Reliability of Deep Submicron CMOS
Investigation of Number of Library Blocks Available to Synthesis Tools and the Resulting Performance

Second Semester Report
Spring Semester 2007

By
Karena Andersen
Andy Repasky

Prepared to partially fulfill the requirements for
EE402

Department of Electrical and Computer Engineering
Colorado State University
Fort Collins, Colorado 80523

Report Approved: ________________________________________
Project Advisor

______________________________
Senior Design Coordinator
Library design is expensive. So it is important to try to find the optimal number of blocks that are needed in a design library. If there is a way to cut down on library design, it would make the designing process easier. This will make it so that one could synthesis more blocks, and in doing this it could make the chip better and even lead to faster, more reliable chips that require less power. Cutting down on the library design also saves a lot of time and money since it will save a lot of unneeded man hours that were used to design the libraries.

There is a lot that can be done to improve library design. This paper will focus on how well the synthesis tools perform with a varying sized library. Multiple runs of the synthesis tools with different limitations on the cells it can choose will show its impact on the whole design.

It was found that the synthesis tool used was independent of library size. Through multiple runs it was found that the tool always chose the smallest block available. Therefore the performance data becomes dependent on the characteristics of the smallest block in the library and not how many blocks it has to choose from.
TABLE OF CONTENTS

Title i
Abstract ii
Table of Contents iii
List of Figures, Charts and Tables iii
I. Introduction 1
II. Background Information 2
III. Experiments 4
IV. Expected Results 6
V. Results 7
VI. Other Work 10
VII. Conclusions 12
References 13
Acknowledgments 14
Appendix A – Management and Budget 15
Appendix B - Definitions 16

LIST OF FIGURES

Figure 1 Synopsys Design Flow 2
Figure 2 Design Compiler Design Flow 3
Figure 3 Flowchart of performance 5
Figure 4 Performance vs. Latches (expected) 6
Figure 5 Flowchart of Pathmill script 11

LIST OF CHARTS

Chart 1 Performance for Random Algorithm 7
Chart 2 Performance of Largest First Algorithm 7
Chart 3 Performance of Smallest first Algorithm 8
Chart 4 Smallest Latch in the Library 9

LIST OF TABLES

Table 1 Drive Strength 8
Chapter I – Introduction

It is expensive to design libraries when creating computer chips. If there was a way to make library design easier and less expensive, it would save a lot of time. There are many different ways to do this. The way we started was to look into finding optimal number of blocks that were used in a cell. If we could find an optimal number, it could cut down on library design, and it could also make a faster and more dependable chip.

One way to optimize the design of the libraries was to look at the number of latches in a block. Latches are the most expensive part to design and they take up the most room. Thus finding an optimal number of latches is beneficial. To find the optimal number, a synthesis tool is used. The tool makes multiple runs, with each run picking from a limited list of latches. After each complete run, one latch is randomly taken away until there are none left. It is expected that with enough runs, an average is found and thus the optimal number of latches.

The purpose of this project is to help find a sufficient number of latches that the Physical Compiler can choose so that it can design blocks that meet its performance requirements as well as keep the library design and maintenance costs down. The flowing sections will discuss background information, how to obtain the desired data, expected results, preliminary results, and also explain so other work that has been done to help later on.
Chapter II – Background Information

Computer chip design requires a large infrastructure of tools, library cells, and knowledge in order to build a competitive product in a timely manner. One of these tools, manufactured by Synopsys, is the Physical Compiler. It allows designers to automatically generate a physical implementation from hardware description language (HDL) code. This tool is highly effective when combined with an efficient library and talented designers. Our project looks specifically at building a library with a sufficient number of latch sizes to optimize the performance of the blocks that are built.

The Physical Compiler goes through many steps when it generates a block. Below in the flow diagram labeled figure 1 is the description of those basic steps. The project only focuses on the first phase of generating a block. In figure 2, we see a more detailed view of this phase “Design Compiler/Physical Compiler.”

Figure 1: Synopsys design flow
During the Design Compiler phase, shown in figure 2, it searches the library for a given block that has functionality, drive strength, and delay that is optimal. For a large block, this process is potentially complicated because it has long paths and large loads to drive. It must still make timing requirements. Once the blocks have been chosen, they are assembled into a netlist. It is then passed on to the rest of the Physical Compiler flow to ultimately obtain layout.

To help Design Compiler meet its sometimes difficult task of timing requirements it would be optimal to have a very large library of cells for it to choose from. Unfortunately, this is not practical. The design and maintenance cost for a library is too high for a large number of cells due to the complexity of design as well as optimizing them for better performance. Due to the needed reliability and complexity of latches and flip-flops, their cost for design and test is even higher than that of a simple logic gate.
Chapter III – Experiments

This experiment was conducted on an actual library and with the Synopsys tools described in chapter two. The goal is to obtain a specific number latch drive strength sizes to design the next generation library, but for the purpose of this paper only the general trends will be discussed. The following flow chart explains the procedure to obtain the optimum number of latches for a library. For this experiment synthesized blocks and a library that have been already created will be used. The library contains only one type, functionality, of latch in many different drive strengths. Latches size is based off of their drive strength. The blocks will be re-run through the Physical Compiler flow with constraints set to exclude one more latch size from possible use in the block. Physical Compiler will then re-synthesize the block and generate timing, area, and power data for the block based on its choices of cells. This process will be repeated until Physical Compiler only has one latch left to choose. The data from all the runs is then compiled into a comma separated values file and is normalized to the initial value where there are no constraints on Physical Compiler.

Each time through the Physical Compiler flow a latch is randomly removed. Simply removing the largest or the smallest latch each time, it was though that the results would be very predictable and not helpful. For instance, if Physical Compiler is only able to choose the smallest latch it will compensate drive strength by adding additional buffers. These additional buffers will effectively take the place of a larger latch but generally will use much more area and power. With the random removal many sets of data can be used and averaged together. The data will then be more independent of latch size than if a latch was removed from the end of the list.

It will also be worthwhile to look at the results when the largest or smallest latch are sequentially removed each iteration. It will be simple to set up this experiment to check the validity of the random removal method.
Figure 3 Flowchart for performance

- Create output file directories
- Load the library cells that can be removed
- Remove library cell
- Run PC
- Copy output files to output directory
- All library cells but one removed?
  - Yes
    - Write data.csv file
  - No
    - Copy output files to output directory
Chapter IV – Expected Results

To get a qualitative measure for results three metrics will be used delay, area, and power. Delay is measured by a Pathmill type tool and is the maximum time for a signal to get from an input of the block to the output. Area is simply the sum of all the areas of the library blocks used. Similarly, power is the sum of the power dissipated in each library cell.

The results for delay, power, and area are all expected to be similar behaving. The fewer latch sizes that the Physical Compiler flow has to choose from the higher the values of performance. There should be a “sweet spot” in a plot of the data that show a point of diminishing returns, which will be our library size.

These predictions are based on the behavior of Physical Compiler. Cells can easily add drive strength with little impact to area or timing inside their box, but with synthesis design the only way to change drive strength is to add buffers. If a non-optimal cell is used, other library blocks are needed to step up or down the drive strength in order to properly drive the load. With the addition of these blocks, they add area, power, and can eventually impact delay to the overall design. There should be a point when synthesis has just enough options to adequately choose cells without needing many buffers, and this is where we are looking to find.

![Figure 4: Performance vs. latch number](image-url)
Chapter V – Results

The first results came from the random algorithm. It is shown in Chart 1 that we get fairly constant area, decreasing power, and decreasing delay when there are more drive strengths to be chosen\(^1\).

Chart 1 Performance of Random Algorithm

Next, Chart 2 illustrates the performance characteristics of a block when the largest drive strength latch is removed each time. This data is not quite what was expected. There should be more variation between latch sizes. This data led us to investigate the possibility that PC chooses the smallest latch every time.

Chart 2 Performance of Largest First Algorithm

\(^1\) Performance numbers are normalized to a value of 1. Values greater than 1 indicate worse performance while values less than 1 are better performance when compared to the nominal value.
Chart 3 shows the performance data when removing the smallest latch. This chart also supports the theory that PC chooses the smallest latch. Because of this the performances gets significantly better each time a smaller latch is available to PC.

![Chart 3 Performance of Smallest First Algorithm](chart3.png)

Through further experimentation we determined that PC does in fact choose the smallest latch. Because of this the performance data that we collected in the first, random test, is not dependent on the library size, rather the smallest drive strength left in the library. Illustrated on Table 1 is a distribution of different runs of PC and what it chose out of the library. This is explicit proof that it chooses the smallest latch.

<table>
<thead>
<tr>
<th>Drive Strength That Was Chosen</th>
<th>Smallest Drive Strength to Be Chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0.00% 0.00% 0.01% 100.00%</td>
<td>1 98.35% N/A N/A N/A</td>
</tr>
<tr>
<td>2 0.03% N/A N/A N/A</td>
<td>4 0.00% 89.73% N/A N/A</td>
</tr>
<tr>
<td>3 0.03% N/A N/A N/A</td>
<td>9 0.01% 8.42% N/A N/A</td>
</tr>
<tr>
<td>4 0.00% 89.73% N/A N/A</td>
<td>14 0.03% 0.00% N/A N/A</td>
</tr>
<tr>
<td>5 0.01% 8.42% N/A N/A</td>
<td>1 98.35% N/A N/A N/A</td>
</tr>
<tr>
<td>6 0.03% 0.02% N/A N/A</td>
<td>9 0.00% 0.02% 74.84% N/A</td>
</tr>
<tr>
<td>7 0.02% 0.00% N/A N/A</td>
<td>10 0.03% 0.02% 12.61% N/A</td>
</tr>
<tr>
<td>8 0.01% 0.00% N/A N/A</td>
<td>11 1.46% 1.79% 12.31% N/A</td>
</tr>
<tr>
<td>9 0.00% 0.02% 74.84% N/A</td>
<td>12 0.01% 0.00% 0.18% N/A</td>
</tr>
<tr>
<td>10 0.03% 0.02% 12.61% N/A</td>
<td>13 0.00% 0.00% 0.05% N/A</td>
</tr>
<tr>
<td>11 1.46% 1.79% 12.31% N/A</td>
<td>14 0.03% 0.00% 0.01% 100.00%</td>
</tr>
</tbody>
</table>

Table 1 Drive Strength
Our expected behavior described in chapter 4 of PC turned out to be false. It was believed that PC would choose a reasonable size latch rather than choosing a small latch and adding intermediate gates increasing the total path length. Illustrated in Chart 4, it is seen that choosing the smallest latch, \( x=0 \), is not the shortest path. Because of this we do not get the best performance. The addition of the extra gates into the path increases all performance metrics. The delay for this given path is also shown in Chart 4 and demonstrates that the best delay corresponds to the shortest path length.

![Chart 4 Smallest Latch in the Library](chart-4.png)
Chapter VI – Other Work

There was some work that was done this semester that wasn’t used yet in finding the collection of results. A script was developed to read and reorganize the output of a Pathmill output. This will help better understand and analyze the data. It is planned to be used later to examine the timing of the data that has been obtained in the experimental runs.

The script was created to find and display the delay from the beginning of a block of combinational logic to each node inside the block. Below, in figure 5, shows the flow of the script. It starts out reading in each line, looking for the beginning at the each path. It loads in the delay for each node until it reaches the end of the path. When the end of the path is found, the delay to each node from the start and the delays from each node to the end were stored. The script then outputs the results in a file that contains the path number, the node and the delay to the node from the beginning of the node. It creates another file that contains again the path number, the node and the delay from the node to the end of the block. After every path, it starts over until all the paths are calculated.

The hope is to find the longest path and the shortest path and find other statistics relating to each path in a block. With this information, we will better understand each cell and be able to look at different factors to make the chip faster.
Figure 5 Flowchart for Pathmill script

1. Start
2. Find beginning of path
3. Store length of time
4. End of Path
   - No
   - Yes (Find delay to and from each node and output the data)
5. End of File
   - No
   - Yes (Finish)
Chapter VI – Conclusions

Through the experiments, we have determined that PC has very little dependence on the number of drive strengths available in a library. Due to this behavior, the optimal solution would be to have a small latch that is used by PC. The majority of the time the drive strength required will be small enough that this latch will be sufficient to meet the performance requirements. In addition to the small latch, there are four or five larger latches that PC can choose. Despite the fact that PC will probably not place these larger latches when needed, a block designer can manually place them when needed.
References

Synopsys Products: RTL synthesis

Physical Compiler Data Sheet
Acknowledgments

During this semester we received a great amount of help from Dr. Tom Chen. He provided general direction and help with specific problems we encountered.
Appendix A – Management and Budget

The management and dividing up of the project was pretty easy. Andy did all of the work that involved the simulations. That included coming up with the different algorithms, starting the simulations and pulling off the results when they were done. Karena did the support for the simulations. That included writing scripts to process the data and to write the scripts that created library blocks that were used in the simulations.

Since we only ran simulations, we didn’t spend any money. All the programs that were used were available to us by the school or were a free download. Most of these tools are common in the workforce, especially those that focus on chip design and other similar projects. For the scripts that were created, we used Perl, which was a free download. All of the testing equipment and compilers were either free or available for our use.
Appendix B – Definitions

Cell – a small set of functionality. (ie. AND gate, OR gate, latch)

Block - a collection of cells to carry out more complex logic