

# **TIMING-DRIVEN CHIP DESIGN**



# Timing-driven chip design

Proefschrift

ter verkrijging van de graad van doctor  
aan de Technische Universiteit Delft,  
op gezag van de Rector Magnificus, Prof.dr.ir. J.T. Fokkema,  
voorzitter van het College voor Promoties,  
in het openbaar te verdedigen  
op maandag 26 april 2004 om 15.30 uur

door

Dignus Johannes JONGENEEL

elektrotechnisch ingenieur  
geboren te Dordrecht

Dit proefschrift is goedgekeurd door de promotor:  
Prof. dr. ir. R.H.J.M. Otten

Samenstelling promotiecommissie:

|                                |   |
|--------------------------------|---|
| Rector Magnificus,             | voorzitter                              |
| Prof. dr. ir. R.H.J.M. Otten,  | Technische Universiteit Delft, promotor |
| Prof. dr. R.K.Brayton,         | University of California at Berkeley    |
| Prof. dr. H. Corporaal,        | Technische Universiteit Eindhoven       |
| Prof. dr. ir. P.M. Dewilde,    | Technische Universiteit Delft           |
| Prof. dr. ir. P.R. Groeneveld, | Technische Universiteit Eindhoven       |
| Dr. ir. N.P. van der Meijs,    | Technische Universiteit Delft           |
| Prof. dr. S. Vassiliadis,      | Technische Universiteit Delft           |

Copyright 2004 ©by Dirk-Jan Jongeneel. All rights reserved. No parts of this book may be reproduced in any form or by any electronic or mechanical means (including photocopying, recording, or information storage or retrieval) without prior permission in writing from the author.

# contents

|          |   |            |
|----------|---|------------|
| <b>1</b> | <b>introduction</b>                                 | <b>1</b>   |
| 1.1      | ic design complexity: chasing moore's law . . . . . | 1          |
| 1.2      | the ic design process . . . . .                     | 4          |
| 1.3      | refinement . . . . .                                | 9          |
| 1.4      | the anatomy of an ic design flow . . . . .          | 11         |
| 1.5      | constant delay design methodology . . . . .         | 13         |
| 1.6      | iteration free design . . . . .                     | 15         |
| <b>2</b> | <b>wire planning</b>                                | <b>19</b>  |
| 2.1      | early timing analysis . . . . .                     | 20         |
| 2.2      | fixing delays . . . . .                             | 22         |
| 2.3      | sketches of a flow . . . . .                        | 25         |
| 2.4      | time budgeting . . . . .                            | 38         |
| 2.5      | hierarchical context . . . . .                      | 41         |
| 2.6      | algorithms . . . . .                                | 47         |
| <b>3</b> | <b>time budgeting</b>                               | <b>49</b>  |
| 3.1      | the problem in a wire planning context . . . . .    | 49         |
| 3.2      | mathematical problem formulation . . . . .          | 52         |
| 3.3      | problem size reduction . . . . .                    | 56         |
| 3.4      | further tableau reductions . . . . .                | 69         |
| 3.5      | enhancing robustness . . . . .                      | 74         |
| <b>4</b> | <b>constant delay mapping</b>                       | <b>77</b>  |
| 4.1      | technology mapping . . . . .                        | 80         |
| 4.2      | area control . . . . .                              | 88         |
| 4.3      | search space control . . . . .                      | 97         |
| 4.4      | experiments and conclusions . . . . .               | 103        |
| <b>5</b> | <b>conclusions</b>                                  | <b>109</b> |

|                         |            |
|-------------------------|------------|
| <b>bibliography</b>     | <b>112</b> |
| <b>summary</b>          | <b>117</b> |
| <b>samenvatting</b>     | <b>121</b> |
| <b>acknowledgements</b> | <b>127</b> |
| <b>biography</b>        | <b>129</b> |

# Chapter 1

## introduction

### 1.1 ic design complexity: chasing moore's law

Forty years ago technologists have produced the first silicon chip containing more than one transistor. At that time, only very few realized the profound significance of this 'birth' of the first Integrated Circuit in a small Fairchild laboratory in Palo Alto. Even fewer people could predict the profound influence it would have on the world decades later. One of those few who did see the significance of the birth of the digital semiconductors was Gordon Moore. In 1963 he predicted that integration density on silicon will quadruple every 3 years. This has proven to be one of the most reliable predictions in the history of industry.

Today, after 40 years of uninterrupted exponential growth in integration density, designs have over 100 million transistors. They pack a kilometer of interconnect wire that is only  $\frac{1}{100}$ th as thin as a human hair. Devices of such complexity are unprecedented in human history. It is also unique compared to other disciplines in engineering. A Boeing 747 airplane, for instance, only has a 5 million components. Moreover, its basic design did not change much in the 35 years since its inception. In contrast, a state of the art PC, is completely outdated after just 3 years. Technologically, microelectronics is driving a new industrial revolution that is affecting all aspects of life.

To be more precise, Moore's law states a doubling of the number of transistors on a chip every 18 month. The Semiconductor Industry Association (SIA) tracks this development and also extrapolates a forward outlook for this de-

velopment. It regularly presents a document known as the SIA roadmap. The key technical data of the latest (2003) update of the Semiconductor Industry Association (SIA) roadmap[36] is shown in table 1.1.

| Year  | 2003 | 2005 | 2007 | 2009  | 2012  | 2015  |
|---|------|------|------|-------|-------|-------|
| Channel length [nm]                           | 65   | 45   | 35   | 28    | 20    | 14    |
| uP Transistors [ $10^6$ ]                     | 153  | 243  | 386  | 614   | 773   | 2454  |
| Global Clock [MHz]                            | 2000 | 3125 | 4883 | 7629  | 14901 | 29103 |
| Local Clock [MHz]                             | 2976 | 5204 | 9285 | 12369 | 20065 | 33403 |
| Local wire length [ $\text{km}/\text{cm}^2$ ] | 0.57 | 0.97 | 1.11 | 1.55  | 2.21  | 3.54  |
| ASIC Package Pins                             | 2400 | 3400 | 4100 | 4600  | 4810  | 6402  |

Table 1.1: 2003 SIA Roadmap for key parameters

In consumer electronics (which represents one of the most competitive markets for semiconductors) a linear perceived increase in utility is obtained by investing an exponential increase in the transistor count. This is referred to as the “law of observed functionality” [8]. Therefore, Moore’s law is merely enabling a linear increase in product functionality. Without such functionality increase there is less incentive for consumers to replace existing equipment. As an example, consider the automotive industry. The transistor count in a car has indeed increased exponentially, mainly to control features such as fuel injection, abs, esp, airbags, satellite navigation, etc. To the consumer, these features appear as regular (linear) progress, providing incremental improvements. The design cost of such incremental features must be kept under control.

The times that IC’s are designed manually have long gone. Electronic Design Automation (EDA) tools are software programs that design and verify an integrated circuits. If the design cost is to be kept constant, the IC design productivity must increase with the same exponential rate as the processing technology. To design a circuit in the same amount of man-months, EDA software tool capacity must increase exponentially as well. The entire increase in productivity must come from the EDA tools and the methodology, since the capabilities of the human brain has been virtually constant.



In a nutshell, productivity increase of the design automation tools must keep up with the complexity increase of IC technology (that is driven by Moore's law). The design productivity of IC's is particularly affected by the following issues:

- *System complexity*: issues that rise from handling the sheer size of the SoC. Algorithms may not be able to handle design steps in reasonable run times. This necessitates a hierarchical design methodology and IP integration with reuse. System complexity scales exponentially with Moore's law.
- *Silicon complexity*: the issues related to silicon manufacturing technology such as device and interconnect parasitics, geometrical and electrical design rules, device reliability and process variability. In a way, silicon complexity is the result of the underlying physics that enable system complexity. In a design flow, silicon complexity increases the number of steps.

Both system and silicon complexity are increasing, leading to a superexponential increase in overall design complexity. Will it remain possible for a small group of humans to design a system of such huge complexity in reasonable amount of time? Based on the historic evidence over the past decades, the answer could be a cautious 'yes'. Despite continuous sceptism, Electronic Design Automation (EDA) tools have kept up with the exponentially increasing transistor count until now. But there is growing evidence that the 'design productivity gap' is widening.

To verify how the design productivity increase performed until now, lets consider the historical data in table 1.2. This shows the key data for the design of a popular family of graphics processors, as presented by Chris Malachovski in 2002 [27]. Graphics processors are representative for the asic design style where the logic is mapped onto a standard cell layout. The design time (measured as the number of months from design inception to tape-out) has remained fairly constant at approximately 12 to 18 months.

Looking at table 1.2, there is clear evidence that design productivity has increased significantly in 9 years. The latest chip required 5 times as many front-end designers and 9 times as many back-end designers than the 1993 chip. Meanwhile the transistor count went up by over 3 orders of magnitude

| Design start | Techn. node | Trans. count [M] | System complexity | Staff     |          |
|--------------|-------------|------------------|-------------------|-----------|----------|
|              |             |                  |                   | front-end | back-end |
| 1993         | 0.50 $\mu$  | 0.75             | 1x                | 1.0x      | 1.0x     |
| 1995         | 0.50 $\mu$  | 1.25             | 1.5x              | 1.2x      | 3.0x     |
| 1996         | 0.35 $\mu$  | 4.0              | 4x                | 1.6x      | 3.0x     |
| 1997         | 0.31 $\mu$  | 7.5              | 7x                | 1.7x      | 4.0x     |
| 1998         | 0.25 $\mu$  | 9.0              | 10x               | 1.5x      | 4.0x     |
| 1998         | 0.22 $\mu$  | 22.0             | 20x               | 2.5x      | 5.0x     |
| 1999         | 0.18 $\mu$  | 25.0             | 22x               | 1.5x      | 4.0x     |
| 1999         | 0.15 $\mu$  | 57.0             | 30x               | 3.5x      | 6.0x     |
| 2000         | 0.15 $\mu$  | 60.0             | 35x               | 1.5x      | 7.0x     |
| 2000         | 0.15 $\mu$  | 63.0             | 40x               | 3.0x      | 7.0x     |
| 2001         | 0.13 $\mu$  | 120.0            | 50x               | 5.0x      | 9.0x     |

Table 1.2: *Design Scale and complexity of a graphics processor chip family over the past 9 years (Source: Nvidia [27]).*

in the same period. Ideally, however, it would have been desirable to be able to design the latest chip with no additional engineers. A productivity gap is widening, but it widens slowly. Will this pace accelerate in the near future?

Another observation that can be derived from the data is that the productivity gap for back-end layout design is widening faster than at the front-end logical design. Apparently physical design is getting tougher than frontend design.

## 1.2 the ic design process

Before going into more detail about the design process steps and automation and optimization it is necessary to draw the big picture for chip design. At the top level, the design engineers start out with a general idea about the function of the chip. An additional set of requirements for performance, cost and design time is also given. In the process of refining these requirements, the designers gradually add additional detail. The design progresses along various levels of abstraction, eventually ending up at the detailed transistor and mask level for

mass production. The process of refinement is represented in figure 1.1 as finding the right way down a triangle.

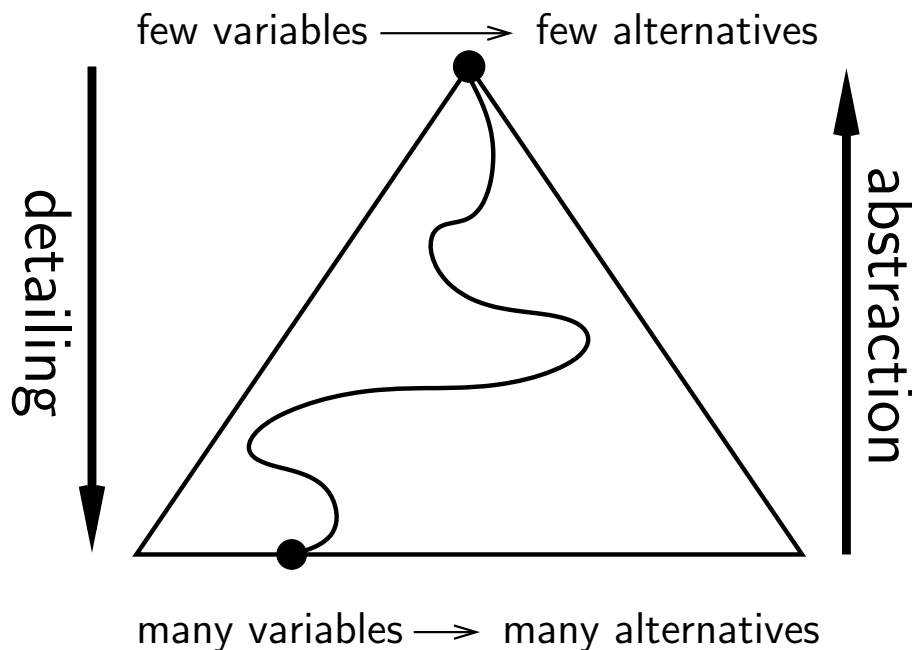


Figure 1.1: *Circuit design: going from abstract to high detail*

The top of the triangle represents the initial situation with only a few variables and objectives. The narrowness represents the small number of possible alternatives. The bottom of the triangle represents much more detail, such as information about sizes and locations of transistors, routes and sizes of wires, etc. But this level allows also for many different alternatives. The aim is to achieve an optimal design by starting from the top ending up at the right spot on the bottom of the triangle. The final design must fulfill the requirements started with at the top.

The transformation from a conceptual level to the detailed transistor level is not possible in one step. Instead, a long chain of smaller steps is performed, gradually stepping from a high level of abstraction to the lowest level. Each level of abstractions are used to obtain manageable tasks going from one level of abstraction to another.

We can distinguish multiple levels of abstraction in this design process, each with its own set of EDA tools (figure 1.2). And each has a different degree of automation. At the bottom level elementary logic gates are designed as layout mask patterns. Each cell contains just a hand full of transistors. A library

of gates is designed for each process technology. This 'gate-level' abstraction already hides a significant amount of process-specific detail.

The gate level network is an interconnected set of instantiations of specific gates in the library. Automatic routing algorithms generate the pattern that interconnects the gates. Other programs automatically place hundreds of thousands of gate instances on the chip surface. This process of placement and routing is called the *physical design* (or *layout synthesis*) of the IC.

*Technology mapping* is performed on a higher level of abstraction to convert an optimized logical network of boolean function nodes into a gate level network. At this point physical detail is introduced using the already abstracted information from the gate library of a process.

At the next abstraction level *logical synthesis* programs generate and optimize a net list of boolean function network nodes from functional RTL description in a hardware description language. Higher levels of design abstraction (behavioral and architectural synthesis) have also been automated. Not every transition of abstraction level is successfully automated. Especially the higher levels synthesis are difficult to automate, but efforts are there. Design space exploration systems are used to evaluate different architectures and languages, like SystemC, are developed to do high level modeling and simulation.

In this thesis we will focus on design automation methodologies at the boundary of the logical and physical domain.

The direction of the vertical arrows in figure 1.2 indicate the design synthesis and analysis paths along the levels of abstraction. Analysis tools verify the result of the synthesis tools. It can be seen as propagating information up the levels of hierarchy. In some cases it is straightforward bookkeeping, such as propagating size information up. In other cases it is detailed simulation to verify whether the circuit is within the required design limits. Analysis is in general straightforward easy, and the results are precise.

Synthesis is the process of moving down in an widening tree of possible implementations. It is in general more difficult and often computationally intensive. During synthesis decisions have to be made based on information that is available at the time. This information can be incorrect or imprecise, which adds a significant amount of uncertainty.

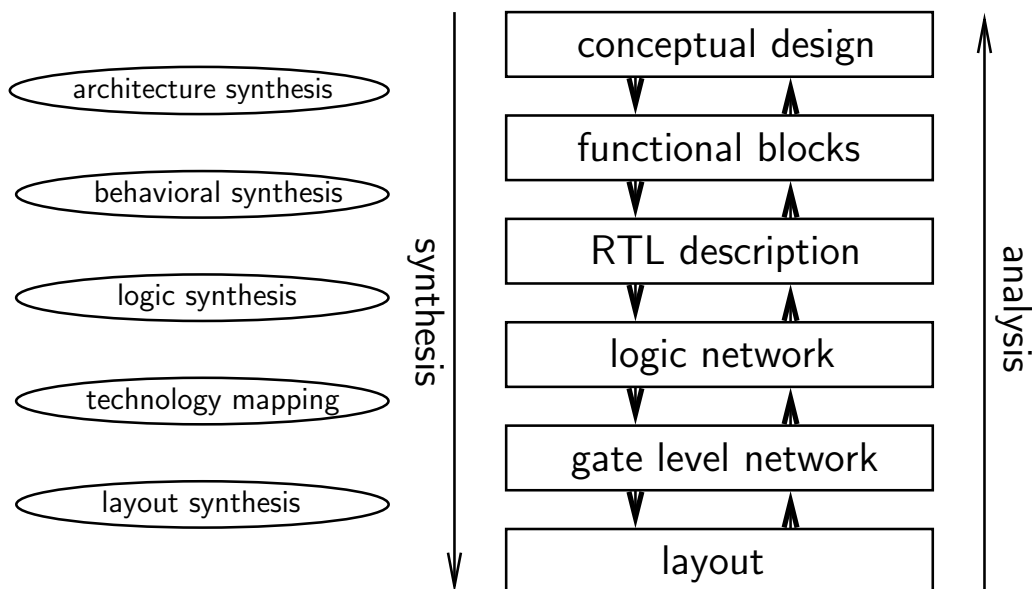


Figure 1.2: *Different levels of abstraction used in circuit design*

The main problem in the automation for the synthesis of higher levels of abstraction is to obtain valid abstract models to operate on. The higher the level of abstraction the harder it gets to model all aspects of the underlying levels in only a few parameters. Only very regular structures like memory or counters or pre-designed IP blocks are a little easier. But still only a small fraction of a complex system on a chip.

Automation of the lower levels of design abstraction has been more successful. This is in part because more accurate models are available at the lower abstraction levels and in part because layout design automation research has a longer tradition.

Accurate design metrics to populate the model are needed to guide the synthesis process along its way from a conceptual design level down to a physical implementation level. Typically, a synthesis tool explores a number of possible ways to go down the design tree. Each time it evaluates the design metrics that represent the quality of the configuration. Based on the outcome it will take a decision.

The final design must trade-off various conflicting design objectives. The most relevant objectives in the context of this thesis are delay (which equates to speed) and area (which equates to production cost). At every abstraction level there is some notion of their value. Typically the most promising solution

is chosen to continue the synthesis process. Since the decision is based on incomplete or imprecise information, it is not unlikely that an abstraction layer analysis step will detect errors or infeasible design constraints.

Algorithms are typically not well equipped to make a trade-offs between delay and area. The objects are as different as apples and peaches. Therefore only one objective is dealt with at a time. This results in a sub-optimal overall solution.

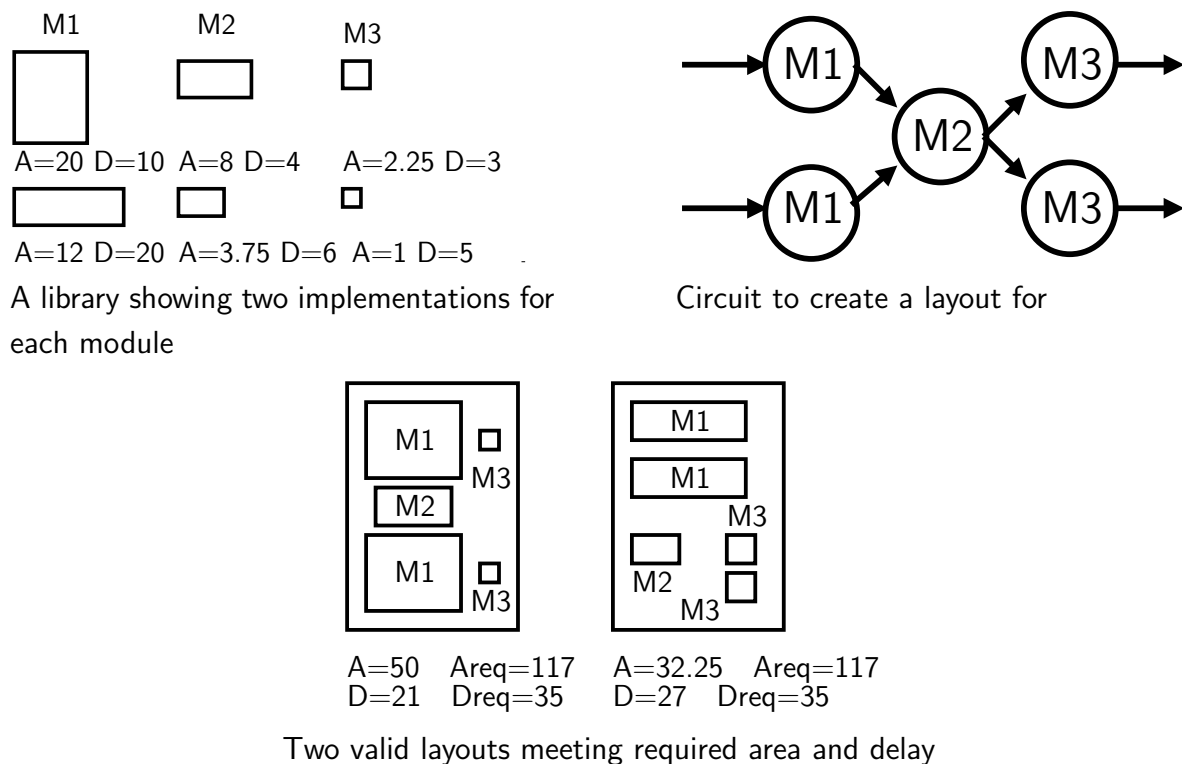


Figure 1.3: *The minimized solution is not always required*

To illustrate this, let's consider the example in figure 1.3. The figure on the top left shows a set of 6 modules that make up the functionality of the chip with a circuit as on the top right. For each such logical module, different layouts can be produced. They can differ in area, shape and speed. The bottom part shows two valid chip implementations that both meet the initial design requirements for area and delay.

The right implementation was produced by a tool that minimizes area. Since delay was not (much) an objective, the speed of the circuit is slower. The left implementation was produced by a tool that only minimizes delay, while disregarding area. As a result, its area is indeed much larger.

Since both circuits are meeting the design requirements, each has spent unnecessary effort in optimizing a single objective: one is too small, the other too fast. Suppose we tighten the design constraints to  $A_{req} < 40$  and  $D_{eq} = 25$ . In this case neither implementation would have been feasible. A better trade-off between area and power could have found such a feasible solution.

The main problem remains that a general optimization towards a single objective disregards the other objectives. In many cases it is much better to only optimize a few critical components, while leaving some 'slack' for all other components. A subsequent optimization towards another objective can pick up this slack, leading to a better trade-off between conflicting objectives. This is one of the main thrusts of this thesis that we will work out in more detail in the remainder of this thesis.

## 1.3 refinement

The process of detailing a design from concept to layout can be seen as a stepwise refinement process [41]. At each abstraction level a synthesis step is performed that refines the design solution. The objective of a refinement step is to fix a single parameter, while postponing decisions on all other design parameters to an as late as possible stage. The problem that is fixed at a step must remain unchanged during subsequent refinement steps. Once a decision is taken, the following steps have to take the result as a constraint. This is the blueprint of a non-iterative paradigm that we'll use throughout this thesis.

Poor modelling of the design metrics will not only result in sub optimal results, it might also result in over constrained sub problems. Modelling errors at higher levels are unavoidable, however. The goal is to conceive a design flow that can deal with the errors. In this way, the task of the subsequent steps is to adapt the reality to the model (rather than vice-versa).

As parameters are fixed and not changed afterwards their order of fixation is important. The predictions can still be imprecise when we really have to make a decision, but this is not as bad as neglecting them in the first place. We will see this to be a major problem of current design flows causing iterations trying to recover.

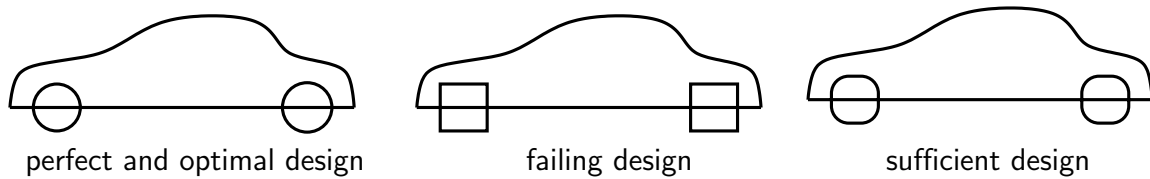


Figure 1.4: *When roundness of wheels is taken into account early a non failing sufficient design might exist which need not to be perfect*

The prediction can be off-target, but in a stepwise refinement paradigm its result should not be completely wrong. Looking for example at a car (figure 1.4), square wheels are very unlikely to be satisfactory and if some roundness was required from the beginning this would not have occurred although they still might not be perfectly round. But at least we do have a somewhat driveable car. If we neglected roundness of wheels in the design process we have to start over again and, with better knowledge now, assume wheels with some roundness. This leads to an unguaranteed number of iteration trying to converge to the somewhat more or less driveable car.

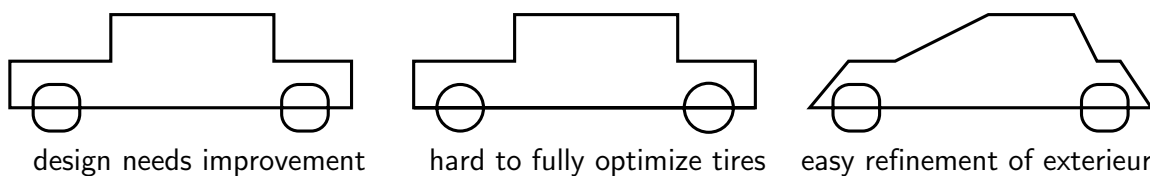


Figure 1.5: *Refining the parts which are the worst costs less than fully optimize an already good enough part*

In case the constraints are tighter than expected, stepwise refinement will still fail on certain aspects. Taking a car as an example again the wheels might not have been chosen to be perfectly round which would be ideal (figure 1.5). If moving the car at some speed at some energy cost is now not meeting our requirements the iteration approach could start to make the wheels rounder, or even perfect round, like an optimal designed and hand crafted car. This might give a solution but could cost a considerable amount of effort. In fact this aspect of the design can be overconstrained. Stepwise refinement would just go on with the best we can get and start refine other aspects. There is also room for improvement with a different more aerodynamic exterior design which would be able to make the car move fast enough with the non perfect wheels and would require much less effort to accomplish.



## 1.4 the anatomy of an ic design flow

Lets consider a generic design flow in more detail. Figure 1.6 shows the major steps in the IC design flow:

1. **Behavioral synthesis** results in an Register Transfer Level (RTL) description of the circuit. In most cases this is a manual process.
2. **Logic synthesis** tools produces a technology-independent net list of modules gates that implement the desired functionality. The optimizations in the stages are performed without taking into account the delay or area characteristics of the IC fabrication process. The main aim is boolean optimization.
3. **Technology mapping** is the process of converting the above description in a net list of gates in a specific technology. The tools attempts to fulfill delay constraints while minimizing the total area. The output is a interconnected net list in which the gates have a specific size, based on the statistical wire load model.
4. **Layout Synthesis** consists of 2 major steps:
  - *Placement* of the gates. The primary objective of the placer is to ensure that the gates are not overlapping and the total wire length is minimized. Issues such as minimizing the length of the most timing critical paths are only taken as secondary objectives. The large number of critical path is often over-constraining the algorithm.
  - *Routing* of the wires to interconnect the gates. This generates the topology and layer of the wires, and with that it sets the parasitic wire capacitances. Some wires will need to detour around obstacles, resulting in a higher than expected parasitic load.

The major problem that we are dealing with lies in step 3, where the optimization algorithms are unaware of the actual parasitic delays of the wires. This is because at that stage, the wires have not yet been laid down. The wire delays are only known as result of the last step. In most cases it is not possible to place the gates such that the timing of the entire circuit is feasible. To

alleviate the timing problem designers are running steps 3 and 4 a number of times in an iterative fashion. Each time the latest parasitic data is fed back into the optimization algorithm of step 3, and each time the circuit is placed again from scratch. This iterative process is not only slow, it is also not guaranteed to converge, especially in the latest processing technologies. If it does not converge, the designer is required to iterate back to steps 1 and 2.

The cause of the problem in the above iterative approach is that the synthesis tool in step 3 made a premature decision on the size of the logic gates. At a later stage, it was not possible to recover from the likely mistakes in this decision.

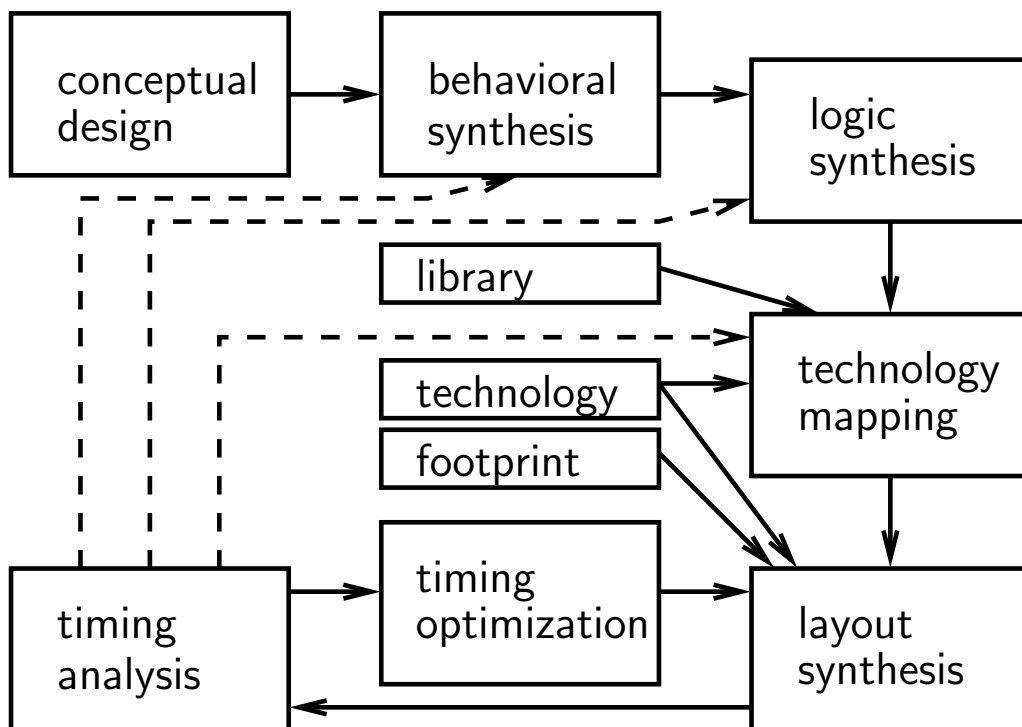


Figure 1.6: A circuit design methodology with possible iterations as dashed arcs

The tighter the constraints become and the more dependencies appear the harder it gets and more iterations are needed with no guaranteed convergence. This leads to an undesired increase and unpredictability of design time. This can not be solved by using more engineers too, especially if dependencies increase. Designing in parallel becomes useless if one part has large influence on an other part.

The key cause of the problems in the above flow is the parasitic capacitance of the metal interconnect. Below  $0.25\mu m$  interconnect starts to dominate the gate delay on a path delay [9]. Therefore the wire parasitics it can no longer be ignored as they were in the time when the flow was conceived.

The delay of a simple unbuffered wire grows quadratically with its length. Therefore it is the small fraction of long wires that cause the timing closure problems in IC design. At the higher levels of abstraction (steps 1, 2 and 3) it is not possible to predict which wires will be long.

It has been shown that the delay of a long wire (larger than 1 mm) can be made to increase linearly with the length using appropriate gate sizing and buffering[32][31]. We will use this effect since it simplifies the model considerably.

Without a good estimate for the parasitic wire delay, logic synthesis is focussing its timing optimization effort on the wrong paths. The actual parasitics of a wire between gates is only known after the circuit has been put through a number of physical design stages. In order to get to these stages a feasible gate-level net list must be available. On its turn, a proper gate net list depends on the expected parasitics. This chicken-and-egg loop was generally fixed by iteration or manual intervention.

## **1.5 constant delay design methodology**

The lengths of these long wires seem hard to predict and model in advance but have a huge impact on performance and design time. If they are disregarded as usually is done, they consume modules already available delay and therefore the total design including wires will not meet all requirements. Lots of tweaking and iterations are needed to try to get to a feasible design as the size of the modules impact their locations which in turn influences the wire lengths between them. It is even very unlikely to get to a solution unless most constraints are far from tight.

The initial solution for dealing with interconnect parasitics was the so-called "wire load model" that models wire parasitics to be dependent on design size and the number of fan-outs on a wire. Statistically, this model is correct, but the variations on individual wires can be off by hundreds of percents. Since

system timing is determined by the *worst case timing* path out of millions of paths, this variation is unacceptable. Ivan Sutherland [39] introduced the concept of logical effort that elegantly captures the first-order relationship between gate size, speed and delay. Larger gates have bigger transistors that can charge the capacitance of a wire quicker, making the path faster. During the placement process the wire length between a gate output and the inputs it drives can be estimated. This translates into an estimated wire capacitance that, on its turn, corresponds to a certain gate size. Adapting the gate size to the parasitic load can keep the path delay constant over a certain range. Meanwhile, downsizing gates that are less timing-critical saves area and reduces power consumption.

Gates with a larger drive strength will also impose a larger parasitic load on their input pins. The larger internal transistor will have a bigger active gate area. In conventional static cmos circuits this relationship is approximately linear: doubling the drive strength of a gate will double the parasitic load of its input pins. This means that this gate will have to be driven by a gate with a larger drive strength. Sizing down a gate will have the opposite effect: a smaller drive strength will be needed to drive its input at a given speed.

In Sutherland's model the gate delay is solely dependent on the ratio of the output load  $C_{load}$  over the input capacitance  $C_{in}$ . The ratio  $\frac{C_{load}}{C_{in}}$  is called the *gain*. Keeping the gain constant during a design flow will keep delay constant [30].

This 'gain based synthesis' technology has been implemented in certain modern layout synthesis systems, e.g. [3]. The guiding concept in gain based layout synthesis is to pick delays beforehand, and keep them fixed throughout the design steps. As the parasitic load  $C_{load}$  varies as a result of the actual placement and routing. Keeping the gain constant means varying the input capacitance by the same amount. The latter implies gate sizing.

Gain-based gate sizing is the first 'line of defence' in maintaining timing correctness. Other techniques, such as buffering, cloning, logic restructuring, and useful skew clock synthesis are also applied fulfill the timing constraints. The net list at the input of such system is quite different from the one that actually ends up in copper wires and silicon gates on the chip.

New delay modeling [20] [38] [16] [17] and usually together with new approaches at certain stages [31] [32] [23] [22] in the design process leads to

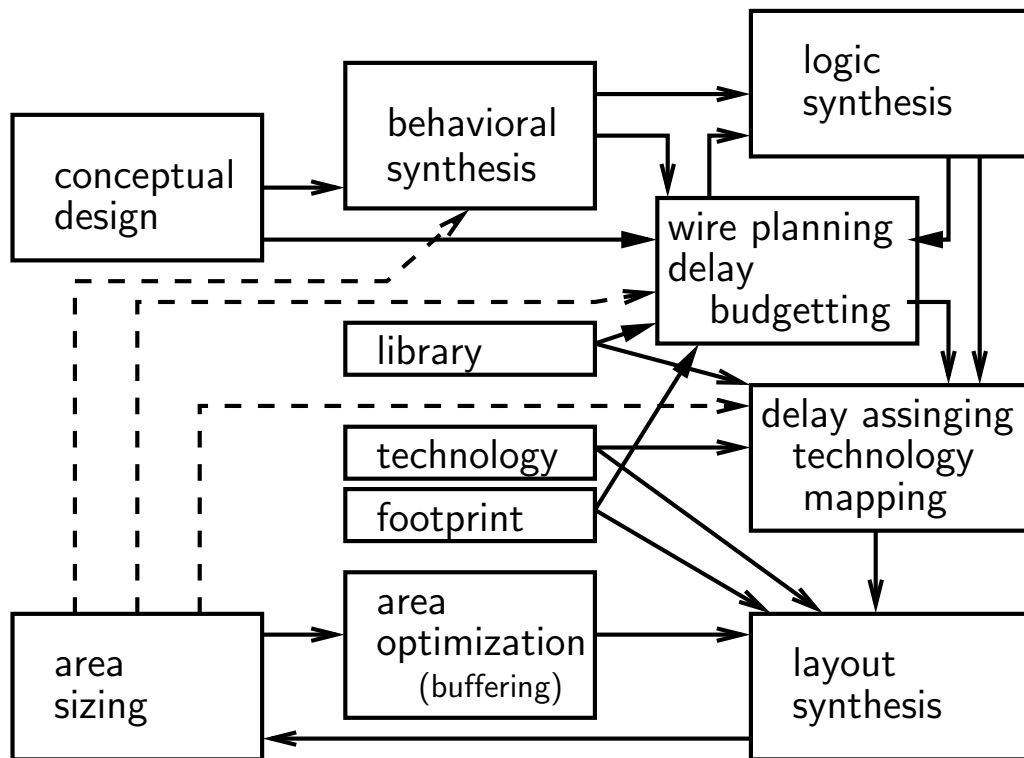


Figure 1.7: A circuit design methodology which keeps delays constant by area adjustments

a design flow keeping the delay constant early in the flow (figure 1.7). Still iteration is needed or desired to change area. Decisions are taken early on in the flow and might be off, and there might be a better structure or architecture with a lower area at the same delay. But at least the delay is given early and area is less restrictive often and merely results in savings if less area is used.

## 1.6 iteration free design

The ideal iteration free design style follows the stepwise refinement paradigm without any iterations. (figure 1.8). For a chip design flow this implies that the delay of long global wires should not be neglected at the beginning. It also requires a recovery mechanism for mismatches between predictions and reality. This process is improved by using better predictions and by deferring some decisions. The a-priori planning of wires can avoid 'surprises' later on in a flow.

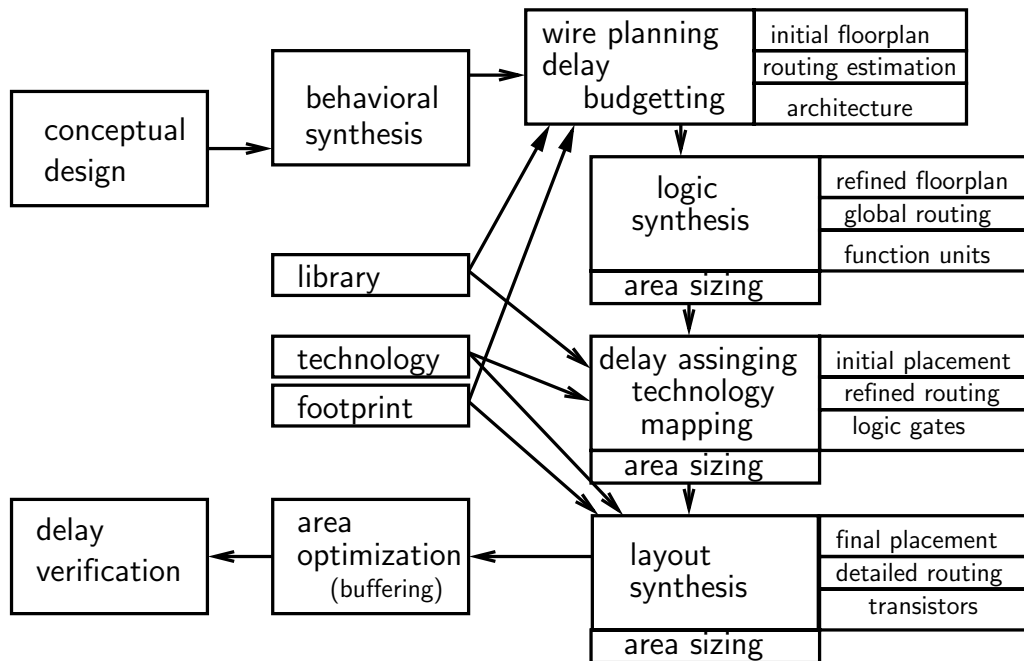


Figure 1.8: *The refinement approach does an early guess of wiring, function and placement and refines this each step more and does not falsification previous taken decisions by iteration*

At first sight, avoiding iteration at any price might seem too costly. Yet we have to keep in mind that optimization of certain design metrics are more for the purpose of automation and ordering of alternatives than a requirement. Of course saving some area might result in a little more profit, but the decision to do the design was based on the requirements set at the top level. As long as those are met, the avoidance of iteration and thus increase in design time is worth a lot too.

Although no iteration and altering of previous decisions is allowed, the sticking to estimated values is not completely true. In fact they are immediately neglected when the estimated values have been refined and the total result has to be presented to a higher level of the design (figure 1.9). Using some estimations some placement is calculated with some area estimations and from that the lower levels of the design have been designed with the given parameters. If now the resulting placement is required, not the previous estimated values and placement is returned, but a new calculated placement with the real values. If the estimation was close to the real values, the same answer would result, but if some differences occur, possibly some being better and some worse, a new placement can be calculated which will consume gracefully the errors and represent still the same footprint to the upper level.

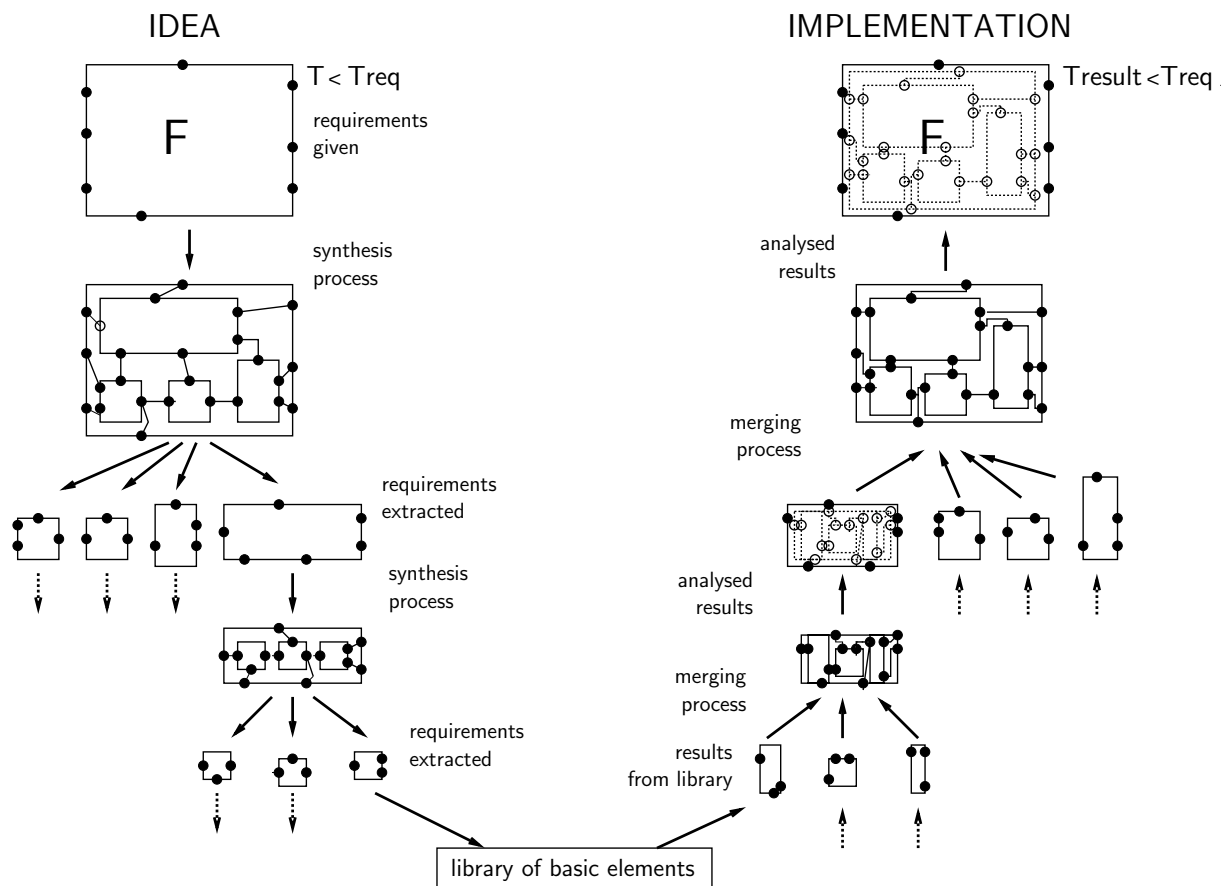


Figure 1.9: *The refinement from top to bottom sets new local constraints for lower levels. The final implementation is the result of successive substitution from the leaf*

Another big advantage of refinement is the possibility to divide a design in independent modules with specified design metric like area and delay which will not influence each other. Therefore they can be independently designed in parallel by several teams. This can avoid the design time increase or even reduce it while the systems become larger and larger. Also reusable components fit in this picture of absence of dependencies very nicely.

Avoiding iterations by refinement and sticking to its decisions and gracefully consume mismatches results in a more predictable design time and an early performance expectation. This expectation is important for exploring different alternatives for a particular design on high levels and thus for an early check point for a feasibility decision. The more predictable design time is good for economic reasons as time to market is better known, but also for the design effort itself as endless and possible non convergent design cycles are avoided.

A good example of this approach is found in floor planning and placement. A floor plan results from a point placement based on interconnection distance metrics. This results in relative locations for a placement based on that floorplan with now area rectangles in stead of points. At that moment the locations in the point placement or floorplan do not count any more but the refined places of the placement. If the floorplan is made using a slicing structure, it has been shown to be able to consume possible errors gracefully. A similar approach will be demonstrated for the global wires which will need to be account for for next generation designs. This procedure will be called wire planning. Another example will show that a different refinement strategy for the transition using technology mapping from a logic network to a gate level network can result in a better optimization. To put it in other words delaying some of the decisions can result in a better match between predictions and results.



## Chapter 2

# wire planning

The interplay between placement and routing is the classical chicken-and-egg problem of layout synthesis. It was obvious that the quality achievable in routing was largely determined by placement. Yet it was difficult to guide placement by those quality criteria. The almost invariably used objective was an estimated total wire length. The wire length in each net was estimated by half the perimeter of the rectangular hull of all pins to be connected by that net. Adding these estimates, whether or not weighted, yields the score of a candidate placement.

When the complexity of the circuits to be integrated reached a level at which automation was a necessity much research went into improving placement results with respect to the real objectives. Global routing served, beside lowering the complexity of the detailed routing task, also as a means to obtain early indications of routing quality. It could be applied on intermediate, rather topological than geometrical, placement data. Besides, it was an aid in avoiding iterations and supporting stepwise refinement.

It was acceptable as long as the main objective was to keep chips small, although routability soon became an issue. It was actually the latter that caused the term *wire planning* to enter the field. It was a way to preserve regularity in interconnection structures over the various layout stages down to the final masks. In the early nineties area and routability were no longer the sole objectives of placement. Performance became an issue as well, now that wire delay became an important component in the overall speed.

## 2.1 early timing analysis

The first reactions to the new situation were ever more detailed analyses of preliminary results, thus introducing global iterations in the design flow (section 1.4).

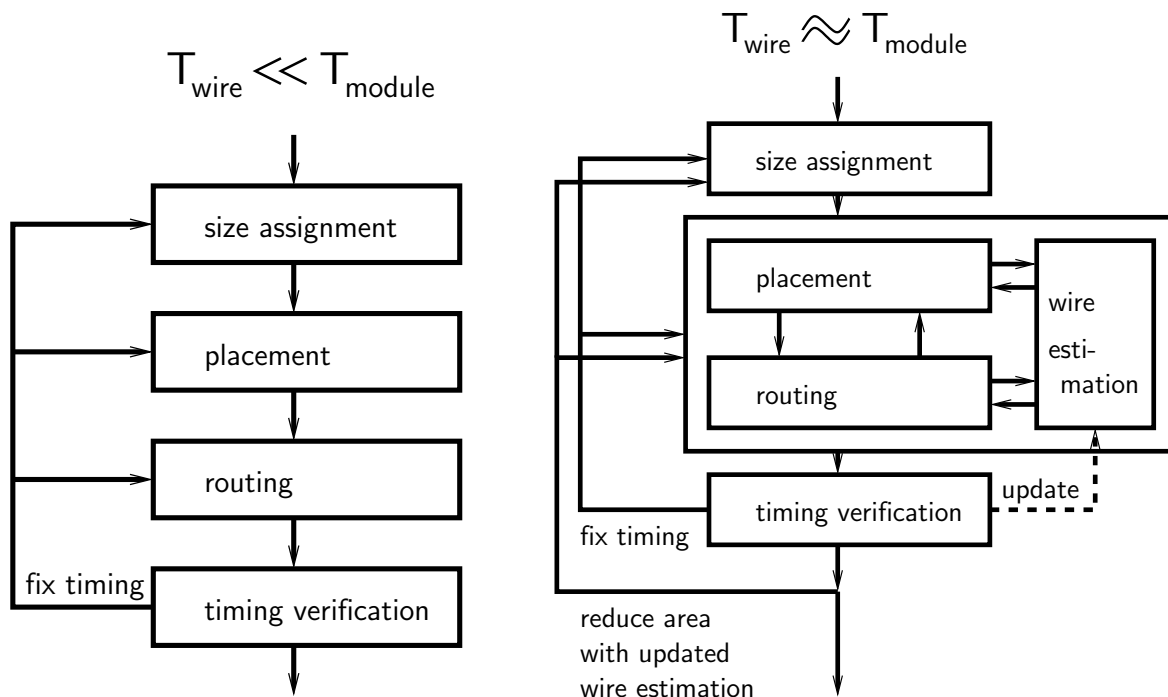


Figure 2.1: *Initially the effect of wires was neglected. As its impact became more profound wire estimation techniques were introduced while placement and routing were more tightly coupled.*

Again, common sense dictated that both a placement and a routing are needed before wire delay can be calculated. For wire geometry was indispensable for such analysis! Modifications in the placement and soon in the higher levels of the design were based on extraction results. But these modifications changed the basis of the extractions as objects had to be moved, sized or resynthesized. The new data might show, beside insufficient improvement on the uncovered weak spots, completely new detrimental aspects requiring adjustments. Even more than with routability and chip size, convergence became an issue and a problem.

Many of the decisions that make a design incur too long delays are taken very early in the trajectory. Iterations cycles therefore became longer and longer, going back to stages before logic synthesis and even architecture levels. Lack of adequate analysis tools closer to where these decisions are taken caused numerous runs through the whole cycle without any assurance that the feasible points in the design space were found. In the meantime integration technology developed further, enabling more complex and faster designs possible. At the same time cycles got longer and extraction more difficult, in the sense that more parasitic effects had to be taken into account. These development disabled global iterations with timing analysis at the bottom even more.

More refinement was therefore introduced in the basic trajectory mainly to avoid long wires where they could not be tolerated. Very basic estimates used in placements are based on the partitioning approaches [19] and are often guided by counting of connections between different modules. If there are lots of connections the modules have to be grouped to have them end up close together. This is assumed to make most wires short and thus their delay effect low. However, a consequence could be that some wires become extremely long between two very loosely connected modules separated by a cluster of densely connected modules.

A more advanced method is to try to minimize the total wire length of a chip based on better estimates. Instead of half the perimeter of the rectangular hull of all pins in the net, more sophisticated approximations of their steiner tree were tried. Other mathematical programming based solutions like force directed methods [11] [28] minimizing the total sum or of manhattan or quadratic pin-to-pin distances were further refined, and extended to use dynamic weighting schemes.

Although these developments postponed the break-down of the traditional methods for some time, the essential problem was not tackled by them. This problem was to identify critical nets and effectively “protect” them from getting too long. Repeated complete analysis to determine criticality is out of the question because too costly and pertinent to a previous situation.

Prospects of iterating through long cycles on the basis of analysis are not good for high performance chip design. Timing as an almost accidental result of optimization with other objectives such as a weighted sum of wire lengths, is not really acceptable for designs at the edge of what is possible in today's technologies. That industry is in need of methodologies that can guarantee the required performance (or report infeasibility). Only a complete shift in paradigm can enable such a methodology.

## 2.2 fixing delays

A shift from an analysis-based modifying approach to one where delays are postulated right from the beginning and every step in the design preserves whenever possible those postulates would be close to what is needed. The developments described in section 1.5 come to mind when such a shift is considered. It demonstrated the potential of keeping the delay of a gate constant by adapting its size to the load. The elaboration of that thought led to a system of equations which can be efficiently and robustly solved [32]. However, the load was purely capacitive in that theory. Resistance in wires was totally neglected. For longer wires this unrealistic. To complement that approach, a method to control wire delay, or rather to keep them fixed from an early stage in the design, is needed (figure 2.2).

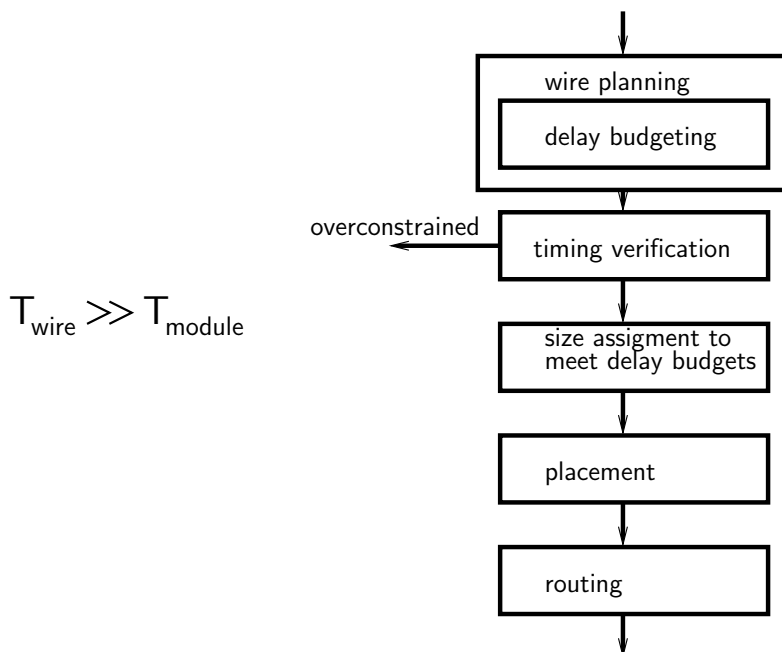


Figure 2.2: *The proposed wireplanning technique is capable to deal with wire delays much better especially when their influence becomes high compared to convectional approaches as used today (figure 2.1)*

Such a methodology fits into a philosophy that wants to avoid global iterations in the design (section 1.3). For once a delay gets fixed it is not supposed to change anymore. If the constant delay paradigm could be adapted to fit into this scheme by trading area for any desirable performance we may have the key ingredients of an approach that can be aptly called *wire planning*. It

would once more be an approach in which decisions are postponed until the maximum amount of information pertinent to that decision is available. This to avoid cutting off parts of the design space that later on might turn out to be badly needed.

Delay consists of wire delay and module delay, and the two are not always clearly separable. If set separately they have to be maintainable separately. Then we can allocate time budgets to each of the contributions, and make sure that they will be realized in the end. If the total budget is too tight this may be signaled at an early stage.

A problem however seems to be that something should be known about lengths of wires in order to assign a certain amount of delay to them. Also placement and routing, following the wire planning, should be able to proceed while respecting the now already fixed wire delays. This means that an a priori known relation between wire delay and wire lengths is needed.

The scenario so far foresees an early setting of the wire and module delays to fixed values from which they are not to deviate in the equal. The basis for wire delay allocation has to be some assumptions on the geometry, such as pin distances. What ever is left after subtracting that wire delay from the windows derived from a timing specification, is available for modules. The effect of loading is supposed to be absorbed in the later decisions. As suggested above, relying on the constant delay methodology to keep the delay of a module in which wire resistance is neglectable seems to be most promising. The price for this is in area, which might be less predictable since it is no longer a matter of taking patterns from a library based on function or specification. Rather the timing specification is generated by the procedure itself, and in such a way that it fits into the available budget.

Of course timing specification can be too tight. That means that not enough area can be made available between the pins to which the specification applies. From a different angle: increasing module size to gain in speeds forces pin-to-pin distances to increase leaving even less time budget to the modules. An early indication of this situation is important.

The trade-off between area and delay, alluded to in the above, is captured in a set of area-delay combinations, pairs of numbers that can be entered into a quadrant of positive coordinates (figure 2.3). If there are several equally fast implementations with different area usage only the one with minimum area at a certain delay is interesting. Those kind of points are *pareto* points. A general rule is that faster *pareto* realizations require larger area.

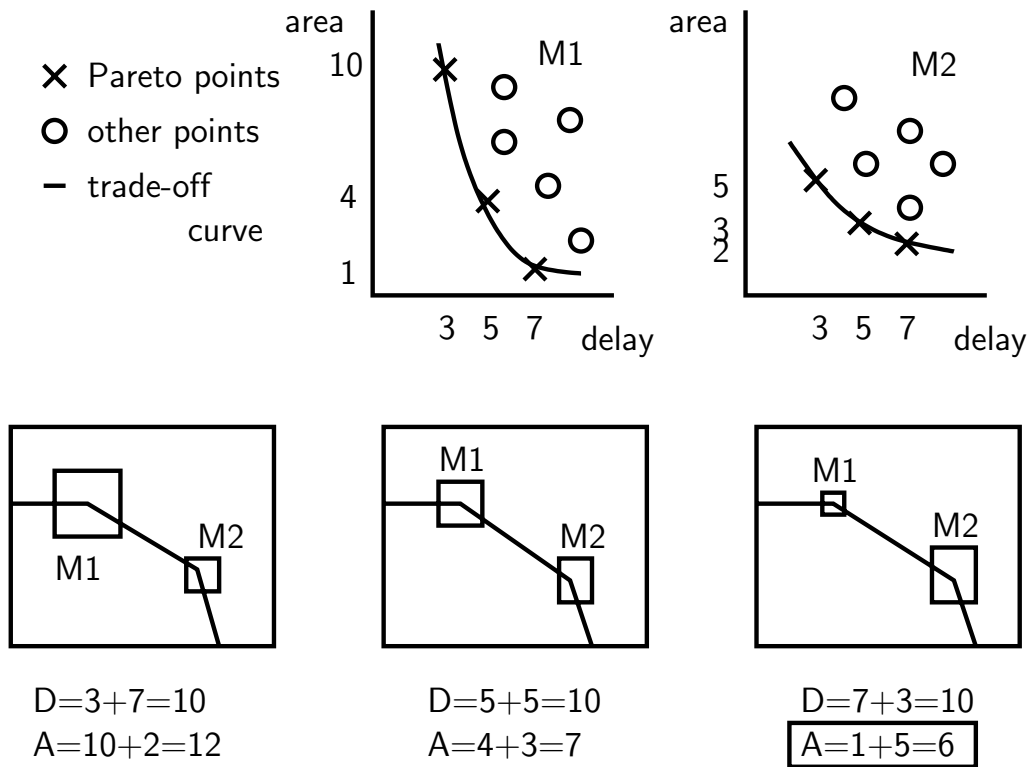


Figure 2.3: *Different modules will have different area-delay points. Only the Pareto points are of interest and span up an area-delay trade-off curve for the module. Combining two modules several areas are possible at equal delay.*

As these area-delay relations are not equal for all modules it can be attractive to assign one module more delay than another such that the total area becomes lower at an equal total delay (figure 2.3). As a result of this effect budgeting can result in a range of total area-delay solutions. Some will be unfeasible because they result in too high total delay. Others can be rejected as an equally fast solution exists at lower area usage. Also solutions which are faster than required and cost more area can be rejected.

So, in fact we are looking for a solution which just meets the timing without using more area than necessary. Keeping in mind the remark that constant delay modeling can change sizes to keep delays fixed it is not a bad idea to minimize the total amount of area used or at least keep it rather low. Minimization under the area constraints has the additional advantage of being conclusive about feasibility when the available space (the “footprint”) is given. It may also of course leave some space open, which is later on available for adjustments to fixed loading or to allocate wires.

Choosing time budgets for the modules does require the capability in synthesis to deliver an instantiation with the required delay. In general in a top down flow it cannot be assumed that all functionally equivalent implementations of a module are available. Working with a trade-off curve that interpolates between pareto points or is fitted through them, can produce such unavailable area-delay combinations. In general a point close to that result will be taken, causing some deviation (for whose resolution the extra open space may be welcome). We will look into more details of the aspects of constant delay and synthesis later in this thesis after showing how these delay budgets of modules can be calculated in a wire planning stage. Changes in the synthesis process are then presented to optimize it for delivering solutions with exactly required delay.

In conclusion, wire planning accounts for wire delays and sets module delays before place and route is even started, and is organized to assure that they will not need to be changed afterwards. It will need some notion about area-delay trade-offs of modules, not necessarily exact, since we rely on the self-fulfilling properties of stepwise refinement. In this way we want to (re-)establish an iteration-free flow and enable early abortion if the task is over-constrained.

## 2.3 sketches of a flow

Now considering the concept of wire planning as sketched in the previous section how can this be turned into an working algorithm in a certain context. First recall the fact that we started with the assumption that it would be a refinement strategy based on the fact that timing requirements are early given. Those can be derived from the system in which the design will be embedded or a targeted speed based on commercial competition considerations.

The required arrival times and arrival times can usually easily be obtained as the circuit will play a role in a total system and the timing at the inputs and outputs can be derived from that. Even in cases of the faster the better designs as in microprocessor competition there is a certain minimum to beat speed and a projected speed area point which allows the design to go through to expect profit. Therefore the delay requirements are assumed to be given in the wire planning context.

So the timing requirements limit the total circuit delay which is the sum of wire and module delays. The wire delay through the circuit depends on the length of the wire which is determined by the locations of the pins that it connects. These locations again are the result of the placement of the modules in a certain amount of space. The modules have also an area and the total area should fit into the proposed space too. As can be seen a number of aspects will have an impact on the context and the realization of wire planning and we will discuss one after another in this section.

### **floorplan**

Wire planning is proposed in the context that often a certain speed is required for a design. Considering this as an external given fact the other main limiting parameter is the total area to use. In general this should be as low as possible as silicon costs money and reduces profits. At the same time the total area usage is never to be allowed to be an arbitrary outcome of a completed design process. Imagine many months of work resulting in a design costing too much silicon to be a competitive product. Therefore before starting a design process an idea about a reasonable total allowable area usage will exist based on allowable costs. So we could also say that the total floorplan area and shape is also fixed and given already before really starting off.

This area and shape is not an impossible thing to come up with. After identifying globally what functionality is required then based on experience some estimations can be done about total area. At the same time there are often some discrete size steps to choose from as well as allowable shapes based on how a die can be divided efficiently. This exercise could even be seen as the result of doing already some steps into the refinement process of the design as far as floorplanning is concerned by determining the shape and size of it.

Another approach would be to start with a guessed oversized floorplan and perform an initial wire planning, place and route as a worst case guess. This will result in required module areas at a given delay performance. Note also that for smaller footprints delay will improve too as wires have to travel less distance and thus resulting into less delay wasted on them. Therefore more delay is available for the modules and the current areas can be seen as upper bounds resulting into an upper bound on total area required. Even multiple area-delay performance points could be created. Based on those and adding space for

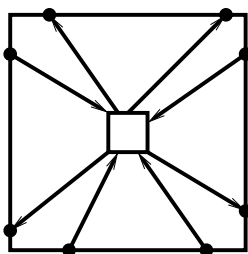


wires and possibly some error correction a particular area and floorplan shape can be chosen and set fixed for a desired delay choice. Now again a floorplan shape and area is obtained and a non-iterative wire planning step can follow.

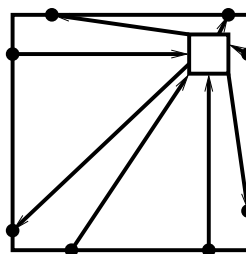
Those initial steps leading to a floorplan are very likely to be done as based on those parameters the feasibility can already be evaluated and if they are fulfilling the requirements and constraints the development process can continue. So a footprint is likely the first thing to be obtained. Therefore for the rest of our discussion the floorplan will also assumed to be given besides the timing requirements.

### pin placement and delay

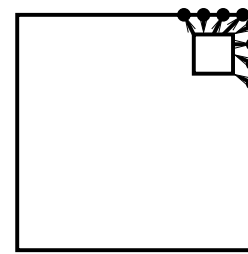
Having obtained a floorplan and timing requirements another important piece of information related to the external environment is missing. The delay requirements can be seen as input pin to output pin delays or equivalent arrival times at the inputs and required arrival times at the output pins. The delays from pin-to-pin depend on the route between them through modules and wires. As the delay in wires depends on the distance traversed on the chip also the distance between the pins is important. What is needed is a pin placement within the floorplan such that the routing and thus wire delays can be determined. Although the exact routing will determine the total length of wires also the pin positions will have a significant impact although more indirect.



worst case input output  
wire length: half perimeter



worst case input output  
wire length: 2 x half perimeter



worst case input output  
wire length:  $\ll$  half perimeter

Figure 2.4: *For the first pin placement the module should be placed in the center to make it a good pin assignment. But a much better pin assignment exists which reduces the delay wasted in wires with a suitable placement.*

Determining the optimal placement of the pins is problematic and can have a dramatic effect on minimum feasible timing (fig 2.4). If the inputs and the outputs of a certain piece of logic function are evenly distributed all around the edge of the floorplan the minimum wire length from an input to an output varies from half the perimeter of the floorplan in case the module is in the center to one times the perimeter of the floorplan in case the module is in a corner. At the same time an optimal pin assignment which places the inputs and outputs close together and close to the logic block which should be placed close to the edge can have almost zero delay in the wires. So there is clearly an optimality aspect in the pin assignment problem.

Note that the minimum length of a wire is at least equal to the manhattan distance between the pins. Therefore the pins should be placed as close together as possible. But most pins will have a relation with multiple other pins and as they cannot be placed all on the same spot and the paths have to go through modules with certain sizes they have to spread out. Often the different paths will result in conflicting desired pin grouping. Therefore the pin-to-pin distance should be made all about evenly short and with a minimum total length.

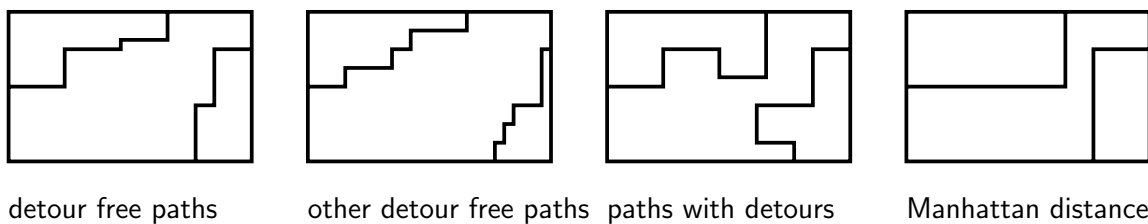


Figure 2.5: *The first two figures show alternatives wire tours without detours followed by one which has detours. All detour free paths have a total length equal to the manhattan distance which is also the minimum length possible.*

Not only the distances of the pins from each other, but also the route from one to the other through the modules determines the total wire length needed and thus delay (fig 2.5). In the ideal case a path makes no detour and the total wire length is minimum given the pin positions. The pin positions determine the minimum needed wire length which is equal to their manhattan distance between the pins. Any route which is longer than that can be regarded to be detouring. Achieving this situation should be the objective of all good placement algorithms as they should minimize the total wire length in general.

So the possible locations of the modules and thus the placement have an influence on the optimality of the wire lengths for a given pin assignment and the other way around. Placements can be adjusted to take the given pin positions into account to achieve this detour free case, but on the other hand avoiding the need for detours should also be already the focus point for the pin assignment process on forehand as its result can be regarded more optimal if it allows this placement to be such that detours can be avoided.

Besides the question of optimality there is also a question of existence. Note that the network structure plays a role here too as it could prevent the existence of such optimal pin and module placement at all. Given all possible pin assignments the network structure might be such that no placement is possible without detours. Duplication of modules and restructuring of functions and network might be needed to get to a feasible optimal pin and module placement at all.

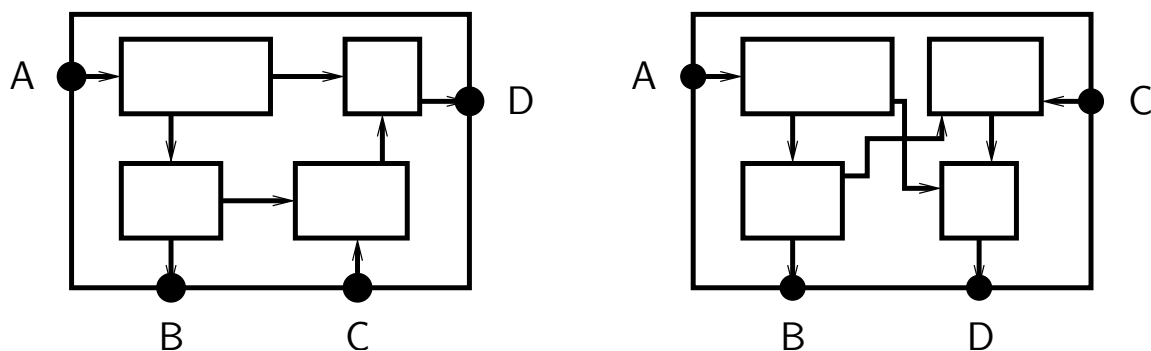


Figure 2.6: *Two results of a common place and route with different pin placements. Note that the pin placement of the second circuit results in longer minimum wire length for A to D and C to B.*

Optimal pin placement is obvious an unsolved problem where even current design methodologies would immediately profit from (fig 2.6). Currently pins are assigned positions more based on structure and embedding. The circuit to be designed will be integrated within a larger design and thus the pins will not be arbitrarily placed and ordered. More often there will be some address or data busses whose signals are often preferably routed next to each other and by that requiring the corresponding pins to be located together and in a structured fashion. Other critical signals will have quite predetermined pin locations. Power, ground and some left over signals do not likely pose nor require some special order or location and therefore can be fixed on pins of the circuit to be designed as seems appropriate in advance.

This pin assignment will however not be optimal at all and very likely will not allow the optimal placements with no detouring paths and minimum total wire length. Then there is still also the existence problem that the network is such that such a combination of pin assignment and placement is not possible at all. Although some changes in network structure might still result in a feasible placement, also the change of some pin positions could be required. Some algebra or transformation techniques have to be developed for this problem which is similar to the proposed methods for the case of keeping such placement structure during synthesis and logic transformations[15].

For the remainder of the discussion a pin placement is assumed to be given and that a corresponding good module placement which minimizes total wire length can be found. In our specific case we expect the pins to be evenly distributed along the sides of the given floorplan although this is not essential.

### **linear wire delay**

Wire delay and wire length has been mentioned a lot already. For now the essence was that a shorter wire in general has less delay. However the explicit relation between length and delay was not yet mentioned. In general this relation is quadratic. Wires twice as long causes 4 times more delay. This effect is exactly what makes the increase of wire length for larger systems on a chip so problematic.

Recently however it has been shown that wire delay can be made linear in length by optimal buffering for “long” wires[32, 31]. It is using the new delay modeling of constant delay as mentioned in the previous section needed to achieve an iteration free design flow. Calculating the optimal number of buffers and their sizes shows that the optimum segmentation length between buffers and size of the buffer only depend on some process parameters. Long is then defined as being one or more times the optimal segment length. For shorter wires adding a buffer does reduce the wire delay quadratic but the added delay of the buffer counters this effect and a net increase results.

It is important to note that those process parameters are different for different layers of wiring of an integrated circuit but constant within a layer. In the same publication it is shown that the lower layers are much slower than the higher layers. This means that knowing the wire distances would still not give accurate wire delays. Therefore something like a layer assignment algorithm would be required before wire planning can use wire delays based on lengths of wires.

Although this is true some assumptions relieves the problem a little. The lowest levels of wire layers will likely be used for local short wires even while they are slower. Otherwise stacks of vias would be needed to go from gate to wires and the other way around and those would block the wires being laid out in the lower levels. Also from a speed point of view it is interesting to use the higher and faster layers for long and otherwise time consuming wires.

wire planning is used for the “long” wires and thus only the higher layers are of interest. Layers seem to have two by two pairwise very similar timing characteristics. Therefore assuming wires to be on only the top two layers would allow a wire planning to be applied even without a layer assignment first. This is what will be assumed in this thesis.

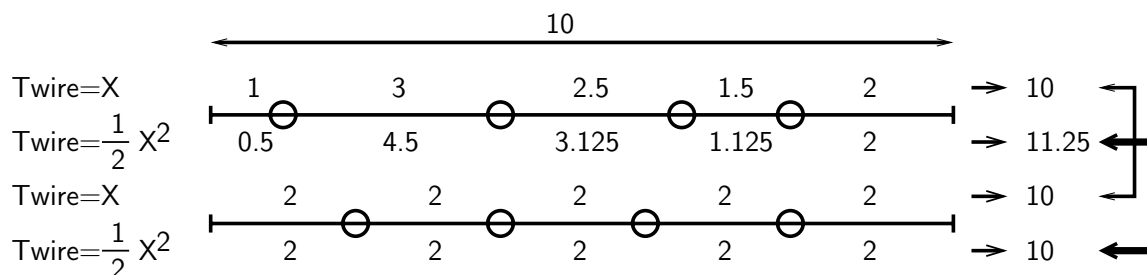


Figure 2.7: *Changing the segmentation of a fixed total length of wire does not change total delay for a linear model but does change the result of the common used quadratic model.*

The reason why linear wire delay is important is not just for the ease of calculation based on distance. The linear delay creates a property which will turn out to be very attractive in our wire planning flow. When a long wire is broken in segments by modules the total wire delay on that path will be equal to the sum of the delays of those segments. Nothing spectacular until now. But now change the segmentation of the wire by moving the models a little such that the lengths of the segments are changed but the total length is still the same (fig 2.7). Now the delay assuming linear wire delay is still equal to the previous case as the sum of the wire segments lengths and thus delays are still the same but in case of non linear delays this would not be true. In that case the sum of two times half the distance is unequal to once the delay of the total distance This linear delay observation is an important property used in the wire planning approach when calculating wire delays for paths through the circuit. Essential is to observe that the exact segmentation does not change the delay of a path as long as the total length is equal.



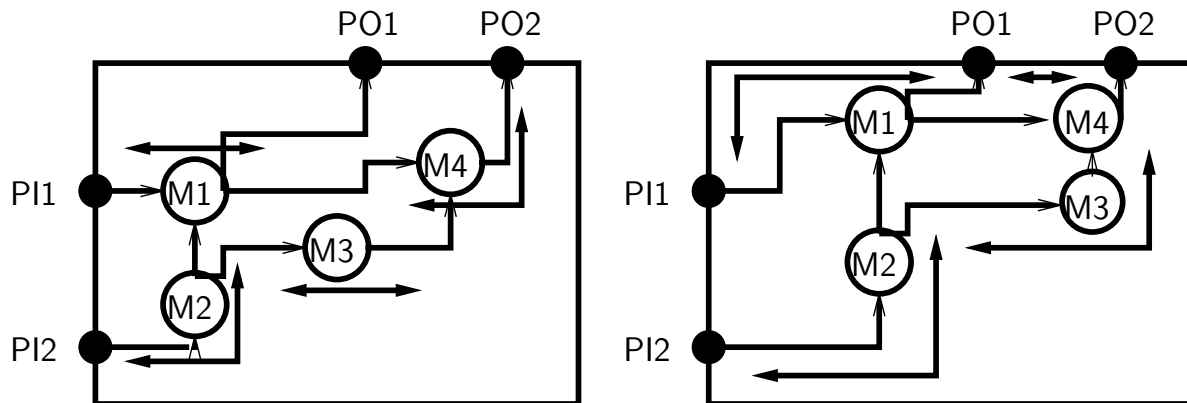


Figure 2.9: *Two equally valid monotonic placements with equal pin-to-pin path delays which is minimum and equals the manhattan distance between the pins.*

Recall that placement should try to minimize total wire length. This is clearly the case when all paths are monotonic. But if they are all monotonic the placement is also a monotonic placement. So before doing any place and route it would not be unrealistic to assume already that it would most likely become a monotonic placement or else a small deviation from it.

One may argue that not all given networks or pin placements will allow this monotonicity to be achieved based on some network structure. Although this is true this can be seen as a limiting factor of current design flows. In that case network transformations and restructuring of the network should be incorporated in the flow to achieve this best possible result. Currently the results of place and route are indeed limited by choices made at the network structure level already.

A set of transformation or high level algebra should be developed for this process. A trivial but impractical example would be to duplicate all logic in the input cone of an output and concentrate it into a single point at the output. The development of more practical transformations is still an open problem. Although this might seem hard, some progress has been made on something like this were monotonicity is preserved and sought during a synthesis optimization step[14].

Using the modeling of gain based delay shows that duplication of a node does not cost extra area as both nodes now drive only a part of the load and thus the area needed for those nodes can be equally smaller. The sum of the load remains equal and therefore the sum of the areas too. A similar reasoning show that splitting a multi output node into single output nodes would not

cost area. This is unfortunately a simplification as the impact of routing wires is not taken into account and the fact that gates will have minimum sizes and thus duplication of such a minimum sized node will produce two minimum sized nodes and thus double area.

For the rest of the current discussion we assume that placement will be able to do a good job and give us a monotonic placement. In all the possible monotonic placements it could produce all paths are monotonic. As a result the total wire length on each path is equal to the manhattan distance of the pins of the paths. As those pin locations are fixed the pin-to-pin path lengths will be the same for all possible monotonic placements.

If all pin-to-pin paths can be considered long the linear wire delay model could be applied for those paths. The total wire length of a path between two pins is the same for a possible monotonic placements and equal equal to the manhattan pin-to-pin distances. The exact locations of the modules due to a specific placement only result into a specific segmentation of this path and wire length. In the context of linear wire delay it was shown that this segmentation is irrelevant as long as the total sum stays always the same. Therefore the total delay in the wires on a path between to pins can be calculated based on the manhattan distance between those pins. This distance has a fixed length and thus delay for all possible placements This delay is also the minimum possible and thus a maximum of the time budget remains for the modules on the path.

The result is that the total wire delay is known and even minimum too already before place and route has been done and will stay that way independent of what the final placement and routing will be. This sound very interesting in a wire planning refinement context based on fixing delay first. As the objective is to do this procedure in an iteration and refinement way the placement and routing should be such that it does not change the wire delay effects on the time budgeting any more. Or to put it the other way around it should be possible to account for wire delays in a manner not impacted by placement and routing later.

wire planning has to assign delays to modules an wires such that their total on all paths meet the given timing requirements between their pins. Normally this would depend on the routing of the wires and therefore placement. Now not any more and what is more even the total amount of delay in the wires is known. So all that is left to do is to assign the remaining delay to the modules.



### constant delay model

So the delay of wires is assumed fix and given and the remaining time from the delay requirement can be budgeted over the modules. Note however that wire planning was presented in an iteration free context. Fortunately the wire delays will not change any more, but it also means that the module delays once fixed are not allowed to change any more.

In traditional gate modeling delay is a result of chosen area and the loading imposed on the output. This can be written as  $D = a * C_{load} + b$  for a given area  $A$ . The loads  $C_{in}$  seen at the inputs of the gate are also directly related to this size. Usually a technology library of gates contains a number of different sizes for every particular gate with the corresponding load depended delay data. Doubling the area  $A$  doubles also the input loads  $C_{in}$  but halves the factor  $a$  in the delay as the transistors can drive twice as much current and therefore the larger gate is faster when driving a large load. Factor  $b$ , which is the parasitic delay remains constant as the internal loads increases just as fast as the transistors can drive more current. Note the linear relations between delay and load for fixed sizes.

Originally the main objective of designs was to minimize the total area use and accept what ever delay result would come out of that. The fact that the delay of a gate was impacted by the size and thus load of another gates was of minor concern. It was the change of importance of required delay that caused this effect to become more of a problem in flows even before the impact of wire delay was noticeable.

How does this traditional delay modeling impact module delays. The delay of a gate of a given size depends on the load and thus sizes of the gates which loads its fanout. On the other hand the size of the current gate and thus load influences the delay of the gate it is loading itself. Choosing a different size and thus different load-delay characteristic to maintain a certain required delay due to a high load change at its fanout will impact the gates at its inputs with higher loads and thus cause higher delays there. As can be seen in figure 2.10 the net resulting delay might actually even be worse. Compensating also the first gate is possible, but the total effect due to the load increase has become a complicated calculation. So when the load would change at the output of a predesigned module which is a network of gates then the delay might change and this problem is not easily fixed.

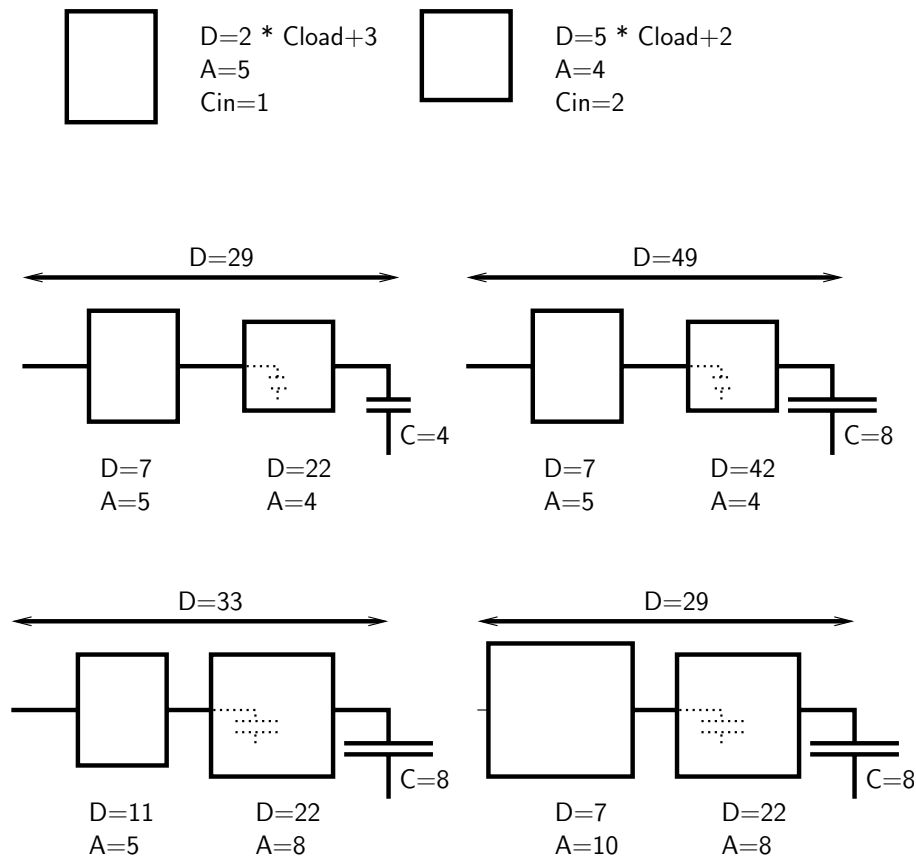


Figure 2.10: A higher load will increase total delay but increasing the size of a gate to compensate the gate delay increase will impact other gates and can in the end result in an increased total delay. The gate at the input has to be compensated too for the increased size of the other gate.

Keeping these module delays completely fixed even by changing loads therefore requires a different suitable delay modeling. As already said in the introduction (par 1.5) there exists a different approach called the constant delay methodology which was recently presented [38] [16] [17]. The essence is that by changing area delay can be kept at a certain fixed value for any load.

One way of applying this idea was to put in the library not fixed sized gates with an load-delay relation but fixed delay gates with an area-load relation. This idea was based on the observation that another linear relation existed besides the one between load and delay which is between load and area. This can also be observed in the previous example where doubling the area when the load doubles results in equal delay.

The other approach was describing the delay in a different way with the same results. In this case delay is written as  $D = f * g + p$  where  $f$  is the electrical effort and  $g$  the logical effort depending on the logic function.  $p$  is the same parasitic delay again like  $b$  was earlier. The electrical effort depends on the loads to be driven and the input load it will represent itself as  $f = \frac{C_{in}}{C_{load}}$ . The corresponding area equals  $A = a_0 * f * C_{load}$ . Using these formulas the parameter  $f$  can be used to do delay calculations and the resulting areas can later be calculated using this  $f$  and the  $a_0$  based on the load to be driven.

This later formulation captures more nicely the constant delay effects than the previous case. At the same time an classical algorithm with the function of areas and delays interchanged and with modified libraries could directly be used based on the first approach. The describing functions of those new fixed delay based modeling and the classical fixed area based modeling can be transformed into each other rather easy. Note that to drive a load which is twice as much two gates can be placed in parallel each driving half the load. As a result the delays of the gates will not change and the total area as well as the total input load is doubled. From a gain based view to obtain equal delay the  $f$  stays the same. As the load doubles the area doubles too. Using the fraction of input load and output load show that at constant  $f$  doubling the output load will double the input load.

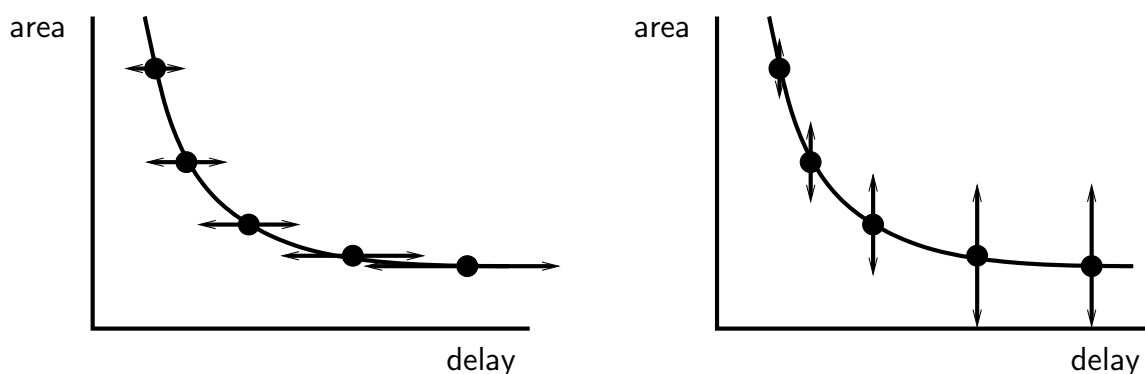


Figure 2.11: In both cases a curve has been drawn to the given implementations of a module for a specific load. The first figure shows the change due to load in case area remains fixed and the second in case delay remains fixed. Both area and delay have an linear relation with respect to the load.

So delays stay now fixed when load changes instead of fixed areas with changing delays as a result (fig 2.11). Recall that a module is a collection of gates. Now a higher load of the module can still result in the same delay by

adjusting the areas of the gates. Although this change will impact the load of the modules which it loads it still does not result in a changed total delay as also the area of that module can be adjusted such that its delay is still the same.

In fact doubling the load will double the module size due to the linear relation between area and load. The double load requires the gate it is connected to to be doubled to preserve its delay. But that does double its loads at its inputs. This doubling is seen again by the gates driving this one and they will be double in size too. So finally the whole module is doubled in size which is also the effect seen in the last image of figure 2.10.

Thus also the delay of a module which is a network of gates can be considered to have a fixed delay as no matter what changes are applied at the output. If the load of the module changes it can be re sized to still meet the required delay. This allows us to set both wire and module delay fixed as was required from refinement.

## 2.4 time budgeting

wire planning is an approach requiring a number of procedures and properties. Things like a good pin placement and a monotonic placement are required, but are in essence not specific for wire planning. Any other flow would benefit from a good pin placement and every good placement procedure should avoid detours as much as possible. The same can be said about constant delay modeling as changing delays due to loads are a common problem.

Layer assignment for wires is a procedure which is a result from the observation that it is possible to use an approach of optimal buffering to create linear wire delay for “long” wires. This linear wire delay modeling could also be applied in other flows. The result is an often easier wire calculation as the quadratic relation is gone. How this linear relation is obtained has been presented in the literature. For now the assumption that the global wires are on the top level is good enough for our discussion. A layer assignment procedure would only add an extra optimization option in wire planning.

The essential task within wire planning is the distribution of the available delay over the modules. Linear wire delay and monotonic placement with a pin placement provide for easy accounting of total wire delay without being affected by the real placement and routing following the wire planning stage. The budgeting of remaining time after subtraction of wire delay from the given timing constraints is therefore worked out further in this thesis.

Time budgeting is a procedure which takes the given network, footprint, pin positions and required delays between them and produces time budgets for the modules such that the path delays through wires and modules meet required the pin-to-pin delays and the total area fits into the footprint. The areas of the modules are directly related to the delays assigned to them.

As already suggested in the introduction of wire planning one can often envision that there will be multiple instances per module possible with different area-delay points. Different architectural choices, variations of implementations of sub functions and even sizing of gates will provide the opportunity of a continuum of area-delay points per module. This is commonly abstracted by the idea of a continues trade-off curve. Although maybe not explicitly written down it is those trade-offs that play an important role in general white board discussions in the first design stages. As a result of those alternatives for each module multiple solutions will exists for the time distribution. Choosing one of them automatically seems to suggest an optimization approach based on the areas and with the constraint being meeting the timing requirements.

This optimization could then best be the minimization of area. As area and delay have an inverse relation maximizing the area would reduce the delay further then the require delays. Besides creating as large as possible spare room between the required area and the provided area of the footprint is very useful from multiple points of view. The assigned time budgets have to be kept fixed under all circumstances. It was already stated that this could be done by using a constant delay approach which essence is that area changes can compensate for otherwise delay changes. Therefore the exact final area usage is not known on forehand and thus creating area slack reduces the risk of problems. Also modules cannot always be placed nicely together due to their shape and also area is required for the routing done later. As the floorplan is given this will leave us most room possible to accommodate all those things. The difference between available area from a footprint and the resulting total area for the modules can be seen as a measure of feasibility. If they are very close to each other not much room is left for wiring and legal placement or size adjustments for fixed timing.

Approaching the time budgeting as an mathematical programming optimization will be worked out in the next chapter. It is directly based on the observations from the previous chapter. When a floorplan shape and area and a pin placement are given the manhattan distance between the pins are known. Assuming a good placement the paths will be monotonic and thus the total wire length of a path is equal to the manhattan distance of its pins. Due to the linear wire delay segmentation and thus placement need not to be known while at the same time the total wire delay can be calculated based on the total wire length. The difference between required delay and wire delay can be budgeted over the modules on a path. As area and delay are related to each other this budgeting will result in an area too. When the total area fits in the footprint a feasible solution has been found. If the delay assignment does not fulfill the requirements it is already clear that the project is overconstrained and thus impossible.

A few remarks can be made on the procedure described above. First of all the total wire length is taken equal to the total path length. In reality they would enter a module, most likely on one side, and after a certain module delay leave it at some other side. Thus actually the total wire length is a little shorter depending on the area and pins of the module. Still it is best to assume the modules to be points and calculate their areas based on that.

wire planning is used in large system on a chip design and thus the number of modules we are looking at is hundreds or even thousands. Within the floorplan they could be assumed small or points. Another point is that this procedure is assumed to be in a hierarchical refinement context. In the next section it will be shown that there is even a compensating effect for this erroneously accounted for wire delay were it would be inside a module. For now note that using this point assumption allows us to proceed without iteration as the resulting areas do not impact the way the calculation was performed.

Another aspect is the loads and the fixed delays of modules. The optimal size of a buffer in an optimal buffered wire has a fixed size depending on layer. As the layer is equal for all wires the size of the buffer and thus the load seen at the output of an optimal buffered wire is fixed too. Therefore in the remainder of the thesis the load dependent part of the delays is left out or put into a constant. This does actually mean working with a known load for all modules. In fact the buffered wires are shielding the loading effects of one module on another. If this load changes however for what ever reason then area changes can be used to compensate.

The slack area results in a feasibility check as lots of potential errors due to bad estimates can be solved by using area. First of all because area can be exchanged for delay. Errors due to detours because of placement problems can then be solved. Indirectly these errors can be caused by final resulting shapes which require some legalization changes of the initial optimal placement causing detours. Another big unknown is the required area for wire routing. Again even in this case when routing causes a detour of a wire again area could be trade for delay to compensate for the higher wire delay.

Another problem is introduced by allowing networks with cycles. These conflict with the detour free assumption as the detour free cycle has length 0. It can actually be regarded related to the pin placement problem. It is common practice in timing oriented procedures to break cycles in the network at latch or flip-flop boundaries and treat the input and outputs of them as outputs respectively inputs of a now cycle free network. Arrival times for those inputs are set to zero and the required time equal to the projected cycle time. As wire planning requires the locations of pins, also the locations of the latches or flip flops and thus virtual pins have to be predetermined during or in cooperation with the normal pin assignment. Their locations are somewhat predetermined already by the fact that clock distribution has to be such that it arrives at every node at the same time. In principal the cycles are contradicting the detour free requirements as their minimum wire length would be zero.

In practice assuming the latch assignments has taken place only a cycle containing a single latch would result in that requirement. First of all this would result in a very local loop where wire delay is of no concern and besides that there would be much more other constraints which would set the delays of logic on the path such that there would likely be time budget left over on the cycle to allow this cycle to detour. Therefore taking the above into account for now networks are assumed cycle free.

## 2.5 hierarchical context

The earlier mentioned aspects like the given footprint and pin placements and pin-to-pin delays also create a natural possibility to apply the idea of wire planning in a hierarchical context. It is very likely that the procedure still has to be applied for the first levels after top level when these wires still might be

considered “long”. That is a length of one or more times the critical length such that their delay is significant compared to module delays. On the lowest levels of the design still fit the capability of current tools as the length of wires will be “short” and thus their delay might be more or less neglected and therefore the current approaches will still work.

wire planning assigns delays to modules and thus a particular module gets pin-to-pin delays for its inputs and outputs. The time budgeting was based on center to center point wires while the actual pins will be assumed on the edge of the module. It is actually also allowed to put pins more inside the module as long as they are placed inside the quadrant around the center which is closes to the other module the pin is connecting to. This has no fundamental impact on the wire planning process and therefore a similar model with pins on the sides will be taken for the modules as this matches the assumption made for the pin placement of the total chip. As said before this will result in a minor change and as the actual length of the wires between pins will be shorter and a little spare delay or slack is created but the total path delays are not influenced.

Using the area-delay trade-off of a module a corresponding area is obtained. This area provides a floorplan area but no shape. This shape is also required as without it no wire planning and time budgeting can be performed for within the module. What is missing is the pin-to-pin wire delay within the module. This is obtained from the pin placement with is related to the floorplan shape and area.

The shape if not known in any better way can be approached by a square of equal area as could also be used during the initial floorplanning process after the wire planning which produced the delay values. Wire planning is assumed to be done in the context of hundreds or maybe a thousands modules. Then modules themselves are relatively small compared to the total area and their exact shape will not have a major impact on exact placement.

These assumed square modules will at least provide a shape and total area available for the modules and can be considered a footprint for these modules. Using these footprints placement can be performed after wire planning and this will give the positions of modules. This placement is also monotonic and thus routing is supposed to be able to create detour free connections and thus a pin will be on the side of the module floorplan box on a place close to the other module. The exact location can vary as the place where a monotonic path between the centers of the modules leaves or enters the modules can be on different places (fig 2.12).



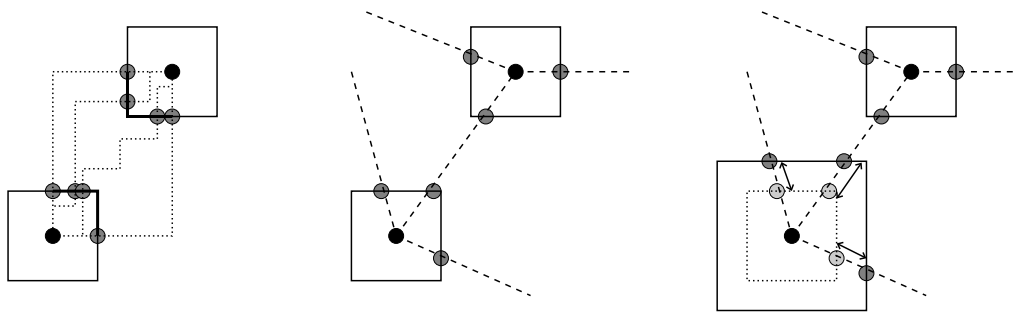


Figure 2.12: *A range of possible pin positions are in general possible. A heuristic to put it on a particular place uses the counterpoints connecting line. When due to an error the size of a module changes increasing the internal wire length and delay this is equal to the extra saved external wire length and delay*

To obtain a complete pin assignment the following heuristic can be used. Put the pin at the point where the direct line between to two centers of the modules crosses the sides of the floorplan box of those modules. These positions will automatically still always allow a detour free connection. In principal the pins could move a little along the side but then care should be taken that the connection can still be guaranteed to be detour free by looking at the pin positions of both modules at the same time. The proposed heuristic avoids this complicating factor.

Using the results of the current wire planning and floorplanning it is then possible to create the required data for the modules in such a way that the same procedure can be applied to the modules itself to refine the design to a lower lever. Note also that those modules can be processed independently and in parallel as they do not depend on each other any more due to the fixed timing and wire delay and likely placement. This process of refinement can go on until long wires do not exist any more and a more conventional flow will produce an implementation. Then the process of assembling the lower levels implementations can take place until the top level is reached again similar as was shown earlier (par 1.6,fig 1.9).

One clear source of potential error in the above mentioned approach is the assumption of square floorplans and thus shapes of modules or the areas of them. This has a direct impact on the budgeting performed through the estimated wire delays based on the pin placement which again is based on these floorplans. If the final implementation would result in more wire delay between pins as they become placed farther apart this could be solved internally by

consuming a little more area and speeding up some paths. Note that at the same time there is some slack delay as the higher level budgeting was assuming center to center points wires. This slack could compensate for the increased internal wire delay and the good news is that in fact this slack increases when pins move farther apart from this center by a changing floorplan.

It might seem contradicting to do a time budgeting of an assumed point placed circuit and then talk about areas and pins on the edges. But wire delay starts with the best knowledge available which is pin placement, delay and circuit network to assign delays to modules. The modules are now refined to have a certain delay and by use of area delay trade off a corresponding area. In the refinement approach the delays are now fixed and kept that way and the area can be adjusted to keep it so. Also the general assumption of a monotonic placement can now be changed into a best monotonic placement which allows for the areas as obtained.

As already argued there refinement is still able to cope with errors and will consume them as gracefully as possible. The choices taken for the refinement process eases this problem too. Errors are due to bad area delay predictions of the modules, different final shape or area of a module impacting placement and even wiring. Still wire planning is giving a feasibility check point at every level for example about meeting the delay within a certain required area. The difference between the required area and available area can also be seen as a credibility measure how likely the design is still to succeed in case of some small errors in estimations. Over estimates are no problem at all, and underestimates might be consumed by using the over estimates or even the left over area. While consuming this spare area modules can be made faster and by that resolve timing conflicts.

Assuming area can be traded for delay this can solve a lot of issues if area is available. As said before choosing the alternative budgeting of delays among modules which has the least area cost is thus attractive. It creates for us the spare room to do adjustments to account for errors, while still keeping the delay which we fixed by wire planning unchanged even when errors occur.

A number of potential error sources exist in the process of hierarchical refinement. The problem of floorplan shapes and areas needed for the induced pin placements for the modules to refine the design on a lower level of hierarchy was already mentioned. A choice has to be made about which estimation to use to be able to continue the process. As already shown the possible change in

shape or size is not likely to have a major impact on the decisions based on it. Some sort of compensation exist between internal wire changes and external wire changes for a module. But there are more differences possible between the assumptions made in the refinement on the way down to lower levels and the final results constructed and being propagated back to the higher levels.

The final placement can differ from the initial floorplan based on the estimated floorplans of the modules. Changes in shape from square to some rectangle might require some shifting around of modules to nicely fit together. The same is true for changes in total size. But even then it is likely that the centers are still close to the initial locations and as long as the locations are such that still a monotonic path exists it does not affect the calculated time budgets. The change of centers is then only changing the segmentation of the path, which was irrelevant. They can be seen as a variation of comparable monotonic placements and will have similar relative positions and thus almost equal pin placements too. This allows for recovering from these possible errors during the assembly of the final results as the actual error in the wires stays small and is partly compensated automatically.

It might also be impossible to change the monotonic point placement into a valid legal placements with the resulting areas and shapes. Extra delay for required detours might then be created by consuming some left over area to speed up one or more modules. On the other hand recall the fact that wires arrive at the boundary of the area and not in the middle resulting in some slack delay. This can also be used for some detouring or non ideal pin placement. Therefore wire planning still work rather well even when a perfect monotonic placement is not available.

A much more severe error is due to bad area-delay estimates. This is not as easy to handle as the other cases which results in small deviations from the original assumption except maybe for an extreme change in floorplan shape. A large change in area compared to the estimation to be able to meet the required delay generates two problems. The major problem is deviation from the optimum for the wire planner result. If this would have been taken into account during wire planning already a much better and different time budgeting might exist. Now it might even render the implementation impossible as there is not enough spare area to overcome this change. The second impact is of course on the placement itself. It will cause a more significant deviation than just some mismatch. At the same time a slicing floorplan can often consume

even large changes quite well and only a part of the placements gets more heavily affected. Most parts will keep rather equal positions and distances and still work out fine. So only for the highly changed module and its close surroundings there should be enough area left to both accommodate the larger footprint of the module as well for exchange into delay to fix errors in wire delays as they become larger now.

Using previous design knowledge could highly reduce this risk as well as a careful design process. When doing a next layer of refinement it would be better to work on the risky elements first and even go to deeper levels already for this particular module. The fact that the designs are decoupled using refinement and hierarchy allows this to be done. At all time the results from a certain deeper level could be propagated back up as being more realistic data. If it turns out that a major change occurs then the high level wire planning could be repeated with the current already further designed modules fixed or if willing to redesign them too with updated information for them. Then the less unknown modules can be implemented with high assurance of success. This aspect increases the advantage of an iteration free approach. Only when allowing redesign of the problematic modules it is not completely iteration free any more. Still the changes due to the iteration are not that costly as most of the design was not fully designed yet.

From the above it is clear that even bad estimations can be handled rather well. At least their impact is not worse compared to the impact of this problem also on common flows. Also in those cases some implicit or explicit trade-off is used in a high level white board discussion. In the context of wire planning refinement this has been made explicit and the effects can be better understood.

In general the hierarchical application of wire planning will not be problematic as long as estimates will be good enough. The procedure fixes in a logical way certain data and provides other required data automatically. Some errors in estimations are automatically compensated or even irrelevant for some of the decisions taken. It speeds up the design process as at every level of detail the delay is assured to be met and the required area can be compared to the available footprint to detect a failure early and also no countless iterations are required.

## 2.6 algorithms

From the previous sections the following can be concluded about required algorithms and procedures for a complete working wire planning flow. As monotonicity is important and dependent on the pin placement a good pin placement has to be found. Unfortunately pin placements can be rather fixed due to the environment and thus the network should be modified when needed. This calls for monotonicity aware transformations for the logic network similar as presented for logic synthesis. The wire planner takes wire delays into account based on wire lengths and these delays can vary over different layers such that a layer assignment procedure is required. Using the provided information and the existence of the required properties an algorithm can assign the optimal time budgets to each of the modules. These budgets can only be assigned when the modules provide area-delay trade-offs and it makes only sense if the assigned delay can be kept that way for the synthesized results. Not all of these procedures will be presented in this thesis. The focus will be on the budgeting step calculating optimal time budgets for the modules and how synthesis can provide a results with exactly the required delay among the complete area-delay curve that could be produced for this module using a delay model which allows delays to be maintained fixed.

The main task of wire planning is to assign delays to modules. This time budgeting should try to choose the alternative with minimum total area use as available left over area can be used to make up for changes while keeping delays fixed in the rest of the refinement. To make maximum use of the power of wire planning and refinement there should be an as large as possible range of delays and thus areas to choose from for each module. The selection of the particular solution for each module from these ranges which is the optimal solution with regard to minimizing the total area shows that this can be cast into an optimization scheme.

This optimization scheme is somewhat similar to transistor sizing techniques to reduce area while meeting delays. Most solutions for that problem use a path by path based optimization approach which does not give a global optimum or are heuristic based. Only for small sizes full optimization can be achieved. Exploring particular properties of the problem at hand in our case resulted in the development of a procedure which takes all paths concurrently into account while still allowing large enough sizes of networks to be handled. This allows for the problem to be solved to a global optimum and is described in the next chapter.

The wire planning approach assigns explicit delays to modules. This differs from common flows where the choice is more often from some instances like usually the fastest when critical and smallest if possible. The refinement approach also requires delays to be kept fixed although some properties like load influences delay but at the same time is not set to a fixed value or might still change. This requires a different logic synthesis and technology mapping approach which we will present as a constant delay synthesis approach. This does not only point toward the more common meaning of a delay which is being kept constant although some parameters might influence it, but also to the fact that it allows to choose a particular delay (constant) among a range of possibilities from minimum area, and thus higher delay, to minimum delay.

Constant delay synthesis comes up with the instance which precisely meets the required delay at the least cost. This can be done by an exploration of a complete area delay space and keeping the interesting points. A well known problem of reducing the complexity of design space exploration is that choosing a solution already at some intermediate level to reduce the size of the design space already at that point can prevent the best possible final solutions from being reached if this intermediate choice was wrong. This approach to reduce the design space is usually applied in common synthesis and technology mapping and can have limited effect for minimum area or delay results as there modeling on intermediate levels can then be quite accurate. But having to provide a range of solutions makes it attractive to explore a design space as large as possible and with the refinement approach in the back of our mind a strategy was applied to relieve those problems of intermediate solutions and at the same time expand the explored search space.

The other requirement of keeping the final delay constant can be fulfilled by choosing another point from the exploration space as soon as something changes, but another view of constant delay synthesis is that it is possible to come up with an implementation which can be adjusted to the required delay by sizing. This is due to an other type of delay and area modeling. It allows efficient resizing of the sizes of the elements inside the module as soon as load would cause a change in delay. It is the combination of design space exploration and sizing which renders these changes in synthesis and technology mapping attractive in the wire planning context and are therefore researched and presented later in this thesis.

# Chapter 3

## time budgeting

In the previous chapter a flow was sketched along which a timing-driven back-end for chip design can be organized. That sketch was however not without lacunae. Some of these can be filled by modifying existing algorithms to fit in the present context. Others require a fresh inquiry and probably a largely new approach. One of those is time budgeting. In this chapter we will present a mathematical formulation of that problem based on the assumption of monotonicity, linear wire delays and convex delay-area trade-offs for the modules. The formulation allows a straight-forward implementation yielding optimal budget allocation, but prohibitive time complexity. We therefore introduce intermediate variables that absorb most of the usually large number of potential substructures at each network node. It produces an equivalent budget allocation, but the problem size grows only linearly with the network size. The chapter is rounded off with a number of improvements, two of which reduce the problem size with another fifty percent by adapting it better to the chosen solver, and one that enhances the robustness of the budget allocation.

### 3.1 the problem in a wire planning context

Time budgeting is the assignment of delays to wires and modules in a functional network under given constraints. Here the constraints concern the space in which the modules have to be allocated, for convenience a rectangle of given dimensions, and the timing requirements for signals at the pins. The delay is usually related to size: for modules this size is area, while for wires it is length.

Faster modules for the same function are likely to demand more area and longer wires cause longer delays. So we assume that at the stage of time budgeting in the context of wire planning the network, the monotonic pin positioning and a complete set of timing requirements to be available.

The derivable pin-to-pin delays constrain the path delays the latter being equal to the sum of the delays on the wire segments  $W$  and the delays of the modules  $M$  on the respective path. The sum of the areas of the modules is constrained by the total available area from the given floorplan. Note that in practice full utilization is mostly neither achieved nor desirable as some area is needed for routing wires and not all modules in the final placement will fit precisely together.

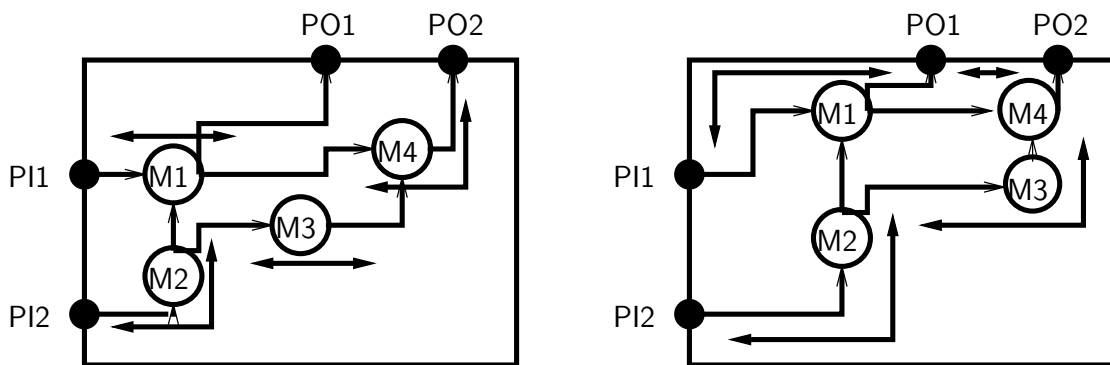


Figure 3.1: A *monotonic wire plan*. Figure 3.2: Another *valid monotonic wire plan*.

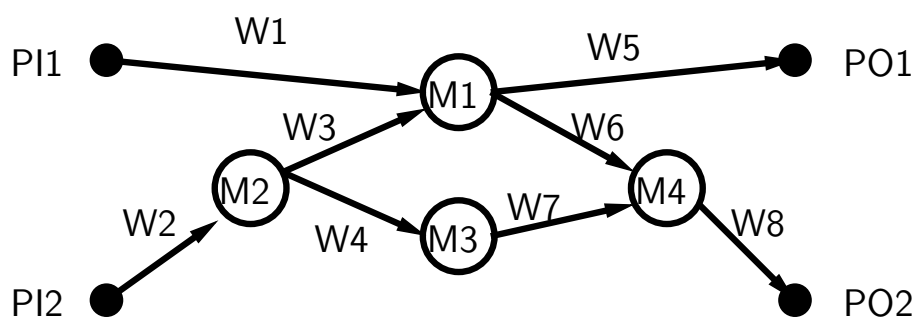


Figure 3.3: A *corresponding network*.

In section 2.3 a number of observations which were made on monotonic placement. Three of them are of particular interest for our approach for time budget allocation to modules and wires. First, the fact that under the optimal segmentation total wire delay is independent of the exact position of modules: it will equal to the total wire length multiplied by some constant factor. Second,



the total sum of the lengths of the wire segments on a path is invariant over all possible monotonic placements of its modules and solely determined by the pin positions. The sum of the wire segments  $W2 + W3 + W6 + W8$  of the circuit in figure 3.3 will be equal for both given monotonic placements of figures 3.1 and 3.2 and will equal the manhattan distance between  $PI2$  and  $PO2$ . The third observation is that this is also true for all other paths between the same primary input and primary output and thus they all will have equal total wire length. The sum of  $W2 + W3 + W6 + W8$  is equal to  $W2 + W4 + W7 + W8$  which in turn is equal to the manhattan distance between  $PI2$  and  $PO2$  for all possible monotonic placements.

So the total sum  $D_{W_{jk}}$  of the delays of the wire segments is always equal and determined by the manhattan distance of the pins  $j$  and  $k$ . The delay of a path is then simply the sum of the delays of the modules  $D_m$  on a path plus the total wire delay  $D_W$  based on manhattan distance. Therefore delay calculations can be done without knowledge of the delays of the individual wire segments on a path, and also without knowledge of the exact positions of the modules.

With the delay due to the wire segments fixed for all monotonic placements the remaining total delay available for the modules on a pin-to-pin path is also fixed for a given timing constraint  $T_{req_{jk}}$  between the same pins. This remaining delay  $T_{req_{jk}} - D_{W_{jk}}$  can be distributed freely over the modules as long as those delays can also be realized by the modules. Of course, not every distribution over modules is equally desirable or some may even be not feasible due to the limited available total area.

In general there will be a whole range of solutions which fulfill both the area and delay constraints. In fact both area and delay have to be just below their required values. Yet, it is not simply the groove of synthesis (1.2) to choose for minimization of one objective while fulfilling exactly some other requirements. Minimizing the total area of the modules under the timing constraints leaves the maximum amount of space for adjustments that might be useful when accommodating wiring, picking library elements or correct for wrongly predicted delays.

## 3.2 mathematical problem formulation

The objective of our time budgeting is to minimize total module area while avoiding timing violation for all paths with respect to the given pin-to-pin delays. This can easily be written down as an mathematical programming problem :

$$\min \sum_{i=0}^{\#modules} A_{m_i}$$

subject to

$$\sum_{i=u \forall m_u \in P_{jk}} D_{m_i}(A_{m_i}) + D_{W_{jk}} \leq T_{req_{jk}} \quad \forall P_{jk} \forall j \forall k$$

where the delay  $D_{m_i}(A)$  is a function of area  $A_{m_i}$  of a module  $m_i$  and the  $D_{m_i}$  are the delays of the modules  $m_u$  on path  $P_{jk}$  which is one of all possible paths between the primary input  $PI_j$  and primary output  $PO_k$  with a calculated wire delay of  $D_{W_{jk}}$  based on the manhattan distance of the pin positions and  $T_{req_{jk}}$  the required time between the two pins.

Often  $D_{m_i}(A)$  is just a number of area-delay points. Enumerating all possible combinations of sizes and delays for all modules is clearly impractical for real designs, because of the high complexities involved. True for an optimal solution which does not waste area or delay only *pareto points* are of interest.  $(d_1, a_1)$  is a *pareto* point if there is no feasible pair  $(d_2, a_2)$  such that:  $(t_2 \leq t_1 \wedge a_2 < a_1) \vee (t_2 < t_1 \wedge a_2 \leq a_1)$ . If for some node an equally fast but larger choice was made it would waste area and thus has not to be considered at all. The same is true for nodes which are slower but just fast enough but at the same time require more area then a faster alternative. Although this will reduce the area-delay pairs to be considered the run-time involved will still be prohibitively long.

A common approach in these cases is to break the problem in sub-tasks and solve those more smaller and manageable tasks, after which the results have to be combined for a complete, generally sub-optimal solution. Optimizing one path at the time is an example. Such an approach is chosen in some transistor

or gate sizing techniques like TILOS[12] where one transistor at the time on a particular path is optimized with heuristics. iDEAS[34] improves on that by optimizing more transistors on that path. Lagrange multipliers were used in a method described in [13]. iCOACH[7]. has an outer and an inner loop The outer loop does some time budgeting based on simple models and then in the inner loop the transistors are tuned to meet those budgets.

Major problems with those methods are convergence and the possibility of getting stuck in a local optimum. The final result is then usually very dependent on the ordering of paths or the initial delay distribution. This is not surprising since paths mutually share segments Adjusting paths several times does not help much in making the result less sensitive to path ordering. It might even create an increased risk of non-convergence when two or more paths cause opposite effects for a common node.

A more fruitful way might be to look at efficient mathematical programming techniques and see whether we can relax some of the constraints without sacrificing significantly in the outcome. The prime candidate for this seems to be the modeling of the area-delay trade-off. Since wire planning is started at an early stage in the design flow, there is still quite some uncertainty in the data, and these “area-delay points” are at best local clouds, that only later in the flow shrink to better defined “dots” and only after refinement is complete become points. In general the data in the earlier stages are just rough estimates or approximations based on previous designs and experience. Therefore changing from a discrete to a continuous area-delay description is quite acceptable.

Maybe the first thought then is to adopt a piece-wise linear trade-off, enabling solvers for linear programming that are well-established and well-researched. It can be polynomial in the size of the tableau, and it is certain to return a global optimum. However, each “piece” adds a constraint (the number of “pieces” will be one less than the number of *pareto* points), and with the high number of paths, it leads to very large problems with still problematic run times.

But there are more continuous functions than piece-wise linear functions to be fitted to a set of *pareto* points in area-delay trade-offs. The mathematical programming would no longer be linear, and only in special cases we may end up with an efficiently solvable non-linear optimization problem. Such a special case is within reach, however, when the approximation of the *pareto* curve can

be a posynomial function. A simple transformation can then turn the problem into a convex optimization problem that not only yields a globally optimum result, but it can also do so in computation times polynomial in the size of the formulation. This technique is called *geometric programming*.

So, in return for some detail and precision, that seldom is justified at the levels where wire planning is applied, we obtain an efficiently solvable optimization problem, though the size of the formulation is still a point of concern. But our chances are much better than when we stick to path-by-path techniques, which usually also use functions to capture the area-delay trade-off, but seldom can guarantee global convergence.

### geometric programming

When a posynomial approximation is used for the trade-off an efficient solvable geometric program with an global optimal solution is obtained. A posynomial function has the following form:

$$f = \sum c_j \prod s_i^{a_{ij}} \quad \text{with } c_j \geq 0 \quad \text{and } a_{ij} \in R$$

In this case the object function as well as the constraints are posynomial functions. A transformation to a new variable  $z$  done through  $s_i = e^{z_i}$  of the posynomial functions results in convex functions again and also general convex programming could be used but would of course be less efficient.

Assuming that some resemblance will exist between basic gates and the modules built using them a posynomial approximation of the trade-off would look like:

$$D(s) = \frac{d}{s} + p \quad \text{and} \quad A(s) = a_0 s \quad \text{which can be changed to} \quad D(A) = \frac{d \cdot a_0}{A} + p$$

It is also possible to use more complex posynomial approximations using higher order terms, but it does not really change the main idea. Taking into account that wire planning is done at a high abstraction level it is reasonable to take only a coarse approximation with a single term. In numerous cases there will be no exact data even, but only extrapolated figures.

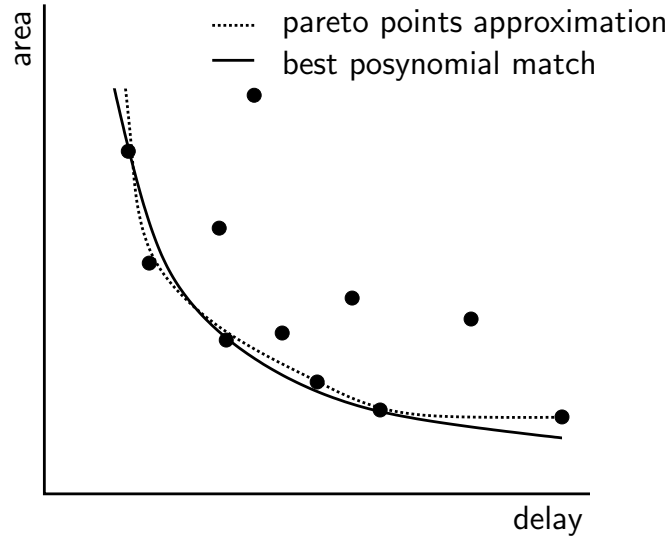


Figure 3.4: A higher order polynomial approximation fitting all pareto points as well as a simple posynomial curve fitted through as many pareto points as possible

For now the delay and delay of a module is assumed to be given as a simple posynomial function:  $D(s) = d/s + p$  and  $A(s) = a_0s$ , where  $s$  is the size we are solving for and which relates the area of a module to the delay. For all modules,  $d$ ,  $p$  and  $a_0$  are constants and given. The total area becomes  $a_{0_1} \cdot s_1 + a_{0_2} \cdot s_2 + \dots + a_{0_n} \cdot s_n$  for the  $n$  modules which is also a posynomial function. The same is true for delay constraints which read now  $d_j \cdot s_{i_1}^{-1} + p_{i_1} + \dots + d_{i_n} \cdot s_{i_n}^{-1} + p_k + D_{W_{jk}} \leq T_{req_{jk}}$  for modules  $i$  on the path between primary input  $j$  and primary output  $k$ . The resulting mathematical program using variable  $s$  is:

$$\min \sum_{i=0}^{\#modules} A_{m_i}(s_i)$$

subject to

$$\sum_{i=u \forall m_u \in P_{jk}} D_{m_i}(s_i) + D_{W_{jk}} \leq T_{req_{jk}} \quad \forall P_{jk} \forall j \forall k$$

where the area  $D_{m_i}(s_i)$  and  $A_{m_i}(s_i)$  are the delay and area of module  $m_i$  expressed in sizing parameter  $s_i$ . The  $D_{m_i}(s_i)$  given by  $u$  are the delays of the

modules  $m_u$  on a particular path  $P_{jk}$  which is one of all the paths between primary input  $PI_j$  and primary output  $PO_k$ . Wire delay  $D_{W_{jk}}$  is based on the manhattan distance and delay  $T_{req_{jk}}$  the required delay between those two pins. To avoid awkward oblong shapes minimum and maximum module dimensions can be imposed by adding extra posynomial constraints in the following form.

$$\minSize_i \cdot s_i^{-1} \leq 1$$

$$1/\maxSize_i \cdot s_i^1 \leq 1$$

The resulting program calculates a global optimum. It takes into account the effects of all possible paths and therefore gives a better result than methods which try to optimize path by path.

The program can be solved for small cases. Larger networks however, can become problematic due to the exploding number of paths: it is realistic to assume an exponential relation between the number of modules  $M$ , and the number of paths  $P$ . In a typical worst case it could be even  $P = O(2^M)$ . Although usually less in practice the number of paths are still overwhelming in realistic cases. The average number of modules on a path is  $O(M)$ , making the total number of terms in the constraints  $O(M \cdot 2^M)$ . The number of variables is equal to the number of modules.

### 3.3 problem size reduction

#### path enumeration

To verify the capabilities of the presented mathematical programming solution experiments are needed. A network of modules, timing information for the modules and input-output pin-to-pin distances are required for the experiments. As no real data was easily available a choice was made to create this data based on a set of well known MCNC benchmarks.

The chosen set was the whole range of C17 to C7552. They are technology mapped using SIS[40] and the included technology library *lib2.genlib*. This offers a network of modules which are represented by library elements with an area-delay model also derived from the used *lib2.genlib* library. Although

this results in lots of small nodes with a single output this is still useful as a real multiple output node could be imagined to be broken into multiple single output nodes as far as the modeling in the mathematical programming is concerned.

For the pip-to-pin distances random lengths limited to some maximum were first added to all edges connecting the modules. The maximum distance of all the paths between to pins using those numbers was taken as the total pip-to-pin distance. The required delay was set between the minimum possible delay and the delay at the minimum area. It is set at  $1/4$  of the total difference from the fastest solution and  $3/4$  from the smallest solution.

The experiments ran on a linux intel 1GHz PC with 512MB memory. Java is used for the handling of the delay and network data and the interfacing to the solver. The network model in java is used to do the ordering of nodes and analyses as well as manipulations on the network. The interfacing to the solver was done through writing and reading files and using matlab. The solver used was a geometric solver in MOSEK [29] which has a matlab toolbox interface.

| Circuit | #modules | #inputs | #outputs | #paths | # modules total in paths |
|---------|----------|---------|----------|--------|--------------------------|
| C17     | 6        | 5       | 2        | 11     | 28                       |
| C432    | 147      | 36      | 7        | 291e3  | 464e4                    |
| C499    | 287      | 41      | 32       | 100e3  | 132e5                    |
| C880    | 225      | 60      | 26       | 8442   | 943e2                    |
| C1355   | 510      | 41      | 32       | 417e4  | 881e5                    |
| C1908   | 349      | 33      | 25       | 196e3  | 3278e3                   |
| C3540   | 740      | 50      | 22       | 225e5  | 509e6                    |
| C5315   | 1081     | 178     | 107      | 395e3  | 764e4                    |
| C6288   | 2371     | 32      | 32       | 538e17 | 481e19                   |
| C7552   | 1682     | 206     | 58       | 428e3  | 663e4                    |

Table 3.1: *Circuit parameters for some mcnc benchmarks*

The following table 3.1 shows numerical data of the networks used. As expected the number of paths increases quickly with higher number of modules. Writing down all paths for C6288 is clearly not feasible and that alone would require a long run time already before doing any solving already. In fact it turned out to be even impossible to run the C432 circuit from both a memory as run time point of view. A very cut down version running on a 4 GB machine had run times well into hours.

Enumeration of all possible paths is clearly a problematic issue. To get some idea how badly this gets out of hands look at figure 3.5. Adding the dashed edge will introduce an additional number of paths equal to the product of the number of outputs of node B and number of inputs of node A. In case  $N_i$  and  $N_o$  are 5 the number of constraints rise from only 10 originally with an additional 25 paths to 35. Adding another additional edge will result in an even higher increase of paths. If the edge C to A is added C 'sees' the  $N_o$  primary outputs of B too through the previously added edge A to B. The same is true for the  $N_i$  primary inputs of A for added edge B to D.

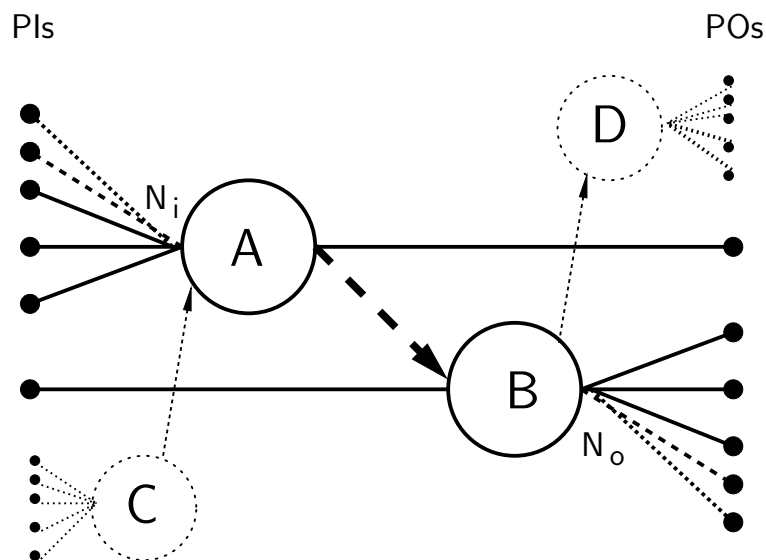


Figure 3.5: Adding the dashed edge will introduce  $N_i * N_o$  additional paths. Adding nodes C and D with their edges will give an even higher increase of number of paths.

### dynamic programming

Table 3.1 shows that enumerating all possible paths is impractical, and this goes then for the mathematical program too. Fortunately there exists a way to circumvent this problem and still take all paths into account. The dependencies among paths which was earlier mentioned as a problem for the path-by-path algorithms can actually be exploited here. By a reformulation of the information contained in the path delay constraints of the mathematical program the total number of constraints reduces dramatically. As no model reduction or heuristic pruning will be used exactly the same problem is solved and the same solution obtained as in the unreduced case.



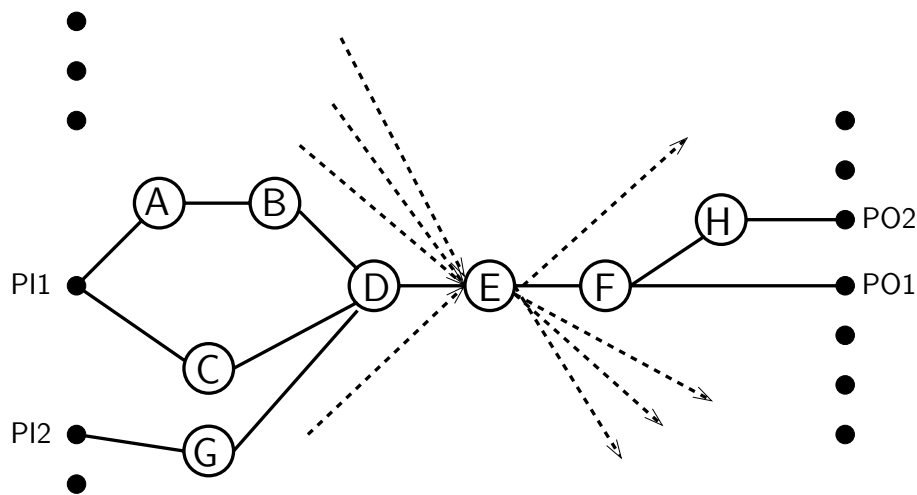


Figure 3.6: *Some circuit paths with partial shared sub paths*

Look at the situation of figure 3.6. Note that all possible paths between a particular primary input  $PI1$  and a particular primary output  $PO1$  have to comply to the same timing constraint. As seen before, due to the monotonic placement assumed in the wire planning context all those paths will have an equal amount of wire length and thus wire delay related to the manhattan distance between this input pin and output pin. Therefore module delays could be separately summed up and compared to the required delay minus the wire delay. Of all those alternative paths only the one with maximum delay has to be known. If this maximum is below the timing requirement minus wire delay then the timing requirement is met. It also means that all other paths meet the timing requirement too. So not all paths have to be set as timing requirement, but only the ones with maximum delay.

Note also that lots of paths share common sub-paths. A very strong example are the paths between  $PI1$  and  $PO2$  which are identical to those between  $PI1$  and  $PO1$  with only  $H$  added. As  $H$  has a specific although unknown location on the chip the amount of wire delay between  $H$  and primary input  $PI1$  is fixed for that location. As a result it is also equal for all paths going through  $H$  and thus also for those from  $PO1$  as well as  $PO2$ . Therefore the amount of module delay added up from the modules on a particular path up to this module  $H$  is also equal for both paths going to  $PO1$  or  $PO2$  which share this particular common sub-path. So the delays of the paths to  $PO2$  are equal to those at  $PO1$  with only the delay of  $H$  added to them. Finally this also means that the path which results in maximum delay at  $PO1$  automatically results into this path with  $H$  added being the path which will be maximum among all paths from  $PO2$ .

A similar observation can be made for the paths  $A B$  and  $C$  between  $PI1$  and  $D$ . For any given position of  $D$  in a monotonic placement the amount of wire delay between  $D$  and  $PI1$  is equal. If the delay of  $A$  and  $B$  is higher than  $C$  then for any path through  $D$  to  $PI1$  the paths with highest delay will take the  $A B$  alternative path. So when searching for all alternative paths with maximum delay at the primary outputs it can already be determined at module  $D$  that only the ones going through  $A B$  have potential and that the amount of this delay will be equal for all paths with maximum delay going through  $D$ .

Having a single optimal alternative option even at intermediate points and the fact that those optimal options are related to each other and shared a lot seems to call for a dynamic programming approach. In general the following properties should exist:

- There exist optimal substructures for an optimal solution of the problem.
- There exist a recursive way to define optimal values for those substructures.
- The optimal solution can be calculated in a bottom up fashion.

Recall that we are searching for a path with highest delay. So within the following discussion the optimality which we look for is the alternative pin-to-pin paths with highest delay. This is the structure of an optimal solution. Substructures are in this context sub-paths. The optimal substructures are part of the optimal structure.

**Theorem 1** *Any sub-path between two modules  $A$  and  $B$  of a larger sub-path which has maximum delay has also the maximum delay among all alternatives between those two modules  $A$  and  $B$ .*

*Proof: Suppose there is a path between primary input  $PI$  and primary output  $PO$  which has the highest delay and goes through a particular path from  $A$  to  $B$ . Now if there would be an alternative between  $A$  and  $B$  with higher delay then the path taken by the path from  $PI$  to  $PO$  then an alternative path between  $PI$  and  $PO$  would exist with a higher total delay. This contradict the fact that we had the path with maximum delay.*

The calculation of the maximum delay value of the optimal substructures can easily be done. The previous theorem already stated that any substructure will be constructed from other optimal substructures as any sub-path can be broken into smaller sub-paths which will be optimal.

**Theorem 2** *The maximum path delay from a certain pin or module  $P$  to the current module  $M$  is equal to the maximum of all maximum path delays coming from the same pin or module to the module  $N$  at any of the  $n$  inputs of the current module plus the delay of the current module.*

*Proof:* Assume maximum path delays from a certain point  $P$  to be known for all  $N_i$  modules at the  $n$  inputs of  $M$ . Then  $n$  alternative paths to  $P$  can be constructed by adding the delay  $D$  of module  $M$  to the values at the modules  $N_i$ . As the same value  $D$  is added to all alternatives the resulting maximum delay path will be based on the maximum of all maximum delays at the  $n$  inputs.

The value for a module can thus be calculated based on its direct inputs. As delay budgeting is pin-to-pin based the logical choice for the reference pin  $P$  in the previous theorem to start from would be a primary input. When arriving at the outputs we know the maximum delay whenever there is a path.

Usually multiple primary inputs have a path to the same primary output. We can easily take all of them into account by identifying a separate maximum path delay for each primary input at each module. They do not interact and the maximum of all inputs is just based on the corresponding maximum delays towards the corresponding primary input. Therefore a topological ordering of all modules starting from the primary inputs would fit this formulation very well. It requires all the modules at the input of a module to be in the ordering before itself is added. As the intermediate variable at a node represents the total delay from a primary input it could be interpreted as an arrival time. A signal starts at a primary input with a certain arrival time, for now assumed to be 0. The delay of a module is added to get the delay at its output. This is the time  $AT$  at which moment the signal arrives at this point. If there are multiple paths from the same primary input coming in at the inputs of a module the maximum is chosen which is the latest moment the signal could arrive at this point. This is very similar to an arrival time calculation where the maximum among all inputs is taken to get the arrival time at the current node. A major difference is that we will have separate calculations and thus variables  $AT_{PI_j}$  for each primary input  $PI_j$  at each module.

Assume again the area delay relations to be given as before such that for a given sizing parameter  $s$  of a module  $m$  the delay  $D_m(s)$  and area  $A_m(s)$  are known when  $s$  is given. The the path with highest delay for given size parameters  $s$  can be efficiently calculated using the ideas presented above.

Also the total area can be calculated. The whole procedure to obtain the arrival times  $AT$  at the primary outputs for a given size parameter assignment could be written as:

The whole procedure could be written as:

ALL MAXIMUM PATH DELAYS

```

1: Order <- topological ordering from primary inputs  $PI$ 
   to primary outputs  $PO$ 
2: for n= 0 to  $|PI|$ 
3:   cre-ate  $AT_{PI_n}=0$ 
4:   for module  $m <-$  Order(0) to Order( $|Order|$ )
5:     for n= 0 to  $|PI|$ 
6:        $AT_{PI_n}^m=0$ 
7:       for i=0 to  $|inputs\ of\ M|$ 
8:          $N=input\ i\ of\ m$ 
9:         if  $AT_{PI_n}^N + D_m(s) > AT_{PI_n}^m$ 
10:           $AT_{PI_n}^m = AT_{PI_n}^N + D_m(s_m)$ 
11: for j= 0 to  $|PI|$ 
12:   for k= 0 to  $|PO|$ 
13:     MAXIMUM_PATH_DELAY( $PI_j, PO_k$ )= $AT_{PI_j}^{PO_k}$ 
14: return MAXIMUM_PATH_DELAY

```

This procedure is much more efficient in determining the highest pin-to-pin delays than enumerating all of them. Enumeration has an exponential complexity of  $O(2^M)$  in the number of modules  $M$ . This procedure is visiting each module only once and for each module all inputs once for each primary input variable. At most  $|PI|$  inputs exist and thus the complexity can be set equal to the number of edges in the network times the number of primary inputs or  $O(PI * E)$ . The number of variables is at most  $|PI|$  at every module and thus of  $O(PI * N)$ .

### useful intermediates

Using the dynamic programming procedure the pin-to-pin paths with maximum delay can efficiently be identified for a given set of sizing parameters  $s$ . The optimization can then be done in another way, by formulating the mathematical program as:

$$\min \sum_{i=0}^{\#modules} A(s_i)$$

subject to

$$AT_{PI_j}^{PO_k} \leq T_{req_{jk}} - D_{W_{jk}} \quad \forall j \in PI, k \in PO$$

and module size limiting constraints

$$\begin{aligned} \minSize_i \cdot s_i^{-1} &\leq 1 & i = 0 \rightarrow \#modules \\ 1/\maxSize_i \cdot s_i^1 &\leq 1 & i = 0 \rightarrow \#modules \end{aligned}$$

where  $PI, PO$  are the sets containing all the primary inputs and primary outputs,  $AT$ s are the maximum arrival times at the primary outputs for the given primary inputs with a given required timing  $T_{req}$  between them and a total wire delay  $D_W$  based on the manhattan distance between  $PI_j$  and  $PO_k$ . The dynamic programming approach would have to generate the variables  $AT$  at the primary outputs for the currently assigned values  $s$  during the optimization which represent the paths with highest delay. However as soon as the optimization changes the sizes and thus areas  $A(s)$  of some gates the corresponding delay  $D(s)$  of a gate will change too. As a result the path delays will change and another path might have the highest delay than the one calculated by the dynamic programming procedure. It is therefore very impractical to have those two separate procedures as it would require a lot of iteration between the two. Fortunately that can be avoided!

The essence of the dynamic procedure was choosing a maximum among the inputs of a node and store this value for further use. Further the calculation had to be done from primary inputs to primary outputs as based on the previous calculation method this is the order in which they become available.

**Theorem 3** *The following set of constraints added to the mathematical program*

$$\begin{array}{l} A \leq G \\ B + C \leq G \\ D + E + F \leq G \end{array} \iff G = \text{MAX}\{(A), (B + C), (D + E + F)\} \text{ if } G \text{ is minimum}$$

*will at all times set  $G$  equal or higher to the sum of variables of the particular constraint with the highest total sum and thus determine a maximum over those alternatives.*

So a variable  $G$  is introduced which will represent the maximum delay among all alternative constraints. Suppose now that in the constraints the variables are the sum of the maximum delay variable of the module at a particular input of the current module and the delay of the current module. Then writing such a constraint for each input will exactly represent the choice of a maximum among all inputs of a module as was required. Note however that  $G$  seems to be allowed to also have a much higher value. This is fortunately not true in our context.

**Theorem 4** *If the following set of constraints is added to the mathematical program*

$$\begin{array}{l} A \leq G \\ B + C \leq G \\ D + E + F \leq G \end{array} \quad G \leq F$$

*All constraints will end up having an equal value and will also be equal to  $F$  when  $G$  is limited by a timing constraint  $F$ .*

*Proof:* The mathematical program tries to minimize area by exchanging it for higher delays. Therefore the constraints will all try to get as large as possible and by that pushing  $G$  higher and higher. The maximum delay is however limited by a timing constraint and thus  $G$  is limited from above. As a result  $G$  becomes equal to the limiting timing constraint and the constraints will grow until they are equal to  $G$ .

So due to the optimization and the exchange of area and delay we get the equality sign in the set of constraints representing the maximum operation of dynamic programming. But the timing limit is only set at the last node.

**Theorem 5** *Adding all MAX operation constraints for all modules continually provide the relation of delays of one module to another generating the maximum delay values in such way that an ordering as with dynamic programming is no longer required.*

*Proof:* Look at the following set of constraints representing the paths between input  $A$  and output  $D$ . There is one alternative for the paths to  $B$  and  $C$  and from those two there are two alternatives to  $D$ .

$$\begin{array}{l} 0 \leq A \quad A + D_{m_1} \leq B \quad B + D_{m_3} \leq D \quad D \leq T_{constraint} \\ A + D_{m_2} \leq C \quad C + D_{m_3} \leq D \end{array}$$

As proven before  $D$  will be equal to the constraint  $T_{constraint}$  as it is bounded by that from above and both constraints will like to be equal to that too as  $D_{m_3}$  will try to be as large as possible. As a result  $B$  and  $C$  will like to be to  $D - D_{m_3}$  or less. But the constraints generating  $B$  and  $C$  also have a module delay trying to be as large as possible and by that trying to maximize  $B$  and  $C$ . But now  $B$  and  $C$  are also limited like  $D$  and therefore also there constraints will have an equal sign and try to set  $A$  to  $B - D_{m_1}$  and  $C - D_{m_2}$  respectively. But  $A$  has been bounded from below by 0 in this case. As there are all equal signs in the constraints the variables represent the maximum delay values at every module. They are bounded between 0 and  $T_{constraint}$  and the module delays  $D_m$  will take on maximum values to make the constraints have equal signs at all times. As the variables are continually updated they will represent the dynamic programming solution instantaneously and at all times during the optimization. Therefore ordering is not required as the solver will take into account all constraints at the same time.

So the module delays  $D_m$  are maximized to create all equal signs to make maximum use of the available delay between upper and lower bound. At the same time their corresponding areas are summed and minimized. As a result there will be an to the module delays  $D_m$  such that the constraints are fulfilled and thus the maximum path delays are equal to the timing given constraints and at the same time have a minimal total area.

### **the bottom line**

The number of extra variables introduced is equal to those generated during the dynamic programming procedure. The number of extra constraints is related to the max operation of dynamic programming. The total is equal to the number of max operations performed in the dynamic programming

procedure. Therefore the number of variables is  $O(PI * N)$  and the number of constraints is  $O(PI * E)$ . This has clearly a much lower (even linear) relation to the network size as compared to the enumeration case.

Using the same time variables  $AT$  as during dynamic programming the following complete mathematical programming problem is obtained:

$$\min \sum_{i=0}^{\#modules} A_{m_i}(s_i)$$

subject to

$$\begin{aligned} AT_{n_u}^I + D_{m_i}^{n_u n_v}(s_i) &\leq AT_{n_v}^I && \forall I \in PIC(n_v), e(n_u, n_v) \in E \\ AT_{PI_j}^{PO_k} &\leq T_{req_{jk}} - D_{W_{jk}} && \forall PI_j \in PI, PO_k \in PO \\ d^{n_u n_v}/s_i + p^{n_u n_v} &\leq D_{m_i}^{n_u n_v}(s_i) && \forall e(n_u, n_v) \in E \end{aligned}$$

and additional size limiting constraints

$$\begin{aligned} minSize_i \cdot s_i^{-1} &\leq 1 && i = 0 \rightarrow \#modules \\ 1/maxSize_i \cdot s_i^1 &\leq 1 && i = 0 \rightarrow \#modules \end{aligned}$$

where  $E, PI, PO$  and  $PIC(n_v)$  are the sets containing all the edges of the network graph, primary inputs, primary outputs and the intersection of PI and the cone of the corresponding output or net  $n_v$  of module  $m$ , respectively, and  $e(n_u, n_v)$  is an arc from net  $n_u$  to net  $n_v$  through a module  $m$ . Of course the the size limiting constraints shown earlier can also be added.

The number of variables is now equal to the sum of the number of primary inputs in the input cones at the outputs of all modules and the module delays themselves. A worst case estimate would be  $O(PI * o * M + i * o * M + M)$  assuming all primary inputs  $PI$  are in the input cone of the  $o$  number of outputs of all the  $M$  modules. Then  $i * o * M$  delay variables for all input-output delay relations of each module and another  $M$  for the size variables  $s$  establish the estimate. In case all input-output delays of a module are equal the middle term would be reduced to  $M$  as well.

All of the constraints except those at the primary outputs and the size limits have only two terms: the previous maximum value and the current delay, that is a size dependent delay part and a fixed delay part. The number of



constraints per output of a module is equal to the number of primary inputs in the input cone or nets  $n_v$  of the modules ( $=PIC(n_v)$ ). Therefore the number of constraints is now in worst case of the order  $O(PI * i * o * M + M + 2 * M)$  where there are  $M$  modules with  $o$  outputs which are all connected to all  $i$  inputs of the module each having all  $PI$  primary inputs in there input cones. Then there are  $M$  delay constraints and another  $2 * M$  size limiting constraints.

One more remark about the size limiting constraints is in order. When those are added the module delays become bounded and therefore not all constraints will have an equal sign. In fact some values will be allowed to float between two limits. Although this still would lead to valid timing, it usually is a problem for many solvers. This can be fixed by adding the resulting maximum delay paths variables to the object function with a very small multiplication term. As a result this will push the resulting delay down if possible at virtually no area cost.

The original exponential complexity is now avoided. It is now linear in the number of nodes and edges of the corresponding timing graph like representation. The number of constraints increases linear with the number of edges in spite of the fact that adding a single edge could still generate many additional paths. This can also easily be observed by looking to the situation of figure 3.5. Adding the dotted edge will introduce only one extra variable due to the new PI in the input cone, and only one extra constraint for this input as there is only one primary input in the input cone.

| Circuit | original |          |          | reduced |        |        | runtime |
|---------|----------|----------|----------|---------|--------|--------|---------|
|         | #vars    | #constr  | #terms   | #vars   | #const | #terms |         |
| C17     | 14       | 31       | 56       | 18      | 48     | 84     | 3s      |
| C432    | 301      | 291e3    | 4640e3   | 2838    | 6205   | 12193  | 14s     |
| C499    | 606      | 100e3    | 13200e3  | 5915    | 10869  | 21307  | 28s     |
| C880    | 476      | 9143     | 95252    | 3025    | 5754   | 11102  | 13s     |
| C1355   | 1052     | 4171e3   | 88101e3  | 9622    | 17159  | 33600  | 90s     |
| C1908   | 723      | 196e3    | 3278e3   | 6061    | 12183  | 23861  | 47s     |
| C3540   | 1502     | 22500e3  | 509e6    | 12422   | 25636  | 50215  | 126s    |
| C5315   | 2269     | 395e3    | 7641e3   | 12326   | 22950  | 43914  | 69s     |
| C6288   | 4774     | 53800e15 | 41800e18 | 34710   | 68695  | 134507 | 1493s   |
| C7552   | 3422     | 428e3    | 6630e3   | 23442   | 44205  | 86189  | 188s    |

Table 3.2: Solver parameter comparison for some mcnc benchmarks

Table 3.3 shows that we are now capable of handling the larger circuits in reasonable time. Circuits with up to  $10e^{18}$  paths are within reach. Even C432 would originally have taken more than the 1500 sec now needed for such large circuits.

To illustrate the approach we offer the example in figure 3.7:

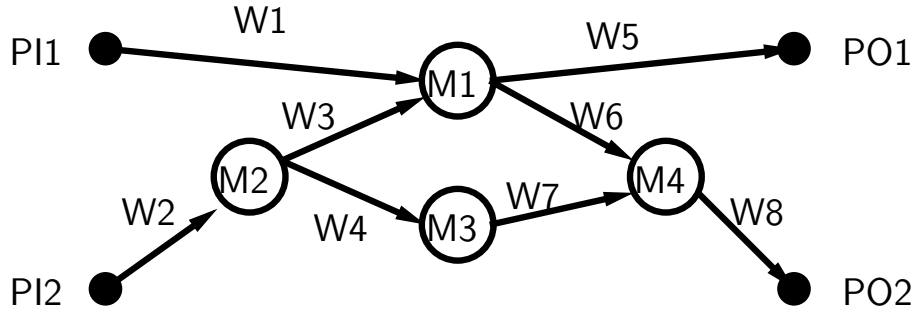


Figure 3.7: A very simple circuit with modules  $M$  and wire segments  $W$  of unknown length

The paths have a total delay of:

$$W1 + M1 + W5 < T_{req_{PI_1 PO_1}}$$

$$W2 + M2 + W3 + M1 + W5 < T_{req_{PI_1 PO_2}}$$

$$W2 + M2 + W4 + M3 + W7 + M4 + W8 < T_{req_{PI_2 PO_2}}$$

$$W2 + M2 + W3 + M1 + W6 + M4 + W8 < T_{req_{PI_2 PO_2}}$$

Using the simplification due to monotonicity this becomes:

$$M1 < T_{req_{PI_1 PO_1}} - D_{W_{PI_1 PO_1}}$$

$$M2 + M1 < T_{req_{PI_2 PO_1}} - D_{W_{PI_2 PO_1}}$$

$$M2 + M3 + M4 < T_{req_{PI_2 PO_2}} - D_{W_{PI_2 PO_2}}$$

$$M2 + M1 + M4 < T_{req_{PI_2 PO_2}} - D_{W_{PI_2 PO_2}}$$

After introducing common terms for maximum values up to a certain primary input yields:

$$0 + M1 < D_{PO_1} \quad D_{PO_1} + M1 < D_{PO_2} \quad D_{PO_2} < T_{PI_1 PO_1} - D_{W_{PI_1 PO_1}}$$

$$0 + M2 < D_{PO_2} \quad D_{PO_2} + M1 < D_{PO_2} \quad D_{PO_2} < T_{PI_2 PO_1} - D_{W_{PI_2 PO_1}}$$

$$\begin{aligned}
D_{PO2_1} + M3 < D_{PO2_3} & \quad D_{PO2_3} + M4 < D_{PO2_4} & \quad D_{PO2_4} < T_{PI2PO2} - D_{W_{PI_2PO_2}} \\
D_{PO2_2} + M4 < D_{PO2_5} & \quad D_{PO2_5} < T_{PI2PO2} - D_{W_{PI_1PO_2}}
\end{aligned}$$

Now result 11 smaller constraints with 16 terms. This may hardly seem an improvement over the initial 4 constraints with 9 terms in total, but this is only a small example and a local result. Moreover the fanouts and fanins have a low maximum of 2 and the paths are short so that there is in fact not much chance on sharing.

### 3.4 further tableau reductions

The huge reductions in problem size by introducing the node-to-node formulation of the previous section brought the complexity of time budgeting down to an acceptable level. Further reduction might be possible for specific solvers. Such is the case with solver as MOSEK[29] whose runtime is quite sensitive to both the total number of terms in the constraints and the total number of variables. Therefore reducing those would directly result in an run time advantage. Two methods from literature are particularly useful in our situation.

#### pruning

Using a pruning strategy like proposed in a timing optimization paper[42] it is possible to reduce in some cases both the number of variables as well as the total number of terms in the constraints at the same time. Note that pruning reduces variables and constraints by reformulating the constraints and not by rejecting variables or constraints of the problem such that the problem which is solved and the solution which is obtained are still exact and the same as for the unmodified case.

The term pruning comes from the network view of the problem. In the original formulation there is a variable at each node and a constraint for each edge relating those variables together with the delay of the node. It is possible to prune a network node and therefore a variable by replacing the now unconnected edges by a new set of edges which connect all the outputs of the nodes at the input of the removed node with the inputs of the nodes at the output of the removed node. Those edges will now be represented by constraints again

relating the variables at the nodes but now contains as an extra term the delay of the pruned node. This will change the total number of constraints but also the number of terms per constraint. This is illustrated in figure 3.8.

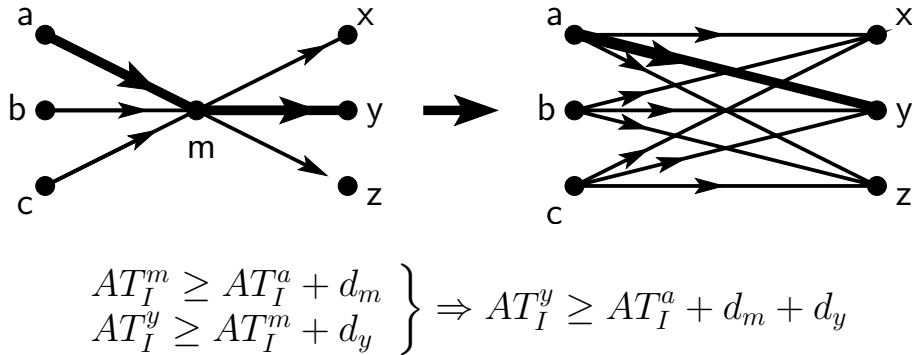
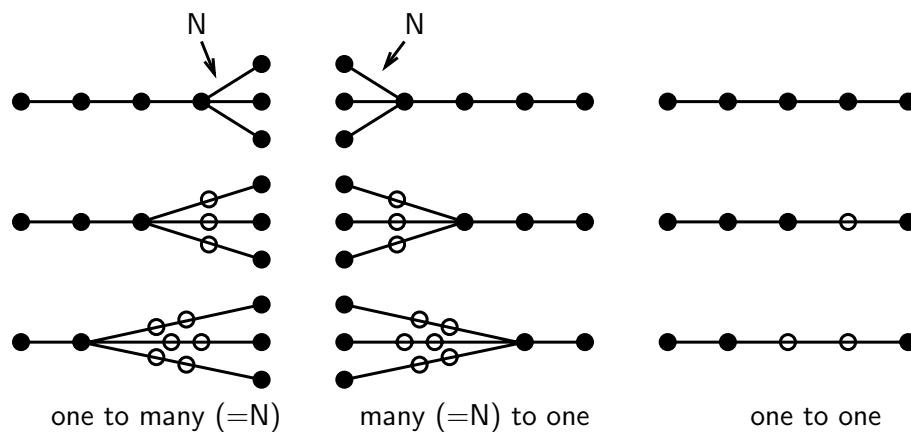


Figure 3.8: *The basic pruning operation with constraints for the bold path shows the elimination of variable  $AT_I^m$  and one constraint.*

Assigning certain weights based on the solvers properties to the removed variable and changed number of constraints as well as number of term in the constraints it is possible to calculate a gain value for different cases of number of inputs and outputs of a node. It turns out that in our case only the single input-single output node as well as the multiple input-single output result always in a gain. The single input-multiple output nodes result in gain for variables and constraints but not in terms and in general does not improve our run time. We get back to this particular instance in the next paragraph. Having more inputs and outputs at the same time results only in gain if it is 2 and 2 and no edge has result from pruning. Therefore this rare case is omitted.

The reduction itself is carried out in a very simple heuristic way. For every possible reduction the network nodes are visited in a topological order from inputs to outputs and pruned if they are of the current type of reduction alternative. first the one input one output nodes are pruned followed by the one input multiple output nodes.

Although the few reduction possibilities might seem very limited they still result in major gains in run time. This was actually also the case for the original paper were not much more cases were an advantage to prune. Another aspect to take into account is that in our case the network has multiple variables representing delay from each primary input in the input cone at each node. For the pruning not the real network but the striped down version for only the variables of a single primary input is of interest. This will have a major



| N      | one to many |    |    | many to one |    |    | one to one |
|--------|-------------|----|----|-------------|----|----|------------|
|        | 2           | 3  | 10 | 2           | 3  | 10 |            |
| vars   | 6           | 7  | 14 | 10          | 15 | 50 | 5          |
| constr | 5           | 6  | 13 | 8           | 12 | 40 | 4          |
| terms  | 10          | 12 | 26 | 16          | 24 | 80 | 8          |
| vars   | 5           | 6  | 13 | 8           | 12 | 40 | 4          |
| constr | 4           | 5  | 12 | 6           | 9  | 30 | 3          |
| terms  | 10          | 13 | 34 | 14          | 21 | 70 | 7          |
| vars   | 4           | 5  | 12 | 6           | 9  | 30 | 3          |
| constr | 3           | 4  | 11 | 4           | 6  | 20 | 2          |
| terms  | 9           | 14 | 48 | 12          | 18 | 60 | 6          |

Figure 3.9: The last two pruning operations always result in a gain. The first one results sometimes in a gain as the increases of the number of terms in the constraints have a counter effect on the other reductions.

impact on the fanout and fanin count of the nodes in the remaining network and result in many more attractive nodes to prune than the original network might suggest.

This reduction method can reduce run time in certain cases with high amount of reduction by more than 50% as can be seen comparing the first two runtime columns in table 3.4. Not all cases improve and even slight increases are noted. But in general those are small circuits where the exact gain is not really clear. The problem here is that the change in run time is a little different for small examples than for larger ones. In general one can conclude that the pruning never results in a loss of run time but can lead to a major improvement.

Note that if the reduction would be applied until all internal nodes were gone the original formulation enumerating all paths would result. There is clearly an optimum between the two formulations of enumerating all paths and using intermediate variables at all nodes.

### **forward versus backward formulation**

The sensitivity of the solver to the number of terms and variables can also be exploited in another way. The previous described method writes down the delay constraints as functions related to the primary inputs in the input cone of each module. But the formulation of the constraints can also be done in reverse order from primary outputs to the primary inputs. The resulting budgeting is exactly the same but the number of variables and constraints can be quite different. In general there will be a majority of modules with much more primary inputs in its input cone than primary outputs in the output cone or the reverse which can be exploited.

In this reverse case the number of delay constraints is related to the number of primary outputs in the output cone of each module. In case the circuit has many inputs and few outputs the structure of the circuit will be more likely to be trees rooted at the primary outputs. Therefore the the number of primary outputs in the output cone of a node will be only few or just one. Then the backwards formulation will have far less variables at the nodes and terms in the constraints as well as number of constraints. If there is only one primary output then there is only one variable at every node and only one constraint for every edge. While at the same time multiple inputs would result in multiple variables and constraints per edge in a forward oriented formulation of the same circuit.

Also pruning can still be applied. Note however that the most effective nodes to prune having multiple inputs and a single output mostly disappear using the reversed orientation of a network with few outputs. It turns out that the change from a forward to backward formulation in general results in a better gain for the same multiple input-single output node than pruning the same node in the original formulation direction. The first pruning type of figure 3.9 can be regarded as the backwards formulation of the second prototype which result in much lower numbers but the effectiveness of pruning is reduced. The same is true starting from a backwards formulation for single

input-multiple output nodes as reversing the direction does turn them into multiple input-single output nodes for the backwards formulation. Still pruning is advantageous afterward and adds still to the gain observed from the reverse formulation as there are still also the one-to-one nodes and not all nodes will have optimal direction.

This is also supported by the data in table 3.4. In the cases that pruning did not gain much in the forward direction much was gained in changing the direction of the formulation. The networks have a construction such that most nodes does not gain much with pruning and thus are likely of the multiple input-single output type of nodes. But it is exactly these type of nodes that gain a lot by reversion the formulation.

| Chip  | #nodes | #paths | run time |             |             |
|-------|--------|--------|----------|-------------|-------------|
|       |        |        | fw       | fw pruned   | bw pruned   |
| C432  | 147    | 291e3  | 14s      | 17s         | <b>5.3s</b> |
| C499  | 287    | 100e3  | 28s      | <b>26s</b>  | 30s         |
| C880  | 225    | 8442   | 13s      | 19s         | <b>5.2s</b> |
| C1355 | 510    | 417e4  | 90s      | <b>44s</b>  | 74s         |
| C1908 | 349    | 196e3  | 47s      | 45s         | <b>36s</b>  |
| C3540 | 740    | 225e5  | 126s     | 114s        | <b>76s</b>  |
| C5315 | 1081   | 395e3  | 69s      | <b>66s</b>  | 76s         |
| C7552 | 1682   | 428e3  | 188s     | 130s        | <b>69s</b>  |
| C6288 | 2371   | 538e17 | 1493s    | <b>528s</b> | 1414s       |

Table 3.3: *Results for different formulations including reduction.*

A quick network traverse forward and backwards including pruning will show which of the two is most effective to use. This fast and cheap preprocessing will yield high gains in run time in most cases and therefore should always be done.

The effects are much less noticeable when there are many input and outputs or only a few of them. On one hand something is saved but at the same time for some other part of the network extra costs are introduced, which is just the reverse when the traversal direction is reversed. In the second case not much is gained first of all as there are only few variables at maximum at a node for each direction.

A mixed direction formulation choosing the best direction which might be different for parts of the network seems to be an even better solution. Problem however is the connection of the variables of the separate parts. The required number of constraints is a multiplication of these numbers of variables for each related connection point. This clearly grows exponential and very soon outweighs the advantage of the best directed formulations.

### 3.5 enhancing robustness

The fully optimized solution has some problems as it hides the real problematic delay paths. Mathematical solvers fully optimize all variables so tight that if afterward a little change occur the total solution can be heavily affected. This is mainly caused by a change in a path which is hard to optimize. But as not every module and path is as critical the change could have been made somewhere in the non-critical areas too with little effect on the total solution. The main problem is however that this non-criticality has been optimized away from observance.

This problem has been dealt with in a timing optimization paper[4] by introducing separation term in the cost function. It tries to create some separation between the resulting delay and the required delay if possible at low cost. Therefore non-critical paths will have slack introduced at them. The more slack assigned the less critical the path is and changes will cost relatively little if needed. Were earlier all outputs equally critical with slack 0 resulting in a “wall” in an ordered slack plot, now there will only be a few with slack 0 which are the real hard to optimize paths (figure 3.10).

The same idea can be easily added to the previous presented geometric program. The only change required from the presented separation is that it should be turned into a term which can be added to a posynomial objective. A separation cost term  $P(T_{req}, AT) = k * \left(\frac{AT}{T_{req}}\right)^q$ . which tries to enlarge the difference between the arrival time( $AT$ ) and the required time( $T_{req}$ ) at a controllable cost is added to the cost function.  $q$  determines how fast the penalty decays with separation, and hence how much separation is “enough” and  $k$  controls the importance of slack as compared to area.



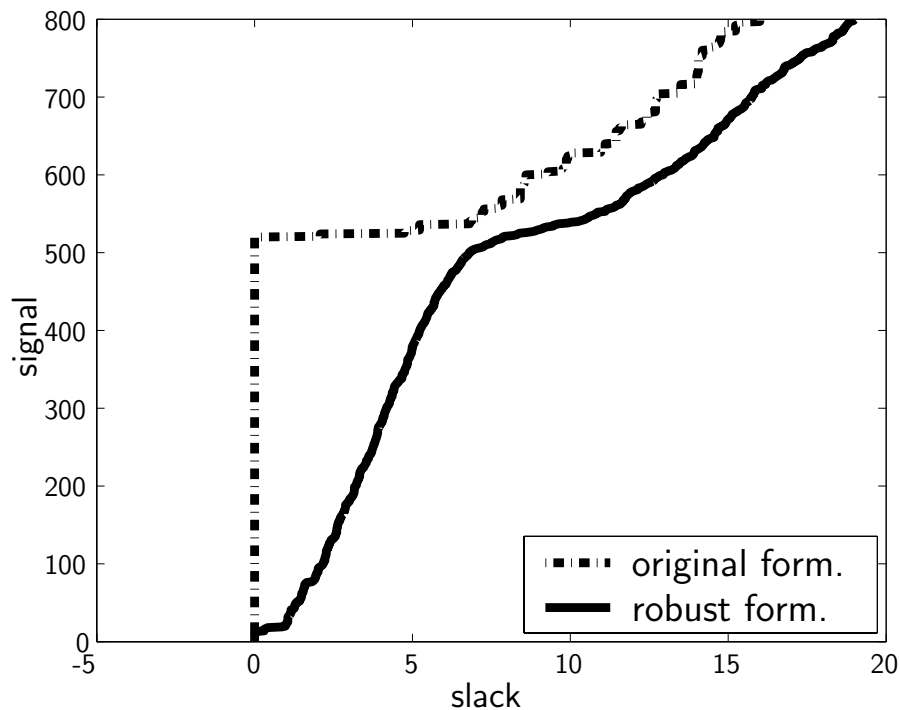


Figure 3.10: Only one to three percent area increase reduces the number of critical paths dramatically

It turns out that this separation term is closely related to the presented term  $P(T_{req}, AT) = k * e^{\frac{-(AT - T_{req})}{\sigma}}$  in the paper with  $k$  is equal and  $\sigma * q = T_{req}$ . This allows direct translation of the effects of the parameters as presented there to our case.

Experiments showed that at area costs of about 3% only a very few paths are really critical. When requiring at least 5% of the required arrival time as slack before denoting a signal non-critical about 20 to 30% of the signals become non-critical. When this requirement is lowered to 3% of the required time this increases quickly up to 70 to 80%. Figure 3.10 shows such an example using circuit C499 with  $k = 80$  and  $q = 40$ .

It requires some tuning of  $k$  and  $q$  to get the desired amount of separation for a certain number of paths at a particular area cost.  $k$  determines mostly the amount of extra area allowed while  $q$  determines the amount of separation and thus the number of non-critical paths and their amount of slack.



# Chapter 4

## constant delay mapping

Wire planning requires a concept of constant delay synthesis. It assigns delay values to modules based on the best known information at that time and then keep the chosen delays fixed at further refinement of the design. This means that an implementation of the logic function of the module must be provided at a required delay at minimum area cost which can be kept at that delay although some parameters influencing it can change.

Conversion of the boolean network into an implementation in only known library cells which can be laid out on a die is done by logic synthesis. This is a refinement step of the design as the general boolean representation is replaced by a more detailed and specific implementation. Thus among a range of possibilities the right one has to be chosen to continue the refinements from there and limit the range of possible final solutions reachable from there at the bottom of the design space triangle (figure 1.1).

Traditionally this conversion is a two phases approach. Based on some rules of thumb without real notion about area or delay the boolean representation is technology independent optimized to a specific boolean network. Then the step of technology mapping [24], also called library binding [10], does based on this biased starting point the technology dependent optimization to an implementation of library cells. Seen in the context of refinement the first optimization chooses already an intermediate refinement point and by that limits the range of reachable solutions in the second step. The complexity of the total task is reduced by first choosing some structure for the boolean network, before also changing it into library cells with delays and areas. But it is also clear from

this view that rules of thumb can put us in the wrong corner before starting the last conversion to library cells (figure 4.1 and 4.2).

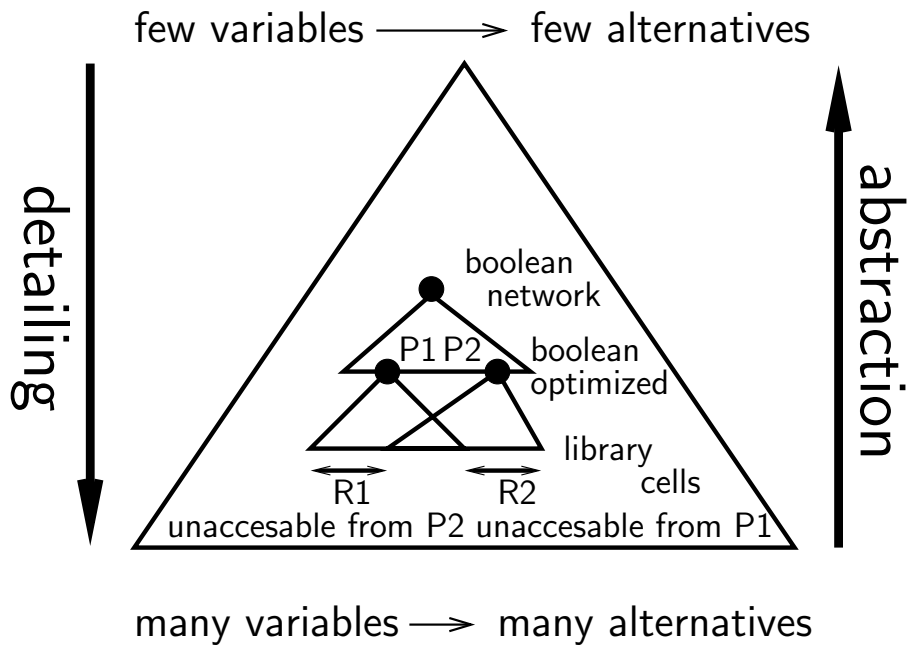


Figure 4.1: *Going from one level of detail into another in two steps can lock out solutions in the intermediate step*

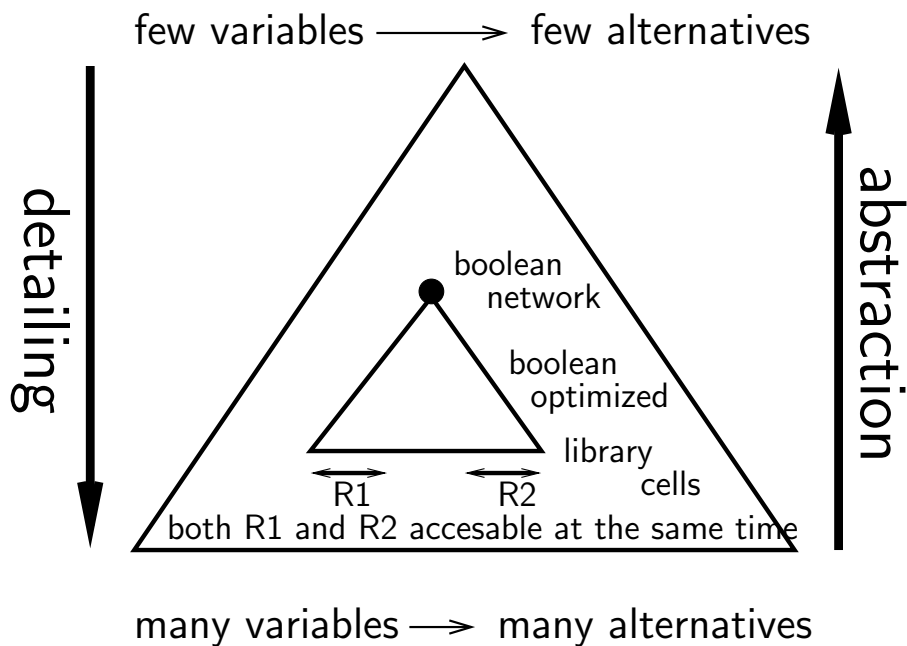


Figure 4.2: *Going from one level of detail into another directly offers a larger search space to be explored at the cost of higher complexity of optimization*

In the context of wire planning the result of technology mapping or actual of the whole synthesis step should be a description of an interconnection of library gates meeting the required delay at a specified load at minimal costs like area and or power. Traditionally technology mapping is producing as fast as possible or small as possible implementations. This has also its impact on the internal working of technology mapping and the delay representations used for the library elements. By merging partly the steps used in logic synthesis a larger search space can be explored at once and by using a different approach for area and delay modeling and evaluation choosing the exact required delay at least cost is enabled.

The library elements used to implement the logic functions are commonly described by a fixed area and a delay which depends on the load. As a result of that the delay of a module will change as soon as something changes at its interface like more wire load or a heavy loading module. Therefore also a different delay modeling is required in the context of wire planning where delay should be fixed while other parameters can still change. This is done using a new delay paradigm where the gate is modeled with a constant delay and which area is adjusted to keep it to that constant while things like load change [20][16]. This delay modeling was already proposed in section 2.2.

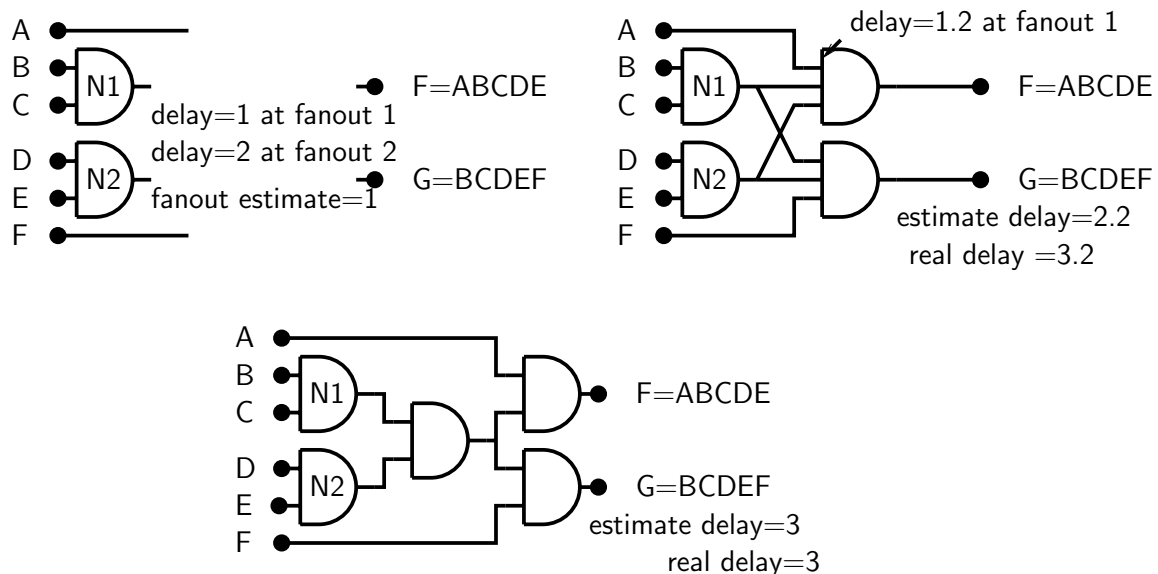


Figure 4.3: The optimal subnetworks based on default fanout of 1 might lead to wrong estimations of the final solutions as shown for two alternatives

This is also an attractive delay model to use during technology mapping itself, as it is based on choosing the best area delay sub-implementation before

the rest of the circuit has been fixed and thus while load is still unknown. A common approach uses load or fanout estimated while determining the best sub-implementation but if it turns out to be off and much higher in the final solution then the calculated estimated delay of the solution will be off too and the real delay will be much higher (fig 4.3). The constant delay model allows to adjust the areas while selecting the final implementation which was predicted to meet a certain delay such that errors does not result in a changing delay value. This assures that an given delay by wire planning can be chosen from the search space and will always be able to meet it for sure.

In the remainder of this chapter the focus is on technology mapping and how it can absorb transformations of technology independent optimization and how the ability of choosing the required delay at least costs can be achieved.

## 4.1 technology mapping

Technology mapping converts a given boolean network into a logically equivalent one consisting of library gates. To ease this process both the library gates and boolean network nodes are expanded into a representation using *base functions*. To do this these base functions should form a logically complete set, that is any function can be realized using these base functions.

Common implementations of mapping procedures use *two-input-nands* and although this in itself is a complete set also often *inverters*. Attractive is that both can be implemented in a single gate in common technologies while this would not be true for two-input-and gates and inverters. Including inverters enables essentially better networks allowing the inverter function itself as well as computational simpler procedures and libraries. In a cost optimizing context it even pays to represent every connection by a series of inverters and add in the library a similar series with cost zero. Richer sets of base functions do not improve the potential of optimization but might complicate procedures[33].

When both the boolean network as well as the library gates are represented by base functions a partitioning of the network can be created with library gates as the partitions. Having the base functions also included in the library guarantees that at least one solution exists. Mapping is first expanding the nodes of the boolean network into sets of nodes only containing basic functions and then do a contraction of sets of nodes into library gates. The expansion

will preserve the original network structure, and as the contraction is based on relative small library gates also this will have only a local effect. The real problem is to find among all possible mappings the one with overall minimum area cost or lowest delay.

This problem of covering an acyclic graph cost optimal is known to be  $\mathcal{NP}$ -hard [21] even if the fanout of all gates is at most 2. Only in the case the fanout is 1, efficient algorithms are known [35][2][24]. This class of networks, also called *convergent networks*, have one primary output and each input terminal of every node has one and only one directed path to that primary output. These are all trees with the exception that nodes are allowed to share primary inputs. They are sometimes called leaf-dags in design automation. In an optimal mapping of such convergent networks every node resulting from the contraction represents a subnetwork which is in the same sense optimal as the subnetwork containing all directed paths from the primary inputs to this node must be optimally mapped as well. This property of optimal substructures in technology mapping for convergent networks therefore calls naturally for dynamic programming.

Cost-optimal technology mapping using dynamic programming is in general a four phased procedure:

1. *expansion* into a network of base functions.  
As multiply logical equivalent representations exist every boolean node is replaced by a balanced network of base functions to disturb the synthesis result the least. Note that optimality of the following steps can only be claimed with respect to this initial expansion. This is closely related to the problems of the limitations imposed on the possible search space due to the intermediate choice of an technology independent optimized network which also leads to suboptimality (figures 4.1 and 4.2).
2. *matching*: determine for each node all matches that can cover a subgraph of the network with this node as its output.  
A string matching algorithm due to Aho and Corasick [1] could be used as proposed in [24] resulting in a matching time proportional to the longest string. But using only two base functions which are topologically distinct allows the application of less sophisticated matching [18]. Convergent networks that are not trees are also allowed in the library, and only have to be tried when primary inputs are involved which can have a fanout larger than 1.

3. *dynamic programming* in topological order

The cost is set to 0 at all primary inputs. For each node the cost of a match is equal to the sum of the costs at the inputs of the match added to the cost of the current match. The match with lowest cost is stored at the node together with the cost value.

4. *covering* by contracting the nodes of the saved match at each surviving node.

Starting from the primary output chose the previous stored best match and proceed at the inputs of this match and again take the best matches there and continue until reaching the primary inputs.

The chosen matches are the library elements that realize the optimal network equivalent with the original network. This is the basic procedure used for a long time even when the objective changed to speed optimal mapping although there are some noteworthy different properties.

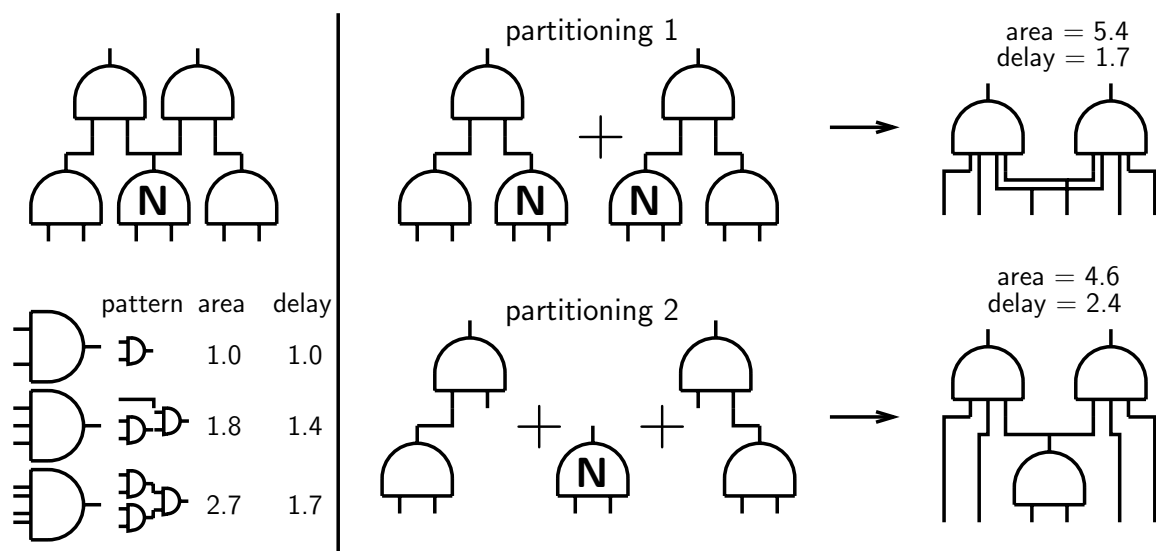


Figure 4.4: *The partitioning in non-convergent networks influences the solutions to be found. In the first case an implicit duplication of node N occurs resulting in more area use but on the other hand a better speed optimal total solution.*

A second important observation is that circuits are in general non-convergent requiring the original circuit to be partitioned in convergent parts as shown in figure 4.4. An easy way is to make every node with fanout more than 1 an primary output of a subcircuit. The positive effect is that at least node duplication is avoided which could otherwise result in higher areas. But the negative



side effect is that the combined optimal mappings might not be the optimal mapping for the total circuit. This is another example where an intermediate decision influences the search space of the refinement into an implementation of library cells.

**change to speed optimal covering**

A remarkable change in technology mapping came up when it was noted that the properties of speed are different from area. It allows to deal with the problem of sub optimality due to the partitioning of non-convergent networks in an elegant way as it is no longer required. It turned out that omission of the requirement of convergence allowed even more extensive changes as also the optimality dependence on the initial expansion could be relieved.

The previous presented technology mapping procedure can also be applied for speed optimal mapping if the delays are load independent as also there the same principal of optimal substructures will apply. Note however that not every convergent subnetwork with only primary inputs *needs* to be optimal to achieve an complete optimal network. The alternative solution in figure 4.5 for the bottom part which is not the speed optimal subcircuit solution will not change the delay of the final solution but costs considerable less area. However essential is that there exist optimal solutions in which these subnetworks are also optimal hence dynamic programming still applies.

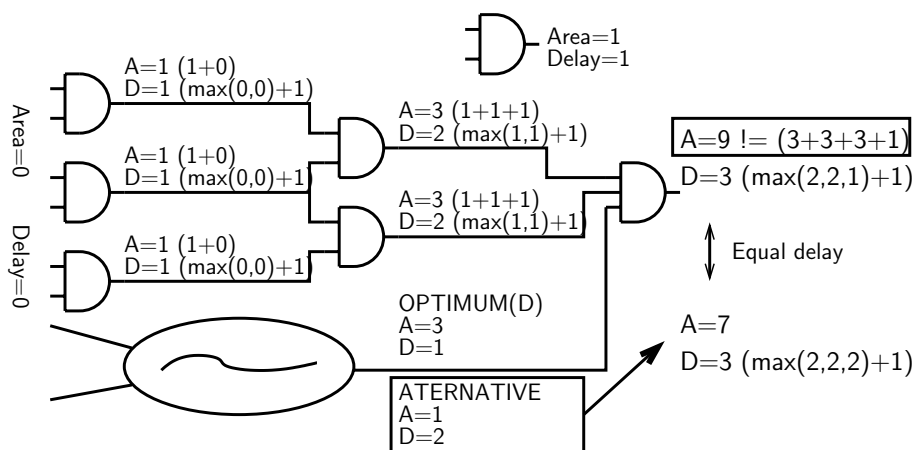


Figure 4.5: Selection of the alternative with less area which is just meeting the required delay for a non-critical subpath results in equal delay but less area. It also illustrates the difference for area and delay calculations with respect to reconvergence in the absence of partitioning

Delay is calculated using the maximum over all inputs. At the point of reconvergence the maximum over the input delays will only depend on the delay value propagated through one of the possible fanouts. Therefore under the assumption of load independent delay fanout is not a problem any more as delay is not an extensive operator like area. In figure 4.5 this effect can be observed for delay and area where the latter can not be calculated correctly without partitioning first. Thus the delay for each node can be directly found in a acyclic network and the partitioning and the possible non-optimality due to that can be avoided. This changes technology mapping from a partitioning into library pattern to a direct covering of a possible acyclic network[25]. It introduces however a duplication problem again as shown in figure 4.6 likewise the case of figure 4.4 and causes an high increase in area consumption.

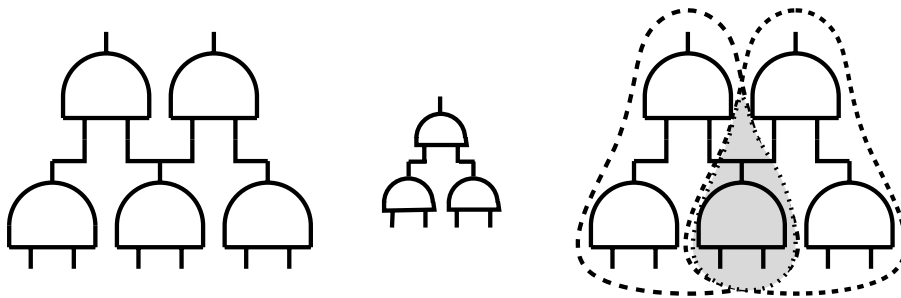


Figure 4.6: Covering the network on the left with two patterns of the type in the middle causes one base function to be covered twice.

Another improvement possible due to covering is the use of choice nodes[26] to encode the multiple logical equivalent representations of the non-unique conversion into base functions in one and the same network. This highly improves the quality of the final result as possible wrong decisions taken too early are avoided. The intermediate step in the refinement of a boolean network to a mapping which would limit the search space on an arbitrary way has been removed. Therefore choice nodes increases the potential for finding optimal mappings enormously.

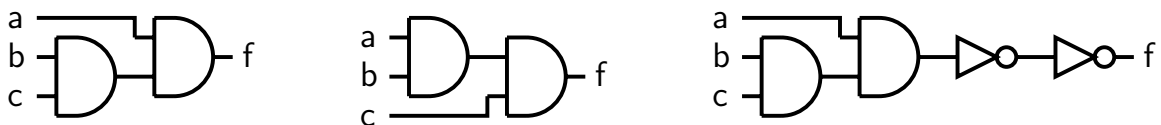


Figure 4.7: Three logical equivalent representations of  $f=abc$

Figure 4.7 shows three networks realizing  $f=abc$ . Choosing only one of them as the input to the mapping procedure already bars several possibly better solutions from popping up in the final solution at all. As this happens at almost every node it is clear that in fact only a small portion of the total available solution space is explored. Recall also the remark made earlier that this is similar to the effect of choosing a particular technology independent optimized circuit in the context of refinement which also limits the reachable search space on forehand.

The authors of [26] even show that the whole space reachable through the algebraic decompositions and factorizations of [5] can be completely contained in a single representation. This removes the limiting factor of a technology independent optimized boolean network choice enabling an even more natural refinement approach. The again enlarged search space can be explored more correctly leading to better overall results. A further reduction allowed is to make sure that no two choice nodes have logically equivalent outputs as in that case they can be merged into one and no base function node should have equivalent inputs as in that case they are equal and can be identified.

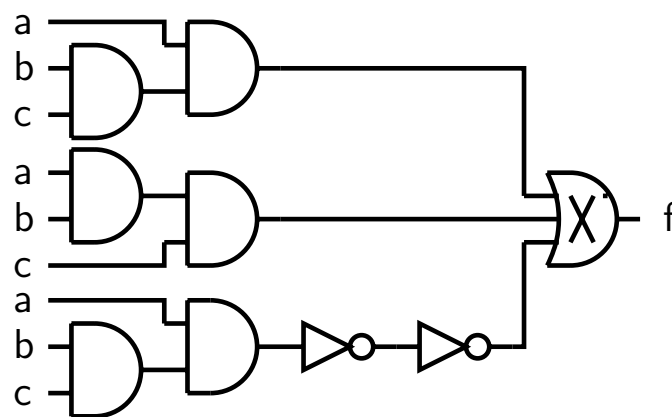


Figure 4.8: *The node marked X is the choice node. Exactly one of the representations at its inputs has to be chosen when covering with a particular library pattern*

A choice node can best be viewed as a selection switch allowing the selection of alternative network parts (fig. 4.8). During covering with library patterns always only one of the inputs of the choice node is active. But all the choices could be tried one after the other.

Applying full reduction starting with a network with the double inverters and a choice node between every other node results in a network with many 4 node cycles of alternating choice nodes and inverters. The configuration of this cycle together with the and nodes whose outputs are connected to one of the two choice nodes is called a *ugate* in [26] and serves as the basic data structure to represent a network with choice nodes. All possible decompositions of  $f=abc$ , thus including those of figure 4.7, can be encoded using 4 ugate and is shown in figure 4.9.

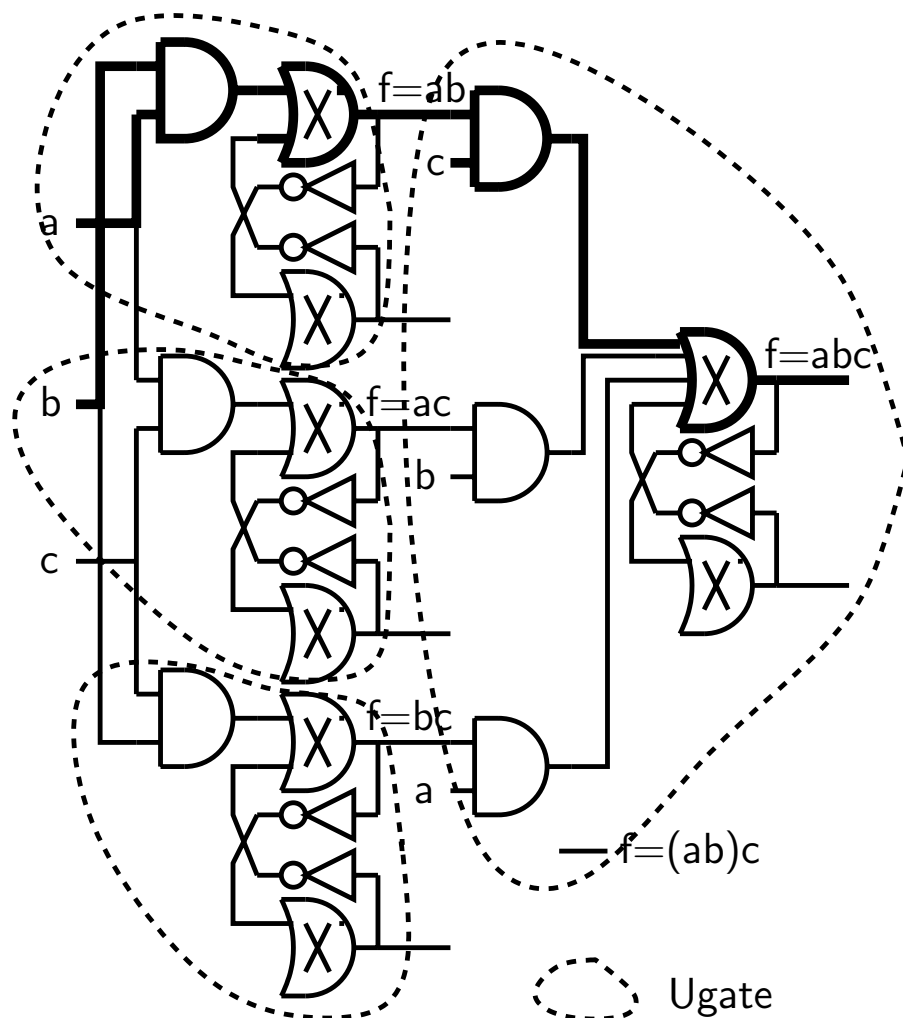


Figure 4.9: A full representation of  $f=abc$  using ugate. One particular decomposition is highlighted

The ugate compactly and elegant captures the multitude of representations enables by choice nodes. It also allows the addition of an arbitrary number of inverters. Besides that the 4 node cycle structure provides always both phases of a function variable allowing additional reductions.

Although these discoveries allow for a larger solution space to be searched and therefore the discovery of better options they also introduce some problems. As already noted the change to covering is likely to cause higher area consumption as duplication will occur (fig 4.6). Another thing pointed out was the over consumption of area due to the fact that speed optimal covering does not always require subnetworks to be optimal, which usually means more area consumption too, to be optimal itself (fig 4.5). There also arise issues about complexity and runtime caused by the fact that a much larger search space has to be explored by evaluating all alternatives at all the choice nodes.

### area and search space control

The observation that for speed optimal technology mapping almost the same algorithms as used for convergent networks can be applied efficiently on non convergent networks directly where the search space does not have to be constrained to only one possible representation in base functions, increases the potential of technology mapping enormously. Note however that the mapping process is no longer a partitioning process but a more general covering on a larger search space. This causes considerable problems in terms of computational run time and memory usage but also in aspects of network quality. They can be traced to two consequences of the innovations:

- *enlarged search space.* The enlarged search space means that many more possible library matches have to be evaluated to pick the best one. During the covering with library gates lots of alternative choices at a gates have to be tried. If a library pattern covers multiple choice nodes the number of alternatives is the product of the alternatives at each choice node. The number of gates and number of choices at those gates of a fully reduced decomposition of a single multiple input and node depends highly on the number of inputs. A six input *and* node result in 57 gates, eight in 247 gates and 10 in 1023 gates. In the last case choice nodes with 511 alternatives exist. In the case of eight inputs it is 127 and in the case of six it is just 31.
- *unbridled area claims.* Speed optimal covering of networks leads often to high and unnecessary area consumption because traditionally all subnetworks are also optimized although this is not always needed for the overall optimum. On non critical paths solutions which are just fast

enough would cost less area while the resulting optimum speed is not affected. Due to covering faster library matches can be found but are likely to duplicate basic function nodes very often resulting in higher area consumption. This problem is even aggravated when choice nodes are allowing even more and thus faster matches to be found. Consequently the resulting networks are often much larger than equally fast and functionally equal alternatives as both [25] [26] report.

Data structures are needed to enable trade-offs between area and speed. Then speed can be controlled and optimized without claiming to unnecessary chip area. Also the enlarged search space due to choice nodes has to be limited to avoid an unmanageable exploded network size and it has to be explored in an efficient way. Both are problematic. No way to constrain it is known to limit the search space without barring possible optimal solutions, so the guarantee of an optimal result has to be given up. Also constructing the required area delay data structures is hampered by the fact that the relation between them is ambiguous.

In the next sections methods are presented to allow a practical implementation which deals with those problems which avoids the a priori partitioning in convergent networks and bias of arbitrary representations and reduces unnecessary area claims.

## 4.2 area control

To control the area consumption data structures to trade off area and delay are needed. Measures have to be taken such that only the needed and required data is calculated and stored to render this solution feasible in large networks. As already pointed out area is problematic to be calculated in a network covering context with fanout points that reconverge later and as multiple covering can occur. A heuristic to deal with this problem is presented. Many possible fanout points are created by the choice node approach, but it should also be noted that in most cases there is only a decomposition choice taken which does not generate fanout points. This should be accounted for by the heuristic. As an heuristic is used based also on structural information of the technology independent optimized network the optimality guarantee has to be given up. Still the a priori partitioning in convergent networks and the bias due to arbitrary representations in base functions is dealt with very well. The heuristics

introduced will in general favor the nodes with multiple fanout, especially for the non-critical paths, because they will be more cost effective. At the same time they also result in more accurate estimates of the heuristics and as a result possible errors will not propagate for from such nodes into the network. A speed optimal mapping can still be guaranteed, while avoiding unnecessary huge area demands when covering directly an acyclic network.

### area delay trade off

In the discussion about speed optimal covering it was pointed out that for an optimal solution not all subcircuits need to be optimal in the same sense. Some subcircuits can become slower while the optimum delay is not changed, but less area is consumed (fig 4.5). The problem however is that until the final cover is chosen the possible non criticality is not known. Therefore multiple distinct area-delay implementations should be stored instead of a single best match during covering search such that area-delay trade off is possible during covering choice. Note that this is a similar problem to the requirement of delay budgeting in a wire planning context to chose a specific delay at the least area cost. Therefore while solving this problem for optimal speed also the ability for choosing the best delay at least area is solved.

To enable these trade-offs between area and speed both should be stored for the matches at each node. It is not hard to understand that only Pareto points are of interest to obtain all possible optimal solutions and thus all others need not to be saved. A point  $(t_1, a_1)$  is a Pareto point if there is no feasible pair  $(t_2, a_2)$  such that:  $(t_2 \leq t_1 \wedge a_2 < a_1) \vee (t_2 < t_1 \wedge a_2 \leq a_1)$ . The curve spanned by the Pareto points is the area-delay trade-off curve for that node. They are created by calculating the area-delay output values of all combinations of all points at all inputs of a possible match. Only those which are Pareto point need to be saved. When adding such a point it could at the same time make other already existing points lose their Pareto status and thus those need to be removed in that case.

Straight forward combination of all  $n_i$  points of all inputs  $i$  costs  $N = n_1 * n_2 * \dots * n_i$  evaluations, and thus increases run time a lot. At the same time at most  $n_1 + n_2 + \dots + n_i$  new points will result. Using the ordering at the inputs the total number of evaluations can be in the same order. The fastest point of the slowest input renders the even faster solutions at other inputs useless as the delay is determined by the slowest input and those other

points will only lead to more area use and thus not to a Pareto point. A change in delay can now only occur due to a slower solution of the currently slowest input and thus this point is changed. Now one of the other inputs could be the slowest and the solutions of all the other inputs need not be faster than that delay. Only if all Pareto points would be spaced in such a way that every time just one input changes and none of the others become uselessly fast the maximum number of  $n_1 + n_2 + \dots + n_i$  evaluations is required but in common situations this will be less.

Although the number of evaluations are limited to the sum of the alternatives of the inputs of a match there still is the problem of rapid increase of Pareto point when moving toward the output node of a circuit. Both, memory and time complexity, are considerably reduced by limiting the total number of Pareto points in a trade-off curve at each node while not affecting results significantly. For all trade-offs at least the fastest and the slowest points are stored. Other nodes are only added if they make a significant difference compared to already stored nodes. This significance could be a fixed area or delay difference, but this could lead to unpredictable worst case number of point at certain nodes which would have a large runtime impact. Therefore the total number of points per trade-off is limited to a certain number using an adaptive difference to separate points apart. This number of points per curve can be used as a trade-off parameter between quality of results and runtime. The result of this pruning is that we generate points which are spaced apart in area (fig 4.10). The same can be done based on delay, but in our context trying to reduce area at a given delay the other spacing is preferred.

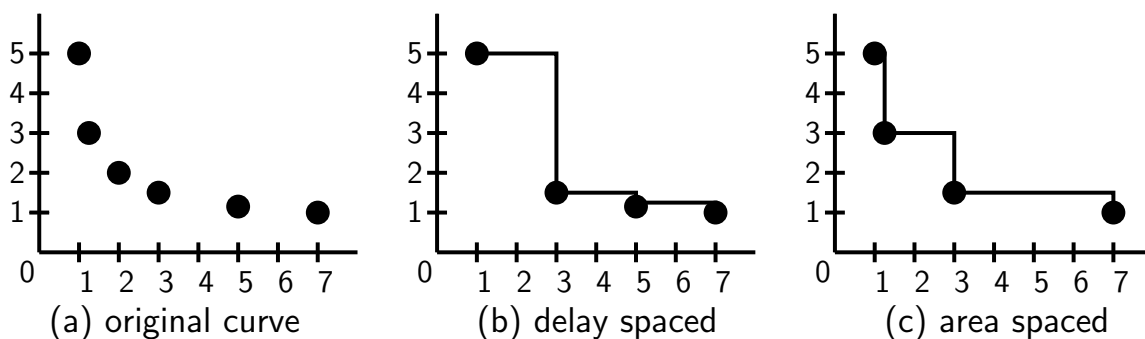


Figure 4.10: Area or delay based pruning of area-delay trade-off curves

The area-delay trade-off curves of all nodes can thus be created in topological order from inputs to outputs. Covering is done in the opposite order: the smallest match which meets the delay constraint is picked from the curve,



and the timing requirements at the inputs are calculated. As a full trade-off is also available at the output it allows to choose the optimal implementation which just meets a required delay: exactly what is needed for delay budgeting in a wire planning context.

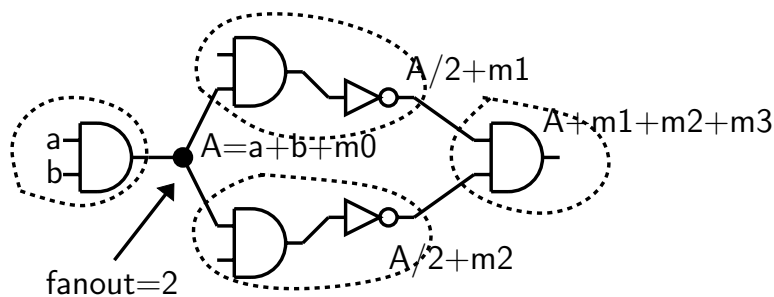
This procedure works fine for load independent delay and convergent networks. Accurate area and delay pairs can be calculated and the result is a mapping with minimal area under given area constraints if all Pareto points would have been kept[6]. Nonconvergence and load dependence complicate this however. The latter is already a problem for any kind of network but is now even more severe as due to covering nonconvergent networks the point of fanout becomes uncertain. Using the proposed constant delay or load independent fixed delay model this is not a dramatic problem any more.

Although the use of a gain based delay allows a full and exact load independent area-delay points calculation as area of input nodes can be related to loading of the current node it has one major drawback. There will be a minimum size which will not be taken into account when applying these kind of calculations which results into the fact that a wrong potential structure was assumed for the first portion of the circuit as the required areas will be much smaller than the minimum sizes. Changing those to minimum sizes result in much higher areas and likely smaller alternatives will exist. Therefore a default load or gain is assumed for the area calculations as also commonly used by others using these delay models[16][37]. Delay will be unaffected by changing loads as it is compensated for, only area estimations might be off in the calculated curves and not exactly the right ones are propagated. All areas will be affected but to almost the same extent although the exact amount might be a little different. Then still the overall spread of area-delay points in a curve is likely to be good and valid and major reduction in area can be achieved.

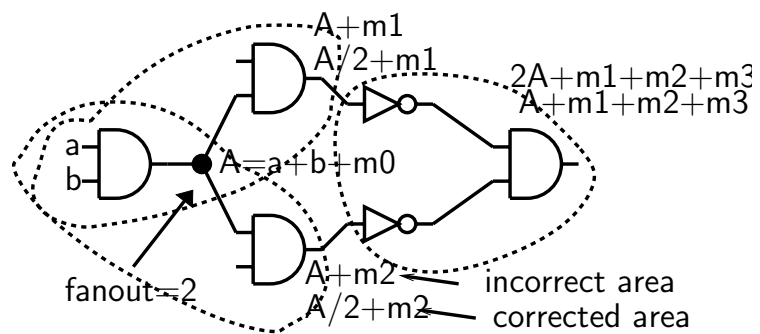
The area estimations are also confounded by the covering of nonconvergent networks. Matches that include multiple fanout points cause duplication of area costs of parts of the network. At reconvergence the total area is estimated too high which results in this point being dropped as being not Pareto.

### crossing numbers

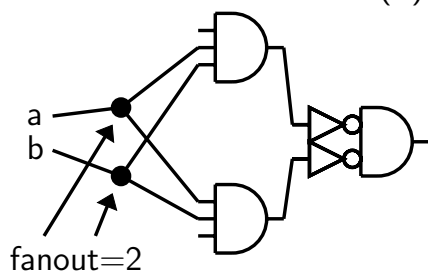
The problem of duplication of area counts in a non-convergent network is a well known problem and was the reason to do partitioning in nonconvergent networks in the first place. Our context of minimum area but speed optimal covering requires area counts. if not accurate then at least usable guesses, to enable trade-offs even for nonconvergent networks. The basic problem is that the area of a subnetwork whose output is used to prepare different inputs for the same gate, is counted to many times.



(a) multiple fanout external to all matches



(b) matches including a multi fanout point



(c) Resulting circuit if matches of (b) are selected

Figure 4.11: *The impact on area calculations of multiple fanout points and covers extending over them.*

A simple and common heuristic is to divide the area at a fanout point by its fanout count such that when reconvergence occurs the areas are added up again and the total area will be correct counting the area at the fanout point only once (fig 4.11a). This will only work for simple cases. In the more complex covering cases such estimates will be wrong as the area at the inputs of a match which extends over multiple fanout points does not get divided for those all fanouts and parts are still counted to often (fig 4.11b). Choice nodes only aggravate the situation by increasing the number of such fanout point and deserve a separate discussion in the next section.

The following heuristic is proposed to accommodate the errors introduced by matches which extend over fanout points and by that hiding them. Assume that all library patterns are trees, which is true with a few leaf dags as exemptions that require a little adaptation and will be discussed at the end. In pattern trees every input has a unique path to the output. When matching such a pattern with a subgraph of the network a one to one correspondence exist between the nodes of those paths and some particular nodes of the network. By adding the fanouts of those nodes for a particular path to one minus their number a *crossing number* is assigned to the input of the path:

$$1 - |P| + \sum_{p \in P} \gamma^+(p)$$

where  $P$  is the set of network nodes on the input-output path of the match and  $\gamma^+(p)$  is the out degree (or fanout) of node  $p$ . Only nodes with fanout unequal to 1 should be considered as the others without multiple fanout do not contribute to the crossing number. This crossing number can be used to represent more faithfully what happens due to fanouts of the nodes in the network on the complete path from input to output in the match. The area is now estimated by first dividing the area at the inputs of the match by the corresponding crossing number before adding them to the area of the library pattern itself.

This effect is illustrated in figure 4.12. When matches completely cover the multiple fanout point still the sum of the area costs of the inputs and the matches correctly add up to the sum of areas at the outputs. If a particular node fans out into other parts then covered by the match then only a part of the area up to there is incorporated in the area at the outputs. The rest is assumed to be taken into account by matches that also tap into this multiple fanout point such that in the total sum of area at the outputs the area at this

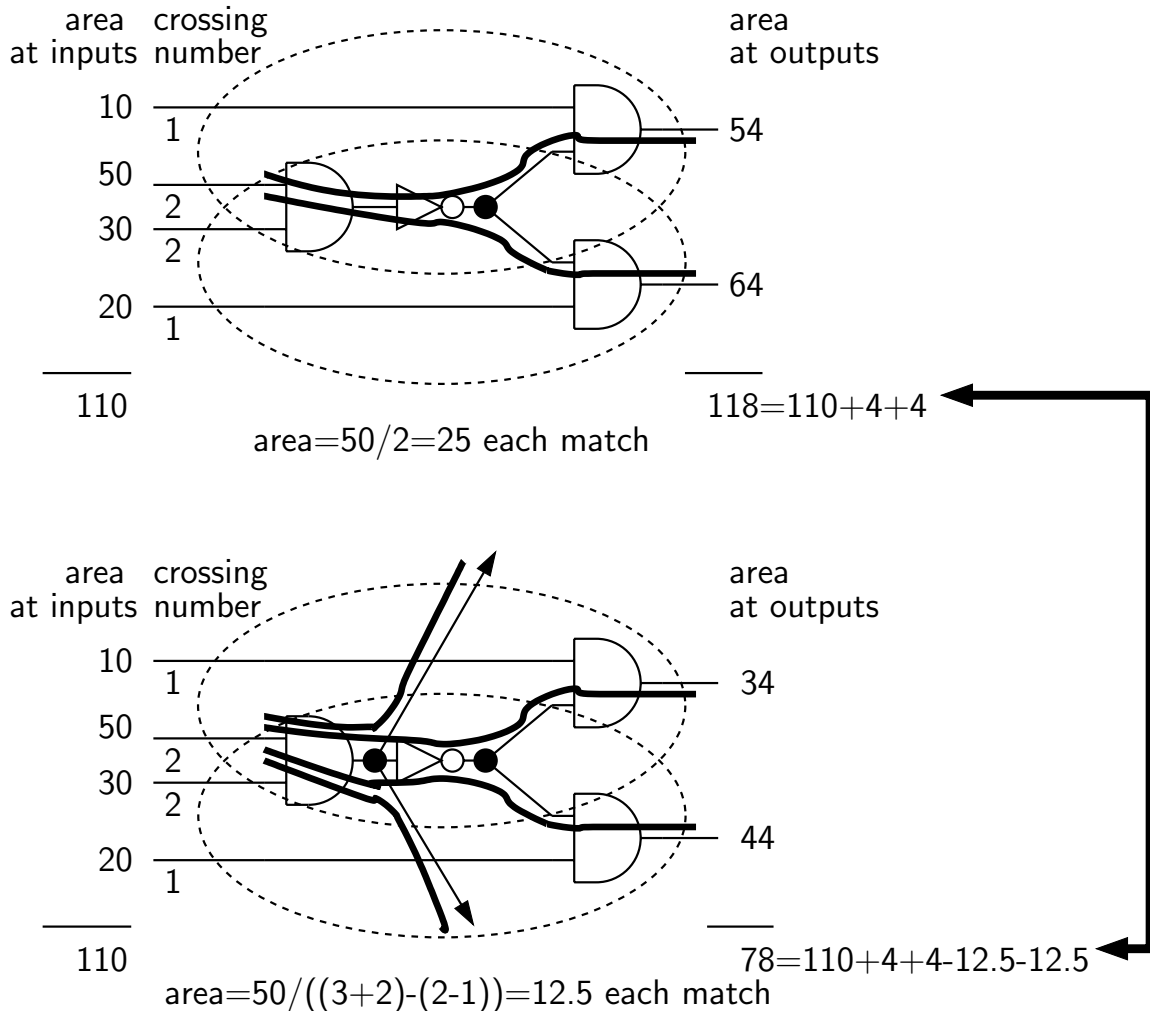


Figure 4.12: Examples of area estimations using crossing numbers. The area of the matches patterns are assumed to be 4.

point is only counted one. So crossing numbers try to avoid to count parts of the network that feed into multiple other parts are counted to many times, and at the same time add up to the correct total when reconvergence occurs.

When inputs of one match are internal nodes of another match (and vice versa) the method becomes less precise but still most area gets divided among multiple matches. Still this error is seldom large as it is compensated by the fact that trade-off curves are used. To obtain the least area use at just the required delay, often duplications of nodes has to be avoided resulting into the fact that multi fanout point are attractive to use. In those cases the area division works fine and only in the few time critical paths fanout node covering will occur and only in this overall limited case errors will occur.

The exception of non tree patterns in the library are an undesired limitation and at least leaf dags should be allowed. The problem is that in that case there are multiple paths to a particular input from the output with possibly different crossing number counts and therefore crossing numbers are ambiguous. In general those library patterns exhibit multiple paths with very similar properties both in length as fanout. To set an unambiguous crossing number at each input the average of all possible crossing numbers corresponding to an input is taken for this input.

### unknown fanout

The crossing number heuristic is based on fanout points of the network to be covered. In a representation with choice nodes however the fanout points are unknown. Take a look at the example of figure 4.13. The configuration on the top is assumed to be a part of a total larger network. The network is matched using a library containing at least patterns for a *two-input and* and a *three-input and* and results in several possible solutions depending on the choices at the choice nodes. Two different choices which produce the upper output are highlighted and they result in a fanout for the gate producing the *and* of *ef* of 1 in the first case and 2 in the second case. So what can be said to calculated crossing numbers in the case of choice nodes.

The ability to increase the potential of technology mapping using choice nodes does not mean that it has to spoil the choices taken during technology independent optimization. Although some optimization power is given up by not extending the network conversion to incorporate logic transformations and thus technology independent transformations it allows us to come up with a way to still apply the proposed heuristic.

A strong correspondence between the network before expansion and after covering is often maintained. The expansion does not introduce any problem and preserves the network structure very well. This is also true for covering using contraction, but even for the more general covering there is no large discrepancies as usually only small patterns are used. The only problematic issue still would be the full reduction, meaning that no two logical equivalent nodes are allowed and therefore is also not applied.

The resulting procedure which keeps the representations similar and therefore provides a way to introduce crossing numbers is the following. The original network is expanded into a network with only two types of nodes: *multiple-*

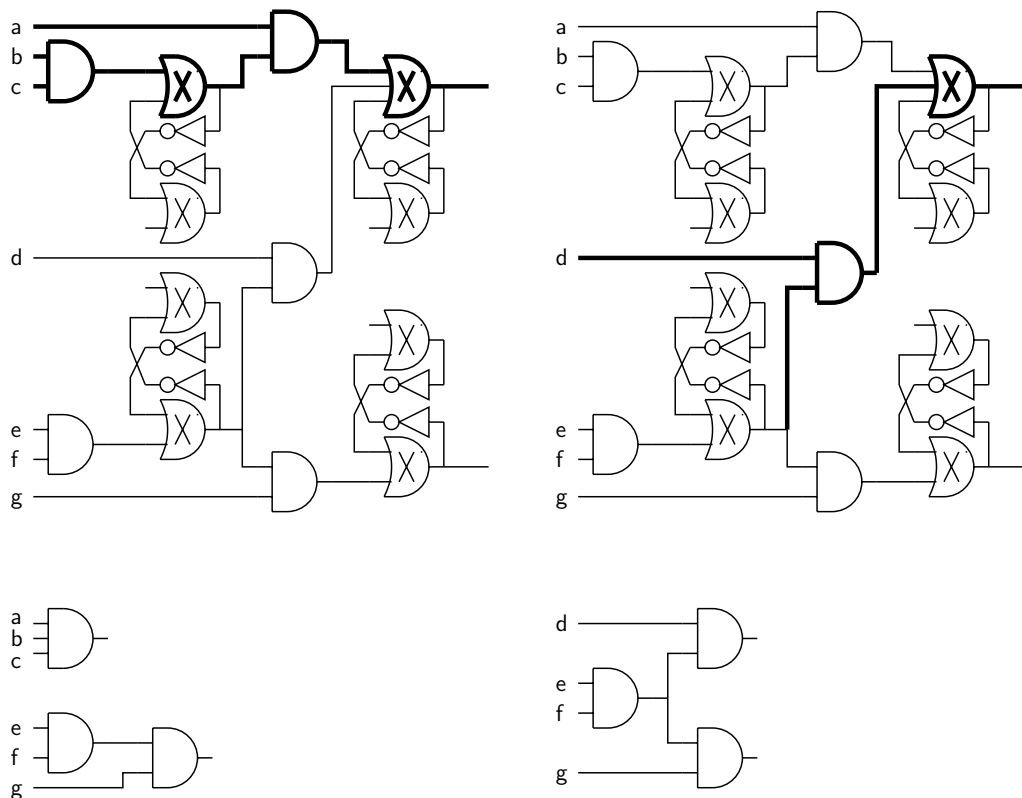


Figure 4.13: The different highlighted matches of the top part which is part of a larger network are leading to different implementations and therefore also different fanouts at certain nodes for the two alternatives.

*input ands* and *inverters*. The number of inputs of the *and* is not restricted to two but up to some other maximum, typically not more than 10 for reasons explained in the following section. Then *ugates* are used as in figure 4.9 to create the full decompositions of the *multi-input ands* which are then glued together taking the inverters into account by taking the right *ugate* outputs. The fanout numbers at the nodes within the decomposition of those *multi-input ands* are set to 1, and the node representing the original output gets the fanout number equal to the network before expansion. With those numbers crossing numbers can be calculated again.

The heuristic is based on the expectation (and now also experience) that the structure of the original network and the result after mapping are highly correlated (fig 4.14). The decomposed node is merely used to select a particular decomposition and thus likely has a fanout of 1 while at multi fanout points matches are chosen which end and implement exactly that point as this will improve sharing. The latter is due to the fact that in an area-delay trade-off

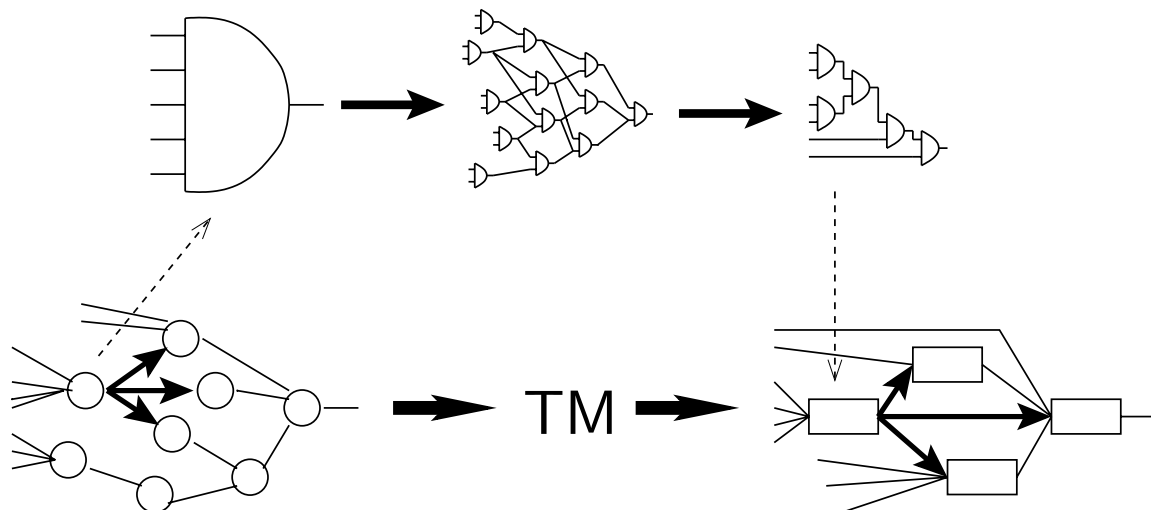


Figure 4.14: *The structure before expansion is likely closely related to the structure after covering when only decompositions of multi-input and nodes is applied.*

context it is advantageous to share as long as delay permits this. Only critical nodes will generate matches extending over fanout points, but as patterns are small the effects are limited and the preference of the other nodes to implement fanout point will avoid propagation of errors a lot.

### 4.3 search space control

The choice nodes allows multiple representations of base functions to be encoded in one network such that the search space is enlarged. It will provide the inclusion and discovery of a wider range of alternatives which leads to greater flexibility in the trade-offs. Faster choices may become available for critical nodes and at the same time more area cost effective ones for the non critical nodes which just have to be fast enough. The growth of the search space, and directly related to that the network, heavily depends on how much of the alternatives are encoded. As said in the previous section the area delay trade-off heuristic uses some structural information of the technology independent optimized circuit and thus not full reduction is applied. But even while only encoding multiple decompositions of *multiple-input ands* in the decomposed network some limit has to be applied to limit the growth. Another problematic issue is the tree matching were multiple choices have to be taken into ac-

count at every choice node it covers. Therefore a different matching strategy is applied which in common technology mapping does not buy much but in the presence of choice nodes results in a dramatic improvement of matching speed.

### input limitations

A major problem is the quick growth of the search space by inclusion of choice nodes. In our case the number of inputs allowed for a node before decomposition determines the increase in an exponential way. The number of u gates generated for an associative operation like an *and* with  $N$  operands or inputs is already  $2^N - 1 - N$ . For every output of a u gate is the combination of subset of operands. All these subsets are in the power set of the operand set. There are  $2^N$  such sets. The empty set and the sets with only one operand should not be counted,  $N + 1$  is therefore subtracted from the cardinality of the power set. Then there are  $2^n$  alternatives to split the inputs of a particular u gate in two groups. Neither should be empty reducing this number by 2 and interchanging the two groups need not to be distinguished. Then the total number of *two input ands* at a choice node is  $(2^n - 2)/2 = 2^{n-1} - 1$ . Now the total number of such gates is obtained by summation over the product of the number of u gates with  $n$ , the  $N$  inputs and the number of gates in such a u gate. Written as  $\sum_{n=2}^N \binom{N}{n} * (2^{n-1} - 1)$  which can be proven to be  $\frac{1}{2}3^N - 2^N + \frac{1}{2}$ . This shows the exponential dependency of the growth of the search space on the number of inputs of the nodes before decomposition.

As a consequence limiting the maximum number of inputs before expansion into choice nodes vastly reduces the size of the expanded network. This has a direct impact on memory use but also a strong effect on runtime as the number of alternatives to be considered during matching depends on choice node alternatives which also reduces quickly as well as the number of choice nodes themselves. The exact limit has to be a compromise between circuit quality and a reasonable runtime and memory usage. As useful libraries do not contain patterns with trees with a depth of more than 6 *and* gates a reasonable limit would be 6 or 8. Still sometimes 10 does make a difference in quality and thus if 6 or 8 does not provide the solution maybe trying much harder is what you are willing to pay for. Even higher limits lead to exorbitant resource usage.



### partial matching

It the matching step with is most affected by the introduction of choice nodes. Traditionally all library patterns corresponding to the subnetwork preceding an given node are reported. By the use of choice nodes the possible alternatives for the preceding subnetwork have grown very fast. For every choice node on the path from the root of the pattern tree to the leaf multiple choice have to be evaluated. But the choices are not independent and the alternative choices for the first node on the path leads to different other choice nodes with multiple alternatives again and the total alternatives for only a single path in the pattern to evaluate are in the order of  $n * n$  assuming each choice node has  $n$  alternatives. Keeping in mind that then all combinations of alternatives paths and subpaths are possible and that  $n$  can already be in the order of 50, 250 or 500 when the input limit for decomposition is set to 6, 8 or 10 respectively it is clear there is a complexity and runtime problem. The size of the subnetwork to be considered depends on the size of the larger patterns and the length of the longest path determine the power of  $n$ .

Note however that most nodes involved have already been investigated before, as nodes are treated in a topological order. Only complete matches have been recorded at that time. Therefore some information might have been lost as the absence or presence of a particular pattern at a certain node will exclude or include the existence of a pattern with a similar subpattern of a node at its fanout. The choice nodes have enlarged the number of subnetworks enormously but at the same time there is also a lot of partial equality.

In [18], under the name *bottum up matching algorithm*, a technique is introduced that reports not only on all complete patterns, but also parts of patterns that match at that node. This enables reusing earlier generated information about matching subpatterns resulting in a considerable speed up without exorbitant memory requirements. For a normal library with a reasonable number of rather similar not too large gates the increase of memory is quite low. The only drawback is that it is restricted to libraries with equal input output delay for all inputs of a gate and to tree patterns only.

Construction of matches out of other matches is illustrated in figure 4.15. The symbol “-” stands for an initial value of a match: it represents the input of a complete match in the end, that means either the output of a complete match or a primary input of the network. Every match is constructed by *and*-ing two other partial matches or extending an existing partial match with an


















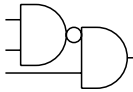
| Input  | Rootcell  | Partial-match   | Library-cell  | Name  | Area | Time |
|--|---|---|---|-------|------|------|
| "_"  |  |  |  | INV1  | 1    | 0.9  |
| "_"<br>"_"   |  |  |  | AND2  | 1    | 1.0  |
|   |  |  |   |       |      |      |
| <br> |  |  |  | AND4  | 4    | 1.4  |
| <br>"_"   |  |  |  | AOI21 | 3    | 1.6  |

Figure 4.15: *Library for partial matching*

*inverter*. The inputs of the (new) partial match are the union of the inputs of the previous partial matches. All partial matches at a node are found by combining all the partials at the inputs of the current node. If some of the partial matches are also complete matches then those are also kept. Note that row three in the library also shows the existence of partial matches which are not complete matches but which are required to find other matches based on this partial. Doing this procedure again in a topological order from inputs to outputs will generate all complete matches in an efficient way.

Figure 4.16 shows an example how a circuit can be mapped using partial mapping. The area and delay calculations are performed as follows:

- The delay of a new partial is set equal to the worst delay of the partials at its inputs. The delay of a complete match is obtained from its corresponding partial match by adding its own delay to the partial match delay. Of all isomorphic partial matches only the one with the lowest delay has to be saved.
- The area of a new partial match is the sum of the areas of the partial matches at the inputs. The area of a complete match is the area of the corresponding partial match plus its own area. For a minimum area result only the partial match with lowest area among all isomorphic partial matches has to be saved.

| Gate | Input            | Library-cell | Result | Time     | Match-type | Comment   |
|------|------------------|--------------|--------|----------|------------|-----------|
| PI:  | "_"              |              |        | 0        | p/c        |           |
| 1:   | "_" "_"<br>a , b | AND2         |        | 0<br>1.0 | p<br>c     |           |
| 2:   | "_" "_"<br>c , d | AND2         |        | 0<br>1.0 | p<br>c     |           |
| 3:   | "_" "_"<br>e , f | AND2         |        | 0<br>1.0 | p<br>c     |           |
| 4:   | "_" "_"<br>"_"   |              |        | 1.0      | p          |           |
|      |                  |              |        | 0        | p          |           |
|      | 2 , 3            | AND2         | "_"    | 2.0      | c          | Keep this |
|      | (c,d),(e,f)      | AND4         | "_"    | 1.6      | c          |           |
| 5:   | "_"<br>          |              |        | 1.0      | p          |           |
|      | 2                | INV1         |        | 0        | p          |           |
|      |                  |              | "_"    | 1.9      | c          |           |
| 6:   | "_" "_"<br>"_"   |              |        | 1.9      | p          |           |
|      |                  |              |        | 1.0      | p          |           |
|      | "_"              |              |        |          |            |           |
|      |                  |              |        |          |            |           |
|      | 1 , 5            | AND2         | "_"    | 2.9      | c          | Keep this |
|      | 1 ,( c , d )     | AOI21        | "_"    | 2.6      | c          |           |
| 7:   | "_"<br>          |              |        | 2.6      | p          |           |
|      |                  |              |        | 1.9      | p          |           |
|      | 6                | INV1         | "_"    | 3.5      | c          |           |
| 8:   | "_" "_"<br>"_"   |              |        | 3.5      | p          |           |
|      |                  |              |        |          |            |           |
|      | "_"              |              |        |          |            |           |
|      |                  |              |        | 2.6      | p          |           |
|      |                  |              |        | 1.9      | p          |           |
|      | "_"              |              |        |          |            |           |
|      |                  |              |        |          |            |           |
|      |                  |              |        |          |            |           |
|      | 4 , 7            | AND2         |        | 4.5      | c          | Keep this |
|      | 4 ,( 1 , 5 )     | AOI21        |        | 3.5      | c          |           |

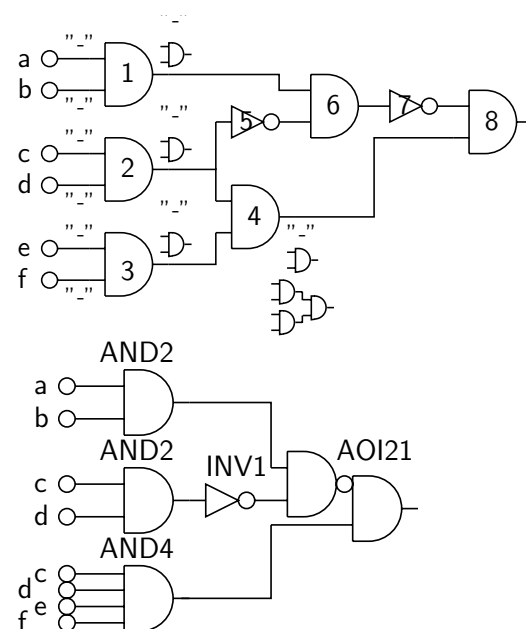


Figure 4.16: Example of partial matching procedure: A list of all steps during the procedure, the network after node 4 has been matched, and the final covering result.

In the example the matching and mapping effort is not much different from conventional matching and mapping and could very well be even more costly. But in the context of choice nodes the improvement is dramatic. It is mainly

due to the observation that there are often lots of similar substructures in all the matches to consider and on top of that a local best choice can be made at each choice node already which avoid the multiplication of choices. This is illustrated in figure 4.17. After the best among 2 times 4 isomorphic partial matches at the choice nodes have been evaluated only a simple combination of those partials lead to the next best partial. Using conventional tree matching 4 times 4 is 16 alternatives should have been evaluated.

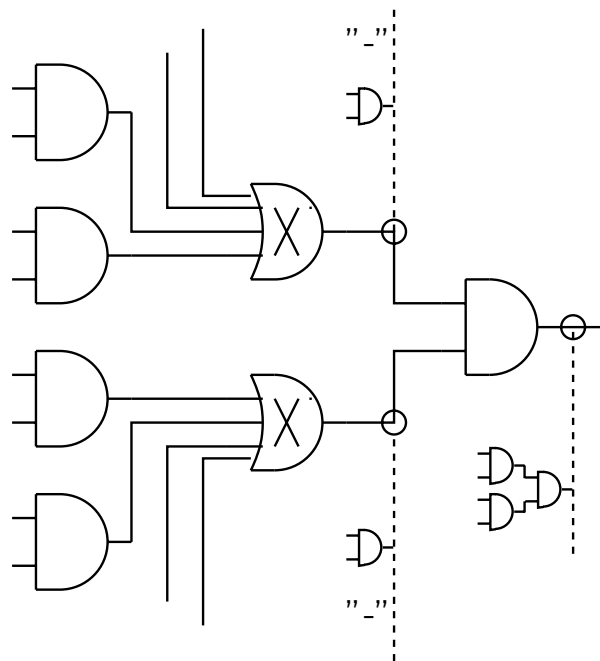


Figure 4.17: *Illustration of runtime improvement due to optimal partial match selection at choice nodes*

In the context of area delay trade-offs not only the fastest or smallest isomorphic partial should be stored but a trade-off. Still the same local reduction due to isomorphism is in place. The same techniques used for complete matches applies for partial matches too. In practice only storing the fastest and smallest partial matches still generates nice trade-off curves for complete matches while avoiding a lot memory and runtime increase.

Also the *crossing numbers* can directly be calculated for the partial matches: the number for a “-” is 1 and whenever a match is formed the numbers at the inputs are increased by the fanout value of the current node minus 1. The area estimates for the partials are adjusted using these numbers. The area of a complete matches is still produced in the same way and automatically crossing numbers have been counted for already.

This partial match approach is only applicable for tree patterns, while libraries are not restricted to that. The main problem is that it introduces dependencies on further decisions which are not possible when choosing locally optimal partial substructures. Relations between inputs, as required for leaf dags, can not be taken into account. In these cases still traditional tree matching has to be applied, but fortunately most patterns are usually trees.

Another restriction is the requirement of equal input output delay for all inputs of a match. This is also required to enable the local decisions of partial matches which resulted in the major gain. In general the worst case could be taken for the delay of the other inputs as well. Usually this does not result in a significant deviation as “skewed” patterns are seldom essential, and besides that they would disturb the assumptions of technology independent stages. Again if really needed the only solution is to resort to the traditional tree matching again, but at a considerable runtime cost.

## 4.4 experiments and conclusions

For the evaluation and experiments a modified library *lib2.genlib* [40] is used. The load independence is obtained by setting the corresponding gate characterization parameters to 0. Rise and fall delay of a gate is set equal to the worst input output delay in the original version of the library. When curves are used the number of points are limited to 10. Also the input limit for decomposition of multiple inputs is set to 10.

A comparison is made to conventional mapping [40][33] based on a single representation with the *two input and* gate and the inverter resulting from technology independent optimizations. Conventional is understood as partitioning the network in nonconvergent subnetworks followed by the application of the algorithm invented by Keutzer [24]. Table 4.1 clearly shows the improvements of the proposed approach. Much smaller as well as faster solution are possible and at equal delay the area is always lower.

Figure 4.18 shows how the different heuristics and procedures impact the result also compared to other state-of-the-art procedures. If the conventional method is extended with choice nodes the search space is enlarged and a faster circuit is found, but at the expense of more area. Extending the conventional method with speed-optimal covering which avoids the introduction of sub op-

| circuit | normal |       | trade-off curves and choice nodes |       |    |          |       |            |    |       |
|---------|--------|-------|-----------------------------------|-------|----|----------|-------|------------|----|-------|
|         | area   | delay | fastest                           |       |    | smallest |       | comparable |    |       |
|         |        |       | area                              | delay | %  | area     | delay | area       | %  | delay |
| C432    | 375    | 12.26 | 424                               | 7.33  | 40 | 310      | 13.56 | 321        | 14 | 12.56 |
| C499    | 694    | 9.63  | 785                               | 6.50  | 32 | 565      | 9.10  | 565        | 18 | 9.10  |
| C880    | 627    | 12.06 | 510                               | 7.02  | 41 | 411      | 10.17 | 411        | 34 | 10.17 |
| C1355   | 801    | 14.97 | 769                               | 7.61  | 49 | 743      | 9.13  | 743        | 7  | 9.13  |
| C1908   | 1173   | 14.02 | 938                               | 9.14  | 34 | 664      | 17.82 | 765        | 34 | 13.98 |
| C2670   | 1503   | 12.03 | 1285                              | 7.14  | 40 | 1050     | 10.71 | 1033       | 31 | 11.88 |
| C3540   | 2101   | 20.37 | 1829                              | 11.84 | 41 | 1510     | 20.72 | 1635       | 22 | 19.83 |
| C5315   | 3352   | 17.14 | 2540                              | 13.28 | 22 | 2186     | 25.23 | 2352       | 29 | 16.84 |
| C6288   | 3378   | 76.56 | 3447                              | 40.00 | 47 | 3223     | 54.00 | 3223       | 4  | 54.00 |
| C7552   | 4253   | 13.95 | 3706                              | 11.30 | 19 | 3157     | 22.26 | 3409       | 19 | 13.34 |

Table 4.1: *Area and delay comparison of the new approach with the conventional matching showing the enlarged search space is really effective*

tinality due to an intermediate selection also enlarges the search space to be reached and does provide a faster circuit at higher area cost again. The combination of the two extensions is even more powerful as on two different ways more alternatives become available resulting in a solution faster than the previous cases but at a quickly rising area cost. To deal with that problem the area control mechanism is used based on the trade off curves. The classical heuristic based on area division by number of fanouts does resolve this area increase problem to some extent, but the crossing number heuristics gives a much better result. Similar results are obtained for all the other circuits.

Table 4.2 shows the effects of the limits which can be imposed on the maximal input count before decomposition and the number of points in the trade-off curves as well as the gain obtained by partial matching. The introduction of partials does not introduce a much higher memory usage as was already expected. Run time clearly benefits from the local optimality choices of partial matches especially with large decompositions and thus high number of alternatives at choice nodes. In the classical case (input limit = 2) only a very slight increase is the result. The exponential nature of number of gates and alternative choices caused by the decomposition of multiple inputs is shown. The effect of the limiting points is more or less linear due to the fact that although the number of possible combinations grows exponential with the

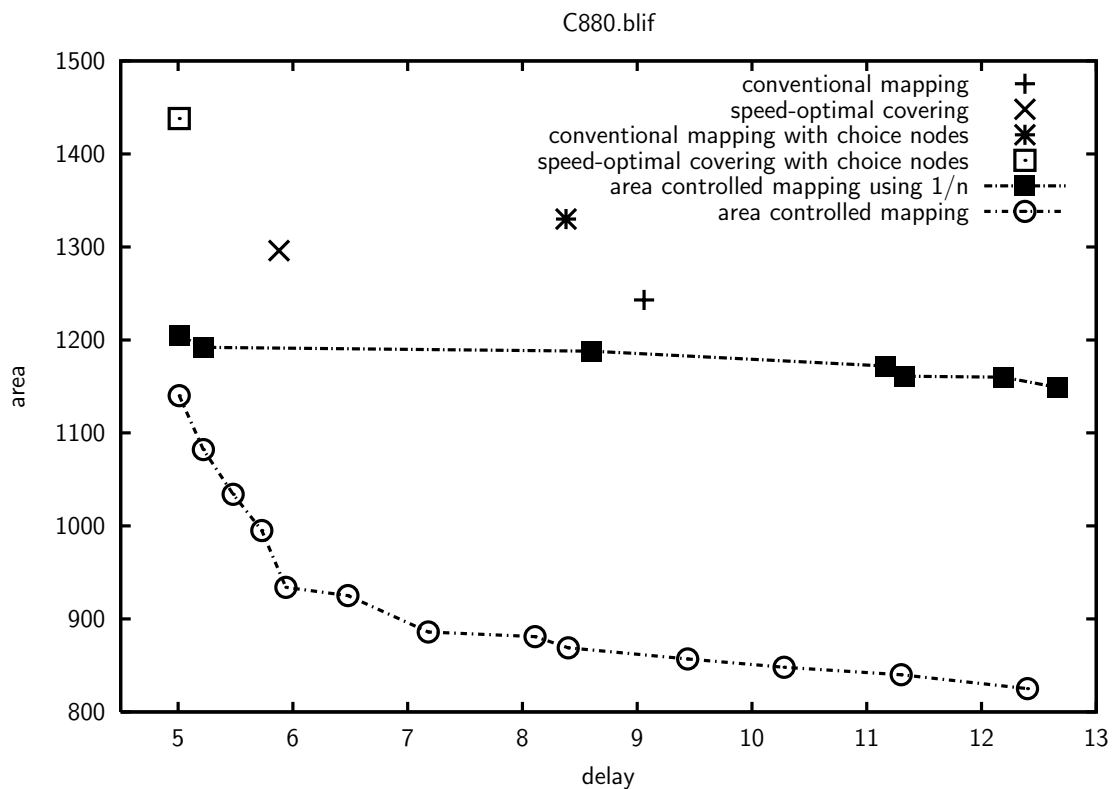


Figure 4.18: Comparison of various mapping procedures applied to C880 shows the effect of the different proposed procedures and heuristics.

number of points, still the needed and useful new points can be calculated in a smart way in only the sum of the number of points as number of evaluations.

It should not come as a surprise that the results are better in all cases. The choice node and covering techniques themselves are enlarging the search space both in an orthogonal way and thus the conventional case should still be in there. Only small errors due to the heuristics might in some cases result in slightly worse solutions.

The use of partial matches does not increase memory requirements a lot as apparently most partials are also complete matches. This is quite logical as commonly a library is constructed as a range starting from simple basic gates to more complex extensions or combinations of them. Therefore at rather low memory cost a high run time gain can be achieved in the context of choice nodes where local optimal partials reduce the negative run time effects of choice nodes.

| runtime/<br>memory<br>decomp | points in trade-off curve |             |             |
|------------------------------|---------------------------|-------------|-------------|
|                              | 1                         | 5           | 10          |
|                              | no partials               |             |             |
| 2                            | 0:02/220k                 | 0:03/359k   | 0:03/455k   |
| 5                            | 0:08/347k                 | 0:17/660k   | 0:25/871k   |
| 10                           | 11:19/1667k               | 32:04/3221k | 52:33/4542k |
|                              | with partials             |             |             |
| 2                            | 0:02/220k                 | 0:03/363k   | 0:04/460k   |
| 5                            | 0:04/631k                 | 0:08/841k   | 0:11/976k   |
| 10                           | 1:04/2039k                | 4:33/3515k  | 7:33/4864k  |

Table 4.2: Compare runtime and memory usage for circuit C432 due to the applied limit on inputs for the decomposition, the pruning in the trade-off curves and the application of partial matching

The application of the suggested solutions also provides new ways to deal with problems of critical multi fanout points (figure 4.19). Besides providing these alternatives also automatic evaluation takes place. One or more buffers can be inserted and parts of the network can be duplicated by different matches that better fit the output requirements. This offers a wider range of possibilities than serial repowering and capacitance splitting and using the trade-off curves the best one is automatically selected.

Remarkable progress has been shown in the research of technology mapping recently. Efficient solutions for convergent networks, with respect as well as speed with load independent delay models, were known well over a decade. Also cost minimization under timing constraints and finding the minimal cost among all fastest mappings could be solved efficiently for those networks. It was also known that extending cost minimization to more general networks was not likely, since the problem was proven to be  $\mathcal{NP}$ -hard. The fact that speed optimization could be solved in polynomial time for general networks was overlooked for quite some time, though virtually the same technique applies. In the meantime, an elegant way of capturing several representations with base functions in a single network was introduced. Also in these cases implicit direct speed optimization does occur but was not identified as such while it partly contributes to the improvements shown.



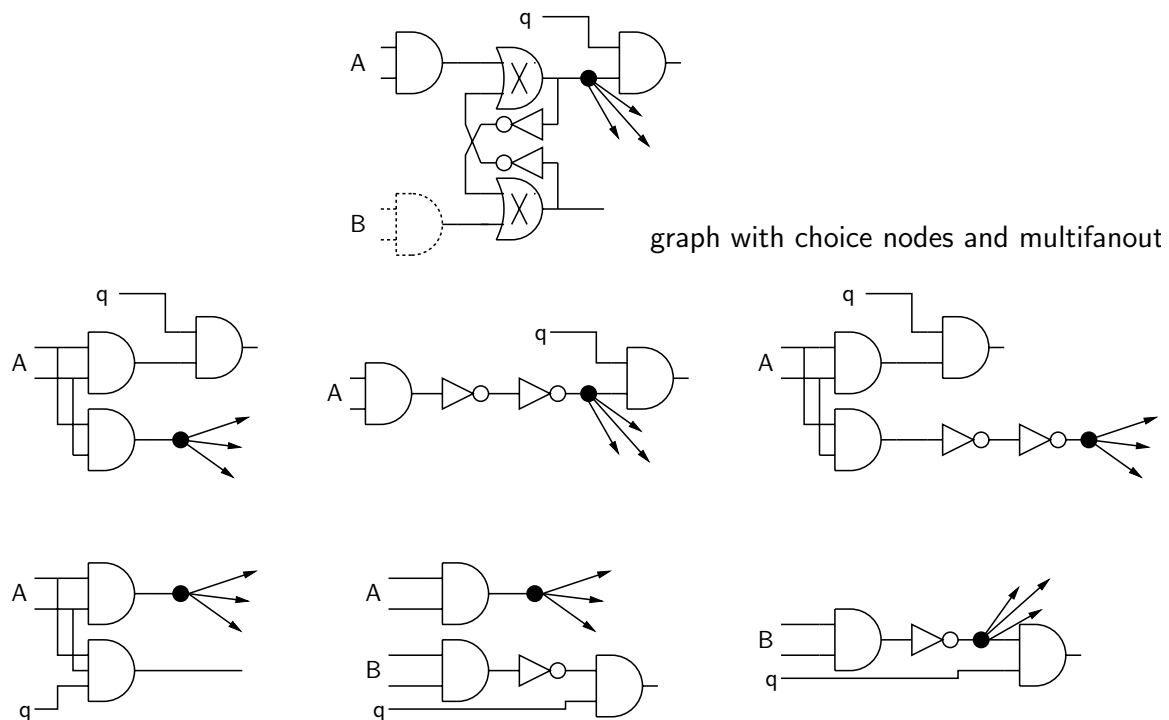


Figure 4.19: *The proposed techniques allow for the generation and automatic selection of a whole range of repowering solutions.*

In this chapter we have shown how to take advantage of these separate discoveries while avoiding failing miserably with other quality characteristics than speed and avoid the consumption of excessive amounts of area which was reported for both discoveries. Also the consumption of run time and memory was addressed allowing a trade-off between those two and the quality of the result. This is important for practical implementation and use.

An interesting observation is also that the different techniques covering, choice nodes and trade-offs are orthogonal. They can independently be applied and at the same time do not interfere with each other when used at the same time. This is also the reason that the results never get worse but only better when more of them are used at the same time. Actually concurrent invocation boosts the improvement up higher than on first sight based on individual improvements. The reason for that is that possible negative size effects of one method are compensated by advantages of others.

The required area control was obtained, not surprisingly, by adoption of area-delay trade-off curves. Without it, speed-optimized networks tend to grow out of proportion due to the optimization of all substructures. In this chapter

a procedure was presented using fanout numbers and choice nodes to enable good area guesses which would otherwise not be possible but required for the application of trade-offs. The impact on runtime is controlled by limiting the maximum number of points in those curves and the efficient generation of new points.

By the application of bottom-up or partial matching which would normally not result in any gain at all we were able to reduce dramatically the run time as it is an efficient technique in the context of choice nodes. It is based on choosing optimal local partial matches which means that only all alternatives at a choice node have to be evaluated once and there is no dependency on other choices at other choice nodes at all.

By keeping some structure in the decomposition process and limit the amount of decomposition heuristics for areas were feasible and runtime could be controlled. As was shown the number of gates and also the number of choices at each of the choice nodes of such gates grows exponential with the number of inputs of a node before decomposition. Considering the library patterns 6 or 8 as a limit works well at relative low costs. Setting the limit to 10 the network starts to grow quickly while only a little effect is notice in the quality of results.

The results presented show the effect and the impact of the heuristics very clearly. In a controllable and efficient way a wider search-space is evaluated which results in a wide range of area minimized delay solutions which can be used to select the required delay with minimum area cost from the delay but getting.

# Chapter 5

## conclusions

This thesis presented a refinement design approach based on the idea of wire planning to avoid the increasing number of design iterations to achieve timing closure. As wires are planned early their effects can be taken into account timely avoiding the need of correction later. Fortunately the delay of the long wires with significant impact on path delays turn out to be linear in length. Due to this linearity the total delay on a path of wire segments and modules with delays does not change when the wire segment lengths changes as long as the total wire length is constant.

The objective of the design process is to minimize the amount of delay wasted in wires and thus minimizes the wire lengths. The total delay in wires on a path from input pin to output pin can however never be smaller than the Manhattan distance between those pins. Therefore a design is optimal when all paths have this minimum length between the input and output pins. This results to paths with a monotonic increasing or decreasing  $x$  and  $y$  coordinates.

As a result the placement of the modules is restricted by the desired monotonic paths. Still multiple solutions are possible as the exact wire segmentation does not influence the total path delays any more. Therefore we can speak of a monotonic floor plan at this stage of freedom where module sizes and thus delays are still free.

Given the timing constraints the remaining delays on a path after subtraction of the wire delays can be assigned to the modules while minimizing the total area. This is done by a time budgeting technique using geometric pro-

gramming. By an efficient reformulation large problems can be handled. Some other small adoption from literature allows us to obtain a little more efficiency and results which are more robust within the context of assigning delays to modules.

Those assigned delay have now to be implemented by logic synthesis. Classical synthesis came from the time that area minimization was the main objective which later changed into minimizing delay to get higher clock performance. In our context we want to obtain a solution just fast enough with minimum area usage. This requires to explore the design space for a range of areas and delays. Recently presented adjustments to this process enables a larger search space with more potential solutions and thus better chance of one meeting just the required delay. To enable the exploration of solutions between fastest and smallest implementation in an efficient and structural way and avoid unnecessary high area claims trade-offs were used.

Within this context the following contribution were presented in this thesis:

- A non iterative design flow was presented and discussed which is based on wire planning, monotonic floor plans and constant delay synthesis. We showed how this can fit well together as the objectives of currently used algorithms are exactly those things taken as assumptions in this flow.
- The problem of delay budgeting was explored and modeled as mathematical programming. Reformulating the constraints resulted in a dramatic reduction of constraints enabling the practical solving of the budgeting problem. Other recently presented additional reductions and extensions were fit to the current case and implemented too resulting in another 50% gain and a robuster context.
- To manage the increased complexity of the enlarge search space and to obtain the objective of just fast enough circuits trade-offs were introduced. Heuristics were introduced the have even better advantage of the different orthogonal techniques used and result in an additional gain.

Although the iteration free approach presented does assume the implicit objective of currently used algorithms not all problems are solved. Not is the first place as the explicit optimization done in those algorithms are derivatives of the actual objectives. For future research the following aspect do show up.

First of all there is pin placement. The minimum wire length is clearly limited by the pin positions. They do also indirectly through the allowable wire paths influence the freedom and availability of a monotonic floor plan. Note that a better pin placement does not only benefit the presented flow. Assuming good routers try to achieve results which are detour free good pin placement would directly also improve these results.

Another point is obtaining monotonicity. It is depending on network topology and wiring between the placed pins. So some more changes can be required after a monotonicity aware pin placement. Some work has been done on network transformation in combination of monotonicity. This can be in the area of logic optimization as well as technology mapping.

Routing should actually already make monotonic wiring, as this would minimize wire length, but in practice different heuristics are used which are less strictly related to this objective. Attention should be paid that no detours are made to avoid congestion due to bad placement. On the other hand research on monotonicity aware routing could directly also give improvements in currently used flows.

The delay budgeting is depending on an area delay estimation. Although meeting the calculated delay is the only thing required to avoid disturbing the path delays, this clearly can have a negative effect on area usage or even placement. A mechanism to reuse and extrapolate area delay data from previous designs would be required. As back and forth interaction is not required for the delay also some sort of exploration or identification of a few points could improve robustness.

A rather new phenomena would be layer assignment. Delay budgeting based is based on the delay not used for wiring. This delay of wires do depend on layer. Therefore connections should also be assigned a layer before actual delay budgeting.



# bibliography

- [1] A.V. Aho and M.J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 6:333–340, 1975.
- [2] A.V. Aho, S.C. Johnson, and J.D. Ullman. Code generation for expressions with common subexpressions. *Journal of the Association of Computing Machinery*, pages 146–160, January 1977.
- [3] Magma Design Automation. *Blast Fusion white paper*. <http://www.magma-da.com>, Cupertino (CA), USA, May 2000.
- [4] X. Bai, C. Visweswariah, P.N. Strenski, and D.J. Hathaway. Uncertainty-aware circuit optimization. In *Proceedings of DAC*, pages 58–63, 2002.
- [5] R.K. Brayton and C.T. McMullen. The decomposition and factorization of boolean expressions. In *Proceedings of ISCAS*, pages 49–54, 1982.
- [6] K. Chaudhary and M. Pedram. A nearly optimal algorithm for technology mapping minimizing area under delay constraints. In *Proceedings of DAC*, pages 491–498, June 1992.
- [7] H.Y. Chen and S.M. Kang. icoach: A circuit optimization aid for cmos high-performance circuits. In *Integration, the journal of VLSI*, pages 185–212, 1991.
- [8] Theo Claasen. The logarithmic law of usefulness. *Semiconductor international*, 1998.
- [9] Jason Cong. An interconnect-centric design flow for nanometer technologies. *Proceedings of the IEEE*, 89(4):505–528, April 2001.
- [10] G. DeMicheli. *Synthesis and optimization of digital circuits*. McGraw-Hill, New York, 1994.

- 
- [11] H. Eisenmann and F.M. Johannes. Generic global placement and floorplanning. In *Proceedings of DAC*, pages 269–274, 1998.
- [12] J.P. Fishburn and A.E. Dunlop. Tilos: A posynomial programming approach to transistor sizing. In *Proceedings of ICCAD*, pages 326–328, 1985.
- [13] L.A. Glasser and D.W. Dobberpuhl. *The design and analysis of VLSI Circuits*. Addison-Wesley, 1985.
- [14] W. Gosti, A. Narayan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Wire planning in logic synthesis. In *Proceedings of ICCAD*, pages 26–33, 1998.
- [15] W. Gosti, A. Narayan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Wireplanning in logic synthesis. In *Proceedings of ICCAD*, pages 26–33, 1998.
- [16] J. Grodstein, E. Lehman, H. Harkness, B. Grundmann, and Y. Watanabe. A delay model for logic synthesis of continuously-sized networks. In *ICCAD*, November 1995.
- [17] J. Grodstein, E. Lehman, H. Harkness, B. Grundmann, and Y. Watanabe. A delay model for logic synthesis of continuously-sized networks. In *Proceedings of the European Design Automation Conference*, pages 458–462, November 1995.
- [18] C.M. Hoffman and M.J. O'Donnell. Pattern matching in trees. *Journal of the Association of Computing Machinery*, pages 68–95, 1982.
- [19] D.J.-H. Huang and A.B. Kahng. Partitioning-based standard-cell global placement with an exact objective. In *Proceedings of ISPD*, pages 18–25, 1997.
- [20] R. Sproull I. Sutherland. The theory of logical effort: designing for speed on the back of an envelope. In *Advanced Research in VLSI*, UC Santa Cruz, 1991.
- [21] J. Bruno and R. Sethi. Code generation for a one-register machine. *Journal of the Association of Computing Machinery*, pages 502–520, July 1976.
- [22] D.J. Jongeneel and R. Otten. Technology mapping for area and speed. *Integration, the VLSI*, pages 45–66, 2000.



- [23] D.J. Jongeneel, R. Otten, Y. Watanabe, and R.K. Brayton. Area and search space control for technology mapping. In *Proceedings of DAC*, pages 86–90, June 2000.
- [24] K. Keutzer. Technology binding and local optimization by dag mapping. In *Proceedings of DAC*, pages 341–347, 1987.
- [25] Y. Kukimoto, R.K. Brayton, and P. Sawkar. Delay-optimal technology mapping by dag covering. In *Proceedings of Design Automation Conference*, June 1998.
- [26] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness. Logic decomposition during technology mapping. *IEEE Transactions on Computer Aided Design*, pages 813–833, August 1997.
- [27] Chris Malachovski. When 10m gates just isn't enough... the gpu challenge. In *Proceedings 39th Design Automation Conference, New Orleans(LA), U.S.A.*, page 375, June 2002.
- [28] F. Mo, A. Tabbara, and R.K. Brayton. A force-directed marco-cell placer. In *Proceedings of ICCAD*, pages 177–180, 2000.
- [29] MOSEK. Mosek. <http://www.mosek.com>.
- [30] Ralph H.J.M. Otten, Lukas P.P.P. van Ginneken, and Narendra V. Shenoy. Speed: new paradigms in design for performance. *Proceedings International Conference on Computer Aided Design, San José(CA), U.S.A.*, page 700, November 1996.
- [31] R.H.J.M. Otten and R.K. Brayton. Planning for performance. In *Proceedings of DAC*, June 1998.
- [32] R.H.J.M. Otten and R.K. Brayton. Performance planning. *Integration, the VLSI*, pages 1–25, March 2000.
- [33] R. Rudell. Logic synthesis for vlsi design. Technical Report UCB/ERL M89/49, University of California, Berkeley, April 1989.
- [34] S.S. Sapatnekar and V.B. Rao. ideas: A delay estimator and transistor sizing tool for cmos circuits. In *Proceedings of Custom Integrated Circuits Conference*, pages 9.3.1–9.3.4, 1990.

- 
- [35] R. Sethi and J.D. Ullman. The generation of optimal code for arithmetic expressions. *Journal of the Association of Computing Machinery*, 17:715–728, 1970.
- [36] SIA. The national technology roadmap for semiconductors. Technical report, SIA, San Jose, 2003.
- [37] L. Stok, M. Iyer, and A.J. Sullivan. Wavefront technology mapping. In *Proceedings of DATE*, pages 531–536, 1999.
- [38] I. Sutherland, R. Sproull, and D. Harris. *Logical Effort: Designing Fast cmos Circuits*. Morgan Kaufmann publishers, November 1996.
- [39] Ivan Sutherland, Bob Sproull, and David Harris. *Logical Effort: designing fast CMOS circuits*. Morgan Kaufmann Publishers, San Francisco, CA, U.S.A., 1999.
- [40] H.J. Touati, C.W. Moon, R.K. Brayton, and A. Wang. Performance-oriented technology mapping. In *Proceedings of the 6th MIT Conference: Advanced Research in VLSI*, pages 79–97, 1990.
- [41] L.P.P.P. van Ginneken. *The Predictor - Adaptor Paradigm*. PhD thesis, University of technology Eindhoven, 1989.
- [42] C. Visweswariah and A.R. Conn. Formulation of static circuit optimization with reduced size, degeneracy and redundancy by timing graph manipulation. In *Proceedings of ICCAD*, pages 244–249, 1999.

# summary

Rapidly advancing process technology will enable man-made devices with complexities that are unprecedented in the history of mankind. The design scale of Integrated Circuits (ICs) is increasing exponentially according to Moore's law, approaching 1,000,000,000 transistors in the coming years. The low cost and huge capabilities of IC devices has already transformed all aspects of our daily life: from cellular phones, through electronic money in chip cards to digital surround sound entertainment centers. The demand for silicon Integrated Circuits that power these devices is expected to continue increasing in the near future. Especially in the area of computer generated imaging (e.g. for games and home systems), there is an unsatiated demand for processing power.

In the past 3 decades, the anatomy and physical size of an Integrated Circuit has remained roughly the same. The active transistor devices are produced at the surface of a silicon semiconductor wafer. To interconnect the components several layers of metal wire tracks are available. Both the wire width and transistor dimensions are shrinking by a factor of 2 every 18 months. Unfortunately, this increase in scale has made certain parasitics - most notably the wire delay - more prominent than in older process generations. One might say that the complex physics of silicon manufacturing is increasing the 'silicon complexity' of the design. Overall the designers of Integrated Circuits are facing a two-fold complexity increase: in terms of the number of components and in terms of battling parasitic electrical effects of nanometers devices.

The times that IC's are designed manually have long gone. Electronic Design Automation (EDA) tools are software programs that design and verify an integrated circuits. We can distinguish multiple levels of abstraction in this design process, each with its own set of EDA tools. At the bottom level there are tools that generate (and check) masks. Automatic routing algorithms are used to generate the interconnect pattern. Other programs automatically place hundreds of thousands of gates instances on the chip surface. This process

of placement and routing is called the *physical design* of the IC. At a higher abstraction level are the *logical synthesis* programs that generate a net list of gates from functional description in a hardware description language. Higher levels of design abstraction have also been automated.

This thesis focusses on the interface between automatic logical and physical design of integrated circuits. The actual placement of the components determines the length of the wires, and with that it sets the amount of the wire delay. This parasitic delay dominates the speed and quality of the result. Therefore, the transition from the logical to the physical domain is where 'the rubber meets the road' in the sense that real transistors and real wires and their parasitics must be dealt with properly. The wire planning concept of this thesis is a major step towards a better synergy between the domains.

It should be emphasized that the new synergetic approaches that are described in this thesis not rely on iteration. In contrast, a conventional design flow does require iteration to converge to a feasible solution. In more detail, the main steps at the bottom end of the design flow are as follows:

1. **Logical synthesis** tools produces a net list of gates that implement the desired functionality. At this time the placement of the components is unknown. Therefore the timing optimization and sizing of the gates can only be performed using a statistical wire load model. The output of the logic synthesis tool is a interconnected net list in which the gates have a specific size, based on the statistical wire load model.
2. **Placement** of the gates. The primary objective of the placer is to ensure that the gates are not overlapping and the total wire length is minimized. Issues such as minimizing the length of the most timing critical paths are only taken as secondary objectives. The large number of critical path is often over-constraining the algorithm.
3. **Routing** of the wires to interconnect the gates. This generates the topology and layer of the wires, and with that it sets the parasitic wire capacitances. Some wires will need to detour around obstacles, resulting in a higher than expected parasitic load.

The major problem with this flow lies in step 1, where the optimization algorithms are unaware of the actual parasitic delays of the wires. The wire delays are only known as result of step 2. In most cases it is not possible to place

the gates such that the timing of the entire circuit is feasible. To alleviate the timing problem designers are running steps 1, 2, and 3 a number of times in an iterative fashion. Each time the latest parasitic data is fed back into the optimization algorithm of step 1, and each time the circuit is placed again from scratch. This iterative process is not only slow, it is also not guaranteed to converge, especially in the latest processing technologies.

The cause of the problem in the above iterative approach is that the synthesis tool in step 1 made a premature decision on the size of the logic gates. At a later stage, it was not possible to recover from the likely mistakes in this decision. A better paradigm is *stepwise refinement*, where at each step in the design flow a single parameter is fixed. All other parameters are left undecided. Therefore, decisions are postponed until as late as possible, when sufficiently accurate data is available to take such decisions.

A good example of a successful stepwise refinement flow for logical and physical design is *constant delay synthesis*. This approach fixes the delay and functionality of a gate in step 1, but postpones the decision on the size of the gate until after placement (step 2). The idea is to use gate sizing is used to maintain the given delay target. Timing and wiring closure can be achieved without iterating back to earlier steps, which speeds up the design flow.

As indicated earlier, the speed and performance of an IC is primarily determined by the parasitic wire delay. It is only the small fraction of long wires (with high delay) that is responsible for the overall chip timing. The main thrust of this thesis is to plan these critical wires beforehand. In chapter 2 we derive a stepwise refinement paradigm for long wires called *wire planning*. We introduce the concept of a *monotonic* floor plan, in which the total wire length (and with that the parasitic delay) from pin to pin is fixed. We will show that it is also minimum if the wires is optimally buffered, and that we may assume that the delay of this wire grows linear with its length. In the wire planning paradigm we can therefore calculate delay budgets for the modules on a path before the actual implementation using placement and routing.

In chapter 3 we address the issue of deriving the proper delay budgets. A straightforward approach for a delay budgeter would be to enumerate all path. This is clearly infeasible as the number of paths grows much faster than the number of components in the paths. To address this issue we derive a novel reduction methodology that can guarantee the same result. We also present additional improvements that improve the speed and make the result more robust.

After budgets have been calculated the synthesis algorithms will have to create a solution which complies to the budgets for each module. Its goal is to generate a circuit that is just fast enough. Excess speed would only result a larger circuit area and higher power consumption. We will present a modified constant delay synthesis approach with these objectives in chapter 4.

In practice designers are looking for a good trade-off between circuit speed and chip area. Therefore it is required to explore a larger search space than just the fastest possible implementation. A well-known approach uses choice-nodes. This is advantageous within the context of stepwise refinement paradigm of this thesis since the circuit topology choice is postponed to a later point in the flow.

Simultaneously the fast implementations as well as the smaller (but slower) coexist. In the enlarged search space faster implementations then usual become available for all parts, including the non-critical parts, resulting in an area blowup. But area and delay can be traded off per module. We address the issue of automatically deriving a feasible trade-off in the context of choice-nodes. The approach yields the overall best trade-off for the complete set of modules.

The larger the search space for possible implementations of the circuits, the longer the algorithm will run. Finding the optimal choices become exponential harder with the number of choices, since all possible combinations of choices have to be evaluated. To alleviate this issue an implementation is presented which controls the search space problem as well as the area blowup.

# samenvatting

De snelle vooruitgang in de procestechnologie maakt door de mens ontworpen systemen mogelijk die complexiteiten bezitten die nog nooit vertoond zijn in de geschiedenis van de mensheid. De omvang van geïntegreerde circuits (ICs) neemt exponentieel toe volgens Moore's Law en benadert de 1,000,000,000 transistors in de komende jaren. De lage kosten en de grote capaciteiten van ICs hebben alreeds alle aspecten van ons dagelijks leven veranderd: van mobiele telefoons, via elektronisch geld in chip kaart tot digitale surround sound systemen. De vraag naar geïntegreerde circuits gemaakt van silicium om al deze systemen te besturen zal naar verwachting blijven groeien in de nabije toekomst. Met name op het gebied van beelden die worden gegenereerd met de computer (b.v. voor spelletjes en thuisbioscopen), is er een onverzadigbare vraag naar meer rekenkracht.

In de afgelopen 3 decennia is de anatomie en fysieke afmeting van een geïntegreerde circuit ongeveer gelijk gebleven. De actieve transistor elementen worden geproduceerd op het oppervlak van een silicium halfgeleider schijf. Om alle componenten met elkaar te verbinden zijn meerdere lagen met metaaldraden beschikbaar. Zowel de afmeting van de draden als die van de transistoren halveren elke 18 maanden. Helaas maakt deze verkleining bepaalde parasitaire effecten - met name tijdsvertraging in draden - steeds prominenter dan bij vorige proces generaties het geval was. Men zou kunnen zeggen dat de fysieke complexiteit van het produceren van silicium de "silicium gerelateerde complexiteit" van het ontwerpen doet toenemen. In het algemeen worden ontwerpers geconfronteerd met een tweeledige toename van complexiteit: in termen van het aantal componenten en in termen van het overwinnen van de parasitaire elektrische effecten van elementen met nanometer afmetingen.

De dagen dat IC's met de hand ontworpen werden liggen ver achter ons. Elektronische Ontwerp Automatisering (Electronic Design Automation – EDA)

gereedschappen zijn software programma's om geïntegreerde circuits te ontwerpen en te verifiëren. We kunnen verschillende abstractie niveaus in het ontwerpproces onderscheiden, elk met zijn eigen set van EDA gereedschappen. Op het laagste niveau zijn er gereedschappen om maskers te genereren (en te controleren). Automatische bedrading algoritmen worden gebruikt om alle interconnecties te maken. Andere programma's plaatsen volautomatisch honderden of duizenden cellen op het chipoppervlak. Dit proces van plaatsing en bedrading wordt het fysiek ontwerpen (physical design) van het IC genoemd. Op hoger abstractie niveau zijn er de logische synthese programma's die een netwerk van cellen produceren vanuit een functionele beschrijving in een hardware-beschrijvings-taal. Ook hogere abstractie niveaus van het ontwerp proces zijn geautomatiseerd.

In dit proefschrift ligt het accent op de interactie tussen het geautomatiseerde logisch ontwerpen en het fysieke ontwerpen. De uiteindelijke plaatsing van componenten bepaalt de lengte van de draden, en daarmee ook de hoeveelheid tijdsvertraging van de draden. Deze parasitaire vertraging domineert de snelheid en de kwaliteit van het uiteindelijke resultaat. Daarom is het de overgang van het logisch domein naar het fysieke domein waar het om draait, in de zin dat er juist op dat moment omgegaan moet worden met de werkelijke transistors en werkelijke draden en hun parasitaire neven effecten. Het concept uit dit proefschrift om bedrading te plannen is een grote stap voorwaarts naar een betere synergie tussen deze domeinen.

Nadrukkelijk moet gesteld worden dat de nieuwe synergetische benaderingen zoals die beschreven zijn in dit proefschrift niet gebaseerd zijn op iteratie. Een conventioneel ontwerptraject daarentegen heeft iteratie nodig om te convergeren naar een oplossing die voldoet. In meer detail zijn de belangrijkste stappen aan het einde van het ontwerp traject als volgt:

1. **Logische synthese** programma's produceren een netlijst van cellen die de gewenste functie implementeren. Op dat moment is de plaatsing van de componenten nog niet bekend. Daarom kan de tijdsvertraging optimalisatie en het aanpassen van de afmetingen van de cellen alleen gebeuren met behulp van een statistisch model van de capacatieve belasting door de draden. Het resultaat van een logische synthese programma is een netlijst van verbonden cellen met een specifieke afmeting gebaseerd op het statistisch draad model.



2. **Plaatsing** van de cellen. Het hoofddoel van de plaatser is te verzekeren dat de cellen niet overlappen en dat de totale draadlengte wordt geminimaliseerd. Zaken als het minimaliseren van de lengte van het meest kritische pad komen pas op de tweede plaats aan de orde. Het grote aantal kritische paden legt vaak te veel eisen op aan het algoritme.
3. **Bedrading** neerleggen om de cellen te verbinden. Dit levert de topologie en bedradingslag op voor de draden. Daarmee bepaalt het ook de parasitaire capaciteiten van de draden. Sommige draden zullen een omweg moeten maken om een obstakel heen wat resulteert in een hogere parasitaire belasting dan verwacht zou worden.

Het grootste probleem met deze aanpak zit hem in stap 1 waar de optimalisatie algoritmen niets weten van de werkelijke parasitaire tijdsvertraging van de draden. De tijdsvertraging van de draden zijn het resultaat van stap 2. In de meeste gevallen is het niet mogelijk om de cellen zo te plaatsen dat de eis van een bepaalde snelheid van het totale circuit gehaald wordt. Om dit snelheidsprobleem op te lossen, past een ontwerper de stappen 1, 2, en 3 een aantal keren in een iteratieve manier toe. Elke keer worden de laatste parasitaire gegevens teruggevoerd in het optimalisatie algoritme van stap 1, en wordt het circuit volledig opnieuw geplaatst. Dit iteratieve proces is niet alleen traag, er is ook geen garantie van convergentie, met name voor de laatste proces technologie.

De oorzaak van het probleem in de bovenstaande iteratieve benadering is de premature beslissing over cel afmetingen door het syntheseprogramma in stap 1. Op een later moment is het niet mogelijk terug te komen op mogelijke fouten over deze beslissing. Een beter paradigma is *stepwise refinement* (stapsgewijze verfijning), waarbij in elke stap van het ontwerp proces slechts een parameter wordt vastgelegd. Alle andere parameters blijven onbeslist. Beslissingen worden zo lang mogelijk uitgesteld tot voldoende nauwkeurige informatie beschikbaar is om de beslissing te kunnen nemen.

Een goed voorbeeld van een succesvolle benadering met *stepwise refinement* van een logische en fysiek ontwerp traject is *constant delay synthesis*. Deze benadering legt de tijdsvertraging en functies van de cellen vast in stap 1, maar stelt de beslissing over de afmetingen van de cellen uit tot na plaatsing (stap 2). Het idee is om het aanpassen van afmetingen van de cellen te gebruiken om ten alle tijden de doelstelling van de gegeven tijdvertraging te handhaven. Het

behalen van de snelheid doelstelling en het bedraden kan nu gedaan worden zonder iteratie naar eerdere stappen. Dit versnelt het ontwerptraject.

Zoals eerder aangegeven is, wordt de snelheid en performance van een IC voornamelijk bepaald door de tijdsvertraging in de draden. Het is slechts een klein aantal lange draden (met hoge tijdsvertraging) die verantwoordelijk zijn voor de uiteindelijke snelheid van de chip. Het belangrijke idee in dit proefschrift is deze kritieke draden van tevoren te plannen. In hoofdstuk 2 leiden we een stepwise refinement paradigma af voor lange draden onder de naam *wire planning* (draden plannen). We introduceren een monotoon plaatsingsplan waarin de totale draadlengte (en daarmee dus de parasitaire tijdsvertraging) van aansluitpin tot aansluitpin vast ligt. We zullen laten zien dat deze tijdsvertraging ook minimaal is als de draden optimaal gebufferd zijn. Tevens laten we zien dat we kunnen aannemen dat de vertraging lineair toeneemt met de lengte. Daarom kunnen we in het wire planning paradigma de tijdsbudgetten voor de cellen op een pad berekenen nog voor de werkelijke implementatie met behulp van plaatsing en bedrading.

In hoofdstuk 3 behandelen we het aspect van het afleiden van de juiste tijdsbudgetten. Een rechttoe rechtaan benadering van een tijdsbudgetter zou bestaan uit het opsommen van alle paden. Dit is natuurlijk onmogelijk aangezien het aantal paden veel sneller groeit dan het aantal elementen in de paden. Om dit aspect aan te pakken, leiden we een nieuwe reductie methodologie af die dezelfde resultaten kan garanderen. Ook presenteren we enkele additionele verbeteringen die de snelheid verbeteren en robuustere resultaten opleveren.

Nadat de budgetten berekend zijn moeten de synthese algoritmen oplossingen produceren voor de modules die overeenkomen met deze budgetten. Het doel is een circuit te maken dat net snel genoeg is. Te veel snelheid zou resulteren in een groter circuit oppervlak en hoger energie verbruik. We zullen een gemodificeerde constant delay benadering met deze doelstelling presenteren in hoofdstuk 4.

In de praktijk zoeken ontwerpers naar een goede afweging tussen circuit snelheid en chip oppervlak. Daarom is het nodig een grotere zoekruimte te doorzoeken in plaats van het zoeken naar slechts de snelste oplossing. Een bekende benadering gebruikt keuze knooppunten. Dit is aantrekkelijk in de context van het stepwise refinement paradigma in dit proefschrift omdat dan de topologie beslissing uitgesteld wordt tot een later punt in het ontwerpproces dan gebruikelijk is.

Op hetzelfde moment bestaan er zowel de snelle implementaties als ook de kleinere (maar tragere). In de vergrote zoekruimte komen snellere implementaties dan gebruikelijk beschikbaar voor alle delen, inclusief de niet kritieke. Dit resulteert in een explosieve toename van oppervlakgebruik. Maar oppervlak en vertraging kunnen afgewogen worden per deel. We zullen het automatische afleiden van een afweging tussen oppervlak en vertraging in de context van keuzeknooppunten behandelen. Deze benadering resulteert in de beste afweging voor de complete set van modules.

Hoe groter de zoekruimte voor mogelijke implementaties van de circuits hoe meer tijd het kost om het algoritme te draaien. Het vinden van de optimale keuzes wordt exponentieel moeilijker met het aantal keuzes, omdat alle mogelijke combinaties van keuzes geëvalueerd moeten worden. Om dit probleem te verlichten presenteren we een implementatie die zowel de controle brengt over het probleem van de zoekruimte als het probleem van explosief groeiend oppervlak.



# acknowledgements

I would like to thank Prof. R. Otten for giving me the chance to do my Ph.D. at the Delft University of Technology. The work was done mainly in the Systems and Circuits group in Delft but later also partly in the Information Communication and Systems group of Eindhoven University of technology. I would like to thank all members of both groups for the various support and good discussions. The time I spend with Mr. Sawkar at Intel Strategic Cad Labs in Hillsboro, USA, have been worth a lot even though this was just before stating my Ph.D. and I would like to thank him for that. I am also grateful to Prof. R. Brayton of University of California Berkely, USA, for the opportunities to spend time in his group and exchange ideas. I met there with Mr. Y. Watanabe and we had some fruitful discussions about choice-nodes. During the stay of Mr. Visweswariah from IBM, I had a chance to take advantage of his knowledge about timing and delay calculations. Many thanks for that. Also many thanks to Prof. Groeneveld for his help in getting the general subject of what design automation is all about in this thesis in a nice shape. Finally I can not forget my family. Thanks to my wife for keeping me alive when I was totally concentrated into writing this thesis but also for her general support. And my daughters for their timed and untimed but often worthwhile interrupts.



# biography

Dirk-Jan Jongeneel was born April 29, 1975 in Dordrecht, the Netherlands. In 1993 he started his studies at the department of Electrical Engineering of Delft University of Technology at Delft. The work for his Masters was done in the Circuits and Systems group of Prof. R. Otten. Part of the master thesis work was done in the group of Prof. Brayton at the University of California Berkeley in the USA. He graduated cum laude on September 15, 1998. From July till September 1998 he visited Intel Strategic Cad Labs in Hillsboro ,Oregon, USA. He extended there the work from his Masters. On December 1998 he started working toward a doctorate in the Circuits and Systems group of the Delft University of Technology in the field of Design Automation. After Prof. Otten changed postions, part of the time the work was done in the Information and Communication Systems group of the Electrical Engineering department of Eindhoven University of Thechnology. He expects to receive this degree based on the work presented in this thesis on April 26, 2004. Currently he is working at Magma Design Automation at its Eindhoven site.

