CREATION OF STANDARD CELL LIBRARIES IN SUB-MICRON PROCESSES

Master thesis performed at the Electronic Devices division at Linköpings Tekniska Högskola
by

Emil Jansson & Torgny Johansson

Reg nr: LiTH-ISY-EX-3611-2005

Supervisor: Atila Alvandpour
Examiner: Atila Alvandpour

**Title**  
Skapande av standardcellbibliotek i sub-mikrona processer  
Creation of standard cell libraries in sub-micron processes

**Author**  
Emil Jansson Torgny Johansson

**Abstract**  
Creating an IC (Integrated Circuit) can be very time consuming if high flexibility of the construction is demanded. This report will try to solve this problem by creating own standard cell libraries, which in turn are more flexible since the user designs them. Having these libraries makes it possible to map VHDL or Verilog code to those libraries, using them instead of predefined cell libraries. The procedure of creating the libraries is quite time consuming, and thus the possibilities of making that procedure automatic, or as automatic as possible, has been examined. Unfortunately some manual labour has to be done, but the process can be speeded up a lot by making parts of it automatic.

**Keyword**  
cell library, cell libraries, automatic generation
# TABLE OF CONTENTS

1 Introduction ....................................................... 1  
1.1 Abstract ..................................................... 1  
1.2 Background ................................................... 1  
1.3 Problem ..................................................... 3  
1.4 Purpose ....................................................... 3  
1.5 Method ....................................................... 4  
1.6 Structure .................................................... 4  
1.7 Extent of master thesis ..................................... 5  

2 Standard cell libraries ........................................... 7  
2.1 Required cells ............................................... 7  
2.1.1 Logical cells .............................................. 7  
2.1.2 Synchronous cells ....................................... 8  
2.1.3 Buffers ................................................... 9  
2.1.4 Optional cells .......................................... 9  
2.2 How many cells to create ................................... 14  
2.3 Contents of cells .......................................... 14  
2.4 Required files .............................................. 16  

3 Guideline .......................................................... 19  
3.1 Creating a correct schematic ................................ 19  
3.2 Creating a correct layout ................................... 20  
3.2.1 The height of cells ..................................... 22  
3.2.2 The width of cells ....................................... 23  
3.2.3 Wiring ................................................... 24  
3.2.4 Pins ...................................................... 24  
3.3 Abstract generator and LEF .................................. 27  
3.3.1 Locating the Technology LEF-file .................... 28  
3.3.2 Creating a verilog description of the cells .......... 28  
3.3.3 Exporting the gds2-file ................................ 29  
3.3.4 Running the Abstract generator ...................... 33  
3.4 Aptivia and TLF .............................................. 39
3.4.1 Preparation ........................................ 39
3.4.2 Running Aptivia .................................... 39
3.5 Synthesis with PKS ................................. 49
3.5.1 Preparations ...................................... 49
3.5.2 Generating the symbol file ..................... 50
3.5.3 Creating macros .................................. 50
3.5.4 Running PKS ...................................... 53
3.6 Place and route in silicon ensemble .......... 56
3.6.1 Preparations ...................................... 56
3.6.2 Creating the verilog description ............. 57
3.6.3 Creating macros .................................. 59
3.6.4 Running SEULTRA ............................. 62
3.7 Backannotation .......................... 63

4 Creating a standard cell library automatically 65
4.1 Automatic layout ............................... 66
4.2 Automatic verilog generation ................. 66
4.3 Automatic LEF and abstract generation .. 66
4.4 Automatic TLF generation ..................... 66
4.5 Automatic PKS .................................... 67
4.6 Automatic place and route in silicon ensemble 67
4.7 Suggestions on a program ..................... 67

5 Conclusions ........................................ 71
References ........................................... 73

Appendix A: Script for SEULTRA
Appendix B: Our cell library
INTRODUCTION

1.1 ABSTRACT

Creating an IC (Integrated Circuit) can be very time consuming if high flexibility of the construction is demanded. This report will try to solve this problem by creating own standard cell libraries, which in turn are more flexible since the user designs them. Having these libraries makes it possible to map VHDL or Verilog code to those libraries, using them instead of predefined cell libraries. The procedure of creating the libraries is quite time consuming, and thus the possibilities of making that procedure automatic, or as automatic as possible, has been examined. Unfortunately some manual labour has to be done, but the process can be speeded up a lot by making parts of it automatic.

1.2 BACKGROUND

IC development is nowadays a huge industry. There is an almost infinite amount of consumer products like mobile phones, processors, televisions, cameras, refrigerators, ovens and cars that in one way or another uses custom IC components. Integrated circuits can provide anything from analog-to-digital conversion to digital filtering and much more.

A digital integrated circuit can be manufactured with a number of different approaches, but they all contain the same basic steps. It all starts with transistors, wiring and all the things that make up the circuit being placed in a layout, designed in a CAD (Computer Aided Design) tool and ends up with that layout being physically created on a chip. The way to create this layout differs depending on design requirements. There are three basic ways to go about.

1. **Full custom** design is when everything in the layout is created manually.
Every single transistor used can be set up as desired, optimized for speed, area or capacitive load etc. Every single wire in the layout is placed manually and the designer has total control over the layout. This is done when the design has very strict requirements and needs to be optimized in one way or another. The obvious advantage is that the layout can be created very carefully to fit the need. On the other hand, this requires a lot of work and time.

2. Semi custom design is when the designer works on a logical gate level. This means that the designer can use gates like nand, inverters, buffers, flip-flops etc. that have already been created and distributed as a cell library by a supplier. The idea is to reuse blocks of logic instead of creating them manually over and over again. Instead of placing every transistor and wire, the designer places logic blocks in the layout that correspond to the desired function. The good thing with semi custom layout is that the required time is decreased and it is far less advanced compared to full custom layout. The downside is that the possibility to optimize the given gates is very limited, so the designer loses some control of the layout. A combination of full and semi custom layout can often be a good approach, where the logical gates are created manually and optimized and then used in a semi custom layout instead of using gates created by a supplier.

3. Automatic design is similar to semi custom design in that it uses pre-created standard cell libraries. The difference is that in the automatic design approach, the layout is created automatically. The work of the designer in this case, is to describe the design in a high level programming language like VHDL or Verilog. The high level description is then fed to the automatic design tools which create a layout that corresponds to the description. This is the fastest way to create a layout of a circuit, unless it’s a very trivial one, like basic logical gates such as an inverter or a nand gate. The automatic design approach is also the one where the designer has least control of the layout. Although the design tools can be told to optimize the generated layout in certain ways, there is just no way for the designer to control exactly how it is generated. Automatic design therefore suffers from creating less optimized layouts compared to full custom design and even semi custom design. However, automatic design is a very useful tool when the design to be created does not have very strict requirements and when the time to market is more important than a fully optimized design.

That being said, this master thesis will deal with automatic design in general and the mentioned cell libraries in particular. As mentioned, the automatic design process uses standard cell libraries created by some supplier, generally
the one that has supplied the technology that is being used (0.35um, 0.18um, 0.13um and so on). While these standard cell libraries are quite flexible they may not fit the requirements. A solution to this would be to create a standard cell library manually and use that in the automatic design. This would give the designer total control of the cells in the library and still make it possible to use automatic design tools when creating layouts, thus allowing both the optimisation of the design building blocks and rapid layout creation.

1.3 PROBLEM

A problem when creating large full and semi custom made layouts is that it’s very time consuming. The advantage is that you get full control over the layout and hence it’s a very flexible solution.

The problem with creating layouts automatically from a high level language description like VHDL or Verilog, is that you get little control over the actual layout and it may become far from optimal concerning area, size and other criteria. The advantage is that it’s not as time consuming as creating a custom made layout.

This master thesis tries to combine the advantages with these two approaches so that a layout may be created from a VHDL or Verilog description, but still be flexible and optimized. This is achieved by manually creating the building blocks for the entire layout, the standard cell library, and thereby creating custom made building blocks that are automatically put together.

The main problems however, are to find a way to make a standard cell library and then make it so the standard cell library created can be integrated into the available tools for automatic design. This means far more than just creating a number of logic gates as there are a multitude of requirements that a standard cell library and the cells themselves in that library must fulfill.

However, the information about how to create a standard cell library is proprietary since companies are selling standard cell libraries and have no interest in giving away this information for free.

1.4 PURPOSE

The purpose of this master thesis is to examine the possibility of making a standard cell library, from which any given design can be created in an auto-
matic design process. The main purpose is to manage to create a standard cell library manually and to integrate that library into a commercial tool for automatic design.

This master thesis will also discuss the possibility of creating the standard cell library itself automatically, hence delivering a solution that will be able to both optimize a design and have it created totally automatically.

1.5 METHOD

The information about standard cell libraries is seldom public. The companies making money in developing cell libraries simply like to keep it a secret. Therefore backward engineering on standard cell libraries has been the way to learn about their structure and what they actually consist of. This means that available standard cell libraries have been studied and tested to get a grip of their functionality and what is used and needed. Some information has been obtained from Cadence newsgroups or the Cadence online documentation (Cadence being the suit of tools used for the creation of layouts and schematics, simulations and similar).

The newsgroup is not an interaction between you and Cadence employees, but simply an interaction with other people familiar with Cadence. Since the University of Linköping does not have a support deal with Cadence, no contact or help from them has been available. In other words, information has been difficult to gather and when errors occur, patience and imagination is the way to solve them.

In addition to backward engineering, trial and error has been applied a lot. This has in many cases been the only way to solve a problem, since information about it hasn’t been available in the public domain.

1.6 STRUCTURE

There are five major steps in creating and testing a standard cell library, and each of them is discussed thoroughly in their respective section. Problems and solutions are also presented as well as a complete guideline on how to create a standard cell library and use that library in an automatic design process. The five major parts are:
1) Create a correct schematic and layout
2) Create abstract and LEF file (Library Exchange Format, used to describe the cells in a standard cell library)
3) Create a TLF file (Timing Library Format, used to characterize the cells in a standard cell library)
4) Synthesis
5) Place and Route

An overview of the design flow is presented in Figure 1.1.

![Figure 1.1: Overview flow chart.](image)

The flow chart in Figure 1.1 will be divided into sub-diagrams in its respective sections, as some of the parts contain several different tasks.

In addition to these five parts, a discussion about the possibility of making these five steps automatic will be presented as well as a part that describes what is needed in a standard cell library.

### 1.7 EXTENT OF MASTER THESIS

This master thesis is limited to describing the creation of a standard cell library consisting of a few standard cells. The small number of cells does not mean that the cell library will not be complete, only that its flexibility will be somewhat limited.

This master thesis will only cover standard cell library construction using
0.35um technology. It is however possible to change the technology with only minor changes in the procedure. All the examples of locations of files, sizes etc. are with regard to the 0.35um technology supplied in the Ams HitKit-3.60b tool.

To read and understand this master thesis, no prior knowledge is required. However, in order to follow the guideline and actually create a standard cell library, basic knowledge about the Cadence tool and full custom design is required as the details of how to create a layout are not a part of this report.
STANDARD CELL LIBRARIES

Cadence works with several standard cell libraries. A cell library consists of different cells, such as logic gates, transistors and pads etc. Each cell in turn consist of different views. These views are used for different purposes. The schematic view, for instance, allows the designer to make a little less detailed construction before starting the layout. The construction on the schematic level can then be used for simulation, and once it is confirmed that the construction is correct, the layout can be constructed. This can be made at even a higher abstraction level by using Verilog.

2.1 REQUIRED CELLS

This part describes what cells that are needed in a standard cell library and the function of those cells.

2.1.1 LOGICAL CELLS

A cell library must be able to create any logical expression possible that might be needed in a synthesis of any design. To do this a number of cells with logical functions is needed. There are plenty of functions that can be implemented e.g. AND, NAND, OR, NOR, XOR and NOT. However, only a few of these are required in a minimum cell library since several functions can be created from a combination of other functions. For example:
Chapter 2 – Standard cell libraries

As shown in Table 2.2, the function \textit{NOR} can be replaced by a number of \textit{AND} and \textit{NOT} functions. Further, \textit{AND} can be replaced with a \textit{NAND} followed by a \textit{NOT} etc.

As described, all logical expressions can be created with a few logical cells, however, the construction will be much more complex and contain many more cells when just using a cell library consisting of a few cells. The minimum requirement for a standard cell library is simply enough cells to be able to create all logical expressions possible, which is fulfilled with just a nand-cell. It’s even possible to create all possible expressions using nothing but a \textit{NOR}-cell but as mentioned, it can be more area-effective to create a number of logical gates so that the place and route tool doesn’t have to create them itself.

It is also wise to create several different cells implementing the same function but with differently sized transistors so that their different fanout load limit, speed and area can be used by the synthesis tools when optimizing a design in respect of these different requirements. This however is not required for a minimum, working cell library.

2.1.2 SYNCHRONOUS CELLS

A cell library also needs synchronous cells so that counters, registers and other cells dependent on a clock signal can be created. Therefore a flipflop must be created, for example a D-flipflop (DFF). The DFF must have complementary output and support \texttt{enable}, \texttt{reset} and \texttt{preset} control signals so that all possible functionality is supported.
2.1.3 BUFFERS

At least one buffer has to be created in the cell library. This is required so that the synthesizer can amplify signals and also delay signals if there are timing constraints requiring that. The buffer should also be a tri-strate buffer so that several buffers can be attached to a databus or similar. This is needed to be able to control the buffers to not output to the bus all at once but only one at a time. As with the other cells it is good to have several different buffers with different driving capabilities and delays to give the synthesizer the option to choose the buffer that is most suitable for the current design.

2.1.4 OPTIONAL CELLS

There are a couple of types of cells that don’t provide extra functionality of the cell library. These cells contribute in a way that makes the placed and routed design demand less manual work after it has been created. These cells are filler cells that help fill the unused area in the design and capacitance cells that add decaps to the design, which otherwise would have had to be added manually.

CAPACITANCE CELLS

At the beginning and ending of each row created, the place and route tool can automatically add a capacitance cell. The only things that a capacitance cell consist of are power and ground bars and rows of n-well and p-substrate contacts as Figure 2.3 shows.
CORE FILLER CELLS

In the unused area in between the core cells that’s left when the place and route tool has placed all the functional cells, filler cells can be placed to fill up the empty space. This is done to add as much coverage as possible automatically and avoid design rule violations. The filler cells also function as decaps. Several differently sized filler cells should be created so that the tool can place larger filler cells in large open spaces and smaller in the smaller open

Figure 2.3: Layout of an ending capacitance cell.
spaces, so that as few filler cells as possible are used, decreasing the complexity and amount of cells used in the final design.

The sizes of the filler cells is technology-dependent. The smallest filler cell should have the minimum width possible to still fulfill all the requirements of a layout. As discussed in more detail in section 3.2, the width of the cells should be a multiple of the spacing in horizontal direction of the routing grid (yPitch). The minimum multiple is 1, therefore the smallest filler cell should have the width of yPitch.

![Minimum sized filler cell](image)

**Figure 2.4**: Minimum sized filler cell.

The minimum sized filler cell in Figure 2.4 doesn’t contain much, and its only function is to relay the ground and power bars in the spaces where it is placed. The only things in the minimum sized filler cell are a ground and power bar, not even n-well and p-substrate contacts.

With only this filler cell no coverage at all would be added to the design so
more filler cells are needed. The next filler cell should be twice as wide as the minimum sized one and contain two transistors, adding coverage and capacitance. One pmos and one nmos transistor are used where the drain of each transistor is connected to the gate of the other one as the schematic in Figure 2.5 shows.

Figure 2.5: Schematic of a filler cell.

However, there is no use in creating schematics of the filler cells unless it is desired to be able to import the final design back into Cadence and testing it with LVS (Layout Versus Schematic). If that is desired then schematics must be created so that the schematic and layout can match. More about this in section 3.6.

Figure 2.6 shows the layout of the filler cell. It is desirable to make the filler cell as compact as possible without breaking any of the layout constraints. These constraints are discussed in detail in section 3.2.
Chapter 2 – Standard cell libraries

Figure 2.6: Layout of filler cell.

The two filler cells described are sufficient for the desired function, but additional larger cells would have some advantages as mentioned earlier. If more filler cells are created they should have the same structure as the one in Figure 2.6. The difference is the size, where the width should increase with a multiple of the yPitch value mentioned earlier, for each additional filler cell that is added. The width and length of the transistors in the cells should increase as well, covering as much area as possible.
2.2 HOW MANY CELLS TO CREATE

As mentioned, only a few cells are required to create a complete standard cell library. However, for the cell library to be somewhat useful, several cells are needed. It’s important to create several differently sized versions of each cell to allow different speeds, loads and driving strengths so that the library is flexible and able to optimize designs and worst paths in designs. Also, although a simple NAND cell is enough to create logical expressions, it is often better to use for example a custom made AND cell instead of connecting several NAND cells to get the same function. Both area and speed can improve if this is done. Therefore it is good to have several logic cells.

The question that arises, then, is how many cells are sufficient? How many different drive strengths are enough for a cell? How many different logic cells are optimal, considering effort required to create them and what gain an extra cell produces. Research by Nguyen Minh Duc and Takayasu Sakurai [1] has shown that a cell library consisting of no more than 20 well chosen cells only has 2% more delay than a cell library of 400 cells. A low number of cells decreases synthesis time and also development time.

2.3 CONTENTS OF CELLS

The cell library contains, as previously mentioned, a number of different cells. These cells in turn contain several files and below is a listing of the files, as well as a brief information about them, required for a cell in a standard cell library. How to create these files and more detailed information about them is available in chapter 3. The contents of the cells are presented below.
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layout</td>
<td>The layout is a file containing the physical layout of a design. This is where transistors, wiring etc. are placed and the layout is a digital representation of what will actually be physically placed on a chip. The layout is needed in order to make a LEF-file and an abstract. Some special constraints, in addition to those when creating an ordinary layout, need to be fulfilled. These constraints will be presented later on in chapter 3.</td>
</tr>
<tr>
<td>Schematic</td>
<td>The schematic is a high level description of a design. Just as in the layout, the schematic contains transistors and wiring, but only as symbols representing them. The schematic is needed in order to generate the TLF-file for the cell library. It needs to be imported into Aptivia, which is the tool used to create TLF-files, and it might as well be useful if an LVS (Layout Versus Schematic) is preferred. The schematics that describes the cells have to be created manually, while the schematic describing the desired construction can be produced automatically by importing a Verilog netlist created by the synthesis tool.</td>
</tr>
<tr>
<td>Abstract</td>
<td>The abstract is a simplified view of the layout. The abstract file, generated by the abstract generator, is used during place and route and it is also from the abstract views that the LEF file is generated. It is actually the abstract file that is being placed in the place and route, rather than the layout. The abstract view is a view quite similar to the layout view.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>Logical</td>
<td>This is simply a high level description of the cells in the library. It is written in the program language Verilog and only contains a brief description declaring the inputs and outputs of the cells. This file is used in the abstract generator.</td>
</tr>
<tr>
<td>LEF</td>
<td>LEF stands for Library Exchange Format and there are two different kinds of LEF-files that are being used. One is supplied by the technology, referred to as the technology LEF, and one is generated through the Abstract generator. The first one is used to create the latter one, which in turn is used both in the synthesis tool PKS (Physically Knowlegable Synthesis) and the place and route tool SEULTRA.</td>
</tr>
<tr>
<td>TLF</td>
<td>TLF stands for Timing Library Format and describes timing and power constraints for the cell. The TLF is generated by a program called Aptivia. A TLF-file is needed in order to synthesize and to place and route. Among other things, rise and fall times, leakage power and different transition times can be included in the TLF-file.</td>
</tr>
</tbody>
</table>

**Table 2.7: Contents of cells.**

### 2.4 REQUIRED FILES

In order to make a standard cell library, some other files are needed as well. These files are not distributed on the cell library like the files in section 2.3 are, however they are needed in the creation of the cell library. Below follows a listing of those files and short descriptions of them. How to create these files
and more information about them is available in chapter 3.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDSII</td>
<td>The GDSII format is a format Cadence uses in order to import and export layouts. An exported GDSII of the cells in the library is used in the Abstract generator where the abstract and LEF-file are generated. The GDSII format is also used to save the automatically generated layout from the place and route tool SEULTRA.</td>
</tr>
<tr>
<td>EDIF</td>
<td>The Electronic Design Interchange Format (EDIF) is a non-proprietary, standard interchange format that uses text to describe electronic design data. A schematic is exported from Cadence as an EDIF in order to create a SYM-file, which is described below. The EDIF contains information on all the cells in the library.</td>
</tr>
<tr>
<td>SYM</td>
<td>The SYM-file is then used in the synthesis tool PKS. It contains information about the cells in the library so that PKS can create symbols that are used to represent a schematic of the synthesized design.</td>
</tr>
<tr>
<td>Verilog</td>
<td>This is a more detailed Verilog description than the logical one used in the Abstract generator. Large parts of this file are the same as in the logical created earlier, but this file extends the logical with information about various transitions in the cells. It is used by the place and route tool SEULTRA.</td>
</tr>
<tr>
<td>SYN</td>
<td>The SYN-file is generated by Aptiva and then converted to a TLF-file. In other words, the TLF-file doesn’t contain any extra information compared to the SYN-file.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Ioplace.ioc</td>
<td>This file is used in order to describe the pin placement of the final layout. It describes the direction and placement of the pins, as well as the pin names. The name of the file is arbitrary, as long as it is correctly referred to in the place and route run macros. The ioplace.ioc file is used in the place and route tool.</td>
</tr>
<tr>
<td>GCF</td>
<td>General Constraint Format (GCF) is a file format generated by PKS, and later on used in SEULTRA. This file contains information about the constraints set in the synthesis, like fanout load limit, voltages, temperatures and clock characteristics. Another GCF-file is needed as well and needs to be created manually. This second GCF-file is used in the place and route tool to import the required TLF-files.</td>
</tr>
<tr>
<td>MAC</td>
<td>This is a macro file for SEULTRA. Although the operations can be made manually, a macro file really helps out a lot. It can be executed from the command line or from inside SEULTRA. The MAC file provides different constraints for the place and route.</td>
</tr>
<tr>
<td>TCL</td>
<td>This is a macro file for PKS. As with the MAC, the operations can be made manually.</td>
</tr>
</tbody>
</table>

*Table 2.8: Required files.*
GUIDELINE

This chapter contains the guideline of how to create a standard cell library. Every section will show where in the flow chart in Figure 1.1 the process is. Some parts in the guideline contain subparts which will be shown in extended flow charts in its respective sections.

3.1 CREATING A CORRECT SCHEMATIC

The creation of the schematic is straightforward. The only thing that might be different from when creating a cell for a full custom design is that pins should not be used for power and ground. Instead global power and ground should be used for those nets.

Create the schematic and then add the global power and ground. How to do this differ in different technologies, but in the 0.35um technology, a library called Gates contains a VDD and a GND cell which should be instanciated in the schematic and then connected to power and ground of the transistors respectively. An example schematic of an inverter is shown in Figure 3.9.
3.2 CREATING A CORRECT LAYOUT

In addition to the general rules of creating a layout there exists a few more rules in order to make the layout work through the entire process. These rules allow the Abstract generator to understand the layout.

At first, the size of the layout has to be a multiple of yPitch horizontally and xPitch vertically (by default 1.3um and 1.4um respectively). However, there are ways of getting around this problem and it is recommended to exclude...
some layers from these constraints. The solution will be presented in the section about Abstract Generator, section 3.3.4.

Another problem is that the pins (actually the metal layer the pins are placed on) have to be on the crossing of a x-pitch y-pitch grid, thus making it impossible to place the pins arbitrarily.

In order to make the Abstract Generator understand where the pins are on the layout, labels have to be added. It is recommended that the label layer is set to PIN m1, but any layer can be used.

Figure 3.10 shows a standard cell in 0.35um technology. Rulers have been placed in the figure to show how the cell conforms to the given constraints that have to be fulfilled in order for the layout to work with the place and route tools.
3.2.1 THE HEIGHT OF CELLS

The height of the cells should be measured between the top of the power bar and the bottom of the ground bar. In Figure 3.10 the layout has a height of 18.2 um which is 14 * xPitch in 0.35 um technology since the xPitch there is 1.3 um. The reason for using the xPitch value instead of the yPitch value when determining the height is that the xPitch stands for the distance between the horizontal routing wires. Thus the cell height must be a multiple of that value. Wiring is further discussed in section 3.2.3.

All the cells in the library must have the same height so that the place and
route tool easily can put them together in rows without complex routing of the power and ground nets. It also allows the place and route tool to “flip” cells so that the same power/ground wire can feed cells both side of it as illustrated in Figure 3.11.

![Nmos of first inverter](image)

**Figure 3.11**: Two inverters using the same ground bar.

### 3.2.2 THE WIDTH OF CELLS

The width of the cells should be measured from the left side of the power and ground bars to the right side of them. The width must be a multiple of the yPitch value in the given technology. Unlike the height, the width of the cells do not have to be the same for each cell as long as they conform to this constraint. Metals besides the ground and power bars should not be placed too close to the edge of the cell. For example, a wire in MET1 should be placed no closer to the edge of the cell than half the minimum MET1 to MET1 distance. If they are there is a risk that the synthesised design will not follow all DRC rules because of wires ending up too close to each other.
3.2.3 WIRING

Depending on the definitions in the technology wiring can be done differently. The direction of MET1 and MET2 is determined in the technology files as horizontal and vertical. This means that the place and route tool will use MET1 for horizontal routing and MET2 for vertical routing which means that the standard cells preferably should use the metals in the same way, that is, use MET1 for horizontal wires and MET2 for vertical wires. Otherwise the routing will be more complex for the place and route tool, but the tool is flexible and can use the metals for routing in other directions as well if it’s possible. Other metal layers like MET3 and MET4 should preferably not be used in the standard cells so that the place and route tool can draw wires in those layers freely.

3.2.4 PINS

The pins must be placed at an intersection of a multiple of the xPitch and yPitch. That is, the pins must be placed on a xPitch*yPitch grid. This can be tricky since there is no way in Cadence to show a grid with different space in the x- and y-direction. The actual pin itself does not have to be on this grid, but some point of the metal it’s on must.

![Figure 3.12: Closeup of the input and output pins.](image)
In Figure 3.12 above, which is a closeup of the pins of the inverter in Figure 3.10, two different approaches of how to place the pins are shown. They are described in the next section.

**INPUT AND OUTPUT PINS**

The input pin a has been placed on a square of MET1 that covers an intersection of the correct grid. Although not the recommended method, this is a fast and easy way of creating the pins since it is simple to get a plate of MET1 to cover an intersection.

On the other hand, the output pin a_bar is put on a standard size MET1 wire where the wire has been placed so that it intersects the grid. This gives a cleaner layout but demands a bit more work. The recommended approach is to carefully measure an intersect point with rulers and then place the wire there.

The pins have to be placed like this because of the routing in the place and route tools where routing of the metals is done using this grid as shown in Figure 3.13.
Over each input and output pin a label must be placed. Pin attributes aren’t saved when the layout is exported to a GDSII out and therefore the labels point out their location. It’s recommended to use a layer like PIN m1 for the labels but any layer can be used.

**POWER AND GROUND**

There are no pins created for the power and ground bars. Instead only labels are placed over them in the same fashion as with the input and output pins. However the ground and power labels must have a global name which is accomplished by adding an exclamation mark after the name. The names vdd! and gnd! are recommended because of their intuitive names, but any name can be used as long as it’s the same in all of the cells and the same name is used everywhere else in the creation of the standard cell library as well. The reason for this is that it would be hard to describe ground and power pins in the high level language (VHDL or Verilog for example) that will be used to

---

**Figure 3.13:** The MET1-MET2 routing grid.
describe the design. That would mean that the input pins for ground and power would not be included in the synthesised design which means that if you would want to import the finished design into Cadence for checking functionality the input pins for ground and power would not be connected to anything in the individual cells that make up the schematic. The schematic would need a lot of work to match the layout, bringing extra work and problems of verifying the functionality.

3.3 ABSTRACT GENERATOR AND LEF

The purpose of using Abstract Generator is to generate an abstract view of the cell and a LEF-file corresponding to the cell library. The abstract view is in turn used in Silicon ensemble during place and route, and so is the LEF-file. There are two different kinds of LEF-files, one that describes the technology and one that describes the library cells. The latter one is the one created by abstract generator. Since this LEF describes all the cells in the library, all those cells have to be processed simultaneously in abstract generator.

Before starting to work with the abstract generator a correct layout must have been created where all constraints are met. Secondly, the technology LEF-file and a layer map table must be located as it is needed in the process. Finally a Verilog description of the cells has to be created and a gds2-file representing the layout has to be exported from Cadence.
3.3.1 LOCATING THE TECHNOLOGY LEF-FILE

The technology LEF-file should be distributed along with the technology that is being used. It can be found in a sub directory to the directory of the technology. For example, when using AMS HitKit-3.60 for 0.35 um technology, the LEF-file can be found in ~ams_hit-3.60/artist/HK_C35/LEF/c35b4/c35b4.lef.

3.3.2 CREATING A VERILOG DESCRIPTION OF THE CELLS

The verilog file describing the cells has a simple syntax naming the cell and the pins and also indicating the direction of pins. The name of the verilog file does not matter but an intuitive name like <library name>.v, where <library name> is replaced with the name of the library that is described, is recommended. An entry of a nand cell with two input pins named a and b and an output pin named out would look like this:

```
module nand2 (a, b, out)
input a, b;
output out;
endmodule
```

Table 3.14: Example of a Verilog description.

Note that there are no ground or power pins specified in the verilog file since there are no such pins present in the layout where global names have been used instead.

Filler cells and cap cells do not have to be included in this logical description since they have no input or output pins.
3.3.3 EXPORTING THE GDS2-FILE

To start the export chose *File -> Export -> Stream* in the Cadence icfb window as shown in Figure 3.15 below.

![Figure 3.15: Export stream.](image)

A new window will open in which the layout to be exported must be chosen. Leave the settings as in Figure 3.16 replacing `<library name>` and `<cell name>` with the actual library name and cell name of the layout that is supposed to be exported.
Choose *User-Defined Data* and in the following window, shown in Figure 3.17, note the path to the *Layer Map Table* as this will be used later in the abstract generation process. Click ok to close the *User-Defined Data* window.
Next click on *Options* and in the window that follows make sure that *Convert PCells to Geometry* is selected as shown in Figure 3.18.
Figure 3.18: Export stream options.

Click Ok in the options window to save the settings and then Ok in the main export stream window to start the export.

If the process finishes without errors the next step is the Abstract generator.
3.3.4 RUNNING THE ABSTRACT GENERATOR

To start the abstract generator, enter:

```
abstract &
```

in a terminal window.

CREATE A NEW LIBRARY

Choose File -> Library -> New. In the dialog that opens choose the directory where to place the library, enter a name for the library and then click ok. A couple of errors complaining about undefined metal layers may appear, but that will be fixed once we import the technology LEF-file.

IMPORT REQUIRED FILES

The abstract generator needs to import a technology LEF-file, the verilog description of the cells and the gds2-files containing the layout of the cells.

To import the LEF choose File -> Import -> LEF. Click the Browse button, locate and select the LEF-file to use then click on Ok to import the file.

Next, choose File -> Import -> Stream (GDSII).
Choose the gds2-files to import and in the Layer Map Table field enter the path to the layer table map that was noted in section 3.3.3 as illustrated by Figure 3.19 and then click ok to import the files.

Finally the verilog description must be imported. Choose File -> Import -> Logical. Enter the path to the verilog file created in 3.3.2 and then click ok to import it.

The files should now have been listed in the abstract window and after every entry there should be a green v in the columns for Layout and Logical. If some of the cells imported are filler cells or cap-cells, then these will not have a mark in the Logical field since they are not included in the verilog description because they have no input or output pins. In Figure 3.20 endcap1, endcap2, fill1 and fill2 are cap cells and filler cells and thus they do not have a mark in the Logical field.
First, all of the cells must be selected. Do this by choosing **Cells -> Select all**.

Next, choose **Tools -> Pins**. In the window that opens, under the **Map** tab, the layer of the label of the pins in the layout will be mapped to the layer of the actual pins. If the labels are in the **PIN m1** layer and the actual pins are in the **MET1** layer, setup the fields as shown in Figure 3.21. If the label or the pins are in different layers then adjust accordingly.

**Figure 3.20**: Abstract generator after all files have been imported.

**PINS**

First, all of the cells must be selected. Do this by choosing **Cells -> Select all**.
The power and ground pin names have already been defined in their respective cells as regular expressions. If other names have been chosen, simply replace the regular expressions with the correct names.

The output pin names must be defined manually and all of the output pin names should be typed in the Output pin names. That is, if one cell has an output named Out and another cell has an output named Q for example, both names must be typed in this cell. If several cells have the same name of their output pins, that name only has to be typed in once.

After the fields have been filled, click the Run button to make abstract generator match the pins.

In the main window some warnings may claim: prBoundary does not enclose all cell view geometry. Specify any missing layers in the Using geometry on layers field. This is because the NTUB layer in the layout results in a cell width and height that is not a multiple of the xPitch and yPitch as discussed.
earlier. This is fine as long as the boundaries of the top left corner of the power bar and the bottom right corner of the ground bar are multiples of those values. Warnings like these lead to an exclamation mark in the PINS column but as long as no errors occur it's ok.

Warnings saying that the cell has no input pins or logical view will occur on the filler and cap-cells which is expected since those layouts do not contain any pins and they are not present in the logical verilog description imported earlier.

If errors do occur saying that there are no pins on the X-Y grid the pins in the layout are not placed correctly and must be fixed as discussed in earlier.

**EXTRACT**

Choose *Flow -> Extract*. No changes need to be made in the window that opens so simply click *Run* to start the extraction process. No errors or warnings should occur. If they do, fix the layout according to the errors.

**ABSTRACT**

It is now time to create the abstract itself. Choose *Flow -> Abstract* and click the *Overlap* tab in the window that opens. If the power and ground bars are the utmost extremities of the layouts then choose *Off* in the *Create overlap boundary* dropdown menu. This will create a rectangular border around the cell where no other cells can intrude in the place and route process.
Figure 3.23: Overlap settings.

If the power and ground bars are not as wide as the widest parts of the cell an overlap needs to be created. That is, if there for example are metal bars “sticking out” on the sides, this has to be accounted for so that the place and route tools can place the cell as close to other cells as possible and also make it overlap the imagined rectangular border of other similar cells. So if this is the case in any of the layouts, choose As needed in the Create overlap boundary dropdown menu. If not, the rectangular border mentioned above will enclose the entire cell, wasting a lot of space when placed since no cells can overlap if the overlap properties have not been set.

When done, click Run and the abstract generation process begins. No errors or warnings should occur.

EXPORT LEF

All steps required to create the LEF-file have been completed and it’s time to export the LEF. Choose File -> Export -> LEF. Type the name of the library in the LEF Filename field as shown in Figure 3.24, replacing <libraryname> with the actual name wanted, without spaces.

Figure 3.24: Export LEF.
Click ok to export the LEF-file and then close the abstract generator. A LEF-file describing the standard cell library has now been created.

3.4 APTIVIA AND TLF

In order to add timing constraints to the construction, a TLF-file has to be generated. This file describes all the different timing aspects and the power constraints of the cell. An example of what it might contain is rise and fall times of transitions, input capacitances on pins to the cells, and so on. This TLF is later on used by Silicon ensemble for place and route. In order to create this file, a program called Aptivia is used.

Aptivia requires a schematic for each cell in order to generate a SYN-file, which later on is converted into a TLF. This is done by issuing the command syn2tlf at the command line. However, it might complain about some errors and it might be necessary to manually change the SYN-file.

3.4.1 PREPARATION

First of all a schematic of the cell to be characterized must have been created. Secondly the simulator include directory must be located as well as the model library of the selected technology. In the case of the 0.35um technology, the include directory is `/ams_hit-3.60/spectre/c35/` and `/ams_hit-3.60/spectre/c35/cmos53.scs` respectively.

3.4.2 RUNNING APTIVIA

To start aptivia, open up the schematic of the cell to be characterized in Cadence. Choose Tools -> Aptivia in the schematic window. A new menu
called Aptivia will appear to the right in the menubar. Choose *Aptivia -> Create test*.

**CREATING A TEST**

In the dialog that appears, choose *Create new workspace* and click *Ok*. In the new dialog type in a name for the project and click *Ok*. Next, a prompt will ask *Would you like to base your project on a previously created project?* Select *No* and click *Next*. In the following dialog enter a description of the project in the *Description* field and then click *Finish*. 

![Create Test dialog]

- **Test name:** inverter
- **Simulation:** Spectre
- **Test type:** Netlist
- **Design location:**
  - **Library:** manualLibrary
  - **Cell:** inverter
  - **View:** schematic
- **Import simulation control:**
  - **No import**
  - **From Analog Artist**
  - **Schematic:**
A dialog like the one shown in Figure 3.25 appears. Check that the correct design has been chosen and then click Ok.

A Spectre test setup window opens. Choose the Includes tab and enter the information about the include directory and model library gathered during the preparations.

![Figure 3.25: Create test.](image)

The path to the include directory should just be typed in as is, but for the model library the correct section of the file must also be defined. Therefore,
type the path to the file followed by a space and \texttt{cmostm}. This example shows the setup when using 0.35 um technology and the section name might be different in another technology. To find out what it is, open the include file in a text editor and check the top of the file where the section name should be found.

Next, select the \textit{Analyses} tab. Here a variety of analyses to be performed on the design can be chosen. Select \textit{Transient} and select \textit{Enable transient analyses}. Enter a simulation time and choose \textit{Conservative}, \textit{Moderate} or \textit{Liberal} depending on how accurate the calculations of the simulation should be. \textit{Liberal} is the least accurate but the fastest. \textit{DC, AC, Noise} and other analyses can also be chosen here if wanted.

![Figure 3.27: Transient analyses.](image)

Choose the \textit{Sim options} tab and in the \textit{tnom} field type desired nominal temperature, that is the default operating temperature of the design. The simulator will then simulate in other temperatures surrounding this nominal temperature.
Click *Ok* to save these settings.

**CREATE A MODEL GENERATOR**

In the main Aptivia window, select *Model Generator* in the treeview to the left. Right-click it and select *New model setup.*

![Figure 3.28: Create new model setup.](image)

A new dialog will open. Type in a name in the *DCM name* field and make sure that *Project test* is selected and that the path entered points to the current design.
Next click *Ok* to start the DCM.

In the *Design* tab in the DCM window, click *Browse* and locate the library and schematic of the cell that should be used then click the *Function* tab.
Figure 3.30: Setting up the function of the cell.

Click on the *Function* dropdown menu, choose *Digital* and then choose the cell that matches the cell that should be characterized. In Figure 3.30 above an inverter is used, thus *Not* has been chosen from the dropdown menu. In the dropdown menus next to the pin labels, choose the corresponding type of each pin. That is, for an input pin select *Input*, select *Output* for an output pin and select *Vdd* and *Vss* for power and ground pins respectively. Also make sure that the voltage levels next to the power and ground pins have the correct values. Nothing on the other tabs needs to be changed, but it is possible to customize things like sweep steps, time steps etc. On the *Defaults* tab one might want to change the value of the *Output load* field so that the output load on the design isn’t larger than it is supposed to handle.

When done, click *Apply*. Do not close the window since the next step also will use it.
GENERATE AND RUN

Everything is now set up and all that is left to do is to run the actual simulations.

In the DCM-setup window that should still be open, click Generate and Run. Click Ok in the dialog that follows, confirming what is supposed to be generated. The simulations will start and they will take quite a while, depending on the length of the simulation stop time and the complexity of the design. A window like in Figure 3.31 will open, showing the status of the analyses.

When the status window says Plan run finished and everything has passed, click the Close button.

EXPORT THE SYN FILE

The simulation has finished and it is time to export the created Synopsys library file, or SYN file, which will then be converted into a Timing Library.
File or TLF. This step should not be done until all cells in the standard cell library have been analysed and simulated since all of the results from the different cells are wanted in the SYN file.

In the DCM setup window choose Tools -> Generate Synopsys library.

![Figure 3.32: Generate Synopsis library.](image)

In the Library file field, type a filename describing where the library file is to be saved. Next type the name of the library in the Library name field. Leave Input libraries as is to include all of the results from the simulations of the different cells.

Click Ok to export the SYN file.

**CONVERTING SYN FILE TO TLF FILE**

To convert the SYN file generated from Aptivia a command line utility called syn2tlf is used. Type:

```
syn2tlf <SYN file>.lib
```

replacing `<SYN file>` with the name of the file exported previously, to convert the SYN file to a TLF file. The TLF file created will get the same name as the SYN file but with `.tlf` as file extesion instead of `.lib`.

However, the `syn2tlf` program is flawed and if errors occur they have to be corrected manually. For example, the program might complain about missing
variables like `nom_voltage` and `nom_temperature`. If so, open the SYN file in a text editor and locate the top section of the file where some other variables are initialized and add:

```plaintext
nom_voltage : 3.3;
nom_temperature : 27;
```

Replace the values with the correct values for the technology/design used. After these changes, run `syn2tlf` again.

Another error is that `syn2tlf` seems to have problems with certain syntaxes of the SYN file. If cells with a clock are used, for example, `syn2tlf` will not understand the description of the clock pin in the SYN file. The solution is to replace those constructions with ones that `syn2tlf` understands. For example the SYN file tries to group `min_pulse_widths` values like this:

```plaintext
pin(clk) {
  direction : input;
  capacitance : 0.01716775 ;
  min_pulse_width () {
    constraint_low : 0.23407 ;
    constraint_high : 0.722961 ;
  }
}
```

**Table 3.33:** Example of grouping in SYN-files.

With the example above, `syn2tlf` will output the following error:

`Fatal! Missing when or sdf_cond attributes in the min_pulse_width`

If this occurs replace the entire `min_pulse_width` group in the example above with:

```plaintext
min_pulse_width_low : 0.23407 ;
min_pulse_width_high : 0.722961 ;
```

Save the SYN file and run `syn2tlf` again. If no fatal errors occur, the generation of the TLF file is complete.
3.5 SYNTHESIS WITH PKS

The tool used for synthesis is called PKS or Physically Knowledgeable Synthesis. The PKS puts the construction together by using the cells defined in the created cell library. This is done on a schematic level, so it is not the layouts that are mended together. There is a lot of manual work to be done here, so using a script file is highly recommended.

The PKS generates a Verilog netlist that is to be used in Place and Route. It also generates GCF (General Constraint Format) files, which also will be used in place and route. These files contain timing constraints, parasitic constraints, area constraints and power constraints.

3.5.1 PREPARATIONS

Before work with PKS can begin there are a couple of prerequisites. First, a TLF-file must have been generated as described in section 3.4. Secondly, a symbol file containing descriptions of the cells in the library for PKS to use as symbols in the generated schematic is needed. A couple of macros are also convenient, one for setting up PKS and one to actually run the commands needed for the synthesis.

Obviously the most important preparation is to create the actual design that is supposed to be synthesized. This can be done in either VHDL or Verilog. How to code Verilog and VHDL will not be further discussed here but an example VHDL code of a 4-bit counter is shown in Table 3.37.
3.5.2 GENERATING THE SYMBOL FILE

To generate the symbol file, in the Cadence icfb window choose Tools -> Export -> Edif 200. In the Library name field, type the name of the library. Observe that there should be no other cells than the standard cells in the library when this export is done. No testbenches or temporary cells should be in the library as they will be included in the symbol file if they are present.

Next, a command line utility called edifconv is used to convert the edif file that was just exported to a symbol file. Type

```
edifconv edif.out
```

in a terminal to convert the file. If a different name than `edif.out` was chosen in the export, change accordingly. This command creates a file called `sym-lib.sym` that will be used later in PKS. Change the name to `<library name>.sym` or similar.

3.5.3 CREATING MACROS

After starting PKS, the program has to be told where to find the required files, for example the TLF-file and the symbol file. Although it’s possible to do this
by running commands in PKS directly, it’s generally a good idea to use macros for this, as macros can be reused, thus saving time instead of typing every command manually every time.

In Table 3.34, an example macro for setting up PKS is provided.

```
# Setup of technology libraries
set AMS_DIR $env(AMS_DIR)
read_tlf <full path to your tlf-file>.tlf
read_tlf $AMS_DIR/ambit/c35_3.3V/c35_IOLIB_4M.tlf
read_tlf $AMS_DIR/ambit/c35_3.3V/c35_IOLIBV5_4M.tlf
# please use the following lines for 3bus cells
# read_tlf <full path to your tlf-file>.tlf
# read_tlf $AMS_DIR/ambit/c35_3.3V/c35_IOLIB_3B_4M.tlf
read_symbol <full path to your sym-file>.sym
read_symbol_update $AMS_DIR/ambit/c35_3.3V/c35_IOLIB_4M.sym
# please use the following lines for 3bus symbols
# read_symbol_update <full path to your sym-file>.sym
# read_symbol_update $AMS_DIR/ambit/c35_3.3V/c35_IOLIB_3B_4M.sym

# Set the operating condition
#set_operating_condition -library <your library name> WORST-MIL -pvt max
#set_operating_condition -library <your library name> BEST-MIL -pvt min
set_operating_condition -library <your library name> TYPICAL -pvt typ
set_operating_parameter -temperature 45
set_wire_load_mode top
set_global target_technology {<your library name> c35_IOLIB_4M}
set_global fix_multiport_nets true
```

Table 3.34: Setup macro for PKS.

Save the macro as `setup.tcl` and make sure to change all `<full path to your tlf-file>` and `<your library name>` to its respective values. This macro is to be used with the `AMS_HIT-KIT 3.60` for the 0.35 um technology. If another technology is used changes have to be made accordingly.
Next, a macro that runs the actual synthesis should be created. Just as the case with the setup macro, this is not needed but it’s a good thing to do in order to avoid repeating the same tasks over and over again.

### Table 3.35: Run macro for synthesis.

<table>
<thead>
<tr>
<th>Synthesize.tcl</th>
</tr>
</thead>
</table>
| #1) Setup of technology libraries & general settings.  
source ./setup.tcl  
set_global echo_commands true  
set_global fanout_load_limit 10 |
| #2) Read the design (VHDL-code).  
read_vhdl <your vhdl>.vhd  
do_build_generic -module <your module> |
| #3) Set constraints (timing etc.).  
set_current_module <your module>  
set_top_timing_module <your module>  
set_clock IDEAL_CLOCK -period 10.0 -wave {0 3.3}  
set_clock_root -clock IDEAL_CLOCK -pos clk  
#set_clock_root -clock IDEAL_CLOCK -neg clkn --needed if clkn is used!!  
set_dont_modify -network [find -port clk]  
#set_clock_uncertainty 0.2 |
| #4) Optimize  
do_optimize |
| #5) Export data. Verilog netlist plus timing file.  
write_verilog <your verilog>.v  
write_gcf_assertions -version 1.4 <your gcf>.gcf |
| #6) Backannotate  
#read_sdf <your sdf-file>.sdf  
#do_optimize |

Save the macro in Table 3.35 as `synthesize.tcl` for example. Note that the first thing this script does is to include the `setup.tcl` created previously, so the setup script is run automatically as well. As in the setup macro, replace all occurrences within `< >` to the values wanted.
The line `read_vhdl <your vhdl>.vhdl` will import the high-level description of the design that is supposed to be synthesized. If the design is described in Verilog instead of VHDL, use `read_verilog <your verilog>.v` instead. The name of the module is the same as the name of the top entity in the VHDL code.

The run macro is a very generic, and can be used on many simple designs. If the design to be synthesized does not need a clock, then remove all lines that affect that. There is a multitude of different options and optimizations that can be done in PKS, but that is beyond the scope of this project. For example, one can optimize for speed or area but that requires that the cell library contains several different versions of the same cell as discussed earlier.

The script will output a verilog netlist and a constraints file (GCF-file) that will be used in the place and route tool.

### 3.5.4 RUNNING PKS

To start PKS, type

```
pks_shell -gui
```

in a terminal. The PKS window will show and at the bottom there is a prompt where commands can be entered. To setup PKS and run the synthesis, enter

```
source synthesize.tcl
```

which will execute the run macro that was created earlier. In the macro specified in Table 3.35, the setup script is automatically executed in the beginning so there is no need to execute it manually. Now the synthesis will start and depending on the complexity of the design this will take some time to finish.

When the execution has ended, the verilog netlist and the constraints file to be
used in place and route is automatically created. The verilog netlist can be imported into Cadence as a schematic, if desired, by choosing **Tools -> Import -> Verilog** in the Cadence icfb window. It is also possible to view the synthesized design within PKS by double clicking the module that was just created in the left pane of the PKS window and then clicking the **Schematic** tab in the right pane. Figure 3.36 shows a synthesized 4-bit counter and Table 3.37 contains the VHDL-code that was used.

**Figure 3.36**: Synthesized 4-bit counter.
Table 3.37: The VHDL-code for a 4-bit counter.

The synthesis step is now finished, but if backannotation from the place and route tool is wanted the PKS window should not be closed yet. More about this in section 3.7.
3.6 PLACE AND ROUTE IN SILICON ENSEMBLE

Now that all the necessary files have been created, it’s time to place and route the layout. The tool used for place and route is called SEULTRA and is a part of Silicon Ensemble. It will be easier to do this by using a macro file. A template for a suitable macro file can be found in Appendix A. This macro file describes, among other things, the area constraints for the layout, which libraries are to be used, and what VHDL or Verilog file it is supposed to work on.

Another file that is needed for place and route is a pin placement file, usually called ioplace.ioc. This file describes the location, name and direction of the pins. A template for this file can be found in Table 3.40.

3.6.1 PREPARATIONS

The most important preparation is to finish the synthesis step so that the verilog netlist and the constraints file have been generated correctly. The TLF-file and LEF-file created earlier must also be available. A Verilog file describing the cells must be created. Furthermore, some setting up of SEULTRA is required, which will be done in macros, just like in the synthesis step.
3.6.2 CREATING THE VERILOG DESCRIPTION

This file is much like the Verilog description used in the abstract generation process and that file can be used as a template. The only thing that differs is that in this file the functions and delays of the cells are described.

In Table 3.38 the description of an inverter and a 2-input nand gate is presented. Templates for most cells can, in the 0.35um technology, be found in \( \sim \text{ams_hit-3.60/verilog/c35b4/c35_CORELIB.v} \).

Create the file and save it as it will be used in the macros for running place and route. All cells in the library except filler and cap cells should be added to the file.
<table>
<thead>
<tr>
<th>Cell_descriptions.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>`celldefine</td>
</tr>
<tr>
<td>`timescale 1ns / 1ps</td>
</tr>
<tr>
<td>// Description : INVERTER</td>
</tr>
<tr>
<td>module inverter (a,a_bar);</td>
</tr>
<tr>
<td>output a_bar;</td>
</tr>
<tr>
<td>input a;</td>
</tr>
<tr>
<td>not (a_bar,a);</td>
</tr>
<tr>
<td>`ifdef functional</td>
</tr>
<tr>
<td>`else</td>
</tr>
<tr>
<td>specify</td>
</tr>
<tr>
<td>// Delays</td>
</tr>
<tr>
<td>( a -=&gt; a_bar) = (1,1);</td>
</tr>
<tr>
<td>`endif</td>
</tr>
<tr>
<td>endmodule</td>
</tr>
<tr>
<td>`endcelldefine</td>
</tr>
</tbody>
</table>

| `celldefine |
| `timescale 1ns / 1ps |
| // Description : 2 input NAND |
| module nand2 (a,b,out); |
| output out; |
| input a,b; |
| nand (out,b,a); |
| `ifdef functional |
| `else |
| specify |
| // Delays |
| ( a -=> out) = (1,1); |
| ( b -=> out) = (1,1); |
| `endif |
| endmodule |
| `endcelldefine |

**Table 3.38:** Verilog description of an inverter and a 2-input nand.
3.6.3 CREATING MACROS

All of the commands in the different macros that will be presented here could be placed in the same file but instead have been divided into different files so that it is easier to get an overview of them.

It is possible to run a script that is provided from the technology manufacturer that creates default scripts which then can be edited. To create these default scripts in the 0.35um technology, enter

```
ams_se -tech c35b4 -vn <verilog netlist>.v -vt <module name>
```

in a terminal. The `<verilog netlist>` is the netlist generated in the synthesis step and the `<module name>` is the same as used in the synthesis.

Open the file `c35b43.3V.gcf` that was created by the script to add the correct TLF-file. The standard cell library created in this project replaces the CORE-LIB library from the 0.35 um technology, thus the line importing the CORE-LIB TLF-file is replaced with the path to the new TLF-file. A template for this script is shown in Table 3.39.
A file that describes the pin placement is also needed. The file, usually called `ioplace.ioc`, is self explanatory and an example of the 4-bit counter is shown in Table 3.40.
### Table 3.40: The pin placement file.

```
Ioplace.ioc

# Copyright (c) 1997 by Cadence. All rights reserved.
# In each of TOP()/BOTTOM()/LEFT()/RIGHT() section, there are # placed IOs. In the IGNORE() section, the IOs are ignored # by the IOPlacer. In every section, the IO syntax could be: #   # for pin: (IOPIN iopinName );         #   # for pad: iopadName orientation ;       #   # for space: SPACE value;              # # The capital words are keywords. orientation is not required. # # The value is the space between the IO above and the IO below it.#

{IOPLACEHEADER (}
  (VERSION 5.3 )
  (DIVIDERCHAR "/" )
  (BUSBITCHARS "[]" )
{)}

{BOTTOM ( # IOs are ordered from left to right )}
{LEFT ( # IOs are ordered from bottom to top (IOPIN clk );
  (IOPIN enable );
  (IOPIN reset );
{)}

{RIGHT ( # IOs are ordered from bottom to top (IOPIN b[3] );
  (IOPIN b[2] );
  (IOPIN b[1] );
  (IOPIN b[0] );
  (IOPIN cout );
{)}

{TOP ( # IOs are ordered from left to right )}
{IGNORE ( # IOs are ignored(not placed) by IO Placer )
{)}
```
If filler cells have been added to the library, then open the `fillcore.mac` file and edit the names of the cells so that they match. Have the smallest cell at the bottom of the file and the largest at the top so that the place and route tool will first try to place the largest cells as illustrated in Table 3.41. Replace the names in bold with names corresponding to the filler cells in the library. Note that more cells than then the two in the example can be used.

![Table 3.41: The fillcore.mac file.](image)

It is now time to edit the main file that does the actual place and route and calls all these other files created. When created by the script it was called `gemma.mac` so open that file in a text editor. This script file is quite large and therefore it has been put in Appendix A.

### 3.6.4 RUNNING SEULTRA

To start SEULTRA enter:

```
seultra -m=60
```

in a terminal. The `-m=60` tells SEULTRA to use no more then 60 MB of memory. The SEULTRA window till open and to start the execution of the
place and route script, choose File -> Execute. In the dialog that opens, locate the gemma.mac file and click Execute. When it’s done executing, a gds2-file that contains the layout, a verilog netlist and an SDF-file have been created. The gds2-file can be imported as a layout into Cadence, the verilog netlist as a corresponding schematic and the SDF-file can be used for backannotation with PKS.

3.7 BACKANNOTATION

To check how well the design meets any specifications a backannotation to PKS from the place and route tool is a nice tool. To do this, go back to the PKS window and at the prompt enter:

```
read_sdf <path to sdf>.sdf
```

The SDF-file has been created by SEULTRA and can be found in the directory of which SEULTRA was run. It is now possible to check the performance of the design by clicking Reports in the menu of PKS. In the submenu there are a number of options such as Timing and Area. Choose Timing, for instance, and in the dialog that opens click Generate report.
In order to make everything previously described as useful as possible, some way of automating the entire, or parts of, the procedure is preferred. This chapter will discuss the possibility of making this procedure automatic and give suggestions on the behaviour of such a program. In other words, the following chapter will suggest properties for a program that automatically does most of the tasks earlier described. Such a program should be written in C++ or Java, and might have parts written in Skill, which is a Cadence programming language. Java would be preferred for its platform independence, and C++ for its speed compared to a Java program.

The main idea of the program is to have some sort of shell where the user could specify what is going to be built. The user could choose from a list of predefined standard cells, and then define the properties of those cells, like transistor width. A complementary solution is that the user could write his own netlist where he defines all the transistors and their interconnections. There exist a program made by two Master thesis students that can interpret that list and produce a layout from that list.

The basic idea is to let the user define the properties and then let the program generate the necessary files. Whether the program should be able to perform place and route as well is up to the programmer to decide.

This program would need to invoke several other programs in order to create all the files that are necessary for a standard cell library. These programs are Abstract generator, Aptivia and Cadence.

The following chapters will discuss the possibilities of making each step automatic, which then could be implemented in a program.
4.1 AUTOMATIC LAYOUT

A program already exists that can generate a layout from a given netlist (written in the syntax specified by that program). This program does not work with large designs, and that’s why the earlier mentioned procedures are necessary. It does make good basic cells though. However, there are flaws in the program, making it useless in its current version. It does not take grid and boundaries into consideration, which are required by Abstract generator. With further development, that program would be very useful.

4.2 AUTOMATIC VERILOG GENERATION

It is very easy to generate a Verilog file for a basic cell. The user could, for instance, choose from a list of standard cells and then let the program generate the code for it.

4.3 AUTOMATIC LEF AND ABSTRACT GENERATION

Since Abstract generator is a program that can use scripts, the possibility of making the procedure in Abstract generator automatic is fairly easy. Two problems do arise. First, the program generating the script might have difficulties in running the script itself in Abstract generator. User interference then becomes necessary. Secondly, there is a problem concerning the errors that might occur and how the program should handle this. It is difficult to write a program that can see the errors that Abstract generator might produce. The solution would be to assume that the layout is error free. This can to a high degree be made possible by setting harsh constraints on the layout and verilog generation so that it cannot produce other than correct layouts (according to what Abstract generator sees as a correct layout). By doing this, all fatal errors should be removed.

4.4 AUTOMATIC TLF GENERATION

In Aptivia, automatic TLF generation becomes a problem. There are several things that have to be carried out manually. A series of simulations have to be run in order for Aptivia to generate a TLF. This is not possible to do in an
automatic manner. The functionality of each cell also has to be defined, and this is also something that has to be done manually. Aptiva can on the other hand run a script from the command line, but it is not possible to let that script carry out the simulations.

4.5 AUTOMATIC PKS

The PKS procedure can, and should, be scripted. There are quite a lot of things to do, so if it has to be redone, a script really helps. There is no problem with running a script in PKS. A template for such a script can be found in Chapter 3.

Another thing that makes the PKS suitable for scripting is that it can be run without any GUI.

4.6 AUTOMATIC PLACE AND ROUTE IN SILICON ENSEMBLE

Silicon Ensemble can take command line arguments and is therefore very suitable for making the process automatic. The assumption that the construction is free from errors has to be made. A suitable size for the construction has to be calculated as well. This can be a problem since it can be difficult to calculate the size of the construction and if it has been made too small the place and route will fail. If the size on the other hand is too large, the construction will be larger than it has to be and a lot of space that could be used for other constructions is lost.

4.7 SUGGESTIONS ON A PROGRAM

This section will suggest what a program that would make the above procedure automatic would look like. It will assume that the already existing program that creates a layout from a user-defined netlist is upgraded so that it can make layouts that are correct for place and route.

A program that creates a full cell library should at first let the user define what cells the library should contain, and the properties of these cells. The user could either write an own netlist over the cell or perhaps choose from a
number of pre-defined standard cells.

When all the cells have been defined, and the layouts and schematics have been generated, the program should invoke Abstract generator, and also create a script for it. The script could be created from a pre-defined template. Assuming that the layouts have been created correctly, the abstract generator can now run the script and create an abstract and a LEF. The Abstract generator can be run with a script from the command line, which certainly makes things a lot easier since the program then itself can start Abstract generator and generate the LEF and abstract.

Now it is time to generate a TLF, and this is where the problems occur. There doesn’t seem to be a way to make aptivia generate a TLF automatically, so the decision is between making a TLF for the cells from a template TLF (which can be difficult, if not impossible) or running Aptivia manually. It is possible to run a script in Aptivia from the command line, but it will be very difficult to make aptivia import and understand a schematic. Another problem is that the defining of the cells seems to have to be done manually. In addition the cells have to be simulated in Aptivia, and there seems to be no way to do this automatically.

The PKS can be run with a script, so the program has to generate a PKS script suitable for the cells. It is possible to make the PKS run a script (or as it is called in PKS, a command file) at the command line.

After the PKS is done, Silicon ensemble has to be run in order to do place and route. This can be done by executing a script from the command line. The program that is suggested has to make some changes in the template of the script. There are several things that the program should let the user define, and then put in the script. These are things such as whether fillcaps and other fillings are to be used, the size of the construction, the pin placement (actually done in another file usually called ioplacement.ioc), and so on. It might even be possible for the program to calculate the smallest possible area to be used, or at least get an approximation of the area needed. The optimal program would be one that lets the user define everything that is necessary and thereafter make the rest automatically. This is unfortunately very difficult to achieve, since some programs (like Aptivia) seem to have to be run and configured manually in order to perform its task. It is certain that a program of this kind can make the procedure a lot less complicated and also speed up the process considerably, even though it might not be able to make the entire process automatic. A program like this would need a lot of effort put in it, especially
if high flexibility is demanded.
CONCLUSIONS

As the purpose first states, the possibility of making a standard cell library from which other constructions could be made, has been examined. A few basic cells, which can be found in Appendix B, has been created. They were first layouted manually, and thereafter run through Abstract generator, thus generating an abstract and a LEF-file. After that, each individual cell was run through Aptivia, generating a TLF for the cell library. Those were the steps carried out in order to make a standard cell library.

After obtaining a standard cell library a construction could be made. The design was written in VHDL and then mapped to the cell library through PKS and Silicon Ensemble. The result was in this case a 4-bit counter, which was layouted by Silicon Ensemble.

The possibility of making the entire process automatic has been examined as well. It seems that Aptivia is the program causing most problems in this aspect. There seems to be no way to make Aptivia generate a TLF-file automatically. However, all the other parts of the process can be performed automatically by using scripts. From that, the step to making a program that generates and runs the scripts isn’t very far. The structure and functionality of such a program has been suggested, but it is doubtful that it will be very useful since Aptivia has to be run manually anyway. On the other hand, a program that created only the layouts and schematics of cells would be useful. Then the user only needs to perform the tasks described earlier in order to create a complete cell library.

In summary, it is doubtful that there is a possibility of generating a complete cell library automatically. Manual entries in programs have to be made, and therefore, a master program that creates and runs all the necessary files will be impossible to create. It is still beneficial for the user to make his/her own cell libraries, since you get the desired cells and can be able to make a construction of those cells in a much shorter period of time compared to making
a full custom layout.
REFERENCES

Implementering av MSDCT i VHDL
APPENDIX A

SCRIPT FOR SEULTRA

```
Gemma.mac

##-- Silicon Ensemble Macro File Template
##-- set colors to make vias visible
set v DRAW.SWIRE.LAYERSET   "1 2 3 4 5 6" ;
set v DRAW.WIRE.LAYERSET     "1 2 3 4 5 6" ;
set v DRAW.SWIRE.4.COLOR 5 ;
set v DRAW.WIRE.4.COLOR 5 ;
set v DRAW.WIRE.5.COLOR 6 ;
set v DRAW.WIRE.5.COLOR 6 ;

set v DRAW.LAYER.ORDER "4 1 2 3 5 6" ;

SET VAR DRAW.PIN.NAME.AT "On"; #Added to see pin names

##-- Set Off Congestion Map Drawing
SET VAR DRAW.SCORE_GRAPHICS.AT OFF ;
```
##-- Set Design Directory
##--
SET VAR DB.DESIGN.DIR "./DB" ;
SET VAR VERIFY.TECHNOLOGY.MIN.FEATURESIZE 5 ;
SET VAR SROUTE.VIA.SNAPMANUFACTURINGGRID TRUE ;
SET VAR SROUTE.STRIPE_SNAP.RGRID "GRID" ;
SET VAR HYPEREXTRACT.RULES.FILE "/sw/cadence/libraries/ams_hit-3.60/artist/HK_C35/LEF/c35b4/c35b4_he.rules" ;

##-- Set Variables for VERILOG Import
##--
SET VAR INPUT.VERILOG.CREATE.IO.PINS FALSE ;
## SET VAR INPUT.VERILOG.ADD.LEADING.DELIM FALSE ;

set var INPUT.VERILOG.GROUND.NET "gnd! gnd3r! gnd3o!" ;
set var INPUT.VERILOG.POWER.NET "vdd! vdd3o! vdd3r1! vdd3r2!" ;
set var INPUT.VERILOG.SPECIAL.NETS "vdd! vdd3o! vdd3r1! vdd3r2! gnd! gnd3o! gnd3r!" ;
set var INPUT.VERILOG.LOGIC.0.NET gnd! ;
set var INPUT.VERILOG.LOGIC.1.NET vdd! ;

##-- Import Library Data
##-- LEF
FINPUT LEF F /sw/cadence/libraries/ams_hit-3.60/artist/HK_C35/LEF/c35b4/c35b4.lef ;
#commected by E&T
#INPUT LEF F /sw/cadence/libraries/ams_hit-3.60/artist/HK_C35/LEF/c35b4/CORELIB.lef ;
#added by E&T
INPUT LEF F <lef file>.lef ;

#INPUT LEF F /sw/cadence/libraries/ams_hit-3.60/artist/HK_C35/LEF/c35b4/35b4.lef ;
INPUT LEF F /sw/cadence/libraries/ams_hit-3.60/artist/HK_C35/LEF/c35b4/IOLIB_4M.lef ;
#INPUT LEF F /sw/cadence/libraries/ams_hit-3.60/artist/HK_C35/LEF/c35b4/IOLIBV5_4M.lef ;
#INPUT LEF F /sw/cadence/libraries/ams_hit-3.60/artist/HK_C35/LEF/c35b4/IOLIB_3B_4M.lef ;
## INPUT LEF F <addional LEF files> ;
Appendix A – SEULTRA Script

Gemma.mac cont’d

###-- CTLF Timing
###-- GCF File
INPUT CTLF INITFILE "/c35b43.3V.gcf" ;

###-- Import Design Data
###-- Verilog
#INPUT VERILOG FILE /sw/cadence/libraries/ams_hit-3.60/verilog/c35b4/ c35_CORELIB.v LIB DesignLib ;
#INPUT VERILOG FILE /sw/cadence/libraries/ams_hit-3.60/verilog/c35b4/ c35_CORELIB_3B.v LIB DesignLib ;
#added by E&T
INPUT VERILOG FILE <logical description file>.v LIB DesignLib ;
INPUT VERILOG FILE /sw/cadence/libraries/ams_hit-3.60/verilog/c35b4/ c35_IOLIB_4M.v LIB DesignLib ;
#INPUT VERILOG FILE /sw/cadence/libraries/ams_hit-3.60/verilog/c35b4/ c35_IOLIBV5_4M.v LIB DesignLib ;
#INPUT VERILOG FILE /sw/cadence/libraries/ams_hit-3.60/verilog/c35b4/ c35_IOLIB_3B_4M.v LIB DesignLib ;

INPUT VERILOG FILE <verilog netlist>.v LIB DesignLib
  REFLIB "DesignLib" DESIGN DesignLib.<module name>:hdl ;

###-- Import Timing Contraints
INPUT GCF FILENAME "<contraints file from synthesis>.gcf" REPORT- FILE "importgcf.rpt";

###-- Define Clock Nets
#change net clk use clock ;
CHANGE NET 'clk' USE CLOCK;

###-- To Set Rows On Grid
SET VAR PLAN.IOROW.SNAPGRID.X 100 ;
SET VAR PLAN.IOROW.SNAPGRID.Y 100 ;

###-- Save design
###--
SAVE loaded ;
##-- Initialize the floorplan
FINIT FLOORPLAN rowu 0.80 rowsp 0 blockhalo 20000
   f a 1 xio 21000 yio 10000 x 200000 y 140000;

WINDOW FIT ;

##-- Place the Periphery Cells
IOPLACE FILENAME ./ioplace.ioc STYLE EVEN;

##-- Cut Rows around Blocks
CUT ROW BLOCKHALO 30000;

##-- Power Routing
BUILD CHANNEL ;

##-- Add Power Rings
CONSTRUCT RING NET "vdd!" NET "gnd!"
   LAYER MET1 CORERINGWIDTH 2000 SPACING 2000 BLOCKRING-WIDTH 1000
   LAYER MET2 CORERINGWIDTH 2000 SPACING 2000 BLOCKRING-WIDTH 1000 ;

SAVE power_plan ;

##-- Add Cap cells
SROUTE ADDCELL MODEL <name of end cap left> PREFIX lcaps
   SPIN vdd! NET vdd! SPIN gnd! NET gnd!
   AREA (-46000000 -46000000 ) (46000000 46000000 ) PREENDCAP ;

SROUTE ADDCELL MODEL <name of endcap right> PREFIX rcaps
   SPIN vdd! NET vdd! SPIN gnd! NET gnd!
   AREA (-46000000 -46000000 ) (46000000 46000000 ) POSTENDCAP ;

##-- Place Standard Cells
SET VAR QPLACE.PLACE.PIN "";

QPLACE NOCONFIG ;

SAVE qplaced ;
##-- Add Filler Cells
##-- Has do be done before routing !!!
EXEC fillcore.mac ;

##-- Fill gaps between periphery cells
EXEC fillperi.mac ;

##-- Finish Power Routing
##-- Follow Pins
CONNECT RING NET "gnd!" NET "vdd!" FOLLOWPIN ;

##-- IO Rings
CONNECT RING
  NET 'vdd3r1!' NET 'vdd3r2!' NET 'vdd3o!'
  NET 'gnd3r!' NET 'gnd3o!' IORING ;

##-- Connect Stripes
CONNECT RING NET "vdd!" NET "gnd!" STRIPE ;

##-- Connect Blocks
CONNECT RING NET "vdd!" NET "gnd!" BLOCK ALLPORT ;

##-- Connect Power Pads
CONNECT RING NET "vdd!" NET "gnd!" IOPAD ALLPORT ;

##-- Route all the nets
SET VAR WROUTE.GROUTE.ONLY FALSE ;
SET VAR WROUTE.FINAL TRUE ;
SET VAR WROUTE.GLOBAL TRUE ;
SET VAR WROUTE.INCREMENTAL.FINAL FALSE ;
WROUTE NOCONFIG ;

##-- Write Logical SDF
SET VAR TIMING.REPORT.LOGICAL.SDF.OUTPUT TRUE;
REPORT DELAY SDFOUTPUT FILENAME <module name>.sdf ;

##-- Save the design
SAVE "final" ;
### Gemma.mac cont’d

<table>
<thead>
<tr>
<th>Script for place and route.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>##-- Save the design as DEF</strong></td>
</tr>
<tr>
<td>OUTPUT DEF FILENAME &quot;JDEF/&lt;module name&gt;.def&quot; ;</td>
</tr>
<tr>
<td><strong>##-- Write RSPF</strong></td>
</tr>
<tr>
<td>REPORT RC FILE &lt;module name&gt;.rspf ;</td>
</tr>
<tr>
<td><strong>##-- Write Verilog --</strong></td>
</tr>
<tr>
<td>OUTPUT VERILOG FILENAME &lt;module name&gt;_se.v;</td>
</tr>
<tr>
<td><strong>##-- Write GDSII File</strong></td>
</tr>
<tr>
<td>OUTPUT GDSII MAPFILE gds2.map STRUCTURENAME &lt;module name&gt; FILE &lt;module name&gt;_se.gds2 UNITS Thousands ;</td>
</tr>
</tbody>
</table>

**Table A.1**: Run script for place and route.
APPENDIX B

OUR CELL LIBRARY

This Appendix describes the standard cell library that has been created. It presents the programs we’ve used, including their versions, and it will present which cells have been constructed in our cell library.

B.1 TECHNICAL INFORMATION

Below follows a listing of programs that have been used in order to create our cell library.

<table>
<thead>
<tr>
<th>Program</th>
<th>Version</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cadence</td>
<td>5.0.33</td>
<td>cadence/5.0.33</td>
</tr>
<tr>
<td>Ams hit-kit</td>
<td>3.60B</td>
<td>ams/3.60</td>
</tr>
<tr>
<td>Abstract generator</td>
<td>5.0.33.14</td>
<td>cadence/5.0.33</td>
</tr>
<tr>
<td>Aptivia</td>
<td>03.2.4</td>
<td>cadence/aptivia</td>
</tr>
<tr>
<td>PKS</td>
<td>05.13</td>
<td>cadence/se</td>
</tr>
<tr>
<td>Silicon ensemble</td>
<td>5.4</td>
<td>cadence/se</td>
</tr>
</tbody>
</table>

Table B.1: Programs used.

The technology that has been used is the 0.35 um technology.

B.2 CELLS CREATED

This section presents a listing of the cells that were created in our standard
cell library. It will also present the layouts and schematics of these cells.

<table>
<thead>
<tr>
<th>Cell type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAND</td>
</tr>
<tr>
<td>INVERTER</td>
</tr>
<tr>
<td>D-FLIPFLOP</td>
</tr>
<tr>
<td>BUFFER</td>
</tr>
<tr>
<td>FILL</td>
</tr>
<tr>
<td>CAP</td>
</tr>
</tbody>
</table>

Table B.2: Cells included in our cell library.

**B.2.1 SCHEMATICS**

Figure B.3: Schematic of a nand gate.
Figure B.4: Schematic of an inverter gate.

Figure B.5: Schematic of a d-flip-flop.

For simplicity, the nand and inverter gates that make up the flip-flop have been replaced by symbols in Figure B.5.
The filler cell in Figure B.7 represents filler cells that are larger than 1*yPitch as discussed in section 2.1.4. The filler cell of minimum size
(1\*yPitch) does not need a schematic at all. Just like the minimum sized filler cell, the capacitance cells do not require a schematic as they only consist of a ground and power bar and VIAs.

**B.2.2 LAYOUTS**

![Layout of a nand gate](image)

**Figure B.8: Layout of a nand gate**
Figure B.9: Layout of an inverter

Figure B.10: Layout of a D-flipflop
Figure B.11: Layout of a buffer

Figure B.12: Layout of a filler cell
Figure B.13: Layout of the filler cell from Figure B.7
B.3 CONSTRUCTION

With the help of the above cells we made a synchronous 4-bit counter. This was made automatically with PKS and Silicon Ensemble and the counter worked properly, as the simulation in the next chapter shows.

The construction was made by mapping Silicon ensemble to our cell library, which made it use our cells in order to build the construction. The size of the construction had to be set arbitrarily and then resized if the resulting construction got too small or too large.

The procedure to create the cell library and the construction is the same as described in this report.

Below is a schematic view of the 4-bit counter.

The construction is quite large since the number of cells in the cell library are quite small. If there would be more cells in the library, the size of the construction would be smaller. This library only contains 2 logical cells from which all other logical cells have to be assembled, and that might increase the size of the construction.
B.4 SIMULATION RESULTS

Below follows the simulation results of the 4-bit counter.

![Simulation of 4-bit counter](image)

**Figure B.15:** Simulation of 4-bit counter.

B.5 FILES

This chapter will present the files that we have created for our cell library. We will show our TLF, LEF, SYM, Verilog and Schematics here.
B.5.1 TLF

The TLF-file will not be presented in its entire form, since that would require too many pages. Instead, the header part and the first cell of the TLF will be presented.
```plaintext
TLF

HEADER(
  LIBRARY("manualLibrary")
  DATE("Tue Jan 18 2005 at 10:48:29")
  VENDOR("Cadence")
  Environment("Nominal")
  TECHNOLOGY("cmos")
  TLF_VERSION("4.4")
  GENERATED_BY("Syn2tlf5.1-s101")
)

// User properties section
DEFINE_ATTRIBUTE(scaling_factors (CELL) (STRING))
DEFINE_ATTRIBUTE(state_variable_map (PIN) (STRING))

// model section
TIMING_Model(manualLibrary_templateMod
  (Spline
    (INPUT_SLEW_AXIS 0.100000 0.120000 0.140000 0.160000 0.180000
      0.200000 0.220000 0.240000 0.260000 0.280000 0.300000)
    (LOAD_AXIS 0.050000 0.060000 0.070000 0.080000 0.090000 0.100000
      0.110000 0.120000 0.130000 0.140000 0.150000)
    data())
  )
)

// properties section
PROPERTIES(
  UNIT(
    AREA_UNIT(1squ)
    CAP_UNIT(1pF)
    CONDUCTANCE_UNIT(1mS)
    CURRENT_UNIT(1mA)
    INDUCTANCE_UNIT(1pH)
    RES_UNIT(1kohm)
    TIME_UNIT(1ns)
    TEMPERATURE_UNIT(1C)
    VOLT_UNIT(1V)
    POWER_UNIT(1nW)
  )
)
```
Appendix B – Our cell library

TLF cont’d

```
Voltage(3.300000)
Temperature(27.000000)
Proc_Mult(1.000000)
Volt_Mult(1.000000)
Temp_Mult(1.000000)
Input_Threshold_Pct(RISE(50.000000) FALL(50.000000))
Output_Threshold_Pct(RISE(50.000000) FALL(50.000000))
Slew_Lower_Threshold_Pct(RISE(20.000000) FALL(20.000000))
Slew_Upper_Threshold_Pct(RISE(80.000000) FALL(80.000000))
Slew_Measure_Lower_Threshold_Pct(RISE(20.000000)
FALL(20.000000))
Slew_Measure_Upper_Threshold_Pct(RISE(80.000000)
FALL(80.000000))
//PVT Conditions
PVT_CONDS(TYPICAL)
  VOLTAGE(3.300000)
  PROC_VAR(0.000000)
  TEMPERATURE(27.000000)
  TREE_TYPE(balanced_tree)
)
PVT_CONDS(BEST)
  VOLTAGE(3.630000)
  PROC_VAR(0.000000)
  TEMPERATURE(0.000000)
  TREE_TYPE(best_case_tree)
)
PVT_CONDS(WORST)
  VOLTAGE(2.970000)
  PROC_VAR(0.000000)
  TEMPERATURE(100.000000)
  TREE_TYPE(worst_case_tree)
)
DEFAULT_PVT_CONDS(TYPICAL)
// WireLoad Models
)
CELL(inverter
  scaling_factors("inverter_SCALING")
// model section
VOLTAGE_MULT_Model(k_volt_fall_transitionMod
  (Linear
    (~ ~ : 1.460865 : -0.13965 )
  )
)
```
<table>
<thead>
<tr>
<th>Model Name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOLTAGE_MULT_Model(k_volt_rise_transitionMod)</td>
<td>Linear</td>
<td>$(-\infty: +1.017176$; $-0.277932)$</td>
</tr>
<tr>
<td>VOLTAGE_MULT_Model(k_volt_cell_fallMod)</td>
<td>Linear</td>
<td>$(-\infty: +1.225337$; $-0.068284)$</td>
</tr>
<tr>
<td>VOLTAGE_MULT_Model(k_volt_cell_riseMod)</td>
<td>Linear</td>
<td>$(-\infty: +1.437501$; $-0.132576)$</td>
</tr>
<tr>
<td>VOLTAGE_MULT_Model(k_volt_pin_capMod)</td>
<td>Linear</td>
<td>$(-\infty: +0.999830$; $0.000051)$</td>
</tr>
<tr>
<td>TEMPERATURE_MULT_Model(k_temp_fall_transitionMod)</td>
<td>Linear</td>
<td>$(-\infty: +0.928840$; $0.002636)$</td>
</tr>
<tr>
<td>TEMPERATURE_MULT_Model(k_temp_rise_transitionMod)</td>
<td>Linear</td>
<td>$(-\infty: +0.931747$; $0.002528)$</td>
</tr>
<tr>
<td>TEMPERATURE_MULT_Model(k_temp_cell_fallMod)</td>
<td>Linear</td>
<td>$(-\infty: +0.971483$; $0.001056)$</td>
</tr>
<tr>
<td>TEMPERATURE_MULT_Model(k_temp_cell_riseMod)</td>
<td>Linear</td>
<td>$(-\infty: +0.976223$; $0.000881)$</td>
</tr>
<tr>
<td>TEMPERATURE_MULT_Model(k_temp_pin_capMod)</td>
<td>Linear</td>
<td>$(-\infty: +1.000008$; $-0.000000)$</td>
</tr>
</tbody>
</table>
### TIMING Model

*ioDelayRiseModel0 manualLibrary_templateMod*

(Spline data

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.253982</td>
<td>0.295203</td>
<td>0.340122</td>
<td>0.380997</td>
<td>0.421523</td>
<td>0.461660</td>
<td></td>
</tr>
<tr>
<td>0.505374</td>
<td>0.550202</td>
<td>0.594768</td>
<td>0.639107</td>
<td>0.683231</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.255955</td>
<td>0.300248</td>
<td>0.342889</td>
<td>0.382470</td>
<td>0.427553</td>
<td>0.471853</td>
<td></td>
</tr>
<tr>
<td>0.515958</td>
<td>0.559656</td>
<td>0.600300</td>
<td>0.641033</td>
<td>0.681563</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.259368</td>
<td>0.302149</td>
<td>0.346665</td>
<td>0.389971</td>
<td>0.429005</td>
<td>0.469439</td>
<td></td>
</tr>
<tr>
<td>0.513256</td>
<td>0.557228</td>
<td>0.602434</td>
<td>0.646798</td>
<td>0.691072</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.264478</td>
<td>0.306151</td>
<td>0.348751</td>
<td>0.393011</td>
<td>0.436642</td>
<td>0.479215</td>
<td></td>
</tr>
<tr>
<td>0.519553</td>
<td>0.559358</td>
<td>0.600282</td>
<td>0.643054</td>
<td>0.688296</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.267712</td>
<td>0.309335</td>
<td>0.349891</td>
<td>0.395664</td>
<td>0.438746</td>
<td>0.482781</td>
<td></td>
</tr>
<tr>
<td>0.526461</td>
<td>0.567950</td>
<td>0.608217</td>
<td>0.648121</td>
<td>0.687832</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.270715</td>
<td>0.314288</td>
<td>0.353677</td>
<td>0.397394</td>
<td>0.442057</td>
<td>0.484364</td>
<td></td>
</tr>
<tr>
<td>0.528460</td>
<td>0.572403</td>
<td>0.616009</td>
<td>0.657417</td>
<td>0.698400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.273120</td>
<td>0.317648</td>
<td>0.360699</td>
<td>0.401912</td>
<td>0.441311</td>
<td>0.484737</td>
<td></td>
</tr>
<tr>
<td>0.529442</td>
<td>0.573755</td>
<td>0.617676</td>
<td>0.661689</td>
<td>0.705777</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.276723</td>
<td>0.320608</td>
<td>0.363888</td>
<td>0.404004</td>
<td>0.445184</td>
<td>0.487556</td>
<td></td>
</tr>
<tr>
<td>0.528903</td>
<td>0.573732</td>
<td>0.618332</td>
<td>0.662967</td>
<td>0.707865</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.281133</td>
<td>0.323007</td>
<td>0.367417</td>
<td>0.409405</td>
<td>0.449611</td>
<td>0.494037</td>
<td></td>
</tr>
<tr>
<td>0.534103</td>
<td>0.573849</td>
<td>0.618009</td>
<td>0.663889</td>
<td>0.700971</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.285525</td>
<td>0.325229</td>
<td>0.367507</td>
<td>0.413931</td>
<td>0.453555</td>
<td>0.499839</td>
<td></td>
</tr>
<tr>
<td>0.540439</td>
<td>0.580084</td>
<td>0.620304</td>
<td>0.663498</td>
<td>0.709543</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.288025</td>
<td>0.330291</td>
<td>0.372621</td>
<td>0.417109</td>
<td>0.459148</td>
<td>0.498326</td>
<td></td>
</tr>
<tr>
<td>0.546186</td>
<td>0.586845</td>
<td>0.627017</td>
<td>0.666263</td>
<td>0.709035</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*ioDelayFallModel0 manualLibrary_templateMod*

(Spline data

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.205733</td>
<td>0.239378</td>
<td>0.274277</td>
<td>0.306262</td>
<td>0.340514</td>
<td>0.376191</td>
<td></td>
</tr>
<tr>
<td>0.406934</td>
<td>0.441251</td>
<td>0.474900</td>
<td>0.508234</td>
<td>0.541634</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.209074</td>
<td>0.241410</td>
<td>0.275938</td>
<td>0.309437</td>
<td>0.346055</td>
<td>0.377794</td>
<td></td>
</tr>
<tr>
<td>0.409630</td>
<td>0.443761</td>
<td>0.475926</td>
<td>0.510830</td>
<td>0.544882</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.212655</td>
<td>0.246129</td>
<td>0.278254</td>
<td>0.312081</td>
<td>0.346025</td>
<td>0.379383</td>
<td></td>
</tr>
<tr>
<td>0.416906</td>
<td>0.448657</td>
<td>0.482699</td>
<td>0.514436</td>
<td>0.546354</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.214508</td>
<td>0.249667</td>
<td>0.283075</td>
<td>0.315220</td>
<td>0.347920</td>
<td>0.382326</td>
<td></td>
</tr>
<tr>
<td>0.416052</td>
<td>0.449427</td>
<td>0.488019</td>
<td>0.521742</td>
<td>0.553525</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLF cont’d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.217557 0.251234 0.284712 0.320126 0.352337 0.384533 0.418123 0.452425 0.486060 0.519430 0.553122)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.221265 0.253502 0.288084 0.321163 0.357266 0.389602 0.421783 0.454071 0.488268 0.522490 0.556107)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.223965 0.256829 0.297366 0.323768 0.357622 0.394487 0.427021 0.459084 0.490918 0.524307 0.558828)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.226652 0.259366 0.293746 0.326091 0.360204 0.397010 0.427824 0.463776 0.495562 0.527394 0.560454)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.229881 0.263717 0.297559 0.330110 0.362454 0.396701 0.430577 0.468081 0.500212 0.531992 0.563824)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.232852 0.265247 0.298787 0.334032 0.366336 0.398677 0.433350 0.467010 0.504550 0.536601 0.568376)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0.235786 0.269571 0.302540 0.337568 0.370193 0.402227 0.435600 0.469913 0.503550 0.540960 0.572943)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TIMING_Model(SlopeRiseModel0 manualLibrary_templateMod(Spline data
| (0.593707 0.696843 0.810443 0.914227 1.015350 1.114880 1.223490 1.336980 1.449140 1.560420 1.671090) |
| (0.590485 0.701764 0.809302 0.908052 1.020990 1.132060 1.242630 1.353080 1.460150 1.563980 1.666630) |
| (0.587329 0.698239 0.810026 0.919283 1.017170 1.118470 1.224710 1.335120 1.449120 1.560290 1.671210) |
| (0.593275 0.697260 0.807041 0.917695 1.027460 1.135880 1.238950 1.339010 1.432330 1.539260 1.654080) |
| (0.593704 0.696599 0.800581 0.916532 1.024200 1.134700 1.244760 1.351960 1.454830 1.555510 1.655510) |
| (0.592910 0.701631 0.797866 0.912509 1.024620 1.130230 1.240890 1.351300 1.461300 1.569280 1.674640) |
| (0.590082 0.701926 0.810524 0.912798 1.009880 1.121930 1.234890 1.346180 1.456490 1.567040 1.678370) |
| (0.582703 0.701195 0.809505 0.908541 1.015523 1.118860 1.223400 1.337370 1.449740 1.561780 1.674690) |
| (0.586869 0.698704 0.810000 0.915296 1.012840 1.128790 1.229210 1.328460 1.439240 1.555730 1.644970) |
| (0.592677 0.693438 0.802080 0.918374 1.016400 1.134730 1.237880 1.338020 1.438050 1.544720 1.661750) |
| (0.586762 0.691422 0.806430 0.917888 1.023650 1.119470 1.243040 1.346690 1.447910 1.545850 1.650350) |) ) ) )
<table>
<thead>
<tr>
<th>TLF cont’d</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMING_Model(SlopeFallModel0 manualLibrary_templateMod)</td>
</tr>
<tr>
<td>(Spline data )</td>
</tr>
<tr>
<td>(0.413961 0.492753 0.571237 0.648612 0.724713 0.799074 0.865818 0.948774 1.028900 1.107980 1.186600)</td>
</tr>
<tr>
<td>(0.416751 0.493652 0.567715 0.646420 0.723132 0.803002 0.881035 0.958395 1.021500 1.097060 1.178970)</td>
</tr>
<tr>
<td>(0.413956 0.493748 0.571052 0.641405 0.721714 0.800336 0.875974 0.956613 1.033400 1.112330 1.191080)</td>
</tr>
<tr>
<td>(0.415332 0.489514 0.570391 0.648245 0.725753 0.795659 0.875436 0.954095 1.027770 1.105350 1.186760)</td>
</tr>
<tr>
<td>(0.415791 0.490698 0.568920 0.646591 0.725228 0.802899 0.867009 0.949120 1.029000 1.107710 1.185230)</td>
</tr>
<tr>
<td>(0.415469 0.492002 0.568091 0.645541 0.722067 0.801883 0.879851 0.957751 1.020290 1.102530 1.181630)</td>
</tr>
<tr>
<td>(0.411464 0.489739 0.568991 0.642500 0.722047 0.796294 0.877871 0.956578 1.034710 1.093910 1.177730)</td>
</tr>
<tr>
<td>(0.415499 0.490815 0.569648 0.646142 0.718749 0.798384 0.876724 0.954537 1.013610 1.111800 1.168850)</td>
</tr>
<tr>
<td>(0.414848 0.487618 0.569115 0.646685 0.723329 0.795300 0.875357 0.948801 1.031730 1.110670 1.188900)</td>
</tr>
<tr>
<td>(0.420260 0.489310 0.567297 0.646390 0.723635 0.800221 0.873112 0.952473 1.026940 1.108950 1.187740)</td>
</tr>
<tr>
<td>(0.418561 0.492708 0.568448 0.643426 0.723549 0.800331 0.868904 0.950676 1.029550 1.104950 1.186180)</td>
</tr>
<tr>
<td>)</td>
</tr>
<tr>
<td>)</td>
</tr>
<tr>
<td>// properties section</td>
</tr>
<tr>
<td>Volt_Mult_Capacitance(k_volt_pin_capMod)</td>
</tr>
<tr>
<td>Temp_Mult_Capacitance(k_temp_pin_capMod)</td>
</tr>
<tr>
<td>Volt_Mult_Propagation(RISE(k_volt_cell_riseMod) FALL(k_volt_cell_fallMod))</td>
</tr>
<tr>
<td>Temp_Mult_Propagation(RISE(k_temp_cell_riseMod) FALL(k_temp_cell_fallMod))</td>
</tr>
<tr>
<td>Volt_Mult_Transition(RISE(k_volt_rise_transitionMod) FALL(k_volt_fall_transitionMod))</td>
</tr>
<tr>
<td>Temp_Mult_Transition(RISE(k_temp_rise_transitionMod) FALL(k_temp_fall_transitionMod))</td>
</tr>
<tr>
<td>PIN(a</td>
</tr>
<tr>
<td>PINTYPE: INPUT )</td>
</tr>
</tbody>
</table>
B.5.2 LEF

The technology LEF is pointless to present since it can be found in the Cadence library. Our generated LEF for our library will however be presented below. As with the TLF, the LEF-file is too large to be presented in its whole, so only a part of it, including the header and one cell, is presented. The rest has the same structure, but describes other cells.
<table>
<thead>
<tr>
<th>Appendix B  – Our cell library</th>
<th>LEF cont’d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VERSION 5.4 ;</td>
</tr>
<tr>
<td></td>
<td>NAMESCASESENSITIVE ON ;</td>
</tr>
<tr>
<td></td>
<td>DIVIDERCHAR &quot;/&quot; ;</td>
</tr>
<tr>
<td></td>
<td>BUSBITCHARS &quot;[]&quot; ;</td>
</tr>
<tr>
<td></td>
<td>UNITS DATABASE MICRONS 1000 ;</td>
</tr>
<tr>
<td></td>
<td>END UNITS</td>
</tr>
<tr>
<td></td>
<td>MANUFACTURINGGRID  0.005000 ;</td>
</tr>
<tr>
<td></td>
<td>ANTENNAINOUTDIFFAREA  100000.000 ;</td>
</tr>
<tr>
<td></td>
<td>ANTENNAOUTPUTDIFFAREA  100000.000 ;</td>
</tr>
<tr>
<td>SITE standard</td>
<td>SYMMETRY y ;</td>
</tr>
<tr>
<td></td>
<td>CLASS CORE ;</td>
</tr>
<tr>
<td></td>
<td>SIZE 1.400 BY 13.000 ;</td>
</tr>
<tr>
<td>END standard</td>
<td></td>
</tr>
<tr>
<td>SITE portCellSite</td>
<td>CLASS PAD ;</td>
</tr>
<tr>
<td></td>
<td>SIZE 0.600 BY 0.600 ;</td>
</tr>
<tr>
<td>END portCellSite</td>
<td></td>
</tr>
<tr>
<td>SITE blockSite</td>
<td>CLASS CORE ;</td>
</tr>
<tr>
<td></td>
<td>SIZE 1.000 BY 1.000 ;</td>
</tr>
<tr>
<td>END blockSite</td>
<td></td>
</tr>
<tr>
<td>SITE ioSite_P</td>
<td>SYMMETRY y ;</td>
</tr>
<tr>
<td></td>
<td>CLASS PAD ;</td>
</tr>
<tr>
<td></td>
<td>SIZE 0.100 BY 340.400 ;</td>
</tr>
<tr>
<td>END ioSite_P</td>
<td></td>
</tr>
</tbody>
</table>
SITE corner_P
SYMMETRY x y r90 ;
CLASS PAD ;
SIZE 340.400 BY 340.400 ;
END corner_P

SITE CoreSite
SYMMETRY Y ;
CLASS core ;
SIZE 1.400 BY 18.200 ;
END CoreSite

MACRO nand2
CLASS CORE ;
FOREIGN nand2 -160.8 -296.275 ;
ORIGIN 160.800 296.275 ;
SIZE 7.000 BY 18.200 ;
SYMMETRY X Y ;
SITE CoreSite ;
PIN a
   DIRECTION INPUT ;
   PORT
      LAYER MET1 ;
      RECT -159.100 -287.375 -158.400 -286.675 ;
      RECT -159.875 -287.250 -158.400 -286.750 ;
      END
   END a
PIN b
   DIRECTION INPUT ;
   PORT
      LAYER MET1 ;
      RECT -157.800 -290.250 -157.100 -289.550 ;
      RECT -159.425 -290.150 -157.100 -289.650 ;
      RECT -159.425 -291.325 -158.925 -288.475 ;
      END
   END b
### LEF cont’d

<table>
<thead>
<tr>
<th>PIN Name</th>
<th>Direction</th>
<th>Layer</th>
<th>Rectangles</th>
</tr>
</thead>
</table>
| **out**  | OUTPUT    | MET1  | -156.650 -290.175 -154.075 -289.675;  
|          |           |       | -158.000 -285.225 -156.150 -284.725;   
|          |           |       | -156.650 -293.725 -156.150 -284.725;   
|          |           |       | -156.850 -293.725 -156.150 -292.225;   
|          |           |       | -158.200 -283.400 -157.500 -281.900;   
| **out**  |           |       | END       |
| **gnd!** | INOUT     | MET1  | -160.800 -296.275 -153.800 -294.275;   
|          |           |       | -159.550 -293.725 -158.850 -292.225;   
|          |           |       | -159.475 -296.275 -158.975 -292.225;   |
| **gnd!** |           |       | END       |
| **vdd!** | INOUT     | MET1  | -160.800 -280.075 -153.800 -278.075;   
|          |           |       | -156.850 -283.400 -156.150 -281.900;   
|          |           |       | -156.750 -283.400 -156.250 -278.075;   
|          |           |       | -159.550 -283.400 -158.850 -281.900;   
|          |           |       | -159.425 -283.400 -158.925 -278.075;   |
| **vdd!** |           |       | END       |
| **nand2** | OBS      | MET1  | -158.200 -293.725 -157.500 -292.225;   |
|          |           |       | END       |

---
B.5.3 SYM

The SYM-file is a small file and the entire file is therefore presented.

```
SYM

# created by edifconv vers. 2.15
symlib 1.2 noname 12 16 30 -i
symref 424 nmos4
symref 491 pmos4
symref 558 ND_C
symref 594 P1_C
symref 630 PD_C
symref 666 VIA1_C
symref 704 inverter
symref 757 inverter20
symref 812 nand2
symref 873 buf2
symref 918 dff
symbol nmos4 * DEF \
  port G inout \ 
  port S inout \ 
  port D inout
symbol pmos4 * DEF \
  port G inout \ 
  port S inout \ 
  port D inout
symbol ND_C * DEF \
  port pdd inout
symbol P1_C * DEF \
  port pdd inout
symbol PD_C * DEF \
  port pdd inout
symbol VIA1_C * DEF \
  port pdd inout
symbol inverter * DEF \
  port a_bar out \ 
  port a in
```
B.5.4 LOGICAL

This is the logical description of the cells and their ports used in the abstract generator. It is not a detailed verilog description, but more like a declaration of modules.

<table>
<thead>
<tr>
<th>Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>module inverter (a, a_bar);</td>
</tr>
<tr>
<td>input a;</td>
</tr>
<tr>
<td>output a_bar;</td>
</tr>
<tr>
<td>endmodule</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>module inverter20 (a, a_bar);</td>
</tr>
<tr>
<td>input a;</td>
</tr>
<tr>
<td>output a_bar;</td>
</tr>
<tr>
<td>endmodule</td>
</tr>
</tbody>
</table>
## B.5.5 VERILOG

The more detailed Verilog code will be presented below.

```verilog
module inverter_correct (a, a_bar);
  input a;
  output a_bar;
endmodule

module nand2 (a, b, out)
  input a, b;
  output out;
endmodule

module buf2 (a, q)
  input a;
  output q;
endmodule

module dff (d, clk, q, q_bar)
  input d, clk;
  output q, q_bar;
endmodule
```

```verilog
// Description  : INVERTER
module inverter (a,a_bar);
output  a_bar;
input   a;
not (a_bar,a);
endmodule
```
Verilog cont’d

```verilog
'ifdef functional
  'else
  specify
  // Delays
  (a -&gt; a_bar) = (1,1);
  endspecify
'endif

endmodule
'endcelldefine

'celldefine
'timescale 1ns / 1ps

// Description : 2 input NAND
module nand2 (a,b,out);
  output  out;
  input   a,b;
  nand (out,b,a);

'ifdef functional
  'else
  specify
  // Delays
  (a -&gt; out) = (1,1);
  (b -&gt; out) = (1,1);
  endspecify
'endif

endmodule
'endcelldefine

celldefine
'timescale 1ns / 1ps
```
// Description : DFF with rising edge clock, q and q_bar outputs

module dff (d,clk,q,q_bar);

output q,q_bar;
input d,clk;

ifdef functional
U_FD_P_NO (buf_Q,d,clk,1'b1);
else
reg notifier;
U_FD_P_NO (buf_Q,d,clk,notifier);
endif

buf (q,buf_Q);
not (q_bar,buf_Q);

ifdef functional
else
specify
// Violation constraints
$setuphold (posedge clk,posedge d,1,1,notifier);
$setuphold (posedge clk,negedge d,1,1,notifier);
// Delays
(posedge clk  => (q  +: d)) = (1,1);
(posedge clk  => (q_bar -: d)) = (1,1);
$width(posedge clk, 1, 0, notifier);
$width(negedge clk, 1, 0, notifier);
endspecify
endif

endmodule
‘endcelldefine
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/

© Emil Jansson and Torgny Johansson