Automatic Synthesis and Optimisation of Asynchronous Data Paths Using Partial Acknowledgement

Yu Zhou

Technical Report Series

NCL-EECE-MSD-TR-2008-130

March 2008
Contact:

yu.zhou@ncl.ac.uk

EPSRC supports this work via GR/S81421 (SCREEN)

NCL-EECE-MSD-TR-2008-130
Copyright © 2008 University of Newcastle upon Tyne

School of Electrical, Electronic & Computer Engineering,
Merz Court,
University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/
University of Newcastle upon Tyne
School of Electrical, Electronic and Computer Engineering

Automatic Synthesis and Optimisation of Asynchronous Data Paths Using Partial Acknowledgement

by

Yu Zhou

A thesis submitted for the degree of Doctor of Philosophy (Ph.D) at
Newcastle University

March 2008
Contents

List of Figures x
List of Tables xiii
List of Algorithms xv
Acknowledgements xvi
Glossary of Abbreviations xvii
Abstract xviii

1 Introduction 1

2 Background 14
   2.1 Partitioning of control and data paths . . . . . . . . . . . 15
   2.2 Terminologies and taxonomies of asynchronous circuits . . . 19
      2.2.1 Operation modes . . . . . . . . . . . . . . . . . . . . 19
      2.2.2 Classification of asynchronous circuits . . . . . . . . 21
2.3 Asynchronous controllers and their synthesis ............ 21
2.4 Asynchronous data paths ........................................ 23
   2.4.1 Dual-Rail (DR) Encoding and asynchronous data path
         protocols .................................................. 23
   2.4.2 Martin’s evaluation programme .............................. 24
   2.4.3 Library design for asynchronous data paths .............. 26
   2.4.4 NCL synthesis flows ..................................... 27
       2.4.4.1 Threshold logic ...................................... 29
       2.4.4.2 NCL data path synthesis ........................... 30
       2.4.4.3 Variable completeness, orphans and orphan
              isolation .............................................. 34
2.5 Conclusion ....................................................... 35

3 Partial acknowledgement: its definitions, properties and
   characteristics ..................................................... 36

   3.1 Definitions of PA .............................................. 37
   3.2 Properties of PA ............................................... 38
   3.3 Levels of PA distribution .................................... 39
       3.3.1 Category 1 .............................................. 40
       3.3.2 Category 2 .............................................. 40
       3.3.3 Category 3 .............................................. 40
   3.4 Asynchronous data paths with the PA requirements ........ 42
3.4.1 Validity checking .............................. 43
3.4.2 Timing relationships ......................... 45
  3.4.2.1 Gate-orphans .......................... 45
  3.4.2.2 Wire-orphans and their timing requirements . 45
3.5 Notes .............................................. 46
3.6 Summary .......................................... 47

4 Design of dual-rail functional modules 48
  4.1 DRFMs and naming methods .................... 49
  4.2 Synthesis procedures of DRFMs ................ 50
    4.2.1 Brief introduction ........................ 50
    4.2.2 Synthesis of the PDN ...................... 51
      4.2.2.1 Synthesis of the PDNs .................. 51
      4.2.2.2 PA by PDNs ............................... 54
    4.2.3 Synthesis of PUN ............................ 56
    4.2.4 Static short currents and dangling outputs .... 57
  4.3 Propagation delays of DRFMs ................... 58
    4.3.1 skewed or unskewed structure ............... 59
    4.3.2 LE for PD computation ........................ 59
  4.4 Notes .............................................. 63
  4.5 Summary .......................................... 64

5 Performance analysis of asynchronous data paths 65
  5.1 Controlling and non-controlling values .......... 66
5.2 Generic sub-variable expressions .......................... 67
5.3 Dominating inputs of DRFMs ............................... 68
  5.3.1 Dominating inputs without PA ........................... 68
  5.3.2 Dominating inputs with PA ............................. 69
  5.3.3 Dominating inputs and early propagation ............... 71
5.4 Input dependent timing simulations ........................ 71
5.5 Min-max static timing analysis ............................. 75
  5.5.1 Arrival times in min STA ............................... 76
  5.5.2 Arrival times in max STA ............................... 78
  5.5.3 Min and max critical paths ............................. 82
  5.5.4 False path problems ................................. 82
5.6 Summary .................................................. 85

6 Synthesis and optimisation of asynchronous data paths
    using partial acknowledgement .............................. 86
  6.1 Problem description ...................................... 87
    6.1.1 Formal description of the problem by MP ............... 88
      6.1.1.1 The variables .................................... 88
      6.1.1.2 The constraints .................................. 89
      6.1.1.3 The objectives .................................. 90
    6.1.2 A formulation example ............................... 90
  6.2 Practical considerations of solving the problem ........... 94
6.2.1 Global optimisation versus local search heuristics ... 94
6.2.2 Factors influencing the computation time ............... 95
6.2.3 Leverage by user-specified knowledge .................. 98
  6.2.3.1 Nodes with one direct fan-out ..................... 98
  6.2.3.2 Nodes with one direct fan-in ...................... 99
  6.2.3.3 Binding the rising and falling phase acknowledgement .................. 99
  6.2.3.4 Constraint relaxation .............................. 99
  6.2.3.5 Nodes with branching priorities .................. 100
6.3 Methods of solving the problem ......................... 100
  6.3.1 Time limited assorted heuristics (pseudo-global optimisation) .................. 101
  6.3.2 Problem decomposition .............................. 103
    6.3.2.1 Level-wise optimisation .................... 104
    6.3.2.2 Farthest acknowledgement ....................... 107
6.4 Design flow and experimental results ................... 108
  6.4.1 Synthesis and optimisation flow .................... 108
  6.4.2 Experimental results ............................... 111
6.5 Summary ................................................. 115

7 Conclusion and future work .............................. 117
  7.1 Summary of the problem and its solution process .... 117
  7.2 Future work ........................................ 122
7.2.1 non-SADT synthesis techniques .................. 122
7.2.2 Synthesis with mixed encoding .................. 123

A  DRFM libraries ............................................ 125

B  An example session of designing asynchronous data paths 129
   B.1 Pre-optimisation mapping and analysis .................. 130
      B.1.1 Technology mapping .................................. 130
      B.1.2 Connectivity and level analysis ...................... 131
      B.1.3 Initial PA mapping and analysis ...................... 134
   B.2 Synthesis and optimisation .............................. 135
   B.3 Post-optimisation PA mapping and analysis .............. 139

Bibliography ................................................. 141
List of Figures

2.1 A synchronous circuit made up of clock (clk), registers and
    data paths ........................................ 16
2.2 Micropipeline ........................................ 17
2.3 A de-synchronised circuit with local handshaking protocols . 18
2.4 Signal transition graph of micropipeline ...................... 22
2.5 Implementation of NAND function block using DL (a) and
    RDL (b) ............................................. 28
2.6 DIMS implementation ................................... 28
2.7 A general M-of-N threshold gate symbol and two 2NCL operators 30
2.8 Full adder implementation by product and sum partitioning . 32
2.9 Adder implementation with output substitution ................ 33

3.1 PA distribution: original SR netlist (a), NCL-D implementa-
    tion (b) and Category 2 implementation (c) .................. 41
3.2 (Minimal) direct fanout sets ................................ 44
3.3 Timing relationship in partially acknowledged data paths ... 46
4.1 The DRFM prototype (a) and an implementation of a MUX2_a↑c (b) ................................. 51
4.2 Schematic of AND2_a↑b↑(a) and AND2_a↑b↓(b) ......................... 57
4.3 Schematic of AND2_a→b− (a), AND2_a* b− (b), and
AND2_a* b* (c), where the input logical efforts, output par-
asitics, and transistor widths are annotated ..................... 61
4.4 Schematic of OR2_a* b↑(a) and OR2_a* b↓(b), where the
input logical efforts, output parasitics, and transistor widths
are annotated .............................................................. 62

5.1 Part of a data path under timing simulation ......................... 72
5.2 Data path example for min-max timing analysis .................. 73
5.3 False path example when using min STA ........................... 84

6.1 schematic of ISCAS benchmark C17 ................................. 91
6.2 Converging and forking networks ................................. 96
6.3 Running times of the problem with different influencing fac-
tors. The results is from BARON’s GO engine run on a win-
dows machine with a 1.67GHz processor and 512MB RAM.
The forking networks have a direct fan-in number of 2. .... 97

6.4 Illustration of the level-wise optimization .......................... 106
6.5 The synthesis and optimisation flow ............................. 109

A.1 pa_min.genlib .............................................. 127
A.2 pa_max.genlib .............................................. 128
B.1 script.tm .................................................. 131
B.2 C17_2.blif .................................................. 132
B.3 script.netlist .............................................. 133
B.4 Schematic of ISCAS benchmark C17 ................. 133
B.5 C17.l .......................................................... 134
B.6 script.pa_init_analysis .................................... 135
B.7 Output from running script.pa_init_analysis ........ 136
B.8 C17_a_G ....................................................... 138
B.9 C17_a_results ............................................... 138
B.10 script.pa_fin_analysis .................................. 140
B.11 Output from running script.pa_fin_analysis ........ 140
List of Tables

5.1 Dominating inputs and the determination criteria of AND2\_a—b— and AND2\_a * b— ................................. 71

6.1 The functions implementing the major steps of the design flow 111

6.2 Benchmark circuits optimised with the time limited assorted heuristics methods for area minimisation and joint area-performance optimisation. The table shows the problem size, running times, area, delays, and the area/delay ratios between the results using the proposed algorithms and those of the fully acknowledged circuits. ................................. 113

6.3 Benchmark circuits optimised with the problem decomposition including the level-wise optimisation and the farthest acknowledgment. The table shows the problem size, running times, area, delays, and the area/delay ratios between the results using the proposed algorithms and those of the fully acknowledged circuits. ................................. 114
B.1 C17.m .................................................. 133
List of Algorithms

1. joint area-performance optimisation using time limited assorted heuristics .................................................. 103
2. An algorithm to level the nodes in a boolean network ................. 104
3. Level-wise optimisation .................................................. 105
4. algorithm for the farthest acknowledgement .......................... 107
Acknowledgements

I would like to express my gratitude to my supervisor, Alex Yakovlev, for his tremendous help, kind encouragement and support during my PhD research. From him, I learned valuable knowledge and experiences on asynchronous circuit design and its automation. More importantly, he taught me the right attitudes towards research and work.

I would like to thank Agnes Madalinski, Sohini Dasgupta, Robin Emery, Danil Sokolov and Andrey Mokhov for their kind help with my programming. I would also like to thank Julian Murphy and Robin Emery for their help with my EDA tool usage. My special thanks go to Ping Wang for her introduction of the basic ideas of asynchrony before my PhD study.

I am grateful to my parents for their constant encouragement and support during my study in Newcastle. I am also thankful to my friends who made my life in UK interesting.

In addition, I would like to thank Robin Emery for reading through my PhD thesis and helping me with my English.

Finally, I would like to acknowledge that this work was supported by the ORS Awards Scheme grant and the EPSRC grant GR/S81421 (SCREEN).
## Glossary of Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B(U)CP</td>
<td>Binate (Unate) Covering Problem</td>
</tr>
<tr>
<td>BN</td>
<td>Boolean Network</td>
</tr>
<tr>
<td>CSC</td>
<td>Complete State Coding</td>
</tr>
<tr>
<td>CH(S)P</td>
<td>Communicating Hardware (Sequential) Process</td>
</tr>
<tr>
<td>DRFM</td>
<td>Dual Rail Functional Module</td>
</tr>
<tr>
<td>MINLP</td>
<td>Mixed Integer Non-Linear Programming</td>
</tr>
<tr>
<td>MP</td>
<td>Mathematical Programming</td>
</tr>
<tr>
<td>NLDM</td>
<td>Non-Linear Delay Model</td>
</tr>
<tr>
<td>N(RZ)</td>
<td>(Non-) Return to Zero</td>
</tr>
<tr>
<td>OR</td>
<td>Operational Research</td>
</tr>
<tr>
<td>PA</td>
<td>Partial Acknowledgement</td>
</tr>
<tr>
<td>P(W)D</td>
<td>Propagation (Wire) delay</td>
</tr>
<tr>
<td>PD(U)N</td>
<td>Pull Down (Up) Network</td>
</tr>
<tr>
<td>(Q)DI</td>
<td>(Quasi-) Delay Insensitive</td>
</tr>
<tr>
<td>RTL</td>
<td>Register-Transfer Level</td>
</tr>
<tr>
<td>SADT</td>
<td>Synchronous-Asynchronous Direct Translation</td>
</tr>
<tr>
<td>SG</td>
<td>State Graph</td>
</tr>
<tr>
<td>SI</td>
<td>Speed Independent</td>
</tr>
<tr>
<td>SPDM</td>
<td>Scalable Polynomial Delay Model</td>
</tr>
<tr>
<td>STG</td>
<td>Signal Transition Graph</td>
</tr>
</tbody>
</table>
Abstract

Parametric variations are one of the biggest implementation issues for deep-submicron technologies. Asynchronous designs apply cause-sequential order relation for circuit timing and are therefore more robust against variations than synchronous counterparts relying on global clocks. However, large overheads from the causal order relation and lack of tool support limit the acceptance of asynchrony.

This thesis discusses the design, synthesis and optimisation of asynchronous data paths that are speed independent. In previous design practices represented by the Null Convention Logic (NCL) flow, each signal is fully acknowledged in that the causal relation exist between a signal and all its direct fan-outs in each data cycle. As a result, the data path is bulky and slow.

Full acknowledgement is relaxed in this thesis to enable smaller and faster implementations. Partial acknowledgement (PA) is presented as the fundamental concept describing the timing relation in a data path and can be distributed at various levels. A library of modules are designed with a wide spectrum of input PA patterns. Rules regulating a data path’s data flows are discussed, and the input-dependent simulation and min-max static analysis are introduced for performance study. Further, PA is associated with the cost of area and (or) performance by the specification of design objectives. Automatic synthesis and optimisation of asynchronous data paths for different objectives is formulated with mathematical programming and solved by
different heuristic algorithms.

Results from running the algorithms on ISCAS benchmarks show the effectiveness of the approaches: the area is reduced by an average factor of 1.26, the min critical path delay by 1.60 and the max critical path delay by 1.27, compared with the fully acknowledged implementations. Furthermore, solution times on the order of tens of seconds for the largest benchmark circuit make the approaches appropriate for large scale applications.
Chapter 1

Introduction

As the feature size of semiconductor products scales down along the technology nodes, it is more and more difficult to control the variations caused by either fabrication process or environment. Variations adversely affect the geometry, performance and power of the products, and have become one of the biggest issues in the semiconductor industry.

Designing circuits with the synchronous regime is naturally more prone to the influence of the uncertainties during fabrication, i.e., variations. This is because of its open-loop and feed-forward control networks [62]. With this type of structure, margins must be predicted and left during the whole design flow by considering possible corners with various conditions of voltage, temperature, etc. These margins significantly increase the clock cycle time in addition to the inherent sequencing overheads of synchronous designs. Statistical timing analysis [1] improves the predictions; however, design tools
utilising statistical process data are lacking as well as the post-manufacturing measurements [62].

Different from synchronous design approaches, asynchronous approaches introduce feed-back control in terms of handshaking for data sequencing [56] in a cause-sequential order. In addition, the delays of combinational logic in asynchronous circuits are measured rather than estimated. All these mechanisms make asynchronous designs more robust to timing variations when compared with their synchronous counterparts. Furthermore, asynchronous circuits are reported to have advantages of low electromagnetic emission [24] and power consumption. The latter one is supported by the fine level controls over activation on demands [62] and reliable operations (and faster speeds) at very low voltages [21].

In spite of all these merits, asynchronous circuits are regarded as hard to design, first because designers are used to the mentality of clock sequencing and second because asynchronous tools and flows are lacking when compared with synchronous ones. These two factors reinforce each other to impede the acceptance of asynchrony; still, there have been significant developments in design automation of asynchronous circuits in the past 10 to 20 years where various flows and tools have appeared for large and real-life design problems. These flows are reviewed in [62] as a counteraction to the views of deficiencies in supporting asynchrony.

One paradigm of these flows is called Synchronous-Asynchronous Direct Translation (SADT), where synchronous gate level net-lists synthesised
from high level specifications, such as Register-Transfer Level (RTL), are translated into asynchronous implementations by applying certain conversion techniques. This is different from compiling circuits from higher level asynchronous specifications (compared with RTL) such as Communicating Sequential Process (CSP) like languages [31] [4]. Two representative flows of SADT are *de-synchronisation* [9] and Null Conventional Logic (NCL) [34]. Both flows support control and data path synthesis but have some differences in data path design - the former uses bundled data scheme while the latter Delay Insensitive (DI) encoding.

SADT techniques do not have to be mastered by designers with a synchronous mindset, and therefore overcome some obstacles to the acceptance of asynchrony. Design legacy is maintained with SADT and thus design costs, the greatest threat to the continuation of the semiconductor road map [20], are reduced. Further, SADT provides a clean separation between functionality and timing [62], and is similar to the roles played by RTL in synchronous designs.

This thesis discusses the design, synthesis and optimisation of asynchronous data paths by SADT. When compared with synchronous data paths, asynchronous paths need more care on timing issues and involve greater design complexity. This is especially true when asynchronous data paths are designed by DI encoding such as Dual Rail (DR) encoding that will be used in this thesis. DR code is a systematic code where there exists a direct correspondence between a binary bit and the DR encoded variable for that bit.
For a data path with systematic encoding, it is easier to detect the completion of computation. Compared with bundled data approaches, data paths with DR encoding have larger overheads in area and power. For example, two wires are used to transmit one bit of binary information, which doubles the area at least. In addition, DR encoding has one signal transition for every transmitted bit, and therefore two times that of the dynamic power consumption of its Single Rail (SR) counterpart at an average transition activity of 50%. However, data paths with DR encoding demonstrate “true” delays based on measurements rather than predictions, and therefore have a higher level of timing robustness. The protocol used in this thesis is known as the “Return to Zero” (RZ) protocol where two cycles are required for every computation of a batch of data: one for the real value computation and the other for resetting. When compared with the data paths designed by another protocol [10] (Non Return to Zero (NRZ)), the RZ data paths are easier to design but have extra null cycles for resetting.

Applying DR encoding to asynchronous circuits was first discussed by Muller [37] and has been continuously studied since then, despite its overheads discussed above. The discussion of background of asynchronous data path synthesis using DR encoding and RZ protocol is split into three aspects: library design, formal methods and synthesis techniques. Design of DR modules can be based on switching CMOS techniques such as Delay Insensitive Minterm Synthesis (DIMS) [57], direct logic [70] and reduced direct logic [39], or NCL operators of threshold logic [50].
Formal methods to design asynchronous data paths using DR codes are discussed in [32]. In this method, the behaviour of a data path is characterised by a CSP like programme satisfying Seitz's strong indication [48], which means any output must wait for all inputs to become valid code words before beginning its evaluation. This global waiting is gradually relaxed through a series of "legal" transformations, until in the end a concurrent composition of programmes are derived which evaluate the functions of the decomposed cells in the data path. The final data path demonstrates a weak indication: an output can begin its evaluation as long as its supportive inputs are valid code words; still, evaluation of the last output waits for all inputs to become valid. However, no techniques are offered as how to decompose a data path and apply these programme transformations.

Automatic synthesis techniques are widely discussed in the NCL design flows in two directions. Techniques in the first direction belong to SADT: the synthesis begins with a structural SR data path netlist and maps each gate in the netlist to a functional module by DR encoding. Mapping procedures in SADT are usually "correct in construction" in satisfying certain hazard-free requirements- such as the "gate-orphan free" in NCL flows- to ensure the timing robustness. With SADT data path synthesis, all the tools developed for synchronous combinational logic synthesis can be reused, and the mapping procedure is simple and straightforward. In contrast, the second direction explores more general synthesis techniques that are not restricted to topology maintaining. In theory, it can start with any behavioural for-
mulation of a data path and generate its asynchronous implementation. For example, one NCL flow begins with multi-value functions and uses sum or product partitioning and boolean substitution techniques for synthesis. As another example, algebraic division techniques are used in [64] to generate the strongly indicating combinational logic. Synthesis techniques in the second direction are intuitively more powerful but complex. It is difficult to ensure that a synthesis step does not introduce hazards without structural manipulations as in SADT; indeed, this is what makes asynchronous data path synthesis distinctively different from traditional 2-level or multi-level combinational logic synthesis techniques [11]. For example, Don’t Care (DC) minimisations should be used with care as simplification with unobservable DCs may lead to hazards.

Acknowledgement is an important concept in the design of asynchronous circuits to avoid hazards and ensure timing robustness. Informally speaking, a transition a is acknowledged by another transition b if a always precedes b in any possible signal transition sequence of a circuit. Acknowledgement introduces the cause-sequential order to a circuit as well as design complexities.

NCL-D [29], one representative approach of designing asynchronous data paths using SADT, maps a gate in a SR netlist with DIMS like DR implementations. NCL-D is very robust against variations: forks fanning into different DIMS-like modules are allowed to have arbitrary delays whereas the correct operation is not violated. However, it carries high penalties for area and
performance because each variable transition is acknowledged by the output transitions of all the functional modules it fans out to in both valid data and null cycles.

In another SADT approach, NCL-X [25] has two parts: computational circuit mapped from a SR netlist and an extra Completion Detection (CD) circuitry. Input acknowledgement is relaxed and only fulfilled by the CD circuitry part. Therefore, NCL-X implementations have reduced areas and improved performance in the computational part. However this approach has an extra burden for routing the interconnections used in CD network.

The research in this thesis is motivated by reducing the costs of a robust asynchronous implementation such as NCL-D but not introducing an extra CD network. To achieve this goal, the acknowledgement relations existed in NCL-D like implementations must be relaxed. For example, acknowledgement can be relaxed to exist between a transition and some of its fanout transitions, or, during one particular data cycle.

However, the level of robustness (e.g. the gate-orphan free condition) is not to be sacrificed after relaxation is applied to an asynchronous data path. Relaxation does not eliminate the acknowledgement relation but introduces some weaker forms of indication relations such as the Partial Acknowledgement (PA) defined in Chapter 3 of this thesis. Informally, a DR variable is partially acknowledged in a particular cycle if its transition in that cycle, regardless of the data values involved, causes a variable transition in the same cycle. With the concept of PA, we can say that data paths synthesised by
NCL-D are fully acknowledged: a variable is partially acknowledged by all its direct fan-outs in both valid and null cycles. By NCL-X, a variable is partially acknowledged by the Muller-C element in the CD circuitry it fans out to.

Extents to which the above relaxation can be applied without violation of timing robustness is explored in this thesis. The minimum requirement that must be satisfied during any relaxation procedure is proposed: each input or internal variable in a data path must be partially acknowledged by at least one of its minimal direct fanout PA sets (defined in Chapter 3). With this requirement, the timing robustness of asynchronous data paths can be maintained.

Flexible forms of PA exist for a relaxed robust asynchronous data path. For example, a variable can be acknowledged by one of its fan-outs in the valid data cycle and another in null cycle. An even weaker form of PA can be found in cases where a variable is partially acknowledge by a group of variables but none of the sub-set of that group (thus this group is a minimal PA set). Distribution of PA at various levels is discussed in detail in Chapter 3.

In this thesis, DR functional modules mapped from gates in a SR netlist are chosen as the cells in Martin's programmes. With this idea, the output of a DR module can be responsible for the indication of some inputs in the valid data cycle and others in the null cycle. Implementation of these functional modules is not difficult using CMOS techniques, as their pull-up
and pull-down networks are well separated for indication during the null and valid data cycle, respectively. The design of these functional modules is presented in Chapter 4 where design procedures, implementation issues, indication verification and delay characterisations are discussed in detail.

PA has a significant influence on the performance of asynchronous data paths. Chapter 5 discusses these influences and how to analyse the performance in asynchronous data paths. An input-dependent timing simulation technique and a static timing analysis (STA) for min-max critical path delays are discussed.

Now that both concepts and library elements supporting the concepts are available, we consider automatic synthesis flows with optimisation objectives. The space for optimisation comes from the flexibility in choosing PA of a variable among its direct outputs, and the leverage on the area and performance associated with these choices. The flows can be formulated by three types of optimisation problems, given the level of how PA can be distributed and the functional modules supporting this distribution.

The first type of formulation is Unate Covering Problem (UCP) [11]. In this flow, only two types of library elements exist: those that partially acknowledge all inputs in both cycles, like DIMS, and those that acknowledge no input, such as the ones used in the computational part of NCL-X. In SADT, a gate can be mapped to only one of these two types. UCP tries to find the most economic way to implement a data path such that the PA of all input and internal variables is “covered” by those DIMS style modules [73] [22].
Solving UCP is NP-hard but beneficial, for this formulation usually translates to a sparse covering matrix that can be efficiently solved using a UCP solver such as mincov in SIS [49].

By expanding the library to include functional modules that can partially acknowledge any subset of inputs, the flow can be formulated as a Binate Covering Problem (BCP) [73]. In this case, PA is still bound for the valid and null cycles as in UCP: if a variable is partially acknowledge by another variable in the valid data (null) cycle, it must be partially acknowledged by the same variable in the null (valid data) cycle. This formulation is binate because constraints introduced by the expanded library cannot be formulated as “yes or no” ones in UCP to map gates to their DIMS implementations.

Finally, when PA of a variable in the valid data and null cycles can be separated and there are modules in the library supporting all possible patterns of how inputs can be acknowledged, the flow can be formulated as Mixed Integer Non-Linear Programming (MINLP), which will be discussed in Chapter 6. In this formulation, PA between each two directly connected variables in the data path is encoded by three-dimensional 0-1 (binary) variables. MINLP tries to find value assignments to these variables to achieve the optimisation of objectives such as circuit area and performance while satisfying the constraints which maintain the robustness level of an implementation. This formulation provides the first unified framework for synthesizing and optimizing asynchronous data paths where robustness is introduced in a cost (area and/or performance) aware manner due to the tuning mechanism of PA at a
very fine level. Previous formulations, such as UCP and BCP, can be viewed as more general cases of this formulation.

These three formulations have different trade-offs in optimisation space and computation complexity: MINLP is finer-grained in the problem formulation level compared with UCP/BCP but more difficult to solve and design. In practice, it is the designers’ decision to choose the appropriate level of formulation. To help solve the problem formulated by MINLP, we introduce different approaches and algorithms in Chapter 6, combining solver configuration, leverage by user specified knowledges and problem decomposition. Some of these algorithms demonstrate run times on the order of tens of seconds for the largest benchmark circuit but with only minor optimisation degradation.

However, even the flow with the finest level of formulation does not fully utilise the distribution of PA supported by its definitions. In particular, a variable can be partially acknowledged by a group of functional modules but not by any one of them. Simpler implementations in terms of area and performance exist in these cases and are discussed in Chapter 3. These scenarios will be involved in future flows in a systematic way, possibly by introducing some substitution techniques which are not based on direct translations.

Experiments on industry level benchmarks are performed to test the reduction of costs in NCL-D like implementations by our flow when formulated as MINLP. Experimental results show that the area is reduced by an average factor of 1.26, the minimum critical path delay by 1.60 and the maximum
critical path delay by 1.27. In addition, solution times on the order of tens of seconds for the largest benchmark circuit make these approaches appropriate for large scale applications.

To summarise, the thesis makes the following main contributions:

(1) PA is defined as the underlying concept to capture the relationship between data path variables. Requirements of PA can be satisfied with different implementations and this flexibility is the basis to optimise an asynchronous data path, during which the timing robustness can be introduced in a cost aware manner.

(2) A design procedure is introduced to synthesise general DR functional modules with various PA patterns of the inputs. A library is designed using this procedure as the building elements to synthesise asynchronous data paths.

(3) Approaches to analyse the performance of data paths satisfying PA requirements are presented. Dominating inputs and their determination criteria are discussed. Based on this concept, delay simulation of a data path can be performed. Relation between PA and early evaluation is discussed. A static approach is proposed to analyse the lower and upper bounds of a data path’s delays.

(4) The problem to synthesise and optimise asynchronous data paths is formulated by MINLP in a design flow. To the best of our knowledge, this is the first attempt towards a unified framework to solve this problem by SADT with a very fine-grained level of tuning.
(5) Various heuristic algorithms are implemented to solve the MINLP problem with different trade-offs between run time and optimisation improvement. These heuristics can solve large scale problems when formulated as MINLP in linear time but with only minor optimisation degradation.
Chapter 2

Background

In this chapter, the background knowledges used in this thesis are discussed. The theme of this thesis is synthesis and optimisation of asynchronous data paths using delay insensitive codes. Still, it would be instructive to begin with the control/data paths partitioning and introduce briefly the design of asynchronous controllers. This is because asynchronous data paths usually involve some timing sequences for “correct” operation. Furthermore, different protocols such as 4-phase, 2-phase, weak and strong indication are only meaningful when the environment of the data paths, the controller and interfaces, is well understood.

Timing issues in data path synthesis have their roots in the concepts and general theories on asynchronous circuits like hazard free conditions and speed-independence. Therefore, we also introduce the terminologies and classification of asynchronous circuits in the literature.
Different aspects of asynchronous data path design using delay insensitive
codes are discussed in detail. They include the library design, formal methods
and flows represented by Null Convention Logic (NCL).

2.1 Partitioning of control and data paths

Both synchronous and asynchronous circuits can be generally partitioned into
control and data paths. Data paths compute the values of arithmetic or logic
functions whereas control paths are supposed to provide correct sequencing
of theses values. Figure 2.1 shows a simple synchronous circuit where a global
clock (clk) controls latching of data paths 1 and 2's outputs by registers 1 and
2, respectively. The clock period must be made larger than the sum of the
critical path cycle time (5 ns in this example) and the setup time. Meanwhile
the hold time is also to be satisfied for stable data sampling. Setup and hold
time are referred to as the sequencing burden [18] as they can eat into clock
cycles that are valuable in high performance applications.

Asynchronous circuits differ from synchronous ones in that the sequencing
are due to local handshaking signals, requests (r) and acknowledgements
(a), rather than global clocks. Protocols for the handshaking signals must
satisfy absence of races and no data loss [62], which can be viewed as the
counterparts of the hold and setup time constraints of synchronous circuits,
respectively.

Asynchronous pipeline controllers, such as those used in the Muller pipeline [37]
and *micropipeline* [61], are famous examples of implementing these handshaking protocols. Asynchronous controllers in their more general forms are discussed in section 2.3. Figure 2.2 shows a three stage *micropipeline*, where data are latched in event based registers (R1-R3) on both rising and falling edges of local clocks (*acknowledgements*, in particular, at different pipeline stages). This protocol is called *two-phase* (or *Non-Return-to-Zero*, NRZ) protocol where a cycle of handshaking activity involves one transition on *request* and the other on *acknowledgement*. Alternatively, *four-phase* (or *Return-to-Zero*, RZ) protocol exists, where a complete handshaking cycle comprises of 2 *two-phase* protocols and has a transition sequence such as \( r+\rightarrow a+\rightarrow r-\rightarrow a- \).

In *micropipeline*, new data can only be read into a register after the current data in this register has been used to evaluate the data path it feeds and the outputs of this data path have been latched in the next stage registers. This is due to the Muller-C elements [37], which synchronise *request* at a stage with *acknowledgement* from the next stage in its complemented
form. As a result, race is avoided. On the other hand, delay elements in request paths matched with the computation times of data paths ensure that the data to be sampled are stable when requests are due. Therefore, the setup time requirement is satisfied. This is the bundled data scheme which has been widely used in asynchronous designs, represented by the Amulet processor series [13] [15] [14]. Alternatively delay insensitive encoding can be applied such that completion of computation can be integrated in the data path without delay matching.

A more recent technique, de-synchronisation [9], tries to convert a synchronous circuit to its asynchronous counterpart by introducing a handshaking protocol that preserves the liveness and equivalent flow to that of its synchronous counterpart. In a de-synchronised circuit, data are driven asynchronously, pretty well in the manner of micropipeline; however, the external behaviour of the circuit, when observed in terms of value sequences through registers on the boundaries of data path clouds, remains the same before and after de-synchronisation is applied.
Figure 2.3: A de-synchronised circuit with local handshaking protocols

Figure 2.3 shows how the synchronous circuit in Figure 2.1 can be mutated into its asynchronous counterpart by de-synchronisation. It serves only for a schematic explanation where liveliness and flow equivalence may not be satisfied. The registers in 2.1 are replaced by level based latches and the clock by local handshaking controller networks. In this circuit, a latch begins sampling the output of a combinational data path only when all the request signals bundled with the inputs of this data path arrive. Similarly, the acknowledgement signal to a latch controller waits all the acknowledgements from the fan-outs of this latch. Again, these synchronisations are implemented here by the Muller-C elements.

Partitioning of a circuit into control and data paths brings a clean sep-
eration between functionality and timing [62]. It is widely accepted that the *Register-Transfer-Level* (RTL) model plays an important role in synchronous *Electronic Design Automation* (EDA): timing issues of the sequential behaviours between registers are well separated from the synthesis and optimisation of combinational functions. For a micro-pipelined circuit in Figure 2.2 and 2.3, handshaking protocols play a similar role to that of the RTL model in synchronous circuit: when a latch is receiving (sending) data, the latches in its immediate previous and following stages are sending (receiving) data, which is analogous to the two phase clocking scheme in synchronous circuits [48].

### 2.2 Terminologies and taxonomies of asynchronous circuits

#### 2.2.1 Operation modes

Different operation modes exist for asynchronous circuits in terms of their assumptions on the circuit-environment interaction. The most influential operation modes are reviewed in the following [56] [17].

1. Fundamental mode Huffman circuits [19] [67]

Circuits under this mode have Finite State Machine (FSM) like specifications and implementations but with additional requirements. Only one input
to the combinational logic is allowed to change at a time to ease hazard elimination. In addition, present states of the circuit must be stable before the combinational logic settles with this change (and therefore worst-case delay is applied, similar to synchronous circuits), this is ensured by adding delays in the feedback paths. Finally, the input changes occur only until the the entire circuit settles (the fundamental assumption).

2. Burst mode [40][72]

In this mode, single input change is extended to multiple changes in a burst, and fundamental assumption still applies to input transitions belonging to different bursts. Furthermore, all bursts are prime in that no burst can be included in another one. More complex hazards are introduced with burst modes; however they are eliminated with the introduced local clocks.

Correct operation of the above two modes assume bounded gate and wire delays. This assumption has problems such as additive skews and difficult delay fault testing, in addition to the worst case behaviour.

3. Input-output mode [38, 37]

In this mode, the fundamental assumption is removed and it is possible to change an input before the circuit has stabilised in response to the previous input change. Circuits working with input-output mode have unbounded delay assumptions for gates and some wires (positive, unknown and unbounded from above).
2.2.2 Classification of asynchronous circuits

Depending on the levels of delay assumption for asynchronous circuits to work “correctly”, a circuit can be Speed Independent (SI), Delay Insensitive (DI), or Quasi Delay Insensitive (QDI). DI circuits operate “correctly” with gates and wires having unbounded delays. It is extremely robust but has a limited application, as it was shown in [30] that only circuits made up of inverters and Muller-C elements, such as a micropipeline controller, can be DI. When circuits work correctly with the unbounded delay assumptions for all gates but only some wires, this type of asynchronous circuits are known as SI or QDI [56].

In SI or QDI circuits, both critical and non-critical forks exist. Non-critical forks are allowed to have arbitrary unbounded delays while critical ones need to satisfy the isochronic fork assumption [3].

2.3 Asynchronous controllers and their synthesis

Behaviour of asynchronous controllers can often be captured by Signal Transition Graphs (STGs) [6] [46], which depict the causal relation between signal transitions of a controller. For example, a micropipeline can be modelled by the STG in Figure 2.4, where $r0^+$ and $r0^-$ are the rising and falling transitions of $r0$ from 0 to 1 and 1 to 0, respectively. According to the token flow
rules, a transition can only be fired when all its incoming arcs are labelled with a dot (a token), and the fire of a transition brings a token to each of its outgoing arcs. In figure 2.4, 4 tokens are marked out to represent the initial states of a \textit{micropipeline} where the acknowledgement signals are pre-set to be 0.

STG and its forerunner \textit{Petri Net} [42] are widely used in the modelling, analysis, synthesis and verification of an asynchronous circuit [7] [71] [26] [23]. From the STG of \textit{micropipeline}, signal propagation cycles related with the circuit performance can be easily found. For example, one cycle coupling two consecutive pipeline stages is \( r0+ \rightarrow a0+ \rightarrow r1+ \rightarrow a1+ \rightarrow a0- \rightarrow r0+ \).

The \textit{State Graph} (SG, or \textit{Reachability Graph}) derived from STG provides a useful format of a circuit from state point of view. From the SG of a
circuit specification with Complete State Coding (CSC), circuit functions can be derived with the help of Boolean minimisation techniques. Furthermore, when the original STG is consistent with the signal transitions and persistent with the internal and output signals, there exists a SI implementation.

Synthesis of asynchronous controllers from their state encoding and boolean minimisations is supported by an automatic tool called Petrify [8]. Though formal in the light of explicit and deterministic encoding, Petrify suffers from an effectiveness problem where the states can explode in a exponential manner. This limitation makes Petrify best suited for control circuits rather than data paths [65].

State explosion problems can be by-passed if a circuit is synthesised by directly mapping the arcs and places in a STG specification to corresponding elements to mimic the token flows of the STG [59] [5] [54].

2.4 Asynchronous data paths

2.4.1 Dual-Rail (DR) Encoding and asynchronous data path protocols

DR code is one class of delay insensitive codes. When DR encoding is applied, a variable $n$ consists of two sub-variables $n^0$ and $n^1$ and can be viewed as the basic level for computation. $n$ conveys the information that is equal to a single bit with binary values in synchronous circuits, where $n^0$ ($n^1$) is
associated with binary value 0 (1). DR code is a systematic (separable) code.

With RZ protocol, the data path with DR encoding runs in two phases: the valid data and null cycles. In a valid data cycle, \( n \) changes from a spacer \((n_s)\) to a valid code word with value 1 \((n_{cw1})\) or 0 \((n_{cw0})\) where the sub-variable \(n^1\) or \(n^0\) is asserted to 1 in a mutually exclusive way, respectively. In the null cycle, \( n \) changes from either \(n_{cw1}\) or \(n_{cw0}\) to \(n_s\), where \(n^1\) or \(n^0\) is de-asserted.

With NRZ protocol, no null cycle is required between two valid data cycles. Transitions of \(n^1\) \((n^0)\) from both 1 to 0 and 0 to 1 can be viewed as the transmission of a binary data of value 1 (0).

Weak indication and strong indication was discussed in [48] as the two types of timing behaviour of an asynchronous data path with delay insensitive coding and RZ protocol. A data path is strongly indicating if no output can become a valid code word (spacer) until all inputs become valid code words (spacers) in a valid data (null) cycle. On the other hand, an output can become a valid code word (spacer) with only a sub-set of the inputs become valid code words (spacers) in a weakly indicating data path; still, evaluation of the last output waits for the last input.

### 2.4.2 Martin’s evaluation programme

A general method is introduced in [32] to design asynchronous data paths. In this method, functional evaluation of a strongly indicating data path is distributed over those of the data path’s composing cells that are concurrent,
through gradual relaxation of the global synchronisation. The function evaluation of a cell, \( C_k \), is described by the programme 2.1 using Communicating Hardware Process (CHP).

\[
( \quad * [B_{t_k} \land v(W_k) \rightarrow y_{t_k} \uparrow] \\
\| * [B_{f_k} \land v(W_k) \rightarrow y_{f_k} \uparrow] \\
\| * [n(W_k) \rightarrow y_{t_k} \downarrow \| y_{f_k} \downarrow] \quad )
\]

(2.1)

By this programme, only one of \( C_k \)'s outputs, \( y_{t_k} \) or \( y_{f_k} \), can be asserted to 1 in evaluating the function during a valid data cycle, if boolean condition \( B_{t_k} \) or \( B_{f_k} \) is true, respectively. Furthermore, the variables in \( W_k \), a superset of \( C_k \)'s inputs, must become valid code words for \( C_k \)'s output becoming a valid code word, and this is called validity checking in valid data cycle by the predicate \( v(W_k) \). Validity checking also exists in the null cycle by the predicate \( n(W_k) \) that all variables in \( W_k \) become spacers.

Validity checking during function evaluation can be optimised: \( W_k \) used in \( v(W_k) \) and \( n(W_k) \) can include different sets of variables. However, no transient input is allowed be introduced with this optimisation which eludes the valid data - null cycle between two valid data wavefronts.

Functional evaluation of a ripple-carry adder is shown in 2.2 with some
optimisation, from which a CMOS VLSI implementation is straightforward.

\[
\begin{align*}
  & \ast [(c_t \land eq(a, b)) \lor (c_f \land dif(a, b)) \rightarrow St \uparrow] \\
  & \parallel \ast [(c_f \land eq(a, b)) \lor (c_t \land dif(a, b)) \rightarrow Sf \uparrow] \\
  & \parallel \ast [(a_t \land bt) \lor ((a_t \lor bt) \land c_t) \rightarrow Dt \uparrow] \\
  & \parallel \ast [(a_f \land bf) \lor ((a_f \lor bf) \land c_f) \rightarrow Dt \uparrow] \\
  & \parallel \ast [\neg c_t \land \neg c_t \rightarrow St \downarrow \| Sf \downarrow] \\
  & \parallel \ast [\neg a_t \land \neg a_t \land bt \land \neg bf \rightarrow Dt \downarrow \| Df \downarrow]
\end{align*}
\]

(2.2)

### 2.4.3 Library design for asynchronous data paths

Various approaches have been proposed to design the library elements for asynchronous data paths with DR encoding and RTZ protocol. Some of these approaches are based on switching CMOS techniques while others on threshold gates.

A straightforward approach to design a DR module is by using two standard logical gates with the function to be implemented and the function’s complement. Output variables of a DR module synthesised by this approach have no indication of its inputs about their validity in either valid data or null cycle. In tradition this type of modules is called *early propagative* or has an *eager evaluation* because the output can be evaluated without waiting for
all inputs under some input values.

Direct logic (DL) is used in [69] for data path synthesis where the pull up networks of a DR static logic are replaced by the cascade of P-typed transistors that are controlled by both sub-variables of the inputs. By this, the output of a DL becomes a spacer only after all inputs become spacers in the null cycle, and we can say DL is strongly indicating in the null cycle. Figure 2.5(a) shows the implementation of a NAND functional module using DL.

The idea behind DL is further developed in Delay Insensitive Minterm Synthesis (DIMS) [58], which is strongly indicating in both valid data and null cycles. In this approach, all the minterms spanned by the input sub-variables are synchronised by the Muller-C elements, whose outputs then contribute to the output sub-variables according to the truth table. Note the number of minterms increases exponentially with the number of inputs. Figure 2.6 shows the implementation of an AND functional module DIMS.

In [39], DIMS is simplified with the Reduced Direct Logic (RDL) by reusing transistors in the pull down network, where general rules are presented to design RDL and avoid short-circuits. Figure 2.5(b) shows the implementation of a NAND functional module using RDL.

2.4.4 NCL synthesis flows

Different flows have been presented based on NCL approaches by Theseus Logic [34]. A distinctive library using threshold logic is reviewed first in this
Figure 2.5: Implementation of NAND function block using DL (a) and RDL (b)

Figure 2.6: DIMS implementation
section. Various flows are then discussed that generate weakly indicating data paths. In addition, the instructive concepts such as input completeness, orphan paths and orphan isolations will be reviewed in this section.

2.4.4.1 Threshold logic

Figure 2.7 shows the symbol of a general M-of-N threshold gate with weighted inputs, where the weight of input $x_i$ is $w_i$ (1 ≤ $i$ ≤ $N$). The function of a weighted M-of-N threshold gate [29] can be expressed as $S(x_1, x_2, ..., x_N) = 1$, if $w_1x_1 + w_2x_2 + ... + w_nx_n ≥ M$, in the valid data cycle and $R(x_1, x_2, ..., x_N) = x_1 ∨ x_2 ∨ ... ∨ x_N$ in the null cycle. By this expression, the output of a M-of-N threshold gate is asserted in the valid data cycle if the weight sum of the asserted inputs is at least $M$, and the output is deasserted in the null cycle only after all the inputs are deasserted. Note that the N-of-N threshold gate with $w_i = 1(1 ≤ i ≤ N)$ is the C-element with N inputs, whereas the 1-of-N threshold gate with $w_i = 1(1 ≤ i ≤ N)$ is the N-input OR gate. CMOS implementations of threshold gates are discussed in [50].

With the help of threshold gates, various threshold functions in the valid data cycle can be implemented with 2NCL operators, binary valued implementations of the symbolic operators based on DR encoding used in NCL. Figure 2.7 shows two examples of the threshold operators of the function $F = AB + BC + AC$ and $F = AB + AC + AD + BCD$. 28 2NCL operators exist for data path synthesis.
2.4.4.2 NCL data path synthesis

There are two general approaches in synthesising a NCL data path. The first approach is an example of SADT and begins with an initial structural description of the data path such as a single-rail netlist. During synthesis, each gate in the original structure is mapped to a DR module with the same function. By this approach, the topology of the initial structure is largely kept. Two representative methods exist in this approach: NCL-D and NCL-X.

In NCL-D[29], the modules used for mapping are strongly indicating. They can be either DIMS, RDL, or 2NCL operators satisfying this requirement. This method is sometimes called the DIMS expansion. Data paths designed using this method is weakly indicating, and validity of all the primary outputs are indications that the computation is done.

In NCL-X [25], the modules used for mapping are standard DR logical gates. As a result the data path is not weakly indicating if it is only made
up of these mapping modules. Extra circuitry is needed to detect completion of computation in the form of a C-element network whose inputs are tapped from input and internal variables in the data path.

The second approach in data path synthesis targets more general multi-level, multi-value combinational logic synthesis that is not based on SADT. A multi-value variable can have multiple sub-variables (2 in the case of DR code) representing its different values in the valid data cycle. In this approach, the functions of a data path in terms of sub-variables are first derived and then decomposed until they can be mapped to a netlist of 2NCL operators in the library. The initial sub-variable functions are often large sum-of-products that cannot be directly mapped. Therefore, product partitioning and sum partitioning are applied to break a product term into stages of smaller ones and divide the sum products into sub-groups, respectively. Moreover, optimisation by mutual exclusive inputs is available when mapping the sub-variable functions to different implementation technologies, such as the switching based technology (CMOS) or the threshold based ones. In this approach, there is no topology maintaining and direct translation, as compared to the first approach, and more design space can be explored.

We use a full adder to explain the second synthesis approach. First the sub-variable functions of a full adder are listed in 2.3, where \( a \) and \( b \) are the additions, \( c \) is the carry-in, \( S \) is the sum and \( D \) is the carry-out.
Figure 2.8: Full adder implementation by product and sum partitioning

\[
S^1 = a^1b^1c^1 + a^1b^0c^0 + a^0b^1c^0 + a^0b^0c^1,
\]

\[
S^0 = a^0b^0c^0 + a^1b^1c^0 + a^1b^0c^1 + a^0b^1c^1,
\]

\[
D^1 = a^1b^1c^1 + a^1b^1c^0 + a^1b^0c^1 + a^0b^1c^1,
\]

\[
D^0 = a^0b^0c^0 + a^1b^0c^0 + a^0b^1c^0 + a^0b^0c^1,
\]

(2.3)

With product partitioning, sum partitioning, and the substitution of \( w = a^0b^0, x = a^0b^1, y = a^1b^0, z = a^1b^1 \), the functions can be rewritten as:

32
\[
S^1 = zc^1 + yc^0 + xc^0 + wc^1 = c^0(x + y) + c^1(w + z), \\
S^0 = wc^0 + zc^0 + yc^1 + xc^1 = c^0(w + z) + c^1(x + y), \\
D^1 = zc^1 + zc^0 + yc^1 + xc^1 = z + c^1(x + y), \\
D^0 = wc^0 + yc^0 + xc^0 + wc^1 = w + c^0(x + y).
\]

The implementation of the adder is shown in Figure 2.8. However this is not the only way to decompose the sub-variable functions. \(D\) signal can be substituted in the evaluation of \(S\) with some tricks of boolean manipulations, and the resulting functions and implementations are listed in 2.4 and Figure 2.9 respectively.
\[
S^1 = a^1 b^1 c^1 + a^1 D^0 + b^1 D^0 + c^1 D^0, \\
S^0 = a^0 b^0 c^0 + a^0 D^1 + b^0 D^1 + c^0 D^1, \\
D^1 = a^1 b^1 + b^1 c^1 + a^1 c^1, \\
D^0 = a^0 b^0 + b^0 c^0 + a^0 c^0.
\]

(2.4)

2.4.4.3 Variable completeness, orphans and orphan isolation

When synthesising NCL data paths using the second approach, the completeness of the variables at the boundaries of the 2NCL operators should be satisfied. That is, by the time when variables on output boundaries become stable, those on input boundaries should also become stable. This means that the operators do not transition their output variables from spacers to valid code words until the input variables are completely valid data words, and then do not transition their output variables to spacers until the input variables are all spacers - very similar indeed to the requirements of strong indication.

The completeness is meant to prevent orphans which can traverse the variable boundaries. Orphans are defined in NCL data paths to be the ineffective paths under an input pattern that do not contribute to the evaluation of the data paths [34]. If an orphan traverses any variable boundary, it be-
comes a gate-orphan. Otherwise it is a wire-orphan. Orphans do not confuse
the correct data evaluation in a valid data cycle where they are produced.
However, orphans can disturb the next valid data cycle if they are not reset
between the two data cycles, as the new data will interweave with the old
ones. Compared with wire-orphans, gate-orphans are more dangerous, and
critical for the timing verification. In Figure 2.9, the gate-orphan is shown
in grey when $a$, $b$ and $c$ change to 0 in the valid data cycle (black lines show
the valid data path leading to the evaluation of a primary output), if $D$ is
not a primary output.

It is straightforward to ensure the completeness in direct mapping ap-
proach. However, more care is needed when designing NCL data paths by
sub-variable decomposition to ensure the completeness of the new internal
variables generated from decomposition. For example, $w, x, y, z$ in Figure 2.8
form a new 4-value variable boundary that ensures the completeness of $a$ and
$b$.

2.5 Conclusion

With the background chapter, the context of this research claimed in the
introduction chapter is clear. Three aspects of asynchronous data path de-
design using delay insensitive codes are discussed - the formal methods, design
libraries and automatic flows. In addition, the terminologies and definitions
of asynchronous circuits are introduced.
Chapter 3

Partial acknowledgement: its definitions, properties and characteristics

Partial Acknowledgement (PA) is the underlying concept in explaining the relationship between variables of an asynchronous data path. Different levels of PA distribution are utilised in the design practice. Furthermore, data paths satisfying PA requirements demonstrate certain characteristics in their behaviour, which will be discussed in this chapter.

In this chapter, PA is formally defined and its properties are discussed, PA distribution is explored at different levels with data path examples, and the behavioural characteristics of a data path satisfying PA requirements are studied.
3.1 Definitions of PA

The rising transition of a variable $n$, $n \uparrow$, is the transition of $n$ from a spacer ($n_s$) to a valid code word of either value 1 ($n_{cw1}$) or 0 ($n_{cw0}$). Conversely, the falling transition of $n$, $n \downarrow$, is the transition of $n$ from a valid code word to a spacer. Therefore, $n \uparrow \in \{n_{s\rightarrow cw0}, n_{s\rightarrow cw1}\}$ and $n \downarrow \in \{n_{cw0\rightarrow s}, n_{cw1\rightarrow s}\}$.

The rising (falling) transitions of a set of variables $N$, $N \uparrow (N \downarrow)$, is the transition of all the signals in $N$ from spacers (valid code words) to valid code words (spacers). As a result, $N \uparrow \in \prod_{n \in N} (n \uparrow)$ and $N \downarrow \in \prod_{n \in N} (n \downarrow)$.

A variable $n$ is partially acknowledged by a set of variables $M$ in the valid data cycle if the rising transition of $n$ causes the rising transitions of at least one variable in $M$, i.e., $\forall n \uparrow : (\exists m \in M : n \uparrow \rightarrow m \uparrow)$.

It is often the case that when $m \uparrow$ is caused by $n \uparrow$, $m \uparrow$ is also caused by the rising transitions of some other variables. However, the definition of PA only focuses on the causality between $n$ and $m$.

A variable $n$ is partially acknowledged by a set of variables $M$ in a null cycle if the falling transition of $n$ causes the falling transitions of at least one variable in $M$, i.e., $\forall n \downarrow : (\exists m \in M : n \downarrow \rightarrow m \downarrow)$.

$n$ is partially acknowledged by $M$ if $n$ is partially acknowledged by $M$ in both the valid data and null cycle.
3.2 Properties of PA

Property 1: PA is transitive. If \( n \) is partially acknowledged by \( m \), and \( m \) is partially acknowledged by \( p \), then \( n \) is partially acknowledged by \( p \).

Other forms of transitivity exist. For example, if \( n \) is partially acknowledged by \( \{m, p\} \), \( m \) is partially acknowledged by \( \{q, s\} \), and \( p \) is partially acknowledged by \( \{s, t\} \), then \( n \) is partially acknowledged by \( \{q, s, t\} \).

Property 2: The partial acknowledgement of \( n \) by \( \{m, p\} \) does NOT equal the partial acknowledgement of \( n \) by \( m \) or \( p \).

We first prove that if \( n \) is partially acknowledged by \( m \) or \( p \), then \( n \) is also partially acknowledged by \( \{m,p\} \). This is easily proved by the definition of partial acknowledgement. We then show a case where \( n \) is partially acknowledged by \( \{m,p\} \) but by neither \( m \) nor \( p \) individually. According to definition 3.1, \( n \) is partially acknowledged by \( \{m,p\} \), if

\[
\forall n \uparrow: (\exists m \uparrow: n \uparrow \rightarrow m \uparrow) \text{ or } (\exists p \uparrow: n \uparrow \rightarrow p \uparrow), \text{ and} \\
\forall n \downarrow: (\exists m \downarrow: n \downarrow \rightarrow m \downarrow) \text{ or } (\exists p \downarrow: n \downarrow \rightarrow p \downarrow). \quad (3.1)
\]

In expression 3.1, if \( n \) is partially acknowledged by \( m \) in the valid data cycle and \( p \) in null cycle, i.e.,

\[
\forall n \uparrow: (\exists m \uparrow: n \uparrow \rightarrow m \uparrow) \text{ and } (\exists p \uparrow: n \uparrow \rightarrow p \uparrow), \text{ and} \\
\forall n \downarrow: (\exists p \downarrow: n \downarrow \rightarrow p \downarrow) \text{ and } (\exists m \downarrow: n \downarrow \rightarrow m \downarrow), \quad (3.2)
\]

\( n \) is not partially acknowledged by \( m \) or \( p \), according to Definition 3.1.

Property 3: The partial acknowledgement of \( n \) by \( \{m,p\} \) in a valid data (null) cycle does NOT equal the partial acknowledgement of \( n \) by \( m \) or \( p \) in
the valid data (null) cycle.

According to definition 3.1, if \( n \) is partially acknowledged by \( m \) or \( p \) in the valid data cycle, then \( n \) is also partially acknowledged by \( \{m, p\} \) in the valid data cycle. In addition, \( n \) is partially acknowledged by \( \{m, p\} \) in the valid data cycle, if

\[
\forall n \uparrow \in n_{s \rightarrow \text{cw}0} : (\exists m \uparrow : n \uparrow \rightarrow m \uparrow) \text{ or } (\exists p \uparrow : n \uparrow \rightarrow p \uparrow), \text{ and }
\forall n \uparrow \in n_{s \rightarrow \text{cw}1} : (\exists m \uparrow : n \uparrow \rightarrow m \uparrow) \text{ or } (\exists p \uparrow : n \uparrow \rightarrow p \uparrow).
\]

(3.3)

In expression 3.3, if the rising transition of \( n \) from \( n_s \) to \( n_{\text{cw}0} \) causes the rising transition of \( m \) from \( m_s \) to \( m_{\text{cw}0} \), and the rising transition of \( n \) from \( n_s \) to \( n_{\text{cw}1} \) causes the rising transition of \( p \) from \( p_s \) to \( p_{\text{cw}1} \), i.e.,

\[
\forall n \uparrow \in n_{s \rightarrow \text{cw}0} : (\exists m \uparrow \in m_{s \rightarrow \text{cw}0} : n \uparrow \rightarrow m \uparrow) \text{ and } \neg(\exists p \uparrow : n \uparrow \rightarrow p \uparrow), \text{ and }
\forall n \uparrow \in n_{s \rightarrow \text{cw}1} : (\exists p \uparrow \in p_{s \rightarrow \text{cw}1} : n \uparrow \rightarrow p \uparrow) \text{ and } \neg(\exists m \uparrow : n \uparrow \rightarrow m \uparrow).
\]

(3.4)

\( n \) is not partially acknowledged by \( m \) or \( p \) in the valid data cycle, according to Definition 3.1.

### 3.3 Levels of PA distribution

Properties 2 and 3 show that the PA enables different levels of distribution over a group of variables. In this section, PA is classified in three categories by their levels of distribution and illustrated by different implementations of an original SR netlist in Figure 3.1(a).
3.3.1 Category 1

In this category, if \( n \) is partially acknowledged by \( m \) in the valid data (null) cycle, \( n \) is also partially acknowledged by \( m \) for the null (valid) data cycle. Therefore, the PA of \( n \) cannot be distributed. Figure 3.1(b) shows this level of distribution by NCL-D where each SR gate is mapped to its corresponding DIMS implementation. As a result, \( b \) is partially acknowledged by every output variable of the functional cell it fans into directly, i.e., \( e \) and \( f \), in both valid data and null cycle.

3.3.2 Category 2

In this category, \( n \) can be partially acknowledged by one variable in the valid data cycle and another variable in the null cycle. Martin’s programme 2.1 and its optimisation is an example of this type of distribution. For example, Figure 3.1(c) shows such an implementation where \( b \) is partially acknowledged by \( e \) for the rising phase and \( f \) for the falling phase. Another example is demonstrated by programme 2.2 where \( a \) and \( b \) are partially acknowledged by \( S \) in the valid data cycle and acknowledged by \( D \) in the null cycle, and \( c \) is partially acknowledged by \( S \) in both valid data and null cycles.

3.3.3 Category 3

This is the highest level of PA distribution, where the transition of \( n \) causes transitions of different variables under different input patterns, in a valid
Figure 3.1: PA distribution: original SR netlist (a), NCL-D implementation (b) and Category 2 implementation (c)

data (null) cycle. It is illustrated by the following data path examples.

Suppose a data path with input signals of \{a, b\} and output signals of \{c, d\} has the following functions:

\[
c^1 = a^1 \land b^1, \quad c^0 = a^0 \lor b^0;
\]

\[
d^1 = a^1 \lor b^1, \quad d^0 = a^0 \land b^0.
\]

In this example, \(a\) and \(b\) are partially acknowledged by \{c, d\} in both the valid data and null cycle. Only PA of \(a\) is examined here, whereas PA of \(b\) can be verified in a similar manner. In the valid data cycle, it can be observed that

\(\forall a \uparrow = a_{s \rightarrow cuw0} : ((\exists d \uparrow = d_{s \rightarrow cuw0} : a \uparrow \rightarrow d \uparrow) \lor (\exists c \uparrow = c_{s \rightarrow cuw0} : a \uparrow \rightarrow c \uparrow)),\)

and

\(\forall a \uparrow = a_{s \rightarrow cuw1} : ((\exists d \uparrow = d_{s \rightarrow cuw1} : a \uparrow \rightarrow d \uparrow) \lor (\exists c \uparrow = c_{s \rightarrow cuw1} : a \uparrow \rightarrow c \uparrow)).\)

In the null cycle, it can be seen that

\(\forall a \downarrow = a_{cuw0 \rightarrow s} : (\exists c \downarrow = c_{cuw0 \rightarrow s} : a \downarrow \rightarrow c \downarrow),\)

and

\(a \downarrow = a_{cuw1 \rightarrow s} : (\exists d \downarrow = d_{cuw1 \rightarrow s} : a \downarrow \rightarrow d \downarrow).\)
In the valid data cycle, PA of $a$ is distributed over the outputs depending on the input values. The rising transition of $a$ from $a_s$ to $a_{cw0}$ causes the rising transition of $d$ from $d_s$ to $d_{cw0}$ if $b^0 = 1$, or causes the rising transition of $c$ from $c_s$ to $c_{cw0}$ if $b^1 = 1$. In addition, the rising transition of $a$ from $a_s$ to $a_{cw1}$ causes the rising transition of $d$ from $d_s$ to $d_{cw1}$ if $b^0 = 1$, or causes the rising transition of $c$ from $c_s$ to $c_{cw1}$ if $b^1 = 1$.

This type of PA distribution is exhibited in another circuit with the functions of:

$$d^1 = a^1 \land b^1 \land c^1, \quad d^0 = a^0 \lor b^0 \lor c^0;$$
$$e^1 = (a^1 \land b^1) \lor (b^1 \land c^1) \lor (a^1 \land c^1), \quad e^0 = (a^0 \lor b^0) \land (b^0 \lor c^0) \land (a^0 \lor c^0);$$
$$f^1 = a^1 \lor b^1 \lor c^1, \quad f^0 = a^0 \land b^0 \land c^0,$$

where primary inputs $a$, $b$ and $c$ are partially acknowledged by the set of primary outputs, $\{d, e, f\}$. Analysis of this example is left for the reader as an exercise.

### 3.4 Asynchronous data paths with the PA requirements

In this section, the characteristics of the behaviour of asynchronous data paths are studied. Suppose the data path is made up of functional cells $C_i$, which has an input variable $k$ and output variable $i$. Furthermore, the data
path satisfies the PA requirements where each primary input and internal variable is partially acknowledged.

3.4.1 Validity checking

To explain the way in which the input validity is checked, two concepts are introduced: the direct PA fan-out set, \( \text{direct} - PA(k) \), and the minimal direct PA fan-out set, \( \text{min} - \text{direct} - PA(k) \), of a variable \( k \). \( \text{direct} - PA(k) \) is a set of functional cells that \( k \) fans into directly and whose outputs partially acknowledge \( k \). \( \text{min} - \text{direct} - PA(k) \) is a \( \text{direct} - PA(k) \) with no subsets that are \( \text{direct} - PA(k) \). In addition, the (minimal) direct PA fan-out sets for the valid data (null) cycle, \( \text{min} - \text{direct} - PA(k \uparrow) \) (\( \text{min} - \text{direct} - PA(k \downarrow) \)), can be defined in a similar manner.

Figure 3.2 illustrates above concepts. In the DR netlist, \( a \) is partially acknowledged by \( \{c, d\} \), but neither \( c \) nor \( d \). Therefore, \( \text{direct} - PA(a) = \text{min} - \text{direct} - PA(a) = \{c, d\} \). Furthermore, \( c \) is partially acknowledged by \( F \) (\( G \)). As a result, \( \text{direct} - PA(c) \in \{F, G, \{F, G\}\} \) and \( \text{min} - \text{direct} - PA(c) \in \{F, G\} \).

With the transitive property of PA, each input and internal variable \( k \) is partially acknowledged by a set of primary outputs, \( M_k \). Therefore, the transitions of all the variables in \( M_k \) from spacers to valid code words indicate that \( k \) has transitioned from a spacer to a valid code word, and the transitions of all the variables in \( M_k \) from valid code words to spacers indicate \( k \) has transitioned from a valid code word to a spacer. In this global manner, the
validity of \( k \) is checked by the primary outputs of the data path, and no transient primary inputs or internal variables exist: every \( k \) goes through the cycle of spacer-valid code word before it is used in the subsequent process of function evaluation.

The function evaluation process of \( C_i \), on the other hand, performs the local validity check of its direct fan-ins. \( k \) can be checked for its validity only if \( k \) has a \( \text{min} - \text{direct} - PA(k) \) that equals \( \{ C_i \} \). If this is the case, \( k \) is included in \( W_k \) of Martin’s functional evaluation programme 2.1 for the predicates of \( v(W_k) \) and \( n(W_k) \). Otherwise, if there exists a \( \text{min} - \text{direct} - PA(k) \) containing functional cells other than \( C_i \), the validity of \( k \) cannot be checked solely by \( C_i \)'s output. However, this does not mean the validity check of \( k \) is eliminated: it is still checked globally.
3.4.2 Timing relationships

3.4.2.1 Gate-orphans

No orphans can traverse the variable boundaries at the input and output of the functional cells in a circuit with the PA requirements. Suppose there is a path traversing through a functional cell $C_i$, from a transition of input $k$ to a transition of $i$. According to the transitivity of PA, this transition causes transitions of at least one primary output of the circuit. In other words, the data path’s functional evaluation depends on the transition of $k$. Therefore, no paths traversing a functional cell are ineffective, i.e., no gate-orphans exist.

3.4.2.2 Wire-orphans and their timing requirements

With the PA requirements, gate-orphans are excluded from a data path, but wire-orphans still exist. Specifically, wire-orphans form a sub-variable $n^x(x = 0 \text{ or } 1)$ that transitions in a valid data cycle including the paths from the forking point of $n^x$ to the inputs of a set of sub-variables $m^x$, where $m^x \in DirectFanout(n^x)$ and $\neg(n^x \rightarrow m^x)$.

Therefore, the complete characterisation of the behaviour of the data paths must include the timing requirement that all wire-orphans are reset before the computation of the next valid data cycle. This timing requirement is more conservative than the iso-chronic fork requirement, and is illustrated in Figure 3.3 in the context of a 3 stage pipeline architecture. In data path 2, a signal $n$ has direct fan-outs in both $m$ and $p$. Further, $n$ is partially
Figure 3.3: Timing relationship in partially acknowledged data paths

acknowledged by $m$ and not by $p$. The wire-orphan path (the black path) must propagate from the output of $n$ to the input of $p$ before the grey path propagates from $n$ through the combinational expression, the registers in L2 and the completion detection circuitry (CD2) to the output of $n$.

3.5 Notes

An acknowledgement is partial for two reasons. First, a transition usually represents only part of the causes of another transition. Second, a transition of a signal can cause the transitions of only a subset of the variables to which it is directly connected.

PA in this chapter is defined on the level of a DR variable. It is adequate for the explanation of the signal relationships of the data paths presented in this thesis. However, the definition of PA can be extended if required. Firstly, PA can be utilised to characterise the relationship among variables with different numbers of values. Secondly, PA can be defined on some lower
levels other than the current level of variables, such as the sub-variable level, in the data paths where the boundaries of some variables are missing. Third, PA can be extended to defined the signal relations where dual spacers and negative gates are applied, such as the design approaches in [55].

### 3.6 Summary

The definitions of PA depict the causality between variables in a data path. This causality and resultant properties enable certain characteristics in asynchronous data paths. They include the global and local validity check of the variables, and freedom from gate-orphans. Only wire-orphans are left in the data paths with the PA requirements, which is less critical and easily manageable. Furthermore, different levels of PA distribution exist in the design practice of asynchronous data paths.
Chapter 4

Design of dual-rail functional modules

Behaviour of a data path with PA requirements has been studied in chapter 3, and it is natural to proceed to the implementations of the data paths. In this chapter, the building blocks, namely Dual Rail Functional Modules (DRFMs), of the data path are designed by formal procedures using CMOS techniques. Factorisation using Boole's expansion theorem is introduced to ensure the PA requirements of an DRFM’s inputs. With the synthesis procedures, DRFMs with a wide spectrum of input PA patterns can be designed as the library for asynchronous data paths, including the standard logic gates with complementary CMOS structures.

The performance of DRFMs is studied in terms of Propagation Delays (PDs) from an input to an output. A linear delay model using Logical Effort
(LE) methods is applied to measure PDs. PDs are utilised for the timing simulations of asynchronous data paths, which are topics of the next chapter.

4.1 DRFMs and naming methods

An input $i$ can be partially acknowledged by $F$'s output in four modes: if $i$ is partially acknowledged in the valid data cycle, then the rising phase (denoted by $\uparrow$) of $i$ is said to be partially acknowledged; otherwise if $i$ is partially acknowledged in a null cycle, the falling phase, denoted by $\downarrow$, of $i$ is partially acknowledged; if $i$ is partially acknowledged in both valid data and null cycle, both rising and falling phase, denoted by $\ast$, of $i$ is partially acknowledged; finally, if $i$ is not partially acknowledged in either valid data or null cycle, neither rising nor falling phase, denoted by $-$, of $i$ is partially acknowledged. As a result, the acknowledgeable phase of $i, p_i$, belongs to the set of $\{\uparrow, \downarrow, \ast, -\}$.

DRFMs synthesised in this section has the following requirements: $F$ is a $min$ $-$ $direct$ $-$ $PA(i \uparrow)$ ($min$ $-$ $direct$ $-$ $PA(i \downarrow)$) of its input $i$ if the rising (falling) phase of $i$ is partially acknowledged by $F$'s output. In addition, only $F$ with complete interface boundaries of dual-rail input and output variables is considered.

A DRFM is named by its logic or arithmetic function followed by the way in which its inputs are partially acknowledged. For example, $\text{AND2}_a\uparrow b\downarrow$ is the 2-input DRFM with an AND function that partially acknowledges the ris-
ing phase of one input and the falling phase of the other. As another example, Martin’s adder by programme 2.2, implements $S$ with PARITY3 $a \uparrow b \uparrow c\ast$ and $D$ with MAJORITY3 $a \downarrow b \downarrow$.

### 4.2 Synthesis procedures of DRFMs

#### 4.2.1 Brief introduction

The structure of DRFM $F$ synthesised in this section consists of a Pull-Up Network (PUN) and a Pull-Down Network (PDN). The prototype of $F$’s structure is shown in Figure 4.1(a). The PDN performs the logic or arithmetic function as well as the PA task of those inputs whose rising phases are partially acknowledged by $F$’s output. On the other hand, the PUN resets $F$ before the evaluation of the next valid data cycle and performs the PA of those inputs whose falling phases are partially acknowledged by $F$’s output.

During synthesising a DRFM, measures against the short circuit currents and dangling outputs in the steady state should be considered. Short circuit currents and dangling outputs are largely avoided in the steady state of a complementary CMOS circuit, where the output is connected to either $V_{DD}$ through the PUN, or $V_{SS}$ through the PDN. However, the two problems arise if this complementary structure is lost.

MUX2 $a \uparrow c\ast$ (Figure 4.1(b)), a multiplexer with the select signal of $c$, is used as one example to illustrate the synthesis procedure.
Figure 4.1: The DRFM prototype (a) and an implementation of a MUX2 _a↑c_ (b)

4.2.2 Synthesis of the PDN

4.2.2.1 Synthesis of the PDNs

Three steps are used to synthesise the PDN of a DRFM $F$.

Step 1: Derive the sub-variable expressions, $F^0$ and $F^1$, of $F$ used in the evaluation of the valid data cycle. Suppose $f$ is the single-rail (SR) boolean function performing the DRFM’s logic or arithmetic functionality. $F^1$ is derived from $f$ by replacing every uncomplemented input $x$ in $f$ with $x^1$ and every complemented input $y$ with $y^0$. $F^0$ is the dual of $F^1$ derived by replacing $+$ (·) in $F^0$ with · (+), and the sub-variable of an input $i^1$ ($i^0$) with $i^0(i^1)$.
For example, the SR boolean function of $\text{MUX2:1}_a \uparrow c \ast$ is $f = ca + c'b$. Therefore, $F^1 = c^1a^1 + c^0b^1$ and $F^0 = (c^0 + a^0)(c^1 + b^0) = c^0c^1 + c^0b^0 + a^0c^1 + a^0b^0$. With boolean minimisations, $F^0 = c^0b^0 + a^0c^1$.

Step 2: Expand $F^1$ and $F^0$ into their factored forms [11], by Boole's expansion theorem, with respect to the inputs whose rising phases are to be partially acknowledged by $F$'s output. If there exists no such inputs, this step is skipped.

In this step, the co-factor of an output sub-variable $F^x$ with respect to an input sub-variable $\bar{y}^i$ (where $x, y = 0$ or 1), $F^x_{|^i}$, is used and computed by $F^x_{|^i} = F^x_{|i^y=1, \bar{y}'=0}$. For $\text{MUX2:1}_a \uparrow c \ast$, $F^1_{a^1} = F^1_{|a^1=1, a^0=0} = c^1 + c^0 \cdot b^1$, and $F^1_{a^0} = F^1_{|a^0=1, a^1=0} = c^0 \cdot b^1$.

$F^x_{|^i}$ can be understood as the function that must be evaluated to 1, if $\bar{y}^i$ is asserted, so that $F^x$ can also be asserted in a valid data cycle. For $\text{MUX2:1}_a \uparrow c \ast$, if $a^1$ is asserted in a valid data cycle, the conditions for $F^1$ to be asserted are either $c^1$ is 1 ($a$ is chosen by the select signal) or both $c^0$ and $b^1$ are 1 ($b$ is chosen). For a DRFM $F$ with AND function where $F^1 = a^1 \cdot b^1$ and $F^0_{a^0} = 0$, it is impossible to assert $F^1$ in a valid data cycle where $a^0$ has been asserted. Similarly, $(F^x_{|^i})_{j^z} (x, y, z = 0$ or 1) can be understood as the function that must be evaluated to 1, if both $i^y$ and $j^z$ are asserted, so that $F^x$ can also be asserted in a valid data cycle.

Suppose $\{x, y, z, ...\}$ are the inputs whose rising phases are to be partially acknowledged by $F$'s output, then $F^1$ and $F^0$ are expanded to the factored forms according to expression 4.1, with the expansion order of $x, y, z$ ...
order of expansion can influence PDs if transistors are reused in the synthesis procedures.

\[
F^1 = x^1 \cdot F^1_{x_1} + x^0 \cdot F^1_{x_0}
\]

\[
= x^1 \cdot (y^1 \cdot (F^1_{x_1})_{y^1} + y^0 \cdot (F^1_{x_1})_{y^0}) + x^0 \cdot (y^1 \cdot (F^1_{x_0})_{y^1} + y^0 \cdot (F^1_{x_0})_{y^0})
\]

\[
= x^1(y^1(z^1 \cdot ((F^1_{x_1})_{y^1})_{z^1} + z^0 \cdot ((F^1_{x_1})_{y^0})_{z^0}) + y^0(z^1 \cdot ((F^1_{x_1})_{y^1})_{z^1} + z^0 \cdot ((F^1_{x_1})_{y^0})_{z^0}))
\]

\[
+ x^0(y^1(z^1 \cdot ((F^1_{x_0})_{y^1})_{z^1} + z^0 \cdot ((F^1_{x_0})_{y^0})_{z^0}) + y^0(z^1 \cdot ((F^1_{x_0})_{y^1})_{z^1} + z^0 \cdot ((F^1_{x_0})_{y^0})_{z^0}))
\]

\[
= ...
\]

\[
F^0 = x^1 \cdot F^0_{x_1} + x^0 \cdot F^0_{x_0}
\]

\[
= x^1 \cdot (y^1 \cdot (F^0_{x_1})_{y^1} + y^0 \cdot (F^0_{x_1})_{y^0}) + x^0 \cdot (y^1 \cdot (F^0_{x_0})_{y^1} + y^0 \cdot (F^0_{x_0})_{y^0})
\]

\[
= x^1(y^1(z^1 \cdot ((F^0_{x_1})_{y^1})_{z^1} + z^0 \cdot ((F^0_{x_1})_{y^0})_{z^0}) + y^0(z^1 \cdot ((F^0_{x_1})_{y^1})_{z^1} + z^0 \cdot ((F^0_{x_1})_{y^0})_{z^0}))
\]

\[
+ x^0(y^1(z^1 \cdot ((F^0_{x_0})_{y^1})_{z^1} + z^0 \cdot ((F^0_{x_0})_{y^0})_{z^0}) + y^0(z^1 \cdot ((F^0_{x_0})_{y^1})_{z^1} + z^0 \cdot ((F^0_{x_0})_{y^0})_{z^0}))
\]

\[
= ...
\]

(4.1)

For example, the expanded sub-variable expressions of MUX2:1\_a\_c*, with the expansion order of \(a, b\), are shown in expression 4.2.
\[ F^1 = a^1(c^1 \cdot (F^0_{a1})_{c1} + c^0 \cdot (F^0_{a1})_{c0}) + a^0(c^1 \cdot (F^0_{a1})_{c1} + c^0 \cdot (F^0_{a0})_{c0}) \]
\[ = a^1(c^1 \cdot 1 + c^0 \cdot b^1) + a^0(c^1 \cdot 0 + c^0 \cdot b^1) \]
\[ = a^1(c^1 + c^0 \cdot b^1) + a^0(c^0 \cdot b^1), \]
\[ F^0 = a^1(c^1 \cdot (F^0_{a1})_{c1} + c^0 \cdot (F^0_{a1})_{c0}) + a^0(c^1 \cdot (F^0_{a1})_{c1} + c^0 \cdot (F^0_{a0})_{c0}) \]
\[ = a^1(c^1 \cdot 0 + c^0 \cdot b^0) + a^0(c^1 \cdot 1 + c^0 \cdot b^0) \]
\[ = a^1(c^0 \cdot b^0) + a^0(c^1 + c^0 \cdot b^0). \]

(4.2)

Step 3: Implement PDNs of \( F^1 \) and \( F^0 \) according to the expression 4.1, where AND function is implemented by series connection of NMOS transistors and OR function by parallel connection. The implementation method is the same as that of the PDNs of complex CMOS gates [35].

4.2.2.2 PA by PDNs

The factorisation and connection methods ensures the PA of the inputs whose rising phases are to be partially acknowledged. In fact, multiplying out the expressions in 4.1 give the expansions with respect to the minterms of the sub-variables of the inputs with the PA requirements. Suppose \( x, y, \) and \( z \) are the only inputs of \( F \) that are required to be partially acknowledged for the rising phases, 4.1 becomes 4.3 after the multiplication out.
\[
F^1 = x^1 \cdot y^1 \cdot z^1 \cdot F_{x^1y^1z^1}^1 + x^1 \cdot y^1 \cdot z^0 \cdot F_{x^1y^1z^0}^1 + x^1 \cdot y^0 \cdot z^1 \cdot F_{x^1y^0z^1}^1 + \\
+ x^1 \cdot y^0 \cdot z^0 \cdot F_{x^1y^0z^0}^1 + x^0 \cdot y^1 \cdot z^1 \cdot F_{x^0y^1z^1}^1 + x^0 \cdot y^1 \cdot z^0 \cdot F_{x^0y^1z^0}^1 \\
+ x^0 \cdot y^0 \cdot z^1 \cdot F_{x^0y^0z^1}^1 + x^0 \cdot y^0 \cdot z^0 \cdot F_{x^0y^0z^0}^1
\]

\[
F^0 = x^1 \cdot y^1 \cdot z^1 \cdot F_{x^1y^1z^1}^0 + x^1 \cdot y^1 \cdot z^0 \cdot F_{x^1y^1z^0}^0 + x^1 \cdot y^0 \cdot z^1 \cdot F_{x^1y^0z^1}^0 + \\
+ x^1 \cdot y^0 \cdot z^0 \cdot F_{x^1y^0z^0}^0 + x^0 \cdot y^1 \cdot z^1 \cdot F_{x^0y^1z^1}^0 + x^0 \cdot y^1 \cdot z^0 \cdot F_{x^0y^1z^0}^0 \\
+ x^0 \cdot y^0 \cdot z^1 \cdot F_{x^0y^0z^1}^0 + x^0 \cdot y^0 \cdot z^0 \cdot F_{x^0y^0z^0}^0
\]

(4.3)

In a valid data cycle, only one of the eight minterms with PA requirements, \( x^a \cdot y^b \cdot z^c \) (\( a, b, c = 0 \) or \( 1 \)), can be asserted. Suppose \( x^1 \cdot y^1 \cdot z^1 \) is asserted, \( F^1 \) will be asserted if \( F_{x^1y^1z^1}^1 \) is evaluated to 1 in this cycle, or otherwise \( F^0 \) will be asserted if \( F_{x^1y^1z^1}^0 \) is evaluated to 1, according to the understanding of co-factors. In either of the cases there exist three NMOS transistors connected in series that are controlled by \( x^1 \), \( y^1 \) and \( z^1 \) individually, according to the implementation approaches in step 3, from ground to an output sub-variable. Therefore, the rising transition of \( x \), \( y \) and \( z \) causes the rising transition of \( F \)'s output, and thus the PA requirement of the rising phases is satisfied.

PA requires that \( x^a \), \( y^b \) and \( z^c \) explicitly exist in every path from ground to an output sub-variable. As a result, they cannot be absorbed by boolean minimisation. For example, if \( F^1 \) in expression 4.2 is minimised to \( a^1c^1 + c^0b^1 \),
where \( a \) is absorbed by merging the terms of \( a^1c^0b^1 \) and \( a^0c^0b^1 \), the PA of \( a \) will be lost in the valid data cycle.

### 4.2.3 Synthesis of PUN

In a valid data cycle, there exist a subset of \( \{x^1, x^0\} \) of input \( x \) that can be asserted when \( F^1 \) or \( F^0 \) is asserted, which is denoted as \( F^1(x) \) or \( F^0(x) \), respectively. For example, in \text{MUX2:1}_{a\uparrow c\star}^\uparrow \), \( F^1(a) = F^0(a) = \{a^1, a^0\} \), \( F^1(b) = F^0(b) = \{b^1, b^0\} \), and \( F^1(c) = F^0(c) = \{c^1, c^0\} \). Sometimes the sub-variables in \( F^1(x) \) \( (F^0(x)) \) equals the explicit sub-variables of \( x \) (defined by support in [11]) appearing in the expressions of \( F^1 \) \( (F^0) \) in 4.1. For example, in \text{AND2}\_a\uparrow b\downarrow \) (Figure 4.2(b)), \( F^1(b) = \{b^1\} \), which is also the explicit sub-variables in \( F^1 \)'s expression in 4.2. However, there are cases where a sub-variable in \( F^1(x) \) \( (F^0(x)) \) does not exist in the expressions of \( F^1 \) \( (F^0) \). For example, in \text{MUX2:1}_{a\uparrow c\star} \), \( b^0 \) does not appear in the expression of \( F^1 \) in 4.2, however, it can be asserted when \( F^1 \) is asserted under the input values of \( a^1 = c^1 = b^0 = 1 \). Therefore, \( b^0 \) is included in \( F^1(b) \).

Suppose that an input \( x \) is to be partially acknowledged by \( F \)'s output for the falling phase, then each sub-variable in \( F^1(x) \) and \( F^0(x) \) controls a PMOS transistor connected in series between \( V_{DD} \) and the output of \( F^1 \) and \( F^0 \), respectively. In this way, no matter whether \( x^1 \) or \( x^0 \) is asserted in a valid data cycle, the de-assertion of \( x^1 \) or \( x^0 \) will cause the de-assertion of an output sub-variable; therefore, PA of the falling phases are ensured. For example, in \text{MUX2:1}_{a\uparrow c\star} \), \( F^1(c) = F^0(c) = \{c^1, c^0\} \) and therefore both
Figure 4.2: schematic of AND2_a↑b↑(a) and AND2_a↑b↓(b)

sub-variables of c control the PUNs, as in Figure 4.1(b).

If no inputs are partially acknowledged by F’s output for the falling phase, the PUNs has the liberty of being designed in a complementary manner to the PDN. Figure 4.2(a) shows such a complementary structure of NAND2_a↑b↑, where static short currents and dangling outputs are avoided in the steady state.

4.2.4 Static short currents and dangling outputs

Any on-path between VDD and VSS should be avoided in F’s steady state. This requires that there exists at least one input sub-variable controlling both NMOS and PMOS transistors in PDN and PUN paths of an output sub-variable that can be simultaneously on. For example, if a^0 is removed from the PUN of F^0 in Figure 4.2(b), static current arises in case where a^0
is asserted, and \( b^1 \) or \( b^0 \) is not asserted yet.

Dangling outputs exist in a non-complementary DRFM structure, and therefore some methods should be taken against the charge leakage problems. Either weak feedback inverters at the outputs or the cross-coupled p-typed transistors, as in DCVSL style [28], can be used.

### 4.3 Propagation delays of DRFMs

Propagation delay (PD) is the time used to propagate an input transition to an output one. PDs of DRFMs are used in the timing simulation of asynchronous data paths that will be discussed in the next chapter. In this thesis, a single switching input at a time is assumed, and no effects from simultaneous switching inputs are considered.

Let \( d_{i,F} \) denote the PD of \( F \) from its input \( i \). \( d_{i,F} \) can be one of the following specific delays, where \( x, y \) is 0 or 1:

\[
\begin{align*}
 d_{i^x,F^y} \uparrow, & \text{ the rising PD from an input sub-variable } i^x \text{ to an output sub-variable } F^y, \text{ if } i^x \text{ controls transistors in the PDN of } F^y; \\
 d_{i^x,F^y} \downarrow, & \text{ the falling PD from an input sub-variable } i^x \text{ to an output sub-variable } F^y, \text{ if } i^x \text{ controls transistors in the PUN of } F^y.
\end{align*}
\]

For example, the PD from input \( c \) to the output of \( \text{MUX2} \_\_c^* \), \( d_{c,F} \), can be \( d_{c^0,F^0} \uparrow, d_{c^0,F^1} \uparrow, d_{c^1,F^0} \uparrow, d_{c^1,F^1} \uparrow, d_{c^0,F^0} \downarrow, d_{c^0,F^1} \downarrow, d_{c^1,F^0} \downarrow \) and \( d_{c^1,F^1} \downarrow \).
4.3.1 skewed or unskewed structure

d_{i^x, F^x} \uparrow \text{ and } d_{i^x, F^x} \downarrow \text{ can be equal, if } F \text{ is designed in a unskewed manner, i.e., both the PUN and PDN have the equal ability to conduct currents. Alternatively in a skewed structure, } d_{i^x, F^x} \uparrow \text{ and } d_{i^x, F^x} \downarrow \text{ are different, and the conducting ability of either PUN or PDN is favoured. In this thesis, only unskewed structures are used.}

4.3.2 LE for PD computation

A linear delay model to compute a gate’s PD has the general form of \( d = k_1 C_L + k_2 \), where \( C_L \) is the load on the gate’s output, \( k_1 \) is a characterised slope and \( k_2 \) is an intrinsic delay. A linear delay model does not consider the effects of input transition times on a gate’s delay.

\( d_{i^x, F^x} \) is computed in this work using the linear equation provided by LE methods, \( d_{i^x, F^x} = g_i h_i + p_{F^x} \), where \( g_i \) and \( h_i \) are the logical and electrical efforts of \( i^x \), and \( p_{F^x} \) is the parasitic delay of \( F^x \). MUX2_a \uparrow c \uparrow is used as an example to illustrate the PD calculation. Transistor widths are computed assuming that an example module has the same driving ability of an ideal inverter, and are annotated in Figure 4.1(b).

The logic efforts of the sub-variables of the inputs are: \( g_{a^0} = g_{a^1} = \frac{2}{3} \), \( g_{b^0} = g_{b^1} = \frac{4}{3} \), \( g_{c^0} = \frac{16}{3} \), and \( g_{c^1} = \frac{12}{3} \). This calculation considers all the input capacitance loaded by an input sub-variable in a DRFM \( F \), as \( F \) is viewed as the basic functional cell in which transistors are strongly coupled.
Alternatively, only the capacitance of the transistors in the conducted path is counted, whereas the capacitance of the unconduted paths are attributed to the path branching effort. Also note that input \( a \) has a smaller logical effort than \( c \), though both of them are partially acknowledged for the rising phases, since \( a \) is expanded before \( c \) in the expression of 4.2.

The parasitic delays of the output sub-variables are: \( p_{F^0} = p_{F^1} = \frac{10}{3} \). All the calculations of logical efforts and parasitic delays are normalised to \( \tau \), the delay of an ideal inverter with no stray capacitance that drives another identical inverter, in a certain technology [60].

With the logical efforts and parasitic delays, PDs can be computed. For example, \( d_{\phi,F^0} = g_{\phi}h_{\phi} + p_{F^0} = \frac{16}{3}h_{\phi} + \frac{10}{3} \), \( d_{\phi,F^1} = g_{\phi}h_{\phi} + p_{F^1} = \frac{16}{3}h_{\phi} + \frac{10}{3} \), \( d_{c_1,F^0} = g_{c_1}h_{c_1} + p_{F^0} = \frac{12}{3}h_{c_1} + \frac{10}{3} \), and \( d_{c_1,F^1} = g_{c_1}h_{c_1} + p_{F^1} = \frac{12}{3}h_{c_1} + \frac{10}{3} \). Also, \( d_{\phi x,F^y} \) is smaller than \( d_{c_1,F^0} \) in a valid data (null) cycle, where \( x, y = 0 \) or 1.

The synthesis procedures discussed in section 2.2 are used in this example to design and analyse the PDs of the following DRFMs: \( AND2\_a \rightarrow b \rightarrow \), \( AND2\_a \ast b \rightarrow \), \( AND2\_a \ast b \ast \), \( OR2\_a \ast b \), and \( OR2\_a \ast b \downarrow \). These modules will be referred to in the following chapters.

Figure 4.3 and Figure 4.4 demonstrate the three DRFMs with AND function and two DRFMs with OR function, respectively. In these figures, transistor widths used for LE computation are annotated, and the input logical efforts and output parasitics are listed. PDs for each module are calculated.

For \( AND2\_a \rightarrow b \rightarrow \), \( d_{a^1,F^1} \uparrow = d_{a^1,F^1} \downarrow = g_{a^1}h_{a^1} + p_{F^1} = \frac{4}{3}h_{a^1} + 2 \); \( d_{a^0,F^0} \uparrow = \)
Figure 4.3: Schematic of $AND2\_a - b\ -$ (a), $AND2\_a \times b\ -$ (b), and $AND2\_a \times b^*\ -$ (c), where the input logical efforts, output parasitics, and transistor widths are annotated.
Figure 4.4: Schematic of $OR2 \_ a \ast b \uparrow$(a) and $OR2 \_ a \ast b \downarrow$(b), where the input logical efforts, output parasitics, and transistor widths are annotated.
\[ d_{a^0,F^0} \downarrow = g_{a^0} h_{a^0} + p_{F^0} = \frac{2}{3} h_{a^0} + 2; \ d_{b^1,F^1} \uparrow = d_{b^1,F^1} \downarrow = g_{b^1} h_{b^1} + p_{F^1} = \frac{4}{3} h_{b^1} + 2; \ d_{b^0,F^0} \uparrow = d_{b^0,F^0} \downarrow = g_{b^0} h_{b^0} + p_{F^0} = \frac{2}{3} h_{b^0} + 2. \]

For AND2 _a*b_, \[ d_{a^1,F^1} \uparrow = d_{a^1,F^1} \downarrow = g_{a^1} h_{a^1} + p_{F^1} = \frac{8}{3} h_{a^1} + \frac{4}{3} ; \ d_{a^1,F^0} \uparrow = d_{a^1,F^0} \downarrow = g_{a^1} h_{a^1} + p_{F^0} = \frac{8}{3} h_{a^1} + \frac{2}{3} ; \ d_{a^0,F^0} \uparrow = d_{a^0,F^0} \downarrow = g_{a^0} h_{a^0} + p_{F^0} = \frac{8}{3} h_{a^0} + \frac{2}{3} ; \ d_{b^1,F^1} \uparrow = d_{b^1,F^1} \downarrow = g_{b^1} h_{b^1} + p_{F^1} = \frac{2}{3} h_{b^1} + \frac{4}{3} ; \ d_{b^0,F^0} \uparrow = d_{b^0,F^0} \downarrow = g_{b^0} h_{b^0} + p_{F^0} = \frac{2}{3} h_{b^0} + \frac{2}{3} . \]

For OR2 _a*b_ \[ d_{a^1,F^1} \uparrow = d_{a^1,F^1} \downarrow = g_{a^1} h_{a^1} + p_{F^1} = 2 h_{a^1} + \frac{10}{3} ; \ d_{a^0,F^0} \uparrow = d_{a^0,F^0} \downarrow = g_{a^0} h_{a^0} + p_{F^0} = \frac{8}{3} h_{a^0} + \frac{4}{3} ; \ d_{a^0,F^0} \uparrow = d_{a^0,F^0} \downarrow = g_{a^0} h_{a^0} + p_{F^1} = \frac{8}{3} h_{a^0} + \frac{4}{3} ; \ d_{b^1,F^1} \uparrow = d_{b^1,F^1} \downarrow = g_{b^1} h_{b^1} + p_{F^1} = \frac{4}{3} h_{b^1} + \frac{10}{3} ; \ d_{b^0,F^0} \uparrow = d_{b^0,F^0} \downarrow = g_{b^0} h_{b^0} + p_{F^0} = \frac{4}{3} h_{b^0} + \frac{4}{3} ; \ d_{b^0,F^0} \uparrow = d_{b^0,F^0} \downarrow = g_{b^0} h_{b^0} + p_{F^1} = \frac{4}{3} h_{b^0} + \frac{2}{3} . \]

For OR2 _a*b_ \[ d_{a^1,F^1} \uparrow = d_{a^1,F^1} \downarrow = g_{a^1} h_{a^1} + p_{F^1} = \frac{10}{3} h_{a^1} + 4; \ d_{a^0,F^0} \uparrow = d_{a^0,F^0} \downarrow = g_{a^0} h_{a^0} + p_{F^0} = \frac{14}{3} h_{a^0} + 2; \ d_{a^0,F^0} \uparrow = d_{a^0,F^0} \downarrow = g_{a^0} h_{a^0} + p_{F^1} = \frac{14}{3} h_{a^0} + 4; \ d_{b^1,F^1} \uparrow = d_{b^1,F^1} \downarrow = g_{b^1} h_{b^1} + p_{F^1} = \frac{10}{3} h_{b^1} + 4; \ d_{b^0,F^0} \uparrow = d_{b^0,F^0} \downarrow = g_{b^0} h_{b^0} + p_{F^0} = \frac{14}{3} h_{b^0} + 2; \ d_{b^0,F^0} \uparrow = d_{b^0,F^0} \downarrow = g_{b^0} h_{b^0} + p_{F^1} = \frac{14}{3} h_{b^0} + 4; \]

### 4.4 Notes

Linear delay model is less precise compared with other complex delay metrics, such as the nonlinear delay model (NLDM) and scalable polynomial delay model (SPDM) from Synopsys. However, it provides a first-order com-
putation which can be done “on the back of an envelope” [60]. Further, the optimisation (in chapter 6) does not depend on the absolute delay values of the DRFMs.

4.5 Summary

This chapter discusses CMOS implementations and delay characterisation of DRFMS, the fundamental building blocks for designing asynchronous data paths. In the synthesis procedures, extra transistors are manipulated to ensure the PA requirements and avoid problems such as static short currents and dangling outputs in the steady state. Furthermore, influences of PA on a data path’s performance are studied in this chapter in terms of the functional modules’ propagation delays.
Chapter 5

Performance analysis of asynchronous data paths

The performance of an asynchronous data path can be influenced by two key factors: the logic functions and how the signals in the data path are partially acknowledged, i.e., the PA pattern. Two polar prototypes of a data path exist. In one type of data paths, every signal is partially acknowledged by all the DRFMs it fans into directly. This type of data path lends itself to a fully acknowledged data path, where PA dominates its data flow. In the second type, no signal is partially acknowledged for either rising or falling phases, and the logic functions dominate the data flow.

To capture these influences, the concept of dominating inputs is introduced and their determination criteria are discussed. A timing simulation technique is introduced for performance analysis of a partially acknowledged
data path when input values are given and the dominating inputs are determined. In addition, methods of static timing analysis are introduced where both min and max critical path delays can be derived. This static technique enables input independent analysis and characterise the true delays of a data path by the lower and upper bounds. These bounds might not be actually hit, and the sensitisation criterion for the critical paths are introduced in the final part of this chapter.

5.1 Controlling and non-controlling values

Concepts of the controlling value \((c(F))\) and non-controlling value \((nc(F))\) of a DRFM \(F\) are introduced in this section. Classifications by controlling and non-controlling values enable a generic study of the data flow of a DRFM. Suppose \(f\) is the logic function implemented by \(F\), \(c(F)\) is the binary logic value which independently determines the logic value of \(f\), whereas \(nc(F)\) is complementary to \(c(F)\). Examples of controlling and non-controlling values include:

- \(c(F)\) and \(nc(F)\) of a DRFM with AND (OR) function are 0 (1) and 1 (0), respectively.
- DRFMs with inverting or buffering functions have controlling values of both 0 and 1.
- DRFMs with XOR or NXOR functions have non-controlling values of
both 0 and 1.

5.2 Generic sub-variable expressions

For a DRFM with the boolean function such as AND or OR, logic functions of its dual-rail outputs can be expressed by controlling and non-controlling values. If there exists an input with the controlling value in the valid data cycle, the evaluation of $F$’s output is supported by this input, as illustrated by expression 5.1.

$$F f(\exists i \in DirectFanin(F):value(i)=c(F)) = \bigvee (i1^{c(F)}, i2^{c(F)}, \ldots, iN^{c(F)}),$$

(5.1)

where $i1, i2, \ldots, iN$ in expression 5.1 are the N inputs of $F$.

In the other case where all inputs have non-controlling values, the evaluation of $F$’s output requires that all inputs become valid data words. Expression 5.2 explains this case.

$$F f(\forall i \in DirectFanin(F):value(i)=nc(F)) = \bigwedge (i1^{nc(F)}, i2^{nc(F)}, \ldots, iN^{nc(F)}),$$

(5.2)

Sub-variable expressions of 5.1 and 5.2 are dual of each other. They help to explain the concept of dominating inputs in the next section when PA is absence from a functional module. For a specific example of this generic expansion, the sub-variable expressions of a DRFM with AND function are $F^1 = a^1 \land b^1$ and $F^0 = a^0 \lor b^0$. 

67
5.3 Dominating inputs of DRFMs

A dominating input of a DRFM $F$ can be understood as the input by whose transition the output is at the first time being supported to make a transition. Criteria to determine dominating inputs are presented in this section, first with and then without the presence of PA. Furthermore, the relation between dominating inputs and early propagation is unveiled.

5.3.1 Dominating inputs without PA

If $F$ has a simple function $f$, and no input is partially acknowledged by $F$’s output in both valid data and null cycle, generic expansions of 5.1 and 5.2 help to explain $F$’s data flows and determination criteria of its dominating inputs. If there exists at least one input with a $c(F)$, the rising transition of $F$’s output is supported by the earliest input with a controlling value. Otherwise if all inputs of $F$ have non-controlling values in the valid data cycle, the rising transition of $F$’s output will wait for the rising transition of the last input.

Data flows in null cycle depend on the residual data from the previous valid data cycle. If all inputs have non-controlling values, the falling transition of $F$’s output is supported by the earliest input. Otherwise, if at least one input has a $c(F)$ in the previous data cycle, the falling transition of $F$’s output will wait for the last input with a $c(F)$.

The following two definitions define the criteria to determine the domi-
nating input of $F$, where no input of $F$ is partially acknowledged by $F$’s output for either rising or falling phase.

**Criterion 5.1** $i$ dominates $F$ in a valid data cycle if $i$ satisfies one of the following requirements:

1. $i$ is the earliest input of $F$ that becomes a valid code word of $c(F)$;
2. $i$ is the last input of $F$ that becomes a valid code word if all of $F$’s inputs have $nc(F)s$ in the valid data cycle.

**Criterion 5.2** $i$ dominates $F$ in a null cycle if $i$ satisfies one of the following requirements:

1. $i$ is the earliest input of $F$ that becomes a spacer if all the inputs to $F$ have $nc(F)s$ in the previous data cycle;
2. $i$ is the last input of $F$ that becomes a spacer and has a $c(F)$ in the previous data cycle.

### 5.3.2 Dominating inputs with PA

PA influences the determination of dominating inputs besides the logic functions: if an input $i$ of $F$ is partially acknowledged by $F$’s output in the valid data cycle, the transition of $F$’s output from a spacer to a valid code word will wait for the transition of $i$ from a spacer to a valid code word, no matter whether $i$ has a controlling or non-controlling value. Similarly, if $i$ is partially acknowledged by $F$’s output in null cycle, the transition of $F$’s output from a valid code word to a spacer will wait for the transition of $i$ from a valid
code word to a spacer.

The dominating inputs of a DRFM are defined in the following two definitions, considering the synergy of both logic functions and PA.

**Criterion 5.3** \( i \) dominates \( F \) in a valid data cycle if \( i \) satisfies one of the following requirements:

(1) \( i \) is the later one of \( a \) and \( b \), where \( a \) is the earliest input of \( F \) that becomes a valid code word of \( c(F) \), and \( b \) is the last input that becomes a valid code word and that is partially acknowledged by \( F \)'s output in the valid data cycle;

(2) \( i \) is the last input that becomes a valid code word, if all the inputs to \( F \) have non-controlling values in the valid data cycle.

**Criterion 5.4** \( i \) dominates \( F \) in a null cycle if \( i \) is the last input that becomes a spacer and is partially acknowledged by \( F \)'s output in null cycle.

For example, the dominating inputs (in sub-columns denoted by d.i.) of \( \text{AND2}_a - b - \) and \( \text{AND2}_a \times b - \) (Figure 4.3(a) and (b), respectively) are listed in Table 5.1 under different input vectors (in the column denoted by \( v \)), where \( e(a, b) \) (\( l(a, b) \)) denotes the earlier (later) input that becomes a valid code word in a valid data cycle, or a spacer in a null cycle. For instance, the dominating input of \( \text{AND2}_a \times b - \) in valid data cycle under \( a^0 = b^0 = 1 \) is \( \text{later} (\text{earlier}(a, b), a) = a \), according to the determination criterion (denoted by crit) of 5.3(1).
### Table 5.1: Dominating inputs and the determination criteria of \( \text{AND2} \_a - b - \) and \( \text{AND2} \_a * b - \)

<table>
<thead>
<tr>
<th>( v )</th>
<th>( \text{AND2} _a - b - )</th>
<th>( \text{AND2} _a * b - )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>valid data cycle</td>
<td>null cycle</td>
</tr>
<tr>
<td></td>
<td>d.i.</td>
<td>crit</td>
</tr>
<tr>
<td>( a^0 = b^0 = 1 )</td>
<td>( e(a, b) )</td>
<td>5.1(1)</td>
</tr>
<tr>
<td>( a^1 = b^1 = 1 )</td>
<td>( a )</td>
<td>5.1(1)</td>
</tr>
<tr>
<td>( a^1 = b^1 = 1 )</td>
<td>( l(a, b) )</td>
<td>5.1(2)</td>
</tr>
</tbody>
</table>

#### 5.3.3 Dominating inputs and early propagation

Early propagation (EP) is the phenomenon that the output becomes a valid code word (spacer) before all inputs become valid code words (spacers) in a valid data (null) cycle. An input \( i \) is an early propagated input in a cycle, if \( i \) is the dominating input and \( i \) is not the last input that makes a transition. For example, In \( \text{AND2} \_a - b - \), \( \text{earlier}(a, b) \) early propagates to the output in the valid data cycle with the input vector of \( a^0 = b^0 = 1 \), whereas no input early propagates in the valid data cycle with \( a^1 = b^1 = 1 \).

#### 5.4 Input dependent timing simulations

One straightforward application of dominating inputs is the input dependent timing simulations of an asynchronous data path under an input vector. This type of simulation is useful to analyse the performance of a data path when the inputs are known \( a \) priori. The notations used in the simulation are illustrated with the help of Figure 5.1. In this figure, the output of DRFM \( c_k \)
drives net $k$, which in turn is connected to the input of DRFM $c_i$; similarly, the output of $c_i$ drives net $i$, which is connected to the input of $c_j$. Let $t(c_i \uparrow)$ and $t(c_i \downarrow)$ denote the arrival times of the rising and falling transitions of $i$ at $c_i$’s output, respectively. Furthermore, $t(k, c_i \uparrow)$ and $t(k, c_i \downarrow)$ denote the arrival times of the rising and falling transitions of $k$ at $c_i$’s input, respectively.

In a valid data cycle, $t(c_i \uparrow) = t(k, c_i \uparrow) + d_{k,c_i} \uparrow$, where $k$ is the dominating input of $c_i$ in the valid data cycle and $d_{k,c_i} \uparrow$ is the rising PD of $c_i$ from its input sub-variable $k^x$ to $c_i$’s output sub-variable $i^x$ that are asserted in this valid data cycle.

In a null cycle, $t(c_i \downarrow) = t(k, c_i \downarrow) + d_{k,c_i} \downarrow$, where $k$ is the dominating input of $c_i$ in the null cycle and $d_{k,c_i} \downarrow$ is the falling PD of $c_i$ from an input sub-variable $k^x$ to $c_i$’s output sub-variable $i^x$ that are de-asserted in the null cycle. In addition, $t(k, c_i \uparrow)$ and $t(k, c_i \downarrow)$ are determined by $t(k, c_i \uparrow) = t(c_k \uparrow) + D_{c_k,c_i} \uparrow$, and $t(k, c_i \downarrow) = t(c_k \downarrow) + D_{c_k,c_i} \downarrow$, respectively. $D_{c_k,c_i} \uparrow$ and $D_{c_k,c_i} \downarrow$ are the rising and falling Wire Delays (WDs) from the output of $c_k$ to the input of $c_i$ in a valid data and null cycle, respectively.

*Example 5.1* Figure 5.2 shows a data path where the DRFMs $c_d$, $c_e$, $c_f$,
Figure 5.2: Data path example for min-max timing analysis

c_g and c_h are AND2_a * b*, AND2_a' - c*, AND2_b - c*, OR2_d * e ↑, and OR2_e ↓ f*, respectively. c_{a'} is the DRFM of inverting function whose propagation delays are assumed to be 0. In this example, timing simulations are performed for the valid data cycle with the input vector of a = 0, b = 0, c = 1. Primary inputs of the data path have the following arrival times: t(a ↑) = 0, t(b ↑) = 1 and t(c ↑) = 2. PDs of the functional modules have been calculated in Example 4.3.2, where electrical efforts is simply calculated as the ratio of fan-out and fan-in numbers. Furthermore, WDs are assumed to be 1.

For c_d, the arrival times of its inputs are:

t(a, c_d ↑) = t(a ↑) + D_{a,c_d} ↑ = 1, and
t(b, c_d ↑) = t(b ↑) + D_{b,c_d} ↑ = 2.

According to Definition 5.3(1), b dominates c_d in this cycle. Suppose d is the second input being expanded when using expression 4.1 to design the PDN,
\[ t(c_d \uparrow) = t(b, c_d \uparrow) + d_{b \varphi, c_d} \uparrow = 2 + (4h_{b \varphi} + \frac{14}{3}) = 2 + 4 \cdot 1 + \frac{14}{3} = 10.7. \]

For \( c_e \), the arrival times of its inputs are:

\[ t(a', c_e \uparrow) = t(a \uparrow) + D_{a,c_e} \uparrow = 1, \text{ and} \]
\[ t(c, c_e \uparrow) = t(c \uparrow) + D_{c,c_e} \uparrow = 3. \]

According to Definition 5.3(2), \( c \) dominates \( c_e \) in this cycle. Therefore,

\[ t(c_e \uparrow) = t(c, c_e \uparrow) + d_{c_1,c_e} \uparrow = 3 + (\frac{5}{3}h_{c_1} + \frac{4}{3}) = 3 + \frac{5}{3} \cdot 2 + \frac{4}{3} = 9.7. \]

For \( c_f \), the arrival times of its inputs are:

\[ t(b, c_f \uparrow) = t(b \uparrow) + D_{b,c_f} \uparrow = 2, \text{ and} \]
\[ t(c, c_f \uparrow) = t(c \uparrow) + D_{c,c_f} \uparrow = 3. \]

According to Definition 5.1(1), \( b \) dominates \( c_f \) in this cycle. Therefore,

\[ t(c_f \uparrow) = t(b, c_f \uparrow) + d_{b \varphi, c_f} \uparrow = 2 + (\frac{5}{3}h_{b \varphi} + 2) = 2 + \frac{5}{3} \cdot 1 + 2 = 5.7. \]

For \( c_g \), the arrival times of its inputs are:

\[ t(d, c_g \uparrow) = t(c_d \uparrow) + D_{c_d,c_g} \uparrow = 11.7, \text{ and} \]
\[ t(e, c_g \uparrow) = t(c_e \uparrow) + D_{c_e,c_g} \uparrow = 10.7. \]

According to Definition 5.3(1), \( d \) dominates \( c_g \) in this cycle. Therefore,

\[ t(c_g \uparrow) = t(d, c_g \uparrow) + d_{c_{d1},c_g} \uparrow = 11.7 + (\frac{5}{3}h_{c_{d1}} + \frac{10}{3}) = 11.7 + \frac{5}{3} + \frac{10}{3} = 17.7. \]

Finally for \( c_h \), the arrival times of its inputs are:

\[ t(e, c_h \uparrow) = t(c_e \uparrow) + D_{c_e,c_h} \uparrow = 10.7, \text{ and} \]
\[ t(f, c_h \uparrow) = t(c_f \uparrow) + D_{c_f,c_h} \uparrow = 6.7. \]

According to Definition 5.3(1), \( e \) dominates \( c_h \) in this cycle. Therefore,

\[ t(c_h \uparrow) = t(e, c_h \uparrow) + d_{c_{e1},c_h} \uparrow = 10.7 + (\frac{10}{3}h_{c_{e1}} + 4) = 10.7 + \frac{10}{3} + 4 = 18. \]

It can be seen from this example that input-dependent timing simulation completely determines the timing behaviour of a partially acknowledged data
path under an input vector. However, when information on input value is not available, some static timing techniques are required to analyse the performance of a data path, which will be introduced in the next section.

5.5 Min-max static timing analysis

Static timing analysis (STA) is an input independent technique to calculate circuit delays. It is widely used in synchronous designs where timing closure must be satisfied. STA is needed for asynchronous data paths as well, by which critical paths can be derived and utilised as the objectives for optimisation (will be studied in Chapter 6). Both min and max STA are discussed in this section.

Min STA captures the early propagation phenomena in a partially acknowledged data path, whereas max STA does not allow any early propagation. In both min and max STA, two bounds exist for the arrival times: the lower bound, which is the earliest time that a transition is propagated to a place, and the upper bound, which is the latest one. Lower and upper bounds of arrival times are calculated in sections 5.5.1 and 5.5.2, using the notations introduced below with a reference to Figure 5.1.

In min STA, the lower and upper bounds of $i$’s arrival time at the output of $c_i$ are denoted by $\min_{lb}(c_i)$ and $\min_{ub}(c_i)$, respectively. The arrival times can be those of $i$’s rising transition, $\min_{lb}(c_i \uparrow)$ and $\min_{ub}(c_i \uparrow)$, in a valid data cycle, or $i$’s falling transition, $\min_{lb}(c_i \downarrow)$ and $\min_{ub}(c_i \downarrow)$,
in a null cycle.

The lower and upper bounds of $k$’s arrival time at the input of $c_i$ are denoted by $\text{min\_lb}(k, c_i)$ and $\text{min\_ub}(k, c_i)$, respectively. In a similar fashion, they are further divided to $\text{min\_lb}(k \uparrow, c_i)$ and $\text{min\_ub}(k \uparrow, c_i)$ in the valid data cycle, and $\text{min\_lb}(k \downarrow, c_i)$ and $\text{min\_ub}(k \downarrow, c_i)$ in a null cycle.

The lower and upper bounds of the propagation delays from $c_i$’s input $k$ to output $i$ are denoted by $\text{min\_d}_{k,c_i}$, which is the minimum PD from $k$ to $i$ over all possible values of the sub-variables $k^x$ and $i^x$, and $\text{max\_d}_{k,c_i}$, the maximum PD over all possible values. Further, $\text{min\_d}_{k,c_i}$ ($\text{max\_d}_{k,c_i}$) can be divided into the rising PDs of $\text{min\_d}_{k,c_i} \uparrow$ ($\text{max\_d}_{k,c_i} \uparrow$) in a valid data cycle and the falling PDs of $\text{min\_d}_{k,c_i} \downarrow$ ($\text{max\_d}_{k,c_i} \downarrow$) in a null cycle.

The lower and upper bounds of Wire Delays (WDs) from the output of $c_k$ to the input of $c_i$ are denoted by $\text{min\_D}_{c_k,c_i}$, which can be either the rising WD of $\text{min\_D}_{c_k,c_i} \uparrow$ in a valid data cycle or falling WD of $\text{min\_D}_{c_k,c_i} \downarrow$ in a null cycle, and $\text{max\_D}_{c_k,c_i}$, which can be either $\text{max\_D}_{c_k,c_i} \uparrow$ or $\text{max\_D}_{c_k,c_i} \downarrow$.

For max STA, $\text{min}$ in the notations of arrival times of min STA is replaced with $\text{max}$, and the notations of PDs and WDs remain the same as those in the min STA.

### 5.5.1 Arrival times in min STA

Early propagation captured by min STA is discussed in two scenarios. In the first scenario, no input is partially acknowledged by $c_i$’s output in a valid data or null cycle, and a transition of every input is assumed to be able to early
propagate to an output one. With this idea, \( \text{min\_lb}(c_i \uparrow) \) and \( \text{min\_ub}(c_i \uparrow) \) are calculated by expressions 5.3 and 5.4.

\[
\text{min\_lb}(c_i \uparrow) = \text{Min}_{k \in \text{DirectFanin}(c_i)}(\text{min\_lb}(k \uparrow, c_i) + \text{min\_d}_{k,c_i} \uparrow) \quad (5.3)
\]

\[
\text{min\_ub}(c_i \uparrow) = \text{Min}_{k \in \text{DirectFanin}(c_i)}(\text{min\_ub}(k \uparrow, c_i) + \text{max\_d}_{k,c_i} \uparrow) \quad (5.4)
\]

In addition, \( \text{min\_lb}(k \uparrow, c_i) \) and \( \text{min\_ub}(k \uparrow, c_i) \) are computed according to expressions 5.5 and 5.6, where \( k \) is driven by \( c_i \).

\[
\text{min\_lb}(k \uparrow, c_i) = \text{min\_lb}(c_k \uparrow) + \text{min\_D}_{c_k,c_i} \uparrow. \quad (5.5)
\]

\[
\text{min\_ub}(k \uparrow, c_i) = \text{min\_ub}(c_k \uparrow) + \text{max\_D}_{c_k,c_i} \uparrow. \quad (5.6)
\]

In the second scenario where some inputs are partially acknowledged by \( c_i \)'s output in a valid data (null) cycle, an output transition waits for the last transition of the input that is partially acknowledged in that cycle. In this case, early propagation is still possible but only for a partially acknowledged input which leads to the the maximum output arrival time when it is assumed to be the dominating input. Calculations of \( \text{min\_lb}(c_i \uparrow) \) and \( \text{min\_ub}(c_i \uparrow) \) in this scenario are according to expressions 5.3’ and 5.4’, whereas \( \text{min\_lb}(k \uparrow, c_i) \) and \( \text{min\_ub}(k \uparrow, c_i) \) are the same to 5.5 and 5.6.

\[
\text{min\_lb}(c_i \uparrow) = \text{Max}_k(\text{min\_lb}(k \uparrow, c_i) + \text{min\_d}_{k,c_i} \uparrow). \quad (5.3')
\]

\[
\text{min\_ub}(c_i \uparrow) = \text{Max}_k(\text{min\_ub}(k \uparrow, c_i) + \text{max\_d}_{k,c_i} \uparrow). \quad (5.4')
\]

In 5.3’ and 5.4’, \( k \in \text{DirectFanin}(c_i) \) and \( k \) is partially acknowledged by \( i \) for the rising phase.

Arrival times in a null cycle using min STA can be analysed two scenarios similar to those in the valid data cycle, and the calculation formula in each of the scenario replaces \( \uparrow \) with \( \downarrow \) in the corresponding scenario in the valid

77
data cycle.

5.5.2 Arrival times in max STA

In max STA, no input transition is allowed to be early propagated, and the output transition waits for the transitions of all the inputs. This is the assumption for calculations 5.7 and 5.8, regardless of PA patterns of an asynchronous data paths.

\[
\text{max}_{lb}(c_i \uparrow) = \text{Max}_k(\text{max}_{lb}(k \uparrow, c_i) + \text{min}_d k, c_i \uparrow). \tag{5.7}
\]

\[
\text{max}_{ub}(c_i \uparrow) = \text{Max}_k(\text{max}_{ub}(k \uparrow, c_i) + \text{max}_d k, c_i \uparrow). \tag{5.8}
\]

In expressions 5.7 and 5.8, \(k \in DirectFanin(c_i)\).

In addition, \(\text{max}_{lb}(k \uparrow, c_i)\) and \(\text{max}_{ub}(k \uparrow, c_i)\) are calculated according to expression 5.9 and 5.10, respectively, where \(k\) is driven by \(c_k\).

\[
\text{max}_{lb}(k \uparrow, c_i) = \text{max}_{lb}(c_k \uparrow) + \text{min}_D k, c_i \uparrow. \tag{5.9}
\]

\[
\text{max}_{ub}(k \uparrow, c_i) = \text{max}_{ub}(c_k \uparrow) + \text{max}_D k, c_i \uparrow. \tag{5.10}
\]

For the null cycle, calculations of the arrival times will be analogous to those in 5.7-5.10 if there exist no inputs being partially acknowledged by \(c_i\)'s output for the falling phases. On the other hand, if there exist at least one input that is partially acknowledged for the falling phase, the calculations of the arrival times are according to expressions 5.6' and 5.7', because only inputs that are partially acknowledged for the falling phases control the PUN due to the synthesis procedures of DRFM.

\[
\text{max}_{lb}(c_i \downarrow) = \text{Max}_k(\text{max}_{lb}(k \downarrow, c_i) + \text{min}_d k, c_i \downarrow). \tag{5.7'}
\]

\[
\text{max}_{ub}(c_i \downarrow) = \text{Max}_k(\text{max}_{ub}(k \downarrow, c_i) + \text{max}_d k, c_i \downarrow). \tag{5.8'}
\]
In 5.7' and 5.8', $k \in \text{DirectFanin}(c_i)$ and $k$ is partially acknowledged by $i$ for the falling phase.

**Example 5.2** Calculate signal arrival times in the valid data cycle of the data path in Figure 5.2 by both min and max STA. The primary inputs, $\{a, b, c\}$, have arrival times of 0, and the lower and upper bounds of WDs are $\text{min}_D = 1$ and $\text{max}_D = 2$.

First, the lower and upper bounds of PDs are estimated by the results from Example 4.3.2, where electrical efforts is simply calculated as the ratio of fan-out and fan-in numbers. For example, the PDs from input $a$ to the output of $c_d$, $d_{a,c_d}$, could be one of $d_{a^1,c_d^1}$, $d_{a^1,c_d^0}$, and $d_{a^0,c_d^0}$, which have the values of 6.7, 9.3, and 8, respectively. As a result, $\text{min}_{d_{a,c_d}} = 6.7$ and $\text{max}_{d_{a,c_d}} = 9.3$.

With similar analysis, the lower and upper bounds of the PDs in this data path are:

\[
\begin{align*}
\text{min}_{d_{b,c_d}} &= 7.3, \quad \text{max}_{d_{b,c_d}} = 10; \\
\text{min}_{d_{a',c_e}} &= 2.7, \quad \text{max}_{d_{a',c_e}} = 3.7; \\
\text{min}_{d_{c,c_e}} &= 5.7, \quad \text{max}_{d_{c,c_e}} = 7.7; \\
\text{min}_{d_{b,c_f}} &= 3.3, \quad \text{max}_{d_{b,c_f}} = 3.7; \\
\text{min}_{d_{c,c_f}} &= 3.3, \quad \text{max}_{d_{c,c_f}} = 3.7; \\
\text{min}_{d_{d,c_g}} &= 4, \quad \text{max}_{d_{d,c_g}} = 6; \\
\text{min}_{d_{e,c_g}} &= 2.7, \quad \text{max}_{d_{e,c_g}} = 4.7; \\
\text{min}_{d_{e,c_h}} &= 6.7, \quad \text{max}_{d_{e,c_h}} = 8.7; \\
\text{min}_{d_{f,c_h}} &= 6.7, \quad \text{max}_{d_{f,c_h}} = 8.7.
\end{align*}
\]
Secondly the arrival times of the DRFMs in this data path are computed by both min and max STA.

For $c_d$, the input arrival times are: $\min_{lb}(a \uparrow, c_d) = 1$, $\min_{ub}(a \uparrow, c_d) = 2$, $\min_{lb}(b \uparrow, c_d) = 1$, $\min_{ub}(b \uparrow, c_d) = 2$. According to expression 5.3' and 5.4',

$$\min_{lb}(c_d \uparrow) = \max(\min_{lb}(a \uparrow, c_d) + \min_{d_{a,c_d}} \uparrow, \min_{lb}(b \uparrow, c_d) + \min_{d_{b,c_d}} \uparrow) = 8.3;$$

$$\min_{ub}(c_d \uparrow) = \max(\min_{ub}(a \uparrow, c_d) + \max_{d_{a,c_d}} \uparrow, \min_{ub}(b \uparrow, c_d) + \max_{d_{b,c_d}} \uparrow) = 12.$$

For $c_e$, the input arrival times are: $\min_{lb}(a' \uparrow, c_e) = 1$, $\min_{ub}(a' \uparrow, c_e) = 2$, $\min_{lb}(c \uparrow, c_e) = 1$, $\min_{ub}(c \uparrow, c_e) = 2$. According to expressions 5.3' and 5.4',

$$\min_{lb}(c_e \uparrow) = \max(\min_{lb}(c \uparrow, c_e) + \min_{d_{c,c_e}} \uparrow) = 6.7;$$

$$\min_{ub}(c_e \uparrow) = \max(\min_{ub}(c \uparrow, c_e) + \max_{d_{c,c_e}} \uparrow) = 9.7.$$

For $c_f$, the input arrival times are: $\min_{lb}(b \uparrow, c_f) = 1$, $\min_{ub}(b \uparrow, c_f) = 2$, $\min_{lb}(c \uparrow, c_f) = 1$, $\min_{ub}(c \uparrow, c_f) = 2$. According to expressions 5.3 and 5.4,

$$\min_{lb}(c_f \uparrow) = \min(\min_{lb}(b \uparrow, c_f) + \min_{d_{b,c_f}} \uparrow, \min_{lb}(c \uparrow, c_f) + \min_{d_{c,c_f}} \uparrow) = 4.3;$$

$$\min_{ub}(c_f \uparrow) = \min(\min_{ub}(b \uparrow, c_f) + \max_{d_{b,c_f}} \uparrow, \min_{ub}(c \uparrow, c_f) + \max_{d_{c,c_f}} \uparrow) = 5.7.$$

For $c_g$, the input arrival times are: $\min_{lb}(d \uparrow, c_g) = 9.3$, $\min_{ub}(d \uparrow, c_g) = 14$, $\min_{lb}(e \uparrow, c_g) = 7.7$, $\min_{ub}(e \uparrow, c_g) = 10.7$. According to
expressions 5.3’ and 5.4’,

\[ \min_{lb}(c_g \uparrow) = \max(\min_{lb}(d \uparrow, c_g) + \min_{d,c_g} \uparrow, \min_{lb}(e \uparrow, c_g) + \min_{d,c_g} \uparrow) = 13.3; \]

\[ \min_{ub}(c_f \uparrow) = \max(\min_{ub}(d \uparrow, c_g) + \max_{d,c_g} \uparrow, \min_{ub}(e \uparrow, c_g) + \max_{d,c_g} \uparrow) = 20. \]

For \( c_h \), the input arrival times are: \( \min_{lb}(e \uparrow, c_h) = 7.7, \min_{ub}(e \uparrow, c_h) = 10.7, \min_{lb}(f \uparrow, c_h) = 5.3, \min_{ub}(f \uparrow, c_h) = 6.7. \) According to expressions 5.3’ and 5.4’,

\[ \min_{lb}(c_h \uparrow) = \max(\min_{lb}(f \uparrow, c_h) + \min_{d,c_h} \uparrow, \min_{lb}(e \uparrow, c_h) + \min_{d,c_h} \uparrow) = 12; \]

\[ \min_{ub}(c_h \uparrow) = \max(\min_{ub}(f \uparrow, c_h) + \max_{d,c_h} \uparrow, \min_{ub}(e \uparrow, c_h) + \max_{d,c_h} \uparrow) = 15.4. \]

The lower and upper bounds of arrival times in the valid data cycle by max STA are equal to those by min STA, except for \( c_h \), where

\[ \max_{lb}(c_h \uparrow) = \max(\max_{lb}(e \uparrow, c_h) + \min_{d,c_h} \uparrow, \max_{lb}(f \uparrow, c_h) + \min_{d,c_h} \uparrow) = 14.4, \] and

\[ \max_{ub}(c_h \uparrow) = \max(\max_{ub}(e \uparrow, c_h) + \max_{d,c_h} \uparrow, \max_{ub}(f \uparrow, c_h) + \max_{d,c_h} \uparrow) = 19.4. \]

This example shows how to calculate the arrival times by min and max STA. When arrival times are computed, the required times and slack times of each signal in the data path can also be derived, using standard STA techniques [47]. Critical paths are the paths in a circuit where every signal in the path has 0 slack times.
5.5.3 Min and max critical paths

Two critical paths generated from the min and max STA calculations are especially important: the min critical path, which is the critical path by min STA with lower bounds of the the propagation and wire delays, and the max critical path, which is the critical path by max STA assuming upper bounds of those delays.

The delay of the min critical path is supposed to be the lower bound of a data path’s true delays with all possible value-dependent data flows, such as early propagation and minimum propagation delays of a DRFM. On the other hand, the delay of the max critical path represents the upper bound of a data path’s true delays, where every input transition is waited for the output’s evaluation and propagated with its maximum PD. As a result, the delay of any path in a circuit falls into the interval between the delays of min critical path and max critical path.

5.5.4 False path problems

False paths problems exist in STA techniques. Recall that the min critical path is derived by assuming that signals are early propagated with their lower bound delays. If these assumptions cannot be supported by any input vector, i.e., the min critical path cannot be sensitised, then the delay of min critical path is not the lower bound a data path’s true delays but an underestimation. On the other hand, the delay of the max critical path
might be an overestimation of the upper bound a data path’s true delays.

Formally, the paths that cannot be sensitised under any input vector are called false paths [33]. In this section, a sensitisation criterion is presented for the partially acknowledged data path. Suppose a min (max) critical path in a data path is $l = (i_1, \ldots, i_n, i_{n+1}, \ldots i_N)$, where $i_1$ is a primary input, $i_N$ is a primary output and $i_{n+1}$ is the output signal of a DRFM that $i_n$ fans into directly, $\forall n : 1 \leq n \leq N - 1$ and $n, N \in Z^+$. The sensitisation criterion of $l$ is determined by sensitisation criterion 5.5, where the influence of wire delays is not considered.

**Criterion 5.5** A min (max) critical path $l$ is sensitised under an input vector $v$ if $i_n$ dominates $i_{n+1}$, and $d_{i_n, i_{n+1}}$ used in calculating the arrival times by the min (max) STA is $\text{min}_d d_{i_n, i_{n+1}}$ ($\text{max}_d d_{i_n, i_{n+1}}$), $\forall n : 1 \leq n \leq N - 1$.

**Example 5.3** The min and max critical paths, $l_{\text{min}}$ and $l_{\text{max}}$, of the data path in example 5.2 in the valid data cycle are $(b, d, g)$ with the delays of 13.3 and 20, respectively. Analyse if $l_{\text{min}}$ and $l_{\text{max}}$ belong to false paths by the sensitisation criterion 5.5.

For $l_{\text{min}}$ to be a true path, $d_{b,c,d}$ should be $\text{min}_d d_{b,c,d}$, which requires that $b^1$ and $d^1$ are asserted in a valid data cycle, according to the results of Example 4.3.2. In addition, $d_{d,c,g}$ should be $\text{min}_d d_{d,c,g}$, which requires that $d^0$ and $g^0$ are asserted in a valid data cycle. According to the DR encoding, $d^0$ and $d^1$ cannot be asserted in one valid data cycle. Therefore, the sensitisation criterion 5.5 cannot be satisfied and $l_{\text{min}}$ is a false path. This false path is marked out by dash in Figure 5.3, as well as the inconsistent signal values.
Figure 5.3: False path example when using min STA

For $l_{\text{max}}$ to be a true path, $d_{b,c_d}$ should be $\max d_{b,c_d}$, which requires that $b^1$ and $d^0$ are asserted in a valid data cycle. This implies that $a^0$ and $a^{1}$ are also asserted in the valid data cycle. Furthermore, $d_{d,c_g}$ should be $\max d_{d,c_g}$, which requires that $d^0$ and $g^1$ are asserted in a valid data cycle. This implies that $e^1$ and $c^1$ are also asserted in this valid data cycle. With this reasoning, there exist a $v$ of $a = 0, b = 1$ and $c = 1$ such that all propagation delays of the DRFMs on the max critical path take their upper bound delays and all signal values are consistent. In addition, with this input vector, $b$ dominates $c_d$ and $d$ dominates $c_g$. Therefore, $l_{\text{max}}$ is a true path.

From this small example, it can be seen that the false path analysis can be a complicated problem. Furthermore, false path problems involve subtlety where for example static and dynamic sensitisation techniques can give different analysis results. For a thorough discussion on false paths, book [33] is recommended.
5.6 Summary

This chapter discusses the overall data flows of an asynchronous data path. Influencing factors on the data flows, such as PA and logic functions, are characterised by the concept of dominating inputs. Determination criterion of dominating inputs are defined, and the relation between dominating inputs and early propagation discussed. Two techniques are discussed to study the performance of an asynchronous data path: the input-dependent timing simulation and the min-max STA. With min-max STA, the critical paths can be derived. In addition, the determination criteria of false paths are introduced.
Chapter 6

Synthesis and optimisation of asynchronous data paths using partial acknowledgement

This chapter discusses the problem of synthesising and optimising asynchronous data paths using partial acknowledgement. The problem is formulated within the framework of mathematical programming (MP) and solved with several algorithms where a MP solver can be invoked. These algorithms reflect our thoughts in handling complex optimisation problems through the synergy of solver configuration, leverage by the user specified knowledge and problem decomposition.

The main procedures in synthesising and optimising asynchronous data paths form a design flow. A group of industrial level benchmarks are syn-
thesised and optimised according to the flow. The results are presented in tables showing the running times of the algorithms as well as the area and delay improvements of these benchmarks.

6.1 Problem description

The synthesis of asynchronous data paths converts a boolean network to an isomorphic DRFM netlist, where each non-primary input node in the network is mapped to a DRFM of the same function. During synthesis, the optimisation of the DRFM netlist is considered for either the area or performance in the synthesis procedure. Further, each input and internal signal in the synthesised DRFM netlist must be partially acknowledged in both the valid data and null cycles.

We formulate the synthesis and optimisation problem within the MP framework and solve the problem with the help of a MP solver. Compared with the previous formulations by the unate and binate covering problems [73], MP enables a more fine-grained description of the problem where the set of acknowledgeable phases expands from $\{*,-\}$ to $\{\uparrow,\downarrow,*,-\}$. In addition, the precision of the objective is improved by counting the overheads of all the DRFMs in a netlist rather than the input-complete overheads as in [73]. With all these extra demands, the complexity of the problem increases.

MP, a powerful tool in the Operational Research (OR), has very wide
applications in war, economics, and engineering. A formulation with MP usually includes variables, constraints and objectives. It is natural to link our problem with MP: the partial acknowledgement relation between the signals is encoded by variables, regulated by constraints, and solved for a particular objective. Section 6.1.1 gives the formal description of the synthesis and optimisation problem followed by an example in 6.1.2.

6.1.1 Formal description of the problem by MP

Given the boolean network $B$ where $i$ is a non-primary output node and $j$ is a non-primary input node in $B$, $DirectFanout(i)$ is the set of $i$'s direct fan-outs and $DirectFanin(j)$ is the set of $j$'s direct fan-ins.

The synthesis and optimisation problem is formally described as a mapping procedure from $B$ to a DRFM netlist $C$, where every $j$ in $B$ is mapped to a DRFM $J$ with the same functionality as that of $j$. The problem is formulated as an MP with the variables, constraints and objectives as defined in the following three subsections.

6.1.1.1 The variables

The fundamental variables in the MP formulation are the three-dimensional variables defined as:

$$G(j, i, p_i), \forall j \in B, i \in DirectFanin(j), p_i \in \{\uparrow, \downarrow, *, -\}, \quad V1$$

where $p_i$ is the transition phase of $I$, the DRFM of $i$.

$G(j, i, p_i)$ determines how $J$ acknowledges its direct fan-in $I$ in circuit $C$:
\[ G(j, i, p_i) = \begin{cases} 
1 & \text{if } J \text{ acknowledges } I \text{ for } p_i \\
0 & \text{if } J \text{ does not acknowledge } I \text{ for } p_i 
\end{cases} \]

The second class of variables, \( A(j) \), defines the areas of DRFMs \( J \):
\[
A(j) = \text{sum}(p \in p_i^{\text{DirectFanin}(j)}; (\prod_{i \in \text{DirectFanin}(j)} G(j, i, p_i)) \cdot J(p)), \forall j \in B, \\
\text{V2}
\]

where \( p \) is an ordered \(|\text{DirectFanin}(j)|\)-tuple of \( p_i \) and \( J(p) \) is \( J \)'s area when \( J \) acknowledges its inputs corresponding to \( p \).

As an example, when \( j \) has two direct fan-ins of \( ip1 \) and \( ip2 \), \( A(j) \) becomes:
\[
\text{sum}((p_{ip1}, p_{ip2}) \in \{\uparrow, \downarrow, *, -\} \times \{\uparrow, \downarrow, *, -\}, G(j, ip1, p_{ip1}) \cdot G(j, ip2, p_{ip2}) \cdot J(p_{ip1}, p_{ip2})).
\]

### 6.1.1.2 The constraints

The first class of constraints requires that each non-primary input DRFM \( J \) in \( C \) partially acknowledges its direct inputs for only one type of transition phase:
\[
\text{sum}(p_i \in \{\uparrow, \downarrow, *, -\}, G(j, i, p_i)) = 1, \forall j \in B, i \in \text{DirectFanin}(j). \quad \text{C1}
\]

The second class of constraints require that each non-primary output DRFM \( J \) in \( C \) is partially acknowledged by at least one of its direct outputs, in both the valid data cycle (C2.1) and the null cycle (C2.2):
\[
\text{sum}(j \in \text{DirectFanout}(i), G(j, i, \uparrow) + G(j, i, *)) \geq 1, \forall i \in B, \quad \text{C2.1}
\]
\[
\text{sum}(j \in \text{DirectFanout}(i), G(j, i, \downarrow) + G(j, i, *)) \geq 1, \forall i \in B. \quad \text{C2.2}
\]
6.1.1.3 The objectives

An objective is a minimisation or maximisation of a function in terms of the MP’s variables. Different objectives can be defined according to the user’s demands. For example, to minimise the overall area of $C$, the objective is:

\[
\text{min}[\text{Area}(C)], \quad \text{O.area}
\]

where $\text{Area}(C) \equiv \text{sum}(j \in B, \text{Area}(j))$.

As another example in performance optimization, to maximise the partial acknowledgement of the “neither” phases by the DRFMs on the critical paths in $C$, the objective is:

\[
\text{max}[\text{sum}(j \in L, G(j, i \in \text{DirectFanin}(j), -))], \quad \text{O.perf1}
\]

where $L$ is the set of nodes whose DRFMs are on the critical paths.

Further, a DRFM netlist can be optimised for one data cycle against the other. For instance,

\[
\text{min}[\text{sum}(j \in L, G(j, i \in \text{DirectFanin}(j), \uparrow) + G(j, i \in \text{DirectFanin}(j), \ast))], \quad \text{O.perf2}
\]

favors the valid data cycle rather than the null cycle. Optimisation with O.perf2 can lead to a skewed circuit [16].

6.1.2 A formulation example

We synthesise ISCAS benchmark circuit C17 (Figure 6.1) and optimise it for the overall area and the delay in the valid data cycle.

The set of the non-primary inputs \{$g_1, g_2, g_3, g_4, g_5, g_6$\}, the set of the non-primary outputs \{$i_1, i_2, i_3, i_4, i_5, g_1, g_2, g_3, g_4$\}, and the set of nodes on
the critical path is \{i_2, g_2, g_3, g_6\}. The area minimisation includes V1,V2,C1,C2
and O.area whereas the performance optimisation includes V1,C1,C2 and
O.perf2. The variables, constraints and objectives used in the formulations
are listed below in detail:

V1:

\begin{align*}
G(g_1, i_1, \uparrow), & \quad G(g_1, i_1, \downarrow), & \quad G(g_1, i_1, \star), & \quad G(g_1, i_1, -), \\
G(g_1, i_2, \uparrow), & \quad G(g_1, i_2, \downarrow), & \quad G(g_1, i_2, \star), & \quad G(g_1, i_2, -), \\
& \quad \cdots & \quad \cdots & \quad \cdots & \quad \cdots \\
G(g_6, g_3, \uparrow), & \quad G(g_6, g_3, \downarrow), & \quad G(g_6, g_3, \star), & \quad G(g_6, g_3, -), \\
G(g_6, g_4, \uparrow), & \quad G(g_6, g_4, \downarrow), & \quad G(g_6, g_4, \star), & \quad G(g_6, g_4, -).
\end{align*}
V2:

\[ A(g1) = G(g1, i1, \uparrow) \cdot G(g1, i2, \uparrow) \cdot G1(\uparrow, \uparrow) + G(g1, i1, \uparrow) \cdot G(g1, i2, \downarrow) \cdot G1(\uparrow, \downarrow) + G(g1, i1, \uparrow) \cdot G(g1, i2, \ast) \cdot G1(\uparrow, \ast) + G(g1, i1, \uparrow) \cdot G(g1, i2, -) \cdot G1(\uparrow, -) + G(g1, i1, \downarrow) \cdot G(g1, i2, \uparrow) \cdot G1(\downarrow, \uparrow) + G(g1, i1, \downarrow) \cdot G(g1, i2, \downarrow) \cdot G1(\downarrow, \downarrow) + G(g1, i1, \downarrow) \cdot G(g1, i2, \ast) \cdot G1(\downarrow, \ast) + G(g1, i1, \downarrow) \cdot G(g1, i2, -) \cdot G1(\downarrow, -) + G(g1, i1, \ast) \cdot G(g1, i2, \uparrow) \cdot G1(\ast, \uparrow) + G(g1, i1, \ast) \cdot G(g1, i2, \downarrow) \cdot G1(\ast, \downarrow) + G(g1, i1, \ast) \cdot G(g1, i2, \ast) \cdot G1(\ast, \ast) + G(g1, i1, \ast) \cdot G(g1, i2, -) \cdot G1(\ast, -) + G(g1, i1, -) \cdot G(g1, i2, \uparrow) \cdot G1(-, \uparrow) + G(g1, i1, -) \cdot G(g1, i2, \downarrow) \cdot G1(-, \downarrow) + G(g1, i1, -) \cdot G(g1, i2, \ast) \cdot G1(-, \ast) + G(g1, i1, -) \cdot G(g1, i2, -) \cdot G1(-, -), \]

and \( A(g2), \ldots, A(g6) \) are defined in similar ways.

C1:

\[ G(g1, i1, \uparrow) + G(g1, i1, \downarrow) + G(g1, i1, \ast) + G(g1, i1, -) = 1, \]
\[ G(g1, i2, \uparrow) + G(g1, i2, \downarrow) + G(g1, i2, \ast) + G(g1, i2, -) = 1, \]
\[ \ldots \]
\[ G(g6, g3, \uparrow) + G(g6, g3, \downarrow) + G(g6, g3, \ast) + G(g6, g3, -) = 1, \]
\[ G(g6, g4, \uparrow) + G(g6, g4, \downarrow) + G(g6, g4, \ast) + G(g6, g4, -) = 1. \]
\[ G(g_1, i_1, \uparrow) + G(g_1, i_1, *) \geq 1, \]
\[ G(g_1, i_2, \uparrow) + G(g_1, i_2, *) + G(g_2, i_2, \uparrow) + G(g_2, i_2, *) \geq 1, \]
\[ \ldots, \]
\[ G(g_6, i_4, \uparrow) + G(g_6, i_4, *) \geq 1. \]

C2.2:

\[ G(g_1, i_1, \downarrow) + G(g_1, i_1, *) \geq 1, \]
\[ G(g_1, i_2, \downarrow) + G(g_1, i_2, *) + G(g_2, i_2, \downarrow) + G(g_2, i_2, *) \geq 1, \]
\[ \ldots \]
\[ G(g_6, i_4, \downarrow) + G(g_6, i_4, *) \geq 1. \]

O.area:

\[ \min[A(g_1) + A(g_2) + \ldots + A(g_6)]. \]

O.perf2:

\[ \min[G(g_2, i_2, \uparrow) + G(g_2, i_3, \uparrow) + G(g_3, g_2, \uparrow) + G(g_3, i_4, \uparrow) + G(g_6, g_3, \uparrow) + G(g_6, g_4, \uparrow) + G(g_2, i_2, \downarrow) + G(g_2, i_3, \downarrow) + G(g_3, g_2, \downarrow) + G(g_3, i_4, \downarrow) + G(g_6, g_3, \downarrow) + G(g_6, g_4, \downarrow)]. \]
6.2 Practical considerations of solving the problem

The variables V1 in the problem formulation are binary variables (also known as 0-1 variables). Therefore, the problem lends itself to mixed integer programming. Further, if a non-linear objective function such as O_area is used, the problem can be approached by a mixed non-linear integer programming (MINLP). MINLP is NP-hard in general [36], which combines two hard problems: integer programming (also known as combinatorial optimisation) and nonlinearity. Two philosophies exist when solving an optimisation problem like MINLP: global optimisation (GO) and local search heuristics. These will be discussed in the next section.

In a boolean network with $n$ non-primary input nodes and an average fan-in number of $k$, the size of V1 and V2 are $4 \cdot n \cdot k$ and $n$, respectively. The number of the quadratic terms in V2 increases exponentially with $k$. In addition, C1 and C2 have $n \cdot k$ and $2 \cdot n$ constraints, respectively.

6.2.1 Global optimisation versus local search heuristics

GO explores the absolute optima of a problem in the presence of multiple local optima. It often involves a global scope search of the problem domain. There are several important strategies in GO including naive approaches [44], complete (enumerative) search [43] and Branch-and-Bound (B&B) algorithms [43].
In contrast to GO, local search heuristics explore the locally optimal solutions with varying qualities. Local searches often start from an initial feasible solution and try to find a better one by applying successive modifications. *Espresso* [45] is a famous and arguably the most successful program for logic minimisation using local searches. Common local search heuristic methods include evolution strategies [66], simulated annealing [41], and tabu search [68].

The solver used in the algorithms to solve the synthesis and optimisation problem is called BARON [52]. BARON supports both GO and local search heuristics. The GO engine in Baron is the “Branch-and-Reduce” (B&R) algorithm, a variant of the B&B algorithm. The local search heuristics in BARON are available at different stages of a solution process. The technical details of BARON can be found in [63].

### 6.2.2 Factors influencing the computation time

The topology of a boolean network impacts upon the problem complexity and thus the running time. To help explain this influence, we will use two prototype networks: a converging network and a forking network. The converging network forms a tree where each node fans out to only one direct output. As a comparison, a node in the forking network will have multiple direct fan-outs. Figure 6.2 shows two examples of the prototypes. Note that real networks have both forking and converging sub-networks.

For a converging network, there is no other choice but to acknowledge a
Figure 6.2: Converging and forking networks

node by its only direct fan-out for both the rising and falling phases. Therefore, it has a nearly constant run time as shown in Figure 6.3 (converging circuit with GO) when solved by the GO engine of BARON. In fact, what makes the converging networks straightforward to solve is analogous to the “row dominance” simplification in solving a unate covering problem [73]. Conversely, forking circuits lead to situations known as “cyclic cores” in the B&R engine. As a result, solving forking networks using the GO engine exhibits an exponential increase in time (forking circuit with GO in Figure 6.3), and thus there are no result for a network with more than six gates within 30 minutes.

Strictness of the constraints in a MP influences the running time. For example, if C2 is relaxed to acknowledge a node by only one of its direct fan-outs, the running time becomes less exponential and a gate number solvable in 30 minutes increases to 10 (forking circuit with GO/CR (constraint
Figure 6.3: Running times of the problem with different influencing factors. The results is from BARON’s GO engine run on a windows machine with a 1.67GHz processor and 512MB RAM. The forking networks have a direct fan-in number of 2.
relaxation) in Figure 6.3). Constraint relaxation will be further discussed in section 6.2.3.

The scope of the problem considered in the optimisation is another factor. The running time is alleviated (forking circuit with level optimisation) for the forking networks with the level-wise optimisation algorithm, a method in problem decomposition (6.3.2). This is demonstrated in Figure 6.3 (forking circuit with level optimisation), where a doubling of gate number from 8 to 16 incurs only a linear run time increase (from 12.7 to 23.8 seconds). As a comparison, increasing gate numbers from 8 to 13 when using GO/CR without level optimisation increases the run time from 85.6 seconds to 2052 seconds.

6.2.3 Leverage by user-specified knowledge

The user’s knowledge of the problem is very valuable when solving a practical MP. Applications of knowledge, like those in 6.2.3.1 and 6.2.3.2, reduce the search space and therefore the computation time without any degradation in the results. Some, like 6.2.3.3 and 6.2.3.4, trade the fine level of the formulation for computation time. Others guide the B&B engine in more meaningful search directions, such as 6.2.3.5.

6.2.3.1 Nodes with one direct fan-out

When node $j$ is the only direct fan-out of $i$, $j$ must acknowledge $i$ for both the rising and falling phase transitions, i.e., $G(j, i, \ast) = 1$ and $G(j, i, \dagger) =$
\[ G(j, i, \downarrow) = G(j, i, -) = 0. \] \(A(j)\) is consequently simplified.

### 6.2.3.2 Nodes with one direct fan-in

A node with one direct fan-in corresponds to a DRFM of a buffering or inverting function, which automatically acknowledges its input for both the rising and falling phases. Supposing node \(i\) is the only direct fan-in of \(j\), we have \(G(j, i, \star) = 1\) and \(G(j, i, \uparrow) = G(j, i, \downarrow) = G(j, i, -) = 0.\)

### 6.2.3.3 Binding the rising and falling phase acknowledgement

If the rising (falling) phase of a signal is always acknowledged by a DRFM which acknowledges the falling (rising) phase of the signal as well, the set of the acknowledgeable phases is shrunk to \(\{\star, -\}\). As a result, the sizes of both \(V1\) and \(V2\) reduce by half, and the constraints are simplified too. For example, C2.1 and C2.2 can be merged into one:

\[
\text{sum}(j \in \text{DirectFanout}(i), G(j, i, \star)) \geq 1, \forall i \in B.
\]

The reduction of variables and constraints narrow the search space; however, the fine-grained formulation is lost by the reduction. The binding was considered in previous work [73] where an optimisation tool called Indie [53] was implemented to test the results.

### 6.2.3.4 Constraint relaxation

The places to acknowledge a signal in the DRFM netlist can be specified by a number rather than the “at least one” condition in C2.1 and C2.2. For
example, C2.1 and C2.2 become:

\[\text{sum}(j \in \text{DirectFanout}(i), G(j, i, \uparrow) + G(j, i, *)) = 1, \forall i \in B, \quad \text{C2.1'}\]

\[\text{sum}(j \in \text{DirectFanout}(i), G(j, i, \downarrow) + G(j, i, *)) = 1, \forall i \in B, \quad \text{C2.2'}\]

if only one place is required for the partial acknowledgement. CR reduces the search space and improves the computation time, as in 6.2.2.

### 6.2.3.5 Nodes with branching priorities

We can guide the B&B procedure by assigning priorities to some variables. The nodes with the highest priorities are considered before other nodes in the B&B for their possible values. A solver guided with meaningful directions no longer searches the problem space blindly. For example, the nodes on the critical paths can be given the highest priorities for branching in performance optimisation.

### 6.3 Methods of solving the problem

Purely solving the problem with the GO engine can be formidable, as indicated by the run-time experiments in 6.2.2. As another example, ISCAS benchmark C432 (with 318 nodes) gives no solution after executing the solver for more than 40 hours using GO on the aforementioned machine. Therefore, we will explore methods that can reduce run time with acceptable results. These methods fall into two general approaches: the first is time limited assorted heuristics and the second is problem decomposition.
6.3.1 Time limited assorted heuristics (pseudo-global optimisation)

The time limited assorted heuristics approach optimises the problem within a period of time. The approach consists of two phases: the preprocessing stage and the B&B procedure.

In the preprocessing stage, multiple local searches are run from randomly generated starting points. In Baron, the option numloc is used to adjust the number of the preprocessing searches. In the B&B procedure phase, the heuristics in the B&R engine are configured with certain options. Further, the B&B procedure is guided by the problem specific knowledge. The length of the B&B procedure is controlled by the maximum number of B&B iterations (maxiter) or the maximum CPU time (maxtime). The main tunable heuristics are:

- the node selection rule when exploring the search tree (nodesel) consists of best bound, last in first out, and minimum infeasibility.

- the branching variable selection strategy (brvarstra) includes longest edge, largest violation and BARON’s dynamic strategy.

- the branching point selection strategy (brptstra) includes $\omega$-branching, bisection-branching, and convex combination of the $\omega$-and bisection-branching.

- the local search option for upper bounding (dolocal) configures the local
search for upper bounding every $n$ iterations using BARON’s dynamic local search decision rule.

Time limited assorted heuristics adopt a “pseudo-global” manner to optimise the problem. In the preprocessing stage, an extensive amount of local samples are tested for feasible solutions. These local searches, or more precisely, the “globalized” extensions of local searches [51], are partial heuristics which nevertheless often bring success in practice. Indeed we hope to find a feasible solution in the preprocessing stage as otherwise it is very difficult for the later B&B to report one, according to our experience. Note however that even this random sampling is not trivial for a large circuit. The best feasible solution found in the preprocessing stage, if any, is passed to the B&B as the upper bound. During the B&B, a true GO is formidable as the convergence of the upper and lower bounds takes a long time. Therefore, we limit the solution time of the B&B and give priorities to certain branching nodes.

The time limited assorted heuristics can be used in two different settings: pure area minimisation or joint area-performance optimisation. The area minimisation is straightforward and comprises the above 2 stages with $O_{area}$ as the objective. Variables $G(j, i, p)$ are given higher priority during the B&B procedure when $j$ is a node on the critical paths. For joint area-performance optimisation, the partial acknowledgement by the nodes on the critical paths is first minimised according to algorithm 1 before the overall area of the DRFM netlist is minimised. In algorithm 1, $G(j, i, p)$ is no longer a variable after $G(j, i, -)$ is assigned a specific value, as reflected by assigning the value
Algorithm 1 joint area-performance optimisation using time limited assorted heuristics

Inputs: Network $B$, set $L$ of the nodes on the critical paths;
Output: $G - file$;

$\forall i \in \text{primary inputs and internal signals of } B$
  if $(0 < |\text{DirectFanout}(i) \cap L| < |\text{DirectFanout}(i)|)$ then
    $\forall j \in \text{DirectFanout}(i) : j \in L$
      $G(j, i, -) := 1$;
      $G(j, i, p).\text{nonvar} := 1$;
  }
$\forall n \in L$
  $G(n, \text{DirectFanin}(n), p).\text{priority} := 1$;

solve the problem for area minimisation;
write the $G$ variables to the $G - file$;

1 to $G(j, i, p).\text{nonvar}$, the non-variable attribute of a variable.

6.3.2 Problem decomposition

Problem decomposition partitions the problem into the smaller sub-problems and then solves one sub-problem at a time. This approach is successful as it only considers the variables and constraints in a certain window, either in space or time, and optimises a sub-objective in that window. Note that finding a good partition scheme is not trivial.

Two decomposition methods are discussed: level-wise optimisation and farthest acknowledgement. Both methods use windows on the network levels generated by algorithm 2. The algorithm annotates each node with a level, where any non-primary input node on level $i$ has at least one direct fan-in on level $i - 1$. 

103
Algorithm 2 An algorithm to level the nodes in a boolean network

Input: a boolean network $B$, the primary input nodes of $B(PI(B))$, the primary output nodes of $B(PO(B))$;
Outputs: $n$.level, the level of each node $n \in B$, and $L_{\text{max}}$, the maximum level of the circuit;
$n$.level := 0, $\forall n \in PI(B)$;
$n$.level := $-1$, $\forall n \notin PI(B)$;
$\forall n \in PO(B)$
  level($n$);
$\forall n \in PI(B)$
  $n$.level := $\min(j$.level $| j \in \text{DirectFanout}(n)) - 1$;
$L_{\text{max}} := \max(n$.level $| n \in B)$;
return $n$.level, $\forall n \in PI(B)$ and $L_{\text{max}}$;
Sub-procedure level($n$):
  level($n$);
  if ($n$.level $= -1$)
    $n$.level := $\max(\text{level}(i) | i \in \text{DirectFanin}(n)) + 1$;

6.3.2.1 Level-wise optimisation

Level-wise optimisation (algorithm 3) handles the network from level 0 to $L_{\text{max}}$, the maximum level of the network, where a sub-problem is formed at each level of the circuit. At level $i - 1$, only the unacknowledged nodes ($UnackedNodes$) up to level $i - 1$ are considered for partial acknowledgement by the nodes on level $i$ ($NodesToAck$). In fact, only a subset of $UnackedNodes$, $NodesToBeAcked$, which have direct fan-outs in $NodesToAck$, can be acknowledged at this time. In addition, the sub-problem optimises only the nodes in $NodesToAck$ for their minimal area.

Figure 6.4 illustrates the optimisation of a sub-problem $i - 1$, where each dashed vertical line represents a level in the sub-network. $UnackedNodes$
Algorithm 3 Level-wise optimisation
Input: Network $B$ after algorithm 2, where $L_{\text{max}}$ is the maximum level of $B$;
Output: $G$ — file;
$\text{nodes}(l) :=$ the set of nodes on level $l$;
Initialize $\text{UnackedNodes, \text{NodesToBeAcked, NodesToAck}}$ to null sets;
for ($l = 0; l < L_{\text{max}}; l++$) {
    $\text{UnackedNodes} := \text{UnackedNodes} \cup \text{Nodes}(l);$  
    $\text{NodesToAck} := \text{nodes}(l + 1);$  
    $\text{NodesToBeAcked} := \{ i \in \text{UnackedNodes} \mid \exists \text{DirectFanout}(i) \in \text{NodesToAck} \};$
    $\text{sub} – G := \{ G(j, i, p) \mid j \in \text{NodesToAck}, i \in (\text{DirectFanin}(j) \cap \text{NodesToBeAcked}) \};$
    $\text{sub} – A := \{ A(j) \mid j \in \text{NodesToAck} \};$
    $\text{sub} – \text{variables} := \text{sub} – G \cup \text{sub} – A;$
    $\text{sub} – C1 := \{ C1 \mid j \in \text{NodesToAck}, i \in (\text{DirectFanin}(j) \cap \text{NodesToBeAcked}) \};$
    $\text{sub} – C2 := \{ C2 \mid i \in \text{NodesToBeAcked}, j \in (\text{DirectFanout}(i) \cap \text{NodesToAck}) \};$
    $\text{sub} – \text{constraints} := \text{sub} – C1 \cup \text{sub} – C2;$
    $\text{sub} – \text{objective} := \text{O\_area, where } j \in \text{NodesToAck};$
    $\text{sub} – \text{problem} := \{ \text{sub} – \text{variables, sub} – \text{constraints, sub} – \text{objective} \};$
    solve the $\text{sub} – \text{problem};$
    if $\text{sub} – \text{problem.status} = \text{OK} \{$
        $\text{UnackedNodes} := \text{UnackedNodes} – \text{NodesToBeAcked};$
    $\forall i \in \text{NodesToBeAcked and } j \in \text{DirectFanout}(i)\{$
        if ($j \notin \text{NodesToAck}$) then{
            $G(j, i, p).\text{nonvar} := 1;$
            $G(j, i, –) := 1;$
            $G(j, i, \uparrow) = G(j, i, \downarrow) = G(j, i, \bullet) := 0;\}$
    $\}$}
includes all the non-primary output nodes on level \( i - 1 \) \({\{3,4}\}\) and the unacknowledged nodes left from levels lower than \( i - 1 \) \({\{2}\}\). \( NodesToAck \) contains the nodes on level \( i \) except the primary inputs \({\{5,6}\}\). In addition, \( NodesToBeAcked = \{2, 4\} \). Based on these sets, a sub-problem is formulated by algorithm 3 including the sub-variables, the sub-constraints and the sub-objectives. For example, \( sub - G \), the subset of variables \( G(j, i, p) \), includes only those with \( j \) in \( NodesToAck \) and \( i \) in \( NodesToBeAcked \), i.e., \( G(5, 4, p), G(6, 4, p) \) and \( G(6, 2, p) \) in Figure 6.4.

After the sub-problem is solved, the nodes in \( NodesToBeAcked \) are excluded from \( UnackedNodes \) and are not required to be further acknowledged by their direct fan-outs with higher levels than \( i \). \( G(j, i, p).nonvar \) is therefore set to 1. For example, \( G(7, 4, p).nonvar = 1 \) in Figure 6.4.


**Algorithm 4** algorithm for the farthest acknowledgement

```
Input: Network $B$ after algorithm 2;
Output: $G - file$;
$\forall i \in$ primary inputs and internal signals of $B${
    $i.ack := 0$;
    $i.farthest := 0$
$\forall j \in DirectFanout(i)${
    if ($j.level > i.farthest$) then
        $i.farthest := j.level$
$\forall j \in DirectFanout(i)${
    if ($j.level = i.farthest$ and $i.ack := 0$) then {
        $G(j,i,\star) := 1$;
        $G(j,i,p).nonvar := 1$;
        $i.ack := 1$;
    } else {
        $G(j,i,-) := 1$;
        $G(j,i,p).nonvar := 1$;
    }
```

6.3.3.2.2 Farthest acknowledgement

For *level-wise optimisation*, nodes are considered to be acknowledged only by their fan-outs with the lowest levels. This “nearest acknowledgement” stems from a belief that nodes tend to have most of their direct fan-outs on the closest levels. However, when the performance of the circuit is the major concern, we may want to acknowledge the nodes by their direct fan-outs with the farthest distance (in levels) since we want to put off the acknowledgement points as close as to the primary outputs. Algorithm 4, the *farthest acknowledgement*, implements such an idea.
6.4 Design flow and experimental results

6.4.1 Synthesis and optimisation flow

Figure 6.5 shows the overall synthesis and optimisation flow. The flow begins with the technology mapping of a boolean network which ensures all the DRFM s are available in the latter PA-mapping stages. After the technology mapping, the network is analyzed for the information needed in the synthesis and optimisation algorithms introduced above. Connection analysis produces the network’s connectivity matrix with fan-in and fan-out information of each node. Level analysis generates the network’s level information if level-wise optimisation and farthest acknowledgement algorithms are launched.

Critical path information needed for performance optimisation can be generated by the initial PA delay analysis of the initial DRFM netlist, which is mapped from the TM mapped network by the initial PA-mapping. The initial PA-mapping maps each non-primary input node j to a certain DRFM. Different initial PA-mappings cast a TM mapped network to different initial DRFM netlists, which in turn have different characteristics in partial acknowledgement. In the experiment, initial PA-mapping tentatively maps a network such that every non-primary input in the mapped netlist acknowledges all its direct input signals.

With the connectivity matrix, level and the critical path information to hand, the problem is formulated and solved according to the algorithms. If
Figure 6.5: The synthesis and optimisation flow
the problem is solvable, a $G$ file indicating how each node should be annotated with a certain DRFM, i.e., a list of $G(j, i, p)$ values, is produced as well as other information of the solution process.

PA-mapping then converts the TM mapped network to the final DRFM netlist, with the help of the $G$ file. Finally the timing analysis of the final DRFM netlist is performed, which includes both the PA min-delay analysis for the min-delay and the PA max-delay analysis for the max-delay.

Two library sources are used in our experimental flow: the TM library for the technology mapping and the PA library for the PA delay analysis. Currently the TM library contains only simple gates with no more than two inputs, whereas the PA library includes all the possible DRFMs for each element in the TM library. In addition, the PA library is divided into the PA-min library used in the PA min-delay analysis and the PA-max library in the PA max-delay analysis. All of these libraries are organised in the .genlib format.

The synthesis and optimisation algorithms are implemented in AIMMS software [2] using its programming languages [27]. The other major steps in the flow (shaded by grey area in Figure 6.5) are implemented by C functions written in the framework of SIS, a system for sequential circuit synthesis [49] [12]. Table 6.1 lists these functions.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>circuit_netlist</td>
<td>connectivity analysis of a network</td>
</tr>
<tr>
<td>node_level</td>
<td>level analysis of a network</td>
</tr>
<tr>
<td>pa_setup</td>
<td>parse a $G$ file for the partial acknowledgement annotation</td>
</tr>
<tr>
<td>pa_mapping</td>
<td>map a network to DRFM netlist using a <em>PA library</em></td>
</tr>
<tr>
<td>delay_min_trace</td>
<td>min delay analysis of an partially acknowledged circuit</td>
</tr>
<tr>
<td>delay_max_trace</td>
<td>max delay analysis of an partially acknowledged circuit</td>
</tr>
</tbody>
</table>

Table 6.1: The functions implementing the major steps of the design flow

### 6.4.2 Experimental results

A series of ISCAS-85 circuits have been synthesised and optimised using the algorithms discussed in this chapter. These test-bench circuits are from industrial designs including a 7-channel interrupt controller (C432), ALUs (C880, C3540) and a $16 \times 16$ multiplier (C6288). The results of the algorithms are presented in Table 6.2 (area minimisation and joint area-performance optimisation using time limited assorted heuristics) and Table 6.3 (*level-wise optimisation* and *farthest acknowledgement*). All the algorithms were run on a Windows machine with a 1.67GHz processor and 512MB RAM.

The first two columns of Table 6.2 and Table 6.3 show the name of the ISCAS circuit and the number of primary inputs, primary outputs and internal nodes. Table 6.3 has an extra column “lev” (the third column) showing the $L_{\text{max}}$ of the circuits.

Table 6.2 includes three column categories: full acknowledgement, area minimisation, and joint area-performance optimisation. Full acknowledgement denotes the implementation of a DRFM netlist where each signal is ac-
knowned by all the direct outputs for both rising and falling phases. Area
minimisation and joint area-performance optimisation categories denote op-
timisations using time limited assorted heuristics. In these two optimisation
algorithms, numloc is set to 500 for C17, C432, C880, C1355 and C499, and
2000 for C1908, maxtime is set to 1800 seconds, nodesel is set to best bound,
brvarstra is set to longest edge and brptstra is set to bisection-branching.
3 sub-columns exist in each column category: area, min-delay (min-d) and
max-delay (max-d) of the synthesised DRFM netlist. In addition, run time
(rt) sub-columns under the area minimisation and joint area-performance op-
timisation categories show the corresponding run times. Note that the run
time is denoted by an X for a circuit if no feasible solutions can be produced
by the algorithms with numloc=2000 and maxtime=1800 seconds.

Table 6.3 has also three categories: full acknowledgement, level-wise optim
isation and farthest acknowledgement. The sub-columns in these categories
have the same meaning as those in table 6.2.

From the tables we can see that joint area-performance optimisation by
time limited assorted heuristics has the best results of joint optimisation for
both area and performance: it improves the area by an average factor of
1.267, min delay by 1.653 and max delay by 1.289. A pure area minimisa-
tion using time limited assorted heuristics exhibits slightly worse results for
performance optimisation compared of joint area-performance optimisation,
where the average min delay improvement factor is 1.613 and max delay
1.274.
those of the fully acknowledged circuits.

The table shows the problem size, running times, area, delays, and
heuristics methods for area minimization and joint area-performance opti-

Table 6.2: Benchmark circuits optimized with the time limited associated

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>C17</th>
<th>C486</th>
<th>C135</th>
<th>C399</th>
<th>C136</th>
<th>C363</th>
<th>C132</th>
<th>C497</th>
<th>C142</th>
</tr>
</thead>
<tbody>
<tr>
<td>C00</td>
<td>5</td>
<td>7</td>
<td>14</td>
<td>34</td>
<td>47</td>
<td>69</td>
<td>96</td>
<td>99</td>
<td>60</td>
</tr>
<tr>
<td>C01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C03</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Area

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>C17</th>
<th>C486</th>
<th>C135</th>
<th>C399</th>
<th>C136</th>
<th>C363</th>
<th>C132</th>
<th>C497</th>
<th>C142</th>
</tr>
</thead>
<tbody>
<tr>
<td>C00</td>
<td>5</td>
<td>7</td>
<td>14</td>
<td>34</td>
<td>47</td>
<td>69</td>
<td>96</td>
<td>99</td>
<td>60</td>
</tr>
<tr>
<td>C01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C03</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Delay

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>C17</th>
<th>C486</th>
<th>C135</th>
<th>C399</th>
<th>C136</th>
<th>C363</th>
<th>C132</th>
<th>C497</th>
<th>C142</th>
</tr>
</thead>
<tbody>
<tr>
<td>C00</td>
<td>5</td>
<td>7</td>
<td>14</td>
<td>34</td>
<td>47</td>
<td>69</td>
<td>96</td>
<td>99</td>
<td>60</td>
</tr>
<tr>
<td>C01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C03</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Power

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>C17</th>
<th>C486</th>
<th>C135</th>
<th>C399</th>
<th>C136</th>
<th>C363</th>
<th>C132</th>
<th>C497</th>
<th>C142</th>
</tr>
</thead>
<tbody>
<tr>
<td>C00</td>
<td>5</td>
<td>7</td>
<td>14</td>
<td>34</td>
<td>47</td>
<td>69</td>
<td>96</td>
<td>99</td>
<td>60</td>
</tr>
<tr>
<td>C01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C03</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 6.3: Benchmark circuits optimised with the problem decomposition approach. The table shows the problem size, running times, area, delays, and the area/delay ratios between the results using the proposed algorithms and those of the fully acknowledged circuits.

<table>
<thead>
<tr>
<th>benchmark name</th>
<th>#/ #n/ #m</th>
<th>full acknowledgement</th>
<th>level-wise optimisation</th>
<th>the farthest acknowledgement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>area</td>
<td>min.d</td>
<td>max.d</td>
</tr>
<tr>
<td>C17</td>
<td>5/2/6</td>
<td>4</td>
<td>120</td>
<td>24.7</td>
</tr>
<tr>
<td>C452</td>
<td>36/7/282</td>
<td>36</td>
<td>4246</td>
<td>265</td>
</tr>
<tr>
<td>C388</td>
<td>68/26/440</td>
<td>32</td>
<td>6028</td>
<td>219.3</td>
</tr>
<tr>
<td>C1355</td>
<td>41/32/552</td>
<td>31</td>
<td>10366</td>
<td>238</td>
</tr>
<tr>
<td>C499</td>
<td>41/32/567</td>
<td>31</td>
<td>8260</td>
<td>198.8</td>
</tr>
<tr>
<td>C1968</td>
<td>33/25/771</td>
<td>37</td>
<td>12340</td>
<td>322.3</td>
</tr>
<tr>
<td>C1540</td>
<td>50/22/1553</td>
<td>52</td>
<td>25340</td>
<td>417</td>
</tr>
<tr>
<td>C5315</td>
<td>178/123/2516</td>
<td>52</td>
<td>4160</td>
<td>334.3</td>
</tr>
<tr>
<td>C6288</td>
<td>32/32/2460</td>
<td>124</td>
<td>47045</td>
<td>1163.9</td>
</tr>
<tr>
<td>C7552</td>
<td>207/188/3860</td>
<td>42</td>
<td>62560</td>
<td>482</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td>209780</td>
<td>3644.8</td>
<td>6033.4</td>
</tr>
</tbody>
</table>

average percentage | 0.796 | 0.669 | 0.827 | 0.811 | 0.607 | 0.773 |
The running times for time limited assorted heuristics are large, proving that even “globalized” local sampling is not trivial for large circuits. As an example, the time taken for one local search increases from 0.02 seconds for C17 to 8.14 seconds for C1908. Further, the number of local searches required to find a feasible solution increases in proportion to the circuit size: 22 feasible solutions were reported during 500 local searches in C499 with area optimisation whereas none were reported for 2000 local searches in C3540.

For problem decomposition, the run times are reduced significantly: the average run time for the farthest acknowledgement is only 2 seconds and the longest run time is 12.5 seconds (C7552). The time reduction comes from the limitation of the search space, which however does not significantly degrade the area improvements of level-wise optimisation and the performance improvements of farthest acknowledgement. Level-wise optimisation shows only about 0.01 area improvement degradation whereas farthest acknowledgement slightly improves the max delay by the joint area-performance optimisation. On the other hand, the delay improvements using level-wise optimisation and the area improvements using farthest acknowledgement are reduced as a trade-off.

6.5 Summary

This chapter describes how partial acknowledgement is used to synthesise an asynchronous DRFM netlist and optimise it for both area and performance.
The synthesis and optimisation procedure is formulated by MP including variables and constraints which encode and regulate the partial acknowledgment relation between the circuit signals.

Different methods have been proposed to solve a problem with a particular objective. The methods show varying trade-offs in their running times, improvement ratios of the area, max delay and min delay. The overall results of the selected benchmarks demonstrate that it is effective to tune a circuit’s performance with partial acknowledgement.
Chapter 7

Conclusion and future work

This chapter conclude our work in this thesis by highlighting the problem we try to solve and summarising the ideas, steps and approaches in solving this problem. Furthermore, possible future research areas related with our work are discussed.

7.1 Summary of the problem and its solution process

This thesis discusses design, synthesis and optimisation of asynchronous data paths using DI encoding. The problems it tries to solve is how to reduce the penalties on area, performance and power consumption of a robust asynchronous implementation by design flows such as NCL-D. This is a real and important problem as robust design techniques against parametric variations
are urgently needed by the semiconductor industry when the technology goes into deep sub-micron, and performance of products must not be significantly sacrificed because of this robustness.

To solve this problem, design flows are called for where the synergy and contributions from library design, analysis, synthesis and optimisation techniques are strongly required. Above all, we need a fundamental concept supporting all levels of the flow and this concept must clearly capture our design intent - reducing the costs of robust asynchronous implementations.

Reducing the cost of robustness means simplifying the indication or acknowledgement relations between signals but not removing them. In the literature, the weakest indication in asynchronous data path is believed to be utilised in Martin’s programme transformations, and demonstrated by the example of a delicately designed adder. We are inspired by Martin’s simplification methods in defining our concepts about Partial Acknowledgement (PA). PA says that a transition of DR variable $a$, regardless of the data value involved, must cause a transition of another variable which $a$ fans into directly. This definition itself is very weak in the indication requirements and can be interpreted in different forms. A variable can be partially acknowledged by each of its direct fanouts or by only one of them; it can be acknowledged by a variable in both valid data and null cycles or only one cycle. The weakest form is that a variable is acknowledged by a group of variables but none of them individually.

Martin’s methods rely on elimination of validity check for some cell in-
puts, under the requirements of not introducing transient inputs, to relax the indication relation. This elimination is thought of entering a very subtle and uncertain territory where developed methodologies are lacking [34]. By contrast, our approaches with PA exclude the transient inputs in the first place with its indication requirements. What is more, we do not consider a validity check can be removed under some conditions like the “functional insurance” by Theorem 5 of Martin’s paper [32], but still think it is being checked with some weak forms like the functional ones in our category 3 of PA distribution. We demonstrate that this level of indication is so weak that its validity can only be checked by some global means.

We are satisfied with the definition of PA as the underlying requirements of the indication relations in a data path. Its distribution is very flexible which is promising in associating the data path synthesis with some optimisation problems; meanwhile, it supports very weak forms of indication, which means the space of optimisation can be quite large.

Our next effort is to design a library of modules that can be used in a flow making use of indication relations characterised by PA. A general procedure is proposed that can generate a DR module with any function and any input PA pattern. There is a practical concern in how many library elements we really need for synthesis. Increasing the types of function and number of inputs may increase the optimisation space but has more complex data management and design issues. Currently we are using a library made up of inverters, buffers, and NAND (NOR) modules with 2 input variables.
Some of these modules can be implemented with pseudo-static CMOS only but others can be designed by the standard static logic gates.

Performance analysis of a data path made up of these functional modules is necessary for the performance optimisation. We have two techniques for performance analysis. The first one simulates the delays under a particular input vector in a certain data cycle. This is totally deterministic as the inputs are known, and the influences by both PA and the functions implemented in the data path are captured by certain rules based on the definition of dominating inputs. This technique is informative but limited to occasions where the inputs are known \textit{a priori}. Alternatively, we develop a static timing analysis method independent of the input values. This technique gives an interval of a data path’s true delays by its min and max critical path delays. In min analysis, data values are supposed to be evaluated as early as possible whereas the opposite assumption holds in max delay analysis. Critical paths thus derived can be false paths that cannot be sensitised according to certain criteria. Therefore, min-max static timing analysis only gives a delay interval with blurred edges.

Synthesis flows are designed with the concept of PA and the supportive library elements. We consider the SADT paradigm with a fixed topology for synthesis. Depending on the set of functional modules that are included in the library and the level of PA distribution, the flow can be formulated by three different optimisation problems. MINLP formulation is discussed in detail in this thesis as the general form of formulating this problem. It has
the finest level of formulation and largest optimisation space but the most complex computation of the three. The other two formulations, unate and binate covering problems, can be viewed as the special cases of MINLP when more limited library elements and stronger forms of PA are used. Different trade-offs exist in the three formulations between the optimisation space and computation complexity, which give designers freedom in choosing the most suitable formulation form.

In the MINLP formulation, each pair of directly connected data path variables is encoded with a set of binary variables denoting the PA relation between the two data path variables. Certain constraints in terms of the binary variables exist such that the PA requirements of an implementation are satisfied. Furthermore, value assignments to these binary variables have different leverage on the costs of implementation such as area or performance. With a formulation like this, the robustness is introduced in a cost-aware manner and thus our design intent is met.

To help solve the MINLP problem, various methods have been proposed in this thesis. Some of them use pseudo global optimisation within a period of time; others decompose the problem into sub-problems that are easier to manage. User specified knowledge can be used to leverage the complexity with or without narrowing the search space. In addition, heuristic algorithms are discussed in cases when certain design objectives are favourable.

Results of using the MINLP flow on industry-level data path synthesis demonstrate that area and performance costs of NCL-D like implementa-
tions can be reduced without significant degradation of the robust level. In particular, the area is reduced by an average factor of 1.26, the min critical path delay by 1.60 and the max critical path delay by 1.27. The synthesised data path is free from gate orphans and has only wire-orphans that are more conservative than the iso-chronic forks in terms of timing assumptions. In addition, heuristic algorithms have linear run times with acceptable degradation of optimisation levels. Running times on the order of tens of seconds for the largest benchmark circuit make these algorithms appropriate for large scale applications.

7.2 Future work

Following areas are proposed for the future work.

7.2.1 non-SADT synthesis techniques

Various techniques exist in the literature to synthesise and minimise 2-level or multi-level combinational logics, such as substitution, division and factorisation. Some of these techniques provide useful insight in the context of asynchronous data path synthesis but must be introduced with care because of the hazard problems. We believe that an important issue in non-SADT synthesis is to have some appropriate representations for the data paths which for example can capture both logic function, as by boolean logics, and the indication relation as by signal transition graphs used in asynchronous con-
controller synthesis.

An equally important issue in non-SADT techniques is complexity control. Non-SADT techniques are intuitively more complex than SADT ones because the search space is larger whereas structural dependence is not available. This makes applications of non-SADT techniques very difficult without problem decomposition. As a result, good heuristic algorithms for circuit/problem decomposition are required.

Concepts of PA are helpful to non-SADT synthesis. First, the requirement of PA definition can be used to check whether a “move” during synthesis is legal or not in terms of hazard introduction. Second, as demonstrated in the category 3 of PA distribution, pure boolean functions can have some indication of the inputs under some circumstances that has close relation with the traditional synthesis techniques within synchronous domains.

### 7.2.2 Synthesis with mixed encoding

Data path synthesis with heterogeneous DI encoding is attractive in that it can bring new optimisation spaces, because different codes have different trade-offs in their resource/energy costs of transmission [34] and ease of completion detection. PA also helps in this research area because it can be expanded to illustrate the indication relation when other forms of DI codes are applied other than DR codes, as mentioned in the notes for Chapter 3.

A special case of this research area is the mix of DR and SR encoding in data path synthesis. DR (and DI) encoding is expensive in area and power
consumption but can have true delays based on measurements rather than predictions. It is meaningful to implement the critical parts of a circuit (for example the critical paths) using DR encoding whereas other parts by SR encoding to make use of the advantages of these different encoding schemes.
Appendix A

DRFM libraries

In this appendix, libraries of DRFMs used in this thesis for synthesising asynchronous data paths are introduced and listed. The library format is `genlib` used in `SIS` [49]. To be concise, the library modules use the convention of `DRFM — function — r fnb`, where \( r, f, n \) and \( b \) denote the number of inputs that are partially acknowledged for the phase of rising, falling, neither and both, respectively. As a result, \( NAND2\_a \downarrow b \uparrow \) and \( NAND2\_a \uparrow b \downarrow \) described in Chapter 4 are referred to as \( NAND2\_1100 \) in the libraries.

A DRFM specified in the library has the following format:

GATE <DRFM-name> <module-area> <DRFM-logic-function>
<pin-info>
...
<pin-info>
<DRFM-name> is the name of a DRFM.
<module-area> is the area of a DRFM estimated in transistor count.
<DRFM-logic-function> is the boolean function of a DRFM.

Each <pin-info> has the following information:
PIN <pin-name> <phase> <input-load> <max-load>
<rise-block-delay> <rise-fanout-delay>
<fall-block-delay> <fall-fanout-delay>

<pin-name> is an input variable of a DRFM, or * to denote that all input pins have identical timing information.

<phase> is the phase of an input pin that is partially acknowledged which can be one of {rising,falling,neither,both}
<input-load> gives the input load of pin.
<max-load> specifies the loading constraint of a DRFM where a default value 999 is used.

<rise-block-delay> and <rise-fanout-delay> are the rise-time parameters for an input pin. <rise-block-delay> is calculated by the parasitic delay and <rise-fanout-delay> by the logical effort when using LE methods.

<fall-block-delay> and <fall-fanout-delay> are the rise-time parameters for an input pin. <fall-block-delay> is calculated by the parasitic delay and <fall-fanout-delay> by the logical effort when using LE methods.

Figure A.1 and Figure A.2 list pa_min.genlib and pa_max.genlib, the libraries for min and max STA respectively. In these two libraries NOR DRFMs are not shown because they have exact dual area and performance to the NAND DRFMs.
GATE nand2_2000 17 O=!a+b;
PIN * RISING 1 999 2.0000 2.0000 2.0000 2.0000
GATE nand2_1100 16 O=!a*b;
PIN a RISING 1 999 1.3333 0.6667 1.3333 0.6667
PIN b FALLING 1 999 1.3333 2.6667 1.3333 2.6667
GATE nand2_1010 13 O=!a*b;
PIN a RISING 1 999 2.0000 1.6667 2.0000 1.6667
PIN b NEITHER 1 999 2.0000 1.3333 2.0000 1.3333
GATE nand2_1001 17 O=!a*b;
PIN a RISING 1 999 1.3333 0.6667 1.3333 0.6667
PIN b BOTH 1 999 1.3333 2.0000 1.3333 2.0000
GATE nand2_0200 18 O=!a*b;
PIN * FALLING 1 999 2.0000 3.0000 2.0000 3.0000
GATE nand2_0110 16 O=!a*b;
PIN a FALLING 1 999 1.3333 2.3333 1.3333 2.3333
PIN b NEITHER 1 999 1.3333 0.6667 1.3333 0.6667
GATE nand2_0101 18 O=!a*b;
PIN a FALLING 1 999 2.0000 3.3333 2.0000 3.3333
PIN b BOTH 1 999 2.0000 3.0000 2.0000 3.0000
GATE nand2_0020 12 O=!a*b;
PIN * NEITHER 1 999 2.0000 1.3333 2.0000 1.3333
GATE nand2_0011 15 O=!a*b;
PIN a NEITHER 1 999 1.3333 0.6667 1.3333 0.6667
PIN b BOTH 1 999 1.3333 1.6667 1.3333 1.6667
GATE nand2_0002 20 O=!a*b;
PIN * BOTH 1 999 2.0000 3.3333 2.0000 3.3333

Figure A.1: pa_min.genlib
GATE nand2_2000 17 O=(a+b);
PIN * RISING 1 999 4.6667 5.3333 4.6667 5.3333
GATE nand2_1100 16 O=(a*b);
PIN a RISING 1 999 3.0000 2.3333 3.0000 2.3333
PIN b FALLING 1 999 3.0000 3.3333 3.0000 3.3333
GATE nand2_1010 13 O=(a*b);
PIN a RISING 1 999 3.6667 2.6667 3.6667 2.6667
PIN b NEITHER 1 999 3.6667 2.0000 3.6667 2.0000
GATE nand2_1001 17 O=(a*b);
PIN a RISING 1 999 3.3333 1.3333 3.3333 1.3333
PIN b BOTH 1 999 3.3333 3.3333 3.3333 3.3333
GATE nand2_0200 18 O=(a*b);
PIN * FALLING 1 999 3.3333 4.6667 3.3333 4.6667
GATE nand2_0110 16 O=(a*b);
PIN a FALLING 1 999 2.6667 3.3333 2.6667 3.3333
PIN b NEITHER 1 999 2.6667 2.3333 2.6667 2.3333
GATE nand2_0101 18 O=(a*b);
PIN a FALLING 1 999 3.3333 2.6667 3.6667 4.6667
PIN b BOTH 1 999 3.3333 4.6667 3.6667 4.6667
GATE nand2_0020 12 O=(a*b);
PIN * NEITHER 1 999 2.0000 1.6667 2.0000 1.6667
GATE nand2_0011 15 O=(a*b);
PIN a NEITHER 1 999 2.3333 0.6667 2.3333 0.6667
PIN b BOTH 1 999 2.3333 2.6667 2.3333 2.6667
GATE nand2_0002 20 O=(a*b);
PIN * BOTH 1 999 4.6667 5.3333 4.6667 5.3333

Figure A.2: pa_max.genlib
Appendix B

An example session of designing asynchronous data paths

This session discusses the automation of asynchronous data paths synthesis and optimisation techniques through an example of ISCAS benchmark C17. It includes three parts where aspects such as design tasks and procedures, file formats, functions and commands involved, and output results are discussed when applicable.

The first part deals with the initial prototyping of an asynchronous data path before optimisations. In this part, a boolean network is tentatively mapped to a fully acknowledged DRFM netlist, and analysed for its connectivity and performance. Commands and scripts for these purposes are described which have been implemented by functions integrated in SIS programme.
The second part considers the formulations of the optimisation problems using AIMMS, an OR modelling language, and briefly discusses the implementation of optimisation algorithms using the same language and the solving of these problems by a MINLP solver known as Baron.

The last part uses the results from solving the problem and maps a boolean network to its final DRFM accordingly. The discussion of this part is parallel with the first part.

B.1 Pre-optimisation mapping and analysis

B.1.1 Technology mapping

Script.tm in Figure B.1 maps a boolean network (BN), which represents a SR circuit netlist, to another network so that each internal node has at most two direct fan-ins in the objective network. This step is required for the following PA mapping steps. Other network decomposition scripts in SIS can also achieve this goal where boolean minimisations are free to be added in. In SIS, BLIF format is used for boolean networks and genlib for libraries. As an example, C17_2.blif is shown in Figure B.2 after mapping C17.blif using nand-nor2.genlib. The command to run this script is SIS> source script.tm.
B.1.2 Connectivity and level analysis

*Script.netlist* in Figure B.3 analyses the connectivity and level information of a BN, which are needed in the synthesis and optimisation algorithms. Related with this script, function *myproj_dfs_recur* applies depth-first search of a BN from primary outputs, encode each node with a positive integer number, *aimms_num*, that will be used in AIMMS programming to denote a network node, and harsh this number using the key of *node_long_name* in SIS data structure *node_t*. Figure B.4 redraws the schematics of C17_2 where each node is annotated by (*node_long_name*, *aimms_num*).

A matrix $M$ is generated as an output of *script.netlist*, where a matrix entry $M(i, j) = 1$ if a node with an encoded number $i$ is a direct fan-out of node $j$. Table B.1 shows such a connectivity matrix of C17_2, *C17.m*, in a format readable to AIMMS.

Level information of a BN is also generated by this script. Figure B.5 shows the file *C17.l* containing C17’s level information in a readable format of AIMMS.
.inputs 1GAT (0) 2GAT(1) 3GAT(2) 6GAT(3) 7GAT(4)
.outputs 22GAT(10) 23GAT(9)
.names 1GAT(0) 3GAT(2) [73]
 0- 1
 -0 1

.names 3GAT(2) 6GAT(3) [70]
 0- 1
 -0 1

.names [70] 2GAT(1) [77]
 0- 1
 -0 1

.names [73] [77] 22GAT(10)
 0- 1
 -0 1

.names [70] 7GAT(4) [75]
 0- 1
 -0 1

.names [75] [77] 23GAT(9)
 0- 1
 -0 1

.end
read_blif C17_2.blif

circuit_netlist

Figure B.3: script.netlist

Figure B.4: Schematic of ISCAS benchmark C17

CircuitNetlist(i,j) := DATA TABLE

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>6</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table B.1: C17.m

133
LevelOfNode(n) := data {
3:0,8:1,4:0,7:0,11:1,1:3,9:3,2:1,6:1,5:2,10:2};

Figure B.5: C17.1

B.1.3 Initial PA mapping and analysis

`Script.pa_init_analysis` in Figure B.6 is run for the following three tasks:

1. Setup the data fields of each node in a BN related with PA information (`pa_t` of `node_t`). This is implemented by function `pa_setup_both`, which initialise the `pa_t` in such a way that the following mapping procedures know that a DRFM implementing a node will acknowledge both rising and falling phases of all its inputs.

2. Initially map a BN to a netlist of DRFMs according to the `pa_t` data value of the network’s nodes. Function `pa_mapping` fulfils this task. After this initial PA mapping, each node in a BN corresponds to a DRFM in library `pa_max.genlib` and delay information is set up for each node’s inputs.

3. Apply critical path analysis to a mapped BN using max STA techniques described in Chapter 5, which are implemented by function `delay_max_trace`.

In `script.pa_init_analysis`, command `pa_analysis` has three options: `g, d` and `t`. These options have the following meanings:
read_blif C17_2.blif
read_library pa_max.genlib
pa_analysis -d 0 -t 1
print_gate -s

Figure B.6: script.pa_init_analysis

d=0: perform initial PA mapping of a BN;
d=1: perform final PA mapping of an optimised BN using a G-file in the g option;

t=0: perform min critical path analysis;
t=1: perform max critical path analysis;

The output from running script.pa_init_analysis are listed in Figure B.7. It reports the progress of performing the above three tasks, a max critical path, and the path delays. Nodes on critical paths are listed from primary inputs to primary outputs with an increasing order indicated in the “num” column. “aimms_num” represents the nodes’ corresponding encoded numbers.

B.2 Synthesis and optimisation

An optimisation problem including its variables, constraints and objectives is first formulated using data types provided by the AIMMS language. The basic data elements in formulating C17_2, and their values after reading
SIS> source script.pa_init_analysis
-The network has 5 primary inputs, 2 primary outputs and 6 internal nodes
-Max analysis of the pre-optimised circuit
--The pa_t information has been set up
--6 nodes have been mapped
--trace the mapped network using max STA analysis
---arrival time of the latest rising output is 42.67
---arrival time of the latest falling output is 42.67
--analysing critical path of the circuit

<table>
<thead>
<tr>
<th>num</th>
<th>node_long_name</th>
<th>aimms_num</th>
<th>arrival</th>
<th>required</th>
<th>slack time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3GAT(2)</td>
<td>4</td>
<td>2.00</td>
<td>2.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>[70]</td>
<td>6</td>
<td>17.33</td>
<td>17.33</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>[77]</td>
<td>5</td>
<td>32.67</td>
<td>32.67</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>22GAT(10)</td>
<td>1</td>
<td>42.67</td>
<td>42.67</td>
<td>0.00</td>
</tr>
</tbody>
</table>

nand2_0002 : 6 (area=20.00)
Total: 6 DRFMs, 120.00 area

Figure B.7: Output from running script.pa_init_analysis

C17.m and C17.l, are summarised below in (1) to (3), where BN nodes are
denoted by their aimms_num.

(1) Basic sets

    CircuitNode, the root set of all nodes in a BN, equals \{1,2,...,11\};
    PI_IS, the set of all primary input and internal nodes, equals \{3,4,7,8,11,2,6,5,10\}
and has the binding index of i;

    IS_PO, the set of all internal nodes and primary outputs, equals \{2,6,5,10,1,9\}
and has the binding index of j;

    Phase, the set of all acknowledgeable phase of DR variables, equals \{rising,falling,both,neither\} and has the binding index of p;

    Fanins(j) contains the direct fan-ins of node j. For example Fanins(2)={3,4};
    Fanouts(i) contains the direct fan-outs of node i. For example Fanouts(6)={5,10};
(2) Variables of the MINLP formulation

\[ G(j \text{ in IS\_PO}, i \text{ in Fanins}(j), p \text{ in Phase}) \text{ are the 3-d binary variables} \]

whose values have been defined in Chapter 6;

\[ A(j \text{ in IS\_PO}) \text{ is a scalar integer variable denoting the area of node } j; \]

TotalArea, the total area of nodes in a BN, is defined by \( \text{sum}(j \text{ in IS\_PO}, A(j)) \);

(3) Constraints

\[ \text{Uni\_PA\_Phase}(j \text{ in IS\_PO}, i \text{ in Fanins}(j)): \text{sum}(p, G(j,i,p))=1; \]

\[ \text{PA\_Rising}(i \text{ in PI\_IS}): \text{sum}(j \text{ in Fanouts}(i), G(j,i,'rising')+G(j,i,'both'))\geq1; \]

\[ \text{PA\_Falling}(i \text{ in PI\_IS}): \text{sum}(j \text{ in Fanouts}(i), G(j,i,'falling')+G(j,i,'both'))\geq1; \]

The AIMMS software provides a GUI interface where these data types can be input and saved under corresponding data fields in a project. The problem defined in a project can be optimised for different objectives such as \( \text{minimise}(\text{TotalArea}) \) in case of an area minimisation.

Heuristic algorithms (algorithms 1-4 in the List of Algorithms) to solve these problems are also programmed using this language. The AIMMS programme language is specialised for mathematical programming where set/matrix manipulations are largely supported. However, it does not support some programming paradigms such as recursion. As a result, functions depending on recursion, such as \texttt{myproj\_dfs\_recur}, must be programmed using C codes under SIS.

Baron solver invoked in these algorithms is one of the solvers accepting the AIMMS modelling language that solves MINLP problems. Suppose C17_2
\[ G(j,i,p) := \text{data table} \]

<table>
<thead>
<tr>
<th></th>
<th>rising</th>
<th>both</th>
<th>falling</th>
<th>neither</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(5,6)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5,8)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1,5)</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1,2)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(6,7)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(6,4)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2,3)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2,4)</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(9,5)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(9,10)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(10,6)</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(10,11)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure B.8: C17_a_G

is solved for minimal area, C17_a_G, the G_file that contains G variable values, is output and shown in Figure B.8. In this file, node j partially acknowledges node i for the phase p if there is a corresponding entry of value 1. Further, C17_a_results in Figure B.9 is produced which contains solution information.

\[ A(j) := \{6:20, 10:20, 9:18, 1:17, 2:15, 5:15\}; \]

\[ \text{TotalArea} := 105; \]

\[ \text{Minimizing TotalArea, Solution Time} := 10.487; \]

Figure B.9: C17_a_results
B.3 Post-optimisation PA mapping and analysis

`Script.pa_fin_analysis` is used to map a BN to an optimised DRFM netlist according to a `G_file`, and apply the min-max STA analysis. An example of `script.pa_fin_analysis` for max STA is shown in Figure B.10 performing the following three tasks:

1. Setup the data fields of each node in a BN related with PA information by parsing the `G_file` generated by solving the optimisation problem. This is implemented by function `pa_setup`, which initialise the `pa_t` in such a way that the following mapping procedures know that a DRFM implementing a node will acknowledge its inputs as indicated by the `G_file`.

2. Map a BN to a netlist of DRFMs according to the `pa_t` data value of the network’s nodes. Function `pa_mapping` fulfils this task.

3. Apply critical path analysis to a mapped BN using max STA techniques described in Chapter 5, which are implemented by function `delay_max_trace`.

Figure B.11 shows the results from running `script.pa_fin_analysis`. It reports the progress of performing the above three tasks and the max critical path delays of the DRFM netlist after optimisation. In addition, the types and
read_blif C17_2.blif
read_library pa_max.genlib
pa_analysis -g C17_a_G -d 1 -t 1
print_gate -s

Figure B.10: script.pa_fin_analysis

SIS> source script.pa_fin_analysis
-The network has 5 primary inputs, 2 primary outputs and 6 internal nodes
-Max analysis of the post-optimised circuit
-The pa_t information has been set up
-6 nodes have been mapped
-trace the mapped network using max STA analysis
—arrival time of the latest rising output is 21.33
—arrival time of the latest falling output is 24.66
nand2_0002: 2 (area=20.00)
nand2_0011: 2 (area=15.00)
nand2_0101: 1 (area=18.00)
nand2_1001: 1 (area=17.00)
Total: 6 DRFMs, 105.00 area

Figure B.11: Output from running script.pa_fin_analysis

numbers of DRFMs in the post-optimised netlist is listed and the circuit area estimated.
Bibliography


143


147


