

Arithmetic Data Value Speculation

A THESIS SUBMITTED TO

THE SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

OF THE UNIVERSITY OF ADELAIDE

By

Daniel R. Kelly

B.E. (Computer Systems)(Hons),

B.Sc. (Maths & Computer Science)

IN FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF DOCTOR OF PHILOSOPHY (MICROELECTRONICS)



Commenced, April 2005

Thesis Submitted, August 2010

DECLARATION OF ORIGINALITY

This work contains no material that has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of the thesis, when deposited in the University Library, being available for loan, photocopying, and dissemination through the library digital thesis collection, subject to the provisions of the Copyright Act 1968.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library catalogue, the Australasian Digital Thesis Program (ADTP) and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

DANIEL R. KELLY

School of Electrical and Electronic Engineering

The University of Adelaide

September 13, 2010

ACKNOWLEDGEMENTS

I wish to acknowledge my principal supervisor, Dr. Braden J. Phillips, for all of his help and guidance in all the things that mattered, including: identifying a stimulating research project; establishing my scholarship that made the project possible; being flexible enough to allow my sometimes backwards approach but firm enough to reign me in where needed; and an attention to detail and conviction in the importance of good, clear writing (if in coming pages you disagree, I am happy to award him blame or credit ;P).

My co-supervisor Dr. Said Al-Sarawi has been invaluable in providing a very focused, objective critique—with an attention to detail and response time that was amazing. I am grateful for the time and resources that Said has committed to me when needed, above and beyond the call of duty.

Dr. Andrew Beaumont-Smith has been extremely generous in providing an industry perspective and finding opportunities to work in the job of a lifetime. Andrew's influence on me has been to contextualise my research and to change my perspective in a way that no one else has. *"Just get your PhD ASAP"*.

Fiona Tselentis has been behind the scenes gently pushing me along and helping in so many ways that she doesn't even know about.

Finally I would like to thank my parents for their support and encouragement. This thesis would not have been possible without them.

This project has been a rewarding and challenging undertaking. I will never, ever do one again. Without further ado, I present my thesis, warts and all.

DANIEL R. KELLY

September 13, 2010

ABSTRACT

Arithmetic approximation is used to decrease the latency of an arithmetic circuit by shortening the critical path delay or the sampling period so that result is not guaranteed to be correct for every input combination. Thus, an acceptable compromise between the circuit latency and the average probability of correctness drives the circuit design. Two methods of arithmetic approximation are:

temporally incompleteness where circuits quote the result before the critical path delay (overclocking); and

logically incompleteness where circuits use simplified logic, so that most input cases are calculated correctly, but the slowest cases are calculated incorrectly.

Arithmetic data value speculation (ADVS) is a speculation scheme based on arithmetic approximation, and is used to increase the throughput of a general purpose processor. *ADVS* is similar to branch prediction, an arithmetic instruction is issued to an exact arithmetic unit and an approximate arithmetic unit which provides an approximate result faster than the exact counterpart. The approximate result is forwarded to dependent operations so they may be speculatively issued. When the exact result is eventually known, it is compared to the approximate result, and the pipeline is flushed if they differ.

This thesis, "*Arithmetic Data Value Speculation*", presents work in the field of digital arithmetic and computer architecture. A summary of current probabilistic arithmetic methods from the literature is provided, and novel designs of approximate integer arithmetic units are presented, including results from logical synthesis. A case study demonstrates approximate arithmetic units used to increase the average throughput of benchmark programs by speculatively issuing dependent operations in a RISC processor.

The average correctness of the approximate arithmetic units are shown to be highly data dependent, results vary depending on the benchmarks being run. In addition, the

Abstract

average correctness when running benchmarks is consistently higher than for random inputs. Simulations show that many arithmetic operations are often repeated in the same benchmark, leading to a high variation in correctness. Speculative gains from one operation can be offset by speculation losses due repeated incorrect approximation of another approximate unit, so typical throughput gains through speculation in a general purpose processor pipeline are low. The minimum threshold correctness of an approximate arithmetic unit used for speculation is shown to be approximately 95%. Logic synthesis is used to determine power, area and timing information for approximate units implemented from novel algorithms, and show a reduction in arithmetic cycle latency for integer operations, and the expense of 50 % leakage and area, and 90 % dynamic power.

Value speculation can be complemented by result caching; repeated pipeline flushes can be avoided if the correct result is know before speculation, the average operation latency can be reduced, and caching can be used for operations that are difficult to approximate.

PUBLICATIONS

The following is a list of publications published or submitted during the Ph.D. candidature by the author.

Published book chapters

1. Daniel R. Kelly and Braden J. Phillips, “Arithmetic data value speculation”, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3740 NCS, pp. 353–366, 2005.
2. Daniel R. Kelly, Braden J. Phillips and Said Al-Sarawi, *Algorithm-Architecture Matching for Signal and Image Processing*, volume 73 of *Lecture Notes in Electrical Engineering*, chapter 4 “Operators for embedded systems: Approximate Multiplication and Division for Arithmetic Data Value Speculation in a RISC Processor”. Springer, 1st edition, December 2010. ISBN: 978-90-481-9964-8.

Published conference papers

3. Daniel R. Kelly, Braden J. Phillips and Said Al-Sarawi, “An open source synthesizable model in VHDL of a 64-bit MIPS-based processor”, in *Proceedings of SPIE—The International Society for Optical Engineering*, vol. 6414, (Adelaide, Australia), 2007.
4. Braden J. Phillips, Daniel R. Kelly and Brian W. Ng, “Estimating adders for a low density parity check decoder”, in *Proceedings of SPIE—The International Society for Optical Engineering*, vol. 6313, (San Diego, CA, United States), 2006.

Publications

5. Braden J. Phillips, Cain D. Schmidt and Daniel R. Kelly, “Recovering data from USB flash memory sticks that have been damaged or electronically erased”, in *e-Forensics '08: Proceedings of the 1st international conference on Forensic applications and techniques in telecommunications, information, and multimedia and workshop*, (ICST, Brussels, Belgium, Belgium), pp. 1–6, ICST (Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), 2008.
6. Daniel R. Kelly, Braden J. Phillips and Said Al-Sarawi, “Approximate unsigned binary integer dividers for arithmetic data value speculation”. In *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Sophia Antipolis, France, September 22–24, 2009. <http://www.ecsi-association.org/ecsi/dasip/dasip09>.
7. Daniel R. Kelly, Braden J. Phillips and Said Al-Sarawi, “Approximate signed binary integer multipliers for arithmetic data value speculation”. In *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Sophia Antipolis, France, September 22–24, 2009. <http://www.ecsi-association.org/ecsi/dasip/dasip09>.
8. Daniel R. Kelly, Braden J. Phillips and Said Al-Sarawi, “Increasing throughput of a RISC system using arithmetic data value speculation”. In *Conference Record of the Forty-Third Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, California, USA, November 1–4, 2009.

CONTENTS

Declaration of Originality	i
Acknowledgements	iii
Abstract	v
Publications	vii
Contents	7
List of Figures	11
List of Tables	15
List of Algorithms	17
List of Acronyms	19
Nomenclature	21
1 Introduction	23
1.1 Research objective	25
1.2 Research outcome	25
1.3 Thesis outline	26
1.4 Research Contributions	28
I Background	31
2 A Brief Review of Computer Architecture and Digital Arithmetic	33

Contents

2.1	Digital arithmetic	34
2.2	SimpleScalar	35
2.2.1	System components	36
2.2.2	Pipeline overview	38
2.2.3	Arithmetic operation latencies	39
2.3	Benchmarks	41
2.3.1	Arithmetic benchmarks	42
2.3.2	Mediabench	43
2.3.3	SPEC	45
2.3.4	Test benchmarks	45
2.3.5	Benchmark sizes	48
2.3.6	Arithmetic operations in benchmark programs	49
2.4	Synthesis	49
3	Theory and Applications of Arithmetic Approximation	51
3.1	Methods of approximation	53
3.1.1	Logical incompleteness	54
3.1.2	Temporal incompleteness	58
3.2	Fundamental arithmetic results	59
3.2.1	Addition and subtraction	59
3.2.2	Approximate parallel prefix adders	76
3.2.3	Multiplication and Division	80
3.3	Applications of approximate arithmetic	81
3.3.1	Addition and subtraction	81
3.3.2	Multiplication	82
3.3.3	Division	83
3.4	Conclusion	84
4	Can ADVS Improve the Performance of a Generic RISC Processor?	85
4.1	Program execution	86
4.2	Integer arithmetic	87
4.2.1	Types of integer arithmetic	89
4.2.2	Signed and unsigned arithmetic	89
4.2.3	Operand magnitude	89

4.3	Floating point arithmetic	95
4.3.1	Types of floating point arithmetic	95
4.3.2	Signed and unsigned arithmetic	96
4.3.3	Operand magnitude	96
4.4	Performance limits	98
4.4.1	Simulation parameters	101
4.4.2	Throughput upper bound	101
4.4.3	Compiler optimisation effects	109
4.4.4	Delay-correctness lower bound	110
4.5	Conclusion	113
5	Preliminary Experiments	115
5.1	Validation of average worst-case carry length	116
5.2	Simple modifications to basic arithmetic circuits	118
5.2.1	Carry propagation in array multipliers	119
5.2.2	Tree multipliers without a CPA	120
5.2.3	Array multiplication with an l bit CPA	124
5.2.4	Restoring division with t cycles	127
5.2.5	Non-restoring division with t cycles	127
5.2.6	Floating point rounding	129
5.3	Conclusion	133
II	Approximate Arithmetic	135
6	Approximate Integer Multiplication	137
6.1	Counters	139
6.2	Multipliers	139
6.2.1	Multiplier topology	141
6.2.2	Allocation schemes	142
6.2.3	Uniform random inputs	144
6.3	Unsigned multiplier results	145
6.3.1	Benchmark data inputs	147
6.3.2	Multiplier error	149
6.4	Signed approximate multiplication	150

Contents

6.4.1	Baugh-Wooley signed multiplication	150
6.4.2	Signed approximate multiplication error	153
6.5	Synthesis	156
6.6	Approximate integer multipliers for ADVS	161
6.7	Conclusion	162
7	Approximate Integer Division	163
7.1	Division algorithms	164
7.1.1	Exact division algorithm	165
7.1.2	Approximating division algorithm	167
7.2	Hardware design	168
7.2.1	Initialisation stage	168
7.2.2	Division stage	170
7.2.3	Accumulation stage	170
7.2.4	Implementation notes	170
7.2.5	Variations	171
7.3	Probability of correctness	173
7.3.1	Approximating dividers	173
7.3.2	Baseline SRT divider	177
7.4	Signed approximate integer division	179
7.5	Synthesis	184
7.6	Approximate integer dividers for ADVS	186
7.7	Conclusion	187
8	Approximate Floating Point Arithmetic	189
8.1	Approximating floating point units	190
8.1.1	Approximation techniques	193
8.2	Approximate FP unit correctness	194
8.2.1	Approximate FP adder correctness	195
8.2.2	Approximate FP multiplier correctness	196
8.2.3	Approximate FP divider correctness	198
8.3	Synthesis	201
8.3.1	Baseline	201
8.3.2	Approximate units	202

8.4	Conclusion	202
 III Application		 205
9	Result Caching	207
9.1	Caching techniques	209
9.1.1	Result caching schemes	209
9.1.2	Result prediction schemes	210
9.2	Results	212
9.2.1	Operand Bit Assertion	212
9.2.2	Cache replacement policies	213
9.2.3	Simulation Results	215
9.2.4	Implementation	215
9.3	Result caches for ADVS	217
9.3.1	65 nm process	218
9.3.2	180 nm process	219
9.4	Conclusion	221
 10	 Approximate Adders in LDPC	 223
10.1	Background	224
10.2	Low density parity check codes	225
10.3	Approximate multioperand adders	226
10.4	LDPC using approximate multioperand adders	233
10.4.1	LDPC check node synthesis	233
10.5	LDPC decoding performance	235
10.6	Conclusions	237
 11	 ADVS Simulation	 239
11.1	Synthesis	240
11.1.1	Pipelined arithmetic units	241
11.1.2	Synthesis summary	248
11.2	ADVS simulation	250
11.2.1	Simulated arithmetic latencies	251
11.2.2	Benchmark simulation with ADVS	252

Contents

11.3	Conclusion	267
12	Conclusions	269
12.1	Summary of Contributions	272
12.2	Future Research	273
IV	Appendices	277
A	GCC man pages	279
A.1	<i>gcc</i> man pages	280
A.1.1	OPTIMIZATION OPTIONS	280
B	Source Code	283
B.1	Probability of correctness	284
B.1.1	<i>backCount.m</i>	284
B.1.2	<i>fixGenRight.m</i>	285
B.2	Logic approximation	286
B.2.1	<i>approx.c</i>	286
B.2.2	<i>approx.h</i>	288
B.2.3	<i>reduce.c</i>	288
B.2.4	<i>io.c</i>	297
B.3	Simple arithmetic benchmarks	304
B.3.1	Dhrystone	304
B.3.2	CalcPi	306
B.4	VHDL multiplier generator	307
B.4.1	<i>multgen.c</i>	307
C	Arithmetic Operands	317
C.1	Arithmetic operands in benchmark programs	318
C.2	Operand bit-assertion tables	335
D	Detailed benchmark descriptions	341
D.1	Mediabench benchmarks	342
D.2	SPEC benchmarks	345
D.2.1	<i>SPEC CINT2000</i>	345

D.3	Test benchmarks	357
D.3.1	fbench	357
D.3.2	ffbench	358
E Arithmetic VHDL source code		359
E.1	32 bit (7; 2) unsigned multiplier	360
E.2	32 bit approximate DI divider	361
E.2.1	Initialisation stage	362
E.2.2	Division stage	364
E.2.3	Accumulation stage	366
E.3	32 bit floating point adder/subtractor	367
E.4	32 bit FP multiplier	370
E.5	32 bit FP divider	373
F SimpleScalar machine instructions		377
F.1	SimpleScalar machine instructions	378
Bibliography		389
Index		391

LIST OF FIGURES

2.1	Overview of the <i>SimpleScalar PISA</i> pipeline.	38
3.2	A dual-rail full-adder cell.	63
3.3	AWCCL for signed two's complement 32 bit additions.	66
3.4	An example of Liu and Lu's approximate adder.	68
3.5	Correctness of N bit adders with a maximum carry length of l bits.	73
3.6	Comparison of 32 bit adder latencies.	75
3.7	An exact and approximate 16 bit Sklansky adder.	77
3.8	An exact and approximate 16 bit Kogge-Stone adder.	78
3.9	An exact and approximate Brent-Kung adder.	79
4.1	Relative proportions of integer and floating-point instructions.	87
4.2	Bit distribution patterns of integer addition and subtraction.	92
4.3	Bit distribution patterns of integer multiplication.	93
4.4	Bit distribution patterns for integer division.	94
4.5	Scatter plot of 32 bit floating point operands in benchmark programs.	97
4.6	Bit assertion distributions of 32 bit FP addition and subtraction operands.	99
4.7	Bit assertion distribution of 32 bit FP multiplication and division operands.	100
4.8	Correctness and latency of benchmarks simulated in <i>SimpleScalar</i>	105
4.9	Maximum throughput gain using single cycle arithmetic in <i>ADVS</i>	107
4.10	Maximum throughput gain using perfect approximate arithmetic in <i>ADVS</i>	108
4.11	Latency of <i>SPEC 172.mgrid</i> benchmark with different optimisation levels.	110
4.12	Delay-correctness using basic flushing.	111
4.13	Delay-correctness using no-resteering flushing scheme.	112
5.1	Correctness of approximate Liu and Lu adders for <i>SPEC CPU95data</i>	117
5.2	Proportion of correct sums with a worst-case carry length of l bits.	119

List of figures

5.3	Partial products in an array multiplier.	120
5.4	Hot map of asserted C_{OUT} bits in an array multiplier.	121
5.5	An 8×8 bit Wallace multiplier with no CPA.	123
5.6	Unsigned 16×16 bit array multipliers with modified CPA adders.	125
5.7	Correctness of unsigned multipliers with a l bit ripple-carry CPA.	126
5.8	Correctness of an array multiplier with an l bit Liu and Lu CPA adder.	126
5.9	Correctness of a restoring divider with t division rounds.	128
5.10	Correctness of non-restoring dividers with t division rounds.	128
5.11	Correctness of a temporally incomplete 16×16 bit array multiplier.	132
6.1	Comparison of exact and approximate counter-based integer multipliers.	141
6.2	Partial product allocation schemes for integer multiplication.	144
6.3	Correlated partial products in integer multiplication.	144
6.4	Histogram of integer multiplication operand magnitude.	147
6.5	Histogram of bit errors for an approximate unsigned integer multiplier.	149
6.6	Relative error of an approximate unsigned (4;2) 32 bit integer multiplier.	150
6.7	Partial products in a Baugh-Wooley integer multiplier.	151
6.8	Comparison of exact and approximate counter-based integer multipliers.	152
6.9	Relative error for an approximate signed (4;2) integer multiplier.	155
6.10	Histogram of bit errors for an approximate signed integer multiplier.	156
6.11	Delay vs. correctness scatter plot for unsigned approximate multipliers.	157
6.12	Delay vs. correctness scatter plot for signed approximate multipliers.	158
7.1	Approximating integer dividers.	172
7.2	Convergence of an approximated integer division.	173
7.3	Correctness of unsigned DI and SMT dividers.	175
7.4	Average correctness for unsigned approximating integer dividers.	176
7.5	P-D diagram of a baseline exact SRT divider.	177
7.6	Delay of a 32 bit variable latency unsigned radix-4 divider.	178
7.7	Average correctness of a signed approximating integer divider.	181
7.8	Delay of a 32 bit variable latency signed radix-4 divider.	183
8.1	Block diagrams of IEEE Std. 754 floating point units.	191
8.2	Correctness of a Liu and Lu floating point significand adder.	195
8.3	Counter input bits vs. correctness in a floating point multiplier.	197

8.4	Average error in the fpDiv approximated significand.	199
8.5	Average correctness of approximated significands in floating point division.	200
8.6	Approximate floating point multiplier latency vs. correctness.	203
9.1	Bit assertion histograms for intMult and intDiv	213
9.2	Hit rate of direct mapped result caches.	216
9.3	Hit rate of LRU result caches with different levels of associativity.	217
9.4	Hit rates of FIFO caches with different levels of associativity.	218
9.5	Comparison of result cache replacement schemes.	219
9.6	Average IPC gain for when implementing result caches.	220
10.1	Graph of LDPC check nodes.	225
10.2	Architecture of a k input check node.	227
10.3	Counters and compressors	229
10.4	Exact (6; 2) compressors using (3; 2) counters.	230
10.5	Exact (7; 2) compressors using (3; 2) counters.	231
10.6	Schematic view of a (3; 2) counter.	231
10.7	Schematic view of a (4; 2) saturating counter.	232
10.8	4 bit approximate compressors.	234
10.9	7 bit compressor LDPC error frequency.	235
10.10	LDPC frame error rate and average decoder iterations vs. SNR.	236
11.1	Pipelined IEEE Std. 754 FP Adder/Subtractor.	244
11.2	Pipelined IEEE Std. 754 FP Multiplier.	246
11.3	Pipelined IEEE Std. 754 FP Divider.	247
11.4	IPC of <i>arithmetic</i> benchmarks in an ADVS-enabled system.	253
11.5	IPC for <i>Mediabench</i> benchmarks in an ADVS-enabled system.	254
11.6	IPC of <i>SPEC</i> benchmarks in an ADVS-enabled system.	255
11.7	IPC of <i>test</i> benchmarks in an ADVS-enabled system.	256
11.8	Operand cache hit rates in an ADVS-enabled system.	257
11.9	Effect of approximate FP arithmetic on throughput in ADVS.	263
11.10	Effect of operand caching in an ADVS-enabled system.	265

LIST OF TABLES

2.1	<i>SimpleScalar</i> configuration used in this thesis.	40
2.2	<i>SimpleScalar</i> arithmetic unit latencies and repeat-rates.	41
2.3	Description of benchmarks in the <i>SPEC CINT2000</i> suite.	46
2.4	<i>SPEC CPU2000</i> benchmarks compiled for use with <i>SimpleScalar</i>	47
2.5	File sizes of benchmarks.	50
3.1	Predictions of the AWCCL for integer addition.	64
3.2	Summary of AWCCL for <i>SPEC CINT2000</i> benchmarks	65
3.3	AWCCL for addition of signed positive and negative operands.	66
3.4	Calculated errors for a 8 bit adder with maximum carry length l bits.	71
3.5	Synthesis results of 32 bit Liu and Lu adders.	74
4.1	Average proportions of instructions executed.	88
4.2	Ratio of signed to unsigned retired instructions in benchmarks.	90
4.3	Proportion of unsigned integer arithmetic operands that are zero.	91
4.4	Signed integer operand values	91
4.5	32 bit FP operand values	96
4.6	64 bit FP operand values	97
4.7	Simulation parameters used for <i>arithmetic</i> and <i>Mediabench</i> benchmarks.	101
4.8	Simulation parameters used for <i>SPEC</i> benchmark simulation.	102
4.9	Throughput of <i>arithmetic</i> benchmarks with single-cycle arithmetic.	102
4.10	Throughput of <i>Mediabench</i> benchmarks with single cycle arithmetic.	103
4.11	Throughput of <i>SPEC</i> benchmarks with single cycle arithmetic.	104
4.12	Average execution time of benchmarks without ADVS.	109
5.1	Correctness of tree multipliers without a CPA.	122
5.2	Floating point rounding results identical to ‘round to nearest’.	131

List of Tables

6.1	Correctness of $(n; m)$ counters with random inputs.	140
6.2	Correctness of unsigned approximate multipliers with random data.	146
6.3	Correctness of unsigned approximate multipliers with benchmark data.	148
6.4	Correctness of signed approximate multipliers with random data.	153
6.5	Correctness of signed approximate multipliers with benchmark data.	154
6.6	Synthesis results for unsigned $(n; m)$ 32×32 bit multipliers	159
6.7	Synthesis results for signed approximate integer multipliers.	160
7.1	An example demonstrating the approximation of $-d$	180
7.2	Peak correctness for DI dividers for random and benchmark data.	182
7.3	Difference in correctness for signed dividers that approximate $-d$	182
7.4	Maximum correctness difference for DI, SMT and GBP dividers.	184
7.5	Synthesis results for unsigned 32 bit integer dividers.	185
8.1	Correctness of 24 bit significand multipliers for fpMult.	196
9.1	History table for a context-based value predictor.	211
9.2	An ordered list of bit assertion frequency of arithmetic operands.	214
9.3	Modelled cache access times in a 65 nm process using CACTI 5.3.	220
9.4	Modelled cache access times in a 65 nm process using CACTI 5.3.	221
9.5	180 nm result caches process using CACTI 4.1.	221
10.1	Truth tables for saturating and reflecting $(4; 2)$ counters.	228
10.2	HSPICE simulation of a $(4; 2)$ counter and a $(4; 2)$ counter.	232
10.3	LDPC check node synthesis results.	235
11.1	Synthesis results for exact integer multipliers.	242
11.2	Synthesis results for approximate integer multipliers.	243
11.3	Synthesis results for pipelined exact integer dividers.	243
11.4	Synthesis results for the pipelined floating point adder.	245
11.5	Synthesis results for the pipelined floating point multiplier.	245
11.6	Synthesis results for the pipelined floating point divider.	247
11.7	ADVS arithmetic unit area cost.	248
11.8	ADVS arithmetic unit dynamic power cost.	249
11.9	ADVS arithmetic unit leakage power cost.	250
11.10	Exact and approximate arithmetic latencies used in ADVS simulation.	251

11.11	Correctness of approximate units in a basic <i>ADVS</i> -enabled system.	258
11.12	Correctness of approximate units with operand caching and <i>ADVS</i>	258
11.13	Operand cache hit rates in an <i>ADVS</i> -enabled system.	259
11.14	Average IPC change for an <i>ADVS</i> -enabled system.	259
11.15	Differences in <i>glibc</i> library values on a x86 platform compared to <i>SimpleScalar</i>	262
C.1	Number of retired integer instructions in the <i>arithmetic</i> benchmarks.	319
C.2	Number of retired integer instructions in the <i>Mediabench</i> benchmarks.	320
C.3	Number of retired integer instructions in the <i>SPEC</i> benchmarks.	321
C.4	Number of retired integer instructions from each <i>test</i> benchmark.	322
C.5	Proportion of integer instructions in the <i>arithmetic</i> benchmarks.	323
C.6	Proportion integer instructions in the <i>Mediabench</i> benchmarks.	324
C.7	Proportion of integer instructions in the <i>SPEC CPU2000</i> benchmarks.	325
C.8	Proportion of integer instructions in the <i>test</i> benchmarks.	326
C.9	Number of floating point instructions in the <i>arithmetic</i> benchmarks.	327
C.10	Number of floating point instructions in the <i>Mediabench</i> benchmarks.	328
C.11	Number of floating point instructions in the <i>SPEC CPU2000</i> benchmarks.	329
C.12	Number of floating point instructions in the <i>test</i> benchmarks.	330
C.13	Proportion of floating point instructions in the <i>arithmetic</i> benchmarks.	331
C.14	Proportion of floating point instructions in the <i>Mediabench</i> benchmarks.	332
C.15	Proportion of floating point instructions in the <i>SPEC</i> benchmarks.	333
C.16	Proportion of floating point instructions in the <i>test</i> benchmarks.	334
C.17	Bit assertion probabilities for signed 32 bit operands.	336
C.18	Bit assertion probabilities for unsigned integer operands.	337
C.19	Bit assertion probabilities for single precision floating point numbers.	338
C.20	Bit assertion probabilities for double precision floating point numbers.	339
C.21	(<i>continued...</i>)	340
F.1	Machine instructions provided in the <i>PISA</i> architecture.	378
F.1	(<i>continued...</i>)	379
F.1	(<i>continued...</i>)	380

LIST OF ALGORITHMS

3.1	<i>logicApprox</i>	55
3.2	<i>BackCount</i>	71

LIST OF ACRONYMS

ADPCM	adaptive differential pulse code modulation
AWCCL	average worst-case carry length, in bits
AWGN	additive white gaussian noise
CCITT	International Telegraph and Telephone Consultative Committee
CMOS	complementary metal oxide semiconductor
DFT	discrete fourier transform
FP	floating point
EPIC	Efficient Pyramid Image Coder
FA	full adder
FFT	fast fourier transform—an efficient algorithm to compute the discrete fourier transform (DFT) and its inverse.
GSM	Global System for Mobile communications: originally from Groupe Spécial Mobile
HA	half adder
IEEE	Institute of Electrical and Electronic Engineers
INT	integer
ISA	instruction set architecture
JPEG	Joint Photographic Experts Group

List of Algorithms

LSB	least significant bit
MPEG	Moving Picture Experts Group
MSB	most significant bit
<i>MUX</i>	Multiplexor
SNR	signal-to-noise ratio
QPSK	quadrature phase-shift keying
PGP	Pretty Good Privacy
SPEC	Standard Performance Evaluation Corporation
SPHERE	SPeech HEader REsources
ULP	Unit of least precision

NOMENCLATURE

Mathematical notation

$f(n) \in O(n)$ $f(n) \leq g(n) \cdot k$, f is bounded above by g (up to constant factor) asymptotically

$f(n) \in \Omega(n)$ $f(n) \geq g(n) \cdot k$, f is bounded below by g (up to constant factor) asymptotically

$f(n) \in \Theta(g(n))$ $g(n) \cdot k_1 \leq f(n) \leq g(n) \cdot k_2$, f is bounded both above and below by g asymptotically

$\lfloor x \rfloor$ floor(x), greatest integer $\leq x$

$\lceil x \rceil$ ceiling(x), least integer $\geq x$

P_M probability of correctness for a multiplier

P_C probability of correctness for a counter

Standard notation

N Number of bits in an operand; word length

a_N Average worst-case carry length for N bit addition

l Length in bits of a carry-chain in an adder

PC Program Counter register

SP Stack Pointer register

Multiplication

n number of input bits to a counter

Nomenclature

m number of output bits from a counter

Division

z dividend

d divisor

q exact quotient after division $q = z/d$

d_H least binary power greater than or equal to d

d_L greatest binary power less than or equal to d

\tilde{d} approximate divisor

\tilde{D} $\log_2(d)$, or the number of zeros after the MSB of d

q_i quotient after round i

r_i remainder after round i

e_i error in q_i after round i

n number of bits in the unsigned binary representation of z and d

k number of bits in the quotient, q

t maximum number of division rounds

f number of fractional bits maintained in q_i

m constant multiplicand in the division round: $m = d - \tilde{d}$

p_1 most significant non-zero partial product in the binary multiplication
 $r_i \times m$

p_2 after p_1 , the next most significant partial product in the binary multiplication
 $r_i \times m$