Information-driven general functional decomposition targeted to gate libraries
Bieganski, S.J.

DOI:
10.6100/IR735585

Published: 01/01/2012

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

Take down policy
If you believe that this document breaches copyright please contact us (openaccess@tue.nl) providing details. We will immediately remove access to the work pending the investigation of your claim.

Download date: 26. Jan. 2019
Information-driven General Functional Decomposition targeted to Gate Libraries

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven,
op gezag van de rector magnificus, prof.dr.ir. C.J. van Duijn,
voor een commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen op
donderdag 25 oktober 2012 om 16.00 uur

door

Szymon Biegański

geboren te Łódź, Polen
Dit proefschrift is goedgekeurd door de promotor:
prof.dr.ir. R.H.J.M. Otten

Copromotor:
dr.ir. L. Józwiak

Druk: Universiteitsdrukkerij Eindhoven

CIP-DATA LIBRARY TECHNISCHTE UNIVERSITEIT EINDHOVEN

Biegański, Szymon J.
A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-3243-8
NUR 959
Trefw.: decompositiemethoden / programmeerbare logische schakelingen / logische schakelingen ; automatentheorie / logische schakelingen ; CAD.
Subject headings: electronic design automation.
Summary

The research results presented in this thesis were obtained in the framework of a PhD research that aimed at proving the applicability of the information-driven approach to circuit synthesis, as well as, the theories of general decomposition and information relationship measures to circuit synthesis using complex gates libraries.

Traditional circuit synthesis methods implemented in today’s automatic logic synthesis tools deal only with some very special cases of possible circuit structures and are not well adjusted to the current and future technologies and synthesis targets. Substantial improvement could only be achieved through development and implementation of a new generation of design paradigms, methods and tools, more suitable in the new situation. Modern micro-electronic technology enables building of extremely large circuits and systems, and offers a great diversity of logic building blocks, while the traditional methods are basically targeted to AND-OR-NOT or MUX circuits, and require a sophisticated technology mapping for other targets. However, even the most sophisticated technology mapping cannot guarantee proper final results, if the initial synthesis is performed without a good relation to the actual target. Moreover, in modern designs, the interconnections increasingly decide all important circuit features (area, speed, power dissipation), and a flexible multi-objective optimization and trade-off exploitation are indispensable. The information-driven approach to circuit synthesis has a potential to resolve all those issues.

Within the project the main issues of library modeling for information-driven synthesis, multi-valued sub-function encoding etc. were researched, and a prototype synthesis method and corresponding EDA-tool were developed, that considerably differ from all other known methods and tools.

The experimental results from the tool that implements the new method demonstrate that the method and tool targeted to the gate-based circuits deliver much better circuits than the other methods and demonstrate that the information-driven general decomposition produces very fast and compact gate-based circuits.
Samenvatting

De gepresenteerde onderzoeksresultaten zijn behaald in het kader van een promotieonderzoek, dat als doel had het bewijzen van toepasbaarheid van een informatiegestuurde benadering voor circuit synthese en de theorie van algemene decompos tie en informatie relaties en acties toegepast op complexe gate bibliotheeken.

Traditionele circuit synthese methodes geïmplementeerd in de huidige automatische logische synthese hulpmiddelen behandelen slechts enkele zeer specifieke gevallen van mogelijke circuit structuren en zijn niet goed voorbereid op huidige en toekomstige technologieën en synthese doelen.

Substantiële verbeteringen konden slechts bereikt worden door de ontwikkeling en implementatie van een nieuwe generatie ontwerp paradigmas, methoden en hulpmiddelen, welke meer geschikt zijn voor de nieuwe situatie.

Moderne micro-electronica technologie maakt het mogelijk om extreem grote circuits en systemen te bouwen en biedt een grote diversiteit aan logische bouwstenen, terwijl de traditionele methodes gericht zijn op AND-OR-NOT of MUX circuits en ze behoeven een geavanceerde technologie mapping voor andere doelen. Echter zelfs de meest geavanceerde technology mapping kan geen garantie geven voor geschikte eindresultaten, als de initiële synthese is uitgevoerd zonder een goede relatie ten opzichte van het actuele doel.

Bovendien bepalen, in moderne ontwerpen, in toenemende mate de interconnecties alle belangrijke circuit eigenschappen (oppervlakte, snelheid, energiever bruik). En flexibele multi-objectieve optimalisatie en afwegingen zijn onmisbaar. De informatie gestuurde benadering voor circuit synthese heeft de potentie om al deze beperkingen op te lossen.

Binnen het project werden de voornaamste problemen van bibliotheek modelleren voor informatie gestuurde synthese, multi-value sub functie encoden, etc onderzocht en een prototype synthese model en bijbehorend EDA-tool werden ontwikkeld, welke aanzienlijk verschillen van alle andere bekende methodes en tools.

De experimentele resultaten van het tool waarin de nieuwe methode is geïmplementeerd demonstreert dat de methode en de tool gericht op gate gebaseerde circuits aanzienlijk betere circuits oplevert dan de andere methodes en demonstreert dat de informatie gestuurde algemene decompositie snelle en compacte gate gebaseerd circuits oplevert.
I would like to express gratitude to everyone who, in one way or another, contributed to successful completion of the research presented in this thesis. First of all, I would like to thank prof. Mario Stevens, who regretfully is no longer amongst us, for accepting me as a new member of the ICS/CND group. I am grateful to prof. Raph Otten for his willingness to succeed prof. Mario Stevens as my first promotor.

I would like to extend my sincere appreciation to dr. Lech Jóźwiak for his faith and confidence in me to carry out this work, for guidance and support during countless hours of interesting discussions.

My fellow researchers deserve special thanks for the time we spent exchanging thoughts and ideas: Artur Chojnacki, Aleksander Ślusarczyk and Dominik Gawłowski. Further, I would like to thank Marja de Mol-Regels and Rian van Gaalen for their assistance with formal issues.

Szczególnie gorąco dziękuje najbliższej rodzinie za wsparcie i zrozumienie podczas całego okresu pracy nad projektem i niniejszą książką.

vii
# Contents

1 **Introduction**  
1.1 Digital circuit design process ................................. 1  
1.2 Rationale ......................................................... 2  
1.3 Subject .................................................................. 4  
1.4 Aim ..................................................................... 5  
1.5 Main contribution .................................................... 6  
1.6 Thesis outline .......................................................... 7  

2 **Basic definitions**  
2.1 Preliminary definitions ................................................. 9  
2.2 Set systems .............................................................. 10  
2.3 Boolean functions and Boolean algebra ......................... 12  
2.4 Representations of Boolean functions ......................... 17  
2.5 Information Relationship Measures ............................. 20  
2.6 Information and information relationship analysis ........... 22  
  2.6.1 Information representation in discrete systems .......... 22  
  2.6.2 Information relationships ..................................... 23  
  2.6.3 Information relationship measures ......................... 26  

3 **Information-driven General Functional Decomposition**  
3.1 Classical functional decomposition .............................. 31  
3.2 General decomposition ............................................... 34  
  3.2.1 Combinational machines ..................................... 35  
  3.2.2 General decomposition theorem ............................ 37  
  3.2.3 Special cases of general decomposition ............... 47  
3.3 Summary .............................................................. 49  

4 **Information-driven Functional Decomposition targeted to gate based technologies**  
4.1 Problem ................................................................. 51  
  4.1.1 The precise research problem formulation ............... 52  
4.2 Decomposition method – strategy ............................... 53  
  4.2.1 Single decomposition step ................................... 61  
  4.2.2 Symbolic sub-function selection .......................... 61  
  4.2.3 Selection of the physical implementation ............... 65  
  4.2.4 Multi-valued sub-function realization ................. 65
4.3 Summary ................................................................. 67

5 Related research ....................................................... 69
  5.1 Functional decomposition approaches .......................... 69
  5.2 Multi-valued sub-function realization .......................... 71
  5.3 Traditional technology mapping .................................. 72
    5.3.1 Complex gates generation .................................... 74
    5.3.2 Terminal Suppressed BDDs in technology mapping ......... 75
    5.3.3 Technology mapping for cell library ......................... 76
  5.4 Summary ............................................................ 79

6 Technology Library Modeling for the purpose of the Information-driven General Functional Decomposition 81
  6.1 Introduction ........................................................ 82
  6.2 Complex gate logic features ...................................... 86
    6.2.1 Boolean function classification ............................. 87
    6.2.2 Expression tree ................................................ 91
    6.2.3 Permutation representation ................................... 92
    6.2.4 Compact minterm representation of a Boolean function .... 95
    6.2.5 Compact minterm translation ................................ 96
  6.3 CMOS gate physical features ..................................... 103
    6.3.1 Active area .................................................... 104
    6.3.2 Delay .......................................................... 110
    6.3.3 Power dissipation .............................................. 113
  6.4 Homogeneous library representation ............................ 118
    6.4.1 Full spectrum of Boolean functions ........................ 118
    6.4.2 Input/Output inversion(s) ................................... 119
    6.4.3 Virtual gates .................................................. 120
  6.5 Library parser ..................................................... 123
    6.5.1 Internal representation of logic expressions ............... 125
    6.5.2 Library pre-characterization ................................ 128
    6.5.3 Symmetry detection .......................................... 131
    6.5.4 Gate instantiation ............................................ 135
  6.6 Summary ............................................................ 140

7 Sub-function realization in the Library-based Information-driven General Functional Decomposition 143
  7.1 Multi-valued sub-function realization .......................... 143
    7.1.1 “Mechanics” of the sub-function realization ............... 144
    7.1.2 Construction and selection of the most promising realizations 145
  7.2 Multi-level sub-function decomposition ........................ 147
  7.3 Direct realization of a multiple-valued sub-function .......... 148
    7.3.1 Method ......................................................... 148
    7.3.2 Direct mapping ............................................... 150
    7.3.3 Convergent realizations ...................................... 155
    7.3.4 Transcoders .................................................. 187
  7.4 Gate-targeted multi-valued sub-function encoding ............. 192
    7.4.1 Maximal Adjacencies .......................................... 194
CONTENTS

7.4.2 Sum of Products and Product of Sums encoding ........................................ 200
7.5 Sub-function quality estimation ................................................................. 204
  7.5.1 Quality assessment .............................................................................. 204
  7.5.2 Comparison of different sub-function physical realizations .............. 207
7.6 Summary ..................................................................................................... 210

8 Experimental results ....................................................................................... 213
  8.1 Measured and compared circuit characteristics ........................................ 214
  8.2 Examples of circuits synthesized with IRMA2GATES ............................ 215
  8.3 Single output functions of MCNC benchmark suite ............................... 233
  8.4 Comparison on symmetric functions ....................................................... 238
  8.5 Comparison on incompletely specified functions ................................. 239
  8.6 Conclusions ............................................................................................... 240

9 Conclusion ....................................................................................................... 243

A Results of experiments .................................................................................... 259
  A.1 Generated single output symmetric functions ....................................... 259

B STDCell technology library ............................................................................. 261
List of definitions

2.1 Compatibility relation .................................................. 10
2.2 Compatible block $CB_\simeq(B)$ ................................. 11
2.3 Set system $\pi_\simeq$ .................................................. 11
2.4 $\leq$ operator defined on set systems ...................... 12
2.5 Maximal compatible block $MB_\simeq(B)$ ................... 12
2.6 Canonical representative of $\simeq$ ......................... 12
2.7 Set system product .................................................. 12
2.8 Boolean algebra ....................................................... 12
2.9 Minterm ................................................................. 13
2.10 Completely specified Boolean function .................. 14
2.11 Incompletely specified Boolean function ............... 14
2.12 Symmetric Boolean function ..................................... 15
2.13 Group symmetry ...................................................... 15
2.14 Hierarchical symmetry ............................................. 16
2.15 Rotational symmetry ................................................ 16
2.16 NPN equivalence ...................................................... 17
2.17 Factored form ......................................................... 18
2.18 Binary decision diagram ............................................ 19
2.19 Terminal suppressed binary decision diagram .......... 19
2.20 Occurrence multiplicity of elementary information .. 21
2.21 Basic information relationships ......................... 23
2.22 Basic abstraction relationships ................................ 24
2.23 Information quantity ................................................ 26
2.24 Abstraction quantity ................................................ 26
2.25 Information measures .............................................. 26
2.26 Abstraction measures .............................................. 27
2.27 Weighted information quantity measure ............... 29
2.28 Weighted information similarity measure .......... 29
2.29 Support of elementary information item ............... 30
3.1 Completely specified combinational machine ............. 35
3.2 Completely specified multiple i/o combinational machine 35
3.3 Incompletely specified multiple i/o combinational machine 36
3.4 Symbol covering ..................................................... 36
3.5 Realization of incompletely specified combinational machine 36
3.6 General composition ................................................ 37
3.7 General composition machine ................................. 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>Input-output (I-O) partition pair</td>
<td>39</td>
</tr>
<tr>
<td>3.9</td>
<td>&quot;General&quot; composition without loops</td>
<td>41</td>
</tr>
<tr>
<td>3.10</td>
<td>&quot;General&quot; composition machine without loops</td>
<td>41</td>
</tr>
<tr>
<td>3.11</td>
<td>&quot;General&quot; decomposition without loops</td>
<td>42</td>
</tr>
<tr>
<td>3.12</td>
<td>Input-output (I-O) set-system pair</td>
<td>45</td>
</tr>
<tr>
<td>4.1</td>
<td>n-tuple</td>
<td>66</td>
</tr>
<tr>
<td>6.1</td>
<td>CMOS complex gate</td>
<td>82</td>
</tr>
<tr>
<td>6.2</td>
<td>NPN equivalence class</td>
<td>88</td>
</tr>
<tr>
<td>6.3</td>
<td>Compact minterm representation</td>
<td>95</td>
</tr>
<tr>
<td>6.4</td>
<td>Formula tree</td>
<td>125</td>
</tr>
<tr>
<td>7.1</td>
<td>Odd- and even-size cycle</td>
<td>161</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The subject of this dissertation is information-driven circuit synthesis for implementation with (CMOS) logic gates.

Section 1.1 places the topic in the context of the overall digital circuit design process, and shows the importance of the circuit synthesis step in the design flow.

In Section 1.3, the demand for new methods of circuit synthesis dedicated to semi-custom ASIC design is explained. The subject, motivation, aim and main contributions of the research performed are described.

The final section of this chapter describes the organization of this thesis.

1.1 Digital circuit design process

A number of different design styles, often called methodologies, have been used for micro-electronic digital circuits design. They are usually sub-divided into custom and semi-custom styles. In the case of custom style, the circuit design is actually fully custom down to the physical level, requiring an extensive effort to optimize each detailed circuit’s feature. Consequently, the effort and cost of custom design are high and the investment has to be compensated by high-quality circuits and/or a large volume production, as in the case of circuits for special applications imposing stringent requirements or large volume general purpose processor or communications circuits.

Semi-custom design is based on the concept of restricting the design freedom to a limited number of circuit primitives, and limited fine-tuning possibilities for most parts of a circuit design. The restriction allows the designer to reuse the well-designed and well-characterized primitives well applicable in many situations, and to focus on their adequate usage and composition. The substantial reduction of the possible number of circuit design choices makes it easier to develop CAD tools for circuit design and optimization, as well as, reduces the design effort and time. The related loss in quality, due to the usage of standard elements, is often small, because fine-tuning a custom design may be extremely difficult, while the automated optimization techniques for semi-custom styles can explore a much wider space of implementation choices than a designer team can afford. Moreover, the standard
elements may be pre-designed in several versions optimized for different objectives related to circuit area, speed or power consumption. Today the number of semi-custom designs is much higher than that of custom designs.

The circuit design process consists of a number of stages. Usually, the circuit is first specified in a Hardware Description Language (HDL), such as Verilog or VHDL. The human readable HDL format is first parsed and translated into a machine readable representation appropriate for further (semi-)automatic synthesis. In result, a symbolic representation of the system is constructed. This step is performed by the HDL-compiler. Subsequently, the symbolic representation is transformed into an effective binary representation. In current compilers, this representation is in the form of an initial generic logic netlist. In most cases, it defines a system of binary functions to be implemented. As the following step, a technology independent logic synthesis step is usually performed. It optimizes the binary representation created in the previous step. In this step various alternative function representations are used, including specific logic networks, BDDs, binary relations, binary function tables, or Boolean formulas. The final circuit synthesis, due to the fact that is performed in a close relation to the target technology, is called technology mapping. Target technology specifies the library of logic primitives that are available in the desired design style. Hence, the mapping phase translates the function representation, usually a logic network being the result of the previous step, into a network of primitives from the technology library. At this stage the crucial features of the design are fixed, or at least can be computed with a very high precision. The total foot-print area of active elements is a sum of all active areas of components. Finally, the placement and routing phase is performed to translate the network of interconnected library components, onto a two-dimensional layout of components on a silicone die.

The circuit synthesis plays a crucial role in the design process. The properties of the final result heavily depend on the selected circuit design. The design decisions are guided by the optimization objectives and due to a complex nature of the optimization targets, such as a circuit area, performance or power dissipation, it is not easy to find the best trade-off between them.

1.2 Rationale

Introduction of the nano CMOS technologies created new opportunities, as well as unusual complexity, and particularly: extremely high device and interconnect densities, extremely small devices’ dimensions, and huge length of interconnects. Due to this complexity, interconnect scalability problems, power supply reduction and very high operating frequencies, many previously negligible phenomena have now a great impact on the circuit correctness and other quality aspects. This results in many new difficult to solve issues, including: power and energy crisis, increased leakage power, interconnect scalability problems and dominating influence of interconnects on major physical circuit characteristics (e.g. area, speed, power dissipation), etc. [126]. Unfortunately, the available circuit synthesis methods and tools do not address well the needs of circuit synthesis for the modern technologies, due to: not accounting for the recently changed importance relationships among various circuit characteristics, not accounting explicitly for timing and power and using some proxy attributes for area
that often do not correlate well with the actual area, not applying, the now necessary, multi-objective circuit optimization and trade-off exploitation and being not effective for many classes of circuits due to making many prior assumptions excluding many possible circuit structures [26, 49, 56].

In result, the proxy synthesis targets of the available logic synthesis methods and tools differ a lot from the actual synthesis targets of circuits implemented in modern technologies. In consequence, a substantial post synthesis technology mapping effort is required. Unfortunately, the technology mapping cannot guarantee proper final results, because the initial circuit synthesis is performed without close relation to the actual synthesis target.

From the above it should be clear that for the modern circuit implementation technologies a new, much more adequate circuit synthesis technology is needed that will enable the following:

- consideration of all possible circuit implementation structures during the synthesis,
- direct synthesis into specific technology targets,
- synthesis of robust, more regular circuits with minimized interconnects,
- explicitly accounting for the actual area, timing and power related information,
- performing the total multi-objective optimization of the circuit's quality and effective trade-off exploitation among the different objectives.

According to our knowledge such a circuit synthesis technology did not exist till now. None of the commercial circuit synthesis tools or published methods has the above features.

A new family of synthesis methods based on the general functional decomposition, that satisfies the above requirements, has been proposed. Jóźwiak proposed a new information-driven approach to circuit synthesis and formulated two theories that support this approach:

- theory of general decomposition of discrete functions and sequential machines [49], and
- theory of information relationships and information relationship measures [50, 51].

Based on this, a new theoretical and methodological framework has been developed for analysis and synthesis of combinational and sequential logic circuits. The framework consists of the information-driven approach to circuit synthesis, theory of general decomposition and the information modeling and analysis apparatus based on information relationships and measures. The framework was successfully applied by Jóźwiak and his collaborators to a variety of problems, including combinational synthesis targeted for LUT technologies and complex-gates libraries [55, 61, 64, 65, 67, 68, 104, 109].

The decompositional approach, and in particular the general decomposition approach proposed by Jóźwiak [49], has the potential of directly building such complex multi-level networks of logic functions, when accounting for the circuit area, speed
and power related information and directly controlling the number and length of interconnections. The combinational synthesis methods based on the general decomposition significantly outperform other methods used in the known academic and commercial tools, demonstrating the effectiveness of this approach [61].

Encouraged by the success of the decompositional synthesis methods for LUT-based technologies, we decided to research the information-driven decompositional approach to synthesis targeted for technologies, based on gates, and particularly using gates libraries.

1.3 Subject

The subject of this thesis is the circuit synthesis for implementation with (CMOS) logic gates and specifically circuit adequate for the modern (nanometer) CMOS technologies. The traditional circuit synthesis methods implemented in today’s automatic logic synthesis tools deal with only some special cases of possible circuit structures and are not well adjusted to the modern technologies and synthesis targets.

Modern micro-electronic technology enables building of large circuits and systems, and offers a great diversity of logic building blocks, while the traditional logic synthesis methods are basically targeted to AND-OR-NOT or MUX circuits and require a sophisticated post-synthesis technology mapping for other targets. However, even the most sophisticated technology mapping cannot guarantee proper final results, if the initial logic synthesis is performed without a good relation to the actual target.

Main advantages of the new information-driven circuit synthesis technology based on the general functional decomposition are the following:

- generality and high flexibility: accounting for all possible circuit realization structures and trade-offs among the circuit area, power consumption and speed,
- direct synthesis into the technology primitives of a given circuit implementation technology (e.g. gates of a given technology library),
- very effective and efficient processing of incompletely specified functions,
- minimization of the number and length of interconnections,
- simplicity and regularity of the circuit structures synthesized,
- enhanced route-ability, low usage of resources, high-speed and low power consumption resulting from the circuits compactness, regularity, and minimized interconnects,
- efficient direct collaboration with physical synthesis, due to the natural ability to directly account for the timing and/or power related information from placement and/or routing.

This all contributes to the superior circuit quality represented by speed, area and power compared to the circuits from the traditional circuit synthesis technologies.
The circuit synthesis presented in this thesis is not divided into the technology independent logic synthesis and technology mapping, but is directly performed into the primitives of a given implementation technology (e.g., gates of a given technology library). An adequate technology library model is one of the input data to the circuit synthesis tool implementing our new information-driven circuit synthesis method. From this library model our tool automatically extracts all the functional and physical information required for the multi-objective circuit synthesis. Information relationships and measures make it possible to control the circuit convergence, compactness and interconnections. Our approach minimizes both the number and length of interconnections and explicitly accounts for the circuit (active) area, timing and power consumption. Since in parallel to information relationships and measures any sort of additional information can be accounted for (as e.g., related to the signal timing or activity) the timing and power driven synthesis, as well as very flexible and precise delay, power and area trade-offs are possible. Although the trade-offs are possible, the synthesized circuits are small, ultra-fast and at the same time low-power. This all together fulfills the requirements of an adequate circuit synthesis for the modern nano CMOS technologies as formulated in Section 1.2.

Within this project the issues of library modeling for the information-driven circuit synthesis and the multi-valued sub-function realization were mainly researched. Based on this research, a prototype circuit synthesis method targeted to (CMOS) gate libraries and corresponding EDA-tool, that considerably differ from all other known methods and tools were developed. The experimental results from the tool that implements the new method targeted to the gate-based circuit show that they both, the method and tool, deliver much better circuits than the other methods and demonstrate that the information-driven general decomposition produces circuits are very fast and compact at the same time.

1.4 Aim

The main objective of the research reported in this thesis was to demonstrate that the information-driven bottom-up general decomposition based on information relationships and measures can efficiently produce gate-based circuits of high quality as represented by speed, area and power consumption. To demonstrate this, it was necessary to develop an adequate circuit synthesis method, implement the corresponding EDA synthesis tool and perform the related experimental research.

The single-step direct circuit synthesis process requires availability of adequately complete and accurate information on the logical and physical features of the technology gates. The corresponding data structures must enable an accurate modeling of this information, effective search for gates during the sub-function construction process, and efficient application of the selected gates in the network under construction. In particular, the single-step circuit synthesis requires:

- an adequate characterization of the gates’ logic features according to different Boolean function classes and physical features related to area, timing and power dissipation,

- a methodology to efficiently provide correspondence between the representa-
tions of multi-valued sub-functions constituted during the decomposition process and the functional representation used for characterization of physical gates from a given technology library.

The completion of the main objective depends on the successful execution of these two partial tasks.

1.5 Main contribution

General decomposition theorem provides framework of methodology to construct functionally correct combinational circuits. It does not, though, provide any procedure for finding systems of partition that produce good decompositions for some objectives, e.g. fast, low area etc. In this thesis, the information-driven circuit synthesis approach is presented that relies on the analysis of the information flow and technology library model. The topic of this thesis is Information-driven General Functional Decomposition targeted to Gate Libraries. The research documented in this thesis is based on the prior research performed by Jóźwiak and Volf documented in [133], continued by Chojnacki and described in [20]. The contribution of this thesis is adaptation of this methodology for completely different technology target. This adaptation is the first important contribution presented in here.

Second contribution, equally important, is development of the technology library model. To the best of our knowledge, up to the moment of finalization of the project presented in this thesis, no one utilized characterization of technology library for the purpose of single step decomposition approach. The method of technology library modeling presented in Chapter 6 is novel and of original creation of the author of this thesis. The novelty is in the fact that the logic synthesis and technology mapping dual-step approach was replaced with a single step synthesis/mapping approach.

Information-driven functional decomposition requires a framework of algorithms and control strategies to steer the process of synthesis. The main contribution of this work in this respect is the development of the effective and efficient heuristics that control decomposition process in such a way that the resulting logic gate networks are fast (critical path is close to minimum) and compact (contain a small number of logic blocks and interconnections).

The main contribution can be split up into the following:

- adaptation of general functional decomposition (Chapter 4),
- an adequate characterization of the gates’ logic features according to different Boolean function classes and physical features related to area, timing and power dissipation (Chapter 6),
- a methodology to efficiently provide correspondence between the representations of multi-valued sub-functions constituted during the decomposition process and the functional representation used for characterization of physical gates from a given technology library (Section 7.3),
- proposing a novel method of finding a mapping of Boolean function directly in a single step of decomposition (Section 7.4).
1.6. THESIS OUTLINE

Only a successful completion of all said parts would have proven the applicability of information relationships and measures theory to the functional decomposition of combinatorial Boolean functions. This fact was shown and discussed in Chapter 8.

1.6 Thesis outline

Chapters 2 and 3 introduce the background knowledge necessary for understanding of this thesis.

In Chapter 2, the necessary theoretical background related to Boolean functions, as well as, information modeling and analysis is introduced. Based on this background, in Chapter 3 the theory of general decomposition is presented. It is the fundamental method adapted for the purpose of decomposition targeted to complex gates libraries. To the best knowledge of the author of this thesis this was never performed to this day. The changes and modifications required by gate libraries specifics are also presented in further details in the following chapters.

In Chapter 4, the information-driven functional decomposition, with an emphasis to gate based technologies target is presented. In this chapter the functional decomposition of combinatorial Boolean functions is presented, with its specific application to circuits for ASIC technology as target of this thesis. Problem formulation can be found there, as well as the description of the solution suggested in this thesis. Special attention is given to the precise formulation of the problem, for which the solution was found during the course of the research presented in this thesis. Further, the usage of the information measures to control the process of decomposition is presented. This connection is presented in a form of an algorithm framework.

Chapter 5 presents the background of research presented in this thesis, mainly the research related to similar methodologies and theories of functional decomposition. It also puts into a perspective the research presented in this thesis with its roots in prior research of Jóźwiak, Volf and Chojnacki.

In Chapter 6 the representation of technology library for the purpose of efficient and effective decomposition is presented. Chapter 7 shows the methods of sub-functions encoding, using the aforementioned representation of technology library. The direct method is presented in details in Section 7.3, while encoding methods in Sections 7.4.1 and 7.4.2. In the next chapter, it is shown that the circuits resulting from the decomposition process performed with IRMA2GATES are in majority of cases smaller and faster than those resulting from other (reference) tool(s). To back up the claim, the results of the experiments with IRMA2GATES tool are discussed in Chapter 8.

The summary of the research results concludes this thesis in Chapter 9. The appendix contains the reference the technology library STDCell used extensively throughout this thesis in examples and comparison experiments.
Chapter 2

Basic definitions

Digital circuits can be classified into two main classes: combinational and sequential. This thesis is devoted to combinational circuit synthesis. In a combinational circuit, the output value only depends on the current input value. The relation between the input values and corresponding output values can be defined in a form of an incompletely or completely specified Boolean function.

This chapter introduces basic concepts of Boolean function and combinational circuit modeling and analysis. Boolean functions and combinational circuits can be considered as information processing systems that combine and abstract information provided to their inputs, represent it in an appropriate way and feed it to their outputs. The analysis of the information and information flows between the inputs, outputs sub-functions and sub-circuits is the basis of the sub-function and the sub-circuit construction and encoding methods presented in this thesis. To represent the information and information flows, we use the set systems and information sets. To express and quantitize the relationships among different information flows, we use information relationships and information relationship measures defined on the corresponding set systems and information sets. The analysis of the information relationships among different information flows in Boolean functions and networks and the implications of the analysis results in an effective sub-function realization, and in consequence, an effective and efficient multiple-level gate network construction is among the key contributions of this work.

2.1 Preliminary definitions

The concepts of partitions, set systems and covers, play a central role in the information analysis apparatus [50, 51, 57, 58]. They are used extensively in this thesis, and are used as representation of information in the modeled circuits. These concepts were used under various names by various researchers, e.g. in [11, 12, 39]. This section presents the nomenclature introduced in [72].
2.2 Set systems

The core concept of the general decomposition theory presented in the Section 3.2 is the notion of set system being a generalization of a notion of partition first presented in [39] by Hartmanis and Stearns. The notion of set system, contrary to partitions, can be used to model incompletely specified functions. This feature is necessary for effective synthesis of complex multiple-level circuits.

Unfortunately, the theory of Hartmanis and Stearns does not ensure a canonical representation of set systems and also has some inconsistent properties [133] with respect to the information relationship theory presented in Section 2.6. Therefore, Volf reformulated the set system theory for LUT synthesis purposes in [133]. In this section the formulation of the set system theory presented in [133] is recalled. Only concepts required for the explanation of the proposed circuit synthesis method will be presented. The interested reader is referred to [39] and [133] for proofs and details of this theory.

Definition 2.1 (Compatibility relation). Let $S$ be a finite non-empty set of symbols. A binary relation $≃$ is called a compatibility relation on $S$ if and only if it is reflexive and symmetric, i.e.:

$$\forall (s, s) \in S \otimes S$$

such that

$$\forall (s, s) \in S \otimes S$$

reflexive property (2.2.2)

and

$$\forall (t, s) \in S \otimes S$$

symmetry property (2.2.3)

Two symbols complying to the compatibility relation will be referred as compatible pair. The notation $a ≃ b$ will be used to denote that $(a, b) \in ≃$ and $(b, a) \in ≃$. The set of all compatible pairs of a certain compatibility relation $≃$ is called a compatibility set.

Two symbols that are not compatible are called an incompatible pair, and the set of all incompatible pairs is called the incompatibility set. The incompatible pair of symbols $s_i, s_j \in S$ is denoted as $s_i \mid s_j$.

Ex. 2.2.1 (A compatibility graph). Let $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$, and $≃ = \{(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (4, 7), (4, 8), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (5, 6), (5, 7), (5, 8), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6), (6, 7), (6, 8), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (7, 7), (7, 8), (8, 1), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (8, 8)\}$. The compatibility relation $≃$ can be represented as a compatibility graph (see Figure 2.1). Each vertex in such a graph is associated with a symbol. There is an edge between the two vertices of that graph if the related symbols constitute a compatibility pair in $≃$. Edges standing for pairs $(s, s)$ are not shown in this graph.

For simplicity we do not explicitly include pairs $(s, s)$ and only one pair of symbols $(s, v)$ from $(s, v)(v, s)$ is shown. Therefore, the compatibility relation in Example 2.2.1 becomes as follows: $≃ = \{(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (4, 7), (4, 8), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (5, 6), (5, 7), (5, 8), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6), (6, 7), (6, 8), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (7, 7), (7, 8), (8, 1), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (8, 8)\}$. The compatibility graph in Figure 2.1 reflects simplified representation.
2.2. SET SYSTEMS

Figure 2.1: A compatibility graph.

**Definition 2.2 (Compatible block \( CB_\simeq (B) \)).** A set \( B \subseteq S \) is called a compatible block with respect to a compatibility relation \( \simeq \subseteq S \otimes S \) (denoted as \( CB_\simeq (B) \)) if and only if each pair of symbols in \( B \) is compatible:

\[
CB_\simeq (B) \iff \forall_{s,t \in B} s \simeq t \quad (2.2.4)
\]

**Definition 2.3 (Set system \( \pi_\simeq \)).** Let \( S \) be a finite non-empty set of elements. A set system \( \pi_\simeq \) is a representation for a compatibility relation \( \simeq \) defined on \( S \) if and only if:

\[
\pi_\simeq \subseteq 2^S \quad (2.2.5)
\]

such that:

1. The subsets are compatible:

\[
\forall_{B \in \pi_\simeq} CB_\simeq (B) \quad (2.2.6)
\]

2. The set system is non redundant:

\[
\forall_{B,B' \in \pi_\simeq} B \subseteq B' \Rightarrow B = B' \quad (2.2.7)
\]

3. All pairs of \( \simeq \) are present:

\[
\forall_{(s,t) \in \simeq} \exists_{B \in \pi_\simeq} \{s,t\} \subseteq B \quad (2.2.8)
\]

The conventional notation [39], also used throughout this thesis, is similar to that for partitions: the blocks, the subsets of set systems are overlined and the symbols in the blocks are comma separated. Semicolons are used to separate blocks. If there
is no doubt as to which compatibility relation the set system represents, then \( \pi \) is used instead of \( \pi_\asymp \).

The zero set system, denoted by \( \pi(0) \), is a set system containing each element of \( S \) in a separate block.

The identity set system, denoted by \( \pi(I) \), is a set system containing all the elements of \( S \) in one block.

In [133] Volf formulated the following definition of the relationship between partial order operator of set systems and the inclusion operator of the corresponding compatibility relations.

**Definition 2.4 (\( \leq \) operator defined on set systems).** Let \( S \) be a finite non-empty set of symbols and let \( \asymp \subseteq S \otimes S \) and \( \asymp' \subseteq S \otimes S \) be two compatibility relations defined on \( S \). Set system \( \pi_\asymp \in \text{SetSys}(\asymp) \) is called smaller or equal to \( \pi_\asymp' \in \text{SetSys}(\asymp') \), written as \( \pi_\asymp \leq \pi_\asymp' \), if and only if \( \asymp \subseteq \asymp' \).

**Definition 2.5 (Maximal compatible block \( MB_\asymp(B) \)).** A set \( B \subseteq S \) is called a maximal compatible block with respect to a compatibility relation \( \asymp \subseteq S \otimes S \) (denoted as \( MB_\asymp(B) \)) if and only if the block is compatible and no other symbol from \( S \) is compatible with all the symbols in \( B \):

\[
MB_\asymp(B) \Leftrightarrow CB_\asymp(B) \land \forall t \in (S \setminus B) \exists s \in B \; s \not\asymp t \tag{2.2.9}
\]

**Definition 2.6 (Canonical representative of \( \asymp \)).** Let \( S \) be a finite non-empty set of symbols. Canonical representative of compatibility relation \( \asymp \subseteq S \otimes S \) is defined as

\[
M_\asymp = \{ B \subseteq 2^S \mid MB_\asymp(B) \} \tag{2.2.10}
\]

**Definition 2.7 (Set system product).** Let \( \pi_\asymp \in \text{SetSys}(\asymp) \) and \( \pi_\asymp' \in \text{SetSys}(\asymp') \). The product of two set systems \( \pi_\asymp \cdot \pi_\asymp' \) is defined as follows:

\[
\pi_\asymp \cdot \pi_\asymp' = \{ B \mid \exists B_1 \in \pi_\asymp \exists B_2 \in \pi_\asymp' \; B = B_1 \cap B_2 \land \forall B_1' \in \pi_\asymp \forall B_2' \in \pi_\asymp' \; B \subseteq B_1' \cap B_2' \Rightarrow B = B_1' \cap B_2' \} \tag{2.2.11}
\]

\( \pi_\asymp \cdot \pi_\asymp' \) is a set system representing the following compatibility relation:

\[
\asymp'' = \{(s, t) \in S \otimes S \mid \exists B \in \pi_\asymp \cdot \pi_\asymp' \; \{s, t\} \in B \} \tag{2.2.12}
\]

i.e. \( \pi_\asymp \cdot \pi_\asymp' \in \text{SetSys}(\asymp'') \). Moreover, \( \pi_\asymp \cdot \pi_\asymp' \leq \pi_\asymp \) and \( \pi_\asymp \cdot \pi_\asymp' \leq \pi_\asymp' \).

### 2.3 Boolean functions and Boolean algebra

**Definition 2.8 (Boolean algebra).** A set \( B = \{0, 1\} \) and two operators is called a Boole algebra if the following conditions are met:

- **cardinality**

\[
\exists x, y \in B [x \neq y] \tag{2.3.1}
\]
• closure

\[ \forall x, y \in B \left[x \cdot y \in B \land x + y \in B\right] \quad (2.3.2) \]

• identity items

\[ \exists 1 \in B \forall x \in B \left[x \cdot 1 = x = 1 \cdot x\right] \quad \exists 0 \in B \forall x \in B \left[x + 0 = x = 0 + x\right] \quad (2.3.3) \]

• commutativity

\[ \forall x, y \in B \left[x \cdot y = y \cdot x\right] \quad \forall x, y \in B \left[x + y = y + x\right] \quad (2.3.4) \]

• complement

\[ \forall x, y \in B \exists x, y \in B \left[x + \overline{y} = 1 \land x \land \overline{y} = 0\right] \quad (2.3.5) \]

• distributivity

\[ \forall x, y, z \in B \left[x + (y \land z) = (x + y) \land (x + z)\right] \quad \forall x, y, z \in B \left[x \land (y + z) = (x \land y) + (x \land z)\right] \quad (2.3.6) \]

Binary Boolean algebra defines two binary operations usually denoted by "+" (alternative, or OR) and "\cdot" (conjunction, or AND), and one unary operator usually denoted through over-lined "\overline{y}" or an apostrophe "'y". A literal is either a variable or negated variable. \( B^n \) stands for the \( n \)-dimensional space defined on \( n \) binary Boolean variables, and referred to as cube.

**Definition 2.9 (Minterm).** A minterm of a Boolean function

\[ f : B^n \rightarrow \{0, 1, *\}^m \]

is a zero-dimension cube (point) involving the product of exactly \( n \) literals.

A minterm in \( n \)-dimensional Boolean space can be represented by binary-valued vector \( v \) of size \( n \), \( v \in \{0, 1\}^n \). To each position \( i (1 \leq i \leq n) \) in vector \( v \), there is assigned a particular binary variable \( v_i \). \( v[i] = 0 \) corresponds to literal \( \overline{v_i} \), \( v[i] = 1 \) corresponds to literal \( v_i \).

A product of \( m \) literals \( (m \leq n) \) represents a \((n-m)\)-dimensional cube sub-space called term. Term in \( n \)-dimensional Boolean space can be expressed by ternary-valued vector \( v \) of size \( n \), \( v \in \{0, 1, -\}^n \). The meaning of symbols 0 and 1 is the same as those in the case of minterm. \( v[i] = - \) denotes that variable \( v_i \) is not present in term and can take any value.

**Ex. 2.3.1.** Let us consider three-dimensional Boolean space \( \{0, 1\}^3 \) (see Figure 2.2) spanned by variables \( A, B, \) and \( C \). Each vertex in this Boolean space can be specified by a certain combination of literals and a corresponding vector, for example, by the product \( \overline{A}BC \) and corresponding vector \( \{010\} \). Any sub-space (sub-cube) can be expressed by the product of literals, for example \( A\overline{C} \), or by corresponding vector \( \{1\overline{0}\} \). A dash ("\-\") denotes the variable \( B \) can be either 0 or 1.
Definition 2.10 (Completely specified Boolean function). A completely specified \(n\)-input, \(m\)-output Boolean function \(f\) is a mapping between Boolean spaces:
\[
f : B^n \rightarrow B^m
\]  
(2.3.7)

\(m\)-output function can be considered a vector of \(m\) single-output functions defined on the same domain.

An incompletely specified Boolean function is defined over the sub-set of \(B^n\). The points where the function is not defined are called don’t care conditions, or simply don’t cares. Don’t care conditions arise when a function \(f\) is embedded in an environment (network of functions). Predecessors of \(f\) may not produce some combinations of values and then vectors related to these combinations never occur on inputs of \(f\). The set of such input vectors of \(f\) is called (local) controllability don’t care set - CDC. On the other hand, successors of \(f\) may not transmit some values of \(f\). Therefore, these values will never be observed on the network output. The set of input vectors of \(f\) related to values not being observed is called (local) observability don’t care set - ODC.

Definition 2.11 (Incompletely specified Boolean function). An incompletely specified Boolean \(n\)-input, \(m\)-output function is a mapping:
\[
\hat{f} : B^n \rightarrow \{0, 1, *\}^m
\]  
(2.3.8)

where \(*\) denotes don’t care condition.
2.3. BOOLEAN FUNCTIONS AND BOOLEAN ALGEBRA

A function decomposition process produces a network of sub-functions that realizes the behavior of the original function. As a result, the decomposition process can produce an incompletely specified sub-function, even though the original function is completely specified.

However, don't care conditions provide additional freedom for logic optimization and can be successfully exploited during decomposition.

Some Boolean functions possess property called symmetry. This class of Boolean functions is very important from the practical point of view. Many of the typical processors’ data path and transmission channels functions are symmetric, and some checking and control functions are also symmetric. On the other hand, symmetries can be successfully exploited in functional decomposition, technology mapping, and binary decision diagrams minimization. Therefore, there was much research in the field of symmetric Boolean functions [16, 24, 33, 58, 75, 78, 103, 120, 127, 137].

Definition 2.12 (Symmetric Boolean function). A single-output function \( \hat{f}(x_1, \ldots, x_n) \) is symmetric with respect to a subset \( SYM = \{x_i| i \in \{1, \ldots, n\}\} \) of all input variables if there exists a don’t care assignment \( \hat{g} \) such that \( \hat{f}(\hat{g}) \) is invariant under any permutation of variables in \( SYM \).

The subset \( SYM \) is called the symmetry set of \( \hat{f} \). \( \hat{f} \) is totally symmetric if \( \hat{f} \) is symmetric in respect to the whole set \( \{x_1, \ldots, x_n\} \). Otherwise, \( \hat{f} \) is partially symmetric. Symmetry set \( SYM \) is a maximal symmetry set if there is no other symmetry set that properly contains \( SYM \). Two disjoint symmetry sets \( SYM_1 \) and \( SYM_2 \) are compatible symmetry sets if there is a don’t care assignment \( \hat{g} \) such that \( \hat{g} \) is invariant under any permutation of variables from \( SYM_1 \) and \( SYM_2 \). If symmetry sets \( SYM_1, SYM_2, \ldots, SYM_k \) exists such that \( SYM_1 \cup SYM_2 \cup \ldots \cup SYM_k = \{x_1, \ldots, x_n\} \) (where \( SYM_i \cap SYM_j = \emptyset, i \neq j \)), \( \hat{f} \) is completely symmetric. The majority function \( f = ab + bc + ac \) and the parity function \( f = a \oplus b \oplus c \) are examples of totally symmetric functions.

The following symmetry categories can be detected using a number of specialized algorithms, to speed up decomposition process:

- groups symmetry
- hierarchical symmetry
- rotational symmetry

Definition 2.13 (Group symmetry). We say that a Boolean function \( f \in B_n \) is group-symmetric or G-symmetric if \( f \) keeps invariant under all permutations of input set \( \pi \) in input subset \( G \) [43].

There are at least two different, nonempty, and disjoint subsets of the sets of inputs of a Boolean function \( f \) that have the following property: there is a set of permutations on these subsets, so that applying the permutations simultaneously on all subsets does not change function \( f \).

Ex. 2.3.2 (Example of group symmetry). Let us consider an example Boolean function of three inputs:
2. BASIC DEFINITIONS

\[ f = A_1(x_1 \lor x_2) \lor B_1(x_3 \lor x_4) \]

Here, \( \{x_1, x_2\} \) and \( \{x_3, x_4\} \) are pairs of symmetric variables, but there is no h-symmetry between them because of the variables \( A_1 \) and \( B_1 \). However, exchanging both \( \{x_1, x_2\} \) and \( \{x_3, x_4\} \) and \( A_1, B_1 \) keeps the function invariant. Therefore, this is what we call g-symmetry between the two subsets of inputs, \( \{x_1, x_2, x_3, x_4\} \) and \( A_1, B_1 \). ■

**Definition 2.14 (Hierarchical symmetry).** Let \( f \in B_n \) be a Boolean function with the input variables \( X = \{x_1, x_2, \ldots, x_n\} \). Let \( X_1, X_2 \in X \) be two subsets of \( X \). \( X_1 \) and \( X_2 \) are hierarchical symmetric (H-symmetric) if and only if \(|X_1| = |X_2| > 1\). \( X_1 \) and \( X_2 \) are maximal symmetry group of \( 1 \), and \( f \) is \( H(X_1, X_2) \)-symmetric, where \( H(X_1, X_2) \) is equal to the subgroup of the permutation group \( \pi_n \) generated by the following set of permutations: \( \{\pi \in \pi_n | \pi(X_1) = X_2 \text{ and } \pi(X_2) = X_1\} \). A group of subsets of \( X_1, X_2, \ldots, X_k \) is H-symmetric if and only if: \( \forall i, j \in \{1, 2, \ldots, k\} : X_i \text{ is H-symmetric to } X_j \).

**Definition 2.15 (Rotational symmetry).** [129] Let \( V_n \) be the vector space of dimension \( n \) over the two element field \( \mathbb{Z}_2 \). Let \( x_i \in \{0, 1\} \) for \( 1 \leq i \leq n \). For \( 1 \leq k \leq n \), we define

\[
\rho^k_i(x_i) = \begin{cases} 
  x_{i+k} & \text{if } i + k > n, \\
  x_{i+k-n} & \text{if } i + k > n.
\end{cases}
\] (2.3.9)

Let \((x_1, x_2, \cdots, x_{n-1}, x_n) \in V_n\). Then we extend the definition

\[
\rho^k_i(x_1, x_2, \cdots, x_{n-1}, x_n) = \left(\rho^k_i(x_1), \rho^k_i(x_2), \cdots, \rho^k_i(x_{n-1}), \rho^k_i(x_n)\right). \] (2.3.11)

A Boolean function on \( n \) variables may be viewed as a mapping from \( V_n \) into \( V_1 \). A Boolean function \( f \) is referred to as rotationally symmetric if and only if

\[
\forall x_1, \ldots, x_n \in V_n, f(\rho^k_i(x_1, \cdots, x_n)) = f(x_1, \cdots, x_n), \text{ for any } 1 \leq k \leq n. \] (2.3.12)

**Ex. 2.3.3 (Example of rotational symmetry).** Let us consider an example Boolean function of three inputs:

\[
f = (x_1 \land x_2) \lor (x_1 \land x_3) \lor (x_2 \land x_3)
\]

Here, a triplet of all \( \{x_1, x_2, x_3\} \) is a set of rotational symmetric variables. Reordering of all variables, i.e. \( \{x_1, x_2, x_3\} \) into \( \{x_2, x_3, x_1\} \) yields identical Boolean function. This type of symmetry we refer to as h-symmetry on a (particular sub-) set of inputs. ■

In the experience of Mohnke et al. [95] rotational-symmetries do not appear very often in practice. E.g. for Actel library only one three-input gate has this feature. Boolean functions can be classified according to the concept of P and NPN equivalence. The set of \( n \)-input functions is classified according to three criteria: the number of functions, the number of P classes and the number of NPN classes.
2.4. REPRESENTATIONS OF BOOLEAN FUNCTIONS

Definition 2.16 (NPN equivalence). Two Boolean function \( f(x) \) and \( g(x) \) are said to belong to the same NPN equivalence class if there are:

- a complementation operator \( \pi \) (input negation),
- a permutation matrix \( \psi \) (input permutation), and
- a complementation operator \( \Phi \) (output negation),

such that the following equation is a tautology [45]:

\[
f \equiv g \iff f = (g \circ X_{\pi} \circ X_{\psi})^\Phi \quad (2.3.13)
\]

The complementation operators of inputs \( \psi \in B^n \) and output \( \Phi \in B \) specify the possible negation of some of its arguments. In other words, two functions belong to the same NPN class, and are called NPN-equivalent, when they are equivalent modulo the negation (\( N \)) of the output, the permutation (\( P \)) of the inputs and the negation (\( N \)) of the inputs. Appropriate input permutation \( \pi \in S_n \) and an appropriate phase assignment: inputs \( \psi \in B^n \) and output \( \Phi \in B \), herein

\[
X_{\pi}(x_1, ..., x_n) := (x_{\pi(1)}, ..., x_{\pi(n)}) \quad (2.3.14)
\]

\[
X_{\psi}(x_1, ..., x_n) := (x_{\psi_1}^1, ..., x_{\psi_n}^n) \quad (2.3.15)
\]

2.4 Representations of Boolean functions

There is a variety of Boolean function representations. Some more, some less suitable for the purposes of functional decomposition, such as Boolean function processing, analysis, storage, etc. Examples of a widely used forms of Boolean function representations are the following:

- tabular forms,
- logic expressions,
- binary decision diagrams,
- subject graphs.

**Tabular Form** is a table that contains input and output parts. The input part corresponds to inputs, and is represented as a vector in \( \{0, 1, -\}^n \), while output part corresponds to outputs, and is represented as a vector in \( \{0, 1, *\}^m \). Input (output) part is defined in the \( n \) (\( m \)) -dimensional Boolean space, where \( n \) (\( m \)) denotes the number of input (output) variables. Each pair of input and output vector denotes an individual function: correspondence of particular input vector to the particular output vector.

**Ex. 2.4.1.** Table 2.1 shows an 4-input, 2-output incompletely specified Boolean function \( \hat{f} : \{0, 1\}^4 \longrightarrow \{0, 1, *\}^2 \).

\[
\hat{f}_{ON}^1 = \pi cd \lor a\overline{c}, \quad \hat{f}_{OFF}^1 = \overline{a}\pi \lor cd \lor ac, \quad \hat{f}_{DC}^1 = \overline{a}\overline{cd} \lor a\overline{\pi d} \lor a\pi d.
\]
**BASIC DEFINITIONS**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>( f_1 )</th>
<th>( f_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.1: A 4-input, 2-output incompletely specified Boolean function.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>( f_1 )</th>
<th>( f_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>~</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>~</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>~</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>~</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>~</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>~</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>~</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>~</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.2: On-set/off-set specification of the function from Table 2.1.

\[
\hat{f}_2^{ON} = \overline{a} \overline{b} \overline{c} \overline{d} \lor \overline{a} \overline{c} \overline{d} \lor \overline{a} \overline{b} \overline{d} \lor \overline{a} \overline{b} \overline{c} \overline{d} \lor \overline{a} \overline{b} \overline{c} \overline{d}, \quad \hat{f}_2^{OFF} = \overline{a} \overline{b} \overline{c} \overline{d} \lor \overline{a} \overline{c} \overline{d} \lor \overline{a} \overline{b} \overline{d} \lor \overline{a} \overline{b} \overline{c} \overline{d} \lor \overline{a} \overline{b} \overline{c} \overline{d} \lor \overline{a} \overline{b} \overline{c} \overline{d}.
\]

An alternative representation can have smaller number of implicants, see Table 2.2. It specifies the on-set and off-set only. To denote a particular function for which a given implicant does not specify input-output relation, a symbol ~ is used in the output part of table. 

**Expression forms.** A single-output completely specified Boolean function can be expressed by formulas of literals, with operators used to denote primitive Boolean relations. Usually, the alternation (+) and conjunction (·) operators are used. All Boolean functions can be expressed in the sum of product of literals form and the product of sums of literals form. Parentheses are used to arbitrarily nest the form to express Multiple-level forms. A particular multiple-level form is the factored form [8].

**Definition 2.17 (Factored form).** A factored form is one and only one of the following:

1. A literal.
2. A sum of factored forms.
3. A product of factored forms.

Operator of negation recognizes a single variables only (represented by literals), and cannot be used on complex expression. Therefore, in the factored forms, \( \bar{a} \land \bar{b} \) is a valid factored form, while \( a \lor b \) is not.

A single Boolean expression cannot represent an incompletely specified Boolean function, but a single expression can characterize an on-set (off-set or dc-set) of the function (see Example 2.4.1).

**Binary Decision Diagrams.** Binary decision diagrams (BDDs) were introduced by Lee [83], later by Ackers in [2], and researched by Randal Bryant to represent single-output Boolean functions. In [9] Bryant presented algorithms for efficient BDDs manipulation.

**Definition 2.18 (Binary decision diagram).** A binary decision diagram (BDD) is a rooted directed acyclic graph (DAG) with vertex set \( V \). The graph has terminal vertices called leaves. To each leaf vertex there is assigned a value 0 or 1 that represents respectively the constant Boolean function 0 (false) and 1 (true). Each non terminal node \( v \in V \) is labeled with a Boolean variable \( x = \text{var}(v) \) and has two children, \( \text{low}(v), \text{high}(v) \in V \).

BDD represents a sequence of binary decisions, which is expressed by a vertex. The edge connecting vertex \( v \) with its \( \text{high}(v) \) is called then (or 1) edge and denotes the true value assigned to a variable \( x = \text{var}(v) \) allocated to that vertex. Respectively, the edge connecting vertex \( v \) with its \( \text{low}(v) \) is called else (or 0) edge and denotes the false value assignment to a variable \( x = \text{var}(v) \). Leaves of the diagram represent the overall value (true - leaf 1, and false - leaf 0) of the series of decisions corresponding to a path from the root vertex to a leaf.

In [111] Reis et.al. introduced a novel BDDs class: Terminal Suppressed BDDs, and addressed the problem of designing integrated circuits without using cell libraries. This task, named library free implementation, involves two main operations: library free technology mapping and library free physical synthesis. Their library free technology mapping method is based on a novel Binary Decision Diagram class allowing direct association of transistors with arcs, resulting in a direct mapping on complex gates (ANDORIs), with full control of the maximum number of serial transistors.

**Definition 2.19 (Terminal suppressed binary decision diagram).** is an un-ordered BDD obeying a set of rules:

- To terminal node (leaf) “true” only “true” edges are connected
- To terminal node (leaf) “false” only “false” edges are connected
- All edges connected to non-terminal node are the same type (either “true” or “false”)
- There is always one path that passes through all non-terminal nodes.

Please refer to Chapter 5 for more detail description of Terminal Suppressed BDD, and CMOS generation technique shown in Figure 5.1.
2.5 Information Relationship Measures

Introduction of information representation in a form of information set gives opportunity for analysis of mutual relationships between signals, variables and set systems representing information streams. Ability to evaluate information in the quantitative terms, makes it possible to measure the amount of information in information streams as a number of atomic information items, or after associating importance or weight to each particular information item to measure the importance of information streams.

To help make use of these characteristics, the theory of information relationships and measures was introduced in [50, 51], defining a.o. the following relationships between information streams:

- **common information** CI (i.e. information that is present in both \( \pi_1 \) and \( \pi_2 \)):
  \[ CI(\pi_1, \pi_2) = IS(\pi_1) \cap IS(\pi_2) \]

- **total (combined) information** TI (i.e. information that is present in either \( \pi_1 \) or \( \pi_2 \)):
  \[ TI(\pi_1, \pi_2) = IS(\pi_1) \cup IS(\pi_2) \]

- **missing information** MI (i.e. information that is present in \( \pi_1 \) but missing in \( \pi_2 \)):
  \[ MI(\pi_1, \pi_2) = IS(\pi_1) \setminus IS(\pi_2) \]

- **extra information** EI (i.e. information that is missing in \( \pi_1 \) but present in \( \pi_2 \)):
  \[ EI(\pi_1, \pi_2) = IS(\pi_2) \setminus IS(\pi_1) \]

- **different information** DI
  \[ DI(\pi_1, \pi_2) = MI(\pi_1, \pi_2) \cup EI(\pi_1, \pi_2) \]

Also for abstraction, several relationships are defined in [50] that describe common, total, missing, extra and different abstraction.

**Ex. 2.5.1 (Information measures).** Consider, for instance, the set systems modeling information flows in the combinational circuit in Figure 2.3. The information delivered by two primary inputs \( d \) is described by the set system: \( \pi_d = \{0, 1, 2, 3, 4, 5, 6\} \)

Different values of input variable \( x_d \) allows us to recognize subset \( \{0, 1\} \) from subset \( \{2, 3, 4, 5, 6\} \). With respect to symbol 0, in other words, knowing the value of \( x_d \), we

<table>
<thead>
<tr>
<th>symbol</th>
<th>inputs</th>
<th>outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2.3: Example 4-input 3-output Boolean function.
2.5. INFORMATION RELATIONSHIP MEASURES

can distinguish symbol 0 from all symbols in block \{2, 3, 4, 5, 6\}. This fact is denoted as elementary information set: \{1\, 2, 1\, 3, 1\, 4, 1\, 5, 1\, 6\}. When combined all elementary information item set related to all symbols the complete elementary information item set carried by variable \(x_d\) is determined by the corresponding information set \(IS(\pi_d) = \{0, 2, 0, 3, 0, 4, 0, 5, 0, 6, 1, 2, 1, 3, 1, 4, 1\, 5, 1\, 6\}\). The information required to calculate the output variable \(s\) is modeled by the set system \(\pi_s = \{0, 2, 5, 1, 3, 4, 6\}\) and the information set \(IS(\pi_s) = \{0, 1, 0, 3, 0, 4, 0, 6, 1, 2, 1, 5, 2, 3, 2, 4, 2, 6, 3, 5, 5, 6\}\). The common information \(CI(\pi_s, \pi_p)\), i.e. the information used by \(s\) and delivered by \(d\) is determined by \(IS(\pi_s) \cap IS(\pi_d) = \{0, 3, 0, 4, 0, 6, 1, 2, 1, 5, 2\}\). The information used by \(s\) but missing in \(d\) is described by \(MI(\pi_s, \pi_d) = IS(\pi_s) \setminus IS(\pi_d) = \{0, 1, 2, 3, 2, 4, 2, 6, 3, 5, 5, 6\}\). The extra information present on \(d\) but not used by \(s\) is \(EI(\pi_s, \pi_p) = IS(\pi_p) \setminus IS(\pi_s) = \{0, 2, 0, 5, 1, 3, 1, 4\}\).

The level of information similarity, dissimilarity, etc, can be defined as the cardinality of corresponding information or abstraction relationships expressed as the number of information (or abstraction) items. This way, the following measures can be defined:

- **information similarity** \(ISIM(\pi_1, \pi_2) = |CI(\pi_1, \pi_2)|\)
- **information difference** \(IDIS(\pi_1, \pi_2) = |DI(\pi_1, \pi_2)|\)
- **information decrease (loss)** \(IDEC(\pi_1, \pi_2) = |MI(\pi_1, \pi_2)|\)
- **information increase (growth)** \(IINC(\pi_1, \pi_2) = |EI(\pi_1, \pi_2)|\)
- **total information quantity** \(TI(\pi_1, \pi_2) = |TI(\pi_1, \pi_2)|\)

For example, the information similarity of the above mentioned variables \(p\) and \(n\) can be measured as \(|CI(\pi_d, \pi_p)| = 10\), the information loss: \(|MI(\pi_d, \pi_p)| = 4\) and the information growth: \(|EI(\pi_d, \pi_p)| = 5\).

In [50, 51] weighted measures were introduced to express a higher importance of some information items over another. The importance of information may be, for instance, related to its availability, i.e. the number of variables at which this information is present.

**Definition 2.20 (Occurrence multiplicity of elementary information).** Occurrence multiplicity \(m\) of an elementary information \(s_i | s_j\) from \(IS(o)\) in \(ISS(X)\) is defined as follows:

\[
m(s_i | s_j)_{IS(o)_{ISS(X)}} = \sum_{x \in X} occ(s_i | s_j)_{IS(o)_{IS(x)}}
\]

where:

\[
occ(s_i | s_j)_{IS(o)_{IS(x)}} = \begin{cases} 
1 : & \text{if } (s_i | s_j) \in IS(o) \cap IS(x) \\
0 : & \text{otherwise}
\end{cases}
\]

If \(m(s_i | s_j)_{IS(o)_{ISS(X)}} = 1\), \(s_i | s_j\) required by \(o\) is provided by only a single variable from \(X\), then \(s_i | s_j\) is called a unique information with respect to \(X\). Unique information is of primary importance.
For the analysis and synthesis of digital information systems, in scope described in this thesis, the analysis of information and information relationships is of primary importance. This analyses were found especially usefull to help make design decisions based on the relationships between the information in different information streams, not only in qualitative but also in quantitative terms.

2.6 Information and information relationship analysis

When designing digital information systems, the design decisions are guided by the design constraints and objectives. During the design process an appropriate measurement mechanism needs to be used to perform reliable qualitative and quantitative analysis and comparison of the design options found.

In this section we explain how the information is represented in the finite discrete systems. We also discuss an adequate measurements apparatus: the analysis apparatus of information relationships and information relationships and measures (IRM) proposed by Jóźwiak [50, 51]. The theory of information relationships and information relationship measures, together with the theory of general decomposition constitutes the theoretical basis for the circuit synthesis methods and tools being the subject of the research reported in this thesis. The theory of general decomposition defines a generator of all correct circuit structures. The apparatus of information relationships and measures allows for the intelligent selective application of this generator. The analysis and measurement results obtained from the apparatus of information relationships and measures are used to control the generator to construct the rapidly convergent small and quick circuits for particular outputs and to effectively re-use common sub-functions.

In Section 2.6.1, the concepts of elementary information item and elementary abstraction item are introduced, information and abstraction modeling with set-systems [39] is explained. In Section 2.6.2 we explain the usage of the concept of elementary information for analysis of information relationships between different information streams. Section 2.6.3 discusses the quantitative measurements of these relationships. It describes simple quantitative measures, normalized measures and measures for modeling of information importance.

2.6.1 Information representation in discrete systems

Let us consider a certain finite set of elements \( S \) called symbols. \( S \) can be a finite set of any elements. Information about symbols (elements of \( S \)) means the ability to distinguish certain symbols from other symbols. If a system is provided with information that distinguishes among certain subsets of symbols from \( S \), it can base its decisions on this information. In particular, for each of the subsets, it can compute a different output.

Basically, information is represented in digital systems by values of some discrete signals (in most of today’s systems, these are binary signals). Therefore, distinguishing between the elements from a certain set \( S \) is realized by different values of certain signals for various subsets of \( S \), by different values of variables that represent
the signals or abstract variables that represent some combinations of simpler variables. Information on elements from $S$ is thus typically represented by some subsets of elements from $S$, so that if a certain value of a certain signal or variable $x$ is known, it is possible to distinguish a certain subset $B$ of elements from $S$ from other subsets or from all other elements of $S$; but it is impossible to distinguish among the elements from $B$.

*Elementary information* describes the ability to distinguish a certain single symbol $s_i$ from another single symbol $s_j$, where: $s_i, s_j \in S$ and $s_i \neq s_j$.

$$I^S = \{(s_i, s_j)\} \quad \text{if } s_i \text{ is distinguished from } s_j \text{ by the modeled information.} \quad (2.6.1)$$

*Elementary information set* describes any set of atomic information, that can be represented by an information set $IS$ defined on $S \times S$ as follows:

$$IS^S = \{(s_i, s_j), \ldots\} \quad \text{if } s_i \text{ is distinguished from } s_j \text{ by the modeled information.} \quad (2.6.2)$$

A short-hand notation of elementary information item is used interchangeably throughout this thesis: instead of $(s_1, s_2)$, we write $s_1|s_2$.

*Elementary abstraction* describes the inability to distinguish a certain single symbol $s_i$ from another single symbol $s_j$. Any set of such atomic portions of abstraction can be represented by an abstraction (compatibility) relation $A$ or abstraction set $AS$ defined on $S \otimes S$ as follows:

$$A^S = \{(s_i, s_j)\} \quad \text{if } s_i \text{ is not distinguished from } s_j \text{ by the modeled information,} \quad (2.6.3)$$

$$AS^S = \{(s_i, s_j), \ldots\} \quad \text{if } s_i \text{ is not distinguished from } s_j \text{ by the modeled information.} \quad (2.6.4)$$

### 2.6.2 Information relationships

During the design of digital information system, a set of tools that describes the relationships between information streams is required. We may ask: what information necessary for computing certain outputs is present in given inputs? What information is missing? What inputs provide the missing information? For instance, the relation between information required to compute a certain output and information provided by the inputs of the machine needs to be established, i.e. what information is provided by which inputs. In this section a number of information relationships are recalled from [20, 50, 51].

The following relationships are specified between two information sets $IS_1$ and $IS_2$ defined on $S \otimes S$ (and between their corresponding set-systems and variables) [20]:

**Definition 2.21 (Basic information relationships).** Common information; i.e., information that is present in both $IS_1$ and $IS_2$:

$$CI(IS_1, IS_2) = IS_1 \cap IS_2 \quad (2.6.5)$$
Total (combined) information; i.e., information that is present either in $I S_1$ or $I S_2$:

$$TI(IS_1, IS_2) = IS_1 \cup IS_2 \quad (2.6.6)$$

Missing information; i.e., information that is present in $I S_1$, but missing in $I S_2$:

$$MI(IS_1, IS_2) = IS_1 \setminus IS_2 \quad (2.6.7)$$

Extra information; i.e., information that is missing in $I S_1$, but present in $I S_2$:

$$EI(IS_1, IS_2) = IS_2 \setminus IS_1 \quad (2.6.8)$$

Different information; i.e., information that is present in one of the information set and missing in the other:

$$DI(IS_1, IS_2) = IS_1 \oplus IS_2.$$ (2.6.9)

Analogous relationships can be defined for the abstraction of two abstraction sets $A S_1$ and $A S_2$.

**Definition 2.22 (Basic abstraction relationships).** Common abstraction:

$$CA(A S_1, A S_2) = A S_1 \cap A S_2 \quad (2.6.10)$$

Total (combined) abstraction:

$$TA(A S_1, A S_2) = A S_1 \cup A S_2 \quad (2.6.11)$$

Missing abstraction:

$$MA(A S_1, A S_2) = A S_1 \setminus A S_2 \quad (2.6.12)$$

Extra abstraction:

$$EA(A S_1, A S_2) = A S_2 \setminus A S_1 \quad (2.6.13)$$

Different abstraction:

$$DA(A S_1, A S_2) = A S_1 \oplus A S_2.$$ (2.6.14)

If $I S(\pi_1), A S(\pi_1)$ are induced by set-system $\pi_1$ and $I S(\pi_2), A S(\pi_2)$ are induced by set-system $\pi_2$ (both defined on a set of symbols $S$), we can derive the properties describing the pairwise correlation between the information and abstraction. This correlation is implied by the fact that $I S(\pi_1) \cup A S(\pi_1) = I S(\pi_2) \cup A S(\pi_2)$ and the fact that $I S(\pi_1) \cap A S(\pi_1) = I S(\pi_2) \cap A S(\pi_2) = \emptyset$.

$$MI(IS(\pi_1), IS(\pi_2)) = IS(\pi_1) \setminus IS(\pi_2)$$

$$= \left( (I S(\pi_2) \cup A S(\pi_2)) \setminus A S(\pi_1) \right) \setminus IS(\pi_2)$$

$$= \left( I S(\pi_2) \setminus AS(\pi_1) \right) \cup \left( AS(\pi_2) \setminus IS(\pi_1) \right) \setminus IS(\pi_2)$$

$$= AS(\pi_2) \setminus AS(\pi_1)$$

$$= EA(AS(\pi_1), AS(\pi_2)) \quad (2.6.15)$$
2.6. INFORMATION AND INFORMATION RELATIONSHIP ANALYSIS

Analogously:

\[ EI(\IS(x_1), IS(x_2)) = MA(\AS(x_1), AS(x_2)) \]  

(2.6.16)

For simplicity sake, from now on, we will use a simplified notation to denote common information, e.g. common information induced by two variables \( x \) and \( y \) \( CI(\IS(x_1), IS(y_2)) \) is written as \( CI(x, y) \).

<table>
<thead>
<tr>
<th>( y )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( y_1 )</th>
<th>( y_2 )</th>
<th>( y_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>{0,1}</td>
<td>{0,1}</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>{0,1}</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>{0,1}</td>
<td>{0,1}</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>{0,1}</td>
<td>1</td>
<td>{0,1}</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>{0,1}</td>
<td>{0,1}</td>
<td>{0,1}</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: An incompletely specified multiple-output Boolean function.

Ex. 2.6.1. Table 2.3 presents an incompletely specified multiple-output Boolean function. To express a collection of possible values i.e., \{0,1\}, we use also a short hand notation using ‘–’ (a hyphen). Inputs symbol set \( 1,\ldots,6 \) represent set of input subspaces on input variables \( x_1,\ldots, x_4 \). Value 0 of input variable \( x_1 \) allows us to recognize subset \( \{1,2,5\} \), while value 1, enables us to recognize subset \( \{3,4,6\} \). In other words, knowing that \( x_1 = 0 \), we know that either symbol 1, 2 or 5 occurred on inputs, but we do not know which of the two symbols occurred. Similarly, knowing that \( x_1 = 1 \), we know that either symbol 3, 4 or 6 occurred, but we do not know which of the three symbols occurred, either. In this case we say that variable \( x_1 \) provides information about (or is able to distinguish) the following pairs of symbols: \( \{1,3\}, \{1,4\}, \{1,6\}, \{2,3\}, \{2,4\}, \{2,6\}, \{3,5\}, \{4,5\} \) and \( \{5,6\} \). On the other hand, variable \( x_1 \) abstracts information about (or is not able to distinguish) the following pairs of symbols: \( \{1,2\}, \{1,5\}, \{2,5\}, \{3,4\}, \{3,6\}, \) and \( \{4,6\} \).

Set-systems induced by input and output variables of the Boolean function from Table 2.3 are the following:

\[
\begin{align*}
\pi_{x_1} &= \{1,2,5,6:3,4,6\} \\
\pi_{x_2} &= \{2,6:1,3,4,5\} \\
\pi_{x_3} &= \{3,5:1,2,4,6\} \\
\pi_{x_4} &= \{1,3,4,5,6:2,3,4,5,6\}
\end{align*}
\]

\[
\begin{align*}
\pi_{y_1} &= \{1,2,3,4,6:1,3,4,5,6\} \\
\pi_{y_2} &= \{1,2,5,6:1,3,4,5,6\} \\
\pi_{y_3} &= \{3,4,5,6:1,2,6\}
\end{align*}
\]

Below the common information of input variable \( x_1,\ldots, x_4 \) and output variable \( y_3 \) is shown.

\[
\begin{align*}
CI(\pi_{x_1}, \pi_{y_3}) &= \{1,4,1,5,2,3,2,4\} \\
CI(\pi_{x_2}, \pi_{y_3}) &= \{2,3,2,4,2,5\} \\
CI(\pi_{x_3}, \pi_{y_3}) &= \{1,3,1,5,2,3,2,5\} \\
CI(\pi_{x_4}, \pi_{y_3}) &= \emptyset
\end{align*}
\]
Variable \( x_4 \) does not provide any information that is necessary to compute the output function of \( y_3 \). This fact is indicated by an empty set of common information \( CI(\pi_{x_1}, \pi_{y_2}) \). Each input provide at least one information item that is not present on any other input variable. Information \( 1|3 \) is provided only by variable \( x_3 \), information \( 1|4 \) is provided only by variable \( x_1 \), and information \( 2|4 \) is provided only by variable \( x_2 \). As a consequence, the minimal set of variables contains all three input variables \( \{x_1, x_2, x_3\} \). All information provided by the minimal set of variables follows:

\[
IS(\pi_{x_1}) \cup IS(\pi_{x_2}) \cup IS(\pi_{x_3}) = \{1|2, 1|3, 1|4, 1|5, 1|6, 2|3, 2|4, 2|5, 2|6, 3|4, 3|5, 3|6, 4|5, 4|6, 5|6\} \supseteq IS(y_3)
\]

Analogously, we can analyze common abstraction, missing information etc.:

\[
\begin{align*}
CA(\pi_{x_1}, \pi_{y_3}) &= \{(1, 2), (3, 4), (3, 6), (4, 6)\} \\
CA(\pi_{x_2}, \pi_{y_3}) &= \{(2, 6), (3, 4), (4, 5), (4, 6)\} \\
CA(\pi_{x_3}, \pi_{y_3}) &= \{(1, 2), (1, 6), (2, 6), (3, 5), (4, 6)\} \\
CA(\pi_{x_4}, \pi_{y_3}) &= \{(1, 6), (2, 6), (3, 4), (3, 5), (3, 6), (4, 5), (5, 6)\}
\end{align*}
\]

\[
\begin{align*}
MI(\pi_{x_1}, \pi_{y_3}) &= \{(1, 5), (2, 5)\} \\
MI(\pi_{x_2}, \pi_{y_3}) &= \{(1, 3), (1, 4), (1, 5)\} \\
MI(\pi_{x_3}, \pi_{y_3}) &= \{(1, 4), (2, 4)\} \\
MI(\pi_{x_4}, \pi_{y_3}) &= \{(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5)\}
\end{align*}
\]

2.6.3 Information relationship measures

The basic information (abstraction) measure is defined as a number of elements in an information (abstraction) set.

**Definition 2.23 (Information quantity).** Let \( IS \) be an information set. Information quantity \( IQ \) provided by \( IS \) is defined as:

\[
IQ(IS) = |IS| \tag{2.6.17}
\]

**Definition 2.24 (Abstraction quantity).** Let \( AS \) be an abstraction set. Abstraction quantity \( AQ \) provided by \( AS \) is defined as:

\[
AQ(AS) = |AS| \tag{2.6.18}
\]

**Definition 2.25 (Information measures).** Let us combine the above definitions with relationships defined in Section 2.6.2. The following measures are defined for two information sets \( IS_1 \) and \( IS_2 \) defined on \( S \otimes S \):

- Information similarity measure: \( ISIM(IS_1, IS_2) = |CI(IS_1, IS_2)| \)
- Total information quantity: \( TIQ(IS_1, IS_2) = |TI(IS_1, IS_2)| \)
- Information decrease measure: \( IDEC(IS_1, IS_2) = |MI(IS_1, IS_2)| \)
2.6. INFORMATION AND INFORMATION RELATIONSHIP ANALYSIS

- **Information increase measure:** \( IINC(IS_1, IS_2) = |EI(IS_1, IS_2)| \)
- **Information dissimilarity measure:** \( IDIS(IS_1, IS_2) = |DI(IS_1, IS_2)| \)

**Definition 2.26 (Abstraction measures).** Analogously, we can define abstraction measures for two abstraction sets \( AS_1 \) and \( AS_2 \) defined on \( S \otimes S \)

- **Abstraction similarity measure:** \( ASIM(\pi_1, \pi_2) = |CA(AS_1, AS_2)| \)
- **Total abstraction quantity:** \( TAQ(AS_1, AS_2) = |TA(AS_1, AS_2)| \)
- **Abstraction decrease measure:** \( ADEC(AS_1, AS_2) = |MA(AS_1, AS_2)| \)
- **Abstraction increase measure:** \( AINC(AS_1, AS_2) = |EA(AS_1, AS_2)| \)
- **Abstraction dissimilarity measure:** \( ADIS(AS_1, AS_2) = |DA(AS_1, AS_2)| \)

Once the information is recognized as important for processing of outputs, it can be categorized as required or useful information, otherwise, we refer to the information that is not necessary to compute output function as redundant information.

In the following definition \( IS_f \) models the information that is required to compute the output of a system and \( IS_x \) models the information provided by system’s input variable(s). Formally, the required information measure and redundant information measure is defined as follows [20]:

- **Required (useful) information measure:** \( IRQ(IS_x, IS_f) = |CI(IS_x, IS_f)| \)
- **Redundant information measure:** \( IRD(IS_x, IS_f) = |MI(IS_x, IS_f)| \)

If information and abstraction sets are induced by certain set-systems and these set-systems are in turn induced by binary or multi-valued variables, we use simplified notation to express the information relationship measures. For example, we write \( ISIM(\pi_x, \pi_y) \) or even \( ISIM(x, y) \) instead of \( ISIM(IS(x), IS(y)) \).

**Ex. 2.6.2.** Some information relationships between the input and output variables of the Boolean function from Example 2.6.1 (Table 2.3) are shown below.

- \( ISIM(\pi_{x_1}, \pi_{y_1}) = |\{1,3,1,4,2,3,2,4\}| = 4 \)
- \( ISIM(\pi_{x_1}, \pi_{y_2}) = |\{2,3,2,4,2,5\}| = 3 \)
- \( ISIM(\pi_{x_1}, \pi_{y_3}) = |\{0\}| = 0 \)
- \( TIQ(\pi_{x_1}, \pi_{x_2}) = |\{1,2,1,3,1,4,1,6,2,3,2,4,2,5,2,6,3,5,3,6,4,5,4,6,5,6\}| = 13 \)
- \( TIQ(\pi_{x_1}, \pi_{x_3}) = |\{1,2,1,3,1,5,2,3,2,5,3,4,3,5,4,5,5,6\}| = 9 \)
- \( IDEC(\pi_{x_1}, \pi_{y_1}) = |\{1,5,2,5\}| = 2 \)
- \( IDEC(\pi_{x_2}, \pi_{y_1}) = |\{1,3,1,4,1,5\}| = 3 \)
- \( IINC(\pi_{x_1}, \pi_{y_2}) = |\{1,6,2,5\}| = 2 \)
- \( IINC(\pi_{x_2}, \pi_{y_2}) = |\{1,2,1,6,3,6,4,6,5,6\}| = 5 \)
- \( IDIS(\pi_{x_1}, \pi_{y_1}) = |\{1,4,1,6,2,5,2,6,3,5,3,6,4,5,6,5,6\}| = 7 \)
- \( IDIS(\pi_{x_2}, \pi_{y_2}) = |\{1,2,1,3,1,4,1,5,1,6,3,6,4,6,5,6\}| = 8 \)

- \( IRQ(\pi_{x_1}, \pi_{y_1}) = |\{1,3,1,4,2,3,2,4\}| = 4 \)
- \( IRQ(\pi_{x_2}, \pi_{y_1}) = |\{2,3,2,4,2,5\}| = 3 \)
- \( IRD(\pi_{x_1}, \pi_{y_2}) = |\{1,6,2,6,3,5,4,5,5,6\}| = 5 \)
- \( IRD(\pi_{x_2}, \pi_{y_2}) = |\{1,3,1,4,1,5\}| = 3 \)
Each of the absolute measures discussed above can be normalized. There is no general normalization factor. The way a normalization factor is selected depends on the aim of the measure. For example, to measure how much information provides an information set $IS$ in relation to a maximal information set we can define the measure as:

$$IQ_N(IS) = \frac{IQ(IS)}{IQ(IS_{max})}$$

(2.6.19)

To know the normalized measure of how much information provides a variable $x$ modeled as $IS_x$ to a function $f$ modeled as $IS_f$, we could define measure:

$$ISIM_N(IS_x, IS_f) = \frac{|CI(IS_f, IS_x)|}{IQ(IS_f)}$$

(2.6.20)

We can also evaluate how much the redundant information variable $x$ provides to the function $f$:

$$FREDI_N(IS_x, IS_f) = \frac{|MI(IS_x, IS_f)|}{IQ(IS_f)}$$

(2.6.21)

and size of this redundant information in relation to information provided by $x$:

$$VREDI_N(IS_x, IS_f) = \frac{|MI(IS_x, IS_f)|}{IQ(IS_x)}$$

(2.6.22)

To know how much common information is in two information streams modeled respectively by $IS_1$ and $IS_2$ we would define the measure as:

$$ISIM_N(IS_1, IS_2) = \frac{|CI(IS_1, IS_2)|}{|TI(IS_1, IS_2)|}$$

(2.6.23)

In a similar way we can define normalized measures of abstraction.

**Ex. 2.6.3.** It is straightforward to prove that maximal information quantity that can be defined on a set of symbols $S$ is $IQ_{max}^S = |S|(|S| - 1)/2$ (equal to the number of all possible symbol pairs). Then the normalized information quantity provided by variable $x_4$ in Example 2.6.1 is

$$IQ_N(\pi_{x_4}) = \frac{IQ(\pi_{x_4})}{IQ_{max}^{\{\pi_{x_1}, \pi_{x_2}, \pi_{x_3}, \pi_{y_1}\}}} = \frac{1}{8}$$

Knowing the information sets of the input variables $x_1, x_2, x_3, x_4$ and output variable $y_3$ from Examples 2.6.1, we can calculate normalized information similarities (the degree of similarity) of these variables to output variable $y_3$:

$$ISIM_N(IS(\pi_{x_1}), IS(\pi_{y_1})) = 4/11 = 0.364$$
$$ISIM_N(IS(\pi_{x_2}), IS(\pi_{y_1})) = 3/11 = 0.273$$
$$ISIM_N(IS(\pi_{x_3}), IS(\pi_{y_1})) = 4/10 = 0.4$$
$$ISIM_N(IS(\pi_{x_4}), IS(\pi_{y_1})) = 0/6 = 0.0$$

(2.6.24)
2.6. INFORMATION AND INFORMATION RELATIONSHIP ANALYSIS

\[ FREDI_N(IS(\pi_{x_1}), IS(\pi_{y_1})) = \frac{2}{6} = 0.333 \]
\[ FREDI_N(IS(\pi_{x_2}), IS(\pi_{y_2})) = \frac{3}{6} = 0.5 \]
\[ FREDI_N(IS(\pi_{x_3}), IS(\pi_{y_3})) = \frac{2}{6} = 0.333 \]
\[ FREDI_N(IS(\pi_{x_4}), IS(\pi_{y_3})) = \frac{6}{6} = 1.0 \] (2.6.25)

The measure introduced in [20] is used to express the importance of information by associating an appropriate importance weight \( w_{IS}(s_i|s_j) \) with each elementary information. For the purpose of information importance modeling, in the research presented here, we have used the same measure. The function [20]:

\[ w_{IS} : S \otimes S \rightarrow [0 \ldots 1] \] (2.6.26)

is called the information weighting function. It associates a real number between 0 and 1 inclusive to each elementary information. Lowest weight (0) denotes lowest importance of that particular elementary information, while highest weight (1) signifies the most important information. The importance of particular elementary information item, might be due to its availability and/or its applicability to compute a certain number of output (Boolean) functions.

**Definition 2.27 (Weighted information quantity measure).**

\[ WIQ(IS) = \sum_{s_i|s_j \in IS} w_{IS}(s_i|s_j) \] (2.6.27)

During the designing or analyzing of information systems, we often face the requirement of a measure that expresses the value of information importance for a particular application. We may, for instance, ask how a particular information set is important to compute a certain Boolean function. The weighted information similarity measure, that is suggested in [20], expresses rank correlation coefficient between the information weight measure and common information of two information sets. These two information sets can be information set that is available in particular (input) variable, and a particular (output) variable that needs to be computed.

**Definition 2.28 (Weighted information similarity measure).** Let \( S \) be a set of symbols and \( IS_1, IS_2 \) be two information sets defined on \( S \otimes S \). The weighted information similarity of \( IS_2 \) to \( IS_1 \) can be calculated as follows:

\[ WSIM(IS_1, IS_2) = \sum_{s_i|s_j \in CI(IS_1, IS_2)} w_{IS}(s_i|s_j) \] (2.6.28)

where \( w_{IS}(s_i|s_j) \) is an importance weight for an elementary information \( s_i|s_j \). [20]

In the method described in [20], a new weighting function is used to model the decrease of information importance with the number of (binary or multi-valued) inputs or intermediate variables \( x \) of a certain function \( f \) at which this information is present. Building this heuristic weighting function, the following assumptions were made [20]:
• \( w(s_i|s_j) = 0 \) if information \( s_i|s_j \) is not required for the computation of a function \( f \),

• \( w(s_i|s_j) = 1 \) if information \( s_i|s_j \) is necessary for the computation of a function \( f \),

• the sum of weights of the less important information cannot dominate the weight of the more important information,

• the weighting function is built in a context of some information set \( IS \), which often denotes the information set \( IS(f) \) induced by a function \( f \).

Definition 2.29 (Support of elementary information item). Let \( X \) be a set of binary or multi-valued variables. A sub-set of binary or multi-valued variables denoted as \( \text{sup}(s_i|s_j)|X \) is the support of elementary information item \( s_i|s_j \) if and only if:

\[
\forall x_n \in \text{sup}(s_i|s_j) \quad (s_i|s_j) \in IS(\pi_{x_n}),
\]

and

\[
\forall x_m \notin \text{sup}(s_i|s_j) \quad (s_i|s_j) \notin IS(\pi_{x_m}).
\]

If the set of variables \( X \) is known, we briefly use notation \( \text{sup}(x_i, x_j) \) from now on.

We call \( s_i|s_j \) a unique information when it is provided by only one binary or multi-valued variable i.e., if and only if \( |\text{sup}(s_i|s_j)| = 1 \).

For further details on the weighted information similarity and its application to support construction process, the interested reader should refer to [20].
Chapter 3

Information-driven General Functional Decomposition

In this chapter the method of general functional decomposition of Boolean combinatorial functions is emphasized. It is the method that was used as a basis in the adaptation of existing algorithms, heuristics and procedures for the purpose of decomposition targeted to complex gates libraries. The changes and modifications required by gate libraries specifics are also presented in further details in following chapters. In this chapter we focus mainly on the main mechanisms that control the process of decomposition.

3.1 Classical functional decomposition

This dissertation focuses on the general functional decomposition of Boolean functions. Therefore, in this chapter only, the review of functional decomposition methods is limited to this kind of functions. For additional information on the topic of application of functional decomposition to the multi-valued functions and/or relations, the interested reader should refer to [11, 48, 104]. The definitions and theorems presented further below are quoted from [20, 49, 72, 106, 124, 133].

Functional decomposition transforms an original function (or relation) into a network of simpler sub-functions (sub-relations). This process is performed recursively step by step, until specific constraints (e.g., implementation of Boolean function out of limited subset) are met and some optimization objectives (physical features, such as area, delay and/or power) are optimized. A network that satisfies structural constraints is called a feasible network. In the case of functional decomposition targeting gates libraries, the structural constraints are defined by a given technology library, which defines the set of Boolean functions directly one-to-one implementable with the library gates. A network that consists of directly mappable functions is called a feasible network or bounded network.

The decomposition theorems provide generators for the correct (partial and final) circuit structures. After each single decomposition step a partial solution of the network is constructed. Throughout the process of decomposition, the behavior of the
original function is preserved. A given Boolean function may be implemented with gates of a given technology library in numerous ways. To minimize the time spent on the exploration of the huge space of (all) possible solutions, heuristic search is performed that uses the correct circuit generator in a selective way.

The key differences between the decomposition methods are in the ways the fundamental issues are tackled: the way correctness of decomposition is ensured and what network structures the aforementioned generators, can produce, and in the heuristic search algorithms used.

The classical methods of functional decomposition are some extensions of the cardinal Shannon’s observation. In [124] Shannon noticed that the more adequately one can decompose a circuit synthesis problem into a combination of simpler problems, the simpler is the final circuit implementation. He coined a property of Boolean function that could be effectively used for decomposition: functional separability. Boolean functions that hold such property can be expressed by the following formula:

\[ f = h(g(x_1, \ldots, x_s), x_{s+1}, \ldots, x_n) = h(g(U), V). \]  

(3.1.1)

Figure 3.1 presents graphically the method of functional decomposition, known as simple disjoint decomposition, based on the property of functional separability. Input variables are split into two subsets: bound-set: \( \{x_1, \ldots, x_s\} \) denoted as \( U \) and free-set: \( \{x_{s+1}, \ldots, x_n\} \) denoted as \( V \). Boolean relation labeled \( h \) is called a composition function, image function of \( f \), or simply successor function. Sub-function \( g \) is often called a bound-set function or predecessor function.

Publication of Shannon lacked a systematic method for finding decompositions of this type. Five years after Shannon’s publication Povarov presented in [106] necessary and sufficient condition for the existence of functional separation in the form of the following theorem:

**Theorem 3.1 (Existence of functional separation [106]).** Boolean function \( f(x_1, x_2, \ldots, x_n) \) has functional separation (simple disjoint decomposition) if and only if for
at least one subset \( \{v_1, v_2, \ldots, v_{n-s}\} = V \) of variables \( x_1, x_2, \ldots, x_n \), where \( 1 < s < n-1 \), such a function \( g \) of the remaining \( s \) variables exists that all different from 0 and 1 cofactors of \( f \) with respect to \( v_1, v_2, \ldots, v_{n-s} \) are equal to \( g \) or \( \overline{g} \).

Aforementioned simple disjoint decomposition scheme is a very specific case of generalized disjoint functional decomposition. This generalization of decomposition scheme was presented by Curtis in [29]. In this approach, more than one bound-set sub-function \( g \) can be constructed in a single decomposition step.

**Theorem 3.2 (Generalized disjoint decomposition).** A switching function \( f(U, V) \) is expressible as a composite function \( h(g_1(U), \ldots, g_k(U), V) \) where the sub-functions are \( k \) in number, if and only if its \( 2^{|U|} \times 2^{|V|} \) decomposition chart has at most \( 2^k \) distinct column vectors. [29]

The minimal number of single output sub-functions \( g_i \) to satisfy correct decomposition is given by \( k \). Convergence is defined as a difference \((|U \cup V| - (|V| + k))\) of total number of input variables and total number of output variables of sub-function \( g \).

Through the application of Curtis’s theorem one can find a number of possible and correct decompositions than does simple disjoint decomposition. It does, though, require finding solution to another problem, known as sub-function binary encoding. Section 7.1 describes this problem in details.

Further advances in functional decomposition were possible with the introduction of the theory of symbolic analysis of decomposition of sequential machines [37, 38]. To decompose incompletely specified (partial) Boolean and multi-valued functions, Roth and Karp in [113] introduced a concept of abstract (symbolic) decomposition of discrete functions. Their proof of the existence of abstract decomposition was based on symbols compatibility relation (see Definition 2.1).

Let \( U, V \) be arbitrary finite sets (of symbols) and let \( E \) be a subset of Cartesian product \( U \times V \). We say that \( u_1, u_2 \in U \) and \( v \in V \) are compatible with respect to some function \( f \) (denoted as \( u_1 \sim u_2 \)) if for all \( v \in V \) such that \((u_1, v) \in E \) and \((u_2, v) \in E \), \( f(u_1, v) = f(u_2, v) \) or \( f(u_1, v) \) is don’t care or \( f(u_2, v) \) is don’t care; otherwise, \( u_1 \) is incompatible with \( u_2 \), denoted \( u_1 \not\sim u_2 \).

**Theorem 3.3 (Existence of abstract decomposition [113]).** Let \( U, V, Z, \) and \( \mathcal{W} \) be arbitrary finite sets (of symbols) and let \( E \) be a subset of Cartesian product \( U \times V \). Let

\[
\begin{align*}
g & : U \to \mathcal{W} \text{ and} \\
\quad \quad h & : \mathcal{W} \times V \to Z.
\end{align*}
\]

(3.1.2) (3.1.3)

Function \( f : E \to Z \) has a decomposition in form

\[
f(u, v) = h(g(u), v) \quad \text{for all} \quad (u, v) \in E
\]

(3.1.4)

if and only if \( U \) can be partitioned into \( k \) classes of mutually compatible elements (symbols) and if and only if \( \mathcal{W} \) has at least \( k \) elements (symbols).
For completely specified functions $f$, the meaning of $k$ is defined as number of maximal equivalence classes. For incompletely specified functions the determination of a minimum number of compatible classes is nontrivial, as the compatibility relation is no longer an equivalence relation. The minimal number $k$ can be derived from the features of incompatibility graph. Value of $k$ corresponds to the clique number or minimal number of colors incompatibility graph can be colored. Graph coloring or partitioning requires solving NP-complete problem [36]. Therefore, the complexity of decomposition of incompletely specified function is larger than completely specified function. The increase of computational complexity is mainly due to an increase of solution space. Larger number of solutions yields also an additional freedom during the decomposition process. This freedom can be used to optimize the final circuit implementation.

In [113] Roth and Karp studied the non-disjoint decomposition. They regarded (non-disjoint) decomposition of function $f$ with input variables consisting of inputs $x_1, x_2, \ldots, x_n$ as follows (see Fig. 3.2):

$$f(x_1, x_2, \ldots, x_n) = h(g_1(U), g_2(U), \ldots, g_m(U), V)$$

(3.1.5)

where $U$ and $V$, are subsets $\lambda = \{x_1, x_2, \ldots, x_s\}$ and $\mu = \{x_t, x_{t+1}, \ldots, x_n\}$ of input variables. If said subsets are disjoint, meaning $\lambda \cap \mu = \emptyset$, the decomposition is referred as disjoint, otherwise, they are called non-disjoint. The examples of disjoint and non-disjoint decomposition can found in [20].

### 3.2 General decomposition

The foundation of general decomposition of sequential machines has been formulated by Jóźwiak in [49] and [72]. The general decomposition theory can be applied to the decomposition of combinational machines, due to the fact that a combinational
3.2. GENERAL DECOMPOSITION

Figure 3.3: Graphical representation of combinational machine.

The theory of general decomposition constitutes one of the foundations of this thesis, and facilitates the decomposition process by providing a structured methodology of network generation. In this section, a part of the general decomposition theory related to combinational machines and combinational circuits synthesis is presented. The definitions and theorems presented in the following sections are quoted from [20, 49, 72, 133].

3.2.1 Combinational machines

In the general decomposition theory, to describe a discrete functions and combinational circuits an mathematical model called a **combinational machine** [49, 133, 134] is used.

**Definition 3.1 (Completely specified combinational machine).** A completely specified combinational machine $M$ is an algebraic system defined by:

$$M = (I, O, \lambda)$$

where:

- $I$ - is a non-empty finite set of input symbols,
- $O$ - is a non-empty finite set of output symbols,
- $\lambda$ - is an output function $\lambda : I \rightarrow O$.

In Figure 3.3 the graphical representation of a combinational machine is shown. It transforms a set of input symbols into a set of output symbols. For practical reasons, the following model, mathematically equivalent, can be used, to model a multiple input and output combinational machine.

**Definition 3.2 (Completely specified multiple i/o combinational machine).**

A completely specified combinational machine $M$ with $n_i$ inputs and $n_o$ outputs is an algebraic system defined by:

$$M^* = (I^*, O^*, \lambda^*)$$

where:

$$I^* = \bigotimes_{1 \leq j \leq n_i} I_j$$ - and $I_j$ is a non-empty set of symbols of input $j$. 


3. INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

\[ O^* = \bigotimes_{1 \leq k \leq n_o} O_k - \text{and } O_k \text{ is a non-empty set of symbols of output } k, \]

\[ \lambda^* \quad \text{- is an multiple input/output function } \lambda^* : I^* \rightarrow O^*. \]

In general case, a combinational machine does not need to be specified for every possible input and output combinations. Such conditions \textit{dc}, or informally \textit{don’t care’s}, for sake of clarity and readability are denoted as "*". The following model takes into account don’t care conditions to increase search space for a machine realization.

**Definition 3.3 (Incompletely specified multiple i/o combinational machine).**
An incompletely specified combinational machine \( M \) with \( n_i \) inputs and \( n_o \) outputs is an algebraic system defined by:

\[ M^{dc} = (I^{dc}, O^{dc}, \lambda^{dc}) \quad (3.2.3) \]

where:

\[ I^{dc} = \bigotimes_{1 \leq j \leq n_i} I_j - \text{and } I_j \text{ is a non-empty set of symbols of input } j, \]

\[ O^{dc} = \bigotimes_{1 \leq k \leq n_o} O_k \cup \{*\} - \text{and } O_k \text{ is a non-empty set of symbols of output } k, \]

\[ \lambda^{dc} \quad \text{- is an multiple input/output function } \lambda^{dc} : I^{dc} \rightarrow O^{dc}. \]

In this thesis we refer to a special case of "generalized" don’t cares \( \textit{gdc} \). The generalized input don’t care condition defined on input \( j \) contains all sub-sets of symbols from \( I_j \), and the analogously generalized don’t care condition defined on output \( k \) contains all sub-sets of symbols from \( O_k \). Here we only consider binary machines and Boolean functions and their binary implementations. In such case \( dc = gdc = \{0, 1\} \).

The definition of realization of an incompletely specified machine, requires an introduction of symbol covering first.

**Definition 3.4 (Symbol covering).** We are given a non-empty set of symbols \( S \) and set \( S^{dc} = S \cup \{*\} \). Symbol \( b \in S^{dc} \) is said to cover symbol \( a \in S^{dc} \) (denoted as \( a \leq b \)) if and only if \( (a = b) \lor (b = *) \).

We say that \( a \) is more specific than \( b \), when \( b \) covers \( a \).

**Definition 3.5 (Realization of incompletely specified combinational machine).**
An incompletely specified machine \( M' = (I', O', \lambda') \) is a realization of an incompletely specified machine \( M = (I, O, \lambda) \) if and only if the function \( \Psi : I \rightarrow I' \) and surjective partial function \( \Theta : O' \rightarrow O \) exists, so that

\[ \forall_{x \in I} \Theta(\lambda'((\Psi(x)))) \leq \lambda(x) \quad (3.2.4) \]
3.2. General decomposition

Figure 3.4 visualizes the structure of a system \((\Psi, M', \Theta)\) introduced in the above definition. Function \(\Psi\) encodes the input symbols \(I\) into a set of symbols \(I'\) being "understood" by the realization machine. Function \(\Theta\) translates (decodes) the output symbols of the realization machine into output symbols of the originally specified machine \(M\). Definition 3.5 requires that for all input symbols \(x \in I\) of the original machine \(M\), the output symbol \(\lambda'(\Psi(x))\) of the realization machine \(M'\) (decoded by \(\Theta(\lambda'(\Psi(x)))\)) is covered by the corresponding output symbol \(\lambda(x) \in O\) of the specification machine \(M\). In other words, the realization machine must not be less specific than the original machine. In this way the behavior of the specification machine is preserved.

An example realization of a combinational machine an interested reader can find in [20].

3.2.2 General decomposition theorem

General composition is formal representation of a network of the interconnected component combinational machines.

Definition 3.6 (General composition). A general composition of \(n\) combinational machines \(M_i : GC(\{M_i\}, \{Con_i\})\) is a structure (network) that consists of:

1. A set of component machines

\[
\left\{ M_i = (I_i^*, O_i, \lambda_i) \mid I_i^* = I_i \otimes I'_i, 1 \leq i \leq n \right\}
\]

2. A set of surjective functions referred to as connection rules

\[
\left\{ Con_i : \bigotimes_{1 \leq j \leq n} O_j \to I'_i, 1 \leq i \leq n \right\}
\]

Every component machine \(M_i\) requires for operation a certain information present in its inputs represented by the set of symbols \(I_i^*\). Input information processed internally is then exposed to component machine output. The set \(I_i^*\) contains all possible combinations of the symbols incoming from the environment the network is embedded in \((I_i)\), and the symbols incoming from the component machines \((I'_i)\), including \(M_i\) as well. To describe the connections of outputs of individual machines \(M_j\) with
inputs of individual machines $M_i$, the Cartesian product of sets of output and input symbols of corresponding machines are provided in connection rule $Con_i$.

In general, a decomposition of a combinational machine $M = (I, O, \lambda)$, a certain composition of $n$ cooperating partial machines $M_i = (I_i, O_i, \lambda_i)$, and the corresponding mappings

$$
\Psi : I \to \bigotimes_i I_i
$$

$$
\Theta : \bigotimes_i O_i \to O
$$

are constructed in such a way that the composition of all partial machines $M_i$ together with the mappings $\Psi$ and $\Theta$ realize machine $M$ [20].

The mapping $\Psi$ can be recognized as the input encoder and its implementation is also an combinational circuit. Input encoder preprocesses the input information for the main component machines. Its main task is to distribute the information in suitable form for each parallel component machine. Similarly, the output decoder $\Theta$ task is to combine the information from parallel concurrent processors and represent it in suitable form for primary outputs, hence the entire machine $M$. In other words, these two machines prepare information from and to the external network, the main machine $M$ is surrounded by. The main implementation of information processing machine is a network of interconnected component sub-machines $M_i$. Interconnection(s) allows for information transfer between concurrent sub-machines. Each component machine can use (partial) information computed by another component machine for the purpose of computation of its own output information. In general, two special cases of composition: serial and parallel. In latter, no connection exists between concurrent component machines. In contrary, in serial composition, machines are ordered, and machine(s) $M_j$ are using the information computed by machine(s) $M_i$, where $j < i$. To summarize the general composition machine can be defined as follows:

**Definition 3.7 (General composition machine).** A general composition $GC$ of $n$ combinational machines defines the general composition machine $M_{GC}(GC) = (I_{GC}, O_{GC}, \lambda_{GC}) = M_{GC}({\{M_i\}}, {\{Con_i\}})$ with:

- **Set of inputs**
  
  $$
  I_{GC} = \bigotimes_i I_i,
  $$

- **Set of outputs**
  
  $$
  O_{GC} = \bigotimes_i O_i,
  $$

- **Realized output function**

  $$
  \lambda_{GC} : I_{GC} \to O_{GC},
  $$

  where

  $$
  \lambda_{GC} : \bigotimes_i \lambda_i(x_i, Con_i(y_1, \ldots, y_n)),
  $$

  and $y_i$ represents the output of component machine $i$. 
3.2. GENERAL DECOMPOSITION

The general composition machine $M_{GC}(GC)$ is a general decomposition of the machine $M$, if and only if, $M_{GC}(GC)$ realizes $M$. No distinction between general composition and the composition machine it defines will be made, until it may lead to misunderstanding.

The correspondence between input and output set of symbols needs to be defined if these two sets for machine $M$ are different.

**Definition 3.8 (Input-output (I-O) partition pair).** Given $M = (I, O, \lambda)$, let $\Pi_I$ be a partition on a set of input symbols $I$ and let $\Pi_O$ be a partition on a set of output symbols $O$. $(\Pi_I, \Pi_O)$ is an input-output partition pair if and only if

$$\forall A \in \Pi_I \exists C \in \Pi_O \lambda(A) \subseteq C \quad (3.2.5)$$

where:

$$\lambda(A) = \{ \lambda(x) | x \in A \} \quad (3.2.6)$$

In other words, $(\Pi_I, \Pi_O)$ is an input-output partition pair if there is unambiguous mapping of $\Pi_I$’s blocks into $\Pi_O$’s blocks. The mapping is unambiguous if and only if, after mapping all symbols from each block of input partition $\Pi_I$ onto corresponding output symbols the set of all output symbols resulting from this mapping is completely contained in a block of output partition $\Pi_O$. The following theorem was proven by Jóźwiak in [49]:

**Theorem 3.4 (Existence of general decomposition).** A combinational machine $M(I, O, \lambda)$ has a general decomposition into $n$ component machines if and only if $n$ partition doubles $(\Pi_I^i, \Pi_O^i)$ exist that satisfy the following conditions:

1. $$\Pi_I^i \cdot \Pi_I'^i \leq \Pi_I^i \quad (3.2.7)$$
   where:
   $$\Pi_I'^i \geq \prod_{i=1}^{i=n} \Pi_I^{i'} \quad (3.2.8)$$

2. $$\Pi_I^i \leq \Pi_I^{i'} \quad (3.2.9)$$

3. $$\left( \prod_{i=1}^{i=n} \Pi_I^{i'}, \Pi_O(0) \right)$$ is I-O partition pair \quad (3.2.10)

The above theorem does not exclude trivial decompositions, such as decompositions containing empty or duplicated component machines. A general decomposition
is said to be not trivial if each of the component machines is necessary for obtaining the output of the machine being decomposed and none of the partial machines is the same as the original. We discuss below the meaning of each condition, using the special case of general decomposition - decomposition with two component machines.

**Theorem 3.5 (General decomposition with two component machines).** A combinational machine $M(I, O, \lambda)$ has a general decomposition with two partial machines without local connections if and only if two partition doubles $(\Pi_I, \Pi_I^*)$ and $(\Gamma_I, \Gamma_I^*)$ exist that satisfy the following conditions:

1.

$$\Gamma_I \cdot \Pi_I' \leq \Gamma_I^* \text{ and } \Pi_I \cdot \Gamma_I' \leq \Pi_I^*$$  \hspace{1cm} (3.2.11)

where:

$$\Pi_I' \geq \Pi_I^* \text{ and } \Gamma_I' \geq \Gamma_I^*$$  \hspace{1cm} (3.2.12)

2.

$$\Pi_I \cdot \Gamma_I \leq \Pi_I^* \text{ and } \Pi_I \cdot \Gamma_I \leq \Gamma_I^*$$  \hspace{1cm} (3.2.13)

3.

$$(\Pi_I^* \cdot \Gamma_I^*, \Pi_O(0)) \text{ is an I-O partition pair}$$  \hspace{1cm} (3.2.14)

Figure 3.5 illustrates the above theorem. The output information $\Pi_I^*$ computed by machine $M$ depends only on information provided by primary inputs represented by $\Pi_I$ and part of information computed by machine $M_2$ represented by $\Gamma_I'$. Similarly, combinational machine $M_2$ computes its output information from $\Gamma_I$ and $\Pi_I'$. Since both machines $M_1$ and $M_2$ are combinational, they have no memory and their outputs have to be calculated from the inputs only. This condition is expressed by equation 3.2.11. Equation 3.2.12 expresses the information processing capacity [133] of the connection rules $Con_{1,2}$ and $Con_{2,1}$: the connection rule can transmit a part of
the output information of a certain component machine to the other machine. Equation 3.2.13 ensures that it is possible to construct the general composition of \( M_1 \) and \( M_2 \) as a legal decomposition, by ensuring that the exchanged information can be computed (directly or indirectly) from the primary input information of the partial machines.\[133\] The last condition (3.2.14) guarantees that the information computed by the component machines is sufficient to calculate the output of the original machine \( M \).

**Definition 3.9 ("General" composition without loops).** A "general" composition without loops of \( n \) combinational machines \( M_i : GCNL(\{M_i\}, \{Con_i\}) \) is a structure (network) that consists of:

1. A set of component machines
   \[
   \left\{ M_i = (I_i^*, O_i, \lambda_i) \mid I_i^* = I_i \otimes I_i', 1 \leq i \leq n \right\}
   \]

2. A set of surjective functions referred to as connection rules
   \[
   \left\{ Con_i : \bigotimes_{1 \leq j < i} O_j \to I_i', 1 \leq i \leq n \right\}
   \]

In a general composition, there is a danger of information loops occurring in the exchanged information. Such loops occurring at the level of elementary (binary) signal lines will result in sequential behavior of the two interconnected combinational circuits which compute \( \lambda_i \) instead of the required combinational behavior. We say that a general composition is legal if and only if the composition \( \lambda^* \) of \( \lambda^* \) is guaranteed to be a function. \[49\]

If it might lead to misunderstanding, no explicit distinction between general composition and general composition machine it defines would be made throughout this thesis.

**Definition 3.10 ("General" composition machine without loops).** A "general" composition without loops \( GCNL \) of \( n \) combinational machines defines the "general" composition machine \( M_{GCNL}(GCNL) = (I_{GCNL}, O_{GCNL}, \lambda_{GCNL}) = M_{GCNL}(\{M_i\}, \{Con_i\}) \) with:

- **Set of inputs**
  \[
  I_{GCNL} = \bigotimes_i I_i,
  \]

- **Set of outputs**
  \[
  O_{GCNL} = \bigotimes_i O_i,
  \]

- **Realized output function**
  \[
  \lambda_{GCNL} : I_{GCNL} \to O_{GCNL},
  \]
where

$$\lambda_{GCNL} : \bigotimes_i \lambda_i(x_i, Con_i(y_i, \ldots, y_n)),$$

and $$y_i$$ represents the output of component machine $$i$$.

**Definition 3.11 ("General" decomposition without loops).** The combinational machine $$M_{GCNL}(GCNL)$$ is a "general" decomposition without loops of machine $$M$$, if and only if $$M_{GCNL}(GCNL)$$ realizes $$M$$.

General decomposition theorem provides framework of methodology to construct functionally correct combination circuits. It does not, though, provide any procedure for finding systems of partition that produce good decompositions for some objectives, e.g. fast, low area etc. In this thesis the information-driven circuit synthesis approach is presented. It relies on the analysis of the information flow structure and relationships in the function to be implemented, as well as, in the circuit under construction, and usage of the results of this analysis controls the construction of the circuit.

The main contribution of this work in this respect is the development of the effective and efficient heuristics that control decomposition process in such a way that the resulting logic gate networks are fast (critical path is close to minimum) and compact (contain a small number of logic blocks and interconnections)

<table>
<thead>
<tr>
<th>I</th>
<th>x_1</th>
<th>x_2</th>
<th>x_3</th>
<th>x_4</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.1: Boolean function $$f$$ used in general decomposition Example 3.2.1.

**Ex. 3.2.1 (Network construction with general decomposition).** Figure 3.1 shows a completely specified Boolean function $$f$$. Let us decompose this function into a network of gates out of limited set of two-input one-output logic blocks. We assume, that each logic block can implement any two-input Boolean function out of set of gates
in Figure 3.6. Technology library pre-characterization provides necessary information about the mappable Boolean functions. In this example for the sake of simplicity we ignore the physical features of gates. Here we emphasize the logic synthesis constrained by the logic functions implemented in physical gates available in technology library. Here, the implementation selection process is based solely on the logic requirements of correct implementation of desired function.

\[
\begin{align*}
\Pi_{\text{NOT}} &= \{ 0, 1 \} \\
\Pi_{\text{AND}} &= \{ 0, 1, 2 \} \\
\Pi_{\text{NAND}} &= \{ 0, 1, 2 \} \\
\Pi_{\text{OR}} &= \{ 0, 1, 2, 3 \} \\
\Pi_{\text{NOR}} &= \{ 1, 2, 3, 0 \} \\
\Pi_{\text{XOR}} &= \{ 0, 1, 2 \} \\
\Pi_{\text{NXOR}} &= \{ 1, 2, 3 \}
\end{align*}
\]

Primary inputs of \( f \) define the set of symbols the machine is processing \( I = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\} \). The output alphabet \( O = \{0, 1\} \). In the first decomposition step, we decompose the original function \( f \) into two parallel machines \( A \) and \( D \) (see Figure 3.7).

Three input component machine \( A \) processes information from three primary inputs: \( x_1, x_2 \) and \( x_3 \) represented respectively by partitions:

\[
\begin{align*}
\Pi_{x_1} &= \{ 0, 1, 2, 3, 4, 5, 6, 7 ; 8, 9, 10, 11, 12, 13, 14, 15 \} \\
\Pi_{x_2} &= \{ 0, 1, 2, 3, 8, 9, 10, 11 ; 4, 5, 6, 7, 12, 13, 14, 15 \} \\
\Pi_{x_3} &= \{ 0, 1, 4, 5, 8, 9, 12, 13 ; 2, 3, 6, 7, 10, 11, 14, 15 \}
\end{align*}
\]

As a side note, the last input \( x_3 \) is also used in the set of input variables of another component machine referred as \( D \) in Figure 3.7. Symbols from all the input variables, represented by partition \( \Pi_A \):

\[
\Pi_A = \Pi_{x_1} \cdot \Pi_{x_2} \cdot \Pi_{x_3} = \{ 000, 001, 010, 011, 100, 101, 110, 111 \}
\]

are translated into component machine own output partition

\[
\begin{align*}
\Pi_A' &= \{ 0, 1, 4, 5, 8, 9 ; 2, 3, 6, 7, 10, 11, 12, 13, 14, 15 \}
\end{align*}
\]

Two input component machine \( D \) takes information from inputs \( x_3 \) and \( x_4 \) represented by partitions:

\[
\begin{align*}
\Pi_{x_3} &= \{ 0, 1, 4, 5, 8, 9, 12, 13 ; 2, 3, 6, 7, 10, 11, 14, 15 \} \\
\Pi_{x_4} &= \{ 0, 2, 4, 6, 8, 10, 12, 14 ; 1, 3, 5, 7, 9, 11, 13, 15 \}
\end{align*}
\]
Information combined on these two inputs together:

\[ \Pi_D = \Pi_{x_3} \cdot \Pi_{x_4} = \{ 0, 4, 8, 12 ; 1, 5, 9, 13 ; 2, 6, 10, 14 ; 3, 7, 11, 15 \} \]

is translated into component machine output partition

\[ \Pi_D^* = \{ 0, 3, 4, 7, 8, 11, 12, 15 ; 1, 2, 5, 6, 9, 10, 13, 14 \} \]

Partition \( \Pi_D^* \) is mappable by two-input AND gate. For details on mapping techniques please refer to Chapter 7.3.2. Two component machines share one of the primary inputs, namely \( x_3 \). We refer to such decomposition as non-disjoint. Furthermore, none of the two machines takes any information from the output of the other machine. Hence, both partitions \( \Pi_A^* \) and \( \Pi_B^* \) are identity partitions \( \Pi(I) \). Formally, the condition 3.2.7 of Theorem 3.4 can be expressed as follows:

\[ \Pi_A \leq \Pi_A^* \quad \text{and} \quad \Pi_D \leq \Pi_D^* \]

The condition 3.2.9 holds, in other words, machines \( A \) and \( D \) can compute their output from their input.

As a next step, the condition 3.2.10 requires verification, that the information provided by machines \( A \) and \( D \) is sufficient to compute the output information of the original function \( f \). It is done with help of the following:

\[ \Pi_A^* \cdot \Pi_D = \{ 0, 4, 8 ; 1, 5, 9 ; 3, 7, 6, 11, 12, 15 ; 2, 6, 10, 13, 14 \} \]

The condition 3.2.10 holds, as the output partition \( \Pi_f \), and \( (\Pi_A^*, \Pi_D^*) \) fulfills the following relation

\[ \Pi_f = \{ 6, 10, 13, 14 ; 0, 1, 2, 3, 4, 5, 7, 8, 9, 11, 12, 15 \} \geq \Pi_A \cdot \Pi_D \]

Thus, \( (\Pi_A^*, \Pi_D^*, \Pi(0)) \) is an input-output partition pair. Component machine \( A \) needs to be further decomposed, because it cannot be mapped using any of the available technology gates. The following partitions show the realization of machine \( A \) decomposed further serially into two smaller sub machines \( B \) and \( C \):

\[ \Pi_B = \Pi_{x_3} \cdot \Pi_{x_4} = \{ 0, 1, 2, 3 ; 4, 5, 6, 7 ; 8, 9, 10, 11 ; 10, 11, 14, 15 \} \]
\[ \Pi_B^* = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 \} \]
\[ \Pi_{x_3} = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 \} \]
\[ \Pi_C = \Pi_{x_3} \cdot \Pi_{x_4} = \{ 0, 1, 4, 5, 8, 9 ; 2, 3, 6, 7, 10, 11 ; 12, 13, 14, 15 \} \]
\[ \Pi_A = \Pi_C \]

Due to a fact that partitions \( \Pi_B^*, \Pi_C^* \) and \( \Pi_A^* \) are defined on \( I \), there is no need for output decoder \( \Theta \). Furthermore, instead of condition 3.2.9 of the Theorem 3.4 the \( \leq \) property can be used.

Since machine \( C \) uses output information from machine \( B \), but machine \( C \) does not use any output information from machine \( D \), it is enough to validate the simplified condition of Theorem 3.4:

1. \( \Pi_B \leq \Pi_B^* \) and \( \Pi_C \cdot \Pi_B \leq \Pi_C^* \), where \( \Pi_B^* \geq \Pi_B \)
2. \( \Pi_B \cdot \Pi_C \leq \Pi_B^* \) and \( \Pi_B \cdot \Pi_C \leq \Pi_C^* \)
3. \( \Pi_B \cdot \Pi_C \leq \Pi_A \)

All information computed within component machine \( B \) is available on input variables of machine \( C \), therefore \( \Pi'_{BC} = \Pi_B \). The condition 3.2.8 satisfaction can be proven as the following partition \( \Pi_B \cdot \Pi_C \) is computed:
\[
\Pi_B \cdot \Pi_C = \{ 0, 4 ; 1, 5 ; 2, 6 ; 3, 7 ; 8, 12 ; 9, 13 ; 10, 14 ; 11, 15 \},
\]

To conclude the correctness of decomposition, one need to prove that condition 3.2.9 is fulfilled as well.
\[
\Pi_B \cdot \Pi_C = \{ 0, 1, 5, 10, 11, 14, 15 ; 2, 6, 8, 12 ; 3, 7, 9, 13 \}
\]

The network of combinational machines being the result of this decomposition is shown in Figures 3.7, while all the components machines (gates) in Figure 3.8.

![Figure 3.7: Network implementing function \( f \) constructed using general decomposition.](attachment:image.png)

A similar concept to the input-output partition pair for incompletely specified function is the set-system partition pair. This extension allows to apply decomposition schemes to incompletely specified functions. This correspondence allows to formulate theories presented by substitution of partition by set-systems.
Definition 3.12 (Input-output (I-O) set-system pair). Given $M = (I, O, \lambda)$, let $\pi_I$ be a set-system on a set of input symbols $I$ and let $\pi_O$ be a set-system on a set of output symbols $O$. $(\pi_I, \pi_O)$ is an input-output set-system pair if and only if

$$\forall A \in \pi_I \exists C \in \pi_O \lambda(A) \subseteq C \quad (3.2.15)$$

where:

$$\lambda(A) = \{ \lambda(x) | x \in A \} \quad (3.2.16)$$

Input-output set-system pair ensures that an incompletely specified realization machine is not less specific than an incompletely specified original machine; therefore, this condition is correct.

The general decomposition theorem of incompletely specified functions presented in details with proof in [72] can be expressed as follows:

Theorem 3.6 (General decomposition of incompletely specified machine).
An incompletely specified combinational machine $M(I, O, \lambda)$ has a general decomposition into $n$ component machines if and only if $n$ set-system doubles $(\pi_i^i, \pi_i^{i*})$ exist and they satisfy the following conditions:

1. $\pi_i^i \cdot \pi_i^{i*} \leq \pi_i^{i*} \quad (3.2.17)$

where:

$$\pi_i^{i*} \geq \prod_{i=1}^{n} \pi_i^{i*} \quad (3.2.18)$$

2. $\prod_{i=1}^{n} \pi_i^i \leq \pi_i^{i*} \quad (3.2.19)$

3. $\left( \prod_{i=1}^{n} \pi_i^{i*}, \pi_O(0) \right)$ is I-O set-system pair. \quad (3.2.20)
3.2. GENERAL DECOMPOSITION

3.2.3 Special cases of general decomposition

Classification originally introduced by Jóźwiak in [49] distinguishes decompositions depending on the connection type. General decomposition, can be seen in Figure 3.5 for two component decomposition, where there is no restriction imposed on connections between component machines. Parallel decomposition, can be seen in Figure 3.9 for two component decomposition, where component machines work independently, since there is no connection between two-component machines. Serial decomposition, can be seen in Figure 3.10 for two component decomposition, where component machines are ordered in a sequence $M_1, M_2, \ldots, M_n$. Machine $M_i$ can use output of machine $M_j$ only if $j < i$, i.e. if $M_j$ is predecessor of $M_i$ in the defined order. Classical decomposition schemes are included in this type.

A special case can be distinguished in general decomposition depending on the presence or absence of the nontrivial input encoder. If in the resulting network, the input encoder can be implemented by appropriate selection and distribution of primary inputs, we refer to such case an input-bit decomposition (see Figure 3.11). The primary inputs are connected directly to particular component machines, and each component machine uses only those inputs that are necessary to compute its output.
In general case, a non-trivial input encoder in the form of a discrete combinational sub-circuit is used. Analogously, a special case in general decomposition can be distinguished depending on presence or absence of the nontrivial output decoder. We refer to a special case of output-bit decomposition, if the output decoder can be reduced to adequate connection of appropriate primary outputs to partial machines. In such case each sub-machine constructs its own part of output separately. This type of decomposition is often called output decomposition or parallel decomposition \[68, 87\]. In general case, a non-trivial output encoder in the form of a discrete combinational circuit is used.

When both aforementioned conditions are fulfilled, the decomposition is both, the input-bit and output-bit one, it is called bit-decomposition. One of such decomposition schemes is known as Roth-Karp abstract decomposition scheme. It is a serial bit decomposition into two component machines, as shown in Figure 3.13. Machine \( M \) specifies an original function \( \lambda : I \rightarrow O \); machine \( M_2 \) realizes function \( g : U \rightarrow W \); and machine \( M_1 \) realizes function \( h : W \times V \rightarrow O \) (see Theorem 3.3). The nontrivial input encoder and output decoder are both simplified to an appropriate selection and distribution of primary inputs and outputs to sub-machines.

The entire information computed by machine \( M_2 \) is used by the machine \( M_1 \) to
compute its own outputs ($\gamma'_I = \gamma_I$). Primary outputs of machine $M$ are connected directly to the outputs of machine $M_1$ ($\pi_I = \pi_O$). The Theorem 3.5 can be rephrased as [20]:

**Theorem 3.7 (Serial bit decomposition with two component machines).** A combinational machine $M(I,O,\lambda)$ has a serial bit decomposition with two partial machines if and only if a set-system $\pi_I$ and set-system double ($\gamma_I, \gamma'_I$) exist that satisfy the following condition:

$$\pi_I \cdot \gamma_I \leq \pi_O$$  \hspace{1cm} (3.2.21)

where $\gamma'_I \geq \gamma_I$.

Machine $M_2$ corresponds to function $g$, and machine $M_1$ corresponds to function $h$ in general decomposition scheme.

Roth-Karp decomposition as special case of the general decomposition scheme with output decoder is shown in Figure 3.14. An empty block with a line through it, depicts an identity function $\pi_I = \pi_I$.

Due to a fact that both, the Ashenhurst’s and Curtis’ decomposition (see Theorem 3.2) are both special cases of the Roth-Karp decomposition, they are also special cases of general decomposition scheme. An interested reader can find more details in [20].

### 3.3 Summary

In this chapter the methods used in the course of research presented in this thesis are presented. The information-driven functional decomposition targeted to gate based technologies was found to be suitable answer to the problem formulated in...
the first section of this chapter. Necessary extensions, modifications of information driven decomposition and auxiliary apparatus were presented in this chapter. They were required to fulfill one of the goals of research presented in this thesis: adaptation of synthesis methodology for a completely different technology target. These methods were primarily used as a base for further extension and modifications, and therefore they are a pivoting point to all algorithms described in this thesis. The information-driven functional decomposition was used successfully in the decomposition targeted to LUT technologies, but there were no attempts to apply it to complex gates libraries. A successful implementation of an experimental tool that proves suitability of the same theory used in combination with gate libraries is, to the best knowledge of the author of this thesis, novel and unique. The proof of this suitability, as well as necessary extensions and additions, is presented in further chapters.
Chapter 4

Information-driven Functional Decomposition targeted to gate based technologies

In this chapter the functional decomposition of combinatorial Boolean functions is presented, with its specific application to circuits for ASIC technology as target of this thesis. Special attention will be given to the precise formulation of problem, for which the solution was found in the research presented in this thesis. Further, the usage of the information theory to control the process of decomposition is presented. This connection is presented in a form of a algorithm framework. These algorithms are controlled by means of information measures. They were all heavily modified algorithms used for LUT technologies, such as input sub-support construction and selection, or created from scratch, such as all sub-function construction algorithms: encodings and direct realizations.

4.1 Problem

The research presented in this thesis is a direct continuation of the former research in circuit theory and automatic synthesis methods performed and/or led by Jóźwiak. In particular, it is an immediate continuation and extension of the research in the information-driven circuit synthesis targeted to LUT based FPGA architectures. In the LUT-based FPGA, the maximum number of inputs of a logic block is limited, but not the Boolean functions that might be implemented by the block. Complex gate library approach, on the other hand, implies more strict set of constraints. Promising results of research presented in [20], in case of symmetric Boolean functions [58], gave a confidence to research a circuit synthesis targeted to technology libraries, which in practical cases contains a large number of gates that implement symmetric Boolean functions.

Functional decomposition methodology of Boolean functions targeted to gate based technologies requires an ambivalent approach to the target technology library
dependency. It simultaneously requires a tight dependency of the notion of *feasibility* to a limited and exactly specified number of Boolean functions in target technology library. On the other hand it requires heuristics and decision algorithms to be as *generic* as possible – to be (completely) independent from the specifics of target CMOS technology. The following section presents the problem and specifies in details the assumptions that were made regarding the network structure and synthesis approach.

4.1.1 The precise research problem formulation

The research presented here is aimed at the development of an effective and efficient logic synthesis method and prototype combinational circuit synthesis tool for *incompletely specified Boolean functions targeting gate library-based ASIC circuits*. With respect to the resulting network structure we assume the following:

1. The target technology library is functionally complete.
2. The ASIC circuits are modeled as gate networks, in such a way that each of their nodes correspond to a particular technology gate from a specified library.
3. Each gate is able to implement a corresponding Boolean function as described in a given technology library.
4. The gates are characterized using information relationships and information relationship measures prior the decomposition process.
5. The gates of a given technology library are first pre-characterized with relevant physical data and adequately modeled in relation to the information relationships and information relationships measures for the purpose of information-driven circuit synthesis based on general functional decomposition.
6. Synthesis methods, algorithms, heuristics sorting criterion’s, etc. are neither biased nor tuned to a specific technology library. They are general and completely target technology independent. However, they process the library- and technology–specific information in such a way that the results of the processing (synthesis decisions) are library– and technology–specific. A target technology library can be for example as “poor” as a single NAND gate, or as “rich” as several hundred different gates, and for each particular library the method and corresponding tool should efficiently produce high-quality circuits.

With respect to the circuit synthesis approach we assume the following:

1. The method will be based on the information-driven approach to circuit synthesis based on the general functional decomposition proposed by Jóźwiak in [49].
2. Information relationships and information relationship measures constitute the basic analysis apparatus that delivers the information necessary for the synthesis decision-making.

\[^{1}\text{The only limitation is functional completeness of a given technology library.}\]
3. The method will implement the general decomposition without loops.

4. The network will be constructed bottom-up; i.e., level by level from the primary inputs to the primary outputs, simultaneously for all the single output functions of a given multiple-output Boolean function.

5. There are three simultaneous direct optimization criteria, namely minimization of three gate network characteristics:
   (a) the network depth (critical path, propagation delay),
   (b) the total (active) area occupied by gates in resulting network,
   (c) the total number and length of interconnections in the network.

   This corresponds to the concurrent speed and area optimization of the resulting network. The method should allow for speed vs. area trade-off, but in the initial version of the tool the gate count and interconnect minimization will be not performed at the cost of a higher number of network levels (i.e. the speed optimization will be of higher priority). The active area optimization consequently leads to implicit static power optimization. The (inter-)connection minimization, including the critical path, on the other hand, implicitly leads to dynamic power minimization.

6. The method and the prototype EDA tool implementing the method have to enable easy modification of the optimization target. Particularly to adopt them to the timing-driven synthesis, i.e. the synthesis that directly accounts for the signal timing information and timing constraints and objectives.

7. Boolean functions to be synthesized are incompletely specified. It is very important to account for the incompletely specified functions, because many practical functions are originally incompletely specified or during the decomposition process are transformed into incompletely specified image function.

8. The main internal representations of Boolean functions used as the basis for information relationship analysis and information-driven synthesis is the set-system representation. Some tabular representation is convenient for other purposes, it will also be used. Transformation between these two representations is straightforward.

9. The method is not intrinsically dependent on any ASIC topology, whatsoever; hence, it is not biased to any particular ASIC architecture. On the other hand, it should directly recognize the key features of the semi-custom ASIC designs.

In the next sections the proposed method is presented in more detail.

4.2 Decomposition method – strategy

General functional decomposition consists of breaking down a complex system of discrete functions into a network of smaller and relatively independent co-operating sub-functions in such a way that the original system’s behavior is preserved, some
specific constraints are satisfied and some objectives are optimized [20]. A sub-function in the network can be any function that satisfies certain specific structural constraints (e.g. regarding the maximum number of inputs). Throughout this thesis, a Boolean function that is directly implemented through a gate in a given (target) technology library will be referred to as directly mappable or feasible. If all sub-functions of a certain logic network are feasible in respect to the target library, the network is feasible in the same respect and can directly be mapped into the actual gate based ASIC circuit, where each logic building block is a gate that implements a desired sub-function. Functions that are not directly implemented through gates in the target technology library will be referred to as infeasible. To implement the infeasible sub-functions, they have to be expressed as networks of feasible sub-functions, i.e. decomposed into such networks, for instance by using the general functional decomposition.

The classical unorganized or top-down scheme of functional decomposition is presented in Figure 4.1. In a single step of the functional decomposition, base function \( f \) is split into two sub-functions: the (possibly multiple-output) predecessor sub-function \( g \) (bound-set function) and successor sub-function \( h \) (image function). This single decomposition step is recursively applied to both the predecessor and successor functions until a feasible network is constructed [20].

Due to a fact that in this approach, both sub-functions resulting from the splitting sub-function \( g \) and \( h \) are decomposed independently and no decomposition order is imposed, the method can potentially produce a (partial) network, where (small) directly mappable parts of the partial network solutions are interleaved with some large infeasible sub-functions. Even though, such an unorganized decomposition approach does not hinder the basic construction process, it greatly impacts the prediction and control ability of crucial (optimization target) characteristics of the constructed structure.

As an example of a partially decomposed circuit, let us consider the circuit presented in Figure 4.2. The original function \( f \) is sub-divided into three main components: image function \( h_3 \) and two predecessor sub-functions \( B \) and \( D \) that were both independently split into two components: feasible predecessor sub-function \( g_1 \) (gate 1) with infeasible successor sub-function \( D \) and \( g_2 \), with \( C \), respectively. Because the sub-networks are not known yet at this stage, it is impossible to determine the signal delay from the primary inputs to points \( C \), \( D \) and \( B \). As a consequence, it is also impossible to evaluate the delay of the entire implementation of \( f \). According to the top-down or not organized scheme, the free processing order of logic blocks will not allow the evaluation of delay to be finished until the entire network becomes feasible. In the successive decomposition step, the output sub-function is decomposed. We have the information about signal arrival time to the primary inputs \( x_7 \), \( x_8 \) and \( x_9 \), but we still cannot determine the arrival time at the inputs \( D \) and \( B \). Consequently, we are unable to determine critical signal path in the network under construction. In contrast, we can notice that, while decomposing \( D \), the critical signal is \( A \). The arrival time of signal \( A \) can be evaluated with accuracy of the delay model being used by the synthesis algorithm.

To achieve a high quality of the constructed networks, a network construction strategy without the aforementioned shortcomings was required. The accurate prediction of all crucial network parameters (e.g. delay, area, power dissipation) is
4.2. DECOMPOSITION METHOD – STRATEGY

Start

for each function $f_i \in f$

$f = f \setminus f_i$

Single decomposition step

$f_i \rightarrow \{h_i, g = \{g_j\}\}$

for each function $g_j \in g$

is $h_i$ feasible?

yes

is $g_j$ feasible?

yes

$f = f + g_j$

no

$\not{\text{yes}}$

$\not{\text{no}}$

$\not{\text{yes}}$

$\not{\text{no}}$

$\not{\text{yes}}$

$\not{\text{no}}$

$\not{\text{yes}}$

$\not{\text{no}}$

$\not{\text{yes}}$

$\not{\text{no}}$

$\not{\text{yes}}$

$\not{\text{no}}$

Stop

yes

$f = \emptyset$

no

$f = f + h_i$

Figure 4.1: Classical top-down decomposition algorithm.
Figure 4.2: Classical top-down network construction example.
possible, if the Boolean function is decomposed in an organized bottom-up manner, i.e. from its primary inputs to primary outputs. The ability to efficiently employ the (footprint-)area and network speed trade-off policies became one of the main objectives of research presented here. In [20, 133, 134], the initial bottom-up decomposition approach for network synthesis was proposed and discussed. In this approach, the network is built level by level, starting from the primary inputs, towards the primary outputs of a given multiple-output Boolean function and its realization circuit. This method assures that the arrival time to each input of the sub-function being decomposed, as well as other timing, power and area-related information is known during the entire decomposition process. This allows for an appropriate design decision-making regarding the structure, timing and other characteristics of the network under construction. The prototype bottom-up construction method developed and implemented in [133] was the preliminary solution used solely for demonstrating the high potential of the bottom-up functional decomposition. The method was further developed [20] to consider also infeasible sub-functions during the decomposition process. For the purpose of research presented in this thesis the method was even further extended and modified to reflect the differences in the technology target of the decomposition. The differences are presented in detail in further part of this Chapter and in Chapter 7.

Figure 4.3 outlines the bottom-up decomposition algorithm presented in detail in [20], and heavily modified for the purpose of decomposition targeted to gate libraries reported in this thesis. Following the bottom-up scheme, the algorithm starts with the first level \((l = 0)\), taking the primary inputs. Using the primary inputs, the first level of sub-function is built. Then, one after another, the successive levels are built. Contrary to the unordered scheme, the algorithm presented in this thesis constructs the network nodes bottom-up. The nodes placed closer to the primary inputs are decomposed sooner than those placed further away. Before a certain single-output function \(f_i \in f_l\) (where \(f_l\) is the set of infeasible functions to be synthesized at level \(l\)) may be decomposed, the support of \(f_i\) is determined \((\text{sup}(f_i))\). The preference is given to the variable \(x_j\) if the origination level (denoted as \(\text{ol}(x_j)\)) is lower than the level \(l\) currently under construction. The levels are counted from primary inputs onwards, starting from zero. For the construction of sub-functions with multi-level physical realization, the impact of a particular sub-function on the critical path depends on the origination levels of its input signals and on the propagation paths (arcs) on the sub-function implementation. The pre-selection of the input variables is not strictly followed, as it is in the case of decomposition for LUT technologies, described in details in [20]. To overcome the problem of limited direct realization and to increase the number of potential sub-function realizations, the algorithm presented here constructs a wider range of supports in comparison with the LUT-targeted construction algorithm (as presented in [20]). When the number of variables is smaller than the number of inputs of an average gate in the target technology library, the algorithm also considers variables available in the current level, to increase a chance of direct physical realization. The final support selection decision is postponed until the physical realizations for the most promising supports are compared in the technology mapping phase.

\[
\text{npv}(f_i) = \{x_j|\text{ol}(x_j) \geq l\}. \quad (4.2.1)
\]
The permissible variables, as defined in [20], \( pv(f_i) = \sup(f_i) \setminus npv(f_i) \) are the ones which are allowed to be used in the construction of a sub-function \( g \) at the level \( l \). Level \( l \) is considered as closed when there is a sufficient number of remaining free-set variables for bound-set construction. Depending on the required optimization strategy this number can range from 2, for strict speed optimization. \( f_i \) is added to the set of functions \( f_{l+1} \) to be decomposed at the next level. If level \( l \) is closed for all functions in \( f \), the entire level is considered closed, and the algorithm starts to build the next level of the network. If sub-function \( g_i \in g \) resulting from the single decomposition step is infeasible, it is added to the set \( F_i \) of functions to be decomposed on level \( l \). It is later re-considered recursively and it is decomposed in the next iteration of the algorithm. If an image function \( h_i \) is infeasible, it is also added to \( F_l \) and is further decomposed.

The actual support selection decision is postponed until the actual physical realizations of the most promising supports are build. The construction is continued when for the most promising supports, the constructed physical realizations do not satisfy the local convergence condition: the non-convergent physical realization is accepted if there is no better, unfeasible option to construct. The actual realization selection is postponed until the physical realization of constructed sub-functions are available, and such features as:

- foot-print area
- propagation paths of all single outputs (it influences the availability levels of particular variables for next steps of decomposition)
- information measures (information distribution), including measures indicating how much:
  - unique information is multiplied among the single-output component sub-functions, and how much
  - redundant information is suppressed.

These measures allow the selection procedure to adequately evaluate the proposed realizations and choose the one which is the most suitable at particular point of decomposition in relation to the required delay vs. area trade-off. In some situations, more than one physical realizations can be selected, due to a fact that not always the proposed supports are mutually exclusive.

In a Figure 4.4, the first step of the bottom-up network construction procedure is presented. As an example of the partially decomposed network, let us consider a network in Figure 4.5. In the first iteration of the algorithm, the infeasible sub-function \( E \) was constructed and recursively decomposed. The feasible gates were constructed in the order: \( g_1, g_2, g_3, \) and \( g_4 \), to follow the bottom-up scheme. Subsequently, the infeasible sub-function \( a \) was constructed and it too is being decomposed recursively starting from its primary inputs, with gate \( g_5 \). It was also decided that the primary input variables \( x_7, x_8, \) and \( x_9 \) would be propagated to the next level. Thereafter, algorithm continues decomposition process until all sub-functions become feasible.

In the consequence of ordered bottom-up decomposition scheme, the infeasible sub-functions can be found always close to the primary outputs. The nodes repre-
4.2. DECOMPOSITION METHOD – STRATEGY

Figure 4.3: Bottom-up decomposition algorithm.
4. INFORMATION-DRIVEN FUNCTIONAL DECOMPOSITION TARGETED TO GATE BASED TECHNOLOGIES

Infeasible sub-function

bound-set U

free-set V

Figure 4.4: Single step of the general functional decomposition.

Infeasible sub-function

Infeasible sub-function

Infeasible sub-function

Figure 4.5: The bottom-up network construction example.
senting the feasible sub-functions are always placed on top of other feasible sub-
functions. This feature allows the accurate estimation of all crucial physical features
of this part of network that lays beneath the level (the network layer) under construc-
tion. All characteristics of the network critical for the delay or area optimization are
known as accurately, as the technique employed to model the delays and loads and
area related information.

4.2.1 Single decomposition step

As the result of a single step of general functional decomposition, the original func-
tion \( f \) is split into two sub-functions (see Figure 4.5): predecessor sub-function \( g \)
and successor sub-function \( h \). The input support of \( f \) is sub-divided into two (not
necessarily disjoint) subsets: bound-set \( U \), being the input support of \( g \), and free-set
\( V \), being a partial input support of \( h \). Outputs of \( g \) constitute the remaining part of the
\( h \)'s support. This single decomposition step is recursively applied to both the prede-
cessor and successor functions until each sub-function in the constructed this way
network can be directly mapped onto a gate from a given technology library. With
every single step \( i \) a new sub-function \( g_i \) is created in the process of construction,
while, at the same time, sub-function \( h_i \) is computed as the results of the same con-
struction processes out of the sub-function \( h_{i-1} \) of the previous stage, and a newly
created sub-function \( g_i \). The starting point is an empty set of sub-functions \( \{g_i\} \),
and infeasible (sub-)function \( h_1 \) equal to the original function \( f \). The construction of
every sub-function \( g_i \) substantially influences the multiple-level network which imple-
ments function \( f \) being decomposed. Every decision made in this step impacts the
properties of the circuit implementing the sub-function \( g_i \) itself, as well as decides
the sub-function \( h_i \). In this way it impacts the overall multiple-level network. The
construction of the sub-function \( g_i \), and input support selection are considered as the
most decisive parts of the functional decomposition process.

Figure 4.6 shows a simplified overview of the decomposition scheme for a single
decomposition step taken from [20] and heavily adopted for the purpose of research
presented in this thesis. Similarly as in decomposition for LUT’s, a simplification of
the input function is, in some specific cases, required to carry on the decomposition
process efficiently. Another possible situation to perform an arbitrary simplification
is when the main bottom-up decomposition algorithm fails to find any good decom-
position for the current function image. If an expansion is required to simplify the
function, the best method of expansion is selected and function \( h_i \) is decomposed
into a set of simpler sub-functions \( f_j \) using the selected expansion. The resulting set
of sub-functions \( \{f_j\} \) is then in turn decomposed recursively.

4.2.2 Symbolic sub-function selection

To implement the multiple valued sub-function \( g \) one needs at least \( l = \lceil \log_2 |\pi_g| \rceil \)
binary functions, where \( |\pi_g| \) represents the number of blocks in \( \pi_g \), i.e., the number
of values of the function \( g \). Each of these \( l \) binary functions requires a logic block to
implement either a binary infeasible sub-function or a feasible binary gate selected
from a given technology library. If \( l > 1 \), the number of \( g \)'s binary functions and logic
blocks required can be further reduced by transferring to \( g \)'s outputs only a part of
4. INFORMATION-DRIVEN FUNCTIONAL DECOMPOSITION TARGETED TO GATE BASED TECHNOLOGIES

Figure 4.6: Single step of the bottom-up decomposition.
information from variables in $U$ that is necessary for computing of $f$. One of possible ways to decrease a number of blocks required for implementation of multiple valued sub-function $g$ is through an appropriate reduction of information represented by $\pi_g$. It might lead to reduction of the number of of set-system's $\pi_g$ blocks and consequently the number of outputs required to implement $l$. To guarantee the function's $f$ behavior realization, any unique information that is not present at the $g$ function's outputs has to be added to the free-set $V$. This makes the required, otherwise missing, information available for further computation of $f$ (see Lemma “Existence of serial decomposition” page 138 in [20]). These extra free-set variables form the set of the repeated variables $R$ ($R = \{x_1, \ldots, x_s\}$). Such decomposition with repeated variables is called non-disjoint. It's example is shown in Figure 4.7, where free-set and bound-set has two common (repeated) variables.

The sub-function construction procedure uses information relationships and measures to decide what information should be transferred from a given support $U$ to the $g$'s output(s) and how this information should be distributed among the different binary outputs of $g$. The selected support $U$ and its corresponding set system $\pi_g$, $\pi_g = \pi_{u} \rightarrow \pi_{g}$, together define the binary-realized multi-valued function of $g$, $G : \pi_u \rightarrow \pi_g$, where each particular value $B_g$ of this function corresponds to a block of the set system $\pi_g$.

Another reason to reduce (unique) information processed in the currently considered sub-function $g$ is lack of direct physical realization. The algorithm for direct sub-function realization implicitly searches for a non-disjoint decomposition. The technology library is extended with pseudo-gates (buffer) having a single wire connecting one of the inputs with output – see Section 6.4 for further details. Their sole purpose is to pass the entire information present, including unique, on one of the input variables through to the one of the outputs. The search for direct realization, treating single-wires as gates implementing trivial Boolean functions, is performed without the need to look for an exception, a special case decomposition. After all, the physical realization cost function sorts out which realization is preferable (see Section 7.5.1 for further details).

Let us define the convergence factor as [20]:

$$\text{conv}(U, R, \pi_g) = |U| - (\lceil \log_2 |\pi_g| \rceil + |R|) \quad (4.2.2)$$

In this section we will assume a single level implementation of sub-function $g$, but for the purpose of circuit synthesis targeted to complex gates libraries a extended measures are presented in details in Section 7.5.1. This convergence factor reflects the actual convergence of a single level sub-function implementation only. To extend this notion for multiple-level implementation of infeasible or virtual sub-function an extended notion of level convergence (see Equation 4.2.3) is introduced to accommodate for a cost factor of convergence being spread among multiple-level realization.

$$\text{conv}_{\text{level}}(U, R, \pi_g, \text{path}) = \frac{\text{conv}(U, R, \pi_g)}{\text{path}} \quad (4.2.3)$$

When the convergence expressed as the difference in the number of input and output variables does not differ, a more sophisticated measure for convergence is used. The difference of number of blocks of input and output setsystems reflects
4. INFORMATION-DRIVEN FUNCTIONAL DECOMPOSITION TARGETED TO GATE BASED TECHNOLOGIES

the information convergence. It reflects the influence of the Boolean function on the information processed in the functional block. Similarly to the signal convergence, as defined in 4.2.2, the information convergence can be defined as:

$$conv_{info}(\pi_U, \pi_g) = |\pi_U| - |\pi_g|$$

Even in case of a low or no convergent block, information convergence allows to compare encodings quality.

In the original method of sub-function selection the minimal-length encoding was used to ensure the highest possible convergence of the constructed logic blocks implementing sub-functions \( \{g_i\} \). Such assumption was adequate in case of LUT targeted synthesis, where every Boolean function is implementable, unless the number of its inputs is lower than the building block limitation of the target technology. The gate-targeted multi-valued sub-function construction procedure, described in detail in Section 7.3.3, uses the minimum code length as the starting point of its search for the best gate realization. It also increases the code length when no good realization was found with a given code length. This approach allows the local minimum of the number of physical gates to be found first and produces a network of the minimum number of inputs to the infeasible block \( h \).

The procedure proposed for finding the most promising decompositions transforms the original multi-valued function of a disjoint decomposition defined by \( \pi_U \) and \( \pi_g \) into a new multi-valued function defined by \( \pi_U^o \) and \( \pi_{g^o} \). The set-system \( \pi_{g^o} \) is a product of output set-systems of either the final physical realization or a set-system of an infeasible, partial solution during the construction process. The set-system computation is described in details in Section 7.1 in Figure 7.1. For a detailed description of the method for merging of blocks of \( \pi_g \) please refer to [20] in Section 6.3.10.
4.2.3 Selection of the physical implementation

A limited number of constructed most promising supports \( U \) along with their corresponding set systems \( \pi_g \), and complete description of the related sub-function (feasible) binary realizations is passed to the final block’s \( g \) multi-valued sub-function selection procedure, which selects the best of them. Due to a fact that the a large number of most promising physical realizations is being compared, there exist both: exclusive and non-exclusive realizations. Exclusive physical realizations are blocking implementation of other realizations, while the non-exclusive ones can effectively be realized and implemented independently, e.g. disjoint supports create a non-exclusive concurrent realizations, while overlapping supports, having a common set of input variables, are creating (concurrent) exclusive realizations. After the mutual exclusiveness dependencies between supports are analyzed, an effective convergence can be established for every non-exclusive physical realization. The effective convergence accounts also for (possible) disjoint decomposition with another gate constructed on the same level.

The selection is based on multiple physical and logical features of all the constructed realizations, such as:

- convergence (and it’s per-level (see Equation 4.2.3) and effective variant),
- the (highest) level of origin of the input variables (the availability level of its output),
- foot-print area (with optional influence of input inverters collapsed into input drivers gates),
- the number of redundant information items transferred from the blocks inputs to outputs and the impact on the number of constructed block,
- the number of unique information items transferred to more than one output of constructed block, effectively multiplying unique information item.

The availability level and foot print area must account for (possible) influence of input inverters collapsed into input drivers gates. Selection procedure is described in details in Section 7.3.2.

4.2.4 Multi-valued sub-function realization

The multi-valued function \( G : \pi_U \rightarrow \pi_g \), is defined by the product set system \( \pi_U \) of its bound-set variables from \( U \) and its output set system \( \pi_g \). The number of the multi-valued function values is equal to the number of blocks of \( \pi_g \). To implement the multiple-valued function in binary hardware, it has to be expressed by a number of binary Boolean functions, each of which can be realized using one or more of the available physical gates from a given target technology library. For the binary realization of the \( n \)-valued function (the encoding of \( n \) blocks of its corresponding \( \pi_g \)), minimum \( l = \lceil \log_2 n \rceil \) binary functions and their corresponding output variables are required, which determines the minimal code length, i.e. the minimum number of outputs in a binary realization of \( G \).
In the course of research presented in this thesis a number of algorithms were
developed to find the (sub-)optimal physical realization of sub-function using gates
from technology library. The sub-function construction algorithm can synthesize a
given sub-function directly onto gates of a given library, or encode sub-function and
implement its encoded version through a recursive decomposition in such a way,
that it is optimized against a selected criterion, either speed or foot-print area. The
encoding algorithms trade the complexity of the sub-function \( g \) with the complexity
of the remaining image function \( h \). One of them is the algorithm that minimizes the
number of terms in the resulting binary sub-functions and maximizes the common
term sharing (derived from the method of Maximal Adjacencies originally developed
by Jóźwiak for the FSM state assignment [69]). This algorithm is presented in details
in Section 7.4.1.

The preferred solution of the sub-function \( g \) realization is a direct synthesis onto
gates from a given technology library. It is performed through the search for aggrega-
tions of gates that together realize \( g \) and comply to the given optimization objectives.
In our case it is constructed through a step-wise construction process of a vector of
Boolean functions. The sub-function \( g \) explicit encoding is only performed if no good
direct realization can be found.

Our synthesis algorithm requires an internal representation for physical gates
composed in a particular manner. Such set of physical gates implementing a multi-
output sub-function is referred to as \( n \)-tuple from now on in this thesis.

**Definition 4.1 (\( n \)-tuple).** A set of \( n \) gates with their inputs in a fixed order connected
to a common subset of the input variables is referred to as \( n \)-tuple. \( N \) single outputs
create a binary output vector (encoded binary output).

The technology gates of \( n \)-tuple with less inputs than in the set of all common
inputs are widened with additional inputs to achieve uniform support representation.
The Boolean function realized by such widened gates’ variants are not account for
additional inputs, hence we refer to them as don’t-care-inputs, or in short DC-inputs.
Each gate included in a particular \( n \)-tuple represents implementation of a certain
single-output sub-function \( g_i \), being a part of the multi-output sub-function \( g \). In general,
a single-output Boolean function that is considered to become a sub-function
within a multi-output sub-function \( g \) can be an incompletely specified function. An
incompletely specified Boolean function can be mapped using more than one gate
from the technology library due to an implementation freedom given by don’t cares
in its logic description. In specific cases, the single output functions \( g_i \) can be completely specified. A completely specified Boolean function can also be implemented
using more than one gate from the complex gate technology library, due to the lack
of obstruction to have more than one implementation of a single-output Boolean
function (e.g. when using complementary gates). It can also be implemented using
different gates implementing the same function that differ in physical features, usu-
ally in the drive-strength (scalability). Summing up, each single-output function can
have more than one technology gate in a given library that implements it. Such a list
of possible candidates, stored in the internal representation of an \( n \)-tuple, gives the
implementation freedom at the final mapping stage.

Having a physical realization for a number of the sub-functions constructed, the
next step is to find the most promising one among them. The solutions are evaluated
with regard to:

- input vector size
- availability level of its inputs
- (predicted) level of their output(s), i.e. network's critical path analysis
- number of repeated input variables
- number of output set-system blocks
- convergence, effective and per-level
- foot-print area

Using this information supplemented with trade-off information among the above criteria the most promising realization is selected.

### 4.3 Summary

In this chapter the fundamentals of the methodology is presented that makes up the framework of algorithms used mainly to implement the experimental synthesis tool IRMA2GATES. Reader also finds out here the explanation of core notions, such as ntuple and definition of (level) convergence of physical implementation of Boolean sub-functions. Before the algorithms and methods that facilitates the process of decomposition will be presented, in the next chapter, the prior art is described to emphasize the novelty of the approach presented in this thesis.
4. INFORMATION-DRIVEN FUNCTIONAL DECOMPOSITION TARGETED TO GATE BASED TECHNOLOGIES
Chapter 5

Related research

To put the presented research into the perspective of existing tools and methodologies in research community and industry, a number of similar and alternative approaches to Boolean function decomposition is presented in this chapter. The main focus is historical background of the theory that was the foundation of this thesis: the information relationships and measures. Further, a selection of synthesis methods are presented as the topic of this thesis is closely related to application of information driven decomposition to ASIC technologies.

5.1 Functional decomposition approaches

In the modern implementation technologies (e.g. modern gate libraries, FPGAs, CPLDs) constraints are not so much imposed on the function type that a logic building block (e.g. gate, LUT, CPLD block) can implement, but rather on the structural dimensions of logic blocks (e.g. the maximum number of serial transistors in a gate or the maximum number of inputs and outputs in a programmable block) and on the interconnections between the logic blocks. For instance, a look-up-table (LUT) of an FPGA is able to implement any function of a limited number of inputs and CMOS libraries include typically all 2- and 3-input and many 4-inputs functions. Also, in modern sub-micron technologies the interconnections more and more often determine the circuit quality and performance (overall active area, speed, power dissipation).

On the other hand, the traditional logic synthesis methods do not consider hard structural constraints. Moreover, they produce some very special cases of possible circuit structures, using some minimal functionally complete systems of logic operators (e.g. AND + OR + NOT), instead of maximal or rich functionally complete systems involving all or most functions satisfying specific structural limits (as represented by the fully programmable logic blocks of LUT-FPGAs or modern gate libraries involving numerous gates).

Additionally, the traditional logic synthesis methods and tools require a postsynthesis technology mapping for other synthesis targets than the minimal functionally complete systems they are based upon. Since the initial logic synthesis is performed without a close relation to the actual synthesis target, technology mapping cannot
guarantee proper final results. These and some other weaknesses in the traditional circuit synthesis methods used in today’s CAD tools caused that the opportunities created by modern micro-electronic technology cannot be exploited effectively. Therefore, new, more adequate circuit synthesis approaches, methods and tools are necessary for the modern synthesis targets. Recently much research has taken place in the field of functional decomposition [6, 13, 22, 34, 49–51, 58–61, 65, 66, 77, 81, 82, 84, 86, 93, 96, 104, 108, 109, 115, 116, 118, 119, 125, 130, 140, 141].

Functional decomposition consists of breaking down a given function \( f \) into a network of interconnected sub-function that realizes \( f \). In practical applications it aims at satisfying certain constraints and optimizing some objectives related to the sub-functions, theirs interconnections and the overall network. General functional decomposition considers any function that satisfies specific structural constraints as a permissible sub-function and is able to directly account for the interconnection structure. This corresponds well with the modern synthesis targets. The functional decomposition approach was proposed by Shannon [124], Povarov [106] and Ashenhurst [3]. It was then extended by Roth and Karp [113], Curtis [29] and Jóźwiak [49, 72].

Jóźwiak proposed a uniform model and theory of general full decomposition that cover both the combinational and sequential circuits and include, as their special cases, all earlier proposed decomposition models and theories. In [49] an extensive discussion can be found on various special decomposition cases covered by the general decomposition model.

Recent progress in functional decomposition has been mainly due to application of Reduced Ordered Binary Decision Diagrams (ROBDDs) and information modeling with set-systems. ROBDD-based functional decomposition of Boolean functions was initiated by Chang and Marek-Sadowska [17], and by Lai and Pedram [82]. They discovered that ROBDD allows for an efficient verification of the existence of a disjoint serial decomposition of a completely specified Boolean function. Additionally, a technique called substitution and reduction was proposed in [17] and was later studied in [130]. In [82], verification of the non-disjoint decomposition was discussed and decomposition method for the incompletely specified Boolean functions was presented. Wurth, Eckl, Legl, et al. [84] proposed a set of implicit algorithms for the predecessor-sub-function support computation and multiple-output decomposition of completely [140] and incompletely specified Boolean functions [34]. Sawada proposed Boolean re-substitution as a method of multiple-output decomposition [119]. Since ROBDDs enable compact representation of many Boolean functions, quite large functions can be decomposed using the ROBDD approach. The decomposition quality, however, very much depends on what is directly “visible” in the ROBDD by a particular variable ordering. Unfortunately, no way was found, so far, for an efficient discovery of the variable ordering for large ROBDDs that would result in an optimal decomposition. Also, in its current form the ROBDD approach is limited to binary functions.

Most of the recent works in the field of functional decomposition are related to FPGAs, but there are also first works in this field addressing the gate libraries [77, 115].

The decomposition approach based on information modeling with set systems was initialized by Hartmanis et al. [39], however, not in relation to Boolean functions, but to sequential machines. It was then adapted by Łuba [86] and Jóźwiak [49, 68, 134, 136] and used by them and their collaborators [13, 22, 49, 58–61, 65, 68, 86,
5.2. MULTIVALUED SUB-FUNCTION REALIZATION

104, 109, 134, 136], as well as by others, for the binary and multivalued function and sequential machine decomposition.

Although set systems enable us to model information in various places of discrete systems (e.g. in particular: inputs, outputs and internal points) and information streams in discrete systems, they themselves do not enable us to analyze the modeled information and relationships between information in various modeled information streams. This was the main reason why the set-system-based approach proposed by Hartmanis did not result in effective and efficient synthesis methods or tools for a quite long time. Hartmanis and others only understood that set systems model information in a vague way. They did not realize however, how the set system does it and what particular information is modeled by a given set system. The break-through was created by Jóźwiak with publication of his theory of information relationship and information relationships measures [50]. While set systems enable us to model information in various places of discrete systems, the information relationships and measures serve for analysis and estimation of the modeled information and information interrelationships. The apparatus of information relationships and measures can be applied to any binary, multiple-valued or symbolic system modeled by any sort of discrete relation, function or sequential machine and can be used in many fields of modern engineering and science, including circuit and architecture synthesis for VLSI systems. It enables designers and tools to analyze information in various information streams and relationships between the information streams, and in this way, provides them with data necessary for effective and efficient design decision-making [50, 51, 58–61, 66, 108].

The circuit synthesis method presented in this thesis differs considerably from all other known methods, and belongs to the class of the functional decomposition approaches based on information modeling with set system. It is based on the information-driven approach to circuit synthesis, bottom-up general functional decomposition [49, 72] and theory of information relationship measures [50, 51]. It does not involve any initial technology-independent logic synthesis followed by technology mapping, but it constructs the circuit directly from gates of a given pre-characterized gate library, using the gate and technology related information (number of gate's inputs, area, delay-related information, etc.) for the synthesis decision-making.

5.2 Multi-valued sub-function realization

In [99] Murgai et al. have noticed that the problem of determining sub-functions \( g \) and \( h \) can be considered as an input-output encoding problem, because functions \( g \) are outputs of the block \( g \) and inputs to the block \( h \). From the viewpoint of the encoding type, one can sub-divide the encoding algorithms into the following classes: [22]

1. **output-encoding**: dedicated to function \( g \) simplification [44],
2. **input encoding**: dedicated to function \( h \) simplification [90, 99],
3. **input-output**: dedicated to concurrent simplification of both sub-functions \( g \) and \( h \) [14, 122].

Different techniques of code assignment to compatible classes were used in previous works as well:
1. strict encoding (unique code assigned to each block of $g$) [44, 82],
2. non-strict encoding (two or more codes can be assigned to a block of $g$) [29, 140].

Moreover, different criteria for evaluation of the encoding quality were proposed in [22]:
1. maximization of the number of the output don’t cares in function $g$ [14, 122],
2. support minimization of function $g$ [44, 140],
3. minimization of the number of compatibility classes in function $h$ [47].

There exist two general alternative methods to realize function $g$:
1. Direct synthesis of the gate network implementing $g$ from the multi-valued function $g$ (i.e. direct mapping of the multi-valued function $g$).
2. Binary encoding of function $g$ and synthesis of the binary sub-functions being result of the encoding.

Ad. 1 According to our best knowledge no works have been published on the direct mapping of the multi-valued function $g$, and consequently it seems that we are the original proposers of this approach.

Ad. 2 Encoding of $g$ and synthesis of the resulting binary functions binary encoding of the multi-valued sub-function $g$ has been researched by several researchers (see [22, 58, 60, 100]) The synthesis of the binary sub-function into gates can be performed using the same general functional decomposition that will decompose the functions into simpler, feasible sub-functions and implementation of the network of feasible sub-functions using complex gate generation or technology mapping into earlier constructed and optimized gates of a given technology library.

5.3 Traditional technology mapping

Combinational logic circuits are very often implemented as multiple-level networks of logic gates. The fine granularity of multiple-level networks provides us with several degrees of freedom in logic design that may be exploited in optimizing area, delay and power consumption or any trade-off among them, as well as in satisfying specific constraints, such as different timing requirements on different input/output paths. The unfortunate drawback of the flexibility in implementing combinational functions as multiple-level networks is the difficulty of modeling and optimizing the complex networks themselves. The structure of a multiple-level combinational circuit, in terms of an interconnection of logic gates, can be described by a logic network. The logic network is an incidence structure relating its modules, representing input/output ports and logic gates, to their interconnection nets. The logic network can be represented by a DAG, with vertices corresponding to network modules and edges representing two-terminal nets to which the original multi-terminal nets are reduced[31].

In the traditional two-step circuit synthesis process that involves the technology independent logic synthesis and technology mapping, the initial synthesis is often performed without close relation to the actual synthesis target, often in a wrong
direction, and the design freedom is used to a large extent on random instead of being carefully exploited for the actual network optimization. Consequently, even a very good technology mapping cannot guarantee adequate final results. Technology mapping is the key link between the technology-independent logic synthesis and the physical design. Technology mapping is also known as Cell-library binding, and it consists of transforming an unbound binary logic network into a bound network, i.e. into an interconnection of components that are instances of elements of a given library [31]. A common approach for achieving library binding is to restrict binding to the replacement of small sub-networks (of the original unbound network) with cell-library instances [74, 89, 114]. This is called network covering by library cells. Covering entails recognizing that a small portion of a logic network can be replaced by a library cell and selecting an adequate number of cell instances from the library to cover the whole logic network. while optimizing some figure of merit, such as area, delay or power dissipation. Mapping algorithms are dependent on the initial network decomposition into base functions. It would be highly desirable to develop algorithms whose solutions depend only on the network behavior [31] or to develop a single-step circuit synthesis process.

Algorithms for library binding were pioneered at AT&T Bell Laboratories by Keutzer [74], who recognized the similarity between the library binding problem and the code generation task in a software compiler. In both cases, a matching problem addresses the identification of the possible substitutions and a covering problem with an optimal selection of matches. To render the problem solvable and tractable, most heuristic algorithms apply two pre-processing steps to the network before covering: decomposition and partitioning.

Decomposition is required to guarantee a solution to the network covering problem by ensuring that each DAG vertex is covered by at least one match. The goal of decomposition in this context is to express all local functions as simple functions, such as two-input NORs or NANDs, that are called base functions. The library must include cells implementing the base functions to ensure the existence of a solution. Indeed, a trivial binding can always be derived from a network decomposed into base functions. Conversely, if no library cell implements a base function $f$, there may exist a vertex of the network whose expression is $f$ and that is not included in any sub-network that matches a library element. Each library gate can be expressed as a specific small logic network of the base function. The choice of the base function is important, especially for approaches based on the structural matching, and it is obviously dependent on the library under consideration.

Different heuristic decomposition algorithms can be used, but attention must be paid, because the network decompositions into the base functions are not unique and strongly affect the quality of the solution. Therefore, it is one of the main reasons for the research presented in this thesis to develop the single-step decomposition scheme, to avoid initial technology-independent network. Heuristics may be used to bias some features of decomposed networks. For example, while searching for a minimal-delay binding, a decomposition may be chosen such that late inputs traverse fewer circuit levels. It is important to stress, that since the partitioning and decomposition steps are heuristic, problem difficulty is reduced. Moreover, their heuristic character can hurt the quality of the solution.

As explained above, the traditional two-step circuit synthesis process that involves
the technology independent logic synthesis and technology mapping often produces inferior synthesis results. To eliminate this problem, we proposed to replace the two-step synthesis with a single-step direct circuit synthesis (direct mapping) into the gates of a given technology library, when directly accounting for the actual implementation costs from the very beginning of the single-step circuit synthesis process. This process requires availability of adequately complete and accurate information on the logical and physical features of the technology gates as well as an effective and efficient usage of this information throughout the whole process from its very beginning. The corresponding data structures must enable an accurate modeling of this information, as well as an effective and efficient search for gates during the sub-function construction process and application of the selected gates in the network under construction. In particular, the single-step synthesis requires:

- an adequate characterization of gates’ logic features and physical features related to area, timing and power dissipation,
- a methodology to efficiently provide correspondence between the representations of multi-valued sub-functions in the decomposition and the functional representations used in the characterization of physical gates from a given technology library.

To guarantee the generality of application of our information-driven circuit synthesis approach to every gate-based technology and every library, the actual circuit synthesis methods, algorithms and heuristics have to be independent of any particular technology or library - the technology or library specific features have only to be used as data for the circuit synthesis methods, algorithms and heuristics.

In the approach presented in this thesis, technology mapping is performed simultaneously with logic synthesis, for each and every sub-function $g_i$, and not for the entire network of sub-functions after the logic synthesis. The process of finding physical representation is performed on a complete sub-function, and not, as it is in traditional technology mapping, on small sub-networks.

Furthermore, the technology mapping presented in Chapter 7 is performed for the multi-valued sub-functions, with binary Boolean functions as the special case of single-output sub-function $g_i$, as opposed to traditional mapping methods, where all sub-networks considered for the covering phase are single-output.

### 5.3.1 Complex gates generation

Nowadays, most of the technology mapping tools is based on matching and covering phases [7, 89]. The first operation, matching, is the capability to recognize the logic equivalence between two logic functions. The second, is the selection of the best solution taking into account some objective parameters. So, technology mapping is the problem of enumerating various groups of cells implementing a logic function, and then select the best group from a pre-characterized library. The performance characterization of a great number of cells (with associated templates) is very expensive. It must be realized and validated for each new technology to be used. This cost of implementation reduced the number of existing cells in a library, limiting the matching and covering possibilities. As a result, standard-cell technology mapping and generation tools exploit only the reduced number of cells and templates contained
in a library. The advantage of standard-cell over automatic layout generation consist of the possibility in optimizing complex functions at the cell level, such as flip-flops and adders [98]. If we consider the great number of different CMOS complex gates available with maximum of 3 or 4 serial transistors (87 and 3503, respectively [112]) it appears that a large number of mapping alternatives is available, resulting in a large space to search performance optimized implementation [110].

In [111] Reis et.al. presented a novel BDDs approach as an alternative to those proposed in [5] to achieve entire control on total and serial transistor quantity in resulting network. Terminal Suppressed BDDs (TBDDs) allow a direct association between edge and transistors, in order to control the overall number of transistors as well as the number of serial transistors. The direct association of transistors with edges avoids explicit application of Morgan's law, simplifying the library free matching phase.

### 5.3.2 Terminal Suppressed BDDs in technology mapping

This novel BDD class is based on the very simple observation, that a CMOS complex gate is a set $S(V_i)$ of variables each of them controlling a pair of switches $[S_0, S_1]$ where $S_0$ is active (on, connected) when $V_i$ is equal to 0 and $S_1$ is active when $V_i$ is equal to 1. This structure is strongly similar to a BDD, but in a BDD there is no separation between the $S_0$ switches (P plan) and the $S_1$ switches (N plan) while in a CMOS complex gate there are two distinct sets of switches. So, we will restrict the topology of a BDD in order to define a special BDD class that can well represent CMOS complex gates.

#### Properties

<table>
<thead>
<tr>
<th></th>
<th>In a CMOS complex gate we can verify the following four properties:</th>
<th>In a TBDD graph we can verify the following four properties:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>Only PMOS transistors are connected to the 1-logic source</td>
<td>Only $S_0$ edges connect the 1-logic source, and so they are suppressed</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt;</td>
<td>Only NMOS transistors are connected to the 0-logic source</td>
<td>Only $S_1$ edges connect the 0-logic source, and so they are suppressed</td>
</tr>
<tr>
<td>3&lt;sup&gt;rd&lt;/sup&gt;</td>
<td>There are two separate sets of switches: NMOS transistors ($S_1$ switches) and PMOS transistors ($S_0$)</td>
<td>All edges arriving to the same node are either $S_0$ or $S_1$</td>
</tr>
<tr>
<td>4&lt;sup&gt;th&lt;/sup&gt;</td>
<td>A gate with more than one input is always composed of a serial/parallel association of transistor</td>
<td>There is always one path that passes through all non-terminal nodes</td>
</tr>
</tbody>
</table>
Rules

Construction of CMOS network from TBDDs is quite straightforward. Each BDD that complies with specified rules for the Terminal Suppressed BDDs represents corresponding CMOS network. Figure 5.1 shows that correspondence on an example of logic function:

\[ F(A, B, C, D) = A \land (B \lor C) \land (D \lor E) \]

This minimized representation of logic function is given by TBDD with maximum cut-set of two and maximum path length of three. In designed CMOS circuit, these numbers correspond to the numbers of MOSFET connected in series of type n and type p respectively. Once the network is constructed, it is easy to predict the size of each transistor by applying simple rules to scale them, knowing only the maximum path each transistor is placed in.

Network construction

As seen in Figure 5.1, network construction is quite straightforward, since Terminal Suppressed BDD complies with enumerated rules and limitations. Maximum number of transistors in series as well as transistor sizing estimation can be extracted directly from the diagram. The overall active area equals to the summary area of all transistor channels neglecting the connection area at this stage of synthesis.

Gates and BDD’s parameters

Once the network was extracted from the BDD, gate parameters such as size (area) and delay time depend on the network topology. Main contribution to the overall gate area is active area and connection. Estimation methods are presented in next chapters (6.3). Area, dissipated power and delay were always considered as trade-off parameters of constructed gate. Methods employed to balance those main parameters must be known to predict the result. In the remainder of this document those methods are closely presented.

More detailed description of properties, usage and features of CMOS gates construction based on analysis of terminal suppressed BDDs can be found in [110].

5.3.3 Technology mapping for cell library

Conventional technology mapping consists of three major tasks. First, Boolean sets are partitioned into an interconnection of single-output sub-networks, with the property that each internal vertex has unit out-degree (i.e. fanout). Then, each sub-network is decomposed into an interconnection of two-input functions (e.g. AND, OR, NAND or NOR). Each sub-network is modeled by a directed acyclic graph (DAG), called subject graph. Finally, each subject graph is covered by an interconnection of library cells. [88]

Matching is the key operation of the technology mapping process. It identifies whether an element of the library can be used to implement a part of a given Boolean function. It can be described as checking the tautology between a given Boolean function, called the target function, and the set of functions representing a library
Figure 5.1: Terminal Suppressed BDDs with corresponding CMOS structure.
element, for any permutation of its (input) variables. We also consider the phase-assignment problem in connection with the matching problem, because they are closely interrelated in affecting the cost of an implementation. Finally, we include the 
\textit{dont'care set} of the target function during the matching operation.

Technology mapping is the choice of the elements from a technology (typically cells from a library) that will be effectively used to implement a given circuit. Traditional technology mapping algorithms are typically divided in three main steps:

- decomposition,
- matching,
- covering.

The decomposition step transforms the initial description of the circuit (typically a DAG) into a forest of trees. This decomposition into trees is common to most technology mapping approaches and the description of the algorithms focuses on the mapping of individual trees [32, 74, 85, 89].

After the circuit DAG is decomposed into trees, each algorithm decomposes the trees into base functions, like NAND/NOT or AND/OR/NOT trees. The data structure representing a decomposed portion of the circuit used to perform the mapping algorithm is called \textit{subject tree}.

The quality of the final implementation effectively depends on this initial decomposition.

A good preliminary decomposition can lead to best mappings, while an unsound one may lead to a low quality implementation with the same mapping algorithm. The matching is a crucial step, which tries to determine which technology elements, like a standard-cell or a complex gate, may be used to implement a set of nodes in the decomposed network. The two major approaches to solve this are structural [32], [74], and the Boolean [89] matching. The algorithm targeting cell generators [1, 5] also includes a matching step, where sets of nodes in the decomposed description are picked to implement complex gates obeying to a restriction in the number of serial transistors.[27]

\textbf{Mapping for completely specified function}

One of the fundamental problems in library based combinational logic synthesis schemes is to answer the question, whether a given Boolean function \( f \) is realizable by any of existing library cells. This involves searching the library for a cell whose inputs, if necessary, may be permuted to derive the function \( f \). Introduction of \textit{Field Programmable Gate Array (FPGA)} further aggravates the problem. Logic mapping targeted to FPGA library is usually more complex than typical technology mapping cases, because of the larger size of the library. This can be readily observed in the case of the ACT1 module developed by Actel Inc whose library comprises of 702 cells, an order of magnitude higher than the standard cell libraries used in technology mapping. \textit{Binary Decision Diagrams (BDDs)} are very much popular for their ability to represent a Boolean function efficiently. However, matching two BDDs under permutations of input variables causes the BDDs to explode very often. This happens because size of a BDD depends very much on the ordering of input variables. The
concept of using signature for verification of equivalence of two circuits was first introduced in [121]. It yields a characteristic value for each of the input variables. Only inputs with the same characteristic value could be matched. Another signature based scheme [121] characterizes the function by its \textit{minterm weight}, \textit{column weight} and \textit{single fault propagation weight}. However, the scheme, as reported by the authors, does not give satisfactory results for certain libraries. For example, the best possible signature proposed in [121] can distinguish nonequivalent 4-input functions with an efficiency around 0.92 only.[121]

\textbf{Mapping for incompletely specified function}

All above presented approaches consider completely specified function to be matched both in library and function under decomposition process. In our approach this is not always strictly the case. To find a gate, within the whole library, which can be used as the physical implementation of a Boolean function, one must find a way to search for completely specified Boolean functions that comply to the constraints given by decomposed function. Introduction of information description could yield possible matching of incompletely specified function under construction with existing gates in library.

\section*{5.4 Summary}

In the traditional, two step decomposition scheme, the quality of final constructed network highly depends on the technology independent algorithms. Mapping algorithms are dependent on the initial network decomposition into base functions. It would be highly desirable to develop algorithms whose solutions depend only on the network behavior [31]. This thesis presents results of research aiming at development of single-step technology mapping algorithms, where the aforementioned shortcomings of traditional methods are missing. In this chapter a number of alternative, multiple-step approaches were presented, to give a reader necessary background facts about the prior art.
Chapter 6

Technology Library Modeling for the purpose of the Information-driven General Functional Decomposition

One of the main goals of the research presented in this thesis was to develop a method and related EDA tool to model complex gate library, with its logic and physical features, in a way suitable for Boolean function decomposition when using the information-driven general decomposition approach to combinational circuit synthesis. The model is used for the single-step technology targeted circuit synthesis, contrary to the commonly used two-stage synthesis. The novelty of such approach is to perform a direct, single-step synthesis into the gates of a given technology library, instead of a dual-step process of technology independent logic synthesis and technology mapping. The advantage of single-step over dual-step approach comes from the fact that the physical characteristics of circuits estimated and evaluated are helping in taking decisions during the decomposition process, otherwise taken arbitrarily or based on coarse measures or purely on characteristics and measures based on logic features. These decisions can be more accurate this way, quicker and obtained after checking much smaller solution space. The single-step direct decomposition process into gates requires comprehensive information on the logic and physical features of the available technology gates. This information must be provided in a way enabling an accurate modeling, effective search and efficient application of the selected gates in the network under construction. This chapter discusses in details the logic gates implementations (Section 6.2.5), with their physical and logical features (Section 6.3.3). Some specific important features that provide aid during the decomposition process and/or allow for area and/or speed optimization, such as symmetries (Section 6.2.2), are discussed in more details. Efficient representations are proposed for these features and their mutual relationships are discussed. Moreover, a special case of internal representation of Boolean function implementation referred to as “virtual gates” is introduced and discussed (Section 6.4.3). Finally, the
main algorithm of the technology library modeling tool is described with its two main parts: the parser and the instantiator (Section 6.5.4).

6.1 Introduction

In general, an integrated circuit (I.C.) technology is a system of means that makes development and production of integrated circuits possible. I.C. technology involves a particular choice of materials that either conduct, insulate, or conduct, depending on some external factors, as well as a related development and production methodology and tools. The materials of the last group are called semiconductors, and the circuit production technology is usually referred to as semiconductor technology. The three kinds of materials interact with each other to allow electrons to flow selectively, and in this way, realize the electrically controlled switches. Two most popular kinds of integrated circuit technologies are MOS (metal-oxide-silicon) and bipolar electronic technologies.

The basic electrical switch is called a transistor. A transistor realized in MOS technology is a device with three terminals: gate, source, and drain. When the voltage between the gate and the source exceeds a certain threshold, electrons start to flow across the transistor between the source and drain terminals. We say that the transistor is conducting or “closed”. If the voltage drops below the threshold, the switch no longer conducts and is now “open”. Coarsely speaking, a MOS transistor can work as a voltage-controlled switch.

Making specific composition of transistor switches, we can construct logic gates that implement various logic operations, such as AND, OR, NOT or more complex gates. In this thesis the term logic gates refers to the physical devices that operate over electrical voltage level ranges represented with symbols like 1 and 0, or true and false. Alternatively, logic symbols can be represented as level of electric current, as well as a sign of voltage or current. These technologies have advantages in certain applications (e.g. very high speed, low noise, high precision, low jitter, high rise/fall transient symmetry requirement). In the research presented in this thesis the most common application is taken into account, and the most generic requirements of digital circuit design are considered.

The logic gates should not be mistaken with transistor gates. Throughout this thesis the term “gate” refers to “logic gate”, and only when explicitly stated to the “transistor gate”.

For example, an AND gate is a circuit with two inputs and one output. It outputs a voltage that corresponds to logic value 1, whenever it determines that both of its inputs are at voltage that corresponds to logic value 1. In all other cases, it outputs a voltage corresponding to logic value 0.

In CMOS technology logic gates can directly implement more complex functions than NOT, NAND or OR. Logic gates implemented in CMOS technology can be defined as follows:

**Definition 6.1 (CMOS complex gate).** A CMOS complex gate is a specific composition of several CMOS transistors which implements a more complex logic function than a simple NAND or NOR. Every composition of transistors, which satisfies a certain set of rules, is a legal gate.
Such rules, depend on the implementation technology. For example, the maximal number\(^1\) of transistors connected in series between either of the power nodes and the output node is one of such rules.

For the current modern CMOS technologies, a typical value of \(k\) is 4. The limit of 4 is in most cases a consequence of scaling the width of channels of transistors. The transistor width scaling is required to achieve symmetrical transient response of both arcs: between positive and negative power rail to the output terminal. Equal (or close to) transient response yields symmetric current capabilities of positive and negative current sources. Value of 4 transistors comes also from the trade-off between the number of transistors, supply voltage and noise margins. When the power dissipation becomes one of the most problematic issue in modern technologies, lowering the supply voltage is one of the main ways to cope with this problem. As a side effect of the lower supply voltage, the lower maximum number of transistors connected in series limits the number of inputs of the Boolean functions directly implemented in a modern technology library. Implementation of “wider” Boolean functions requires appropriate specific connections of simpler transistor structure that can directly be realized as gates for a given \(k\). Implementation of Boolean functions in the form of a multi-stage connection of simple Boolean functions, through factorization, is widely used. Such an approach increases the delays from particular inputs to the output. The approach presented here is based on general functional decomposition and it takes into account the physical parameters of building blocks. Every physical gate can have a wide range of different delays for particular inputs. It is important that the difference between the mutually symmetric inputs is negligible, as W. Nöth, U. Hinsberger, and R. Kolla have discussed in [102]. Usually, a complex-gate library represents a large collection of gates. Several techniques have been developed to perform the physical design step of building a complex-gate from its transistor graph, as the one of the placement optimization presented in [114, 131].

\(^1\)Usually denoted as \(k\).
An ASIC vendor library is usually represented as a so-called *phantom library*. The library cells are represented as empty boxes or phantoms, not by transistor structures, but they contain enough information for the layout design and other circuit synthesis tasks. The customer would only see the bounding box or abutment box in a phantom version of the cell. After the customer completes the layout, he/she hands off a net-list to the ASIC vendor that fills in the empty boxes (phantom instantiation) before manufacturing the chip.

![CMOS complex gate layers](image)

**Figure 6.2: CMOS complex gate layers [128].**

The customer that completes an *ASIC* design using a particular cell library also owns the masks that are used to manufacture his/her *ASIC*. Such a situation is called the customer-owned tooling (*COT*, pronounced “see-oh-tee”). An example of a mask-set for a single gate is presented in Figure 6.2. A library vendor usually develops the cell library using information about the process supplied by a given *ASIC* foundry. An *ASIC* foundry (in contrast to an *ASIC* vendor) only provides manufacturing, with no design support.

Currently, we distinguish three types of semiconductor companies:

- **The chip makers** (like Intel or AMD) who design, manufacture and sell their
own chips. They are often designated as *IDM* (Integrated Device Manufacturers).

- The **fabless** semiconductor companies (like nVidia, Xilinx or NXP) who design and sell their chips but outsource the manufacturing to foundry companies.
- The **foundry** companies (like Taiwan Semiconductor Manufacturing Company (TSMC) or United Microelectronics Corporation (UMC)) who manufacture the chips designed and sold through their customers.

A fabless semiconductor company specializes in the design and sale of hardware or hardware/software devices implemented on semiconductor chips. It achieves an advantage by outsourcing the fabrication of the hardware devices to a specialized semiconductor manufacturer called a semiconductor foundry or “fab”. A fabless company may concentrate its research and development resources on the end market without being required to invest resources in modern semiconductor technology. For this reason they are also known as *IP* companies, because their primary product consists of licenses in patents, trade secrets, mask works, and other forms of *intellectual property* (IP) [139].

A cell library that meets the foundry specifications, is referred to as a qualified cell library. The qualified cell libraries are expensive (possibly several hundred thousand dollars), but a library is usually qualified at several foundries. This allows to shop around for the most attractive production conditions. This means, that for the high-volume production buying an expensive library can be cheaper in a long run than other solutions.

Another alternative is to develop a cell library in-house. Many large computer and electronic system companies make this choice. Most of the cell libraries designed today are still developed in-house, despite the fact that the process of library development is a complex and very expensive task. In both cases, for the circuit synthesis (decomposition) process there is a specific set of features that describes each individual library building block (complex gate) necessary for the synthesis process. The package, containing the delivery that describes in details the library basic building blocks, is usually formed in a set of computer files, and stored for retrieval on a local or remote machine. Before starting the actual circuit synthesis process, it has to be firstly interpreted into an internal form of a data-structure appropriate for further processing during the circuit synthesis.

Each cell in an *ASIC* cell library contains the following information [128]:

- physical layout,
- behavioral model,
- Verilog/VHDL model,
- detailed timing model,
- test strategy,
- circuit schematic,
- cell icon,
- wire-load model,
- routing model.
Nowadays, ASIC design is usually performed using a predefined and precharacterized library of cells. While designing this library, the original library designer had to optimize the speed and foot-print area of each gate without knowing the actual application that the gate will be used for – i.e., how large a wire and fanout load it will be driving in the final product. Being aware of the source and effect of these trade-offs will make it easier to understand how to design a circuit adequately, using the library cells.

In the past, when the manual gate-level entry was usual, designers felt more comfortable with libraries offering a wide range of functions and appreciated the effort made by the library vendors to increase the size of the libraries [128]. Strictly speaking, the smallest set of functionally complete library can be as small as a single 2-input NAND (or NOR) gate. It is sufficient to construct any Boolean function. Nowadays, the commercially available libraries, even though could be that small, usually involve many gates and feature scalable gate implementations. To increase the set of design options, the library contains multiple gate implementations in a wide range of balance between the foot-print area, speed and power dissipation, The wide spectrum of possible design choices enhances the quality of the automated circuit synthesis and place and route tools.

In the next two section we will focus on the logical and physical features that characterize each gate in technology library. The logic features characterize the ability of certain physical gate to perform required logic operation, while the physical features help to analyze the “fitness” of particular physical gate for the instantiation.

6.2 Complex gate logic features

The traditional technology mapping (based on DAG-covering) involves the construction of an intermediate network. It’s construction is performed with no strict relation to the final technology target, and in most cases is performed using logic blocks of an arbitrarily selected logic functionality and physical features [31, 32, 40, 123]. The intermediate network has, however, a significant influence on the quality of the final network. Not having any information of the technology target during its construction has a negative impact on the quality of the final result. To overcome this problem, a single-step direct mapping into the technology primitives (e.g. gates) should be performed, where the network under construction is build in one step, tightly related to both, the logic description of the gates from the target technology library, and the physical features of the gates.

The single-step technology-targeted circuit construction approach proposed in this thesis requires:

- characterization of gates according to the Boolean function classes (see Section 6.2.1),
- characterization of gates physical features (see Section 6.3),
- a methodology to provide an effective and efficient transformation between the representations of multi-valued sub-functions and the physical gates from a given technology library (see Section 7.3 and Section 7.4),
- technology library description in an appropriate form (see Section 6.4).
6.2. COMPLEX GATE LOGIC FEATURES

The technology library constructed in a form of list of all available technology gates with:

- each gate represented using uniform description,
- all its relevant logic and physical features available in efficient data structures throughout the whole gate instance library,

will be referred to as homogeneous technology library from now on in this thesis. The technology library characterization provides all the required data for the single-step circuit synthesis. In this way it allows for the construction of the target logic network using the gates from a given technology library, right from the very beginning of the circuit synthesis process, and accounting for the gates physical features. The characterization must transform the original information and physical parameters of gates into a corresponding internal form suitable for the information-driven general functional decomposition process. In particular, it must prepare the logical characterization of each gate function from the information view-point, to allow the information driven synthesis.

The homogeneous instance library forms a homogeneous search space for the sub-function construction algorithms, which simplifies the algorithms and enhances their efficiency. Each gate has at most $n!$ different applications, expressed with its different representatives, i.e. non-equivalent, distinct functions realized by the gate for all its possible input permutations. Each gate representative is characterized by its input permutation and corresponding Boolean function it implements for this input permutation.

The logic features that characterize every gate in technology library, facilitating the mapping and sub-function constructing algorithms, are described in the successive sections, namely:

- Boolean function realized in gate, represented in a different forms, suitable in different algorithms, including compact minterm representation,
- optional symmetries detected,
- inputs' relations in respect to the base gate (permutation),
- optional input inverters.

These feature list is further extended with additional information obtained during the creation of the virtual gates instances (see Section 6.4.3), additional gates with extra input and output inverters (see Section 6.4.2) and extended gate instances with widened input supports through addition of DC-inputs (see Section 6.2.3). The extra features helps organize gate instance library in such a way that during the construction, the main algorithm finds the required information regarding gates without a need to search through the entire gate instance library all over again. For example, the gate with added input inverters provides a reference to the base gate – without input inverters.

6.2.1 Boolean function classification

The digital circuit synthesis is a complex process in which many operations are applied repeatedly [28]. The efficiency of the synthesis process substantially depends
on the size of a solution search space under consideration. This is also the case of the function matching phase performed during the technology mapping [32], where a function (or only a part of it) to be implemented is matched against the cells from a given library. Sometimes this matching is limited to the cells with a certain maximum number of inputs. The matching search space can be substantially reduced when exploiting various specific properties of Boolean functions, such as their different symmetries. The goal of classifying Boolean functions according to their P and NPN equivalence (see Definition 2.16) is to explore the network under construction, the modifications related to the De Morgan’s laws and simple polarization transformations on the gates from a given technology library. The notion of NPN equivalence defined in Definition 2.16 is in close relation to the homogeneous technology library. The original target technology library description given in the form of an ASCII file is parsed and converted into an internal library model, suitable for an efficient library search with our multi-valued, sub-function construction procedures. The library pre-characterization process prepares and fills the data structures containing all and only the information about each gate of a given technology library that is necessary for an effective circuit synthesis performed through the information-driven general decomposition.

In the most general case, Boolean matching implies resolving if two Boolean functions are the same under negation of inputs, permutation of inputs, or negation of outputs [25]:

**Definition 6.2 (NPN equivalence class).** Boolean functions that are equivalent under negation of inputs form an N-equivalent class, under permutation of inputs a P-equivalent class, and under negation of inputs, permutation of inputs, or negation of outputs, form an NPN-equivalent class.

N transformation changes the polarization of input and/or output signals. In homogeneous technology library the change of polarization is simply realized through the insertion of an inverter. The cost of this transformation is defined by the technology library itself. P transformation is realized through the permutation of gates inputs, to fit the actual application in the network under construction. The input set permutation boils to appropriate connection of input signals to the inputs of a particular gate. Therefore, the implementation of permutation does not influence the cost of the resulting network, neither in additional area, nor the additional delay.

A set of Boolean functions is considered to be NPN-equivalent, if for each two functions of this set there exist a variant of NPN transformations, that transforms one of the functions into the second one. The NPN transformations allow to represent gates in a uniform way where all physical gates are represented by gate instances having uniform structure throughout the instance library. For details please refer to the Definition 2.16

The permutation and input and output negation transformations extend the definition of Boolean matching for the logic synthesis, as follows: two single-output combinational functions $f(x)$ and $g(x)$ (with the same number of support variables) match, when they are NPN-equivalent. During the logic synthesis, a (not) completely specified sub-function is matched against the list of gate representative in the homogeneous technology library. To exploit all distinct input set permutations of available physical gates, a list of gate representative is introduced in the form of a homoge-
The Boolean matching selects a particular representative to be used in the network under construction. The corresponding physical gate is fitted into the network, and adequately connected. The three transformations determine the way a particular gate is connected to the network:

- which input must be inverted using the input inverters – specified as inversion mask,
- which input support variable must be connected to which gate input – specified as inputs permutation vector,
- whether the output requires negation.

The fact that many different 2-input functions may have the same gate-level implementations naturally introduces the concept of \( P \) equivalence. \( P \) equivalence between 2 functions is obtained when it is possible to achieve identical values for both truth table outputs by permuting the function inputs. Functions that are \( P \) equivalent can be grouped into \( P \) classes. For instance, the functions \( f_3 \) and \( f_5 \) in Table 6.1 are \( P \) equivalent. Figure 6.2 shows all the 12 different \( P \) classes of 2-input functions. It is important to note that despite the existence of 16 different 2-input functions, there are only 12 different 2-input \( P \) classes. The circuits used to implement each \( P \) class are also shown in Figure 6.2. Four \( P \) classes are composed by 2 functions, while eight \( P \) classes are composed by only one function. The most important property of the \( P \) equivalent functions is that they all can be implemented with the same circuit.
Table 6.1: Logic implementations for all the 16 different 2-input Boolean functions.

Table 6.2: The 12 different 2-input P classes.
6.2. COMPLEX GATE LOGIC FEATURES

Table 6.3: The 4 different 2-input NPN classes.

(or cell from a library). Therefore, it is possible to implement any of the 2-input functions with a single cell from a library involving one gate implementation for each P class (for more details see [28]).

From the collection of P equivalence classes in Figure 6.3, it is noticeable that even among some different equivalence classes, there exist similarities in implementation. For instance, functions $f_1, f_2, f_4, f_7, f_8, f_{11}, f_{13}$ and $f_{14}$ have gate implementations based on a single NAND gate plus some inverters. These functions may be grouped into a NPN equivalence class. Figure 6.3 shows all the 4 different NPN classes of 2-input functions, one NPN class per line. It is important to note that despite the existence of 16 different 2-input functions, there are only 4 different 2-input NPN classes. There are 2 NPN classes composed of 2 functions, one NPN class composed of 4 functions, and one NPN class composed of 8 functions. NPN equivalent functions can be implemented with the same circuit (gate) plus some inverters (used in the negation operations for the inputs and the output, if necessary). This way, it is possible to explore applications of physical gates from the target library as one of representative gate for every modification defined in NPN operators. This approach is specially useful when the cost of the inverter is insignificant in comparison with the implementation costs of an average technology gate available in the target library. The presence of NPN modifications allows subsequent improvement of constructed network, when the decomposition explicitly (or even implicitly) stacks inverter gates, that can be collapsed into the gate with complementary equivalent gate present in target library.

6.2.2 Expression tree

Among the Boolean function representations very popular are tabular forms, logic expressions and binary decision diagrams. Another representation, especially useful for completely specified Boolean functions with limited number of input variables is the expression tree, also referred to as operand tree. It is especially outstanding in comparison to the three other representations, when it comes to the symmetry detection. Having a function represented as an expression tree, it is straightforward
to find the group symmetries. It requires a number of simple set operations to automate the process if finding a maximal set of inputs, that comprises a group of symmetric inputs.

![Figure 6.4: Example of the expanded gate with one extra DC-input (A).](image)

6.2.3 Permutation representation

The homogeneous technology library requires that every gate is pre-characterized and represented by a set of representatives. The gate representatives define all possible applications of a given physical gate, when the non-symmetric inputs are taken into account. Each representative together with its corresponding physical gate, having a fixed order of inputs, describe one possible gate application. As such they can be screened for a match with the desired Boolean sub-functions and later used in the actual network construction. The representatives related to one particular physical gate differ in the input set assignment. Internally, every representative has a fixed order of inputs, because each refers to the same basic physical gate. What makes the difference, is the input set permutation, which basically translates the external input set indexes into the ordered internal gate inputs.

![Figure 6.5: Simple permutation representation.](image)

To create a uniform representation one must develop a methodology to process Boolean functions in such a way that every implementable function is represented in a set of functions with a fixed input order. Therefore, an efficient method to translate between different representations is necessary, as well as, efficient representation of the fixed input order. Such a representation in the form of an input permutation is discussed in this section.
6.2. COMPLEX GATE LOGIC FEATURES

The P equivalence, presented in Section 6.2.1, gives an opportunity to explore the function input permutations during the matching phase. To effectively represent the input ordering in a particular gate representative, we introduce the input matrix. It acts as a “patch panel”, and allows for all the possible input connections of \( n \) gate inputs to \( m \) external signals, where \( n \) and \( m \) are not necessarily equal, but \( n \) must be smaller or equal to \( m \).

The main aim of permutations in the library representation is to represent all possible applications of a particular gate implementing a particular Boolean function, when an order of inputs is considered to be fixed. In such a case, a pair: the Boolean function of a given gate with a fixed order of inputs and a particular input permutation can be used to represent an application of a gate. All such pairs represent each and every possible applications of the gate corresponding to its entire input permutation set.

The permutation operators include comparison and symbol translation. The permutation representation introduced here allows for a simple translation operation between Boolean functions representations, introduced and presented in details further in 6.2.4, because it boils to (the low-cost) bit-wise operators. Furthermore, the size of the computation data in our case is also small. The typical input size does not exceed 6 (usually 4) inputs.

To reduce the memory usage and computation complexity we proposed a very simple permutation representation. In this representation each input variable order is denoted as a series of labels (see Figure 6.6).

![Figure 6.6: Example of the input variables indices vector.](image)

The process of permutation presented in Figure 6.5 represents the process of labels reordering. When a certain gate representative (instance) is being used during the decomposition process to construct a circuit implementing a given sub-function, its corresponding permutation vector is transformed into the form of the input switch board. All the internal inputs of a physical gate are connected accordingly to the given permutation to the particular corresponding external inputs.

To further simplify the entire process, we proposed the modulo\((n)\) notation, where all variables have assigned a label as a digit in modulo\((n)\) for \( n \) variables considered. This implies the possibility to re-assign (move) a single label to another spot in ordered inputs vector through the use of the simple addition operation. The label index will be increased by modulo\((n)\), the value of shift. To represent the lack of permutation, one must perform no shifts in the basic permutation. Vector of shifts, in such a case, would consist of all zeros (null shift, see Figure 6.7). The aforementioned example (Fig. 6.5) denoted in a shift notation is given in Figure 6.8. The same example denoted in the modulo\((n)\) notation is given in Figure 6.9. The permutation representation presented is also easily applicable for the support expansion, because the equilibrium of sizes is irrelevant during the transformation process. The translation between the different permutation representations of the same Boolean function requires the input symbol reordering.
6. TECHNOLOGY LIBRARY MODELING FOR THE PURPOSE OF
94 THE INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

Figure 6.7: Null shift vector.

Figure 6.8: Simple permutation of Fig. 6.5 represented in *shift notation*.

Figure 6.9: Simple permutation of Fig. 6.5 represented in *modulo(n) notation*.

Figure 6.10: Example of input variable vector permutation with expansion into a wider support.

Figure 6.11: Example of input variable vector permutation with shrinking into a narrower support.
6.2. COMPLEX GATE LOGIC FEATURES

Table 6.4: Compact Boolean function representation.

<table>
<thead>
<tr>
<th>input symbol</th>
<th>inputs</th>
<th>output</th>
<th>signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0 1 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0 1 1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1 0 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1 0 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1 1 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1 1 1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

6.2.4 Compact minterm representation of a Boolean function

A Boolean function can be represented in a number of ways (BDDs [9], expression trees, expressions [5, 117], terms, covers, blankets, true-tables, etc). Every representation has its advantages and drawbacks. For the purpose of the research presented in this thesis, the Boolean function representation has to ensure the following:

- efficient comparison for the equivalence check,
- easy translation between the function's term and minterm representation and vice versa,
- easy computation of the information measures necessary for the information driven decomposition.

On the other hand, due to the small size of the input sets of the typical technology library gates, the memory efficiency is not very critical. Most of technology libraries provide gates of the input size in the range of up to 5 or 6 inputs, with a few exceptions of completely symmetric wide (N)AND or (N)OR gates.

In our case, the small input size of a sub-function, allows to efficiently represent and process the Boolean functions using minterms. The number of inputs in a typical sub-function directly implementable with a gate is less than 6 (usually less or equal to 4). Therefore, the memory requirements to store and process data do not hinder the actual decomposition algorithm.

The size of the data structure to represent a sub-function using minterm representation grows exponentially with input set size. However, due to a fact that the input set size is smaller than 6 (usually even lower or equal to 4) the data structure storage requirements do not represent any issue for modern computers. The minterm representation allows for modeling, storage, processing and usage of the technology library in the final sub-function construction in a very efficient way. The memory consumption and processing complexity is insignificantly low, due to the small sizes of the sub-functions’ input supports.

**Definition 6.3 (Compact minterm representation).** Any Boolean function for which its input order is fixed (e.g. ascending literal order), can be defined with a single vector of output values, referred to as the function signature, or label.
Since each physical gate implements a corresponding completely specified function, the output set-system of a physical gate is actually a bi-block partition. Therefore, every minterm symbol can be uniquely assigned either to the onset or offset. Consequently, a very simple information structure containing either onset or offset specification is sufficient to un-ambiguously represent a completely specified Boolean function (Figure 6.4). The corresponding bi-partition for the function from Figure 6.4 will be written as:

\[ \pi_{ex} = \{ 0, 2, 3 \mid 0, 1, 4, 5, 6, 7 \} \]

To define a Boolean function of \( n \) inputs one must represent the input-output mapping or correspondence. For the sake of simplicity in all kinds of function manipulations (such as: comparison, etc) a common canonical representation must be developed and used. Such representation used in the technology library characterization presented in Chapter 6 has to satisfy the requirements outlined below:

- a function must be completely specified with every input combination having its corresponding un-ambiguous binary output value, either 0 or 1,
- every input combination is represented as a minterm,
- all input combinations are ordered using an arbitrary but the same sorting, e.g. sorted in the ascending (descending) numerical order, when minterm on input combination is considered as a binary number,
- the ordered output values are stored (internally) in the form of a (linear) vector of binary values.
- optionally a human readable form may be required, and for this purpose any form of translation of the signature into a corresponding (a decimal or hexadecimal) integer can be used.

For a Boolean function of \( n \) inputs, the length of a bit-vector representing its signature is equal to the number of all unique minterm input combinations \( (k = 2^n) \). For the modern 32-bit architecture computers, it gives the size of 5 inputs to fill the byte-wise arithmetic logic unit completely. For wider supports, where signature bit-vector significantly exceeds the size of the machines’ computation capabilities, the processing needs to be performed in parts. Every additional input pin causes the signature bit-vector to double in size. Summing up, the performance of the algorithms processing the compact minterm representations for functions of up to 5 inputs is even for modern processors, but it drops exponentially with every additional functions’ input.

### 6.2.5 Compact minterm translation

The compact Boolean function representation cannot be used for larger functions with many inputs due to a too high processing complexity. To overcome this problem, when needed, Boolean functions have to be translated back and forth between the term and minterm descriptions. For instance, a Boolean function representing a sub-function being decomposed is translated into its minterm representation, then...
6.2. COMPLEX GATE LOGIC FEATURES

<table>
<thead>
<tr>
<th>input symbol</th>
<th>inputs ( i_1 )</th>
<th>( i_2 )</th>
<th>( i_3 )</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.5: Term-minterm correspondence.

<table>
<thead>
<tr>
<th>input symbol</th>
<th>minterms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2,3</td>
</tr>
<tr>
<td>1</td>
<td>4,6</td>
</tr>
<tr>
<td>2</td>
<td>0,1</td>
</tr>
<tr>
<td>3</td>
<td>5,7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input symbol</th>
<th>minterms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.6: Minterm representation of desired Boolean function.

In such form is being processed, to be consecutively translated back to one of its term representations. To be synthesized using the gates from the homogeneous technology library (see Section 6.4), a sub-function \( g \) must be first translated into the compact minterm representation using the procedure described below. It can be later translated (explicitly if necessary or implicitly) into the corresponding term representation, through a simple (min-)term merging.

**Ex. 6.2.1 (Minterm representation).** The process of translation into minterm representation of a 3-input completely specified Boolean function is presented in Table 6.5.

The function's compact minterm representation (as e.g. in the example presented in Figure 6.6) can be directly used for Boolean matching, to find a set of the technology gates which implements the desired function.

To perform the Boolean matching phase the bit-vector representing the signature of the desired sub-function is matched against the signatures of all gate representatives available in the technology library. Since the representatives library considers NPN-equivalence classes, the search results in a set of representatives of gates so that each of them implements the desired Boolean function, eventually accompanied with the input and/or output inverters. Since the desired sub-function is completely
specified, the search is performed with the signature: "11000101" or \(0_{10}c5\). A corresponding two-block set-system is constructed with one block (denoted as on-set or '1' block) exactly reflecting the signature of desired function:

"11000101" or \(0_{10}c5\),

while the other block (denoted the off-set or '0' block) is computed as an byte-wise Boolean complement of the on-set:

"00111010" or \(0_{10}3a\).

This set-system represented in the notation used through-out this thesis and in the other literature is given below:

\[
\pi_{ex} = \{0, 2, 3, 4, 6: 0, 1, 5, 7\}
\]

Let us assume the target technology library to be the same as presented in Figure 3.6 used for Example 3.2.1. Because, the library does not contain a gate that directly implements desired function, construction algorithm searches for the best \(n\)-tuple that will become the first level of constructed network.

\[
\begin{align*}
IS(\pi_{ex}) &= \{0, 2, 0, 3, 0, 4, 0, 6, 1, 2, 1, 3, 1, 4, 2, 6, 1, 5, 2, 7, 3, 5, 3, 7, 4, 5, 4, 7, 5, 6, 6, 7\} \\
AS(\pi_{ex}) &= \{0, 1, 0, 5, 0, 7, 1, 5, 1, 7, 2, 3, 2, 4, 2, 6, 3, 4, 2, 6, 4, 6, 5, 7\}
\end{align*}
\]

Information provided by input variables:

\[
\begin{align*}
\pi_{i1} &= \{0, 1, 2, 3 : 4, 5, 6, 7\} \\
IS(\pi_{i1}) &= \{0, 4, 0, 5, 0, 6, 0, 7, 1, 4, 1, 5, 1, 6, 1, 7, 2, 4, 2, 5, 2, 6, 2, 7, 3, 4, 3, 5, 6, 3, 7\} \\
AS(\pi_{i1}) &= \{0, 1, 0, 2, 0, 3, 1, 2, 1, 3, 2, 3, 4, 5, 6, 6, 7, 5, 6, 7\}
\end{align*}
\]

\[
\begin{align*}
\pi_{i2} &= \{0, 1, 4, 5 : 2, 3, 6, 7\} \\
IS(\pi_{i2}) &= \{0, 2, 0, 3, 0, 6, 0, 7, 1, 2, 1, 3, 1, 6, 1, 7, 2, 4, 2, 5, 3, 4, 3, 5, 4, 6, 4, 7, 5, 6, 5, 7\} \\
AS(\pi_{i2}) &= \{0, 1, 0, 4, 0, 5, 1, 4, 1, 5, 2, 3, 2, 6, 2, 7, 3, 6, 3, 7, 4, 5, 6, 7\}
\end{align*}
\]

\[
\begin{align*}
\pi_{i3} &= \{0, 2, 4, 6 : 1, 3, 5, 7\} \\
IS(\pi_{i3}) &= \{0, 1, 0, 3, 0, 5, 0, 7, 1, 2, 1, 4, 3, 6, 4, 2, 3, 2, 5, 2, 3, 4, 3, 6, 4, 5, 4, 7, 5, 6, 6, 7\} \\
AS(\pi_{i3}) &= \{0, 2, 0, 4, 0, 6, 1, 3, 1, 5, 1, 7, 2, 4, 2, 6, 3, 5, 3, 7, 4, 6, 5, 7\}
\end{align*}
\]

A number of alternative physical implementations is found by construction algorithm. One of the most promising realizations consists of a pair of gates implementing all required information to process primary output. Another promising realization is using non-disjoint decomposition, with just a single physical gate and a repeated variables that realizes the remainder of required information. For sake of simplicity and clarity we discard other, less interesting possible realization, and for further analysis in this example let us limit the search space just for these two partial solutions.
Table 6.7: Minterm representation of sub-function $g$ for two alternative variants of physical realization.

1 convergent non-disjoint realization using gates-pair:

- **physical gate** nanf201 implementing Boolean function $a = (i_1 \land i_3)$,
- **physical gate** norf201 implementing Boolean function $b = (i_1 \lor i_2)$.

2 non-convergent realizations using single gate with repeated variables:

- **physical gate** nanf201 implementing Boolean function $a = (i_1 \land i_3)$,
- repeated variable $i_1$, implementing trivial Boolean function $b = i_1$,
- repeated variable $i_2$, implementing trivial Boolean function $c = i_2$.

From now on we will follow two alternative network construction, to compare two exclusive alternative options of network construction. In the actual construction algorithm that is presented here, only one, the most promising alternative is selected. Selection is based on the basis of cost, both, of the sub-function $g$ and predicted cost of physical implementation of block $h$. In ordered bottom-up functional decomposition, we can precisely assess the cost of implementation of already created sub-function(s) $g$. On the cost of un-feasible function $h$, we can only predict on the basis of number of input variables, and roughly based on the complexity of the function itself.

Gate pair that realizes convergent sub-function $g^1$ variant implements the following setsystem:

\[
\pi^1_{g_1} = \left\{ \begin{array}{c} 0 \\ 5, 7 \\ 6 \\ 1, 2, 3, 4, 6 \\ 1 \end{array} \right. \\
\pi^1_{g_2} = \left\{ \begin{array}{c} 0 \\ 2, 3, 4, 5, 6, 7 \\ 0, 1 \\ 1 \end{array} \right. 
\]

\[
IS(\pi^1_{g_1}) = \{ 0, 5, 7, 1, 2, 3, 4, 6, 1, 5, 2, 7, 3, 5, 4, 7, 5, 6, 7 \} \\
AS(\pi^1_{g_1}) = \{ 0, 0, 8, 4, 0, 6, 1, 2, 1, 3, 1, 0, 6, 2, 3, 2, 4, 2, 6, 3, 4, 3, 6, 4, 5, 7 \} \\
IS(\pi^1_{g_2}) = \{ 1, 2, 3, 0, 4, 5, 0, 6, 0, 7, 1, 2, 3, 4, 6, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7 \} \\
AS(\pi^1_{g_2}) = \{ 0, 1, 2, 3, 2, 4, 2, 5, 2, 6, 2, 7, 3, 4, 3, 5, 3, 6, 3, 7, 4, 5, 4, 7, 5, 6, 5, 7, 6, 7 \} 
\]
The product setsystem of $g^1$ is the following:

$$\pi^1_g = \pi^1_{g_a} \cup \pi^1_{g_b} = \{00, 10, 11\}$$

$$IS(\pi^1_{g_a}) \cup IS(\pi^1_{g_b}) = \{0, 1, 2, 3, 4, 5, 6, 7, 10, 11\} \supset IS(\pi_{ex})$$

The signal convergence of constructed $g^1$ sub-function, according to Formula 4.2.2:

$$\text{conv}([i_1, i_2, i_3], \{\}, \pi^1_g) = 3 - [\log_2 |\pi^1_g|] + 0 = 1$$
Table 6.9: Minterm representation of remainder sub-function $h$.

of three physical gates: nanf201, norf201 and nanf251, occupies 80 area units of foot-print area and critical path is two levels long.

We can produce analogous steps for the second possible physical implementation, the non-convergent sub-function $g$ with single physical gates and two repeated input variables. The output set-system of sub-function $g$ is calculated as product of output set-system of physical gate and two set-systems of input variables:

$$
\pi^2_{g_a} = \{0, 5, 7 ; 0, 1, 2, 3, 4, 6\}
$$

$$
\pi^2_{g_b} = \pi_{i_2} = \{0, 1, 4, 5 ; 2, 3, 6, 7\}
$$

$$
\pi^2_{g_c} = \pi_{i_3} = \{0, 2, 4, 6 ; 1, 3, 5, 7\}
$$

The product setsystem of $g^2$ is the following:

$$
\pi^2_g = \pi^2_{g_a} \cup \pi^2_{g_b} \cup \pi^2_{g_c} = \{001, 011; 100, 101, 110, 111\}
$$

The signal convergence of constructed $g^2$ sub-function, according to Formula 4.2.2 is equal to zero. Therefore, the only method to indicate convergence of block $g^2$ is to show convergence through a notion of information convergence according to Equation 4.2.4:

$$
\text{conv}_{in,f_o}(\pi_U, \pi^2_g) = 8 - 6 = 2
$$

In example presented here, two options were considered. Either signal convergent non-disjoint, with two physical gates, both with two inputs, or signal non-convergent, but information convergent disjoint, with just one two-input physical gate, and two repeated variables. Only non-disjoint partial solution leads to decrease the number of input variables to the un-feasible sub-function $h$. The non signal-convergent block is still information-convergent, as defined in Equation 4.2.4.

Because block $g^2$ did not caused decrease of number of input signals to remainder sub-function $h$, the influence of sub-function $g^2$ is purely through translation of
symbols from one representation to another, as well as elimination of redundant information, and multiplication of necessary information.

\[ \text{IRD}(\pi_{i_1,i_2,i_3}, \pi_{e_x}) = \{0[1,0,5,0,7,1,5,1,7,2,3,2,4,6,3,4,3,6,4,6,5,7]\} = 12 \]

\[ \text{IRD}(\pi_{\pi_{e_x}^2}, \pi_{e_x}) = \{0[5,0,7,1,5,1,7,2,4,2,6,3,4,3,6,4,6,5,7]\} = 10 \]

**Obtained three-input sub-function \( h \) can be directly mapped using blf10 gate, implementing Boolean function \( y = (g_a \land (i_1 \lor i_2)) \):**

The resulting gate network, presented in Figure 6.14 consists of two physical gates: nanf201 and blf10, occupies 56 area units of foot-print area and critical path is two levels long.

In this section the logic features that characterize every gate in the technology library were presented. The characteristics presented here facilitate creation of the pre-characterized gate library. The logic features along side with the physical features of every gate describe the fitness and cost of using particular gate in particular circuit. To complete the picture of all features that describe physical gates in pre-characterized library, the next section focuses on the physical characteristics.
6.3 CMOS gate physical features

In the traditional circuit synthesis approaches an initial (simplistic) gate network is usually constructed first, as a result of a direct HDL compilation of another (similar) process. The next step is the technology independent optimization of the initial network performed by the technology independent logic synthesis, which results in an optimized intermediate network. The optimization is performed observing several logic level factors correlated to some degree with the predicted costs of physical realization, such as the circuit area or delay. Subsequently, the intermediate network resulted from the technology independent logic synthesis is used as a starting point for the technology mapping. Here, the actual elements (e.g. gates) delivered by a particular technology are used and technology dependent optimizations are performed. Unfortunately, only the last step has a direct relation to the actual synthesis target. The first two steps rely on a very coarse cost estimations that are often weakly correlated with the actual synthesis target. The direct single step circuit synthesis, described in this thesis, performs the target network construction when directly using both the technology primitives (gates) and the actual implementation costs from the very beginning of the synthesis process. Through direct exploitation of the information provided by the technology library and accounting for the actual costs during the only circuit synthesis step, the resulting network is constructed in direct relation to the actual synthesis target. This results in superior circuits in most cases.

To construct the network in a single step, the homogeneous technology library must provide a comprehensive information on both the logical description of the directly available sub-functions and their physical features. Consequently, the proposed direct single-step synthesis requires a novel approach to the technology li-
Technology Library Modeling for the Purpose of the Information-Driven General Functional Decomposition

The library model has to allow for un-biased design decisions based on optimization objectives during the decomposition, for an easy trade-off tie-break for the user selectable optimization targets, such as footprint area or critical path delay, etc.

The gate physical features considered during the logic synthesis are the following:

- footprint area,
- capacitive input load for every input,
- intrinsic delays for every arc (input/output path), separately rise and fall,
- output driver’s sink and source current capability,
- static power dissipation (if not provided by the technology library active foot),
- dynamic power, proportional to fan-in and activity on inputs.

The static power dissipation is considered to be proportional to active footprint area, and if not provided by the technology library, the area minimization target implicitly minimizes also the static power dissipation. The dynamic power component, if not provided by the technology library, is considered to be proportional to the fan-out, interconnection length and activity factor. Therefore, the interconnection length and number minimization, as the side effect, minimizes also the dynamic power component of the constructed circuit.

6.3.1 Active area

During the circuit synthesis based on the bottom-up general decomposition the already synthesized circuit part can be processed to compute its actual silicon area, delay, power consumption, etc. This information can be used to compare, with a high precision the quality of different realizations of the same sub-function in the same context of the earlier synthesized part of the circuit under synthesis.

In a complex gates library the cells are usually designed to have the same drive strength as a minimum-size inverter. Keeping in mind the logic ratio of \( p \)- and \( n \)-channel transistors, all \( k \) transistors connected in series are made \( k \) times wider to keep the drive strength as shown in the above figure. We can calculate the area of the transistors in a logic cell (ignoring for the moment the routing area, drain area, and source area) in units of a minimum-size \( n \)-channel transistor – we call these units logical squares [128]. We call the transistor area the logical area.

For example, the logical area of a 1X drive cell, \( \text{OA122I1} \), implementing Boolean function \( Z = ((A \land B) \lor (C \land D)) \), as shown in Figure 6.16, is calculated as follows:

- \( n \)-channel transistor sizes: \( 3/1+4 \cdot (3/1) \)
- \( p \)-channel transistor sizes: \( 2/1+4 \cdot (4/1) \)

the total logical area = \( 2 + (4 \cdot 2) + (5 \cdot 6) = 33 \) logical squares

The Figure 6.17 shows a single-stage \( \text{AOI221I} \) cell, implementing Boolean function \( Z = ((A \lor B) \land (C \lor D)) \), with width/length ratios of 8/3, 8/3 and 6/3, respectively. The calculation of the logical area (for output strength 1) is as follows:

- \( n \)-channel transistor sizes: \( 1/1+4 \cdot (2/1) \)
- \( p \)-channel transistor sizes: \( 6/1+4 \cdot (6/1) \)

active area = \( 1+(4 \cdot 2)+(5 \cdot 6) = 39 \) unit squares [128].
To take into account also the drain area and source area above, the estimation rules must consider the source and drain connection sharing in the parallel transistor connection.

The different P/N ratio yields to different cell’s logic area. The fixed P/N ratio makes the area prediction easier, but the utilization of a gate delay models gives a very good results in respect to the gate transition time, and balanced raise and fall response time. This approach explicitly represent the dependency of delay with P/N width ratio and load, and was presented in [30].
The connection area depends mainly on the employed fabrication technology and quality of the routing techniques used to perform the Placement and Routing (P&R) phase of circuit design. The actual P&R phase is out of the scope of this thesis, and therefore, the impact of routing is considered here on making use of statistical
6.3. CMOS GATE PHYSICAL FEATURES

analysis. The number of metalization layers is the number of metal (in modern technologies aluminum \((Ag)\) or copper \((Cu)\)) layers used for the medium and long circuit interconnections.

Each extra metal layer gives more design freedom in creating interconnection resulting in more compact and dense design. Although, there are no Design Rule Checks \((DRC's)\) for minimal distances between metalization masks and poly-silicon and active areas of \(p\) and \(n\) well, there are minimal spacings to vias and contact which are always present within metalization rectangle masks to create interconnection to active elements \([107]\) p.878). Metal connections are also responsible for power transfer to every active element \((gate)\) in the circuit \([41]\). Metal interconnections are the main reason of parasitic capacitances in the circuits where the multi-layer technology is exploited, and in such a way, increase the power dissipation in the smaller, denser, functionally identical solutions of any digital circuit, in comparison with their larger implementation with a smaller number of metalization layers.

It is a trade off between the area and power dissipation in the interconnection section of overlapping section of final implementation \((OTC – over-cell-routing – yields to minimize area, at the cost of other features)\). The routing area, for given design rules with two metal layers, is about 50% of the chip area. For design rules with three metal layers, it is possible to aggressively use the over-the-cell routing to reduce the routing area. There is no consensus in how much area is used in these processes for routing. However, if we consider that it is possible to reduce half the routing tracks, we can assume the routing is for design with three metal layers equals to 25% of the chip area \([98]\).

The general decomposition method presented in this thesis minimizes the number and lengths of interconnections. The method explicitly creates the connections as short and as local as possible, through the algorithm preferring the construction of successive levels with inputs from the levels directly underneath.

This facilitates both placement and routing. It also results in a very good correlation between the area and delay predictions before P&R and actual results after P&R. The function of cost of P&R highly depends on the number and length of connections.

Extra input/output buffers (non- and inverting) and area-delay trade-offs

Although an extra buffer introduces additional delay resulting from its own transition delay, it improves the drive capability on the corresponding path. Buffer insertion is considered as a fanout optimization (gain-based algorithm for the near–continuous buffer libraries was presented in \([80]\)). The algorithm is based on the interdependence of the input pin capacitance and inverter size, which is synonymous to the buffer since the non-inverting buffers are usually constructed out of the inverters. A continuous buffer library contains buffers, which are available in continuous sizes. Each buffer of a given type obeys a size-independent linear delay equation:

\[
\text{delay} = p + l \cdot g
\]  

where \(g\) (referred to as gain) is defined as \(\text{load}/C_{in}\), \(p\) and \(l\) are independent of the size of the inverter and \(C_{in}\) is the input pin capacitance of the inverter, last stage
6. TECHNOLOGY LIBRARY MODELING FOR THE PURPOSE OF
THE INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

delay contribution. Using the concept of gain, a class of fanout optimization problems
with a specific optimization objective is shown to be exactly solvable [80].

The design of a circuit that drives large capacitive load with a minimum delay is
important when driving the off-chip loads. It should also be considered in the large
fanout cases. The properly designed inverter string used as a buffer, significantly
decreases the delay at the cost of an additional area. For the non-inverting buffer,
an odd number of stages depends on the quotient of the input loads (see Equation (6.3.2)):

\[ N = \ln \frac{C_{\text{load}}}{C_{\text{in}1}} \quad (6.3.2) \]

where \( C_{\text{in}1} \) is the input capacitance of the first inverter, \( C_{\text{load}} \) is the load driven by
the entire considered buffer [107].

Sizing of a complex gate is becoming critical in case of a large fan-in gate (having
a large number of transistors connected in series) and required to drive high load.
It results in a gate occupying significant area when scaled according to the rule of
the symmetric drive ability. An alternative solution with an extra output buffer would
cancel the need to scale the main gate transistors to extreme sizes. A single, output
buffer (inverter) scaled to drive the same load, occupies a much smaller area, helping
balance the area/delay trade-off.

**Ex. 6.3.1 (Boolean function matching for gate library).** Consider function \( F \) con-
sisting of one (min-)term presented in Table 6.11. Each of the two inputs and the

\[
\begin{array}{c|ccc}
\text{B} & 0 & 1 \\
\hline
\text{A} & 0 & 0 & 1 \\
1 & 0 & 0 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>B</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.11: Karnaugh table of 2-input function for example 6.3.1.

output can be inverted yielding eight possible variations of the true table 6.11. These
variations are presented in Figure 6.19.

The output of a function under consideration can be complemented to obtain the
negated function as a result of a construction with a buffer/inverter on its output,
hence a two stage solution.

Let us use the logic area method to estimate the size of the active devices of
gates under the construction for each of above proposed encoding. Area scaling
according to the Logical effort of the inverter equals to 1 and 4, respectively, giving
eight possible results shown in Figure 6.20. When the area constraint is the main
optimization parameter of the gate selection, the first solution (Figure 6.20a) is the
best one. The sizing problem arises when the large fanout case occurs, i.e. when
the gate under construction must be scaled up to drive a large number of inputs of
the next network level. The highlighted solution 6.20a gives in such case, i.e. for

\[ \text{Due to a implicit minimization of length and number of interconnections we are not expecting high figures of fan-out.} \]

\[ \text{N.B. All inverters foot-print area is equal to } 3 \text{ units in all equations.} \]
fanout equals to 4 (4 inputs to drive) the active area equals to 32 units. In the same condition the solution with the output inverter (Figure 6.20e) scales up to the active area of 22 units, which results from 10 area units of NOR gate and 12 area units of the scaled inverter. The introduction of the input buffer in some cases gives a relief to the previous stage driver, as its input additional buffer is driving a branch. In literature, this case is considered as a B type buffer, and an extra output buffer as type A [76].
Although, the solution gives the smallest gate as a result, the lack of the output buffer causes that it does not scale as well as the others. Connecting extra inputs (increasing fanout) results in extra delay caused by the low driving capability of the two serially connected pull-down NMOS transistors. The results can be observed as the increase of the fall-time in transient response. Using a complementary function in the case of a large fanout, yields to increase of the delay time (intrinsic delay of the gate itself treated as a gate and the buffer together), but improves the gate scaling ability.

Through the analysis of possible physical realizations of desired Boolean function, an optimizing algorithm, through a series of trade-off decisions, can choose an optimal option for the composed network under construction. Such an algorithm presented in Chapter 7 takes into consideration a number of possible physical realizations with different number of input and output inverters. Depending on a particular input configuration, the input and output inverters can be “collapsed” into their corresponding input drivers’ gates, or the output phase can be made redundant, allowing for the selection of physical realization occupying less area, and/or influencing the critical path of a network under construction to smaller degree than other, alternative solutions.

6.3.2 Delay
Gate delay estimation based on a simple RC delay model
CMOS gates and complete circuits can be modeled using simple RC model, when the accuracy of the intrinsic capacitances and resistances is sufficient when compared with the real-life measurements.

The delay is a result of the pull-up and pull-down resistances, and together with the parasitic capacitance at the output of the cell, , the intrinsic output capacitance) and the load capacitance (or extrinsic output capacitance), . If we
assume a constant value for $R_{pd}$, the output reaches a lower trip point of 0.35 when

$$0.35 \cdot V_{DD} = V_{DD} \exp\left[\frac{-t_{PDf}}{R_{pd}(C_{out} + C_p)}\right]$$

An output trip point of 0.35 is convenient because $\ln(1/0.35) = 1.04 \approx 1$ and thus

$$t_{PDf} = R_{pd}(C_{out} + C_p) \cdot \ln\left(\frac{1}{0.35}\right) \approx R_{pd}(C_{out} + C_p)$$  \hspace{1cm} (6.3.3)$$

The expression for the rising delay (with a 0.65 output trip point) is identical in form. Thus, delay increases linearly with the load capacitance. The load capacitance is often measured in terms of a standard load – the input capacitance presented by a particular cell (often an inverter or two-input NAND cell).

Any logic cell can be scaled by a scaling factor $s$ (transistor gates become $s$ times wider, but the gate lengths stay the same), and as a result the pull resistance $R$ will decrease to $R/s$ and the parasitic capacitance $C_p$ will increase to $sC_p$. The total cell delay then scales as follows:

$$t_{PD} = \frac{R}{s} (C_{out} + sC_p)$$  \hspace{1cm} (6.3.4)$$

The above equation can be rewritten using the input capacitance of the scaled logic cell $C_{in} = sC$ :

$$t_{PD} = R \cdot \frac{C_{out}}{C_{in}} + RC_p$$  \hspace{1cm} (6.3.5)$$

Finally, the delay can be normalized using the time constant formed from the pull resistance $R_{inv}$ and the input capacitance $C_{inv}$ of the minimum-size inverter:

$$d = \frac{(RC)(\frac{C_{out}}{C_{inv}}) + RC_p}{R} = f + p$$  \hspace{1cm} (6.3.6)$$
The time constant $\tau$ (tau),

$$\tau = R_{\text{inv}} C_{\text{inv}}$$  \hspace{1cm} (6.3.7)

is a basic property of any CMOS technology.

In classic approach, delays are measured in terms of $\tau$ as the delay time unit. Above equation explicitly shows direct dependence of driving ability of pull-up and pull-down transistors in CMOS output branches to differentiate the rise and fall response accordingly to the driving strength.

Summing up, the total delay of complex gate can be accumulated into a single equation Equation 6.3.8:

$$t_{\text{total}} = t_I + t_S + t_C + t_T$$  \hspace{1cm} (6.3.8)

where:

- $t_I$ denotes intrinsic delay inherent in the gate and independent of particular instantiation.
- $t_S$ denotes slope delay caused by the ramp time of the input signal.
- $t_C$ denotes connect media delay to an input pin (wire delay).
6.3. CMOS GATE PHYSICAL FEATURES

- $t_T$ denotes transition delay caused by loading of the output pin.

The delay component of complex gate that depends on the load of the output pin

$$t_T = R_{\text{driver}}(C_{\text{wire}} + C_{\text{pins}}) \quad (6.3.9)$$

where:

- $t_I$ is taken from the technology library through the timing parameters:
  - intrinsic\_rise
  - intrinsic\_fall
- $t_T$ is taken from the technology library through the timing parameters:
  - rise\_resistance
  - fall\_resistance

6.3.3 Power dissipation

Energy (power) is consumed in a system when it is switched on, i.e.:

- during its start-up
- during its actual operation
- in its standby mode.

In standby mode, a system is not operating (its signals are not changing), and consequently, it only consumes the static power. During the actual operation a system consumes both the static and dynamic power.

There are two components of power dissipation in CMOS circuits:

- static power,
- dynamic power.

Power dissipation is becoming the blocking issue in modern CMOS technologies. In the 90nm technologies, roughly 30 – 40% of the total power consumption is attributed to the static power, with leakage power being the dominating component of the static power. Due to the high leakage power that quickly increases with each technology generation, in the future technologies the static power consumption is expected to dominate the dynamic power (unless no new drastically better transistor technology will be introduced). Since the dynamic power is dissipated during the actual operation and static power, also when a device is not operating but switched on, the energy consumption component due to the increasing leakage power will grow even faster than the dynamic power component.
6. TECHNOLOGY LIBRARY MODELING FOR THE PURPOSE OF
THE INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

Static power

Static power dissipation derives from constant current flow through switched off active devices. The static power dissipation breaks down to the following components:

- Reverse-bias leakage current through parasitic diodes formed by source/drain diffusion and n-well diffusion.
- Sub-threshold conduction (current that flows when \( V_{in} < V_{thn} \)), which become more important as power supply is scaled down.
- Gate leakage due to the tunneling effect current through the gate dielectric of a MOS transistor.

Since in static CMOS one branch is conducting when the other is off, the latter one limits the static current. Ideal MOSFET switch model must be replaced with one that takes into account also off-mode channel resistance or more precisely sub-threshold mode current. The MOSFET transistor operating in this region, is said to be in the weak inversion region. This current is mainly due to diffusion current between the drain and the source, similar to the bipolar junction transistor (BJT). The total drain current \( I_{DS,tot} \) is the sum of the weak and the strong inversion components. We know that the strong inversion component is zero when \( V_{GS} < V_{THN} \).

The weak inversion component is modeled by: \[ I_{subth} = A e^{-\frac{q}{kT}(V_G - V_S - V_{TH0} - \gamma' V_S + \eta V_{DS})} \left( 1 - e^{-\frac{q V_{DS}}{kT}} \right) \] (6.3.10)

where

\[ I_{exp} = \mu_0 C_{ox} \cdot e^{1.8} \cdot \frac{W}{L} \frac{D_L}{D_L} \left( \frac{kT}{q} \right)^2 e^{1.8} \] (6.3.11)

\( V_G, V_D \) and \( V_S \) are the gate voltage, drain voltage and source voltage of the transistor respectively. The bulk is connected to ground, \( V_{TH0} \) is the zero bias threshold voltage. The body effect for small values of \( V_S \) is very nearly linear. It is represented by the term \( \gamma' V_S \), where \( \gamma' \) is the linearized body effect coefficient, \( \eta \) is the DIBL coefficient, representing the effect of \( (V_{DS} = V_D - V_S) \) on threshold voltage. \( C_{OX} \) is the gate oxide capacitance. \( \mu_0 \) is the zero bias mobility. \( n \) is the subthreshold swing coefficient of the transistor [19].

The sub-threshold slope parameter \( n \) is given by \[ n = n_0 + n_B V_{BS} + n_D V_{DS} \] (6.3.12)

The static power dissipation caused by leakage current depends also on input states. Low supply voltage requires the device threshold to be reduced in order to maintain performance. As the device threshold voltage is reduced, it results in an exponential increase of leakage current in the sub-threshold region. The leakage power is no longer negligible in such low voltage circuits.

At the system-level design the leakage power can be estimated by the following formula [15]:

\[ P_L = k \cdot N \cdot I_L \cdot V \] (6.3.13)
Dynamic power (switching)

Dynamic power is due to changes of system's signals, and is composed of:

- switching power - due to charging and discharging of the internal (parasitic) capacitances of the transistors and interconnections,
- short-circuit (crow-bar current) power - due to the simultaneous conduction of the CMOS P and N counterparts creating a direct conducting path from the power supply to the ground for a short time.

The first component is a dynamic power component, $P_d$, which corresponds to the charging and discharging of the load capacitance. Assuming a step input and a repetition frequency of $f_p$, the average dynamic power, $P_d$, is expressed as:

$$P_d = a \cdot C_L \cdot V_{dd}^2 \cdot f_p$$  \hspace{1cm} (6.3.14)

For sake of simplicity the analyzed circuit contains a single (lumped) capacitor, representing total capacitance of input gates of all MOS structures. Scaling transistor dimensions causes decrease of dissipated power due to a quadratic decrease of these intrinsic capacitances. Greater scaling impact gives supply voltage decrease. The consequence of supply voltage decrease is the decrease of thresholds and margins, because they scale linearly with supply voltage. To achieve a high level of stability a trade-off must be achieved between parameters in equation 6.3.14. For fixed fabrication technology frequency and supply voltage can be tinkered to meet the temperature specification of used fabrication technology. Temperature of active area of an CMOS circuit depends on dissipated power, or more precisely power density.
Switching power dissipation is highly input activity dependent. On the logic level design no sufficient information is available to process and considered to control the switching power factor. Since the switching power depends on the capacitance of loads, the gates inputs and interconnections, it is therefore indirectly optimized through optimization of interconnects and fan-outs. Area minimization induces switching power minimization. With every new technology generation, dynamic power substantially decreases due to the supply voltage and oxidation film capacitances reduction. However, the dynamic power growth, due to the higher operating frequencies and signal switching activities, often compensates its reduction. Consequently, dynamic power remains a significant component of the total power consumption.

Short-circuit power (crowbar current)

Slow rise and fall times will increase crowbar current of driven gates. In [101] a closed–form expression for short-circuit power dissipation, which takes short-channel effects into consideration, of CMOS gates is presented. The calculation results show good agreement with the SPICE simulation results over wide range of load capacitance and channel length. The change in the short-circuit power, $P_S$, caused by the scaling relation to the charging and discharging power, $P_D$, shows that basically power ratio, $P_S/(P_D + P_S)$, will not change with scaling if $V_{TH}/V_{DD}$ is kept constant.

System power reduction approaches

The techniques to reduce the system power and energy consumption can be subdivided into:

- technology techniques
- circuit design techniques, and
- system design techniques.

Technology techniques comprise all sorts of solutions related to semiconductor material and process technologies that decrease the power/energy consumption, as application of special materials or specific technological process or device parameters (e.g. high-K materials, different gate dielectric thickness or transistor length in different transistors, etc.).

Platform design techniques involve all kinds of platform architecture, circuit and usage mode design solutions that save or enable savings of power/energy (e.g. parallel architectures, specific low-power IP blocks or interconnect organizations, multiple clock and/or voltage domains, gated clocks and/or sleep-mode circuitry, etc.)

Fortunately, the power-saving circuit and system design techniques have also a high power reduction potential, while often delivering the power savings on a much lower price. The power-saving circuit design techniques comprise various techniques that decrease the dynamic and/or static power consumption and include:

- adequate data-path and FSM synthesis (e.g. data-path and FSM decompositions, FSM state assignment, usage of data-path units with application-specific word-length, etc.) to minimize both the static and dynamic power,
6.3. CMOS GATE PHYSICAL FEATURES

- circuit area and interconnect minimization to reduce both the static and dynamic power,
- switching activity minimization (including clock gating) to reduce the dynamic power,
- leakage power reduction, including: passive leakage power reduction through input vector identification that minimizes leakage in standby mode, active leakage power reduction through adequate signal polarity selection for logic blocks, and leakage aware placement and routing, etc.

All of the techniques named above are tackled, either directly or indirectly, during the information-driven general decomposition described in details in this thesis.
6. TECHNOLOGY LIBRARY MODELING FOR THE PURPOSE OF THE INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

6.4 Homogeneous library representation

One of the main goals of technology library pre-characterization step is to prepare an efficient gate representation of all physical gates in a form that is usable during functional decomposition. All gates need to be represented in homogeneous manner to facilitate construction process during search and selection. Homogeneous library developed during the research presented here is aiming at this goal. In this, each physical gate is represented by representatives. All representatives create a mean to uniformly present all physical gates regardless the number of inputs and inputs permutation. This form creates possibility to match each individual technology gate in a number of ways, that differs regarding the input permutation, but also allows to easily compare suitability of gates with a different numbers of inputs.

Similar approach was introduced independently in 2003 by Alan Mishchenko and referred to as supergates [94]. The approach is similar, e.g. the pre-generation performed once per technology library. This is why the supergate generation has an additional advantage of reducing the total run-time of mapping by pre-computing and re-using the mapping information, which depends on the library but not on the netlist to be mapped [94]. The goal here is to obtain a cost function of physical implementation of a particular Boolean function, in terms of number of levels of critical path or foot-print area, depending on the optimization target.

6.4.1 Full spectrum of Boolean functions

For a given number of binary input variables \((n)\), the number of all Boolean functions of \(n\) variables is given by Equation 6.4.1.

\[
Q_n = 2^{2^n}
\]  

(6.4.1)

This number of functions includes also the constant functions and identify functions, and therefore, it shows the magnitude, but do not exactly the number of useful \(n\)-input Boolean functions.

Each \(n\) variable function has \(2^n\) possible minterms, resulting in a truth table with \(2^n\) symbols, each representing distinct input combination (minterm). This is shown in Figure 6.12 for the case of 2-input Boolean functions. The \(2^n\) possible minterms, or lines of the truth table, give the number of bits in each column of the truth table.

This way, the output columns of the truth table characterize a given Boolean function as a binary number of \(2^n\) bits. As there are \(2^{2^n}\) numbers on \(2^n\) bits, there are \(2^{2^n}\) possible different functions of \(n\) inputs resulting in a truth table. All possible input-output mappings are presented in Table 6.12.

A short-hand naming scheme is proposed (see [28]) to efficiently, quickly and un-ambiguously describe each Boolean function of a small number of inputs. For the purpose of the research presented in this thesis, where the number of inputs of a feasible sub-function (usually) does not exceed 4, the naming scheme requires for an \(n\)-input function a vector of bit of size equal to \(2^n\), from now on referred as the function signature or label, and denoted as hexadecimal number, with leading \(0x\), as usually presented in (computer) literature. This naming scheme is designed in a close relation to the compact minterm representation (see Section 6.2.4). The mutual similarities are explored during both, the technology library pre-characterization and
### 6.4. HOMOGENEOUS LIBRARY REPRESENTATION

#### Boolean input vector

<table>
<thead>
<tr>
<th>Boolean function</th>
<th>input vector $AB$</th>
<th>$f_0$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{12}$</th>
<th>$f_{13}$</th>
<th>$f_{14}$</th>
<th>$f_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>base-n short-hand naming scheme (signatures)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>dec</td>
<td>bin</td>
<td>hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x0</td>
<td>zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0x1</td>
<td>$A \lor B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0x2</td>
<td>$A \land B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0x3</td>
<td>$A$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0x4</td>
<td>$A \land B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0x5</td>
<td>$B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0x6</td>
<td>$A \oplus B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0x7</td>
<td>$A \land B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>0x8</td>
<td>$A \land B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>0x9</td>
<td>$A \oplus B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>0xA</td>
<td>$B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>0xB</td>
<td>$A \lor B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>0xC</td>
<td>$A$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>0xD</td>
<td>$A \lor B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>0xE</td>
<td>$A \lor B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>0xF</td>
<td>one</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.12: All the 16 different 2-input functions.

During the general functional decomposition, one of the important features of the naming scheme is the following: when a Boolean function is inverted, its base-$n$ short-hand label is equal to ones’ complement of the original label. For example, negation of $\oplus$ (XOR) function represented as a label 0x6 is 0x9, which in turn, is equal to a label of $\oplus$ (NXOR) function.

The most important reasons of constructing the naming scheme presented for the internal representation of Boolean functions, is its (storage) efficiency, ease of manipulations, and obviously, un-ambiguity.

Table 6.13 shows the total number of Boolean functions of up to 6 inputs. Their number is related to the number of their inputs, but not all of the Boolean functions are useful. Two of them are very specific functions: represent constants, either constant true or constant false. Other ones represent simple (direct) identity relations between single inputs and output and negations of a single input. Such functions can be simply implemented in virtual technology library as interconnections, buffers or inverters. As such, these functions do not perform any significant transformation of information from its inputs to its outputs. In the $n$-tuple construction algorithm, the buffers are used to mock the repeated variables and implicitly create a non-disjoint variant of decomposition.

#### 6.4.2 In/Output inversion(s)

The polarization invariance of information is fully explored when the gates from a given technology library are modified with a special input polarisator. Equipped in the permutation vector (input’s switching board shown in Section 6.2.3) it creates a
complete methodology to produce an extended gate library. Together with the input inverters every physical gate can be coupled with an output inverter to negate the Boolean function it realizes.

The aforementioned inverters coupled either with the inputs or outputs of the physical gates, create an implementation freedom of a polarization invariance. When the gate selection process is finished, the virtual gate together with its coupled input/output inverters is being placed into the network under construction. The De Morgans laws and simple polarization change are explored to simplify the network during the functional decomposition stage.

6.4.3 Virtual gates

The practical technology libraries do not contain gates for some functions having at most \( n \) inputs. Consequently, only a fraction of all possible Boolean functions having at most \( n \) inputs can be directly mapped using the physical gates of a given library. The directly mappable functions always include the completely symmetric Boolean function of (N)AND and (N)OR, together with (input) inverters that greatly increase possible applicability of these gates. Nevertheless, a vast majority of Boolean functions, for example of up to 4 variables is not covered. To facilitate the accurate assessment of the area, timing and power dissipation costs during the decomposition process, we decided to create an extended gate library consisting of the actual physical gates and virtual gates. The virtual gates represent optimal complete decompositions into physical gates of the simple (single-output) Boolean functions of up to \( n \)-inputs that are not directly implemented with the physical library gates and inverters. The extended gate library includes implementations of all single-output Boolean functions of up to \( n \)-inputs.

It is possible, but not necessary, to build all the virtual gates and the maximal functionally complete extended library in advance. The extended library can also be build stepwise according to the actual needs during the decomposition process of

<table>
<thead>
<tr>
<th>number of ...</th>
<th>functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputs</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>65536</td>
</tr>
<tr>
<td>5</td>
<td>approx. 4.29 ( \times ) ( 10^9 )</td>
</tr>
<tr>
<td>6</td>
<td>approx. 1.84 ( \times ) ( 10^{19} )</td>
</tr>
<tr>
<td>7</td>
<td>approx. 3.40 ( \times ) ( 10^{25} )</td>
</tr>
<tr>
<td>8</td>
<td>approx. 1.15 ( \times ) ( 10^{37} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( n )</td>
<td>( 2^n - 1 )</td>
</tr>
</tbody>
</table>

Table 6.13: Number of all possible Boolean functions in a function of number of input variables.
functions that require particular functions of up to \( n \) inputs that are not implemented with physical gates or earlier constructed virtual gates.

For every single-output Boolean function, that is not directly implementable using a single physical gate, the multi-gate implementation is search for in the form of a *virtual gate*. It is done so, to facilitate the cost prediction procedures with accurately computed foot-print-area and delay costs of the single-output Boolean functions constructed as larger blocks. It is also used as pre-fabricated drop-in placement when the synthesis is finished.

The internal structure of a virtual gate gives complete and accurate information about the optimal multi-level and multi-gate realization of a particular constructed convergent block implementing a certain single output Boolean function.

**Construction method**

Multilevel virtual gates are created by constructing and selecting the best of the following options:

- complementary of a physical gate: area enlarged by the area of inverter, the number of levels incremented by one of the output inverter (Figure 6.27) - delay equals to the sum of the corresponding gates intrinsic delays,

- combination of two or more (\( n \)) physical gates, driving an output physical gate having \( n \) inputs; the area consists of the sum of areas of all the physical gates, the number of levels computed as the result of critical path computation in the
6. TECHNOLOGY LIBRARY MODELING FOR THE PURPOSE OF
THE INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

Figure 6.27: Complementary gate of physical one.

Figure 6.28: Combination of $n$ physical (virtual) gates driving the $n$-input output gate.
network resulting in the virtual gate network (Figure 6.28) - delay equals to the sum of the corresponding gates intrinsic delays on the critical path,

- combination of two or more \( n \) virtual gates, driving the output gate having \( n \) inputs; the area, the number of levels and delay computed accordingly (Figure 6.28).

The selection algorithm takes into account several aspects, presented in the Figure 6.29. Depending on the optimization criterion, the realizations are compared using the presented decision flow. Several different realizations are created using the methods presented above and using the optimization criterion. The best of them is selected, and stored for further usage. Such simple decision rules are sufficient for the strict local area or speed optimization. The virtual gate library consisting of each Boolean function realized strictly according to given optimization criteria. It can be used for the circuit construction following the same optimization criteria.

Usage

A library of gate instances extended with the required virtual gates is used as a homogeneous Boolean function library during the function decomposition process. Every instance provides information about its area, as well as the number of levels and delay cost of a particular multi-gate implementations of a corresponding Boolean function. The costs vary within wide boundaries, as presented in Table 6.30. In the decomposition targeted to LUT FPGAs, the cost of implementation of any Boolean function with up-to \( n \) inputs, where \( n \) denotes the width of LUTs, is uniform. Every Boolean function, out of all (exceeding the 65 000), can be implemented using a single LUT primitive\(^4\). Such block occupies uniform foot-print area and influences the critical paths with a uniform number of levels.

On the other hand, in case of an ASIC targeted decomposition, the foot-print area and the number of levels (delay costs) can be as low as a single area unit and single level (inverter delay), and as high as areas and levels (delays) several times larger than the smallest. Refer to Figure 6.30 for areas and delays across the spectrum of Boolean functions of three inputs.

6.5 Library parser

In our approach, not any single feature related to a specific technology library is hard-coded into any of the algorithms or heuristics. The technology or library specific features constitute a part of data for the algorithms and heuristics. The features of a particular CMOS technology and corresponding library are this way exploited during the decomposition to optimize the resulting circuit, but the decomposition method is general and can be used for any gate library. To insulate the process of decomposition from a specific technology library, the library is preprocessed in the initialization phase. In this section, the reprocessing phase is presented, where the technology library represented in a form of a text file is interpreted by the library input parser, to

\(^4\)With exception of trivial Boolean functions implementable with a single wire connection between one of its input and single output.
Figure 6.29: Decision flow diagram for construction algorithm of virtual gates.
provide the data required by the information-driven general functional decomposition process.

First, we will take a closer look at the logic expressions used for the internal Boolean function representation. Further, we will get familiar with the logical and physical features of CMOS gates that characterize them and determine the decisions during the decomposition process.

In the research presented in this thesis the library parser was developed and tested on a number of technology libraries, varying in size and application. Most of the experiments presented in the appendices were performed using the MCNC technology library, for the comparison with other tools, such as Berkeley’s SIS. Commercially available libraries tend to have richer and larger offering of technology gates. Such library used as the example of rich commercially available technology library is Austria Micro System AMS c35b3 0.35 µm.

### 6.5.1 Internal representation of logic expressions

**Definition 6.4 (Formula tree).** A DAG representation of a Boolean relation with nodes carrying either operator or input variable, where all operator nodes have at least one child-node and input variable node neither child-node, is referred to as subject graph, formula tree, expression (or operator) tree in this thesis. Every internal node represents operator, while each leaf node represents operand to these operators or constants. In computer science it is also known as abstract syntax tree (AST).
The method of translation from the input format (usually the flat representation of Boolean formula) involves the string parser which outputs the tree structure containing the resulting formula tree. The most popular algorithm is Shunting Yard algorithm presented in Figure 6.32.

The internal function representation of a library gate is of low importance for the speed of the actual circuit synthesis process. It is important for the library pre-characterization only, and is needed for the instances creation. As such, it is used for calculation of a true table, which for small number of inputs (usually up to 4 inputs) is not computationally costly. For gates of size 6, 7 or 8 being mostly symmetric, the size of a corresponding tree is small as well.

The formula tree like structures are being used for the Boolean matching and covering problem solutions [18, 73, 74, 79]. The approach presented in this thesis uses the equivalence recognition based on the equivalence of the compact minterm representation (presented in Section 6.2.4). The formula tree (DAG) representation is used here for the preliminary symmetry detection. It is much easier to manipulate and process than a (flat) logic expression.

The approach presented here involves analysis of a function represented as a tree. In literature [79] it is also known as operator tree since the root node represents the root operator of the entire formula and all nodes below represent either operands (input(s)) or child operators. Each operator node with its child-nodes represents a complete hierarchy of a formula retaining priorities of operators. Single child nodes represent unary operators, such as inversion.

Relation between an operator tree and a complex gate, both implementing a given Boolean function, boils to the fact that the structure of an operator tree reflects the internal structure of a corresponding complex gate. Internal structure of a complex gate in its canonical form is isomorphic with its Boolean function representation in the form of a corresponding operator tree. The exceptions include polarization variants, of all inputs and outputs. According to the De Morgan laws, the binary operators duality allows for function transformations. Using properties of Boolean algebra, all Boolean functions can be transformed according to their associativity, commutativity, absorption, distributivity and completeness.

N.B. Construction \((A \lor \overline{A})\) gives constant logic 1 and as a result complete irrelevance of the first input \((A)\) to the computation of the Boolean function. In conjunction with logical AND with the rest of formula it creates don’t-care condition which gives opportunity to describe DC inputs altogether in the uniform formula represen-
Figure 6.32: Shunting Yard algorithm.
tion. Keeping narrower gates in the uniform-width gate library among “native” wide physical gates helps to search for solutions using the single homogeneous library, implicitly creating non-disjoint decomposition.

6.5.2 Library pre-characterization

The library pre-characterization process prepares and fills the data structures containing all information about each gate of a given library that is necessary for the effective circuit synthesis performed through the information driven general functional decomposition. At every particular step of the general functional decomposition a relevant part of this information is used to find the most suitable gate implementations of the multi-valued sub-function constructed in this step. In our approach, the library characterization involves finding all possible functions realized by each gate for its all possible input permutations. This information is stored in a form that is suitable for the future comparisons with the data computed during the general functional decomposition process. Each of the library gates has at most $n!$ individual representatives, i.e. all possible non-equivalent, distinct functions realized by the gate for its all possible input permutations. The gate representatives allow for checking all possible gate applications in all possible conditions. Each such gate representation is characterized by the corresponding variant of its input permutation, and the corresponding Boolean function it implements for this input permutation.

Due to symmetries (see Section 6.5.3), the number of representatives corresponding to a particular gate can be substantially reduced. The representatives set size reduction depends on the particular library. The more symmetric or partially symmetric gates the particular library contains, the greater its representative size reduction. Typical libraries contain a lot of completely and partially symmetric gates, and this enables a significant reduction. For example, an Actel library containing 50 gates of four inputs subjected to such characterization, produced a library of representatives of size of only circa 400 representatives. In this case, the maximum possible number of representatives would reach 1200 instances, if permutations would not exploit symmetries in the characterization process. Another part of the library, the part which contains all gates with number of inputs equal to 3, would produce for all distinctive 43 gates 168 gate instances if symmetries are taken into account, or 258 instances otherwise. The same practice of symmetries exploitation for the library size reduction applies to gates with 2 input which in case of Actel with 4 gates in the library, yields 18 instances instead of 56. The typical reduction potential can be as high as several times, and therefore, it makes much sense to adequately explore and exploit this potential. Such a pre-characterized library, containing representatives for all possible permutations of the gate inputs as separate gate instances, enables each n-input instance to be considered as a potential candidate in the decomposition process. Moreover, the library pre-characterization process, prepares all information about each available gate instance needed for the decision-making during the functional decomposition process in a suitable form. The gate instance information includes the following:

logic features:

- **realized logic function** in a number of representations:
– Boolean function stored in original formula format,
– compact minterm representation of the function’s set-system representation,
– information set representation,

• **width** – support size,
• **input and output inverters** (polarization) – potentially *collapsed*,
• **paired gate** that introduces the complementary Boolean function (output phase) – if present,
• **symmetric inputs** group(s), hierarchical and rotational symmetries.

**physical features**:

• **gate foot-print area** – die size,
• **gate delay** – separately for every arc (input pin to output pin delay), represented as:
  – intrinsic delay:
    * rise,
    * fall,
  – output driver capabilities:
    * rise (pull-up) strength,
    * fall (pull-down) strength,
• **power dissipation** or power density (if provided).

**Logic features** contain the specification (see Section 6.2) of the logic function realized, denoted in the min-term representation, a corresponding output set-system, and as an information set that particular function realizes.

In our information-driven decomposition approach, the required set of elementary information items of a given sub-function has to be covered by the information set computed by a particular gate or several gates (e.g. in *n-tuple*). The comparison process of the required and delivered information sets is performed multiple times during the decomposition process. Optimization of the elementary information representation has a significant impact on the performance of the entire synthesis process.

The foot-print area of a given gate determines the area cost of usage of the gate in the resulting network. During the synthesis process, the area of a particular gate is used to determine its suitability of application in a given decomposition set in comparison to other candidate gates and to compute the area of the resulting network. The computation of the total area occupied during the decomposition process does not account for the area of interconnections. However, in our decomposition process the number and lengths of interconnections are both minimized. Moreover, the modern multi-layer interconnection technologies allow for a flexible compact placement and routing, and consequently, the impact of routing is small in the case of short and medium length interconnections. In the circuits from our synthesis method, that minimizes the number and length of interconnections, this result in a very good correlation between pre-P&R and post-P&R area results.
cell(nanf201) {
  area : 24;
  cell_footprint : "nan2";
  pin(A) {
    direction : input;
    capacitance : 1;
  }
  pin(B) {
    direction : input;
    capacitance : 1;
  }
  pin(Z) {
    direction : output;
    function : "(A B)’";
    timing() {
      intrinsic_rise : 1;
      intrinsic_fall : 1;
      rise_resistance : 0.2;
      fall_resistance : 0.2;
      slope_rise : 0.0;
      slope_fall : 0.0;
      related_pin : "A B";
    }
  }
}

Figure 6.33: Excerpt from a text file description of technology gate library.

Usually, technology libraries provide quite accurate values of physical parameters, either obtained from modeling (such as timing parameters, such as in Equation 6.3.8) or extracted from the resulting development masks (such as in Figure 6.2) and net-lists (as in Equation 6.3.9 after extraction from ASCII input file 6.33).

Our parser analyzes the input ASCII file description of a given technology library and fills-in the appropriate library model data structures for further processing. The example NAND gate, (Figure 6.33) labeled as nanf201, occupies the foot-print area of 24 area units\(^5\). The cell footprint label is ignored (as of now 20 Feb 2007) in the current version of the library parser. Further, each pin, either as input or output is described with appropriate physical, and logical features. For the input pins, the presented example provides the value of the input capacitance per input pin. The capacitive load of the input pins in units is defined in technology library preamble, for example

$$\text{capacitive_load_unit} (0.1, \text{ff})$$

The input load is used to predict the delays resulting from (dis-) charging input capacitance through the gate’s output drivers using equation 6.3.8. The presented

\(^5\)The area units defined in technology library preamble, if any.
example defines the output pin with its physical parameters:

- function, presented as formula
- detailed timing parameters:
  - fixed delay from input to output pins
    * intrinsic\_rise
    * intrinsic\_fall
  - output driver current source/sink capability
    * rise\_resistance
    * fall\_resistance
  - slope sensitivity factor
    * slope\_rise
    * slope\_fall

Our approach uses simple current source/sink model of the CMOS driver structure and RC load model [42]. It gives sufficient quality approximations and can also be used for the long interconnection delay prediction [42, 46, 105, 132], together with the interconnection length prediction [91].

Summing up, the physical features involve the physical parameters relevant for the logic level synthesis. All this information is taken into account during the general functional decomposition process to make adequate decisions regarding the gate application in decomposition and to compute the overall physical parameters of the entire network under construction. For instance, to update the critical path delay when considering the gate’s internal delay, the fanout and interconnection delay. In the case of power dissipation, only a quite coarse approximation can be achieved if the input activities and other power-related information are not known. The ability to drive a certain number of inputs can be considered, while fanout upper-bound is known in some cases.

In parallel to the individual gate parameters, during the execution of the main general functional decomposition algorithm, we are also interested in the information on the size of particular sub-libraries containing the gates of a certain input size. For the input support construction algorithm, it is important to have information which support sizes are the most popular among the entire library population, of what support sizes there are no gates instances available, etc. In the decomposition stage, the support feasible size (analogously to the feasibility in LUT meaning) is the size equal to the width of gates that have the most gate instances in the technology library, as seen in Table 6.14.

### 6.5.3 Symmetry detection

Among the total number of $2^n$ Boolean functions there is a significant number of symmetric Boolean functions. Many symmetric functions can be compactly represented using different data structures, as for instance, different functional expressions, cubes, decision diagrams, etc. This feature often substantially reduces the
memory required to store a symmetric function. In hardware realizations, the symmetric functions often require fewer gates than the other types of functions.

On the other hand, the majority of gates in the typical technology libraries, are fully or partially symmetric. Finding the symmetries for every gate helps to exploit specific features of symmetric Boolean functions.

There exist a number of methods for the symmetry detection [16, 77, 78, 97, 103, 138], usually targeted to a specific symmetry types. In the process of the pre-characterization, a generalized symmetry detection algorithm must be used, that is able to detect all the existing symmetries. The detection of all symmetries allows us to minimize the number of input permutations and corresponding gate instances.

Each gate in the library has to be examined for existing symmetries among inputs. Knowledge of symmetry features allows to group gates into classes of symmetries, namely group symmetries. Exploiting those symmetries for each of those classes we can minimize number of needed permutation to compute. E.g. instead of $1200$ comparison (50 gates of 4-input gates yields $4!$ permutation for each gate) we would have $400$ instances only (8 gates of total symmetry, 12 of $3^{rd}$ order symmetry, 12 of $2^{nd}$ order, 13 with two group of $2^{nd}$ order and 5 of none-symmetry inputs) for Actel Library (1996).

The level of symmetry exploitation depends heavily on used library. For example because 3 among all 12 of gates only with double pair group symmetry (aabb) in Actel Library has a feature of $2^{nd}$ level hierarchy symmetry ($H$), as presented in [95], we could leave out this feature and still obtain quite good speed up caused by $1^{st}$ level symmetry exploitation during the library pre-characterization process. As we group gates into symmetry classes, calculate abstraction and information transfer function, we can search for gates which comply to the constraints by given function under decomposition. The pre-characterization process gives us the information of possible elementary information items carried out by particular gate but all those items are calculated for one fixed permutation of input bits. Each other possible permutation could be calculated "on the fly" to limit resource usage, but since memory occupied by elementary information items of four-input functions is considerably small (all possible elementary information items of four input gates yields $16^2/2$ items between $2^4$ symbols) we could keep them all in memory for further computations. Instead of permuting the symbols for each gate in library we could permute symbols of given constraint elementary information items.

Among the following symmetry categories, only the group symmetry is directly detected using a specialized algorithm:

- groups symmetry (see Definition 2.13),
- hierarchical symmetry (see Definition 2.14),
• rotational symmetry (see Definition 2.15).

For typical complex gates, the library intrinsic (and detected) symmetries allow to substantially decrease the number of the input permutations needed to distinctly represent all functions realized by possible variations of gates. Such a representation allows to keep the gates with all possible nonequivalent permutation of inputs, as a homogeneous library, with no need to compute permutations every time a search of a particular Boolean function is performed. Since a large majority of symmetries relates to the first kind of symmetry presented (maximum $n$-th order group symmetry) it is possible to treat all gates with less than the maximum number of inputs as if they were at most $n$-th order symmetric gates. For example, for each $k$-1-input gate in $k$-input representation in quantity of $k$ times more representatives are needed. An extra $k$-th input is represented by an extra input asymmetric to all the other inputs.

There exist several ways to describe a function realized by a gate. Therefore, there are several approaches to detect different kinds of symmetries.

Knowledge that some variables are symmetric in a given function, can be used in many ways. In particular, it helps to produce better variable orders for the Binary Decision Diagrams (BDDs) and related data structures (e.g., Algebraic Decision Diagrams). It has been observed that there often exists an optimum order for a BDD wherein symmetric variables are contiguous [35].

The simplest way to detect the group symmetries is through detection of all pairwise symmetries, and having those pairwise symmetries, to create their graph, representing mutual symmetry relation, to be later searched for the maximum cliques. These cliques represent the group symmetries of the first level. To obtain information about the possible second level group symmetries (hierarchical symmetries - $h$-symmetries), a certain condition must be satisfied. Namely, there must exist at least two groups of equal sizes of the totally symmetric inputs. That implies at least four input variables in a gate. For instance, a four input totally asymmetric gate will be denoted as $(abcd)$ while a four input gate with two groups of two totally symmetric inputs will be denoted as $(aabb)$, wherein $(aa)$ and $(bb)$ denote both pairs of symmetric variables. This notation does not give any information about the $h$-symmetry between those two pairs. To find the $h$-symmetry we have to replace each pair of variables with one single variable, namely $(A)$ and $(B)$, representing replaced pair $(aa)$ and $(bb)$ respectively. Since variables in the symmetric groups on the first level are totally symmetric we can discard all of them, but one. The function obtained this way, with inputs representing separate groups of the original symmetric inputs, is then checked recursively for any existing second level hierarchical symmetry.

The symmetry detection for a Boolean function represented as a formula requires analysis of a string containing the entire function with all its literals denoted commonly as letters, with unary operators standing before or after the literals, and binary operators standing as prefix, infix or suffix. It is quite a simple algorithm to find symmetries, but on the condition that we know all the relations between the operator notations used and ordering of the library notation.

During the instance creation two distinct, at first glance, input permutations yield identical output set-systems. Rotational symmetries that were not detected in the basic expression tree analysis still yield Boolean function invariance for particular permutation sets (Fig. 6.15). Such symmetries, and all others should be detected during the instances creation. Simple check of the function equivalence with gate
representatives already processed, is sufficient to prove existence of any symmetry that creates opportunity to decrease the number of the instances created to a functionally complete list of all possible unique instances.

Having a function represented as an expression tree (see Section 6.2.2), it is straightforward to find the group symmetries. All groups of operands connected to a common operator node are intersected pairwise to create unique groups. Such unique groups represent symmetry groups we are looking for.

In the example presented as gate example in Figure 6.4 (further analyzed in Section 6.2.2) such groups would include the following:

- A
- BC
- D

Since all the three groups are disjunctive there is only one symmetry group of size larger than one \((B,C)\). Remaining two trivial groups, each containing a single element, are not taken into account.

Other types of symmetries, such as hierarchical \((h)\) symmetry and rotational \((r)\) symmetry, due to their rather rare presence in the library gates, may be, with an insignificant computation cost, detected exhaustively. For detailed description of methods and algorithms to detect symmetries please refer to Section 6.5.3.

In this way, we can detect any yet undetected symmetries, such as hierarchical symmetries, and because of a small size of the problem instances, it does not take much computation resources to detect such symmetries.

The physical parameters (e.g. delay) are not always equal for the input pins considered as symmetric in the logical sense. Although, Nöth et.al. have noticed [102] that: *The delay difference between the slowest and the fastest input of any gate in the library does not exceed the delay of the fastest input of all gates.* Further, they claim that: *There always is an optimal mapping, in which for every gate the fastest signal of a signal list is connected to the slowest input of the gate.* In the real-life technology libraries the highest delay differences occur between the inputs that are mutually asymmetric. The delay differences between the symmetric inputs are usually insignificant, and therefore, can be neglected.
6.5. LIBRARY PARSER

Table 6.15: Three input function with rotation symmetry property.

<table>
<thead>
<tr>
<th>input symbol</th>
<th>inputs</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The parser function fills the physical gate library and pre-processes the physical and logical parameters. The next step of the library modeling involves creation of a series of homogeneous sub-libraries containing representatives of all physical gates with a fixed number of inputs. The sub-libraries are created, depending on determined size of the widest gate necessary for the process of decomposition, starting from 2 inputs up to the number denoting the feasible size. Each homogeneous sub-library of size \( i \), created in this step, contains all representatives of the physical gates that have equal or less inputs to/than \( i \). In the latter case, the unused inputs of the representatives are referred to, in this thesis, as don’t care - inputs (DC-inputs). All DC-inputs are mutually symmetric and increase the number of possible input permutations. For example, the 2 input NAND gate with 2 symmetric inputs, yields 3 representatives in the 3 input sub-library, with DC-input on the first, second and third input accordingly.

The Figure 6.36 presents the input permutation vector with the corresponding fixed-order inputs vector.

Additionally to the corresponding input vector permutation and reference to the technology gate data structure, the following data is computed for every representative to speed up the search for the appropriate gates during the matching phase:

**logic features :**

- a Boolean function signature (see Section 6.2.4),
- a set of elementary information items realized by the corresponding Boolean function (see Equation 2.6.1),
6. TECHNOLOGY LIBRARY MODELING FOR THE PURPOSE OF
THE INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

Figure 6.35: Transition delay time with 0.015pF load and input slope rise time 0.1ns, for NOR gates with two upto eight inputs.

Figure 6.36: Representatives of technology gate NAND in 3-input sub-library.
- a bit-mask representing DC-inputs (see Section 6.2.2),
- a bit-mask representing the inverted inputs (see Section 6.4.2),
- a permutation vector (see Section 6.2.3);

**cross references to**: 
- the corresponding gate implementing complementary Boolean function,
- the corresponding gate implementing Boolean function without inverted inputs.\(^6\)

<table>
<thead>
<tr>
<th>(\text{in} )</th>
<th>(\text{inputs} )</th>
<th>(\text{out} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>(B)</td>
<td>(0)</td>
</tr>
<tr>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
<tr>
<td>(1)</td>
<td>(0)</td>
<td>(1)</td>
</tr>
<tr>
<td>(2)</td>
<td>(1)</td>
<td>(0)</td>
</tr>
<tr>
<td>(3)</td>
<td>(1)</td>
<td>(1)</td>
</tr>
</tbody>
</table>

Table 6.16: Function signature for the only representative of 2-NAND gate of 2 input sub-library.

The example of the 2-input physical NAND gate yields just one single representative in 2-input sub-library. For sub-libraries with larger number of inputs, the number of representatives increases.\(^6\)

\(^6\) In case of multi-level implementation with inverters gates on inputs.
6. TECHNOLOGY LIBRARY MODELING FOR THE PURPOSE OF
THE INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

Table 6.17: Function signatures for representatives of 2-NAND gate in 3 permutations of 3 input sub-library.

<table>
<thead>
<tr>
<th>in</th>
<th>inputs</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) 01-

<table>
<thead>
<tr>
<th>in</th>
<th>inputs</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) 0-1

<table>
<thead>
<tr>
<th>in</th>
<th>inputs</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(c) -01

of representatives, associated with a given physical gate, is equal to the number of all possible distinct input permutations. For the sub-library containing the 2-input representatives, the physical NAND gate yields a single representative, for the sub-library with the 3-input representatives it yields 3 (see Table 6.17) and for the sub-library with the 4-input representatives it yields 5 representatives (see Table 6.18);

As the Tables 6.17 and 6.18 show, a single physical NAND gate can be used in several different input configurations. Counting only the input permutations, it can be used as 2 input gate, 3 input gate with 3 distinct positions of DC-input, and in 5 different applications with 5 distinct permutations of 2 input physical gate with 2 additional DC-inputs, creating the 4 input representative input connections. As a result, the sub-library of 2 input gate representative is extended with one extra item, and the sub-libraries of 3 and 4 inputs with 3 and 5 extra items, respectively.

The representatives created in the sub-libraries for 2-input physical NAND gate have the following signatures:

- for 2-input sub-library:
  - 01:1110 (0xe)

- for 3-input sub-library:
  - 01:11111000 (0xfc)
  - 001:11111010 (0xfa)
  - 01:11101110 (0xee)

- for 4-input sub-library:
  - 0001:111111111111110000 (0xffff0)
  - 0001:111111111111011000 (0xffcc)
  - 0001:111111111110101010 (0xffaa)
  - 0001:111111101011110101 (0xffa)
  - 0001:111101110111110111 (0xeee)
Table 6.18: Function signatures for representatives of 2-NAND gate in 5 permutations of 4 input sub-library.
For the simplicity sake, when the input permutation is irrelevant, we omit the permutation information when we refer to a gate instance. Instead, a unique suffix identification will be used throughout this thesis to distinguish a different gate instance. For example a gate instances $oaif2201_1$ and $oaif2201_2$ are implemented with the very same physical gate from the technology library. The differ in the inputs permutation only.

From the above, it is clearly visible that a simple comparison of the function signatures is sufficient to recognize the Boolean function equivalence. The function signature in the form of a bit vector can be very effectively manipulated, stored in memory, and processed by the CPU arithmetic-logic unit using bit-wise operations.

Additional gate instances not having physical realizations in a given technology library are buffer gates, whose sole purpose is to mock wires in implicit non-disjoint decomposition construction. Such a special gate is added to the list of available gates and posses exceptional physical features:

- it occupies no footprint area,
- its input load is equal to the input load of gate(s) it drives,
- it does not influence the critical path.

Such a buffer, even though does not process information in any way, still is considered when computing logic and physical features of a $g$ sub-function physical realization, when the convergence measure takes the repeated variables into account for the number of outputs.

6.6 Summary

Technology library modeling was a crucial step in the road to achieve goals described in Section 1.4. It was one of the two main goals required to implement a direct single-step synthesis process. Itself, it consists of several sub-goals that were described in this chapter. The library modelling required the following problems to be solved:

- pre-processing - appropriate parser of input file describing technology library, that prepares internal, very efficient information structure and provides description of technology gates available for synthesis process,
- analysis of logic features exploited during creation of uniform representation of logic gates,
- homogeneous representation for efficient application in the decomposition process:
  - compact minterm representation makes the matching phase efficient,
  - permutation representation allows,
  - input inversion(s),
  - symmetries detection,
  - virtual gates generation.
• estimation of the final network quality, based on physical features helping estimate (predict) costs of the design components, to facilitate speed/area balanced optimization.

In this chapter the reader was familiarized with the creation of the building blocks of the decomposition. The link between the physical implementation with their features, and the logical description of Boolean function implemented in them is created in the model of technology library. In the next chapter the sub-functions construction algorithms are presented to actually make the creation of gate network possible.
6. TECHNOLOGY LIBRARY MODELING FOR THE PURPOSE OF
THE INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION
Chapter 7

Sub-function realization in the Library-based Information-driven General Functional Decomposition

A Boolean function with at most $k$ inputs is called $k$-feasible. In case of LUT FPGAs, each $k$-feasible binary function can be directly mapped into a $k$-input LUT. This is somewhat different for gate libraries. In a typical modern gate library, all 2-input binary functions have their direct gate representation, and similarly as in the case of LUT FPGAs can be directly mapped onto a corresponding gate. Also, most 3-input functions have their direct gate representations in the library, but only several percent of all possible 4-input functions, and even smaller fraction of 5 and 6-input functions can be directly mapped onto the corresponding library gates. For more inputs (e.g. 8 or 16), when at all present in modern technology libraries, only some specific Boolean functions (e.g. AND, OR) have their direct gate representations. This means that for $k>2$ not every $k$-feasible binary function can be directly mapped onto a corresponding library gate and for $k>4$ only some specific functions can be directly mapped. Consequently, a much more sophisticated multi-valued sub-function construction procedure is required for the synthesis of the gate-based circuits than for the LUT-based circuits.

7.1 Multi-valued sub-function realization

In this section we focus on the realization of multi-valued sub-functions using a number of methods supporting decomposition of Boolean functions to gate libraries. First, we explain how the binary encoding realization relates to direct construction process. Further, direct construction method is described with particular focus on convergent and non-convergent realizations. Next, three methods of binary encoding are presented: Maximal Adjacencies, support minimizing and encoding based on sum of
product and product of sum. Finally, the selection criteria are presented, that are used to select realization of particular mult-valued sub-function as a part of network under construction.

### 7.1.1 “Mechanics” of the sub-function realization

The multi-valued function \( G : \pi_U \rightarrow \pi_g \) is defined by the product set system \( \pi_U \) of its bound-set variables from \( U \) and its output set system \( \pi_g \). The number of the multi-valued function values is equal to the number of blocks of \( \pi_g \). To implement the multiple-valued function in hardware, it has to be translated into a set of binary Boolean functions, each of which can be realized using one or more of the available technology gates from a target technology library. For the binary realization of the \( n \)-valued function (the encoding of \( n \) blocks of its corresponding \( \pi_g \)), minimum \( l = \lceil \log_2 n \rceil \) binary functions and their corresponding output variables are required, which determines the minimal code length. The code length \( l \), together with the number of binary input variables in the free set \( |V| \), determine the total number of inputs to the block \( h \) in decomposition of \( f \) into \( g \) and \( h \).

The multi-valued sub-function construction procedure discussed in this section uses the minimum code length as the starting point of its search for the best gate realization. It also increases the code length when no realization was found for the current code length. See Figure 7.34 for the sub-function construction algorithm. This approach finds the local minimum of the number of physical gates to be found first and produces the minimum number of inputs to the un-feasible block \( h \). Many research results (o.e. [20, 21, 57, 60, 62, 109]) indicate that the signal convergence maximization at the successive synthesis levels results in (near) optimal networks in most cases, both in respect to the network area and number of logic levels. Thus, the explicit minimization of the number of outputs on the successive levels of the network under construction produces in result (near) optimal complete networks. The accumulated local optimization of the number of outputs of logic block \( g \) minimizes the total number of signals in constructed network. Therefore, in the work reported in this thesis, we use the minimal length codes, when possible. There is, however, a substantial difference between the LUT targeted decomposition, where each multi-valued Boolean function is always mappable on a set of binary sub-functions directly implementable with LUTs corresponding to the minimum-length code, and the gate targeted decomposition, where some multi-valued functions are not directly mappable using a single-level of gates available in the target technology library. Therefore, in the gate targeted decomposition method presented in this thesis, in parallel to the convergent single-level gate realizations of the multi-valued function \( g \), some special non-convergent realizations and multi-level gate realizations are used.

The sub-function construction procedure uses information relationships and measures to decide what information should be transferred from a given support \( U \) to the \( g \)'s output(s) and how this information should be distributed among the different binary outputs of \( g \). The selected support \( U \) and its corresponding set system \( \pi_g \), \( \pi_g = \bullet \pi_g \), together define the binary-realized multi-valued function of \( g \), \( G : \pi_u \rightarrow \pi_g \), where each particular value \( B_g \) of this function corresponds to a block of the set system \( \pi_g \) (Figure 7.1).

The binary functions \( \{g_i\} \), that compute values for the particular binary outputs of
7.1. MULTI-VALUED SUB-FUNCTION REALIZATION

$$\pi_g = \{ B_0^0 \cap B_0^2 ; B_0^0 \cap B_1^1 ; B_1^1 \cap B_1^0 ; B_1^1 \cap B_1^1 ; B_1^1 \cap B_1^1 \}$$

$$\pi_1^1 = \{ B_0^1 ; B_1^1 \}$$  
$$\pi_2^1 = \{ B_0^0 ; B_1^1 \}$$

Figure 7.1: Binary realization of multi-valued function $g$.

$g$, are defined by their input supports $U_i \ (U_i \subset U \text{ and } \cup U_i = U)$ and a set of two-block set systems $\pi_g^i \ (i = 1..l)$ – one two-block set system $\pi_g^i$ for each binary output of $g$. The number of values of the function is equal to the number of blocks of $\pi_g$.

7.1.2 Construction and selection of the most promising realizations

In this chapter we are focusing on two alternative methods of finding a feasible realization of multi-valued sub-function $g$. These two ways can, theoretically, find two identical realizations of sub-function $g$. The difference lays in the method of reaching the result. In some cases it is quicker and easier to find a promising solution using the direct construction approach, in another a direct encoding is reaching the goal quicker.

In order to implement a multi-valued function $g$ in binary hardware, it has to be transformed into a set of binary functions by assigning a binary code to each block of $\pi_g$. The binary code assignment implicitly defines a set of two-block set systems $\pi_g^i \ (i = 1..l)$ – one two-block set system $\pi_g^i$ for each binary output variable of $g$ (see Figure 7.2). For a minimum-length encoding, this set involves $l = \lceil \log_2 n \rceil$ two-block set systems. Block $B_0^i$ of a particular $\pi_g^i$ is the union of the $\pi_g^i$’s blocks that have value 0 at the $i$-th position of the assigned code. Block $B_1^i$ is the union of the $\pi_g^i$’s blocks that have value 1 at the $i$-th code position. Conversely, the code assignment of multi-valued function $g$ is defined by a set of binary functions $\{g_i\}$ which implement $g$. A particular block of $\pi_g^i$ corresponds to the intersection of blocks $B_0$ or $B_1$ of particular $\pi_g^i$, depending on the value at the output of $i$-th binary function $g_i$ corresponding to $i$-th position of the assigned code. The selection of set of binary functions $g_i$ realized by gates from the technology library explicitly constructs feasible realization of multi-valued function $g$.

Usually the codes with minimum length are used, because they maximally reduce the number of binary functions that implement $g$ and the $h$’s input support. The resulting network is usually more compact and easier to further decompose, when the number of the $g$’s outputs is smaller. It is possible to build a set of $l$ two-block set systems $\{\pi_g^i\}$ (a set of $l$ binary functions) with less items of unique or almost unique information than in the original $\pi_g$ (multi-valued function). This is achieved by repeating the unique or almost unique information items in many different set systems $\pi_g^i$ and results in a higher occurrence multiplicity $m$ of the repeated items. The originally unique information items become non-unique. With growing repetition of
the originally unique or almost unique information items at different binary outputs of $g$, function $h$ tends to be easier to decompose. The information originally most difficult to transfer - the unique or almost unique information – is made easier to transfer, because it is present at more outputs of $g$ being inputs to $h$. Moreover, information repetition causes growth of common information computed by different binary functions of $g$, and thus increases the chance for good common sub-functions for various binary functions of $g$. Therefore, our sub-function realization procedure solves the problem corresponding to the following encoding problem: Find such minimum length assignment of binary codes to blocks of $\pi_g$ that the number of unique or almost unique elementary information items in $\{\pi_g\}$ is minimal.

The $g$’s output set system $\pi_g$ is created by merging some blocks of the $g$’s input set system $\pi_U$ that is induced by the selected support $U$. Even if particular information is originally not unique in $f$, i.e. it is provided by several input variables from $Z = \{x_i | x_i \in X \setminus C\}$, it may become unique as a result of construction of $\pi_g$, if it is delivered in $f$ only by the variables from $g$’s support $U$, and in consequence, only by the multi-valued variable corresponding to $\pi_g$.

Due to the fact that a very limited fraction of Boolean functions can be directly implemented using gates, a more sophisticated algorithm must be introduced for the gate-based circuits that constructs several sub-functions, for a given set of the most promising supports, and then selects the best of them for the actual implementation of network under construction. The realization quality and cost estimation is described in further section of this chapter (See Section 7.5.1 and Section 7.5.2).

In the general case, the construction procedure takes the most promising quartet $(U, R, p, \pi_g)$ (Section 5) subdivided into the layers of equal convergence and subsequently, ordered within the layers taking into account all the remaining support quality factors. It considers the quartets $(U, R, p, \pi_g)$ in the layer order, starting from the highest-convergence layer, and within each layer, in the order of their quality.

The list of promising supports is therefore sorted with the descending potential convergence. In the list supports are placed in “layers” of equal potential convergence, and treated as equally good within the convergence layers. To limit the computation effort the list is also limited to the convergence for which the number of support is larger or equal than the search beam defined by user. If in the scope of the currently considered convergence layer the number of supports having the convergent single-level gate implementations reaches the upper limit of $ns$, the search finishes. Otherwise, the convergence target is lowered and supports from the next lower-convergence layer are examined, until the upper limit of $ns$ is reached or all
supports from all layers are examined. It may happen that a certain input support $U$, even being potentially well convergent, does not have any convergent single-level direct gate implementation in a given technology library. In this case, the next supports are examined, as long as there are non-processed promising supports left. Otherwise, the unfeasible supports need to be taken into account. For such, wider but promising, in terms of potential convergence, unfeasible supports are added to the pool of supports. Further, other algorithms for unfeasible sub-functions are employed for the construction of sub-network propositions. For details on encoding methods please see Section 7.4.1 - Maximal-Adjacencies and 7.4.2 dedicated to SOP/POS encoding. In most cases the first, Maximal-Adjacencies encoding is used, unless the heuristics based on information measures decides to use SumOfProduct/ProductOfSum style of encoding or support minimizing encoding. Once finished constructing a feasible sub-network prototypes of sub-functions for all input supports, the comparison algorithm prepares the quality assessments for each proposition, to find the most optimal realization with respect to the optimization target. Next, the selected realization is inserted into network under construction, and decomposition process continues.

7.2 Multi-level sub-function decomposition

When the sub-function construction algorithm (see Section 7.3) fails to find a single-level realization for a given Boolean function, it is still possible to construct its multi-level decomposition using gates from a given technology library. If the circuit speed is the main objective, during this process it is usually preferable to construct a sub-function realization with more outputs than originally intended. The function's convergence is preserved and one or more extra gate levels are added to the network. This way, slightly increase the delay on the path(s) under consideration. If the circuit speed is not the main objective or the sub-function is clearly not on any (sub-) critical path the preference may be opposite. In the example (see Figure 7.30), both solutions presented occupy (almost) the same area, comply to given logic objectives and transfer the same required information from their inputs to their outputs. They differ in the number of outputs, output set-systems and information distribution. The realization, with the superior convergence, is much worse than the other realization regarding the delay. Instead of one level of two single-output gates, there are two single-output gates stacked one on top of the other.

The realization of the multi-level decomposition of a multi-valued sub-function $g$ may be achieved using several competitive approaches:

- Construction of a convergent sub-function through a direct construction of the corresponding multi-level single-output sub-functions, created as optimal realizations of all possible Boolean functions. Presented in Section 6.4.3 virtual gates prepared during the library characterization implement single-output Boolean functions using available physical gates. However, the area optimization performed on all virtual gates individually does not take into account reusage of common gates when virtual gates are combined together in the decomposed sub-function. This is the main drawback of this method. The main
advantage is, on the other hand, that the existing search methods for convergent decompositions can be applied, where the virtual gates can be treated the same as the other gates in the technology library. In the post-processing of the network, such virtual gates are being resolved into sub-networks of physical gates available in target technology library.

• Decomposition of the function with a custom multilevel algorithm. Such an algorithm constructs the next level of the function using, in the support selection of the next levels of a function, also the outputs of the gates already accepted for the network under construction. These gates have on their outputs already processed information and the next gates must process only the lacking rest of the information. Ultimately, the implementation obtained this way consists of two gates, connected together, increasing in the worst possible way the path delay.

• Encoding of the sub-function according to a selected algorithm. The final synthesis is performed recursively on created this way binary sub-function. Due to a fact, that no methods were published for the multi-valued sub-function encoding, they were closely researched in functional decomposition for the gate-based realization. This thesis focuses on development of such methods and comparison with several known encoding methodologies.

The following two sections describe in detail two main approaches of mapping: direct realization in Section 7.3 and gate-targeted multiple-valued sub-function encoding in Section 7.4. The goal of the direct realization method is to construct several physical realizations by directly applying gates from a given technology library and according to the quality of solutions select the best one. On the other hand, the encoding method aims at finding an encoding that simplifies the so constructed sub-functions and results in optimal physical realizations of the sub-functions decomposed recursively.

7.3 Direct realization of a multiple-valued sub-function

This section focuses on the method of finding the realization of multiple-valued sub-function through construction of multi-level and multi-output binary sub-function directly. First, the description of generic procedures of construction and selection will be presented. Further, we will focus on both, the convergent and non-convergent cases of multi-output realizations, as they have a different application in the construction of final network.

7.3.1 Method

Since each \( k \)-feasible binary function has a direct implementation with \( k \)-input LUT (CLB), finding appropriate binary logic implementations of the multi-valued sub-functions \( g \) corresponding to the most promising bound-sets \( U \) is for LUT FPGAs reasonably simple. It consists of an adequate binary encoding of \( g \) [22, 59, 61]. The procedure of finding the most promising gate implementations for \( g \) is much more
complicated. A direct single-level gate implementation of \( g \) may even not exist because the gates needed may not be present in the library. One of possible resolutions of the problem is realization of a sub-function similar to \( g \) and of similar quality. This is equivalent of delayed selection of \( g \) for a given \( U \) after checking the existence of a single-level gate realization for \( g \). Therefore, in the first place our procedure tries to construct for each support \( U \) a direct single-level gate implementation of \( g \) or of a similar function \( g^* \) that has the same support as \( g \) and the same or not much worse convergence. If for all constructed supports \( U \) no convergent single-level gate implementation can be found, it constructs some convergent two-level or non-convergent implementations. To find the direct single-level gate implementations, it considers the case \( k \leq 4 \) as the general case and all input supports (and corresponding gates) with more than 4 inputs as some special cases. In the special cases it just directly checks if some of a few gates corresponding to a certain special case can be used in the decomposition effectively. In the general case, it takes the most promising triples \( (U, R, \pi_g) \) subdivided into the layers of equal convergence, and subsequently, ordered within the layers taking into account all the remaining support quality factors. It considers the triples \( (U, R, \pi_g) \) in the layer order, starting from the highest-convergence layer, and within each layer, in the order of their quality. It tries to construct a set of the corresponding gate implementations of sub-functions \( g \) for a number of the most promising supports. For each particular support \( U \), the procedure aims to find a set of the corresponding gate implementations of \( g \), by trying to construct \( n \)-tuples of such binary functions that the functions satisfy the following conditions:

- transfer all unique information from \( U \) to their outputs,
- can be directly mapped onto corresponding library gates,
- maximize convergence of the sub-function \( g \),
- optimize a given area/delay trade-off,
- optimize transfer of the remaining information from \( U \).

Direct realization of binary sub-functions, depending on sub-functions’ number of outputs, is presented in three following Sections:

- the direct mapping (see Section 7.3.2),
- the convergent realization (see Section 7.3.3),
- the special case of direct realizations for non-convergent sub-functions, transcoders (see Section 7.3.4).

The direct mapping is performed when sub-function \( g \) can be mapped using a single single-output gate from technology library, with help of either input or output inverters. The convergent realization is performed for multiple output \( g \) sub-functions, balancing the complexity between sub-function \( g \) and remaining image function \( h \). The main purpose of the special case of direct non-signal-convergent realizations (transcoders) is to improve decomposability of the resulting function \( h \) through reorganization of transferred information.

The direct realization search is quick and requires much less resources for computation, than the two remaining methods. If some, well convergent input support \( U \)
exists and the construction algorithm finds a (good quality) direct implementation of sub-function $g$, a multiple-output binary realization of the sub-function $g$ corresponding to a given support $U$ is found by constructing an adequate binary encoding of $g$. Subsequently, the best possible two-level gate implementation of $g_i$ is found using either the general decomposition or various sorts of special two-level logic synthesis.

If there are no well convergent supports or the amount of redundant information in the support variables is high, information trans-coders are constructed. From the set of implementations for sub-functions $g$ constructed as described above, either the implementation that maximizes convergence of the sub-function $g$ and optimizes a given area/delay trade-off is selected or the best trans-coder is selected and used as a new part of a circuit under construction. Finally, a new function $h$ is computed by expressing $f$ in new variables. This sub-function $h$ is either directly implementable or it is further decomposed using the above described process.

### 7.3.2 Direct mapping

In the decomposition targeted to FPGA technology, feasibility criterion of building blocks is based on the maximum number of binary inputs of sub-functions. The signal phases are disregarded, since the phase transformation for LUT's can be achieved with neither additional cost of area, nor additional delay. For technology mapping targeted to a closed set of gates implementing a limited number of Boolean functions, feasibility criterion must be revisited. For the purpose of direct mapping, the feasibility criterion is modified to account for several special cases. To map a Boolean function, differing on only the inputs phase(s) from a function implemented in a gate library, a special phase-independent mapping feasibility criterion should be introduced. The corresponding feasibility check examines if through a simple signal negation, the sub-function can be directly mapped using one of the library gates (see Figure 7.3), i.e.:

- the single output sub-function can be directly mapped onto a gate from the technology library,
- or
- the single output sub-function can be directly mapped onto a gate from the technology library after a minor modification (negation) of its inputs or output phase (note the similarity to the NPN equivalence 2.16).

The procedure employed to check if physical gate is implementing a given Boolean function boils down to searching through library gates for gates that implement all required elementary information items.

**Ex. 7.3.1.** *Direct mapping algorithm comprises of a number of steps. See the diagram in Figure 7.4 for details. The algorithm starts with analysis of information that is available on sub-support variables. Information is categorized with respect to uniqueness. A set of elementary information items that are necessary to compute output function is created and used to sift through gate library. The first step of direct construction are necessary to establish the necessary and required conditions that*
7.3. DIRECT REALIZATION OF A MULTIPLE-VALUED SUB-FUNCTION

constructed sub-function must comply to. The analysis of information present in sub-support gives a number of elementary information items sets. The most important one is unique information set. If a graph where edges are placed between nodes set of unique elementary information items can be colored using $c$ colors, the number $c$ denotes the graph’s chromatic number. The chromatic number corresponds to the number of binary outputs $i$ of multiple-output block $g$, that the unique information can be encoded to. The following formula defines the relationship:

$$i = \lceil \log_2(n) \rceil \quad (7.3.1)$$

Let us consider a small and simple single-output Boolean function that can be easily mapped into physical gate(s) from technology library. The example presented in Figure 7.3 shows the physical realization of a Boolean function, for which compact notation denotes $0x fd$. To implement function one needs a single 3 input NAND gate and an inverter. The NPN operators (see Definition 2.16) for this example consist of:

- input permutation [102],
- input inverter vector [100].
1) Support analysis
   a) Polarization mask computation
   b) Expressing information in minterms
   c) Elementary information item computation

2) Library sifting

3) Post-production $n$-tuple improvement (CMOS gates selection)

Figure 7.4: Construction step in direct mapping of single-output sub-function algorithm.
7.3. DIRECT REALIZATION OF A MULTIPLE-VALUED SUB-FUNCTION

The input permutation shows how to connect external inputs to a fixed-ordered input vector of physical gate instance (see Section 6.2.3). The input inverter vector specifies the phases of input variables (see Section 6.4.2), either positive, or negative, where an additional input inverter is needed to be inserted, or a driving gate would need to be replaced with a complementary one.

Negation of a signal can be achieved through the introduction of inverter, which is available in every target technology library\(^1\), or by negation of the function which provides the input signal\(^2\). The first case can be identified via search for a gate that implements the required Boolean function, that is represented in the common minterm representation (see Section 6.2.4). Translation from the term representation into the ordered compact minterm representation gives the complete information about requirements to which the constructed sub-function must comply. Since the information is phase invariant, but physical gates are not, additional data must be gathered to explore the possibility of phase inversion of the input signals. Such information is provided by the inputs and includes:

- the network level on which the input signal is available - relative to the sub-function under construction,
- the input inversion cost - the area increase and additional network level(s),
- whether the additional network level(s) increase the critical path.

Such information is especially beneficial when the constructed sub-function is able to well exploit the signal negation, either through the driver replacement, or extra inverter inclusion, i.e. when the only cost is a small additional foot-print area without increase of the critical path delay. In case of the (relatively) rich libraries, where a quite large subset of all functions is available in both polarizations in direct representation, the variable inversion realized as a complementary gate substitution is/can be even cheaper when the foot-print area is concerned.

The direct mapping is performed when the incompatibility graph of information that must be preserved (i.e. of unique information) can be colored with 2 colors. In such a case, the incompatibility graph has the bi-partitioning property (i.e. it's chromatic number is equal to 2). A graph with bi-partitioning property can be represented as a two block setsystem, with eventual don't care minterm symbols placed in both blocks, or the actual bi-partition, if there are no don't care symbols. Such symbols carry irrelevant information in respect to the target function. The set-system prototype determines the absolute minimum of information items, that the sub-function \( g \) under construction must implement. Depending on the number of the aforementioned irrelevant symbols, the prototype can be fully specified in a number of ways (in \( 2^h \) where \( h \) is the number of irrelevant symbols, to be precise) using (bi-) partitions. The following example shows the steps taken by direct construction procedure in case of mapping an incompletely specified sub-function into physical gates from technology library.

---

1. This is true for all practical technology libraries.
2. Output inversion or replacement with one that implements complementary Boolean function.
Ex. 7.3.2 (Setsystem prototyping). Let us consider the setsystem prototype of given 3 input Boolean function defined as follows:

\[ \pi_{U_2} = \{ 1, 2, 4, 7, 0, 2, 3, 4, 5, 6, 7 \} \]

It can be implemented with gates corresponding to the following output (bi-) partitions or their complementary equivalences:

- \( \Pi_{U_1} = \{ 1, 2, 4, 7 ; 0, 3, 5, 6 \} \)
- \( \Pi_{U_2} = \{ 1, 2, 4, 0, 3, 5, 6, 7 \} \)
- \( \Pi_{U_3} = \{ 1, 2, 7, 0, 3, 4, 5, 6 \} \)
- \( \Pi_{U_4} = \{ 1, 2, 0, 3, 4, 5, 6, 7 \} \)
- \( \Pi_{U_5} = \{ 1, 4, 7, 0, 2, 3, 5, 6 \} \)
- \( \Pi_{U_6} = \{ 1, 4, 0, 2, 3, 5, 6, 7 \} \)
- \( \Pi_{U_7} = \{ 1, 7, 0, 2, 3, 4, 5, 6 \} \)
- \( \Pi_{U_8} = \{ 1, 0, 2, 3, 4, 5, 6, 7 \} \)

The bi-partitions determine the search space for the direct implementation and the best direct implementation. Moreover, in the case of virtual gate library (see Section 6.4.3) they give description of the desired functions as described in section 6.4.2. But instead of searching for matching bi-partitions, it is more efficient to look for a physical gate among technology library gates that implements a limited number of required elementary information items. In presented example, due to presence of don’t cares, the set of information items consists of only three items: \( 1|6, 1|5, 1|3, 0|1 \). The search cost is linear with the size of elementary information items. Sifting procedure returns here with five gate candidates. The list becomes the list of potential physical implementation of sub-function under consideration.

Due to the fact that the inputs can be inverted through the driver gate replacement, the presented example covers a sub-set of possible solutions. Out of all possible, there could be solutions that require polarization change of particular input variables. Polarization change can be achieved either by insertion of an additional inverter gate, or by replacement of a gate that drives that input. The first option is always possible, while the latter one can only be implemented when the driver gate has its own anti-equivalent gate available in the library or among the virtual gates. Depending on the cost of the additional area and/or the extra level(s) of the required polarization, the actual cost of the solutions constructed must consider the cost of the additional input polarizations changes through the cost of the potential input and
output inverters. When the sub-function $g$ implements the primary output, its phase is critical and must be realized. In this cases the eventual output inverter always increases the critical path. In any other case, sub-function $g$ implements one of nodes within the network under construction and its polarization is irrelevant. Eventual output inverter can be omitted due to phase invariance. Same rule applies to the case of De Morgan’s laws, when alternative solutions differ on a number of input inverters, but actual inverters are not placed in the network, because the driver gates are substituted with the complementary ones.

### 7.3.3 Convergent realizations

In the method presented in this thesis, the construction of the most promising input supports is very similar to the support construction in the circuit synthesis method targeting LUT FPGAs developed earlier [59, 60]. Therefore, below we provide only some most general information on the process of sub-support construction that is necessary for a proper understanding of this thesis and we focus on the sub-function direct construction. More detailed discussion of the input support construction, illustrated with examples, can be found in [60] and its part also in Section 4.2.2. Direct construction of a convergent decomposition results in encoding of Boolean function using the minimal or close to minimal length encoding, and search for the best physical realization of such constructed multi-output sub-function. Effective convergence of complete realizations must also consider the number of logic levels of the circuit constructed for the sub-function. For example, in case of a sub-function with 4 inputs and 2 outputs convergence is equal to 2, but on the span of 2 levels, it is equally good as convergence equal to 1 in the case of a sub-function of 3 inputs and 2 outputs on the span of a single level. Such a convergence normalization in relation to the number of logic levels is required when several various realizations must be compared and some of them selected for optimal results. The way to compare multi-output, and multi-level sub-function is presented in Section 7.5.1.

It is searching for the set of single-output Boolean function implemented by particular physical gates from the given technology library, referred to as $N$-tuples throughout this thesis (see Definition 4.1 in Section 4.2.4), or a set of pre-constructed virtual gates during the technology pre-characterization (see Section 6.4). The size of search space of all the possible realizations can be expressed as the following Formula 7.3.2:

$$C^m_k = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

(7.3.2)

where:
- $n$ - size of a given technology library,
- $k$ - the number of gates comprising a $N$-tuple.

The vast size of search space creates computation complexity problem which is one of the main challenges of the research presented in this thesis.

The following sections are describing the method of finding the (close to) optimal solution of multi-output sub-functions, through a number of steps similar to the direct mapping procedure. The direct mapping algorithm steps were presented in flowdiagram in Figure 7.4 and can be compared with the algorithm outline presented in
7. SUB-FUNCTION REALIZATION IN THE LIBRARY-BASED INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

Start

max_levels = 1

Create sub-library limited by max_levels

Yes

Construct

Success

Not

Fail

Yes

max_levels += 1

Figure 7.5: Construction algorithm for convergent sub-function.
7.3. DIRECT REALIZATION OF A MULTIPLE-VALUED SUB-FUNCTION

Section 7.3.3. In further sections, we focus on the major differences of convergent construction with respect to direct mapping, namely: solution search procedure, that creates initial seed partial solution (see Section 7.3.3) and implicitly creates multi-valued Boolean sub-function by mutating the partial solution until it becomes final. Selection of starting point of search process is important as determines the size of potential search space. Once we establish the critical points of information graph that has to be split into a number of bicolored partitions, we can efficiently limit search space. It is done by selecting the most cumbersome elementary information item, that can be found when the odd-sized cycles are computed (see Definition 7.1). The process of mutation will be described in Section 7.3.3, where we focus on the methods used to minimize search space (estimated by Equation 7.3.2) through a proper ordering of elementary information items during mutation process. It can be achieved using the same criteria of estimating the potential implications on search space size, as when the initial starting point was selected: the odd-size cycle criterion. Every mutation step results in a number of possible partial solution. Therefore, criteria of quality and implementation cost estimation are needed to efficiently compare partial solution during the process. The double beam algorithm is employed to ease searching process via elimination of less promising partial solutions. The quality estimation is described in Section 7.3.3. It is used for during and when mutation process is finished. The most suitable final solution is then post-processed and used as sub-circuit in final network under construction.

Algorithm outline

The purpose of convergent sub-function $g$ construction algorithm is to find a (close to) optimal physical realization of a part of the network under construction. All possible k-feasible, multi-level solutions greatly outnumber the solutions that are (close to) optimal. The sub-function construction algorithm presented in this thesis employs heuristics to limit the search space while preserving high quality solutions in the reduced space. The heuristics are based on the following assumptions:

- the component single-output circuits of the (close to) optimal solution are constructed as (close to) optimal realizations of single-output Boolean functions,
- the search space is limited by means of ordering the construction steps, as well as, analysis, evaluation and selection heuristics based on information relationships measures.

The support construction algorithm, presented in Section 4.2, provides a set of the most promising input sets for the sub-function $g$ construction algorithm. For every support proposed and selected by the support construction algorithm, steps presented in flowdiagram in Figure 7.6 are performed.

The initial steps are identical as the analysis of elementary information items in support input set is equally important. The successive steps differ as the library sifting needs to be replaced with a procedure that searches for an $n$-tuple. Post-processing step is also similar: the constructed alternative solutions are compared together, with respect to their cost and quality.

Due to the fact that polarization of signals can be only manipulated using an additional inverter gate, its cost with respect to the additional area and potential
Figure 7.6: Single step in sub-function construction algorithm.
increase of critical path length needs to be included in the cost analysis of particular physical realization that requires polarization change. For every input which cannot produce both polarizations at zero or low cost we prefer to build sub-function which would use the existing polarization. The Boolean values at particular position in the mask expresses the fact that the inverters at the corresponding inputs should be avoided.

Information relationships and measures influence with what priorities redundant and unique elementary information are considered when the output setsystem of sub-function is created. When quantity of redundant information is much greater than unique (non-redundant) information, the sub-function build algorithm must take special care for the elimination of redundant information and avoiding of potential redundant information multiplication on sub-functions’ outputs.

To efficiently proceed with elementary information item assignment (as described in details in Section 7.3.3), the elementary information items are being sorted according to the following rules:

1. First elementary information items which are involved in larger number of odd-sized cycles (see Definition 7.1).
2. Elementary information items which are covered by a smaller number of candidate gates get priority.
3. Elementary information items least available at a given support get precedence (see Table 7.1).

The analysis of elementary information items present on the inputs of selected boundset gives an insight into the input support, on which the construction is currently being performed. The analysis is used to select the starting point for construction algorithm, which throughout this dissertation is referred to as seed. The proposed construction algorithm distributes the information items among \( n \) single-output sub-functions creating, this way, the binary realization of the multi-valued sub-function \( g \). The process of distribution is performed “one-by-one” and starts from the seed. Each elementary information item is being assigned onto a particular single-output function, keeping the logical constraints explained below. Boolean function is specified in the form of the information graph composed of the information present on function’s support inputs. To construct multiple output sub-function \( g \), information must be distributed among binary outputs of sub-function \( g \). If the information required to compute given multi-valued Boolean function is represented as (master) graph, then information item distribution among single-output binary functions defines the division of (master) graph into mono-chromatic information graphs of particular binary component functions. Logical constraint that every information sub-graph must fulfill at this step is the constraint of bi-colorability, hence the information represented in the form of a setsystem is in fact a two block set system or partition.

To limit the assignment search space it is important to sort the information items. The ordering should account for the following in factors:

- the number of odd-length cycles a particular information item is involved in,
- the saturation color of both symbols involved in a given information item,
- the availability of a particular information item at the input variables (uniqueness),
7. SUB-FUNCTION REALIZATION IN THE LIBRARY-BASED
INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

• the number of binary outputs at which a particular information item is already blocked, due to the bi-colorability criterion.

The saturation color of a particular node in information graph is the number of adjacent nodes with which a given node is incompatible. The maximal saturation color of a given information graph gives (indirect) indication of the chromatic index.

The binary realization construction algorithm considers the elementary information items one-by-one until all the information required to be processed in a sub-function under construction is assigned. Each intermediate solution constructed this way consists of a set of (not) completely specified single-output binary Boolean functions together with their possible physical realizations. With the progress of the elementary information item assignment, the Boolean function specification becomes more and more complete. Each assigned elementary information item adds a particular (or both) symbol (minterm) either to block zero or one and at the same time induces other elementary information items, for symbol(s) (minterm(s)) already assigned to complementary block. The induced information item(s) needs to be taken into account and clarified either as the relevant information, used to minimize the unique information (information item multiplication), or as redundant information. Together with the increasing completeness of the function specification, less and less technology gates implement a particular binary Boolean function. Thus, the corresponding list of technology gates implementing a particular single output of the sub-function \( g \) under construction is shrinking with the progress of the construction algorithm. Each time a new elementary information item is assigned, a feasibility check is performed to screen the list for the gates that do not fulfill the mapping constraint anymore. The algorithm must be very fast, because it is used extensively throughout the entire sub-function construction procedure.

The elementary information assignment and processing of the related changes in the corresponding single-output Boolean function will be referred to as the mutation.

The mutation process involves the following main steps:

• selection of the outputs which are able to accommodate a successive elementary information item,
• assignment of the information item to the output(s) selected in previous step, in both polarizations, if possible,
• quality evaluation of the realizations constructed this way,

Seed creation

The main goal of a seed creation process is the selection of a good starting point for further elementary information items assignment that limits the search space for successive assignment(s) and results in high-quality solutions. This boils to the pre-assignment of certain unique information items to particular output bits (which later must not be re-assigned). This pre-assignment is based on the multi-valued sub-function input support analysis from the information viewpoint. The input support analysis gives precise information about the elementary information items carried by particular input variables. Among other things, it determines:
7.3. DIRECT REALIZATION OF A MULTIPLE-VALUED SUB-FUNCTION

<table>
<thead>
<tr>
<th>in</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7.1: Simple 3-input Boolean function decomposed in Example 7.3.3.

- the unique information set of information relevant to compute the output function,
- the double information set of the relevant information,
- the redundant information set of information not needed to compute the output function,
- potential odd-sized cycles in the information graph of unique information.

To perform the elementary information item distribution among the single outputs, one must find the most problematic information items to start with. The one which hinder the assignment process in to the highest degree of all information items. This allows to significantly limit the search-space for the successive assignment steps while preserving the high-quality of the constructed solution.

The graph representing the elementary information items can have sub-graphs of specific properties. For the direct construction the most crucial feature is the length of cycles found in the information graph.

**Definition 7.1 (Odd- and even-size cycle).** A cycle or circuit of length $n$ is a closed path without self-intersections; equivalently, it is a connected graph with degree 2 at every vertex. Its vertices can be labeled $v_1$, ..., $v_n$, so that the edges are $v_i v_{i-1}$ for each $i = 2, ..., n$ and $v_n v_1$. A cycle with an even number of vertices is called an even cycle; a cycle with an odd number of vertices is called an odd cycle.

Apart from a lack of any physical realization of a certain Boolean function on a particular number of outputs, the existence of an odd-size cycle in the information graph disallows construction using a small number of outputs (single output gates). To cover such information an alternative way of expression of information set that contains an odd-size cycle needs to be used. One of such method is to extend the number of outputs with an extra output (gate) to distribute elementary information item(s) that together composes the odd-size cycle.

**Ex. 7.3.3 (Concurrent information distribution).** Let us take into consideration a simple 3-input Boolean function to decompose into a technology library presented in Figure 3.6 on the page 43.
Figure 7.7: Examples of odd-size cycle with 3, 5 and 7 nodes.
7.3. DIRECT REALIZATION OF A MULTIPLE-VALUED SUB-FUNCTION

Setsystem representation of target Boolean function:

\[ \pi_{U_f} = \{ 0 | 1, 2, 3, 5 ; 1 | 0, 4, 6, 7 \} \]

The elementary information set required to implement a given Boolean function consists of items:

\[ IS(U_f) = \{ 1 | 7, 2 | 7, 3 | 7, 5 | 7, 1 | 6, 2 | 6, 3 | 6, 5 | 6, 0 | 5, 4 | 0, 5, 1 | 4, 2 | 4, 3 | 4, 0 | 3, 0 | 2, 0 | 1 \} \]

Even tough the graph is a bipartition, due to the lack of gates implementing single output Boolean function, the algorithm constructs two-output sub-function \( g \), to distribute elementary information items between.

\[ IS(U_{g_0}) = \{ 1 | 7, 2 | 7, 3 | 7, 5 | 7, 1 | 6, 2 | 6, 3 | 6, 5 | 6 \} \]
\[ IS(U_{g_1}) = \{ 0 | 5, 4 | 5, 1 | 4, 2 | 4, 3 | 4, 0 | 3, 0 | 2, 0 | 1 \} \]

Where output 0 was implemented in single output function \( g_0 \) using NAND gate and
output 1, in \textit{g}_1 using \textit{NOR} gate, corresponding setsystems are as follows:

\[
\pi_{U_{g_0}} = \{ 0, 7 ; 1, 2, 3, 5 \} \\
\pi_{U_{g_1}} = \{ 0, 1, 2, 3, 5 ; 4 \}
\]

Information distribution allows implementation of the first level of a resulting network with only two two-input physical gates. Output setsystem computed as a product of setsystems of two binary signals is given as:

\[ IS(U_f) = \{ 1 | 7, 2 | 7, 3 | 7, 5 | 7, 1 | 6, 2 | 6, 3 | 6, 5 | 6, 0 | 5, 4 | 5, 1 | 4, 2 | 4, 3 | 4, 0 | 3, 0 | 2, 0 | 1 \} \]

Corresponding graphs are presented in Figures 7.8a and 7.8b, respectively.

The bi-colorability condition is fulfilled when neither of the information items closes an odd-length cycle in the information graph. Therefore, such information items, that could cause odd length-cycle in a particular output, must be implemented on one of the other outputs of the sub-function under construction. There are several different cases of (see Figure 7.9) an elementary information item assignment. An elementary information item can be either:

\begin{itemize}
  \item \textbf{a)} assigned as the first information item,
  \item \textbf{b)} bounded by the already assigned information items so that the existing partial solution constructed determines the only option of assignment,
  \item \textbf{c)} assigned together with the earlier assigned information item(s), but due to the polarization freedom, it can be assigned in two distinct polarizations,
  \item \textbf{d)} blocked by the earlier assigned information items.
\end{itemize}

The information items analysis gives also information about all objectives presented in the previous sub-chapter. Every elementary information item is not allowed to be assigned together with particular even number of other elementary information on the same output bit, because altogether they would create an odd-length cycle. Such complementary (\textit{concurrent}) information items to the information item being assigned at a given step create an odd-length cycle, and therefore, render information set impossible to be encoded on a single output bit. The concurrent information item completes the odd-size cycle, and therefore, makes it impossible to realize the resulting function as a multi-output function using a certain number of single output gates.

\textbf{Ex. 7.3.4 (Concurrent information item).} Presence of information items forming odd-size cycles in information graph, renders them restricted on certain outputs. Detection of elementary information items that can potentially “close” the odd-size cycle, is helpful to determine the set of problematic information items and find the set of solutions sooner, through the elimination of the not-promising partial solutions at the early stage of direct construction algorithm. In Figure 7.10 the concurrent information items for the incomplete information graphs consisting of elementary information items:
Figure 7.9: Elementary information item \( n_3 | n_4 \) assignment options.
7. SUB-FUNCTION REALIZATION IN THE LIBRARY-BASED
INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

Figure 7.10: Concurrent elementary information items.

- for cycle of size 3 with two information items: $n_1|n_3$ and $n_2|n_3$ the concurrent information item is $n_1|n_2$;
- for cycle of size 5 with four information items: $n_{11}|n_{12}, n_{13}|n_{14}, n_{14}|n_{15}$ and $n_{15}|n_{11}$ the concurrent information item is $n_{12}|n_{13}$.
- for cycle of size 7 with six information items: $n_{21}|n_{22}, n_{22}|n_{23}, n_{24}|n_{25}, n_{25}|n_{26}$ and $n_{27}|n_{21}$ the concurrent information item is $n_{23}|n_{24}$.

The number of these concurrent information items creating merging restrictions determines the size of a problem. In consequence, to avoid encoding obstacles, one should start elementary information item distribution process with the most problematic one. It is the one that cannot be encoded together on a single output bit (concurrent information items) with the maximum number of other information items. To find the most problematic information items for bi-partitioning, the ones with highest saturation color sum are searched for. Saturation color is cumulated over both symbols of a particular information item. The selection of which one of the information items that together compose an odd-size cycle is not assigned on same binary output, is left to the solution quality estimation and comparison procedures. The partial solutions with different variants of information distribution are compared and based on the quality assessment. The better solution is considered for further evaluation and processing.

For the information matrix example shown in Table 7.2, the highest number of obstacles occurs for the following information items: 1|3, 1|5 and 3|5 in the form of saturation colors of symbols. Each of the three symbols: 1, 3 and 5 have: 5, 5 and 7 other incompatible symbols, respectively and are pairwise incompatible as well. Symbol 5 as the one with the highest occurrence frequency in the information items is the first candidate to create the initial seed.

In cases of a tie between the information items (here: 1|5 and 3|5) regarding the sum of the saturation colors of both symbols the information item is specified upon, we had to find a tie-break algorithm to decide which one of those information items (1|5, 3|5) to encode on a separate output bit.
For both elementary information item pairs, the saturation color sums are equal to 12. However, when we combine the sets of symbols with which both 1 and 5 is incompatible together we obtain larger common set than in case of pair 3 and 5. In case of pair 1 and 3 we combine pair of sets: (1|2, 1|3, 1|5, 1|6, 1|8) and (1|3, 3|4, 3|5, 3|6, 3|8), and in case of pair 3 and 5 we combine the same set corresponding to symbol 1 with (0|5, 1|5, 3|5, 4|5, 5|6, 5|7, 5|8) obtaining for pairs:

- 1|5 symbols: (0, 2, 3, 4, 6, 7, 8) with common set: (3, 6, 8)
- 3|5 symbols: (0, 1, 4, 6, 7, 8) with common set: (1, 4, 6, 8)

We can therefore, conclude that there is a higher probability of the search space limitation by the couple (3|5), hence it would be better to assign it separately. On the other output bit pair of (1|3) and (1|5) will be encoded then.

Another measure helping to decide which information item presents the most of restrictions during the assignment process is to check for its presence in the odd-length cycles. As an example let’s take:
symbol 1’s cycles: symbol 3’s cycles: symbol 5’s cycles:

\[(1, 3, 5)\]  \[(1, 3, 5)\]  \[(1, 3, 5)\]  
\[(1, 3, 6)\]  \[(1, 3, 6)\]  \[(1, 5, 6)\]  
\[(1, 3, 4, 5, 6)\]  \[(3, 4, 6)\]  \[(3, 5, 6)\]  
\[(1, 3, 4, 5, 8)\]  \[(3, 5, 6)\]  \[(3, 5, 8)\]  
\[(3, 5, 8)\]  \[(5, 7, 8)\]  \[(1, 3, 4, 5, 6)\]  
\[(1, 3, 4, 5, 8)\]  \[(1, 3, 4, 5, 8)\]

Combining it altogether, we obtain the complete list of the all odd-length cycles:

\[(1, 3, 5), (1, 3, 6), (1, 5, 6), (3, 4, 5), (3, 4, 6), (3, 5, 6), (1, 3, 4, 5, 6), (1, 3, 4, 5, 8)\]

According to the numbers of the odd-length cycles, elementary information item 3|5 is the most problematic elementary information item, hence it is the most suitable candidate to limit the search space to the highest possible extent. It will block most of the remaining elementary information items and limit their assignment options. This will lower the number of possible variants to explore and evaluate. It is important to keep the cycle description for the length greater than 3 not as a set of symbols, because the order of symbols in a cycle is also relevant. For example, the cycle of size 5 containing symbols 1, 2, 3, 4, 5 might be constructed with edges in a different order, e.g. as \(1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1\), but could also be ordered as \(1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 1\), etc. Keeping the cycle information in form of set of edges would improve efficiency as well, due to the check of the edge inclusion in a cycle.

When the initial selection procedure chooses a pair of the information items which both are included in an odd-length cycle to be assigned to two different outputs, it solves the problem of the odd-length cycle. The selection which section of the odd-length cycle to encode separately has to be based on the sub-function support analysis. To obtain the minimal support for every single-output sub-function in the multi-output decomposition, it is necessary to minimize the number of common inputs for different outputs.

After the assignment of pair 3|5 to the first output, what is left of all the odd-length cycles, is the following:

\[(1, 3, 6), (1, 5, 6), (3, 4, 6), (5, 7, 8)\]

Next, the pairs most populated by the odd-length cycles are left – on this stage pair 1|6 and 3|6. To break a tie we can compare the sum of saturation colors of both incompatible symbols in the pairs:

- 1|6 symbols: \(2, 3, 4, 5, 8\), with the common set: \(3, 5\)
- 3|6 symbols: \(1, 4, 5, 7, 8\), with the common set: \(1, 4, 5\)
The combination of these measures (the odd-length cycle participation and the saturation colors) helps the selection algorithm to make a decision of the initial information choice. Within all unique information, the algorithm should select for the same output those with the largest common support. Starting with the first output, as the first information item select the one which is:

- unique,
- the least available,
- (when not unique) available in only a single polarization,
- participating in the highest number of the odd-length cycles,
- denoting incompatibility between the most saturated symbols.

As a result, we have sorted elementary information items, with the most difficult items at the beginning and least problematic at the end of list.

In the example matrix shown in Table 7.2 we present a very obvious minimal common support of information item couple: 1|2 and 1|4 as well as the following group of items: 4|6, 4|6, 5|6, 5|7, 5|8, 7|8, hence such groups would most likely be processed together as the result of support sharing.

With every generation of the partial solutions created, the list of yet-to-assign elementary information items is processed, and sorted, because the factors contributing to the difficulty of assignment change with every development step.

Each single elementary information item is characterized by a number of features relevant for the information assignment process:

- hard constraints related to:
  1. **Availability** in the currently constructed circuit level – whether a particular information item is available on the boundset only (unique in the boundset of current decomposition step) or on both the boundset and the freeset. Each unique elementary information item must be implemented inside the sub-circuit that implements the sub-function under construction. Depending on which of the above two cases had taken place, the reaction should be as follows:
    
    (a) **Unique information**, i.e. not available anywhere else (e.g. within freeset), but on the selected boundset. Regardless its availability on the boundset it must be implemented in the currently constructed sub-circuit, and, preferably on more than one output, for the purpose of the unique information multiplication. Such an information item:
    - must be pre-assigned to one of the outputs,
    - can be re-assigned (moved) to any other output (or copied to multiply unique information), unless such an operation would yield the output permutation,
    - must not be un-assigned (removed).
    
    (b) **Non-unique information**, i.e. available also on the freeset. It does not have to be pre-assigned into a seed. High availability of the elementary information item does not necessarily make the gate selection easier. Since the entire gate library contains the gates with fixed
Table 7.3: Elementary information item availability and polarity on three-input support.
input polarity, with option of duality of outputs, the option to choose the polarity of a certain information item (seen as an input polarity) increases the level of freedom during the gate selection. Such an information item:

- does not have to be necessarily pre-assigned,
- can be re-assigned among the outputs, if found to improve solution, regarding quality of the gate candidates,
- can be harmlessly un-assigned, if found to became obstacle to acquire solution of a higher quality.

2. **Odd-length** cycles - both symbols of a given elementary information item are components of an odd-length cycle. In a graph representation this elementary information item is one of the edges in the cycle. Depending on the cycle size and the number of outputs there is a limited number of possible information item pre-assignment, namely the cycle partitioning. To limit the search space one can select optional pre-assignment, regarding:

- assign-ability obstacles for each single output, what actually boils to determine the possibility of multiplying certain information items, preferably those of which both symbols are involved in a odd-length-cycle and unique,
- gate candidate quality,
- minimum common support size of each single output.

- obstacles regarding:

1. **Availability**, inside the current selected boundset. The number of occurrences determines the level of freedom during the selection of the input from which a certain information is taken for further processing. A particular information item available on a particular input is bounded together with other information items available on the same input variable, unless it is solitary.

2. **Merge-ability** - a high sum of saturation colors computed as function of saturation colors of both blocks taken separately.

In consequence, every elementary information item has to be tagged with the following attributes for further processing:

- **BOOL unique** - true when unique on the selected support in the respect to the entire level,
- **int availability** - determine the number of occurrences inside the selected boundset,
- **BOOL odd_size_cycle** - true when it is a component of an odd-length cycle,
- **BOOL polarity_availability** - true in the case of a non-unique information item for which both polarities are available.

All the aforementioned elementary information item characteristics were computed when the support was analyzed in terms of the elementary information items.

To sum up, the elementary information item used to create the starting point is selected upon analysis of the following aspects:
• odd-length cycle participation,
• availability - uniqueness,
• polarity availability - input inverters,
• number of other symbols, the two symbols of a given elementary information item are incompatible with, creating other unique elementary information items,
• number of physical gates that implements it ("popularity").

The pre-selected information items for the first output are always selected from among those with the highest degree of hard constraints. The decision made for the subsequent outputs determines to a larger and larger extent the shape of the entire information distribution. The information item selected for the second output is decided to be separated from the information item pre-selected for the first output. Therefore, this decision is the most important for the further elementary information item assignment.

Realization of a single output sub-functions involves, at most, the phase selection only and since the information is phase independent, it applies only to each single output sub-function that implements a primary output.

The approach presented above must be preceded with a check of a possible decomposition with the input inverters. Such a decomposition adds an extra footprint area of the included inverters, but maintains the imposed target convergence, hence it is preferred over the non-disjoint decomposition.

Seed mutation

The actual search for the most promising solutions starts with an initial partial solutions representing seeds and is performed through the evolution of partial solutions until some final solutions are found. The solution construction algorithm considers many possible cases, performs a sophisticated analysis and uses the results of the analysis in a sophisticated construction performed in the framework of an effective and efficient double-beam search algorithm. In every step of the seed mutation process a new breed of partial solutions is obtained, evaluated, compared and the most promising are selected for further mutations.

The main issue in the information item selection to be followed during the seed mutation is to choose such an element that limits the search space of potential solutions to a largest extend. Since order of the assignment, in respect to joining the information items to be encoded on the common or separate outputs, is irrelevant, such an approach sooner restricts the search space resulting in a more efficient search.

N-tuple representation of (partial) solutions contains (among other) information about which physical gates from technology library and in what particular variant (input permutation and input inverters) can be used to implement particular binary output. Such list of possible physical gates is referred to, in this thesis, as a list of physical candidates. The definition of n-tuples refers to a single candidate per binary output, but for the sake of simplicity and efficiency, each binary output has assigned its own list of candidates gates. The actual candidate that is considered as eventual physical implementation is the one that is selected from the list of candidates.
as the most suitable. At the beginning of the \( n \)-tuples creation, when the candidates lists lengths change drastically, very rough measure equal to the length of the shortest candidates lists helps to predict which evaluated \( n \)-tuples have the highest probability to be finished. Later during \( n \)-tuples creation (while still significant part of necessary information is still left to be assigned), the \( n \)-tuples are sorted according to the measure which assesses predicted \( n \)-tuple quality when the \( n \)-tuple creation will be finished. Because the major measure of the \( n \)-tuple quality is the redundant information presence, the quality measure assesses upper-bound of redundant information measure at the time it is still not yet known for sure which technology gates particular \( n \)-tuple would consists of. While the implementation progresses, it becomes clearer which cmos-gates will be used in every evaluated \( n \)-tuple and the redundant information measure becomes closer to the actual final figures.

When the elementary information assignment process is finished, all partial solutions are finished and the most promising one becomes the final solution. It could be further improved in terms of processed information with a little or no cost of area increment. Every \( n \)-tuple has a lists of possible gates that can be used for each binary output, one candidate list per binary output. Each of them satisfy the primary constraint of processing the unique information, and the choice of which one to select is left for the final improvement algorithm. Initially the lists of candidate lists are sorted with a static sorter which puts the smallest (in footprint area) gates first.

During the seed growth, the new elementary information items must be inserted into a number of outputs. To explore all possible partial solutions, a given information item is assigned to every possible output(s). The search space is limited by the feasibility check of partial solutions, only. Therefore, to reassure (close to) minimal search space, appropriate measures are applied to order elementary information items to achieve minimal search space. All those endeavors yield in exploitation of all possible (partial) solutions.

**Ex. 7.3.5 (elementary information items assignment).** For example, consider two mutually independent elementary information items: \((a|b)\) and \((c|d)\). Such information items called also elementary (or atomic) dichotomies merged together give two distinct non-equivalent setsystems, denoted as:

- symbol \( a \) together with \( c \), and \( b \) with \( d \), which yields setsystem of \((ac, bd)\)
- symbol \( a \) together with \( d \), and \( b \) with \( c \), which yields setsystem of \((ad, cb)\)

The sub-function construction algorithm will produce both possible configurations. According to the algorithm presented on Figure 7.11, both partial solutions (assignments) are produced and screened for possible physical realizations.

A complementary output gives the possibility to encode the upper level of the network being constructed with a larger freedom of selection of the inputs phases. In such a case, keeping two (almost) equivalent gate candidates until the decomposition of the next level is completed gives an opportunity to choose the phase of the intermediate variables to decrease the fanout and number of potential inverters/buffers. Such an approach involves a special treatment of the inputs during the information items computation, because the symbols are input-phase dependent.

The algorithm which is responsible for the phase manipulation, can be used during the network construction. In the bottom-up decomposition, when a part of the
Start

- remove item from pool
  - is there an EII in the pool?
    - yes
      - remove EII from pool
        - physical realization found?
          - yes
            - take the next EII
          - no
            - case of assigned option
              - block
              - complement
              - first
              - free
            - choose polarization
              - assign both polarizations
              - polarization irrelevant
              - polarization relevant
        - no
          - take the next output
          - success
    - no
      - take the next EII
      - assign
      - polarization irrelevant
      - physical realization found?
        - yes
          - remove EII from pool
        - no
          - case of assigned option
            - block
            - complement
            - first
            - free

Figure 7.11: Single step of mutation (assignment) algorithm.
network is mapped, the algorithm can search for the potential collapse candidates among the part of network below the currently built level. The collapse candidates, are searched among the potential predecessor sub-functions, which can be replaced with a complementary equivalent, to provide a phase change at the small cost of small area difference. Each network path is examined whether it is possible and profitable to collapse the output inverter on the path into the driving gate. Such collapsing involves insertion of negators at the inputs of other gates that use the complementary signal (reference to “forks”). The gate replacement would change the fanout of the replaced gate depending on the fork configuration which drives the given gate. The speed versus area trade-off optimization gives an opportunity to choose a smaller area solution with higher fanout, or vice-versa.

Figure 7.12: Multioutput sub-function support minimization.

The subsupport size has to be extended to accommodate a new information item in cases of lack of:

- common sub-support part between already assigned,
- gate candidate which covers information available on the current subsupport.

Therefore, it is important to follow certain rules while selecting the outputs for assigning the currently considered information item. The preferred output(s) to assign has:

- the largest common support part among support of the information set already assigned to the output and support of the information output under consideration,
- the least quality deterioration of the gate candidate list, calculated as the quality deterioration caused by this assignment. The deterioration occurs due to a swift of gate candidates, and removal of gates that do not cover the new elementary information item.

To estimate the quality of a partial solution, the \( n \)-tuple contains, for each output, the list of single output physical gates (candidates), that could be used to implement a given (partial) solution. The lists are created as a result of sifting through either an entire technology library, in case of assignment of elementary information to an empty output, or a result of sifting through a candidate list that was created in previous steps of \( n \)-tuple mutation. The sifting procedure used to create candidate lists is
the very same procedure as used to find single-output realizations in direct method described in Section 7.3.2. Each insertion of an information item effects in narrowing of the pool of the gate candidates. Every such pool is kept sorted, so at any given moment of partial solution construction, the most promising physical gates out of the entire list is located in the front of candidate list. The criteria for the sorting of gate candidates were discussed in Section 7.3.3.

The process of sifting through the list of gates, either the entire technology library or through the candidate gates for specific single output of \(n\)-tuple, boils down to a check whether a specific elementary information item is present or not among information items of sifted gate. Sifting is performed with elementary information item that is being assigned and every physical candidate gate that does not distinguish said information item, is removed. Similar rule applies to the gate candidates with less inputs than the support width. The gates in the technology library having less inputs than the input support of the sub-function under construction, can be used in a form of the input-extended representatives. Additionally to having all the possible distinct representatives having the inputs permuted, all gates are extended up-to the size of the maximum size considered as feasible (see Section 6.4 for details of homogeneous technology library). For example, two input NAND gate is represented as one representative of two inputs, but also as three representatives having three inputs each, four representants having four inputs each, and so on. Every extended gate has been extended with don’t care input(s) (further referred to as DC-input(s). Such DC-inputs are asymmetric to all of the non-DC-inputs and symmetric to each other. There have to be \(n\) times more gate instances in the pre-characterized library just due to exploitation of all the possible input assignments in case of the \(n\)-input support in consideration of narrower gates (size of \(n-1\)). A special approach is necessary here to make use of pre-characterized library. After the information item selection, we can realize that the minimal support is smaller than entire support of constructed sub-function. It can be the case when the physical gate, that realizes a given single output of constructed sub-function, is a physical gate with less inputs than complete support of constructed sub-function. If then there exist a solution of physical realization using a combination of single output gates with less inputs than a complete support of constructed multiple output sub-function, such solution achieves support minimization goal.

The main goal of sub-function construction in the functional decomposition targeted to gate libraries is to construct (encode) the binary outputs of sub-function \(g\) in such a way that both feasibility and quality objectives are met. To achieve that aim, each step of the elementary information assignment is followed by a feasibility check (see Section 7.3.2). Such a check swifts all gate candidates which were already checked if they comply to the previous, less strict, objectives of all necessary information items, when previous elementary information item was assigned. This time candidate lists are swift to check with stronger objectives of larger (expanded with an extra one EII) set of the necessary information items.

To compare such information items represented in the computer memory as bitvector, only the bit by bit comparison is needed to be performed. To compare the bit vectors shorter or equal than the machine data path word length, one machine cycle is sufficient to complete. Therefore, it is a very efficient procedure.
7.3. DIRECT REALIZATION OF A MULTIPLE-VALUED SUB-FUNCTION

Table 7.4: Karnaugh table of Boolean function sym1of4 decomposed in Example 7.3.6.

<table>
<thead>
<tr>
<th>in</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Gate candidates sorting criteria

The data structure holding partial solutions (\(n\)-tuples), also holds the references to the actual physical gates, that can be later used to implement the final sub-function solution in the network under construction.

The sorting criteria are based on the analysis of physical and logical features and fitness of the gate candidates to the actual conditions, in which the gate would be used in the final network.

The most important logical feature criterion is the number of the elementary information items carried by an individual gate. This way one can analyze the information distribution on the sub-function outputs, as for instance, how many times a particular unique information was multiplied, etc. A less important criterion is the number of the information items that are un-important for the realization of function under decomposition, but anyway realized by the candidate gate (induced).

The entire information assignment algorithm boils to the distribution of all the sub-function’s necessary information among the binary outputs in the way to transfers all unique (preferably multiply) and suppress as much as possible redundant information items. During the assignment process we can control the feasibility and quality of a solution through the library sifting process and gate candidate analysis.

Ex. 7.3.6 (Convergent sub-function construction). As an example of construction of convergent sub-function \(g\), let us examine the Boolean function of sym1of4 decomposed using the technology library stdcell, as in Appendix B. Such function asserts a logic value high for a combination in which only one of all four primary inputs is in the high logic state.

Two block setsystem representing this Boolean function is given by the following formula:

\[
\pi_{F_{term}} = \{ 4, 5, 6, 7, 8, 9, 10 ; 0, 1, 2, 3 \}
\]
Since the purpose of this example is to show the construction of a convergent sub-function, and not the support selection, the Boolean function selected for the example is fully symmetric. Any of its any primary input sub-support yields exactly the same component sub-function. Sub-support construction algorithm creates a set of most promising supports, limited by the width of search beam, namely:

- four input, including all four primary inputs, convergent to a single binary output,
- a number of three input sub-function supports, convergent to two binary outputs.

Let us first analyze the input variables and their corresponding set-systems:

- \( \pi_A = \{ 1, 2, 3, 4, 5, 6, 7 ; 0, 5, 6, 7, 8, 9, 10 \} \)
- \( \pi_B = \{ 0, 2, 3, 4, 5, 9, 10 ; 1, 5, 6, 7, 8, 9, 10 \} \)
- \( \pi_C = \{ 0, 1, 3, 4, 7, 8, 10 ; 2, 5, 6, 7, 8, 9, 10 \} \)
- \( \pi_D = \{ 0, 1, 2, 4, 6, 8, 9 ; 3, 5, 6, 7, 8, 9, 10 \} \)

Each input variable is driven here by the primary input. Therefore, it is not possible to invert any variable through the replacement of a driver gate with a complementary equivalent. The change of phase of input signal is possible by the cost of an extra inverter.

Let us consider the supports suggested by the boundset construction algorithm in the order of the maximal potential convergence. First, the four input support containing all four primary inputs is considered. Its potential convergence equal to three can potentially be achieved using a single physical gate of four inputs. Unfortunately, the target technology library does not offer a physical gate that implements the required Boolean function. After translation of the Boolean function into the minterm representation, the function is given by the following setsystem:

\[
\pi_{G_{\text{minterm}}} = \pi_{F_{\text{minterm}}} = \{ 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15 ; 0, 1, 2, 4, 8 \}
\]

The set of elementary information items induced by the required setsystem implies the unique information set and is given by the \( IS(G_{\text{term}}) \):

\[
IS(G_{\text{minterm}}) = \{ 0|10, 1|10, 2|10, 3|10, 0|9, 1|9, 2|9, 3|9, 0|8, 1|8, 2|8, 3|8, 0|7, 1|7, 2|7, 3|7, 0|6, 1|6, 2|6, 3|6, 0|5, 1|5, 2|5, 3|5, 0|4, 1|4, 2|4, 3|4 \}
\]

The term representation is then translated into corresponding minterm representation according to the term-minterm correspondence, and as such, the example elementary information item in term \( 1|10 \) requires four minterm information items to be represented in minterms, namely: \( 2|9, 2|11, 2|13 \) and \( 2|15 \).

The set of elementary information items induced by the required setsystem implies the relevant information set for the computation of Boolean function being decomposed, and is given by the \( IS(G_{\text{minterm}}) \), sorted according to the algorithm presented in Section 7.3.3:
7.3. DIRECT REALIZATION OF A MULTIPLE-VALUED SUB-FUNCTION

<table>
<thead>
<tr>
<th>term</th>
<th>minterms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>12,13,14,15</td>
</tr>
<tr>
<td>6</td>
<td>6,7,14,15</td>
</tr>
<tr>
<td>7</td>
<td>10,11,14,15</td>
</tr>
<tr>
<td>8</td>
<td>3,7,11,15</td>
</tr>
<tr>
<td>9</td>
<td>5,7,13,15</td>
</tr>
<tr>
<td>10</td>
<td>9,11,13,15</td>
</tr>
</tbody>
</table>

Table 7.5: Term minterm correspondence for Boolean function in Example 7.3.6.

\[ IS(G_{\text{minterm}}) = \{ 0|1,0|2,2|3,1|3,3|4,0|4,4|5,2|5,1|5,4|6,2|6,1|6,4|7,2|7,1|7,7|8, \\
6|8,5|8,3|8,0|8,8|9,4|9,2|9,1|9,8|10,4|10,2|10,1|10,8|11,4|11, \\
2|11,1|11,8|12,4|12,2|12,1|12,8|13,4|13,2|13,1|13,8|14,4|14,2|14, \\
1|14,8|15,4|15,2|15,1|15 \} \] (7.3.3)

The unique information set is equal to the relevant due to a lack of repeated values, therefore every relevant elementary information item for primary output is automatically unique.

The construction algorithm in the next step searches for a set of physical gates, represented by a Boolean functions that they implement, that together cover the required information set. The minterm representation of unique elementary information items implies a two block setsystem of the required Boolean function: Therefore, the sub-function construction algorithm, first of all, tries to build a single output sub-function \( g \) that implements a given Boolean function described by the following translated into minterms a bi-partition:

\[ \pi_{G_{\text{minterm}}} = 0,1,2,4,8 \] (7.3.3)

The search is performed to find two alternative physical realizations, with positive 7.3.3 and negative 7.3.4 polarization.

\[ \pi_{G_{\text{minterm}}}^{\text{neg}} = 0,1,2,4,8 ; 3,5,6,7,9,10,11,12,13,14,15 \] (7.3.4)

For decomposition targeted to the technology library presented in Appendix B, there is no such physical gate that implements either positive nor negative polarization. To continue the construction algorithm tries to construct a sub-function \( g \) with lower convergence - with more than one single output physical gates. For the given unique information set there is no such pair of gates within the target technology library that can implement a given Boolean function, construction algorithm continues to search...
for a triplet. First of all the initial n-tuple seed is created with only the first elementary information item from the pool of unique information items assigned to only one of the outputs, i.e. 0|1, to produce a partial solution with three binary outputs. This seed n-tuple will look as follows:

- $O_0$ elementary information item assigned a single elementary information item 0|1, and realized by 122 physical gates from technology library (stdcell.lib, see Appendix B);
- $O_1, O_2$ with no information assigned yet, and therefore with no physical gate that can be used to implement it;

The process of assignment continues with the consecutive unique elementary information item out of sorted list: 0|2 to produce two partial solutions, a new breed of solutions. This step results in creation of second generation of partial solutions. In this case, there are two partial solutions:

1. assigned together:

   - $O_0$ with two elementary information items 0|1 and 0|2 assigned together, to produce a bi-partition 0|1, 2;
   - $O_1, O_2$ with no information assigned yet, and therefore no physical gate can be used to implement;

2. assigned separately:

   - $O_0$ with a single elementary information item 0|1 assigned;
   - $O_1$ with a single elementary information item 0|2 assigned;
   - $O_2$ with no information assigned yet, and therefore no physical gate can be used to implement;

The list of constructed solutions is then again taken by the mutation procedure (see Section 7.3.3) to produce next generation (breed) of partial solutions with an even higher level of precision and greater number of unique elementary information items assigned, and therefore, closer to the final solution. Due to the fact that the permutation of outputs does not change the Boolean function implemented by constructed sub-function, the construction algorithm disregards alternative partial solutions that differs only with the order of outputs to the already created ones.

The selected n-tuple consists of the following physical gates:

- norf401_0: implements Boolean function $A \lor B \lor C \lor D$ with input permutation 0123
- oaf2201_2: implements Boolean function $((A \lor B) \land (C \lor D))$ with input permutation 0231
- oaf2201_1: implements Boolean function $((A \lor B) \land (C \lor D))$ with input permutation 0213

The selection criteria on which the above decision was made is the result of the comparison of the following features of the constructed solutions in the order:
### Table 7.6: A sorted list of $n$-tuple found by the construction algorithm in Example 7.3.6.

<table>
<thead>
<tr>
<th>idx</th>
<th>area</th>
<th>$n$-tuple</th>
<th>setsystem</th>
<th>information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$</td>
<td>m</td>
</tr>
<tr>
<td>0</td>
<td>120</td>
<td>norf401_0, oaf2201_2, oaf2201_1</td>
<td>3, 5</td>
<td>28, 20</td>
</tr>
<tr>
<td>1</td>
<td>120</td>
<td>norf401_0, oaf2201_2, oaf2201_0</td>
<td>3, 5</td>
<td>28, 20</td>
</tr>
<tr>
<td>2</td>
<td>120</td>
<td>norf401_0, aoif2201_0, norf401_0</td>
<td>3, 5</td>
<td>30, 28</td>
</tr>
<tr>
<td>3</td>
<td>88</td>
<td>norf201_1, norf201_4, oaf2201_2</td>
<td>4, 7</td>
<td>28, 16</td>
</tr>
<tr>
<td>4</td>
<td>104</td>
<td>xorf201_0, norf201_5, oaf2201_1</td>
<td>4, 7</td>
<td>29, 20</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
<td>xorf201_0, xorf201_5, oaf2201_1</td>
<td>4, 8</td>
<td>27, 20</td>
</tr>
<tr>
<td>6</td>
<td>88</td>
<td>norf201_3, norf201_2, oaf2201_2</td>
<td>4, 7</td>
<td>30, 24</td>
</tr>
<tr>
<td>7</td>
<td>104</td>
<td>xorf201_0, norf201_5, oaf2201_0</td>
<td>4, 7</td>
<td>21, 28</td>
</tr>
<tr>
<td>8</td>
<td>120</td>
<td>xorf201_0, xorf201_5, oaf2201_0</td>
<td>4, 7</td>
<td>15, 28</td>
</tr>
<tr>
<td>9</td>
<td>104</td>
<td>norf301_0, blf01_1, oaf2201_2</td>
<td>4, 6</td>
<td>28, 18</td>
</tr>
<tr>
<td>10</td>
<td>120</td>
<td>norf301_0, mux201_2, oaf2201_1</td>
<td>4, 6</td>
<td>30, 24</td>
</tr>
<tr>
<td>11</td>
<td>96</td>
<td>norf301_0, blf01_1, blf10_0</td>
<td>4, 6</td>
<td>28, 30</td>
</tr>
</tbody>
</table>
• signal convergence (the higher the better),
• number of blocks of output setsystem after merging (the lower the better),
• the number of levels of (virtual) sub-function, if the next level is final,
• signal convergence cost (the lower the better),
• effective convergence,
• the number of levels, after a potential inverter collapse procedure is applied, if the optimization objective is speed (the lower the better),
• the quantity of redundant elementary information items still present on sub-function outputs (the lower the better),
• the number of blocks of the product setsystem of sub-function outputs (the lower the better),
• the quantity of unique elementary information items still being unique on the sub-function outputs (the lower the better),
• the foot-print area (the lower the better).

It is worth noting, that the information quantities have higher priority than the foot-print area. It is especially visible when a special type of non-convergent sub-function (transcoders) are constructed, where more stress is put on the quality of the information restructuring than the local minimization of area. The effective encoding implemented in the synthesized this way sub-function $g$ is presented in Figure 7.7.

The encoding of the selected physical realization expressed in the same terms as the product setsystem is given below as a setsystem:

<table>
<thead>
<tr>
<th>min-term</th>
<th>inputs</th>
<th>outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D X Y Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 0 1 0 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 0 1 0 0 1 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 1 1 0 0 0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 0 0 1 0 1 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 1 0 1 0 1 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 0 1 1 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 1 1 1 0 0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 0 0 0 1 1 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 1 0 0 1 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 0 1 0 1 1 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 1 1 0 1 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 0 0 1 1 0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 1 0 1 1 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 0 1 1 1 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 1 1 1 1 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.7: Karnaugh table of the Boolean function implemented by the physical realization of the encoded $g$ sub-function from Example 7.3.6.
Another promising sub-support provided by the support construction algorithm is a three input support with potentially two binary outputs:

- ABC
- ABD
- ACD
- BCD

Due to symmetry in the function under consideration each of four supports is implementable by the same Boolean function. The only difference is the input permutation and connection to a different primary inputs. Because of that, let us consider support ABC for further analysis. The other three supports (ABD, ACD and BCD) differ on the minterm-term correspondence only. The information set relevant for the computation of primary output present on the inputs of subsupport ABC is represented below:

\[ IS(U_{ABC}) = \{0|10, 1|10, 2|10, 3|10, 1|9, 2|9, 3|9, 0|8, 1|8, 2|8, 3|8, 4|7, 1|7, 2|7, 3|7, 0|6, 2|6, 3|6, 0|5, 1|5, 2|5, 0|4, 1|4, 3|4\} \]

To compute the unique information set one must subtract the elementary information items present on the free-set \( W \) which, in case of support ABC, is a single input D

\[ IS(U_{ABC}) \cap IS(D) = \{0|10, 1|10, 2|10, 3|10, 2|9, 0|8, 1|8, 2|8, 3|8, 4|7, 1|7, 2|7, 3|7, 0|6, 2|6, 3|6, 0|5, 1|5, 2|5, 0|4, 1|4, 3|4\} \]

Table 7.8 shows the translation of term symbols is performed according to the computed term-minterm correspondence.

\[ IS(U^\text{min}_{ABC}) \cap IS(D^\text{min}) = \{0|1, 0|2, 2|3, 1|3, 0|3, 3|4, 0|4, 4|4, 5|2, 5|1, 5|0, 5|5, 4|6, 2|6, 1|6, 0|6, 4|7, 2|7, 1|7, 0|7\} \]

Redundant information set, present in the information set of a subsupport select for current construction, and not needed by the primary output:

\[ IS(U^\text{red}_{ABC}) = \{3|7, 5|7, 6|7, 3|6, 5|6, 3|5\} \]

The construction algorithm begins with the creation of seed with a single unique elementary information item 0|1 and continues assignment with all unique information items. As the result, a complete solution for a given three input sub-support a physical realization is one of \( n \)-tuples listed in 7.9.
7. SUB-FUNCTION REALIZATION IN THE LIBRARY-BASED INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

Table 7.8: Term minterm correspondence of sub-support ABC for Boolean function in Example 7.3.6.

<table>
<thead>
<tr>
<th>n-tuple</th>
<th>input</th>
<th>gate</th>
<th>permutation</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>norf201</td>
<td>0_1</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>norf201</td>
<td>_01</td>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.9: Complete solution for a given three input sub-support a physical realization.

<table>
<thead>
<tr>
<th>n-tuple</th>
<th>input</th>
<th>gate</th>
<th>permutation</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>norf201</td>
<td>0_1</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>norf201</td>
<td>_01</td>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.10: Karnaugh table of Boolean function implemented by physical realization of encoded sub-function in Example 7.3.6.

<table>
<thead>
<tr>
<th>min-term</th>
<th>inputs</th>
<th>outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  B  C</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>0  0  0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1  1  0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2  0  1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3  1  1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4  0  0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5  1  0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6  0  1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7  1  1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
7.3. DIRECT REALIZATION OF A MULTIPLE-VALUED SUB-FUNCTION

Due to a symmetry both realizations present very same utilization of two gates from the gates library. The effective encoding implemented in synthesized $g$ sub-function is presented in Figure 7.10.

The encoding of selected physical realization expressed in the same terms as product setsystem is given below as setsystem:

$$\pi_{G_{\text{term}}} = \{00, 01, 01, 01, 11; 00, 01, 01, 11, 00, 01, 01, 11\}$$

To evaluate the physical realization, the selection algorithm computes the information relationships and measures and compares solutions pairwise in a common representation. The common support of two given sub-supports consists of all inputs from both sub-supports. Comparison of, for example, sub-support $ABCD$ and $ABC$, requires computation of physical realization of sub-function with inputs $ABC$ in common symbols with physical realization of sub-function with inputs $ABCD$. That boils to the computation of a sub-function with inputs $ABC$, as it would be extended with an extra input $D$ and extra output. For details please refer to Section 7.5.1. Out of the two mutually exclusive sub-functions $g$, the selection algorithm selects the four input function with three binary gates. For comparison of sub-functions of a different input set size, a common information channel is established. To accomplish this, a smaller sub-function is expanded with an additional input, to make its input support set exactly the same as of the wider sub-function. The two considered solutions are evaluated with regard to:

- input vector size,
- level of availability of its inputs,
- (predicted) level of their output(s), i.e. network’s critical path analysis,
- number of repeated variables,
- number of blocks in output setsystem,
- convergence normalized against the number of levels of physical realization,
- foot-print area.

Due to the difference in the number of blocks in the output setsystem, the preferred solution is sub-function with four inputs and three outputs.

The next step is to include selected sub-function into a network under construction, calculate the remaining image function $h$, and continue with decomposition of the remaining infeasible image function $h$ recursively. The remaining $h$ function is presented in Table 7.11.

After translation of the remaining image function $h$, the three-input Boolean function is decomposed recursively. Again the algorithm searches for a direct single-output physical gate that implements a given Boolean function in symbol terms as in Karnaugh table from Table 7.11:

$$\pi_{h_{\text{term}}} = \{0, 0, 0; 0, 0, 0\}$$

and in minterm symbols:
Table 7.11: Karnaugh table of Boolean function remaining in function $h$ after insertion of sub-function $g$ in Example 7.3.6.

\[ \pi_{h_{\min\text{term}}} = \{ 0, 1, 2, 3, 4, 5, 6 \} \]

Analysis of gates that drive the input variables reveals the following information:

1. Variable $0$ implements a bipartition

\[ \pi_{g_0} = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 \} \]

The driver is a cmos gate orf401 with complementary equivalent (norf401 in tech library. Complementary driver gate has no input inverters. Complementary cost: gate replacement: -8.

2. Variable $1$ implements a bipartition

\[ \pi_{g_1} = \{ 0, 1, 2, 4, 5, 6, 8, 9, 10 \} \]

and is driven by cmos gate aoif2201 with complementary equivalent (with inverters on all inputs aoif2201) in technology library. Complementary driver gate has 4 input inverters that introduces one extra network level. Footprint area cost of gate replacement: extra 64 area units.

3. Variable $2$ implements a bipartition

\[ \pi_{g_2} = \{ 0, 1, 2, 3, 4, 8, 12 \} \]

and is driven by cmos gate aoif2201 with complementary equivalent (with inverters on all inputs aoif2201) in technology library. Complementary driver gate has 4 input inverters that introduces one extra network level. Footprint area cost of gate replacement: extra 64 area units.

All three variables: 0, 1 and 2 together carry information necessary and sufficient to compute primary output function.
\[ \pi_g = \pi_{g_0} \cdot \pi_{g_1} \cdot \pi_{g_2} = \begin{cases} 011 \quad 100 \quad 101 \quad 110 \quad 111 \\ 0, 7, 11, 13, 14, 15 ; 3, 12 ; 5, 6, 9, 10 ; 1, 2, 4, 8 \end{cases} \]

The phase change on any of input variable would increase footprint area, and/or add extra gates in the network, especially inputs 0 and 2. Even though the sub-function \( h \) being decomposed comprises the last level of the network, and output phase is critical, it is still considered as a viable solution.

\[ \pi_{h \text{minterm}} = \begin{cases} 0 \quad 1 \\ 2, 3, 5, 6 ; 0, 1, 2, 3, 4, 6, 7 \end{cases} \]

The positive polarization variant of a given Boolean function is implementable by two gates from the technology library:

- **andf301** with one input inverter 100
- **norf301** with two input inverters 110

On the other hand the negative variant is implementable by another two gates from the technology library:

- **nanf301** with one input inverter 100
- **orf301** with two input inverters 110

The final selection is based on the analysis of possible solutions taking into account the following aspects:

- foot-print area cost,
- number of levels of the final network, including the possible output inverter necessary in case of selection of negative variant,
- cost of input inverters, and/or replacement of driver gates by the complementary ones, that can increase the number of network levels.

The selected positive variant requires a single driver gate to be replaced with a complementary one.

### 7.3.4 Transcoders

**Introduction**

Information transcoders are constructed in the cases when no (good) convergent direct gate implementations can be found. Transcoders are created as a result of decomposition into the \( n \)-tuple with an equal number of outputs and inputs which helps to reorganize the information transferred, get rid of the redundant information and multiply the unique information. The synthesis goal is not changed drastically comparing to the basic (convergent) sub-functions, but the algorithm becomes less strict in relation to the minimization of the area and number of input inverters, which in the case of transcoders often help to create a better solution.
Ex. 7.3.7 (Application of transcoders). The Figure 7.13 shows a proper usage of transcoders that allows for achieving a high convergence of the consecutive levels. Transcoder is a specific sub-function that does not converge in the number of binary signals, but converges in information. Transcoder translates information present on its support, transfers all unique information needed to construct the required function, when at the same time removing as much (preferably all) redundant information, as possible. To obtain a high redundant information rejection ratio, the gates that are used to construct a transcoder should have preferably many inputs. The more inputs the more redundant information is covered. Moreover, there are more possibilities of information distribution (restructuring) on the larger set of the corresponding outputs. Having in mind the hard constraint of covering the entire unique information required to construct the multivalued function, every constructed gate n-tuple is compared to the previously constructed and the best of them is selected for network construction.

![Figure 7.13: Transcoders usage example 7.3.7 (sym3of8).](image)

As an example let us examine benchmark sym3of8. That Boolean 8-input function described in pla format, with on-set completely specified using 56 minterms, which implies together with the complementary 200 minterms of the off-set, 11200 information items to completely specify the Boolean function in minterms. The introduction of two 4-input transcoders, as presented in Figure 7.13, limits this quantity down to 1500 information items on the transcoder outputs (second level of circuit network). It is more than 7 fold redundant information reduction, achieved through the information restructuring only. Redundant information reduction and unique information restructuring allows for construction of convergent sub-functions on next level. In the presented example, the consecutive network level comprises of one single-output sub-function of 4 inputs and one three-output sub-function, also of 4 inputs. Effectively on the space of two circuit levels convergence achieved is 8 signals to 4 signals, which is quite difficult to obtain in average circuits. As a result of the transcoder introduction on the first network level, the resulting function is much easier to decompose, yielding a quick and compact circuit. As shown on the example for benchmark sym3of8, SIS produces a network occupying almost twice as much area, more than twice as many gates, and at the same time an almost twice as long critical path. The almost identical relation between results obtained from our tool and
Figure 7.14: Karnaugh table for input and output information of transcoders in sym3of8 benchmark.

**SIS** on other symmetric benchmarks (that includes: sym3of8, sym5of8, sym23of8, sym235of8, 9sym) demonstrates on extraordinary fitness of our method for this type of Boolean functions.

As a final result of decomposition for the standard cell library Figure 7.15 shows complete example of gate network implementing benchmark sy3of8, the single output Boolean function taken from MCNC benchmark suite.

In the Figure 7.13, due to incompatibility of network representation in the graphs, the network produced by IRMA2GATES includes the output node, which is not incorporated in the critical path. This output node is omitted in the SIS network graph, as it is not representing a logic gate. This is true for all network graphs throughout this thesis.

**Transcoders creation algorithm**

The algorithm for transcoder creation differs from the n-tuple creation algorithm only in the way the particular solution quality measures are computed and considered when partial and final solutions are being compared. The transcoder construction algorithm gets a beam-size set of the proposed supports from the main support build algorithm. For each of them it tries first to build sub-functions with convergence that is a result of greedy sub-function output product set system merging. If failed, a very similar transcoder construction algorithm builds some non-convergent sub-functions, which in turn become transcoders. For the detailed description of the transcoder construction algorithm please refer to Section 7.3.3.

**Information relationship measures**

To assess the quality of a transcoder, information relationships and information measures are computed. A high quality transcoder characterizes high redundant infor-
information rejection ratio and duplication of the unique information items. To calculate the rejection ratio we introduce the measure expressed through the following formula:

\[
R_r = \sum_{RI} (mul_{in}(inc) - mul_{out}(inc))
\]

(7.3.5)

where

- \(mul_{in}(inc)\) denotes the number of occurrences of information items on inputs of the sub-function
- \(mul_{out}(inc)\) denotes the number of occurrences of information items on outputs of the sub-function

Factor \(mul_{in}(inc)\) remains constant for every sub-function having the same input support. Factor \(mul_{out}(inc)\) is proportional to the transparency, which shows how many individual redundant information is passed through to sub-function outputs. It takes into account also possible information multiplication, meaning, on how many binary outputs of multi-output sub-function \(g\) the redundant information is available.

To characterize quantities of the covered redundant information, the histogram of quantities is computed and used for the solution comparison. An array of quantities of length equal to the number of outputs keeps the quantities of the redundant information covered by gates of \(n\)-tuple.
The important aspect is the complete number of non-desired information items still present on the information channel, which still must be suppressed on the next level of the network. First of all, the following solution are considered the most suitable and are placed in the first positions of the sorted (limited) list of partial solutions:

- cover the lowest quantity of the redundant information items;
- cover redundant information on the smallest set of binary outputs of the sub-function component gates (the histograms of redundant information presence are compared starting from the highest quantities downwards);
- reject the most redundant incompatibility pairs that are present on the multiple inputs of the sub-function support. Rejection ratio is computed as cumulated difference of the information item availability on outputs and inputs of a considered sub-function (Eq. 7.3.5);

The redundant information quantity is computed as the total number of the redundant elementary information items which are still present on the transcoder outputs, when the overall quantity of the redundant input information is less than non redundant information. Otherwise, redundant information measure computes cumulated quantity of output redundant information to assess redundant information avoidance.

Beside the elimination of redundant information, every sub-function constructed also has to fulfill the constraint specified by the required information that is needed to be covered and transferred on the sub-function outputs. Experiments performed so far show that keeping the required information duplicated which means that every elementary information item is available on two (or a number close to two), yields in the most optimal distribution of information items.

The generic algorithm for the \( n \)-tuple construction described in details in the Chapter 7.3.3, is adapted for the special case of transcoders construction to focus on different characteristics desired in the non-convergent sub-functions. During the solution construction, just as in the generic mode, prefers solutions that has a higher chance to be successfully finalized. Among the completed solutions, quality measure becomes primary decision drive. High quality transcoder is characterized by:

- significantly lower foot-print area\(^3\),
- high redundant information rejection ratio (see Section 7.3.5),
- low quantities of redundant information still covered and eventually multiplied by multioutput sub-function,
- reduction of unique information, as many of unique elementary information items covered, in more than one binary output, as possible; hence increase availability of unique information on the next level, but not (much) more than two occurrences,
- high fanin, CMOS gates, that takes part in \( n \)-tuple have many inputs to maximize information combination, there is high correlation between high redundant information dumping and gates width,

\(^3\)With the exception of area difference equal or less (comparator window size) than the foot-print area of inverter gate or any other gate available within the technology library, the smallest available gate.
7. SUB-FUNCTION REALIZATION IN THE LIBRARY-BASED INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

• lack of the input inverters, which eventually increases the critical path,
• lower foot-print area.

Switching the priority from the least occupied area to the high information re-
structuring on the first levels of network results in well convergent, small and high-
speed circuits. For the price of the extra area occupied by the wide gates on the primary inputs, that effectively translates information for subsequent levels, yields in a short critical path and small area of the remaining sub-circuit implementing \( h \). As an example let us consider four output benchmark MCNC rd84. SIS decomposed it into network with critical path on three out of four outputs, namely 1, 0 and 3, with critical path lengths 7, 7 and 9, respectively (see Figure 7.16a, 7.19a, and 7.17a). Introduction of transcoders decreases critical paths by 50% to 3, 6 and 6 (see Figure 7.16b, 7.19b, and 7.17b). Investigation of information quantities on transcoders inputs and outputs proves that the redundant information rejection is very high also in this benchmark. This dramatic difference of the critical path length in case of outputs 3 (length 7 for SIS and 6 for IRMA) and 0 (length 8 downto 6, respectively) is due to the introduction of the input transcoders. Specifically:

(0) for output 0 transcoders introduced on the first level, decreases the quantity of relevant information (from 16440 minterm elementary information items (120 minterms in onset, 137 minterms in offset) down to just 324 (18 minterms in onset, and 18 minterms in offset as well)),

(1) for output 1 generic algorithm of IRMA produced convergent network, with no need to introduce information transcoders,

(2) for output 2 transcoders are also not necessary in network construction,

(3) similarly to output 0, the transcoders insertion much influences the redundant information quantities, bringing 15228 minterms (94 minterms) elementary information items on primary inputs to 4935 (21 minterms) on the transcoders outputs, making it much easier to process information on the successive levels.

The benchmarks presented here show the influence of the redundant information reduction, unique information duplication and information restructuring on the network simplification. Benchmark 9sym is on the other hand a good example that symbol translation alone, without a major redundant information reduction, could also help to decompose subsequent levels. The input information quantities are the following: 420 minterms of onset and its complementary 92 offset minterms. The transcoder translates this information into 462 minterms in offset and 50 minterms in onset. The resulting network occupies approximately five times less area, with a twice as short critical path, comparing to the result of decomposition obtained from SIS (see Figure 7.20).

### 7.4 Gate-targeted multi-valued sub-function encoding

The following sub-sections describe methods developed for the encoding of sub-
function \( g \). Both of the methods are aiming at the simplification of the encoded
Figure 7.16: Circuit networks obtained from the decomposition of output 0 of benchmark rd84.

Figure 7.17: Circuit networks obtained from the decomposition of output 1 of benchmark rd84.

Figure 7.18: Circuit networks obtained from the decomposition of output 2 of benchmark rd84.
sub-functions and at the same time of the remaining image function \( n \). Tough contradictory, both targets can be achieved to some extent and with an efficient trade-off algorithm the overall network can be optimized. These methods use different approaches to minimize the complexity of the resulting encoded sub-function \( g \), being the same as the original \( g \), but expressed as a combination of several single-output binary functions through encoding. The single output functions can be further decomposed into small and fast sub-circuits.

### 7.4.1 Maximal Adjacencies

One of the encoding methods that we adopted to the multi-valued sub-function \( g \) encoding is the method of Maximal Adjacencies (MaxAd) originally developed by Jóźwiak for the FSM state assignment introduced in [69].

**Goal**

The reason to use the Maximal Adjacency algorithm in the multivalue sub-function \( g \) encoding, is the minimization of the number of terms in the resulting binary sub-functions and maximization of the common term sharing among the individual single-output sub-functions \( g \) by this algorithm. In result, simple compact binary two-level AND-OR sub-functions are obtained that, additionally if larger, often share some common terms or large sub-terms. Due to their simplicity some of the sub-functions may be mappable to simple library gates and the other consist of just a few gates. This yields (maximally) two-level and compact sub-functions \( g \) realizations. This also strongly correlates with the number of information items, resulting in a small number
7.4. GATE-TARGETED MULTI-VALUED SUB-FUNCTION ENCODING

Figure 7.20: Comparison of decomposed networks of 9sym benchmark.

(a) SIS1.3

(b) IRMA2GATES

Figure 7.21: Visualization of Karnaugh table for input and output information of transcoders in 9sym benchmark.
of information items and many common items, i.e. sub-functions relatively easy to decompose.

The silicon area (SA) estimation for random logic (RL) two-level implementation was introduced by Jóźwiak in [71]:

$$SA(\text{RL}) = c_1 \cdot \sum_{i=1}^{n_{PT}} n_{PT_i} + c_2 \cdot \sum_{j=1}^{N_F(k)} n_{APT_j} + c_3(k)$$  \hspace{1cm} (7.4.1)

where:
- $n_{PT}$ – total number of product terms in the Boolean functions,
- $n_{LPT_i}$ – number of literals in the product term $i$,
- $n_{APT_j}$ – number of active product terms for a function $j$ in the Boolean representation,
- $N_F(k)$ – number of functions in the Boolean representation ($N_F(k)$ grows linearly with $k$),
- $c_3(k)$ – parameters which increase with $k$ - for a given constant $k$,
- $c_1, c_2$ – some constants values.

Analysis of $SA(\text{RL})$ and the results of many experiments led Jóźwiak to the conclusion that, in most cases, the best assignments can be obtained for a minimal code length $k$. Consider the assignments starting from assignments with a minimum length $k$[71].

In order to minimize the silicon area for a given $k$, the function which grows with the total number of product terms in the Boolean function, with the number of literals in each product term $n_{PT}$ and the number of active product terms $n_{APT}$ for each Boolean function should be minimized. In order to obtain a representation of a given Boolean function $j$ with a minimal number of active product terms in the Boolean representation, a minimal set of terms has to be chosen from the set of all possible terms of a function which covers all the minterms. Each product term of the order $n$ covers $n^2$ minterms and includes $(k - n)$ literals. Fortunately, the chance of obtaining fewer terms representing a given Boolean function increases rapidly with the order of the terms used in a cover and, at the same time, the number of literals in those terms decreases. The total number of product terms, for all the Boolean function representing function with $n_{APT}$ and decreases with the number of common product terms $u_{COMPT}$ and the multiplicity $u_{COMPT_i}$ of each common term $l$[71] :

$$n_{PT} = \sum_{j}^{N_F} n_{APT_j} - \sum_{i=1}^{u_{COMPT}} u_{COMPT_i} - 1$$  \hspace{1cm} (7.4.2)

Based on these findings, Jóźwiak formulated the following heuristics[71]:

In order to find near optimal assignments of a given length $k$, look for those assignments where the resulting Boolean functions implementing the multi-valued function are represented by a small number of large product terms and many common terms that are used several times[71].

The adjacency of minterms represents a possibility of constructing product terms of higher orders. Having more adjacent minterms, there is a chance of covering all the minterms with fewer, larger product terms.

Finally, Jóźwiak formulated the following heuristic[71]: In order to find near optimal encoding of a given length $k$, look for those encodings were the resulting Boolean


7.4. GATE-TARGETED MULTI-VALUED SUB-FUNCTION ENCODING

function describes the values of the output variables with a large number of adjacent “1”s and “0”s; give a preference to common adjacencies. For a more precise explanation of the method of maximal adjacencies see [69], [71] and [70].

Algorithm outline

The algorithm of maximal adjacencies encodes the input product setsystem minimized by the main IRMA algorithm. Minimization is based on setsystem block merging based on the information measures. It was presented in details in [20]. For every block of the setsystem a code is being assigned during the encoding procedure. The encoding length is minimal. The encoding procedure involves the following steps:

1. Computation of the internal data structure used during the main encoding:
   (a) symmetrical matrix of adjacency coefficients.
2. Creation of all possible code assignments with a given code length and given number of blocks to be encoded.
3. Creation of all possible distinct permutations of previously created encodings. These codes are being checked for duplicates, with regards of output permutation and all possible polarization configuration.
4. Out of all the constructed codes a sub-set of codes is being selected with the top value of Adjacency Measure.
5. All the selected encodings are being tried to build the SOP/POS realizations of their corresponding functions $g$ composed of the resulting single-output component sub-functions $\{g_i\}$.
6. Select the best SOP/POS realizations regarding the delay, footprint area and number of complex gates.
7. Construct the best possible functional decompositions for the single output functions $\{g_i\}$ from the selected most promising encodings.
8. Select the best realization of the obtained in the two previous steps.

The resulting SOP/POS realization is placed into the network under construction without the output AND/OR gate, to allow for a (near-) optimal construction in the successive levels. Such an approach gives the opportunity to search for the most suitable realization, either with the AND/OR gate as constructed in the former step, or any other gates, depending on the outcome of the support construction in the following steps of the decomposition.

The internal data structure of MAXAd consists of a symmetric coefficient matrix. The coefficient matrix contains the Adjacency Measure coefficients for every pair of output setsystem blocks. To compute these coefficients, we must check how close are these blocks placed in the binary space. We must count the number of occurrences of the situation where the blocks of output setsystem corresponds to the adjacent blocks of input setsystem. To compute it, we first calculate the correspondence between the binary input space and the minimized output setsystem.
Table 7.12: The binary input space correspondence to the output set-system blocks indexes.

Usage of such matrix to compute the Adjacency Measure is presented in a pseudo-code below:

```plaintext
for minterm m in input_space do {
    for each b in inputs do {
        n = compute_adj_minterm(m, b)
        if (n > m) then {
            sm = output_block_for_minterm(m)
            sn = output_block_for_minterm(n)
            increment_adj_matrix_at_index(sm, sn)
        }
    }
}
```

The adjacent minterm computed in `compute_adj_minterm` differs on a bit position denoted as `b`. The increment of the AdjacencyMeasure coefficient occurs for the pair `(m,n)`, for which minterm `m` is smaller than minterm `n`, to avoid the multiple accountability.

The matrix obtained in the previous procedure helps to find encoding for which the Hamming-distance matrix is maximally matched to the AdjacencyMeasure matrix. The measure of the match can be expressed in the score of Correlation.

To create all possible encodings of length `k` one has to create all binary codes in the range of `0` to `2^k-1`. The obtained codes are selected in the next step for further computation. To create encodings of output setsystem with `m` blocks, one must select `m` codes and assign them to the output blocks. The number of all possible assignments is equal to the number of all possible subsets of size `m` of the full set of elements of size `2^k`, except for the of encodings which yield identical results due to symmetries.

In the encodings population created we must find all the encodings for which the correlation between the Hamming distance matrix and the Adjacency Measure matrix is the highest. The correlation measure (`score`) is represented as the sum:

\[
Correlation = \sum_{i=0}^{m-2} \sum_{j=i+1}^{m-1} \frac{AdjMeasure(code(i), code(j))}{HammingDistance(code(i), code(j))}
\] (7.4.3)

The encodings with the highest Correlation measure are being preserved in a list of accepted encodings.

The next step is to create the promising SOP/POS realizations for all single-output
subfunctions \( \{g_i\} \). For the cost assessment the same procedures as in the SOP/POS encoding being employed to ensure the saneness of the results.

**Ex. 7.4.1 (Example of MaxAd encoding).** In this example four different encodings of the same sub-function \( g \) is presented. The sub-function \( g \) is a four-input single-output. The incompatibility relation between 16 symbols are shown in Figure 7.22.

After merging the input product setsystem, we get the following four block set-system:

\[
\Pi_{\text{node}} = \{ 0, 5, 9, 12, 13, 14, 15 ; 1, 2, 4, 8 ; 3, 6, 10, 11 ; 7, 12, 13, 14, 15 \}
\]

The symmetrical matrices of merge cost, adjacency and distant coefficients that corresponds to the set-system that will be encoded in this example are presented in Table 7.23, Table 7.24 and Table 7.25 respectively.

Next, the solution that produces the most compact two level description for both binary output sub-function components is selected. Let us compare two encoding

Figure 7.24: Adjacencies matrix.
variants: variant $A \{0; 1; 3; 2\}$ and variant $B \{0; 3; 1; 2\}$.

\[
\Pi_{\text{enc}A} = \{00; 01; 11; 10\} \quad \Pi_{\text{enc}B} = \{00; 11; 01; 10\}
\]

For the variant $\{0; 1; 3; 2\}$ the MaxAd correlation is 33 and predicted cost of two level implementation occupying area 344, while for variant $\{0; 3; 1; 2\}$, these figures are 24 and 616, respectively. The encoding with the highest correlation coefficient between Hamming distance matrix the Adjacency Measure matrix is variant $\{0; 1; 3; 2\}$, hence it is selected for the final implementation of block node97 in presented example.

<table>
<thead>
<tr>
<th></th>
<th>binary</th>
<th>hamming</th>
<th>maxad</th>
<th>random</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>248</td>
<td>280</td>
<td>256</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>96.87 %</td>
<td>109.37 %</td>
<td>100.00 %</td>
<td>106.25 %</td>
</tr>
<tr>
<td>levels</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>150.00 %</td>
<td>150.00 %</td>
<td>100.00 %</td>
<td>150.00 %</td>
</tr>
<tr>
<td>gate count</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>100.00 %</td>
<td>114.28 %</td>
<td>100.00 %</td>
<td>114.28 %</td>
</tr>
<tr>
<td>connections</td>
<td>20</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>95.24 %</td>
<td>100.00 %</td>
<td>100.00 %</td>
<td>100.00 %</td>
</tr>
</tbody>
</table>

Table 7.13: The comparison of four alternative realization of four input sub-function in Example 7.4.1.

In Table 7.13 all four synthesized sub-functions variants encoded with binary (successive codes), hamming (multiplication of unique information items), maximal adjacencies (simplification of logic in sub-function $g$) and random encodings are presented. The corresponding networks are shown in Figure 7.26.

### 7.4.2 Sum of Products and Product of Sums encoding

#### Goal

The goal of the Sum of Product and Product of Sums (SOP/POS) encoding is to find the minimum SOP/POS realization which is feasible for a given complex gate technology library. Similarly as the MAXAd method, the sub-function constructed using this method, represents the first (lower) physical level of a complete SOP/POS realization. The first level consists of gates which individually compute separate terms for the product (POS) or sum (SOP) realization. Normally, outputs of such
7.4. GATE-TARGETED MULTI-VALUED SUB-FUNCTION ENCODING

Figure 7.26: Networks of sub-function \( node_{97} \) in benchmark \( s_{i10_2}.fr.dc.0.50.pla \) encoded using four alternative encodings.

(a) hamming  (b) binary  
(c) random  (d) maxad

Figure 7.27: Example of SOP realization of function \( f \) consisting of four terms.
gates must be then connected to the inputs of the gate which actually produce the corresponding binary function output (sum/product gate). However, our method does not perform this connection process and sum/product gate placement immediately, leaving the final decision on gates used and its actuation to the next decomposition level.

**Ex. 7.4.2 (Sum of Products realization).** The Figure 7.27 shows an example of such encoding for a function \( f \) which consists of four terms. Complex gates necessary to realize such a sub-function include **AND** gates of 3 inputs, because the minimal support for each of the terms is equal to 3. The output **sum** requires the usage of **OR** gate with four inputs, one gate per each **product** gate.

<table>
<thead>
<tr>
<th>Term</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 7.14: Karnaugh table for Example 7.4.2 binary function \( f \).

The minimal term coverage consists of four onset terms presented in Figure 7.15.

<table>
<thead>
<tr>
<th>Term</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) onset

<table>
<thead>
<tr>
<th>Term</th>
<th>0</th>
<th>0</th>
<th>-</th>
<th>-</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(b) offset

Table 7.15: Term coverage of example Boolean function.

The minimal realization can be selected from the two possible realizations: onset or offset cover. During the selection procedure the algorithm must consider all aspects of the physical realization. In the presented example, there are two alternative implementations (see Figure 7.28) to select, namely:

1. **sum of product of the onset terms**, shown in Figure 7.28.a,
2. **product of sum of the offset terms**, shown in Figure 7.28.b.

**Sum of product** requires four product gates – four three-input **AND** gates (andf201, area 32) and a four-input **OR** gate (orf401, area 48), while **product of sum** only two two-input **OR** (orf201, area 32) gates and a two-input **AND** gate (andf201, area 32). Consequently the total, active foot-print occupied by the alternative realizations differ significantly, rendering the selection process based on the physical features, trivial.

Remaining step is decision which out the two alternative realization is use in the final network. It is based solely on the physical feature comparison. Because both realizations take two level of physical gates the only feature to differentiate the
7.4. GATE-TARGETED MULTI-VALUED SUB-FUNCTION ENCODING

Figure 7.28: Sum-of-product and Product-of-sum realizations of Example 7.4.2.

<table>
<thead>
<tr>
<th></th>
<th>SOP</th>
<th>POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>176</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>100.00 %</td>
<td>54.54 %</td>
</tr>
<tr>
<td>levels</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>100.00 %</td>
<td>100.00 %</td>
</tr>
<tr>
<td>gate count</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>100.00 %</td>
<td>60.00 %</td>
</tr>
<tr>
<td>connections</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>100.00 %</td>
<td>37.50 %</td>
</tr>
</tbody>
</table>

Table 7.16: The comparison of alternative realization of sum of product and product of sum realization in Example 7.4.2.

physical cost of realization is foot-print area and number of connections. Table 7.16 presents the comparison, with the foot-print areas taken from the STDCell library (see Appendix B).

Algorithm outline

The technology library defines the restrictions for the SOP/POS realization in a form of:

1. the existence of (a wide) AND and OR gates and their negations,
2. the size of the widest AND and OR gate,

The features of physical realizations which are taken into account to assess the quality and feasibility of the SOP/POS realizations are:

1. the minimal number of terms,
To implement a Boolean function described by a pair of onset and offset, one can use the SOP or POS when implementing the onset or offset terms correspondingly. In consequence, the best realization can be selected from two possible physical realizations: POS - product of sums or SOP - sum of products. SOP/POS encoding procedure takes both into account and selects the better one as the final physical realization.

To consider the cost of either of realizations one must take into account only the sum of the footprint areas, as the predicted delay defined as the number of levels, is equal for all SOP/POS realizations.

Every term in the Boolean function described as the sum of products requires the appropriate logic gate to produce the term, i.e. the term with a minimal support of \( k \) requires the AND gate with \( k \) inputs. Every term, when necessary, requires also input inverters on inverted inputs, i.e. the term with \( n \) literals in the negative polarization requires \( n \) input inverters in the physical realization. To minimize the requirements of the input inverters encoding algorithm utilizes the input inverters sharing for the same literals in different terms. In the network the sharing of common inverters is represented as fork-ing of inverted signals. The sum is implemented in OR gate with as many inputs as the number of product terms. Analogously, every term in the Boolean function described as the product of sums requires the appropriate logic gate to produce the term, i.e. the term with a minimal support of \( k \) requires the OR gate with \( k \) inputs. Similarly as in the case of sum of products, every term, when necessary, requires also input inverters on inverted inputs. The Boolean product of sums is implemented in OR gate with as many inputs as the number of product terms. The final gate count, the physical cost of implementation of sum of products and product of sums, can be expressed in the cost function. Having compared both realizations, the designer can decide which implementation is preferable, according to the optimization criteria.

### 7.5 Sub-function quality estimation

#### 7.5.1 Quality assessment

In this section we focus on the methods of comparison of different realizations of the same Boolean function. All construction algorithms, presented in this chapter, reassure the functional correctness of all realizations. The features that are taken into account when physical realizations of the sub-functions are compared to construct or select the best possible one are the following:

- effective convergence (quality),
- implementation cost,
- encoding quality.
Effective convergence is an ultimate quality measure, for the selection procedure. Subsequent selections are based on other features such as:

- implementation cost:
  - area cost,
  - delay (output level),
- encoding quality:
  - reduction of unique and redundant information,
  - eventual (output) phase(s) correction requirement.

Physical features

Beside the information about the physical features, such as the area and delays, the library also describes the number of physical levels of a (multiple-level or virtual) gate realization. This information, combined with the information of levels at which the input supports are available, allows us to analyze the fitness of multiple-gate (multiple-level) realizations to particular input variables.

An additional level of gates in the physical realization increases the length of some signal path. The method of technology library representation presented in this thesis allows for using the multi-level super-gates, which do not necessarily have symmetric, equal length signal paths for every input. This implies a need to analyze how the physical gates can influence the delay of the resulting network (see Figure 7.29).

![Figure 7.29: Additional level in constructed network due to multi-level network of sub-function $g$.](image)

The proposed modified bottom-up algorithm gives precedence to the physical realizations which are driven by the inputs from the lower levels of the network under construction, and the physical realizations that have outputs to the lower levels of the network. This way the construction algorithm estimates the influence of the constructed sub-function on the overall network and implicitly decomposes the network level by level, until an algorithm moves on to the higher level, because it runs-out of "good" supports to build sub-functions on the current level.
7. SUB-FUNCTION REALIZATION IN THE LIBRARY-BASED INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION

The impact of a sub-function under construction on the overall network can be determined using the following formula:

\[ l = \max\{\text{in}_g\} + \max\{\text{out}_h\} - \max\{\text{out}_h\} \]  \hfill (7.5.1)

where:
- \( l \) – number of additional levels,
- \( \text{in}_g \) – vector containing path length on corresponding inputs of physical realization of sub-function \( g \),
- \( \text{out}_h \) – vector containing level of availability of inputs of sub-function \( g \).

The same method applies when, algorithm calculates the actual delays, computed as the differences between arrival times.

\[
\begin{align*}
(a) & \text{ single output - 2 levels} \\
(b) & \text{ 3 outputs - single level}
\end{align*}
\]

Figure 7.30: Two alternative synthesis steps (partial realizations) presented as a sub-functions with equal level convergence (\( \text{conv}_{\text{level}} = 2 \)).

Logic features

In general, merging some blocks of a set system reduces the amount of information provided by this set system. Let us define the block merging cost \( \text{bmc} \) for any two blocks of \( \pi_g \) as the sum of weights of the elementary information items removed by the merging [20]:

\[
\text{bmc}(B_k, B_l) = \sum_{s_i \in B_k, s_j \in B_l, (s_i | s_j) \in IS(\pi_U)} w(s_i | s_j) \]  \hfill (7.5.2)

Term \( \text{bmc} \) describes how many and how important (unique, almost unique, etc.) elementary information items will be lost if we merge blocks \( B_k \) and \( B_l \) together. Every block merging process has an impact on the \( \text{bmc} \) measure but also on the feasibility of its physical realization using gates from target technology library. Because feasibility is crucial for gate targeted synthesis, the cost measure is used as subsequent comparison step in sorting algorithm. In the direct mapping, the encodings of binary sub-functions are built as the consequence of selected gates. For details in construction algorithm and comparison decisions, please refer for direct mapping to Section 7.3.2, for convergent realizations to 7.3.3 and for special case
of non-convergent, transcoders realizations to 7.3.4. For other sub-function construction procedures, gate targeted encodings, the comparison is based on similar features as for the LUT targeted decomposition, due to a lack of technology constraints limiting the realizations. Further details of the encoding quality assessment are described in corresponding Sections 7.4.1 and Section 7.4.2.

To effectively compare two multi-level sub-networks with respect to their convergence, the notion of signal convergence (see Equation 4.2.2) needs to be extended to account for the influence of number of levels on span of which a signal convergence was achieved. The equation expressing the level convergence was presented in Equation 4.2.3.

The implementation cost is a function of several physical aspects that shows how much it cost to accept a particular constructed sub-function physical realization in the network under construction.

### 7.5.2 Comparison of different sub-function physical realizations

The supports comparison algorithm, due to the fact that sub-support input sets can be both disjoint or have common sub-sets of inputs, must be able to compare physical realizations of such sub-functions. The different input supports sets of the sub-functions \( \{g_i\} \) to be compared need to be transformed into a common representation. The common representation must be used to be able to compare the influence of different realizations of different sub-functions \( g_i \), on the network under construction. To find a common ground (common representation) for comparison, a common information channel must be created. It is created through computation of a (common) super set of input variables for the sub-functions physical pairwise comparison. Both sub-functions \( g_i \) for a pairwise comparison are expressed in a common set of symbols, based on a superset of input variables.

![Comparison of physical realizations of wider and narrower sub-supports.](image)

One of the reasons to increase the number of supports constructed during the promising supports construction is to increase the chances of finding a good direct physical realization, but also to find a potential mutually non-exclusive realizations. Among the most promising supports, there are supports that do not have common input variables, and as such are good candidates to be realized together in fi-
nal network. An example of two mutually exclusive physical realization is shown in Figure 7.32. The construction algorithm based on the effective level convergence of both realizations: one with a block with three outputs \{y_{a0}, y_{a1}, y_{a2}\} with input support \{x_0, x_1, x_2, x_3\}, and concurrently the realization with a block with just two outputs \{y_{b0}, y_{b1}\}, with input support \{x_0, x_1, x_2\}. The latter realization allows to construct a second sub-function of two separately constructed \(n\)-tuples: using gates from technology library in \(n\)-tuple \(g_{twinA}\). Due to exclusiveness of solutions (both are processing separate sub-sets of inputs, and computing separate sets of outputs) both sub-functions, \(g_{twinA}\) and \(g_{twinB}\), can be (and in this case will be) used to construct circuit. Effectively, two multi-output sub-functions \(\{g_{twinB}, g_{twinB}\}\) create a disjoint decomposition on the first level of circuit network. This way, we can compare multiple exclusively disjoint sub-functions realizations with other realizations, to find a highly convergent components of network under construction.

![Figure 7.32: Comparison of physical realizations of mutually exclusive level physical realizations.](image)

In Figure 7.33 the information channel for realization A was widened with input variable \(x_3\), while realization B with variable \(x_0\). The common input variable set consists of four inputs \(\{x_0, x_1, x_2, x_3\}\). The outputs of realization A (B) is extended with repeated variable \(x_3\) (\(x_0\)), extending the output variable set with one extra binary output.

![Figure 7.33: Comparison of physical realizations of two different sub-supports.](image)

This way it is possible to compare the potential influence of different concurrent
physical realizations, which cannot be accepted together. The algorithm has to select one of them and scrap the remaining due to exclusiveness of sub-supports of different solutions. To compare them, a number of their features is used, namely:

- inputs set size,
- level(s) on which its outputs will be available,
- level(s) on which its inputs are taken from,
- number of repeated variables,
- number of blocks of output setsystem (with eventual extra variables of common support),
- effective convergence (computed on span of single block level),
- actual cost of effective convergence expressed in area units,
- decrement of number of blocks of setsystem in the sub-function product setsystem.

First of all, the precedence is given to a sub-function realization of which the input signals are taken from lower levels at the network under construction. Secondly, the realization which implies less duplicated variables is preferred. Further, the minimization of the number of blocks in the output product setsystem is taken into account. The output product setsystem is computed for the common information channel to compare both realizations against a common ground. Then, if either of the two solutions is having more inputs, this one is taken as the more promising. If the resolution is still not taken, the solution with higher effective convergence is preferred. Next, if the number of variables of the entire network converges to the number for which the target technology library provides a majority of gates, meaning there is high probability it is last constructed level of the network, the foot-print area criterion gets precedence.

When several, not necessarily disjoint, input supports are compared regarding their physical sub-function realizations, a set of physical and logical features of all physical realizations is considered in the selection process. The physical realizations that have common inputs in its input supports sets, are mutually exclusive. The consequence of selection of one of them is resignation of all other concurrent alternative realizations. The selection has to be, therefore, performed with utmost care, as it always influences the quality of final network.
In this chapter the methods to construct sub-function physical realization are presented. These methods were either adapted based on the methods formerly used for LUT technology, or created exclusively for the purpose of the research presented in this thesis. Some of these methods are a consequence of a direct mapping of Boolean sub-function into gate library technology, in a single-step approach. Because to our best knowledge, in the research community, there were no attempts to develop a single-step synthesis methodology, such encoding had to be created from
scratch. In Section 7.3.2 the direct sub-function encoding method is described, which
' together with technology library enables direct synthesis method. Other methods of
sub-function encodings, as an alternative to the direct ones, create an opportunity to
search for the physical implementation solution in some specific cases. These meth-
ods are facilitating the construction process in such a way that they split the decom-
position process into a number of smaller ones. Weakly convergent sub-functions
can be also implemented in multi-level sub-network, starting with transcoders. They
can be also encoded into a set of single-output, simpler to decompose sub-functions
using a number of encoding methods. In the end the method of sub-function com-
parison is presented, to tackle the problem of multi-objective implementation cost
and quality estimation. The methodology presented in this chapter enables the de-
composition algorithms presented in Chapter 4 to complete the construction process
with a network of physical gates. Functional decomposition together with technology
library model and sub-function realization methodology creates a framework of meth-
ods and tools to experimentally prove suitability of information-driven decomposition
to complex gates libraries. Having the complete knowledge about the methodology
introduced or recalled in this thesis, in the next chapter the results of experiments
performed using these methods are presented.
7. SUB-FUNCTION REALIZATION IN THE LIBRARY-BASED INFORMATION-DRIVEN GENERAL FUNCTIONAL DECOMPOSITION
A prototype CAD tool for the automatic information-driven decomposition based on circuit synthesis targeted to gate, called IRMA2GATES, was developed in the framework of the research presented in this thesis. IRMA2GATES was built on the top of early version of IRMA2FPGAS, a similar tool, but targeted to LUT-based FPGA circuit synthesis. Due to the differences between the two synthesis targets explained in Section 4.1, the main new parts of IRMA2GATES include a tool for modeling of technology library for the purpose of information-driven general decomposition base circuit, presented in Chapter 6, and a tool for sub-function construction in bottom-up general decomposition targeted to gates. IRMA2GATES was used to perform the experimental research reported in this thesis. The main aim of this experimental research was to verify if the information-driven approach to circuit synthesis proposed by Jóźwiak, can produce high-quality gate-based circuits efficiently and to evaluate the circuit synthesis method presented in the scope of the research work reported in this thesis and implemented in IRMA2GATES. Another aim was to produce several examples to further explain various parts of our circuit synthesis method. To experimentally verify library modeling process and related library parsing and modeling tool, we modeled, among others, the MCNC, STDcell and AMS c35b3 libraries. We also used the models to synthesize several thousands various circuits. The results of these experiments demonstrated that the proposed library modeling process, as well as library parsing and modeling tool, work correctly and are adequate for the single-step information-driven circuit synthesis. The complete circuit construction method discussed in this thesis has been implemented in our information-driven circuit synthesis tool IRMA2GATES. Section 8.1 provides a brief description of circuit characteristics used to compare the circuit synthesis results from IRMA2GATES and SIS. In Section 8.2 a number of example circuits synthesized with IRMA2GATES will be presented. In Section 8.3 the reason and motivation to create extended benchmark set will be presented, with detailed description of methods to create the Boolean functions. Section 8.3 contains comparison of circuits constructed by IRMA2GATES to the results from SIS. Finally, in Section 8.4 we present comparison of results on the category of symmetric Boolean functions. It is well known from the literature [22, 53, 58] and experiments that the traditional circuit synthesis tools, including SIS, produce low-quality circuit results for the most symmetric functions, while our
8. EXPERIMENTAL RESULTS

IRMA2GATES (IRMA2FPGAS) produces extremely good results.

8.1 Measured and compared circuit characteristics

The following characteristics will be compared to show effectiveness of the information-driven general decomposition process performed by tool developed in the course of research presented in this thesis:

Area  The total active areas of particular gates are taken from the target technology library. This figure accounts for the active area of transistor channels, source and drains connections, and potential internal connections between component transistors. It also considers the required spacing between wells and other minimal clearings due to DRC limitations.

Gate levels  The length of critical path - an indication of the circuit delay computed as the maximum number of gates through which any input signal needs to travel in the resulting network to reach one output.

Gatecount  The total number of physical gates in the resulting network. Compared to active area, it indicates what size of gates prefers a given synthesis methods: gates occupying small or large area.

Connections  The total number of connections among the gates circuits. Compared to gate-count it indicates what kind of gates prefers a given synthesis methods: simple (with small number of inputs), or wide (with great number of inputs). Number of connections represents the complexity of the interconnection-network circuits and indicates how difficult it is to place and route the circuit.

If not stated otherwise, the figures presented in this report are representing relative values of the above mentioned characteristics, with respect to the reference results obtained from the synthesis using Berkley’s SIS 1.3 combinational circuit synthesis tool [123]. The relative comparison between the results obtained from our experimental tool IRMA2GATES and SIS is represented as fraction of the corresponding quantitative measures, for these two tools, i.e.:

\[ \Delta = 100\% \cdot \frac{F_{\text{sis}}}{F_{\text{irma}}} \]  \hspace{1cm} (8.1.1)

where:

- \( F_{\text{irma}} \) - represents a result obtained from IRMA2GATES,
- \( F_{\text{sis}} \) - represents a result obtained from SIS.

This representation helps to easily spot the quality difference between the two alternative realizations. The value of \( \Delta \) equal to 100% marks the equality, while values larger or smaller than 100% shows in percent how much a particular feature is smaller or larger respectively (with respect to the referenced SIS realization).

Power consumption is very strongly correlated with area. Therefore, for the purpose of the experiments performed for this thesis the area results were also used to represent the power results [10].
8.2 Examples of circuits synthesized with IRMA2GATES

In this section a number of example circuits synthesized with IRMA2GATES will be discussed to demonstrate which kind of circuits synthesizes IRMA2GATES and to further explain and demonstrate application of some specific procedures used in the synthesis method presented in this thesis and implemented in IRMA2GATES. In particular, the following will be demonstrated:

- speed vs area trade-off for individual sub-functions and its influence on the entire resulting network,
- application of decomposition of a (primary level) sub-function into non-convergent sub-networks (in terms of signal-convergence, see Equation 4.2.2 for details),
- synthesis with use of a presynthesized virtual gates (see Section 6.4.3 for details),
- usage of Shannon and/or Davio expansion in the general decomposition based synthesis.

Ex. 8.2.1 (Single output sub-function '0' of benchmark rd73). This example was selected to show the influence of an adequate selection of a sub-function realization on the entire decomposition. It demonstrates how important it is to select an adequate sub-function realization, especially in the early decomposition stages. In this example two alternative realizations of a sub-function placed in the first network level are presented. These two realizations differ with respect to the number of binary outputs and signal convergence. Consequently, they also differ with respect to the number of inputs of the block \(h\), i.e. of the sub-function that remains after a sub-function \(g\) is constructed and substituted in function \(F\).

Benchmark rd73 from MCNC benchmark suite is a three output benchmark with all three single output component sub-functions being completely symmetric in traditional meaning, where every pair of primary inputs consists of two completely symmetric inputs. Every two inputs can be freely exchanged without consequence of changing the Boolean function.

Let us analyze the decomposition, and specifically, the construction of network of the first of primary output. The function of the first output is completely specified Boolean function and the minimal input support required to calculate the primary output \(0\) is a full set of all seven primary inputs. For this reason, sub-support construction algorithm creates the number of sub-function supports, that is specified as the beam-search size, in this case six, but could be any arbitrarily selected number. Since the function is symmetric, the proposals are basically identical and therefore, the sub-function construction can be performed for one of them and the results can be used for each of the remaining supports.

- \((0, 4, 5, 6)\),
- \((0, 2, 4, 6)\),
- \((0, 1, 5, 6)\),
- \((0, 3, 4, 6)\),
- \((0, 1, 2, 6)\) and
Due to this fact, all sub-function supports proposals are identical with respect to the Boolean function that sub-function implements, as well as logical and physical characteristics of optimal implementation. The information that needs to be processed in the sub-functions constructed using these supports is identical. Therefore, it is sufficient to perform construction for one of the supports proposals. Let us now consider one of the sub-supports for the sub-function construction. All four realizations are identical. It is impossible to make a decision based on the information measures to select a particular sub-support along with a physical implementation of the sub-function. The construction algorithm constructed two convergent proposals, one with signal convergence one and other with signal convergence of two.

\[ g_A \] sub-function realized by two virtual single output components: \( \text{exh}_6996 \) and \( \text{exh}_7ee8 \),

\[ g_B \] sub-function realized by three virtual single output components: \( \text{exh}_1668 \), \( \text{exh}_837f \) and \( \text{exh}_013e \),

Sub-function realizations \( g_A \) and \( g_B \) are both a result of aggregation of virtual blocks (two or three), each implementing single output component sub-function. The first realization, sub-function referred further in this example as \( g_A \), is a straightforward implementation of a set-system that resulted from merging of the sub-functions input product set-system. The latter, referred as \( g_B \), is a result of selection of the best one out of a number of different alternative mergings of the sub-function’s product set-systems and its implementations using a single output virtual gate. An extra freedom was given here by increasing the number of outputs, and therefore, the maximum number of the allowed output product set-system blocks from original 4 (of original 2 binary outputs) to 8 (to 3 outputs).

The comparison of both proposals is presented in Table 8.1. The decision taken by the heuristic algorithm is straightforward. The difference gives an obvious advantage for the sub-function that has higher level convergence (see Equation 4.2.3 for the definition). Even though, there is a greater freedom of encoding in the latter realization, the physical realization obtained in this example was superior to the first realization.

After substitution of this sub-function, the partially decomposed network is presented in Figure 8.3. The free-set inputs remaining, are the three not yet used for any other sub-function. They can be used to build a convergent block with signal convergence of one. The sub-function construction algorithm prepares a sub-function of the three inputs and two outputs with two single output virtual sub-functions: \( \text{exh}_96 \) and \( \text{exh}_17 \). There are two alternative realizations of virtual sub-function \( \text{exh}_17 \).

Before carrying on with the insertion of sub-function on the first level, the main construction algorithm considers other, possible options. In the next decomposition step the following options are considered:

1 feasible (2 blocks, best one with 3 inputs and 2 outputs) placed in the first level,

2 Shannon expansion (with \( \text{var} [v][14_0] : 1 \)),

3 continue the decomposition process with the construction of the next level.
8.2. EXAMPLES OF CIRCUITS SYNTHESIZED WITH IRMA2GATES

Figure 8.1: Convergent realization of sub-function from benchmark rd73.0 presented in Example 8.2.1 using \textit{exh\_6996} and \textit{exh\_7ee8} in the two output sub-function variant $g_A$.

Figure 8.2: Convergent realization of sub-function from benchmark rd73.0 presented in Example 8.2.1 using \textit{exh\_1668}, \textit{exh\_837f} and \textit{exh\_013e} in the three-output sub-function variant $g_B$. 
Table 8.1: The comparison of two alternative realizations of four-input sub-function $g$ in Example 8.2.1.

<table>
<thead>
<tr>
<th></th>
<th>2 outputs</th>
<th>3 outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>280</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td>100 %</td>
<td>120 %</td>
</tr>
<tr>
<td>levels</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>100 %</td>
<td>100 %</td>
</tr>
<tr>
<td>gate count</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>100 %</td>
<td>128 %</td>
</tr>
<tr>
<td>convergence</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>100 %</td>
<td>50 %</td>
</tr>
<tr>
<td>level convergence</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>100 %</td>
<td>50 %</td>
</tr>
</tbody>
</table>

Figure 8.3: The partially decomposed function $F$ of benchmark $rd73.0$ after the first synthesis step in Example 8.2.1.
The decision taken by the construction algorithm can be different, depending on the optimization criterion. The summary of the resulting networks physical features are presented in Table 8.2. If the level construction is continued, the next sub-function will be constructed using the primary inputs that were not used for the sub-support of previously constructed sub-function. This is the preferred option in case of the speed optimization. Let us first compare these two alternative solutions. Once a sub-function of 3 inputs and 2 outputs is inserted (substituted), the remaining Boolean function calculating the primary output has only four inputs and can be implemented using a single 4-input virtual gates. The remaining sub-network is fully implementable using (presynthesized) target technology library gates. The same decision, as pre-

<table>
<thead>
<tr>
<th>IRMA</th>
<th>speed</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>536</td>
<td>480</td>
</tr>
<tr>
<td>100 %</td>
<td>100 %</td>
<td>89.5 %</td>
</tr>
<tr>
<td>levels</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>100 %</td>
<td>100 %</td>
<td>125 %</td>
</tr>
<tr>
<td>gate count</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>100 %</td>
<td>100 %</td>
<td>85.7 %</td>
</tr>
<tr>
<td>connections</td>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>100 %</td>
<td>100 %</td>
<td>102.6 %</td>
</tr>
</tbody>
</table>

Table 8.2: The comparison of two alternative realizations of benchmark rd.73.0 with different optimization targets.
sent above, would result in continuation of network construction, in case of area optimization. An alternative solution will be obtained when the decision to carry on with construction of the first level is not taken. The difference is that in this one the optimization target is minimization of the area. The initial decomposition steps for speed and area optimization criterion are thus the same. The construction algorithm that follows the traditional bottom-up scheme was modified based on the observation that to obtain quick circuits, the processing of information needs to be performed in the first levels of constructed network. This fact can be seen in the difference of two networks obtained from two optimization targets, in Figure 8.5.

Ex. 8.2.2 (Sub-function encoding in benchmark 5xp1.fr.4). In the following example we present an asymmetric function processed by an experimental synthesized tool IRMA2GATES. The results are compared to the results obtained from SIS. Benchmark 5xp1.fr.4 is selected to show how a proper selection of supports and
8.2. EXAMPLES OF CIRCUITS SYNTHESIZED WITH IRMA2GATES

<table>
<thead>
<tr>
<th>name</th>
<th>area</th>
<th>levels</th>
<th>gate-count</th>
<th>connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIS</td>
<td>288</td>
<td>138.50%</td>
<td>5</td>
<td>166.66%</td>
</tr>
<tr>
<td>IRMA</td>
<td>208</td>
<td>100.00%</td>
<td>3</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

Table 8.3: The comparison of circuits synthesized for benchmark 5xp1.fr.4 from Example 8.2.2 by IRMA2FPGAS and SIS.

an adequate encoding of sub-functions constructed using the selected sub-support, influences the quality of the resulting network. Due to the small size the resulting networks obtained with either, the speed or area are equal.

The quality of the resulting network is primarily decided by the selection of the first single-output sub-function with 4 inputs during the construction of first network level. It is recursively implemented in the network as a two level virtual complex gates, comprising of four gates. The three gates are depicted in Figure 8.6 as lv[8_0], lv[8_1] and lv[8_2], while the fourth (output) gate of the selected virtual gate is labeled lv[7_0]. The remaining and the last step of decomposition requires selection of two input gate placed in the final level of the circuit. Due to the fact that the target technology library contains all Boolean functions of two inputs, this step is trivial.

The support selection algorithm produces a number of potentially convergent sub-supports. The most promising supports with the physical implementation of sub-function are listed in Table 8.4. Let us consider variant 3 presented in this example: sub-function realization using two virtual gates exh_lae5 and exh_6999. Each component gate involves 3 physical gates. Virtual gate exh_lae5 is constructed with two first level gates: xorf201 and blf01, and output gate xorf201. Virtual gate exh_6999 is constructed with two first level gates: xorf201 and nanf201, and output gate xorf201. Because the first level gates share the input signals and both virtual gates require identically connected physical gates xorf201, resources can be re-used for the advantage of saved area and connections. Similarly, simple arithmetic sum of all internal connections in the component virtual gates does not necessarily equal the resulting number of connections in the combined sub-network, as the shared connections are counted only once. In the Table 8.5 the final results are listed, with their corresponding active area, number of levels along critical path, as well as the number of physical gates in the network and number of connections between the gates.

All solutions from IRMA2GATES presented in Table 8.5 are faster than reference realization obtained from SIS tool and most of them are still smaller at the same time.

Ex. 8.2.3 (Synthesis of symmetric function rd84.fr.0.pla using transcoders).

The following example presents the transcoders usage for benchmark rd84.fr.0. The primary role of transcoders placed in the first level of the network under construction is to reconstruct information presented in primary inputs to be processed in the successive network levels. The restructuring of information using different encodings, involves translation of information items to preserve or multiply the unique encod-
<table>
<thead>
<tr>
<th>outputs</th>
<th>support</th>
<th>(virtual) gate</th>
<th>area</th>
<th>levels</th>
<th>gate count</th>
<th>connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{i_0};i_1;{i_2;i_3}</td>
<td>exh_17a8</td>
<td>168</td>
<td>2</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>{i_0;i_1;{i_2;i_3}}</td>
<td>exh_lae5</td>
<td>112</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exh_6999</td>
<td>104</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>combined*</td>
<td>176</td>
<td>2</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>{i_1;i_2;i_3;i_6}</td>
<td>xorf201</td>
<td>40</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>blf01</td>
<td>32</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nanf201</td>
<td>24</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>combined*</td>
<td>96</td>
<td>1</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>{i_1;i_2;i_3;i_6}</td>
<td>exh_lae5</td>
<td>112</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exh_6999</td>
<td>104</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>combined</td>
<td>176</td>
<td>2</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>{i_0;i_1;i_3;i_6}</td>
<td>exh_87f0</td>
<td>112</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exh_lee1</td>
<td>104</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>combined</td>
<td>176</td>
<td>2</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>{i_0;i_1;i_3;i_6}</td>
<td>exh_1e8</td>
<td>168</td>
<td>2</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exh_9993</td>
<td>72</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>combined</td>
<td>240</td>
<td>2</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>{i_0;i_1;i_2;i_3}</td>
<td>exh_17a8</td>
<td>168</td>
<td>2</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exh_aeae</td>
<td>112</td>
<td>2</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>combined</td>
<td>280</td>
<td>2</td>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>{i_1;i_2;i_3;i_6}</td>
<td>exh_2c</td>
<td>64</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exh_lae5</td>
<td>112</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exh_0999</td>
<td>72</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>combined</td>
<td>176</td>
<td>2</td>
<td>5</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 8.4: The complete list of all sub-functions considered in the first decomposition step of benchmark 5xp1.fr.4 in Example 8.2.2.

*The combined number of physical gates does not necessarily equal the arithmetic sum of gates counts of component virtual gates of a single output sub-functions. The possibility of sharing gates creates an opportunity of resource reuse and decreasing complexity of the resulting sub-network.
8.2. EXAMPLES OF CIRCUITS SYNTHESIZED WITH IRMA2GATES  

<table>
<thead>
<tr>
<th></th>
<th>area</th>
<th>levels</th>
<th>gates</th>
<th>conn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>208</td>
<td>3</td>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>72.22%</td>
<td>60%</td>
<td>55.55%</td>
<td>80%</td>
</tr>
<tr>
<td>1</td>
<td>240</td>
<td>3</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>83.33%</td>
<td>60%</td>
<td>66.66%</td>
<td>75%</td>
</tr>
<tr>
<td>2</td>
<td>248</td>
<td>3</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>86.11%</td>
<td>60%</td>
<td>77.77%</td>
<td>90%</td>
</tr>
<tr>
<td>3</td>
<td>240</td>
<td>3</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>83.33%</td>
<td>60%</td>
<td>66.66%</td>
<td>75%</td>
</tr>
<tr>
<td>4</td>
<td>232</td>
<td>3</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>80.55%</td>
<td>60%</td>
<td>66.66%</td>
<td>75%</td>
</tr>
<tr>
<td>5</td>
<td>304</td>
<td>4</td>
<td>8</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>105.55%</td>
<td>80%</td>
<td>88.88%</td>
<td>95%</td>
</tr>
<tr>
<td>6</td>
<td>320</td>
<td>3</td>
<td>9</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>111.11%</td>
<td>60%</td>
<td>100.00%</td>
<td>125%</td>
</tr>
<tr>
<td>7</td>
<td>248</td>
<td>4</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>86.11%</td>
<td>80%</td>
<td>77.77%</td>
<td>90%</td>
</tr>
<tr>
<td>SIS</td>
<td>288</td>
<td>5</td>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>100.00%</td>
<td>100%</td>
<td>100.00%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 8.5: The complete list of all final synthesis results obtained during decomposition of benchmark 5xp1.fr.4 in Example 8.2.2 depending on which sub-function variant was initially selected referenced to the decomposition obtained from SIS.

![Network of 5xp1.fr.4 synthesized by SIS and IRMA](image)

Figure 8.6: Network of 5xp1.fr.4 synthesized by SIS and IRMA.
tion items, and at the same time, to expel the redundant information items. The original single output benchmark function of rd84.0 requires processing of 16440 elementary information items. With the introduction of two transcoders consisting of four four-input physical gates each, IRMA2GATES can decrease this number to just 324 elementary information items. Thanks to this reorganization of information, the remaining function is much simpler to decompose and synthesize. The effect of utilization of properly selected eight physical gates on primary level of the network under construction is a high quality, signal-convergent cone of the sub-network, connected between the outputs of the transcoder and primary output of final network.

The Karnaugh tables visualizations in Figure 8.7 shows the transformation of information representation between the inputs and outputs of constructed transcoders. The transcoder is constructed to guarantee no loss of information required to compute the original Boolean function. The first Karnaugh table shows the original form of Boolean function of rd84.0, while the second one shows the Boolean function that remains to be decomposed and synthesized in the successive network levels after using the transcoders. For the sake of clarity of readability, these two Karnaugh tables were simplified to show black-fill squares for on-set minterms, leaving no fill in off-set minterms. The difference between the Karnaugh tables and the difference in the difficulty of implementation of these two functions is huge. Knowing this, it is not surprising that the circuit produced by IRMA2GATES is so much better than the circuit from SIS.

<table>
<thead>
<tr>
<th>name</th>
<th>area</th>
<th>levels</th>
<th>gate-count</th>
<th>connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIS</td>
<td>1168</td>
<td>169.76%</td>
<td>9</td>
<td>180.00%</td>
</tr>
<tr>
<td>IRMA</td>
<td>688</td>
<td>100.00%</td>
<td>5</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

Table 8.6: The comparison of two alternative realization SIS vs IRMA: rd84.0.
Ex. 8.2.4 (Synthesis of a large asymmetric function alu2.fr1.pla). The following example demonstrates a good example of Shannon expansion application. Due to the complexity of function, described by 198 product terms and low convergence of
The only one sub-support constructed by support construction algorithm, decomposition procedure splits problem by means of Shannon expansion.

Introduction of Shannon decomposition with input "j" as a select input produces two factor sub-function: \( f_1 \) with 9 inputs and 172 product terms, and \( f_0 \) with 6 inputs and 26 product terms. Factor sub-function \( f_1 \) is further expanded again, into yet smaller, and easier to decompose sub-function factors: \( f_{1.1} \) of 8 inputs and 96 product terms, and \( f_{1.0} \) of 8, and 71 product terms. The variable "g" was chosen as selector input variable for output multiplexer. After these two steps the complexity of sub-function \( f_{1.1} \) is still high, therefore simplification procedure is used yet another two times: once using Davio expansion, and then again Shannon. Davio expansion yielded two factor sub-function: one \( f_{1.1.1} \) with 7 inputs and 54 product terms, and \( f_{1.1.0} \) with just 4 and only 14 product terms. Shannon expansion applied to factor function \( f_{1.1.1} \), gives smaller and simpler factor sub-functions: \( f_{1.1.1.1} \) (6 inputs and 29 product terms), and \( f_{1.1.1.0} \) (6 inputs and 26 product terms). For further details on the simplification scheme used in this example please refer to Chapter 6.3.12 in [20]. The networks produced by both tools seems equally complex by looking at the Figures 8.9 and 8.10, but the resulting circuits, when compared with respect to the occupied area and number of levels in the critical path, differ significantly. IRMA2GATES produced the circuit that occupies almost half of the active area and one quarter faster circuit. The number of connections is, just as the occupied area, half of the number in the resulting network produced by SIS, while the number of
physical gates even less than half, which proves again, that the information-driven synthesis, coupled with single-step technology mapping tends to use less but larger (wider) gates.

The following example shows how an adequate implementation of $f$ is obtained using a library of presynthesized virtual gates. In Figure 8.11 two alternative resulting circuits are presented to show the influence of the optimization criteria. The optimization criteria change the behavior of quality assessment procedures presented in Chapter 7.5.1.

**Ex. 8.2.5 (Large asymmetric function clip.fr.2.pla).** The optimization target influences the decisions taken during the decomposition process. In this example, the area is optimized and the adequately selected virtual gates of input support of size 4 is selected. It is subsequently decomposed into physical gates, just as unfeasible sub-functions. A substantial difference is that in case of virtual gates there is already guarantee of mapping. Boolean function, implemented in virtual gate, was already synthesized during the presynthesis and the cost of sub-function application in the network under construction is known beforehand. This knowledge helps to determine which realization is preferable at the time of selection. When optimized for speed, blocks selected to be placed in constructed network are using the non-disjoint decomposition (with shorter critical path) and occupying less area than virtual gates counter-candidates.
Figure 8.11: Network of clip.fr.2.pla synthesized by IRMA with two different optimization criteria.
8.2. EXAMPLES OF CIRCUITS SYNTHESIZED WITH IRMA2GATES

<table>
<thead>
<tr>
<th>speed</th>
<th>area</th>
<th>levels</th>
<th>gate-count</th>
<th>connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIS</td>
<td>1056</td>
<td>9</td>
<td>37</td>
<td>89</td>
</tr>
<tr>
<td>IRMA</td>
<td>704</td>
<td>5</td>
<td>18</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>area</th>
<th>speed</th>
<th>area</th>
<th>levels</th>
<th>gate-count</th>
<th>connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIS</td>
<td>1056</td>
<td>9</td>
<td>37</td>
<td>205.55%</td>
<td>197.77%</td>
</tr>
<tr>
<td>IRMA</td>
<td>704</td>
<td>5</td>
<td>18</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

Table 8.8: The comparison of circuits synthesized for benchmark clip.fr.2 by SIS and IRMA2FPGAS with two alternative optimization criteria.

Figure 8.12: Network of clip.fr.2.pla synthesized by SIS.

Ex. 8.2.6 (Large symmetric function e64e.fr.2.pla). The algorithms implemented in IRMA2GATES, when selecting particular implementation of Boolean function, does not treat any gate from technology library differently. It might be easily visible in case of relatively simple but large (also in terms of number of inputs) Boolean functions, in which the wide (with great number of inputs) technology gates, such as wide AND’s and OR’s, can provide great aid in achieving highly convergent network. This particular example shows how the convergent support construction quality, represented in terms of (potential and actual) signal convergence, influences the area optimization of the final network. Since the area occupied by physical gate is not linearly proportional to the number of its inputs (fan-in) \(^1\), it is profitable to use wider physical gates,

\(^1\)In case of typical technology library, including stdcell library.
to achieve a better efficiency in terms of signal convergence per area unit. The average number of inputs in case of the network produced by SIS is 3.13 inputs, while in case of IRMA it is 3.94. Thanks to the higher average signal convergence of all logic nodes in the networks, less gates (nodes) are needed to achieve the same depth of resulting network.

Furthermore, the area occupied by the network is smaller, as well as the number of physical gates comprising the network and number of connections among them. The number of levels is equal in case of IRMA2GATES and SIS, as it is already the minimum length possible.

<table>
<thead>
<tr>
<th>name</th>
<th>area</th>
<th>levels</th>
<th>gate-count</th>
<th>connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIS</td>
<td>760</td>
<td>3</td>
<td>100.00%</td>
<td>72</td>
</tr>
<tr>
<td>IRMA</td>
<td>664</td>
<td>3</td>
<td>100.00%</td>
<td>67</td>
</tr>
</tbody>
</table>

Table 8.9: The comparison of circuits synthesized for benchmark e64e.fr.12 by IRMA2FPGAS and SIS.

Ex. 8.2.7 (Trigonometric function). The following example shows the result of decomposition for the circuit of the first quadrant of trigonometric (sinus) function decomposed as \(n\)-input to \(m\)-output function. The original function \(\sin(x)\) (first quadrant) is discretized into \(2^{10}\) levels of all 1024 possible combinations of 10-output binary function. Each of \(m\) outputs were decomposed separately as \(m\) single output Boolean functions. The total resource utilization comparison of synthesized networks resulted from IRMA2GATES experimental tool and SIS is presented in Table 8.10.

<table>
<thead>
<tr>
<th></th>
<th>SIS</th>
<th>IRMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Sigma) area</td>
<td>14368</td>
<td>86.26%</td>
</tr>
<tr>
<td>critical path</td>
<td>14</td>
<td>175.00%</td>
</tr>
<tr>
<td>(\Sigma) gate-count</td>
<td>493</td>
<td>108.35%</td>
</tr>
<tr>
<td>(\Sigma) connections</td>
<td>1274</td>
<td>107.96%</td>
</tr>
</tbody>
</table>

Table 8.10: The comparison of circuits synthesized for benchmark \(\sin(8\text{in}/10\text{out})\) by IRMA2FPGAS and SIS.

IRMA2GATES constructed an almost twice faster circuit than SIS and of comparable area (with almost 16% larger active area, but with more than 7% less interconnects). The individual single-output component functions synthesized separately can be compared in Tables 8.12 and 8.11.

The logic function complexity is similarly distributed among the single-output functions components. Both in case of the sizes of resulting networks and lengths of their critical paths are similarly correlated to the complexity of each single output component Boolean function. Thanks to better area/speed trade-off optimization, it was possible to create much faster overall circuit. The slowest circuit synthesized by SIS, with 14 levels, implements the \(i^{th}\) output. It is over three times slower than the simplest and quickest component circuit: the \(i^{th}\) output with 4 levels.
Figure 8.13: Network of e64e.fr.12.pla synthesized by SIS and IRMA.
Table 8.11: The complete results of realization with SIS: separate outputs of \( \sin(8 \text{ in}/10 \text{ out}) \).

<table>
<thead>
<tr>
<th>output</th>
<th>SIS area</th>
<th>SIS levels</th>
<th>SIS gate-count</th>
<th>SIS connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>128</td>
<td>100.0%</td>
<td>4</td>
<td>100.00%</td>
</tr>
<tr>
<td>1</td>
<td>368</td>
<td>93.87%</td>
<td>5</td>
<td>125.00%</td>
</tr>
<tr>
<td>2</td>
<td>680</td>
<td>77.98%</td>
<td>8</td>
<td>114.28%</td>
</tr>
<tr>
<td>3</td>
<td>1112</td>
<td>72.00%</td>
<td>9</td>
<td>112.50%</td>
</tr>
<tr>
<td>4</td>
<td>1608</td>
<td>93.48%</td>
<td>11</td>
<td>157.14%</td>
</tr>
<tr>
<td>5</td>
<td>2008</td>
<td>85.66%</td>
<td>11</td>
<td>157.14%</td>
</tr>
<tr>
<td>6</td>
<td>2152</td>
<td>91.80%</td>
<td>14</td>
<td>175.00%</td>
</tr>
<tr>
<td>7</td>
<td>2016</td>
<td>94.03%</td>
<td>9</td>
<td>128.57%</td>
</tr>
<tr>
<td>8</td>
<td>2000</td>
<td>78.86%</td>
<td>11</td>
<td>137.50%</td>
</tr>
<tr>
<td>9</td>
<td>2296</td>
<td>87.50%</td>
<td>8</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

Table 8.12: The complete results of realization with IRMA2GATES: separate outputs of \( \sin(8 \text{ in}/10 \text{ out}) \).

<table>
<thead>
<tr>
<th>output</th>
<th>IRMA area</th>
<th>IRMA levels</th>
<th>IRMA gate-count</th>
<th>IRMA connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>128</td>
<td>4</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>392</td>
<td>4</td>
<td>12</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>872</td>
<td>7</td>
<td>27</td>
<td>70</td>
</tr>
<tr>
<td>3</td>
<td>1544</td>
<td>8</td>
<td>44</td>
<td>118</td>
</tr>
<tr>
<td>4</td>
<td>1720</td>
<td>7</td>
<td>49</td>
<td>122</td>
</tr>
<tr>
<td>5</td>
<td>2352</td>
<td>7</td>
<td>64</td>
<td>164</td>
</tr>
<tr>
<td>6</td>
<td>2344</td>
<td>8</td>
<td>62</td>
<td>161</td>
</tr>
<tr>
<td>7</td>
<td>2144</td>
<td>7</td>
<td>60</td>
<td>147</td>
</tr>
<tr>
<td>8</td>
<td>2536</td>
<td>8</td>
<td>68</td>
<td>178</td>
</tr>
<tr>
<td>9</td>
<td>2624</td>
<td>8</td>
<td>65</td>
<td>179</td>
</tr>
</tbody>
</table>

The same quickest component function is synthesized identically by IRMA2GATES, because it is already the most optimal realization. The slowest component function is only twice slower than the fastest. There are more component functions that bear the same number of levels among the slowest component functions synthesized by IRMA2GATES, than the ones synthesized by SIS. Even though the critical path lengths equal to 8 levels is the same as the critical path lengths for four separate component circuits, the combined critical path of the resulting circuit is also equal to 8. This result is almost twice shorter than the critical path of circuit obtained from SIS. This example shows that the information-driven bottom-up general decomposition based on information relationships and measures, can produce well balanced multi-output Boolean networks, even without using the common sub-function sharing, and lays a good grounds for a multi-output timing driven synthesis.
8.3 SINGLE OUTPUT FUNCTIONS OF MCNC BENCHMARK SUITE

One part of the result quality comparison from IRMA2GATES and SIS was performed on a widely recognized MCNC benchmark set. To avoid the biasing due to sharing of sub-functions between different outputs of a multi-output Boolean function, and only compare the results of sub-functions construction in multi-level circuit synthesis, a set of single-output Boolean functions was prepared, through splitting the multi-output function from the MCNC benchmark-suite. Over two thousands of single-output functions were constructed this way and examined. In all experiments, for the circuit synthesis with SIS, the following script was used:

```
sweep;
eliminate -1
simplify -m nocomp
eliminate -1
sweep; eliminate 5
simplify -m nocomp
resub -a
fx
resub -a;
sweep
eliminate -1;
sweep
full_simplify -m nocomp
read_library stdcell.genlib
map
phase -g
```

In the above script, the technology library STDCell stdcell.genlib loaded with the directive `read_library` is the same technology library as used for the circuit synthesis with IRMA2GATES. The contents of STDCell library is presented in Appendix B. The comparison of the circuit synthesis results from our experimental tool IRMA2GATES to the results from SIS on the complete single-output MCNC benchmark set is presented in a graphical form in Figure 8.14. Even though the total gate area of the circuits obtained from SIS is slightly smaller, the difference is less than 3.5%, while IRMA2GATES constructs much faster circuits than SIS - the
difference with respect to the speed of resulting circuits is, on average, exceeding 20%. Moreover, the slightly smaller gate area is compensated by more and longer interconnects, what results in a larger interconnect area for the circuits from SIS. This clearly shows the high effectiveness of the method presented in this thesis in producing fast and compact circuits. In Figure 8.15, the very high effectiveness of the information-driven general decomposition for symmetric functions is presented. It was already noticed in [58] on the symmetric sub-set of the MCNC benchmarks set. IRMA2GATES constructed both: more than 25% faster and more than 25% smaller circuits than SIS.

The MCNC benchmark set was used as the basic set of Boolean functions to experimentally analyze the work of the synthesis method considered in this thesis and related tool IRMA2GATES. The main experimental analysis task was to compare the quality of circuits synthesized with IRMA2GATES to these from SIS for a number of different classes of Boolean functions.

The benchmarks were subdivided into classes depending on values of the following three function characteristics: the function symmetry, the specification (whether they are completely or incompletely specified Boolean functions) and the size of their input sets.

Due to the fact that the degree of function symmetry can be expressed as a number of symmetric input pairs, the benchmarks were subdivided into:

- completely symmetric - all inputs are mutually symmetric, i.e. each input variable can be swapped with any of other input variable and this does not change the function value,
- partially symmetric/asymmetric - neither completely symmetric nor asymmetric,
- asymmetric - no input is symmetric with any other input.

In relation to the number of don’t cares the functions have been subdivided into:

- completely specified (without don’t cares),
- incompletely specified (less than 20% of don’t cares),
- weakly specified (more than 20% of don’t cares).

With respect to the input size, three function categories were distinguished:
8.3. SINGLE OUTPUT FUNCTIONS OF MCNC BENCHMARK SUITE

<table>
<thead>
<tr>
<th>Classification</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Symmetric</strong></td>
<td>191</td>
<td>22</td>
<td>73</td>
</tr>
<tr>
<td><strong>Partially Spec.</strong></td>
<td>93</td>
<td>122</td>
<td>46</td>
</tr>
<tr>
<td><strong>Asymmetric</strong></td>
<td>375</td>
<td>226</td>
<td>165</td>
</tr>
<tr>
<td><strong>Incompletely Spec.</strong></td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Partially Spec.</strong></td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Asymmetric</strong></td>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Weakly Spec.</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Partially Spec.</strong></td>
<td>7</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td><strong>Asymmetric</strong></td>
<td>8</td>
<td>18</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 8.13: The classification of the single-output MCNC functions.

- small functions (less or equal than 8 inputs),
- medium functions (more than 8, less or equal than 16 inputs),
- large functions (more than 16 inputs).

After performing this categorization of the MCNC benchmark set into corresponding classes, it is clearly visible that the MCNC benchmark set is far from being representative. It does not at all cover most classes of incompletely and weakly specified functions (see Table 8.13). Also the medium symmetric and large partially symmetric functions are clearly underrepresented in the MCNC benchmark set. Therefore, for the analysis and comparison of the IRMA2GATES to SIS on the incompletely specified functions and for the total analysis and comparison, the completely specified MCNC functions were used as base to produce incompletely and weakly specified functions through a random injection of don’t cares in the original MCNC functions. Because most of the multiple-output functions in the MCNC set involve very similar or even identical single-output functions, these repeated Boolean functions had to be eliminated to create a result not biased to particular functions or kind of functions. Failing to do so might lead to over-representation of a certain type of Boolean functions, hence biasing of the results. A well representative benchmark set of Boolean function should contain a spectrum of Boolean functions representative to all possible kinds of functions, with respect to their computation characteristics, as well as their size and complexity. The matter of size is resolved through limitation of the smallest benchmarks to minimum 4 input variables and balancing the sets of small, medium and large functions. Smaller benchmarks do not reveal the influence of the logic synthesis method, but rather the efficiency of the technology mapping alone. Beside the number of inputs, the function complexity can also be determined by the number of terms in functions’ term description. Even though the term description is not guaranteed to be canonical, in case of the number of terms as low as a few, one might consider the synthesis of Boolean function of such size as trivial, and one should expect the result to be (in most cases, if not always) optimal, with respect to both the area and speed of resulting network. It is, therefore, pointless to include...
Table 8.14: The number of product terms of small single-output MCNC functions.

<table>
<thead>
<tr>
<th>name</th>
<th>.p</th>
<th>name</th>
<th>.p</th>
<th>name</th>
<th>.p</th>
<th>name</th>
<th>.p</th>
</tr>
</thead>
<tbody>
<tr>
<td>e64.fr.10</td>
<td>5</td>
<td>cm42a.fr.9</td>
<td>5</td>
<td>cm42a.fr.8</td>
<td>5</td>
<td>cm42a.fr.7</td>
<td>5</td>
</tr>
<tr>
<td>cm42a.fr.6</td>
<td>5</td>
<td>cm42a.fr.5</td>
<td>5</td>
<td>cm42a.fr.4</td>
<td>5</td>
<td>cm42a.fr.3</td>
<td>5</td>
</tr>
<tr>
<td>cm42a.fr.2</td>
<td>5</td>
<td>cm42a.fr.1</td>
<td>5</td>
<td>cc.fr.1</td>
<td>5</td>
<td>b9.fr.9</td>
<td>5</td>
</tr>
<tr>
<td>b9.fr.17</td>
<td>5</td>
<td>pdc.fr.17</td>
<td>4</td>
<td>pdc.fr.16</td>
<td>4</td>
<td>misex2.fr.17</td>
<td>4</td>
</tr>
<tr>
<td>ldd.fr.7</td>
<td>4</td>
<td>ldd.fr.6</td>
<td>4</td>
<td>ldd.fr.5</td>
<td>4</td>
<td>ldd.fr.4</td>
<td>4</td>
</tr>
<tr>
<td>i7.fr.60</td>
<td>4</td>
<td>i1.fr.12</td>
<td>4</td>
<td>i1.fr.10</td>
<td>4</td>
<td>f51m.fr.6</td>
<td>4</td>
</tr>
<tr>
<td>ex5.fr.1</td>
<td>4</td>
<td>ex5.fr.0</td>
<td>4</td>
<td>e64.fr.11</td>
<td>4</td>
<td>cm162a.fr.4</td>
<td>4</td>
</tr>
<tr>
<td>b9.fr.16</td>
<td>4</td>
<td>apex6.fr.89</td>
<td>4</td>
<td>alu2.fr.2</td>
<td>4</td>
<td>5xp1.fr.7</td>
<td>4</td>
</tr>
<tr>
<td>pcler8.fr.1</td>
<td>3</td>
<td>lal.fr.5</td>
<td>3</td>
<td>i1.fr.4</td>
<td>3</td>
<td>duke2.fr.4</td>
<td>3</td>
</tr>
<tr>
<td>cu.fr.9</td>
<td>3</td>
<td>apex6.fr.46</td>
<td>3</td>
<td>apex6.fr.0</td>
<td>3</td>
<td>alu2.fr.3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 8.15: The classification of functions from the extended benchmark set created by using completely specified functions of the original MCNC set and extending them by adding random DC-set conditions to create their incompletely and weakly specified modifications, and by removing the duplicated (very similar) functions.

<table>
<thead>
<tr>
<th></th>
<th>small</th>
<th>medium</th>
<th>large</th>
</tr>
</thead>
<tbody>
<tr>
<td>completely spec.</td>
<td>symmetric</td>
<td>40</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>partially s.</td>
<td>65</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>asymmetric</td>
<td>120</td>
<td>140</td>
</tr>
<tr>
<td>incompletely spec.</td>
<td>symmetric</td>
<td>22</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>partially s.</td>
<td>28</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>asymmetric</td>
<td>99</td>
<td>257</td>
</tr>
<tr>
<td>weakly spec.</td>
<td>symmetric</td>
<td>48</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>partially s.</td>
<td>154</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td>asymmetric</td>
<td>407</td>
<td>660</td>
</tr>
</tbody>
</table>

Such small functions into the benchmark set. Another reason to omit them is due to limited option to “inject” DC terms. Replacement of random product terms with DC terms requires a reasonably high number of product terms to begin with.

To expand the sub-set of medium and large symmetric functions, a number of generated symmetric functions were produced and added into the medium category. The generated functions are Boolean functions that compute an arbitrary selected polarization output, for an arbitrarily selected number of high (low) states present in function inputs. Such function can be, for example, a 10 input function that produces state 1 on its output for every appearance of either 1, 3, 5, 7 or 9 zeros on the function inputs. The complete list of generated functions is shown in Table 8.16. Column “count” denotes the number of 1s (0s) that are counted by a given generated function. Column “inverted inputs” enumerates the input indexes that are, as the name
Table 8.16: The list of generated symmetric functions, with their incompletely and weakly specified modifications.

<table>
<thead>
<tr>
<th>name</th>
<th>count</th>
<th>inverted inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_i10_30.fr.pla</td>
<td>1,3,5,7,9</td>
<td>2,4,6,8</td>
</tr>
<tr>
<td>s_i10_31.fr.pla</td>
<td>2,4,6,8</td>
<td></td>
</tr>
<tr>
<td>s_i10_32.fr.pla</td>
<td>1,3,5,7,9</td>
<td>2,4,6,8</td>
</tr>
<tr>
<td>s_i10_33.fr.pla</td>
<td>2,4,6,8</td>
<td>1,3,5,7,9</td>
</tr>
<tr>
<td>s_i10_34.fr.pla</td>
<td>1,3,5,7,9</td>
<td>0,1,3,7</td>
</tr>
<tr>
<td>s_i10_35.fr.pla</td>
<td>4,5,7</td>
<td>5</td>
</tr>
<tr>
<td>s_i10_36.fr.pla</td>
<td>0,3,4,5,6,7</td>
<td>0,2,6,7,8,9</td>
</tr>
<tr>
<td>s_i10_37.fr.pla</td>
<td>5</td>
<td>0,2,4,5,7</td>
</tr>
<tr>
<td>s_i10_38.fr.pla</td>
<td>1,3,5,7</td>
<td>0,2,4,5,7</td>
</tr>
<tr>
<td>s_i10_39.fr.pla</td>
<td>0,2,4,5,8,9</td>
<td></td>
</tr>
<tr>
<td>s_i10_40.fr.pla</td>
<td>0,4,5,6,7,8</td>
<td>1,5,9</td>
</tr>
<tr>
<td>s_i10_41.fr.pla</td>
<td>1,8,9</td>
<td>0,2,4,5,6,7</td>
</tr>
<tr>
<td>s_i10_42.fr.pla</td>
<td>5</td>
<td>1,3,5,7,8</td>
</tr>
<tr>
<td>s_i10_43.fr.pla</td>
<td>1,2,3,4,6</td>
<td>1,2,7,8</td>
</tr>
</tbody>
</table>

Figure 8.16: Comparison of IRMA2GATES to SIS on the extended benchmark set.

suggests, logically inverted. For each completely specified generated symmetric function a number of incompletely and weakly specified variants were derived from a completely specified base function. The modification involves creation of additional DC terms in on-set and/or off-set in the following quantities 0/10, 10/0, 0/50, 15/15, 25/25, 35/35 and 50/0, where the first number denotes percentage of DC terms in the on-set, while second in the off-set.

Finally, the complete extended benchmark set with the number of functions in particular classes is presented in Table 8.17.

The comparison of IRMA2GATES to SIS on the complete extended benchmark set is presented in Figure 8.16. It clearly demonstrates that if the MCNC benchmark set is made more representative than IRMA2GATES, it wins much more upon SIS in all compared parameters than on the original - non representative MCNC benchmark set.
8.4 Comparison on symmetric functions

To show the especially high quality of circuit synthesis using IRMA2GATES for symmetric Boolean functions, we prepared a well representative benchmark set of symmetric functions that consists of all symmetric functions from the MCNC suite, a number of symmetric industry examples and some generated symmetric benchmarks. The Figure 8.15 shows the comparison of results from IRMA2GATES and SIS on the set of symmetric single-output Boolean functions. The generated single output functions are listed in Table 8.16. Next to the completely specified base functions a number of their incompletely and weakly specified variants were created, making two extra categories of symmetric benchmarks: incompletely specified and weakly specified.

In Figure 8.17 the synthesis results from our IRMA2GATES are compared to the results from SIS 1.3 [123] regarding the area, gate-count and number of gate levels on the critical path (delay) for several MCNC benchmarks [92] and other popular functions. The results from IRMA2GATES are on average 42% better regarding area, 70% better regarding the number of gates, and 30% better regarding the number of gate levels than from SIS.

Table 8.17: The classification of the complete extended benchmark set.

<table>
<thead>
<tr>
<th>Type</th>
<th>Symmetric</th>
<th>Partially Spec.</th>
<th>Asymmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completely Spec.</td>
<td>64/40/74</td>
<td>76/60/45</td>
<td>109/117/94</td>
</tr>
<tr>
<td>Incompletely Spec.</td>
<td>47/47/77</td>
<td>42/67/88</td>
<td>88/165/200</td>
</tr>
<tr>
<td>Weakly Spec.</td>
<td>113/125/109</td>
<td>124/201/156</td>
<td>176/207/192</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Area</th>
<th>142.36%</th>
<th>Levels</th>
<th>130.95%</th>
<th>Gate Count</th>
<th>169.94%</th>
<th>Connections</th>
<th>137.93%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>125%</td>
<td>150%</td>
<td>175%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.17: Comparison of IRMA2GATES to SIS on the MCNC symmetric single-output Boolean functions.
The following comparisons refer to the extended symmetric functions benchmark set created for the unbiased comparison of results between IRMA2GATES and SIS. The first one (Figure 8.18) contains the comparison of results for circuits optimized for speed, while the second one (Figure 8.19) the results for circuits optimized for area. The reference results in both cases are the very same results produced by SIS 1.3, therefore the influence of optimization can be seen when these two total results are compared together. The optimization for area helped to achieve better results with respect to the active area (109.9% for speed goal vs. 110.5% for area optimization goal). The optimization for speed gave, on average, significantly faster results (133.1% for speed goal vs. 130.8% for area optimization goal). Sub-function construction procedures responsible for optimization are presented in Chapter 7.5.1.

The graphs in Figure 8.20 and 8.21 present the comparison of results obtained from the synthesis of incompletely specified functions. The benchmark set used in this test was obtained from various industry examples, including:

- combinational logic of final state machines,
- conditional logic corresponding to Verilog/VHDL case statements,
- a sub-set of incompletely specified functions from the MCNC benchmark suite.

8.5 Comparison on incompletely specified functions

The graphs in Figure 8.20 and 8.21 present the comparison of results obtained from the synthesis of incompletely specified functions. The benchmark set used in this test was obtained from various industry examples, including:

- combinational logic of final state machines,
- conditional logic corresponding to Verilog/VHDL case statements,
- a sub-set of incompletely specified functions from the MCNC benchmark suite.
As noted in [59], the synthesis method previously developed to target LUT-based FPGAs in a form of an experimental tool IRMA2FPGAS is especially effective for symmetric functions or obtained from symmetric functions by “don’t care” insertion. The results of comparison presented in this sections show that the synthesis method is still effective for synthesis that targets technology gate library. The synthesis method based on the bottom-up general functional decomposition and theory of information relationship measures information-driven can produce good results, regardless of the technology target.

![Comparison of IRMA2GA TES to SIS on the extended benchmark set for the weakly specified single-output Boolean functions.](image1)

Figure 8.20: Comparison of IRMA2GATES to SIS on the extended benchmark set for the weakly specified single-output Boolean functions.

8.6 Conclusions

In this Chapter, the results of a new information-driven circuit synthesis method was presented. The features of resulting circuits that were used to characterize the quality obtained from the compared tools were presented in the introductory part. Further, a number of examples of circuit synthesized by experimental tool IRMA2GATES were presented to highlight the role of specific aspects of the decomposition method implemented in it. Specifically, the role and usage of a number of its aspects were highlighted:

- an adequate selection of a sub-function realization throughout the entire process of decomposition,

![Comparison of IRMA2GATES to SIS on the extended benchmark set for the incompletely specified single-output Boolean functions.](image2)

Figure 8.21: Comparison of IRMA2GATES to SIS on the extended benchmark set for the incompletely specified single-output Boolean functions.
8.6. CONCLUSIONS

- an area/speed trade-off as a key feature during the decomposition, important in case of asymmetric functions,
- an application of signal non-convergent sub-functions, presented in details in Section 7.3.4,
- usage of Shannon expansion in case of a large asymmetric functions, helps (bi-)partition a large function into a smaller and simpler co-factor sub-function,
- presynthesized virtual gate library gives a very precise prediction of the physical costs of sub-function utilization in the circuit under construction,
- presynthesized virtual gate library helps synthesize a large asymmetric function with relatively simple logic and promotes utilization of wider physical gates.

Presented examples had shown the significant improvements of information driven decomposition against traditional methods of synthesis. With a special attention, a number of examples were presented to show applicability and suitability of information-driven synthesis method to guide the sub-functions construction process, help to balance the speed-area tradeoff, produce well signal-convergent resulting circuits, with help of non-convergent sub-functions at the first levels of synthesized circuit, as well as a number of examples showing various heuristics described in previous chapters that control the sub-functions constructions.

Further, the motivation and methods used to create an extended benchmark set and the comparison results based on this benchmark set were presented.

In Figure 8.17 the synthesis results from our IRMA2GATES are compared to the results from the well known UC Berkeley's tool SIS 1.3 [123] regarding the area, gate-count and number of gate levels on the critical path (delay) for several MCNC benchmarks [92] and other popular functions. The results from IRMA2GATES are on average 42% better regarding area, 70% better regarding the number of gates and 30% better regarding the number of gate levels than from SIS. These results demonstrate that IRMA2GATES is especially effective for the symmetric, quasi-symmetric and incompletely specified functions, producing significantly faster and smaller circuits.
Chapter 9

Conclusion

The main objective of the research reported in this thesis was to demonstrate that the information-driven bottom-up general decomposition based on information relationships and measures can produce high-quality results and can be performed efficiently for circuits composed of logic gates. This had to be performed through the development of an adequate circuit synthesis method, implementation of the corresponding EDA synthesis tool and execution of the related experimental research.

The circuit synthesis method is based on the prior research performed by Jóźwiak, Volf and Chojnacki [20, 22, 49, 51, 58, 63, 108, 133, 134, 136]. The method is a direct extension of the method proposed by Jóźwiak and Chojnacki for LUT-based FPGA circuits [23, 59–62, 135]. The extension required development of two major new parts: gate library modeling for the purpose of the information-driven decomposition (see Chapter 6) and gate-targeted sub-function construction in the information-driven decomposition (see Chapter 7), as well as, several secondary new parts and smaller modifications. The development of the two major new parts involved an extensive research of the related problems [52–56].

The method considerably differs from all other known methods. It is based on our original information-driven approach to circuit synthesis and replaces the traditional dual-step process of technology independent logic synthesis. Instead, it consists of technology mapping with a single-step direct circuit synthesis (direct mapping) into the gates of a given technology library, when directly accounting for the actual implementation costs. The single-step circuit synthesis process requires availability of adequately complete and accurate information on the logical and physical features of the technology gates from the very beginning of the circuit synthesis process, as well as, an effective and efficient usage of this information throughout the whole process. To satisfy these requirements, we developed a new library modeling method (see Chapter 6). The method is implemented in the form of a library parsing and modeling tool that automatically creates an adequate library model in the form of a homogeneous Boolean function realization library, through constructing efficient data structures and filling them with the required information on the gates’ logic and physical features. This library model enables an effective and efficient multi-valued sub-function construction in the information-driven decomposition process, as well as, the multi-objective circuit optimization and effective trade-off exploitation among...
its area, delay and power consumption.

Our tools construct substantially faster and smaller circuits than SIS and enable a multi-objective circuit optimization, trade-off exploitation and very flexible circuit structuring.

The sub-function construction methods developed and researched are facilitating the single-step circuit synthesis process in the actual search for the physical realization of sub-functions in the form of a network of interconnected gates from the target technology library. The sub-function construction methodology involves two different major approaches: direct construction and construction involving encoding of the multi-valued sub-function. Both methods are aiming at the simplification of the resulting single-output sub-functions, as well as, at the same time, are aiming at the simplification of the remaining image function $h$. Though contradictory, both targets can be achieved to some extent, and with an efficient trade-off algorithm the overall network can be optimized. These two methods, the sub-function direct construction and the encoding of the multi-valued sub-function, use different approaches to minimize the complexity of the resulting encoded sub-function $g$, being the same as the original $g$, but expressed as a combination of several single-output binary functions through encoding. The single output functions can be further decomposed into small and fast sub-circuits. Thanks to the single-step synthesis approach, and the close ties with the target technology through the quality estimation using a pre-characterized technology library, the optimization target can be easily balanced between area, speed, or any combination of physical features describing every gate in technology library. The direct construction algorithm gives the complete information about the quality of the circuit partially constructed in the early stage of the synthesis process. This allows avoiding the multi-stage decomposition process in which the resulting network is improved through partial re-decomposition of the resulting network, but instead it produces (close-to) optimal networks. The direct construction method is an original invention of the author of this thesis. To our knowledge, to this day, neither in the industrial synthesis tools, nor in the scientific publication, there was no similar method described, nor used. It is one of the key novel developments of the research project documented in this thesis.

The new developed method, has been implemented in the form of an automatic circuit synthesis tool targeted to gate realizations of circuits: IRMA2GATES. The tool has been used for extensive experimental research of the developed method. The main aim of the experiments was to verify if the information-driven general decomposition approach is able to produce high-quality gates-based circuits efficiently. Another aim was to analyze the quality of the main methods developed in the scope of this research. Next aim was to provide several synthesis examples to illustrate and better explain several important aspects of the new circuit synthesis method developed. Among others, the following aspects were further explained when using the examples: speed vs area trade-offs of individual sub-functions, non-(signal)-convergent sub-functions, synthesis using virtual gates and function simplification using expansions. The experimental research has demonstrated that substantially better synthesis results can be obtained from the information-driven general decomposition than from the traditional methods of synthesis represented by SIS in the experiments. A number of the experimental results clearly demonstrates a very high effectiveness of the information-driven synthesis method to a wide range of Boolean
functions (especially symmetric and incompletely specified). The results obtained from our experimental tool IRMA2GATES are on average substantially better than those obtained from SIS. The difference is even greater for the symmetric (up to 30% better regarding speed, and 42% regarding area) and incompletely specified Boolean functions (more than 10% regarding area and more than 30% regarding speed). Based on the fundamental differences of the two circuit synthesis methodologies, compared in the course of this research, we conclude that the high quality of the results obtained from our experimental tool are mainly the consequence of the information-driven general decomposition approach, bottom-up decomposition process and single-step construction of sub-functions therein.

Summing up, we developed a new, effective, efficient and very flexible circuit synthesis technology adequate for the modern synthesis targets involving logic gates. The results of the experimental research clearly demonstrate that our information-driven single-step circuit synthesis method outperforms the traditional dual-step synthesis methods represented by SIS. Consequently, the research presented in this thesis demonstrates that the information-driven approach to circuit synthesis based on the general decomposition and information relationship measures is an effective and efficient approach to the synthesis of Boolean functions targeted to gate libraries. From the above, it should be evident, that the objectives of the research reported in this thesis have been fully realized.
Bibliography


symmetric functions to the CA-type FPGAs. In Proceedings of the 38th Mid-


[26] Jason Cong and Kirill Minkovich. Optimality study of logic synthesis for lut-
based fpgas. In FPGA ’06: Proceedings of the 2006 ACM/SIGDA 14th inter-
national symposium on Field programmable gate arrays, pages 33–40, New York, NY, USA, 2006. ACM.

[27] Vinicius Correia and Andre; Reis. Advanced technology mapping for standard-
cell generators. In SBCCI ’04: Proceedings of the 17th symposium on Inte-

puter Society.

[29] H. A. Curtis. A generalized tree circuits. Journal of the Association for Com-


[34] K. Eckl, C. Legl, and B. Wurth. An implicit approach to functional decomposi-


[92] MCNC. Collaborative Benchmarking Laboratory, Department of Computer Science at North Carolina State University. http://www.cbl.ncsu.edu/.


[126] SIA. The 2005 international technology roadmap for semiconductors, 2005. San Jose, CA, USA.


## Appendix A

### Results of experiments

#### A.1 Generated single output symmetric functions

<table>
<thead>
<tr>
<th>Circuit name</th>
<th>IRMA A</th>
<th>IRMA L</th>
<th>IRMA C</th>
<th>SIS A</th>
<th>SIS L</th>
<th>SIS C</th>
<th>difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_i10_1.fr.dc.0.20</td>
<td>1928</td>
<td>13</td>
<td>61</td>
<td>3096</td>
<td>15</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>s_i10_1.fr.dc.0.50</td>
<td>1624</td>
<td>9</td>
<td>48</td>
<td>2888</td>
<td>13</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>s_i10_1.fr.dc.0.70</td>
<td>1312</td>
<td>10</td>
<td>41</td>
<td>2672</td>
<td>12</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>s_i10_1.fr.dc.10.10</td>
<td>1504</td>
<td>10</td>
<td>44</td>
<td>3312</td>
<td>11</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>s_i10_1.fr.dc.20.0</td>
<td>1840</td>
<td>12</td>
<td>58</td>
<td>3368</td>
<td>13</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>s_i10_1.fr.dc.25.25</td>
<td>1536</td>
<td>10</td>
<td>45</td>
<td>3608</td>
<td>11</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>s_i10_1.fr.dc.35.35</td>
<td>1648</td>
<td>11</td>
<td>50</td>
<td>3416</td>
<td>10</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>s_i10_1.fr.dc.50.0</td>
<td>1592</td>
<td>10</td>
<td>48</td>
<td>4456</td>
<td>11</td>
<td>151</td>
<td></td>
</tr>
<tr>
<td>s_i10_1.fr.dc.70.0</td>
<td>1840</td>
<td>11</td>
<td>54</td>
<td>3376</td>
<td>12</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>s_i10_1.fr</td>
<td>1616</td>
<td>11</td>
<td>51</td>
<td>2968</td>
<td>12</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>s_i10_1.fr.inv</td>
<td>1752</td>
<td>10</td>
<td>55</td>
<td>2944</td>
<td>13</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>s_i10_1.fr.dc.0.20</td>
<td>1880</td>
<td>10</td>
<td>57</td>
<td>4008</td>
<td>12</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>s_i10_2.fr.dc.0.50</td>
<td>1728</td>
<td>10</td>
<td>51</td>
<td>3936</td>
<td>11</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>s_i10_2.fr.dc.70.0</td>
<td>1728</td>
<td>10</td>
<td>54</td>
<td>3032</td>
<td>11</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>s_i10_2.fr.inv</td>
<td>2104</td>
<td>11</td>
<td>51</td>
<td>4104</td>
<td>15</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>s_i10_2.fr</td>
<td>1560</td>
<td>10</td>
<td>50</td>
<td>4256</td>
<td>12</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>s_i10_2.inv</td>
<td>1512</td>
<td>9</td>
<td>44</td>
<td>4480</td>
<td>16</td>
<td>151</td>
<td></td>
</tr>
<tr>
<td>s_i10_3.fr.dc.0.20</td>
<td>1704</td>
<td>9</td>
<td>41</td>
<td>2056</td>
<td>10</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>s_i10_3.fr.dc.0.50</td>
<td>2432</td>
<td>13</td>
<td>77</td>
<td>2464</td>
<td>11</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>s_i10_3.fr.inv</td>
<td>3536</td>
<td>13</td>
<td>104</td>
<td>2512</td>
<td>12</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>s_i10_3.fr.dc.10.10</td>
<td>1704</td>
<td>11</td>
<td>49</td>
<td>2808</td>
<td>11</td>
<td>95</td>
<td></td>
</tr>
</tbody>
</table>

*continued on next page*
### A. RESULTS OF EXPERIMENTS

<table>
<thead>
<tr>
<th>name</th>
<th>IRMA A</th>
<th>IRMA L</th>
<th>IRMA C</th>
<th>SIS A</th>
<th>SIS L</th>
<th>SIS C</th>
<th>difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_i10_3.fr.dc.20.0</td>
<td>1432</td>
<td>10</td>
<td>40</td>
<td>5000</td>
<td>13</td>
<td>167</td>
<td>&lt;</td>
</tr>
<tr>
<td>s_i10_3.fr.dc.25.25</td>
<td>1416</td>
<td>10</td>
<td>42</td>
<td>5344</td>
<td>13</td>
<td>175</td>
<td>&lt;</td>
</tr>
<tr>
<td>s_i10_3.fr.dc.35.35</td>
<td>1536</td>
<td>10</td>
<td>46</td>
<td>5984</td>
<td>13</td>
<td>197</td>
<td>&lt;</td>
</tr>
<tr>
<td>s_i10_3.fr.dc.50.0</td>
<td>2128</td>
<td>12</td>
<td>65</td>
<td>6368</td>
<td>15</td>
<td>209</td>
<td>&lt;</td>
</tr>
<tr>
<td>s_i10_3.fr.dc.70.0</td>
<td>1856</td>
<td>8</td>
<td>51</td>
<td>5376</td>
<td>12</td>
<td>177</td>
<td>&lt;</td>
</tr>
<tr>
<td>s_i10_3.fr.inv</td>
<td>1464</td>
<td>9</td>
<td>40</td>
<td>1544</td>
<td>12</td>
<td>51</td>
<td>&lt;</td>
</tr>
<tr>
<td>s_i10_3.fr</td>
<td>1456</td>
<td>8</td>
<td>41</td>
<td>1880</td>
<td>10</td>
<td>62</td>
<td>&lt;</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>57400</strong></td>
<td><strong>335</strong></td>
<td><strong>1714</strong></td>
<td><strong>122584</strong></td>
<td><strong>401</strong></td>
<td><strong>4106</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Ratio (%)</strong></td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>213</td>
<td>119</td>
<td>239</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B

STDCell technology library
cell(invf101) {
    area : 16;
    cell_footprint : "inv";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "A'";
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A"
        }
    }
}

Figure B.1: NOT

cell(norf201) {
    area : 24;
    cell_footprint : "nor2";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(B) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "(A+B)'";
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A B"
        }
    }
}

Figure B.2: NOR
cell(norf301) {
  area : 32;
  cell_footprint : "nor3";
  pin(A) {
    direction : input;
    capacitance : 1;
  }
  pin(B) {
    direction : input;
    capacitance : 1;
  }
  pin(C) {
    direction : input;
    capacitance : 1;
  }
  pin(Z) {
    direction : output;
    function : "(A+B+C)'";
    timing() {
      intrinsic_rise : 1;
      intrinsic_fall : 1;
      rise_resistance : 0.2;
      fall_resistance : 0.2;
      slope_rise : 0.0;
      slope_fall : 0.0;
      related_pin : "A B C";
    }
  }
}

Figure B.3: NOR3

cell(norf401) {
  area : 40;
  cell_footprint : "nor4";
  pin(A) {
    direction : input;
    capacitance : 1;
  }
  pin(B) {
    direction : input;
    capacitance : 1;
  }
  pin(C) {
    direction : input;
    capacitance : 1;
  }
  pin(D) {
    direction : input;
    capacitance : 1;
  }
  pin(Z) {
    direction : output;
    function : "(A+B+C+D)'";
    timing() {
      intrinsic_rise : 1;
      intrinsic_fall : 1;
      rise_resistance : 0.2;
      fall_resistance : 0.2;
      slope_rise : 0.0;
      slope_fall : 0.0;
      related_pin : "A B C D";
    }
  }
}

Figure B.4: NOR4
cell(nanf201) {
    area : 24;
    cell_footprint : "nan2";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(B) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "(A B)';
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A B";
        }
    }
}

Figure B.5: NAND2

cell(nanf301) {
    area : 32;
    cell_footprint : "nan3";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(B) {
        direction : input;
        capacitance : 1;
    }
    pin(C) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "(A B C)';
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A B C";
        }
    }
}

Figure B.6: NAND3
cell(nanf401) {
  area : 40;
  cell_footprint : "nan4";
  pin(A) {
    direction : input;
    capacitance : 1;
  }
  pin(B) {
    direction : input;
    capacitance : 1;
  }
  pin(C) {
    direction : input;
    capacitance : 1;
  }
  pin(D) {
    direction : input;
    capacitance : 1;
  }
  pin(Z) {
    direction : output;
    function : "(A \land B \land C \land D)';
    timing() {
      intrinsic_rise : 1;
      intrinsic_fall : 1;
      rise_resistance : 0.2;
      fall_resistance : 0.2;
      slope_rise : 0.0;
      slope_fall : 0.0;
      related_pin : "A \land B \land C \land D";
    }
  }
}

<table>
<thead>
<tr>
<th>area</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell footprint</td>
<td>nanf401</td>
</tr>
<tr>
<td>function</td>
<td>A \land B \land C \land D</td>
</tr>
<tr>
<td>pin A,B,C,D</td>
<td>input</td>
</tr>
<tr>
<td>capacitance</td>
<td>1</td>
</tr>
<tr>
<td>intrinsic rise</td>
<td>1</td>
</tr>
<tr>
<td>intrinsic fall</td>
<td>1</td>
</tr>
<tr>
<td>rise resistance</td>
<td>0.2</td>
</tr>
<tr>
<td>fall resistance</td>
<td>0.2</td>
</tr>
<tr>
<td>slope rise</td>
<td>0.0</td>
</tr>
<tr>
<td>slope fall</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Figure B.7: NAND4
cell(orf201) {
    area : 32;
    cell_footprint : "or2";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(B) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "A+B";
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A B";
        }
    }
}

Figure B.8: OR2

cell(orf301) {
    area : 40;
    cell_footprint : "or3";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(B) {
        direction : input;
        capacitance : 1;
    }
    pin(C) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "(A+B+C)";
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A B C";
        }
    }
}

Figure B.9: OR3
cell (orf401) {
    area : 48;
    cell_footprint : "or4";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(B) {
        direction : input;
        capacitance : 1;
    }
    pin(C) {
        direction : input;
        capacitance : 1;
    }
    pin(D) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "A+B+C+D";
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A B C D";
        }
    }
}

Figure B.10: OR4

cell (andf201) {
    area : 32;
    cell_footprint : "and2";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(B) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "A B";
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A B";
        }
    }
}

Figure B.11: AND2
cell(andf401) {
  area : 48;
  cell_footprint : "and4";
  pin(A) {
    direction : input;
    capacitance : 1;
  }
  pin(B) {
    direction : input;
    capacitance : 1;
  }
  pin(C) {
    direction : input;
    capacitance : 1;
  }
  pin(D) {
    direction : input;
    capacitance : 1;
  }
  pin(Z) {
    direction : output;
    function : "A B C D";
    timing() {
      intrinsic_rise : 1;
      intrinsic_fall : 1;
      rise_resistance : 0.2;
      fall_resistance : 0.2;
      slope_rise : 0.0;
      slope_fall : 0.0;
      related_pin : "A B C D";
    }
  }
}

Figure B.13: AND4

cell(andf301) {
  area : 40;
  cell_footprint : "and3";
  pin(A) {
    direction : input;
    capacitance : 1;
  }
  pin(B) {
    direction : input;
    capacitance : 1;
  }
  pin(C) {
    direction : input;
    capacitance : 1;
  }
  pin(Z) {
    direction : output;
    function : "A B C";
    timing() {
      intrinsic_rise : 1;
      intrinsic_fall : 1;
      rise_resistance : 0.2;
      fall_resistance : 0.2;
      slope_rise : 0.0;
      slope_fall : 0.0;
      related_pin : "A B C";
    }
  }
}

Figure B.12: AND3
cell(aoi2201) {
    area : 40;
    cell_footprint : "aoi2201";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(B) {
        direction : input;
        capacitance : 1;
    }
    pin(C) {
        direction : input;
        capacitance : 1;
    }
    pin(D) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "(A B+C D)'";
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A B C D";
        }
    }
}

Figure B.14: AOI2201

cell(bl01) {
    area : 32;
    cell_footprint : "bl01";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(B) {
        direction : input;
        capacitance : 1;
    }
    pin(C) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "(A B+C D)'";
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A B C D";
        }
    }
}

Figure B.15: BL01
cell(oaif2201) {
  area : 40;
  cell_footprint : "oaif2201";
  pin(A) {  
    direction : input;
    capacitance : 1;
  }
  pin(B) {
    direction : input;
    capacitance : 1;
  }
  pin(C) {
    direction : input;
    capacitance : 1;
  }
  pin(D) {
    direction : input;
    capacitance : 1;
  }
  pin(Z) {  
    direction : output;
    function : "((A+B)(C+D))'";
    timing() {  
      intrinsic_rise : 1;
      intrinsic_fall : 1;
      rise_resistance : 0.2;
      fall_resistance : 0.2;
      slope_rise : 0.0;
      slope_fall : 0.0;
      related_pin : "A B C D";
    }  
  }  
}

cell(blf10) {
  area : 32;
  cell_footprint : "blf10";
  pin(A) {
    direction : input;
    capacitance : 1;
  }
  pin(B) {
    direction : input;
    capacitance : 1;
  }
  pin(C) {
    direction : input;
    capacitance : 1;
  }
  pin(Z) {  
    direction : output;
    function : "((A+B)(C+D))'";
    timing() {  
      intrinsic_rise : 1;
      intrinsic_fall : 1;
      rise_resistance : 0.2;
      fall_resistance : 0.2;
      slope_rise : 0.0;
      slope_fall : 0.0;
      related_pin : "A B C D";
    }  
  }  
}
cell (ao2201) {
    area : 56;
    cell_footprint : "ao2201";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(B) {
        direction : input;
        capacitance : 1;
    }
    pin(C) {
        direction : input;
        capacitance : 1;
    }
    pin(D) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "A \land B \lor C \land D";
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A \land B \land C \land D";
        }
    }
}

Figure B.18: AO2201
cell(xorf201) {
    area : 40;
    cell_footprint : "xor2";
    pin(A) {
        direction : input;
        capacitance : 1;
    }
    pin(B) {
        direction : input;
        capacitance : 1;
    }
    pin(Z) {
        direction : output;
        function : "A \oplus B";
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A B";
        }
    }
}

Figure B.19: XOR2

cell(xorif201) {
    area : 48;
    cell_footprint : "xori2";
    pin(A) {
        direction : input;
    }
    pin(B) {
        direction : input;
    }
    pin(Z) {
        direction : output;
        function : "A \oplus B";
        timing() {
            intrinsic_rise : 1;
            intrinsic_fall : 1;
            rise_resistance : 0.2;
            fall_resistance : 0.2;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "A B";
        }
    }
}

Figure B.20: NXOR2
cell(norf251) {
  area : 32;
  cell_footprint : "nor25";
  pin(A) {
    direction : input;
    capacitance : 1;
  }
  pin(B) {
    direction : input;
    capacitance : 1;
  }
  pin(Z) {
    direction : output;
    function : "A' B";
    timing() {
      intrinsic_rise : 1;
      intrinsic_fall : 1;
      rise_resistance : 0.2;
      fall_resistance : 0.2;
      slope_rise : 0.0;
      slope_fall : 0.0;
      related_pin : "A B";
    }
  }
}

Figure B.21: NOR25

cell(nanf251) {
  area : 32;
  cell_footprint : "nan25";
  pin(A) {
    direction : input;
    capacitance : 1;
  }
  pin(B) {
    direction : input;
    capacitance : 1;
  }
  pin(Z) {
    direction : output;
    function : "A' B";
    timing() {
      intrinsic_rise : 1;
      intrinsic_fall : 1;
      rise_resistance : 0.2;
      fall_resistance : 0.2;
      slope_rise : 0.0;
      slope_fall : 0.0;
      related_pin : "A B";
    }
  }
}

Figure B.22: NAND25
<table>
<thead>
<tr>
<th>area</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell_footprint</td>
<td>mux201</td>
</tr>
<tr>
<td>function</td>
<td>((A \land S) \lor (B \land S))</td>
</tr>
<tr>
<td>timing()</td>
<td></td>
</tr>
<tr>
<td>intrinsic_rise</td>
<td>1</td>
</tr>
<tr>
<td>intrinsic_fall</td>
<td>1</td>
</tr>
<tr>
<td>rise_resistance</td>
<td>0.2</td>
</tr>
<tr>
<td>fall_resistance</td>
<td>0.2</td>
</tr>
<tr>
<td>slope_rise</td>
<td>0.0</td>
</tr>
<tr>
<td>slope_fall</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Figure B.23: MUX201
Biography

Szymon Biegański was born in 1975 in Łódź, Poland.

After receiving primary and secondary education in Łódź, he enrolled in 1995 at the Faculty of Electrical, Electronic, Computer and Control Engineering of the Technical University of Łódź in the Department of Microelectronics and Computer Science. The academic year 1999/2000, the last year of his Master studies, he spent at the University of Ghent in Belgium, where he worked on his master thesis, and took part in a project in the framework of Erasmus Student Exchange Programme. In 2000, he graduated from Łódź University of Technology in the specialty Microelectronics and Computer Science.

In year 2001, he joined the faculty of Electronics of Technische Universiteit Eindhoven in Eindhoven, the Netherlands as a Ph.D. candidate. He performed there the research on general decomposition and encoding for Complex Gates Technology Library implementation under the supervision of prof.ir. Mario Stevens, dr.ir. Lech Jóźwiak and prof.dr.ir. R.H.J.M. Otten. The results of this research are presented in this thesis.