

Computational RAM: Implementation and Bit-Parallel Architecture

by

Christian Cojocaru

**A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfilment of the requirements
for the degree of
Master of Engineering**

**Ottawa-Carleton Institute for Electrical Engineering
Department of Electronics
Carleton University
Ottawa, Ontario
Canada K1S 5B6**

January 1995

Copyright 1995, Christian Cojocaru



National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services Branch

Direction des acquisitions et
des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Author's Acknowledgement

Author's Acknowledgement

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-612-08957-6

Canada

Name CHRISTIAN COJOCARIU

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

COMPARATIONAL AND HISTORICAL AND

0544

U·M·I

SUBJECT TERM

SUBJECT CODE

22 PARALLEL ARCHITECTURE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0403

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
 General 0289
 Ancient 0290
 Linguistics 0291
 Modern
Literature 0401
 General 0294
 Classical 0295
 Comparative 0297
 Medieval 0298
 Modern 0316
 African 0591
 American 0305
 Asian 0352
 Canadian (English) 0355
 Canadian (French) 0593
 English 0311
 Germanic 0312
 Latin American 0315
 Middle Eastern 0313
 Romance 0314
 Slavic and East European

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
 General 0321
 Biblical Studies 0319
 Clergy 0320
 History of 0322
 Philosophy of 0469
Theology

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
 Archaeology 0326
 Cultural 0327
 Physical
Business Administration 0310
 General 0272
 Accounting 0770
 Banking 0454
 Management 0338
 Marketing 0385
Canadian Studies
Economics 0501
 General 0503
 Agricultural 0505
 Commerce Business 0508
 Finance 0509
 History 0510
 Labor 0511
 Theory 0358
Folklore 0366
Geography 0351
Gerontology
History 0578
 General

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science
 General 0615
 International Law and Relations 0616
 Public Administration 0617
Recreation 0814
Social Work 0452
Sociology 0626
 General 0627
 Criminology and Penology 0938
 Demography 0631
 Ethnic and Racial Studies
 Individual and Family Studies 0628
 Industrial and Labor Relations 0629
 Public and Social Welfare 0630
 Social Structure and Development 0700
 Theory and Methods 0344
Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
 General 0285
 Agronomy
 Animal Culture and Nutrition 0475
 Animal Pathology 0476
 Food Science and Technology 0359
 Forestry and Wildlife 0478
 Plant Culture 0479
 Plant Pathology 0480
 Plant Physiology 0817
 Range Management 0777
 Wood Technology 0746
Biology 0306
 General
 Anatomy 0287
 Biostatistics 0308
 Botany 0719
 Cell 0329
 Ecology 0329
 Entomology 0353
 Genetics 0369
 Limnology 0793
 Microbiology 0410
 Molecular 0307
 Neuroscience 0317
 Oceanography 0416
 Physiology 0433
 Radiation 0821
 Veterinary Science 0778
 Zoology 0472
Biophysics 0786
 General
 Medical 0760

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
 General 0300
 Audiology 0992
 Chemotherapy 0567
 Dentistry 0350
 Education 0769
 Hospital Management 0758
 Human Development 0982
 Immunology 0564
 Medicine and Surgery 0347
 Mental Health 0569
 Nursing 0570
 Obstetrics and Gynecology 0380
 Occupational Health and Therapy 0354
 Ophthalmology 0381
 Pathology 0571
 Pharmacology 0419
 Pharmacy 0572
 Physical Therapy 0382
 Public Health 0573
 Radiology 0574
 Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences
Chemistry 0485
 General 0749
 Agricultural 0486
 Analytical 0487
 Biochemistry 0488
 Inorganic 0738
 Nuclear 0490
 Organic 0491
 Pharmaceutical 0494
 Physical 0495
 Polymer 0754
 Radiation 0405
Mathematics
Physics 0605
 General 0986
 Acoustics
 Astronomy and Astrophysics 0606
 Atmospheric Science 0608
 Atomic 0748
 Electronics and Electricity 0607
 Elementary Particles and High Energy 0798
 Fluid and Plasma 0759
 Molecular 0609
 Nuclear 0610
 Optics 0752
 Radiation 0756
 Solid State 0611
Statistics 0463
Applied Sciences
Applied Mechanics 0346
Computer Science 0984

Engineering 0537
 General 0538
 Aerospace 0539
 Agricultural 0540
 Automotive 0541
 Biomedical 0542
 Chemical 0543
 Civil 0544
 Electronics and Electrical 0348
 Heat and Thermodynamics 0545
 Hydraulic 0546
 Industrial 0547
 Marine 0794
 Materials Science 0548
 Mechanical 0794
 Metallurgy 0551
 Mining 0552
 Nuclear 0549
 Packaging 0765
 Petroleum 0554
 Sanitary and Municipal System Science 0790
 Geotechnology 0428
 Operations Research 0796
 Plastics Technology 0795
 Textile Technology 0994

PSYCHOLOGY

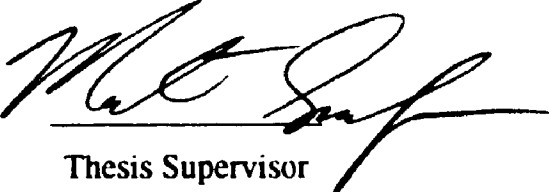
General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451



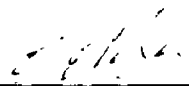
The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

*Computational RAM: Implementation
and Bit-Parallel Architecture*

submitted by Christian Cojocaru, Dipl.Eng.
in partial fulfilment of the requirements for
the degree of Master of Engineering



Thesis Supervisor



Chairman,
Department of Electronics

Ottawa-Carleton Institute for Electrical Engineering
Department of Electronics
Faculty of Engineering
Carleton University
January 9, 1995

Abstract

Computational RAM (C*RAM) was previously proposed as a low-cost Single Instruction Multiple Data computing architecture integrating simple processing elements (PE) into a semiconductor memory circuit. The work presented in this thesis advances the C*RAM project in two directions. First, a 64 processor array was designed and integrated with a 64 kbit ASIC memory module in a 0.8 μm triple-metal BiCMOS technology. Interdependent circuit and physical design have led to a dense PE array, taking 2.7% of the memory area. Functional C*RAM chips have been tested, allowing the project to continue with system and application work. Second, a novel bit-parallel oriented PE architecture is proposed and a 512 PE Parallel Computational RAM (PC*RAM) is designed and implemented. The new PE array can execute bit-parallel add/subtract, comparison and shift-and-add integer multiplication on programmable length words. All bit-parallel operations are performed without intermediate memory access, resulting in speed, power consumption and system interface advantages over the more traditional bit-serial SIMD approach.

Acknowledgments

Part of the direct financial support for this project was provided by Miconet, a National Centres of Excellence programme of the Natural Sciences and Engineering Research Council of Canada (NSERC). Integrated circuit fabrication was provided by Northern Telecom Electronics (NTE), both directly and through the Canadian Microelectronics Corporation (CMC). CMC also supplied design software and cell libraries. Test equipment was supplied by NSERC through their Industrial Research Chair programme. I would like to gratefully acknowledge the support of these organizations.

The collaboration with the Memory Design Group in Bell-Northern Research allowed this project to attain real-life dimensions. I would like to thank David Somppi for his support and Randall Gibson for very useful technical discussions.

Duncan Elliott from University of Toronto has introduced me to his Computational RAM world, with brief but dense and challenging e-mail messages. I hope his pioneering work will graduate someday into a real technology product.

I would like to mention here the very enjoyable atmosphere in the Department of Electronics at Carleton University, with too many friends to list. Dave Skoll served as a walking manual for the IC design software. Luc Lussier helped with test board design and examples on how to make things happen. Dr. Martin Lefebvre supervised my initial layout work. Thanks to all.

I had the great opportunity to have Dr. Martin Snelgrove as supervisor. His capacity to synchronize on my sometimes sketchy ideas on very short notice, understand them and suggest alternatives was essential in guiding this work in the right direction. I would like to thank Martin for his talent to communicate his vast electronics (but not only) knowledge, and for his admirable patience and generosity in helping his students.

Finally, I would like to express my heartfelt gratitude to my parents, Ecaterina and Dan. They have left their home in Romania to recreate its familiar atmosphere here, helping me to concentrate on my studies as they did for so many years.

Contents

1 Introduction to C*RAM	1
1.0 The C*RAM Concept	1
1.1 Conventional Computers and Integrated Circuits Technology	3
1.2 Parallel Computers - Brief Overview	3
1.3 Semiconductor Memory as an IC Category	5
1.4 C*RAM - An SIMD-IC Memory Hybrid	6
1.5 Computational RAM History	7
1.6 Thesis Contributions	7
1.7 Thesis Organization	8
2 SIMD and Semiconductor Memory	10
2.1 Single Instruction Multiple Data (SIMD) Computers	10
2.2 SIMD Machines Similar to C*RAM	12
2.2.1 The AIS-5000 Parallel Processor	13
2.2.2 The SRC Processor-In-Memory (PIM)	13
2.2.3 The NEC Integrated Memory Array Processor (IMAP)	14
2.3 SRAM and DRAM	16
2.3.1 Static Random Access Memory (SRAM)	16
2.3.2 Dynamic Random Access Memory (DRAM)	18
2.4 C*RAM Economics	20
2.4.1 DRAM Economics	20
2.4.2 C*RAM Area and Yield Issues	21

3 Processing Element Circuit and Physical Design	24
3.0 Introduction	24
3.1 Architecture of the Baseline Processing Element	24
3.1.1 Communication between PEs	26
3.1.2 Conditional Execution Support	28
3.1.3 Baseline PE Control Lines	29
3.2 Processing Element CMOS Circuit Design - Strategy	30
3.3 8-to-1 Multiplexor ALU	31
3.3.0 Circuit Alternatives	31
3.3.1 Logic and Physical Order of the Selection Lines	37
3.3.2 ALU Transistor Sizing	38
3.3.3 ALU Layout	40
3.4 ALU-Register Loop	43
3.5 Register Design	45
3.5.1 Shift-Left/Right Implementation	46
3.5.2 Dual-Access Latch	47
3.5.3 Dual-Access Latch Transistor Sizing	49
3.5.4 Single-Access Latch Transistor Sizing	51
3.6 Bus-tie Circuit Implementation	53
3.6.1 Circuit Alternatives	53
3.6.2 Proposed Bus-tie Circuit Implementation	56
3.7 Processing Element Layout	61
3.8 Summary	65
4 BATMOS Computational RAMs	66
4.0 Using a Platform RAM	66
4.1 BNR's Modular SRAM	67
4.1.1 SRAM Modularity Parameter for C*RAM	68
4.2 C*RAM Data Input/Output	70
4.2.1 Output Data Circuitry	75
4.2.2 Input Data Circuitry	76
4.2.3 Input/Output Buffers	77

4.3 C*RAM Instruction Multiplexing	78
4.4 Timing Generation	80
4.4.1 External Timing	80
4.4.2 Internal Timing Circuitry	81
4.4.3 Timing Generation in the 512 PE C*RAM	83
4.5 64 PE C*RAM Cycles	86
4.5.1 Memory Write Cycle	86
4.5.2 Memory Read and Page-mode Read Cycles	86
4.5.3 C*RAM Operate Cycle	87
4.5.4 Read-Operate Cycle	87
4.5.5 Operate-Write Cycle	88
4.5.6 Read-Operate-Write Cycle	88
4.6 Synopsis of BATMOS Computational RAMs	89
5 The Bit-Parallel Oriented Processing Element	91
5.0 Motivation	91
5.1 Programmable Segments Bus-tie (PSB) Network	92
5.1.1 Use of the Programmable Segments Bus-tie	94
5.1.2 Circuit Implementation	96
5.1.3 PSB Simulation	98
5.2 Shift Operations with Word Boundaries	101
5.2.1 Circuit Implementation	103
5.3 Dedicated Add/Subtract Ripple-Carry Circuit	104
5.3.1 Ripple-Carry Circuit Implementation	107
5.3.2 Ripple-Carry Chain Simulation	109
5.4 Bit-Parallel Integer Multiplication with the Extended PE	113
5.4.1 Extended Register Set	113
5.4.2 Bit-Parallel Multiply Example	117
5.4.3 Signed Multiply	120
5.4.4 Bit-Parallel Multiply Execution Time	121
5.5 System Advantages of Bit-Parallel C*RAM	122
5.6 Summary	124

6 C*RAM Testing	125
6.0 Introduction	126
6.1 Test Setup Hardware	126
6.1.1 C*RAM Test Board	128
6.2 Test Software	129
6.2.1 C*RAM Testing with the CRAMTest Program	130
6.2.2 Automatic Memory Testing	131
6.3 Test Results	131
7 Proposed Developments and Conclusions	132
7.1 Independent PE Addressing	132
7.1.1 Proposed Solution - Dual-Mode Column Decoder	133
7.2 Future C*RAM Work	134
7.2.1 Processing Element and C*RAM Architecture	135
7.2.2 C*RAM System Hardware	135
7.2.3 C*RAM Software	136
7.3 Conclusions	137
Bibliography	138
Appendices	142
A C64p1k Schematics	142
B C64p1k Pinout	157
C C64p1k Test Board Schematic	158

List of Figures

1.1(a) Random Access Memory (RAM)	2
1.1(b) Computational RAM (C*RAM)	2
1.2(a) Conventional computer	2
1.2(b) C*RAM enhanced computer	2
2.1 Example SIMD architecture	11
2.2 SIMD PE intercommunication networks	11
2.3(a) Full CMOS SRAM cell	16
2.3(b) HRL SRAM cell	16
2.4 Generic SRAM structure	17
2.5 DRAM core cell	18
2.6 Generic DRAM (after [Taka93])	19
3.1 Baseline PE (BPE) architecture	25
3.2 Two ALUs executing the XOR3 function	26
3.3 Shift-right data flow	27
3.4 Bus-tie data flow	28
3.5 Conditional write-back to memory using the W register	29
3.6 MUX8 static CMOS implementation	32
3.7 DYN5N dynamic logic implementation of the 8-to-1 multiplexor	33
3.8 Spice simulation of DYN5N	34
3.9 DYN4N dynamic implementation of MUX8	35
3.10 Pseudo-dynamic DYN4N with latch-back PMOST (<i>PLATCH</i>)	36
3.11 ALU operation cycle	37

3.12(a) ALU propagation delay function of NMOST width w_{nmx8}	39
3.12(b) ALU discharge evaluation time function of w_{nmx8}	39
3.13 ALU precharge, discharge and total energy as function of w_{nmx8}	40
3.14 ALU floor-plan	41
3.15 ALU multiplexor layout	42
3.16 An example of an CMOS edge-triggered D flip-flop	43
3.17 CMOS D-latch, requiring 10 transistors	44
3.18 Timing of ALU operation, followed by register write	45
3.19 Final circuit for single-access D-latch	46
3.20 Dual-access D-latch	48
3.21 A symmetric dual-access D-latch	48
3.22 Writing a "1" then a "0" in the dual-access latch (YR waveform)	50
3.23(a) LOW-HIGH latch write time t_{WLH} function of access NMOST width	50
3.23(b) Energy spent in latch state reversal	50
3.24(a) LOW-HIGH latch write time t_{WLH} function of write driver size	51
3.24(b) Energy spent in latch reversal function of write driver size (NMOST width)	51
3.25 Layout of X and Y registers in two adjacent PEs	52
3.26 Symbolic diagram of Bus-tie circuitry	53
3.27 Global wired-OR with precharged bus	54
3.28 Bidirectional Bus-tie with transmission gates	55
3.29 Active Bus-tie circuit proposed by Elliott	56
3.30 Proposed compact Bus-tie circuit	57
3.31 Array connection of 64 Bus-tie circuits	58
3.32 Spice simulation of worst-case 64 PE Bus-tie cycle	59
3.33 Bus-tie circuit layout in two adjacent PEs	60
3.34 Processing Element layout floor-plan	62
3.35 Dual Processing Element layout	64
4.1 BNR's synchronous SRAM timing	67
4.2 SRAM organization for C*RAM (C64p1k)	69
4.3 1 bit memory interface	70
4.4 64 PE C*RAM Input/Output organization	71

4.5	Logic function of I/O Data Decoder	72
4.6	Practical arrangement of the I/O Decoder	73
4.7	Decoding section in the I/O Decoder	74
4.8	Data output circuitry in the I/O Decoder	75
4.9	Data input circuitry in the I/O Decoder	76
4.10	Input/Output buffers	77
4.11	TTOP register and associated enable/driver AND gates	79
4.12	Example of C*RAM specific timing signals	81
4.13	Timing Generator PE standard cycle	84
4.14	Timing Generator Bus-tie cycle	84
4.15	Timing Generator electrical schematic	85
4.16	C*RAM Write cycle	86
4.17(a)	C*RAM Read cycle	86
4.17(b)	C*RAM Page-mode Read cycle	86
4.18	C*RAM Operate cycle	87
4.19	C*RAM Read-Operate cycle	87
4.20	C*RAM Operate-Write cycle	88
4.21	C*RAM Read-Operate-Write cycle	88
4.22	64 PE Computational RAM floor-plan	90
5.1	Bus-tie circuit in the Baseline PE	93
5.2	The Programmable Segments Bus-tie (PSB) Network	93
5.3	PSB circuit implementation within a PE	97
5.4	PSB Network connections	98
5.5	Programmable Segments Bus-tie network simulation	99
5.6	64 PE PSB circuit	100
5.7	Global Shift-right data flow	101
5.8	A restricted configuration for Segmented Shift	102
5.9	Introducing shift boundaries while conserving the Global Shift	102
5.10	Flexible implementation of Segmented Shift	103
5.11	Logic gate multiplexor	104
5.12	Transmission gate multiplexor	104

5.13	Carry block in the Extended PE	106
5.14	Carry circuit schematic	108
5.15(a)	A chain of 8 Carry blocks	111
5.15(b)	Detail of Carry chain circuit	111
5.16	Carry chain Spice simulation	111
5.17	Layout of Carry blocks in adjacent PEs	112
5.18	Data registers in the Extended PE	113
5.19	Extended PE (XPE) architecture	115
5.20	Dual (Left-Right) Extended PE layout	116
5.21	Bit-serial vs. Bit-parallel C*RAM systems	122
5.22	32-bit memory subsystem with x4, x8 and x16 DRAMs	123
6.1	C*RAM test setup	128
7.1	Column decoder controlled by PE registers in an SRAM C*RAM	133

List of Tables

2.1	Comparison of BATMOS C*RAMs and CRAM-like SIMD machines/ICs	15
3.1	Examples of logic functions of 3 variables	25
4.1	Platform SRAM Modularity Range	68
4.2	Synopsis of BATMOS C*RAM chips	89
5.1-5.8	Bit-Parallel Multiplication Example	118

Abbreviations and Terms

ALU = Arithmetic and Logic Unit

ASIC = Application Specific Integrated Circuit

BiCMOS = Bipolar and CMOS technology

BPE = Baseline Processing Element

CMOS = Complementary Metal-Oxide-Semiconductor

C*RAM = Computational Random Access Memory

CS = Carry Save; a redundant way of representing numbers with 2 bits per digit

DRAM = Dynamic RAM

FP = Floating Point

GPIB = General Purpose Interface Bus; another name for the HPIB

HPIB = Hewlett-Packard Interface Bus: an interface for programmable instruments

LSB = least significant bit

MIMD = Multiple Instruction Multiple Data

MOST = Metal-Oxide-Semiconductor (MOS) transistor

MSB = most significant bit in a multibit word (by convention here in the leftmost position)

PCB = Printed Circuit Board

PC*RAM = Parallel Computational RAM

PE = Processing Element

RAM = Random Access Memory

SA = Sense Amplifier: circuit that amplifies a reduced differential voltage from the memory core into full logic levels.

SIMD = Single Instruction Multiple Data

SIMM = Single In-line Memory Module; a compact way of mounting multiple memory chips on a small PCB.

SRAM = Static RAM

V_{th} = MOS transistor threshold voltage

Chapter 1

Introduction to C*RAM

1.0 The C*RAM Concept

Computational Random Access Memory (abbreviated in the following as C*RAM) is an integrated circuit which adds a large number of minimal processing elements to a semiconductor memory array, Figure 1.1 [Elli92]. A hypothetical commercial C*RAM would have 2048 Processing Elements (PE), each having 100-150 transistors, integrated in a 16 megabit Dynamic RAM. The objective is to enhance the memory subsystem of a computer with low-cost local computing power, while conserving its conventional storage function, Figure 1.2. The main advantages of this computing structure are:

- a. it uses the wide internal bandwidth inherent in high-density memories, currently wasted or sparsely used;
- b. avoids or minimizes the power dissipated in processor-memory data transport by keeping applicable computation inside memory, and
- c. it is an incremental change of the well-established conventional computer.

The processing elements are organized in a Single Instruction Multiple Data (SIMD) linear array, the most economical in terms of silicon area. This memory localized computing structure has a data parallel nature and maps well to applications in the area of image and video processing, recently gaining more importance in the workstation/personal computer market. A large number of other applications can benefit from the fine grain parallelism of C*RAM: scientific (numerical)

computation, database search, data compression, neural and genetic algorithms. In order to achieve production yields and market pricing comparable to semiconductor memory, the extra silicon area taken by the processing elements has to be small, estimated at maximum 20% of the memory chip.

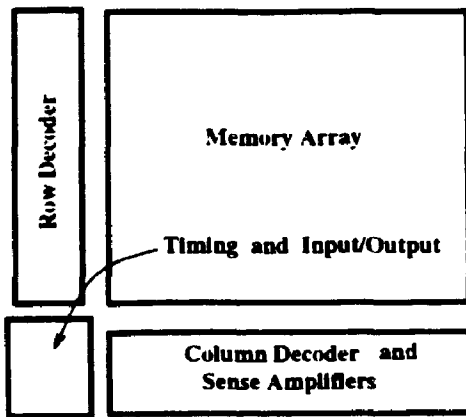


Fig.1.1(a) Random Access Memory (RAM)

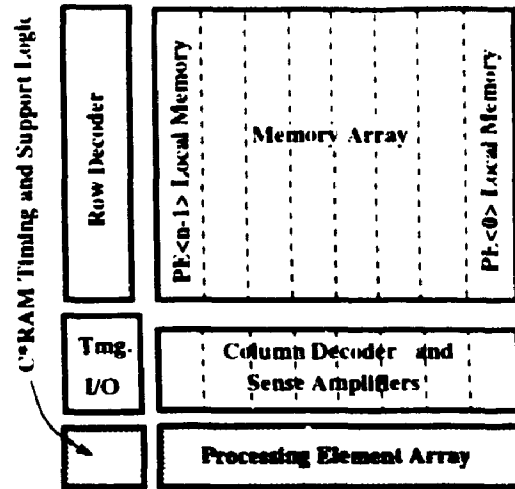


Fig.1.1(b) Computational RAM (C*RAM)

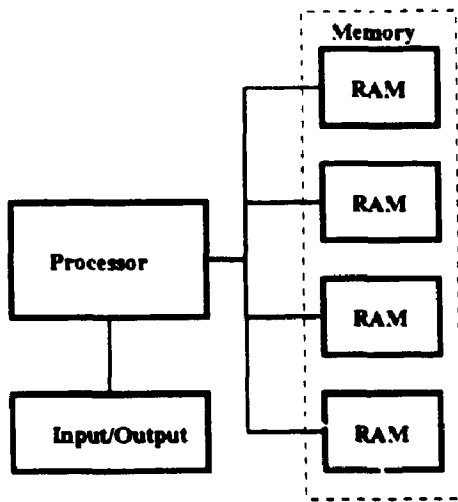


Fig.1.2(a) Conventional computer

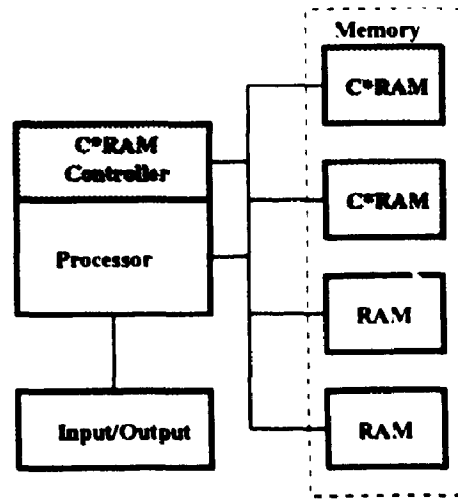


Fig.1.2(b) C*RAM enhanced computer

1.1 Conventional Computers and Integrated Circuits Technology

The vast majority of computers in use today are organized around the structure called the Von Neumann architecture (also referred to as "conventional computer"). This structure has two main functional blocks, the processor and the memory. The processor does the computation (logic and arithmetic) and system control functions, executing instructions and working on data stored in the memory. Both instructions and data are in binary format and are identified by their address in the memory. The processor does the address housekeeping, i.e. keeps track of what kind of information is stored at which address, in an otherwise uniform binary space.

The tremendous advances in semiconductor integrated electronic circuits in the last three decades have resulted in million-transistor processors being integrated in a single chip (microprocessors) and have allowed the memory to attain single-chip capacities of 256 megabit. While chip densities have increased for both elements, the microprocessor speed of operation has advanced much more than that of integrated memories. The original communication channel between processor and memory has become more and more of a bottleneck, defined by its bandwidth in bytes/second. The economics of memory pricing have dictated the most significant variation on the von Neumann structure, namely the introduction of a fast (but expensive) small memory named "cache" between the fast microprocessor and the slow (but inexpensive) large main memory. A processor+memory single chip system has not become a viable alternative for general purpose computers because programs have surpassed hardware advances by demanding even more instruction and data memory.

1.2 Parallel Computers - Brief Overview

Functional limits are on the horizon for the conventional structure. Processing speed is the main limit, bringing with it power and thermal constraints. Processor clocking frequencies have attained today (end 1994) 200 MHz for mainstream silicon technology (DEC Alpha, [IESp93]). Beside the speed gap between processors and memory, there are growing technological problems associated with increasing clock frequencies, especially in the areas of signal integrity and heat dissipation [Schu93].

One solution to these limits is parallel processing [Alma93], [PRI491], [Patt94]. A parallel computer has multiple processors executing instructions simultaneously. A vast number of structures can be described in this manner. An early classification was made by Flynn [Flynn66] and has the categories:

1. Single Instruction-stream Single Data-stream computers (SISD), the conventional architecture.
2. Single Instruction-stream Multiple Data-stream (SIMD): all processors execute the same instruction.
3. Multiple Instruction-stream Multiple Data-stream (MIMD): the processors execute independent program flows, processing different data.

Computers with multiple processors or execution units have been built under the generic categories of superscalar architectures, vector supercomputers, massively parallel processors or Very Long Instruction Word (VLIW). Superscalar is a characteristic associated with the latest generation of microprocessors, and refers to the simultaneous execution of two to four instructions, typically arithmetic operations. Vector supercomputers (e.g. the Cray machines) have an intermediate number (e.g. 4 to 64) of very powerful processing units and typically use exotic technologies (e.g. gallium arsenide logic chips, liquid Freon cooling). In a similar vein, recent generations of mainframe computers have multiple processors (4 to 8) and use non-mainstream technologies (bipolar emitter coupled logic, special cooling) [IESp89]. In recent years, a number of massively parallel computers (MPC) have been developed by clustering tens or hundreds of conventional microprocessors with a memory system. These MPCs have been built in both SIMD and MIMD categories.

While building the hardware proved feasible for a majority of machines, the clear problem that appeared in the process was defining the software (operating system, languages, compilers) that would make efficient use of the machine. The problem is especially difficult for MIMD machines, where multiple program threads have to use common resources and exchange data. The result of the expensive hardware and of the non-standard software is that the parallel machines occupy a very high-end market niche (although a growing one), far from the popularity of workstations or personal computers.

1.3 Semiconductor Memories as an Integrated Circuits Category

The semiconductor technology advances that lead to the integration of multi-million transistor processors have also created a number of specializations within the Integrated Circuits (IC) industry, with fairly distinct technical cultures. There is first a distinction between full-custom ICs, like processors and memory, and Application Specific ICs (ASICs). The ASICs are typically built by customizing the metal interconnect layers overlaid on a standard pre-designed transistor structure, called gate array. There is a large variety of other ASIC design methods, including standard cell, sea-of-gates, and in the last decade, Field Programmable Gate Arrays (FPGA).

In the full-custom category, the specialization occurred between the random logic ICs, such as the microprocessors and the semiconductor memory. Within the latter there are distinctive categories, but the largest market segment is the Random Access Memory (RAM), with two different technologies, the Static RAM (SRAM) and the Dynamic RAM (DRAM). Due to the difference in the basic memory cell, DRAM offers four times more storage capacity than SRAM, but is also four to six times slower. Because DRAM prices are four to six times lower than equivalent SRAM, the main memory in the workstation and personal computers is built with DRAM. The high end computers (mainframes, supercomputers) use SRAM for the main memory. The big market for SRAMs though is as cache memory. A cache memory bridges the speed gap between the fast microprocessors (90 MHz for Intel Pentium used in today's high end personal computers) and the cost-effective but slow DRAM main memory (e.g. 70 ns access time, 120 ns cycle time).

Both SRAM and DRAM share the block structure in Figure 1.1(a). Details on the differences are given in Chapter 2. The memory core (the actual storage area) is organized in rectangular blocks, composed of rows and columns. In each memory read access, a row of information is amplified to full logic levels by the row of sense amplifiers. Out of the large number of bits amplified (e.g. 2048 in a 4 Mb DRAM), only a small fraction (e.g. 1, 4, 8) are selected as chip output, transmitted on the system bus and used for processing by the Central Processing Unit (CPU, i.e. the microprocessor). The C*RAM concept is built on the intention to use the large internal memory bandwidth in order to do SIMD computation within the memory chip.

1.4 C*RAM - An SIMD-IC Memory Hybrid

The C*RAM concept tries to establish a common point between the architecture intensive SIMD machine world and the circuitry intensive SRAM/DRAM memory world. The C*RAM starts from the memory structure and builds an SIMD machine by adding processors along one of the memory array dimensions. All trade-offs are resolved in favor of keeping the resulting structure as close as possible to memory, in terms of area and system interface. The reason is that C*RAM is intended to substitute part of the main memory (as shown in Figure 1.2(b)) with the goal to add low-cost SIMD capability to conventional computers. Computation that maps well to the SIMD model benefits thus not only from an extremely large bandwidth, but also from the low-power allowed by keeping data transport on chip.

In view of its higher density and lower cost, DRAM is assumed to be the platform of choice for building C*RAM. Since DRAM functions by periodically regenerating the information stored in the core, it offers a higher internal bandwidth than SRAM. Large capacity SRAMs could be used for more specialized or higher end machines, or as a stage in building system and applications experience.

The main problem that faces C*RAM is the economics of high-density memories, discussed in Chapter 2. Technical issues in a C*RAM system are the gap between memory input/output speed and the internal computing power (I/O vs. internal bandwidth), and the rather small size of local memory. The forces at work for standard DRAM are also addressing these C*RAM problems. First, faster interfaces, such as Synchronous DRAM, are becoming more popular. As an alternative, the already established second serial access port (similar to the Video DRAM common in performance video buffers) could be used in high-end systems. Secondly, the local memory size issue has still to be decided by real applications, while ever increasing DRAM densities move the limit higher.

As is usually the case in technology history, fairly recently it surfaced that similar ideas are being developed by a number of groups in the world. Some of the closest concepts are the Processor-in-Memory (PIM) developed at the Supercomputing Research Centre in U.S.A. and the Nippon Electric Company Integrated Memory Array Processor (IMAP) chip. These chips are presented in

more detail in Chapter 2.

1.5 Computational RAM History

This section briefly presents the efforts made by the Canadian group in the field of **integrating an SIMD-memory hybrid**. The first SIMD experiments were made by Snelgrove and Loucks at University of Toronto in 1978, with a machine called VASTOR [Louc80]. Efforts to **integrate VASTOR** were hampered by the crudeness of tools/technology available in the academic environment. A 64 Processing Elements 8 kb SRAM based C*RAM prototype in 1.2 μm CMOS technology was designed by Duncan Elliott, at University of Toronto, in 1989, using a simplified PE (no interprocessor communication). In 1990 Elliott started designing a DRAM based C*RAM in cooperation with Mosaid Inc, a project mentioned in [Elli92], but the foundry partner upgraded its process and the project was abandoned.

1.6 Thesis Contributions

There are two main directions where this thesis advances the C*RAM project. First, a 64 Processing Element C*RAM part is designed and implemented in the most advanced technology available in Canada at this time, Northern Telecom's 0.8 μm BiCMOS, using a 64 kb memory module made available by Bell-Northern Research. One major thrust within this implementation was the design of an extremely area-efficient processing element, that demonstrates what the limits are in adding PEs to ASIC memories. On the same level with the compactness criterion was the design of PE circuitry safe-guarded against potential technology hazards, such as leakage currents. The overall goal of this implementation was to obtain functional parts, allowing the C*RAM project to continue with building a system and hence insuring the "reality" of the concept in the eyes of possible applications partners.

Secondly, this work explores novel structures of C*RAM processing elements. Although Elliott's Baseline PE is very simple and hence allows for many variations/enhancements, the proposed Extended PE attempts to maximize the arithmetic power of the PE while maintaining a close control of the silicon area expenses required. The Baseline PE simplicity was a result of its DRAM target. The C*RAMs described here are based on ASIC SRAM, which, due to its more relaxed

area constraints, lead to extending the PE functionality. The Extended PE could be adopted on a DRAM platform, with a corresponding loss in processor count.

A first enhancement to the Baseline PE is the so called "Programmable Segments Bus-tie" (PSB), which is implemented in a second C*RAM chip, still with 64 PEs but a smaller 512 bit local memory (referred to as Cs64p512). This chip also demonstrates a novel system oriented C*RAM feature, the integration of a RAM serial access port.

The original bit-parallel oriented PE array is designed into a third C*RAM chip, with 512 PE and 480 bit local memory (C512mp512). Beside the Programmable Segments Bus-tie, the new PE introduces support for Programmable Word Boundaries (PWB) and a hard-wired ripple-carry chain, resulting in flexible word length bit-parallel operation. The Extended PE demonstrates the minimal hardware structure capable of performing bit-parallel integer multiplication entirely within the PE array, i.e. without intermediary memory access, resulting in speed, power and system interface advantages over the classical bit-serial approach. This final chip (currently in fabrication) also includes a tighter timing circuit, close to what a commercial chip is expected to feature. In general, the interface of this chip reflects the experience accumulated during this work, including a first attempt to design a C*RAM subsystem.

1.7 Thesis Organization

Chapter 2 gives first an overview of SIMD machines and discusses projects similar to C*RAM. It presents then more details on the SRAM and DRAM memory categories and their suitability as starting platforms for building C*RAM. The economic aspects of memory and C*RAM are presented at the end.

Chapter 3 begins by presenting the architecture of the Baseline Processing Element (BPE), defined by Elliott. Most of the chapter is dedicated to the detailed CMOS circuit and physical design of the BPE.

Chapter 4 describes the design of the Computational RAMs implemented in Northern Telecom BiCMOS technology (BATMOS). The chapter introduces the industrial family of modular SRAM

used as C*RAM platform, then presents the C*RAM specific logic blocks, including the Input/Output circuitry, timing generation and C*RAM instruction multiplexing. Detailed timing diagrams of C*RAM cycles are shown at the end of the chapter.

Chapter 5 introduces the original Bit-Parallel Processing Element architecture and presents the arguments in favor of bit-parallel oriented C*RAM. A detailed presentation of the circuit modifications and additions to the Baseline PE is followed by a step-by-step description of a bit-parallel integer multiply example, as executed by the new PE architecture.

Chapter 6 presents the experimental results. The BATMOS C*RAM hardware and software test environment is described here.

Chapter 7 discusses ideas that could be implemented in future generations of C*RAM, focussing on adding independent PE addressing with a minimum of area overhead. The chapter ends with a summary of this work and with a discussion of possible developments of the C*RAM project.

Chapter 2

SIMD and Semiconductor Memory

This chapter will present in more detail the two technical categories C*RAM is based on, SIMD machines and semiconductor integrated memories, together with the adaptations required of both in order to integrate them into the Computational RAM SIMD-memory hybrid.

2.1 Single Instruction Multiple Data (SIMD) Computers

SIMD machines consist of multiple processing elements (PEs) executing the same instruction, each PE acting on different data stored in its local memory space, Figure 2.1, [Hord90], [Alma89]. Compared to their classic counterpart, MIMDs, these machines offer a much more tractable software environment and the hardware is easier to build. The reason is that the single program flow does not need the hardware and software complexity required to synchronize the MIMD multiple instruction and data flows. SIMD is best mapped to the “data parallel” programming model, for which the same processing sequence is applied to a large set of data. Most SIMD machines built so far have used large numbers (hundreds or thousands) of very simple processors, being referred to as Massively Parallel SIMD.

In general, when examining an SIMD machine, the following characteristics should be identified:

1. Complexity of the individual PE.
2. PE intercommunication network.

3. Independent local memory addressing capability.
4. Local memory size and PE-memory integration level.

PE complexity can be characterized by data word length, number of internal registers, functionality of the arithmetic and logic unit (ALU).

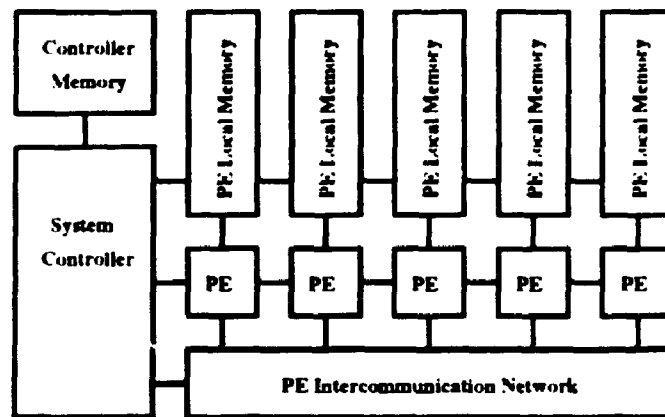


Figure 2.1 Example SIMD architecture

One important criterion that differentiates between SIMD machines is the intercommunication network between processors. Some typical examples are linear array, mesh (bidimensional array), tree network and hypercube, of which the first three are shown in Figure 1.4 a,b,c respectively. The linear array network is the simplest, with one PE connected to just two neighbors. The mesh array arranges the PEs in a bidimensional configuration, with one PE connected to its four neighbors.

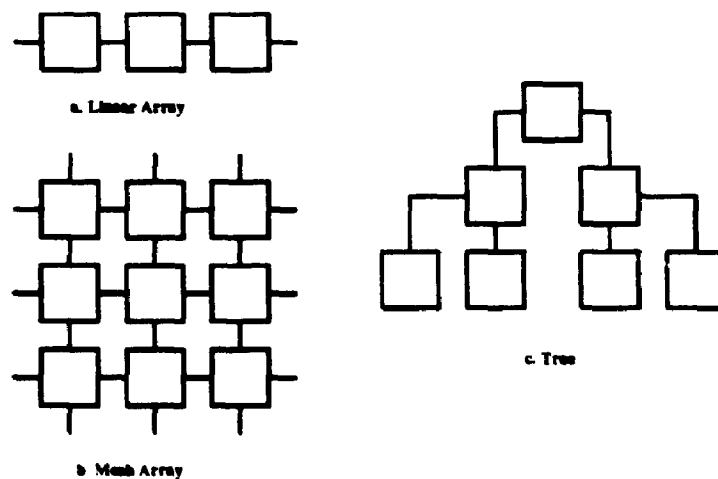


Figure 2.2 SIMD PE intercommunication networks

The hypercube network extends the spatial cube arrangement to n-dimensional (above 3 dimensions) spaces, each PE being connected to n others. The various connection networks try to anticipate and minimize the communication costs of certain classes of application programs. By providing increased complexity in the communication network, the architecture designers drive up the complexity and cost of hardware, in terms of board interconnections and/or in terms of silicon area. To date, the most commercial success in the SIMD category has been associated with the rather simple mesh-connected arrays (DAP, MPP).

The linear array and mesh networks are well suited for VLSI integration because of their regularity. C*RAM attempts to adapt a PE array to a memory structure, and, due to the minimal supplementary area constraints discussed in section 2.4, the linear array is the best choice. The PEs (assuming no interconnection) are forced to match the unidimensional arrangement of the sense amplifier array. A PE intercommunication network has to be then unfolded and mapped to the rectangular area adjacent to the PE array. The left-right communication is the natural choice for the linear array, taking only two routing channels per link. More complicated communication networks will require more routing channels, exponentially more routing channels for the higher-order networks. When the network has to be extended between chips, the linear and mesh networks are the only ones to extend with a low and constant number of pins, an important factor in setting the chip cost.

2.2 SIMD Machines Similar to C*RAM

The last twenty years have seen a number of massively parallel SIMD machines being developed. Some of them, like MPP (Massively Parallel Processor) from MasPar, DAP (Distributed Array Processor) from Active Memory Technology and Connection Machines (CM-2, CM-5) from Thinking Machines Corp. are currently in commercial use. These machines demonstrate that there is a market for applications that are handled well by the SIMD computing paradigm. On the other side, these SIMD machines target a niche high-end market. They are typically large and expensive, because they use simple processors in low integration-level chips separated from the local memory. Their computing power is related to the number of processors, hence more of them means more PE chips, more memory chips, more boards and interconnections, driving the final cost high. Using separate local memory, the number of PEs integrated on a chip becomes limited by the number of package pins required for the memory interface.

This section will present three architectures similar to C*RAM, in order to show the extent of interest in this class of computers and to present classes of applications. The SIMD literature is much richer in machine references [PRI491], but we restrict the discussion to those architectures that use a linear organization of the PE array with unidimensional communication networks. The assumption here is that the nature of the intercommunications network has a major impact on how applications are written for a particular machine and on the performance of these applications. The AIS-5000 was chosen because it is highly similar to the C*RAM Baseline PE array, while the PIM and IMAP were chosen because they integrate the PEs with their local memories into a single chip, coming very close (PIM especially) to the basic C*RAM idea.

2.2.1 The AIS-5000 Parallel Processor [Smit88]

This is a commercial SIMD machine which has 1024 single-bit PEs in the maximum configuration. While the PEs are physically organized in a linear array, the system software can emulate a bidimensional array. The main difference to C*RAM is that the PEs are not integrated in the memory. Instead, gate array ASICs with 8 PE/chip are connected to commodity SRAM chips. The authors claim that separate PE and memory is an advantage because it permits use of the low-cost standard SRAM. The unspecified disadvantage though is physical size: the maximum configuration of the machine has 8 printed circuit boards, with associated high costs and reliability problems. The PE is built around a 16-to-1 multiplexor, similar to the C*RAM (which has an 8-to-1 multiplexor, Chapter 3) and other single-bit SIMD machines.

The AIS-5000 is used commercially in machine vision tasks, but in [Smit88] other general purpose functions are evaluated (content addressable memory, bit string classifier, vector correlation). The designers of the machine demonstrate that for many image-based algorithms in use, the linear (unidimensional) array is superior in price/performance to the popular mesh arrays.

2.2.2 The SRC Processor-in-Memory (PIM) [Gokh92]

An integrated SIMD project developed at the Supercomputing Research Centre (SRC) in Maryland, USA, the PIM is a chip that integrates 64 PEs with 2 kb local memory. Again, the PEs are

connected in a linear array. The PE structure is more complex than the baseline C*RAM PE, by an estimated factor of 4-5, complex enough to admit a pipelined operation. There are three local registers and an ALU that generates three logic functions. The PIM PEs execute one command every 70 ns. Interprocessor communication is performed through a so called "parallel prefix network", a global OR function and a programmable-partition OR (POR) network. The parallel prefix network allows PEs at fixed positions to send data to groups of neighboring PEs. The programmable-partition OR network can execute logic-OR on power of 2 groups of PIM chips. The PIM to system interface is a 4-bit memory data bus, inherited apparently from the SRAM the chip was built upon. A number of PIM chips have been designed into a single-board SIMD subsystem that plugs into the system bus of a SUN workstation.

The applications discussed are cross-correlation and prime number searching, but the machine was probably designed for intensive cryptography analysis problems, as suggested by the affiliation of SRC to the Institute for Defense Analyses. A very recent development (Sept.1994) [EET816] has Cray Computer Corp. announcing they will use an extended PIM chip for a CRAY machine memory system, the project being sponsored by the U.S. National Security Agency.

The PIM project and its recent development are very close to C*RAM in both basic concept - small area PE array added to memory - and system organization. One of the important differences is that the SRC team does not mention DRAM as a possible platform for the PIM chip.

2.2.3 The NEC Integrated Memory Array Processor (IMAP) [Yama94]

This chip was introduced at the 1994 ISSCC by researchers from Nippon Electric Company, targeting real-time low-level image processing applications. The IMAP integrates 64 8-bit processing elements with a 2Mb SRAM, the highest level of integration recorded for C*RAM-like chips. The main peculiarity of the IMAP chip is the fact that each PE can independently address a 256-word memory page in the local memory. This feature is demonstrated as important for certain operations (image histogram, Hough transforms), where it substantially reduces execution time over an architecture without independent addressing. The PE structure is closer to typical 8-bit microprocessors (18 register file, 8-bit ALU, shifter block, multiplier look-up table) hence it is much more complex than a C*RAM PE. This PE results in an 50% memory-50% processors bal-

ance in IMAP (as suggested by the chip microphotograph). The PE clock is 40 MHz while the SRAM speed is 80 MHz. The chip has block redundancy, and is implemented in an aggressive 0.55 μm BiCMOS technology with SRAM enhancements. Although not explicitly mentioned in the paper, the independent PE addressing feature must require replicating a row decoder block for each local memory, because the cited 128 columns per row are not sufficient to supply a 256 word page. The resulting area overhead could be acceptable for a 50%-50% processor to memory balance, but not for a 10%-90% hybrid that targets replacing memory, as is the case of C*RAM.

Table 2.1 summarizes characteristics of C*RAM and of the architectures presented above. Also included in Table 2.1 are the two most representative C*RAM chips designed for this work, discussed in detail in Chapter 4. In the table, the PE complexity entry refers to the relative PE transistor count, with the Baseline C*RAM PE discussed in Chapter 3 as reference. When the actual number was not available, an estimate was made based on the PE functional description. Similarly, the communication network complexity refers to the number of inter-PE channels, relative to the Baseline PE.

Table 2.1 Comparison of BATMOS C*RAMs and C*RAM-like SIMD machines/ICs

	C*RAM C512mp512	C*RAM C64p1k	AIS-5000	SRC PIM	NEC IMAP
Memory-integrated PEs	yes	yes	no	yes	yes
PEs/chip	512	64	8	64	64
Word length (bits)	1 or 2n, n>0	1	1	1	8
PE count • Word length	512	64	8	64	512
Local Memory (bits)	480	1024	32 kb	2048	32 kb
PE Cycle Time (ns)	15	20	100	70	25
PE Rel. Complexity	2	1	3 (est)	5 (est)	92
Network Rel. Complexity	1.5	1	1 (est)	4 (est)	16
External I/O Width	8	8	4	4	4
Redundancy	no	no	no	no	block
Design/Technology	FC-ASIC	FC-ASIC	GateArray	-	SRAM

2.3 SRAM and DRAM

2.3.1 Static Random Access Memory (SRAM)

A block-level diagram of an SRAM is shown in Figure 2.4. The basic memory cell, represented symbolically as two back-to-back inverters, is represented at the transistor level in Figure 2.3. The SRAM cell is a positive feed-back structure (latch) with two stable states. There are three different implementations for the cell, depending on what kind of elements are used as the cell loads: diffused (bulk) PMOS transistors (in a standard CMOS process), Figure 2.3(a), high resistivity polysilicon loads (also called giga-ohm poly), Figure 2.3(b), or thin-film PMOS transistors deposited over the rest of the devices in the cell. The last two alternatives are process enhancements used specifically for SRAM design in order to obtain compact cell layout, and are not available in general-purpose ASIC processes.

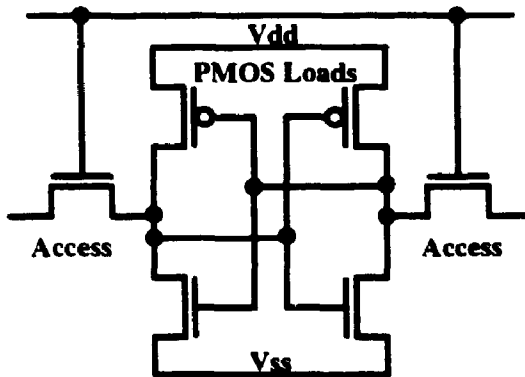


Fig.2.3(a) Full CMOS SRAM cell

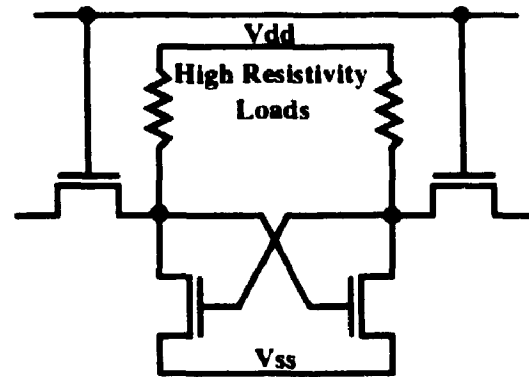


Fig.2.3(b) HRL SRAM cell

A sketchy description of an SRAM read cycle follows. The SRAM address is partitioned in two sections, serving as inputs to the row and column decoders. Most conventional SRAMs are asynchronous (i.e. there is no clock in the interface), and an Address Transition Detect (ATD) circuit initiates an access cycle once the address lines are stable for a certain duration. The row decoder selects and activates one Word Line (WL) that turns on all the access transistors in the selected row. The selected cells begin driving the associated Bit Lines (BL) and their complements, BL.B (Bit Line Bar), which are typically precharged HIGH outside active cycles. The cell sees the bit lines as large capacitive loads, to be discharged by the cell NMOS drivers to the sensitivity level

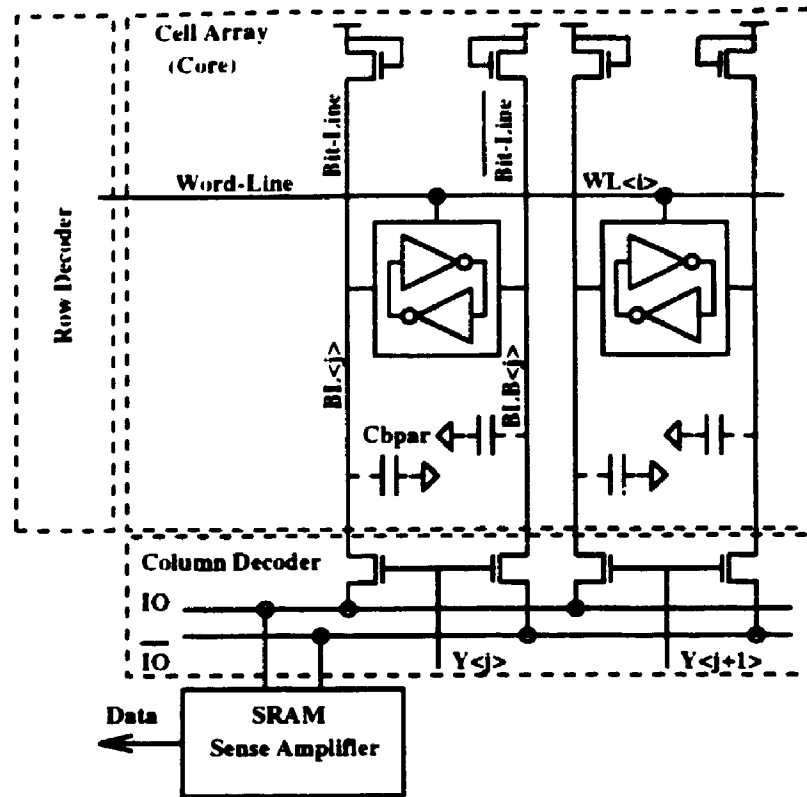


Figure 2.4 Generic SRAM structure

of the sense amplifier. The column decoder selects one (assuming one bit interface) BL/BLB pair and connects it to the differential I/O bus. This bus is the differential input to the SRAM amplifier which regenerates full logic levels and drives an output buffer. The signal carried by both the bit line pairs and by the I/O bus is a small differential voltage (tens or hundreds millivolt), with a DC level close to the precharge voltage.

A key point when building an SRAM-based C*RAM is that one sense amplifier is associated with a group of array columns. For C*RAM, the number of sense amplifiers dictates the maximum number of PEs, because the PE input needs the full logic level from the SA output. The main SRAM design trade-off is between sensing speed on one side and low-power and high density on the other side: more columns connected to the same sense amp is good for density and power (there are fewer SAs overall) but bad for sensing speed because of the increased capacitance of the I/O lines, leading to a smaller sensed signal. For example, in one 1 Mb SRAM [Wada87] there is one SA for every 16 columns, for a total of 128 SAs per chip, and the resulting access time is 37 ns. In another 1 Mb SRAM using similar technology, [Mats87], there are only 8 columns per SA

for a total of 256 SAs and a shorter 25 ns access time. This last SRAM has in fact a two-level hierarchy of sense amplifiers, four of the first-level SAa being connected to one second-level SA. In both 1 Mb SRAMs, there are a total of 512 rows and 2048 columns, organized in a number of blocks, hence there are 512 cells in one bit column.

2.3.2 Dynamic Random Access Memory (DRAM)

The basic 1T-DRAM cell, shown in Figure 2.5, has just one access transistor and one storage capacitor. The information is stored as electrical charge on the cell capacitor, and is periodically read and rewritten - a process called refresh - in order to compensate for charge leakage. There are no circuit variations but there are plenty of technological alternatives to implement the basic cell. Up to and including most 1 Mb DRAMs, the basic cell was planar [Prin91] and DRAM was considered the semiconductor process technology driver. In the quest to reduce cell size and manufacture larger capacity DRAMs, the DRAM foundries were the first to decrease the feature size (finest line in the lithographic process). Starting with the 4 Mb level, the planar cell was replaced with three-dimensional structures (stacked capacitor, trench capacitors), and the DRAM processes, while still very aggressive, became too specialized to be representative of other semiconductor categories. For example, most ASICs are connection-limited by the number of metal layers available, rather than logic gate-limited by the process lithographic resolution.

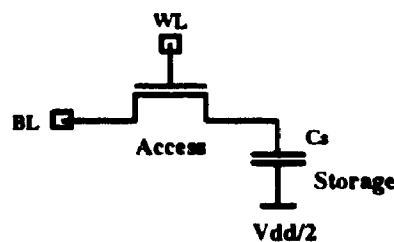


Figure 2.5 DRAM core cell

A DRAM block diagram is shown in Figure 2.6. Compared to SRAM, the main difference is the presence of one sense amplifier for every column, localized between the bit lines and the Input/Output (I/O) lines. In the large majority of DRAMs, the sense amplifier (SA) is a latch (similar to the SRAM cell but without access transistors) whose power supply connection is controlled by a transistor. One SA per column is necessary to do the refresh operation, i.e. to regenerate the full

logic levels and drive them back onto the bit lines in order to refresh the cell charge information. The SA layout is physically constrained now to the width of one memory column, which is in the order of microns for current generations DRAMs.

For a DRAM-C*RAM, it seems that the maximum number of PEs can be much bigger when compared to SRAM-C*RAM. In reality, even with the simple PE circuitry described in Chapter 3, the extremely small SA pitch requires that one PE correspond to a number of SAs (e.g. one PE for every 4, 8 or 16 SAs, depending on DRAM generation).

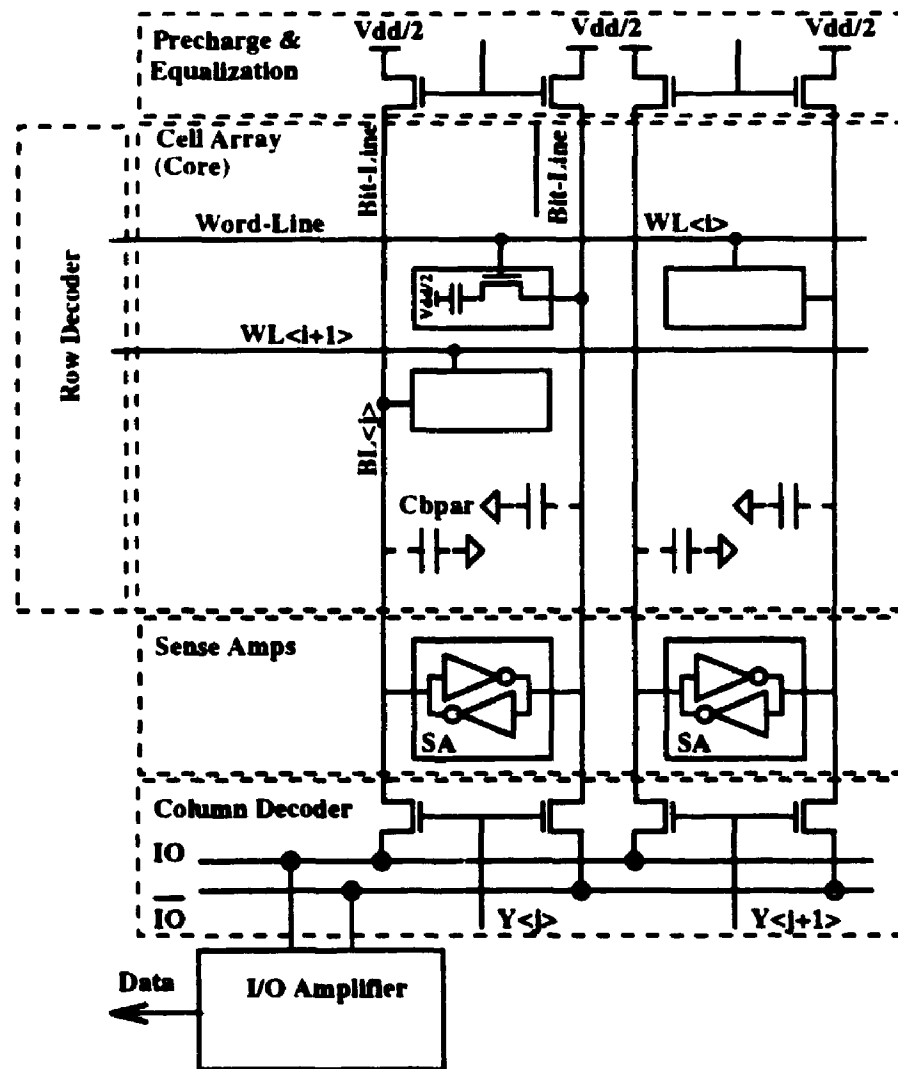


Figure 2.6 Generic DRAM (after [Taka93])

In DRAM design, the main parameter is the sensing voltage presented to the sense amplifier

input. This parameter depends on the ratio of cell storage capacitance (on the order of 50 fF for 4 Mb DRAM) to the bit line parasitic capacitance, because this is the reduction factor applied to the cell voltage when the cell is selected for read-out. The limited cell capacitance combined with the parasitic bit line capacitance limits the number of DRAM cells connected in a column to a typical value of 128. In the SRAM examples presented above, there were 512 cells in a column. The SRAM can afford more cells per column because the cell contains active elements, which, given enough time, will drive the desired voltage differential on the bit lines. As a result of the smaller number of cells per column, hence more total columns, the DRAM will have more sense amplifiers than an equivalent size SRAM with an hypothetical 1 SA/column.

2.4 C*RAM Economics

2.4.1 DRAM Economics

As mentioned in the first chapter, DRAM is considered the preferred platform for C*RAM implementation, because of its predominance in computer main memory subsystems and video frame buffers.

The problem with DRAM-based C*RAM is the access to DRAM fabrication technology, rather than designing and building the actual chip [IBMV94]. DRAM is a commodity IC category which require a very sizable capital investment (in the order of hundreds of millions US dollars). Once operational, the DRAM facility is geared towards high volume production to recover the capital investment, hence is quasi inaccessible for experimental prototyping. It has been noted that the time to recover capital costs rather than technological constraints dictates the DRAM generation advance interval. These facts explain why while there are a number of researchers trying DRAM+logic ideas [Lipo90], there are no actual proof-of-concept chips.

In economics terms, C*RAM represents the "technological push" aspect rather than the "market pull". There are no working applications to demonstrate the advantages of C*RAM over existing platforms. This is partly because there is no C*RAM platform to enable experimenting with applications. A solution to this problem is the development of a software emulator, as close to the final hardware as possible. This work is underway at Carleton University [Cast94], based on the

on the C*RAM chips presented in this work. A possible way to approach DRAM C*RAM is building a system based on high-density SRAM C*RAM. Such systems will address a higher-end market (because of the DRAM-SRAM price differential), but could be an intermediate stage in building up applications and market awareness of the technology.

2.4.2 C*RAM Area and Yield Issues

We assert that in order for C*RAM to become a viable substitute for at least part of computer memory, the supplementary area taken by the PE array has to be a small fraction of an equivalent capacity memory. The C*RAM chip cost includes the package and the silicon die costs. The C*RAM pinout is expected to have 3-4 extra pins over the corresponding DRAM (Chapter 5). This would have been a problem a few years ago, when DRAM packages were highly standardized. Today (1994), the DRAM market displays a much larger variety of interfaces (wide word, Synchronous DRAM, Extended Data Out, Rambus DRAM) and packages, and having four more pins is no longer an iconoclastic idea. Hence, the price of the C*RAM package is estimated to have a minor differential above the average DRAM package.

Chip area is one of the two factors affecting the manufacturing yield (ratio of good chips to total chips) and in turn the yield is the main factor determining the die cost.

The yield Y of memory chips can be estimated using the following formula [Prin91]:

$$(2.1) \text{ Poisson: } Y = \exp(-A \cdot D)$$

where:

A = chip area.

D = defect density of the manufacturing line.

The formula shows an exponential decrease of the yield with an increasing area, which supports the argument for minimal area increase of C*RAM.

In order to improve manufacturing yields, especially in the first years of production, DRAM

designs include a number of redundant rows and columns. Upon detection of defective rows or columns, a spare element is activated typically by blowing on-chip fuses. The spare columns or rows are physically situated at the periphery of the array. The procedure is effective in increasing the final yield while keeping the same access time because there is no interdependence between rows or columns.

In contrast, most SIMD applications require some form of intercommunication between the PEs. The Baseline PE, presented in Chapter 3, has closest neighbor shift-left/right links. The Bit-parallel Extended PE array presented in Chapter 5 depends in large measure on local communication. Since PEs are directly associated with memory columns, the PE order imposed by local communication is transferred to the memory columns. The conclusion is that memory column redundancy methods cannot be practically applied to the PE array, if defective PEs are detected. Replacing a PE located in the middle of the array with a spare PE located at one extremity would have a destructive impact on operating speed, a 40 μm long line driven by close to minimum size transistors being replaced with a 5 mm one (numbers are estimated values).

The practical alternative to maintain acceptable C*RAM yields against PE defects is hence keeping the PE array area to a minimum. Memory row redundancy methods are still applicable to correct row errors in the memory array. C*RAMs with defects localized within the PE array could be converted to functional DRAMs using both row and column redundancy, assuming the testing and packaging procedures allow it. A more in-depth study of C*RAM redundancy issues is necessary in preparation for a high-density DRAM-based C*RAM design.

As a final note, it should be mentioned here that even if the die and package costs are maintained close to those of DRAM, the price of C*RAM will be dictated by the size of the market. A price decrease curve is expected as more C*RAM applications are written and the chip volume ramps up. A rather non-linear market development scenario would involve a C*RAM association with a mass-market application, the best example being High Definition Television (HDTV). It is estimated that an HDTV receiver will need 64 Mbit of 50-ns DRAM [Chri94]. C*RAM could perform the decompression, echo-cancellation and image resizing functions, relieving the memory subsystem from the rather expensive 50-ns specification. It is becoming more and more obvious in the digital video industry that the processor-cache-DRAM architecture is not well suited for

continuous data stream processing. This realization seems to be behind the push for fast DRAM interfaces, while at the most extreme end of the opinion spectrum has already appeared the idea of moving pixel processing into DRAM.

Chapter 3

Processing Element - Circuit and Physical Design

3.0 Introduction

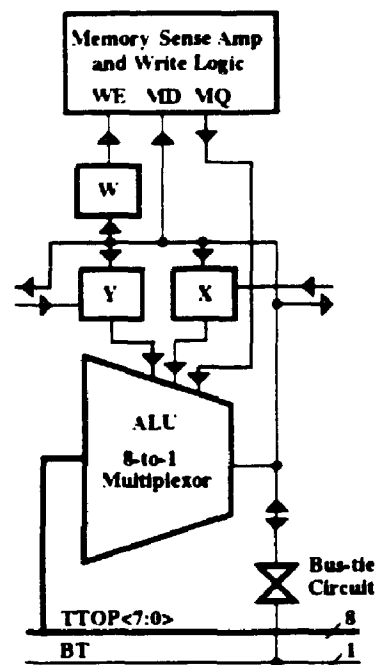
This chapter starts by describing in detail the Processing Element (PE) architecture as defined by Duncan Elliott in [Elli92]. This structure is referred to in the following as the Baseline PE (BPE), to establish a reference for subsequent discussions on architecture modifications in Chapter 5. The main concern here will be for an area efficient PE implementation in BNR/NT's 0.8 μm BiCMOS technology (BATMOS), observing the layout constraint imposed by the memory sense amplifier width (Chapter 4). An important design objective is safe-guarding the design against potential technology risks, such as the poorly modeled leakage currents. Speed and power performance take second place to functional reliability. It will be shown that a very dense PE layout imposes interaction between the circuit design and physical design (layout) levels, involving trade-offs between circuit area and performance. The three metallization layers available in BATMOS are demonstrated to be very important in achieving an extremely dense PE layout.

3.1 Architecture of the Baseline Processing Element

Figure 3.1 presents the Baseline PE. The main components are three 1-bit data registers, one arithmetic and logic unit (ALU) and a bus-tie circuit. The ALU is in fact an 8-to-1 multiplexer. The three selection inputs are sourced by two data registers together with a third line from the local memory (termed M or MQ). In this context, the term "local" refers to circuitry and/or data

that is particular to one PE, while the term "global" refers to circuitry/data/operations common to all PEs. The 8-bit input to the multiplexor is common to all PEs and carries the global instruction.

Figure 3.1
Baseline PE (BPE)
Architecture



The multiplexor output value, termed "ALU Result" or "ALU", can be written into any local register or into the local memory, when the corresponding control signals are activated. In Figure 3.1, the ALU output is represented as bidirectional, driving the registers inputs and being driven by the Bus-tie circuit. The actual circuit implementation will be presented later in the chapter.

This structure can execute any boolean function of three variables, by broadcasting the corresponding truth table on the 8-bit multiplexor input bus. There are thus 256 executable functions. In particular, two functions of interest are the ones that implement a full adder, presented in Table 3.1 with the corresponding 8-bit Truth Table OPERATION code (TTOP):

Table 3.1 Examples of logic functions of 3 variables

Logic Function	Expression	TTOP<7:0>
XOR3(Y, X, M)	$Y \text{ xor } X \text{ xor } M$	1001_0110
CARRY(Y, X, M)	$Y * X + Y * M + X * M$	1110_1000
X_IDENTITY(Y, X, M)	X	1100_1100

where xor, *, + denote respectively the exclusive-or, logic-and, logic-or functions and the overline denotes the logic complement. A full-bit add requires hence two ALU operations. As an example, Figure 3.2 presents two PEs performing a XOR3 evaluation, in the case $Y=1, X=0, M=1$ (Y is the most significant bit of the multiplexor selection bus, M the LSB) and $Y=0, X=1$ and $M=0$.

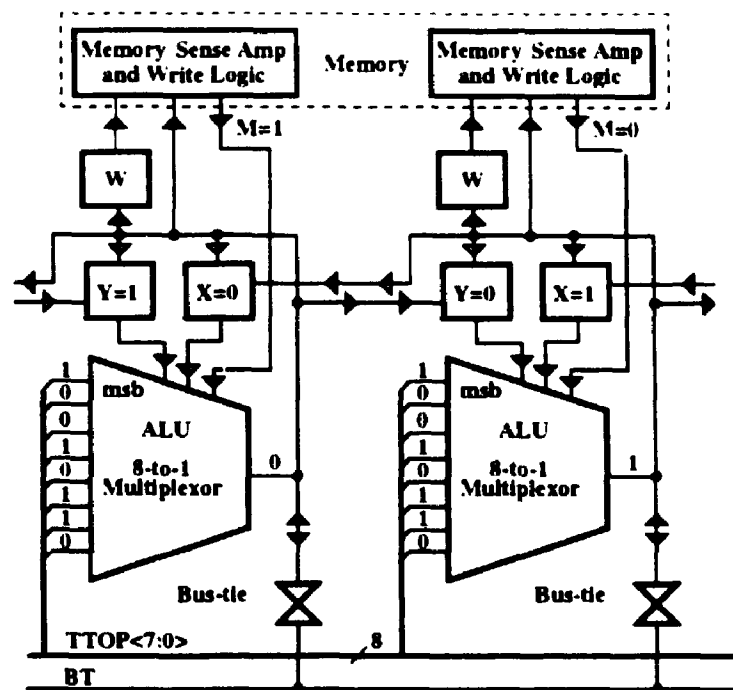


Figure 3.2 Two ALUs executing the XOR3 function

An ALU operation is followed by writing the result into one of the local registers, neighboring register (in the case of shift) or local memory.

3.1.1 Communication between PEs

As discussed in Chapter 2, this is an important topic in the field of SIMD machines, since the performance of many SIMD algorithms depends on the inter-processor communication scheme. The more complex the network, i.e. the bigger the number of links between the processors, the bigger the price paid in silicon area and packaging size. Since C*RAM has severe constraints of compatibility with conventional memory, the communication scheme adopted has to be the least expensive in terms of silicon area and package pins. This implies a unidimensional network, taking a minimum of routing channels and requiring a minimum of multiplexing logic within the PE.

The baseline PE has two levels of inter-processor communication (IPC):

1. Closest neighbor shift-left and shift-right communication. (Figure 3.3)]
2. Global wired-function. The actual logic function is either wired-OR or wired-AND, depending on the circuit (Figure 3.4).

The shift left/right connections use hard-wired register destinations. The baseline PE uses the left neighbor X register as a shift-left destination and the right neighbor Y register for shift-right. In both cases, the shifted bit is the result of the last ALU operation. Details on the reasons for this particular implementation of shift operations are presented in section 3.3.1.

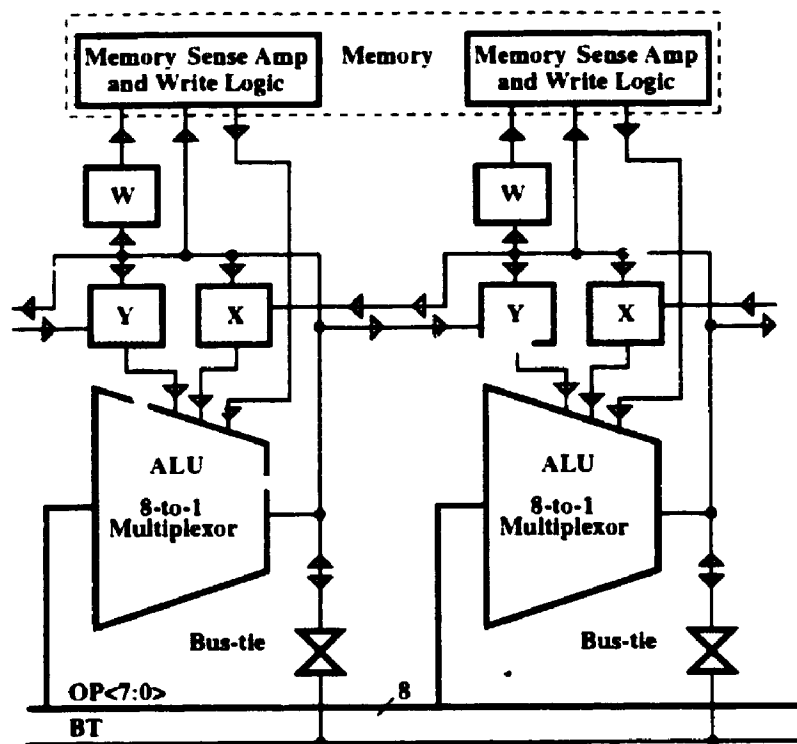


Figure 3.3 Shift-right data flow

The wired-function network, also termed bus-tie, is a one-to-many communication circuit which allows one PE's result line to affect all other result bits. The BATMOS C*RAMs implement the wired-OR function across groups of 64 PEs. Section 3.4 will show how the wired-OR function is the result of PE circuit level design. A wired-AND can be obtained in software by complementing the operands and the result, according to de Morgan's logic theorem.

Figure 3.4
Bus-tie data
flow

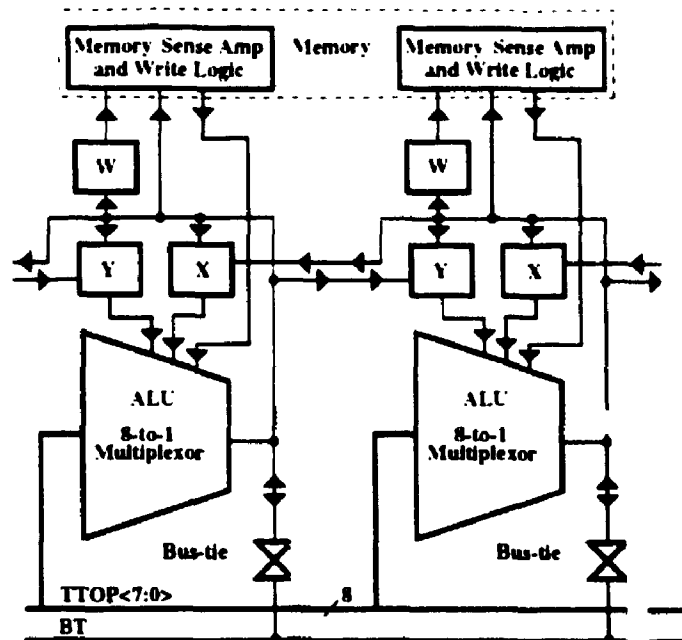


Figure 3.4 shows symbolically the data flow during a bus-tie operation. Under the control of the Bus-tie Enable line (not represented), all bidirectional bus-tie circuits (bowtie shaped in the figures) are turned on. After a predetermined time necessary for worst-case end-to-end propagation, the operation may be followed by a register/memory write. This is not necessary though, because beside setting the ALU result lines to the same value, the bus-tie operation makes this value available on a C*RAM output pin. This output is typically used by a C*RAM bank controller to branch program execution.

3.1.2 Conditional Execution Support

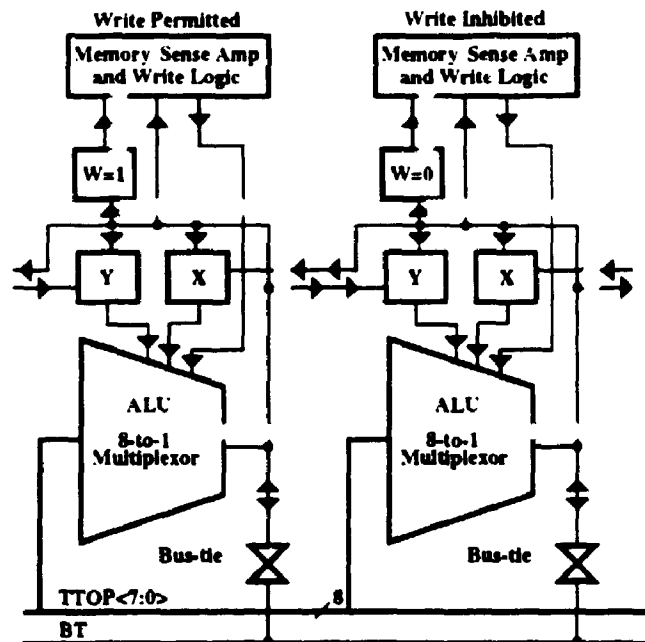
Typically, SIMD machines provide the means for some processors to “sit out” a sequence of instructions, based on a previously computed local condition. This corresponds to a parallel “if-then-else” instruction, where the “then” sequence is executed by the processors having the “true” condition bit set, while the “else” sequence is executed by the rest of PEs.

The circuitry that implements this functionality in the baseline PE is shown in Figure 3.5. A third local register called W (for Write) holds the condition bit, written again by the ALU Result line. For the baseline PE array, the “sitting out” does not happen, instead all PEs execute all instruc-

tions. The condition bit stored in *W* only affects the write-back to memory. This assumes that the memory interface offers a local Write Enable input, which is controlled by the *W* register. Since register *W* only affects the memory write-back, all local registers *X* and *Y* will be affected by an instruction writing *X* or *Y*.

Figure 3.5

Conditional write-back to memory using the *W* register



3.1.3 Baseline PE Control Lines

The previous figures have shown various data flows in the PE array, but have not explicitly presented the control signals. To summarize the previous presentation, the following is a list of all control signals needed in the Baseline PE:

a. ALU: There are no explicit ALU control lines to indicate operations to be performed (e.g. ADD, SUB, NEG, etc.). Instead, the operation to be executed is implicit in the 8-bit truth table (TTOP). The operands are always the two bits stored in registers *X* and *Y* and the bit last read from memory, denoted *M*.

b. Registers.

1. WX: Write X Register
2. WY: Write Y Register
3. WW: Write W Register

- 4 SLX: Shift Left into X (write left X register)
5. SRY: Shift Right into Y (write right Y register)

c. Bus-tie circuit

6. BTEN: Bus-tie Enable

These last six lines constitute the Control Opcode (COP). Together with the 8-bit TTOP they form the global C*RAM Operation Code. Section 4.4 will present the method of providing the C*RAM Opcode and the associated support circuitry. Beside the C*RAM Opcode, there are a number of timing signals used to sequence the control events. The C*RAM timing constraints are discussed in sections 3.3 and 3.4 and the implementation of a timing circuit is presented in Chapter 4.

3.2 Processing Element CMOS Circuit Design - Strategy

The main objective of PE circuit design is to find the minimal area circuitry that has the functionality described above. C*RAM computing power is directly related to the number of PEs integrated in the memory chip. This number is estimated in the 512-4096 range for C*RAMs based on industrial density memories. At the same time, the supplementary area taken by the PE array has to be maintained for pricing reasons to an estimated maximum of 15-20% of the memory area. This constraint is justified by the objective of eventually replacing part of standard memory with C*RAM. Hence, to maintain comparable pricing, the area and manufacturing yield have to be similar. It will be shown in the following that the circuit level and physical level (layout) design interact with each other, and circuit design decision can be affected by layout compaction criteria.

There are two major alternatives for implementing the PE circuits in CMOS technology: static logic and dynamic logic, the latter with a number of variations [West93]. Dynamic logic requires less silicon area than its static counterpart (except for very simple logic functions), hence will be the preferred implementation method. The price is paid in a more complicated timing scheme, requiring at least one clock signal. A timing circuit though is a one-time area cost since it will support the entire PE array (or a number of arrays). The area taken by the timing block is more than offset by the area saved in a more compact PE, multiplied by the number of PEs.

Although BATMOS offers vertical NPN bipolar transistors, the PE does not benefit from their use because of its highly local interconnections. In digital circuits, bipolar transistors (when available) are used to drive large capacitive loads (long lines with high fan-out), whereas in the PE the longest track is in the order of 50 μm . The only exception is the global bus-tie circuit, but the option of using bipolar transistors in the PE circuitry is excluded for three reasons:

1. Size - the minimum size bipolar transistor is almost four times bigger than the minimum size MOS transistor;
2. Latch-up hazard - bipolar transistors in saturation inject substrate currents, which can trigger latch-up (low resistance path between supplies through an SCR-like device) in a CMOS structure.
3. Design portability - typical memory processes do not have bipolar transistors, hence a BiCMOS PE will be hard to adapt to other processes.

3.3 8-to-1 Multiplexor ALU

3.3.0 Circuit Alternatives

A static gate-level implementation of the *mux8* function is shown in Figure 3.6. As indicated in the figure, this circuit would require 84 MOS transistors (MOSTs), 42 PMOS and 42 NMOS transistors. In the figure, the number of transistors in each gate is shown by n , nt being the total. To facilitate size comparisons with alternative circuits, we can assume for now that all 84 transistors have minimum gate length $lp=lmin$ and $ln=lmin$, all PMOSTs have a width of $wp=wpmux$ and all NMOSTs have $wn=wnmux$. To compensate for the almost three times lower mobility of PMOST carriers compared to NMOS carriers, we estimate $wpmux=2.5*wnmux$. The static implementation chosen for reference has three levels of logic, where instead of an 8-input NAND gate at the output there are two 4-input NAND gates followed by a 2-input OR gate. The three logic level circuit is the practical implementation despite the extra four transistor in the OR gate. This is because the chain of eight series NMOS transistors in the 8-input NAND gate is slower than a cascaded 4-input NAND with a 2-input OR. This is confirmed by practical rules-of-thumb in ASIC design with BATMOS which exclude more than five NMOS transistors in series in a logic gate.

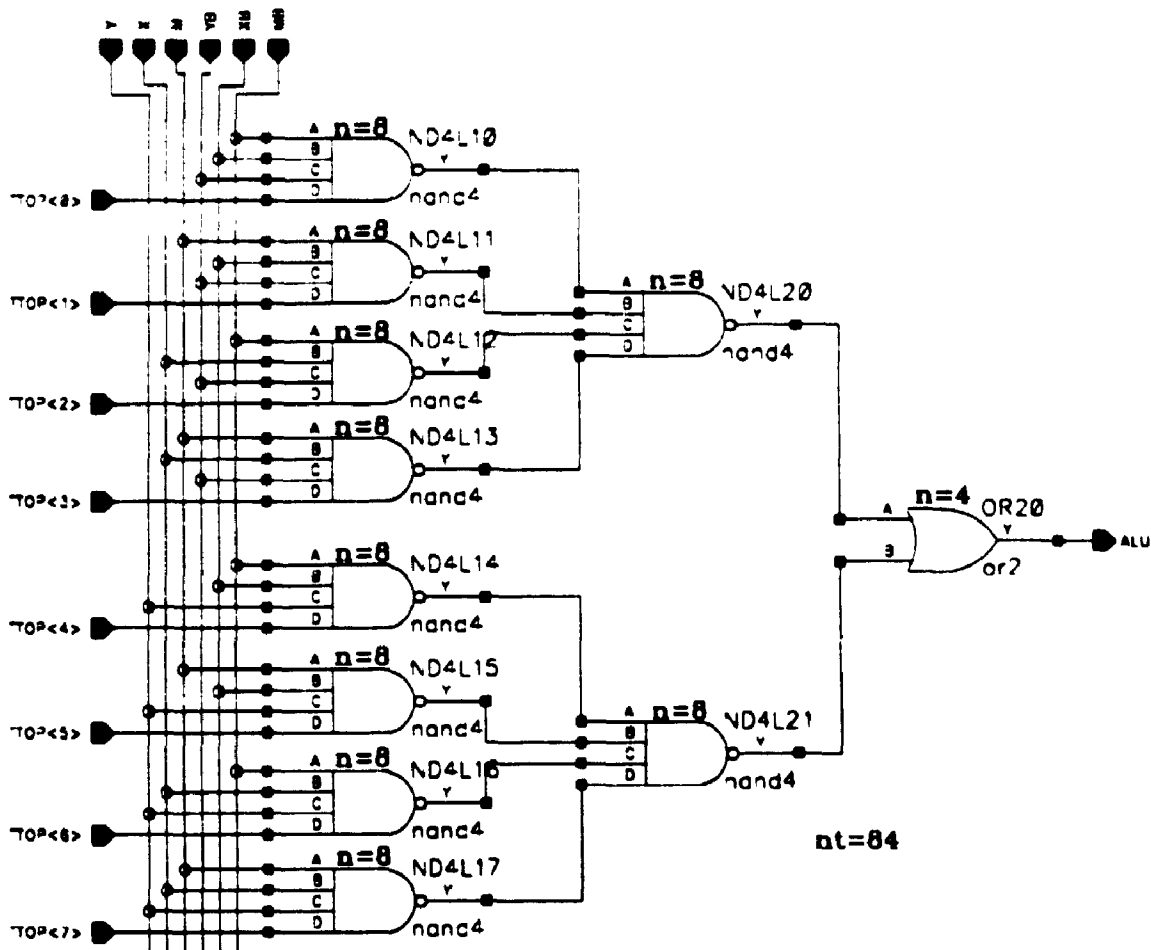


Figure 3.6 MUX8 static CMOS implementation

The same *mux8* function implemented with CMOS dynamic logic (a potential domino logic section) appears in Figure 3.7. The circuit, named DYN5N to indicate the five series NMOST, requires only 34 transistors. There is only one PMOST, the precharge transistor PPRCH. For size estimation, the precharge PMOS will have $lp=lmin$ and $wp=wpmux$. The factor of two reduction in transistor count translates into a factor of almost three in area savings because of:

1. the $wpmux=2.5*wnmux$ sizing done in order to equalize transition (rise/fall) time, and
2. the elimination of n-wells (required by PMOS device diffusion), which require large spacings from both n- and p-device wells.

For NMOSTs, 32 transistors have the same $ln=lmin$ and $wn=wnmux$ as before, while the bottom NEVAL has an estimated $wn=4*wnmux$. By comparing the total transistor area for the static and dynamic alternatives, the dynamic implementation becomes by far the preferred choice.

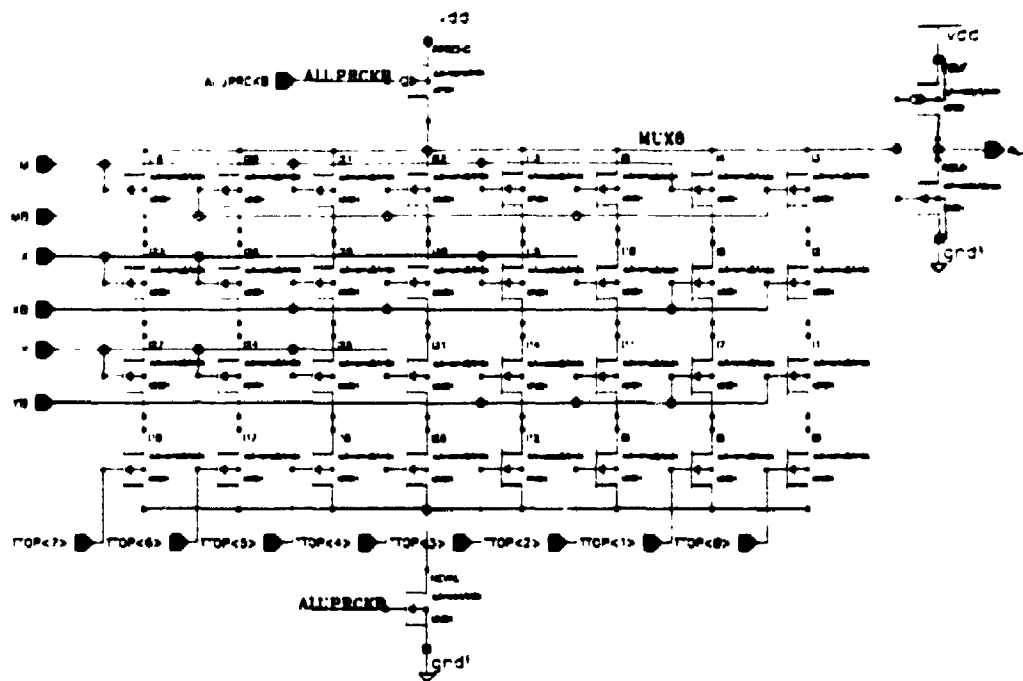


Figure 3.7 DYN5N dynamic logic implementation of the 8-to-1 multiplexor

The dynamic logic DYN5N circuit requires one clock signal, ALUPRCKB, in order to control the precharge/evaluation phases (the convention used in this document appends the letter "B" at the end of a signal name to denote the inverted signal). The functioning is as follows: during ALUPRCKB = LOW (precharge), the gate and parasitic capacitance on node MUX8 are charged to the positive supply v_{dd} . During ALUPRCKB = HIGH (evaluation), the precharge PMOS is turned OFF and the NMOS evaluation transistor NEVAL is turned ON. The precharged node stays HIGH or is discharged LOW according to the logic values of selection inputs Y, X, M, their complements and the eight truth table signals TTOP<7..0>. The dynamic node is isolated from the rest of circuitry by the inverting buffer PBUF/NBUF. The gate capacitance of this inverter contributes to the node capacitance of the precharged node together with the drain diffusion capacitance of the top eight NMOS transistors in the multiplexor. The logic value evaluated at node MUX8 has to be stored in a register/latch before it is changed by charge leakage through reverse biased junctions and MOS subthreshold currents. Both leakage currents are difficult to model for numeric simulation, hence the time interval available before charge leaking affects the node value is an unknown and potentially risky design issue.

Another problem with this dynamic multiplexor structure is that the NMOST evaluation chain

between the dynamic node MUX8 and the v_{ss} node has five series transistors. In this chain, only the evaluation transistor NEVAL has the source at the substrate potential v_{ss} . The other four NMOS transistors are affected by the back-bias effect and their effective threshold voltage V_{th} is increased. This results in a longer evaluation time $t_{HL,MUX8}$, hence it slows the overall ALU cycle. A Spice simulation of the circuit with transistor sizes identical to those in the final layout is shown in Figure 3.8. The ALU propagation delay $t_{prop,ALU}$ is measured from the midpoint of TTOP rising edge to the midpoint of resulting ALU edge.

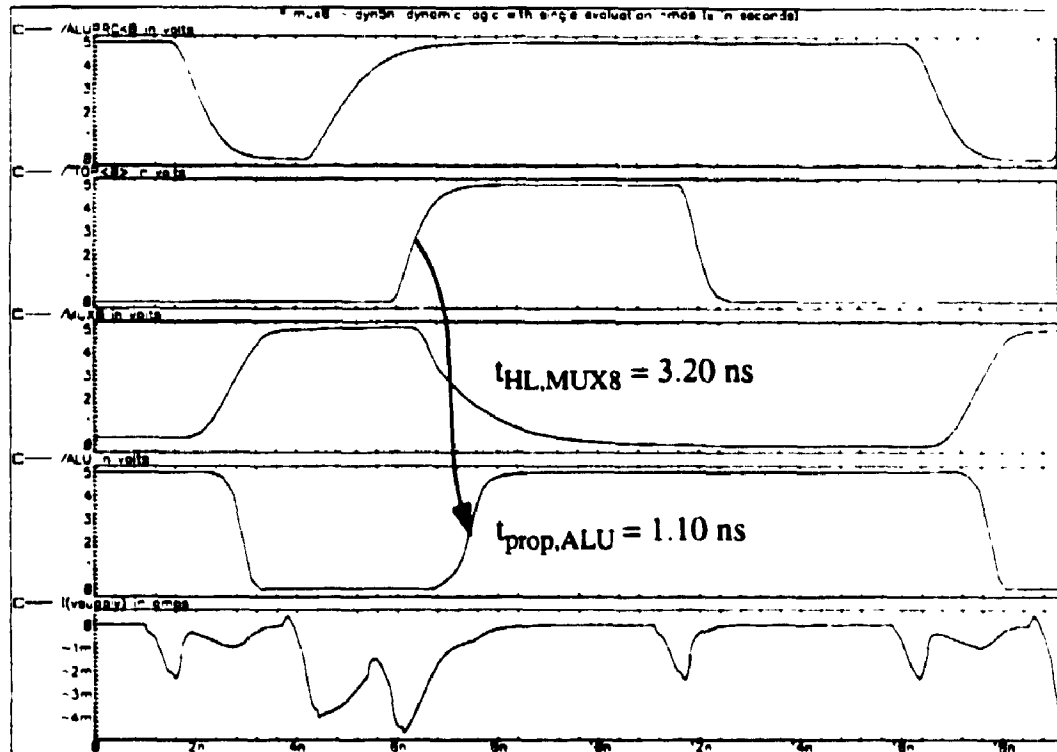


Figure 3.8 Spice simulation of DYN5N (typical BATMOS, 70°C)

A circuit alternative is shown in Figure 3.9, referred to as DYN4N. Here the evaluation transistor is removed hence the NMOST chain is reduced to four transistors. In order to keep the evaluation phase non-overlapped with the precharge phase, the timing information previously carried by the ALUPRCKB signal is included now in each TTOP<k> signal. These signals are now labeled TTOPCK<k>, to stress that they are part of PE timing. Compared to DYN5N, there is also a small advantage in area, from removing the larger NEVAL NMOST, estimated at $wn=4 \cdot wn_{mux}$. Looking beyond the strict PE circuitry, the disadvantage of DYN4N is that the support circuitry needs now to generate eight timing signals TTOPCK<7..0>, instead of the previous one, ALUPRCKB.

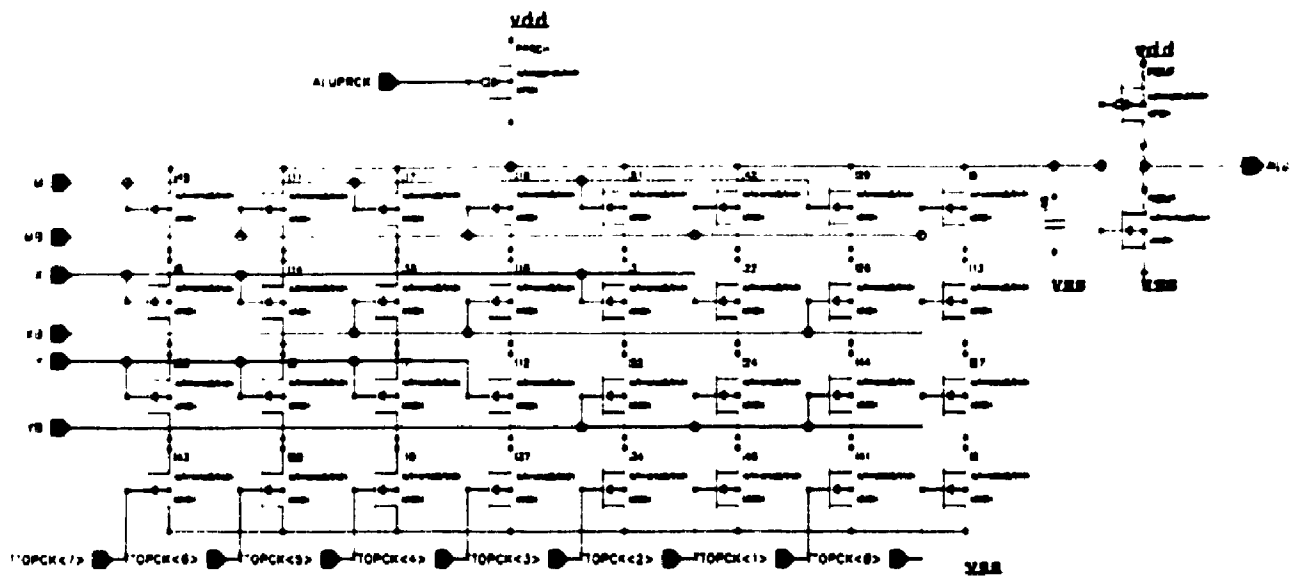


Figure 3.9 DYN4N dynamic implementation of MUX8

As mentioned above, the dynamic logic multiplexor function relies on the charge stored in the precharged node capacitance. Assuming that during the evaluation phase ($ALUPRCKB=HIGH$) there is no conductive path to vss , the dynamic node MUX8 is initially kept HIGH by the charge stored in the node. Eventually, the node is discharged LOW by two types of leakage current: 1. the reverse biased pn junction current flowing through the substrate, and 2. the subthreshold current of the NMOS transistors. As mentioned before, there is a degree of uncertainty regarding how long the charge stored in the node capacitance will maintain the node HIGH. On the other side, there was no experimental data available to estimate this time. Discussions with designers from the Memory Group in Bell-Northern Research have confirmed the unreliability of MOS charge leakage simulations.

In view of the risk factor caused by a pure dynamic logic implementation, a latch-back PMOS transistor is added to the multiplexor, transistor PLATCH shown in Figure 3.10. Its gate is connected to the output signal ALU driven by the inverting buffer. After precharge, node MUX8 is HIGH and ALU is LOW, hence PLATCH is turned on. Assuming the evaluation phase results in MUX8 staying HIGH, transistor PLATCH provides now a conductive path to vdd to compensate the charge leaked via substrate and subthreshold currents. Hence, MUX8's HIGH logic state is latched by the inverter PBUF/NBUF connected back-to-back with the "half-inverter" PLATCH.

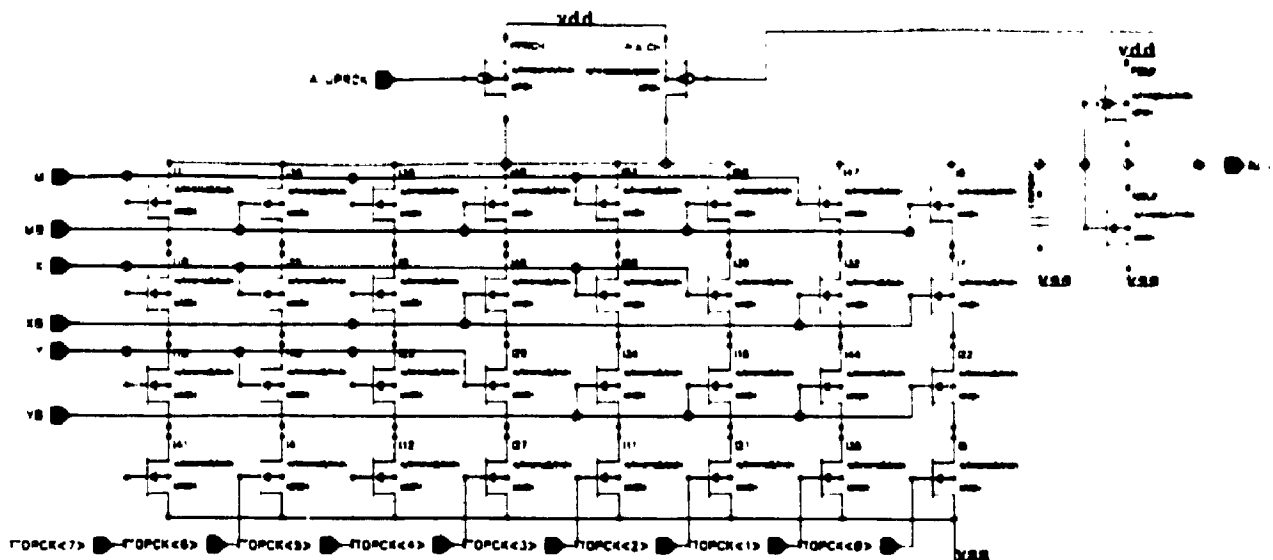


Figure 3.10 Pseudo-dynamic DYN4N with latch-back PMOST (*PLATCH*)

The disadvantage of the PMOS latch-back appears when MUX8 evaluates LOW. If the evaluation phase turns on an NMOS path to v_{ss} , the four series NMOS transistors have to force MUX8 to a LOW state despite PLATCH being turned on. Hence, there is extra short-circuit current flowing between supplies. Also, evaluating MUX8 LOW will take longer than in the pure dynamic logic circuit. Since the leakage currents are small, PLATCH can be “weak”, i.e. can have a small w/l ratio, minimizing the negative impact on switching power and delay when MUX8 goes LOW. We choose $w = w_{pmin}$, the minimum device diffusion width in BATMOS, and $l = l_{platch}$, as large as permitted by layout, since increasing the length means extra area taken by PLATCH.

A final modification to the ALU circuit is brought by adding a weak NMOS latch-back NLATCH to form a complete latch (back-to-back inverters) on the ALU output. The goal is to avoid leaving the node MUX8 to float during the write-register phase, after it was discharged LOW in the evaluation phase. To avoid a logic hazard in the ALU-registers loop, discussed in detail in section 3.4, both ALUPRCKB and the TTOPCK signals are inactivated during the write-register phase.

A Spice simulation of an ALU multiplexor cycle is shown in Figure 3.11. Compared to the previous simulation (Figure 3.8), we notice the longer ALU propagation time, $t_{prop, ALU}$, despite having only four NMOST in the evaluation chain. The extra delay is caused by the presence of the weak latch-back inverter, which opposes both precharge and evaluation.

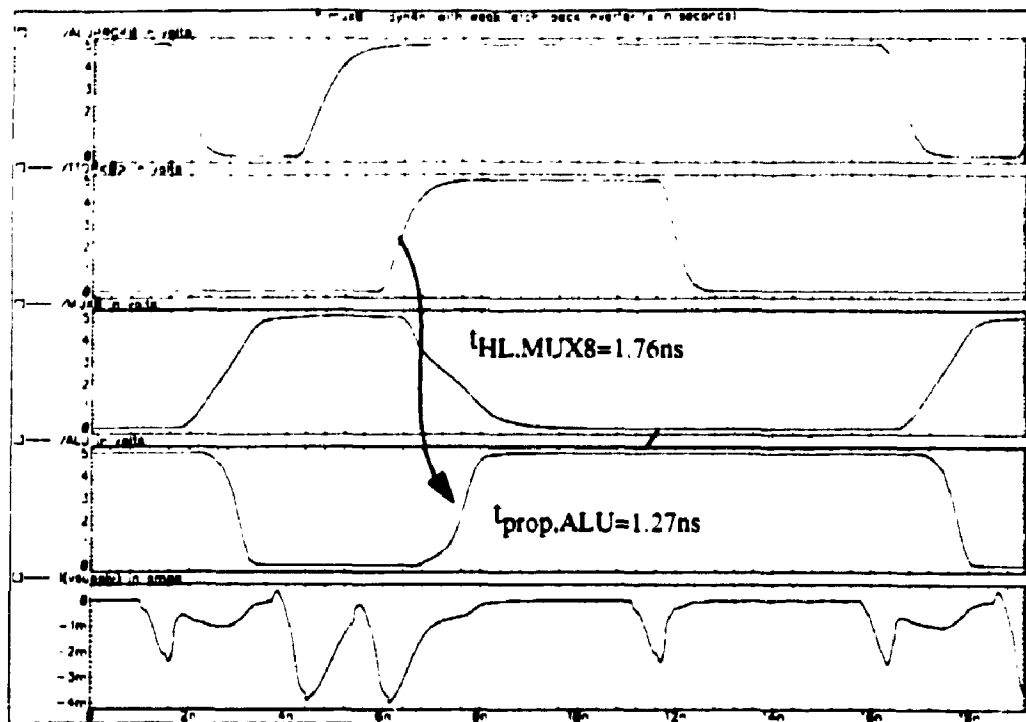


Figure 3.11 ALU operation cycle

3.3.1 Logic and Physical Order of the Selection Lines

Two more elements have to be decided in relation with the ALU multiplexor:

- a. the binary weight of the selection inputs X, Y and M.
- b. the physical ordering of the NMOS transistors associated with the three selection lines in the NMOS evaluation chain.

We notice that the two are independent, i.e. assuming for example Y is the most significant logic selection line, it can be chosen to control any of the three NMOS levels in the four transistors chain (the bottom fourth is reserved for TTOPCK<x>.). The first decision is totally arbitrary, and we choose the alphabetical ordering to correspond with the least significant to most significant sequence: M is the least significant selection line and Y is the most significant.

The physical ordering of the selection lines can be decided analyzing how the signals X, Y and M can change during an ALU cycle. The objective is to minimize charge sharing within the 32 NMOST evaluation group. Charge sharing happens in dynamic logic structures during the evalu-

ation phase, when the charge initially stored in the precharged node is redistributed on the internal nodes of the evaluation group as a result of transistors being turned ON by changing inputs. Charge sharing would be a critical effect to consider if the multiplexor were a pure dynamic logic structure. This is no longer the case, since the PMOST latch-back was introduced to compensate for leakage currents. A second positive effect of this latch-back transistor is that it will also compensate any charge variation as a result of charge redistribution. Nevertheless, even with the latch-back transistor in place, we are still interested in reducing the noise at the ALU output by minimizing charge-sharing. Also, in view of a future pure dynamic implementation of the ALU, we maintain the same design criteria in order to ensure a portable layout. The actual order of the selection signals is chosen in section 3.4, after the difference between X and Y on one side and M on the other side is explained.

3.3.2 ALU Transistor Sizing

This section presents the decisions made in sizing the ALU transistors. The usual speed-power trade-off takes a second place to constraints imposed by layout area/pitch. The trade-offs would have to be reevaluated for a different C*RAM design, if for example low-power is favored over speed.

The main parameter influencing area, speed and power consumption is the width of the NMOS transistors in the ALU evaluation block, w_{nm8} . The practical minimum value for transistor width in BATMOS is 1.8 μm . Because of the final layout arrangement of the ALU multiplexor within the PE the maximum value for w_{nm8} is 4 μm . A number of Spice simulations were run for this fairly limited range of w_{nm8} values. Figure 3.12(a) shows the ALU propagation time, as defined by the interval between the rising edge of TTOPCK<k> and the resulting transition on the ALU signal (shown by the arrow in Figure 3.8). As expected, despite the bigger diffusion capacitance of the precharged node caused by the larger transistor width, the ALU propagation time decreases to a minimum of 1.27 ns when $w_{nm8}=4 \mu\text{m}$.

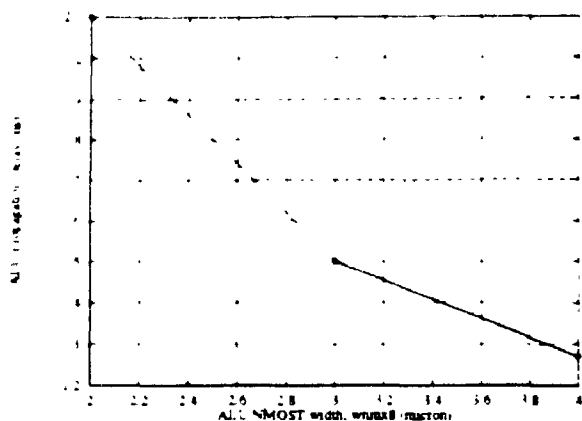


Fig.3.12(a) ALU propagation delay function of NMOST width w_{nm8}

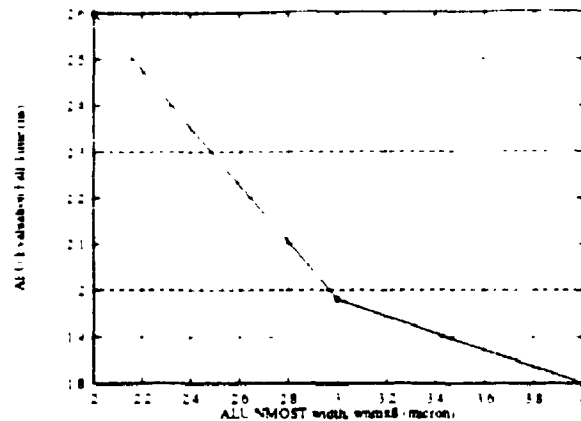


Fig.3.12(b) ALU discharge evaluation time function of w_{nm8}

Figure 3.12(b) shows the 90%-10% fall time of the precharged node (MUX8) voltage, as function of w_{nm8} .

Figure 3.13 shows the energy consumption during ALU operation for the same range of transistor widths. Separately shown are the energies consumed during ALU precharge and ALU evaluation (actual discharge, i.e. MUX8 goes LOW). It can be seen that the precharge energy is rather high when compared to the evaluation one. The explanation is the cross-bar current flowing through the ALU precharge PMOST and the NMOST latch-back. The latter has the practical minimum width ($1.8 \mu\text{m}$) and the maximum length permitted by the layout. Since the latch-back NMOST was the last addition to the PE layout, the maximum length available was $1.4 \mu\text{m}$. The sum of the precharge and evaluation energies is also shown in Figure 3.13, where a minimum is observed in the range of interest. In order to compute the average power, the energy is divided by the average operation period, which depends on the system clock frequency and the actual program being executed. Note that the total energy graph in Figure 3.13 corresponds to the worst-case instruction cycle, with a precharge and a discharge in one cycle. In reality, an ALU might not consume precharge energy because it was not discharged in the previous cycle, or might not be discharged because the current operation results in the precharged node staying HIGH (i.e. $\text{ALU}=\text{logic-0}$).

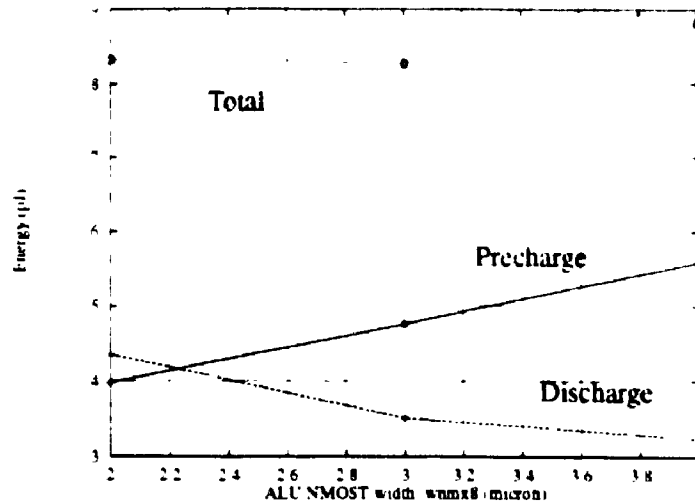


Fig.3.13 ALU precharge, discharge and total energy as function of $wnmx8$

The global consumption of the PE array depends again on the program executed and the data operated on.

We choose $wnmx8=4\mu\text{m}$ in order to obtain the maximum speed within the available area, at the expense of more than minimum power. In a well characterized technology of a commercial C*RAM, the latch-back inverter might be eliminated, resulting in higher speed and lower power within a smaller area.

3.3.3 ALU Layout

Figure 3.14 shows the block floor-plan of the ALU, with the approximative relative block sizes. The width and length in microns of each transistor are also shown. Most of the area is taken by the 32 NMOSTs evaluation group. The physical layout is reproduced in Figure 3.15.

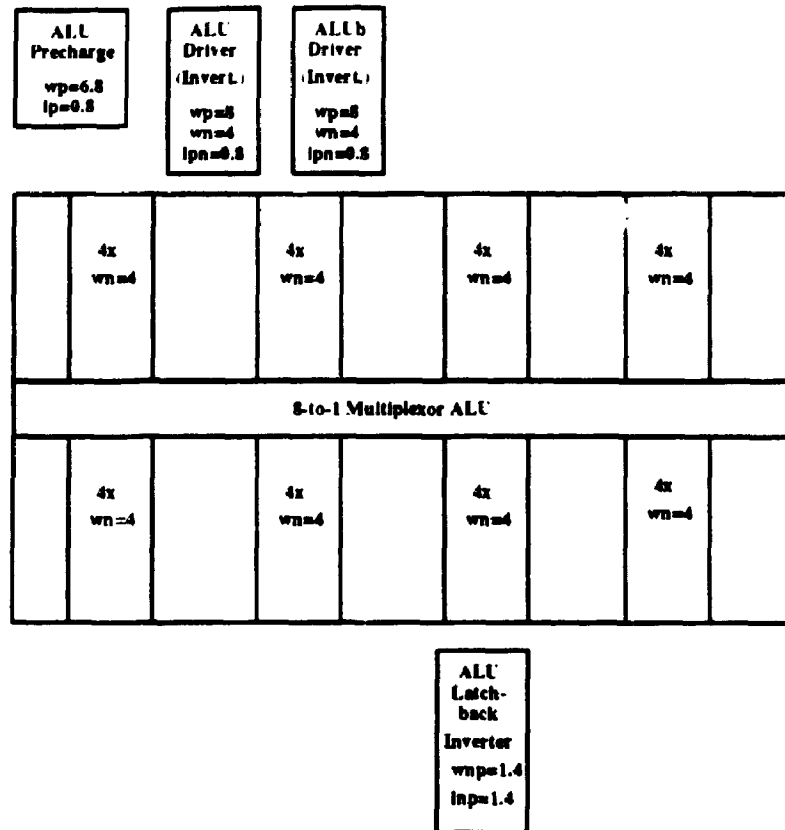


Figure 3.14 ALU floor-plan

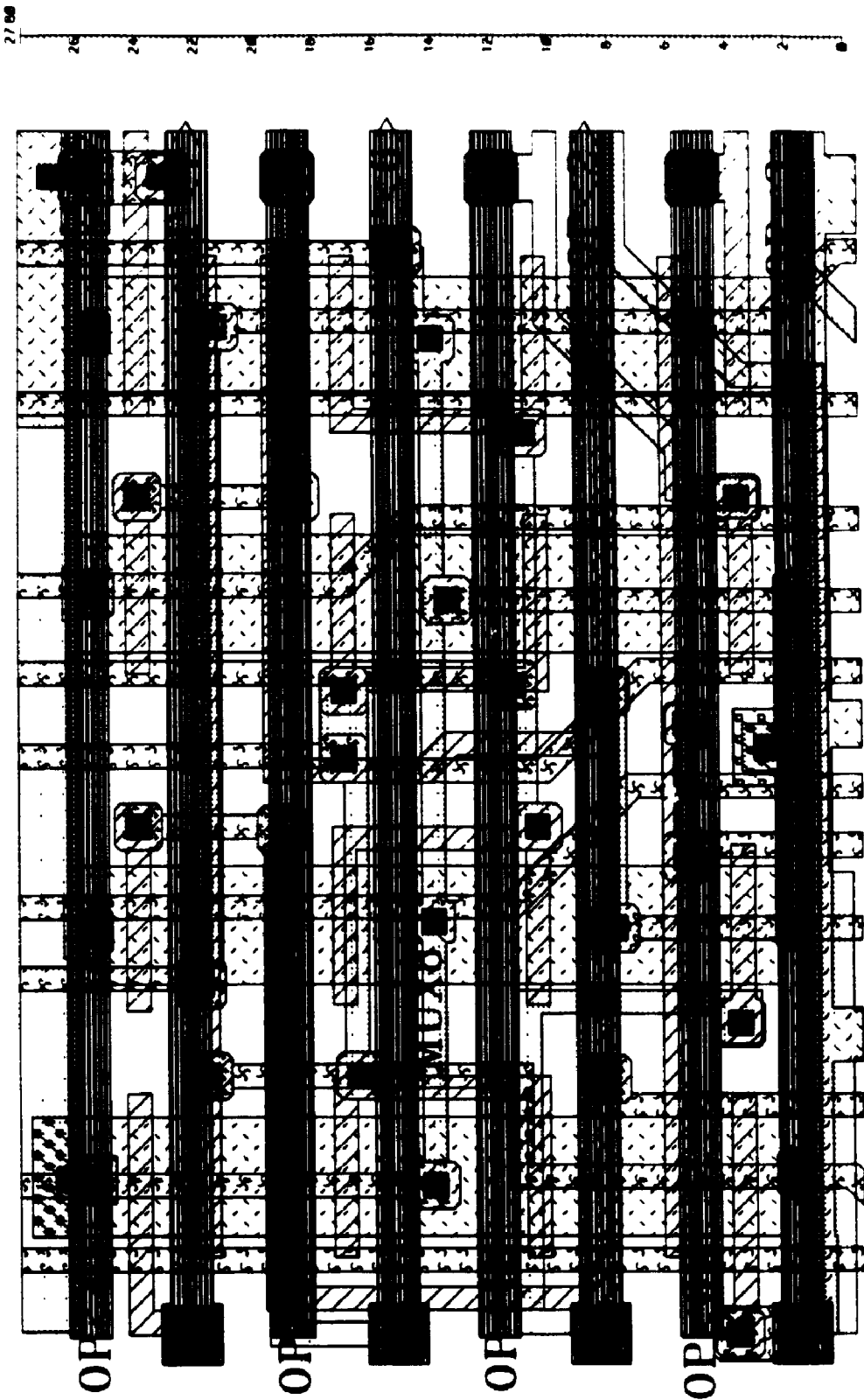


Figure 3.15 ALU multiplexor layout

3.4 ALU-Register Loop

As is the case with all ALU+registers logic structures, there is a potential logic feed-back problem in the C*RAM PE. The ALU result is the data input to the registers and at the same time, the outputs from registers X and Y are inputs to ALU. The usual way to prevent a logic instability loop is to use edge-triggered flip-flops (ETFF) for the register bank and for the ALU output. Such a flip-flop is in fact a combination of two cascaded level-controlled latches, clocked in antiphase, as shown in Figure 3.16. This structure isolates the flip-flop output from its input at any time except during the transition time of one of the clock edges (rising or falling). As a lexical note, the word register typically implies a set (one or more) of edge-triggered flip-flops (most often a D-type flip-flop). In the following, the PE storage elements X, Y and W will be also referred to as "registers", especially in a logic/architecture context, even if their implementation is different from an edge-triggered flip-flop.

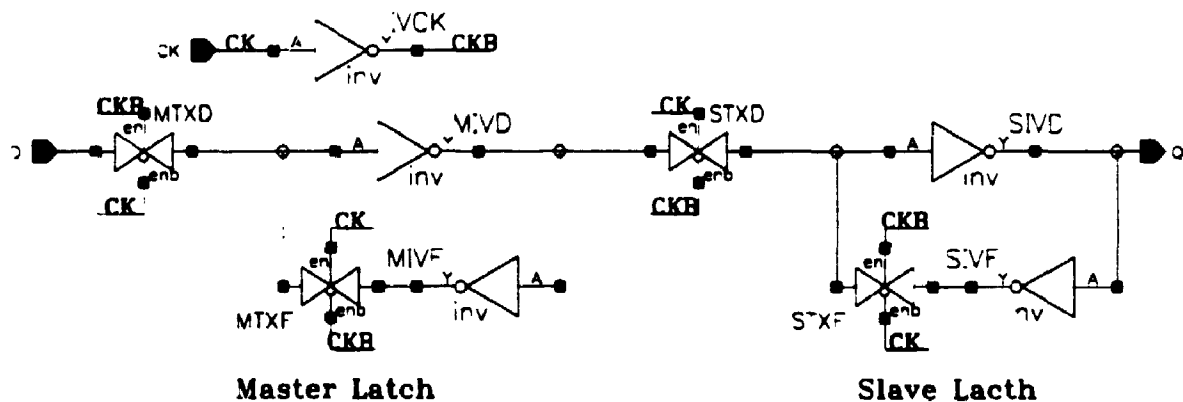


Figure 3.16 An example of a CMOS edge-triggered D flip-flop

Since the PE layout area is the most critical design constraint, edge-triggered flip-flops are ruled out as register implementation due to their large transistor count. The alternative is to use level-controlled latches, which use half (or less) of the elements of an ETFF. A typical gate-level CMOS D-latch structure is shown in Figure 3.17, where the control clock is labeled WD, for Write Data. During WD=HIGH the positive feed-back loop through transmission gate TXF is interrupted and the latch receives the new data. During WD=LOW, the input transmission gate TXD is off and positive feedback reinforces the logic state previously written. We remark also that during WD=HIGH, the latch output is the complement of the input, delayed by the propagation through one transmission gate and one inverter. This is the "transparent" state of the latch.

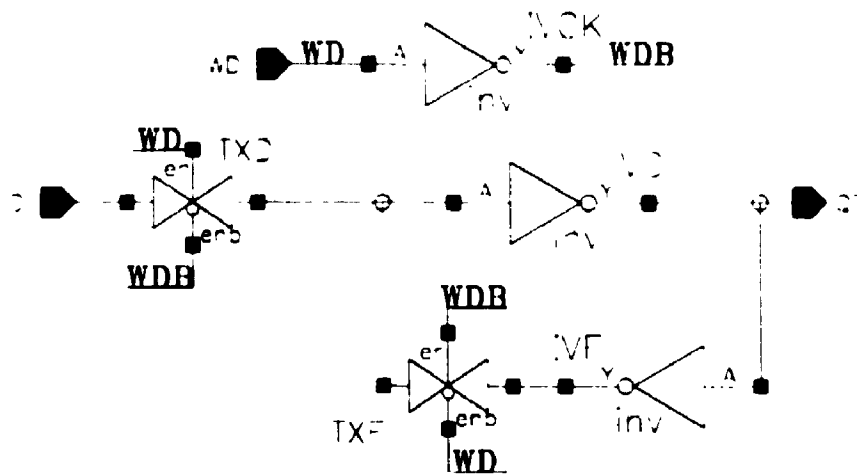


Figure 3.17 CMOS D-latch, requiring 10 transistors

Since now there is no second level of latching as in the edge-triggered flip-flop, a new latch input appears complemented after one inverter delay at the output of the latch, which is also one of the selection inputs to the ALU. Hence, a latch state change might modify the result of a pure combinational ALU. This in turn will affect the value written in the latch since the ALU propagation delay is very likely shorter than WL pulse duration. This loop is the logic hazard referred to above. A mechanism to remove this hazard has to be provided to inactivate the ALU during the write latch phase. This mechanism consists in timing the write latch phase so that it will not overlap with the ALU precharge and evaluation phases. The justification for this PE timing phase is as follows. We consider again the ALU multiplexor in Figure 3.10, and we assume that neither precharge nor evaluation are active. Then the precharge PMOST PPRCH is OFF hence there is no conductive path to v_{dd} . Also OFF are the bottom eight NMOSTs controlled by signals $TTOPCK\langle 7..0 \rangle$ so there is no conductive path to v_{ss} . In this case, any change on the six selection signals X, XB, Y, YB, M, MB will not affect the dynamic node logic value, except through charge-sharing. Since changes on X and Y and their complements XB and YB can happen only during the write latch phase, we conclude that this must be a third PE timing phase, non-overlapped with ALU precharge and evaluation.

Because of the synchronous registered nature of the base SRAM (presented in Chapter 4), selection signals M and MB are not in the same category as X, Y and their complements. In the local memory interface in Figure 3.1, there is no direct (i.e. combinational) connection between memory input MD and local memory output M (same as MQ). Memory output M only changes after a

read cycle, so it is not affected by a write cycle that updates a memory location with new data sourced by MD, connected to the ALU output. The effect on C*RAM timing is that there is no special constraint on the Write Memory phase, once the new ALU result is available. Obviously, the hold time on input MD relative to the memory clock has to be observed before starting the ALU precharge phase of the subsequent cycle.

Although the DYN5N circuit is not used in the BATMOS PE implementation, looking back in Figure 3.7 we notice that timing signals would have to be modified as a result of the necessity to turn OFF both the precharge and evaluate transistors during the write latch phase. ALUPRCKB can no longer be used to time the evaluation phase. Instead, an independent control signal ALUEVCK must be introduced to control the gate of NMOST NEVAL.

A graphic summary of PE timing is illustrated in Figure 3.18. The design of circuits that generate the three-phase sequence is discussed in Chapter 4.

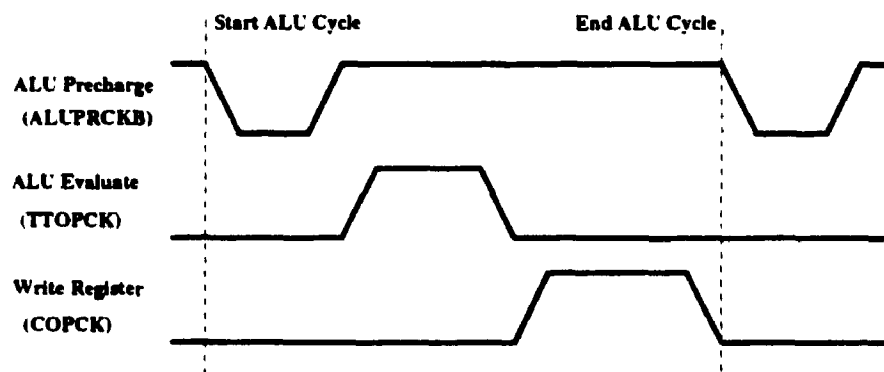


Figure 3.18 Timing of ALU operation followed by register write

3.5 Register Design

The example latch structure shown in Figure 3.17 can still be simplified in order to reduce its layout size. A full transmission gate (TXD and TXF in the figure) is an expensive gate since it requires four transistors: one PMOS/NMOS pair for the actual gate and another pair for the inverter that generates the clock complement (this inverter can be shared though by a number of gates, as is the case in the D latch). As a first step, the feedback transmission gate TXFB can be

eliminated. This leaves the two inverters connected permanently in the "back-to-back" positive feed-back configuration. The drawback of this circuit is that the driver that writes the latch has to overcome the state held by the positive feedback in order to write the opposite value. This will result in a combination of extra power and extra transition time in the process of changing latch state.

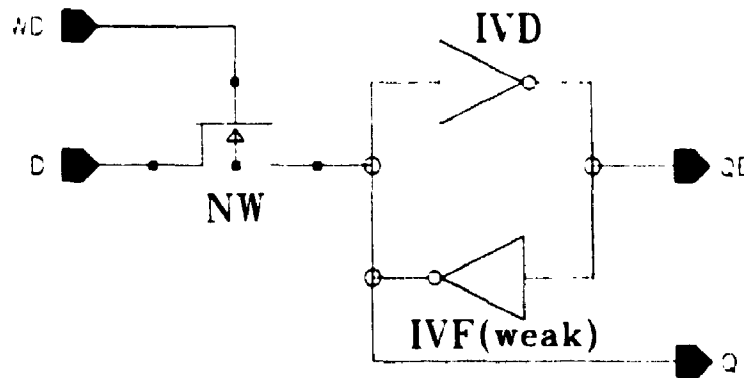


Figure 3.19 Final circuit for single-access D-latch

A second modification is to replace the input transmission gate TXIN with an NMOST pass transistor. This eliminates three transistors (the clock inverter is no longer needed), including two area consuming PMOSTs. The disadvantage of using a single NMOST as latch access (write) transistor appears when writing a HIGH logic state into the latch. The HIGH voltage level is degraded by a V_{nthb} voltage drop on the access NMOS, where V_{nthb} is the threshold voltage increased by the body effect.

3.5.1 Shift-Left/Right Implementation

As presented at the beginning of this chapter, the Baseline PE supports nearest-neighbor left/right communication between PEs. The circuit design problem is how to best (i.e. minimal area) implement this feature using the circuit resources already in place. A straightforward implementation of a shift register, as suggested by the function to implement, would again require edge-triggered flip-flops. The reason is again the necessity to isolate the input from the output, so that during one clock cycle the data propagates strictly from one stage to the next one, or equivalent, there is no data feed-through across multiple stages.

An obvious design decision is to use the X and Y latches for communications. The expensive alternative would be to use two separate latches to build the shift left/right registers. In this case, in order to operate on them, the contents of these extra two latches would have to be transported to X or Y, because these are the registers with direct access to the ALU.

Since X and Y are level-controlled latches, a sequence of X/Y latches cannot be cascaded (output connected to the input of next latch) to form a shift register, because of the feed-through problem. The solution is to use the timing already in place for the ALU and the fact that the outputs of X and Y are selection inputs into the ALU multiplexor. A shift (left or right) will be a sequence of two steps, involving an ALU operation:

- a. bring the contents of X (Y) to the ALU output, executing an Identity-X (Identity-Y) operation (see also Table 3.1 at the beginning of this chapter).
- b. write the ALU result into the neighboring X for shift left (neighboring Y for shift-right), during the write-latch phase.

We remark that the master-slave structure of an edge-triggered flip-flop is thus emulated by interposing an ALU between two latches, and using the non-overlapped phases of ALU operation (precharge+evaluate) and write-latch in a manner similar to the antiphase flip-flop clocking. This manner of implementing the shift left/right communication has one functional restriction: shift-left and shift-right cannot be executed simultaneously, since they both pass through the same ALU. Hence, we do not have the perfect equivalent of two hardware independent shift registers, which is the functionality price paid for this very compact circuit. On the other side, the shift operation is not restricted to X or Y only as data sources. Since what is shifted is the ALU result, any logic function of Y, X and M can be computed and the result transmitted to the neighboring PE.

3.5.2 Dual-access latch

There are now two input data sources for each local register, the local ALU and the neighboring ALU. The minimal way to implement this second input is to add a second access NMOST connected to the input node, as shown in Figure 3.20. A number of layout configurations were investigated for this double-access latch. The conclusion was that adding a second access NMOST

creates a highly irregular layout, with a large inactive area surrounding the latch. Even the layout of a pair of such latches was still irregular.

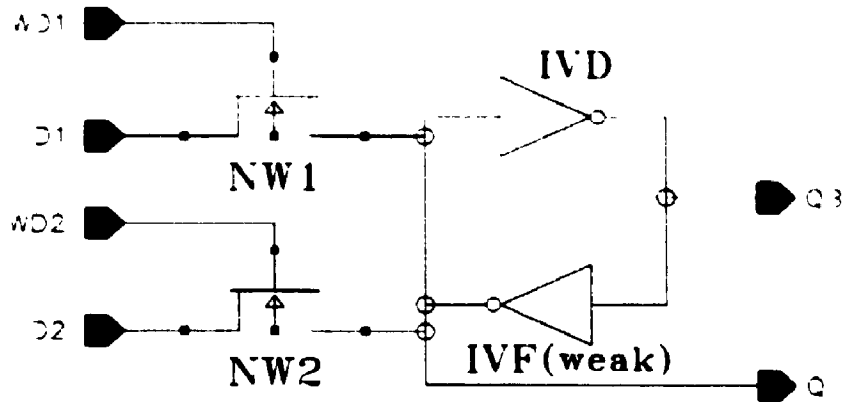


Figure 3.20 Dual-access D-latch

An alternative circuit suggested by this layout problem is shown in Figure 3.21. Here, the second input is connected to the “complement” node of the latch. The structure is now symmetrical, and so is the layout. There are, though, both logic and circuit issues with this structure. In the logic plan, the second input has to be driven by the complement of the value to be stored in the latch. Since the PE latches are only written by the ALU output (local or neighbor), this implies that an extra inverter is needed in the ALU to generate the result complement ALUB. In designing the PE layout, the addition of a second inverter proves to be inexpensive because of the regularity of the circuit and localized connections.

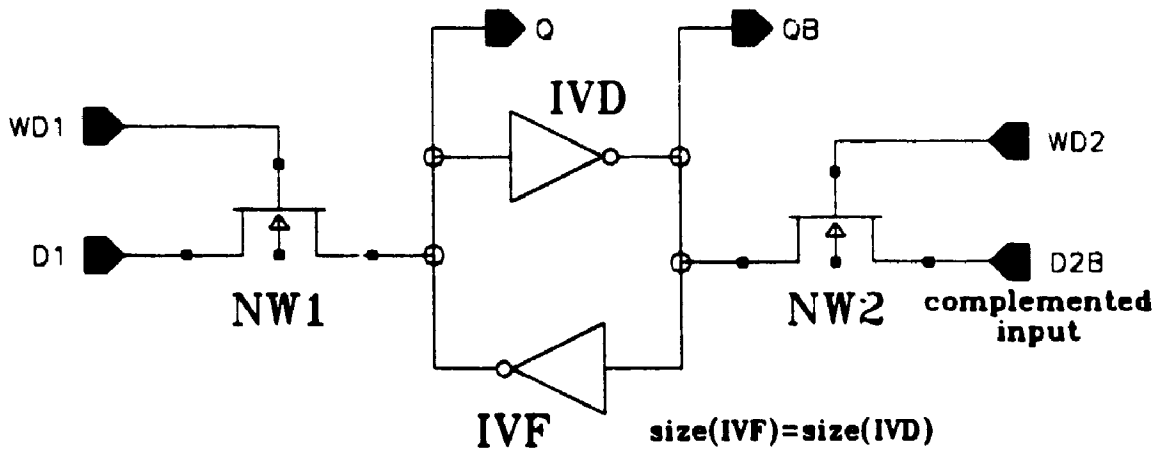


Figure 3.21 A symmetric dual-access D-latch

The circuit problems appear when we consider the fact that the latch has to be symmetric. There is no longer a "weak" feed-back inverter, the two inverters are equally sized since both latch nodes are now input nodes. The resulting structure is identical to the six transistor SRAM cell (6-T SRAM). The main difference is that a 6-T SRAM cell is written differentially, i.e. the two data inputs are always complements of each other and the two access transistors are turned on by the same control signal (termed wordline in RAM design). In our case, the two inputs are independent, hence the write access is single-ended. Since the latch is now symmetric, writing the latch with the complement state (latch flipping) is more difficult. This is especially valid for writing a HIGH logic state into a latch holding a LOW, because of the V_{th} voltage loss on the access NMOS.

3.5.3 Dual-Access Latch Transistor Sizing

The combination of data write driver, access NMOS transistor and dual-input latch is essential for the PE function, hence it will be analyzed in more detail in the following. The objective is to size the transistors in order to obtain reliable latch writing, while keeping the layout area to a minimum.

We start by choosing minimum physical layout for the four transistors in the latch, i.e. minimum width and length in BATMOS. A number of Spice simulations are run in order to determine the write (access) NMOS transistor size, w_{nwr} , while examining the write delay time and the power consumption. The critical write operation is overwriting the latch holding a logic-0 with a logic-1 state.

The simulations show that below $w_{nwr}=2.4 \mu\text{m}$ the latch cannot be forced to change states (in the simulation, the driver PMOST has $w=8 \mu\text{m}$). Therefore, the range of variation for w_{nwr} starts at $2.4 \mu\text{m}$ and ends arbitrarily at $4 \mu\text{m}$, to keep the PE length as small as possible. Since the write transistor gate runs along the PE length, a wider transistor means a longer PE. Simulations are run for $w_{nwr}=2.4, 3.2$ and $4.0 \mu\text{m}$. The waveforms of two ALU operations with different results each followed by a latch write are shown in Figure 3.22. The current consumption of the circuitry in one PE is also represented (bottom waveform). It can be seen that writing a logic-1 in the dual-access symmetrical latch is the longest transition in the PE cycle.

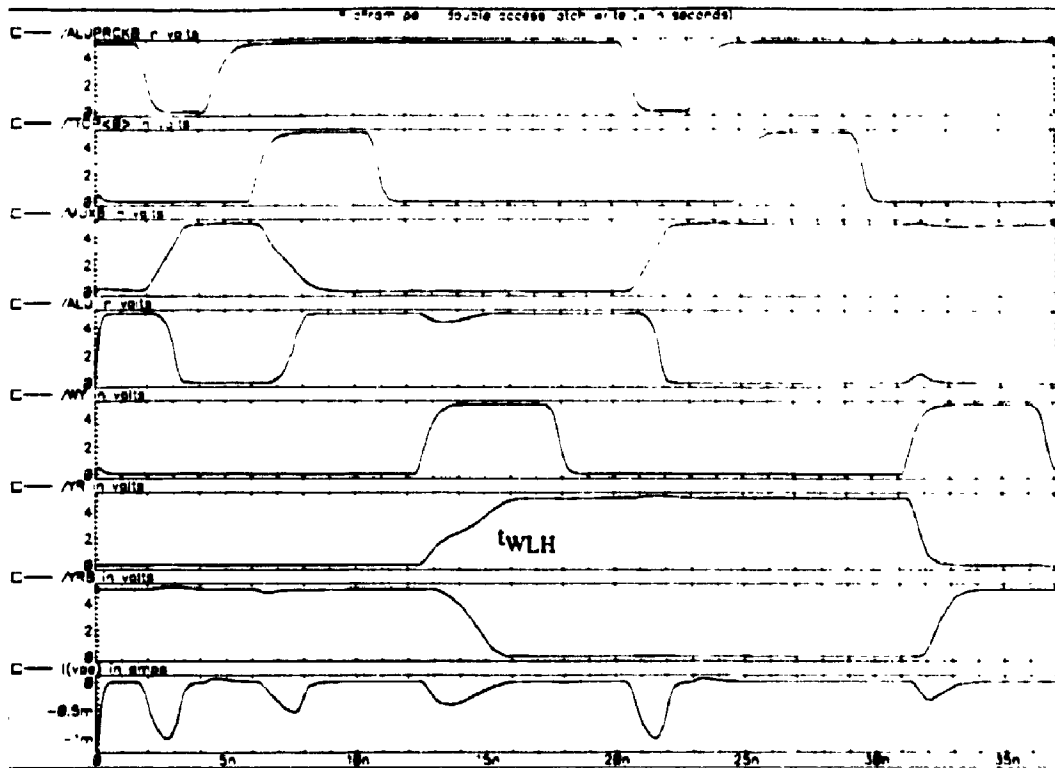


Figure 3.22 Writing a "1" then a "0" in the dual-access latch (YR waveform)

Figure 3.23(a) shows the latch LOW-to-HIGH write time, t_{WLH} , for the three values of write NMOST width. Figure 3.23(b) presents the energy spent in overwriting the latch, for both types of logic state reversal. It can be seen in both figures that using the largest write transistor ($w_{nwr}=4 \mu\text{m}$) results in the fastest write transition and also in the lowest power dissipated in the process. For the non-critical HIGH to LOW transition, the transition time and power consumption are approximately the same, irrespective of the access transistor width

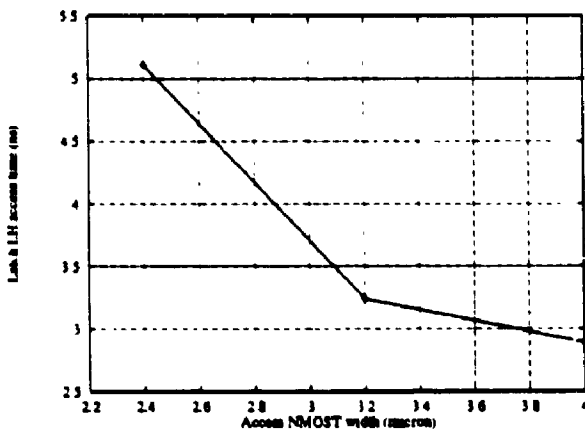


Fig. 3.23(a) LOW-HIGH latch write time t_{WLH} function of access NMOST width

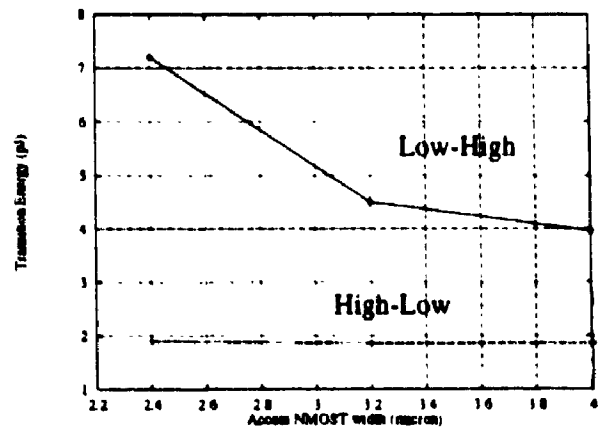


Fig.3.23(b) Energy spent in latch state reversal

The final transistor size selection in the PE is done for the ALU inverting buffer, left to this section because of its participation in the latch write operation. First, a relative PMOS/NMOS=2 width ratio is chosen. The normal practice is to choose PMOS/NMOS=2.5 in order to equalize transition times and bring the switching threshold in the middle of the voltage range. In [Schu93] though, a lower ratio is recommended for low-power CMOS design. Simulations are run for three widths, 4/2, 6/3 and 8/4, while again the critical latch write operation is examined. The results are shown in Figure 3.24(a) for the write delay t_{WLH} and Figure 3.24(b) for energy consumption during the transition. It can be seen that the size of the ALU inverting buffer has little impact on the latch write, with slightly better delay and power consumption for the biggest inverter. We could hence choose the smallest size, if a single latch write is a considered. For PE programming flexibility, we want to reserve the possibility of the ALU writing both latches (X and Y) at the same time. In this case, the ALU driver has to be large in order to fight two NMOS transistors in the worst case (both latches in logic-0). We choose hence an ALU inverter with an $8\mu\text{m}/4\mu\text{m}$ PMOS/NMOS width ratio.

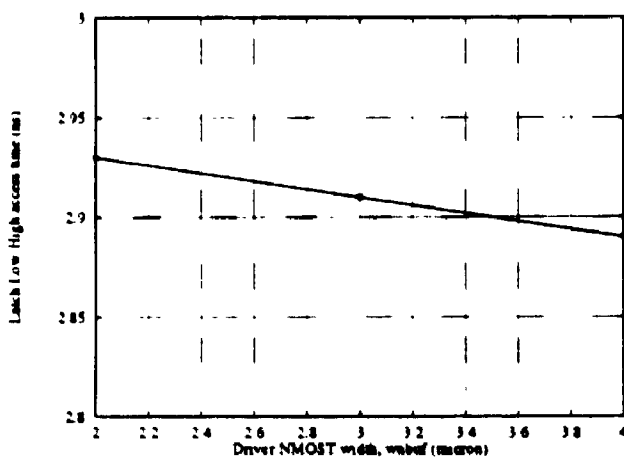


Fig. 3.24(a) LOW-HIGH latch write time t_{WLH} function of write driver size

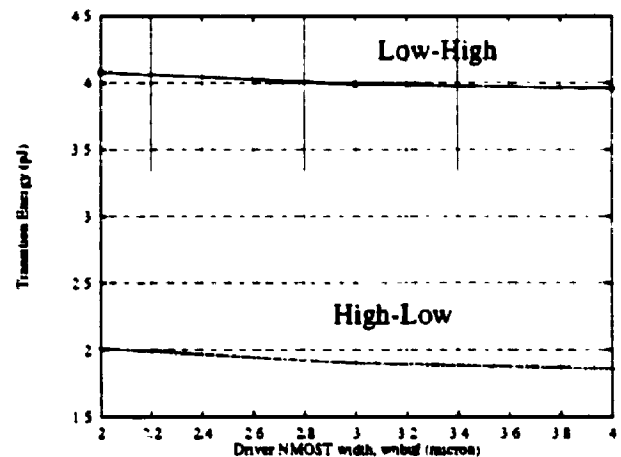


Fig.3.24(b) Energy spent in latch reversal function of write driver size (NMOST width)

3.5.4 Single-Access Latch Transistor Sizing

Register W (Write Enable) is written by a single source. The single access D-latch (Figure 3.19) can be used in this case. The direct path inverter IVD has minimum size transistors. The weak feed-back inverter IVF is obtained by using transistors with minimum width but more than minimum length. The layout constrains this length to $1.6\mu\text{m}$. The complemented ALU output ALUB

is routed as latch writing source. The inverting direct path of the latch will drive thus the ALU logic value as local write enable to the memory interface.

The layout of the X and Y register pair is shown in Figure 3.25

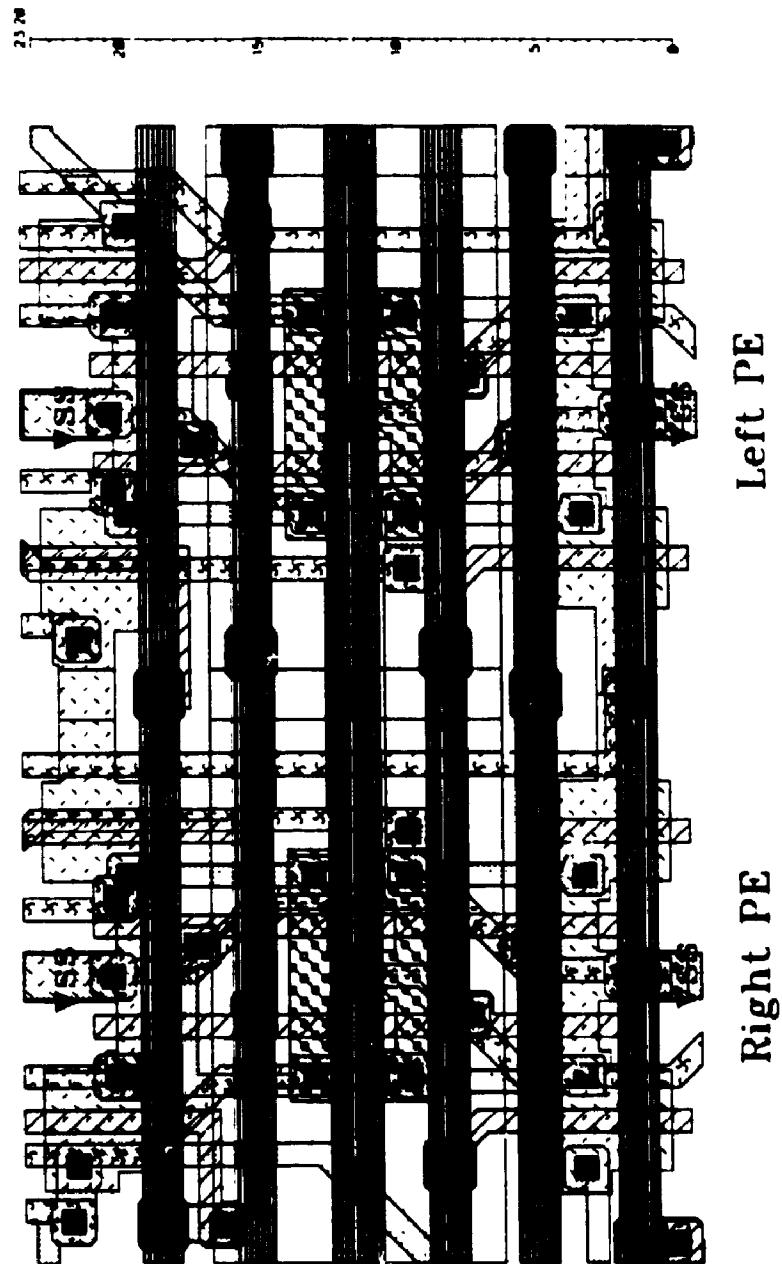


Figure 3.25 Layout of X and Y registers in two adjacent PEs
(vertical of page is layout horizontal)

3.6 Bus-tie Circuit Implementation

The second form of interprocessor communication defined in the Baseline PE is the **one-to-many bus-tie circuit**. The actual logic function realized by bus-tie is either **wired-AND** or **wired-OR**, depending on the actual PE circuitry. The BATMOS C*RAM chips implement the **wired-OR**, relative to the ALU outputs. It is indeed a form of interprocessor communication, because the **result** is fed back to all PEs, so that at the end of a bus-tie cycle all PEs have the same ALU result. A symbolic representation of the bus-tie circuit is shown in Figure 3.26, suggesting the actual circuit implementation based on a precharged line. The transmission gates indicate bidirectional links between the precharged ALU multiplexor nodes MUX8<i> and a common bus-tie line BT.

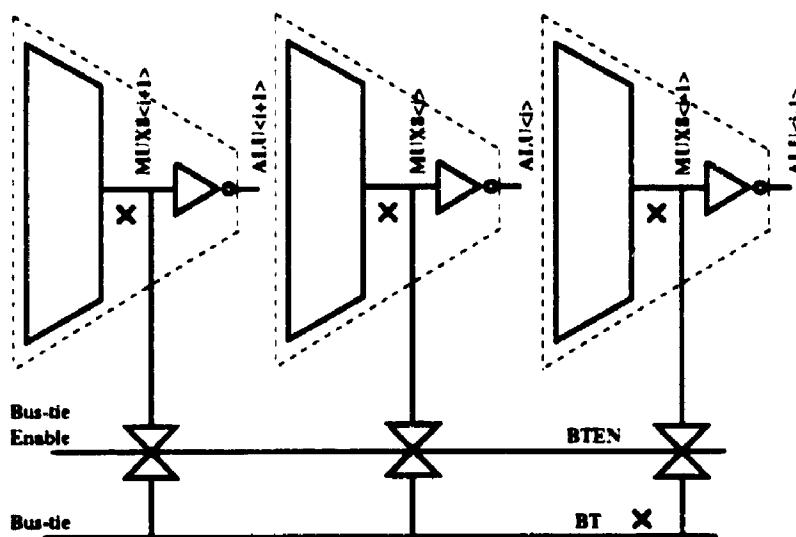


Figure 3.26 Symbolic diagram of Bus-tie circuitry (x=precharged nodes)

3.6.1 Circuit alternatives

We examine first for completeness a CMOS static logic realization of a global OR function in a 64 PE array. This would imply having a 64 input static OR gate for the 64 PE BATMOS C*RAM and an extra 2-input OR gate for every PE to feed back the result. The 64-input OR gate, implemented practically as a tree structure (with 2- or 4-input OR gates), would add a large area overhead required by routing channels. It would also create layout mismatching with the **linear PE array structure**. For these reasons, the static 64-input OR is totally impractical for bus-tie implementation.

A much more realistic solution to the large input number OR/AND problem in memory design involves using dynamic logic with precharged busses. Figure 3.27. The circuit is similar to the dynamic logic multiplexor discussed in section 3.1, with two differences: 1. there is a higher number of NMOS discharge branches to ground, typically 64-256 (as opposed to 8 in the 8-to-1 multi-

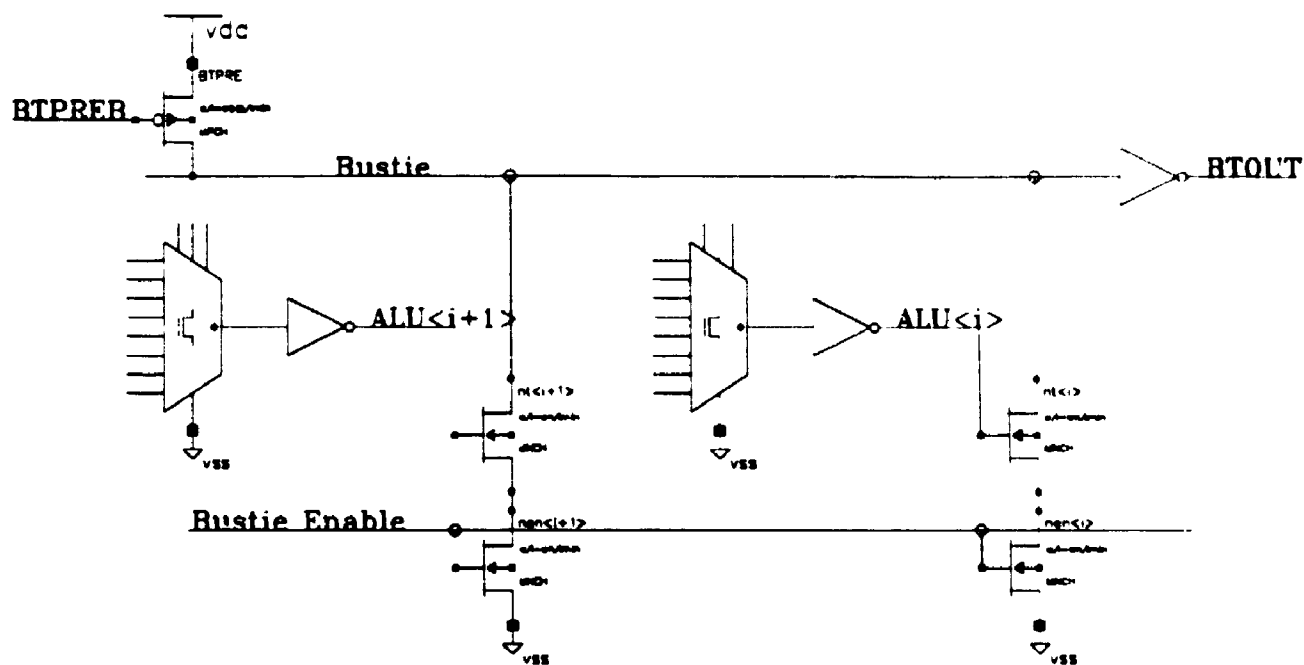


Figure 3.27 Global wired-OR with precharged bus

plexor); 3. the evaluation branches only have two NMOS transistors, one is the actual OR gate input while the other is the evaluation enable transistor (this is the case of the NMOS access transistor in the 6-T SRAM cell, relative to the bit-lines). The precharge PMOS BTPRE brings the Bustie line HIGH before a bus-tie cycle, under the control of active LOW signal BTPREB. Non-overlapped with the BTPREB pulse, the active HIGH Bustie Enable line controls the evaluation phase.

This circuit solves the problem of generating a global OR in a compact layout, but it is not an actual "bus-tie" since it does not modify the ALU outputs (it is not bidirectional). A circuit that attempts to do this function is shown in Figure 3.28. Here, the bus-tie circuit is an actual transmission gate, connecting the precharged node of the ALU multiplexor to the precharged BUSTIE

line. In the worst case bus-tie operation, one ALU evaluation path (4 series NMOSTs) should dis-

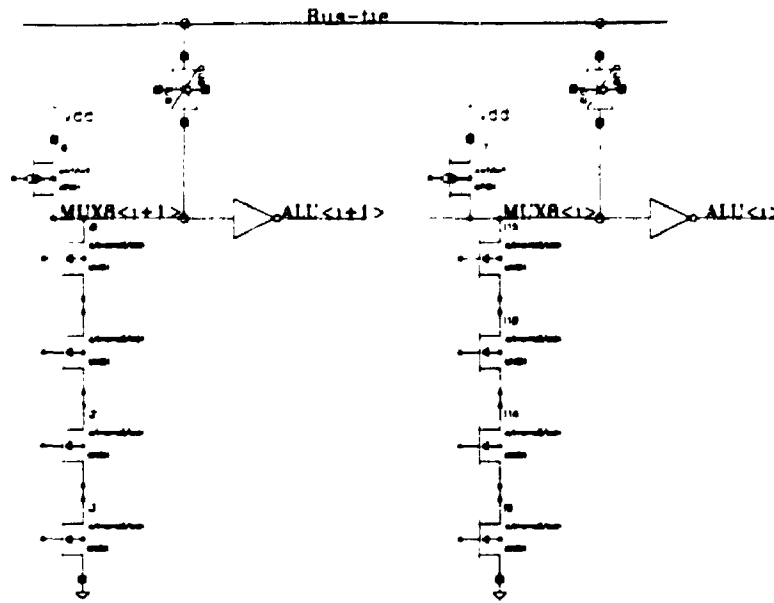


Figure 3.28 Bidirectional bus-tie with transmission gates

charge the Bus-tie line through the local transmission gate, and all other 63 precharged ALUs through one more level of series transmission gates. For the BATMOS C*RAM using pseudo-dynamic ALUs with latch-back PMOSTs, the circuit in Figure 3.28 will practically not manage to realize bidirectionality. This happens because in the worst case bus-tie there are 63 parallel latch-back PMOS transistors opposing the single discharge path of four series NMOSTs. The conclusion is that the bus-tie circuit should be “active”, i.e. include a buffering/amplifier gate.

The circuit proposed by Elliott in [Elli92] as bus-tie implementation is shown in Figure 3.29. This active “transmission gate” is composed of two NOR2 gates, *penor* and *bmor*, and two NMOS discharge transistors. The functioning is as follows: assume ALU<k> has a result of logic-1, which means the precharged node MUX8<k> has been discharged LOW. If the Bustie Enable line is activated (LOW, for this circuit), then the output of nor gate *penor* goes HIGH, turning ON the NMOS transistor connected at its output. This creates a discharge path for Bus-tie, which starts going LOW. At some point in this transition, the outputs of nor gates *bmor* start going HIGH, thus turning on the associated NMOS transistors. This in turn will create a discharge path for each ALU multiplexor: all MUX8<i> nodes that have not already evaluated LOW as result of their opcode start discharging. As a result, the actions presented above start happening for all PEs. in a regenerative manner. The final state has the Bus-tie line and all MUX8 nodes discharged,

which corresponds to all ALU lines HIGH.

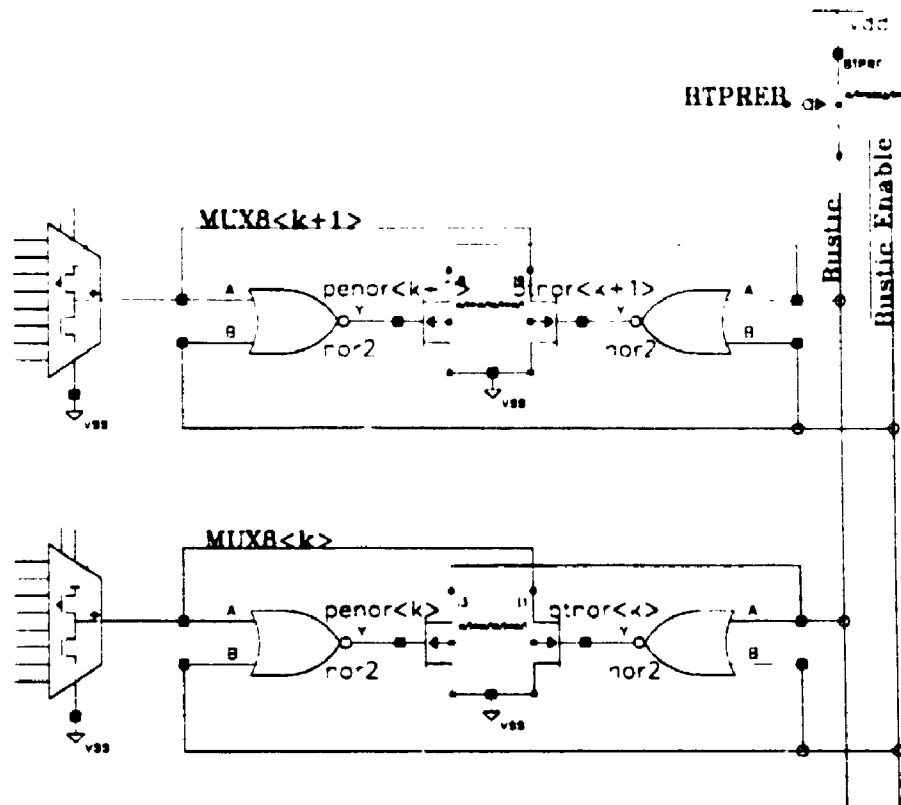


Figure 3.29 Active bus-tie circuit proposed by Elliott (two shown)

From the above qualitative description, it can be inferred that the more MUX8 nodes are discharged initially, the faster the bus-tie operation is. The worst-case corresponds to the "one-to-all" case, with just one MUX8 evaluating LOW initially. The C*RAM timing obviously has to cover this worst-case operation. Actual waveforms are presented below, for the final bus-tie circuit.

3.6.2 Proposed Bus-tie Circuit Implementation

Examining the previous circuit (Figure 3.29) in relation with the desired bus-tie function, we observe that a more compact circuit can be designed by accepting a different area-delay trade-off. Since the circuit uses two nor gates to implement the active bus-tie, there are four PMOSTs which impose a large area penalty. We are looking for an implementation using the minimum number of transistors, and in particular the minimum PMOS transistor count.

The new circuit is shown in Figure 3.30. It uses one PMOS transistor and three NMOS transistors.

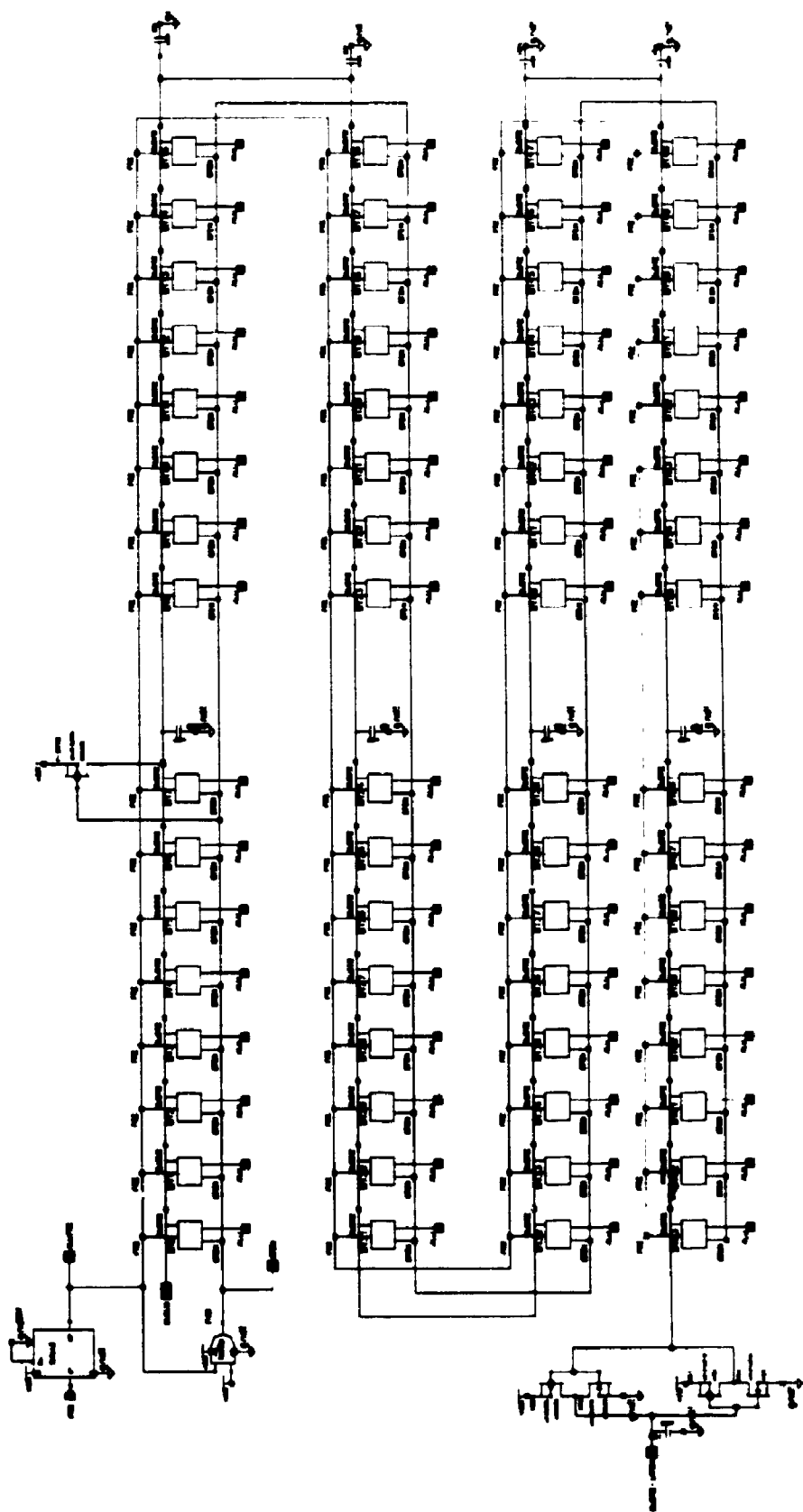


Figure 3.31 Array connection of 64 Bus-tie circuits

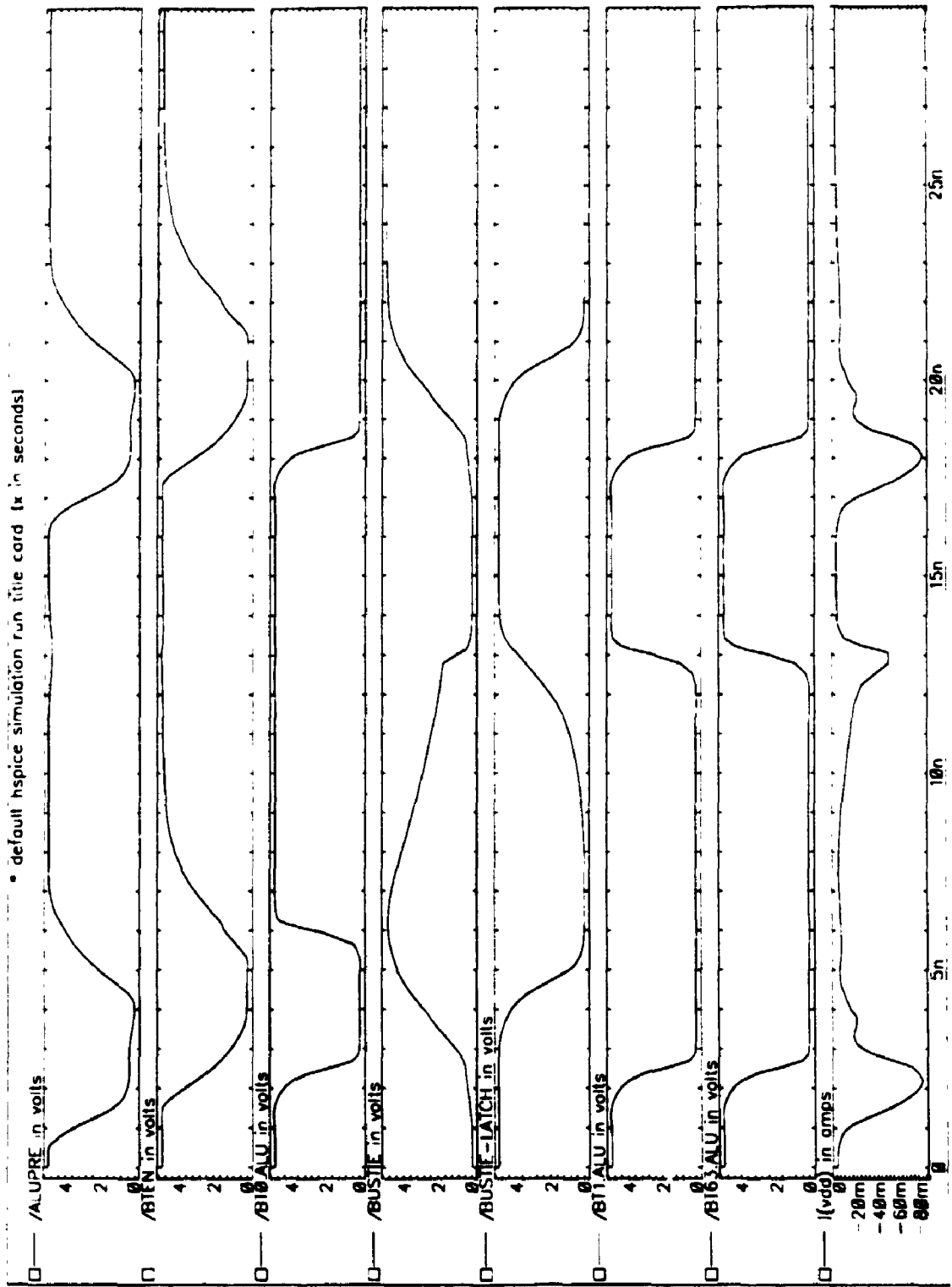


Figure 3.32 Spice simulation of worst-case 64 PE Bus-tie cycle

delaying effect of a back-to-back inverters latch connected to the Bus-tie line (Figure 3.31, lower right side). As in the previous pseudo-dynamic circuits, the latch was added in order to protect a HIGH result against leakage currents. 1. Bus-tie discharge waveform shows a longer initial slope, while the single evaluation path is active, followed by a shorter but steeper section at the end, once the positive feed-back loops involving all other ALUs have closed.

Looking at the supply current waveform, we notice peaks during ALUs and Bus-tie precharge (84 mA) and during the final part the regenerative feed-back (51 mA). The Bus-tie line precharge PMOS transistor has a width of $40\mu\text{m}$ in order to insure the fast precharge of the large capacitive load. The final layout of the bus-tie circuit is shown in Figure 3.33.

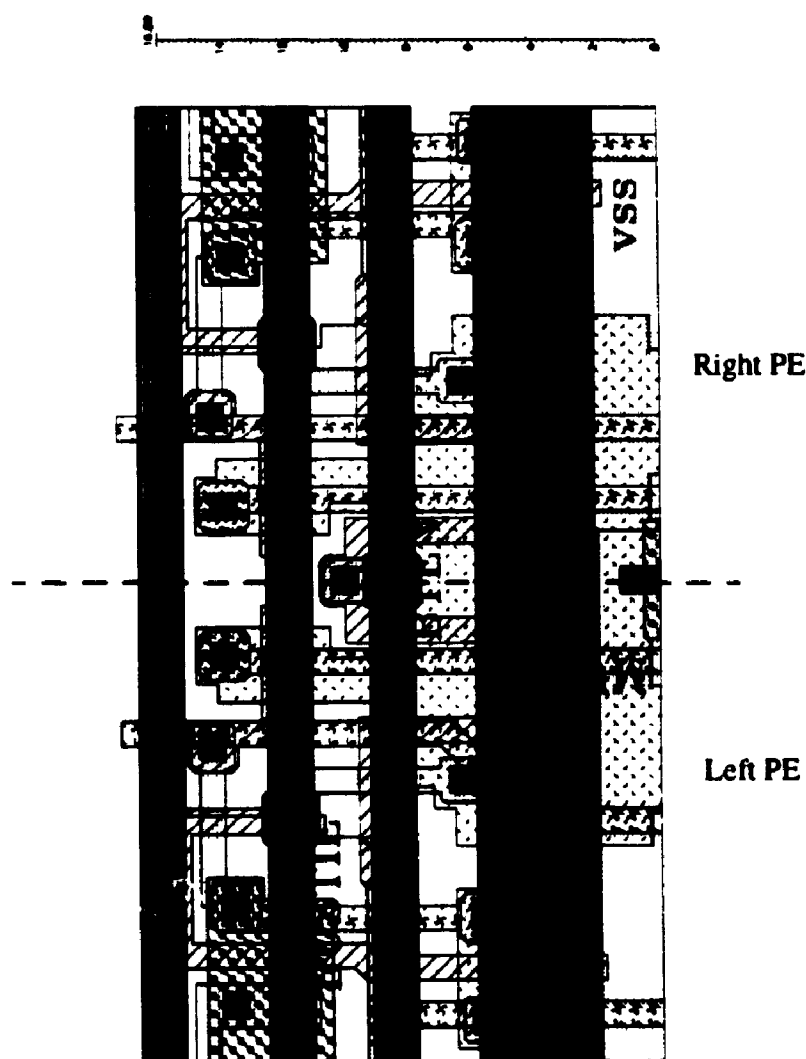


Figure 3.33 Bus-tie circuit layout in two adjacent PEs

3.7 Processing Element Layout

A large part of the overall design effort went into creating a very dense processing element layout. As mentioned above, this goal has influenced a number of circuit design decisions, especially the dual access latch and the shift left/right circuit. The main layout constraint is imposed by the pitch of the memory sense amplifier, which sets the width of the PE layout. For the SRAM memory used in this project, this parameter is in the order of 40 μm . Since the PE width is thus fixed, minimizing layout area is equivalent to designing a minimum length PE.

One of the issues of interest related to PE layout is its portability to other processes. Obviously, the PE layout is dependent on the BATMOS design rules. The main factor though is the metallization system, i.e. the number of metal layers and the nature of contacts/vias (stacked or not). The PE layout area, as expected, is extremely dependent on the three metal layers available in BATMOS and on the availability of stacked contacts and vias. Some of the interconnections are made using device diffusion wells and the gate polysilicon layers. These layers do not pose problems in porting, since they are available in any CMOS process. Hence, PE layout porting is feasible towards other processes with a similar metallization system, after appropriate scaling to account for different design rules. Such processes are fairly current today (1994), as can be seen by consulting the documentation of various gate array ASIC manufacturers (IBM Microelectronics even offers five metallization layers).

A PE symbolic floor-plan is shown in Figure 3.34. Approximately half of the area is taken by the ALU multiplexor. The main difficulties in laying out the PE are encountered during interconnection routing. The first metal layer (METAL1) is used for local connections and mostly horizontal short connections. A typical use is internal latch wires. METAL2 is used to route mostly vertical connections. These first two metal layers require 0.8 μm width and 1.2 μm spacing [Hada91] for a minimum pitch of 3.0 μm . The global connections are routed in the lowest resistivity TOPMETAL. The global lines run horizontally across the PE array. TOPMETAL imposes a minimum width of 1.0 μm and spacing of 1.4 μm . Larger values are used for both TOPMETAL width and spacing in order to minimize electromigration and the probability of manufacturing errors. Very localized connections are made using the GATE layer (polysilicon). Typical examples are again latches and also the ALU multiplexor. A limited number of connections are made through the

device diffusion layer.

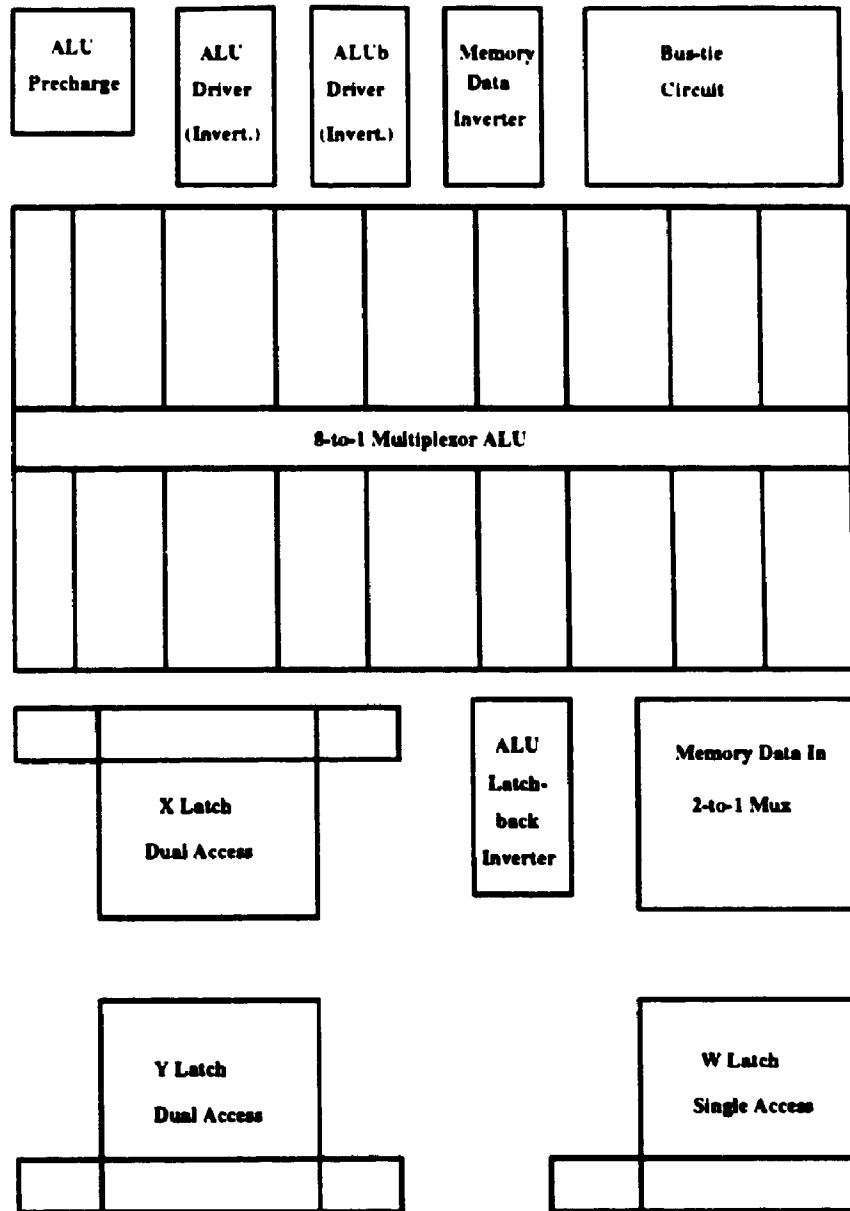


Figure 3.34 Processing Element layout floor-plan
(vertical of page is also vertical of layout)

The supply/ground connections run horizontally in METAL1 or TOPMETAL, with vertical stubs in METAL3. In general, the connection density in both directions reaches the maximum over the ALU multiplexor and the support circuitry at the PE top, and is close to maximum in the remaining area. Maximum density means that no extra connections can be routed in the specified area.

Two PEs are arranged in a vertically mirrored configuration. The main benefit is area compaction as a result of two PEs sharing common elements. Examples of such elements are the source diffusion of the ALU precharge transistor and the source diffusion of the large evaluation transistor in the bus-tie circuit. Also shared are connections from the TOPMETAL global control lines, and segments of interconnect originating from these. Beside reducing layout area, the shared TOPMETAL contacts result in a smaller parasitic capacitance on the global lines.

The final layout is shown in Figure 3.35 for a pair of PEs, which is the basic cell in building the array. The area taken by one PE is approximately $40 \times 84 \mu\text{m}^2$. This area includes two 2-to-1 multiplexors sourcing the memory data and write enable inputs, circuits that will be discussed in Chapter 4. For comparison, the PE designed in the $1.2 \mu\text{m}$ double-metal CMOS4S technology [EILP92] occupies an area of $36 \times 552 \mu\text{m}^2$. The BATMOS PE takes thus less than 17% of the CMOS4S one. It should be noted here that the CMOS4S PE did not implement the shift-left and shift-right communication. As discussed in section 3.3.2, the implementation of shift operations imposes the design of a dual-access latch, which in turn requires a trade-off between area and power/speed, with high costs for both options. Also, shift-left/right requires two more routing channels between PEs. Factoring these two elements in the comparison, it is estimated that for complete implementations the BATMOS PE would take less than 12% of the CMOS4S PE. The main reasons for this large area reduction are:

1. the extra level of metallization available in BATMOS;
2. the availability of stacked contacts and vias (CMOS4S requires non-overlapped contact and via);
3. the amount of layout work involved in designing the PE; and
4. the smaller $0.8 \mu\text{m}$ feature size of BATMOS compared to $1.2 \mu\text{m}$ in CMOS4S.

As another reference point, the 8-bit processing element in the IMAP chip (see section 2.2.3) has about 7,000 transistors and occupies an area of $419 \times 2,628 \mu\text{m}^2$ [Yama94]. Compared to the 76 transistors BATMOS Baseline PE, the IMAP PE has 92 times more transistors while its area is 327 times bigger, despite the lower $0.55 \mu\text{m}$ feature size, but partly justified by the availability of only two metal layers.

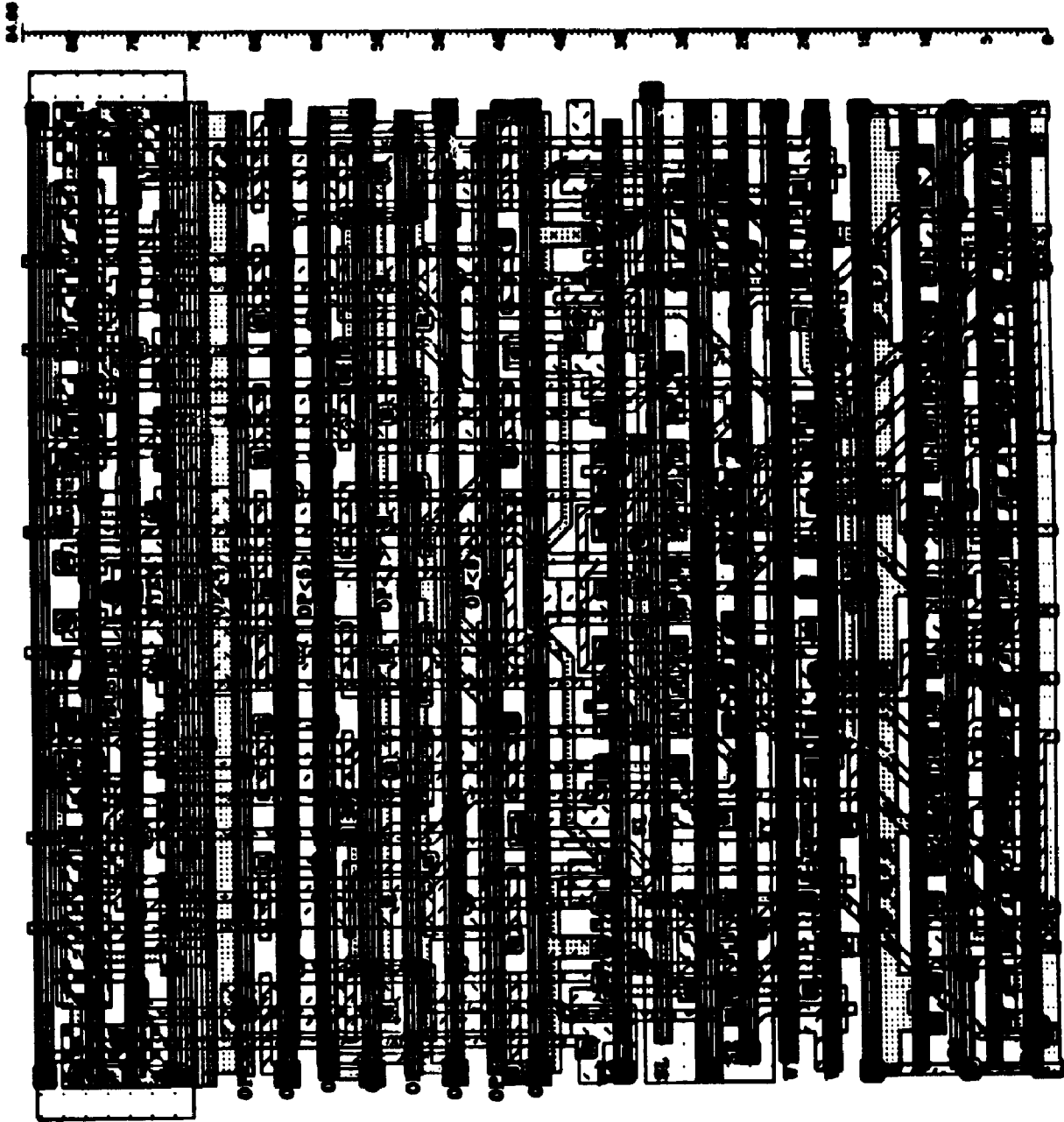


Figure 3.35 Dual Processing Element layout (layout' vertical is page horizontal)

3.8 Summary

The C*RAM Processing Element proposed by Elliot [Elli92] was implemented in a 0.8 μm triple metal level BiCMOS technology (Northern Telecom's BATMOS). Use of precharged logic blocks, layout-aware circuit design and careful full-custom layout have resulted in a very compact 40 x 84 μm^2 PE area. Functional safety is insured by latching the precharged nodes, accepting thus a small area, speed and power penalty in exchange for static-like operation.

Chapter 4

BATMOS Computational RAMs

4.0 Using a Platform RAM

In order to design a Computational RAM, the PE array has to be grafted to a given memory, or the memory and PEs are codesigned from scratch. The second approach will very likely yield a high-performance C*RAM, while requiring much more design resources. Memory design is at the border between digital and analog worlds. A good memory design, with high density and yield, requires experience with the given technology, and this can only be obtained during a number of design/fabrication iterations. The context of the present work - one-man team, academic environment, limited time-span - required the limited resources to be directed towards implementing the new concepts. The alternative might very well be an ambitious try with no results to advance the project. The conclusion is that using an existing memory design meets best the overall goals of the C*RAM project.

This work benefitted from a collaboration with the Memory Design Group in Bell-Northern Research... This group supports BNR's ASIC design activity with modular multi-port embedded SRAM and DRAM macro-cells. These designs are described in two papers [BNRM92], [BNRM93], and some of the information required to understand C*RAM design decisions will be reiterated in section 4.1.

For C*RAM, the main thing of interest in memory is the number of sense amplifiers that can be activated at once. This sets the upper limit on the number of PEs on chip, a limit attained if the PE layout width (pitch) can be made as fine as the sense amplifier pitch. The maximum number of sense amplifiers in a BNR SRAM memory module is 64. This means that a memory block can supply at most 64 PEs. In order to support a larger number of PEs, the sense amplifiers and column decoder would have to be modified, by redesigning smaller pitch column circuitry. Modifying the BNR design amounts to modifying dense full-custom layout circuitry driven by tight self-timed clock signals. Again, this was deemed to involve a large amount of work outside the main thrust of the project, increasing the overall risk. A design decision was made not to attempt modifications of the baseline memory. The alternative for integrating more than 64 PEs on a C*RAM is to use multiple memory modules.

4.1 BNR's Modular SRAM

BNR's RAM design is in fact a family of modular SRAMs, including single-, dual- and four-ported modules, using similar building blocks in the periphery. The SRAMs have a synchronous (i.e. clocked) interface, which makes the rest of the systems to see the memory as an addressable edge triggered register. Once the memory is activated by the rising edge of the clock (Figure 4.1), a self-timing scheme generates all other timing signals and completes the cycle irrespective of the clock duty cycle. The self-timing circuitry uses model paths to track process, supply voltage and temperature variations, resulting in a maximum operating frequency of 180 MHz. This tight self-timing scheme is one of the reasons modifications in the memory design were avoided.

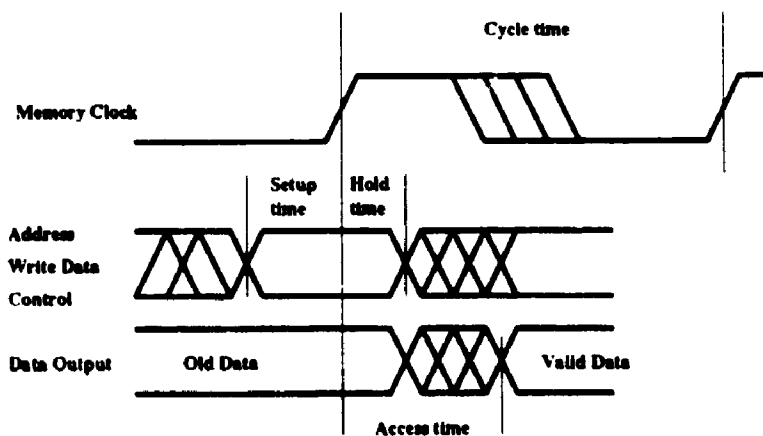


Figure 4.1 BNR's synchronous SRAM timing

The single-port SRAM core memory cell is the typical 6-transistor CMOS memory cell. Since the BATMOS process does not have any memory specific enhancements, such as high-resistivity polysilicon loads, the cell area is rather large, $131.3 \mu\text{m}^2$ [BNRM92]. For comparison, SRAM cells designed using a process with similar feature size ($0.8 \mu\text{m}$) but including SRAM-specific enhancements have an area of $44 \mu\text{m}^2$ [Wada87]. It should be noted here that a recent analysis of SRAM design for low-power [Taka93] points towards a return to the 6-transistor CMOS cell when low-power is essential, since its static power consumption is three orders of magnitude lower than that of the high-resistivity polysilicon load cell.

4.1.1 SRAM Modularity Parameters for C*RAM

The modularity range for BNR's SRAM family is presented in Table 4.1 [BNRM92].

Table 4.1 Platform SRAM Modularity Range

Parameter	Min	Step	Max
Words	8	1	8K
I/O Bits per word	1	1	64
Total bits	8	-	64K
Rows	4	2	256
Columns	2	2	256
Columns per I/O bit	2	2	32
Row address bits	2	1	8
Column address bits	1	1	5

a. Single-port vs. multi-port

The dual-port option is an interesting one, since it maps well with the dual nature of C*RAM: computation and storage. The two ports are totally independent: for example, a read cycle can be executed by both ports simultaneously on the same cell. This is an obvious advantage for the C*RAM system, since computation can be done at the same time with reading previous results or writing new data for future computation. As expected, the disadvantage of the dual-port SRAM

resides in its larger $197.8 \mu\text{m}^2$ memory core cell. This is an increase of 51% over the single port cell area. Since the extra memory cell area would result in a smaller PE local memory, the dual-port option is rejected in favor of the single-port memory. The dual-ported C*RAM might find use in very specialized applications, involving high-speed real-time data processing.

b. Word length (n)

As mentioned before, this parameter specifies the number of sense amplifiers active in a read cycle, hence the maximum number of PEs. We choose the longest word length available, $n=64$.

c. Rows and columns (r and c)

We are interested in having the largest local memory, once the number of PEs has been decided. Hence we choose the biggest memory block available, which has $r=256$ physical rows and $c=256$ physical columns. Since we already chose the word length $n=64$, the resulting logic organization of the memory is 1024 words \times 64 bit/word. There are 2 column address lines and 8 row address lines for a total of 10 address lines for the memory block. The column decoder selects one bit from every group of four physically adjacent columns, for a total of 64 out of 256 columns decoded in one memory cycle.

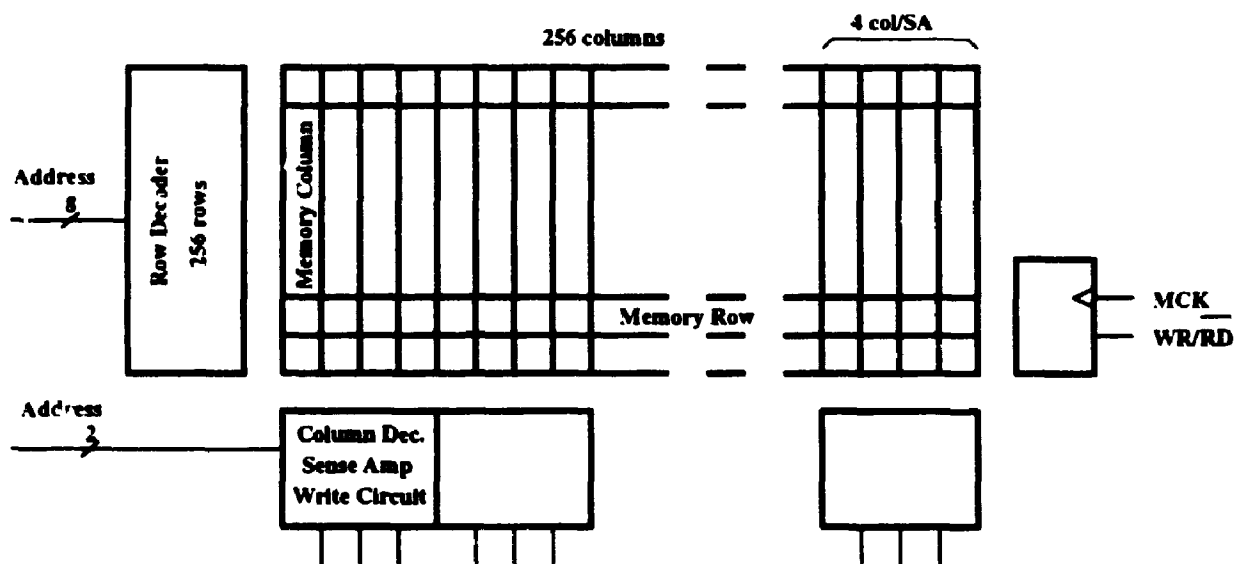


Figure 4.2 SRAM organization for C*RAM (C64p1k)

For the 512 PE C*RAM, memory size had to be traded off in favor of the higher PE count. The chip uses eight memory blocks, each with half the number of rows and the same number of col-

umns (hence still 64 PE per block). There are thus seven row address lines for a total of nine. The same half maximum size block was used for the Cs64p512 C*RAM, because of the limited silicon area available.

The local interface between the memory sense amplifier/write circuitry and the processing element is shown in Figure 4.3. There are separate lines for data input and output. The memory output (MQ) is registered, i.e. it keeps the value of the last read cycle until a subsequent read cycle. (as can be seen in Figure 4.1). There is a Local Write Enable (WE) mask that qualifies a Global Write Enable. During a memory write cycle, write is inhibited for the bits that have WE=LOW.

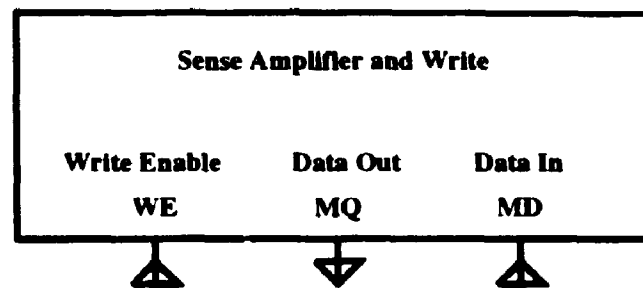


Figure 4.3 1 bit memory interface

4.2 C*RAM Data Input/Output

The chosen value for the C*RAM word length, $n=64$, would normally require a 64 bit wide input/output data bus and 64 dedicated pins on the package. This will result in a package with around 100 pins, after adding the address and control lines and an appropriate number of power and ground pins. A large number of such supply pins is necessary to attenuate the power/ground voltage bouncing, triggered by a large number of output pads switching simultaneously (as a rule of thumb in BATMOS, there should be one pair of supply pins for every four output pins. This rule should be qualified by the output driver speed - slower speed results in a smaller bounce). The large package would defeat the initial requirement of compatibility with typical memory chip packaging, in order to build compact C*RAM arrays on small PCBs (in SIMM manner). Also, a 64 bit internal data bus would occupy a fairly large amount of silicon: assume at least some segments are routed in TOPMETAL with the minimum pitch of $2.4\mu\text{m}$; then 64 parallel lines will take $153.6\mu\text{m}$, which is almost twice the PE length.

For future generations of commercial memory, the 64-bit input/output data bus is not unrealistic. The 256Mb DRAMs are in the experimental phase at the moment of this writing, and most of these chips use an 8- or 16-bit wide bus. There is at least one DRAM manufacturer (Samsung) mentioning an optional x64 organization of their future commercial 256 Mb DRAM [EET813], beside the x8 and x16.

For this project, there are good reasons to decide on a smaller data bus. Typical choices are 8 or 16 bit, with 8 bit being the most common data bus width in present day SRAMs. For DRAMs, the most popular data bus width now is the x4, with the x8 organization commanding a 5-10% price premium. 16-bit data bus organizations are also available for 16 Mb DRAMs, at 10-15% price premiums over x4. We adopt an 8 bit wide data bus, for the reasons discussed above, and also to reduce the risk involved in manually laying out a wider data bus (there is a fair amount of repetitive, hence error-prone, layout work involved in wide busses).

With a 64 bit wide memory interface and an 8 bit wide I/O data bus, a combinational circuit has to be interposed between the two, to route 8 bit words to and from the eight 8 bit memory channels. This can be seen as an extra level of column decoding, since it selects from 64 "columns" of sense amplifiers (although there is only one row), as shown in Figure 4.4. We label this logic level "I/O Data Decoder" or "I/O Decoder".

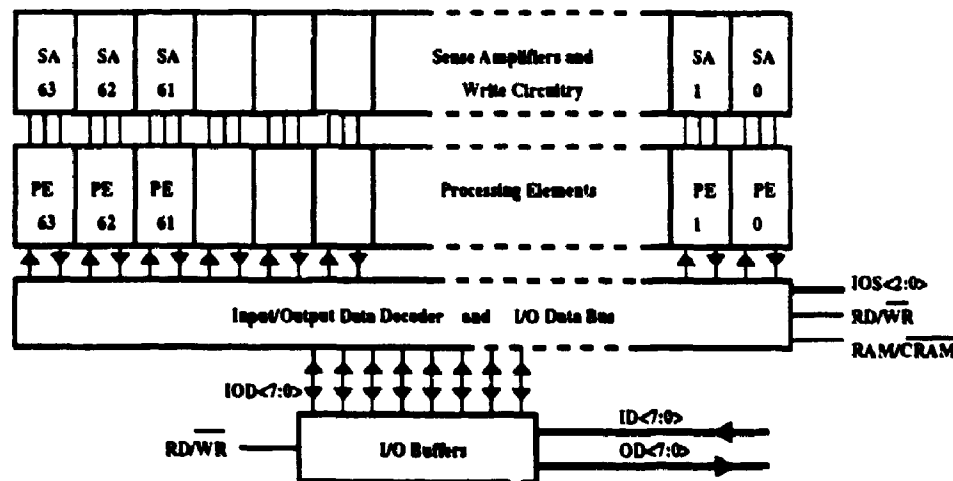


Figure 4.4 64 PE C*RAM Input/Output organization

There is one important restriction imposed on this extra level of column decoding, when compared to conventional memory column decoding. In the latter, there is no restriction on the mapping of the selected columns to the I/O data lines. This could result in a multi-bit I/O word being scattered in non-adjacent physical columns when written into the memory. Whether the word is scattered (even in multiple memory chips within a memory subsystem) or not is of no consequence in a conventional system, since the opposite operation is performed when the word is read from memory: the bits are recomposed on the system data bus. By contrast, in a C*RAM system we are interested in keeping a data word in adjacent logical columns, since this arrangement corresponds to the local shift left/right communication between adjacent PEs. The adjacency condition is an absolute must for the bit-parallel oriented C*RAM (discussed in Chapter 5), which uses groups of adjacent PEs to process multi-bit words.

Directly mapping the above description, one way to implement the I/O Data Decoder would be by grouping eight 8-to-1 bidirectional multiplexers, Figure 4.5. In fact, the adjacency condition discussed above makes this implementation very expensive in terms of routing area, if the multiplexers are of logic gate type. For example, the leftmost 8-to-1 multiplexor, corresponding to bit 7 (MSB) in the I/O data word, has to select between columns numbered (counting from right to left) 63, 55, 47, 39, 31, 23, 15, 7 as shown in Figure 4.5. The next multiplexor, corresponding to bit 6, selects between columns numbered 62, 54, 46, 38, 30, 22, 14, 6, and so on. This structure has a large number of cross-connections, hence it requires a large wiring area.

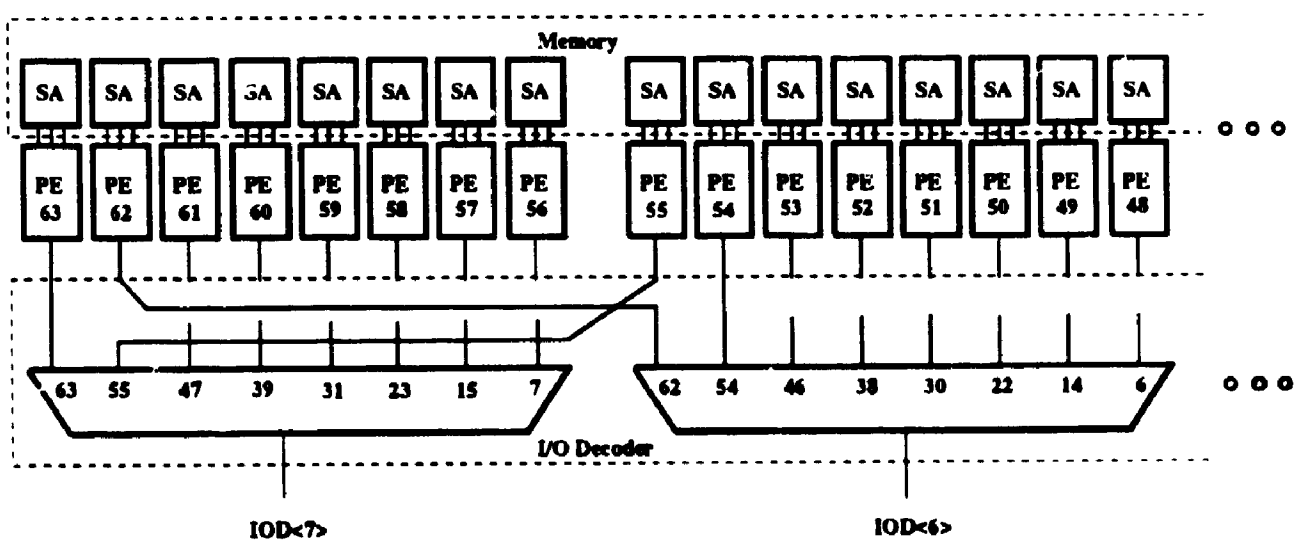


Figure 4.5 Logic function of I/O Data Decoder (2 out of 8 multiplexors shown)

A much more effective structure is the usual “bus and decode” arrangement of a column decoder, presented in Figure 4.6. The eight multiplexors are now distributed over the length of the array, by connecting the outputs of interest to a common line via tri-state drivers. By appropriately connecting the three selection lines to the decoding gates we can route a byte on the I/O data bus in adjacent logical columns (one logical column corresponds to one sense amp/write circuit).

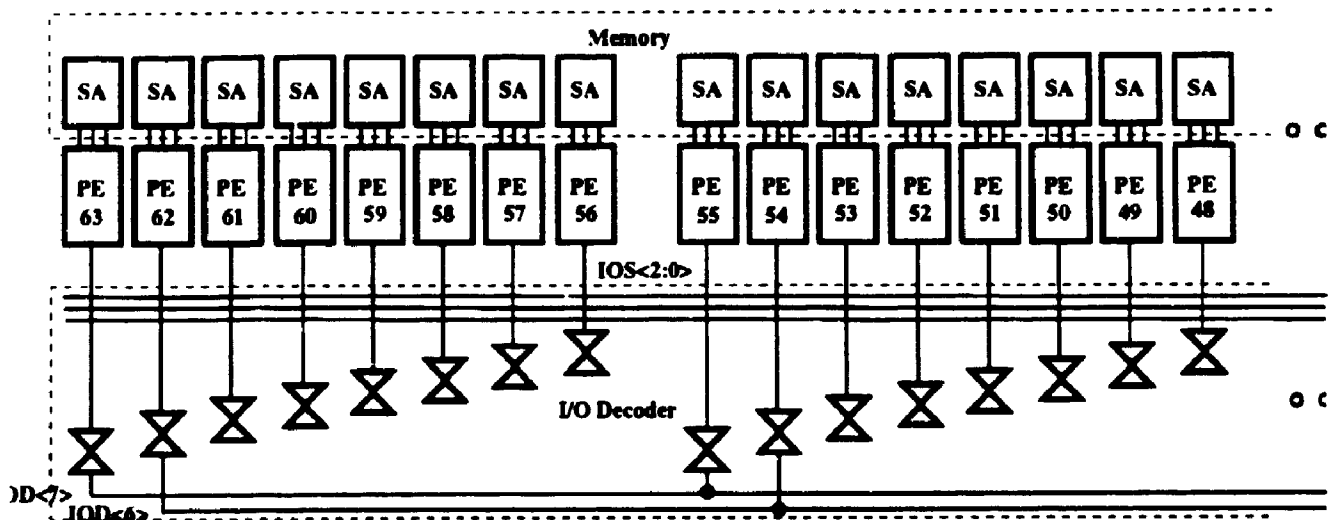


Figure 4.6 Practical arrangement of the I/O Decoder

Three new selection lines are necessary to select an 8-bit word from the 64 bits available. These lines are labeled $IOS\langle 2:0 \rangle$, for Input/Output Select. The actual decoding circuitry is presented in Figure 4.7. Apparently, one 3-input AND gate would be enough to decode one 8-bit word, since the 8 adjacent bits are selected together. In fact, this is an area expensive solution, because of the connections from this single gate to the eight tri-state drivers. These connections would extend horizontally to the eight destinations. They would have to be routed in TOPMETAL, since METAL1 (closest to silicon) is taken with local interconnections and METAL2 routes vertical links. In fact, as the final layout will show, the TOPMETAL layer has to accommodate the eight I/O data lines, the three IOS lines and two more control signals (explained later). Adding more TOPMETAL lines will extend the interconnect only area.

The more compact alternative is to replicate the 3-input decoding AND gate for every PE, using area beneath the TOPMETAL connections already in place. The final decoding structure is shown in Figure 4.7: one NAND gate is shared between two IOD section, with one inverter for every section (bottom row of inverters in the figure). The inverter output is the actual selection signal, termed $WSx\langle i \rangle$, for Word Select (e.g. $WS7\langle 5 \rangle$ selects bit 5 of the seventh word: all eight $WS7\langle i \rangle$ signals have the same logic value).

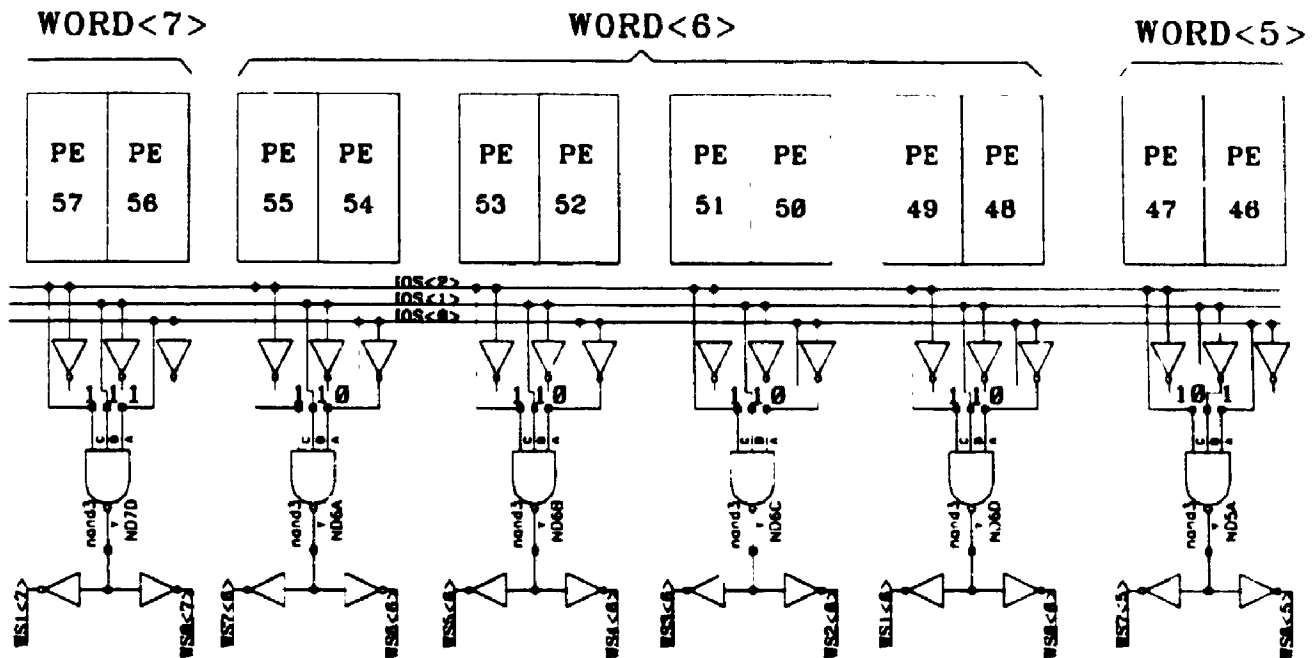


Figure 4.7 Decoding section in the I/O Decoder

As suggested in Figure 4.7, the three inverters that generate the IOS selection lines complements are also replicated for every selection cell. In designing the decoder layout, the various decoding signals are obtained by overlaying eight different connection cells, with eight patterns of GATE polysilicon and METAL1. These patterns decide whether an IOS line (a GATE wire) or its locally generated complement (a METAL1 wire) is connected to the input of the NAND gate. By keeping only the simple interconnect patterns in different layout cells, the correctness of the decoder can be quickly verified through visual inspection.

4.2.1 Output Data Circuitry

The circuit that controls the connections from memory outputs to the I/O data bus is shown in Figure 4.8. The figure shows connections from the two most significant memory word bits (63 and 62) to I/O lines 7 and 6, respectively. The tri-state driver function is implemented by adding transmission gates in series with the memory output driver (the MQ<i></i> lines). The enable control signal is activated when there is a memory access (RAM/CRAMB=HIGH), the access is a read cycle (RD/WRB=HIGH) and only for 8 bits out of the 64 available, as indicated by WS<k> (Word Select) from the decoding gate. RD/WRB is the global read/write signal required by the SRAM control interface (the SRAM module uses the complemented signal). RAM/CRAMB is a new control signal that identifies RAM operation when HIGH and C*RAM operation when LOW. All the elements shown in Figure 4.8 are laid out in the pitch of two PEs, and do not pose special layout problems.

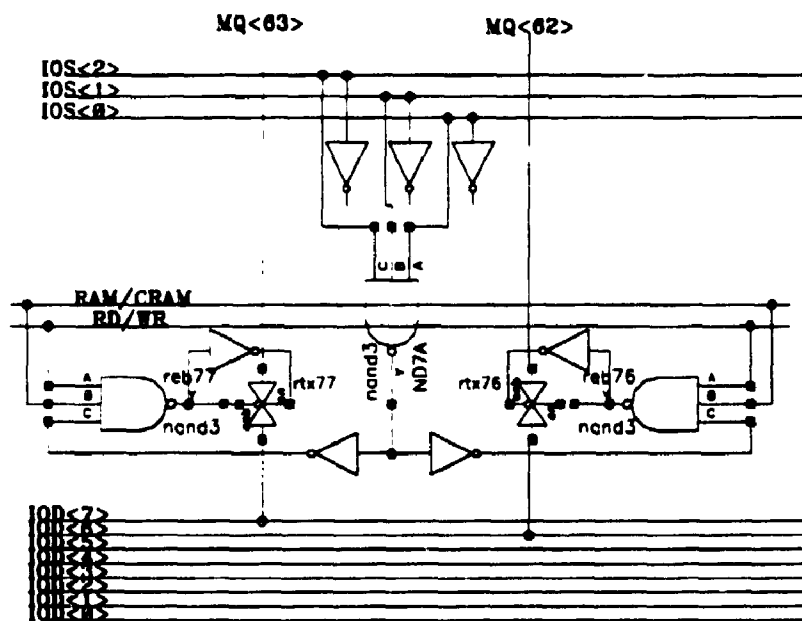


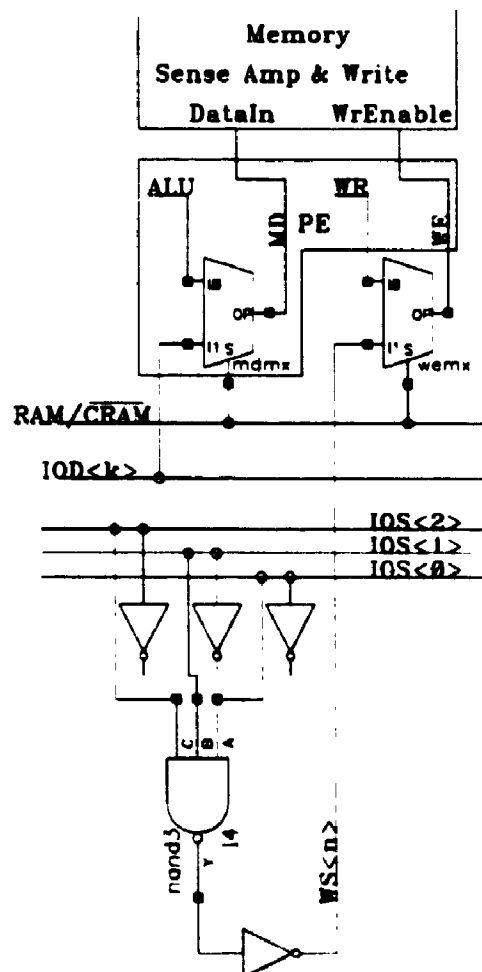
Figure 4.8 Data output circuitry in the I/O Decoder

4.2.2 Input Data Circuitry

Figure 4.9 shows a segment of the circuitry connecting the bidirectional I/O data bus with the memory data inputs. The data routing element is the *mdmx* multiplexor, controlled by the RAM/CRAMB signal. During memory access, RAM/CRAMB=HIGH and the byte on the I/O bus is routed to all 8 input byte channels. The write operation is only validated for one channel, by controlling the appropriate local Write Enable inputs of the memory. These lines are sourced by the *wemx* multiplexors, controlled by RAM/CRAMB. During memory access (RAM=HIGH), the *wemx* multiplexor selects the Word Select (WS) signal as memory write enable.

During C*RAM Operate-Write cycles (RAM/CRAMB=LOW), the memory data input comes from the ALU line and the local write enable comes from the PE W register output (WR in the figure). The enclosure labeled PE (Figure 4.9) shows that the input data multiplexor *mdmx* was laid out inside the PE rectangular area (see the PE floor-plan in Figure 3.34).

Figure 4.9
Data input circuit
in the I/O Decoder



The write enable multiplexor is laid out beneath the $\text{IOS}\langle 2:0 \rangle$ TOPMETAL lines, but is included in the $40 \times 84 \mu\text{m}^2$ PE area cited at the end of the previous chapter. Although multiplexors *mdmx* and *wemx* are discussed here for reasons of presentation completeness, they are considered part of the BATMOS C*RAM PE. In the hypothetical RAM-PE codesign mentioned at the beginning of this chapter, the column decoder would be designed to account for the PE/Input-Output multiplexing.

4.2.3 Input/Output Buffers

This level of buffering is interposed between the bidirectional I/O bus running along the PE array as part of the I/O decoder and the C*RAM input/output pins. Figure 4.10 shows the buffer group, physically situated at the middle of the I/O decoder (Figure 4.4). The eight output buffers drive the long lines to the I/O pins. They also reshape the memory output signals deteriorated by the passage through the I/O Decoder transmission gates and the large capacitive loads.

The input buffers are tri-stateable, because they drive the same I/O lines driven by the memory output drivers through transmission gates. The input buffers are active during a memory write cycle (RAM/CRAMB-HIGH and RD/WRB=LOW). The WR signal is the complement of RD/WRB. As shown in the figure, the input buffers receive the 8-bit $\text{DIN}\langle 7:0 \rangle$ bus from the I/O pads and the output buffers generate a separate 8-bit $\text{DOUT}\langle 7:0 \rangle$ output bus.

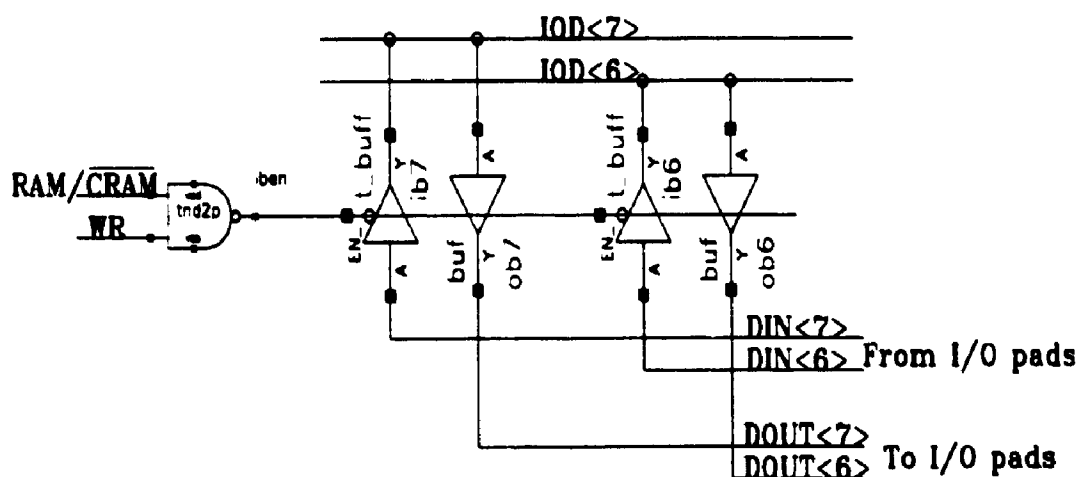


Figure 4.10 Input/Output buffers (for bits 7 and 6)

4.3 C*RAM Instruction Multiplexing

This section presents the C*RAM block that feeds the operation code (opcode) to the PE array. The opcode of the 64 PE C*RAMs consists of 14 bits: the 8 bit Truth Table Opcode (TTOP) and the 6 bit Control Opcode (COP), presented in detail in Chapter 3. For future generation of C*RAMs, the opcode length is expected to increase, as more functionality can be added in a similar area due to shrinking process features.

There are four possibilities to bring the opcode word inside the C*RAM: 1. using dedicated package pins; 2. multiplexing on the address bus; 3. multiplexing on the data bus and 4. a combination of the last two options. The alternative of providing separate opcode pins is rejected because it increases the package pin count, hence the size and price of C*RAM. Multiplexing the opcode on the data bus is shown below to be the preferred solution.

Examining a C*RAM Read-Operate-Write cycle (waveforms presented at the end of the chapter), we observe there are two events on the address bus (the row read and write addresses) and none on the data bus (assuming the opcode comes on a separate bus). Multiplexing the opcode on the address bus would add a third event in between the previous two, while still leaving the data bus unoccupied. The data bus is hence the better candidate for transmitting the opcode, because it allows a more relaxed system timing. The previous discussion applies to a fast SRAM based C*RAM, as is the case of BATMOS C*RAMs. In a hypothetical DRAM C*RAM, a memory write or read cycle requires two events on the address bus, the row address strobed by RAS and the column address strobed by CAS. The address bus is thus halved (compared to an equivalent sized SRAM, which does not use address multiplexing), so for example a 16 Mb DRAM organized as 16 Mword x 1b/word only needs 12 address lines. This address bus is thus insufficient to carry the 14 opcode bits of the 64 PE C*RAM, so opcode multiplexing would be again required. There would be now six address events in a DRAM C*RAM Read-Operate-Write cycle (assuming the read and write address are in different rows, also called pages in DRAM jargon). Even if the RAS and CAS events have non-critical timing, due to the long row access time (30 ns for example), six address events and the corresponding strobes are very demanding on the C*RAM controller, and would become a real problem as the operating speed increases. The data bus appears again as the preferred solution for multiplexing the C*RAM opcode.

On the other side, we have already decided on a byte-wide data bus, for reasons related to supply lines bouncing and layout compactness. We are thus constrained to use both the data and address buses to transport the opcode. In a C*RAM operation cycle, the TTOP is used first to evaluate an ALU function, then the COP dictates the destination of the result. Hence, the 8-bit TTOP has to be provided first and therefore is assigned to the data bus, which does not have any other assignment. The 6-bit COP is assigned to a section of the address bus, multiplexed with the read/write row addresses. We arbitrarily choose the six least significant address lines to carry the COP bits.

Because the C*RAM address and data buses are now multiplexed buses (address and COP, data and TTOP respectively), the opcode has to be sampled into a dedicated internal register. In normal memory cycles, the RAM address and data are sampled into memory registers by the memory clock MCK (see Figure 4.1). In the C*RAM opcode register, there are two register sections for TTOP and COP, each loaded by its own clock: TTOPLD and COPLD. These two sampling signals have to be provided by the C*RAM timing block, or supplied directly from off-chip.

Figure 4.11 shows the opcode support circuit (TTOP only). The opcode information stored in register is applied to the PE array under the control of three other timing signals. One AND gate for each opcode signal is used to enable and drive the corresponding global PE array line. The signal MTTOPCK (Main TTOPCK) marks the ALU evaluation period, generating the TTOPCK<*i*> signals, according to the TTOP register contents. Similarly, the signal WCK is active during the write register phase. The third timing signal, BTCK, is dedicated to the Bus-tie control line.

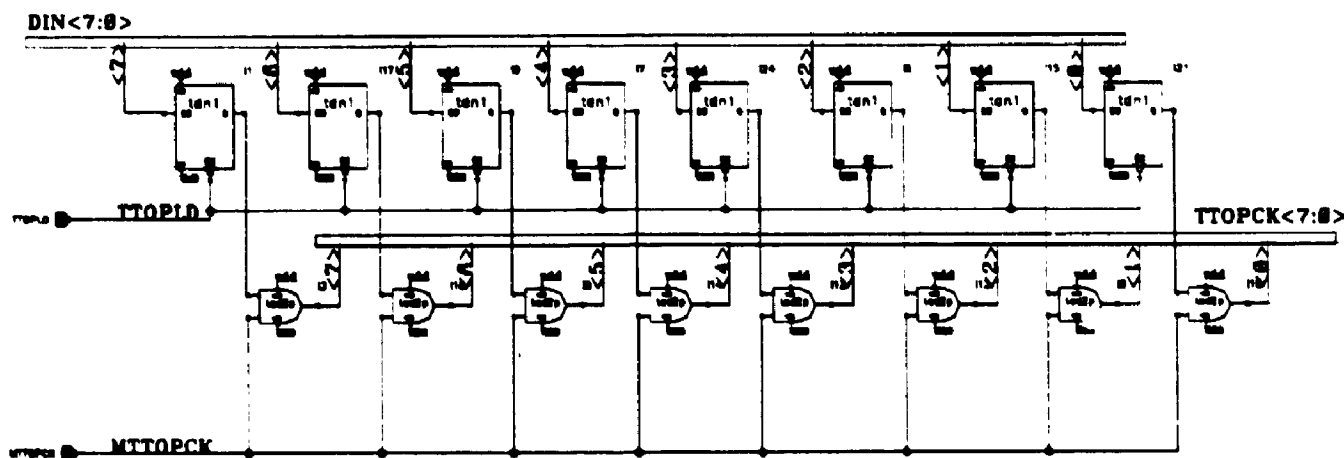


Figure 4.11 TTOP register and associated enable/driver AND gates

4.4 Timing Generation

This section presents the two timing schemes provided for the 64 PE BATMOS C*RAMs: external timing and internal timing based on external strobe and clock. The external timing was adopted to allow for complete independent control on the timing. The internal timing circuit relies on two external signals to generate the timing clocks. The two resulting sets of timing signals are multiplexed under the selection of the External Control (XCTRL) pin.

Summarizing information from the previous sections, the timing signals to be provided are (given in parentheses are older names that appear in layout or initial schematic revisions):

1. ALUPRCKB (ALUPREB): the active LOW ALU precharge signal.
2. TTOPCK: the active HIGH ALU evaluation clock.
3. COPCK (REGCK): the active HIGH write register clock.
4. BTENCK: the active HIGH bus-tie enable clock.
5. TTOPLD (OPLOAD): the truth table opcode load clock; edge-active.
6. COPLD (FLOAD): the control opcode load clock; edge-active.

4.4.1 External Timing

The simplest way of controlling the timing for C*RAM is to dedicate separate chip pins for each signal needed and provide these signals from off-chip. This method uses minimal hardware inside the C*RAM (input pads/buffers and metal connections) hence it minimizes the probability that a silicon defect localized outside the PE array will cause a non-functional C*RAM. On the other side, while the method is justified for a small series of prototype C*RAMs, it is undesirable for a commercial C*RAM since it requires a bigger package for the extra timing pins.

As discussed in Chapter 3, there are a number of restrictions with which these signals have to comply: ALUPRCKB, TTOPCK and COPCK have to be mutually non-overlapping; BTENCK can not overlap ALUPRCKB. A typical timing diagram is shown in Figure 4.12. The TTOPLD and COPLD signals are sampling clocks for the two opcode sections, hence require appropriate setup and hold times of data on the sampled busses.

When the timing signals are coming from off-chip, the external controller has to insure the required phase relationships. If the constraints are not met, the result is either excessive power consumption or/and logic error. The former appears when the precharge clock is overlapped with the evaluation or bus-tie clocks, resulting in increased supply cross-bar current. Logic errors are caused by overlapping the ALU evaluation phase with the Register Write phase, since the registers are always connected to the ALU selection inputs. If the two phases overlap, the ALU-Register loop closes and the initial ALU result can be corrupted, as discussed in detail in Chapter 3.

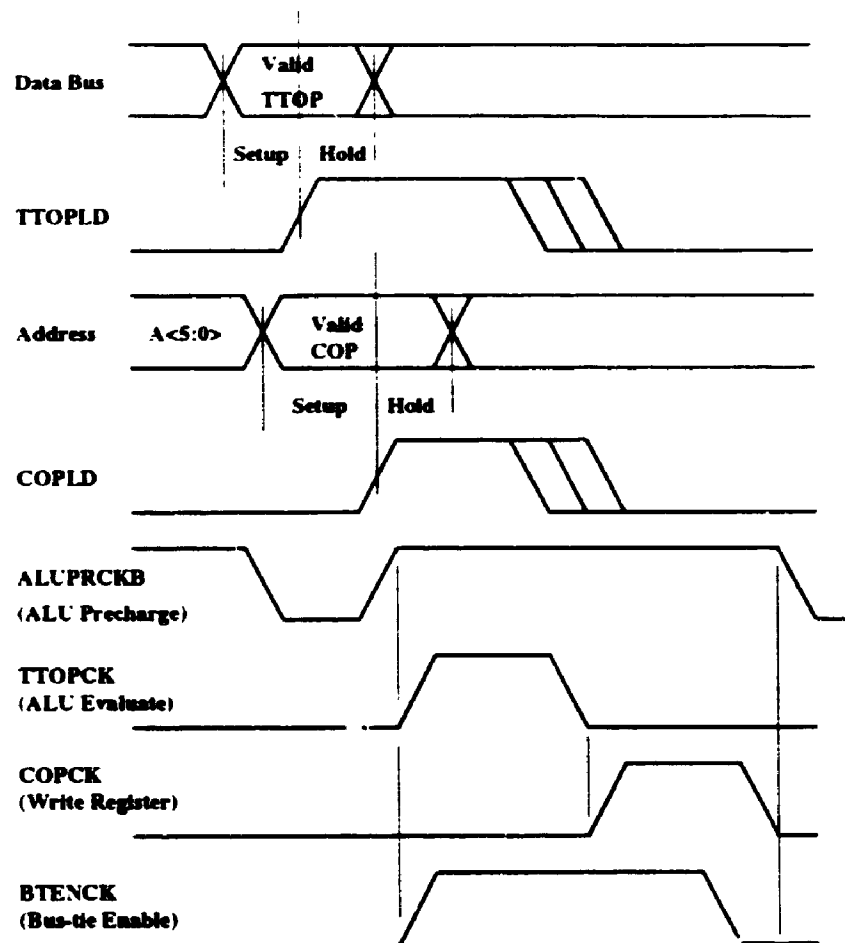


Figure 4.12 Example of C*RAM specific timing signals

4.4.2 Internal Timing Circuitry

This circuit was designed in order to reduce the number of timing signals brought from off-chip, as an intermediate stage between prototype and industrial C*RAM. There are obvious advantages

to the internal timing option: a smaller C*RAM package: faster operating speed, and a simpler C*RAM controller. The only disadvantage is a small increase in the C*RAM peripheral circuitry area, but this is far outweighed by advantages. The electrical schematic is shown in Figure 4.15 (the prefix TG stands for Timing Generator). There are three off-chip inputs: OPS (Operate Strobe), CCK (C*RAM Clock) and RESET. The RESET signal is used only after power-up, to bring the state machine based on D flip-flops to the initial state. OPS and CCK are the actual timing signals. There are two type of cycles the circuit can generate, a standard ALU cycle and a Bus-tie cycle. A fourth input, BTCYC, which is one of the bits in the Control Opcode register decides which cycle will be executed.

The rising edge of OPS serves to sample the truth table opcode (TTOP) on the 8-bit data bus. The same edge marks the beginning of both types of cycles. After OPS going HIGH, all other transitions in the circuit are dictated by CCK, starting with the first CCK rising edge. The circuit was designed to accept a free-running CCK, with OPS signalling the beginning of a cycle. Both rising and falling CCK edges are used to clock flip-flops in the circuit. This results in a higher operating frequency compared to the case where only one type of CCK edge is active. CCK is supplied from off-chip, hence it is speed-limited by the printed circuit board track capacitance and chip I/O buffers. With CMOS I/O buffers and standard PCB tracks (without controlled impedance), CCK is limited to 50-60 MHz. Using both CCK edges (rising and falling) practically doubles the timing generator clocking frequency.

The circuit is not a synchronous one: transitions are caused as a result of CCK edges but CCK is not the common clock for all flip-flops. Instead, a scheme whereby the output of a flip-flop is used to clock the next flip-flop in the chain is adopted. The main result of this interlocking scheme is that different clock pulses generated will be non-overlapped, as required by the constraints of dynamic circuitry.

A standard ALU cycle is shown in Figure 4.13 and a Bus-tie cycle appears in Figure 4.14. In both cycles, CCK transitions happen 5 ns apart, equivalent to 100 MHz CCK operation. While the circuit has been simulated at higher frequencies, 100 MHz is already difficult (if not impossible) to sustain in a C*RAM system built on a standard printed circuit board.

As can be seen in both cycles, the circuit forces the PE array to be in precharge when inactive (i.e. outside an active cycle). A RESET HIGH pulse is first applied to initialize the circuit. This signal can be derived from the system power-on reset. Then, the OPS rising edge starts the cycle by enabling the circuit to register CCK transitions. The OPS rising edge also samples the TTOP from the data bus. The first CCK rising edge after the OPS transition has two effects: it generates a COPLD rising edge that samples the Control Opcode from the address bus, and inactivates the precharge signal, bringing ALUPREB HIGH. The transition on ALUPREB in turn activates the ALU opcode evaluation, MTTOPCK=HIGH (Main TTOPCK), ensuring thus no overlap between the two. The next event on CCK, a falling edge, causes MTTOPCK to go LOW terminating the ALU evaluation. The transition on MTTOPCK (rather than CCK directly) activates the write register phase, MCOPCK=HIGH, again ensuring no overlap. The end of the write register pulse is determined by the third event on CCK, a rising edge. This transition causes MCOPCK to go LOW, which in turn ends the cycle by bringing ALUPREB=LOW, i.e. back to precharge.

The Bus-tie cycle is similar. The difference is that the cycle is one CCK period longer, in order to allow for worst-case bus-tie propagation. A shortcoming of the design is that it generates the write register pulse MCOPCK at the same time as in the standard cycle, as opposed to the end of the bus-tie enable signal BTENCK.

The Timing Generator layout was created by manually placing and routing standard cells from the BATMOS TCELL library. The cells are placed in two horizontal rows, with two *vss* strips on top and bottom and a common central *vdd* strip. Interconnections are routed in the two top metal layers, METAL2 (vertical) and TOPMETAL (horizontal). The final layout is fairly large because of the bulky D flip-flops in the TCELL library. A custom layout could reduce the size to an estimated 25% but would require considerable layout and verification effort.

4.4.3 Timing Generation in the 512 PE C*RAM

The C512mp512 C*RAM uses only one external timing signal, CCK. In contrast to the 64 PE C*RAMs, the PE array is not normally in precharge. The PE cycle starts on the rising edge of CCK. An internal one-shot (monostable) circuit uses this edge to generate a precharge pulse. The ALU evaluation lasts from the end of precharge to the falling edge of CCK. The CCK LOW pulse

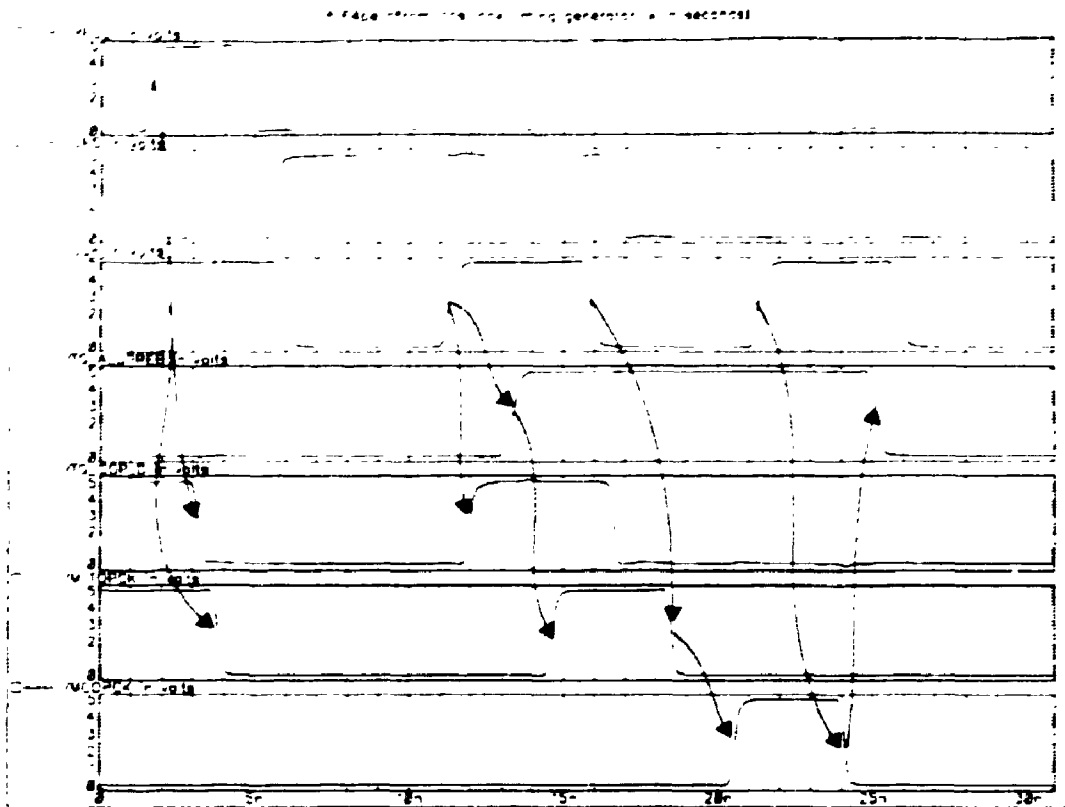


Figure 4.13 Timing Generator PE standard cycle

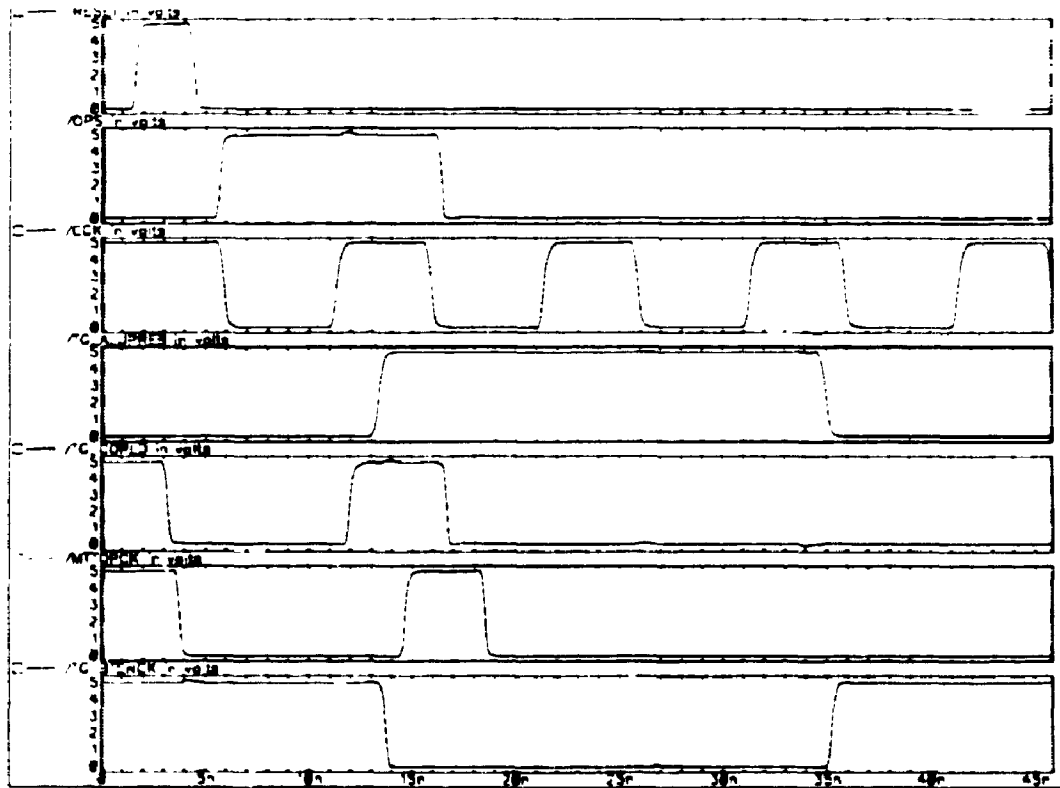


Figure 4.14 Timing Generator Bus-tie cycle

is the Write Register phase, ended by the beginning of the following cycle. The one-shot circuit delay is given by a chain of gates. The initial version used the ALUPRCKB line to self-time the one-shot circuit, but simulations showed that the resulting pulse was too short for effective pre-charge. As an alternative to the monostable timing circuit, the OPS/CCK timing generator (Figure 4.15) can be selected when the external pin ST (Self-Timed) is set LOW

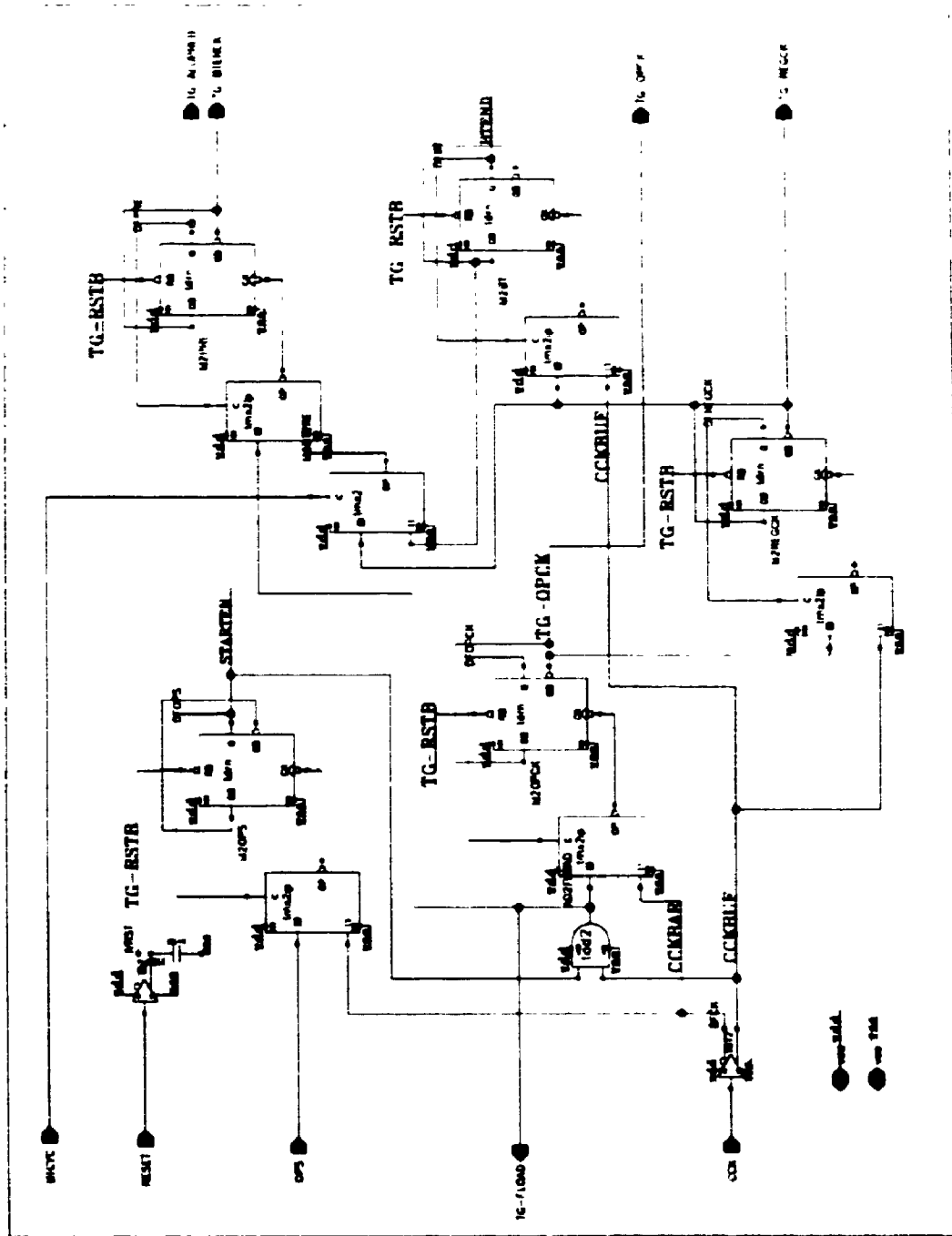


Figure 4.15 Timing Generator electrical schematic

4.5 64 PE C*RAM Cycles

To summarize functional and timing information described in this chapter, the following is a synopsis of the C*RAM cycle types. For system design purposes, all setup times are 5 ns, hold times are 0 ns, access time is better than 20 ns for read cycles and better than 10 ns for page-mode read. CCK period is 20 ns. The exact values will be measured when the high-speed test set-up becomes available (Chapter 6).

4.5.1 Memory Write Cycle

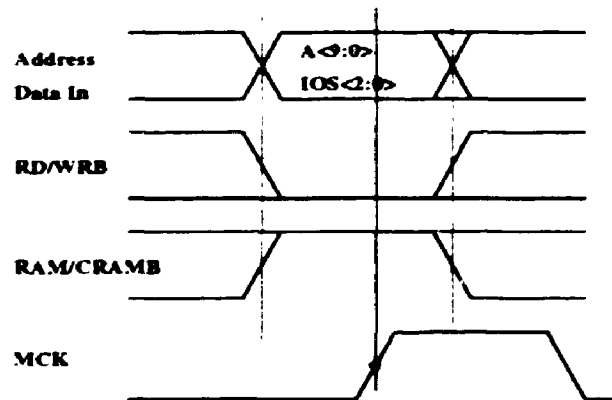


Figure 4.16 C*RAM Write cycle

4.5.2 Memory Read and Page-mode Read Cycles

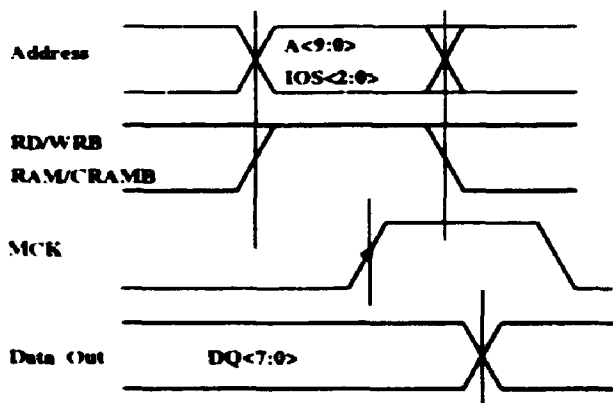
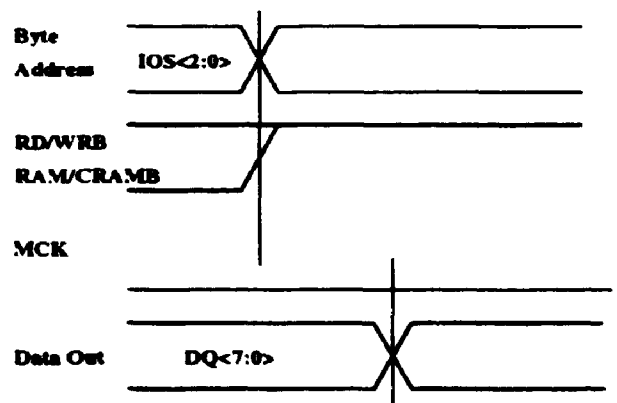


Fig.4.17(a) C*RAM Read cycle



(b) C*RAM Page-mode Read cycle

4.5.3 C*RAM Operate Cycle

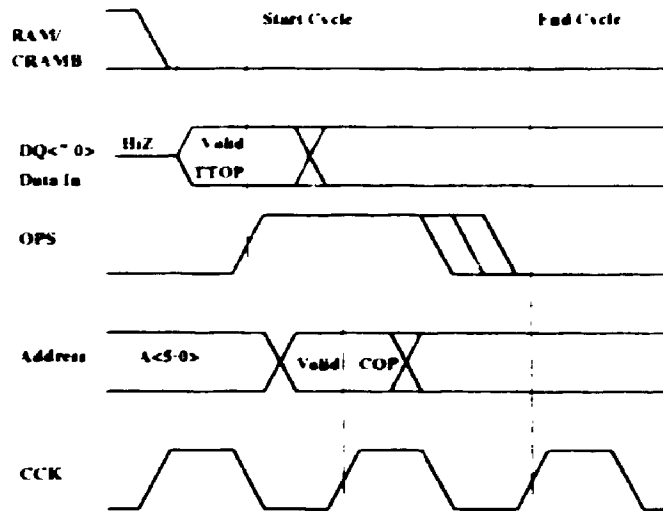


Figure 4.18 C*RAM Operate cycle

4.5.4 Read-Operate Cycle

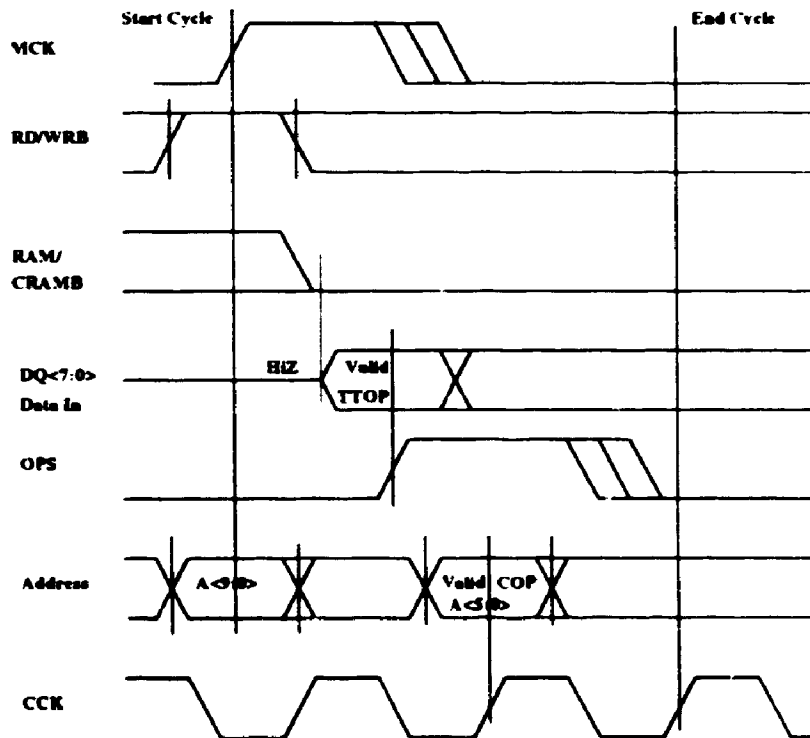


Figure 4.19 C*RAM Read-Operate cycle

4.5.5 Operate-Write Cycle

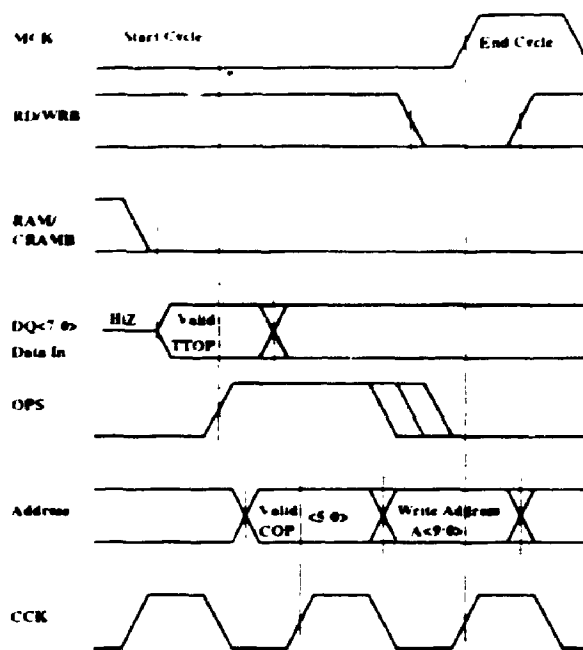


Figure 4.20 C*RAM Operate-Write cycle

4.5.6 Read-Operate-Write Cycle

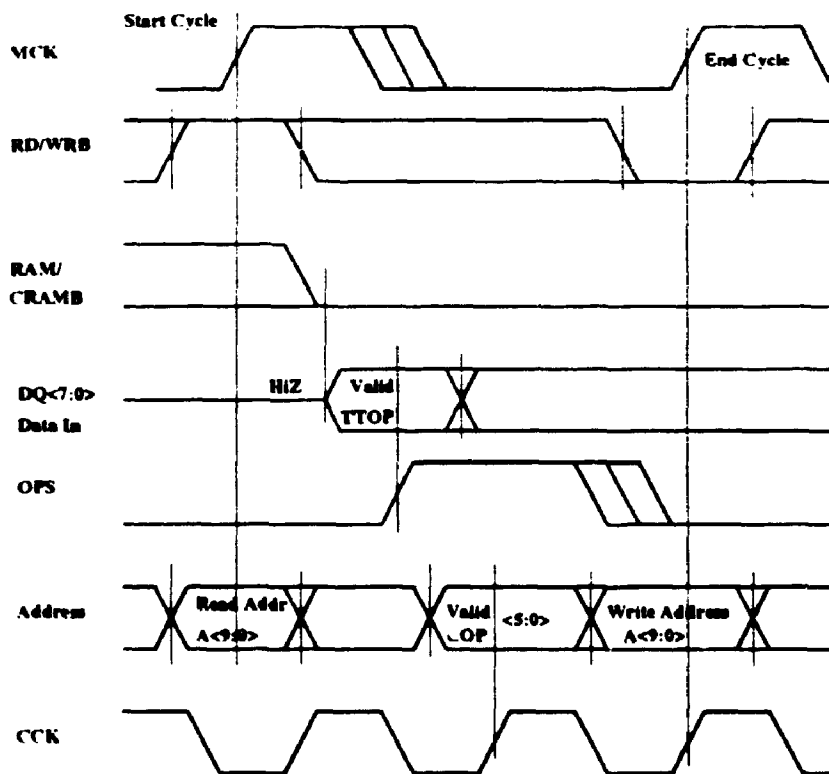


Figure 4.21 C*RAM Read-Operate-Write cycle

4.6 Synopsis of BATMOS Computational RAMs

Table 4.2 presents a summary of BATMOS C*RAMs characteristics.

Table 4.2 Synopsis of BATMOS C*RAM chips

	C64p1k	Cs64p512	C512mp512
Number of PEs	64	64	512
Local PE memory (bit)	1024	512	480
Total memory (bit/byte)	64kb/8kB	32kb/4kB	240kb/30kB
RAM organization (wordlength)	x8	x8	x8
Serial access	no	yes	no
PE length (μm)	84	96	184
PE transistor count	76	84	147
PE array/memory ratio(%)	2.7	5.5	10
PE: Segmented Bus-tie	no	yes	yes
PE: Boundaries (S&B regs.)	no	no	yes
PE: Ripple-carry	no	no	yes
PE: Alternate registers	no	no	yes
C*RAM timing source	TG-2pin/Extern	TG-2pin	ST-1pin/TG-2pin
Pin count (Actual package)	44 (84)	44 (68)	44/68
Silicon source	CMC grant	BNR internal	CMC grant
Date layout completed	Oct. 1993	Nov. 1993	Aug. 1994
Date chip received	May 1994	Apr. 1994	Feb. 95 (est)
Chip highlight	First func. BATMOS C*RAM	Serial I/O	Bit-parallel Multiply (See Ch.5)

Figure 4.22 shows the block floor-plan of the 64 PE C*RAMs. Complete electrical schematics of the C64p1k chip are shown in Appendix A, and the chip pinout information is presented in Appendix B.

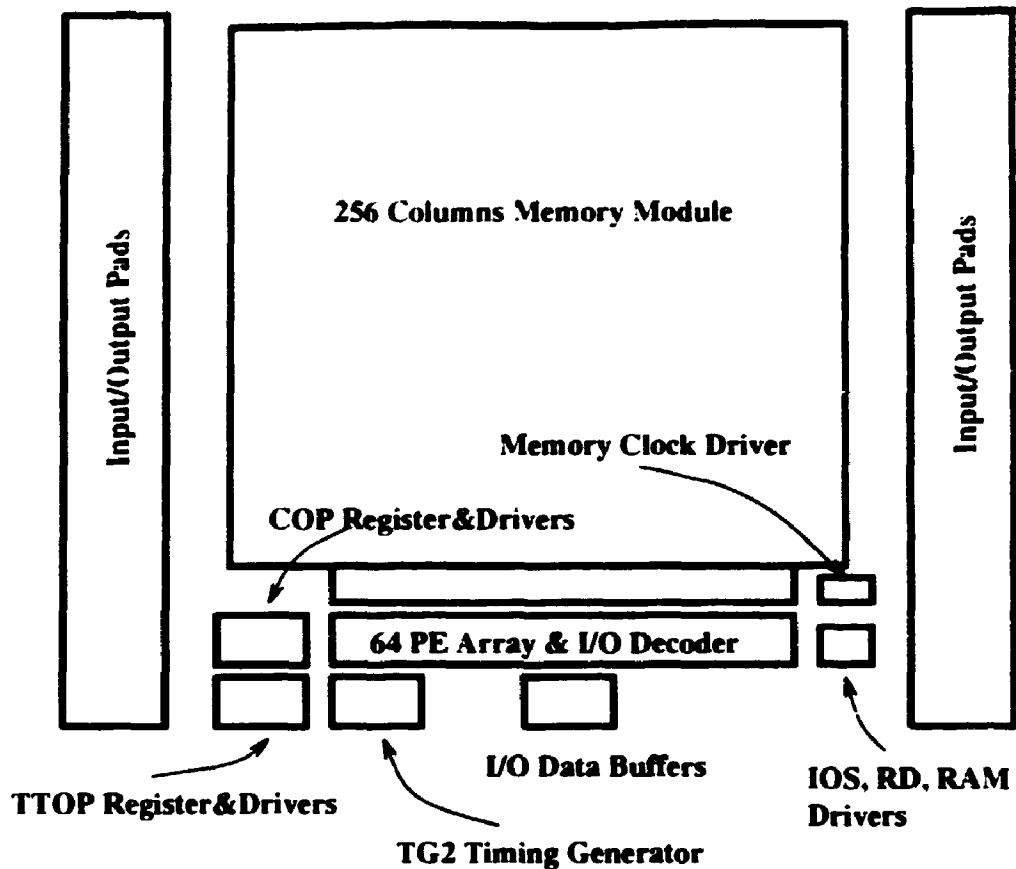


Figure 4.22 64 PE Computational RAM floor-plan

Chapter 5

The Bit-Parallel Oriented Processing Element

5.0 Motivation

This section will present additional functionality added to the Baseline PE over the course of this work. Two different reasons stand behind these developments. First, the Baseline PE [Elli92] conforms to an already fairly classical view of the massively parallel single-bit processor SIMD machine. As discussed in [Tani91], this type of machine is considered to exhibit a weak arithmetic performance in floating point intensive applications, because of the bit-serial nature of multibit arithmetic. Floating point multiplies are especially time demanding on bit-serial machines. Even the simpler case of integer multiply presents the problem of multiple redundant memory cycles to retrieve the multiplicand and store the partial product of every partial addition. There is therefore a need to go beyond the single-bit PE. Some of the SIMD machines presented in literature address this problem by using multibit PEs. For example, the MP1 from MasPar uses 4 bit PEs, which, as noted in [Tani91] still require serialization for larger words. The IMAP discussed in the first chapter uses rather conventional 8 bit PEs, since it is targeted for 8-bit pixels image processing. The often-cited Connection Machine CM-2 from Thinking Machines Corp. added 32-bit floating point processors for every group of 32 single-bit PEs inherited from the previous generation.

Secondly, in the case of our particular implementation using ASIC SRAM as a C*RAM platform, heavily optimizing the PE layout area resulted in a 97%-3% partition between a fast sub-10ns

memory and slow processing elements. There is thus a double imbalance, first between the relative areas, then between the operating speeds. This leaves space for enhancing the PE functionality by using supplementary area, bringing the PE/RAM ratio to the initial estimate of 10-15%. It should be reminded here that the original estimate was aimed at DRAM-based C*RAM.

All the PE enhancements introduced here are geared towards bit-parallel arithmetic. The driving philosophy is to move to the circuit level those often used operations that require a sequence of instructions on the Baseline PE, being speed limited thus by off-chip constraints. The author considers that one novelty of these circuits consists in the word length flexibility they allow. This parameter is alterable dynamically (i.e. at any time during program execution) by writing certain PE registers. Thus, one of the main advantages claimed for bit-serial PEs, the possibility to adapt the word length to the data requirements, is also conserved for the bit-parallel computation. Also, the Extended PE includes all the functionality of the original PE, hence can still be used for bit-serial computation. C*RAMs employing the new PE structure are thus dual-mode C*RAMs: bit-serial and bit-parallel. The author believes that the C512mp512 C*RAM, which implements the Extended PE, is the first dual-mode SIMD building block with complete word length flexibility.

A first modification relates to interprocessor communication, transforming the global bus-tie connection into a programmable segments bus-tie, discussed in section 5.1. Then, circuitry that will adapt the shift operations to bit-parallel operation by defining word boundaries is introduced in section 5.2. This circuitry, together with an extended ALU presented in section 5.3, addresses the arithmetic weakness problem by providing a hardwired add/subtract ripple-carry chain. The final section of the chapter shows how all these PE additions/modifications together with an extended register set are used to perform bit-parallel integer multiplication, keeping all data and intermediary results within the PE array.

5.1 Programmable Segments Bus-tie (PSB) Network

The bus-tie network of the Baseline PE was discussed in section 3.4, and a functional schematic is shown in Figure 5.1. During the implementation work for this circuit, it appeared that bit-parallel oriented functionality can be added with a minimal amount of hardware. The new function is shown in Figure 5.2. Instead of a global bus-tie that connects all the PEs (more precisely, the pre-

charged ALU nodes) when activated, the segmented bus-tie allows groups of adjacent PEs to be connected in the same manner.

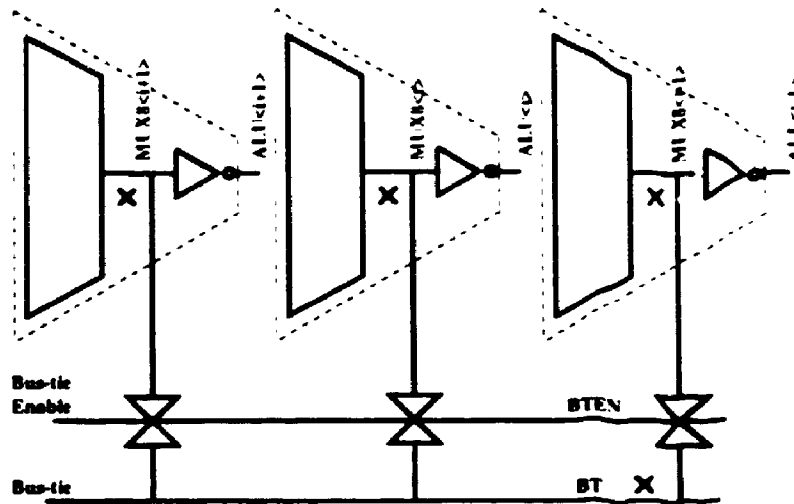


Figure 5.1 Bus-tie circuit in the Baseline PE (x denotes a precharged node)

A number of options are available when deciding how to partition the PEs into groups. The boundaries of the groups could be fixed in the simplest case, for example delimiting groups of 8 PEs. In this case though, a separate circuit should be maintained to implement the global bus-tie. Instead, the segmented bus-tie was designed with programmable segments. The control line of

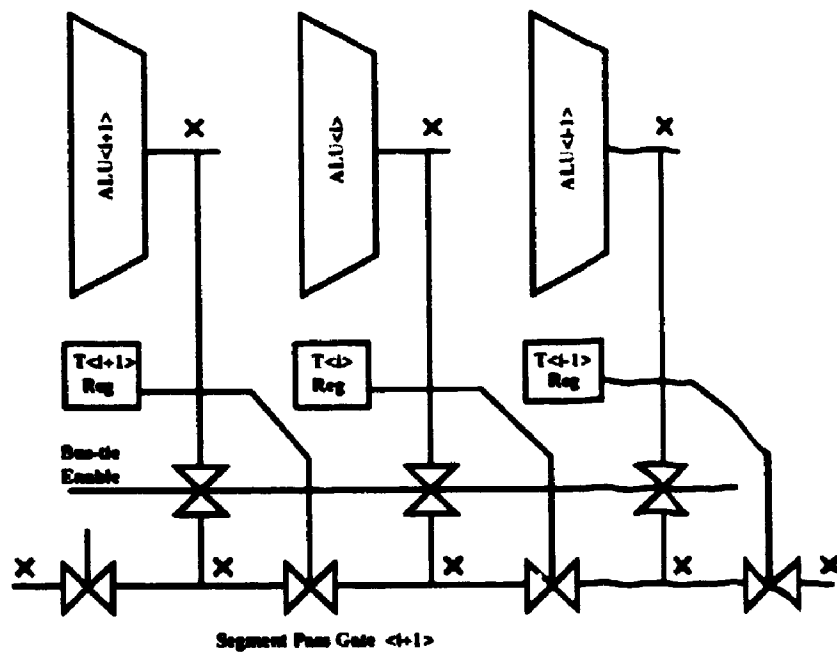


Figure 5.2 The Programmable Segments Bus-tie (PSB) Network

every transmission gate in the bus-tie chain in Figure 5.2 is connected to the output of a new register, the T (Tie) register. This register can be written by the ALU result line when a new WT (Write Tie Register) control line is activated, hence can be programmed dynamically. T register number $\langle i-1 \rangle$ is chosen to control the segment between PE $\langle i-1 \rangle$ and PE $\langle i \rangle$. When all the transmission gates are enabled, the circuit functions as the global bus-tie, although it is slower than the single line bus-tie (a quantitative comparison is presented later in the Simulation section).

5.1.1 Use of the Programmable Segments Bus-tie (PSB)

This section will show how the PSB circuit is a first step towards a C*RAM more oriented towards multi-bit operation. Similar to the global Wired-OR performed by the original bus-tie network, the PSB network executes a parallel Wired-OR for every PE group (segment).

We are working on 4-bit words, but in a bit-parallel manner, i.e. the four bits of a word are processed at the same time by four adjacent PEs, and all groups of four PEs do the same operation in parallel (obviously on different data). Now let us assume we want to propagate the rightmost bit (conventionally here the least significant bit, LSB) to all other positions, e.g. the initial configuration in say the X registers is xxx1 (where x stands for unknown value, 0 or 1), and the desired final configuration is 1111. This operation has to be replicated for all groups of four PEs, as in the example below (the convention pbbbb denotes parallel 4-bit bbbb words):

0110 1101 0001 1100 ...
or pb₁b₂b₃b₄

transforms into:

.... 0000 1111 1111 0000 ...
or pb₄b₄b₄b₄

This operation, called here "bit extension", could be used in a bit-parallel multiply sequence, as will be shown in section 5.4. Bit extension is very cumbersome to execute on the baseline PE C*RAM, since it involves repeated Shift and OR operations, as shown below:

0. Load X with the data to be operated on.

1. Execute an AND operation between X and the mask p0001. This will keep the bit of interest, the LSB, and mask out the rest (i.e. set them to "0"). The mask is assumed to be available in a "useful constants" area of the memory.

2. Clear Y (i.e. set all Y registers to 0), in preparation for next steps.

Start of loop:

3. Execute an OR between X and Y, with the result stored in Y. The Y register will hold the desired result at the end of the loop.

4. Shift-left X. The hardwired destination is X. The current LSB bit moves one position to the left, while its place is taken by a "0" (see step 1). This "0" will leave the corresponding Y bit unaffected in the subsequent X OR Y operations.

5. Repeat from step 3, until the initial LSB bit reaches the MSB position.

The bit extension execution time for n bit words, $t_{BE}(n)$, is:

$$t_{BE}(n) = t_{OP} + t_{OP} + 2(n-1) \cdot t_{OP} = 2t_{OP} + 2(n-1) \cdot t_{OP}$$

where t_{OP} is the ALU operation time.

The same operation is executed by the following sequence with the PSB circuit:

0. Load X with the memory row to be operated on.

1. Load the word boundary mask p1110 from memory and write it into T registers (this assumes that the boundary mask data has been previously defined, as part of program initialization. The mask can be created by replicating the byte 11101110 enough times to fill a C*RAM logic row). It is likely that the mask is already loaded, since the program might have already processed 4-bit words with the same boundaries.

2. Load Y with the mask p0001 (again, previously prepared in memory). This time, the mask is specific to our example operation.

3. AND X,Y, activate the bus-tie enable, and write the result back to X. The AND will clear the leftmost three bits to zero while leaving the LSB unmodified. With the result available at the ALU output, the bus-tie will generate the Wired-OR within the 4-bit group, then the result is

stored in the X registers (arbitrary choice, could be Y).

For speed benchmarking, we disregard the time required to prepare the bit-masks, since they are "non-recurring costs" for software. Hence, the execution time $t_{BE,PSB}(n)$ is:

$$t_{BE,PSB}(n) = 2t_{OP} + t_{PSB}(n) = 2t_{OP} + (n-1) \cdot t_{UPSB}$$

where

$t_{PSB}(n)$ = propagation time over the n-bit long bus-tie segment

t_{UPSB} = unit PSB propagation time from one PE to the next

The difference between the two execution times relative to the word-length n is that the first takes $2(n-1)$ ALU operation times, which are in the 10-20 ns range (or even slower if the C*RAM system cannot match the internal C*RAM speed), while the second takes n-1 unit PSB propagation times, which are in the 0.4-0.5 ns range (as shown in section 5.1.3). The PSB network has moved thus the bit propagation process to the circuit level from the software level.

5.1.2 Circuit Implementation

Figure 5.3 shows the transistor level implementation of the PSB circuit within one PE. The new elements in the actual bus-tie circuit are the two PMOS transistors PBTPRE and PBTLTCH. PBTPRE is the local precharge transistor for the BTLOC (Bus-tie Local) node. Since the Bus-tie metal line was replaced with a chain of pass gates, global precharge is replaced with local precharge for each bus-tie segment. The transmission gates between segments are in fact just NMOS transistors. This is possible because the dynamic bus-tie nodes are all precharged to logic High, after which the only value to be propagated by the PSB network is LOW, handled well by NMOS (no threshold voltage loss). In other words, there is no need for the PMOS in the full transmission gates since the PSB network either stays with all nodes HIGH, or propagates a LOW. The other new PMOS, the latch-back transistor PBTLTCH, is not absolutely necessary for circuit functioning. As in the case of the dynamic ALU (see section 3.1), it was added because the discharge

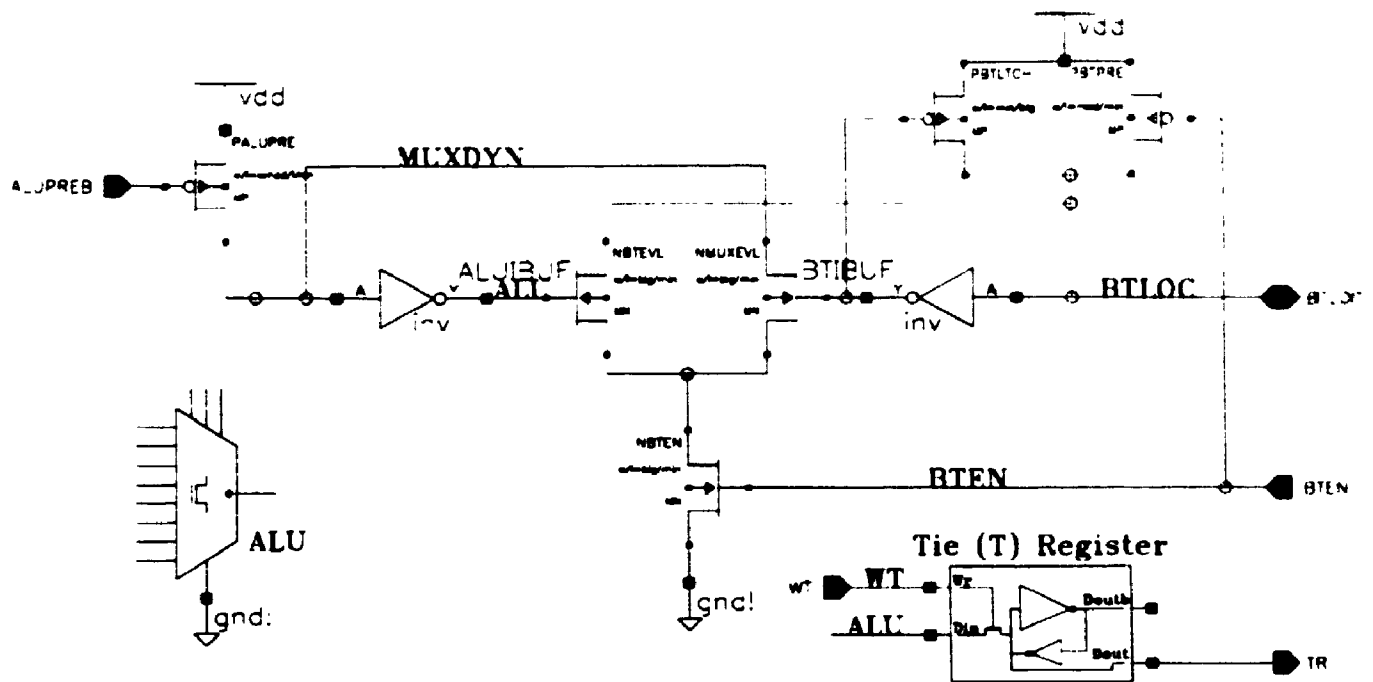


Figure 5.3 PSB circuit implementation within a PE

time of the dynamic BATMOS circuits cannot be accurately simulated and predicted. Nevertheless, while the ALU evaluation is fairly fast (under 4 ns), the PSB network in the worst case has to propagate a LOW from one end of the array to the other (see simulations below). Hence, it cannot rely on the parasitic capacitance to maintain all dynamic nodes to HIGH. PBTLTCH is thus added to insure the HIGH value is latched after precharge. The obvious disadvantage is a slower propagation speed, since the evaluation NMOS transistors have to fight the latch-back when discharging the precharged node.

Outside the actual bus-tie circuit, 5 more transistors are needed for the Tie latch, Figure 5.3. This is a single-access latch written by the ALU buffered result line. Its output is permanently connected to the gate of the NMOS pass transistor connecting the BTLOC nodes of adjacent PEs, as shown in Figure 5.4.

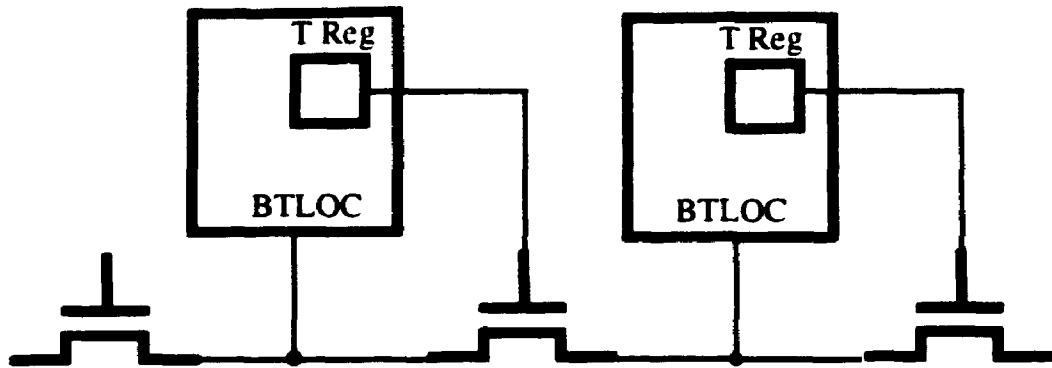


Figure 5.4 PSB Network Connections

5.1.3 Programmable Segments Bus-tie Simulation

Simulation waveforms of the Programmable Segments Bus-tie are shown in Figure 5.5. The circuit simulated is a chain of 64 PEs, with all connecting NMOS enabled, as shown in Figure 5.6. At one end of the chain, the PE ALU<0> dynamic node evaluates to LOW (i.e. ALU=HIGH, because of the inverting buffer). This value propagates to the other end of the chain (BT63/ALU). Figure 5.5 shows that the segmented bus-tie network propagates the source signal (BT0/ALU) to the other end of the array in 27.8 ns. The propagation time from one PE to the adjacent one is close to 0.45 ns. Hence, for 8-bit words, any PE can communicate its ALU value to the others in under 4 ns.

It was pointed out earlier that the PSB network is slower than the simpler single line bus-tie. A global wired-OR operation, beside affecting the results of all PEs, makes its result available on one of the C*RAM output pins (BTOUT). The C*RAM bank controller, responsible for issuing instructions, typically uses the BTOUT bit to conditionally branch to a different section in the program. If the branch on BTOUT instruction is often used, a sizable speed penalty is incurred due to the execution time difference between the 27.8 ns PSB global-OR and the 6 ns single-line global-OR.

In fact, the PSB time can be halved by connecting the two ends of the network to the inputs of an

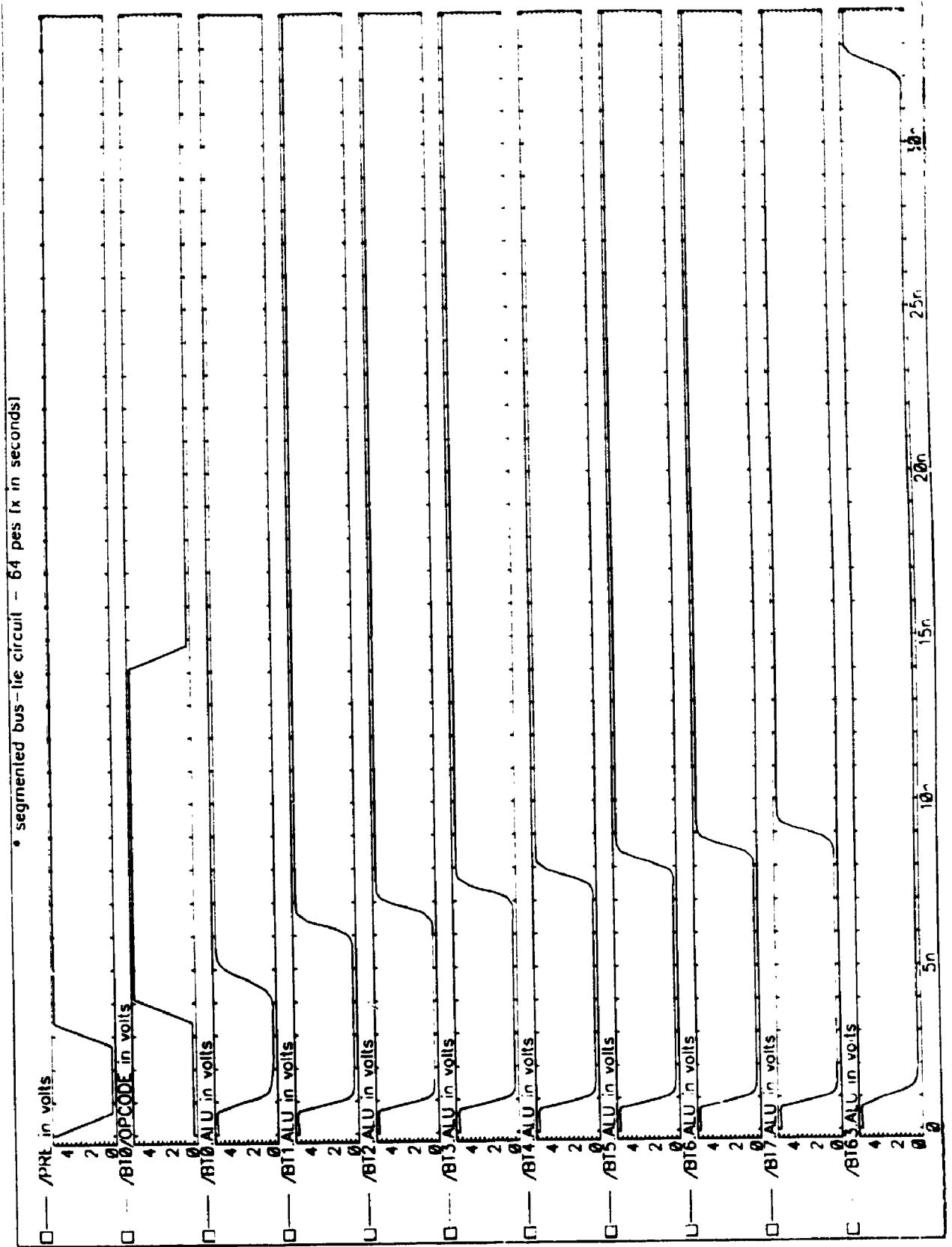


Figure 5.5 Programmable Segments Bus-tie (PSB) Network simulation

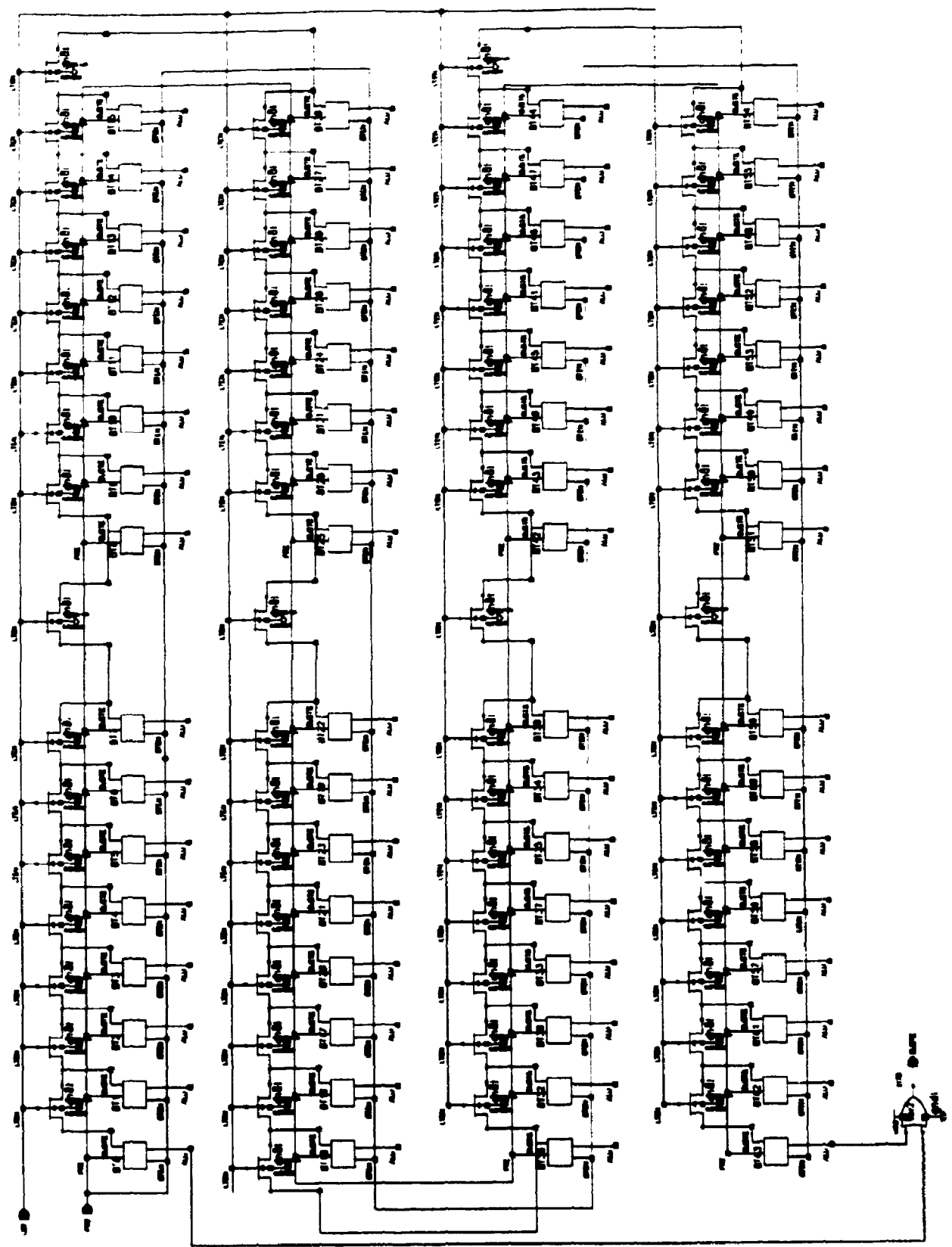


Figure 5.6 64 PE PSB Circuit

OR gate, Figure 5.6 (lower left corner), whose output feeds the BTOUT pin. The worst-case propagation time seen by the chip output is now from the middle of the network to any of its ends. Assuming a 1 ns delay from the array extremity to the OR gate inputs, the global bus-tie result can be available at the output of the OR gate in under 15 ns. This is the data that is made available at the BT output pin of the C*RAM chip. While the BTOUT pin receives the new bus-tie value in 15 ns, the PSB network continues to propagate it to the rest of the array, with the previously mentioned end-to-end worst-case time of 27.8 ns. The difference can be used by the C*RAM controller to schedule its internal operations in order to start the next instruction by the time the network has finished evaluating. Another suggestion for the controller is to implement variable instruction timing according to the word length, since shorter words require less time to execute the segmented bus-tie. This option is only possible if the masks are constants known to the controller.

5.2 Shift Operations with Word Boundaries

A second PE development also oriented towards bit-parallel operation is described in this section. It involves providing circuitry that will define the boundary between words during the shift left/right operations. Considering the Baseline PE operation, the shift operations act uniformly across all the PE array, overwriting the X register with the neighboring right PE ALU result (for shift-left), or the Y register with the neighboring left PE result (shift-right, see Figure 5.7). In the figure, the output of the multiplexor is now the output of the ALU inverting buffer, as opposed to the previous figures where the precharged ALU node, MUX8, (marked with an x) was of interest.

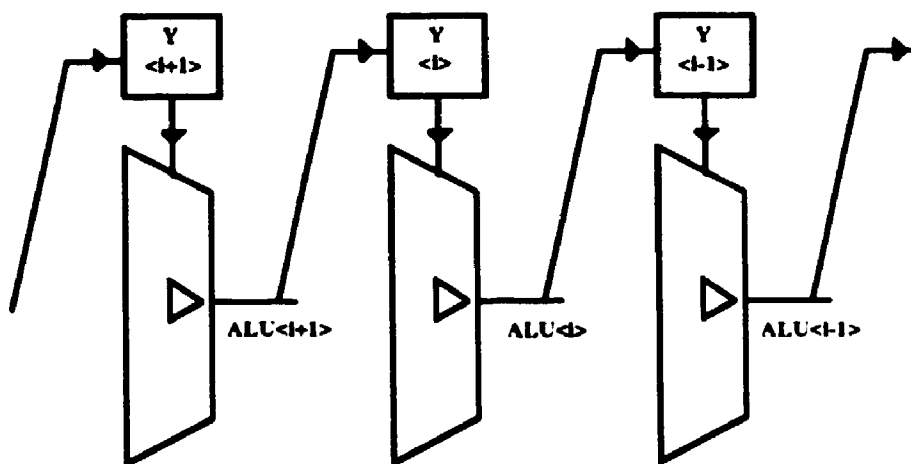


Figure 5.7 Global Shift-right data flow

We are interested in breaking this uniformity by introducing boundaries where the destination register (X or Y) receives a specified value instead, as shown for example in Figure 5.8. In the figure, there is a fixed boundary between $PE\langle i \rangle$ and $PE\langle i-1 \rangle$. When a shift operation is executed, register Y of $PE\langle i-1 \rangle$ receives the fixed value "1".

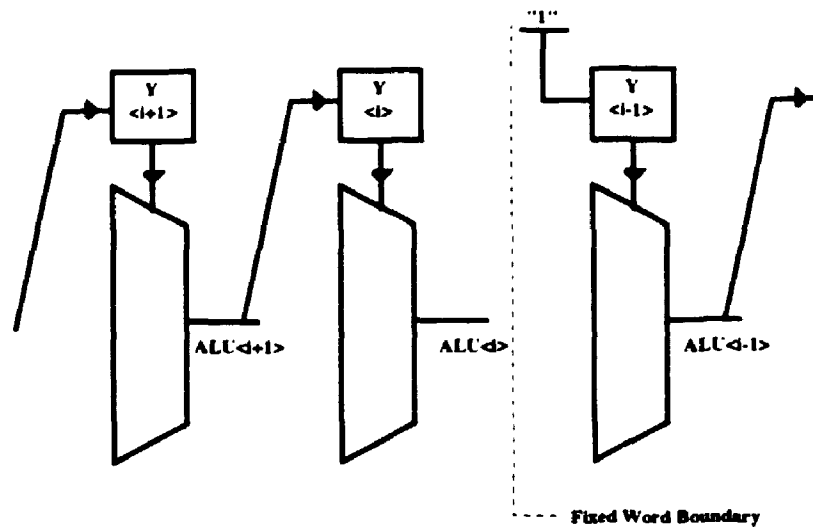
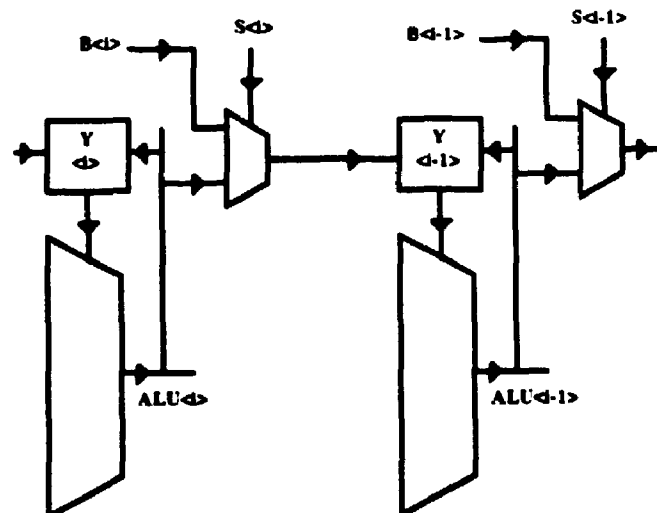


Figure 5.8 A restricted configuration for Segmented Shift

Beside the fixed boundary and the hardwired value, the disadvantage of this particular example is that the global shift operation is no longer possible. A circuit that restores the global shift is shown in Figure 5.9, where the shifted bit is now selected between the ALU result and a boundary value B . The multiplexor selection line S should select transmitting B wherever a word boundary is desired. In a global shift operation, all multiplexors select the ALU result.

Figure 5.9

Introducing shift boundaries while conserving the global shift



The main design options to be considered hence are fixed/programmable boundaries and fixed/programmable values to be inserted at the boundary. As expected, the fixed alternatives are less expensive in hardware but also less flexible. They are deemed to be an excessive constraint on the experimental C*RAM architecture, therefore the programmable alternative is chosen. Complete programmability of the boundaries implies adding two more 1-bit registers to the new PE. Figure 5.10: one will hold a bit designating the PE as a boundary PE, the S (Select) register. The second new register will hold the bit to be shifted into the adjacent PE at a word boundary, the B (Bypass) register.

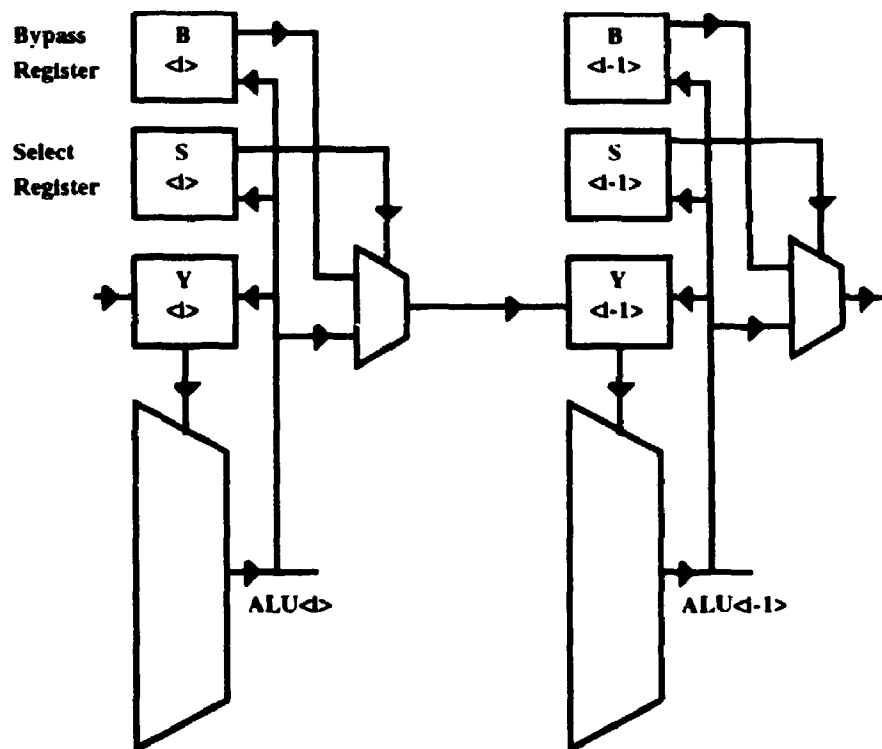


Figure 5.10 Flexible implementation of Segmented Shift

5.2.1 Circuit Implementation

The two registers S and B are implemented as single access latches, presented in section 3.3. There is no need for double access since the two registers have a rather specialized function. The 2-to-1 multiplexor is a new building block for the extended PE and its implementation is discussed below.

There are two different ways to implement the 2-to-1 multiplexor: standard CMOS logic, Figure 5.11 and transmission gate logic, Figure 5.12. Comparing transistor counts, the first design uses 7 PMOS and 7 NMOS transistors, while the second uses 3 NMOST/3 PMOST. Hence, due to the clear area advantage, the transmission gate implementation is preferred.

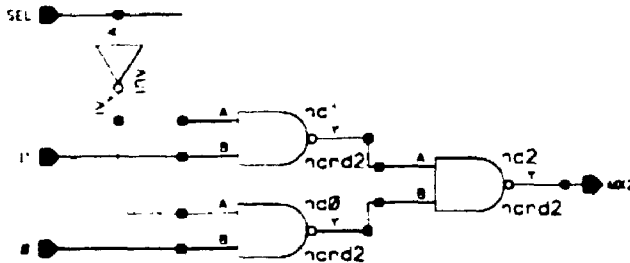


Figure 5.11 Logic gate multiplexor

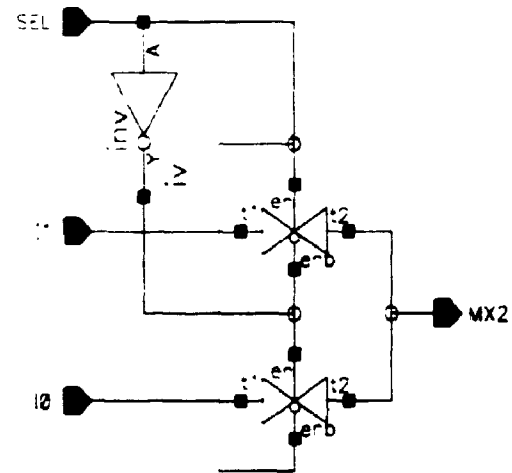


Figure 5.12 Transmission gate multiplexor

In fact, the area could be apparently further reduced by using NMOS pass transistors instead of the full transmission gates. The problem though is that the multiplexor output has to write a latch through an access NMOST. Adding a second series NMOST would be equivalent to doubling the channel length of the latch access transistor (or equivalent, halving the width), which has a negative impact on the symmetric latch LOW-HIGH state reversal, as shown by the simulations in section 3.5.3.

5.3 Dedicated Add/Subtract Ripple-Carry Circuit

So far we have introduced the Programmable Segments Bus-tie (PSB) and the segmented shift in order to favor bit-parallel PE operation. Part of the utility of these circuits will be shown in this section. One of the most important operations in a computer is integer addition/subtraction. Also very important operations, comparisons are in fact subtractions, which in 2's complement number representation are executed by the same adder/subtractor hardware. Hence, the natural next step in enhancing the PE is to add circuitry dedicated to this function, and investigating a trade-off between the amount of extra hardware and the functionality.

As shown in section 3.1, a 1-bit full adder consists of two logic functions of three variables: exclusive-or (XOR3) and carry-out (CY3, or the 2-of-3 majority function). With the Baseline PE, a multibit add/subtract is performed in a bit serial manner: two separate ALU cycles are required to generate XOR3 and CY3 results, with the carry-out result kept in one local register to be used in the next significant bit full add.

Assuming we want to execute an n -bit bit-parallel integer add/subtract on the Baseline PE, a problem is posed by the carry-out propagation from the LSB to MSB. If the carry-out is to be generated by the current ALU, we still need n PE cycles to compute and propagate the carry-outs, plus one final cycle to compute the sum (XOR3), this time though for all n bits at once. Comparing to the $2n$ cycles required by the bit-serial version, the bit-parallel add needs $n+1$ cycles to operate on n times fewer words. Hence, there is a speed advantage for individual adds/subs, but for a large number of operations there is a slight disadvantage in the throughput of the bit-parallel program.

Note that in order to do the bit-parallel add, the segmentable shift circuit is a must. The shift left (for example) would be used to transport the carry-out from one bit to the next significant bit. Hence, word boundaries are necessary in order to provide a known carry-in for the LSB (0 for add, 1 for subtract) and also to prevent the carry-out from an MSB from corrupting the subsequent CY3 operations in the neighboring word.

The previous discussion is valid for the case of Baseline PE, whose single-output ALU is the 8-to-1 multiplexor. The obvious idea to speed-up the bit-parallel add/subtract is to add one more ALU logic function unit, capable of computing the CY3 function. The immediate question is whether this function unit should be programmable, i.e. able to generate other logic functions beside the CY3, similar to the original ALU. We decide against this alternative, since the area of an 8-to-1 multiplexor represents the bulk of the PE. Also, as shown below, extra hardware beside the function generator itself is needed for the operation, so we need a minimum area carry generator. Hence, the new function unit is hardwired to generate only the CY3 function, and will be referred to as the Carry block or ALU Carry.

Examining the data storage required by the bit-parallel ALU, we observe that two operands and a

carry-out word have to be stored in PEs. These three operands are inputs to the final XOR3 operation. One local PE register, X or Y, has to be reserved for storing the resulting carry-out. The other register would store one of the operands while the remaining operand would be supplied by M as the last row read from memory. Storing the carry-out in X or Y means adding one more access transistor beside the existing two. Since the original PE layout is too dense to support this modification, a new register called C (for Carry) is added. The output of C is directly connected to one input in the Carry block. Since the contents of C is one operand in the final XOR3 executed by the ALU, C has to have access to one of the three selection inputs supplied by Y, X and M. This is done by multiplexing C and one of the three, by adding one 2-to-1 multiplexor and the corresponding selection control line, as part of the Control Opcode (COP).

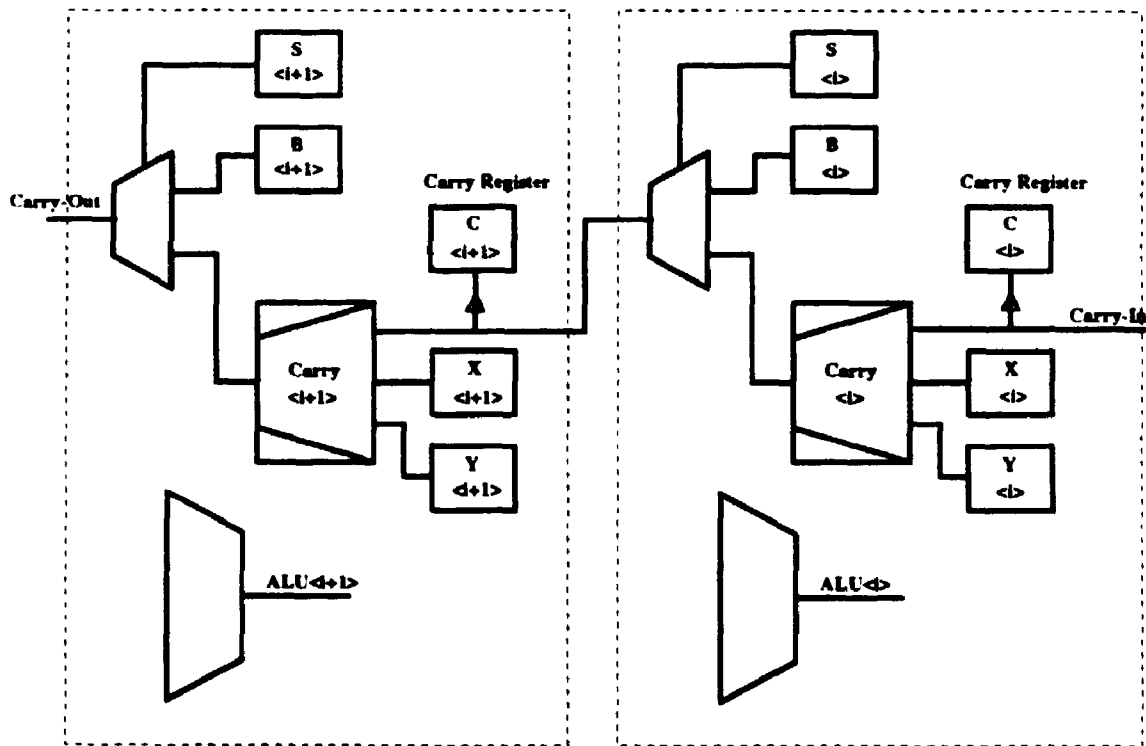


Figure 5.13 Carry block in the Extended PE

Examining Figure 5.13, we note that the Select (S) and Bypass (B) registers introduced to implement the Segmented Shift also intervene in carry-out propagation with a similar purpose. They are used to define the word boundary, in conjunction with a new 2-to-1 multiplexor that selects

between the Carry Block result and the Bypass register value. The selection line is controlled by the S (Select) register. As in the case of Segmented Shift, at a word boundary S will select the Bypass value which will represent carry-in for the neighboring word. Add and subtract operations can be done at the same time after the appropriate operand complementing, because Bypass can supply a carry-in of "0" for adds or a carry-in of "1" for subtracts, after being programmed accordingly.

The final ripple-carry circuit presents one variation compared to the circuit in Figure 5.13 (above): the 2-to-1 multiplexor is removed from the carry propagation path every other PE, Figure 5.15. This results in faster propagation of the ripple-carry. The word boundaries can now be defined in increments of 2 bits, which is not a problem since most operations use 8-bit words or more. In fact, a largest 4 bit increment might represent the right balance between ripple-carry speed and word length flexibility. This option was rejected because it involved managing the design of more PE cells (left PE or right PE, with or without the multiplexor), hence would have added extra risk to the project.

5.3.1 Ripple-Carry Circuit Implementation

For the same reasons discussed in the BPE circuit implementation, the ripple-carry circuit is designed as a dynamic-logic circuit. The transistor level schematic is shown in Figure 5.14. The dynamic logic alternative eliminates the bulky PMOS section with the exception of one precharge PMOS. Also, it reduces the capacitance on the output node, hence it speeds up the carry propagation. As in the case of the ALU multiplexor and Segmented Bus-tie circuits, there is a weak PMOS latch-back transistor to protect the HIGH state on the precharged node. Unlike the ALU multiplexor, there is no weak NMOS latch-back, since the carry block is either in precharge or in evaluation (controlled by the same signal CYPRE), hence a path to either supply is always open. The situation of the PMOS latch-back is very similar to that of Segmented Bus-tie, as the duration of the evaluation phase can be long, depending on the current word length. In a well controlled technology the latch-back PMOS could be removed in order to gain extra speed.

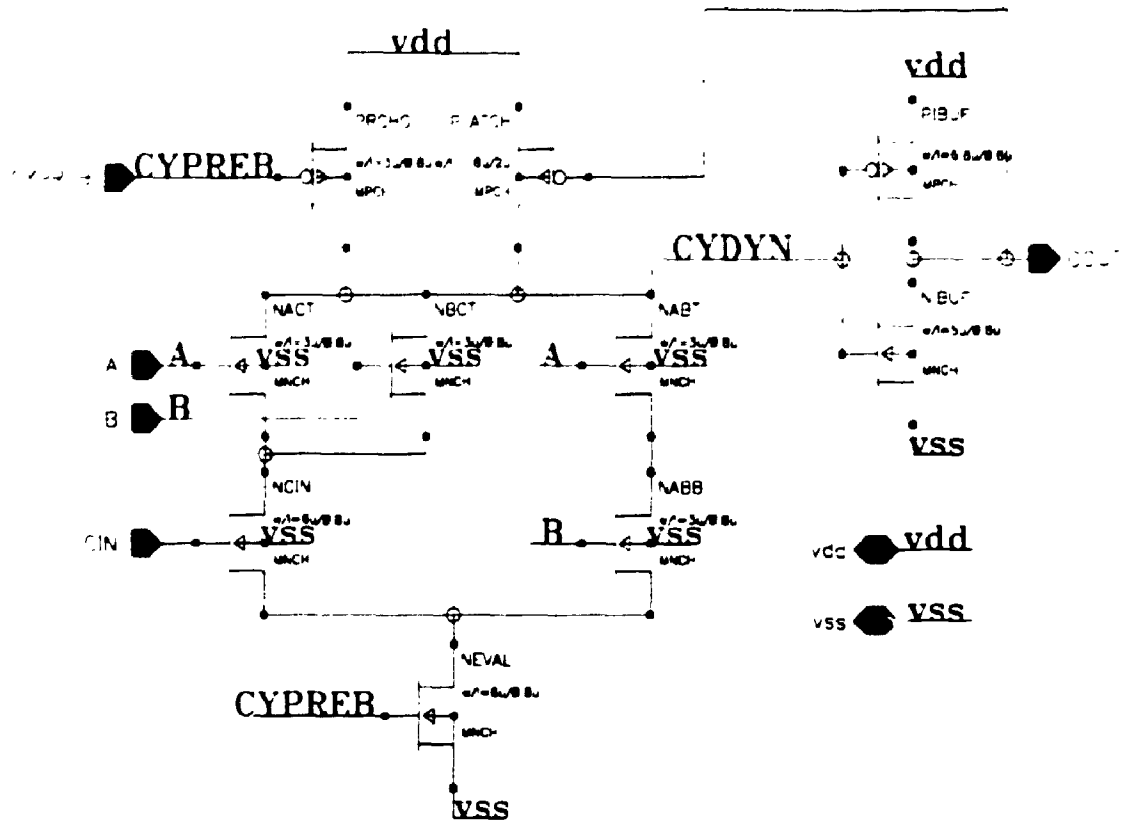


Figure 5.14 Carry circuit schematic

Discussing qualitatively transistor sizing for the carry circuit, Figure 5.14, we have the following criteria. The most critical transistors for carry propagation are NCIN and NEVAL, hence they will be the largest in the circuit (in terms of w/l). The latch-back PLATCH has the lowest w/l , hence minimum width and the biggest length as permitted by layout. The inverting buffer (PIBUF and NIBUF) has to be powerful enough to drive the carry-out through one transmission gate 2-to-1 multiplexer, one latch access NMOST and into the Carry latch of the following PE. The other NMOS evaluation transistors, NACT, NBCT, NABT and NABB have all minimum length and as large a width as permitted by the layout. The final transistor sizes are constrained by the PE area available and are shown in Figure 5.14.

In fact, a ripple-carry chain is a cascade of carry blocks. A cascaded sequence of dynamic logic blocks would normally require some form of latching (like clocked CMOS inverters) between blocks, and multiple evaluation cycles to propagate the results from one end to the other. Or, a sequence of dynamic logic blocks might avoid intermediary latching if it uses alternate n and p

evaluation blocks, with the speed or area penalty imposed by the p block.

Examining the ripple carry propagation, we remark that if at any point in the chain the result of the evaluation is a logic-1 value (considered at the output of the inverting buffer), there is no possibility of this result having to be modified back to logic-0 as a result of a new value on carry-in. In other words, considering Carry-out as the 2-of-3 majority function, if initially only 2 out of the 3 inputs were logic-1, the fact that the third input later becomes logic-1 as a result of carry propagation will not affect the result of the initial evaluation. This monotonicity property makes the ripple carry chain very amenable to dynamic logic implementation: only one precharge phase is necessary, after which the chain is left in evaluation for the duration of worst case LSB to MSB propagation time.

5.3.2 Ripple-Carry Chain Simulation

Figure 5.15 presents a ripple-carry circuit with 8 stages. As explained above, there is one word boundary multiplexor every other stage, for a total of four. The circuit implementation of the 2-to-1 multiplexor is based on transmission gates, as discussed in section 5.2. All MOST sizes represent the actual final layout dimensions in the C512mp512 chip. In order to evaluate propagation speed, the circuit in Figure 5.15 is simulated with typical BATMOS parameters, at a temperature of 75°C. The inputs are set up to create the worst-case rippling, from LSB to MSB: one operand is all logic-1, the other is all logic-0 and the initial carry-in is logic-1. The simulation waveforms are shown in Figure 5.16. The ripple-carry cycle starts with a fixed duration precharge phase (CYPREB= LOW), then the circuit is left in evaluation. The total cycle time is dependent on the number of stages:

$$T_{\text{Carry Cycle}}(n) = T_{\text{Precharge}} + 0.5 \cdot n \cdot T_{\text{DPE Carry}}$$

where:

$T_{\text{Precharge}}$ = duration of CYPREB=LOW

n = even number of bits (i.e. number of PEs)

$T_{\text{DPE Carry}}$ = carry propagation time through a double PE block (including one multiplexor)

For our circuit, the values resulting from simulation are: $T_{DPE\ Carry} = 1.2$ ns. Considering $T_{pre-charge} = 3$ ns, we have $T_{Carry\ Cycle(8)} = 7.8$ ns, $T_{Carry\ Cycle(16)} = 12.6$ ns and $T_{Carry\ Cycle(32)} = 22.2$ ns.

The C512mp512 C*RAM chip, which has 512 PEs, is capable of executing $N_{BPOP} = 16$ 32-bit additions in parallel. A bit-parallel add requires first a ripple-carry cycle, followed by an exclusive-or (logic sum) operation of the resulting carry word with the two operands. Assuming 40 MHz system operation and a rate of one C*RAM instruction per cycle (25 ns), then 16 32-bit adds are executed in $T_{add32} = 50$ ns. By dividing T_{add32} by the number of additions, we obtain an equivalent time of 3.125 ns/chip/operation. This value is comparable to the operation time of a family of experimental stand-alone 32-bit ALUs implemented in 0.25 μm SRAM CMOS, reported in 1993, [Suzu93].

Trying to extrapolate to DRAM-based C*RAM, we have to weight the following factors:

- a. DRAM are notorious for using "slow" transistors, hence a disadvantage compared to BATMOS:
- b. the virtual DRAM C*RAM will use a more advanced technology than 0.8 μm , and this will compensate for the slower transistors;
- c. there will be more PEs in a DRAM C*RAM, for example 2048 PEs in a 16 Mb DRAM, hence even if the execution time is somewhat higher, the equivalent per operation time is estimated to be at least two times smaller than stand-alone ALUs (one C*RAM chip considered).
- d. the extra area taken by the Extended PE will represent a bigger percentage of the DRAM area, or equivalent, there would be fewer PEs for the same percentage of area increase.

The 2048 PEs in a 16 Mb DRAM estimation mentioned above has factored in point d. The argument is based on the design work done by D.Elliott on an actual 4 Mb DRAM platform. The resulting C*RAM had 2048 PEs for a 15% increase in the initial area. It is assumed thus that by going to the next DRAM generation, the 16 Mb. the number of PEs can be doubled to 4096 for about the same 15% area increase, or alternatively, the same number of Extended PEs is conserved.

The layout of two Carry Blocks in adjacent PEs is shown in Figure 5.17.

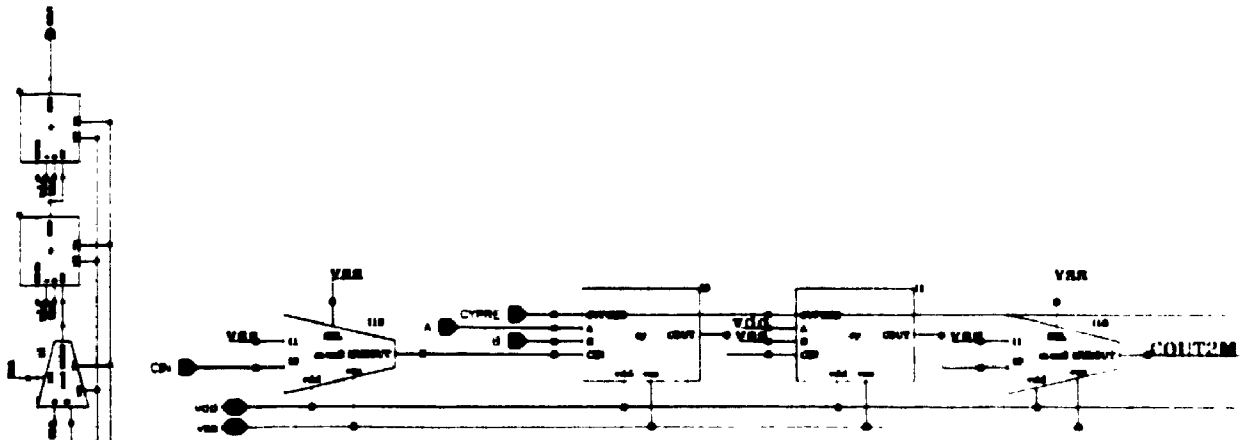


Figure 5.15(a) (Left): A chain of 8 Carry blocks

Figure 5.15(b) Detail of Carry chain circuit

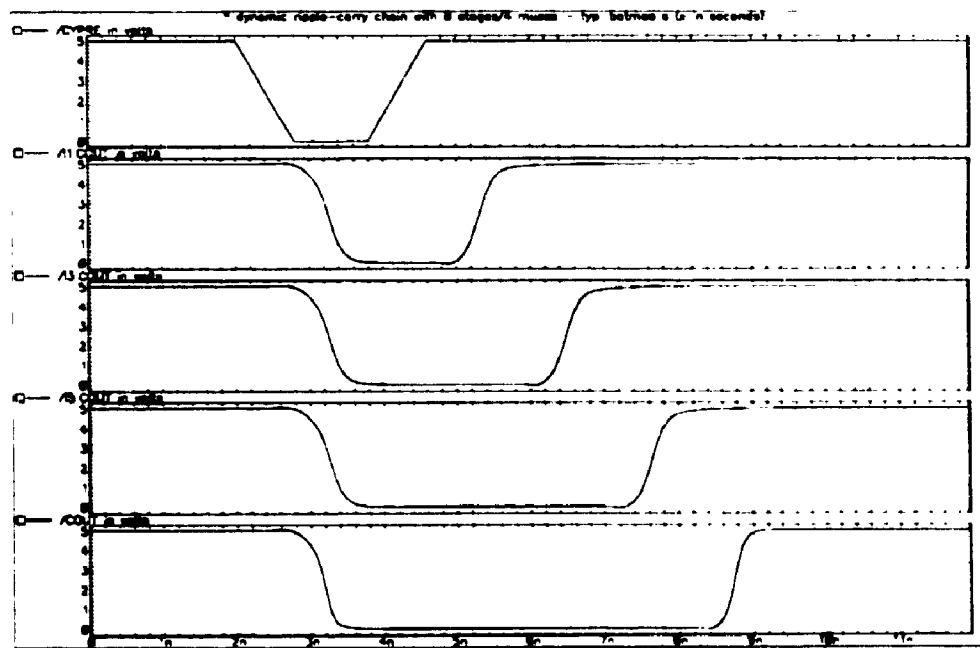


Figure 5.16 Carry chain Spice simulation

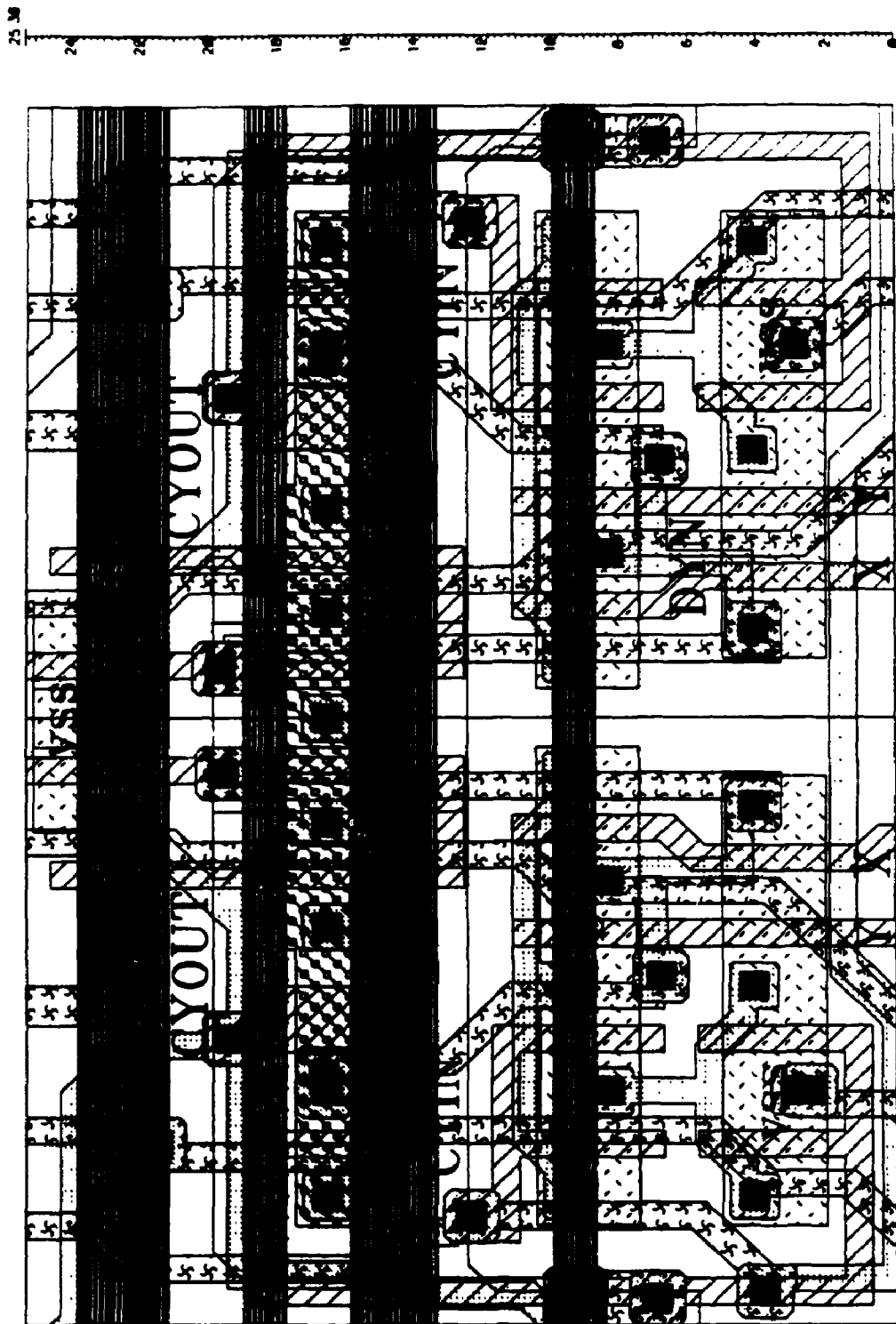


Figure 5.17 Layout of Carry blocks in adjacent PEs

5.4 Bit-parallel Integer Multiplication with the Extended PE

This section demonstrates how all the supplementary hardware added to the Baseline PE is used in performing bit-parallel integer multiplication. The operation is performed as a sequence of add and shift operations. The main characteristic of the multiplication is that it is done entirely within the PE array, without any memory access after the operands have been brought into the PE registers. The first advantage of this approach over bit-serial multiplication is shorter execution time, because it avoids redundant memory access to retrieve the multiplicand for each partial add. The speed advantage is bigger for DRAM C*RAM, for which memory cycle times are 5 to 10 times longer than SRAM. The advantage of maintaining the whole sequence within the PE array is low power, for both SRAM and DRAM based C*RAMs, because it avoids the sense amplifier power consumption.

5.4.1 Extended Register Set

The Baseline PE offers only two data registers, X and Y. The third data input to the ALU, M, is assumed to be registered in the memory interface circuitry. In order to support the bit-parallel multiply, four new registers are defined in the Extended PE. Figure 5.18 shows this extended data register set.

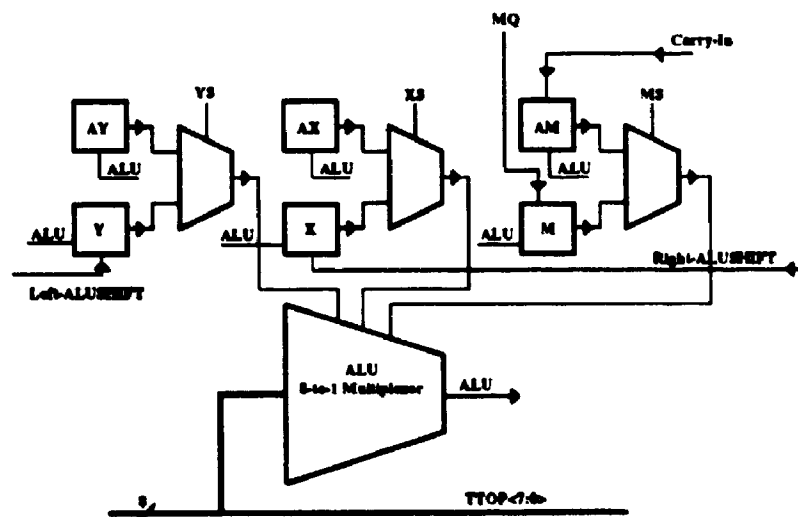


Figure 5.18 Data registers in the Extended PE

The M register is now the explicit source of data line M. M is a dual-access latch and can be writ-

ten by both the ALU output and the local memory output. Three more registers, AX, AY and AM constitute the "alternate register set". As Figure 5.18 shows, the word "alternate" indicates that only one in a register - alternate register pair can source the ALU multiplexor at one time. The selection is made by three 2-to-1 multiplexors, controlled by three new global signals, XS, YS and MS. All data registers are writable by the ALU output, which adds four more global write-register signals, WAX, WAY, WAM and WM.

The choice of this organization was directed by the "minimum hardware" criterion. The 2-to-1 multiplexor is very compact when implemented with minimum size transmission gates, as shown before.

In a previous section, it was said that the result of the hardwired ripple-carry chain can be written into a new register, temporarily called C. This register has to have access to the ALU multiplexor in order for a carry result to be used in the subsequent XOR operation, hence C should be in fact one of the six data registers. Since X, Y and M are already dual-access latches, the practical choices remain the single-access AX, AY and AM. Further restrictions are imposed by the Extended PE layout, where AX and AY are added close to X and Y in a dense connection area. The AM register remains thus the only alternative as Carry-out destination.

The Extended PE architecture is presented in Figure 5.19 and the complete layout of two adjacent Extended PEs (Left and Right) is shown in Figure 5.20.

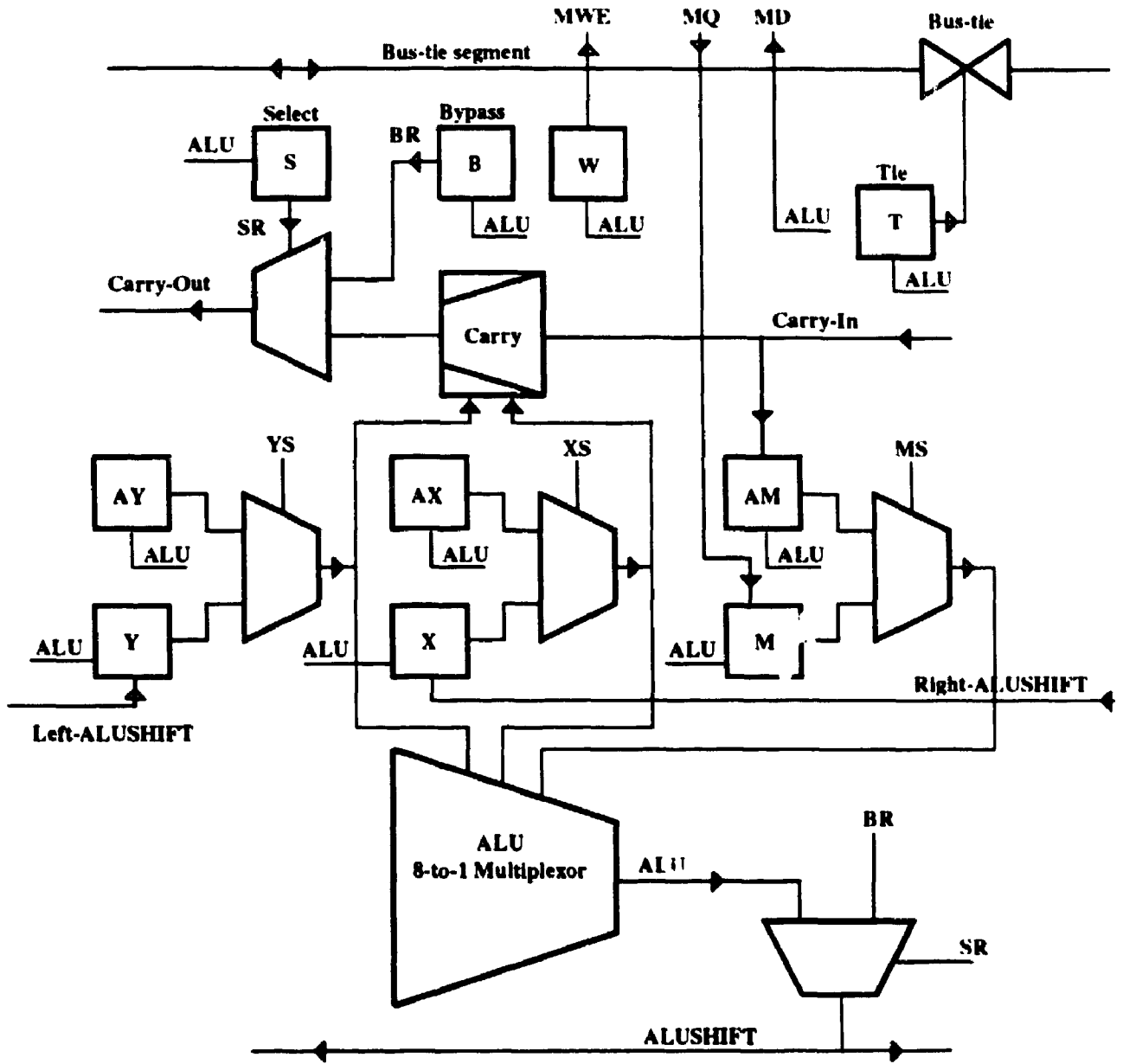


Figure 5.19 Extended PE (XPE) architecture

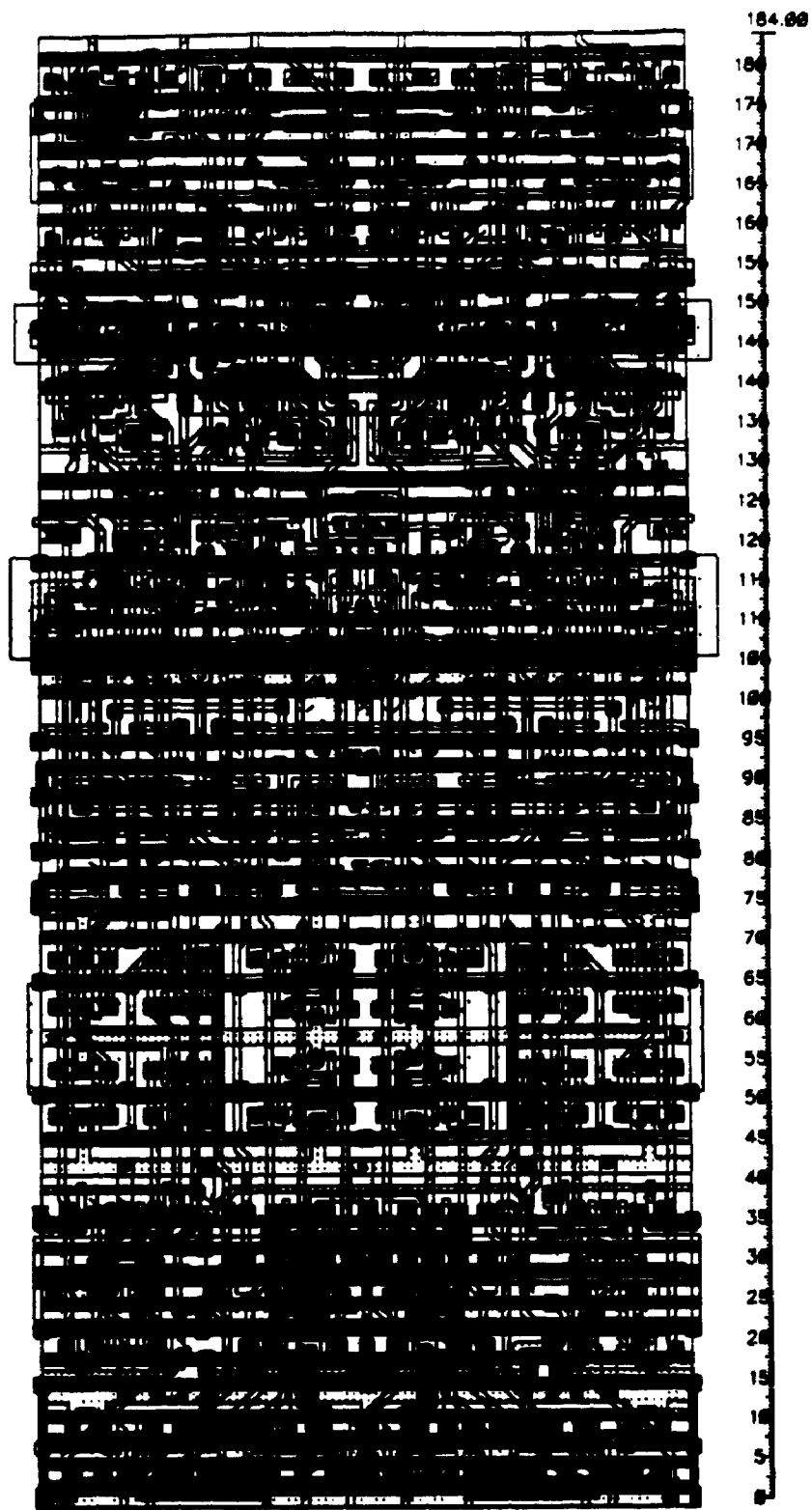


Figure 5.20 Dual (Left-Right) Extended PE layout

5.4.2 Bit-Parallel Multiply Example

Since a generic presentation would be fairly difficult to follow, an actual example is shown. We assume we are working on 4-bit words. The result of the multiplication is an 8-bit number. Although the numbers are signed integers represented in 2's complement, the example uses only positive numbers. In general, an m by n multiplication gives a result of length $m+n$. If p is the total number of processors, then $p/(m+n)$ is the number of multiplications being performed in parallel (assuming p is divisible by $m+n$).

The example multiplication is presented in Table 5.1 through Table 5.8, each table corresponding to a C*RAM basic operation. Only the first cycle is shown, corresponding to the first partial product. The multiplication procedure takes n such cycles. In each table, the operands of the operation to be executed are in bold characters. The result of the current operation appears in a shaded box in the subsequent table. The arrows in the X and Y register boxes are intended as a reminder that these registers are hardwired destinations of Shift-left and Shift-right, respectively. The table format is intended to suggest which operands are available to the ALU: there are three columns, corresponding to the three ALU operands. From each column, only one of the two registers can source the ALU selection line.

The register assignment is as follows:

X = temporary results; also used to shift left the multiplicand

Y = the multiplier **B** initially, then right-shifted versions of it, one bit shift after each partial sum.

AX = the 8-bit "1" mask: p0000_0001

AY = running sum (partial product), holding the final 16-bit result at the end of the loop.

M = the multiplicand **A** initially, then left-shifted versions of it.

AM = temporary carry-out for partial adds and scratch register

T = the 8-bit "complement 1" mask: p1111_1110. This breaks the bus-tie into 8 bit segments.

S = p1000_0000. This will cause MSB of **B** (Bypass) to be shifted left/right instead of MSB ALU.

B = p0xxx_xxxx. This provides the 0 left-shifted as multiplicand LSB after each partial sum.

W = don't care. There is no write-back to the memory, so Write Enable is not used.

The W, T, S and B registers keep their respective values over the entire multiplication procedure and are therefore shown only in the first table.

Table 5.1 AM = Bus-tie(AX and Y)

AM (Carry)	AX=Mask	AY=Sum	S (Select)	T (Tie)
(000)_000	0000_0001	0000_0000	1000_0000	1111_1110
(000)_0110	0000_0110	0000_0101	0xxxx_xxxx	xxxx_xxxx
M=A	<- X=A or 0	Y=B ->	B (Bypass)	W (WrEn)

Step 1: B and Mask, followed by bus-tie, result in AM; result is either 1111_1111 or 0000_0000
 Uses: the ALU (2 operands) and the Programmable Segments Bus-Tie

Table 5.2 X = AM and X

AM (Carry)	AX=Mask	AY=Sum
1111_1111	0000_0001	0000_0000
0000_0110	0000_0110	0000_0101
M=A	<- X=A or 0	Y=B ->

Step 2: X is now either the multiplicand A ($A \cdot 1 = A$) or constant-0 ($A \cdot 0 = 0$).
 Uses: the ALU (2 operands)

Table 5.3 AM = Ripple-carry(X, AY)

AM (Carry)	AX=Mask	AY=Sum
1111_1111	0000_0001	0000_0000
0000_0110	0000_0110	0000_0101
M=A	<- X=A or 0	Y=B ->

Step 3: Generate Carry of running Sum and current term (A or 0)

Uses: the hard-wired Ripple-carry block. The two operands of the Carry block are (X or AX) and (Y or AY).

Table 5.4 $AY = AM \text{ xor } X \text{ xor } AY$

AM (Carry)	AX=Mask	AY=Sum
0000_0000	0000_0001	0000_0000
0000_0110	0000_0110	0000_0101
M=A	<- X=A or 0	Y=B ->

Step 4: Compute new partial product; ALU is executing XOR3.

Uses: the ALU (3 operands).

Table 5.5 $RX = \text{Shift-right}(Y)$

AM (Carry)	AX=Mask	AY=Sum
0000_0000	0000_0001	0000_0110
0000_0110	0000_0110	0000_0101
M=A	<- X=A or 0	Y=B ->

Step 5: Shift right the multiplier B; an unknown bit value is shifted into MSB.

Uses: the ALU (1 operand) and the Shift-right connection (Y is the hardwired destination)

Table 5.6 $LX = \text{Shift-left}(M)$

AM (Carry)	AX=Mask	AY=Sum
0000_0000	0000_0001	0000_0110
0000_0110	0000_0110	x000_0010
M=A	<- X=A or 0	Y=B ->

Step 6: Shift left the multiplicand A into left X register; a "0" is shifted into LSB.

Uses: the ALU (1 operand) and the Shift-left connection (X is the hardwired destination)

Table 5.7 $M = X$

AM (Carry)	AX=Mask	AY=Sum
0000_0000	0000_0001	0000_0110
0000_0110	0000_1100	x000_0010
M=A	<- X=A or 0	Y=B ->

Copy the shifted A from X to M. This is the operation that might be eliminated with a different left-right connectivity scheme

Table 5.8 AM = Bus-tie(AX and Y)

AM (Carry)	AX=Mask	AY=Sum
(xxx)_ (xxx)	0000_0001	0000_0110
0000_1100	0000_1100	x000_0010
M=A	<- X=A or 0	Y=B ->

Restart the cycle.

5.4.3 Signed Multiply

In generalizing the previous procedure to signed multiplication (in 2's complement representation), two elements have to be accounted for:

- a. sign-extension of the multiplicand over the $m+n$ final product word length, and
- b. sign-weighted final partial add: the last shifted multiplicand (corresponding to the multiplier MSB) has to be negated before being added to the partial product, if the multiplier is a negative number.

The sign-extension of multiplicand replicates its MSB to fill an $m+n$ long word. For example: xxxx_1001 becomes 1111_1001 after sign extension (here, $m=n=4$). This can be accomplished with a procedure very similar to the first step in a partial sum generation in the above algorithm:

1. Isolate the multiplicand MSB with the mask 0000_1000
2. Use segmented bus-tie to replicate the MSB
3. Mask the previous result with 1111_0000 and OR this result with the original multiplicand.

To negate the multiplicand function of the multiplier sign, the following extra steps are needed before the final partial add:

1. Execute a XOR between the shifted multiplicand and the word that replicates the multiplier MSB.
2. Overwrite the "1" mask in AX with the result of the same mask ANDed with the replicated

multiplier sign. The result is 1 where subtraction is needed and 0 for addition.

3. Add the previous result to the final product.

5.4.4 Bit-parallel Multiply Execution Time

We assume n -bit word length for both operands and a sign-extended multiplicand available at the beginning of multiply. Examining the tables presented in the first subsection, the time required to complete a partial add is:

$$t_{PA} = bt(2n) + t_{OP} + rc(2n) + t_{OP} + t_{OP} + t_{OP} = bt(2n) + rc(2n) + 5 \cdot t_{OP}$$

where:

$bt(n)$ = word length dependent segmented bus-tie execution time

$rc(n)$ = word length dependent ripple carry execution time

t_{OP} = ALU execution time

For the n by n multiply, including reading the two operands from memory and writing the result back to memory (memory read and write are each done in one access time t_{ACC}):

$$t_{MPY}(n) = n \cdot t_{PA} + 3 t_{ACC}$$

By amortizing this time over the total number of operations executed in parallel, in a system with c C*RAM chips with p processors each, we have:

$$t_{MPY,eq}(n) = cp/2n \cdot t_{MPY}(n) = cp/2n \cdot (n \cdot t_{PA} + 3 t_{ACC})$$

For example, we assume $n=8$, which implies that both bus-tie and ripple-carry propagate over 16 bits. Also, we assume 50 MHz system operation, which allows for memory access time, $bt(16)$ and $rc(16)$ to be performed within the 20 ns system clock cycle each. In this case:

$$t_{MPY}(8) = 8 \cdot (7 \cdot 20 \text{ ns}) + 3 \cdot 20 \text{ ns} = 1,180 \text{ ns}$$

The 512 PE C*RAM can execute $pn=512/16=32$ multiplies in parallel, which gives an equiva-

lent time:

$$t_{MPY,eq} = 1,180/32 = 36.875 \text{ ns}$$

Obviously, the equivalent execution time can be reduced by amortizing the multiply time over a larger number of C*RAM chips.

5.5 System Advantages of Bit-Parallel C*RAM

Going beyond the speed advantages of basic bit-parallel operations, there are a number of system-level arguments that favor bit-parallel over bit-serial computation. A system advantage of the bit-parallel oriented C*RAM is the elimination of the transposing cache (TC, also called corner-turning cache) from the C*RAM system. Bit-serial computation requires a transposing cache (Figure 5.21) in order to store multibit words column-wise, as opposed to row-wise, which is the natural memory storage method. Eliminating the TC from the system reduces the overall hardware complexity, and simplifies the programmer's view of the C*RAM system. Also, it facilitates coexistence of data allocated to the system processor with C*RAM data within the C*RAM address space.

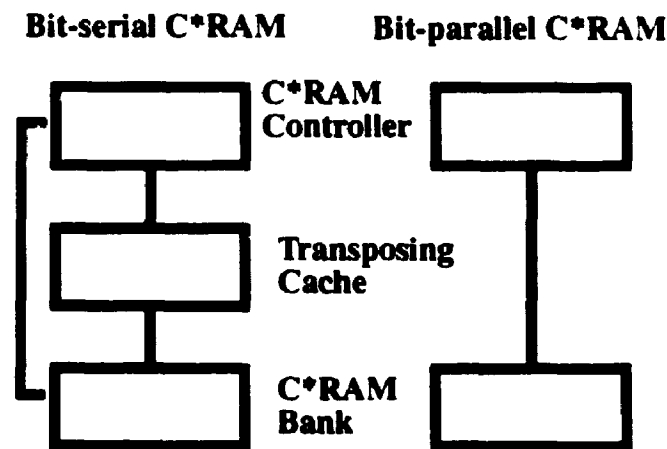


Figure 5.21 Bit-parallel vs. Bit-serial C*RAM systems

All these advantages are enhanced when the individual C*RAM chips have a wide data interface (relative to the system bus width), which allows a data word to be transferred in fewer cycles. It is assumed that the bits of a RAM word are stored in adjacent logical columns (i.e. adjacent PEs) Since byte granularity is common in today's 32-bit systems, a byte-wide data interface is the minimum practical option for bit-parallel C*RAM systems. Figure 5.22 presents three alternatives of implementing a 32-bit wide memory subsystem with three different DRAM data bus organization: 4-bit, 8-bit and 16-bit wide. The x4 organization is the least preferred for a PC*RAM system, because writing a byte (the smallest word manipulated by most systems) involves two x4 DRAM chips. Since the main processor cannot address a half-byte, extra hardware is needed to implement the effect of single DRAM chip write operations. The x16 organization is best for byte and double-byte words. It requires two cycles for 32-bit words (two 16-bit writes to the same chip), but no extra hardware since the chips can be selectively enabled by the processor.

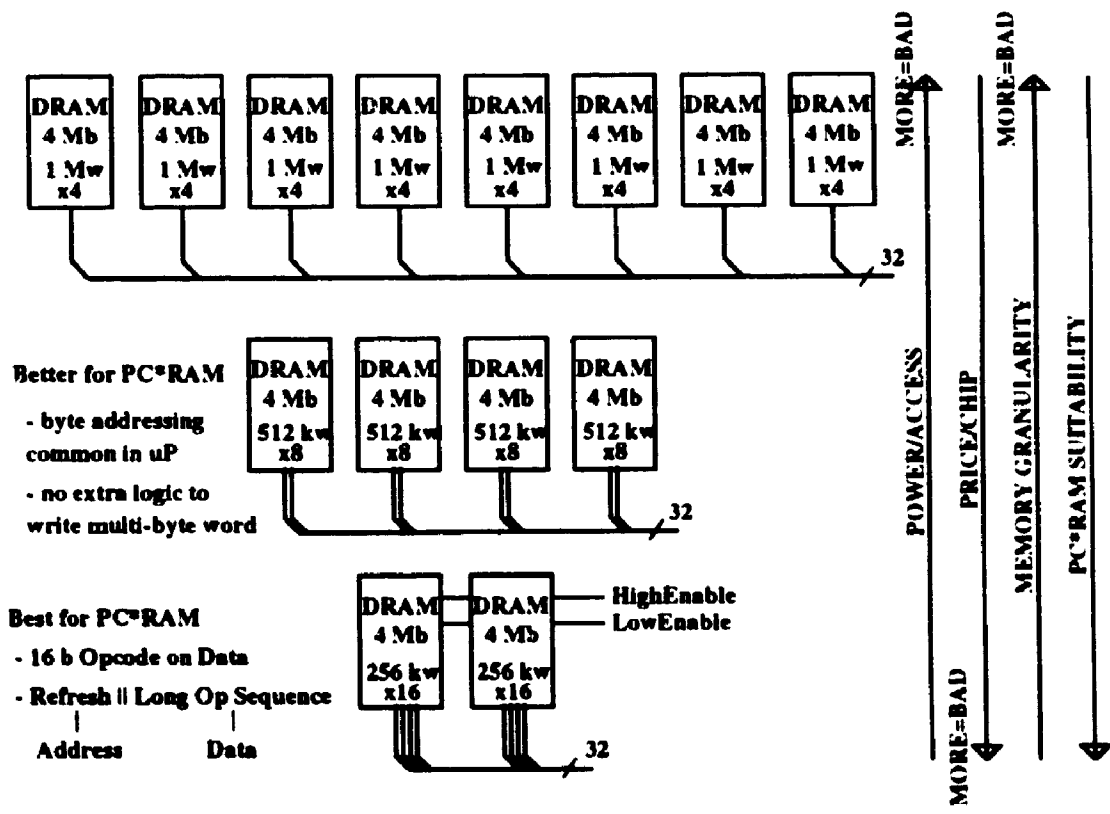


Figure 5.22 32-bit memory subsystem with x4, x8 and x16 DRAMs

More system flexibility is allowed starting with the 16-bit data bus. The full opcode can be now transported on the data bus, allowing DRAM refresh (which involves the address lines) to be per-

formed in parallel with extended C*RAM operation sequences. The 16-bit opcode contains the 8-bit Truth Table Opcode and up to 256 combinations of encoded control codes.

5.6 Summary

An original Bit-Parallel Processing Element architecture was introduced in this chapter, based on four circuit structures: the Programmable Segments Bus-tie network, the boundary registers Select and Bypass, the Ripple-carry chain and the extended register set. The new PE has 147 transistors, being twice more complex than the Baseline PE, and was designed into the 512 PE C512mp512 C*RAM. While the new PE is still a single-bit structure, the PE array can now execute bit-parallel add/subtracts, comparisons and integer multiplications. All these operations are performed within the PE array, without involving any intermediary memory access, executing thus faster and taking less power than their bit-serial counterparts. The operand word length is flexible in increments of two, allowing software programmable trade-offs between the word length and the number of operations executed in parallel by the C*RAM subsystem. The bit-parallel oriented C*RAM has a simpler system interface, as it does not require word transposing hardware. It also has the potential for a simpler programmer memory model, since the same word (address) can be processed by either the serial processor or a group of C*RAM PEs.

Chapter 6

BATMOS C*RAM Testing

6.0 Introduction

This chapter describes the testing environment created for the two versions of C*RAM chips fabricated during the project, C64p1k and Cs64p512. The primary goal of this environment is functional testing of the chips. A second objective, once functional parts have been identified, is to measure functional parameters, e.g. speed. As mentioned at the beginning of this work, the C*RAM project has a long-term nature, including investigating PE architectures, C*RAM chips, C*RAM system and eventually system software and applications. In view of a future C*RAM chip design cycle, the test environment design tried to avoid hard-wiring characteristics of the current chips. Nevertheless, elements like chip pinout have a direct impact on the test board design, hence the portability of the test system is inherently limited.

A C*RAM chip typically consists of 90-95% memory and 5-10% processing elements. C*RAM testing can thus be divided in a memory testing phase and PE testing phase. Memory testing has the purpose of identifying functional memory locations by generating a failed-bit map. The typical memory testing cycle involves writing bit patterns to all memory locations and comparing the written data against the data read. Various bit patterns and read/write orders can be applied in order to identify circuit causes for possible failures [Prin91]. For exhaustive memory testing in an industrial environment, a large number of read/write cycles have to be generated, hence the need

for a specialized memory tester to minimize test time. In the academic lab environment, the testing time of a small number of prototypes is less important, hence general types of instruments can be used.

For C*RAM chips that communicate to the exterior only via the random port (as is the case with C64p1k), finding at least one functional memory row is a prerequisite of PE array testing. Otherwise, completely functional PEs cannot have their results examined by an external system. An alternative testing method is available for the Cs64p512 chip, which has a dual serial in/out port connected to the memory array.

The C*RAM test system has to provide a high degree of flexibility. This is required by both the complexity of memory testing and the programmable nature of the C*RAM PEs. The best solution for designing a flexible test system is to use programmable instruments and design the test procedures into the system controller software.

6.1 Test Setup Hardware

The equipment used in building the test system consists of data (pattern) generators, logic analyzer, power supplies and an IBM compatible personal computer. The test configuration is shown in Figure 6.1. The actual test instruments are manufactured by Hewlett-Packard and they were available in the Department of Electronics at Carleton U. The data generators and the logic analyzer are programmable over the General Purpose Instruments Interface (GPIB), a standardized version of the Hewlett-Packard Interface Bus (HPIB). The GPIB system controller is a general purpose IBM AT compatible computer retrofitted with a National Instruments GPIB interface card. This card has software interfaces for two BASIC interpreters/compiler (GWBASIC and QBASIC) and one C compiler (Microsoft C).

A brief presentation of the data generator and logic analyzer follows:

1. The HP8081A Data Generator: this is a fairly old instrument (dating from around 1975), whose main characteristics are:

- up to 16 data channels, 1 strobe channel (usable as data) and two clock channels

- 50 MHz operation
- pulse timing (delay and width factor) programmable on the clock channels and four data channels.
- 1024 bit/channel memory

2. The HP1652B Logic Analyzer:

- 80 channels
- 10 ns sampling period
- 2048 bit/channel memory

The C*RAM chips need a fairly large number of independent signals for testing, 30 for the C64p1k, hence one 8081A Data Generator in the maximum configuration (16+1 data channels) was insufficient. A second 8081A is used in the test setup to contribute 12+1 more data channels, allocated for RAM/C*RAM address generation (only one maximum configuration 16 channel 8081A was available in the department). The first data generator was selected to provide the control signals and the data input lines.

Using two data generators involves synchronization. The 8081A provides connectors for synchronization of two instruments, but the corresponding cable assembly was missing and the information to build it proved difficult to obtain. A temporary solution was adopted, as shown in Figure 6.1. One clock channel from one 8081A (the "master" generator) is used to clock the other 8081A. The master 8081A uses the internal clock source while the slave 8081A gets its clock from the external BNC connector. This configuration introduces a sizable delay between the two sets of data, measured as 90ns. Accounting for setup times, this test configuration can be used for C*RAM testing at clock speeds below approximately 8 MHz (i.e. 125ns intervals between events).

During GPIB programming, both data generators receive the data words necessary to stimulate a C*RAM cycle. The "RUN" command is then sent first to the slave generator, which starts waiting for clock pulses on the external input. A "RUN" command is now sent to the master generator, which outputs its data word and at the same time clocks the slave. As mentioned above, the slave data word appears approximately 90 ns later. Between sending the two "RUN" commands, the

software executes an idle loop, in order to allow the slave generator to interpret the command.

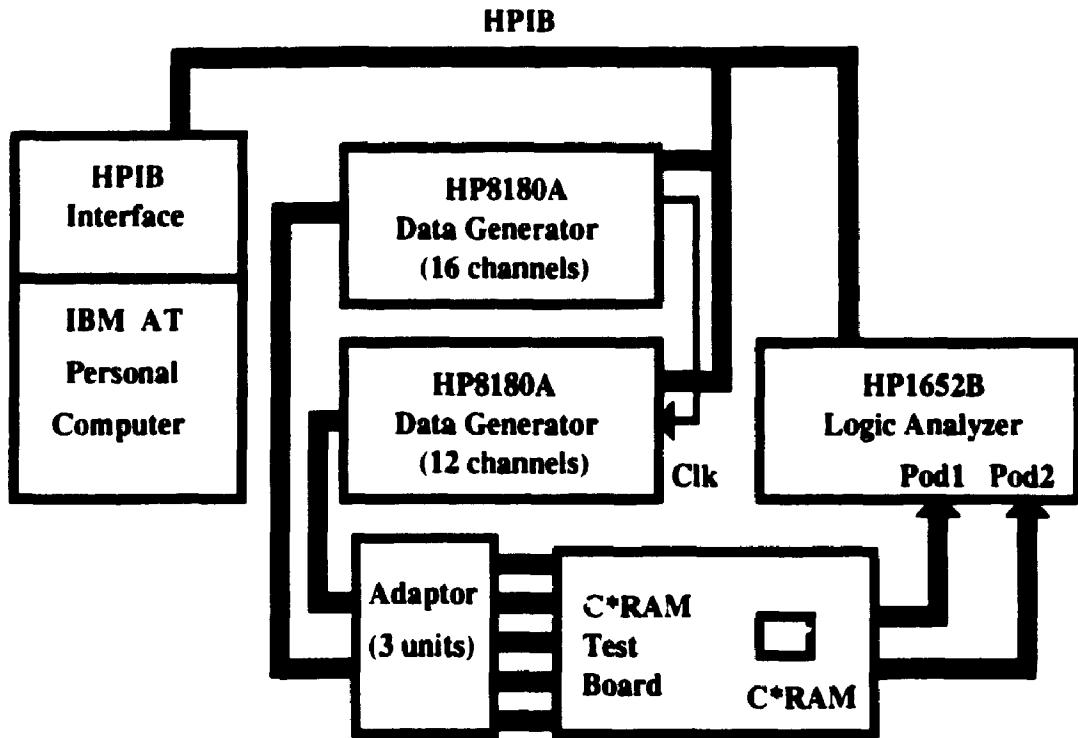


Figure 6.1 C*RAM test setup

6.1.1 C64p1k C*RAM Test Board

This is a custom built 4-layer printed circuit board that holds the C*RAM chip under test. The electrical schematic is shown in Appendix C. The board has five 2x8 header connectors to accommodate inputs from data generator(s), and two 2x20 header connectors for the logic analyzer pods, Figure 6.1. The data generator connectors are compatible with connectors from HP16520A/16521A pattern generator/extender, a pair of cards that mount into the HP16500 logic analyzer mainframe (available in the department). The pattern generator itself was not available at the design date, but it would constitute a more elegant and compact testing solution than using the two HP8081A's.

The board includes a 74245 bidirectional transceiver to implement the byte-wide bidirectional data bus. A second 74245 transceiver is used as a unidirectional buffer to power the C*RAM data

outputs.

6.2 Test Software

The 1000 line test program CRAMTest was developed using the BASICA (or GWBASIC) interpreter. Although this is a fairly primitive environment, it was available and adequate for writing a test program. The main disadvantage is the difficulty to modify source code since this sometimes requires line renumbering. Under the PC Windows environment though, this disadvantage is alleviated since the source code can be edited in a separate text editor window, outside the BASICA interpreter window (obviously, the source code has to be reloaded afterwards).

As a general idea, the test program is in fact a sketch of a C*RAM development system. The important restriction is that C*RAM programs cannot be emulated by this "sketch", it can only execute single instructions in an interactive manner. The program offers a menu of memory and C*RAM instructions, and executes the selected operation after the appropriate parameters have been entered. These parameters are translated in bit configurations sent to the data generators. Then, the data generators are activated and the C*RAM outputs are sampled by the logic analyzer, previously programmed for the appropriate trigger condition. The data sampled by the logic analyzer is read into the GPIB controller, where the test program selects the bit configurations and translates them into memory data.

The following operations can be executed within this environment:

1. Memory Write
2. Memory Read
3. C*RAM Operate-Write
4. Serial Data In (for Cs64p512)
5. Serial Data Out (Cs64p512)
6. Serial register store (Cs64p512)
7. Serial register load (Cs64p512)
8. Memory range test

6.2.1 C*RAM Testing with the CRAMTest Program

This section presents an example sequence of operations used to verify whether a C*RAM chip is functional. All feed-back from the chip is verified via memory read operations. The read values can be checked in two ways, on the logic analyzer screen, or as values returned by the program (which in turn reads them from the logic analyzer).

a. RAM verification: a sequence of memory write/read operations.

Example: the following sequence is written starting with address 100h (h denotes hexadecimal):

```
M(100h) <= 55h ; M(101h) <= AAh ; M(102h) <= 55h ; M(103h) <= AAh ;  
M(104h) <= 55h ; M(105h) <= AAh ; M(106h) <= 55h ; M(107h) <= AAh.
```

The sequence is then read back, verifying the memory block and the I/O Data Decoder. The data chosen, 55h and AAh, are both alternating sequence of "0" and "1". The goal is to detect short circuits between adjacent data lines, that could occur in the chip itself or in the test setup.

b. PE array and C*RAM support circuitry testing:

b1. The first step is to enable PE write-back to memory, by writing "1" in all W registers. This is done by executing a PE operation (Operate-Write) with the all-1 "1111_1111" opcode, the register destination W, and an arbitrary row address, except 32h.

b2. A memory complement operation can be exercised now, irrespective of the undefined contents of registers X and Y. First, a read operation is executed, from any location between 100h and 107h. This will bring the entire 64 bit logic row in the implicit M register (the register associated with the memory sense amplifiers). Then, an Operate-Write is executed, with the opcode M_COMPLEMENT=0101_0101, register destination "none" and row address destination 32h (i.e. the row starting at byte address 100h). A sequence of memory reads from address 100h to 107h should result now in the bit-complements of the bytes written at step a.: M(100h) = AAh; M(101h) = 55h, etc.

A thorough verification of the ALU and registers was done by iterating through the eight binary combinations of register values, and sending the power of 2 truth tables and their complements.

For example, setting $Y=0$, $X=1$, $M=1$ and sending $TTOP=2^3=(000)_1(00)$ should result in $ALU=1$, while operating on $TTOP=1111_0111$ should result in $ALU=0$.

Bus-tie testing was done by filling a 64-bit logic row with all-0 except one bit position in one of the 8 bytes, executing `M_IDENTITY` and Bus-tie followed by memory write, and verifying the all-1 64-bit result by reading out the eight FFh bytes.

6.2.2 Automatic Memory Testing

The test software provides the possibility of automatically testing a memory section. The option is invoked with the "M" entry in the main menu. The start and end testing address are required as parameters. Alternating byte values are used for successive write cycles. The two byte values can be modified by editing one line in the test program.

6.3 Test Results

To the date of this writing, four C64p1k chips have tested, focussing on the PE array. All four arrays proved functional, with the exception of one PE in one of the chips. Arbitrary memory locations tested during PE verification were all functional. A memory test loop was run on a 32 kb section (half the memory on-chip), and resulted in 86 bad locations. The results are not final because the test environment was not stable, being affected by two intermittent problems: the GPIB command interpretation delay mentioned above (discovered subsequently), and an imperfect clock link, due to impedance mismatch. Also, C*RAM testing at higher clock frequencies (above 8 MHz) is not feasible with the present setup. The availability of an HP16520A/16521A data generator module would eliminate these problems. Use of a third programmable data generator as a clock source is being investigated.

Two serial-access Cs64p512 C*RAM chips were also tested, using a different test board. A problem was discovered with the internal RD/WRB buffer/inverter layout, resulting in the impossibility to access the memory through the random access port. The serial port was used to write/read the memory core. The two PE arrays were functional, allowing functional verification of the Programmable Segments Bus-tie network.

Chapter 7

Proposed Developments and Conclusions

The Processing Element modifications discussed in Chapter 5 and implemented in the C512p512 C*RAM chip add flexible bit-parallel operation capabilities. They address thus the arithmetic weakness of the single bit PE. Section 7.1 will discuss one feature that might be implemented in future versions of C*RAM, especially versions based on high density SRAM/DRAM platforms where the column decoder circuitry is available for modification.

7.1 Independent PE Addressing

One often discussed problem with most SIMD machines is the “single-address” (or single-reference) aspect that hides behind the “multiple-data” characteristic. Concretely, these multiple data come from the same “parallel address”. In the case of current C*RAMs, this “parallel address” is a memory row address. The problem with the single address is that common software operations like indirect addressing or table look-up are not directly supported by the hardware: all processors are forced to work on data coming from the same address. Instead, the operations have to be synthesized in software and their execution time depends exponentially on the size of the problem: it takes $O(2^n)$ time to perform indirect addressing with an n -bit address [Tani91].

The NEC IMAP chip discussed in Chapter 2 is an exception to the single address problem. The IMAP 8-bit processors can contribute an 8-bit segment to their 12-bit local memory address,

hence they can independently of each other address data within a 256 word page. The disadvantage is the area overhead incurred to provide local addressing. The overhead comes from the necessity to replicate a row decoder block for each local memory. This is acceptable for the specialized IMAP chip where a 7000 transistor PE has a 32 kb local memory, but not for C*RAM where a 76 transistor PE has a 1-4 kb local memory and targets replacing memory at minimal cost.

7.1.1 Proposed Solution - Dual-Mode Column Decoder

The configuration proposed here tries to find a pragmatic mid-point solution, providing limited individual PE addressing space but without involving the overhead of extra row decoders. Obviously, the remaining option is to provide some form of PE control over the column decoder.

The SRAM-based C*RAM chips implemented in this work (Chapter 4) have a one-to-one correspondence between PEs and sense amplifiers (SAs). In between the memory core and the SAs, there is a column decoding level, selecting one bit out of four physical columns (for the particular modularity values used in the BATMOS C*RAMs), Figure 4.2. Thus, there appears the possibility that two registers in the PEs might control a section of the column decoder, Figure 7.1. This control would be multiplexed with the one exercised by the external address bits during standard RAM operation. The 2-of-4 decoding would be a minimal step compared to absence of local PE decoding.

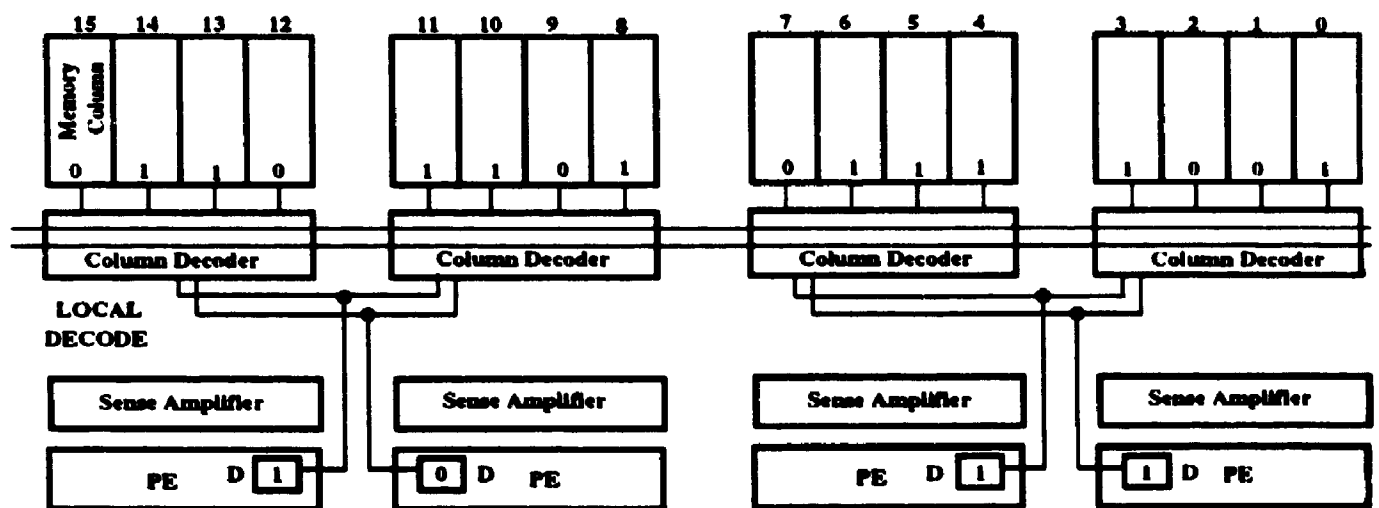


Figure 7.1 Column decoder controlled by PE registers in an SRAM C*RAM

Two variations can be imagined for the locally controlled column decoder: single PE control and multiple PE control. In the first version, all the registers needed to control local decoding are within one PE. In our case, two registers are needed to decode the four physical columns. The limitation of this alternative is in the small number of register afforded by one PE. The Baseline PE has only two local registers, insufficient to address anything above four columns. The Extended PE has six data registers, which is enough to cover up to 64 columns, but some registers might be allocated to hold local variables during program execution.

In the second approach, the local column decoder is controlled by a set of registers belonging to a number of PEs. This is the case represented in Figure 7.1. For example, the pair of Y registers in adjacent PEs controls two adjacent sections of the local decoder. The local addressing autonomy is restricted now to groups of two or multiple of two PEs. The advantage is the extensibility of the approach. If, for example, there are 16 physical columns behind a sense amplifier instead of 4, then groups of 4 Y registers will constitute the local address. There will be a logarithmic reduction in granularity, with 4 PEs being the local addressing increment instead of the previous 2.

Assuming next that we discuss a DRAM-based C*RAM (see also the DRAM section in Chapter 2), we would have a much smaller column pitch, e.g. 2 μm or less [Prin91]. The DRAM SA has the same pitch as the column circuitry. At the same time, the PE itself will decrease somewhat in size, but much less than the column pitch or SA. All these will result in a PE layout that fits in the pitch of a number of columns, for example 8, 16 or 32, as opposed to the one-to-one correspondence achievable in SRAM C*RAM. The DRAM C*RAM hence imposes modifications of the column decoder from the beginning, in order to match the single bit PE interface to the 8/16/32 lines coming from the SAa within one PE pitch. Local column decoding can be thus designed in together with the C*RAM adapted column decoder.

7.2 Future C*RAM Work

The following is a sketch of possible future stages of the C*RAM project.

7.2.1 Processing Element and C*RAM Architecture

- Investigate faster bit-parallel PE architectures: the current Extended PE propagates the carry for every partial product in a bit-parallel multiplications procedure. A faster solution could be performing the intermediary additions in carry-save format and propagating the carry only after the last addition. The speed advantage is bigger for longer words, because the longer ripple-carry cycles are eliminated. A seventh register with access to the ALU would be needed to hold the second bit of the carry-save digit. Modifications might also be required in the local communication scheme, to account for the carry-save carry single-position shift, and in the register-ALU configuration, to make the new register contents available to the ALU.
- Investigate floating point operation support. Floating point (FP) add appears to be more difficult than FP multiply, because different operands have to be shifted by different amounts in preparation for addition, contravening thus to the single-instruction model. The PE could be extended to provide a means to inhibit shift operations when a certain local condition is met.
- Explore associative RAM access. The PE information could be used to specify the data read from memory, as an alternative to (or in combination with) conventional RAM addressing.
- C*RAM program stored within the chip itself. The C*RAM operation code is currently multiplexed on the external data bus (in the optimal case). It could be sourced by the internal data bus, as a result of one or more memory reads from a program area. Short, continuous program loops could be thus executed without the intervention of the C*RAM controller. Moreover, multiple C*RAM chips could execute independent programs, combining into a MIMD or SIMDs. As a note, the 512 PE C512mp512 chip has come close to providing this capability, the required multiplexing stage being designed in, but not the timing.

7.2.2 C*RAM System Hardware

- Build a C*RAM system using the BATMOS C*RAMs. Its probable form will be a personal computer add-on card, with eight C64p1k chips. A PC bus will have to be selected. A conservative choice is suggested for the initial work, very likely the popular but dated 16-bit/8.33 MHz

ISA bus. Next step should be a performance card using a variation of the 32-bit/33 MHz PCI (Peripheral Component Interconnect) bus, a standard with confirmed growing popularity.

- As part of the previous: design the C*RAM controller. Suggested implementation is a high-density Field Programmable Gate Array (FPGA, e.g. the XILINX 4000 series). Stages of the design would include writing and simulating a VHDL model of the controller. The first design iteration should be capable of reading C*RAM instructions from a program memory and send them to the C*RAM bank with the appropriate timing. This controller should also be able to branch the program according the wired-OR result read from the C*RAM bank. In a second iteration, the controller should provide a high-speed link between an I/O port (for example, a video port) and the C*RAM bank.

- In parallel, or as a separate project, the Processor-in-Memory (PIM) chips to be produced by National Semiconductor for Cray Computer (section 2.2.2) and the NEC IMAP (section 2.2.3) chips should be explored, depending on their availability. As discussed in Chapter 2, the PIM device is closer to the C*RAM concept as a general-purpose SIMD building block, but oriented towards fewer PEs per chip with larger local memories. The IMAP chip is more specialized, targeting real-time image processing and adopting a 50% PE - 50% memory balance. A study should be conducted in order to determine the category of applications best suited for each approach.

7.2.3 C*RAM Software

- First stage is to write the system development software, including a microassembler (pure C*RAM instructions, i.e. TTOP+COP), an assembler - C*RAM plus controller instructions, and eventually a high level language compiler. This work has been started at Carleton University by Arthur Castonguay. As a result of the growing worldwide interest in the data-parallel programming model, a standardization effort is in progress to define an appropriate high level language, called Data Parallel C Extensions (DPCE). Hence, the C*RAM software work stands a good chance to converge with a stable high-level language standard.

- In parallel with the assembler development, a C*RAM software emulator should be made available in order to allow applications work to evolve in the absence of hardware. The emulator

should provide a complete visualization of the PE array and memory. A visual feed-back of memory data layout is important because of the interaction between computation and physical data location.

7.3 Conclusions

The work presented in this document advances the Computational RAM project in two directions.

First, a 64 Processing Element, 64 kb Computational RAM (named C64p1k) has been designed and implemented in a 0.8 μm BiCMOS technology (BATMOS). Benefitting from the availability of an industrial grade memory module designed at Bell-Northern Research, this work has focussed on the circuit and physical design of the Processing Element array (Chapter 3). The result was a very dense array, occupying less than 3% of the memory area, and capable of functioning at speeds of 75 MHz (as shown by simulations). A number of C64p1k chips were tested and proven functional (Chapter 6). The availability of these functional C*RAM chips opens up the possibility of building a C*RAM subsystem, integrating it within a workstation or personal computer and demonstrating support software and applications.

Secondly, by analyzing recognized weaknesses in the pure bit-serial approach on which the Baseline PE relies, this work introduces the bit-parallel oriented C*RAM (Chapter 5). The extensions brought to the BPE make the PE array capable of performing bit-parallel integer multiplication without intermediate memory access, resulting in fast and low-power operation. This original Extended PE is two times bigger than the Baseline PE and is claimed to be the minimal hardware structure that can support bit-parallel multiply in SIMD manner with flexible word length. The new PE structure was designed into a 512 PE BATMOS C*RAM (C512mp512), currently in fabrication, the first SIMD-memory integrated hybrid to provide both bit-serial and bit-parallel computation with a minimum of hardware added to the memory.

The name Parallel Computational RAM (PC*RAM) is proposed for this class of dual-mode, bit-serial and bit-parallel SIMD-memory hybrid, as the author expects it to become a general-purpose storage-computing devices category.

Bibliography

- [Alma89] George S. Almasi and Allan Gottlieb, "Highly Parallel Computing", Benjamin/Cummings, 1989, ISBN 0-8053-0177-1
- [BNRM92] Allan L. Silburt *et al.*, "A 200 MHz 0.8 μm BiCMOS Modular Memory Family of DRAM and Multiport SRAM", *Proceedings of the IEEE 1992 Custom Integrated Circuits Conference*, Boston, May 3-6, 1992
- [BNRM93] Allan L. Silburt *et al.*, "A 180-MHz 0.8 μm BiCMOS Modular Memory Family of DRAM and Multiport SRAM", *IEEE Journal of Solid-State Circuits*, Vol. 28, No. 3, March 1993
- [Cast94] Arthur Castonguay and Christian Cojocaru, "C*RAM Micro-Assembly Language, v0.7", 1994, Technical Report CUDOE-HSL-94-03
- [Chri94] Lauren Christopher, "HDTV Basics and ASICs", Educational Session, *Custom Integrated Circuits Conference 1994*
- [EET816] "Contract lets them develop unusual Processor-in-Memory Technology. NSA taps Cray Computer, National", *Electronic Engineering Times*, Issue 816, Sept. 26, 1994, pp. 39-40.
- [EET813] "Samsung claims 256-Mbit DRAM", *Electronic Engineering Times*, Issue 813.

Sept. 5, 1994, p. 2.

- [Elli92] Duncan G. Elliott, W.M. Snelgrove and M. Stumm, "A Memory-SIMD Hybrid and Its Application to DSP", *Proceedings of the IEEE 1992 Custom Integrated Circuits Conference*, Boston, May3-6, 1992
- [Elli92] Duncan G. Elliott, "Computational RAM", Private Presentation
- [Flynn66] M.J. Flynn, "Very high-speed computing systems", *Proceedings IEEE* 54:12 (December), pp. 1901-1909
- [Gokh92] Maya Gokhale *et al.*, "A Massively Parallel Processor-In-Memory Array and its Programming Environment", Technical Report SRC-TR-92-076, Nov. 1992
- [Hada91] R. Hadaway, *et al.*, "A Sub-micron BiCMOS Technology for Telecommunications", *Journal of Microelectronics Engineering*, Vol.15, pp. 513-516, 1991
- [Hord90] Michael R. Hord, "Parallel Supercomputing in SIMD Architectures", CRC Press, 1990, ISBN 0-8493-4271-6
- [IBMV94] Private meeting with the IBM Dram Design Group, Essex Junction, Vermont, 1994
- [IESp89] "Supercomputer experts predict expansive growth", *IEEE Spectrum*, February 1989, pp. 26-33
- [IESp93] Linda Geppert, "Not your father's CPU", *IEEE Spectrum*, December 1993, pp. 20-25
- [Lipo90] G.J. Lipovsky, "Dynamic Systolic Associative Memory Chip", *Proceedings of the International Conference on Application Specific Array Processors*, 1990
- [Louc80] W.M. Loucks, W.M. Snelgrove and S.G. Zaky, "VASTOR: A Microprocessor

Based Associative Vector Processor for Small Scale Applications". *Proceedings of the 1980 International Conference on Parallel Processing*

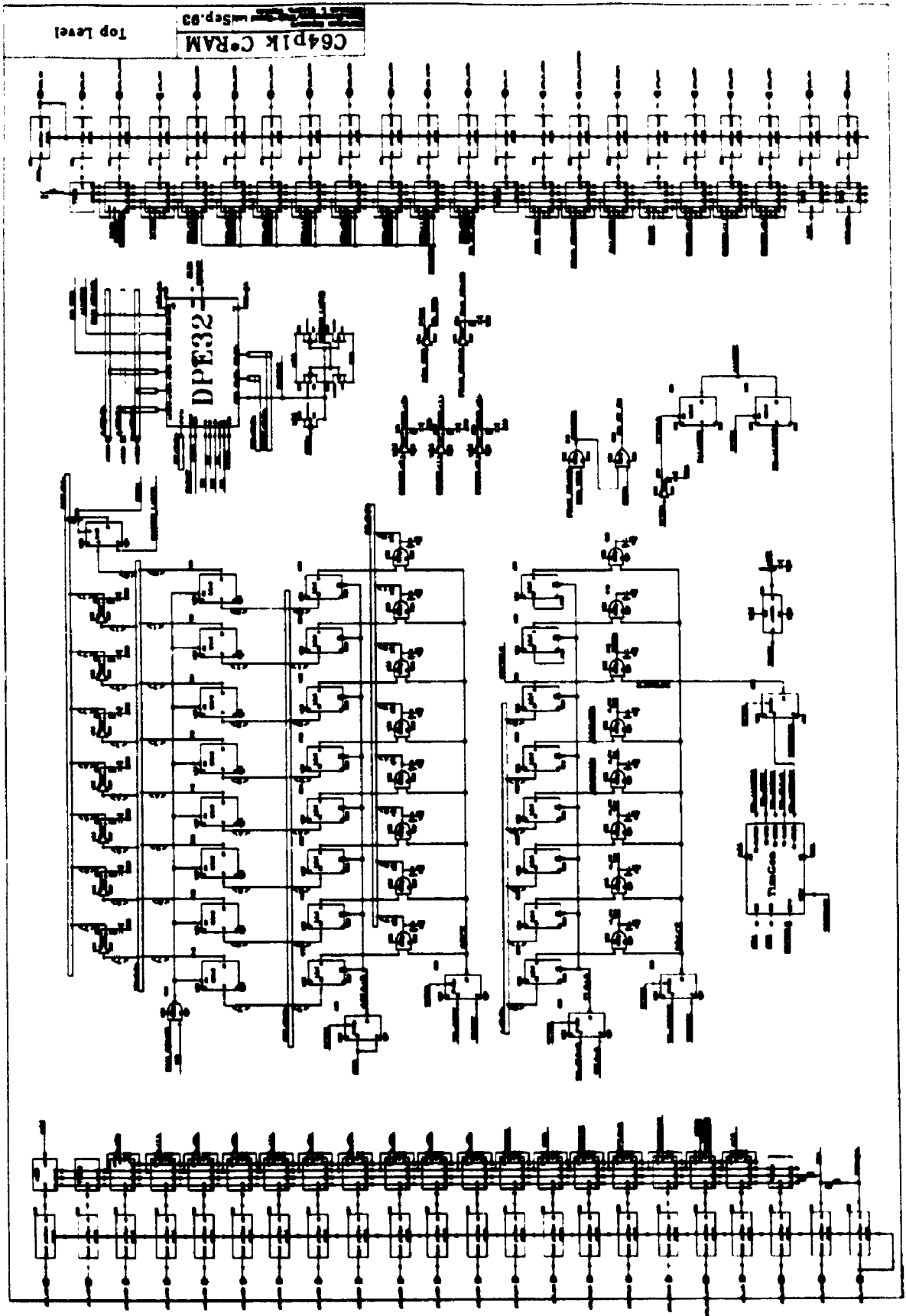
- [Mats87] Masataka Matsui *et al.*, "A 25-ns 1-Mbit CMOS SRAM with Loading-Free Bit Lines", *IEEE Journal of Solid-State Circuits*, Vol. SC-22, No. 5, October 1987
- [Patt94] David A. Patterson and John L. Hennessy, "Computer Organization & Design. The Hardware/Software Interface", Morgan Kaufmann Publishers, 1994, ISBN 1-55860-281-X
- [PRI491] Proceedings of the IEEE, Special Issue on Massively Parallel Computers, April 1991
- [Prin91] Betty Prince, "Semiconductor Memories, A Handbook of Design, Manufacture and Applications", Second Edition, John Wiley & Sons Ltd., 1991, ISBN 0 471 92465 2
- [Schu93] Joseph Schutz, "Low Power Design Techniques for Microprocessors", *1993 IEDM Short Course on Low Voltage and Low Power Devices*
- [Smit88] Lorenz A. Schmitt and Wilson S. Stephen, "The AIS-5000 Parallel Processor", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 3, May 1988
- [Suzu93] M. Suzuki *et al.*, "A 1.5-ns 32-b CMOS ALU in Double Pass-Transistor Logic", *IEEE Journal of Solid-State Circuits*, Vol. 28, No. 11, November 1993
- [Taka93] Takada Masahide, "Low Power Memory Design", Short Course Program, *IEDM 1993*
- [Tani91] Steven L. Tanimoto, "Memory Systems for Highly Parallel Computers", *Proceedings of the IEEE*, April 1991, pp. 403-415

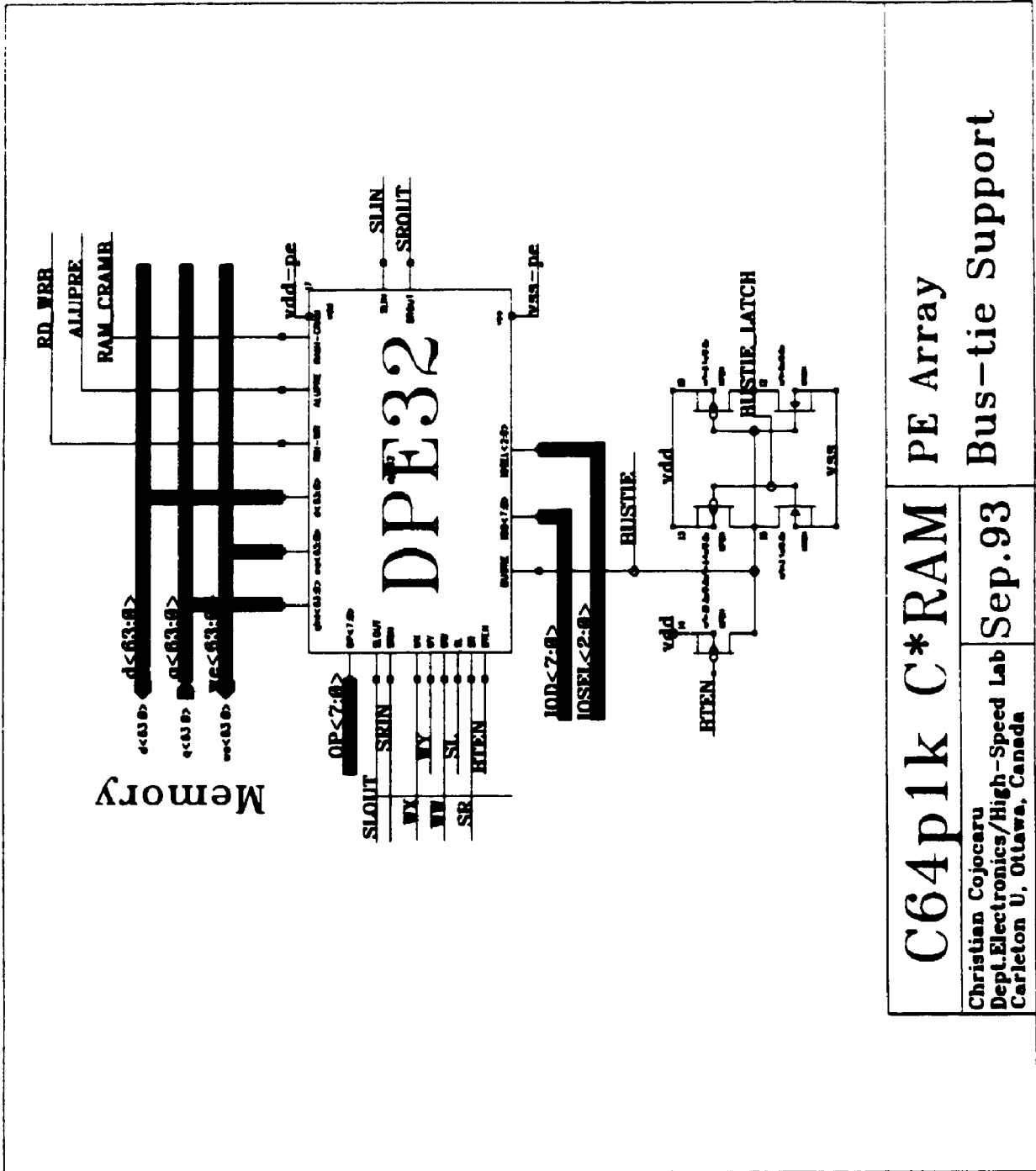
- [Wada87] Tomohisa Wada *et al.*, "A 34-ns 1-Mbit CMOS SRAM Using Triple Polysilicon", *IEEE Journal of Solid-State Circuits*, Vol. SC-22, No. 5, October 1987
- [West93] Neil H.E. Weste and Kamran Eshraghian, "Principles of CMOS VLSI Design, A Systems Perspective", Second Edition, Addison-Wesley, 1993, ISBN 0-201-52276-6
- [Yama94] N. Yamashita *et al.*, "A 3.84 GIPS Integrated Memory Array Processor LSI with 64 Processing Elements and 2 Mb SRAM", *1994 IEEE International Solid-State Circuits Conference*, paper FA 15.2, pp. 260-261

Appendix A

C64p1k C*RAM Schematics

- 143 Top Level
- 144 PE Array and Bus-tie Support
- 145 PE Array Connections (two PEs)
- 146 Left PE
- 147 Right PE
- 148 TTOP Register and Drivers
- 149 COP Register and Drivers
- 150 I/O Data Buffers and D<0>/BTOUT Multiplexing
- 151 ALUPRE Mux & Driver / Data Out Enable / Control Drivers
- 152 Timing / Memory Clock Driver
- 153 Data Bus Pads (top right pads)
- 154 Control / IOS / MCK Pads (bottom right pads)
- 155 Address Bus Pads (top left pads)
- 156 Timing Pads (bottom left pads)

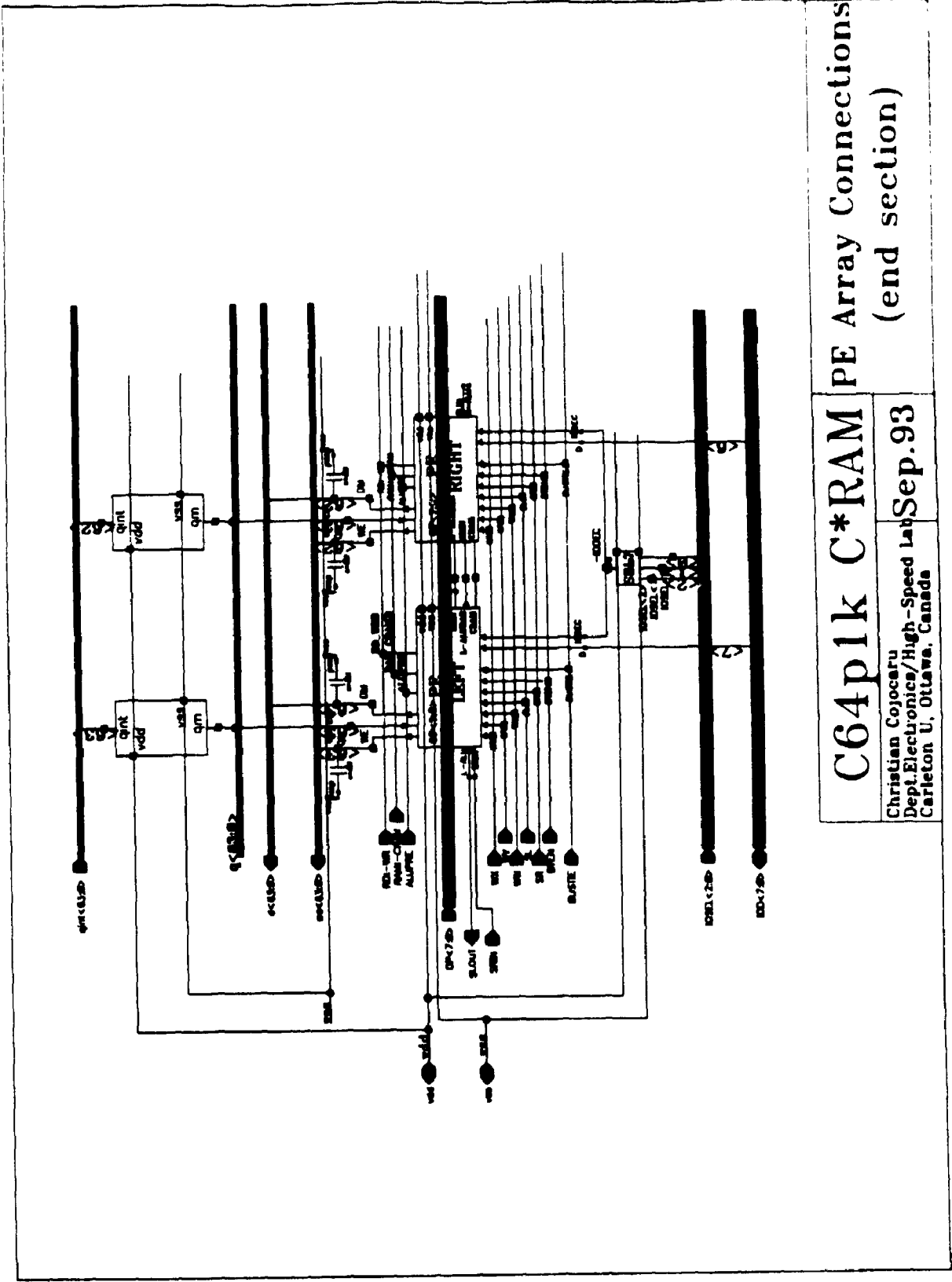




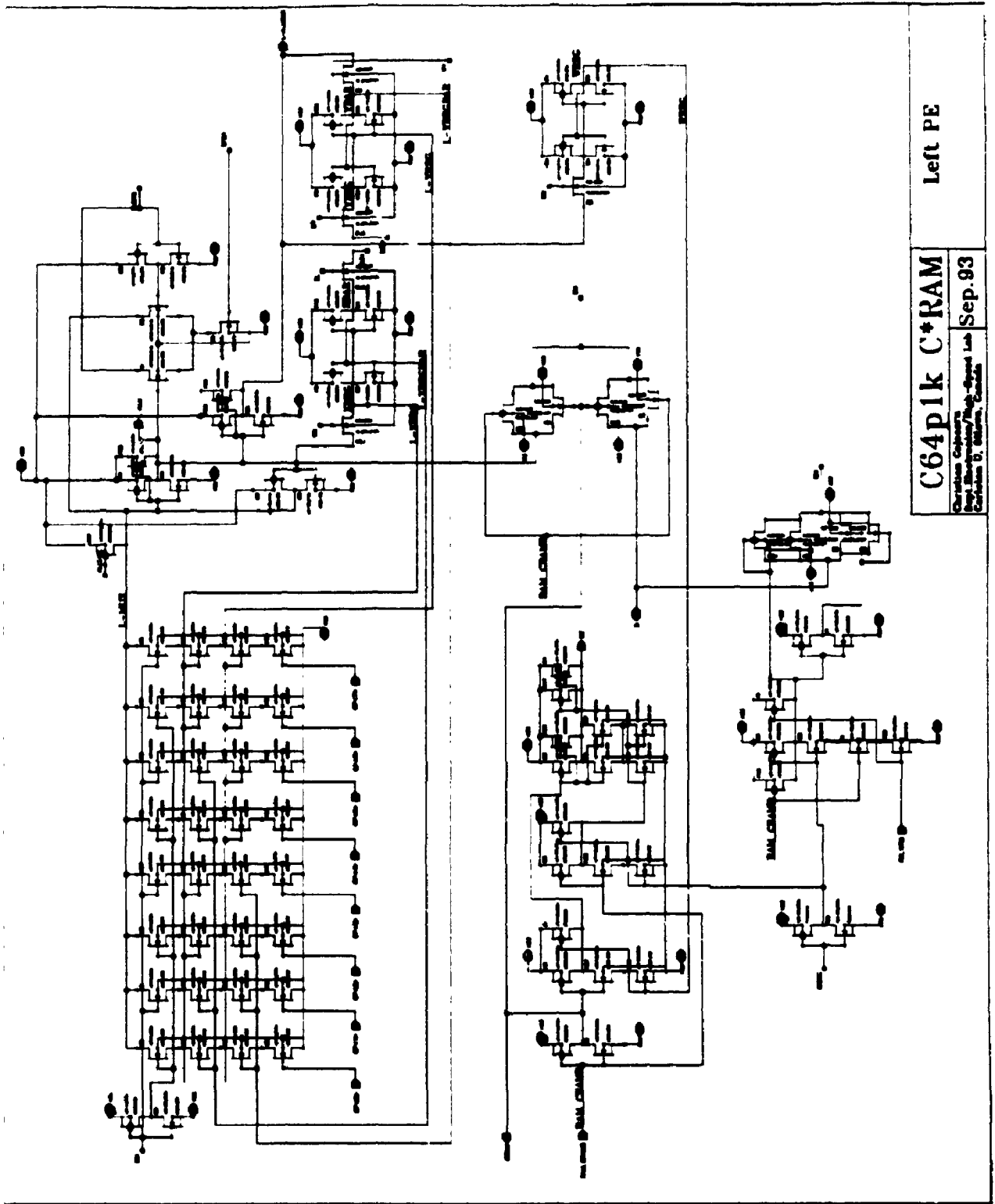
C64p1k C*RAM PE Array
 Bus-tie Support

Christian Cojocaru
 Dept. Electronics/High-Speed Lab
 Carleton U., Ottawa, Canada

Sep.93



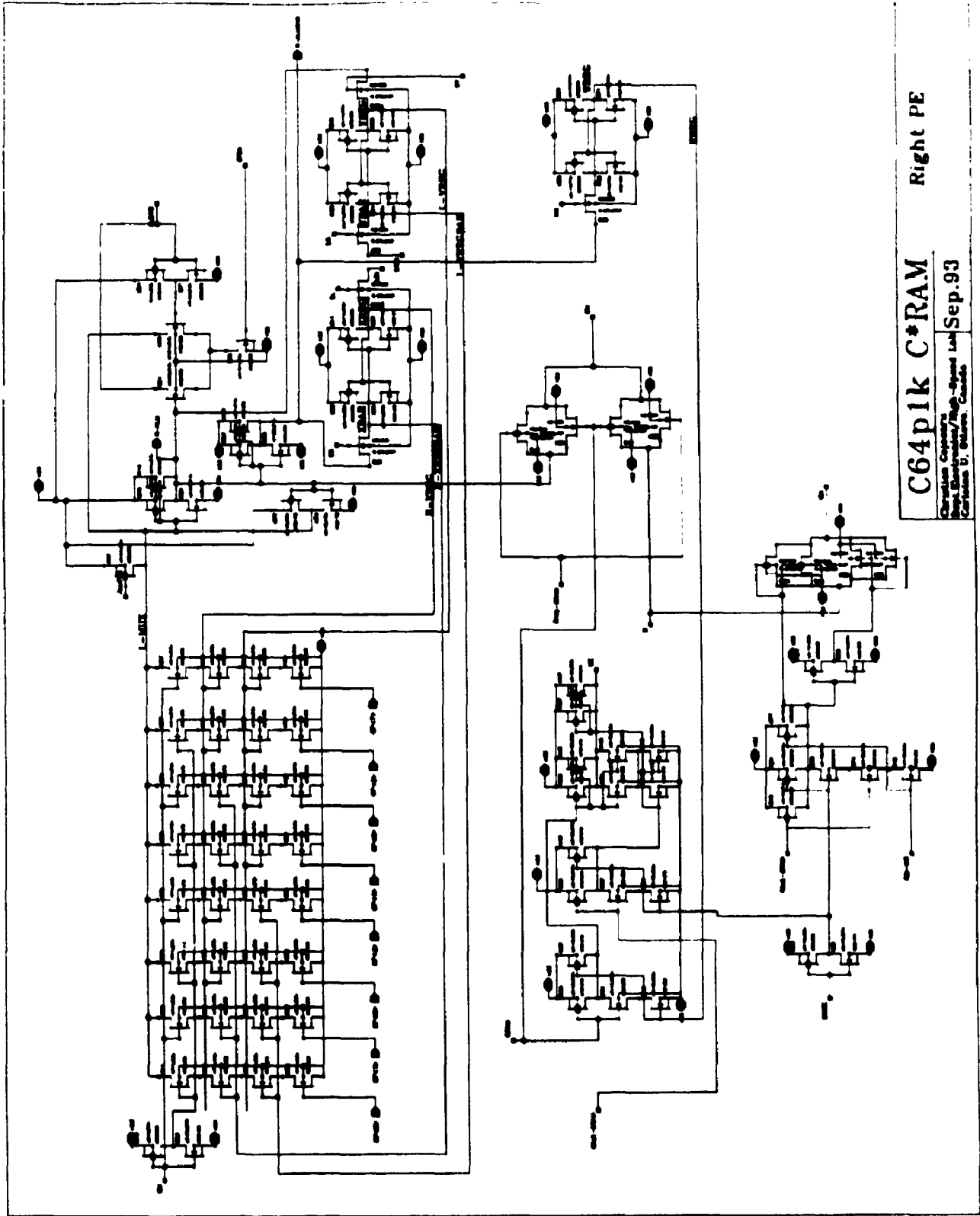
C64p1k C*RAM PE Array Connections (end section)
Christian Cojocaru Dept. Electronics/High-Speed Lab Carleton U., Ottawa, Canada



Left PE

C64p1k C*RAM

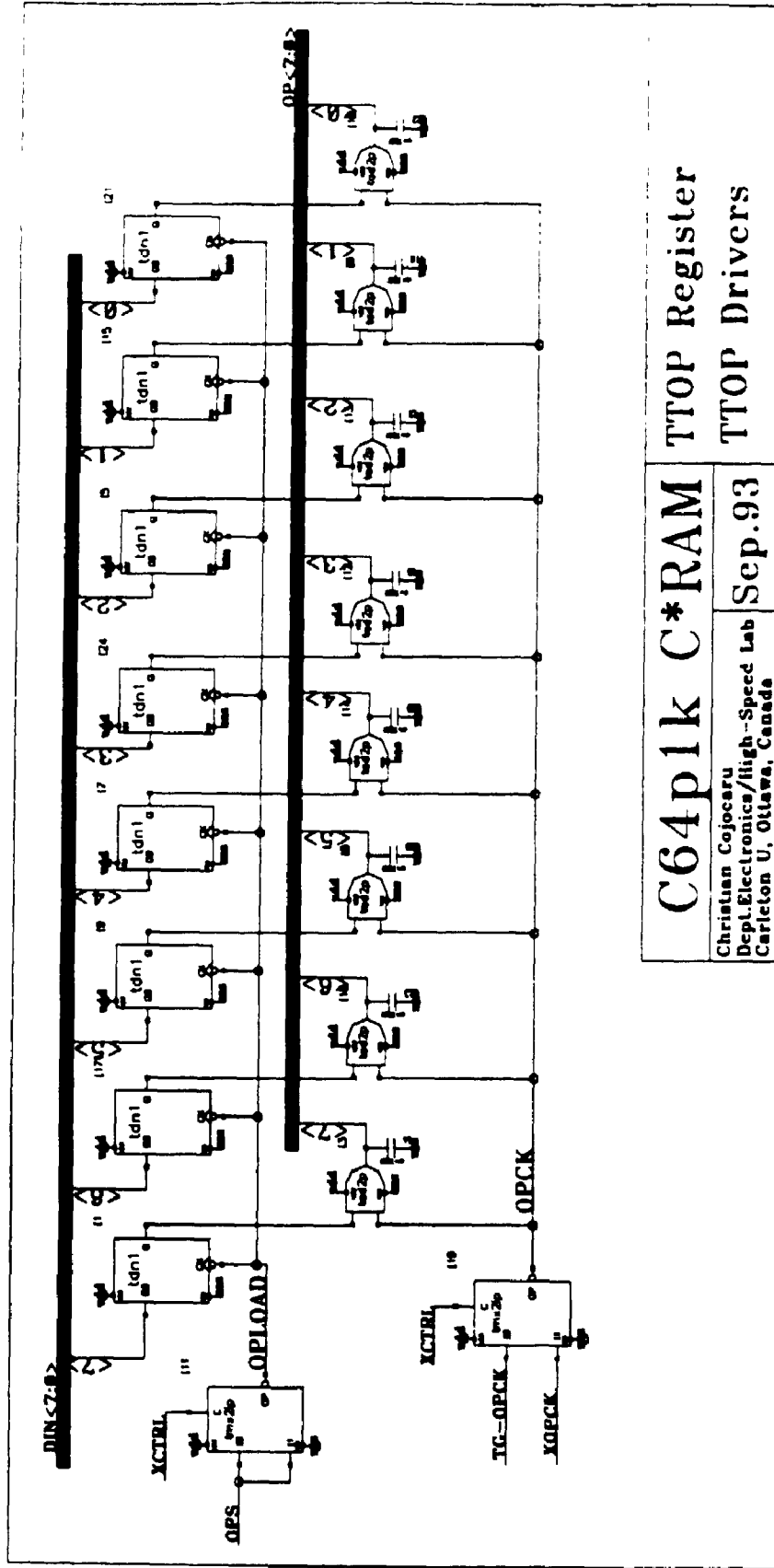
Charles Colwell
Digital Illustration/High-Speed Lab
Sep. 93
Carlson D. Walker, Canada



Right PE

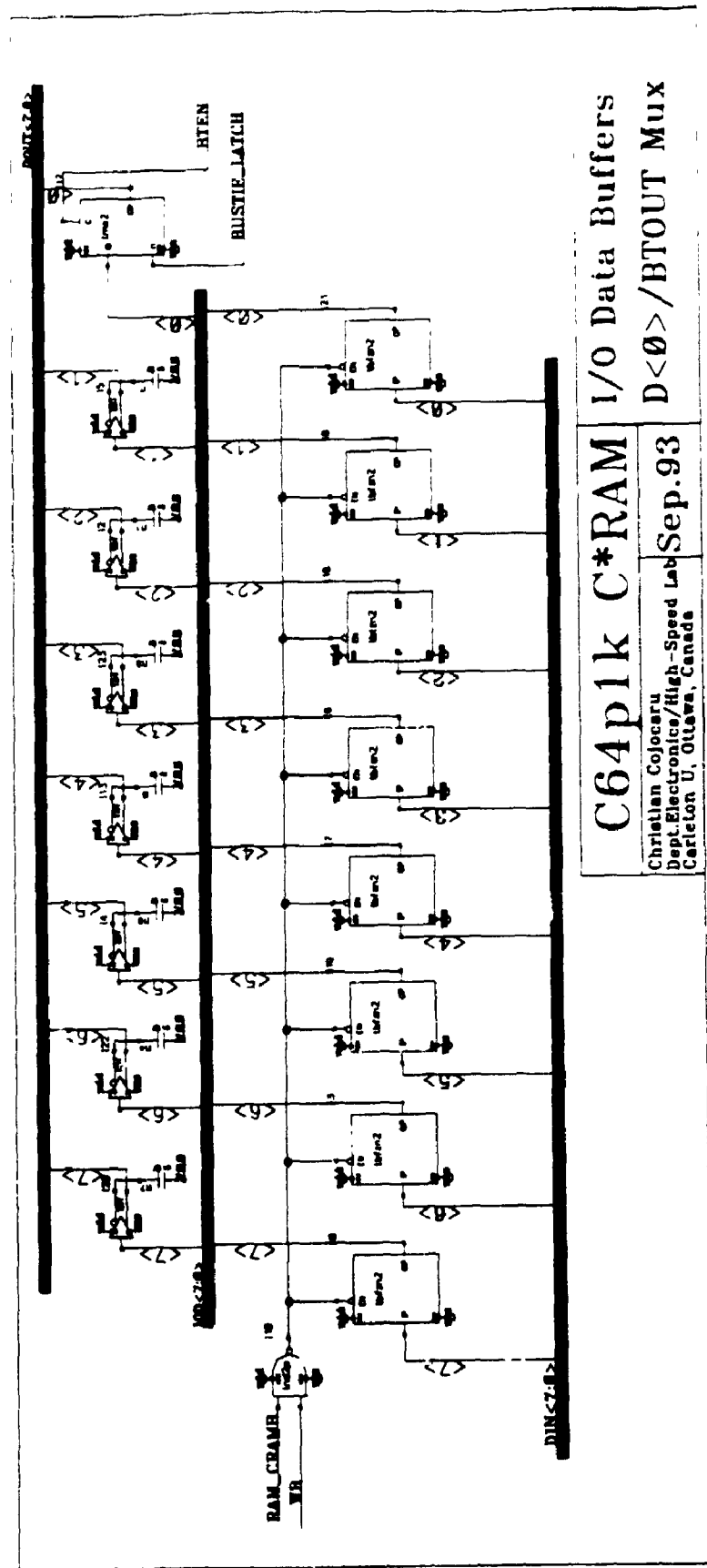
C64p1k C*RAM

Copyright © 1983
Shugart Associates, Inc.
All Rights Reserved
Sep. 93



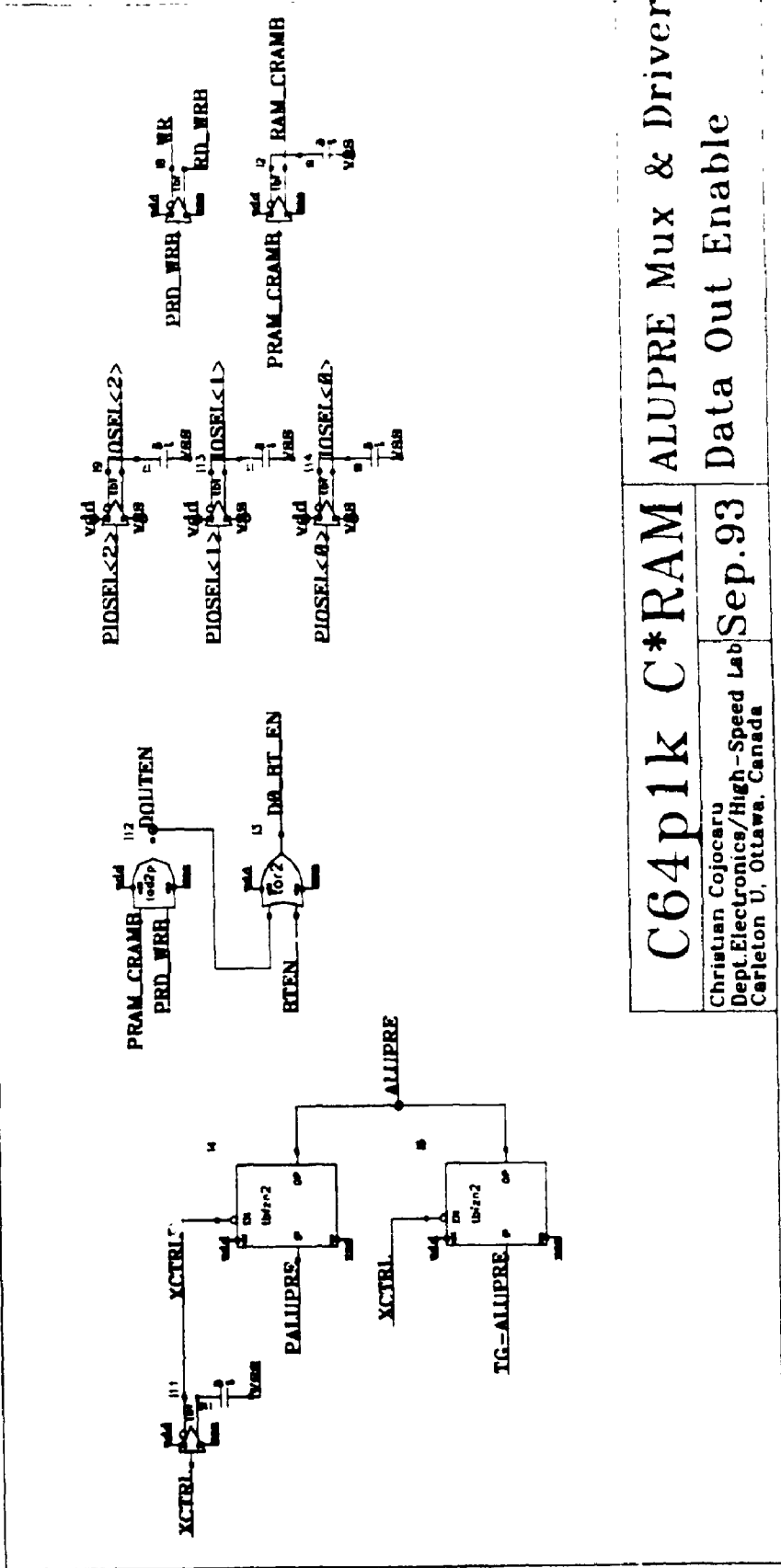
C64p1k C*RAM TTOP Register
TTOP Drivers

Christian Cojocaru
Dept. Electronics/High-Speed Lab
Carleton U. Ottawa, Canada
Sep.93



C64p1k C*RAM I/O Data Buffers
 D<0>/BTOUT Mux

Christian Cojocaru
 Dept. Electronics/High-Speed Lab
 Carleton U., Ottawa, Canada
 Sep.93



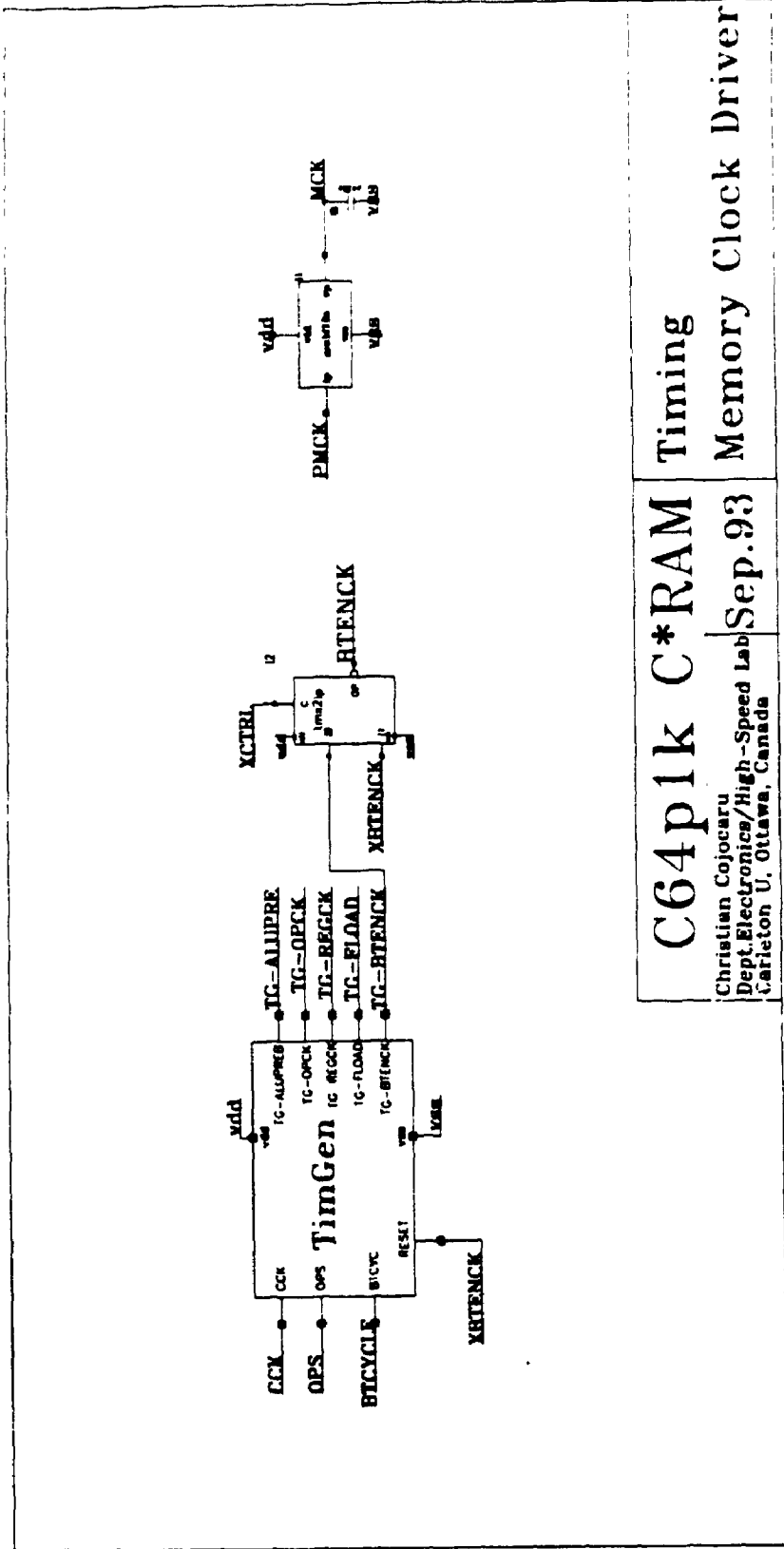
C64p1k C*RAM

ALUPRE Mux & Driver

Christian Cojocaru
 Dept. Electronics/High-Speed Lab
 Carleton U, Ottawa, Canada

Data Out Enable

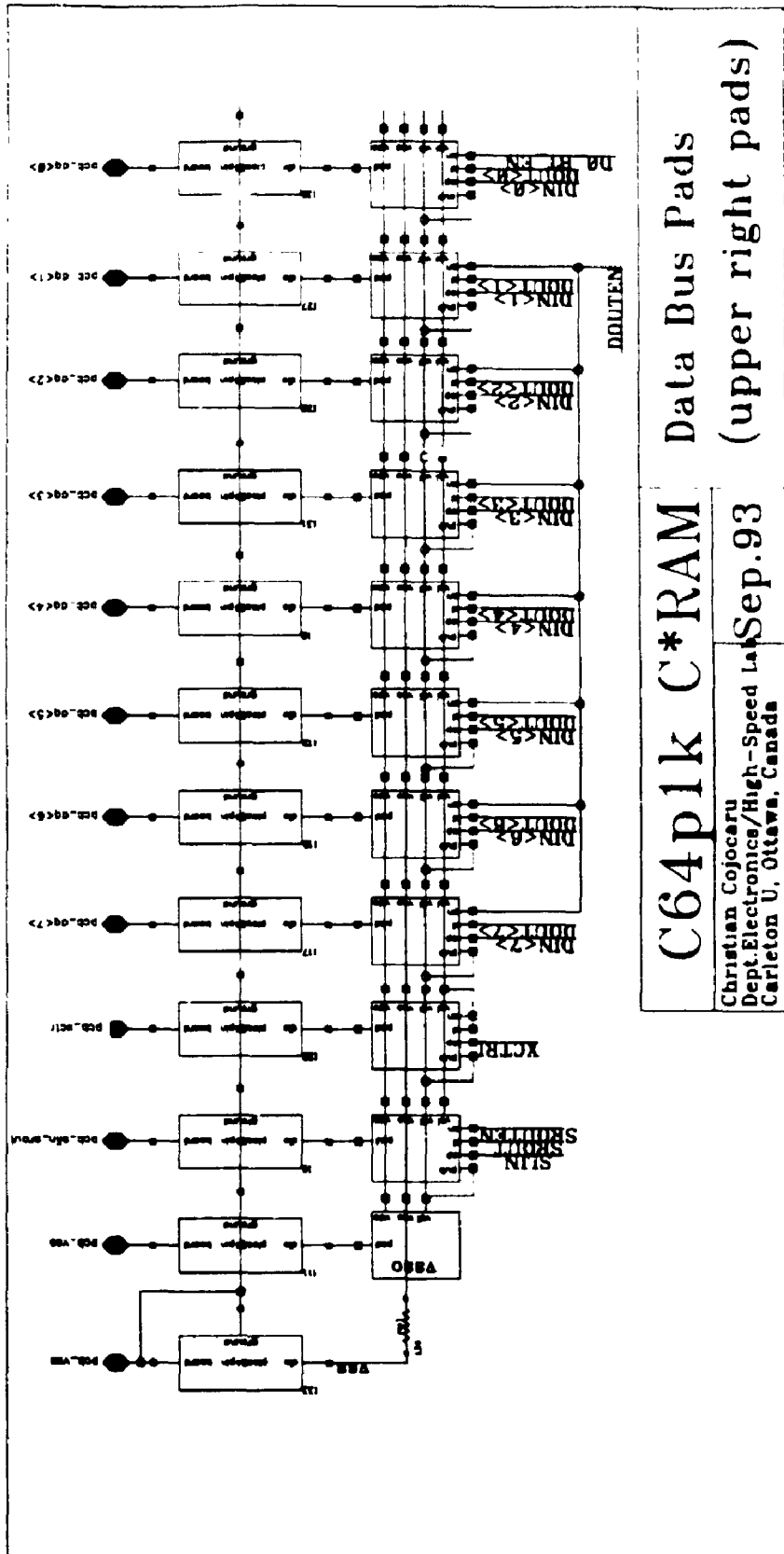
Sep.93



C64p1k C*RAM

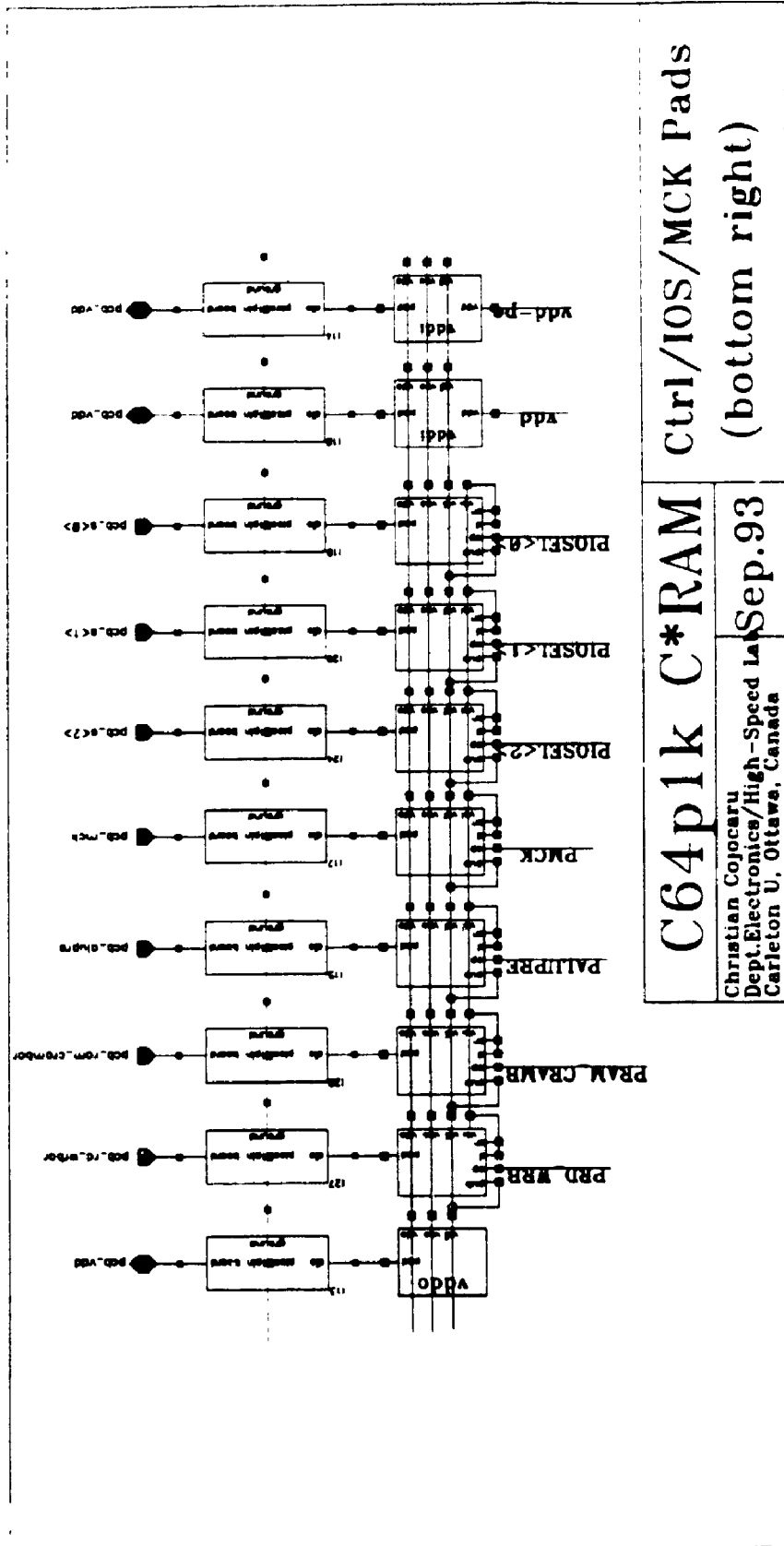
Christian Cojocaru
 Dept. Electronics/High-Speed Lab
 Carleton U., Ottawa, Canada

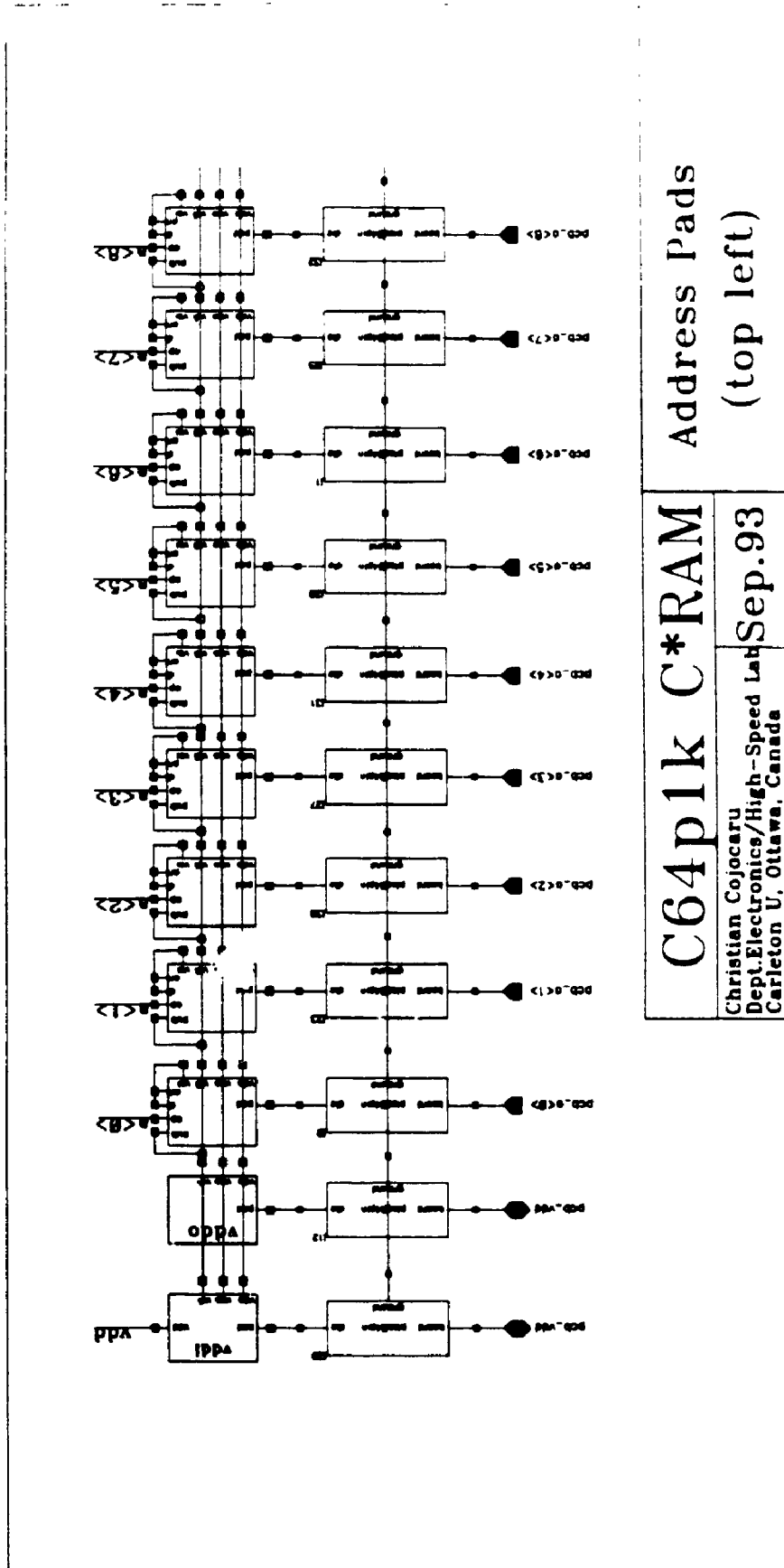
Timing
 Memory Clock Driver



C64p1k C*RAM Data Bus Pads (upper right pads)

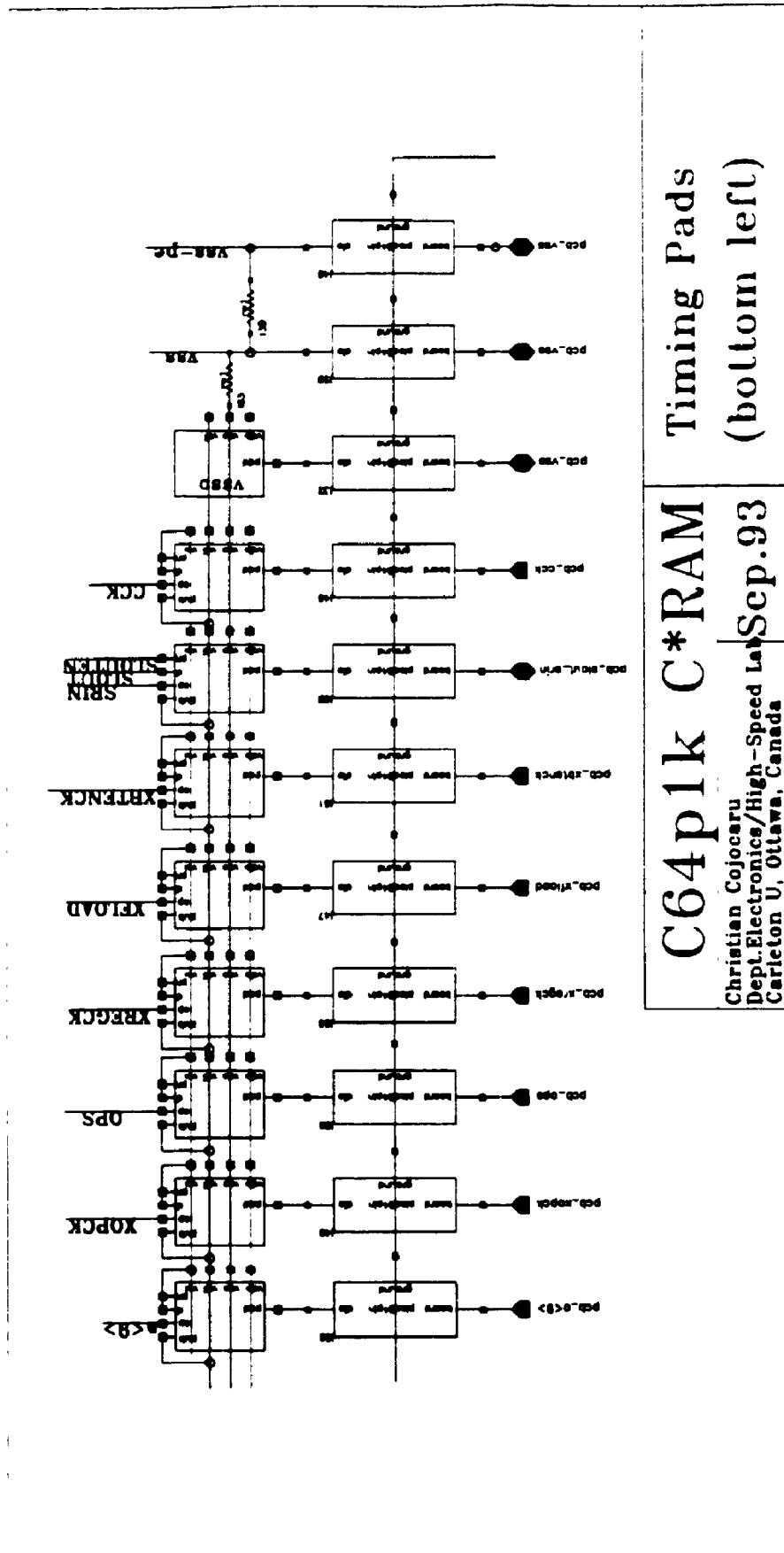
Christian Cojocaru
Dept. Electronics/High-Speed Lab
Carleton U., Ottawa, Canada





C64p1k C*RAM Address Pads
(top left)

Christian Cojocaru
Dept. Electronics/High-Speed Lab
Carleton U. Ottawa, Canada
Sep.93

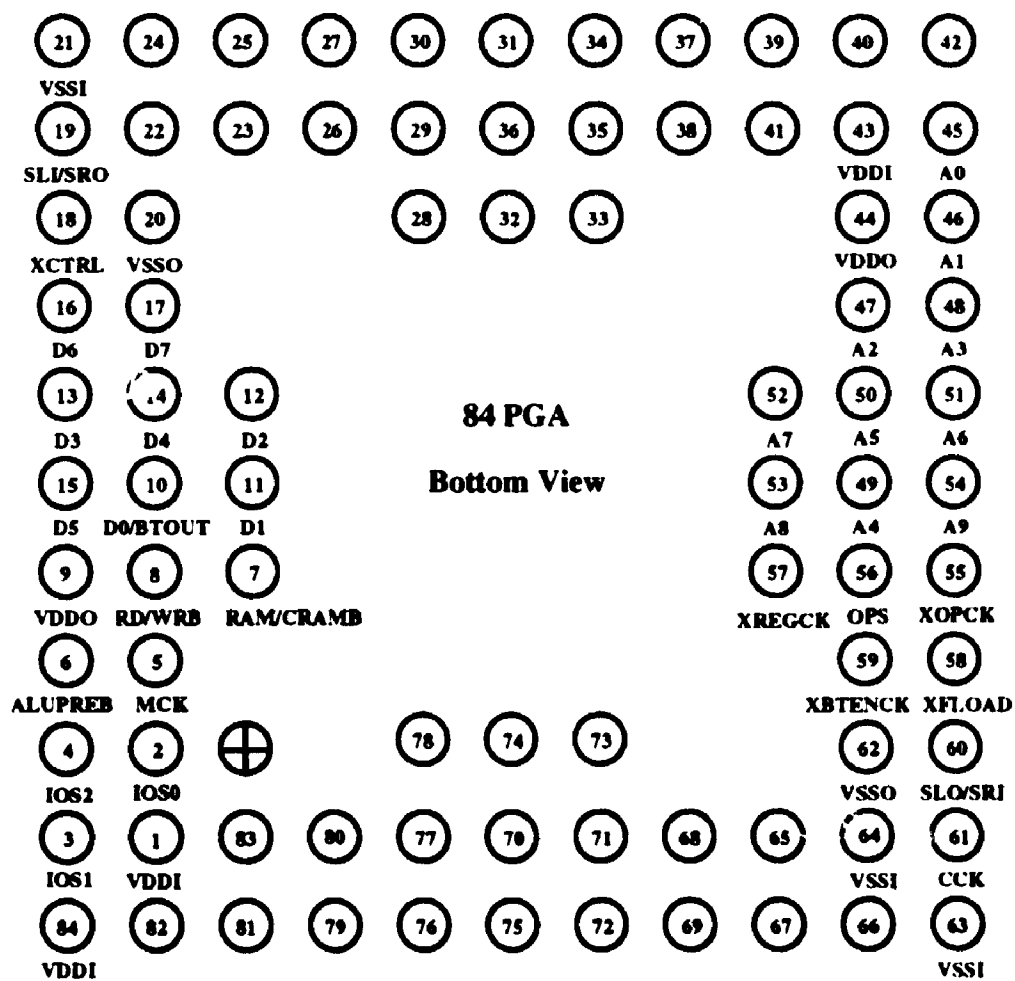


C64p1k C*RAM
Timing Pads
(bottom left)

Christian Cojocaru
Dept. Electronics/High-Speed Lab
Carleton U., Ottawa, Canada

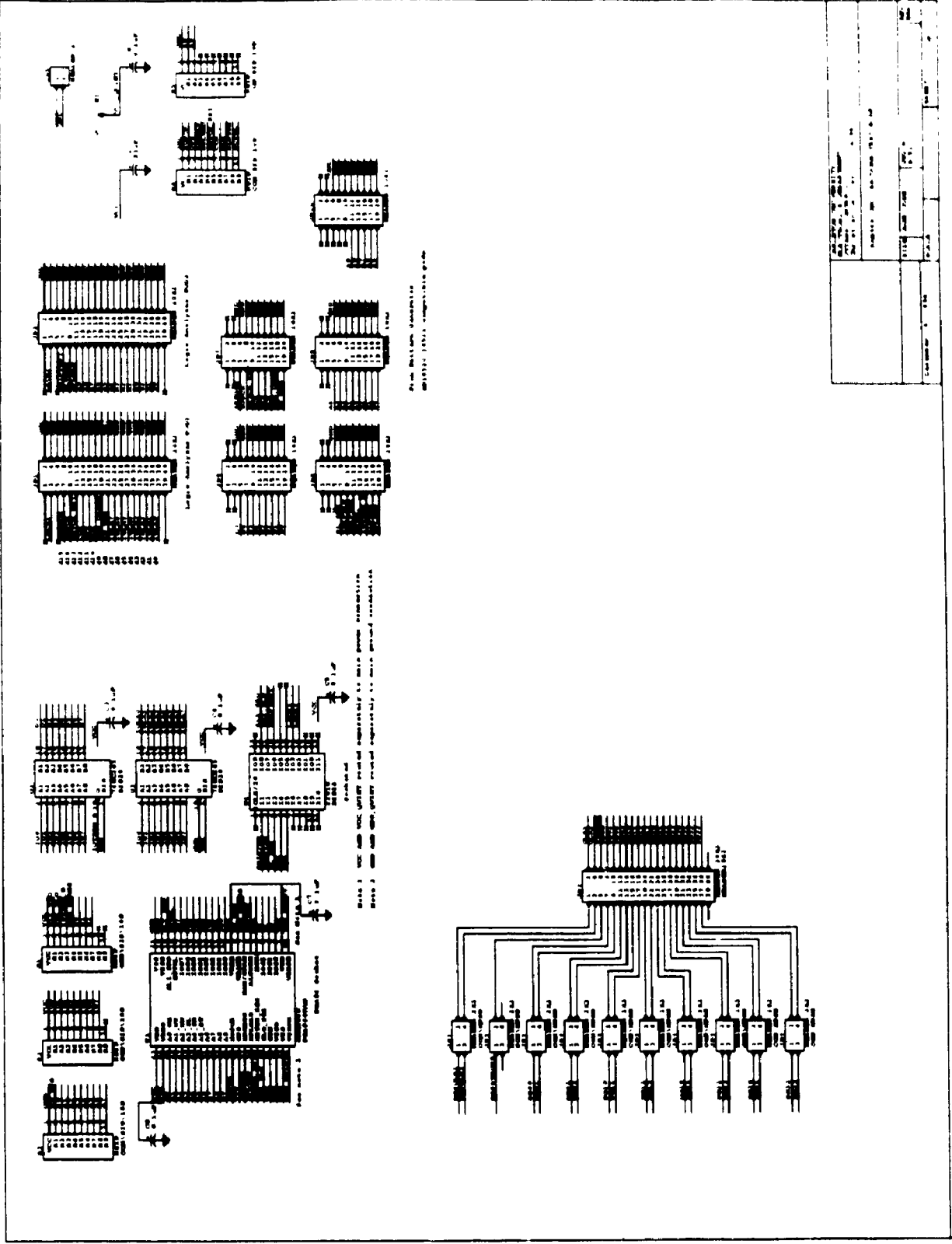
Appendix B

C64p1k C*RAM Pinout



Appendix C

C64p1k Test Board



REV. 1	1964	100-1000	100-1000
REV. 2	1965	100-1000	100-1000
REV. 3	1966	100-1000	100-1000
REV. 4	1967	100-1000	100-1000
REV. 5	1968	100-1000	100-1000
REV. 6	1969	100-1000	100-1000
REV. 7	1970	100-1000	100-1000
REV. 8	1971	100-1000	100-1000
REV. 9	1972	100-1000	100-1000
REV. 10	1973	100-1000	100-1000
REV. 11	1974	100-1000	100-1000
REV. 12	1975	100-1000	100-1000
REV. 13	1976	100-1000	100-1000
REV. 14	1977	100-1000	100-1000
REV. 15	1978	100-1000	100-1000
REV. 16	1979	100-1000	100-1000
REV. 17	1980	100-1000	100-1000
REV. 18	1981	100-1000	100-1000
REV. 19	1982	100-1000	100-1000
REV. 20	1983	100-1000	100-1000
REV. 21	1984	100-1000	100-1000
REV. 22	1985	100-1000	100-1000
REV. 23	1986	100-1000	100-1000
REV. 24	1987	100-1000	100-1000
REV. 25	1988	100-1000	100-1000
REV. 26	1989	100-1000	100-1000
REV. 27	1990	100-1000	100-1000
REV. 28	1991	100-1000	100-1000
REV. 29	1992	100-1000	100-1000
REV. 30	1993	100-1000	100-1000
REV. 31	1994	100-1000	100-1000
REV. 32	1995	100-1000	100-1000
REV. 33	1996	100-1000	100-1000
REV. 34	1997	100-1000	100-1000
REV. 35	1998	100-1000	100-1000
REV. 36	1999	100-1000	100-1000
REV. 37	2000	100-1000	100-1000
REV. 38	2001	100-1000	100-1000
REV. 39	2002	100-1000	100-1000
REV. 40	2003	100-1000	100-1000
REV. 41	2004	100-1000	100-1000
REV. 42	2005	100-1000	100-1000
REV. 43	2006	100-1000	100-1000
REV. 44	2007	100-1000	100-1000
REV. 45	2008	100-1000	100-1000
REV. 46	2009	100-1000	100-1000
REV. 47	2010	100-1000	100-1000
REV. 48	2011	100-1000	100-1000
REV. 49	2012	100-1000	100-1000
REV. 50	2013	100-1000	100-1000
REV. 51	2014	100-1000	100-1000
REV. 52	2015	100-1000	100-1000
REV. 53	2016	100-1000	100-1000
REV. 54	2017	100-1000	100-1000
REV. 55	2018	100-1000	100-1000
REV. 56	2019	100-1000	100-1000
REV. 57	2020	100-1000	100-1000
REV. 58	2021	100-1000	100-1000
REV. 59	2022	100-1000	100-1000
REV. 60	2023	100-1000	100-1000
REV. 61	2024	100-1000	100-1000
REV. 62	2025	100-1000	100-1000
REV. 63	2026	100-1000	100-1000
REV. 64	2027	100-1000	100-1000
REV. 65	2028	100-1000	100-1000
REV. 66	2029	100-1000	100-1000
REV. 67	2030	100-1000	100-1000
REV. 68	2031	100-1000	100-1000
REV. 69	2032	100-1000	100-1000
REV. 70	2033	100-1000	100-1000
REV. 71	2034	100-1000	100-1000
REV. 72	2035	100-1000	100-1000
REV. 73	2036	100-1000	100-1000
REV. 74	2037	100-1000	100-1000
REV. 75	2038	100-1000	100-1000
REV. 76	2039	100-1000	100-1000
REV. 77	2040	100-1000	100-1000
REV. 78	2041	100-1000	100-1000
REV. 79	2042	100-1000	100-1000
REV. 80	2043	100-1000	100-1000
REV. 81	2044	100-1000	100-1000
REV. 82	2045	100-1000	100-1000
REV. 83	2046	100-1000	100-1000
REV. 84	2047	100-1000	100-1000
REV. 85	2048	100-1000	100-1000
REV. 86	2049	100-1000	100-1000
REV. 87	2050	100-1000	100-1000
REV. 88	2051	100-1000	100-1000
REV. 89	2052	100-1000	100-1000
REV. 90	2053	100-1000	100-1000
REV. 91	2054	100-1000	100-1000
REV. 92	2055	100-1000	100-1000
REV. 93	2056	100-1000	100-1000
REV. 94	2057	100-1000	100-1000
REV. 95	2058	100-1000	100-1000
REV. 96	2059	100-1000	100-1000
REV. 97	2060	100-1000	100-1000
REV. 98	2061	100-1000	100-1000
REV. 99	2062	100-1000	100-1000
REV. 100	2063	100-1000	100-1000