Self-Timed Logic and the Design of Self-Timed Adders

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2010

By

Balasubramanian Padmanabhan

School of Computer Science

Contents

List of Tables 6

List of Figures 8

Glossary of Abbreviations 12

Abstract 20

Declaration 21

Copyright 22

Dedication 23

Acknowledgements 24

1 Introduction 25

1.1 Motivation and Context……28

1.2 Research Contributions……29

1.3 Publications……30

1.4 Structure of the Thesis……31

2 Fundamentals of Asynchronous Circuits 33

2.1 Handshake Mechanism and Data Representation……33

2.2 Bounded and Unbounded Delay Models……39

2.3 C-element and Indicatability……47

2.4 Function Block – Definition and Characterisation……52

2.5 Summary……54

3 Self-Timed Combinational Logic 55

3.1 Seitz's Method……57

3.2 Singh's Approach……60

3.3 Direct Logic and Reduced Direct Logic Styles……61

3.4 Delay Insensitive Minterm Synthesis Technique……64

3.5 Dual-Rail Combinational Logic……66

3.6 David et al.'s Approach……67

3.7 Toms' Approach……72

3.8 Folco et al.'s Approach……………………………………………………….……….75
3.9 Circuits with Partial Acknowledgement……………….…………………...79
3.10 NCL Based Methods……………………………………………………….…….81
3.10.1 NCL_D Approach…………………………………………………….…….84
3.10.2 NCL_X Approach……………………………………………..………86
3.10.3 Block-Level Relaxation……………………………………………….87
3.11 Summary……………………………………………………………….……...88
4 **Function Block Realisation** **90**
4.1 Terminologies and Definitions……………………………………..………91
4.1.1 Support Set and Dependency Set………………………….……….91
4.1.2 Cubes Support Intersection Set and Cubes Dependency Intersection Set……………………………………………………………………..92
4.1.3 Cubes Relativity Set……………………………………..……….....92
4.1.4 Variables Identification……………………………………..………92
4.2 Orthogonality and SI Decomposition……………………….………...93
4.2.1 Mutual Orthogonality Set and Degree……………………………...93
4.2.2 Sum-of-Products and Orthogonal Sum-of-Products Forms…………..94
4.2.3 Criteria for SI Decomposition……………………………..……..94
4.2.4 Primary and Secondary SI Cubes……………………………..95
4.2.5 Datapaths Employing 1-of-*n* Codes…………………………….……..96
4.3 General Synthesis Models……………………………………….………98
4.3.1 Architecture for Strongly Indicating Circuits………………..……..98
4.3.2 Modifications to Suit Weak-Indication Timing Model……………...100
4.4 Set Theory Based Heuristic for Compact Realisation of Function Blocks…..102
4.4.1 Deriving MOSOP Expression from MSOP Expression……….......102
4.4.1.1 Support Set and Dependency Set……………………………104
4.4.1.2 CSI Set, CDI Set and Polarity Eliminated CDI Set……….....104
4.4.1.3 Describing Mutually Disjoint Cubes……………………...…105
4.4.1.4 Disjoint Set and Degree of Disjointness………………...…...105
4.4.1.5 Deducing MDSOP Form………………...……….........106
4.4.1.6 MDSOP Cost of Combinational Benchmarks……….……..109

4.4.1.7 MOSOP Cost of Function Blocks……………………………112
4.4.2 System Configuration…………………………………...………...116
4.5 Summary…………………………………………………………...……...122
5 **Self-Timed Carry-Ripple Adders** **124**
5.1 Single-Bit Adders and Their Evaluation…………………………….………..124
5.1.1 Using Dual-Rail Data Encoding……………………………………..125
5.1.2 Employing Hybrid Input Data Encoding…………………………....131
5.2 Dual-Bit Adder Designs and Their Evaluation……………………….……...135
5.2.1 Adopting Dual-Rail Data Encoding……………………………….……136
5.2.1.1 Local Weak-Indication……………………………..……137
5.2.1.2 Global Weak-Indication…………………………….………140
5.2.1.3 Comparative Evaluation…………………………………….141
5.2.2 Incorporating Heterogeneous Data Encoding………………………..143
5.3 Hybrid Adders…………………………………………………..…………147
5.4 Bottlenecks with Increase in Order…………………………….…………149
5.5 Redundant Logic Insertion……………………………………..…………150
5.5.1 Impact on a Single-Bit Adder Based on Hybrid Input Encoding…....151
5.5.2 Impact on Dual-Bit Adders Utilising Dual-Rail Encoding………..…152
5.5.3 Impact on Dual-Bit Adders Adopting Heterogeneous Encoding…....155
5.6 Summary…………………………………………………………..……157
6 **Self-Timed Section Carry Based Carry-Lookahead Adders** **159**
6.1 Background……………………………………………………………..160
6.2 Section Carry Based CLA Architectures…………………………….……164
6.2.1 Type 1 Architecture – Fundamental Topology………………………164
6.2.2 Type 2 Architecture – Topology with Least Significant RCA Section ……………………………………………………………..166
6.3 Evaluation with Two-Bit CLA Generator……………………………………168
6.4 Evaluation with Four-Bit CLA Generator……………………...………170
6.5 Summary and Inferences……………………………………………….….173
7 **Self-Timed Multi-Operand Addition** **176**
7.1 Tree Constructs………………………………………………………….176

7.2 Bit-Partitioning Scheme……………………………………………..………178

7.2.1 CSA Based Multi-Operand Addition…………………………...……180

7.2.2 Compressor Based Multi-Operand Addition…………………...……181

7.3 Evaluation and Comparison………………………………………………….184

7.4 Summary and Inferences…………………………………………....……..188

8 Conclusions and Scope for Further Work 192

8.1 Summarising the Thesis Contributions…………………………………..…..192

8.2 Multi-Level Synthesis of Weak-Indication Circuits……………………..…..195

References 198

Appendix 214

Word Count – 59,051

List of Tables

2.1 Data representation in dual-rail and 1-of-4 encoding schemes……………….………38

4.1 SOP and DSOP cubes for some combinational benchmarks, generated by Espresso………………………………………………………….…………111

4.2 DSOP cubes of various combinational benchmarks corresponding to different methods………………………………………………………………………112

4.3 Cost of ST realisation of combinational benchmark functions……………..……..…115

4.4 Delay, area and power metrics corresponding to different approaches for implementing C17 function block…...……..120

4.5 Delay, area and power metrics corresponding to different ST approaches for realising 32:1 MUX function block…………..…...………………………………...……120

4.6 Delay, area and power corresponding to different ST approaches for realising 1:32 DEMUX function block…..……………...………………………………...……..122

5.1 Delay and area metrics corresponding to different 32-bit ST RCAs………………...130

5.2 Listing critical path elements of the various ST RCAs………………………………130

5.3 Delay, area and power metrics of 32-bit ST RCAs incorporating HIE……..……..…134

5.4 Delay, area and power of various dual-rail dual-bit adder based 32-bit ST RCAs……………………………………………………………………………..…...142

5.5 Delay, area and power parameters of heterogeneously encoded 32-bit ST RCAs incorporating DB adders (with *extra logic*)………………………………..……...146

5.6 Delay, area and power parameters of 32-bit heterogeneously encoded ST RCAs (without *extra logic*)……………………………………………………….……...147

5.7 Delay, area and power metrics of 32-bit dual-rail encoded hybrid ST RCAs……..…148

5.8 Delay, area and power metrics of dual-bit and triple-bit adder based 48-bit ST RCAs……………………………………………………………………………..…...150

5.9 Comparing delay, area and power parameters of redundant and non-redundant 32-bit hybrid input encoded ST RCAs……………………………………………..…….152

5.10 Comparing delay and area metrics of redundant and non-redundant logic dual-bit adder based 32-bit ST RCAs employing DRE……………………………………...……...155

5.11 Comparing delay, area and power of hybrid and non-hybrid DSSC_CCAO_global adder *incorporating redundant logic*, evaluated using a 32-bit ST RCA structure……………………………………………………………..……………..155

5.12 Delay, area and power of NRL and RL dual-bit adder based 32-bit ST RCAs with HE…………………………………………………………………….……...156

5.13 Reference text for various ST adder blocks…………………………………….…157

6.1 Delay, area and power for 32-bit ST addition based on Type 1 topology with 2-bit CLA………………………………………………………………………..………168

6.2 Delay, area and power parameters of 32-bit ST addition based on Type 2 adder architecture with two-bit SCBCLA……………………………………….………170

6.3 Delay, area and power for 32-bit ST addition based on Type 2 adder architecture with two-bit CLA and RCA topology corresponding to Seitz, DIMS and Toms methods……………………………………………………………..………..171

6.4 Delay, area and power metrics for 32-bit ST addition based on Type1, Type 2 and Hybrid adder architectures with four-bit SCBCLA logic……………………..……..172

6.5 Comparing ST ripple carry and hybrid CLA adders in terms of delay and area components for different word widths……………………………………..…..175

6.6 Comparing ST ripple carry and hybrid CLA adders in terms of power dissipation for different word widths……………………………………………………...…...175

7.1 Delay, area and power parameters corresponding to bit-partitioned CSA based ST addition of 8 input operands, each of size 32 bits………………………..………..185

7.2 Delay, area and power metrics corresponding to bit-partitioned compressor based ST addition of 8 input operands, each of size 32 bits…………………………...…..…186

7.3 Area and power figures for bit-partitioned ST addition of 8 inputs, each of width 32 bits, using RCAs and CLA adders……………………………………….…..…….190

List of Figures

1.1 A typical synchronous system stage …………………………………………...…..…25

1.2 A typical asynchronous system stage……………………………………………..…….29

2.1 Bundled-data encoding and 2-phase handshaking…………………………………..…...34

2.2 Timing diagram of a 2-phase handshake protocol [71]…………………………….……34

2.3 Delay-insensitive (dual-rail) data encoding and 4-phase handshaking……………..…35

2.4 Timing diagram of a 4-phase handshake discipline [71]……………………….…….…36

2.5 Fundamental mode system configuration…………………………………………….……40

2.6 Illustration of isochronic fork assumption with respect to a primary gate input……...44

2.7 Illustration of DI, QDI and SI delay models…………………………………..……45

2.8 Schematic, specification and circuit realisation of a 2-input C-element…………..……..47

2.9 Different gate level implementations of a 2-input Muller element…………….……..49

2.10 Different gate level implementations of a 3-input Muller element…………….……..51

2.11 Gate level realisation of a 4-input Muller element…………………………….……...52

2.12 Depicting strong and weak-indication phenomena………………………..………..54

3.1 Enumerating the state space based on input order………………………..………...56

3.2 Seitz's strong-indication full adder………………………………………….………..59

3.3 Seitz's weak-indication full adder……………………………………….……………60

3.4 Realisation of a 4-input AND gate based on Singh's approach…………….………...61

3.5 2-input AND gate based on direct logic style…………………………….………..62

3.6 Martin's full adder (reduced direct logic style)…….………………………..……..63

3.7 Hazards due to naïve decomposition of a C-gate………………………..………64

3.8 Strong-indication full adder based on DIMS approach………………….…………65

3.9 Weak-indication full adder based on DIMS approach…………………….………65

3.10 DRCL realisation of F = *ab* + *cd* …………………………………..…….………..67

3.11 Realising X = *a'b'c* + *abc'* and Y = *a'bc'* + *ab'c* using David et al.'s method……..…..70

3.12 Weakly indicating realisation, based on modifications to David et al.'s method……..71

3.13 Full adder synthesised using Toms' approach…………………………………..…….75

3.14 BDD and circuit solution for F = ($p \oplus q \oplus r$) based on Folco et al.'s method……...77

3.15 Full adder synthesised using Folco et al.'s approach…………………………..……...79

3.16 Implementing a multi-output circuit on the basis of partial acknowledgement……....81
3.17 Static CMOS M-of-N implementation for Z = *ab* + *cd*……………………….……83
3.18 NCL_D system configuration [126] [127]………………………………..…...…84
3.19 NCL_D based logic equivalent for the function, Z = *ab* + *cd* ………………….…….85
3.20 NCL_X system configuration [127]…………………………………………….…….86
3.21 NCL_X equivalent for the function, Z = *ab* + *cd* ……………………………….……87
4.1 Typical ST system configuration………………………………………..…….99
4.2 Depicting PSICs and SSIC……..………………………………………………99
4.3 SI decomposed strongly indicating realisation of *check* function block……………..100
4.4 System architecture for weak-indication circuits……………………………………101
4.5 Cube-variable support matrix of SOP form of *newtag*, based on Espresso…………110
4.6 Cube-variable support matrix of DSOP form of *newtag*, based on Espresso………..110
4.7 Cube-variable support matrix of DSOP form of *newtag*, based on the MDSOP heuristic…………………………………………………………………..……..111
4.8 System topology in support of the proposed MOSOP heuristic for function blocks……………………………………………………………...………117
4.9 Illustrating CD of intermediate outputs for MUX logic tree based on Seitz's method……………………………………………………………………..121
5.1 *n*-bit dual-rail encoded ST carry-ripple adder architecture…………….……..…….127
5.2 Weakly indicating realisation of Petrify based full adder block……………………..128
5.3 Proposed weak-indication full adder module…………………………….…....129
5.4 Architecture of the *n*-bit hybrid input encoded ST RCA…………….……..……...131
5.5 Hybrid input encoded full adder based on Toms' approach……………..……..…...132
5.6 Proposed hybrid input encoded full adder block…………………………..…...133
5.7 ST system architecture highlighting input protocol conversion and data processing………………………………………………….………………...133
5.8 Analysis of switching power dissipation of various 32-bit ST RCAs…………..……135
5.9 Dual-rail encoded *n*-bit carry-ripple adder architecture featuring local indication……………………………………………………………..………137
5.10 Weak-indication DSSC_CCO adder module…………………………..…….138
5.11 Weakly indicating DSSC_CCAO_local adder block………………………….……139

5.12 DSSC_CCAO_global adder module…………………………………………..……..141
5.13 Heterogeneously encoded dual-bit adder block based *n*-bit ST RCA structure……..144
5.14 Weakly indicating heterogeneously encoded dual-bit adder module, corresponding to local indication……………………...……………………….……..……...144
5.15 ST system configuration handling heterogeneously encoded inputs and outputs..145
5.16 Modifications to the ST system configuration handling heterogeneously encoded inputs and outputs to pave the way for global indication……………….…...…146
5.17 Highlighting the area expense of dual-bit adder blocks in comparison with single-bit adder modules…………………………………….………………......…...148
5.18 Hybrid input encoded ST full adder with logic redundancy………………..……..151
5.19 Showing redundant logic insertion in DSSC_CCO adder module…………….……153
5.20 Highlighting redundant logic insertion in DB_HE_local adder module…………..……156
5.21 Depicting the delay, power and area metrics of various ST adder blocks for performing 32-bit addition based on the RCA topology…………………………………………158
6.1 Carry output description using generate and propagate signals……………………..….161
6.2 Output carry representation on the basis of generate, propagate and kill functions………………………………………………………………..……….162
6.3 Carry output representation assuming P = *a*+*b*…………………………..………..163
6.4 Type 1 ST section carry based CLA adder architecture based on DRE…………….…165
6.5 Type 1 ST section carry based CLA adder architecture based on HIE…………….…..166
6.6 Type 2 ST section carry based CLA adder topology based on DRE……………..….167
6.7 Type 2 ST section carry based CLA adder topology based on HIE………………….…...168
6.8 Relative comparison of area occupancy of two-bit CLA module designs………..….169
6.9 Highlighting the merit of HIE CLA adder topology over HIE RCA topology with respect to delay parameter………………………………………………………….…..172
6.10 Portraying the power increase of HIE CLA adder architecture in comparison with HIE RCA architecture……………………………………….…………………...…….173
7.1 Illustration of bit-partitioned multi-input addition scheme………………..………178
7.2 ST version of a typical *n*-bit CSA for adding four operands………………….………..180
7.3 ST logic compressor based multi-input adder to add four operands…..…………..…181

7.4 A synchronous (4:2) logic compressor implemented using two full adders…………182
7.5 A synchronous (4:2) compressor design based on discrete gates…………...……….182
7.6 ST (4:2) logic compressor block with carry input…………………………..………..183
7.7 ST (4:2) logic compressor module without input carry………………………..…….184
7.8 Area comparison of ST (4:2) compressors realised using different methods.............188
7.9 Describing positioning of CLA adders as a replacement for RCAs in the multi-operand adder...............……………………………………………………………………..…189
7.10 Latency metrics for CLA adders and carry-ripple adders to perform bit-partitioned ST addition of 8 inputs, each of width 32 bits…………………..………………..……....190

Glossary of Abbreviations

General Acronyms:

ack **–** Acknowledgement (signal)

ALU – Arithmetic and Logic Unit

AND2 – 2-input AND gate

AND3 – 3-input AND gate

AND4 – 4-input AND gate

AO222 – Complex gate implementing the function, F = AB + CD + EF

AO2222 – Complex gate implementing the function, F = AB + CD + EF + GH

BDD – Binary Decision Diagram

CAD – Computer-Aided Design

CD – Completion Detection

CDI – Cubes Dependency Intersection (set)

CDI$_{\mathbf{PE}}$ **–** Polarity Eliminated Cubes Dependency Intersection (set)

CE2 – 2-input C-element

CE3 – 3-input C-element

CE4 – 4-input C-element

CEN – C-element Network

CMOS – Complementary Metal Oxide Semiconductor

CR – Cubes Relativity (set)

CSI – Cubes Support Intersection (set)

DAG – Directed Acyclic Graph

D(C) – Dependency set of a cube, C

DDJ – Degree of Disjointness

DEMUX – Demultiplexer

DI – Delay-Insensitive

DIMS – Delay-Insensitive Minterm Synthesis

DJ – Disjoint (set)

DMO – Degree of Mutual Orthogonality

DRCL – Dual-Rail Combinational Logic

DRE – Dual-Rail Encoding

DRN – Dual-Rail Network

DSOP – Disjoint Sum of Products

ITRS – International Technology Roadmap for Semiconductors

LI – Literals Identification

MDD – Multi-valued Decision Diagram

MDSOP – Minimum Disjoint Sum-of-Products

MIMO – Multiple-Input Multiple-Output

MO – Mutual Orthogonality (set)

MOSOP – Minimum Orthogonal Sum-of-Products

MSOP – Minimum Sum-of-Products

MUX – Multiplexer

NCL – Null Convention Logic

NCL_D – Null Convention Logic utilising DIMS technique

NCL_X – Null Convention Logic with Explicit Completion Detection

nMOS – n-type Metal Oxide Semiconductor

OBDD – Ordered Binary Decision Diagram

OR2 – 2-input OR gate

OR3 – 3-input OR gate

ORN – OR-gates Network

OSOP – Orthogonal Sum-of-Products

OUTN – Output Network

PLA – Programmable Logic Array

pMOS – p-type Metal Oxide Semiconductor

PSIC – Primary Speed-Independent Cube

QDI – Quasi-Delay-Insensitive

req – Request (signal)

RISC – Reduced Instruction Set Computer

ROBDD – Reduced Ordered Binary Decision Diagram

RTZ – Return-To-Zero

SBDD – Shared Binary Decision Diagram

S(C) – Support set of a cube, C

SI – Speed-Independent

SIA – Semiconductor Industry Association

SIC – Speed-Independent Cube

SIMCAO – Strongly Indicating MUX with C-elements, AND gates and OR gates

SIMCO – Strongly Indicating MUX with C-elements and OR gates

SOP – Sum-of-Products

SSIC – Secondary Speed-Independent Cube

ST – Self-Timed

VI – Variables Identification

WIDCAO – Weakly Indicating DEMUX with C-elements, AND gates and OR gates

WIDCO – Weakly Indicating DEMUX with C-elements and OR gates

Adder Acronyms:

CCSA – Carry Completion Sensing Adder

CLA – Carry-Lookahead (adder)

Compressor based (with RCAs) – Bit-partitioned multi-operand adder employing compressors for the input field partitions with RCAs constituting the final stage of the partitions as well as the final multi-operand adder stage

Compressor based (with CLA adders) – Bit-partitioned multi-operand adder employing compressors for the input field partitions with CLA adders constituting the final stage of the partitions as well as the final multi-operand adder stage

CPA – Carry-propagate Adder

CSA – Carry Save Adder

CSA tree based (with CLA adders) – Bit-partitioned multi-operand adder employing CSA tree structures for the partitions with CLA adders constituting the final stage of the partitions as well as the final multi-operand adder stage

CSA tree based (with RCAs) – Bit-partitioned multi-operand adder employing CSA tree structures for the partitions with RCAs constituting the final stage of the partitions as well as the final multi-operand adder stage

DB – Dual-Bit

DB_HE_global – Dual-bit (adder) based on heterogeneous encoding corresponding to global weak-indication

DB_HE_local – Dual-bit (adder) based on heterogeneous encoding corresponding to local weak-indication

Decomposed_DIMS_DSSC – Dual-bit adder based on the speed-independent decomposed version of DIMS approach that adopts dual-rail encoding

DIMS_compressor_DRE – (4:2) logic compressor design based on the speed-independent decomposition of DIMS solution, utilising dual-rail encoding

DIMS_DRE (strong) – Single-bit strong-indication adder based on DIMS approach adopting dual-rail data encoding

DIMS_DRE (weak) – Single-bit weak-indication adder based on DIMS approach adopting dual-rail data encoding

DSSC – Dual-Sum Single-Carry (adder)

DSSC_CCAO_global – Dual-Sum Single-Carry (adder) with C-elements, Complex gates, AND gates and OR gates corresponding to global weak-indication

DSSC_CCAO_local – Dual-Sum Single-Carry (adder) with C-elements, Complex gates, AND gates and OR gates corresponding to local weak-indication

DSSC_CCO – Dual-Sum Single-Carry (adder) with C-elements, Complex gates and OR gates

Folco et al._DRE (weak) – Weak-indication single-bit adder based on Folco et al.'s method, adopting dual-rail data encoding

G – Generate (function)

HE – Heterogeneous Encoding

HIE – Hybrid Input Encoding

Hybrid_DSSC_CCAO_global – Hybrid adder incorporating both DSSC_CCAO_global and SSSC_DRE adders that corresponds to global weak-indication

Hybrid_DSSC_CCAO_local – Hybrid adder incorporating both DSSC_CCAO_local and SSSC_DRE adders that corresponds to local weak-indication

Hybrid_DSSC_CCO – Hybrid adder incorporating both DSSC_CCO and SSSC_DRE adders

Hybrid_TSSC_CCO – Hybrid adder featuring an optimal combination of TSSC_CCO and SSSC_DRE adder modules

Modified David et al._DRE (strong) – Modified David et al.'s single-bit adder based on dual-rail data encoding featuring strong-indication

Modified David et al._DRE (weak) – Modified David et al.'s single-bit adder based on dual-rail data encoding featuring weak-indication

Modified_Seitz_DSSC – Dual-bit adder adopting dual-rail data encoding based on a modification of Seitz’s approach

MOSOP_compressor_DRE – (4:2) logic compressor design based on the MOSOP heuristic, utilising dual-rail encoding

OS – Overturned Stairs (tree structure)

P – Propagate (function)

Petrify_DRE (strong) – Single-bit strong-indication version of full adder synthesised using Petrify tool, based on dual-rail data encoding

Petrify_DRE (weak) – Single-bit weak-indication version of full adder synthesised using Petrify tool, based on dual-rail data encoding

RCA – Ripple Carry Adder

SCBCLA – Section Carry Based Carry-Lookahead

Seitz_compressor_DRE – (4:2) logic compressor design based on a modification of Seitz's approach, based on dual-rail data encoding protocol

Seitz_DRE (strong) – Seitz's single-bit adder featuring strong- indication based on dual-rail data encoding

Seitz_DRE (weak) – Seitz's single-bit adder featuring weak-indication based on dual-rail data encoding

Singh_DRE (strong) – Singh's single-bit adder based on dual-rail data encoding featuring strong-indication

SOL – Sum Only Logic

SSSC_DRE – Single-Sum Single-Carry (adder) adopting dual-rail data encoding

SSSC_DRE (CPA) – A simple carry-propagate adder featuring single-bit adders that are based on conventional dual-rail data encoding

SSSC_DRE – Hybrid – Hybrid section carry based CLA adder constructed using single- sum single-carry adder modules based on dual-rail data encoding, including a 3-bit CLA generator module in the most significant nibble position

SSSC_DRE (Hybrid with 4-bit CLA) – Hybrid 4-bit CLA logic based adder incorporating single-bit adder modules, 4-bit CLA modules and a 3-bit CLA module in the most significant nibble position

SSSC_DRE – Type 1 – Type 1 section carry based CLA architecture constructed using single-sum single-carry adder modules based on dual-rail data encoding

SSSC_DRE – Type 2 – Type 2 section carry based CLA architecture constructed using single-sum single-carry adder modules based on dual-rail data encoding

SSSC_HIE – Single-Sum Single-Carry (adder) with Hybrid Input Encoding

SSSC_HIE_NRL – Single-Sum Single-Carry Hybrid Input Encoded (adder) with Non-Redundant Logic

SSSC_ HIE_NRL – Hybrid – Hybrid section carry based CLA adder constructed using single-sum single-carry adder modules with no redundant logic based on hybrid input encoding, including a 3-bit CLA generator module in the most significant position

SSSC_HIE_NRL – Type 1 – Type 1 section carry based CLA architecture constructed using single-sum single-carry adder modules with no redundant logic based on hybrid input encoding

SSSC_ HIE_NRL – Type 2 – Type 2 section carry based CLA architecture constructed using single-sum single-carry adder modules with no redundant logic based on hybrid input encoding

SSSC_HIE_RL – Single-Sum Single-Carry (adder) based on hybrid input encoding incorporating redundant logic

SSSC_HIE_RL (2-bit CLA – Type 1) – Type 1 CLA adder architecture built using single-bit adders based on hybrid input encoding with redundant logic and 2-bit CLA modules.

SSSC_HIE_RL (2-bit CLA – Type 2) – Type 2 CLA adder architecture built using single-bit adders based on hybrid input encoding with redundant logic and 2-bit CLA modules.

SSSC_HIE_RL (4-bit CLA – Type 1) – Type 1 CLA adder architecture built using single-bit adders based on hybrid input encoding with redundant logic and 4-bit CLA modules.

SSSC_HIE_RL (4-bit CLA – Type 2) – Type 2 CLA adder architecture built using single-bit adders based on hybrid input encoding with redundant logic and 4-bit CLA modules.

SSSC_HIE_RL (4-bit CLA – Hybrid) – Hybrid CLA adder architecture built using single-bit adders based on hybrid input encoding with redundant logic and 4-bit CLA modules but with a 3-bit CLA module in the most significant nibble position.

SSSC_HIE_RL (CPA) – A simple carry-propagate adder featuring single-bit adders that are based on hybrid input encoding with redundant logic

SSSC_ HIE_RL – Hybrid – Hybrid section carry based CLA adder constructed using single-sum single-carry adder modules with redundant logic based on hybrid input encoding, including a 3-bit CLA generator module in the most significant position

SSSC_HIE_RL – Type 1 – Type 1 section carry based CLA architecture constructed using single-sum single-carry adder modules with redundant logic based on hybrid input encoding

SSSC_ HIE_RL – Type 2 – Type 2 section carry based CLA architecture constructed using single-sum single-carry adder modules with redundant logic based on hybrid input encoding

Sync_ST_compressor_DRE – Proposed (4:2) logic compressor design employing dual-rail encoding, based on a translation of the synchronous version

Sync_ST_compressor_HIE – Proposed (4:2) logic compressor design employing hybrid input encoding for primary inputs, based on a conversion of the synchronous version

Toms_compressor_DRE – (4:2) logic compressor design based on Toms' approach, based on dual-rail encoding

Toms_compressor_HIE – (4:2) logic compressor design based on Toms' approach, based on hybrid input data encoding

Toms_DB_HE – Toms' dual-bit adder based on heterogeneous encoding

Toms_DRE (strong) – Strongly indicating single-bit adder based on Toms' approach, employing dual-rail data encoding

Toms_DSSC – Dual-bit adder synthesised using Toms' approach that utilises dual-rail data encoding

Toms_HIE – Hybrid input encoded single-bit adder synthesised using Toms' approach, utilising dual-rail data encoding

TSSC – Triple-Sum Single-Carry (adder)

TSSC_CCO – Triple-Sum Single-Carry (adder) with C-elements, Complex gates and OR gates

Abstract

The unorthodox methods usually employed for synthesising self-timed combinational logic incur substantial area overhead. A novel heuristic is proposed on the basis of set theory to considerably alleviate the problem of input state space explosion that besets function block realisations featuring several concurrent inputs. The heuristic has been implemented in Java and a system configuration in support of this heuristic is also presented. The proposed heuristic also forms the basis for realising many self-timed adders. The performance potential of various single-bit and dual-bit adder blocks, which adopt widely preferred homogeneous or heterogeneous delay-insensitive data encoding styles, are analysed on the basis of the self-timed carry-ripple adder architecture. Within this framework, hybrid adder schemes are also considered. With the intent of significantly reducing the datapath delay, the concept of redundant logic insertion has been put forward. Subsequently, to further improve the latency of dual-operand adders, self-timed section carry based carry-lookahead architectures have been proposed that outperform the basic self-timed carry-propagate adder topology. Finally, a bit-partitioning scheme for self-timed addition of multiple operands is described and a new self-timed logic compressor design is discussed. The impact of carry save adder and compressor tree structures, forming part of the input field partitions, on multi-operand addition is analysed through a case study, showing that the latter may be preferable compared to the former for self-timed multi-input addition.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Dedicated to my parents,

Mr. Padmanabhan Ramachandran

&

Mrs. Sumathi Padmanabhan

And my grandmothers,

Smt. Ganga Baghi Rathi (late)

&

Smt. Jayalakshmi

Acknowledgements

I thank my God for blessing me with health and strength throughout my PhD programme.

I express my gratitude to my supervisor, Dr. Doug Edwards for the opportunity to pursue my PhD in UK. I thank him for his kind support and encouragement. Especially, I thank him for his very helpful comments and suggestions with regard to the draft versions of my thesis.

I thank my advisor, Prof. Steve Furber for directing me to my supervisor when I first contacted him mentioning my research interest. I also thank him for his support and encouragement, especially at an important stage of my research study.

I thank the EPSRC, UK for the significant partial financial support and the School of Computer Science for the bursary.

I thank my dear family members, whose love, care and concern, great support and encouragement have been invaluable and have brought me thus far. Especially, I thank my parents for their financial support at the start of my research and also in helping to pay the difference in my tuition fee during the three years of my PhD.

Finally, I thank all the members of the APT group, especially Will, for providing me some study materials at the start of my research, having few discussions in between and for certain valuable comments on selected portions of my thesis draft, Jeff and Eustace for some inputs about EDA tools in the initial stages, Sohini for the ACiD-WG 2005 summer school handouts and few clarifications, Jeremy for proof-reading a couple of chapters of my thesis draft, Viv Woods for his encouragement and Danil of Newcastle University for some interesting exchanges. Thanks are also due to Charlie and Lilian for trouble-shooting the system (not a self-timed system!) faults as *early* as possible.

Chapter 1

Introduction

A majority of the present-day digital systems are clock based or synchronous, which assume that signals are binary and time is discrete. In general, synchronous systems comprise a number of subsystems that change from one state to another depending on a global clock signal, with flip-flops (registers) being used to store the different states of the subsystems. A conventional synchronous system is portrayed by figure 1.1.

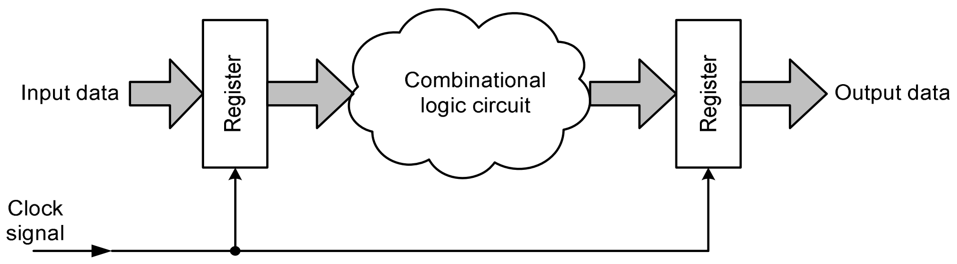


Figure 1.1: A typical synchronous system stage

The state updates within the registers are carried out on the rising edge (positive edge) or falling edge (negative edge) of the global clock – single edge triggering. The state of the global clock permits either data loading or data storage. Since the overall clock utilisation is only 50% for single edge triggered systems, double edge triggered flip-flops were subsequently proposed in the literature with the motive of increasing the system throughput as data can be loaded on both the rising and falling clock edges and data is retained when the clock signal does not toggle [1] [2]. However, this usually comes at the expense of a larger silicon footprint due to greater number of transistors and more interconnects for the dual edge triggered flip-flop and consequently leads to more power consumption. Preserving the original data rate as that of single edge triggered flip-flop designs whilst operating at half the system clock frequency might be helpful in reducing the dynamic power dissipation as the transitions could be reduced by half, but eventually this may be offset by more leakage power dissipation [2], which is becoming dominant in deep submicron technologies. Moreover, this mechanism tends to forego the advantages associated with single edge triggering in that its set-up and hold times are larger compared to conventional flip-flops and any deviation from its 50% duty cycle can lead to timing failures in critical paths upsetting the system behaviour [3]. In addition, it is more sensitive to noise apart from introducing complexity in system design and as such, the specification on jitter tolerance is more stringent which complicates the design of the system phase lock loop. As a result, synchronous designs with rising or falling edge triggering have been predominant, being the mainstream of digital system architectures; nevertheless, it is becoming increasingly difficult to overcome some fundamental limitations inherent in this approach.

The International Technology Roadmap for Semiconductors (ITRS) predicts that system-wide synchronisation is becoming infeasible owing to increasing silicon complexity [33]. A clock-based system can operate correctly only if all parts of the system see the clock at the same time, which can happen only if the delay on the clock wire is negligible. However, with advances in technology, the systems tend to get bigger and bigger in terms of the number of transistors and as a result the delay on the clock wires can no longer be ignored. The problem of clock skew is thus a major bottleneck for many system designers. Since the clock signal controls all flip-flops to sample and store their input data synchronously, it tends to be highly loaded and the problem becomes more severe. A widely preferred solution is to distribute the global clock using a clock network (clock tree) with clock buffers and thereby control the clock skew. Consequently, this results in an increase in the capacitance of the clock net and also suffers from increased activity (typically two transitions per net per cycle), even ignoring possible hazard activity on such nets.

The primary factors that govern the clock skew in a typical synchronous digital system are as follows:

- resistance, capacitance and inductance of the interconnection material used for the clock distribution network
- clock distribution network architecture, buffering schemes and clock buffers used
- fabrication process variation over the chip area
- number of processing elements in the system and the load presented by each element to the clock distribution network
- rise and fall times and the clock frequency

Various clock distribution strategies have been developed, with the most common and general approach being the use of buffered trees for equipotential clock distribution. However, to distribute high-speed clock signals, symmetric trees like the H-tree are preferred compared to the asymmetric buffered clock distribution tree structure. The H-tree network is the most widely used clock distribution network [4] – [6] to minimise the clock skew. It was shown in [7] that for an N × N array of processing elements, the clock pulse rise time and the clock skew associated with it are $O(N^3)$. Hence, with increase in N, the clock skew is likely to increase rapidly and become a stumbling block. Therefore, a distributed buffering scheme is often resorted to for synchronous digital integrated circuits by introducing buffers in the clock distribution network. However, the disadvantages of this approach are the extra area overhead and the increase in design sensitivity to process variations. Also, it has to be noted that buffers are the primary source of the total clock skew within a well-balanced clock distribution network. Since global clock periods are now commonly less than half a nanosecond, variations in delay by tens of picoseconds can seriously degrade the performance and reliability of high-speed synchronous systems [8]. With Moore's law [9] having been a driving force through process generations, supported by continual innovations in processes and device materials [10], to relentlessly pursue after greater integrated circuit densities, and with variability of process and device parameters assuming ever greater significance [11] [12] as devices are scaled down to more narrow dimensions, the above problem might only get exacerbated. The bottom-line is that clock management is becoming increasingly difficult and solving it in today’s high-speed complex system-on-chip designs appears to be a complex and costly affair.

The second major problem faced by designers is power dissipation, which is a very important metric that has gained significance with the phenomenal growth of portable electronics. For mobile electronic applications, the average power consumption has become the most critical design concern. For maximum efficiency, all gates in the system should be performing useful work. However in synchronous systems, this is not usually the case. Consequently, synchronous systems tend to consume more power than necessary. Many gates switch unnecessarily since they are connected to the clock and not because they have to process new input data. However, to circumvent this problem, clock gating is widely employed so as not to enable those sub-systems that are not required for any useful activity. The biggest gate is the clock driver itself which might occupy considerable area and must switch even if a small part of the system has something useful to do: the global clock, in general, was found to account for 15%-45% of the system power budget [13] and in a processor case study [14], it was found to be responsible for 34% of the total system power dissipation.

1.1 Motivation and Context

The problems of clock skew and power dissipation have been the major drivers for the worldwide resurgence of interest in asynchronous design – notable major projects include [15] – [29]. The design of clock-free or asynchronous systems has thus become attractive for digital system designers during the past two decades although asynchronous logic was explored from the infancy of integrated circuit design [30] - [32]. But synchronous design provided a far more efficient vehicle for exploiting the technology in commercial applications. The 2006 Semiconductor Industry Association's (SIA's) ITRS report on design stated that the percentage of designs driven by handshake clocking (asynchronous signalling) would rise from 11% in 2008 to 40% by 2020. The latest ITRS update on design [33] predicts that design re-use (as a percentage of all logic) would increase from a current figure of 38% to 55% by 2020. Over this period, parameter uncertainty (as a percentage effect on sign-off delay) is projected to increase from 10% to 25%. In fact, *reliability* has been labelled as one of the five crosscutting design challenges, which drives home the point that design robustness is becoming an increasing priority in deep submicron technologies. The above projections tend to forecast and necessitate a considerable shift in the design paradigm from conventional synchronous logic to asynchronous logic, as the latter benefits owing to its ability to tolerate supply voltage, process parameter and temperature variations [15]. Due to the absence of a global clock reference, asynchronous circuits tend to have better noise and electro-magnetic compatibility properties than synchronous circuits [34]. Also, they feature greater modularity permitting convenient design reuse [36]. Asynchronous operation by itself does not imply low power, but often suggests low power opportunities based on the observation that asynchronous circuits only consume power when and where active [35] [36]. The recent demonstration of the potential advantages of the world's first 8-bit physically flexible asynchronous microprocessor design over a synchronous flexible version in terms of power and noise figures by Karaki et al. from Seiko Epson's Technology Platform Research Centre [37], which utilises

4-phase handshaking and quasi-delay-insensitive design style, endorses the future of self-timed design techniques for even unconventional electronics.

Asynchronous circuits assume that signals are binary but the notion that time is not discrete. An *asynchronous system* is one in which there is no global synchronisation within the system; subsystems within the system are synchronised locally by the communication protocols between them. The results produced by the subsystems in an asynchronous system can be consumed by other subsystems as soon as they are generated without having to wait for a global clock tick. Moreover in asynchronous systems, a sub-system can easily be replaced by another subsystem with the same functionality but with different performance, but this is not a straightforward task in case of a synchronous system as the clock period might have to be recomputed. An asynchronous system stage that involves request/acknowledge handshake (signal exchange) signalling protocol is shown in figure 1.2. However, robust asynchronous systems embed the request information within the data wires and are usually referred to as *self-timed systems*. Self-timed systems are characterised by the absence of any timing reference to which all the operations are synchronised – being in stark contrast to synchronous systems where all operations are synchronised to the global clock signal.

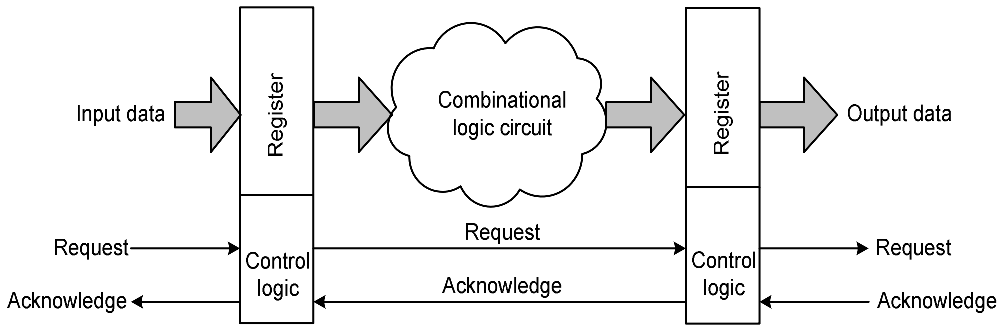


Figure 1.2: A typical asynchronous system stage

1.2 Research Contributions

Based on the research undertaken on self-timed combinational logic realisation and especially with respect to datapath elements, the original contributions of this thesis are summarised as follows:

- Formulation of speed-independent decomposition rules using set-theoretic principles.
- General multi-level synthesis models to realise strong or weak-indication combinational logic, which consider the entire input space.
- A set theory based heuristic for compactly synthesising combinational logic of arbitrary size as self-timed circuits and a system configuration in support of the proposed heuristic.
- Design of self-timed carry-ripple adders which feature local or global indication property and proposition of the concept of logic redundancy insertion for delay reduction.
- Self-timed section carry based carry-lookahead architectures that greatly minimise the latency of dual-operand addition in comparison with the ripple carry topology.
- A combinational bit-partitioning strategy addressing self-timed multi-operand addition and the design of a self-timed logic compressor.

1.3 Publications

The following list of publications gained and papers to be submitted for review corresponds to the contributions resulting from this research work.

- P. Balasubramanian and D.A. Edwards, “Efficient realization of strongly indicating function blocks,” *Proc. IEEE Computer Society Annual Symposium on VLSI*, pp. 429-432, 2008.
- P. Balasubramanian and D.A. Edwards, “A new design technique for weakly indicating function blocks,” *Proc. 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pp. 116-121, 2008.
- P. Balasubramanian and D.A. Edwards, “A delay efficient robust self-timed full adder,” *Proc. 3rd IEEE International Design and Test Workshop*, pp. 129-134, 2008.
- P. Balasubramanian and D.A. Edwards, “Power, delay and area efficient self-timed multiplexer and demultiplexer designs,” *Proc. 4th IEEE International Conf. on Design and Technology of Integrated Systems in Nanoscale Era*, pp. 173-178, 2009.
- P. Balasubramanian, D.A. Edwards and C. Brej, “Self-timed full adder designs based on hybrid input encoding,” *Proc. 12th IEEE Symposium on Design and Diagnostics of*

Electronic Circuits and Systems, pp. 56-61, 2009.
- P. Balasubramanian and D.A. Edwards, "Dual-sum single-carry self-timed adder designs," *Proc. IEEE Computer Society Annual Symposium on VLSI*, pp. 121-126, 2009.
- P. Balasubramanian and D.A. Edwards, "Heterogeneously encoded dual-bit self-timed adder," *Proc. 5th IEEE Conf. on Ph.D. Research in Microelectronics and Electronics*, pp. 120-123, 2009.
- P. Balasubramanian and D.A. Edwards, "Self-timed realization of combinational logic," *Accepted for presentation in the 19th International Workshop on Logic and Synthesis*, 2010.
- P. Balasubramanian and D.A. Edwards, "Redundancy insertion and latency reduction in self-timed adder blocks," to be submitted for review.
- P. Balasubramanian and D.A. Edwards, "Self-timed section carry based carry-lookahead adder architectures," to be submitted for review.

1.4 Structure of the Thesis

The organisation of this thesis is as follows:

- Chapter 2 discusses the fundamentals of self-timed systems. Specifically, the basics underlying robust asynchronous datapath logic implementation are explained.
- In Chapter 3, various self-timed combinational logic realisation schemes are reviewed.
- In Chapter 4, new terminologies are proposed to describe logic operations on the basis of set theory. Necessary criteria for speed-independent datapath logic decomposition are discussed and a general multi-level synthesis model is proposed for strong or weak-indication function block designs based on the dual-rail data encoding protocol, which can be extended to address any 1-of-*n* data encoding scheme. A set theory based procedure to derive two-level minimum orthogonal sum-of-products form is elucidated and the complexity involved in extending this heuristic to multiple levels is highlighted. A system configuration proposed to facilitate strongly or weakly indicating function block implementations on the basis of the above heuristic is then presented. Examples of a multiplexer and demultiplexer are considered to highlight the benefits of this strategy.
- Single-bit and dual-bit self-timed adder designs that utilise homogeneous or heterogeneous delay-insensitive data encoding schemes are discussed in Chapter 5. Modifications to a

speed-independent adder, in order to properly embed the property of indication (acknowledgement) into it are also mentioned. The carry-ripple adder topology has been considered for evaluation of the adder modules. The concept of redundant logic insertion that facilitates significant delay reduction in a logic cascade is then explained.

- In Chapter 6, novel self-timed section carry based carry-lookahead architectures for reducing the latency associated with conventional self-timed dual-operand carry-ripple addition are presented. An analytical estimation of the hardware complexity involved in realising higher-order carry-lookahead modules is also provided.
- Chapter 7 first discusses conventional tree structures for performing multi-operand addition. A combinational bit-partitioning scheme for performing self-timed multi-operand addition is then described. A (4:2) logic compressor is designed and addition of multiple operands using adders/logic compressors is separately examined through a case study.
- An overall summary of the thesis contents is first discussed in Chapter 8. Next, the issues involved in extending the two-level heuristic that has been proposed to implement arbitrary combinational logic functions as self-timed circuits into multiple levels are mentioned, and a feasible solution is then presented. This presents an interesting direction for further research in the domain of self-timed logic.

Chapter 2

Fundamentals of Asynchronous Circuits

The fundamentals of asynchronous systems such as handshake protocols, bundled-data and delay-insensitive data encoding schemes, modes of operation, various classes of asynchronous circuits based on the timing models adopted, Muller's C-element and the concept of indication, and the notion of a function block are discussed briefly in this chapter. References [38] – [40] provide a good introduction and comprehensive overview of asynchronous design methodologies in general. This chapter is intended to provide only a snapshot of the relevant details, with emphasis on topics of interest in the context of the subject matter of this thesis.

2.1 Handshake Mechanism and Data Representation

Asynchronous systems come in many flavours with the most prominent among them being bundled-data and dual-rail data encoding schemes. The communication protocol among these systems can also assume two forms: 2-phase (transition signalling) and 4-phase (level-sensitive signalling). Bundled-data encoding with 2-phase signalling and dual-rail data encoding with 4-phase signalling have been the popular choices in asynchronous circuit design until now and so they will be described here to provide relevant background information. In fact, dual-rail data encoding with level sensitive signalling continues to attract attention, as it is tolerant to variations in logic elements and communicating signal wires and hence has become attractive for deep submicron technologies [29] [37] [67].

The bundled-data protocol uses a request wire and an acknowledge wire and a set of single-rail data wires for data communication between the sender and receiver, as shown in figure 2.1. Hence, apart from the data bundle, there are two control wires: request (*req*) and acknowledge (*ack*). Together, they form a channel or medium of communication. In a typical 2-phase handshake protocol, the sender initiates the handshake mechanism with the data at hand by issuing a request to the receiver (by a transition on the *req* wire) and the receiver accepts the data and issues an acknowledgement (by a transition on the *ack* wire) to the sender. This completes a single transaction and sets the tone for a subsequent transaction, as depicted in the timing diagram of figure 2.2. The relations represented by solid arrows are functional constraints, while those indicated by dashed arrows are domain constraints. The crosshatched areas of input data and output data signify the time periods (intervals) during which data values may change; otherwise the data are stable and defined.

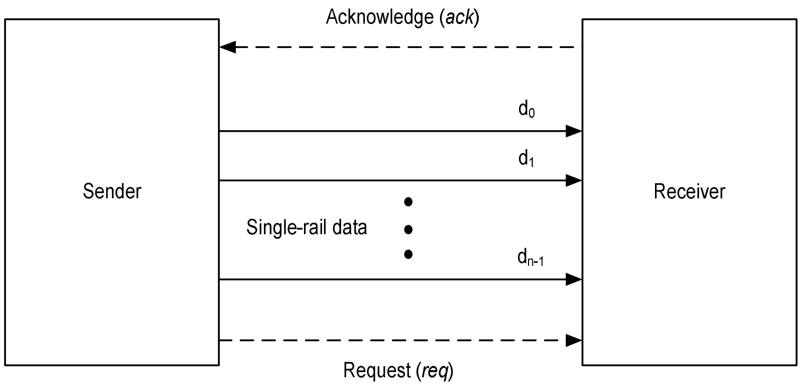


Figure 2.1: Bundled-data encoding and 2-phase handshaking

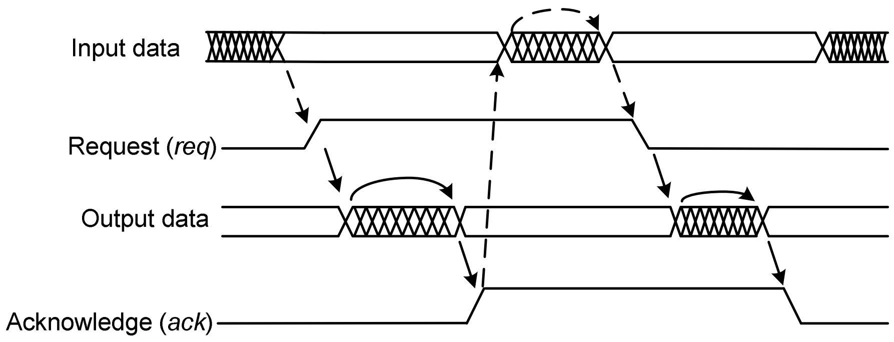


Figure 2.2: Timing diagram of a 2-phase handshake protocol [71]

It can be noticed that every transition on the *req* wire, both falling and rising, initiates a new request. Two-phase signalling is particularly useful for the realisation of high-speed Micropipelines [41]. Alternatively, the receiver can initiate the handshake process. Hence, if the sender is the active party who initiates the handshaking with the receiver being the passive party, then the channel is called *push channel* as the sender pushes the data to the receiver. Alternatively, if the receiver is the active party with the sender being the passive party, then the channel is known as *pull channel* as the receiver pulls the data from the sender. Traditionally, the request wire is used to inform the receiver about the validity of the data on the data bundle. This inherently places a constraint on the *req* wire, known as the bundling constraint. According to this constraint, the *req* wire must be asserted only after the bundled-data is valid at the receiver end. This is necessary to ensure that data wavefronts do not overlap and the system does not enter into a deadlock state. In other words, after a transition in the *ack* wire from the receiver to the sender that the data has been used; the sender can send the next set of data to the receiver. A micropipeline is delay-insensitive once the bundling constraints are met. Since there is no upper bound on the delays between consecutive events and even though the *req* wire is asserted after the data becomes valid at the transmitter end, arbitrary wire delays mean that this condition may not hold at the receiver; therefore, bundled-data protocols are not delay-insensitive.

In contrast to bundled-data encoding, dual-rail encoding does not use a separate *req* wire; instead the *req* signal is embedded within the data wires. Moreover, each data wire *x* is represented using two data wires *x*0 and *x*1, as shown in figure 2.3.

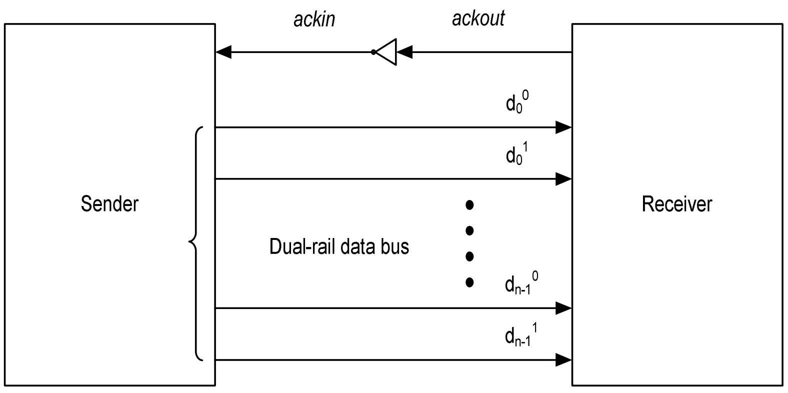


Figure 2.3: Delay-insensitive (dual-rail) data encoding and 4-phase handshaking

A transition on the *x*0 wire indicates that a *zero* has been transmitted, while a transition on the *x*1 wire indicates that a *one* has been transmitted. Since the request is embedded within the data wires, a transition on either *x*0 or *x*1 informs the receiver about the validity of the data. The condition of both *x*0 and *x*1 being a zero at the same time is referred to as the *spacer* or *empty data*. Both *x*0 and *x*1 are not allowed to transition simultaneously as it is illegal and invalid. The channel requiring (n+2) data wires in a bundled-data system would now comprise (2n+1) data wires with dual-rail signalling; nonetheless, the latter approach makes the signalling robust and therefore can tolerate random variations in wire delays when the bundling constraint cannot be guaranteed. With reference to figure 2.3, the 4-phase handshake protocol can be explained as follows[1]:

- The dual-rail data bus is initially in the spacer state. The sender transmits the codeword (valid data). This results in 'low' to 'high' transitions on the bus wires (i.e. any one of the rails of all the dual-rail signals is assigned a logic 'high' state), which correspond to non-zero bits of the codeword.
- After the receiver receives the codeword, it drives the *ackout* (*ackin*) wire 'high' ('low').
- The sender waits for the *ackin* to go 'low' and then resets the data bus (i.e. it is driven to the spacer state).
- After an unbounded, but finite (positive) amount of time, the receiver drives the *ackout* (*ackin*) wire 'low' ('high'). A single transaction is now said to be complete and the system is ready to proceed with the next transaction.

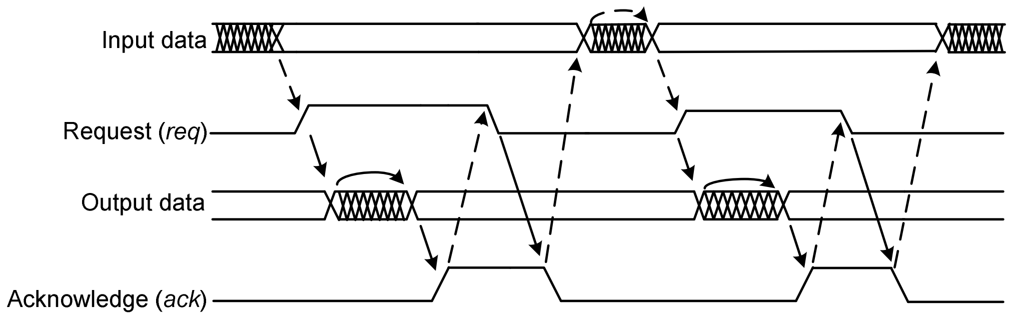


Figure 2.4: Timing diagram of a 4-phase handshake discipline [71]

The timing diagram for the 4-phase asynchronous signalling protocol is shown in figure 2.4, with the *req* wire explicitly shown to describe the handshaking process. It can be observed that four transitions are required to complete a transaction with this approach and there is an intermediate return-to-zero (RTZ) phase of both the *req* and *ack* wires preceding

[1] The explanation remains valid for data representation using any delay-insensitive data encoding scheme.

every transaction as the signalling convention is level-sensitive: *valid data* corresponding to logic *high* and *spacer data* corresponding to logic *low*. Comparing figures 2.2 and 2.4, it can be seen that the number of transactions processed during the same time interval is double in case of the bundled-data system employing 2-phase signalling compared to the encoded system employing 4-phase signalling convention. Though in principle every transition represents a meaningful event in case of the 2-phase protocol, such interface implementations require more logic and are more complex as circuits that process transitions require state. The increase in logic complexity may lead to more power consumption than was saved by fewer control transitions. This was the case with the low-power asynchronous ARM processor, AMULET1 [16]. The lack of a distinct low power advantage led to an improved implementation, AMULET2e [22], in which RTZ signalling convention was employed and performance improvement and power reduction over AMULET1 were reported [42]. Although the improvements were effected owing to enhanced design expertise and architectural improvements as well, nevertheless, it substantiated the reasoning that power and performance attributes cannot be solely judged on the basis of the number of control transitions per event.

It was mentioned earlier that in case of the dual-rail data encoding scheme, the input combination of both *x*0 and *x*1 being simultaneously *high* is not allowed because the coding scheme is *unordered* [43]. A coding scheme is said to be unordered, when none of its code words is contained in any other codeword. In simple terms, the positions of ones in a codeword are never a subset of the positions of ones in a different codeword (example, '01' and '10'). In fact, the dual-rail code is the simplest member of the general family of delay-insensitive *m*-of-*n* codes [44], where *m* lines are asserted 'high' out of a total of *n* physical lines to represent a codeword and the size (i.e. the number of unique symbols) of a generic *m*-of-*n* code is given by the binomial co-efficient *n* choose *m* = $\frac{n!}{m!(n-m)!}$. The dual-rail code is ideally suited for representing a single bit of information. To represent two bits of information at a time, the dual-rail code can be concatenated as shown in Table 2.1 or can equivalently be represented by means of a single 1-of-4 code. Indeed, the 1-of-4 encoded values of single-rail inputs given in Table 2.1 represent only one of many possible encodings and an arbitrary choice is considered here for illustration. As can be seen from Table 2.1, two non-redundant bits of information are represented at a time by asserting only half of the physical lines as logic 'high' in a 1-of-4 code in comparison with a dual-rail code, though both require the same number of physical lines. As a result, the 1-of-4 encoding approach requires only half as many transitions as that of a dual-rail encoding approach. Consequently, the dynamic power dissipation of the former scheme is very likely to be better than that of the latter due to reduced switching activity. This phenomenon was confirmed with the practical example of an ARM thumb instruction decoder [45]. However, considering the additional encoding and decoding circuitry required for 1-of-4 encoded datapaths in comparison with dual-rail code [50], the power savings gained are likely to diminish. This shall be articulated when dealing with self-timed dual-bit adders in Chapter 5.

Single-rail inputs		**Dual-rail encoded data**		**1-of-4 encoded data**			
A	B	(A1 A0)	(B1 B0)	E0	E1	E2	E3
0	0	(0 1)	(0 1)	0	0	0	1
0	1	(0 1)	(1 0)	0	0	1	0
1	0	(1 0)	(0 1)	0	1	0	0
1	1	(1 0)	(1 0)	1	0	0	0

Table 2.1: Data representation in dual-rail and 1-of-4 encoding schemes

Though higher order encoding schemes are possible, nevertheless, apart from the dual-rail (or 1-of-2) code that allows easier mapping between conventional binary functions, the other widely used delay-insensitive code is the 1-of-4 code. This is due to the reason that for self-timed datapaths, encoding (by sender) and membership test and decoding (by receiver) are important aspects and consequently, encoding and decoding complexity is dependent on the message space to be coded [46]. Completion detection circuits, which detect the arrival of valid/empty data, perform the membership test [47] for the delay-insensitive unordered codes. In short, completion detection circuits perform validity/neutrality test of input code words. Both dual-rail and 1-of-4 codes can be considered to belong to the class of one-hot codes or 1-of-*n* codes, which is a subset of the generic family of *m*-of-*n* codes [44] [48]. The other important reason for the non-usability of higher order 1-of-*n* codes is the degradation of coding efficiency, and so self-timed logic realisations are usually based on dual-rail and 1-of-4 codes. Given this, higher order *m*-of-*n* codes with better efficiency are preferred for inter-chip communication [49] [50]. The efficiency of a code is determined by the rate R (that specifies the number of bits per wire) and is equal to $\log_2 M/n$ [44]. Here, M represents the size of the code or the number of data values represented and *n* denotes the number of physical wires. In general, a 1-of-*n* code can represent *k* single-rail inputs, where $k = \log_2 n$. For example, to represent 8 single-rail binary inputs, a direct one-hot code representation would demand 256 physical lines (1-of-256 code), whereas only 16 physical lines would be required by a dual-rail coding scheme and a similar count for a 1-of-4 encoding scheme. Hence, for encoding with direct one-hot codes, the number of physical lines required is of $O(2^k)$ and therefore concatenation of lower order 1-of-*n* codes might lead to a better solution. When a dual-rail code and a 1-of-4 code are used to represent exactly one bit and two bits of information respectively, they are said to be *complete* [51]. A code is said to be complete, if and only if it contains all code words as implied by its definition. Even with one missing codeword, it would be labelled *incomplete*. A delay-insensitive coding scheme, in general, is required to be unordered and complete [74]. In this context, it may be noticed that the coding efficiency of complete dual-rail and 1-of-4 data encoding schemes is equal to 0.5.

2.2 Bounded and Unbounded Delay Models

Asynchronous circuit design methodologies can generally be categorised based on the timing models. Bounded delay models assume that the delay in all circuit elements and wires is known (thereby bounded). Circuits based on this model, coupled with the fundamental mode assumption, are generally referred to as Huffman circuits [31]. This is shown in figure 2.5. There are two basic assumptions underlying this model: i) only one input to the circuit is allowed to change at a time, and ii) the present-state entries of the combinational logic can change only after the logic has settled in response to a new input – this condition, when viewed along with the first constraint leads to the understanding that multiple input changes would necessitate multiple iterations by the non-regenerative logic thereby increasing the number of cycles required to complete computation.

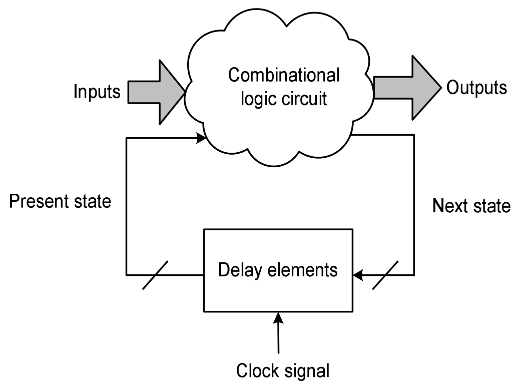


Figure 2.5: Fundamental mode system configuration

The fundamental mode restriction basically implies that before every external input transition, the entire system should have settled to a stable state with respect to a previous input transition. Burst-mode design style, developed by Nowick et al. [52] [53], based on an earlier work by Davis et al. [54], still requires the fundamental mode assumption but only between transitions in different input bursts. Instead of the overly restrictive fundamental mode assumption, burst-mode design permits simultaneous application of multiple inputs or a burst of inputs (i.e. multiple input changes are permitted), which is comparable with a synchronous system specification. No input burst can be a subset of another input burst leaving the same state, so that the system can determine when a complete input burst has occurred without any ambiguity. Fundamental mode and burst-mode circuits explicitly add delays in order to avoid certain hazard cases and are therefore non-robust design styles in the presence of variability. This also complicates the delay-fault testing of such circuits. Even though a circuit may not have any defects that can cause incorrect function, it might contain defects that could slow its operation. The delay-fault models attempt to classify these types of defects and their effects on the circuit. The two basic models used are: (i) gate delay or transitional fault model [55], where a single gate is assumed to take too long to produce an output, and (ii) the path delay fault model [56], where certain paths in the circuit may take too long to be exercised. In a typical synchronous circuit, such faults would require the chip to be clocked at a slower rate, whereas in an asynchronous circuit, since there is no clock-based synchronisation, it may end up in incorrect circuit operation that might be difficult to fix. Though delay-fault testing is not solely an issue with bounded delay models and may be problematic for unbounded delay models as well, the less pervasive timing assumptions of the latter facilitate comparatively easier testing, as they are more robust than the former. The bottom-line in fundamental mode circuits [32] is that environmental timing constraints are assumed, where the environment must wait for a circuit to stabilise before responding to the outputs. This requirement is identical to the hold time requirement for a simple latch or a flip-flop [57].

Input/output mode circuits [58] do away with such timing assumptions in that the environment is allowed to respond to a circuit's outputs without any timing constraints. In this case, outputs may be generated only after certain inputs have occurred and where next inputs may be generated only after certain outputs have occurred. The following circuit models operate in input/output mode without timing assumptions on when the environment should respond to the circuit:

- Delay-insensitive
- Quasi-delay-insensitive
- Speed-independent

A *delay-insensitive* (DI) circuit is designed to operate correctly irrespective of the delays of its gates and the delays encountered in the communicating signal wires, i.e. unbounded (arbitrary, but positive and finite) gate delay and wire delay models are assumed. This is the most robust of all the unbounded delay models, as such circuits are guaranteed to be *correct by construction* meaning they require no timing verification and can tolerate fluctuations in process parameters, temperature and noise and can also be ported between different technologies with ease, featuring excellent design modularity. The conceptualisation of such a circuit evolved from the Macromodules project by Clark and Molnar [59] [60] and was subsequently formalised by Udding [61]. It was shown in [62] and [63] that C-elements and inverters are the only DI elements as they satisfy certain criteria (mentioned in the next paragraph) and so, unfortunately, the class of pure DI circuits would be very limited, comprising only such elements. It was proved in [64] that gate-level realisation of such elements are not DI, that is to say, internal to the component, timing assumptions must be satisfied, while externally the component operates in a DI fashion.

In order that a circuit should be DI, certain conditions need to be satisfied. Before proceeding further, let us define the terminologies 'guard' and 'production rule'. The rising and falling signal transitions of signal *a* are denoted by $a\uparrow$ and $a\downarrow$ respectively. The post-condition for the result of a transition of type $a\uparrow$ is *a* (i.e. *a* is true), and the post-condition for the result of a transition of type $a\downarrow$ is $\neg a$ (i.e. *a* is false). For a Boolean inequality operation, with inputs *x, y* and output *z*, the following production rules specify the conditions for $z\uparrow$ and $z\downarrow$:

- $(\neg x \wedge y) \vee (x \wedge \neg y) \Rightarrow z\uparrow$
- $(\neg x \wedge \neg y) \vee (x \wedge y) \Rightarrow z\downarrow$

The conditions imposed on the input variables which lead to transition of type $z\uparrow$ and $z\downarrow$ to take place are referred to by the notations, G_u and G_d, which are known as the *guards* of the *production rules*, where a production rule characterises the Boolean condition on the input variables (guard) that leads to a specific signal assignment on the output variable: $G_u \Rightarrow z\uparrow$ and $G_d \Rightarrow z\downarrow$.

A DI circuit should satisfy the conditions of *non-interference* and *stability* [63]. The simultaneous execution of both the production rules (G_u and G_d) of a circuit would lead to malfunctioning of the circuit and therefore they should not be executed in parallel, i.e. they should be mutually exclusive. In that event, the production rules are said to be *non-interfering*. The other critical issue is the possibility for hazards, which can be eliminated by enforcing the stability of the guard of a production rule. The guard of a production rule is said to be *stable* if it cannot be falsified before the output corresponding to it has been derived (i.e. the output has transitioned). It was shown by Martin in [63] that no glitch[2] or hazard can corrupt the values of the variables, if a circuit fulfils stability and non-interference criteria and therefore to uphold these, two axioms need to be satisfied by DI circuits. In order to implement the stability criterion, the *acknowledgement theorem* [63] needs to be satisfied. According to this axiom, every non-final transition in a DI circuit should have a successor transition, i.e. a transition on the input of every gate, excepting that which produces the final output, should be accompanied by a transition on its output so that the transition on the gate input can be acknowledged (*indicated*). The axiom needs to be satisfied for both rising and falling events. Also, the *unique-successor-set theorem* [63] has to be satisfied, i.e. the set of computations in a DI

[2] Glitches produced as a result of electrical effects, such as crosstalk and noise, are however excluded.

circuit should adhere to the unique-successor-set property. According to this property, the set of non-final successor nodes that experience a transition as a result of a transition on an initial gate output node should be unique and the condition needs to be upheld for both rising and falling events. From the above discussion, it can be inferred that an OR gate is not a DI component. Following the Kleene star notation [65], the input sequence for an exemplar 3-input OR gate in which every transition on an input could be acknowledged is given below. It is obvious from the sequence that the 3-input OR-gate with inputs *x*, *y*, *v* and output *z*, with a simultaneous transition on two or more inputs, would fail to satisfy the acknowledgement property. In general, for an asynchronous circuit to be classified as a DI circuit, it is imperative that all transitions need to be acknowledged.

- $((x\uparrow; z\uparrow; x\downarrow; z\downarrow)^*; (y\uparrow; z\uparrow; y\downarrow; z\downarrow)^*; (v\uparrow; z\uparrow; v\downarrow; z\downarrow)^*)^*$

Since the class of pure DI circuits is very limited, a weakest compromise to delay-insensitivity was introduced, known as the *isochronic fork* assumption [62] [63]. DI circuits with isochronic fork assumptions are referred to as *quasi-delay-insensitive* (QDI) circuits, but it is not necessary that every fork be an isochronic fork in a QDI circuit [63]. The isochronic fork assumption has been defined by Martin in [63] as follows: "*In an isochronic fork, when a transition on one output is acknowledged, and thus completed, the transitions on all outputs are acknowledged, and thus completed*". In simple terms, a fork refers to a node or junction, from where signal wires branch out and therefore an isochronic fork assumption implies that the value (signal value, say '0' or '1') on all the branching-out wires from the fork is similar at any time instant. Technically, the difference between the delays in the branches of the fork is considered to be negligible in comparison with the delays in the gate elements and also the switching thresholds in the different gates to which the fork is an input are nearly the same. Though both these assumptions appear to be difficult to realise in smaller geometries, the isochronic fork assumption is usually confined to relatively very small circuit areas. However, when not implemented carefully at the circuit level, these may give to hazardous circuit behaviour, which could not have been suspected, and so the approach of uniform logic threshold voltages was proposed in [66]. In deep submicron technologies, verification at the layout level may usually be necessitated in order to strictly enforce this assumption and the extent of verification would generally depend upon wiring lengths, differences in gate construction and variation in switching thresholds. Adding delay elements to compensate for longer wire lengths may be one practical solution to this issue. Nevertheless, isochronicity is an essential assumption to facilitate design of non-trivial DI circuits. Intuitively, it can be observed, that the isochronic fork assumption avoids hazardous circuit behaviour by ensuring ideal circuit behaviour, i.e. it resolves the uncertainty in the arrival of signals to different destinations (branches) from the same source (node) by enforcing uniformity. Figure 2.6 shows an isochronic fork and the corresponding signal transitions for a simple conjunction of identical inputs. Given this, the AND gate should not be construed as a DI component, for reasons that shall be described in the next section. A recent work by Martin et al. [67] showing that the main building blocks of QDI logic, including realisation of the isochronicity assumption can be successfully implemented even in nano-CMOS technologies, where stricter design rules and large parameter variations could be expected, is an encouraging pointer towards the feasibility of this approach in the nano-CMOS era. In fact, it was shown that all QDI computations are deterministic, i.e. all Turing-computable functions have a QDI implementation [68]. In view of these, it can be understood that a DI circuit conforms to the unbounded delay model for both gates and wires, while a QDI circuit conforms to the unbounded delay model for gates and wires, with the exclusion of certain forks (subsequently certain wires, which are the fan-outs of the fork) called isochronic forks (equivalently, isochronic branches – those branches of the isochronic fork with unacknowledged transitions [67]).

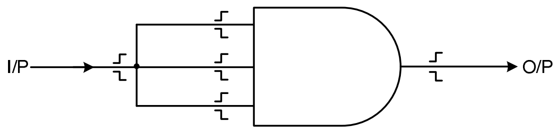


Figure 2.6: Illustration of isochronic fork assumption with respect to a primary gate input

The concept of *speed-independency* originates from Muller's work of the 1950's and 60's [30] [31] [69] and a speed-independent (SI) circuit is one which operates correctly regardless of gate (or component) delays; wires are assumed to have zero or negligible delay – unbounded gate delay and bounded wire delay. From the earlier discussion, it can be identified that QDI circuits assume zero delays with respect to isochronic forks (subsequently their branches) and so the description of a SI circuit basically necessitates that every fork be an isochronic fork. Technically speaking, wire delays are typically accounted for in the components according to the model and subsequently, wires are assumed to be ideal (i.e. zero delay). For this reason, a SI circuit is commonly referred to as a QDI circuit in practice.

Referring to the circuit fragment depicted in figure 2.7(a), d_{g1}, d_{g2} and d_{g3} represent the delay values of gates $g1$, $g2$ and $g3$ respectively, while d_{w1}, d_{w2} and d_{w3} signify the delay values of the corresponding nets. For the DI delay model, d_{g1}, d_{g2}, d_{g3}, d_{w1}, d_{w2} and d_{w3} can be arbitrary, while in case of the QDI delay model; d_{w2} is assumed to be equal to d_{w3} with f being construed as an isochronic fork junction. Considering the SI delay model, $d_{w1} = d_{w2} = d_{w3} = 0$, but the wire delays are accounted for in the delay of gate $g1$, whose output acts as inputs for gates $g2$ and $g3$. Hence, the delay of gate $g1$ is modelled as d_{g1}+d_{w1}+d_{w2} or d_{g1}+d_{w1}+d_{w3} as shown in figure 2.7(b).

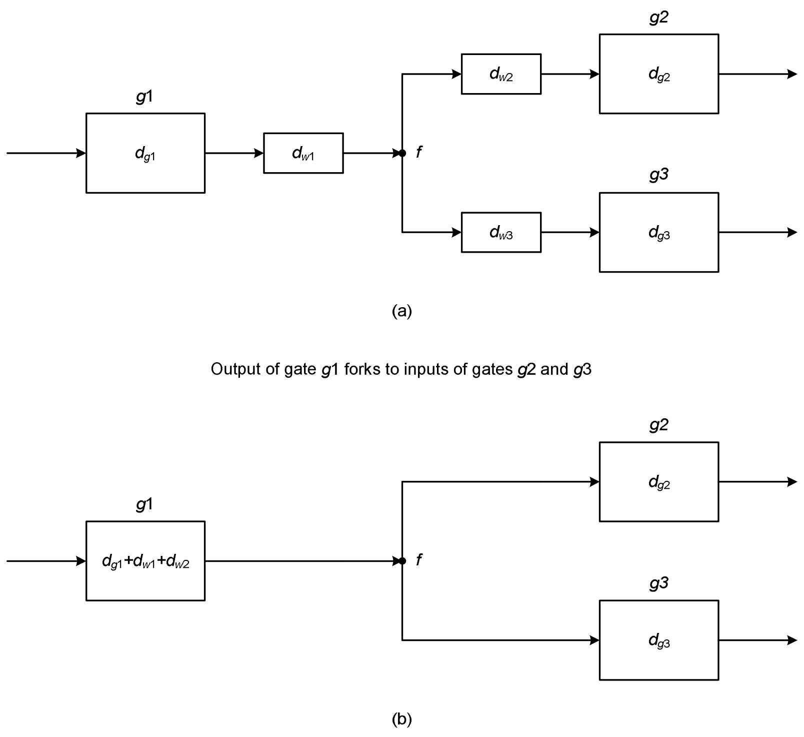


Figure 2.7: Illustration of DI, QDI and SI delay models

Asynchronous systems have been increasingly referred to as self-timed systems in the literature, as the term 'self-timed' implies that a system is governed by its own timing and not controlled externally by a common clock reference, i.e. the term 'asynchronous' has become synonymous with the term 'self-timed', though the latter is only a classification of the general category of asynchronous design that corresponds to a robust implementation. The term 'self-timed' was coined by Seitz [70] [71], and a self-timed (ST) system is either a single ST element or a legal interconnection of ST elements. An *element* or a group of elements is said to be contained in an *equipotential region*, where signals may be treated as identical at all points in a wire, i.e. wires incur negligible delay. A ST element can be speed-independent. There are literally no timing assumptions on the communication between the regions; i.e. communication between equipotential regions is DI. There are no timing constraints at the system level in a ST system and its correct operation can be subjected to only satisfying a topological constraint with respect to the interconnection of elements, with the system being composed of correctly functioning elements. As a result, systems based on bundled-data protocol are also referred to as ST systems in the literature, which are not robust, and the hazards associated with the underlying combinational logic are only hidden within the handshake mechanism through the use of delay-matching elements, which are optimised to reflect at least the worst-case delay of the combinational logic. In the true sense, ST systems would refer to a robust classification of asynchronous systems, when they adopt delay-insensitive data encoding and usually follow a RTZ signalling convention. Such systems are construed to be *self-checking*, at least with respect to single stuck-at faults [72]. In fact, this robust approach has its roots in the pioneering work of Muller et al. [30] [31] [69]. Given this, ST systems could employ a range of timing assumptions in general. At this juncture, we provide two clear justifications to our argument in this regard: Martin compares a ST circuit to a DI (i.e. in practice QDI) circuit in [62] and in [71]; Seitz refers to his ST full adder as a SI circuit. Therefore, ST design, in general, can be defined as that which guarantees correctness of circuit operation irrespective of delays associated with design components or that encountered in the communication signal wires and such tend to possess the inherent capability to absorb the parametric variations of devices. Henceforth, we shall use the term 'self-timed' to lay emphasis on only robust asynchronous designs, which in general, incorporate some delay-insensitive data encoding mechanism with 4-phase signalling.

2.3 C-element and Indicatability

The C-element, introduced by Muller [69], is an important gate widely used in asynchronous circuits and is the key element for implementing robust asynchronous logic. Many custom static and dynamic transistor level solutions have been proposed for this gate functionality in the literature [41] [65] [73] - [75]. The symbol, Boolean equation and a transistor level realisation of the 2-input C-element (CE2) with weak feedback are shown in the figure below.

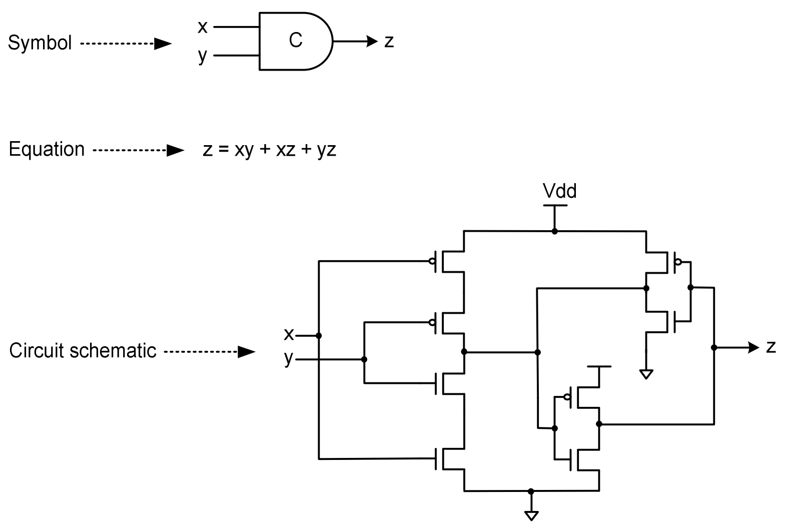


Figure 2.8: Schematic, specification and circuit realisation of a 2-input C-element

The CE2 outputs a 'high', when both its inputs are 'high' and outputs a 'low', when both its inputs become 'low'. In general, a random size C-element waits for all its inputs to become high (low) before producing a similar logic level at its output. Hence, it is also referred to as a *rendezvous* element, ST element [71] or DI element [62], as it governs the rendezvous of input signals in that, it acts only after all input events have arrived, i.e. it synchronises different events, and as such, it is referred to as a *symmetric* C-element. When the inputs are different, it retains its previous state and so the C-element is also referred to as a *state-holding element*. It basically implements the AND function for events, such that if a specific transition takes place at one input and it is coincident with, or followed by, a similar transition on the other input(s), then that transition will be presented to the output. The production rules for CE2 with inputs *x*, *y* and output *z* are specified as follows:

- $x \wedge y \Rightarrow z\uparrow$
- $\neg x \wedge \neg y \Rightarrow z\downarrow$

The C-element is an important DI operator, as it is *indicatable* [76] and therefore it satisfies the acknowledgement axiom [63] – this is the property of the symmetric C-element, which is widely used in 4-phase signalling conventions that employ DI codes. In [76], Varshavsky mentions that if the input(s) of a circuit are indicated to its output(s) then the circuit is indicatable. The condition for circuit indicatability is stated thus: "*The necessary condition for a circuit to be indicatable is that all functions in the SIF (system of inherent functions) of the circuit are isotonous (antitonous) in increasing variables and antitonous (isotonous) in decreasing variables in all allowed input transitions*". This condition is analogous to and can be described through the characteristic of a non-inverting buffer. Assuming '*a*' as the input and '*z*' as the output of a buffer cell, the following input and output transitions are valid: $a \Rightarrow z\uparrow$ and $\neg a \Rightarrow z\downarrow$. When the output of a C-element becomes 'high', then it highlights the fact that all its inputs have become 'high' and when the output of the C-element changes from 'high' to 'low', then it would imply that all its inputs have become 'low'. So the output of the C-element properly *indicates* the state of its inputs transitions or, the concurrent arrival of all the inputs are duly *acknowledged* (reflected) by the output. Because the output of the C-element properly acknowledges or *determines* the *complete* arrival of similar values on all its inputs (i.e. the state of the inputs can be determined from the output transition), the C-gate[3] is said to be *input-complete* [77] or *logically determined* [78]. A gate (circuit) that does not indicate the arrival of all its inputs on its output unambiguously is said to be input-incomplete or not logically determined.

A rudimentary gate level realisation, standard cell realisation and the SI realisation of the CE2 are shown in figure 2.9. The elementary gate level realisation, shown in figure 2.9(a) could lead to hazardous circuit behaviour. This can be understood from the following sequence of transitions: $(a\uparrow,\ b\uparrow) \rightarrow m\uparrow \rightarrow y\uparrow$. Subsequently, this will result in $l\uparrow$ and $n\uparrow$, which cannot be properly acknowledged by the OR gate. Such non-acknowledgeable (unobservable) transitions on gate output nodes are referred to as *gate orphans*, which need to be avoided as they may lead to improper circuit operation or cause malfunctioning of the

[3] The C-gate is also identified as an input-complete gate or non-relaxed gate in this thesis.

circuit (unpredictable circuit behaviour leading to possible erroneous output states). *Wire orphans*, though less problematic than gate orphans can arise in a circuit, but they are overcome with the assumption of equipotential regions or isochronic forks, eliminating the need for acknowledgement of a transition on a wire fork on all its fan-out branches, i.e. wire orphans are due to non-isochronic branches. To avoid gate orphans, timing assumptions are necessary, which in turn complicate the verification process. If the environment can be assumed to be sufficiently slow changing, then the elementary gate level realisation can be considered to be safe.

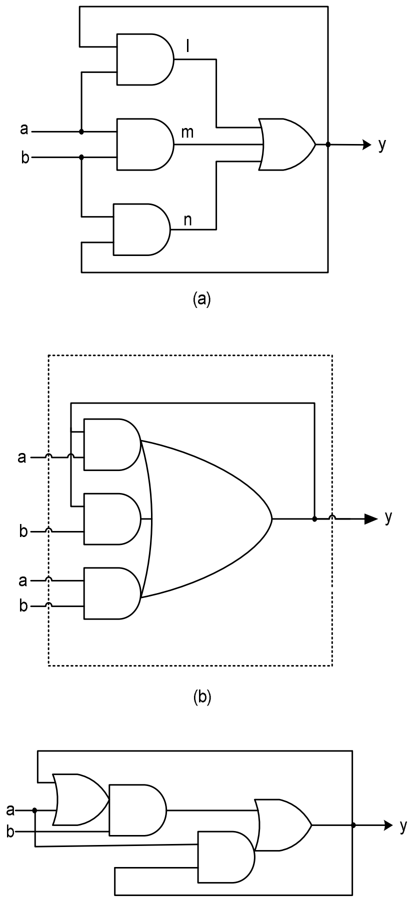


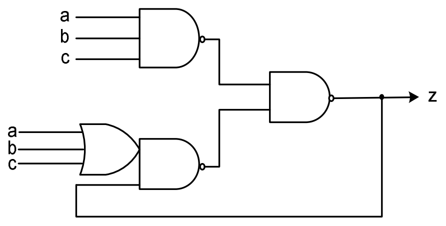
Figure 2.9: Different gate level implementations of a 2-input Muller element

A better approach would be to eliminate this timing assumption by a process called *merging* – discussed in detail in the context of cell merging for asynchronous threshold networks in [79]. If the three 2-input AND gates and the 3-input OR gate can be merged together and replaced by a single complex gate (AO222 cell) as shown in figure 2.9(b), then there would not be any gate orphans and thereby the circuit is said to preserve *gate orphan freedom*. In this case, the transition on *y* would be acknowledged by a transition on the downstream logic. Merging is possible whenever there is a choice of an element in the base function set for technology mapping which implements the combined functionality of discrete logic gates. Given this, the granularity of the base-function set additionally impacts the optimisation potential [80]. In general, it can be concluded that composing a larger gate from smaller gates (i.e. merging) could help in elimination of hazards, while naive decomposition of a larger gate into smaller gates [81] [82] by utilising the associative property [71] can give rise to hazards. Figure 2.9(c) shows the SI realisation of 2-input C-element functionality synthesised using Petrify tool [83], requiring two complex gates – OA12 and AO12 cells. Since the standard cell realisation requires only a single complex gate, it was preferred for all the simulations corresponding to this research work.

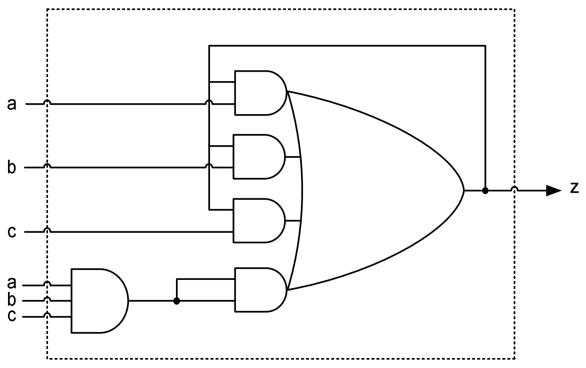
Since provision of state-holding elements with good granularity might be vital for realising high performance designs, 3-input and 4-input C-element functionality has also been realised using the gates of a standard cell library (130nm UMC CMOS process technology). The Boolean expression and the production rules corresponding to the 3-input symmetric C-element, with inputs *a*, *b*, *c* and output *z* is given as follows:

- $z = abc + az + bz + cz$
- $a \wedge b \wedge c \Rightarrow z\uparrow$
- $\neg a \wedge \neg b \wedge \neg c \Rightarrow z\downarrow$

The gate level representation of the 3-input C-element based on the *extended isochronic fork assumption* [84] is given in figure 2.10(a), while the more robust proposed implementation is shown in figure 2.10(b). While an isochronic fork corresponds to a delay assumption applied to the inputs of gates that are connected to the branches of a fork, the notion of an extended isochronic fork introduces a delay assumption on the outputs of the inverting CMOS gates connected to the branches of a fork. The so-called extended isochronic fork assumption simplifies the implementation of sequential gates by further weakening the original isochronicity assumption. The implementation depicted in figure 2.10(b) utilises the combination of a conventional 3-input AND gate and an AO2222 complex gate. It may be that many standard cell libraries might not have the AO2222 cell as one of their constituents; in which case, the 3-input and 4-input C-elements can be decomposed in a SI fashion by employing 2-input C-elements.



(a)



(b)

Figure 2.10: Different gate level implementations of a 3-input Muller element

A possible robust realisation of the 4-input C-element is depicted by figure 2.11. In this context, it should be noted that efficient gate level implementations of high fan-in Muller elements are largely technology-dependent.

An inverting buffer with input *x* and output *z* can be classified as a DI operator and is governed by the following production rules.

- $\neg x \Rightarrow z\uparrow$
- $x \Rightarrow z\downarrow$

Apart from the NOT gate, many other conventional logic gates (such as AND, OR, NAND, NOR, XOR and XNOR) fail to satisfy the acknowledgement property. For example, if the output of an XNOR gate is 'high', then no claim can be made regarding the certainty of the state of its inputs, i.e. whether all its inputs are 'low' or 'high'. Similarly, after this, if the XNOR gate output switches to 'low', then again a greater ambiguity is introduced as one cannot be sure as to which input(s) change caused the output to switch from 'high' to 'low'. Hence, it is said to be non-indicating, as it does not unambiguously reflect the arrival of its inputs on its output. In case of simple logic gates such as AND and OR, the acknowledgement property is satisfied only for rising or falling transitions respectively and not both.

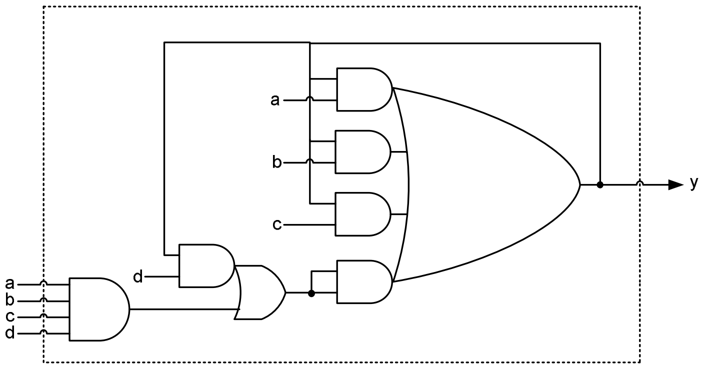


Figure 2.11: Gate level realisation of a 4-input Muller element

2.4 Function Block – Definition and Characterisation

A *function block* is the asynchronous equivalent of a synchronous combinational logic circuit [40]. However, in addition to satisfying the requisite functionality, it should also be transparent to the handshaking as implemented by its surrounding latches. Most robust function block designs adhere to a 4-phase handshaking convention for simplicity of implementation and can employ any DI data encoding scheme. The outputs of a function block are also entrusted with the responsibility of indicating the completion of computation within it, i.e. to say, whether all internal nodes have attained the correct steady-state value. There should also not be any dangling inputs or outputs within the function block. Seitz classified a function block into two basic robust categories depending on their indicating mechanism as either *strongly indicating* or *weakly indicating* [71]. It was also proved therein that a legal interconnection of strongly or weakly indicating function blocks is itself a strong or weak-indication function block. This property allows composition of a larger function block from smaller ones. Besides, QDI combinational logic circuits naturally tend to have inverter free realisations regardless of the indication property [51]. Relatively less robust (requiring more timing assumptions) models of function blocks, viz. relative timing [85] and monotonic Boolean networks [86], have also been proposed in the literature and these basically trade-off robustness for improved performance gains. Nevertheless, in this thesis, we shall exclusively deal with only the two most robust categories, listed in descending order of safety.

- *Strong-indication*: In this case, the function block waits for all inputs (valid/spacer) to arrive before it starts to compute and produce all outputs (valid/spacer). The sequencing constraints are briefly mentioned below:
    - ➢ All inputs become defined (valid)/undefined (spacer) before any output becomes defined/undefined, i.e. any or all output(s) become defined/undefined only after all inputs become defined/undefined
    - ➢ All outputs become defined/undefined before any input becomes undefined/defined
- *Weak-indication*: In this case, the function block starts to compute and produce outputs (valid/spacer) even with a subset of the inputs (valid/spacer). However, Seitz's weak timing specifications require that at least one output (valid/spacer) should not have been produced until after all inputs (valid/spacer) have arrived. The sequencing constraints are as follows:
    - ➢ Some inputs become defined (undefined) before some outputs become defined (undefined), i.e. some outputs could become defined (undefined) only after at least some inputs become defined (undefined)
    - ➢ All inputs become defined (undefined) before all outputs become defined (undefined), i.e. all outputs could become defined (undefined) only after all inputs become defined (undefined)
    - ➢ All outputs become defined (undefined) before any input becomes undefined (defined)

The signalling scheme for strong and weak-indication timing regimes in terms of the inputs and outputs is illustrated graphically in figure 2.12. From the preceding discussion, it can be understood that the C-gate and inverter can be identified as strongly indicating elements. In general, the maximum datapath delay can be reduced in the case of weak-indication circuits compared to their strong-indication counterparts through relaxation of indication constraints for all but one of the function block outputs by incorporating relaxed gates.

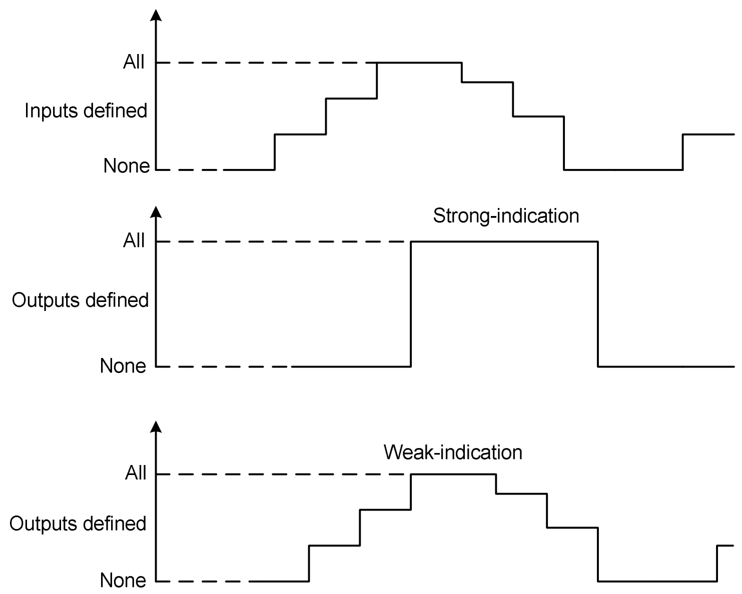


Figure 2.12: Depicting strong and weak-indication phenomena

2.5 Summary

In this chapter, the fundamentals of asynchronous circuits, relevant to the subsequent contents of this thesis have been described in detail. The topics discussed include handshake signalling protocols, delay-insensitive data encoding schemes, various unbounded delay models, Muller C-element and the concept of indication (acknowledgement), and classification and specification of robust timing regimes.

Chapter 3

Self-Timed Combinational Logic

Dual-rail encoding (DRE) is a widely used DI data encoding convention for robust asynchronous designs as it is a systematic code (input data being embedded in the encoded data) [88], and interest in 1-of-4 encoding is attributable to its inherent low power advantage. However, since many logic realisation schemes adopt DRE as a standard, a number of well-known ST combinational logic realisation schemes shall be analysed on the basis of DRE with examples.

ST implementation of combinational logic continues to be a field of important research activity on its own accord as it is inherently beset with the problem of *input state space explosion*, which poses exponential complexity with increase in the number of concurrent inputs; dealing with this problem is indeed a troublesome task [89]. Such is the gravity of the problem that even a standard tool preferred for SI synthesis of asynchronous controllers viz. Petrify [83] is unable [90] to synthesise a two-bit adder functionality (comprising 10 inputs and 6 outputs, in dual-rail format) in a SI fashion with the assumption of inputs being fed from and outputs being provided to the external environment. A ST realisation typically satisfies the acknowledgement property and the unique-successor-set property [63]. This is facilitated, if the *monotonic cover condition* [91] that ensures hazard-free implementation of SI circuits is incorporated into the description of the logic functionality. In simple terms, the monotonic cover constraint requires that only one product term in a sum-of-products implementation is allowed to assume a logic 'high' at any time in case of either *set* (true output) or *reset* (false output) functions [40]. Obviously, this requirement needs to be satisfied by the circuit only in the states that are reachable. In general, this would entail enumeration of the entire input state space that consists of distinct input combinations. Thus, an exponential increase in computational complexity of $O(2^n)$ is exhibited for even a gradual increase in the number of primary function inputs by $O(n)$ as shown in figure 3.1, with n being the number of concurrent single-rail inputs – this is commonly referred to as the input state space explosion problem and is a major bottleneck for implementation of random combinational logic as ST circuits. This phenomenon usually manifests while attempting robust asynchronous logic design. However, research has been pursued to alleviate the problem of input space explosion.

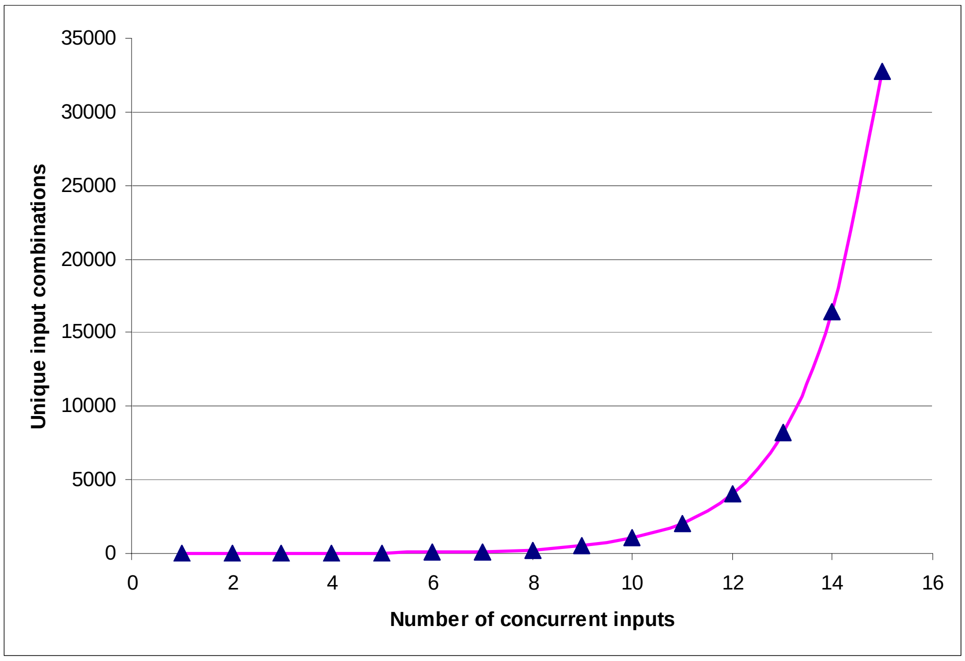


Figure 3.1: Enumerating the state space based on input order

Many properly indicating logic realisation schemes usually suffer from large area overheads and this has restricted direct function block implementations to usually those with fewer inputs and outputs, of which the fundamental datapath element viz. half/full adder has generally served as the exemplar circuit. Function block implementation of larger combinational circuits would incur at least double the area penalty of a conventional synchronous realisation, besides rendering the synthesis scheme practically infeasible for large functions. However, many approaches have been proposed and they differ in the way of dealing with this problem by either:

a. Assuming the entire space without suggesting a suitable decomposition procedure or

b. Confining themselves to only full custom solutions for smaller functions of practical interest or

c. Circumventing this problem considerably in different ways by usually relying upon a standard synchronous solution base and then constructing asynchronous solutions (de-synchronisation), with the additional provision of availability of full custom library gates that are made available as part of a standard cell library or

d. Resorting to SI logic decomposition by considering the entire input space or

e. Reduction of entire input space consideration with the exception of at least a single output by way of possible SI logic optimisations or

f. Addressing the problem of indication of all the primary inputs by way of partial acknowledgement, starting from a synchronous solution base (de-synchronisation), but assuming/requiring the presence of certain custom logic gates in the library.

Among these, the third approach has been dominant, having its roots in some of the earlier approaches, as it does not synthesise asynchronous circuits based on specifications such as communicating processes [92] or signal transition graphs [93]. Instead, it relies upon synchronous CAD tools for initial synthesis and then replaces every gate in the synchronous circuit with a dual-rail encoded gate pair in a template based fashion, which are subsequently mapped using NULL convention logic operators [77]. These operators are based on threshold logic [94] and are made available as custom elements in a standard cell library.

3.1 Seitz's Method

Seitz's approach to self-timed design [71], in its basic form, can be envisaged as an AND-OR two-level implementation of logic functions. It resembles the two-level AND-OR logic corresponding to the standard C-element architecture, which comprises first-level AND gates assuming unbounded fan-in and OR gates in the next logic level, which could be decomposed into multiple levels in an arbitrary fashion. C-elements are then used to join the set and reset functions corresponding to each signal [91] [95]. However, as with the standard C-element architecture, synthesis of larger combinational functions is practically infeasible as they could contain several concurrent inputs [91] [96]. Seitz's method basically requires generation of AND logic operators (assuming unbounded fan-in) for each of the 2^n unique minterms[4] of a

[4] A *minterm* is a canonical product of the input variables of a Boolean function, while a cube is any product of input literals. A *literal* is a Boolean variable (say, *a*) or its complement (*a*').

Boolean function specified by *n* inputs. Assuming no bounds on the fan-in of the AND gates present in the first logic level is a restriction to avoid gate orphans. The canonical product terms are appropriately combined by OR gates (which are permitted to have bounds on fan-in) according to the function description – this forms the main functional part of the implementation. Since the AND gate indicates only when inputs become 'ones', separate OR-logic which performs a logical disjunction of all the dual-rail primary input signals is required in order to acknowledge when inputs become 'zeroes'. Therefore, when the outputs of the main functional part and the separate OR-logic are synchronised by means of C-elements, on the whole, the function block is said to *strongly indicate* the arrival of all the primary inputs. On the other hand, if any one dual-rail output (i.e. a true and false output) of the main functional part is synchronised with the output of the extra OR-logic by means of two C-elements, then all but that dual-rail output may become defined/undefined before all the primary inputs have become defined/undefined. In this case, the function block satisfies Seitz's weak-indication timing constraints and is therefore said to be *weakly indicating*. Weakly indicating realisations permit logic optimisation, and incur more area compared to a strong-indication version for the functional part due to more product terms but require fewer C-elements for synchronisation purpose. Also, such realisations pave the way for reducing the overall system latency if a number of subsystems are connected in a linear cascade, assuming only certain outputs would propagate as inputs between the subsystems. A good example of this would be a conventional ripple carry adder (RCA) consisting of a cascade of full adder modules, with the carry output of a less significant adder module serving as the input carry for a more significant adder module. The strong and weak-indication realisations of a full adder based on Seitz's approach are shown in figures 3.2 and 3.3 respectively. The fundamental equations governing a full adder with dual-rail inputs ($a1$, $a0$), ($b1$, $b0$), ($cin1$, $cin0$) and dual-rail outputs ($Sum1$, $Sum0$), ($Cout1$, $Cout0$) are as follows:

$$Sum1 = a0b0cin1 + a0b1cin0 + a1b0cin0 + a1b1cin1 \qquad \textbf{(3.1)}$$

$$Sum0 = a0b0cin0 + a0b1cin1 + a1b0cin1 + a1b1cin0 \qquad \textbf{(3.2)}$$

$$Cout1 = a0b1cin1 + a1b0cin1 + a1b1cin0 + a1b1cin1 \qquad \textbf{(3.3)}$$

$$Cout0 = a0b0cin0 + a0b0cin1 + a0b1cin0 + a1b0cin0 \qquad \textbf{(3.4)}$$

The weak-indication adder takes into account the fact that the output carry of an adder module could be defined as soon as its input operands become defined, depending on carry-kill ($a0$=$b0$=1) or carry-generate ($a1$=$b1$=1) conditions. Thus, the carry output equations can be optimised as:

$$Cout1 = a0b1cin1 + a1b0cin1 + a1b1 \qquad \textbf{(3.5)}$$

$$Cout0 = a0b1cin0 + a1b0cin0 + a0b0 \qquad \textbf{(3.6)}$$

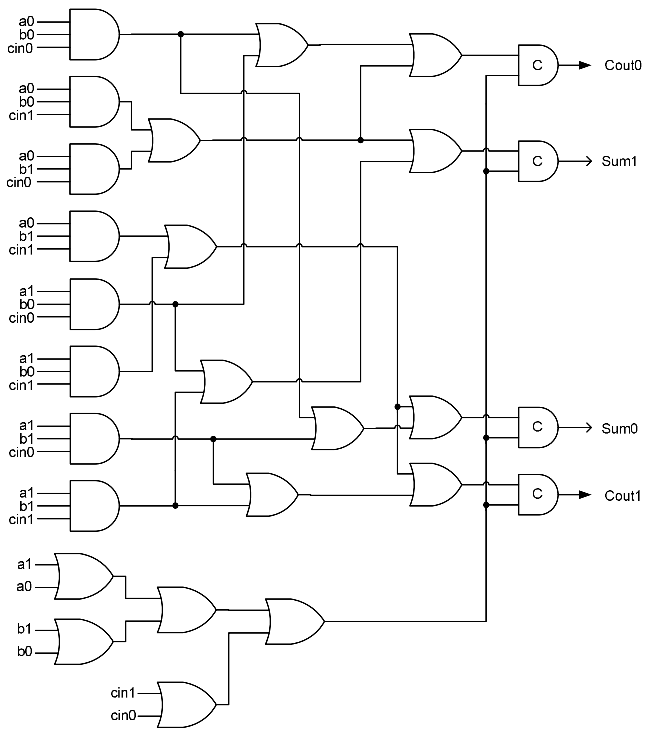


Figure 3.2: Seitz's strong-indication full adder

Comparing the adder circuitry depicted by figures 3.2 and 3.3, it can be noticed that the strong-indication adder increases the datapath delay, while the weak-indication adder incorporates a fast carry propagation path as only the sum output depends on all the inputs, while the carry output do not always, i.e. the indication is distributed between the function outputs in case of the latter realisation. It is to be noted that the completion detection logic, when not implemented as a single OR gate due to fan-in restrictions of the cell library, would necessitate timing assumptions.

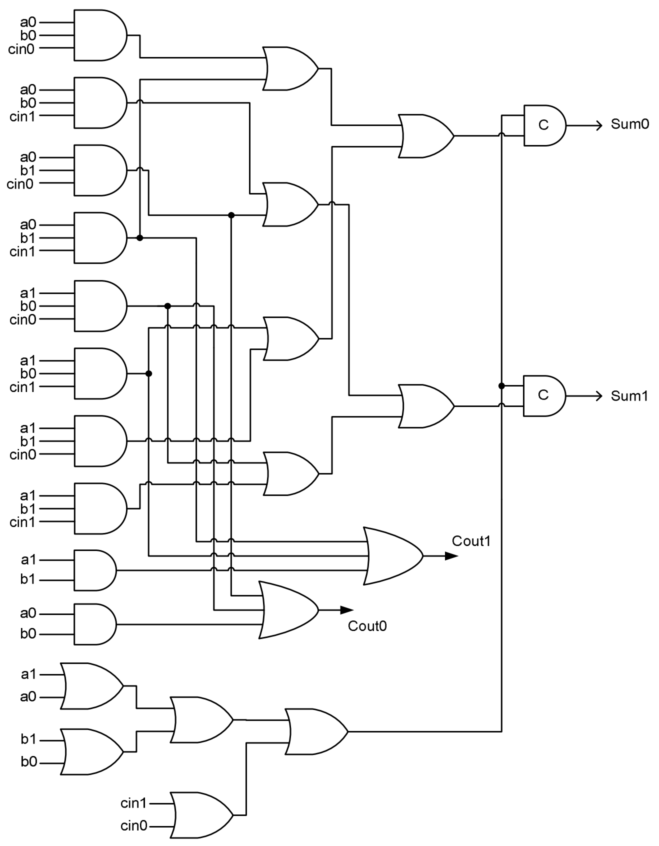


Figure 3.3: Seitz's weak-indication full adder

3.2 Singh's Approach

Singh's approach [97] targets ST implementation of the desired functionality by first partitioning the entire input space, constructing smaller modules for all the different partitions and then combining all those smaller modules appropriately in order to realise the required logic. This gives rise to decompositions of fine granularity restricted to minimum fan-in and eventually increases the logic depth. With an increase in the number of inputs, the number of partitions would increase considerably and, given the extensive usage of state-holding elements to perform logical conjunctions, it would exacerbate the area overhead for even medium sized functions besides degrading the delay metric. Also, no clear method has been portrayed for SI decomposition of function blocks with many inputs. An efficient implementation of a 4-input AND gate based on this approach is shown in figure 3.4, where (*a*1, *a*0), (*b*1, *b*0), (*c*1, *c*0), (*d*1, *d*0) are the dual-rail inputs and (Y1, Y0) serves as the dual-rail output. Multiple acknowledgements might result on some wire forks, which may be useful in simplifying the isochronicity assumption.

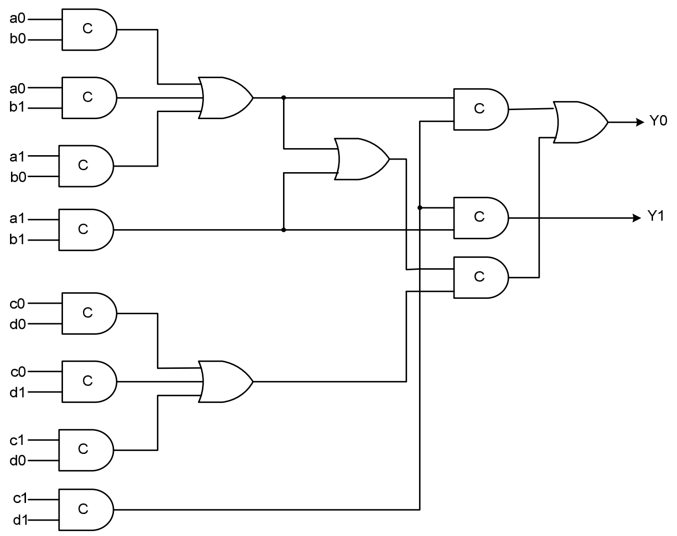


Figure 3.4: Realisation of a 4-input AND gate based on Singh's approach

3.3 Direct Logic and Reduced Direct Logic Styles

Among the four transistor level full custom function block styles proposed in [98], namely static logic, direct logic, semi-controlled precharge logic and full-controlled precharge logic, the direct logic can be classified as DI (QDI) as arbitrary delays on the input wires of a gate do not give rise to stale data values. The direct logic realisation technique basically considers a merger of C-elements and OR gates functionality. In other words, function blocks based on the direct logic style incorporate full custom complex gate constructions for both the true and false outputs. The transistor level realisation consists of pull-down and pull-up networks, made up of nMOS and pMOS transistor stacks respectively for both the output rails of the dual-rail output. The nMOS transistor stack corresponding to both the outputs comprises a unique path for each of the 2^n minterms related to '*n*' inputs of the function block, wherein several signal paths could share the same transistors. The path between output and ground is established by the nMOS transistor network that is used for indication of the spacer to valid data changes on all the inputs. The pMOS network consists of a series stack of 2*n* transistors that establishes a signal path between the supply and output when all the inputs become spacers, and is used to indicate the valid to spacer data signal changes on all the inputs. Therefore two identical pMOS transistor networks are required to indicate the RTZ phase of the inputs on the dual-rail output. For functions with multiple outputs, separate function blocks need to be constructed for each of the individual outputs (which are also dual-rail encoded). The direct logic implementation satisfies Seitz's strong-indication specifications as the outputs are computed only after the arrival of all the inputs. A major limitation of this function block implementation style being the size of the transistor stacks (especially, pMOS transistor network), as the delay associated with the reset phase is heavily influenced by the size of the pMOS transistor stack that offers a higher load with even three or four primary inputs; the nMOS transistor stack adds a large parasitic component to the delay. A 2-input AND gate based on the direct logic realisation style is shown in figure 3.5, where *a*1, *a*0, *b*1 and *b*0 are the inputs, while (Y1, Y0) signifies the dual-rail output.

In case of the reduced direct logic method, the responsibility of indication of input variables is distributed between the output variables. In other words, the outputs are collectively responsible for indicating all the inputs. The motivation for this is to reduce the complexity associated with the direct logic and to relax the strong-indication constraints so as

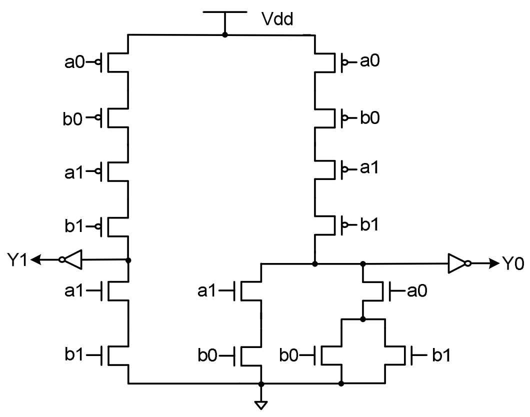


Figure 3.5: 2-input AND gate based on direct logic style

to make the function block weakly indicating. Martin's full adder design [99] corresponds to the reduced direct logic style and is shown in figure 3.6. (*a*1, *a*0), (*b*1, *b*0) and (*cin*1, *cin*0) are the dual-rail inputs, while (*Sum*1, *Sum*0), (*Cout*1, *Cout*0) are the dual-rail outputs.

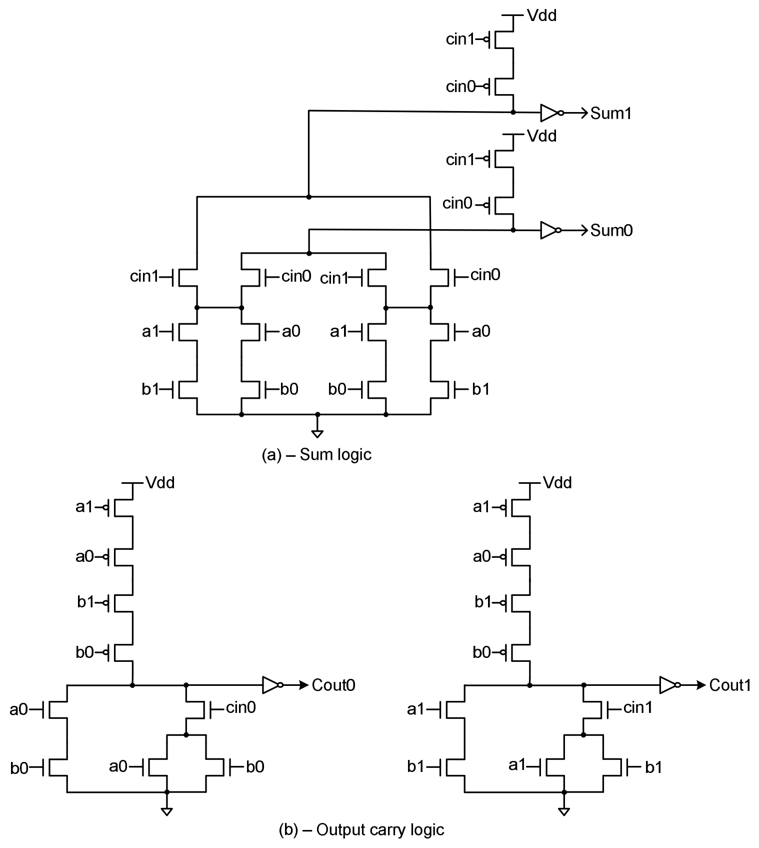


Figure 3.6: Martin's full adder (reduced direct logic style)

Seitz's weak-indication timing regime requires that at least one function block output (i.e. an encoded rail of any output) should not become defined (undefined) unless all its inputs have become defined (undefined). Here, it can be seen that the indication of (*a*1, *a*0), (*b*1, *b*0) and (*cin*1, *cin*0) when becoming a spacer (i.e. during the RTZ phase), is distributed between the sum and carry outputs, while in case of the adder based on the direct logic style, such distribution could not be found as it is strongly indicating. The carry output circuitry is based on majority logic that asserts the sufficient arrival of any two of the three inputs during the set phase, while the sum circuit asserts the arrival of all the inputs. The transistor count for the reduced direct logic based full adder is 42, as opposed to 60 transistors for the direct logic based full adder, thus effecting a savings in device count by 30% whilst enabling reduction in delay, as the former exhibits actual case latency when adding valid data and propagates empty values in constant time; the full adder based on direct logic style features constant worst-case latency in both the scenarios.

3.4 Delay Insensitive Minterm Synthesis Technique

The delay-insensitive minterm synthesis (DIMS) technique [100] is identical to the Seitz's approach discussed earlier in the sense that it requires listing of all the 2^n canonical product terms of the Boolean function governed by '*n*' inputs. The canonical product terms are OR-ed together according to the function description. However, instead of realising the product terms by AND gates, they are realised using C-gates. In its actual form, it can be envisaged as a two-level implementation consisting of C-gates in the first logic level and OR gates in the second logic level. The C-elements are assumed to have unbounded fan-in, while the OR gates can be decomposed arbitrarily as, at this stage there is a one-hot code representation, since for every function input only one C-gate (which represents a canonical product term) would get activated during the set phase. Similar to Seitz's approach, the DIMS technique also assumes unbounded fan-in for the gates present in the first level and therefore a naive decomposition based on the associative axiom could lead to hazards (creation of gate orphans) as illustrated in figure 3.7(a), while on the contrary merging could eliminate hazards as depicted by figure 3.7(b). Referring to figure 3.7(a), given the input transitions ($a\uparrow$, $b\uparrow$), only X↑ results while the steady state of Y is maintained. Therefore, the transition on the intermediate node X is not reflected on the output Y, which is construed as a gate orphan. Considering figure 3.7(b), given the input sequence ($a\uparrow$, $b\uparrow$), the steady state of Y is maintained and unless $c\uparrow$ occurs, Y↑ does not occur thereby avoiding gate orphans.

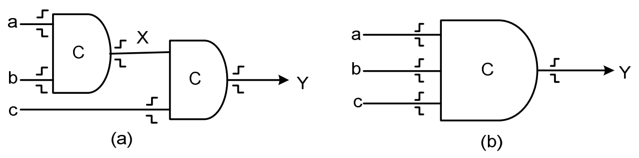


Figure 3.7: Hazards due to naïve decomposition of a C-gate

In general, reduction of Boolean equations is not allowed in the DIMS approach as it could violate the cover constraint. The function block constructed with the outputs expressed in disjunctive normal form utilising all the minterms is strongly indicating, as none of the outputs would become defined/undefined until all the inputs have become defined/undefined. The DIMS approach is similar to an earlier work by Anantharaman [101], but has been subsequently extended into a standard technique for implementation of arbitrary multiple output function blocks in [100]. The strong-indication full adder realised according to the DIMS approach is shown in figure 3.8.

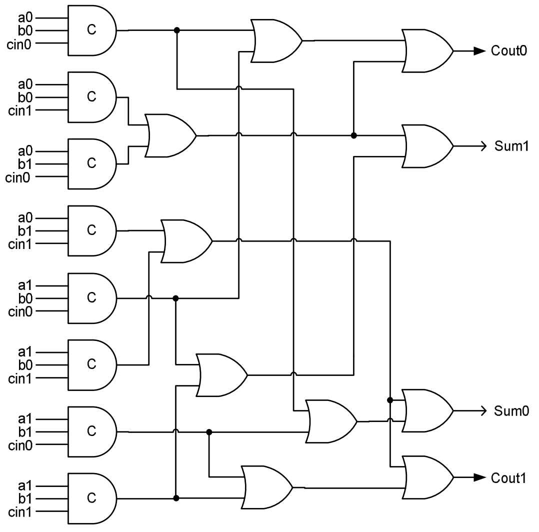


Figure 3.8: Strong-indication full adder based on DIMS approach

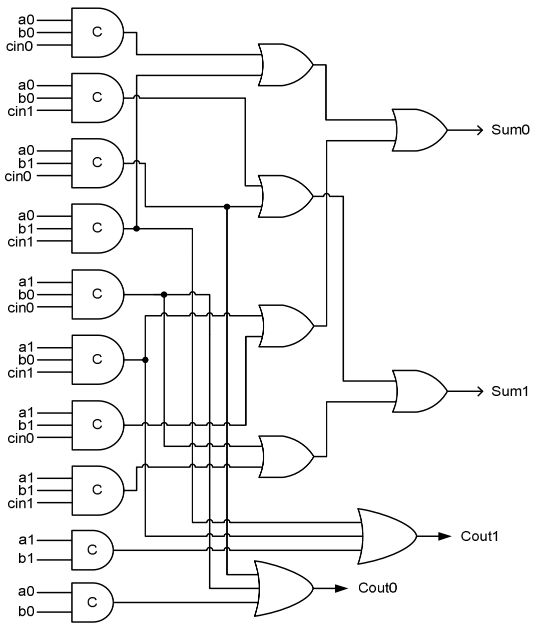


Figure 3.9: Weak-indication full adder based on DIMS approach

Isochronic fork assumptions are made with regard to the primary inputs as they feed many C-gates, while the forks that feed the OR-gates need not be isochronic. Similar to the case of Seitz's weak-indication adder, the carry-kill and carry-propagate conditions can be enforced thereby making the function block weakly indicating, as shown in figure 3.9.

3.5 Dual-Rail Combinational Logic

The dual-rail combinational logic (DRCL) style utilises De-Morgan's theorems of Boolean algebra to implement a combinational logic function in an asynchronous style by replacing each gate by its dual-rail equivalent (dual-rail pair). DRCL, as the name implies, is suitable for translation of synchronous circuits into asynchronous circuits based on the DI dual-rail data encoding protocol alone and is not generic. The expression for the false output of a logical operator is derived from the complement of the Boolean equation corresponding to its true output, expressed in sum-of-products form. Thus this approach could harness the strength of traditional synchronous logic design. The aim of this style is to facilitate asynchronous logic realisation using conventional logic gates, which are available as standard cells, thereby reducing the area expense. When such discrete gates are used, it is important to ensure the completion of computation at the internal nodes of the realisation apart from guaranteeing the complete arrival of all the inputs. We consider two scenarios for the DRCL equivalent of a Boolean function, say, F = $ab + cd$, as shown in figure 3.10, to clarify the necessity for ensuring proper indication of signal events at the primary inputs as well as intermediate output nodes and to describe how wire and gate orphans could possibly result.

1. Assuming all the data inputs to be currently spacers, when $a0$ and $c0$ become defined intermediate signals $x0$ and $y0$ would become defined and eventually F0 would be defined. Assuming that $b0$ and $d0$ also become defined subsequently, they would not be acknowledged by the intermediate signals or by the corresponding output in the present evaluation phase resulting in wire orphans.

2. Let us assume that $a1$ and $b1$ become defined after a RTZ phase. This would lead to defining of the intermediate signal $x1$. Assuming that $c1$ and $d1$ become subsequently defined during the current evaluation phase, F1 could have become defined as a result

of *x*1 alone becoming defined and hence a late transition on *y*1 would not be acknowledged by the primary output giving rise to a gate orphan.

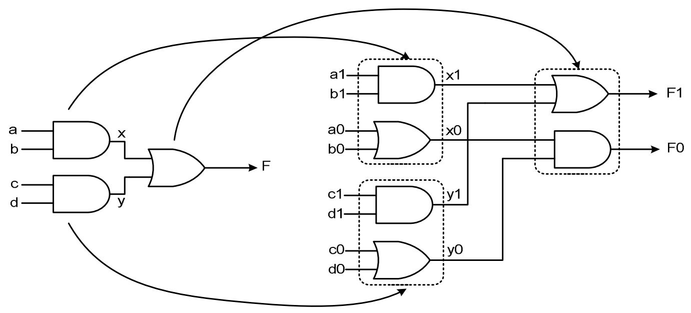


Figure 3.10: DRCL realisation of F = *ab* + *cd*

From the above discussion it becomes clear that the DRCL scheme is basically *non-indicating* and, as such, it generally conforms to *eager evaluation* owing to the fact that even with a subset of the function block inputs becoming defined, all the outputs could become defined regardless of the lately arriving inputs. Hence, it does not adhere to strong or weak-indication timing constraints. The NCL_X approach discussed in section 3.10.2 utilises the DRCL style for implementing the functional part and deals with ways to eliminate the problems of wire and gate orphans through the provision of explicit completion detectors.

3.6 David et al.'s Approach

David et al.'s method [103] consists of deriving the Boolean expression for the false rail[5] of a function output by complementing the logical equation corresponding to its true rail. With respect to the implementation, four sub-networks are involved – ORN, CEN, DRN and OUTN. The ORN (OR-gates subnet) consists of *n* two-input OR gates, where '*n*' refers to the number of single-rail primary inputs. Each 2-input OR gate is used to logically sum the signal values of both the rails of a dual-rail input in order to detect the proper arrival of valid or spacer data. The CEN is basically a multiple-input C-element, which is used to synchronise

[5] 'F1' and 'F0' correspond to the true rail and false rail of a dual-rail encoded function output 'F'. They are also called 'true' and 'false' outputs of a function block.

the outputs of all the 2-input OR gates to detect the proper and complete arrival of all the inputs during both the set and reset phases. When all the dual-rail inputs are driven to the spacer state, the outputs of all the OR gates in the ORN would become low and the simultaneous reset of all the inputs would be confirmed by the output of CEN. Similarly, when valid data is impressed upon the dual-rail inputs, the outputs of all the OR gates in the ORN would transition and the transitions on all the OR gates would be subsequently acknowledged by a transition on the CEN output signal.

The DRN represents the dual-rail network that implements the desired combinational logic functionality. It is a *monotonic network* and monotonicity is achieved by ensuring that when all the inputs are undefined, all the outputs are reset and during the *defining interval*, an output should have changed at most once. The defining interval is specified as the interval between the time instances when all the inputs are undefined to the time instance when all the outputs (i.e. any one rail of each dual-rail output) become defined. The DRN is typically composed of a two-level AND-OR realisation for the true outputs and a two-level OR-AND realisation for the false outputs. Of course, the second level OR gates in the AND-OR network can be decomposed arbitrarily as only one of its inputs would experience a transition. No specialised decomposition strategy has been formulated as part of this approach and it does not require the listing of all the unique input combinations, thereby paving the way for reducing the area overhead considerably in comparison with many other earlier approaches; however, this conclusion is actually dependent on the function specification. The size of the combinational logic realisation would be limited by the fan-in of the basic gates available in the cell library in a practical scenario. Though the method hints at employing conventional logic minimisation algorithms for realising the DRN, it could possibly give rise to gate orphans, especially because of the OR-AND logic. With respect to the AND-OR logic realising the true function outputs, gate orphans could be avoided only by imposing no bounds on the fan-in of AND gates as random decompositions could easily lead to gate orphans. Even with an unbounded fan-in relaxation, the OR-AND logic which realises the complementary function outputs could still give rise to hazards. This will be described shortly with an example. The logic implementation for the false function output is complementary to that of its true output, i.e. to say the DRN implementation utilises the DRCL style.

OUTN refers to the output subnet, which is composed of state-holding elements that are used to synchronise the outputs of CEN and DRN. OUTN would require a 2-input C-element for each rail of the output produced by the DRN and is mainly meant to retain the DRN outputs until all the inputs become defined (undefined) in the set (reset) phase in order to maintain compatibility with the proper sequencing of events in relation to the external environment (i.e. with the inputs fed from and the outputs fed to the environment in a sequence according to the handshake protocol). Though the method may appear to enforce strong-indication property, it suffers from the drawback that the implementation of the underlying combinational logic functionality (as part of the DRN) may not be SI and therefore not ST in a strict sense. This reasoning is substantiated by the following example.

Let us consider a logic function specified by X = $a'b'c + abc'$ and Y = $a'bc' + ab'c$. The equations for the true and false function block outputs in dual-rail format are given by:

$$X1 = a0b0c1 + a1b1c0 \quad \textbf{(3.7)}$$

$$X0 = (a1+b1+c0)\,(a0+b0+c1) \quad \textbf{(3.8)}$$

$$Y1 = a0b1c0 + a1b0c1 \quad \textbf{(3.9)}$$

$$Y0 = (a1+b0+c1)\,(a0+b1+c0) \quad \textbf{(3.10)}$$

The realisation of the above functionality according to David et al.'s approach is depicted by figure 3.11. The different sub-networks (ORN, CEN, DRN and OUTN) are highlighted in the diagram. It is to be noted that IX1, IX0, IY1 and IY0 are logically equivalent to X1, X0, Y1 and Y0 respectively. The isochronicity assumption can be extended to all the forks associated with the primary inputs. When the true outputs are asserted 'high', the indication of internal signals is proper as the AND gate waits for all its inputs to become 'high' before producing a 'high' output. The critical issue is concerned with the indication of internal signals *n*2, *n*3, *n*4 and *n*5, which are the intermediate outputs of the OR-AND logic, and towards this end we first consider an instance.

Let us assume that the false function outputs (X0, Y0) have become defined after feeding in appropriate valid input data (say, *a*1 = *b*1 = *c*1 = 1). This would have been possible with the internal signals (*n*2, *n*3, *n*4 and *n*5) experiencing transitions. When spacer data is applied to the circuit during the RTZ phase, even with *a*1 and *b*1 being reset, *n*2 and *n*5 are reset and the false function outputs could evaluate to the correct spacer state irrespective of the reset of the remaining internal signals (*n*3 and *n*4), implying that the outputs have not indicated the attainment of the correct steady-state value in all the internal nodes. From this, we understand that the circuit exhibits the phenomenon of *early reset* (i.e. outputs being reset in an eager fashion with only a subset of input signals), and it mainly results due to the fact that bounds are associated with gate delays (especially those of the two-level OR-AND logic). Hence, it can be concluded that the DRN subnet is not SI, as the SI timing model specifies unbounded delay for logical operators. On the contrary, if timing assumptions could be simplified by assuming that both *n*3 and *n*4 would be reset simultaneously similar to that of *n*2 and *n*5, then the circuit operation would be correct. Nevertheless, such a timing assumption would only add to the complexity of the verification process at the layout level.

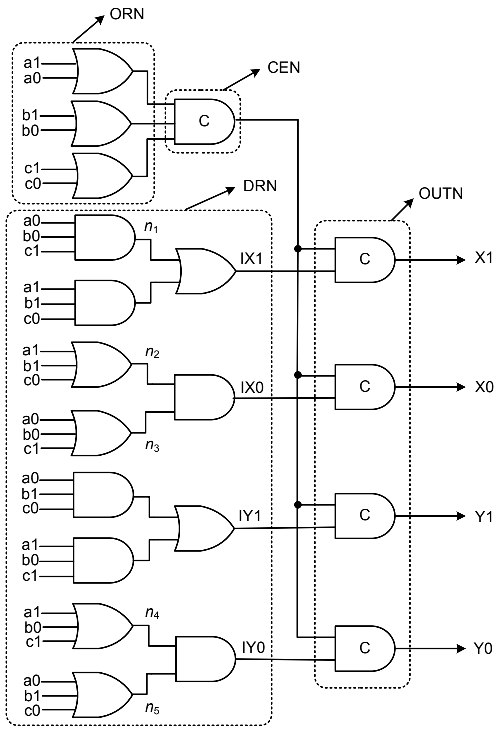


Figure 3.11: Realising X = *a'b'c* + *abc'* and Y = *a'bc'* + *ab'c* using David et al.'s method

Realising the false function outputs of the DRN subnet using OR-CE logic instead of OR-AND logic may be a simple solution to foster better synchronisation between events. However, this may not be sufficient to ensure gate-orphan freedom within the DRN subnet. To comment on this, let us consider another scenario by assuming that $a0 = b0 = c1 = 1$, after a RTZ phase. In this case, $n1\uparrow \rightarrow$ IX1↑, but the transition on $n3$ will not be followed by a transition on IX0 resulting in a gate orphan. The two scenarios considered thus far demonstrate how gate orphans could inherently manifest in the circuit, which consequently affects its robustness characteristic.

To resolve the above problem of eager reset, a novel solution is proposed in this thesis. The false function outputs typically employ two-level OR-AND logic and the AND gates cannot be decomposed in a random fashion without avoiding gate orphans as described earlier. However, if the OR-AND logic can be realized using complex gates via cell merging as opposed to discrete gates, as shown in the DRN subnet of figure 3.12, the timing assumptions could then be simplified. Further to this, a relaxation can be allowed such that the output of CEN need not be synchronised with all the dual-rail outputs of the DRN, but rather with only a single output (say, X or Y). With these modifications, the circuit would now feature the property of weak-indication and be robust, eliminating the need for sophisticated timing assumptions. The resulting circuit based on the above modifications is shown below.

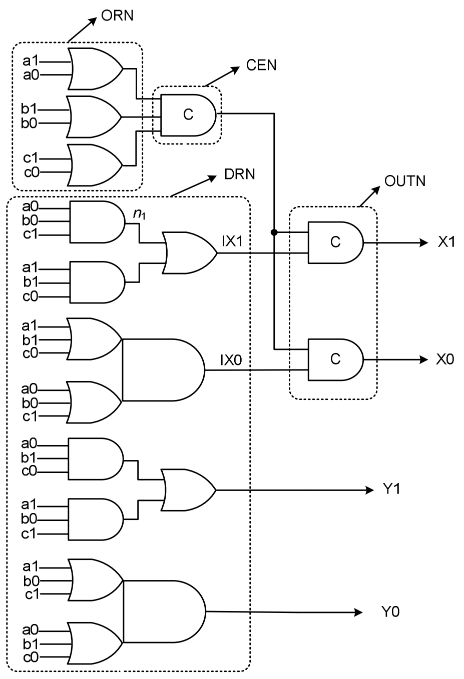


Figure 3.12: Weakly indicating realisation, based on modifications to David et al.'s method

By referring to figure 3.12, it can be seen that even with a subset of DRN inputs becoming undefined (say, *b*1 = *c*1 = 0), given the application of input data in the earlier phase as *a*1 = *b*1 = *c*1 = 1, IX0 and IY0 of the DRN would become undefined. Subsequently, with *a*1 becoming undefined, though its reset may not be indicated by the DRN, with isochronicity assumption imposed on the primary function block inputs, the CEN output confirms the reset of *a*1 and X0 and Y0 would then be reset. With respect to the second scenario considered previously, it should be obvious that a gate orphan does not arise since there are no intermediate nodes with respect to the OR-AND logic. Thus, proposing cell merger and synthesis of logic using complex gates to achieve a robust asynchronous circuit realization, have solved the problem of circuit orphans.

However, it may be observed that even with the proposed modifications the circuit would be power-hungry, as when valid data is applied the cover constraint can be imposed only on the AND-OR logic of the DRN subnet realising the true outputs. All the OR logic corresponding to the compound implementation of OR-AND, which realise the false outputs, would have to transition in order to produce valid outputs, based on appropriate inputs. Consequently, the dynamic power dissipation would be high due to greater switching activity. In addition, the need for complex gates to realise the false outputs of the DRN subnet could render the implementation practically infeasible with regard to modern standard cell libraries, due to the requirement for gates with a large fan-in and featuring a sophisticated functionality.

3.7 Toms' Approach

Toms' procedure for SI synthesis of combinational logic circuits [104] is based on utilising the techniques proposed for multi-level logic synthesis [105], such as extraction of single-cubes[6] and multiple-cubes (must be a sum of two or more cubes) by means of solving the rectangle covering problem, which is actually based on a very efficient sparse-matrix representation developed by Rudell [106]. The extracted cubes have to be then re-substituted as intermediate variables into the original expressions. While *extraction* is the process of identifying and creating some intermediate common sub-functions and variables, *substitution* (also known as

[6] A cube is a product of different literals, where a literal refers to a variable appearing in its normal (*x*) or complementary form (*x'*).

re-substitution) is the process of substituting a function X into a function Y such that Y is expressed as a function of its original inputs and X. Both these operations use techniques that are analogous to Boolean multiplication and division. In fact, 'division' plays a key role in multi-level logic optimisation. There are two types of division operations viz. algebraic and Boolean. Algebraic division, also known as weak division is faster in comparison to Boolean division (strong division), which is algorithmically more complex but capable of producing better results. In general, algebraic methods are fast because the logic function is treated as a polynomial and hence fast methods of manipulation are available [107]. Boolean factoring is generally non-polynomial and such procedures usually involve complexity [108].

Given two logic functions, say f and g, if there is an operation which generates expressions h and r, such that $f = gh + r$, where gh is an algebraic product (i.e. g and h have no common variable between them or disjoint support), then this operation is referred to as algebraic division. 'g' is referred to as the quotient and 'r', the remainder of the division operation. For example, if $f = abd + bcd + a'c + b'd'$ and $g = a + c$, the algebraic division could then yield

$$f = gh + r = bd(a+c) + a'c + b'd' \qquad \textbf{(3.11)}$$

On the other hand, Boolean division uses the identities of Boolean algebra for factorisation[7] of logic expressions (For example, $yy' = 0$, $yy = y$ and $y+y' = 1$ for a variable y). Thus, if in the expression $f = pq + t$, pq is a Boolean product (i.e. when p and q do not have disjoint support), then the division of f by p is called a Boolean division. This division operation could yield the following expression.

$$f = pq + t = (bd+a')\,(a+c) + b'd' \qquad \textbf{(3.12)}$$

The quotient resulting from an algebraic division of an expression, say Z, by a cube, say c (i.e. Z/c) is called the *kernel* of Z, if there are at least two cubes in the quotient and the cubes are governed by a disjoint support. The notion of *kernel* of an algebraic expression was introduced by Brayton et al. in [109], as a means of finding sub-expression(s) that are common to two or more expressions. In fact, all operations used to find kernels are algebraic. The cube divisor c, used to obtain the kernel is called the *co-kernel*. Different co-kernels may produce

[7] Factoring is the translation of a function expressed in SOP or disjunctive normal form into a parenthesised form having a minimum number of literals.

the same kernel: hence the co-kernel of a kernel is not unique. If a kernel has no kernels except itself, it is said to be a level-0 kernel. A kernel is said to be of level *k*, if it has at least one level (*k*-1) kernel but no kernel except itself, of level *k* or greater. Let us consider the following function,

$$Z = abcd + adg + b'dfg + b'cdef \tag{3.13}$$

The quotient obtained from the division of Z by the cube *ad* is

$$Z/_{ad} = bc + g \tag{3.14}$$

The quotient resulting from the division of Z by *b'f* is

$$Z/_{b'f} = dg + cde \tag{3.15}$$

$Z/_{ad}$ is a kernel of Z, since it has two cubes and they are cube-free – therefore *ad* is a co-kernel of Z. But $Z/_{b'f}$ is not said to yield a kernel since variable *d* is common to both the cubes of the quotient, resulting from the division operation.

In conventional multi-level logic synthesis, single-cube extraction is referred to as *condensation* and multiple-cube extraction is referred to as *distillation*. In general, the *distill* algorithm is preceded by a kernelling algorithm, where a larger subset of algebraic divisors is generated. Using these divisors, *distill* performs multiple-cube decomposition and factorisation. This is followed by the *condense* algorithm, which performs single-cube decomposition. Often, the heuristic would involve several iterations of each round of extraction and re-substitution. However, there could be cases when divisions could not extract multiple-cubes or even single-cubes and these depend upon the logic functionality. Toms' method [151] is based on performing distillation and/or condensation operations in a SI fashion for multi-level synthesis of combinational logic, provided certain conditions of substitution are upheld [104]. The approach basically considers the entire input state space and therefore it would encounter the problem of input space explosion. The resulting solutions are strongly indicating to reduce the complexity of the substitution. It enables decomposition of cubes expressed in any *m*-of-*n* encoding style through a technology-independent synthesis involving 2-input C-elements and OR gates, with the assumption of isochronic forks, and thereby it can be classified as QDI. A dual-rail full adder synthesised using this approach is shown in figure 3.13. The primary multiple and single-cubes extracted are given by (3.20) – (3.22), while substitutions are visible in the remainder of the equations.

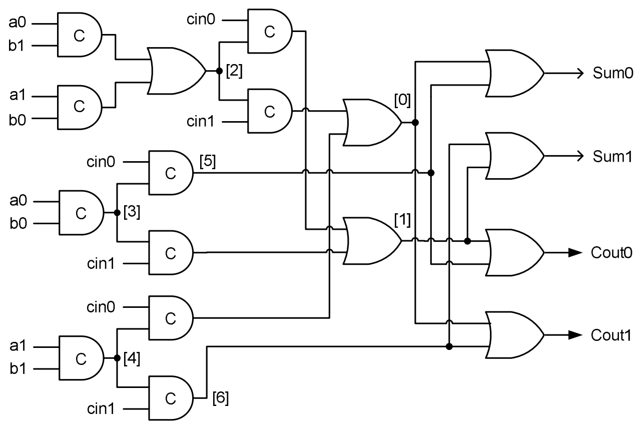


Figure 3.13: Full adder synthesised using Toms' approach

$$Sum0 = [0] + [5],\ Sum1 = [1] + [6] \qquad \textbf{(3.16)}$$

$$Cout0 = [1] + [5],\ Cout1 = [0] + [6] \qquad \textbf{(3.17)}$$

$$[0] = cin1[2] + cin0[4] \qquad \textbf{(3.18)}$$

$$[1] = cin0[2] + cin1[3] \qquad \textbf{(3.19)}$$

$$[2] = a1b0 + a0b1 \qquad \textbf{(3.20)}$$

$$[3] = a0b0 \qquad \textbf{(3.21)}$$

$$[4] = a1b1 \qquad \textbf{(3.22)}$$

$$[5] = cin0[3] \qquad \textbf{(3.23)}$$

$$[6] = cin1[4] \qquad \textbf{(3.24)}$$

3.8 Folco et al.'s Approach

Folco et al.'s approach [110] bears a similarity with the previous approach in the sense that the synthesis of combinational logic as QDI circuits is performed assuming 2-input C-elements and OR gates. However, the resulting circuits could satisfy strong or weak-indication constraints. This approach makes use of algorithms for constructing reduced ordered binary decision diagrams (ROBDD) [111] as the basis for its synthesis strategy with the exception that logical conjunctions are perceived as achieved through C-element functionality rather than AND gates. The technology mapping has been subsequently done targeting a 130nm standard cell library [112] that includes some custom asynchronous elements created on the basis of the STMicroelectronics CMOS process, following the structural pattern matching algorithm proposed by Zhao et al. [113].

To help with further discussion, some basic concepts of binary decision diagrams are first explained. The *binary decision diagram* (BDD), named so by Akers [114], after it was introduced as a binary decision program concept by Lee [115] to represent switching circuits, is a rooted directed acyclic graph (DAG) and is a canonical representation of a logic function [111]. It is typically constructed by a recursive application of Shannon's expansion theorem[8]. The BDD for a logic function generally has two terminal nodes of out-degree (fan-out) zero labelled '0' or '1' and a set of variable nodes of out-degree two. The nodes in the first level of the BDD are referred to as root nodes (*sources*) and the nodes that represent the constants ‘0’ and ‘1’ are referred to as terminal nodes (*sinks*). The rest of the nodes present between the sources and sinks are generally referred to as intermediate or non-terminal (*non-sink*) nodes. The two outgoing edges from a non-sink variable node are labelled as the 1-edge and 0-edge (directly arising from the assignment of the logical value of '1' and '0' to that variable), and may converge on two other distinct nodes that are referred to as its *child nodes*. The child nodes are said to be the *successors* (one-successor and zero-successor) of a parent node. A variable is associated with every node, excepting the terminal node, whose out-degree is zero. The BDD for the logic function F = ($p \oplus q \oplus r$) is given in figure 3.14(a). The 1-edges are drawn as solid lines and the 0-edges are drawn as dotted lines in the diagram. The dual-rail synthesised circuit is portrayed by figure 3.14(b).

[8] According to Shannon's expansion theorem, a function can be decomposed by means of a variable(s) of the function as $F = aF_a + a'F_{a'}$, where F_a and $F_{a'}$ are known as the positive and negative residues of the function, obtained by assigning binary values of '1' and '0' to the variable a of function F respectively.

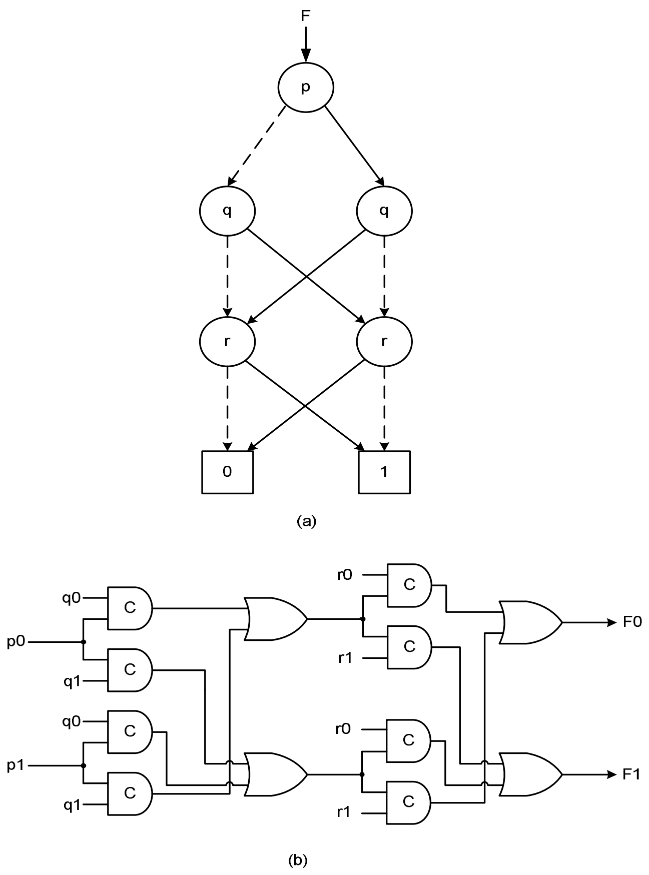


Figure 3.14: BDD and circuit solution for F = ($p \oplus q \oplus r$) based on Folco et al.'s method

A BDD is *ordered* (i.e. OBDD) if on all paths through the graph the variables follow a certain ordering. An OBDD is called *reduced* (i.e. ROBDD), if it incorporates the properties of 'uniqueness' and 'non-redundancy' [116]. The concept of ROBDD is traced back to the seminal work of Bryant in the 1980's [111]. Any completely or incompletely specified Boolean function has a unique ROBDD and therefore any other OBDD for the function constructed using a different order of variables would have more nodes. In other words, ROBDD is unique for a given logic function when the order of the variables is fixed [116]. Although the use of BDDs for synthesis of dual-rail QDI datapath circuits is helpful, it is to be borne in mind that it is generally difficult to find the best order for larger problems in a reasonable processing time. Dynamic variable ordering heuristics can be invoked for this purpose but may pose significant computational complexity, as it is not possible to efficiently compute an optimal variable ordering [117]. Also, for a system of inherently complex functions, construction of

BDDs may be impractical due to the exponential increase in computational complexity due to the size and/or variable ordering [111] [116]. When the notion of BDDs is extended to address multi-valued logic, they are referred to as multi-valued decision diagrams (MDD) [116]. The synthesis mechanism corresponding to this approach considers the use of reduced ordered MDDs for data encoding using a DI 1-of-*n* code, which boils down to the usage of ROBDDs for dual-rail data encoding. MDDs corresponding to datapath and CD logic are named as direct MDD and acknowledgement MDD respectively.

The synthesis of a dual-rail full adder shall be considered for the purpose of illustration. ROBDDs exhibit the notion of mutual-exclusion and this plays a vital role in realising QDI circuits, since the cover constraint would be inherently satisfied at the gate level. A set of BDDs representing many functions with the same variable ordering can be combined into a uniquely combined graph by transforming the individual BDDs which tend to have sharing of sub-graphs. Such a BDD is referred to as a *shared* BDD (SBDD) [118]. In general, by sharing all the isomorphic (similar) sub-graphs completely, no two nodes that express the same function co-exist in the graph. The use of SBDDs to represent the set of Boolean functions is helpful as it not only reduces the storage requirement for the nodes but also simplifies the equivalence checking of Boolean functions. Though the usage of SBDDs has not been explicitly mentioned as part of this approach, it might have been considered. However, care should be taken as translation of SBDD representation of a multi-output function into a ST circuit realisation could probably result in gate orphans. The full adder circuit taking into account logic sharing is shown in figure 3.15. It can be observed from the diagram that while the sum outputs depend on all the inputs for evaluation, the carry outputs need not as they utilise the carry-kill and carry-generate conditions. Thus the full adder is found to adhere to weak-indication constraints.

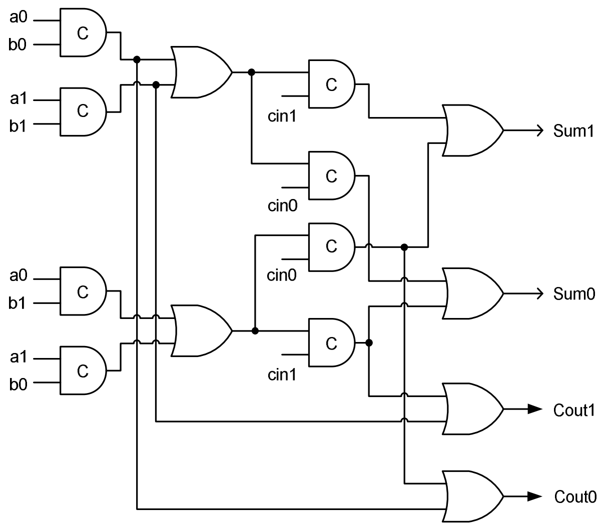


Figure 3.15: Full adder synthesised using Folco et al.'s approach

3.9 Circuits with Partial Acknowledgement

The DRCL style, discussed earlier, often exhibits *early propagation* at the gate level when producing a single dual-rail output, or at the block level when producing multiple outputs [119]. In case of early output logic [120], even when a subset of inputs becomes defined, all the outputs of the circuit could become defined (*early evaluation*) and/or even with a subset of inputs becoming undefined, all the outputs could become undefined (*early reset*), which is a characteristic inherent in the DRCL style. In comparison with the DIMS approach, the verification demand for checking the timing closure of inter-module wires is high in case of the DRCL style [121]. Despite this drawback, the DRCL style is appealing as it incurs only a minimum area expense (approximately 2× compared to an equivalent synchronous combinational logic, in the absence of local completion detectors). On the other hand, the DIMS approach always paves the way for robust evaluation and reset, meaning that it is not early propagative but suffers from a high area overhead. Therefore, a trade-off between robustness and circuit size is obvious between these two approaches. Circuits based on partial acknowledgement [121] basically correspond to weak-indication and try to utilise the advantage of DRCL and DIMS approaches. Each and every synchronous gate is replaced by either a robust asynchronous dual-gate pair equivalent or by an early propagative dual-gate pair equivalent, such that the circuit would be weakly indicating on an overall basis. This essentially means that inputs which are indicated by a gate output(s) in a robust fashion need not be indicated by other gates to which also it is fed in a robust fashion, thereby *relaxing* (weakening) the indication constraints for the remnant (i.e. inputs, which are fed to non-relaxed gates can also be fed to relaxed gates if necessary; unique inputs should not be allowed to be associated only with relaxed gates). Hence, this method inherently paves the way for distribution of acknowledgement of the circuit inputs so that all the circuit variables are covered by the outputs collectively. A similar approach appears to have been concurrently proposed in [122], but the difference between the two being that [122] performed mapping targeting NCL macros made available as part of a specialised standard cell library fully characterised for area and delay, provided by Theseus Logic Inc., evaluating the designs with respect to three cost metrics: number of relaxed nodes, area and maximum path delay, while [121] addressed area (transistor count) as the cost function considering pseudo-static implementations of the NCL operator macros [124]. A multi-output combinational logic function, consisting of four inputs (a,b,c,d) and three outputs (X,Y,Z), is used to explain this method from a gate level perspective. The logic realisation is given in figure 3.16.

$$X = ab \qquad \textbf{(3.25)}$$

$$Y = ab'c + bc'd \qquad \textbf{(3.26)}$$

$$Z = cd \qquad \textbf{(3.27)}$$

From the Boolean equations listed above, we consider an implementation method.

- If single-rail inputs (a,b) and (c,d) are considered to be uniquely associated with outputs X and Z respectively, they can then be partially acknowledged by output Y.

Given this scenario, outputs X and Z can be realised using the DIMS approach individually, while output Y can take advantage of the DRCL style. Hence, the input signals are collectively acknowledged by outputs X and Z in a robust fashion, while the logic for realising output Y corresponds to *eager evaluation* (i.e. it is early propagative).

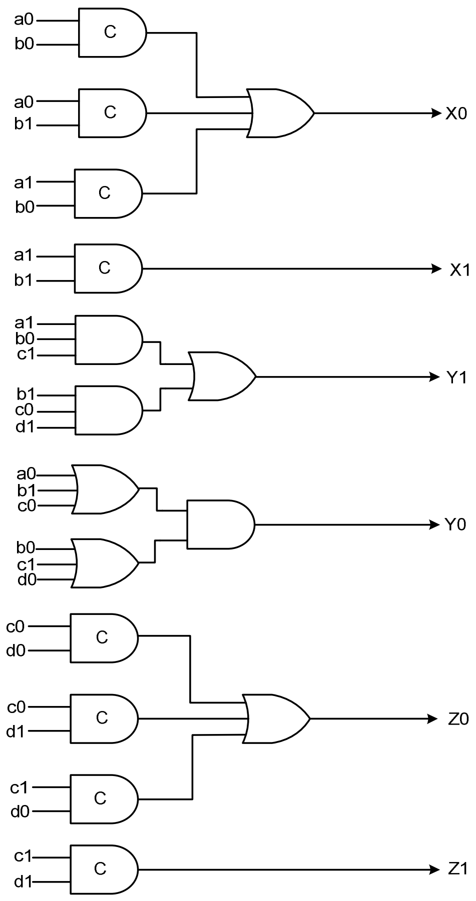


Figure 3.16: Implementing a multi-output circuit on the basis of partial acknowledgement

The number and type of logic operators required for the above realisation, taking cognisance of the elements available in the 130nm Faraday (UMC) CMOS commercial cell library[9] (along with the proposed semi-custom implementations of C-element functionality) is as follows: 8 CE2, 2 AND3, 1 AND2, 4 OR3 and 1 OR2.

3.10 NCL Based Methods

NULL convention logic (NCL)[10] utilises symbolic completeness of expression to achieve ST behaviour and was proposed as a consistent clock-free logic suitable for asynchronous digital circuit synthesis [77]. A symbolically complete expression depends only on the relationship of the symbols present in the expression without a time reference. In fact, DI data encoding

[9] AND gates and OR gates have maximum fan-in of 4 and 3 respectively in this standard cell library.
[10] K.M. Fant and S.A. Brandt, “Null convention logic system,” US Patent 5828228, October 1998.

schemes combine data and control information into one mixed signal path to eliminate reference to evaluation instants. Indeed, 'NULL' is identical to the spacer (empty data) in a typical 4-phase handshake protocol. A typical NCL circuit comprises a number of smaller function modules (also called 'primitive modules') interconnected together to achieve the desired functionality. The primitive modules (otherwise called as NCL macros[11]) are M-of-N threshold gates with hysteresis. Static, semi-static and dynamic implementations of arbitrary M-of-N gates along with the technique for reset initialisation are highlighted in [124]. An M-of-N gate operates on signals which could be either DATA (i.e. valid data) or NULL (i.e. spacer data) and exhibits *threshold*/*hysteresis* behaviour. The M-of-N gate output could transition to DATA, if M inputs out of a total of N inputs become DATA (threshold behaviour) but can attain the NULL state from a DATA state only after all the N inputs become NULL, i.e. the output remains at DATA until all the N inputs have become NULL (hysteresis behaviour). N-of-N and 1-of-N gates are special cases of the generic family of M-of-N threshold gates and they represent C-gates and OR gates respectively. Cases wherein M > 1 and M < N are unique, which require determination of the topology after possible logic optimisation. Due to the limitation imposed by the technology on the transistors stack size, the maximum value of N is restricted to at most 4 in most modern CMOS libraries and therefore gates with higher fan-in are decomposed into the form of multi-level circuit structures. Table 1 in [125] lists all the 27 distinct proprietary NCL macros (library elements) along with their Boolean equations that are used to realise NULL convention logic based ST circuits.

The static CMOS M-of-N implementation of a sample function, Z = $ab + cd$ is shown below in figure 3.17 for the sake of illustration. The implementation can be basically split into 4 blocks: 'Go to NULL', 'Hold NULL', 'Go to DATA' and 'Hold Data'. The 'Go to NULL' and 'Hold DATA' block topologies are a 'standard' and consist of a series arrangement of p-type transistors and parallel arrangement of n-type transistors respectively, where the number of p-type and n-type transistors are identical, governed by the function inputs. The 'Go to DATA' topology is constructed directly from the function, as logic optimisation is not possible, whereas the 'Hold NULL' topology is derived by considering the dual of the function as follows:

[11] K.M. Fant and G.E. Sobelman, "Null convention threshold gate," US Patent 5664211, February 1997.

$$Z' = (ab + cd)' = (a' + b')\,(c' + d') \qquad \textbf{(3.28)}$$

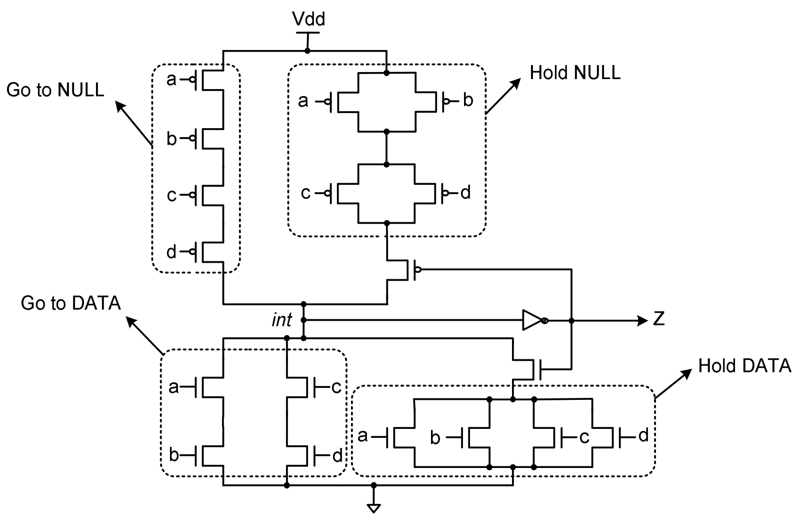


Figure 3.17: Static CMOS M-of-N implementation for Z = *ab* + *cd*

It could be observed from the circuit diagram that the topologies of 'Go to DATA' and 'Hold NULL' blocks are not identical and so the circuit does not feature the *self-dual property*, which is a special attribute exhibited in only a subset of all possible M-of-N threshold gate realisations (for example, majority logic). Analysing the operation, it can be seen that when all the inputs (here, N = 4) are NULL, the 'Go to NULL' and 'Hold NULL' blocks are ON whilst the 'Go to DATA' and 'Hold DATA' blocks are OFF and Z is driven to NULL. If any one of the inputs becomes DATA (say, signal *a* becoming '1'), the 'Go to NULL' block turns OFF but the 'Hold NULL' block would remain ON. However, this does not cause a change in the gate output, since a signal path still remains established between V_{dd} and the intermediate node '*int*'. Supposing, subsequently input *b* also becomes DATA, the 'Hold NULL' block turns OFF and the 'Go to DATA' block turns ON – thereby a signal path gets established between '*int*' and ground, resulting in a discharge of potential and as a consequence Z is forced to DATA. It may be seen that for Z to again switch state from DATA to NULL, the 'Go to NULL' block has to turn ON which is possible only if all the N inputs become NULL. From the preceding discussion, it becomes clear that Z can switch from NULL to DATA even with two function inputs (say, *a* and *b* or *c* and *d*) becoming logic '1', while Z can switch from DATA to NULL only if all the function inputs attain logic '0'.

Within the NCL paradigm, ST logic design methods were developed which start from a synchronous netlist. They mostly take advantage of the DIMS approach/DRCL style for obtaining the gate-level asynchronous equivalent followed by technology mapping targeting the NCL macros, which are made available as custom additions to a cell library. Two well-known methods and a recent approach proposed in this context are discussed further.

3.10.1 NCL_D Approach

The NCL_D technique was proposed by Ligthart et al. [126] as a regular method for NCL implementation by banking on the DIMS approach. It satisfies Seitz's weak-indication constraints and ensures input completeness [77], assuming isochronic fork assumptions. In its basic form, it derives the asynchronous equivalent of a synchronous circuit by first mapping the optimised synchronous network into two-input gates. Then, it represents each signal wire as a dual-rail pair and directly translates the two-input logic gates into threshold gate pairs with limited optimisation of a threshold network in order to preserve the DI property. A typical NCL_D system is shown in figure 3.18.

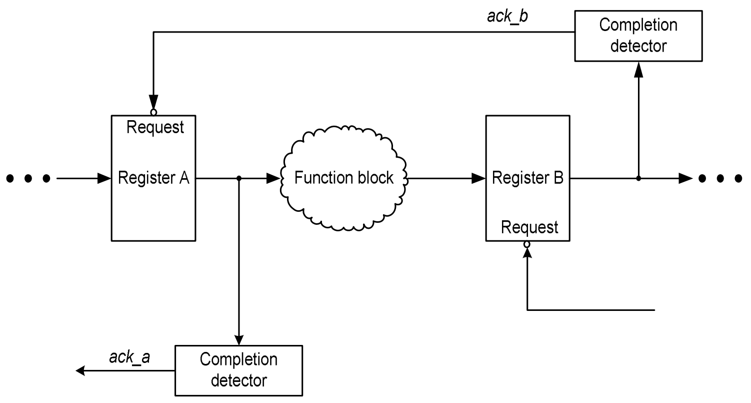


Figure 3.18: NCL_D system configuration [126] [127]

Let us assume that initially all the registers are in the NULL state and the acknowledge signals (*ack_a* and *ack_b*) are also low. Thus the request line of register A is active and is ready to accept new DATA. When the inputs become defined, the DATA wavefront is passed through register A onto the function block for processing and its outputs are fed as inputs to the next stage register B in the pipeline. The completion detector generating the *ack_a* signal performs the validity/neutrality tests of the input codeword during the set and reset phases respectively. Currently, it would check the validity of the DATA at its inputs and subsequently asserts the *ack_a* signal to logic high if the check is true. This signal disables the previous register's request line and prepares it for storing the next NULL wavefront. Collision between different DATA wavefronts are avoided by means of alternating reset and set phases (i.e. NULL-DATA-NULL). The main advantages associated with this approach are the simplicity of translation of a synchronous circuit into an asynchronous equivalent whilst paving the way for automatic verification of DI properties during implementation. This is because the gates used for realising the ST equivalent are input-complete, thereby proving that the circuits are correct by construction. As a result, though the robustness of the resulting circuits is high, unfortunately, due to ensuring locality of DI property verification and with little room for optimisation, it incurs high area overhead and the resultant circuits may be slow. Further optimisations to the NCL_D approach by means of cell merging and optimal technology mapping were suggested in [79]. The simple gate-level asynchronous equivalent of the example Boolean function considered previously is given in figure 3.19.

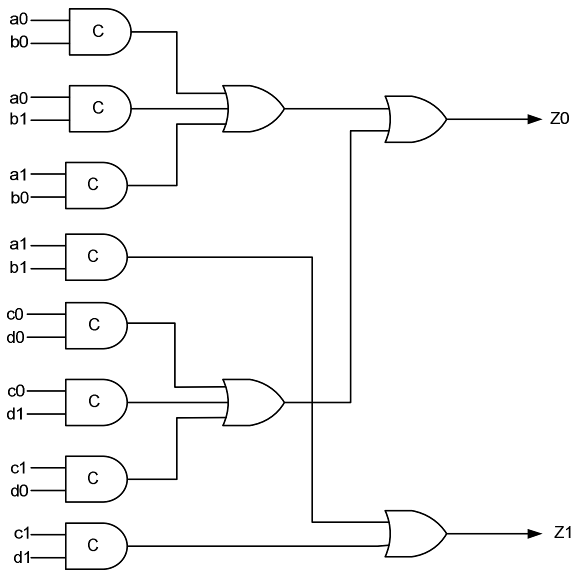


Figure 3.19: NCL_D based logic equivalent for the function, Z = *ab* + *cd*

3.10.2 NCL_X Approach

NCL design incorporating explicit completion detection is referred to as NCL_X, proposed by Kondratyev et al. [127]. A typical NCL_X system description is shown in figure 3.20.

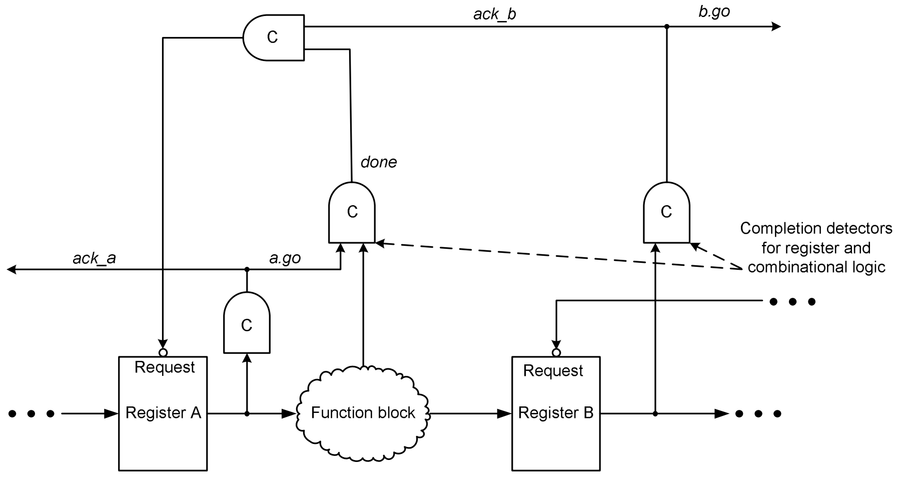


Figure 3.20: NCL_X system configuration [127]

Separate completion detectors are made available for the combinational logic and registers. It can be noticed that the NCL_X technique realises a NCL circuit using separate completion and functional parts allowing room for optimising each of them whilst guaranteeing completeness of inputs. The *a.go* signal basically corresponds to the completion detection of the input signals, which is synchronised with that of the functional part to generate the *done* signal. The NCL_X approach relies extensively on the DRCL style for implementing the functional part, i.e. input-complete gates are scarcely used. As a result, NCL_X circuits consume less area than their NCL_D counterparts. Because the DRCL style pertains to eager evaluation, gate and wire orphans could result. To eliminate the problem of gate orphans, provision of explicit local completion detectors (depicted by OR gates drawn using discontinuous lines in figure 3.21) is done in order to unambiguously ascertain the state of the internal nodes by means of OR-ing both the wires of each intermediate dual-rail signal [76] [102]. It can be intuitively observed that the total number of internal completion detectors required would be equal to the total number of gates in the original synchronous netlist which were subsequently replaced by a dual-rail gate pair equivalent in addition to the one associated with the primary output. This would eventually lead to more activity in the intermediate nodes, but the activity in the functional part gets offset in comparison with the NCL_D realisation. The problem of wire orphans with respect to the primary inputs in the functional part is eliminated by the acknowledgement of transitions on wires in the completion part with the assumption of isochronic forks. If certain input signals are fully acknowledged in the functional part, then they can be eliminated from the completion part to avoid redundancy in detection. With respect to power dissipation, both NCL_D and NCL_X approaches were found to result in similar figures [127]. The gate-level NCL_X equivalent for the example function considered previously is shown in figure 3.21.

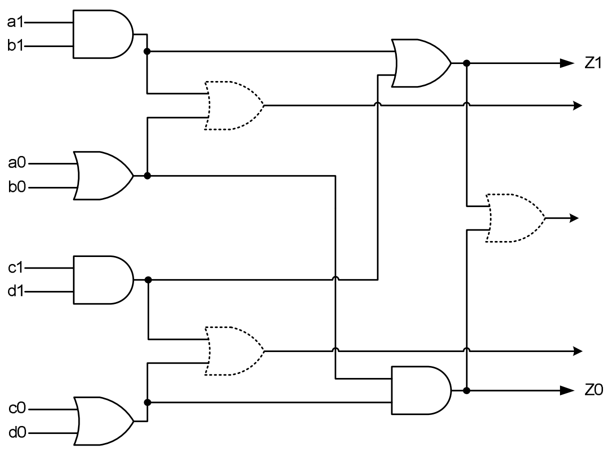


Figure 3.21: NCL_X equivalent for the function, Z = *ab* + *cd*

3.10.3 Block-Level Relaxation

The block-level relaxation technique [128] is an extension and generalisation of the gate-level relaxation approach to logic blocks with fewer inputs and multiple outputs [121] [122] by either distributing the responsibility of indication of all the inputs between different outputs (*distributive* implementation approach) or by confining the responsibility wholly to a single dual-rail output while relaxing the other outputs (*biased* implementation approach). Both these approaches actually correspond to different ways of achieving weak-indication. Input-complete and input-incomplete gates (circuits) are otherwise referred to as non-relaxed and relaxed gates (circuits). For example, the DIMS approach facilitates non-relaxed circuits while the DRCL style leads to relaxed circuit implementations. The approach elucidated in section 3.9 attempts to realise each and every gate in an input-complete fashion or in early propagative style and therefore it corresponds to gate-level relaxation, whereas the block-level technique is suitable for dealing with smaller logic blocks on an individual basis and is beneficial for iterative logic circuits which involves a cascade of similar logic elements probably allowing more optimisation opportunities on a holistic basis. It may be noticed that both gate-level and block-level relaxation approaches actually tend to incorporate the weak-indication property into the resulting circuit in order to reduce the datapath delay whilst ensuring input-completeness. Depending on the logic network specification, it may be that both these techniques might converge to similar synthesis solutions in cases. For example, in case of the sample multi-output function block considered in section 3.9, the block-level relaxation approach can also lead to a similar robust solution. Nevertheless, the mapping is performed targeting the NCL macros made available in a standard cell library.

3.11 Summary

A number of ST combinational logic realisation techniques have been presented and analysed in this chapter that spans nearly three decades of research activity in this domain. Broadly, three different directions pursued so far can be identified – i) ST logic implementation restricted to fewer inputs and outputs (Singh's, Seitz's, Anantharaman's and DIMS approaches), ii) ST logic design methods based on well-established synchronous synthesis concepts such as multi-level synthesis by considering the entire input space or utilising the concept of reduced ordered decision diagrams facilitating moderate reduction in input space, though they may not be scalable (Toms' and Folco et al.'s approaches respectively) and iii) ST logic realisation utilising DIMS and/or DRCL styles, but starting from an initial synchronous description (Ligthart et al., Kondratyev et al., Zhou et al., and Jeong et al.). Of these, the former two are technology-independent schemes while the latter methods are largely technology-dependent.

Seitz's and DIMS methods are generally useful for circuit conceptions at a theoretical level, of which the latter method has been popular owing to its robustness (QDI property) and no requirement of separate completion detectors. Toms' and Folco et al.'s procedures can be used for multi-level realisation of ST logic, but suffer from the problem of state explosion, which restricts their application to circuits with fewer inputs. Especially, in case of the former method, practical realisation of ST circuits comprising many inputs (greater than about 10 inputs) and/or outputs would not be feasible, due to the consideration of the entire input space. NULL convention logic based methods have been popular and within the ambit of NCL based design, synthesis of weak-indication circuits by exploring the opportunity for local relaxation appears to offer a good trade-off between robustness and design metrics optimisation, as they strike a balance between NCL_D and NCL_X approaches. The NCL_D is dependent on the DIMS approach while the NCL_X approach relies on the DRCL style, at the block level. In fact, the DIMS approach and the DRCL style have been an inspiration for a plethora of ST logic design methods. Circuits with partial acknowledgement try to utilise the advantages inherent in these two different methods, seeking to attain a compromise between robustness and area. This is due to the fact that NCL_D circuits are slower and occupy more area but are input-complete, while NCL_X style leads to faster circuit implementations whilst mitigating the area overhead but employ extra completion detectors. In terms of power dissipation, both NCL_D and NCL_X methods are likely to be on par with each other since the power saved through compact logic realisations in case of the latter compared to the former tends to diminish due to the power dissipation of the internal completion detectors.

This thesis envisions a whole new multi-level synthesis strategy for weakly indicating realisations of arbitrary combinational logic and towards this end a novel two-level synthesis procedure is presented in the next chapter that restrains the state space expansion considerably. Initial insights into a tangible solution for extending the two-level synthesis procedure to multiple levels are suggested in the final chapter as an opportunity for further work.

Chapter 4

Function Block Realisation

Following a description of relevant terminologies, generalised multi-level synthesis models for strong and weak-indication realisation of combinational logic functions are presented. Due to the entire state space requirement for these models, an efficient two-level heuristic has been subsequently proposed and implemented, which is found to considerably alleviate the problem of input space explosion. A system configuration in support of this heuristic is also put forward that facilitates weak-indication solutions. In general, weakly indicating circuits are preferable due to the flexibility in relaxing the indication constraints that is absent in the case of strong-indication circuits. The heuristic has its roots in a novel set theory based procedure, used to derive minimum orthogonal sum-of-products expression from a minimum sum-of-products form. The heuristic is suitable for facilitating two-level ST implementation of combinational logic, but its extension to multiple levels involves considerable complexity and difficulty. This is because the opportunity for performing SI decomposition within a compressed input space cannot always be guaranteed whilst striving to preserve gate orphan freedom. Hence, multi-level synthesis of weak-indication circuits as an extension of the proposed heuristic warrants further research. However, preliminary insights towards achieving a palpable solution that does not compromise on the property of robustness are mentioned in the final chapter.

Before proceeding further, basic terminologies that will be often used in this chapter with regard to non-DI and DI datapaths are elucidated. Commonly used definitions of terminologies in synchronous domain are retained [87] [107] [123] and are correlated with the discussion on asynchronous logic for the sake of clarity.

- A *literal* is a symbol referring to a propositional variable (x) or its complement (x'). In case of DRE, the notion of a literal is used to refer to either the true-bit ($x1$) or the false-bit ($x0$) representation of a variable (x) respectively.

- A *cube* is defined as a logical product[12] (conjunction) of different literals, where a variable appears in only one of its symbolic notations. For example, *a'b*, *abc'd* are cubes or *product terms*. In the case of DRE, a cube specifies a logical conjunction of the true-bits or false-bits of different variables. *a*0*b*1 and *a*1*b*1*c*0*d*1 are the respective equivalent dual-rail encoded product terms of the single-rail cubes mentioned earlier.
- A *cover* is a set of irredundant product terms and the cardinality of a cover is the number of cubes comprising the cover.
- The product term is also referred to as the *prime-implicant*. A prime-implicant is said to be *essential* if it cannot be removed without affecting the cover, otherwise it is classified as *non-prime*. The *sum term* implies a logical sum (disjunction) of literals and is called the *prime-implicate*.

4.1 Terminologies and Definitions

Terminologies governing set theory based logic operations on DI datapaths are defined in this section, which are subsequently used to explain the process of SI logic decomposition in section 4.2. These are helpful in developing multi-level synthesis models for strong and weak-indication circuits. However, they could be of use to guide the process of SI decomposition in general. Unless otherwise stated, the following sections address dual-rail encoded datapaths.

4.1.1 Support Set and Dependency Set

The *support*[13] set of a cube C, S(C), enumerates the input variables that are a constituent of the cube. On the other hand, a cube's *dependency set* D(C) entails the listing of all the input literals that are a function of the cube for its evaluation to logic 'high'. For a cube C specified by *a*1*b*0*c*1*d*1, its S(C) and D(C) are:

[12] The Muller element typically serves as the logical conjunction operator, which is a non-relaxed gate. The AND gate is the equivalent relaxed conjunction operator (for the case of transitions).
[13] The term 'support' signifies a *set*. But the term 'set' has been additionally used as a suffix to maintain uniformity with the other set definitions to be introduced in this thesis. The support of a function (product term) is the set of variables appearing in the function (product term) [107] [123].

$S(C) = \{a,b,c,d\}$ **(4.1)**

$D(C) = \{a1,b0,c1,d1\}$ **(4.2)**

4.1.2 Cubes Support Intersection Set and Cubes Dependency Intersection Set

The cubes support (dependency) intersection set viz. CSI (CDI) set, marks the intersection of the support (dependency) set of two cubes, characterised by the variables (literals) that are common to/shared between the support (dependency) set of both the cubes. For example, with $D(C_1)$ and $D(C_2)$ specified by $\{a0,b0,c0,d0\}$ and $\{a0,b0,c0,d1\}$ respectively, their corresponding CSI and CDI sets are given by,

$CSI\ [S(C_1), S(C_2)] = \{a,b,c,d\}$ **(4.3)**

$CDI\ [D(C_1), D(C_2)] = \{a0,b0,c0\}$ **(4.4)**

4.1.3 Cubes Relativity Set

The cubes relativity (CR) set of a cube with respect to another cube identifies (isolates) the unique literals in the former compared to the latter. The CR set of cube C_1 relative to cube C_2 is obtained by computing the set-theoretic difference of the dependency set of the former cube and the CDI set of both the cubes. It basically amounts to finding the relative complement of the CDI set of both the cubes in the dependency set of cube C_1.

$CR\ [C_1, C_2] = D(C_1) \setminus CDI\ [D(C_1), D(C_2)]$ **(4.5)**

Similarly, CR set of C_2 with respect to C_1 is given by,

$CR\ [C_2, C_1] = D(C_2) \setminus CDI\ [D(C_1), D(C_2)]$ **(4.6)**

For example, with $D(C_1) = \{a1,b1,c0,d1,e0,g1\}$ and $D(C_2) = \{b1,c1,d1,e0,g0\}$,

$CR\ [C_1, C_2] = \{a1,c0,g1\}$, $CR\ [C_2, C_1] = \{c1,g0\}$ **(4.7)**

4.1.4 Variables Identification

Variables identification (VI) is an operation that is typically performed on the CR set yielding a set of variables as elements. It is equivalent to characterising the support of a CR set. With respect to (4.7),

$CR\_VI\ [C_1, C_2] = \{a,c,g\}$, $CR\_VI\ [C_2, C_1] = \{c,g\}$ **(4.8)**

4.2 Orthogonality and SI Decomposition

In this section, the relationships that govern orthogonal cubes[14], dual-rail sum-of-products and orthogonal sum-of-products forms, and SI decomposition of logic based on DRE by extraction of shared functionality and substitution are described.

4.2.1 Mutual Orthogonality Set and Degree

The essential relations to be satisfied in order that two cubes C_1 and C_2 may be said to be orthogonal to each other are given below.

$$|S(C_1)| \geq 1,\ |S(C_2)| \geq 1 \qquad \textbf{(4.9)}$$

$$|CR\ [C_1, C_2]| \geq 1,\ |CR\ [C_2, C_1]| \geq 1 \qquad \textbf{(4.10)}$$

The support set of any cube should consist of at least a single variable, which is specified by (4.9). According to (4.10), there should be at least one distinct element in $D(C_1)$ relative to $D(C_2)$ and vice-versa; otherwise there would not be a possibility for C_1 and C_2 to exhibit mutual orthogonality. But the satisfying of conditions (4.9) and (4.10) alone cannot guarantee the existence of an orthogonal relationship between C_1 and C_2, since CR $[C_1, C_2]$ and CR $[C_2, C_1]$ may contain literals associated with different variable indexes. For example, cubes C_i and C_j specified by dependency set elements $\{a1,b0,c1\}$ and $\{d1,e0\}$ respectively, satisfy the inequalities given in (4.9) and (4.10) but are not orthogonal. Hence, if and only if the inequality mentioned in (4.11) is additionally satisfied, then can the two cubes C_1 and C_2 be labelled as mutually orthogonal, for it shall then be guaranteed that two different instances (literals) of the same support variable would be present in either of the sets being intersected.

$$CR\_VI\ [C_1, C_2] \cap CR\_VI\ [C_2, C_1] \neq \emptyset \qquad \textbf{(4.11)}$$

The integer measure of the number of variables responsible for establishing the orthogonal relationship between two cubes is called the degree of mutual orthogonality (DMO). Given that (4.9) – (4.11) are true, the mutual orthogonality (MO) set that comprises orthogonal variables and the DMO between C_1 and C_2 are given by,

$$MO = CR\_VI\ [C_1, C_2] \cap CR\_VI\ [C_2, C_1] \qquad \textbf{(4.12)}$$

$$DMO = |CR\_VI\ [C_1, C_2] \cap CR\_VI\ [C_2, C_1]| \qquad \textbf{(4.13)}$$

[14] Two cubes are said to be orthogonal to each other if their logical product results in a '0'. For example, $x0y0$ and $x1y1$ are said to be mutually orthogonal.

4.2.2 Sum-of-Products and Orthogonal Sum-of-Products Forms

A sum-of-products (SOP) expression consists of a disjunction of standard product terms, each of which involves a conjunction of literals. If the number of terms in a SOP form is the least possible, then the SOP is referred to as minimum SOP (MSOP). An orthogonal sum-of-products (OSOP) form [76] consists of product terms that are all orthogonal to each other, i.e. the cubes do not overlap. Every cube is orthogonal to every other cube in an OSOP expression and therefore it would inherently satisfy the monotonic cover constraint. An OSOP form with the least number of product terms can be called minimum OSOP (MOSOP) form. A MSOP form or MOSOP form comprising an identical single cube are said to be equivalent. The MSOP and MOSOP expressions for the carry output of a dual-rail full adder are given by (4.14), (4.15) and (4.16), (4.17) respectively, where (*a*1,*a*0), (*b*1,*b*0) and (*cin*1,*cin*0) are the dual-rail inputs and (*Cout*1,*Cout*0) is the dual-rail output.

$$Cout1_{\mathrm{MSOP}} = a1b1 + b1cin1 + a1cin1 \qquad \textbf{(4.14)}$$

$$Cout0_{\mathrm{MSOP}} = a0b0 + b0cin0 + a0cin0 \qquad \textbf{(4.15)}$$

$$Cout1_{\mathrm{MOSOP}} = a1b1 + a0b1cin1 + a1b0cin1 \qquad \textbf{(4.16)}$$

$$Cout0_{\mathrm{MOSOP}} = a0b0 + a1b0cin0 + a0b1cin0 \qquad \textbf{(4.17)}$$

4.2.3 Criteria for SI Decomposition

The necessary criteria for performing SI logic decomposition by way of extracting shared logic between two mutually orthogonal cubes C_1 and C_2 are listed below. These are also useful to achieve decomposition up to a finer granularity of elements specified in the base-function set (i.e. the cell library).

$$|S(C_1)| > 1,\ |S(C_2)| > 1 \qquad \textbf{(4.18)}$$

$$S(C_1) = S(C_2) \qquad \textbf{(4.19)}$$

$$|CR\ [C_1, C_2]| = |CR\ [C_2, C_1]| = 1 \qquad \textbf{(4.20)}$$

The first constraint conveys that there should be at least two elements in the support set of both the cubes, which is obviously mandatory for decomposition, as a cube with a singleton support set reduces to a simple wire.

The second relation ensures that the variables of both the cubes are identical, which is an essential criterion for considering extraction of logic shared between them. Cubes with disjoint supports cannot feature any commonality. Assuming that a function consists of a number of cubes, where the support set elements corresponding to all the cubes are distinct, the function could then only be classified as read-once[15] as it would have been implicitly expressed in its MSOP form [129]. In terms of dependency sets, (4.19) implies that $|D(C_1)| = |D(C_2)|$, i.e. C_1 and C_2 are said to be *equipollent* (equal in size and strength).

The third condition is vital, as its fulfilment would point to an opportunity for performing SI decomposition of equipollent cubes, which satisfy relations (4.18) and (4.19). It essentially means that there is only one unique literal in C_1 relative to C_2 and vice-versa. Given the above conditions are upheld, it becomes obvious that the equality $|CDI\ [D(C_1), D(C_2)]| = |D(C_1)|-1 = |D(C_2)|-1$ would hold good.

If two cubes C_a and C_b are not orthogonal to each other and if they satisfy (4.18), with $|S(C_a)| > |S(C_b)|$, then a possibility for SI decomposition could exist even though C_a and C_b are not equipollent, provided $D(C_b) \subseteq D(C_a)$. To explain this, let us assume that C_a and C_b are given by $a1b0c0d0e1$ and $b0d0$ respectively. Provided the activation of C_b would be certainly acknowledged by the next level logic, C_a can be expressed as the conjunction of cubes $a1c0e1$ and C_b. Indeed, C_b should belong to the cover of a function output different from that of the cover comprising C_a. Both C_a and C_b cannot be present in the same function cover as C_b would be said to contain C_a, where C_a becomes the covered cube and C_b is the covering cube that absorbs C_a. Also, C_a and C_b cannot be present in different rails (of the dual-rail) of the same encoded function block output as then the system could enter into an illegal state (both 'set' and 'reset' functions could be asserted 'high' simultaneously). Nevertheless, this sort of SI decomposition does not normally occur in case of indicating circuit synthesis models (especially, in our multi-level synthesis models), which consider the entire input space, and is mentioned here only as a supplementary information targeting scenarios that do not feature such an assumption.

4.2.4 Primary and Secondary SI Cubes

The need for SI decomposition arises whenever larger cubes are present in a function which cannot be implemented directly due to fan-in restrictions of the cell library and therefore have to be expressed in terms of smaller physically realisable cubes. With the previously mentioned

[15] A function is said to be *read-once*, if each variable appears only once in its factored form [129]. For example, the function $F = xy+yz = x(y+z)$ is read-once.

conditions satisfied between two mutually orthogonal cubes, say C_1 and C_2, a common cube can be extracted from among them, which we shall refer to as the speed-independent cube (SIC). If the SIC that can be extracted is labelled as C_3, then the following properties hold good: $D(C_3) \subseteq D(C_1)$ and $D(C_3) \subseteq D(C_2)$. Similarly, $S(C_3) \subseteq S(C_1)$ and $S(C_3) \subseteq S(C_2)$. The size of cube C_3 would then be governed by (4.21), while its literals are specified by (4.22).

$$|S(C_3)| = |S(C_1)|-1 = |S(C_2)|-1 \qquad \textbf{(4.21)}$$

$$D(C_3) = \text{CDI}\ [D(C_1), D(C_2)] \qquad \textbf{(4.22)}$$

After the process of SI decomposition, the variables and literals of the parent cubes C_1 and C_2 can be enumerated using (4.23) and (4.24), (4.25) respectively.

$$S(C_1) = S(C_2) = S(C_3) \cup \text{CR\_VI}\ [C_1, C_2] = S(C_3) \cup \text{CR\_VI}\ [C_2, C_1] \qquad \textbf{(4.23)}$$

$$D(C_1) = D(C_3) \cup \text{CR}\ [C_1, C_2] \qquad \textbf{(4.24)}$$

$$D(C_2) = D(C_3) \cup \text{CR}\ [C_2, C_1] \qquad \textbf{(4.25)}$$

Cubes C_1 and C_2 can be called primary SICs (PSICs) if their support sets are found to be a function of all the primary input variables. Given this, C_3 can be referred to as the secondary SIC (SSIC) and it is usually substituted into PSICs as an intermediate node. The PSIC is basically a canonical product term comprising all the primary input variables of the function block, while the SSIC is a standard product term formed from a subset of the support set variables of the function. In general, a function comprising a single realisable cube is said to contain a SIC. From the preceding discussions, it may be intuitively observed that whenever two cubes become candidates for SI decomposition, they are necessarily orthogonal but not vice-versa. For example, C_m and C_n represented by their dependency sets $D(C_m) = \{a0,b1,c1,d0\}$ and $D(C_n) = \{a0,b1,c0,d1\}$ are orthogonal, yet no common logic can be extracted from among the two cubes in a SI fashion as (4.20) is not satisfied between them. This observation holds good for ST datapaths adopting any DI data encoding scheme.

4.2.5 Datapaths Employing 1-of-*n* Codes

The concepts discussed above serve as a basis for the following discussion on ST datapaths incorporating arbitrary one-hot codes. We shall first discuss this on the basis of the 1-of-4 code to provide a specific illustration. As mentioned in Chapter 2, two single-rail inputs can be represented using a 1-of-4 code symbolically. To avoid confusion that could result from similar symbolic variable assignments, a condition is imposed whereby the representation of each unique pair of single-rail inputs by a 1-of-4 code equivalent should be distinct in terms of the symbol variables used for a corresponding mapping. Notwithstanding, the set definitions mentioned above would not be adequate to address ST datapaths that employ arbitrary combinations of generic 1-of-*n* codes. For example, four single-rail inputs (*m*,*n*,*o*,*p*) are first converted into two-pairs and are mapped as highlighted in (4.26) and (4.27), which constitutes a valid representation. On the contrary, the mapping (*m*,*n*) ↔ (*q*0,*q*1,*q*2,*q*3) and (*o*,*p*) ↔ (*q*4,*q*5,*q*6,*q*7) is classified as non-permissible because the VI operation would yield the same element. Nevertheless, to permit such similar symbolic variable assignments, a different mechanism is adopted and is stated in the thesis appendix.

$$(m,n) \leftrightarrow (q0,q1,q2,q3) \qquad \textbf{(4.26)}$$

$$(o,p) \leftrightarrow (r0,r1,r2,r3) \qquad \textbf{(4.27)}$$

Let us assume that a function F is dependent on six input variables (*a*,*b*,*c*,*d*,*e*,*f*), and expressed by the disjunction of two cubes X and Y, which are specified by *a'bcd'e'f* and *a'b'c'd'ef* in single-rail format. With the pairs of input variables (*a*,*b*), (*c*,*d*) and (*e*,*f*) mapped to symbolic notations (*i*0,*i*1,*i*2,*i*3), (*j*0,*j*1,*j*2,*j*3) and (*k*0,*k*1,*k*2,*k*3) respectively, and assuming a similar encoding assignment as shown in Table 2.1, we have the dependency sets of cubes X and Y described as thus:

$$\mathrm{D(X)} = \{i2,j1,k2\} \qquad \textbf{(4.28)}$$

$$\mathrm{D(Y)} = \{i3,j3,k0\} \qquad \textbf{(4.29)}$$

Referring to relations (4.9) – (4.11), it can be seen that cubes X and Y jointly satisfy them and hence they are categorised as orthogonal cubes. But, since they do not mutually satisfy (4.20), SI decomposition does not become possible and they may just remain as SICs.

Let us now consider a datapath of odd width and let (*a*,*b*,*c*,*d*,*e*,*f*,*g*,*h*,*i*) be the single-rail inputs. Let us have the following permissible mappings utilising the 1-of-2, 1-of-4 and 1-of-8 DI codes for a ST datapath.

$$a \leftrightarrow (a0,a1) \qquad \textbf{(4.30)}$$

$$(b,c,d) \leftrightarrow (m0,m1,m2,m3,m4,m5,m6,m7,m8) \qquad \textbf{(4.31)}$$

$$(e,f) \leftrightarrow (n0,n1,n2,n3) \qquad \textbf{(4.32)}$$

$$g \leftrightarrow (g0,g1) \qquad \textbf{(4.33)}$$

$$(h,i) \leftrightarrow (p0,p1,p2,p3) \qquad \textbf{(4.34)}$$

The mapped representation $b'c'd' \leftrightarrow m8$ and $bcd \leftrightarrow m0$ is assumed. Let two cubes Z_1 and Z_2 be specified by $a'b'c'd'e'fgh'i'$ and $a'bcde'fgh'i'$ respectively in their single-rail format. Referring to Table 2.1 again for the 1-of-4 coded value assignments corresponding to a pair of single-rail inputs, the dependency sets of the encoded cubes are given as,

$$D(Z_1) = \{a0, m8, n2, g1, p3\} \quad \textbf{(4.35)}$$

$$D(Z_2) = \{a0, m0, n2, g1, p3\} \quad \textbf{(4.36)}$$

It can be seen that after encoding, cubes Z_1 and Z_2 satisfy (4.9) – (4.11). Also, (4.20) is satisfied between them, as their MO set is singleton. Hence, it can be inferred that cubes Z_1 and Z_2 are not only orthogonal to each other but they could also be candidates for SI decomposition, thereby extraction of logic shared between them becomes feasible.

4.3 General Synthesis Models

The general multi-level synthesis models for strong and weak-indication circuits are discussed in this section. Though they suffer from the input space explosion phenomenon, they are helpful to practically implement function blocks comprising many concurrent inputs.

4.3.1 Architecture for Strongly Indicating Circuits

The general system configuration for realisation of combinational logic as strong-indication circuits is shown below in figure 4.1, which is typical of circuits employing DI data encoding and a 4-phase handshake protocol and it resembles the system architecture portrayed by figure 3.18. The function block is strongly indicating and would usually encompass a tree type structure internally, with all the logical conjunctions achieved through Muller elements. But unlike the DIMS approach, the logic could be conveniently spread over multiple levels in a straightforward manner whilst preserving speed-independency. The disadvantage is that the function block size grows exponentially with the number of primary inputs. Nevertheless, this problem is faced by all existing strong-indication circuit synthesis methods.

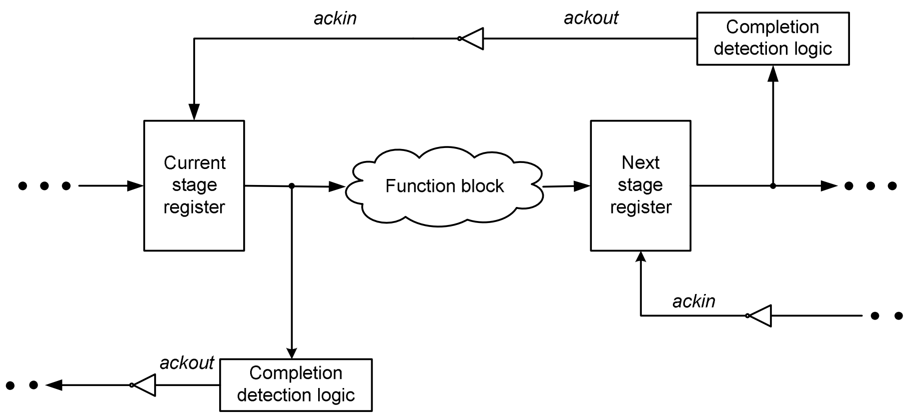


Figure 4.1: Typical ST system configuration

The function block would usually consist of primary and secondary SICs. An illustration of the presence of PSICs and a SSIC is shown in figure 4.2, for a sample case of five inputs highlighting a logic tree. It could be seen that the PSICs are mutually orthogonal as they adhere to the conditions stated in section 4.2.1, and SI decomposition is feasible as the constraints implied by (4.18) – (4.20) are satisfied.

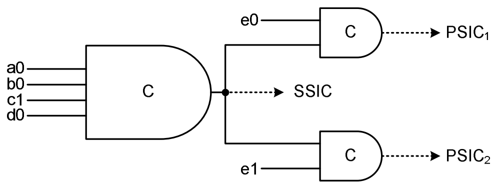


Figure 4.2: Depicting PSICs and SSIC

If the number of primary function block inputs is specified by *n* single-rail inputs (i.e. 2*n* dual-rail inputs), and for $n > 4$, the logic depth required to realise the PSICs can be estimated as (n-3) for an optimal utilisation of the base function set comprising C-gates with a maximum fan-in of 4. The number of PSICs is governed by 2^n. In general, the first logic level can accommodate inputs according to the maximum granularity specified in the base-function set (maximum fan-in of C-gates), given by *l*. Hence, the number of C-elements in the k^{th} logic level can be estimated as $2^{(k+l-1)}$, where $k = 1,2,\ldots,(n-l+1)$. Here, the parameter (n-l+1) denotes the maximum number of logic levels corresponding to the robust implementation of a function block. SSICs are realised till the $(n\text{-}l)^{\text{th}}$ logic level and the PSICs, which are all distinct are derived in the subsequent logic level, which can then be logically summed up to implement the desired functionality. An illustration of strongly indicating implementation of a combinational benchmark, *check*, which comprises 4 single-rail inputs (*a,b,c,d*) and produces a single output (F) is shown in figure 4.3, assuming the base-function set to be composed of only CE3s and CE2s. SSICs correspond to the outputs of the first-level C-gates, while PSICs are obtained as outputs of the second-level C-gates.

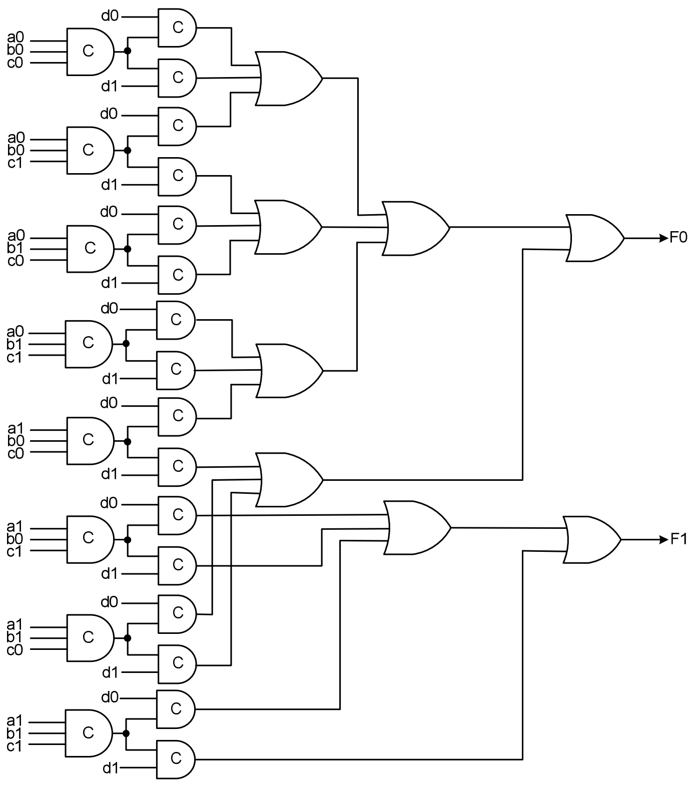


Figure 4.3: SI decomposed strongly indicating realisation of *check* function block

4.3.2 Modifications to Suit Weak-Indication Timing Model

The system configuration featuring slight modifications to suit the weak-indication timing regime is shown in figure 4.4. In contrast to a strong-indication function block, the weak-indication block relies upon relaxed gates for the first logic level to facilitate conjunctions, thereby possibly reducing some area cost. With respect to figure 4.3, equivalent relaxed gates would now replace the non-relaxed gates in the first logic level. However, this would lead to ambiguity in determining the proper reset of primary inputs fed to the first level of logic as it relaxes the strong-indication constraints. So to resolve this, an extra synchronisation block is introduced in comparison with the architecture shown in figure 4.1. The *synchroniser* is meant to synchronise any DI encoded output pair of the function block with a portion of the current stage completion detection (CD) logic circuitry (here, CD for the first-level inputs of the function block viz. *a*1,*a*0,*b*1,*b*0,*c*1,*c*0) using C-gates. This ensures that Seitz's weak-indication timing constraints are satisfied by way of guaranteeing that all but one of the encoded outputs (valid/spacer) is not produced until after all the encoded inputs (valid/spacer) have arrived.

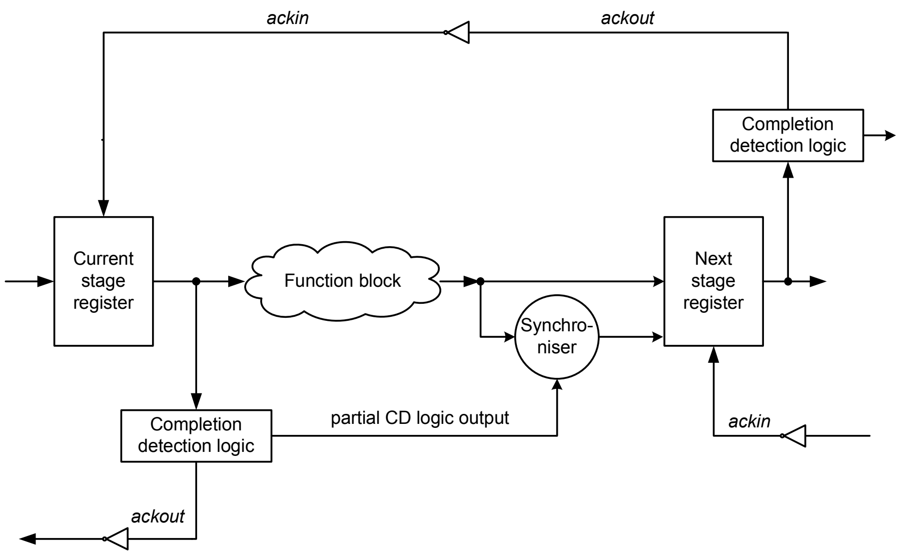


Figure 4.4: System architecture for weak-indication circuits

The outputs of the functional part (function block) and the synchroniser together define the weak-indication property for the entire ST combinational logic implementation. If all the function block outputs were passed onto the next stage through the synchroniser module, then the system would be classified as strongly indicating. However, this would necessitate buffering of the partial CD logic output, as it has to be synchronised with all the logic block outputs. Hence, within the ambit of the system configuration shown above, it may be concluded that weak-indication circuits might consume relatively less area, suffer less delay and demand less power than their strong-indication counterparts and so the former may be preferable compared to the latter.

4.4 Set Theory Based Heuristic for Compact Realisation of Function Blocks

Entire state space consideration is the main drawback with the earlier generalised approach although it permits SI decomposition to be conveniently extended over multiple levels. As a result, function blocks featuring several concurrent inputs cannot be implemented efficiently following this approach. To pave the way for compact realisation of function blocks containing several inputs, an aggressive strategy is proposed in this section. Significantly, it is found to contain the exponentially expanding state space, thereby being useful for addressing larger function block specifications. It also forms a very good starting point for multi-level synthesis of weak-indication circuits.

4.4.1 Deriving MOSOP Expression from MSOP Expression

In section 4.2, guidelines to identify mutually orthogonal cubes and the conditions to be satisfied for performing SI decomposition of such cubes were discussed in detail. In this section, a set theory based heuristic to derive minimum disjoint SOP (MDSOP) expression from a MSOP expression is presented. Obtaining MDSOP solutions for the true and false outputs of a logic function would amount to finding MOSOP form for the function block on the whole, which has been described using DRE. In other words, though the DSOP and OSOP expressions comprise non-overlapping product terms, the OSOP form is distinguished in that it features encoded outputs which are dependent on encoded inputs. Hence, it may be concluded that the efficiency of the MDSOP heuristic could have a direct impact on the effectiveness of the resulting MOSOP solutions. Before proceeding further with respect to deducing MDSOP solutions, some background information is provided.

A Boolean function, *f*, is a mapping of type *f*: $\{0,1\}^n \rightarrow \{0,1,d\}$, where '*d*' denotes a *don't care* condition. If '*d*' does not exist, then the function *f* is said to be completely specified or two-valued, otherwise it is called incompletely specified or ternary. Each of the 2^n nodes in the Boolean space corresponds to a canonical product term (minterm). The ON-set, OFF-set and DC-sets of *f* correspond to those minterms that are mapped to 1, 0 and *d* respectively. Conventional two-level logic minimisers typically consider the ON-sets and DC-sets of all the function outputs simultaneously to reduce the number of essential cubes, necessary to realise the desired multi-output logic functionality by taking into account the essential prime implicants that could be shared between different outputs. Of course, a minimum or reduced SOP form of a multi-output logic function specification can be obtained using a standard two-level logic minimiser: Espresso [130]. In this context, it may be observed that the ON-sets and DC-sets of a multiple-input, multiple-output (MIMO) logic function can be considered simultaneously to obtain the reduced SOP expressions corresponding to the true-rail of the function block outputs (i.e. true outputs), while the OFF-sets of the MIMO function can be considered separately to derive the minimised SOP expressions corresponding to the false rail of the function block outputs (i.e. false outputs). Though this approach appears to be attractive as it could significantly reduce the input space requirement, it may very likely neglect the two important constraints that need to be satisfied for robust asynchronous logic designs: *cover constraint* and *indication*. It was mentioned in section 4.2.2, that if all the cubes of a reduced SOP expression are non-overlapping, then the expression would implicitly satisfy the constraint that only one product term of a function output should become activated for the case of transitions. Eventually, either the true rail or false rail of each function output would alone assume a logic ‘high’ state during the set phase. Towards this end, Espresso can be used to generate DSOP expressions for both the true and false rails of a MIMO function block so that the cover constraint can be satisfied.

A Boolean equation is said to be in DSOP form, if it is described by a logical sum of product terms that are all disjoint [131], i.e. no two product terms cover a common minterm in their expanded form. A DSOP form with the least number of product terms is known as minimum DSOP (MDSOP) form. While SOP minimisation can be likened to a set covering problem, DSOP minimisation can be likened to the problem of finding a minimum exact disjoint cover, which is NP-hard [131]. For example, the number of essential prime implicants comprising the SOP expression of an Achilles’ heel function [131] is given by $O\left(n/2\right)$, while the number of essential prime cubes constituting its DSOP expression is specified by $O\left(2^{n/2}-1\right)$, where ‘n’ represents the number of distinct primary single-rail inputs. DSOPs have been traditionally used in several applications in CAD areas, for example, calculation of spectra of Boolean functions [132] – [134] or as a starting point for the minimisation of

Exclusive-OR SOP logic [135] [136], which in turn forms the backbone of synthesis schemes for reversible logic circuits [137] [138] that have applications related to the field of quantum computing. It has been found that DSOP solutions generated by Espresso are generally far from the optimum, especially in case of functions with several inputs because it considers group minimisation of all the function outputs. An alternative approach would be to consider deriving MDSOP solutions for the MIMO function outputs on an individual basis from their MSOP forms using heuristics or resort to constructing ROBDDs for the entire MIMO function as they inherently incorporate the mutual-exclusiveness property, leading to the DSOP forms. Nevertheless, the latter approach may suffer from memory space requirements for higher order functionality and therefore the former method might be preferable. In this context, a number of methods have been proposed by researchers [139] – [144], predominantly considering utilisation of ROBDDs or adopting some evolutionary programming techniques. A majority of these methods were found to yield a far better solution for many case studies in comparison with the solution obtained using Espresso.

Some relevant terminologies are defined in the backdrop of the conventional bundled-data encoding (single-rail) protocol for the sake of clarity, before propounding the set theory based method to derive MDSOP expression from a MSOP form.

4.4.1.1 Support Set and Dependency Set

The *support set* of a cube C, S(C), entails the enumeration of all the variables that are a function of the cube, while a cube's *dependency set* D(C) entails the enumeration of all its literals in their actual form (complemented or uncomplemented) for its evaluation to a logic '1'. For a cube C specified by *ab'c'd*, its S(C) and D(C) are:

$$\mathrm{S(C)} = \{a,b,c,d\} \qquad \textbf{(4.37)}$$

$$\mathrm{D(C)} = \{a,b',c',d\} \qquad \textbf{(4.38)}$$

4.4.1.2 CSI Set, CDI Set and Polarity Eliminated CDI Set

The intersection of the support set of two cubes (dependency set of two cubes) is characterised by the variables (literals) that are common to the support set (dependency set) of both the cubes.

The polarity eliminated CDI (CDI_PE) set consists of all the literals of the CDI set represented in their normal (uncomplemented) form. The elements of the CDI_PE set can be obtained by performing the VI operation on the CDI set.

For example, with D(C_1) and D(C_2) specified by $\{a',b,c,d\}$ and $\{a',b',c,f\}$ respectively, the CSI, CDI and CDI_PE sets are given by,

$$\text{CSI}\ [S(C_1), S(C_2)] = \{a,b,c\} \qquad \textbf{(4.39)}$$

$$\text{CDI}\ [D(C_1), D(C_2)] = \{a',c\} \qquad \textbf{(4.40)}$$

$$\text{CDI\_PE}\ [D(C_1), D(C_2)] = \{a,c\} \qquad \textbf{(4.41)}$$

4.4.1.3 Describing Mutually Disjoint Cubes

When two Boolean cubes C_1 and C_2 are said to be mutually disjoint, the following inequalities hold. Showing that their negations are false proves the inequalities.

$$|\text{CSI}\ [S(C_1), S(C_2)]| \geq 1 \qquad \textbf{(4.42)}$$

$$|\text{CDI}\ [D(C_1), D(C_2)]| \geq 0 \qquad \textbf{(4.43)}$$

Let us assume that the first inequality $|\text{CSI}\ [S(C_1), S(C_2)]| \geq 1$ is not valid, which would only imply that $|\text{CSI}\ [S(C_1), S(C_2)]| = 0$ and therefore $\text{CSI}\ [S(C_1), S(C_2)] = Ø$. The second inequality would be automatically satisfied as a fall-out as we negate the current assumption, since $|\text{CDI}\ [D(C_1), D(C_2)]| < 0$ is an invalid starting argument because any cube would consist of at least a single literal. Given the present assumption ($\text{CSI}\ [S(C_1), S(C_2)] = Ø$), it becomes obvious that no variable would be shared between C_1 and C_2 and the function can only be then classified as read-once implying that it is basically expressed in its MSOP form [129]. In such a situation, the product terms of the function would definitely overlap [129] and so there is no mutual-exclusiveness exhibited between the cubes. When the function is expressed in its canonical form, the common cube(s) would eventually be manifested. The above discussion assumed C_1 and C_2 to be non-identical cubes. Provided C_1 and C_2 are identical, a stronger constraint needs to be enforced, which is given in the next section.

4.4.1.4 Disjoint Set and Degree of Disjointness

Two cubes are said to be disjoint if their conjunction (achieved by the Boolean AND operator) yields a *null*. The disjoint (DJ) set identifies the input variables that are responsible for making two Boolean cubes (say C_1 and C_2) mutually disjoint. It is given by the set-theoretic difference of CSI and CDI_PE, corresponding to cubes C_1 and C_2. In other words, it amounts to finding the relative complement of CDI_PE in CSI.

$$DJ\ [C_1, C_2] = CSI\ [S(C_1), S(C_2)] \setminus CDI\_PE\ [D(C_1), D(C_2)] \qquad \textbf{(4.44)}$$

The degree of disjointness (DDJ) gives the integer measure of the number of primary inputs, which are responsible for making two cubes C_1 and C_2 mutually disjoint. It is given by (4.45). Any two non-overlapping cubes would satisfy the inequality condition highlighted by (4.46).

$$DDJ = |DJ\ [C_1, C_2]| \qquad \textbf{(4.45)}$$

$$DDJ \geq 1 \qquad \textbf{(4.46)}$$

If between any two dissimilar cubes, say C_1 and C_2, the above inequality constraint would not get satisfied, i.e. DDJ = 0, then the possible reason for this scenario is depicted by (4.47), which is a direct consequence of the cubes featuring a disjoint support; for example, with cubes C_1 and C_2 represented by *abcd* and *efg* respectively. This would then revert back to the situation described in section 4.4.1.3, wherein the cubes would be found to overlap when expanded, thereby no disjointness shall exist between them. Consequently, the condition described by (4.48) would be satisfied. In this case, DDJ is zero and so there is no variable that could make the two cubes disjoint as they would be governed by the properties mentioned in (4.49) and (4.50). If C_1 and C_2 are similar, then $DJ\ [C_1, C_2] = DJ\ [C_2, C_1] = Ø$.

$$CSI\ [S(C_1), S(C_2)] = CDI\_PE\ [D(C_1), D(C_2)] = Ø \qquad \textbf{(4.47)}$$

$$DJ\ [C_1, C_2] = Ø \qquad \textbf{(4.48)}$$

$$|CSI\ [S(C_1), S(C_2)]| = |CDI\ [D(C_1), D(C_2)]| \qquad \textbf{(4.49)}$$

$$CSI\ [S(C_1), S(C_2)] = CDI\_PE\ [D(C_1), D(C_2)] \qquad \textbf{(4.50)}$$

4.4.1.5 Deducing MDSOP Form

The heuristic proposed to deduce MDSOP expression from a MSOP expression is explained from a high-level perspective by means of the following steps.

- *Step 1*: Obtain the MSOP form of a logic function.
- *Step 2*: Compare each cube with every other cube in the MSOP form to check whether they are mutually disjoint. If and only if each cube exhibits mutual disjointness with every other cube in the MSOP form, then go to Step 11, else proceed with Step 3.
- *Step 3*: Enumerate all the overlapping pairs of cubes that have a non-disjoint support. If only pairs of cubes with disjoint support exist, go to Step 8, else proceed with Step 4.

- *Step 4*: From among the overlapping pairs of cubes that feature a non-disjoint support, choose that pair of cubes, which comprises the highest degree of logic sharing among its constituents. If many such pairs of cubes exist, which exhibit a similar highest degree of commonality then an arbitrary choice is resorted to.
- *Step 5*: Use the *distributive axiom* [$ab+ac = a(b+c)$] to extract the kernel. Apply the converse of the *absorption axiom* of Boolean algebra ($a+a'b = a+b$) to transform the kernel comprising overlapping cubes with disjoint support into non-overlapping cubes with a non-disjoint support. Apply the distributive property of Boolean algebra [$a(b+c) = ab+ac$] to enumerate the product terms. Should the kernel comprise cubes whose dimensions are greater than unity, apply the *identity axiom* ($a+a' = 1$) to the least sized cube. Then use the distributive law to enumerate the products.
- *Step 6*: Check whether any cube contains any other cube in the function; if so, the covering cube absorbs the covered cube. Also, check whether any cube is duplicated in the logic function. If so, the redundancy is eliminated by applying the *idempotency axiom* [$a+a = a$].
- *Step 7:* Go to Step 2.
- *Step 8*: Consider any two cubes with a disjoint support, which also have the least support set cardinalities. If many choices result, then a random selection is made. Between such a pair of cubes, the identity law of Boolean algebra is applied to any of the pair of cubes considered, which would result in a cube expansion by making use of the distributive axiom.
- *Step 9*: If any cube is found to cover any other existing cube in the function, the covered cube is discarded and the covering cube is alone retained. Logic duplication is eliminated using the idempotent law.
- *Step 10:* Return to Step 2.
- *Step 11*: Terminate the routine as the desired MDSOP solution has been obtained.

The proposed procedure is followed for all the MSOP expression(s) of the output(s) of a logic function, independently and in parallel. In general, a function would be specified by several inputs and outputs. The MSOP solution for a function is obtained through multi-output minimisation, by using a standard logic minimiser: Espresso [130]. The *logical correctness* of the MDSOP solution is guaranteed by the Boolean axioms used, which are well-established and proven properties. The *functional correctness* of the MDSOP solution is ensured by comparison of each cube with every other cube forming the cover of each function output. However, such a comparison is performed as part of the processing. The combinational equivalence of a MSOP form and its corresponding MDSOP form is confirmed through the 'Dverify' option of Espresso. The final cost of the MDSOP solution derived for a MSOP specification is represented by the count of all the unique input cubes, some/all of which may eventually be shared between the various function outputs. Depending upon the initial logic description, several iterations of some/all of the above steps may be required in order to arrive at the final solution. In general, the opportunity for making use of the absorption axiom is first exploited with respect to a logic expression before considering the usage of the identity axiom, as the former would only increase the dimension of a cube whereas the latter would increase the number of cubes. A possible peephole optimisation might involve determining a good choice of variable order whilst applying the axioms to the enumerated sets; here, variable ordering is similar to the concept of variable sifting used to facilitate a compact BDD construction [145] [146].

The following serve as guidelines for the proposed method to derive MDSOP form from a MSOP form, articulating some important underpinning set-theoretic operations.

- As mentioned in Step 2, to ascertain whether all the cubes corresponding to each unique function output are non-overlapping, the following set operations are to be satisfied: DJ $[C_x, C_y] \neq Ø$ and hence DDJ ≥ 1 between any pair of arbitrary cubes C_x and C_y comprising a function output, where $(x, y) \in i$ and $x \neq y$, given $F_{MDSOP} = \sum_{i=1}^{k} C_i$, where the MDSOP form of a function output F is said to contain k cubes.
- To enumerate the overlapping cubes as mentioned in Step 3, between say C_m and C_n in a logic expression, the condition CSI $[S(C_m), S(C_n)] \neq Ø$ should be upheld.
- To single out a pair of cubes that feature a non-disjoint support but differ by only a single literal as described in Step 4, the following equality relationship has to be satisfied: $|\text{CDI } [D(C_m), D(C_n)]| = |D(C_m)| - 1 = |D(C_n)| - 1$. Also, amongst any three overlapping cubes, say C_m, C_n and C_o of a logic expression, if the following hold good, viz. CDI $[D(C_m), D(C_n)] \neq Ø$, CDI $[D(C_n), D(C_o)] \neq Ø$ and CDI $[D(C_m), D(C_o)] \neq Ø$, the choice of the pair of cubes made during the first iteration would depend upon the

highest value corresponding to |CDI [D(C_m), D(C_n)]|, |CDI [D(C_n), D(C_o)]| and |CDI [D(C_m), D(C_o)]|. In case of a tie between the cardinalities of CDI sets, an arbitrary choice is made. The sorting procedure considers all the distinct cubes of the expression. In general, for the worst-case, a computational complexity of $O\left(\frac{k(k-1)}{2}\right)$ would be involved, where 'k' refers to the number of non-redundant cubes comprising a function output.

- When a cube C_m is said to cover another cube C_n in an output expression as mentioned in Step 6, then the following would be true: D(C_m) ⊆ D(C_n) and |D(C_n)| > |D(C_m)|, in which case C_m is alone retained and C_n is discarded. Additionally, if there exists a cube C_p in the expression, and C_q is newly introduced due to a set operation, if D(C_p) = D(C_q), then the cubes C_p and C_q are said to be identical. To avoid logic duplication, either C_p or C_q is discarded and the other retained.
- Between a pair of cubes, say C_m and C_n, featuring a disjoint support as specified in Step 8, the condition CSI [S(C_m), S(C_n)] = Ø is satisfied.

4.4.1.6 MDSOP Cost of Combinational Benchmarks

The MDSOP heuristic mentioned above has been implemented in Java and has been used to generate results for some combinational benchmarks described in PLA format [147], to comment on its potential in comparison with other DSOP heuristics. However, it should be noted that the MDSOP heuristic, as such, corresponds to purely *synchronous logic*. Minimum SOP and DSOP forms of a MCNC benchmark [147], *newtag*, obtained using Espresso are represented by means of the cube-variable support matrices of figures 4.5 and 4.6 respectively for illustration purpose. The benchmark has a single output and its support set is composed of 8 elements. The cube-variable support matrix is an O($m \times n$) matrix, where 'm' specifies the number of irredundant cubes of the function (rows of the matrix) and 'n' refers to the number of unique input variables of the function (columns of the matrix). A '1' entry at the intersection of a particular row and column index (a_{mn}) implies the existence of a variable in its normal form, while '0' and '-' entries signify the inverted and don't care states of the variable respectively. The conjunction of all the variables in a row, appearing in either their normal or complementary forms describes the cube corresponding to that row of the matrix.

The logic function is expressed as F = $\sum_{i=1}^{m} C_i$, i.e. the summation of *m* non-redundant cubes that may have a maximum dimension, *n*. In the cube-variable support matrix of a DSOP form, $a_{pq} \neq a_{rq}$, for any (*p*, *r*) of *m* with respect to at least a column *q* of the matrix, where $p \neq r$ and *q* signifying a column index. Figure 4.7 depicts the cube-variable support matrix of the DSOP form of *newtag* corresponding to the MDSOP heuristic. It can be inferred from this example that the cost (number of essential cubes) of the proposed heuristic is similar to the cost of the SOP solution of Espresso, and thus the former has effected reduction in cost by 43% when compared with the DSOP solution of Espresso.

$$\begin{array}{r|cccccccc|}
 & a & b & c & d & e & f & g & h \\
\hline
df'h' & - & - & - & 1 & - & 0 & - & 0 \\
de'h' & - & - & - & 1 & 0 & - & - & 0 \\
df'g' & - & - & - & 1 & - & 0 & 0 & - \\
de'g' & - & - & - & 1 & 0 & - & 0 & - \\
de'f' & - & - & - & 1 & 0 & 0 & - & - \\
c & - & - & 1 & - & - & - & - & - \\
b' & - & 0 & - & - & - & - & - & - \\
a & 1 & - & - & - & - & - & - & -
\end{array}$$

Figure 4.5: Cube-variable support matrix of SOP form of *newtag*, based on Espresso

$$\begin{array}{cccccccc}
a & b & c & d & e & f & g & h \\
\hline
- & 0 & - & - & - & - & - & - \\
- & 1 & - & 1 & - & 0 & 0 & - \\
- & 1 & - & 1 & 0 & 1 & 0 & - \\
- & 1 & - & 1 & - & 0 & 1 & 0 \\
- & 1 & - & 1 & 0 & 1 & 1 & 0 \\
- & 1 & - & 1 & 0 & 0 & 1 & 1 \\
- & 1 & 1 & 0 & - & - & - & - \\
- & 1 & 1 & 1 & 1 & 1 & - & - \\
- & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
- & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & - & - & - & - \\
1 & 1 & 0 & 1 & 1 & 1 & - & - \\
1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1
\end{array}$$

Figure 4.6: Cube-variable support matrix of DSOP form of *newtag*, based on Espresso

$$\begin{array}{cccccccc}
a & b & c & d & e & f & g & h \\
\left[\begin{array}{cccccccc}
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & - \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & - \\
0 & 1 & 0 & 1 & 0 & 0 & - & - \\
0 & 1 & 1 & - & - & - & - & - \\
0 & 0 & - & - & - & - & - & - \\
1 & - & - & - & - & - & - & -
\end{array}\right]
\end{array}$$

Figure 4.7: Cube-variable support matrix of DSOP form of *newtag*, based on the MDSOP heuristic

A number of MCNC/LGSynth93 combinational benchmark problems [147] [148] were considered to comment on the potential of the proposed heuristic. Table 4.1 shows the number of essential cubes for SOP and DSOP forms of various benchmarks, obtained using Espresso.

Benchmark name	**Number of inputs**	**Number of outputs**	**# Cubes in SOP form**	**# Cubes in DSOP form**
5xp1	7	10	44	62
alu4	14	8	575	3551
b12	15	9	43	654
clip	9	5	120	359
cordic	23	2	914	22228
max1024	10	6	274	776
misex1	8	7	12	18
misex2	25	18	28	29
mlp4	8	8	128	206
rd53	5	3	31	31
rd73	7	3	127	127
rd84	8	4	255	255
x7dn	66	15	538	1697
xor5	5	1	16	16
Z9sym	9	1	86	190

Table 4.1: SOP and DSOP cubes for some combinational benchmarks, generated by Espresso

Benchmark	**[130]**	**[139]**	**[140]**	**[141]**	**[142]/[143]**	**[144]**	**Proposed**
5xp1	62	70	-	-	82	79	**48**
alu4	3551	-	-	-	1545	1372	**1206**
b12	654	**57**	-	-	60	60	62
clip	359	**162**	-	-	262	212	167
cordic	22228	-	-	-	19763	8311	**6687**
max1024	776	-	-	-	444	-	**362**
misex1	18	**15**	-	-	34	34	**15**

misex2	29	**28**	-	-	30	29	**28**
mlp4	206	-	-	-	203	-	**155**
rd53	**31**	**31**	-	-	35	35	**31**
rd73	**127**	**127**	-	-	147	147	**127**
rd84	**255**	-	-	-	294	294	**255**
x7dn	1697	-	-	-	**1091**	-	1228
xor5	**16**	-	**16**	**16**	**16**	**16**	**16**
Z9sym	190	-	186	**148**	-	-	171

Table 4.2: DSOP cubes of various combinational benchmarks corresponding to different methods

Table 4.2 lists the number of cubes in the DSOP solution, rendered by various heuristics for the combinational benchmarks listed in Table 4.1. The ‘dash’ in certain positions of the above tabular column indicates the unavailability of result for the benchmark corresponding to a specific method in the literature. The optimal solution for a benchmark based on a particular method(s) is highlighted in ‘bold-face’. From Table 4.2, it can be observed that the proposed heuristic has facilitated optimal/near-optimal solutions for a majority of problems. Amongst the following functions, *alu4*, *cordic*, *max1024* and *x7dn* are relatively bigger specifications, with *alu4*, *max1024* and *x7dn* having been classified as *hard* problems in the original Espresso benchmark suite. Firstly, in comparison with the DSOP solution rendered by Espresso, it can be seen that the proposed method enables a substantial reduction in the number of essential cubes by 65%. Compared to the SOP expression generated using Espresso, the proposed MDSOP heuristic based solution is found to be greater by 3.3×, while the DSOP solution of Espresso is more expensive than its logically equivalent SOP format by 9.5×. With respect to the larger problems, the proposed method enables a reduction in the number of essential DSOP cubes by 15.5% in comparison with the best solution rendered by the other heuristics. When considering all the benchmarks, the proposed heuristic facilitates a cost reduction of the order of 14.3%, on a mean basis.

4.4.1.7 MOSOP Cost of Function Blocks

The MOSOP form of an asynchronous logic block (function block) can be obtained by invoking the MDSOP heuristic for both the true and false rails of the encoded function block outputs. To this end, the MOSOP heuristic has been implemented in Java on the basis of the MDSOP procedure. Asynchronous dual-rail equivalents of several combinational benchmark specifications [147] [148] were considered to estimate the cost of their equivalent MOSOP forms, i.e. both the true and false rails of the dual-rail encoded combinational function outputs were considered. The MOSOP cost shown in column 4 of Table 4.3 reflects the number of unique cubes, of which some/many/all may be found to be shared between the various output rails. The count of the number of cubes, assuming no logic sharing, is also given in the tabular column (column 5). It can be seen that, on an average, the MOSOP heuristic has resulted in solutions, which encompass approximately 22% logic (cubes) sharing, but can be higher in specific cases. For example, in case of *ex5*, an 83% reduction in the number of cubes has been possible due to shared logic. Overall, the extent of logic sharing achieved appears to be significant considering the fact that the ON-set (including the DC-set) of a true output is non-identical with the OFF-set of a false output. However, in case of function blocks that comprise only two outputs (dual-rail), for example *9sym*, *newill*, *ryy6*, *sym10*, *t481* and *Z9sym*, it should be obvious that hardly any logic sharing is feasible.

The benchmarks highlighted in 'bold-face' correspond to dual-rail asynchronous equivalents of *hard* combinational logic specifications. MSOP forms corresponding to both the true and false rails of the encoded combinational function (function block) form the content of the input file of the package, while the output file reflects the MOSOP form of the entire function block, which also takes cognisance of the cubes that could be shared between various outputs. The runtime for obtaining the solutions of different problems is also given in Table 4.3. These correspond to the heuristic running in a Windows XP environment on a Dell Precision machine (T3400), installed with Linux and Windows OS alongside many software packages, consisting of an Intel Core2 Duo processor (2.4GHz) and a 1GB RAM. The processing time taken by the package primarily depends on the number of inputs and the logic description. Hence, functions with many inputs and a bigger logic specification would require more processing time than functions with many inputs and a smaller description. As a result, the number of outputs does not matter much compared to the number of inputs. A closer look at *9sym*, *newill*, *ryy6*, *sym10*, *t481* and *Z9sym* in terms of their number of inputs, resultant orthogonal product terms, and processing times substantiates the above observation.

The significance of the proposed scheme in comparison with other standard approaches, such as DIMS or Seitz's methods, may be understandable from the large size of the function blocks considered. For example, considering the benchmark *soar*, the proposed method results in only 1269 product terms, whereas DIMS or Seitz's approach would address the entire input space, which is massive as it is of O(2^{83}). Other strong [104] or weak-indication [110] approaches are unlikely to cope with such larger specifications.

Function block	**# Inputs in dual-rail format**	**# Outputs in dual-rail format**	**# Orthogonal product terms**		**Runtime (Minutes: Seconds)**
			After sharing	**Before sharing**	
5xp1	14	20	93	122	0:0
9sym	18	2	251	251	0:2
al2	32	94	310	472	0:0
alcom	30	76	134	226	0:0
alu4	**28**	**16**	**2711**	**2803**	**3:16**
amd	28	48	358	613	0:0
apex3	108	100	1351	2504	0:9
apla	20	24	205	246	0:0
b3	64	40	1298	1507	1:7
b12	30	18	102	109	0:0
bcd	52	76	6831	7942	10:15
chkn	58	14	523	526	0:17
clpl	22	10	41	41	0:0
cps	48	218	3390	5001	0:36
dk17	20	22	132	164	0:0
dk48	30	34	120	132	0:0
duke2	44	58	705	955	0:1
e64	130	130	2376	3033	3:55
ex4	**256**	**56**	**1062**	**1062**	**0:5**
ex5	16	126	346	2054	0:2
exep	60	126	3177	3591	0:27
gary	30	22	401	596	0:1
ibm	**96**	**34**	**1365**	**1366**	**0:14**
in3	70	58	496	813	0:1
in4	64	40	1396	1673	1:11
inc	14	18	82	134	0:0
intb	60	28	2320	2320	3:10
jbp	**72**	**114**	**636**	**869**	**0:1**
luc	16	54	175	496	0:0
max512	18	12	328	431	0:1
max1024	20	12	663	826	0:6
misex1	16	14	45	95	0:0
misex2	50	36	207	272	0:0
misex3	28	28	3826	4422	3:5
misg	**112**	**46**	**189**	**192**	**0:0**
mish	**188**	**86**	**163**	**173**	**0:0**
misj	**70**	**28**	**58**	**69**	**0:0**
mlp4	16	16	319	403	0:0
mp2d	28	28	166	196	0:0
newapla	24	20	82	94	0:0
newcpla1	18	32	126	159	0:0
newill	16	2	19	19	0:0

opa	34	138	572	1464	0:2
p82	10	28	78	164	0:0
pdc	**32**	**80**	**1358**	**1740**	**0:4**
rd53	10	6	46	71	0:0
rd73	14	6	190	294	0:0
rd84	16	8	365	589	0:2
risc	16	62	128	212	0:0
root	16	10	124	180	0:0
ryy6	32	2	155	155	0:4
sao2	20	8	202	258	0:0
shift	**38**	**32**	**200**	**212**	**0:0**
soar	**166**	**188**	**1269**	**1566**	**0:3**
spla	32	92	1577	2209	0:7
sqn	14	6	90	102	0:0
sym10	20	2	478	478	0:24
t1	42	46	384	501	0:0
t481	32	2	2142	2142	26:23
ti	**94**	**144**	**1388**	**2500**	**0:5**
ts10	**44**	**32**	**272**	**512**	**0:0**
vg2	50	16	632	647	0:6
x6dn	78	10	432	603	0:5
x7dn	**132**	**30**	**3711**	**3752**	**2:1**
Z9sym	18	2	243	243	0:1

Table 4.3: Cost of ST realisation of combinational benchmark functions

A recent method proposed for weak-indication circuit realisation of combinational logic [149], which also corresponds to two-levels assuming unbounded fan-in of primary input gates may also not be suitable for comparison, as it has targeted only function blocks with fewer inputs or addressed small clusters of bigger benchmarks. For example, the maximum cluster size derived from a combinational benchmark viz. C6288 [150], was limited to 16 dual-rail inputs and 18 dual-rail outputs while the encoded circuit consists of 64 dual-rail inputs and outputs. In fact, the method of [149] is prone to indirectly suffer from the problem of input space explosion and the function block size restricts its scalability. This is because, although it does not consider the entire input state space initially unlike that of [104] [151], but during the process of distributing the indication of inputs between different function block outputs, it is forced to selectively expand the reduced input cubes to accommodate the missing inputs. The process of selective expansion will have an adverse impact on the size of the function block depending on its initial specification, eventually making its realisation unrealistic to a great extent as it would be drastically affected by the exponentially expanding input space. In contrast, the proposed scheme has considered substantially bigger logic specifications (256 dual-rail inputs) that are significantly larger compared to those that could be processed by any of the above methods in a reasonable timeframe. Also, the evidence that only fewer cubes were required for realising even function blocks based on hard combinational problems highlights the fact that many cubes were found to be shared between the different outputs in the MOSOP form, based on the proposed heuristic. The issue of variable/literal ordering has not been considered as part of the proposal and it remains to be seen whether it could benefit in terms of facilitating more logic sharing. But this might be at the expense of increasing the computational complexity of the heuristic. It may be understandable that the function block realisation based on the proposed method has only ensured that the cover constraint is satisfied thereby avoiding the possibility for occurrence of gate orphans, but without considering the system indication aspect.

4.4.2 System Configuration

The system architecture that externally takes care of the indication phenomenon without casting the responsibility on the function block is shown in figure 4.8. Basically, this system configuration is robust and generally favours a weak-indication circuit realisation. In comparison with the topology shown in figure 4.4, there are two fundamental differences. With the exclusion of a dual-rail function block output that is synchronised with the output of the CD logic of the current stage (rather than the partial output corresponding to the CD logic of the current stage, as in the previous configuration) all other outputs can be directly fed to the next stage. The function block corresponds to two-levels of logic instead of being described in terms of multiple-levels as with the earlier architecture. Nevertheless, the system topology depicted in figure 4.8 could facilitate affordable implementation of function blocks of larger sizes, given the substantially reduced number of primary input cubes as opposed to considering the entire state space, which is evident from the values listed in Table 4.3. In fact, function block realisations of bigger logic specifications such as those mentioned in Table 4.3, on the basis of the other methods mentioned above would be highly cumbersome, as they would directly or indirectly encounter the problem of input space explosion, which has been largely annihilated in case of the proposed approach. It is assumed that there are no bounds imposed on the fan-in of first-level gates implementing primary input cubes while logical disjunctions can be constrained by bounds as that of [71] [91] [100] [149]. Also, given the system topology of figure 4.8, and considering a two-level implementation, the primary input cubes can be realised using relaxed conjunction operators rather than resorting to state-holding elements, which may be beneficial from area/power/delay perspectives.

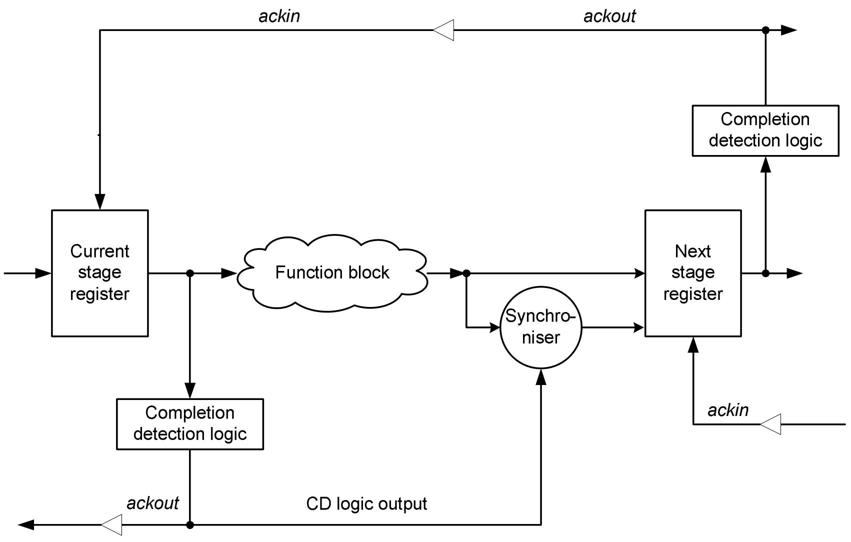


Figure 4.8: System topology in support of the proposed MOSOP heuristic for function blocks

The proposed two-level heuristic forms a good starting point for multi-level synthesis of combinational logic as weak-indication ST circuits, which nevertheless involves substantial complexity and difficulty as was echoed in [104] [149]. However, a preliminary solution has been put forward [152]. In simple terms, it basically amounts to finding suitable candidates for performing SI decomposition from within the MOSOP form of a function block specification whilst recognising the granularity of the base function set. Since the provision for SI decomposition may not always be manifest within the function block, SI decomposition may not always be feasible. Naïve decomposition cannot guarantee gate orphan freedom of the resulting solutions, rendering them Q^nDI, subsequently affecting the circuit's robustness, where '*n*' refers to the number of levels through which a gate orphan could propagate. It has been inferred from a number of case studies that though the proposed preliminary solution is not universal, it could be helpful in facilitating optimal solutions for some widely known logic functionality of arbitrary size, for example, multiplexer (MUX), demultiplexer (DEMUX), adder, encoder, decoder and comparator. An iterative nature is inherent in such functionality, probably due to the variable symmetricity, which allows them to be cascaded to derive higher order specifications. For example, the sum and output carry functions of a typical adder cell are symmetric[16] with respect to permutation of their variables.

Let us consider a small benchmark from the ISCAS '85 benchmark set, C17, to differentiate between the different approaches. The function block comprises 10 dual-rail inputs and 4 dual-rail outputs. Table 4.4 lists the delay, area and power metrics corresponding to various approaches. The optimal values of the design parameters corresponding to a particular approach are highlighted in 'bold-face' – this procedure shall be followed for all the subsequent tabular columns to be presented in this thesis. Since the synchroniser module, in effect, forms a part of the function block in terms of satisfying the indication constraints; the term 'datapath' would be used to refer to either the function block or a combination of the function block and the synchroniser (and also any other elements in the forward path).

The delay parameter refers to the maximum propagation delay encountered in the datapath (ideally, maximum delay encountered in the function block). The delay metric was estimated using PrimeTime. To avoid the notion of a clock source, the option of a virtual clock was used that only acts as a remote reference to restrain the input and output ports of the design. To ensure that all the valid data paths of the function block design will be reported, the dynamic loop breaking technique has been utilised [153]. Thus, no timing arc disabling was reported when static timing analysis was performed; dynamic timing analysis is not possible within the PrimeTime environment. The area and power metrics correspond to the input registers, CD logic and function block. The delay and power metrics consider estimated parasitics (resulting from enabling of the automatic wire load selection feature), in addition to the parameters associated with the actual components. The area metric gives a combined account of the area of all the logic cells and has been obtained as part of the PrimeTime framework. The total/average power dissipation is the summation of dynamic and static power components, where dynamic power is in turn the gross of switching and internal power consumption figures. NC-Verilog has been used for functional simulation and also to obtain the switching activity files corresponding to the gate level simulations of Verilog descriptions.

[16] A function F is totally symmetric, if any permutation of the variables in F does not change its definition. In a partially symmetric function, any permutation of a subset of the variables does not alter its specification.

Input data were applied to the function blocks at specific intervals through test benches, which model the environment. The switching activity files obtained were subsequently used for power estimation using PrimeTime PX. For all the simulation results to be presented in this thesis, which would correspond to various function blocks, the design parameters estimation mechanism detailed above has been adopted uniformly.

The test bench for the C17 function block corresponds to the input trace of its logic specification and the inputs were fed to the circuit at intervals of 2ns. The simulations targeted the best-case PVT corner (supply voltage = 1.32V, junction temperature = -40°C) of the high-speed 130nm Faraday (UMC) bulk CMOS standard cell library. All the circuit inputs were configured to possess the driving strength of the minimum sized inverter of the cell library, while the outputs were associated with fanout-of-4 (FO-4) drive strength. Suitable buffering for the acknowledge input was provided, where necessary, to eliminate timing violations. Furthermore, the logic corresponding to function blocks of various methods were optimised for minimum delay. For all the forthcoming simulation results to be mentioned in this thesis, the above drive capability for the function block inputs and outputs, manual delay oriented technology dependent logic optimisation of the function blocks corresponding to different methods, and similar electrical library specification can be assumed. Since identical registers and a similar CD circuit were used for all the realisations, the area and power metrics can be deemed to approximately reflect that of the actual ST combinational logic, paving the way for a legitimate comparison between different ST logic realisation schemes.

With only two levels of logic depth required for the physical implementation of C17 block, Architecture_C17 (strong and weak) refers to the C17 circuit realisation based on the strong and weak-indication circuit architectures discussed in sections 4.3.1 and 4.3.2 respectively, while MOSOP_C17 (strong and weak) denotes the C17 function block realisation based on the architecture discussed earlier through the illustration in figure 4.8. It can be observed from Table 4.4 that the MOSOP_C17 (weak) design features the least datapath delay whilst being beneficial in terms of area and power metrics as well. The main reason for the optimality of MOSOP solutions is attributed to the presence of relaxed gates in the first logic level.

Realisation style	**Delay (ns)**	**Area (μm^2)**	**Power (µW)**
Architecture_C17 (strong)	1.4	1232	432.6
Architecture_C17 (weak)	1.3	888	352.1
MOSOP_C17 (strong)	1.3	364	257.8
MOSOP_C17 (weak)	**1.2**	**340**	**233.9**

Table 4.4: Delay, area and power metrics corresponding to different approaches for implementing C17 function block

Let us consider two more problem cases viz. 32:1 MUX logic and 1:32 DEMUX logic, to strengthen the previous observation. The MUX function block considered here consists of 74 inputs and 2 outputs in dual-rail format, while the DEMUX function block considered comprises 12 inputs and 64 outputs. Table 4.5 lists the delay, area and power parameters of the MUX logic realised using various approaches.

Realisation style	**Delay (ns)**	**Area (μm^2)**	**Power (µW)**
Seitz method (strong)	2.5	5130	1805.9
DIMS method (strong)	2.8	8306	1924.3
Toms method (strong)	3.0	7105	1734.1
SIMCO (strong)	**1.8**	3416	931.8
SIMCAO (strong)	**1.8**	**3048**	**880.1**

Table 4.5: Delay, area and power metrics corresponding to different ST approaches for realising 32:1 MUX function block

SIMCO refers to the direct **s**trong-**i**ndication design for the **M**UX logic utilising only **C**-gates and **O**R gates, while SIMCAO refers to the direct **s**trongly **i**ndicating **M**UX realisation featuring **C**-gates, **A**ND gates and **O**R gates. For both these designs, the proposed MOSOP heuristic is used greatly reducing the input space requirements. Since direct function block realisation is not possible with other approaches, given the large number of inputs, tree structures were resorted to – four 8:1 MUXes in the first level and a 4:1 MUX in the second level. In turn, an 8:1 MUX was composed from two 4:1 MUXes in the first level and a 2:1 MUX in the second level, with the 4:1 MUX evolved as a tree structure involving two 2:1 MUXes in the first level and a 2:1 MUX in the second level. It can be observed from Table 4.5 that both the proposed realisations exhibit less datapath delay, area and power metrics, with the SIMCAO MUX block outperforming the other implementations. The inputs were fed into the MUX function block at intervals of 4ns and the test bench comprised all the unique input sequences that are possible for the MUX functionality. Tree structures based on the proposed approach would not be optimal with respect to delay/power due to increased number of logic levels. For example, the efficient 32:1 MUX logic tree based on the proposed approach, constructed using a cascade of 8:1 MUXes and a 4:1 MUX resulted in a delay of 1.9ns, area of 2766µm^2 and total power dissipation of 982.4µW.

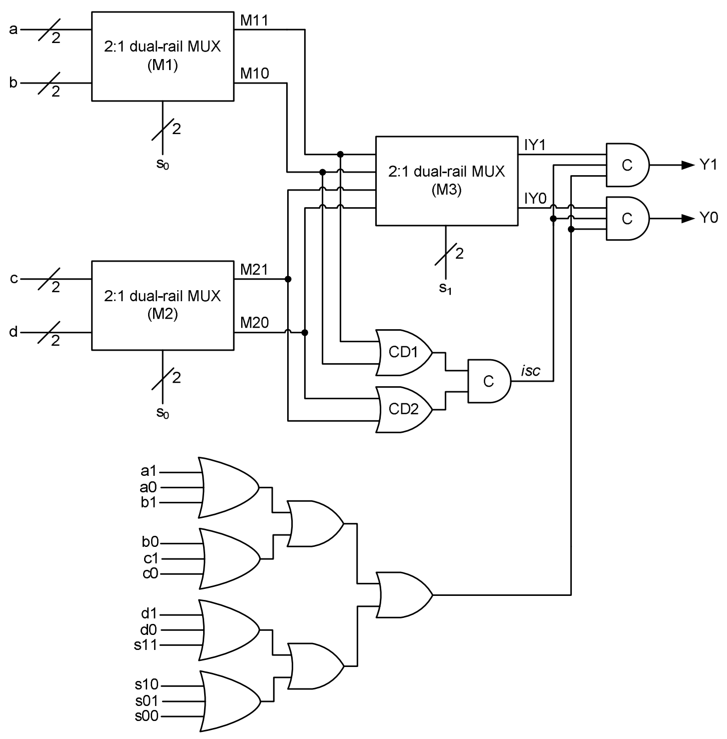


Figure 4.9: Illustrating CD of intermediate outputs for MUX logic tree based on Seitz's method

DIMS and Toms' approaches typically employ only C-elements and OR gates, whereas Seitz's method incorporates AND-OR logic. As a result, a simple cascade of MUXes would lead to gate orphans in case of the latter. Hence, in order to avoid the possibility of gate orphans, local completion detectors (CD1 and CD2) were provided as shown in figure 4.9 for a 4:1 MUX logic block. Here, (M11, M10) and (M21, M20) represent the intermediate outputs of the 2:1 MUXes in the first level of the cascade, while (Y1, Y0) represent the actual output of the function block. The internal signal '*isc*' guarantees the arrival of the intermediate MUX outputs before the actual outputs are produced. It is to be noted that timing assumptions are made with regard to ensuring completion detection of the primary dual-rail inputs.

Considering the 1:32 DEMUX logic, function blocks corresponding to different approaches were derived in a similar way. Two types of direct implementations were preferred based on the proposed MOSOP heuristic as tree structures were found to degrade the delay metric – WIDCO (**w**eak-**i**ndication **D**EMUX realisation with only **C**-gates and **O**R gates) and WIDCAO (**w**eakly **i**ndicating **D**EMUX logic based on **C**-elements, **A**ND gates and **O**R gates). Weak-indication realisations for the DEMUX logic block are possible due to the presence of multiple outputs, unlike the case with MUX logic. The tree structures for the DEMUX logic corresponding to the other methods are the converse of the MUX structures – a 1:4 DEMUX in the first level of the cascade and four 1:8 DEMUXes in the second level of the cascade. But unlike the latter, the logic granularity is relatively less for the present case, i.e. the 1:4 and 1:8 DEMUXes were directly synthesised based on the other methods. Internal CDs were provided for the DEMUX logic constructed using Seitz's approach in a manner similar to that described earlier for the MUX logic, to ensure gate-orphan freedom. Table 4.6 gives the delay, area and power metrics for the different DEMUX function block implementations. The test patterns consisted of unique input sequences that would detect the passage of data to all the dual-rail outputs, and were fed at intervals of 4ns. Again, for the DEMUX logic, the proposed approach was found to yield superior results with respect to delay, area and power metrics, as could be seen below.

Realisation style	Delay (ns)	Area (μm^2)	Power (μW)
Seitz method (weak)	1.6	2002	559.9
DIMS method (strong)	2.0	3460	720.4
Toms method (strong)	2.0	2022	612.1
WIDCO (weak)	**1.3**	1535	334.1
WIDCAO (weak)	1.4	**1285**	**322.4**

Table 4.6: Delay, area and power corresponding to different ST approaches for realising 1:32 DEMUX function block

4.5 Summary

This chapter first discussed general multi-level synthesis models for strong or weak-indication implementations of arbitrary combinational logic by employing SI decomposition. However, to overcome the problem of state space explosion inherent in the multi-level technique, an efficient set theory based approach has been presented that is useful for realisation of function blocks featuring several concurrent inputs. The set theory based MOSOP heuristic forms the basis for adder, carry-lookahead module and compressor realisations that would be discussed in the subsequent chapters of this thesis. The set theory based MOSOP method greatly alleviates the problem of input space explosion and facilitates compact realisations of larger function block specifications, which is otherwise cumbersome with many other techniques. Some sample case studies were considered to demonstrate the benefits of the proposition.

Chapter 5

Self-Timed Carry-Ripple Adders

A study of the operations performed by an ARM processor's ALU revealed that additions constituted nearly 80% of them [154]. About 72% of the instructions of a prototype RISC machine resulted in addition/subtraction operations [155]. In fact, addition was found to be the most frequently encountered operation amongst a set of real-time digital signal processing benchmarks [156]. In general, integer addition plays a very important and dominant role in digital computer systems.

In this Chapter, we shall extensively consider adder cells synthesised using various approaches that form the fundamental datapath elements, and evaluate their performance based on the carry-ripple or ripple carry adder (RCA) topology. The impact of a dual-bit adder cell in reducing the delay metric of the basic adder topology is investigated and the bottleneck in extending the hierarchy further is discussed. Single-bit[17] and dual-bit[18] adders based on homogeneous and heterogeneous DI data encoding schemes are considered and a comparative analysis is performed. Also, the usefulness of a hybrid combination of single-bit and dual-bit adders within a RCA structure is studied, which is found to feature only a minor optimisation potential. Finally, the concept of redundant logic insertion, introduced with the aim of further minimising the delay of the addition operation is elucidated through case studies. In general, it could help in minimising the latency of iterative logic circuits.

5.1 Single-Bit Adders and Their Evaluation

We shall first discuss single-bit adders that are based on dual-rail encoding, which shall be followed by a discussion of adders employing hybrid input encoding that involves a combination of dual-rail code and 1-of-4 code.

[17] The term 'single-bit adder' is used to emphasise the point that the adder is used to add one augend and one addend bit at a time along with an extra input carry. It is commonly called the *full adder*.
[18] The term 'dual-bit adder' is used to refer to an adder that can add two augend and addend bits simultaneously, taking into account the extra carry input as well.

5.1.1 Using Dual-Rail Data Encoding

Let us first consider the logical equations corresponding to the outputs of a full adder.

$$Sum1 = a0b0cin1 + a0b1cin0 + a1b0cin0 + a1b1cin1 \qquad \textbf{(5.1)}$$

$$Sum0 = a0b0cin0 + a0b1cin1 + a1b0cin1 + a1b1cin0 \qquad \textbf{(5.2)}$$

$$Cout1 = a0b1cin1 + a1b0cin1 + a1b1cin0 + a1b1cin1 \qquad \textbf{(5.3)}$$

$$Cout0 = a0b0cin0 + a0b0cin1 + a0b1cin0 + a1b0cin0 \qquad \textbf{(5.4)}$$

From the equations listed above, it can be understood that an adder circuitry strictly realising the products using C-elements adheres to the strong-indication timing regime and therefore strong-indication adders are bound by constant delay. However, by distributing the indication between sum and carry outputs, as the carry outputs are required to propagate between successive stages in a typical adder cascade, the output carry can be optimised as,

$$Cout1 = a0b1cin1 + a1b0cin1 + a1b1 \qquad \textbf{(5.5)}$$

$$Cout0 = a0b1cin0 + a1b0cin0 + a0b0 \qquad \textbf{(5.6)}$$

Thus, the weak-indication adder can take into account the fact that the output carry of an adder module could become defined as soon as its input operands become defined, depending on carry-kill ($a0=b0=1$) or carry-generate ($a1=b1=1$) conditions. In simple terms, whenever the carry-propagate ($a0=b1=1$ or $a1=b0=1$) condition does not occur, the dual-rail carry output of any k^{th} stage of a n-bit adder could become defined as soon as its primary input operands become defined and thereby the sum outputs of the $(k+1)^{th}$ stage could become defined with its primary input operands also becoming defined concurrently. As a result, the lengthy carry propagation chain is avoided under this ideal condition. Acknowledging the fact that worst-case carry propagation may not always occur, the computation speed of weak-indication adders could vary depending on data characteristics. Thus, in contrast to strong-indication adder blocks, weakly indicating adder blocks could take advantage of data dependent computations. Though one could say that the worst-case carry propagation encountered in a weak-indication adder is approximately $O(\log_2 n)$, where n signifies the adder size or word length, it generally represents the lower bound on the worst-case time complexity for binary addition [157]. Therefore, the average time taken for sum outputs to evaluate, that varies as $\log_2 n$ after the input operands become defined, holds good mainly for random data inputs [154] [158]. In case of [154], for typical data processing operations on a

32-bit asynchronous ALU (which formed a part of the AMULET processor utilising bundled-data encoding protocol), the carry chain was found to be approximately 18 bits long, i.e. greater than $n/2$, whereas address calculations entailed a chain length of about 9 bits viz. carry propagation greater than $n/4$, for the Dhrystone benchmark. For input vectors corresponding to some benchmarks, it was found in [158] that many carry chains exceeded a propagation length of 8 $(n/4)$ and many were about 24 bits long $(3n/4)$; though some demanded propagation over the entire length of the carry chain.

Strong-indication adder designs always exhibit worst-case latency; weak-indication counterparts will be faster but their delay depends on the input vectors as actual case latency is a feature inherent in case of relaxed adders (example, weakly indicating adders). Since dynamic timing analysis is not feasible and early termination effects cannot be captured whilst using an industry standard gate level timing analyser, as mentioned in the previous Chapter, we adopt a rather convenient policy of evaluating the different adders statically. But at the same time, we acknowledge the fact those adders satisfying Seitz's weak-indication criteria are preferable since the carry logic would have been optimised for faster carry propagation between successive stages. However, we compare the weak-addition adders with their strong-indication counterparts with respect to power dissipation, assuming a random input data distribution corresponding to the lower bound. In fact, proposals for data path optimisation addressing reduction of critical path delay in relaxed but robust asynchronous circuits (both logic and arithmetic circuits) have been put forward in recent literature using an NCL style [122] [128]. The objective therein has been to reduce circuit latencies for worst-case data processing. Probably, the motivation for this might have been due to the fact that latency and area overheads are generally higher for robust asynchronous circuits compared to their non-robust counterparts due to their inherent style of construction. In this context, an interesting analogy can be found in the design of a carry-lookahead adder that promises reduced (logarithmic) maximum datapath delay in comparison with a basic carry-ripple adder (linear). Discussion about the former in the context of ST logic is done in the next Chapter.

A number of full adders corresponding to various ST approaches were realised using standard cells and evaluated on the basis of the fundamental carry-propagate adder viz. RCA topology. The architecture of the *n*-bit ST RCA is depicted in figure 5.1. The adder on the right-hand-side represents the least significant position and that on the left-hand-side specifies the most significant position. The carry-propagates in a ripple fashion from the least significant to the most significant adder, while the sum output of a particular stage is produced depending on the value of the augend and addend bits of that stage inclusive of its carry input.

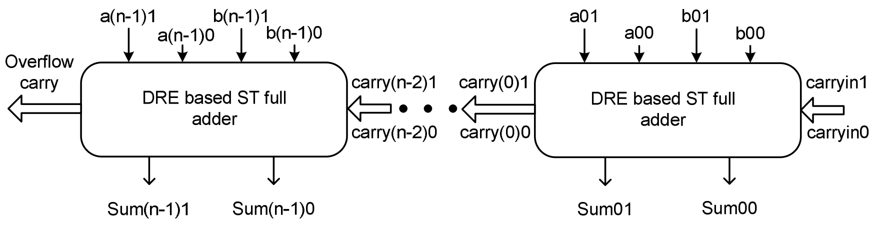


Figure 5.1: *n*-bit dual-rail encoded ST carry-ripple adder architecture

Next, we discuss the transformation of a SI adder into an indicating function block. Petrify [83] is a tool predominantly used for the SI synthesis of asynchronous control logic but is unsuitable for synthesising datapath logic due to its inability to handle many concurrent inputs. However, the full adder functionality could be synthesised in a SI fashion. To achieve this, a pure, unique choice Petri Net model of the same has been first described. Next, to perform the synthesis routine and to maintain uniformity, a sample library in *genlib* format, comprising the elements of the 130nm UMC standard cell library was created and the technology mapping process was set to target the gates of this sample library. Nevertheless, the resulting circuitry is purely SI and so is unable to properly indicate the complete arrival of all the inputs on its outputs, collectively or non-collectively without any ambiguity. For example, when the circuit has to shift from the valid data state to the spacer state, even with any dual-rail input assuming the spacer state, all the dual-rail primary outputs can be reset after all the intermediate outputs have stabilised giving rise to ambiguity with regard to determining the correct steady state of all the dual-rail primary inputs. However, assuming the circuit to be initially in the spacer state and with the application of valid data, only after the sufficient arrival of the required dual-rail inputs, the encoded outputs would be asserted ‘high’. Hence, this circuit attribute is similar to that which is seen in a typical Seitz’s implementation and therefore, an extra CD logic needs to be included (shown enclosed within dotted lines in figure 5.2) for synchronisation with at least a dual-rail output (in this case, the sum output) to impose indication constraints on the module.

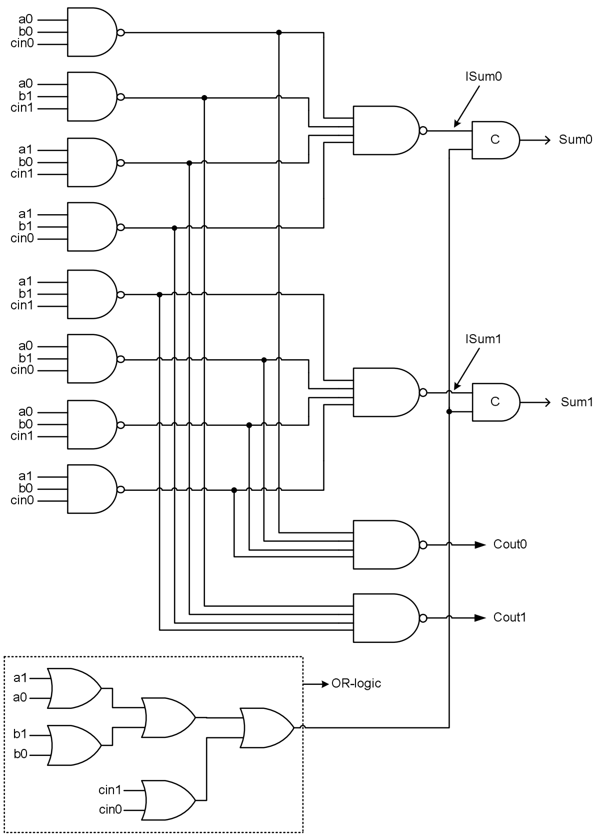


Figure 5.2: Weakly indicating realisation of Petrify based full adder block

The proposed full adder design (henceforth referred to as the SSSC_DRE adder – single-sum, single-carry adder based on DRE) is portrayed by figure 5.3 [180]. It was manually designed in a semi-custom style in order to investigate two important issues: i) how the responsibility of indication can be confined to the sum output alone, thereby freeing the carry signal from indication constraints, and ii) how logic redundancy can be made implicit in a ST design to enable reduction in latency. It can be noticed that the sum output is strongly indicating, while the carry output is eager and synthesised by means of a complex gate, viz. an AO222 cell. Even with the arrival of any subset of the inputs, the carry outputs could become defined/undefined, while the sum outputs would wait for the arrival of all the inputs to become defined/undefined. Thus, the full adder, on the whole, satisfies Seitz's weak-indication timing constraints and there is no distribution of inputs indication. This style of implementation was later labelled as the biased approach in [128] targeting ST circuit optimisation at a block level. However, our method differs from the above approach in that the latter relies upon DRCL style for realising those outputs, which have been chosen as candidates for relaxation. The

SSSC_DRE adder has also served as the forerunner for the analysis and proposition of the novel concept of logic redundancy insertion (explicit), dealt with in detail in section 5.5.

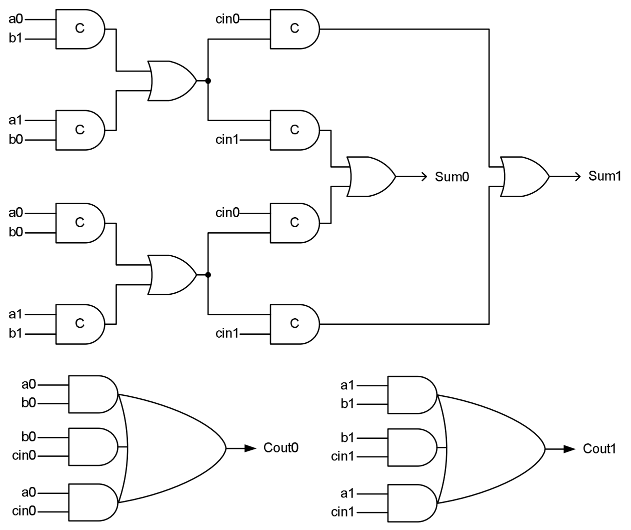


Figure 5.3: Proposed weak-indication full adder module

Full adders corresponding to different ST approaches have been constructed in a semi-custom fashion and have all been optimised for delay (latency, as described in section 4.4.2), individually, so as to pave the way for a straightforward comparison. Table 5.1 gives the maximum datapath delay, area and power parameters of the various full adder modules for performing 32-bit addition based on the ST RCA topology. The nature of indication of the various adders is also given in Table 5.1. The test bench corresponds to the input trace of a simple combinational benchmark, *dc1*, which comprises thirty eight input sequences and the test vectors were supplied to the adders at intervals of 25ns. In these, the input profile of the combinational function was duplicated on the least significant nibble position of the dual-rail adder bits, while the more significant adder stages were assigned with zeroes (i.e. the false rails of the dual-rail adder inputs were assigned with 1's, whilst the true-rails of the dual-rail adder inputs were assigned with 0's). Similar input patterns were used for all the adder simulations performed as part of this research work. The presence of an input carry was assumed for approximately half of the input patterns. Referring to [154], it can be seen that application of random input data resulted in a maximum carry propagation of about 4-bits for an adder of size 32-bits in an ARM processor case study, which is found to be typical of operations such as memory address calculations and branching operations. The input profile of the logic function considered, when used to govern the inputs to an adder block is found to limit the maximum carry propagation length to the least significant nibble position.

Adder realisation style	Delay (ns)	Area (μm^2)	Power (μW)
Seitz_DRE (strong)	12.8	8329	493.1
Seitz_DRE (weak)	6.5	7689	445.6
Singh_DRE (strong)	10.7	8297	444.4
Modified David et al._DRE (strong)	20.8	11753	895.1
Modified David et al._DRE (weak)	15.0	10985	833.2
DIMS_DRE (strong)	13.8	10089	415.3
DIMS_DRE (weak)	12.8	10665	462.8
Petrify_DRE (strong)	13.3	8009	491.5
Petrify_DRE (weak)	7.0	7241	433.5
Folco et al._DRE (weak)	8.0	**6633**	**371.7**
Toms_DRE (strong)	10.6	7561	376.9
SSSC_DRE (weak)	**5.8**	7081	407.7

Suffix DRE is used to explicitly convey the fact that the adders are dual-rail encoded.

Table 5.1: Delay and area metrics corresponding to different 32-bit ST RCAs

It may be useful to know that the adder circuits were optimised with the motive of maintaining a balance between effective logic sharing and minimal logic depth while giving more preference to the latter. The difference in delay between various adders is attributable to the elements present in their respective critical datapaths, which are given below.

Adder realisation style	Critical path elements
Seitz_DRE (strong)	OR2+OR2+CE2
Seitz_DRE (weak)	AND3+OR3
Singh_DRE (strong)	CE2+OR3
Modified David et al._DRE (strong)	OR2+CE3+CE2
Modified David et al._DRE (weak)	OR3+CE2+CE2
DIMS_DRE (strong)	CE3+OR2+OR2
DIMS_DRE (weak)	CE3+OR3
Petrify_DRE (strong)	OR2+OR2+CE2
Petrify_DRE (weak)	NAND3+NAND4
Folco et al._DRE (weak)	CE2+OR2
Toms_DRE (strong)	CE2+OR2+OR2
SSSC_DRE (weak)	AO222

Table 5.2: Listing critical path elements of the various ST RCAs

As can be seen from Tables 5.1 and 5.2, weak-indication adders are preferable in comparison with strong-indication adders from the delay perspective. This is obvious in case of Seitz's, modified David et al.'s, DIMS and Petrify based adders. Amongst all the adders, modified David et al.'s adders (taking into account the simple modifications to David et al.'s method, as mentioned in section 3.6) were found to be inferior in terms of delay and also high in power consumption; this is because of the OR-CE logic, used to realise the false rails of the sum and carry outputs, which resulted in considerably higher switching activity as all the intermediate OR gate outputs would have to be asserted 'high'. The false bits of the sum and carry outputs were not realised in a robust fashion using complex gates, due to the unavailability of suitable gates in the cell library. In terms of area and power dissipation, Folco et al.'s adder is found to be the best. It can be inferred that the proposed adder features the least datapath delay among all the other adders, reporting a reduction in comparison with the weakly indicating Seitz's adder by 11%. At the same time, the SSSC_DRE adder occupies less area than Seitz's weak-indication adder by 15% and dissipates less power to the tune of 8.5%.

5.1.2 Employing Hybrid Input Data Encoding

The term 'hybrid input encoding' (HIE) refers to a mix of at least two different DI data encoding schemes as adopted for the inputs. Considering the single-bit adder block, the augend and addend bits can be encoded using a 1-of-4 code, while the carry input, sum and carry outputs can adopt the dual-rail code, i.e. hybrid or heterogeneous encoding of primary inputs and homogeneous encoding of primary outputs are done. The motivation for the use of a 1-of-4 code being reduced switching activity in comparison with a simple dual-rail code; however, given the extra encoding circuitry required for the 1-of-4 code with respect to the dual-rail code (which serves as the base or reference encoding protocol), the power savings gained are found to be substantially reduced for the case study of a 32-bit ST RCA.

The structure of the *n*-bit hybrid input encoded ST RCA is shown below, which is similar to the topology shown in figure 5.1, but with the only exception that the augend and addend single-rail inputs are now encoded using a 1-of-4 code.

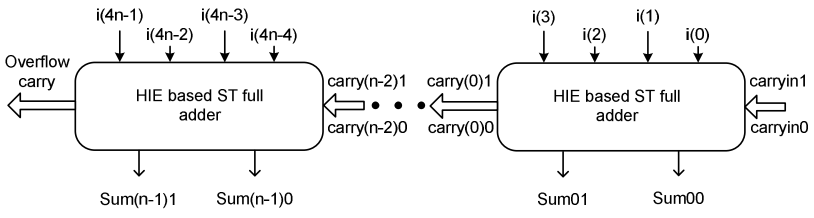


Figure 5.4: Architecture of the *n*-bit hybrid input encoded ST RCA

The basic equations governing a full adder block utilising HIE are as follows:

$$Sum1 = i3cin1 + i2cin0 + i1cin0 + i0cin1 \quad \textbf{(5.7)}$$

$$Sum0 = i3cin0 + i2cin1 + i1cin1 + i0cin0 \quad \textbf{(5.8)}$$

$$Cout1 = i2cin1 + i1cin1 + i0cin0 + i0cin1 \quad \textbf{(5.9)}$$

$$Cout0 = i3cin0 + i3cin1 + i2cin0 + i1cin0 \quad \textbf{(5.10)}$$

In the above expressions, (*i*0, *i*1, *i*2, *i*3) represents the 1-of-4 encoded equivalent of the augend and addend inputs (*a*, *b*), with a similar encoding mechanism adopted as shown in Table 2.1. The full adder block realising the above equations using C-gates (based on a direct translation from DRE to hybrid encoding representation) would be dubbed strongly indicating. The hybrid input encoded full adder based on Toms' approach is shown below.

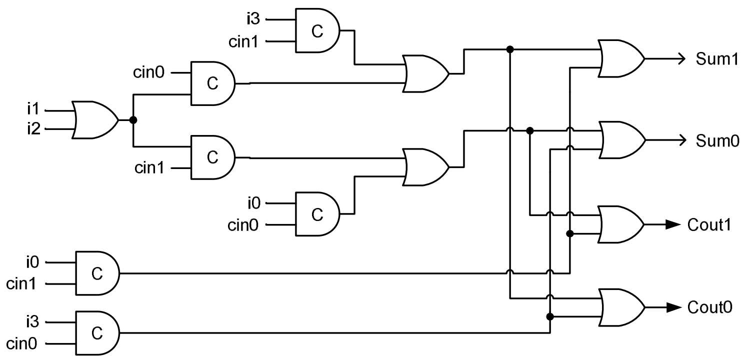


Figure 5.5: Hybrid input encoded full adder based on Toms' approach

An analysis of HIE for a full adder is undertaken and the proposed adder design based on hybrid encoding of input data, that features carry output logic optimisation is shown in figure 5.6. Henceforth, it shall be identified as the SSSC_HIE_NRL adder, where the acronym NRL expands as non-redundant logic. This is to emphasise that all the gates that are a constituent of this adder are non-redundant, and also to distinguish it from another version of a SSSC_HIE full adder that incorporates redundant gates, which shall be discussed in the later portion of this Chapter (in section 5.5). It could be observed that the sum outputs are entrusted with the responsibility of inputs indication, while the carry output could evaluate to the correct state when carry-kill or carry-generate condition occurs, without having to wait for the input carry signal.

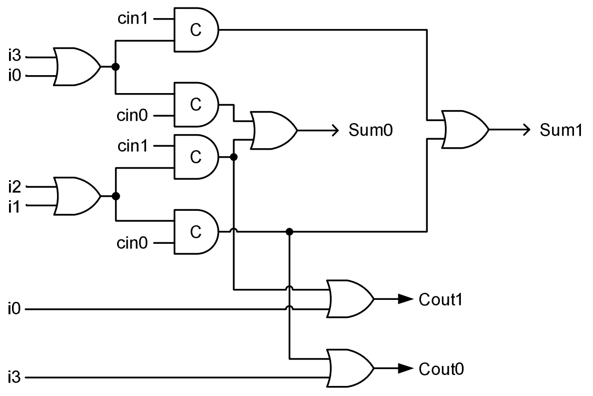


Figure 5.6: Proposed hybrid input encoded full adder block

The system configuration that supports embodying HIE for inputs and DRE for outputs is shown below, which is a slightly modified version of the typical ST system architecture.

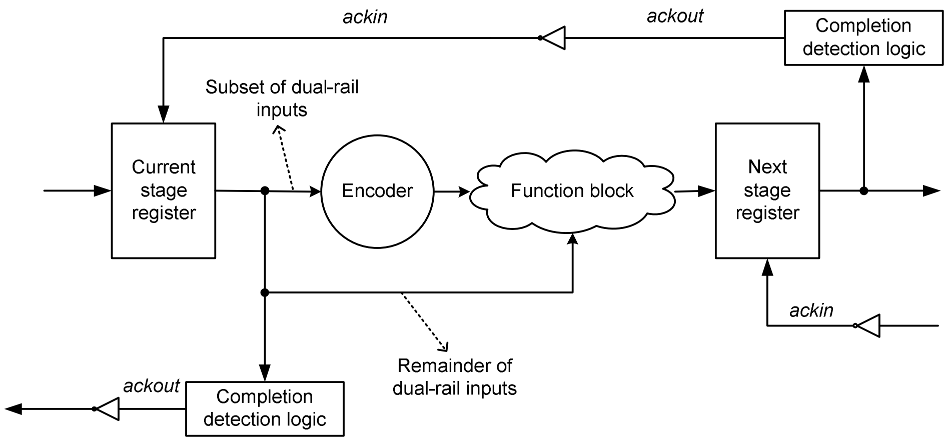

ackin
ackout
Completion detection logic
Subset of dual-rail inputs
Current stage register
Encoder
Function block
Next stage register
ackin
Remainder of dual-rail inputs
ackout
Completion detection logic

Figure 5.7: ST system architecture highlighting input protocol conversion and data processing

A subset of the actual dual-rail inputs (here, augend and addend inputs) are fed to the protocol conversion circuit for data encoding using a 1-of-4 code and then supplied to the function block. The remnant of the dual-rail inputs (here, input carry) is fed as it is to the function block for processing, along with the 1-of-4 encoded inputs. The function block outputs, which are purely dual-rail encoded are stored and subsequently fed to the next stage.

In Table 5.3, the delay, area and power figures *specified within brackets* are the ones to be considered and correspond to the case when the extra encoding logic is also included to obtain the 1-of-4 encoded data from the dual-rail encoded data. The values given outside brackets refer purely to the delay, area and power parameters of the hybrid input encoded adders that do not have the extra encoding circuitry, and are just listed here to highlight the offset in power savings when encoding is employed. The cost of encoding is 28 transistors per bit, due to the implementation of the CE2 functionality by means of a complex gate. A similar test bench was used as that of the earlier case and the test vectors were also fed at the same rate. The SSSC_HIE_NRL adder features the least delay and area in comparison with Toms_HIE adder, benefitting from the weak-indication phenomenon, with the latter suffering from 35% more delay and an increase in area overhead by 14%. Nevertheless, the average power components of both the adders are comparable. Comparison with adders based on other ST methods is not possible as they basically correspond to the dual-rail signalling convention, while Toms' approach is suitable for realising circuits employing arbitrary *m*-of-*n* codes.

Adder realisation style	**Delay (ns)**	**Area (μm^2)**	**Power (μW)**
Toms_HIE	10.6 (10.8)	5721 (7561)	**224.0** (374.4)
SSSC_HIE_NRL	**7.8 (8.0)**	**4793 (6633)**	224.8 **(371.9)**

Table 5.3: Delay, area and power metrics of 32-bit ST RCAs incorporating HIE

It might be interesting to study the switching power aspect of the different adders to ascertain how the hybrid input encoded adders (whilst incorporating the protocol conversion circuit) would fare in comparison with the dual-rail encoded adders. The results of this analysis are depicted in figure 5.8. It can be seen that SSSC_HIE_NRL and Folco et al._DRE adders exhibit relatively less switching power in comparison with all the other adders.

Comparing Tables 5.1 and 5.3, it can be noticed that for performing a 32-bit ST addition operation, the SSSC_DRE adder is preferable to the SSSC_HIE_NRL adder as the former exhibits reduced latency in comparison with the latter by 28%. On the other hand, with respect to area and total power consumption, the latter reports less area occupancy and lower total power dissipation by 6.3% and 8.8% respectively, thus maintaining area and power advantage over the former.

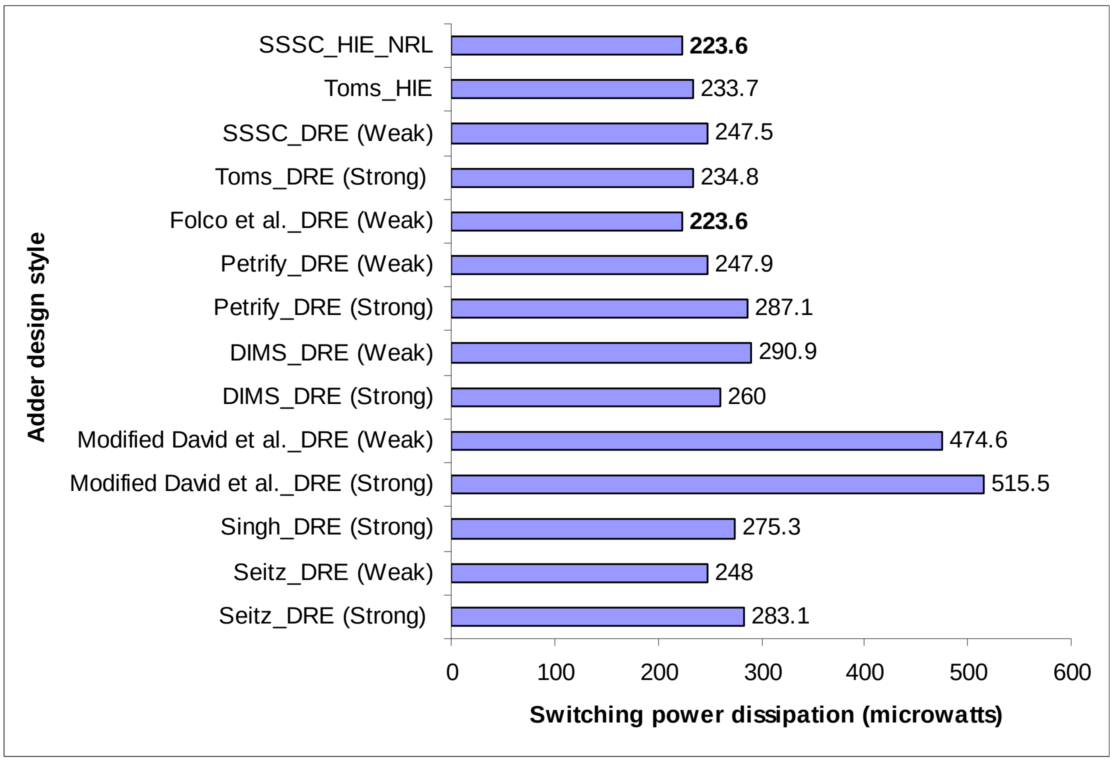


Figure 5.8: Analysis of switching power dissipation of various 32-bit ST RCAs

5.2 Dual-Bit Adder Designs and Their Evaluation

The main motivation for the dual-bit adder stems from the observation that if a simple series cascade of dual-bit adders is envisaged, then the number of stages that the carry signal has to traverse would be halved in comparison with a linear cascade employing only conventional single-bit adders. The approach is interesting as it has the potential to roughly halve the worst-case datapath delay of an *n*-bit carry-ripple adder employing full adders, assuming stage delay to be the same, where '*n*' is the adder size. Nevertheless, this would be feasible only at the expense of an area increase as the input state space now gets quadrupled even though the number of stages is halved. To examine this issue further and to comment on its usefulness, dual-bit adder designs based on homogeneous and heterogeneous data encoding schemes are considered and analysed in detail in this section. While *homogeneous encoding* procedure refers to a similar DI data encoding scheme adopted for all the inputs and outputs of a function block, *heterogeneous encoding* mechanism refers to a combination of at least two different DI encoding schemes, as adopted for the inputs and outputs. The dual-bit adder shall either employ dual-rail data encoding (homogeneous encoding scheme) for all its inputs and outputs or a combination of dual-rail and 1of-4 encoding for its inputs and outputs (heterogeneous encoding scheme).

5.2.1 Adopting Dual-Rail Data Encoding

A dual-bit adder block basically consists of five single-rail inputs $a1$, $a0$, $b1$, $b0$ and cin and three single-rail outputs $Cout$, $Sum1$ and $Sum0$, where ($a1$,$a0$) and ($b1$,$b0$) could represent the addend and augend inputs and cin, the carry input. Output $Cout$ is the overflow bit or output carry signal of the addition process, and $Sum1$ and $Sum0$ are the most significant and least significant sum outputs respectively.

The MOSOP form for the dual-rail encoded dual-bit adder block is given below. Out of 46 cubes comprising the different encoded outputs, 12 cubes are found to be shared and thus a total of 34 distinct cubes are found to comprise the MOSOP form of the dual-bit adder block. Decomposition of larger cubes can be subsequently performed in a SI fashion [152] and logic sharing is feasible, since suitable candidates could be ascertained from within the MOSOP expressions corresponding to the encoded outputs.

$$\begin{aligned} Cout1 = {} & a10a00b11b01cin1 + a11a00b10b01cin1 + a10a01b11b00cin1 + a11a01b10b00cin1 \\ & + a10a01b11b01 + a11a01b10b01 + a11b11 \end{aligned} \tag{5.11}$$

$$\begin{aligned} Cout0 = {} & a11a01b10b00cin0 + a10a01b11b00cin0 + a11a00b10b01cin0 + a10a00b11b01cin0 \\ & + a11a00b10b00 + a10a00b11b00 + a10b10 \end{aligned} \tag{5.12}$$

$$\begin{aligned} Sum11 = {} & a11a01b10b00cin0 + a10a01b11b00cin0 + a11a00b10b01cin0 + a10a00b11b01cin0 \\ & + a11a00b11b01cin1 + a11a01b11b00cin1 + a10a00b10b01cin1 + a10a01b10b00cin1 + \\ & a10a01b10b01 + a11a00b10b00 + a10a00b11b00 + a11a01b11b01 \end{aligned} \tag{5.13}$$

$$\begin{aligned} Sum10 = {} & a11a01b10b00cin1 + a10a01b11b00cin1 + a11a00b10b01cin1 + a10a00b11b01cin1 \\ & + a10a01b10b00cin0 + a10a00b10b01cin0 + a11a01b11b00cin0 + a11a00b11b01cin0 + \\ & a11a00b11b00 + a11a01b10b01 + a10a01b11b01 + a10a00b10b00 \end{aligned} \tag{5.14}$$

$$Sum01 = a01b00cin0 + a00b01cin0 + a00b00cin1 + a01b01cin1 \tag{5.15}$$

$$Sum00 = a01b01cin0 + a01b00cin1 + a00b01cin1 + a00b00cin0 \tag{5.16}$$

The above equations can be used to facilitate adder implementations that correspond to either *local* or *global* weak-indication. The architecture of the *n*-bit ST RCA that comprises dual-bit adder modules is portrayed by figure 5.9.

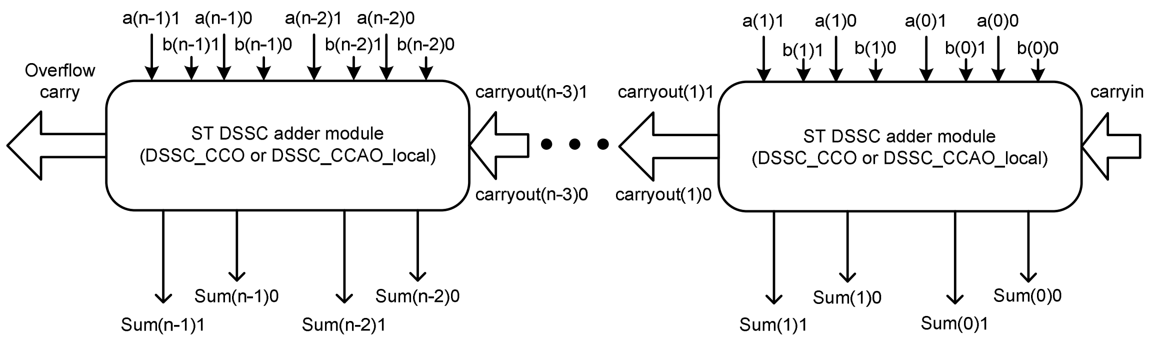


Figure 5.9: Dual-rail encoded *n*-bit carry-ripple adder architecture featuring local indication

5.2.1.1 Local Weak-Indication

The dual-bit adder realised using C-elements, complex gates and OR gates shall be referred to as the DSSC_CCO adder (with the acronym DSSC expanding as dual-sum, single-carry) and that employing C-elements, complex gates, AND gates and OR gates shall be called as the DSSC_CCAO_local adder. They are depicted through figures 5.10 and 5.11 respectively. In case of local indication, each and every dual-bit adder module is individually responsible for indicating the arrival of all its specific inputs (collective indication), in which case the system architecture would be governed by the configuration shown in figure 4.1.

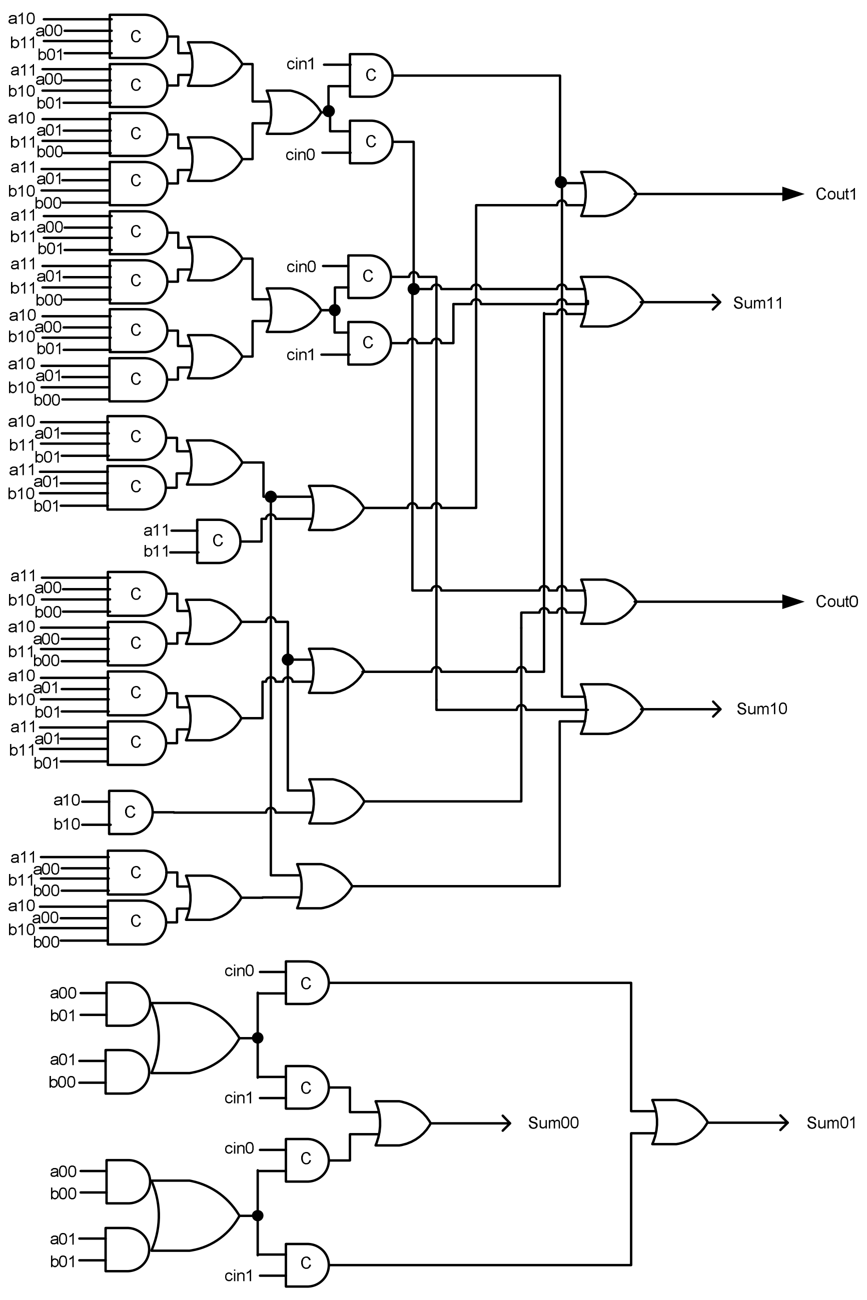


Figure 5.10: Weak-indication DSSC_CCO adder module

The DSSC_CCAO_local adder module is derived through the following modifications to the DSSC_CCO adder block, shown above.

- The input-complete gates in the first level are replaced by input-incomplete gates
- The above modification necessitates the inclusion of an additional synchronisation

circuitry for detecting completion (shown by dotted lines in figure 5.11) to ensure that weak-indication criteria are upheld collectively by the module's outputs.

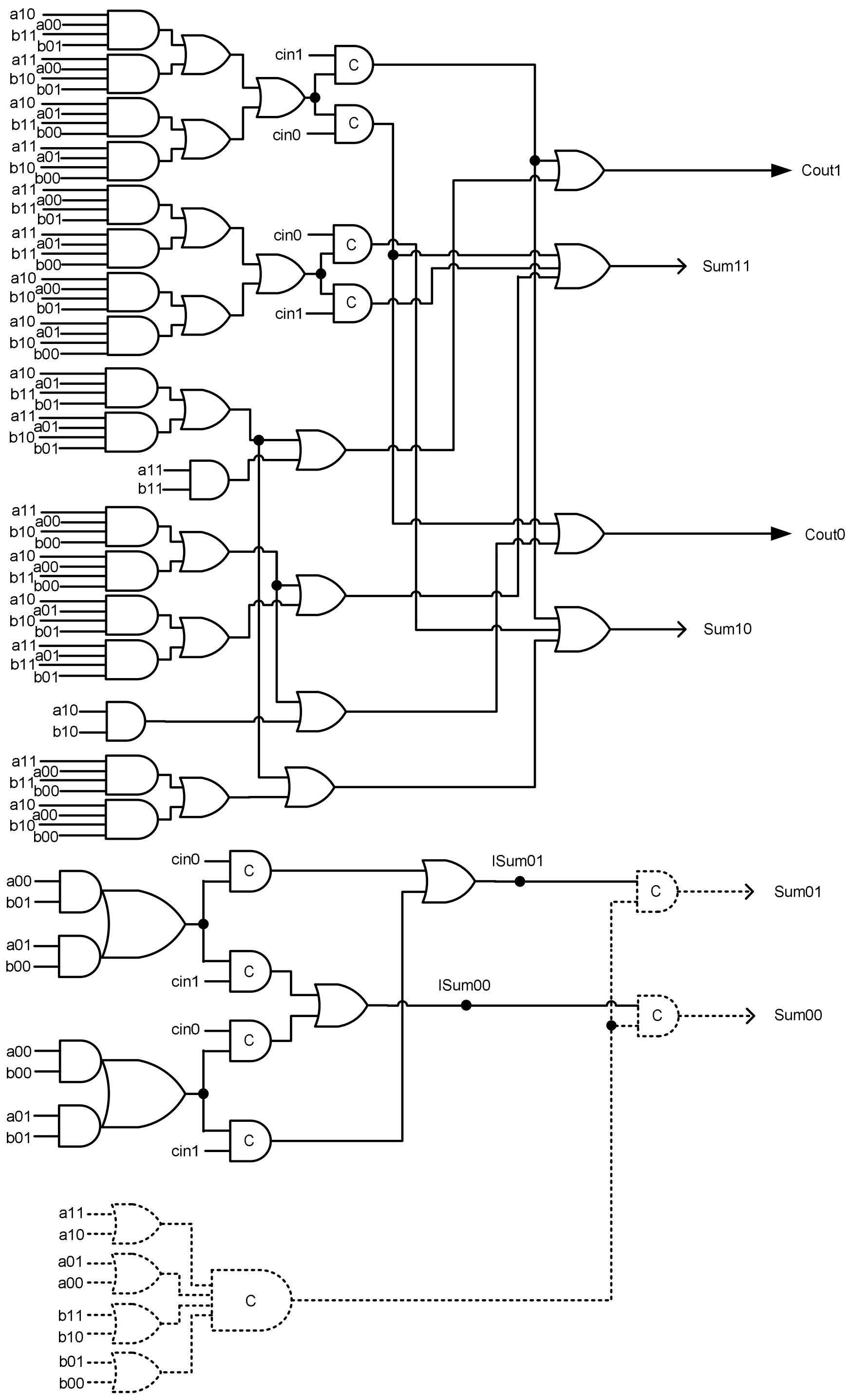


Figure 5.11: Weakly indicating DSSC_CCAO_local adder block

It can be seen from figure 5.11 that *ISum*01 and *ISum*00 are logically equivalent to *Sum*01 and *Sum*00. When certain augend/addend inputs and input carry become undefined, the carry output and most significant sum output can become undefined. However, only when all the inputs (addend/augend) become undefined, can the least significant sum output become undefined. The problem of wire orphans gets eliminated with the isochronic fork assumption imposed on the primary inputs of the adder block. It can be noticed that both DSSC_CCO and DSSC_CCAO_local adder modules are input-complete on an overall basis.

5.2.1.2 Global Weak-Indication

The dual-bit adder module used to construct a ST adder featuring global weak-indication is shown in figure 5.12, and shall be identified as the DSSC_CCAO_global adder. This module is basically a derivative of the DSSC_CCO adder block with the first level non-relaxed gates replaced by their relaxed equivalents. As such, it does not satisfy either strong or weak-indication specifications. It is early propagative in the sense that eager reset is possible but not eager evaluation. This is because the encoded sum and carry outputs would collectively indicate the transitions on the adder inputs including the input carry. In comparison with the DSSC_CCAO_local adder block shown in figure 5.11, the DSSC_CCAO_global adder does not consist of any local detectors and is not input-complete as well. Nevertheless, the adder module by itself is gate-orphan-free.

Input-completeness is guaranteed on the entirety by the *n*-bit ST RCA topology, specified by the system architecture shown in figure 4.4. In this case, the partial CD logic output would correspond to synchronisation of all the augend and addend adder inputs excluding the carry input, as a transition on any of the double rails of the input carry would be acknowledged by the sum outputs of the least significant dual-bit adder stage (probably by its corresponding carry output as well) thus facilitating multiple acknowledgement, which is welcome. Therefore, considering the overall system architecture, it can be said that with the exception of the least significant sum output (*Sum*01,*Sum*00), the rest of the sum outputs and the intermediate carry outputs are relaxed with respect to indication of the main adder operands, thereby paving the way for potential benefits in terms of delay, area and power.

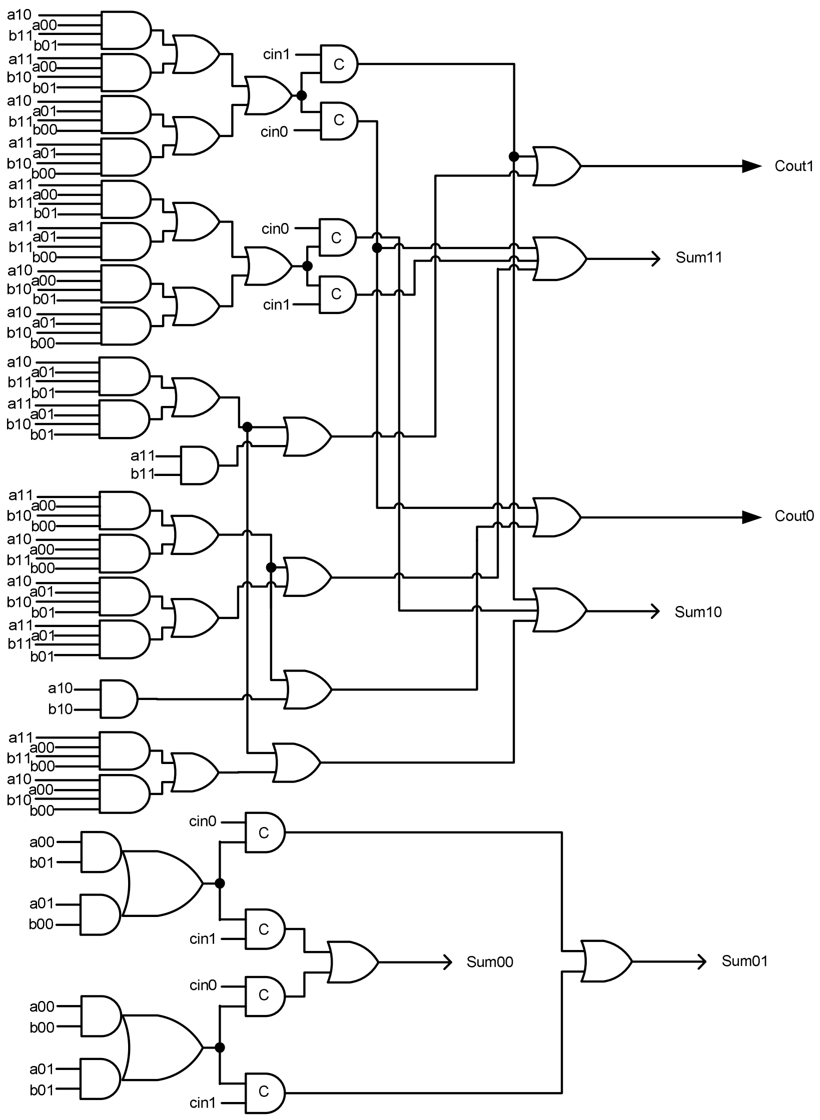


Figure 5.12: DSSC_CCAO_global adder module

5.2.1.3 Comparative Evaluation

A number of dual-bit adder modules based on different ST design methods were constructed and evaluated based on a 32-bit ST RCA architecture. Dual-bit adders based on Seitz's, DIMS and Toms' approaches were also considered for comparative evaluation. In fact, due to the limitation of the granularity of the base function set for mapping (maximum fan-in of AND gates and C-elements is 4), the logic corresponding to Seitz's and DIMS approaches required modification and SI decomposition respectively. The DIMS dual-bit adder implementation is based on the SI decomposition mechanism discussed in section 4.3.1, and Seitz's dual-bit adder block incorporates the concepts discussed in sections 4.3.1 and 4.3.2 respectively, i.e. the DIMS dual-bit adder features two-levels of C-gates, while Seitz's dual-bit adder has a first level composed of AND gates followed by a second level comprising C-elements to implement higher fan-in gates. However, weak-indication versions of these methods were considered, which could facilitate faster carry propagation to the successive stages. Table 5.4 reports the delay, area and power parameters of the various ST dual-bit adders, with input data sequences applied every 15ns for power estimation purpose.

Adder realisation style	Delay (ns)	Area (μm^2)	Power (μW)
Modified_Seitz_DSSC	12.8	16521	932.6
Decomposed_DIMS_DSSC	12.8	21833	1026.0
Toms_DSSC	9.4	10793	693.1
DSSC_CCO	5.9	14921	871.9
DSSC_CCAO_local	5.7	10041	839.1
DSSC_CCAO_global	**5.6**	**8833**	**648.3**

Table 5.4: Delay, area and power of various dual-rail dual-bit adder based 32-bit ST RCAs

To begin the analysis, let us first consider the case of Toms' adder. The delay of the 32-bit ST RCA utilising Toms_DRE single-bit adder is 10.6ns (from Table 5.1); in comparison, the delay of the 32-bit ST RCA utilising Toms_DSSC dual-bit adder blocks is 9.4ns. Though from a theoretical point of view, the latter value is significantly greater than half of the former value, it is mainly attributable to three important factors: many logic stages to be traversed to produce the least significant carry output signal (CE2 + OR2 + CE2 + OR2 + CE2 + 3OR2), more elements in the critical path (CE2 + 3OR2, as opposed to CE2 + 2OR2 for the single-bit adder) and increased loading due to an area increase by 1.43×, on average. Indeed, these three factors were found to be practically responsible for the degradation of the ideal delay metric in case of the entire dual-bit adder module based ST RCAs.

Among the three versions of the DSSC adders, the DSSC_CCAO_global adder exhibits reduced datapath delay, area and power, though the critical path comprises only a CE2 and OR2 for all the three adders – thanks to the indication being taken care of externally (globally) rather than confining it to the adder block. Compared to the DSSC_CCAO_global adder, the SSSC_DRE adder reports an increase in delay by 4% and decrease in area by 64%. However, in terms of total power, for the similar set of test patterns and simulation conditions, the SSSC_DRE adder is found to be more expensive than the DSSC_CCAO_global adder by 5%, dissipating 678.8μW. The power advantage for the latter mainly results from its system configuration, sharing the logic with the preceding CD stage.

5.2.2 Incorporating Heterogeneous Data Encoding

A dual-bit adder block based on heterogeneous encoding (HE) can represent the augend; addend inputs and sum outputs by a 1-of-4 code, while the input and output carry signals can be represented using a dual-rail code. With such an encoding mechanism, the minimised expressions for the function block outputs are given below. It is to be noted that the 1-of-4 code assignments for the augend inputs, addend inputs and the sum outputs are the reverse of the assignments given in Table 2.1. The *n*-bit ST RCA topology that encompasses heterogeneously encoded dual-bit adder modules is portrayed by figure 5.13.

$$Cout1 = a0b3cin1 + a1b2cin1 + a2b1cin1 + a3b0cin1 + a1b3 + a2b2 + a3b1 + a2b3 + a3b2 + a3b3 \qquad \textbf{(5.17)}$$

$$Cout0 = a0b3cin0 + a1b2cin0 + a2b1cin0 + a3b0cin0 + a0b0 + a0b1 + a0b2 + a1b0 + a1b1 + a2b0 \qquad \textbf{(5.18)}$$

$$Sum3 = a0b3cin0 + a1b2cin0 + a2b1cin0 + a3b0cin0 + a0b2cin1 + a1b1cin1 + a2b0cin1 + a3b3cin1 \qquad \textbf{(5.19)}$$

$$Sum2 = a0b2cin0 + a1b1cin0 + a2b0cin0 + a3b3cin0 + a0b1cin1 + a1b0cin1 + a2b3cin1 + a3b2cin1 \qquad \textbf{(5.20)}$$

$$Sum1 = a0b1cin0 + a1b0cin0 + a2b3cin0 + a3b2cin0 + a0b0cin1 + a1b3cin1 + a2b2cin1 + a3b1cin1 \qquad \textbf{(5.21)}$$

$$Sum0 = a0b0cin0 + a1b3cin0 + a2b2cin0 + a3b1cin0 + a0b3cin1 + a1b2cin1 + a2b1cin1 + a3b0cin1 \qquad \textbf{(5.22)}$$

The circuit realisation that synthesises (5.17) – (5.22) is depicted in figure 5.14. Basically, it considers utilisation of input-complete gates and shall be referred to as the DB_HE_local adder (DB – dual-bit). It could be seen that the module adheres to weak-indication timing constraints *locally*. The 1-of-4 encoded sum outputs strongly indicate the arrival of all the function block inputs, while the dual-rail encoded carry output can be relaxed with respect to ensuring completeness of inputs.

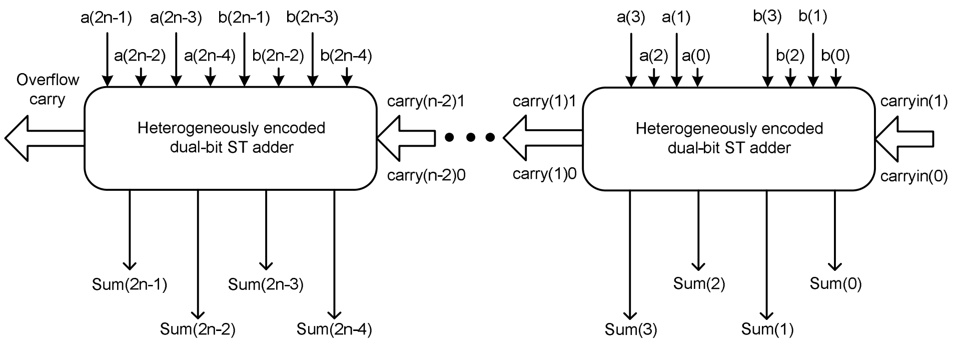


Figure 5.13: Heterogeneously encoded dual-bit adder block based *n*-bit ST RCA structure

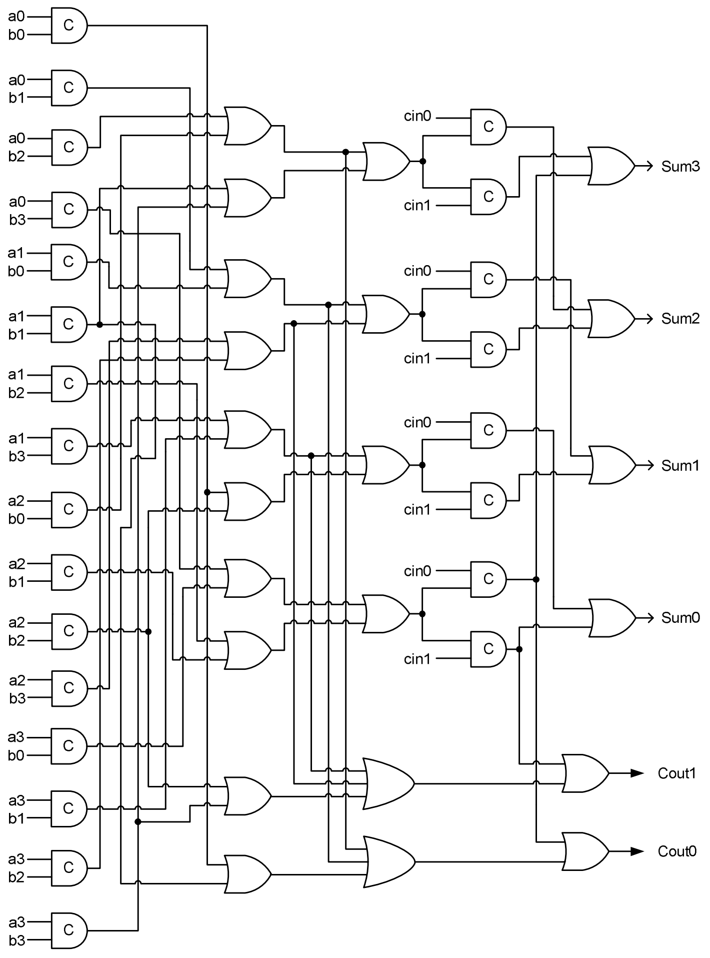


Figure 5.14: Weakly indicating heterogeneously encoded dual-bit adder module, corresponding to local indication

The ST system configuration that supports the RCA topology is illustrated in the diagram that follows. A subset of the dual-rail inputs (augends and addends) is 1-of-4 encoded before being fed to the function block while the remaining inputs (input carry) are fed as is. The non-dual-rail outputs produced by the logic block (sum outputs) are decoded before being passed onto the next stage, while the dual-rail outputs (output carry) are driven to the next stage as such. As mentioned earlier, the encoding cost is 28 transistors per bit. The cost of decoding is 12 transistors per bit.

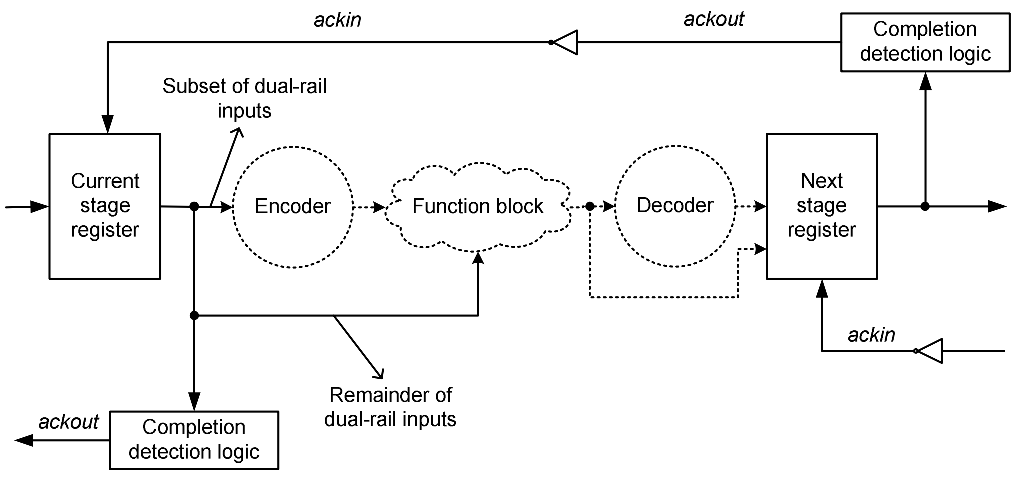

ackin
ackout
Completion detection logic
Subset of dual-rail inputs
Current stage register
Encoder
Function block
Decoder
Next stage register
ackin
Remainder of dual-rail inputs
ackout
Completion detection logic

Figure 5.15: ST system configuration handling heterogeneously encoded inputs and outputs

If the logically determined gates in the first level of figure 5.14 are replaced by non-logically determined gates, then proper local indication cannot be guaranteed by the above system architecture as the function block could be reset in an eager fashion but would evaluate robustly. Hereafter, it is identified as the DB_HE_global adder. Therefore, a modification is necessary, which is illustrated in figure 5.16, in the context of the dual-bit adder employing heterogeneous data encoding protocol. Basically, it serves as a replacement for the datapath represented using dotted lines in the above figure. In fact, not all the distinct outputs of the function block need to be synchronised with the logical sum of the homogeneously encoded input signals (OR-logic block output), but only an encoded DB adder sum output would suffice. Similar to the previous case, the least significant encoded dual-bit adder sum output (*Sum*3,*Sum*2,*Sum*1,*Sum*0) is alone synchronised with the output of the OR-logic block, while the other adder sum outputs (including the intermediate and output carries) could be relaxed. Thus, the overall system configuration would now correspond to global indication.

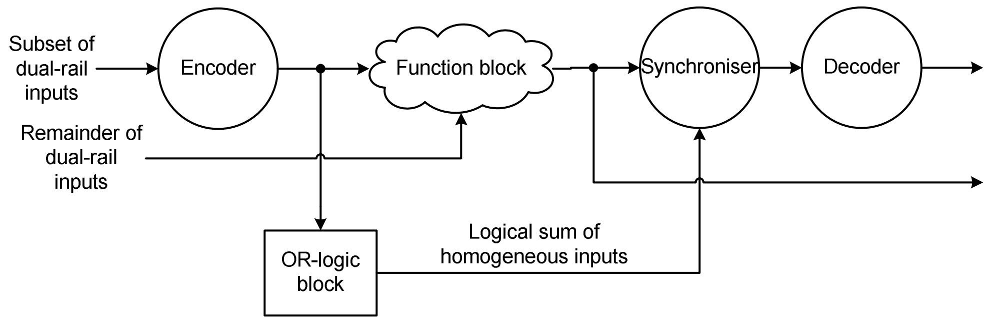

Subset of dual-rail inputs
Encoder
Function block
Synchroniser
Decoder
Remainder of dual-rail inputs
OR-logic block
Logical sum of homogeneous inputs

Figure 5.16: Modifications to the ST system configuration handling heterogeneously encoded inputs and outputs to pave the way for global indication

Table 5.5 below highlights the delay, area and power metrics of three different heterogeneously encoded dual-bit adder modules for ST addition of size 32 bits, based on the fundamental carry-propagate adder architecture shown in figure 5.13. The test bench and simulation conditions are the same as that of the earlier case.

Adder realisation style	Delay (ns)	Area (μm^2)	Power (µW)
Toms_DB_HE	9.0	12121	695.9
DB_HE_local	5.8	10889	688.4
DB_HE_global	**5.7**	**9594**	**685.5**

Table 5.5: Delay, area and power parameters of heterogeneously encoded 32-bit ST RCAs incorporating DB adders (with *extra logic*)

The critical path elements in case of the Toms_DB_HE adder are CE2 + 3OR2, while in case of the proposed adders CE2 + OR2 are the recurring elements in the longest signal path. This explains the reason for the higher datapath delay in case of the former. Due to the replacement of primary input-complete gates by input-incomplete gates, the DB_HE_global adder exhibits the least datapath delay. Also, it benefits from reduced loading and area occupancy due to the above, because a CE2 requires 1.4× more area than an AND2. From

Table 5.5, it is clear that the DB_HE_global adder dissipates the least average power and also has the least area requirement. However, when comparing the DB_HE_global adder with the DSSC_CCAO_global adder of Table 5.4, it could be inferred that the latter is preferable in terms of the design metrics estimated. Though this may be surprising, the main reason for this can be attributed to the *extra logic* (encoder, OR-logic block that performs completion detection, synchroniser and decoder) present in the datapath apart from the main functional logic in case of the former. To confirm this, simulations have been carried out assuming the presence of mixed protocol datapaths. The simulation results listed in Table 5.6 substantiate the above reasoning.

Adder realisation style	**Delay (ns)**	**Area (μm^2)**	**Power (μW)**
Toms_DB_HE	8.9	9978	457.3
DB_HE_local	5.6	8746	451.8
DB_HE_global	**5.4**	**7002**	**388.7**

Table 5.6: Delay, area and power parameters of 32-bit heterogeneously encoded ST RCAs (without *extra logic*)

5.3 Hybrid Adders

It was stated in section 5.2.1.3 that the least significant dual-bit adder stage actually incurred significant delay and this effect was observable in the entire dual-bit adder block based ST RCAs. Hence, a minor logic optimisation to mitigate this drawback has been resorted to. This involved optimal replacement of dual-bit adder module(s) in the least significant position(s) by single-bit adder block(s) so that the overall delay and area parameters could be reduced. Area would also get reduced as a typical dual-bit adder block consumes more area than a pair of single-bit adders, as shown in figure 5.17. ‘Series1’ highlighted in blue represents the normalised area measure of the dual-bit adder modules in comparison with the single-bit adder block, while ‘Series2’ highlighted in brown represents the normalised area measure of the dual-bit adder modules compared to a pair of single-bit adder blocks.

The carry-ripple adder structure would now feature a combination of single-bit and dual-bit adder modules resulting in the hybrid architecture. Since the dual-rail encoded dual-bit adder block based ST RCA is found to be more efficient than the heterogeneously encoded dual-bit adder module based ST RCA with respect to delay, area and power from Tables 5.4 and 5.5, the simulation results corresponding to analysis of the former are given in Table 5.7.

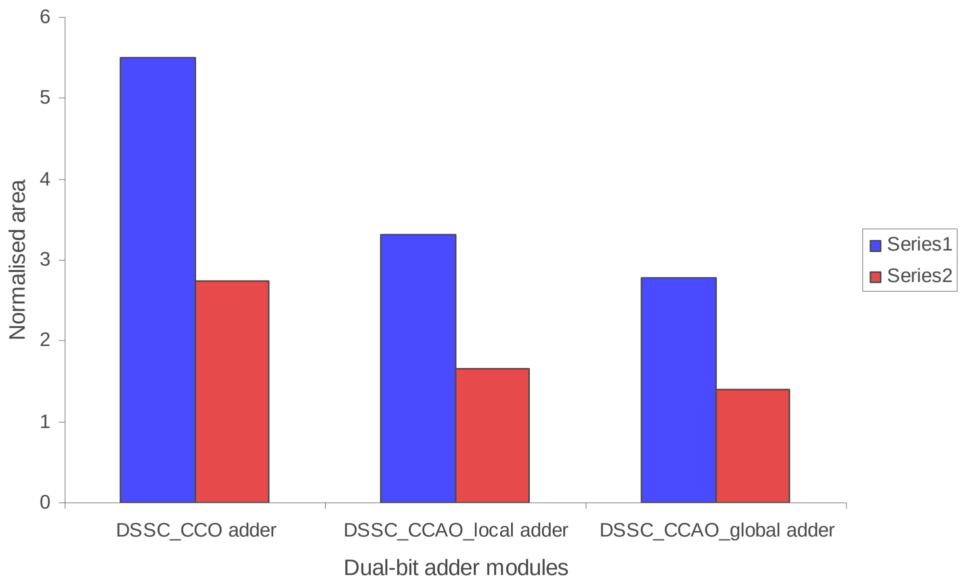


Figure 5.17: Highlighting the area expense of dual-bit adder blocks in comparison with single-bit adder modules

Adder realisation style		Delay (ns)	Area (μm^2)	Power (μW)
Dual-bit adders	DSSC_CCO	5.9	14921	871.9
	DSSC_CCAO_local	5.7	10041	839.1
	DSSC_CCAO_global	5.6	8834	**648.3**
Hybrid adders	Hybrid_DSSC_CCO	5.5	13941	847.6
	Hybrid_DSSC_CCAO_local	5.5	9856	828.8
	Hybrid_DSSC_CCAO_global	**5.4**	**8726**	650.2

CD logic for DSSC_CCAO_global and Hybrid_DSSC_CCAO_global are different from the rest of the adders

Table 5.7: Delay, area and power metrics of 32-bit dual-rail encoded hybrid ST RCAs

The Hybrid_DSSC_CCO adder is derived by substituting four stages of the SSSC_DRE adder in place of two least significant DSSC_CCO adder modules, while in case of Hybrid_DSSC_CCAO_local and Hybrid_DSSC_CCAO_global adders, two stages of SSSC_DRE adder modules in the least significant positions were found to be an optimal replacement. Any further inclusion of the single-bit adder block was only found to have a detrimental effect on the datapath delay. It can be seen from Table 5.7 that, in general, the hybrid ST RCAs exhibit slightly reduced delay, area and power over the dual-bit adder based RCAs. The conclusion is that the Hybrid_DSSC_CCAO_global adder reports only marginal reduction in delay and area parameters (3.5% and 1% respectively) compared to the DSSC_CCAO_global adder, and is associated with a negligible increase in total power. The logic optimisation method put forward in section 5.5 presents a rather efficient technique to decrease the adder latency.

5.4 Bottlenecks with Increase in Order

To evaluate whether an increase in the order of a fundamental adder block would be helpful, given the possibility of the hybrid architectural scheme, triple-bit ST adder designs were explored. A dual-rail encoded triple-sum, single-carry (TSSC) adder block consists of 14 inputs and 8 outputs. It is to be noted that, in comparison with the DSSC adder module, the input space would now be quadrupled, and as a result the area demand would also increase significantly. It was found that a TSSC adder block based on C-elements, complex gates and OR gates (TSSC_CCO adder) is larger compared to the individual DSSC_CCO and SSSC_DRE adder modules by 1.5× and 4.9× respectively – eventually this implies higher loading effect, more logic levels, greater delay and increased power consumption.

With respect to the output carry signal generated by an adder module, it was estimated through the principle of mathematical induction that the number of irredundant cubes in the MDSOP expression would be of O($2^{n+1}-1$), which approximates an exponential order, and consequently, this will have an adverse impact on the overall adder delay as the adder size increases. To confirm this, simulations were performed under similar conditions assuming operand sizes of 48-bits, and the results shown in Table 5.8 vindicate the above observation. 24 stages are required for implementing the dual-bit adder block based ST RCA, while 16 stages are required for realising the triple-bit adder module based ST RCA.

The Hybrid_DSSC_CCO adder comprises four stages of the SSSC_DRE adder while the Hybrid_TSSC_CCO adder incorporates six stages of the SSSC_DRE adder. The hybrid scheme has benefitted the latter in comparison with its true version by effecting an area decrease of 8.7% and enabling reduction in delay and total power dissipation by 20.5% and 12% respectively. However, the Hybrid_TSSC_CCO adder exhibits increased delay and area compared to the Hybrid_DSSC_CCO adder by approximately 6% and 49% respectively. It may be understandable that an area increase may not always result in a proportionate increase in average power in case of ST logic unlike the case quite common in synchronous designs, due to the activation of unique signal paths in the former. Therefore, the former dissipates only 12% more average power than the latter whilst featuring an area expense of 1.5×.

Adder realisation style	**Delay (ns)**	**Area (μm^2)**	**Power (μW)**
DSSC_CCO	9.1	22639	1399.6
TSSC_CCO	11.6	34849	1747.5
Hybrid_DSSC_CCO	**8.7**	**21389**	**1377.4**
Hybrid_TSSC_CCO	9.2	31819	1537.8

Table 5.8: Delay, area and power metrics of dual-bit and triple-bit adder based 48-bit ST RCAs

5.5 Redundant Logic Insertion

This section deals with an efficient method for reducing the datapath latency of an ST dual-bit adder based RCAs by means of a novel concept called *redundancy insertion*. In general, the concept can be extended to effect latency reduction in iterative logic circuits, which would comprise a cascade of basic building blocks. Redundancy insertion, in general, implies inclusion of extra redundant logic into the actual non-redundant implementation (which synthesises a specific functionality) with the intention of speeding up the propagation of certain signals, which would be required to drive (act as inputs for) the subsequent stages, without affecting the original functionality.

Redundancy can be incorporated into a function block implementation by careful duplication of similar logic and can be expected to pave the way for multiple acknowledgements, which may be useful in simplifying the timing assumptions in a ST realisation. Additionally, it could facilitate the faster reset of logic during the RTZ handshake protocol with a constant latency. Logic redundancy achieved through input-incomplete gates, basically introduces the weak-indication property into the circuit as it relaxes the indication (acknowledgement) constraints of those function block outputs that are considered as candidates for optimisation. It can either be implicit (for example, the SSSC_DRE adder) or explicit in the design and herein we consider explicit insertion of redundant logic with the objective of decreasing the critical path delay metric. The minor drawbacks of this approach are marginal increases in area and power parameters. Since logic duplication is involved, switching activity would increase due to multiple acknowledgements, subsequently pushing up the dynamic power and thereby increasing the average power dissipation. However, the area and power overheads may be insignificant depending upon the functionality and its initial non-redundant implementation, and the degree of logic redundancy subsequently resorted to. We will now consider six case studies to demonstrate the benefits of the redundancy insertion scheme based on the ST RCA architecture, where logic redundancy addresses the carry output since it is required to propagate between successive stages. Redundant logic insertion was performed manually with respect to all the ST adder circuits considered here.

5.5.1 Impact on a Single-Bit Adder Based on Hybrid Input Encoding

Let us first consider the ST full adder functionality based on HIE to explain how logic redundancy can be achieved through the insertion of input-incomplete gates.

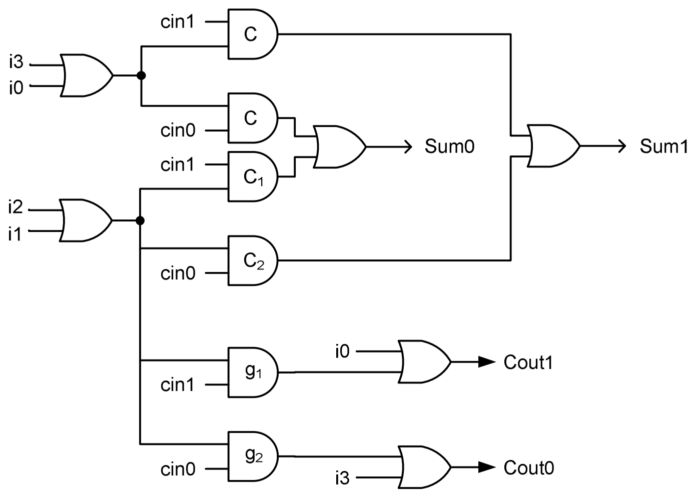


Figure 5.18: Hybrid input encoded ST full adder with logic redundancy

In the above figure, gates C_1 and C_2 denote 2-input C-elements, while gates g_1 and g_2 represent 2-input AND gates. It can be noticed in the diagram that the logic realised by C_1 and C_2 are equivalent to that of g_1 and g_2 respectively, for transitions. It can be seen in figure 5.6 that g_1 and g_2 are not present and hence redundancy is explicit in the present design, henceforth referred to as the SSSC_HIE_RL adder. This proves to be beneficial in two ways. During the spacer phase, all the sum outputs could be reset in a parallel fashion, as the dual-rail carry output of the kth stage could be reset based on its 1-of-4 encoded augend and addend inputs, and the dual-rail sum output of the (k+1)th stage would then depend only on the dual-rail carry input of its preceding stage. There is also a benefit in terms of improving computation speed during the valid data phase. This would be obvious by comparing the designs portrayed by figures 5.6 and 5.18; it can be observed that the carry propagation path delay is less in case of the SSSC_HIE_RL adder than the SSSC_HIE_NRL adder. This is further substantiated by the results shown below for the case of a 32-bit addition, based on the ST carry-ripple adder topology, with inputs fed every 15ns to the adder circuits.

Adder realisation style	**Delay (ns)**	**Area (μm^2)**	**Power (μW)**
SSSC_HIE_NRL	8.0	**6633**	**619.1**
SSSC_HIE_RL	**5.9**	6953	630.2

Table 5.9: Comparing delay, area and power parameters of redundant and non-redundant 32-bit hybrid input encoded ST RCAs

It is evident from the results tabulated above that the SSSC_HIE_RL adder achieves latency reduction over the SSSC_HIE_NRL adder by 26.3%, whilst reporting associated increases in area and average power parameters by 4.8% and 1.8% respectively.

5.5.2 Impact on Dual-Bit Adders Utilising Dual-Rail Encoding

We now analyse the effect of redundant logic insertion in a ST dual-bit adder module based on DRE. Figure 5.19 depicts the redundant gates (shaded AND gates) inserted into a typical DSSC_CCO adder module. The non-redundant adder block would not feature *rg*1 and *rg*2, and so one of the inputs for the OR2 gates producing *Cout*1 and *Cout*0 would be the outputs of C-elements (C_1 and C_2), which are nets *gn*2 and *gn*3 respectively that were wire forks earlier. But in the redundant version, the OR2 gates producing *Cout*1 and *Cout*0 consider *gn*1 and *gn*4 as inputs respectively. Again, for the case of transitions, *rg*1 and *rg*2 are functionally equivalent to C_1 and C_2.

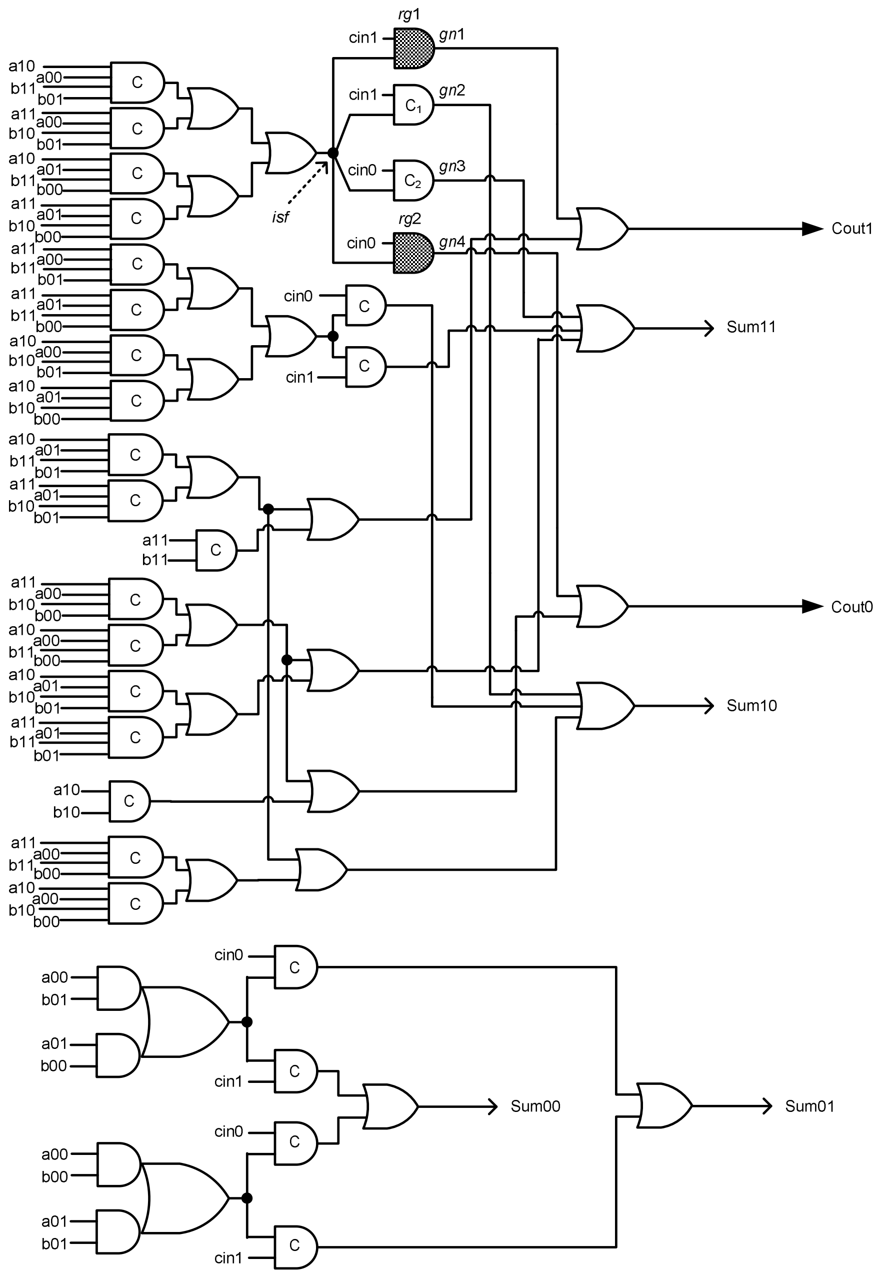


Figure 5.19: Showing redundant logic insertion in DSSC_CCO adder module

The gate output node labelled '*isf*' signifies an isochronic fork junction. Referring to figure 5.19, it can be observed that *isf*↑ would be followed by either *gn*2↑ or *gn*3↑ in case of the non-redundant DSSC_CCO adder block and by (*gn*1↑, *gn*2↑) or (*gn*3↑, *gn*4↑) in case of the DSSC_CCO adder module that incorporates logic redundancy – this signifies the possible multiple acknowledgements. It can also be noted that there becomes available a provision for fast or eager reset during the RTZ phase after insertion of logic redundancy. During the spacer phase, all the sum outputs could be reset in a parallel fashion, as the carry output of the previous adder's stage could be reset even by its corresponding augend and addend inputs without having to wait for an input carry from its preceding stage. The above condition would however be strictly enforced only in case of the DSSC_CCO adder as it mainly incorporates gates that are input complete, and it could be relaxed in case of DSSC_CCAO_local and DSSC_CCAO_global adder modules, since they widely employ gates that are input-incomplete. Nevertheless, in all the redundant logic adders, the sum output(s) of the $(i+1)^{th}$ adder stage could evaluate based on the carry input from its previous i^{th} stage and there does not arise the need for carry propagation over the entire length during the reset phase that might have occurred for the set phase. The advantage of latency reduction gained by introduction of redundant logic is attributable to the lower datapath delay encountered, as the critical path in every dual-bit adder stage comprises only input-incomplete gates instead of a mix of input-complete and input-incomplete gates in the original non-redundant version.

The results corresponding to the simulations of non-redundant and redundant logic dual-bit adders, performed under similar conditions, are given below. The increase in latency for the non-redundant logic adders corresponding to their redundant versions and the area and power overheads for the latter in relative comparison with the former are specified within brackets in Table 5.10. Non-redundant logic dual-bit adders are found to suffer from 29.3% delay increase compared to their redundant counterparts, on an average.

A visual inspection of Table 5.10 reveals that the area expense is very minimal for the adders incorporating logic redundancy, with virtually no increase in average power dissipation. The redundant logic included DSSC_CCAO_global adder based ST RCA was alone optimised further following the hybrid approach discussed earlier using two stages of the SSSC_DRE adder, as its non-redundant version was found to be efficient amongst all the dual-bit adders based on either DRE or HE.

Adder realisation style		Delay (ns)	Area (μm^2)	Power (μW)
Non-redundant logic	DSSC_CCO	5.9 (28.3%)	14921	871.9
	DSSC_CCAO_local	5.7 (29.5%)	10041	839.1
	DSSC_CCAO_global	5.6 (30.2%)	**8833**	**648.3**
Redundant logic	DSSC_CCO	4.6	15081 (1.1%)	875.3 (0.4%)
	DSSC_CCAO_local	4.4	10201 (1.6%)	842.5 (0.4%)
	DSSC_CCAO_global	**4.3**	8993 (1.8%)	651.7 (0.5%)

Table 5.10: Comparing delay and area metrics of redundant and non-redundant logic dual-bit adder based 32-bit ST RCAs employing DRE

It can be seen in Table 5.11 that there is practically no difference between the non-hybrid and hybrid versions in terms of power dissipation, while a marginal difference occurs with respect to delay and area.

Adder realisation style	Delay (ns)	Area (μm^2)	Power (μW)
DSSC_CCAO_global	4.3 (2.4%)	8993 (1.3%)	**651.7**
Hybrid_DSSC_CCAO_global	**4.2**	**8875**	653.3 (0.2%)

Table 5.11: Comparing delay, area and power of hybrid and non-hybrid DSSC_CCAO_global adder *incorporating redundant logic*, evaluated using a 32-bit ST RCA structure

5.5.3 Impact on Dual-Bit Adders Adopting Heterogeneous Encoding

Lastly, the impact of redundant logic insertion on heterogeneously encoded dual-bit adders is briefly analysed in this section. Logic redundancy, as introduced into a typical DB_HE_local dual-bit adder module, is portrayed in figure 5.20 with the input-incomplete gates marked by *rg*1 and *rg*2. Similar notations have been used as that of figure 5.19 so that the discussions of the previous section would hold well for this scenario too. The non-redundant DB_HE_local block was shown in figure 5.14. The latency increase suffered by the non-redundant adders compared to their redundant counterparts and the area and power expense of the latter in relative comparison with the former are given in Table 5.12 to arrive at a quick inference.

Adder realisation style		**Delay (ns)**	**Area (μm^2)**	**Power (μW)**
Non-redundant logic	DB_HE_local	5.8 (26.1%)	10889	688.4
	DB_HE_global	5.7 (29.5%)	**9594**	**685.5**
Redundant logic	DB_HE_local	4.6	11049 (1.5%)	691.9 (0.5%)
	DB_HE_global	**4.4**	9754 (1.7%)	688.9 (0.5%)

Table 5.12: Delay, area and power of NRL and RL dual-bit adder based 32-bit ST RCAs with HE

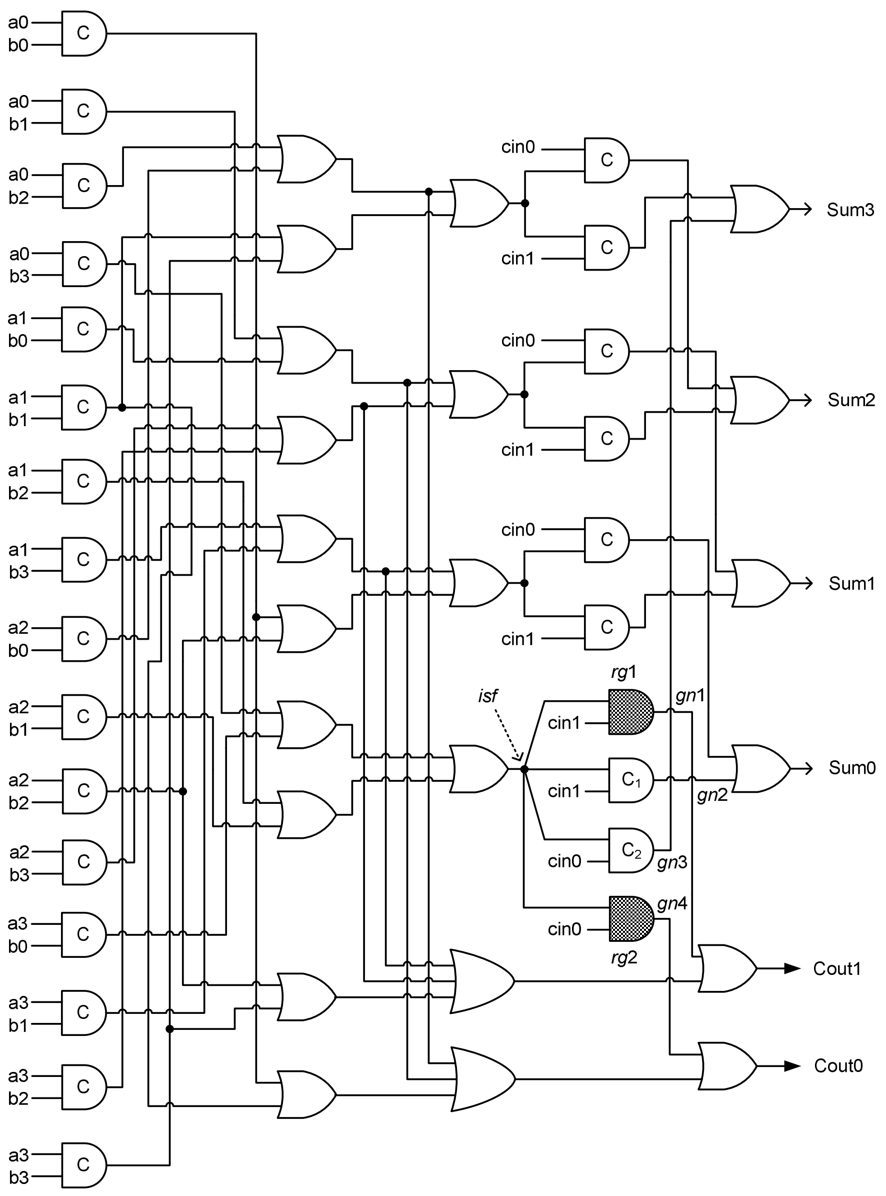


Figure 5.20: Highlighting redundant logic insertion in DB_HE_local adder module

From Table 5.12, it can be inferred that approximately 28% reduction in latency has been achieved by means of logic redundancy, which is significant, while noting that this is accompanied by only insignificant area and power penalties. As with the earlier case studies, the present example also underlines the appreciable benefit resulting from inclusion of redundant logic in a ST arithmetic circuit.

5.6 Summary

A plethora of ST adder blocks and logic optimisations targeting reduced latency have been analysed and evaluated in this chapter on the basis of the fundamental carry-ripple adder topology. The individual adder modules were configured to utilise either a homogeneous DI data encoding scheme for both the inputs and outputs, or a hybrid DI data encoding mechanism for the inputs and a homogeneous data encoding method for the outputs, or a heterogeneous DI data encoding protocol for both the inputs as well as outputs, depending upon the adder block size.

Adder type	Referencing
SSSC_DRE (Implicit redundant logic)	Add1
SSSC_HIE_NRL (Non-redundant logic)	Add2
DSSC_CCO (Non-redundant logic)	Add3
DSSC_CCAO_local (Non-redundant logic)	Add4
DSSC_CCAO_global (Non-redundant logic)	Add5
DB_HE_local (Non-redundant logic)	Add6
DB_HE_global (Non-redundant logic)	Add7
Hybrid_DSSC_CCO (Non-redundant logic)	Add8
Hybrid_DSSC_CCAO_local (Non-redundant logic)	Add9
Hybrid_DSSC_CCAO_global (Non-redundant logic)	Add10
SSSC_HIE_RL (Redundant logic)	Add11
DSSC_CCO (Redundant logic)	Add12
DSSC_CCAO_local (Redundant logic)	Add13
DSSC_CCAO_global (Redundant logic)	Add14
Hybrid_DSSC_CCAO_global (Redundant logic)	Add15
DB_HE_local (Redundant logic)	Add16
DB_HE_global (Redundant logic)	Add17

Table 5.13: Reference text for various ST adder blocks

Table 5.13 lists the referencing for the various proposed weak-indication ST adder blocks, whose design parameters corresponding to a 32-bit addition operation are portrayed in figure 5.21. The delay, power and area parameters of the different 32-bit ST RCAs are specified in 10^{-9}(s), 10^{-4}(W) and 10^{-9}(m^2) units respectively.

Overall, the various adder designs considered corroborate the fact that inclusion of logic redundancy leads to dramatic savings in terms of latency with only marginal area and power penalties, which is evident from figure 5.21. Amongst the different ST adders analysed, the hybrid DSSC_CCAO_global adder (Add15) featuring redundant logic is found to exhibit the *least* datapath delay for performing 32-bit addition. With respect to power and area, the SSSC_HIE_NRL adder (Add2) and Folco et al._DRE adder report optimum figures of 619µW and 6633µm^2 respectively. They report a similar latency of 8ns for 32-bit addition. In comparison, the hybrid DSSC_CCAO_global adder encounters less datapath delay by 48%, while the former adders dissipate less average power by 5.3% and demand less area by 25.3% compared to the latter.

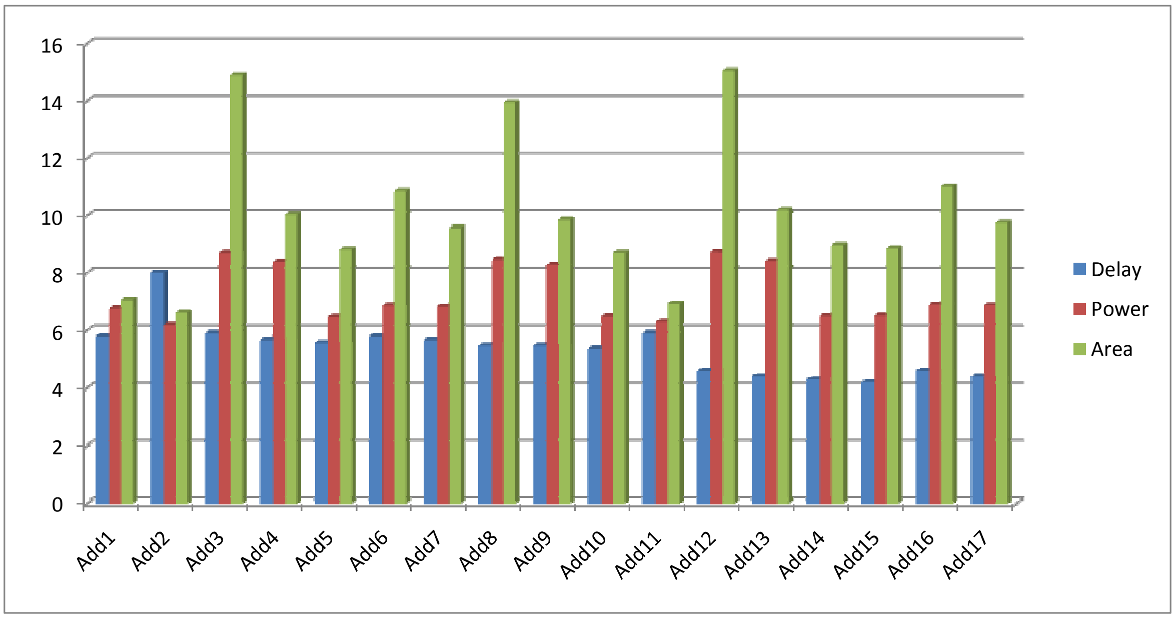


Figure 5.21: Depicting the delay, power and area metrics of various ST adder blocks for performing 32-bit addition based on the RCA topology

Chapter 6

Self-Timed Section Carry Based Carry-Lookahead Adders

RCAs were found to occupy the least area and dissipate less average/maximum energy per addition, next only to the Manchester carry chain adder, in relative comparison with many high-speed adder architectures of the synchronous domain [159]. Though the basic carry-propagate adder structure is relatively simple and easier to implement in accordance with the ST design style, it suffers from a linear increase in datapath delay proportional to the word width. It has been found that adder topologies such as carry-select [160] and conditional-sum logic [161] lend themselves to square-root time addition [162].

Carry-Lookahead (CLA) adders represent a widely used high-speed carry-propagate scheme for performing addition in logarithmic time [162], unlike the case with RCAs. In general, the design of a CLA adder is based on the principle that by examining the augend and addend bits, it is possible to predict/determine the carry signals beforehand and thereby reduce the delay that could otherwise be expected in a stage-by-stage propagation scenario. However, obtaining a ST derivative of the synchronous CLA architecture in a straightforward fashion is likely to give rise to gate orphans, mainly because of the propagate and generate signals that are generated for each adder stage, which are subsequently used for producing the CLA signal corresponding to a group of adder inputs. ST design procedures that rely upon the DRCL style may be helpful in permitting a direct translation of the synchronous architecture to robust asynchronous style, but can be expected to incur approximately thrice the area penalty in comparison with the former while accounting for the presence of explicit completion detectors as well. Nevertheless, our interest here is on facilitating a generalised gate level logic synthesis followed by an optimal SI decomposition with the intention of realising CLA modules that are inherently ST and compact. Hence, in this connection, ST section carry based CLA (SCBCLA) architectures are proposed in this chapter that bear some similarity with the traditional CLA scheme, and are evaluated on the basis of DRE and HIE protocols.

6.1 Background

We will now consider three scenarios to illustrate the problem of gate orphans inherent in recursive carry formulations. Let us first consider the basic equation governing the carry output signal, represented in single-rail format.

$$Cout = ab + (a \oplus b)\, cin \qquad \textbf{(6.1)}$$

$$\mathrm{G} = ab,\ \mathrm{P} = a \oplus b \qquad \textbf{(6.2)}$$

In (6.2), G and P signify *generate* and *propagate* signals. It can be interpreted from the above equations that an output carry is *generated* if both the operand bits are 1, while an incoming carry is *propagated* to the output if the operand bits are mutually exclusive. Hence, with notations P_i and G_i denoting the generate and propagate functions of a random adder stage i, we have,

$$\mathrm{C}_i = \mathrm{G}_i + \mathrm{P}_i\mathrm{C}_{i-1} \qquad \textbf{(6.3)}$$

It may be that (6.3) can be thought of as a second-order equation, since G_i and P_i can be further expressed in terms of the primary inputs of a generic adder stage. In general, there is a carry at stage i, if there is a carry-generated at stage i or, if there is a carry that is generated at stage i-1, which is propagated to stage i. This notion can be extended to *predict* the carry signal at any arbitrary adder stage and therefore the above equation is basically recursive in nature. Unwinding the recursion implicit in (6.3), for each stage, would yield the following i^{th} order equation, where C_{-1} represents the carry input to the least significant adder stage.

$$\mathrm{C}_i = \mathrm{G}_i + \mathrm{P}_i\mathrm{G}_{i-1} + \mathrm{P}_i\mathrm{P}_{i-1}\mathrm{G}_{i-2} + \ldots. + \mathrm{P}_i\mathrm{P}_{i-1}\mathrm{P}_{i-2}\ldots\mathrm{P}_0\mathrm{C}_{-1} \qquad \textbf{(6.4)}$$

Representing (6.3) in dual-rail encoded format, after obtaining dual of the positive-rail output and subsequent logic transformation to satisfy the cover constraint, we have,

$$\mathrm{C}_i^1 = \mathrm{G}_i^1 + \mathrm{G}_i^0\mathrm{P}_i^1\mathrm{C}_{i-1}^1 \qquad \textbf{(6.5)}$$

$$\mathrm{C}_i^0 = \mathrm{G}_i^0\mathrm{P}_i^0 + \mathrm{G}_i^0\mathrm{P}_i^1\mathrm{C}_{i-1}^0 \qquad \textbf{(6.6)}$$

It can be seen that C_i^1 and C_i^0 are expressed in their MOSOP forms. With respect to these second-order equations, gate-orphan freedom cannot be guaranteed even when the cubes are physically realised without any decomposition, in which case the inference would hold good for the recursive formulation of (6.4) as well. The possibility for the occurrence of gate and wire orphans is clarified through the following discussion. Figure 6.1 depicts the carry output (C_1) formulation using generate and propagate signals, as given by (6.5) and (6.6), where (a_1^1, a_1^0), (b_1^1, b_1^0) and (C_0^1, C_0^0) represent the dual-rail augend, addend and carry inputs of an adder stage and (C_1^1, C_1^0) specify the dual-rail carry output generated from this stage. In figure 6.1, *is*1 and *is*3 represent the true and false rail expressions of the carry-generate signal, while *is*4 and *is*2 correspond to the true and false rail expressions of the carry-propagate signal. For the condition when carry-generate function becomes valid, the following sequence of transitions occurs: ($a_1^1\uparrow$, $b_1^1\uparrow$) $\rightarrow$ *is*1↑ $\rightarrow$ $C_1^1\uparrow$. Even if the transition $C_0^1\uparrow$ occurs, since the intermediate output signals *is*3 and *is*4 did not fire, the transition at gate output node *is*2 (*is*1↑ $\rightarrow$ *is*2↑) is said to give rise to a gate orphan and $C_0^1\uparrow$ results in a wire orphan.

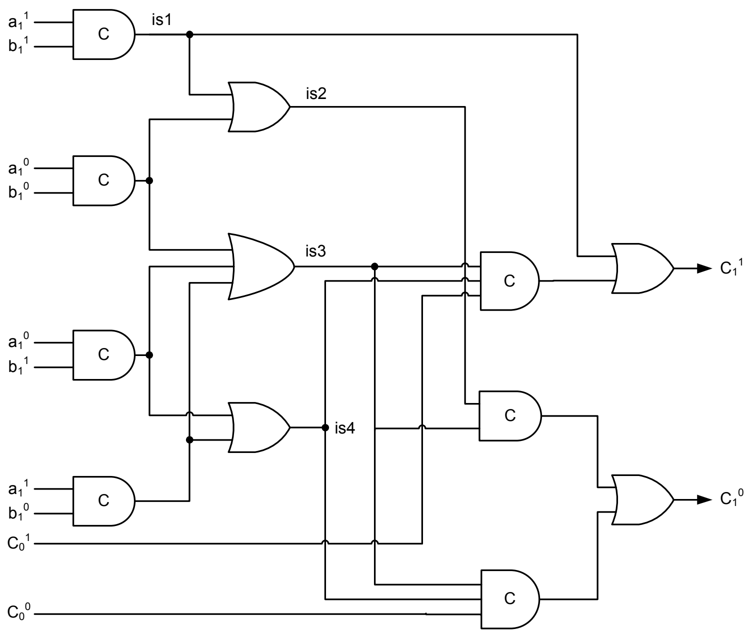


Figure 6.1: Carry output description using generate and propagate signals

Alternatively, the carry equations can also be represented taking into account the carry-kill condition, apart from generate and propagate conditions, which signifies the state when both the augend and addend inputs of an adder stage assume a logic low state. This state avoids the generation of a carry signal from this stage and also prevents the propagation of an input carry to the output.

$$C_i^1 = G_i^1 + G_i^0 P_i^1 C_{i-1}^1 \qquad \textbf{(6.7)}$$

$$C_i^0 = K_i^1 + K_i^0 P_i^1 C_{i-1}^0 \qquad \textbf{(6.8)}$$

The logic realising the above equations, taking account of carry-generate, propagate and kill conditions, is depicted by figure 6.2. In figure 6.2, *im*1 and *im*4 represent the true and false rail expressions of the carry-generate signal, while *im*5 and *im*2 correspond to the true and false rail expressions of the carry-kill signal. The equation pertaining to the carry-propagate condition is realised by the intermediate node *im*6. Let us consider a worst-case scenario to describe how orphans occur in the circuit. It can be seen from figure 6.2 that (a_1^1↑, b_1^1↑) → *im*1↑ → C_1^1↑. The following also occurs: *im*1↑ → *im*2↑. But the transition on the intermediate gate output *im*2 will not be subsequently acknowledged by the output *im*3, for a transition on the carry input C_0^1. This leads to the creation of gate and wire orphans.

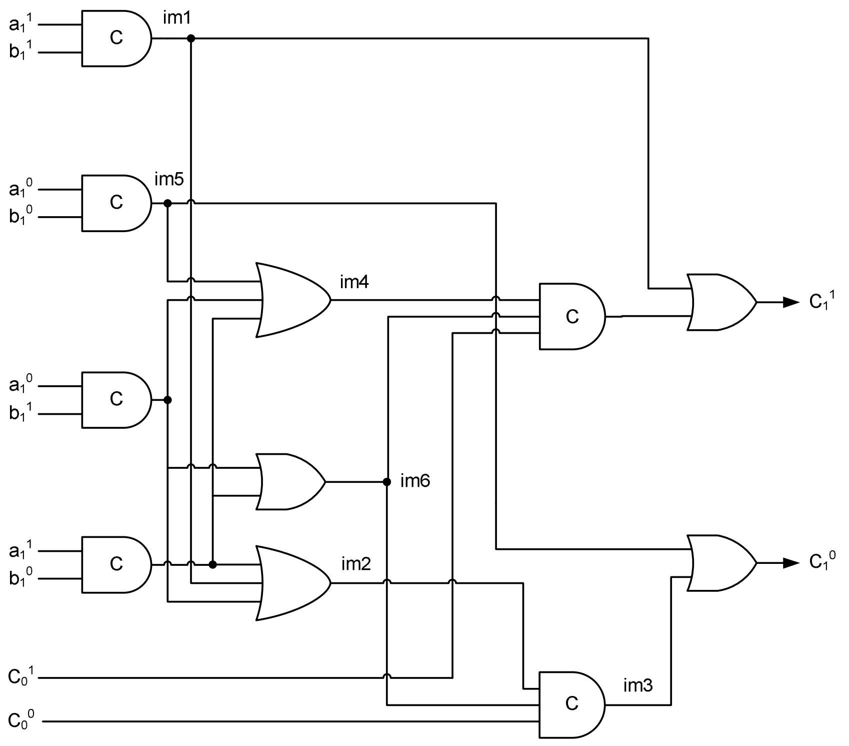


Figure 6.2: Output carry representation on the basis of generate, propagate and kill functions

Lastly, we analyse the situation when the carry-propagate signal is expressed in its simplified version as P = $a + b$, when the output carry is given as $Cout = ab + (a + b)cin$. This result in the dual-rail carry output to be expressed in its MSOP form as,

$$Cout1 = a1b1 + a1cin1 + b1cin1 \quad \textbf{(6.9)}$$

$$Cout0 = a0b0 + a0cin0 + b0cin0 \quad \textbf{(6.10)}$$

Based on the recursive formulation of (6.5) and (6.6), the carry-lookahead function is then implemented as shown in figure 6.3. The true and false rail expressions of the carry-generate signal are specified by *ig*1 and *ig*5, while the true and false rail expressions of the carry-propagate condition are signified by *ig*2 and *ig*6 respectively in the figure below.

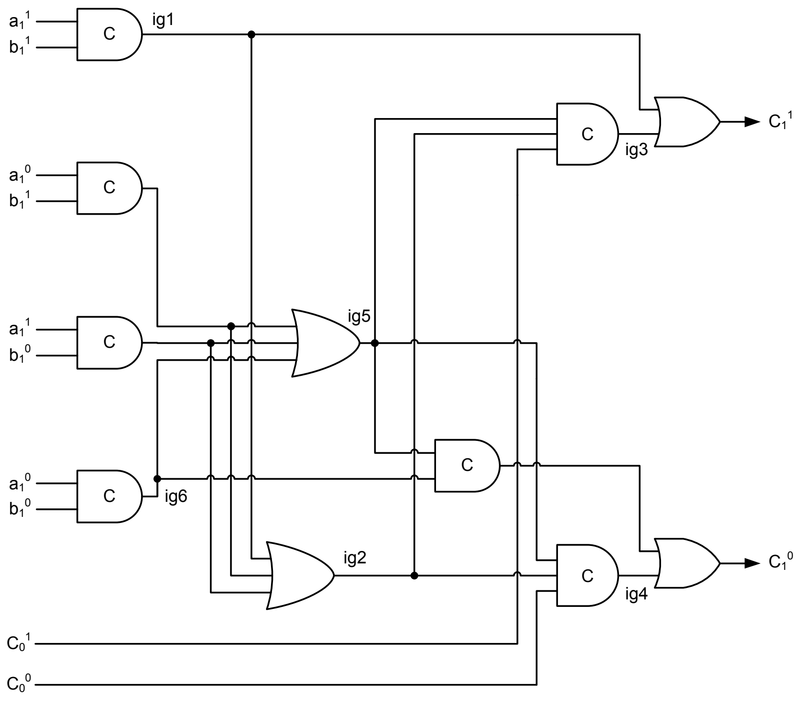


Figure 6.3: Carry output representation assuming P = *a*+*b*

Here, whenever the adder inputs are different the circuit would be gate orphan free and this would be guaranteed even when the carry-kill condition occurs. However, when the carry-generate condition becomes valid, *ig*1↑ → C_1^1↑. But *ig*1↑ → *ig*2↑ and the transition on *ig*2 would not be acknowledged by *ig*3 or *ig*4, which gives rise to a gate orphan. At this juncture, the transition on any of the input carry rails would be classified as a wire orphan.

Thus, the preceding discussions have demonstrated the problematic issue of gate orphans and/or wire orphans, possible in case of a recursive asynchronous carry synthesis. However, the problem with recursive equations can be overcome if the essential logic transformations to satisfy the cover constraint are performed on a first-order equation, and this would therefore necessitate reduction of a k^{th} order carry equation to the first-order that would actually involve unfolding cube expansions. As a consequence, the need for stage-wise propagate and generate signals is deemed unnecessary.

A better way to achieve this is to invoke the MOSOP heuristic for an initial two-level MSOP form of the carry equation, and for physical implementation, SI decomposition can then be resorted to since suitable candidates could be ascertained from within the MOSOP form – this is owing to the carry output function exhibiting symmetricity with respect to permutation of its literals. It has been deduced through the principle of mathematical induction that the MOSOP expression for the carry output of a q-bit CLA would consist of $(2^{q+1}-1)$ logical conjunctions with the support set of the cube of maximum dimension comprising $(2q+1)$ literals. Owing to the existence of an exponential relationship between the CLA size and the resulting number of product terms, CLA sizes in case of a ST implementation would have to be restricted so as to gain the maximum benefit in terms of reduced latency using this topology. Nevertheless, it has been intuitively observed that CLA logic of any size would be practically feasible through SI logic decomposition.

6.2 Section Carry Based CLA Architectures

Two CLA adder architectures have been conceived, bearing in mind the spatial demand of a robust asynchronous implementation and they are discussed in this section.

6.2.1 Type 1 Architecture – Fundamental Topology

The Type 1 architecture bears some similarity with a block CLA adder featuring intra-group carry ripple [161], which is the structure of a typical CLA adder. However, it mainly differs in that propagate and generate signals corresponding to each single-bit adder stage need not be computed – hence the term ‘section carry’. Figure 6.4 depicts the Type 1 architecture of the proposed section carry based CLA adder adopting DRE.

The q-bit CLA module generates a CLA signal corresponding to a section/group of q-bits of the adder operands. To this end, it accepts a carry input from a previous stage/section. The CLA signal corresponding to a section is used to feed the subsequent CLA unit in the cascade and also the next adder element in the sequence. Thus, the sum outputs of the adder

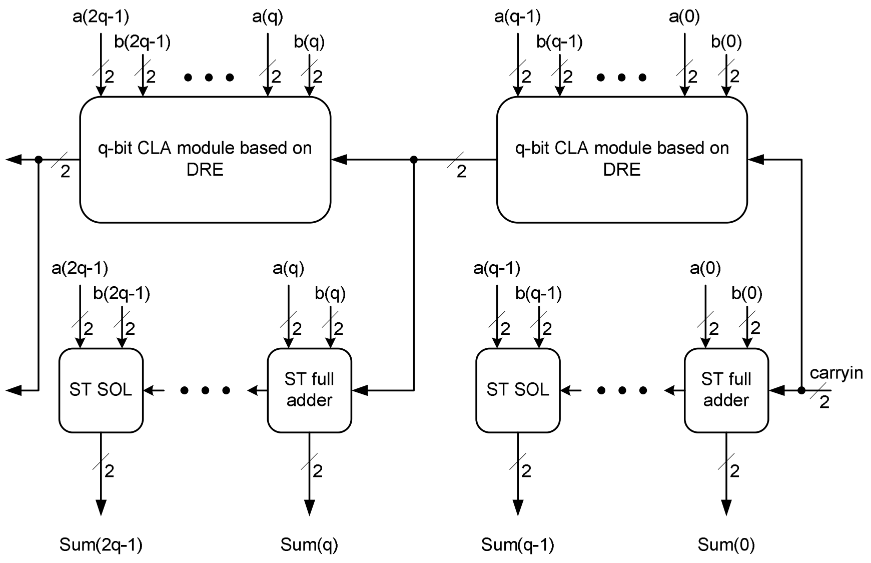


Figure 6.4: Type 1 ST section carry based CLA adder architecture based on DRE

can be produced simply by a rippling of the carry signal within each section, while the carry output of a section can be produced simultaneously and be quickly passed onto the next section to generate the lookahead signal of that stage. As a result, there arises an opportunity for optimising the CLA logic at the expense of the sum producing logic, i.e. the sum outputs of a section can assume the collective responsibility of indicating all the input operands of that section, while the CLA unit corresponding to a section can be freed from adhering to indication constraints, permitting it to be early propagative whilst ensuring that the realisation is free from the problem of gate orphans.

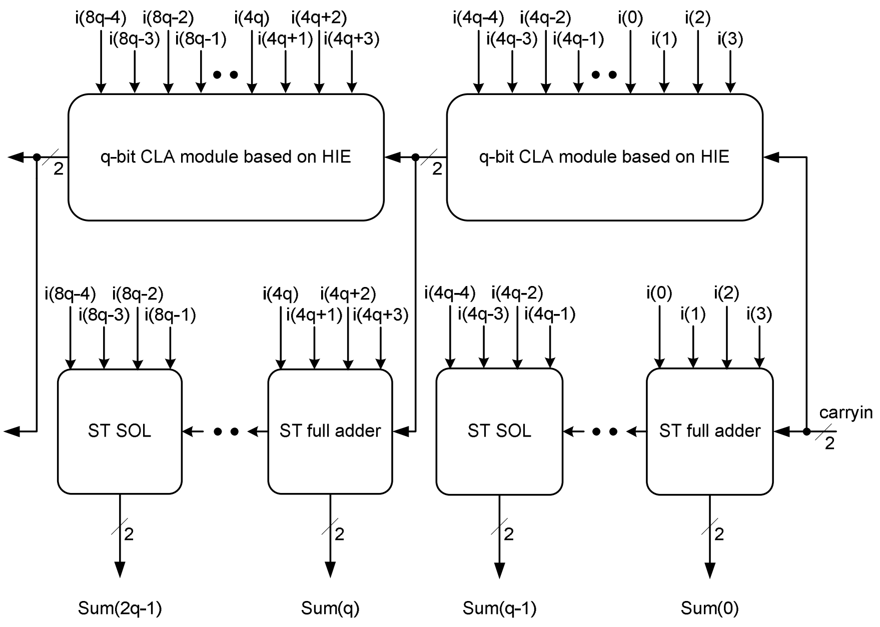

i(8q-4) i(8q-2) i(8q-3) i(8q-1) i(4q) i(4q+2) i(4q+1) i(4q+3)
i(4q-4) i(4q-2) i(4q-3) i(4q-1) i(0) i(2) i(1) i(3)
q-bit CLA module based on HIE
q-bit CLA module based on HIE
2
2
ST SOL
ST full adder
ST SOL
ST full adder
carryin
2
Sum(2q-1)
Sum(q)
Sum(q-1)
Sum(0)

Figure 6.5: Type 1 ST section carry based CLA adder architecture based on HIE

The acronym 'SOL' stands for *sum only logic*, which accepts an augend, addend and carry input and processes them to produce a sum output. In fact, the SOL unit can be derived from the basic full adder module. Hence, according to the Type 1 topology, for an *n*-bit adder comprising *q*-bit CLA units, $\left(\frac{n}{q}-1\right)$ CLA modules would be required as CLA signal generation is necessitated only till the penultimate section. The Type 1 CLA adder structure for a hybrid input encoded datapath is shown in figure 6.5, which features a combination of dual-rail and 1-of-4 codes.

6.2.2 Type 2 Architecture – Topology with Least Significant RCA Section

From a physical implementation perspective, it can be anticipated that substantial delay would be encountered in the least significant CLA section, as opposed to the successive CLA sections in case of the Type 1 architecture. This was indeed observed during simulations, where the critical path in case of the least significant dual-rail encoded CLA unit consists of AND4, CE2, 3OR2 gates (5CE2, 5OR2), while the critical path in the least significant hybrid input encoded CLA block comprises AO2222, CE2, OR2 gates (AND4, CE2, 2OR3, 2OR2)

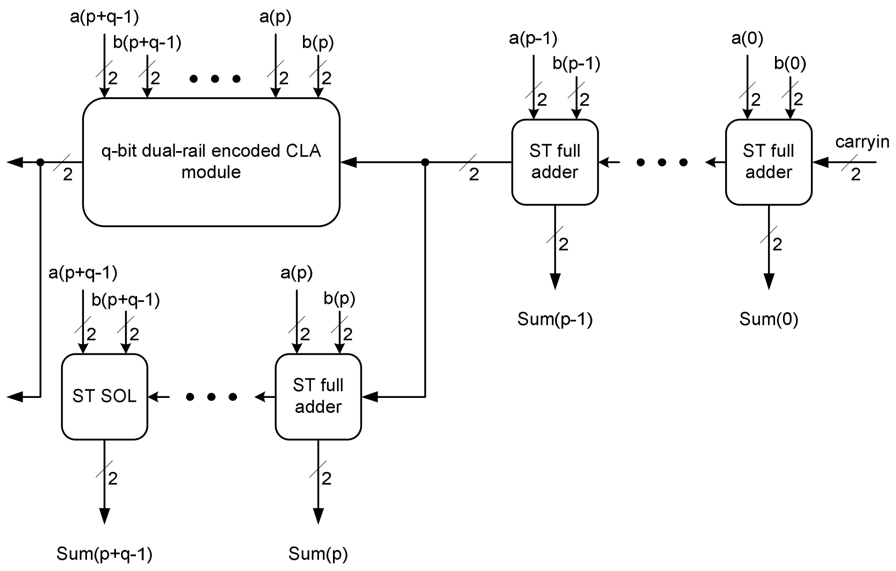


Figure 6.6: Type 2 ST section carry based CLA adder topology based on DRE

for two-bit (four-bit) lookahead carry signal generation, based on the proposed design. Since the latency of the least significant CLA module (two or four bits) was found to be higher than what could be expected from a series cascade of individual adder sections, the least significant CLA adder section can preferably be replaced by a simple carry-propagate adder section paving way for marginal reduction in terms of delay, area and power parameters. With this modification, the structure of the Type 2 CLA adder architecture would be as shown in figure 6.6, for the case of dual-rail encoded datapaths. Hence, the Type 2 topology is basically a refinement of the Type 1 structure primarily targeting delay optimisation. As a result, the number of CLA units required, would in general be specified by $\left(\frac{n-p}{q}-1\right)$, where n, p and q are assumed to be even. Here, p signifies the number of full adder stages present in the least significant positions of the adder architecture. The Type 2 topology for hybrid input encoded datapaths is portrayed by figure 6.7.

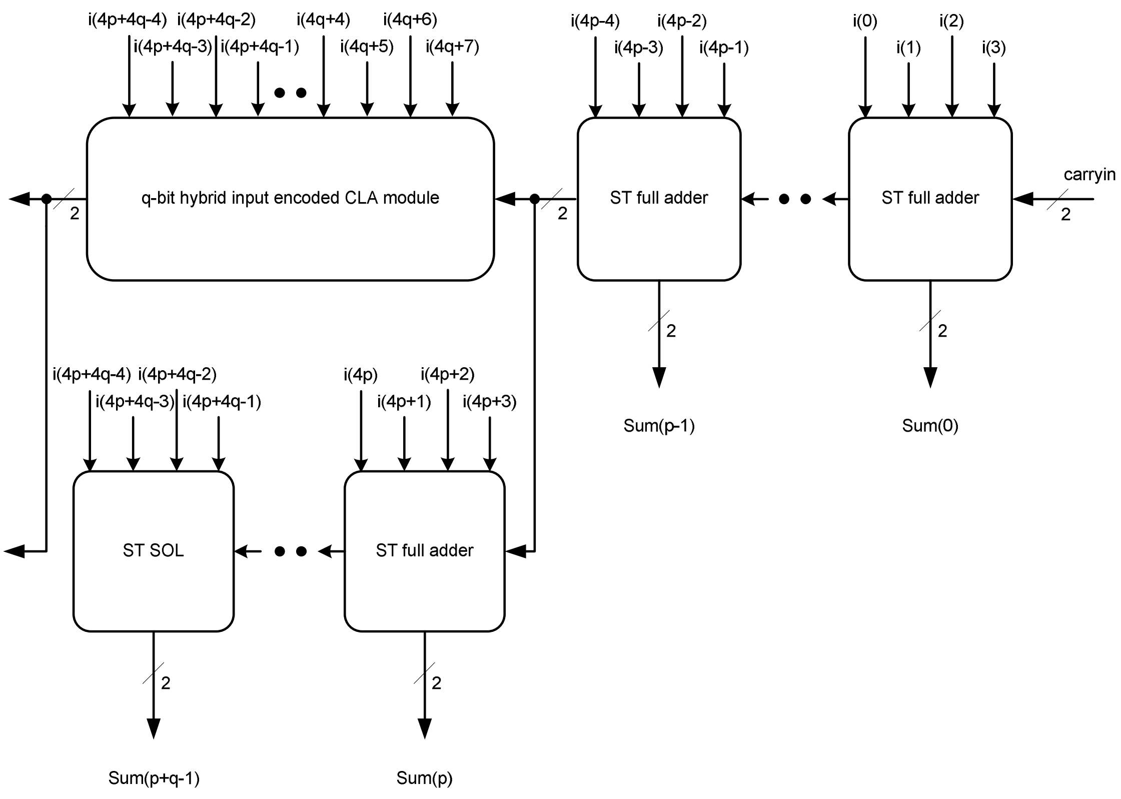


Figure 6.7: Type 2 ST section carry based CLA adder topology based on HIE

6.3 Evaluation with Two-Bit CLA Generator

Simulations have been performed with a two-bit CLA generator module addressing both the Type 1 and Type 2 architectures, on the basis of a variety of approaches. Table 6.1 shows the delay, area and power metrics corresponding to Type 1 topology, while Table 6.2 lists the same for the Type 2 architecture.

Adder realisation style	Delay (ns)	Area (μm^2)	Power (μW)
Seitz_DRE	13.7	17229	1168.9
DIMS_DRE	14.9	25245	1233.8
Toms_DRE	10.2	14191	927.4
SSSC_DRE	5.5	9016	755.0
Toms_HIE	7.2	12331	807.1
SSSC_HIE_NRL	**5.1**	**7593**	**659.4**
SSSC_HIE_RL	5.2	7763	663.2

Table 6.1: Delay, area and power for 32-bit ST addition based on Type 1 topology with 2-bit CLA

The SSSC_HIE_NRL adder is found to exhibit the least latency, area and power parameters among the Type 1 architecture adders. Though the SOL and two-bit CLA modules for the SSSC_HIE_NRL adder and SSSC_HIE_RL adder are the same, the difference between the two is with respect to the single-bit adder synthesised, i.e. in comparison with the latter, the former occupies 0.89× area. This could eventually have had an impact on the delay metric as well; less power, due to absence of redundant gates. In comparison with dual-rail encoded adders, hybrid input encoded adders are preferable as they facilitate relatively compact circuit realisations. Figure 6.8 highlights the size of the two-bit ST CLA logic corresponding to various methods. It can be seen that the proposed weak-indication hybrid input encoded 2-bit CLA unit is an order of magnitude smaller than the dual-rail encoded 2-bit CLA modules and even in comparison with the strongly indicating 2-bit CLA unit based on HIE – thanks to a significant shrinkage of the input space.

It should be noted that for the case of DRE, Type 2 architecture generally gives relatively less datapath delay than the Type 1 architecture. This is primarily attributable to the elements found in the critical path of the dual-rail encoded 2-bit CLA modules. For example, in case of the proposed adder, the critical path of the dual-rail encoded 2-bit CLA unit consists of AND4, CE2 and 3OR2 gates, while in case of the RCA section of size 2, the carry signal would only encounter the propagation delay associated with two AO222 cells, and hence

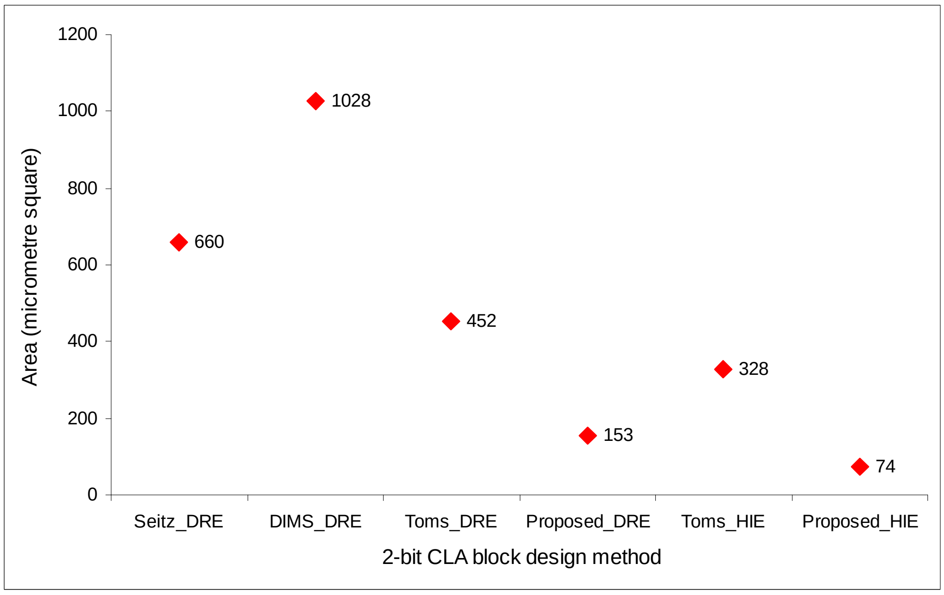


Figure 6.8: Relative comparison of area occupancy of two-bit CLA module designs

Adder realisation style	Delay (ns)	Area (μm^2)	Power (μW)
Seitz_DRE	13.3	16593	1145.7
DIMS_DRE	14.8	24273	1208.8
Toms_DRE	10.1	13479	906.1
SSSC_DRE	5.3	8887	749.7
Toms_HIE	7.1	12013	794.0
SSSC_HIE_NRL	5.2	**7529**	**656.8**
SSSC_HIE_RL	**5.2**	7709	660.9

Table 6.2: Delay, area and power parameters of 32-bit ST addition based on Type 2 adder architecture with two-bit SCBCLA

comparatively lower delay. In case of the Type 2 architecture, the SSSC_HIE_RL adder can now be expected to feature the least delay due to the presence of a RCA section in the least significant adder stages. It has already been discussed in section 5.5.1 how the SSSC_HIE_RL adder achieves latency reduction over the SSSC_HIE_NRL adder at the expense of more silicon (refer Table 5.9). But, on an overall basis, it is found that the Type 1 architecture incorporating the SSSC_HIE_NRL adder exhibits the least datapath delay. This is mainly because, with respect to the SSSC_HIE_RL adder incorporated into the Type 2 configuration, the critical path elements corresponding to the RCA section would be OR2, 2AND2 and 2OR2 gates, whereas in case of the SSSC_HIE_NRL adder, they would correspond to OR2, 2CE2 and 2OR2 gates. However, in case of the Type 1 configuration, the SSSC_HIE_NRL adder would only experience the propagation delay associated with the datapath comprising AO2222, CE2 and OR2 cells, thereby resulting in a reduced latency. By comparing Tables 6.1 and 6.2, it can be seen that the SSSC_HIE_NRL adder belonging to the Type 2 architecture dissipates the least average power, mainly due to the absence of a CLA module in the least significant stage. The test vectors were fed to the adder blocks every 15ns for the simulations.

6.4 Evaluation with 4-Bit CLA Generator

This section investigates extending the levels of lookahead from 2-bits to 4-bits. Simulations have been performed with a 4-bit ST CLA structure employing DRE and HIE protocols. In case of Seitz's, DIMS or Toms' approaches, for the case of DRE, the input space enumeration would be of $O(2^9)$ and therefore the resulting CLA logic would be massive. Due to this reason, such realisations are not considered here, as they are not likely to be of benefit in terms of optimising the delay parameter owing to an associated increase in the number of logic levels. This observation is further reinforced by the inference, which can be derived from the values given in Table 6.3. The delay, area and average power values mentioned within brackets in Table 6.3 refer to that of the ST RCA architecture for addition of size 32 bits, as discussed in the previous Chapter.

Adder realisation style	**Delay (ns)**	**Area (μm^2)**	**Power (μW)**
Seitz_DRE	13.3 (6.5)	16593 (7689)	1145.7 (741.8)
DIMS_DRE	14.8 (12.8)	24273 (10665)	1208.8 (770.5)
Toms_DRE	10.1 (10.6)	13479 (7561)	906.1 (627.6)

Table 6.3: Delay, area and power for 32-bit ST addition based on Type 2 adder architecture with two-bit CLA and RCA topology corresponding to Seitz, DIMS and Toms methods

Seitz's and DIMS approaches suffer a delay penalty of approximately 105% and 16% respectively, while in case of Toms' procedure there is a decrease in latency by 4.7%, for the section carry based Type 2 CLA topology in comparison with the basic carry-ripple addition scheme. Nevertheless, in case of the latter, the delay reduction is at the expense of an increase in average power by 44.4% and area requirement by 78.3%. In case of Seitz's and DIMS approaches, delay degradation is accompanied by increased area and power metrics as well, which can be seen in Table 6.3.

The results obtained for a 4-bit ST CLA structure based on the proposed approach relating to Type 1, Type 2 and Hybrid architectures are listed in Table 6.4. The hybrid configuration involves supplementing the three single-bit adder stages (preceding the most significant full adder block) in the most significant nibble position of a section carry based CLA adder (Type 2 in case of DRE and Type 1 in case of HIE) by a 3-bit CLA module in order to effect a marginal reduction in delay, and only constitutes a peephole optimisation that is accompanied by associated area and power overheads. With respect to DRE, the 3-bit and 4-bit CLA units are 1.6× and 2.7× bigger compared to the 2-bit CLA module; in case of HIE, the 3-bit and 4-bit CLA units are 3.1× and 7.8× bigger in relative comparison with the 2-bit lookahead version.

Adder realisation style	**Delay (ns)**	**Area (μm^2)**	**Power (μW)**
SSSC_DRE – Type 1	4.3	9769	770.6
SSSC_DRE – Type 2	4.0	**9385**	757.3
SSSC_DRE – Hybrid	3.8	9601	765.5
SSSC_HIE_NRL – Type 1	3.8	10609	681.6
SSSC_HIE_NRL – Type 2	4.1	10041	**672.8**
SSSC_HIE_NRL – Hybrid	**3.6**	10829	687.3
SSSC_HIE_RL – Type 1	3.7	10859	689.4
SSSC_HIE_RL – Type 2	3.7	10301	680.8
SSSC_HIE_RL – Hybrid	**3.6**	11069	694.5

Table 6.4: Delay, area and power metrics for 32-bit ST addition based on Type1, Type 2 and Hybrid adder architectures with four-bit SCBCLA logic

The 4-bit ST CLA logic based hybrid configuration based on HIE incorporating the SSSC_HIE_RL adder, that corresponds to the Type 1 architecture, was found to exhibit the least delay, while the Type 2 architecture employing the SSSC_DRE adder was found to occupy the least area. As can be seen from Table 6.4, the Type 2 architecture featuring the SSSC_HIE_NRL adder was found to dissipate the least average power.

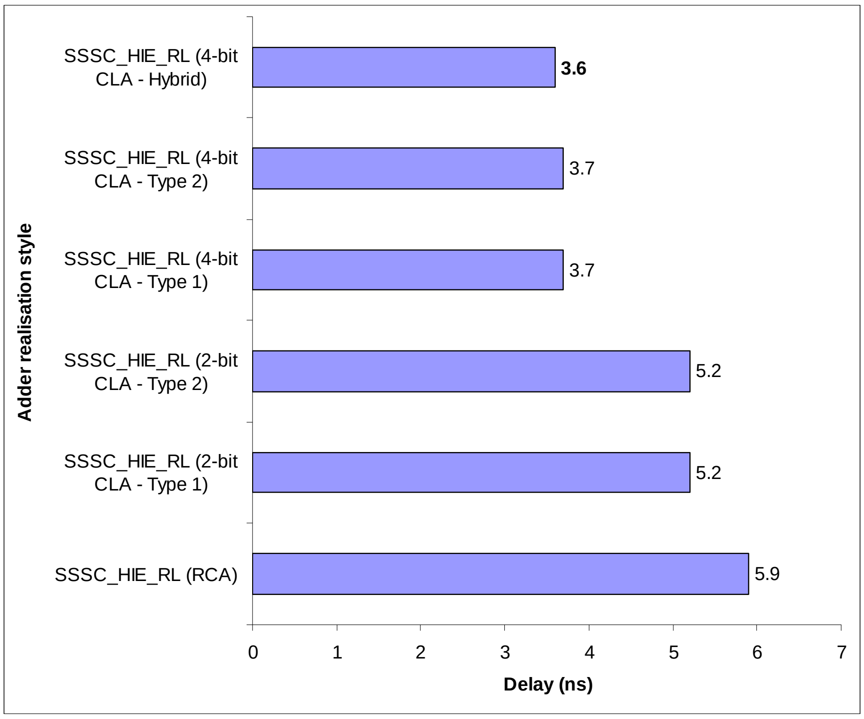


Figure 6.9: Highlighting the merit of HIE CLA adder topology over HIE RCA topology with respect to delay parameter

The performance advantage gained by the improved carry-propagate adder (section carry based CLA adder) scheme over the basic carry-propagate adder (RCA) architecture is illustrated by figure 6.9 for the case of 32-bit addition, incorporating the SSSC_HIE_RL adder. On an average, latency reduction of 12% and 37% has been achieved for the CLA structure employing 2-bit CLA and 4-bit CLA units respectively, in comparison with the RCA topology. In this context, the following chart depicts the compromise made in terms of average power dissipation for the ST lookahead architecture compared to the ST carry-ripple architecture. Evidently, the 2-bit and 4-bit CLA topologies exhibit a relative increase in total power dissipation over the RCA architecture by 5% and 9% respectively.

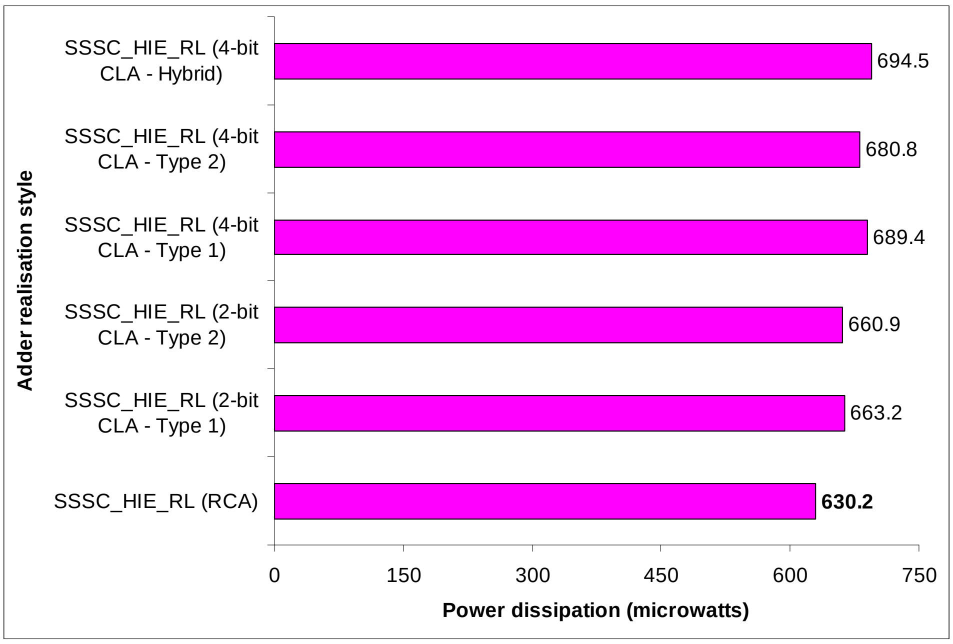


Figure 6.10: Portraying the power increase of HIE CLA adder architecture in comparison with HIE RCA architecture

6.5 Summary and Inferences

High-speed ST adder architectures based on the concept of CLA have been discussed in this Chapter. ST CLA logic realisation based on the section carry formulation has been put forward. Owing to the exponential spatial demand, section carry based lookahead architectures may not be a viable option with many ST approaches and to confirm this, CLA structures corresponding to some methods were constructed and optimised for minimum delay. It is to be noted that the MOSOP heuristic has been helpful in substantially minimising the input space consideration from an initial O(2^{2q+1}) to O($2^{q+1}-1$) with respect to the number of essential prime implicants required, thereby facilitating feasible construction of CLA structures.

With respect to power and area, the Type 2 two-bit CLA architecture incorporating the SSSC_HIE_NRL adder is found to be superior (656.8μW and 7529μm^2). With respect to delay, the SSSC_HIE_NRL adder (hybrid adder based on Type 1 architecture) founded upon a 4-bit lookahead carry is preferable (3.6ns). Compared to the former, the latter facilitates delay reduction by 30.8%, but the former dissipates less power and occupies less area by 4.4% and 30.5% respectively.

The main issue that has been found to hinder the translation of a synchronous CLA structure into a robust gate-level asynchronous implementation is the problem of orphans (gate orphans and/or wire orphans), which appear to be inherent in a recursive carry formulation that affects the robustness property of the adder implementation. This reason also appears to negatively impact gate level ST realisation of parallel prefix adders [163] – [166], where the prefix operation is also recursive involving generate and propagate signals. Nevertheless, such adders could be realised following the block-level relaxation approach [128]. But the parallel prefix adder architecture might experience a greater delay than the proposed SCBCLA architecture since the full P/G (FPG) block [128], in its input-complete version, would incur the delay equivalent of a CE4, an OR3 and two OR2 gates (assuming maximum fan-in of an OR gate is 3). For a 32-bit ST adder, the critical path is likely to comprise five such FPG blocks and the delays due to the above elements can be roughly multiplied and added to the propagation delay of the overflow carry logic in addition to the delay of the initial PG stage. In contrast, for the 32-bit SCBCLA based on a hybrid format of the Type 2 architecture, the longest path traversed would result in a delay equivalent of five full adders, six 4-bit CLAs and a 3-bit CLA. The delay experienced in each ST CLA is equivalent to the delay of a CE2 and an OR2, though the delay values vary according to the CLA unit size.

It is worth studying the potential benefits of the proposed lookahead scheme for addition involving higher operand sizes and the results of this analysis for adders of size 32, 48 and 64 bits are presented in Tables 6.5 and 6.6. In case of the dual-rail encoded CLA adder, the hybrid configuration considered here is founded upon the original Type 2 architecture. The percentage figures given within brackets in Tables 6.5 and 6.6 represent the corresponding reduction in latency and increases in area and average power for the ST CLA adder scheme in relative comparison with the ST RCA scheme.

It can be concluded that the proposed carry-lookahead scheme enables significant reduction in critical datapath delay compared to the basic carry-propagate adder structure at the expense of more area and power dissipation. It has been found that a hierarchical arrangement of CLA units does not guarantee gate-orphan freedom owing to the problem inherent in a recursive carry formulation, as described in section 6.1.

Adder size	Realisation style	Delay (ns)	Area (μm^2)
32 bits	SSSC_DRE (RCA)	5.8	**7081**
	SSSC_DRE (Hybrid with 4-bit CLA)	**3.8** (-34.5%)	9601 (+35.6%)
48 bits	SSSC_DRE (RCA)	8.3	**10611**
	SSSC_DRE (Hybrid with 4-bit CLA)	**4.8** (-41.6%)	14667 (+38.2%)
64 bits	SSSC_DRE (RCA)	10.9	**14129**
	SSSC_DRE (Hybrid with 4-bit CLA)	**6.6** (-39.7%)	19721 (+39.6%)

Table 6.5: Comparing ST ripple carry and hybrid CLA adders in terms of delay and area components for different word widths

Adder size	Realisation style	Power (μW)
32 bits	SSSC_DRE (RCA)	**678.8**
	SSSC_DRE (Hybrid with 4-bit CLA)	765.5 (+12.8%)
48 bits	SSSC_DRE (RCA)	**1011.4**
	SSSC_DRE (Hybrid with 4-bit CLA)	1150.5 (+13.8%)
64 bits	SSSC_DRE (RCA)	**1351.1**
	SSSC_DRE (Hybrid with 4-bit CLA)	1697.1 (+25.6%)

Table 6.6: Comparing ST ripple carry and hybrid CLA adders in terms of power dissipation for different word widths

Chapter 7

Self-Timed Multi-Operand Addition

In Chapters 5 and 6, ST dual-operand addition was dealt with. In this Chapter, ST addition of multiple operands is considered. Multi-input addition is an operation widely prevalent in both multiplication and computation of vector inner products [167] [168]. Various tree structures that are available for multi-operand addition are first discussed in brief. Next, a bit-partitioning strategy that parallelises the addition of multiple operands of arbitrary size is described, with an analysis of carry save adders/logic compressors forming part of the input field partitions.

7.1 Tree Constructs

The carry save adder (CSA) is useful for handling addition of many numbers and therefore suitable for building multipliers and digital filters, where complicated additions are required. Unlike the basic carry-propagate adder (CPA), in a CSA, the carry output signal of the current bit at a level is not transferred to the next-bit adder of the same level as the carry input signal; instead, it is transferred to the next-bit adder in the lower level as the carry input signal. A CSA tree can reduce n binary numbers to two numbers in O(log n) levels [168]. A fast logarithmic time dual-operand adder can then be used to add the two resulting numbers. Hence, CSAs were predominantly used in various tree structures for performing multi-input addition.

The rudimentary tree structure [167], also called as the iterative CSA array, is a straightforward way to accumulate partial products. An n-operand array would consist of $(n-2)$ CSAs and a final CPA stage. As a result, the time complexity of the fundamental array topology would be the summation of the propagation delay of the CSA tree governed by a height of $(n-2)$ and the propagation delay associated with the CPA stage, which is approximately linear. Wallace trees [169] are known for their optimal computation time; in fact, they constitute the theoretically fastest adders when reducing multiple operands to two outputs using CSA trees [170]. In Wallace trees, the number of operands is reduced at the earliest opportunity by employing $n/3$ full adders for all the *m* columns, where ‘*n*’ specifies the number of single-rail operands and ‘*m*’ denotes the size of each operand. This procedure tends to minimise the overall delay by making the final CPA stage as compact as possible. Although the Wallace tree guarantees the lowest overall delay, it requires the largest number of wiring tracks (vertical feed-throughs between adjacent bit-slices), thereby compounding their wiring complexity [171]. The iterative CSA array and Wallace trees represent two extremes in the spectrum of multi-operand addition [168]. While the former features the simplest and regular structure, it is also the slowest; the latter is the fastest, but is also the most difficult structure to implement. Other tree structures proposed for multi-operand addition lie between these two extremes, permitting tradeoffs between regularity and speed [167]. While Wallace used a word-level description of his trees, Dadda gave a refined presentation of the same concept at the bit-level [172]. In Dadda trees, the number of operands is reduced to the next lower number in comparison with the Wallace tree using the fewest number of full adders and half adders possible, i.e. combining of partial product bits takes place as late as possible and this usually leads to a simple CSA tree, unlike Wallace’s method where partial products are combined at the earliest opportunity. The former strategy minimises the number of full and half adders at the expense of a wider CPA, while the latter tends to make the width of the final CPA smaller. Wallace’s and Dadda’s strategies for constructing CSA trees give rise to Wallace and Dadda tree multipliers. An analysis of Dadda and Wallace multiplier delays was performed for different multiplier sizes [173], and it was found that the former showed improvement in speed compared to the latter by 9%-14%; however, this work assumed the presence of only discrete logic gates (AND2, OR2 and INV cells). It has been clarified in [168] that the above strategies, which achieve logarithmic depth reduction based on CSA trees, tend to suffer from the drawback of an irregular structure that subsequently complicates the design and layout. Additionally, connections of varying lengths and complex signal paths lead to logic hazards and signal skew that would have negative implications for power and performance parameters. Overturned-stairs (OS) tree structures [174] can be designed systematically paving way for a simple and regular interconnection scheme in comparison with the Wallace tree, whilst achieving similar speed performance in certain cases. The balanced delay tree [175], on the other hand, requires the smallest number of wiring tracks but suffer from greater delay compared to the OS trees. Nevertheless, it has been widely understood that iterated or recursive structures that would feature a greater degree of structural regularity, less hardware complexity and promise high-speed, such as those incorporating parallel counters or logic compressors, are preferable compared to CSA based tree structures [162] [167] [168] [170] [174] [176].

7.2 Bit-Partitioning Scheme

In CSAs, row-wise parallel addition is performed, where the tree height grows with the increase in the number of input operands by an approximate linear order. Here, a bit-partitioning strategy is considered, which would involve splitting the entire group of operands horizontally into sub-groups as desired, and the results of the sub-groups can then be summed up to produce the final sum. The bit-partitioning approach to multi-input addition is illustrated through figure 7.1, where addition of *n* binary operands, with each operand of size *m* bits is considered, assuming *n* is even. A 'dot' represents a bit position in the figure below.

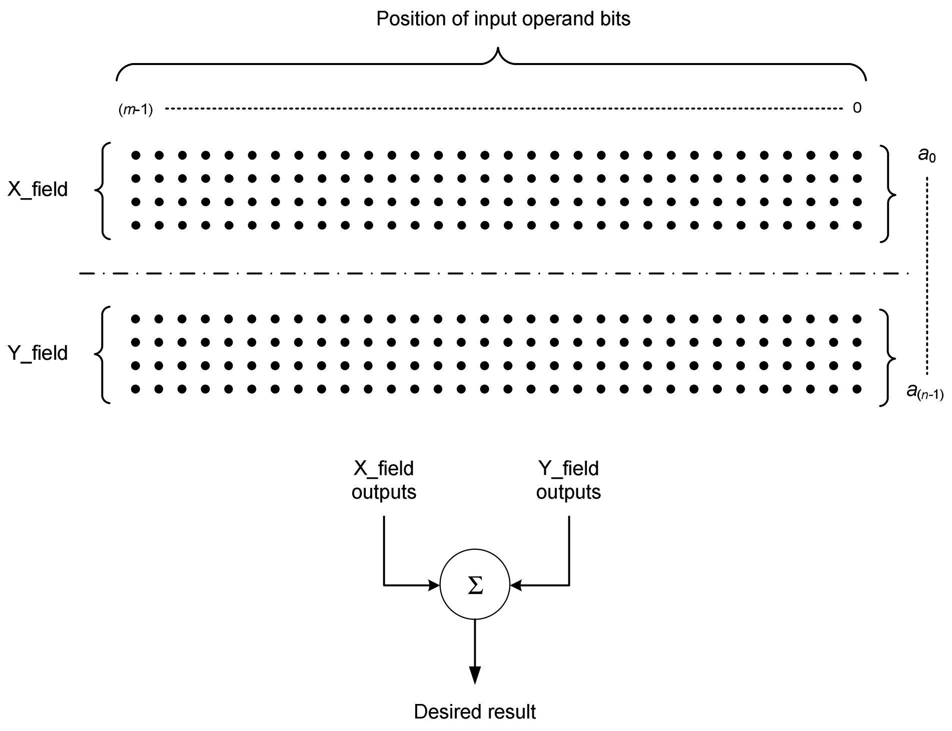


Figure 7.1: Illustration of bit-partitioned multi-input addition scheme

The entire set of input operands ($a_0,....,a_{n-1}$) is divided into two equal-sized groups, namely X_field (that comprises inputs, $a_0,...,a_{(n-1)/2}$) and Y_field (consisting of inputs, $a_{(n+1)/2},...,a_{n-1}$). Addition within the individual fields can be performed either using CSAs or with logic compressors. The sum bits generated from these individual fields can then be added together using a dual-operand adder. Herein, we can use a carry-ripple adder for performing summation of the outputs of X and Y data fields. Use of a RCA stage would help in further evaluating the ST full adder blocks discussed previously.

In general, the combinational bit-partitioning procedure might only effect a slight improvement in delay when many operands have to be added, by way of performing parallel column wise addition of row-wise partitions. For example, considering the addition of 32 single-rail operands, each of size 32-bits, the critical path delay of the multi-operand adder is equivalent to 8 full adder delays (assuming the Wallace bound) and the delay of a 36-bit RCA stage. On the other hand, with eight equal-sized input field partitions, the maximum path delay could be reduced by 2 full adder days. If say 16 operands are to be added, then they could be initially partitioned into 4 fields (say, V, W, X and Y). The outputs of input fields V and W can be combined into an intermediate output field; likewise with input fields X and Y. The sum outputs corresponding to the intermediate output fields can then be added to obtain the desired final result. Alternatively, the outputs of the four input fields can be added together using a single multi-input adder to produce the required result. It can be noticed that additions within the partitions are done in parallel, while the final adder stage that could comprise a simple CPA performs serial computation.

The procedure is scalable and may benefit moderately in terms of latency reduction, as opposed to employing conventional combinational tree type structures for problems of higher dimensions. Also, a high regularity would be implicit within the overall architecture as the input partitions are being replicated. Henceforth, we shall discuss about ST CSAs and logic compressors in the following sections, as employed for the input field partitions, whose evaluation is our main interest with regard to this Chapter.

7.2.1 CSA Based Multi-Operand Addition

Figure 7.2 shows the ST equivalent of a traditional synchronous CSA structure, used for addition of four dual-rail encoded binary numbers (*a*,*b*,*c*,*d*), each of size *n* bits, and the (*n*+1) sum outputs produced are also in dual-rail format. Inputs and outputs with subscript zero correspond to the least significant bits and those with the maximum subscript notation represent the most significant bits. As shown in figure 7.2, there are three adders in three levels – two levels of CSAs and one level of RCA to add four input operands. In each CSA, the output carry signal of the current bit at a level is not transferred to the next bit adder of the same level as the input carry. Instead, the output carry is transferred to the next bit adder in the lower level as the carry input signal. In the top-level adder, three numbers (*a*,*b*,*c*) are added simultaneously, i.e. the bits corresponding to any number could act as the input carries for the full adders of the first level CSA. In the next lower level, an extra number (*d*) is added. The adder in the bottom level is a conventional carry-ripple adder that produces the final sum. The propagation delay of the whole multi-operand adder, in general, is equal to the sum of the delay of two full adder cells in the first two levels and the delay associated with the basic CPA of the final level.

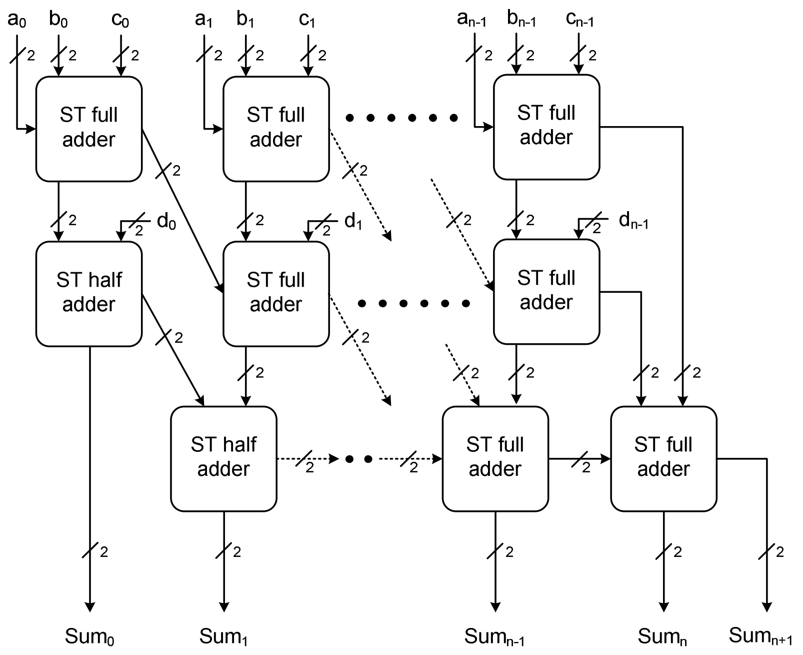


Figure 7.2: ST version of a typical *n*-bit CSA for adding four operands

7.2.2 Compressor Based Multi-Operand Addition

Instead of using CSAs for the partitions, logic compressors can also be employed for adding multiple input operands as shown in figure 7.3. The (4:2) compressor [177] usually takes in five inputs (four inputs in the absence of an input carry) including a carry input from the preceding stage and produces three outputs – two carry outputs, with one carry (*ICarry*) propagating as carry input to the compressor block of the next column in the same row, while the sum (*Sum*) and carry (*Cout*) outputs are fed as inputs to the final stage carry-ripple adder. In essence, it is a 5-bit column adder [168].

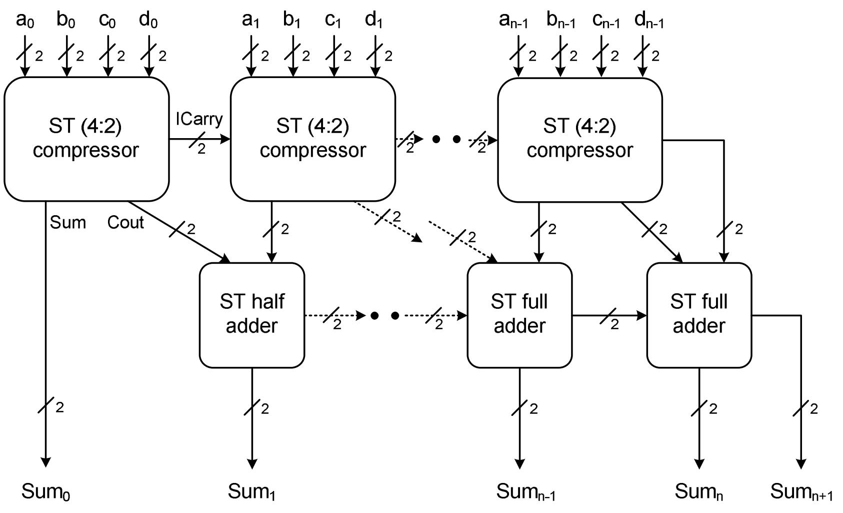


Figure 7.3: ST logic compressor based multi-input adder to add four operands

The efficient realisation of a (4:2) compressor block is necessary for multi-operand addition. It is usual practice to realise compressors using full adder blocks [168] [176] that constitutes a scalable approach rather than synthesising them as a single block – this is because of the input space demand. A typical (4:2) compressor design [178] using two full adder modules is shown in figure 7.4. The ST version of a (4:2) compressor can then be easily derived by replacement of synchronous full adder modules by equivalent ST blocks. It may be noticeable that the compressor shown in figure 7.4 treats a full adder as a CSA and thus the compressor logic is equivalent to that realised by the CSA tree (first two levels, preceding the RCA stage) of figure 7.2.

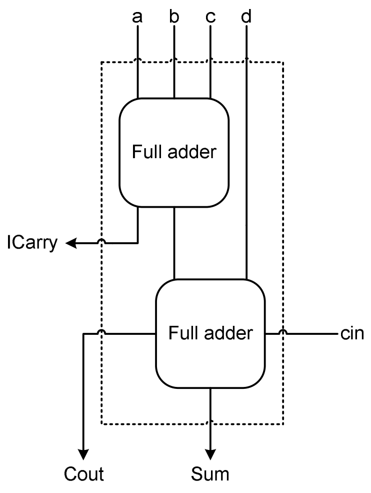


Figure 7.4: A synchronous (4:2) logic compressor implemented using two full adders

Alternatively, a (4:2) compressor can also be realised using discrete gates as shown in figure 7.5 [179]. The proposed weak-indication design of the (4:2) compressor (with input carry), shown in figure 7.6, is based on a translation of the synchronous version given below.

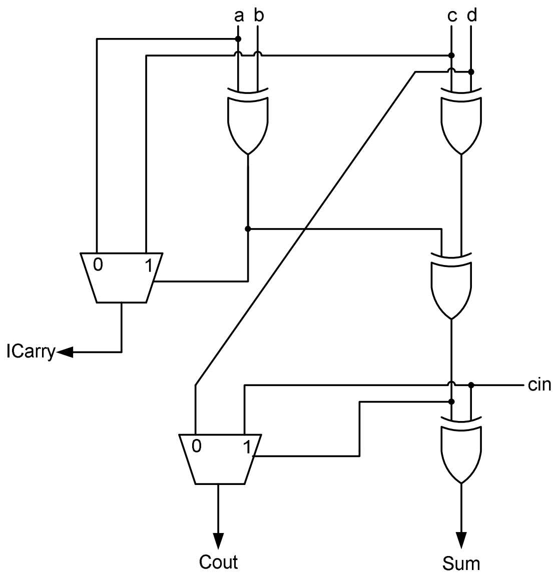


Figure 7.5: A synchronous (4:2) compressor design based on discrete gates

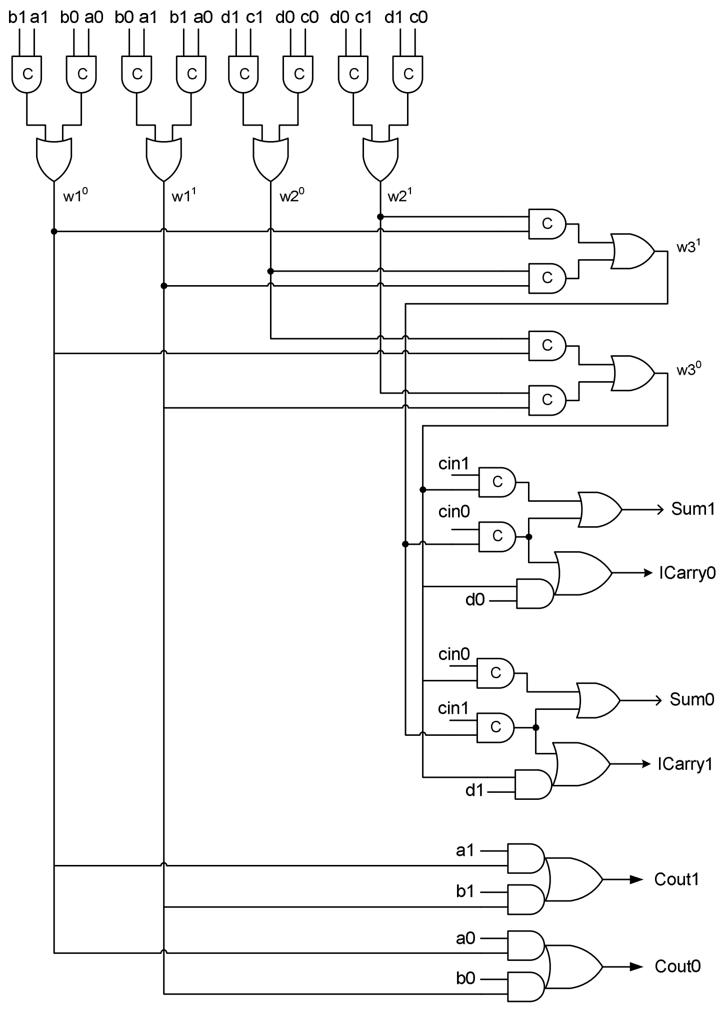


Figure 7.6: ST (4:2) logic compressor block with carry input

The multi-level expressions corresponding to the proposed compressor design shown in figure 7.6 that utilises DRE (hereafter, identified as Sync_ST_compressor_DRE) are given below. Given these, the synthesis of a compressor module without input carry would be straightforward and is portrayed by figure 7.7.

$$Sum1 = w3^0 cin1 + w3^1 cin0 \quad \textbf{(7.1)}$$

$$Sum0 = w3^0 cin0 + w3^1 cin1 \quad \textbf{(7.2)}$$

$$ICarry1 = d1w3^0 + cin1w3^1 \quad \textbf{(7.3)}$$

$$ICarry0 = d0w3^0 + cin0w3^1 \quad \textbf{(7.4)}$$

$$Cout1 = a1w1^0 + c1w1^1 \quad \textbf{(7.5)}$$

$$Cout0 = a0w1^0 + c0w1^1 \quad \textbf{(7.6)}$$

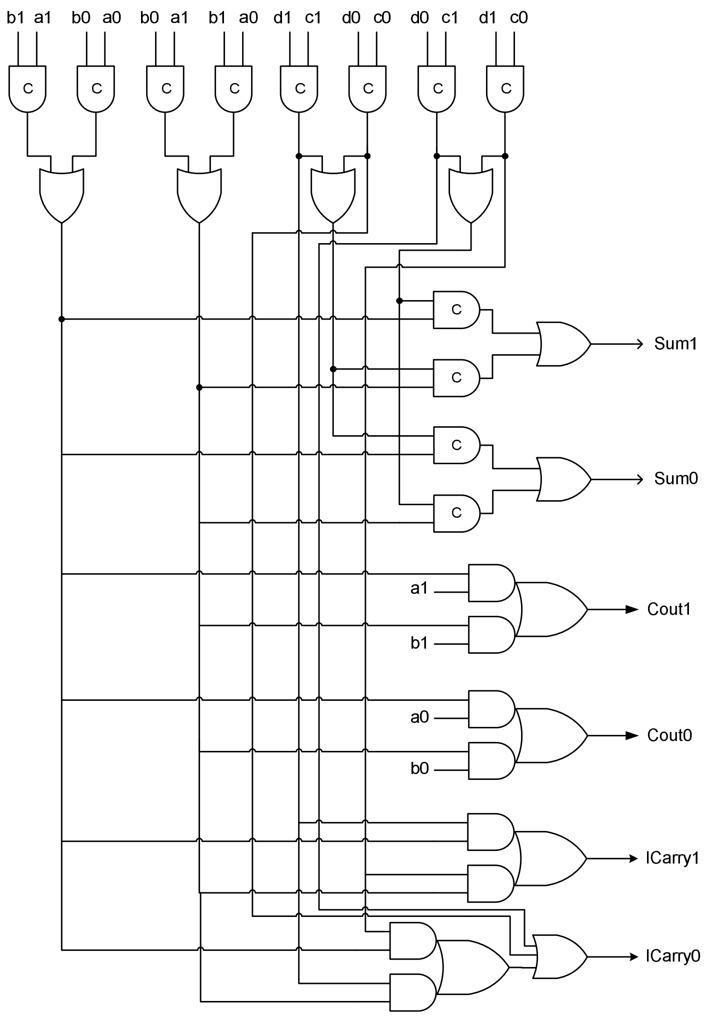


Figure 7.7: ST (4:2) logic compressor module without input carry

From the above diagrams, it may be apparent that the ST compressor realisations correspond to the weak-indication timing discipline, as only the sum output strongly indicates the arrival of all the inputs, while the intermediate and actual carry outputs do not and they are allowed to evaluate/reset in an eager fashion.

7.3 Evaluation and Comparison

In order to analyse the efficacy of CSAs and compressors forming part of the partitions in case of a multi-operand adder, an example scenario of ST addition of 8 single-rail (16 with DRE) input operands, each of size 32 bits (64 bits in case of DRE) was considered. The inputs were divided into two equal input fields and the individual summation result of these two partitions is composed of 34 intermediate single-rail sum outputs, which were then added using a carry-ripple adder to generate the final result that consists of 35 single-rail (70 in dual-rail format) sum outputs. The delay, area and power parameters of this bit-partitioned addition process, assuming CSAs for the input field partitions is given in Table 7.1. Similar input patterns were used as that of the previous adder simulations for the multi-input adder and they were fed at intervals of 25ns to the entire adder, slower than the earlier case (15ns), taking into account the delay of Modified David et al._DRE adder. Weak-indication adders corresponding to various ST design methods were constructed and also subsequently optimised for minimum latency.

Adder realisation style	Delay (ns)	Area (μm^2)	Power (μW)
Seitz_DRE	9.2	45805	3068.3
Singh_DRE	14.7	49959	2931.3
Modified David et al._DRE	19.9	68663	6079.9
DIMS_DRE	16.6	66303	3245.0
Petrify_DRE	9.7	42701	2943.5
Folco et al._DRE	10.8	**38457**	**2311.9**
Toms_DRE	14.1	44866	2397.3
SSSC_DRE	**9.0**	41586	2740.6

Table 7.1: Delay, area and power parameters corresponding to bit-partitioned CSA based ST addition of 8 input operands, each of size 32 bits

By comparing Table 7.1 with Table 5.1, a near similar trend can be observed with respect to all the adders in terms of delay, area and power metrics. It can be seen from Table 7.1, that the Seitz_DRE adder reports a delay increase of only 2% in comparison with the SSSC_DRE adder, which facilitates the least latency amongst all the other multi-input adders, but the former occupies more area to the tune of 10% and suffers from enhanced average power dissipation of 12%. With respect to area and power, Folco et al.'s adder was found to be optimal; however it experiences degradation in delay compared to the proposed adder by 20%.

The design metrics corresponding to multi-operand addition based on the bit-partitioning scheme that considers logic compressors in the input field partitions are given in Table 7.2. (4:2) logic compressors based on Seitz, DIMS, Toms and MOSOP approaches were constructed following a semi-custom design style, with subsequent delay-oriented logic optimisations where feasible. For example, the DIMS weak-indication compressor involved SI logic decomposition, while Seitz's weak-indication compressor entailed logic decomposition of higher fan-in AND gates and replacement of the second-level AND gates by state-holding gates so as to guarantee gate orphan freedom of the physical implementation. The Sync_ST_compressor_HIE represents the equivalent of the Sync_ST_compressor_DRE design that adopts a combination of dual-rail and 1-of-4 codes to encode the inputs (HIE). The increase in datapath delay and area expense for all the adders in relative comparison with the Sync_ST_compressor_DRE based multi-input adder and the overhead in terms of average power for all the adders in comparison with Toms_compressor_DRE based multiple input adder have been highlighted in Table 7.2 to facilitate a quick comparison. The proposed compressor based multi-input adder utilising DRE fares better than that adopting the HIE protocol for the first layer of the first level of the input field partitions in terms of delay, area and power. However, with respect to average power dissipation, the Sync_ST_compressor (DRE and HIE) based multi-input adder implementations tend to consume more than Toms' solutions. This is probably attributable to the greater number of complex gates used in the former and the optimisation that was effected in case of the latter (4 OR2 gates were reduced after logic optimisation of the original synthesis solution).

Adder realisation style	**Delay (ns)**	**Area (μm^2)**	**Power (μW)**
Seitz_compressor_DRE	9.7 (10%)	77611 (1.91×)	3605.3 (49.1%)
DIMS_compressor_DRE	17.3 (97%)	111757 (2.75×)	3974.4 (64.3%)
Toms_compressor_DRE	22.5 (157%)	51950 (1.28×)	**2418.5**
MOSOP_compressor_DRE	8.9 (1%)	72286 (1.78×)	3401.6 (40.6%)
Sync_ST_compressor_DRE	**8.8**	**40608**	2588.6 (7%)
Sync_ST_compressor_HIE	8.9 (1%)	42124 (1.04×)	2667.4 (10.3%)

Table 7.2: Delay, area and power metrics corresponding to bit-partitioned compressor based ST addition of 8 input operands, each of size 32 bits

From Tables 7.1 and 7.2, it can be inferred that the bit-partitioned multi-operand adder employing CSAs (compressors based on CSAs) for the partitions are preferable with respect to power, delay and area in case of Seitz, DIMS and Toms approaches. This is likely because of the greater input space consideration for a direct compressor realisation as opposed to a full adder. The MOSOP_compressor_DRE based multi-input adder however features a similar delay as that of the SSSC_DRE adder based multi-input adder; this is because of the finer SI logic decomposition attainable in case of the former. But due to the greater number of cubes, the area and power parameters are high for the former compared to the latter. On the other hand, the Sync_ST_compressor_DRE based multi-input adder is found to be optimal from the delay perspective. For its respective value of maximum datapath delay, it can be seen that the SSSC_DRE adder based multi-operand adder consumes more area and dissipates more power by 2% and 6% respectively.

It has been observed that hybrid input encoded multi-input adders generally tend to suffer from an increase in delay, area and power parameters over their dual-rail encoded counterparts. The dual-rail and hybrid input encoded versions of the proposed compressor design serves as an illustration in this regard. This is very likely due to the fact that only the primary inputs of the multi-operand adder can be grouped together using the HIE mechanism, while all the intermediate and primary outputs necessitate maintaining the dual-rail convention. As a result, the possible reductions in power dissipation and area requirement of the HIE compressor logic tends to be nullified by the extra power dissipation and area occupancy of the associated encoding circuitry. Hence, encoding of primary inputs in a heterogeneous fashion does not appear to have a beneficial impact on the resultant multi-input adder implementations. This effect is likely even in case of bit-partitioned multi-input addition, which employs CSAs for the input field partitions. Hence, it is opined that DRE might be an optimum DI data encoding convention for effectively implementing ST multi-operand addition in general (also with regard to the bit-partitioning scheme), as opposed to any other heterogeneous input-encoding scheme. This is in contrast to the observation that has been made with respect to ST dual-operand addition in the previous Chapter.

Compressor designs based on many ST logic realisation methods tend to exacerbate the area requirement and eventually have an adverse impact on delay and power metrics due to an increase in the number of logic levels and library elements. This is because the (4:2) compressor logic would quadruple the input space consideration in comparison with a full adder block. Figure 7.8 graphically describes the area expenditure of various ST compressor realisations that make use of DRE or HIE protocols. ‘Series1’ represents the area of a (4:2) ST compressor block without carry input and ‘Series2’ signifies the area assuming input carry.

The values mentioned in the 'vertical bars' of the bar chart specify the area for a cell-based implementation. Indeed, the area figures correspond to that of optimised designs of the respective ST methods.

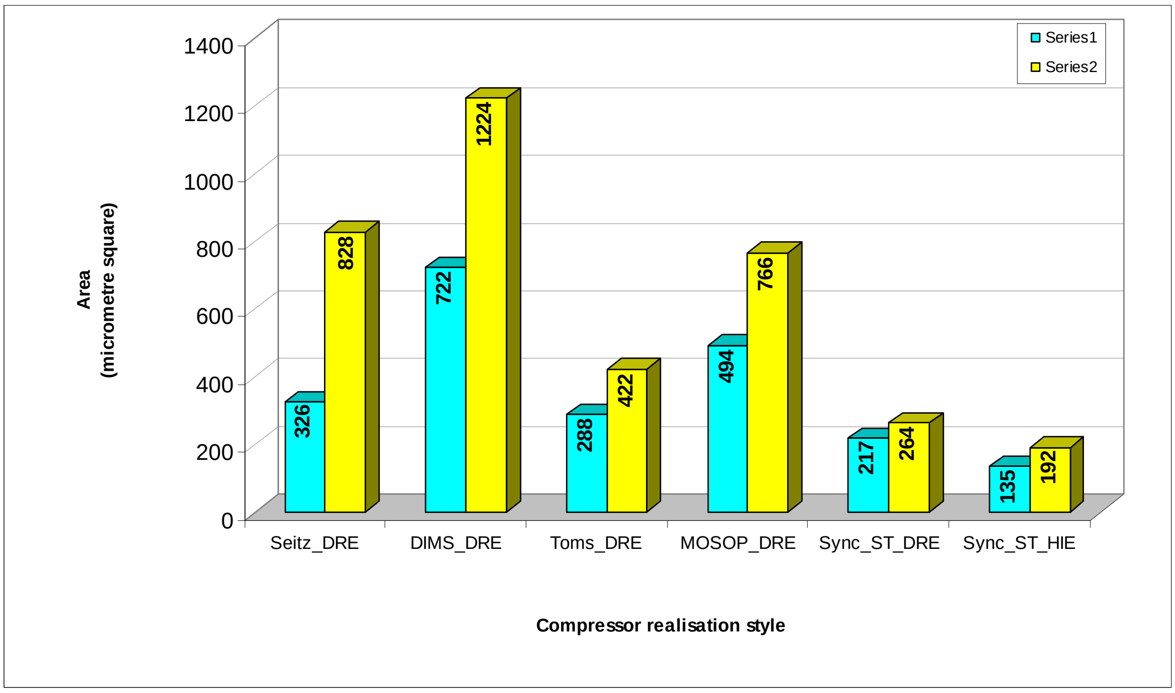


Figure 7.8: Area comparison of ST (4:2) compressors realised using different methods

7.4 Summary and Inferences

ST multiple input addition based on a bit-partitioning strategy has been discussed in this Chapter. The impact of CSAs and compressors on the parallel input field partitions has been analysed for the case study of an addition involving 8 single-rail input operands, each of width 32 bits. It is observed that the CSA tree structure and compressor tree structure based multi-input adders exhibit a near similar performance with regard to this case study, but it appears that the Sync_ST_compressor_DRE based multi-operand adder might be a good design choice from the viewpoint of delay, area and power. Consequently, it could be of use in building higher order ST logic compressors.

In fact, ST CLA adders can be employed in place of the ST RCAs to minimise the latency of multi-operand addition, as highlighted in figure 7.9. Of course, this can be expected to incur penalties in terms of area and power metrics due to the greater area overhead for CLA adders over RCAs.

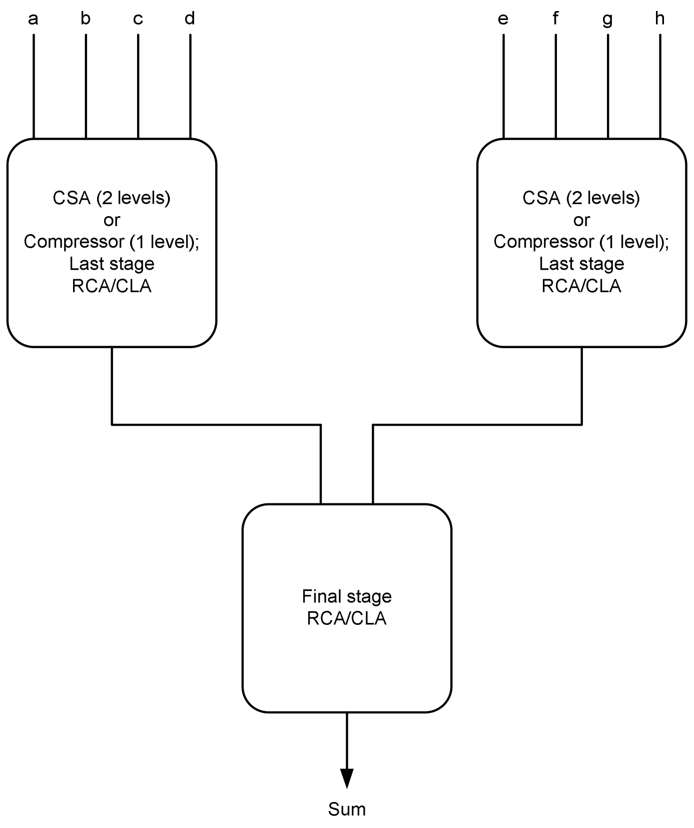

a
b
c
d
e
f
g
h
CSA (2 levels) or Compressor (1 level); Last stage RCA/CLA
CSA (2 levels) or Compressor (1 level); Last stage RCA/CLA
Final stage RCA/CLA
Sum

Figure 7.9: Describing positioning of CLA adders as a replacement for RCAs in the multi-operand adder

Figure 7.10 portrays the delay reduction that could be attained by incorporating the proposed ST section carry based CLA adders instead of the ST carry-ripple adders, in the intermediate and final dual-operand carry propagate adder stages of the multi-operand adder that employs a CSA tree (SSSC_DRE adder) or compressor tree (Sync_ST_compressor_DRE) for the input field partitions.

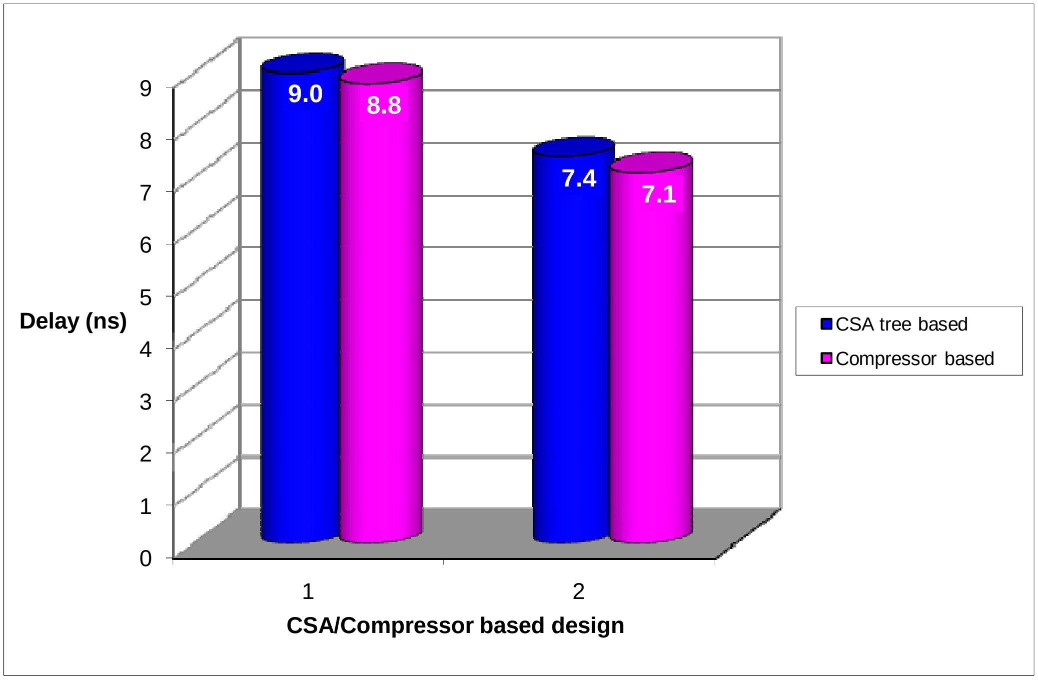


Figure 7.10: Latency metrics for CLA adders and carry-ripple adders to perform bit-partitioned ST addition of 8 inputs, each of width 32 bits

Label '1' on the X-axis represents the delay values corresponding to the usage of RCAs, while label '2' signifies the delay figures obtained by utilising hybrid CLA adders that are based on the Type 2 configuration, which comprise 3-bit and 4-bit lookahead carry logic. It is evident that CLA adders enable a reduction in latency by around 19%, on an average, for this case study.

Adder realisation style	**Area (μm^2)**	**Power (μW)**
CSA tree based (with RCAs)	**41586**	**2740.6**
CSA tree based (with CLA adders)	49314	2915.0
Compressor based (with RCAs)	**40608**	**2588.6**
Compressor based (with CLA adders)	48336	2763.1

Table 7.3: Area and power figures for bit-partitioned ST addition of 8 inputs, each of width 32 bits, using RCAs and CLA adders

However, the downside being associated increases in area and power figures, as can be seen in Table 7.3. Hence, the RCA topology incorporated multi-input adders exhibit less area expenditure and dissipate less power than the multi-input adders that feature the CLA topology by 16% and 6% respectively, on a mean basis. Hence, there is a trade-off involved between the above two approaches in terms of delay and area/power.

It should be noted that the exemplar circuit considered here and the analysis that accompanied might only be a representative of the proposition and just serves as a demonstration of the concept. For example, to add eight binary operands of arbitrary size, a compressor tree based realisation would encounter a delay of two compressor modules and a final CPA stage, whereas the bit-partitioned adder could be expected to have a critical signal path traversed through a compressor block, a full adder block and the final CPA stage. It is also anticipated that though a generic grouping formulation may be helpful with respect to addition of operands of random size, the actual bit-partitioning decision may better follow a case-by-case analysis approach when concerned with physical implementation. This is because many permutations may be feasible and any one of them could merit consideration when targeting optimisation of delay/area/power. It is to be noted that only a straightforward bit-partitioning strategy has been considered here. Other optimisations might include determining an optimal pipelining strategy along with considering ST implementation of other conventional techniques for multi-operand addition [167] [168] [176] [178]. These are left for further investigation.

Chapter 8

Conclusions and Scope for Further Work

In this final Chapter, a concise description of the thesis contributions is provided along with certain insights, followed by a brief discussion of a direction for future research.

8.1 Summarising the Thesis Contributions

ST design presents an attractive alternative to conventional synchronous design, especially in the current/future era where issues such as reliability and variability tend to assume greater significance than quality-of-results. Hence, pursuing research in the domain of ST logic is deemed necessary and plausible, owing to the fact that ST designs are inherently elastic and hence they are better placed to tackle the challenges of process, device and parameter variations at the logic level rather than traditional synchronous design methods. However, they necessitate unorthodox design methodologies, which involve greater complexities that in turn complicate the synthesis process.

In brief, this thesis makes the following major contributions towards advancement of existing knowledge in the domain of ST design:

- A set theory based heuristic for efficient self-timed synthesis of random combinational logic specifications
- A mechanism of achieving global indication with the intent of reducing delay/area
- Proposition of redundant logic insertion as an elegant method to speed-up dual-operand addition based on a RCA topology, with negligible area/power overheads
- ST section carry based CLA adder architectures, which further minimise the latency of dual-operand addition in comparison with the basic CPA structure
- A bit-partitioning scheme for multi-operand addition and a ST logic compressor design

The major issue that has been dealt with in Chapter 4 is the reduction of input space, which expands exponentially causing a state explosion, without compromising the robustness criterion. This has been possible through the proposition of a novel heuristic, based on the principles of set theory, to derive MDSOP forms from MSOP expressions. The heuristic has been implemented in Java and its efficacy vis-à-vis other approaches has been analysed with respect to some combinational benchmarks. However, this heuristic purely corresponds to synchronous logic. Hence, it has been subsequently extended to address ST designs embodying DI dual-rail encoded datapaths and the resultant MOSOP procedure has also been implemented in Java. To prove the feasibility of the implementation with respect to obtaining solutions for larger problems in reasonable time, asynchronous equivalents of numerous combinational benchmark specifications (i.e. function blocks) that comprise several inputs and outputs have been targeted. On an average, the algorithm for function block realisations has attained 22% logic sharing. Direct symbolic translation coupled with signal insertion may be used to implement datapaths featuring higher order 1-of-*n* codes. Furthermore, a system configuration has also been proposed in support of the proposed MOSOP heuristic so as to retain the benefits of the synthesis scheme. Examples of a multiplexer and demultiplexer were considered to validate the benefits of the proposition.

It appears that asynchronous adders were conceived as early as the 50's [181] – the carry-completion sensing adder (CCSA), which is perceived to be the asynchronous equivalent of a synchronous RCA. But a majority of the efforts relating to implementation of arithmetic circuits in asynchronous design style were mainly focussed on the non-DI bundled-data protocol that does not correspond to a robust signalling convention; a representative list includes [182] – [192]. This may be because the underlying combinational logic is usually similar to that that could be implemented using synchronous design methods. Also, relatively less modification would be required even with encoding of certain signals in dual-rail format, For example, the carry signal in a typical CCSA is alone dual-rail encoded, while the rest of the signals are retained in single-rail format. In contrast, ST arithmetic circuit realisations (i.e. those with no matched delays) have received relatively little attention. In this regard, this thesis has specifically dealt with this issue from a synthesis perspective by drawing inputs from the proposed MOSOP heuristic. In Chapter 5, various carry-ripple adders corresponding to different ST approaches have been constructed and analysed with the intent of minimising the datapath latency within this architecture. To this end, firstly, various single-bit and dual-bit adders based on DRE, HIE or HE have been implemented. Secondly, a hybrid scheme involving a mix of single-bit and dual-bit adder modules has been considered that only facilitated meagre delay reduction. Finally, to achieve dramatic reduction in datapath delay, the concept of redundant logic insertion was put forward and its potential has been demonstrated by comprehensively analysing its impact on single-bit and dual-bit adders that are incorporated into the basic CPA structure.

To achieve non-linear computation, even for the worst-case, a new ST section carry based CLA topology was proposed in Chapter 6. In this connection, two slightly different architectural styles (Type 1 and Type 2) were examined and their benefits were studied based on 2-bit and 4-bit lookahead carry generators. CLA adders based on many other methods were also constructed for the purpose of comparative evaluation. A theoretical estimation of the complexity involved in realising higher order lookahead modules was also provided. Further, a peephole optimisation involving a hybrid scheme within these structural styles was performed and it was found that the Type 2 configuration based on the 4-bit lookahead carry is suitable for dual-rail encoded datapaths, while the Type 1 configuration based on a 4-bit lookahead is optimal for hybrid input encoded datapaths from a delay perspective. Also, the problems involved in direct translation of conventional synchronous lookahead architecture into its ST equivalent had been elucidated.

Considering ST dual-operand addition, dealt with in Chapters 5 and 6, the proposed section carry based CLA scheme has the best delay metric. The hybrid SSSC_HIE_NRL adder when incorporated into a 4-bit lookahead carry structure (corresponding to the Type 1 architecture) leads to critical path delay reduction by 14.3% compared to the hybrid DSSC_CCAO_global adder, which is the fastest with regard to the basic CPA topology. Nevertheless, the latter features less area occupancy and lower power dissipation by 18% and 5% respectively. In general, ST RCAs comprising single-bit adder modules were found to occupy less area and dissipate less average power than their CLA counterparts but suffer from delay degradation. For example, the SSSC_HIE_NRL adder corresponding to the basic ST RCA topology reports the least power and area requirement amongst all the dual-operand adders, exhibiting reduced power dissipation and area occupancy compared to the fastest CLA adder by 10% and 39% respectively, but accompanied with a corresponding increase in delay to the tune of 122%. Hence, in view of these, it can be stated that the ST SCBCLA adder features approximately half the longest datapath delay of a ST RCA structure whilst suffering from 11% increased power dissipation and 63% more area expenditure.

In Chapter 7, ST multi-input addition has been dealt with and a simple bit-partitioning scheme has been described. The advantage of compressors over CSA trees has been explained through simulation results for the case study of a bit-partitioned addition, comprising eight single-rail input operands, each of size 32-bits. In this connection, compressor based and CSA based bit-partitioned multi-input adders based on different ST approaches have also been constructed. It has been inferred that, in general, compressors could be beneficial in terms of delay, while CSAs might occupy less area and enjoy low power dissipation when used for the partitions. It is opined that this observation might be a generalisation for ST addition involving multiple input operands. Given this, the exact benefit with respect to delay for the former, and area and power for the latter may however vary depending on the number of input operands and their corresponding sizes.

8.2 Multi-Level Synthesis of Weak-Indication Circuits

An important direction for further research is to extend the two-level MOSOP heuristic proposed in this thesis to multiple levels, especially with the aim of propounding a universal solution for weakly indicating realisations of arbitrary combinational logic. This entails greater complexity mainly due to the need for preserving the cover constraint over multiple levels of logic, so that the unique successor set and acknowledgement criteria can be upheld. Nevertheless, this is currently achievable with relative ease with respect to a restricted class of functions (certain logic and arithmetic circuits, which are generally weighted functions), where suitable candidates for SI decomposition tend to become available from within the MOSOP expressions. This seems to have been possible owing to the reason that the functions tend to exhibit full or partial symmetricity with respect to permutation of its variables or literals. At present, this constitutes only a preliminary solution [152], but the search is for a universal method that could address any random functionality.

Decomposition of larger cubes into smaller physically realisable cubes primarily depends upon the possibility for performing SI decomposition. It may be that in many cases, such an opportunity may not exist (i.e. the cubes corresponding to different function outputs may not be orthogonal to each other in the first place) or even if it does, the size of the SSIC may still exceed the maximum element size constituting the base function set. In such situations, the opportunity for finding a mutually orthogonal SSIC/PSIC needs to be explored whose existence cannot be guaranteed. It may be that these issues can likely be overcome through selective expansion of certain PSICs, but again the decomposed solution may not be physically realisable. Therefore, the likelihood for the non-availability of suitable orthogonal cubes for SI decomposition and the impossibility in effecting decomposition to a finer level were found to be the major reasons that hamper multi-level implementation of combinational logic as weak-indication circuits.

To overcome these drawbacks, a possible solution is suggested in this thesis. The concept of 'complementary cubes insertion' is proposed to supplement the above procedure that can act as a driver for facilitating weakly indicating realisations of any combinational logic specification. In simple terms, this amounts to introducing complementary OFF-set/ON-set cubes in a rail of the encoded function block output, whose complementary rail actually contains an indecomposable ON-set/OFF-set cube respectively. The introduction of such complementary cubes would not affect its functionality. This is because, in general, p completely specified cubes corresponding to the true rail of a function output implies that q completely specified cubes would belong to the false rail of that function output, where $2^m = (p + q)$ signifies the Boolean space, with m referring to the number of elements comprising the support set of the indecomposable cube. Therefore, the procedure would lead to inclusion of redundant cubes, which is unavoidable. It should be noted that this would not necessarily require consideration of the entire input space, since m may not be equal to n, where n represents the number of concurrent single-rail inputs. With the availability of 2^m cubes, the problem of SI decomposition of larger sized cubes gets solved as all the 2^m distinct cubes exhibit mutual orthogonality between them and therefore they can be decomposed up till a specified granularity of the base function set. However, it should be borne in mind that for the worst-case scenario m may become equal to n, though this may rarely occur. Even in such a case, there may be a good likelihood of the existence of many cubes that may match with the new cubes introduced and duplication of cubes can then be eliminated.

In addition, considering the notion of covering cubes and covered cubes, a larger sized cube corresponding to a function block output can be conveniently expressed with reference to a smaller sized cube (that belongs to a different function block output) through substitution as described in section 4.2.3. This assumes that the smaller sized cube acts as the covering cube and the larger sized cube is the covered cube. Given the above insights, the effectiveness of the proposed solution may ultimately depend upon the nature of logic functionality considered and the initial MOSOP solution obtained. Implementation of this procedure and further analysis targeting asynchronous equivalents of combinational benchmarks, exploring the opportunities for global/local optimisations and studying the beneficial impact of various DI data encodings within this framework are reserved as future work.

References

1. S.H. Unger, "Double-edge-triggered flip-flops," *IEEE Trans. on Computers*, vol. 30, no. 6, pp. 447-451, June 1981.
2. M. Pedram, Q. Wu and X. Wu, "A new design of double edge triggered flip-flops," *Proc. Asia and South Pacific Design Automation Conference*, pp. 417-421, 1998.
3. W. Chung, T. Lo and M. Sachdev, "A comparative analysis of low-power low-voltage dual-edge-triggered flip-flops," *IEEE Trans. on VLSI Systems*, vol. 10, no. 6, pp. 913-918, December 2002.
4. A.L. Fisher and H.T. Kung, "Synchronizing large VLSI processor arrays," *IEEE Trans. on Computers*, vol. C-34, no. 8, pp. 734-740, August 1985.
5. H.B. Bakoglu, J.T. Walker and J.D. Meindl, "A symmetric clock-distribution tree and optimized high-speed interconnections for reduced clock skew in ULSI and WSI circuit," *Proc. IEEE International Conf. on Computer Design*, pp. 118-122, 1986.
6. E.G. Friedman and S. Powell, "Design and analysis of a hierarchical clock distribution system for synchronous standard cell/macrocell VLSI," *IEEE Journal of Solid-State Circuits*, vol. SC-21, no. 2, pp. 240-246, April 1986.
7. S.Y. Kung and R.J. Gal-Ezer, "Synchronous versus asynchronous computation in VLSI array processors," *Proc. of the SPIE*, vol. 341, pp. 53-65, 1982.
8. E.G. Friedman, "Clock distribution networks in synchronous digital integrated circuits," *Proc. of the IEEE*, vol. 89, no. 5, pp. 665-692, May 2001.
9. G.E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, April 1965.
10. G.D. Wilk, R.M. Wallace and J.M. Anthony, "High-κ gate dielectrics: current status and materials properties considerations," *Journal of Applied Physics*, vol. 89, no. 10, pp. 5243-5275, 2001.
11. S.R. Nassif, "Design for variability in DSM technologies," *Proc. International Symp. on Quality Electronic Design*, pp. 451-454, 2000.
12. K. Bernstein, D.J. Frank, A.E. Gattiker, W. Haensch, B.L. Ji, S.R. Nassif, E.J. Nowak, D.J. Pearson and N.J. Rohrer, "High-performance CMOS variability in the 65-nm regime and beyond," *IBM Journal of Research and Development*, vol. 50, no. 4/5, pp. 433-450, 2006.
13. D. Singh, "Prospects for low power microprocessor design," *Keynote Address*, *Proc.*

International Workshop on Low Power Design, October 1994.

14. D. Brooks, V. Tiwari and M. Martonosi, “Wattch: a framework for architectural level power analysis and optimizations,” *Proc. International Symp. on Computer Architecture,* pp. 83-94, 2000.
15. A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic and P.J. Hazewindus, “The first asynchronous microprocessor: the test results,” *ACM SIGARCH Computer Architecture News*, vol. 17, no. 4, pp. 95-98, June 1989.
16. S.B. Furber, P. Day, J.D. Garside, N.C. Paver and J.V. Woods, “AMULET1: a micropipelined ARM,” *Digest of Papers, IEEE Comp. Society International Conference (COMPCON),* pp. 476-485, 1994.
17. T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako and A. Takamura, “TITAC: design of a quasi-delay-insensitive microprocessor,” *IEEE Design and Test of Computers*, vol. 11, no. 2, pp. 50-63, April 1994.
18. A.J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth and U. Cummings, “The design of an asynchronous MIPS R3000 processor,” *Proc. 17th Conf. on Advanced Research in VLSI,* pp. 164-181, 1997.
19. T. Nanya, A. Takamura, M. Kuwako, M. Imai, T. Fujii, M. Ozawa, I. Fukasaku, Y. Ueno, F. Okamoto, H. Fujimoto, O. Fujita, M. Yamashina and M. Fukuma, “TITAC-2: a 32-bit scalable-delay-insensitive microprocessor,” *Proc. HOT Chips IX*, pp. 19-32, 1997.
20. H. van Gageldonk, K. van Berkel, Ad Peeters, D. Baumann, D. Gloor and G. Stegmann, “An asynchronous low power 80C51 microcontroller,” *Proc. 4th International Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp. 96-107, 1998.
21. M. Renaudin, P. Vivet and F. Robin, “ASPRO-216: a standard-cell QDI 16-bit RISC asynchronous microprocessor,” *Proc. International Symp. on Advanced Research in Asynchronous Circuits and Systems,* pp. 22-31, 1998.
22. S.B. Furber, J.D. Garside, P. Riocreux, S. Temple, P. Day, J. Liu and N.C. Paver, “AMULET2e: an asynchronous embedded controller,” *Proc. of the IEEE,* vol. 87, no. 2, pp. 243-256, February 1999.
23. S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken and B. Agapiev, “RAPPID: an asynchronous instruction length decoder,” *Proc. International Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp. 60-

70, 1999.

24. S.B. Furber, D.A. Edwards and J.D. Garside, “AMULET3: a 100 MIPS asynchronous embedded processor,” *Proc. International Conf. on Computer Design*, pp. 329-334, 2000.
25. J.D. Garside, W.J. Bainbridge, A. Bardsley, D.M. Clark, D.A. Edwards, S.B. Furber, J. Liu, D.W. Lloyd, S. Mohammadi, J.S. Pepper, O. Petlin, S. Temple and J.V. Woods, “AMULET3i – an asynchronous system-on-chip,” *Proc. International Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp. 162-175, 2000.
26. A. Semenov, A.M. Koelmans, L. Lloyd and A. Yakovlev, “Designing an asynchronous processor using Petri nets,” *IEEE Micro*, vol. 17, no. 2, pp. 54-64, March-April 1997.
27. M. Lewis and L. Brackenbury, “CADRE: an asynchronous embedded DSP for mobile phone applications,” *Design Automation for Embedded Systems*, vol. 6, pp. 451-475, 2002.
28. A.J. Martin, M. Nystrom, K. Papadantonakis, P.I. Penzes, P. Prakash, C.G. Wong, K.S. Ko, B. Lee, E. Ou, J. Pugh, E.-V. Talvala, J.T. Tong and A. Tura, “The Lutonium: a sub-nanojoule asynchronous 8051 microcontroller,” *Proc. 9th International Symp. on Asynchronous Circuits and Systems*, pp. 14-23, 2003.
29. L.A. Plana, P.A. Riocreux, W.J. Bainbridge, A. Bardsley, S. Temple, J.D. Garside and Z.C. Yu, “SPA – a secure Amulet core for smartcard applications,” *Microprocessors and Microsystems*, vol. 27, pp. 431-446, 2003.
30. D.E. Muller, “Asynchronous logics and application to information processing,” in H. Aiken and W.F. Main (Eds.), *Switching Theory in Space Technology*, Stanford University Press, 1963.
31. R.E. Miller, *Switching Theory Volume 2: Sequential Circuits and Machines*, John Wiley & Sons, New York, 1965.
32. S.H. Unger, *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, New York, 1969.
33. Semiconductor Industry Association's International Technology Roadmap for Semiconductors reports, Available: http://www.itrs.net/reports.html
34. G.F. Bouesse, G. Sicard, A. Baixas and M. Renaudin, “Quasi delay insensitive asynchronous circuits for low EMI,” *Proc. 4th International Workshop on Electromagnetic Compatibility of Integrated Circuits*, pp. 27-31, 2004.
35. K. van Berkel, R. Burgess, J. Kessels, Ad Peeters, M. Roncken, and F. Schalij, “A fully-

asynchronous low-power error corrector for the DCC player," *Proc. 41st IEEE International Solid State Circuits Conference,* pp. 88-89, 1994.

36. K. van Berkel, M.B. Josephs and S.M. Nowick, "Scanning the technology: applications of asynchronous circuits," *Proc. of the IEEE*, vol. 87, no. 2, pp. 223-233, February 1999.
37. N. Karaki, "Asynchronous design: an enabler for flexible microelectronics," Invited Talk, *Proc. 12th IEEE International Symp. on Asynchronous Circuits and Systems*, pp. xii, 2006.
38. S. Hauck, "Asynchronous design methodologies: an overview," *Proc. of the IEEE*, vol. 83, no. 1, pp. 69-93, January 1995.
39. Al Davis and S.M. Nowick, "An introduction to asynchronous circuit design," *Technical Report* UUCS-97-013, Computer Science Dept., University of Utah, September 1997.
40. J. Sparso and S.B. Furber (Eds.), *Principles of Asynchronous Circuit Design: A Systems Perspective,* Kluwer Academic Publishers, 2001.
41. I.E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720-738, June 1989.
42. S.B. Furber and P. Woods, "Four-phase micropipeline latch control circuits," *IEEE Trans. on VLSI Systems*, vol. 4, no. 2, pp. 247-253, June 1996.
43. B. Bose, "On unordered codes," *IEEE Trans. on Computers*, vol. 40, no. 2, pp. 125-131, February 1991.
44. T. Verhoeff, "Delay-insensitive codes: an overview," *Distributed Computing*, vol. 3, no. 1, pp. 1-8, 1988.
45. D.W. Lloyd and J.D. Garside, "A practical comparison of asynchronous design styles," *Proc. 7th International Symp. on Asynchronous Circuits and Systems*, pp. 36-45, 2001.
46. V. Akella, N.H. Vaidya and G.R. Redinbo, "Limitations of VLSI implementation of delay-insensitive codes," *Proc. 26th Annual International Symp. on Fault-Tolerant Computing*, pp. 208-217, 1996.
47. S.J. Piestrak, "Membership test logic for delay-insensitive codes," *Proc. International Symp. on Asynchronous Circuits and Systems*, pp. 194-204, 1998.
48. C.V. Freiman, "Optimal error detection codes for completely asymmetric binary channels," *Information Control*, vol. 5, pp. 64-71, March 1962.
49. S.B. Furber, A. Efthymiou and M. Singh, "A power-efficient duplex communication system," *Proc. Asynchronous INTerfaces: tools, techniques and implementations (AINT)*

Workshop, 2000.

50. W.J. Bainbridge, W.B. Toms, D.A. Edwards and S.B. Furber, "Delay-insensitive, point-to-point interconnect using *m*-of-*n* codes," *Proc. 9th International Symp. on Asynchronous Circuits and Systems*, pp. 132-140, 2003.
51. S.J. Piestrak and T. Nanya, "Towards totally self-checking delay-insensitive systems," *Proc. 25th International Symp. on Fault-Tolerant Computing*, pp. 228-237, 1995.
52. S.M. Nowick and D.L. Dill, "Synthesis of asynchronous state machines using a local clock," *Proc. International Conf. on Computer Design*, pp. 192-197, 1991.
53. K. Yun, D. Dill and S.M. Nowick, "Synthesis of 3D asynchronous state machines," *Proc. International Conf. on Computer Design*, pp. 346-350, 1992.
54. Al Davis, B. Coates and K. Stevens, "The post office experience: designing a large asynchronous chip," *Proc. 26th Annual Hawaii International Conf. on Systems Sciences*, vol. 1, pp. 409-418, 1993.
55. J.A. Waicukauski, E. Lindbloom, B.K. Rosen and V.S. Iyengar, "Transition fault simulation," *IEEE Design and Test of Computers*, vol. 4, no. 2, April 1987.
56. G.L. Smith, "Model for delay faults based upon paths," *Proc. International Test Conference*, pp. 342-349, 1985.
57. E.J. McCluskey, *Logic Design Principles: with emphasis on testable semi-custom circuits*, Prentice-Hall, Englewood Cliffs, NJ, 1986.
58. J.A. Brzozowski and J.C. Ebergen, "Recent developments in the design of asynchronous circuits," *Technical Report* CS-89-18, Dept. of Computer Science, University of Waterloo, 1989.
59. W.A. Clark, "Macromodular computer systems," *AFIPS Proc. Spring Joint Computer Conference*, 1967.
60. W.A. Clark, and C.E. Molnar, "Macromodular computer systems," *Computers in Biomedical Research*, B.D. Waxman and R. Stacey (Eds.), pp. 45–85, vol. IV, 1974, Academic Press, NY.
61. J.T. Udding, "A formal model for defining and classifying delay-insensitive circuits and systems," *Distributed Computing*, vol. 1, no. 4, pp. 197-204, 1986.
62. A.J. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," *Distributed Computing*, vol. 1, no. 4, pp. 226-234, December 1986.

63. A.J. Martin, "The limitation to delay-insensitivity in asynchronous circuits," *Proc. 6th MIT Conf. on Advanced Research in VLSI*, pp. 263-278, MIT Press, 1990.
64. J.A. Brzozowski and Jo C. Ebergen, "On the delay-sensitivity of gate networks," *IEEE Trans. on Computers*, vol. 41, no. 11, pp. 1349-1360, November 1992.
65. J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory Languages and Computation*, 1st Edition, Addison-Wesley Publishing Company, 1979.
66. K. van Berkel, "Beware the isochronic fork," *Integration, the VLSI Journal*, vol. 13, no. 2, pp. 103-128, June 1992.
67. A.J. Martin and P. Prakash, "Asynchronous nano-electronics: preliminary investigation," *Proc. 14th IEEE International Symp. on Asynchronous Circuits and Systems*, pp. 58-68, 2008.
68. R. Manohar and A.J. Martin, "Quasi-delay-insensitive circuits are Turing-complete," Invited Paper, *Proc. International Symp. on Advanced Research in Asynchronous Circuits and Systems*, 1996.
69. D.E. Muller and W.S. Bartky, "A theory of asynchronous circuits," *Proc. International Symp. on the Theory of Switching*, part I, pp. 204-243, Harvard University Press, 1959.
70. C.L Seitz, "Self-timed VLSI systems," *Proc. Caltech Conf. on VLSI*, January 1979.
71. C.L Seitz, "System Timing," in *Introduction to VLSI Systems*, C. Mead and L. Conway (Eds.), pp. 218-262, Addison-Wesley, Reading, MA, 1980.
72. I. David, R. Ginosar and M. Yoeli, "Self-timed is self-checking," *Journal of Electronic Testing: Theory and Applications*, vol. 6, no. 2, pp. 219-228, April 1995.
73. A.J. Martin, "Formal program transformations for VLSI circuit synthesis," in *Formal Development of Programs and Proofs*, E.W. Dijkstra (Ed.), pp. 59-80, Addison-Wesley, 1989.
74. T.-Y. Wuu and S.B.K. Vrudhula, "A design of a fast and area efficient multi-input Muller C-element," *IEEE Trans. on VLSI Systems*, vol. 1, no. 2, pp. 215-219, June 1993.
75. M. Shams, Jo C. Ebergen and M.I. Elmasry, "Optimizing CMOS implementations of the C-element," *Proc. IEEE International Conf. on Computer Design*, pp. 700-705, 1997.
76. V.I. Varshavsky (Ed.), *Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete systems*, Chapter 4: Aperiodic Circuits, pp. 77-85, (Translated from the Russian by Alexandre V. Yakovlev), Kluwer

Academic Publishers, 1990.

77. K.M. Fant and S.A. Brandt, "NULL convention logic: a complete and consistent logic for asynchronous digital circuit synthesis," *Proc. International Conf. on Application Specific Systems, Architectures and Processors*, pp. 261-273, 1996.
78. K.M. Fant, *Logically Determined Design: Clockless System Design with NULL Convention Logic*, John Wiley & Sons, 2005.
79. C. Jeong and S.M. Nowick, "Optimal technology mapping and cell merger for asynchronous threshold networks," *Proc. 12th IEEE International Symp. on Asynchronous Circuits and Systems*, pp. 128-137, 2006.
80. K. Keutzer, "DAGON: technology binding and local optimization by DAG matching," *Proc. 24th ACM/IEEE Design Automation Conference*, pp. 341-347, 1987.
81. P. Siegel and G. De Micheli, "Decomposition methods for library binding of speed-independent asynchronous designs," *Proc. IEEE/ACM International Conf. on Computer-Aided Design*, pp. 558-565, 1994.
82. S.M. Burns, "General conditions for the decomposition of state-holding elements," *Proc. 2nd International Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp. 48-57, 1996.
83. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Trans. on Information and Systems*, vol. E80-D, no. 3, pp. 315-325, 1997.
84. K. van Berkel, F. Huberts and Ad Peeters, "Stretching quasi delay insensitivity by means of extended isochronic forks," *Proc. 2nd Working Conf. on Asynchronous Design Methodologies*, pp. 99, 1995.
85. K. Stevens, R. Ginosar and S. Rotem, "Relative timing," *IEEE Trans. on VLSI Systems*, vol. 11, no. 1, pp. 129-140, February 2003.
86. J. Cortadella, A. Kondratyev, L. Lavagno and C. Sotiriou, "Coping with the variability of combinational logic delays," *Proc. IEEE International Conf. on Computer Design*, pp. 505-508, 2004.
87. R.K. Brayton, "Factoring logic functions," *IBM Journal of Research and Development*, vol. 31, no. 2, pp. 187-198, March 1987.
88. B. Bose and D.J. Lin, "Systematic unidirectional error-detecting codes," *IEEE Trans. on*

Computers, vol. C-34, no. 11, pp. 1026-1032, November 1985.

89. A.J. Martin and M. Nystrom, “Asynchronous techniques for system-on-chip design,” *Proc. of the IEEE*, vol. 94, no. 6, pp. 1089-1120, June 2006.
90. J. Cortadella, *Private Communication*, May 2009.
91. A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen and A. Yakovlev, “Basic gate implementation of speed-independent circuits,” *Proc. 31st ACM/IEEE Design Automation Conference*, pp. 56-62, 1994.
92. A.J. Martin, “Programming in VLSI: from communicating processes to delay-insensitive circuits,” in *Developments in Concurrency and Computation*, C.A.R. Hoare (Ed.), pp. 1-64, Addison-Wesley, 1990.
93. T.-A. Chu, “Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications,” *PhD Thesis*, MIT Laboratory for Computer Science, June 1987.
94. P.M. Lewis II and C.L. Coates, *Threshold Logic*, Wiley, New York, 1967.
95. P.A. Beerel and T.H.-Y. Meng, “Automatic gate-level synthesis of speed-independent circuits,” *Proc. IEEE/ACM International Conf. on Computer-Aided Design*, pp. 581-586, 1992.
96. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, *Logic Synthesis of Asynchronous Controllers and Interfaces*, Springer-Verlag Publishing, 2002.
97. N.P. Singh, “A design methodology for self-timed systems,” *M.Sc. Thesis*, MIT Laboratory for Computer Science Technical Report TR-258, February 1981.
98. T.E. Williams, “Self-timed rings and their application to division,” *PhD Thesis*, Stanford University, May 1991.
99. A.J. Martin, “Asynchronous datapaths and the design of an asynchronous adder,” *Formal Methods in System Design*, vol. 1, no. 1, pp. 117-137, July 1992.
100. J. Sparso and J. Staunstrup, “Delay-insensitive multi-ring structures,” *Integration, the VLSI Journal*, vol. 15, pp. 313-340, 1993.
101. T.S. Anantharaman, “A delay insensitive regular expression recognizer,” *IEEE VLSI Technical Bulletin*, vol. 1, no. 2, pp. 3-15, 1986.
102. C.D. Nielsen, “Evaluation of function block designs,” *Technical Report* ID-TR:1994-135, Dept. of Computer Science, Technical University of Denmark, January 1994.

103. I. David, R. Ginosar and M. Yoeli, “An efficient implementation of Boolean functions as self-timed circuits,” *IEEE Trans. on Computers*, vol. 41, no. 1, pp. 2-11, January 1992.

104. W.B. Toms and D.A. Edwards, “Efficient synthesis of speed independent combinational logic circuits,” *Proc. 10th Asia and South Pacific Design Automation Conference*, pp. 1022-1026, 2005.

105. R.K. Brayton, G.D. Hachtel and A.L. Sangiovanni-Vincentelli, “Multilevel logic synthesis,” *Proc. of the IEEE*, vol. 78, no. 2, pp. 264-300, February 1990.

106. R. Rudell, “Logic Synthesis for VLSI Design,” *PhD Thesis*, University of California, Berkeley, 1989.

107. S. Devadas, A. Ghosh and K. Keutzer, *Logic Synthesis*, Mc-Graw Hill Series on Computer Engineering, Mc-Graw Hill Inc., 1994.

108. T. Stanion and C. Sechen, “Boolean division and factorization using binary decision diagrams,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 13, no. 9, pp. 1179-1184, September 1994.

109. R.K. Brayton and C.T. McMullen, “The decomposition and factorization of Boolean expressions,” *Proc. IEEE International Symp. on Circuits and Systems*, pp. 49-54, 1982.

110. B. Folco, V. Bregier, L. Fesquet and M. Renaudin, “Technology mapping for area optimized quasi delay insensitive circuits,” *Proc. International Conf. on Very Large Scale Integration*, pp. 146-151, 2005.

111. R.E. Bryant, “Graph-based algorithms for Boolean function manipulation,” *IEEE Trans. on Computers*, vol. C-35, no. 8, pp. 677-691, August 1986.

112. P. Maurine J.B. Rigaud, F. Bouesse, G. Sicard and M. Renaudin, “Static implementation of QDI asynchronous primitives,” *Proc. International Workshop on Power and Timing Modeling, Optimization and Simulation*, J.J. Chico and E. Macii (Eds.), *LNCS*, vol. 2799, pp. 181-191, 2003.

113. M. Zhao and S.S. Sapatnekar, “A new structural pattern matching algorithm for technology mapping,” *Proc. 38th Annual ACM/IEEE Design Automation Conference*, pp. 371-376, 2001.

114. S.B. Akers, “Binary Decision Diagrams,” *IEEE Trans. on Computers*, vol. C-27, no. 6, pp. 509-516, 1978.

115. C.Y. Lee, “Representation of switching circuits by binary-decision programs,” *Bell*

Systems Technical Journal, vol. 38, pp. 985-999, July 1959.

116. C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design*, Springer-Verlag Berlin Heidelberg, New York, 1998.

117. B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," *IEEE Trans. on Computers*, vol. 45, no. 9, pp. 993-1002, September 1996.

118. S. Minato, N. Ishiura and S. Yajima, "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation," *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 52-57, 1990.

119. T. Chelcea, G. Venkataramani and S.C. Goldstein, "Area optimizations for dual-rail circuits using relative-timing analysis," *Proc. 13th IEEE International Symp. on Asynchronous Circuits and Systems*, pp. 117-128, 2007.

120. C.F. Brej and J.D. Garside, "Early output logic using anti-tokens," *Proc. 12th International Workshop on Logic and Synthesis*, pp. 302-309, 2003.

121. Y. Zhou, D. Sokolov and A. Yakovlev, "Cost-aware synthesis of asynchronous circuits based on partial acknowledgement," *Proc. IEEE/ACM International Conf. on Computer-Aided Design*, pp. 158-163, 2006.

122. C. Jeong and S.M. Nowick, "Optimization of robust asynchronous circuits by local input completeness relaxation," *Proc. Asia and South Pacific Design Automation Conference*, pp. 622-627, 2007.

123. R. Murgai, R.K. Brayton and A.L. Sangiovanni-Vincentelli, *Logic Synthesis for Field-Programmable Gate Arrays*, Kluwer Academic Publishers, Norwell, MA, 1995.

124. G.E. Sobelman and K. Fant, "CMOS circuit design of threshold gates with hysteresis," *Proc. IEEE International Symp. on Circuits and Systems*, vol. 2, pp. 61-64, 1998.

125. S.C. Smith, R.F. DeMara, J.S. Yuan, D. Ferguson and D. Lamb, "Optimization of NULL convention self-timed circuits," *Integration, the VLSI Journal*, vol. 37, pp. 135-165, 2004.

126. M. Ligthart, K. Fant, R. Smith, A. Taubin and A. Kondratyev, "Asynchronous design using commercial HDL synthesis tools," *Proc. 6th International Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp. 114-125, 2000.

127. A. Kondratyev and K. Lwin, "Design of asynchronous circuits by synchronous CAD tools," *IEEE Design and Test of Computers*, vol. 19, no. 4, pp. 107-117, July-August 2002.

128. C. Jeong and S.M. Nowick, "Block-level relaxation for timing-robust asynchronous circuits based on eager evaluation," *Proc. 14th IEEE International Symp. on Asynchronous Circuits and Systems*, pp. 95-104, 2008.

129. I. Newman, "On read-once Boolean functions," in M.S. Paterson (Ed.), *Boolean Function Complexity*, pp. 25-34, London Mathematical Society Lecture Note Series 169, Cambridge University Press, 1992.

130. R.K. Brayton, G.D. Hachtel, C.T. McMullen and A.L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, MA, 1984.

131. T. Sasao (Ed.), *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, Dordrect, The Netherlands, 1999.

132. B.J. Falkowski, "Calculation of Rademacher-Walsh spectral coefficients for systems of completely and incompletely specified Boolean functions," *Proc. IEEE International Symp. on Circuits and Systems*, vol. 3, pp. 1698-1701, 1993.

133. B.J. Falkowski and C.-H. Chang, "Paired Haar spectra computation through operations on disjoint cubes," *IEE Proc. Circuits, Devices and Systems*, vol. 146, no. 3, pp. 117-123, August 1999.

134. M.A. Thornton, R. Drechsler and D.M. Miller, *Spectral Techniques in VLSI CAD*, Kluwer Academic Publishers, Boston, MA, 2001.

135. T. Sasao, "EXMIN2: a simplification algorithm for exclusive-OR-sum-of-products expressions for multiple-valued-input two-valued-output functions," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 12, no. 5, pp. 621-632, May 1993.

136. A. Mishchenko and M. Perkowski, "Fast heuristic minimization of exclusive-sums-of-products," *Proc. International Workshop on Applications of Reed-Muller Expansion in Circuit Design*, pp. 242-250, 2001.

137. D. Maslov, "Efficient reversible and quantum implementations of symmetric Boolean functions," *IEE Proc. Circuits, Devices and Systems*, vol. 153, no. 5, pp. 467-472, October 2006.

138. P. Gupta, A. Agrawal and N.K. Jha, "An algorithm for synthesis of reversible logic circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2317-2330, November 2006.

139. B.J. Falkowski, I. Schafer and C.-H. Chang, "An effective computer algorithm for the

calculation of disjoint cube representation of Boolean functions," *Proc. 36th IEEE Midwest Symp. on Circuits and Systems*, pp. 1308-1311, 1993.

140. T. Kozlowski, "Application of Exclusive-OR Logic in Technology Independent Logic Optimization," *PhD Thesis*, University of Bristol, January 1996.

141. L. Shivakumaraiah and M.A. Thornton, "Computation of disjoint cube representations using a maximal binate variable heuristic," *Proc. 34th IEEE Southeastern Symposium on System Theory*, pp. 417-421, 2002.

142. G. Fey and R. Drechsler, "Utilizing BDDs for disjoint SOP minimization," *Proc. 45th IEEE Midwest Symp. on Circuits and Systems*, vol. 2, pp. II-306- II-309, 2002.

143. G. Fey and R. Drechsler, "A hybrid approach combining symbolic and structural techniques for disjoint SOP minimization," *Proc. 11th Workshop on Synthesis And System Integration of Mixed Information technologies*, pp. 54-60, 2003.

144. N. Drechsler, M. Hilgemeier, G. Fey and R. Drechsler, "Disjoint sum of product minimization by evolutionary algorithms," in *EvoWorkshops* 2004, G.R. Raidl et al. (Eds.), *Lecture Notes in Computer Science*, vol. 3005, pp. 198-207, 2004.

145. M. Fujita, H. Fujisawa and Y. Matsunaga, "Variable ordering algorithms for ordered binary decision diagrams and their evaluation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 12, no. 1, pp. 6-12, January 1993.

146. R. Drechsler and W. Gunther, "History-based dynamic BDD minimization," *Integration, the VLSI Journal*, vol. 31, no. 1, pp. 51-63, November 2001.

147. S. Yang, "Logic synthesis and optimization benchmarks user guide: version 3.0," *distributed as part of the MCNC International Workshop on Logic Synthesis*, 1991.

148. K. McElvain, "LGSynth93 benchmark set: version 4.0," *distributed as part of the MCNC International Workshop on Logic Synthesis*, 1993.

149. W.B. Toms and D.A. Edwards, "Prime Indicants: a synthesis method for indicating combinational logic blocks," *Proc. 15th IEEE International Symp. on Asynchronous Circuits and Systems*, pp. 139-150, 2009.

150. F. Brglez and H. Fujiwara, "A neural netlist of 10 combinational benchmark circuits and a target translator in Fortran," *Proc. IEEE International Symp. on Circuits and Systems*, 1985.

151. W.B. Toms, "Synthesis of Quasi-Delay-Insensitive Datapath Circuits," *PhD Thesis*,

University of Manchester, 2006.

152. P. Balasubramanian and D.A. Edwards, "A new design technique for weakly indicating function blocks," *Proc. 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pp. 116-121, 2008.

153. *PrimeTime User Guide: Advanced Timing Analysis*, Version A-2007.12, December 2007.

154. J.D. Garside, "A CMOS VLSI implementation of an asynchronous ALU," *Proc. IFIP Working Conf. on Asynchronous Design Methodologies*, pp. 181-192, 1993.

155. J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 1990.

156. D.C. Chen, L.M. Guerra, E.H. Ng, M. Potkonjak, D.P. Schultz and J.M. Rabaey, "An integrated system for rapid prototyping of high performance algorithm specific data paths," *Proc. IEEE Conf. on Application Specific Array Processors*, pp. 134-138, 1992.

157. S. Winograd, "On the time required to perform addition," *Journal of the ACM*, vol. 12, no. 2, pp. 277-285, April 1965.

158. Y. Liu, "Power-Efficient Embedded Processing," *PhD Thesis*, University of Manchester, 2005.

159. C. Nagendra, M.J. Irwin and R.M. Owens, "Area-time-power tradeoffs in parallel adders," *IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 43, no. 10, pp. 689-702, October 1996.

160. O.J. Bedrij, "Carry-select adder," *IRE Trans. on Electronic Computers*, vol. EC-11, pp. 340-346, 1962.

161. J. Sklansky, "Conditional-sum addition logic," *IRE Trans. on Electronic Computers*, vol. EC-9, no. 2, pp. 226-231, June 1960.

162. A.R. Omondi, *Computer Arithmetic Systems: Algorithms, Architecture and Implementations*, Prentice Hall International (UK) Ltd., 1994.

163. P.M. Kogge and H.S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. on Computers*, vol. C-22, no. 8, pp. 786-793, August 1973.

164. R.E. Ladner and M.J. Fischer, "Parallel prefix computation," *Journal of the ACM,* vol. 27, no. 4, pp. 831-838, October 1980.

165. R.P. Brent and H.T. Kung, "A regular layout for parallel adders," *IEEE Trans. on Computers*, vol. C-31, no. 3, pp. 260-264, March 1982.

166. T. Han and D.A. Carlson, "Fast area-efficient VLSI adders," *Proc. 8th IEEE Symp. on Computer Arithmetic*, pp. 49-56, 1987.

167. K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*, John Wiley & Sons Inc., New York, 1979.

168. B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, New York, 2000.

169. C.S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. on Electronic Computers*, vol. EC-13, no. 1, pp. 14-17, February 1964.

170. W. Waser and M.J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*, Oxford University Press, New York, 1985.

171. P. Reusens, W.H. Ku and Y.H. Mao, "Fixed-point high-speed parallel multipliers in VLSI," in *VLSI Systems and Computations*, H.T. Kung et al. (Eds.), pp. 301-310, Springer-Verlag, New York, 1981.

172. L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, no. 5, pp. 349-356, March 1965.

173. W.J. Townsend, E.E. Swartzlander Jr. and J.A. Abraham, "A comparison of Dadda and Wallace multiplier delays," *Proc. SPIE Advanced Signal Processing Algorithms, Architectures and Implementations XIII*, Franklin T. Luk (Ed.), vol. 5205, pp. 552-560, 2003.

174. Z.-J. Mou and F. Jutand, "Overturned-stairs adder trees and multiplier design," *IEEE Trans. on Computers*, vol. C-41, no. 8, pp. 940-948, August 1992.

175. D. Zuras and W.H. McAllister, "Balanced delay trees and combinational division in VLSI," *IEEE Journal of Solid-State Circuits*, vol. SC-21, no. 5, pp. 814-819, October 1986.

176. Mi Lu, *Arithmetic and Logic in Computer Systems*, John Wiley & Sons, Inc., NJ, 2004.

177. A. Weinberger, "4:2 carry-save adder module," *IBM Technical Disclosure Bulletin*, vol. 23, January 1981.

178. I. Koren, *Computer Arithmetic Algorithms*, Prentice-Hall International (UK), London, 1993.

179. K. Prasad and K.K. Parhi, “Low-power 4-2 and 5-2 compressors,” *Proc. 35th Asilomar Conf. on Signals, Systems and Computers*, vol. 1, pp. 129-133, 2001.

180. P. Balasubramanian and D.A. Edwards, “A delay efficient robust self-timed full adder,” *Proc. 3rd IEEE International Design and Test Workshop*, pp. 129-134, 2008.

181. B. Gilchrist, J.H. Pomerene and S.Y. Wong, “Fast carry logic for digital computers,” *IRE Trans. on Electronic Computers*, vol. 4, no. 4, pp. 133-136, December 1955.

182. B.E. Briley, “Asynchronous adder circuit,” US Patent 4338676, June 1982.

183. M. Renaudin and B. El Hassan, “The design of fast asynchronous adder structures and their implementation using DCVS logic,” *Proc. IEEE International Symp. on Circuits and Systems*, vol. 4, pp. 291-294, 1994.

184. D.J. Kinniment, J.D. Garside and B. Gao, “A comparison of power consumption in some CMOS adder circuits,” *Proc. International Workshop on Power and Timing Modeling Optimization and Simulation*, 1995.

185. S.M. Nowick, K.Y. Yun, A.E. Dooply and P.A. Beerel, “Speculative completion for the design of high-performance asynchronous dynamic adders,” *Proc. 3rd International Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp. 210, 1997.

186. D. Johnson and V. Akella, “Design and analysis of asynchronous adders,” *IEE Proc. Computers and Digital Techniques*, vol. 145, no. 1, pp. 1-8, 1998.

187. P. Corsonello, S. Perri and G. Cocorullo, “A 56-bit self-timed adder for high speed asynchronous datapath,” *Proc. 6th IEEE International Conf. on Electronics, Circuits and Systems*, vol. 1, pp. 37-41, 1999.

188. A. De Gloria and M. Olivieri, “Completion-detecting carry select addition,” *IEE Proc. Computers and Digital Techniques*, vol. 147, no. 2, pp. 93-100, March 2000.

189. G.A. Ruiz and M.A. Manzano, “Compact 32-bit CMOS adder in multi-output DCVS logic for self-timed circuits,” *IEE Proc. Circuits, Devices and Systems*, vol. 147, no. 3, pp. 183-188, June 2000.

190. A. Amin, F. Maadi, “Double-rail encoded self-timed adder with matched delays,” *Proc. 10th IEEE International Conf. on Electronics, Circuits and Systems*, vol. 3, pp. 1172-1175, 2003.

191. I. Obridko and R. Ginosar, “Low energy asynchronous adders,” *Proc. 11th IEEE International Conf. on Electronics, Circuits and Systems*, pp. 164-167, 2004.

192. Y. Liu and S. Furber, “The design of an asynchronous carry-lookahead adder based on data characteristics,” *Lecture Notes in Computer Science,* vol. 3728/2005, V. Paliouras, J. Vounckx and D. Verkest (Eds.), pp. 647-656, 2005.

Appendix

Datapaths Employing 1-of-*n* Codes with Similar Symbolic Variable Assignments

This section describes a mechanism to identify mutually orthogonal cubes and states the relations to be satisfied for SI decomposition while dealing with datapaths that employ any combination of arbitrary one-hot codes and opting for a similar symbolic variable assignment. As stated in section 4.1, the following discussion is relevant in the context of general multi-level synthesis models for indicating realisations of arbitrary combinational logic functions.

We shall consider an example using the 1-of-4 code for the purpose of illustration. As mentioned in Chapter 2, two single-rail inputs can be represented using a 1-of-4 code symbolically. Let us assume that a function F is dependent upon 6 input variables (*a*,*b*,*c*,*d*,*e*,*f*) and expressed by the disjunction of two cubes X and Y, specified by *a'bcd'e'f* and *a'b'c'd'ef*. The pairs of single-rail input variables (*a*,*b*), (*c*,*d*) and (*e*,*f*) are now represented by code group sets, say *g*1, *g*2 and *g*3, where *g*1, *g*2 and *g*3 are specified by {*i*0,*i*1,*i*2,*i*3}, {*i*4,*i*5,*i*6,*i*7} and {*i*8,*i*9,*i*10,*i*11} respectively. In general, to represent a group of *m* single-rail inputs using a 1-of-*n* code, the corresponding code group set of the one-hot representation would require 2^m distinct elements, where $n = 2^m$. Assuming a similar encoding assignment as shown in Table 2.1, we have X = *i*2*i*5*i*10 and Y = *i*3*i*7*i*8.

From the one-hot code representation of cubes X and Y, it is apparent that enumerating their support sets would not convey any useful meaning as '*i*' serves as the common symbolic variable index. Therefore, the notion of dependency set would be used extensively. In this context, the basic criterion to be satisfied by two cubes P_1 and P_2, which may be mutually orthogonal is given by the following:

$$|D(P_1)| \geq 1, \; |D(P_2)| \geq 1 \qquad \textbf{(A.1)}$$

This equation conveys that P_1 and P_2 consist of at least a single literal corresponding to a code group set. Instead of ascertaining the MO set (that corresponds to the variables of the CR set), the orthogonality relation between two cubes is established based on the value of

DMO. DMO is determined in a different way for 1-of-*n* codes assuming similar symbolic variable assignments with variables identification being replaced by the notion of literals identification (LI), corresponding to each code group. LI is an operation performed on all the CR sets addressing each code group individually and yields a single literal that belongs to a unique code group set. For example, with respect to cubes X and Y, we get the following:

$$\text{CR\_LI\_}g1\ [\text{X, Y}] = i2,\ \text{CR\_LI\_}g2\ [\text{X, Y}] = i5,\ \text{CR\_LI\_}g3\ [\text{X, Y}] = i10 \qquad \textbf{(A.2)}$$

$$\text{CR\_LI\_}g1\ [\text{Y, X}] = i3,\ \text{CR\_LI\_}g2\ [\text{Y, X}] = i7,\ \text{CR\_LI\_}g3\ [\text{Y, X}] = i8 \qquad \textbf{(A.3)}$$

Hence, in addition to (A.1), the conditions to be satisfied such that two random cubes P_1 and P_2 adopting any higher-order 1-of-*n* encoding protocol can be dubbed as mutually orthogonal are given below, where *k* can refer to any unique code group.

$$\text{CR\_LI\_}gk\ [P_1, P_2] \neq \varnothing,\ \text{CR\_LI\_}gk\ [P_2, P_1] \neq \varnothing \qquad \textbf{(A.4)}$$

$$\text{CR\_LI\_}gk\ [P_1, P_2] \cap \text{CR\_LI\_}gk\ [P_2, P_1] = \varnothing \qquad \textbf{(A.5)}$$

(A.4) and (A.5) essentially mean $|\text{CR\_LI\_}gk\ [P_1, P_2]| = |\text{CR\_LI\_}gk\ [P_2, P_1]| = 1$, for any *k*. In simple terms, considering a specific code group set, there should be a unique literal in P_1 relative to P_2 and vice-versa. DMO is basically an integer count of the number of times that (A.4) and (A.5) gets satisfied between the pair of cubes considering the entire distinct code group sets. It is imperative that DMO $\geq$ 1 be upheld between two cubes exhibiting mutual orthogonality. DMO = 0 implies that both the cubes are identical (in case of equipollent cubes). In general, it could hint at the possibility of two cubes constituting an output cover function to become activated for an input combination, thereby violating the monotonic cover constraint. Referring back to (A.2) and (A.3), it can be concluded that X is orthogonal to Y.

The conditions that may be deemed sufficient to allow SI decomposition of two mutually orthogonal cubes P_1 and P_2 (i.e. the pair of cubes satisfying (A.4) and (A.5)), based on arbitrary 1-of-*n* codes, are given below:

$$|D(P_1)| > 1,\ |D(P_2)| > 1 \qquad \textbf{(A.6)}$$

$$|D(P_1)| = |D(P_2)| \qquad \textbf{(A.7)}$$

$$|\text{CDI}\ [D(P_1), D(P_2)]| = |D(P_1)|\text{-}1 = |D(P_2)|\text{-}1 \qquad \textbf{(A.8)}$$

(A.6) implies that there should be at least two literals in both P_1 and P_2 in order that an opportunity for SI decomposition can be explored. (A.7) mandates that the dimension of both the cubes should be equal. Lastly, (A.8) conveys that CDI [D(P_1), D(P_2)] $\subseteq$ D(P_1) and CDI [D(P_1), D(P_2)] $\subseteq$ D(P_2), signifying that P_1 and P_2 differ only with respect to a single code group set element.