

Associative Digital Network Theory

An Associative Algebra Approach to Logic, Arithmetic and State Machines

NICO F. BENSCHOP

Preface

This book is intended for researchers at industrial laboratories, teachers and students at technical universities, in electrical engineering, computer science and applied mathematics departments, interested in new developments of modeling and designing digital networks (*DN* : state machines, sequential and combinational logic) in general, as a combined math/engineering discipline. As background an undergraduate level of modern applied algebra¹ will suffice. Essential concepts and their engineering interpretation are introduced in a practical fashion with examples. The motivation in essence is: the importance of the unifying associative algebra of function composition (viz. semigroup theory) for the practical characterisation of the three main functions in computers, namely sequential logic (state-machines), arithmetic and combinational (Boolean) logic.

Known principles of discrete mathematics, especially finite semigroups, residue arithmetic and boolean logic (lattices) are interpreted in terms of practical *DN* design issues. The main three levels of state machine synthesis form a natural 'top down' hierarchy of associative algebras :

<i>Application</i>	<i>Algebra type</i>	<i>Syntax</i>	<i>Objects</i>	<i>Operations</i>
sequential logic	associative	$(ab)c = a(bc)$	functions	sequencing
arithmetic	commutative	$ab = ba$	numbers	(+) (.)
combinat'l logic	idempotent	$aa = a$	sets	\cup \cap

Historically, non-commutative and idempotent algebras diverged from arithmetic in the nineteenth century. Our aim is to emphasize again their arithmetic nature, for practical engineering purposes such as efficient synthesis of binary logic and state machines. The 'static' (combinational, idempotent $x^2 \equiv x$) and 'iterative' (commutative, $x^{i+1} = x^i x = x x^i$) aspects can be modeled by finite residue arithmetic. Apart from the two non-commutative components of memory type (branch- and reset- machines, shown to be each others dual), non-commutative aspects of sequential behaviour can be represented by coupling functions between components.

¹Birkhoff-Bartee 1970 - *Modern Applied Algebra*
Hartmanis-Stearns 1970 - *Algebraic Structure of Sequential Machines*

- In the first of three parts, on state machines (Ch.1-4), an introductory chapter recalls basic principles in theory and practice. The five basic components of sequential behaviour (with indecomposable semigroup) are derived, with ways to couple them efficiently - only required in the non-commutative case. They define the five basic *types* of state machines for network composition.

- In the second part, on combinational (Boolean) logic (Ch.5,6) the concept of *spectrum* as a characteristic sequence of numbers, is borrowed from Fourier analysis for order-independent (symmetric) synthesis of Boolean functions (*BFs*). A useful *arithmetic* compositional rule holds: the spectrum of a product of functions (of disjoint inputs) is the product of the component spectra. In fact Boole (1854) introduced his algebra - a calculus of binary properties - as an idempotent form of arithmetic. This allows convolution-like composition rules (as in linear filters), to be developed.

Symmetric *BFs* are implemented as a crossing-free and compact orthogonal grid network of *MOS* transistors in the silicon plane, to obtain a regularly structured *VLSI* implementation. Simply removing transistors from such grid yields *planar BFs* with the desired crossing-free property, covering a majority of Boolean functions. Using this representation, the complexity of *BFs* grows polynomial, and not exponential, with the number of inputs. It appears that by permuting and/or inverting the n inputs, each BF_n of at most four inputs is planar.

A fast $O(n^2)$ algorithm for symmetric logic synthesis is developed, and applied to optimize fault-tolerant logic using Hamming- or product- codes for error correction, with synthesized gate count as cost criterium.

- The third and last part, on arithmetic (Ch.7-11), analyses residue arithmetic with two extremal types of prime related moduli: p^k and $m_k = p_1 p_2 \dots p_k$ typical for 'sequential' resp. 'parallel' arithmetic. By expanding $r \bmod m$ residues with a 'carry' c as multiple of modulus m : $n = cm + r$, integer arithmetic obtains a dual focus on *closure*- and *generative* properties of residues and carry, as independent resp. dependent network components.

This balanced approach to arithmetic provides new insights into old and well known problems in finite *additive* number theory (Fermat, Goldbach, Waring: Ch.8,9,10) with practical engineering results. For instance each odd residue mod 2^k is a unique signed power of 3, allowing efficient log-arithmetic over bases 2 and 3 [patent US-5923888]. Moreover, a binary log-arithmetic microprocessor (32 bits, in 0.18μ *CMOS* technology) is described, designed as part of a European *Esprit* project², comparing favourably with the known floating point arithmetic devices.

Nico F. Benschop ♠ Amspade Research, Geldrop, The Netherlands, Oct. 2008.

Acknowledgements:

The author is grateful to Philips Research (Eindhoven, NL) for allowing to publish this material, developed during his 32 years there. Also gratefully acknowledged are the contributions, to chapters 6 and 11, of colleagues Richard Kleihorst, René van der Vleuten (Philips Research), G.Muurling, prof. J.Simonis (TU Delft), and of *Esprit* project co-workers Chris Softley, dr. Nick Coleman (Univ. Newcastle UK) and Rudolf Matousek, prof. Jiri Kadlec (UTIA, Prague).

²Esprit 33544 *HSLA*, 1999-2002, main contractor Univ.Newcastle (dpt.*ECE*) UK

Contents

– PART 1 – Sequential Logic: <i>Finite Associative</i>	1
1 Introduction	1
1.1 Sequential and combinational Logic	2
1.2 Five basic state machines, as network components	4
1.3 Subset/partition, local/global, additive/mult've	7
1.3.1 Associative closure: semigroup and sub-semigroup	7
1.3.2 Preserved partition: congruence and image	8
1.4 Integer arithmetic: residues with carry	10
2 Simple Semigroups and the Five Basic Machines	12
2.1 State Machine: Sequential Closure and Rank	12
2.2 Basic Machines and Simple Semigroups	14
2.2.1 Iterations : monotone, periodic, idempotent	15
2.2.2 Ordered idempotents H for combinational logic	15
2.2.3 The five minimal semigroups and basic machines	16
2.3 Equivalent idempotents: memory components L, R	18
2.4 Maximal Subgroups: periodic G	23
2.5 Constant Rank Machines, and simple semigroups	24
3 Coupling State Machines	28
3.1 Introduction	28
3.2 No coupling: semigroup $Z(\cdot) \bmod m$, composite m	29
3.3 Machine decomposition: right congruence suffices	34
3.4 Cascade composition: full groups FG_3 and FG_4	36
3.5 Decomposing the full- and alternating group over four states	41
3.6 Decomposing simple groups $AG_n \subset FG_n$ for $n > 4$	45
3.7 Loop composition superfluous	49
4 General Network Decomposition of State Machines	51
4.1 Introduction	51
4.2 Implementing $M = (Q, A)$ by its alphabet A	52
4.2.1 Decomposition by local input closures	53
4.3 Bottom-up rank driven decomposition of $S = A^*/Q$	53
4.4 Partial direct products, unused codes, efficiency	54
4.5 Example	54
4.5.1 Top-down decomposition by local input closures	57

4.5.2	Global decomposition by maximal iterative components	57
4.6	Invariants: ordered commuting idempotents	59
– PART 2 – Combinational Logic: <i>Commuting Idempotents</i>		69
5	Symmetric and Planar Boolean Logic Synthesis	69
5.1	Introduction	69
5.2	Logic Synthesis independent of input ordering	70
5.2.1	Orthogrid plot and rank spectrum	71
5.2.2	Factoring paths by a planar node	71
5.3	Symmetric and Threshold BF' s	73
5.3.1	Symmetric functions 'count'	73
5.3.2	T -cell library , threshold logic cells	74
5.4	Planar cut and factoring	75
5.5	Fast symmetric synthesis: quadratic in nr. inputs	76
5.6	Experiments and conclusion	77
5.7	Planar Boolean logic synthesis	77
5.7.1	All BF_n are planar upto $n=4$ inputs	78
6	Fault Tolerant Logic with Error Correcting Codes	85
6.1	Introduction	85
6.2	Fault tolerant IC design environment	86
6.2.1	Implementation at register transfer level	87
6.2.2	Protecting registers and connections	88
6.3	Three error correction methods for logic circuits	89
6.3.1	Majority voting	89
6.3.2	Hamming codes (block codes)	90
6.3.3	Product codes (array codes)	90
6.4	Demonstration of experimental circuit	91
6.5	Results for typical designs	96
6.6	Conclusions	99
– PART 3 – Finite Arithmetic: <i>Associative, commutative</i>		101
7	Fermat's Small Theorem extended to $r^{p-1} \bmod p^3$	101
7.1	Introduction	101
7.1.1	Divisors $r p \pm 1$ and residues $(p \pm 1)^p \bmod p^3$	103
7.2	Lattice structure of semigroup $Z(\cdot) \bmod q$	104
7.2.1	Distinct $e^{p-1} \bmod p^3$ for idempotents $e \in Z_{p-1}$	105

7.3	Distinct $r^{p-1} \pmod{p^3}$ for divisors $r p \pm 1$	107
7.3.1	Idempotents of $Z_{p+1}(\cdot)$ and divisors of $p + 1$	107
8	Additive structure of units group mod p^k, with carry extension for a proof of Fermat's Last Theorem	110
8.1	Introduction	111
8.2	Structure of the group G_k of units	112
8.3	Cubic root solution in core, and core symmetries	114
8.3.1	Another derivation of the cubic roots of 1 mod p^k	115
8.3.2	Core increment symmetry mod p^{2k+1} , asymmetry mod p^{3k+1}	116
8.4	Symmetries as functions yield 'triplets'	118
8.4.1	A triplet for each unit n in G_k	119
8.4.2	The <i>EDS</i> argument extended to non-core triplets	120
8.5	Relation to Fermat's Small and Last Theorem	122
8.5.1	Proof of the FLT inequality	122
8.6	Conclusions and Remarks	124
9	Additive structure of $Z(\cdot)$ mod m_k (squarefree) and Goldbach's conjecture	128
9.1	Introduction	128
9.2	Lattice of groups	130
9.2.1	Ordering of commuting idempotents	130
9.2.2	Lattice of idempotents: add vs join	131
9.3	Primes, composites and neighbours	132
9.3.1	Each idempotent's successor is in G_1 or G_2	133
9.4	Euclidean prime sieve	134
9.4.1	Pair sums of carry extended units	135
9.4.2	Induction base: pair sums of primes in $G(3)$	135
9.4.3	Excluding composites in $G(k)$, baseprimes and 1 as summands	137
9.5	Proving <i>GC</i> via <i>GR(k)</i> by induction on k	139
9.6	Conclusions	140
10	Powersums $\sum x^p$ represent residues mod p^k, from Fermat to Waring	142
10.1	Introduction	142
10.2	Core increments as coset generators	144
10.3	Core extensions: A_k to F_k , and pairsums mod p^k	145
10.4	Conclusions	150
11	Log-arithmetic, with single and dual base	152
11.1	Log-arithmetic with dual base 2 and 3	152

11.1.1	Proposed new binary number code	153
11.1.2	Example	154
11.1.3	Application to multipliers	155
11.1.4	Signed magnitude binary code over bases 2 and 3	155
11.1.5	Addition in log code: 'odd' arithmetic (base 2 and 3)	156
11.2	European Logarithmic Microprocessor <i>ELM</i>	158
11.2.1	Introduction: Log-arithmetic with single base 2	159
11.2.2	Log-arithmetic algorithms, an overview	160
11.2.3	Data format, range and precision	162
11.2.4	Measurement of Accuracy	163
11.2.5	Conventional <i>LNS</i> Addition and Subtraction	164
11.2.6	New Error Correction Algorithm	166
11.2.7	Error Correction for Subtraction	169
11.2.8	Adder/Subtractor design and evaluation	169
11.2.9	Architecture and performance	171
11.2.10	VLSI Implementation	172
11.2.11	The <i>ELM</i> : some more architectural details	173
11.2.12	Accuracy comparisons <i>LNS</i> vs. <i>FLP</i>	174
11.2.13	The TMS-320C6711	176
11.2.14	Conclusion	177
–	Bibliography	181
–	Index	186