was still some margin on the timing. However, the utilization of the hardware accelerator was only about 49 %, which is really bad. Utilization should be somewhere around 70 % at this stage and later when the clock tree and power net are added closer to 80 %. Lower utilization means bigger area, fewer chips per wafer and in the end higher cost. Many different floorplans based on the same layout as the one the ASIC vendor used were tested, with modifications in terms of reduced area. All of them passed timing without problem but generated more congestion. The hardware accelerator is apparently very sensitive to congestion. The rule of thumb is to keep the horizontal and vertical congestion percentage below one per cent. If this is true, then the detailed router should be able to fix the rest according to Cadence. This was later proven wrong.

4.3 Different floorplans

The designer of the hardware accelerator had a few ideas on how to floorplan the design. He presented these ideas to the ASIC vendor but they chose not to do it that way, which later became clear why. The designer’s ideas were based on the ring structure described earlier in section 4.1. He wanted the eight blocks of the design to be organized like in figure 13.

![Figure 13: Schematic view of the organization of instances in the hardware accelerator.](image)

This way the ring of used instances could easily be increased or decreased when more or less memory would be needed. Figure 14 shows how the eight instances interact with each other given different sizes of the ring.
A couple of floorplans based on the ring idea were constructed. All of them had the eight instances placed according to figure 13, but different variations of the instance assembly were tested. Also different variations of the floorplan size as well as the size and spacing of each instance were tested. Figure 15 shows an example of one tested floorplan based on this concept.

The problem with this floorplan was not timing, but rather congestion. The placing algorithm works poorly when not given a large, coherent area to work on. In this floorplan it had to place all the logic in between the instances and the memories. The empty area in the top right corner was more or less left unused, while the center part of the floorplan suffered from major congestion. When some logic was forced into the upper right corner, the congestion got even worse. The big problem with this floorplan is that no matter where you place the controller that communicates with all of the eight instances, there will be major congestion. The best alternative seems to be a large empty space in the middle where the placing algorithm can freely place all the logic. With other words, the floorplan that the ASIC vendor produced (see figure 16) is good.
In the floorplan that the ASIC vendor used (figure 16), the memories are grouped together in their eight respective instances. The reason for why the ASIC vendor chose the placement pattern of the instances that they did, other than that it produces the best timing, remains unknown. There does not seem to be any logic explanation to why that particular order of the instances should be the best. Different variations of the exact same floorplan, only with the eight instances reshuffled, were tested, but they all came out slightly worse in terms of timing and congestion. The conclusion is thus that they must have made a qualified guess and proceeded from there with trial and error, to get optimum timing results.

4.4 Utilization

As previously mentioned, the hardware accelerator suffered from low utilization. In order to fix this, the size of the floorplan was reduced while keeping as much as possible of the layout. Figure 17 shows a narrower version of the ASIC vendor's floorplan (figure 16) that was used.
This floorplan worked fairly well. The timing was no problem and the congestion was only a little higher than in the floorplan that the ASIC vendor used: 0.21% versus 0.04%. The utilization was better though, 62% versus 49% in the actual floorplan. It turned out that any attempt to make the floorplan smaller, and thus increasing the utilization, caused the congestion to grow.

Another attempt to increase utilization was the floorplan in figure 18. This time effort was put into making the total area smaller while keeping the big, empty and coherent surface in the middle. The result showed a good 78% utilization, but the congestion, once again, got worse and rose to 1.44%. Congestion of this size was later proven to cause violations after routing and could therefore not be accepted.
4.5 Routed hardware accelerator

When using the workflow described in appendix A, First Encounter produced the results shown in figure 19.

<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>0.089</td>
<td>0.091</td>
<td>0.089</td>
<td>0.582</td>
<td>N/A</td>
<td>0.089</td>
</tr>
<tr>
<td>TNS (ns):</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>N/A</td>
<td>0.000</td>
</tr>
<tr>
<td>Violating Paths:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>All Paths:</td>
<td>30458</td>
<td>29130</td>
<td>1815</td>
<td>363</td>
<td>N/A</td>
<td>455</td>
</tr>
</tbody>
</table>

Density: 48.807%
Routing Overflow: 0.00% H and 0.04% V
Real DRV (fanout, cap, tran): (1561, 2, 0)
Total DRV (fanout, cap, tran): (1832, 198, 0)

Figure 19: Best timing results of the hardware accelerator before route.

The figure above clearly shows that there are no timing problems and only a slight congestion (routing overflow) of 0.04 %. This should, according to Cadence’s rule of thumb, be possible to route without any problems since it is well below 1 %. However, when the design was routed, many violations appeared.

Most of the routing violations were antenna violations due to the process antenna effect. The process antenna effect is a phenomenon that can appear when the area of the layer connected to the gate is large relative to the area of the gate. Static charges can accumulate during manufacturing and then discharge through the gate, which can lead to damage of the gate and cause the chip to fail. To avoid this problem there are mainly two solutions. The first one is to change the routing so that the area of the layer connected to the gate becomes smaller, and the other one is to insert diodes that will protect the gate by providing an alternate discharge path.

In First Encounter there is a routing option called “Insert Diodes” available in the routing form. By default, the “Fix Antenna” option is selected, but not the “Insert Diodes” which is the key to actually fixing the antenna problems since the problem could not be fixed by the router alone. With this option selected followed by a new route, almost all antenna violations disappeared. The remaining geometry violations could most likely be fixed by some manual moving of pins et cetera. First Encounter sometimes makes silly errors like placing two pins on top of each other although there is free space available. These errors can most likely be fixed by hand by a back-end designer.

In conclusion, Cadence’s rule of thumb seems to hold. As long as congestion is kept below 1 %, the design is probably routable with some help of a good back-end designer. This, of course, differs from design to design and the hardware accelerator seemed to be extremely sensitive to congestion. The floorplan that produced 0.04% congestion was routable after some work, but the smaller floorplan with higher utilization in figure 17, which only had 0.21% congestion, produced a lot more violations that might not be as easy to get rid of. With other words, the “below 1% rule” is a rule of thumb that works in most cases but perhaps not always. The results pre- and post-routing correlate good enough to be able to use prototyping, that is stop before route, as a fair estimate of the final result. [10]
5 Test with a whole ASIC

The next step was to test First Encounter's performance on a whole ASIC. This ASIC, called “ASIC_A” in this report, consists of several Phoenix-cores, four hardware accelerators and a lot of memories. The first test performed, after successfully loading the design with the raw netlists, was to do an automatic floorplan followed by a prototype placement. The automatic floorplan worked and produced a floorplan fairly quick, but the prototype placement caused First Encounter to crash. After several new attempts the prototype placement was finally left undone. Even though no brilliant result were expected from the prototyping if it had worked, it would still have been interesting to get a rough estimate of what the tool could perform without help.

The next test on ASIC_A was carried out with the Phoenix-core and the hardware accelerator implemented as macros. These macros were based on the best versions of the Phoenix-core and the hardware accelerator that had been produced earlier. The command

saveModel

builds a macro which is possible to import in another design. Again, the automatic floorplan worked, but the prototyping still caused First Encounter to crash. This was not a good thing. If there was something wrong with the netlists an error message would be much better. Below is the only information given by the tool:

“Encounter terminated by internal (SEGV) error/signal…”

After many failed attempts with new versions of the netlists, completely new runs on both the Phoenix-core and the hardware accelerator followed by new generation of block data for hierarchical flow (saveModel) and some editing in the “ASIC_A.sdc” file, a workaround was finally found. If “pre-place optimization” was excluded from the prototype placement, it worked. It would have been extremely helpful if the tool would have indicated that the pre-place optimization was causing problem. Then it would probably have been found a lot quicker and not as in this case, by a lucky guess.

The option “pre-place optimization” basically only removes all cells with the specified buffer footprint. This should not, according to Cadence, have any significant impact on the prototyping results. Still it would have been interesting to know what was causing First Encounter to crash during the pre-place optimization. It could be another indication that the netlists are in bad shape and that Design Compiler performs poorly. Still, the program should not crash, rather point out the problem. [1]

5.1 Gate count and runtimes on ASIC_A

ASIC_A chip is a big ASIC with almost 9 million gates including the instances of the Phoenix-cores and the hardware accelerators. The runtimes running ASIC_A differed a lot compared to the hardware accelerator and the Phoenix-core. A complete run, basically the run-script in appendix A (except for the N2N-optimization command since it is not necessary on the top level), took around 24 hours to execute. Due to the fact that ASIC_A mainly consists of hard macros, like instances of the Phoenix-core and the hardware accelerator together with a lot of memories, it was hard to measure the design’s size in terms of gates. The command
reportGateCount

counts ALL the gates, including the ones inside the hard macros and memories, and reported almost 35 million gates. By adding the switch

-stdCellOnly

to reportGateCount, blocks and I/O cells are supposed to be ignored. The result was now slightly less than one and a half million gates, but then the instances of the Phoenix-cores and the hardware accelerators had been ignored. Gate count is not an exact concept, rather a crude estimate that different tools and different vendors calculate differently. The number 9 million, as mentioned earlier, comes from the ASIC vendor post-layout. Since gate count numbers differs a lot, the results reported from the ASIC vendor post-layout, which is also the numbers that Ericsson usually uses, will be used in throughout this report.

Given the previous discussion regarding gate count, it was hard to say if the runtime was adequate at this point. Later on, after a lot of work had been done with another design (the CRA, see section 6) in First Encounter, it became clearer that the runtime for ASIC_A was reasonable. [1]

5.2 Routing of ASIC_A

From the previous designs worked on, it had been learned that routing is a very time consuming task. During the shorter prototyping workflow, “Trial routing” is performed which is a much quicker and less accurate routing procedure than the full detail router, “Nanorouter” performs. A couple of tests with the Nanorouter, started by issuing the command

globalDetailRoute

were performed on the smaller designs in order to check correlation between the prototyping and the final results. Since the hardware accelerator took over 24 hours to route and post-route optimize, nanorouting of ASIC_A was never intended. A nanorouting attempt was conducted however, and the result made it clear that it is not feasible to include it in the prototyping workflow for designs of ASIC_A’s size.

Nanoroute and post-route optimization on ASIC_A had run for over eight days, and it was almost done, when the whole LSF-system (Load Sharing Facility) crashed. The crash was probably not caused by First Encounter, but it nevertheless ruined the test and the results were lost. The routing test of ASIC_A was not all in vain though. It made it clear that detailed routing of blocks this size is not meant for prototyping, at least with today’s hardware. At the most the routing of ASIC_A used 37 Gb of memory. Since the Opteron machine it ran on has 32 Gb of RAM installed and the job was probably not alone on that machine, it had to swap memory from the hard drive which drastically increases runtime. First Encounter has a switch that allows it to run on several machines in parallel, something that might be useful in cases like this one. More about parallel execution can be found in section 9.
It was later learned that the extended runtime for routing of the whole ASIC_A, might have been because of the fact that the instances of the Phoenix-core and the hardware accelerator were not previously nanorouted. First Encounter then probably nanorouted each and every one of them which could explain the massive runtime. If the Phoenix-core and the hardware accelerator had been nanorouted before saved as hard macros, then First Encounter would probably only have to nanoroute the logic in between the macros in ASIC_A, which probably would have been a lot quicker. [1]
CRA – the Chip Rate Accelerator

The chip rate accelerator is one of Ericsson’s new designs and is a part of a big project. Only the CRA by itself is bigger (latest version approximately 2.8 million gates reported by `reportGateCount` in First Encounter, which makes it roughly four and a half time bigger than the Phoenix-core) than many of Ericsson’s previous whole ASIC:s. This design is not yet finished, so this time there were neither performance numbers to use as reference nor any gate count numbers from the ASIC vendor (therefore all gate count numbers regarding the CRA are from First Encounter). Three different versions of the CRA netlist were tested in First Encounter.

The ASIC project is going to be manufactured in 65 nm. This means new technology libraries and new routines. The ASIC vendor does not use First Encounter themselves, but they do support integration of it in their own design methodology.

6.1 First version of the CRA netlist

The first tests on the CRA were performed with an early version of the netlist that did not yet have all the functionality. The constraint file (.sdc) had a clock period of 3.5 ns, which equals a frequency of about 286 MHz, specified. Since this was an unfinished design, timing was not expected to be met. This first attempt was mainly an experiment and meant to be used as a reference. The CRA design does not contain any memories to place in a floorplan. The only work performed on the floorplan was assigning the input pins to the left side and output pins to the right side. The ASIC vendor’s libraries support routing in eleven layers, but Ericsson wanted to see how this design behaved in six layers, so the parameter

```plaintext
setMaxRouteLayer
```

were set to five which permits routing in layers zero through five. A standard run including N2N optimization of the netlist, full timing driven placement and in place optimization gave the results in figure 20.

![Time Design Summary](image)

- **Density**: 92.399%
- **Real DRV (fanout, cap, tran)**: (2022, 0, 0)
- **Total DRV (fanout, cap, tran)**: (2023, 0, 0)

Figure 20: Timing results from first complete run on the first version of the CRA netlist.

The timing was as expected not good, but the utilization (density) gave some valuable information. 92 % utilization is too high so the size of the square floorplan was therefore increased 25 % to 3,100 x 3,100 µm in order to decrease utilization and hence prevent congestion.
A new run was conducted and it took about 38 hours, of which the N2N accounted for about 22 hours, on a 64-bit AMD Opteron (all runs with the CRA were performed on Opteron machines). This seemed all right considering the size of the design.

6.2 Second version of the CRA netlist

In the second version of the netlist the total gate count had been reduced from about 3 to 2.3 million gates, even though an extra block (called block_A) had been added. The reason for the lower gate count was due to improved synthesis techniques and it was now reasonable to assume that the runtimes would be shortened, something that was later proven wrong. The constraint file (.sdc) specified an increased clock period that was now 4 ns, which equals 250 MHz. To save time, the first test with First Encounter was performed without N2N optimization. The results showed a worst slack of about 3 ns and a high utilization of about 95 %. After importing the previously constructed 3,100 x 3,100 \( \mu m \) floorplan used for the first netlist, the utilization decreased to about 62.5 % and the worst slack to about 2 ns. The runtime was about the same as with the previous version of the netlist (not counting the 22 hours for the N2N optimization), about 15 hours.

With N2N optimization included in the work flow, runtimes increased drastically. From 38 hours to 128 hours! The results are shown in figure 21 below.

---

**Figure 21**: Results from a complete run including N2N optimization on the second version of the CRA netlist.

Only about 200 ps improvement in terms of slack and an increase of about 27 % in utilization. This clearly shows that the N2N algorithm encountered some problems. First Encounter reported about 2.3 million gates prior to N2N optimization, and 4.8 million gates after. Why was all this logic added?

A probable cause of the excessive increase of utilization and runtime could be that the N2N optimization is trying to optimize something that is simply impossible to improve. At the beginning of a N2N optimization run, a pre-check is performed. This pre-check looks for timing inconsistency by checking correlation between RC (Cadence RTL Compiler) and First Encounter. If correlation is not within 5 %, N2N terminates. This was the case when N2N was run on the second version of the CRA netlist. The first version of the netlist passed the check without problems though. To bypass the pre-check and force N2N to run anyway, the switch

```
-bypassPreCheck
```
was enabled. The bypass switch was also used during N2N optimization on ASIC_A, but then the correlation was just barely off and the results turned out ok anyway. On the second version of the CRA netlist however, the correlation was way off and bypassing the pre-check obviously did not solve the problem. Instead, it resulted in a huge netlist and extremely long runtime.

The conclusion from this run was that the bypass switch for N2N optimization should be used with caution. The pre-check should not fail, and if it does something is probably seriously wrong, which will most likely result in unusable results and extremely long runtimes as in the previous example. Even though the bypass switch was engaged, the tool should not continue for such a long time if timing cannot be improved. The enormous increase in gate count and runtime are never worth the negligible performance increase in terms of timing. It would be much better if the tool had some kind of timeout setting that would trigger when no performance increase has been reached for a certain time, instead of just keep on going and going. The lengthy run also occupied a lot of valuable computation resources for over a week. The process was run on one of few Opteron machines and it used 60 Gb of memory. Since the machine “only” have 32 Gb of RAM installed, this meant major swapping (which of course by itself add to runtime drastically) and stealing of memory from other processes running on the same machine. Very high memory usage is usually a sign that something has gone wrong, which could for example generate a warning or a “Do you want to proceed?” query.

All together this is unwanted behavior, especially when the results are no good. The problem that caused First Encounter to behave like this was most likely RTL errors in block_A made by an Ericsson designer. Therefore it would be very desirable if First Encounter could recognize this and point out the problem, instead of trying to fix it in vain.

6.3 Third version of the CRA netlist

The second version of the CRA netlist clearly suffered from some problems. It could not be determined exactly what the cause was that made First Encounter to behave the way it did. It is not important either, since the design was at an early stage. What is important is the fact that a human mistake caused First Encounter to perform poorly.

The third version of the CRA netlist had again undergone some serious remodeling and was also using a newer version of the standard cell library (.lef and .lib files). Block_A, which is a quite small block but hard to implement timing wise, had now been pipelined in order to meet timing constraints. More blocks including a several very large multipliers and adders had been added.

In an attempt to try and see correlation between the gate count and the runtime, the table in figure 22 was made. The runtime for N2N on the third netlist was above 30 hours which gave a significantly lower gates-per-hour ratio compared to the run on the first netlist, as can be seen in figure 22 below. The second netlist had by far the worst gates-per-hour ratio which indicates that something is wrong. According to these results there does not seem to be an obvious connection between the gate count and the runtime, but according to Cadence the runtime should be proportional to the gate count. This is probably true given that the tool does not run into problems while optimizing or placing the design. “Problems” generally means that the algorithm has to perform further iterations to reach, or try to reach, the goal that is set, which then of course is reflected on the runtime.
<table>
<thead>
<tr>
<th>CRA netlist version</th>
<th>Gate count</th>
<th>Runtime (h)</th>
<th>Ratio (gates/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>2.99M</td>
<td>21.75</td>
<td>137k</td>
</tr>
<tr>
<td>Second</td>
<td>2.28M</td>
<td>108.25</td>
<td>21k</td>
</tr>
<tr>
<td>Third</td>
<td>2.80M</td>
<td>33.50</td>
<td>84k</td>
</tr>
</tbody>
</table>

Figure 22: N2N runtimes on different versions of the CRA netlist.

The constraints (.sdc) file for the third netlist still stated a period of 4 ns (250 MHz) and this time the design had actually met timing during synthesis. The whole run including N2N, full timing driven placement and optimization took about 58 hours. The timing results are given in figure 23 below.

![TimeDesign Summary](image)

Figure 23: Timing results for the third version of the CRA netlist.

As figure 23 shows, the negative slack is now reduced to about one nano second. The constraint file (.sdc) still specifies an uncertainty of 800 ps which might be a bit pessimistic. With the enormous number of gates after N2N on the second netlist in mind, the floorplan for the third version of the CRA netlist was decreased to 3,000 x 3,000 µm, which seems a bit low. Normally it would have been alright at this stage in the design flow, but on the CRA only six out of eleven available layers are used, which means that for example the power net can occupy its own layer. Therefore not much more are going to be added in the current six layers, and hence the utilization could be higher.

There is still functionality left to be added to the CRA before it is finished. Even though none of the tested netlists were ready for sign off, this is an excellent example of how First Encounter can be used to ensure that everything is as expected and to minimize the risk of unpleasant surprises later on in the design flow.

### 6.4 Attempt to shorten the long N2N runtime

When the second netlist was still the newest one, and it was not yet known that it was the netlist that caused the long runtime, an attempt to shorten the long N2N run was made. Since the N2N process seemed to be working on something impossible and not giving up, the idea of increasing the clock period and hence give the N2N process more space to easier meet timing, came up. The worst slack was around two nano seconds, so the clock period was simply increased with two nano seconds. The expectation was now that the design would meet timing without having to optimize for an extended period of time.
The run took three and a half days which is about one day less than without the extra two nano seconds. The results were not as good as expected, but there was still a difference. The N2N runtime had been decreased by about 22%, but apparently the N2N algorithm still had to work long and hard to optimize the netlist. The idea that the two nano seconds of worst slack were met early in the N2N run and that the rest of the time was spent trying to improve this was partly proven right. The reason for the long runtime was most likely the faulty netlist. The conclusion is that abnormally long runtimes most likely are due to constraints that are impossible to satisfy, which in turn can be caused by bad netlists.
Wire-Load Models (WLM)

In order to produce the best possible netlist from the RTL code, the front-end designer would like knowledge about placement to be able to estimate net parasitics. Unfortunately, this information is not available since the placement cannot take place before the actual netlist is ready. This resolves in a classical chicken-and-egg problem.

A common solution to the chicken-and-egg problem is to estimate the net parasitics with a wire-load model (WLM). The WLM contains information about capacitance, resistance and area of the wire for a given fanout. The synthesis tool has complete control over the netlist but the physical layout plays a big role for the resulting timing. Back-end place and route information can be communicated through a WLM.

The ASIC vendor usually derives WLM from statistical data collected from previous designs. The WLM is basically a lookup table that maps the fanout of the net with the corresponding capacitance and resistance. [7], [8]

7.1 ZWL – Zero Wire-load Models

The zero wire-load model is an optimistic estimation WLM. Instead of trying to accurately determine an estimation of a design’s timing, ZWL assumes the best possible timing scenario. ZWL resembles an area-based WLM with the smallest possible wire lengths. Since the results are always optimistic, it is unlikely that a design that fails ZWL analysis will produce successful timing results after physical placement.

ZWL is a simple technique that together with a sufficient margin has proven to work well for previous designs in 90 nm and up. Now, in 65 nm, results are starting to differ too much from reality. Today, most of the timing delays reside within the net rather than in the logic gates.

Synthesis tools, like Cadence RC and Synopsys DC (Design Compiler), have libraries containing timing information about different nets. Unfortunately, the libraries do not contain information about all possible fanouts. If, for example, a WLM for a net with a fanout of 60 is needed, but the libraries only contain WLM for fanouts of 40 and 100, then linear interpolation is used to estimate the capacitance and resistance values for the net with the fanout of 60. A possible solution to avoid this interpolation problem is called CWLM. [8]

7.2 CWLM – Custom Wire-load Models

Custom wire-load models are what the name implies: WLM custom fitted for a specific design. CWLM can be generated using First Encounter after a design has been imported. By issuing the command

wireload

First Encounter generates wire-load models and output the models in hierarchical and flat formats. See figure 24 below.
The hierarchical format considers only nets that cross the boundaries of child modules within the module, while the flat format considers all nets within the module and its child modules. Both the flat and hierarchical formats are available in different formats. The flat format was used since it considers all nets.

CWLM can be generated anytime during the workflow in First Encounter. There is no definite right or wrong answer to the question when to generate them, since the CWLM is an estimate. The point at which they are generated in the flow has some small effect on the overall error in the wire estimate. In general the results are probably relatively more accurate the later in the flow they are generated. Once the CWLM are generated, they can then be used to rerun synthesis and hopefully produce a better netlist. [1]

7.3 CWLM on the CRA

To test CWLM generation, First Encounter was run on the CRA according to the example run-script in appendix A upon which CWLM was generated. The CWLM were then handed over to aid the re-synthesis of the CRA. Unfortunately this is where the experiment ended. It turned out that Ericsson did not have the required licenses to run synthesis in Synopsys Design Compiler with CWLM. This is something they intend to deal with later on, but for now it was not possible.

Despite the unfinished experiment, the work with CWLM in First Encounter was not all in vain, even though it would have been interesting to see the effects of CWLM integration in the synthesis. It was learned how and when to generate the CWLM and that it all went smoothly.

7.4 N2N with CWLM

Since it was not possible to rerun synthesis with the CWLM generated, a different approach was tried. When running the N2N optimization there is a switch called CWLM. Using this switch causes a CWLM to be generated and applied to the current N2N optimization run. According to Cadence it could be a valid alternative as to re-synthesis with CWLM. A comparison is of course impossible due to the lack of results from the previous experiment, but the results could still be compared to when not using CWLM at all.
7.4.1 Second netlist

The N2N optimization with the switch CWLM engaged took much longer than usual. Previously, the N2N optimization has taken around 22 hours (first CRA netlist), but now with the CWLM switch activated, it took 4 days and 8.5 hours. The great increase in runtime was probably mostly due to the faulty netlist as discussed previously, but at the time of the experiment this was not yet discovered. The results were not as good as expected either. There was still quite a long way to go to meet timing on the CRA, but a comparison could still be made. The worst slack was only decreased with about two hundred pico seconds compared to the run without the CWLM switch activated, see figure 25 below.

---

timeDesign Summary
---

<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>1.838</td>
<td>1.838</td>
<td>-1.772</td>
<td>-0.412</td>
<td>N/A</td>
<td>1.009</td>
</tr>
<tr>
<td>TNS (ns):</td>
<td>-71024.1</td>
<td>-15266.2</td>
<td>-55680.7</td>
<td>-77.173</td>
<td>N/A</td>
<td>0.000</td>
</tr>
<tr>
<td>Violating Paths:</td>
<td>59255</td>
<td>17243</td>
<td>41225</td>
<td>787</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>All Paths:</td>
<td>85941</td>
<td>36829</td>
<td>45016</td>
<td>4096</td>
<td>N/A</td>
<td>5</td>
</tr>
</tbody>
</table>

Density: 79.958%
Real DRV (fanout, cap, tran): (942, 1, 0)
Total DRV (fanout, cap, tran): (948, 6, 5)
---

Figure 25: Timing results for the CRA with the CWLM switch for the N2N optimization.

Based on this experiment only, it was definitely not worth the extra runtime for only 200 ps improvement. Since the second netlist contained errors that caused the runtimes to increase a lot, another run using the third netlist was performed.

7.4.2 Third netlist

As it was later discovered, the second netlist contained errors that caused the runtimes to increase drastically. Therefore another run with the CWLM switch for the N2N optimization was conducted, now on the third version of the CRA netlist. The goal here was to determine if the timing would improve more than 200 ps when using a working netlist.

This time the run was quicker. After about 62 hours the result in figure 26 below, was produced. The result shows about one nano second of worst slack. This is not a big improvement compared to the results in figure 23, which represent an equal run, part from the CWLM switch for the N2N optimization. The difference is 105 ps and three percent increase in utilization, despite the use of a bigger floorplan. The run with the CWLM switch for N2N was conducted on a square floorplan with the side 3,100 µm and the run without CWLM was conducted on a square floorplan with the side 3,000 µm. The difference in runtime was not so pronounced, only about four hours.
Density: 76.005%
Real DRV (fanout, cap, tran): (7054, 0, 0)
Total DRV (fanout, cap, tran): (7055, 0, 0)

Figure 26: Timing results from the third version of the CRA netlist with CWLM generation activated for N2N.

This sums up to the conclusion that CWLM generation during the N2N optimization results in a slim advantage. A small increase in runtime and a noticeable increase in gate count, for only 105 ps less worse slack, are not worth the while. Hence, the CWLM switch for N2N optimization can be omitted.
8 Multi Threshold Voltage

Multi threshold voltage, or multiVt as it is usually called, is a technique that among other things can improve timing by trading leakage power for frequency. Basically it lets you use transistors with different threshold voltages in the same design. A transistor with a low threshold voltage is fast and by using it, timing can be improved. The drawback is that transistors with low threshold voltage have a high leakage current when shut off, see section 8.1. The trick with multiVt is to use low threshold voltage transistors on timing critical paths and high threshold voltage transistors where timing allows in order to minimize power dissipation.

8.1 Subthreshold leakage

The two principal sources of power dissipation in today's transistors are static and dynamic power. Dynamic power has always been a factor and comes from the repeated capacitance charge and discharge on the output of the gates when the transistors are turned on and off. Smaller technologies have allowed a reduced supply voltage, and since dynamic power is proportional to the square of the supply voltage, this has lead to significant reductions in power consumption. Static power is the current that leaks through the transistor when it is turned off. This subthreshold leakage has for long been so small that it has not been considered. Today, however, with shrinking technologies and reduced supply voltages, also the threshold voltage has been reduced which has lead to an exponential increase in subthreshold leakage, see figure 27 below. As the figure shows, static power dissipation is now the dominating factor. [3]

![Dynamic Power vs Leakage Power](image)

Figure 27: Leakage power grows exponentially with shrinking technologies. [4]
8.2 MultiVt in First Encounter

The configuration file (design_name.conf) specifies the library files that First Encounter use. Traditionally mostly high threshold voltage (highVt) cells have been used in order to minimize static power dissipation and hence only the ASIC vendor specific highVt libraries have been included in the configuration file. To enable multiVt in First Encounter one simply just includes the regular threshold voltage (regularVt) and the low threshold voltage (lowVt) libraries in the configuration file as well. First Encounter will then choose among the available libraries and select the one that it sees fits the best.

First Encounter tends to use lowVt cells to meet timing, which means that the number of lowVt cells increases during optimization. Since this has a bad effect on leakage power the following command is supposed to reduce the number of lowVt cells back to a minimum:

```
optLeakagePower
```

When no parameters are given to the command, it will use the following default settings:

```
- postRoute - highEffort - checkTNS
```

which in most cases work well. The default settings will among other things worsen slack for paths are better than the target slack, if leakage power can be improved. No paths will of course be worsened more than up to the target slack. [1]

8.2.1 MultiVt during N2N

It is also possible to enable the use of multiVt libraries during the netlist to netlist optimization (runN2NOpt). This is done by the switch

```
-multiVtEffort [medium | high | low]
```

The switch will cause N2N to use the multiVt libraries during leakage power optimization. Hence, the effort level (medium, high, low) is in terms of leakage power optimization effort. When set to "low", highVt libraries will be used on non-timing-critical paths and lowVt libraries will be used on timing-critical paths. The multiVtEffort level was set to "low" in the following experiments. [1]

8.3 MultiVt on the CRA

As designs today are becoming more and more complex and more and more timing-critical, all available methods for reaching timing needs to be looked upon. The latest version of the CRA netlist (version three) that was tested earlier still had a worst slack of about one nano second. With the intention to test multiVt in First Encounter, the third netlist of the CRA was used.

To get the best possible results in terms of timing, all libraries (highVt, regularVt and lowVt) were included. Normally the lowVt libraries probably would not have been included, since they cause a drastic increase of leakage power. The timing results are displayed in figure 28 below.
Figure 28: Timing results for the CRA (third netlist version) with multiVt libraries.

As the results show, the CRA now met timing (positive slack). There is still an uncertainty of 800 ps, which is a generous margin. The utilization is below 50 % which is a bit too low. However, decreasing the floorplan, which was square with the side 3,200 µm, some should not affect timing too much. The interesting parameter to look on at this point is the leakage power. The command

gave the following results after the multiVt run:

First Encounter used lowVt cells in 90 % of the design which resulted in a total leakage power of about 356 mW. To put this in perspective, the same command gave the following result on the CRA with only highVt:

With only highVt libraries, the total leakage power was about 57 mW, which is more than seven times less than with multiVt libraries. The leakage when using multiVt libraries is too high. If timing cannot be met without using 90 % lowVt libraries, then the design must be redone and re-synthesized. In the current state the chip would leak way too much power. [1]

8.3.1 MultiVt on the CRA during N2N

As mentioned in section 8.2.1, multiVt can be applied already during the N2N optimization via a switch. With the switch activated, the N2N runtime roughly doubled and the utilization grew from about 47 % to 67 %. This together with the fact that the timing was not improved, but worsened, makes the multiVt switch for N2N optimization questionable. The timing results are displayed in figure 29 below.
timeDesign Summary

<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>-0.044</td>
<td>-0.044</td>
<td>0.131</td>
<td>0.124</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TNS (ns):</td>
<td>-2.704</td>
<td>-2.704</td>
<td>0.000</td>
<td>0.000</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Violating Paths:</td>
<td>177</td>
<td>177</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>All Paths:</td>
<td>65463</td>
<td>56851</td>
<td>4161</td>
<td>4451</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Density: 66.674%
Real DRV (fanout, cap, tran): (3788, 0, 0)
Total DRV (fanout, cap, tran): (3789, 0, 0)

Figure 29: Timing result for the CRA (third netlist version) with multiVt libraries and the multiVt switch for N2N optimization.

The worst slack is assuredly only 44 ps, which with the 800 ps uncertainty in mind, is not much. However, the multiVt feature for N2N can definitely be questioned since it only added cells, runtime and slack. Could it be that the multiVt switch worked in favor for the leakage power? The command reportLeakagePower gave the following printout:

Total leakage power = 357973.118466uW
Cell usage statistics:
Library CU65HP_sc_rvt_STD, 639370 cells (31.004343%), 43605.324076uW (12.181172%)
Library CU65HP_sc_lvt_STD, 873947 cells (42.379456%), 310634.830352uW (86.776016%)
Library CU65HP_sc_hvt_STD, 548878 cells (26.616203%), 3732.964038uW (1.042806%)

Total leakage power shows almost 358 mW, which is higher than in the previous run where the multiVt switch for N2N optimization was not used. Interesting to note is that the lowVt cell usage is much lower, only 42%. The reason for this is the higher gate count caused by the long, inefficient N2N run.

8.4 MultiVt cost function

The multiVt cost function that weighs timing, leakage power and area against each other during optimization, seems to be a bit off. A couple of experiments using the CRA netlist without N2N optimization and with only highVt and regularVt libraries were performed. These came out without about 400 ps of positive slack. Evidently, lowVt cells are not needed in order to meet timing on the CRA. First Encounter should not use lowVt cells unless absolutely necessary, since it has a very bad impact on static power dissipation. It is never worth trading massive leakage power for a slight save in area or extra positive slack. As soon as timing is met, all effort should be put into decreasing leakage power. There is no point in having a chip that meets timing with a one nano second margin, but leaks all lot of power.

8.5 MultiVt conclusion

The results from the tests performed using multiVt are clear. By using multiVt, timing can be much improved, but at the cost of leakage power. First Encounter tends to use almost only lowVt libraries with the default settings which results in unacceptable leakage. The optLeakagePower command, whose purpose is to remove lowVt cells that were not needed, did not do much good since First Encounter had decided that lowVt libraries had to be used.
The multiVt switch for N2N optimization did not do any good at all and should therefore not be used. The switch probably has some meaning that motivates its existence, but none that were discovered during these tests.

In conclusion can be mentioned that multiVt is a powerful technique that is easy to implement in First Encounter in order to meet timing, unfortunately at the cost of leakage power. Also, when using multiVt it is probably best to exclude lowVt at first, in order to see if it is enough with highVt and regularVt to meet timing (see section 8.4).
9 Timesaving techniques

This section presents two different techniques to decrease runtimes in First Encounter.

9.1 Multi-threading

First Encounter uses the term “multi-threading” for the use of multiple CPU:s (Central Processing Unit) in parallel on the same machine. For example if you have a four CPU Opteron machine, then Nanoroute can use all 4 CPU:s during detailed routing and it will be around 4 times faster. Each extra CPU requires an additional Encounter license or a special “Route Accelerator” license.

The settings for multi-threading are, according to Cadence, easy to set up and can be found in the “Encounter User Guide” [9]. Unfortunately, Nanoroute is the only command that currently supports multithreading and it is not included in the First Encounter workflow. Cadence are working on multithreading support for other parts of the workflow as well, so maybe in a later version we will see support for the tasks included in the prototyping workflow.

9.2 Super-threading

Super-threading is also a way of accelerate routing, but it uses several machines cross a network, for example in an LSF-cluster like the one Ericsson uses. Unlike multi-threading, the performance is not linear with number of machines as there is an overhead with network management. Also here an additional license per extra machine is required.

Super-threading can be used with RC which is used during N2N optimization. Since the N2N optimization accounts for a big part of the First Encounter workflow, this was an interesting option. The syntax for setting it up was somewhat complex and the error messages from the Encounter console were hard to interpret. Since Cadence could not provide sufficient help, the super-threading with N2N could never be evaluated.

Unfortunately the super-threading never produced any results. However, as long as super-threading is only supported during N2N, and not the whole N2N run for that matter, it is probably not worth the extra effort and money (extra licenses) to use it. [1], [9]
10 **Using First Encounter**

This section briefly and simplified describes how to start and run a First Encounter session. Runtimes, bugs and requested features are also discussed.

10.1 **Recommended use of First Encounter**

First Encounter is possible to run in graphical mode, via a command line or by using a combination of the two. The last alternative is probably the best one. On one hand, certain tasks as for example creating the floorplan, would be very difficult to perform using only the command line. On the other hand, due to often very long runtimes, making a .tcl (Tool Command Language) script of the command line commands is indispensable to be able to perform overnight runs.

When starting First Encounter for the first time on a new design, it is strongly recommended that a new unique work directory is created. This is because some default directories containing timing reports et cetera and some other temporary files might otherwise be overwritten, especially if several First Encounter sessions are running at the same time. Suggested naming policy for these work directories could be the current date followed by directory named run[number], for example:

```
dec20/run2/
```

When the work directory is set up, it is probably easiest to start First Encounter in graphical mode, from this directory, by issuing the command

```
encounter
```

Once the program has started the first step is to import the design. The “Import Design” form needs information about where the following files are located:

- test.v – The netlist
- test.sdc – The constraints file
- The corresponding library files - .lef and .lib files

As soon as the paths to the above files are entered, it is practical to save the configuration to a .conf file. This way it is easier to load the design next time and also possible to do so via the command line. When the design has finished loading, the floorplan can be created and saved to a “.fp” file.

It is now possible to perform the whole run from the command line by entering

```
encounter -init start_script.tcl
```

which will cause the script `start_script.tcl` to be executed once First Encounter has started. The configuration file (.conf) and the floorplan file (.fp) will be used in the script to load the design. See appendix A for an example run-script and appendix B for an example output from the run-script.
This is a very convenient way of running First Encounter. It is often possible to reuse old scripts by making small adjustments, for example if you want to try out a couple of different floorplans with the exact same workflow.

10.2 Runtimes

A typical First Encounter run on ASIC_A netlist including N2N optimization, timing driven placement and a couple of optimizations preCTS (basically the suggested run-script in appendix A) takes about twenty hours on an 64-bit AMD Opteron machine with 32Gb of RAM (running Red Hat Linux EL 3.0), which is pretty much the fastest machine available today.

Once the N2N optimization is done on a particular netlist, this does not have to be repeated unless the netlist is changed. Using the saved N2N version of the netlist saves a lot of time since this step is very time consuming, typically about nine hours on ASIC_A. Also, the last part of the script, `saveModel` in appendix A, is used to save necessary block level information for top level implementation and can be omitted until this is required.

When designs are getting even bigger, runtimes increase. The CRA for example, which is not bigger than ASIC_A if gate count is compared, takes much longer time to execute a “prototype run” on. The reason for the longer runtime despite the fewer gates, is because most of ASIC_A’s gates are located in memories and hard macros. To get manageable runtimes on the CRA, the N2N optimization can be omitted. If the synthesis is run with optimal settings, then this step should not be needed. A complete prototype run (according to the script in appendix A) minus the N2N optimization on the CRA, takes somewhere around 30 to 40 hours. This can seem like a long time, but in the end it will probably be worth it and end up saving valuable lead time in a project.

10.3 Bugs

This section presents some of the bugs found in the program.

- When saving a relative floorplan, the orientation of the macros are not saved. The only way to save the correct orientation is to manually edit the `floorplanFileName.fp.relFPlan` file and enter the correct values. This can be very exhausting.

- Many times if something goes wrong, the program will just terminate. One example is the prototyping of ASIC_A’s netlist that always caused the program to terminate if the default option “pre-place optimization” was selected.

- It seems like the newer version (SOC61USR1) crashes easier. Many things that worked fine in the previous version now cause the program to crash. One example is clicking the “Schematics” tab in the “Timing Path Analyzer” within the “Timing Debug” window, which always causes a crash.
10.4 Requested features

Below is a list of features that would be nice to have in First Encounter.

- A break command that would cancel the current command and return to the prompt would be very helpful. The only way to cancel a command right now is to hit ctrl-C twice which will cause the whole program to terminate.

- More information about what went wrong if the program crashes.

- Display the physical view of the floorplan during a run. During a run the encounter window becomes inactive and does not properly display your design. If you have run several runs simultaneously it might be hard to remember which one is which.

- As mentioned in section 2.3, blocks can be rotated in 65 nm but should not be. This is something that the tool could make the user aware of.
11 What does First Encounter do?

This section briefly describes what the different major steps of the First Encounter prototyping workflow, as defined in this evaluation, do.

11.1 N2N optimization

The N2N optimization, started by issuing the command `runN2NOpt`, brings pre-placement synthesis capabilities into First Encounter by using RTL Compiler Ultra to re-map and re-optimize the gate-level netlist in order to improve timing and area. The command compares timing correlation between RTL Compiler and Encounter. The netlist is then restructured to simplify gates in the critical paths to allow for easier placement and optimization later in the flow, if the timing correlates. If it does not, the correlation check can be overridden with the switch `--bypassPreCheck`, which should be used with caution (see section 6.2).

The command offers many options but the recommended use specified in the manual “Encounter Text Command Reference” [1], which is also how it has been used in the example script (appendix A), gives good results. [1], [2]

11.2 Floorplanning

This step includes placing the memories, I/O pins and logic blocks. The placing of the logic blocks are, according to previous experiences, better left for First Encounter to do. There is a command that performs automatic floorplanning, but the results are poor. It is much better to manually place the memories using relative floorplanning, which is a technique that allows for accurate and precise placement.

The floorplanning is probably the task that Ericsson eventually could do better than the ASIC vendors. This is due to the fact that Ericsson knows the inner workings of the design and can use this information to place blocks so that they can interact with each other in an optimum way. [1]

11.3 Placement

When the floorplanning is done, the design is ready to be placed. This can be done using the quick prototype placement or the more accurate timing driven placement. The prototype placement can be used to test different floorplans in order to get a rough estimate of the timing. It should however be used with caution. It has been noticed that floorplans with fairly good results after prototype placement can produce bad results after optimization, and floorplans with worse results after prototype placement can produce better results after optimization. With other words, the prototype placement command is a very rough estimate and its results should not be trusted unless after optimization.

The placement command `placeDesign` also offers a lot of options and honors the settings of a number of other commands. To obtain best possible timing results, `placeDesign` should be run with the option `timingDriven`, which is default, and also with high congestion effort which is set by the command `setPlaceMode congHighEffort`. The command also performs pre-place optimization which is enabled by default.
There is also an option called `modulePlan` that is disabled unless specified by the user. The option `modulePlan` is set with the command `setPlaceMode` which `placeDesign` honors. The command is supposed to improve wire length, timing, relative module location and congestion results during the early phase of global placement. The manual states that the command might not improve large designs. The command was tested several times on different designs and it always produced slightly worse results than without `modulePlan`. This was a little unexpected since Cadence really seemed to believe in this feature. [1]

11.4 Optimization

Optimization is run by issuing the command `optDesign` and one of the following switches, depending on the current stage in the design flow: `-preCTS`, `-postCTS` or `-postRoute`. The command should be run incremental with the switch `-incr` if it has already been run previously in order to save time. `optDesign` performs the following:

- Corrects design rule violations
- Reduces total negative slack
- Optimizes setup time for all path groups on the initial pass and then, if the worst slack is not on a register-to-register path, optimizes the register-to-register paths on the second pass.
- Fixes hold time violations (optional).
- Performs useful skew optimization (optional).
- Reclaims area (optional).

...by using some or all of the following (depending on the specified options) techniques:

- Adds buffers
- Resizes gates
- Restructures the netlist
- Remaps logic
- Swaps pins
- Deletes buffers
- Moves instances
- Applies useful skew

`optDesign` honors the settings from the following commands:

- `setAnalysisMode`
- `setTrialRouteMode`
- `setOptMode`
However, when specifying `-preCTS`, `-postCTS` or `-postRoute`, the minimum recommended and appropriate set of parameters are automatically set. [1]

11.4.1 Useful skew

Useful skew is an interesting feature that is worth mentioning. Encounter can make use of it to improve timing. The purpose of useful skew is to use excess positive slack to help violating paths to meet timing, by speeding up the clock to the path begin point. To determine whether there is an opportunity for useful skew on the critical path, slack is measured on each side of the path endpoints. During pre-CTS optimization, you can borrow from the left side (A) of the violated path to advance the clock, see figure 30 below.

Figure 30: Example of useful skew.

Useful skew is a technique that has been available for some years now. In 65 nm, when meeting timing has become more and more difficult, all techniques that might improve timing are needed and much welcome. Optimization with useful skew is enabled by the command

```
setOptMode -usefulSkew
```

which is honored by optDesign. [2]
12 Profitability

To say in advance if a purchase of a tool will be profitable is of course hard, if not impossible. One can however, make a good estimation based on facts and previous experiences. To investigate how many, if any, engineering hours that will be saved if First Encounter is brought into the design flow, the time spent on the hardware accelerator was looked upon a little closer. The reason why the hardware accelerator was chosen for this estimation, was that the faults with its netlist were fairly easy fixable and would probably have been detectable with First Encounter.

The hardware accelerator went back and forth four times between Ericsson and the ASIC vendor, due to problems in the netlist that kept the ASIC vendor from closing timing and manufacture the chip. Most likely all of the problem areas in the hardware accelerator should have been detected using First Encounter, and hence avoided. For the sake of this estimation it has been assumed that all of the faults in the netlist could have been avoided if First Encounter would have been used.

As mentioned earlier, the change log for the hardware accelerator is not very exhaustive. Therefore, the involved designers had to provide the information in person. There was mainly one person who fixed the issues in the netlist that the ASIC vendor pointed out. He estimated that he spent two days for each of the four iterations to correct the VHDL code. There was also another person that redid the synthesis and signoff each time, and he estimated that he spent half a day per iteration. This results in:

\[(2 + 0.5) \text{ days} \times 4 \text{ iterations} = 10 \text{ days} = 2 \text{ work weeks}\]

On the one hand, if First Encounter would have been used, two work weeks worth of time and money for one engineer would have been saved here. On the other hand, using First Encounter would probably have taken at least as long. Therefore, measuring the profitability in terms of engineering hours saved and translate this into money, will not lead to positive results. The big saving here lies in lead time, which is really the most important saving. Two weeks of saved lead time translates into five million SEK according to the example below, which would definitely and by itself motivate the purchase of First Encounter. Additional lead time might also introduce the risk that Ericsson fails to deliver on time and risk missing the marketing window, and hence the whole project might be worthless, or they might miss an important contract with the consequence that the deal goes to a competitor. These kinds of effects can lead to losses in the hundred million kronor range.

There are also situations where an overshot delivery date can result in expensive fines that have to be paid for each day that the product is not delivered. A risk with long lead times is that the old and often more expensive technique is forced to be used longer. This can also lead to big losses. Consider the following example:

A new board that might be 1 000 kronor cheaper per unit compared to the old one, is delayed three months. The monthly manufacturing rate might be 10 000 units. This results in:

\[1,000 \text{ kronor} \times 3 \text{ months} \times 10,000 \text{ units/month} = 30,000,000 \text{ kronor}\]
Hence, increased time to market can cause huge economical effects, so shorter developing time and more accurate delivery dates are of great importance. A new technology can mean yearly savings between ten and one hundred million SEK for Ericsson. Buying First Encounter is an investment that might not pay off during the first project due to initial costs and learning time overhead. Ericsson will probably not save money on engineering hours plus that the tool itself costs a lot, but the big saving will be seen in lead time.

One also has to bear in mind that the floorplans that were mainly used during this evaluation were the ones that the ASIC vendor produced. Creating a good floorplan is something that takes a lot of experience and practice. With other words, just by using First Encounter does not guarantee that everything will be correct the first time around and that the first floorplan also will be the final. However, Ericsson would probably manage this and over time become as good, if not better, as the ASIC vendors at creating floorplans, given their knowledge about the inner workings of the designs.

The projects are getting more and more advanced and timing as well as power and area are factors that are becoming critical. Given this, the lead times will most likely not become shorter by themselves in the future, rather the opposite. It will probably become extremely hard, if not impossible, to design a chip that meets timing without any back-end knowledge and just by using ZWL synthesis. A tool that provides physical knowledge and estimates if a certain netlist is possible to implement in hardware or not, will probably become a necessity. As mentioned earlier, shortened lead times often results in big savings. And it is these savings that more than well will pay for the tool.

In conclusion Ericsson will most likely save money on using First Encounter. Engineering hours will probably not be saved during the project, since one or more persons have to put in about as much extra time as were previously spent on correcting errors, to run the tool. With First Encounter in the design flow, the projects will probably be ready earlier which means shorter lead time and shorter time to market, something that is always desirable. It is also here that the big savings will lie. Not only the project related costs mentioned earlier, but also the fact that more projects can be completed each year.

12.1 *Ericsson’s lead time goal*

Ericsson has a goal to cut the lead times (as of 2005) in half throughout the whole developing organization by 2008. The goal is set high and maybe not fully achievable, but the use of First Encounter in the design flow will definitely help in reaching it.
13 Recommendation

First Encounter is a very advanced and useful tool. It is fairly easy to use given a run-script. Since the runtimes when using First Encounter on large designs are quite long, it would probably be enough to use it in a prototyping manner to try out different floorplans and detect potential problem areas. It is definitely worth using on the smaller components of a design, for example like the work performed during this evaluation on the Phoenix-core and the hardware accelerator. On designs of sizes similar to the Phoenix-core, the runtimes of the tool becomes more manageable and it is possible to experiment with several different floorplans in order to find an optimal one. Later, when it is time for top-level implementation, the smaller designs can be placed as single macros. Even though runtimes are long on larger designs, the information that First Encounter offers is very valuable and will make life easier for everyone that is involved in the project. The recommendation is therefore to bring First Encounter into Ericsson's design flow.

13.1 Expensive licenses

Tools of this caliber do not come cheap. The price for one license of Cadence First Encounter is somewhere in the range of a few hundred thousand US dollars. To be able to use and work with the tool efficiently, no less than two licenses would probably be needed. The main reason for this is due to the long runtimes when further work with the tool is not possible. Since it is probably desirable to continue working and testing different floorplans while running an optimization, at least two licenses are needed. If more than one person is involved in using First Encounter, several more licenses are welcome. As a reference can be mentioned that during this evaluation, four licenses were accessible and many times all of them were used. Despite that the licenses are expensive, Ericsson would probably end up saving money in the long run by using this tool.
Conclusions

The goal with this evaluation was basically to evaluate First Encounter in order to see if Ericsson could make use of it in their design flow. Even though First Encounter is not a flawless tool that solves all problems, it gets the job done and provides valuable information. The following questions were to be answered (section 1.3) and the answers are positive.

- Is it possible to identify and prevent problems in 65 nm by using physical placement and fast prototyping?

Problems in 65 nm can be identified and appropriate measures can be taken, in time, to correct these. The experiments with the hardware accelerator in section 4 confirm this.

- Is the tool easy to use and learn?

The tool is easy to use given a run-script and fairly easy to learn and understand, especially if the user possesses some basic back-end knowledge.

- How long are the runtimes on a typical Ericsson ASIC?

The runtimes depend mostly on the size of the design, but can increase drastically if the tool runs into problems. Extended runtimes are usually a sign of that something has gone wrong. Around 30 hours is not uncommon for a standard prototyping flow on a large design.

- How many licenses are needed?

When it comes to number of licenses, two is probably a minimum. Due to the long runtimes, the work in First Encounter would progress very slowly if only one license were available. If several people will work with First Encounter then more licenses are desirable.

- How is the current design flow affected if physical placement is introduced?

The current design flow would be affected in the way that at least one person would start the work in First Encounter as soon as a first netlist is available, see the design flow in figure 1. By bringing First Encounter into the design flow, not as much time would be spent going back and redoing previous steps in the flow.

- How is the interface towards the ASIC vendors affected?

The interface towards the ASIC vendor would change in the way that a draft floorplan would be provided by Ericsson to the AISC vendor which would speed up and simplify the back-end work. Different ASIC vendors have different methods for cooperation involving First Encounter.

Cadence First Encounter is a powerful tool for prototyping that could help solve the problem of bad correlation between synthesis and physical implementation. By performing tests with different floorplans, the designers can get a preview of what the ASIC vendor otherwise would have confirmed later on in the design flow. Problem areas and pure coding errors can be found at an early stage when they are easy to correct, which saves valuable time later on in the design flow, and hence shorten the time to market.
As designs keep getting more complex and new technology nodes emerge, new approaches for chip design are needed. The traditional way of relying on ZWL during synthesis and the back-end designers to meet timing, is outdated. To narrow the gap between the chip designer and the ASIC vendor, more information must be allowed to be exchanged. The problems with multiple iterations back and forth between the parts are well known and discussed more thoroughly in section 12. A method to increase correlation between synthesis and actual results after implementation in hardware, is very much desirable, if not required. First Encounter is a possible solution.
References


[3] Leakage Current: Moore’s Law Meets Static Power, Published by the IEEE Computer Society 2003, Nam Sung Kim, Todd Austin, David Blaauw, Trevor Mudge (University of Michigan, Ann Arbor), Krisztián Flautner (ARM Ltd.), Jie S. Hu, Mary Jane Irwin, Mahmut Kandemir, Vijaykrishnan Narayanan (Pennsylvania State University):
http://www.ece.northwestern.edu/~rjoseph/ece510-fall2005/papers/static_power.pdf

[4] Leakage power graph, originally from Intel:

[5] Cadence’s website:
http://www.cadence.com/


http://vlsicad.ucsd.edu/Publications/Conferences/c125.pdf


Appendix A - Example run-script for First Encounter

```tcl
# Set variables #
set name design_name
set floorplan ../design/floorplan.fp
set config ../design/config.conf

# Create run dir, load config and floorplan #
file mkdir $name
loadConfig $config
loadFPlan $floorplan

# Start date/time #
set date [clock format [clock seconds]]
set fileId [open $filename "w"]
puts $fileId $name
puts -nonewline $fileId "START: "
puts $fileId $date
close $fileId

# Load ASIC vendor specific settings #
source vendor_script.tcl

# Set maximum number of layers to route on #
setMaxLayers integer_number

# Start date/time for N2N #
set date [clock format [clock seconds]]
set fileId [open $filename "a"]
puts -nonewline $fileId "N2N start: "
puts $fileId $date
close $fileId

# Run netlist-to-netlist optimization #
runN2NOpt -effort high -inDir $name/n2n.input -outDir $name/n2n.output -report all -saveToDesignName $name/$name\_n2n.enc
```
#### Stop date/time for N2N ####

```bash
set date [clock format [clock seconds]]
set fileId [open $filename "a"]
puts -nonewline $fileId "N2N stop: "
puts $fileId $date
close $fileId
```

#### Start date/time for place ####

```bash
set date [clock format [clock seconds]]
set fileId [open $filename "a"]
puts -nonewline $fileId "N2N start: "
puts $fileId $date
close $fileId
```

#### Set place mode and place design ####

```bash
setPlaceMode -ignoreSpare
setPlaceMode -ignoreScan
setPlaceMode -timingDriven
setPlaceMode -noModulePlan
placeDesign -inPlaceOpt
```

#### Stop date/time for place ####

```bash
set date [clock format [clock seconds]]
set fileId [open $filename "a"]
puts -nonewline $fileId "IPO start: "
puts $fileId $date
close $fileId
```

#### Save design ####

```bash
saveDesign $name/$name\_placed.enc
```

#### Start date/time for IPO ####

```bash
set date [clock format [clock seconds]]
set fileId [open $filename "a"]
puts -nonewline $fileId "IPO start: "
puts $fileId $date
close $fileId
```

#### IPO ####

```bash
optDesign -preCTS
```
set date [clock format [clock seconds]]
set fileId [open $filename "a"]
puts -nonewline $fileId "IPO stop: "
puts $fileId $date
close $fileId

optDesign -preCTS -incr
optDesign -preCTS -incr

saveDesign $name/$name\_preCTS.enc

saveModel -preCTS -outdir $name/$name\_hier_data

timeDesign -prePlace -outDir $name/timingReports -reportOnly

summaryReport -outDir $name/summaryReport

set date [clock format [clock seconds]]
set fileId [open $filename "a"]
puts -nonewline $fileId "STOP: "
puts $fileId $date
close $fileId

exit
Appendix B – Example output from the example run-script

The contents of the file time.txt located in the directory created by the script:

design_name
START: Mon Jan 08 09:13:11 MET 2007
N2N start: Mon Jan 08 09:13:12 MET 2007
Place start: Tue Jan 09 13:25:17 MET 2007
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Håkan Roos