Asynchronous Circuit Design. Chris J. Myers
Copyright © 2001 by John Wiley & Sons, Inc.
ISBNs: 0-471-41543-X (Hardback); 0-471-22414-6 (Electronic)

Asynchronous Circuit Design

Asynchronous Circuit Design. Chris J. Myers
Copyright © 2001 by John Wiley & Sons, Inc.
ISBNs: 0-471-41543-X (Hardback); 0-471-22414-6 (Electronic)

Asynchronous Circuit Design

Chris J. Myers

A Wiley-Interscience Publication
JOHN WILEY & SONS, INC.
New York / Chichester / Weinheim / Brisbane / Singapore / Toronto

ISBN 0-471-22414-6.

This title is also available in print as ISBN 0-471-41543-X.

To Ching and John

Asynchronous Circuit Design. Chris J. Myers
Copyright © 2001 by John Wiley & Sons, Inc.
ISBNs: 0-471-41543-X (Hardback); 0-471-22414-6 (Electronic)

Contents

Preface xiii

Acknowledgments xvii

1 Introduction 1
1.1 Problem Specification 1
1.2 Communication Channels 2
1.3 Communication Protocols 4
1.4 Graphical Representations 8
1.5 Delay-Insensitive Circuits 10
1.6 Huffman Circuits 13
1.7 Muller Circuits 16
1.8 Timed Circuits 17
1.9 Verification 20
1.10 Applications 20
1.11 Let's Get Started 21
1.12 Sources 21
Problems 22

2 Communication Channels 23
2.1 Basic Structure 24

2.2 Structural Modeling in VHDL 27
2.3 Control Structures 31
2.3.1 Selection 31
2.3.2 Repetition 32
2.4 Deadlock 34
2.5 Probe 35
2.6 Parallel Communication 35
2.7 Example: MiniMIPS 36
2.7.1 VHDL Specification 38
2.7.2 Optimized MiniMIPS 48
2.8 Sources 52
Problems 53

3 Communication Protocols 57
3.1 Basic Structure 57
3.2 Active and Passive Ports 61
3.3 Handshaking Expansion 61
3.4 Reshuffling 65
3.5 State Variable Insertion 66
3.6 Data Encoding 67
3.7 Example: Two Wine Shops 71
3.8 Syntax-Directed Translation 73
3.9 Sources 80
Problems 82

4 Graphical Representations 85
4.1 Graph Basics 85
4.2 Asynchronous Finite State Machines 88
4.2.1 Finite State Machines and Flow Tables 88
4.2.2 Burst-Mode State Machines 91
4.2.3 Extended Burst-Mode State Machines 93
4.3 Petri Nets 100
4.3.1 Ordinary Petri Nets 100
4.3.2 Signal Transition Graphs 111
4.4 Timed Event/Level Structures 116
4.5 Sources 120
Problems 121

5 *Huffman Circuits* 131

5.1 *Solving Covering Problems* 132

5.1.1 *Matrix Reduction Techniques* 134

5.1.2 *Bounding* 137

5.1.3 *Termination* 137

5.1.4 *Branching* 138

5.2 *State Minimization* 140

5.2.1 *Finding the Compatible Pairs* 141

5.2.2 *Finding the Maximal Compatibles* 143

5.2.3 *Finding the Prime Compatibles* 145

5.2.4 *Setting Up the Covering Problem* 148

5.2.5 *Forming the Reduced Flow Table* 154

5.3 *State Assignment* 154

5.3.1 *Partition Theory and State Assignment* 155

5.3.2 *Matrix Reduction Method* 157

5.3.3 *Finding the Maximal Intersectibles* 158

5.3.4 *Setting Up the Covering Problem* 161

5.3.5 *Fed-Back Outputs as State Variables* 163

5.4 *Hazard-Free Two-Level Logic Synthesis* 165

5.4.1 *Two-Level Logic Minimization* 165

5.4.2 *Prime Implicant Generation* 166

5.4.3 *Prime Implicant Selection* 168

5.4.4 *Combinational Hazards* 169

5.5 *Extensions for MIC Operation* 171

5.5.1 *Transition Cubes* 172

5.5.2 *Function Hazards* 172

5.5.3 *Combinational Hazards* 173

5.5.4 *Burst-Mode Transitions* 176

5.5.5 *Extended Burst-Mode Transitions* 177

5.5.6 *State Minimization* 180

5.5.7 *State Assignment* 183

5.5.8 *Hazard-Free Two-Level Logic Synthesis* 183

5.6 *Multilevel Logic Synthesis* 188

5.7 *Technology Mapping* 189

5.8 *Generalized C-Element Implementation* 193

5.9 *Sequential Hazards* 194

5.10 *Sources* 196

Problems 199

6 Muller Circuits 207
6.1 Formal Definition of Speed Independence 208
6.1.1 Subclasses of Speed-Independent Circuits 210
6.1.2 Some Useful Definitions 212
6.2 Complete State Coding 216
6.2.1 Transition Points and Insertion Points 217
6.2.2 State Graph Coloring 219
6.2.3 Insertion Point Cost Function 220
6.2.4 State Signal Insertion 222
6.2.5 Algorithm for Solving CSC Violations 223
6.3 Hazard-Free Logic Synthesis 223
6.3.1 Atomic Gate Implementation 225
6.3.2 Generalized C-Element Implementation 226
6.3.3 Standard C-Implementation 230
6.3.4 The Single-Cube Algorithm 238
6.4 ·Hazard-Free Decomposition 243
6.4.1 Insertion Points Revisited 245
6.4.2 Algorithm for Hazard-Free Decomposition 246
6.5 Limitations of Speed-Independent Design 248
6.6 Sources 249
Problems 251

7 Timed Circuits 259
7.1 Modeling Timing 260
7.2 Regions 262
7.3 Discrete time 265
7.4 Zones 267
7.5 POSET Timing 280
7.6 Timed Circuits 289
7.7 Sources 292
Problems 293

8 Verification 295
8.1 Protocol Verification 296
8.1.1 Linear-Time Temporal Logic 296
8.1.2 Time-Quantified Requirements 300
8.2 Circuit Verification 303
8.2.1 Trace Structures 303

8.2.2 Composition 305
8.2.3 Canonical Trace Structures 308
8.2.4 Mirrors and Verification 310
8.2.5 Strong Conformance 312
8.2.6 Timed Trace Theory 314
8.3 Sources 315
Problems 316

9 Applications 321
9.1 Brief History of Asynchronous Circuit Design 322
9.2 An Asynchronous Instruction-Length Decoder 325
9.3 Performance Analysis 329
9.4 Testing Asynchronous Circuits 330
9.5 The Synchronization Problem 332
9.5.1 Probability of Synchronization Failure 334
9.5.2 Reducing the Probability of Failure 335
9.5.3 Eliminating the Probability of Failure 336
9.5.4 Arbitration 340
9.6 The Future of Asynchronous Circuit Design 341
9.7 Sources 342
Problems 346

Appendix A VHDL Packages 347
A.1 nondeterminism.vhd 347
A.2 channel.vhd 348
A.3 handshake.vhd 355

Appendix B Sets and Relations 359
B.1 Basic Set Theory 360
B.2 Relations 362

References 365

Index 393

Preface

An important scientific innovation rarely makes its way by gradually winning over and converting its opponents: it rarely happens that Saul becomes Paul. What does happen is that its opponents gradually die out and that the growing generation is familiarized with the idea from the beginning.

—Max Planck

I must govern the clock, not be governed by it.

—Golda Meir

All pain disappears, it's the nature of my circuitry.

—nine inch nails

In 1969, Stephen Unger published his classic textbook on asynchronous circuit design. This book presented a comprehensive look at the asynchronous design methods of the time. In the 30 years hence, there have been numerous technical publications and even a few books [37, 57, 120, 203, 224, 267, 363, 393], but there has not been another textbook. This book attempts to fill this void by providing an updated look at asynchronous circuit design in a form accessible to a student who simply has some background in digital logic design.

An asynchronous circuit is one in which synchronization is performed without a global clock. Asynchronous circuits have several advantages over their synchronous counterparts, including:

1. *Elimination of clock skew problems.* As systems become larger, increasing amounts of design effort is necessary to guarantee minimal skew in the arrival time of the clock signal at different parts of the chip. In an asynchronous circuit, skew in synchronization signals can be tolerated.

2. *Average-case performance.* In synchronous systems, the performance is dictated by worst-case conditions. The clock period must be set to be long enough to accommodate the slowest operation even though the average delay of the operation is often much shorter. In asynchronous circuits, the speed of the circuit is allowed to change dynamically, so the performance is governed by the average-case delay.

3. *Adaptivity to processing and environmental variations.* The delay of a VLSI circuit can vary significantly over different processing runs, supply voltages, and operating temperatures. Synchronous designs have their clock rate set to allow correct operation under some allowed variations. Due to their adaptive nature, asynchronous circuits operate correctly under all variations and simply speed up or slow down as necessary.

4. *Component modularity and reuse.* In an asynchronous system, components can be interfaced without the difficulties associated with synchronizing clocks in a synchronous system.

5. *Lower system power requirements.* Asynchronous circuits reduce synchronization power by not requiring additional clock drivers and buffers to limit clock skew. They also automatically power-down unused components. Finally, asynchronous circuits do not waste power due to spurious transitions.

6. *Reduced noise.* In a synchronous design, all activity is locked into a very precise frequency. The result is nearly all the energy is concentrated in very narrow spectral bands at the clock frequency and its harmonics. Therefore, there is substantial electrical noise at these frequencies. Activity in an asynchronous circuit is uncorrelated, resulting in a more distributed noise spectrum and a lower peak noise value.

Despite all these potential advantages, asynchronous design has seen limited usage to date. Although there are many reasons for this, perhaps the most serious is a lack of designers with experience in asynchronous design. This textbook is a direct attempt at addressing this problem by providing a means for graduate or even undergraduate courses to be created that teach modern asynchronous design methods. I have used it in a course which includes both undergraduates and graduates. Lectures and other material used in this and future courses will be made available on our Web site: `http://www.async.elen.utah.edu/book/`. This book may also be used for self-study by engineers who would like to learn about modern asynchronous design methods. Each chapter includes numerous problems for the student to try out his or her new skills.

The history of asynchronous design is quite long. Asynchronous design methods date back to the 1950s and to two people in particular: Huffman and Muller. Every asynchronous design methodology owes its roots to one of these two men. Huffman developed a design methodology for what is known today as *fundamental-mode* circuits [170]. Muller developed the theoretical underpinnings of *speed-independent* circuits [279]. Unger is a member of the "Huffman School," so his textbook focused primarily on fundamental-mode circuit design with only a brief treatment of Muller circuits. Although I am a student of the "Muller School," in this book we present both design methods with the hope that members of both schools will grow to understand each other better, perhaps even realizing that the differences are not that great.

Since the early days, asynchronous circuits have been used in many interesting applications. In the 1950s and 1960s at the University of Illinois, Muller and his colleagues used speed-independent circuits in the design of the ILLIAC and ILLIAC II computers [46]. In the early days, asynchronous design was also used in the MU-5 and Atlas mainframe computers. In the 1970s at Washington University in St. Louis, asynchronous macromodules were developed [87]. These modules could be plugged together to create numerous special-purpose computing engines. Also in the 1970s, asynchronous techniques were used at the University of Utah in the design of the first operational dataflow computer [102, 103] and at Evans and Sutherland in design of the first commercial graphics system.

Due to the advantages cited above, there has been a resurgence of interest in asynchronous design. There have been several recent successful design projects. In 1989, researchers at Caltech designed the first fully asynchronous microprocessor [251, 257, 258]. Since that time, numerous other researchers have produced asynchronous microprocessors of increasing complexity [10, 13, 76, 134, 135, 138, 191, 259, 288, 291, 324, 379, 406]. Commercially, asynchronous circuits have had some recent success. Myranet uses asynchronous circuits coupled with *pipeline synchronization* [348] in their router design. Philips has designed numerous asynchronous designs targeting low power [38, 136, 192, 193]. Perhaps the most notable accomplishment to come out of this group is an asynchronous 80C51 microcontroller, which is now used in a fully asynchronous pager being sold by Philips. Finally, the RAPPID project at Intel demonstrated that a fully asynchronous instruction-length decoder for the x86 instruction set could achieve a threefold improvement in speed and a twofold improvement in power compared with the existing synchronous design [141, 142, 143, 144, 330, 367].

In the time of Unger's text, there were perhaps only a handful of publications each year on asynchronous design. As shown in Figure 0.1, this rate of publication continued until about 1985, when there was a resurgence of interest in asynchronous circuit design [309]. Since 1985, the publication rate has grown to well over 100 technical publications per year. Therefore,

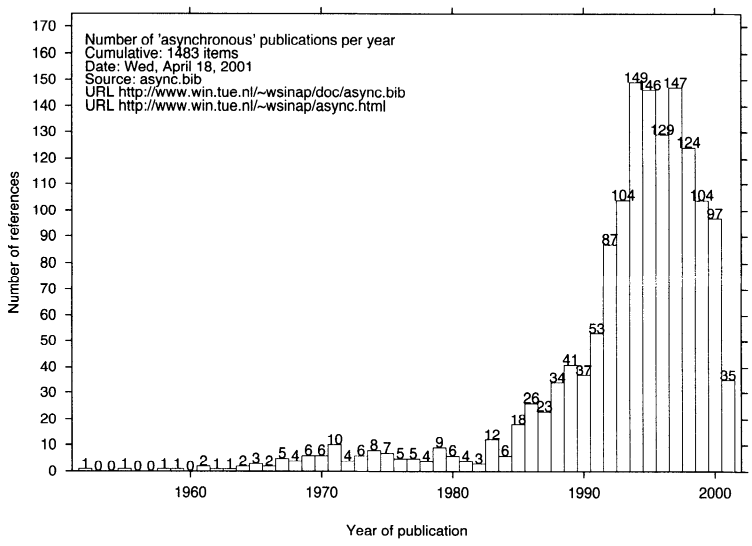

Fig. 0.1 Number of asynchronous publications per year.

although Unger did a superb job of surveying the field, this author has his work cut out for him. In the sources section at the end of each chapter, the interested reader is pointed to an extensive bibliography (over 400 entries) to probe deeper. Although an attempt has been made to give a flavor of the major design methodologies being developed and used, it is impossible even to reference every paper published on asynchronous design, as the number of entries in the asynchronous bibliography [309] now exceeds 1400. The interested reader should consult this bibliography and the proceedings from the recent symposiums on asynchronous circuits and systems [14, 15, 16, 17, 18, 19, 20].

The book is organized as follows. In Chapter 1 we introduce the asynchronous design problem through a small example illustrating the differences among the various timing models used. In Chapter 2 we introduce the concept of asynchronous communication and describe a methodology for specifying asynchronous designs using VHDL. In Chapter 3 we discuss various asynchronous protocols. In Chapter 4 we introduce graphical representations that are used for asynchronous design. In Chapter 5 we discuss Huffman circuits and in Chapter 6 we describe Muller circuits. In Chapter 7 we develop techniques for timing analysis and optimization which can lead to significant improvements in circuit quality. In Chapter 8 we introduce methods for the analysis and verification of asynchronous circuits. Finally, in Chapter 9 we give a brief discussion of issues in asynchronous application.

Chris J. Myers

Salt Lake City, Utah

Acknowledgments

I am indebted to Alain Martin and Chuck Seitz of Caltech, who turned me onto asynchronous design as an undergraduate. I would also like to thank my graduate advisors, Teresa Meng and David Dill of Stanford University, who taught me alternative ways of looking at asynchronous design. My former officemate, Peter Beerel (USC), through numerous heated discussions throughout the years, has taught me much.

I would like to thank Erik Brunvand (Utah), Steve Nowick (Columbia), Peter Beerel (USC), Wendy Belluomini (IBM), Ganesh Gopalakrishnan (Utah), Ken Stevens (Intel), Charles Dike (Intel), Jim Frenzel (U. of Idaho), Steven Unger (Columbia), Dong-Ik Lee (KJIST), and Tomohiro Yoneda (Titech) for their comments and advice on earlier versions of this manuscript. I'm also grateful to the comments and ideas that I received from my graduate students: Brandon Bachman, Jie Dai, Hans Jacobson, Kip Killpack, Chris Krieger, Scott Little, Eric Mercer, Curt Nelson, Eric Peskin, Robert Thacker, and Hao Zheng. I would like to thank the students in my course on asynchronous circuit design in the spring of 2000 for putting up with the rough version of this text. I am grateful to Sanjin Piragic for drawing many of the figures in the book. Many other figures are due to `draw_astg` by Jordi Cortadella (UPC) and `dot` by Eleftherios Koutsofios and Stephen North (AT&T).

I would like especially to thank my family, Ching and John, for being patient with me while I wrote this book. Without their love and support, the book would not have been possible.

C.J.M.

Asynchronous
Circuit Design

Index

A

Absolute complement, 361
Absorbed, 167
Absorption, 188, 362
Acknowledge, 4, 57, 61
Actions, 116
Active, 6, 61
Adjacent, 87
Advance time, 271, 314
Adverse example, 280
AFSM, 89
    See also asynchronous finite state machine
Aliases, 42
Allowed sequences, 208
    See also trace
Always operator, 297
  timed, 303
AMULET, 323
Antisymmetric, 360, 363
Arbiter, 340
Arbitration, 340
    See also synchronization
Arc, 86
Architecture, 25
Assert statement, 59
Assign, 4, 59, 63
Associative, 188, 362
Asymmetric
  choice nets, 108
  confusion, 107
Asynchronous
  circuit design
    future?, 341
    history, 322
  finite state machines, 8, 88
  handshake, 2
  instruction-length decoder, 325
    See also RAPPID
  microprocessors, 323
    AMULET, 323
    Caltech, 323
  pager, 324
  timing, 2
Atomic gate, 225
  implementation, 225
Attribute, 27
Autofailure, 309
  manifestation, 308
Average-case
  logic optimization, 327
  performance analysis, 329

B

Base functions, 190
BCP algorithm, 134
    See also covering problem

bounding, 137
branching, 138
example, 149
reduction, 135
termination, 137
Binary
decision diagram, 198
relations, 362
Binate covering problem, 133
Bipartite graph, 88
Blocks, 155
BM machines, 92
See also burst-mode
edge-labeling functions, 92
maximal set property, 92
state diagram, 92
unique entry point, 92
Boolean
matching, 191
matrix, 157
Bounded
delay, 14
gate and wire delay, 132
response time, 300
timing constraint, 118, 260
Bounding, 137
Branching, 138
Bridging fault model, 332
Buffer, 58
Bundled data, 61
Bundling constraint, 62
Burst-mode, 196
See also BM machines
generalized C-elements, 193
hazard-free logic synthesis, 183
hazards, 176
multilevel logic synthesis, 188
sequential hazards, 194
state assignment, 183
state machines, 91
state minimization, 180
synthesis, 171
technology mapping, 189
transition, 176

C

CALL module, 74
Caltech asynchronous microprocessor, 323
Candidate implicant, 233
Canonical
DBM, 268
trace structure, 310
Case statements, 31
Causal, 281
Chain, 363
Change diagrams, 111, 121
Channel, 3, 23
active, 61
init_channel, 25, 28
package, 25, 348
passive, 61
port, 61
probe, 35
receive, 26
parallel, 36
send, 26
parallel, 35
type, 25
VHDL modeling, 24
Characteristic marking, 102
Choke, 310
Circuit verification, 303
Class set, 145
Clause, 133
Closed, 148
cover, 148
system, 25
Closure, 145
clause, 235
constraint, 148
section, 235
Column dominance, 136
Combinational
hazards
burst-mode, 176
dynamic 0 → 1, 170
dynamic 1 → 0, 170
extended burst-mode, 177
multiple-input change, 173
single-input change, 169
static 0-hazard, 169
static 1-hazard, 169
optimization, 228, 236
Communicating sequential processes, 21
Communication
channels, 3
See also channel
basic structure, 24
protocols, 4
See also handshake
basic structure, 58
Communitive, 362
Compatibility table, 141
Compatible
conditionally, 142
list, 144
maximal, 140
output, 141
pairs, 140–141
prime, 140, 145
states, 13
unconditionally, 141

Complement, 167
Complete
  circuit, 11, 208
  state code, 216
  state coding, 216
    algorithm, 223
    insertion point cost function, 220
    insertion points, 217
    state graph coloring, 219
    state signal insertion, 222
    transition points, 217
  sums, 167
Completely specified partition, 156
Component, 25
  declarations, 29
  instantiations, 29
Composition, 305
Compulsory transition, 94
Concurrent, 106
  statement section, 25
Conditional
  clause, 95
  input bursts, 95
Conditionally compatible, 142
Conflict, 107
  relation, 116, 119
Conformance, 20, 303, 308
Conformation equivalent, 308
Conforms, 303, 308
Confusion, 107
  asymmetric, 108
  symmetric, 108
Consensus, 167
Consistent
  signal sets, 305
  state assignment, 113
Constraint matrix, 133
Contain, 165
Context signals, 214, 240, 242
Conv_integer, 27
Correctness constraints, 231
Counterflow pipeline, 324
Cover, 148, 166, 230
Covering
  clause, 234
  constraint, 148, 231, 237
    violation, 241
  problem, 132
    See also BCP algorithm
    binate, 133
    context signals, 241–242
    prime implicant selection, 168
    speed-independent logic synthesis, 234
    state assignment, 161
    state minimization, 148
    unate, 133
  section, 235
Critical race, 155
CSC solver algorithm, 223
Cube, 165
  approximations, 315
Cycle, 87
  simple, 87
Cyclic core, 137

D

Data
  encoding
    bundled data, 61
    dual-rail, 58, 67
  hazards, 49
Data-driven processors, 323
DC-set, 165
Dead, 103
Deadlock, 34, 66
Declaration section, 25
Decomposition, 190
    See also technology mapping
  insertion points, 245
  speed-independent, 243
Delay, 26, 59
  elements, 11
  fault model, 332
  faults, 330
  problem of the first kind, 251
Delay-insensitive circuits, 11
  limits, 12
Delayed-reset domino, 342
DeMorgan's theorem, 188, 362
Deterministic, 32
Dichotomies, 156
Difference bound matrix, 268
Digraph
  acyclic, 88
  simple, 88
Direct transition, 155
Directed
  acyclic graph, 88
  don't cares, 93
  edge, 86
  graph, 86
Disabling rules, 118
Discrete-time, 265
  example, 265
  worst-case complexity, 265
Disengageable strong precedence arcs, 121
Disjoint, 361
  collection, 361
Distance 1 apart, 171
Distributive, 167, 188, 362
  state graphs, 214
Domain, 362

Dual-rail, 67
  protocol, 58
Duals, 362
Dummy timer, 268
Dynamic
  $0 \to 1$ hazard
    multiple-input change, 175
    single-input change, 170
  $0 \to 1$ transition, 169
  $1 \to 0$ hazard
    multiple-input change, 175
    single-input change, 170
  $1 \to 0$ transition, 169
  function hazard, 173
  hazard-free
    compatible, 181
    implicant, 185
    prime implicant, 185
  hazards
    generalized C-elements, 193
    multiple-input change, 175

E

Edges, 86
Empty set, 361
Enable module, 74
Enabled, 102, 118
  event, 118
Enabling event, 118
End
  cube, 177
  point, 172
  subcube, 178
  transitions, 217
Enters, 87
Entity, 25
Entrance constraint, 231, 237
  violation, 241
Enumerated types, 25
Environment, 4
Equilibrium, 112
Equipotential regions, 249
Equivalence
  class, 363
  relation, 363
Equivalent, 209
Essential
  hazard, 195
  primes, 168
  rows, 134
  variable, 134
Evans and Sutherland, 323
Events, 10, 116, 260
Eventually operator, 297
  timed, 303
Excitation
  cube, 238
  region, 213
  states, 212
Excited, 112
Expired, 118, 260
Extended
  burst-mode, 93
    See also XBM machines
    dynamic hazard problem, 179
    generalized C-elements, 193
    hazard-free logic synthesis, 183
    hazards, 177
    multilevel logic synthesis, 188
    sequential hazards, 194
    state assignment, 183
    state machines, 93
    state minimization, 180
    synthesis, 171
    technology mapping, 189
    transition, 177
  free-choice nets, 108
Extensions, 250

F

Failure
  exclusion, 310
  traces, 304
Failure-free, 307
Fault model
  bridging, 332
  delay, 332
  isochronic fork, 331
  stuck-at, 331
Faults, 330
Feedback delay requirement, 195
Finite state machine, 88
  Mealy machine, 89
  Moore machine, 89
Firing sequence, 102
Flow relation, 100
Floyd's algorithm, 269
Followed, 209
For loop, 33
Formal verification, 20
Four-phase handshaking, 5, 64
Fractional component, 262
Free-choice nets, 107
Function, 363
  hazard, 172
    dynamic, 173
    static, 173
Fundamental-mode, xv, 132, 195

G

Gate sharing, 237
Generalized

  C-elements, 193, 226
    burst-mode, 193
    extended burst-mode, 193
    speed-independent, 226
  transition cube, 177
Glitch, 165
Globally
  asynchronous locally synchronous, 338
  synchronous locally asynchronous, 338
Good extension, 250
Graph, 85
  basic definitions, 85
  bipartite, 88
  connected, 87
  directed, 86
  order, 87
  size, 87
  strongly connected, 88
  undirected, 86
Graphical representations, 8, 85
Greatest
  lower bound, 364
  member, 364
Guard, 4, 59, 63
Guard_and, 18, 60
Guard_or, 59

H

Handshake, 2
  assign, 59
  guard, 59
  guard_and, 60
  guard_or, 59
  package, 58, 355
  protocols, 57
  vassign, 60
Handshaking expansion, 61
  dual-rail encoding, 67
  four-phase, 64
  lazy-active, 65
  probe, 71
  reshuffling, 65
  state variable insertion, 66
  two-phase, 62
  VHDL modeling, 58
Hazard, 13, 165
  burst-mode, 176
  dynamic $0 \rightarrow 1$, 170
  dynamic $1 \rightarrow 0$, 170
  dynamic function, 173
  extended burst-mode, 177
  function, 172
  generalized C-elements, 193
  preserving transformations, 188
  static 0, 169
  static 1, 169
  static function, 173
Hazard-free
  decomposition
    algorithm, 246
    speed-independent, 243
  logic synthesis
    atomic gate implementation, 225
    burst-mode, 183
    extended burst-mode, 183
    generalized C-element implementation, 226
    multiple-input change, 183
    single-cube algorithm, 238
    single-input change, 165
    speed-independent, 223
    standard C-implementation, 230
Hierarchical verification, 308
Hold time, 2, 195, 334
Huffman, xv
  circuits, 13, 131
  flow table, 8, 90
  school, 131

I

I-nets, 111
Idempotent, 167, 362
Identities, 361
Identity, 362
If-then-else statements, 31
ILLIAC II, xv, 322
Image, 363
Implicant, 166
  candidate, 233
  excitation region, 231
Implied
  state, 113
  states, 232
  value, 112
In, 28
  transition, 155
Incident
  from, 87
  on, 87
  to, 87
Incompatible, 141
Incompletely specified
  Boolean function, 165
  partition, 156
Infinite loop, 33
Initial
  marking, 100
  state, 116
Initially marked rules, 116
Init_channel, 28
Inout, 28
Input, 4

burst, 91
conditional, 95
Insertion point, 217
complete state coding, 217
cost function, 220
decomposition, 245
filters, 246
Instruction statistics, 327
Interface nets, 121
Intersect, 361
Intersectible, 159
Intersection, 158, 165, 361
Into, 363
Inverse, 362
delete, 305
IPCMOS, 345
Isochronic fork, 16
fault model, 331

J

Joins, 87

K

K-bounded Petri net, 103
Karnaugh maps, 14

L

L0-live, 103
L1-live, 103
L2-live, 103
L3-live, 103
L4-live, 103
Labeled Petri net, 100
Lazy-active, 65
Least
member, 364
upper bound, 364
Leaves, 87
Level, 10
expression, 118
signaling, 5, 64
Library, 25
Linear
inequalities, 268
ordering, 363
Linear-time temporal logic, 296
Literal, 165
Live, 103, 111
Petri net, 103
Liveness, 103
Lk-live, 104
Logic
minimization, 131, 165
optimization
average-case, 327
synthesis
burst-mode, 183
combinational hazards, 169
extended burst-mode, 183
multiple-input change, 183
prime implicant generation, 166
prime implicant selection, 168
single-input change, 165
speed-independent, 223
two-level, 165
Loop, 33
Lower bound, 364
algorithm, 137
LTL formulas
checking validity, 298

M

M-nets, 111, 120
Macromodules, xv, 322
Marked graph, 107
component, 110
Marking, 9, 100
Matching/covering, 190
Matrix reduction method, 157
Maximal
compatibles, 140, 143
independent set, 137
intersectible, 159
member, 364
set property
BM machine, 92
XBM machines, 94, 96
Mealy machine, 89
Mean time between failure, 335
Member, 360
Merge gate, 73
Metastable state, 333
Microprocessors
asynchronous, 323
AMULET, 323
Caltech, 323
Minimal
member, 364
state, 214
MiniMIPS example, 36
block diagram, 36
channel-level diagram, 38
decode block, 43
dmem block, 47
execute block, 45
fetch block, 42
imem block, 41
instruction formats, 36
optimized, 48
decode block, 49
structural VHDL code, 39
write_back process, 52

Minimum-transition-time
  state assignment, 155
Minterm, 165
Mirror, 310
Model checking, 20, 296
Modeling timing, 260
Monotonicity, 314
Moore machine, 89
Muller, xv, 322
  C-element, 7
  circuits, 16, 207
  school, 207
Multilevel logic synthesis
  burst-mode, 188
  extended burst-mode, 188
  multiple-input change, 188
Multiple-input change, 132
    See also burst-mode and extended burst-mode
  combinational hazards, 173
  dynamic hazards, 175
  hazard-free logic synthesis, 183
  multilevel logic synthesis, 188
  sequential hazards, 194
  state assignment, 183
  state minimization, 180
  static hazards, 174
  synthesis, 171
  technology mapping, 189
Mutual exclusion element, 337
  CMOS circuit, 338

N

n-ary relations, 362
Next state operator, 296
Nonadjacent, 87
Noncritical race, 155
Nondeterminism
  delay, 26, 59
  package, 25, 32, 347
  selection, 26, 32
Nondeterministic, 32
Nondisabling rules, 118
Normal flow table, 155, 173
Normalize, 271

O

OFF-set, 165
ON-set, 165
One-to-one, 363
Onto, 363
Open
  generalized transition cube, 177
  transition cube, 172
Ordered
  n-tuple, 362
  pair, 362
  partitions, 159
  triple, 362
Ordinary Petri net, 100
Others, 32
Out, 28
Output, 4
  burst, 91
  compatible, 141
  semi-modular, 212
  stuck-at fault model, 331

P

Packages, 25
Pair chart, 141
Parallel
  communication, 35
  composition
    syntax-directed translation, 78
Partial order, 315, 363
Partially ordered set, 280, 363
Partition, 155
  list, 156
  theory, 155
Partitioning, 190
Passive, 6, 61
Peephole optimizations, 78
    See also syntax-directed translation
  call module optimization, 78
  enable module optimization, 78
  merge module optimization, 78
  select module optimization, 78
Performance analysis, 329
  average-case, 329
Persistent, 111
Petri net, 9, 85, 100
    See also signal transition graph
  asymmetric
    choice nets, 108
  classifications, 107
  dead, 103
  enabled, 102
  firing sequence, 102
  flow relation, 100
  free-choice nets, 107
    extended, 108
  initial marking, 100
  k-bounded, 103
  L0-live, 103
  L1-live, 103
  L2-live, 103
  L3-live, 103
  L4-live, 103
  labeled, 100
  live, 103
  liveness, 109

  Lk-live, 104
  marked graph, 107
  marking, 100
    characteristic, 102
  MG component, 110
  ordinary, 100
  places, 100
  postset, 100
  preset, 100
  reachability graph, 104
  reachable, 102
    markings, 102
  safe, 103
  safety, 109
  SM component, 110
  state machines, 107
  transition, 100
Philips pager, 324
Pipeline synchronization, xv, 345
Pipelining, 48
Places, 100
Port, 61
  declarations, 27
  map, 30
  modes, 28
  type, 28
POSET, 280
  algorithm, 280
  example, 281
  matrix, 280
  timing, 280
Possible traces, 304
Postset, 100
Power set, 361
Prefix-closed trace structure, 304
Premature firing, 331
Premax, 271
Preset, 100
Prime compatibles, 140, 145
  algorithm, 145
Prime implicant, 166
  essential, 168
  generation, 166
  selection, 168
Principle
  of abstraction, 360
  of duality, 362
  of extension, 360
Privileged cubes, 175, 184
Probe, 35
  handshaking expansion, 61, 71
Process statement, 26
Product, 165
Progress, 314
Project, 270
Propagation delay, 334
Propositional logic, 296
Protocol verification, 296

Q

Q-flop, 345
Q-modules, 345
Quantum synchronizers, 345
Quasi-delay insensitive, 16, 323
    See also speed-independent
Quiescent states, 213
Quotient set, 363

R

$\mathcal{R}$-related, 209
$\mathcal{R}$-sequence, 209
Racing, 155
Range, 362
RAPPID, xv, 325, 343
  microarchitecture, 327
  tag unit, 328
RAW hazard, 49
Reachability graph, 104
Reachable, 87, 102
  markings, 102
Recanonicalization, 269
Receive, 26
  parallel, 36
  syntax-directed translation, 76
Receptive, 304
Reduce algorithm, 135
Reduction, 135
Reflexive, 360, 363
Region function, 230
Regions, 262
  example, 262
  worst-case complexity, 265
Register locking, 49
Relative
  complement, 361
  timing, 316
Renaming, 305
Repetition, 31–32
  for loop, 33
  infinite loop, 33
  while loop, 33
Reply, 322
Request, 4, 57, 61
Required cubes, 174–175, 184
Reset region, 213
Reshuffling, 6, 65
Response time, 334
Restrict, 270
ρ-related to, 362
Row
  covers, 158
  dominance, 135

includes, 158
length, 138
Rules, 116, 260

S

Safe, 111
Petri net, 103
substitute, 312
Safety failures, 314
Satisfied, 118, 260
Scan paths, 332
Selection, 26, 31
case, 31
deterministic, 31
function, 32
if-then-else, 31
non-deterministic, 32
syntax-directed translation, 76
Selector module, 73
Self-checking, 331
Self-reference, 401
Self-reseting domino, 342
Self-timed, 249
See also speed-independent
Semi-modular, 212
output, 212
Send, 26
parallel, 35
syntax-directed translation, 76
Sensitivity list, 35
Sequencing event, 116
Sequential
composition
syntax-directed translation, 77
hazard, 194
hazards
burst-mode, 194
extended burst-mode, 194
multiple-input change, 194
Set, 360
builder notation, 360
region, 213
Setup time, 2, 195, 334
Shared row assignments, 197
Shortest path problem, 269
Signal
assignment
syntax-directed translation, 74
in TEL structure, 116
in VHDL, 25
transition graph, 100, 111
See also Petri net
Simple
ordering, 363
path, 87
Simply ordered set, 363
Single-cube algorithm, 238
Single-cycle transitions, 112
Single-input change, 132
combinational hazards, 169
hazard-free logic synthesis, 165
state assignment, 154
state minimization, 140
Siphon, 109
Skew-tolerant codes, 80
Specification, 4
Speed-independent, xv–16, 209, 322
See also Muller circuits,quasi-delay insensitive, and self-timed
atomic gate implementation, 225
complete state coding, 216
formal definition, 208
generalized C-element implementation, 226
hazard-free decomposition, 243
hazard-free logic synthesis, 223
limitations, 248
output semi-modular, 212
semi-modular, 212
single-cube algorithm, 238
standard C-implementation, 230
totally sequential, 210
Stable, 155, 238
state, 90
Standard C-implementation, 230
Start
cube, 177
point, 172
subcube, 178
transitions, 217
State, 208
minimization
burst-mode, 180
assignment, 14, 131
burst-mode, 183
extended burst-mode, 183
multiple-input change, 183
outputs as state variables, 163
single-input change, 154
diagram, 89, 209
graph, 112, 207, 210
coloring, 219
labeling function, 112
machine, 85
component, 110
machines, 107
minimization, 14, 131
extended burst-mode, 180
multiple-input change, 180
single-input change, 140
signal insertion, 222
transitions, 112

variable, 12, 66
Static
0 → 0 transition, 169
1 → 1 transition, 169
0-hazard
multiple-input change, 174
single-input change, 169
1-hazard
multiple-input change, 174
single-input change, 169
function hazard, 173
hazards
multiple-input change, 174
Std_logic, 25, 58
std_logic_vector, 25
STG properties
live, 111
persistent, 111
safe, 111
single-cycle transitions, 112
Stoppable clock, 336
Strong
conformance, 314
precedence arcs, 121
until operator, 296–297
Strongly connected, 88
Structural
modeling, 27
pattern matching, 190
Stubborn sets, 315
Stuck-at fault model, 331
output, 331
Subset, 360
Success traces, 304
Sum-of-products, 165
SUN counterflow pipeline, 324
Switching region, 213
Symmetric, 360, 363
confusion, 107
difference, 361
Synchronization, 321
See also arbitration
error, 335
failure, 2, 333
eliminating the probability of, 336
probability of, 334
reducing the probability of, 335
problem, 321, 332
Synchronous timing, 2
Syntax-directed translation, 73
parallel composition, 78
peephole optimizations, 78
receive statement, 76
selection statement, 76
send statement, 76
sequential composition, 77
signal assignment, 74
while loop, 74

T

Tag unit, 328
τ-partitions, 156
Technology mapping, 189
See also decomposition
burst-mode, 189
extended burst-mode, 189
multiple-input change, 189
TEL structures, 85, 116, 260
Temporal logic, 296
linear-time, 296
timed LTL, 300
Terminal class, 209
Terminating transition, 94
Termination, 137
Ternary relation, 362
Testing asynchronous circuits, 330
Time separation of events, 292
Time-quantified requirements, 300
Timed
always operator, 303
circuit, 17, 259, 289
tag unit, 328
event/level structure, 9
eventually operator, 303
LTL, 300
interval operators, 303
sequences, 261
state space exploration, 262
states, 260
trace theory, 314
trace, 314
until operator, 300
Timers, 260
Timing failure, 314
Tokens, 9
Total state transition relation, 296
Totally sequential, 210
Trace, 100, 303
See also allowed sequences
structure, 303–304
canonical, 310
prefix-closed, 304
theory, 303
Transit time, 249
Transition, 100
compulsory, 94
cube, 172
cubes, 171
dynamic 0 → 1, 169
dynamic 1 → 0, 169
labeling function, 111
points, 217

complete state coding, 217
restrictions, 218
signaling, 5, 63
static 0 → 0, 169
static 1 → 1, 169
subcubes, 175
terminating, 94
Transitive, 360, 363
Trap, 110
Trigger signals, 214
Trigger
cube, 239
signals, 240
Two-level logic minimization, 165
See also logic synthesis
Two-phase handshaking, 5, 63
Two-sided timing constraint, 329
Types, 25

U

U-v path, 87
Unacceptable variable, 134
Unate covering problem, 133
Unbalanced logic trees, 327
Unbounded gate delay model, 16, 207
Unconditionally compatible, 141
Unfoldings, 315
Unger, xiii
Union, 361
Unique
entry point, 92
state assignment, 113
state code, 216
Universal set, 361
University of Utah, 323
Unspecified, 156
Until operator
strong, 296
timed, 300
weak, 297
Untimed state, 260
Update
poset algorithm, 280
zone algorithm, 270
Upper bound, 364

V

Vacuous, 7, 65
Validation, 20
Vassign, 60
Verification, 296
circuit, 303
protocol, 296
Vertices, 85
VHDL constructs
aliases, 42
architecture, 25
assert statement, 59
attribute, 27
buffer, 58
case, 31
comments, 25
component, 25
declarations, 29
instantiations, 29
concurrent statement section, 25
control structures, 31
declaration section, 25
entity, 25
enumerated types, 25
for loop, 33
if-then-else, 31
infinite loop, 33
library, 25
others, 32
packages, 25
port
declarations, 27
map, 30
modes, 28
type, 28
process statement, 26
repetition, 31
selection, 31
signals, 25
structural modeling, 27
types, 25
while loop, 33
Violating states, 240
covering constraint, 241
entrance constraint, 241
generalized C-element implementation, 240
standard C-implementation, 241
Violations
unresolvable, 242

W

Wait statement, 35
Weak
precedence arcs, 121
until operator, 297
Well-ordered, 364
While loop, 33
syntax-directed translation, 74
Wine shop example
discrete-time, 265
LTL, 297
patron handshaking model
3-bit dual-rail, 70
dual-rail, 68
four-phase bundled data, 64

two wine shops, 72
two-phase bundled data, 63
POSETS, 281
problem specification, 1
regions, 262
shop handshaking model
dual-rail, 68
four-phase bundled data, 64
lazy-active, 66
reshuffled, 65
two-phase bundled data, 63
STG models, 116
synchronization problem, 332
syntax-directed translation, 73
timed LTL, 303
timed, 260
two patron example, 96
STG models, 116
TEL structure model, 120
two wine shops, 71
VHDL channel model, 24
VHDL handshaking model, 58
VHDL structural model, 27
winery handshaking model
3-bit dual-rail, 69
dual-rail, 67
four-phase bundled data, 64
two wine shops, 72
two-phase bundled data, 62
zones, 271

X

XBM machines, 93
See also extended burst-mode
directed don't cares, 93
edge-labeling functions, 99
formal model, 98
maximal set property, 94, 96

Z

Zones, 267
example, 271